# LABWORK
# COURSE: DISTRIBUTED SYSTEMS
# CHAPTER 2: ARCHITECTURES

## 1. Microservices

### 1.1. Contents

We now try to practice constructing a basic microservices architecture in using Kubernetes, a portable, extensible open-source platform for managing containerized workloads and services. We use also Docker to create containers.

### 1.2. Requirements

#### 1.2.1. Theory

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

#### 1.2.2. Hardwares

- Laptop/PC on Windows

#### 1.2.3. Softwares

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

### 1.3. PRACTICAL STEPS

**Installation**

- Install VirtualBox: https://www.virtualbox.org/wiki/Downloads
- In order to install tool kubectl on Windows, we have to install *Chocolatey* before: https://chocolatey.org/docs/installation#installing-chocolatey
- Now, install kubernetes tool with this command:

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

Make sure the installation is successfully with this command:

```
>kubectl version
```

- Install the file *minikube-windows-amd64.exe*:
https://github.com/kubernetes/minikube/releases

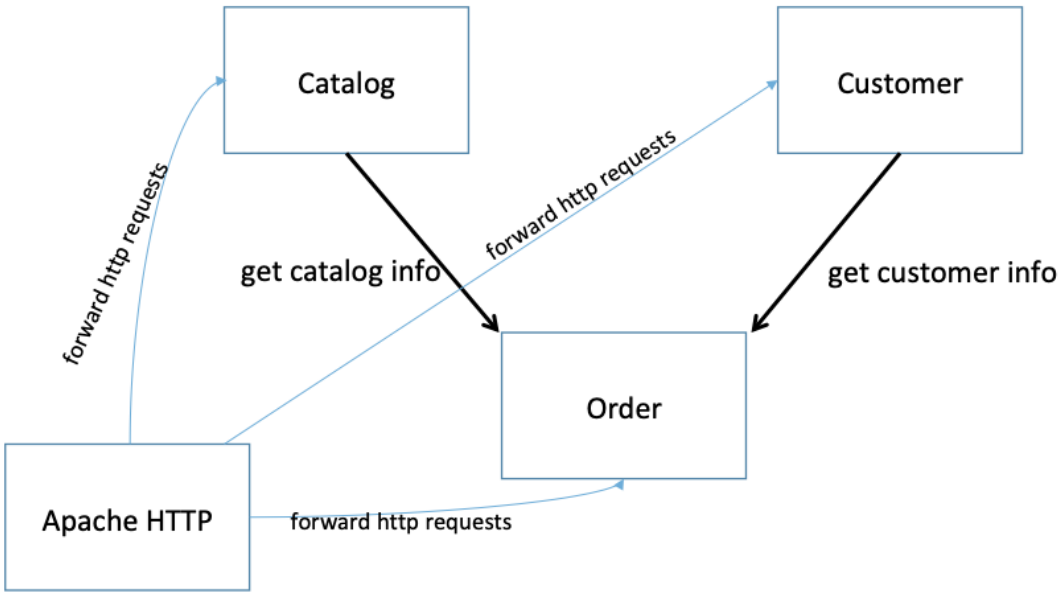
You have to rename it to *minikube.exe* and add it to your path. (Here is how to edit your path variable: https://www.java.com/en/download/help/path.xml)

- To install Docker, there is a problem on Windows here. We know that Docker for Windows requires Hyper-V to work, however VirtualBox does not work with Hyper-V enabled (?!?). So, there is a solution here: Use the Docker Toolbox (instead of official Docker for Windows): https://docs.docker.com/toolbox/overview/

**Building web application in using microservices architecture**

Now we will develop a simple web application in using microservices architecture. Concretely, we construct 4 services as follows:

1. Service *Apache HTTP*: Apache HTTP is used to provide the web page of the demo at port 8080. It also forwards HTTP requests to the 3 other microservices. This is not really necessary as each service has its own port on the Minikube host but it provides a single point of entry for the whole system. Apache HTTP is configured as a reverse proxy for this. Load balancing is left to Kubernetes.
2. Service *Order*: to process orders. This service is connected to customer service and catalog service to get information.
3. Service *Customer*: to handle customer data.
4. Service *Catalog*: to handle the items in the catalog.




The source code is available at the link:
https://github.com/anhth318/microservices-demo

Use Git Bash, go to the folder where you want to place the folder of source code, type:

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

Go into the folder *microservices-demo*, then run:
`./mvnw clean package -Dmaven.test.skip=true`
or on the Windows:
`mvnw.cmd clean package -Dmaven.test.skip=true`
The command above is to build/re-build the 3 mentioned services.
Next, you create an account at the public Docker Hub: https://hub.docker.com/

Run the docker toolbox in your machine in double clicking on the icon *Docker Quickstart Terminal*.

Now, you will put our 4 services into docker images. After, you will upload it to the public Docker Hub server:
First, you have to log in to DockerHub. Open a terminal window:

```
>docker login
```

After, you provide the username and password of your DockerHub account.

Now, go to the working folder:

For the service *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
```

You do the same thing with 3 other services.

| Question 1: What are the commands did you use? |
|---|

| Question 2: Open the website Docker Hub and login with your account. What's new in your docker hub repository? |
|---|

FYI, Minikube is a lightweight Kubernetes implementation that creates a VM on your local machine and deploys a simple cluster containing only one node. Minikube is available for Linux, macOS, and Windows systems.

Now, it's time to use Minikube tool to create a cluster with only one *Kubernetes node* (a virtual machine) that will hosts all your running applications/services (under type of pods) handled by a *Kubernetes master.*

```
>minikube start
```

You deploy the services in using uploaded docker images in your Docker Hub.
You open the file *microservices.yaml* and replace all my docker hub account (anhth) with your own account of every line that begins with the word *image*, like the one below:

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

Save the file and close it. Run the command below to deploy the images in your Docker Hub:

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

The command above creates Pods. Pods might contain one or many Docker containers. In this case each Pod contains just one Docker container. Also services are created. Services have a cluster wide unique IP address and a DNS entry. Service can use many Pods to do load balancing.

Use this command to show all the information of your kubernetes cluster:

```
>kubectl get all
```

Question 3: What is the status of these created pods? Now, wait few minutes and re-type this command, what is the new status of these pods?

For more details, run `kubectl describe services`. This also works for pods (`kubectl describe pods`) and deployments (`kubectl describe deployments`).

You can see the logs of a pod (replace your pod ID):

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

You can even open a shell in a pod:

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

You wait until all the status of the pods change to "Running", that means it's ready to run your application. Type the command below:

```
>minikube service apache
```

It will open the web page of the Apache httpd server in the web browser.
Now, enjoy your running application.

Click to the link “Customer”, you will see all the customers’ names. After, click “Add Customer” to add new customer.
Return to the Home page, do the same thing with Catalog and add new items.
Return to the Home page, click on the link “Order”, and try to add some orders.

In the end, don’t forget to remove all the services and deployments:

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

and stop the cluster:

```
>minikube stop
```

# 2. JMS and DDS architectures

## 2.1. Contents

In this labwork’s section, we'll construct 2 architectures based on JMS and DDS. The goal is to help you understand the theory of event-based architecture.

## 2.2. Requirements

### 2.2.1. Theory

- Handling Unix OS
- Base of event-based architecture, the publish/subscribe model.
- Java and C++ language

### 2.2.2. Hardware

- PC with Ubuntu OS

### 2.2.3. Software

## 2.3. PRACTICAL STEPS

### 2.3.1. JMS

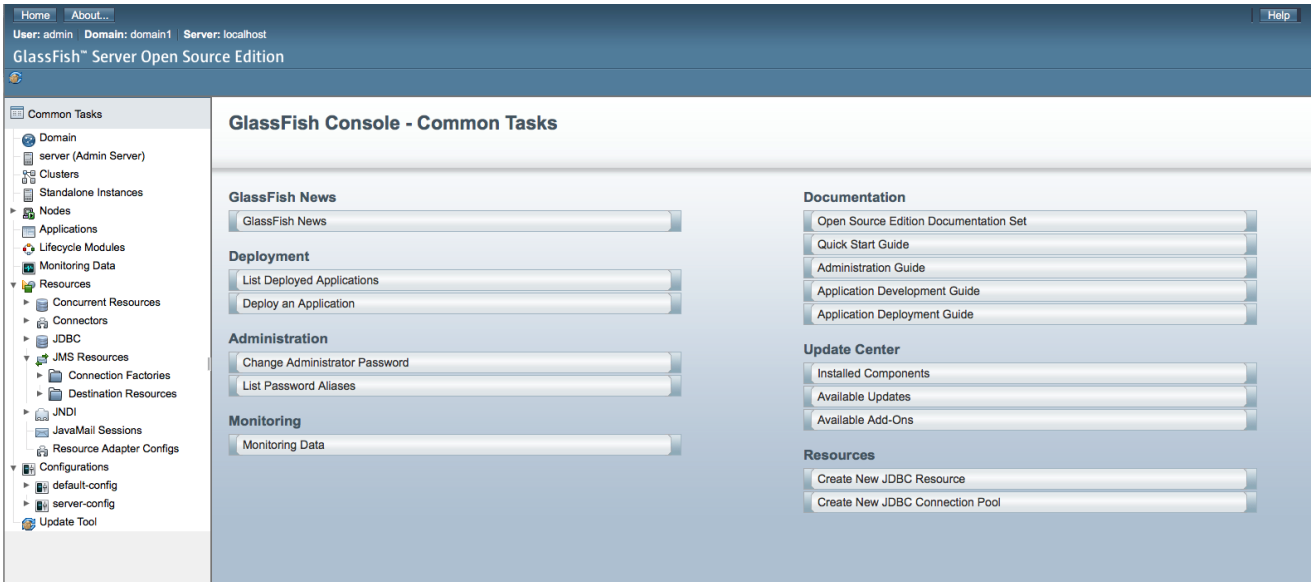
JMS (Java Message Service) is an API that provides the facility to create, send and read messages. It provides loosely coupled, reliable and asynchronous communication. JMS is also known as a messaging service.

**Install the server glassfish:**

- Download it at this link http://download.java.net/glassfish/4.1.1/release/glassfish-4.1.1.zip
- Unzip to the folder glassfish4.
- Start the glassfish:

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

Now, server glassfish has made running a domain called *domain1.* In addition, glassfish supports the web interface in the port 4848. Use your web browser and go to address:
*http://localhost:4848*
You'll see the web interface as below. Make attention to the *JMS Resources* part where we have to create *Connection Factories* and *Destination resources*.




**Question 4:** What is the role of application server glassfish?

**Creating 2 JNDI**

Now, we have to create 2 JNDI: *myTopicConnectionFactory* and *myTopic*.
Normally, you can use the web interface, however there will be some errors. So, you are recommended to create 2 JNDI in using command line.
Go to the folder *glassfish4/bin/* and type the command:

```
./asadmin
```

Create resource Connection Factory

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

Then, you will be asked the name of JNDI, type *myTopicConnectionFactory*

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

Create resource Destination:

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

In the same way, you type the jndi name: *myTopic*

Go to the web interface to check if these two JNDI have been created.

**Question 5:** Why do we need to create the 2 JNDI above?

**Construct the Sender and Receiver.**

In this step, you will use the Java language, you are recommended to use IDE Eclipse.

Create a new project and name it: *JMSTopicProject*.
Attention: you have to add the following libraries into the project:

- *gf-client.jar*: in the folder `glassfish4/glassfish/lib`
- *javax.jms.jar*: download it from the Internet.

Create three files representing 3 classes: *MySender.java, MyReceiver.java,* and *MyListener.java*
Here are the content of these files:

*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);}
   }
}
```

Congratulations! You have been done. Now it's time to test the Sender and the Receiver.

**Question 6:** Explain the message passing method of Sender and Receiver in basing on the theory of event-based architecture.

### 2.3.2. DDS

In this section, we will take a look at the DDS model. Concretely, we will install the open source software OpenDDS.

**Install OpenDDS**

You have to install the 3 following programs:

- C++ compiler
- GNU Make
- Perl

Download the file .tar.gz (the newest version) at this link:
http://download.ociweb.com/OpenDDS/

Decompress the file:

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

Go the decompressed folder and type the 2 following commands:

```
./configure
make
```

Type this command to configure the environment parameters:

```
source setenv.sh
```

Go to the folder:

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

Create and edit the content of the file *rtps.ini* as follows:

*[common]*
*DCPSGlobalTransportConfig=$file*
*DCPSDefaultDiscovery=DEFAULT_RTPS*
*[transport/the_rtps_transport]*
*transport_type=rtps_udp*

Start the subscriber:

```
./subscriber -DCPSConfigFile rtps.ini
```

Open a new tab to run publisher:
(attention: you have also to run the *source setenv.sh* command in the new tab)
Go to the same folder and run:

```
./publisher -DCPSConfigFile rtps.ini
```

| Question 7: Compare the JMS and DDS. |
| --- |