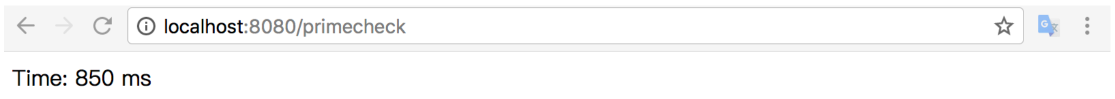
CSC8110: Cloud Computing

Name: Ruidong Wang

Student No. 170294526

Task 1:

In task 1, first, I used “Docker search nclcloudcomputing/javabenchmarkapp” to find the image “nclcloudcomputing/javabenchmarkapp”. And use “Docker run –p 8080:8080 nclcloudcomputing/javabenchmarkapp” to download and run this image. And then use my browser to http://localhost:8080/primecheck. I can saw the view like the following figure.



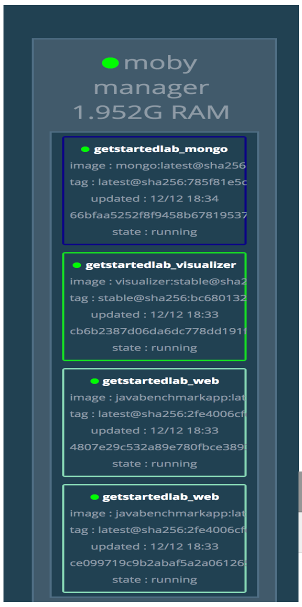
```
Time: 850 ms
```

Explanation:

In task 1, the first command is to find weather the Docker service have this image. The second command is to run the image, the Docker will download the image if there is no this image in local. “-p 8080:8080” means the set the post 8080 in the image to the post 8080 in my computer.

Task 2

Task 2 need I to design a multi service application in a Docker environment. In task 2 I written a docker-compose.yml file to build a multi service file. And then I used the “docker swarm init” to set my computer to become a swarm manager. After this I used “docker stack deploy -c docker-compose.yml getstartedlab” to start the multi service. And then use my browser to http://localhost:8080/primecheck and http://localhost:88/ . I can saw the view like the following figure.



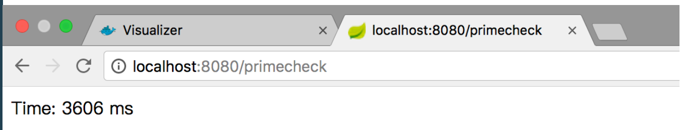
```
moby
manager
1.952G RAM

getstartedlab_mongo
image : mongo:latest@sha256
tag : latest@sha256:785f81e5c
updated : 12/12 18:34
66bfaa5252f8f9458b67819537
state : running

getstartedlab_visualizer
image : visualizer:stable@sha2
tag : stable@sha256:bc680132
updated : 12/12 18:33
cb6b2387d06da6dc778dd191
state : running

getstartedlab_web
image : javabenchmarkapp:lat
tag : latest@sha256:2fe4006cf
updated : 12/12 18:33
4807e29c532a89e780fbce389
state : running

getstartedlab_web
image : javabenchmarkapp:lat
tag : latest@sha256:2fe4006cf
updated : 12/12 18:33
ce099719c9b2abaf5a2a06126
state : running
```



localhost:8080/primecheck

Time: 3606 ms

Explanation:

In the docker-compose.yml file I design three parts of this multi service application. The first one is nclcloudcomputing/javabenchmarkapp, next is the explain of the configuration of this image:

replicas: 2: this means create two instances of this Docker image.

resources: limits: cpus: "0.3" memory: 500M: this is the limits of the instances.

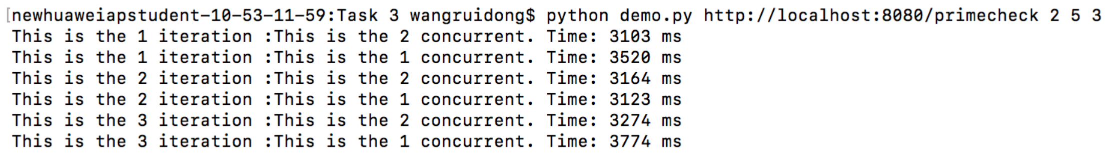
ports: - "8080:8080": this means setting the port of this image. set the post 8080 in the image to the post 8080 in my computer.

The second one is mongo, this instance is the service for mongoDB and I set the post 27017 in mongoDB image to the post 3306 in my computer.

The third one is visualizer. This image is set as a node manager and set the post 8080 in the image visualizer to the post 88 in my computer.

Task 3

In task 3 I need to design a small application to connect the Docker image instance to get the information from the website. I run the application and get the information like the following picture:



```
[newhuaweiapstudent-10-53-11-59:Task 3 wangruidong$ python demo.py http://localhost:8080/primecheck 2 5 3
 This is the 1 iteration :This is the 2 concurrent. Time: 3103 ms
 This is the 1 iteration :This is the 1 concurrent. Time: 3520 ms
 This is the 2 iteration :This is the 2 concurrent. Time: 3164 ms
 This is the 2 iteration :This is the 1 concurrent. Time: 3123 ms
 This is the 3 iteration :This is the 2 concurrent. Time: 3274 ms
 This is the 3 iteration :This is the 1 concurrent. Time: 3774 ms
```

Explanation:

In this test, the command line has 4 parameters. http://localhost:8080/primecheck is the URL which the application will get information form the website. 2 is the number of concurrent calls, that means the application will call for the URL 2 at the same time. 5 is the Interval between each set of concurrent calls, that means the application calls the URL have 5 seconds interval between each set. 3 means the application will call the URL 3 sets. So, in this test the application call for the URL 6 times in total. And the information gets form the website only the time which means the time server response the call. In order to know the concurrent and iteration time, I print more information about each call.

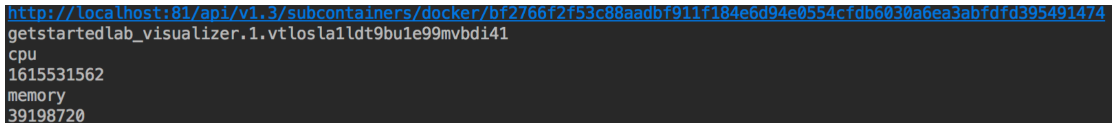
Discussion and Inference:

In task 3, I test some set of data. If I call the URL only set and one concurrent, the average time is nearly 3000ms. If I call the URL 10 set and 10 concurrent in each set, the average time is about 150000ms and the reaction time showed an uptrend, rising to about 170000ms basically stable. And I also test non-extreme data.

In my inference, the Docker image server the more concurrent access the server receives, the greater the pressure on the server, the longer the response time is, and the message is blocked. So, with the 10 with set 10 concurrent, the server need to spend 50 times time as long as the 1 set with 1 concurrent spent. This response time also depend on the machine performance.

Task 4

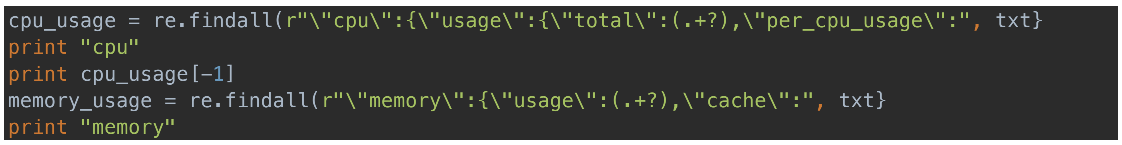
In task 4, I need to use the cAdvisor to get the information of the containers. Cadviosr is the monitoring tool for Google to monitor the resource information of a single node. The Cadvisor provides a straightforward, single-node multi-container resource monitoring capability. Through the cAdvisor we can see the information of the CPU usage and the memory usage. According to the web site provided by the coursework, I get the information of json about the cAdvisor home page. From this json data, I get the ID of the sub-containers, and add the ID string to the URL to get a new URL about the sub-containers. Getting into the sub-containers URL, I can get the information of CPU usage and the memory usage. And by this way, I got the usage information of the CPU and the memory. The following is a data I got from the URL. The first line is the URL of the sub-container. The second line is the name of the sub-container. The third line and fourth line are the usage of the CPU. The fifth and sixth line are the usage of the memory.



```
http://localhost:81/api/v1.3/subcontainers/docker/bf2766f2f53c88aadbf911f184e6d94e0554cfdb6030a6ea3abfdfd395491474
getstartedlab_visualizer.1.vtlosla1ldt9bu1e99mvbdi41
cpu
1615531562
memory
39198720
```

Explanation:

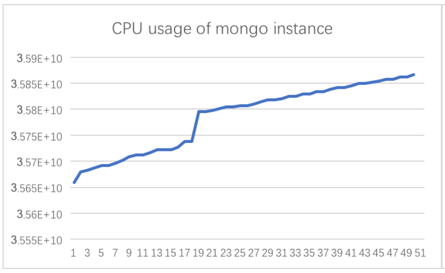
The hard part of task 4 is how to parse the json code. In my code, I use the python to realize this function. Python has a function of intercepting strings. So I'm going to classify the data that I need to have the uniform character of the data that I need. I my code `r"\"cpu\":{\"usage\":{\"total\":(.+?),\"per_cpu_usage\":"` is for acquiring the CPU usage. In this (.+?) represent the data which I want to get. After achieving this data, it is still not the final data which I want to get. The cpu_usage is the type of list in python which with many string of the CPU usage, and what I need is only one and the last one. So I get the cpu_usage[-1], this means I get the last one data from this list.

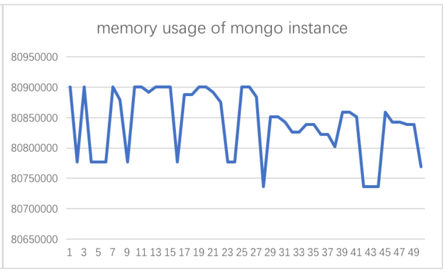


```
cpu_usage = re.findall(r"\"cpu\":{\"usage\":{\"total\":(.+?),\"per_cpu_usage\":", txt}
print "cpu"
print cpu_usage[-1]
memory_usage = re.findall(r"\"memory\":{\"usage\":(.+?),\"cache\":", txt}
print "memory"
```

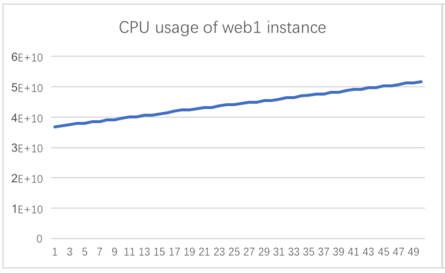
Discussion and Inference:

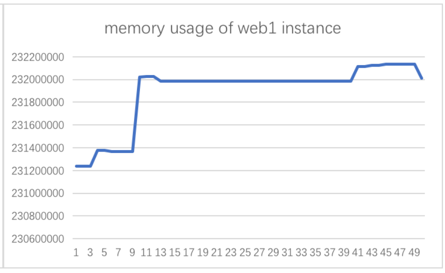
In task 4, I get the data from 5 containers and 50 set from each container. I used continuous 50 requests for access, whilst running the load generator.



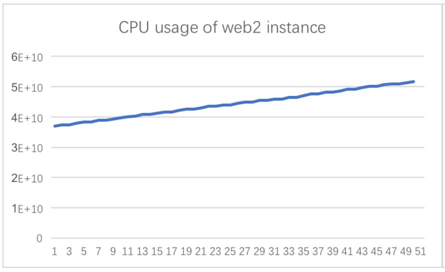


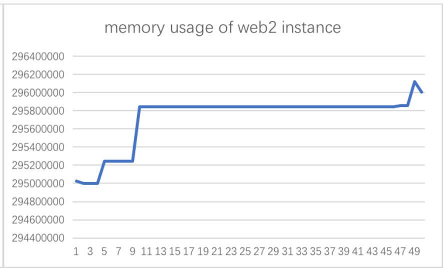
This is the usage of CPU and memory in mongo instance. From the CPU usage chart, we can know the usage of CPU is on the rise. But the memory usage is messy.



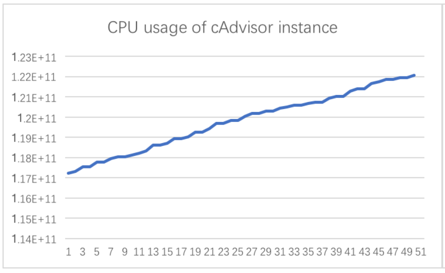


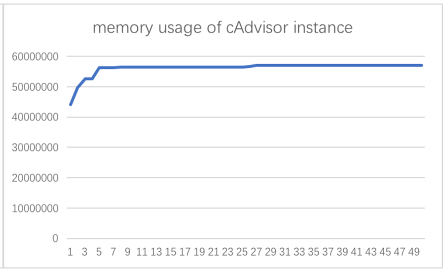
This is the usage of CPU and memory in web1 instance. It created by the Docker image nclcloudcomputing/javabenchmarkapp. From the CPU usage chart, we can know the usage of CPU is on the rise. Through the memory chart, the usage of memory used has started to rise gradually, and then tends to stabilize and fluctuate slightly in the end.



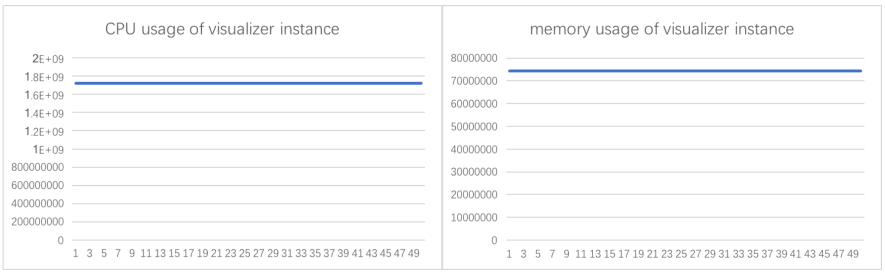


This is the usage of CPU and memory in web2 instance. It same as the web1 instance created by the Docker image nclcloudcomputing/javabenchmarkapp. So, the usage of the CPU and the memory are basically similar with web1.





This is the usage of CPU and memory in cAdvisor instance. From the CPU usage chart, we can know the usage of CPU is on the rise. Memory usage rose to a certain level tends to be stable.



This is the usage of CPU and memory in visualizar instance. From two charts, The CPU usage and the memory usage are consistently on a horizontal line. That means the visualizer instance is always in full load operation.

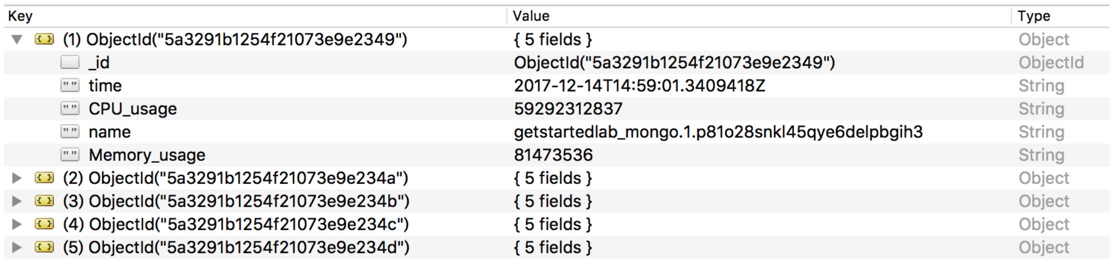
According to all the chart analysis, the usage of CPU is raising or in full load operation. But the memory usage is different.

The memory usage of mongo instance is fluctuating. The memory usage of cAdvisor tends to be stable. And the memory usage of visualizer is in full load operation.

The memory usage of web1 and web2 have the same trend. At first, the usage of memory used has started to rise gradually. The reason for this situation is that the load generator is running. So, the pressure of the server gradually increases and the usage of memory gradually increases. After the increasing the usage of memory tends to stabilize. And this is because the processing speed of the server has reached the upper limit.

Task 5

Task 5 is based on task 4, in task 5 I need to insert the data which I got from task 4 into mongoDB. MongoDB is a database based on distributed file storage, is a non-relational database. The following picture is the data I insert into the mongoDB. I used the database visualization tool Robo 3T to see the data in mongoDB.



Key	Value	Type
▼ (1) ObjectId("5a3291b1254f21073e9e2349")	{ 5 fields }	Object
_id	ObjectId("5a3291b1254f21073e9e2349")	ObjectId
time	2017-12-14T14:59:01.3409418Z	String
CPU_usage	59292312837	String
name	getstartedlab_mongo.1.p81o28snkl45qye6delpbgih3	String
Memory_usage	81473536	String
▶ (2) ObjectId("5a3291b1254f21073e9e234a")	{ 5 fields }	Object
▶ (3) ObjectId("5a3291b1254f21073e9e234b")	{ 5 fields }	Object
▶ (4) ObjectId("5a3291b1254f21073e9e234c")	{ 5 fields }	Object
▶ (5) ObjectId("5a3291b1254f21073e9e234d")	{ 5 fields }	Object

Explanation:

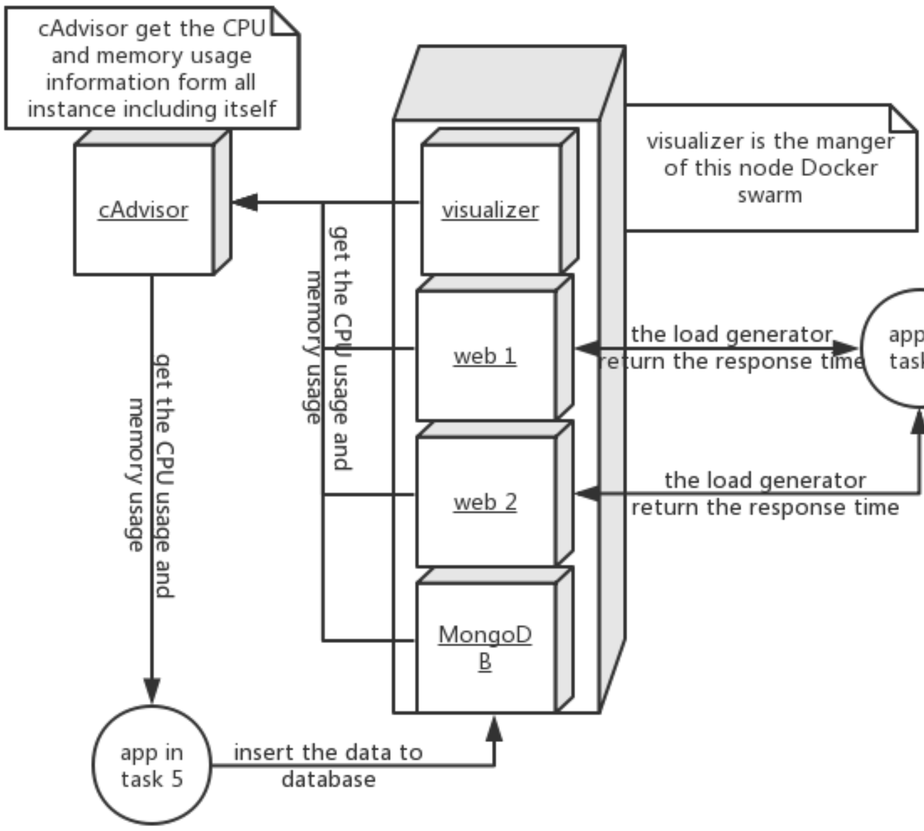
The following code is the way to insert the data into the mongoDB.

```
connection = pymongo.MongoClient('localhost', 3306)
tdb = connection.CloudComputing
post = tdb.task_5
post.insert({"name" : name, "CPU_usage" : cpu_usage[-1], "Memory_usage" :
memory_usage[-1], "time" : timestamp[-1]})
```

The first step of the code is to connect the mongoDB. And then set the name of the database. In my code, my database name is CloudComputing. The third step is set the sheet name. In my code, the sheet name is task_5. And the final step is insert the data. In my code, I insert the name, the CPU_usage, the Memory_usage and the time into the database.

System model

System UML diagram


cAdvisor get the CPU and memory usage information form all instance including itself
cAdvisor
visualizer
web 1
web 2
MongoDB
visualizer is the manger of this node Docker swarm
get the CPU usage and memory usage
get the CPU usage and memory usage
the load generator return the response time
the load generator return the response time
app in task 3
app in task 5
insert the data to database

In this system, visualizer is the manager of the node Docker swarm. Web1 and web2 is test instance created by Docker image nclcloudcomputing/javabenchmarkapp. MongoDB is the database which to save the data of usage of the CPU and memory. CAdvisor is the tool instance to get the CPU and memory usage data. The function of app in task 3 is the load generator, adding pressure to the web 1 and web 2. The function of app in task 5 is to get the CPU usage and memory usage data from cAdvisor and to insert the data to the mongoDB.