EVALUATION DOCUMENT

PLOTS SHOWING LATENCY OF DIFFERENT TYPES OF REQUESTS

After testing in LOCAL MACHINE :

With CACHE turned on-

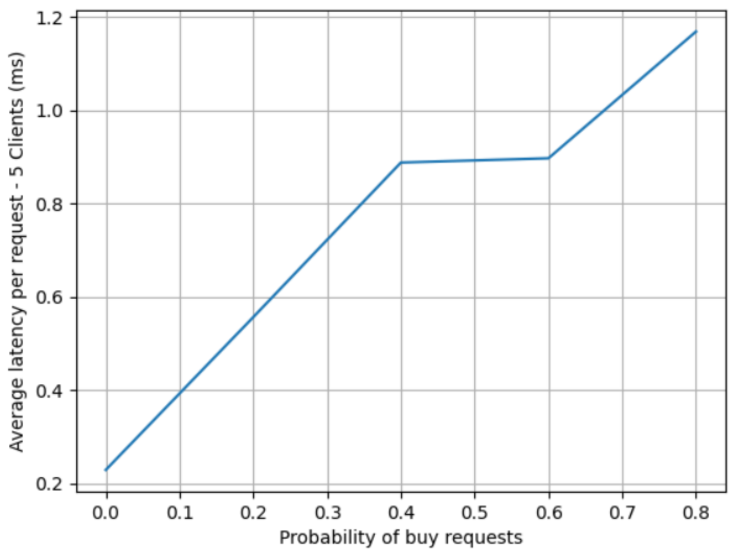


Fig 1(a) - Graph displaying the average latency per requests for 5 clients in milliseconds vs probability of a follow up purchase request when cache is turned on in local machine

P(0) = 0.25 ms

P(0.2) = 0.58 ms

P(0.4) = 0.88 ms

P(0.6) = 0.89 ms

P(0.8) = 1.18 ms

Without CACHE -

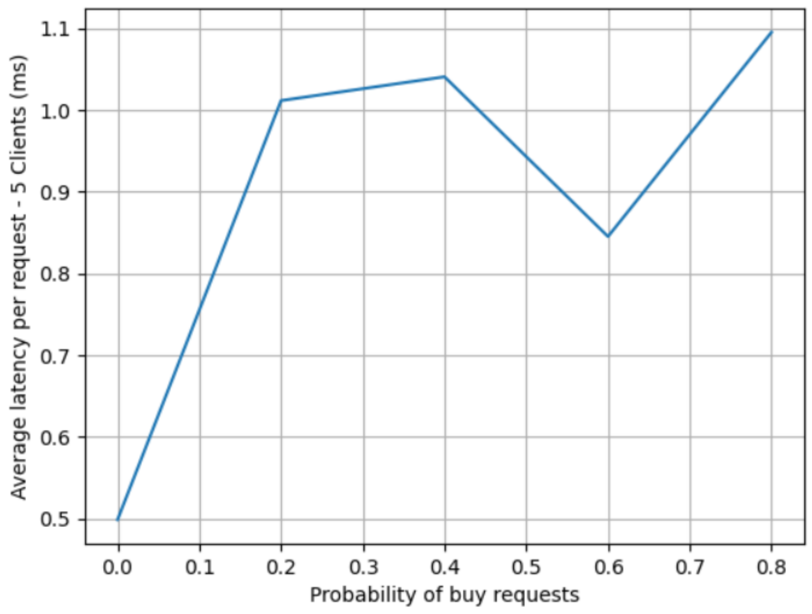


Fig 1(b) - Graph displaying the average latency per requests for 5 clients in milliseconds vs probability of a follow up purchase request when cache is turned off in local machine

P(0) = 0.50 ms

P(0.2) = 1.01 ms

P(0.4) = 1.04 ms

P(0.6) = 0.85 ms

P(0.8) = 1.1 ms

After testing in AWS :

With CACHE turned on-

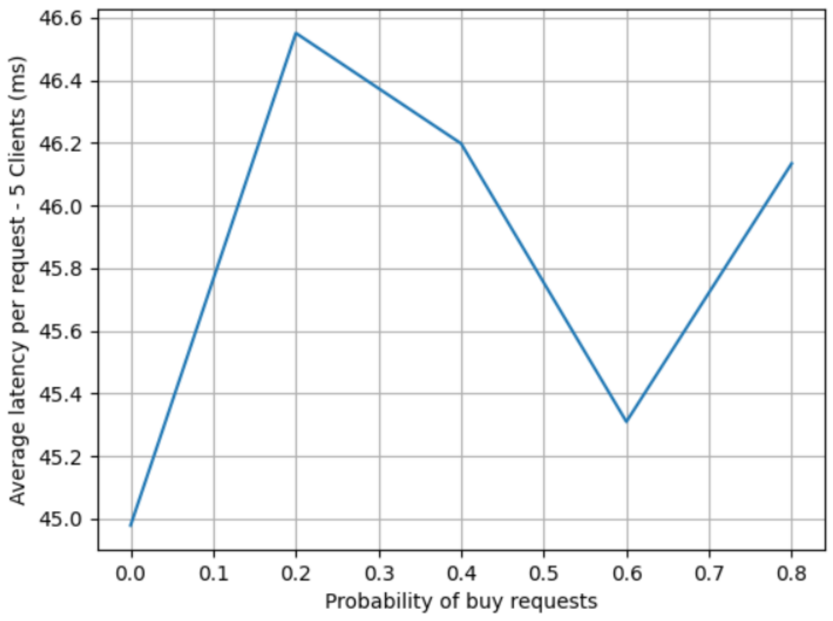


Fig 2(a) - Graph displaying the average latency per requests for 5 clients in milliseconds vs probability of a follow up purchase request when cache is turned on in AWS

P(0)	= 45 ms
P(0.2)	= 46.5 ms
P(0.4)	= 46.2 ms
P(0.6)	= 45.32 ms
P(0.8)	= 46.18 ms

With CACHE turned off-

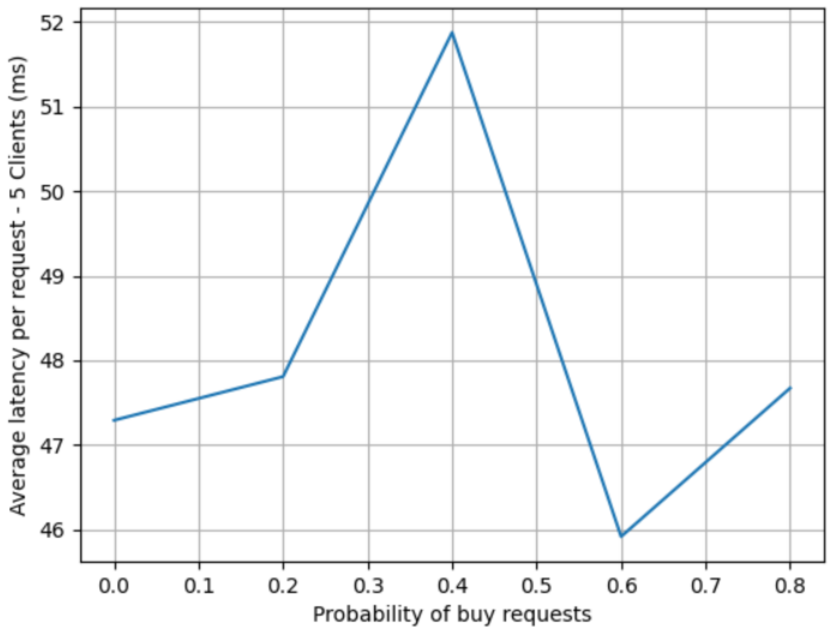


Fig 2(b) - Graph displaying the average latency per requests for 5 clients in milliseconds vs probability of a follow up purchase request when cache is turned off in AWS

P(0) = 47.4 ms

P(0.2) = 47.8 ms

P(0.4) = 51.9 ms

P(0.6) = 46 ms

P(0.8) = 47.8 ms

Comparing the results of the above experiments

After testing in LOCAL MACHINE - with cache and without cache :

Tested -	P(0)	P(0.2)	P(0.4)	P(0.6)	P(0.8)
With Cache on(in ms)	0.25	0.58	0.88	0.89	1.18
With Cache off(in ms)	0.50	1.01	1.04	0.85	1.1

According to our observations, when the application is tested in a local machine, the average latencies of the application is varying from 0.25 ms to 1.18 ms when the cache is turned on and it's varying from 0.50 ms to 1.1 ms when the cache is turned off. This clearly depicts the advantages of having caching. Caching increases the performance of the application by serving users with cached output and decreases the round trips to the server. The application's performance is also displaying the same.

After testing in AWS - with cache and without cache :

Tested -	P(0)	P(0.2)	P(0.4)	P(0.6)	P(0.8)
With Cache on(in ms)	45	46.5	46.2	45.32	46.18
With Cache off(in ms)	47.4	47.8	51.9	46	47.8

According to our observations, when the application is tested in AWS, the average latencies of the application is varying from 45 ms to 46.18 ms when the cache is turned on and it's varying from 47.4 ms to 47.8 ms when the cache is turned off. The average latencies are not showing any particular trend in general with caching turned on and off. The advantage of caching is clearly displayed above. The average latencies of the application is slightly less when the cache is turned on compared to the average latencies of the application when cache is turned off.

Comparison of results as buy probability increases -

We can see a clear increase in the latency on local, when the buy probability increases. But we are seeing few dips in the latency wrt buy probability graph for AWS. The latency evaluation on an AWS instance mostly depends on the routing latency or the internet traffic. Hence we are not seeing a clear increase in the graph. We are seeing different graphs every time we run the latency test on an AWS machine.

Comparison the results between local machine and AWS -

According to our observations, the average latency of the application is less when the application is tested locally (in both cases - with cache turned on and off). It varies between 0.25 ms to 1.18 ms when the application is tested locally. The average latency is varying between 45 ms to 47.8 ms when the application is tested in AWS (in both cases - with cache turned on and off). Since the requests have to be served by the application deployed in AWS, the requests should be routed to the AWS server, which takes more time . So, the average latency is high when an application is tested in AWS.

What happens when a replica goes down ?

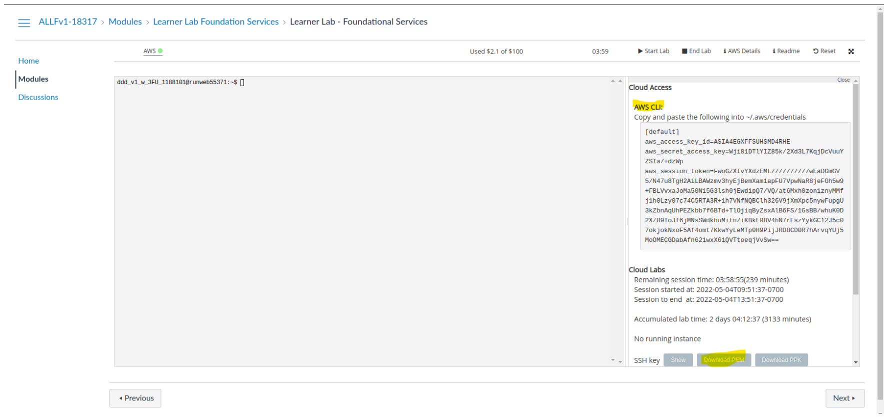
We have observed that the client does not notice the failures when a replica goes down. We have not observed any impact on the client. When the front service finds that the leader node is not responsive, it will redo the leader election and select the node with the highest id number. This process is completely transparent to the clients.

All the services that are available at a certain point of time, have the same order log with them as they are in synchronization. And as soon as a crashed service comes back online, its logs are updated with all the missing order details that were placed when it was crashed and its logs will be in sync with the other orders services logs, which means all the order service replicas end up with the same order log files.

Steps for Deploying and testing the application in AWS:

Instructions on how to deploy the application on AWS:

Step 1 : After starting the lab, click on AWS Details and get the credentials(key, secret) and copy the code and save it to $HOME/.aws/credentials on Linux



```
ddd_v1_w_3FU_1188101@runweb55371:~$
```

Cloud Access

AWS CLI:

Copy and paste the following into ~/.aws/credentials

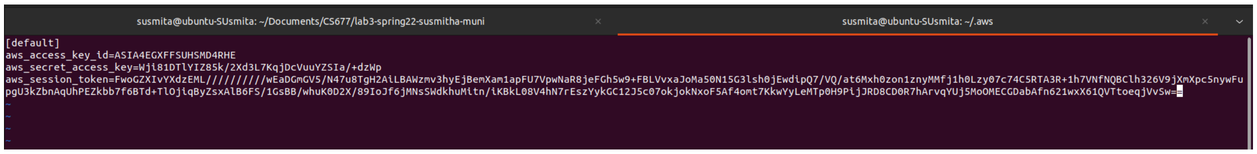
```
[default]
aws_access_key_id=ASIA4EGXFFSUHSMD4RHE
aws_secret_access_key=Wji81DTlYIZ85k/2Xd3L7KqjDcVuuY
ZSIa/+dzWp
aws_session_token=FwoGZXIvYXdzEML//////////wEaDGmGV
5/N47u8TgH2AiLBAWzmv3hyEjBemXam1apFU7VpwNaR8jeFGh5w9
+FBLVvxaJoMa50N15G3lsh0jEwdipQ7/VQ/at6Mxh0zon1znyMMf
j1h0Lzy07c74C5RTA3R+1h7VNfNQBClh326V9jXmXpc5nywFupgU
3kZbnAqUhPEZkbb7f6BTd+TlOjiqByZsxAlB6FS/1GsBB/whuK0D
2X/89IoJf6jMNsSWdkhuMitn/iKBkL08V4hN7rEszYykGC12J5c0
7okjokNxoF5Af4omt7KkwYyLeMTp0H9PijJRD8CD0R7hArvqYUj5
MoOMECGDabAfn621wxX61QVTtoeqjVvSw==
```

Cloud Labs

Remaining session time: 03:58:55(239 minutes)
Session started at: 2022-05-04T09:51:37-0700
Session to end at: 2022-05-04T13:51:37-0700

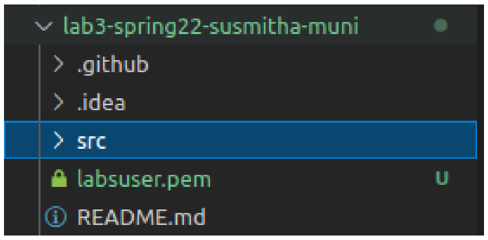
Accumulated lab time: 2 days 04:12:37 (3133 minutes)

No running instance

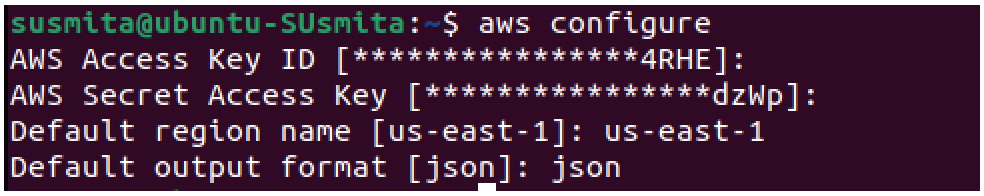

susmita@ubuntu-SUsmita: ~/Documents/CS677/lab3-spring22-susmitha-muni
susmita@ubuntu-SUsmita: ~/.aws

```
[default]
aws_access_key_id=ASIA4EGXFFSUHSMD4RHE
aws_secret_access_key=Wji81DTlYIZ85k/2Xd3L7KqjDcVuuYZSIa/+dzWp
aws_session_token=FwoGZXIvYXdzEML//////////wEaDGmGV5/N47u8TgH2AiLBAWzmv3hyEjBemXam1apFU7VpwNaR8jeFGh5w9+FBLVvxaJoMa50N15G3lsh0jEwdipQ7/VQ/at6Mxh0zon1znyMMfj1h0Lzy07c74C5RTA3R+1h7VNfNQBClh326V9jXmXpc5nywFu
pgU3kZbnAqUhPEZkbb7f6BTd+TlOjiqByZsxAlB6FS/1GsBB/whuK0D2X/89IoJf6jMNsSWdkhuMitn/iKBkL08V4hN7rEszYykGC12J5c07okjokNxoF5Af4omt7KkwYyLeMTp0H9PijJRD8CD0R7hArvqYUj5MoOMECGDabAfn621wxX61QVTtoeqjVvSw==
~
~
~
~
```

Step 2 : Download the “labsuser.pem” file and keep it in the current working directory



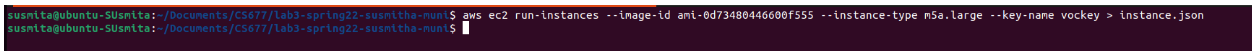
Step 3 : Configure AWS settings by running the command - “aws configure”



```
susmita@ubuntu-SUsmita:~$ aws configure
AWS Access Key ID [****************4RHE]:
AWS Secret Access Key [****************dzWp]:
Default region name [us-east-1]: us-east-1
Default output format [json]: json
```

Step 4 : Start the “m5a.large” EC2 instance by running the below command

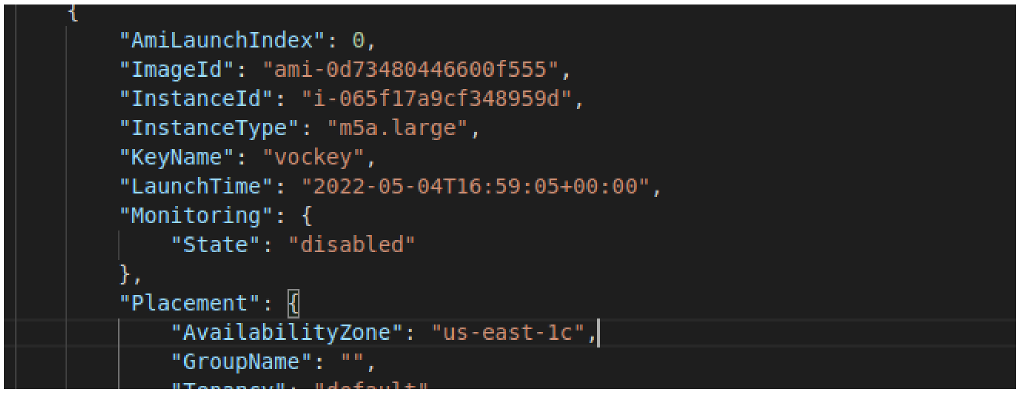
aws ec2 run-instances --image-id ami-0d73480446600f555 --instance-type m5a.large --key-name vockey > instance.json



```
susmita@ubuntu-SUsmita:~/Documents/CS677/lab3-spring22-susmitha-muni$ aws ec2 run-instances --image-id ami-0d73480446600f555 --instance-type m5a.large --key-name vockey > instance.json
susmita@ubuntu-SUsmita:~/Documents/CS677/lab3-spring22-susmitha-muni$
```

Step 5 : Open the instance.json file and copy the instance-id

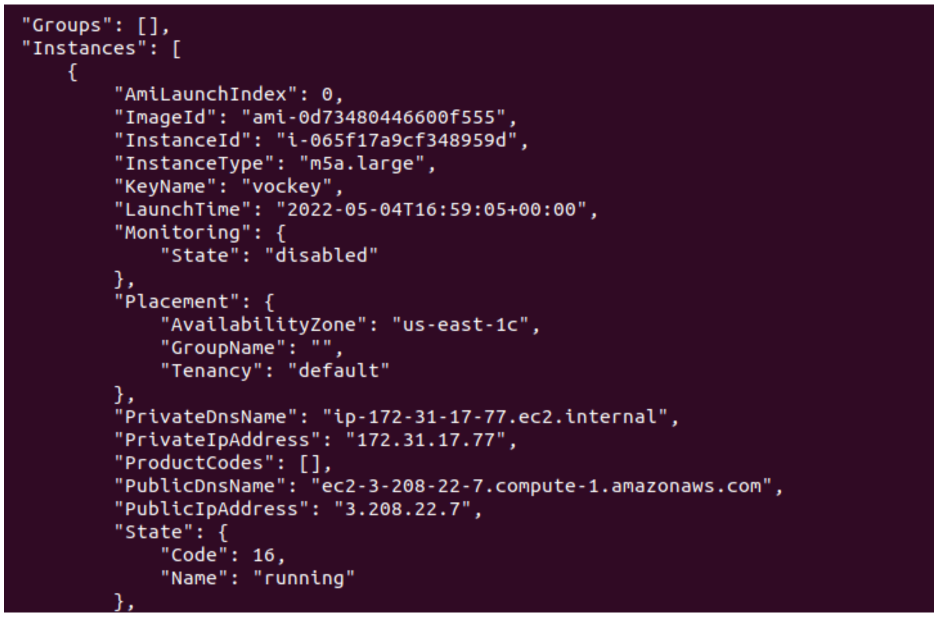
Created instance id - "i-065f17a9cf348959d",



```
        {
            "AmiLaunchIndex": 0,
            "ImageId": "ami-0d73480446600f555",
            "InstanceId": "i-065f17a9cf348959d",
            "InstanceType": "m5a.large",
            "KeyName": "vockey",
            "LaunchTime": "2022-05-04T16:59:05+00:00",
            "Monitoring": {
                "State": "disabled"
            },
            "Placement": {
                "AvailabilityZone": "us-east-1c",
                "GroupName": "",
```

Step 6 : Check the status of the instance by running the below command

aws ec2 describe-instances --instance-id <your-instance-id>



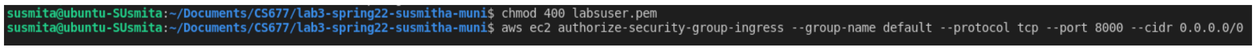
```
"Groups": [],
"Instances": [
    {
        "AmiLaunchIndex": 0,
        "ImageId": "ami-0d73480446600f555",
        "InstanceId": "i-065f17a9cf348959d",
        "InstanceType": "m5a.large",
        "KeyName": "vockey",
        "LaunchTime": "2022-05-04T16:59:05+00:00",
        "Monitoring": {
            "State": "disabled"
        },
        "Placement": {
            "AvailabilityZone": "us-east-1c",
            "GroupName": "",
            "Tenancy": "default"
        },
        "PrivateDnsName": "ip-172-31-17-77.ec2.internal",
        "PrivateIpAddress": "172.31.17.77",
        "ProductCodes": [],
        "PublicDnsName": "ec2-3-208-22-7.compute-1.amazonaws.com",
        "PublicIpAddress": "3.208.22.7",
        "State": {
            "Code": 16,
            "Name": "running"
        },
```

Step 7 : Copy the public DNS name and keep it as we're accessing the created EC2 instance using the public DNS name of the instance.
Public DNS name - ec2-3-208-22-7.compute-1.amazonaws.com

Step 8 : Now set the right permission for the PEM key by running the following command
chmod 400 labsuser.pem

Step 9 : We need to authorize port 8000 in (used by ssh) in the default security group

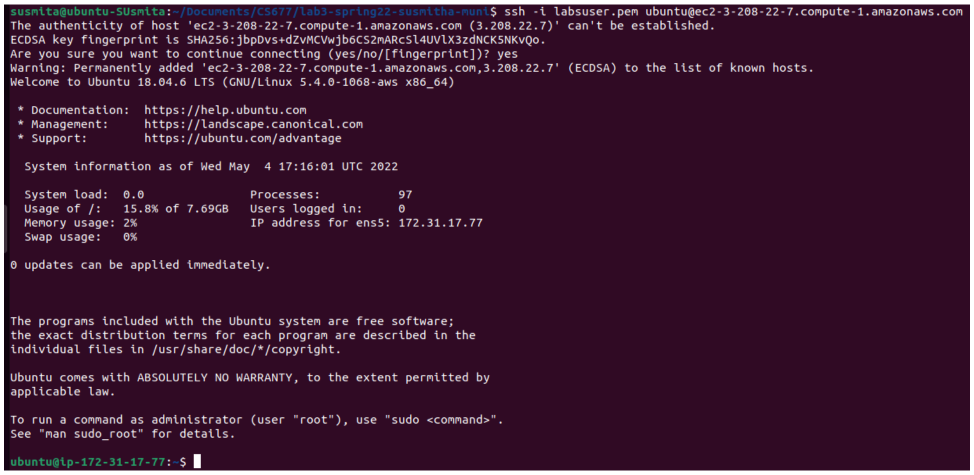
aws ec2 authorize-security-group-ingress --group-name default --protocol tcp --port 8000 --cidr 0.0.0.0/0



```
susmita@ubuntu-SUsmita:~/Documents/CS677/lab3-spring22-susmitha-muni$ chmod 400 labsuser.pem
susmita@ubuntu-SUsmita:~/Documents/CS677/lab3-spring22-susmitha-muni$ aws ec2 authorize-security-group-ingress --group-name default --protocol tcp --port 8000 --cidr 0.0.0.0/0
```

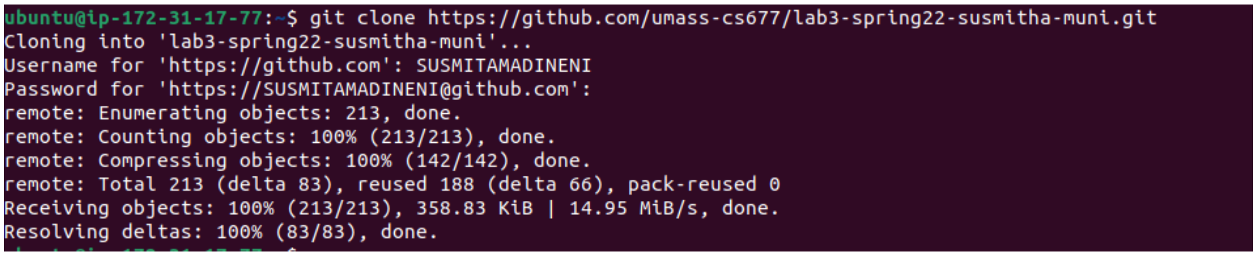
Step 10 : Now connect to the create EC2 instance by running the following command in the terminal

ssh -i labsuser.pem ubuntu@<your-instance's-public-DNS-name>



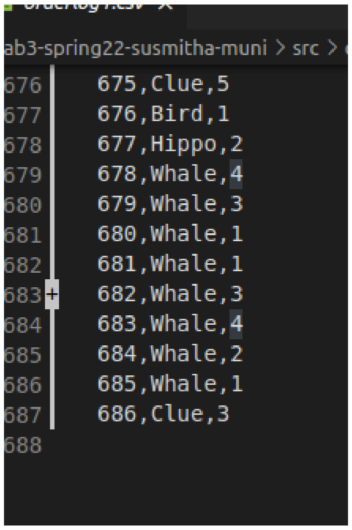
```
susmita@ubuntu-SUsmita:~/Documents/CS677/lab3-spring22-susmitha-muni$ ssh -i labsuser.pem ubuntu@ec2-3-208-22-7.compute-1.amazonaws.com
The authenticity of host 'ec2-3-208-22-7.compute-1.amazonaws.com (3.208.22.7)' can't be established.
ECDSA key fingerprint is SHA256:jbpDvs+dZvMCVwjb6CS2mARcSl4UVlX3zdNCK5NKvQo.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-208-22-7.compute-1.amazonaws.com,3.208.22.7' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 5.4.0-1068-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Wed May  4 17:16:01 UTC 2022

  System load:  0.0               Processes:           97
  Usage of /:   15.8% of 7.69GB   Users logged in:     0
  Memory usage: 2%                IP address for ens5: 172.31.17.77
  Swap usage:   0%

0 updates can be applied immediately.



The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-17-77:~$
```

Step 11 : Now clone all the code to the created instance using the git clone command -



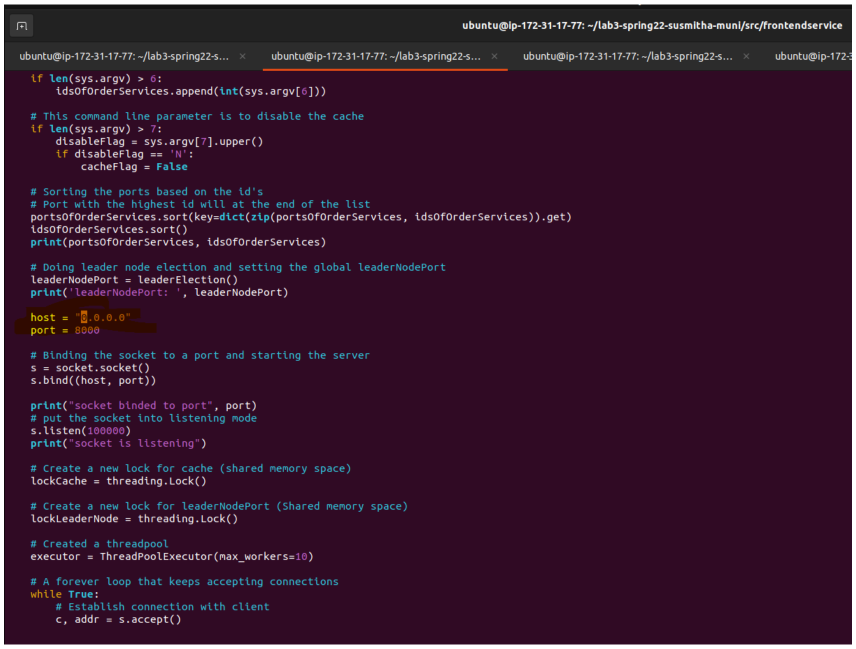
```
ubuntu@ip-172-31-17-77:~$ git clone https://github.com/umass-cs677/lab3-spring22-susmitha-muni.git
Cloning into 'lab3-spring22-susmitha-muni'...
Username for 'https://github.com': SUSMITAMADINENI
Password for 'https://SUSMITAMADINENI@github.com':
remote: Enumerating objects: 213, done.
remote: Counting objects: 100% (213/213), done.
remote: Compressing objects: 100% (142/142), done.
remote: Total 213 (delta 83), reused 188 (delta 66), pack-reused 0
Receiving objects: 100% (213/213), 358.83 KiB | 14.95 MiB/s, done.
Resolving deltas: 100% (83/83), done.
```

Step 12 : Now ssh into the instances and up the services as below -
Sample orderlog1.csv file -



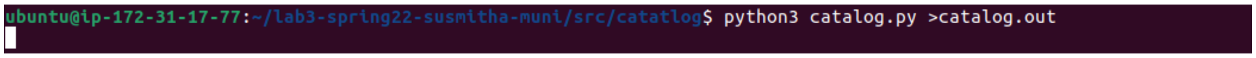
```
675,Clue,5
676,Bird,1
677,Hippo,2
678,Whale,4
679,Whale,3
680,Whale,1
681,Whale,1
682,Whale,3
683,Whale,4
684,Whale,2
685,Whale,1
686,Clue,3
```

For testing the application - With cache :
Before running the services , change the hostname of frontend as highlighted below -



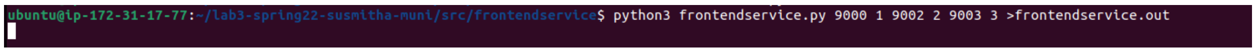
```
ubuntu@ip-172-31-17-77: ~/lab3-spring22-susmitha-muni/src/frontendservice
ubuntu@ip-172-31-17-77: ~/lab3-spring22-s...   ubuntu@ip-172-31-17-77: ~/lab3-spring22-s...   ubuntu@ip-172-31-17-77: ~/lab3-spring22-s...   ubuntu@ip-172-3
    if len(sys.argv) > 6:
        idsOfOrderServices.append(int(sys.argv[6]))

    # This command line parameter is to disable the cache
    if len(sys.argv) > 7:
        disableFlag = sys.argv[7].upper()
        if disableFlag == 'N':
            cacheFlag = False

    # Sorting the ports based on the id's
    # Port with the highest id will at the end of the list
    portsOfOrderServices.sort(key=dict(zip(portsOfOrderServices, idsOfOrderServices)).get)
    idsOfOrderServices.sort()
    print(portsOfOrderServices, idsOfOrderServices)

    # Doing leader node election and setting the global leaderNodePort
    leaderNodePort = leaderElection()
    print('leaderNodePort: ', leaderNodePort)

    host = "0.0.0.0"
    port = 8000

    # Binding the socket to a port and starting the server
    s = socket.socket()
    s.bind((host, port))

    print("socket binded to port", port)
    # put the socket into listening mode
    s.listen(100000)
    print("socket is listening")

    # Create a new lock for cache (shared memory space)
    lockCache = threading.Lock()

    # Create a new lock for leaderNodePort (Shared memory space)
    lockLeaderNode = threading.Lock()

    # Created a threadpool
    executor = ThreadPoolExecutor(max_workers=10)

    # A forever loop that keeps accepting connections
    while True:
        # Establish connection with client
        c, addr = s.accept()
```

a. Run the catalog service by going to the catatlog folder present in the src directory



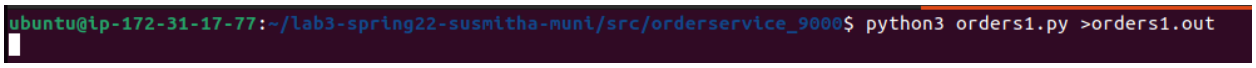
```
ubuntu@ip-172-31-17-77:~/lab3-spring22-susmitha-muni/src/catatlog$ python3 catalog.py >catalog.out
```

b. Run the frontendservice.py like below by navigating to the frontendservice folder in src directory



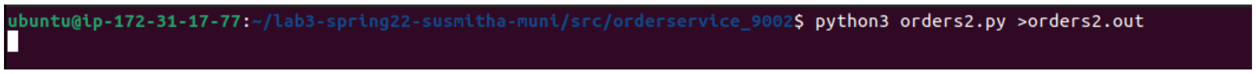
```
ubuntu@ip-172-31-17-77:~/lab3-spring22-susmitha-muni/src/frontendservice$ python3 frontendservice.py 9000 1 9002 2 9003 3 >frontendservice.out
```

c. Run the orderservices now by navigating to the respective folders
Orders 1 - Go to orderservice_9000 folder and run orders1.py file



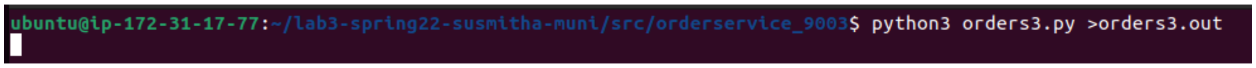
```
ubuntu@ip-172-31-17-77:~/lab3-spring22-susmitha-muni/src/orderservice_9000$ python3 orders1.py >orders1.out
```

Orders2 - Go to orderservice_9002 folder and run orders2.py



```
ubuntu@ip-172-31-17-77:~/lab3-spring22-susmitha-muni/src/orderservice_9002$ python3 orders2.py >orders2.out
```

Order3 - Got to orderservice_9003 folder and run order3.py

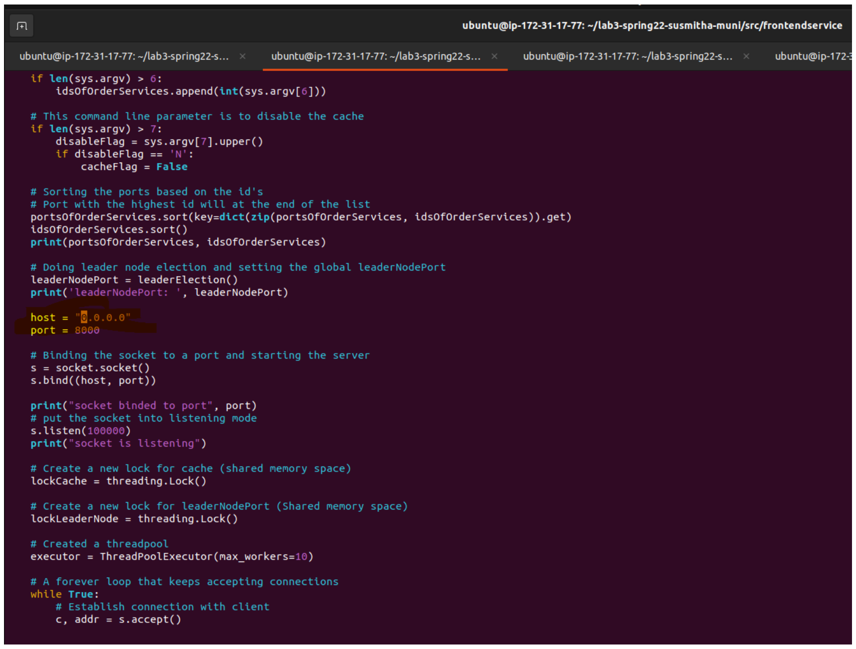


```
ubuntu@ip-172-31-17-77:~/lab3-spring22-susmitha-muni/src/orderservice_9003$ python3 orders3.py >orders3.out
```

d. Now run the infinite client and verify the output files generate for each service in the respective folders.(Output files generate are present in the docs directory for AWS)

For testing the application without cache -

Before running the services , change the hostname of frontend as highlighted below -


ubuntu@ip-172-31-17-77: ~/lab3-spring22-susmitha-muni/src/frontendservice
ubuntu@ip-172-31-17-77: ~/lab3-spring22-s...
ubuntu@ip-172-31-17-77: ~/lab3-spring22-s...
ubuntu@ip-172-31-17-77: ~/lab3-spring22-s...

```
if len(sys.argv) > 6:
    idsOfOrderServices.append(int(sys.argv[6]))

# This command line parameter is to disable the cache
if len(sys.argv) > 7:
    disableFlag = sys.argv[7].upper()
    if disableFlag == 'N':
        cacheFlag = False

# Sorting the ports based on the id's
# Port with the highest id will at the end of the list
portsOfOrderServices.sort(key=dict(zip(portsOfOrderServices, idsOfOrderServices)).get)
idsOfOrderServices.sort()
print(portsOfOrderServices, idsOfOrderServices)

# Doing leader node election and setting the global leaderNodePort
leaderNodePort = leaderElection()
print('leaderNodePort: ', leaderNodePort)

host = "0.0.0.0"
port = 8000

# Binding the socket to a port and starting the server
s = socket.socket()
s.bind((host, port))

print("socket binded to port", port)
# put the socket into listening mode
s.listen(100000)
print("socket is listening")

# Create a new lock for cache (shared memory space)
lockCache = threading.Lock()

# Create a new lock for leaderNodePort (Shared memory space)
lockLeaderNode = threading.Lock()

# Created a threadpool
executor = ThreadPoolExecutor(max_workers=10)

# A forever loop that keeps accepting connections
while True:
    # Establish connection with client
    c, addr = s.accept()
```

a. Run the catalog service by going to the catatlog folder present in the src directory

```
ubuntu@ip-172-31-17-77:~/lab3-spring22-susmitha-muni/src/catatlog$ python3 catalog.py >catalog.out
```

b. Run the frontendservice.py like below by navigating to the frontendservice folder in src directory and provide the following command

```
ubuntu@ip-172-31-17-77:~/lab3-spring22-susmitha-muni/src/frontendservice$ python3 frontendservice.py 9000 1 9002 2 9003 3 n > frontendservice.out
```

c. Run the orderservices now by navigating to the respective folders
Orders 1 - Go to orderservice_9000 folder and run orders1.py file

```
ubuntu@ip-172-31-17-77:~/lab3-spring22-susmitha-muni/src/orderservice_9000$ python3 orders1.py >orders1.out
```

Orders2 - Go to orderservice_9002 folder and run orders2.py

```
ubuntu@ip-172-31-17-77:~/lab3-spring22-susmitha-muni/src/orderservice_9002$ python3 orders2.py >orders2.out
```

Order3 - Got to orderservice_9003 folder and run order3.py

```
ubuntu@ip-172-31-17-77:~/lab3-spring22-susmitha-muni/src/orderservice_9003$ python3 orders3.py >orders3.out
```

d. Now run the infinite client and verify the output files generated for each service in the respective folders.(Output files generate are present in the docs directory for AWS)