the Availability Digest

www.availabilitydigest.com
@availabilitydig

GitLab Suffers Massive Backup Failure Due to a Fat Finger

April 2017

GitLab offers a web-based Git repository manager with wiki and issue-tracking features. (A *wiki* is a website that allows collaborative editing of its content and structure by its users.) *Git* is an open-source program code repository for tracking changes in computer files. It is used primarily for software development, coordinating work on the code files among many developers for collaboration. Git was developed by Linus Torvalds in 2005 for development of the Linux kernel.

GitLab offers several online and on-premises applications that allow users to code, test, and deploy software projects. It has over 1,400 contributors from companies such as Sony, IBM, NAS, and CERN.

GitLab Loses 300 GB of Data

On January 31, 2017, GitLab suffered a massive data loss when a technician accidentally deleted 300 GB of data from a production database. This error was compounded by the inability to restore the database from the company's backups. GitLab was down for about 36 hours before it was able to recover its database.

The Accidental Deletion of Production Data

An increase in load on a GitLab PostgreSQL database caused the replication of data to its standby system to become compromised. The increase in load was caused by a background job that was trying to remove a GitLab employee and all of his data after he was flagged for abuse by a troll (later determined to be an error by the troll). The database load was further aggravated by a spam attack that was in progress at the same time.

The increased load caused severe database server performance problems, leading to multiple resource failures. Backups were not generated properly, and others could not be restored. Replication of data to the secondary database began to lag far behind, and primary segments were being removed before they could be replicated. Consequently, data replication eventually failed. This required that the secondary be resynchronized. Several automated repair attempts failed, leading a system administrator to take manual action.

In order to restore data replication, the database directory of the secondary database had to be empty. Clearing the secondary database directory was not an automated process. It had to be done manually. Furthermore, the process was not documented properly.

The system administrator undertook the task of clearing the secondary database directory. However, instead of wiping the directory on the secondary server, the system administrator's command erroneously referred to the primary server. About 300 GB of live production data was consequently deleted erroneously by the GitLab sysadmin before the error was discovered and the deletion command terminated.

By this time, about eighteen hours had passed since the GitLab database became inoperable.

The Failed Restoration Process

The restoration process took another eighteen hours before GitLab was back up and running. The long restoration process was due to having to restore the database using a copy of a staging database hosted on a slower Azure VM in a different region. The staging database was needed because GitLab had not enabled Azure disk snapshots for its database servers. The staging database was six hours old.

The normal recovery procedure would have been to fail over to the secondary database server. However, the secondary database had been deleted as part of the attempt to restore database replication.

The standard backup procedure failed because it was using PostgreSQL 9.2 whereas GitLab was running PostgreSQL 9.6. Furthermore, the backup failure was compounded by the fact that it failed silently without notification. This was caused by notification emails being rejected because they were not properly signed by DMARC (Domain-Based Message Authentication, Reporting, and Conformance).

Azure snapshots could also have been used for recovery. GitLab did not enable these snapshots because it assumed that its other backup procedures were sufficient. Besides, restoring Azure snapshots could take days.

Why weren't the backup and restore processes tested on a regular basis? Because there was no ownership. Nobody was responsible for testing these procedures.

Lessons Learned

GitLab communicated quickly and openly with its users and customers to explain what had happened, how the outage was being fixed, and how it is being prevented in the future. In a blog, it explained in detail what had happened and what it is doing to prevent a reoccurrence of the problem.[1]

5,000 projects and 700 new user accounts were affected by the outage. However, no client data was lost. The only losses were for comments and bug report issues. No repositories or wikis were lost.

The outage affected only GitLab.com. Customers using GitLab's platform on-premise were not affected. GitLab is improving its procedures. An ideal environment is one in which you *can* make mistakes but easily and quickly recover from them

GitLab stated that none of its five deployed backup/replication technologies were working reliably or were set up in the first place

As GitLab's experience shows, backups don't matter if you can't restore your database from them. IT admins need to make a solid habit of testing their backup and recovery capabilities. Having a fail-safe backup solution for your critical systems is of paramount importance.

Statistics show that human error is the main cause of data loss. That was certainly true in GitLab's case. This is a further example of a caution we have expressed many times in the *Availability Digest*. Humans need redundancy too. If a critical command is to be entered that could wreak havoc with your system, it should be entered by two people – one to type the command and the other to check it before it is executed. If GitLab had followed this simple procedure, its outage would not have happened.

[1] Postmortem of database outage of January 31, *GitLab*; February 10, 2017. https://about.gitlab.com/2017/02/10/postmortem-of-database-outage-of-january-31/

Acknowledgements

Information for this article was taken from the following sources:

GitLab suffers major backup failure after data deletion incident, *Tech Crunch*; February 1, 2017.
Postmortem of database outage of January 31, *GitLab*; February 10, 2017.
Lessons Learned from GitLab's Massive Backup Failure, *Storagecraft Recovery Zone*; February 15, 2017.
What to Learn from GitLab's Recent Data Backup Disaster, *Windows IT Pro*; February 24, 2017.
GitLab, *Wikipedia*.