Performance Assessment Report

This report is used to assess the performance of the web site, which provides a runnable (through Docker Compose) setup in a GitHub repository with about 800 book covers.

The website deployed in the following environments:

- Hardware: Macbook Pro 13-inch, 2019, Intel Core I5, 8GB RAM
- Software:
  - Docker: Docker Desktop 4.17.0 (99724)
  - Docker Compose version v2.15.1
  - Java: Java(TM) SE Runtime Environment (build 1.8.0_202-b08)
  - Apache Jmeter: 5.4.1

There are two HTTP GET method endpoints implemented by the web server:

- GetALL: `GET /`. Gets all books.
- Search: `GET /search?q={title}`. Searches books by `title`.

Performance concerns

A well-performed application should focus on the following aspects:

1. **Response time**: Response time also plays a significant role in assessing website performance. Response time of excessive length can lead to user access problems, or web servers need to wait for client responses.
2. **Connection stability**: Network connection stability is another key metric to assess network performance. Poor network connectivity may lead to user access issues, or web servers need to wait for server response.
3. **Throughput**: Throughput is another important metric to assess website performance. If the website experiences performance bottlenecks when processing large amounts of data, it may cause users unable to access the site, or web servers need to wait for client responses.

Measures

The perfomance concerns above can be assessed as follows:

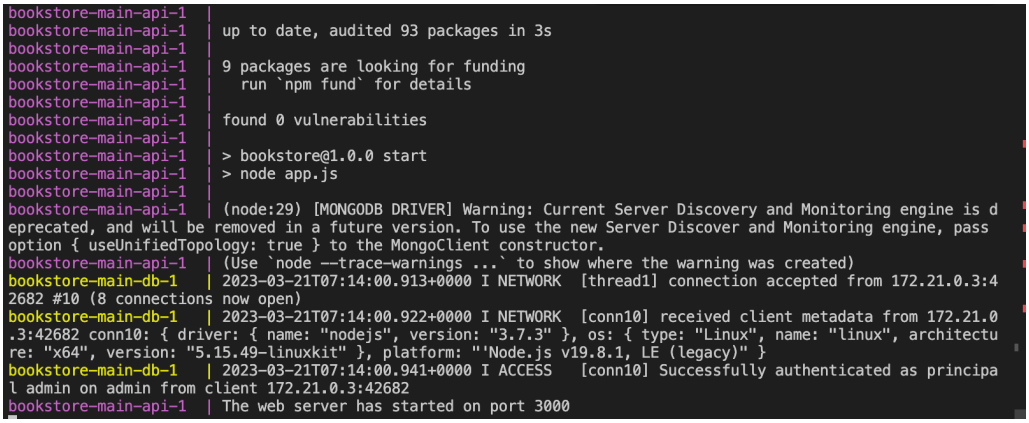
1. **Response time**, which measures how long it takes to process a request and respond in milliseconds. The indicator can be used to assess the performance of single request, and it is good if the average response time is less than 1000 ms。Otherwise, the logic of the endpoints need to be optimized.

2. **Connection stability**，which measures if web servers handle network traffic reliably and smoothly. It is percentage by which packets are dropped due to congestion in the connection. The packet loss rate of a well-designed server should be less than 3%. Otherwise, the infrastructure should be improved.
3. **Throughput** , which measures the rate at which data can be processed within an application without causing unacceptable delays. This metric is commonly expressed as requests per second (req/s). Considering the resource limit of the deployment，the endpoint that can handle over 10 of page requests per second is considered to be performing well. Less than 10 indicates poor performance.

Measurement process

We conducted a performance test using Jmeter.

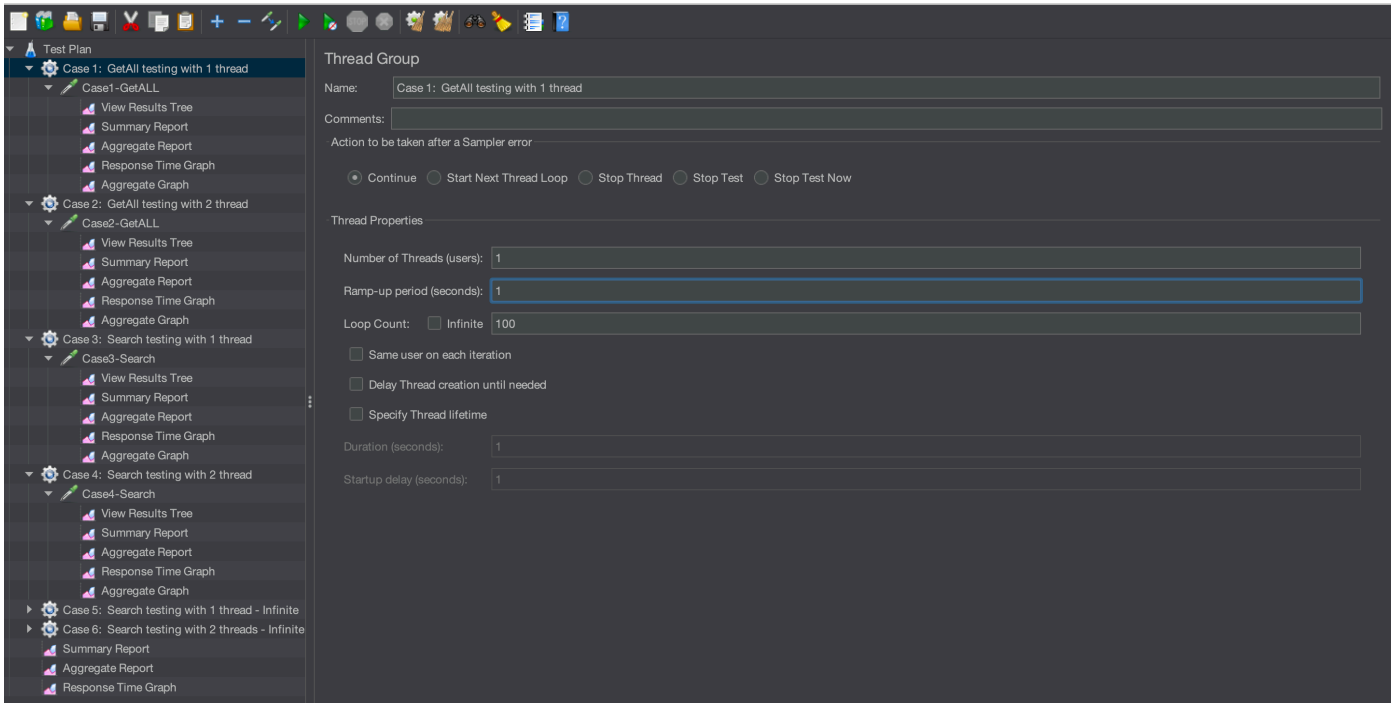
First，We deployed the web on our notebook computer. We installed the software dependencies mentioned before, cloned the code repository to the local machine, and checked the accuracy of the repository, data, and configuration. After that, we launched the application using Docker Compose.



Next，Jmeter was launched。A test plan was devised to better validate the performance of the endpoints.

Case info	Endpoint	Thread	Loop Count
Case 1: GetAll testing with 1 thread	GetALL: `GET /`	1	100
Case 2: GetAll testing with 2 threads	GetALL: `GET /`	2	100
Case 3: Search testing with 1 thread	`GET /search?q={title}`	1	100
Case 4: Search testing with 2 threads	`GET /search?q={title}`	2	100
Case 5: Search testing with 1 threads (30s)	`GET /search?q={title}`	1	Infinite
Case 6: Search testing with 2 threads (30s)	`GET /search?q={title}`	2	Infinite
Case 7: Search testing with 1 thread (Infinite)	`GET /search?q={title}`	1	Infinite

Here is the screenshot of Jemter.



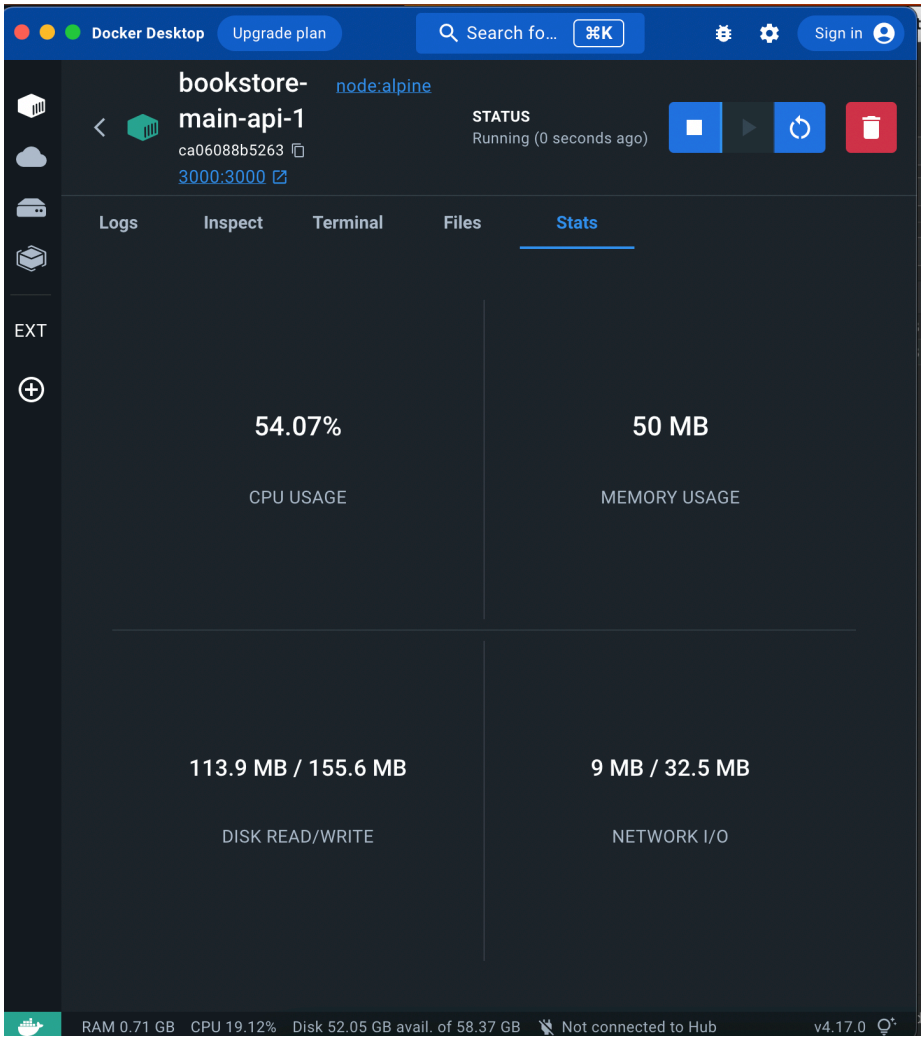
Measurement results

After executing the test plan, we have obtained the following testing results:

Case	Samples	Average Response time(ms)	Connection stability	Throughput	Assessment
Case 1	100	86	100%	11.5/s	✅
Case 2	92	110	92%	6.9/s	❌
Case 3	100	35	100%	27.8/s	✅
Case 4	100	38	100%	15.7/s	✅
Case 5	1117	26	100%	37.2/s	✅
Case 6	1214	40	100%	34.1/s	✅
Case 7	2321	23	99%	41.3/s	✅

From the six testing cases, we found that:

1. Case 1 to 4, we limited the total number of requests. However, In case 2, the application crashed. After analysising the code, we found that the full data exchange between api container and db container is the root cause of the crash.
2. In case 5 and 6, we conducted timed tests on the Search endpoint that performed well. The Search endpopint can reduce the amount of data transferred, which increases the stability of the system and shows better throughput data.
3. We tested a single thread and two threads.The results showed that the system performed better when handling multiple requests from a single thread.
4. In case 7, the program experienced a crash after completing 2312 requests. Docker stats revealed that the memory of the api container was completely used up at the time of the crash.



Next steps

If you had more time, what additional steps would you take for performance assessment of this system?

Would you automate tasks or integrate them in the development process? Why/why not?

1. Using load balancing and running multiple api container in Docker to improve the throughput of the system.
2. Increasing the resource limit of each single container to make the system work more smoothly.
3. Optimizing the program logic and reducing the data exchange of api.