GitLab

PostgreSQL at GitLab.com

Jose Cores Finotto, Staff Database Reliability Engineer, Infrastructure

Speaker

- My name is 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.

Agenda

- Gitlab
- Team
- Architecture
- Problems and Goals
- Pgbouncer
- Upgrades
- Postgres Checkup

Gitlab Values

1

Collaboration

Work asynchronously with fully remote workforce (org)

Use GitLab to build GitLab, there's an Issue and/or Merge Request for everything

2

Results

Track outcomes, not hours

3

Efficiency

Straightforward solutions win. Complexity slows cycle time.

4

Diversity

Remote-only tends toward global diversity, but we still have a ways to go.

Hire those who add to culture, not those who fit with it. We want **cultural diversity** instead of cultural conformity.

5

Iteration

Minimum Viable Change (MVC) if the change is better than the existing solution, ship it.

6

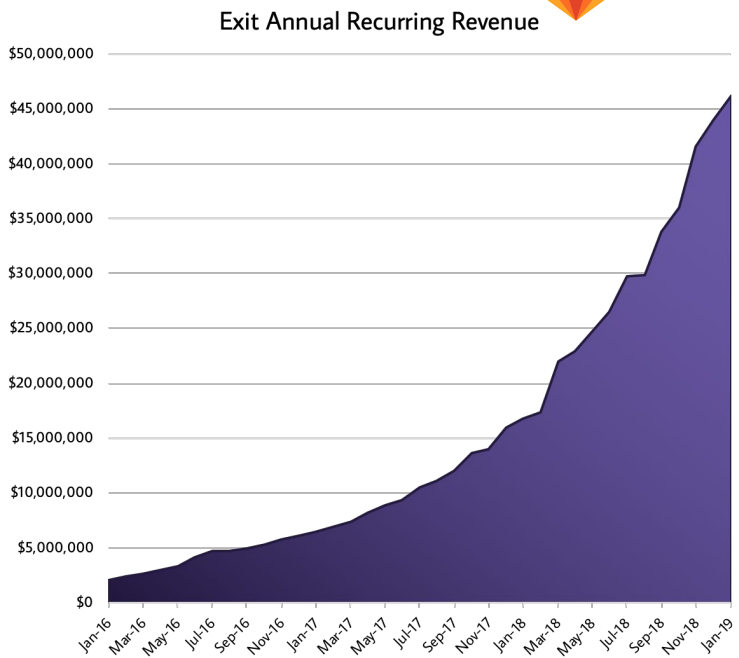
Transparency

Everything at GitLab is **public by default**: Strategy, Roadmap, Quarterly Goals, Handbook, and Issue Trackers

GitLab.com in numbers:

We have a hosted version of Gitlab:

- Over 25 million daily git pull operations.
- Upwards of 3K requests/second.
- More than 4k git requests per second.
- 650.000 git pushes a day.
- 40k to 60k transactions per second on the database
- 8 database replicas
- Database size : **5 TiB**



The team

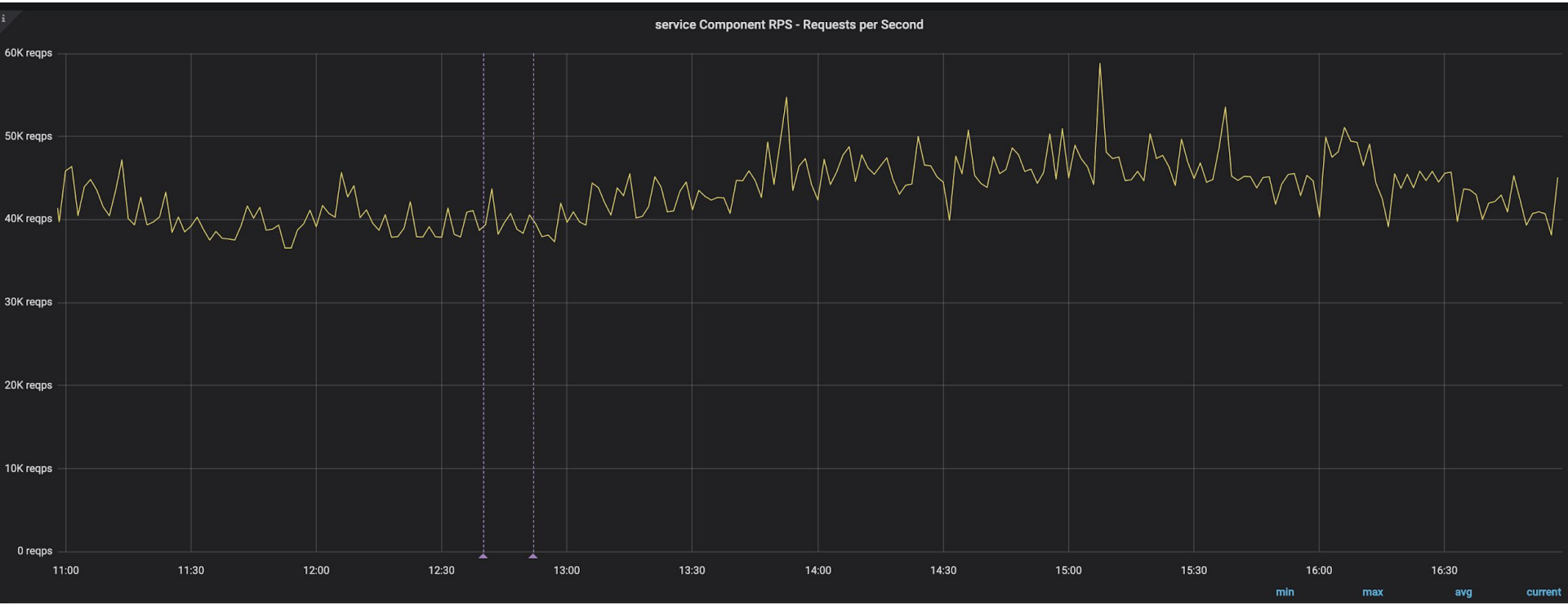
I have the pleasure of collaborating with 2 companies with extensive technical knowledge :

- OnGres - founded by Álvaro Hernández.
- Postgres.ai - founded by Nikolay Samokhvalov.
- And the support of the SRE team.

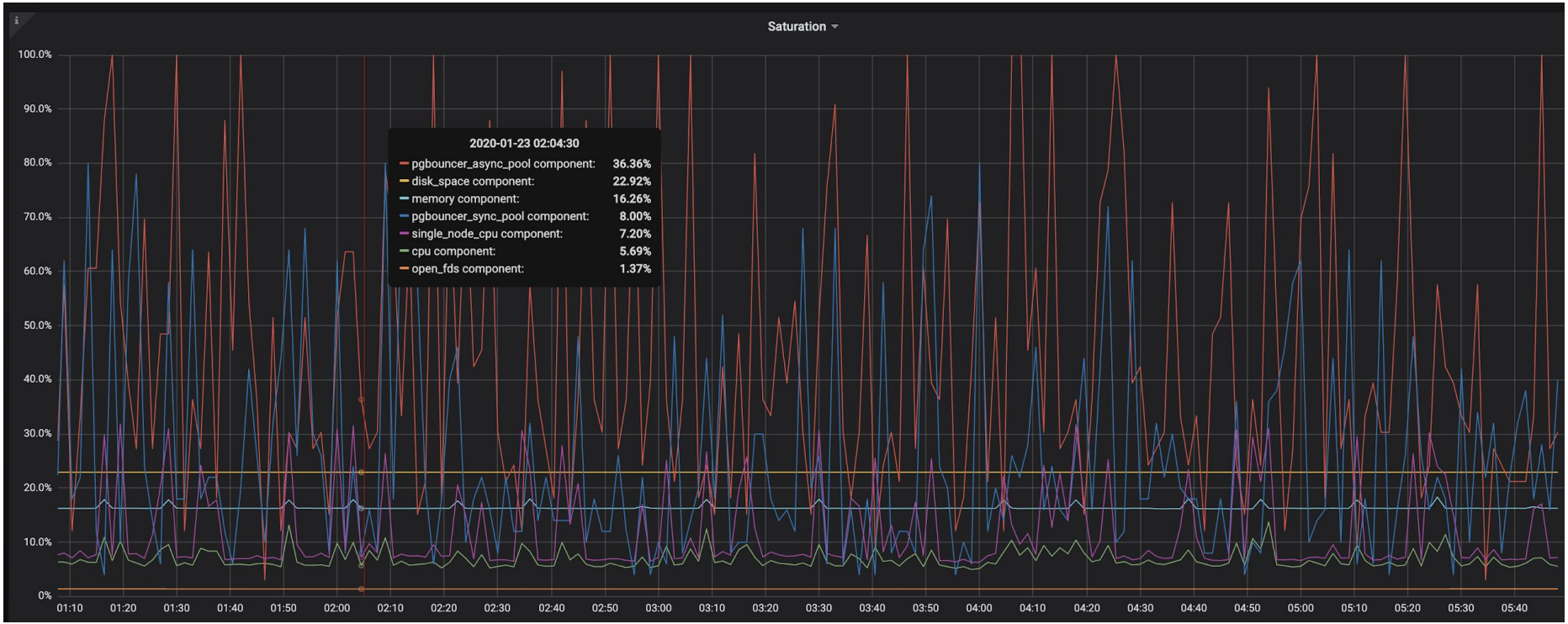
Projects and Issues

- PGBouncer saturation issue
- Delayed Replica
- Replicas without traffic
- Consul Setup
- Postgres-Checkup : performance on demand!
- Joe bot

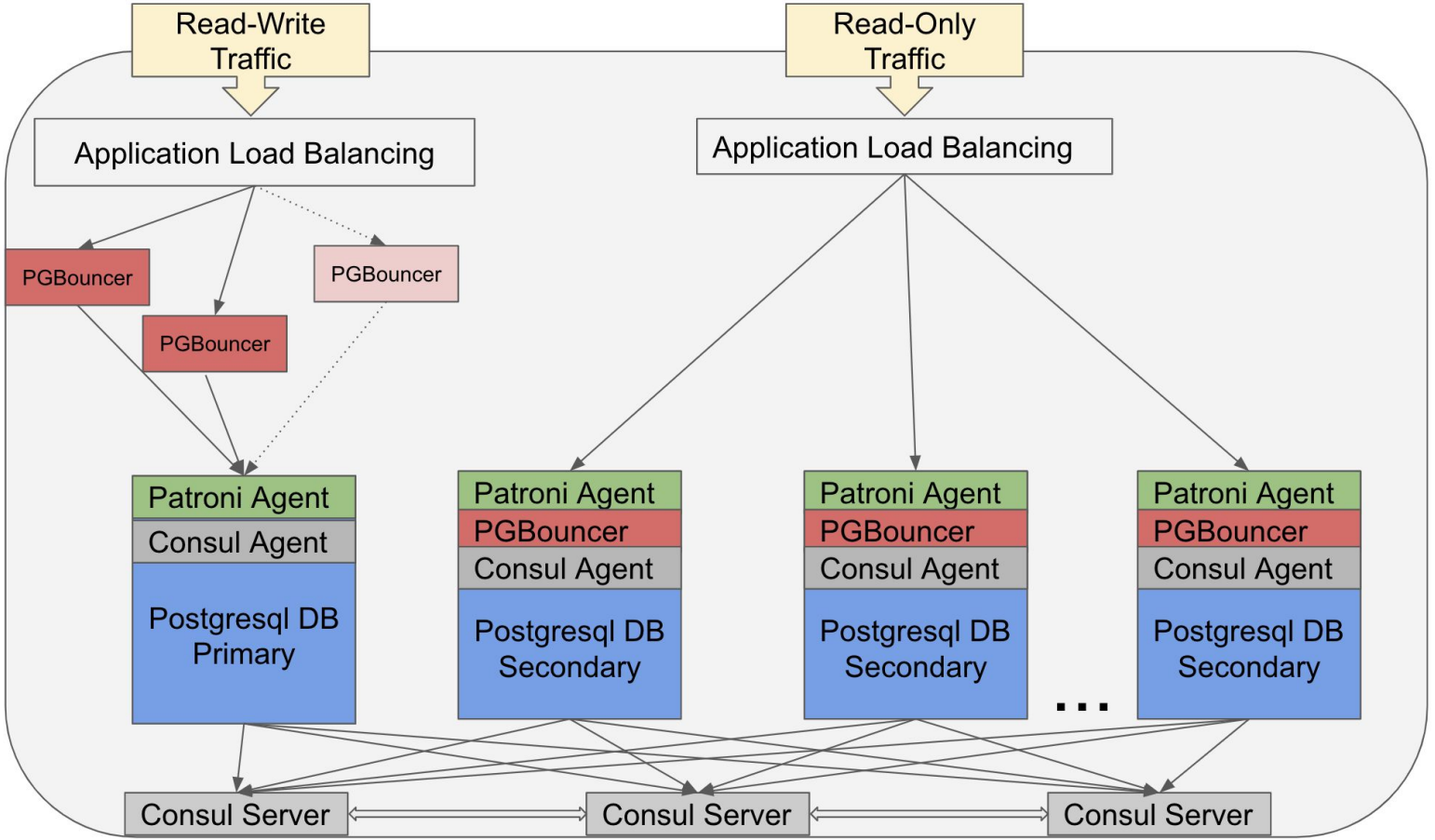
PGBouncer Traffic - Request per second



PGBouncer Saturation



PGBouncer Architecture in the beginning of 2019



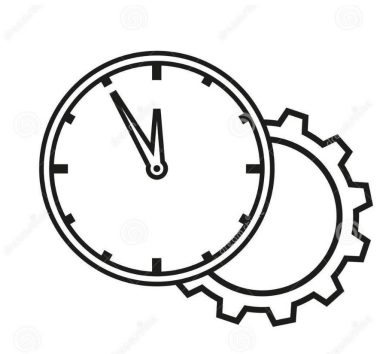
PGBouncer Pool Changes

Pool sizes Before :

Sync : 70

Async : 80

Geo : 10



Pool sizes After :

Read Write :
Sync 50
Async : 33
Geo : 1

Read only :

Sync : 40
Async : 2
GEO : 1

PGBouncer Read Only Changes

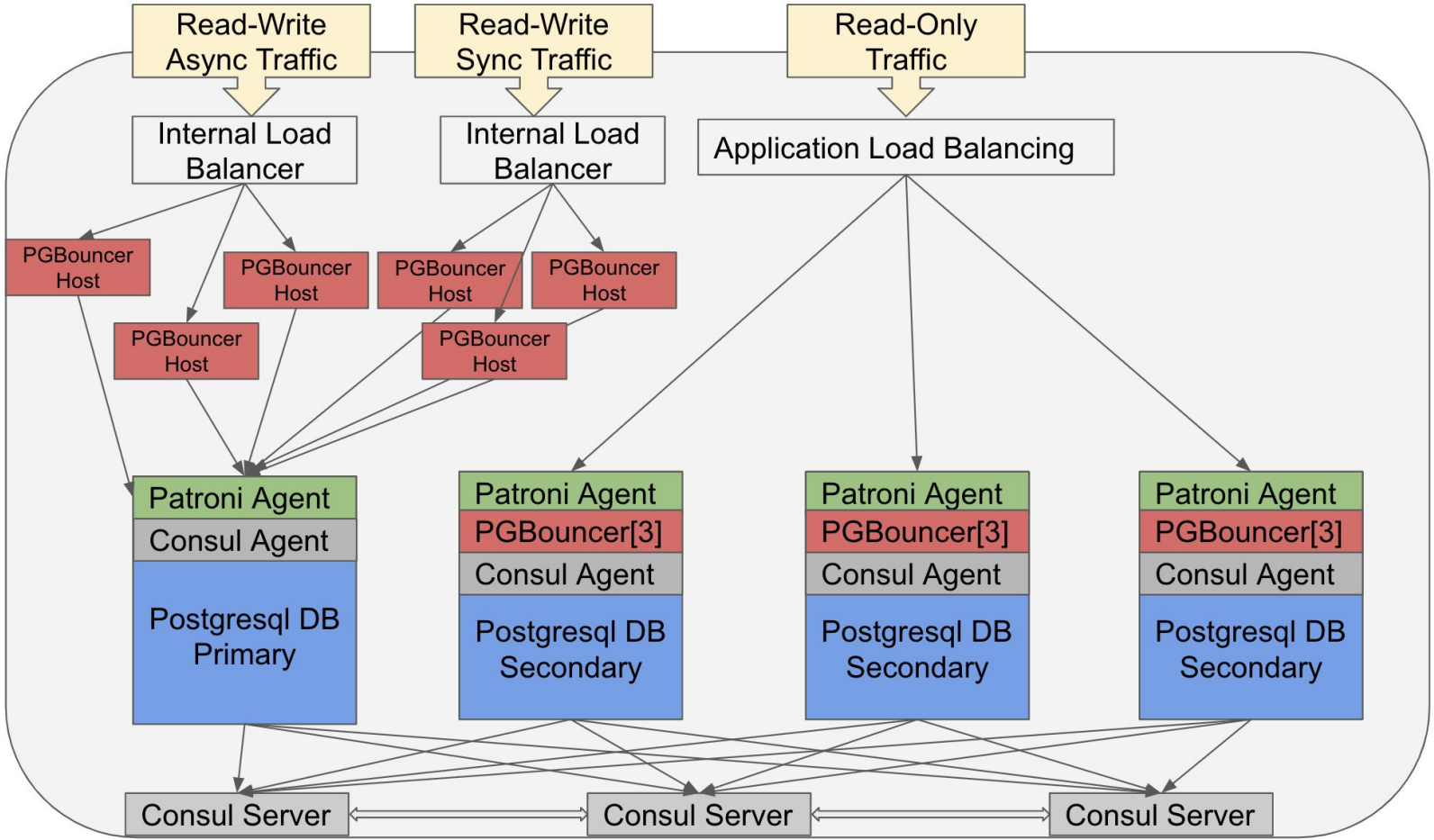
- The application had a limit of one port for pgbouncer host.
- The engineering team added support for more ports.
- Initially, we added one extra pgbouncer per read-only replica.
- Later on, we added a third instance of pgbouncer per read-only replica.
- We rolled out the instances and later we added them to the list of pgbouncers that are available. We use consul service for this process.
- Our performance problems with the read only databases was resolved.

PGBouncer Read Write Changes

- Our load balancer strategy wasn’t performing as expected.

- We noticed that the traffic was pretty different from the Async and Sync traffic. And the shared pools were affecting the performance.
- We decided to split the traffic between Async and Sync type of sessions.
- The easiest way to implement a better load balancing was by adding an extra ILB (Internal Load Balancer).
- and split the traffic in 3 nodes for each type of traffic.
- We rolled out the new ILB, and the extra pgbouncer nodes. Adding the new ILB to be used by the application, and the new nodes to the Sync ILB.

PGBouncer Architecture in the end of 2019



Delayed Replica

- We have a delayed replica that has the WALs being applied with a 12-hour delay.
- Our Intention here is to have a point to recover in case of accidental deletes, or the rollout of some features that caused some damage.
- This replica saved us several times.

Replicas without traffic

- We added the following 2 tags at the patroni config on 2 database read-only replicas :

```
tags:
  nofailover: true
  noloadbalance: true
```

- With these config nodes will not receive traffic and they are not candidates to become the primary.

What is the reason for it?

- We have these replicas up to date, and if any node fails, we have a fast replacement node ready.
- In case of a replica failure, we needed 4 hours to recreate a new node, and we faced a degraded performance till the replica is back.

Consul Setup

- After facing several failovers, due to possible networks glitches or hardware issues, we focused on making our consul setup more resilient :

- Adding in the consul/patroni setup:
  - checks[]
  - Raising the parameter : retry_timeout to 60.

- This change avoids the execution of Serf checks and raises the amount of time Patroni waits before restarting Postgres (in case of network failure), to avoid restarting the database in short networks glitches.

- It is important to note that Patroni developers are removing Serf checks by default (see this issue)

Postgres-checkup

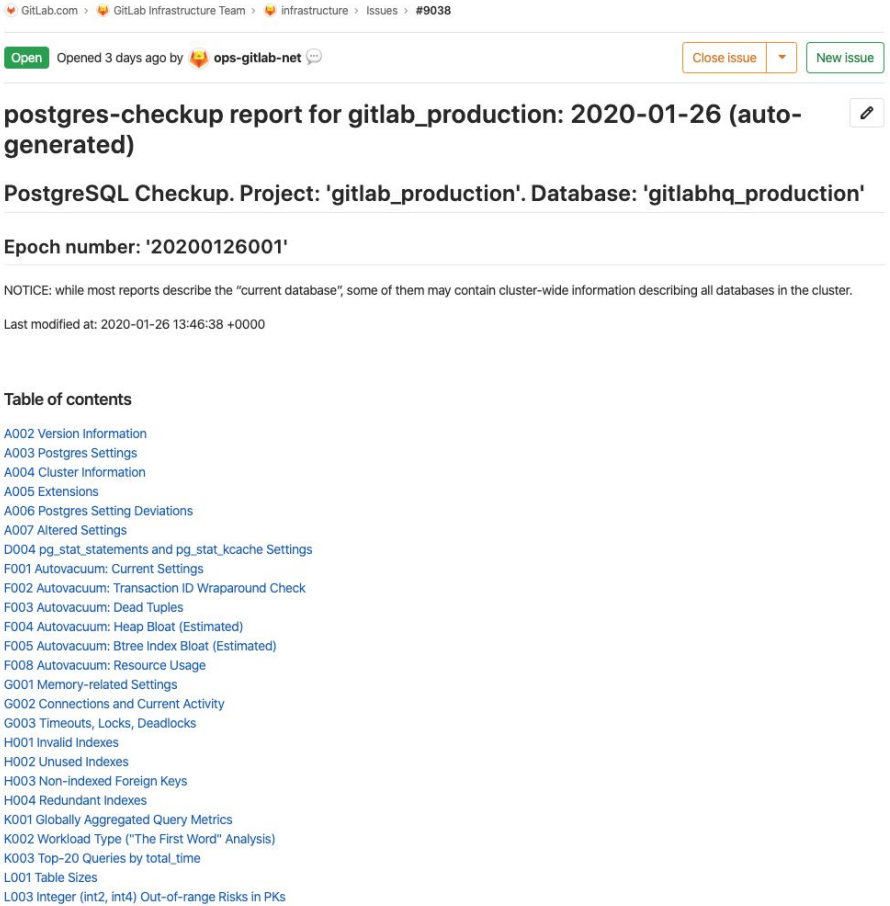
Nikolay and his team develop postgres-checkup (https://gitlab.com/postgres-ai/postgres-checkup) -- a tool for automated health-checks of Postgres databases, that contains:

- 28 reports, checking various aspects of Postgres production database health and performing detailed SQL workload analysis.
- Reports contain 3 detailed parts: observations, conclusions, and recommendations.
- Very lightweight checks, unobtrusive activities working well under heavy load, in large databases. Does not require any setup on the servers.
- Multi-node analysis: the master is checked together with its replicas.

Postgres-checkup

- GitLab.com database is checked twice per week. Some of the benefits are:
  - SQL workload analysis
  - query optimization is now a routine, periodical process.
  - The control over table and index bloat is established.
  - control over unused and redundant indexes.
  - Growth trend analysis is simplified.
  - Some old tables still use int4 primary keys, and postgres-checkup helps to control the ID values growth.
  - We now have a history of actual settings across all Postgres nodes, helpful for troubleshooting of various issues.

Postgres-checkup



GitLab.com › GitLab Infrastructure Team › infrastructure › Issues › #9038

Open Opened 3 days ago by ops-gitlab-net

Close issue | New issue

postgres-checkup report for gitlab_production: 2020-01-26 (auto-generated)

PostgreSQL Checkup. Project: 'gitlab_production'. Database: 'gitlabhq_production'

Epoch number: '20200126001'

NOTICE: while most reports describe the "current database", some of them may contain cluster-wide information describing all databases in the cluster.

Last modified at: 2020-01-26 13:46:38 +0000

Table of contents

Postgres-checkup

K003 Top-20 Queries by total_time

Observations

Data collected: 2020-01-26 13:40:15 +0000 UTC
Current database: gitlabhq_production

Master (`10.220.16.106`)

Start: 2020-01-26T13:05:41.311485+00:00
End: 2020-01-26T13:38:39.862806+00:00
Period seconds: 1978.55132
Period age: 00:32:58.551321

Error (calls): 0.00 (0.00%)
Error (total time): 0.00 (0.00%)

The list is limited to 20 items.

# (query id)	Query	Calls	▼ Total time	Rows
1 (337560700)	WITH RECURSIVE "namespaces_cte" AS ((SELECT "namespaces"."id", "members"."access_level" FROM "namespaces" INNER JOIN "members" ON "namespaces"."id" = "members"."source_id" WHERE "members"."type" IN (?) AND "members"."source_type" = ? AND "namespaces"."type" IN (?) AND "members"."user_id" = ? AND "members"."requested_at" IS NULL) UNION (SELECT "namespaces"."id", group_group_links.group_access AS access_level FROM "namespaces" INNER JOIN "group_group_links" ON "group_group_links"."shared_group_id" = "namespaces"."id" INNER JOIN "members" ON "group_group_links"."shared_with_group_id" = "members"."source_id" AND "members"."user_id" = ? WHERE "namespaces"."type" IN (?)) UNION (SELECT "namespaces"."id", GREATEST("members"."access_level", "namespaces_cte""."access_level")	72,694 36.74/sec 1.00/call 0.47%	1,976,352.32 ms 998.889 ms/sec 27.187 ms/call 41.08%	304,880,540 154.10K/sec 4.20K/call 43.10%

Joe bot

Postgres.ai's primary focus is building non-production environments to allow rapid development and testing processes.

- The main tool is called Database Lab (https://gitlab.com/postgres-ai/database-lab). It allows super-fast cloning of large databases:
- GitLab.com database, currently being 5 TiB in size, is cloned in less than 2 seconds.
- As a result, engineers get full-sized short-lived database copies (called "local thin clones") which are ready for any experiments such as verifying index ideas, SQL performance troubleshooting, or preparing database changes.

Database Lab

Joe bot

- On top of Database Lab works bot Joe, an SQL optimization assistant (https://github.com/postgres-ai/joe).

- It lives in our Slack channel, #database-lab, where more than 70 engineers use it to:
  - troubleshoot SQL query performance
  - get optimization insights
  - verify various optimization ideas (such as new indexes).

Joe bot

It is surprisingly simple:

- An engineer provides an SQL to Joe.
- Joe launches a session, based on local *thin clones* provided by Database Lab.
- Independent Postgres server working on the thin clone is used to get quick plan using EXPLAIN and then full execution plan using EXPLAIN (BUFFERS, ANALYZE), with actual numbers such as timing and buffer pool usage.
- The engineer gets the EXPLAIN ANALYZE details, with performance summary and recommendations.

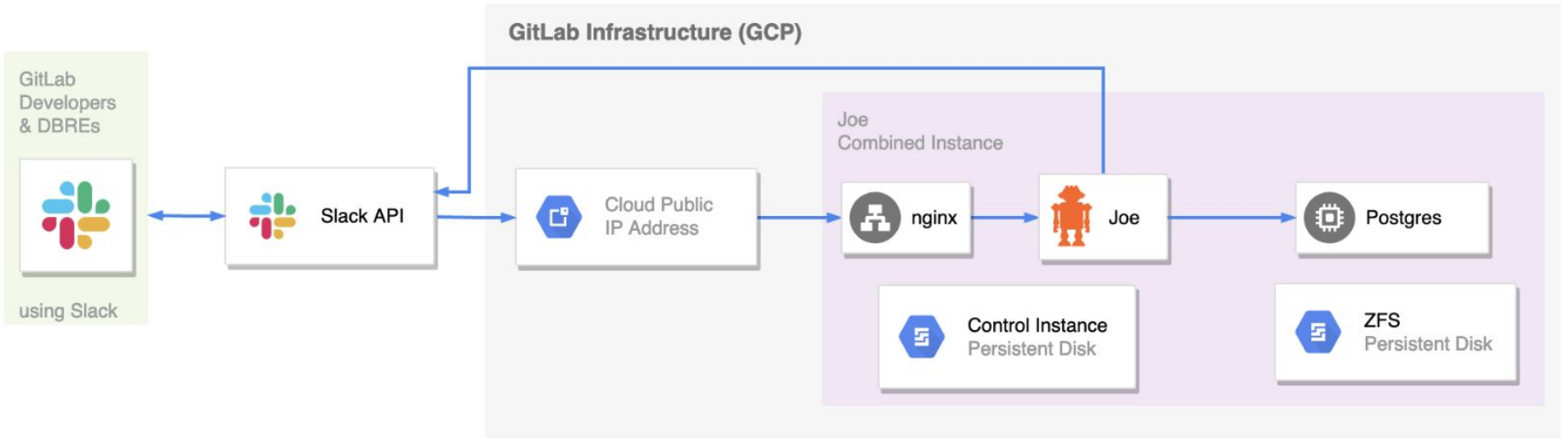
Joe bot

- On this independent database clone, the query is executed and EXPLAIN information is provided, with actual numbers such as timing and buffer pool usage.

- The engineer gets the EXPLAIN ANALYZE details, with performance summary and recommendations. More about Joe bot in GitLab:

  https://gitlab.com/gitlab-com/www-gitlab-com/merge_requests/24156/

SQL optimization lifecycle



Contribute