IBM Cloud
Introduction to Containers and Kubernetes with IBM Cloud Private (ICP)

Hands-on Workshop

Lab Guide

© Copyright IBM Corp. 2018. All rights reserved

Notices and Disclaimers

© Copyright IBM Corporation 2018.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

This document is current as of the initial date of publication and may be changed by IBM at any time. Not all offerings are available in every country in which IBM operates.

IBM, the IBM logo and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at ibm.com/legal/copytrade.shtml

Other company, product and service names may be trademarks or service marks of others

Table of Contents

Section 1: Container Basics **4**
- **Section 1: Lab Instructions** **5**
- **Section 1: Lab Summary** **10**

Section 2: Data Persistence in Docker **11**
- **Section 2: Lab Instructions** **12**
- **Section 2: Lab Summary** **13**

Section 3: Getting Started with Kubernetes in IBM Cloud Private **14**
- **Section 3: Lab Instructions** **15**
- **Section 3: Lab Summary** **17**

Section 4: Deploy your Application to Kubernetes **18**
- **Section 4: Lab Instructions** **19**
- **Section 4: Lab Summary** **25**

Section 5: Observing Kubernetes Resiliency **26**
- **Section 5: Lab Instructions** **27**
- **Section 5: Lab Summary** **28**

Section 6: Deploying a Microservices Application in ICP **29**
- **Section 6: Lab Instructions** **31**
  - Build Containers 33
  - Kubernetes name service. 46
  - Cluster IP address 47
  - Host Names or Proxy Server 47
  - Ingress Service 47

© Copyright IBM Corp. 2018. All rights reserved

Section 1: Container Basics

Purpose:	For the first 2 sections, we will be using a sample application, a variation of the mobile game 2048. You will see how we create a Docker image from this application and run it as a container. This section introduces container basics. You will learn how to create, run, inspect and manage containers. Also, you will work through establishing console access within the container.

Tasks:	Tasks you will complete in this lab exercise include: • Connect to the Docker environment • Creating a Docker Image for an Application • Running containers • Inspecting containers • Container process monitoring • Container shell access

© Copyright IBM Corp. 2018. All rights reserved

Section 1: Lab Instructions

Step	Action
1	**<u>Login to the Docker Environment</u>** __ a. Your environment is on a cloud hosted Linux server. You can access this environment using the URL provided by your instructor. __ b. Once logged in, open a Gnome terminal window from the desktop. Next verify that docker is accessible by typing the following command: ~$ docker container run hello-world Verify that the output is similar to the following: 1. bash LouMacBookPro:~ louis$ docker container run hello-world Hello from Docker! This message shows that your installation appears to be working correctly. To generate this message, Docker took the following steps: 1. The Docker client contacted the Docker daemon. 2. The Docker daemon pulled the "hello-world" image from the Docker Hub. (amd64) 3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading. 4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal. To try something more ambitious, you can run an Ubuntu container with: $ docker run -it ubuntu bash Share images, automate workflows, and more with a free Docker ID: https://cloud.docker.com/ For more examples and ideas, visit: https://docs.docker.com/engine/userguide/ LouMacBookPro:~ louis$
2	**<u>Build a Docker Image for an Application</u>** __a. Before we can work with a container, we will need to first build an image for our 2048 application. First, we will make of copy of the application code to your home directory: ~$ cp -R /labs/2048_master . (don't forget the "." at the end) ~$ cd 2048_master __b. These files are the application code required to run the game. Notice there is a file called "Dockerfile" in the top directory of the unzipped files. The Dockerfile is the file you create that instructs Docker how to create and package the application into a Docker image. In this case, the file has already been created for you. Open the file and browse its contents. It will look similar to the figure below:

© Copyright IBM Corp. 2018. All rights reserved

Step	Action

Dockerfile

```
#FROM is the base image for which we will run our application
FROM nginx:latest

RUN ["apt-get","update"]
RUN ["apt-get","install","-y","vim"]

# Copy files and directories from the application
COPY index.html /usr/share/nginx/html
COPY favicon.ico /usr/share/nginx/html
COPY Rakefile /usr/share/nginx/html
COPY style/ /usr/share/nginx/html/style/
COPY meta/ /usr/share/nginx/html/meta/
COPY js/ /usr/share/nginx/html/js/

# Tell Docker we are going to use this port
EXPOSE 80
```

The commands in this file instruct Docker to use a simple web service (nginx) as a base image (nginx is automatically pulled from Docker Hub when the image is built. The file then copies the application code into a directory structure within the image (in /usr/share). Finally, port 80 is exposed in order to enable access to the game from our Web Browser.

__c. Now you can build the image by running the following command:

~ $ docker build -t 2048_image . *(don't forget the "." at the end)*

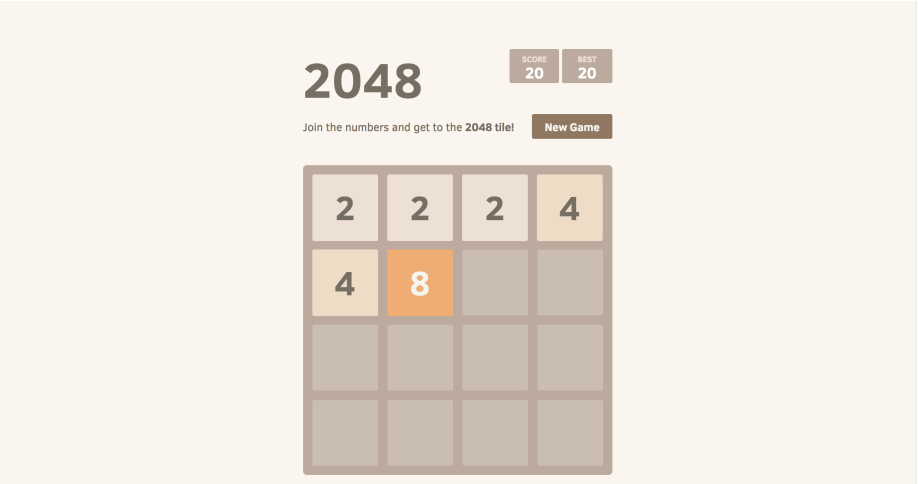
__d. Docker will now build the image. You can confirm this by running the following command and observing that an image named "2048_*image*" is listed:

~$ docker images

```
[user01@dlsol0129163851 2048_master]$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED              SIZE
user01_image        latest              56156c8f775e        About a minute ago   155MB
<none>              <none>              0f16eb39c0f6        3 hours ago          155MB
nginx               latest              3f8a4339aadd        5 weeks ago          108MB
hello-world         latest              f2a91732366c        2 months ago         1.85kB
```

You have now successfully taken an existing application and created a Docker image from it.

© Copyright IBM Corp. 2018. All rights reserved

Step	Action
3	**<u>Run a Container</u>** __a. Now that you have an image, we will now run the 2048 application as a container. To do this, run the following command: ***Your instructor will assign you a port a unique port number to use for the remained of the lab.*** ~$ docker container run --name *2048_container* -p *31005*:80 2048_*image* The container you just created is an instance of your image running as a process. There is no limit to the number of containers that can be run from an image. Commands: **--name** – Specify a unique name for the container service. If omitted Docker will create a random, human readable name. **-p** – Specify that the container internal port (80) be exposed to <your port> on the host. __b. Open a browser and navigate to: http://localhost:31005. A page will open with the game, as shown below:  You have now successfully run your first container!!
3	**<u>Stop/Delete a Container</u>** __ a. You can stop the container by typing cntrl-c ~$ <Cntrl-c>

© Copyright IBM Corp. 2018. All rights reserved

Step	Action
	__ b. Verify that the container is no longer running: ~$ docker container ps __ c. Although the container is not running it still exists: ~$ docker container ps -a [user01@dlsol0129163851 2048_master]$ docker ps -a CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES 6fec536a73eb user01_image "nginx -g 'daemon ..." About a minute ago Exited (0) 5 seconds ago user01_container -a, --all: Show all containers (default shows just running) __ d. Remove the container: ~$ docker container rm *2048_container* Containers can be removed either by their name or container id
4	**<u>Inspect a Running Container</u>** __a. Run a new Docker container for the game: ~$ docker run --publish <your port>:80 --detach --name 2048_container 2048_image You should be brought back to the terminal prompt (the “detach” option runs the container as a background process) __b. Open a browser and navigate to “TBD”. You should be prompted with the game again. __c. You can run a variety of commands to get information on the status of a running container. These commands. can be useful when troubleshoot an environment or application. For example, inspecting the meta-data for running container: ~$ docker container inspect 2048_container and, Stream live performance container metrics: ~$ docker container stats 2048_container __d. Clean up ~$ docker container rm -f 2048_container

© Copyright IBM Corp. 2018. All rights reserved

Step	Action
	Commands: **-d, --detach** - Run the container in the background.
5	**<u>Run Shell Inside a Container</u>** __a. We can also directly access a container via a command shell. It allows you to directly login to the container's command prompt; enabling you to troubleshoot application issues or update the content of a running container. First run the container again: ~$ docker container run --name 2048_container -d -p <your port>:80 2048_image __b. Next, we will use the following command to open a shell prompt into the container: ~$ docker exec -it 2048_container bash __c. Run Linux commands in container: For example, # ls -tal // List directories and files. # exit // Exit shell __d. Delete the container: ~$ docker rm -f 2048_container Commands: -i - Run interactively -t - Create pseudo tty -a - Attach to STDIN, STDOUT or STDERR exec - Run a command in a running container run - Run a command in a new container

© Copyright IBM Corp. 2018. All rights reserved

Section 1: Lab Summary

In this section you learned how to create new containers based on images stored in Docker Hub. You also learned how to interact with containers both from the outside (top, inspect, stats, …), and from the inside (docker exec and run). Access to the Docker service via tty was demonstrated and you learned how to run Linux commands inside the container just as if you were working with a Linux OS.

© Copyright IBM Corp. 2018. All rights reserved

Section 2: Data Persistence in Docker

Purpose:	In this section, you will see one method of how data from a container can be persisted, even after a container is removed. Unless such persistence is established, any changes made to a container's data are deleted once the container is deleted. The method we will use below is Docker Volumes. With Volumes, Docker controls a location for persistent storage on your local machine that persists once a container is deleted.

Tasks:	Tasks you will complete in this lab exercise include: • Create and work with Docker volumes

© Copyright IBM Corp. 2018. All rights reserved

Section 2: Lab Instructions

Step	Action
1	**<u>Docker Volumes</u>** __a. Let's run our game application in a new container, except this time we will include an option (-v (or volume)) to instruct Docker to persist the content of a specific directory on your local machine: ~$ docker container run -d --name 2048_container -p <your port>:80 -v myvol:/usr/share/nginx/html 2048_image __b. Open bash shell on container and navigate the /usr/share/nginx/html directory: ~$ docker container exec -it 2048_container bash # cd /usr/share/nginx/html __c. Create a new file in the html folder containing the phrase, "This is my file". # echo "This is my file" > myfile Confirm the file "myfile" is listed in the directory and exit the container. # ls root@1f5d5f84c4a4:/usr/share/nginx/html# ls 50x.html Rakefile favicon.ico index.html js meta myfile style root@1f5d5f84c4a4:/usr/share/nginx/html# # exit __d. We will now remove the container using the command: ~$ docker rm -f 2048_container __e. Now, we can create a new container, referencing the persistent volume and confirm that our file is still present: ~$ docker container run -d --name 2048_container -p 8080:80 -v myvol:/usr/share/nginx/html 2048_image ~$ docker container exec -it 2048_container bash

© Copyright IBM Corp. 2018. All rights reserved

Step	Action

	# cd /usr/share/nginx/html
	
	# ls
	
	``root@1f5d5f84c4a4:/usr/share/nginx/html# ls
	50x.html Rakefile favicon.ico index.html js meta myfile style
	root@1f5d5f84c4a4:/usr/share/nginx/html#```
	
	# cat myfile
	
	``root@a9703c89b049:/usr/share/nginx/html# cat myfile
	This is my file
	root@a9703c89b049:/usr/share/nginx/html#```
	
	Volumes are extremely useful for local development projects. You can maintain several volumes to which you can attach a new directory or database that fits a specific purpose.

Section 2: Lab Summary

In this lab you were introduced to one way to persist data on the host file system. With volumes the container references a volume object on the local file system.

© Copyright IBM Corp. 2018. All rights reserved

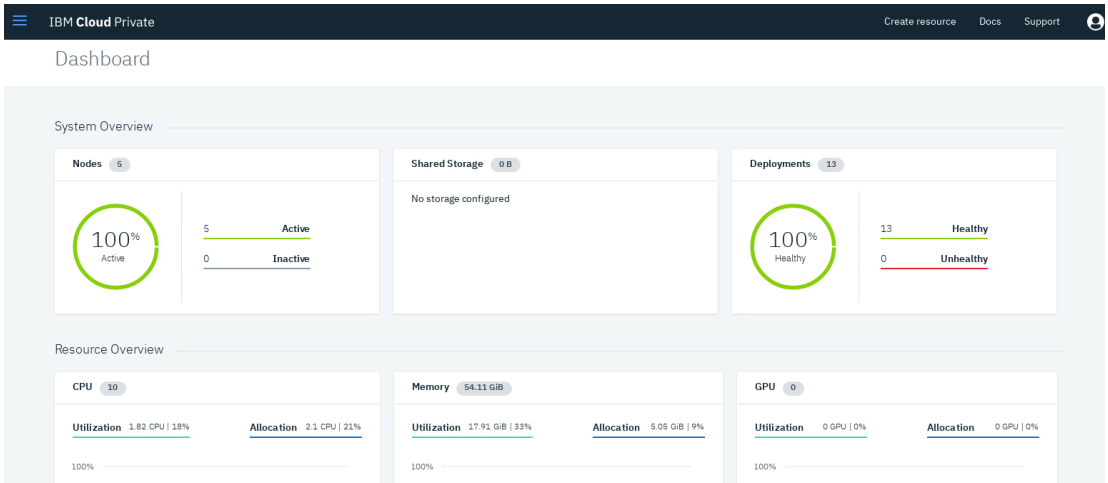
Section 3: Getting Started with Kubernetes in IBM Cloud Private

Purpose:	In this lab you will learn how to configure your environment to work with a Kubernetes cluster within IBM Cloud Private (ICP)

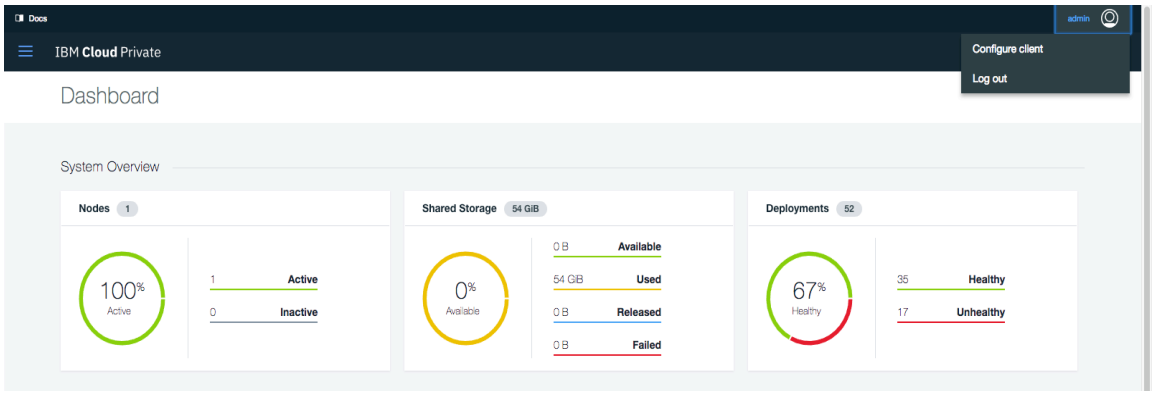
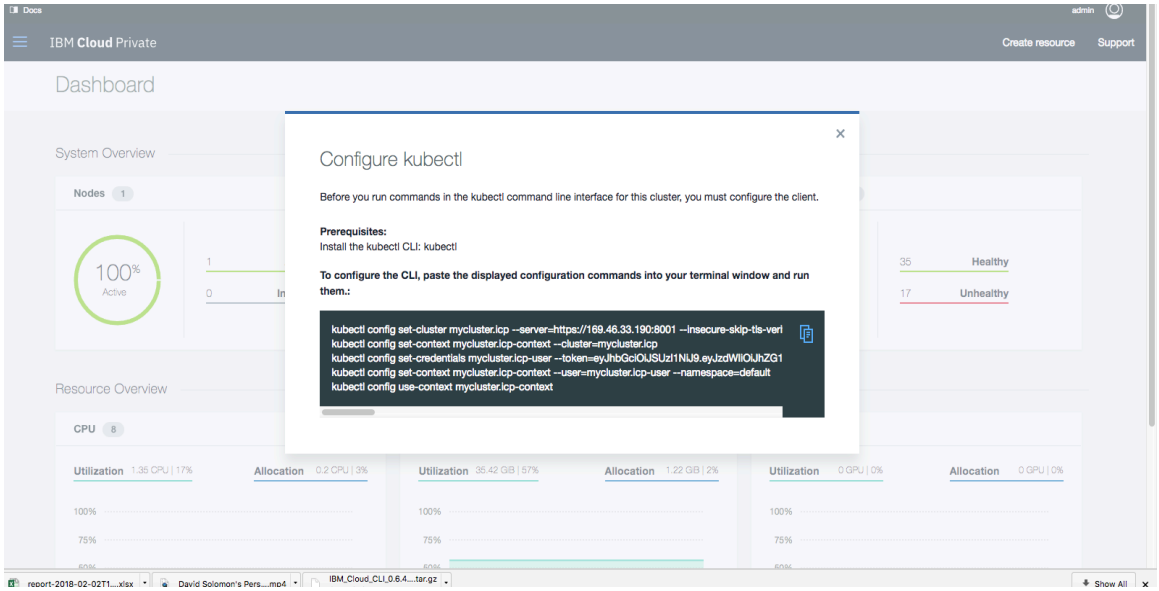
Tasks:	Tasks you will complete in this lab exercise include: • Access the IBM Cloud Private Dashboard • Access the ICP Kubernetes configuration settings • Configure your environment to use the ICP cluster

© Copyright IBM Corp. 2018. All rights reserved

Section 3: Lab Instructions

Step	Action
1	**<u>Launch the ICP Dashboard</u>** a. ICP has a centralized dashboard and control center. This dashboard is similar to the classic Kubernetes dashboard but provides additional enterprise services and features (e.g, data science, security). Open the dashboard by double clicking on the Web Console icon on the desktop. Login with username: admin/ password: admin. Click on the hamburger menu at the top and select "Dashboard"  You will notice that this ICP instance is a 5-node Kubernetes cluster.
2	**<u>Configure your Environment for ICP</u>** a. In order to interact with and control the ICP cluster from a command line using kubectl, you will need to first configure your environment to direct all kubectl commands to the ICP cluster. Fortunately, ICP helps with this by quickly providing the appropriate configuration settings for the cluster. On the ICP Dashboard, click on the word "admin" at the top left of the page next

© Copyright IBM Corp. 2018. All rights reserved

Step	Action
	to the symbol. You will then see two options, “Configure Client” and “Logout”. Select “Configure Client”.  Once selected, a dialog box called “Configure kubectl” will appear. This box contains the commands that need to be run in your local environment (the Linux environment we used for the Docker portion of this Lab) in order to properly configure kubectl to interact with the ICP cluster.  Now, copy these commands (either manually or using the blue copy symbol in the dialog box). b. Now, copy these commands (either manually or using the blue copy symbol on

© Copyright IBM Corp. 2018. All rights reserved

Step	Action
	the upper right of the dialog box). c. Open a Gnome terminal and paste these commands at a command prompt (you may need to press Return for the last command to run). [user01@dlsol0129163851 2048_master]$ kubectl config set-cluster mycluster.icp --server=https://169.46.33.190:8001 --insecure-skip-tls-verify=true Cluster "mycluster.icp" set. [user01@dlsol0129163851 2048_master]$ kubectl config set-context mycluster.icp-context --cluster=mycluster.icp Context "mycluster.icp-context" created. [user01@dlsol0129163851 2048_master]$ kubectl config set-credentials mycluster.icp-user --token=eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsImF0X2hhc2giOiJFdVVuWi1fNERUdm1GSEZZcU 5CUW13IiwiaXNzIjoiaHR0cHM6Ly9teWNsdXN0ZXIuaWNwOjk0NDMvb2lkYy9lbmRwb2ludC9PUCIsImF1ZCI6IjAwMWFhNzQ3ZDZkZDIxYzMwNmR1ZmQyMTYxZDdiZWM0IiwiZXhwIjoxNTE3NjI3NzY3LCJpYXQiOjE1MTc1ODQ1N jd9.jYnI7XgIzD2Jj7Gx5cccPjSGd7CAVDe2e6PP4kCnJwVLADSx42RAMPxKVEMvKK0hUdecVUU4pS8cI-Sx6-zms12koOxQWqIn_caB6liKhkYvoqX-2mVRWbxcc7XmBAMVAM3K0HYgKn-dLgzDFBt-H-ipb7s4gMklz9aZdaeobH9 qA7Z7PS53aRjL4ZWIosynDycugbsVKLoysdU_IJAIZeWg14mPP4wl2JTohd73IEM5GLKaA65upwRyVz9OB_c7p0LGtB2FTS62WeCdwUvN3lVHOsUynA1dv6xuT9YJYajTb0Ulb8Oo8f0jeY5oxugHObmj5Ihwit2DvSe1Vw2oOQ User "mycluster.icp-user" set. [user01@dlsol0129163851 2048_master]$ kubectl config set-context mycluster.icp-context --user=mycluster.icp-user --namespace=default Context "mycluster.icp-context" modified. [user01@dlsol0129163851 2048_master]$ kubectl config use-context mycluster.icp-context Switched to context "mycluster.icp-context". [user01@dlsol0129163851 2048_master]$ You have now successfully configured your environment to start working with Kubernetes and IBM Cloud Private.

Section 3: Lab Summary

In this section, you learned how to access the ICP Dashboard and setup a your environment to interact with a Kubernetes cluster on ICP.

© Copyright IBM Corp. 2018. All rights reserved

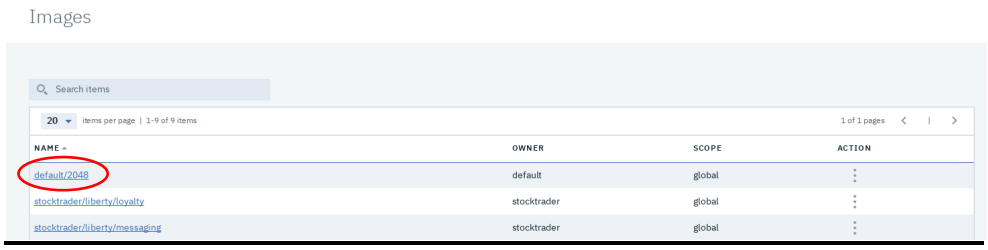
Section 4: Deploy your Application to Kubernetes

Purpose:	In this lab you will learn how to deploy an application to Kubernetes.

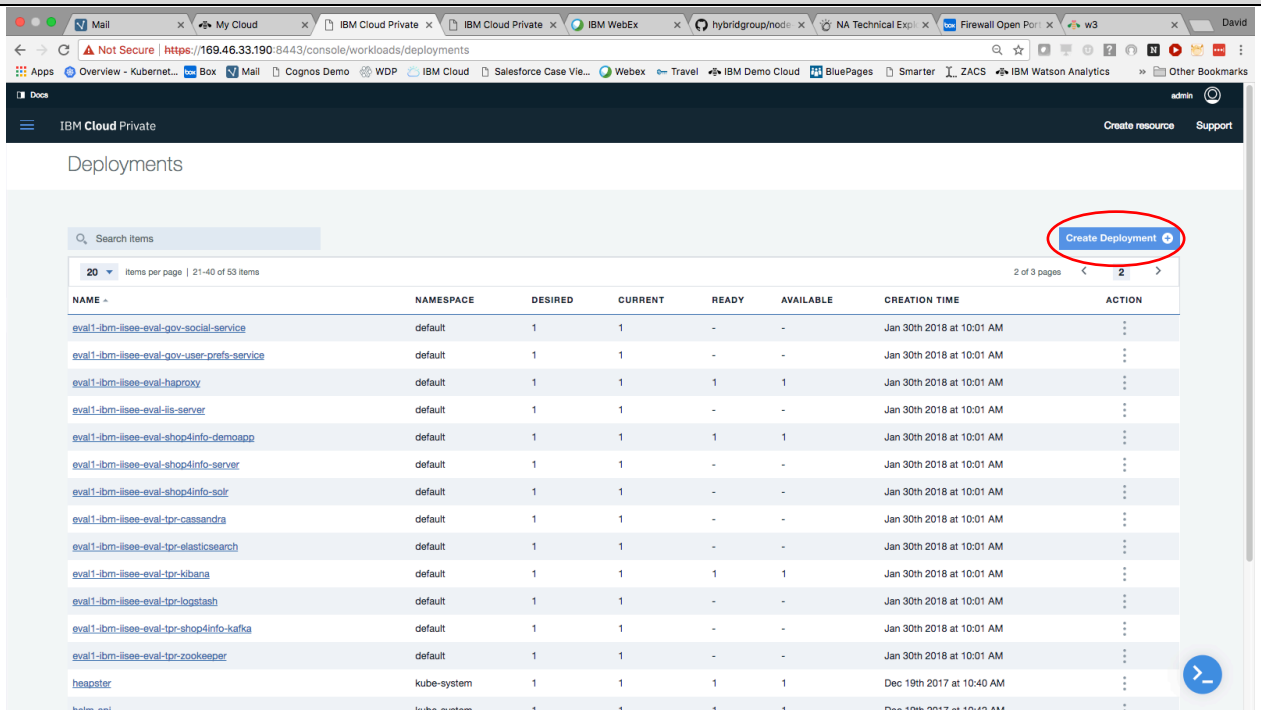
Tasks:	Tasks you will complete in this lab exercise include: • Deploy a Docker application to Kubernetes • Expose the application through a service • Access the running application

© Copyright IBM Corp. 2018. All rights reserved

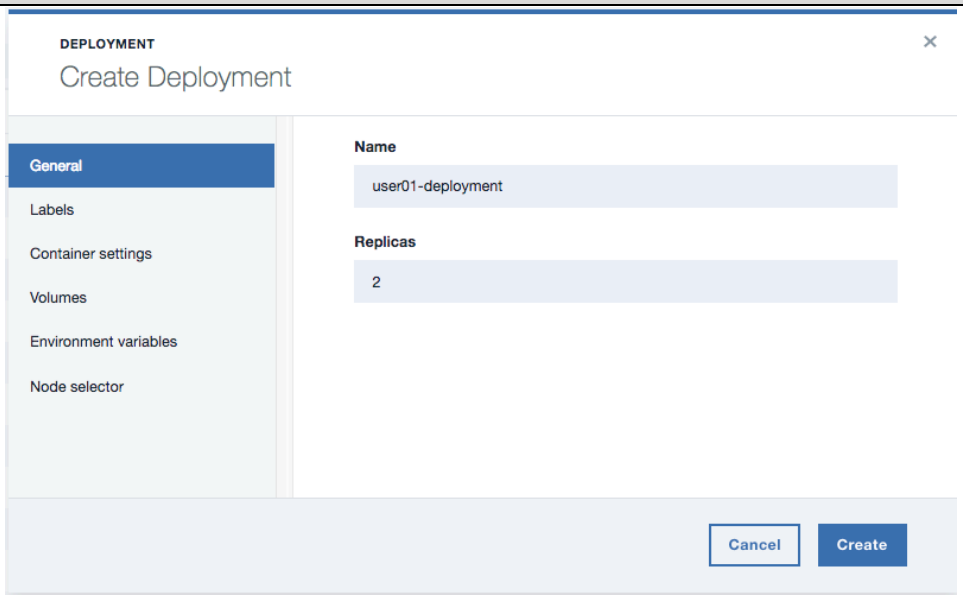
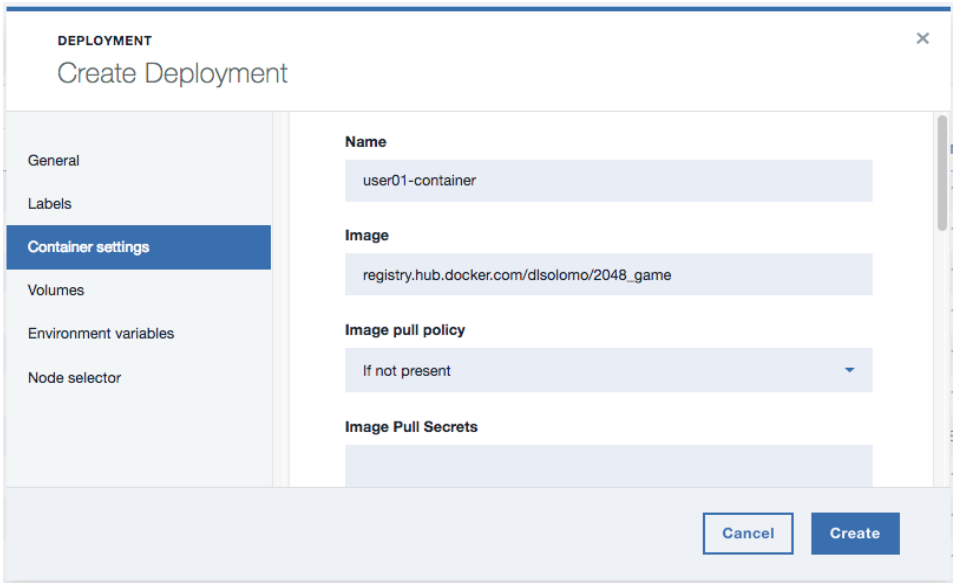
Section 4: Lab Instructions

Step	Action
1	**<u>Copy the 2048 Image to the ICP Private Docker Registry</u>** __a. In a real-world scenario, the enterprise applications you will deploy on ICP should only be accessible through a private registry, such as the one included with ICP. From there, we can then deploy our applications. In order to copy our 2048 image to this private registry, run the following commands from the same terminal window you used in the previous lab: ~$ docker login poticpcluster.icp:8500 -u admin -p admin ~$ docker tag 2048_image poticpcluster.icp:8500/default/2048:1.0 ~$ docker push poticpcluster.icp:8500/default/2048:1.0 __b. We will now confirm that the image is in the ICP registry. Return to the ICP dashboard and from the hamburger menu, select Catalog→Images. __c. Confirm that the image is listed. 
1	**<u>Create a new deployment</u>** __a. We will now deploy our game application to your ICP Cluster. Access the ICP Dashboard and select “Workloads” and then “Deployment” from the hamburger menu. __b. Select the “Create Deployment” button on the upper right of the page:

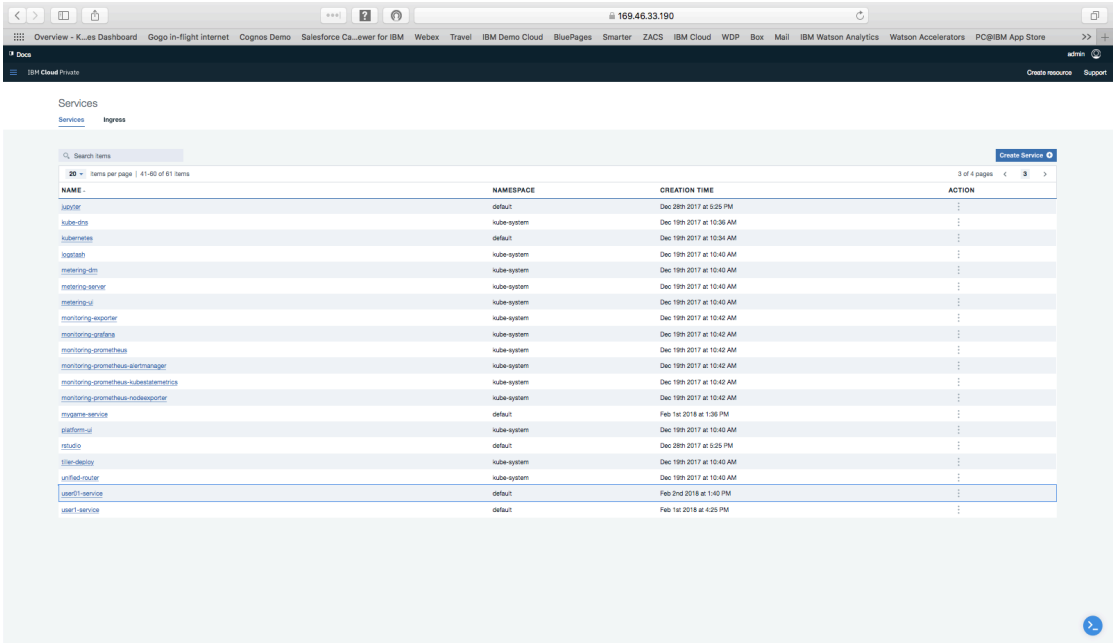
© Copyright IBM Corp. 2018. All rights reserved

Step	Action
	 __c. A create deployment form will appear. Complete the form using the following settings; as shown below and click “Create”. This will create a deployment with 2 Pods: In the “General” tab: Name= 2048-deployment Replicas= 2 In the “Container settings” tab: Name=2048-container Image= poticpcluster.icp:8500/default/2048:1.0

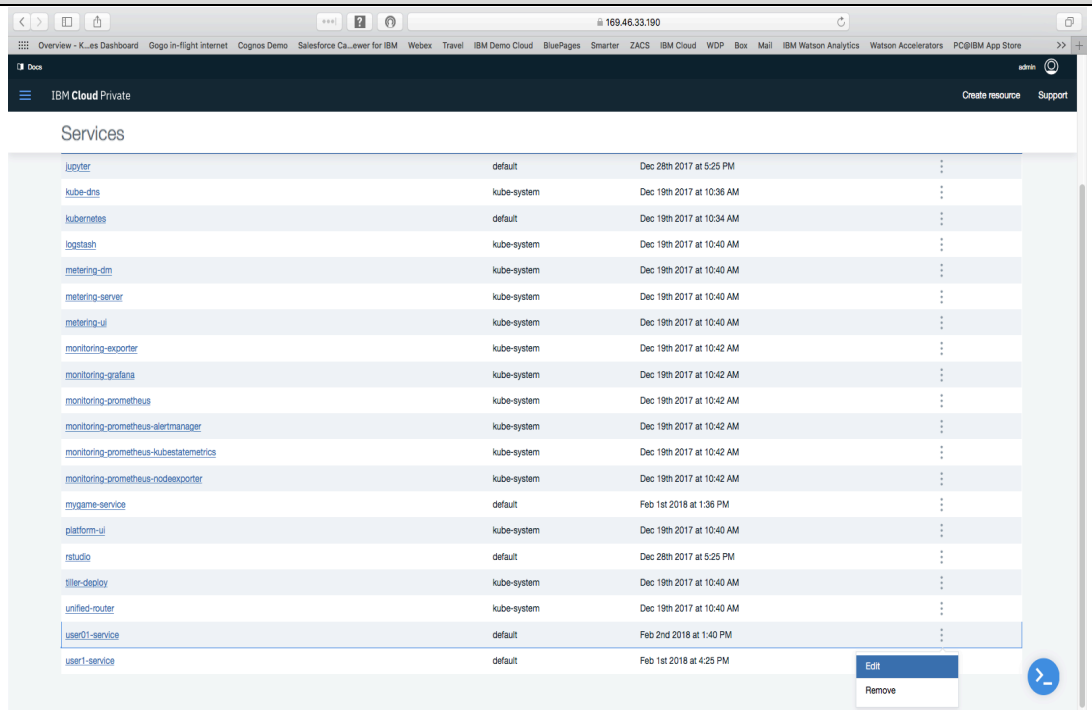
© Copyright IBM Corp. 2018. All rights reserved

Step	Action
	 Click “Create”. 
2	**<u>Exposing the application through a service</u>** __a. In order to interact with your application from outside the cluster, you wll need to create a service which provide an endpoint to expose the application. To do this, enter the following command to create a new service.

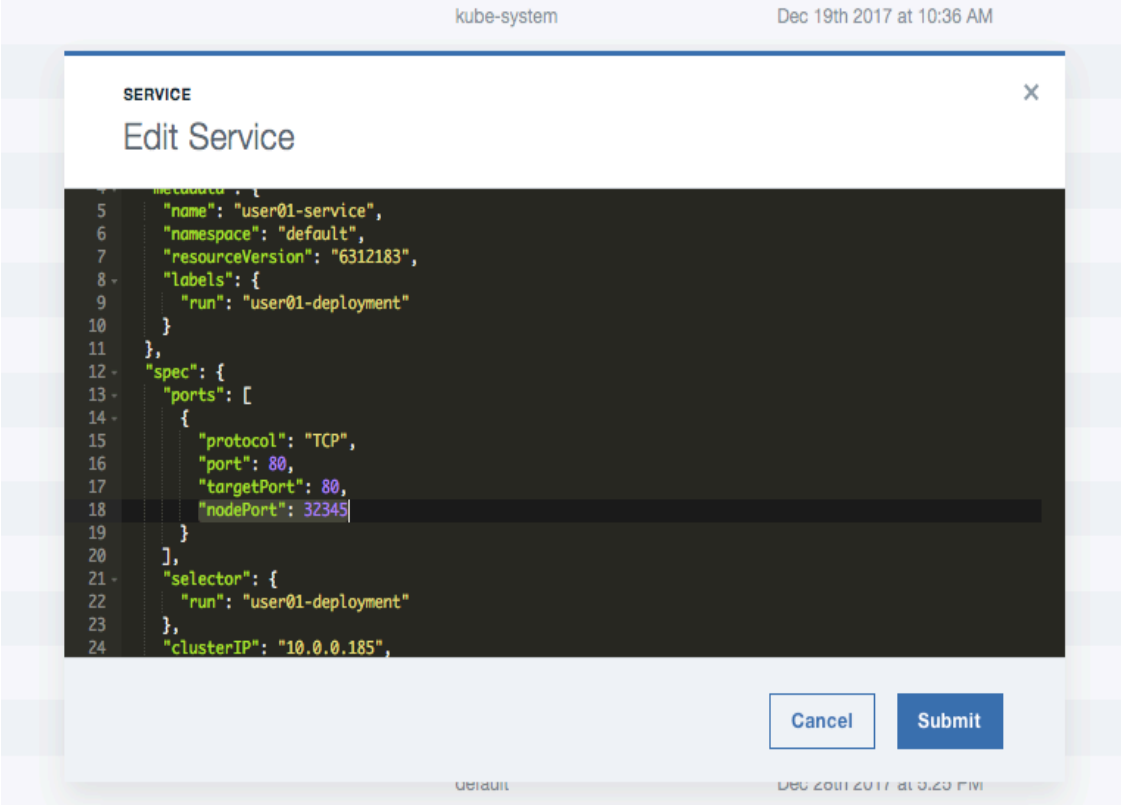
© Copyright IBM Corp. 2018. All rights reserved

Step	Action
	~$ kubectl expose deployment 2048-deployment --type=NodePort --name *2048-service* __b. Confirm the output is as shown below: [user01@dlsol0129163851 2048_master]$ kubectl expose deployment user01-deployment --type=NodePort --name user01-service service "user01-service" exposed [user01@dlsol0129163851 2048_master]$ __c. Return to the ICP dashboard. Under the “Workloads” menu option, select “Services”. The list of services will appear. Confirm your service (you may have to navigate to the 3rd or 4th page) is listed.  __d. When you expose a service, Kubernetes automatically assigns a unique port that the cluster will listen to on behalf of your application. This port is typically in the 30000-32000 range.

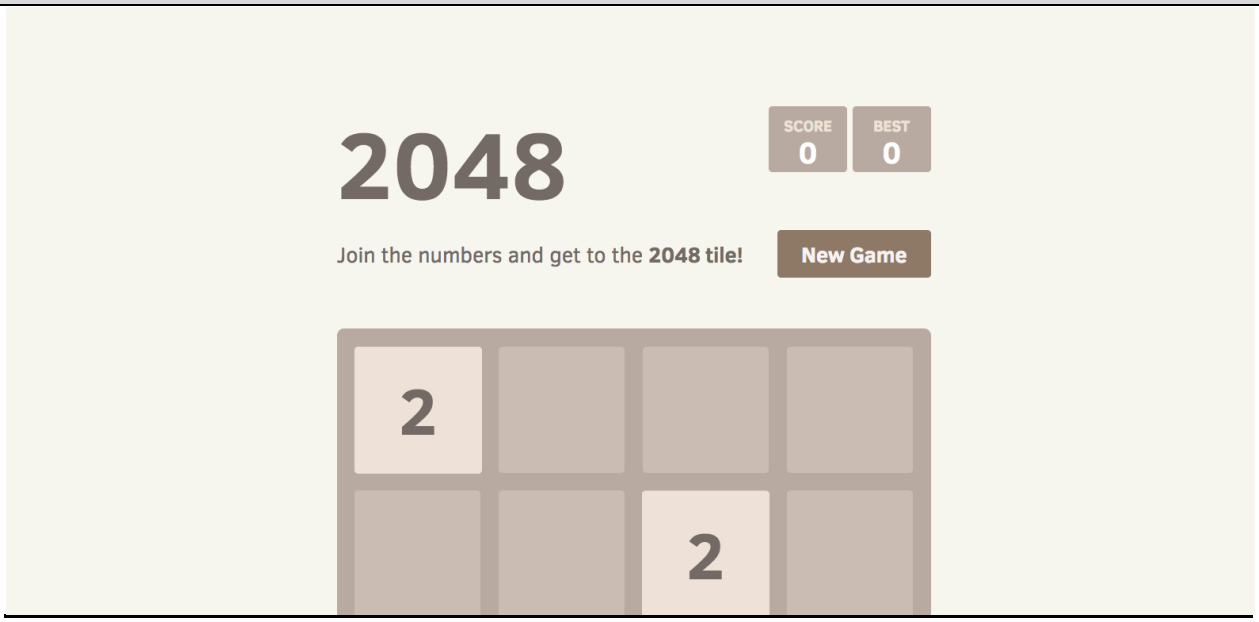
© Copyright IBM Corp. 2018. All rights reserved

Step	Action
	 __e. An editing window will appear that will allow you to edit the YAML code that defines the service. Locate the “NodePort” field and replace the port number with the <your port> used previously, as shown below.

© Copyright IBM Corp. 2018. All rights reserved

Step	Action
	 __f. Click Cancel. You have now successfully enabled your application running in the Kubernetes cluster to be accessed from the outside.
3	**<u>Access the Running Application</u>** __a. To access the application, go to your browser and enter the following URL and verify that you can access the application, as shown below: 192.168.142.102:<*your port* >

© Copyright IBM Corp. 2018. All rights reserved

Step	Action
	

Section 4: Lab Summary

In this section, you learned how to deploy an Docker application to Kubernetes, how to enable it to be access from the outside world, and how to access it.

© Copyright IBM Corp. 2018. All rights reserved

Section 5: Observing Kubernetes Resiliency

Purpose:	In this lab, you will learn how Kubernetes recovers from a container failure.

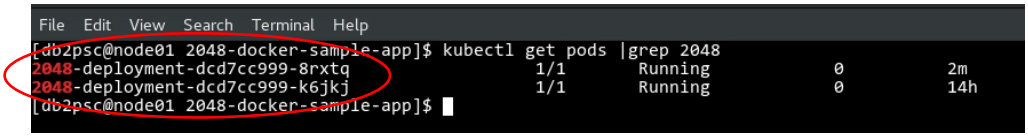
Tasks:	Tasks you will complete in this lab exercise include: • Create a new deployment with multiple Pods • Explore the ReplicaSet policy • Simulate a pod failure • Observer how the cluster quickly recovers from the failure to retain the number of available pods

© Copyright IBM Corp. 2018. All rights reserved

Section 5: Lab Instructions

Step	Action
1	**<u>Explore the ReplicaSet Policy</u>** __a. In the previous section, note that you deployed the application with 2 replicas. We will now examine these ReplicaSet in more detail. As you may recall, a ReplicaSet manages a policy that governs the how and when Pods are deployed, including the recovery of a failed Pod. This recovery is based a policy established during or after a deployment. Return to the ICP Dashboard. Go to the deployment list under “Workloads” and then “Deployments” and select the deployment you previously created. Note that there are now 2 PODs for this deployment. Pods Search items 20 items per page | 1-2 of 2 items — 1 of 1 pages — 1 NAME | NAMESPACE | STATUS | HOST IP | POD IP | READY | START TIME | ACTION 2048-deployment-dcd7cc999-8rxtq | default | Running | 192.168.142.104 | 10.1.204.78 | 1/1 | Jul 13th 2018 at 8:17 AM 2048-deployment-dcd7cc999-k6jki | default | Running | 192.168.142.105 | 10.1.250.227 | 1/1 | Jul 12th 2018 at 5:32 PM Also note under the “ReplicaSets” section that the desired number of pods is set to 2. This means that the RepliSet will always attempt to maintain 2 pods up and running to service this application. ReplicaSets Name | Desired | Current 2048-deployment-dcd7cc999 Jul 12th 2018 at 5:32 PM | 2 | 2
2	**<u>Simulate a Pod Failure</u>** __a. We will now use a kubectl command to simulate the failure of a pod. To do this, find the Pod IDs for the running Pods using the following command: ~$ kubectl get pods | grep 2048 The command will list all the running pods and their names. Identify the 2 pods associated with your application, as shown below:

© Copyright IBM Corp. 2018. All rights reserved

Step	Action
	 __b. Enter the following command to delete one of the Pods (it does not matter which one). Copy the name from the output of the previous step. ~$ kubectl delete pods <*the name of one of your Pods*>.
3	**<u>Observe that the Cluster Recovers from the Failure</u>** __a. Wait approximately 30 seconds and run the following command again and notice that one of the pods now has a different name. This is because when we deleted the other pod, the ReplicaSet rules immediately ensured that a new pod was created to ensure continuity, reliability, and quality of servicing the application. ~$ kubectl get pods | grep 2048

Section 5: Lab Summary

In this section, you learned how Kubernetes can quickly recover from a Pod failure.

© Copyright IBM Corp. 2018. All rights reserved

Section 6: Deploying a Microservices Application in ICP

Purpose:

In previous labs, you worked with a single container application (the 2048 game). However, in a real-world situation, the value of ICP is in being able to quickly deploy and manage complex applications which may consist of many microservices. In this lab, you will learn how you can use ICP to deploy such an application.

The example application used here is based on the work from the IBM Cloud team and is available at https://github.com/IBMStockTrader

The microservices stock trader application is based on the following Docker containers.

Component	Docker container
Db2	`store/ibmcorp/db2_developer_c:11.1.3.3-x86_64`
MQ	`store/ibmcorp/mqadvanced-server-dev:9.0.3`
Redis	`ibmcom/redis-ha:4.0.6-r0`
Liberty	`store/ibmcorp/websphere-liberty:javaee7`
Liberty-Portfolio	`poticpcluster.icp:8500/stocktrader/liberty/portfolio:1.0.1`
Liberty-Trader	`poticpcluster.icp:8500/stocktrader/liberty/trader:1.0.1`
Liberty-Loyalty	`poticpcluster.icp:8500/stocktrader/liberty/loyalty:1.0.1`
Liberty-Notify-Twitter	`poticpcluster.icp:8500/stocktrader/liberty/notify-twitter:1.0.1`
Liberty-Notify-Slack	`poticpcluster.icp:8500/stocktrader/liberty/notify-slack:1.0.1`
Liberty-Messaging	`poticpcluster.icp:8500/stocktrader/liberty/messaging:1.0.1`
Liberty-stockquote	`poticpcluster.icp:8500/stocktrader/liberty/stockquote:1.0.1`
Nodejs-Trader	`poticpcluster.icp:8500/stocktrader/nodejs/trader:1.0.1`

- Due to time restrictions, running instances Db2, MQ and Redis have already been created in the cluster
- In this lab exercise, we describe the process of building other containers for different microservices components and deploy them to the IBM Cloud Private cluster.
- The `Portfolio` microservice communicates with `Db2` for persistence storage of data in relational tables. This microservice receives HTTP requests (GET, PUT, POST and DELETE) from either Liberty based `Trader` GUI and Node.js-based Trader GUI.
- The `Portfolio` microservice using JMS puts messages in IBM `MQ` and `Messaging` microservice consumes those messages from the MQ.

© Copyright IBM Corp. 2018. All rights reserved

- The `Loyalty` microservice determines the loyalty level of a given portfolio owner, based on their total portfolio value. It provides notifications whenever the loyalty level changes. When it detects a change in level, it does a POST to an IBM Cloud Function (earlier aka `OpenWhisk`) action sequence, which builds a message and posts it to a Slack channel (#slack-test on ibm-cloud.slack.com) using `notify-slack` microservice.

- The `notify-twitter` microservice sends a tweet via `@IBMStockTrader` account on Twitter.

- Both `notify-twitter` and `notify-slack` microservices use the same network service and if both of these microservices are installed, the message to either Slack channel or Twitter will be random. The `Itsio` routing rules could be used to determine as which gets used and under what conditions.

- The `stockquote` microservice gets the price of a specified stock. It hits an API in API Connect, which drives a call to '*Quandl.com*' to get the actual data. This service uses `Redis` for caching. When a quote is requested, it first checks to see if the answer is in the cache, and if so, whether the quote is less that 24 hours old. (Quandl only returns the previous business day's closing price.) If so, just use that. Otherwise (or if any exceptions occur communicating with `Redis`), it drives the REST call to API Connect as usual, then adds it to `Redis` so it's there for next time.

- ***Note:*** Due to the time constraints of this lab session, we have automated many of the deployment tasks in a series of scripts that you will you review and run.

- ***Note:*** The runtime components of this application have already been pre-built, since it is not the intent of this lab to focus on building the application. The github link above provides details on how the application can be built.

Tasks:	Tasks you will complete in this lab exercise include: • Build Docker Container for the Stock Trader Microservices • Push the Docker Containers to the IBM Private Registry • Create the Db2 tables for the application • Deploy the Microservices • Expose the microservice application • Check the Redis Server and MQ • Run the Stock Trader Application

© Copyright IBM Corp. 2018. All rights reserved

Section 6: Lab Instructions

Step	Action

1	**Building the Docker Containers for Each Microservice**
	
	__a. Switch to the GNOME Terminal command line.su
	
	__b. Switch to root by typing su. Login with the password "password".
	
	__c. Enter a bash shell by typing bash.
	
	__d. Type cd8 to switch the lab directory to 08ms (microservices).
	`[root@node01 07ta]# cd8`
	`[root@node01 08ms]#`
	
	__e. Review 00-build-docker-containers-DO-NOT-RUN-IF-NO-Internet script. [Do not run.]
	`CWD=$PWD`
	`DIRS=$(find . -mindepth 1 -maxdepth 1 -type d -printf '%f\n' | sort)`
	
	`for dir in $DIRS`
	`do`
	`  echo ==========`
	`  echo $dir - Running dockerbuild`
	`  echo ==========`
	`  cd $dir`
	`  ./01-builddocker`
	`  cd $CWD`
	`  echo ==========`
	`  echo`
	`done`
	
	__f. The above script runs the 01-builddocker script from each of the subdirectories. This script creates the Docker container for each microservice.
	
	__g. Run ls -l
	`[root@node01 08ms]# ls -l`
	`total 32`

© Copyright IBM Corp. 2018. All rights reserved

Step	Action

```
-rwxr-xr-x 1 root   root    796 Apr 16 09:58 00-build-docker-
containers-DO-NOT-RUN-IF-NO-Internet
-rwxr-xr-x 1 root   root    816 Apr 16 10:25 01-push-image-to-
local-registry
-rwxr-xr-x 1 db2psc db2psc  973 Apr 16 10:27 02-deploy-docker-
containers
drwxr-xr-x 3 db2psc db2psc  231 Apr 16 10:18 03-portfolio
drwxr-xr-x 3 db2psc db2psc  147 Apr 16 10:18 04-trader
drwxr-xr-x 3 db2psc db2psc  147 Apr 16 10:19 05-stock-quote
drwxr-xr-x 3 db2psc db2psc  147 Apr 16 10:19 06-messaging
drwxr-xr-x 3 db2psc db2psc  189 Apr 16 10:19 07-notification-
slack
drwxr-xr-x 3 db2psc db2psc  191 Apr 16 10:19 08-notification-
twitter
drwxr-xr-x 3 db2psc db2psc  147 Apr 16 10:19 09-loyalty-level
drwxr-xr-x 3 db2psc db2psc  158 Apr 16 10:19 10-trader-nodejs
-rwxr-xr-x 1 db2psc db2psc 3295 Apr  9 22:59 20-kgl
-rwxr-xr-x 1 db2psc db2psc  506 Apr  8 14:09 30-
setImagePullAlways
-rwxr-xr-x 1 db2psc db2psc  521 Apr  8 14:09 40-
setImageIfNotPresent
-rwxr-xr-x 1 db2psc db2psc  449 Apr  8 07:50 50-cleanall
-rwxr-xr-x 1 db2psc db2psc  197 Apr  9 23:25 post
```

__h. The directories from `03-portfolio` through `10-trader-nodejs` are microservices directories.

__i. Run `cd 03-portfolio`

```
[root@node01 08ms]# cd 03-portfolio/
[root@node01 03-portfolio]#
```

__j. Run `ls -l`

```
[root@node01 03-portfolio]# ls -l
total 3852
-rwxr-xr-x 1 db2psc db2psc     865 Apr 16 10:08 01-builddocker
-rwxr-xr-x 1 root   root      1112 Apr 16 10:18 02-pushdocker
-rwxr-xr-x 1 db2psc db2psc    1251 Apr  8 21:51 03-createsecrets
-rwxr-xr-x 1 db2psc db2psc     696 Apr  6 14:08 04-deploydocker
-rwxr-xr-x 1 db2psc db2psc     952 Apr  6 14:08 05-createtables
drwxr-xr-x 5 db2psc db2psc      74 Apr 10 13:06 config
-rw-r--r-- 1 db2psc db2psc 3905812 Apr  3 23:43 db2jcc4.jar
-rw-r--r-- 1 db2psc db2psc    2324 Apr  9 22:24 deploy.yaml
-rw-r--r-- 1 db2psc db2psc     141 Apr  4 00:16 Dockerfile
-rw-r--r-- 1 db2psc db2psc     474 Apr  8 07:59 tables.sql
```

© Copyright IBM Corp. 2018. All rights reserved

Build Containers

__k. Review `01-builddocker` [Do not run].

```
[root@node01 03-portfolio]# cat 01-builddocker
NAMESPACE=stocktrader

echo ==========================================================
echo Create name space : $NAMESPACE
echo ==========================================================

cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: $NAMESPACE
EOF

-  - - -

IMAGENAME=liberty/portfolio:1.0.1

echo CLUSTERNAME=$CLUSTERNAME

docker build -t $CLUSTERNAME.icp:8500/$NAMESPACE/$IMAGENAME -f
Dockerfile .
```

__l. The above script creates a `stocktrader` namespace and runs `docker build` command to build the container as per the name using `-t` switch.

__m. Review `Dockerfile`.

```
[root@node01 03-portfolio]# cat Dockerfile
FROM store/ibmcorp/websphere-liberty:javaee7
ADD config /config
ADD db2jcc4.jar ./
RUN installUtility install --acceptLicense defaultServer
```

__n. The above `Dockerfile` uses the WebSphere Liberty base image. (We have already downloaded the base image.)

> **Note:** If the Docker image is not present, Docker downloads the image from Docker Store. Please refer to Appendix-A for the procedure to download IBM Docker containers.

© Copyright IBM Corp. 2018. All rights reserved

Step	Action

	__o. It then adds the config folder (`ADD config /config`) to the base image, copies `db2jcc4.jar` to root of the image and runs `InstallUtility` to create the default server.
	
	__p. In the microservices environment, each bundled, similar components are stored in their own Docker container and just deploying an individual container serves the purpose of continuous improvement and delivery.
	
	__q. Run `tree config`.
	
	`[root@node01 03-portfolio]# tree config/`
	`config/`
	`├── apps`
	`│ └── Portfolio.war`
	`├── configDropins`
	`│ └── defaults`
	`│ └── keystore.xml`
	`├── resources`
	`│ └── security`
	`│ └── key.jks`
	`└── server.xml`
	
	__r. Note that this directory comes from the development organization or the CICD (Continuous Improvement and Delivery) mechanism through GitHub (or any other source control) though Jenkins will trigger the build process, build container and deploy to the right environment.
	
	__s. In this session, we described those processes to show individual components so that you can build your pipeline using SCM (Source Control Mechanism) and Jenkins.
	
	__t. Note that we have `Portfolio.war` in `apps` directory. The security files `key.jks` and `keystore.xml` in their respective directories.
	
	__u. Review `server.xml` – through which Liberty uses features, security, JDBC data sources and more.
	
	`[root@node01 03-portfolio]# cat config/server.xml`
	`<server description="Portfolio server">`
	`  <featureManager>`
	`  <feature>microProfile-1.3</feature>`
	`  <feature>jdbc-4.1</feature>`
	`  <feature>jndi-1.0</feature>`
	`  <feature>appSecurity-2.0</feature>`
	`  <feature>openapi-3.0</feature>`
	`  </featureManager>`
	`- - - -`

© Copyright IBM Corp. 2018. All rights reserved

Step	Action

```
    <connectionManager id="DB2-Connections"
         minPoolSize="5" maxPoolSize="50"/>
    <dataSource id="PortfolioDB"
jndiName="jdbc/Portfolio/PortfolioDB"
         connectionManagerRef="DB2-Connections"
         isolationLevel="TRANSACTION_READ_COMMITTED">
        <jdbcDriver>
            <library name="DB2" description="DB2 JDBC driver
jar">
                <file id="db2jcc4" name="/db2jcc4.jar"/>
            </library>
        </jdbcDriver>
        <properties.db2.jcc
               serverName="${env.JDBC_HOST}"
               portNumber="${env.JDBC_PORT}"
               databaseName="${env.JDBC_DB}"
               user="${env.JDBC_ID}"
               password="${env.JDBC_PASSWORD}"/>
    </dataSource>

-  - - -

    <webApplication id="Portfolio" name="Portfolio"
          location="Portfolio.war" contextRoot="/portfolio">
        <application-bnd>
            <security-role id="StockTrader" name="StockTrader">
                <special-subject type="ALL_AUTHENTICATED_USERS"
id="IBMid"/>
            </security-role>
        </application-bnd>
    </webApplication>
</server>
```

__v. Note the features this Liberty server uses through `featureManager` section.

__w. Review the JDBC connection properties defined through environment variables. We use Kubernetes secrets to provide these values and then Kubernetes transfers them to the environment variables when starting the container. We will demonstrate this connection later in this section.

__x. We will not run `00-build-docker-containers-DO-NOT-RUN-IF-NO-Internet` since we already built containers to save time.

© Copyright IBM Corp. 2018. All rights reserved

Step	Action

2	**Push Docker Containers into the Private Registry**
	
	__a. IBM Cloud Private provides a local private registry to which we push the Docker container. Usually, the CICD process (through Jenkins) pushes the Docker container to the IBM Cloud Private local registry.
	
	__b. Run `cd8` to change the lab directory.
	
	`[root@node01 03-portfolio]# cd8`
	`[root@node01 08ms]#`
	
	__c. Run `cat 03-portfolio/02-pushcontainer`
	
	`[root@node01 08ms]# cat 03-portfolio/02-pushdocker`
	
	`IMAGENAME=liberty/portfolio:1.0.1`
	
	`echo CLUSTERNAME=$CLUSTERNAME`
	
	`docker login $CLUSTERNAME.icp:8500 -u $DEFAULTUSERNAME -p $DEFAULTPASSWORD`
	`docker push $CLUSTERNAME.icp:8500/$NAMESPACE/$IMAGENAME`
	
	__d. After logging in to the local Docker registry, the Docker push command is used to copy the image to the IBM Cloud Private registry.
	
	__e. Review `01-push-image-to-local-registry`
	
	`[root@node01 08ms]# cat 01-push-image-to-local-registry`
	`- - - -`
	
	`CWD=$PWD`
	`DIRS=$(find . -mindepth 1 -maxdepth 1 -type d -printf '%f\n' | sort)`
	
	`for dir in $DIRS`
	`do`
	`  echo ==========`
	`  echo $dir - Running dockerbuild`
	`  echo ==========`
	`  cd $dir`
	`  ./02-pushdocker`
	`  cd $CWD`

© Copyright IBM Corp. 2018. All rights reserved

Step	Action

```
   echo
==========================================================
   echo
done
```

__f. The above script runs `02-pushdocker` in all subdirectories to push the Docker image to the local registry.

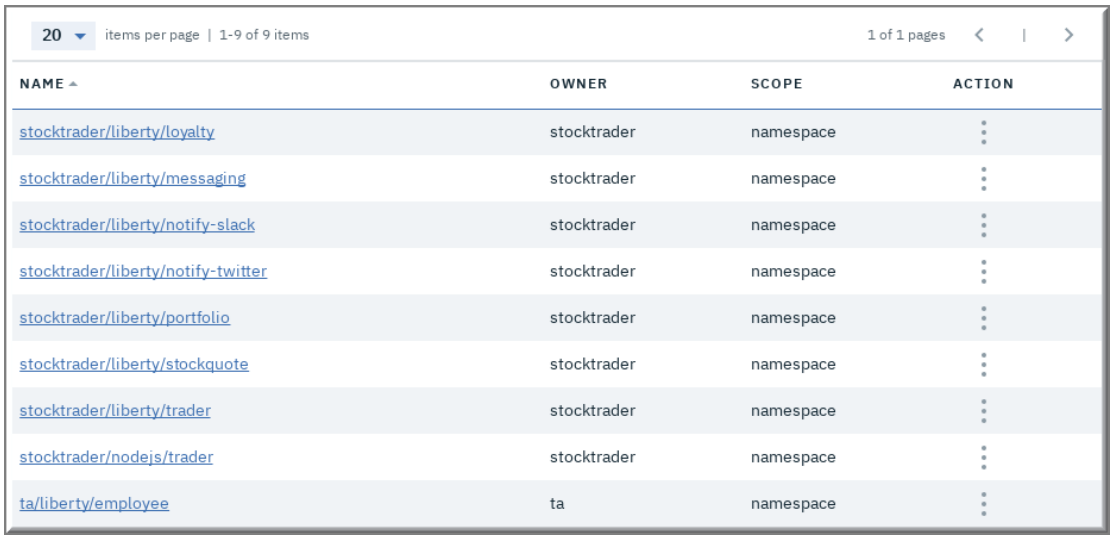
__g. Run `01-push-image-to-local-registry`

```
[root@node01 08ms]# ./01-push-image-to-local-registry
-  ---

CLUSTERNAME=poticpcluster
==========================================================
Push image to the local registry
==========================================================

WARNING! Using --password via the CLI is insecure. Use --
password-stdin.
Login Succeeded
The push refers to a repository
[poticpcluster.icp:8500/stocktrader/liberty/portfolio]
```

__h. The above script pushes all containers to the IBM Cloud Private local registry.

__i. Switch to the web UI.

__j. Click `Hamburger` ☰ ⇨ `Catalog` ⇨ `Images`.



© Copyright IBM Corp. 2018. All rights reserved

Step	Action
	__k. The images are now stored in IBM Cloud Private registry and through our deployment process, the images can be pulled by any worker node. __l. Switch back to the command line.

3	**<u>Create Database Tables</u>**
	
	__a. Change directory to `03-portfolio`.
	
	`[root@node01 08ms]# cd 03-portfolio/`
	
	__b. Review script `05-createtables`
	
	`[root@node01 03-portfolio]# cat 05-createtables`
	
	`DB2POD=$(kubectl -n default get pods --selector app=dev-ibm-db2oltp-dev -o jsonpath='{.items[].metadata.name}')`
	
	`kubectl -n default cp ./tables.sql $DB2POD:/tmp`
	
	`kubectl -n default exec -it $DB2POD -- /bin/bash -c "su - db2psc -c \"db2 -tvf /tmp/tables.sql\""`
	
	__c. The script determines the Db2 pod name using label `app= dev-ibm-db2oltp-dev`. We then copy the create table script to `/tmp` folder of the Db2 container and then run `kubectl exec` command to run the Db2 command to create tables.
	
	__d. Run `05-createtables`
	
	`[root@node01 03-portfolio]# ./05-createtables`
	
	`Get the db2 pod name`
	`Db2 pod name = dev-ibm-db2oltp-dev-0`
	`CONNECT TO PSDB`
	
	`  Database Connection Information`
	
	`  Database server  = DB2/LINUXX8664 11.1.3.3`
	`  SQL authorization ID  = DB2PSC`
	`  Local database alias  = PSDB`
	
	`CREATE TABLE Portfolio ( owner  VARCHAR(32) NOT NULL, total  DOUBLE, loyalty VARCHAR(8), PRIMARY KEY(owner) )`
	`DB20000I  The SQL command completed successfully.`
	
	`CREATE TABLE Stock ( owner  VARCHAR(32) NOT NULL, symbol  VARCHAR(8) NOT NULL, shares  INTEGER, price  DOUBLE, total  DOUBLE, dateQuoted DATE, FOREIGN KEY (owner) REFERENCES Portfolio(owner) ON DELETE CASCADE, PRIMARY KEY(owner, symbol) )`
	`DB20000I  The SQL command completed successfully.`
	
	`CONNECT RESET`

© Copyright IBM Corp. 2018. All rights reserved

Step	Action
	`DB20000I  The SQL command completed successfully.`

4	**Deploy Microservices**
	
	__a. Review `02-deploy-docker-containers`
	
	`[root@node01 08ms]# cat 02-deploy-docker-containers`
	`CWD=$PWD`
	`DIRS=$(find . -mindepth 1 -maxdepth 1 -type d -not -path ./07-notification-slack -printf '%f\n' | sort)`
	
	`for dir in $DIRS`
	`do`
	`  cd $dir`
	`  ./03-createsecrets`
	`  ./04-deploydocker`
	`  cd $CWD`
	`done`
	
	__b. The above script runs `03-createsecrets` and `04-deploydocker`.
	
	__c. Run `cat */03-createsecrets`
	
	`[root@node01 08ms]# cat */03-createsecrets`
	
	`# jwt - json web token secret`
	`kubectl -n stocktrader \`
	`  create secret generic jwt \`
	`  --from-literal=audience=stock-trader \`
	`  --from-literal=issuer=http://stock-trader.ibm.com`
	
	`# Db2 secret`
	`kubectl -n stocktrader \`
	`  create secret generic db2 \`
	`  --from-literal=id=db2psc \`
	`  --from-literal=pwd=password \`
	`  --from-literal=host=dev-ibm-db2oltp-dev.default.svc.cluster.local \`
	`  --from-literal=port=50000 \`
	`  --from-literal=db=PSDB`
	
	__d. Scroll to see that we created a secret object for each microservice (if applicable), which provides runtime credentials.
	
	__e. For example, notice the Db2 secret, which has the name of the database, user ID, password, host name and the port number. These values from secret through deploy.yaml are passed to the container in the form of environments variables and then the `server.xml` picks up these values from the container environment variables.

© Copyright IBM Corp. 2018. All rights reserved

Step	Action

__f. Run `kubectl -n stocktrader get secret db2 -o yaml`

```
[root@node01 08ms]# kubectl -n stocktrader get secret db2 -o yaml
apiVersion: v1
data:
  db: UFNEQg==
  host:
ZGV2LWlibS1kYjJvbHRwLWRldi5kZWZhdWx0LnN2Yy5jbHVzdGVyLmxvY2Fs
  id: ZGIycHNj
  port: NTAwMDA=
  pwd: cGFzc3dvcmQ=
kind: Secret
metadata:
  creationTimestamp: 2018-04-10T02:33:45Z
  name: db2
  namespace: stocktrader
  resourceVersion: "81980"
  selfLink: /api/v1/namespaces/stocktrader/secrets/db2
  uid: 97abbd2f-3c67-11e8-970f-005056271837
type: Opaque
```

__g. Note the values of the secret are stored in encoded form.

__h. For example: If you want to see the Db2 password, run `echo cGFzc3dvcmQ= | base64 -d`

```
[root@node01 08ms]# echo cGFzc3dvcmQ= | base64 -d
password[root@node01 08ms]#
```

__i. The encoded value is `password`.

__j. Review `03-portfolio/deploy.yaml`

```
[root@node01 08ms]# cat 03-portfolio/deploy.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: portfolio
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: portfolio
        solution: stocktrader
        id: portfolio
        version: 1.0.1
    spec:
      containers:
      - name: portfolio
```

© Copyright IBM Corp. 2018. All rights reserved

Step	Action

```
        image: poticpcluster.icp:8500/stocktrader/liberty/portfolio:1.0.1
        env:
          - name: JDBC_HOST
            valueFrom:
              secretKeyRef:
                name: db2
                key: host
          - name: JDBC_PORT
            valueFrom:
              secretKeyRef:
                name: db2
                key: port
          - name: JDBC_DB
            valueFrom:
              secretKeyRef:
                name: db2
                key: db
          - name: JDBC_ID
            valueFrom:
              secretKeyRef:
                name: db2
                key: id
          - name: JDBC_PASSWORD
            valueFrom:
              secretKeyRef:
                name: db2
                key: pwd
          - name: JWT_AUDIENCE
            valueFrom:
              secretKeyRef:
                name: jwt
                key: audience
          - name: JWT_ISSUER
            valueFrom:
              secretKeyRef:
                name: jwt
                key: issuer
        ports:
          - containerPort: 9080
          - containerPort: 9443
        imagePullPolicy: Always
---
#Deploy the service
apiVersion: v1
kind: Service
metadata:
  name: portfolio-service
  labels:
    app: portfolio
spec:
  type: NodePort
  ports:
    - name: http
      protocol: TCP
      port: 9080
      targetPort: 9080
    - name: https
```

© Copyright IBM Corp. 2018. All rights reserved

Step	Action

```
      protocol: TCP
      port: 9443
      targetPort: 9443
  selector:
    app: portfolio
---
#Configure the ingress
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: "nginx"
    ingress.kubernetes.io/affinity: "cookie"
    ingress.kubernetes.io/session-cookie-name: "route"
    ingress.kubernetes.io/session-cookie-hash: "sha1"
    ingress.kubernetes.io/secure-backends: "true"
    ingress.kubernetes.io/app-root: "/portfolio"
  name: portfolio-ingress
spec:
  rules:
  - host:
    http:
      paths:
      - path: /portfolio
        backend:
          serviceName: portfolio-service
          servicePort: 9443
```

__k. The portfolio microservice is deployed in Kubernetes cluster through the aforesaid `deploy.yaml` file.

__l. The salient features of the above `deploy.yaml` are as follows:

- ✓ The docker container `poticpcluster.icp:8500/stocktrader/liberty/portfolio:1.0.1` (from ICP registry) is used to deploy portfolio microservice. It has been given a label `app` set to `portfolio`.
- ✓ The `JDBC_HOST` environment variable to the docker container is mapped to Kubernetes secret `db2` parameter `host` and other parameters as well.
- ✓ The Liberty application server is using two ports 9080 (HTTP) and 9553 (HTTPS).
- ✓ The network service is named `portfolio-service` and the selector label is set to `app:portfolio` – which is the glue between network service and the Docker container. This is how the network traffic is routed. The type of the service is NodePort – which allows connections from the proxy server (or any worker node) to these exposed ports.

© Copyright IBM Corp. 2018. All rights reserved

Step	Action

✓ The optional routing is done by defining ingress named as `portfolio-ingress` with path set to `/portfolio` and this ingress is tied to the network service `portfolio-service.`

✓ The advantage of using the ingress is to reach out to path without having to specify port number and this is done through an Ingress Controller which is a reverse proxy provided through `nginx`.

__m. Review `03-portfolio/04-deploydocker`

```
[root@node01 08ms]# cat 03-portfolio/04-deploydocker
-  ---

echo ============================================================
echo Running command \"kubectl --namespace stocktrader apply -f
deploy.yaml\"
echo ============================================================
kubectl --namespace stocktrader apply -f deploy.yaml
```

__n. After `deploy.yaml` is created for each microservice, the `kubectl apply -f` is used to deploy the objects.

__o. After we have seen the above deployment procedure, we can now deploy all microservices.

__p. Run `02-deploy-docker-containers`

```
[root@node01 08ms]# ./02-deploy-docker-containers
==========================================================
Create Secrets and build Docker Containers
==========================================================
03-portfolio - Running dockerbuild, create secrets and docker deploy
==========================================================
Create secrets for trader container

secret "jwt" created
secret "db2" created
secret "ingress-host" created
==========================================================
Deploy Liberty Docker container for trader
==========================================================
Running command "kubectl --namespace stocktrader apply -f deploy.yaml"
==========================================================
deployment "portfolio" created
service "portfolio-service" unchanged
ingress "portfolio-ingress" configured
==========================================================
04-trader - Running dockerbuild, create secrets and docker deploy
==========================================================
Create secrets for trader container
secret "jwt" created
secret "oidc" created
==========================================================
Deploy Liberty Docker container for trader
```

© Copyright IBM Corp. 2018. All rights reserved

Step	Action
	``` ========== ========== Running command "kubectl --namespace stocktrader apply -f deploy.yaml" ========== deployment "trader" created service "trader-service" unchanged ingress "trader-ingress" unchanged ========== 05-stock-quote - Running dockerbuild, create secrets and docker deploy ========== Create secrets for trader container ========== secret "redis" created ========== Deploy Liberty Docker container for stock ========== Running command "kubectl --namespace stocktrader apply -f deploy.yaml" ========== deployment "stockquote" created service "stock-quote-service" unchanged ingress "stock-quote-ingress" unchanged ========== 06-messaging - Running dockerbuild, create secrets and docker deploy ========== Create secrets for messaging container ========== secret "mq" created ========== Deploy Liberty Docker container for messaging ========== Running command "kubectl --namespace stocktrader apply -f deploy.yaml" ========== deployment "messaging" created ========== 08-notification-twitter - Running dockerbuild, create secrets and docker deploy ========== Create secrets for notification twitter container ========== secret "twitter" created ========== Deploy Liberty Docker container for notification twitter ========== Running command "kubectl --namespace stocktrader apply -f deploy.yaml" ========== deployment "notification-twitter" created service "notification-service" unchanged ingress "notification-ingress" unchanged ========== 09-loyalty-level - Running dockerbuild, create secrets and docker deploy ========== Create secrets for notify-level ========== Deploy Liberty Docker container for loyalty ========== Running command "kubectl --namespace stocktrader apply -f deploy.yaml" ========== deployment "loyalty-level" created service "loyalty-level-service" unchanged ingress "loyalty-ingress" unchanged ========== 10-trader-nodejs - Running dockerbuild, create secrets and docker deploy ========== Create secret for ingress-controller to switch to tradr instead of trader ```

© Copyright IBM Corp. 2018. All rights reserved

Step	Action

```
==========

secret "ingress-host" created
==========
Deploy Liberty Docker container for loyalty
==========
Running command "kubectl --namespace stocktrader apply -f deploy.yaml"
==========
deployment "tradr" created
service "tradr-service" unchanged
ingress "nodejs-trader-ingress" configured
==========
```

__q. Run the commands to see the status of deployments and pods.

__r. Run `kubectl -n stocktrader get deployments`

```
[root@node01 08ms]# kubectl -n stocktrader get deployments
NAME                    DESIRED   CURRENT   UP-TO-DATE
AVAILABLE   AGE
loyalty-level           1         1         1            1
2m
messaging               1         1         1            1
2m
notification-twitter    1         1         1            1
2m
portfolio               1         1         1            1
2m
stockquote              1         1         1            1
2m
trader                  1         1         1            1
2m
tradr                   1         1         1            1
2m
```

__s. Run `kubectl -n stocktrader get pods`

```
[root@node01 08ms]# kubectl -n stocktrader get pods
NAME                                     READY     STATUS
RESTARTS   AGE
loyalty-level-7b58569b9b-f62pt           1/1       Running   0
3m
messaging-559cf6f4cf-rbgf4               1/1       Running   0
3m
notification-twitter-585b96f845-kgjbx    1/1       Running   0
3m
portfolio-7c568d6cb8-s7vc4               1/1       Running   0
3m
stockquote-dbf546b67-fs55t               1/1       Running   0
3m
```

© Copyright IBM Corp. 2018. All rights reserved

Step	Action

```
trader-5c5ff75c5d-r42jh                    1/1       Running   0
3m
tradr-84784b4d9f-j6vfx                     1/1       Running   0
3m
```

5 **Expose Microservice Application**

__a. The entry point for the application that IBM Cloud Team has built starts with the trader microservice using path `/trader/summary` using secured HTTPS port – 9443.

__b. There are multiple ways this application can be run – this is explained to demonstrate how network services work in Kubernetes.

Kubernetes name service.

__c. Run `kubectl -n stocktrader get services`

```
[root@node01 08ms]# kubectl -n stocktrader get services
NAME                    TYPE       CLUSTER-IP   EXTERNAL-IP   PORT(S)
AGE
loyalty-level-service   NodePort   10.0.0.114   <none>
9080:32410/TCP,9443:30472/TCP   6d
notification-service    NodePort   10.0.0.211   <none>
9080:30855/TCP,9443:32737/TCP   6d
portfolio-service       NodePort   10.0.0.195   <none>
9080:32646/TCP,9443:30104/TCP   6d
stock-quote-service     NodePort   10.0.0.62    <none>
9080:32224/TCP,9443:32306/TCP   6d
trader-service          NodePort   10.0.0.68    <none>
9080:32388/TCP,9443:32389/TCP   6d
tradr-service           NodePort   10.0.0.99    <none>        3000:31007/TCP
6d
```

__d. Note the name of the trader service – which is `trader-service` using NodePort and `http` port `9080` is mapped to Node Port `32388` and HTTPS port `9443` mapped as `32389`. We have explicitly defined these ports through deploy.yaml and you will see the same values when you run the command in your lab environment.

__e. The name `trader-service` is in name space stocktrader so the Kubernetes fully qualified domain name (FQDN) will be `trader-service.stocktrader.svc.cluster.local`

__f. You can run this application from within ICP cluster as `https://trader-service.stocktrader.svc.cluster.local:9443/trader/summary`

__g. Note that we have used the local port since we are using local service name. The local port and local Kubernetes FQDN are not visible outside the cluster.

© Copyright IBM Corp. 2018. All rights reserved

Cluster IP address

__h. You can use cluster IP address by examining the output of `kubectl -n stocktrader get service trader-service`

```
[root@node01 08ms]# kubectl -n stocktrader get service trader-service
NAME              TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)
AGE
trader-service    NodePort    10.0.0.68     <none>         9080:32388/TCP,9443:32389/TCP
6d
```

__i. The cluster IP address is 10.0.0.68 (It may be different in your case).

__j. The URL to access the application can be `https://10.0.0.68:9443/trader/summary`

__k. Note that we use a local port as we have direct access to the local IP address of the pod.

Host Names or Proxy Server

__l. To access this application from outside the IBM Cloud Private cluster, one has to come through the proxy server. In our lab environment, the node01 (192.168.142.101) is the proxy server.

__m. The URL to access the application is `https://192.168.142.101:32389/trader/summary`

__n. Note that we are using the node port when accessing the application through proxy server.

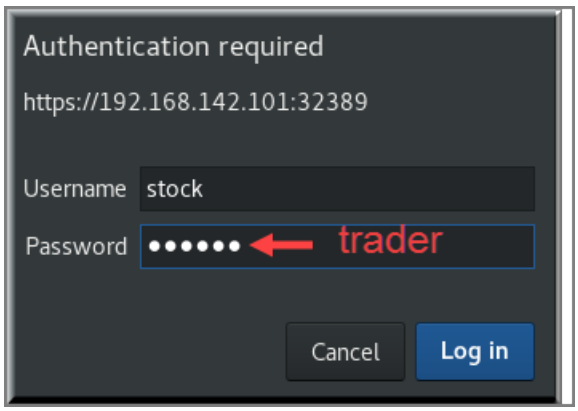
__o. Normally, access to the master and workers nodes is prohibited in the actual environment. But in our environment, you have access to these nodes from outside. You could use any nodes and the routing is handled by Kubernetes automatically. For example: You could use URL `https://192.168.142.103:32389/trader/summary` and the routing to the appropriate pod is automatic.

> **Note:** The internal routing is managed by iptables rules defined by the Kubernetes when a network service is defined.

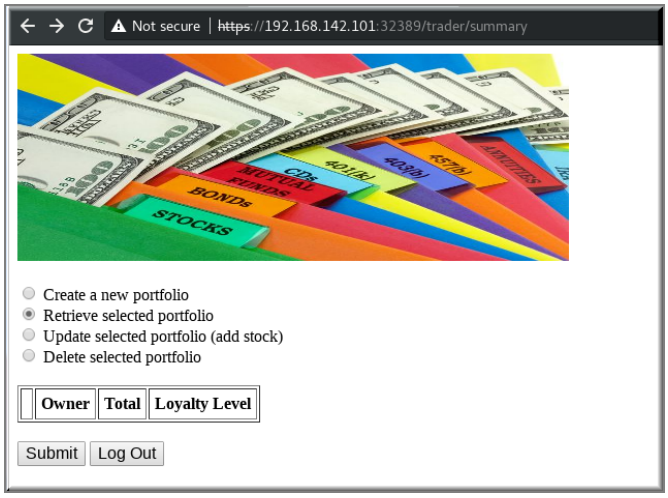
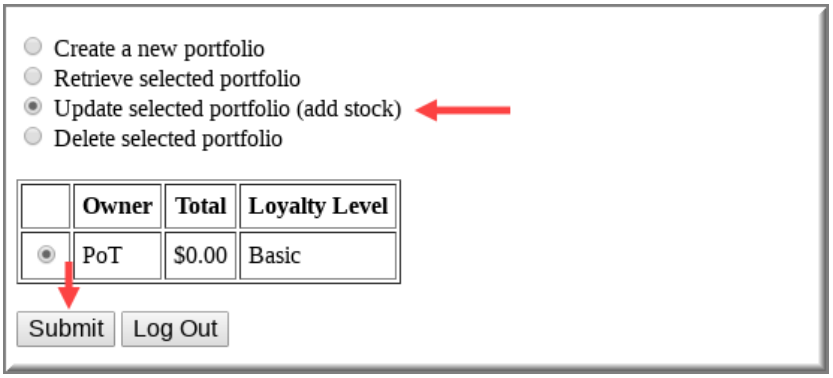
Ingress Service

__p. An Ingress is a Kubernetes resource that lets you configure an HTTP load balancer for your Kubernetes services. Such a load balancer usually exposes your services to clients outside of your Kubernetes cluster. In other words, Kubernetes ingress is a

© Copyright IBM Corp. 2018. All rights reserved

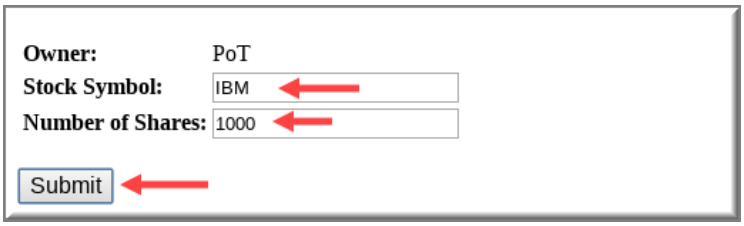
Step	Action
	collection of routing rules that govern how external users access services running in a Kubernetes cluster.
6	**<u>Run the Microservice Application</u>** __a. Open a new browser tab to run the application. __b. Type URL: `https://192.168.142.101:32389/trader/summary` __c. Type Username `stock` and Password `trader`. Click `Log in`.  __d. You should see the main summary page. Note that this is the server JSP with no use of client-side scripting and typically represents a legacy UI. This page is serviced by the `trader` microservice. Later, we will see Node.js-based web UI which can be plugged in to show the strengths of the microservices-based architecture in which the UI can be independent of the model and controller and easily replaceable.

© Copyright IBM Corp. 2018. All rights reserved

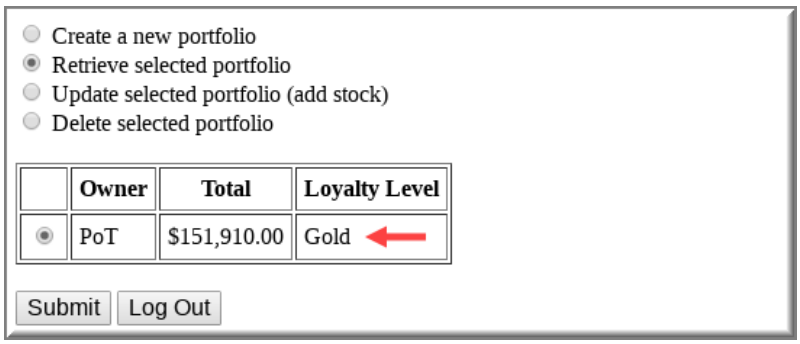
Step	Action
	 __a. Tick `Create a new portfolio`. Click `Submit`. __b. Type Owner `PoT` and click `submit`. __c. Tick `Update selected portfolio (add stock).` Click `Submit.`  __d. Note: You need an Internet connection as this request routes to `https://www.quandl.com/` to retrieve the stock quote at the end of the previous day. __e. Type Stock Symbol `IBM` and specify `1000` stocks. Click `Submit`.

© Copyright IBM Corp. 2018. All rights reserved

Step	Action



__f. The `Loyalty Level` changes to `Gold` with the following screen.



__g. Open a new tab in the browser and type URL `https://twitter.com/ibmstocktrader` and you should see the message posted at the Twitter site.

__h. If you do not see the Twitter message, check the logs of the `messaging` microservice.

__i. Run `kubectl -n stocktrader get pods`

© Copyright IBM Corp. 2018. All rights reserved

Step	Action

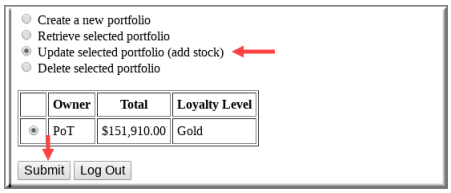
```
[root@node01 08ms]# kubectl -n stocktrader get pods
NAME                                      READY     STATUS
RESTARTS   AGE
loyalty-level-7b58569b9b-sfg8g            1/1       Running   2
5h
messaging-559cf6f4cf-h6kzx                1/1       Running   0
2m
notification-twitter-585b96f845-9nv85     1/1       Running   2
5h
portfolio-7c568d6cb8-6bkg4                1/1       Running   2
5h
stockquote-dbf546b67-4ldh7                1/1       Running   2
5h
trader-5c5ff75c5d-lgnpq                   1/1       Running   2
5h
tradr-84784b4d9f-kpckw                    1/1       Running   2
5h
```

__j. Note the messaging pod name and get logs.

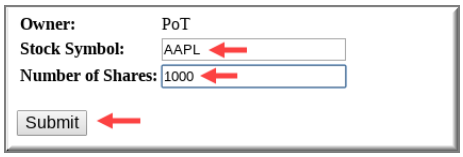
__k. Run `kubectl -n stocktrader logs messaging-559cf6f4cf-h6kzx`

__l. Change last two suffix verbs, as per your output.

__m. Select `Update selected portfolio (add stock)`



__n. Add 1000 stock shares to your portfolio for AAPL.



__o. Switch to the command line.

7	**Explore Db2 Records**

__a. Run `kubectl -n default get pods`

```
[root@node01 08ms]# kubectl -n default get pods
```

© Copyright IBM Corp. 2018. All rights reserved

Step	Action

```
NAME                                                READY     STATUS    RESTARTS
AGE
dev-ibm-db2oltp-dev-0                               1/1       Running   0
9h
helm-local-repo-crm8v                               1/1       Running   7
7d
qdev-ibm-mq-0                                       1/1       Running   8
6d
rdev-ibm-redis-ha-dev-sentinel-5cfc58cb87-677sd     1/1       Running   7
7d
rdev-ibm-redis-ha-dev-server-5ff558dd6f-chvgg       1/1       Running   7
7d
```

__b. Note the name of the Db2 pod and we will use this name in next command.

__c. Run `kubectl -n default exec -it dev-ibm-db2oltp-dev-0 su - db2psc`

```
# kubectl -n default exec -it dev-ibm-db2oltp-dev-0 su - db2psc
Last login: Tue Apr 17 00:33:22 UTC 2018
```

__d. You are inside the Db2 container, logged in as `db2psc` instance user.

__e. Run `db2 connect to PSDB` to connect to PSDB database.

```
[db2psc@dev-ibm-db2oltp-dev-0 ~]$ db2 connect to PSDB

   Database Connection Information

 Database server        = DB2/LINUXX8664 11.1.3.3
 SQL authorization ID   = DB2PSC
 Local database alias   = PSDB
```

__f. Run the following commands: 1. `db2 list tables`, 2. `db2 "select * from stock"` and 3. `db2 "select * from portfolio"`

```
[db2psc@dev-ibm-db2oltp-dev-0 ~]$ db2 list tables

Table/View                      Schema          Type  Creation time
------------------------------- --------------- ----- --------------------------
PORTFOLIO                       DB2PSC          T     2018-04-17-00.07.51.819260
STOCK                           DB2PSC          T     2018-04-17-00.07.52.795422

  2 record(s) selected.

[db2psc@dev-ibm-db2oltp-dev-0 ~]$ db2 "select * from stock"

OWNER                            SYMBOL   SHARES      PRICE                    TOTAL                    DATEQUOTED
-------------------------------- -------- ----------- ------------------------ ------------------------ ----------
PoT                              IBM            1000   +1.51910000000000E+002   +1.51910000000000E+005 03/27/2018
PoT                              AAPL           1000   +1.68340000000000E+002   +1.68340000000000E+005 03/27/2018

  1 record(s) selected.

[db2psc@dev-ibm-db2oltp-dev-0 ~]$ db2 "select * from portfolio"

OWNER                            TOTAL                    LOYALTY
-------------------------------- ------------------------ --------
PoT                                +3.20250000000000E+005 Gold

  1 record(s) selected.
```

© Copyright IBM Corp. 2018. All rights reserved

Step	Action

	__g. Type `exit` to log out from the container.
	`[db2psc@dev-ibm-db2oltp-dev-0 ~]$ exit`
	`logout`

8	**Explore Redis Records**
	
	__a. Run `kubectl -n default get pods`
	`[root@node01 08ms]# kubectl -n default get pods`
	`NAME READY STATUS RESTARTS AGE`
	`dev-ibm-db2oltp-dev-0 1/1 Running 2 4h`
	`helm-local-repo-fj9cj 1/1 Running 4 8h`
	`qdev-ibm-mq-0 1/1 Running 0 1h`
	`rdev-ibm-redis-ha-dev-sentinel-68db4dc96-9lgkr 1/1 Running 0 57m`
	`rdev-ibm-redis-ha-dev-sentinel-68db4dc96-g4zvd 1/1 Running 0 57m`
	`rdev-ibm-redis-ha-dev-sentinel-68db4dc96-qsgz6 1/1 Running 0 57m`
	`rdev-ibm-redis-ha-dev-server-85d8f665d-2vpfk 1/1 Running 0 57m`
	`rdev-ibm-redis-ha-dev-server-85d8f665d-77t55 1/1 Running 0 57m`
	`rdev-ibm-redis-ha-dev-server-85d8f665d-q85kw 1/1 Running 0 57m`
	
	__b. We have three copies of the `redis` server running. How do we know which one is the master?
	
	__c. Run `kubectl -n default get pods -l redis-role=master`
	`[root@node01 08ms]# kubectl -n default get pods -l redis-role=master`
	`NAME READY STATUS RESTARTS AGE`
	`rdev-ibm-redis-ha-dev-server-85d8f665d-77t55 1/1 Running 0 59m`
	
	__d. Highlight the `redis-ha-dev-server` and select the full name to copy.
	
	__e. Run `kubectl -n default exec -it rdev-ibm-redis-ha-dev-server-85d8f665d-77t55 bash`
	`# kubectl -n default exec -it rdev-ibm-redis-ha-dev-server-85d8f665d-77t55 bash`
	`bash-4.4#`

© Copyright IBM Corp. 2018. All rights reserved

Step	Action

	__f. Replace the suffix in the above-mentioned name as per the output in your command line.
	
	__g. Run `redis-cli ping` and it should return the response as `pong`.
	`bash-4.4# redis-cli ping`
	`pong`
	`127.0.0.1:6379>`
	
	__h. Run `redis-cli` to get the command line prompt.
	
	__i. Type `info`
	
	__j. Scroll through the `Redis` server statistics.
	
	__k. Type `keys *`
	`127.0.0.1:6379> keys *`
	`1) "AAPL"`
	`2) "IBM"`
	`127.0.0.1:6379>`
	
	__l. Note IBM and AAPL stock quotes cached in the `Redis` server.
	
	__m. Type `get IBM` and `get AAPL` to see the cached values.
	`127.0.0.1:6379> get IBM`
	`"{\"symbol\":\"IBM\",\"date\":\"2018-03-27\",\"price\":151.91}"`
	`127.0.0.1:6379> get AAPL`
	`"{\"symbol\":\"AAPL\",\"date\":\"2018-03-27\",\"price\":168.34}"`
	
	__n. Type `exit` to quit `redis-cli` and `exit` again to quit the Redis container.
	`127.0.0.1:6379> exit`
	`bash-4.4# exit`
	`exit`
	`[root@node01 08ms]#`
	
	__o. Note that we run only one Redis server and one sentinel (replicated) server. In actual environment, we would run minimum 3 Redis server and 3 sentinel servers.

© Copyright IBM Corp. 2018. All rights reserved