SRE

A Perfect DNS Storm

ADAM MARGHERIO
5 NOV 2018 · 4 MIN READ

***tl;dr** - a combination of `ndots`, the DNS resolver implementation in the musl library, and a flawed DNS resolution implementation in Netty caused repeated failures in DNS resolution for Kubernetes-deployed services.*

DNS resolution can be a tricky problem to diagnose when it happens inconsistently. Adding Kubernetes and container-based applications into the mix add an additional layer of networking complexity that can make failures even harder to troubleshoot. Recently, we had a recurring situation where DNS resolution was failing for Kubernetes-external services that took a full day to diagnose and remedy.

The resulting postmortem had quite a few items that I believe are worth calling out for Kubernetes-deployed and containerized applications. The timeline of issue resolution was extended by several factors such as my own lack of experience in SRE and operational areas and several red herrings (called out further along in the post) that derailed investigative efforts for a while.

The initial problem and a possible fix

Recently, services importing and using our Redis-dependent library reported service crashes on start-up to the ops members of our team. Looking at the resulting Spring Boot logs showed that the applications were failing to resolve a Redis hostname during bean initialization.

We started troubleshooting the reported problem and confirmed the initially reported issue. We found that the Azure Redis Cache being used was one for a previously-unknown instance. We decided to update the URI in the configuration repository to use a legitimate Redis Cache instance, pushed the changes to the configuration repository, and began restarting dependent services to pick up the configuration changes. The first services restarted displayed positive results, so we believed we had solved the problem.

DNS resolution failures continue

It took quite a few days for the issues to resurface. We began seeing hostname resolution failures in Kibana and started another investigation into the cause. After some digging, it was decided that something had to be wrong with DNS resolution in the Kubernetes clusters, so we added a trailing dot to the Redis connection URI - i.e. `this.is.our.redis.uri.` to force authoritative DNS resolution for the hostname in question.

After rolling the changes out into our dev environment, we found that the host resolution problems began decreasing, chalked it up as a fix, and moved forward with pull requests to implement the trailing dot for the rest of the environments.

The next day (and after the merge of the PR), more DNS resolution failures. This turned into a legitimate incident and we started triaging the problem to determine the true impacts. After digging in, we tried using a self-started Busybox instance to test DNS resolution for the Redis hostname - it passed with flying colors. So the Kubernetes cluster wasn't doing anything different with DNS than what was expected and passed through the non-cluster hostname for external resolution.

We retrieved the contents of the `/etc/resolv.conf` file from one of the pods and began searching based on the contents below:

```
nameserver 10.10.10.10
search ns.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

While researching DNS resolver configuration, we also noticed that the connection count for the Redis cache was significantly higher than expected - close to 95% utilization. We believed that may be causing Redis to cave in under stress and cause the connection failures experienced by new services. A quick confirmation showed that we were erroneously setting the maximum connection pool size and a quick pull request and version cut of the library in question got that under control, shaving around 70% off of our connection count.

While the Redis connection pool updates were happening, several items came up in our research. Namely...

- A new understanding of DNS resolution and how `etc/resolv.conf` works got us to understand the `ndots` option and what it means for hostnames
- Additional error details buried in the stack trace revealed a new issue - `Server name value of host_name cannot have the trailing dot`, indicating this version of Netty did not allow the trailing dot in a hostname
- Additional claims about Kubernetes ndots settings and the problems it causes

The immediate applied fix was to update the `ndots` configuration for deployed services using an additional `dnsConfig` block inside the Kubernetes deployment manifests. Since the hostname in question had four dots, an `ndots: 4` setting worked just fine for our case.

The fixes start flowing

We started rolling out the `ndots` fix to a few services to determine if it resolved the DNS issue and it seemed to work for the interim. We informed the other teams that they may need to configure their `ndots` option to use a value of `4` to force external DNS resolution in certain cases and what problems may arise inside Kubernetes with the new value.

We don't leverage any internal service discovery, so the chances of significant impacts from lowering the `ndots` value was an acceptable risk for us to take.

During additional research on the issue for true root cause and contributing factors, we came across two interesting issues that flipped our understanding of the problem:

- We weren't the only ones that seemed to have problems with DNS resolution for non-Kubernetes hosts
- The Alpine MUSL library has some pretty serious DNS resolution caveats
- Netty DNS resolution didn't work as expected with the version we were using (4.1.30)

Given those three new datapoints, we came to a few conclusions:

- Kubernetes `ndots` was the catalyst for a combination of DNS resolution issues
- Netty's DNS resolver does not work as expected (this has since been resolved)
- The Alpine caveats for DNS resolution can cause serious issues if the first nameserver response is `NXDOMAIN`

Lessons learned

I spent more time than I'd care to admit learning how DNS resolution works, how Kubernetes handles in-cluster vs. out-of-cluster DNS entries, and the issues facing the musl DNS resolution API. I added several Debian, `glibc`-based Docker base images into our image library and my team is testing those images to see if they resolve our DNS problems.

The easiest workaround to restore most services was an update to the `ndots` value that lowered it to an acceptable threshold. That workaround won't hold forever though, and we still have DNS resolution errors from time to time.

Ultimately, I think the move away from Alpine base images to slim Debian images will fix our problems at the container image level and updates to our dependencies that leverage newer Netty versions will fix the rest.

I'd love to see better alerting and monitoring for stuff like this, but network failures happen and erroneous alerts for one-off DNS failures is not the desired state. There's still a lot for me to figure out in this space, but I'm hoping some of what I wrote here can save a few people some troubleshooting later on.

I'm also always open to better solutions, so always feel free to reach out on Twitter if you have a better fix or you've tackled this on your own - I'd love to hear what you did!

KUBECON
KubeCon 2018, Day One Recap
Day 1 of KubeCon is in the books. I'm planning on writing up a better post once the entire conference is done, but I wanted to call out a few things I really enjoyed from the day. The opening keynotes were great, but in
ADAM MARGHERIO
12 DEC 2018 · 2 MIN READ

BASH
Loading Multiple Kubernetes Configs
Since I've started moving towards more Kubernetes cluster operations and management in my day job, I've run into an immediate pain that is managing multiple cluster configs and easily switching from one config to another. The Kubernetes documentation for managing multiple cluster configs is
ADAM MARGHERIO
17 AUG 2018 · 1 MIN READ