GitLab.com database incident

Feb 1, 2017 · 5 min read · Leave a comment

GitLab

Update: please see our postmortem for this incident

Yesterday we had a serious incident with one of our databases. We lost six hours of database data (issues, merge requests, users, comments, snippets, etc.) for GitLab.com. Git/wiki repositories and self-managed installations were not affected. Losing production data is unacceptable and in a few days we'll publish a post on why this happened and a list of measures we will implement to prevent it happening again.

Update 6:14pm UTC: GitLab.com is back online

As of time of writing, we're restoring data from a six-hour-old backup of our database. This means that any data between 5:20pm UTC and 11:25pm UTC from the database (projects, issues, merge requests, users, comments, snippets, etc.) is lost by the time GitLab.com is live again.

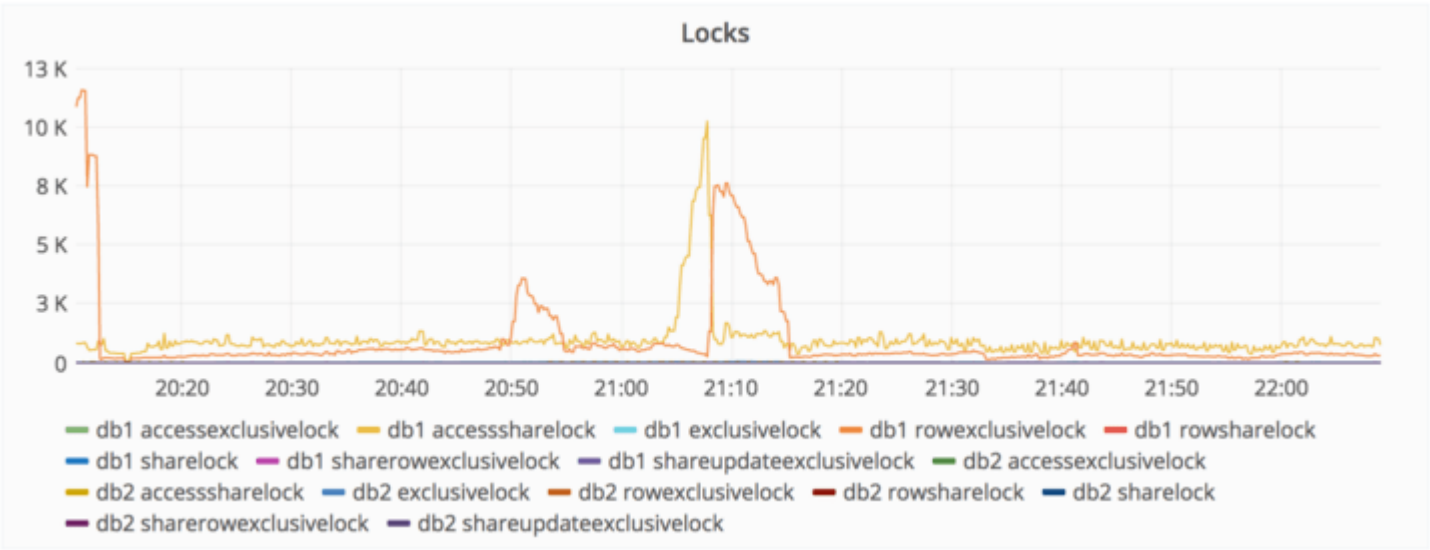
Git data (repositories and wikis) and self-managed instances of GitLab are not affected.

Read below for a brief summary of the events. You're also welcome to view our active postmortem doc.

First incident

At 2017/01/31 6pm UTC, we detected that spammers were hammering the database by creating snippets, making it unstable. We then started troubleshooting to understand what the problem was and how to fight it.

At 2017/01/31 9pm UTC, this escalated, causing a lockup on writes on the database, which caused some downtime.

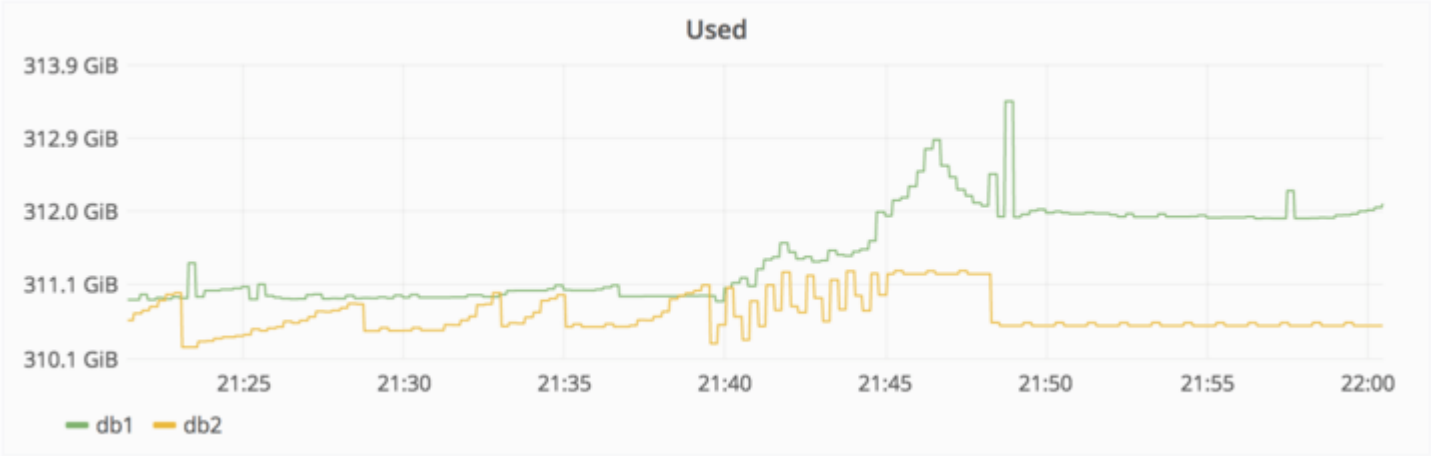


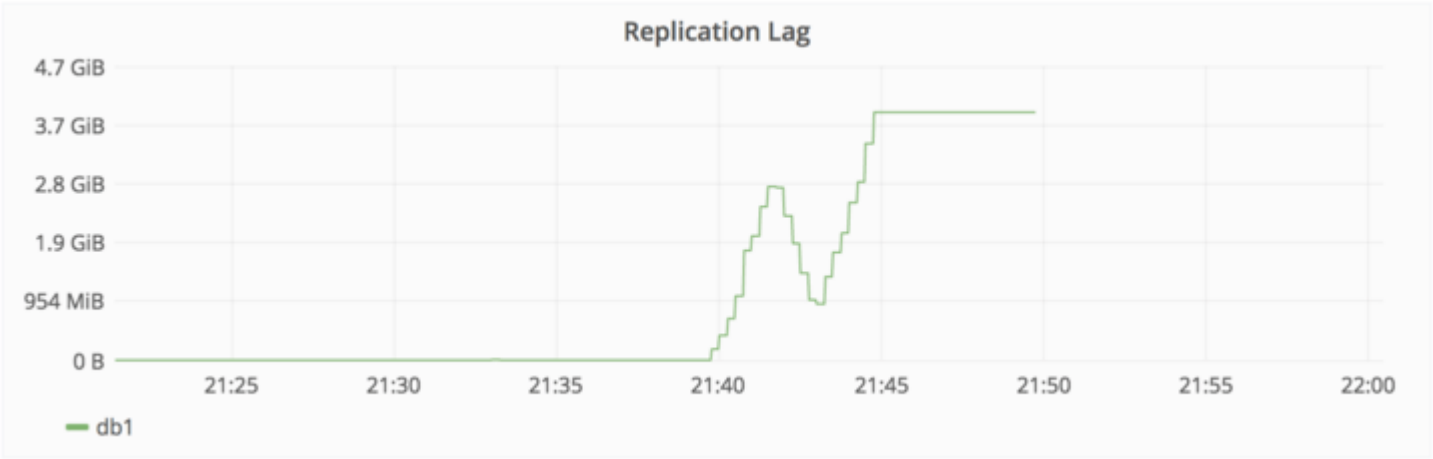
Actions taken

- We blocked the spammers based on IP address
- We removed a user for using a repository as some form of CDN, resulting in 47 000 IPs signing in using the same account (causing high DB load)
- We removed users for spamming (by creating snippets)

Second incident

At 2017/01/31 10pm UTC, we got paged because DB Replication lagged too far behind, effectively stopping. This happened because there was a spike in writes that were not processed ontime by the secondary database.





Actions taken

- Attempt to fix `db2`, it's lagging behind by about 4 GB at this point
- `db2.cluster` refuses to replicate, `/var/opt/gitlab/postgresql/data` is wiped to ensure a clean replication
- `db2.cluster` refuses to connect to `db1`, complaining about `max_wal_senders` being too low. This setting is used to limit the number of `WAL (= replication)` clients
- *Team-member-1* adjusts `max_wal_senders` to `32` on `db1`, restarts PostgreSQL

- PostgreSQL complains about too many semaphores being open, refusing to start
- *Team-member-1* adjusts `max_connections` to `2000` from `8000`, PostgreSQL starts again (despite `8000` having been used for almost a year)
- `db2.cluster` still refuses to replicate, though it no longer complains about connections; instead it just hangs there not doing anything
- At this point frustration begins to kick in. Earlier this night *team-member-1* explicitly mentioned he was going to sign off as it was getting late (23:00 or so local time), but didn't due to the replication problems popping up all of a sudden.

Third incident

At 2017/01/31 11pm-ish UTC, *team-member-1* thinks that perhaps `pg_basebackup` is refusing to work due to the PostgreSQL data directory being present (despite being empty), decides to remove the directory. After a second or two he notices he ran it on `db1.cluster.gitlab.com`, instead of `db2.cluster.gitlab.com`.

At 2017/01/31 11:27pm UTC, *team-member-1* - terminates the removal, but it's too late. Of around 300 GB only about 4.5 GB is left.

We had to bring GitLab.com down and shared this information on Twitter:

Problems encountered

- LVM snapshots are by default only taken once every 24 hours. *Team-member-1* happened to run one manually about six hours prior to the outage because he was working in load balancing for the database.
- Regular backups seem to also only be taken once per 24 hours, though *team-member-1* has not yet been able to figure out where they are stored. According to *team-member-2* these don't appear to be working, producing files only a few bytes in size.
- *Team-member-3*: It looks like `pg_dump` may be failing because PostgreSQL 9.2 binaries are being run instead of 9.6 binaries. This happens because omnibus only uses Pg 9.6 if

data/PG_VERSION is set to 9.6, but on workers this file does not exist. As a result it defaults to 9.2, failing silently. No SQL dumps were made as a result. Fog gem may have cleaned out older backups.

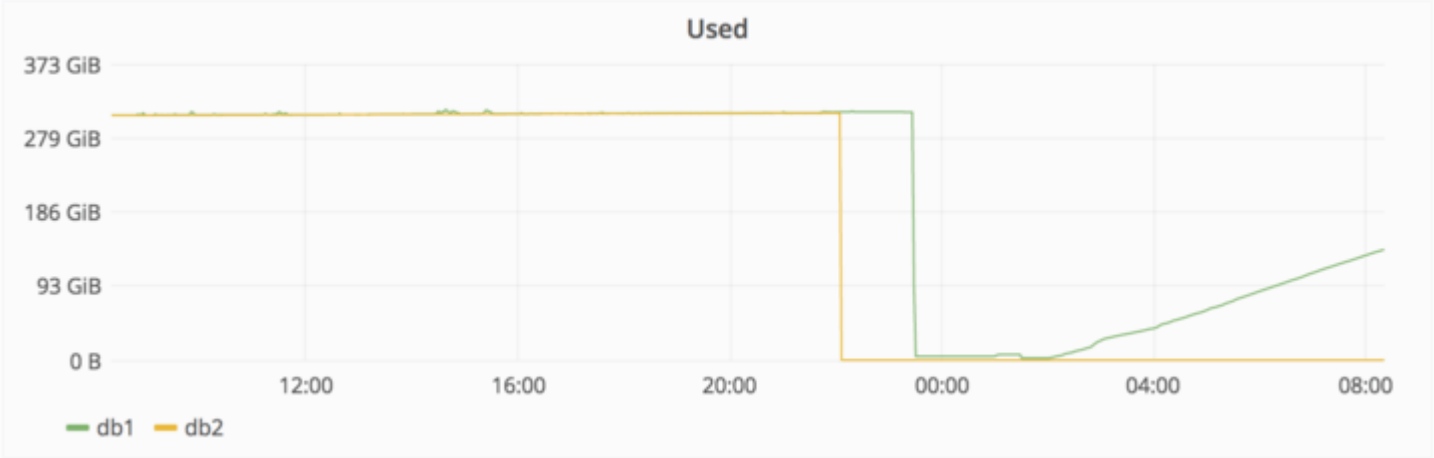
- Disk snapshots in Azure are enabled for the NFS server, but not for the DB servers.
- The synchronisation process removes webhooks once it has synchronised data to staging. Unless we can pull these from a regular backup from the past 24 hours they will be lost
- The replication procedure is super fragile, prone to error, relies on a handful of random shell scripts, and is badly documented
- Our backups to S3 apparently don't work either: the bucket is empty
- So in other words, out of five backup/replication techniques deployed none are working reliably or set up in the first place. We ended up restoring a six-hour-old backup.
- pg_basebackup will silently wait for a master to initiate the replication progress, according to another production engineer this can take up to 10 minutes. This can lead to one thinking the process is stuck somehow. Running the process using "strace" provided no useful information about what might be going on.

Recovery

We're working on recovering right now by using a backup of the database from a staging database.

- 2017/02/01 00:36 - Backup `db1.staging.gitlab.com` data
- 2017/02/01 00:55 - Mount `db1.staging.gitlab.com` on `db1.cluster.gitlab.com`
- Copy data from staging `/var/opt/gitlab/postgresql/data/` to production `/var/opt/gitlab/postgresql/data/`
- 2017/02/01 01:05 - `nfs-share01` server commandeered as temp storage place in `/var/opt/gitlab/db-meltdown`
- 2017/02/01 01:18 - Copy of remaining production data, including `pg_xlog` tar'ed up as `20170131-db-meltodwn-backup.tar.gz`

Below a graph showing the time of deletion and subsequent copying in of data.



Also, we'd like to thank everyone for the amazing support we've received on Twitter and elsewhere through #hugops