Sorry for late submission since I requested a one week deadline extension from Prof. Ian due to personal emergency.

Assignment 2

CS 6650, Fall 2023, Ximing Liang

Q1

GitHub Repository URL

https://github.com/ximing0116/CS6650-Assignment2

Q2

A short description of your data model (5 points) - Please state size of image used if not using the stock image, and also Database/File storage solution.

Below is my data model:

```
CREATE TABLE albums (
  albumID serial PRIMARY KEY,
  artist VARCHAR(255) NOT NULL,
```

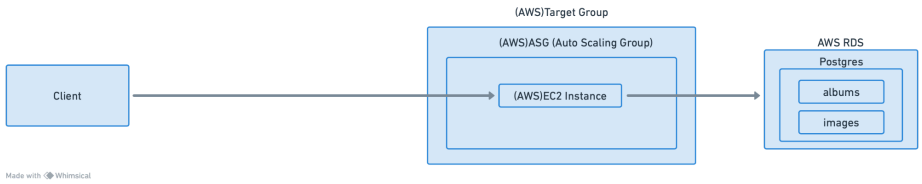
```
    title VARCHAR(255) NOT NULL,
    year VARCHAR(4) NOT NULL
);
CREATE TABLE images (
    imageID serial PRIMARY KEY,
    albumID INT REFERENCES albums(albumID),
    imageData BYTEA NOT NULL
);
```

There are a few benefits:

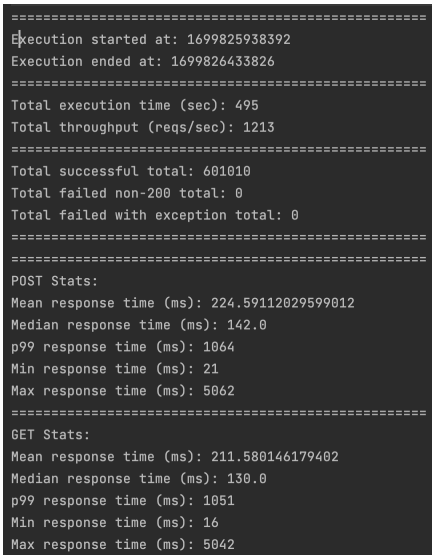
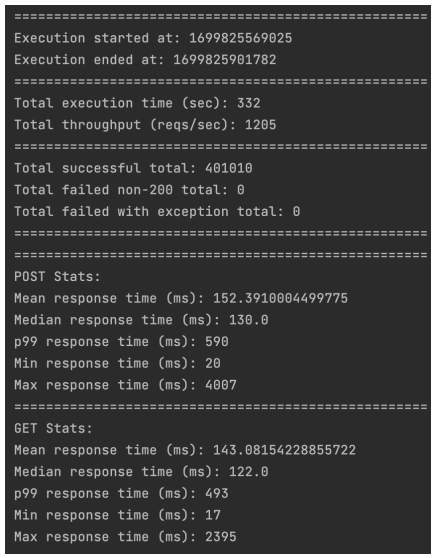
- Images are separated from albums profile information, which follows the principle of normalization to avoid storing duplicate images for multiple albums sharing the same image.
- Provides flexibility to potentially store albums and images tables in different hardwares due to their distinct storage and access pattern. For example, the albums table may be updated more frequently than the images table, therefore storing them together incurs unnecessary overhead.

Q3

Output windows for the 3 client configuration tests run against a single server/DB (5 points)



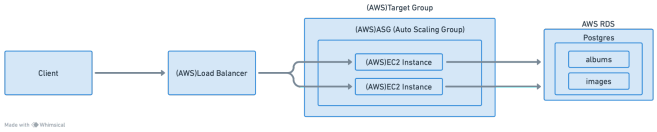
Parameters	Screenshot	Throughput	Failure	Post Stats	Get Stats

10, 30, 2		1213	0	Mean:224 Median:142 P99:1064 Min:21 Max:5062	Mean:211 Median:130 P99:1051 Min:16 Max:5042
10, 20, 2		1205	0	Mean:152 Median:130 P99:590 Min:20 Max:4007	Mean:143 Median:122 P99:493 Min:17 Max:2395

10, 10, 2	``` == Execution started at: 1699825350441 Execution ended at: 1699825515252 == Total execution time (sec): 164 Total throughput (reqs/sec): 1219 == Total successful total: 201010 Total failed non-200 total: 0 Total failed with exception total: 0 == == POST Stats: Mean response time (ms): 77.81175882411759 Median response time (ms): 67.0 p99 response time (ms): 278 Min response time (ms): 21 Max response time (ms): 2046 == GET Stats: Mean response time (ms): 68.93534653465346 Median response time (ms): 60.0 p99 response time (ms): 301 Min response time (ms): 17 Max response time (ms): 1225 ```	1219	0	Mean:77 Median:67 P99:278 Min:21 Max:2046	Mean:68 Median:60 P99:301 Min:17 Max:1225

Q4

Output windows for the 3 client configuration tests run against a two load balanced servers/DB (15 points)



Parameters	Screenshot **Please note the use of ALB endpoint**	Throughput	Failure	Post Stats	Get Stats

10, 30, 2	pool-2-thread-289 - POST request to http://6650-1-1876946533.us-west-2.elb.amazonaws.com:8080/IGORTON/AlbumStore/1.0.0/albums succeeded! AlbumId: 1320104 pool-2-thread-289 - GET request to http://6650-1-1876946533.us-west-2.elb.amazonaws.com:8080/IGORTON/AlbumStore/1.0.0/albums/1320104 succeeded! Response: {"artist":"ximing1","title":"dora","year":"2014"} ========== Execution started at: 1699827392403 Execution ended at: 1699827898864 ========== Total execution time (sec): 497 Total throughput (reqs/sec): 1208 ========== Total successful total: 601010 Total failed non-200 total: 0 Total failed with exception total: 0 ========== ========== POST Stats: Mean response time (ms): 234.21479950668331 Median response time (ms): 160.0 p99 response time (ms): 1322 Min response time (ms): 22 Max response time (ms): 24169 ========== GET Stats: Mean response time (ms): 203.63502325581396 Median response time (ms): 141.0 p99 response time (ms): 1187 Min response time (ms): 17 Max response time (ms): 8345		0	Mean:234 Median:160 P99:1322 Min:22 Max:24169	Mean:203 Median:141 P99:1187 Min:17 Max:83451208
10, 20, 2	pool-2-thread-184 - POST request to http://6650-1-1876946533.us-west-2.elb.amazonaws.com:8080/IGORTON/AlbumStore/1.0.0/albums succeeded! AlbumId: 1020094 pool-2-thread-184 - GET request to http://6650-1-1876946533.us-west-2.elb.amazonaws.com:8080/IGORTON/AlbumStore/1.0.0/albums/1020094 succeeded! Response: {"artist":"ximing1","title":"dora","year":"2014"} ========== Execution started at: 1699827819262 Execution ended at: 1699827358874 ========== Total execution time (sec): 339 Total throughput (reqs/sec): 1180 ========== Total successful total: 401010 Total failed non-200 total: 0 Total failed with exception total: 0 ========== ========== POST Stats: Mean response time (ms): 163.32962851857408 Median response time (ms): 101.0 p99 response time (ms): 1105 Min response time (ms): 20 Max response time (ms): 18314 ========== GET Stats: Mean response time (ms): 135.59679104477613 Median response time (ms): 86.0 p99 response time (ms): 886 Min response time (ms): 17 Max response time (ms): 4702	1180	0	Mean:163 Median:101 P99:1105 Min:20 Max:18314	Mean:135 Median:86 P99:886 Min:17 Max:4702

10, 10, 2	pool-2-thread-97 - POST request to http://6650-1-1876966533.us-west-2.elb.amazonaws.com:8080/IGORTON/AlbumStore/1.0.0/albums succeeded! AlbumId: 820084 pool-2-thread-97 - GET request to http://6650-1-1876966533.us-west-2.elb.amazonaws.com:8080/IGORTON/AlbumStore/1.0.0/albums/820084 succeeded! Response: {"artist":"ximing1","title":"dora","year":"2014"} == Execution started at: 1699824793748 Execution ended at: 1699826964219 == Total execution time (sec): 179 Total throughput (reqs/sec): 1179 == Total successful total: 201810 Total failed non-200 total: 0 Total failed with exception total: 0 == == POST Stats: Mean response time (ms): 83.98912188788912 Median response time (ms): 67.0 p99 response time (ms): 415 Min response time (ms): 21 Max response time (ms): 3697 == GET Stats: Mean response time (ms): 67.89658415841585 Median response time (ms): 58.0 p99 response time (ms): 333 Min response time (ms): 17 Max response time (ms): 3281	1179	0	Mean:83 Median:67 P99:415 Min:31 Max:3697	Mean:67 Median:58 P99:333 Min:17 Max:3281

Q5

Output window for optimized server configuration for client with 30 Thread Groups. Briefly describe what configuration changes you made and what % throughput improvement you achieved (15 points)

Optimization 1: DB connection pooling

Incorrperated https://github.com/brettwooldridge/HikariCP in my code to pool the DB connections, instead of creating a new connection per request. Results reported in Q3 and Q4 already used this optimization.

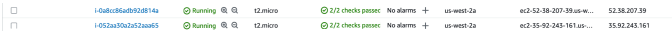
Without this optimization, many non-200 were returned due to connection errors to DB. This optimization is required to complete the tests without error.

Optimization 2: Co-locate Availability Zone

Noticed EC2 instances, DB instances and Load balancers are created in different availability zones, which can cause network overhead since availability zones are geographically spread.

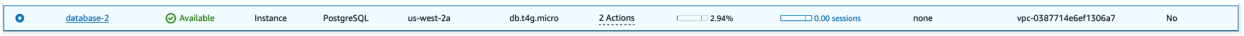
Recreated load-balancer and RDS to match the availability zone of the EC2 instances.

EC2 in us-west-2a



Load-balancer in us-west-2a

DB in us-west-2s

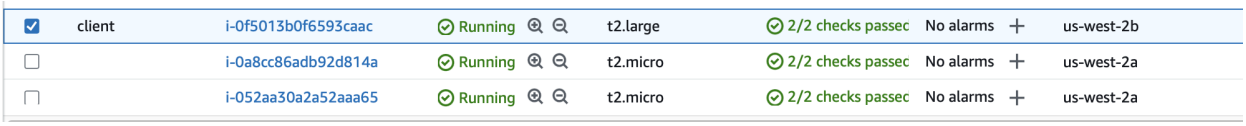


Optimization 3: Run the Client from an EC2 Instance

After checking the CPU/Memory/Network metrics of the ec, db and load-balancer instances, I was not able to find a bottleneck, in particular I observed that all CPU usages are below 50%, no memory pressure and network PPS/Throughput is way under limit.

I started to suspect that the bottleneck is actually the client running on my laptop, for the following reasons:

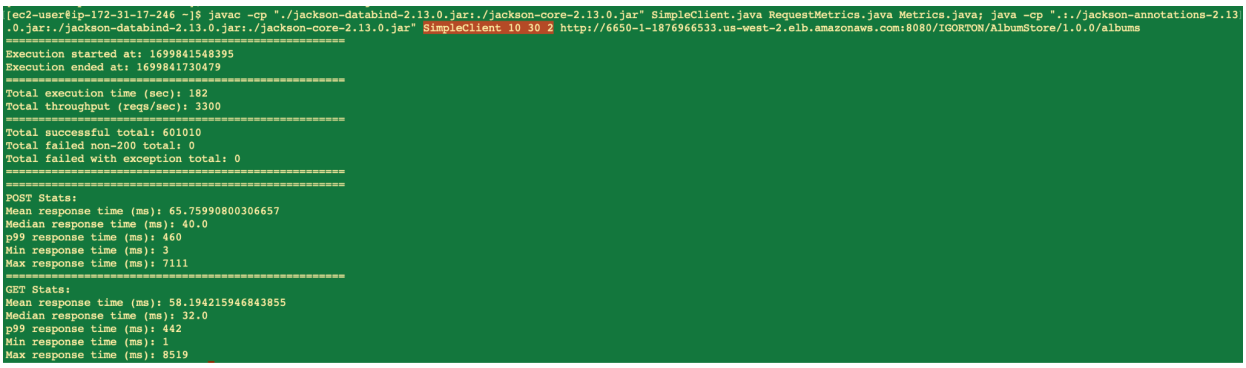
1. My laptop is far-away (a few hundred miles) from the us-west data center. The networking latency between aws data center and my laptop can be high.
2. The uplink of home networking is slow and can limit throughput.



Final Result

- t2.large instance as my client
- 10 connections pool to Postgres
- All services located in the same availability zone

This shows how I ran the client code in an EC2 instance.



Q3, Q4, Q5 Comparison

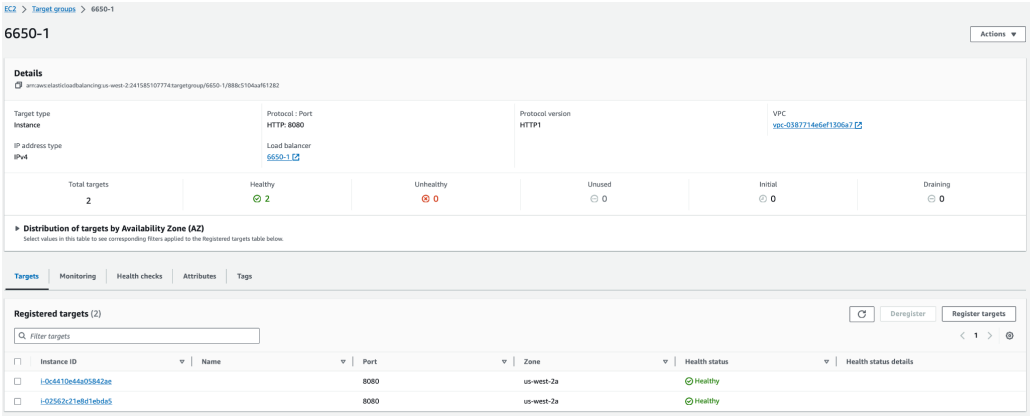
Comparison of performance for the (10 30 2) Test across Q3, Q4 and Q5. All latency in millisecond.

	Throughput	Error	Mean	Median	P99	Min	Max
Q3	1213 rps/s	0	Post:224 Get:211	Post:142 Get:130	Post:1064 Get:1051	Post:21 Get:16	Post:5062 Get:5042
Q4	1208 rps/s	0	Post:234 Get:203	Post:160 Get:141	Post:1322 Get:1187	Post:22 Get:17	Post:24169 Get:83451
Q5	3300 rps/s	0	Post:65 Get:58	Post:40 Get:32	Post:460 Get:442	Post:3 Get:1	Post:7111 Get:8519

Appendix

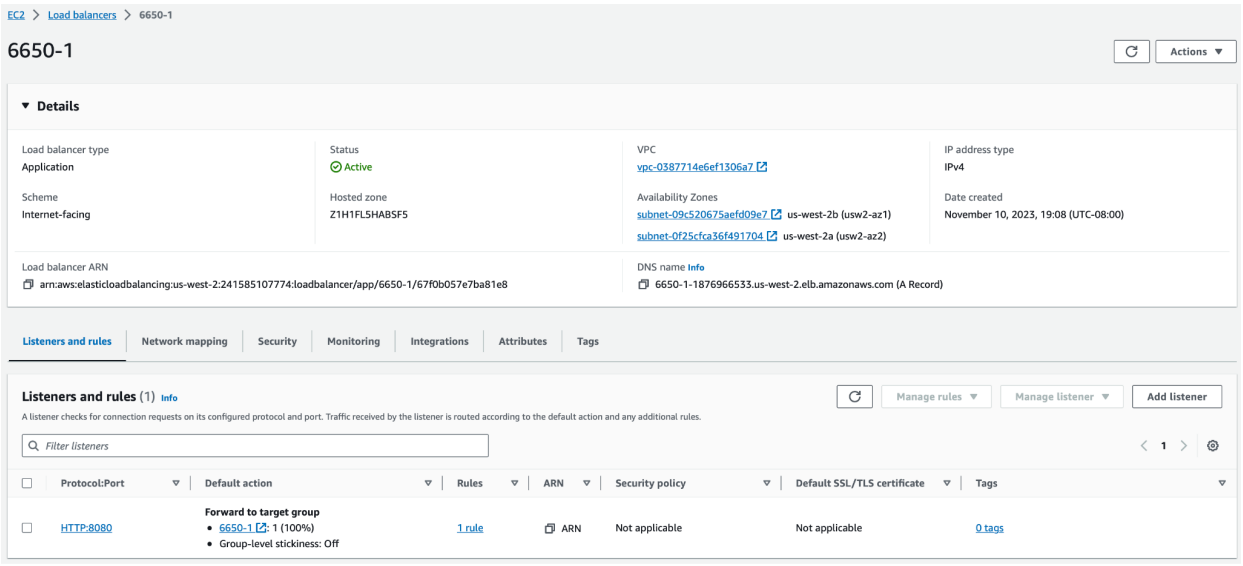
Target Group

Associated with 2 healthy EC2 instances



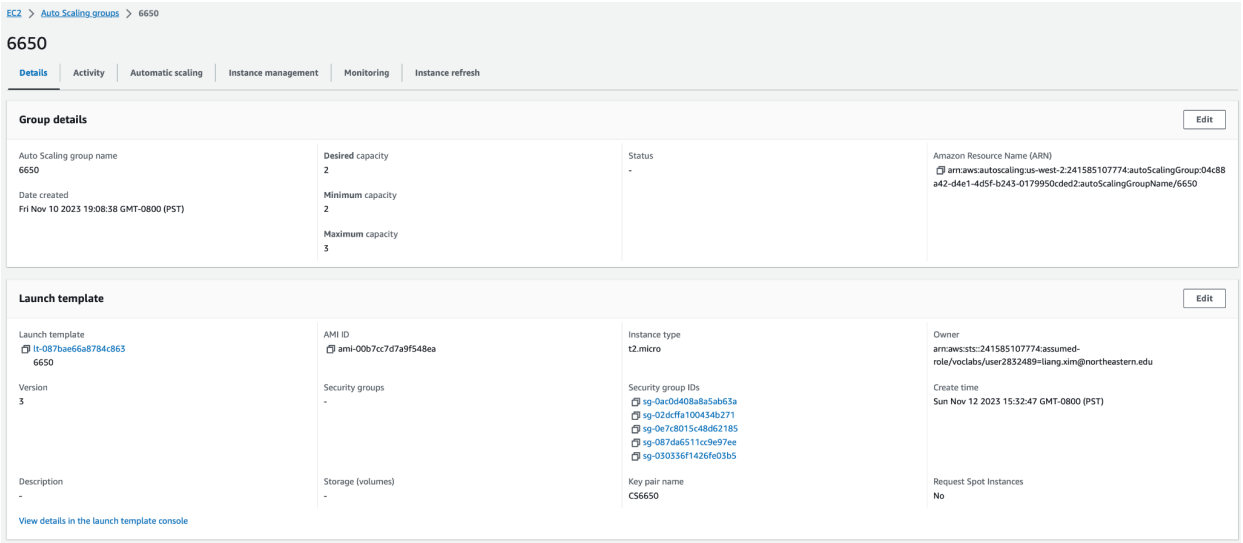
Load Balancer

ALB forwards to the target group shown above



Auto Scaling Group

ASG that manages the EC2 Instances



RDS

Postgres DBs Running in RDS as the Storage Engine