Cloud Design Patterns: Implementing a DB Cluster

Advanced Concepts of Cloud Computing

2025

Abstract

In this lab assignment, you are asked to set up MySQL cluster on Amazon EC2 and implement cloud design patterns. You are tasked with implementing and deploying an architecture by adding the Proxy and the Gatekeeper patterns. In the first part of the assignment, you will install, configure, and deploy a MySQL stand-alone. Next, you will apply your newly acquired skills to implement a distributed cluster of MySQL databases. You are asked to report your results and analysis by producing a report using LATEX format.

1 MySQL Standalone and Sakila

- Create 3 t2.micro instances and install MySQL stand-alone on each of them. To install MySQL, please follow the instructions described in Install MySQL on Ubuntu. For this assignment, one of these instances will be the **manager** and the other two will serve as **workers**.
- Once the setup for the MySQL server is complete, you must install the Sakila database on each of the instances so that they all have the same data. Follow the steps in here: Sakila sample database installation.
- To make sure that the setup was carried out correctly, you should run `sysbench` on each instance. You can follow the steps below:
  - Install sysbench: `sudo apt-get install sysbench -y`
  - Prepare the DB for sysbench with `mysql-user` and `mysql-password` being the user and password you configured MySQL with:

```
sudo sysbench /usr/share/sysbench/oltp_read_only.lua --
   mysql-db=sakila --mysql-user="USER" --mysql-password=
   "PASSWORD" prepare
```

  - Run sysbench:

```
sudo sysbench /usr/share/sysbench/oltp_read_only.lua --
   mysql-db=sakila --mysql-user="USER" --mysql-password=
   "PASSWORD" run
```

If there were no problems and all is carried out correctly, you will be able to see the benchmarking results.

2 Cloud Patterns

In order to implement the cluster, you will be implementing the two cloud patterns described below.

2.1 Proxy

This pattern improves scalability by separating read and write operations in a MySQL cluster. It uses a manager node, worker nodes, and a proxy to control all access to the database:

- All *WRITE* operations (such as *INSERT*, *UPDATE*, *DELETE*, or any change to the data) must be sent to the **manager**. After the manager processes the write, the updated data is replicated to all worker nodes.
- All *READ* operations (queries that only retrieve data and do not modify it) are sent to the **workers**. This distributes the read load and increases scalability.

No component is allowed to access the database nodes directly. Every request must go through a local proxy. The Proxy decides whether a request is a READ or a WRITE, and then forwards it to the correct node (manager or one of the workers). This ensures that:

- The system has a single entry point for database requests.
- No instance can directly modify another instance's data.
- Data remains consistent across all nodes through replication from the manager.

As such, **if an operation does not change the data in the DB, it should be handled by the workers and o.w. handled by the manager**. When the manager changes the data in its own instance, it DOES NOT execute the same query on the workers. Instead, it REPLICATES its current state to the workers. Please follow the steps in this tutorial How To Set Up Replication in MySQL to setup and configure replication.

2.2 The Gatekeeper

This pattern describes a way of brokering access to the storage. This is a typical security best practice and serves to minimize the attack surface of system roles. This is done by communicating over internal channels and only to other roles that are part of the pattern. The Gatekeeper pattern takes two roles that play the gatekeeping game. **This means that the gatekeeper is not a single instance but is actually comprised of two instances. (1) the Gatekeeper and (2) the Trusted Host**:

- **Gatekeeper** is the only internet-facing instance. It receives all user/API requests, checks authentication, authorization, and input validity. It runs with limited trust.
- **Trusted Host** is an internal-facing instance. It only receives validated requests from the Gatekeeper. It has access to the rest of the system (e.g., the proxy or database manager), but is not directly exposed to the internet.

In the context of this assignment, **your Proxy *acts as the Trusted Host***. Consider the following scenario where we need to read/write some data from/to the DB:

1. A user sends a request to the Gatekeeper.
2. The Gatekeeper checks:
   - Is the user authenticated and authorized?
   - Is the query safe (no DROP TABLE, DELETE ALL, etc.)?
3. If the request is valid, the Gatekeeper forwards it to the Trusted Host (Proxy).
4. The Proxy decides whether the operation is a **READ** or a **WRITE**:
   - READ $\rightarrow$ send to a worker node.
   - WRITE $\rightarrow$ send to the manager node.
5. The result is sent back through the Gatekeeper to the user.

3 Implementing Cloud Patterns

3.1 Proxy

Once your proxy has determined the type of the query (READ or WRITE), it needs to decide where to send it. In this manner, the Proxy also acts as a load balancer; for example, when the cluster is under load (multiple READ/WRITE to the DB), it forwards the requests to the instances with less load. So, your proxy needs to have three forwarding strategies:

- Direct hit: meaning that you will directly forward incoming requests to MySQL master node (regardless whether it is READ or WRITE) and there will be no logic to distribute data.

- Random: meaning that when we receive a READ query, we randomly select a worker and send the request to it.
- Customized: meaning that similar to Assignment 1, you should measure the ping time of all workers and forward the message to one with lower response time.

Proxy pattern requires one t2.large instance as the server which will route requests to your database cluster.

3.2 The Gatekeeper

You need to create one t2.large instance for the Gatekeeper. There is no need to setup an extra Trusted Host instance since your Proxy will also perform that job. Please pay attention that the Trusted Host (Proxy) instance **MUST** be secured in any way that might expose it. You must take care of all security aspects that might put this machine at risk. That means that the firewall, unused services, ports, and even IPtable should be refined, tuned, and well cared for.

4 Benchmarking your cluster

You should be sending 1000 write requests and 1000 read requests to your cluster for each implementation of the proxy pattern. In your report and demo, you should show that these requests are received by the cluster and are processed appropriately.

5 Automation and Infrastructure as Code

Your solution should be end-to-end automated. As such, you will need to develop your solutions using AWS' SDKs depending on the programming language that you prefer: AWS SDKs

6 Working in Groups

This is your final assignment. ***It should be done individually.***

7 Report

You need to submit the report alongside your codebase. To present your findings and answer questions, you must write a lab report on your results using the handover documentation format that you used for the previous assignments. **For the demos, you can record a video of going over your code and upload it to Moodle.** In your report, you should explain:

- Benchmarking MySQL with sysbench.

- Implementation of The Proxy pattern.
- Implementation of The Gatekeeper pattern.
- Benchmarking the cluster.
- Describe clearly how your implementation works.
- Summary of results and instructions to run your code.

8 Evaluation

A single final submission for this assignment is due on the date specified in Moodle for each person. You need to submit one PDF file per person. All necessary codes, scripts, and programs must be sufficiently commented and attached to your submission. For the demo, you need to record a video in which you explain your code, your decision process, and finally your results. Your assignment will be graded on presentation and content. Please submit your PDF report and push your code to a GitHub repository.