Evaluation Document Lab 3

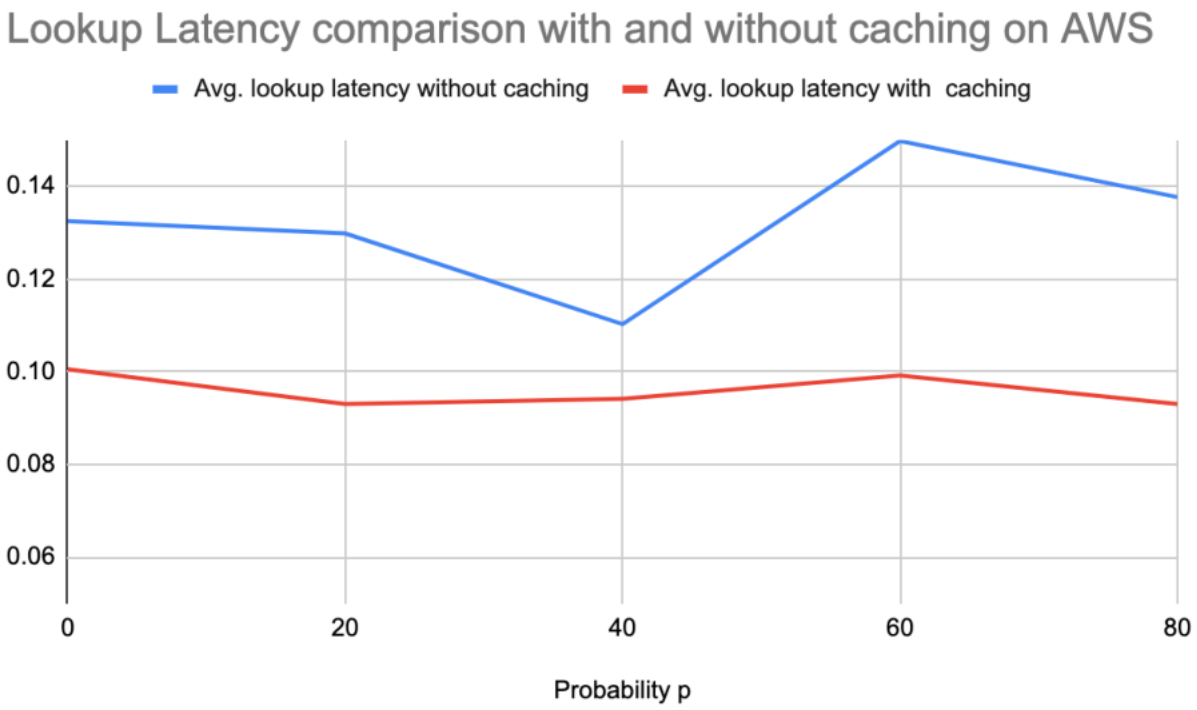
Deploying the application on AWS

1. Configure your AWS credentials by using AWS Learner Academy creds and inputting them on $HOME/.aws/credentials.
2. Use aws configure to set the region and instance type. We are using us-east-1 and json type respectively.
3. Now, create an instance of m5a.large type by using an ubuntu image.
4. On running describe-instance command, we will get the public-dns url of that instance. We would need that url to validate our services whether they are running or not and to perform load tests.
5. Also, from the AWS learner academy, download the PEM file which we will utilize to for ssh login into the ubuntu image.
6. Use the command ssh -i labuser.pem ubuntu@<public_dns_name> to login into it.
7. Once, install, please install docker and docker-compose on the image by running the following command:
    a. sudo apt update
    b. sudo apt install apt-transport-https ca-certificates curl software-properties-common
    c. curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
    d. echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
    e. sudo apt update
    f. sudo apt install docker-ce docker-ce-cli containerd.io
    g. Check docker installed by running this command : docker --version
    h. To install docker-compose, follow these:
    i. sudo curl -L https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose
    j. sudo chmod +x /usr/local/bin/docker-compose
    k. docker-compose version
8. Once, docker is set up, use git clone <git-url>.
9. Checkout to the correct branch.
10. Run sudo docker-compose build to build the service and then sudo docker-compose up -d to start them.

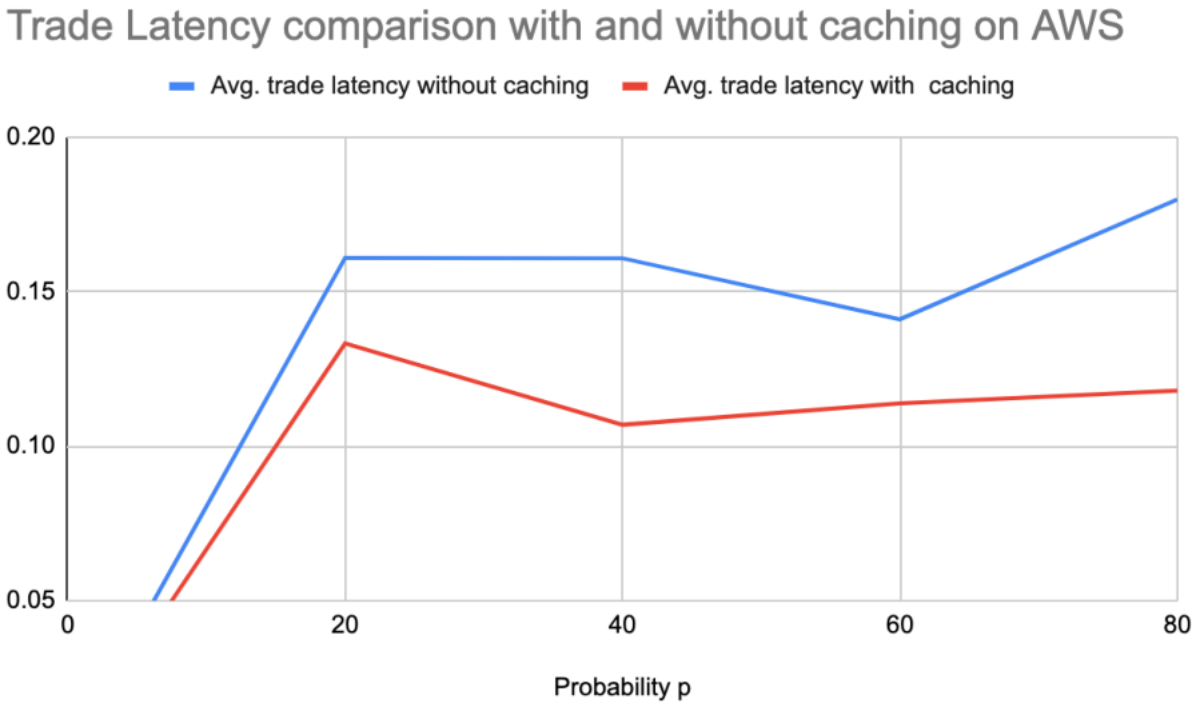
11. Use the public dns name and the relevant api contracts to get the desired outcomes

Graphs and Tables:

Probability p	Avg. lookup latency without caching	Avg. lookup latency with caching
0	0.13245967	0.1005565882
20	0.129845823	0.09302734733
40	0.1102920457	0.09418708682
60	0.1497795571	0.09919638038
80	0.1376590123	0.09302669764



Probability p	Avg. trade latency without caching	Avg. trade latency with caching
0	0	0
20	0.1608954353	0.1331902842
40	0.1607435012	0.1068954468
60	0.1409723987	0.1138203208
80	0.1797624537	0.1179550091



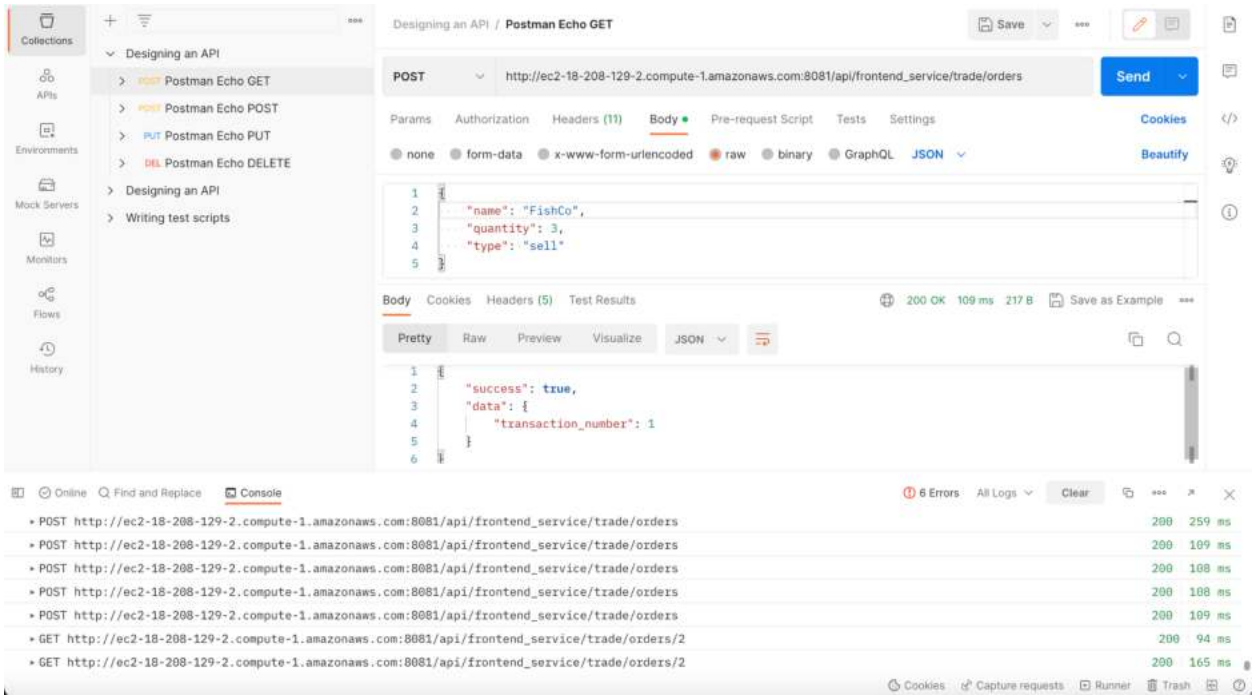
Q) how much benefits does caching provide by comparing the results?

Ans) On comparing all the above values and graphs, we can clearly see that enabling caching has resulted in better latency times and improved the overall performance

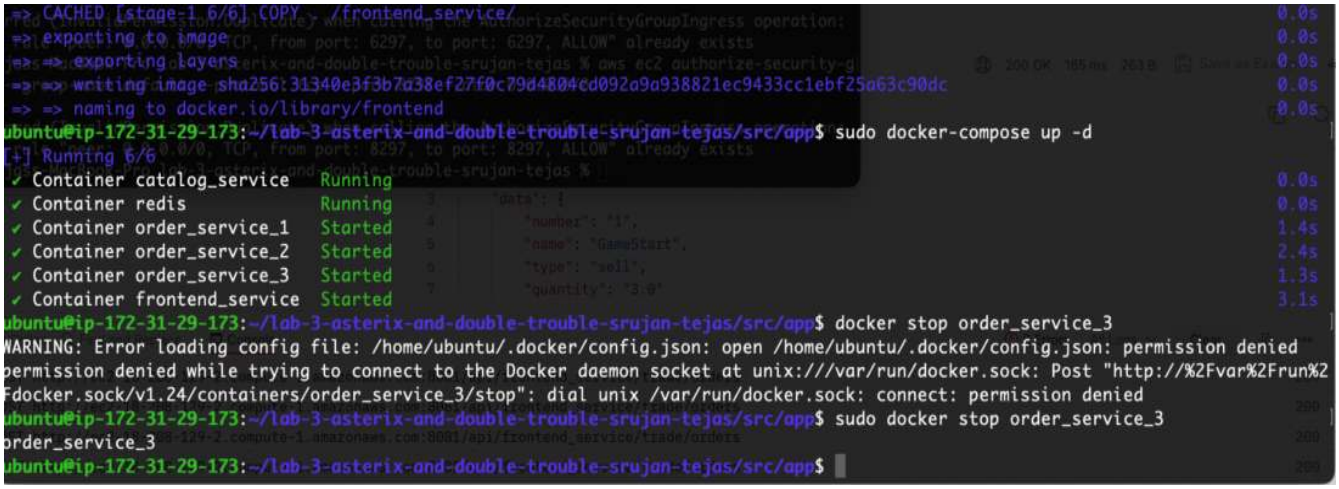
Q) Simulate crash failures by killing a random order service replica while the client is running, and then bring it back online after some time. Repeat this experiment several times and make sure that you test the case when the leader is killed. Can the clients notice the failures? or are they transparent to the clients? Do all the order service replicas end up with the same database file?

Ans)

GET calls before killing replicas-

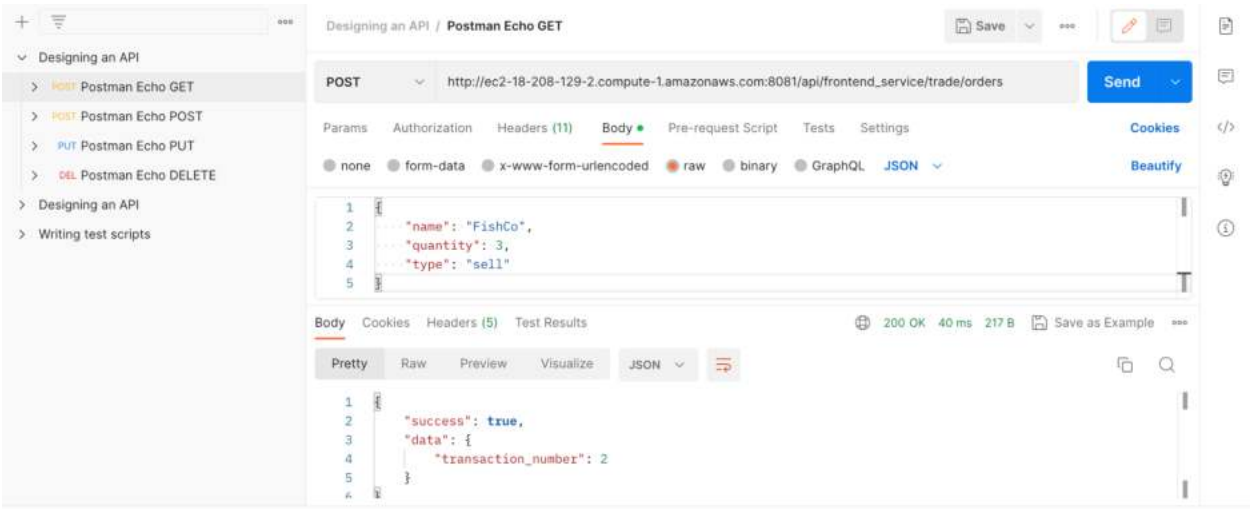


Killing an order service replica-



.

GET calls after killing replica-



As we can see from the above screenshots and also redoing this several times, it is pretty clear that clients do not notice the failures, and all the order service replicas end up with the same database file since we have implemented sync_db method which ensures that all replicas have the same database file which is crucial if a failure takes place.