Maintaining the quorum - running stateful workloads on Kubernetes

With the popularity of the Kubernetes project growing within the community, more and more end-users are starting to use it to run stateful workloads. While Kubernetes provide foundations to easily run these class of services orchestrating them poses unique operation challenges in order to maintain quorum and ensure no service downtime. In this session we would like to discuss best-practices and different solutions to run effectively stateful workloads adopted by the community as well area of improvement to the Kubernetes project.

Possible topics for conversation:

- NoExecute taints
- Non preemptable priority classes
- PDB API
- Controllers
- …

Lead(s):
Notetaker(s): Mars Toktonaliev
Description:
Running stateful workloads on Kubernetes
Notes:
Stateful workloads:

- Require minimum # of replicas
- Might run some consensus/replication protocol
- Require specific knowledge/tooling to manage the lifecycle of service instances
- Have some kind of node affinity
- The reality is often each instance is a “pet” instead of a “catlle”

StatefulSet API:

- Stable application instances naming
- First class support for persistent storage (via volumeClaimTemplates specs field)
- Predictable deployment and scale-out logic

Taints:

- Allow workloads to opt-out from taint-based eviction
- Allow to assign workloads to a certain class of Nodes

PDB: PodDisruptionBudget

- Allow to influence scheduling decisions
- Allow to prevent involuntary disruption by operators

Controllers/Operators:

- Allows to build higher-order operation on top of core Kubernetes resources
- Streamline the lifecycle management of stateful workloads on Kubernetes

What are the open challenges?:

- Often require dealing with complexity of managing all the APIs/features
- Still have ad-hoc patterns to dedicated nodes to a specific workload in a clusters
- Not all Kubernetes APIs respects PDBs (e.g. taint-based eviction issues a Pod deletion, instead of calling the eviction API)
- Lack of PDB expressiveness (how to prevent a specific instance from deletion?)
- … more!

NOTES CARRIED OVER FROM ANOTHER DOC:

Maintaining the Quorum: Stateful Workloads

"All the typos are features". Did slides in 5min.

Different classes of stateful workloads.
- Some need minimum number of replicas
- sometimes have replicaiton protocol between pods, want to integrate service names
- not all replicas are the same
  - interrupting a leader is more expensive than interrupting a follower
- all require specific lifecycle knowledge to manage them. Will never be generic.
- There;s also some kind of node affinity for hardware or local volumes
- stateful workloads need to be "petted"

I think we have a great foundation for running stateful workloads.

StatefulSets is dry, is a good foundation for very simple clusters. It offers predictable naming. Also first-class support for persistent storage. Predictable deployment logic.

It's about giving control to the operator, delegating control to the user.

Then there's taints. Allows workloads to opt-out using NoExecute. Gives us certain nodes in the cluster that only certain workloads can run on. This node is not ready, let's ignore it. Combine with nodeSelects. Users can add their own taints. But very hard to decide toleration since they can be customized. Might want to use taint-based eviction for moving nodes.

PDBs: allow user to define a budget of how many instance can be evicted of a workload. Tell the scheduler if the pods can be rescheduled. And prevent draining wiping out a workload.

Controllers/Operators: adds higher-order automation on Kubernetes, on top of Kubernetes. Makes it easier for the user of the operator.

It's very complicated to run stateful workloads and get them right. Need to be able to set the tolerations and affinity. The API is very expressive, you can implement dedicated node pools, but there are so many ways to do it. Not everything in Kubernetes respects PDB, so you have to set up tolerations as well. And PDB isn't very expressive. This makes sense for the platform, but it's difficult as the owner of the workload.

PDB gives us how many instances you can take down, but what if I want to specify that you need to take down the nodes one a time synchronously? Or prevent taking down the leader?

Katerina: another problem of the statefulset workload upgrade policy. You can't really contol it. They end up writing their own controller. They use the ondelete strategy, but then the state doesn't get updated.

With large clusters you have zonal nodes in one cluster -- taints and tolerations can be too strict when you're trying to force zonal distribution, can result in services become unavailable. This other feature has the "scheduleanyway" property. Sometimes one pod connects to one service (didn't get this).

Doing updates is a problem we're solving again and again. Ondelete is also problematic, pods get deleted for lots of reasons.

Would like us to be in a place were we don't need users to implement their own replication controllers on top of the logic. Users need to write orchestration.

Resource management is also a problem, because kubernetes nodes don't like pods that consume all the resources. Also the OOM-killer hates databases. But the Runtime work on QoS may solve this.

Discussion around slices. The problem is that setup is static, and data changes over time.

Everything is very undefined, people are implementing it again and again. Slices might be away to do it but would need to integrate it with databases.

It's important to get the API right to prevent disruption. Need to keep looking into the workload.