BÀI THỰC HÀNH
MÔN HỌC: HỆ PHÂN TÁN
CHƯƠNG 2: Kiến trúc

1. Kiến trúc Microservices

1.1. Nội dung

Ở bài thực hành này chúng ta sẽ xây dựng một kiến trúc microservices đơn giản bằng cách sử dụng Kubernetes, một nền tảng mở để quản lý những dịch vụ dạng container. Chúng ta sẽ sử dụng Docker để tạo ra các containers.

1.2. Yêu cầu

1.2.1. Lý thuyết

- Microservices
- Kube fundamentals: Pods, Services, Deployments et al.
- Docker

1.2.2. Phần cứng

- Laptop/PC on Windows

1.2.3. Phần mềm

- VirtualBox
- Docker
- The kubernetes command line tool *kubectl*
- The minikube binary
- Git bash

1.3. Các bước thực hành

Cài đặt

- Cài đặt VirtualBox: https://www.virtualbox.org/wiki/Downloads

- Để cài đặt *kubectl* trên Windows, chúng ta sẽ cần phải cài đặt *Chocolatey* trước: https://chocolatey.org/docs/installation#installing-chocolatey

- Bây giờ hãy cài đặt công cụ kubernetes với lệnh sau:

```
>choco install kubernetes-cli
```

Kiểm tra lại xem đã cài được chưa bằng lệnh sau:

```
>kubectl version
```

- Bây giờ hãy cài *minikube-windows-amd64.exe*: https://github.com/kubernetes/minikube/releases

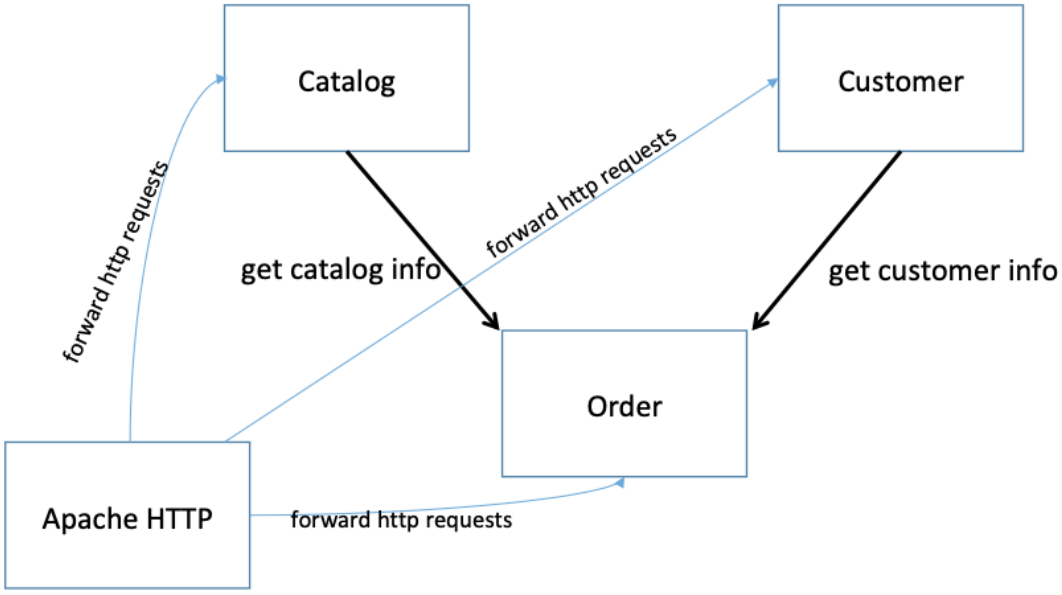
Hãy đặt tên lại cho file vừa tải là *minikube.exe* và thêm nó vào *Path*. (Nếu chưa biết thêm biến môi trường vào Path thế nào thì xem hướng dẫn ở link sau: https://www.java.com/en/download/help/path.xml)

- Đối với bạn nào dùng Windows thì sẽ bị vấn đề khi cài song song Docker và VirtualBox là tại vì Docker trên Windows thì cần Hyper-V, còn VirtualBox thì lại không hoạt động được với Hyper-V. Vì vậy, giải pháp đó là thay vì sử dụng Docker thông thường thì hãy sử dụng Docker Toolbox: https://docs.docker.com/toolbox/overview/

Sử dụng kiến trúc microservices để xây dựng ứng dụng web

Bây giờ chúng ta sẽ phát triển một ứng dụng web đơn giản bằng việc sử dụng kiến trúc microservices. Cụ thể chúng ta sẽ xây dựng 4 microservices sau:

1. Dịch vụ *Apache HTTP*: Apache HTTP được sử dụng để cung ứng dịch vụ trang web ở cổng 8080. Nó sẽ làm nhiệm vụ là chuyển tiếp các HTTP requests đến cho 3 microservices khác. Lý do sử dụng dịch vụ này là vì thật sự không cần thiết nếu chúng ta xây dựng cho mỗi microservices một cổng riêng trên host của minikube, thay vì thế chúng ta sử dụng chung 1 điểm vào cho cả 3 dịch vụ (xem hình dưới). Apache HTTP được cấu hình như một reversy proxy cho hệ thống này. Cơ chế cân bằng tải thì sẽ do Kubernetes đảm nhiệm.
2. Dịch vụ *Order*: Dịch vụ này để xử lý các đơn đặt hàng của người dùng. Dịch vụ này kết nối với dịch vụ Customer và Catalog để lấy thông tin.
3. Dịch vụ *Customer*: dịch vụ này có nhiệm vụ quản lý dữ liệu khách hàng.
4. Dịch vụ *Catalog*: Dịch vụ này quản lý các danh mục hàng hóa.



Mã nguồn có thể xem ở link sau: https://github.com/anhth318/microservices-demo

Sử dụng Git Bash, vào thư mục mà bạn muốn đặt thư mục tải về:

```
>git clone https://github.com/anhth318/microservices-demo.git
```

Vào thư mục *microservices-demo*, chạy lệnh sau:
`./mvnw clean package -Dmaven.test.skip=true`
hoặc nếu dùng Windows thì chạy lệnh:
`mvnw.cmd clean package -Dmaven.test.skip=true`

Những lệnh trên dùng để build/re-build 3 dịch vụ trên.
Sau đó bạn cần phải lên tạo tài khoản trên Docker Hub: https://hub.docker.com/

Chạy docker toolbox ở máy của bạn bằng cách click đúp chuột vào biểu tượng *Docker Quickstart Terminal*.

Bây giờ, bạn hãy đưa 4 dịch vụ ở trên vào dạng ảnh (images) của docker. Sau đó bạn sẽ tải lên máy chủ Docker Hub. Nhưng trước tiên, bạn cần đăng nhập vào Docker Hub ở dòng lệnh bằng lệnh sau:

```
>docker login
```

Gõ lại username và mật khẩu tài khoản của bạn đã tạo trên DockerHub.

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 tự như trên với 3 dịch vụ còn lại.

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?

Minikube là một công cụ để triển khai Kubernetes một cách đơn giản. Nó sẽ tạo một máy ảo và triển khai một cluster đơn giản có chứa một nút thực thi duy nhất.

Minikube có thể chạy trên nhiều nền tảng hệ điều hành bao gồm Linux, MacOS, Windows.
Bây giờ, hãy sử dụng Minikube để tạo một cluster bao gồm 1 nút thực thi Kubernetes (máy ảo) duy nhất, nó sẽ chạy các ứng dụng/dịch vụ (dưới dạng các pods) và điều khiển bởi *Kubernetes master*.

```
>minikube start
```

Bạn triển khai các dịch vụ bằng cách sử dụng các ảnh docker mà bạn đã tải nó lên DockerHub. Bạn hãy mở file microservices.yaml và thay thế tất cả những chỗ nào xuất hiện tên tài khoản docker hub là anhth bằng tên tài khoản của bạn, cụ thể là những dòng bắt đầu bằng từ *image*, như dòng sau:

```
    - image: docker.io/anhth/microservice-kubernetes-demo-apache:latest
```

Tiếp đó, chạy lệnh sau để triển khai các ảnh trên Docker Hub đó:

```
>kubectl apply -f microservices.yaml
```

Câu lệnh trên tạo ra các Pods. Một Pods có thể chứa một hay nhiều Docker Containers. Trong ví dụ này mỗi Pod chỉ chứa một Docker container. Các dịch vụ cũng đã được tạo. Các dịch vụ có duy nhất một địa chỉ IP và một bản ghi DNS. Các dịch vụ có thể sử dụng các Pods để cân bằng tải.

Sử dụng lệnh sau để cho hiện tất cả các thông tin của Kubernetes master của bạn:

```
>kubectl get all
```

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?

Để xem chi tiết hơn hãy gõ lệnh `kubectl describe services`. Lệnh nãy cũng chạy được cho các pods (`kubectl describe pods`) và các deployments (`kubectl describe deployments`).

Bạn cũng có thể xem logs của một pod (thay thế bằng ID của pod của bạn):

```
>kubectl logs catalog-269679894-60dr0
```

Bạn thậm chí có thể mở 1 shell của pod của bạn:

```
>kubectl exec catalog-269679894-60dr0  -it /bin/sh
```

Hãy chờ đến khi trạng thái của tất cả các pods của bạn chuyển thành "Running", lúc đó là lúc bạn đã sẵn sàng để chạy ứng dụng của mình. Hãy gõ lệnh sau:

```
>minikube service apache
```

Với lệnh trên thì bạn có thể mở được trang web thông qua Apache httpd server ở trên trình duyệt web của bạn.
Bây giờ hãy chạy ứng dụng.
Ấn vào "Customer", bạn sẽ thấy tất cả các tên của customer. Sau đó ấn vào "Add customer", bạn sẽ tạo thêm được khách hàng mới.
Trở về trang chủ Home, làm tương tự với Catalog.
Trở về trang chủ Home, ấn vào Order, và hãy thử thêm vào vài yêu cầu mua hàng.

Sau khi kết thúc, đừng quên xóa toàn bộ các dịch vụ và các deployments:

```
>kubectl delete service apache catalog customer order
>kubectl delete deployments apache catalog customer order
```

và tắt cluster:

```
>minikube stop
```

2. Kiến trúc JMS và DDS

2.1. Nội dung

Ở bài thực hành này chúng ta sẽ làm về 2 mô hình kiến trúc JMS và DDS. Mục đích của bài thực hành sẽ giúp các bạn nắm vững và hiểu hơn lý thuyết của 2 khái niệm này.

2.2. Điều kiện

2.2.1. Kiến thức

Sử dụng thành thạo hđh Unix
Các kiến thức về mô hình Publish/Subscribe đã học trên lớp lý thuyết.
Kỹ năng lập trình Java và C++

2.2.2. Phần cứng

Máy tính cài hđh Ubuntu

2.2.3. Phần mềm

Máy phải có cài JDK 8.0 trở lên.

2.3. Các bước thực hành

2.3.1. JMS

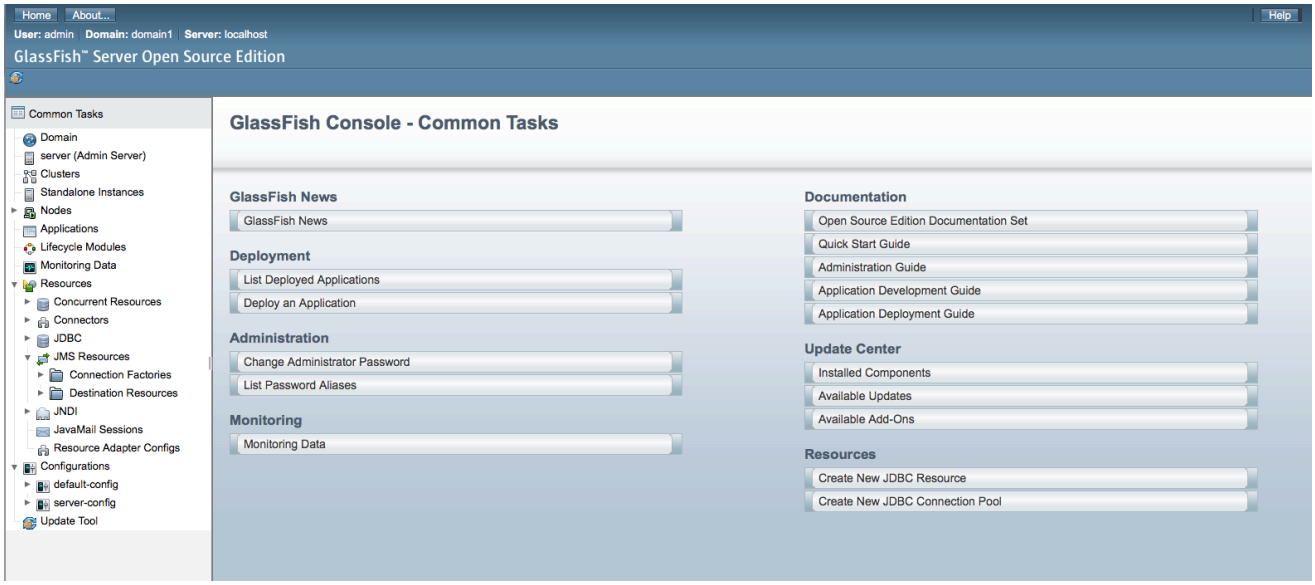
Chúng ta biết JMS hỗ trợ 2 mô hình là Point-to-Point và Publish/Subscribe. Ở bài thực hành này chúng ta sẽ tập trung vào mô hình P/S.
Đầu tiên chúng ta phải cài đặt một application server. Chúng ta sẽ chọn một server nguồn mở là glassfish.

Cài đặt server glassfish:

- Download về tại địa chỉ http://download.java.net/glassfish/4.1.1/release/glassfish-4.1.1.zip
- Giải nén ra thư mục glassfish4.
- Khởi động glassfish bằng lệnh

```
glassfish4/bin/asadmin start-domain
```

Lúc này server glassfish đã chạy một domain là domain1. Ngoài ra glassfish còn hỗ trợ giao diện web trên cổng 4848. Các bạn mở trình duyệt và vào địa chỉ http://localhost:4848
Các bạn sẽ thấy giao diện web như hình dưới đây. Hãy chú ý vào phần JMS Resources, đó là phần chúng ta phải tạo Connection Factories và Destination resources.



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

Tạo 2 JNDI

Bước tiếp theo chúng ta phải tạo 2 JNDI là *myTopicConnectionFactory* và *myTopic*.

Thông thường có thể làm bằng giao diện web, tuy nhiên làm theo cách này rất hay bị lỗi. Vì vậy khuyến khích tạo 2 JNDI bằng cách gõ lệnh. Chú ý, các lệnh được gõ sau khi vào thư mục `glassfish4/bin/` và gõ lệnh `./asadmin`
Tạo resource Connection Factory

```
asadmin>create-jms-resource --restype
javax.jms.TopicConnectionFactory
```

Sau đó bạn sẽ được hỏi tên của jndi, gõ là *myTopicConnectionFactory*

```
Enter the value for the jndi_name
operand>myTopicConnectionFactory
```

Tạo resource Destination:

```
asadmin> create-jms-resource --restype javax.jms.Topic
```

Tương tự, khi được hỏi jndi name thì gõ vào là *myTopic*

Vào giao diện web và kiểm tra xem 2 jndi đã được tạo hay chưa.

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

Tạo chương trình Sender và Receiver.
Bước này sẽ là lập trình bằng ngôn ngữ Java, khuyến khích chạy chương trình bằng IDE Eclipse.
Mở Eclipse, tạo 1 project chung, đặt tên là JMSTopicProject.
Chú ý, cần phải add thêm các thư viện sau vào project:

- *gf-client.jar*: lấy trong thư mục `glassfish4/glassfish/lib`
- *javax.jms.jar*: có thể tải về từ Internet.

Tạo 3 file đại diện cho 3 lớp sau: MySender.java, MyReceiver.java, và MyListener.java
Các đoạn mã nguồn cho 3 file trên như sau:

File: MySender.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import javax.naming.*;
import javax.jms.*;

public class MySender {
   public static void main(String[] args) {
      try
      {   //Create and start connection
         InitialContext ctx=new InitialContext();
         TopicConnectionFactory f=(TopicConnectionFactory)ctx.lookup("myTopic
ConnectionFactory");
         TopicConnection con=f.createTopicConnection();
         con.start();
         //2) create queue session
         TopicSession ses=con.createTopicSession(false, Session.AUTO_ACKNOW
LEDGE);
         //3) get the Topic object
         Topic t=(Topic)ctx.lookup("myTopic");
         //4)create TopicPublisher object
```

```
            TopicPublisher publisher=ses.createPublisher(t);
            //5) create TextMessage object
            TextMessage msg=ses.createTextMessage();

            //6) write message
            BufferedReader b=new BufferedReader(new InputStreamReader(Syste
m.in));
            while(true)
            {
                System.out.println("Enter Msg, end to terminate:");
                String s=b.readLine();
                if (s.equals("end"))
                    break;
                msg.setText(s);
                //7) send message
                publisher.publish(msg);
                System.out.println("Message successfully sent.");
            }
            //8) connection close
            con.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```

File: MyReceiver.java

```
import javax.jms.*;
import javax.naming.InitialContext;

public class MyReceiver {
    public static void main(String[] args) {
        try {
            //1) Create and start connection
            InitialContext ctx=new InitialContext();
            TopicConnectionFactory f=(TopicConnectionFactory)ctx.lookup("myTopic
ConnectionFactory");
            TopicConnection con=f.createTopicConnection();
            con.start();
            //2) create topic session
            TopicSession ses=con.createTopicSession(false, Session.AUTO_ACKNOW
LEDGE);
            //3) get the Topic object
            Topic t=(Topic)ctx.lookup("myTopic");
            //4)create TopicSubscriber
            TopicSubscriber receiver=ses.createSubscriber(t);

            //5) create listener object
            MyListener listener=new MyListener();

            //6) register the listener object with subscriber
            receiver.setMessageListener(listener);

            System.out.println("Subscriber1 is ready, waiting for messages...");
            System.out.println("press Ctrl+c to shutdown...");
            while(true){
                Thread.sleep(1000);
```

```
        }
      }catch(Exception e){System.out.println(e);}
   }

}
```

File: MyListener.java

```
import javax.jms.*;
public class MyListener implements MessageListener {

   public void onMessage(Message m) {
      try{
      TextMessage msg=(TextMessage)m;

      System.out.println("following message is received:"+msg.getText());
      }catch(JMSException e){System.out.println(e);}
   }
}
```

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.

2.3.2. DDS

Ở phần này chúng ta sẽ thực hành để tìm hiểu cách vận hành của mô hình DDS. Cụ thể, chúng ta sẽ cài đặt chương trình nguồn mở OpenDDS.

Cài đặt OpenDDS

Máy của bạn trước khi cài OpenDDS phải cài 3 chương trình sau:

- C++ compiler
- GNU Make
- Perl

Download file .tar.gz phiên bản mới nhất từ link sau:
http://download.ociweb.com/OpenDDS/

Giải nén file bằng lệnh

```
tar -xvzf OpenDDS-3.8.tar.gz
```

Vào thư mục vừa giải nén, gõ 2 lệnh sau để cài đặt:

```
./configure
make
```

Gõ lệnh sau để cài đặt các thông số đường dẫn môi trường:

```
source setenv.sh
```

Sau đó vào thư mục

```
cd OpenDDS-3.8/tests/DCPS/Messenger/
```

Tạo và soạn nội dung file rtps.ini như sau:
[common]
DCPSGlobalTransportConfig=$file
DCPSDefaultDiscovery=DEFAULT_RTPS
[transport/the_rtps_transport]
transport_type=rtps_udp

Sau đó khởi động subscriber:
```
./subscriber -DCPSConfigFile rtps.ini
```

Mở một tag khác để chạy publisher:
(chú ý, vẫn phải chạy lệnh source setenv.sh ở tab mới)
Sau đó vào thư mục như trên và chạy publisher:
```
./publisher -DCPSConfigFile rtps.ini
```

Câu hỏi 4: So sánh JMS và DDS.

2.4. Kết luận