Database Performance at Gitlab.com

Jose Cores Finotto,
GitLab,
Staff Database Reliability Engineer

Nikolay Samokhvalov,
Postgres.ai,
Founder
Speaker: Jose Cores Finotto

- 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.
Speaker: Nikolay Samokhvalov

- Database systems:
  - 2002-2005:
  - since 2005:

- Worked on XML data type and functions (2005-2007)

- Long-term community activist – #RuPostgres, Postgres.tv

- Conferences Program Committee

- Current business: Postgres.ai
Agenda

- GitLab
- Architecture and challenges
- Performance analysis
- postgres-checkup
- Joe Bot & Database Lab
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
The open source project

- Used by more than 100,000 organizations
- A community of 3,000+ code contributors

We release every month on the 22nd and there is a publicly viewable direction for the product. Learn more from our blog →
GitLab Inc. is an open-core company that sells subscriptions that offer more features and support for GitLab. Learn about open core ➔

GitLab, the product is a complete DevOps platform, delivered as a single application, fundamentally changing the way Development, Security, and Ops teams collaborate. Learn more about our product ➔

All remote with 1297 team members

Over 30 million estimated registered users

Located in 67 countries
**2011**

GitLab, the open source project began.

Most of our internal procedures can be found in a publicly viewable 5000+ page handbook and our objectives are documented in our OKRs.

---

**2015**

We joined Y Combinator and started growing faster. Join our team.

Our mission is to change all creative work from read-only to read-write so that everyone can contribute. This is part of our overall strategy.

---

Our values are Collaboration, Results, Efficiency, Diversity, Inclusion & Belonging, Iteration, and Transparency (CREDIT) and these form an important part of our culture.

Our Tanuki (Japanese for raccoon dog) logo symbolizes our values with a smart animal that works in a group to achieve a common goal, you can download it on our press page.
The definitive guide to remote work [Download the playbook]

Discover a more streamlined way to work

<table>
<thead>
<tr>
<th>Manage</th>
<th>Plan</th>
<th>Create</th>
<th>Verify</th>
<th>Package</th>
<th>Secure</th>
<th>Release</th>
<th>Configure</th>
<th>Monitor</th>
<th>Protect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subgroups</td>
<td>Issue Tracking</td>
<td>Source Code Management</td>
<td>Continuous Integration (CI)</td>
<td>Package Registry</td>
<td>SAST</td>
<td>DAST</td>
<td>Continuous Delivery</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audit Events</td>
<td>Time Tracking</td>
<td>Code Review</td>
<td>Code Quality</td>
<td>Container Registry</td>
<td>Fuzz Testing</td>
<td>Dependency Scanning</td>
<td>Pages</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audit Reports</td>
<td>Boards</td>
<td>Wiki</td>
<td>Code Testing and Coverage</td>
<td>Helm Chart Registry</td>
<td>Dependency Proxy</td>
<td>License Compliance</td>
<td>Review Apps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compliance</td>
<td>Epics</td>
<td>Static Site Editor</td>
<td>Load Testing</td>
<td>Release Evidence</td>
<td>Secret Detection</td>
<td>Secret Detection</td>
<td>Advanced Deployments</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Management</td>
<td>Roadmaps</td>
<td>Web IDE</td>
<td>Web Performance</td>
<td>Accessibility Testing</td>
<td>Vulnerability Management</td>
<td>Vulnerability Management</td>
<td>Feature Flags</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Analytics</td>
<td>Service Desk</td>
<td>Live Preview</td>
<td>Usability Testing</td>
<td>Merge Trains</td>
<td>Git LFS</td>
<td>Git LFS</td>
<td>Release Orchestration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DevOps Reports</td>
<td>Requirements Management</td>
<td>Snippets</td>
<td>Accessibility Testing</td>
<td>Merge Trains</td>
<td>Git LFS</td>
<td>Git LFS</td>
<td>Secrets Management</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value Stream Management</td>
<td>Design Management</td>
<td>Source Code Management</td>
<td>Continuous Integration (CI)</td>
<td>Package Registry</td>
<td>SAST</td>
<td>DAST</td>
<td>Continuous Delivery</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

GitLab is a complete DevOps platform, delivered as a single application.
Feature development matrix

Exponential rate of product improvement

- Source Code Issues
- CI Wiki Labels
- GitLab Shell Groups Side by Side Diff
- Group Milestones Audit Logs Multiple LDAP servers
- Oauth Support GitLab.com CI Runner Docker Support
- Pipelines Container Registry Environments Issue Boards Cycle Analytics Time Tracking Review Apps Auto Deploy
- Board Milestones Deploy Boards Prometheus Integration Burndown Charts Canary Deploy Code Quality Auto DevOps Beta Kubernetes Epics SAST

about.gitlab.com/direction
GitLab.com in numbers:

We have a hosted version of Gitlab:

- Over 40 million daily git pull operations.
- More than 6k git requests per second.
- 750,000 git pushes a day.
- 60k to 80k transactions per second on the database.
- 7 database replicas and 1 primary.
- Database size: 9 TiB.
- Hardware architecture GCP 96 cores with 624 GiB of RAM.
Current Architecture

- Read-Write Async Traffic
  - Internal Load Balancer
    - PGBouncer Host
    - Patroni Agent
      - Consul Agent
      - Postgresql DB Primary
    - PGBouncer Host
  - PGBouncer Host
  - PGBouncer Host
  - Consul Agent

- Read-Write Sync Traffic
  - Internal Load Balancer
    - PGBouncer Host
    - Patroni Agent
      - Consul Agent
      - Postgresql DB Secondary
    - PGBouncer Host
  - PGBouncer Host
  - PGBouncer Host
  - Consul Agent

- Read-Only Traffic
  - Application Load Balancing
    - PGBouncer Host
    - Patroni Agent
      - Consul Agent
      - Postgresql DB Secondary
    - PGBouncer Host
  - PGBouncer Host
  - PGBouncer Host
  - Consul Agent

- Consul Server
  - Consul Server
  - Consul Server
Performance degradation analysis

Database performance peak - 12 of January - 16:06 AM

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

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.
Performance degradation analysis

Jose Finotto @Finotto · 1 day ago

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

Query:

```sql
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%2C20%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%2C%20instance%3D%22patroni-06-db-gprd.c.gitlab-production.internal%3A9187%22%7D%5B1m%5D)%0A%20%20)%0A)&g0.tab=0
Performance degradation analysis

The outputs are:

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])
})
Performance degradation analysis

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

Execute
- insert metric at cursor

Graph
Console

<table>
<thead>
<tr>
<th>Element</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(queryid=“3926004648916863976“)</td>
<td>0.8178287140222236</td>
</tr>
<tr>
<td>(queryid=“638689082264677652“)</td>
<td>0.690979611506127</td>
</tr>
<tr>
<td>(queryid=“716430218213446947“)</td>
<td>0.5237485821202116</td>
</tr>
<tr>
<td>(queryid=“850769644791286491“)</td>
<td>0.2840517462796186</td>
</tr>
<tr>
<td>(queryid=“909562959370256100“)</td>
<td>0.2500309345328653</td>
</tr>
<tr>
<td>(queryid=“402488551284107289“)</td>
<td>0.23028561521334467</td>
</tr>
<tr>
<td>(queryid=“717285160720443674“)</td>
<td>0.20701351823647401</td>
</tr>
<tr>
<td>(queryid=“2298083782068575032“)</td>
<td>0.15770600044224</td>
</tr>
<tr>
<td>(queryid=“5002940052336095544“)</td>
<td>0.1247541151170224</td>
</tr>
<tr>
<td>(queryid=“736671010424350814“)</td>
<td>0.12231413964376164</td>
</tr>
</tbody>
</table>
Performance degradation analysis

Jose Finotto @Finotto · 1 day ago

Those queryids are the following SQL statements:

<table>
<thead>
<tr>
<th>QueryId</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>3926004648916863976</td>
<td><code>SELECT &quot;ci_builds&quot;.* FROM &quot;ci builds&quot; INNER JOIN &quot;projects&quot; ON &quot;projects&quot;.&quot;id&quot; = &quot;ci_builds&quot;.&quot;project_id&quot; JOIN (SELECT &quot;ci_builds&quot;.&quot;project_id&quot; = project_features.&quot;project_id&quot; LEFT JOIN (SELECT &quot;ci_builds&quot;.&quot;project_id&quot;, COUNT(*) AS count(*) FROM &quot;ci_builds&quot; WHERE &quot;ci_builds&quot;.&quot;type&quot; = $1 AND (&quot;ci_builds&quot;.&quot;status&quot; IN ($2)) AND &quot;ci_builds&quot;.&quot;runner_id&quot; IN (SELECT &quot;ci_r&quot;.&quot;runner_id&quot; FROM &quot;ci_r&quot; WHERE &quot;runner_type&quot; = $3)) GROUP BY &quot;ci_builds&quot;.&quot;project_id&quot;) AS project_builds ON ci_builds.&quot;project_id&quot; IN (SELECT ci_builds.&quot;project_id&quot; FROM &quot;ci_builds&quot; WHERE ci_builds.&quot;status&quot; IN ($4)) AND ci_builds.&quot;runner_id&quot; IS NULL AND &quot;projects&quot;.&quot;shared_runners_enabled&quot; = $6 AND (project_features.&quot;builds_access_level&quot; IS NULL OR project_features.&quot;builds_access_level&quot; &gt; $7) AND (&quot;projects&quot;.&quot;visibility_level&quot; = $9 OR EXISTS (WITH RECURSIVE &quot;base_and_ancestors&quot; AS (SELECT &quot;namespaces.id&quot; = projects.&quot;namespace_id&quot;) UNION (SELECT &quot;namespaces.id&quot; FROM &quot;namespaces&quot; WHERE &quot;namespaces.id&quot; = &quot;base_and_ancestors&quot;.&quot;parent_id&quot;) SELECT $10 FROM &quot;base_and_ancestors&quot; A) UNION (SELECT &quot;namespaces.statistics&quot; ON namespace_statistics WHERE &quot;namespace.id&quot; = namespaces.&quot;id&quot; WHERE &quot;namespace.&quot;name&quot; = COALESCE(namespaces.&quot;shared_runners_minutes_limit&quot;, $11, $12) = $13 OR COALESCE(namespaces.&quot;shared_runner_minutes_limit&quot; + COALESCE(namespaces.&quot;extra_shared_runners_minutes_limit&quot;, $17), $18) * $19)) AND (NOT EXISTS (SELECT &quot;taggings&quot;.&quot;taggable_type&quot; = $21 AND &quot;taggings&quot;.&quot;context&quot; = $22 AND (taggable_id = ci_builds.&quot;id&quot; OR taggable_id = ci_builds.&quot;id&quot;)) ORDER BY COALESCE(project_builds.&quot;running&quot;, $25) ASC, ci_builds.&quot;id&quot; ASC) /application/web/correlation_id:01EVX3GF3VGA6TYFMR82EJFN/</code></td>
</tr>
<tr>
<td>-6386890822646776524</td>
<td><code>SELECT &quot;users&quot;.* FROM &quot;users&quot; INNER JOIN &quot;project_authorizations&quot; ON &quot;users&quot;.&quot;id&quot; = &quot;project_authorizations&quot;.&quot;project_id&quot; = $1 /application/web/correlation_id:LmzS5aAf8Vpa/</code></td>
</tr>
<tr>
<td>716430218231446947</td>
<td><code>UPDATE &quot;ci_builds&quot; SET &quot;runner_id&quot; = 380987, &quot;status&quot; = 'running', &quot;started_at&quot; = '2020-10-29 21:00:00' &quot;updated_at&quot; = '2020-10-29 21:00:54', &quot;lock_version&quot; = 2 WHERE &quot;ci_builds&quot;.&quot;id&quot; = 82015777 /application/web/correlation_id:42e9HF2IXC9/</code></td>
</tr>
</tbody>
</table>
| 6507699644791286491 | `SELECT SUM(("project_statistics"."repository_size" + "project_statistics"."lfs_objects_size") - "project_inner JOIN routes.rs ON rs.resource_id = projects.id AND rs.resource_type = 'Project' INNER JOIN "project_statistics"
"project_id" = "projects"."id" WHERE (rs.path LIKE 'gitlab-org/%') AND ("project_statistics"."repository_size") > "projects"."repository_size_limit" AND "projects"."repository_size_limit" /application/web/controller:merge_requests.action:index;correlation_id:HlfxW7Ir86b/` |
We had the following top 10 statements *by total calls* in execution during this peak:

Query:

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

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.moment_input=2021-01-08%2014%3A15%3A00&g0.max_source_resolution=0s&g0.expr=topk(10%2C20%0A%20%20%20sum%20by%20(queryid)%20(%0A%20%20%20rate(pg_stat_statements_calls%7Benv%3D%22gprd%22%2C%20monitor%3D%22db%22%2C%20type%3D%22patroni%22%2C%20instance%3D%22patroni-06-db-gprd.c.gitlab-production.internal%3A9187%22%7D%5B1m%5D)%0A%20%20)%0A)&g0.tab=0

Edited by Jose Finotto 1 day ago
Performance degradation analysis

The outputs are:

topk(10, 
    sum by (queryId) {
        rate(pg_stat_statements_calls{env="gprdc", monitor="db", type="patroni", instance="patroni-06-db-gprdc.cgitlab-production.internal:9187"}[1m])
    }
)

Graph: Console

- 15m  →  2021-01-12 16:15  →  10  →  stacked  Only raw data
### Performance degradation analysis

Those query IDs are the following SQL statements:

<table>
<thead>
<tr>
<th>QueryId</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>833913155023572892</td>
<td>SELECT $1</td>
</tr>
<tr>
<td>73367110635711796</td>
<td>SELECT &quot;projects&quot;.* FROM &quot;projects&quot; WHERE &quot;projects&quot;.&quot;id&quot; = $1 LIMIT $2 /application/web/controller/issues/action:index,correlation_id:tt4UcI9K5U9/</td>
</tr>
<tr>
<td>6769309683899657633</td>
<td>SELECT &quot;routes&quot;.* FROM &quot;routes&quot; WHERE &quot;routes&quot;.&quot;source_id&quot; = $1 AND &quot;routes&quot;.&quot;source_type&quot; = $2 LIMIT $3 /application/web/controller/issues/action:index,correlation_id:tt4UcI9K5U9/</td>
</tr>
<tr>
<td>6974950735891200787</td>
<td>SELECT &quot;namespaces&quot;.* FROM &quot;namespaces&quot; WHERE &quot;namespaces&quot;.&quot;id&quot; = $1 LIMIT $2 /application/web,correlation_id:e7d284e6-07ff-4c0e-a44a-e6880d46b20a/</td>
</tr>
<tr>
<td>6749620766035719574</td>
<td>SELECT &quot;taggings&quot;.* FROM &quot;taggings&quot; WHERE &quot;taggings&quot;.&quot;taggable_id&quot; = $1 AND &quot;taggings&quot;.&quot;taggable_type&quot; = $2 /application/web/controller:projects/action:show,correlation_id:ZidjvemXja/</td>
</tr>
<tr>
<td>6504150523421693673</td>
<td>SELECT &quot;tags&quot;.* FROM &quot;tags&quot; INNER JOIN &quot;taggings&quot; ON &quot;tags&quot;.&quot;id&quot; = &quot;taggings&quot;.&quot;tag_id&quot; WHERE &quot;taggings&quot;.&quot;taggable_id&quot; = $1 AND &quot;taggings&quot;.&quot;taggable_type&quot; = $2 AND (taggings.context = $3 AND taggings.tagger_id IS NULL) /application/web,correlation_id:dnT2GxhKuX2/</td>
</tr>
</tbody>
</table>
| -2372450153195223637| SELECT $1 AS one FROM ((SELECT "ci_runners".* FROM "ci_runners" INNER JOIN "ci_runner_projects"."runner_id" = "ci_runners"."id" WHERE "ci_runner_projects"."project_id" = $2) UNION ALL (SELECT "ci_runners".* FROM "ci_runners" INNER JOIN "ci_runner_namespaces" ON "ci_runner_namespaces"."id" = "ci_runners"."id" INNER JOIN "namespaces" ON "namespaces"."id" = "ci_runner_namespaces"."namespace_id" AND "namespaces"."type" = $3 WHERE "namespaces"."id" IN (WITH RECURSIVE "base_and_ancestors" AS (SELECT "namespaces".* FROM "namespaces" INNER JOIN "projects" ON "projects"."namespace_id" = "namespaces"."id" WHERE "namespaces"."type" = $4 AND "projects"."id" = $5) UNION (SELECT "namespaces".* FROM "namespaces", "base_and_ancestors" WHERE "namespaces"."id" IN (SELECT "namespaces"."id" FROM "namespaces" UNION ALL SELECT "namespaces"."base_namespace_id" FROM "namespaces"))) WHERE "base_and_ancestors"."id" IN (SELECT "namespaces"."id" FROM "namespaces") LIMIT $1)) WHERE "ci_runners"."id" = $1)
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

- Weekly tech audit reports that augment the existing monitoring (prometheus, postgres_exporter, grafana, thanos):
  - track Postgres and components versions
  - track settings and setting deviations
  - bloat control (tables, indexes)
  - index health (invalid, unused, redundant, etc)
  - deep query analysis
  - object sizes
  - int4 PKs
  - ... and more
## H002 Unused Indexes

### Observations

Data collected: 2021-01-11 13:40:38 +0000 UTC  
Current database: github_production  
Stats reset: 6 mon 27 days 14:25:00 ago (2020-06-13 23:13:01 +0000 UTC)

### Never Used Indexes

This list is limited to 50 items. Total: 178.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>ci_builds</td>
<td>index_ci_builds_on_protected</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>165.60 GB</td>
<td>7.27 TB</td>
</tr>
<tr>
<td>2</td>
<td>ci_builds</td>
<td>index_ci_builds_on_user_id_and_created_at_and_type_eq_ci_build</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>45.52 GB</td>
<td>0.92 TB</td>
</tr>
<tr>
<td>3</td>
<td>ci_builds</td>
<td>index_ci_builds_on_queued_at</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20.01 GB</td>
<td>0.92 TB</td>
</tr>
<tr>
<td>4</td>
<td>merge_request_diffs</td>
<td>index_merge_request_diffs_on_external_diff_score</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8.41 GB</td>
<td>27.54 GB</td>
</tr>
<tr>
<td>5</td>
<td>projects</td>
<td>index_projects_on_runners_token</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3.90 GB</td>
<td>4.77 GB</td>
</tr>
<tr>
<td>6</td>
<td>projects</td>
<td>index_projects_on_mirror_last_successful_update_at</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3.90 GB</td>
<td>4.77 GB</td>
</tr>
<tr>
<td>7</td>
<td>projects</td>
<td>index_projects_on_last_repository_check_failed</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3.90 GB</td>
<td>4.77 GB</td>
</tr>
<tr>
<td>8</td>
<td>projects</td>
<td>index_projects_on_pending_delete</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3.90 GB</td>
<td>4.77 GB</td>
</tr>
<tr>
<td>9</td>
<td>users</td>
<td>index_users_on_accepted_term_id</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3.90 GB</td>
<td>4.77 GB</td>
</tr>
<tr>
<td>10</td>
<td>ci_runners</td>
<td>index_ci_runners_on_is_shared</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.03 GB</td>
<td>337.54 MB</td>
</tr>
<tr>
<td>11</td>
<td>merge_request_metrics</td>
<td>index_mr_metrics_on_target_project_id_merged_at_time_to_merge</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16.00 GB</td>
<td>6.86 GB</td>
</tr>
<tr>
<td>12</td>
<td>notes</td>
<td>note_mentions_temp_index</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16.00 GB</td>
<td>6.86 GB</td>
</tr>
<tr>
<td>13</td>
<td>namespaces</td>
<td>index_namespaces_on_shared_and_extra_runners_minutes_limit</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.27 GB</td>
<td>2.56 GB</td>
</tr>
<tr>
<td>14</td>
<td>namespaces</td>
<td>index_namespaces_on_idap_sync_last_update_at</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.22 GB</td>
<td>2.56 GB</td>
</tr>
</tbody>
</table>
## K003 Top-50 Queries by total_time

### Observations

Data collected: 2021-01-11 13:40:41 +0000 UTC
Current database: gitlabhq_production

**Master (10.220.16.106)**

- **Start:** 2021-01-11T13:05:07.091966+00:00
- **End:** 2021-01-11T13:07.728772+00:00
- **Period seconds:** 1990.6368
- **Period age:** 00:33:10.636604

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

The list is limited to 50 items.

<table>
<thead>
<tr>
<th># (query id)</th>
<th>Query</th>
<th>Calls</th>
<th>▼ Total time</th>
<th>Rows</th>
<th>shared_bikes_hit</th>
<th>shared_bikes_read</th>
<th>shared_bikes_dirtied</th>
<th>share</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (-63986890622646777000)</td>
<td>SELECT &quot;users&quot;.* FROM &quot;users&quot; INNER JOIN &quot;project_authorizations&quot; ON &quot;users&quot;.&quot;id&quot; = &quot;project_authorizations&quot;.&quot;user_id&quot; WHERE &quot;project_authorizations&quot;.&quot;project_id&quot; = 1 AND &quot;application_web.correlation_id&quot;.column=&quot;mySqlVpsa&quot;/Full query</td>
<td>72,767</td>
<td>1,140,899.14 ms</td>
<td>7,371,979</td>
<td>33,869,906 hits</td>
<td>816,616 hits</td>
<td>4,870 blks</td>
<td>148</td>
</tr>
<tr>
<td>2 (-72320844478659837000)</td>
<td>WITH RECURSIVE &quot;namespaces_cte&quot; AS (SELECT &quot;namespaces&quot;.&quot;id&quot;, &quot;members&quot;.&quot;access_level&quot; FROM &quot;namespaces&quot; INNER JOIN &quot;members&quot; ON &quot;namespaces&quot;.&quot;id&quot; = &quot;members&quot;.&quot;source_id&quot; WHERE &quot;members&quot;.&quot;type&quot; = 1 AND &quot;members&quot;.&quot;source_type&quot; IS NULL AND (&quot;access_level&quot; IS $0)) UNION SELECT &quot;namespaces&quot;.&quot;id&quot;, LEAST(&quot;members&quot;.&quot;access_level&quot;, &quot;group_group_links&quot;.&quot;group_access&quot;) AS access_level FROM &quot;namespaces&quot; INNER JOIN &quot;group_group_links&quot; ON &quot;group_group_links&quot;.&quot;shared_group_id&quot; = 1 AND &quot;members&quot;.&quot;source_id&quot; = 1 AND &quot;members&quot;.&quot;source_type&quot; = 1 AND &quot;members&quot;.&quot;requested_at&quot; = 1 AND &quot;members&quot;.&quot;user_id&quot; IS NULL AND (&quot;namespaces&quot;.&quot;type&quot; = 1) UNION (SELECT &quot;namespaces&quot;.&quot;id&quot;, JOIN &quot;group_group_links&quot; ON &quot;group_group_links&quot;.&quot;shared_group_id&quot; = 1 AND &quot;members&quot;.&quot;source_id&quot; = 1 AND &quot;members&quot;.&quot;source_type&quot; = 1 AND &quot;members&quot;.&quot;requested_at&quot; = 1 AND &quot;members&quot;.&quot;user_id&quot; IS NULL AND (&quot;namespaces&quot;.&quot;type&quot; = 1))</td>
<td>41,162</td>
<td>995,280.30 ms</td>
<td>89,469,506</td>
<td>504,881,421 hits</td>
<td>33,165 hits</td>
<td>954 blks</td>
<td>1 blks</td>
</tr>
</tbody>
</table>
## K002 Workload Type ("The First Word" Analysis)

### Observations

Data collected: 2021-01-11 13:40:41 +0000 UTC
Current database: gitlabhq_production

**Master (10.220.16.106)**

Start: 2021-01-11T13:05:57.091968+00:00
End: 2021-01-11T13:39:07.728772+00:00
Period seconds: 1950.6398
Period age: 00:33:10.636804

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

<table>
<thead>
<tr>
<th>#</th>
<th>Workload type</th>
<th>Calls</th>
<th>▼ Total time</th>
<th>Rows</th>
<th>shared_bkgs_hit</th>
<th>shared_bkgs_read</th>
<th>shared_bkgs_dirtied</th>
<th>shared_bkgs_written</th>
<th>blk_read_time</th>
<th>blk_write_time</th>
<th>kcsac</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>select</td>
<td>41,827,896</td>
<td>5,019,032.35 ms</td>
<td>252,120 ms</td>
<td>668,048</td>
<td>604,199,044</td>
<td>6,911,927</td>
<td>95,009</td>
<td>931,653</td>
<td>1,000 ms</td>
<td>268,892 ms</td>
</tr>
<tr>
<td>2</td>
<td>with</td>
<td>752,893</td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>1,000,000</td>
</tr>
<tr>
<td>3</td>
<td>update</td>
<td>999,462</td>
<td>755,405.99 ms</td>
<td>379,405 ms</td>
<td>471,537</td>
<td>489,497</td>
<td>48,672</td>
<td>1,211,149</td>
<td>608,42</td>
<td>97,82</td>
<td>65,42</td>
</tr>
<tr>
<td>4</td>
<td>insert</td>
<td>837,581</td>
<td>402,76</td>
<td>379,405 ms</td>
<td>471,537</td>
<td>489,497</td>
<td>48,672</td>
<td>1,211,149</td>
<td>608,42</td>
<td>97,82</td>
<td>65,42</td>
</tr>
</tbody>
</table>

**Replica servers:**

Replica (10.220.16.101)

Start: 2021-01-11T13:05:57.091968+00:00
End: 2021-01-11T13:39:07.728772+00:00
Period seconds: 1819.1504
Period age: 00:30:19.180435

<table>
<thead>
<tr>
<th>#</th>
<th>Workload type</th>
<th>Calls</th>
<th>▼ Total time</th>
<th>Rows</th>
<th>shared_bkgs_hit</th>
<th>shared_bkgs_read</th>
<th>shared_bkgs_dirtied</th>
<th>shared_bkgs_written</th>
<th>blk_read_time</th>
<th>blk_write_time</th>
<th>kcsac</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>select</td>
<td>20,300,433</td>
<td>20,300,433</td>
<td>20,300,433</td>
<td>20,300,433</td>
<td>20,300,433</td>
<td>20,300,433</td>
<td>20,300,433</td>
<td>20,300,433</td>
<td>20,300,433</td>
<td>20,300,433</td>
</tr>
</tbody>
</table>
# K001 Globally Aggregated Query Metrics

## Observations

Data collected: 2021-01-11 13:40:41 +0000 UTC  
Current database: gilabhq_production

### Master (10.220.16.106)

Start: 2021-01-11T13:05:57.091988+00:00  
End: 2021-01-11T13:07:26.772284+00:00  
Period seconds: 1990.6388  
Period age: 00:33:10.639804

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

<table>
<thead>
<tr>
<th>Calls</th>
<th>Total time</th>
<th>Rows</th>
<th>shared_bikes_hit</th>
<th>shared_bikes_read</th>
<th>shared_bikes_dirtied</th>
<th>shared_bikes_written</th>
<th>blk_read_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>44,458,526</td>
<td>7,350,264.51 ms</td>
<td>161,387,670</td>
<td>1,186,643,394 bikes</td>
<td>8,726,150 bikes</td>
<td>2,496,148 bikes</td>
<td>4,740 bikes</td>
<td>1,875,765.69 ms</td>
</tr>
<tr>
<td>22.34K/sec</td>
<td>3692.41 ms/sec</td>
<td>81.69K/sec</td>
<td>3.15sec/call</td>
<td>2.69sec/call</td>
<td>0.00 sec/call</td>
<td>0.00 sec/call</td>
<td>0.00 sec/call</td>
</tr>
<tr>
<td>1.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

### Replica servers:

#### Replica (10.220.16.102)

Start: 2021-01-11T13:06:51.049781+00:00  
End: 2021-01-11T13:07:29.619440+00:00  
Period seconds: 1819.18944  
Period age: 00:30:19.185435

<table>
<thead>
<tr>
<th>Calls</th>
<th>Total time</th>
<th>Rows</th>
<th>shared_bikes_hit</th>
<th>shared_bikes_read</th>
<th>shared_bikes_dirtied</th>
<th>shared_bikes_written</th>
<th>blk_read_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>21,516,698</td>
<td>9,395,805.91 ms</td>
<td>31,311,433</td>
<td>4,887,890,546 bikes</td>
<td>12,921,880 bikes</td>
<td>0 bikes</td>
<td>402,350 bikes</td>
<td>866,082.39 ms</td>
</tr>
<tr>
<td>11.83K/sec</td>
<td>5163.77 ms/sec</td>
<td>17.22K/sec</td>
<td>1.14sec/call</td>
<td>1.00sec/call</td>
<td>0.00 sec/call</td>
<td>0.00 sec/call</td>
<td>0.00 sec/call</td>
</tr>
<tr>
<td>1.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

#### Replica (10.220.16.102)

Start: 2021-01-11T13:05:52.314892+00:00  
End: 2021-01-11T13:06:30.051124+00:00  
Period seconds: 1846.7463  
Period age: 00:30:46.7463

<table>
<thead>
<tr>
<th>Calls</th>
<th>Total time</th>
<th>Rows</th>
<th>shared_bikes_hit</th>
<th>shared_bikes_read</th>
<th>shared_bikes_dirtied</th>
<th>shared_bikes_written</th>
<th>blk_read_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>23,305,903</td>
<td>9,934,511.91 ms</td>
<td>32,988,938</td>
<td>13,969,380 bikes</td>
<td>5,015,880,782 bikes</td>
<td>0 bikes</td>
<td>514,997 bikes</td>
<td>758,241.41 ms</td>
</tr>
<tr>
<td>12.35K/sec</td>
<td>8379.46 ms/sec</td>
<td>17.87K/sec</td>
<td>2.72sec/call</td>
<td>7.57sec/call</td>
<td>0.00 sec/call</td>
<td>0.00 sec/call</td>
<td>0.00 sec/call</td>
</tr>
<tr>
<td>1.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
</tbody>
</table>
- boost development of fast-growing PostgreSQL-based projects using 
  *thin cloning and high level of automation*
Non-production environment weaknesses are reasons of multiple development problems

Development bottlenecks
(with standard staging DB)

- Bugs: difficult to reproduce, easy to miss
- Not 100% of changes are well-verified
- SQL optimization is hard
- Each non-prod big DB costs a lot
- Non-prod DB refresh takes hours, days, weeks

Frictionless development
(with Database Lab)

- Bugs: easy to reproduce, and fix early
- 100% of changes are well-verified
- SQL optimization can be done by anyone
- Non-prod DB refresh takes seconds
- Extra non-prod DBs doesn’t cost a penny
Database experiments – traditional approach
Database experiments on thin clones

Production
Thin clones – copy-on-write

- Shared data blocks
- Extra blocks for changes

- Thick copy of production (any size)
- Thin clone (size starts from 1 MB, depends on changes)
Database experiments on thin clones – yes and no

Yes
- Check execution plan – Joe bot
  - EXPLAIN w/o execution
  - EXPLAIN (ANALYZE, BUFFERS)
    - (timing is different; structure and buffer numbers – the same)
- Check DDL
  - index ideas (Joe bot)
  - auto-check DB migrations
- Heavy, long queries: analytics, dump/restore
  - No penalties! (think hot_standby_feedback, locks, CPU)

No
- Load testing
- Regular HA/DR goals
  - backups
    - (but useful to check WAL stream, recover records by mistake)
  - hot standby
    - (but useful to offload very long-running SELECTs)
Database Lab – Open-core model

**Database Lab Engine**

Open-source (AGPLv3)

- Thin cloning
- Automated provisioning and data refresh
- Data transformation, anonymization
- Supports managed Postgres (AWS RDS, etc.)

**Platform**

SaaS (pricing model: $ per TiB)

- Web console (GUI)
- Access control, audit
- History, visualization
- Support

[https://gitlab.com/postgres-ai/database-lab](https://gitlab.com/postgres-ai/database-lab)

[https://postgres.ai/](https://postgres.ai/)

– follow the links and start using it for your databases
SQL optimization using Database Lab and Joe bot
Automated checks of database migrations (DDL) using full-size thin clones provided by Database Lab

Before Database Lab:

- Developers test DDL on tiny databases, using only synthetic data, not seeing real behavior
- Before each release, DDL is tested on staging – a reduced/old/modified data set (~5-10% of real size)
- Manual code review. Very rarely the change is tested on a production clone

Issues with deploying DB migrations were not uncommon

An example:
https://gitlab.com/gitlab-com/gl-infra/production/-/issues/2802
Automated checks of database migrations (DDL) using full-size thin clones provided by Database Lab

With Database Lab:

- Separate project
  - security: limited access, firewall
  - isolation: reduced codebase and no extra components
  - connected to DLE API, able to use `dblab clone`

- On any CI build in the main project ("gitlab") has DDL, then:
  - a CI build in this special project is triggered
  - DDL is auto-verified on a fresh clone (lag <6h) provided by DLE
  - detailed artifacts are available to the Database Team and Infrastructure
    - Output
    - `pg_stat_***`
    - production timing estimates
  - summary is automatically posted as an MR comment
**Database Lab “Observed sessions”**

**Summary**

<table>
<thead>
<tr>
<th>Status</th>
<th>X Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session</td>
<td>#34</td>
</tr>
<tr>
<td>Project</td>
<td>demo</td>
</tr>
<tr>
<td>DLE instance</td>
<td>#5</td>
</tr>
<tr>
<td>Duration</td>
<td>3m. 5s</td>
</tr>
<tr>
<td>Created</td>
<td>2 months ago</td>
</tr>
<tr>
<td>Branch</td>
<td>transform</td>
</tr>
<tr>
<td>Commit</td>
<td>3641504e3832ef37fa768a6f0b0b2e25501</td>
</tr>
<tr>
<td>Triggered by</td>
<td>Anotly</td>
</tr>
<tr>
<td>PR/URI</td>
<td><a href="https://github.com/postgres-all-examples/mmerge_request/2">https://github.com/postgres-all-examples/mmerge_request/2</a></td>
</tr>
</tbody>
</table>

**Checklist**

- Dangerous locks is not observed during the session
  - 13 intervals with locks of 10 allowed
- Session duration is within allowed interval
  - (2m, 1s of 1s allowed)

**Observed intervals and details**

<table>
<thead>
<tr>
<th>Duration</th>
<th>Started at</th>
<th>16:43:19 14.09.20 19 UTC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2020-11-03</td>
<td>14:59:20 UTC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Duration</th>
<th>Started at</th>
<th>16:43:19 14.09.20 19 UTC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2020-11-03</td>
<td>14:59:20 UTC</td>
</tr>
</tbody>
</table>
Demo time
Summary

- PostgreSQL database health check is automated
- 150+ engineers now do these activities:
  - get EXPLAIN for any query for production database (not being blocked and not blocking others)
  - get insights of how DDL behaves before submitting MR for DB migration review
  - learn SQL by example (using full-size databases!)
- Database team has
  - Way to conduct various database experiments without need to provision new nodes and/or wait for long data refresh
  - DB migration reviews are pre-checked automatically in 100% of cases, with prediction of what would happen during production deployment

*Control over SQL performance and scalability is improved*

*Downtime and performance degradation risks are eliminated*
Contribute

Everyone can contribute
Thank you. Please feel free to follow up!

Jose Finotto
jfinotto@gitlab.com
LinkedIn: linkedin.com/in/jose-c-bb4a2178/

Nikolay Samokhvalov
nik@postgres.ai
Twitter: @samokhvalov
LinkedIn: linkedin.com/in/samokhvalov/