Stateful Kubernetes in the Cloud

“Kubernetes is one of the fastest growing open source software projects in history,” according to the Cloud Native Computing Foundation (CNCF) Kubernetes Project Journey Report notes. Maybe you’ve heard it has something to do with containers and distributed systems. Or, maybe you heard Jim Zemlin, Executive Director of the Linux Foundation, recently proclaim, “Kubernetes is becoming the Linux of the cloud.”

By Paul Burt, Technical Product Marketing Engineer at NetApp and a member of SNIA.

What’s less well known is that a lot of stateful and data-centric workloads are also starting to make their way to Kubernetes. Here is a look at which workloads and databases are a good fit for Kubernetes. If you’re looking for more educational resources on Kubernetes, don’t miss the links to great content at the end of this article.

Stateful Workloads

Are containers and Kubernetes becoming a better fit for stateful workloads? The answer is yes and no.

Running stateful workloads on Kubernetes has been possible for some time now. However, it might seem to an outsider that Kubernetes features that support stateful workloads are hard to find. For example, a text search for “stateful” on the Kubernetes 1.16 changelog (a recent release) returns no results. If a source change was made for StatefulSets, or “Stateful workloads” we would hope to see a result. Of course, confirming functionality in a piece of large and complex software by doing a free text search is never quite that simple.

The point here is that stateful workloads are running the same way now as they were on many prior versions of Kubernetes. Part of the reason is stateful workloads are difficult to generalize. In the language of Fred Brooks (author of *The Mythical Man Month*), we could say that stateful workloads are essentially complex. In other words, they must be addressed with a complex solution.

Solutions like operators (sets of design patterns) are tackling some of this complexity, and recent changes to storage components indicate progress is occurring elsewhere. A good example is in the 1.14 release (March 2019), where local persistent volumes graduated to general availability (GA). That’s a nice resolution and fix to the complaint on this blog from

2016, which said, *"Currently, we recommend using StatefulSets with remote storage. Therefore, you must be ready to tolerate the performance implications of network attached storage. Even with storage optimized instances, you won't likely realize the same performance as locally attached, solid state storage media."*

Local persistent volumes fix some of the performance concerns around running stateful workloads on Kubernetes. Has Kubernetes made progress? Undeniably. But are stateful workloads still difficult? Absolutely.

Transitioning databases and complex stateful work to Kubernetes requires a steep learning curve. Running a traditional database through your current setup typically does not require additional knowledge. Your current database just works, and you and your team already know everything you need to know to keep that going.

Running stateful applications on Kubernetes requires you gain knowledge of init containers, persistent volumes (PVs), persistent volume claims (PVCs), storage classes, service accounts, services, pods, config maps and more. This large learning requirement has remained one of the biggest challenges to running stateful workloads on Kubernetes. Moving to Kubernetes can be very rewarding, but it also comes with a large cost which is why cautionary tales still abound.

What databases are best suited to run on Kubernetes? Mainly those which have embraced the container revolution which has been happening over the past five years. Typically, those databases have a native clustering capability. For example, CockroachDB has great documentation available with examples showing how to set it up on Kubernetes.

On the other hand, Vitess provides clustering capabilities on top of MariaDB or MySQL to better enable these solutions to run on Kubernetes. Vitess has been accepted as an incubation project by the CNCF, so there is a lot of development expertise behind it to ensure it runs smoothly on Kubernetes.

Traditional relational databases like Postgres or single-node databases like Neo4j are fine for Kubernetes. The big caveat is that these are not designed to scale in a cloud native way. That means the responsibility is on you to understand the limits of those databases, and any services they might support. Scaling these pre-cloud solutions tends to require sharding, or other similarly tricky techniques. As long as we maintain a comfortable distance from the point where we'd need to scale a pre-cloud database with any strategy other than, "put it on a bigger machine," we should be fine.

Cloud vs. On-Premises

The decision to run Kubernetes in the cloud vs. on-premises is usually dictated by external factors. In the cloud we tend to benefit most from the managed services or elasticity. All of the major cloud providers have database offerings which can be exposed to Kubernetes environments as a service: for example, Amazon Aurora, Google Cloud SQL, and Microsoft Azure DB. These offerings can be appropriate if you are a smaller shop without a lot of database architects.

However, regulatory requirements like GDPR, specific country requirements, or customer requirements may mandate we run on-premises. This is often where the concept of data gravity comes into effect – it is easier to move compute to your data rather than moving your data to compute. This is one of the reasons why Kubernetes is popular. Wherever your data lives, you'll find you can bring your compute closer (with minimal modifications), thanks to the consistency provided by Kubernetes and containers.

Overcoming Complexity

Aside from the steep learning curve, it can seem like there are a number of other major challenges that come with Kubernetes. These challenges are often due to other design choices, like adopting microservices, infrastructure as code, etc. These other approaches are shifts in perspective, that change how we have to think about infrastructure.

As an example, James Lewis and Martin Fowler note how a shift toward microservices will complicate storage: *"As well as decentralizing decisions about conceptual models, microservices also decentralize data storage decisions. While monolithic applications prefer a single logical database for persistent data, enterprises often prefer a single database across a range of applications - many of these decisions driven through vendors' commercial models around licensing. Microservices prefer letting each service manage its own database, either different instances of the same database technology, or entirely different database systems - an approach called Polyglot Persistence."*

Failing to move from a single enormous database to a unique datastore per service can lead to a distributed monolith. That is, an architecture which looks superficially like microservices, but behind the scenes still contains many of the problems of a monolithic architecture.

Kubernetes and containers align well with newer cloud native technologies like microservices. It's no surprise then, that a project to move toward microservices will often involve Kubernetes. A lot of the perceived complexity of Kubernetes actually happens to be due to these accompanying approaches. Microservices and other approaches are often paired with Kubernetes, and anytime we stumble over a rough area, it can be easy to just blame Kubernetes for the issue.

Kubernetes and these newer development methodologies are popular for a number of reasons. They've been proven to work at extreme scale at companies like Google and Netflix. When done right, development teams also seem more productive.

If you are a business operating at a larger scale and struggling with productivity, this probably sounds like a great solution. On the other hand, if you have yet to feel the pain of scaling, all of this Kubernetes and microservices stuff might seem a bit silly. There are a number of good reasons to avoid Kubernetes. There are also a number of great reasons to embrace it. We should be mindful that the value of Kubernetes and associated technologies is very dependent on where our businesses are on, "feeling the pain of scaling."

Kubernetes is flexible, and if we find running projects outside of it easier, we still have that option. Kubernetes is designed so that it's easy to mix and match with other solutions.

Kubernetes Resources

There are a multitude of educational resources on Kubernetes. The SNIA Cloud Storage Technologies Initiative (CSTI) has a great three-part "Kubernetes in the Cloud" webcast series. They're viewable on-demand here Kubernetes in the Cloud (Part 1), Kubernetes in the Cloud (Part 2)

Kubernetes in the Cloud (Part 3) - (Almost) Everything You Need to Know about Stateful Workloads or watch them on the SNIAVideo YouTube channel.

As part of this series, the CSTI has also compiled more than 25 links of useful and informative resources during our live webcast. You can access all of them here.

SNIA™ CSTI | CLOUD STORAGE TECHNOLOGIES

The SNIA Cloud Storage Technologies Initiative (CSTI) is committed to the adoption, growth and standardization of storage in cloud infrastructures, including its data services, orchestration and management, and the promotion of portability of data in multi-cloud environments. To learn more about the CSTI's activities and how you can join, visit snia.org/cloud.