The Struggles of Kubernetes Key-Value Stores

CS4991 Capstone Report, 2024

James Draudt
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
wzg6qh@virginia.edu

ABSTRACT

As Cloud Native Infrastructure continues to evolve rapidly, its built-in key-value store, ETCD, has struggled to keep pace due to a lack of maintainers and a heavy footprint. To address this issue, Kine has been developed as a shim for ETCD, enabling the use of alternative key-value store technologies within a cluster. By incorporating Kine, users can specify an endpoint flag during cluster creation to direct the cluster to a different database, which then serves as the key-value store. Some Kubernetes distributions have Kine built-in as an intermediary, while others require it to be run externally. However, any distribution can leverage this technology to bypass ETCD, leading to potential performance improvements such as reduced memory usage, faster task completion, and enhanced cluster availability. Exploring various backend options for Kine is crucial, particularly in determining which database technologies are most effective for Kubernetes. While multi-clustering via Kine is an intriguing concept, further testing is needed to assess its viability as a replacement for multi-cluster organizers like K0smotron.

1. INTRODUCTION

Cloud Native Infrastructure, and more specifically Kubernetes Clusters, allows modern technology to thrive. Many companies that haven't yet adopted these microservice architectures are considering doing so, as "[m]igration to the cloud has been a popular topic in industry and academia in recent years." However, "[d]espite the many benefits of the cloud, such as high availability and scalability, most on-premise application architectures are not fully prepared to exploit the benefits of this environment" (Balalaie et al., 2016). The main challenges seem to be reliability and ease of use for these new and complex cloud architectures. Cloud Native Architecture relies heavily on Kubernetes Clusters, which make programs more modular, allowing each component to work independently on any computer while still integrating seamlessly with the rest of the system. A key part of these architectures is ETCD, the built-in key-value store for Kubernetes. This database is vital for the cluster's operation, as it stores essential data.

Despite ETCD's importance in Cloud Native Infrastructure, it struggles with issues such as complexity, size, and updates. These challenges can affect performance since "both the Kubernetes control plane and the deployed application depend strongly, and sometimes unexpectedly, on the performance of the ETCD database" (Larsson et al., 2020). Because of these issues, some companies have sought workarounds, leading to the creation of Kine. Kine translates ETCD calls into a language that other databases can understand. This raises several questions: Which databases are best suited for Kubernetes key-value storage? How does the size of a cluster affect key-value store requirements? And finally,

what other uses could Kine technology have? To answer these questions I researched and experimented with different database backends, determined which work best with Kine through looking at the resulting data, and explored potential new uses for this technology.

2. RELATED WORKS

ETCD has well-documented limitations, many of which are highlighted in the paper by Larsson et al. (2020). This research discusses the performance link between ETCD and Kubernetes, revealing several underlying issues with ETCD. When analyzing inconsistent results, their "data indicate[d] that at the core of the problem is the ETCD database and its performance". This paper was among the first to expose the inherent complexity of ETCD and its potential to hinder performance, inspiring the need for new key-value store technologies.

Several resources were instrumental in my work with Kine. First, the Kine GitHub page (vitorsavian, 2024) provided the foundational knowledge I needed to experiment with different backend databases and understand how Kine operates. Much of my research was built on the insights provided by this page, and without it, I would have faced significant difficulties in utilizing this technology.

In addition, Denis (2023) introduced the idea of combining Kine with YugabyteDB, a distributed PostgreSQL database, which significantly advanced my research. Using this approach, I was able to create a unique distributed key-value store capable of operating globally and connect multiple clusters to a single database—a discovery that became one of the most interesting aspects of my project.

These three sources were pivotal in shaping my understanding of the problem and guiding me toward effective solutions. Without them, I would not have been able to grasp the complexity of the ETCD issues or develop viable alternatives.

3. PROJECT DESIGN

The proposed design consists of three main components: a Kubernetes distribution with Kine integration, Kine itself, and YugabyteDB. Combined, these elements enable administrators to set up their own distributed database, monitor it easily, and scale it to meet the needs of their cluster.

3.1 Kubernetes Distribution with Kine

The first part of the design is a Kubernetes distribution that includes Kine. Many newer, lightweight distributions have adopted Kine as it offers more flexibility, with some even moving away from using ETCD as the default backend. Among the options tested, K3s proved to be the best. It is lightweight and provides simple Kine commands that allow for quick database connectivity. While Kine can be used with any Kubernetes setup, without built-in support, it must be run as a standalone service, which adds extra work and system overhead. Another option tested was K0s, but it was found to be less user-friendly. However, with time and additional documentation, K0s could become a better choice for smaller clusters.

3.2 Setting up Kine with K3s

The next step is getting Kine running with K3s. Kine acts as a translator, converting ETCD-compatible calls into standard SQL, which is used by many other databases. This allows for simple cluster connections using

just a few flags. To do this, a flag is passed to K3s on startup with the --datastore-endpoint option. The connection string format is [DATABASE TYPE]://root:$[PASSWORD]@[LINK TYPE]([DATABASE IP])/kine. For more detailed instructions on connecting specific databases, I recommend referring to Kine's GitHub page or other relevant tutorials. Alternatively, this configuration can be set in the cluster's YAML file.

When configured properly, the cluster should perform similarly to one using ETCD. Depending on the backend database, performance may vary slightly, but when set up correctly, these differences will have a minimal impact on overall cluster performance.

3.3 Replacing ETCD with YugabyteDB

The most critical part of this design is replacing ETCD with YugabyteDB. YugabyteDB offers the same reliability as ETCD while providing a significantly improved user experience, increased customizability, and more robust monitoring capabilities.

YugabyteDB is a distributed PostgreSQL database, meaning it maintains copies of the same database across different nodes. This redundancy ensures that even if one copy is lost, the data remains accessible. Administrators can choose the location of these copies and optimize performance based on their locality. Yugabyte supports replication across local networks, regions, or even continents. Using techniques like sharding and leader tables, these copies remain synchronized efficiently, with minimal overhead. Additionally, Yugabyte allows for geo-isolated databases, where some information is shared across nodes while other data is confined to specific regions. This feature is especially useful for clusters that must comply with data privacy regulations.

YugabyteDB's customizable distribution and replication ensure that the database remains operational even if a majority of the copies are lost. This is achieved using the Raft Consensus Algorithm, which handles data replication. As long as one copy remains, the data is preserved and can be quickly restored after a simple restart.

A major advantage of YugabyteDB over ETCD is the elimination of a single point of failure. In ETCD, the database runs within the cluster, which creates a vulnerability. With YugabyteDB, however, the database can be distributed across multiple locations. If one copy is running within the cluster and two others are outside it, the system remains secure even if the cluster experiences downtime. Additionally, if the cluster goes offline, it can reconnect to the database backend and resume operation without issue. This contrasts with ETCD, where complex commands and buggy processes are required to restore from a snapshot.

3.4 Monitoring with YugabyteDB

Another significant advantage of YugabyteDB is its free, open-source monitoring software. This tool provides numerous utilities that allow users to identify potential issues before they become serious problems. This solves the challenge faced with ETCD, where setting up reliable monitoring was tedious and error prone.

3.5 Review

This design utilizes a Kubernetes distribution with Kine integration, sets up a Kine datastore backend, and connects to a YugabyteDB distributed database. This approach offers equal or greater reliability compared to ETCD, while significantly improving customizability, monitoring, and ease of use.

4. RESULTS

This model exceeded expectations in multiple areas. Not only did it meet the anticipated goals of reliability and ease of use, but it also demonstrated an unexpected capability for seamless multi-clustering. This feature allows multiple clusters to connect to a single Yugabyte database, enabling them to work together and balance loads automatically.

Setting up Yugabyte with K3s proved to be straightforward, requiring just one flag for K3s and a few simple commands to configure the distributed database. The monitoring tools were highly customizable and performed exceptionally well. They provided clear visibility into any issues, detailed information about each database copy, and useful warnings that helped troubleshoot problems with ease. Overall, Yugabyte lived up to its reputation for simplicity and proved to be a strong option as a key-value store.

Through testing, it was discovered that connecting two clusters to the same backend enabled effortless multi-clustering. This system allowed for automatic workload distribution, even when work was initially assigned to only one cluster. It also provided failover recovery, seamlessly transferring processes from a failed cluster to the operational one, and offered comprehensive monitoring across all clusters. This solution has the potential to replace current multi-clustering technologies, which are still challenging to use and prone to bugs. While further work is needed to ensure this approach is viable long-term, the results are extremely promising.

5. CONCLUSION

This project highlights the growing need for efficient and scalable cloud-native infrastructure, particularly in key components such as Kubernetes' key-value store. As cloud infrastructure adoption accelerates, every element within this ecosystem must be capable of handling increased demand. However, ETCD—the default key-value store for Kubernetes—struggles to meet the requirements of larger clusters, making its replacement both necessary and beneficial. My investigation into alternative solutions revealed workarounds for ETCD, allowing for the deployment of customizable clusters suited to various sizes and demands.

By leveraging Kine to bypass standard Kubernetes calls, I found that any SQL-compliant database could potentially replace ETCD and enable effective cluster management. In particular, YugabyteDB stood out as a strong candidate, offering high availability, configurability, and efficiency, especially for larger clusters. An unexpected finding was the capability of a single key-value store to support multi-cluster management, a feature that could serve as a viable alternative to current, less mature multi-cluster management solutions.

6. FUTURE WORK

Replacing ETCD requires further exploration and testing. For future work, a comprehensive benchmarking of various SQL databases should be conducted to assess the viability of different backends for Kubernetes. Additionally, long-term validation is needed to determine how well these SQL databases perform in production environments over extended timeframes. Further configuration testing with YugabyteDB is also recommended to optimize node setups for maximum availability and efficiency.

In terms of multi-cluster management, more extensive testing and development are required. Due to time constraints, I could only evaluate this framework briefly, so additional research would help identify and address potential limitations. I anticipate some challenges with this setup; however, if these can be overcome, multi-clustering on a single

key-value store could unlock significant advantages previously untapped in cloud infrastructure.

7. ACKNOWLEDGMENTS

I would like to express my gratitude to my intern team and mentor for supporting my decision to pursue this niche topic, which enabled me to explore innovative solutions. I am also thankful to Denis (2023) for an exceptional tutorial on setting up clusters with YugabyteDB, without which my work would not have been possible.

REFERENCES

Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Migrating to cloud-native architectures using microservices: An experience report. In A. Celesti & P. Leitner (Eds.), *Advances in Service-Oriented and Cloud Computing* (pp. 201–215). Springer International Publishing. https://doi.org/10.1007/978-3-319-33313-7_15

Larsson, L., Tärneberg, W., Klein, C., Elmroth, E., & Kihl, M. (2020). Impact of etcd deployment on Kubernetes, Istio, and application performance. *Software: Practice and Experience*, *50*(10), 1986–2007. https://doi.org/10.1002/spe.2885

Denis, M. (2023, July 28). *Kubernetes evolution: Transitioning from etcd to distributed sql - dzone*. Dzone.Com. https://dzone.com/articles/kubernetes-evolution-transitioning-from-etcd-to-di

Vitorsavian (2024). *Kine/examples/minimal. Md at master · k3s-io/kine*. GitHub; k3s. https://github.com/k3s-io/kine/blob/master/examples/minimal.md