

Hazelcast

by Alexander Edinger, Jonas Eppard and Fernand Hoffmann

TODO: General Introduction

1 Key-Value-Stores

TODO:

1.1 History of Key-Value-Stores

TODO: Change

1.2 Working principle of Key-Value-Stores

TODO:

1.3 Comparison of Key-Value-Stores to relational databases

TODO:

2 Hazelcast in Theory

Hazelcast, or formally known as Hazelcast IMDG, is an In-Memory Datagrid written in Java by the Hazelcast corporation. It is a cluster storage and combines "Data at Rest" with "Data in Motion" by providing a streaming architecture for dynamic data together with low-latency storage for static data.(Hazelcast, [2022i](#)) (Hazelcast, [2022p](#))

2.1 Architecture of Hazelcast

At the core of the Hazelcast platform stands the distributed architecture of cluster nodes with the streaming engine and the low-latency storage. A cluster consists of at least one node. The streaming engine is in the form of a Kappa-Architecture which, compared to a Lambda-Architecture, provides the same methodology for data processing. The low-latency storage provides availability and partition tolerance with possible consistency through the use of the CP-subsystem. Hazelcast offers different possibilities for a client to connect to the cluster through the use of APIs and Connectors. Implementations exist for multiple languages, including Java, .NET, C++, Node.js, Python and Go.(Hazelcast, 2022p) (Hazelcast, 2022f) Encapsulating the Connectors and the cluster are the security features of Hazelcast for clients and nodes. This includes the authentication of new nodes and clients, as well as the encryption of traffic between the nodes or between a client and the cluster.

2.2 Hazelcast in CAP-Theorem

The CAP-Theorem states, that a distributed system can not be partition tolerant, consistent and highly available at the same time, but can only fulfill two of the three properties. A 2010 article further adds, that partition tolerance cannot be sacrificed, because sacrificing partition tolerance would require a network to never drop any messages, which in reality cannot exist.(Brewer, 2000a) (Hale, 2010)

Hazelcast is by default an AP-system, favoring availability over consistency. Since each client interacts with the cluster through a gateway node and node failures can happen in a distributed system, Hazelcast replicates the data on backup nodes to ensure availability.(Hazelcast, 2022a) (Hazelcast, 2022l)

Some distributed datastructures, for example a Lock or a Semaphore, require strong consistency rather than high availability. For such cases, Hazelcast offers a CP-subsystem, which is built upon the cluster and allows for consistent storage of specified distributed objects. The subsystem is based on the RAFT consensus algorithm, through which the participating nodes in the subsystem can reach a consensus on correct data by periodic heartbeats and votings.(Hazelcast, 2022a) (Hazelcast, 2022b) (Woos et al.,

2016)

Since Hazelcast is an In-Memory Data Grid, no persistence happens by default. To strengthen consistency of data structures, persistence can be configured for nodes in the CP-subsystem to persist the data to a non-volatile storage.(Hazelcast, 2022b)

2.3 Hazelcast is BASE

The acronym BASE stands for Basically Available, Soft-State and Eventual Consistency and is used to describe a system with those properties (Brewer, 2000a). Hazelcast fits all of the three properties due to the following reasons: Due to Hazelcast being an AP-system, it prioritizes availability and can thus be considered basically available. Through periodic heartbeats between the cluster nodes, a failure in nodes can be detected and data from backup nodes can be fetched.(Hazelcast, 2022d)

Due to the high availability offered by Hazelcast, an exact state of the cluster cannot be determined to a certain point in time. This is further amplified when choosing one-phase commits for transactions, where the commit log is not copied to other nodes. Consequently, a node in the process of writing data can be interrupted due to a node failure and leave the system in an inconsistent state. (Hazelcast, 2022c)

To prevent increasing entropy in the cluster, Hazelcast synchronizes the nodes in periodic intervals. The primary node, the node which is delegated to store the data, sends a summary of the data to its backup nodes. The backup nodes compare the data with their version and in the case of an inconsistency, the synchronisation process is triggered.(Hazelcast, 2022j)

3 Hazelcast in Practice

The goal of this section is to test Hazelcast in practice. With this goal in mind, we will first install Hazelcast, and then we will try to implement some example data structures. We will take a look at the Usability of the Hazelcast interface.

In this section following limitations are taken into account: First only the Open-Source Hazelcast Platform is used. Second, only the Hazelcast Command Line Interface (CLI) is used. Therefore, the Hazelcast cloud service

Viridian is not used and will not be evaluated. Furthermore, the Hazelcast Clients outside the Hazelcast CLI are not used, and will not be evaluated. The data structure used for testing is taken from a previous lecture. **TODO:**

3.1 Installation of Hazelcast

There are multiple ways to install Hazelcast. Due to the fact that Hazelcast is open-source and provides Binaries. Therefore, it's possible to build the Hazelcast Platform from source. However, it is easier to download prebuild binaries. For Linux there is a Debian package available. For macOS there is a brew package available, but for Windows there is no package or installer available. Therefore, under Windows you have to download prebuild binaries, if you don't want to build Hazelcast from source. Another option is to use docker. (Hazelcast, 2023b)

We tried the Hazelcast installation on multiple operating systems, but weren't able to test the installation on macOS. Therefore, we will only describe the installation on Linux and Windows, and will take a look at the installation via docker. The installation on Linux is very easy and straight forward: First you will download the public key of the Hazelcast repository and add the repository to your package manager sources:

```
wget -qO - https://repository.hazelcast.com/api/gpg/key/public | gpg
  --dearmor | sudo tee /usr/share/keyrings/hazelcast-archive-keyring
  .gpg > /dev/null
echo "deb [signed-by=/usr/share/keyrings/hazelcast-archive-keyring.
gpg] https://repository.hazelcast.com/debian stable main" | sudo
tee -a /etc/apt/sources.list
```

Listing 1: Adding the Hazelcast repository to the package manager sources under Linux (Debian) (Hazelcast, 2023b)

After that you will be able to install Hazelcast via the package manager like any other package:

```
sudo apt update && sudo apt install hazelcast
```

Listing 2: Installing Hazelcast under Linux (Debian) (Hazelcast, 2023b)

The installation under Windows was a bit more complicated. First Hazelcast did not provide a Windows installer. Therefore, we had to download the binary from the Hazelcast website. This binary wasn't able to run directly,

because it needed the `JAVA_HOME` environment variable to be set. First we tried to install a Java Runtime Environment (JRE) and set the `JAVA_HOME` variable accordingly. However, the JRE was not able to run the Hazelcast binary and a Java Development Kit (JDK) was needed. Therefore, we installed the JDK set the path, and then the Hazelcast binary was able to run. However, personally we found the installation with docker the easiest. First, the Hazelcast image is downloaded from the docker hub:

```
docker pull hazelcast/hazelcast
```

Listing 3: Downloading the Hazelcast image from the docker hub (Hazelcast, 2023b)

After that the local Hazelcast Cluster is started as specified in the documentation:

```
docker network create hazelcast --network
docker run \
  -it \
  --network hazelcast --network \
  --rm \
  -e HZ_CLUSTERNAME=hello-world \
  -p 5701:5701 hazelcast/hazelcast
```

Listing 4: Starting the Hazelcast Cluster using the docker image (Hazelcast, 2023c)

For the following example the docker installation will be used.

3.2 Example in Hazelcast

In this section an example data structure will be implemented using the Hazelcast CLI. The Hazelcast CLI is run as a docker container. The data structure is taken from a previous lecture and shown in Fig. 1. The data

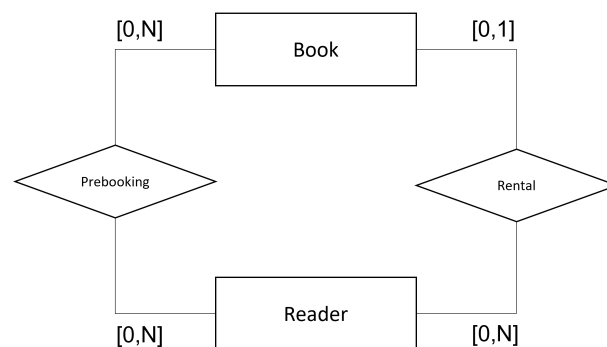


Figure 1: Example data structure (see also Mairhofer, 2021)

structure models a library with books. The library has multiple books and multiple readers. Each book can be rented by one reader. The reader can rent multiple books. And if a book is rented by a reader, another reader can't rent it, but order a prebooking for it. This data structure was given to us in an example in a previous lecture by Mairhofer (2021). The exact fields of the data structure are not important for this example. The important part is that the data structure requires foreign keys to model the relations.

Hazelcast is a distributed key-value store, as described in ???. It does not provide a relational database, and it has no support for foreign keys. It is possible to implement relations with embedded objects or with user defined fields. To define the Relations between the object classes we will use user defined fields. Embedded objects have the problem, that there are hard to manage by hand and difficult to implement many-to-many relations. In the user defined fields we can define the fields like a foreign key in a relational database, but Hazelcast does not provide any safety checks. Therefore, a user has to implement safety checks by hand, if the consistency in the relations are important. In this example we will not write an application and only take a look at the Hazelcast CLI. Therefore, we will not implement any safety checks. But Hazelcast provides a way to select and join different maps. Thus, it is possible to implement the relations and query those relations inside the Hazelcast CLI.

To conclude our example we have successfully implemented a data structure with foreign keys in the key-value store Hazelcast. However, the relations are not checked by the Hazelcast platform and the user has to implement the checks to ensure that the relations are consistent. Therefore, we advise using Hazelcast as a key-value store and not to try using the Hazelcast platform to implement complex relations.

4 Conclusion and Recommendations

Hazelcast is a database that can be used for a variety of different tasks. It offers scalability due to the cluster architecture, the streaming engine makes it viable for fast processing. Due to the default prioritization of availability over consistency, it is suited to handle high amounts of data,

while also retaining the possibility of consistency through the CP-subsystem.

Hazelcast especially excels in dynamic environments due to its built-in streaming engine that allows dynamic data to be evaluated and transformed in real-time. The variety of clients and connectors allows for a simple integration of Hazelcast in an application.

Due to the many features that Hazelcast offers, an initial solution built with Hazelcast can grow complex quickly and switching databases from Hazelcast to another provider is coupled with difficulties, because Hazelcast does not offer a direct exportation of data.

In the future, NoSQL-databases are projected to rise in market size. Hazelcast will likely partake in this trend as a Multi-model database with a focus on Key-Value storage. The flexibility to process dynamic data through its streaming engine, as well as the integration of ML-Ops makes Hazelcast a unique database solution worth considering in the future.

References

- Ab architect's view of the hazelcast platform [Medium: White Paper]. (2023). Retrieved March 9, 2023, from https://hazelcast.com/thank-you/?resource_id=2006&form_page=https%3A%2F%2Fhazelcast.com%2Fresources%2Farchitects-view-hazelcast%2F
- Author, T., & Test, T. (1990). *Test title* (T. Editor, Ed.). Test Publisher.
- baeldung. (2016, October 21). *An introduction to hazelcast / baeldung*. Retrieved April 10, 2023, from <https://www.baeldung.com/java-hazelcast>
- Brewer, D. E. A. (2000a). Towards robust distributed systems.
- Brewer, D. E. A. (2000b). Towards robust distributed systems.
- Coda Hale. (2010, October 7). *You can't sacrifice partition tolerance* [Codahale.com]. Retrieved March 13, 2023, from <https://codahale.com//you-cant-sacrifice-partition-tolerance/>
- Fischer, J., & Lynch, A. (1985). Impossibility of distributed consensus with one faulty process.
- Haerder, T., & Reuter, A. (1983). Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15(4), 287–317. <https://doi.org/10.1145/289.291>
- Hale, C. (2010, October 7). *You can't sacrifice partition tolerance* [Codahale.com]. Retrieved April 10, 2023, from <https://codahale.com//you-cant-sacrifice-partition-tolerance/>
- Hazelcast. (n.d.-a). *Hazelcast topologies*. Retrieved April 12, 2023, from <https://docs.hazelcast.com/hazelcast/5.0/topologies>

- Hazelcast. (n.d.-b). *Kappa architecture* [Hazelcast]. Retrieved April 10, 2023, from <https://hazelcast.com/glossary/kappa-architecture/>
- Hazelcast. (2022a). *CAP theorem* [Hazelcast]. Retrieved April 10, 2023, from <https://hazelcast.com/glossary/cap-theorem/>
- Hazelcast. (2022b). *CP subsystem*. Retrieved April 10, 2023, from <https://docs.hazelcast.com/imdg/4.2/cp-subsystem/cp-subsystem>
- Hazelcast. (2022c). *Creating a transaction interface*. Retrieved April 10, 2023, from <https://docs.hazelcast.com/imdg/4.2/transactions/creating-a-transaction-interface>
- Hazelcast. (2022d). *Failure Detector Configuration*. Retrieved April 12, 2023, from <https://docs.hazelcast.com/imdg/4.2/clusters/failure-detector-configuration>
- Hazelcast. (2022e). *Feature comparison* [Hazelcast]. Retrieved April 10, 2023, from <https://hazelcast.com/product-features/feature-comparison/>
- Hazelcast. (2022f). *Getting Started with a Hazelcast Client*. Retrieved April 12, 2023, from <https://docs.hazelcast.com/hazelcast/latest/clients/hazelcast-clients>
- Hazelcast. (2022g). *Hazelcast*. Retrieved March 7, 2023, from <https://github.com/hazelcast>
- Hazelcast. (2022h). *Hazelcast - first application*. Retrieved April 10, 2023, from https://www.tutorialspoint.com/hazelcast/hazelcast_first_application.htm
- Hazelcast. (2022i). *Hazelcast IMDG - leading open source in-memory data grid • hazelcast* [Hazelcast]. Retrieved April 10, 2023, from <https://hazelcast.org/imdg/>
- Hazelcast. (2022j). *Hazelcast's replication algorithm*. Retrieved April 10, 2023, from <https://docs.hazelcast.com/hazelcast/5.0/consistency-and-replication/replication-algorithm>
- Hazelcast. (2022k). *Hazelcast/hazelcast* [GitHub]. Retrieved March 7, 2023, from <https://github.com/hazelcast/hazelcast>
- Hazelcast. (2022l). *HazelVision episode 06 - CAP theorem - YouTube*. Retrieved April 10, 2023, from https://www.youtube.com/watch?v=6DHxX_-fA8Y&list=PLhAaDrEJmCb3lpFQ6kDOkf_9wSdRp1cgx&index=10
- Hazelcast. (2022m). *Micro batch processing* [Hazelcast]. Retrieved April 10, 2023, from <https://hazelcast.com/glossary/micro-batch-processing/>
- Hazelcast. (2022n). *Overview of hazelcast distributed objects*. Retrieved April 10, 2023, from <https://docs.hazelcast.com/imdg/4.2/data-structures/overview#:~:text=Hazelcast%20has%20two%20types%20of%20distributed%20objects%20in,stores%20the%20whole%20instance%2C%20namely%20non-partitioned%20data%20structures.>
- Hazelcast. (2022o). *SQL*. Retrieved March 9, 2023, from <https://docs.hazelcast.com/hazelcast/latest/sql/sql-overview>
- Hazelcast. (2022p, November 3). *What is the hazelcast platform? | hazelcast explainer*. Retrieved April 10, 2023, from <https://www.youtube.com/watch?v=UpIgHzKbMp0>
- Hazelcast. (2023a). *Clients & languages* [Hazelcast]. Retrieved March 13, 2023, from <https://hazelcast.com/clients/>
- Hazelcast. (2023b). *Installing hazelcast open source*. Retrieved March 7, 2023, from <https://docs.hazelcast.com/hazelcast/latest/getting-started/install-hazelcast>
- Hazelcast. (2023c). *Start a local cluster in docker*. Retrieved April 8, 2023, from <https://docs.hazelcast.com/hazelcast/latest/getting-started/get-started-docker>

- Kreps, J. (2022). *Questioning the lambda architecture - o'reilly radar*. Retrieved April 10, 2023, from <http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.html>
- Mairhofer, K. (2021, April 14). Aufgabe SQL Buchausleihe.
- Ongaro, D., & Ousterhout, J. (n.d.). In search of an understandable consensus algorithm.
- Woos, D., Wilcox, J. R., Anton, S., Tatlock, Z., Ernst, M. D., & Anderson, T. (2016). Planning for change in a formal verification of the raft consensus protocol. *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, 154–165. <https://doi.org/10.1145/2854065.2854081>