# BÀI THỰC HÀNH SỐ 1

# CÁC HỆ THỐNG PHÂN TÁN VÀ ỨNG DỤNG

# CHƯƠNG 1: TỔNG QUAN VÀ KIẾN TRÚC HPT

**Họ và tên: Nguyễn Duy Khánh** Mã lớp: 157542

**MSSV: 20225019** Mã học phần: IT4611

## 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?


Đường dẫn đến file html chứa nội dung mặc định của trang web vừa xem là: **/var/www/html/index.html**

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

Cổng mặc định của dịch vụ www là **80**.

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

Trả lời:

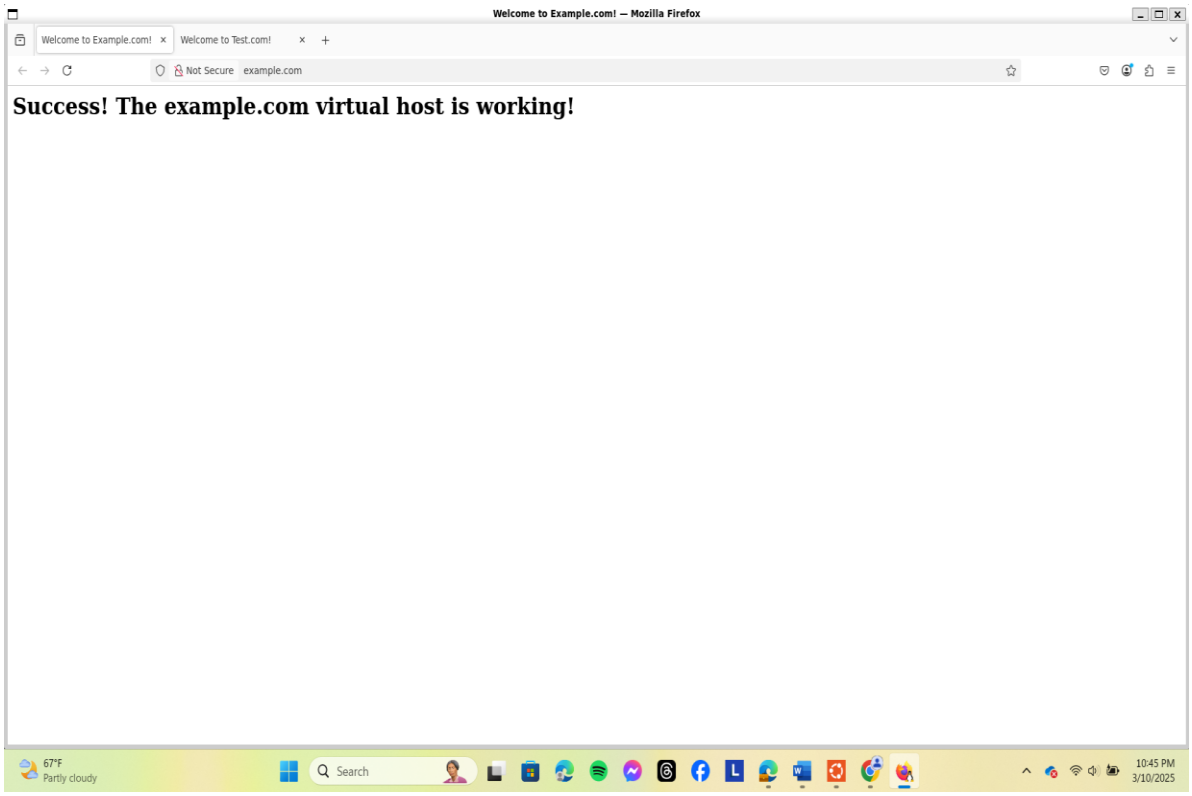
Quyền mang số 755 có nghĩa là:

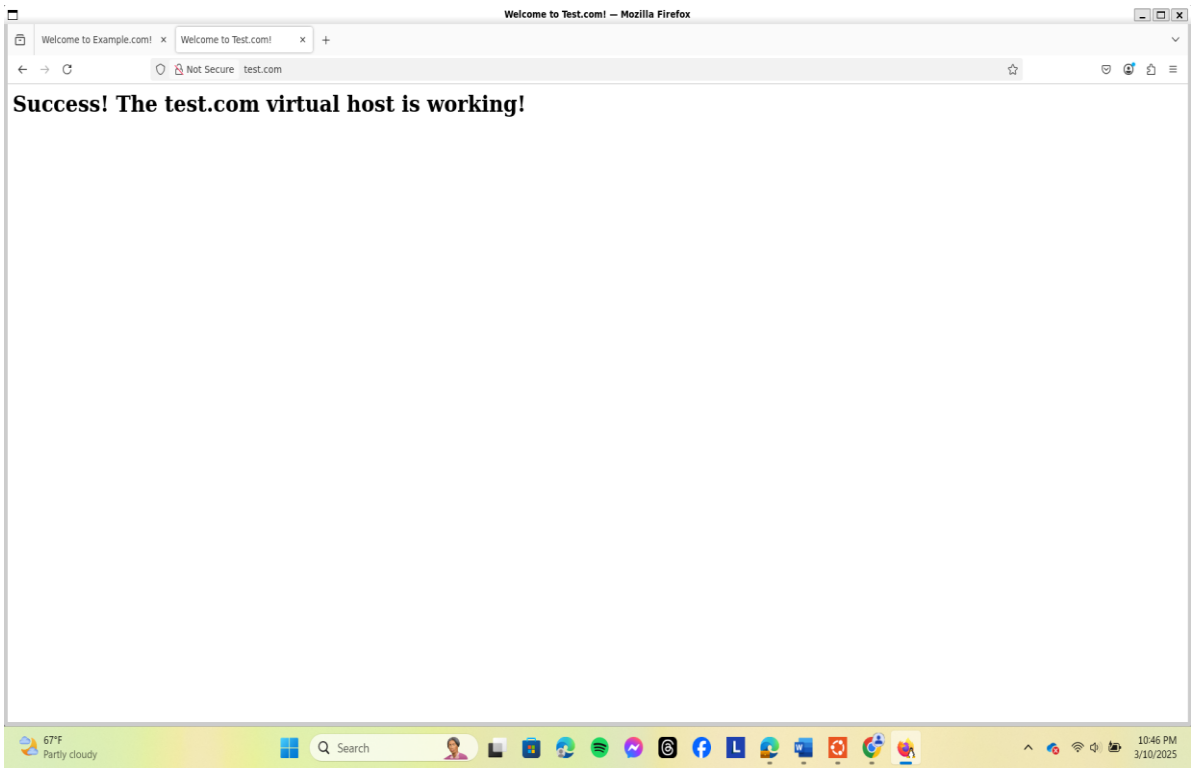
7 = 4 + 2 + 1: Người sở hữu thư mục có quyền đọc thư mục(read), chỉnh sửa (write) và thực thi(execute) đối với các file bên trong thư mục đó

5: Những người cùng nhóm (group) chỉ có quyền đọc và thực thi

5: Những người còn lại(others) cũng chỉ có quyền đọc và thực thi

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




Giải thích: Khi 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, Khi đó thì các dòng “Success! The example.com virtual host is working” được hiển thị cùng tiêu đề của trang

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

Trả lời: Có thể truy cập được 2 trang web trên từ máy tính khác cùng mạng LAN

## 2. Interface trong Java

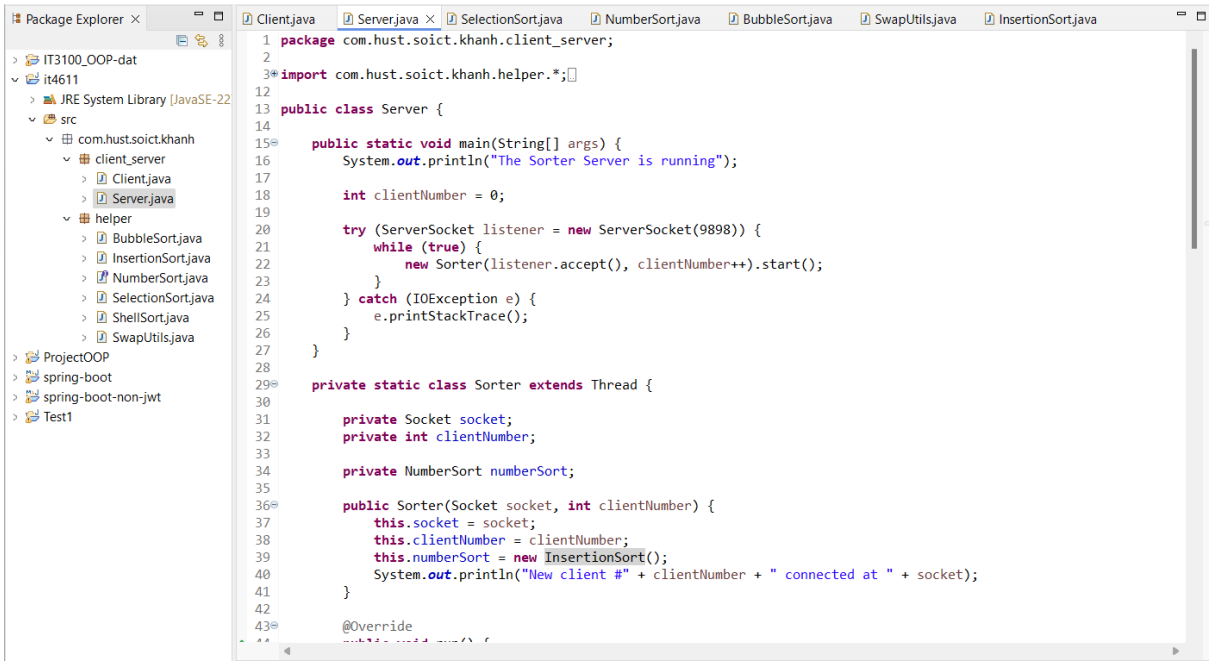
Câu hỏi 6: Hãy tự viết một đoạn code để thực hiện 1 vòng lặp while sao cho nó sẽ nhận các số mà người dùng gõ và gửi về server, cho đến khi nào người dùng gõ ký tự rỗng rồi ấn enter.

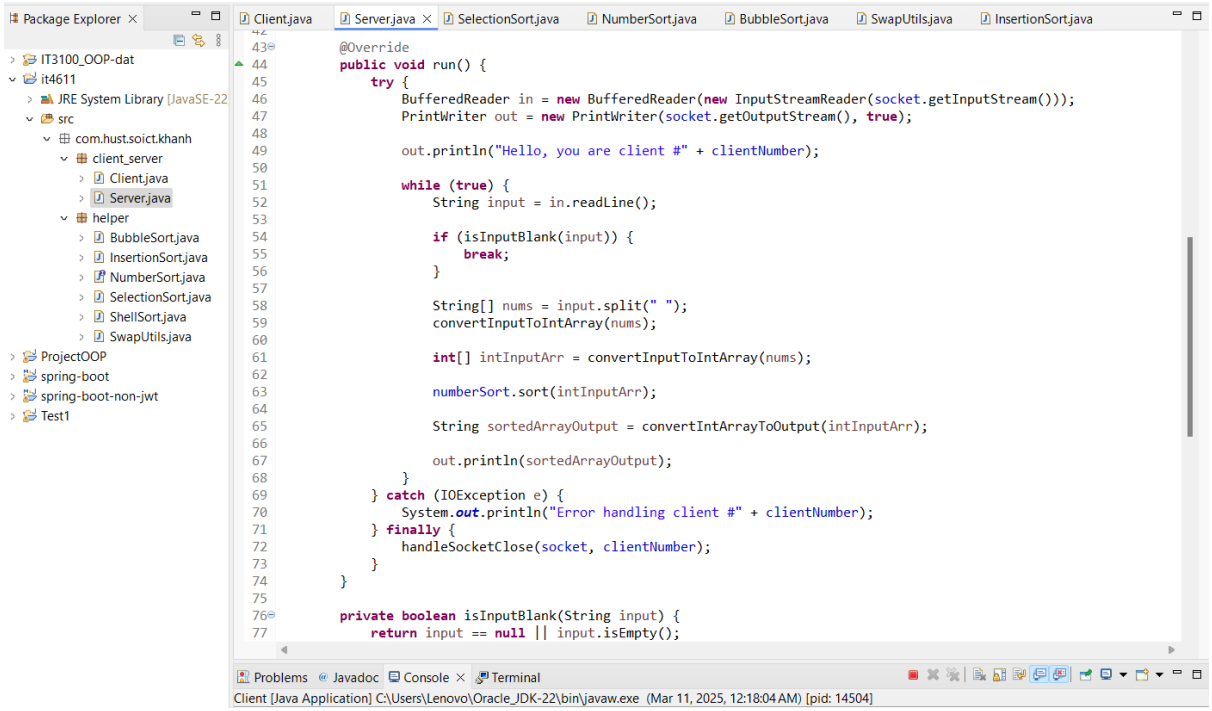
Gợi ý: hãy dùng lệnh sau để nhận xâu ký tự người dùng gõ vào: String message = scanner.nextLine();


```
package com.hust.soict.khanh.client_server;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

public class Client {
	public static void main(String[] args) {
		Socket socket;
		try {
			socket = new Socket("127.0.0.1", 9898);
			BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
			PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
			System.out.println(in.readLine());
			Scanner scanner = new Scanner(System.in);
			while (true) {
				String message = scanner.nextLine();
				if (message.equals("")) {
```

```
                    break;
                } else {
                    out.println(message);
                    System.out.println(in.readLine());
                }
            }
            socket.close();
            scanner.close();
        } catch (IOException e) {
            System.out.println("Cant connect to server");
            e.printStackTrace();
        }
    }
}
```

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






Vai trò của phương thức run():

- Trong đoạn mã trên, class Sorter được kế thừa từ Thread, phương thức run() được override, nó được gọi ngay khi Thread chạy. Ở hàm main(), ta khởi tạo một đối tượng kế thừa từ Thread và sử dụng phương thức start() để bắt đầu. Khi bắt đầu thread bằng cách gọi hàm start(), phương thức run() sẽ tự động được thực hiện bởi JVM.
- Vai trò của phương thức run(): Thông báo Id của client đang kết nối, lấy chuỗi số từ phía client gửi lên ngăn cách nhau bởi dấu cách, chuyển chuỗi thành mảng integer sau đó sắp xếp theo thứ tự tăng dần và gửi lại về cho client mảng String thông qua luồng kết nối giữa Client – Server
- Phương thức này được gọi khi có yêu cầu gửi lên từ phía client mới muốn kết nối Server có số hiệu cổng trùng với số hiệu cổng của server (9898).

## 3. Kiến trúc Microservice:

```
C:\Users\Lenovo>docker push khanh12toan2k72/microservice-kubernetes-demo-catalog:latest
The push refers to repository [docker.io/khanh12toan2k72/microservice-kubernetes-demo-catalog]
c8dd6fec4403: Pushed
5c58c0c8b5c1: Mounted from khanh12toan2k72/microservice-kubernetes-demo-customer
124765dfe9ce: Mounted from khanh12toan2k72/microservice-kubernetes-demo-customer
fd9537f0cb20: Mounted from khanh12toan2k72/microservice-kubernetes-demo-customer
3ab952050949: Mounted from khanh12toan2k72/microservice-kubernetes-demo-customer
3b0427d99930: Mounted from khanh12toan2k72/microservice-kubernetes-demo-customer
latest: digest: sha256:e52d898c00351180b49ec0e822ba1e8da419603cfcb26a132e6a05c4aba19f05 size: 1577
```

Bây giờ hãy vào thư mục bạn vừa tải về ở trên để build và tải docker image của từng dịch vụ lên DockerHub:
Đối với dịch vụ *apache:*

```
>docker build --tag=microservice-kubernetes-demo-apache
apache

>docker tag microservice-kubernetes-demo-apache

your_docker_account/microservice-kubernetes-demo-
apache:latest

>docker push your_docker_account/microservice-kubernetes-
demo- apache
```

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

- Dịch vụ Order:

```
>docker build --tag=microservice-kubernetes-demo-order
microservice-kubernetes-demo-order

>docker tag microservice-kubernetes-demo-order

khanh12toan2k72/microservice-kubernetes-demo-order:latest

>docker push khanh12toan2k72/microservice-kubernetes-demo-order
```

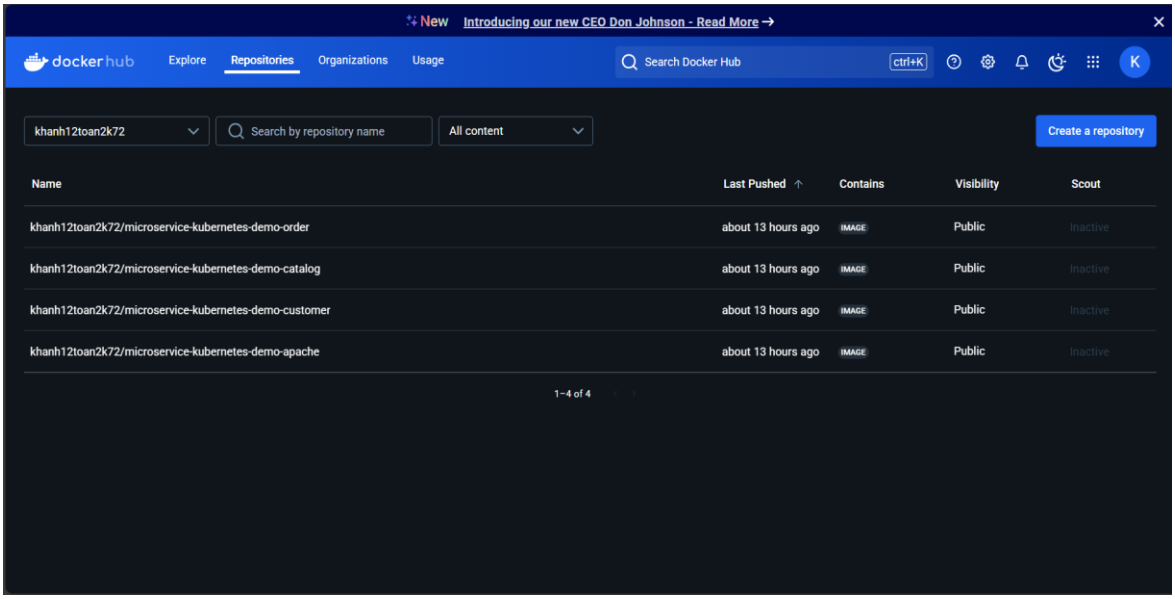
- Dịch vụ Customer:

```
>docker build --tag=microservice-kubernetes-demo-customer
microservice-kubernetes-demo-customer

>docker tag microservice-kubernetes-demo-customer

khanh12toan2k72/microservice-kubernetes-demo-customer:latest

>docker push khanh12toan2k72/microservice-kubernetes-demo-
customer
```

- Dịch vụ Catalog:

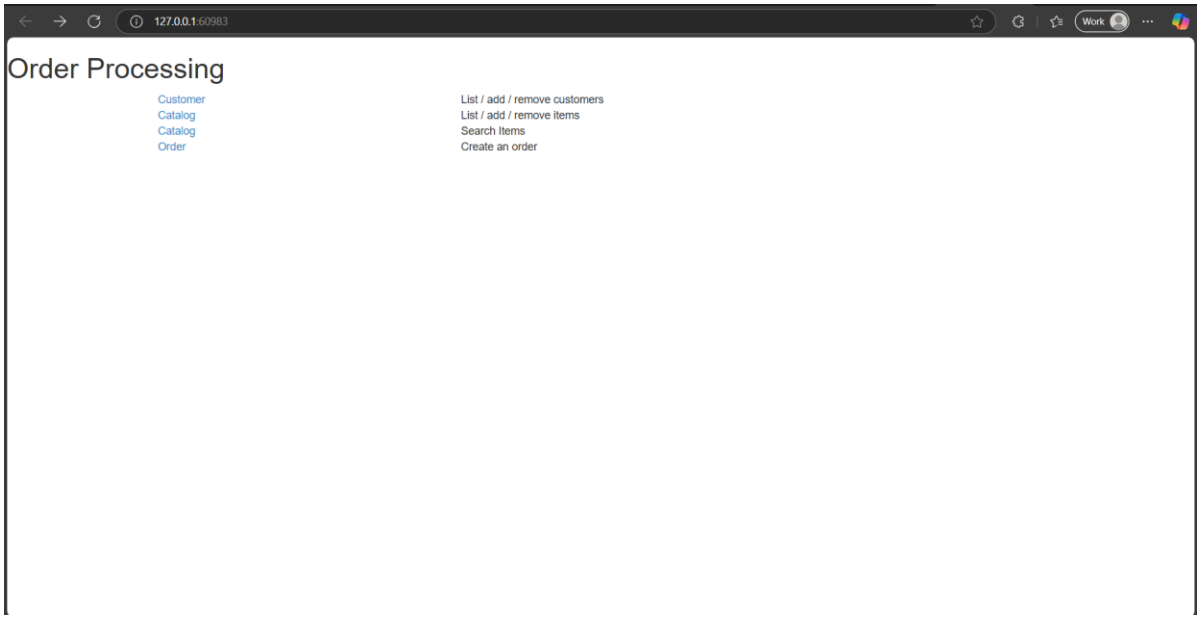
```
>docker build --tag=microservice-kubernetes-demo-catalog
microservice-kubernetes-demo-catalog

>docker tag microservice-kubernetes-demo-catalog

khanh12toan2k72/microservice-kubernetes-demo-customer:latest

>docker push khanh12toan2k72/microservice-kubernetes-demo-
catalog
```

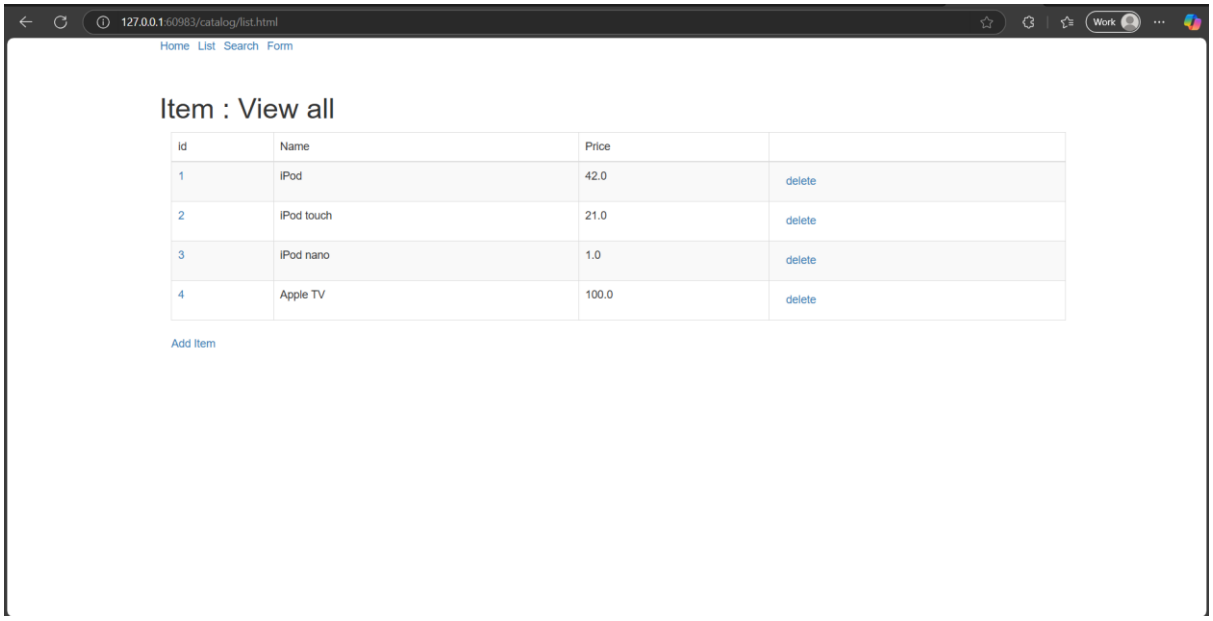
Câu hỏi 2: Vào trang web DockerHub và đăng nhập vào tài khoản của bạn. Bạn thấy những gì mới xuất hiện trên docker hub repository của bạn?



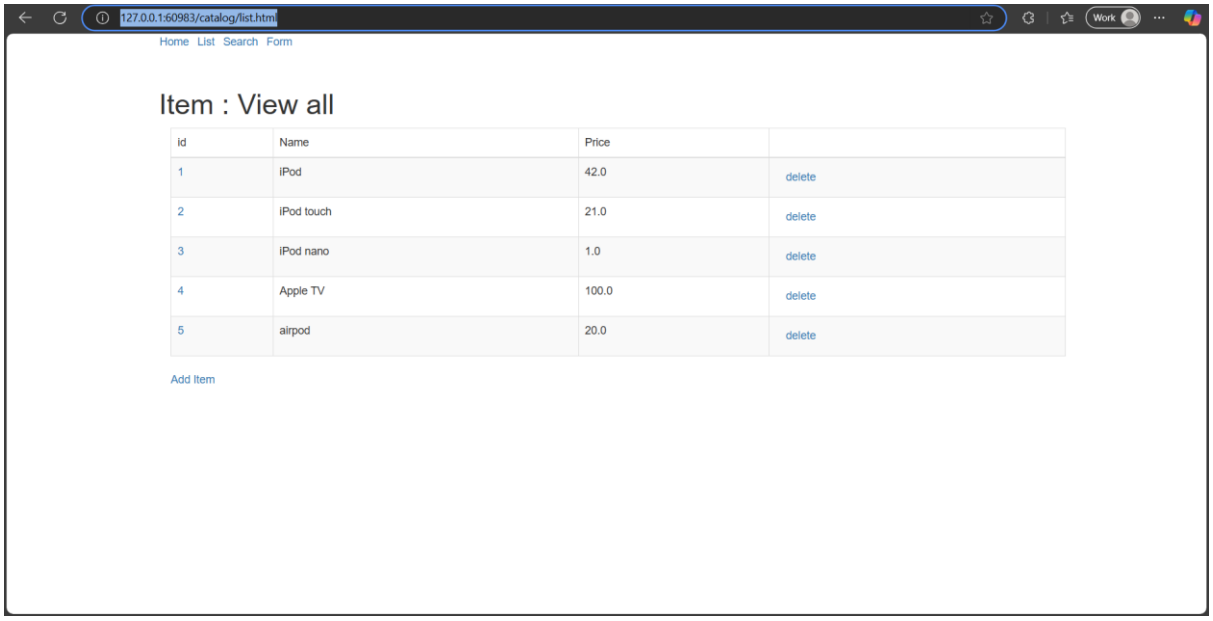

Ta thấy các repositories khanh12toan2k72/microservice-kubernetes-demo-apache, khanh12toan2k72/microservice-kubernetes-demo-catalog, khanh12toan2k72/microservice-kubernetes-demo-customer, khanh12toan2k72/microservice-kubernetes-demo-order

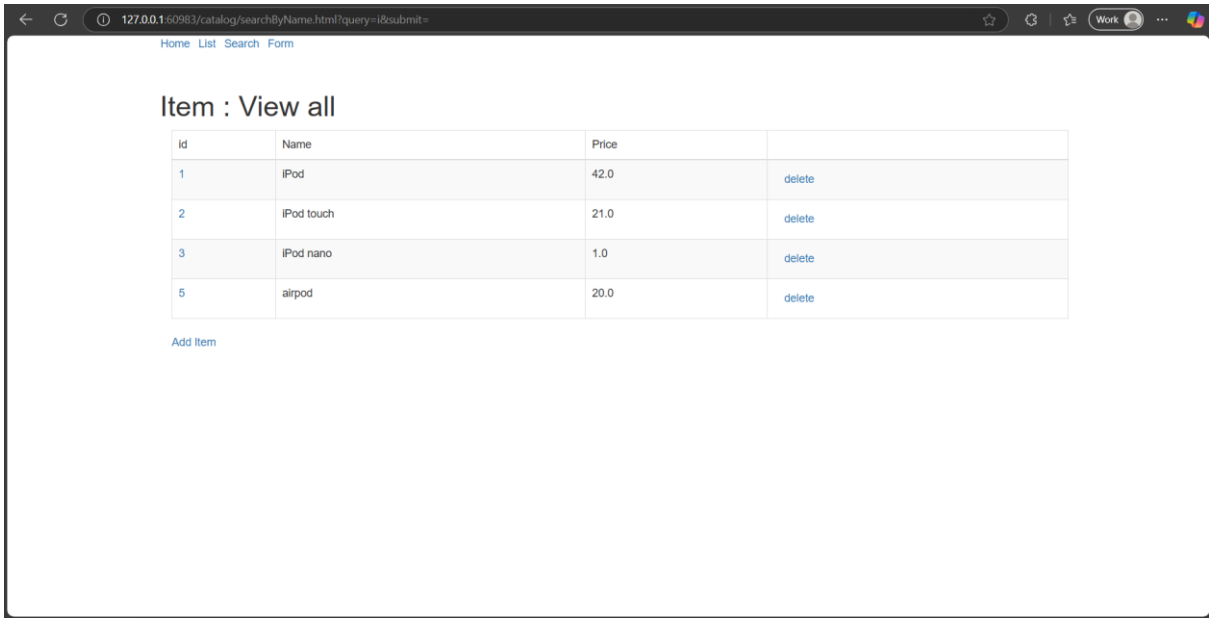
```
C:\Users\Lenovo\Downloads\it4611\microservices-demo>kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
apache-c447cf48f-48t2k      1/1     Running   0          39s
catalog-6b9676d474-k2hlk    1/1     Running   0          39s
customer-6777b4467-t5l9c    1/1     Running   0          39s
order-55dbb6cd69-p5c7h      1/1     Running   0          39s
```

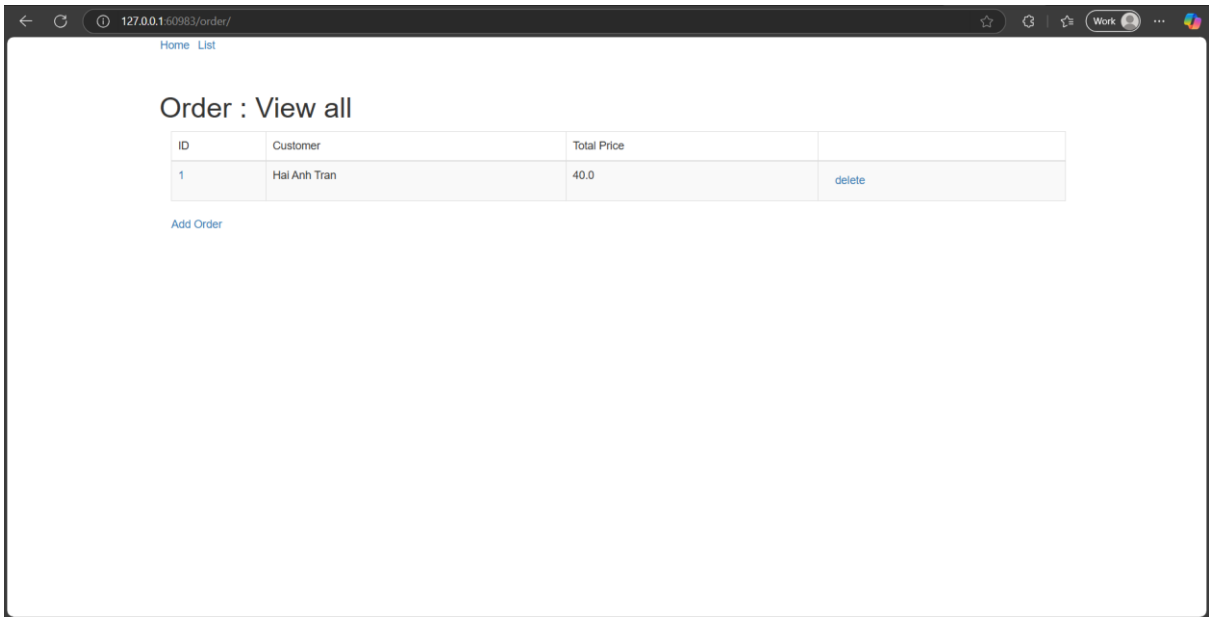





Ví dụ sau khi thêm mới sản phẩm “airpod” có giá 20.0





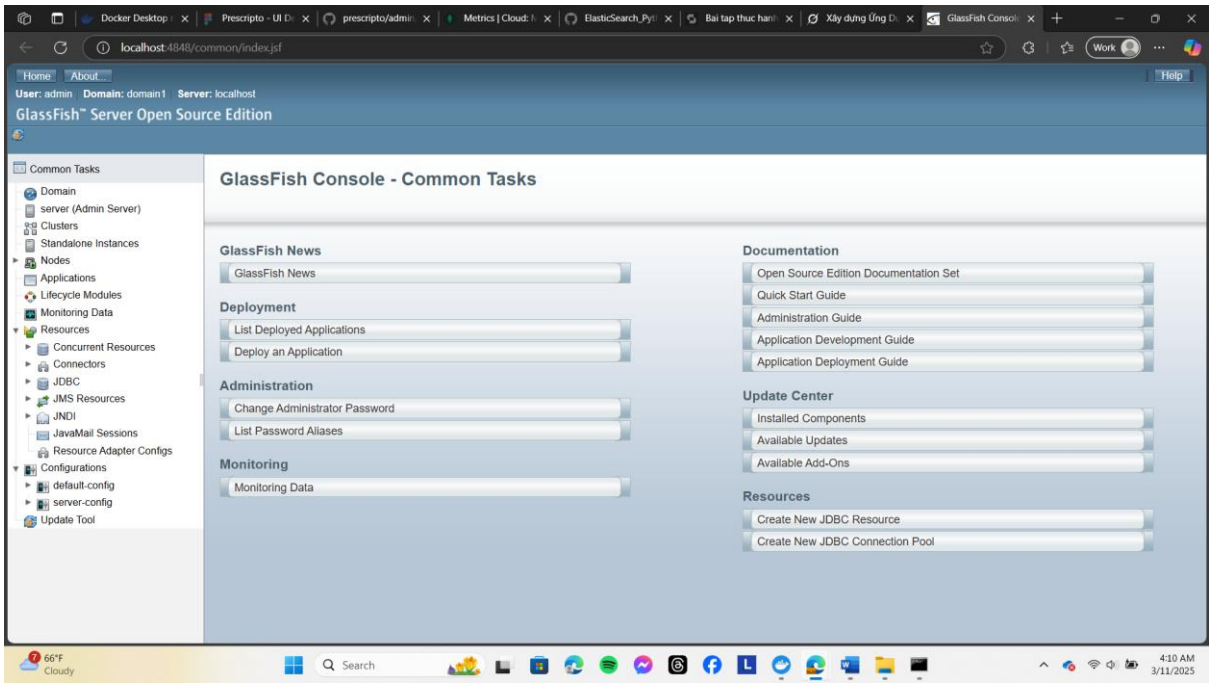




```
C:\Users\Lenovo\Downloads\it4611\microservices-demo>kubectl delete service apache catalog customer order
service "apache" deleted
service "catalog" deleted
service "customer" deleted
service "order" deleted

C:\Users\Lenovo\Downloads\it4611\microservices-demo>kubectl delete deployments apache catalog customer order
deployment.apps "apache" deleted
deployment.apps "catalog" deleted
deployment.apps "customer" deleted
deployment.apps "order" deleted

C:\Users\Lenovo\Downloads\it4611\microservices-demo>minikube stop
* Stopping node "minikube"  ...
* Powering off "minikube" via SSH ...
```

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?

Khi các pods mới được tạo, nó sẽ có trạng thái mặc định là ContainerCreating, sau đó khoảng 1 phút, nó chuyển sang trạng thái Running.

## 4. Kiến trúc JMS và DDS:




```
Command create-jms-resource failed.
asadmin> create-jms-resource --restype javax.jms.TopicConnectionFactory
Enter the value for the jndi_name operand> myTopicConnectionFactory
Connector resource myTopicConnectionFactory created.
Command create-jms-resource executed successfully.
```

```
asadmin> create-jms-resource --restype javax.jms.Topic
Enter the value for the jndi_name operand> myTopic
Administered object myTopic created.
Command create-jms-resource executed successfully.
```

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

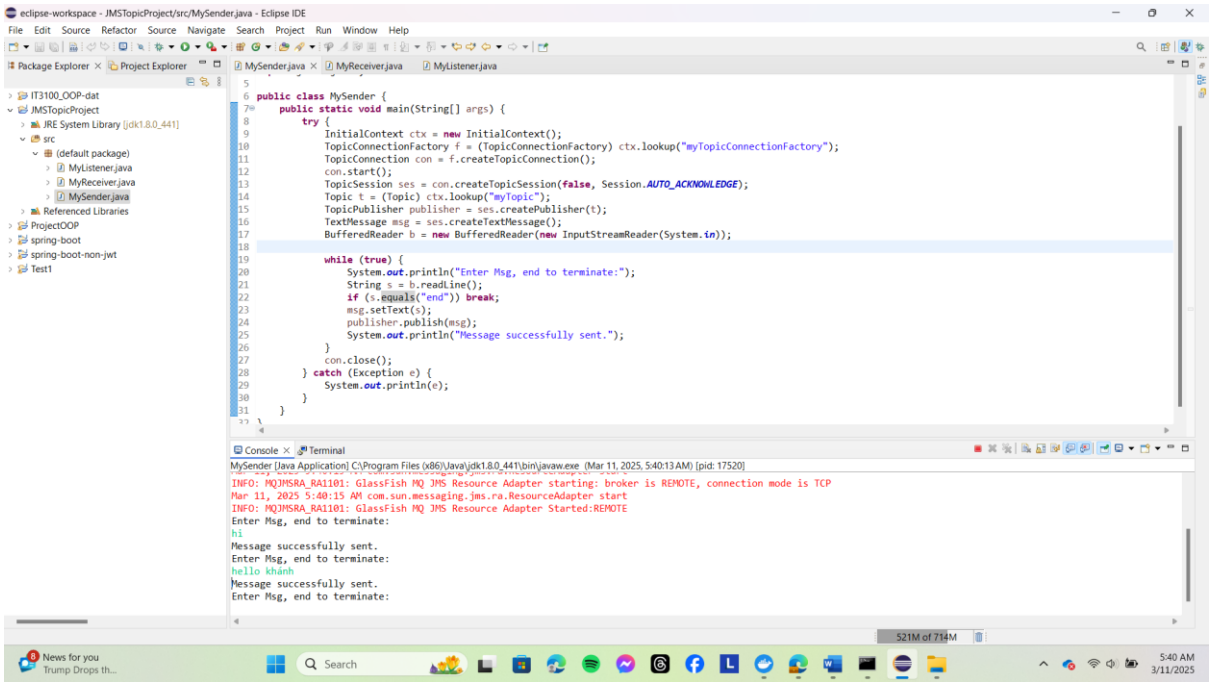
**GlassFish** là một **application server**, đóng vai trò trung gian giữa ứng dụng Java (thường là ứng dụng doanh nghiệp) và hệ thống phần cứng/mạng, cung cấp môi trường thực thi và các dịch vụ cần thiết. GlassFish cung cấp môi trường để triển khai và chạy ứng dụng JMS (Java Message Service) theo mô hình Publish/Subscribe, GlassFish cung cấp các dịch vụ quản lý tài nguyên, ngoài ra nó còn hỗ trợ giao tiếp giữa các thành phần trong ứng dụng, cấu hình dễ dàng bằng giao diện đồ họa

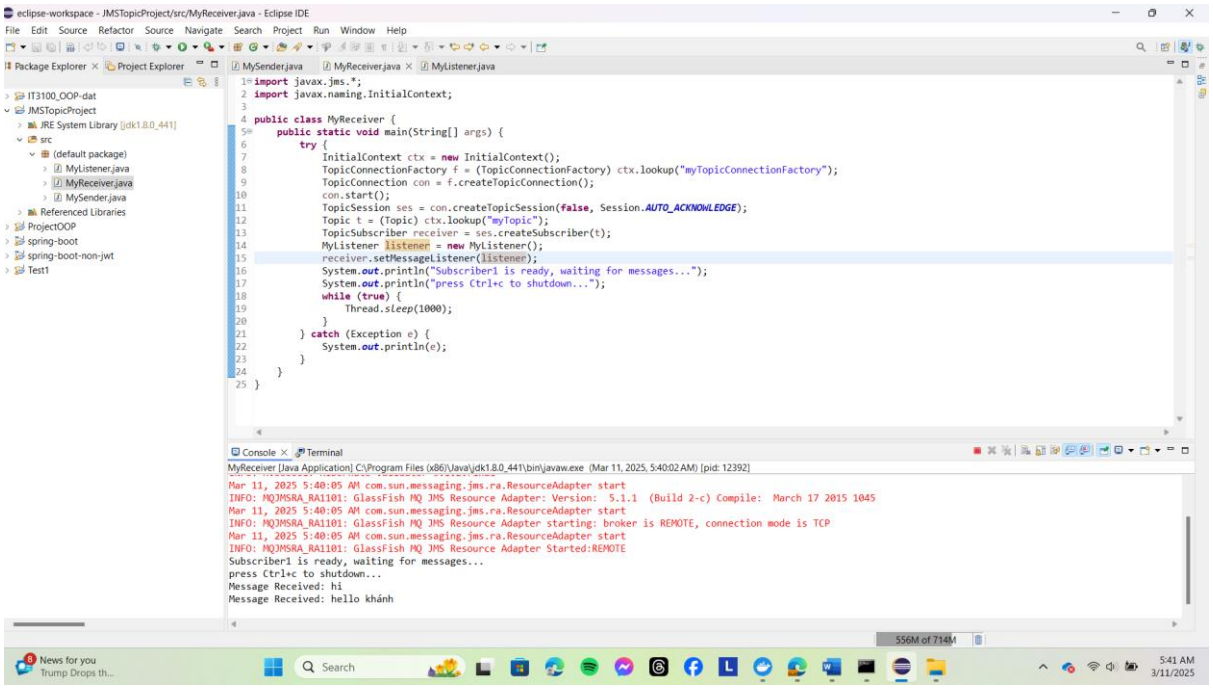
Câu hỏi 2: Tại sao lại phải tạo 2 JNDI như trên?

Cần tạo 2 JNDI là myTopicConnectionFactory và myTopic vì chúng đảm nhận hai vai trò khác nhau nhưng đều thiết yếu để ứng dụng có thể gửi và nhận tin nhắn

myTopicConnectionFactory: Là một **Connection Factory**, giúp ứng dụng của bạn thiết lập kết nối đến JMS provider (GlassFish). Nó chứa các thông tin cấu hình cần thiết để giao tiếp với hệ thống nhắn tin.

myTopic: Là một **Topic**, tức là điểm đến của tin nhắn trong mô hình Publish/Subscribe. Publisher gửi tin nhắn đến Topic này, và các Subscriber đăng ký với Topic sẽ nhận được tin nhắn.




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 chuyền và nhận thông điệp của Sender và Receiver.

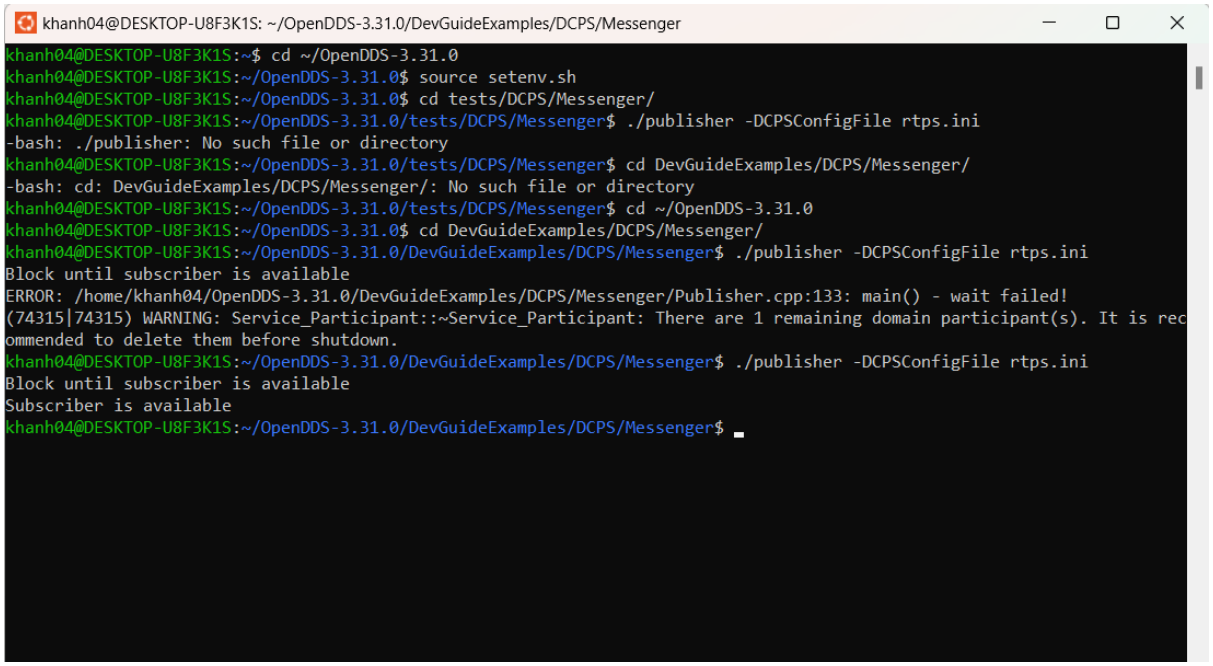
- **Sender (MySender.java)**: Đóng vai trò là **Event Producer**. Nó gửi các tin nhắn (sự kiện) đến một Topic.
- **Receiver (MyReceiver.java)**: Đóng vai trò là **Event Consumer**. Nó đăng ký lắng nghe các tin nhắn từ Topic và xử lý chúng thông qua MyListener.
- **Topic (myTopic)**: Đóng vai trò là **Event Channel**. Đây là kênh trung gian mà Sender gửi tin nhắn đến và Receiver nhận tin nhắn từ đó.

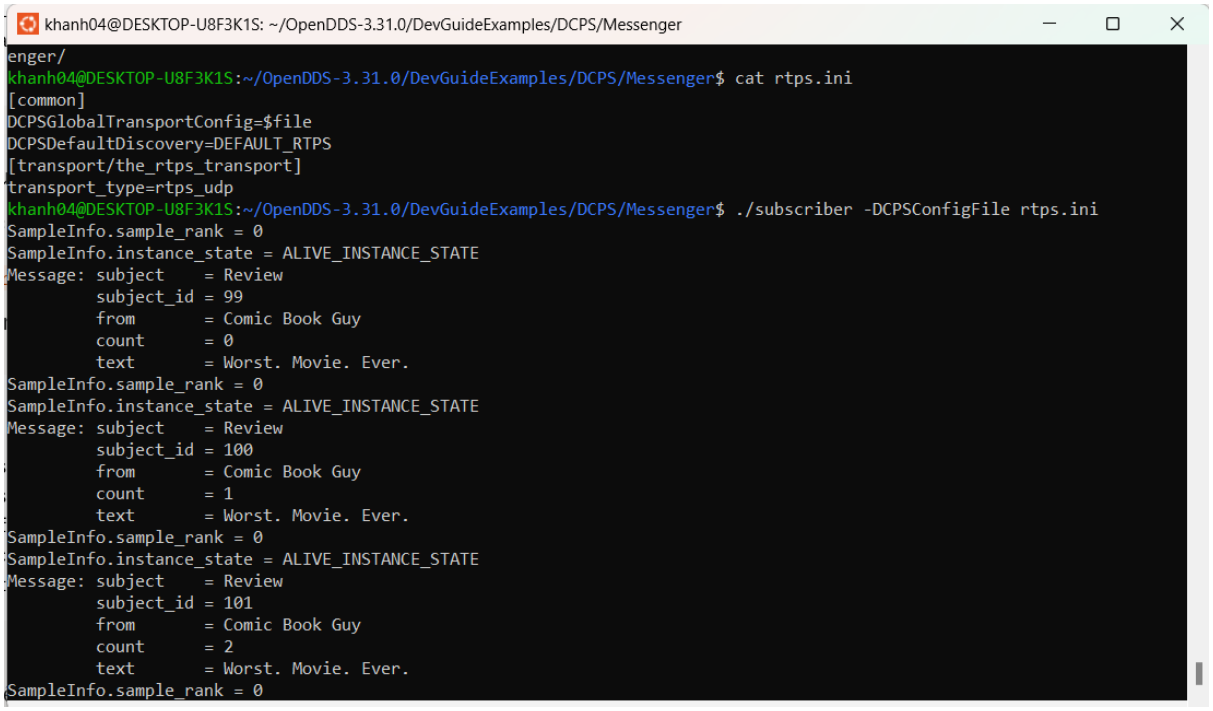
**Cơ chế chuyền (Sender):**

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 chơ chế roolback 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.

| Tiêu chí | JMS | DDS |
|---|---|---|
| **Tên đầy đủ** | Java Message Service (Dịch vụ Tin nhắn Java) | Data Distribution Service (Dịch vụ Phân phối Dữ liệu) |
| **Loại** | API Middleware Hướng Tin nhắn Java | Middleware Mạng |
| **Mục đích** | Dùng để gửi tin nhắn giữa hai hoặc nhiều client | Phân phối dữ liệu, sự kiện và lệnh giữa các node |
| **Kiến trúc** | Tập trung (Centralized) | Phi tập trung (Decentralized) |
| **Mô hình giao tiếp** | Cho phép giao tiếp giữa các thành phần của ứng dụng phân tán | Triển khai mô hình publish/subscribe để gửi và nhận dữ liệu, sự kiện, lệnh |
| **Cách hoạt động** | Sử dụng hàng đợi (Queue) hoặc chủ đề (Topic) để gửi/nhận tin nhắn | Tự động xử lý tất cả các khía cạnh của việc phân phối tin nhắn giữa các node |
| **Triển khai phổ biến** | ActiveMQ, GlassFish, RabbitMQ | OpenDDS, RTI Connext, Vortex OpenSplice |
| **Tính năng nổi bật** | Được sử dụng phổ biến trong các ứng dụng doanh nghiệp | Tự động xử lý chuyển đổi publisher dự phòng nếu publisher chính gặp sự cố |