Cloud Design Patterns: Implementing a DB Cluster

Advanced Concepts of Cloud Computing

2024

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 installations are correct, you should run `sysbench` on each instance. You can follow the steps in here: Benchmark MySQL server Performance with Sysbench. If your setups are correct, you will be able to see the benchmarking results.

  UPDATE: The sysbnech code in the tutorial above is deprecated. To install sysbench and run the benchmarks, please follow these steps:

  - 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
```

2 Cloud Patterns

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

2.1 Proxy

This pattern uses data replication between manager/worker databases and a proxy to route requests and provides read scalability on a relational database. *WRITE* requests are handled by the manager and replicated on its workers, while *READ* requests are processed by workers. When applying this pattern, components must use a local proxy whenever they need to retrieve or write data. This means that no instance is able to **directly** change the state of another instance. This means that 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 **ONLY**. When the manager changes the data in its own instance, the proxy should change the data on all the other instances in the cluster as well.

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: The gatekeeper and the trusted host. There is one Internet-facing web role that handles requests from users (the gatekeeper). The Gatekeeper is suspicious and does not trust any requests it receives. It validates the input and runs in partial trust. After the internet-facing instance validates a request, it sends it to the internal-facing instance (the trusted host) for subsequent processing. When some hacker manages to successfully attack the web role, there is no sensitive data there.

Figure 1 shows the flow of how your solution should work.

Your requests are sent to the gatekeeper. The gatekeeper re-routes the validated requests to the trusted host, and the trusted host re-routes them to the proxy where depending on their type (read or write op) they are re-routed to the corresponding manager/workers.

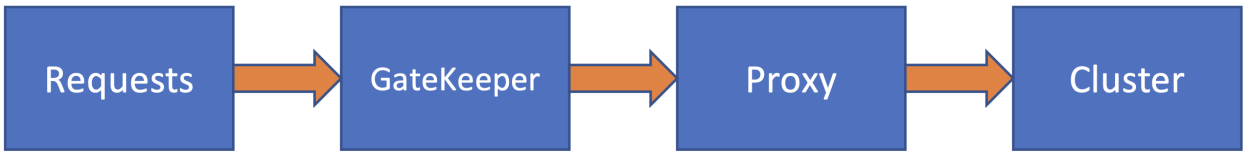

Figure 1: The cluster architecture

3 Implementing Cloud Patterns

3.1 Proxy

Three implementations are required for this pattern:

- Direct hit: meaning that you will directly forward incoming requests to MySQL master node and there will be no logic to distribute data.
- Random: you should implement a code that will randomly choose a slave node on MySQL cluster and forward the request to it.
- Customized: you should measure the ping time of all the servers and forward the message to one with less response time.

Proxy pattern requires one t2.large instance as the server which will route requests to MySQL Cluster.

3.2 The Gatekeeper

You need to create one t2.large instance for the gatekeeper and another t2.large instance for the trusted host. The application must be deployed on the gatekeeper, and it will send requests to the manager of the clustered MySQL database. Please pay attention that the trusted host machine 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. This means that during your demo, running the commands for downloading and setting up Docker, deploying the containers, sending the requests, and getting the results **SHOULD** be done by running a simple bash script. 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 clusters.
- 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.