



# Ecosistemas modernos sobre Postgres en Producción

---

Nerdearla España 2025

## Sobre Emanuel Calvo / tr3s.ma

*Staff Infrastructure Engineer at Workato*



**Database/Infrastructure Engineering. Consultor independiente.**

Anteriores compañías: OnGres, Percona, Pythian, 2ndQuadrant, entre otras.



# Agenda

---

 Postgres Hoy

 Alta Disponibilidad y Escalado Vertical

 Respaldos

 Poolers y balanceadores

 Extensiones

 Monitoreo

 Escalamiento Horizontal

 Upgrades (Blue/Green, Seamless)

## Laboratorios



# Presentación



# Sobre PostgreSQL

Característica	Descripción
Open Source	<i>El GNU/Linux de las bases de datos.</i> Releases anuales estables.
Versatilidad	Desde contenedores hasta bare metal. Presente en la mayor cantidad de proveedores en la nube, kubernetes e incluso desde el <a href="#">browser</a> . Sandbox de <a href="#">psql</a> .
Funcionalidades	ACID, Framework de extensiones, Integrabilidad (CDC, FDWs).

# ¿Postgres para todo?

---

- Existen dos trends que se dieron estos últimos años:
  - Use Postgres for everything
  - Does not fit for all
- Es muy posible que para el 90% de los casos, Postgres cubra las necesidades.
- Las limitaciones pueden darse tanto a nivel de escalamiento vertical como horizontal.

# Alternativas (I)

---

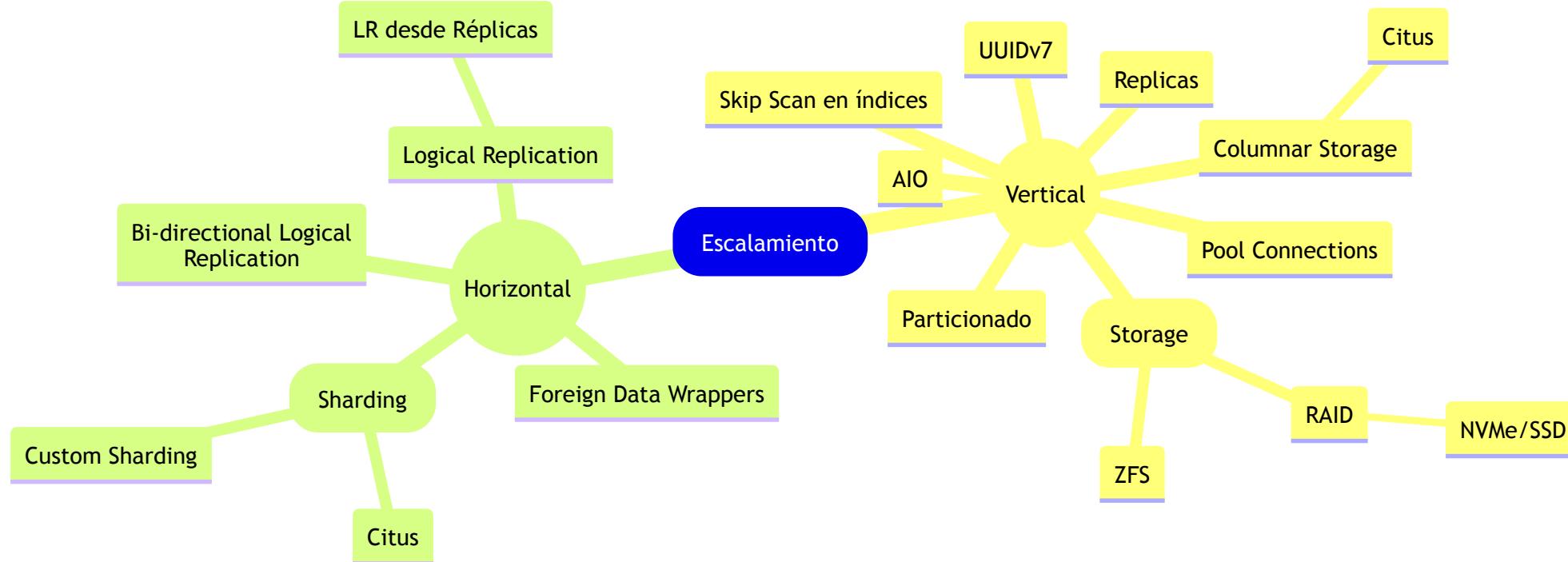
Tecnología	Alternativa
Elasticsearch	tsquery/tsvector, pgvector, <a href="#">ParadeDB</a>
MongoDB	jsonb, pgvector, <a href="#">FerretDB</a>
Redis	Unlogged tables, hstore
OLAP/Snowflake	<a href="#">pg_lake</a> , <a href="#">pg_mooncake</a> , <a href="#">pg_duckdb</a>
Queue	<a href="#">pgmq</a> , Listen/Notify

## Alternativas (II)

---

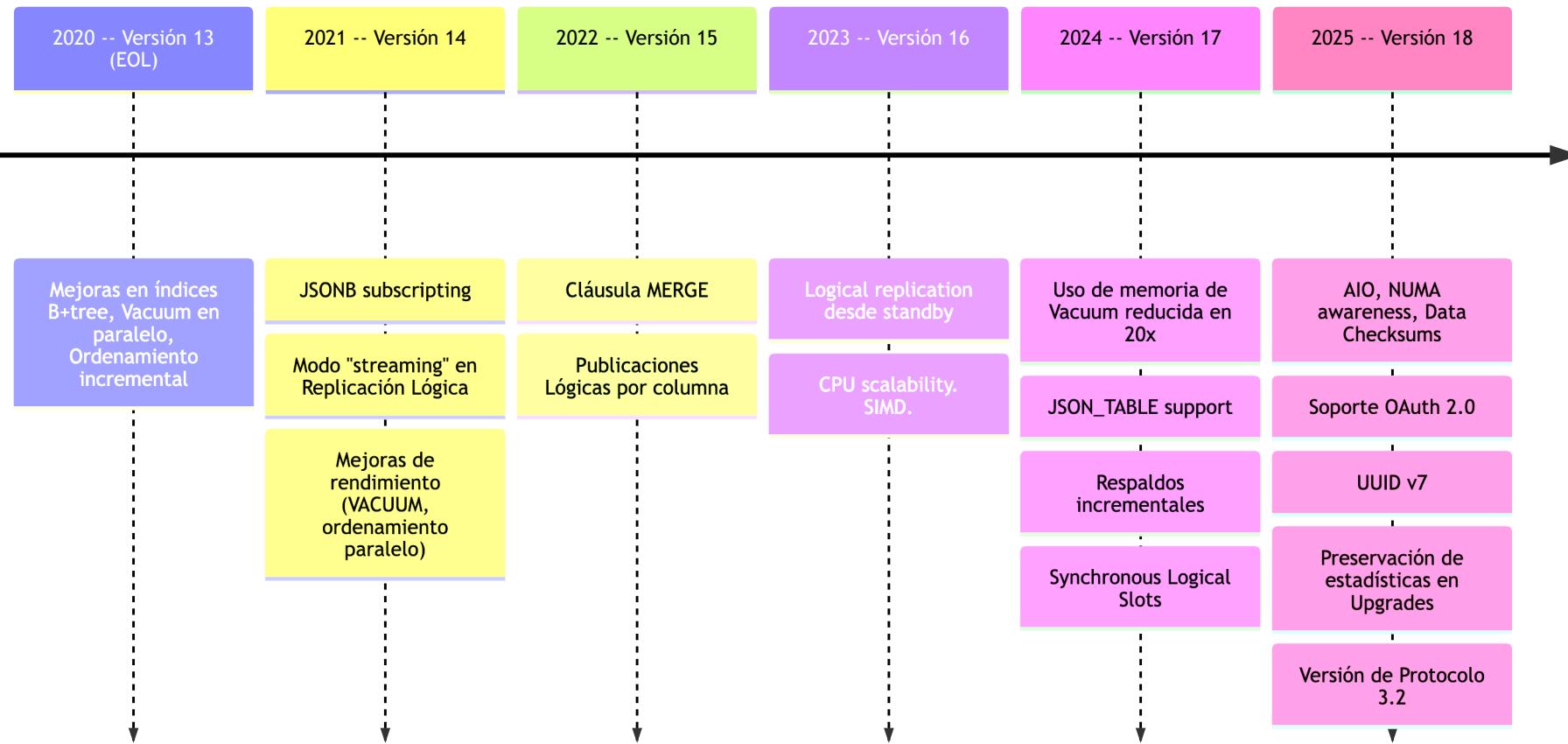
Tecnología	Alternativa
Pub/Sub	Particionado, Logical Decoding. <a href="#">Topic Partitions</a>
Desarrollo de APIs	<a href="#">PostgREST</a> , <a href="#">Prest</a>
Time Series	<a href="#">TigerData</a> ( <a href="#">TimescaleDB</a> )
Spatial	<a href="#">PostGIS</a>
Live Materialized Views	<a href="#">Epsio</a>

# Escalamiento



# Feature Timeline

## PostgreSQL Releases (2020-Present)



# Sumario: PostgreSQL 14–17

---

- v14: Modo Pipeline en libpq, tipos multirango, particionado online.
- v15: Replicación lógica por columna, security by default.
- v16: SIMD (Single Instruction, Multiple data) , parallelism, SQL/JSON constructors, replicación lógica en standbys.
- v17: Vacuum memory revolution (Radix Algorithm), complete SQL/JSON, respaldos incrementales, slots de replicación lógica síncronos (failover estables).

| PGFeatureDiff

# Postgresql v18 (1)

---

- ⚠️ | Asynchronous I/O (AIO) provee un estimado de 2/3x en mejora de rendimiento.  
[io\\_uring/liburing support commit](#)
  - Impacta en lecturas secuenciales y bitmap scans, además de una significativa mejora en el rendimiento de VACUUM. [No siempre es mejor tenerlo activado.](#)
  - Valores de `io_method` pueden ser: `worker` , `sync` , `io_uring` . Número de `workers` controlado en `io_workers` .
  - Monitoreo de Operaciones de IO: `pg_aios` .
  - Cálculos CRC32 con una mejora de rendimiento de **0.5x a 3x** en instrucciones AVX-512 (AMD e Intel) para cálculo de rutas. [Article](#)

## (2) UUID v4 vs v7

-  | **UUID v7.** Importante para: escalamiento horizontal y distribución de datos.
  - UUID v4: Todos bits aleatorios excepto por la versión (4 bits) y *variant* (2 bits).
  - Mejor distribución, ordenamiento por *timestamp*, mejores tiempos de inserción, menor cantidad de *splits* de páginas de índices.
  - Funciones "helper" como ej. `uuid_extract_timestamp(uuidv7())`
  - Ya existía una extensión para utilizar la versión, pero ahora es parte del core.

```
0199198f-e9d0-749d-9336-816392664f87
----- Timestamp 48 bits millisecond since epoch
      - Version 4 bits
      --- Random
          - Variant 2 bits
              ----- Random
```

## (3) NUMA (Non-Uniform Memory Access)

- NUMA awareness:
  - Presentaciones y artículos: [PGConf.EU](#), [Thread on NUMA observability](#), y [este artículo](#).
- Requiere opción de compilación `--with-libnuma`.

```
postgres=# select numa_zone_id, count(*) from pg_buffercache group by numa_zone_id;
NOTICE:  os_page_count=32768 os_page_size=4096 pages_per_blk=2.000000
      numa_zone_id | count
-----+-----
           6 | 16127
           1 |   256
           1 |     1
```

## (4) Otras características relevantes

- **⚠️ | Soporte *B-tree Skip Scan*** (no es necesario especificar las primeras columnas del índice en filtros). `OR/IN` se convierten en `ANY(array)`
- Soporte OAuth 2.0. `oauth_validator_libraries` setting en `pg_hba.conf`.  
`ssl_tls13_ciphers` para soporte de TLS 1.3. Negociación de TLS directa.
- **⚠️ | Preservación de estadísticas en upgrade**, migración paralela y swapping de directorios.
- **⚠️ | MD5 deprecation warning**
- **⚠️ | Data Checksums** por defecto.
- **⚠️ | Versión de Protocol** `3.2` (última actualización en 2003, 7.4).

## Alta Disponibilidad y Escalamiento Vertical

---

# Configuración

---

- [PGTune](#): Configuración de Postgres para diferentes hardware.
- [postgresqlco.nf](#): Toda la información de la configuración de Postgres.

# Replicación

---

- Streaming Replication
  - Utilizada para PITR, DR y escalamiento *vertical* (creación de réplicas).
  - Permite replicación síncrona. 
  - Es a nivel de *cluster de datos*.
- Logical Replication (escalamiento vertical u horizontal).
  - Modelo pub/sub.
  - Logical Decoding + `pgoutput` plugin.
  - Synchronous Slots (+17). ( `CREATE SUBSCRIPTION... failover=true` ).
  - Permite publicar desde una réplica. (+16).
- Logical Decoding
  - Permite utilizar un plugin distinto ( `test_decoding` , `wal2json` , ...).

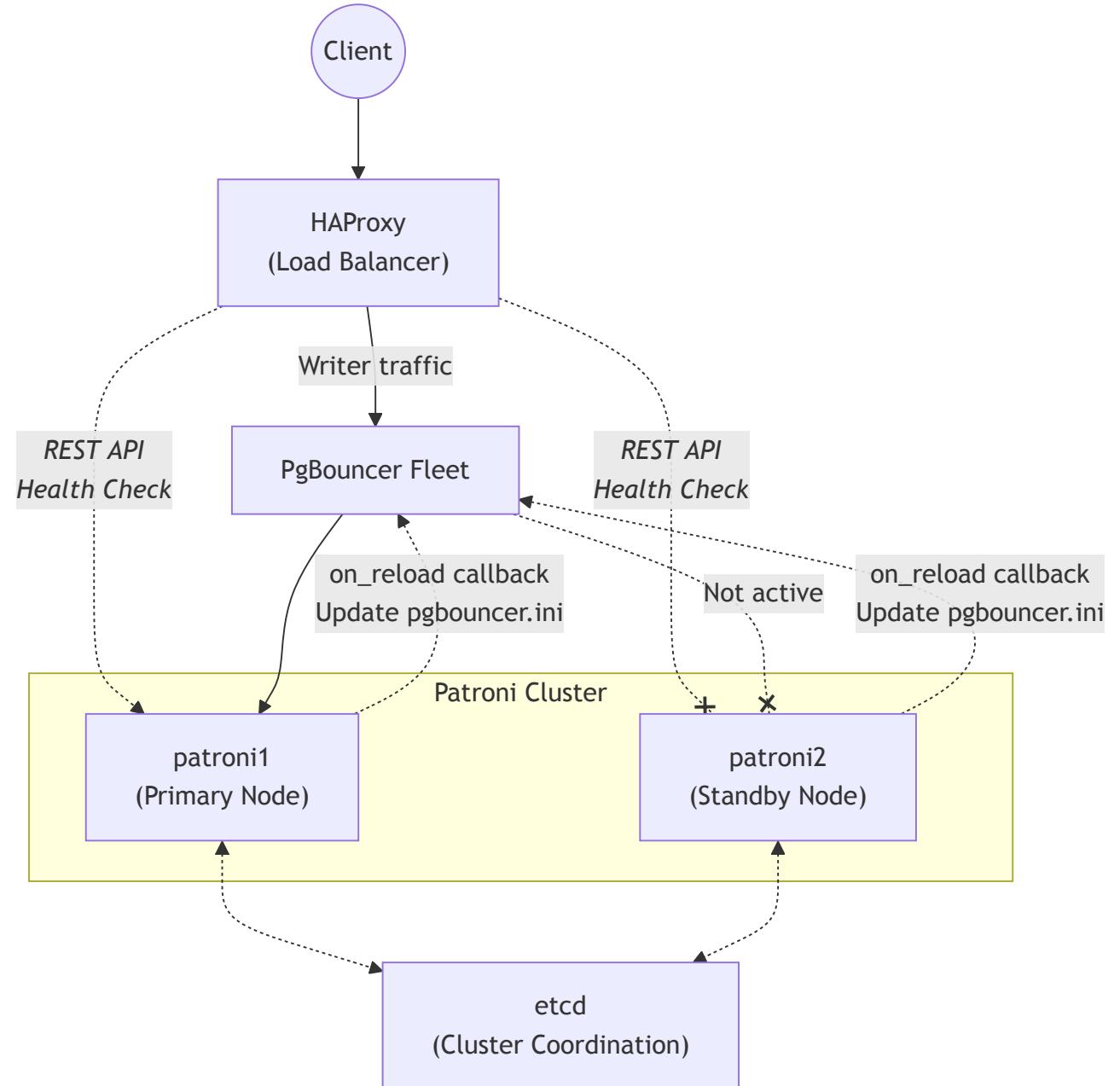
# Soluciones de (o con) Alta Disponibilidad

---

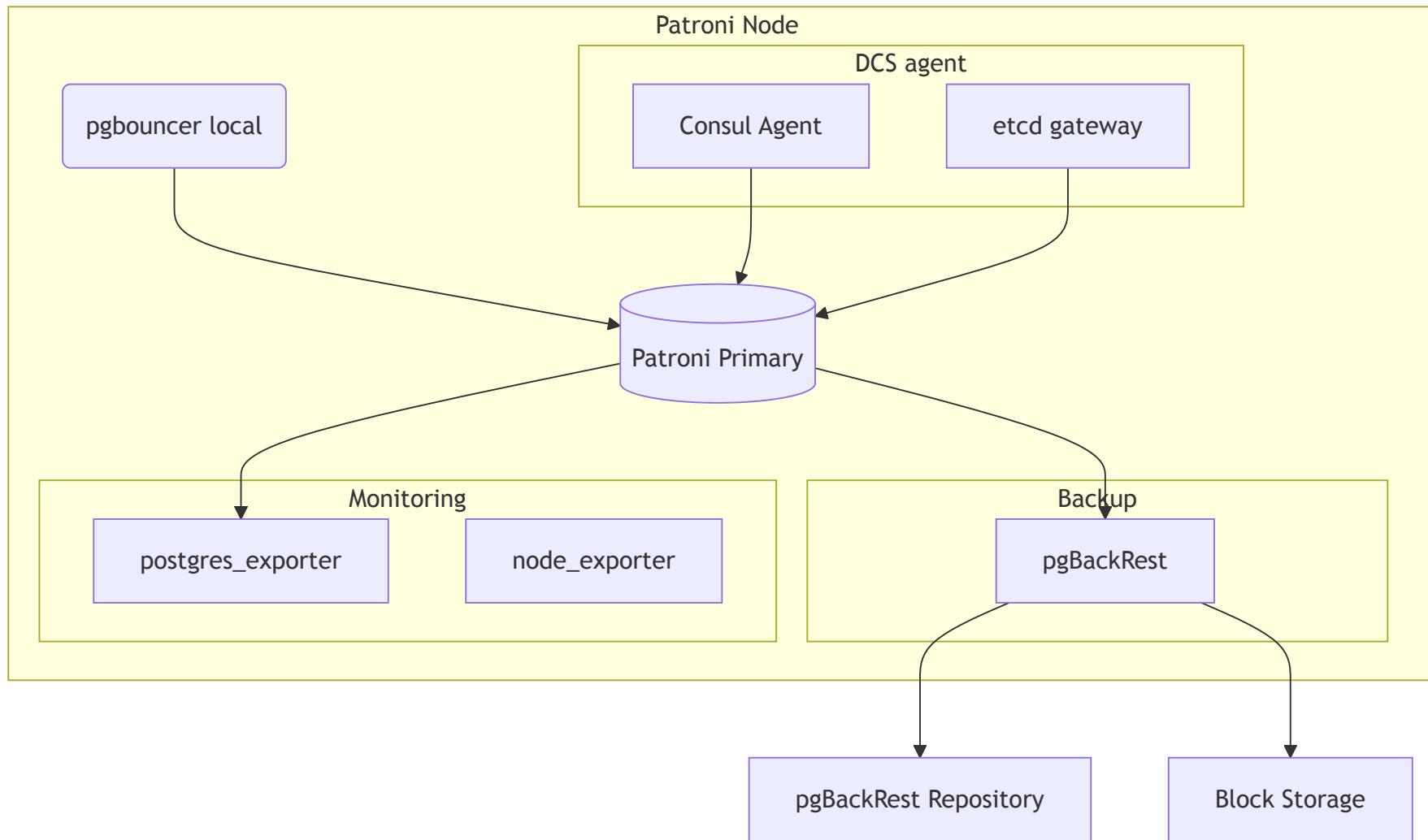
- Utilizan DCS (Distributed Consensus Service) para la elección de líder.
  - Patroni.
  - Stolon. Incluye proxies.
  - pg\_auto\_failover. Provee consenso propio.
- Yugabyte. Replicación basada en protocolo RAFT.
- Citus provee su mecanismo de HA a través de *working groups*.
- EDB Distributed.
  - Multi-master con Replicación Lógica.

## Lab:Patroni 101

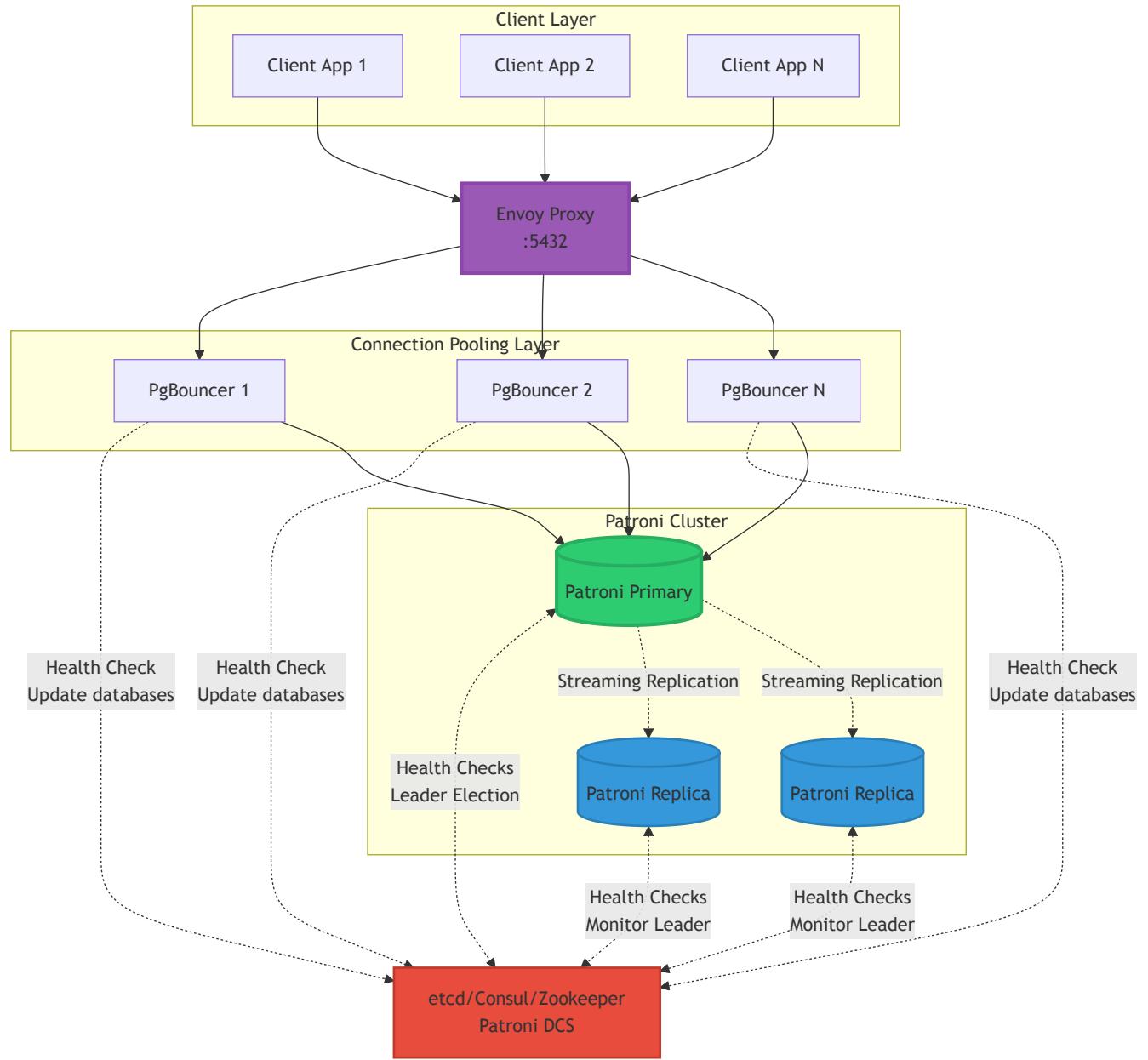
- HAproxy Entrypoint + checks
- PgBouncer Pools
- Patroni callbacks



# Dentro de un nodo de Patroni

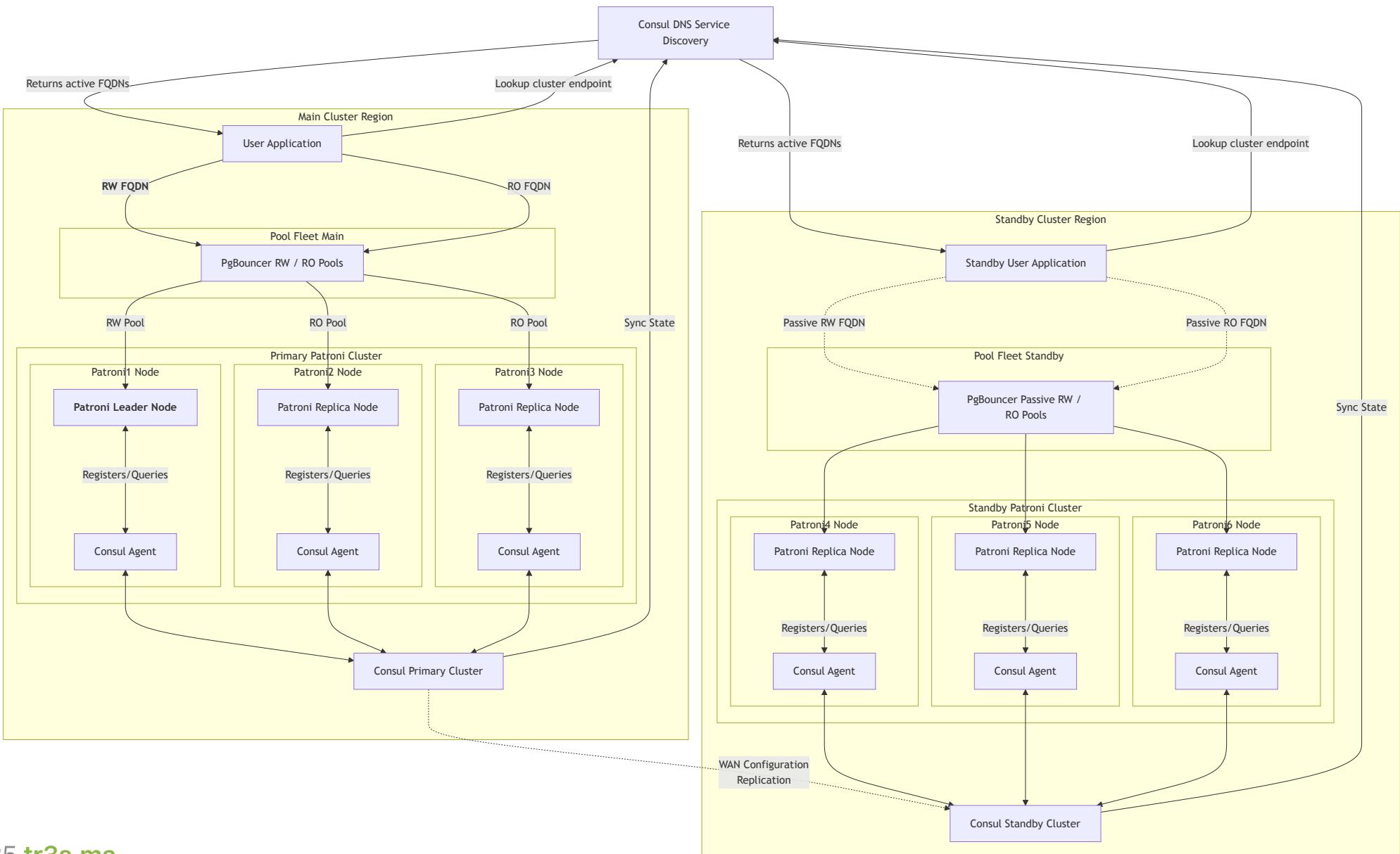


# Patroni Básico



## Multi region Patroni and Consul

---



# Columnar Storage

---

- TigerData Columnar Compression
- Citus Columnar Storage
  - cstore\_fdw
- Hydra
- pg\_mooncake. Mirror a Apache Iceberg.

# Respaldos (Físicos)

---

- pgBackRest
  - Soporta paralelismo, incrementales, almacenamiento en Block Storage y repositorios on-premise.
- Barman
- WAL-G
- Backup & Recovery
  - Respaldos *full* or incrementales desde la versión 17.

Los respaldos físicos permiten recuperación en caso de desastre y levantar réplicas sin impactar otros nodos.

# Respaldos (Lógicos)

---

- Buena práctica: Siempre tener respaldos lógicos con `pg_dump`.

Los respaldos lógicos son útiles para manipulación de datos, restauración en otros ambientes o para mover bases pequeñas.

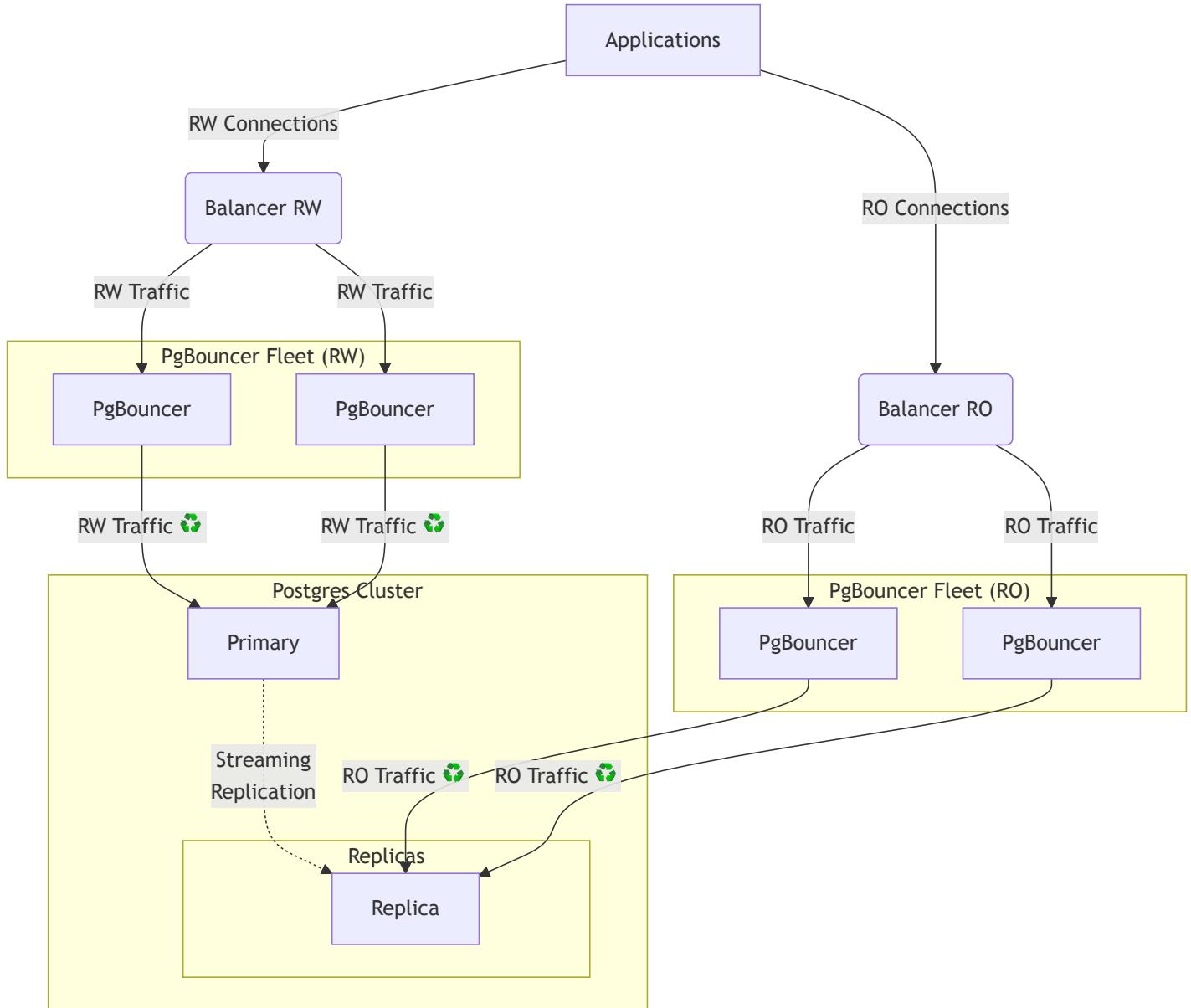
## Balanceo o entrypoint

- [pgpool-II](#): Pool, balanceo y clustering.
- [HAProxy](#)
- [Envoy](#): Soporta reporte de métricas de cada consulta. Open Telemetry.

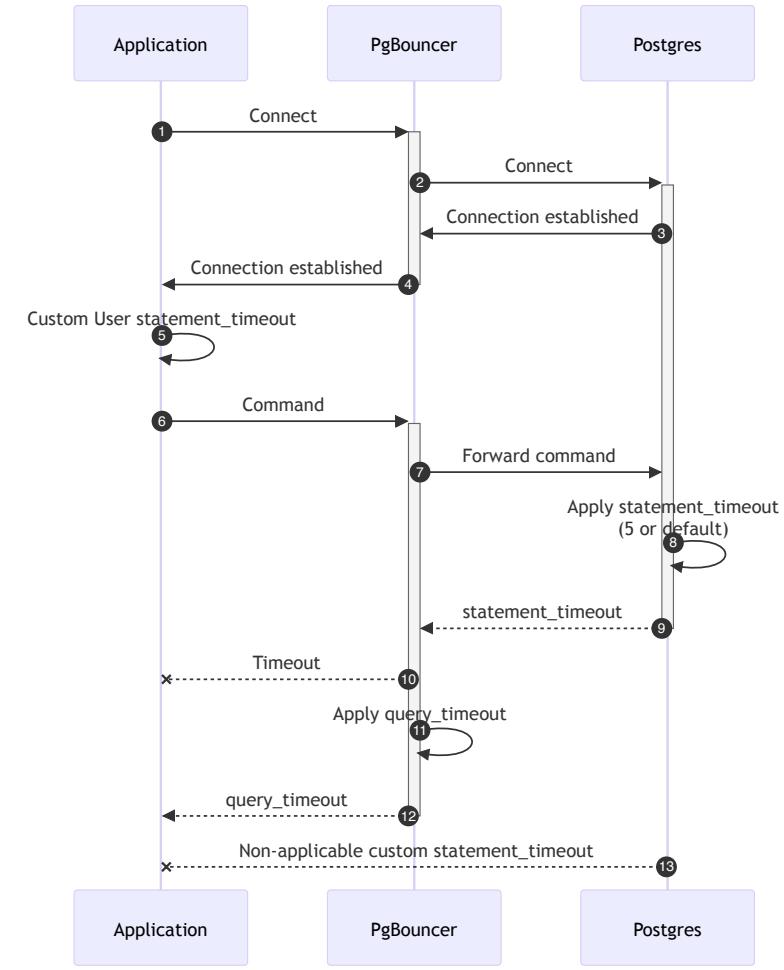
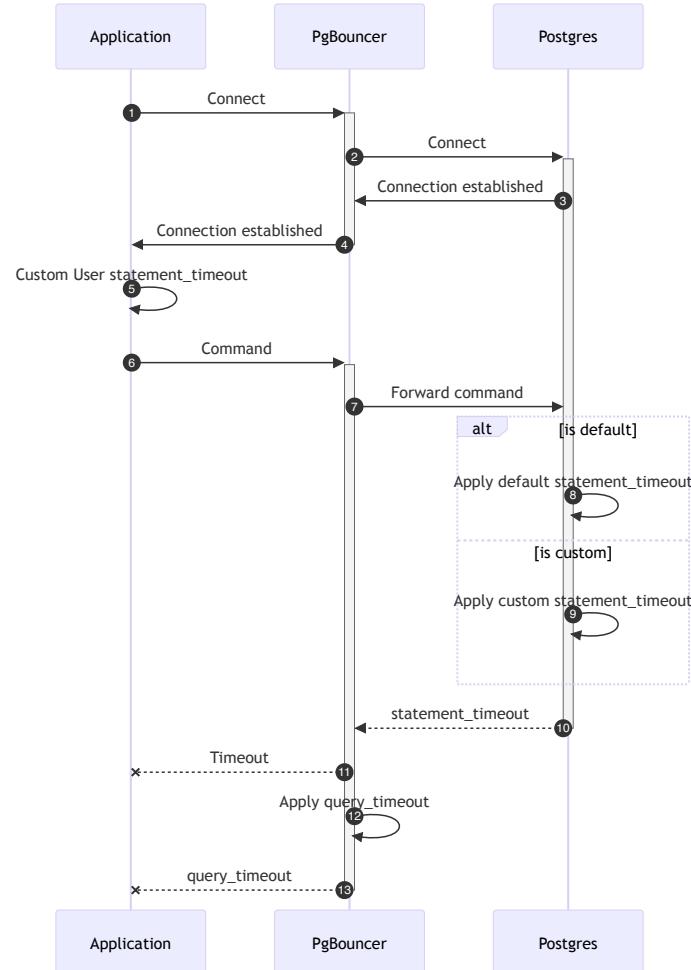
## Poolers (Control de conexiones)

- PgBouncer: Single Thread, opción por defecto.
- pgcat: Soporta Sharding por Hash.
- pgdog: Soporte de sharding por hash.
- Odyssey
- ProxySQL: Soporta *query rewrite*.
- AWS RDS Proxy.

# Ejemplo de Pooling Fleet



# Timeouts de conexiones en cascada



# Monitoreo (1)

---

- Prometheus + Grafana + postgres\_exporter.
- Open Telemetry
- pgAnalyze. Recomendación de índices .
- Percona Monitoring and Management
- Ejemplo de Logs Distribuidos en StackGres.

## Monitoreo (2)

---

### eBPF

- Cilium / Cloud Native Operator
- eBPF pgtracer



## Escalamiento Horizontal

---

# Soluciones de Escalamiento Horizontal

- Citus.
  - Columnar Storage, Sharding y Replicación. Uso de *coordinators* y *workers*.
- Yugabyte
- Multigres / Vitess-like
- Bi-directional Logical Replication
- Foreign Data Wrappers (FDW)
  - postgres\_fdw
  - FDWs

# Bidirectional Logical Replication

---

- `pglogical` permite configurar el comportamiento de la replicación lógica.
  - `pglogical.conflict_resolution` ( `error` , `apply_remote` , `keep_local` ,  
`last_update_wins` , `first_update_wins` )
  - `shared_preload_libraries = 'pglogical'` + `wal_level = 'logical'`
  - `pglogical.replicate_ddl_command`
- Desde versión 16
- BDR

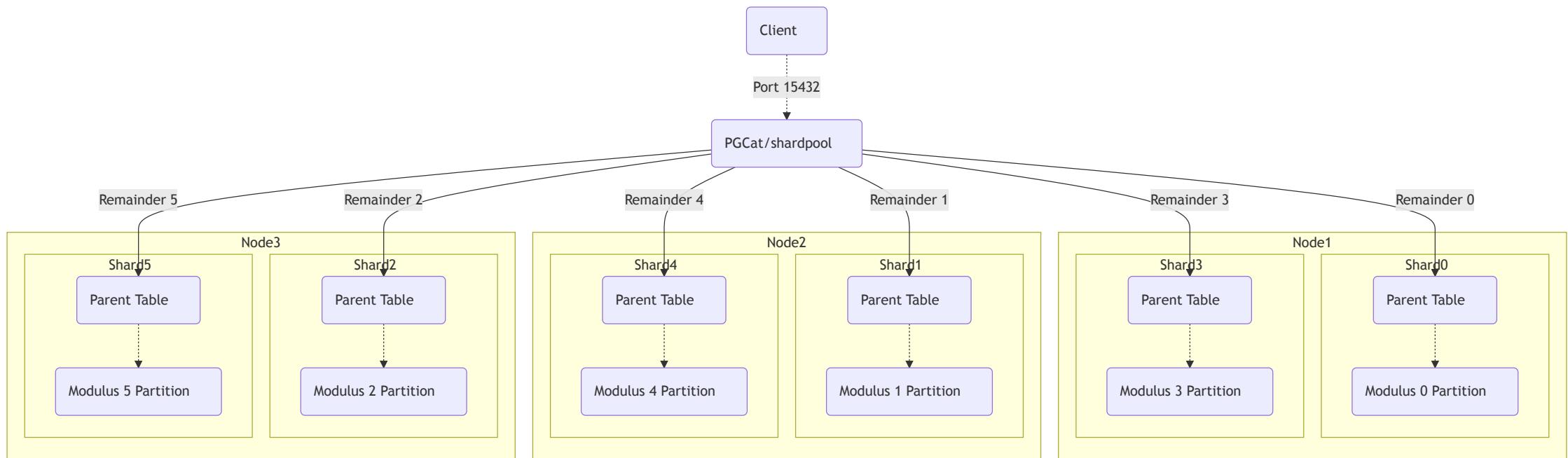
# ETL, extracción

---

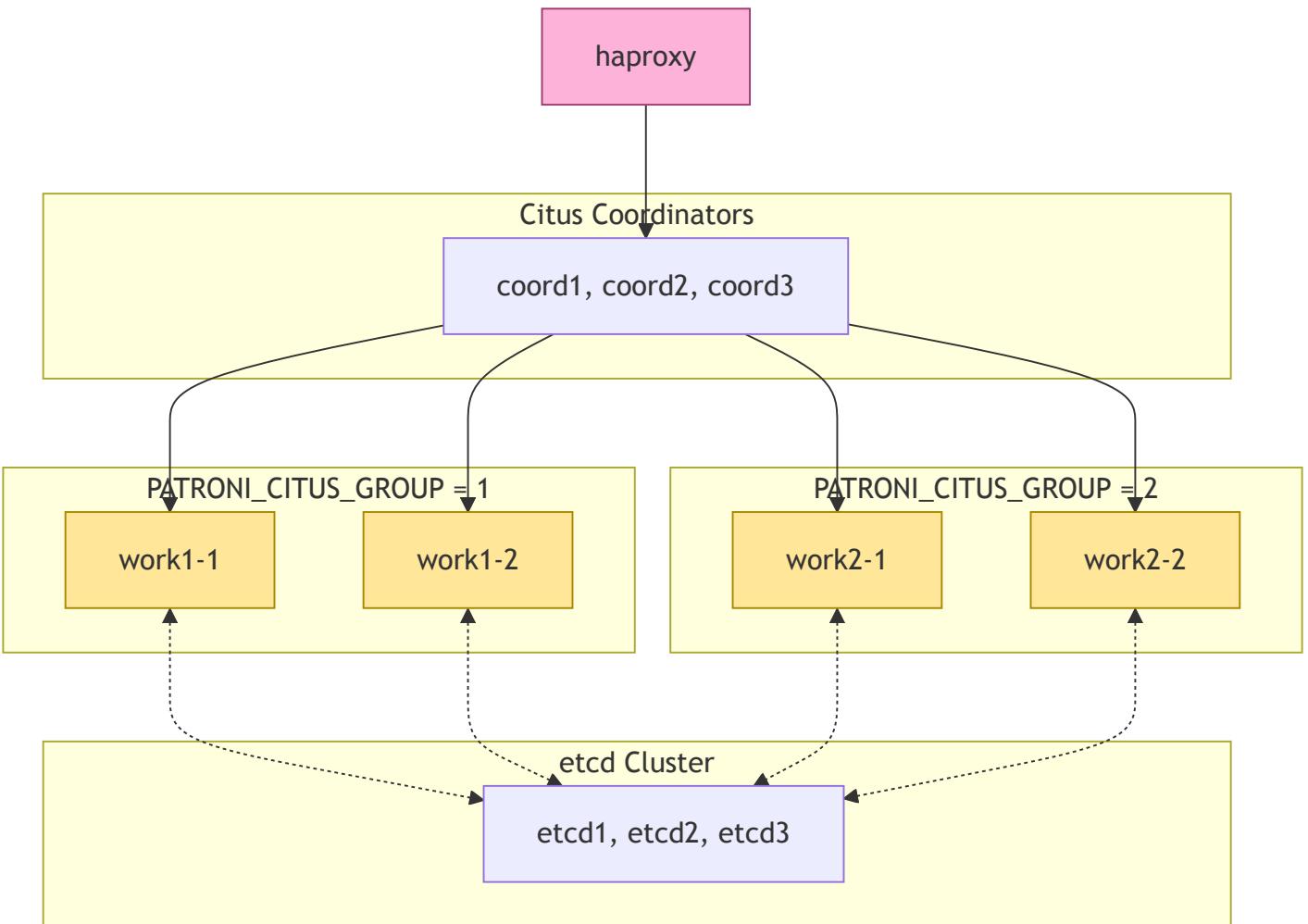
- [etl by Supabase](#)
- [dbt](#)



## Sharding por Hash con pgcat



# Citus (1)



## Citus (2)

```
CREATE EXTENSION IF NOT EXISTS citus;

-- Registro de workers
SELECT master_add_node('worker1', 5432);
SELECT master_add_node('worker2', 5432);

-- Creación de tablas distribuidas
SELECT create_distributed_table('companies',      'id');
SELECT create_distributed_table('campaigns',       'company_id');

-- Creación de índices distribuidos por PK
SELECT create_distributed_index('companies',      'id');
SELECT create_distributed_index('campaigns',       'id');
```



# Upgrades

---

## Seamless Upgrades (Fase de Upgrade)

- Con snapshot+LR (ideal para clústeres "grandes"):
  - Crear (cluster) snapshot, tomar el LSN (Logical Sequence Number) del cluster restaurado y aplicar `pg_upgrade` (o `modify-db-cluster` en RDS/Aurora).
  - Configurar LR con el LSN desde Origin -> Destination cluster.
    - ```
roname=$(psql_destination "
SELECT 'pg_'||oid::text
FROM pg_subscription
WHERE subname = '${SUBSCRIPTION_NAME}';")
```
    - En PG vanilla: `LSN=$(psql -c "SELECT pg_current_wal_lsn();")`
    - En Aurora: `LSN=$(psql -c "SELECT
aurora_volume_logical_start_lsn();")`
    - `SELECT pg_replication_origin_advance('${roname}', '<LSN>');`

## Lab:Upgrade con LR "plano"

- Con Logical Replication:
  - Crear LR con `copy_data = true`.
  - Recomendado `disable_on_error` y `streaming=on`.
  -  WARNING: Cuidado con extensiones que actualizan metadatos, como Postgis. Usar publicaciones con definición expandida.

## Fase de Switchover

- Requiere una capa de pooling que permita controlar las conexiones.
- Flow:
  - PAUSE
  - Configuración Pool (ej. sección de databases en pgbounce).
  - Drain active connections y lag=0.
  - RESUME en PgBouncer.
- Upgrades con LR

# Limitaciones y consideraciones en Upgrades

---

- <v17: `ANALYZE` antes del switchover.
- Conexiones directas al cluster de origen tienen que ser paradas y redirigidas.
- Ciertas extensiones, como Postgis, tienen tablas de metadatos que tienen que ser excluidas.
- Las versiones de las extensiones en destino tienen que ser actualizadas si se utiliza un `snapshot+LR`.
- `track_commit_timestamp` habilitado es muy útil para debuguear, pero 1) agrega cierto overhead y 2) pierde su utilidad cuando se usa `streaming=parallel`.

# Operadores k8s / Soluciones integradas

---

- [Cloud Native PostgreSQL](#)
- [Crunchy Data](#)
- [Zalando](#)
- [StackGres](#)
- [Neon](#)
- [Pigsty](#)
- [Omnigres](#)

# Extensiones

---

Links y extensiones relevantes:

- [TDE](#)
- [pg\\_oidc\\_validator](#)
- [+1000 extensiones](#)
- [PGXN](#)
- [pglogical](#)
- [openai extension / read](#)

# Menciones Especiales

---

- OrioleDB
  - Almacenamiento y Cómputo desacoplado.
- OCI Images / PGA
  - Imágenes de contenedores dinámicas (docir). PGA (Postgres Anywhere)

## Referencias/Links (1)

- [PostgreSQL 18: 10 Powerful New Features Devs Need to Know](#)
- [PostgreSQL 18 Release Notes](#)
- [Why upgrade? \(depesz.com\)](#)
- [Neon article about features](#)

## Referencias/Links (2)

- [More DBA perspective features in v18](#)
- [Postgres with dynamic containers](#)
- [Postgres as OCI images](#)
- [PGTune](#)
- [Multiregion with Patched Patroni and Consul](#)
- [Howtos](#)

# ¡Gracias!

---



- [Workato careers](#)
- [Senior Infrastructure Engineer \(ML/AI\)](#)
- [Senior Infrastructure Engineer \(Core Infra - Sec/Net\)](#)
- [Senior Infrastructure Engineer \(Core Infra\)](#)