

ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



BÁO CÁO

Hệ phân tán

LAB 01

Sinh viên thực hiện: **Nguyễn Thanh Hà**

Mã số sinh viên: **20210298**

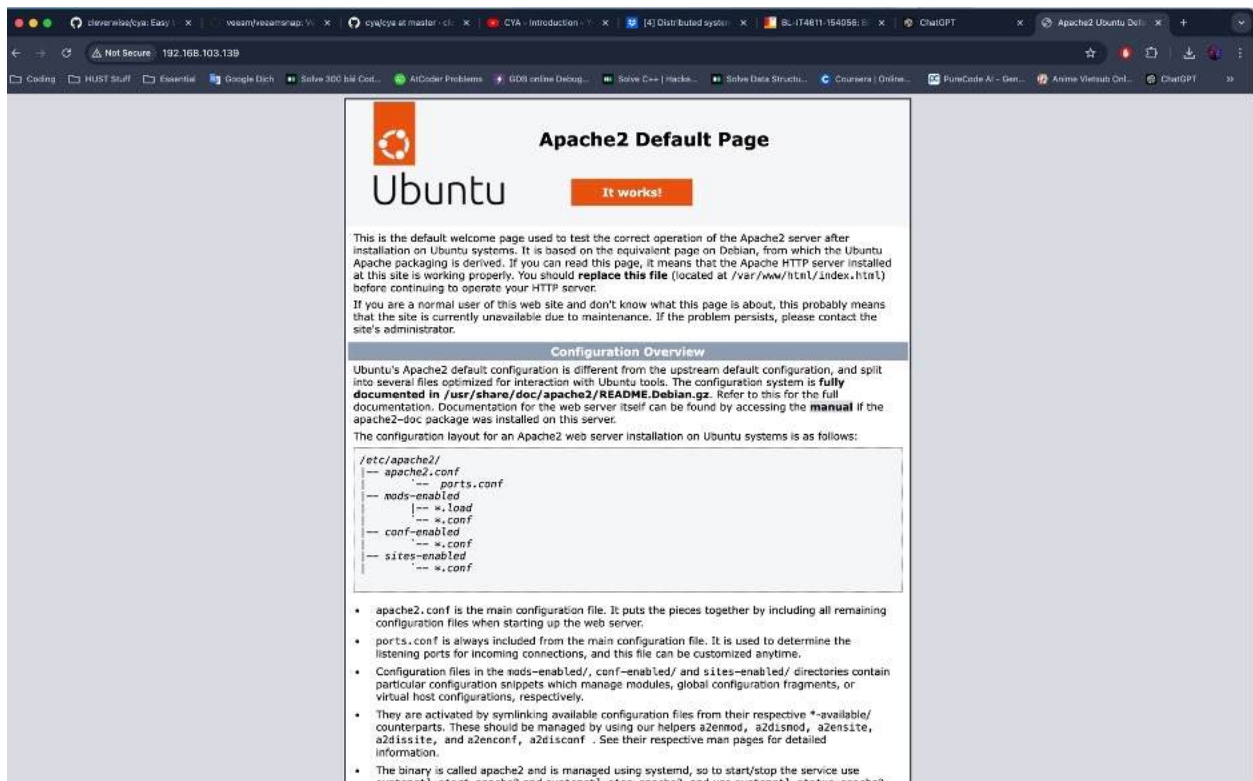
Giảng viên: **TS. Trần Hải Anh**

Hà Nội, 9/2024

1. Web server apache2

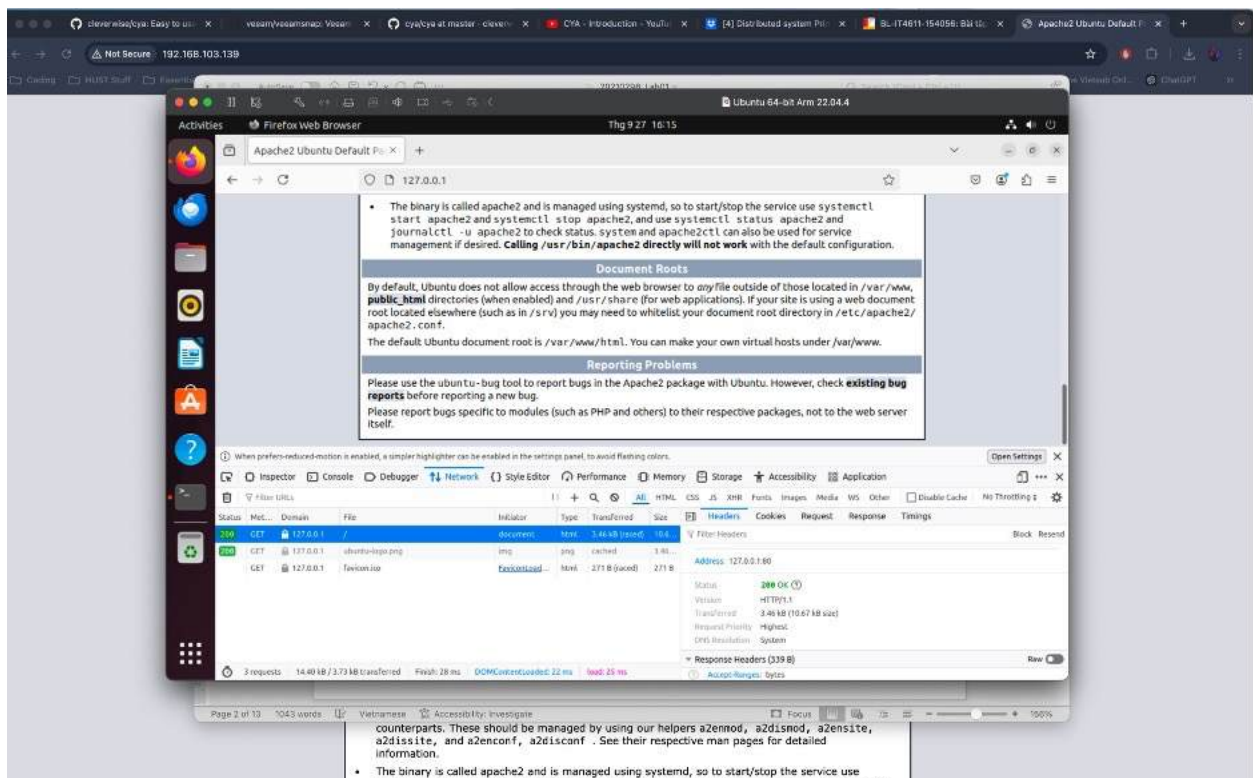
- Câu hỏi 1: Đường dẫn đến file html chứa nội dung mặc định của trang web các bạn vừa xem là gì?

TL: /var/www/html/index.html



- Câu hỏi 2: Cổng mặc định của dịch vụ www là gì?

TL: Cổng 80



- Câu hỏi 3: Hãy giải thích quyền mạng số 755 là gì?

Quyền mạng số 755 có nghĩa là:

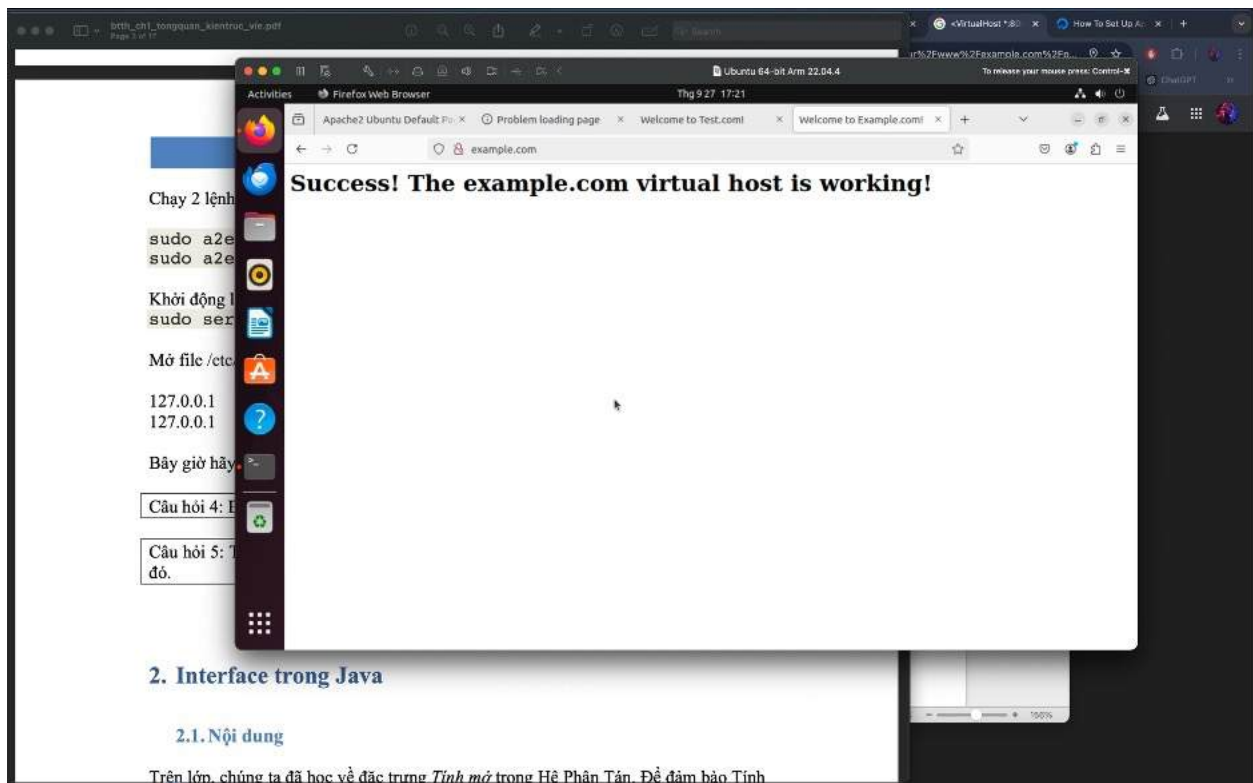
$7 = 4 + 2 + 1$: Người sở hữu thư mục có quyền đọc thư mục (read), chỉnh sửa thư mục (write) và liệt kê các thư mục bên trong (execute).

$5 = 4 + 0 + 1$: Những người cùng nhóm chỉ có quyền đọc thư mục (read), liệt kê các thư mục và file bên trong (execute).

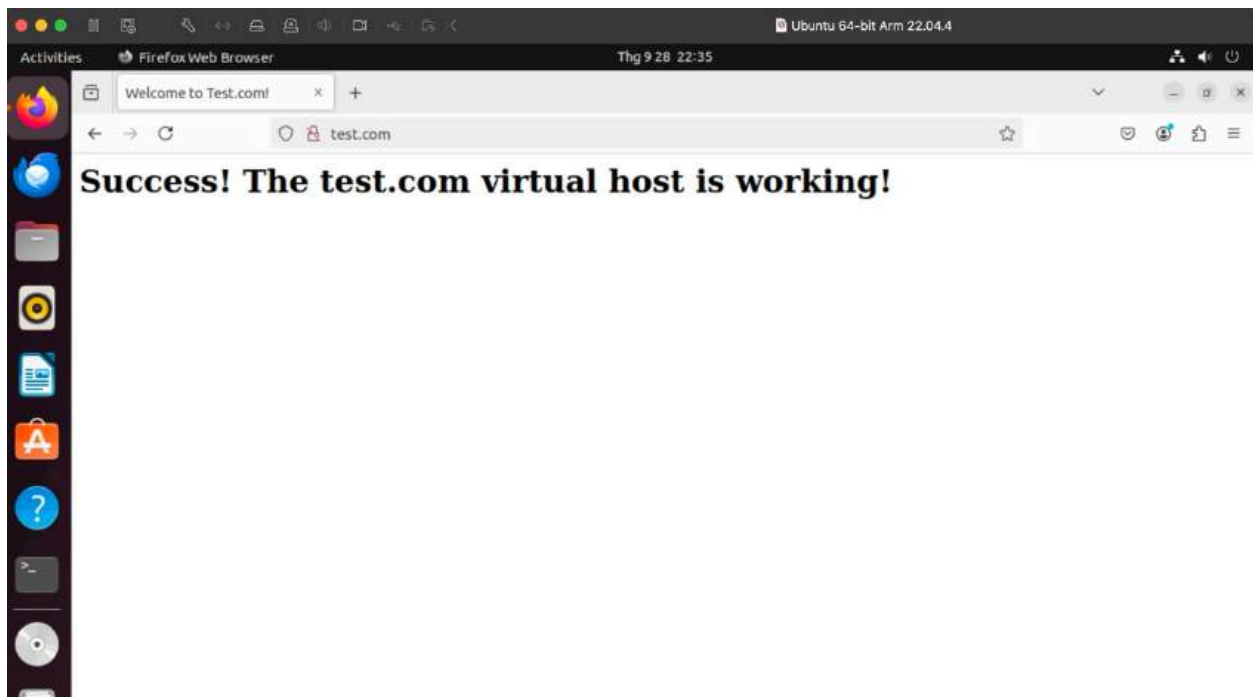
$5 = 4 + 0 + 1$: Những người còn lại chỉ có quyền đọc thư mục (read), liệt kê các thư mục và file bên trong (execute).

Câu hỏi 4: Bạn quan sát thấy nội dung gì sau khi gõ 2 địa chỉ trên? Giải thích.

- “Welcome to Example.com!”, “Success! The example.com virtual host is working!” trong phần nội dung.



- “Welcome to Test.com!”, “Success! The test.com virtual host is working!” trong phần nội dung.



- - Giải thích: Khi ta nhập vào 2 địa chỉ là example.com và test.com, trình duyệt người dùng sẽ gửi yêu cầu tải trang web đó lên 2 server có tên là test.com và example.com. Sau đó, Apache sẽ truy cập vào đường dẫn thư mục /var/www/test.com/public_html hoặc example.com/public_html và lấy ra tất cả các file cấu thành lên trang là 2 file index.html, chạy và hiển thị lên trình duyệt nội dung của nó là nội dung mà ta nhìn thấy

Câu hỏi 5: Thử truy cập từ các máy tính khác trong cùng mạng LAN vào 2 trang web đó.

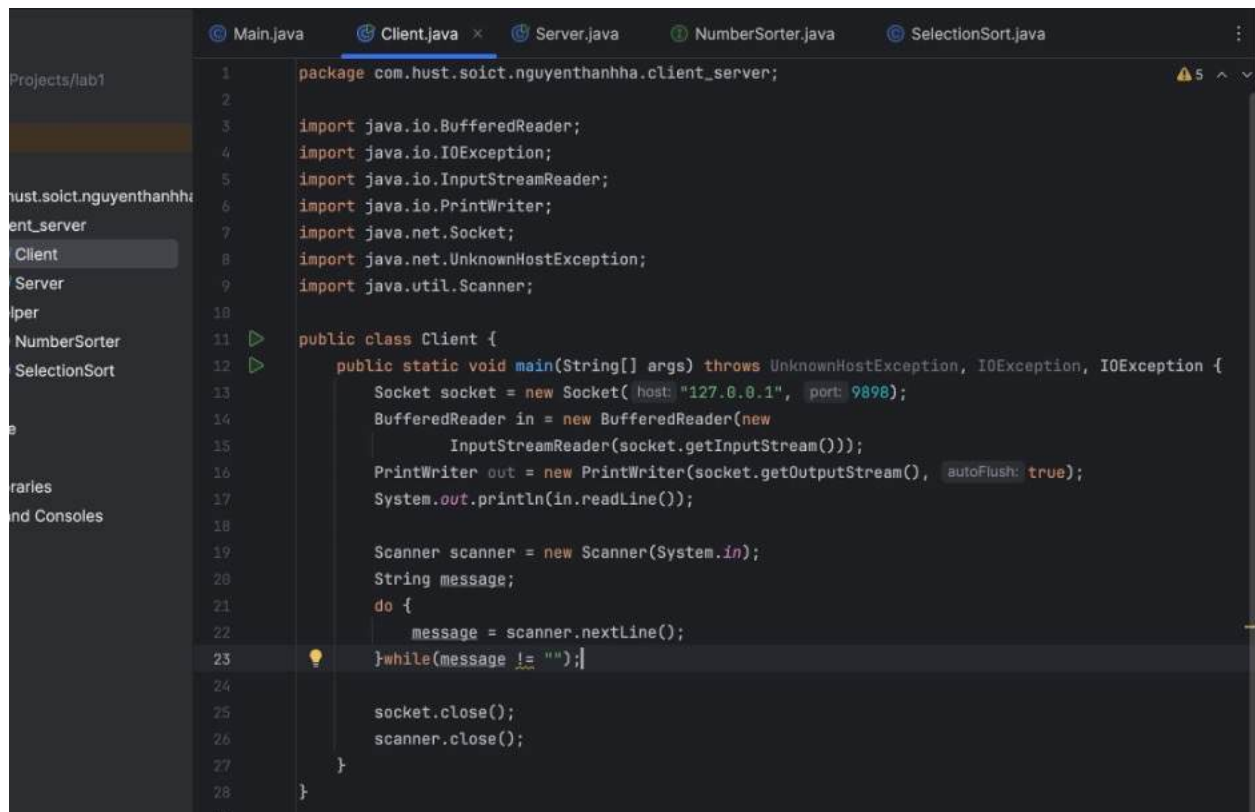
- Hiện tại không thể truy cập được.
- Phải cấu hình file hosts và proxy mới có thể truy cập được.

2. Interface trong Java

- *Câu hỏi 6: Hãy tự viết một đoạn code để thực hiện 1 vòng lặp while sao cho nó sẽ nhận các số mà người dùng gõ và gửi về server, cho đến khi nào người dùng gõ ký tự rỗng rồi ấn enter.*

Gợi ý: hãy dùng lệnh sau để nhận xâu ký tự người dùng gõ vào: `String message = scanner.nextLine();`

-TL:



```
1 package com.hust.soict.nguyenthanhha.client_server;
2
3 import java.io.BufferedReader;
4 import java.io.IOException;
5 import java.io.InputStreamReader;
6 import java.io.PrintWriter;
7 import java.net.Socket;
8 import java.net.UnknownHostException;
9 import java.util.Scanner;
10
11 public class Client {
12     public static void main(String[] args) throws UnknownHostException, IOException, IOException {
13         Socket socket = new Socket("127.0.0.1", 9898);
14         BufferedReader in = new BufferedReader(new
15             InputStreamReader(socket.getInputStream()));
16         PrintWriter out = new PrintWriter(socket.getOutputStream(), autoFlush: true);
17         System.out.println(in.readLine());
18
19         Scanner scanner = new Scanner(System.in);
20         String message;
21         do {
22             message = scanner.nextLine();
23         } while (message != "");
24
25         socket.close();
26         scanner.close();
27     }
28 }
```

- Câu hỏi 7: Vai trò của phương thức run là gì? Khi nào thì nó được gọi?

- TL: Vai trò của phương thức run(): Lấy một chuỗi số từ client gửi lên cách nhau bằng khoảng trống, chuyển chuỗi thành mảng integer sau đó sắp xếp theo thứ tự tăng dần và gửi lại về cho client. Nó sẽ chạy đoạn code thông qua một luồng kết nối client-server.
- Nó được gọi khi: Có yêu cầu gửi lên từ phía client có số hiệu cổng trùng với số hiệu cổng của server. Phương thức run() sẽ được gọi tự động khi phương thức start() được gọi để tạo một luồng mới.

3. Kiến trúc Microservices

Build/rebuild 3 dịch vụ:


```
mvnw
...
[INFO] Results:
[INFO]
[INFO] Tests run: 0, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] --- maven-jar-plugin:3.2.0:jar (default-jar) @ microservice-kubernetes-demo-order ---
[INFO] Building jar: /Users/hant/microservice-kubernetes-demo/microservice-kubernetes-demo-order/target/microservice-kubernetes-demo-order-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.3.4.RELEASE:repackage (repackage) @ microservice-kubernetes-demo-order ---
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] Reactor Summary:
[INFO]
[INFO] microservice-kubernetes-demo ..... SUCCESS [ 9.383 s]
[INFO] microservice-kubernetes-demo-customer ..... SUCCESS [ 17.791 s]
[INFO] microservice-kubernetes-demo-catalog ..... SUCCESS [ 4.982 s]
[INFO] microservice-kubernetes-demo-order ..... SUCCESS [ 4.234 s]
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 36.543 s
[INFO] Finished at: 2024-09-26T20:08:57:00
[INFO] Final Memory: 41M/168M
[INFO]
...
~/microservice-kubernetes-demo
```

Câu hỏi 1: Hãy thực hiện gõ những lệnh tương tự như trên với 3 dịch vụ còn lại.

```
#!/bin/sh
if [ -z "$DOCKER_ACCOUNT" ]; then
    DOCKER_ACCOUNT=nthhaf
fi
docker build --tag=microservice-kubernetes-demo-apache apache
docker tag microservice-kubernetes-demo-apache $DOCKER_ACCOUNT/microservice-kubernetes-demo-apache:latest
docker push $DOCKER_ACCOUNT/microservice-kubernetes-demo-apache
...
docker build --tag=microservice-kubernetes-demo-catalog microservice-kubernetes-demo-catalog
docker tag microservice-kubernetes-demo-catalog $DOCKER_ACCOUNT/microservice-kubernetes-demo-catalog:latest
docker push $DOCKER_ACCOUNT/microservice-kubernetes-demo-catalog
...
docker build --tag=microservice-kubernetes-demo-customer microservice-kubernetes-demo-customer
docker tag microservice-kubernetes-demo-customer $DOCKER_ACCOUNT/microservice-kubernetes-demo-customer:latest
docker push $DOCKER_ACCOUNT/microservice-kubernetes-demo-customer
...
docker build --tag=microservice-kubernetes-demo-order microservice-kubernetes-demo-order
docker tag microservice-kubernetes-demo-order $DOCKER_ACCOUNT/microservice-kubernetes-demo-order:latest
docker push $DOCKER_ACCOUNT/microservice-kubernetes-demo-order
...
What's next:
View a summary of image vulnerabilities and recommendations → docker scout quickview
Using default tag: latest
The push refers to repository [docker.io/nthhaf/microservice-kubernetes-demo-order]
ef90a027f64e: Pushed
258503c0ee88: Mounted from nthhaf/microservice-kubernetes-demo-customer
9bec2abb7470: Mounted from nthhaf/microservice-kubernetes-demo-customer
98016afd555f: Mounted from nthhaf/microservice-kubernetes-demo-customer
00b0243baac3: Mounted from nthhaf/microservice-kubernetes-demo-customer
fb40e446f1da: Mounted from nthhaf/microservice-kubernetes-demo-customer
141bfa9b93fc: Mounted from nthhaf/microservice-kubernetes-demo-customer
latest: digest: sha256:48955cb51fe6371054bfc54bc836c5793f8507497106f69418533d72c2246b33 size: 1784
...
~/microservice-kubernetes-demo
```

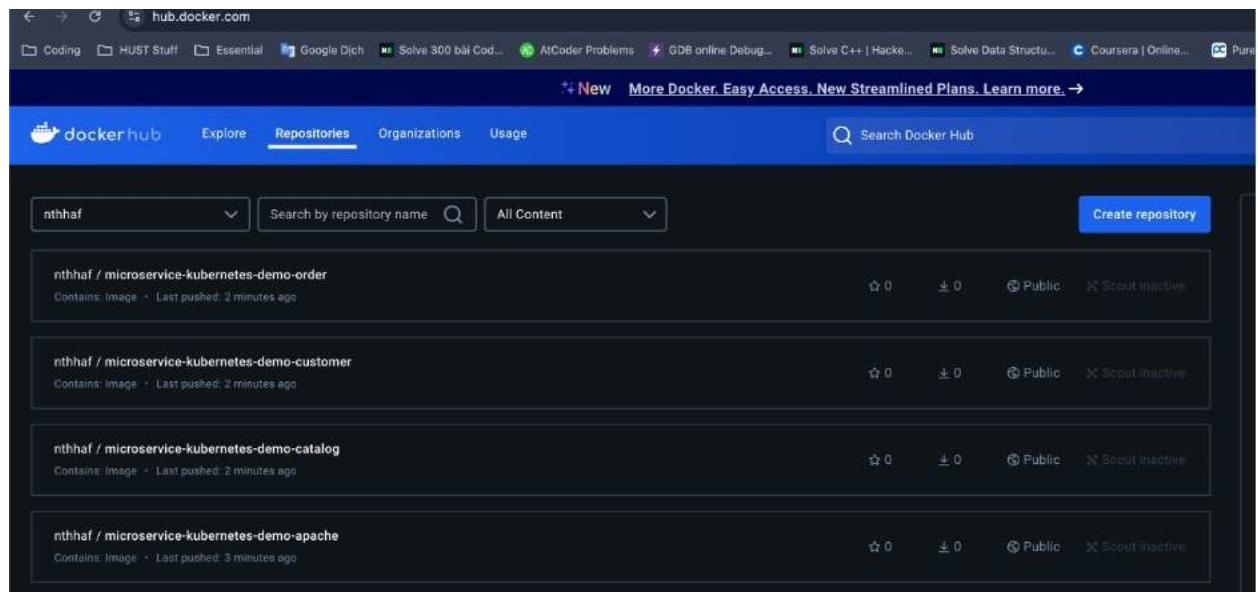
Câu hỏi 2: Vào trang web DockerHub và đăng nhập vào tài khoản của bạn. Bạn thấy những gì mới xuất hiện trên docker hub repository của bạn?

nthhaf/microservice-kubernetes-demo-apache

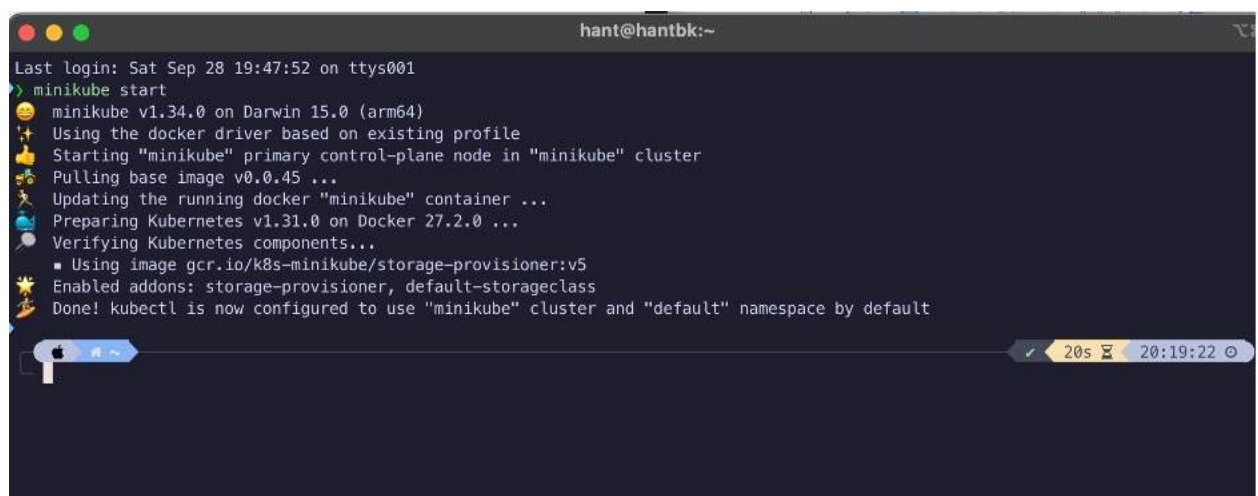
nthhaf/microservice-kubernetes-demo-catalog

nthhaf/microservice-kubernetes-demo-customer

nthhaf/microservice-kubernetes-demo-order



minikube start



kubectl apply -f microservices.yaml

```
> External Libraries
> Scratches and Consoles
190 | run: order
191 | type: LoadBalancer
192 | status:
193 | loadBalancer: {}
194 |

Terminal Local x + v
> kubectl version
Client Version: v1.31.1
Kustomize Version: v5.4.2
Server Version: v1.31.0
> kubectl apply -f microservices.yaml
deployment.apps/apache created
deployment.apps/catalog created
deployment.apps/customer created
deployment.apps/order created
service/apache created
service/catalog created
service/customer created
service/order created
```

kubectl get all

```
hant@hantbk:~
> kubectl get all
NAME                                     READY   STATUS    RESTARTS   AGE
pod/apache-7cd8f994b9-rzp2p            1/1     Running   0           71s
pod/catalog-65bd69fc5d-5nnrv           1/1     Running   0           71s
pod/customer-59fdd75475-vv4lh          1/1     Running   0           71s
pod/order-74f698f4fc-bkkvx            1/1     Running   0           71s

NAME                TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
service/apache      LoadBalancer  10.104.69.19   <pending>      80:30712/TCP     71s
service/catalog     LoadBalancer  10.108.146.6   <pending>      8080:30302/TCP   71s
service/customer    LoadBalancer  10.111.242.206 <pending>      8080:32670/TCP   71s
service/kubernetes  ClusterIP     10.96.0.1      <none>         443/TCP          45m
service/order       LoadBalancer  10.104.222.40  <pending>      8080:30782/TCP   71s

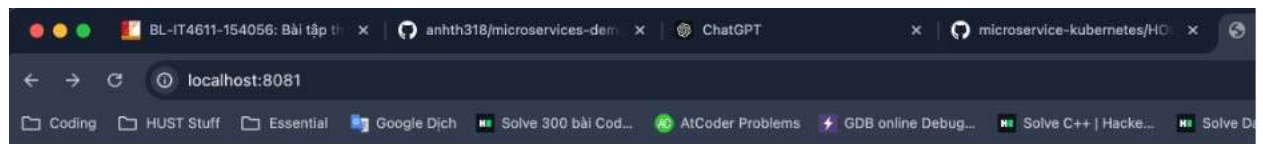
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/apache  1/1     1             1           71s
deployment.apps/catalog  1/1     1             1           71s
deployment.apps/customer  1/1     1             1           71s
deployment.apps/order    1/1     1             1           71s

NAME                DESIRED   CURRENT   READY   AGE
replicaset.apps/apache-7cd8f994b9  1         1         1       71s
replicaset.apps/catalog-65bd69fc5d  1         1         1       71s
replicaset.apps/customer-59fdd75475  1         1         1       71s
replicaset.apps/order-74f698f4fc    1         1         1       71s
```

Câu hỏi 3: Trạng thái (status) của các pods vừa mới tạo được là gì? Bây giờ, hãy chờ vài phút và gõ lại lệnh đó, trạng thái mới của các pods giờ đã chuyển thành gì?

TL: Ban đầu các pods mới tạo ở trạng thái “ContainerCreating”, sau đó chuyển sang trạng thái “Running”.

kubectl port-forward deployment/apache 8081:80

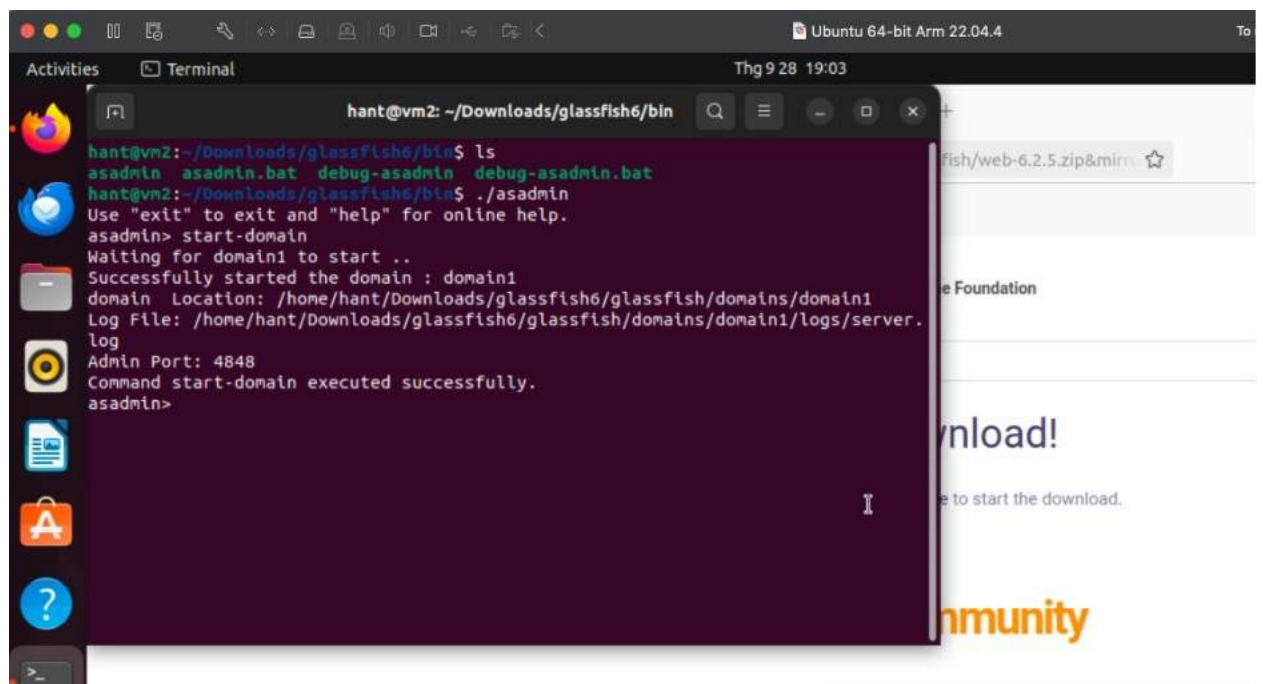


Order Processing

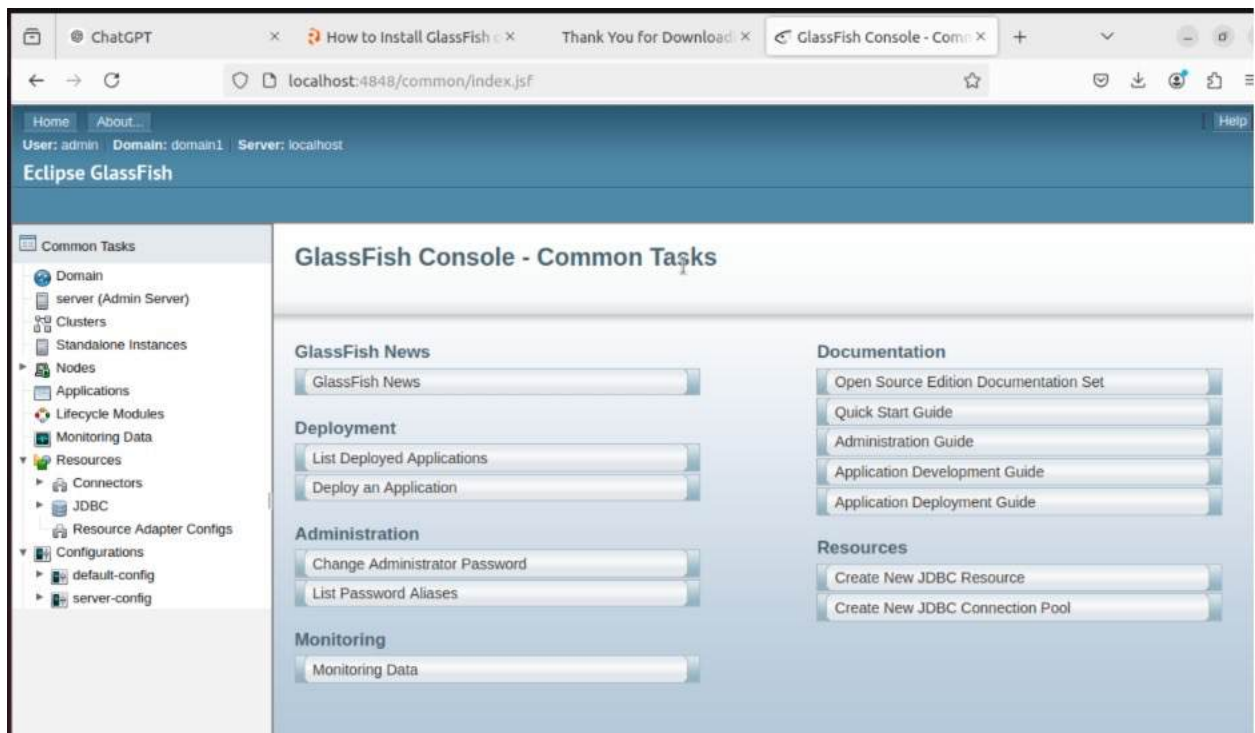
Customer
Catalog
Catalog
Order

List / add / remove customers
List / add / remove items
Search Items
Create an order

4. Kiến trúc JMS và DDS

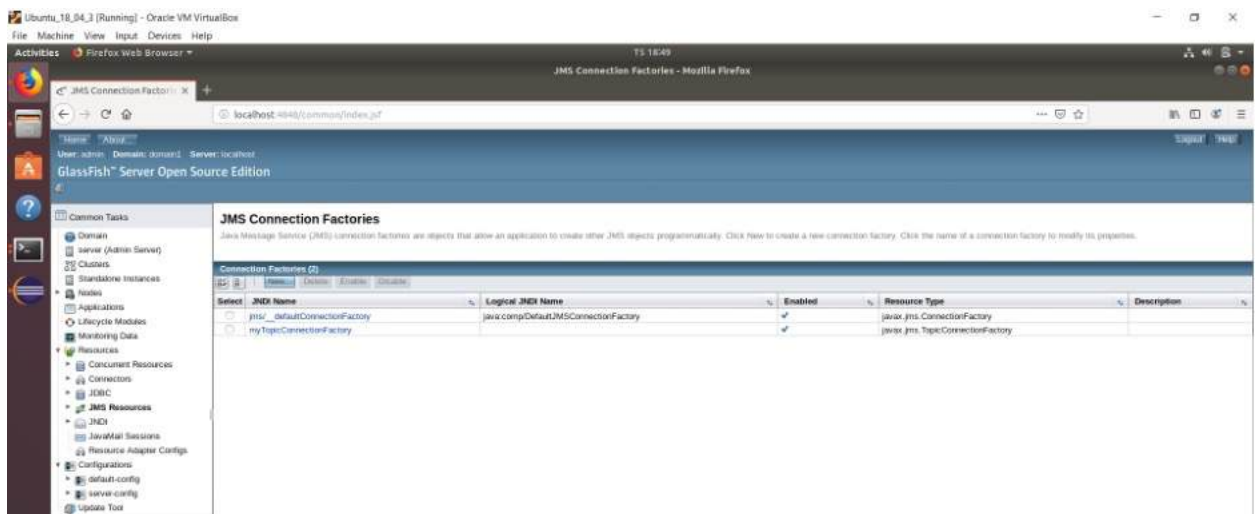


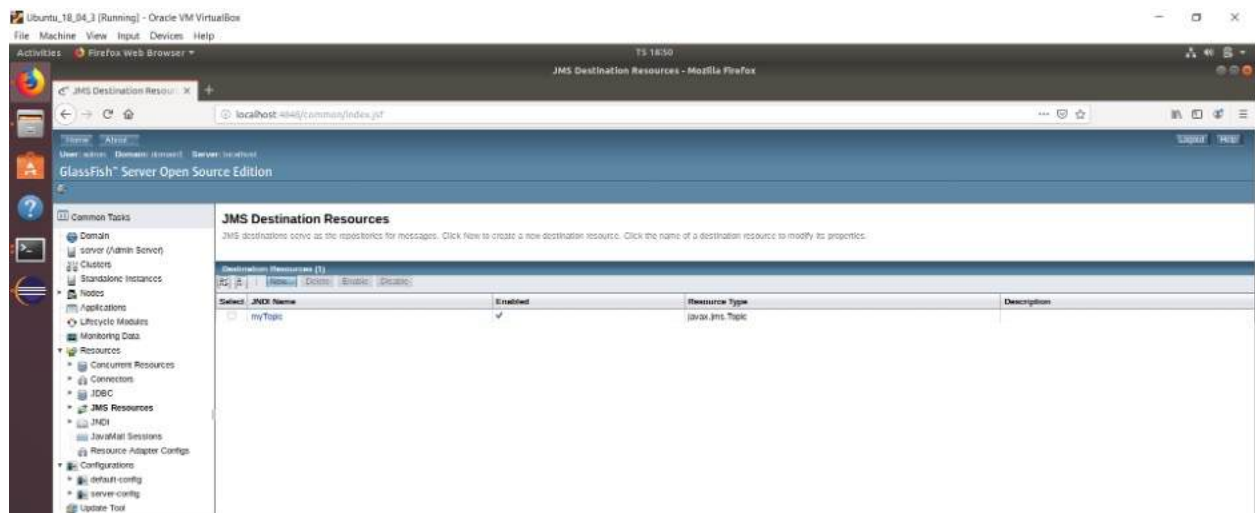
Câu hỏi 1: Giải thích vai trò của application server glassfish.



- Glassfish cung cấp một ứng dụng server cho phép triển khai các ứng dụng web viết bằng Java.
- Glassfish cho phép nhiều nhà phát triển có khả năng mở rộng, tích hợp các công nghệ trên web và cấu hình dễ dàng bằng giao diện đồ họa.

Câu hỏi 2: Tại sao lại phải tạo 2 JNDI như trên?





- Phải tạo 2 JNDI vì

myTopicConnectionFactory để GlassFish tạo một connector connection pool và connector resource.

myTopic để GlassFish tạo connector admin object resource.

Câu hỏi 3: Sau khi chạy thử chương trình Sender và Receiver, vận dụng lý thuyết kiến trúc hướng sự kiện đã học trên lớp để giải thích cơ chế chuyển và nhận thông điệp của Sender và Receiver.

Cơ chế chuyển và nhận thông điệp của Sender và Receiver:

- Cơ chế chuyển (Sender):

Khi có một nguyên nhân, kích thích làm thay đổi một sự kiện nào đó, bên Sender sẽ gửi một thông điệp (message) cho bên nhận. Thông điệp ấy được nằm trong bộ nhớ tạm, nếu có tín hiệu chuyển phát (commit) mới thì thông điệp mới được chuyển sang cho người nhận.

- Cơ chế nhận (Receiver):

Khi nhận được một thông điệp được gửi đến. Nếu chưa kịp xử lý thì bên nhận đưa tạm vào bộ nhớ đệm, nếu bộ nhớ đầy thì báo lỗi và sử dụng cơ chế rollback gửi lại cho người gửi. Trong quá trình xử lý gặp lỗi thì phía bên nhận cũng phải báo lại cho bên gửi

Câu hỏi 4: So sánh JMS và DDS.

JMS (Java Message Service)

- **Ngôn ngữ:** JMS là một API dành cho Java để xử lý giao tiếp không đồng bộ giữa các ứng dụng.
- **Kiến trúc:** JMS chủ yếu dựa trên mô hình tin nhắn publish-subscribe và point-to-point.
 - **Point-to-Point (P2P):** Một người gửi gửi tin nhắn đến một hàng đợi (queue), và một người nhận lấy tin nhắn từ hàng đợi đó.
 - **Publish-Subscribe:** Nhiều người nhận có thể đăng ký để nhận tin nhắn từ cùng một nguồn.
- **Chất lượng dịch vụ (QoS):** JMS có thể đảm bảo độ tin cậy, nhưng điều này phụ thuộc vào cách triển khai của nhà cung cấp.
- **Ứng dụng:** JMS được sử dụng chủ yếu trong các ứng dụng doanh nghiệp (Enterprise Applications), nơi cần giao tiếp không đồng bộ giữa các thành phần hệ thống.
- **Hệ sinh thái:** JMS chủ yếu gắn liền với nền tảng Java và các ứng dụng doanh nghiệp (Java EE).

DDS (Data Distribution Service)

- **Ngôn ngữ:** DDS là một tiêu chuẩn cho hệ thống truyền thông hướng dữ liệu (data-centric communication). DDS hỗ trợ nhiều ngôn ngữ lập trình khác nhau như C++, Java, Python, v.v.
- **Kiến trúc:** DDS sử dụng mô hình publish-subscribe, nhưng nó tập trung vào trao đổi dữ liệu thay vì tin nhắn đơn lẻ.
 - **Publish-Subscribe:** Giống như JMS, nhưng DDS cung cấp tính năng mạnh mẽ hơn trong việc quản lý dữ liệu, bao gồm việc mô tả các luồng dữ liệu và điều chỉnh chất lượng dịch vụ (QoS) cho từng loại dữ liệu.
- **Chất lượng dịch vụ (QoS):** DDS cung cấp các mức QoS rất chi tiết, bao gồm độ trễ, độ tin cậy, và khả năng xử lý sự kiện thời gian thực.
- **Ứng dụng:** DDS thường được sử dụng trong các hệ thống nhúng (embedded systems), các hệ thống yêu cầu độ trễ thấp và tính ổn định cao như trong lĩnh vực quân sự, y tế, hay xe tự lái.
- **Hệ sinh thái:** DDS được sử dụng rộng rãi trong các hệ thống phân tán thời gian thực và các hệ thống IoT (Internet of Things).