

## Lab 06 Application Modernization with Java EE Microservices and Liberty

### Contents

|     |  |    |
|-----|--|----|
| 6.1 | Introduction.....  | 1  |
| 6.2 | Microservices.....   | 2  |
| 6.3 | Let's get started.....   | 3  |
| 6.4 | Conclusion .....   | 32 |
|     | Appendix 1 – Servlet Filter .....                                | 33 |
|     | Appendix 2 – Image Service Implementation .....                  | 35 |
|     | Appendix: SkyTap Tips for labs.....                              | 37 |
| 6.5 | How to use Copy / Paste between local desktop and Skytap VM..... | 37 |

In this lab exercise, we start to refactor a monolith Java EE application by moving one of its functions into a microservice and modifying the application to access the microservice via REST.

Over time, this pattern of moving services from an application to a microservice will allow for the eventual sunset of the initial monolithic application, once all functions have been deployed as microservices.

### 6.1 Introduction

This is “**Lab 06 – App Modernization with Java EE Microservices and Liberty**” from an IBM Cloud Pak for Applications & App Modernization Proof of technology (PoT). The labs are not required to be executed in order. And, you may skip labs, and only perform the labs that suit your desired learning objectives.

#### The full set of labs in the PoT are:

- Lab01 - Getting started with Docker
- Lab02 - Explore RedHat OpenShift Container Platform
- Lab03 - Getting started with Kubernetes
- Lab04 – Liberty application deployment using Operators
- Lab05 – IBM Cloud Pak for Applications - App Modernization using Transformation Advisor
- Lab06 – App Modernization with Java EE Microservices and Liberty**
- Lab07 – Using Tekton pipelines for CI/CD of microservices to RedHat OpenShift Container Platform

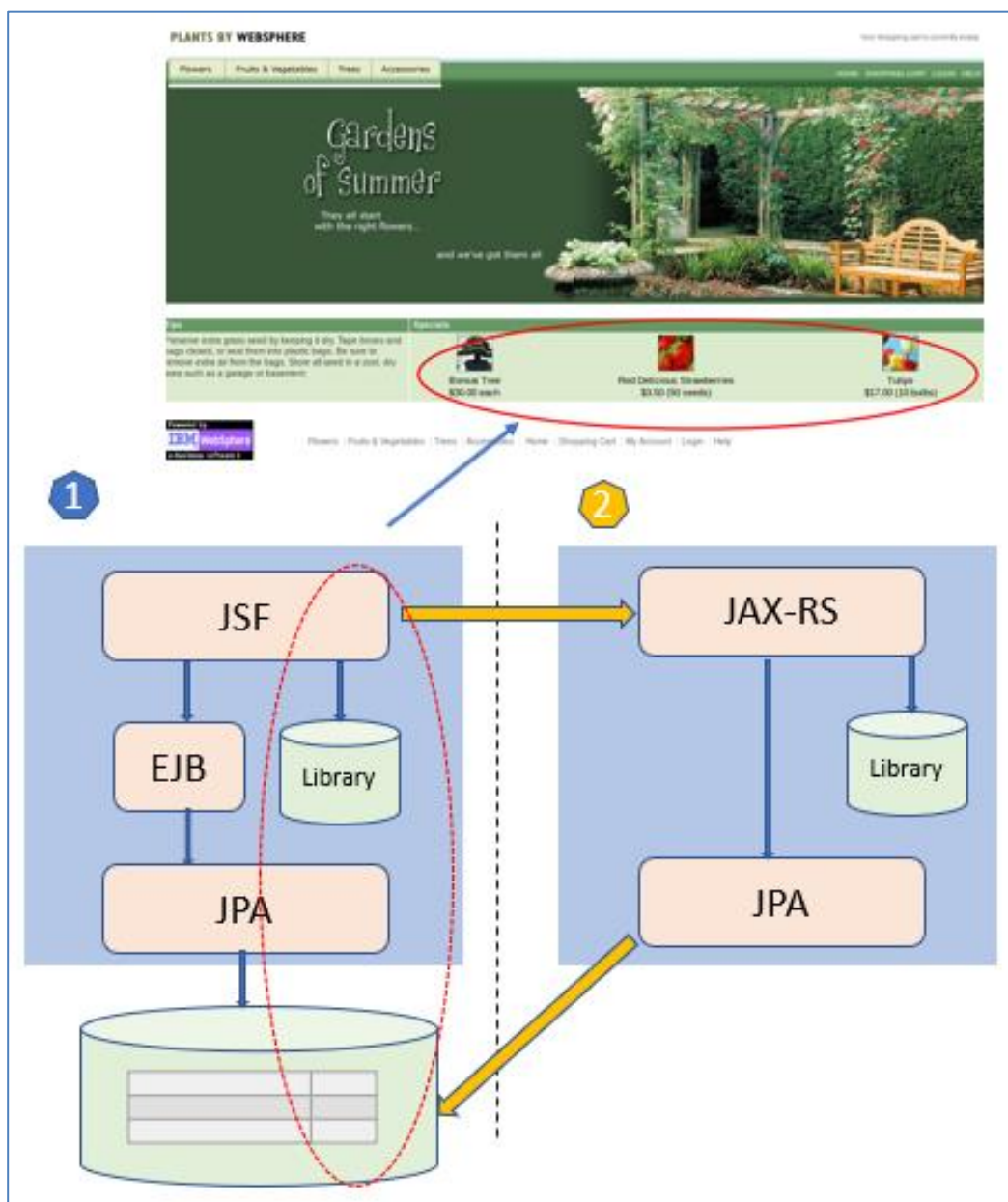
## 6.2 Microservices

The Plants By WebSphere application has been broken into two parts for this lab:

The original monolith ( **1** in the image below) has been modified with a Servlet Filter to intercept requests for images and redirect the requests to a new microservice application ( **2** in the image below).

The microservice exposes the image service via a JAX-RS interface which accesses the image library in the application EAR file and images in the database via JPA reusing the same code that was used in the monolith.

**Appendix 1** and **Appendix 2** contain the [Servlet Filter](#) and [Image Service](#) code

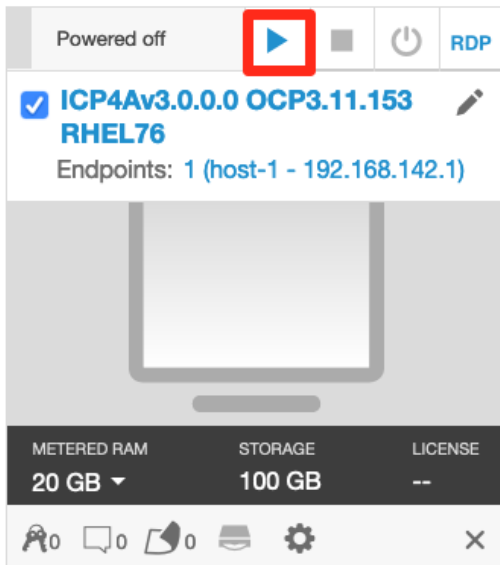


## 6.3 Let's get started

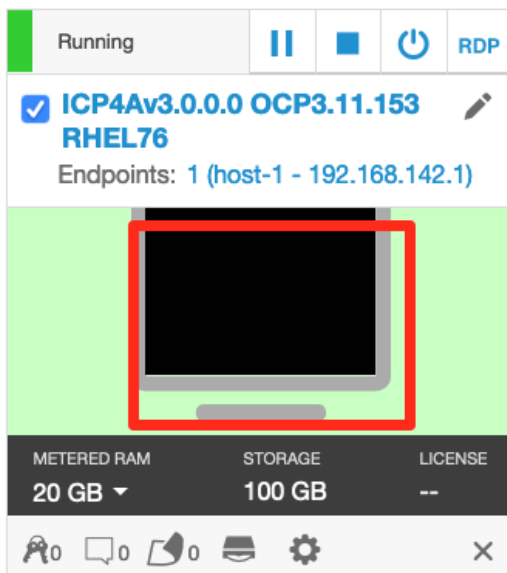
First, launch the lab environment and login to the VM.

On your laptop/workstation, locate the [ICP4Av3.0.0.0 OCP3.11.153 RHEL76](#) virtual machine

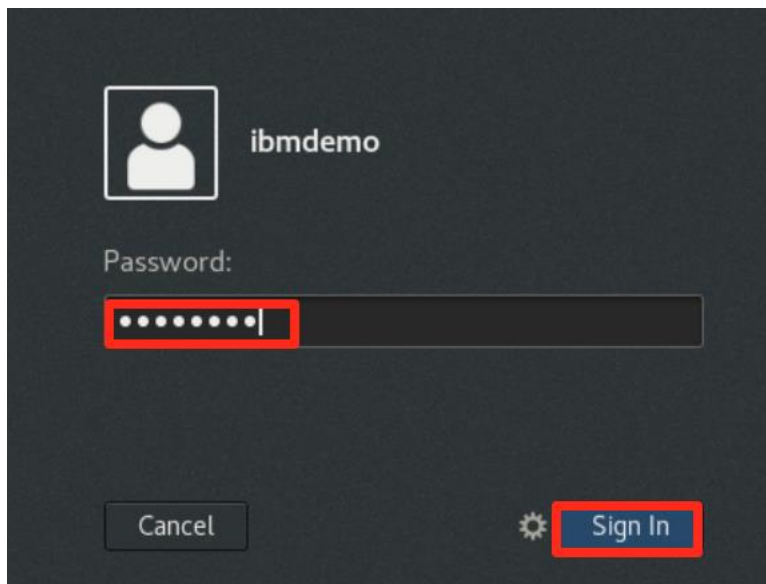
1. The VM should already be running. If not, Launch the Lab environment by clicking the **Run this VM** icon.



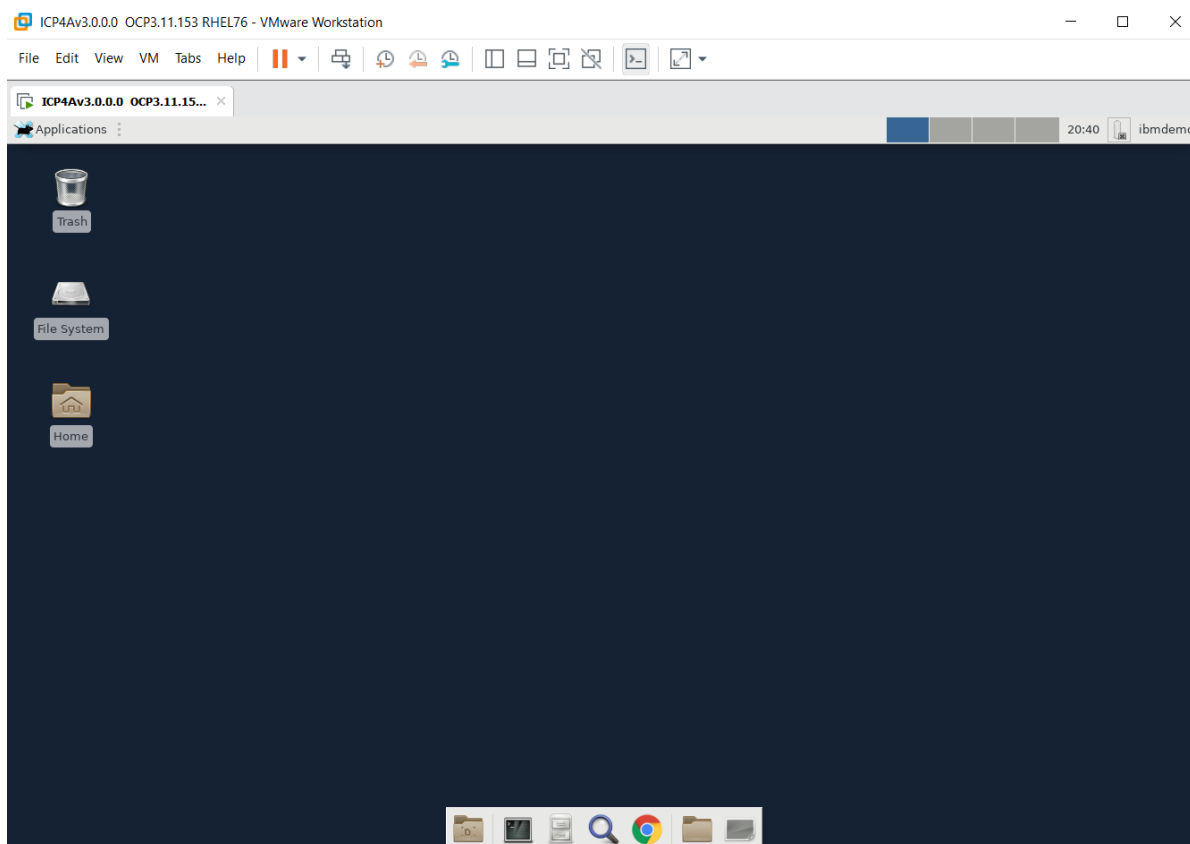
2. After the VM is running, click its icon to access the VM's desktop.



- \_\_3. After the VM machine powers on, log with the `ibmdemo` user using the password `password`.



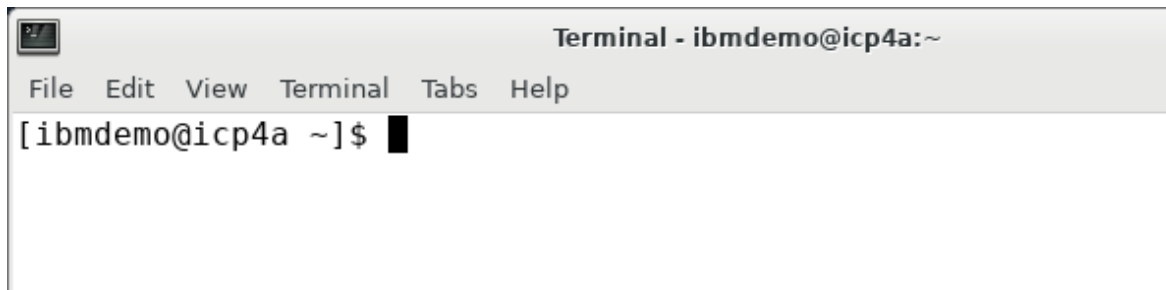
The `ICP4Av3.0.0.0 OCP3.11.153 RHEL76` virtual machine running and its Desktop is displayed in a web browser window.




- \_\_4. Click [Terminal](#) from the bottom of the desktop to open a command line terminal.



You'll be running in the terminal as the user [ibmdemo](#)



|   |  |
|---|--|
|  | <b>Note:</b> Please note that if needed root access can be obtained with <a href="#">sudo su -</a> |
|---|--|

- \_\_5. Type `oc login` to login to OpenShift. Use `ocpadmin` for the username and `ocpadmin` (note the "1", not "l") for the password

```
ibmdemo@icp4a]$ oc login
Authentication required for https://icp4a.pot.com:8443 (openshift)
Username: ocpadmin
Password:
Login successful.

You have access to the following projects and can switch between them with 'oc
project <projectname>':

* default
  istio-system
  kabanero
  knative-eventing
  knative-serving
  knative-sources
  kube-public
  kube-service-catalog
  kube-system
  lab3
  lab4
  lab5
  management-infra
  openshift
  openshift-console
  openshift-infra
  openshift-logging
  openshift-metrics-server
  openshift-monitoring
  openshift-node
  openshift-node-problem-detector
  openshift-pipelines
  openshift-sdn
  openshift-web-console
  operator-lifecycle-manager
  ta
Using project "default".
```

- \_\_6. Type `oc new-project lab6` which will create a new project named lab5, and switch your context to that project

```
[ibmdemo@icp4a ~]$ oc new-project lab6
Now using project "lab6" on server "https://icp4a.pot.com:8443".

You can add applications to this project with the 'new-app' command. For
example, try:

    oc new-app centos/ruby-25-centos7~https://github.com/sclorg/ruby-ex.git

to build a new example application in Ruby.
[ibmdemo@icp4a ~]$
```

- \_\_7. Enter `cd ~/student/lab6` then type `ls` to go to the lab directory and list the contents.

```
[ibmdemo@icp4a ~]$ cd ~/student/lab6

[ibmdemo@icp4a lab6]$ ls
pbwis  pbwv2
```

There are two directories;

- `pbwis` – Contains the deployment artifacts for the PlantsByWebSphere Image Service microservice
- `pbwv2` – Contains the modified PlantsByWebSphere application which incorporates a Servlet Filter to direct image access requests to the Image Service

- \_\_8. We're going to deploy the microservice first so type `cd pbwis` then type `ls`

```
[ibmdemo@icp4a lab6]$ cd pbwis
[ibmdemo@icp4a pbwis]$
[ibmdemo@icp4a pbwis]$ ls
01-builddocker.sh          04-deployApplication.sh  operator
02-createSecret.sh         99-cleanUpLab6.sh       src
03-createOperatorArtifacts.sh Dockerfile                target
```

\_\_9. Review the Dockerfile, using `cat Dockerfile` command.

```
[ibmdemo@icp4a pbwis]$ cat Dockerfile

#If on slow network comment the line below
FROM docker.io/ibmcom/websphere-liberty:19.0.0.6-kernel-ubi-min

# If on slow network uncomment the line below, image has Liberty server and features
#FROM lab6:latest

COPY src/main/liberty/config/server.xml /config/
COPY target/plantsbywebsphereimageservice.ear /config/apps/
COPY src/main/liberty/lib/DB2Libs/db2jcc4.jar /config/resources/DB2Libs/

USER root
RUN chown default:root -R /opt/ibm/wlp/usr/servers/
USER 1001

RUN configure.sh

[ibmdemo@icp4a pbwis]$
```

When executed, the Dockerfile performs the following actions as it constructs the Docker image:

- Pulls the spwcified websphere-liberty docker image from dockerhub
- Copies the Liberty serverxml configuration file to the /config folder
- Copies the PlantsByWebSphere EAR to the /config/apps directory
- Copies the DB2 JDBC Driver to the /config/resources/DB2Libs directory

\_\_10. Review the contents of the **server.xml** that is used to configure the Liberty server

`cat src/main/liberty/config/server.xml`

- \_\_a. Note the list of features listed. These are the only features used by the new image service microservice. This is a subset of the full set of features required by the PlantsByWebSphere monolith application.

```
<!-- Enable features -->
<featureManager>
<feature>jpa-2.0</feature>
  <feature>jndi-1.0</feature>
  <feature>json-1.0</feature>
  <feature>jdbc-4.0</feature>
  <feature>servlet-3.0</feature>
  <feature>jaxrs-1.1</feature>
  <feature> beanValidation-1.0</feature>
  <feature>transportSecurity-1.0</feature>
  <feature>localConnector-1.0</feature>
</featureManager>
```

- \_\_b. Review the JDBC resource configuration in the server.xml.

**Things to note:**

- The DB2 JDBC library is referenced from the directory location the Dockerfile placed it.
- The database connection information is pulled from environment variables. These environment variables are created from a Kubernetes secret that you will create later in the lab.

```
<library id="DB2JCCLib">
  <fileset dir="/config/resources/DB2Libs" includes="db2jcc4.jar"/>
</library>

  <dataSource id="db2xa" jndiName="jdbc/PlantsByWebSphereDataSource" type="javax.sql.XADataSource">
    <jdbcDriver libraryRef="DB2JCCLib"/>
    <properties.db2.jcc
      <property name="serverName" value="${env.JDBC_HOST}" />
      <property name="portNumber" value="${env.JDBC_PORT}" />
      <property name="databaseName" value="${env.JDBC_DB}" />
      <property name="user" value="${env.JDBC_ID}" />
      <property name="password" value="${env.JDBC_PASSWORD}" />
    </properties.db2.jcc>
  </dataSource>
```

\_\_11. Review the **01-builddocker.sh** script by typing `cat 01-builddocker.sh`

As you can see this script does the following:

- builds the Docker image
- tags it for RHOSCP internal registry
- authenticates with the RHOSCP registry
- pushes the image to the RHOSCP internal registry

```
#!/bin/bash
#
# IBM Cloud Pak for Applications - Proof of Technology
#
# Purpose: Build and Push Liberty Docker image for plantsby websphere image service

IMAGENAME=pbwis
PROJECT=lab6

echo =====
echo Build Liberty Docker image for $IMAGENAME
echo =====
echo
docker build -t $IMAGENAME .

echo =====
echo Tag Liberty Docker image for $IMAGENAME
echo =====
echo
docker tag $IMAGENAME:latest docker-registry.default.svc:5000/$PROJECT/$IMAGENAME:latest
docker login -u $(oc whoami) -p $(oc whoami -t) docker-registry.default.svc:5000

echo =====
echo Push Liberty Docker image for $IMAGENAME
echo =====
echo
docker push docker-registry.default.svc:5000/$PROJECT/$IMAGENAME:latest
```

**\_\_12. Run the 01-builddocker.sh script by typing: `./01-builddocker.sh`**

```
[ibmdemo@icp4a pbwis]$ ./01-builddocker.sh
=====
Build Liberty Docker image for pbwis
=====

Sending build context to Docker daemon 7.064 MB
Step 1/8 : FROM docker.io/ibmcom/websphere-liberty:19.0.0.6-kernel-ubi-min
---> 4b96d9ab9f54
Step 2/8 : COPY src/main/liberty/config/server.xml /config/
---> 20ec58119710
Removing intermediate container 7654d2496dfa
Step 3/8 : COPY target/plantsbywebsphereimageservice.ear /config/apps/
---> 949cf902324f
Removing intermediate container b13f041ca7e8
Step 4/8 : COPY src/main/liberty/lib/DB2Libs/db2jcc4.jar
/config/resources/DB2Libs/
---> a9c8a20a0a43
Removing intermediate container cead85c753c3
Step 5/8 : USER root
---> Running in a851ffdfbf4a
---> 43210173d884
Removing intermediate container a851ffdfbf4a
Step 6/8 : RUN chown default:root -R /opt/ibm/wlp/usr/servers/
---> Running in 54b0af011592
---> 0b73be669497
Removing intermediate container 54b0af011592
Step 7/8 : USER 1001
---> Running in a8099919e061
---> 883578afeb4b
Removing intermediate container a8099919e061
Step 8/8 : RUN configure.sh
---> Running in b2b8a658c8c4

+ WLP_INSTALL_DIR=/opt/ibm/wlp
+ SHARED_CONFIG_DIR=/opt/ibm/wlp/usr/shared/config
+ SHARED_RESOURCE_DIR=/opt/ibm/wlp/usr/shared/resources
+ SNIPPETS_SOURCE=/opt/ibm/helpers/build/configuration_snippets
+ SNIPPETS_TARGET=/config/configDropins/overrides
+ mkdir -p /config/configDropins/overrides
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == client ']'
+ '[' '' == embedded ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
```

```

+ installUtility install --acceptLicense defaultServer
Checking for missing features required by the server ...
The server requires the following additional features: servlet-3.0
transportsecurity-1.0 beanvalidation-1.0 jndi-1.0 json-1.0 localconnector-1.0
jdbc-4.0 jaxrs-1.1 jpa-2.0. Installing features from the repository ...
Establishing a connection to the configured repositories ...
This process might take several minutes to complete.

Successfully connected to all configured repositories.

Preparing assets for installation. This process might take several minutes to
complete.

Additional Liberty features must be installed for this server.

To install the additional features, review and accept the feature license
agreement:
The --acceptLicense argument was found. This indicates that you have
accepted the terms of the license agreement.

Step 1 of 22: Downloading servlet-3.0 ...
Step 2 of 22: Installing servlet-3.0 ...
Step 3 of 22: Downloading ssl-1.0 ...
Step 4 of 22: Installing ssl-1.0 ...
Step 5 of 22: Downloading transportSecurity-1.0 ...
Step 6 of 22: Installing transportSecurity-1.0 ...
Step 7 of 22: Downloading beanValidation-1.0 ...
Step 8 of 22: Installing beanValidation-1.0 ...
Step 9 of 22: Downloading jndi-1.0 ...
Step 10 of 22: Installing jndi-1.0 ...
Step 11 of 22: Downloading json-1.0 ...
Step 12 of 22: Installing json-1.0 ...
Step 13 of 22: Downloading localConnector-1.0 ...
Step 14 of 22: Installing localConnector-1.0 ...
Step 15 of 22: Downloading jdbc-4.0 ...
Step 16 of 22: Installing jdbc-4.0 ...
Step 17 of 22: Downloading jaxrs-1.1 ...
Step 18 of 22: Installing jaxrs-1.1 ...
Step 19 of 22: Downloading jpa-2.0 ...
Step 20 of 22: Installing jpa-2.0 ...
Step 21 of 22: Validating installed fixes ...
Step 22 of 22: Cleaning up temporary files ...

All assets were successfully installed.

Start product validation...
Product validation completed successfully.
+ find /opt/ibm/fixes -type f -name '*.jar' -print0
+ sort -z
+ xargs -0 -n 1 -r -I '{}' java -jar '{}' --installLocation /opt/ibm/wlp
+ find /opt/ibm/wlp -perm -g=w -print0
+ xargs -0 -r chmod -R g+rw
+ /opt/ibm/wlp/bin/server start

Starting server defaultServer.
Server defaultServer started with process ID 112.

```

```

+ /opt/ibm/wlp/bin/server stop

Stopping server defaultServer.
Server defaultServer stopped.
+ rm -rf /output/resources/security/ /output/messaging /logs/console.log
/logs/ffdc /logs/messages.log /logs/messages_20.04.27_20.20.48.0.log
/opt/ibm/wlp/output/.classCache
+ chmod -R g+rxw /opt/ibm/wlp/output/defaultServer
+ find /opt/ibm/wlp -type d -perm -g=x -print0
+ xargs -0 -r chmod -R g+rxw
---> a803ae3a35cc
Removing intermediate container b2b8a658c8c4
Successfully built a803ae3a35cc
=====
Tag Liberty Docker image for pbwis
=====

Login Succeeded
=====
Push Liberty Docker image for pbwis
=====

The push refers to a repository [docker-registry.default.svc:5000/lab6/pbwis]
1cb271121baf: Pushed
655834fb121c: Pushed
d630675ed863: Pushed
9acbf2e93749: Pushed
1392bf3629ed: Pushed
be36d206af93: Mounted from lab5/plantsbywebsphereeee6
ba04059ad9a3: Mounted from lab5/plantsbywebsphereeee6
71532d3a56e4: Mounted from lab5/plantsbywebsphereeee6
790bcf471d32: Mounted from lab5/plantsbywebsphereeee6
fe274995fb89: Mounted from lab5/plantsbywebsphereeee6
9649117d0875: Mounted from lab5/plantsbywebsphereeee6
9e19e22c9a42: Mounted from lab5/plantsbywebsphereeee6
e9417d2583e6: Mounted from lab5/plantsbywebsphereeee6
481324a7ba6d: Mounted from lab5/plantsbywebsphereeee6
26429bebe019: Mounted from lab5/plantsbywebsphereeee6
latest: digest:
sha256:1195de40a5454afad4cccc484ec370df316acfba09f43418fa02b614f375d54e size:
3466

```

**Note:** Several layers from the image may be used from WebSphere Liberty images that may already exist on the file system in the lab environment. This is expected and OK.

- \_\_1. Type `cat 02-createSecret.sh` to review the script file.

The secret creates the credentials and address information for application access to the DB2 database used by both the PlantsByWebSphere Image Service microservice and the modified PlantsByWebSphere application

```
[ibmdemo@icp4a appmodern]$ cat 02-createSecret.sh
#!/bin/bash
#
# IBM Cloud Pak for Applications - Proof of Technology
#
# Purpose: Create secrets for PBW database access

PROJECT=lab5

echo =====
echo Create secrets for PBW database access
echo =====
echo

kubectl -n $PROJECT delete secret db2-secret > /dev/null 2>&1

# clear test used below for illustration
# encoded values employed typically

# Db2 secret
kubectl -n $PROJECT \
  create secret generic db2-secret \
    --from-literal=JDBC_ID=db2inst1 \
    --from-literal=JDBC_PASSWORD=db2inst1 \
    --from-literal=JDBC_HOST=192.168.142.130 \
    --from-literal=JDBC_PORT=50000 \
    --from-literal=JDBC_DB=PBW

[ibmdemo@icp4a appmodern]$
```

- \_\_13. Run the 02-createSecret.sh script by typing: `./02-createSecret.sh`

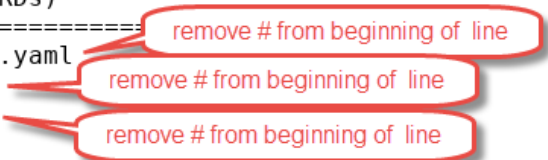
```
[ibmdemo@icp4a pbwis]$ ./02-createSecret.sh
=====
Create secrets for PBW database access
=====

secret/db2-secret created
```

- \_\_2. **The 03-createOperatorArtifacts.sh** script has been created to consolidate the creation of the 3 custom resource definition (CRD's) for the v0.3.0 Open Liberty operator, the required ServiceAccount, Role, and RoleBinding Kubernetes resources for the Liberty Operator.
- \_\_a. Enter the command: `gedit 03-createOperatorArtifacts.sh` to edit the file. Ensure the three lines shown below are **UNCOMMENTED**.
  - \_\_b. Remove the `#` from the three lines if they are commented out (This will uncomment them)
  - \_\_c. Save and close the file

```
# Uncomment the lines below to create the CRD's if lab 4 is not completed

#echo =====
#echo " Create the custom resource definitions (CRDs) "
#echo =====
#oc apply -f operator/application/application-crd.yaml
#oc apply -f operator/application/traces-crd.yaml
#oc apply -f operator/application/dumps-crd.yaml
echo =====
```



These commands create the Custom Resource Definitions (CRDs) for the Open Liberty Operator which only needs to be performed once in a K8s/RHOCP cluster

We modified the CR and security files so that the names and labels specify "*plantsbywebsphere6-operator*" in order to create artifacts specific to this deployment

Using the following commands, you can review the CRD files which define all the operator required resources:

```
cat operator/application/application-crd.yaml
cat operator/application/dumps-crd.yaml
cat operator/application/traces-crd.yaml
cat operator/deploy/role_binding.yaml
cat operator/deploy/role.yaml
cat operator/deploy/service_account.yaml
```

- \_\_14. Type `./03-createOperatorArtifacts.sh` to run the script
- \_\_a. Then type `oc get pods` until the pbwis-operator pod is running

```
[ibmdemo@icp4a pbwis]$ ./03-createOperatorArtifacts.sh
=====
create Liberty operator ServiceAccount, Role, and RoleBinding
=====
serviceaccount/pbwis-operator created
role.rbac.authorization.k8s.io/pbwis-operator created
rolebinding.rbac.authorization.k8s.io/pbwis-operator created
=====
deploy Liberty operator pod
=====
deployment.apps/pbwis-operator created
=====
Run command "oc get pods "
wait until the plantsbywebosphere image service operator pod is ready
before running next script
=====
[ibmdemo@icp4a pbwis]$ oc get pods
NAME                                READY    STATUS    RESTARTS   AGE
pbwis-operator-f8c5ddb8b-9rh6f      1/1      Running   0           24s
```

- \_\_15. Once the operator pod is running type: `./04-deployApplication.sh` which will deploy the application

```
[ibmdemo@icp4a pbwis]$ ./04-deployApplication.sh
=====
deploy the application
=====
openlibertyapplication.openliberty.io/pbwis created
```

- \_\_16. Type `oc get pods` until both the **pbwis operator** and the **pbwis application** are running and ready

```
[ibmdemo@icp4a pbwis]$ oc get pods
NAME                                READY    STATUS    RESTARTS   AGE
pbwis-7b45487c8d-525mc              1/1      Running   0           54s
pbwis-operator-547dff98b4-4kwns      1/1      Running   0           3m
[ibmdemo@icp4a pbwis]$
```

- \_\_\_17. Type `oc get all` to ensure that the pbwis application pod is **Ready 1/1** and **Running** and that the additional artifacts such as the `service/pbwis` for accessing the microservice has been created

```
[ibmdemo@icp4a pbwis]$ oc get all
```

| NAME                               | READY | STATUS  | RESTARTS | AGE |
|------------------------------------|-------|---------|----------|-----|
| pod/pbwis-7b45487c8d-skds9         | 1/1   | Running | 0        | 33s |
| pod/pbwis-operator-f8c5ddb8b-9rh6f | 1/1   | Running | 0        | 10m |

| NAME          | TYPE      | CLUSTER-IP    | EXTERNAL-IP | PORT(S)  | AGE |
|---------------|-----------|---------------|-------------|----------|-----|
| service/pbwis | ClusterIP | 172.30.13.191 | <none>      | 9080/TCP | 33s |

| NAME                           | DESIRED | CURRENT | UP-TO-DATE | AVAILABLE |
|--------------------------------|---------|---------|------------|-----------|
| deployment.apps/pbwis          | 1       | 1       | 1          | 1         |
| deployment.apps/pbwis-operator | 1       | 1       | 1          | 1         |

| NAME                                     | DESIRED | CURRENT | READY | AGE |
|--|---------|---------|-------|-----|
| replicaset.apps/pbwis-7b45487c8d         | 1       | 1       | 1     | 33s |
| replicaset.apps/pbwis-operator-f8c5ddb8b | 1       | 1       | 1     | 10m |

| NAME                                 | DOCKER  | REPO       |
|--------------------------------------|---------|------------|
| imagestream.image.openshift.io/pbwis | docker- |            |
| registry.default.svc:5000/lab6/pbwis | latest  | 9 days ago |

| NAME   | REASON | AGE | READY |
|--|--------|-----|-------|
| clusterchannelprovisioner.eventing.knative.dev/in-memory         |        | 51d | True  |
| clusterchannelprovisioner.eventing.knative.dev/in-memory-channel |        | 51d | True  |

| NAME  | READY | REASON |
|---|-------|--------|
| clusteringress.networking.internal.knative.dev/route-17de13e9-fe3a-11e9-9829-000c29ef9df2 |       |        |

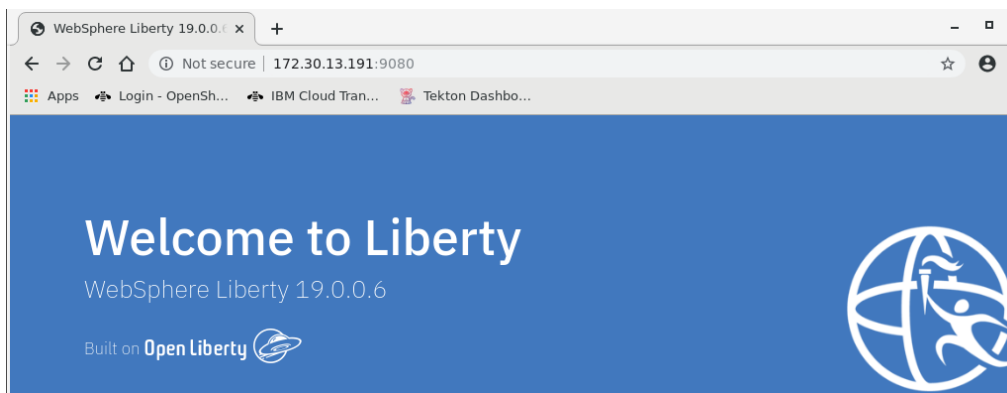
\_\_18. You can access the application using the **ClusterIP**

\_\_a. Open the **Chrome browser** and using the **ClusterIP**

URL: <http://<cluster-ip>:port>

Example: <http://172.30.13.191:9080>

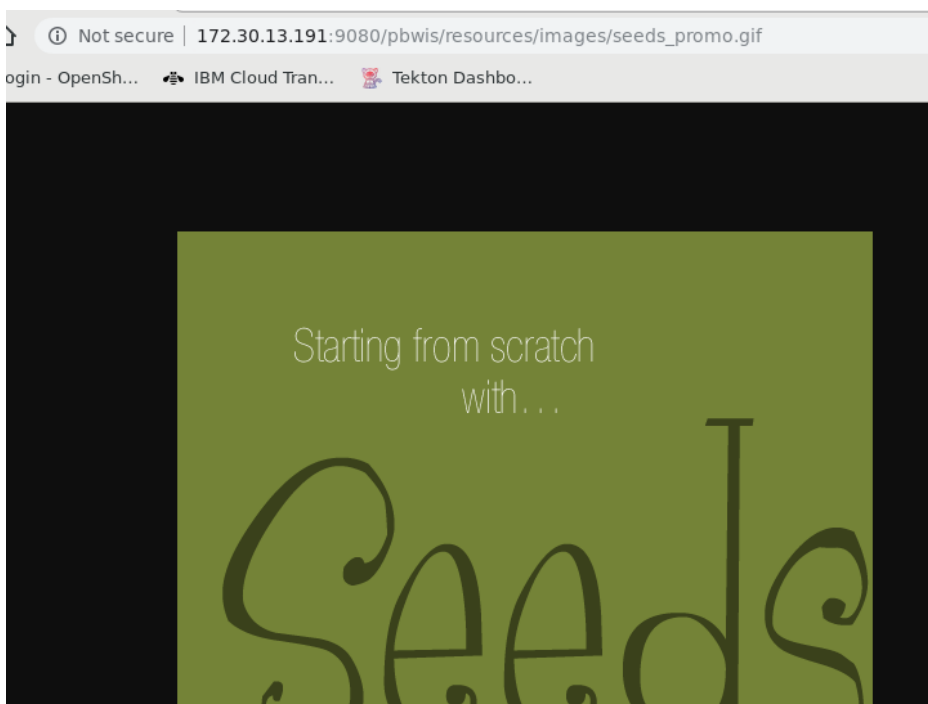
**NOTE that your Cluster-IP address will very likely be different**



\_\_19. Add **/pbwis/resources/images/seeds\_promo.gif** to the URL in the browser,

Example: [http://172.30.13.191:9080/pbwis/resources/images/seeds\\_promo.gif](http://172.30.13.191:9080/pbwis/resources/images/seeds_promo.gif) to test accessing a static image

**Note substitute your Cluster-IP address**





You only tested the image service rendering image from a file. Later in the lab, you will also verify the new Image Service microservice render images from the DB2 database.

At this point in the lab, you have not populated the database with the images. That will happen later in the lab.

- \_\_20. Now that the image service is running, type `cd ~/student/lab6/pbwv2` to switch to the directory with the modified PlantsByWebSphere deployment artifacts, then type `ls` to list the files.

```
[ibmdemo@icp4a pbwis]$ cd ~/student/lab6/pbwv2

[ibmdemo@icp4a pbwv2]$ ls
01-buildDocker.sh          04-deployApplication.sh  Dockerfile  src
03-createOperatorArtifacts.sh  99-cleanUp.sh           operator    target
```

Note that aside from the omission of a script to create a secret for DB2, the artifacts and layout is identical to the pbwis directory.

**NOTE:** There's no need to create a secret for accessing DB2 from this project (namespace), since that was accomplished several steps previously by running the 02-createSecret.sh script for the image service

- \_\_21. Review the **01-buildDocker.sh** script: `cat 01-buildDocker.sh`

As you can see, this script builds the Docker image, tags it, authenticates with the RHOCp registry, then pushes the image to the RHOSCP image registry.

- \_\_22. Run the 01-buildDocker.sh script: `./01-buildDocker.sh`

**Note:** The expected output from the script is illustrated below:

```
[ibmdemo@icp4a pbwv2]$ ./01-buildDocker.sh
=====
Build Liberty Docker image for plantsbywebsphereee6v2
=====
Sending build context to Docker daemon 7.265 MB
Step 1/8 : FROM docker.io/ibmcom/websphere-liberty:19.0.0.6-kernel-ubi-min
---> 4b96d9ab9f54
Step 2/8 : COPY src/main/liberty/config/server.xml /config/
---> afbddcd23d43
Removing intermediate container 0693c4cda72d
Step 3/8 : COPY target/plantsbywebsphereee6v2.ear /config/apps/
---> bdb41924c704
Removing intermediate container 32e542a56fa9
Step 4/8 : COPY src/main/liberty/lib/DB2Libs/db2jcc4.jar
/config/resources/DB2Libs/
---> 2993baf28ace
Removing intermediate container a8c9fc73cec8
Step 5/8 : USER root
---> Running in 9dc14fcd3fa4
---> c200ba537d55
Removing intermediate container 9dc14fcd3fa4
Step 6/8 : RUN chown default:root -R /opt/ibm/wlp/usr/servers/
---> Running in daef2bb69cea

---> 59d86df3fb32
Removing intermediate container daef2bb69cea
Step 7/8 : USER 1001
---> Running in eb5a5e85d18a
---> d987747c860a
Removing intermediate container eb5a5e85d18a
Step 8/8 : RUN configure.sh
---> Running in 705039707a8b

+ WLP_INSTALL_DIR=/opt/ibm/wlp
+ SHARED_CONFIG_DIR=/opt/ibm/wlp/usr/shared/config
+ SHARED_RESOURCE_DIR=/opt/ibm/wlp/usr/shared/resources
+ SNIPPETS_SOURCE=/opt/ibm/helpers/build/configuration_snippets
+ SNIPPETS_TARGET=/config/configDropins/overrides
+ mkdir -p /config/configDropins/overrides
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == client ']'
+ '[' '' == embedded ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ installUtility install --acceptLicense defaultServer
Checking for missing features required by the server ...
The server requires the following additional features: jsp-2.3
transportsecurity-1.0 jsf-2.2 servlet-3.1 jndi-1.0 cdi-1.2 javamail-1.5
beanvalidation-1.1 ejblite-3.2 jpa-2.0. Installing features from the
repository ...
Establishing a connection to the configured repositories ...
This process might take several minutes to complete.
```

Successfully connected to all configured repositories.

Preparing assets for installation. This process might take several minutes to complete.

Additional Liberty features must be installed for this server.

To install the additional features, review and accept the feature license agreement:

The --acceptLicense argument was found. This indicates that you have accepted the terms of the license agreement.

```

Step 1 of 32: Downloading el-3.0 ...
Step 2 of 32: Installing el-3.0 ...
Step 3 of 32: Downloading servlet-3.1 ...
Step 4 of 32: Installing servlet-3.1 ...
Step 5 of 32: Downloading jsp-2.3 ...
Step 6 of 32: Installing jsp-2.3 ...
Step 7 of 32: Downloading ssl-1.0 ...
Step 8 of 32: Installing ssl-1.0 ...
Step 9 of 32: Downloading transportSecurity-1.0 ...
Step 10 of 32: Installing transportSecurity-1.0 ...
Step 11 of 32: Downloading jsf-2.2 ...
Step 12 of 32: Installing jsf-2.2 ...
Step 13 of 32: Downloading jndi-1.0 ...
Step 14 of 32: Installing jndi-1.0 ...
Step 15 of 32: Downloading cdi-1.2 ...
Step 16 of 32: Installing cdi-1.2 ...
Step 17 of 32: Downloading javaMail-1.5 ...
Step 18 of 32: Installing javaMail-1.5 ...
Step 19 of 32: Downloading beanValidation-1.1 ...
Step 20 of 32: Installing beanValidation-1.1 ...
Step 21 of 32: Downloading ejbLite-3.2 ...
Step 22 of 32: Installing ejbLite-3.2 ...
Step 23 of 32: Downloading servlet-3.0 ...
Step 24 of 32: Installing servlet-3.0 ...
Step 25 of 32: Downloading beanValidation-1.0 ...
Step 26 of 32: Installing beanValidation-1.0 ...
Step 27 of 32: Downloading jdbc-4.0 ...
Step 28 of 32: Installing jdbc-4.0 ...
Step 29 of 32: Downloading jpa-2.0 ...
Step 30 of 32: Installing jpa-2.0 ...
Step 31 of 32: Validating installed fixes ...
Step 32 of 32: Cleaning up temporary files ...

```

All assets were successfully installed.

Start product validation...

Product validation completed successfully.

```

+ find /opt/ibm/fixes -type f -name '*.jar' -print0
+ sort -z
+ xargs -0 -n 1 -r -I '{}' java -jar '{}' --installLocation /opt/ibm/wlp
+ find /opt/ibm/wlp -perm -g=w -print0
+ xargs -0 -r chmod -R g+rw
+ /opt/ibm/wlp/bin/server start

```

```

Starting server defaultServer.
Server defaultServer started with process ID 114.
+ /opt/ibm/wlp/bin/server stop

Stopping server defaultServer.
Server defaultServer stopped.
+ rm -rf /output/resources/security/ /output/messaging /logs/console.log
/logs/ffdc /logs/messages.log /logs/messages_20.05.07_23.38.14.0.log
/opt/ibm/wlp/output/.classCache
+ chmod -R g+rxw /opt/ibm/wlp/output/defaultServer
+ find /opt/ibm/wlp -type d -perm -g=x -print0
+ xargs -0 -r chmod -R g+rxw
---> 2bf2d7286547
Removing intermediate container 705039707a8b
Successfully built 2bf2d7286547
=====
Tag Liberty Docker image for plantsbywebsphereee6v2
=====

Login Succeeded
=====
Push Liberty Docker image for plantsbywebsphereee6v2
=====

The push refers to a repository [docker-
registry.default.svc:5000/lab6/plantsbywebsphereee6v2]
32144845d0e4: Pushed
a4d46fbb98ff: Pushed
6bca9750c253: Pushed
1d92d2c5de99: Pushed
06d3987fedcd: Pushed
be36d206af93: Mounted from lab6/pbwis
ba04059ad9a3: Mounted from lab6/pbwis
71532d3a56e4: Mounted from lab6/pbwis
790bcf471d32: Mounted from lab6/pbwis
fe274995fb89: Mounted from lab6/pbwis
9649117d0875: Mounted from lab6/pbwis
9e19e22c9a42: Mounted from lab6/pbwis
e9417d2583e6: Mounted from lab6/pbwis
481324a7ba6d: Mounted from lab6/pbwis
26429bebe019: Mounted from lab6/pbwis
latest: digest:
sha256:0ce467f37954c36e1afe97ea23956efee555bc3b6d6ae5204bf1fe3cebe79231 size:
3466
[ibmdemo@icp4a pbwv2]$

```

Note that several layers from the image create for the image service are reused, this because both rely on the same underlying WebSphere Liberty image.

- \_\_23. **The 03-createOperatorArtifacts.sh** script has been created to consolidate the creation of the 3 custom resource definition (CRD's) for the v0.3.0 Open Liberty operator, the required ServiceAccount, Role, and RoleBinding Kubernetes resources for the Liberty Operator.

These commands create the Custom Resource Definitions (CRDs) for the Open Liberty Operator which only needs to be performed once in a K8s/RHOCP cluster

**Note:** We modified the CR and security files so that the names and labels specify “plantsbywebsphereee6v2” in order to create artifacts specific to this deployment

**Note:** The “open-liberty-operator” has been replaced in multiple places with “plantsbywebsphereee6v2” to create artifacts specific to deployment of this application.

- \_\_24. Type `./03-createOperatorArtifacts.sh` to run the script, then type `oc get pods` until the plantsbywebsphereee6v2-operator pod is running

```
[ibmdemo@icp4a pbwv2]$ ./03-createOperatorArtifacts.sh
=====
create Liberty operator ServiceAccount, Role, and RoleBinding
=====
serviceaccount/plantsbywebsphereee6v2-operator created
role.rbac.authorization.k8s.io/plantsbywebsphereee6v2-operator created
rolebinding.rbac.authorization.k8s.io/plantsbywebsphereee6v2-operator created
=====
deploy Liberty operator pod
=====
deployment.apps/plantsbywebsphereee6v2-operator created
=====
Run command "oc get pods "
wait until the plantsbywebsphere image service operator pod is ready
before running next script
=====
[ibmdemo@icp4a pbwv2]$ oc get pods
```

| NAME   | READY | STATUS  | RESTARTS | AGE |
|--|-------|---------|----------|-----|
| pbwis-7b45487c8d-ms2k9                           | 1/1   | Running | 0        | 17m |
| pbwis-operator-f8c5ddb8b-84vz1                   | 1/1   | Running | 0        | 17m |
| plantsbywebsphereee6v2-operator-5c97784776-wjt9k | 1/1   | Running | 0        | 6s  |

\_\_25. Review the **application-cr** used for this lab by typing:

```
cat operator/application/application-cr.yaml
```

```
[ibmdemo@icp4a pbwv2]$ cat operator/application/application-cr.yaml
# Revised Application CR for Open Liberty Operator v0.3
apiVersion: openliberty.io/v1beta1
kind: OpenLibertyApplication
metadata:
  name: plantsbywebsphereeee6v2
spec:
  replicas: 1
  applicationImage: docker-registry.default.svc:5000/lab6/plantsbywebsphereeee6v2
  tag: latest
# Add readiness and liveness probes
  ports:
    readinessProbe:
      httpGet:
        path: /
        port: 9080
        initialDelaySeconds: 3
        periodSeconds: 5
    livenessProbe:
      httpGet:
        path: /
        port: 9080
        initialDelaySeconds: 40
        periodSeconds: 10
#Expose a route
  expose: true
#Environment entries from db2-secret to access PBW
  envFrom:
    - secretRef:
        name: db2-secret
[ibmdemo@icp4a pbwv2]$
```

- The file creates an [OpenLibertyApplication](#) instance named `plantsbywebsphereeee6v2`.
- It used the docker image in the RHOSCP registry, specified by the [applicationImage](#).
- It created a [readinessProbe](#) and a [livenessProbe](#) to monitor the application.
- A [route](#) is created to expose the application for external access
- [environmental variables](#) ([envFrom](#)) are imported for use by the application from a secret(`db2-secret`) (which contains database information and credentials)

- \_\_26. Once the operator pod is running type `./04-deployApplication.sh` which will deploy the application using the parameters specified in the `application-cr.yaml` (from step 25 above)

```
[ibmdemo@icp4a pbwv2]$ ./04-deployApplication.sh
=====
deploy the application
=====
openlibertyapplication.openliberty.io/plantsbywebsphereee6v2 created
```

- \_\_27. Type `oc get pods` until both the `plantsbywebsphereee6v2` and the `plantsbywebsphereee6v2-operator` pods are running and ready

```
[ibmdemo@icp4a pbwv2]$ oc get pods
```

| NAME   | READY | STATUS  | RESTARTS | AGE |
|--|-------|---------|----------|-----|
| pbwis-7b45487c8d-525mc                           | 1/1   | Running | 0        | 56m |
| pbwis-operator-547dff98b4-4kwns                  | 1/1   | Running | 0        | 59m |
| plantsbywebsphereee6v2-7b9f65d5dc-b758z          | 1/1   | Running | 0        | 35s |
| plantsbywebsphereee6v2-operator-6d5649b894-bjkk8 | 1/1   | Running | 0        | 10m |

```
[ibmdemo@icp4a pbwv2]$
```

- \_\_28. Type `oc get all` to ensure that the application pod is running and the additional artifacts such as the service for accessing the application has been created

```
[ibmdemo@icp4a pbwv2]$ oc get all
```

| NAME   | READY | STATUS  |   |
|--|-------|---------|---|
| RESTARTS AGE   |       |         |   |
| pod/pbwis-7b45487c8d-ms2k9                           | 1/1   | Running | 0 |
| 17m  |       |         |   |
| pod/pbwis-operator-f8c5ddb8b-84vz1                   | 1/1   | Running | 0 |
| 17m  |       |         |   |
| pod/plantsbywebsphereee6v2-7b9f65d5dc-fx74w          | 1/1   | Running | 0 |
| 18s  |       |         |   |
| pod/plantsbywebsphereee6v2-operator-5c97784776-wjt9k | 1/1   | Running | 0 |
| 53s  |       |         |   |

| NAME                           | TYPE      | CLUSTER-IP    | EXTERNAL-IP |
|--------------------------------|-----------|---------------|-------------|
| PORT(S) AGE                    |           |               |             |
| service/pbwis                  | ClusterIP | 172.30.13.191 | <none>      |
| 9080/TCP 17m                   |           |               |             |
| service/plantsbywebsphereee6v2 | ClusterIP | 172.30.58.200 | <none>      |
| 9080/TCP 18s                   |           |               |             |

| NAME  | DESIRED | CURRENT | UP-TO- |
|---|---------|---------|--------|
| DATE AVAILABLE AGE                              |         |         |        |
| deployment.apps/pbwis                           | 1       | 1       | 1      |
| 1 17m   |         |         |        |
| deployment.apps/pbwis-operator                  | 1       | 1       | 1      |
| 1 17m   |         |         |        |
| deployment.apps/plantsbywebsphereee6v2          | 1       | 1       | 1      |
| 1 18s   |         |         |        |
| deployment.apps/plantsbywebsphereee6v2-operator | 1       | 1       | 1      |
| 1 53s   |         |         |        |

| NAME   | DESIRED | CURRENT |
|--|---------|---------|
| READY AGE  |         |         |
| replicaset.apps/pbwis-7b45487c8d                           | 1       | 1       |
| 1 17m  |         |         |
| replicaset.apps/pbwis-operator-f8c5ddb8b                   | 1       | 1       |
| 1 17m  |         |         |
| replicaset.apps/plantsbywebsphereee6v2-7b9f65d5dc          | 1       | 1       |
| 1 18s  |         |         |
| replicaset.apps/plantsbywebsphereee6v2-operator-5c97784776 | 1       | 1       |
| 1 53s  |         |         |

| NAME  | DOCKER | REPO        |
|---|--------|-------------|
| imagestream.image.openshift.io/pbwis                  | latest | 1 hour ago  |
| registry.default.svc:5000/lab6/pbwis                  | latest | 17 mins ago |
| imagestream.image.openshift.io/plantsbywebsphereee6v2 | latest | 17 mins ago |
| registry.default.svc:5000/lab6/plantsbywebsphereee6v2 | latest | 17 mins ago |

| NAME  | HOST/PORT                       |
|---|---------------------------------|
| route.route.openshift.io/plantsbywebsphereee6v2 | plantsbywebsphereee6v2-         |
| lab6.apps.icp4a.pot.com                         | plantsbywebsphereee6v2 9080-tcp |
| None  |                                 |

| NAME   | READY |
|--|-------|
| clusterchannelprovisioner.eventing.knative.dev/in-memory         | True  |
| 53d  |       |
| clusterchannelprovisioner.eventing.knative.dev/in-memory-channel | True  |
| 53d  |       |

| NAME  | READY | REASON |
|---|-------|--------|
| clusteringress.networking.internal.knative.dev/route-17de13e9-fe3a-11e9-9829-000c29ef9df2 | True  |        |

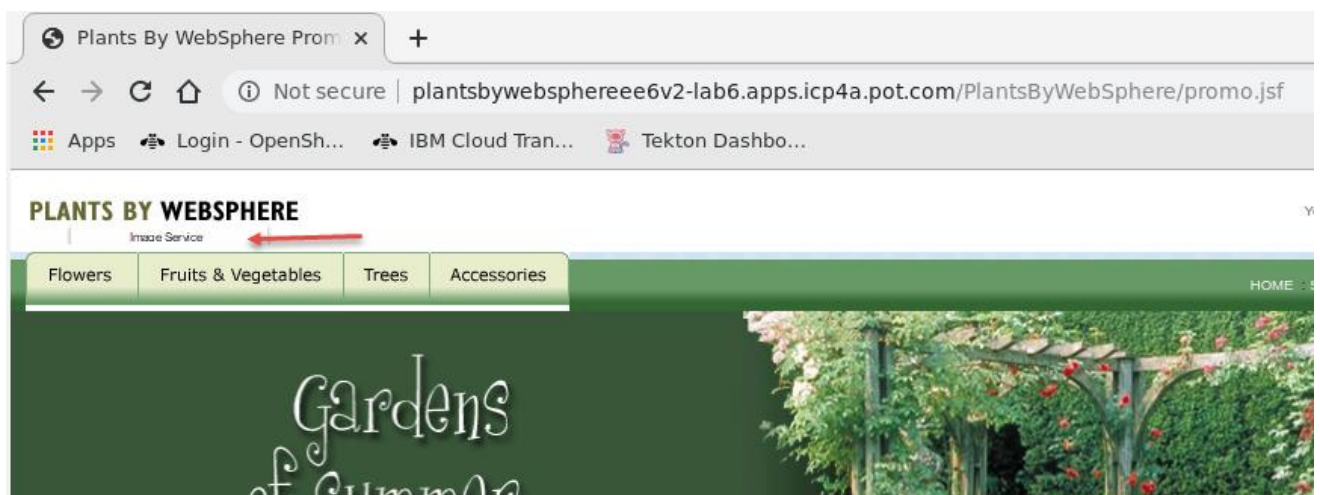
[ibmdemo@icp4a pbwv2]\$

29. Access the PlantsByWebSphere version 2 application using the rote that was created.

For the plantsbywebsphereee6v2 there is a route that has been created to access the application.

The route was configured in the **application.cr** file with the addition of the “**expose: true**” key value pair. A route allows for external access to the application.

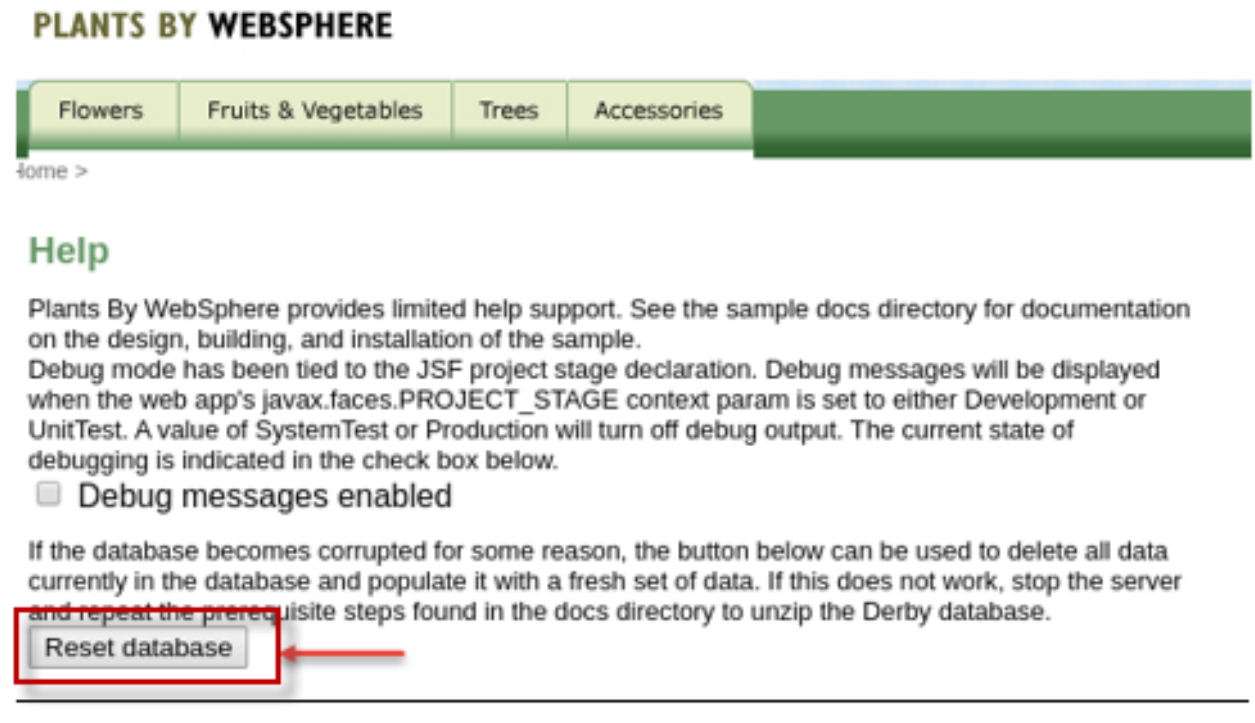
Use the PATH; plantsbywebsphereee6v2-lab6.apps.icp4a.pot.com to construct the following URL in a browser <http://plantsbywebsphereee6v2-lab6.apps.icp4a.pot.com/PlantsByWebSphere>



- \_\_30. Now, use the application to populate the DB2 database used by the application.
- \_\_a. Click on [Help](#) in the upper right-hand corner of the Plants By WebSphere page



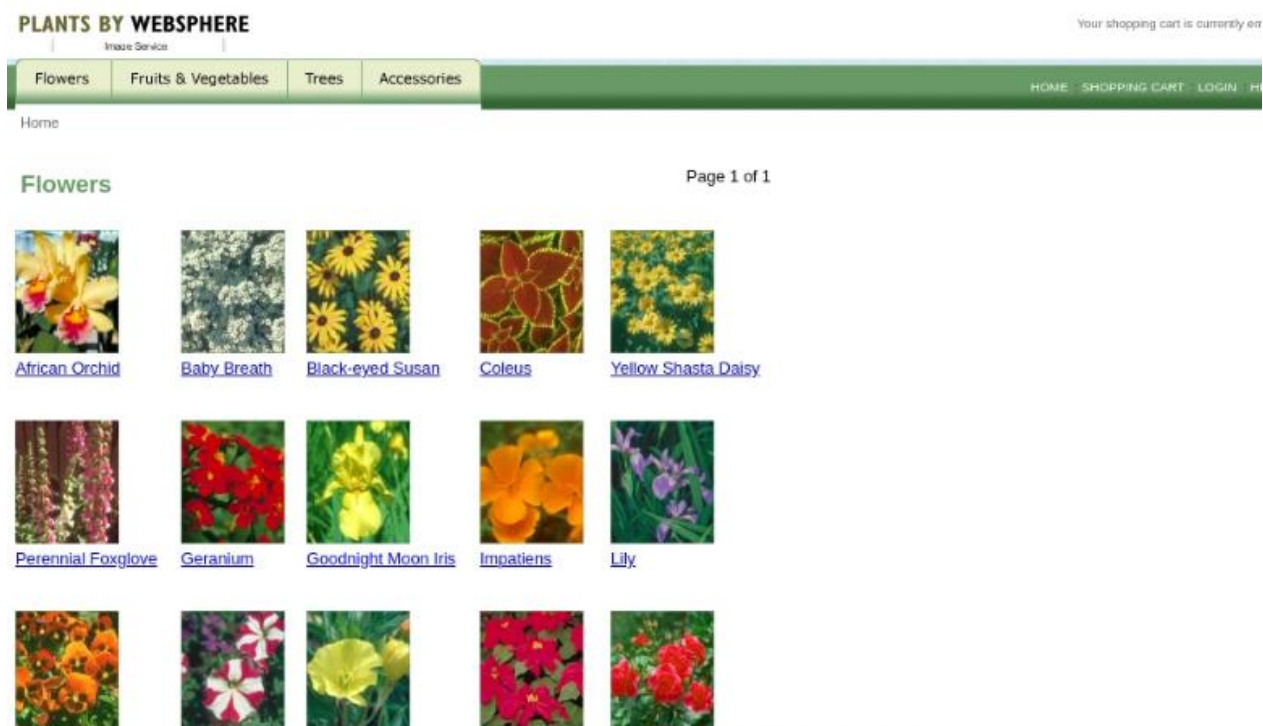
- \_\_b. Click on [Reset database](#) which will populate the database tables, which is required before navigating through the application.



- \_\_31. Navigate to the Flowers tab, to view the images of flowers in the catalog.



- \_\_32. If the application and the new Image Service microservice is working properly, you will see the list of flowers and their images, as queried from the database via the image service microservice.



- \_\_33. If you would like further evidence that the images are being accessed from image service, you can enter the following commands, which will delete the pbwis Image service microservice.

```
oc delete deployment.apps/pbwis-operator
oc delete deployment.apps/plantsbywebsphereee6v2-operator
oc delete deployment.apps/pbwis
```

```
[ibmdemo@icp4a pbwv2]$ oc delete deployment.apps/pbwis-operator
deployment.apps "pbwis-operator" deleted

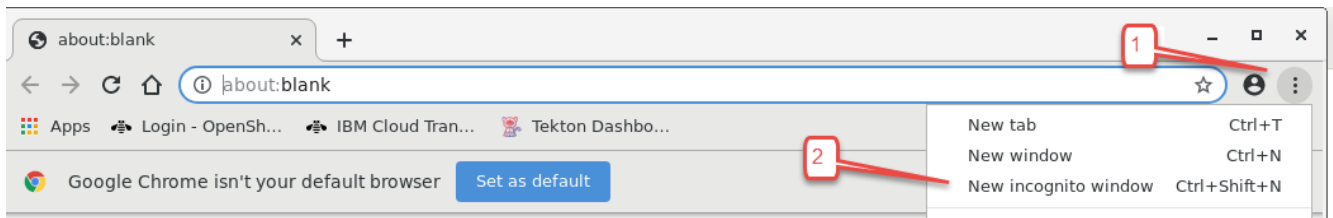
[ibmdemo@icp4a pbwv2]$ oc delete deployment.apps/plantsbywebsphereee6v2-operator
deployment.apps "plantsbywebsphereee6v2-operator" deleted

[ibmdemo@icp4a pbwv2]$ oc delete deployment.apps/pbwis
deployment.apps "pbwis" deleted
```

- \_\_34. The enter `oc get pods` until the only pod running is the **plantsbywebsphereee6v2** pod.

```
[ibmdemo@icp4a pbwv2]$ oc get pods
NAME                                READY   STATUS    RESTARTS   AGE
plantsbywebsphereee6v2-7b9f65d5dc-b758z  1/1     Running   0           22m
[ibmdemo@icp4a pbwv2]$
```

- \_\_35. Close all open browsers, then re-open the browser and click the **control icon** (three vertical dots) and click on **New incognito window** (this to ensure that content is not cached)



\_\_36. Enter the following URL in a incognito browser window: <http://plantsbywebsphereee6v2-lab6.apps.icp4a.pot.com/PlantsByWebSphere>

\_\_a. Notice that the images are missing from the page



\_\_37. To redeploy the image service, enter the following commands from a shell open in **/home/ibmdemo/student/lab6/pbvw2** directory.

```
cd ~/student/lab6/pbwis
./03-createOperatorArtifacts.sh
./04-deployApplication.sh
```

```
[ibmdemo@icp4a pbvw2] cd ~/student/lab6/pbwis

[ibmdemo@icp4a pbwis]$ ./03-createOperatorArtifacts.sh
=====
create Liberty operator ServiceAccount, Role, and RoleBinding
=====
serviceaccount/pbwis-operator unchanged
role.rbac.authorization.k8s.io/pbwis-operator configured
rolebinding.rbac.authorization.k8s.io/pbwis-operator unchanged
=====
deploy Liberty operator pod
=====
deployment.apps/pbwis-operator created
=====
Run command "oc get pods "
wait until the plantsbywebsphere image service operator pod is ready
before running next script
=====
[ibmdemo@icp4a pbwis]$ ./04-deployApplication.sh
=====
deploy the application
=====
openlibertyapplication.openliberty.io/pbwis configured
```

\_\_38. Enter `oc get pods` until the pbis pod is running and ready

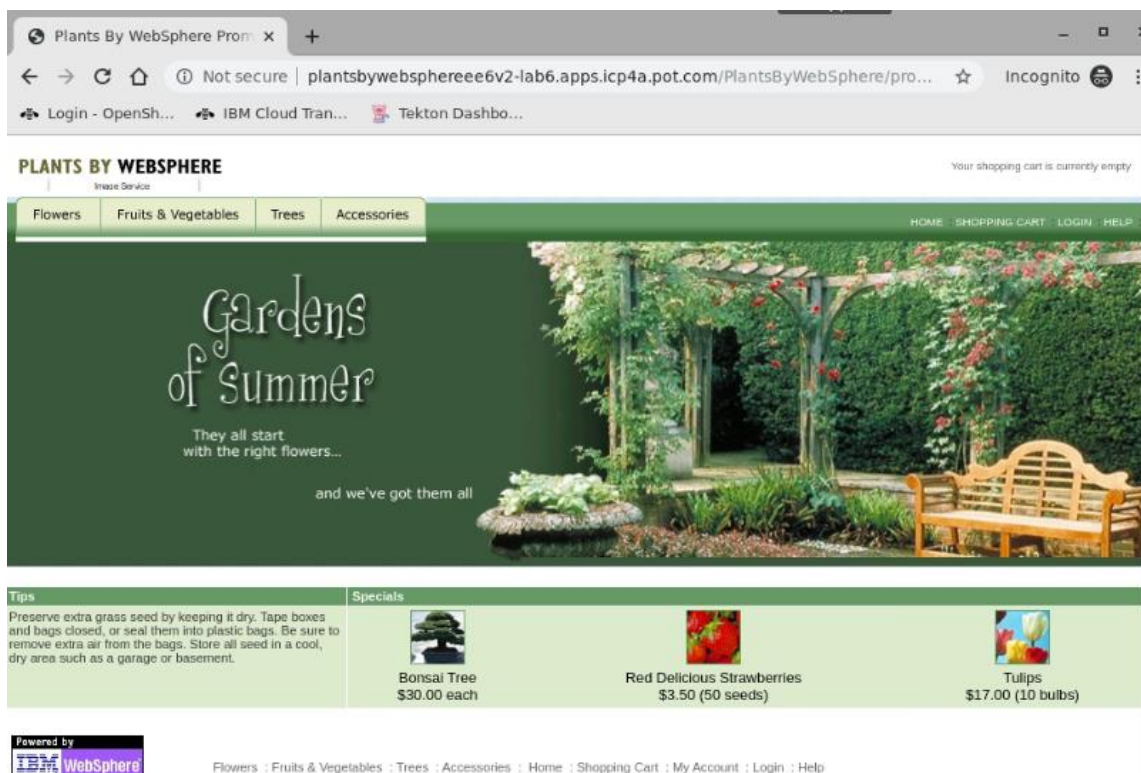
```
[ibmdemo@icp4a pbwis]$ oc get pods
```

| NAME                                     | READY | STATUS  | RESTARTS | AGE |
|--|-------|---------|----------|-----|
| pbwis-7b45487c8d-j4hmq                   | 1/1   | Running | 0        | 55s |
| pbwis-operator-547dff98b4-psgcq          | 1/1   | Running | 0        | 1m  |
| plantsbywebsphereeee6v2-7b9f65d5dc-b758z | 1/1   | Running | 0        | 35m |

```
[ibmdemo@icp4a pbwis]$
```

\_\_39. Enter the following URL in a incognito browser window <http://plantsbywebsphereeee6v2-lab6.apps.icp4a.pot.com/PlantsByWebSphere>

\_\_a. The images are once again being displayed since the image service is back on-line.



## 6.4 Conclusion

Congratulations you have completed the lab and started your journey from monolithic applications to microservices.

## End of Lab 06: Application Modernization with Microservices

## Appendix 1 – Servlet Filter

```

package com.ibm.websphere.samples.pbw.war;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.annotation.*;

/**
 * Servlet Filter implementation class RedirectFilter
 */
@WebFilter(filterName="RedirectFilter",
    servletNames={"com.ibm.websphere.samples.pbw.war.ImageServlet", "FacesServlet"},
    urlPatterns="*" )

public class RedirectFilter implements Filter {

    // Read the Kubernetes service environment variables for the Image Service and
    // construct the Image Service service endpoint
    String httpPre = "http://";
    String colon = ":";
    String pbwisIp = System.getenv("PBWIS_SERVICE_HOST");
    String pbwisPort = System.getenv("PBWIS_SERVICE_PORT");
    String pbwisUrl = "/pbwis";
    String serviceEndpoint = httpPre+pbwisIp+colon+pbwisPort+pbwisUrl;

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
    throws IOException, ServletException {

        if(response.isCommitted() || (serviceEndpoint == null)) {
            //can't do anything as the response has already been committed
            chain.doFilter(request, response);
            return;
        }
        HttpServletResponse resp = (HttpServletResponse) response;
        HttpServletRequest req = (HttpServletRequest) request;
        if("ln=images".equals(req.getQueryString())) {
            String path = req.getServletPath();
            if(path.endsWith(".jsf")) {
                String resource = path.substring(path.lastIndexOf('/'), path.lastIndexOf('.'));
                // resp.sendRedirect(serviceEndpoint + "/images/resources" + resource);
                resp.sendRedirect(serviceEndpoint + "/resources/images" + resource);
                return;
            }
        }
    }
}

```

```
    }
    if(req.getServletPath().endsWith("/ImageServlet")) {
        //      resp.sendRedirect(serviceEndpoint + "/images/product/inventory/" +
req.getParameterMap().get("inventoryID")[0]);
        resp.sendRedirect(serviceEndpoint + "/product/inventory/" +
req.getParameterMap().get("inventoryID")[0]);
        return;
    }
    chain.doFilter(request, response);
}

public void init(FilterConfig fConfig) throws ServletException {
    // TODO Auto-generated method stub
}

public void destroy() {
    // TODO Auto-generated method stub
}
}
```

## Appendix 2 – Image Service Implementation

```
// Enables JAX-RS
package com.ibm.websphere.samples.pbw.ms.image;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("/product/*")

public class ImageApplication extends Application
{

}

// ImageService REST endpoint for the monolith to access the image service

package com.ibm.websphere.samples.pbw.ms.image;

import java.io.IOException;
import java.io.OutputStream;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.persistence.EntityManager;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;
import javax.ws.rs.core.Response.Status;
import javax.ws.rs.core.StreamingOutput;

@Path("/inventory")
public class ImageService {

    private EntityManager getEntityManager() {
        try {
            Context ctx = new InitialContext();
            return (EntityManager)ctx.lookup("java:comp/env/images/em");
        } catch (NamingException e) {
        }
        return null;
    }

    @GET
    @Produces({"image/jpeg"})
    @Path("/{inventoryID}")
    public Response getMessage(@PathParam("inventoryID") String inventoryID) {
        Inventory inv = getInv(inventoryID);
        if (inv != null) {
            final byte[] retval = inv.getImgbytes();
            StreamingOutput stream = new StreamingOutput() {
```

```

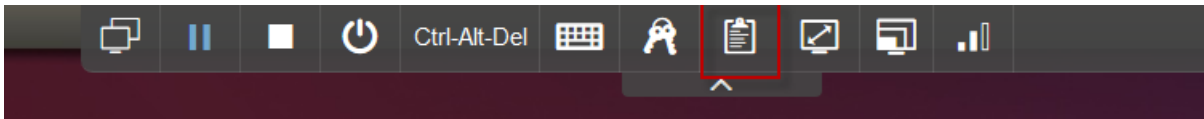
        public void write(OutputStream output) throws IOException,
WebApplicationException {
            output.write(retval);
        }
    };
    return Response.ok(stream).build();
}
return Response.status(Response.Status.NOT_FOUND).build();
}
private Inventory getInv(String inventoryID) {
    EntityManager em = getEntityManager();
    if (em == null) {
        return null;
    }
    return (Inventory)em.find(Inventory.class, inventoryID);
}
}
```

## Appendix: SkyTap Tips for labs

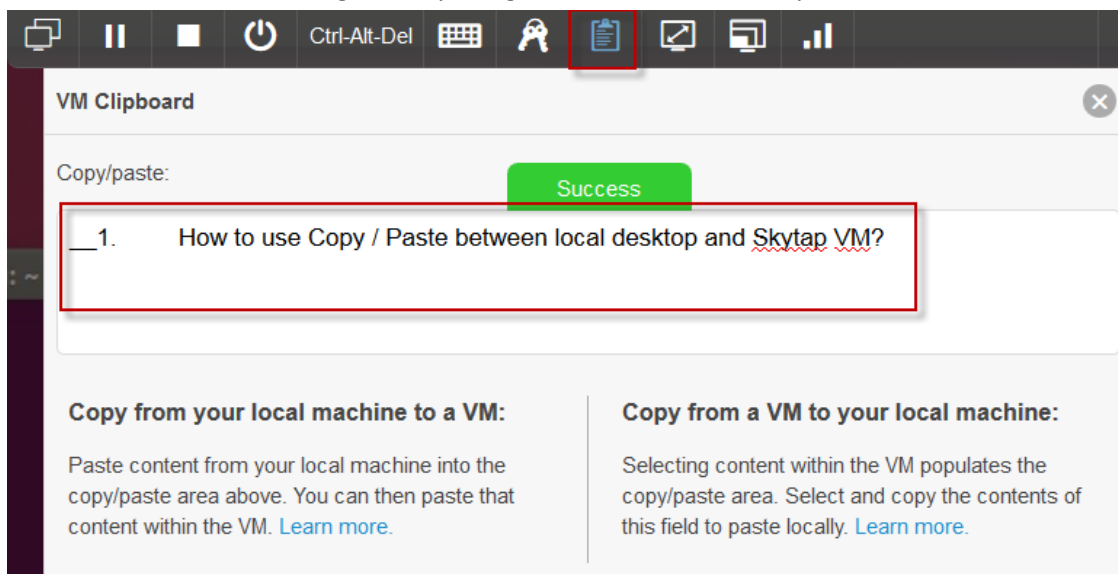
### 6.5 How to use Copy / Paste between local desktop and Skytap VM

Using copy / Paste capabilities between the lab document (PDF) on your local workstation to the VM is a good approach to more efficiently work through a lab, while reducing the typing errors that often occur when manually entering data.

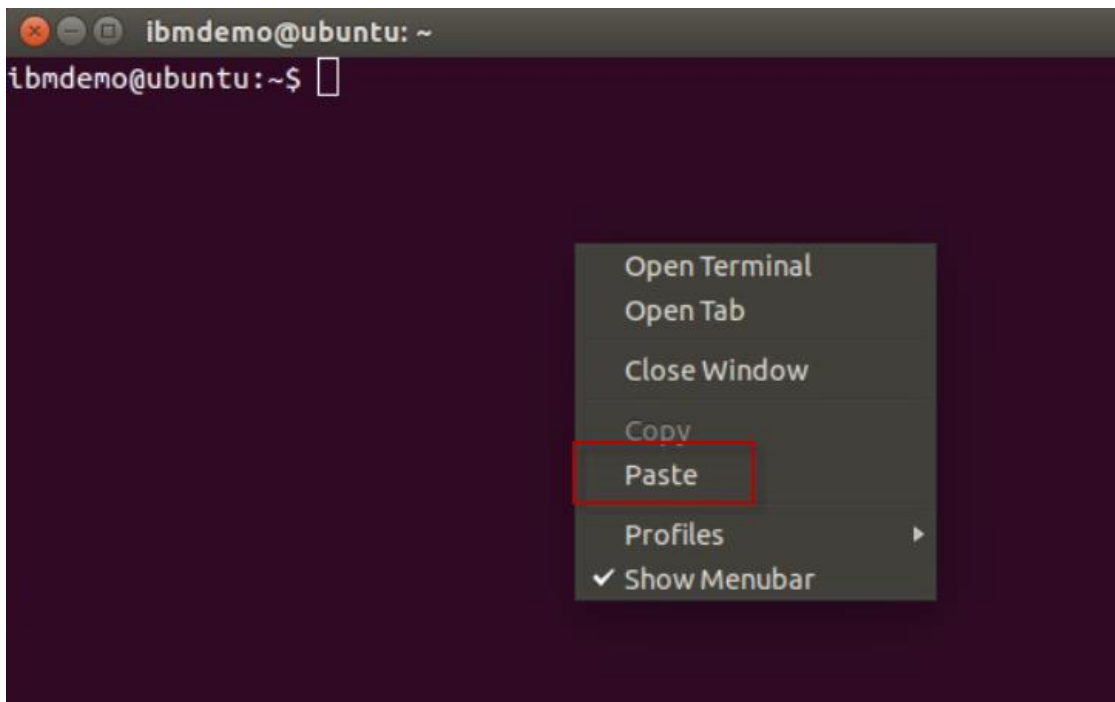
- \_\_1. In SkyTap, you will find that any text copied to the clipboard on your local workstation is not available to be pasted into the VM on SkyTap. So how can you easily accomplish this?
  - \_\_a. First copy the text you intend to paste, from the lab document, to the clipboard on your local workstation, as you always have (CTRL-C)
  - \_\_b. Return to the SkyTap environment and click on the Clipboard at the top of the SkyTap session window.



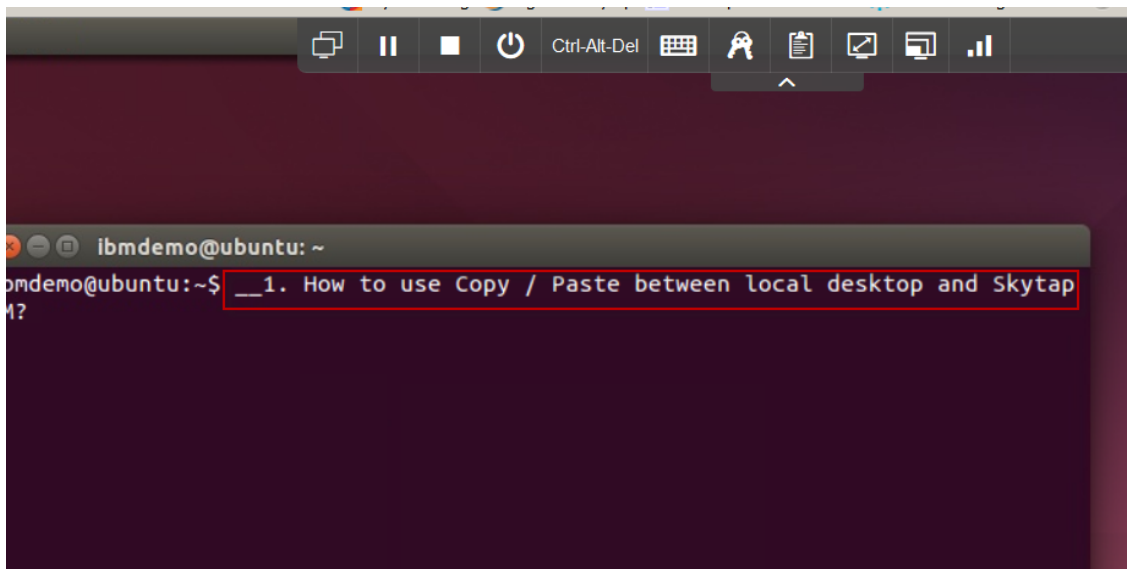
- \_\_c. Use **CTRL-V** to paste the content into the Copy/paste VM clipboard. Or use the **paste** menu item that is available in the dialog, when you right mouse click in the clipboard text area.



- \_\_d. Once the text is pasted, just navigate away to the VM window where you want to paste the content. Then, use **CTRL-C**, or right mouse click & us the **paste menu item** to paste the content.



\_\_e. The text is pasted into the VM



**Note:** The very first time you do this, if the text does not paste, you may have to paste the contents into the Skytap clipboard twice. This is a known Skytap issue. It only happens on the 1<sup>st</sup> attempt to copy / paste into Skytap.