Don't Panic: Kubernetes and Docker

Wednesday, December 02, 2020

Authors: Jorge Castro, Duffie Cooley, Kat Cosgrove, Justin Garrison, Noah Kantrowitz, Bob Killen, Rey Lejano, Dan “POP” Papandrea, Jeffrey Sica, Davanum “Dims” Srinivas

Update: Kubernetes support for Docker via `dockershim` is now deprecated. For more information, read the deprecation notice. You can also discuss the deprecation via a dedicated GitHub issue.

Kubernetes is deprecating Docker as a container runtime after v1.20.

You do not need to panic. It’s not as dramatic as it sounds.

TL;DR Docker as an underlying runtime is being deprecated in favor of runtimes that use the Container Runtime Interface (CRI) created for Kubernetes. Docker-produced images will continue to work in your cluster with all runtimes, as they always have.

If you’re an end-user of Kubernetes, not a whole lot will be changing for you. This doesn’t mean the death of Docker, and it doesn’t mean you can’t, or shouldn’t, use Docker as a development tool anymore. Docker is still a useful tool for building containers, and the images that result from running `docker build` can still run in your Kubernetes cluster.

If you’re using a managed Kubernetes service like GKE, EKS, or AKS (which defaults to containerd) you will need to make sure your worker nodes are using a supported container runtime before Docker support is removed in a future version of Kubernetes. If you have node customizations you may need to update them based on your environment and runtime requirements. Please work with your service provider to ensure proper upgrade testing and planning.

If you’re rolling your own clusters, you will also need to make changes to avoid your clusters breaking. At v1.20, you will get a deprecation warning for Docker. When Docker runtime support is removed in a future release (currently planned for the 1.22 release in late 2021) of Kubernetes it will no longer be supported and you will need to switch to one of the other compliant container runtimes, like containerd or CRI-O. Just make sure that the runtime you choose supports the docker daemon configurations you currently use (e.g. logging).

So why the confusion and what is everyone freaking out about?

We’re talking about two different environments here, and that’s creating confusion. Inside of your Kubernetes cluster, there’s a thing called a container runtime that’s responsible for pulling and running your container images. Docker is a popular choice for that runtime (other common options include containerd and CRI-O), but Docker was not designed to be embedded inside Kubernetes, and that causes a problem.

You see, the thing we call “Docker” isn’t actually one thing—it’s an entire tech stack, and one part of it is a thing called “containerd,” which is a high-level container runtime by itself. Docker is cool and useful because it has a lot of UX enhancements that make it really easy for humans to interact with while we’re doing development work, but those UX enhancements aren’t necessary for Kubernetes, because it isn’t a human.

As a result of this human-friendly abstraction layer, your Kubernetes cluster has to use another tool called Dockershim to get at what it really needs, which is containerd. That's not great, because it gives us another thing that has to be maintained and can possibly break. What's actually happening here is that Dockershim is being removed from Kubelet as early as v1.23 release, which removes support for Docker as a container runtime as a result. You might be thinking to yourself, but if containerd is included in the Docker stack, why does Kubernetes need the Dockershim?

Docker isn't compliant with CRI, the Container Runtime Interface. If it were, we wouldn't need the shim, and this wouldn't be a thing. But it's not the end of the world, and you don't need to panic—you just need to change your container runtime from Docker to another supported container runtime.

One thing to note: If you are relying on the underlying docker socket (`/var/run/docker.sock`) as part of a workflow within your cluster today, moving to a different runtime will break your ability to use it. This pattern is often called Docker in Docker. There are lots of options out there for this specific use case including things like kaniko, img, and buildah.

What does this change mean for developers, though? Do we still write Dockerfiles? Do we still build things with Docker?

This change addresses a different environment than most folks use to interact with Docker. The Docker installation you're using in development is unrelated to the Docker runtime inside your Kubernetes cluster. It's confusing, we understand. As a developer, Docker is still useful to you in all the ways it was before this change was announced. The image that Docker produces isn't really a Docker-specific image—it's an OCI (Open Container Initiative) image. Any OCI-compliant image, regardless of the tool you use to build it, will look the same to Kubernetes. Both containerd and CRI-O know how to pull those images and run them. This is why we have a standard for what containers should look like.

So, this change is coming. It's going to cause issues for some, but it isn't catastrophic, and generally it's a good thing. Depending on how you interact with Kubernetes, this could mean nothing to you, or it could mean a bit of work. In the long run, it's going to make things easier. If this is still confusing for you, that's okay—there's a lot going on here; Kubernetes has a lot of moving parts, and nobody is an expert in 100% of it. We encourage any and all questions regardless of experience level or complexity! Our goal is to make sure everyone is educated as much as possible on the upcoming changes. We hope this has answered most of your questions and soothed some anxieties! ❤️

Looking for more answers? Check out our accompanying Dockershim Removal FAQ *(updated February 2022)*.

← Previous

Next →