Cloud Computing: Kubernetes Application Performance Improvement Using In-Memory Database

Binoy Kurikaparambil Revi

Independent Researcher, USA
Email: binoyrevi@live.com

Abstract

With remarkable advancements in computing power and the emergence of Kubernetes clusters, application deployment in the cloud has effectively addressed numerous challenges related to accessibility, availability, security, and scalability. This transformation is particularly evident for complex applications that demand substantial computing resources, which can now run efficiently on platforms like Azure Kubernetes Service. This service provides access to high-performance GPUs and CPUs, allowing applications to harness computing powers for their computational needs. As powerful computing resources become increasingly accessible, application performance and functionality expectations have evolved significantly. Today, most modern applications are intricately reliant on data for training machine learning models, data mining, or executing advanced image processing tasks. This heavy reliance on data necessitates the ability to process large datasets with remarkable speed and accuracy. The In-Memory database has emerged as an invaluable solution to meet these stringent requirements. This approach enables data storage in memory, accelerating computational algorithms and enhancing fast caching capabilities. It streamlines application data processing and offers a significant advantage over traditional methods that rely on files or conventional on-disk databases.

Keywords: Kubernetes, In-Memory Database, Redis, Docker

Introduction:

Complex and resource-intensive computation and data processing applications present significant challenges that can be grouped into two primary areas. Firstly, the algorithms and overall system design involved in these applications can be quite complex, often requiring deep expertise and careful consideration of numerous domain-related challenges. Secondly, the necessity to execute these algorithms and manage extensive datasets can be both critical and complicated, mainly when timely results are essential for decision-making processes. In this demanding environment, introducing additional overhead—such as managing incoming data streams and preparing this data for computation—can hinder performance and efficiency. This is precisely where the advantages of in-memory databases become invaluable for designing and implementing complex applications. By leveraging an in-memory database, systems can effectively handle data at remarkable speeds, dramatically boosting processing efficiency and response times.

Furthermore, in-memory databases enhance data caching capabilities, allowing frequently accessed data to be stored in a manner that vastly improves data transaction speed. This fact is particularly advantageous for applications requiring rapid access to large volumes of data. Additionally, including Publish and Subscribe features within in-memory databases is highly beneficial for the design of cloud-based applications,

enabling seamless communication between various data sources and ensuring that data is instantaneously available to the subscribers on publishing. Integrating in-memory databases significantly enhances the functionality and performance of complex computational applications deployed on Kubernetes clusters.

Related Studies:

Yonatan Gottesman, Joel Nider, Ronen Kat, Yaron Weinsberg, and Michael Factor[1] from IBM Research have conducted research on modifying the well-known Redis database to be compatible with Storage Class Memory. This modification resulted in a significant average latency reduction of 43% and an average throughput increase of 75%.

Abdullah Talha Kabakus and Resul Kara[2] extensively evaluated various read and write cycles across different record sizes and databases. The results were impressive, clearly demonstrating that the Redis database is more efficient than other databases in their experiments.

Security validation is a crucial aspect of any technology, including databases used in the cloud. Numerous studies have been conducted on various in-memory databases. One notable research paper is titled “Security Assessment of NoSQL MongoDB, Redis, and Cassandra Database Managers,”[3] authored by Ricardo Andrés González Sánchez, Davor Julián Moreno Bernal, and Hector Dario Jaimes Parada. The findings from their study indicate that no database is 100% secure from injection attacks. However, it is evident from their analysis that Redis demonstrates moderate security compared to the others.

Another notable and thorough evaluation of Redis and MongoDB can be found in a study by Nadia Ben Seghier and Okba Kazar. Their paper, "Performance Benchmarking and Comparison of NoSQL Databases: Redis vs MongoDB vs Cassandra Using YCSB Tool,"[4] focuses on benchmarking three major databases: Redis, MongoDB, and Cassandra. They, along with several other researchers, conclude that Redis is more efficient in read operations, which include Read Only, Read Mostly, Heavy Update, and Read-modify-write scenarios.

These studies summarize that Redis is a strong candidate for in-memory database needs when designing a complex application, whether it is intended to run on the cloud or on-premises.

Problem Description:

One key behavior of heavy applications running on the Kubernetes cluster is that it may require some standard protocol to communicate with an external application or device. At the same time, the application may not have enough bandwidth to handle the data, process the data, and set up the data in an acceptable form for the data processing or computational module to process this data. There are different programming methods to handle the data read from the external applications, which are not efficient enough. Some techniques[1] used are given below:

1. Application Data Structure: This can be visualized as the application directly reads and manages data in data structures like linked lists, as a List, or something similar. However, this must be developed and maintained as part of the application software. Synchronization of incoming data is an overhead to the application, especially when the application is developed for time-critical tasks.
2. Multi-Threading: This method is preferred over the application's main process thread reading data. Additional threads must be developed as part of application development and must be maintained and monitored. This is, again, a complex task.
3. Traditional database: An on-disk or hybrid database may look like a problem solver, as external applications can write data to it, and the application software needs to use the data from the database to perform computation or data processing. However, this is inefficient in a time-critical project, as it can slow down the entire system.

In-Memory Database Solution:

The In-Memory database is a program that creates and manages the database purely in the memory and does not use the disk. Even if the In-Memory database is managed on the memory, most of the In-Memory database providers provide the option to have a persistent database. From the research studies by various researchers[2], it seems clear that the Redis Database is the most efficient In-Memory data that can be used to manage the data in a time-critical application[4]. Two major key features Redis provides are a very efficient Key-Value pair database and a publish-subscribe mechanism for data exchange.

1. Key-Value Pair database: This is Redis' basic database functionality. The application can store data as a key-value pair, where the key is unique, and the value can be updated at any time[5]. This is simple, and most programming languages welcome it as it provides a highly efficient mechanism to store programming data like start-up configurations, intermediate data, user profiles, etc…
2. Publish/Subscribe (Pub/Sub) Mechanism: The Redis Pub/Sub model is an excellent design option for creating an efficient software architecture to develop the Kubernetes application that can seamlessly receive data or messages from other applications instantaneously and send messages to other applications to act on that message instantaneously. With this, asynchronous communication can be implemented much more efficiently without tightly coupling the communication module to the application. In the Redis Pub/Sub model, Redis acts as a broker that manages various named data channels. The broker accepts requests from various applications to subscribe to an existing Redis Channel. Applications that are subscribed to a named channel will get all the messages published to the channel. Similarly, Redis accepts request applications to publish to an existing named channel, and the publisher application can publish messages to the corresponding channel.

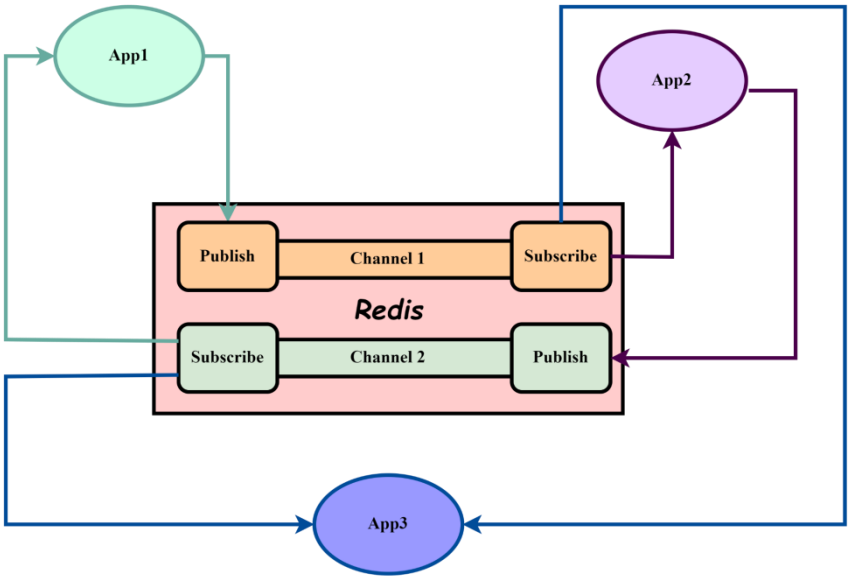

Figure 1: Redis Pub/Sub Model

Figure 1 illustrates the Redis Pub/Sub model. In this model, Application App1 is a publisher to channel 1 and a subscriber to channel 2. Application App2 is a publisher to channel 2 and a subscriber to channel 1. Application App 3 is not a publisher but a subscriber to channel 1 and channel 2. If application App1 publishes any message on channel 1, App2 and App3 receive those messages instantaneously. Similarly, if App2 publishes any messages to channel 2, App1 and App3 receive the messages instantaneously. This model is instrumental in creating architecture for cloud infrastructure with multiple internal and external applications including applications running on the Kubernetes cluster.

Using Redis In-Memory Database in Kubernetes Application:

The basic and most essential thing in building a Kubernetes application[8] is to build the Docker image, and without a doubt, the Docker file is the key to building the image. It is easy for a developer to write a docker file from scratch if the resources and the versions of the operating system, packages, and tools are known. Redis provides Redis Docker Image that is built on Ubuntu to start the application development.

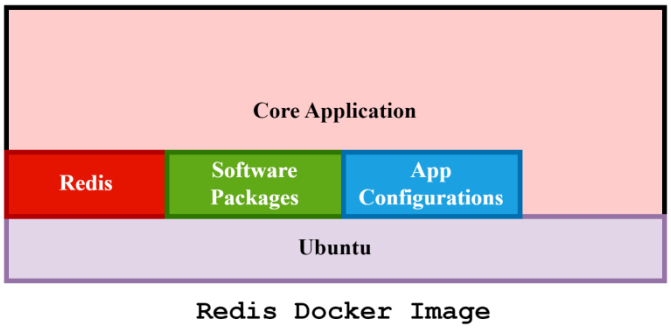

Figure 2: Redis/Ubuntu Docker Image

Figure 2 provides a view of a typical docker image stack that is configured for an application to run. On the docker file, the Redis/Ubuntu image can be pulled, and the application and its dependencies can be added using the following algorithm:

- ➔ Use the "FROM" command to add the Redis Image to the Docker file
- ➔ Use the RUN command, run the Ubuntu OS update command using the apt-get command
- ➔ Use the RUN command, install the necessary package to support the application execution using the apt-get command
- ➔ Use the "COPY" setup, copy the application to the predefined folder from the local folder
- ➔ Use the "EXPOSE" command, open a dedicated port for communication. Redis uses this port to communicate with other applications using the PUB/SUB mechanism.
- ➔ Using the "CMD" setup to start the application.

Finally, once the Docker file is completed, build it using the docker build command.

For the Kubernetes application, the docker images are generally added to the cloud container registry. The images are then deployed to the kubernates cluster from the container registry. Figure 3 give the pictorial representation of Container Registry and deployment to the Kubernates cluster.

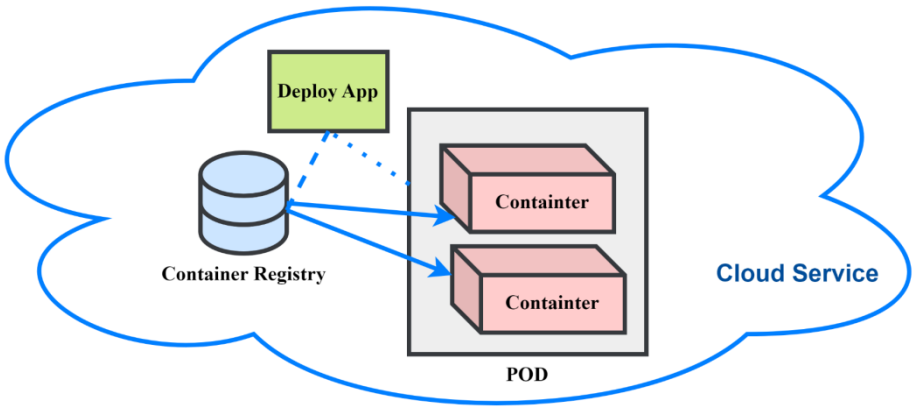

Figure 3: Storage and Kubernetes Deployment of Docker Image

Conclusion:

Kubernetes services by Cloud providers provide a great platform to deploy heavy and complex domain-specific applications. In-memory databases like Redis provide an excellent data management mechanism to store and act on the data in Memory, thereby reducing latency and increasing the throughput and efficiency of the application. On top of this, Redis provides a publish/subscribe mechanism to send messages to the applications that like to receive the message and listen to messages from the applications that are expected to send the message. This removes the application's overhead from managing data transfers. Now, the overhead is entirely handled by Redis Data Broker. In the cloud, the publish and subscribe mechanism makes even more sense, as the container in Kubernetes has a specific address and port that can be used to communicate with the application. Overall, the In-Memmory database is an excellent option when designing real-time or near real-time software for both On-Prem and cloud requirements.

References:

1. Yonatan Gottesman, Joel Nider, Ronen Kat, Yaron Weinsberg, and Michael Factor. 2016. Using Storage Class Memory Efficiently for an In-memory Database. In Proceedings of the 9th ACM International on Systems and Storage Conference (SYSTOR '16). Association for Computing Machinery, New York, NY, USA, Article 21, 1. https://doi.org/10.1145/2928275.2933273
2. Abdullah Talha Kabakus, Resul Kara, A performance evaluation of in-memory databases, Journal of King Saud University - Computer and Information Sciences, Volume 29, Issue 4, 2017, Pages 520-525, ISSN 1319-1578, https://doi.org/10.1016/j.jksuci.2016.06.007
3. R. A. G. Sánchez, D. J. M. Bernal and H. D. J. Parada, "Security assessment of Nosql Mongodb, Redis and Cassandra database managers," 2021 Congreso Internacional de Innovación y Tendencias en Ingeniería (CONIITI), Bogotá, Colombia, 2021, pp. 1-7, doi: 10.1109/CONIITI53815.2021.9619597
4. N. B. Seghier and O. Kazar, "Performance Benchmarking and Comparison of NoSQL Databases: Redis vs MongoDB vs Cassandra Using YCSB Tool," 2021 International Conference on Recent Advances in Mathematics and Informatics (ICRAMI), Tebessa, Algeria, 2021, pp. 1-6, doi: 10.1109/ICRAMI52622.2021.9585956.
5. S. Chen, X. Tang, H. Wang, H. Zhao and M. Guo, "Towards Scalable and Reliable In-Memory Storage System: A Case Study with Redis," 2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, 2016, pp. 1660-1667, doi: 10.1109/TrustCom.2016.0255.
6. E. Tang and Y. Fan, "Performance Comparison between Five NoSQL Databases," 2016 7th International Conference on Cloud Computing and Big Data (CCBD), Macau, China, 2016, pp. 105-109, doi: 10.1109/CCBD.2016.030.
7. A. Gupta, S. Tyagi, N. Panwar, S. Sachdeva and U. Saxena, "NoSQL databases: Critical analysis and comparison," 2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN), Gurgaon, India, 2017, pp. 293-299, doi: 10.1109/IC3TSN.2017.8284494.
8. W. Viktorsson, C. Klein and J. Tordsson, "Security-Performance Trade-offs of Kubernetes Container Runtimes," 2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Nice, France, 2020, pp. 1-4, doi: 10.1109/MASCOTS50786.2020.9285946.
9. J. Patel and T. Halabi, "Optimizing the Performance of Web Applications in Mobile Cloud Computing," 2021 IEEE 6th International Conference on Smart Cloud (SmartCloud), Newark, NJ, USA, 2021, pp. 33-37, doi: 10.1109/SmartCloud52277.2021.00013.