Closed (moved) Opened 2 years ago by **John Jarvis**

GitLab.com outage 2017-09-14

Summary

Gitlab.com became slow and sometimes unavailable from 05:40 to 08:25 UTC on 2017-09-14. Git operations over ssh were not affected.

Preliminary Root Cause Analysis

We deployed 10.0RC2 during the night since that time we saw a huge increase in Gitaly GRPC method calls to the method FindAllBranchNames. Apparently there was a change in the code that moved the check for branch counts from being executed through the model object (which memoizes the result) out to a separate object. This translated in the rails application hammering Gitaly with a lot of calls, and spending the time waiting on Gitaly.

Timeline

- 2:11 - 10.0RC2 deploy completed
- 6:15 - The oncall got a page because the master branch, the page auto resolves immediately
- 6:19 - Someone complains about slowness in the gitlab.
- 6:25 - We see high load in NFS-08 - all the other hosts are idling
- 6:46 - We check that all the backend services are idling away, not resource constrained of any kind.
- 6:48 - We check that NFS timeouts are not happening.
- 6:51 - We stop elasticsearch indexing just to reduce pressure on NFS servers
- 6:58 - We check for IO constraints in NFS-08 - it's not constrained.
- 7:02 - Tweet sent "we are investigating slow performance after the 10.0 upgrade"
- 7:07 - We check that prometheus metrics are not enabled in the admin panel.
- 7:08 - We test using canary only to discover that it's returning errors 500 from yesterday deploy. (how did this happen?)
- 7:09 - We open a war room and all of prod eng jumps in.
- 7:15 - We restart canary workers to see if it recovers, it does not.
- 7:16 - Quick discussion leads to think that rollback is the only sane option as all the backend/infrastructure is doing fine, but the front end is struggling.
- 7:17 - A production engineer walks into both #development and #releases and issues a " `@channel` we need a developer in this call Ö'r ỹ Zoom link to discuss if we can rollback the version"
- 7:19 - Developers join the call.
- 7:21 - There's a discussion around rolling back the version, given that the deployed version is 10.0 there are many breaking changes, we continue discussing if the application will support a model so changed. Down the line we decide test deploying 9.5.4 in staging without running any migration to test viability of this change, this proves to not be possible as multiple errors start raising. This prevents us from testing deploying the previous version in canary as a test. It will simply not work.
- 7:36 - Noticed FindAllBranchNames number of calls is way up
- 7:40 - Investigation around controllers to see which one is being impacted, we reach Repository#show, in which we notice a huge increase in #method_missing Wall times.
- 7:49 - feature `gitaly_branch_names` turned off. The whole site goes down.

Discussion around this behavior: we realize that gitaly branch names is taking all the load, by removing the method we moved all the load to the front end and took the whole site down. This means that Gitaly is coping with the load but not rails because of the use of rugged.

- 7:52 - feature `gitaly_branch_names` turned on again through a rails console, the API is timing out on Marvin.
- 8:00 - We run a couple of methods to measure how much time they get to execute (from https://gitlab.slack.com/archives/C101F3796/p1505376112000153 onwards)
- 8:05 - We narrow it down to the method branch_count (ish) not being memoized, which translates in a lot of calls being pushed to Gitaly, this in time is slowing down execution of the whole rails app.
- 8:17 - Apply a hot patch https://gitlab.slack.com/archives/C101F3796/p1505377017000146
- 8:27 - With the hotpatch applied everything starts to recover https://gitlab.slack.com/archives/C101F3796/p1505377637000351
- 8:31 - We tweet again (https://gitlab.slack.com/archives/C101F3796/p1505377888000518) -> https://twitter.com/gitlabstatus/status/908246817678209024 The outage is over.

Incident Analysis

- How was the incident detected?
- Is there anything that could have been done to improve the time to detection?
- How was the root cause discovered?
- Was this incident triggered by a change?
- Was there an existing issue that would have either prevented this incident or reduced the impact?

Root Cause Analysis

GitLab.com started responding slowly and then experienced an outage.

1. ť̓ Dăǿ Requests were making hundreds of calls to `Gitlab::Git::Repository#has_visible_content?` on each invocation. Each of these calls was making a Gitaly once per call, resulting in hundreds of round-trips to Gitaly on each request. After the Gitaly feature was turned off, these calls started using Rugged, which created a surge in NFS traffic which took the site down.

2. ť̓ Dă were hundreds of calls being made to `Gitlab::Git::Repository#has_visible_content?` ? These calls were originally made to `app/models/Repository#has_visible_content?` . However, this method was removed in https://gitlab.com/gitlab-org/gitlab-ce/merge_requests/13773 as part of a clean-up in preparation for some new Gitaly endpoints.

3. ť̓ Dă, if the method was still being used, did it not break the build when it was removed? Stay with me here: `app/models/Repository` uses a `method_missing` method to intercept all missing methods and delegate them to corresponding methods on `Gitlab::Git::Repository` . So `app/models/Repository#has_visible_content?` automagically delegated to `Gitlab::Git::Repository#has_visible_content?` after its removal. While both implementations of this method were identical: `branch_count > 0` , `app/models/Repository#has_visible_content?` version called `app/models/Repository#branch_count` while `Gitlab::Git::Repository#has_visible_content?` called `app/models/Repository#has_visible_content?` . The difference here is that `app/models/Repository#branch_count` is cached, while `Gitlab::Git` objects do not cache (by design). Since `Gitlab::Git::Repository#branch_count?` is not cached, this led to each invocation of `app/models/Repository#has_visible_content?` making a call down to the storage layer (that being either Gitaly or NFS)

4. ť̓ Dă was this not picked up by the developer and reviewers? The cognitive overhead of recognising and avoiding an issue like this in the codebase is incredibly high. There is an expectation if things work locally and the test suite passes, the code is good. In this case, the tests passed.

5. ť̓ Dă do we not test for situations like this? Extra work needs to be done to test for situations like this, and this work had not been done.

So ultimately I would put this down to code complexity, cognitive overload and failure of our test suite to pick up a performance degredation,.

Fortunately, these checks are now part of our codebase, in https://gitlab.com/gitlab-org/gitlab-ce/merge_requests/13996. From now on, any accidental changes of this sort will automatically be detected and cause the build to break, making developers and reviewers aware of the issue.

Additionally, the further lower the cognitive overhead involved in this part of the codebase, I think we should ửŕ ŎŶķĶǿŕ removing `method_missing` from `app/models/Repository` to get a better idea of what methods are actually being called onto this class.

What went well

- There was enough monitoring to identify the problem.
- Feature flags allowed to test an assumption in production.

What can be improved

- There was a 1h service degradation that happened while production was trying to troubleshoot the problem until the outage was escalated to development for troubleshooting help, we could formalize an escalation path for production troubleshooting with developers to reduce the MTTR.
- Improve performance degradation test suite (already being worked on https://gitlab.com/gitlab-org/gitlab-ce/merge_requests/13996)
- Code implicitness increases the cognitive overhead to identify this kind of problems. We should consider removing some of the implicitness.

Corrective actions

- https://gitlab.com/gitlab-org/gitlab-ce/merge_requests/13996 automate some performance degradation tests

Guidelines

- Blameless Postmortems Guideline
- 5 whys

Edited 2 years ago by Pablo Carranza [GitLab]

Linked issues 0

John Jarvis @jarv changed the description 2 years ago

Toon Claes 🌴 @toon changed the description 2 years ago

Jarka Košanová ⛱ @jarka · 2 years ago Developer

MR with the fix: https://gitlab.com/gitlab-org/gitlab-ce/merge_requests/14264 - this has to be included in the RC3

Edited by Jarka Košanová 2 years ago

Pablo Carranza [GitLab] @pcarranza-gitlab added app bug label 2 years ago

Hayk Galstyan 🤖 @HaykoKoryun · 2 years ago

@jarv are the following sections left blank on purpose?: **Incident Analysis**, **Root Cause Analysis**, **What went well**, **What can be improved**

Edited by Hayk Galstyan 2 years ago

Daniele Valeriani [GitLab] @omame-gitlab changed title from **GitLab.com outage outage 2017-09-14** to **GitLab.com outage 2017-09-14** 2 years ago

Daniele Valeriani [GitLab] @omame-gitlab · 2 years ago

@HaykoKoryun we're still working on this.

Pablo Carranza [GitLab] @pcarranza-gitlab · 2 years ago

@HaykoKoryun you can read the preliminary root cause analysis as "what happened" (https://gitlab.com/gitlab-com/infrastructure/issues/2767#preliminary-root-cause-analysis) - we are still digging in the "why did this happened" as @omame mentioned.

Pablo Carranza [GitLab] @pcarranza-gitlab mentioned in issue #2694 (closed) 2 years ago

Pablo Carranza [GitLab] @pcarranza-gitlab assigned to @andrewn and unassigned @jarv 2 years ago

Pablo Carranza [GitLab] @pcarranza-gitlab · 2 years ago

I'm assigning to @andrewn as he's assigned to dive deep in the post mortem.

Andrew Newdigate @andrewn · 2 years ago Maintainer

Thanks @pcarranza. I'll focus on this Monday.

Pablo Carranza [GitLab] @pcarranza-gitlab removed milestone 2 years ago

Pablo Carranza [GitLab] @pcarranza-gitlab changed milestone to %WoW ending 2017-09-19 2 years ago

Ghost User @ghost1 · 2 years ago

@andrewn any update?

Andrew Newdigate @andrewn · 2 years ago Maintainer

Root Cause Analysis

GitLab.com started responding slowly and then experienced an outage.

1. ť Ďăǒ Requests were making hundreds of calls to `Gitlab::Git::Repository#has_visible_content?` on each invocation. Each of these calls was making a Gitaly once per call, resulting in hundreds of round-trips to Gitaly on each request. After the Gitaly feature was turned off, these calls started using Rugged, which created a surge in NFS traffic which took the site down.
2. ť Ďă were hundreds of calls being made to `Gitlab::Git::Repository#has_visible_content?` ? These calls were originally made to `app/models/Repository#has_visible_content?` . However, this method was removed in https://gitlab.com/gitlab-org/gitlab-ce/merge_requests/13773 as part of a clean-up in preparation for some new Gitaly endpoints.
3. ť Ďă, if the method was still being used, did it not break the build when it was removed? Stay with me here: `app/models/Repository` uses a `method_missing` method to intercept all missing methods and delegate them to corresponding methods on `Gitlab::Git::Repository` . So `app/models/Repository#has_visible_content?` automagically delegated to `Gitlab::Git::Repository#has_visible_content?` after its removal. While both implementations of this method were identical: `branch_count > 0` , `app/models/Repository#has_visible_content?` version called `app/models/Repository#branch_count` while `Gitlab::Git::Repository#has_visible_content?` called `app/models/Repository#has_visible_content?` . The difference here is that `app/models/Repository#branch_count` is cached, while `Gitlab::Git` objects do not cache (by design). Since `Gitlab::Git::Repository#branch_count?` is not cached, this led to each invocation of `app/models/Repository#has_visible_content?` making a call down to the storage layer (that being either Gitaly or NFS)
4. ť Ďă was this not picked up by the developer and reviewers? The cognitive overhead of recognising and avoiding an issue like this in the codebase is incredibly high. There is an expectation if things work locally and the test suite passes, the code is good. In this case, the tests passed.
5. ť Ďă do we not test for situations like this? Extra work needs to be done to test for situations like this, and this work had not been done.

So ultimately I would put this down to code complexity, cognitive overload and failure of our test suite to pick up a performance degredation,.

Fortunately, these checks are now part of our codebase, in https://gitlab.com/gitlab-org/gitlab-ce/merge_requests/13996. From now on, any accidental changes of this sort will automatically be detected and cause the build to break, making developers and reviewers aware of the issue.

Additionally, the further lower the cognitive overhead involved in this part of the codebase, I think we should ùŕ ŐŸąĶǗ removing `method_missing` from `app/models/Repository` to get a better idea of what methods are actually being called onto this class.

Pablo Carranza [GitLab] @pcarranza-gitlab · 2 years ago

+1 to remove the implicit handle of methods, they should just fail in testing.

Pablo Carranza [GitLab] @pcarranza-gitlab changed the description 2 years ago

Pablo Carranza [GitLab] @pcarranza-gitlab · 2 years ago

I updated the description with the RCA, thanks @andrewn

@sitschner I'm going to close this issue as there is nothing else to do from our side. We should review the further actions as I think that there are a couple more actions to do outside of what was already done.

Pablo Carranza [GitLab] @pcarranza-gitlab closed 2 years ago

→ **Andrew Newdigate** @andrewn moved to production#243 (closed) 1 year ago

Please register or sign in to reply