Lab 04 Application Deployment and Management with Operators

Contents

4.1 Introduction

This is "**Lab 04 - Liberty application deployment using Operators**" 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.

This lab assumes basic familiarity with Docker for building images, running containers, and employing Kubernetes to deploy applications and route application traffic. This lab will introduce Operators which are the preferred mechanism in Red Hat OpenShift Container Platform (RHOCP) for application packaging, deployment, and management.

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

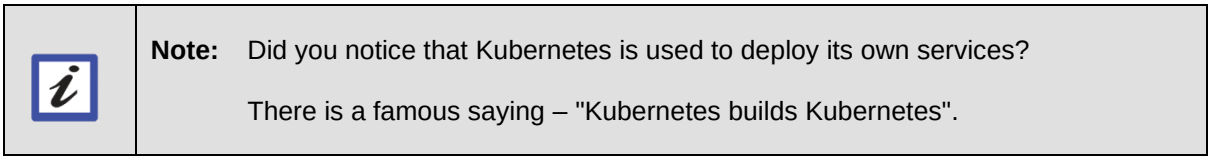
4.2 Operators

Operators in Kubernetes are extensions to the Kubernetes API implemented as an application specific controller for managing applications. Unlike the built-in Kubernetes controllers that are part of the cluster control plane, operators are deployed as applications.

Operator instances are defined by a Custom Resource (CR) which create an Operator deployment associated with a specific Custom Resource Definition (CRD). Stated another way, Operators are defined by CRDs and deployed using a CR, and in turn an Operator manages a CR application

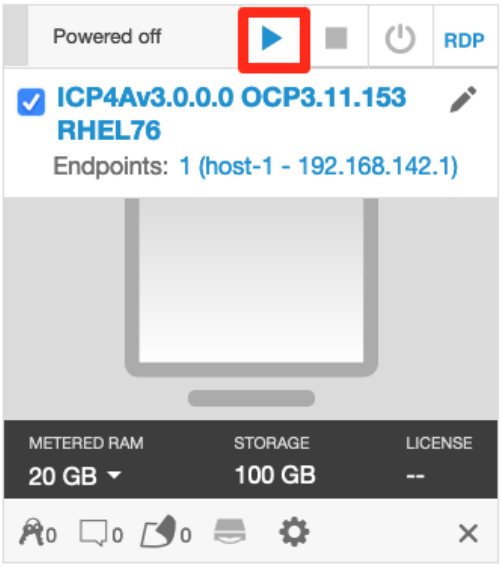
deployment which is defined by a CRD.

This use of CR instances defined by a CRD, the operator, to manage CR instances defined by a CRD, the application, is an example of Kubernetes being built with Kubernetes.

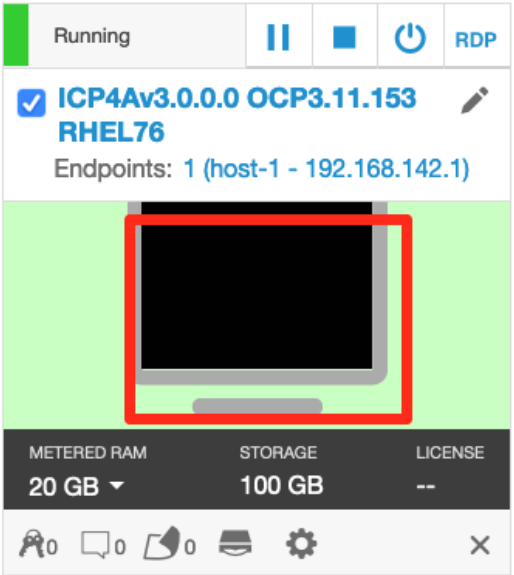


4.3 Let's get started

__1. On your laptop/workstation, locate the `ICP4Av3.0.0.0 OCP3.11.153 RHEL76` virtual machine. The VM should already be running. If not, Launch the Lab environment by clicking the **Run this VM** icon.

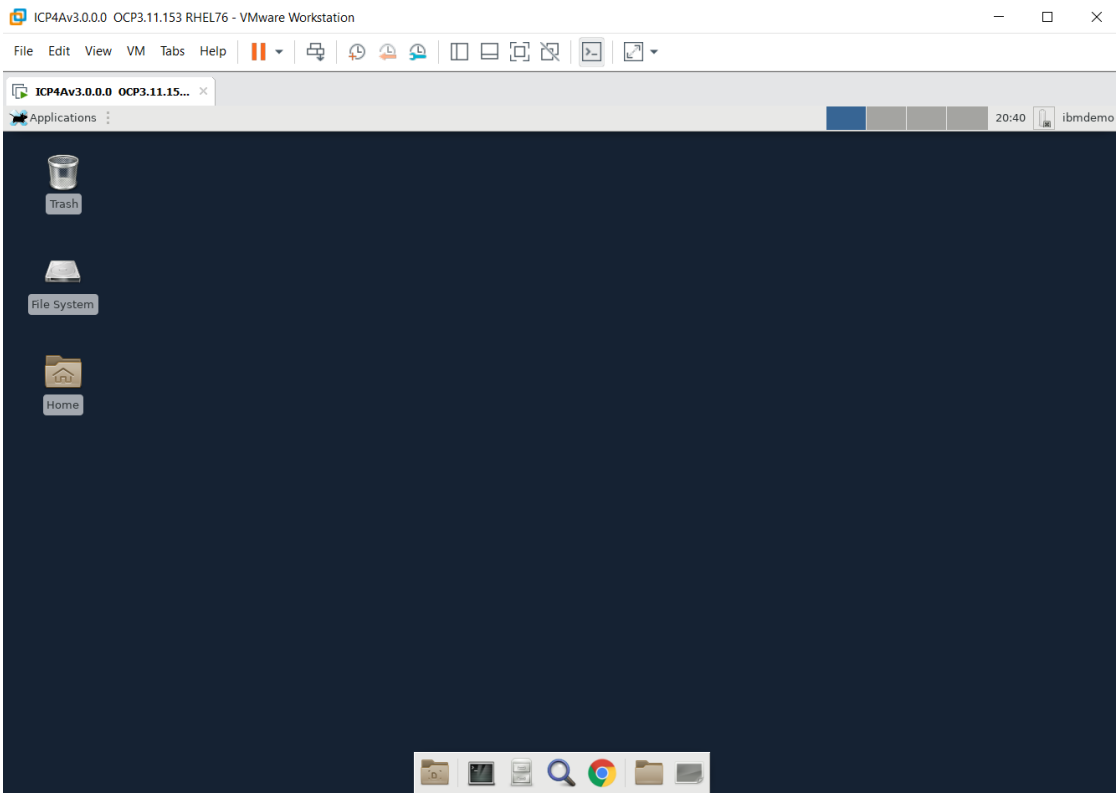


__2. After the VM is running, click its icon to access the VM's desktop.



__3. After the VM machine powers on, log with the `ibmdemo` user using the password `passw0rd`

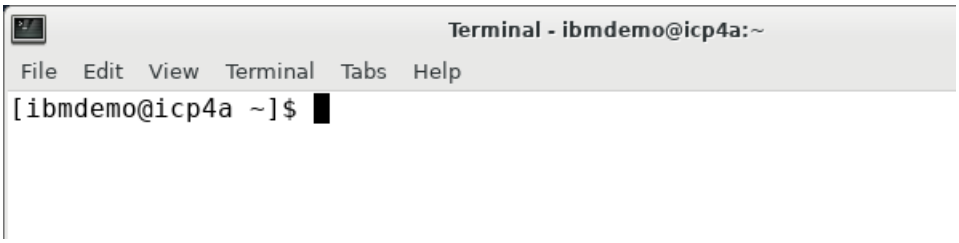
The `ICP4Av3.0.0.0 OCP3.11.153 RHEL76` virtual machine running and its Desktop is displayed in a web browser window.



	Note: Refer to the **Appendix** in this lab guide for details for using Copy / Paste between the lab guide and the lab environment.

__1. 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`



	**Note:** Please note that if needed root access can be obtained with `sudo su -`

__2. Create a Docker image.

_a. Type `cd student/lab1`

_b. Build a Liberty docker image named simpleapp by typing `docker build -t simpleapp .`

(note the "." at the end of the command, which will build an image named `simpleapp` using the Dockerfile in the local directory ".")

```
[ibmdemo@icp4a lab1]$ docker build -t simpleapp .

Sending build context to Docker daemon 11.78 kB
Step 1/7 : FROM docker.io/ibmcom/websphere-liberty:19.0.0.6-kernel-ubi-min
 ---> 7810d7fa4666
Step 2/7 : COPY server.xml /config/
 ---> Using cache
 ---> 6edb30c0af77
Step 3/7 : COPY ServletApp.war /config/apps/
 ---> a3723ec150d9
Removing intermediate container 07c7632f0288
Step 4/7 : USER root
 ---> Running in e13bdf4044d8
 ---> 187f8d0995ed
Removing intermediate container e13bdf4044d8
```

```
Step 5/7 : RUN chown default:root -R /opt/ibm/wlp/usr/servers/defaultServer
 ---> Running in 94ee6f6f54b4

 ---> 44b1984ac0c4
Removing intermediate container 94ee6f6f54b4
Step 6/7 : USER 1001
 ---> Running in b8131fa83b3e
 ---> 78589551606b
Removing intermediate container b8131fa83b3e
Step 7/7 : RUN configure.sh
 ---> Running in 5cb8b3f4a436

+ 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 ']'
+ '[' '' == client ']'
+ '[' '' == embedded ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ installUtility install --acceptLicense defaultServer
Checking for missing features required by the server ...
The server requires the following additional features: servlet-3.1.  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 4: Downloading servlet-3.1 ...
Step 2 of 4: Installing servlet-3.1 ...
Step 3 of 4: Validating installed fixes ...
Step 4 of 4: 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 103.
+ /opt/ibm/wlp/bin/server stop

Stopping server defaultServer.
Server defaultServer stopped.
+ rm -rf /output/resources/security/ /output/messaging /logs/console.log /logs/messages.log
/logs/messages_19.12.04_20.44.34.0.log /opt/ibm/wlp/output/.classCache
+ chmod -R g+rwx /opt/ibm/wlp/output/defaultServer
+ find /opt/ibm/wlp -type d -perm -g=x -print0
+ xargs -0 -r chmod -R g+rwx
 ---> 59258a04abcf
Removing intermediate container 5cb8b3f4a436
Successfully built 59258a04abcf
```

__3. Type `cd ~/student/lab4`

```
[ibmdemo@icp4a student]$ cd ~/student/lab4
[ibmdemo@icp4a lab4]$ $
```

__4. Login to OpenShift

_a. Type `oc login` and then enter `ocpadmin` for the username and `ocpadm1n` (note the “1”, not “i”) for the password

```
ibmdemo@icp4a lab4]$ 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
    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".
```

This lab will use a new OpenShift project. An **OpenShift project** is a Kubernetes namespace with some additional annotations which set the scope for the Objects, such as pods, services, replication controllers, etc.;

Policies which are rules for the allowed actions; Constraints (or quotas) for each kind of object, as well as Service Accounts for the project.

__5. Type `oc new-project lab4` which will create the lab4 project and switch your context to that project

```
[ibmdemo@icp4a lab4]$ oc new-project lab4

Now using project "lab4" on server "https://icp4a.pot.com:8443".
```

__6. Before starting to review and modify the operator artifacts, run the following commands to tag and push the simpleapp Docker image used with this lab to the local RHOCP image registry

```
docker tag simpleapp:latest docker-registry.default.svc:5000/lab4/simpleapp:latest

docker login -u $(oc whoami) -p $(oc whoami -t) docker-registry.default.svc:5000

docker push docker-registry.default.svc:5000/lab4/simpleapp:latest
```

```
[ibmdemo@icp4a lab4]$ docker tag simpleapp:latest docker-registry.default.svc:5000/lab4/simpleapp:latest

[ibmdemo@icp4a lab4]$ docker login -u $(oc whoami) -p $(oc whoami -t) docker-registry.default.svc:5000

Login Succeeded

[ibmdemo@icp4a lab4]$ docker push docker-registry.default.svc:5000/lab4/simpleapp:latest

The push refers to a repository [docker-registry.default.svc:5000/lab4/simpleapp]
08bc8e08e9a0: Layer already exists
fa160ac04f3f: Layer already exists
fd222008331e: Layer already exists
a052c31f2baf: Layer already exists
b78712e11f32: Layer already exists
0c65517b3677: Layer already exists
c6d9c7cf5338: Layer already exists
be36d206af93: Layer already exists
ba04059ad9a3: Layer already exists
71532d3a56e4: Layer already exists
790bcf471d32: Layer already exists
fe274995fb89: Layer already exists
9649117d0875: Layer already exists
9e19e22c9a42: Layer already exists
e9417d2583e6: Layer already exists
481324a7ba6d: Layer already exists
26429bebe019: Layer already exists
latest: digest: sha256:154e90f0a854f5c0337a24174265406c7d64241e3472780bb055ab3885129276 size: 3874
[ibmdemo@icp4a lab4$
```

Note: If your results do not match those above, you need to check to see if you are working in the right OpenShift project, it should be lab4

The **docker tag** command tags the docker image for the RHOCP registry

The **docker login** command logs you into the OpenShift internal registry, using the OpenShift username and password that you are currently logged in OpenShift.

The **docker push** command pushes the docker image to the OpenShift internal registry

__7. Type `ls` to review the contents of this directory.

```
[ibmdemo@icp4a lab4]$ ls

01-createnfspv.yaml  02-createpvc.yaml  99-cleanUpLab4.sh  operator
```

__8. Create a Persistent Volume and Persistent Volume Claim

This lab will use both a Kubernetes **PersistentVolume** and a Kubernetes **PersistentVolumeClaim** employing NFS.

A **PersistentVolume** (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned.

A **PersistentVolumeClaim** (PVC) is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources.

Two yaml files `01-createnfspv.yaml` and `02-createpvc.yaml` have been provided to provision these two Kubernetes objects. Review these yaml files to be familiar with their contents

_a. Type kubectl apply -f `01-createnfspv.yaml` to create the persistent volumes

(note the use of `kubectl`, **not** `oc this because` pv's are allocated for the cluster, not a project)

```
[ibmdemo@icp4a lab4]$ kubectl apply -f  01-createnfspv.yaml
persistentvolume/vol01 created
persistentvolume/vol02 created
persistentvolume/vol03 created
persistentvolume/vol04 created
persistentvolume/vol05 created
persistentvolume/vol06 created
persistentvolume/vol07 created
persistentvolume/vol08 created
persistentvolume/vol09 created
persistentvolume/vol10 created
persistentvolume/vol11 created
persistentvolume/vol12 created
persistentvolume/vol13 created
persistentvolume/vol14 created
persistentvolume/vol15 created
persistentvolume/vol16 created
persistentvolume/vol17 created
persistentvolume/vol18 created
persistentvolume/vol19 created
persistentvolume/vol20 created
[ibmdemo@icp4a lab4]$
```

_b. Type `oc apply -f 02-createpvc.yaml` to create the persistent volume claim

(note the use of `oc` to create the persistent volume claim scoped to the project, do **not use** `kubectl`)

```
[ibmdemo@icp4a lab4]$ oc  apply -f  02-createpvc.yaml
persistentvolumeclaim/simpleapp-serviceability created
[ibmdemo@icp4a lab4]$
```

4.4 Deployment using Operators

__1. Type `ls`

```
[ibmdemo@icp4a lab4]$ ls
01-createnfspv.yaml  02-createpvc.yaml  99-cleanUpLab4.sh  operator
```

In addition to the yaml files and a script (.sh) file, there is a single directory `operator` which is the Liberty Operator that we will use to deploy Liberty application in this lab.

The Liberty Operator is included in IBM Cloud Pak for Applications offering and is also available in the public Operator Hub.

__2. Type `cd operator` then type `ls` There are two directories; `application` and `deploy`

```
[ibmdemo@icp4a lab4]$ cd operator

[ibmdemo@icp4a operator]$ ls
application  deploy
```

__3. We'll start by reviewing the Operator file in the `application` directory,

_a. type `cd application` then type `ls`

```
[ibmdemo@icp4a operator]$ cd application
[ibmdemo@icp4a application]$ ls
application-cr.yaml
openliberty.io_openlibertydumps_cr.yaml
openliberty.io_openlibertyapplications_crd.yaml
openliberty.io_openlibertytraces_crd.yaml
openliberty.io_openlibertyapplications_cr.yaml
openliberty.io_openlibertytraces_cr.yaml
openliberty.io_openlibertydumps_crd.yaml
```

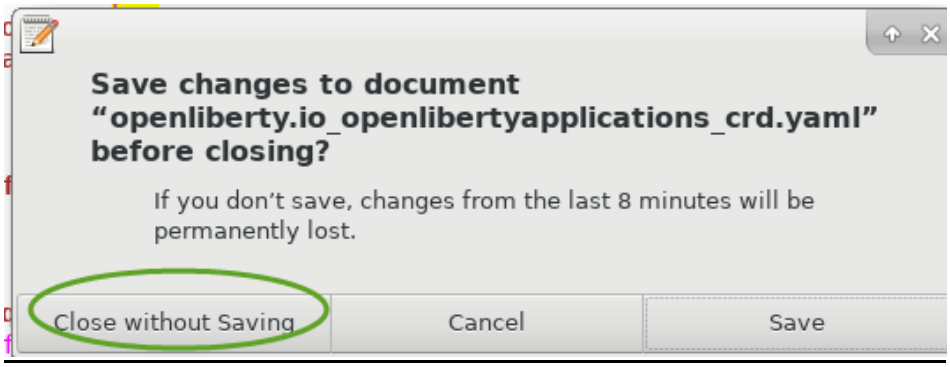
This directory has several artifacts, some are **Custom Resource Definition** (CRD) files, as denoted by the `crd` in the name, the others are **Custom Resource** (CR) files as denoted by the `cr` in the name.

__4. Let's start by reviewing the CRD files, first the application CRD

_a. Type `gedit openliberty.io_openlibertyapplications_crd.yaml`

This file defines all the attributes associated this CRD; pods, containers, environmental variables, services, routes, etc. Scroll down to review the file, then click the `x` in the upper right-hand corner to close the file when you are finished reviewing the file.

If you've accidently made a change to the file, you'll be prompted with the following warning



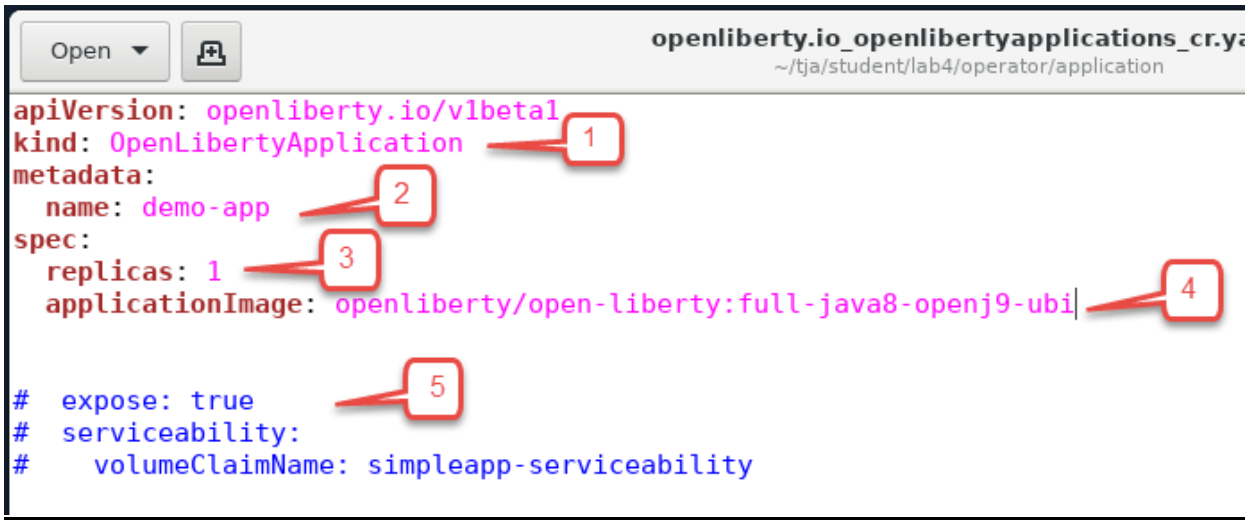
Click `Close without Saving` to exit since we don't want to make changes to this file.

__5. Review the other 2 CRD files using the "gedit" editor as you did in the previous step

_a. `openliberty.io_openlibertydumps_crd.yaml`

_b. `openliberty.io_openlibertytraces_crd.yaml`

__6. Now let's look at the CR files, first the application CR by typing `gedit openliberty.io_openlibertyapplications_cr.yaml`



1. Creates a CR instance `kind` of `OpenLibertyApplicatio`n (defined in the CRD)
2. Assigns the `name` of `demo-app` to the CR (optional)
3. Specifies the number of replicas in the deployment (optional)
4. Specifies the name of the image to be used to create the application (required)
5. Also included, but commented out, are some additional optional attributes which will be used later in the lab

Note" Only the `kind` parameter value and the `applicationImage` parameter value are required to deploy an `OpenLibertyApplication` CR instance, the rest are optional.

KEEP THE EDITOR OPEN. You will modify the contents of the CR in the next step.

__7. We're going to modify this CR for the lab, so that the Kubernetes deployment resources associated with this CR have a readily recognizable name. Make the following changes to the CR

_a. As shown below, change the `name` value (1) from `demo-app` to `simpleapp`

_b. As shown below, change the `applicationImage` parameter (2) from `openliberty/open-liberty:full-java8-openj9-ubi` to docker-registry.default.svc:5000/lab4/simpleapp:latest (this is image tagged and pushed previously in this lab)

_c. Click `Save`

_d. Click the `x` to close the file

__8. At this point, we can create the Custom Resource Definitions (CRDs) associated with the OpenLibertyOperator.

_a. type the following commands to create the Liberty Operator Custom Resource Definitions.

```
oc apply -f openliberty.io_openlibertyapplications_crd.yaml
oc apply -f openliberty.io_openlibertydumps_crd.yaml
oc apply -f openliberty.io_openlibertytraces_crd.yaml
```

```
[ibmdemo@icp4a application]$ oc apply -f
openliberty.io_openlibertyapplications_crd.yaml
customresourcedefinition.apiextensions.k8s.io/openlibertyapplications.openliberty.io
created
[ibmdemo@icp4a application]$ oc apply -f openliberty.io_openlibertydumps_crd.yaml
customresourcedefinition.apiextensions.k8s.io/openlibertydumps.openliberty.io created
[ibmdemo@icp4a application]$ oc apply -f openliberty.io_openlibertytraces_crd.yaml
customresourcedefinition.apiextensions.k8s.io/openlibertytraces.openliberty.io created
[ibmdemo@icp4a application]$
```

Creation of the CRDs for the OpenLibertyOperator only needs to be performed once per cluster

__9. List the new OpenLiberty CRDs in the cluster

oc get crd | grep openliberty

```
[ibmdemo@icp4a application]$ oc get crd | grep openliberty

openlibertyapplications.openliberty.io              2020-07-23T19:09:57Z
openlibertydumps.openliberty.io                     2020-07-23T19:10:11Z
openlibertytraces.openliberty.io                    2020-07-23T19:10:29Z
[ibmdemo@icp4a application]$
```

__10. Next, you will deploy the OpenLibertyOperator

_a. Type `cd ../deploy` then type `ls`

```
[ibmdemo@icp4a application]$ cd ../deploy

[ibmdemo@icp4a deploy]$ ls
operator.yaml  role_binding.yaml  role.yaml  service_account.yaml
```

The yaml files in this directory are used to deploy the operator and create security resources used by the operator.

Before deploying them, you will modify them so that the operator and security resources have a readily recognizable name associated with the application

__11. Edit the operator Deployment by typing `gedit operator.yaml`

_a. Change the 5 occurrences indicated below of `open-liberty-operator` to `simpleapp-operator`

_b. Click `Save` (in the upper right corner)

_c. Click the `x` (in the upper right corner) to close the file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: open-liberty-operator
spec:
  replicas: 1
  selector:
    matchLabels:
      name: open-liberty-operator
  template:
    metadata:
      labels:
        name: open-liberty-operator
    spec:
      serviceAccountName: open-liberty-operator
      containers:
        - name: open-liberty-operator
          image: openliberty/operator:0.3.0
          command:
          - open-liberty-operator
          imagePullPolicy: Always
          env:
            - name: WATCH_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: OPERATOR_NAME
              value: "open-liberty-operator"
```

change "open-liberty-operator" to "simpleapp-operator"

do not change

do not change

do not change

__12. Edit the **RoleBinding** resource by typing `gedit role_binding.yaml`

_a. Change the 3 occurrences indicated below of `open-liberty-operator` to `simpleapp-operator`

_b. Click `Save` (in the upper right corner)

_c. Click the `x` (in the upper right corner) to close the file

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: open-liberty-operator
subjects:
- kind: ServiceAccount
  name: open-liberty-operator
roleRef:
  kind: Role
  name: open-liberty-operator
  apiGroup: rbac.authorization.k8s.io
```

change "open-liberty-operator" to "simpleapp-operator"

__13. Edit the Role resource by typing `gedit role.yaml`

_a. Change the 2 occurrences indicated below of `open-liberty-operator` to `simpleapp-operator` (you'll need to scroll down to find the 2nd one)

_b. Click `Save` (in the upper right corner)

_c. Click the `x` (in the upper right corner) to close the file

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  creationTimestamp: null
  name: open-liberty-operator
rules:
- apiGroups:
```

```
- apiGroups:
  - apps
  resourceNames:
  - open-liberty-operator
  resources:
  - deployments/finalizers
  verbs:
  - update
- apiGroups:
```

__14. Edit the **ServiceAccount** resource by typing `gedit service_account.yaml`

_a. Change the occurrence indicated below of `openliberty-operator` to `simpleapp-operator`

_b. Click `Save` (in the upper right corner)

_c. Click the `x` (in the upper right corner) to close the file

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: open-liberty-operator
```

__15. Deploy the **ServiceAccount**, **Role** and **RoleBinding** resources, using the following commands:

```
oc apply -f service_account.yaml
oc apply -f role.yaml
oc apply -f role_binding.yaml
```

```
[ibmdemo@icp4a deploy]$ oc apply -f service_account.yaml
serviceaccount/simpleapp created

[ibmdemo@icp4a deploy]$ oc apply -f role.yaml
role.rbac.authorization.k8s.io/simpleapp-operator created

[ibmdemo@icp4a deploy]$ oc apply -f role_binding.yaml
rolebinding.rbac.authorization.k8s.io/simpleapp-operator created
```

__16. Now deploy the operator by typing `oc apply -f operator.yaml` then type `oc get pods` every few seconds until the operator pod is Running and Ready 1/1 (as shown below)

```
[ibmdemo@icp4a deploy]$ oc apply -f operator.yaml
deployment.apps/simpleapp-operator created

[ibmdemo@icp4a deploy]$ oc get pods
NAME                                 READY     STATUS              RESTARTS   AGE
simpleapp-operator-5d7446f446-5sx9h   0/1       ContainerCreating   0          8s

[ibmdemo@icp4a deploy]$ oc get pods
NAME                                 READY     STATUS    RESTARTS   AGE
simpleapp-operator-5d7446f446-5sx9h   1/1       Running   0          15s
```

If your results do not match those above, you need to check to see if you are working in the right OpenShift project, it should be **lab4**.

__17. Type `oc get all` which will list all the resources in the project, note that the deploying the operator resulted in a **pod**, **deployment**, and **replicaset** being created

```
[ibmdemo@icp4a deploy]$ oc get all
NAME                                        READY     STATUS    RESTARTS   AGE
pod/simpleapp-operator-5d7446f446-5sx9h     1/1       Running   0          1m

NAME                                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/simpleapp-operator    1         1         1            1           1m

NAME                                             DESIRED   CURRENT   READY     AGE
replicaset.apps/simpleapp-operator-5d7446f446    1         1         1         1m

NAME                                          DOCKER REPO
TAGS      UPDATED
imagestream.image.openshift.io/simpleapp      docker-
registry.default.svc:5000/lab4/simpleapp      latest    16 hours ago

NAME                                                                  READY     REASON
AGE
clusterchannelprovisioner.eventing.knative.dev/in-memory              True
2d
clusterchannelprovisioner.eventing.knative.dev/in-memory-channel      True
2d

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

__18. With the operator running, now deploy the CR using the file modified earlier to provide explicit reference to the simpleapp application

```
oc apply -f
../application/openliberty.io_openlibertyapplications_cr.yaml
```

```
[ibmdemo@icp4a deploy]$ oc apply -f
../application/openliberty.io_openlibertyapplications_cr.yaml

openlibertyapplication.openliberty.io/simpleapp created
[ibmdemo@icp4a deploy]$
```

__19. Type `oc get pods` to check the status of the **application** pod. While this pod should start very quickly, you may need to repeat this command a couple of times until the pod is Running and **Ready 1/1** (as shown below)

```
[ibmdemo@icp4a deploy]$ oc get pods
NAME                                  READY     STATUS    RESTARTS   AGE
simpleapp-55ff8cbb7-snmnj             1/1       Running   0          3s
simpleapp-operator-5d7446f446-jdxng   1/1       Running   0          4m
```

__20. Type `oc get all` which will list all the resources in the project

Note that the in addition to the **operator** pod, deployment, and replicaset, there's now a pod, deployment and replicaset for the **simpleapp** application, as well as a ClusterIP service and a route for simpleapp.

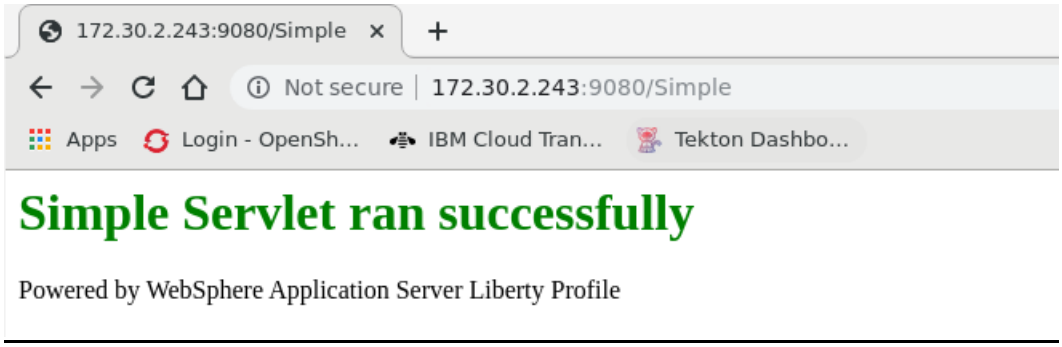
```
[ibmdemo@icp4a deploy]$ oc get all
NAME                                      READY     STATUS    RESTARTS   AGE
pod/simpleapp-55ff8cbb7-snmnj             1/1       Running   0          2m
pod/simpleapp-operator-5d7446f446-jdxng   1/1       Running   0          4m

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
service/simpleapp    ClusterIP   172.30.2.243   <none>        9080/TCP   2m

NAME                                    DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/simpleapp               1         1         1            1           2m
deployment.apps/simpleapp-operator      1         1         1            1           4m

NAME                                               DESIRED   CURRENT   READY     AGE
replicaset.apps/simpleapp-55ff8cbb7                1         1         1         2m
replicaset.apps/simpleapp-operator-5d7446f446      1         1         1         2m

NAME                                          DOCKER REPO
TAGS      UPDATED
imagestream.image.openshift.io/simpleapp      docker-
registry.default.svc:5000/lab4/simpleapp      latest    20 minutes ago

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

__21. You can access the application using the **ClusterIP**

_a. Open the Chrome browser and using the **ClusterIP** for your deployment construct the following URL http://<ClusterIP>:9080/Simple

in the example above, this is http://172.30.2.243:9080/Simple

Note: the ClusterIP in your environment will be different



__22. Now let's change the number of replicas to scale the application. Then expose the application to the outside world via a Route. You will simply change the parameters in the CR for the application and apply the updates.

Type `gedit ../application/openliberty.io_openlibertyapplications_cr.yaml`

_a. Change the `replicas` from `1` to `2` (as indicated below)

_b. Remove the # prior to expose parameter

_c. Click `Save` (in the upper right corner)

_d. Click the `x` (in the upper right corner) to close the file

__23. Apply the changes to the OpenShift Cluster

_a. Type `oc apply -f ../application/openliberty.io_openlibertyapplications_cr.yaml`

_b. type `oc get pods` to see the following results

- New pods being created in the existing replicaset
- New pods being created in the new replicaset
- The pods in the existing repilcaset terminating

```
[ibmdemo@icp4a deploy]$ oc apply -f
../application/openliberty.io_openlibertyapplications_cr.yaml
openlibertyapplication.openliberty.io/simpleapp configured
[ibmdemo@icp4a deploy]$ oc get pods
NAME                                  READY     STATUS              RESTARTS   AGE
simpleapp-6bd779d769-fdq45            1/1       Running             0          11m
simpleapp-6bd779d769-m4fps            1/1       Terminating         0          6s
simpleapp-758689c4fd-46rzl            0/1       ContainerCreating   0          2s
simpleapp-758689c4fd-chc8z            1/1       Running             0          6s
simpleapp-operator-5d7446f446-2xs7z   1/1       Running             0          12m
[ibmdemo@icp4a deploy]$ oc get pods
NAME                                  READY     STATUS        RESTARTS   AGE
simpleapp-6bd779d769-m4fps            1/1       Terminating   0          28s
simpleapp-758689c4fd-46rzl            1/1       Running       0          24s
simpleapp-758689c4fd-chc8z            1/1       Running       0          28s
simpleapp-operator-5d7446f446-2xs7z   1/1       Running       0          12m
[ibmdemo@icp4a deploy]$ oc get pods
NAME                                  READY     STATUS    RESTARTS   AGE
simpleapp-758689c4fd-46rzl            1/1       Running   0          34s
simpleapp-758689c4fd-chc8z            1/1       Running   0          38s
simpleapp-operator-5d7446f446-2xs7z   1/1       Running   0          12m
```

__24. type `oc get rs` to see the following results of the two replicasets

```
[ibmdemo@icp4a deploy]$ oc get rs
NAME                            DESIRED   CURRENT   READY     AGE
simpleapp-6bd779d769            0         0         0         11m
simpleapp-758689c4fd            2         2         2         43s
simpleapp-operator-5d7446f446   1         1         1         12m
```

Now, let's view the and test the route that we created by setting expose: true in the CR.

__25. Type `oc get route` (or `oc get all` which will list the route along with the pods, etc)

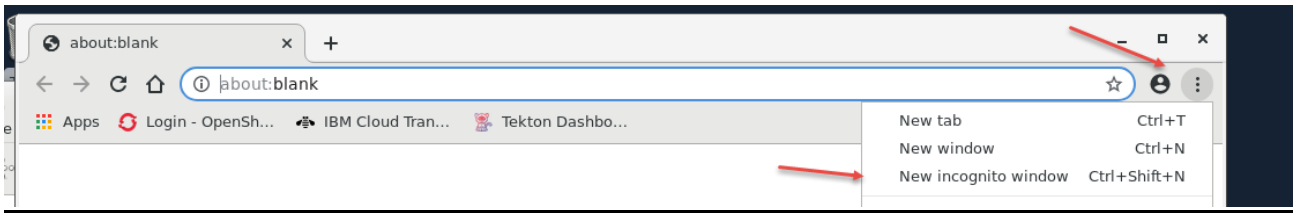
```
[ibmdemo@icp4a deploy]$ oc get route
NAME        HOST/PORT                              PATH      SERVICES    PORT
TERMINATION   WILDCARD
simpleapp   simpleapp-lab4.apps.icp4a.pot.com                 simpleapp   9080-tcp
None
```

__26.Construct the browser URL `<route>/Snoop`

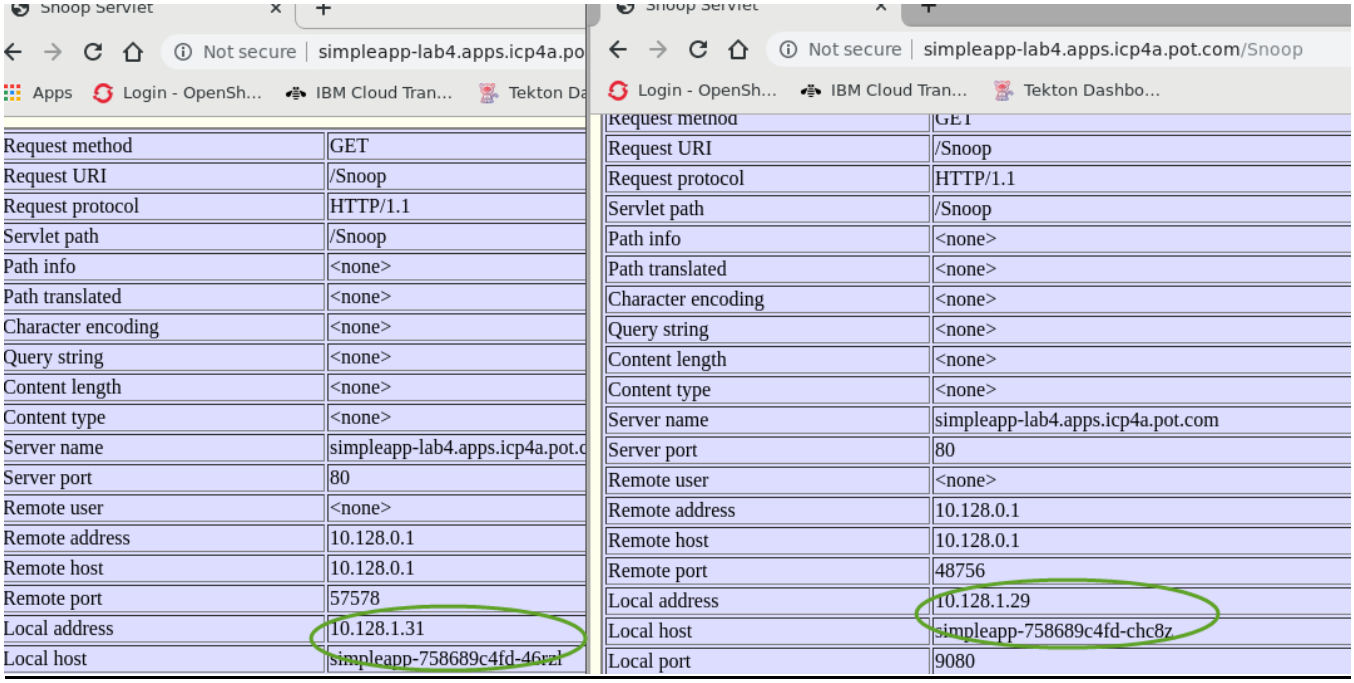
simpleapp-lab4.apps.icp4a.pot.com/Snoop based on the route shown above

Next, you will test the route to the application, and that traffic is load balanced between the two pods that are running the simpleapp application.

__27.Open one incognito window (this to avoid HTTP session pining requests to one pod)



__28. Enter the URL from above in both windows, note that the **local address** and **localhost** differ in each Snoop output. It corresponds to the **IP inside the pod** and **the pod** handing the request. The route provides a common path to all instances (you may need to reload on of the browsers a couple times to see different pods (local IP address handling the request)


Snoop Servlet
Not secure | simpleapp-lab4.apps.icp4a.po
Apps
Login - OpenSh...
IBM Cloud Tran...
Tekton Da
Request method GET
Request URI /Snoop
Request protocol HTTP/1.1
Servlet path /Snoop
Path info <none>
Path translated <none>
Character encoding <none>
Query string <none>
Content length <none>
Content type <none>
Server name simpleapp-lab4.apps.icp4a.pot.c
Server port 80
Remote user <none>
Remote address 10.128.0.1
Remote host 10.128.0.1
Remote port 57578
Local address 10.128.1.31
Local host simpleapp-758689c4fd-46rzl
Not secure | simpleapp-lab4.apps.icp4a.pot.com/Snoop
Login - OpenSh...
IBM Cloud Tran...
Tekton Dashbo...
Request method GET
Request URI /Snoop
Request protocol HTTP/1.1
Servlet path /Snoop
Path info <none>
Path translated <none>
Character encoding <none>
Query string <none>
Content length <none>
Content type <none>
Server name simpleapp-lab4.apps.icp4a.pot.com
Server port 80
Remote user <none>
Remote address 10.128.0.1
Remote host 10.128.0.1
Remote port 48756
Local address 10.128.1.29
Local host simpleapp-758689c4fd-chc8z
Local port 9080

4.5 Day Two Operations using Operators

__1. There are currently two “day two operations” provided by the Open Liberty Operator: **dumps** and **tracing**. Both a configured in a similar manner

_a. Add storage to the pods for serviceability (dumps and traces) by changing the CR for the application deployment

_b. Deploying a CR for tracing or dumps

Though this lab will only demonstrate tracing

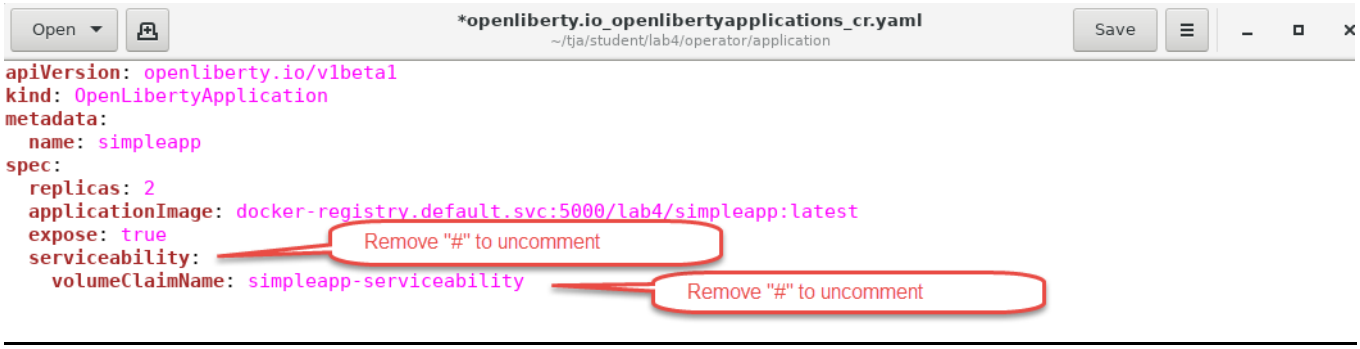
__2. Type `gedit ../application/openliberty.io_openlibertyapplications_cr.yaml`

_a. Remove the # prior to `serviceability` parameter

_b. Remove the # prior to the `volumeClaimName` parameter

_c. Click `Save` (in the upper right corner)

_d. Click the `x` (in the upper right corner) to close the file



__3. Type `oc apply -f ../application/openliberty.io_openlibertyapplications_cr.yaml`

to deploy the updated application deployment

__4. Type `oc get pods` to see the old pods terminating and new pods starting. waiting until the new pods are `Running` and `Ready`

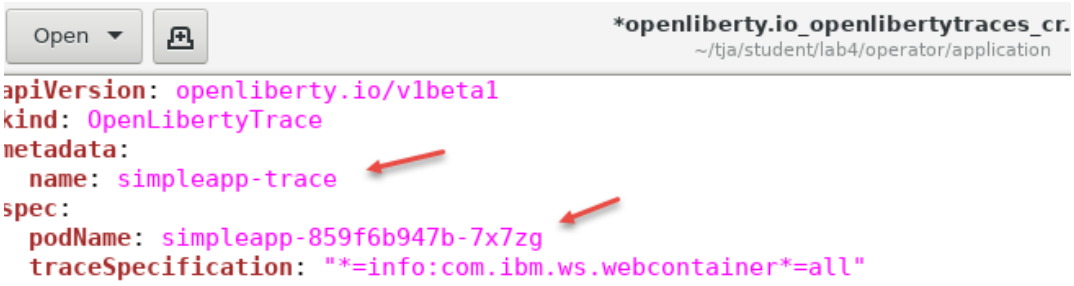
```
[ibmdemo@icp4a deploy]$ oc get pods
NAME                                READY     STATUS        RESTARTS   AGE
simpleapp-758689c4fd-chc8z          0/1       Terminating   0          9m
simpleapp-859f6b947b-7x7zg          1/1       Running       0          18s
simpleapp-859f6b947b-cl9rp          1/1       Running       0          14s
simpleapp-operator-5d7446f446-2xs7z 1/1       Running       0          1h
[ibmdemo@icp4a deploy]$ oc get pods
NAME                                READY     STATUS    RESTARTS   AGE
simpleapp-859f6b947b-7x7zg          1/1       Running   0          19s
simpleapp-859f6b947b-cl9rp          1/1       Running   0          15s
simpleapp-operator-5d7446f446-2xs7z 1/1       Running   0          1h
[ibmdemo@icp4a deploy]$
```

__5. Copy the name one of the pods from your list (e.g. `simpleapp-859f6b947b-7x7zg` as shown above)

__6. Type `gedit ../application/openliberty.io_openlibertytraces_cr.yaml`

_a. Replace `example-trace` with `simpleapp-trace`

_b. Replace `Specify_Pod_Name_Here` with the pod name copied from above (the pod name from your environment will differ)



_c. Click `Save` and `x`, both in the upper right, to close gedit

__7. Type `oc apply -f ../application/openliberty.io_openlibertytraces_cr.yaml` to deploy the updated configuration for tracing.

```
[[ibmdemo@icp4a deploy]$ oc apply -f
../application/openliberty.io_openlibertytraces_cr.yaml

openlibertytrace.openliberty.io/simpleapp-trace created
```

__8. Type `oc get oltrace` (oltrace is a abbreviation for openlibertytrace.openliberty.io) to insure that the trace is enabled (`True`)

```
[ibmdemo@icp4a deploy]$ oc get oltrace
NAME              PODNAME                       TRACING
simpleapp-trace   simpleapp-859f6b947b-7x7zg    True
```

__9. Type `oc get pv` to see which NFS shared storage directory is being used for the trace output by the persistent volume claim.

In the output below the pv `vol 12` has been bound `to simpleapp-serviceability.` This means that the output from the trace located in files system location `/share/share12/lab4/simpleapp-serviceability.` There are 20 directories, `share01` through `share20` under `/share`

```
[ibmdemo@icp4a deploy]$ oc get pv
<list abbreviated>
vol11             20Gi       RWO            Recycle          Available
pot                      1d
vol12             20Gi       RWO            Recycle          Bound       lab4/simpleapp-
serviceability                    pot                      1d
vol13             20Gi       RWO            Recycle          Available
pot
<list abbreviated>
```

__10. Open a new terminal window, type `cd /share/share<vol#>/lab4/`

Example: `cd /share/share12/lab4/` in the case above, the volume will likely differ for you)

__11. Type `ls` to list the contents of the directory

```
[ibmdemo@icp4a ~]$ cd /share/share12/lab4

[ibmdemo@icp4a lab4]$ ls
simpleapp-859f6b947b-7x7zg
```

__12. A directory is created for every pod with an Open Liberty Trace `simpleapp-859f6b947b-7x7zg` in this example (your pod number will differ as previously noted)

__13. Type `cd <your pod number>` then type `ls`

```
[ibmdemo@icp4a lab4]$ cd simpleapp-859f6b947b-7x7zg/
[ibmdemo@icp4a simpleapp-859f6b947b-7x7zg]$ ls
messages.log  trace.log
```

Two log files are output to this directory: `messages.log` and `trace.log`

__14. The files can be examined using `cat, view` or `tail` or other tools, but since the files are owned by root, you'll need to use the `sudo` command e.g `oc`

Note: sudo password: passw0rd

```
[ibmdemo@icp4a simpleapp-859f6b947b-7x7zg]$ sudo tail -f trace.log
```

__15. The output from the `tail` command above will look like the following, varying slightly based on how quickly you access the trace.log

```
3/14/20 20:36:33:811 GMT] 0000003c id=00000000
m.ws.webcontainer.collaborator.WebAppTransactionCollaborator < postInvoke RETURN null
[3/14/20 20:36:33:812 GMT] 0000003c id=00000000 com.ibm.ws.webcontainer.webapp.WebApp
1 finishEnvSetup exit
[3/14/20 20:39:11:816 GMT] 0000004e id=00000000 com.ibm.ws.webcontainer.webapp.WebApp
1 startEnvSetup enter
[3/14/20 20:39:11:830 GMT] 0000004e id=00000000
m.ws.webcontainer.collaborator.WebAppTransactionCollaborator > preInvoke ENTRY null
true
[3/14/20 20:39:11:833 GMT] 0000004e id=00000000
m.ws.webcontainer.collaborator.WebAppTransactionCollaborator < preInvoke RETURN null
[3/14/20 20:39:11:836 GMT] 0000004e id=00000000 com.ibm.ws.webcontainer.webapp.WebApp
1 startEnvSetup exit
[3/14/20 20:39:11:841 GMT] 0000004e id=00000000 com.ibm.ws.webcontainer.webapp.WebApp
1 finishEnvSetup enter
[3/14/20 20:39:11:843 GMT] 0000004e id=00000000
m.ws.webcontainer.collaborator.WebAppTransactionCollaborator > postInvoke ENTRY null
null true
[3/14/20 20:39:11:845 GMT] 0000004e id=00000000
m.ws.webcontainer.collaborator.WebAppTransactionCollaborator < postInvoke RETURN null
[3/14/20 20:39:11:847 GMT] 0000004e id=00000000 com.ibm.ws.webcontainer.webapp.WebApp
1 finishEnvSetup exit
[3/14/20 20:41:49:849 GMT] 00000056 id=00000000 com.ibm.ws.webcontainer.webapp.WebApp
1 startEnvSetup enter
```

To generate trace based on application requests, application traffic needs to be directed to the pod that trace is enabled on.

Since there are two pods in the **simpleapp** deployment and trace was only enabled for one pod, using the **cluster address** for a pod is the quickest way to directly access a specific pod.

The **cluster address** is different than a **ClusterIP**. The ClusterIP is a Cluster service address for accessing all pods in a deployment. The **cluster address** is the internal address used to access a pod in a cluster, which is only accessible for a node in the cluster.

Fortunately, this PoT is using an All In One cluster, so we can access pods using the cluster address.

__16. To determine the **cluster ip address** for **the pod you enabled trace on**, run the following commands for the <pod-name> that you enabled trace on.

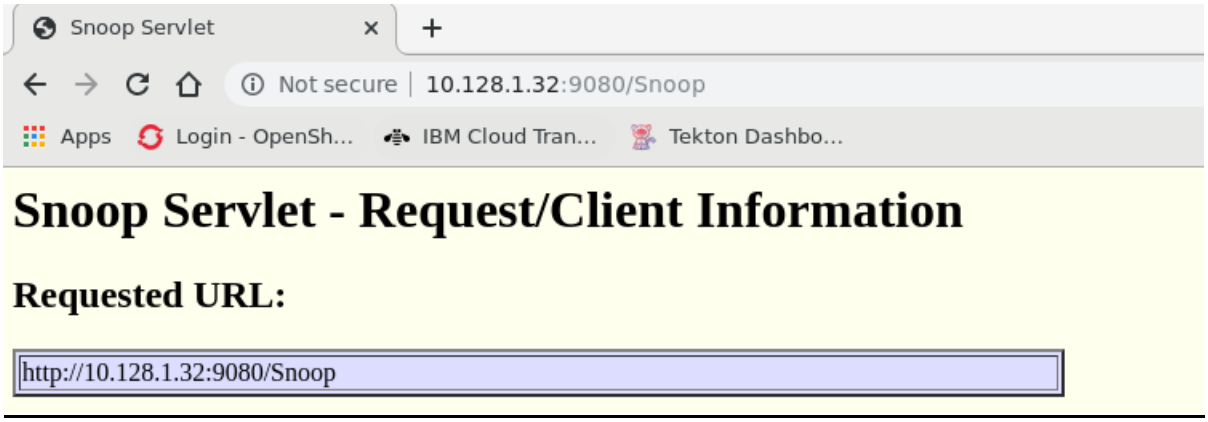
_a. Go back to the first terminal window where the OCP CLI is running and type `oc get pods`

_b. Type `oc describe pod <pod-name> | grep IP`

Note: substitute your pod-name for the one used here

```
[ibmdemo@icp4a share]$ oc describe pod simpleapp-859f6b947b-7x7zg | grep IP

IP:                 10.128.1.32
```

Enter the following URL in the browser `<cluster address>:9080/Snoop` in the case of the IP address above the URL is `10.128.1.32:9080/Snoop` (Your IP will likely differ).



__17. Refresh the request several times in the browser will monitoring the trace output in the second terminal window where `tail -f trace.log` is running trace output. Output as shown will scroll by as the requests are processed.

NOTE it may take several seconds for the request trace output to be written from the pod to the log on the file system.

```
handleRequest complete for--> [/Snoop], mapped webApp context
[com.ibm.ws.webcontainer31.osgi.webapp.WebApp31@551251ea[ServletApp#ServletApp.war
```

__18. Enter `Ctl+C` in the command shell where `tail -f traces.log` is running to exit `tail`

```
[3/14/20 20:51:19:314 GMT] 000000b8 id=00000000 com.ibm.ws.webcontainer.webapp.WebApp
1 finishEnvSetup exit
^C
[ibmdemo@icp4a simpleapp-859f6b947b-7x7zg]$
```

__19. Perform the following steps once you're finished

_a. Close all browser windows

_b. In an open command shell type `cd ~/student/lab4`

_c. In the command shell used in step "b" above, type `./99-cleanUpLab4.sh`

_d. Close all open command shells

```
[3/14/20 20:51:19:314 GMT] 000000b8 id=00000000 com.ibm.ws.webcontainer.webapp.WebApp
1 finishEnvSetup exit
^C
[ibmdemo@icp4a simpleapp-859f6b947b-7x7zg]$
```

4.6 Conclusion

You have now seen how deploy an application using an operator, as well as how an operator can be employed to perform "day 2" administration tasks for the application deployment.

End of Lab 04 – Liberty application deployment using Operators

Appendix: SkyTap Tips for labs

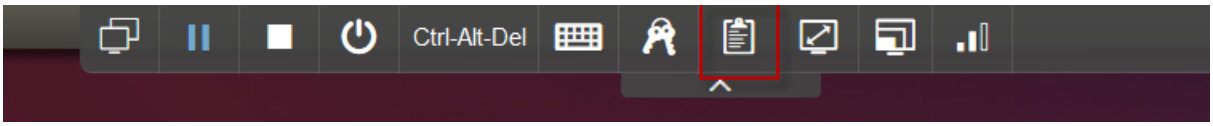
4.7 How to use Copy / Paste between local desktop and Skytap VM

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.

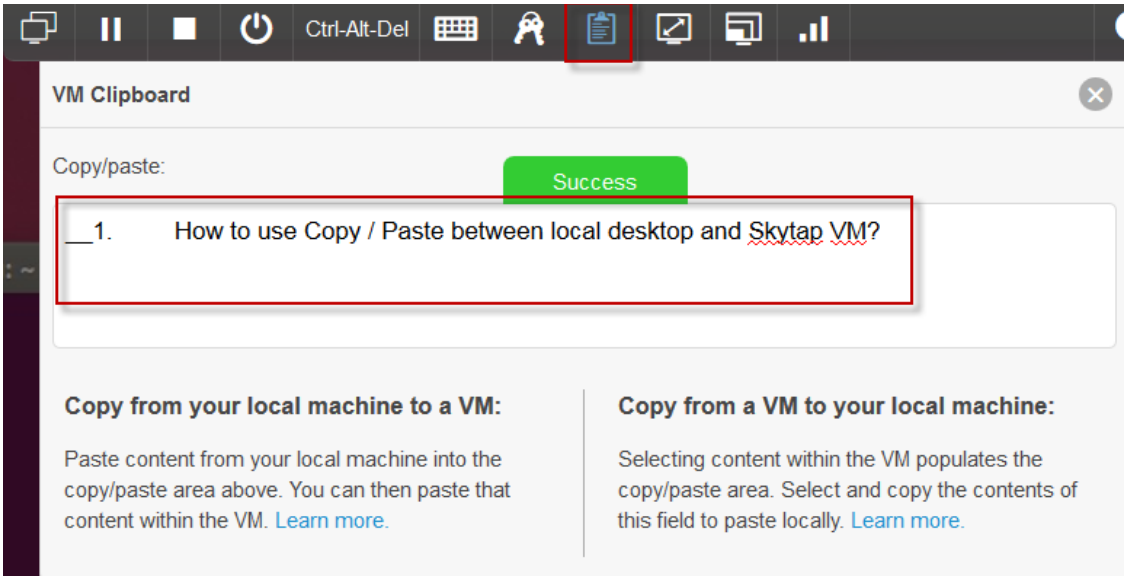
__1. In SkyTap, you will find that any text copied to the clipboard on your local workstation is not available to be pasted into the VM on SkyTap. So how can you easily accomplish this?

__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)

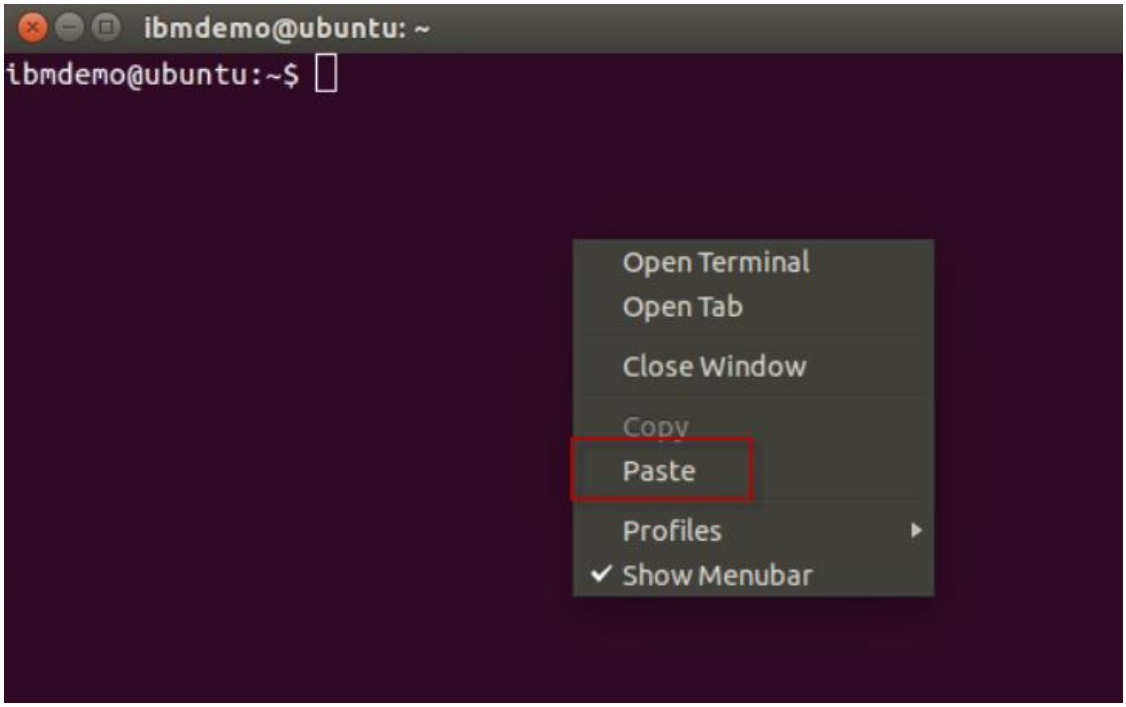
__b. Return to the SkyTap environment and click on the Clipboard at the top of the SkyTap session window.



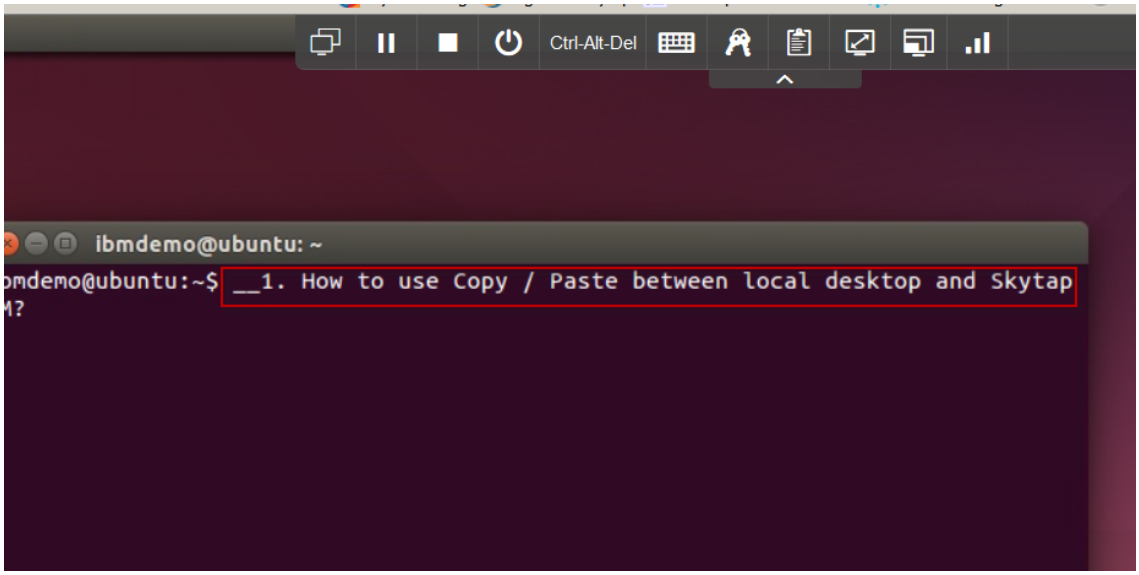
__c. Use **CTRL-V** to paste the content into the Copy/paste VM clipboard. Or use the **paste** menu item that is available in the dialog, when you right mouse click in the clipboard text area.



__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 1st attempt to copy / paste into Skytap.