GitLab

PostgreSQL at GitLab.com

Speaker — Jose Cores Finotto

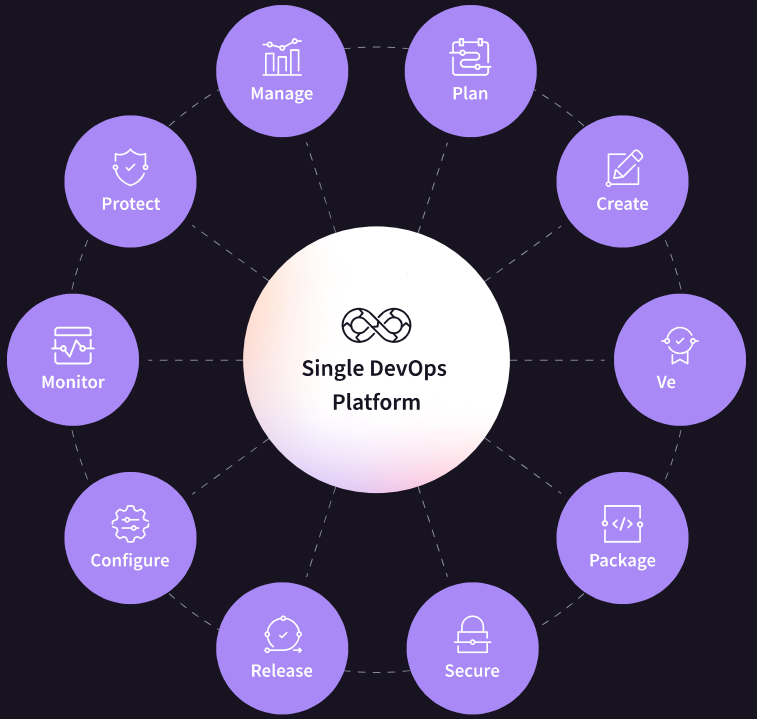
- I work with the Infrastructure team at GitLab.
- I have been a part of the GitLab team since September 2018.
- Background in large organizations with extensive experience in Infrastructure, especially in relational databases.

Speaker — Alexander Sosna

- Senior Database Reliability Engineer in the GitLab infrastructure team
- Joined GitLab ~ 1 year ago (10.2021)
- Strong background in Open Source Infrastructure with a focus on databases and PostgreSQL
- <alexander@sosna.de>

Agenda

- GitLab
- Key Specs
- Architecture
- Peak Performance Analysis
- Decomposition
- Links and Resources
- Questions and Answers

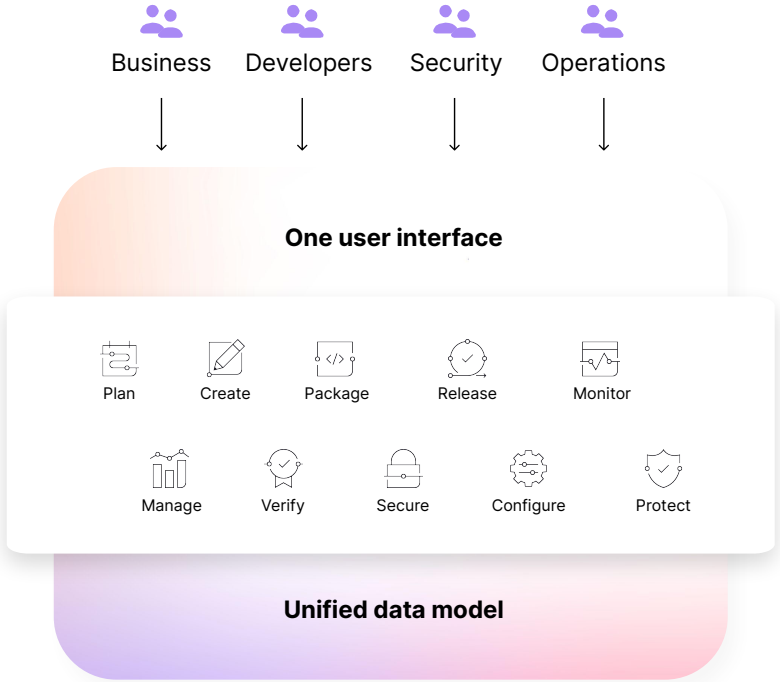


GitLab

The One DevOps Platform

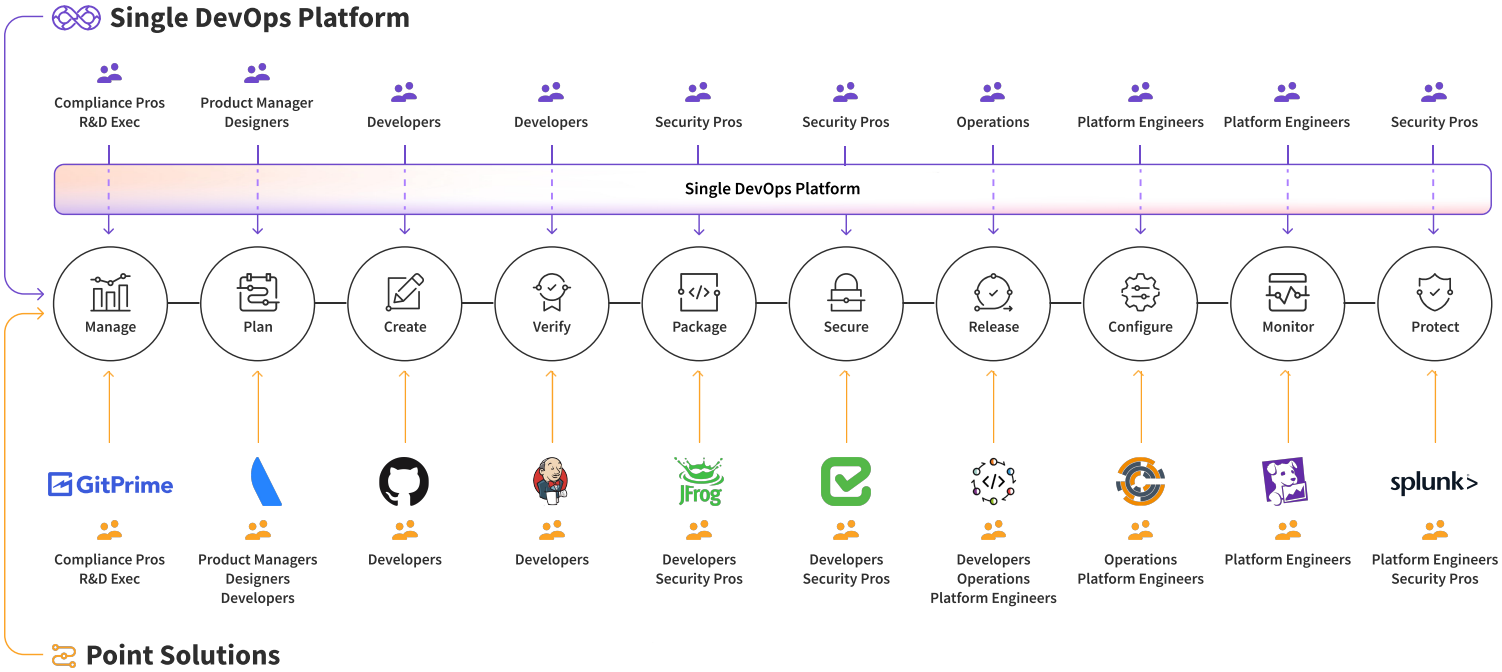
for software innovation

- Project planning
- Source code management
- Continuous integration
- Infrastructure configuration
- Incident monitoring
- Application security
- And so much more...



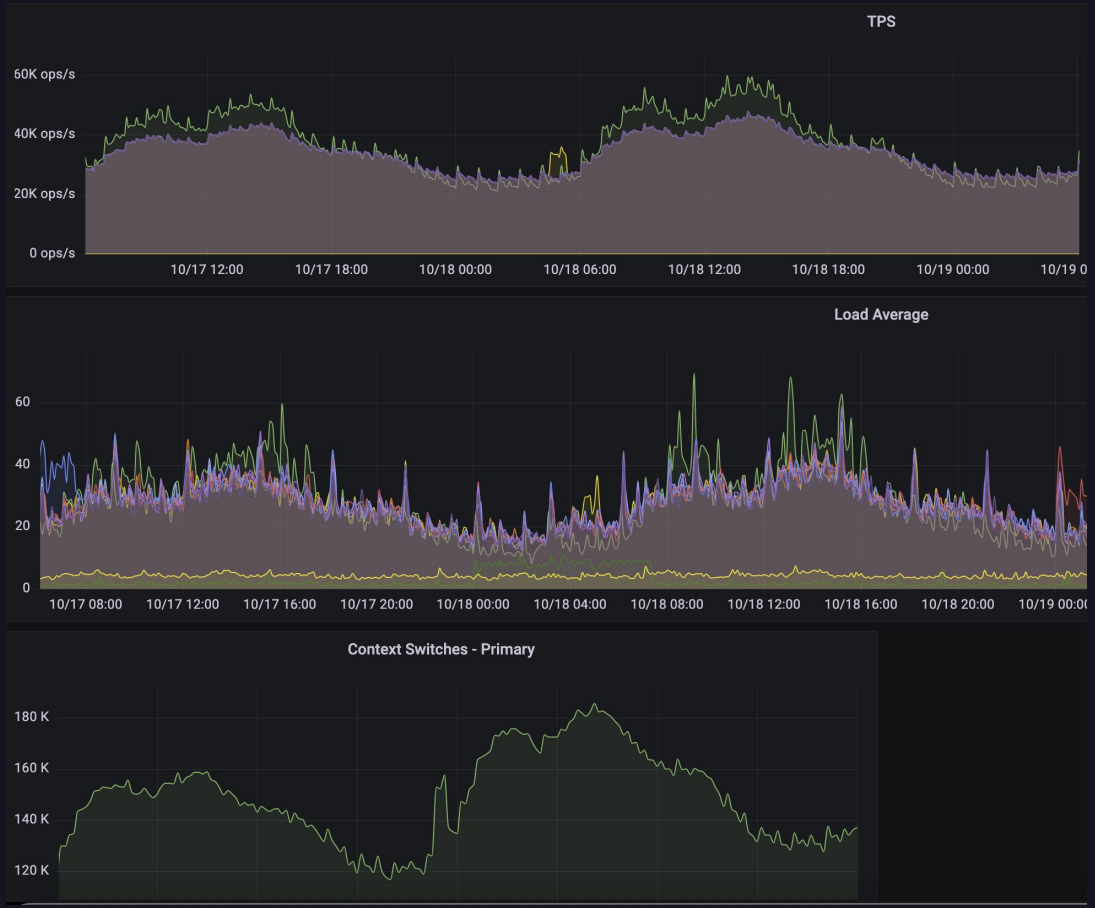
Collaborate across personas

Deliver faster, more efficiently, with reduced risk



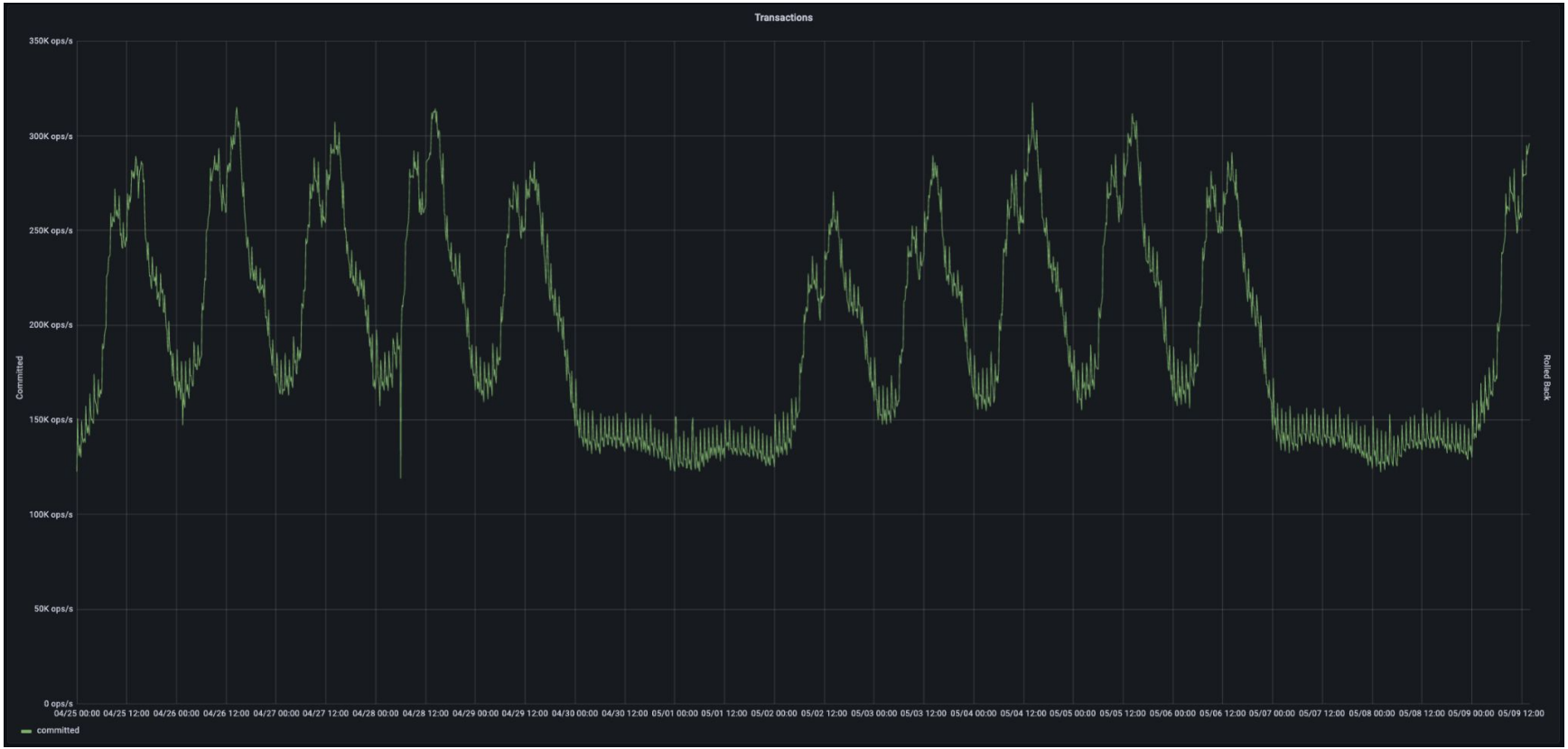
Key Specs

Key Specs


TPS
60K ops/s
40K ops/s
20K ops/s
0 ops/s
10/17 12:00
10/17 18:00
10/18 00:00
10/18 06:00
10/18 12:00
10/18 18:00
10/19 00:00
Load Average
60
40
20
0
10/17 08:00
10/17 12:00
10/17 16:00
10/17 20:00
10/18 00:00
10/18 04:00
10/18 08:00
10/18 12:00
10/18 16:00
10/18 20:00
Context Switches - Primary
180 K
160 K
140 K
120 K

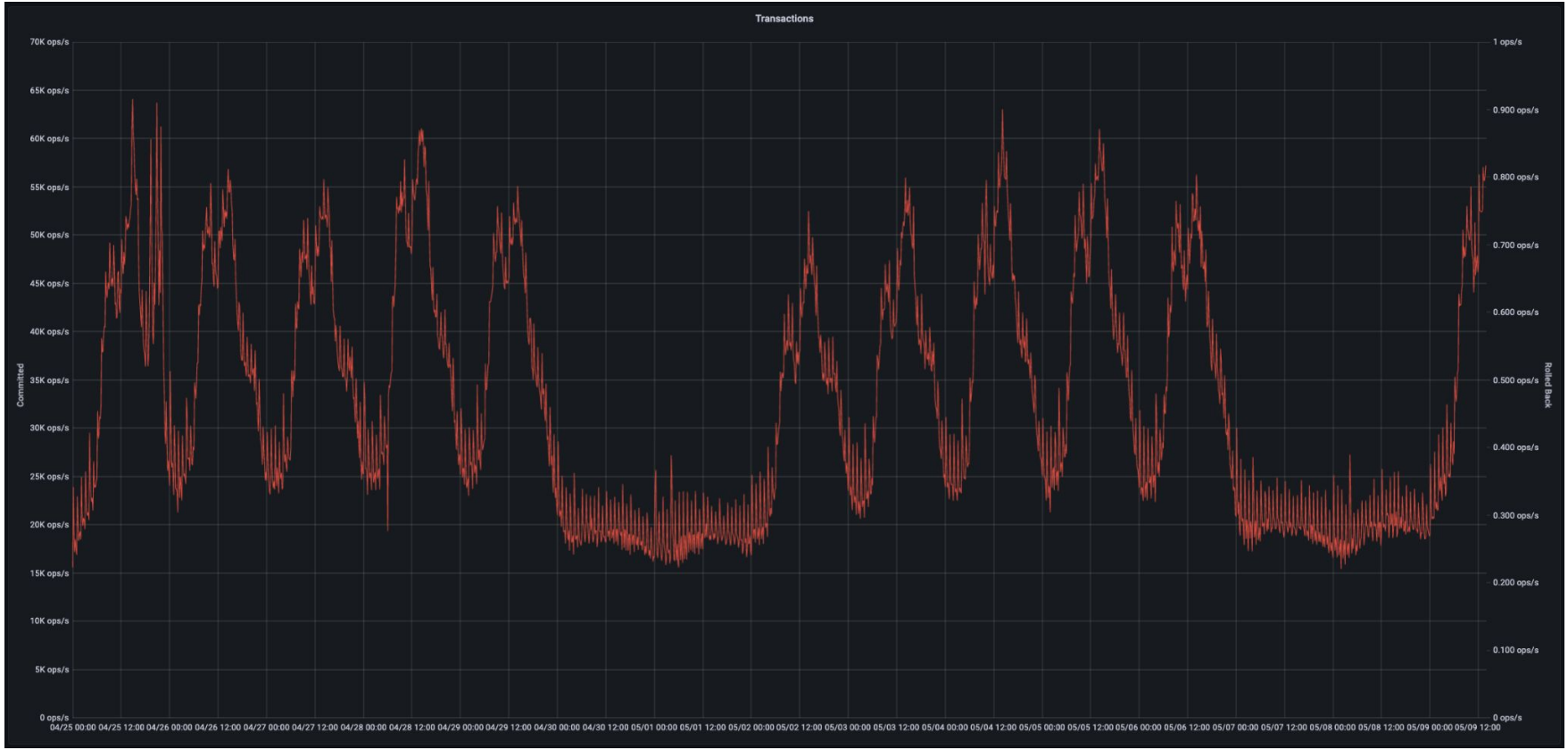
Key Specs - Read Transactions per Second

Between ~130.000 and ~300.000 TPS on the standbys



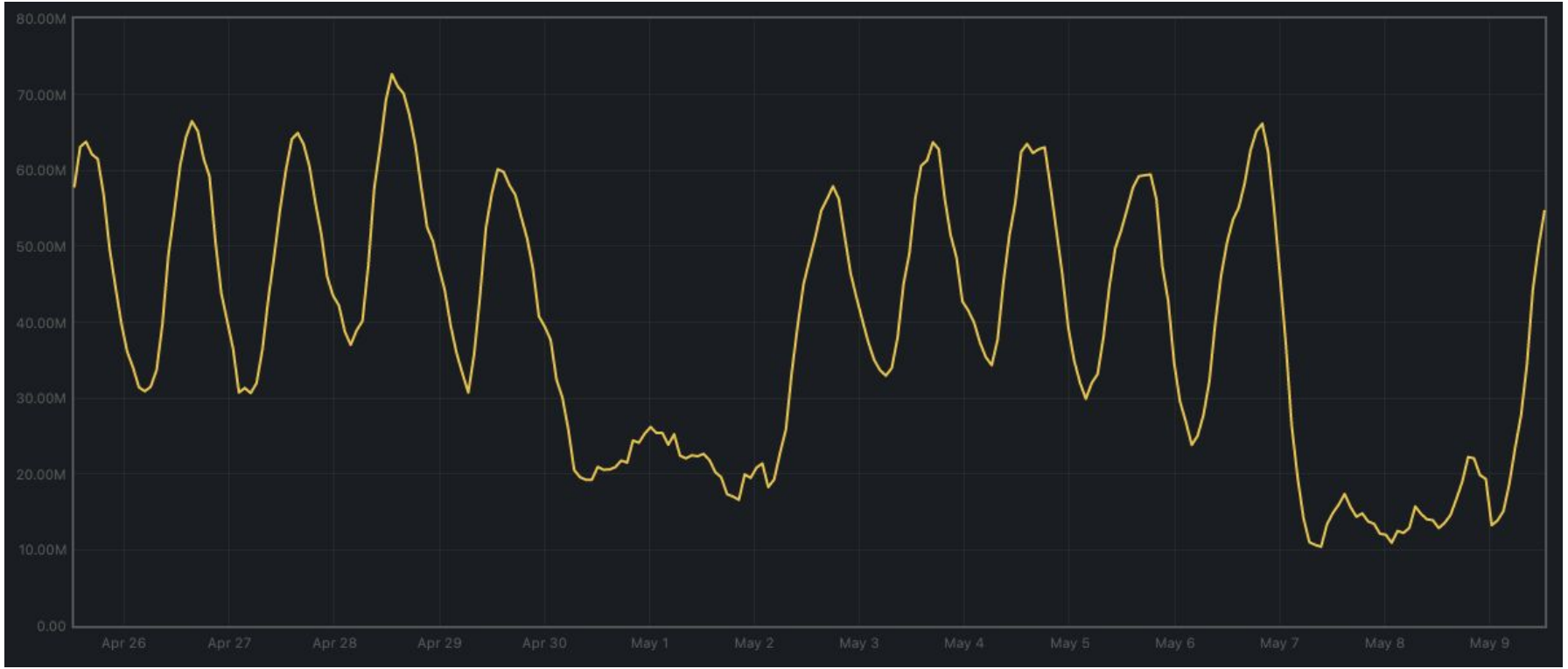
Key Specs - R/W Transactions per Second

Between ~20.000 and ~60.000 TPS on the primary



Key Specs - WAL creation

`rate(pg_xlog_position_bytes, [6h])` **between ~15MB/s and ~65MB/s WAL creation at all times**

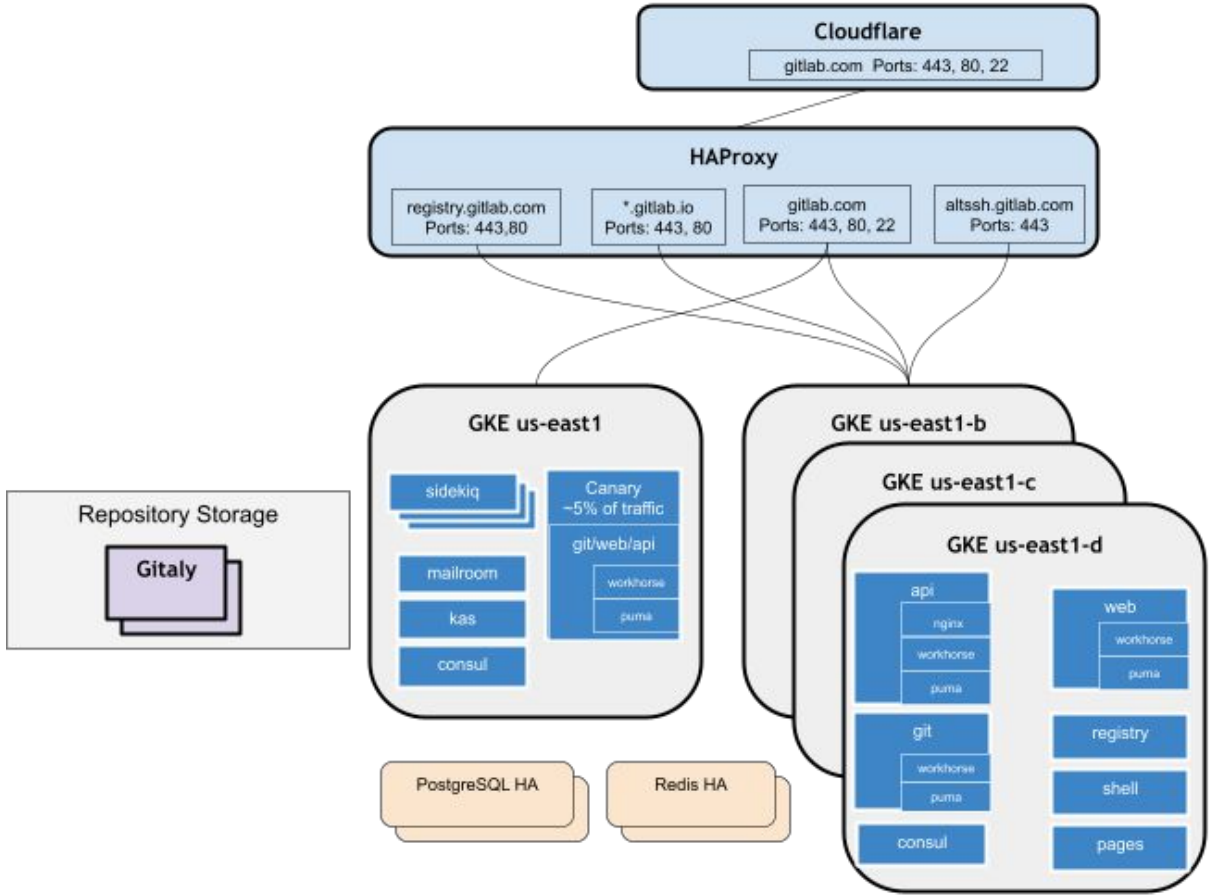


Architecture

2 db clusters with 8 nodes

96 vcpus 624 GB ram

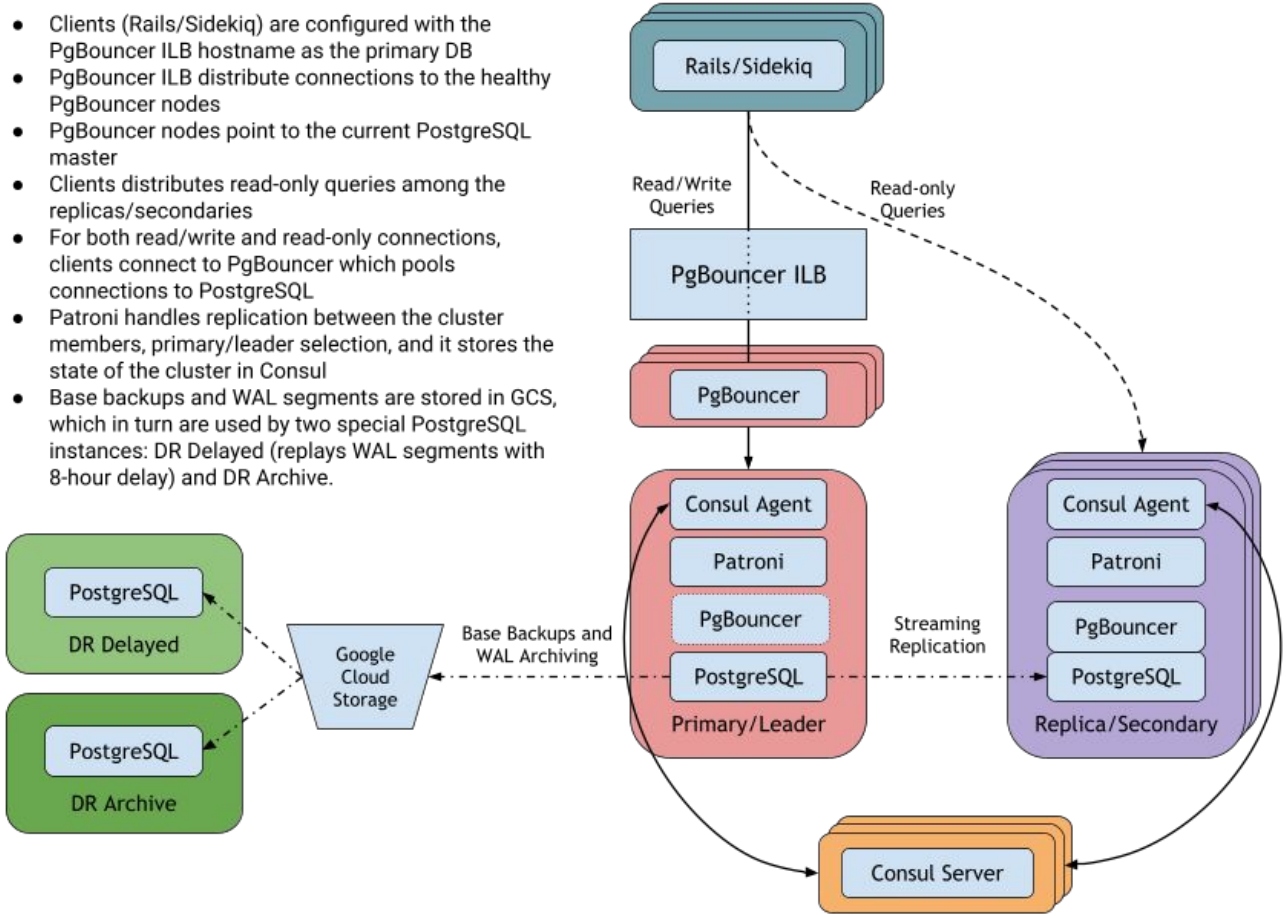
GitLab.com Architecture



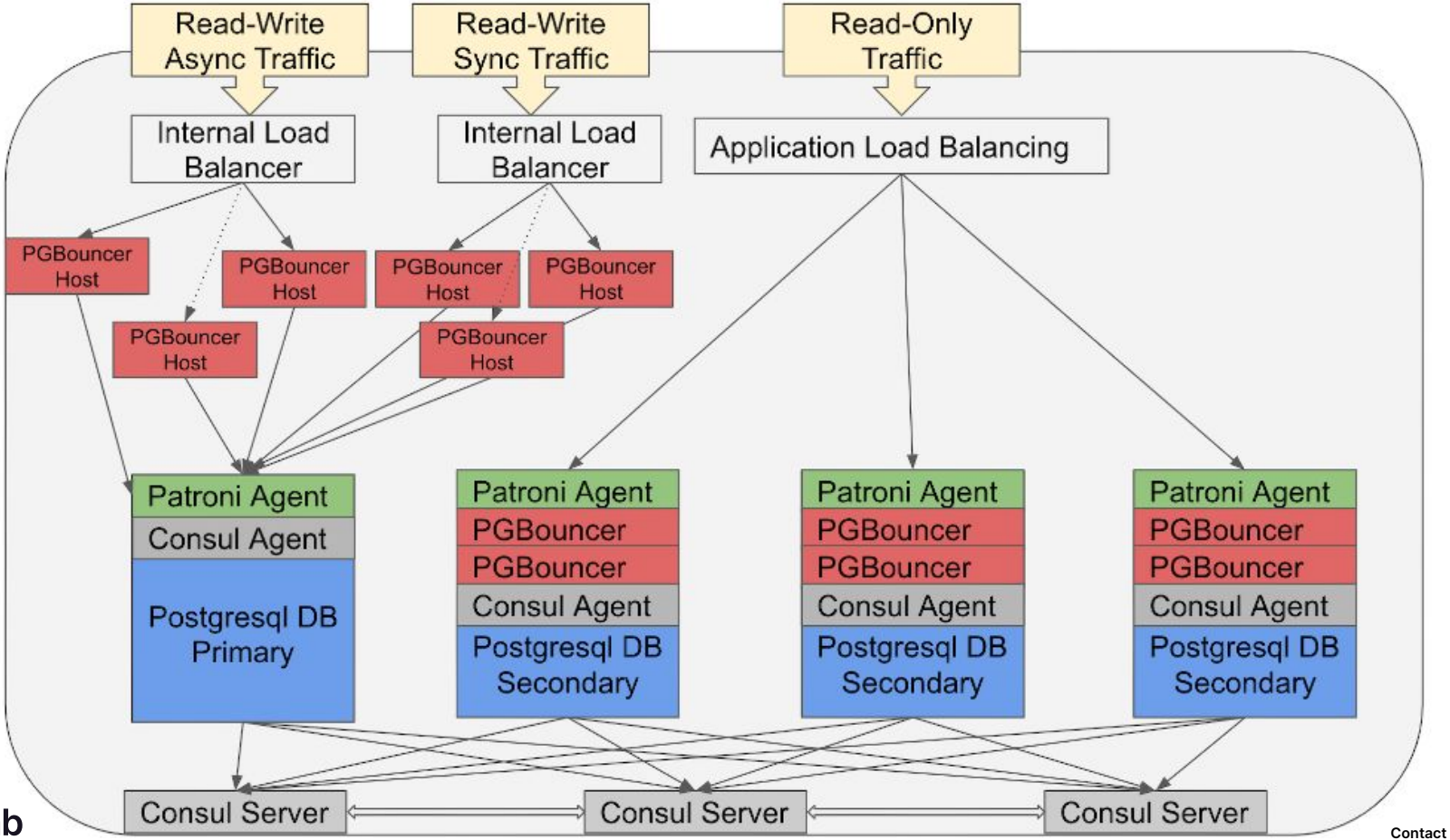
GitLab

Legacy PostgreSQL Architecture

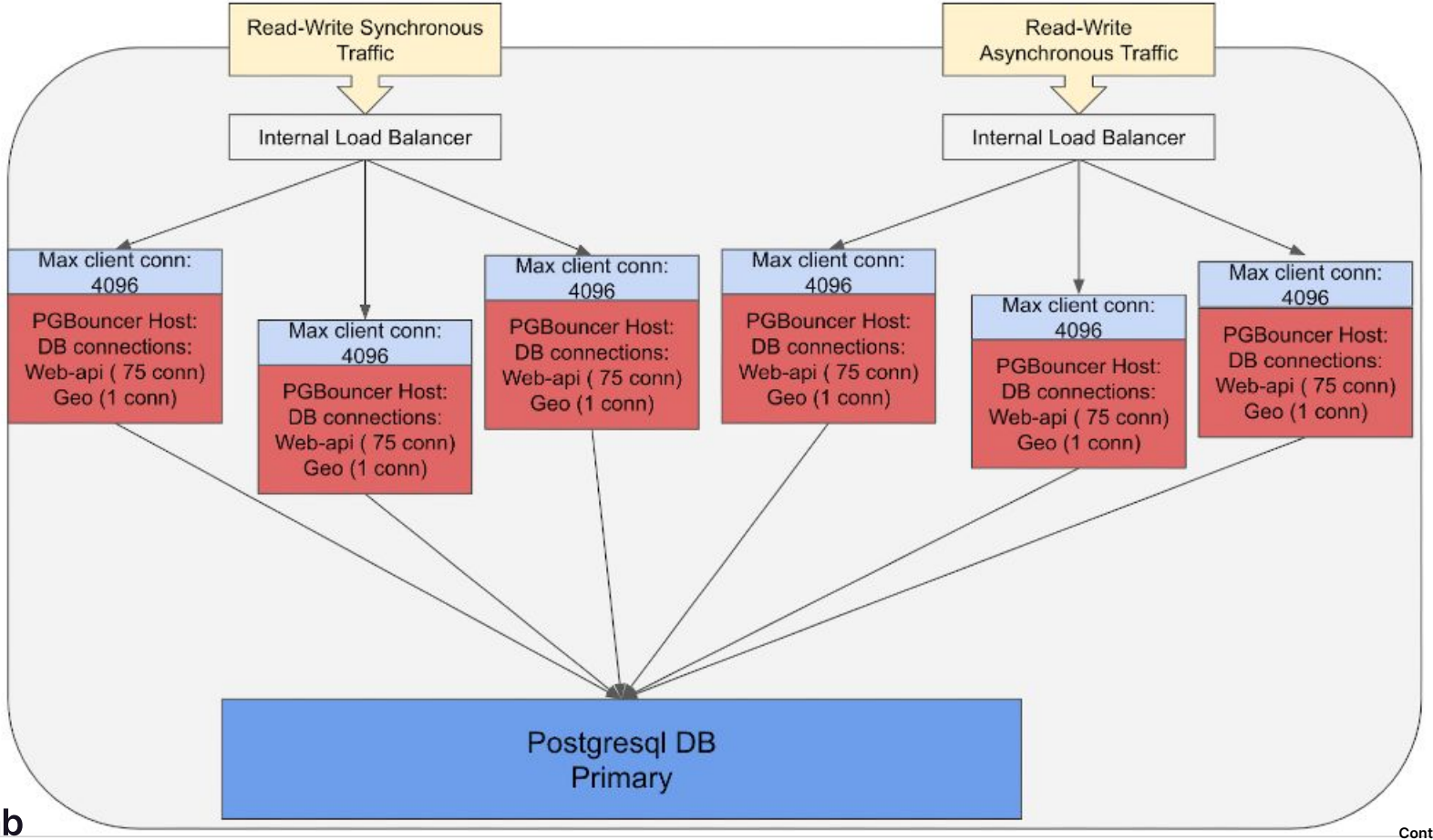
- Clients (Rails/Sidekiq) are configured with the PgBouncer ILB hostname as the primary DB
- PgBouncer ILB distribute connections to the healthy PgBouncer nodes
- PgBouncer nodes point to the current PostgreSQL master
- Clients distributes read-only queries among the replicas/secondaries
- For both read/write and read-only connections, clients connect to PgBouncer which pools connections to PostgreSQL
- Patroni handles replication between the cluster members, primary/leader selection, and it stores the state of the cluster in Consul
- Base backups and WAL segments are stored in GCS, which in turn are used by two special PostgreSQL instances: DR Delayed (replays WAL segments with 8-hour delay) and DR Archive.



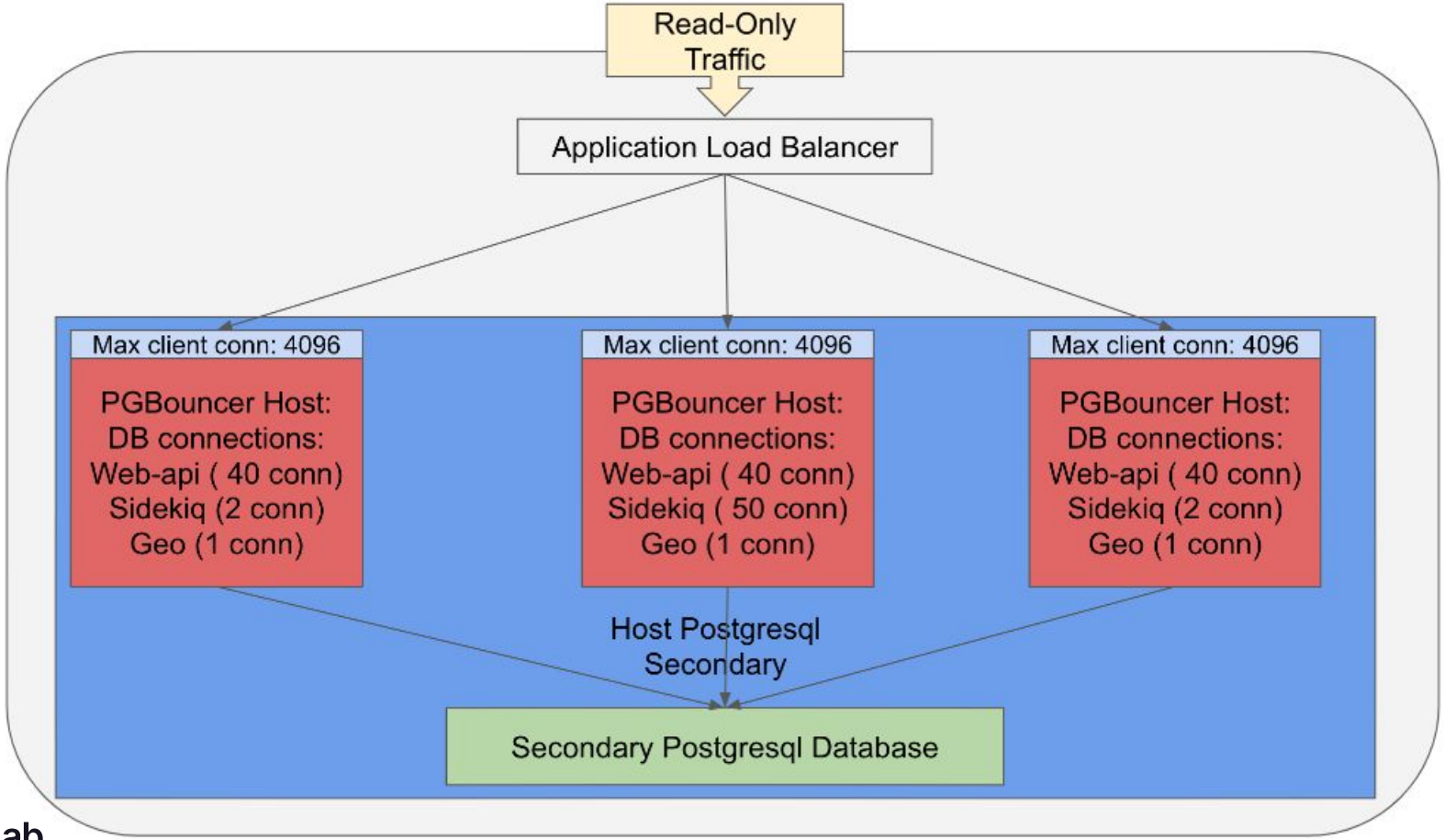
Legacy PostgreSQL Architecture - Data Diagram



Read and Write Requests

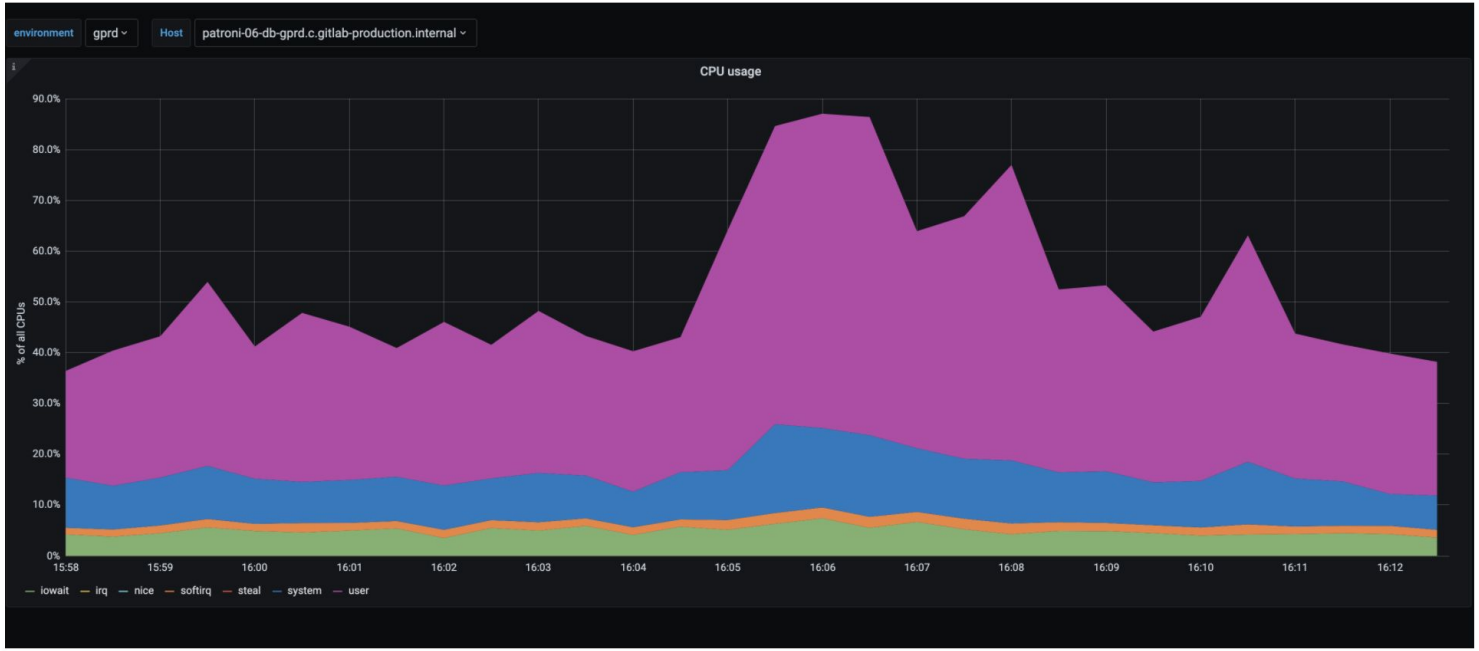


Read-Only Requests



Database performance peak

The following CPU utilization peak started at 16:05, reaching 87%:



- [x] Evaluate the analysis report, metrics and queries. If applies, create new issues with the label `infradev` or `datastores` to propose new improvements to the database cluster overall.

Peak Performance Analysis

Jose Finotto @Finotto · 1 day ago

Owner

We had the following top 10 statements **by total time** in execution during this peak:

Query:

```
topk(10,
  sum by (queryid) (
    rate(pg_stat_statements_seconds_total{env="gprd", monitor="db", type="patroni",instance="patroni-06-db-gp
  )
)
```

In this analysis, we are considering a 15 minutes interval.

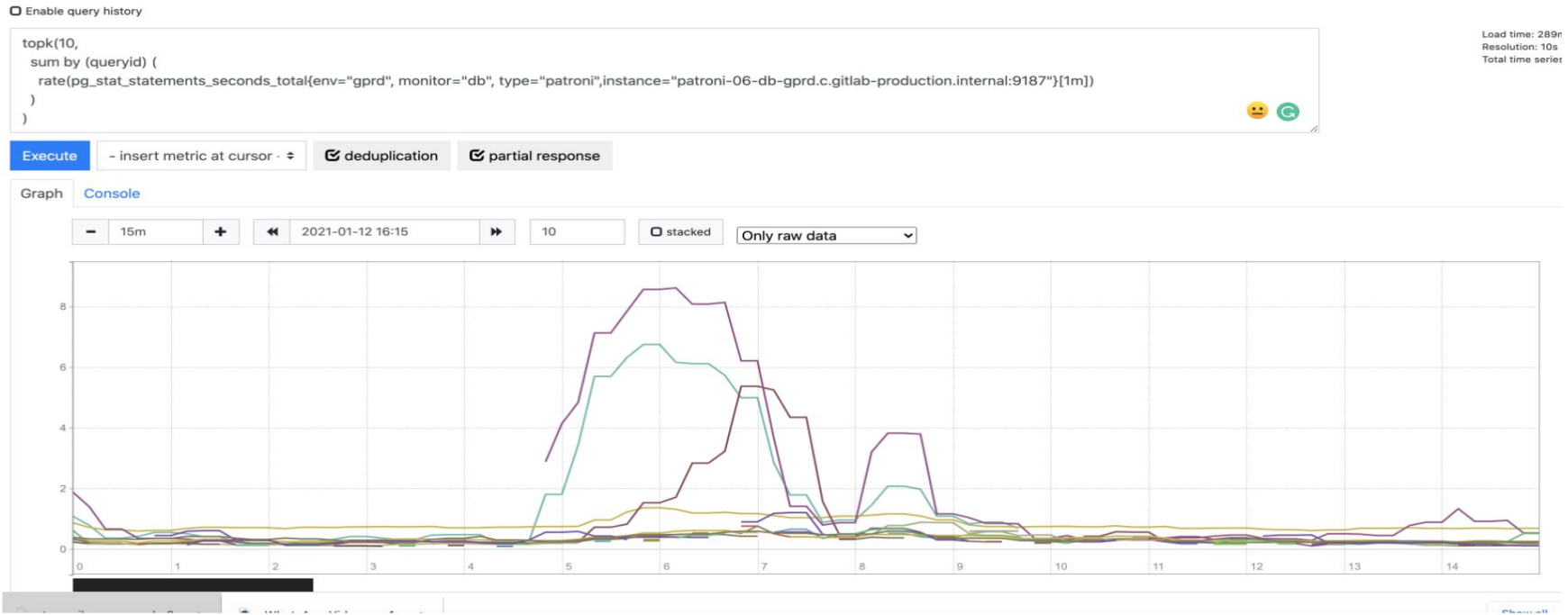
https://thanos-query.ops.gitlab.net/graph?g0.range_input=15m&g0.end_input=2021-01-12%2016%3A15&g0.step_input=10&g0.max_source_resolution=0s&g0.expr=topk(10%2C%20%0A%20%20sum%20by%20(queryid)%20(%0A%20%20%20%20rate(pg_stat_statements_seconds_total%7Benv%3D%22gprd%22%2C%20monitor%3D%22db%22%2C%20type%3D%22patroni%22%2Cinstance%3D%22patroni-06-db-gprd.c.gitlab-production.internal%3A9187%22%7D%5B1m%5D)%0A%20%20)%0A)&g0.tab=0

Peak Performance Analysis

Jose Finotto @Finotto · 1 day ago

Owner

The outputs are:



Peak Performance Analysis

Enable query history

```
topk(10,
  sum by (queryid) (
    rate(pg_stat_statements_seconds_total{env="gprd", monitor="db", type="patroni",instance="patroni-06-db-gprd.c.gitlab-production.internal:9187"}[1m])
  )
)
```

Load time: 183ms
Resolution: 10s
Total time series: 1

Execute | - insert metric at cursor - | deduplication | partial response

Graph | Console

2021-01-12 16:15:00

Element	Value
{queryid="3926004648916863976"}	0.8178287140222235
{queryid="-6386890822646776524"}	0.6909796111596127
{queryid="716430218221344694 7"}	0.5237485621202116
{queryid="6507699644791286491"}	0.2640517462795186
{queryid="9095629593792855100"}	0.2503059345329853
{queryid="-402488551284107289"}	0.23028561521334467
{queryid="1712385180720443674"}	0.20701351823647401
{queryid="2298083782068675032"}	0.157706000044224
{queryid="-5002940052336095544"}	0.12475411511170224
{queryid="7366711010424350814"}	0.12231413964376164

Remove Grap

Add Graph

Peak Performance Analysis

Jose Finotto @Finotto · 1 day ago

Owner

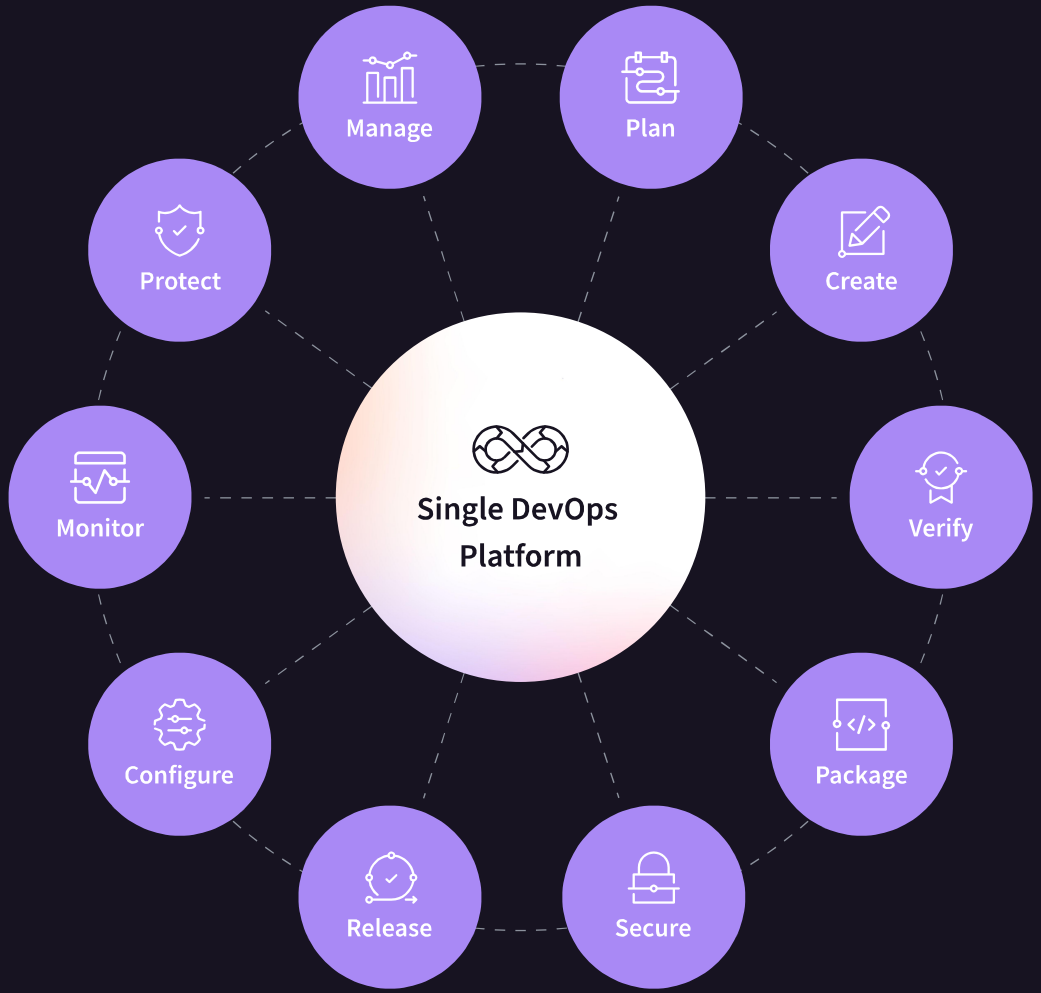
Those queryIds are the following SQL statements:

QueryId	Query
3926004648916863976	SELECT "ci_builds".* FROM "ci_builds" INNER JOIN "projects" ON "projects"."id" = "ci_builds"."project ci_builds.project_id = project_features.project_id LEFT JOIN (SELECT "ci_builds"."project_id", count() *"ci_builds"."type" = $1 AND ("ci_builds"."status" IN ($2)) AND "ci_builds"."runner_id" IN (SELECT "ci_r "ci_runners"."runner_type" = $3) GROUP BY "ci_builds"."project_id") AS project_builds ON ci_builds.p ("ci_builds"."status" IN ($4)) AND "ci_builds"."runner_id" IS NULL AND "projects"."shared_runners_en = $6 AND (project_features.builds_access_level IS NULL or project_features.builds_access_level > $7 ("projects"."visibility_level" = $9 OR (EXISTS (WITH RECURSIVE "base_and_ancestors" AS ((SELECT ' (namespaces.id = projects.namespace_id)) UNION (SELECT "namespaces".* FROM "namespaces", "b "namespaces"."id" = "base_and_ancestors"."parent_id")) SELECT $10 FROM "base_and_ancestors" AS namespace_statistics ON namespace_statistics.namespace_id = namespaces.id WHERE "namespace (COALESCE(namespaces.shared_runners_minutes_limit, $11, $12) = $13 OR COALESCE(namespace_ COALESCE((namespaces.shared_runners_minutes_limit + COALESCE(namespaces.extra_shared_run COALESCE(namespaces.extra_shared_runners_minutes_limit, $17)), $18) * $19)))) AND (NOT EXISTS "taggings"."taggable_type" = $21 AND "taggings"."context" = $22 AND (taggable_id = ci_builds.id) AN ORDER BY COALESCE(project_builds.running_builds, $25) ASC, ci_builds.id ASC */application:web,correlation_id:01EVX3GF3VGAVE6TYFMR82EJFN/*
-6386890822646776524	SELECT "users".* FROM "users" INNER JOIN "project_authorizations" ON "users"."id" = "project_auth "project_authorizations"."project_id" = $1 */application:web,correlation_id:Lmz5Aaf8Vpa/*
7164302182213446947	UPDATE "ci_builds" SET "runner_id" = 380987, "status" = 'running', "started_at" = '2020-10-29 21:0("updated_at" = '2020-10-29 21:00:54.568589', "lock_version" = 2 WHERE "ci_builds"."id" = 8201577 */application:web,correlation_id:4ze9HF2IXC9/*
6507699644791286491	SELECT SUM((("project_statistics"."repository_size" + "project_statistics"."lfs_objects_size") - "projec INNER JOIN routes rs ON rs.source_id = projects.id AND rs.source_type = 'Project' INNER JOIN "proj "project_statistics"."project_id" = "projects"."id" WHERE (rs.path LIKE 'gitlab-org/%') AND ("project_s "project_statistics"."lfs_objects_size") > "projects"."repository_size_limit" AND "projects"."repository_s */application:web,controller:merge_requests,action:index,correlation_id:HlfxW7Ir8b1/*

Decomposition

Improve Performance, Enable Future Growth

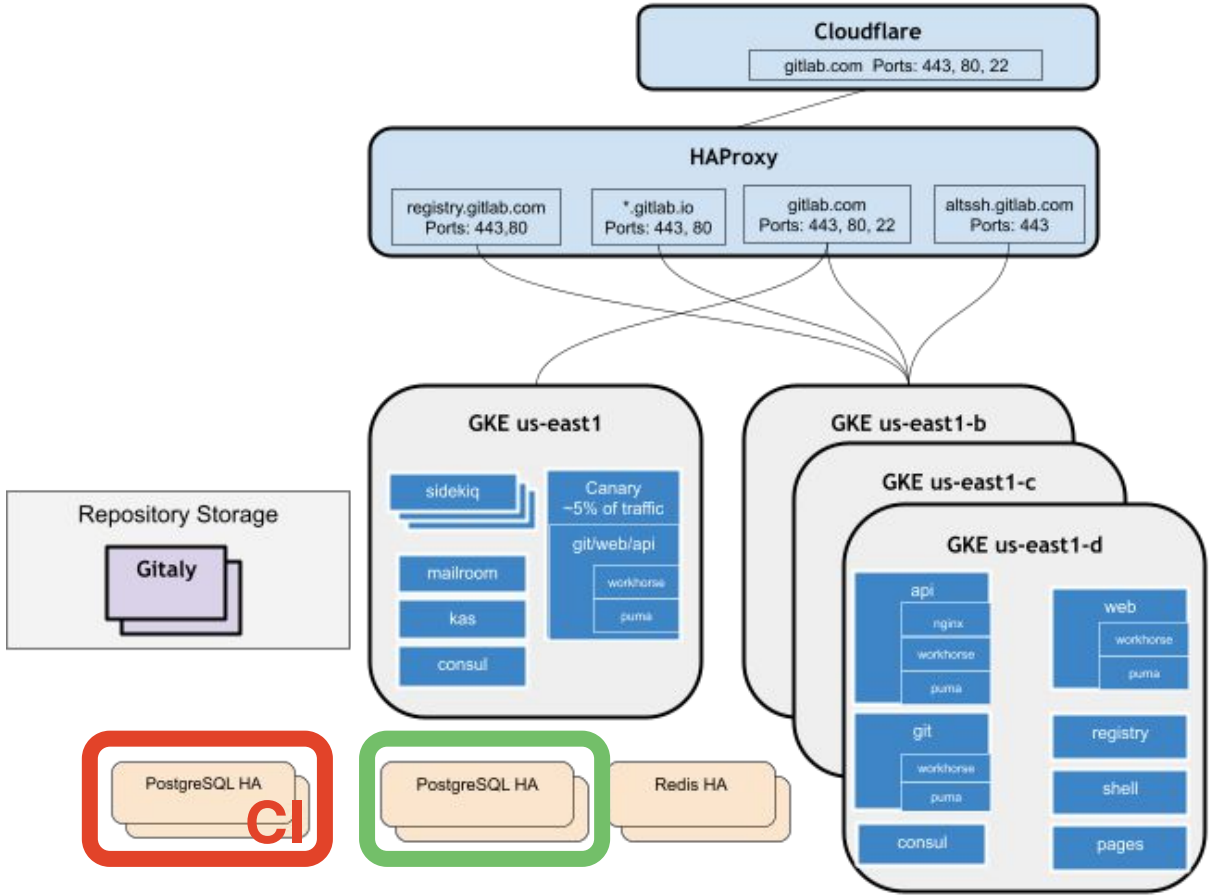
- Use bigger machines, currently: *n1-standard-96*
- More hot standbys for read only scaling
- Separate different workloads
- ...



GitLab

Contact GitLab

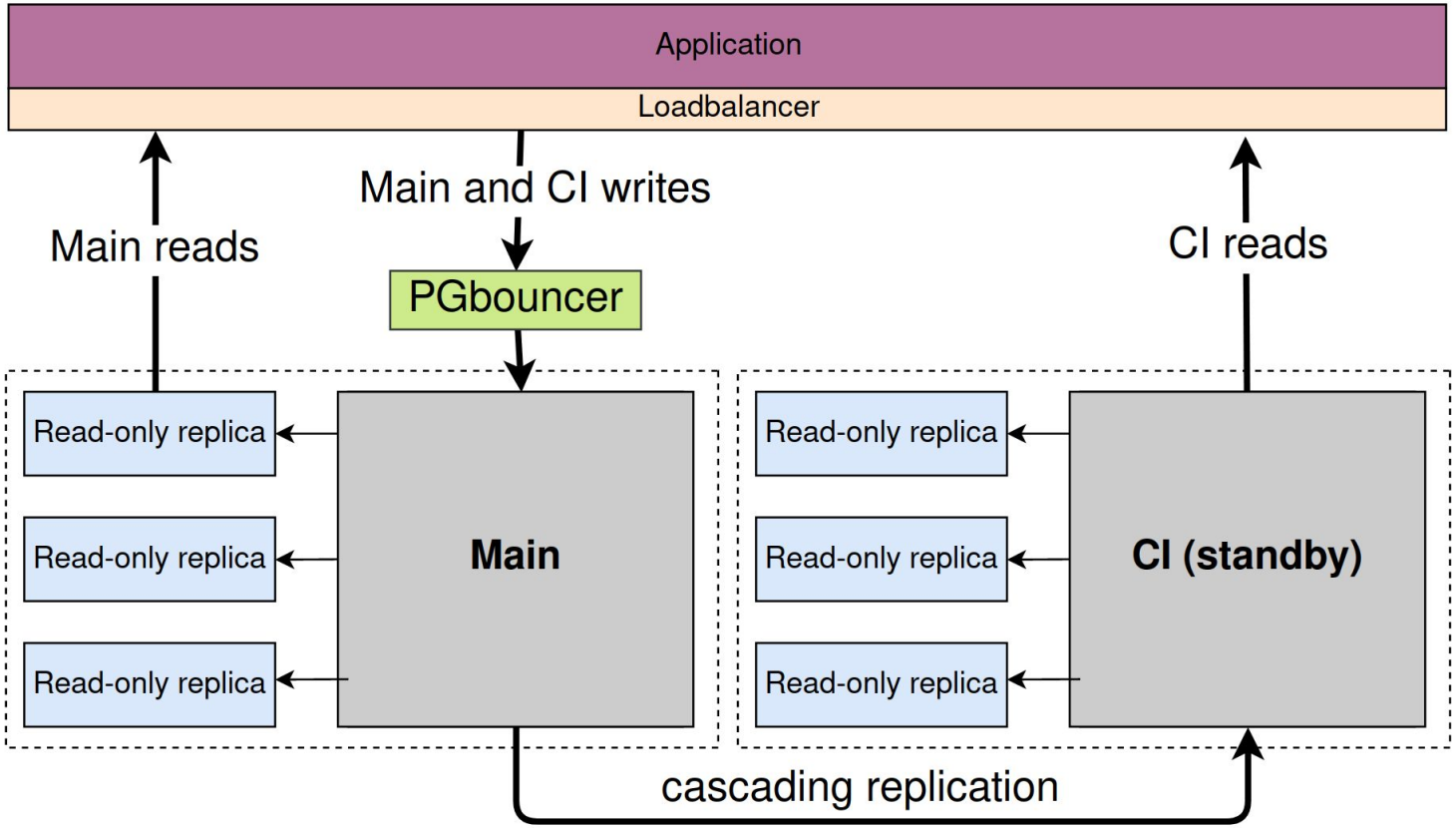
New GitLab.com Architecture



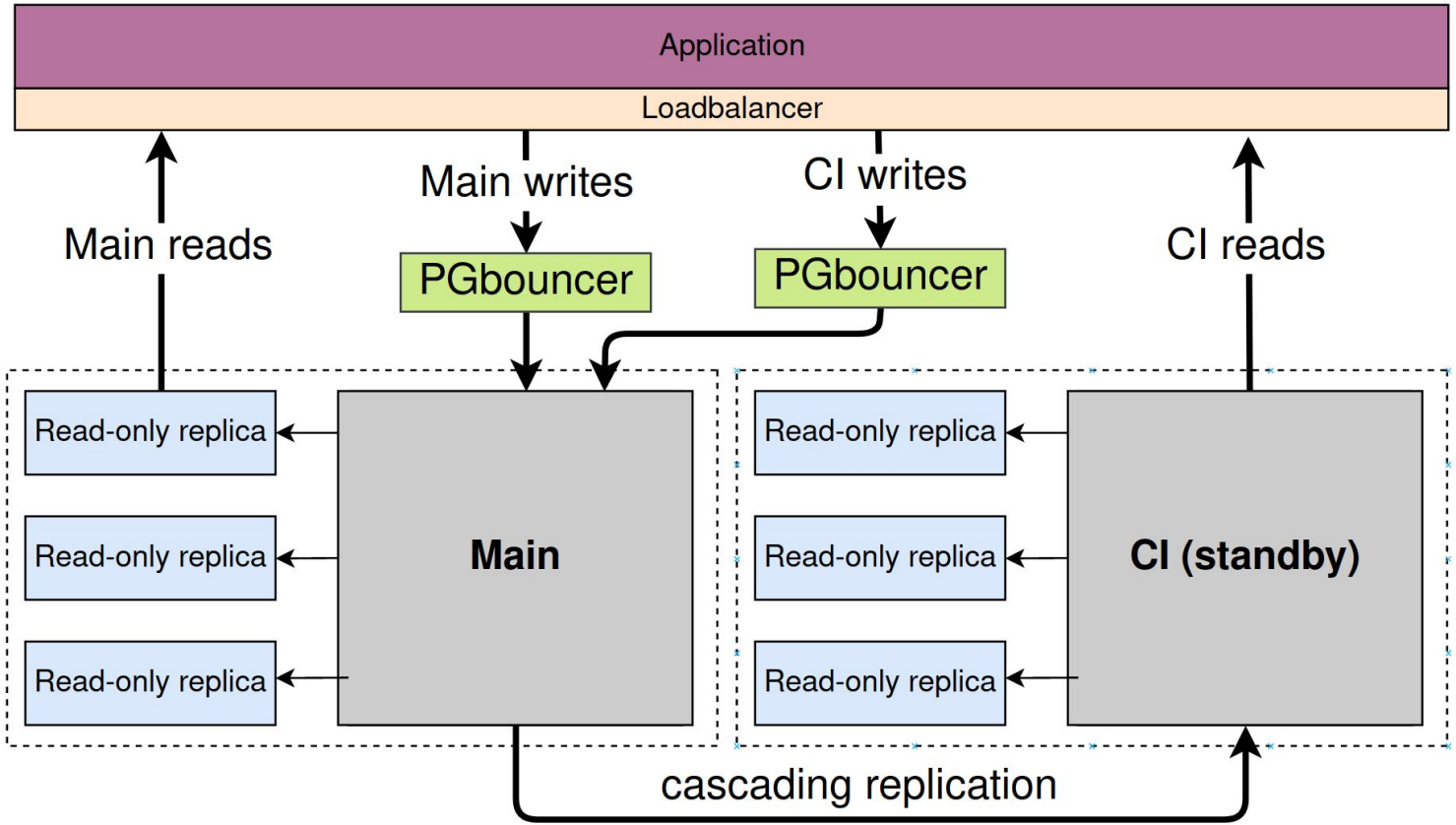
Decomposition Benefits

- Better write scaling through dedicated primaries
- Better tuning for the specific workload
- Smaller instances with less DML (INSERT, UPDATE, DELETE)
  - Faster backup and restore
  - Less stress on replication and archive
  - New standbys quicker to create and catch up
  - Faster VACUUM, less dead tuple
  - Major upgrades are faster
  - Lesser TXID consumption and fewer wraparounds

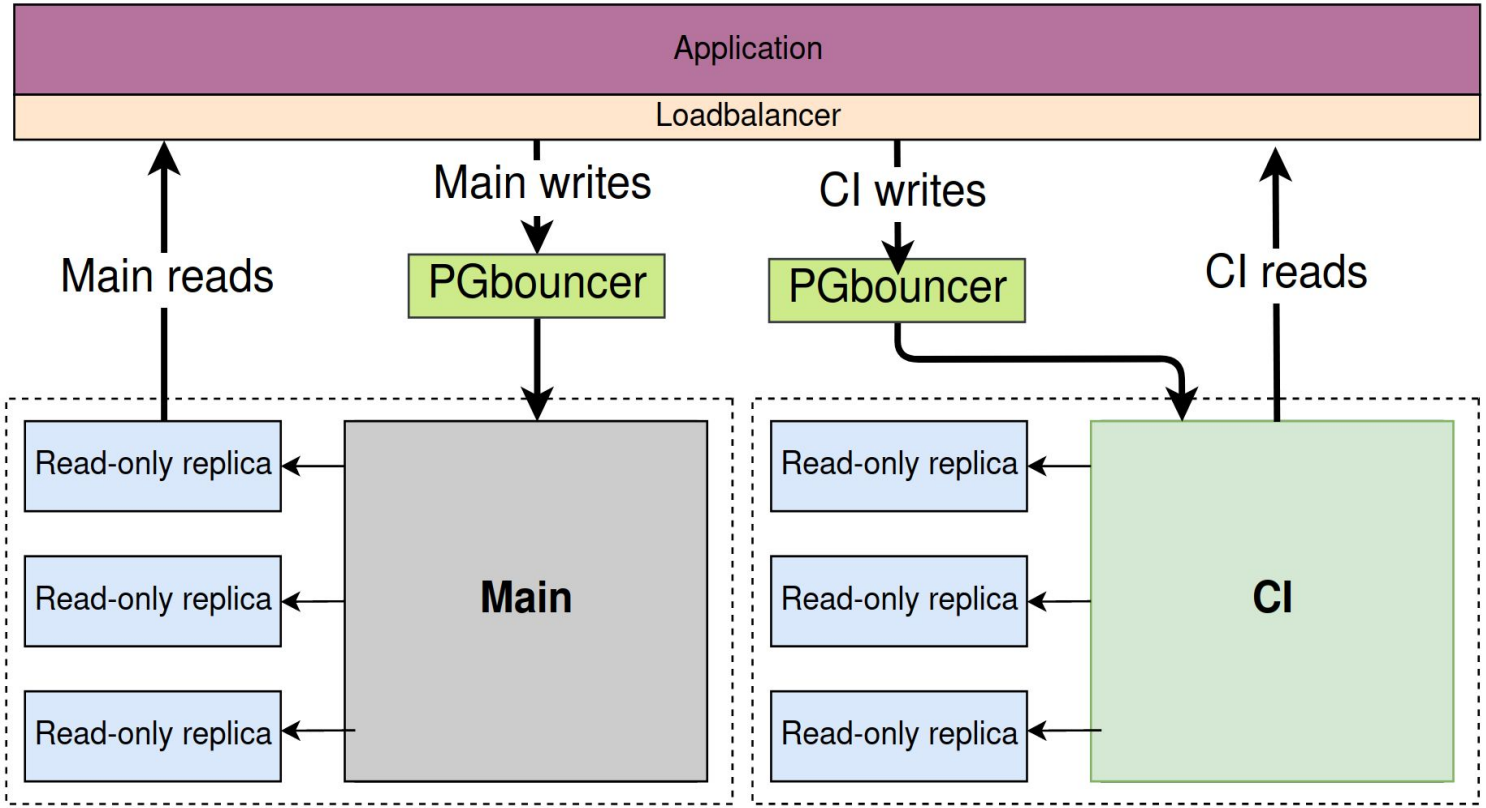
1. Clone as Standby Cluster



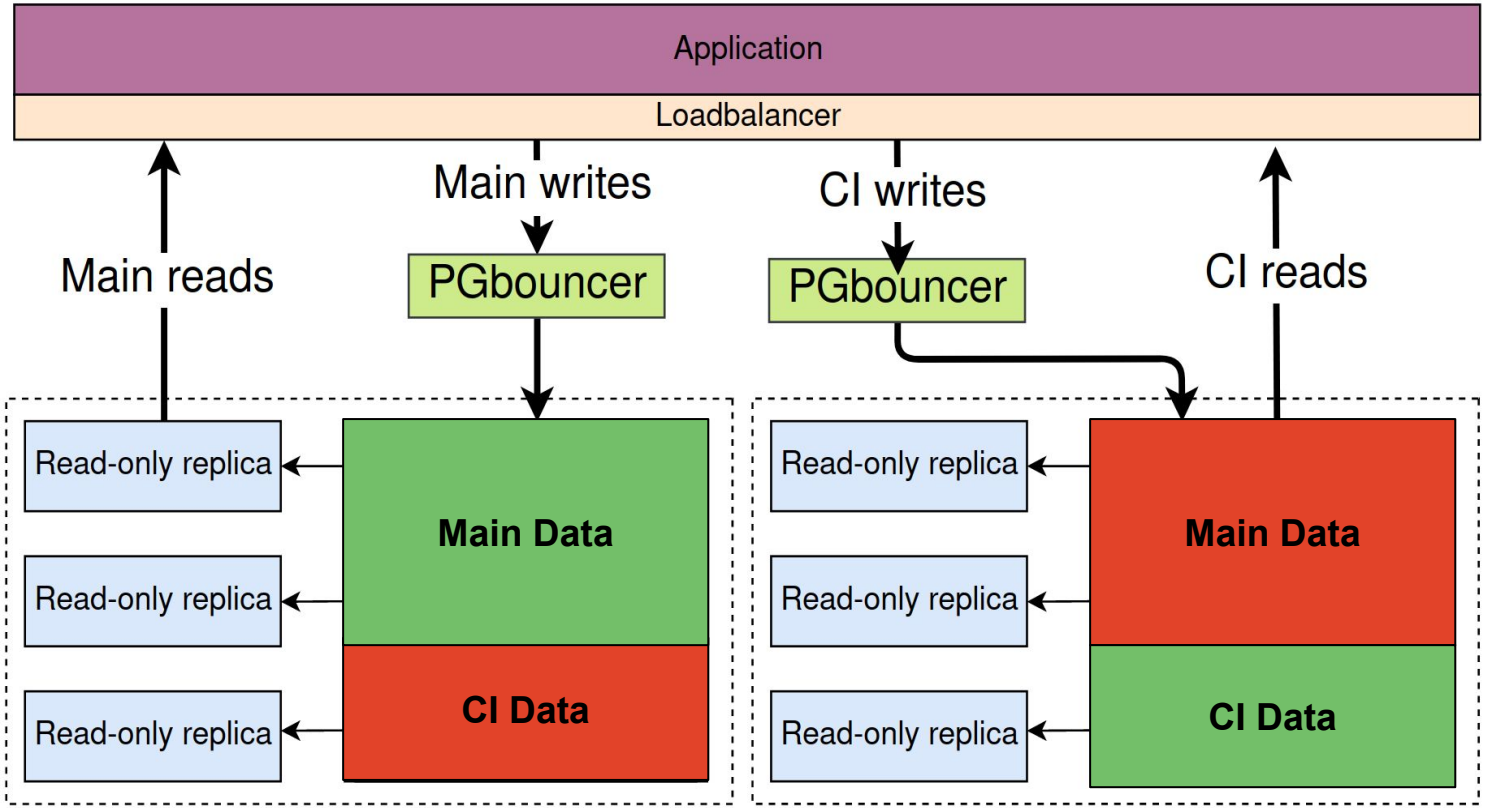
2. Dedicated CI Write Endpoint



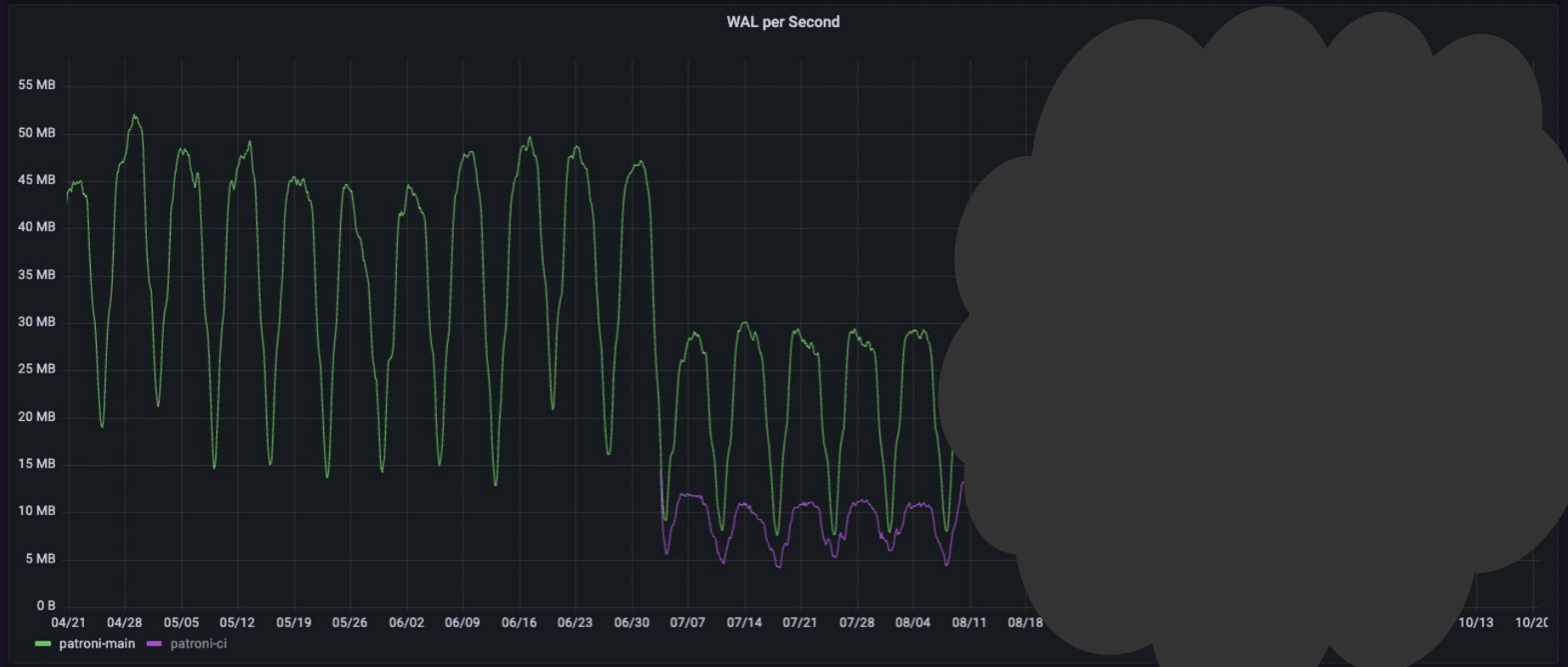
3. Promotion and Switchover



4. Cleanup and Scale Down

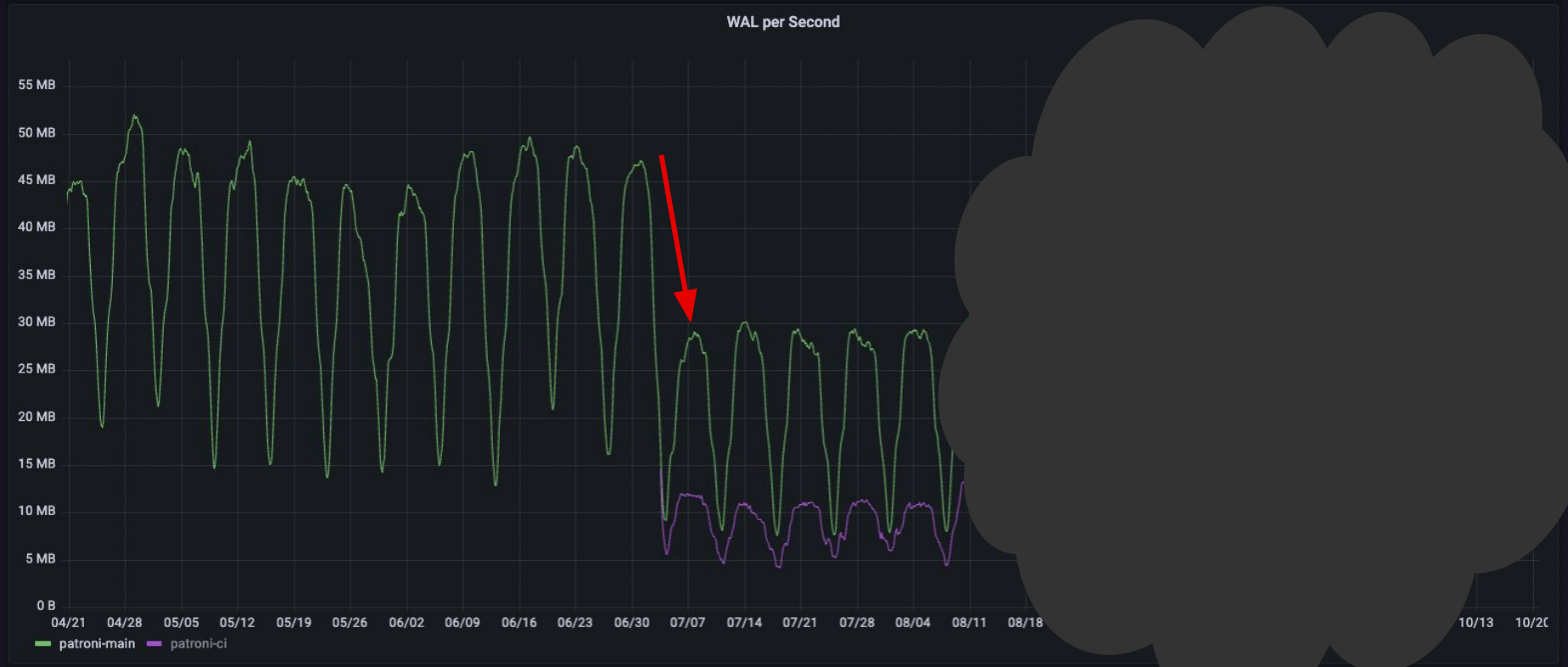


Decomposition - WAL Traffic



rate(pg_xlog_position_bytes[48h])

Decomposition - WAL Traffic



rate(pg_xlog_position_bytes[48h])

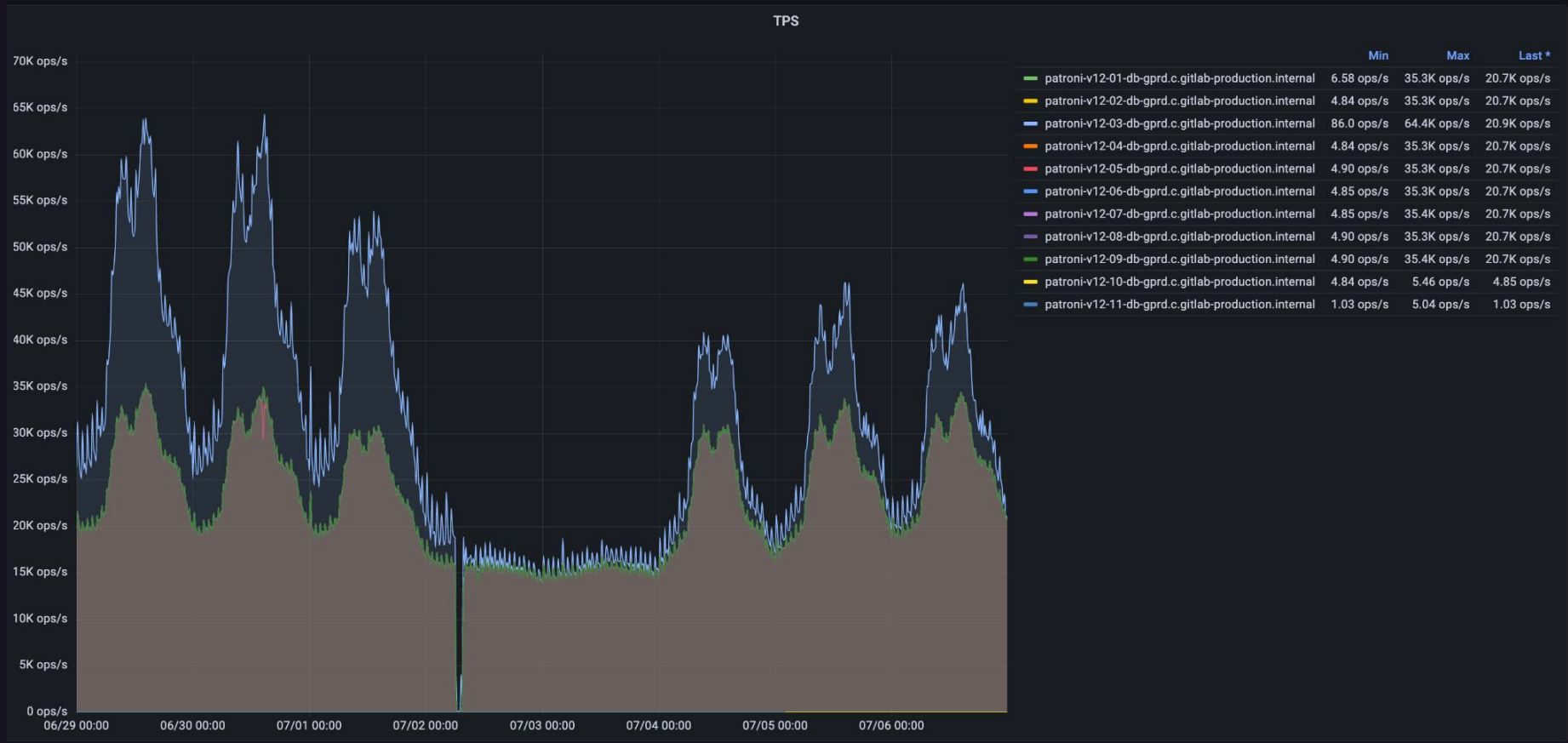
Decomposition - WAL Traffic

rate(pg_xlog_position_bytes[48h])

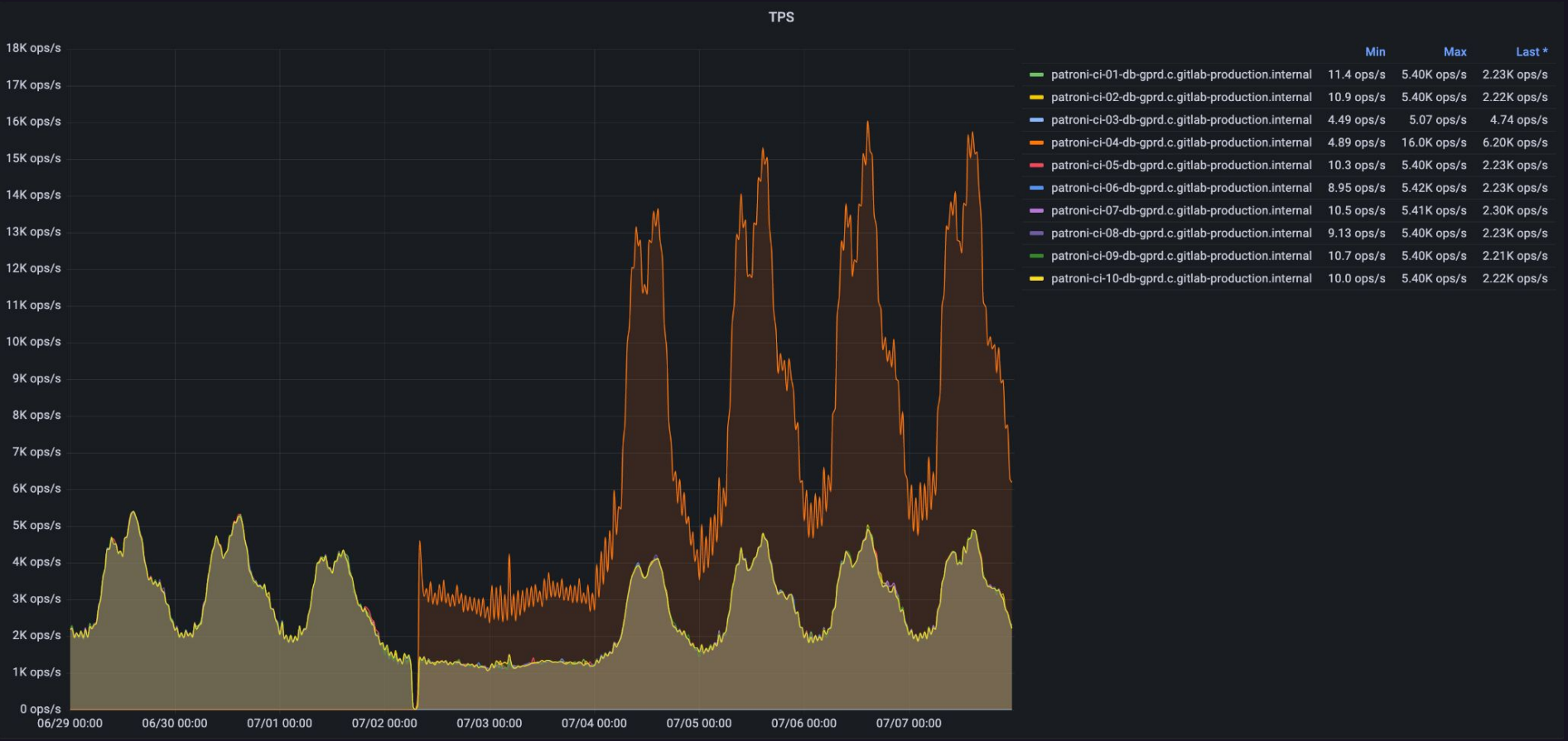
Decomposition - WAL Traffic

rate(pg_xlog_position_bytes[48h])

Decomposition - Main - TPS



Decomposition - CI - TPS



Decomposition - VACUUM

Additional Key Improvements

- Data size
  - Main 22 TB ⇒ 13 TB
  - CI 22 TB ⇒ 11 TB
- Significantly reduced dead tuples on Main
- Significantly reduced load caused by VACUUM
- Reduced write load on single node
- Average Sidekiq query duration reduced by factor >=5

WE ARE HIRING DBRE!

- All open positions boards.greenhouse.io/gitlab
- DBRE Americas boards.greenhouse.io/gitlab/jobs/4783681002
- DBRE APAC: boards.greenhouse.io/gitlab/jobs/6419765002

Resources

Resources

- GitLab: about.gitlab.com
- The Handbook: about.gitlab.com/handbook
- Our RDBMS: about.gitlab.com/handbook/engineering/infrastructure/database
- Decomposition:
  - about.gitlab.com/blog/2022/08/04/path-to-decomposing-gitlab-database-part1
  - about.gitlab.com/blog/2022/08/04/path-to-decomposing-gitlab-database-part2
  - about.gitlab.com/blog/2022/08/04/path-to-decomposing-gitlab-database-part3
- Jose Cores Finotto: about.gitlab.com/company/team/#Finotto
- Alexander Sosna: about.gitlab.com/company/team/#alexander-sosna

Questions?!