**Cloud Storage System**

Dworschak Richard 1200121

**Architecture:**

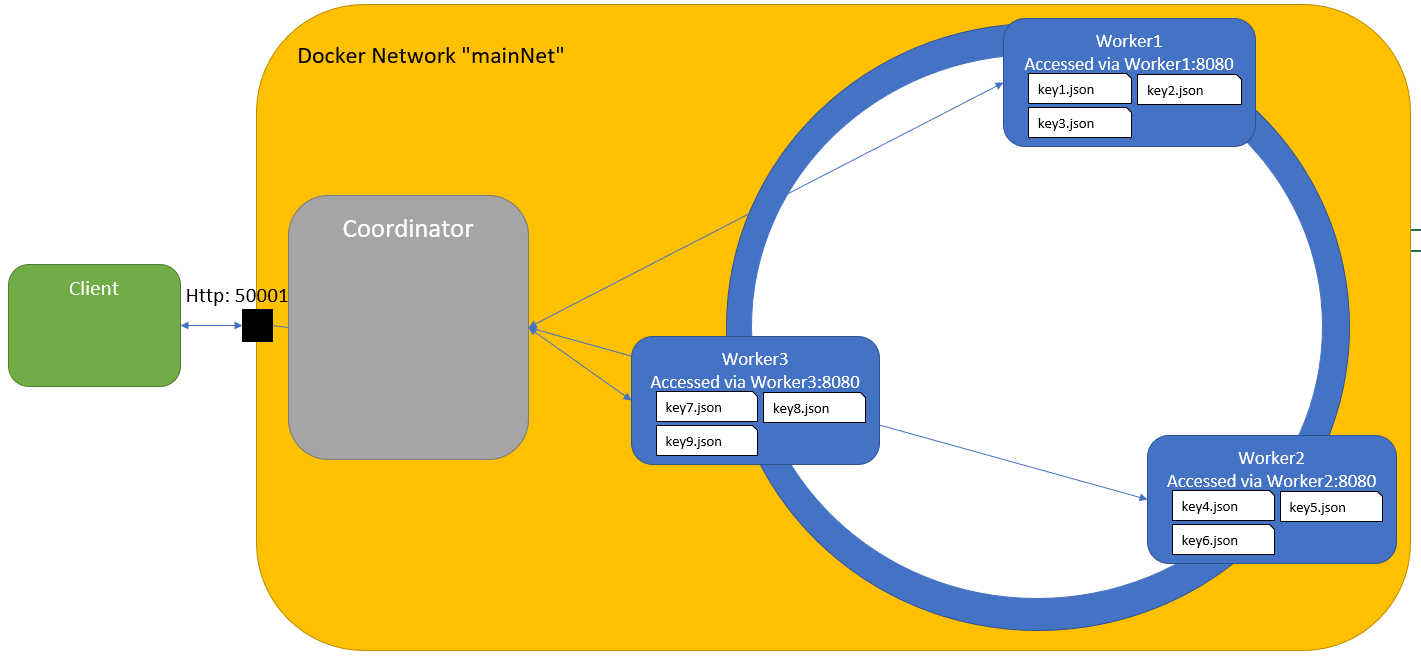
We decided on the Consistent Hashing for our System. We push Servers on a Ring like Structure and want to map the key-value pairs to the corresponding server, so it is distributed evenly. Check the Section “Hashing Method” on which Hashing Function we used and why

For our System we created three Parts:

1. Worker
2. Coordinator
3. Client

The Big Picture

The Coordinator functions as Master for our Consistent Hashing and is placed as Interceptor between the Client and the Workers. The Coordinator knows how many Workers are currently on the ring and where to redirect the Clients Request. The Workers don’t know each other and act on their own. The Workers can’t be reached from outside since all traffic needs to go through the Coordinator. For testing, the Workers are open to 10101 to 1010n where n is the number of workers on the Ring. We can access the Workers from the Client directly but only for testing.



Worker

The Worker represents the Bucket/Server of a Consistent Hashing Algorithm. It stores many files and can accept insert, delete and search Requests from outside. We decided to make the Worker a Docker image so we can create multiple containers to run separately. With that we can assure that each Bucket/Server runs on a different process and is isolated from all other Buckets/Servers. The downside of this decision is, that we can’t start the Cloud Storage System solely from the Client. Instead we start the Worker separately with a docker-compose or multiple docker runs.

Each Worker is assigned a Volume to work with which is mounted to the /shared directory. This volume represents the Bucket filled with files with a single key-value pair. The filename consists of the key of the corresponding value which is stored as JSON-Format inside the file. With that we can just check if a certain file (the key) exists within the Worker without having an in-memory array to load/persist on Worker start-up and shut-down.

The Worker itself hosts a small distribution of a tomcat server which serves on Port 8080. Each Method inside the Worker

Coordinator

Client

The Client is a Standalone Java Jar with dependencies build with Maven. Check the “How to Run” Section to see how you start it. The Client is a testing ground to check the functionality of the System. It is a console application with accepts simple user Inputs to send to the Coordinator. For a more advanced look at the client check the Section “How to Use the Client”

The Client uses multiple csv files to store a timestamp, the action used, the time it needed to compute and what the Coordinator returned from the Request. If you test single Methods the Client puts that into the file CSV\_Timestamp\_<currentTimeStamp> where the current Timestamp is when the main method is firstly called. With this we can have separate logs for multiple client restarts.

A second CSV file is generated and named by the user himself. In this File, the Client saves a predefined Test Method which automatically tests all the functions of the Coordinator/Worker and prints it to the corresponding CSV File

**Hashing Function**

**How to Setup:**

Worker:

The Worker run as Docker Containers on seperate processes to simulate a distributed System. To start up the System you need to follow these Steps

1. Create a Network named "mainNet" with docker network create mainNet
2. You can startup individual docker images by:
   1. Choosing an unique identifier for the Docker Image
   2. Create a volume with whatever you wanna call it (You can use the id in the name)
   3. Build the Docker Image with docker build -t RepoName directory (This can take awhile since its download the Maven Dependencys aswell
   4. Run the Docker file with docker run --rm -v Volume\_name:/shared --name ContainerName --network mainNet -p external\_port:8080 repoName

You can also use the Docker-compose file and start the containers. Before you use the docker compose you need to build the Worker and name the image worker. The docker-compose has 4 predefined Containers with volumes attached. The Volumes are created automatically if they not exist

Coordinator:

Client:

The Client is a Maven build executable Jar in Command Line Move to the Client Folder and package the jar with mvn clean package To execute the Jar you must at least have java 1.8 installed It creates a fat Jar that can be executed with: java -jar Client-jar-with-dependencies.jar

**How to Use the Client**

Start-up the Client described in the “How to Run” Section of this Report.

After Startup you are presented with a Menu an what you can do with the Client to Test the System. Some of the Methods require additional user input which is separated with ENTER.

“c”, import CSV:

Imports the Netflix.csv which is located in src/main/resources into the Client. Without that you cannot insert new Data to the Buckets. The Client saves all Records in an accessible Array.