École Polytechnique de Montréal

Final assignment

Work done by Quoc Tuan Pham

LOG8415E - Advanced Concepts of Cloud Computing

Group 01

Work presented to

Vahid Majdinasab

Polytechnique Montréal

Submitted on December 27th, 2023

Table of Contents

GitHub link : https://github.com/tuanquocpham123456/log8415e/tree/main

[Part 1] Demo of my results: https://youtu.be/JhSiSl5i_OE

[Part 2] Code explanation: https://youtu.be/jepVk-Tn0rk

Benchmarking MySQL stand-alone vs. MySQL Cluster

For the setup, I mostly followed this tutorial in order to configure the MySQL cluster : https://www.digitalocean.com/community/tutorials/how-to-create-a-multi-node-mysql-cluster-on-ubuntu-18-04

You will be able to find inside the manager.sh and worker.sh all the necessary commands in order to setup everything.

For the benchmark report, the MySQL Standalone setup involves preparing a database with 100,000 records using the default storage engine, executing a mixed workload benchmark with 6 threads for 60 seconds, and finally cleaning up the database. On the other hand, the MySQL Cluster setup prepares the Cluster with the NDB storage engine, conducts a similar benchmark, and performs cleanup. Both setups use Sysbench to simulate OLTP read/write workloads. Here are the results:

Metric	**MySQL Standalone**	**MySQL Cluster**
Total Transactions	**16262**	9443
Transactions per Second	**270.95/s**	157.31/s
Total Queries	325240	188860
Queries per Second	**5418.97/s**	3146.13/s
Ignored Errors	0	0
Reconnects	0	0
Total Time	60.0166s	60.0271s
Events (avg/stddev)	2710.3333/4.53	1573.8333/5.98
Execution Time (avg/stddev)	60.0039/0.00	60.0055/0.01
Latency (ms)		
- Min	7.52	23.52
- Avg	**22.14**	38.13
- Max	115.32	1505.47
- 95th Percentile	32.53	49.21
- Sum	360023.23	360033.14

Table #1 : Read and write benchmark

Comparison:

- In terms of total transactions, the standalone setup performed better with 16,262 compared to 9,443 in the cluster.
- The transactions per second metric also favors the standalone configuration, with 270.95/s as opposed to 157.31/s in the cluster.

- However, the cluster demonstrates a higher average latency (38.13 ms) compared to the standalone (22.14 ms), suggesting a longer response time on average.
- The queries per second metric favors the standalone setup with 5,418.97/s, while the cluster achieved 3,146.13/s.
- Both configurations exhibited no ignored errors or reconnects, indicating stability during the benchmark.
- The total time for both setups is almost identical at around 60 seconds.

While the standalone setup outperforms in terms of transaction volume and speed, the cluster demonstrates competitive performance with slightly higher latency. The choice between the two may depend on specific workload requirements and the trade-off between transaction throughput and response time. Further analysis and testing may be necessary to determine the most suitable configuration for the intended use case. For our case, it does seem strange that the MySQL Standalone mostly outperforms the MySQL cluster since the cluster should be able to distributes its load effectively among its nodes to be more efficient. Maybe a different test scenario will have the cluster outperform the standalone one since our test scenario was with 100 000 queries since with 1 000 000 the test could not be run since the table will be full and we will get this error:

FATAL: mysql_stmt_execute() returned error 1114 (The table 'sbtest1' is full) for query 'INSERT INTO sbtest1 (id, k, c, pad) VALUES (?, ?, ?, ?)'

FATAL: `thread_run' function failed: /usr/share/sysbench/oltp_common.lua:488: SQL error, errno = 1114, state = 'HY000': The table 'sbtest1' is full

Implementation of The Proxy pattern

In my project, I implemented the Proxy pattern using Python and MySQL. The Proxy pattern provides a surrogate or placeholder for another object to control access to it. In this case, the proxy is a Python script that forwards requests to a MySQL database cluster. I used Terraform to set up the infrastructure, which includes the proxy instance and the MySQL database cluster. The Terraform configuration is in the **log8415e/sql** folder.

I set up a MySQL cluster that includes a manager node and several worker nodes. The setup scripts for the manager and worker nodes are **manager.sh** and **worker.sh**, respectively. The manager node manages the cluster, and the worker nodes store the data. I set up a proxy instance that runs a Python script. The setup script for the proxy instance is **proxy.sh**. The Python script **proxy.py** runs on the proxy instance and forwards requests to the MySQL database cluster. The Python script **proxy.py** listens for HTTP requests. When it receives a request, it forwards the request to the MySQL database cluster. The script uses the PyMySQL library to connect to the MySQL database and execute SQL queries.

I used Postman to send HTTP requests to the Python script running on the gatekeeper instance. The body of the HTTP request contains an SQL query, which the Python script forwards to the MySQL database cluster.

Implementation of The Gatekeeper pattern

In my project, I also implemented the Gatekeeper pattern, which is a variant of the Proxy pattern. The Gatekeeper pattern provides a surrogate or placeholder for another object to control access to it, similar to the Proxy pattern. However, the Gatekeeper adds an additional layer of complexity by implementing a strategy to determine how to forward requests.

After setting up the infrastructure and MySQL cluster as described in the Proxy pattern implementation, I proceeded with the following steps:

1. Setup the Gatekeeper: I set up a gatekeeper instance that runs a Python script. The setup script for the gatekeeper instance is **gatekeeper.sh**. The Python script **gatekeeper.py** runs on the gatekeeper instance and forwards requests to the MySQL database cluster based on a specified strategy.

2. Implement the Strategy: In the **gatekeeper.py** script, I created a Facade for three strategies for forwarding requests: direct hit, random, and customized. The strategy is specified in the HTTP request sent to the gatekeeper. Inside the **proxy.py** script is where those strategies are really implemented.

3. Run the Gatekeeper: The Python script **gatekeeper.py** listens for HTTP requests. When it receives a request, it extracts the strategy from the request parameters and forwards the request to the MySQL database cluster based on the chosen strategy. The script uses the PyMySQL library to connect to the MySQL database with a SSHTunnelForwarder and execute SQL queries.

4. Send Requests to the Gatekeeper: I used Postman to send HTTP requests to the Python script running on the gatekeeper instance. The body of the HTTP request contains an SQL query, which the Python script forwards to the MySQL database cluster based on the specified strategy. The body also contains a code which represents which strategy to use for: 1 for direct hit, 2 for random and 3 for customized.

Describe clearly how your implementation works.

When everything is setup, we open our Postman app and do our HTTP request like the one in the Figure 3. Then after each of our request, we can either see our result from the Postman console or from the gatekeeper or proxy view console seen from Figure 1 and 2 respectively.

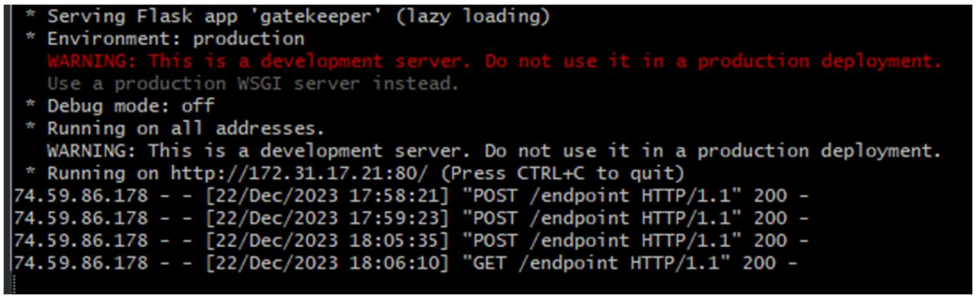

Figure 1 Gatekeeper view

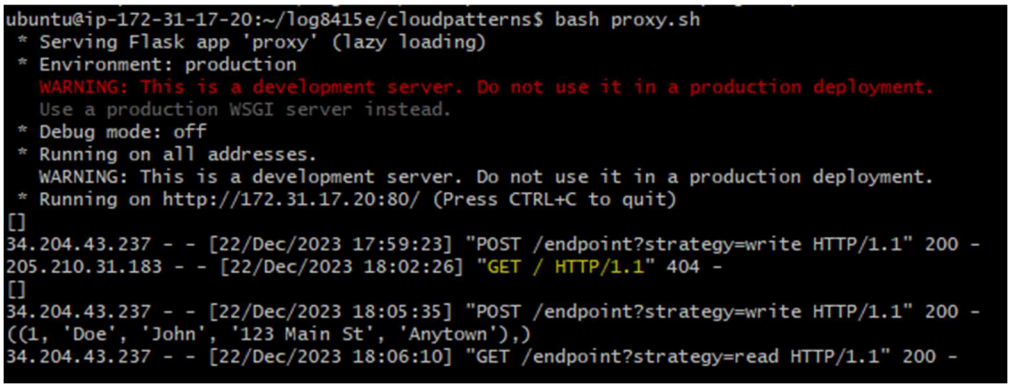

Figure 2 Proxy view

Instructions to run your code and summary of my results

Instructions to run the code

[Part 1] Demo of my results: https://youtu.be/JhSiSl5i_OE

[Part 2] Code explanation: https://youtu.be/jepVk-Tn0rk

Terraform

In order to lunch the infrastructure with all the instances, you need to run terraform apply inside the log8415e/sql folder.

MySQL

Inside all the next instances, after connecting via ssh, you will need to clone the repository before doing the rest of the commands:

git clone https://github.com/tuanquocpham123456/log8415e.git

cd log8415e/sql

MySQL Standalone

bash standalone.sh

It will install Sakila, populate the database structure and run the Sysbench benchmark.

MySQL Manager

bash manager.sh

bash manager_sysbench.sh

The first command will setup the MySQL cluster manager. The second command will install Sakila, populate the database structure and run the Sysbench benchmark.

After creating the proxy instance, run these commands in order to create a user associated with the proxy instance:

sudo mysql

mysql> CREATE USER 'ubuntu'@'proxy_public_ip_adress' IDENTIFIED BY 'admin';

mysql> GRANT ALL PRIVILEGES ON . TO 'ubuntu'@'proxy_public_ip_adress' WITH GRANT OPTION;

mysql> FLUSH PRIVILEGES;

mysql> exit

On all the MySQL workers

bash worker.sh

This will set up each worker and connect it to the MySQL cluster manager.

Gatekeeper pattern

In order to run the gatekeeper pattern, you need to run the following commands inside the gatekeeper instance after connecting via ssh:

git clone https://github.com/tuanquocpham123456/log8415e.git

cd log8415e/cloudpatterns

bash init.sh

bash gatekeeper.sh

Proxy pattern

In order to run the proxy pattern, you need to run the following commands inside the proxy instance after connecting via ssh:

git clone https://github.com/tuanquocpham123456/log8415e.git

cd log8415e/cloudpatterns

bash init.sh

bash proxy.sh

SQL queries

Open the Postman app in order to run the SQL queries as a body of a POST or GET request with the gatekeeper public ip adress. Don't forget to add a code inside a form-data which represents :

1 for a direct hit

2 for random

3 for customized

Here is an example of a POST request with the body of the query and the result of a select query inside Postman:

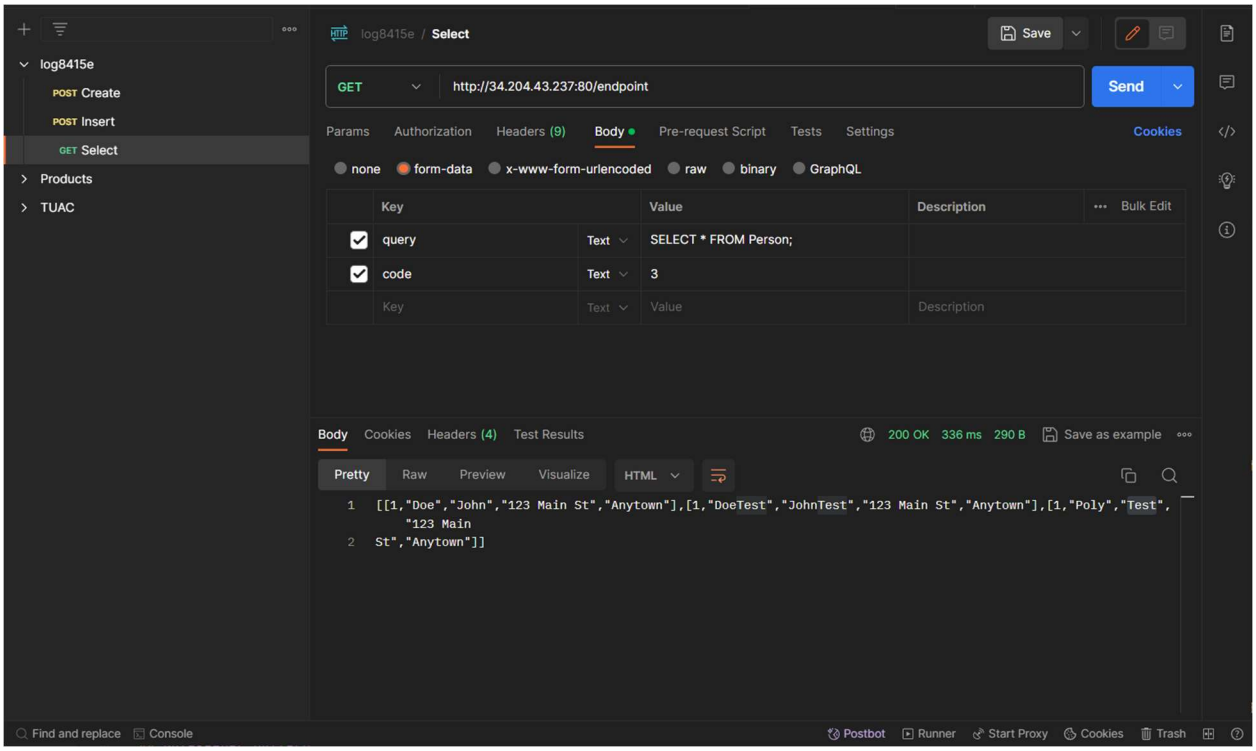

Figure 3 Postman query

Summary

In my project, I implemented two cloud design patterns: the Proxy and Gatekeeper patterns, using Python, MySQL, and Terraform.

For the Proxy pattern, I created a Python script that acts as a proxy for a MySQL database cluster. The script receives HTTP requests, forwards them to the MySQL database cluster, and returns the results to the client.

For the Gatekeeper pattern, I extended the Proxy pattern by implementing a strategy in the Python script to determine how to forward requests. The script receives HTTP requests, determines the strategy for forwarding the request based on the user choice, forwards the request to the MySQL database cluster, and returns the results to the client.

The infrastructure setup, including the proxy and gatekeeper instances and the MySQL database cluster, was managed using Terraform and with some bash scripts. I used Postman to send HTTP requests to the gatekeeper instance. The body of the HTTP request contains an SQL query, which the Python scripts forward to the MySQL database cluster based on the specified strategy.