Comparison with Docker Swarm

This lesson is all about the comparison of Kubernetes Namespaces with the Docker Swarm equivalent.

WE'LL COVER THE FOLLOWING

> Docker Swarm does not have anything like Kubernetes Namespaces.

We cannot split a Swarm cluster into sections. Therefore, we can finish this comparison by saying that Kubernetes is a clear winner regarding this feature since Docker Swarm doesn't have Namespaces. But, that would not be entirely accurate.

The Similarities

Docker Swarm stacks are, in a way, similar to Kubernetes Namespaces. All the services in a stack are uniquely identified through a combination of a stack name and the names of services inside it. By default, all services within a stack can communicate with each other through the stack's default network. Services can speak with those from other stacks only if they are explicitly attached to the same network.

All in all, each Swarm stack is separated from other stacks. They are, in a way, similar to Kubernetes Namespaces.

The Differences

Even though Docker Swarm stacks do provide a functionality similar to Kubernetes Namespaces, their usage is limited. If, for example, we'd like to split the cluster into production and testing, we'd need to create two potentially large Swarm stack files. That would be impractical.

Moreover, Kubernetes Namespaces can be associated with resource quotas, policies, and quite a few other things. They do act as genuinely separate clusters. Swarm stacks, on the other hand, are meant to group services into logical entities. While some of the features in Kubernetes Namespaces and Docker Swarm stacks coincide, this is still a clear **win for Kubernetes**.

Some might argue that they are useful only for bigger clusters or organizations with many teams. We think that's an understatement. Namespaces can be applied to many other use-cases.

For example, creating a new Namespace for every continuous integration, delivery, or deployment pipeline is a beneficial practice. We get a unique scope for names, we can mitigate potential problems through resource quotas, and we can increase security. At the end of the process, we can remove the Namespace and all the objects we created inside it.

Conclusion

Kubernetes Namespaces are one of the things that make Kubernetes a more likely candidate for teams that are in need of big clusters as well as those relying heavily on automation. Among the features we compared so far, this is the first real differentiator between the two platforms.

> Kubernetes is the winner of this round.

Let’s move on to the next chapter that covers everything about the security and access mechanism of a Kubernetes cluster.