Cloud Design Patterns: Implementing a DB Cluster

LOG8415: Advanced Concepts of Cloud Computing

Amin Nikanjam
Département Génie Informatique et Génie Logiciel
École Polytechnique de Montréal, Québec, Canada
amin.nikanjam[at]polymtl.ca

Student: Matilda Sundberg

November 22, 2024

Contents

1 Benchmarking MySQL with Sysbench

To prepare the MySQL databases for benchmarking, the manager and the workers were configured. On the manager instance, MySQL and Sysbench were installed, followed by the setup of the Sakila database with its schema and data. Binary logging and replication were enabled, appointing the manager as the primary server for the worker intances. The Sakila database was then prepared for benchmarking using Sysbench. On the worker nodes, MySQL was installed and Sysbench was set up. Each worker was configured as a replication slave to the manager. To verify that the replication had worked the command `SHOW SLAVE STATUS` was used.
The benchmarking process was automated using SSH commands to execute Sysbench remotely on both the manager and worker instances. The following command was used to initiate the benchmarking:

```
sudo sysbench /usr/share/sysbench/oltp_read_only.lua \
  --mysql-db=sakila \
  --mysql-user=root \
  --mysql-password=12345hej \
  run > sysbench_results_worker.txt
```

The results of these tests were collected and saved locally in the `/sysbench` map.

2 Implementation of The Proxy Pattern

The Proxy Pattern was implemented to route database queries in the MySQL cluster, where read queries go to workers while handling write queries on the manager node. The implementation included three routing strategies: *direct hit*, *random selection*, and *customized selection*. The proxy application reads the IP addresses of the manager and worker nodes from a `config.json` file. The main operations are defined in the FastAPI application, with endpoints for each routing strategy:

- `/directhit` for direct routing to the manager.
- `/random` for random worker selection.
- `/custom` for latency-based worker selection.

Each endpoint processes the request payload and forwards it to the appropriate database node using HTTP POST requests.

The proxy application was developed using FastAPI and deployed on a t2.large ec2 instance. The application listens for incoming requests on port 5002 and forwards them to the appropriate database node based on the specified strategy. Below is an overview of the implemented routing mechanisms:

2.1 Direct Hit

In this strategy, all incoming requests are routed directly to the manager node. The client specifies the operation type (read or write) in the request payload, and the proxy forwards the request to the corresponding endpoint on the manager.

2.2 Random Selection

To distribute read queries among the worker nodes, the random selection strategy randomly chooses one of the two workers to handle the query, by using Python's `random.choice()` function. Write queries are still forwarded to the manager.

2.3 Customized Selection

The customized selection routes read requests by measuring the latency of each worker node using ICMP ping commands. The worker with the lowest latency is chosen to handle the query. Write queries remain directed to the manager.

3 Implementation of The Gatekeeper Pattern

The Gatekeeper Pattern consists of two components (two t2.large instances): the *Gatekeeper* (internet-facing) and the *Trusted Host* (internal-facing). The Gatekeeper is the public-facing component that handles incoming requests from clients. It performs the initial validation of the requests and routes them to the Trusted Host for further processing. The validation process includes:

- Checking the presence of an authentication key (`auth_key`) to ensure authorized access.
- Validating the query and operation type (read or write).

The Gatekeeper application was implemented using FastAPI and deployed on port 5000. It forwards validated requests to the Trusted Host via HTTP POST.

The Trusted Host is the internal-facing component that processes requests received from the Gatekeeper. The Trusted Host routes the validated requests to the Proxy instance. This component was also implemented using FastAPI and deployed on port 5001. Communication between the Trusted Host and the Proxy occurs over an internal network to ensure secure data transfer. Both the Gatekeeper and the Trusted Host were configured using Python scripts and deployed on separate instances. They each use a `config.json` file to manage the IP addresses of the Trusted Host and the Proxy server. The implementation follows these key steps:

1. The Gatekeeper validates client requests using predefined rules (authentication key, query, and operation type).
2. Validated requests are forwarded to the Trusted Host over an internal channel.
3. The Trusted Host processes the requests and forwards them to the Proxy server.

4 Benchmarking the Clusters

To evaluate the performance of the MySQL cluster, 1,000 read and write requests were sent to each endpoint. An actor was added (write-request) to the Sakila database through the manager, followed by a read request via the workers to verify the correctness of the write operation. This process was repeated 1,000 times for the `/random`, `/custom`, and `/directhit` endpoints.

5 Implementation Description

Below is a detailed description of the key steps involved in the implementation:

1. **Instance Creation:** The Gatekeeper, Trusted Host, Proxy, and Manager MySQL t2.large instances are created using their respective setup scripts. Each instance is configured with a FastAPI application tailored for routing requests, handling authentication, or database operations as needed.

2. **Replication Setup:** The replication setup for the MySQL cluster is initiated by fetching the `MASTER_LOG_FILE` and `MASTER_LOG_POS` from the Manager node. These values and the Managers private IP are used to configure the Worker instances as replication slaves.

3. **Configuration and Deployment:** Using SSH, each instance is configured with a `config.json` file that contains the private IP addresses required for routing. The FastAPI applications are then executed on the respective instances to enable routing and handling of database queries.

4. **Benchmarking:** Sysbench is used to benchmark both the Manager and Worker nodes. The benchmarking results are retrieved from the instances and stored in files for later analysis.

5. **Security Configuration:** To enhance security, the security groups of all instances, except the Gatekeeper, are changed to private mode. This ensures that only internal communication is allowed, and external access is restricted to the Gatekeeper.

6 Summary of Results and Instructions to Run the Code

The results were successful, with all the write and read requests being executed and passed without errors. Below is an example of the response from a write request:

```
WRITE SUCCESS: 7 - 200 - [
  "/custom: {
    'write': {
      'status': 'success',
      'data': 'Write operation completed'
    }
  }"
]
```

This example demonstrates that the `/custom` endpoint was used and the write operation was successfully completed, as indicated by the `status: 'success'` and the HTTP status code `200`.

Here is an example of a response of a read request:

```
READ SUCCESS: 24 - 200 - ["/random: {'Read from Worker 1': {
  'status': 'success',
  'data': [
    {'actor_id': 2224,
     'first_name': 'Test_24',
     'last_name': 'Andersson_24',
```

```
    'last_update': '2024-11-21T20:56:14'}
  ]
}}"]
```

The read request retrieves an actor that had been added in a previous write request. The data shows that the actor was correctly retrieved, including details like `actor_id`, `first_name`, `last_name`, and `last_update`. We also see that Worker 1 was selected by the random selector.

6.1 To Run the Code

To execute the code, deploying the cloud pattern and running the tests, follow these steps:

1. Make the script executable by running the following command:

   ```
   chmod +x run_all.sh
   ```

2. Run the script:

   ```
   ./run_all
   ```

The results will be stored in the `test_results` directory and the `sysbench` directory.