GRAVITATION

# Experiences with running PostgreSQL on Kubernetes

JAN 22, 2018 BY ABRAHAM INGERSOLL

## Introduction

Below is a transcript of an interview with our CTO, Sasha Klizhentas, about his experience running PostgreSQL on Kubernetes. In this interview, we discuss the challenges involved, open source and commercial tools that can help and other alternatives to managing stateful applications on Kubernetes.

For some background, Gravitational specializes in running applications across a variety of infrastructure footprints with the help of Kubernetes. The applications our customers deploy need a persistent data store to go along with their stateless

ARTICLES BY TOPIC

announcements

company

engineering

gravitational

gravity

kubernetes

programming

security

ssh

teleport

microservices. Making things more complicated is the fact that the majority of our deployments are on-premises private SaaS, so we can not rely on cloud services like AWS RDS.

## Challenges with running Postgres on Kubernetes

**Abe:** If someone wants to run Postgres or a similar database on Kubernetes where should they start?

**Sasha:** It's really hard to do. The hardest thing in running Postgres on Kubernetes is to understand that Kubernetes is not aware of the deployment details of Postgres. A naive deployment could lead to complete data loss.

> Kubernetes is not aware of the deployment details of Postgres. A naive deployment could lead to complete data loss.

Here's a typical scenario when that happens. You set up streaming replication and let's say the first master is up. All the writes go there and they asynchronously replicate to the standby. Then suddenly the current master goes down but the asynchronous replication has a huge lag caused by something like a network partition. If the naive failover leader election algorithm kicks in or the administrator who doesn't know the state manually triggers failover, the secondary becomes the master. That becomes the source of truth. All of the data during that period is lost because all of the writes that were not replicated disappear. Whenever the admin recovers the first master it's no longer the master any more and it has to completely sync the state from the second node which is now the master.

**Abe:** Have you seen this? Seen this with clusters you support at Gravitational?

**Sasha:** Yeah, that was a real data loss pattern we saw with asynchronous replication that caused loss.

**Abe:** You're talking about classic slony or the streaming replication features that have been built into Postgres since 9.x?

**Sasha:** Asynchronous replication sends operations to the followers / standby nodes. Those modifications could be changes to the state, writes, or creating new values. Whenever a chunk of this data is lost there should be a mechanism that tells the receiving node that its data is out of sync. In Postgres, there is a mechanism that helps to track replication lag. But there's no authority that analyzes this data that is built into Postgres, yet, that will help whoever is doing leader election to complete it.

> Kubernetes, makes it deceptively easy to implement basic failover and leader election.

The Citus Data team, by the way, handles replication correctly in their integration of Kubernetes with Postgres. They track replication state and replication lag - they have it all built in. But, if you are DIY, whoever builds Kubernetes deployments with failover should keep this in mind. Kubernetes, makes it deceptively easy to implement basic failover and leader election. For example, a user can create a deployment with replication factor of one and that's your leader election.

## Using Kubernetes StatefulSets for Native

## High Availability

**Abe:** What about Kubernetes StatefulSets? If you're coming in just needing a simple non-HA deployment, is it fair to think, "oh I could use StatefulSets to do this?"

**Sasha:** That's true. It's the simplest deployment that will probably work if someone has this concept of a volume of data that travels with the database. The replication is no longer necessary as-is because if the volume is persistent and it has its own snapshots and backups there's no need to replicate. However, with on-premises Kubernetes (which is what we mostly deal with), the implementation of the volumes is usually based on network filesystems, mounted filesystems like NFS, or filesystems like Ceph that simulate a block device through object storage. The clear downside of that is the increased latency. It's really hard to maintain a large, multi-tenant Ceph cluster and always have low latency for Postgres. You will probably still encounter data loss.

> That's the major flaw of all filesystem based replication mechanisms - they lack domain specific

> knowledge about the data type they are replicating and whether it's corrupted or not.

Imagine there are writes and suddenly Postgres goes down for any number of random reasons. Whatever is replicating those writes for Postgres on behalf of block storage has no domain specific knowledge about a replication log that was written to the disk. So it doesn't know what was corrupted or what state the replication log is and can not successfully recover it. That's the major flaw of all filesystem based replication mechanisms - they lack domain specific knowledge about the data type they are replicating and whether it's corrupted or not.

## Kubernetes opens up additional challenges for managing state

**Abe:** So it sounds like just Postgres in general (or any replicated SQL database) is just going to be a challenge unless you have specifically engineered, at a core engine level, for it to be Kubernetes or orchestrator-aware?

**Sasha:** Think about Kubernetes, in this case, as just a supporting system that gives you basic building blocks. It can give you the building block of a process that can be constantly up and be reliably migrated to another machine if one machine goes down. Or it can give you an invariant of the single leader out of several process running at any time. Kubernetes is really good at that. And it removes a lot of friction from engineers building those systems because before Kubernetes they had to reimplement all of these patterns themselves every time they would roll Zookeeper or any other distributed storage system that should exhibit those behaviors or those patterns.

> “
>
> The building blocks are there but you can still misuse them and have data loss as a result. It’s probably even easier to do so because the entry level is now lower.

So with Kubernetes, DIY of the orchestration is no longer necessary. You have better building blocks. But the complex part is to create domain-specific system that takes those building blocks and properly configures them - knows when to elect a leader or not, when is it safe, when is it not. The building blocks are there but you can still misuse them and have data loss as a result. *It's probably even easier to do so because the entry level is now lower.* Anyone can write a Kubernetes operator for Postgres but looking at the implementations you can just spot how they'll lose data.

**Abe:** If I am someone who is shipping some sort of SaaS product via AWS or GKE - I love the workflow of containers, I love RDS or Google Cloud SQL. But then I have a large enterprise buyer, or I blow my cloud budget and absolutely have to deploy to an on-prem datacenter, is your first instinct to not even use Kubernetes for stateful data? You'd recommend using bare machines or VMs? Here's your DB, it's special, put it on your SAN? You would even default to looking for the DBA within the existing enterprise IT shop and using their existing Postgres knowledge instead of trying to build a Kubernetes-based solution?

**Sasha:** Yeah, with on-prem, traditional deployments using manual failover and SAN would be easier and safer. That would be my first choice. I would only try to manage the lifecycle of a stateful workload under an orchestrator if I had no other choice.

**Abe:** Wow, I've worked within a lot of enterprises and I can't even imagine how long it'd take to get time on that DBA team's backlog.

**Abe:** One of the other interesting things I've heard you say in the past is that etcd, as soon as it can't fsync fast enough will freeze Kubernetes in its current state. You can't do anything with it.

**Sasha:** Yeah managing distributed database is hard, which is why a DIY approach is challenging.

## Managing Postgres on Kubernetes is a full time job

**Abe:** So what do you do if you absolutely must use Postgres on Kubernetes, for whatever reason.

> Find a team or solution that is solely focused on Postgres and pair them with a team that knows Kubernetes well.

**Sasha:** To wrap up the conversation about Postgres, there is a way. Find a team or solution that is solely focused on Postgres and pair them with a team that knows Kubernetes well. I mentioned them earlier but the team at Citus Data is building a very sophisticated system that turns Postgres into a clustered solution. That's why we eventually ditched the concept of trying to build our own home-grown system from open source bits and now we're collaborating with the Citus folks. Just bring them our Kubernetes knowledge and we're using their Postgres knowledge. Their principal engineer, Marco Slot, is a distributed systems expert by training so he knows what he's doing with Postgres very well and Postgres is the team's core focus.

## When to put stateful workloads on Kubernetes

**Abe:** If you start to rip your database out, or you don't have database or a distributed systems experts on staff building Kubernetes clusters, what's the rule of thumb for when you should start thinking about an orchestrator vs just shipping your MVP without it, worrying about clustering later. When do you cross that threshold?

> The rule of thumb – that we've learned the hard way – with stateful workloads is to think very deeply about the problem space before committing to using Kubernetes to host it.

**Sasha:** The rule of thumb – that we've learned the hard way – with stateful workloads is to think very deeply about the problem space before committing to using Kubernetes to host it. It's very tempting but unless you have these specialized domain-specific solutions, it's not quite practical. The companies that should be looking into Kubernetes to automate their existing HA solutions are specific professionals in their field like Citus or 2ndQuadrant. Those are the companies that should be looking at Kubernetes. They already know all of the intricacies of deployments of highly available databases and their workloads. They should just start using Kubernetes as a faster tool to build the same primitives they would have built anyway.

For example, Citus have built their own monitoring solution, an agent, that should be running on one of the nodes that tracks the status. When I talked to them, I suggested they just use Kubernetes TPRs and spin up the deployment with a ReplicaSet of one and your agent will have consistent view of the world. That's where Kubernetes shines. But still, all the information that's shipped to this agent should be sent by them. And state machines that know when to elect, re-elect and all that stuff should be built by Citus' distributed systems experts.

**Abe:** TPRs? You mean CRDs? My favorite Helm package maintainer from Wisconsin likes to call them CURDS. Custom Resource Definitions, they replaced TPRs.

**Sasha:** Ha! Yes, "curds."

## Using NoSQL on Kubernetes

**Abe:** So then the next place my mind immediately goes to is what about Cassandra or something that's closer to the actual consistency model that you require for your use case, instead of just defaulting to a SQL DB? Do you recommend that folks stop thinking of simply wanting SQL, instead start thinking about how they actually need their data persisted and how they're using it? Are the operational model of the NoSQL options are easier to mate with Kubernetes or are you just starting to mix so many different concerns at that point that…wait, you're giving me dirty looks!

> “
> The specific database is not really the point. The complexity lies in deeply understanding how the database works and the workloads.

**Sasha:** The specific database is not really the point. The complexity lies in deeply understanding how the database works and the workloads. Let's say you have a perfect deployment of Postgres right now that uses asynchronous replication. You will still have edge cases that lose data.

Anyone who wants to deploy Cassandra on Kubernetes should also be aware that Kubernetes doesn't know anything about Cassandra hinted handoffs and when they should occur.

If you have five nodes and you create a deployment out of them and two of them went down and you have large dataset, Kubernetes will happily spin up two more nodes and add them to the cluster. What will likely happen in reality is that the nodes will start rebalancing the data to the point that the whole cluster will go down. It will be busy shuffling the data around and won't be able to serve reads. So that's a good example where Kubernetes has no idea about the workloads, the distribution of the data or the replication mechanisms. So there should be a system that uses Kubernetes mostly for the scheduling bits and Kubernetes should not be the one deciding whether it is safe or not to add two more replicas.

A good example of an operator that does this well is the etcd operator.

It is is aware of whether it is safe or not to add nodes, recover the cluster, if it is broken beyond repair, etc. but it's a lot of work. If you look at the code base, it's five to ten thousand lines of code.

## Human operators are still far from obsolete

**Abe:** Is it plausible just to use etcd for storing state directly?

**Sasha:** Maybe, but it's way safer, way easier, more programmatic to use a conservative deployment of the database for now.

**Abe:** If you were doing pure native etcd by way of CRDs? Essentially you're in the domain of the distributed systems stuff and then it's safer because you're there?

**Sasha:** If you host a really loaded database (of any kind), the amount of intelligence and operations monitoring to make a proper decision is enormous. You have to really know if it is safe to, lets say, bring down this node, for Postgres or for Cassandra depending on what type of workload it is. So to get an automatic operator to a useable state, the workload should be understood well enough so that the heuristics built in to the operator will always work because there is no human failsafe.

> DBAs are the operators right now and the intelligence that humans possess and the decision making process that they can take with the analysis of the state - you have to build something really, really sophisticated to replicate a good DBA team.

DBAs are the operators right now and the intelligence that humans possess and the decision making process that they can take with the analysis of the state - you have to build something really, really sophisticated to replicate a good DBA team. Almost like a self-driving car, in production! And that’s where I think the gap in the industry is right now. Oh Kubernetes will solve all your problems?! No! It will not. Because you have to be really as smart as the operations person who’s spent five years understanding Postgres.

**Abe:** And you need a PhD in Kubernetes.

**Sasha:** Not PhD, but.

**Abe:** You need to understand the primitives and how it works.

**Sasha:** So imagine you’ve deployed Postgres inside Kubernetes. Before you had tools, pgsql, pgtop, and all that stuff. And let’s say you want to build a replica. You use ansible to build your replicas. But if you put this whole thing inside Kubernetes, then first you have to either reinvent those tools or second, make them cluster-aware. Hey how do you connect to pg master now? How do you know, out of this deployment, which Postgres is the master and which one is the replica? There is no standard way - you have to build your own automation.

So in a way Kubernetes makes it way more complex because before everything was static. You had this sensible host file, this is your leader, these are your followers and everything was clear. But now you're just looking at this thing and there are three pods. Which one is leader? You don't know. So you have to go into each one and try to somehow detect if it is master or not.

**Abe:** One last question. Why do you hate Kubernetes?

**Sasha:** I don't hate it! I use it everyday and it's an amazing system. I'm just saying that Kubernetes, right now, supports some workloads really really well, like stateless workloads that are CPU intensive that can be easily scaled horizontally, with easy to understand patterns that we currently put behind load balancers.

Other workloads, Kubernetes just doesn't support well and likely will not support well in the near future because it's really hard to do. It's not magic, as some of the hype surrounding it would have you believe.

kubernetes gravity

Twitter Facebook LinkedIn

Hacker News

## Want to stay informed?

Subscribe to our newsletter to get articles and product updates.

Email Address SIGN UP

## Connect with Us

Dec 2, 2017

**Migrating To Kubernetes**

By Sasha Klizhentas

Nov 3, 2017

**Announcing Gravity v4**

By Sasha Klizhentas

Oct 19, 2017

**Troubleshooting Kubernetes Networking Issues**

By Sasha Klizhentas

## Start Using Gravity Today

Deploy and remotely manage cloud-native applications on premises with Gravity.

DEMO GRAVITY DOWNLOAD GRAVITY

Subscribe to our Newsletter

Your email address SEND

| PRODUCTS | LEARNING | COMPANY | GET IN TOUCH | CONNECT |
|---|---|---|---|---|
| Teleport | Resources | About Us | Customer Support | Community |
| Gravity | Teleport Docs | Careers | info@gravitational.com | Forum |
| Teleconsole | Gravity Docs | The Blog | Phone: (855) 818 9008 | Github |
| | Multi-Cloud | Events | | Twitter |
| | Advantage | Press & Media | | |
| | Federal Agencies | | | |