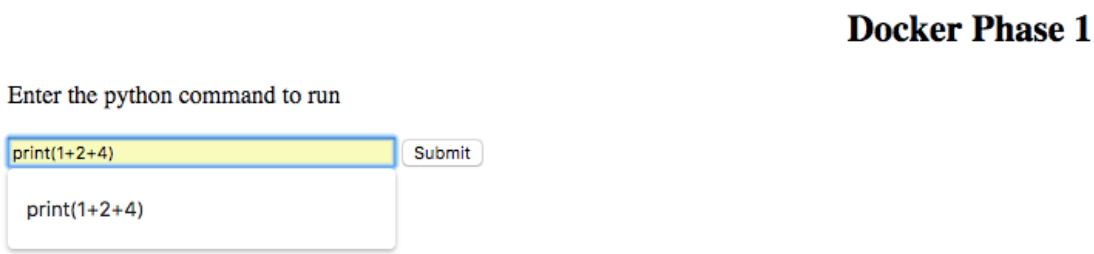
CSE 586
PROJECT 2: EMULATING PUB/SUB DISTRIBUTED SYSTEM USING DOCKER CONTAINERS
Report
Name: Rajiv Ranjan (rajivran) 50249099

Phase1: Download Docker, install it and learn how it works. Create a web interface where you can input ("TextBox") a small program in any language of your choice, and a command ("button") that will load the program in the Docker and execute it. The result of execution is display on the web output area.

Ans:
For Phase 1 what I am doing is that, I am taking input from a Text box see the UI below:
It takes in input any python script. So, enter a python script in the text box in the Ui and click on the SUBMIT button.

Give some python command as input:



Click on the submit button. The output is as below:

The output is below

7

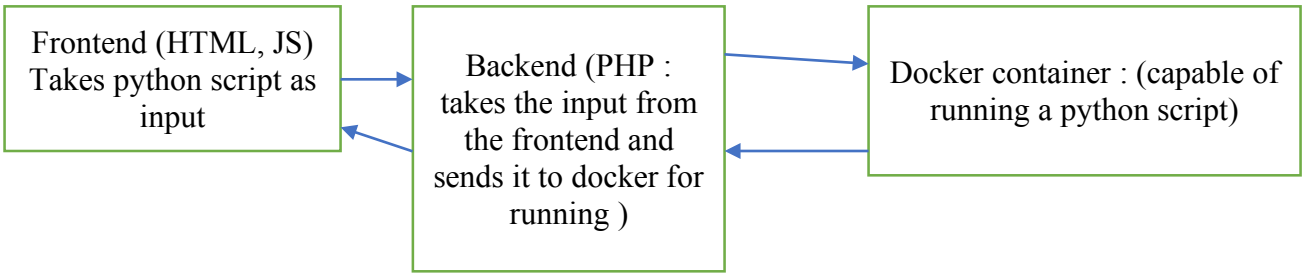
Discuss the choice of technologies and UI details in your report. Comment you code appropriately. Provide a detailed design representation like figure 1, customized to your design and technologies you used.

Soln: The technologies used are as follows:
1) HTML and JS for the front-end part.
2) PHP for the backend part
3) Docker for hosting and running the python script
4) Python for the choice of the language to run on docker
HTML is the frontend language used for web development so it was the obvious choice. Also Java script was used because the front end part needed some button on click and other events associated with it.
PHP was used because I had used PHP for project 1. So, familiarity with PHP was the main reason for choosing it. Also the main requirement of the project was to run any language on a docker container and then show it on aw web interface.

Python was chosen because I am specializing in ML some have done a lot of projects in Python. Hence it has become my favorite language.

So a docker image with the capability of running Python 3 was created first and then after a container was spawned from it which is the running entity, this runs the python script and shows the output.

Structure:



Phase 2: (30%) Now that you are familiar with Docker, implement a centralized version of the pub/sub application. You will need to implement the functions described in the Figure 1, the event generator and subscription generator for fully testing the application.

Soln: A centralized version of the Pub/Sub system has been implemented. Here is my design:

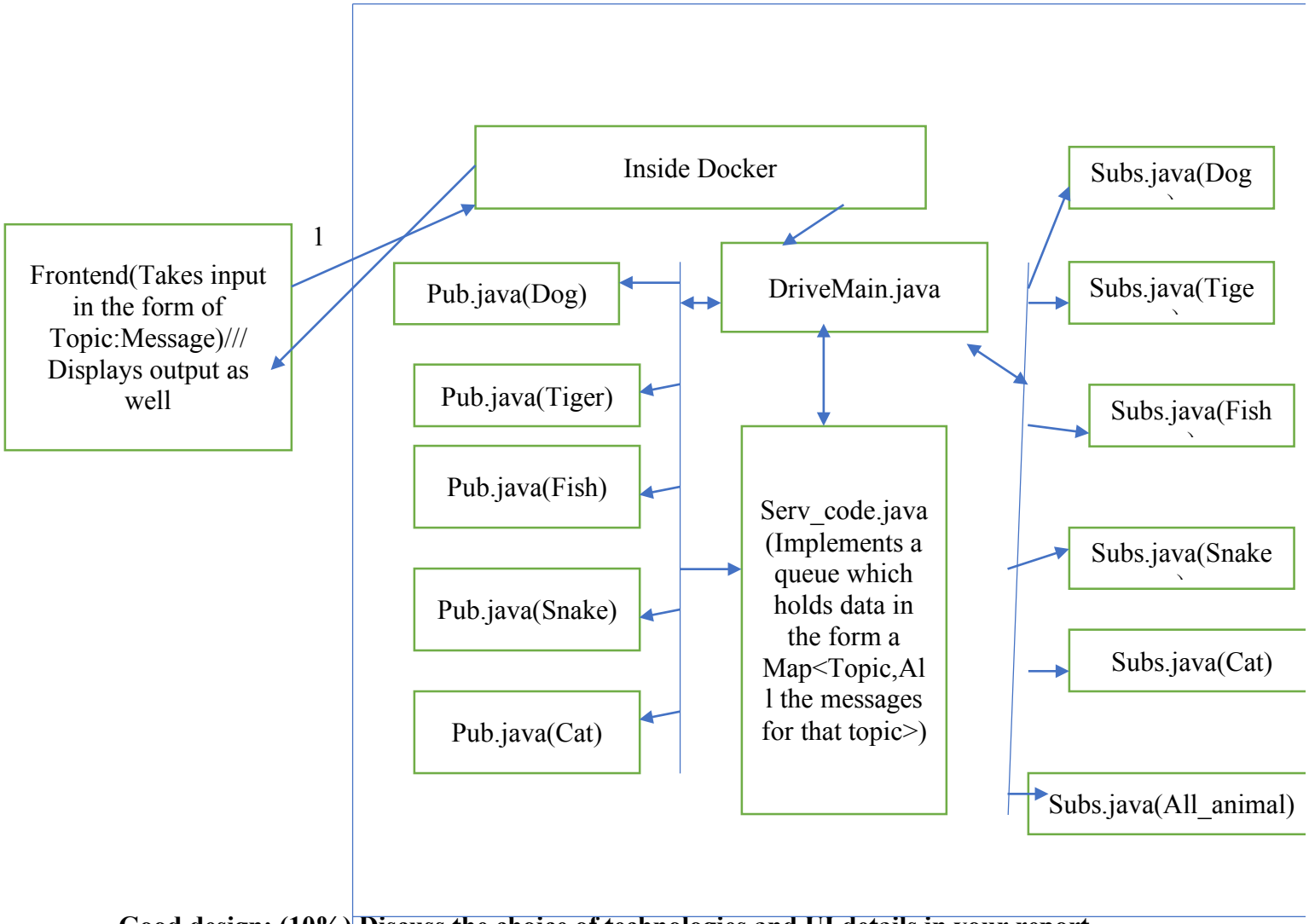
I am using Java for this phase 2 of the project. Since it is a centralized application. So the entire application runs on only one docker container. Application takes input in the form of topic and message.
Topic: message. Here is an example:

```
Dog:adasd
Cat:cat1
Tiger:adsad
Fish:Whale
Fish:shark
Snake:viper
Dog:husky
Snake:rattle_snake
Cat:cat2
Tiger:royal_bengal
Cat:cat3
Tiger:white_tiger
```

There are 5 topics on which you can send messages in the above-mentioned format. Topics are:
1)Cat 2) Dog 3) Fish 4) Snake 5) tiger

So the input takes the input message in the following format (Topic:message).
We have publishers for each of these topics. There is a publisher for each topic. And the publisher sends message pertaining to that topic only. After that ae create subscribers which can subscribe to each of these topics. Each subscriber can subscribe to one or more topics and there is a subscriber that has subscribed to all the topics. This has done to show the functionality in a better way. Now there is a broadcast/notify function which sends the messages to each to subscriber. There is a pub_sub serv code which caters to this. This uses a queue to store messages. The queue stores messages in the form of a Map (key is topic and value are the several messages published to each subscriber). Once the message is broadcasted or is notified then the queue is emptied so that the same message is not sent to the same subscriber again.
Secondly, if the publisher sends a new message on some topic, then again, the message is sent to the particular subscriber. Only the new message is sent this time and along with the old message we are showing it again. Again a docker capable of running Java 11 has been spawned and used.

Structure:



Good design: (10%) Discuss the choice of technologies and UI details in your report. Comment you code appropriately. Provide a detailed design representation like figure 1, customized to your design and technologies you used.

Soln:
Technologies used:
1) HTML and JS for the front-end part.
2) PHP for the backend part
3) Docker for hosting and running the java files
4) Java for the choice of the language to run on docker
HTML is the frontend language used for web development so it was the obvious choice. Also Java script was used because the front end part needed some button on click and other events associated with it.
PHP was used because I had used PHP for project 1. So, familiarity with PHP was the main reason for choosing it. Also the main requirement of the project was to run any language on a docker container and then show it on aw web interface.

Java was chosen because I some experience of working on Java and I like the fact that it is hard coded language. Hence it has become my favorite language.

So, a docker image with the capability of running Java 11 was created first and then after a container was spawned from it which is the running entity, this runs the Java files and shows the output.

We have a driver class called the DriveMain.java
It facilitates the different objects required for running the entire pub/sub system. I have created 5 publisher objects (one for each topic Cat, Dog, Fish, Tiger, Snake). What is being done is done is that the UI main_new.html takes input in the format specified above and after that a file is created that stores this info called abc.txt. all the files are added to a docker image which is capable of running Java 11 along with this input file.

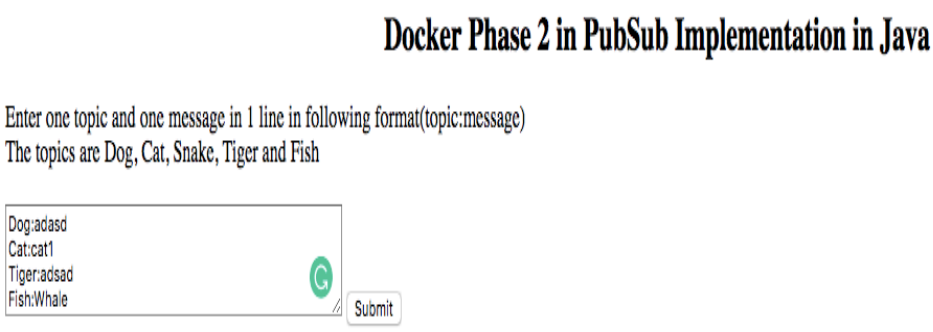
Nao a docker container is spawned with this image. This runs the input file on the docker and shows the output on the UI. Also after creating the publishers we create objects of specific topic called subscribers which subscribe to their corresponding topic which are published via the publishers. All of the published messages are sent to servcode.java which is essentially a queue which stores messages in the form of a Map this is in the form of Key value pair. (Key is the topic and value are the messages). After that we have, created an object of the serv_code object which publishes/notifies/broadcasts all the messages in it's queue to all the necessary recipients who have subscribed for that topic. After that the queue is emptied. This has been done so that the messages sent to the subscribers are not sent again.

After that just for testing purpose we create two messages (by hard-coding it in the code itself and then again repeat the process to see if it works and we see that the dog subscriber has all the previous messages as well as the new messages). This proves that our architecture not only works for the messages sent initially but also controls the messages sent afterwards.

Also we have created another subscriber called al-animal which is subscribed to all the topics and hence it receives messages for all the topics.

The inputs are taken randomly and also the events are generated randomly.

Input:



Output:

The output is below

```
Messages_of_Dog_Subscriber_are:
Message_Topic--->Dog:[adasd,husky,]

Messages_of_Cat_Subscriber_are:
Message_Topic--->Cat:[cat1,cat2,cat3,]

Messages_of_Tiger_Subscriber_are:
Message_Topic--->Tiger:[adsad,royal_bengal,white_tiger,]

Messages_of_Snake_Subscriber_are:
Message_Topic--->Snake:[viper,rattle_snake,]

Messages_of_Fish_Subscriber_are:
Message_Topic--->Fish:[Whale,shark,]

Messages_of_All_Animal_Subscriber_are:

Message_Topic--->Dog:[adasd,husky]
Message_Topic--->Cat:[cat1,cat2,cat3]
Message_Topic--->Snake:[viper,rattle_snake]
Message_Topic--->Tiger:[adsad,royal_bengal,white_tiger]
Message_Topic--->Fish:[Whale,shark]

Publishing_2_more_Dog_Messages_HardCoding...

Messages_of_Dog_Subscriber_now_are:
Message_Topic--->Dog:[adasd,husky]
Message_Topic--->Dog:[GOGO_DOGGY,SLEEPY_DOGGY]
```

Phase 3: (25%) Next step is to implement the distributed version of the pub/sub as described by rendezvous-based routing described in Figure 6.12 of Coulouris text.

Soln: Implement the pub/sub in a distributed fashion where the different pub sub systems are running on different Dockers.

For phase 3 also I am using Java as my language of implementation.
The implementation details are below:

The docker file used for this was taken from here. This docker file was used to build the image for this phase of the project.

https://www.cs.rit.edu/~ph/files/Dockerfile
There are two components of the project mainly:

1) Pub_sub server : running on a docker container
2) Pub_sub agents(multiple): each running on a separate docker container

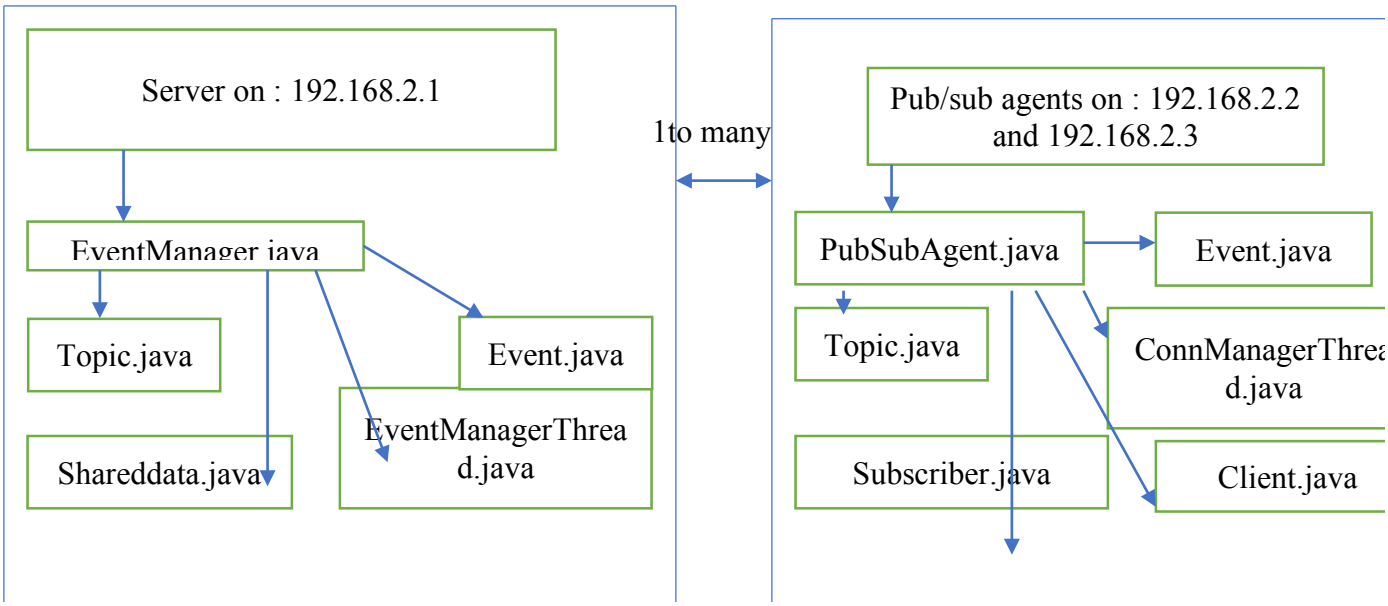
I have defined a subnet on docker that is capable of running and using upto IP addresses at time. The following command was used:
docker network create --driver=bridge --set=192.168.2.0/24 --gateway=192.168.2.10 new_subnet
Name of the subnet: is new_subnet

I am using 192.168.2.1 for running the server which facilitates the message to all the connected nodes which have subscribed for the message to be passed to them.
For the execution purposes, we run the pub/sub on 2 different servers on 192.168.2.2 and 192.168.2.3. but as we see we can run it on 23 different servers as we see now because our subnet has 24 different available slots.

The following files run the server(192.168.2.1):

1)EventManager.java
2)EventManagerThread.java
3)Topic.java
4)Event.java
5)Shareddata.java

1



The following files run as Pub/sub agents on 192.168.2.2 and 192.168.2.3
1) PubSubAgent.java
2)Subscriber.java
3) Topic.java
4) Event.java
5) Client.java
6) ConnManagerThread.java

The details of the execution are in readme file. Very important to follow the instructions as stated.

Execution:
Craete a subnet first:
docker network create --driver=bridge --set=192.168.2.0/24 --gateway=192.168.2.10 new_subnet

Inspect it using following command:
docker network inspect new_subnet

```
Rajivs-MacBook-Air:~ rajivranjan$ docker network inspect new_subnet
[
    {
        "Name": "new_subnet",
        "Id": "9bcad7d1a389ad6e50b229b52721f09c1e05e0dc0536f4f30438ba5c7f36332f",
        "Created": "2018-11-06T04:32:44.7336312Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "192.168.2.0/24",
                    "Gateway": "192.168.2.10"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

Go to the location where the Dockerfile is kept along with other java files. Build the image from the link provided in the document above:
Use the following commands :
docker build -t peiworld/csci652:latest .
docker run peiworld/csci652:latest
docker run --net new_subnet --ip 192.168.2.1 -it peiworld/csci652:latest

A new docker starts here with an id. Open another terminal and copy all the server related files to the docker container using it's id.

docker cp Event.java 8023ec9c84d5:/
docker cp EventManager.java 8023ec9c84d5:/
docker cp EventManagerThread.java 8023ec9c84d5:/
docker cp Topic.java 8023ec9c84d5:/
docker cp SharedData.java 8023ec9c84d5:/

Run the following commands to run the java file on the docker id:
javac EventManager.java
java EventManager

you will get something like below:

```
Rajivs-MacBook-Air:Applications rajivranjan$ cd /Applications/MAMP/htdocs/phase_3
Rajivs-MacBook-Air:phase_3 rajivranjan$ ls
Client.java                     EventManager.java               Subscriber.java
ConnManagerThread.java          EventManagerThread.class        Topic.class
ConnectionManager.java          EventManagerThread.java         Topic.java
Dockerfile                      PubSubAgent.java                main_new.html
Event.class                     Publisher.java                  next.php
Event.java                      SharedData.class                src
EventManager.class              SharedData.java
Rajivs-MacBook-Air:phase_3 rajivranjan$ /Applications/MAMP/htdocs/phase_3
-bash: /Applications/MAMP/htdocs/phase_3: is a directory
Rajivs-MacBook-Air:phase_3 rajivranjan$  docker build -t peiworld/csci652:latest .
Sending build context to Docker daemon    130kB
Step 1/3 : FROM ubuntu:16.04
 ---> 4a689991aa24
Step 2/3 : MAINTAINER Peizhao Hu, http://cs.rit.edu/~ph
 ---> Using cache
 ---> d681ac084d64
Step 3/3 : RUN apt-get update &&     apt-get upgrade -y &&     apt-get install -y  software-properties-common &&     add-
pt-repository ppa:webupd8team/java -y &&     apt-get update &&     echo oracle-java8-installer shared/accepted-oracle-lic
nse-v1-1 select true | /usr/bin/debconf-set-selections &&     apt-get install -y oracle-java8-installer &&     apt-get in
tall -y openssh-server net-tools iputils-ping ant nmap wget git vim build-essential &&     apt-get clean
 ---> Using cache
 ---> 2796eaaa0190
Successfully built 2796eaaa0190
Successfully tagged peiworld/csci652:latest
Rajivs-MacBook-Air:phase_3 rajivranjan$  docker run peiworld/csci652:latest
Rajivs-MacBook-Air:phase_3 rajivranjan$  docker run --net new_subnet --ip 192.168.2.1 -it peiworld/csci652:latest
root@6e54939c82d6:/# javac EventManager.java
root@6e54939c82d6:/# java EventManager
Waiting for request
```

This server is ready to accept connenctions now.

Start a new terminal now and then run the following commands to run an instance of pub/sub agent on 192.168.2.2

Copy the following files to the pub sub agent: bcea39ca8430 is the docker id (change it)
docker cp Client.java bcea39ca8430:/
docker cp ConnManagerThread.java bcea39ca8430:/
docker cp PubSubAgent.java bcea39ca8430:/
docker cp Subscriber.java bcea39ca8430:/
docker cp Event.java bcea39ca8430:/
docker cp Topic.java bcea39ca8430:/

Run the following commands on a new docker id terminal:

javac PubSubAgent.java
java PubSubAgent 192.168.2.1

Do the same on a new terminal just change the ip to 192.168.2.3 in
docker run --net new_subnet --ip 192.168.2.3 -it peiworld/csci652:latest

Copy the following files to the pub sub agent: bcea39ca8430 is the docker id (change it)
docker cp Client.java bcea39ca8430:/
docker cp ConnManagerThread.java bcea39ca8430:/
docker cp PubSubAgent.java bcea39ca8430:/
docker cp Subscriber.java bcea39ca8430:/
docker cp Event.java bcea39ca8430:/
docker cp Topic.java bcea39ca8430:/

Run the following commands on a new docker id terminal:

javac PubSubAgent.java
java PubSubAgent 192.168.2.1

1)Now the first step is to advertise a new topic say “UB”.
Do it from terminal 1.

```
Last login: Mon Nov 12 21:43:04 on ttys001
Rajivs-MacBook-Air:~ rajivranjan$ docker run --net new_subnet --ip 192.168.2.2 -
it peiworld/csci652:latest
root@a3e05378d304:/# javac PubSubAgent.java
root@a3e05378d304:/# java PubSubAgent 192.168.2.1
Enter 1 to Advertise
Enter 2 to Subscribe
Enter 3 to Publish
Enter 4 to Unsubscribe
1
Enter Topic
UB
0
Connected
Enter 1 to Advertise
Enter 2 to Subscribe
Enter 3 to Publish
Enter 4 to Unsubscribe
New Topic: UB
```

2)Then go to terminal 2 and subscribe to this topic

```
rajivranjan — root@9e42f4cd626e: / — docker run --net new_subnet --ip 192.1...
Last login: Mon Nov 12 21:44:49 on ttys003
Rajivs-MacBook-Air:~ rajivranjan$ docker run --net new_subnet --ip 192.168.2.2 -
it peiworld/csci652:latest
docker: Error response from daemon: Address already in use.
Rajivs-MacBook-Air:~ rajivranjan$ docker run --net new_subnet --ip 192.168.2.3 -
it peiworld/csci652:latest
root@9e42f4cd626e:/# javac PubSubAgent.java
root@9e42f4cd626e:/# java PubSubAgent 192.168.2.1
Enter 1 to Advertise
Enter 2 to Subscribe
Enter 3 to Publish
Enter 4 to Unsubscribe
New Topic: UB
2
Enter Topic
UB
Connected
Subscribed
Enter 1 to Advertise
Enter 2 to Subscribe
Enter 3 to Publish
Enter 4 to Unsubscribe
```

3)Come back to the terminal 1 and then publish a new msg on this topic.

```
Connected
Enter 1 to Advertise
Enter 2 to Subscribe
Enter 3 to Publish
Enter 4 to Unsubscribe
New Topic: UB
3
Enter Topic
UB
Topic found
Enter Title
DS
Enter Content
DOCKER
Connected
Published
Enter 1 to Advertise
Enter 2 to Subscribe
Enter 3 to Publish
Enter 4 to Unsubscribe
```

3) We see that the terminal 2 which was subscribed to the topic has got this message.

```
New Topic: UB
2
Enter Topic
UB
Connected
Subscribed
Enter 1 to Advertise
Enter 2 to Subscribe
Enter 3 to Publish
Enter 4 to Unsubscribe
Published: UB DS
```

Good design: (10%) Discuss the choice of technologies and UI details in your report. Comment you code appropriately. Provide a detailed design representation like figure 1, customized to your design and technologies you used.

Soln: This is a true distributed system in the sense that we have hosted the different pub sub system on different docker containers and also, we have used a different container to host the server. The server always runs and awaits connection from the different pub/sub agents. Also, we see that we can have multiple in fact upto 23 connections to server at a time.

I used a subnet in docker to define the connection name and scope. The choice was obvious because the whole project was supposed to be deployed via docker.

Also given the fact that one server and multiple pub/sub agents make it a one to many systems which is in fact a good design.

The language used was java because it provides good functionalities for multi-threaded programming.
MOST IMPORTANT: A peer to peer network has been designed where each node acts as publisher as well as subscriber.

Directory structure:
There are two main folders for the project 2.

1) Project2_rajivran_P1_P2.tar.gz
2) Project2_rajivran_P3_Docs.tar.gz

Project2_rajivran_P1_P2.tar.gz contains two folders:
1)phase_1 2) phase_2

phase_1 contains the following files:
1)main_new.html
2)next.php
3)abc.py
4)Dockerfile
5)execu_ta.sh
6)readme.txt

phase_2 contains the following files:
1)main_new.html
2)next.php
3)abc.txt
4)Dockerfile
5)DriveMain.java
6)DriveMain.class
7)info.class
8)info.java
9)PubCode.class
10)PubCode.java
11)serv_code.class
12)serv-code.java
13)subs.class
14)subs.java
15)test

Project2_rajivran_P3_Docs.tar.gz contains 1 folder and 1 file:
1)phase_3 2)Project2_rajivran_report.pdf

phase_3 has the following files:
1)Client.java
2)ConnectionManager.java

3)ConnManagerThread.java
4)Dockerfile
5)Event.java
6)EvebtManager.java
7)EventManagerThread.java
8)main_new.html
9)next.php
10) Publisher.java
11) SharedData.java
12)Subscriber.java
13)Topic.java
14)readme.txt

Please read the readme files very carefully to run the project especially for phase3.