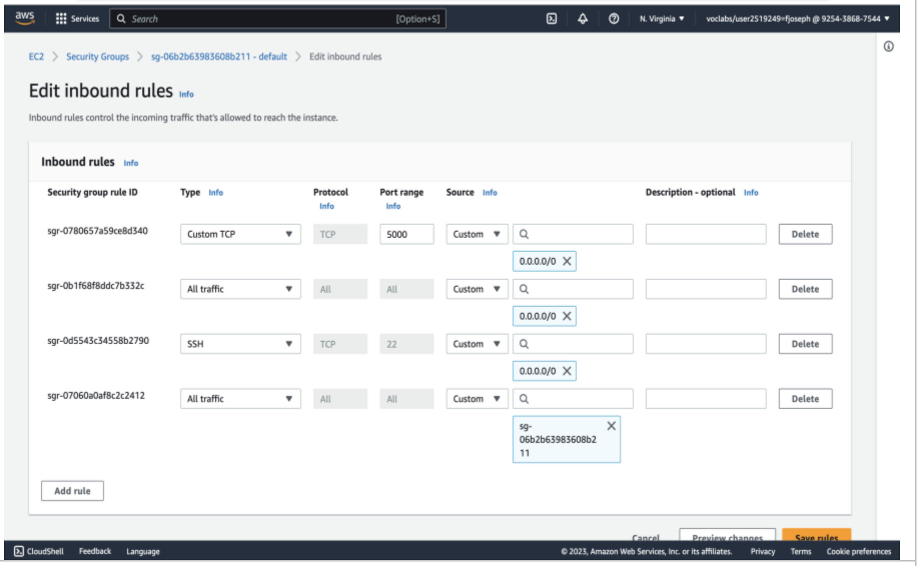
Evaluation Document

Steps to deploy application in AWS

Once Installing AWS CLI , Copy the credentials from AWS details in learner Lab and save it under .aws/credentials folder in the system. Also save the PEM file into the working directory.
The below commands are executed to deploy the instance and application on AWS.

1. Configure AWS Settings	aws configure
2. Create an `m5a.xlarge` instance in the `us-east-1` region on AWS	aws ec2 run-instances --image-id ami-0d73480446600f555 --instance-type m5a.large --key-name vockey > instance.json
3. To find public DNS Name	aws ec2 describe-instances --instance-id i-0ef4c1c6bb0a0d418 o/p : "PublicDnsName": " ec2-54-226-103-56.compute-1.amazonaws.com",
4. setting the right permission for the PEM key.	chmod 400 labuser.pem
5. allows ssh access from anywhere	aws ec2 authorize-security-group-ingress --group-name default --protocol tcp --port 22 --cidr 0.0.0.0/0
6. Accessing EC2 instance created via SSH	ssh -i labsuser.pem ubuntu@ec2-54-226-103-56.compute-1.amazonaws.com
7. Upgrading pip and Installing flask on instance	sudo apt-get update sudo apt install python3-pip Pip3 install flask
8. Copy the folder from local machine to ec2 instance	scp -i labsuser.pem -r "IdeaProjects/DOS-677/Lab3/lab-3-asterix-and-double-trouble-femimol-priyanka/Flask" ubuntu@ec2-54-226-103-56.compute-1.amazonaws.com:my_dir

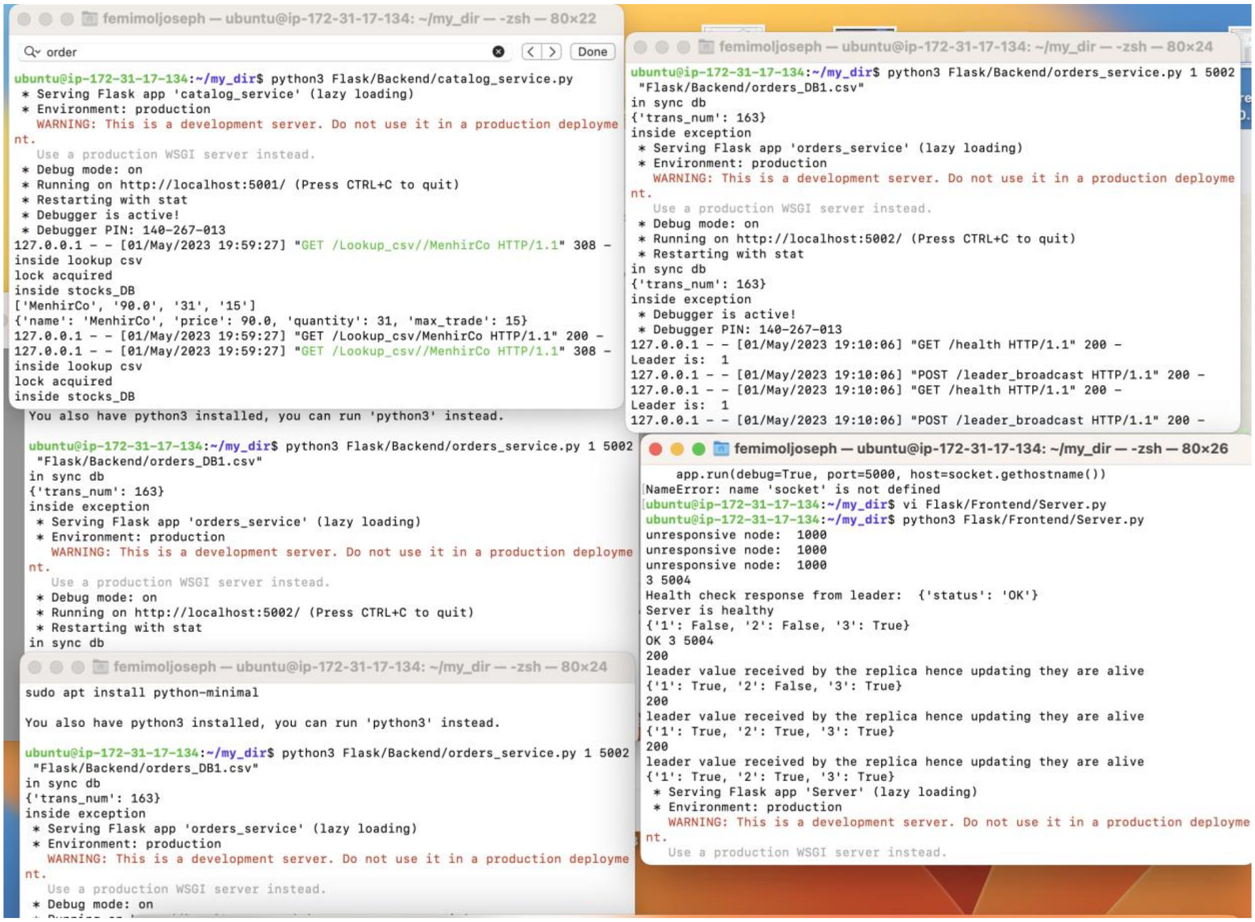
<table>
<tr>
<td>9. Modified security group inbound rules to allow all traffic.</td>
<td>


EC2 > Security Groups > sg-06b2b63983608b211 - default > Edit inbound rules
<b>Edit inbound rules</b> Info
Inbound rules control the incoming traffic that's allowed to reach the instance.
<b>Inbound rules</b> Info
<table>
<tr><th>Security group rule ID</th><th>Type Info</th><th>Protocol Info</th><th>Port range Info</th><th>Source Info</th><th></th><th>Description - optional Info</th><th></th></tr>
<tr><td>sgr-0780657a59ce8d340</td><td>Custom TCP</td><td>TCP</td><td>5000</td><td>Custom</td><td>0.0.0.0/0</td><td></td><td>Delete</td></tr>
<tr><td>sgr-0b1f68f8ddc7b332c</td><td>All traffic</td><td>All</td><td>All</td><td>Custom</td><td>0.0.0.0/0</td><td></td><td>Delete</td></tr>
<tr><td>sgr-0d5543c34558b2790</td><td>SSH</td><td>TCP</td><td>22</td><td>Custom</td><td>0.0.0.0/0</td><td></td><td>Delete</td></tr>
<tr><td>sgr-07060a0af8c2c2412</td><td>All traffic</td><td>All</td><td>All</td><td>Custom</td><td>sg-06b2b63983608b211</td><td></td><td>Delete</td></tr>
</table>

</td>
</tr>
<tr>
<td>10. Started Order servers on multiple shells.</td>
<td>• Python3 Flask/Backend/orders_service.py 1 5002 "Flask/Backend/orders_DB1.csv"
• Python3 Flask/Backend/orders_service.py 2 5003 "Flask/Backend/orders_DB2.csv"
• Python3 Flask/Backend/orders_service.py 3 5004 "Flask/Backend/orders_DB3.csv"</td>
</tr>
<tr>
<td>11. Started catalog and Frontend Server</td>
<td>• Python3 Flask/Backend/catalog_service.py
• Python3 Flask/Frontend/Server.py</td>
</tr>
</table>

- Modify the client to connect to the frontend with the Public IPv4 address (*54.226.103.56*).
- Modify frontend server.py and catalogserver.py to get hostname of frontend by socket.gethostname().
- Finally,multiclient.sh has been executed for different p values from 0.0 to 0.8 incremented by 0.2 for 5 clients concurrently and recorded the latencies for each type of request.

Screenshots of Microservices deployed in AWS



femimoljoseph — ubuntu@ip-172-31-17-134: ~/my_dir — -zsh — 80×22
order
Done

```
ubuntu@ip-172-31-17-134:~/my_dir$ python3 Flask/Backend/catalog_service.py
 * Serving Flask app 'catalog_service' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://localhost:5001/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 140-267-013
127.0.0.1 - - [01/May/2023 19:59:27] "GET /Lookup_csv//MenhirCo HTTP/1.1" 308 -
inside lookup csv
lock acquired
inside stocks_DB
['MenhirCo', '90.0', '31', '15']
{'name': 'MenhirCo', 'price': 90.0, 'quantity': 31, 'max_trade': 15}
127.0.0.1 - - [01/May/2023 19:59:27] "GET /Lookup_csv/MenhirCo HTTP/1.1" 200 -
127.0.0.1 - - [01/May/2023 19:59:27] "GET /Lookup_csv//MenhirCo HTTP/1.1" 308 -
inside lookup csv
lock acquired
inside stocks_DB
```

femimoljoseph — ubuntu@ip-172-31-17-134: ~/my_dir — -zsh — 80×24

```
ubuntu@ip-172-31-17-134:~/my_dir$ python3 Flask/Backend/orders_service.py 1 5002
 "Flask/Backend/orders_DB1.csv"
in sync db
{'trans_num': 163}
inside exception
 * Serving Flask app 'orders_service' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://localhost:5002/ (Press CTRL+C to quit)
 * Restarting with stat
in sync db
{'trans_num': 163}
inside exception
 * Debugger is active!
 * Debugger PIN: 140-267-013
127.0.0.1 - - [01/May/2023 19:10:06] "GET /health HTTP/1.1" 200 -
Leader is:  1
127.0.0.1 - - [01/May/2023 19:10:06] "POST /leader_broadcast HTTP/1.1" 200 -
127.0.0.1 - - [01/May/2023 19:10:06] "GET /health HTTP/1.1" 200 -
Leader is:  1
127.0.0.1 - - [01/May/2023 19:10:06] "POST /leader_broadcast HTTP/1.1" 200 -
```

```
 You also have python3 installed, you can run 'python3' instead.

ubuntu@ip-172-31-17-134:~/my_dir$ python3 Flask/Backend/orders_service.py 1 5002
 "Flask/Backend/orders_DB1.csv"
in sync db
{'trans_num': 163}
inside exception
 * Serving Flask app 'orders_service' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://localhost:5002/ (Press CTRL+C to quit)
 * Restarting with stat
in sync db
```

femimoljoseph — ubuntu@ip-172-31-17-134: ~/my_dir — -zsh — 80×26

```
    app.run(debug=True, port=5000, host=socket.gethostname())
NameError: name 'socket' is not defined
ubuntu@ip-172-31-17-134:~/my_dir$ vi Flask/Frontend/Server.py
ubuntu@ip-172-31-17-134:~/my_dir$ python3 Flask/Frontend/Server.py
unresponsive node:  1000
unresponsive node:  1000
unresponsive node:  1000
3 5004
Health check response from leader:  {'status': 'OK'}
Server is healthy
{'1': False, '2': False, '3': True}
OK 3 5004
200
leader value received by the replica hence updating they are alive
{'1': True, '2': False, '3': True}
200
leader value received by the replica hence updating they are alive
{'1': True, '2': True, '3': True}
200
leader value received by the replica hence updating they are alive
{'1': True, '2': True, '3': True}
 * Serving Flask app 'Server' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
```

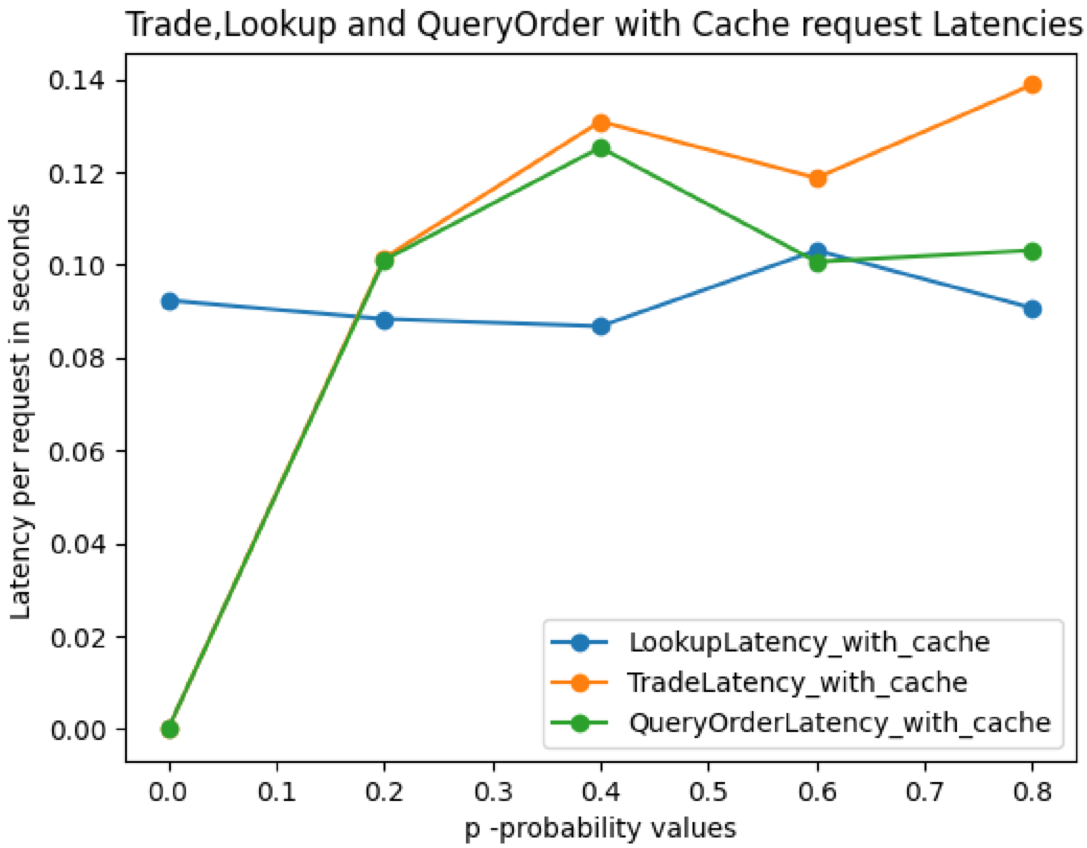
femimoljoseph — ubuntu@ip-172-31-17-134: ~/my_dir — -zsh — 80×24

```
sudo apt install python-minimal

You also have python3 installed, you can run 'python3' instead.

ubuntu@ip-172-31-17-134:~/my_dir$ python3 Flask/Backend/orders_service.py 1 5002
 "Flask/Backend/orders_DB1.csv"
in sync db
{'trans_num': 163}
inside exception
 * Serving Flask app 'orders_service' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: on
```

Measurement Results and Plots

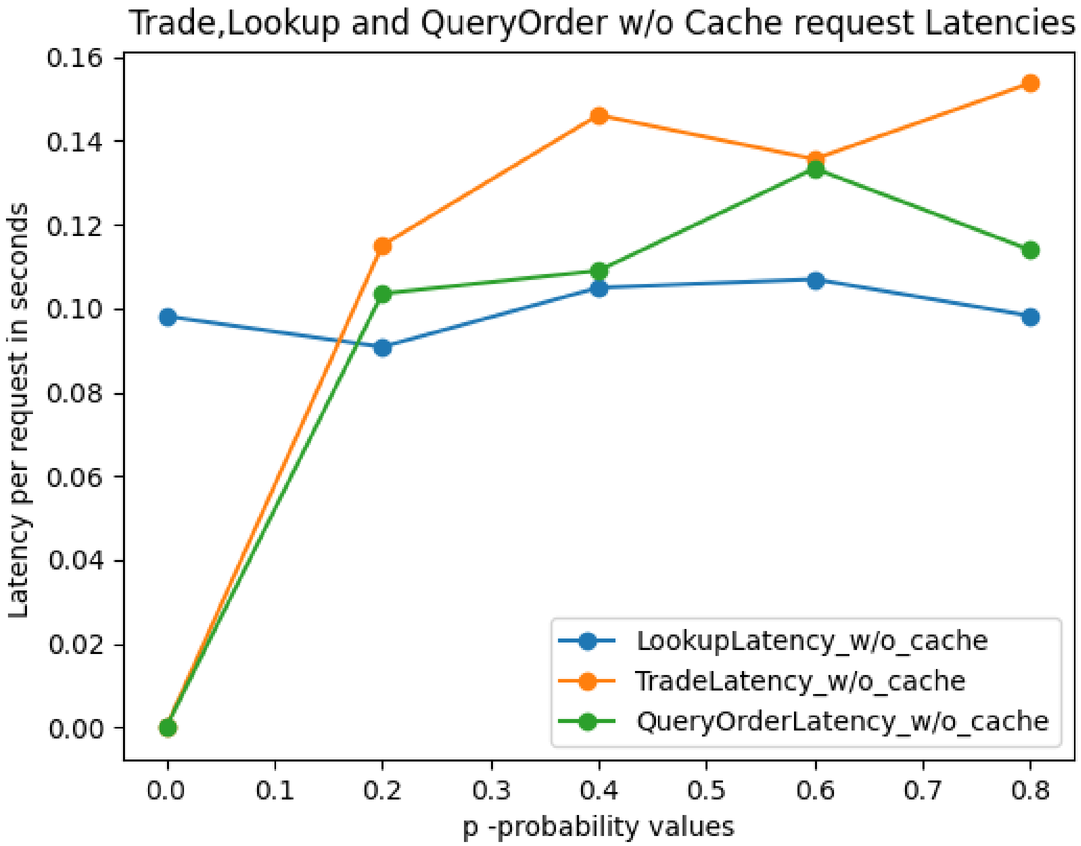
Latency of different type of requests with cache ON

Values of P = [0.0,0.2,0.4,0.6,0.8]
Corresponding lookup latencies =
[0.09238722076,0.08837939248,0.08680263334999999,0.10314589375000006,
0.09073009303333335]
Corresponding trade latencies =
[0,0.10152078325,0.13092528625,0.11875795747916663,0.1388598229]
Corresponding query latencies =
[0,0.10109291700000012,0.11531389599999994,0.10073018320000009,0.1031739029999999
1]



Latency of different type of requests without cache

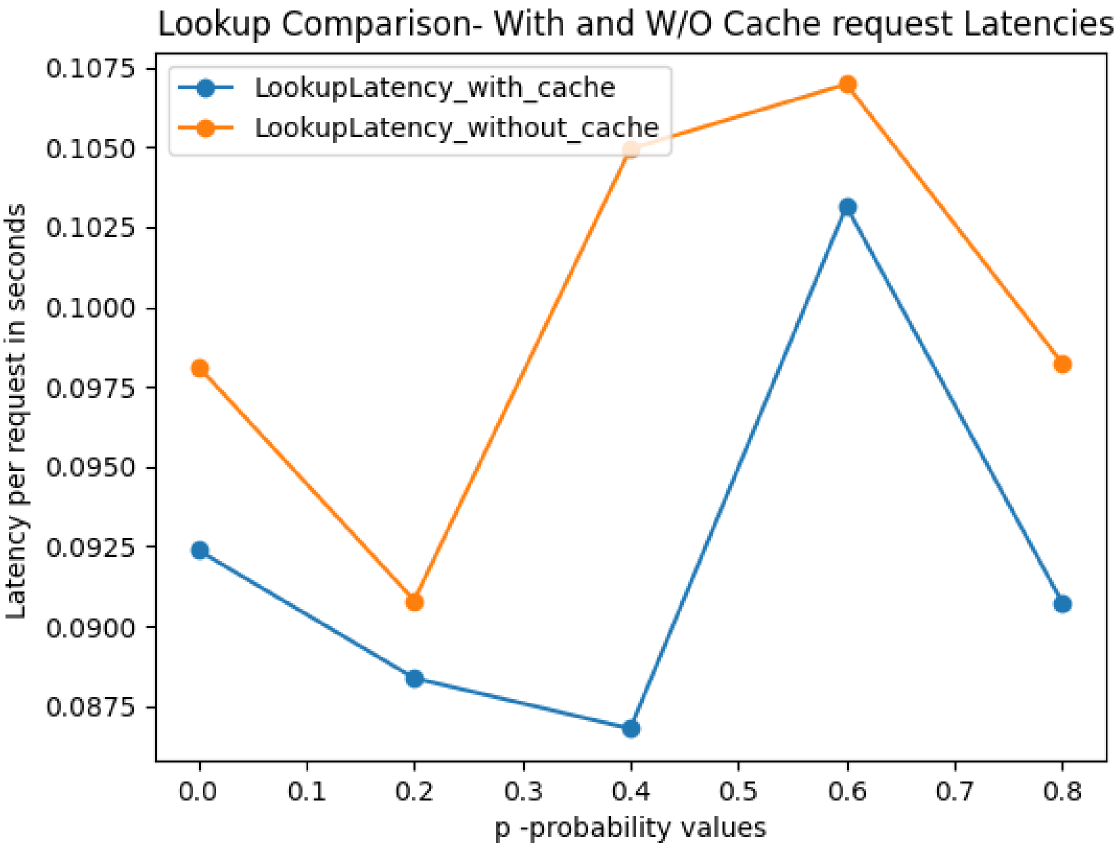
Values of P = [0.0,0.2,0.4,0.6,0.8]
Corresponding lookup latencies =
[0.09814131338000001,0.09081838609999998,0.10495242915,0.10697311422,0.09822577090000004]
Corresponding trade latencies =
[0,0.11504124999999998,0.14613660844999998,0.13564629721333338,0.15374233350000002]
Corresponding query latencies =
[0,0.10353120850000008,0.10900712295000003,0.13338719433333326,0.11393465266666676]



Comparison of With and Without Cache Lookup requests

Lookup latency w/o cache = [0.09814131338000001,0.09081838609999998,0.10495242915,0.10697311422,0.09822577090000004]
Lookup latency with cache = [0.09238722076,0.08837939248,0.08680263334999999,0.10314589375000006, 0.09073009303333335]

The lookup request with cache is faster than without cache lookup request. The latency is significantly reduced due to less response time while caching previous lookup responses in the frontend server. The lookup request with cache at p=0.6 is 0.1031 where, w/o cache is 0.1069.



Questions

12. simulate crash failures by killing a random order service replica while the client is running, and then bring it back online after some time. Repeat this experiment several times and make sure that you test the case when the leader is killed. Can the clients notice the failures? (either during order requests or the final order checking phase) or are they transparent to the clients?

    Answer: No. The clients are not able to notice the failures at any time and client work smoothly irrespective of the leader crashes. Fault tolerance is working as expected.

13. Do all the order service replicas end up with the same database file?
    Answer:

    Yes, the database is synced correctly whenever a transaction happening in orders service across all replicas using our new sync_DB functionality added.