This Grace is not overrated

Kubernetes' dirty endpoint secret and Ingress

Grace is overrated

Fri, Jul 26, 2019

At Ravelin we've migrated to Kubernetes (on GKE). This has been very successful. We've got pod disruption budgets coming out of our ears, our statefulsets are very stately, and rolling node replacements run without a hitch.

The last piece of the puzzle is to move our API layer from the old VMs into our kubernetes cluster. For this we need to set up an Ingress so the API can be accessed from the outside world.

At first this seems straight-forward. We just define the ingress controller, tinker with terraform to get some IP addresses and Google takes care of nearly everything else. And it all works like magic. Great!

But we begin to notice our integration tests are occasionally receiving 502 errors. And there begins a journey that I'll save you the pain of reading about by cutting directly to the final conclusions.

Graceful shutdown.

Everybody talks about Graceful shutdown. But you really shouldn't do it in Kubernetes. Or at least not the Graceful shutdown you learned at your mother's knee. This level of grace is unnecessary to the point of danger in the world of Kubernetes.

The Good Place

Here's how everyone would love to think that removing a pod from a service or a load balancer works in Kubernetes.

1. The replication controller decides to remove a pod.
2. The pod's endpoint is removed from the service or load-balancer. New traffic no longer flows to the pod.
3. The pod's pre-stop hook is invoked, or the pod receives a SIGTERM.
4. The pod 'gracefully shuts down'. It stops listening for new connections.
5. The graceful shutdown completes, and the pod exits, when all its existing connections eventually become idle or terminate.

Unfortunately this just isn't how it works.

The Real Story

Much of the documentation hints that this isn't how it works but it doesn't spell it out. The big issue in this process is that step 2 does not happen before step 3. They happen at the same time. With normal services removing the endpoints is so quick you are unlikely to notice a problem. But ingresses are usually quite a bit slower to react, so issues become very readily apparent. The pod may receive the SIGTERM quite some time before the change in endpoints is actioned at the ingress.

This has the consequence that "Gracefully shutting down" is really not what the pod should do. It will receive new connections and it must continue to process them or the client will receive 500 errors and the whole wonderful story of seemless deploys and scaling will begin to fall apart.

This is what really happens.

1. The replication controller decides to remove a pod.
2. The pod's endpoint is removed from the service or load-balancer. For ingresses this may take some time, and new traffic will continue to be sent to the pod.
3. The pod's pre-stop hook is invoked, or the pod receives a SIGTERM.
4. The pod should largely ignore this, keep running, and keep serving new connections. If it can, it can hint to its clients that they should move on elsewhere. If it uses HTTP it might want to set "Connection": "close" in headers on responses.
5. The pod exits only when its termination grace period expires and it is killed with SIGKILL.
6. Be sure that this grace period is longer than it takes to reprogram your load balancer.

If it's 3rd party code and you can't change it's behavior then the best you can do is to add a pre-stop lifecycle hook that sleeps for the length of the grace period so the pod will just continue serving as if nothing happened.