POLYTECHNIQUE MONTRÉAL

LOG8415E

ADVANCED CONCEPTS OF CLOUD COMPUTING

Project
Scaling Databases and Implementing Cloud Patterns

Authors:
2009913 - Jordan Mimeault

Lab Instructor:
Vahid Majdinasab

Instructor:
Amin Nikanjam

https://github.com/JordMim/log8415e-project.git

decembre 23, 2022

Contents

1 How my implementation works. 2

2 Benchmarking MySQL standalone vs cluster (1 master / 3 slaves) 3

3 Implementation of the Proxy pattern 4

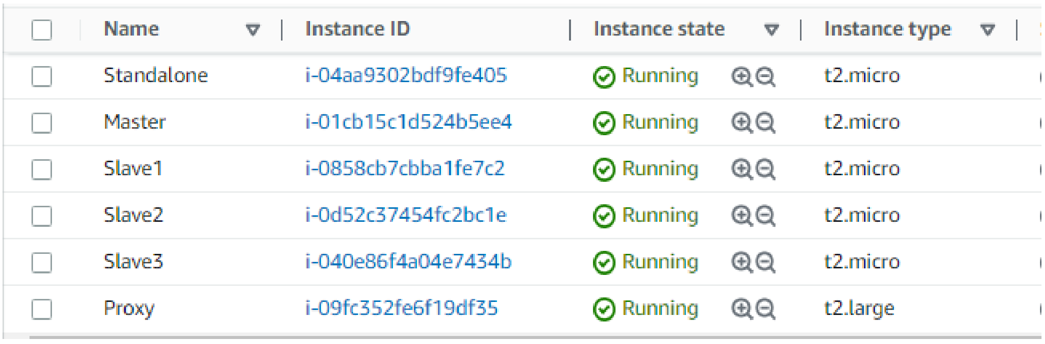
4 Instructions to run the code 5

1 How my implementation works.

For the standalone instance, I created a t2.micro instance with a newly created ssh key with AWS that I stored in my git repo. This key will allow me to connect to this instance by ssh in the future. After creating this instance, I followed the tutorial for initializing the instance correctly. After installing the sakilaDB step, I performed the benchmark with sysbench.

For the cluster, I have created 4 t2.micro instances of which one is the master and the others are the slaves. I created a specific security group for this type of instance to allow the traffic between them on port 1186 (used by ndb_mgm). I followed the tutorial on how to setup a MySQL cluster with multiple nodes. After setting up my 4 instances, I launched my master and connect the 4 slaves to it using the public dns address. After connecting all my slaves to my master, I performed the benchmark on the master node.

For the proxy, I created a t2.large instance with the same security group as the master/slaves instances. I didn't have time to finish my implementation. I created a flask_app for redirecting to the correct route (direct, random and custom) and performing the query on the chosen node. I had a problem with creating a tunnel between the master node and the proxy instance so I didn't finish the complete implementation. But I was able to send requests between Postman and my proxy instance.



Name	Instance ID	Instance state	Instance type
Standalone	i-04aa9302bdf9fe405	Running	t2.micro
Master	i-01cb15c1d524b5ee4	Running	t2.micro
Slave1	i-0858cb7cbba1fe7c2	Running	t2.micro
Slave2	i-0d52c37454fc2bc1e	Running	t2.micro
Slave3	i-040e86f4a04e7434b	Running	t2.micro
Proxy	i-09fc352fe6f19df35	Running	t2.large

2 Benchmarking MySQL standalone vs cluster (1 master / 3 slaves)

For benchmarking the MySQL standalone instance, I used sysbench with 6 threads. After running the tests, the results are the following:

```
SQL statistics:
queries performed:
read: 190414
write: 54388
other: 27196
total: 271998
transactions: 13595 (226.48 per sec.)
queries: 271998 (4531.30 per sec.)
ignored errors: 6 (0.10 per sec.)
reconnects: 0 (0.00 per sec.)

General statistics:
total time: 60.0244s
total number of events: 13595

Latency (ms):
min: 8.29
avg: 26.48
max: 117.91
95th percentile: 34.33
sum: 360051.76

Threads fairness:
events (avg/stddev): 2265.8333/7.20
execution time (avg/stddev): 60.0086/0.01
```

After benchmarking the standalone instance, I used the same technique for testing the cluster. The obtained results are the following:

```
SQL statistics:
queries performed:
read: 169232
write: 48341
other: 24171
total: 241744
transactions: 12083 (201.34 per sec.)
queries: 241744 (4028.14 per sec.)
ignored errors: 5 (0.08 per sec.)
reconnects: 0 (0.00 per sec.)

General statistics:
total time: 60.0117s
total number of events: 12083

Latency (ms):
min: 10.92
avg: 29.79
max: 187.19
95th percentile: 44.17
sum: 360000.67

Threads fairness:
events (avg/stddev): 2013.8333/4.06
execution time (avg/stddev): 60.0001/0.00
```

As you can see, the standalone server performed more read, write and other queries than the cluster. In other words, the standalone performed better than the cluster, which is very strange considering that the cluster can divide tasks between the slaves for a better performance.

3 Implementation of the Proxy pattern

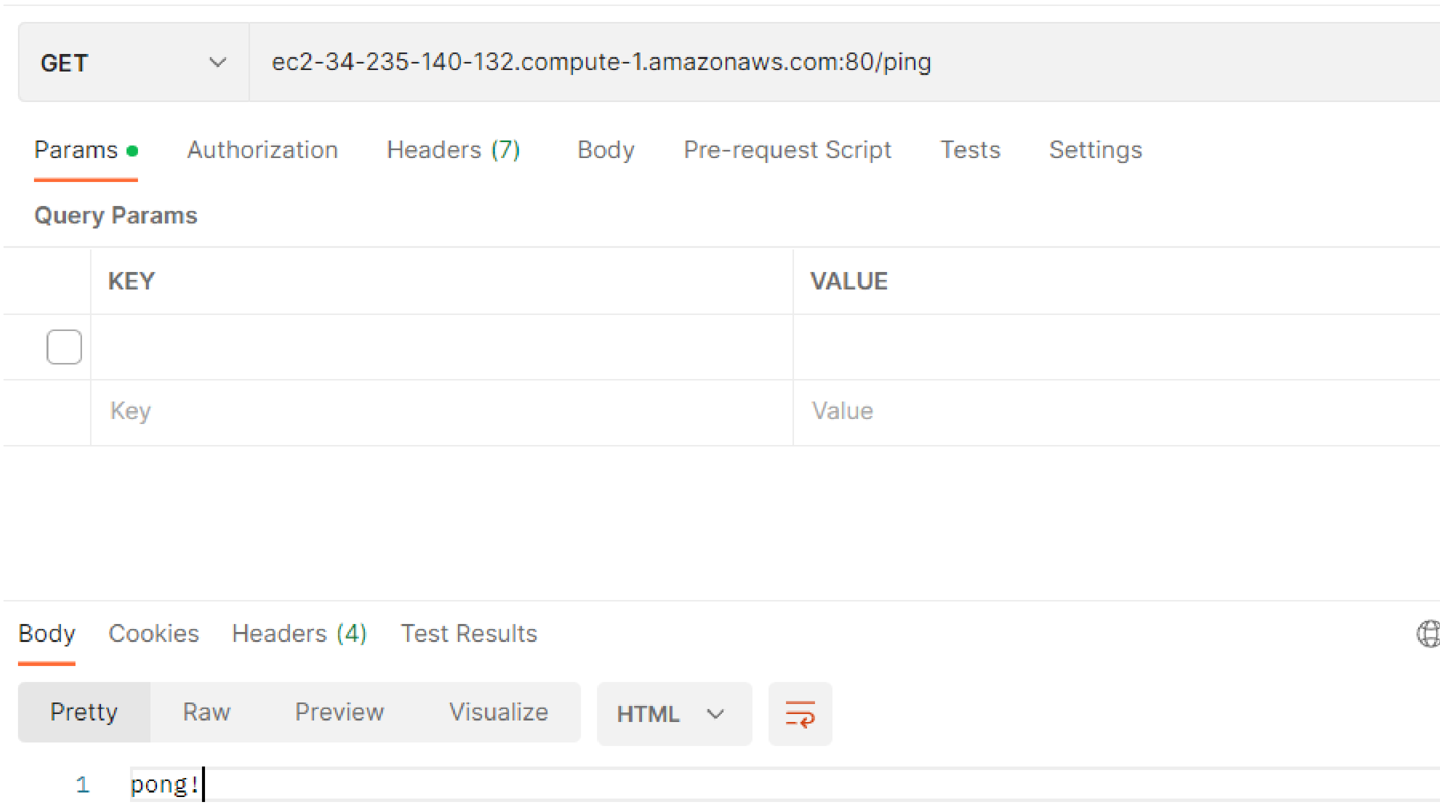
As mentionned in the first part of the report, the proxy pattern was not finished. All I did was creating a flask app that is on my github. The proxy instance used git clone to fetch the app and run it. After running the app on the proxy instance, I was able to send request using POSTMAN. An exemple of request is :

ec2-34-235-140-132.compute-1.amazonaws.com/direct.

This request will hit the proxy /direct endpoint and will performed the querying on the right node between the master and the slaves. Unfortunately, the tunnel created by openSSH was not correctly created so I didn't have result to show.

4 Instructions to run the code

To run the code, you need to connect to the proxy instance and git clone my github. After cloning the repo, go in the ”log8415e-project_app” folder and run ”app.py”. After running, the app, the proxy is now activated and you can send request with a requests sender app like Postman.
Request example:



```
ec2-34-235-140-132.compute-1.amazonaws.com:80/ping
```

```
pong!
```