COMPSCI 677 Spring 2022

Lab 3: Caching, Replication and Fault Tolerance

Team Members: Maoqin Zhu, Yixiang Zhang

Deployment on AWS

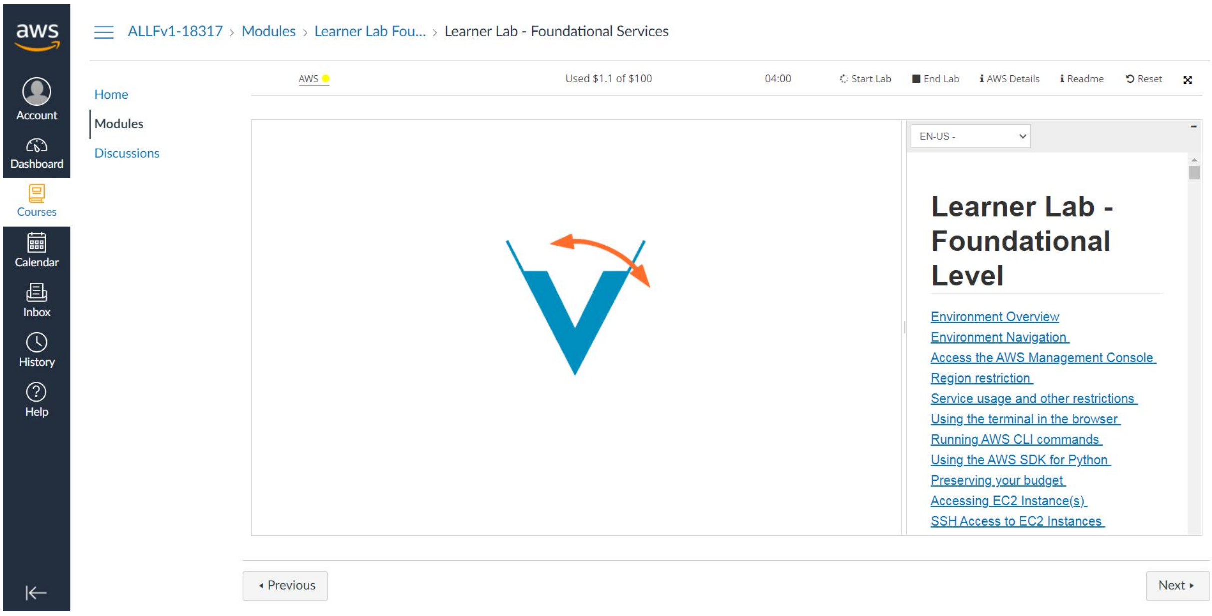
This could be a simple tutorial about how to deploy our online application on AWS Cloud. We are providing you enough details about the deployment configuration in this part.

Step 1- Install AWS CLI

Just follow the steps described in the following link:
https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

Step 2- Obtain AWS Crendentials.

For each UMass student, we can get our own credentials through AWS Academy. First we direct to this page, and just click "Start Lab" button as follows. It may take several minutes to start the AWS environment.

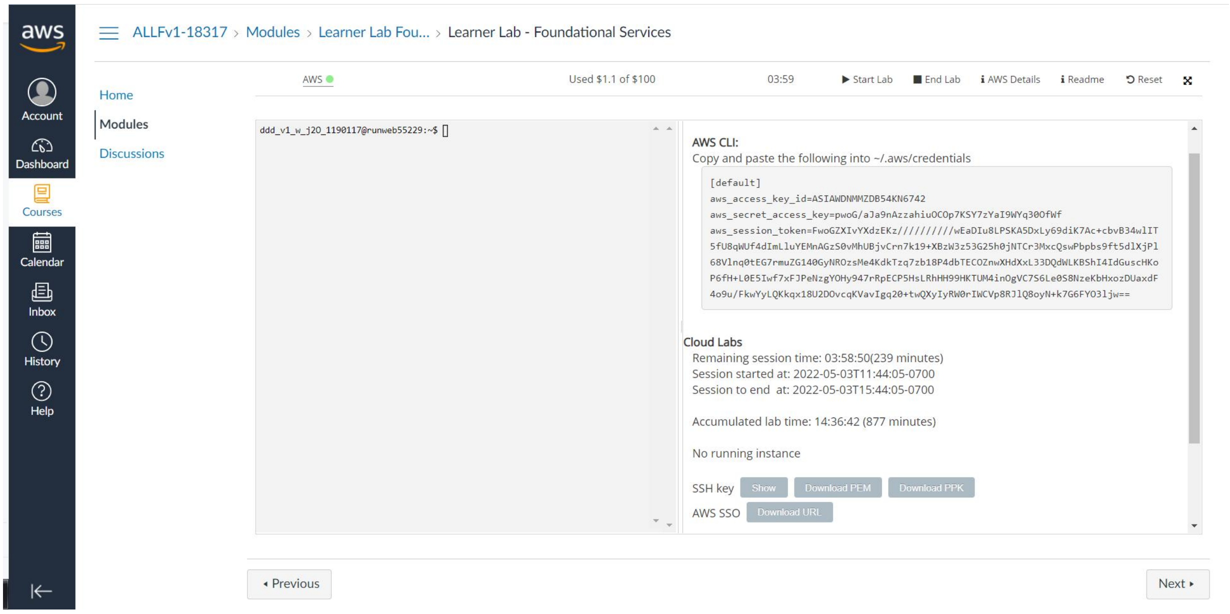

aws
Account
Dashboard
Courses
Calendar
Inbox
History
Help
ALLFv1-18317 › Modules › Learner Lab Fou... › Learner Lab - Foundational Services
Home
Modules
Discussions
AWS
Used $1.1 of $100
04:00
Start Lab
End Lab
AWS Details
Readme
Reset
EN-US
Learner Lab - Foundational Level
Environment Overview
Environment Navigation
Access the AWS Management Console
Region restriction
Service usage and other restrictions
Using the terminal in the browser
Running AWS CLI commands
Using the AWS SDK for Python
Preserving your budget
Accessing EC2 Instance(s)
SSH Access to EC2 Instances
Previous
Next

Click the "AWS Details" button, and then click the "Show" button following AWS CLI. The code that appeared in the text box below contains the credentials that the AWS CLI uses to interact with AWS.

What you should do on your Linux/macOS machine is that:

1) Copy the code and save it to **$HOME/.aws/ credentials**.

2) Download PEM key and save it to your working directory.



```
ddd_v1_w_j20_1190117@runweb55229:~$ []
```

AWS CLI:
Copy and paste the following into ~/.aws/credentials

```
[default]
aws_access_key_id=ASIAWDNMMZDB54KN6742
aws_secret_access_key=pwoG/aJa9nAzzahiuOCOp7KSY7zYaI9WYq30OfWf
aws_session_token=FwoGZXIvYXdzEKz//////////wEaDIu8LPSKA5DxLy69diK7Ac+cbvB34wlIT
5fU8qWUf4dImLluYEMnAGzS0vMhUBjvCrn7k19+XBzW3z53G25h0jNTCr3MxcQswPbpbs9ft5dlXjPl
68Vlnq0tEG7rmuZG140GyNROzsMe4KdkTzq7zb18P4dbTECOZnwXHdXxL33DQdWLKBShI4IdGuscHKo
P6fH+L0E5Iwf7xFJPeNzgYOHy947rRpECP5HsLRhHH99HKTUM4inOgVC7S6Le0S8NzeKbHxozDUaxdF
4o9u/FkwYyLQKkqx18U2DOvcqKVavIgq20+twQXyIyRW0rIWCVp8RJlQ8oyN+k7G6FYO3ljw==
```

Cloud Labs
Remaining session time: 03:58:50(239 minutes)
Session started at: 2022-05-03T11:44:05-0700
Session to end at: 2022-05-03T15:44:05-0700

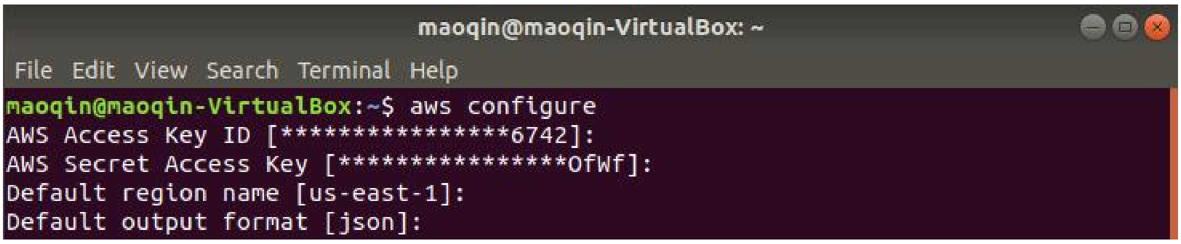
Accumulated lab time: 14:36:42 (877 minutes)

No running instance

SSH key Show Download PEM Download PPK
AWS SSO Download URL

Step 3- Configure Your AWS Settings

Simply run **$aws configure** on your machine as follows. By creating a credentials file in previous step, we can just press enter to skip certain processes.


maoqin@maoqin-VirtualBox: ~
File Edit View Search Terminal Help

```
maoqin@maoqin-VirtualBox:~$ aws configure
AWS Access Key ID [****************6742]:
AWS Secret Access Key [****************OfWf]:
Default region name [us-east-1]:
Default output format [json]:
```

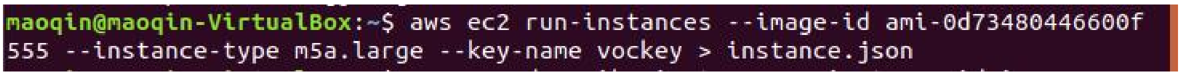
Step 4- Start Our EC2 Instance

When creating the EC2 instance, there are some options for you. Our selection in this lab are listed as below:

Instance Type: **m5a.large**

AMI ID: **ami-0d73480446600f555**

Hence we type the following command on our local machine.



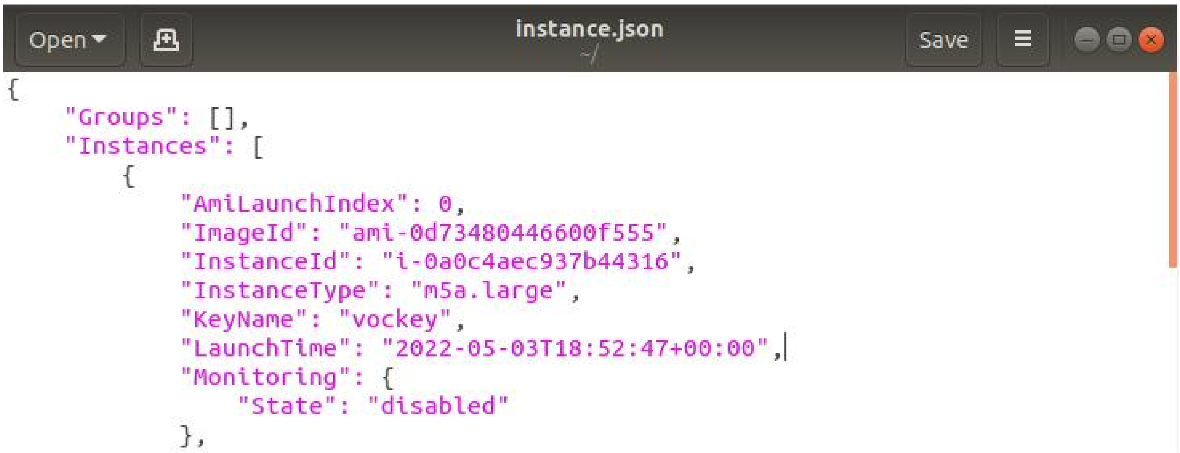
```
maoqin@maoqin-VirtualBox:~$ aws ec2 run-instances --image-id ami-0d73480446600f
555 --instance-type m5a.large --key-name vockey > instance.json
```

Step 5- Checkout the Public IP

First, type the following command on AWS Academy console:

```
$ aws ec2 describe-key-pairs
```

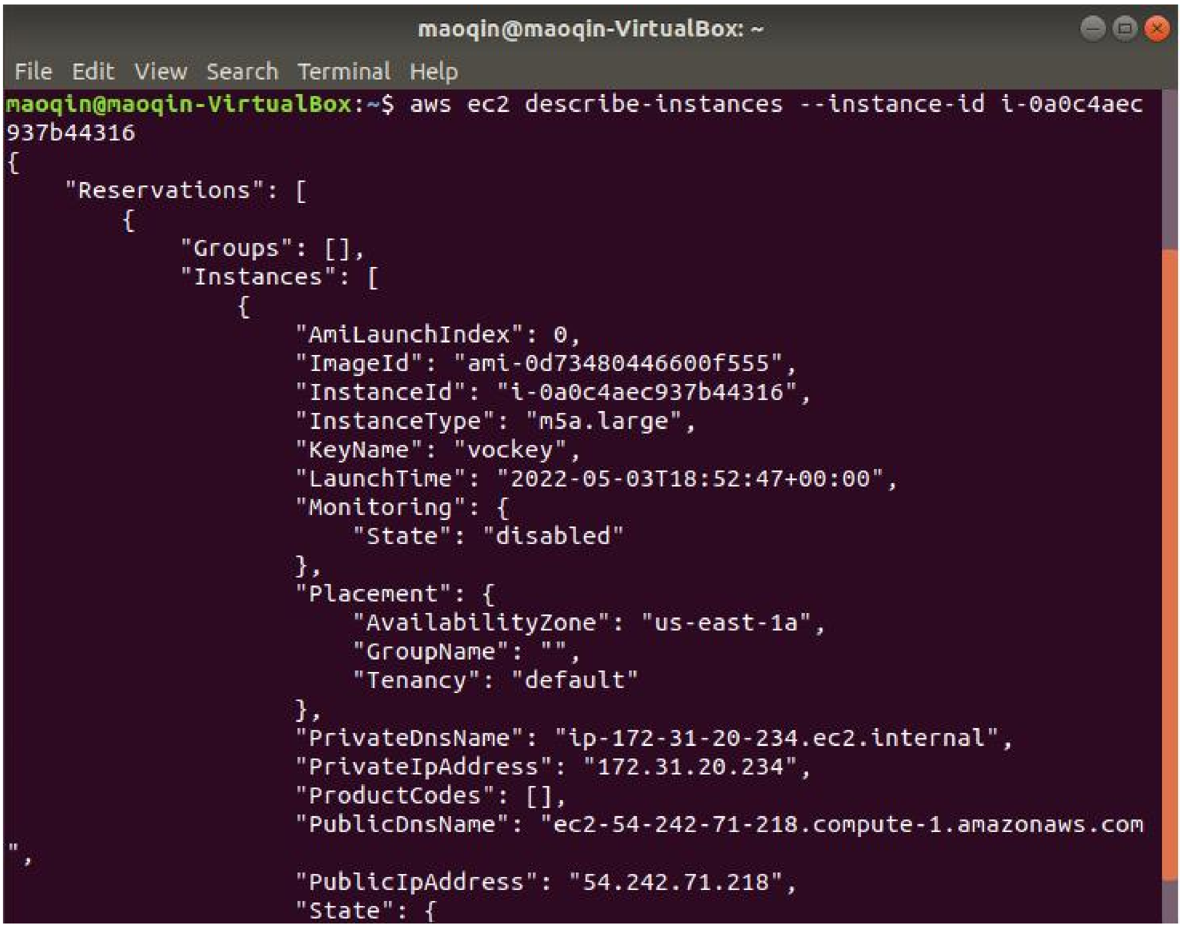
Then we will have an “instance.json” file on our local machine.



```
{
    "Groups": [],
    "Instances": [
        {
            "AmiLaunchIndex": 0,
            "ImageId": "ami-0d73480446600f555",
            "InstanceId": "i-0a0c4aec937b44316",
            "InstanceType": "m5a.large",
            "KeyName": "vockey",
            "LaunchTime": "2022-05-03T18:52:47+00:00",
            "Monitoring": {
                "State": "disabled"
            },
```

Second, according to our instance id, we can checkout the public IP address of our EC2 instance as follows:

```
$ aws ec2 describe-instances --instance-id <our-instance-id>
```


maoqin@maoqin-VirtualBox: ~
File Edit View Search Terminal Help

```
maoqin@maoqin-VirtualBox:~$ aws ec2 describe-instances --instance-id i-0a0c4aec
937b44316
{
    "Reservations": [
        {
            "Groups": [],
            "Instances": [
                {
                    "AmiLaunchIndex": 0,
                    "ImageId": "ami-0d73480446600f555",
                    "InstanceId": "i-0a0c4aec937b44316",
                    "InstanceType": "m5a.large",
                    "KeyName": "vockey",
                    "LaunchTime": "2022-05-03T18:52:47+00:00",
                    "Monitoring": {
                        "State": "disabled"
                    },
                    "Placement": {
                        "AvailabilityZone": "us-east-1a",
                        "GroupName": "",
                        "Tenancy": "default"
                    },
                    "PrivateDnsName": "ip-172-31-20-234.ec2.internal",
                    "PrivateIpAddress": "172.31.20.234",
                    "ProductCodes": [],
                    "PublicDnsName": "ec2-54-242-71-218.compute-1.amazonaws.com
",

                    "PublicIpAddress": "54.242.71.218",
                    "State": {
```

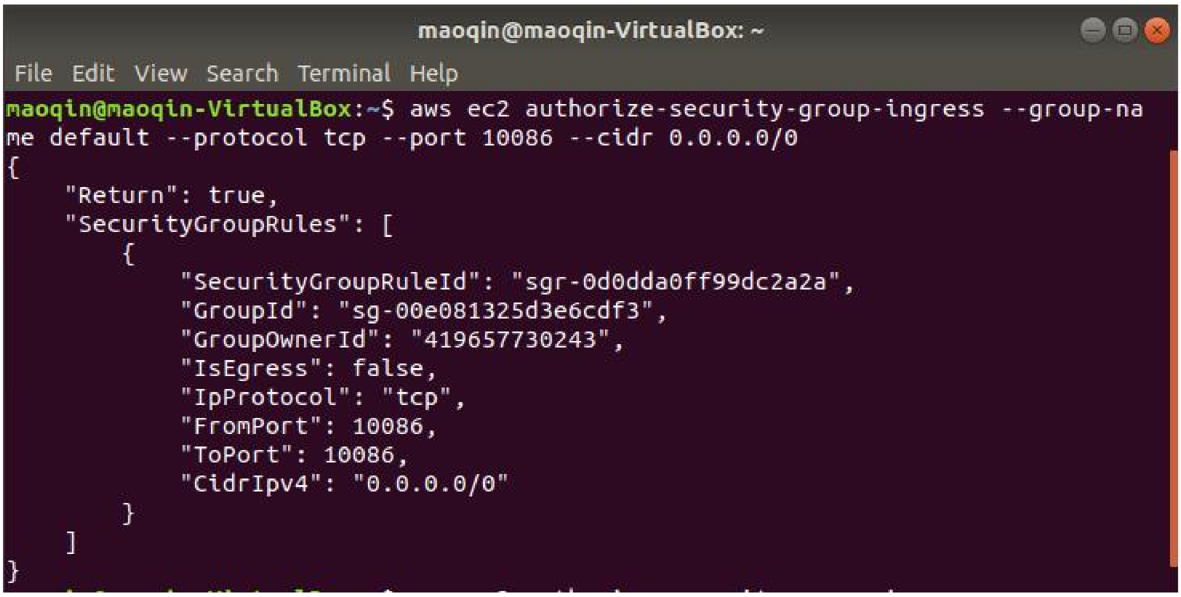
Step 6- Open Specific Ports on Our EC2 Instance

First of all, there are some ports to be opened in this lab:

SSH: **22**
Front-end Service: **6060**
Catalog Service: **10086**
Order Service: **10010-10012**

Type the following command on your local machine to open the ports:

$ aws ec2 authorize-security-group-ingress --group-name default --protocol tcp --port <number> --cidr 0.0.0.0/0


maoqin@maoqin-VirtualBox: ~
File Edit View Search Terminal Help

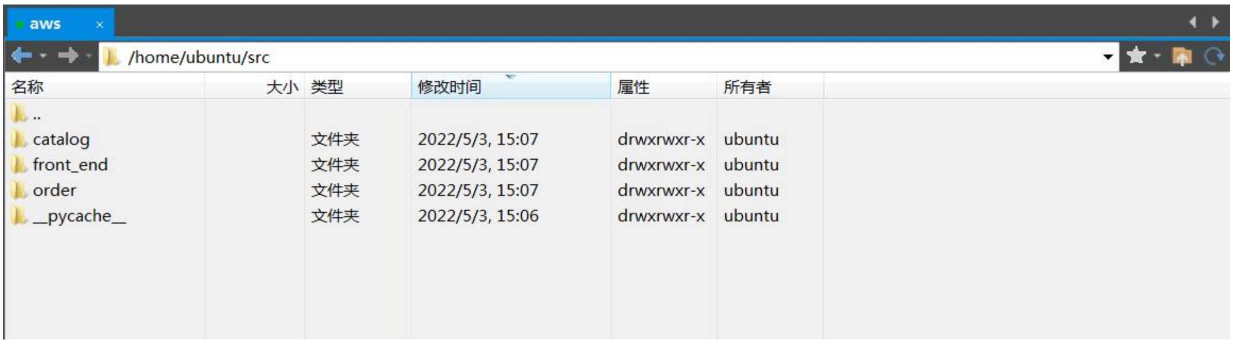
```
maoqin@maoqin-VirtualBox:~$ aws ec2 authorize-security-group-ingress --group-na
me default --protocol tcp --port 10086 --cidr 0.0.0.0/0
{
    "Return": true,
    "SecurityGroupRules": [
        {
            "SecurityGroupRuleId": "sgr-0d0dda0ff99dc2a2a",
            "GroupId": "sg-00e081325d3e6cdf3",
            "GroupOwnerId": "419657730243",
            "IsEgress": false,
            "IpProtocol": "tcp",
            "FromPort": 10086,
            "ToPort": 10086,
            "CidrIpv4": "0.0.0.0/0"
        }
    ]
}
```

Step 7- Upload Our Source Code on EC2 Instance

In general, you can type command on your terminal to access our EC2 instance via SSH. Here we will use the PEM file we download previously for getting the permission. And then upload your source code on EC2 instance by Git.

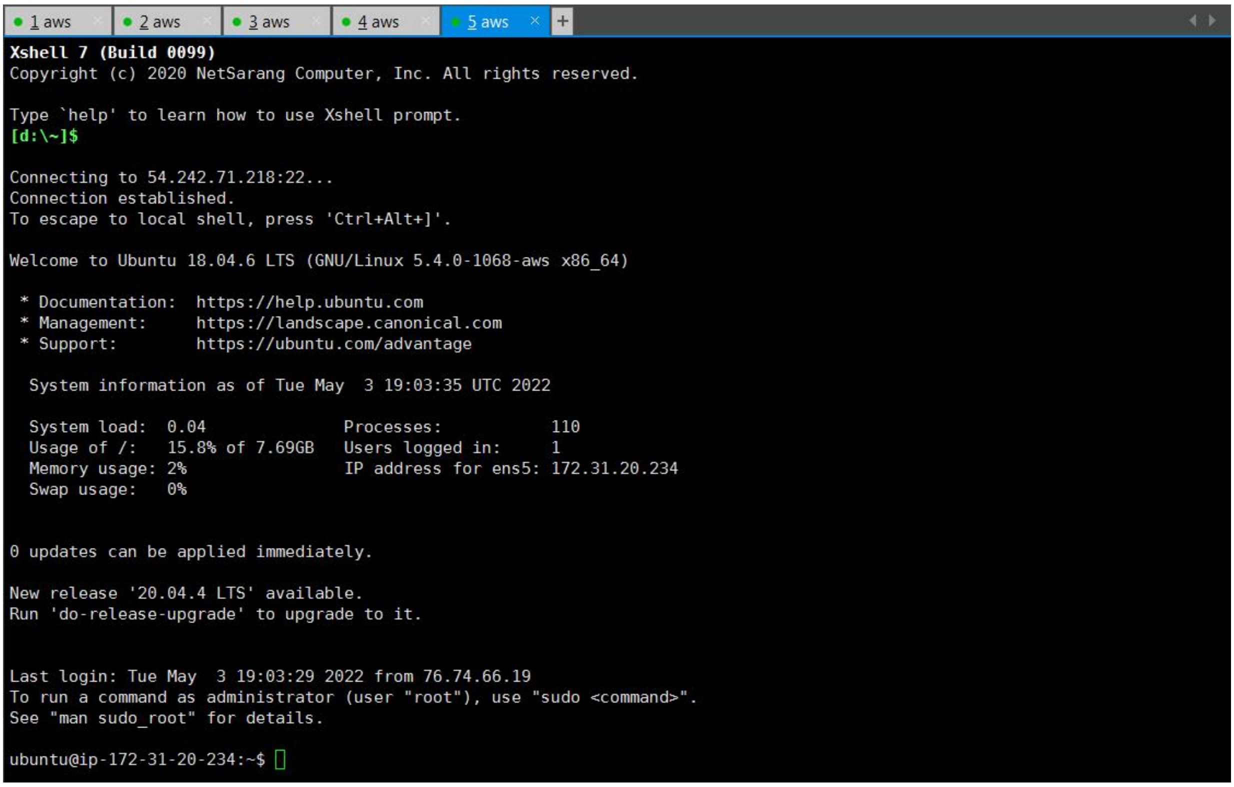
Here we adopt the tools **XShell** and **Xftp7** to help us access our EC2 instance and upload source code via SSH. We are using the PEM file too.

Xftp7 View:



名称	大小	类型	修改时间	属性	所有者
..					
catalog		文件夹	2022/5/3, 15:07	drwxrwxr-x	ubuntu
front_end		文件夹	2022/5/3, 15:07	drwxrwxr-x	ubuntu
order		文件夹	2022/5/3, 15:07	drwxrwxr-x	ubuntu
__pycache__		文件夹	2022/5/3, 15:06	drwxrwxr-x	ubuntu

XShell View:



```
Xshell 7 (Build 0099)
Copyright (c) 2020 NetSarang Computer, Inc. All rights reserved.

Type `help' to learn how to use Xshell prompt.
[d:\~]$

Connecting to 54.242.71.218:22...
Connection established.
To escape to local shell, press 'Ctrl+Alt+]'.

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 Tue May  3 19:03:35 UTC 2022

  System load:  0.04               Processes:           110
  Usage of /:   15.8% of 7.69GB    Users logged in:     1
  Memory usage: 2%                 IP address for ens5: 172.31.20.234
  Swap usage:   0%


0 updates can be applied immediately.

New release '20.04.4 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Tue May  3 19:03:29 2022 from 76.74.66.19
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-20-234:~$
```

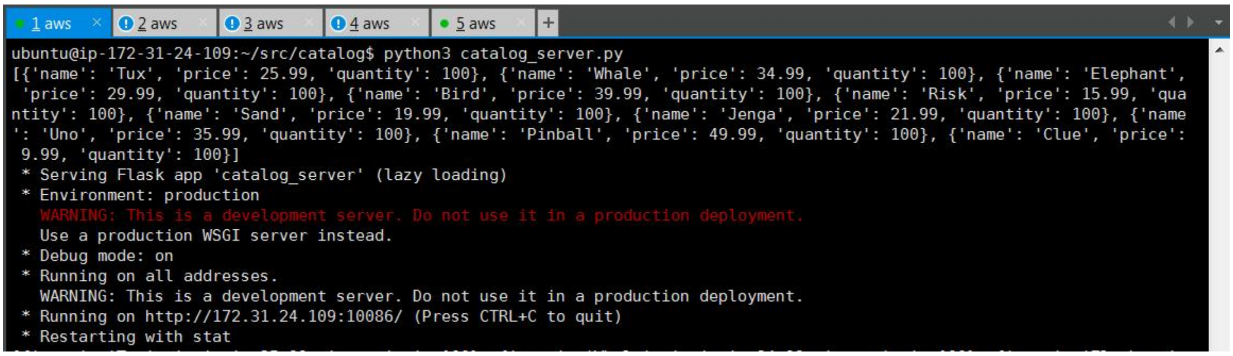
Step 8- Start Our Application

Notice that since we use several modules in our design, we recommend you run following commands on our EC2 instance at first:

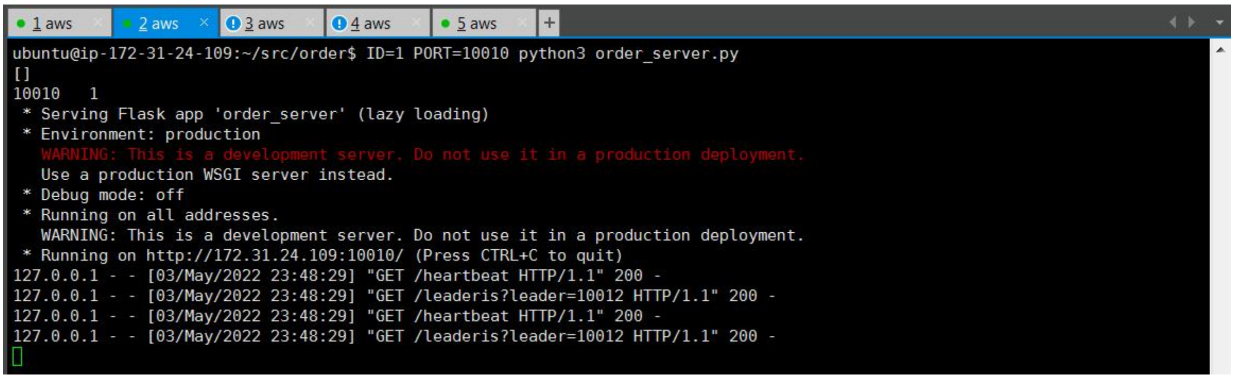
Update System: **$ sudo apt-get update**
Install pip3: **$ sudo apt-get -y install python3-pip**
Install Flask: **$ pip3 install flask**

Now in order to start our application, simply open 5 terminals on our AWS EC2 instance, and run micro services in each terminal in specific order. After cd into different directories, type the following commands.

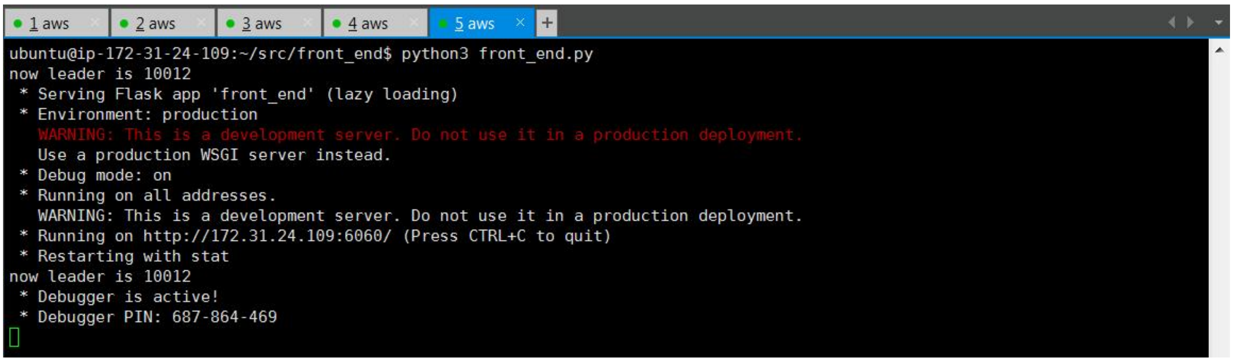
Terminal 1: **$ python3 catalog_server.py**
Terminal 2: **$ ID=1 PORT=10010 python3 order_server.py**
Terminal 3: **$ ID=2 PORT=10011 python3 order_server.py**
Terminal 4: **$ ID=3 PORT=10012 python3 order_server.py**
Terminal 5: **$ python3 front_end.py**



```
ubuntu@ip-172-31-24-109:~/src/catalog$ python3 catalog_server.py
[{'name': 'Tux', 'price': 25.99, 'quantity': 100}, {'name': 'Whale', 'price': 34.99, 'quantity': 100}, {'name': 'Elephant',
 'price': 29.99, 'quantity': 100}, {'name': 'Bird', 'price': 39.99, 'quantity': 100}, {'name': 'Risk', 'price': 15.99, 'qua
ntity': 100}, {'name': 'Sand', 'price': 19.99, 'quantity': 100}, {'name': 'Jenga', 'price': 21.99, 'quantity': 100}, {'name
': 'Uno', 'price': 35.99, 'quantity': 100}, {'name': 'Pinball', 'price': 49.99, 'quantity': 100}, {'name': 'Clue', 'price':
 9.99, 'quantity': 100}]
 * Serving Flask app 'catalog_server' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.31.24.109:10086/ (Press CTRL+C to quit)
 * Restarting with stat
```



```
ubuntu@ip-172-31-24-109:~/src/order$ ID=1 PORT=10010 python3 order_server.py
[]
10010   1
 * Serving Flask app 'order_server' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.31.24.109:10010/ (Press CTRL+C to quit)
127.0.0.1 - - [03/May/2022 23:48:29] "GET /heartbeat HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2022 23:48:29] "GET /leaderis?leader=10012 HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2022 23:48:29] "GET /heartbeat HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2022 23:48:29] "GET /leaderis?leader=10012 HTTP/1.1" 200 -
```



```
ubuntu@ip-172-31-24-109:~/src/front_end$ python3 front_end.py
now leader is 10012
 * Serving Flask app 'front_end' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.31.24.109:6060/ (Press CTRL+C to quit)
 * Restarting with stat
now leader is 10012
 * Debugger is active!
 * Debugger PIN: 687-864-469
```

Testing & Evaluation

In terms of screenshots of our functional tests & load tests, please check out the **"output" file** for details. In this document, we mainly focus on analyzing the results we have.

1. Load Test Results

Our testing codes can automatically sends **1000** Query, Buy or queryOrder requests. Python unittest can help measure the total latency seen by clients in this case. Hence, in terms of average latency for each request, we should divide the total time by **1000**. For different type of requests, we repeatedly run 5 clients at the same time, and measure the total latency seen by each client.

Table1: Load test results

Request Type	*Average Latency / s*
Query	*0.07258*
Buy	*0.08021*
queryOrder	*0.06128*

2. Caching Test Results

In order to estimate how much benefits does caching. We are firstly measuring the latency seen by each client for different type requests with caching turned on. Change the probability p of a follow up buy request from 0 to 80%, with an increment of 20%, and record the result for each p setting. And then do the same experiments but with caching turned off.

For each experiment with different p, we are testing **multiple times**, and record the **average latency**. The final results for different p with caching turned on are shown as follows:

Table2: Test result with caching turned on

Probability	*Average Query Latency / s*	*Average Buy Latency / s*
0	*0.04702*	*n/a*
0.2	*0.04563*	*0.05703*
0.4	*0.04303*	*0.05764*
0.6	*0.04193*	*0.05713*
0.8	*0.04589*	*0.05772*

Then we make a plot showing the values of p on the X-axis and response time/latency on the Y-axis.

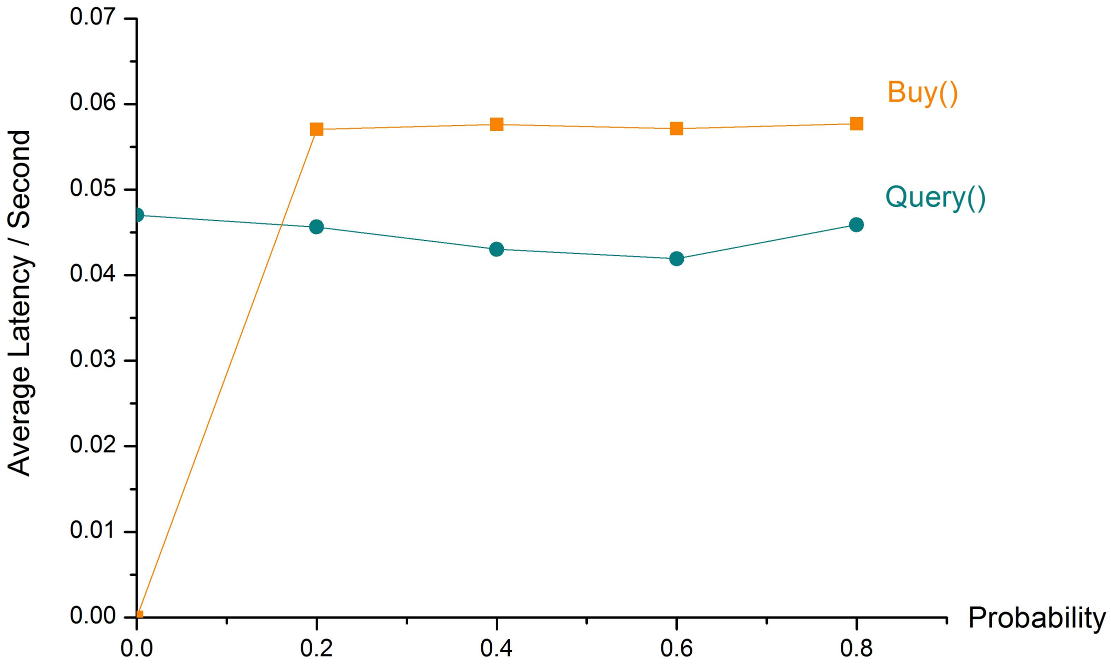


Figure1: Test result with caching turned on

And then just do the same experiments but with caching turned off. The final results for different p with caching turned off are shown as follows:

Table3: Test result with caching turned off

Probability	*Average Query Latency / s*	*Average Buy Latency / s*
0	*0.04754*	*n/a*
0.2	*0.04767*	*0.05917*
0.4	*0.04845*	*0.06005*
0.6	*0.04775*	*0.05976*
0.8	*0.04927*	*0.06240*

Then we make a plot showing the values of p on the X-axis and response time/latency on the Y-axis.

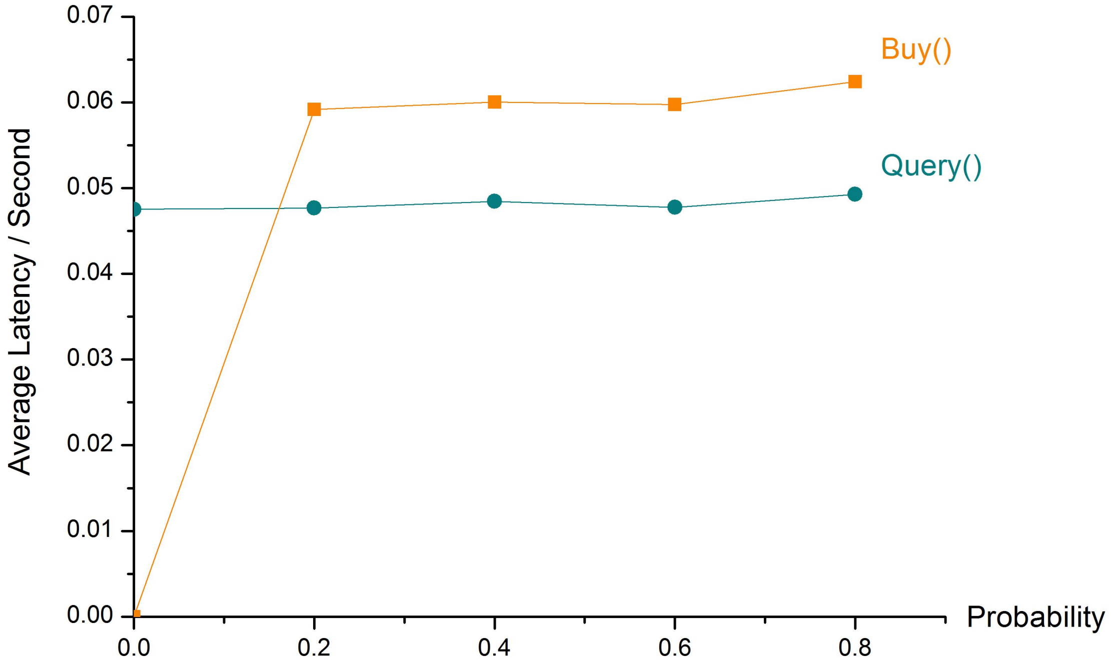


Figure2: Test result with caching turned off

Question 1

Estimate how much benefits does caching provide by comparing the results?

Solution:

As **table 2**, **figure 1**, **table 3** and **figure 2** shown above, the difference of average latency between different caching state (on/off) can be represented as follows:

Probability	*Query Latency Difference / s*	*Buy Latency Difference / s*
0	*0.00052*	*n/a*
0.2	*0.00204*	*0.00214*
0.4	*0.00542*	*0.00241*
0.6	*0.00582*	*0.00263*
0.8	*0.00338*	*0.00468*

Hence, we could say both the average query latency and the average buy latency with caching on is actually smaller than that with caching off.

Furthermore, the average benefits of caching for different types of requests are:

$$average\ benefits\ for\ query\ =\ 0.00344\ seconds\ better$$

$$average\ benefits\ for\ buy\ =\ 0.00297\ seconds\ better$$

3. Fault Tolerance Test Results

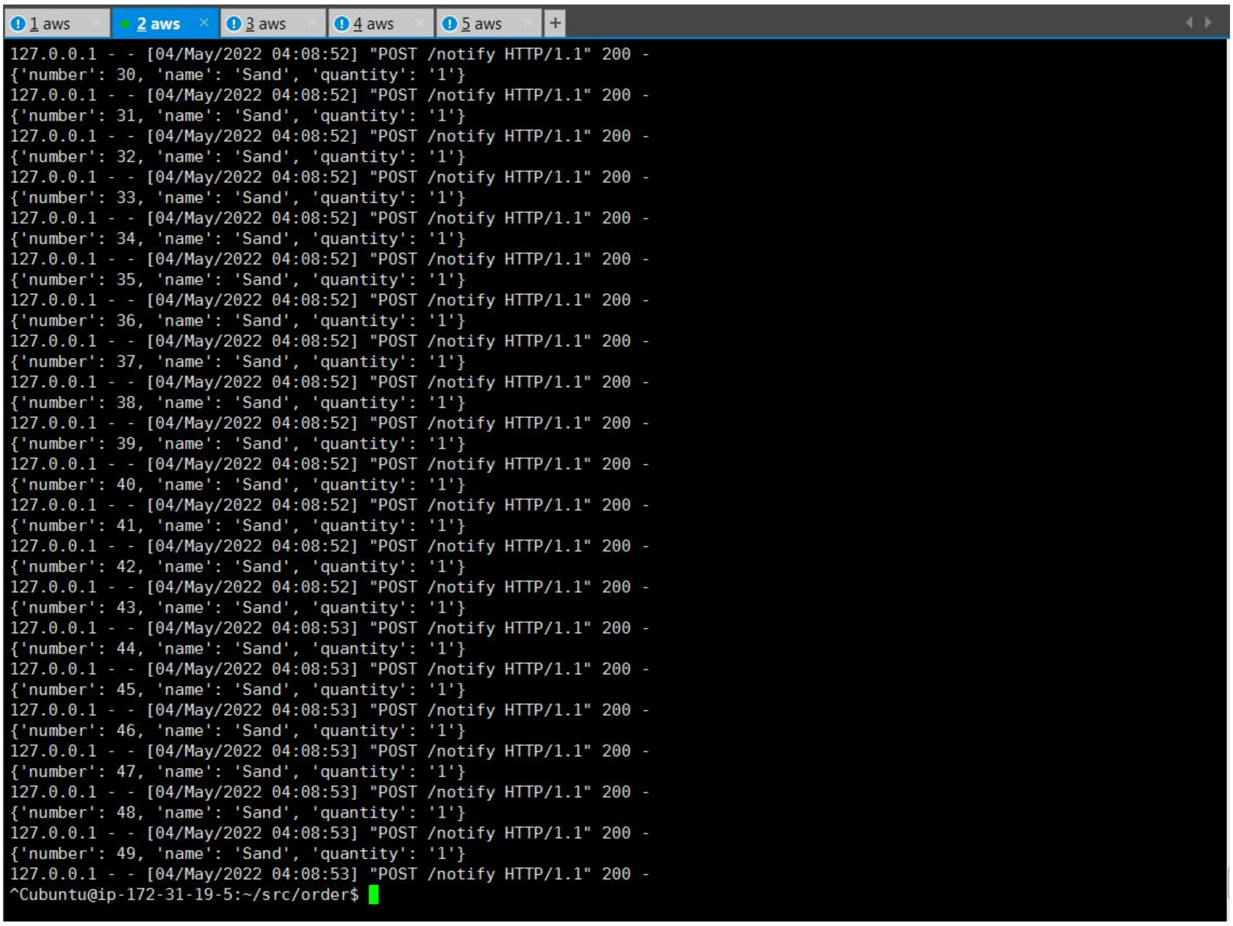
Finally, we are simulating crash failures by killing a random order service replica while the clients is running, and then bring it back online after some time.

Specifically, our **test case** follows the steps as described below:

Client Terminal: we are sending **1000** buy requests using the code in load test.

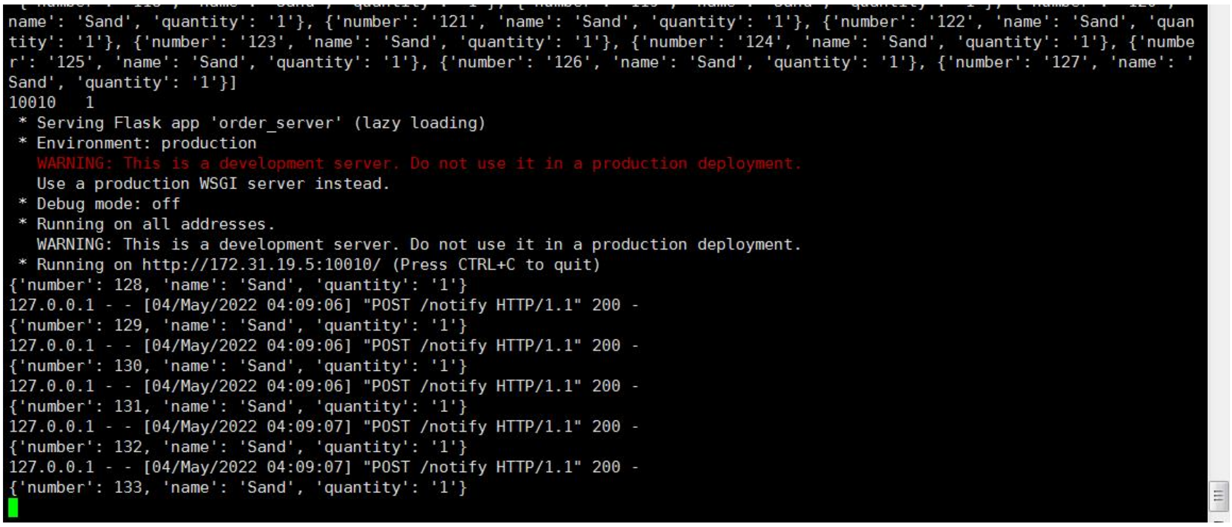
$ FRONT=<IP address> python3 -m unittest -v test_load.TestLoadPerformance.test_load_buy

Crash the follower with id = 1: terminate the node with **port = 10010** & **id=1**



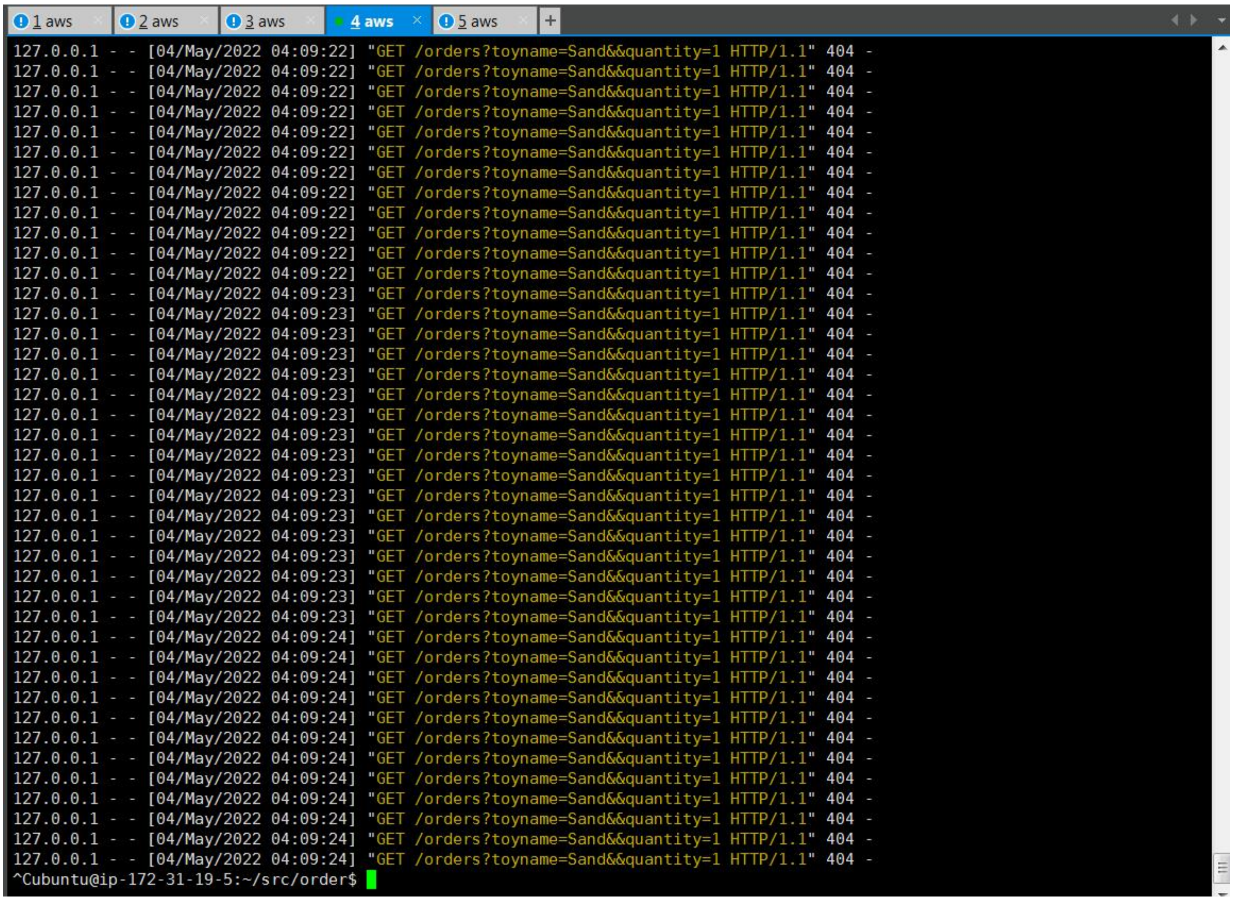
```
127.0.0.1 - - [04/May/2022 04:08:52] "POST /notify HTTP/1.1" 200 -
{'number': 30, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:52] "POST /notify HTTP/1.1" 200 -
{'number': 31, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:52] "POST /notify HTTP/1.1" 200 -
{'number': 32, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:52] "POST /notify HTTP/1.1" 200 -
{'number': 33, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:52] "POST /notify HTTP/1.1" 200 -
{'number': 34, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:52] "POST /notify HTTP/1.1" 200 -
{'number': 35, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:52] "POST /notify HTTP/1.1" 200 -
{'number': 36, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:52] "POST /notify HTTP/1.1" 200 -
{'number': 37, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:52] "POST /notify HTTP/1.1" 200 -
{'number': 38, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:52] "POST /notify HTTP/1.1" 200 -
{'number': 39, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:52] "POST /notify HTTP/1.1" 200 -
{'number': 40, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:52] "POST /notify HTTP/1.1" 200 -
{'number': 41, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:52] "POST /notify HTTP/1.1" 200 -
{'number': 42, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:52] "POST /notify HTTP/1.1" 200 -
{'number': 43, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:53] "POST /notify HTTP/1.1" 200 -
{'number': 44, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:53] "POST /notify HTTP/1.1" 200 -
{'number': 45, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:53] "POST /notify HTTP/1.1" 200 -
{'number': 46, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:53] "POST /notify HTTP/1.1" 200 -
{'number': 47, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:53] "POST /notify HTTP/1.1" 200 -
{'number': 48, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:53] "POST /notify HTTP/1.1" 200 -
{'number': 49, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:08:53] "POST /notify HTTP/1.1" 200 -
^Cubuntu@ip-172-31-19-5:~/src/order$
```

Restart the follower with id = 1: restart the node with **port = 10010** & **id=1**



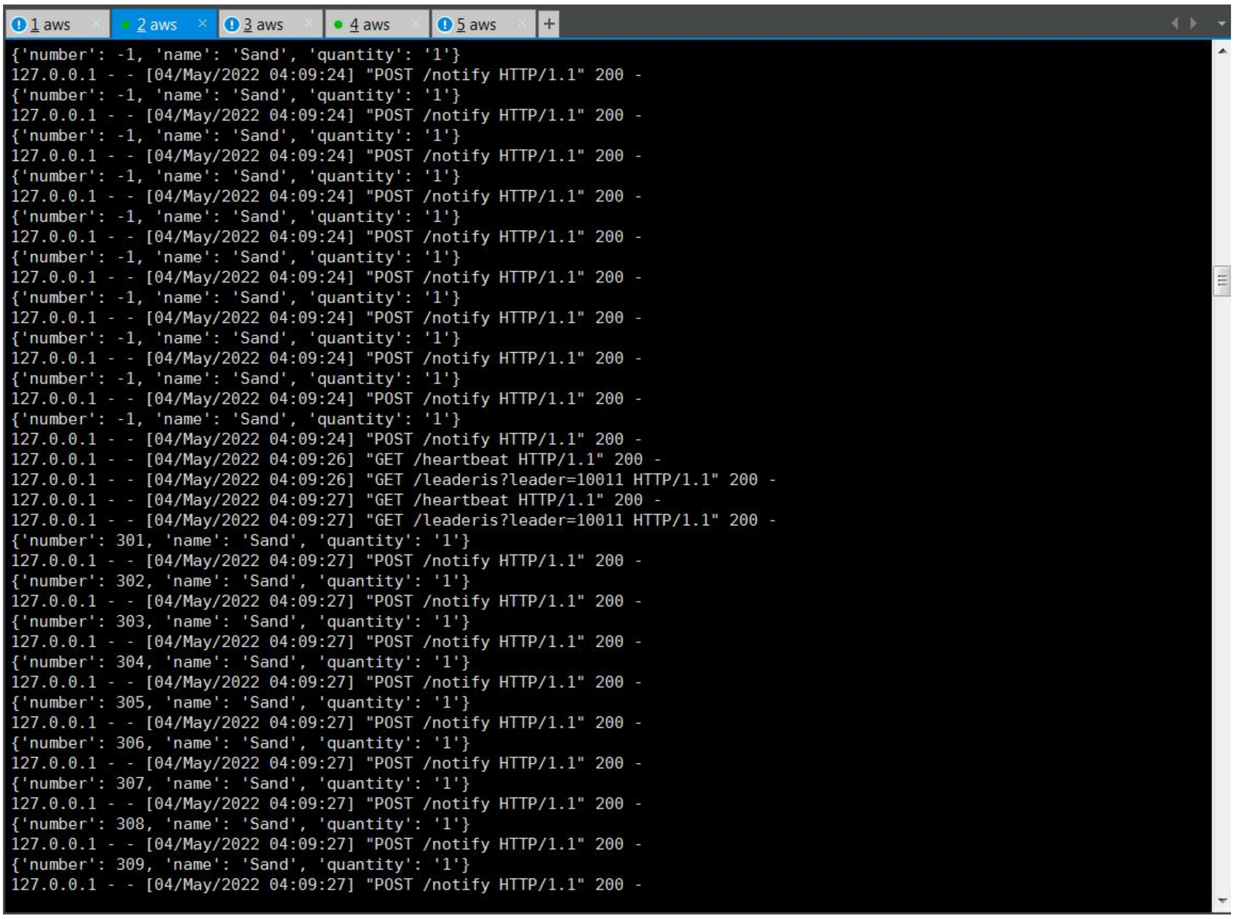
```
name': 'Sand', 'quantity': '1'}, {'number': '121', 'name': 'Sand', 'quantity': '1'}, {'number': '122', 'name': 'Sand', 'quan
tity': '1'}, {'number': '123', 'name': 'Sand', 'quantity': '1'}, {'number': '124', 'name': 'Sand', 'quantity': '1'}, {'numbe
r': '125', 'name': 'Sand', 'quantity': '1'}, {'number': '126', 'name': 'Sand', 'quantity': '1'}, {'number': '127', 'name': '
Sand', 'quantity': '1'}]
10010   1
 * Serving Flask app 'order_server' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.31.19.5:10010/ (Press CTRL+C to quit)
{'number': 128, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:06] "POST /notify HTTP/1.1" 200 -
{'number': 129, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:06] "POST /notify HTTP/1.1" 200 -
{'number': 130, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:06] "POST /notify HTTP/1.1" 200 -
{'number': 131, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:07] "POST /notify HTTP/1.1" 200 -
{'number': 132, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:07] "POST /notify HTTP/1.1" 200 -
{'number': 133, 'name': 'Sand', 'quantity': '1'}
```

Crash the leader with id = 3: terminate the node with **port = 10012** & **id=3**

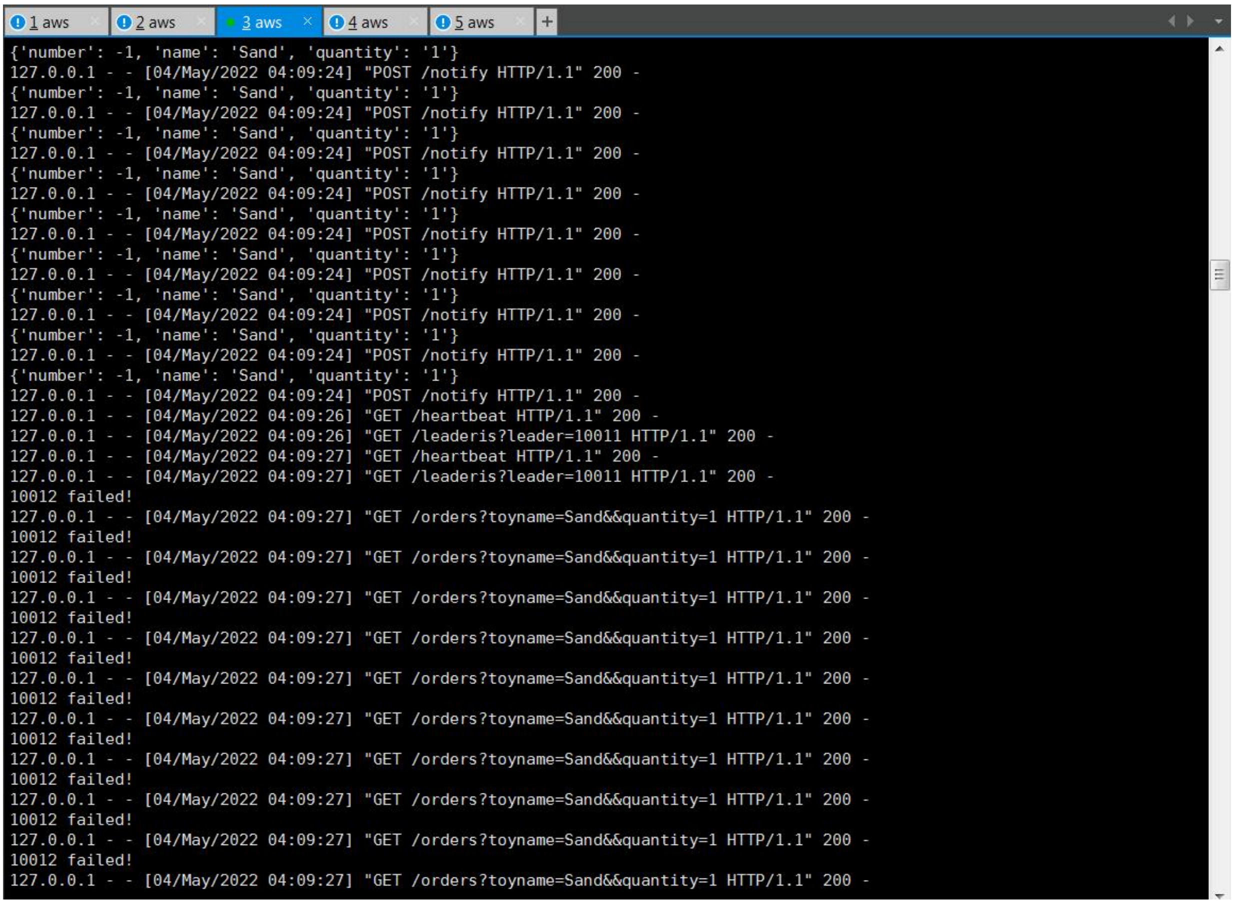


```
127.0.0.1 - - [04/May/2022 04:09:22] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:22] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:22] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:22] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:22] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:22] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:22] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:22] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:22] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:22] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:22] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:22] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:23] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:23] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:23] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:23] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:23] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:23] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:23] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:23] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:23] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:23] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:23] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:23] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:23] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:23] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:23] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:23] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:23] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:24] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:24] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:24] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:24] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:24] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:24] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:24] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:24] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:24] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:24] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:24] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
127.0.0.1 - - [04/May/2022 04:09:24] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 404 -
^Cubuntu@ip-172-31-19-5:~/src/order$
```

New leader notification: since leader is crashed, front end performed the **leader election**, and notify other nodes who is the new leader.

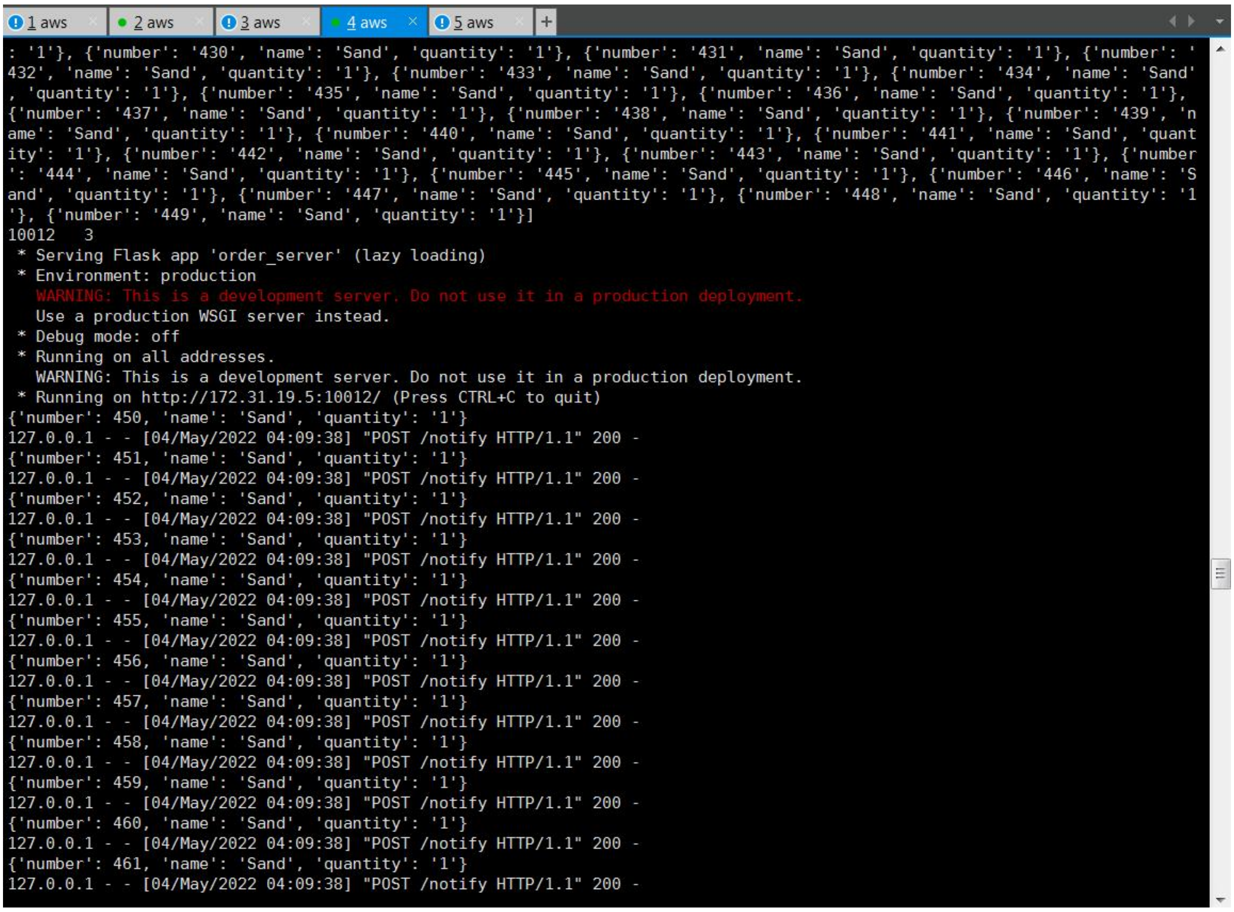


```
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
127.0.0.1 - - [04/May/2022 04:09:26] "GET /heartbeat HTTP/1.1" 200 -
127.0.0.1 - - [04/May/2022 04:09:26] "GET /leaderis?leader=10011 HTTP/1.1" 200 -
127.0.0.1 - - [04/May/2022 04:09:27] "GET /heartbeat HTTP/1.1" 200 -
127.0.0.1 - - [04/May/2022 04:09:27] "GET /leaderis?leader=10011 HTTP/1.1" 200 -
{'number': 301, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:27] "POST /notify HTTP/1.1" 200 -
{'number': 302, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:27] "POST /notify HTTP/1.1" 200 -
{'number': 303, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:27] "POST /notify HTTP/1.1" 200 -
{'number': 304, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:27] "POST /notify HTTP/1.1" 200 -
{'number': 305, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:27] "POST /notify HTTP/1.1" 200 -
{'number': 306, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:27] "POST /notify HTTP/1.1" 200 -
{'number': 307, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:27] "POST /notify HTTP/1.1" 200 -
{'number': 308, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:27] "POST /notify HTTP/1.1" 200 -
{'number': 309, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:27] "POST /notify HTTP/1.1" 200 -
```



```
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
{'number': -1, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:24] "POST /notify HTTP/1.1" 200 -
127.0.0.1 - - [04/May/2022 04:09:26] "GET /heartbeat HTTP/1.1" 200 -
127.0.0.1 - - [04/May/2022 04:09:26] "GET /leaderis?leader=10011 HTTP/1.1" 200 -
127.0.0.1 - - [04/May/2022 04:09:27] "GET /heartbeat HTTP/1.1" 200 -
127.0.0.1 - - [04/May/2022 04:09:27] "GET /leaderis?leader=10011 HTTP/1.1" 200 -
10012 failed!
127.0.0.1 - - [04/May/2022 04:09:27] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 200 -
10012 failed!
127.0.0.1 - - [04/May/2022 04:09:27] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 200 -
10012 failed!
127.0.0.1 - - [04/May/2022 04:09:27] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 200 -
10012 failed!
127.0.0.1 - - [04/May/2022 04:09:27] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 200 -
10012 failed!
127.0.0.1 - - [04/May/2022 04:09:27] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 200 -
10012 failed!
127.0.0.1 - - [04/May/2022 04:09:27] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 200 -
10012 failed!
127.0.0.1 - - [04/May/2022 04:09:27] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 200 -
10012 failed!
127.0.0.1 - - [04/May/2022 04:09:27] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 200 -
10012 failed!
127.0.0.1 - - [04/May/2022 04:09:27] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 200 -
10012 failed!
127.0.0.1 - - [04/May/2022 04:09:27] "GET /orders?toyname=Sand&&quantity=1 HTTP/1.1" 200 -
```

Restart the follower with id = 3: restart the node with **port = 10012** & **id=3**



```
: '1'}, {'number': '430', 'name': 'Sand', 'quantity': '1'}, {'number': '431', 'name': 'Sand', 'quantity': '1'}, {'number': '
432', 'name': 'Sand', 'quantity': '1'}, {'number': '433', 'name': 'Sand', 'quantity': '1'}, {'number': '434', 'name': 'Sand'
, 'quantity': '1'}, {'number': '435', 'name': 'Sand', 'quantity': '1'}, {'number': '436', 'name': 'Sand', 'quantity': '1'},
{'number': '437', 'name': 'Sand', 'quantity': '1'}, {'number': '438', 'name': 'Sand', 'quantity': '1'}, {'number': '439', 'n
ame': 'Sand', 'quantity': '1'}, {'number': '440', 'name': 'Sand', 'quantity': '1'}, {'number': '441', 'name': 'Sand', 'quant
ity': '1'}, {'number': '442', 'name': 'Sand', 'quantity': '1'}, {'number': '443', 'name': 'Sand', 'quantity': '1'}, {'number
': '444', 'name': 'Sand', 'quantity': '1'}, {'number': '445', 'name': 'Sand', 'quantity': '1'}, {'number': '446', 'name': 'S
and', 'quantity': '1'}, {'number': '447', 'name': 'Sand', 'quantity': '1'}, {'number': '448', 'name': 'Sand', 'quantity': '1
'}, {'number': '449', 'name': 'Sand', 'quantity': '1'}]
10012   3
 * Serving Flask app 'order_server' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.31.19.5:10012/ (Press CTRL+C to quit)
{'number': 450, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:38] "POST /notify HTTP/1.1" 200 -
{'number': 451, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:38] "POST /notify HTTP/1.1" 200 -
{'number': 452, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:38] "POST /notify HTTP/1.1" 200 -
{'number': 453, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:38] "POST /notify HTTP/1.1" 200 -
{'number': 454, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:38] "POST /notify HTTP/1.1" 200 -
{'number': 455, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:38] "POST /notify HTTP/1.1" 200 -
{'number': 456, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:38] "POST /notify HTTP/1.1" 200 -
{'number': 457, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:38] "POST /notify HTTP/1.1" 200 -
{'number': 458, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:38] "POST /notify HTTP/1.1" 200 -
{'number': 459, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:38] "POST /notify HTTP/1.1" 200 -
{'number': 460, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:38] "POST /notify HTTP/1.1" 200 -
{'number': 461, 'name': 'Sand', 'quantity': '1'}
127.0.0.1 - - [04/May/2022 04:09:38] "POST /notify HTTP/1.1" 200 -
```

Question 2

Can the clients notice the failures (either during order requests or the final order checking phase) or are they transparent to the clients? Do all the order service replicas end up with the same database file?

Solution:

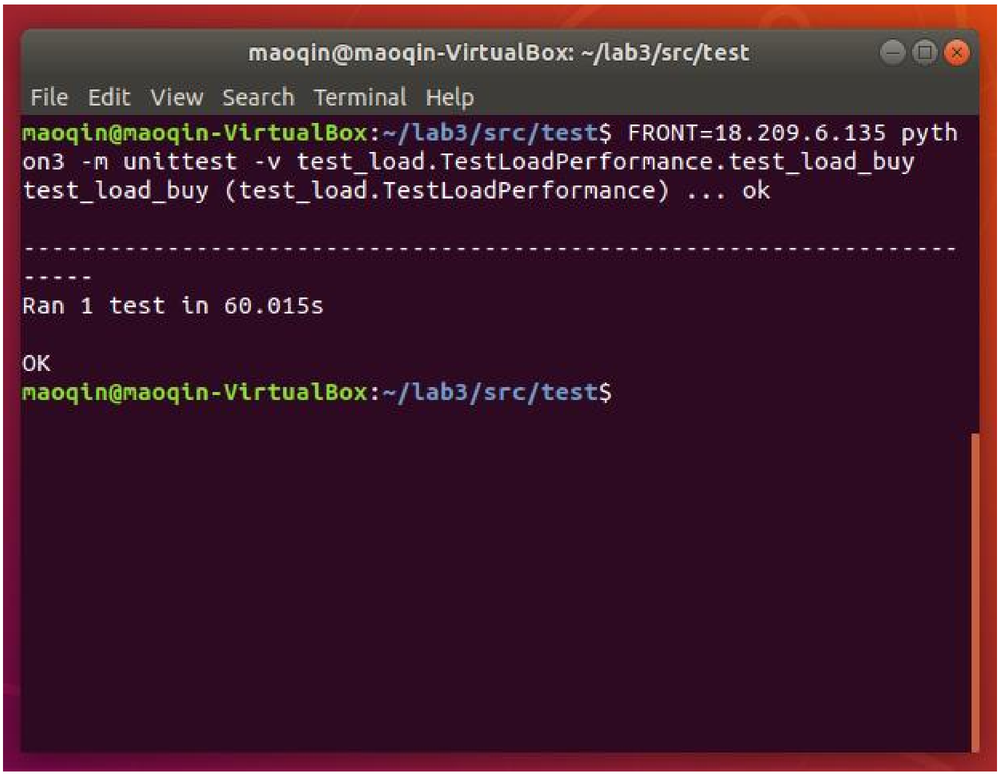
First of all, we are specifying two important factors:

Total latency: If the number of latency is too high, clients will notice the crash.

The number of lines in order log file: In our design, if an order is placed successfully, it will be recorded into order log with an increment unique ID. If a buy is unsuccessful (invalid toy name or out of stock), it will be recorded into order log with ID = -1. Hence, in our test case, if the number of lines in order log is much less than **1000**, it indicates that there are many TCP packets loss during the communication. The clients cannot even receive a successful / unsuccessful response. So we could say the clients can notice the crash in this scenario.

For the test case described previously, the results of those two factors are:

Total latency seen by clients:



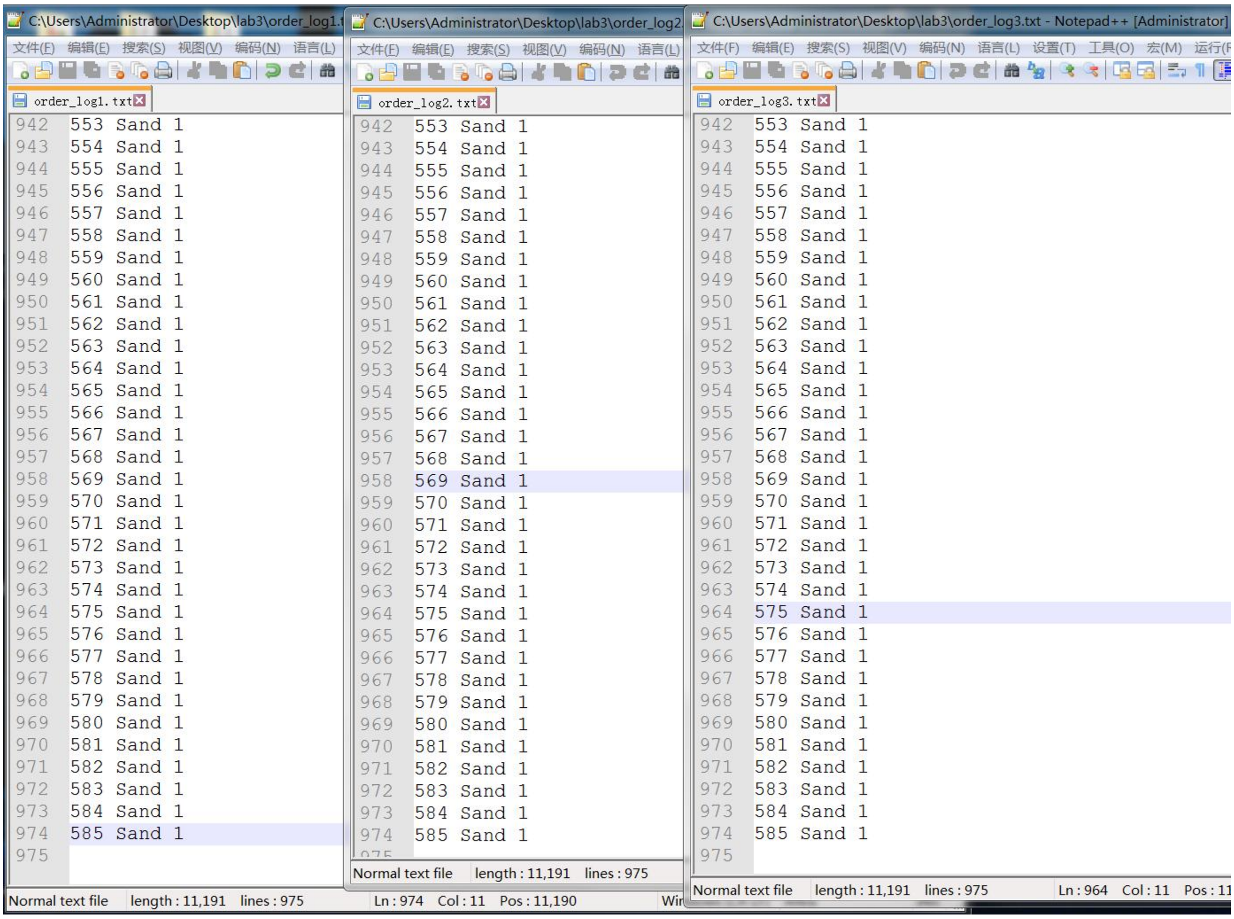
maoqin@maoqin-VirtualBox: ~/lab3/src/test
File Edit View Search Terminal Help

```
maoqin@maoqin-VirtualBox:~/lab3/src/test$ FRONT=18.209.6.135 pyth
on3 -m unittest -v test_load.TestLoadPerformance.test_load_buy
test_load_buy (test_load.TestLoadPerformance) ... ok

----------------------------------------------------------------
-----
Ran 1 test in 60.015s

OK
maoqin@maoqin-VirtualBox:~/lab3/src/test$
```

Order log at each order server:



C:\Users\Administrator\Desktop\lab3\order_log1.t
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L)
order_log1. txt
C:\Users\Administrator\Desktop\lab3\order_log2
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L)
order_log2. txt
C:\Users\Administrator\Desktop\lab3\order_log3.txt - Notepad++ [Administrator]
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(F
order_log3. txt

```
553 Sand 1
554 Sand 1
555 Sand 1
556 Sand 1
557 Sand 1
558 Sand 1
559 Sand 1
560 Sand 1
561 Sand 1
562 Sand 1
563 Sand 1
564 Sand 1
565 Sand 1
566 Sand 1
567 Sand 1
568 Sand 1
569 Sand 1
570 Sand 1
571 Sand 1
572 Sand 1
573 Sand 1
574 Sand 1
575 Sand 1
576 Sand 1
577 Sand 1
578 Sand 1
579 Sand 1
580 Sand 1
581 Sand 1
582 Sand 1
583 Sand 1
584 Sand 1
585 Sand 1
```

Normal text file length : 11,191 lines : 975 Ln : 974 Col : 11 Pos : 11,190 Wir

```
553 Sand 1
554 Sand 1
555 Sand 1
556 Sand 1
557 Sand 1
558 Sand 1
559 Sand 1
560 Sand 1
561 Sand 1
562 Sand 1
563 Sand 1
564 Sand 1
565 Sand 1
566 Sand 1
567 Sand 1
568 Sand 1
569 Sand 1
570 Sand 1
571 Sand 1
572 Sand 1
573 Sand 1
574 Sand 1
575 Sand 1
576 Sand 1
577 Sand 1
578 Sand 1
579 Sand 1
580 Sand 1
581 Sand 1
582 Sand 1
583 Sand 1
584 Sand 1
585 Sand 1
```

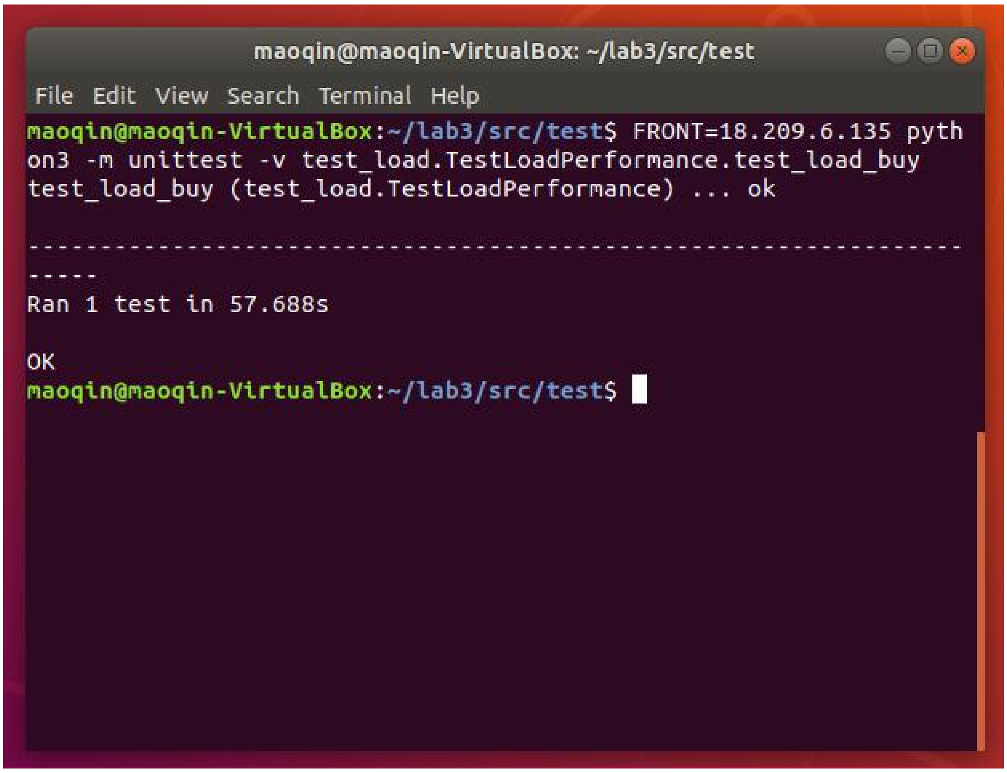
Normal text file length : 11,191 lines : 975

```
553 Sand 1
554 Sand 1
555 Sand 1
556 Sand 1
557 Sand 1
558 Sand 1
559 Sand 1
560 Sand 1
561 Sand 1
562 Sand 1
563 Sand 1
564 Sand 1
565 Sand 1
566 Sand 1
567 Sand 1
568 Sand 1
569 Sand 1
570 Sand 1
571 Sand 1
572 Sand 1
573 Sand 1
574 Sand 1
575 Sand 1
576 Sand 1
577 Sand 1
578 Sand 1
579 Sand 1
580 Sand 1
581 Sand 1
582 Sand 1
583 Sand 1
584 Sand 1
585 Sand 1
```

Normal text file length : 11,191 lines : 975 Ln : 964 Col : 11 Pos : 11

In order to evaluate in what degree the clients can notice the failure, we do the same experiment **without artificial crashes**.

Total latency seen by clients without artificial crashes:



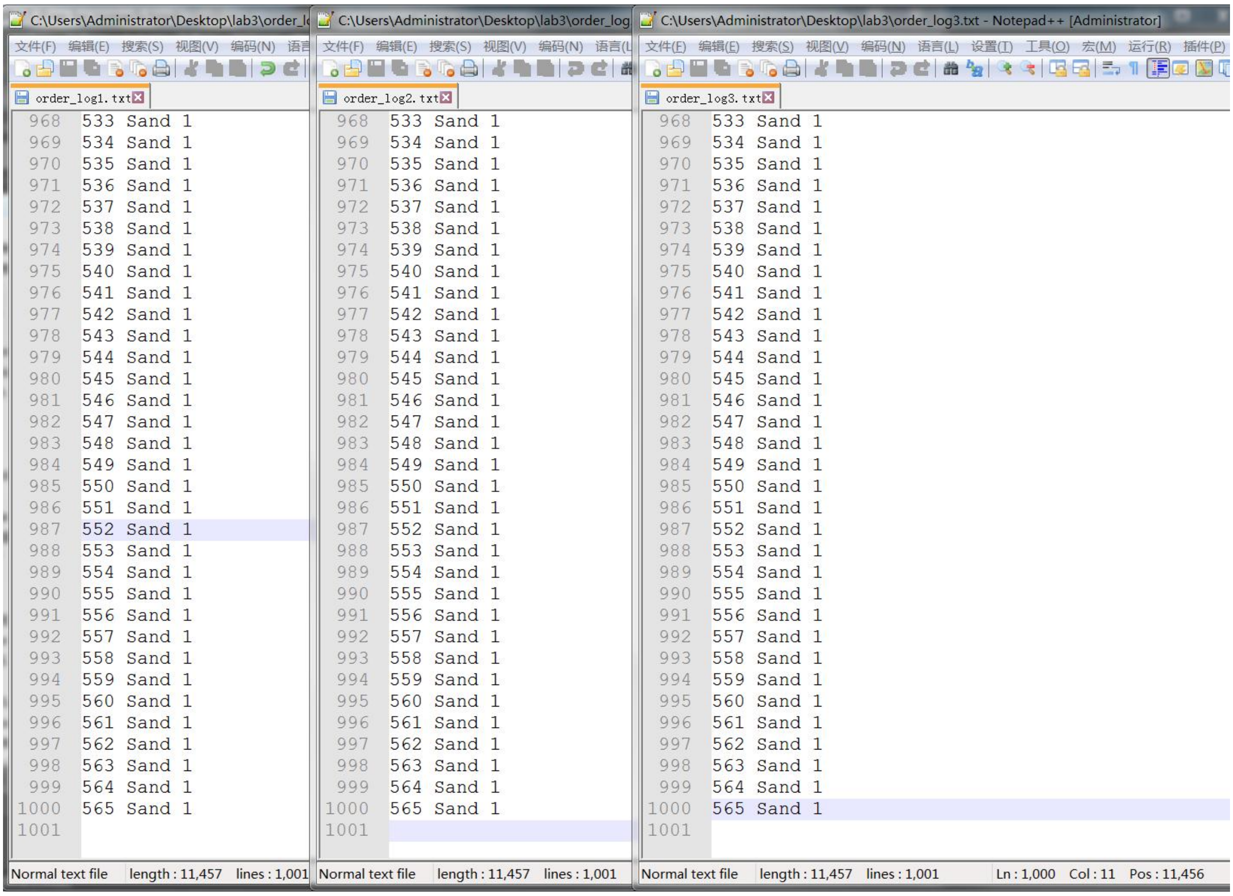
maoqin@maoqin-VirtualBox: ~/lab3/src/test
File Edit View Search Terminal Help

```
maoqin@maoqin-VirtualBox:~/lab3/src/test$ FRONT=18.209.6.135 pyth
on3 -m unittest -v test_load.TestLoadPerformance.test_load_buy
test_load_buy (test_load.TestLoadPerformance) ... ok

----------------------------------------------------------------------
-----
Ran 1 test in 57.688s

OK
maoqin@maoqin-VirtualBox:~/lab3/src/test$ 
```

Order log at each order server without artificial crashes:


文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P)

C:\Users\Administrator\Desktop\lab3\order_l

order_log1.txt

```
533 Sand 1
534 Sand 1
535 Sand 1
536 Sand 1
537 Sand 1
538 Sand 1
539 Sand 1
540 Sand 1
541 Sand 1
542 Sand 1
543 Sand 1
544 Sand 1
545 Sand 1
546 Sand 1
547 Sand 1
548 Sand 1
549 Sand 1
550 Sand 1
551 Sand 1
552 Sand 1
553 Sand 1
554 Sand 1
555 Sand 1
556 Sand 1
557 Sand 1
558 Sand 1
559 Sand 1
560 Sand 1
561 Sand 1
562 Sand 1
563 Sand 1
564 Sand 1
565 Sand 1
```

Normal text file length : 11,457 lines : 1,001

C:\Users\Administrator\Desktop\lab3\order_log

order_log2.txt

```
533 Sand 1
534 Sand 1
535 Sand 1
536 Sand 1
537 Sand 1
538 Sand 1
539 Sand 1
540 Sand 1
541 Sand 1
542 Sand 1
543 Sand 1
544 Sand 1
545 Sand 1
546 Sand 1
547 Sand 1
548 Sand 1
549 Sand 1
550 Sand 1
551 Sand 1
552 Sand 1
553 Sand 1
554 Sand 1
555 Sand 1
556 Sand 1
557 Sand 1
558 Sand 1
559 Sand 1
560 Sand 1
561 Sand 1
562 Sand 1
563 Sand 1
564 Sand 1
565 Sand 1
```

Normal text file length : 11,457 lines : 1,001

C:\Users\Administrator\Desktop\lab3\order_log3.txt - Notepad++ [Administrator]

order_log3.txt

```
533 Sand 1
534 Sand 1
535 Sand 1
536 Sand 1
537 Sand 1
538 Sand 1
539 Sand 1
540 Sand 1
541 Sand 1
542 Sand 1
543 Sand 1
544 Sand 1
545 Sand 1
546 Sand 1
547 Sand 1
548 Sand 1
549 Sand 1
550 Sand 1
551 Sand 1
552 Sand 1
553 Sand 1
554 Sand 1
555 Sand 1
556 Sand 1
557 Sand 1
558 Sand 1
559 Sand 1
560 Sand 1
561 Sand 1
562 Sand 1
563 Sand 1
564 Sand 1
565 Sand 1
```

Normal text file length : 11,457 lines : 1,001 Ln : 1,000 Col : 11 Pos : 11,456

	Total Latency / s	*Packet Loss*
Artificial Crash	*60.015*	*26*
Normal Case	*57.688*	*0*

As you can see, the difference of total latency between crashed case and normal case is really small. And in crashed case, the **packet loss rate** is:

$$packet\ loss\ rate\ =\ 26/1000\ =\ 0.026$$

On balance, we can give our **conclusion** as follows:

1) Clients almost cannot notice the failures (either during order requests or the final order checking phase). The crashes are transparent to the clients

2) As shown above, all the order servers end up with the same database file.