LOG8415E - Advanced Concepts of Cloud Computing

Final Project – Scaling Databases and Implementing Cloud Design Patterns

Report

Nam VU (2230468)

December 25, 2023
Polytechnique Montréal – Fall 2023

Contents

Handover Documentation Report

Project Title

Scaling Databases and Implementing Cloud Design Patterns

Report Date

December 25, 2023

Team Members

- Nam Vu (2230468)

Executive Summary

In this last lab assignment, we are asked to setup a MySQL cluster on Amazon EC2 and implement two Cloud patterns: Proxy and Gatekeeper. We reused and extended the codebase from the previous projects to setup and configure the AWS infrastructure programmatically using Python and boto3. Details of the implementation and benchmarking results are presented in the following sections.

---

1 Introduction

Deploying a database on the cloud is a common task in the industry. A cluster deployment is often preferred over a standalone server for better performance and availability. In our setup, the implementation of two Cloud patterns, Proxy and Gatekeeper, offers additional performance and security benefits.

2 Project Objectives

Different objectives are fixed for this project:

- Deploy MySQL on Amazon EC2 in two different configurations: standalone server and cluster.
- Benchmark the performance of the two deployments with sysbench.
- Implement two Cloud patterns: Proxy and Gatekeeper.

3 Approach

As with the other assignments, we use Python and boto3 to setup and configure the AWS infrastructure. In addition, we also rely this time on Jinja2 to template the diverse shell scripts used to setup the Ubuntu 22.04 EC2 instances depending on their roles. All Python dependencies are managed with Poetry and the project is version-controlled with Git.

4 Tasks and Responsibilities

Nam Vu

- MySQL deployments
- Benchmarking
- Patterns implementation

5 Progress and Achievements

All architectures can be easily deployed following the instructions in the `README.md` file of the project repository.

5.1 MySQL deployments

In this section we deploy MySQL in two different configurations: standalone server and cluster. We then also load the `sakila` sample database into both deployments.

5.1.1 Standalone server

Directory: `src/standalone`

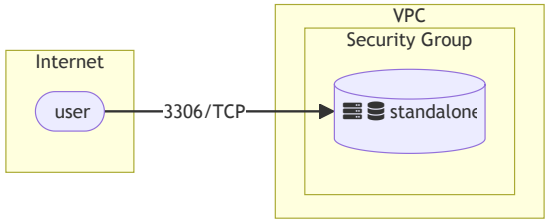

Figure 1: Standalone MySQL server

In this simple configuration, we deploy a single *t2.micro* instance running Ubuntu 22.04 and install the latest version of MySQL Server from the system APT repositories.

We allow remote connections to the database by modifying the `bind-address` option in the MySQL configuration file and update the root password. We also open the port 3306 on the AWS security group to allow incoming connections from the Internet.

Finally, we load the `sakila` sample database using the `mysql` command-line client.

Figure 1 illustrates this configuration.

5.1.2 Cluster

Directory: `src/cluster`

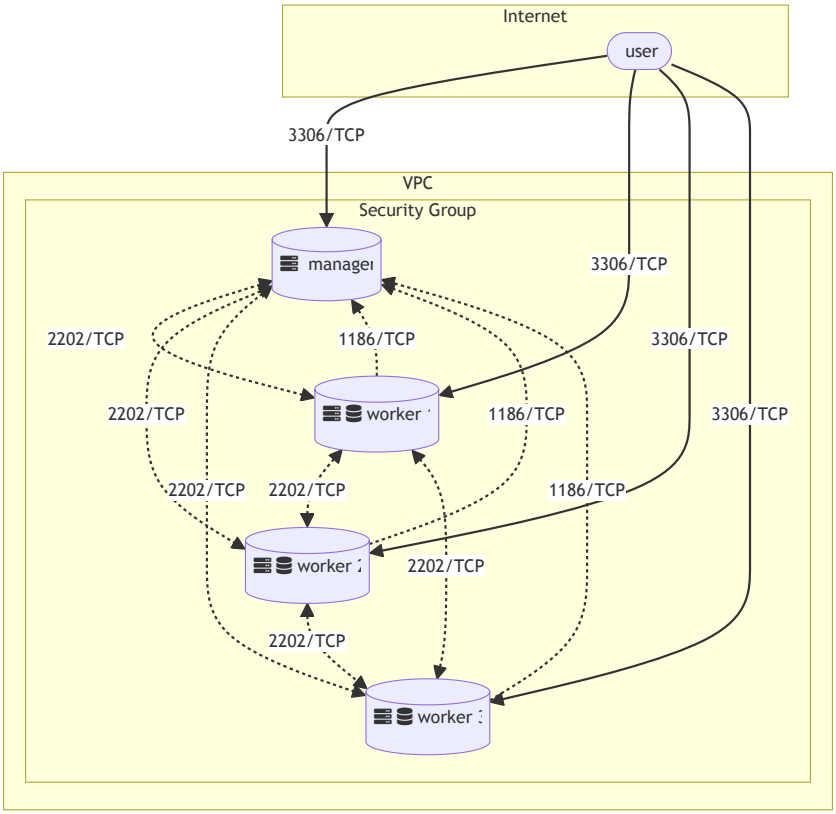

Figure 2: MySQL cluster

This configuration requires four *t2.micro* instances running Ubuntu 22.04.

First, we need to setup the official MySQL APT repository that provides cluster-compatible packages on all instances.

Then, we install the required packages on each node:

- Manager node: `mysql-cluster-community-management-server` (`mgmd` service, 1186/TCP) and `mysql-cluster-community-server` (`mysqld` service, 3306/TCP)
- Worker node: `mysql-cluster-community-data-node` (`ndbd` service, 2202/TCP) and `mysql-cluster-community-server` (`mysqld` service, 3306/TCP)

`mgmd` will coodinate the cluster from the manager node while `ndbd` will store the data on worker nodes. The `mysqld` service is installed on all nodes to allow API access to the database.

Using Jinja2 templates, we dynamically generate the configuration files for each node and copy them to the appropriate location.

We also load the `sakila` sample database on each node and open the port 3306 on the AWS security group to allow incoming connections from the Internet.

Figure 2 illustrates this configuration.

5.2 Benchmarking

File: `tools/benchmark.sh`

For the benchmarking, we use the `sysbench` tool to generate a workload on the database. We chose the `oltp_read_write` test that simulates a mixed OLTP workload accessing a database with 1,000,000 rows. The test is run for 60 seconds across 6 threads.

5.3 Cloud patterns

Directory: `src/patterns`

To offer better read scalability and security, we implement two Cloud patterns: Proxy and Gatekeeper. Proxy will act as a "smart" load balancer between the client and the database cluster nodes while Gatekeeper will help minimize the attack surface by filtering the incoming traffic.

Figure 3 shows the complete architecture of the project with the cluster behind a Proxy and a Gatekeeper.

5.3.1 Proxy

File: `src/patterns/apps/proxy.ts`

The proxy is a Deno (`https://deno.com`) TypeScript application that offers a simple REST API to query the database. 3 (POST) routes are available:

- `/direct`: Direct hit, all requests are forwarded to the master
- `/random`: Random hit, one slave is randomly selected
- `/customized`: Measure the latency of each node and select the fastest one

If the query is a write statement (e.g. `INSERT`, `UPDATE`, `DELETE`), the request is always replicated to all nodes and the response from the master is returned to the client.

5.3.2 Gatekeeper

Files: `src/patterns/apps/{gatekeeper,trusted}.ts`

Gatekeeper is composed of two instances, one public-facing (referred to as "gatekeeper") and one private (referred to as "trusted host").

They both also run Deno TypeScript applications. The gatekeeper forwards all incoming requests to the trusted host that will sanitize the query and forward it to the proxy. The response is then returned to the client.

5.3.3 Security Considerations

We use multiple security groups to restrict the access to the different instances:

- The cluster security group allows all internal traffic from the 4 instances and the traffic from the proxy on port 3306 (mysqld).
- The proxy security group only allows incoming traffic from the trusted host on port 9000 (Deno Proxy application).
- The trusted host security group only allows incoming traffic from the gatekeeper on port 8000 (Deno Trusted Host application).
- The gatekeeper security group allows all incoming traffic from the Internet on port 3000 (Deno Gatekeeper application).

We also use the internal IP addresses of the instances to communicate between them inside the VPC.

The trusted host is responsible for sanitizing the incoming queries to prevent SQL injections. Because we allow any query to be executed on the database, we cannot use prepared statements to prevent this kind of attack. Instead, we rely on a simple regular expression to filter out the dangerous characters. It is far from a perfect solution but it is enough for the scope of this project.

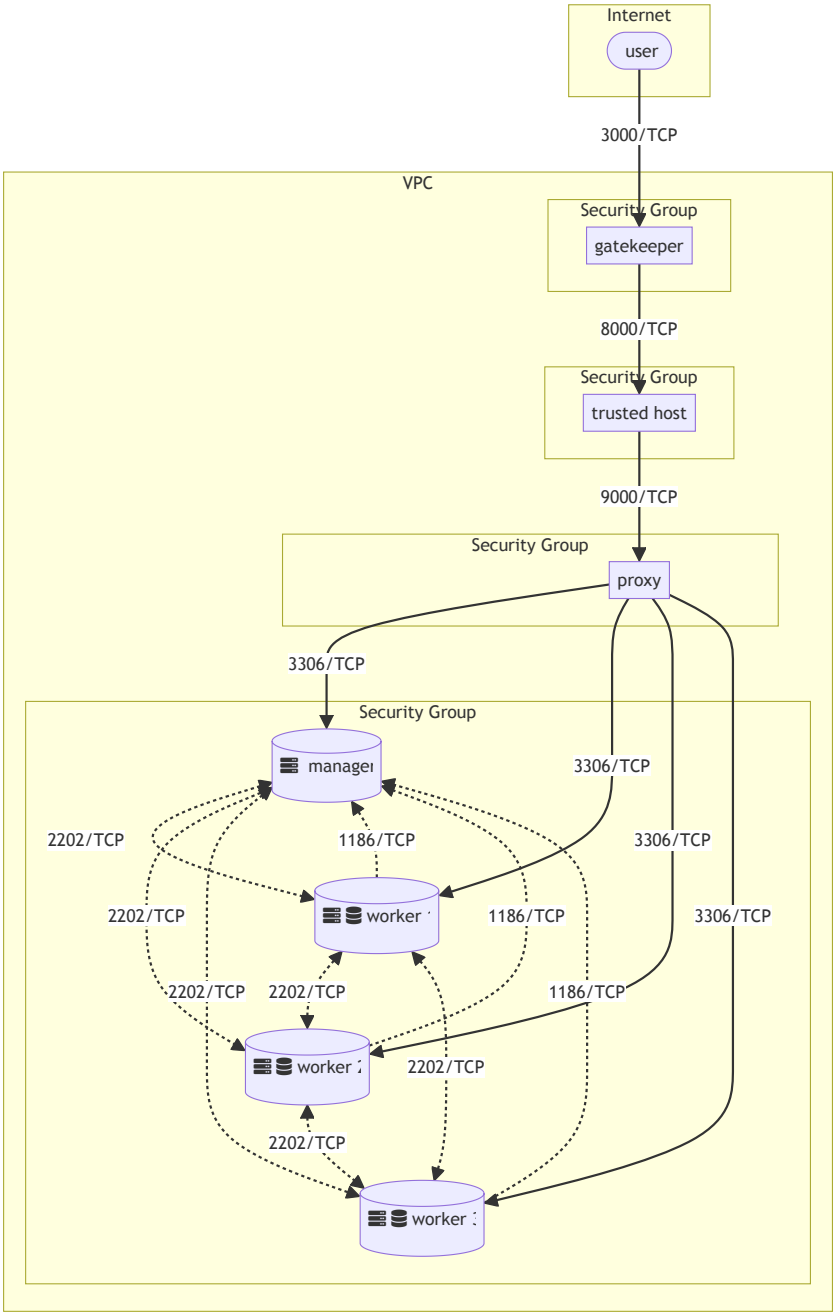

Figure 3: Cloud patterns

6 Results and Outcomes

6.1 MySQL deployments

6.1.1 Standalone server

Once the deployment is complete, we can simply connect to the database using the `mysql` command-line client. Figure 4 shows a simple query on the loaded `sakila` database.

```
INFO:paramiko.transport:Connected (version 2.0, client OpenSSH_8.9p1)
INFO:paramiko.transport:Authentication (publickey) successful!
INFO:paramiko.transport.sftp:[chan 0] Opened sftp connection (server version 3)
INFO:paramiko.transport.sftp:[chan 0] sftp session closed.
INFO:standalone.setup:Public IP: 54.198.81.211
(log8415-project-py3.12) namvu@MacBook-Pro-de-Nam ~/D/N/log8415-project (main)> /usr/local/opt/mysql-client/bin/mysql -h54.198.81.211 -uroot -pP@ssw0rd saki
la
mysql: [Warning] Using a password on the command line interface can be insecure.
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 8.0.35-0ubuntu0.22.04.1 (Ubuntu)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SELECT * from customer LIMIT 3;
+-------------+----------+------------+-----------+-----------------------------------+------------+--------+---------------------+---------------------+
| customer_id | store_id | first_name | last_name | email                             | address_id | active | create_date         | last_update         |
+-------------+----------+------------+-----------+-----------------------------------+------------+--------+---------------------+---------------------+
|           1 |        1 | MARY       | SMITH     | MARY.SMITH@sakilacustomer.org     |          5 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           2 |        1 | PATRICIA   | JOHNSON   | PATRICIA.JOHNSON@sakilacustomer.org |          6 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
|           3 |        1 | LINDA      | WILLIAMS  | LINDA.WILLIAMS@sakilacustomer.org |          7 |      1 | 2006-02-14 22:04:36 | 2006-02-15 04:57:20 |
+-------------+----------+------------+-----------+-----------------------------------+------------+--------+---------------------+---------------------+
3 rows in set (0.09 sec)
```

Figure 4: Standalone query

6.1.2 Cluster

Figure 5 shows a deployed cluster infos: here we have 172.31.95.195 as the manager node (coordinator, `ndb_mgmd`) and 172.31.28.31, 172.31.40.222 and 172.31.6.155 as worker nodes (responsible for storing data, `ndbd`). Every node exposes the MySQL API (`mysqld`) that can be accessed on their respective port 3306.

```
ubuntu@ip-172-31-95-195:~$ sudo ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: ip-172-31-95-195.ec2.internal:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     3 node(s)
id=2    @172.31.28.31  (mysql-8.0.35 ndb-8.0.35, Nodegroup: 0, *)
id=3    @172.31.40.222  (mysql-8.0.35 ndb-8.0.35, Nodegroup: 0)
id=4    @172.31.6.155  (mysql-8.0.35 ndb-8.0.35, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @172.31.95.195  (mysql-8.0.35 ndb-8.0.35)

[mysqld(API)]   4 node(s)
id=5    @172.31.95.195  (mysql-8.0.35 ndb-8.0.35)
id=6    @172.31.28.31  (mysql-8.0.35 ndb-8.0.35)
id=7    @172.31.40.222  (mysql-8.0.35 ndb-8.0.35)
id=8    @172.31.6.155  (mysql-8.0.35 ndb-8.0.35)
```

Figure 5: Cluster informations

6.2 Benchmarking

We ran `tools/benchmark.sh` on both standalone and cluster deployments (we benchmarked the master of the later). The results are shown respectively in Figures 6 and 7.

The cluster deployment appear to be slightly slower than the standalone server (2941.76 vs 3072.47 queries per second). This can be due to the overhead of the cluster management and the network latency between the manager and worker nodes.

```
+ sysbench --mysql-host=52.203.134.6 --mysql-user=root --mysql-passwo
sysbench 1.0.20 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 6
Initializing random number generator from current time


Initializing worker threads...

Threads started!

SQL statistics:
    queries performed:
        read:                            129122
        write:                           36892
        other:                           18446
        total:                           184460
    transactions:                        9223   (153.62 per sec.)
    queries:                             184460 (3072.47 per sec.)
    ignored errors:                      0      (0.00 per sec.)
    reconnects:                          0      (0.00 per sec.)

General statistics:
    total time:                          60.0343s
    total number of events:              9223

Latency (ms):
         min:                                    26.27
         avg:                                    39.05
         max:                                   119.14
         95th percentile:                        50.11
         sum:                                 360156.45

Threads fairness:
    events (avg/stddev):           1537.1667/4.30
    execution time (avg/stddev):   60.0261/0.00
```

Figure 6: Standalone benchmark

```
+ sysbench --mysql-host=3.91.92.244 --mysql-user=root --mysql-passwor
sysbench 1.0.20 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 6
Initializing random number generator from current time


Initializing worker threads...

Threads started!

SQL statistics:
    queries performed:
        read:                            123704
        write:                           35344
        other:                           17672
        total:                           176720
    transactions:                        8836   (147.09 per sec.)
    queries:                             176720 (2941.76 per sec.)
    ignored errors:                      0      (0.00 per sec.)
    reconnects:                          0      (0.00 per sec.)

General statistics:
    total time:                          60.0706s
    total number of events:              8836

Latency (ms):
         min:                                    28.47
         avg:                                    40.77
         max:                                   146.01
         95th percentile:                        51.02
         sum:                                360231.88

Threads fairness:
    events (avg/stddev):           1472.6667/4.99
    execution time (avg/stddev):   60.0386/0.01
```

Figure 7: Cluster benchmark

6.3 Cloud patterns

The patterns module deploys a cluster (Figure 8) behind a proxy, a trusted host and a gatekeeper.

The gatekeeper is accessible from the Internet on port 3000. It receives POST requests from the client on `/direct`, `/random` and `/customized` routes with the query in the body (`query` field). After forwarding to the trusted host and proxy, the cluster response is returned back to the client.

```
ubuntu@ip-172-31-43-243:~$ sudo ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> show
Connected to Management Server at: ip-172-31-43-243.ec2.internal:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     3 node(s)
id=2    @172.31.4.25  (mysql-8.0.35 ndb-8.0.35, Nodegroup: 0, *)
id=3    @172.31.92.71  (mysql-8.0.35 ndb-8.0.35, Nodegroup: 0)
id=4    @172.31.30.16  (mysql-8.0.35 ndb-8.0.35, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @172.31.43.243  (mysql-8.0.35 ndb-8.0.35)

[mysqld(API)]   4 node(s)
id=5    @172.31.43.243  (mysql-8.0.35 ndb-8.0.35)
id=6    @172.31.4.25  (mysql-8.0.35 ndb-8.0.35)
id=7    @172.31.92.71  (mysql-8.0.35 ndb-8.0.35)
id=8    @172.31.30.16  (mysql-8.0.35 ndb-8.0.35)
```

Figure 8: Patterns cluster

Sample requests are shown in Figures 9, 10 and 11. The first request is a read only query (`SELECT`) on `/random` that will be forwarded to a random worker node. The second request is a write query (`INSERT`) on `/direct` that will be replicated to all nodes (the response is the one from the master). The third request is a read only query (`SELECT`) on `/customized` that will be forwarded to the fastest node (according to ping time). This last query confirms that our previous write was successfully replicated to all nodes.

Figure 12 shows the logs of the different components. We can see that the gatekeeper (left) receives the first request on `/random` and forwards it to the trusted host. The trusted host (middle) sanitizes the query and then forwards it to the proxy (right) that chooses a random worker to execute this read only query. The second query on `/direct` is a write query that is replicated to all nodes. Finally, the third query on `/customized` requires the proxy to ping all nodes and select the fastest one to execute the query.

Figure 9: Gatekeeper request (SELECT, `/random`)

Figure 10: Gatekeeper request (INSERT, `/direct`)

Figure 11: Gatekeeper request (SELECT, `/customized`)

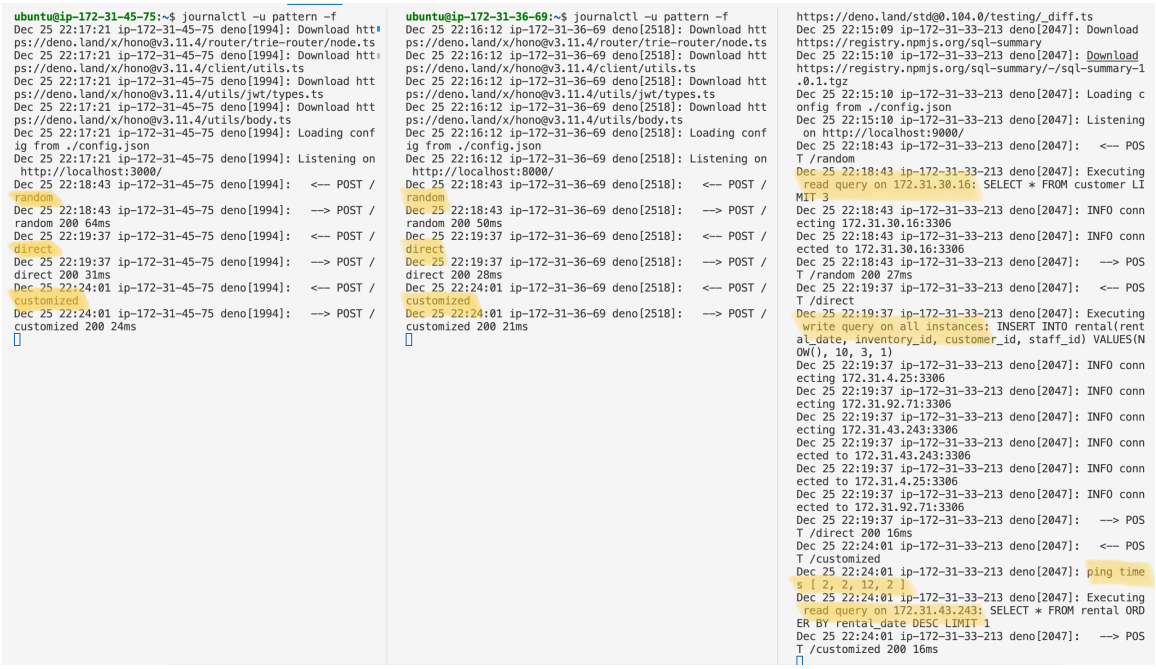

Figure 12: Patterns apps logs (gatekeeper, trusted host, proxy)

7 Lessons Learned

- You better just go through the official ugly documentation of MySQL instead of relying on pretty but outdated third-party tutorials to set up things properly.
- Security groups are an easy and efficient way to restrict the access to the instances.
- Templates are very useful to dynamically generate configuration files for multiple instances.

8 Recommendations

- The *sakila* example database is not meant to be used in a (NDB) cluster: it relies on the `InnoDB` engine that is not the one provided by the MySQL NDB Cluster (`NDBEngine`). That makes this database loaded into the instance locally and not replicated across the three workers. As a consequence for this assignement, we had to manually load the database on the manager and each worker node and rely on the Proxy to replicate the write queries to all nodes (`NDBEngine` should have done that automatically).
- The proxy can be improved to detect `mysqld` failures and automatically remove the instance from the pool of available nodes.
- The Gatekeeper, the Trusted Host and the Proxy can be scaled horizontally to improve the availability of the system (in the current configuration if one of the three instances fails, the whole system is down).

9 Conclusion

In this project, we were able to efficiently setup a MySQL cluster on Amazon EC2 and implement two Cloud patterns: Proxy and Gatekeeper. Security concerns were also addressed by restricting the access to the instances and sanitizing the incoming queries. Finally, we benchmarked the performance of the two deployments and found that the cluster was slightly slower than the standalone server.

All the code was developed in Python and TypeScript while keeping in mind performance (`asyncio` concurrency) and maintenability (modularity, reusability). That allowed us from the start to easily extend the previous projects codebase and implement the new features required for this assignment.

Attachments

Git Repository

`https://github.com/NextFire-PolyMTL/log8415-project`

List of authors:

- Nam Vu: `NextFire <git@yuru.moe>`

Demo Video

`https://drive.google.com/file/d/1eNEslekpXstN5InSCTaYVMzah81akDhY/view`

0:00-12:30 Code walkthrough, 12:30-end Execution demo

Changes since the recording:

- (proxy) Manager node is now included in the considered instances for the `/customized` route (as required by the assignment)
- (proxy) The full response object from the cluster is now returned (instead of only the `rows` field). This is reflected in the Figures 9, 10 and 11.

Signatures

Nam V.