Distributed Systems
Autumn Semester 2020
Dr. Valerio Schiavoni, Prof. Peter Kropf
Assistant: Isabelly Rocha

Project 3 — Part II
Distributed Redis Cluster

Part II Due: 22 December 2020, 23:59

1 Introduction

At this point we should already be familiar with Docker and Redis. Now, let's see how we could connect the Redis instances with a real application and investigate the trade-offs of different setups. For that, we will create a simple TODO List Spring Boot application. Spring Boot[*] is an open source Java-based framework used to create a micro Service. No implementation is needed as the application's source code is already given to you. However, if you are interested in more details, please refer to the official documentation indicated here.

2 Requirements

- This project can be executed and completed on the dedicated server machine that each of you was provided on our cluster.
- If you prefer to run on your personal machine, this is also possible. You only need docker and docker-compose installed.
- This work is individual (*i.e.*, no pairs, no groups).

Task 0 - Setup

You must be able to SSH into your server and enter into the `spring-boot-redis` directory. From a Linux-based system, issue the following command with the appropriate values for username and machine identifier:

[*]https://spring.io/projects/spring-boot

```
$ ssh [your-user-name]@[MACHINE_ID].maas
$ cp -r shared_public/spring-boot-redis/ spring-boot-redis/
$ cd spring-boot-redis/
$ ls
```

If you choose to do the project on your own machine, then you can either copy the files from the server or download them from ilias.

The `spring-boot-redis` directory contains the files needed to create the application as well as all the Docker and Redis related files. See the listing below showing these files.

```
Dockerfile
mvnw
mvnw.cmd
pom.xml
redis-conf
src
```

Under `src` you will find all the source files that you need. In the directories `redis-conf` and `scr/../resources` you find all the Docker related files, and in `src/../java` everything that is specific to the application.

Task 1 - Creating a Simple Spring Boot TODO List App

In this first task, let's start by creating a simple TODO list app with Spring Boot. To keep it simple, we will be creating this app for a single user only. In this first version we simply store one simple TODO list in the app as cache. No database is being used at the moment.

Listing 1 shows the class `TodoListController.java`. Have a look at this code to better understand the application which we are creating.

Listing 1: TodoListController.java

```
@RestController("/")
public class TodoListController {

        private ToDoList sample = new ToDoList(1, Arrays.asList(new ToDo(1,
            "Comment", false), new ToDo(2, "Clap", false),
                        new ToDo(3, "Follow Author", false)), false);

        @GetMapping
        public ToDoList getList() {
                return sample;
        }

        @GetMapping("/todo/{id}")
        public Optional<ToDo> getToDo(@PathVariable("id") long id) {
                return sample.getTasks().stream().filter(x -> x.getId() ==
                    id).findFirst();
        }

        @PutMapping("/todo")
        public ToDoList addToDo(@RequestBody ToDo toDo) {
                sample.getTasks().add(toDo);
                return sample;
        }

        @PostMapping("/todo/{id}")
        public ToDo updateToDo(@PathVariable("id") long id, @RequestBody
            ToDo toDo) throws Exception {
                Optional<ToDo> item = sample.getTasks().stream().filter(x
                    -> x.getId() == id).findFirst();
                if (item.isPresent()) {
                        item.get().setCompleted(toDo.isCompleted());
                        return item.get();
                } else {
                        throw new Exception("To Do item not found");
                }
        }
}
```

Now, let's connect this application with Redis. For that, first download and start the basic Redis Docker image using the commands shown below.

Listing 2: Download and start Redis Docker image.

```
$ docker pull redis
$ docker run --rm -p 4025:6379 -d --name redis-1 redis redis-server
```

In order to make the application talk to Redis we will use Spring Boot cache. Spring comes with several annotations that you can add to work Redis cache. Add `@EnableCaching` in your `Application.java` config to enable these annotations. See bellow.

Listing 3: Application.java

```
@EnableCaching
@SpringBootApplication
public class Application {

        public static void main(String[] args) {
                SpringApplication.run(Application.class, args);
        }

}
```

Now, to verify that the Spring Boot App can talk to Redis, let's build and run our docker app using the following commands.

Listing 4: Build and run docker app.

```
$ DOCKER_BUILDKIT=1 docker build -t learnings/spring-boot-redis-cluster .
$ docker run --rm -p 4024:4024 --name spring-boot-redis learnings/spring-
    boot-redis-cluster
```

After running the application, if you visit `http://localhost:4024/app/` you will get the error `ConnectTimeoutException:  connection timed out.`

Report the command's output and try to understand why this exception happens.

Note: if you are using one of the server machines, then you can use the following command to create a tunnel connection and after that you should be able to access the address given above in your local browser. The command assumes that you have your ssh config file properly configured (i.e., your public key and user are associated with this address).

Listing 5: Tunnel to connect with application running on remove machine.

```
$ ssh -L 4024:yahoocluster-x.maas:4024 ds2020@clusterinfo.unineuchatel.ch
```

Expected Task 1 results. For this task, you must report:

- The output of the commands in Listing 4.
- Explanation of the error encountered when trying to access the application.

Task 2 - Fixing Connection Timed Out Error

To fix the problem we encountered in Task 1, we will need to use Docker's network. By running the commands below, we can create a `spring-redis-network` and connect our Redis and Spring Boot images to it.

Listing 6: Create a new network and connect to it.

```
$ docker network create spring-redis-network
$ docker network connect spring-redis-network redis-1
```

Now, using the following command, we can look for the IP Address of our Redis instance.

Listing 7: Inspect the created network.

```
$ docker inspect spring-redis-network
```

Then, we can use the `IPv4Address` in the previous command's output to update the Redis host described in `application.yml`.

Note: we still use the port 6379 as we are in the docker network, exposed port to the host is 4025.

Finally, let's build and run our image for the application one more time with the commands described in Listing 4. However, this time add the parameter `--net spring-redis-network` to the `run` command in order to connect the application with the network we just created.

After that, if you try to access again the application at `http://localhost:4024/app/` then this time you should see the todo list items.

Please verify that the cache has been created in the Docker Redis image by running the following command.

Listing 8: Verify cache.

```
$ docker exec -it redis-1 redis-cli --scan
```

Expected Task 2 results. For this task, you must report:

- The list of items outputted by the application.
- The output of command in Listing 8.

Task 3 - Sharding with Redis Clusters

Until here we have built a basic spring boot application with Redis cache. But what if we are dealing with large data that can't be contained in one single node. As you already know, Redis supports clusters to shard your data across multiple nodes.

So let's stop our Redis image and build a cluster. For stopping the currently running container use the following command.

Listing 9: Stop Redis.

```
$ docker stop redis-1
```

Now let's spin two more Redis nodes to build a cluster. For that we also need to pass a Redis config file which is located in the `redis-conf` directory. Please remember to update your user if running in the server or the whole path if working locally.

Listing 10: Start the shards.

```
$ docker run --rm --net spring-redis-network -v /home/user_x/spring-boot-
    redis/redis-conf:/redis_config -p 4025:6379 -d --name redis-1 redis
    redis-server /redis_config/node1.conf

$ docker run --rm --net spring-redis-network -v /home/user_x/spring-boot-
    redis/redis-conf:/redis_config -p 4026:6379 -d --name redis-2 redis
    redis-server /redis_config/node2.conf

$ docker run --rm --net spring-redis-network -v /home/user_x/spring-boot-
    redis/redis-conf:/redis_config -p 4027:6379 -d --name redis-3 redis
    redis-server /redis_config/node3.conf
```

Now the Redis images have started in cluster mode, but we still need to create a cluster to bind them together. We can do a master slave configuration, but for now we just need data sharding and we are creating a cluster without failover.

For that, you need to first check the new IP Addresses using the command in Listing 7. Use the output to create a cluster by running the following command.

Listing 11: Create a redis cluster.

```
$ docker exec -it redis-1 redis-cli --cluster create <address1>:6379 <
    address2>:6379 <address3>:6379
```

The expected output here should be the hash slots allocation and a cluster check. Please report this to show that you successfully created the cluster.

After that, you can edit the `application.yml` file with the same nodes addresses which you used to create the cluster in the following format.

Listing 12: Redis section of application.yml.

```
redis:
  cluster:
    nodes:
      - <address1>:6379
      - <address2>:6379
      - <address3>:6379
    maxRedirects: 2
```

Then you can re-run the application and check that it's still working. For that, note that you first need to rebuild the Docker image and start the application using the commands in Listing 4. Remember to use the `--net` parameter to define the network we created.

Finally, the current cluster configuration can be verified with the following command from any of the 3 nodes.

Listing 13: Re-run the application.

```
docker exec -it redis-x redis-cli cluster nodes
```

Expected Task 3 results. For this task, you must report:

- The output of the command in Listing 11.
- The output of the command in Listing 13.

Task 4 - Do We Need Sentinels?

For this task, let's stat by converting our cluster to master/slave mode. Let's add 3 more nodes that will work as slave of our 3 master nodes. For that, we need to first stop the 3 currently running nodes and restart with the following commands.

Listing 14: Remove the running nodes and re-run the cluster in master/slave mode .

```
# Stop running containers
```

```
$ docker stop redis-1
$ docker stop redis-2
$ docker stop redis-3

# Start redis nodes
$ docker run --rm --net spring-redis-network -v /home/user_x/spring-boot-
    redis/redis-conf:/redis_config -p 4025:6379 -d --name redis-1 redis
    redis-server /redis_config/node1.conf

$ docker run --rm --net spring-redis-network -v /home/user_x/spring-boot-
    redis/redis-conf:/redis_config -p 4026:6379 -d --name redis-2 redis
    redis-server /redis_config/node2.conf

$ docker run --rm --net spring-redis-network -v /home/user_x/spring-boot-
    redis/redis-conf:/redis_config -p 4027:6379 -d --name redis-3 redis
    redis-server /redis_config/node3.conf

# Start slaves
$ docker run --rm --net spring-redis-network -v /home/user_x/spring-boot-
    redis/redis-conf:/redis_config -p 5025:6379 -d --name redis-1-slave
    redis redis-server /redis_config/node1-replica.conf

$ docker run --rm --net spring-redis-network -v /home/user_x/spring-boot-
    redis/redis-conf:/redis_config -p 5026:6379 -d --name redis-2-slave
    redis redis-server /redis_config/node2-replica.conf

$ docker run --rm --net spring-redis-network -v /home/user_x/spring-boot-
    redis/redis-conf:/redis_config -p 5027:6379 -d --name redis-3-slave
    redis redis-server /redis_config/node3-replica.conf
```

Now, just like before, let's inspect our docker network to find out the address we need to use to create the cluster with the following commands.

Listing 15: Check the IP Adresses and create the new cluster.

```
# Check the IP addresses of the nodes
$ docker inspect spring-redis-network

# Create the cluster with the IP addresses you just checked
$ docker exec -it redis-1 redis-cli --cluster create <address1>:6379 <
    address2>:6379 <address3>:6379 <address4>:6379 <address5>:6379 <address6
    >:6379 --cluster-replicas 1
```

If all goes well you should see a similar message with the hash slots allocation but this time on 6 nodes. Now, to test the application you need to modify the `application.yml` file again with the 3 master's IP addresses in the same format shown in Listing 12. If you build the application image and run it one more time using the command in Listing 4 (with the network parameter), then you can verify that the application is still working as before.

Finally, in order to check if you need sentinels or not, you can stop one of the masters with the `stop` command that you already know (Listing 9). Run the following command before and after doing so to verify your cluster and compare the two different outputs.

Listing 16: Check the IP adresses and create the new cluster.

```
$ docker exec -it redis-1 redis-cli cluster nodes
```

Report the output of the command in Listing 16 before and after you stop one of the master nodes. Verify and report if the application is still working after one of the master nodes is stopped. Reply the question whether or not we need sentinels in this setup and explain your answer.

Hint: If you stop one of the master nodes and it is not automatically restored by the cluster, this might indicate that sentinels are necessary. Otherwise, try to compare with the scenario in *Project 3 - Part I* where you created a Sentinel Redis Cluster. Discuss the differences/similarities that you can observe between these two setups (e.g., which scenario or application is more appropriated for one or another, etc).

Expected Task 4 results. For this task, you must report:

- The output of command in Listing 16 before and after stopping one of the master nodes.
- Report whether or not the application is still working after the node stops and comment the observed behavior (e.g., is that what you expected, etc.).
- Answer whether or not sentinels are needed in this setup together with an explanation and discussion about the topic.

Submission

The work is individual. No exceptions. The following documents must be uploaded on ILIAS:

1. A pdf version of your report. Each student has to implement his/her own version of the report. This report should contain all your interesting results (in table and/or plot formats), as well as a pertinent analysis and evaluation.
2. A zip file containing all the files you produced (PDF report, xml files, etc.)

Please respect the following point:

- Make sure that your compressed document is of reasonable size (no need for images of high-quality) and upload it on ILIAS.

Note that the fact that you strictly followed all the instructions concerning the way to provide your report and results (formatting, type of file, deadline,...) will be taken into account to establish your mark.