Technology Conversations

Docker Clustering Tools Compared: Kubernetes vs Docker Swarm

Kubernetes and Docker Swarm are probably two most commonly used tools to deploy containers inside a cluster. Both are created as helper tools that can be used to manage a cluster of containers and treat all servers as a single unit. However, they differ greatly in their approach.

Kubernetes

Kubernetes is based on Google's experience of many years working with Linux containers. It is, in a way, a replica of what Google has been doing for a long time but, this time, adapted to Docker. That approach is great in many ways, most important being that they used their experience from the start. If you started using Kubernetes around Docker version 1.0 (or earlier), the experience with Kubernetes was great. It solved many of the problems that Docker itself had. We could mount persistent volumes that would allow us to move containers without loosing data, it used flannel to create networking between containers, it has load balancer integrated, it uses etcd for service discovery, and so on. However, Kubernetes comes at a cost. It uses a different CLI, different API and different YAML definitions. In other words, you cannot use Docker CLI nor you can use Docker Compose to define containers. Everything needs to be done from scratch exclusively for Kubernetes. It's as if the tool was not written for Docker (which is partly true). Kubernetes brought clustering to a new level but at the expense of usability and steep learning curve.

Docker Swarm

Docker Swarm took a different approach. It is a native clustering for Docker. The best part is that it exposes standard Docker API meaning that any tool that you used to communicate with Docker (Docker CLI, Docker Compose, Dokku, Krane, and so on) can work equally well with Docker Swarm. That in itself is both an advantage and a disadvantage at the same time. Being able to use familiar tools of your own choosing is great but for the same reasons we are bound by the limitations of Docker API. If the API doesn't support something, there is no way around it through Swarm API and some clever tricks need to be performed.

We'll explore those two tools in more details based on their setup and features they provide for running containers in a cluster.

Setting Up

Setting up Docker Swarm is easy, straightforward and flexible. All we have to do is install one of the service discovery tools and run the *swarm* container on all nodes. Since the distribution itself is packed as a Docker container, it works in the same way no matter the operating system. We run the *swarm* container, expose a port and inform it about the address of the service discovery. It could hardly be easier than that. We can even start using it without any service discovery tool, see whether we like it and when our usage of it becomes more serious, add etcd, Consul or some of the other supported tools.

Kubernetes setup is quite more complicated and obfuscated. Installation instructions differ from OS to OS and provider to provider. Each OS or a hosting provider comes with its own set of instructions each of them having a separate maintenance team with a separate set of problems. As example, if you choose to try it out with Vagrant, you are stuck with Fedora. That does not mean that you cannot run it with Vagrant and, let's say, Ubuntu or CoreOS. You can, but you need to start searching for instructions outside the official Kubernetes Getting Started page. Whatever your needs are, it's likely that the community has the solution but you still need to spend some time searching for it and hoping that it works from the first attempt. The bigger problem is that the installation relies on a bash script. That would not be a big deal in itself if we would not live in the era where configuration management is a must. We might not want to run a script but make Kubernetes be part of our Puppet, Chef or Ansible definitions. Again, this can be overcome as well. You can find Ansible playbooks for running Kubernetes or you can write your own. None of those issues are a big problem but, when compared with Swarm, they are a bit painful. With Docker we were supposed not to have installation instructions (aside from a few `docker run` arguments). We were supposed to run containers. Swarm fulfils that promise and Kubernetes doesn't.

While some might not care about which discovery tool is used, I love the simplicity behind Swarm and the logic "batteries included but removable". Everything works out-of-the-box but we still have the option to substitute one component with the other. Unlike Swarm, Kubernetes is opinionated tool. You need to live with the choices it made for you. If you want to use Kubernetes, you have to use etcd. I'm not trying to say that etcd is bad (quite contrary) but if you prefer, for example, to use Consul you're in a very complicated situation and would need to use one for Kubernetes and the other for the rest of your service discovery needs. Another thing I dislike about Kubernetes is its need to know things in advance, before the setup. You need to tell it the addresses of all your nodes, which role each of them has, how many minions there are in the cluster and so on. With Swarm, we just bring up a node and tell it to join the network. Nothing needs to be set in advance since the information about the cluster is propagated through *gossip*.

Set up might not be the most important difference between those tools. No matter which tool you choose, sooner or later everything will be up and running and you'll forget any trouble you might have had during the process. You might say that we should not choose one tool over the other only because one is easier to set up. Fair enough. Let's move on and speak about differences in how you define containers that should be run with those tools.

Running Containers

How do you define all the arguments needed for running Docker containers with Swarm? You don't! Actually, you do but not in any form or way different from the way you were defining them before Swarm. If you are used to run containers through Docker CLI, you can continue using it with (almost) the same commands. If you prefer to use Docker Compose to run containers, you can continue using it to run them inside the Swarm cluster. Whichever way you're used to run your containers, chances are that you can continue doing the same with Swarm but on a much larger scale.

Kubernetes requires you to learn its CLI and configurations. You cannot use *docker-compose.yml* definitions you created earlier. You'll have to create Kubernetes equivalents. You cannot use Docker CLI commands you learned before. You'll have to learn Kubernetes CLI and, likely, make sure that the whole organization learns it as well.

No matter which tool you choose for deployments to your cluster, chances are you are already familiar with Docker. You are probably already used to Docker Compose as a way to define arguments for the containers you'll run. If you played with it for more than a few hours, you are using it as a substitute for Docker CLI. You run containers with it, tail their logs, scale them, and so on. On the other hand, you might be a hard-core Docker user who does not like Docker Compose and prefer running everything through Docker CLI or you might have your own bash scripts that run containers for you. No matter what you choose, it should work with Docker Swarm.

If you adopt Kubernetes, be prepared to have multiple definitions of the same thing. You will need Docker Compose to run your containers outside Kubernetes. Developers will continue needing to run containers on their laptops, your staging environments might or might not be a big cluster, and

so on. In other words, once you adopt Docker, Docker Compose or Docker CLI are unavoidable. You have to use them one way or another. Once you start using Kubernetes you will discover that all your Docker Compose definitions (or whatever else you might be using) need to be translated to Kubernetes way of describing things and, from there on, you will have to maintain both. With Kubernetes everything will have to be duplicated resulting in higher cost of maintenance. And it's not only about duplicated configurations. Commands you'll run outside of the cluster will be different from those inside the cluster. All those Docker commands you learned and love will have to get their Kubernetes equivalents inside the cluster.

Guys behind Kubernetes are not trying to make your life miserable by forcing you to do things "their way". The reason for such big differences is in a different approaches Swarm and Kubernetes are using to tackle the same problem. Swarm team decided to match their API with the one from Docker. As a result, we have (almost) full compatibility. Almost everything we can do with Docker we can do with Swarm as well only on a much larger scale. There's nothing new to do, no configurations to be duplicated and nothing new to learn. No matter whether you use Docker CLI directly or go through Swarm, API is (more or less) the same. The negative side of that story is that if there is something you'd like Swarm to do and that something is not part of the Docker API, you're in for a disappointment. Let us simplify this a bit. If you're looking for a tool for deploying containers in a cluster that will use Docker API, Swarm is the solution. On the other hand, if you want a tool that will overcome Docker limitations, you should go with Kubernetes. It is power (Kubernetes) against simplicity (Swarm). Or, at least, that's how it was until recently. But, I'm jumping ahead of myself.

The only question unanswered is what those limitations are. Two of the major ones were networking and persistent volumes. Until Docker Swarm release 1.0 we could not link containers running on different servers. Actually, we still cannot link them but now we have *multi-host networking* to help us connect containers running on different servers. It is a very powerful feature. Kubernetes used flannel to accomplish networking and now, since the Docker release 1.9, that feature is available as part of Docker CLI.

Another problem were persistent volumes. Docker introduced them in release 1.9. Until recently, if you persist a volume, that container was tied to the server that volume resides. It could not be moved around without, again, resorting to some nasty tricks like copying volume directory from one server to another. That in itself is a slow operation that defies the goals of tools like Swarm. Besides, even if you have time to copy a volume from one to the other server, you do not know where to copy since clustering tools tend to treat your whole datacenter as a single entity. You containers will be deployed to a location most suitable for them (least number of containers running, most CPUs or memory available, and so on). Now we have persistent volumes supported by Docker natively.

Both networking and persistent volumes problems were one of the features supported by Kubernetes for quite some time and the reason why many were choosing it over Swarm. That advantage disappeared with Docker release 1.9.

The Choice

When trying to make a choice between Docker Swarm and Kubernetes, think in following terms. Do you want to depend on Docker itself solving problems related to clustering. If you do, choose Swarm. If something is not supported by Docker it will be unlikely that it will be supported by Swarm since it relies on Docker API. On the other hand, if you want a tool that works around Docker limitations, Kubernetes might be the right one for you. Kubernetes was not built around Docker but is based on Google's experience with containers. It is opinionated and tries to do things in its own way.

The real question is whether Kubernetes' way of doing things, which is quite different from how we use Docker, is overshadowed by advantages it gives. Or, should we place our bets into Docker itself and hope that it will solve those problems? Before you answer those questions, take a look at the release 1.9. We got persistent volumes and software networking. We also got *unless-stopped* restart policy that will manage our unwanted failures. Now there are three things less of a difference between Kubernetes and Swarm. Actually, these days there are very few advantages Kubernetes has over Swarm. On the other hand, Swarm uses Docker API meaning that you get to keep all your commands and Docker Compose configurations. Personally, I'm placing my bets on Docker engine getting improvements and Docker Swarm running on top of it. The difference between the two is very small. Both are production ready but Swarm is easier to set up, easier to use and we get to keep everything we built before moving to the cluster; there is no duplication between cluster and non-cluster configurations.

My recommendation is to go with Docker Swarm. Kubernetes is too opinionated, hard to set up, too different from Docker CLI/API and at the same time it doesn't have real advantages over Swarm since the Docker release 1.9. That doesn't mean that there are no features available in Kubernetes that are not supported by Swarm. There are feature differences in both directions. However, those differences are, in my opinion, not major ones and the gap is getting smaller with each Docker release. Actually, for many use cases there is no gap at all while Docker Swarm is easier to set up, learn and use.

This entry was posted in Docker and tagged Cluster, Container, Data Center, Docker, Docker Compose, Kubernetes on November 4, 2015 [http://technologyconversations.com/2015/11/04/docker-clustering-tools-compared-kubernetes-vs-docker-swarm/] by Viktor Farcic.

8 thoughts on "Docker Clustering Tools Compared: Kubernetes vs Docker Swarm"

Zed

November 5, 2015 at 5:07 am

great write up, thanks

Ward K Harold
November 11, 2015 at 4:23 pm

We've used Kubernetes (k8s) in production since the 1.0 launch. Swarm wasn't production ready so it wasn't really a choice. We also appreciated that we were getting a decade of Google's experience building and running clusters for free.

Once we were running in production we realized that the particular container technology is actually a very small part of a cluster solution. The cluster services are the bread and butter. We could replace Docker with Rocket in our cluster and aside from a slightly different workflow for building images no one would notice. If we were tied to the Docker API that wouldn't be possible.

As for setup, while we got comfortable doing k8s installs both in the cloud and on local boxes, we ultimately decided to go with Google Container Engine, effectively outsourcing DevOps to Google. That also was not an option with Swarm, though it may be in the future.

katenkabarabash
December 3, 2015 at 9:40 am

What about compare those tools to Mesos? It would be interesting for me

Daniel Middleton (@monokal)
November 5, 2015 at 1:19 pm

Is it not the case that Swarm is more of a "fire and forget" deployment mechanism as opposed to Kubernetes which has replication controllers and a service abstraction which can transparently maintain a "desired state" as the underlying hosts fail? Admittedly I could be missing something, but I see Swarm more as a deployment tool, and Kubernetes more of a larger-scale orchestration platform for high availability.

Viktor Farcic Post author
November 5, 2015 at 4:25 pm

It is true that Kubernetes can maintain the desired state and Swarm cannot. However, with, for example Consul watches, it is fairly easy to do the same. Ping a service and if it doesn't respond, re-deploy using the same process that was run initially. I think that the problem is bigger than simply maintaing the state. Sometimes random things happen and server stops working. That is a good case where re-deployment done by Kubernetes is useful (even though it's fairly easy to do the same with Swarm/Consul combination). In many other cases, things failed for a reason that needs to be fixed in a way that is not a simple re-deployment. Someone needs to be notified with all the information and investigate why things failed. For example, it can be a memory leak that needs to be addressed by changing the code and not by simply re-deploying continuously.

I might have been too harsh on Kubernetes. It's a great tool. However, I think that its incompatibility with Docker API is the main problem. On the other hand, Docker is implementing features currently available in Kuberentes. The best example is networking and persistent volumes. Those were indeed one of the main arguments in favor of Kubernetes.

Actually, if something more powerful than Swarm is needed, I think that Mesos is the right choice especially since it supports non-Docker deployments as well (no matter how much I like Docker, there are cases, like Hadoop, when it is not appropriate). I think that Kubernetes is somewhere in between. Not as friendly as Swarm nor as powerful as Mesos. On top of that, Swarm will probably be available on top of Mesos before it gets to be integrated with Kubernetes.

Pingback: Links 5/11/2015: Framing Linus Torvalds, NetBeans IDE 8.1 | Techrights

Pingback: Docker Clustering Tools Compared: Kubernetes vs Docker Swarm | Technology Conversations on WordPress.com http://technologyconversations.com/2015/11/04/docker-clustering-tools-compared-kubernetes-vs-docker-swarm/

Pingback: Docker集群工具比对：Kubernetes vs Docker Swarm | 小样儿(ShowYounger)