POLYTECHNIQUE MONTRÉAL

LOG8415E – Advanced Cloud Computing

Final Assignement

Cloud Design Patterns : Implementing a DB Cluster

By :
Laurel, Tremblay (2141998)

Submitted to :
Vahid, Majdinasab

December 4th, 2024, Montreal

Table of Contents

Introduction **2**

Sysbench results for MySQL cluster **2**

Cluster Setup **4**

Benchmark Results **7**

Instructions **9**

Conclusion **10**

Introduction

This project focuses on creating a secure and scalable MySQL cluster on Amazon EC2, showcasing the application of the Proxy and Gatekeeper cloud design patterns. The implementation features a distributed MySQL architecture, where a manager node and worker nodes synchronize through replication to manage database queries effectively. A Proxy server routes read and write requests using three different methods : direct hit, random and customized (the node with the lowest ping time is chosen). To enhance security, the Gatekeeper pattern introduces an external-facing gatekeeper and an internal trusted host, ensuring all requests are validated and securely relayed. The entire process (from instance setup and database configuration to dependency installation and benchmarking) was fully automated with AWS SDKs and Python, allowing for efficient deployment and seamless testing.

Sysbench results for MySQL cluster

The Sysbench benchmarking tool was used to evaluate the performance of the MySQL cluster. Tests were conducted on the manager node and both worker nodes to assess their ability to handle database queries efficiently.

The following results highlight the performance of the manager node, which processes all write operations in the cluster :

```
SQL statistics:
    queries performed:
        read:                            123018
        write:                           0
        other:                           17574
        total:                           140592
    transactions:                        8787   (878.45 per sec.)
    queries:                             140592 (14055.20 per sec.)
    ignored errors:                      0      (0.00 per sec.)
    reconnects:                          0      (0.00 per sec.)

General statistics:
    total time:                          10.0004s
    total number of events:              8787

Latency (ms):
         min:                                    0.81
         avg:                                    1.13
         max:                                    6.20
         95th percentile:                        1.93
         sum:                                 9972.38

Threads fairness:
    events (avg/stddev):           8787.0000/0.00
    execution time (avg/stddev):   9.9724/0.00
```

FIGURE 1 – Sysbench results for the manager node.

Similarly, the worker nodes were tested for their performance in handling read-only queries. The results are as follows :

```
SQL statistics:
    queries performed:
        read:                            124754
        write:                           0
        other:                           17822
        total:                           142576
    transactions:                        8911   (890.85 per sec.)
    queries:                             142576 (14253.58 per sec.)
    ignored errors:                      0      (0.00 per sec.)
    reconnects:                          0      (0.00 per sec.)

General statistics:
    total time:                          10.0003s
    total number of events:              8911

Latency (ms):
         min:                                    0.79
         avg:                                    1.12
         max:                                    6.55
         95th percentile:                        1.89
         sum:                                 9973.68

Threads fairness:
    events (avg/stddev):           8911.0000/0.00
    execution time (avg/stddev):   9.9737/0.00
```

FIGURE 2 – Sysbench results for the first worker node.

```
SQL statistics:
    queries performed:
        read:                            125426
        write:                           0
        other:                           17918
        total:                           143344
    transactions:                        8959   (895.65 per sec.)
    queries:                             143344 (14330.32 per sec.)
    ignored errors:                      0      (0.00 per sec.)
    reconnects:                          0      (0.00 per sec.)

General statistics:
    total time:                          10.0008s
    total number of events:              8959

Latency (ms):
         min:                                    0.80
         avg:                                    1.11
         max:                                    5.59
         95th percentile:                        1.89
         sum:                                 9973.41

Threads fairness:
    events (avg/stddev):           8959.0000/0.00
    execution time (avg/stddev):   9.9734/0.00
```

FIGURE 3 – Sysbench results for the second worker node

Cluster Setup and Pattern Implementations

In this implementation, a MySQL cluster was set up and benchmarked utilizing the Proxy and Gatekeeper cloud design patterns. The architecture was designed with the following components :

- **Manager Node :** A t2.micro instance responsible for managing write operations as the primary database node.
- **Worker Nodes :** Two t2.micro instances configured as replicas of the manager node to handle read operations.
- **Proxy Server :** A t2.large instance that routes database queries to either the manager or worker nodes based on the configured proxy mode (direct hit, random, or customized).
- **Gatekeeper :** A t2.large instance that serves as the gateway to the system, managing the validation and routing of incoming requests.
- **Trusted Host :** A t2.large instance that acts as an intermediary between the Gatekeeper and the Proxy server, ensuring secure communication.

All nodes in the cluster were configured to enable efficient data replication and seamless query routing. The Sakila database was installed on both the manager and worker nodes to ensure consistency throughout the cluster.

To securely manage incoming requests, the following security measures were implemented :

- Security groups were defined to restrict IP addresses that could access each instance. These groups allowed only SSH connections (TCP on port 22) and TCP connections on port 5000, with access limited to the preceding instance in the architecture.
- The Gatekeeper is the only component exposed to the internet, acting as the first line of defense.
- The Trusted Host is accessible only from the Gatekeeper, ensuring that internal communication remains secure.
- The Proxy server is only accessible by the Trusted Host. It handles write SQL queries (INSERT, UPDATE, DELETE) by forwarding them to the Manager node and directs read queries based on the mode selected (direct hit, random, or customized).
- The mode of the Proxy server can be dynamically adjusted via a POST request sent to the Gatekeeper at 'http ://<Gatekeeper ip> :5000/mode', where the mode (e.g., "mode" : "RANDOM") is specified in the request body.
- The Manager is restricted to being accessible solely by the Proxy, preventing direct access to the master database.
- As the master node, the Manager is responsible for handling all write operations. When it receives a write request, the Manager updates its local database and replicates the changes to both worker nodes (slave nodes).
- The Worker nodes are only accessible by the Manager for write operations and by the Proxy (for read operations when the mode is set to random or customized).

Proxy Pattern Implementation

The Proxy pattern is implemented to optimize the routing of database queries, balancing the workload between the Manager and Worker nodes. The Proxy server handles read and write operations as follows :

- **Write Requests :** All write operations (INSERT, UPDATE, DELETE) are forwarded to the Manager node, the master node in the MySQL cluster. This ensures data integrity by processing write operations at the master level.
- **Read Requests :** Read operations are handled differently depending on the configured proxy mode :
  - **Direct Hit Mode :** All read queries are forwarded directly to the Manager node.
  - **Random Mode :** In this mode, the Proxy server routes read queries to either of the Worker nodes at random
  - **Customized Mode :** This mode routes read queries to the Worker node with the lowest response time (based on ping times).
- The Proxy server dynamically adjusts the routing of read queries based on the selected mode.

Gatekeeper Pattern Implementation

The Gatekeeper pattern is implemented to enhance security by controlling external access to the MySQL cluster. It ensures that only authenticated and trusted requests are allowed to interact with the internal components. The Gatekeeper pattern works as follows :

- **Internet-facing Gatekeeper :** The Gatekeeper is the only component in the cluster that is exposed to the internet. It acts as a security filter, validating incoming requests before forwarding them to the Trusted Host.
- **Request Validation :** When the Gatekeeper receives a request, it performs input validation and security checks to ensure the request is legitimate. If the request is validated, it is forwarded to the Trusted Host for further processing.
- **Internal Communication :** The Gatekeeper only communicates with the Trusted Host, ensuring that sensitive internal data is not exposed to external sources. The Trusted Host then securely relays requests to the Proxy server, which handles routing to the Manager or Worker nodes.
- **Access Control :** The Gatekeeper, Trusted Host, and Proxy server are each controlled by strict security groups, ensuring that only authorized instances can communicate with one another.

Benchmark Results

The benchmark tests were conducted across three different proxy modes : DIRECT HIT, RANDOM, and CUSTOMIZED. The following results were obtained for each mode :

Benchmark Results : Direct Hit Mode

Number of Requests per Instance :

— Manager : 2000

— Worker 1 : 0

— Worker 2 : 0

Average Response Time per Instance (in seconds) :

— Manager : 0.1333

— Worker 1 : N/A

— Worker 2 : N/A

In the Direct Hit mode, all requests were routed to the manager node. As a result, the manager handled all 2000 requests, and no read queries were sent to the worker nodes.

Benchmark Results : Random Mode

Number of Requests per Instance :

— Manager : 1000

— Worker 1 : 496

— Worker 2 : 504

Average Response Time per Instance (in seconds) :

— Manager : 0.1536

— Worker 1 : 0.1108

— Worker 2 : 0.1102

In the Random mode, read queries were distributed between the worker nodes. The manager still handled write requests. Worker 1 and Worker 2 shared the load of 1000 read requests, with an approximately equal distribution.

Benchmark Results : Customized Mode

Number of Requests per Instance :

— Manager : 1000

— Worker 1 : 326

— Worker 2 : 674

Average Response Time per Instance (in seconds) :

— Manager : 0.1514

— Worker 1 : 0.1205

— Worker 2 : 0.1212

In the Customized mode, read queries were directed to the worker node with the lowest response time, based on ping measurements. Worker 2 received more requests due to its quicker response time compared to Worker 1.

These benchmark results highlight the differences in performance and load distribution between the three proxy modes. The following observations can be made :

— In **Direct Hit mode**, all queries were directed to the manager node, which resulted in higher response times for the manager and no requests for the worker nodes.

— In **Random mode**, requests were more evenly distributed between the manager and the worker nodes, leading to more balanced load handling and slightly lower response times on the worker nodes.

— **Customized mode** showed a more optimized routing of read queries, where Worker 2, with the lower ping time, handled more queries, resulting in better performance compared to the random distribution.

Instructions

Here are the steps to run the code :

- Copy and paste your AWS CLI Cloud Access credentials into /.aws/credentials

- Open a terminal in project folder and enter the following commands :

```
pip install -r "requirements.txt"

./start.sh
```

Github repository : https ://github.com/liotrique/LOG8415E-final-assignement

Conclusion

In conclusion, the benchmark results demonstrate the effectiveness of the Proxy pattern in distributing database queries across the manager and worker nodes. Each proxy mode (Direct Hit, Random and Customized) offers different performance trade-offs. Direct Hit mode centralizes all traffic on the manager, resulting in higher response times, while Random mode and Customized mode improve load distribution and optimize read query handling, with Customized mode offering the best performance by routing queries based on latency. These results provide valuable insights into the scalability and efficiency of the cluster under varying configurations.