Comparison with Docker Swarm

This lesson is a comparison between Kubernetes resource management and Docker Swarm equivalent.

WE'LL COVER THE FOLLOWING

The Similarities

Resource management can be divided into a few categories. We need to define how much memory and CPU we expect a container will use and what are the limits. This information is crucial for a scheduler to make “intelligent” decisions when calculating where to place containers. In this aspect, there is no essential difference between Kubernetes and Docker Swarm. Both are using requested resources to decide where to deploy containers and limits when to evict them. Both of them are, more or less, the same in this aspect.

How can we know how much memory and CPU to dedicate to each of our containers?

That’s one of the questions we hear way too many times. The answer is simple. Collect metrics, evaluate them, adjust resources, take a break, repeat.

Where do we collect metrics?

Wherever you want. Prometheus is a good choice.

Where will it get metrics?

Well, it depends on which scheduler you use. If it’s Docker Swarm, you’ll need to run a bunch of exporters. Or, you might be brave enough and try the experimental feature that exposes Docker’s internal metrics in Prometheus

format. You might even be enthusiastic enough to think that they will be enough for all your monitoring and alerting needs. Maybe, by the time you read this, the feature is not experimental anymore. On the other hand, Kubernetes has it all, and so much more. You can use Metrics Server, or you might discover that it is too limiting and configure Prometheus to scrape metrics directly from Kubernetes API. Kubernetes exposes a vast amount of data. More than you'll probably ever need. You will be able to fetch memory, CPU, IO, network, and a myriad of other metrics and make intelligent decisions not only about the resources your containers require but about so much more.

To make things clear, you can get the same metrics no matter whether you're running Kubernetes or Docker Swarm. The major difference is that Kubernetes exposes them through its API, while with Swarm you'll have to struggle between the decisions whether to use its limited metrics or go into the trouble of setting up the exporters like cAdvisor and Node Exporter. Most likely, you'll discover that you'll need both the metrics from Swarm's API and those from the exporters. Kubernetes has a more robust solution, even though you might still need an exporter or two. Still, having most, if not all, of the metrics you'll need from its API is a handy thing to have.

The Differences

Frankly, the differences in the way we retrieve metrics from the two schedulers are not of great importance. If this would be where the story about resources ends, we'd conclude that both solutions are, more or less, equally good. But, the narrative continues. This is where similarities stop. Or, to be more precise, this is where Docker Swarm ends, and Kubernetes only just began.

Kubernetes allows us to define resource defaults and limitations that are applied to containers that do not specify resources.

It allows us to specify resource quotas that prevent accidental or malicious over-usage of resources. The two combined with Namespaces provide very powerful safeguards. They give us some of the means with which we can design the system that is truly fault tolerant by preventing rogue containers, uncontrolled scaling, and human errors from bringing our clusters to a grinding halt. Don't think, even for a second, that quotas are the only thing required for building a robust system. It isn't the only piece of the puzzle, but it is a significant one never the less.

Namespaces combined with quotas are important. We'd even say that they are crucial. Without them, we would be forced to create a cluster for every group, team, or a department in our organizations. Or, we might have to resort to further tightening of the processes that prevent our teams from exploiting the benefits behind container orchestrators.

Concluding Remarks

If the goal is to provide freedom to our teams without sacrificing cluster stability, Kubernetes has a clear edge over Docker Swarm.

> This battle is won by Kubernetes, but the war still rages.

OK. We exaggerated a bit with the words *battle* and *war*. It's not a conflict, and both communities are increasing collaboration and sharing ideas and solutions. Both platforms are merging. Still, for now, Kubernetes has a clear edge over Docker Swarm on the subject of resource management.

In the next chapter, we will experience how to create a production-ready cluster.