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

Project 3 — Part I
Distributed Redis Cluster

Part 1 Due: 22 December 2020, 23:59

1 Introduction

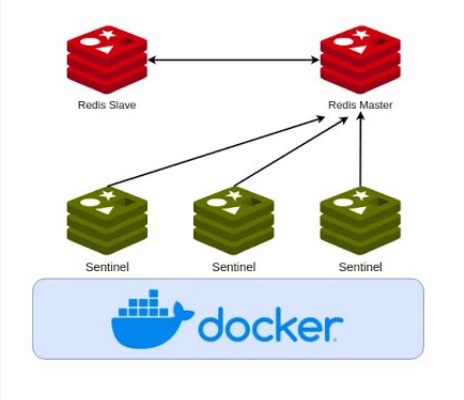

Figure 1: Distributed Sentinel Redis Cluster on top of Docker.

Redis Cluster* provides a way to run a Redis installation where data is automatically shared across multiple Redis nodes. Redis Cluster also provides some degree of availability during partitions, that is in practical terms the ability to continue the operations when some nodes fail or are not able to communicate. However the cluster stops to operate in the event of larger failures (for example when the majority of masters are unavailable).

To mitigate this problem, one option is to use Redis Sentinel† which provides high availability for Redis. In practical terms this means that using Sentinel you can create a Redis deployment that resists without human intervention certain kinds of failures. Redis Sentinel also provides other collateral tasks such as monitoring, notifications and acts as a configuration provider for clients.

Please refer to the official Redis documentation for more details.

*https://redis.io/topics/cluster-tutorial
†https://redis.io/topics/sentinel

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 `redis-cluster` directory. From a Linux-based system, issue the following command with the appropriate values for username and machine identifier:

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

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

The `redis-cluster/` directory contains the file `docker-compose.yml` and a directory named `sentinel` containing the files needed for the sentinel nodes. See the listing below showing these files.

```
docker-compose.yml
sentinel/Dockerfile
sentinel/entrypoint.sh
sentinel/sentinel.conf
```

Inspect the `docker-compose.yml` file. Note that we have 3 types defined there: master, slave and sentinel. In the previous project, this Docker configuration files ware automatically generated for you from the `.xml` file of the topology you defined. Now that you are using it directly, try to understand the structure of it. Refer to the Docker documentation for more details [‡].

Listing 1: Content of docker-compose.yml

[‡]https://docs.docker.com/compose

```
master:
  image: redis:3
slave:
  image: redis:3
  command: redis-server --slaveof redis-master 6379
  links:
    - master:redis-master
sentinel:
  build: sentinel
  environment:
    - SENTINEL_DOWN_AFTER=5000
    - SENTINEL_FAILOVER=5000
  links:
    - master:redis-master
```

The example deploys a redis cluster with a master node, a single slave and a single sentinel. Note that the master and slave are using the image `redis:3` which will automatically be downloaded if you don't have it in your local repository yet.

Task 1 - Creating Redis Cluster

In this task you should use the files provided to you and described above to create an initial Redis Cluster with a single master and worker.

To deploy the initial redis cluster you can use the following command.

Listing 2: Start redis cluster.

```
docker-compose up --build -d
```

To verify if your deployment is working you can check the list of running containers by running the following command.

Listing 3: List the running Docker containers.

```
docker ps
```

The expected output is a list of 3 running containers named redis-cluster-with-sentinel_sentinel_1, redis-cluster-with-sentinel_slave_1 and redis-cluster-with-sentinel_master_1. All 3 should be in the `up` state when you run the command above.

To clean up Docker and remove all the running containers, you can use the following command.

Listing 4: Clean up Docker containers.

```
1 docker rm -f $(docker ps -a -q)
```

For the next tasks you will need this deployment, so you don't need to use this command now. This is just to inform you how to clean up the environment once you're done or if you would like to start over for some reason.

Note: the command on Listing 4 will delete **all** the running containers that you have at the moment you run the command. Be aware of that in case you have other Docker containers running which are not related to this project.

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

- The output of the command on Listing 3 showing that the cluster has been deployed.

Task 2 - Redis Benchmark

In this task you should run the *GET* and *SET* operations of the redis-benchmark to evaluate the performance of our current cluster.

For that, you first need to find the address of your master node which can be done by running the following command.

t

Listing 5: Find out master node address.

```
1 docker-compose exec sentinel redis-cli -p 26379 SENTINEL get-master-addr-by
    -name mymaster
```

In this command, *mymaster* refers to the registered name of the master service on sentinel. Check the file *sentinel/sentinel.conf* for more details on this. The command above should output 1) address and 2) port.

Use the master information outputted from the command in Listing 5 as parameter to the redis-benchmark as showed below.

Listing 6: Run redis-benchmark.

```
redis-benchmark -t set,get -h <masteraddress>
```

Now, use the parameters *-n* and *-d* to vary the number of requests and data size of `GET` and `SET` value in bytes, respectively. Plot two graphs showing how the duration of these benchmarks vary once these parameters are increased. Try to choose values which show a relevant outcome and explain your results.

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

- The output of the command on Listing 6 showing the redis-benchmark with the default total number of requests (i.e., 100000).
- A plot where x is the number of requests and y the time it took to complete.
- A plot where x is the data size and y the time it took to complete.
- At least a paragraph discussing the differences observed when increasing the number of requests and data size.

Task 3 - Scale-Up Cluster

In this task you should scale the cluster up. In order to do that, you can simply make use of the docker *scale* command. Refer to Listing 7 and Listing 8 below to see how this command can be used in this specific scenario.

Listing 7: Scale the number of slaves.

```
docker-compose scale slave=4
```

Listing 8: Create new sentinels.

```
docker-compose scale sentinel=3
```

Now, if we run again the command described in Listing 3, you should see 6 containers running. All of them in the `up` status. Please report the output of this command for the new cluster.

Moreover, if we re-run the command in Listing 6 we now get the benchmark results of a cluster consisting of 4 slaves instead of one single slave. First, do this test with the default values

and observe if/how this benchmark differs from the previous cluster setup. Report and explain your observations.

Then, use the parameters *-n* and *-d* in the same way you chose to do for Task 2. See if the observations you just made also holds for these different values. Report your results in a form of a plot and comment your observetions.

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

- The output of the command on Listing 3 showing that your cluster is scaled up.
- The output of he command on Listing 6 showing the redis-benchmark with the default total number of requests (i.e. 100000).
- A plot where x is the number of requests and y the time it took to complete.
- A plot where x is the data size and y the time it took to complete.
- At least a paragraph discussing the differences observed between running these benchmarks on the previous cluster and on the scaled up cluster.

Task 4 - Full Benchmark

In this task you should do a more complete evaluation of the redis distributed setup.

Now that you are familiar with how to scale up and down a redis cluster as well as how to use the available benchmarks, let's make a more complete analysis of the cluster.

In this task, based on the commands you used so far, you should further scale the cluster and perform more tests than the GET and SET we used so far.

To report your results, you don't need to submit the command's outputs but instead produce plots which will make the visualization more clear and highlight the most interesting findings you made.

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

- A plot (or several) where the x-axis is the number of slaves and the y-axis is the metric of interest of the chosen tests.
- Explanations of the observations.

Task 5 - Sentinels

In this task you should test the sentinel service. For that, let us consider the setup with 4 slaves and 3 sentinels.

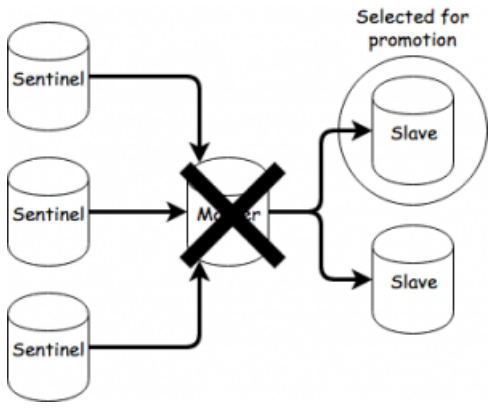

Figure 2: Sentinel service illustrated on the scenario where the master encounters a problem.

If the master service goes down:

1. Each "sentinel" service verifies that master is down. When specified number of "sentinel" service (quorum) verifies that master is down, goes next step.
2. Sentinels will wait that the master service goes up again on the specified time. "Automatic Failover" starts if master doesn't go to up. Sentinel services select from one of slaves as master by election.
3. Finally selected master service configuration changes are distributed to all "redis" services.

Remember that the command in Listing 5 shows the master service ip and master service port. Before performing the changes below, run it to see what is the current master information.

Now let's try to simulate a scenario where the master is not working by pausing it. This can be done using the following command.

Listing 9: Pause master node.

```
1 docker-compose pause master
```

If you want to verify that the master is indeed paused, you can run the command from Listing 3 and you should see that he container corresponding to the master is still listed but now in state

paused. Once this is done, re-run the command in Listing 5 and compare its output with the one from before performing the changes.

To get more details information on what happens once the master encounters a problem, you can have a look in the logs. For that, use the following command.

Listing 10: Get sentinels logs.

```
1 docker-compose logs sentinel
```

Investigate the logs output and explain the behavior you observe.

Once you are done and want to bring the cluster back to the regular state, you can run the following command.

Listing 11: Unpause master node.

```
1 docker-compose unpause master
```

After running this command, verify that all the containers are up and running with the command from Listing 3 again.

Expected Task 5 results. For this task, you must report:

- The output of Listing 5 before and after pausing the master.
- The output of Listing 10.
- Explanation of the observed behavior.

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.