

**Atypon Java and DevOps Bootcamp (Fall 2022)**

**Capstone Project**

**Decentralized Cluster-Based NoSQL DB System**

**Student Name : Diya Momani**

**Email :** [**diyamomani0@gmail.com**](mailto:diyamomani0@gmail.com)

**Github :** [**https://github.com/DiyaMomani**](https://github.com/DiyaMomani)

**Table of Contents**

**Introduction3**

What is the Decentralized Cluster-Based NoSQL DB? 3

What is the data model I used? 8

Why I used JSON files? 9

Schema9

What I used to build database?10

**Type chapter title (level 1)4**

Type chapter title (level 2)5

Type chapter title (level 3)6

**Type chapter title (level 1)4**

Type chapter title (level 2)5

Type chapter title (level 3)6

**Type chapter title (level 1)4**

Type chapter title (level 2)5

Type chapter title (level 3)6

**Type chapter title (level 1)4**

Type chapter title (level 2)5

Type chapter title (level 3)6

**Type chapter title (level 1)4**

Type chapter title (level 2)5

Type chapter title (level 3)6

**Type chapter title (level 1)4**

Type chapter title (level 2)5

Type chapter title (level 3)6

**Type chapter title (level 1)4**

Type chapter title (level 2)5

Type chapter title (level 3)6

**Type chapter title (level 1)4**

Type chapter title (level 2)5

Type chapter title (level 3)6

**Type chapter title (level 1)4**

Type chapter title (level 2)5

Type chapter title (level 3)6

**Introduction**

**What is the Decentralized Cluster-Based NoSQL DB?**

In this project I was asked to create Decentralized Cluster-Based NoSQL DB System, A decentralized cluster-based NoSQL database system is a distributed database system that operates on a peer-to-peer network architecture. It allows multiple nodes to share and store data without the need for a centralized authority. The system is designed to scale horizontally, meaning it can add nodes to the network as the data grows.

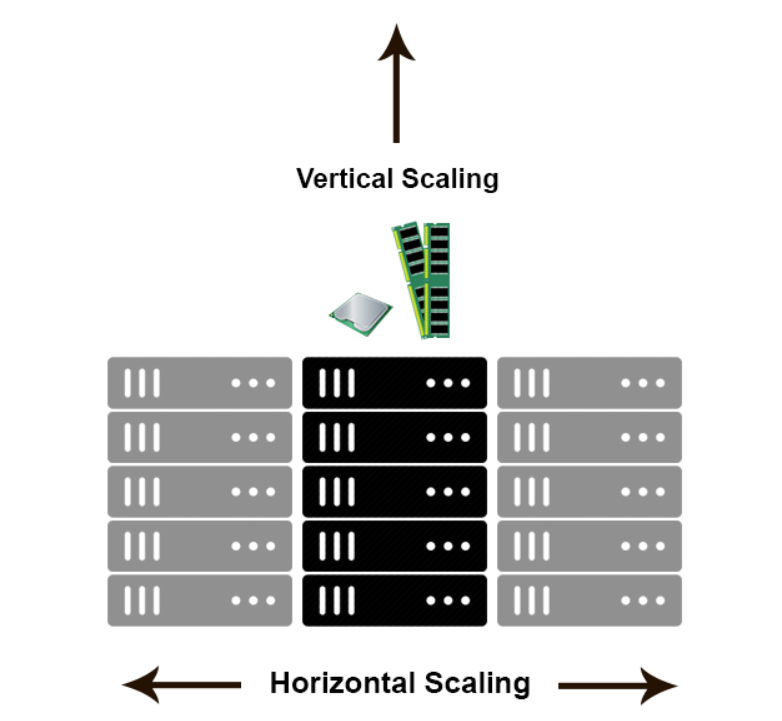


Figure-1 : shows the horizontal scale and how it is more scalable.

NoSQL, which stands for "not only SQL," is a type of database management system that doesn't rely on a traditional relational database structure. Instead, it uses a schemaless data model that allows for more flexible and dynamic data storage. NoSQL databases can store large amounts of unstructured data, making them ideal for big data applications.

NoSQL databases are better suited for applications that require scalability, flexibility, availability, performance, and cost-effectiveness, whereas SQL databases may be a better fit for applications that require strict adherence to a data schema and complex query operations.

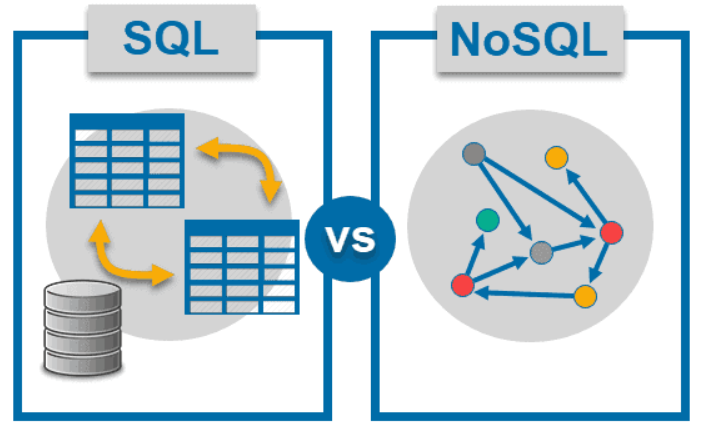


Figure-2 : shows NoSQL vs SQL databases .

In contrast to traditional centralized database systems, where data is stored on a single server, a decentralized cluster-based NoSQL database system allows data to be distributed across multiple nodes. This means that if one node fails, the other nodes can still access the data, ensuring high availability and fault tolerance.

The cluster-based NoSQL database system is also designed to be highly scalable. It can add new nodes to the network as needed to increase storage capacity and processing power. This allows the system to handle increasing amounts of data and growing user demand without any disruption to the service.

There are several types of decentralized cluster-based NoSQL database systems, each with its own unique architecture and features. Some examples include Cassandra, Riak, and Couchbase. And this is an explanation on Casandra database.

Diagram

Description automatically generated

Figure-3: shows the Casandra database.

A decentralized cluster-based NoSQL database system is a powerful tool for handling large amounts of unstructured data in a distributed environment. By using a peer-to-peer network architecture and a schemaless data model, these systems can provide high availability, fault tolerance, and scalability for modern applications. As the demand for big data solutions continues to grow, we can expect to see more businesses adopting these decentralized NoSQL database systems to meet their data storage and processing needs.

In this project I focused on completing all the requirements completely , I created 3 main Intellj projects : worker, bootstrap and demo. I created database called worker and used data structures to store, organize, and manipulate data efficiently, making it easier to access and process. I used multithreading and locks to improve performance, responsiveness, resource sharing and synchronization, and also I used hashing to efficiently store and retrieve data in a data structure. I focused on security by using tokens. Top of Formalso I made my code follows the Clean Code principles and solid principles .

**What is the data model I used?**

There is three types of NoSQL database :Key-Value store, Document store and Graph store, I used Document store database because It allows for a flexible data model, as data is stored in documents, which can contain nested and complex data structures, making it easier to store data as it is.



Figure-4: shows the document data model.

**Why I used JSON files?**

I used JSON files because they are Human-readable that means JSON is easy to read and understand for both humans and machines. Also JSON is a lightweight data format, which means that it requires less storage space compared to other data formats. JSON allows for a flexible data model, which is well-suited for document model NoSQL databases. It can represent complex data structures making it easier to store data as it is, without the need for data normalization. I used them to store the users, admins and schemas.

**Schema**

a schema is a logical data model that describes the organization and structure of a database. It can be thought of as a blueprint for a database. It provides a formal framework for organizing and storing data, and it specifies the rules that govern the relationships between the various elements of the database.

There is two types of schemas : strict and flexible , A strict schema is a data model that defines a fixed structure for data, with specific rules and constraints that must be followed. A strict schema is a data model that defines a fixed structure for data, with specific rules and constraints that must be followed.

In my project I gave the user the freedom to choose the type he want , he can enter the schema and store records depends on it , but in demo I allow the user to enter strict schema and the only condition of this schema that he should enter "Id" value.

**What I used to build database?**

I used **Java SE** as main programming language, Maven as build tool and dependency management, **Spring Boot** to create application .

**Database Implementation**

**Over View of system**

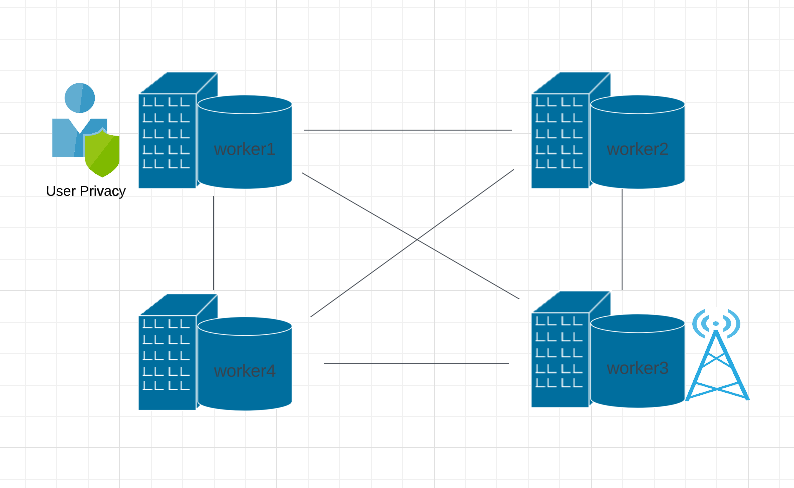
****

Figure-5: shows the over view of the system

I have four servers : worker1, worker2, worker3 and worker4, each of them has database/s to store records , each worker sends the records it receives to affinity node, and then affinity node will send it to the cluster, and from cluster to all other workers.

**The structure of database**

**The flow of data**

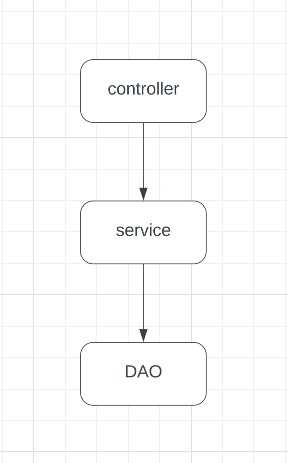
****

Figure-6: shows the general flow of data

Any request that will be send to server will follows this flow , first it will received by controller and send it to the affinity and check if the user or admin authorized , then send it to the cluster , and from cluster to all workers , and inside workers the controller will invoke the required function in service class, and this function will invoke the required function inside DAO class which will perform the changes required in database.

I will explain how to create database and the structure of it.

**Create database**

The user can create database by sending a request to the create Database URL, then the database will be created in this structure .

So, when creating database it will create database folder called with the name specified by the user , and create an empty folder called schemas inside it to store the schema JSON file/s that contain the schema/s.

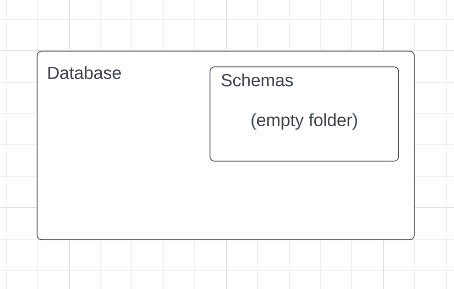


Figure-8: shows the structure of database before adding collection/s.

**Create collection**

The user can create database by sending a request to the create Collection URL, then the collection will be created in this structure .

So, when creating database it will create two empty JSON files called with the name specified by the user , one of them will be inside the schema folder and the another one out schema folder but inside database folder, and this is explanation of the structure.

