Closed Opened 1 year ago by John Jarvis

Postmortem for the loss of public IP for customers with custom pages domains

Summary

While at the summit, the production team got together to start the process of deleting parts of our old Azure infrastructure that were no longer needed. We had document reviewing the list of things to remove here: https://docs.google.com/document/d/1yzVNKmW7tRaDgsYgsmMAqfg-9lAGZa4ByIddmNluZ6U/edit#heading=h.sbbjjgkwvh1k

This document was to be reviewed by the team for what all we needed to remove. Removing from the azure console was chosen since Terraform no longer executed cleanly against the Azure environment and Azure + Terraform has proven to be difficult in the past.

When the ExternalLBProd resource group was deleted, we quickly noticed that the GitLab pages load balancer and proxy were included in the resource group. For people that had GitLab pages still pointing at the old IP address in Azure, this removal then broke their pages along with docs.gitlab.com. We were able to change the DNS record for docs, but needed to make a blog post and tweet to inform users of the now needed DNS change: https://about.gitlab.com/2018/08/28/gitlab-pages-update/

We filed a support case with Microsoft to see if we could reclaim the public IP that we had used and were informed 2.5 hours later that it would not be possible.

Incident issue - production#436 (closed)

Impact & Metrics

- What was the impact of the incident? (i.e. service outage, sub-service brown-out, exposure of sensitive data, ...)

Customers who had custom dns for their GitLab pages saw connection timeouts for their pages sites. This was a loss of availability, but no dataloss, for the subset of customers affected.

- Who was impacted by this incident? (i.e. external customers, internal customers, specific teams, ...)

2107 customers were impacted by this incident. This list of customers are those who had custom A records for their GitLab pages site. This required them to update their DNS A record in advance of the sunset grace period. A full list of customers can be viewed in this spreadsheet (internal only) production#436 (closed)

- How did the incident impact customers? (i.e. preventing them from doing X, incorrect display of Y, ...)

Customers were notified by email that they needed to update their dns records in advance of the grade period previously given for updating DNS to the new endpoint in GCP.

- How many customers were affected?

2107 customers were affected.

Detection & Response

- How was the incident detected? - The incident was detected immediately when the resource was released.
- Did alarming work as expected? - Yes.
- How long did it take from the start of the incident to its detection? - Immediately

Root Cause Analysis

1. Why did pages stop working for customers with custom domains pointed to Azure? - The IP address reserved for the load balancer was released

2. Why was the azure GitLab pages IP address released? - The reserved IP address was removed as part of maintenance on the Azure infrastructure?

3. Why were we unable to recover the IP address? - Once a reserved IP is released it goes into a shared pool and is impossible to recover.

4. Why was this maintenance performed? - In order to reduce alert noise, decrease the number of VMs to monitor, reduce cost.
5. Why was the maintenance not performed in staging first? - The staging environment was previously removed and in Azure, there was not parity between those two environments.
6. Why didn't this maintenance work involve a dry run? - While the maintenance was written down and reviewed the dry run was not executable because the changes were made directly in the Azure console.
7. Why were was the maintenance not done using automation? - The terraform environment for both production and staging in Azure had drifted in such a way it could not be used in this way to clean up the resources.

Additional Questions

- What can we do to get tools with sanity checks to do as many changes for us as possible?

Commit to using the tools and squashing technical debt when it arises. Technical debt killed the Azure terraform structure because it was allowed to be viewed as "why bother making it work, we're just going to throw it away" for close to ten months.

- What can we do to be more confident in the review of potential changes - for deletes, can we trust async only?

Async only wasn't the issue - we all sat in a room synchronously and agreed before execution. However, going forward for deletes - for the time being, I recommend a spin-down and hold cool down period, and then a delete. Although improvements have been made in the new GCP environment how can we be sure this specific incident will never happen again?

We can implement lifecycle rules that prevent an IP address resource from being removed as a failsafe, even if the automated tooling tries to remove it.

What went well

- Identified quickly
- The entire team was readily available to come up with ideas and swarm with potential ways to fix the issue at hand.

What can be improved

- We had abandoned using Terraform against Azure due to configuration atrophy and code rot, making it impossible to perform a terraform plan against the Azure environment.
- Because the entire environment in staging was torn down in Azure we did not have a way to validate the changes against staging first, before production.
- We failed to check at the object level for the important items that had been identified to stay and only reviewed larger "resource group" aggregators.
- Because the tooling was not in place it was not possible to do a "dry run" of the deletion, instead we were deleting resources at the group level.

Corrective actions

- **The production and staging environment should never drift, these need to be kept in lockstep so that changes can be first validated in staging, and then production. In Azure we had a staging environment that was never quite the same as production and not monitored.**
  - COMPLETED: Chef - An improvement that was made in GCP is a check to ensure that there is no configuration drift in our chef roles. https://dev.gitlab.org/cookbooks/chef-repo/pipelines In some cases drift is expected where we validate a change in the staging environment in production but this should only the case for a short period of time.
  - COMPLETED: Terraform - In GCP there is a common configuration for both staging and production. The definition of resources are shared https://gitlab.com/gitlab-com/gitlab-com-infrastructure/tree/master/shared/gstg-gprd variables are set to control capacity but the topology of production and staging moves in lockstep.
  - COMPLETED: Alerting - Ensure that all alerts are the same for production and staging, there should be no alerts defined that are limited in scope to the production environment unless there is a good reason to. gitlab-com/runbooks!733 (merged)
- **There should be both custom configured A record and non-custom endpoints tested with external probes and alerting, for both staging and production. By the time an ip is released in production it will be too late, this alert should alert the oncall in staging.**
  - SCHEDULED, due Sept 22nd: #5058 (closed)

- **All infrastructure creation and deletion should be controlled with terraform, nothing should be removed manually through the administration console.**
  - Currently all infrastructure in GCP is controlled through terraform and chef, nothing is done manually through the console.
    - SCHEDULED, due Sept 22nd. Using CICD for terraform - #4872 (closed)
    - SCHEDULED, due Sept 22nd.. Using CICD for chef - #4856 (closed)
- **We should have extra safeguards in place so that if automation accidentally attempts to release an IP that cannot be recovered, it will fail.**
  - COMPLETED - https://gitlab.com/gitlab-com/gitlab-com-infrastructure/merge_requests/533
- List issues that have been created as corrective actions from this incident.
- For each issue, include the following:
  - - Issue labeled as corrective action .
  - Include an estimated date of completion of the corrective action.
  - Incldue the named individual who owns the delivery of the corrective action.

Guidelines

- Blameless Postmortems Guideline
- 5 whys

Edited 1 year ago by John Jarvis

Linked issues 0

Related merge requests 1

Removes places where we were filtering for the gprd environment.
gitlab-com/runbooks!733

John Jarvis @jarv changed the description 1 year ago

John Jarvis @jarv mentioned in issue #4991 (closed) 1 year ago

Gerardo Lopez-Fernandez @glopezfernandez · 1 year ago Owner

One important take-away from this outage is that we need to better evaluate risk and treat some potentially dangerous changes (especially those that involve deletions, for instance) as a **maintenance proper**. This mindset would entail preflight checks (which might have led us to check for objects inside resource groups), a more appropriate time window to execute the maintenance (since availability is our primary driver, we might have concluded there was less of a reason to do this fast and we might have evaluated alternate approaches to meet the requirements outlined here), and careful development and review of the plan of attack (which might have provided more async time to think about potential issues).

John Jarvis @jarv · 1 year ago Owner

> One important take-away from this outage is that we need to better evaluate risk and treat some potentially dangerous changes

Agreed - we may want to classify this as anything that is deleted (through terraform automation) requires additional scrutiny. This is something we can pick up in a pipeline check for our terraform automation.

John Jarvis @jarv · 1 year ago Owner

the remaining corrective actions for this incident are the automation for chef and terraform which are currently underway. I will close this unless there are more corrective actions to add.

John Jarvis @jarv closed 1 year ago

Please register or sign in to reply