



# Distributed Operation Systems

## Part2

BAZAR.COM: A MULTI-TIER ONLINE BOOK STORE

OLA JAMAL SAADEH SAADEH | 11612344

# INTRODUCTION:

In continuation of the previous lab (Lab 1) where we developed a Node.js application with SQLite for managing books, this lab focuses on optimizing the application for increased popularity and reduced request processing time. Key enhancements include replication, caching, load balancing, and consistency mechanisms.

# OPERATIONS:

- **REPLICATION:**

Implemented two replicas for catalog and order servers on distinct IPs (local machine and VMware) for high availability, maintaining communication via a shared port.

- **CONSISTENCY:**

Utilized a caching mechanism with the "Least Recently Used" (LRU) principle, employing the cashMid middleware to check and retrieve frequently accessed data from the cache before resorting to external queries.

- **LOAD BALANCING:**

Employed a round-robin algorithm with two counters for catalog and order servers, dynamically switching between replicas (1 and 2) to distribute incoming traffic evenly. The introduction of a port variable enhances load balancing by selecting the IP address for each server.

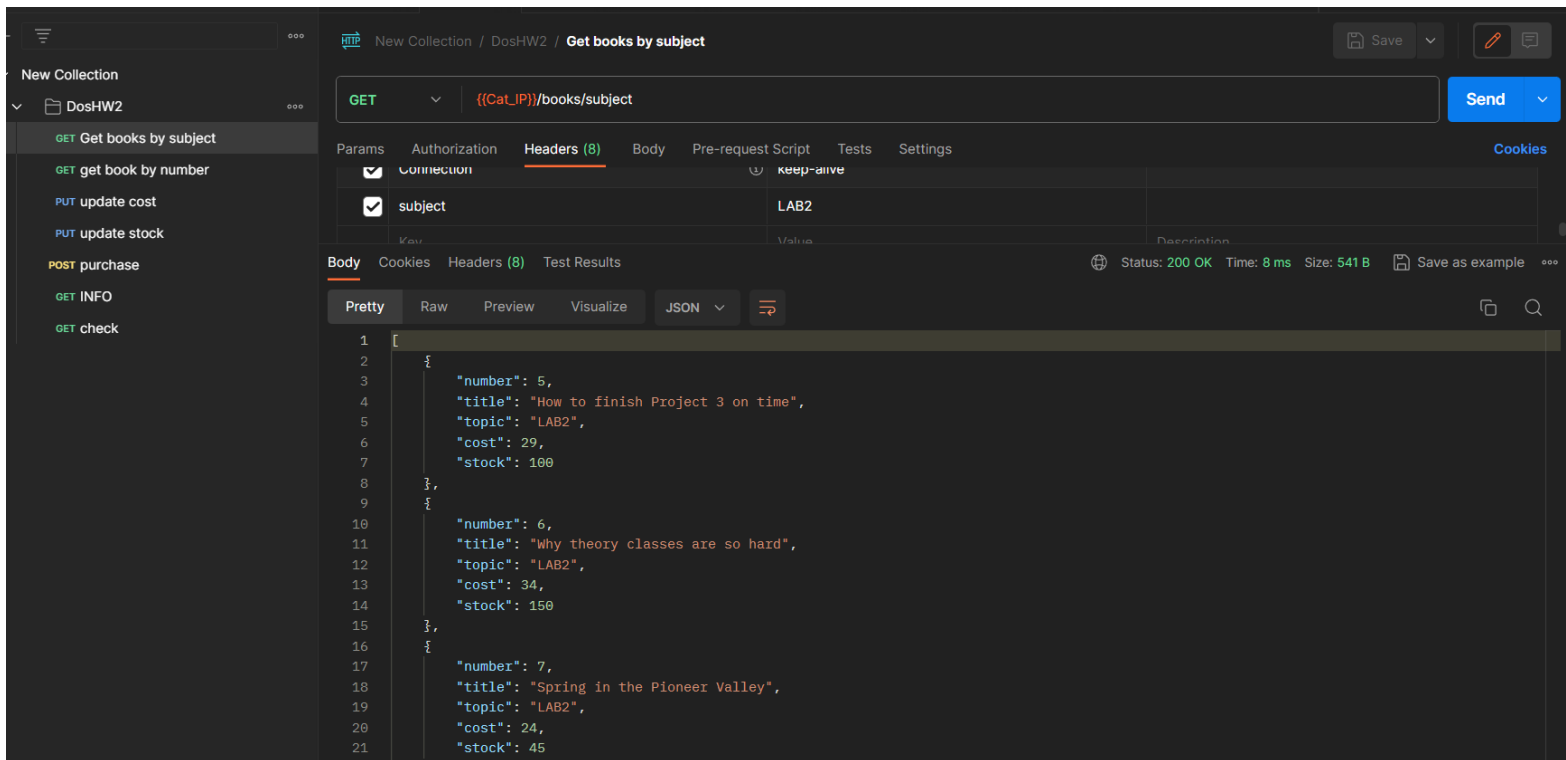
We utilized Docker in our system design, deploying instances of each server independently within containers. This approach provides isolation and flexibility, allowing each container to run its services seamlessly. Docker's containerization technology enhances system scalability, simplifies deployment, and ensures efficient resource utilization

# RUN PROGRAM:

To implement and test the system functionalities:

## 1. Postman Usage:

Utilized Postman for sending requests and visualizing responses on the front end. This streamlined the testing and verification process, ensuring seamless communication.



## 2. Server Deployment:

servers with distinct IP addresses, assigning each replica to a unique port. This setup allowed for isolated testing and ensured that each replica operated independently.

# STEPS:

## 1. Database post new book additions.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\DosProject01a\Catalog> SQLITE3 books.db
SQLite version 3.44.2 2023-11-24 11:41:44 (UTF-16 console I/O)
Enter ".help" for usage hints.
sqlite> SELECT * FROM books;
1|How to get a good grade in DOS in 40 minutes a day|distributed systems|30|87
2|RPCs for Noobs|distributed systems|25|200
3|Xen and the Art of Surviving Undergraduate School|undergraduate school|123|58
4|Cooking for the Impatient Undergrad|undergraduate school|15|105
sqlite> INSERT INTO books (number, title, topic, cost,stock) VALUES (5, 'How to finish Project 3 on time', 'LAB2', 29, 100)
, (6, 'Why theory classes are so hard', 'LAB2', 34, 150), (7, 'Spring in the Pioneer Valley', 'LAB2', 24, 45);
sqlite> SELECT * FROM books;
1|How to get a good grade in DOS in 40 minutes a day|distributed systems|30|87
2|RPCs for Noobs|distributed systems|25|200
3|Xen and the Art of Surviving Undergraduate School|undergraduate school|123|58
4|Cooking for the Impatient Undergrad|undergraduate school|15|105
5|How to finish Project 3 on time|LAB2|29|100
6|Why theory classes are so hard|LAB2|34|150
7|Spring in the Pioneer Valley|LAB2|24|45
sqlite>
```

## 3. At first Catalog server 1 and Catalog server 2 in different Ip's

```
ound 2 vulnerabilities (1 moderate, 1 high)
run `npm audit fix` to fix them, or `npm audit` for details
osaadeh@localhost Catalog]$ node index.js
[INFO] 31-12-2023 04:24:01:962 ==> "Server is running on port 4000" --- (Catalog/index.js:94)

PS D:\DosProject01a\Catalog> nodemon index.js
[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
[INFO] 31-12-2023 06:48:08:952 ==> "server is running on port 4000" --- (Catalog/index.js:96)
```

## 4. Order server 1 and Order server 2 in different Ip's

```
[osaadeh@localhost Order]$ node index.js
[INFO] 31-12-2023 04:25:55:792 ==> "server is running on port 5000" --- (Order/index.js:80)

[nodemon] starting `node index.js`
otherIP: http://192.168.116.1:
myIP: http://localhost:
process.env.HOST: localhost
process.env.otherIP: 192.168.116.1
[INFO] 31-12-2023 05:22:27:628 ==> "server is running on port 5000" --- (Order/index.js:84)
```

Info req: at first time (no cache) 15ms:

The screenshot shows the Postman interface for a GET request to `{{Front_IP}}/info`. The request is configured with the following headers:

Key	Value	Description
Cache-Control	no-cache	
Postman-Token	<calculated when request is sent>	
Host	<calculated when request is sent>	
User-Agent	PostmanRuntime/7.36.0	
Accept	*/*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
booknumber	2	

The response is shown in the Body tab, formatted as JSON:

```
{
  "number": 2,
  "title": "RPCs for Noobs",
  "topic": "distributed systems",
  "cost": 25,
  "stock": 200
}
```

Status: 200 OK, Time: 15 ms, Size: 356 B

```
05:38:31:805 ==> "*****" --- (FrontEnd\index.js:45)
05:38:31:805 ==> "Cache doesn't contain the requested data" --- (FrontEnd\index.js:54)
05:38:31:806 ==> "Querying books related to the book number : 2" --- (FrontEnd\index.js:81)
05:38:31:806 ==> "Exploring on server : http://192.168.116.1:4000" --- (FrontEnd\index.js:82)
05:38:31:813 ==> "the book is: {\\"number\\":2,\\"title\\":\\"RPCs for Noobs\\",\\"topic\\":\\"distributed systems\\"
)"
```

At second time: (caching) 4m

The screenshot shows the Postman interface for the same GET request to `{{Front_IP}}/info`. The headers are identical to the first request. The response is shown in the Body tab, formatted as JSON:

```
{
  "number": 2,
  "title": "RPCs for Noobs",
  "topic": "distributed systems",
  "cost": 25,
  "stock": 200
}
```

Status: 200 OK, Time: 4 ms, Size: 356 B

(Previously cached data used; no new server request initiated.)

```
INFO] 31-12-2023 05:40:05:159 ==> "*****" --- (FrontEnd\index.js:38)
INFO] 31-12-2023 05:40:05:159 ==> "*****" --- (FrontEnd\index.js:45)
INFO] 31-12-2023 05:40:05:160 ==> "Cache lookup successful - Request found" --- (FrontEnd\index.js:48)
```

**Search req:** at first time (no cache) This request was processed by Server following the implementation of load balancing.

response time: 68 milliseconds before caching, with load

New Collection / DosHW2 / search

GET {{Frontend\_IP}}/search

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Key	Value	Description
Accept	*/*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
topic	LAB2	

Body Cookies Headers (8) Test Results

Status: 200 OK Time: 68 ms Size: 541 B Save as example

Pretty Raw Preview Visualize JSON

```
[
  {
    "number": 5,
    "title": "How to finish Project 3 on time",
    "topic": "LAB2",
    "cost": 29,
    "stock": 100
  },
  {
    "number": 6,
    "title": "Why theory classes are so hard",
    "topic": "LAB2",
    "cost": 34,
    "stock": 150
  },
  {
    "number": 7,
    "title": "Spring in the Pioneer Valley"
```

```
[INFO] 31-12-2023 07:05:56:078 ==> "server is running on port 6000" --- (FrontEnd\index.js:127)
[INFO] 31-12-2023 07:06:02:480 ==> "*****" --- (FrontEnd\index.js:38)
[INFO] 31-12-2023 07:06:02:481 ==> "*****" --- (FrontEnd\index.js:45)
[INFO] 31-12-2023 07:06:02:482 ==> "Cache doesn't contain the requested data" --- (FrontEnd\index.js:54)
[INFO] 31-12-2023 07:06:02:483 ==> "Querying books related to the topic LAB2" --- (FrontEnd\index.js:61)
[INFO] 31-12-2023 07:06:02:484 ==> "Exploring on server : http://localhost:4000" --- (FrontEnd\index.js:62)
[INFO] 31-12-2023 07:06:02:535 ==> "Results for the Search : [{\"number\":5,\"title\": \"How to finish Project 3 on time\", \"stock\":100},{\"number\":6,\"title\": \"Why theory classes are so hard\", \"topic\": \"LAB2\", \"cost\":34, \"stock\":150},{\"number\":7,\"title\": \"Spring in the Pioneer Valley\", \"topic\": \"LAB2\", \"cost\":24, \"stock\":45}]" --- (FrontEnd\index.js:68)
```

With caching: 5 ms

GET {{Front\_IP}}/search

Accept \*/\*

Accept-Encoding gzip, deflate, br

Connection keep-alive

topic LAB2

Status: 200 OK Time: 5 ms Size: 541 B

```
[{"number": 5, "title": "How to finish Project 3 on time", "topic": "LAB2", "cost": 29, "stock": 100}, {"number": 6, "title": "Why theory classes are so hard", "topic": "LAB2", "cost": 34, "stock": 150}, {"number": 7, "title": "Spring in the Pioneer Valley"}
```

```
[INFO] 31-12-2023 07:07:40:882 ==> "*****" --- (FrontEnd\index.js:38)
[INFO] 31-12-2023 07:07:40:884 ==> "Cache lookup successful - Request found" --- (FrontEnd\index.js:41)
[INFO] 31-12-2023 07:07:43:170 ==> "*****" --- (FrontEnd\index.js:38)
[INFO] 31-12-2023 07:07:43:171 ==> "Cache lookup successful - Request found" --- (FrontEnd\index.js:41)
```

## Purchase req: Data Before Purchase, Ensuring Consistency:

purchase for the item with ID = 1, two times:

First attempt using a different IP than the second.

```
[INFO] 31-12-2023 05:25:26:901 ==> "Start a purchase order for the book with number: 1" --- (FrontEnd\index.js:108)
[INFO] 31-12-2023 05:25:26:901 ==> "Order processing on the server: http://192.168.116.1:5000" --- (FrontEnd\index.js:109)
[INFO] 31-12-2023 05:25:26:985 ==> "Successful payment for the selected item." --- (FrontEnd\index.js:115)
[INFO] 31-12-2023 05:27:30:868 ==> "*****" --- (FrontEnd\index.js:105)
[INFO] 31-12-2023 05:27:30:869 ==> "Start a purchase order for the book with number: 1" --- (FrontEnd\index.js:108)
[INFO] 31-12-2023 05:27:30:869 ==> "Order processing on the server: http://localhost:5000" --- (FrontEnd\index.js:109)
[INFO] 31-12-2023 05:27:30:961 ==> "Successful payment for the selected item." --- (FrontEnd\index.js:115)
[INFO] 31-12-2023 05:30:46:753 ==> "*****" --- (FrontEnd\index.js:38)
```

## Database State in Catalog After Two Purchase Requests from Server: (Stock Decreased by 2)

```
sqlite> SELECT * FROM books;
1|How to get a good grade in DOS in 40 minutes a day|distributed systems|90|171
2|RPCs for Noobs|distributed systems|25|200
3|Xen and the Art of Surviving Undergraduate School|undergraduate school|123|58
4|Cooking for the Impatient Undergrad|undergraduate school|15|105
5|How to finish Project 3 on time|LAB2|29|100
6|Why theory classes are so hard|LAB2|34|150
7|Spring in the Pioneer Valley|LAB2|24|45
sqlite> SELECT * FROM books;
1|How to get a good grade in DOS in 40 minutes a day|distributed systems|90|169
2|RPCs for Noobs|distributed systems|25|200
3|Xen and the Art of Surviving Undergraduate School|undergraduate school|123|58
4|Cooking for the Impatient Undergrad|undergraduate school|15|105
5|How to finish Project 3 on time|LAB2|29|100
6|Why theory classes are so hard|LAB2|34|150
7|Spring in the Pioneer Valley|LAB2|24|45
sqlite> |
```

## Experimental Evaluation and Measurements:

- Compute the average response time (query/buy) of your new systems. What is the response time with and without caching? How much does caching help?

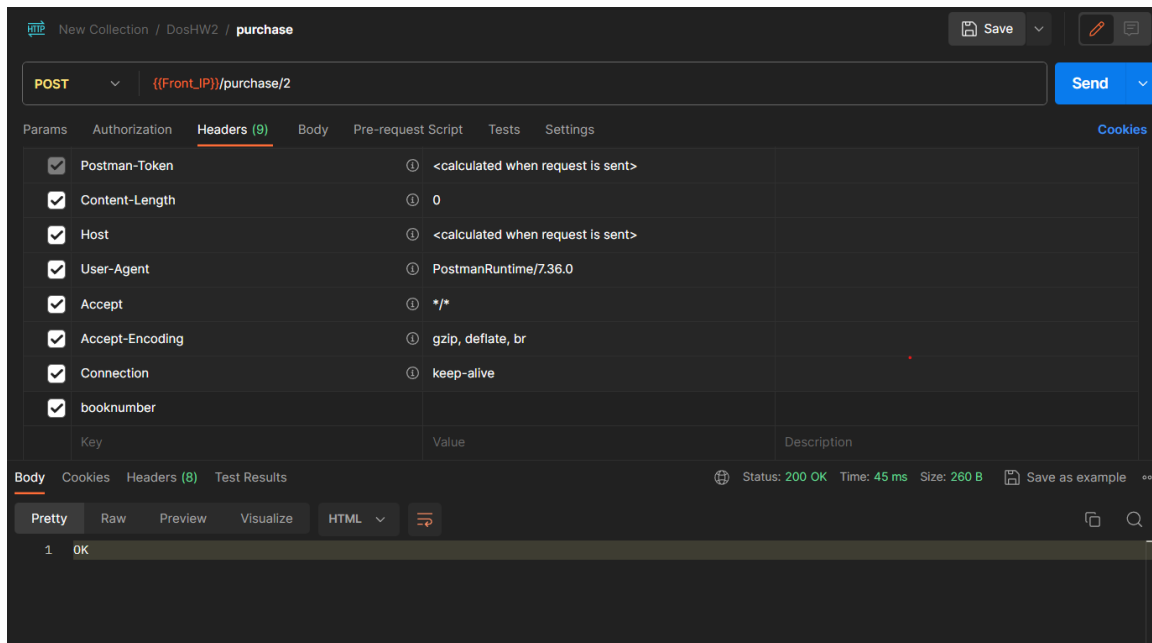
	Time without cache	Time with cache
Info req	15 ms	4 ms
Search req	68 ms	5 ms

Performance:  $((15+68)/2) / ((5+4)/2) = 41.5 / 4.5 = 9.2222$

it helps with 9 times better performance.



- Construct a simple experiment that issues orders or catalog updates (i.e., database writes) to invalidate the cache and maintain cache consistency. What are the overhead of cache consistency operations? What is the latency of a subsequent request that sees a cache miss?



During the purchase phase, updating the price took approximately 45ms, involving cache updates and consistency checks across catalog replicas. This process contributes to increased overhead, particularly in the purchase and update operation, as cache consistency introduces complexities.

The latency is notably affected in cases of cache misses, where data retrieval from backend servers incurs a substantial time cost, resulting in extended response times for requests encountering cache misses.

In first part about docker build images for three servers and run containers













Show three containers

```
PS D:\DosProject01a\FrontEnd> docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
8e42d3a5270f	express-frontend-app	"docker-entrypoint.s..."	2 seconds ago	Up 2 seconds	0.0.0.0:6000->6000/tcp	express-frontend-app
22bc8627fc0f	express-order-app	"docker-entrypoint.s..."	5 minutes ago	Up 5 minutes	0.0.0.0:5000->5000/tcp	express-order-app-co
ab29a2d3d7ac	express-catalog-app	"docker-entrypoint.s..."	12 minutes ago	Up 12 minutes	0.0.0.0:4000->4000/tcp	express-catalog-app-

Search

Only show running containers

	Name	Image	Status	CPU (%)	Port(s)	Actions
<input type="checkbox"/>	 <a href="#">express-808d711d</a>	<a href="#">express-orc</a>	Running	0%	<a href="#">5000:5000</a>	  
<input type="checkbox"/>	 <a href="#">express-25f94683</a>	<a href="#">express-ca</a>	Running	0.03%	<a href="#">4000:4000</a>	  
<input type="checkbox"/>	 <a href="#">express-540aa0c9</a>	<a href="#">express-fro</a>	Running	0%	<a href="#">6000:6000</a>	  

And if I run images ls these three images:

```
PS D:\DosProject01a\FrontEnd> docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
express-frontend-app	latest	85ab86ce030c	15 minutes ago	1.13GB
express-order-app	latest	2e38847f0ec5	16 minutes ago	1.13GB
express-catalog-app	latest	f36ee310fce7	18 minutes ago	1.13GB
catalog-node-app	latest	8d2e88c13061	23 hours ago	1.13GB

CONCLUSION:

In summary, the implemented replication, caching, load balancing, and consistency mechanisms have notably improved system performance and fault tolerance. However, challenges arise during operations requiring cache consistency, leading to increased overhead and latency for updates. Despite these challenges, the system demonstrates enhanced reliability and efficiency in scenarios where the benefits of replication and caching outweigh the complexities introduced.