An-Najah National University

Computer Engineering Department

Distributed Operation Systems - 10636456

Microservices Containerization Project – Part 2

Students: Jamal SaadEddin
Omar Qaneer

Instructor: Dr. Samer Arandi

Jan,2024

Introduction

This lab initiative focuses on the improvement of Bazar com's online store efficiency because starting to address the increasing demand and customer concerns over delays in processing orders. Continuing from Lab 1, the project centres on enhancing request processing and handling multiple tasks effectively by adding replication, caching and consistency mechanisms to improve upon this capability.

Work Process

First Part (In-memory Cache)

1. We installed node-cash package by running this command in the terminal.

npm install node-cache –save

2. Add these to use node-cache in your application

```
const NodeCache = require("node-cache");
const myCache = new NodeCache();
```

3. We use get and set methods for cache processes

```
+   if (startOfUrl.startsWith("/info")) {
+     // Using regular expression to extract the id
+     const extractedId = req.originalUrl.match(/\d+/);
+     const id = extractedId[0];
+
+     const cachedBook = myCache.get(`book:${id}`);
+
+     if (cachedBook) {
+       res.json(cachedBook);
+       console.log("Inside Cache");
+     } else {
+       // Fetch book data from backend
+       axios
+         .get(serverUrl + req.originalUrl)
+         .then((response) => {
+           res.json(response.data);
+           var book = response.data;
+           myCache.set(`book:${id}`, book, 300); // Cache for 5 minutes
+         })
+         .catch((err) => {
+           console.error(err);
+         });
+     }
+   }
```

returns true if cache hit, false if miss

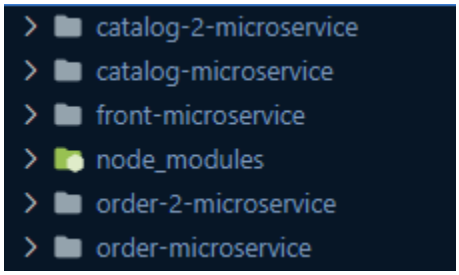
cache hit: return response from cache

cache miss: get data from database

store new fetched data in cache

Second Part (Replication & Load Balance)

1. We created 2 new replicas for catalog and order servers



2. Implement round-robin load balance algorithm
    a. Define 2 new vars in the front server, to store the last server used for both order and catalog servers.

```
+ var lastCatalogServerUsed = 1;
+ var lastOrderServerUsed = 1;
```

    b. On each request, switch between the 2 replicas of catalog servers.

```
-     const serverUrl = "http://localhost:3001";
+     var serverUrl = "";
+     if (lastCatalogServerUsed === 1) {
+       serverUrl = "http://localhost:3004";
+       lastCatalogServerUsed = 2;
+     } else {
+       serverUrl = "http://localhost:3001";
+       lastCatalogServerUsed = 1;
+     }
```

    c. Re-do this for switching between order servers.

```
-     const serverUrl = "http://localhost:3002";
+     var serverUrl = "";
+     if (lastOrderServerUsed === 1) {
+       serverUrl = "http://localhost:3003";
+       lastOrderServerUsed = 2;
+     } else {
+       serverUrl = "http://localhost:3002";
+       lastOrderServerUsed = 1;
+     }
```

3. Synchronize database writes across replicas for consistency
    a. On each server for both catalog and order, add the other replica database to change on it, to ensure that the writes are done in all database replicas.

```
  const db = new sqlite3.Database("bookstore.db");
+ const db1 = new sqlite3.Database("../catalog-microservice/bookstore.db");
```

    b. Write on the other database replica.

```
+   await db1.run("UPDATE books SET quantity = ? WHERE itemNumber = ?", [
+     quantity,
+     id,
+   ]);
+
```

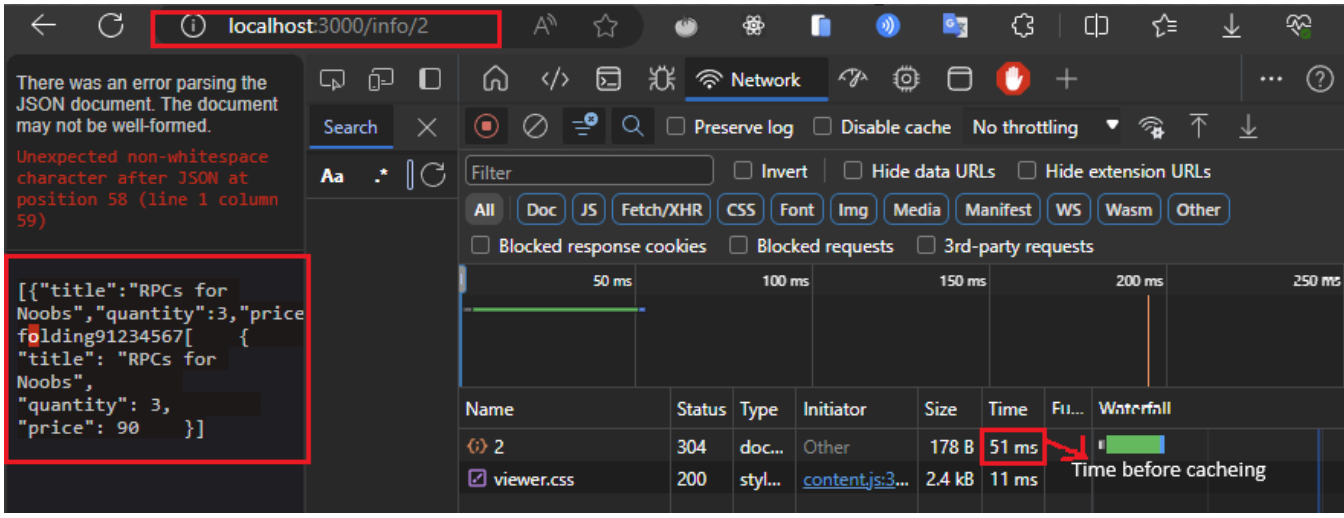
    c. Make sure to do these 2 steps on all replica servers for (order & catalog).

Results

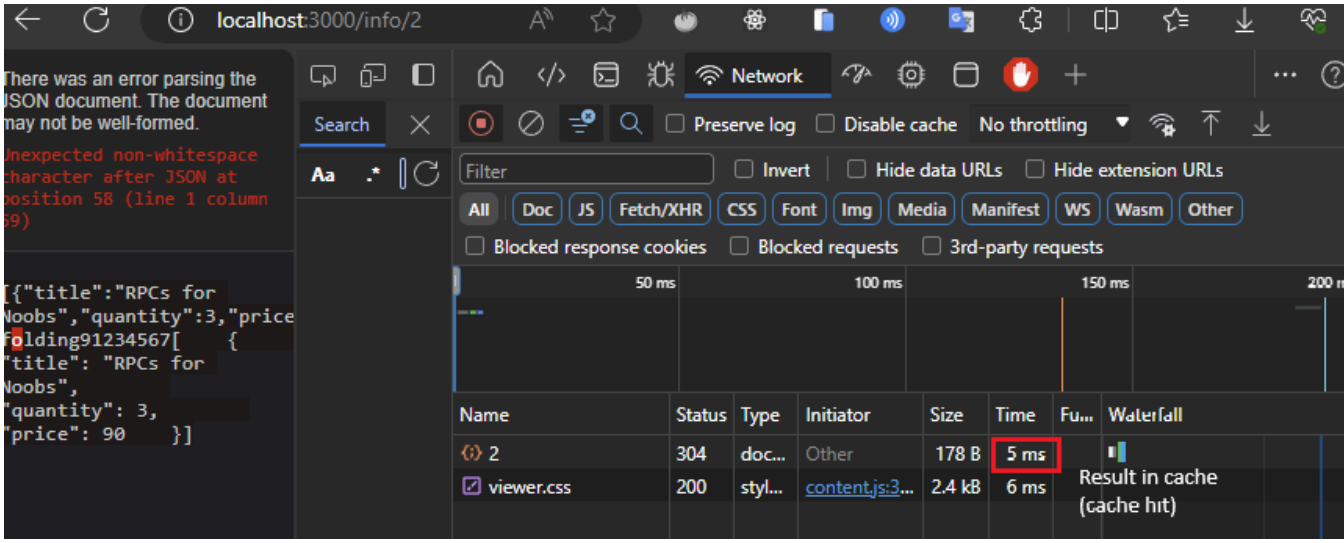
Run all servers by docker compose up

```
:\Users\DELL\Desktop\MicroservicesContainerization>docker compose up
+] Building 9.5s (18/18) FINISHED                                                                                  docker:default
=> [catalog2 internal] load build definition from Dockerfile                                                                0.1s
=> => transferring dockerfile: 158B                                                                                         0.0s
=> [catalog2 internal] load .dockerignore                                                                                   0.1s
=> => transferring context: 2B                                                                                              0.0s
=> [order2 internal] load metadata for docker.io/library/node:16-alpine                                                     3.4s
=> [catalog2 auth] library/node:pull token for registry-1.docker.io                                                         0.0s
=> [order2 1/5] FROM docker.io/library/node:16-alpine@sha256:a1f9d027912b58a7c75be7716c97cfbc6d3099f3a97ed84aa49            0.0s
=> [catalog2 internal] load build context                                                                                   1.7s
=> => transferring context: 15.64MB                                                                                         1.6s
=> CACHED [order2 2/5] WORKDIR /app                                                                                         0.0s
=> CACHED [catalog2 3/5] COPY package.json .                                                                                0.0s
=> CACHED [catalog2 4/5] RUN npm install                                                                                    0.0s
=> [catalog2 5/5] COPY . .                                                                                                  0.9s
=> [catalog2] exporting to image                                                                                            0.4s
=> => exporting layers                                                                                                      0.4s
=> => writing image sha256:45b9a6f66cb92daa2434c0e325bed7492d23eeb04ab36ec60d0a1e17dc16bbee                                 0.0s
=> => naming to docker.io/library/microservicescontainerization-catalog2                                                    0.0s
=> [order2 internal] load build definition from Dockerfile                                                                  0.1s
=> => transferring dockerfile: 158B                                                                                         0.0s
=> [order2 internal] load .dockerignore                                                                                     0.1s
=> => transferring context: 2B                                                                                              0.0s
=> [order2 internal] load build context                                                                                     0.8s
=> => transferring context: 4.24MB                                                                                          0.7s
=> CACHED [order2 3/5] COPY package.json .                                                                                  0.0s
=> CACHED [order2 4/5] RUN npm install                                                                                      0.0s
=> [order2 5/5] COPY . .                                                                                                    0.5s
=> [order2] exporting to image                                                                                              0.3s
=> => exporting layers                                                                                                      0.2s
=> => writing image sha256:760643da6b1c41edff40c508a2dbbfa1cdfee3925c975869caa85e5a61e188c0                                 0.0s
=> => naming to docker.io/library/microservicescontainerization-order2                                                      0.0s
+] Running 5/5
 Container microservicescontainerization-catalog2-1   Cr...                                                                0.3s
 Container microservicescontainerization-catalog-1    Rec...                                                               0.4s
 Container microservicescontainerization-front-end-1  R...                                                                0.3s
 Container microservicescontainerization-order2-1     Crea...                                                              0.2s
 Container microservicescontainerization-order-1      Recre...                                                             0.3s
```

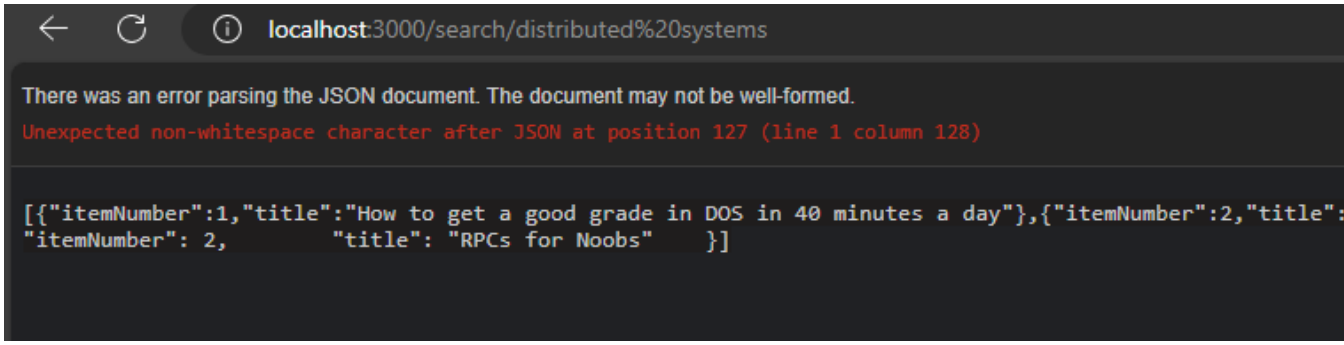
1. Result is not stored in cache yet...



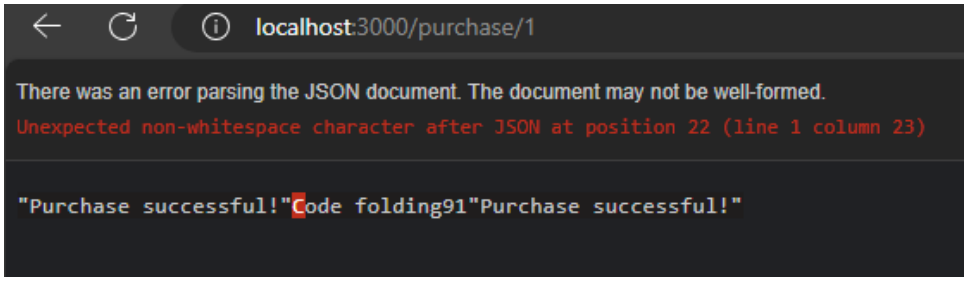
2. Reload the page, cache will hit now...



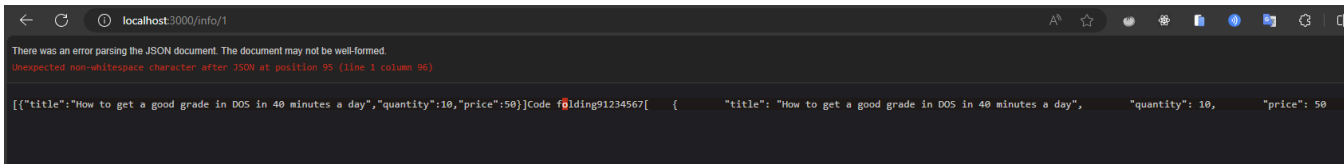
3. Search by topic



4. Purchase book by id



5. Info by book id



Measurements:

Info (without cache)	Info (with cache)	Search (without cache)	Search (with cache)	Purchase	
51	5	38	5	39	Time in “ms”
43	5	45	7	13	Time in “ms”
38	4	37	6	9	Time in “ms”
55	5	40	5	12	Time in “ms”
45	7	41	5	8	Time in “ms”
43	5	45	17	14	Time in “ms”
43	6	36	5	10	Time in “ms”
61	5	37	6	7	Time in “ms”
49	8	35	5	9	Time in “ms”
47	5	43	5	12	Time in “ms”
47.5	5.5	39.7	6.6	13.3	Average

- **How much does caching help?** It reduced latency in the response.
- **What is the overhead of cache consistency operations?** Communication overhead arises from the exchange of invalidation messages among cache nodes, imposing an extra load on the server.
- **What is the latency of a subsequent request that sees a cache miss?** The delay in a subsequent request experiencing a cache miss is greater compared to a cache hit. This discrepancy stems from the need to request data from the server in the case of a cache miss, leading to prolonged retrieval times.

GitHub Repo Link:

https://github.com/Omar-Qaneer/MicroservicesContainerization

End, Thanks