Cloud Design Patterns: Implementing a MySQL Cluster with Proxy and Gatekeeper

LOG8415E — Final Assignment 2025

Amine Zerouali

November 22, 2025

1 Introduction and Scope

The goal of this assignment is to:

- Provision three MySQL nodes (1 manager, 2 workers), install the *Sakila* dataset, and validate standalone performance with `sysbench`.
- Apply the **Proxy** pattern to separate READ and WRITE traffic and support three routing strategies: direct, random, custom (lowest latency worker).
- Apply the **Gatekeeper** pattern to expose a single public entry point that authenticates and validates requests, forwarding only to the internal Proxy (Trusted Host).
- Automate end-to-end (provisioning, bootstrap, deployment, benchmark) and provide a reproducible handover.

2 System Architecture

Roles and responsibilities

- **Gatekeeper** (t2.large, HTTP:80, public): Validates requests (API key, basic SQL safety) and forwards to the Proxy over the private network.
- **Proxy / Trusted Host** (t2.large, HTTP:8080, private): Classifies SQL as READ/WRITE and applies the selected strategy:
  - *direct*: all traffic to the manager.
  - *random*: READ to a random worker, WRITE to manager.
  - *custom*: READ to lowest-latency worker (measured via connect + `SELECT 1`); fallback to random; WRITE to manager.
- **MySQL manager** (t2.micro): Receives all writes; replicates to workers (GTID).
- **MySQL workers** (2 × t2.micro): `read_only` replicas; serve reads only.

Network security (Security Groups)

- Internet → Gatekeeper:80 (only).

- Gatekeeper → Proxy:8080 (intra-VPC).
- Proxy ↔ MySQL:3306 (intra-VPC, plus intra-MySQL for replication).

3 Implementation Overview

Infrastructure as Code & bootstrap

Provisioning and inventory are handled by `Final/scripts/boto_up_final.py`. MySQL installation, Sakila import, replication and `sysbench` runs are orchestrated by:

- Manager: `Final/db/setup_manager.sh`
- Workers: `Final/db/setup_worker.sh`
- Orchestration: `Final/scripts/bootstrap_mysql.sh`

Each script is idempotent where practical and includes retries for transient apt locks on fresh instances.

Gatekeeper

The Gatekeeper is a FastAPI service on port 80 (root) that:

- Authenticates with `X-API-Key`.
- Validates input SQL with a basic denylist (blocks `DROP/TRUNCATE/ALTER`, etc.).
- Forwards valid requests to the internal Proxy at `http://<proxy_priv_ip>:8080/query`.

Deployment: `Final/scripts/deploy_gatekeeper.sh`.

Proxy (Trusted Host)

The Proxy is a private FastAPI service on port 8080 that:

- Classifies SQL as READ or WRITE (READ includes `SELECT/SHOW/DESC/DESCRIBE/EXPLAIN` without `FOR UPDATE`).
- Implements *direct*, *random*, and *custom* strategies (as described above).
- Connects using minimal MySQL credentials (default `app/password`, DB `sakila`), configurable via environment variables.

Deployment: `Final/scripts/deploy_proxy.sh`.

4 Benchmarking MySQL with sysbench (Standalone)

To ensure each node is correctly installed and accessible, we run:

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

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

On the manager we use the `app` user; on workers the scripts temporarily toggle read-only to allow `sysbench` to create its tables, then restore `read_only/super_read_only`.

5 Cluster Benchmark via Gatekeeper

Methodology

We send 1000 READ and 1000 WRITE requests per strategy (`direct`, `random`, `custom`) through the Gatekeeper using `Final/benchmark/bench.py`. Concurrency defaults to 100 and can be overridden via the `CONCURRENCY` environment variable. Results are written to `Final/benchmark/results/results.csv`.

Results

Type	Strategy	Success	Total (s)	Avg (s)
READ	direct	1000	3.6333	0.003633
WRITE	direct	1000	3.8279	0.003828
READ	random	1000	2.9120	0.002912
WRITE	random	1000	3.9527	0.003953
READ	custom	1000	2.8926	0.002893
WRITE	custom	1000	3.8684	0.003868

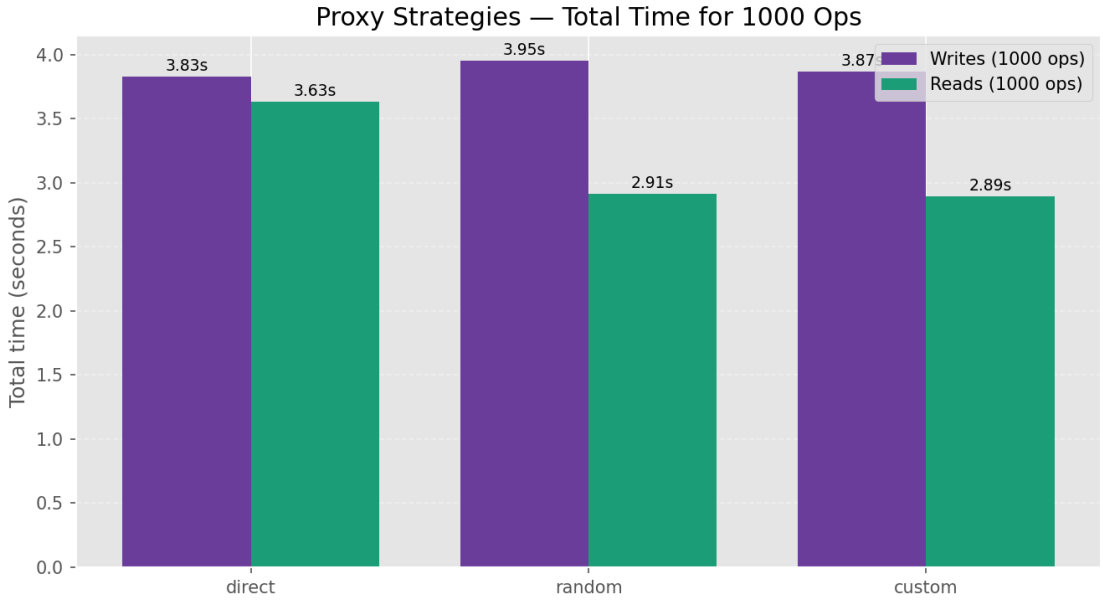

Figure 1: Total time for 1000 reads/writes per strategy (our run). Lower is better.

Analysis

For reads, **random/custom** outperform **direct**, confirming the benefit of distributing READ load across workers. For writes (always sent to the manager), performance is similar across strategies, with minor overhead from selection in the *random/custom* code paths.

6 How the Implementation Works (Pattern Perspective)

Proxy pattern

All database requests traverse the Proxy. The Proxy:

1. Classifies SQL as READ or WRITE.
2. For READ: selects a worker (random or min-latency); for WRITE: targets the manager.
3. Executes the SQL and returns results/errors to the caller.

Data consistency is maintained by MySQL replication from the manager to the workers.

Gatekeeper pattern

The Gatekeeper is the *only* internet-facing role. It:

1. Authenticates and validates input (deny unsafe SQL).
2. Forwards validated requests to the trusted Proxy role over internal networking.
3. Returns Proxy results back to the client.

This reduces attack surface and prevents direct database exposure.

7 Summary of Results and Reproduction Instructions

Key findings

- `sysbench` validates MySQL readiness on all nodes (manager + workers).
- GTID replication successfully mirrors manager writes to workers with workers `read_only`.
- READ strategies that leverage workers (random/custom) reduce total time vs direct reads to the manager.

Prerequisites

```
# AWS credentials and region
export AWS_ACCESS_KEY_ID="..."
export AWS_SECRET_ACCESS_KEY="..."
export AWS_SESSION_TOKEN="..."         # if applicable
export AWS_DEFAULT_REGION="us-east-1"

# EC2 key pair name and local key path
export KEY_NAME="assignment_final"
export KEY_PEM="/path/to/assignment_final.pem"

# Python deps
pip install -U pip
pip install -r Final/requirements.txt
```

One-command end-to-end run

```
bash Final/scripts/run_demo.sh
```

Quick checks

```
# Gatekeeper health
curl -s http://<GK_PUBLIC_IP>/health

# Read via Gatekeeper (random worker)
curl -s -H "X-API-Key: changeme" -H "Content-Type: application/json" \
  -X POST "http://<GK_PUBLIC_IP>/query?strategy=random" \
  -d '{"sql":"SELECT COUNT(*) AS cnt FROM film;"}'
```

Teardown

```
python Final/scripts/boto_down_final.py
```

8 Conclusion

We have delivered a reproducible, automated implementation of the Proxy and Gatekeeper patterns for a MySQL cluster on AWS. Validation with `sysbench` confirms correct MySQL setup, and cluster-level benchmarks demonstrate the benefit of routing reads to replicas. The handover includes code, scripts, and instructions to run, validate, and tear down the environment.