IBM Cloud

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

Hands-on Workshop

Lab Guide

Notices and Disclaimers

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

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

Section 1: Lab Instructions

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>

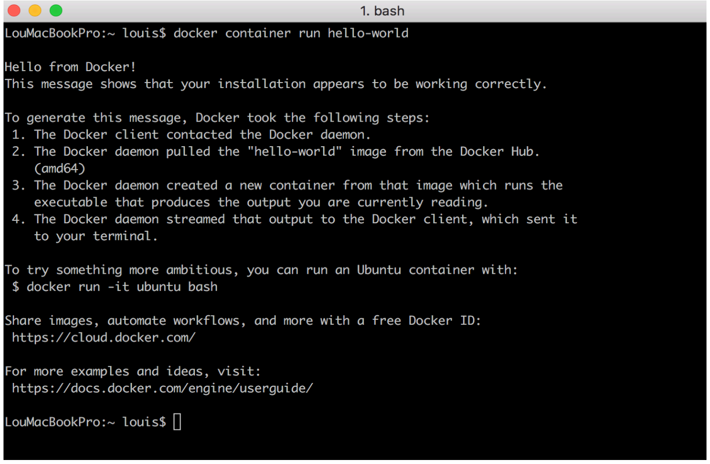
Login to the Docker Environment

__ 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:



```
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$ 
```

</td></tr>
<tr><td>2</td><td>

Build a Docker Image for an Application

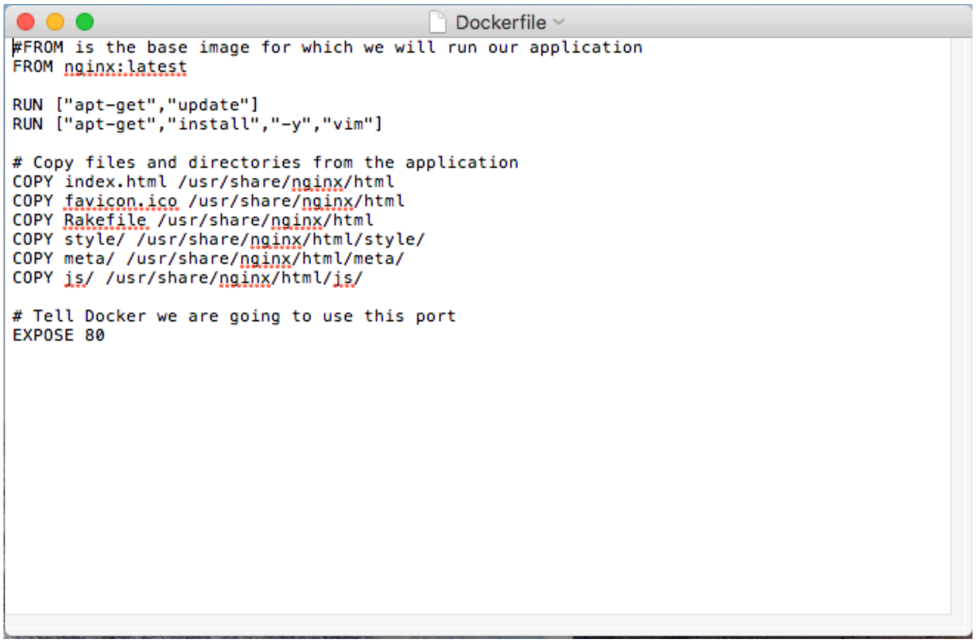
__a. Before we can work with a container, we will need to first build an image for our 2048 application. First, we will make of copy of the application code to your home directory:

```
~$ 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:

</td></tr>
</table>

Step	Action



```
#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
```

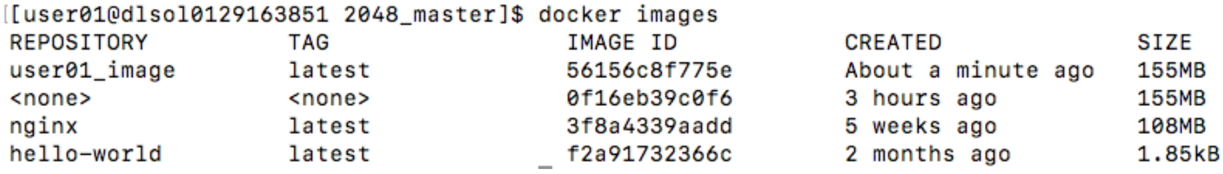
The commands in this file instruct Docker to use a simple web service (nginx) as a base image (nginx is automatically pulled from Docker Hub when the image is built. The file then copies the application code into a directory structure within the image (in /usr/share). Finally, port 80 is exposed in order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

Step	Action

3

Run a Container

__a. Now that you have an image, we will now run the 2048 application as a container. To do this, run the following command:

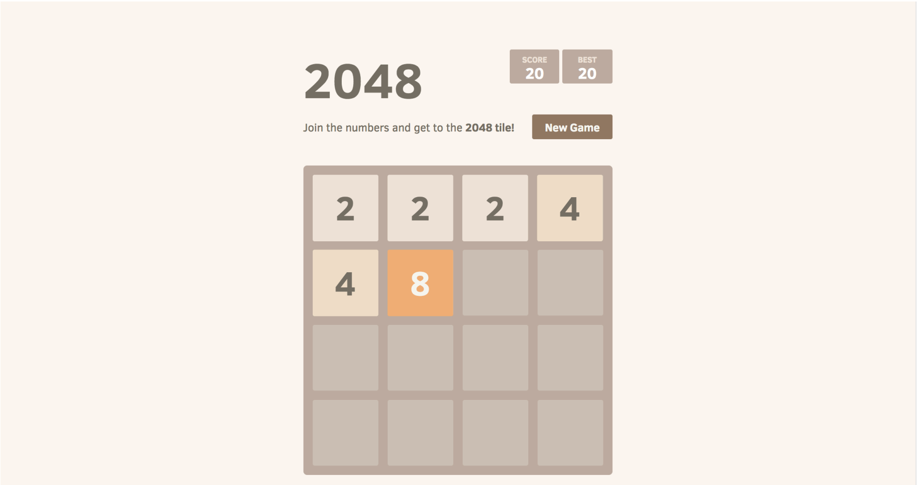
Your instructor will assign you a port a unique port number to use for the remained of the lab.

```
~$ 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

Stop/Delete a Container

__ a. You can stop the container by typing cntrl-c
~$ <Cntrl-c>

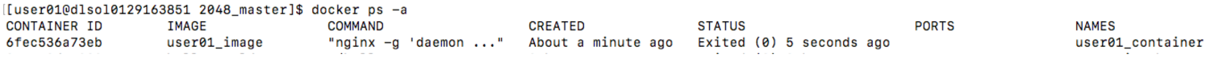
Step | Action

__ 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

Inspect a Running Container

__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
```

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

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.

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

Section 2: Lab Instructions

Step	Action
1	**Docker Volumes**

__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:

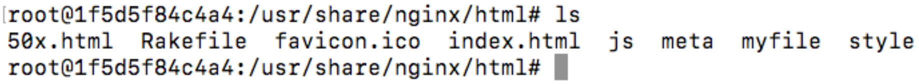
```
~$ docker container exec -it 2048_container bash
# cd /usr/share/nginx/html
```

__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
```

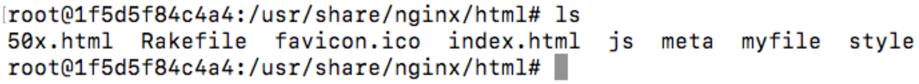
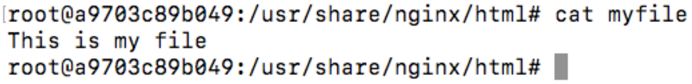
__d. We will now remove the container using the command:

```
~$ 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
```

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

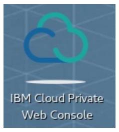
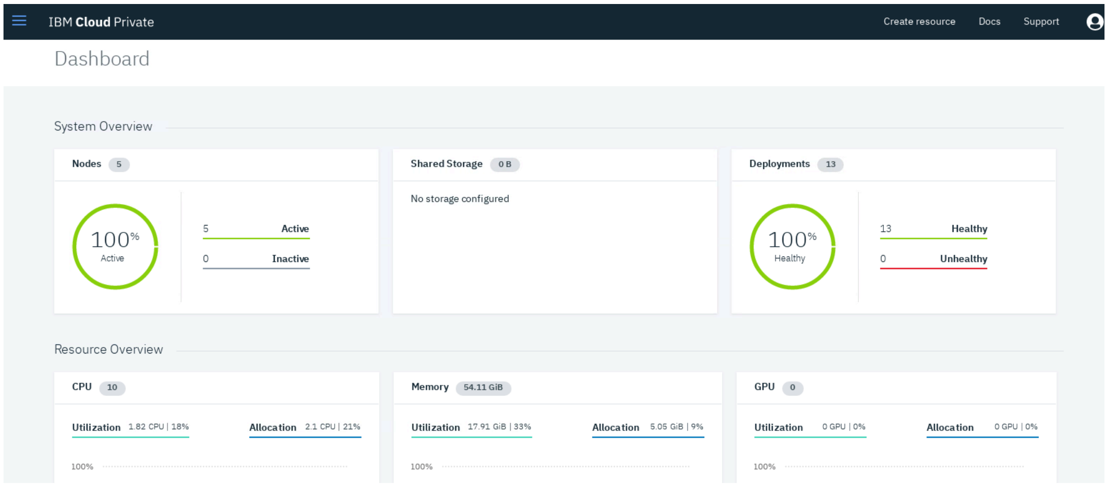
In this lab you were introduced to one way to persist data on the host file system. With volumes the container references a volume object on the local file system.

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)

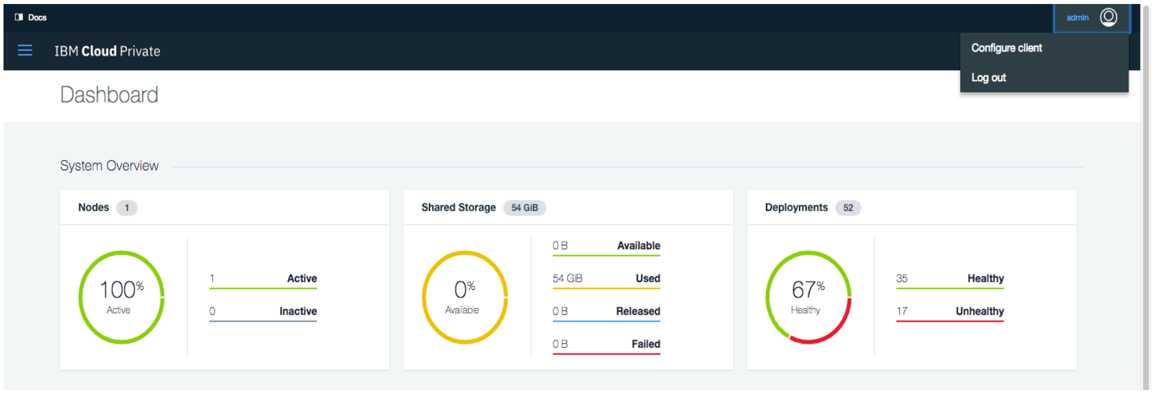
Tasks:	Tasks you will complete in this lab exercise include: • Access the IBM Cloud Private Dashboard • Access the ICP Kubernetes configuration settings • Configure your environment to use the ICP cluster

Section 3: Lab Instructions

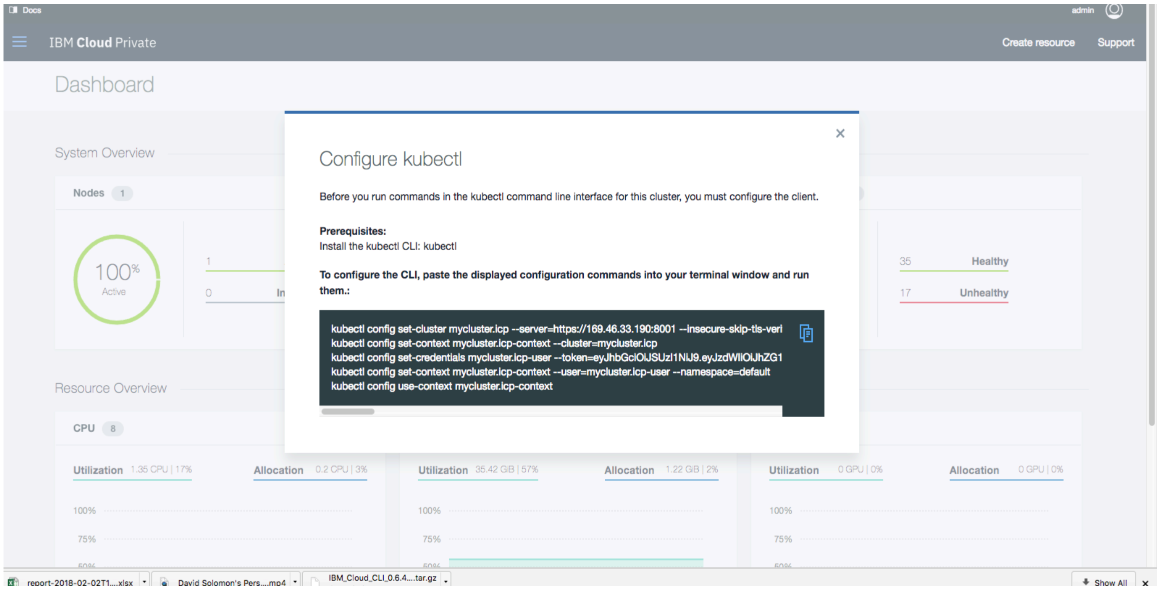
Step	Action
1	**Launch the ICP Dashboard** a. ICP has a centralized dashboard and control center. This dashboard is similar to the classic Kubernetes dashboard but provides additional enterprise services and features (e.g, data science, security). Open the dashboard by double clicking on the Web Console icon on the desktop.  Login with username: admin/ password: admin. Click on the hamburger menu at the top and select “Dashboard”  IBM Cloud Private Create resource Docs Support Dashboard System Overview Nodes 5 100% Active 5 Active 0 Inactive Shared Storage 0 B No storage configured Deployments 13 100% Healthy 13 Healthy 0 Unhealthy Resource Overview CPU 10 Utilization 1.82 CPU | 18% Allocation 2.1 CPU | 21% 100% Memory 54.11 GiB Utilization 17.91 GiB | 33% Allocation 5.05 GiB | 9% 100% GPU 0 Utilization 0 GPU | 0% Allocation 0 GPU | 0% 100% You will notice that this ICP instance is a 5-node Kubernetes cluster.
2	**Configure your Environment for ICP** 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

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.



Configure kubectl

Before you run commands in the kubectl command line interface for this cluster, you must configure the client.

Prerequisites:
Install the kubectl CLI: kubectl

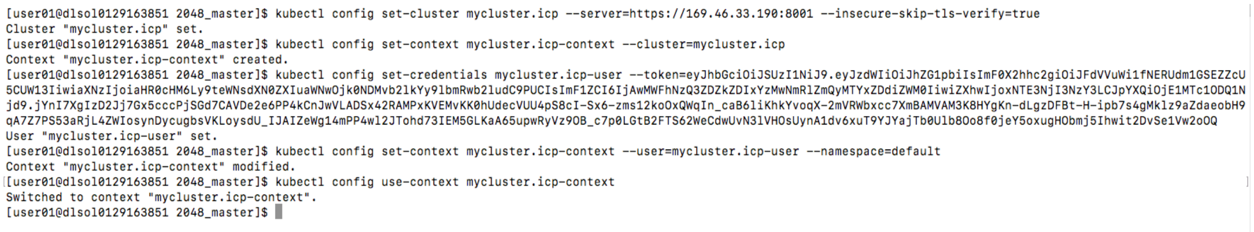
To configure the CLI, paste the displayed configuration commands into your terminal window and run them.:

```
kubectl config set-cluster mycluster.icp --server=https://169.46.33.190:8001 --insecure-skip-tls-veri
kubectl config set-context mycluster.icp-context --cluster=mycluster.icp
kubectl config set-credentials mycluster.icp-user --token=eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJhZG1
kubectl config set-context mycluster.icp-context --user=mycluster.icp-user --namespace=default
kubectl config use-context mycluster.icp-context
```

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

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`
	`5CUW13IiwiaXNzIjoiaHR0cHM6Ly9teWNsdXN0ZXIuaWNwOjk0NDMvb2lkYy9lbmRwb2ludC9PUCIsImF1ZCI6IjAwMWFhNzQ3ZDZkZDIxYzMwNmRlZmQyMTYxZDdiZWM0IiwiZXhwIjoxNTE3NjI3NzY3LCJpYXQiOjE1MTc1ODQ1N`
	`jd9.jYnI7XgIzD2Jj7Gx5cccPjSGd7CAVDe2e6PP4kCnJwVLADSx42RAMPxKVEMvKK0hUdecVUU4pS8cI-Sx6-zms12koOxQWqIn_caB6liKhkYvoqX-2mVRWbxcc7XmBAMVAM3K8HYgKn-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.

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

Section 4: Lab Instructions

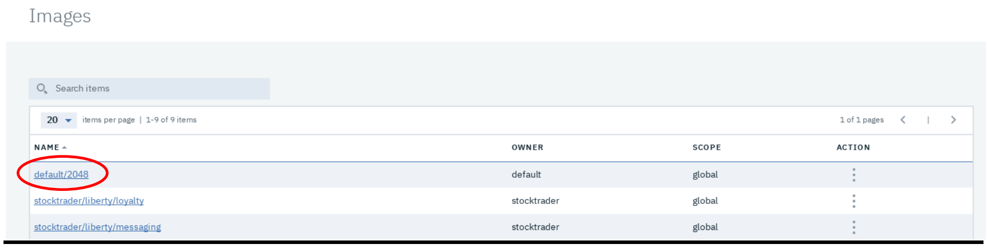
Step	Action
1	**Copy the 2048 Image to the ICP Private Docker Registry**

__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.



Images

Search items

20 items per page | 1-9 of 9 items — 1 of 1 pages

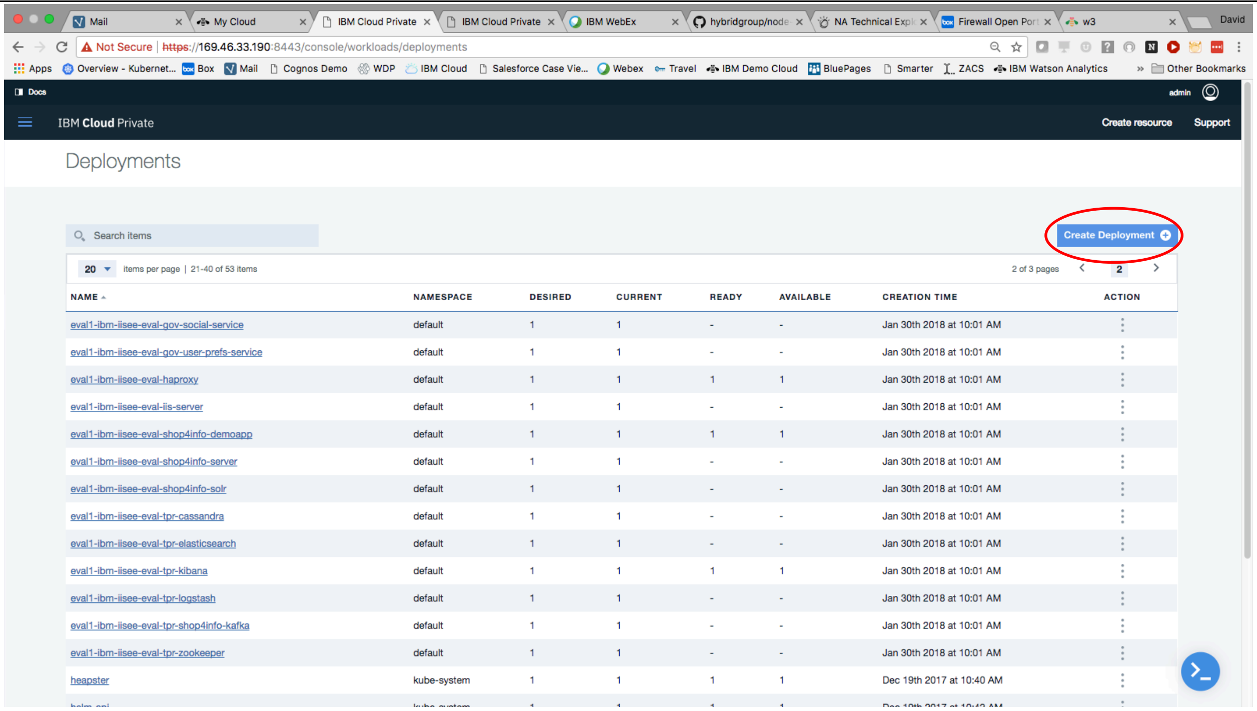
NAME	OWNER	SCOPE	ACTION
default/2048	default	global	⋮
stocktrader/liberty/loyalty	stocktrader	global	⋮
stocktrader/liberty/messaging	stocktrader	global	⋮

Step	Action
1	**Create a new deployment**

__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:

Step	Action



Deployments

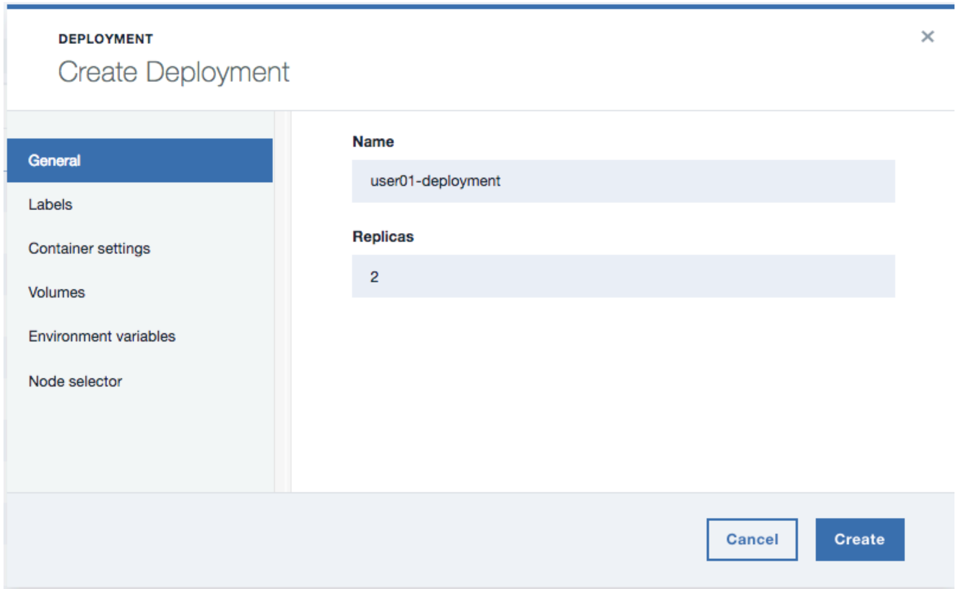
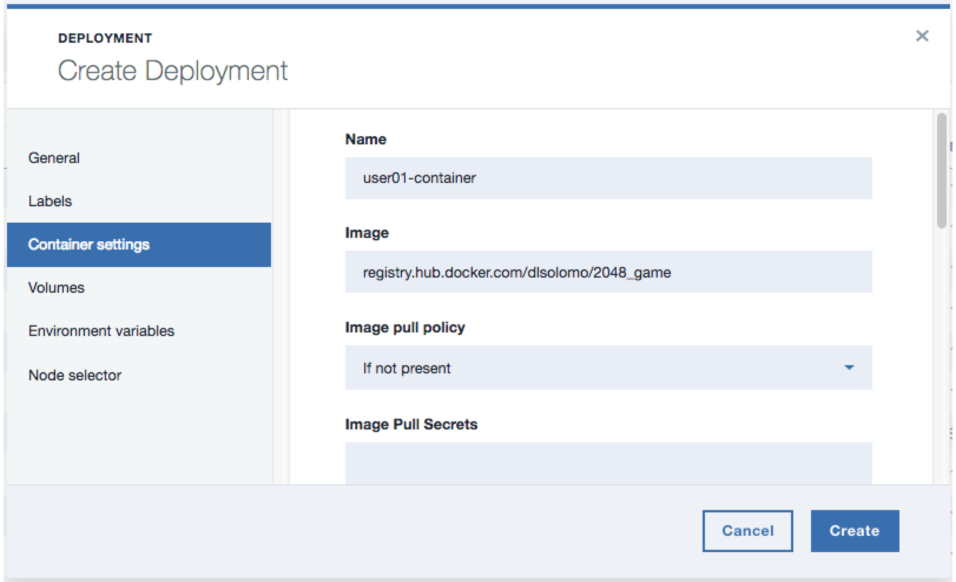
20 items per page | 21-40 of 53 items — 2 of 3 pages — 2

NAME	NAMESPACE	DESIRED	CURRENT	READY	AVAILABLE	CREATION TIME	ACTION
eval1-ibm-iisee-eval-gov-social-service	default	1	1	-	-	Jan 30th 2018 at 10:01 AM	⋮
eval1-ibm-iisee-eval-gov-user-prefs-service	default	1	1	-	-	Jan 30th 2018 at 10:01 AM	⋮
eval1-ibm-iisee-eval-haproxy	default	1	1	1	1	Jan 30th 2018 at 10:01 AM	⋮
eval1-ibm-iisee-eval-iis-server	default	1	1	-	-	Jan 30th 2018 at 10:01 AM	⋮
eval1-ibm-iisee-eval-shop4info-demoapp	default	1	1	1	1	Jan 30th 2018 at 10:01 AM	⋮
eval1-ibm-iisee-eval-shop4info-server	default	1	1	-	-	Jan 30th 2018 at 10:01 AM	⋮
eval1-ibm-iisee-eval-shop4info-solr	default	1	1	-	-	Jan 30th 2018 at 10:01 AM	⋮
eval1-ibm-iisee-eval-tpr-cassandra	default	1	1	-	-	Jan 30th 2018 at 10:01 AM	⋮
eval1-ibm-iisee-eval-tpr-elasticsearch	default	1	1	-	-	Jan 30th 2018 at 10:01 AM	⋮
eval1-ibm-iisee-eval-tpr-kibana	default	1	1	1	1	Jan 30th 2018 at 10:01 AM	⋮
eval1-ibm-iisee-eval-tpr-logstash	default	1	1	-	-	Jan 30th 2018 at 10:01 AM	⋮
eval1-ibm-iisee-eval-tpr-shop4info-kafka	default	1	1	-	-	Jan 30th 2018 at 10:01 AM	⋮
eval1-ibm-iisee-eval-tpr-zookeeper	default	1	1	-	-	Jan 30th 2018 at 10:01 AM	⋮
heapster	kube-system	1	1	1	1	Dec 19th 2017 at 10:40 AM	⋮

__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

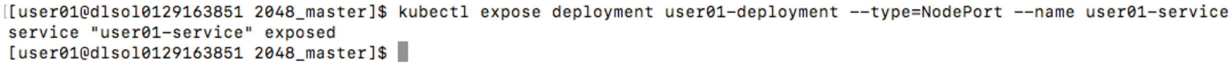
In the “Container settings” tab:
Name=2048-container
Image= poticpcluster.icp:8500/default/2048:1.0

Step	Action
	 DEPLOYMENT Create Deployment General Labels Container settings Volumes Environment variables Node selector Name user01-deployment Replicas 2 Cancel Create Click “Create”.  DEPLOYMENT Create Deployment General Labels Container settings Volumes Environment variables Node selector Name user01-container Image registry.hub.docker.com/dlsolomo/2048_game Image pull policy If not present Image Pull Secrets Cancel Create
2	**Exposing the application through a service** __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.

Step	Action

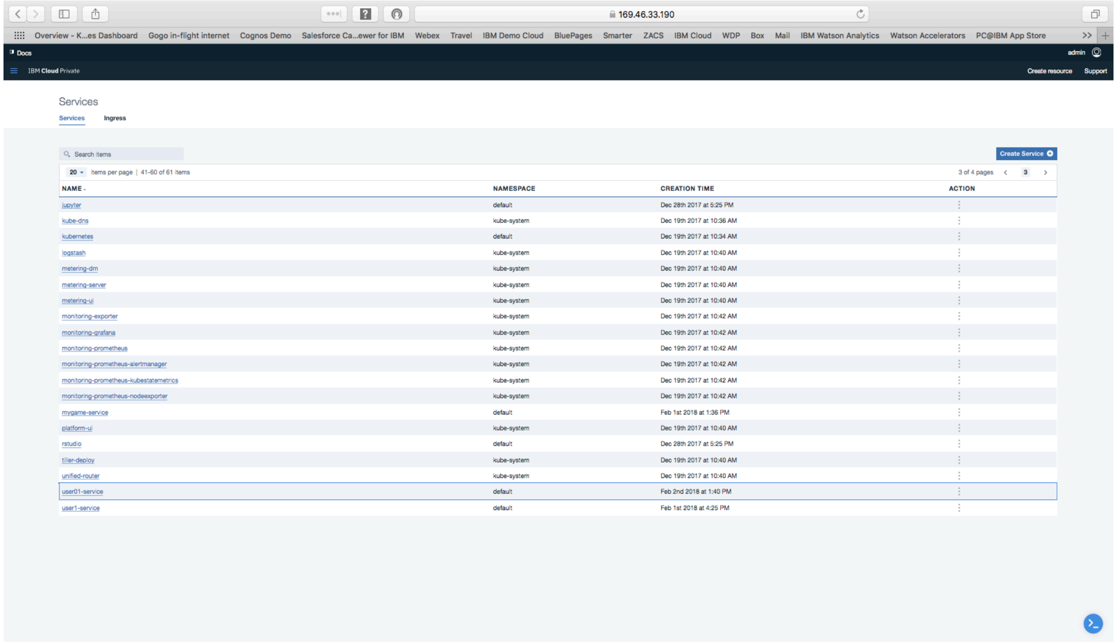
~$ kubectl expose deployment 2048-deployment --type=NodePort --name *2048-service*

__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]$ 
```

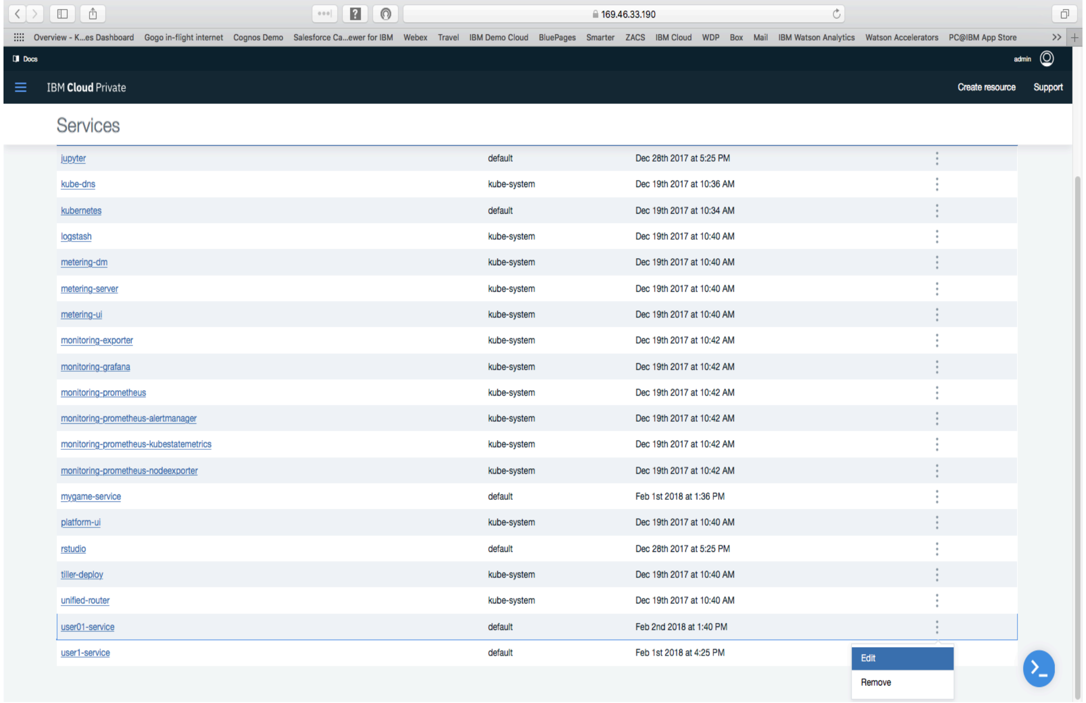
__c. Return to the ICP dashboard. Under the “Workloads” menu option, select “Services”. The list of services will appear. Confirm your service (you may have to navigate to the 3rd or 4th page) is listed.



20 items per page | 41-60 of 61 items   3 of 4 pages < 3 >

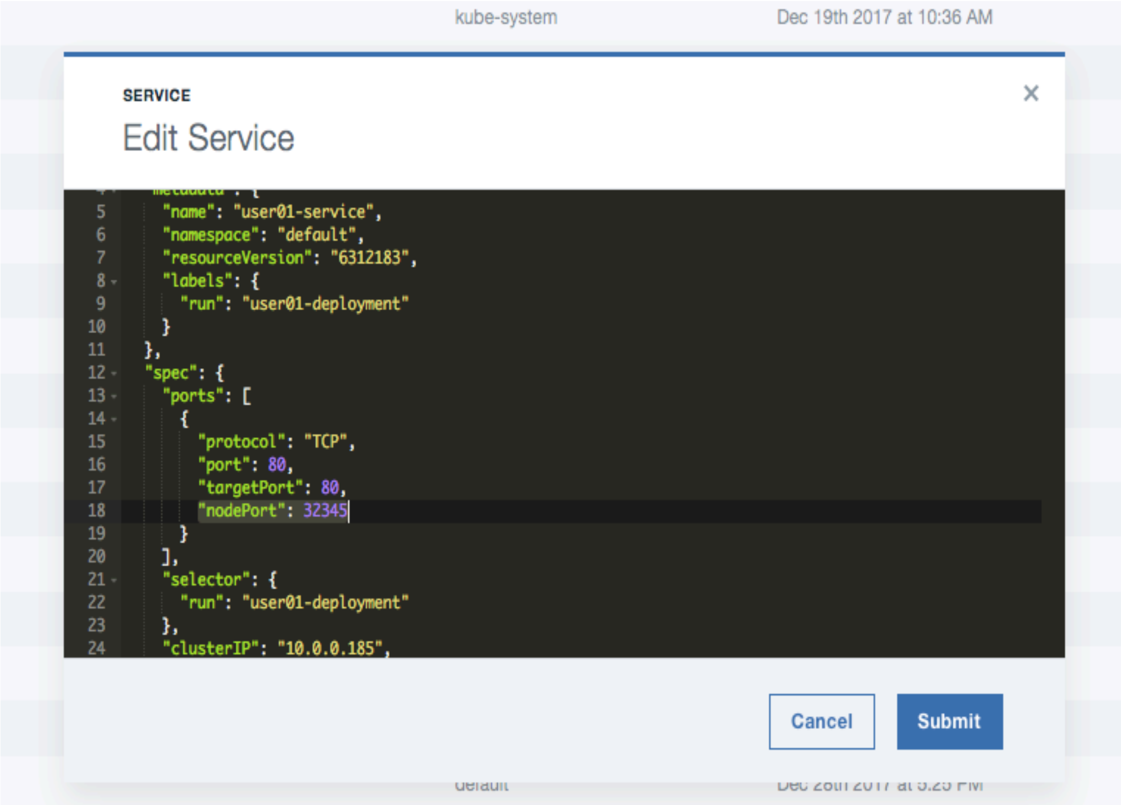
NAME	NAMESPACE	CREATION TIME	ACTION
jupyter	default	Dec 28th 2017 at 5:25 PM	⋮
kube-dns	kube-system	Dec 19th 2017 at 10:36 AM	⋮
kubernetes	default	Dec 19th 2017 at 10:34 AM	⋮
logstash	kube-system	Dec 19th 2017 at 10:40 AM	⋮
metering-dm	kube-system	Dec 19th 2017 at 10:40 AM	⋮
metering-server	kube-system	Dec 19th 2017 at 10:40 AM	⋮
metering-ui	kube-system	Dec 19th 2017 at 10:40 AM	⋮
monitoring-exporter	kube-system	Dec 19th 2017 at 10:42 AM	⋮
monitoring-grafana	kube-system	Dec 19th 2017 at 10:42 AM	⋮
monitoring-prometheus	kube-system	Dec 19th 2017 at 10:42 AM	⋮
monitoring-prometheus-alertmanager	kube-system	Dec 19th 2017 at 10:42 AM	⋮
monitoring-prometheus-kubestatemetrics	kube-system	Dec 19th 2017 at 10:42 AM	⋮
monitoring-prometheus-nodeexporter	kube-system	Dec 19th 2017 at 10:42 AM	⋮
mygame-service	default	Feb 1st 2018 at 1:36 PM	⋮
platform-ui	kube-system	Dec 19th 2017 at 10:40 AM	⋮
rstudio	default	Dec 28th 2017 at 5:25 PM	⋮
tiller-deploy	kube-system	Dec 19th 2017 at 10:40 AM	⋮
unified-router	kube-system	Dec 19th 2017 at 10:40 AM	⋮
user01-service	default	Feb 2nd 2018 at 1:40 PM	⋮
user1-service	default	Feb 1st 2018 at 4:25 PM	⋮

__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.

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td></td><td>


Services
<table>
<tr><td>jupyter</td><td>default</td><td>Dec 28th 2017 at 5:25 PM</td></tr>
<tr><td>kube-dns</td><td>kube-system</td><td>Dec 19th 2017 at 10:36 AM</td></tr>
<tr><td>kubernetes</td><td>default</td><td>Dec 19th 2017 at 10:34 AM</td></tr>
<tr><td>logstash</td><td>kube-system</td><td>Dec 19th 2017 at 10:40 AM</td></tr>
<tr><td>metering-dm</td><td>kube-system</td><td>Dec 19th 2017 at 10:40 AM</td></tr>
<tr><td>metering-server</td><td>kube-system</td><td>Dec 19th 2017 at 10:40 AM</td></tr>
<tr><td>metering-ui</td><td>kube-system</td><td>Dec 19th 2017 at 10:40 AM</td></tr>
<tr><td>monitoring-exporter</td><td>kube-system</td><td>Dec 19th 2017 at 10:42 AM</td></tr>
<tr><td>monitoring-grafana</td><td>kube-system</td><td>Dec 19th 2017 at 10:42 AM</td></tr>
<tr><td>monitoring-prometheus</td><td>kube-system</td><td>Dec 19th 2017 at 10:42 AM</td></tr>
<tr><td>monitoring-prometheus-alertmanager</td><td>kube-system</td><td>Dec 19th 2017 at 10:42 AM</td></tr>
<tr><td>monitoring-prometheus-kubestatemetrics</td><td>kube-system</td><td>Dec 19th 2017 at 10:42 AM</td></tr>
<tr><td>monitoring-prometheus-nodeexporter</td><td>kube-system</td><td>Dec 19th 2017 at 10:42 AM</td></tr>
<tr><td>mygame-service</td><td>default</td><td>Feb 1st 2018 at 1:36 PM</td></tr>
<tr><td>platform-ui</td><td>kube-system</td><td>Dec 19th 2017 at 10:40 AM</td></tr>
<tr><td>rstudio</td><td>default</td><td>Dec 28th 2017 at 5:25 PM</td></tr>
<tr><td>tiller-deploy</td><td>kube-system</td><td>Dec 19th 2017 at 10:40 AM</td></tr>
<tr><td>unified-router</td><td>kube-system</td><td>Dec 19th 2017 at 10:40 AM</td></tr>
<tr><td>user01-service</td><td>default</td><td>Feb 2nd 2018 at 1:40 PM</td></tr>
<tr><td>user1-service</td><td>default</td><td>Feb 1st 2018 at 4:25 PM</td></tr>
</table>
__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.
</td></tr>
</table>

Step	Action



Edit Service

```
    "name": "user01-service",
    "namespace": "default",
    "resourceVersion": "6312183",
    "labels": {
      "run": "user01-deployment"
    }
  },
  "spec": {
    "ports": [
      {
        "protocol": "TCP",
        "port": 80,
        "targetPort": 80,
        "nodePort": 32345
      }
    ],
    "selector": {
      "run": "user01-deployment"
    },
    "clusterIP": "10.0.0.185",
```

__f. Click Cancel.

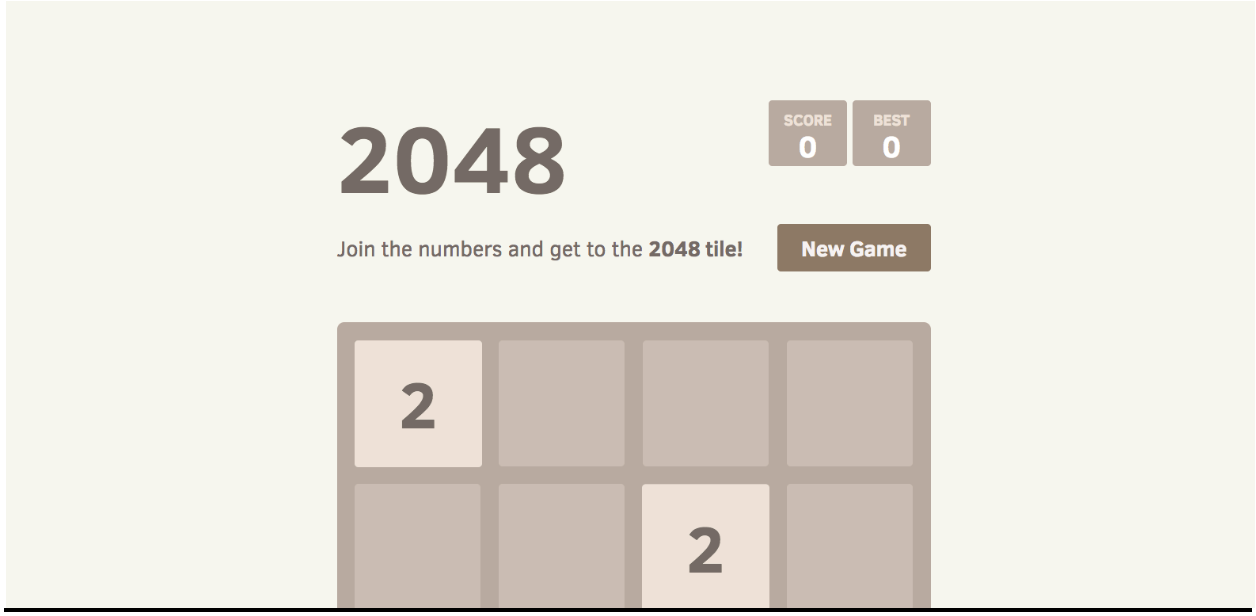
You have now successfully enabled your application running in the Kubernetes cluster to be accessed from the outside.

3

Access the Running Application

__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* >

Step	Action
	 Join the numbers and get to the **2048 tile!**

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.

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

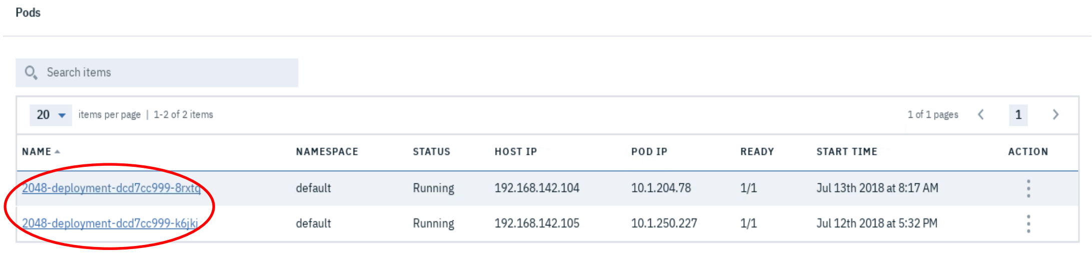
Step	Action

1 **Explore the ReplicaSet Policy**

__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.



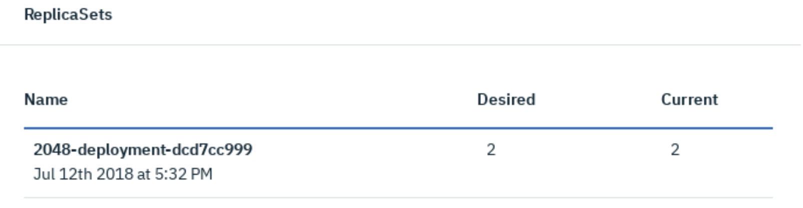
Pods

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-k6jkj	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

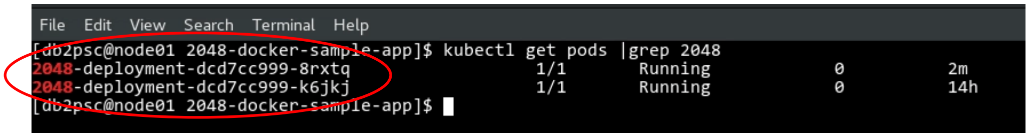
2 **Simulate a Pod Failure**

__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:

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td></td><td>



<pre>
[db2psc@node01 2048-docker-sample-app]$ kubectl get pods |grep 2048
2048-deployment-dcd7cc999-8rxtq   1/1   Running   0   2m
2048-deployment-dcd7cc999-k6jkj   1/1   Running   0   14h
[db2psc@node01 2048-docker-sample-app]$ █
</pre>

__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 <<i>the name of one of your Pods</i>>.

</td></tr>
<tr><td>3</td><td>

<b><u>Observe that the Cluster Recovers from the Failure</u></b>

__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

</td></tr>
</table>

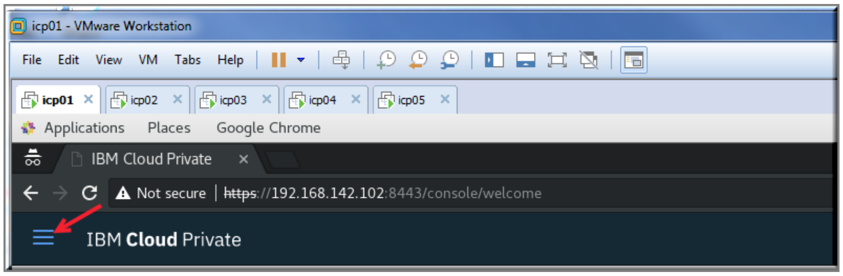
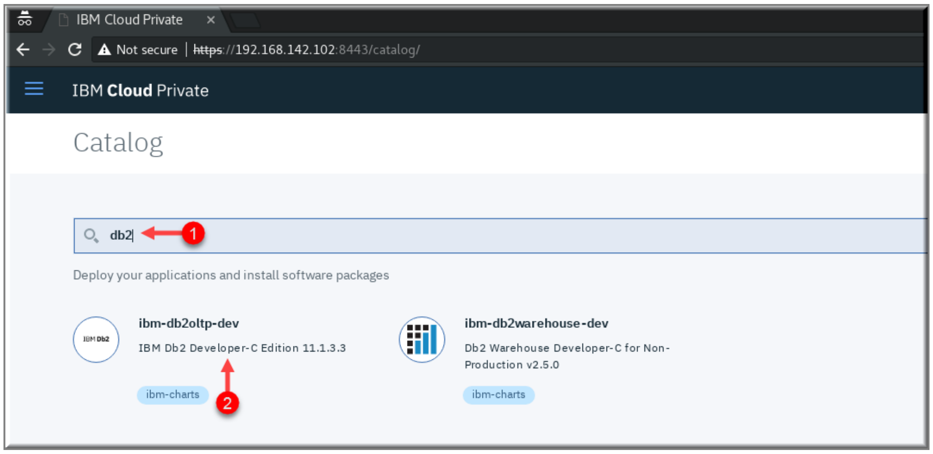
Section 5: Lab Summary

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

Section 6: Deploying Services Using the ICP Catalog

Purpose:	In this lab, you will learn how you can quickly deploy IBM and 3rd-party services using the ICP catalog. The services you will be deploying will be used in a later lab to support our deploying of a full microservices application.

Tasks:	Tasks you will complete in this lab exercise include: • Deploy db2 • Deploy IBM MQ • Deploy Redis for in-memory cache

Step	Action
1	**Deploy db2** __a. Switch to the web GUI of IBM Cloud Private. __b. Click `Hamburger` ☰ as shown. [Remember this for future reference.]  icp01 - VMware Workstation File Edit View VM Tabs Help icp01 icp02 icp03 icp04 icp05 Applications Places Google Chrome IBM Cloud Private Not secure | https://192.168.142.102:8443/console/welcome IBM Cloud Private __c. Expand `Catalog` and click `Helm Charts`. __d. The Catalog page shows software offerings from IBM and as well as Open Source. __e. Type `db2` in the search box to narrow down the search. Click `ibm-db2oltp-dev`  IBM Cloud Private Not secure | https://192.168.142.102:8443/catalog/ IBM Cloud Private Catalog db2 1 Deploy your applications and install software packages ibm-db2oltp-dev IBM Db2 Developer-C Edition 11.1.3.3 ibm-charts 2 ibm-db2warehouse-dev Db2 Warehouse Developer-C for Non-Production v2.5.0 ibm-charts __f. Review the `IBM Db2 Developer-C Helm Chart` page as shown. Read the prerequisites – 1. Docker container and 2. Persistent Storage for Db2 __g. Since Docker containers are ephemeral, we need persistent storage.

Step	Action

The Db2 Helm Chart supports two options for persistent storage.

__1. Using predefined Persistent Volume and Persistent Volume Claim.

__2. Dynamic provisioning

In IBM Cloud Private, the persistent volume support is for HostPath, NFS, vSphere cloud provider and Gluster. IBM Spectrum Scale (GPFS) will be added soon.

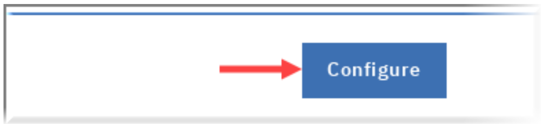
In our lab exercise, we use Gluster dynamic provisioning.

Note that Gluster can have raw volumes from Tier-1 SAN storage. This allows you to deploy production-grade, monolithic databases using the `StatefulSet` capability of the IBM Cloud Private to get all new features and functions of SVC (Service Volume Claim), PVC (Persistent Volume Claim), dynamic provisioning and loose coupling between applications and databases.

The applications can still be microservices based, connecting to tier-1 scalable, production-grade database such as Db2 deployed in the IBM Cloud Private environment.

Configure Db2

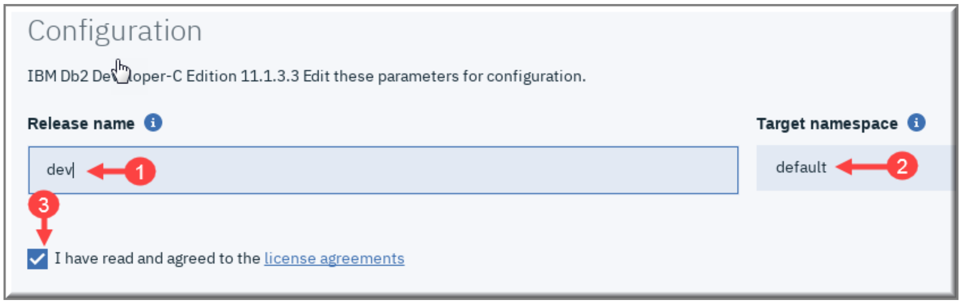
__h. Scroll towards the bottom of the page and click `Configure`.



__i. Enter the Release name **dev**.

__j. Select the Target namespace `default`.

__k. Check the box `I have read and agreed to the License agreements.`



Configuration

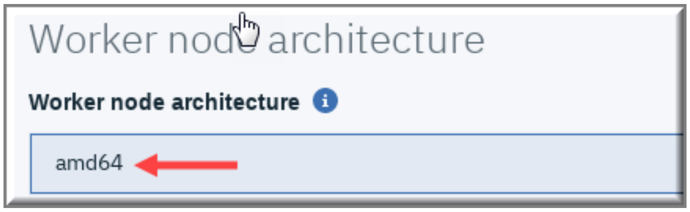
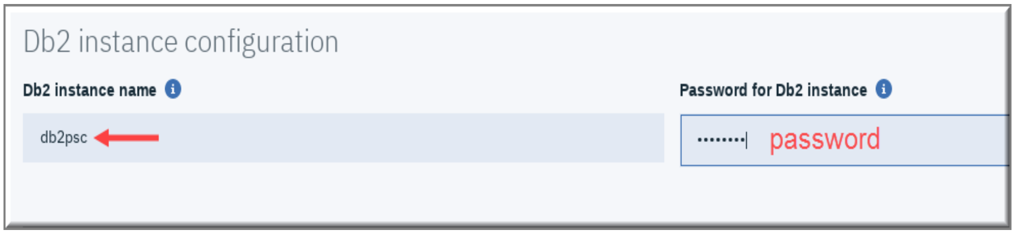
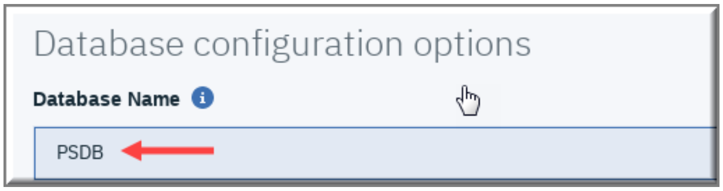
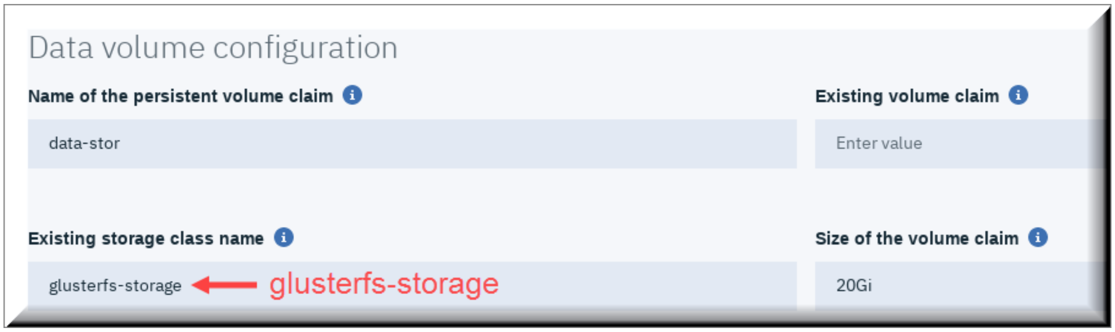
IBM Db2 Developer-C Edition 11.1.3.3 Edit these parameters for configuration.

Release name: dev

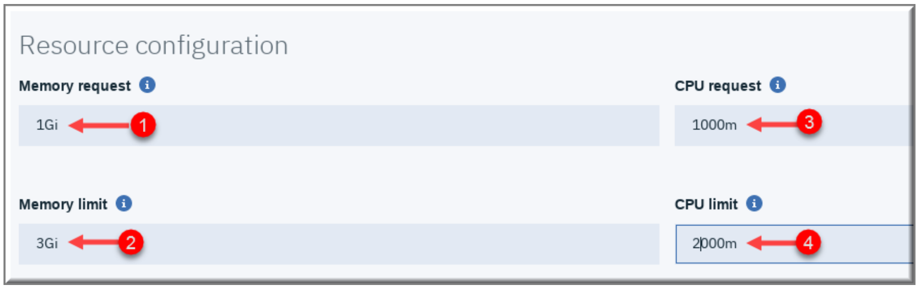
Target namespace: default

☑ I have read and agreed to the license agreements

__l. Scroll to the `Worker node architecture` and select `amd64` from the drop-down menu.

Step	Action
	 Worker node architecture Worker node architecture: amd64 __m. Scroll to the *Db2 instance configuration*, type the *Db2 instance name* `db2psc` and the *Password for Db2 instance* `password`.  Db2 instance configuration Db2 instance name: db2psc Password for Db2 instance: •••••••• password __n. In *Database configuration options*, type *Database Name* `PSDB.`  Database configuration options Database Name: PSDB __o. Scroll to *Data volume configuration*, and type `glusterfs-storage` in *Existing storage class name*. [Note: You must not skip this step.]  Data volume configuration Name of the persistent volume claim: data-stor Existing volume claim: Enter value Existing storage class name: glusterfs-storage glusterfs-storage Size of the volume claim: 20Gi __p. Scroll to the Resource configuration and change the Memory request to 1Gi, Memory limit to 3Gi, CPU request to 1000m and CPU limit to 2000m.

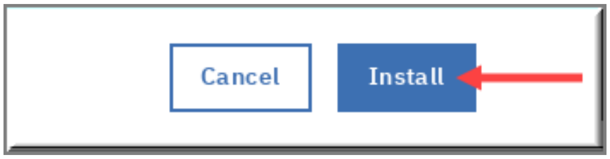
Step	Action



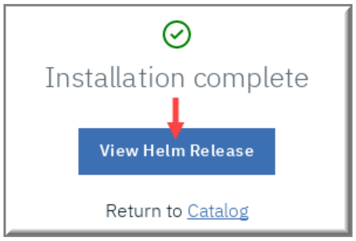
Resource configuration

Memory request: 1Gi
CPU request: 1000m
Memory limit: 3Gi
CPU limit: 2000m

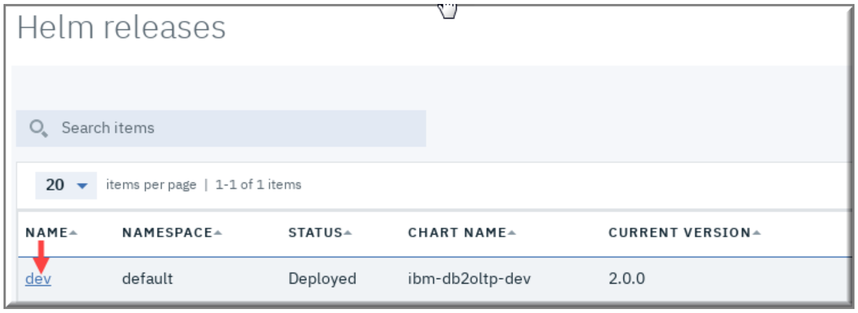
__q. Click Install.



__r. Click View Helm Release.



__s. Click the just-deployed Helm release dev.



Helm releases

Search items

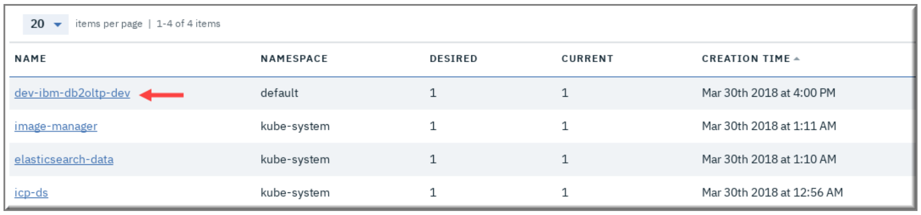
20 items per page | 1-1 of 1 items

NAME	NAMESPACE	STATUS	CHART NAME	CURRENT VERSION
dev	default	Deployed	ibm-db2oltp-dev	2.0.0

__t. Click Hamburger ☰, expand Workloads and click StatefulSets.

__u. Click dev-ibm-db2oltp-dev.

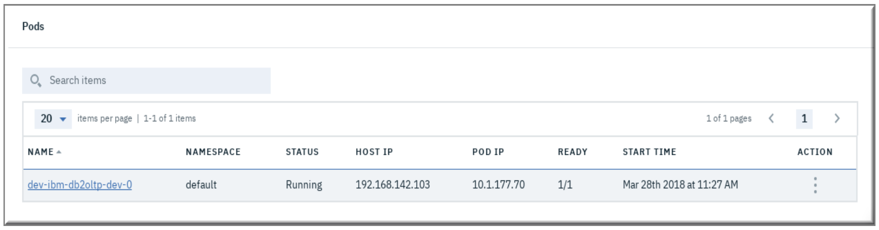
Step	Action



20 items per page | 1-4 of 4 items

NAME	NAMESPACE	DESIRED	CURRENT	CREATION TIME
dev-ibm-db2oltp-dev	default	1	1	Mar 30th 2018 at 4:00 PM
image-manager	kube-system	1	1	Mar 30th 2018 at 1:11 AM
elasticsearch-data	kube-system	1	1	Mar 30th 2018 at 1:10 AM
icp-ds	kube-system	1	1	Mar 30th 2018 at 12:56 AM

__v. In `Pods` section, notice that the Db2 Docker image was deployed. It runs on one of the worker nodes.



Pods

Search items

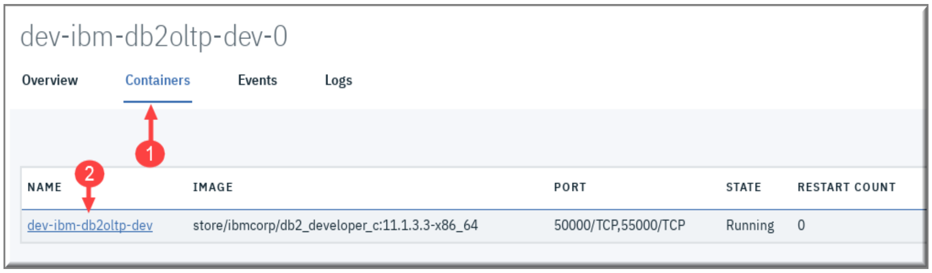
20 items per page | 1-1 of 1 items — 1 of 1 pages — 1

NAME	NAMESPACE	STATUS	HOST IP	POD IP	READY	START TIME	ACTION
dev-ibm-db2oltp-dev-0	default	Running	192.168.142.103	10.1.177.70	1/1	Mar 28th 2018 at 11:27 AM	⋮

__w. Click `dev-ibm-db2oltp-dev-0`

__x. The `Overview` screen displays.

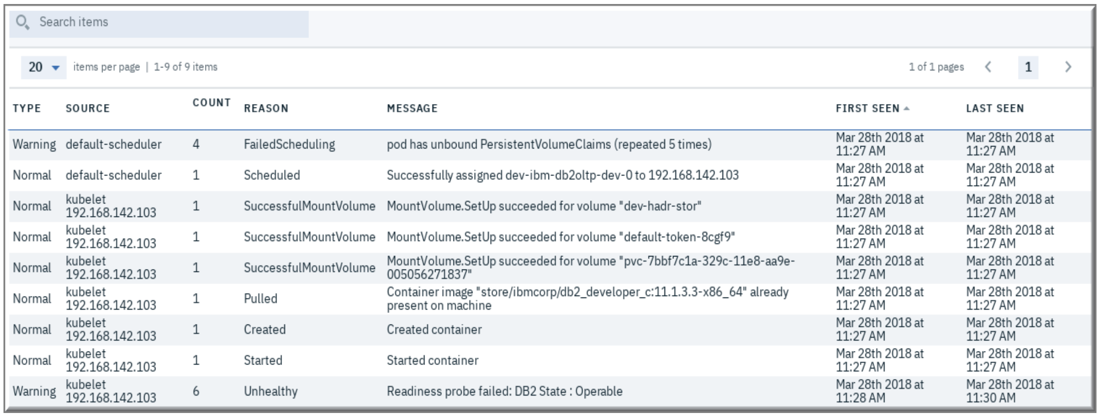
__y. Click `Containers`. Click `dev-ibm-db2oltp-dev`.



dev-ibm-db2oltp-dev-0

NAME	IMAGE	PORT	STATE	RESTART COUNT
dev-ibm-db2oltp-dev	store/ibmcorp/db2_developer_c:11.1.3.3-x86_64	50000/TCP,55000/TCP	Running	0

__z. Click Events.



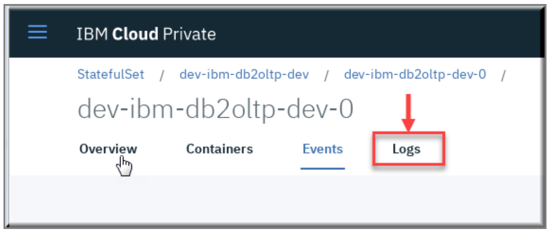
Search items

20 items per page | 1-9 of 9 items 1 of 1 pages 1

TYPE	SOURCE	COUNT	REASON	MESSAGE	FIRST SEEN	LAST SEEN
Warning	default-scheduler	4	FailedScheduling	pod has unbound PersistentVolumeClaims (repeated 5 times)	Mar 28th 2018 at 11:27 AM	Mar 28th 2018 at 11:27 AM
Normal	default-scheduler	1	Scheduled	Successfully assigned dev-ibm-db2oltp-dev-0 to 192.168.142.103	Mar 28th 2018 at 11:27 AM	Mar 28th 2018 at 11:27 AM
Normal	kubelet 192.168.142.103	1	SuccessfulMountVolume	MountVolume.SetUp succeeded for volume "dev-hadr-stor"	Mar 28th 2018 at 11:27 AM	Mar 28th 2018 at 11:27 AM
Normal	kubelet 192.168.142.103	1	SuccessfulMountVolume	MountVolume.SetUp succeeded for volume "default-token-8cgf9"	Mar 28th 2018 at 11:27 AM	Mar 28th 2018 at 11:27 AM
Normal	kubelet 192.168.142.103	1	SuccessfulMountVolume	MountVolume.SetUp succeeded for volume "pvc-7bbf7c1a-329c-11e8-aa9e-005056271837"	Mar 28th 2018 at 11:27 AM	Mar 28th 2018 at 11:27 AM
Normal	kubelet 192.168.142.103	1	Pulled	Container image "store/ibmcorp/db2_developer_c:11.1.3.3-x86_64" already present on machine	Mar 28th 2018 at 11:27 AM	Mar 28th 2018 at 11:27 AM
Normal	kubelet 192.168.142.103	1	Created	Created container	Mar 28th 2018 at 11:27 AM	Mar 28th 2018 at 11:27 AM
Normal	kubelet 192.168.142.103	1	Started	Started container	Mar 28th 2018 at 11:27 AM	Mar 28th 2018 at 11:27 AM
Warning	kubelet 192.168.142.103	6	Unhealthy	Readiness probe failed: DB2 State : Operable	Mar 28th 2018 at 11:28 AM	Mar 28th 2018 at 11:30 AM

__aa. Note the events. If there is any error, delete the Helm chart through Hamburger ☰ ⇨ Workload ⇨ Helm Releases ⇨ dev and click the 3 vertical dots sign to delete and start all over. The error might be due to missing parameters before installing the chart.

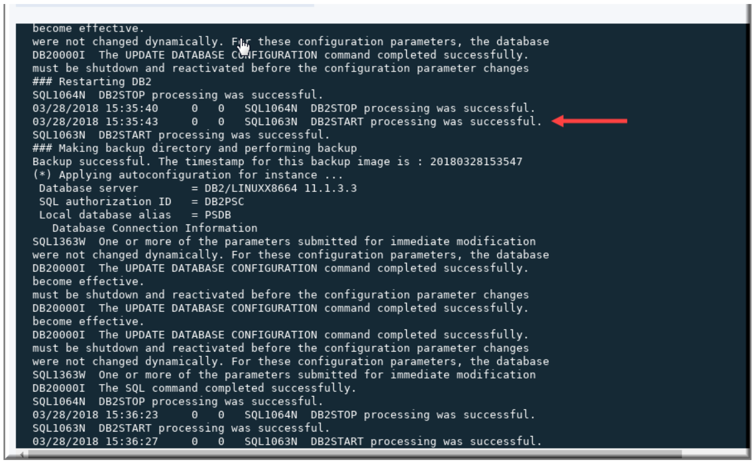
__bb. If there is warning, ignore it. Let's examine the logs. Click Logs.



__cc. These logs are directly coming from the Db2 Docker container running in Kubernetes pod. We will see, in later sections, how to open a Db2 command line shell inside the container and check the logs manually.

__dd. Scroll to the message DB2START processing was successful. The database creation happens automatically. Note this may take several minutes.

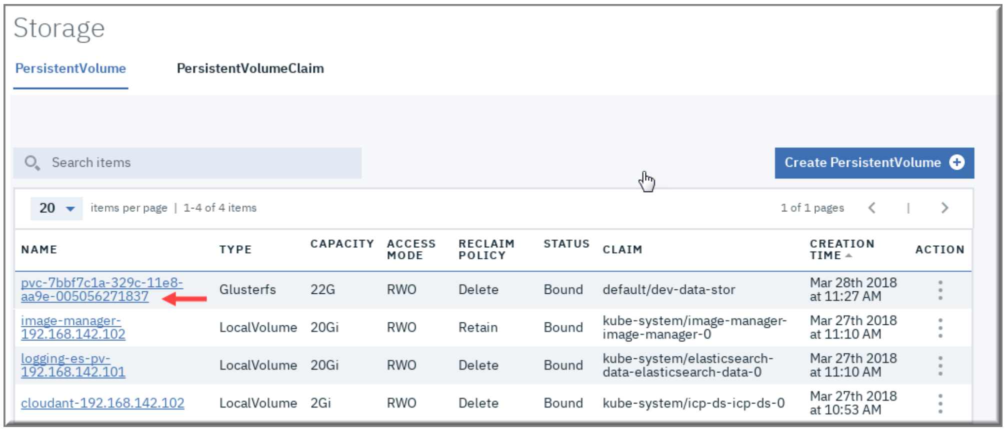
Step	Action



```
become effective.
were not changed dynamically. For these configuration parameters, the database
DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
must be shutdown and reactivated before the configuration parameter changes
### Restarting DB2
SQL1064N  DB2STOP processing was successful.
03/28/2018 15:35:40     0   0   SQL1064N  DB2STOP processing was successful.
03/28/2018 15:35:43     0   0   SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.
### Making backup directory and performing backup
Backup successful. The timestamp for this backup image is : 20180328153547
(*) Applying autoconfiguration for instance ...
 Database server        = DB2/LINUXX8664 11.1.3.3
 SQL authorization ID   = DB2PSC
 Local database alias   = PSDB
   Database Connection Information
SQL1363W  One or more of the parameters submitted for immediate modification
were not changed dynamically. For these configuration parameters, the database
DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
become effective.
must be shutdown and reactivated before the configuration parameter changes
DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
become effective.
DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
must be shutdown and reactivated before the configuration parameter changes
were not changed dynamically. For these configuration parameters, the database
SQL1363W  One or more of the parameters submitted for immediate modification
DB20000I  The SQL command completed successfully.
SQL1064N  DB2STOP processing was successful.
03/28/2018 15:36:23     0   0   SQL1064N  DB2STOP processing was successful.
SQL1063N  DB2START processing was successful.
03/28/2018 15:36:27     0   0   SQL1063N  DB2START processing was successful.
```

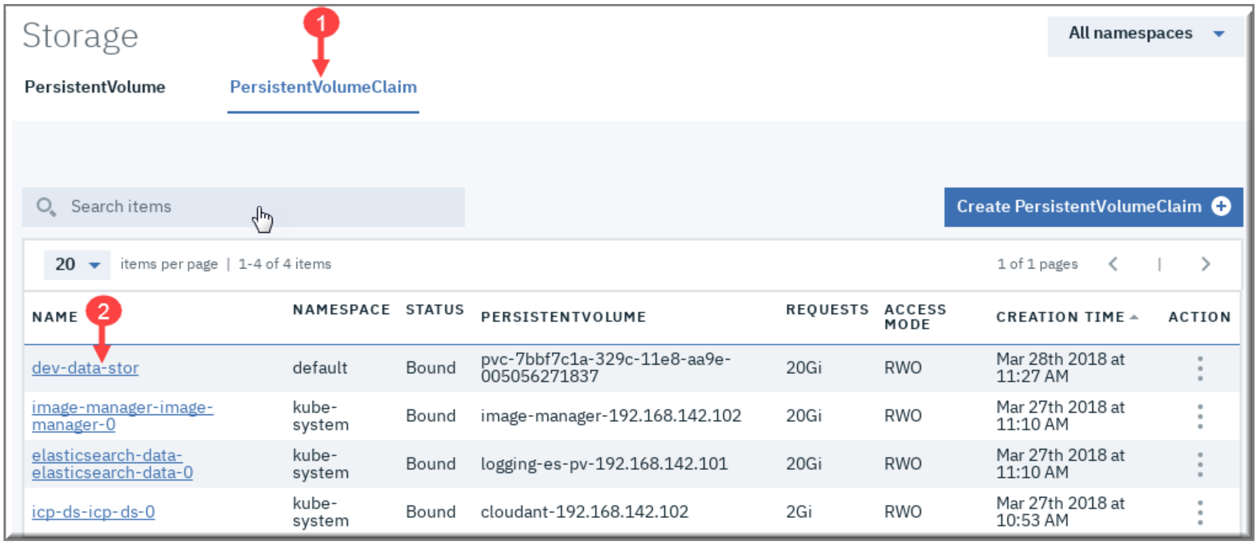
Explore Persistent Volumes

__ee. Click `Hamburger` ☰ ➪ `Platform` ➪ `Storage`.



NAME	TYPE	CAPACITY	ACCESS MODE	RECLAIM POLICY	STATUS	CLAIM	CREATION TIME	ACTION
pvc-7bbf7c1a-329c-11e8-aa9e-005056271837	Glusterfs	22G	RWO	Delete	Bound	default/dev-data-stor	Mar 28th 2018 at 11:27 AM	⋮
image-manager-192.168.142.102	LocalVolume	20Gi	RWO	Retain	Bound	kube-system/image-manager-image-manager-0	Mar 27th 2018 at 11:10 AM	⋮
logging-es-pv-192.168.142.101	LocalVolume	20Gi	RWO	Delete	Bound	kube-system/elasticsearch-data-elasticsearch-data-0	Mar 27th 2018 at 11:10 AM	⋮
cloudant-192.168.142.102	LocalVolume	2Gi	RWO	Delete	Bound	kube-system/icp-ds-icp-ds-0	Mar 27th 2018 at 10:53 AM	⋮

__ff. Click Persistent Volume Claim ➪ dev-data-stor



NAME	NAMESPACE	STATUS	PERSISTENTVOLUME	REQUESTS	ACCESS MODE	CREATION TIME	ACTION
dev-data-stor	default	Bound	pvc-7bbf7c1a-329c-11e8-aa9e-005056271837	20Gi	RWO	Mar 28th 2018 at 11:27 AM	⋮
image-manager-image-manager-0	kube-system	Bound	image-manager-192.168.142.102	20Gi	RWO	Mar 27th 2018 at 11:10 AM	⋮
elasticsearch-data-elasticsearch-data-0	kube-system	Bound	logging-es-pv-192.168.142.101	20Gi	RWO	Mar 27th 2018 at 11:10 AM	⋮
icp-ds-icp-ds-0	kube-system	Bound	cloudant-192.168.142.102	2Gi	RWO	Mar 27th 2018 at 10:53 AM	⋮

__gg. Note that Type GlusterFS persistent volume claim (dev-data-stor) was created automatically by Kubernetes as we had defined the used storage class, which was created during the IBM Cloud Private install.

> **Note:** The mechanics of dynamic provisioning of volumes is requested through REST API from Kubernetes Master node and the request is handled by Heketi REST API server that interfaces with the Gluster docker containers running in all worker nodes.

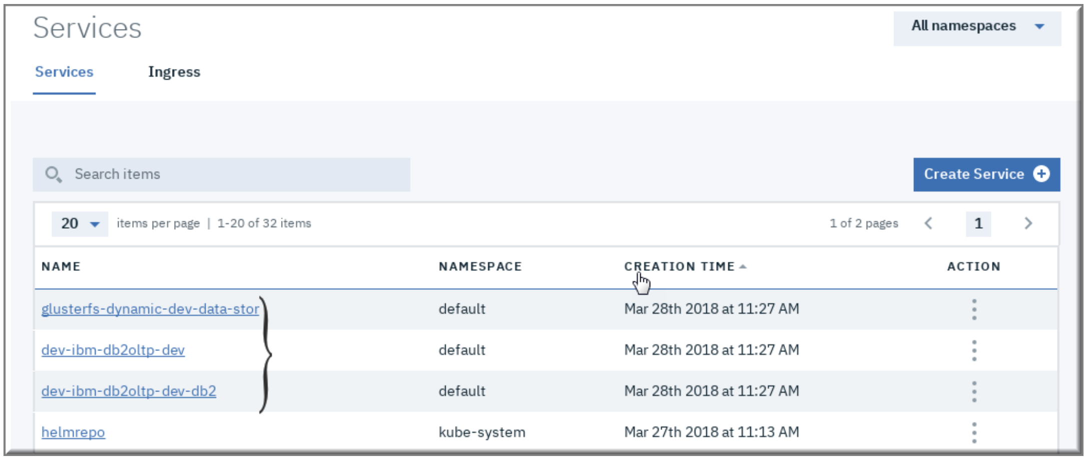
__hh. Click Events and check the message.

__ii. Note: Individual information can be obtained through kubectl commands. You can find the appropriate command from the GUI screen. We explore these in later sections of this lab.

Explore Network Service

__jj. Click Hamburger ☰ ➪Network Access ➪ Services

Step	Action



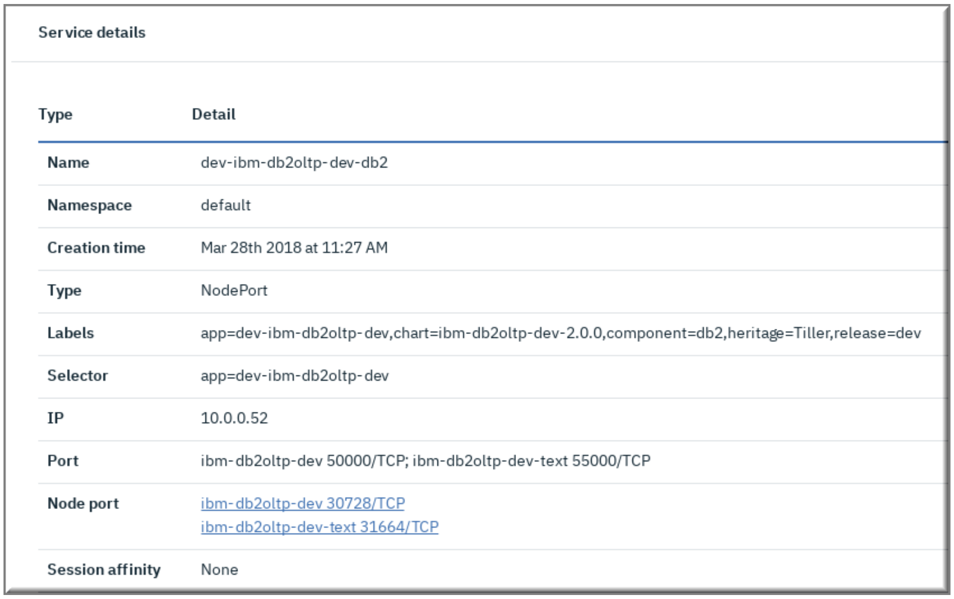
Services

NAME	NAMESPACE	CREATION TIME	ACTION
glusterfs-dynamic-dev-data-stor	default	Mar 28th 2018 at 11:27 AM	⋮
dev-ibm-db2oltp-dev	default	Mar 28th 2018 at 11:27 AM	⋮
dev-ibm-db2oltp-dev-db2	default	Mar 28th 2018 at 11:27 AM	⋮
helmrepo	kube-system	Mar 27th 2018 at 11:13 AM	⋮

__kk. Notice three services created through the Helm chart for proper communication to and from pods running on workers and the external applications.

> **Note:** The network services are provided by Kubernetes based on the type which can be `ClusterIP` or `NodePort`. Note that Db2 can be deployed on any available worker node (affinity can be defined, if desired) and cluster IP addresses are private addresses visible within the cluster only.
>
> The services are interfaces between external and internal communication and are managed internally by the `iptables`. [Refer to the second lab for details.]

__ll. Click `dev-ibm-db2oltp-dev-db2`



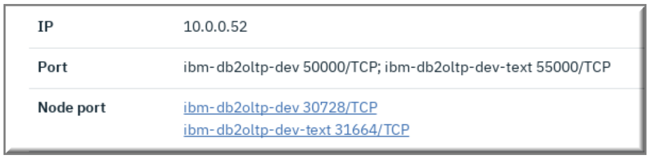
Service details

Type	Detail
Name	dev-ibm-db2oltp-dev-db2
Namespace	default
Creation time	Mar 28th 2018 at 11:27 AM
Type	NodePort
Labels	app=dev-ibm-db2oltp-dev,chart=ibm-db2oltp-dev-2.0.0,component=db2,heritage=Tiller,release=dev
Selector	app=dev-ibm-db2oltp-dev
IP	10.0.0.52
Port	ibm-db2oltp-dev 50000/TCP; ibm-db2oltp-dev-text 55000/TCP
Node port	ibm-db2oltp-dev 30728/TCP ibm-db2oltp-dev-text 31664/TCP
Session affinity	None

Step	Action

__mm. Note that this service is of the type NodePort, which allows traffic to be routed from the external network through workers' IP addresses to the Docker containers running within the same pod.

__nn. Review the Port and NodePort definition ibm-db2oltp-dev 30728/TCP.



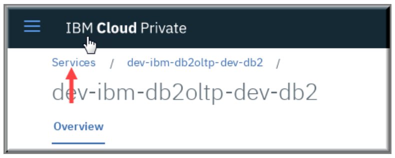
IP	10.0.0.52
Port	ibm-db2oltp-dev 50000/TCP; ibm-db2oltp-dev-text 55000/TCP
Node port	ibm-db2oltp-dev 30728/TCP ibm-db2oltp-dev-text 31664/TCP

__oo. Conclusions from above:

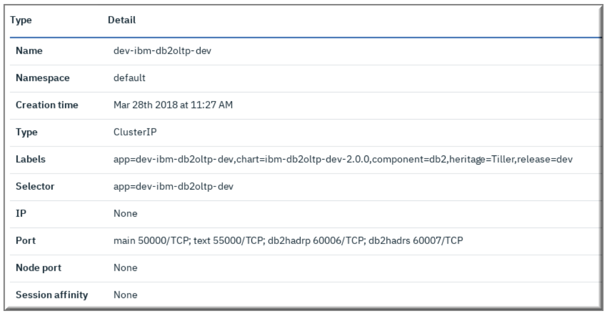
- ✓ Db2 is running in a container with an IP address 10.0.0.52 (This could differ in your case.) which is on host 192.168.142.103 (This may also differ for you. You can determine this by looking at Hamburger ☰ ➪ Workloads ➪ StatefulSets)
- ✓ Two TCP ports 50000 (db2 instance) and 55000 (Text search) are available.
- ✓ Through NodePort, access is available to 10.0.0.52:50000 and 10.0.0.52:55000 via any host IP of workers' node at port 30728 and 31664.

ℹ	**Note:** The Db2 Docker container always uses port 50,000 and 55,000 and as many containers can be deployed. But each will have different high port through which connections from external network can be made. This abstraction provides agility and elasticity and helps in automation.

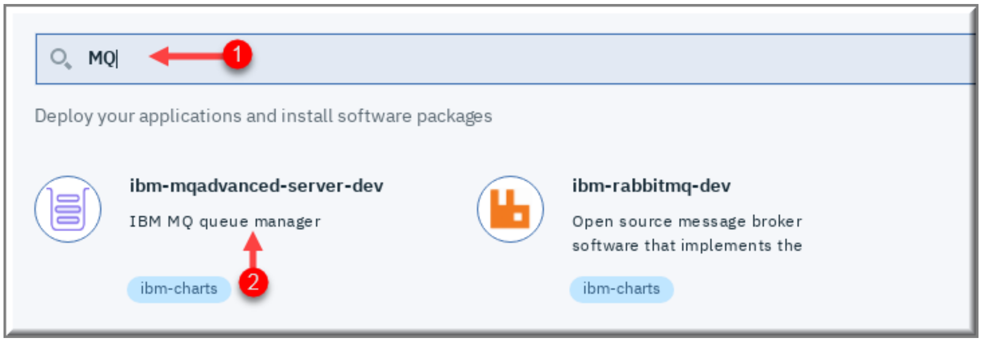
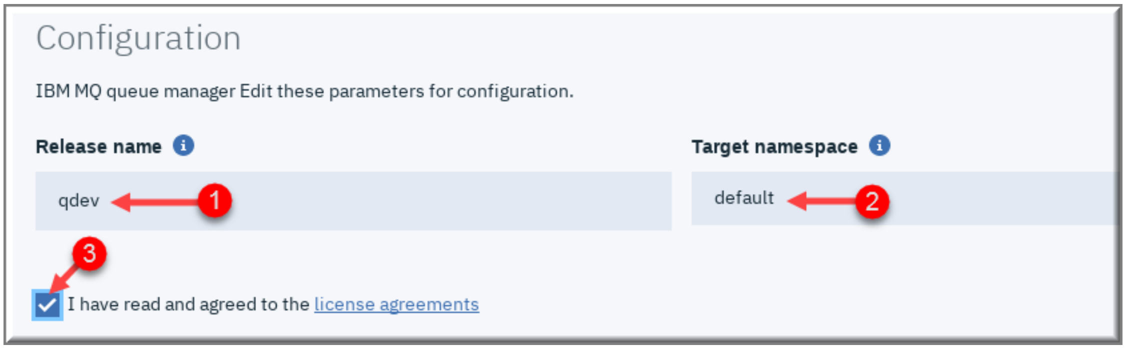
__pp. Click Services.



__qq. Click dev-ibm-db2oltp-dev



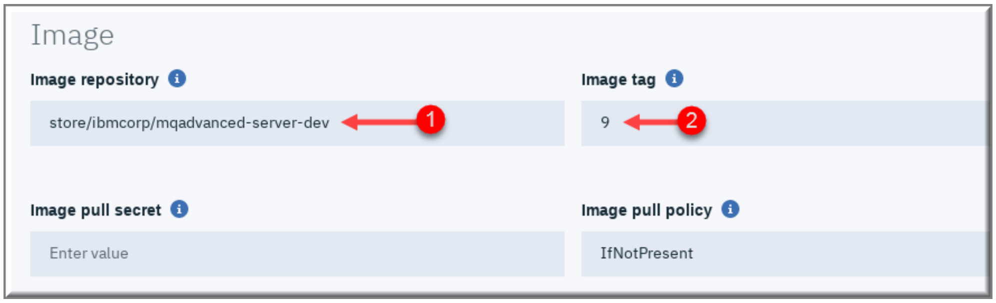
Type	Detail
Name	dev-ibm-db2oltp-dev
Namespace	default
Creation time	Mar 28th 2018 at 11:27 AM
Type	ClusterIP
Labels	app=dev-ibm-db2oltp-dev,chart=ibm-db2oltp-dev-2.0.0,component=db2,heritage=Tiller,release=dev
Selector	app=dev-ibm-db2oltp-dev
IP	None
Port	main 50000/TCP; text 55000/TCP; db2hadrp 60006/TCP; db2hadrs 60007/TCP
Node port	None
Session affinity	None

Step	Action
	__rr. This network service is assigned the type ClusterIP which restricts access to within the cluster and keeps it isolated. **Note:** We have seen all the essential ingredients for an automated install and deployments of Db2 in a private cloud environment run by IBM Cloud Private. This is a new paradigm that DBAs should adopt to stay current in a rapidly changing technological environment.
2	**<u>Deploy IBM MQ</u>** __a. Click Hamburger ☰ ⇨ Catalog ⇨ Helm Charts __b. Type MQ in the search box. Click IBM MQ queue manager.  MQ Deploy your applications and install software packages ibm-mqadvanced-server-dev IBM MQ queue manager ibm-charts ibm-rabbitmq-dev Open source message broker software that implements the ibm-charts __c. Review the IBM MQ Helm Chart readme file. __d. Click Configure. __e. Please pay attention and enter the information correctly to avoid online download of the image. __f. Type *Release name* qdev, select *Target namespace* default and check I have read and agreed to the license agreement.  Configuration IBM MQ queue manager Edit these parameters for configuration. Release name: qdev Target namespace: default ☑ I have read and agreed to the license agreements

Step	Action

__g. Scroll to the Image section.

__h. For *Image Repository*, type store/ibmcorp/mqadvanced-server-dev and Tag 9. [Type carefully. No typos, please.]



Image

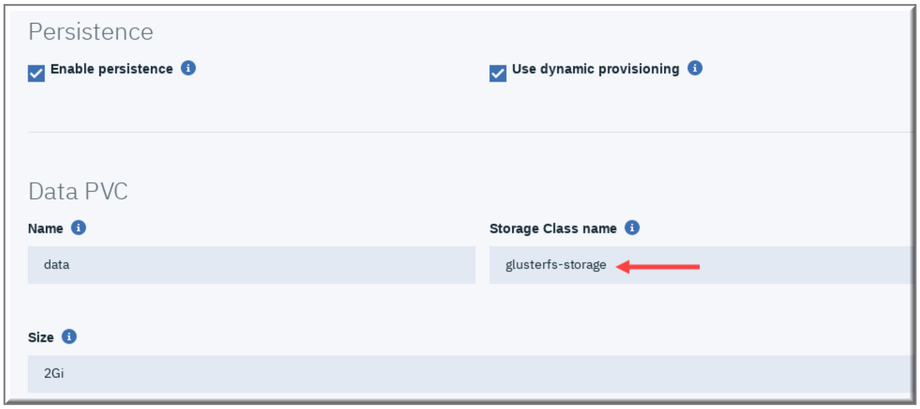
Image repository: store/ibmcorp/mqadvanced-server-dev

Image tag: 9

Image pull secret: Enter value

Image pull policy: IfNotPresent

__i. Scroll to Persistence, Enter glusterfs-storage in *Storage Class name*.



Persistence

☑ Enable persistence

☑ Use dynamic provisioning

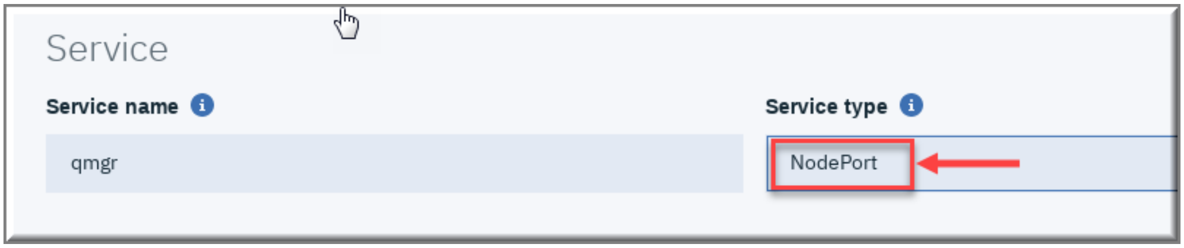
Data PVC

Name: data

Storage Class name: glusterfs-storage

Size: 2Gi

__j. In the Service section, change *Service type* to NodePort.



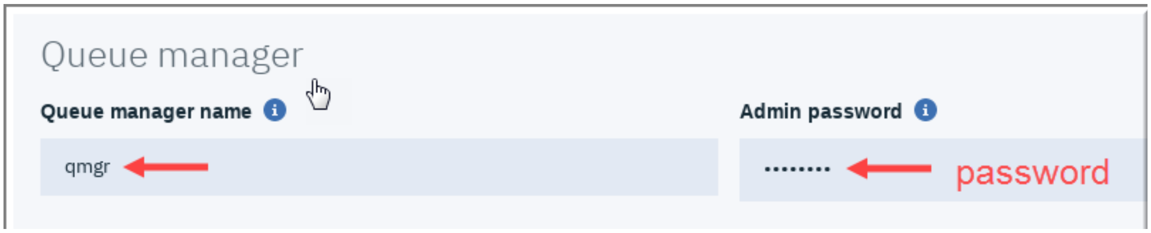
Service

Service name: qmgr

Service type: NodePort

__k. In *Queue manager*, type *Queue manager name* qmgr, *Admin password* as password and do not specify any password for *App*.

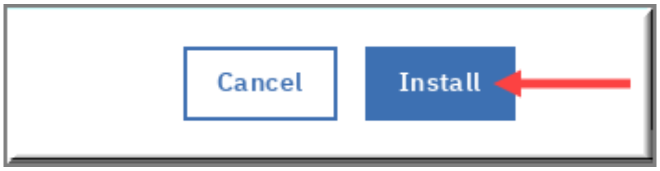
Step	Action


Queue manager

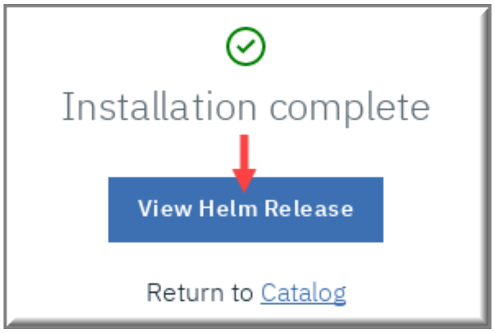
Queue manager name: qmgr

Admin password: ••••••••

__l. Click Install.

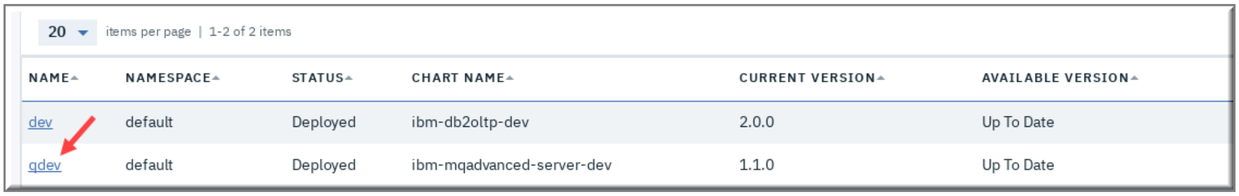


__m. Click View Helm Release.


Installation complete

Return to Catalog

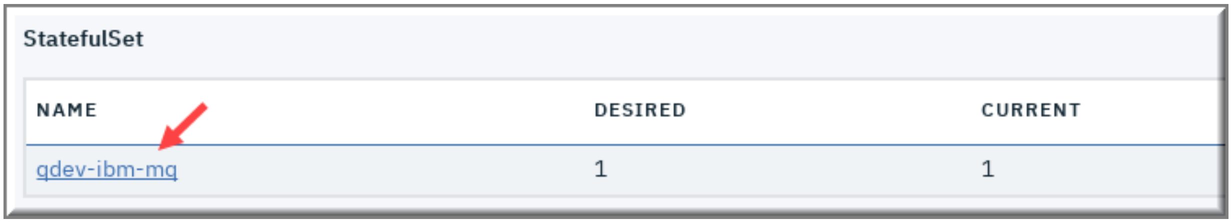
__n. Click qdev


20 items per page | 1-2 of 2 items

NAME	NAMESPACE	STATUS	CHART NAME	CURRENT VERSION	AVAILABLE VERSION
dev	default	Deployed	ibm-db2oltp-dev	2.0.0	Up To Date
qdev	default	Deployed	ibm-mqadvanced-server-dev	1.1.0	Up To Date

__o. Note fourartifacts Secret, Service, StatefulSet and Pod were created.

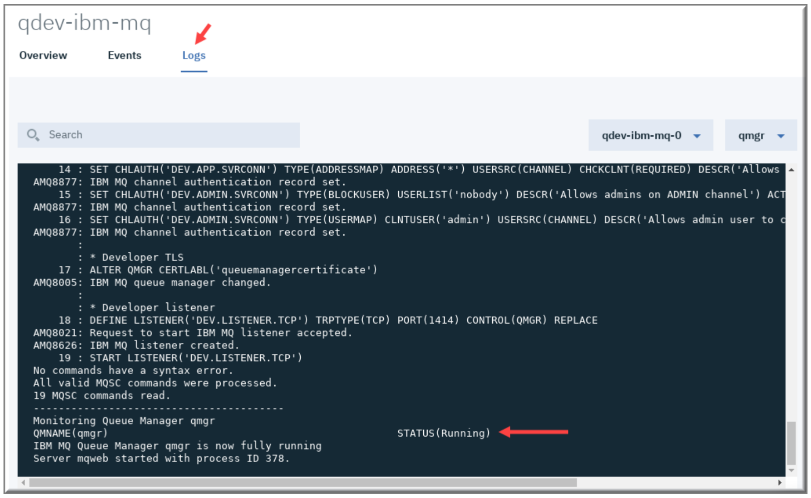
__p. Click *StatefulSet* qdev-ibm-mq.



StatefulSet

NAME	DESIRED	CURRENT
qdev-ibm-mq	1	1

__q. Click Logs. The MQ operational status should display as running.

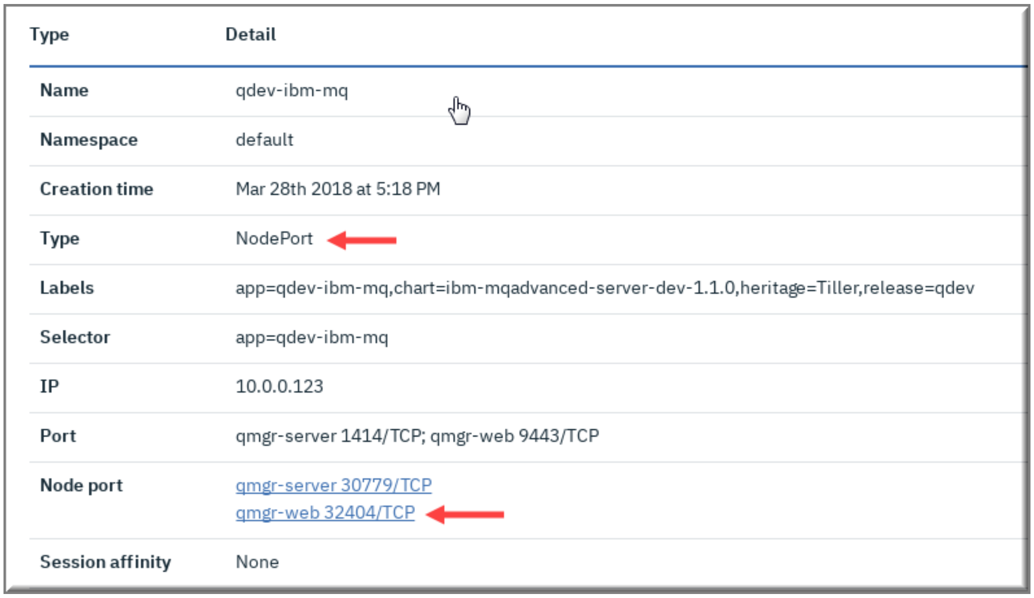


```
     14 : SET CHLAUTH('DEV.APP.SVRCONN') TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(CHANNEL) CHCKCLNT(REQUIRED) DESCR('Allows
AMQ8877: IBM MQ channel authentication record set.
     15 : SET CHLAUTH('DEV.ADMIN.SVRCONN') TYPE(BLOCKUSER) USERLIST('nobody') DESCR('Allows admins on ADMIN channel') ACT
AMQ8877: IBM MQ channel authentication record set.
     16 : SET CHLAUTH('DEV.ADMIN.SVRCONN') TYPE(USERMAP) CLNTUSER('admin') USERSRC(CHANNEL) DESCR('Allows admin user to c
AMQ8877: IBM MQ channel authentication record set.
        :
        : * Developer TLS
     17 : ALTER QMGR CERTLABL('queuemanagercertificate')
AMQ8005: IBM MQ queue manager changed.
        :
        : * Developer listener
     18 : DEFINE LISTENER('DEV.LISTENER.TCP') TRPTYPE(TCP) PORT(1414) CONTROL(QMGR) REPLACE
AMQ8021: Request to start IBM MQ listener accepted.
AMQ8626: IBM MQ listener created.
     19 : START LISTENER('DEV.LISTENER.TCP')
No commands have a syntax error.
All valid MQSC commands were processed.
19 MQSC commands read.
----------------------------------------
Monitoring Queue Manager qmgr
QMNAME(qmgr)                                          STATUS(Running)
IBM MQ Queue Manager qmgr is now fully running
Server mqweb started with process ID 378.
```

__r. Click Hamburger ☰ ⇨ Network Access ⇨ Services

__s. Locate and click qdev-ibm-mq

Step	Action



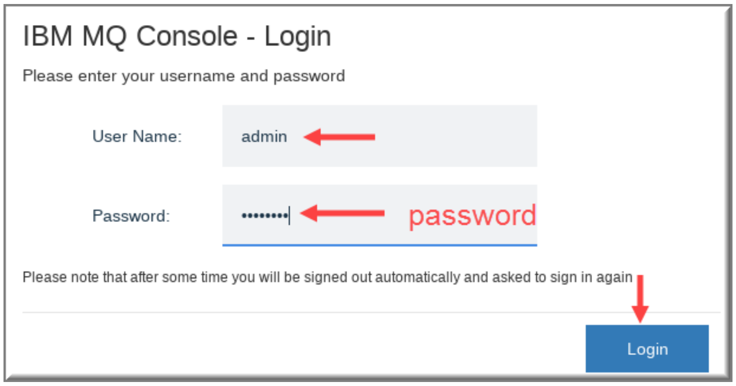
Type	Detail
Name	qdev-ibm-mq
Namespace	default
Creation time	Mar 28th 2018 at 5:18 PM
Type	NodePort
Labels	app=qdev-ibm-mq,chart=ibm-mqadvanced-server-dev-1.1.0,heritage=Tiller,release=qdev
Selector	app=qdev-ibm-mq
IP	10.0.0.123
Port	qmgr-server 1414/TCP; qmgr-web 9443/TCP
Node port	qmgr-server 30779/TCP qmgr-web 32404/TCP
Session affinity	None

__t. Note the *Service type* `NodePort` and note the high port number for `qmgr-web.` In our case, this is `32404` (Your case might be different.)

__u. Open the second tab in the browser.

__v. Type the address `https://192.168.142.101:<your port>` (Replace the port number with the one you see in your case.)

__w. You are directed to `IBM MQ Console`. Type User Name: `admin` and Password: `password`.



IBM MQ Console - Login

Please enter your username and password

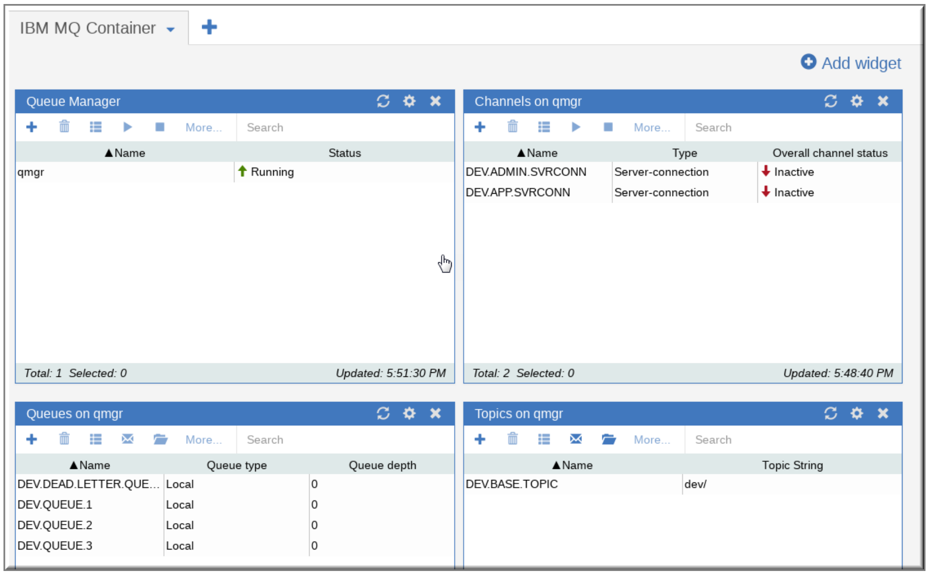
User Name: admin

Password: ••••••••

Please note that after some time you will be signed out automatically and asked to sign in again

Note: Be patient. We have only one CPU and one SSD drive. We are running five VMs and more than 120+ docker containers.

__x. The IBM MQ console displays. Allow time for it to load all panes.



Queue Manager

▲Name	Status
qmgr	Running

Total: 1 Selected: 0 Updated: 5:51:30 PM

Channels on qmgr

▲Name	Type	Overall channel status
DEV.ADMIN.SVRCONN	Server-connection	Inactive
DEV.APP.SVRCONN	Server-connection	Inactive

Total: 2 Selected: 0 Updated: 5:48:40 PM

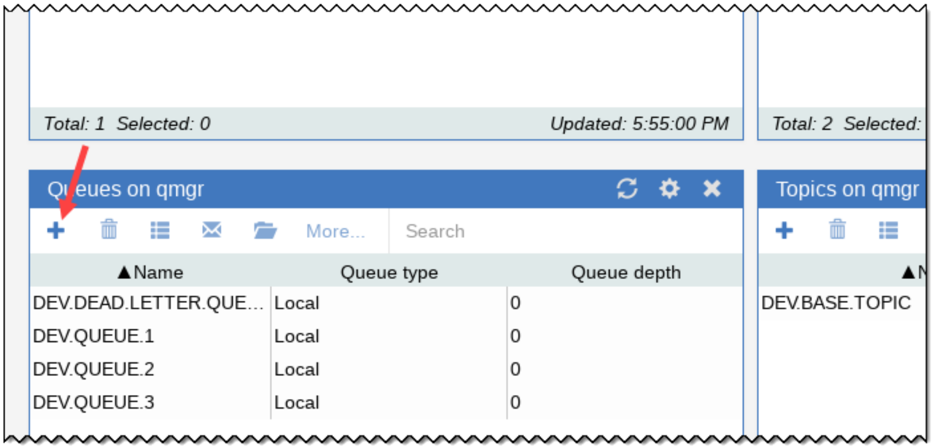
Queues on qmgr

▲Name	Queue type	Queue depth
DEV.DEAD.LETTER.QUE...	Local	0
DEV.QUEUE.1	Local	0
DEV.QUEUE.2	Local	0
DEV.QUEUE.3	Local	0

Topics on qmgr

▲Name	Topic String
DEV.BASE.TOPIC	dev/

__y. Click + in the bottom left `Queues on qmgr` to add a Queue.



Total: 1 Selected: 0 Updated: 5:55:00 PM

Total: 2 Selected:

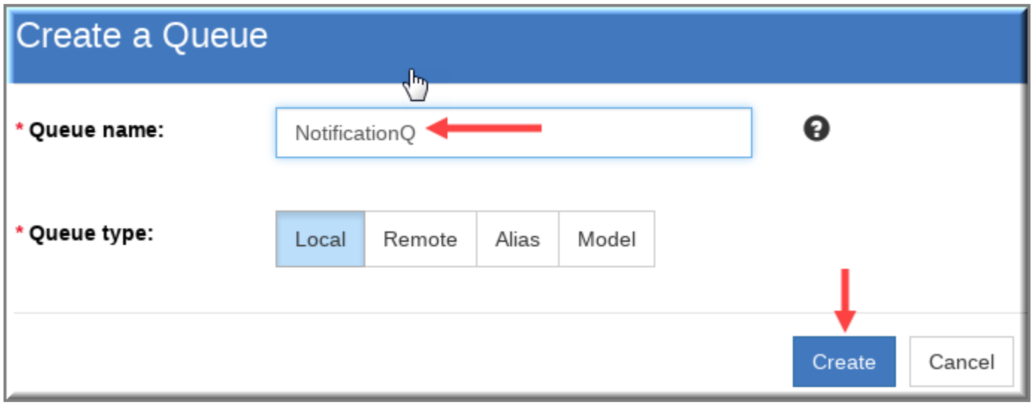
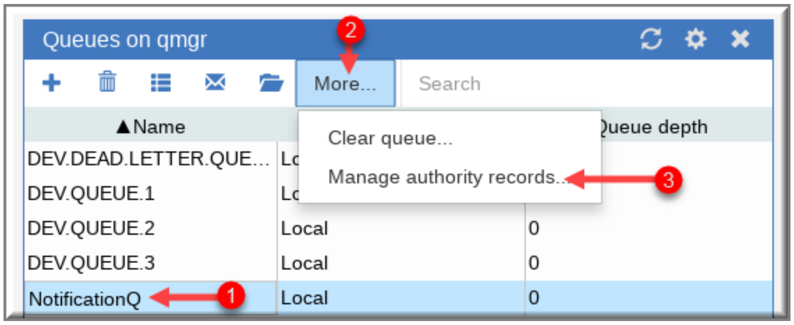
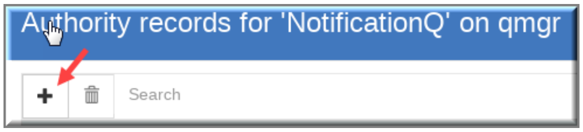
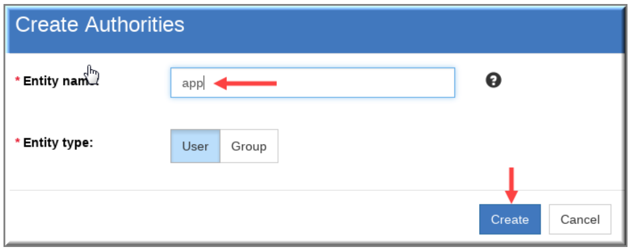
Queues on qmgr

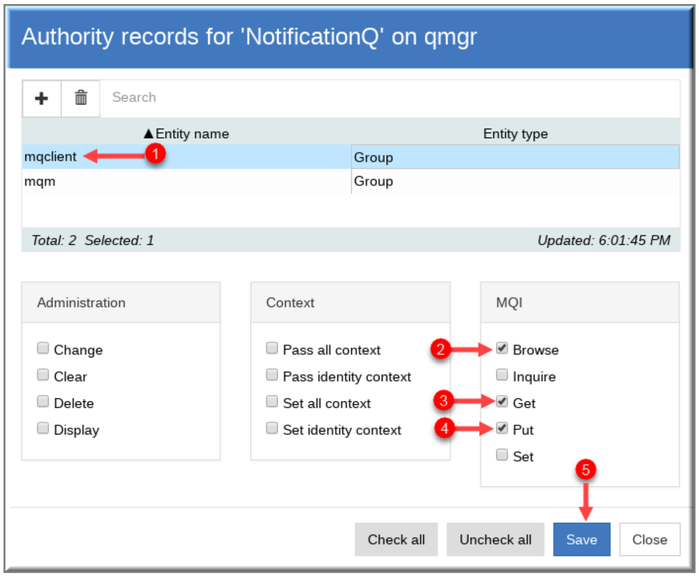
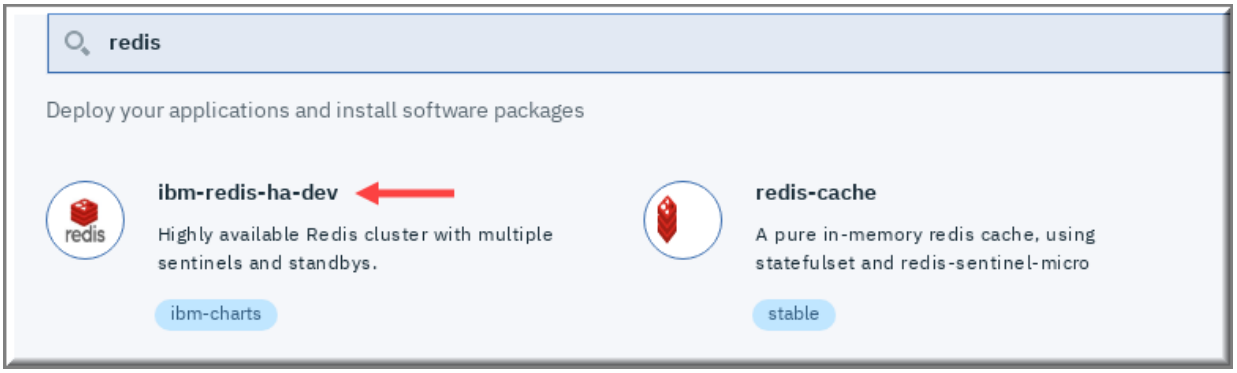
▲Name	Queue type	Queue depth
DEV.DEAD.LETTER.QUE...	Local	0
DEV.QUEUE.1	Local	0
DEV.QUEUE.2	Local	0
DEV.QUEUE.3	Local	0

Topics on qmgr

▲N
DEV.BASE.TOPIC

__z. Type `NotificationQ` (Type carefully. No typos please.) and click `Create`.

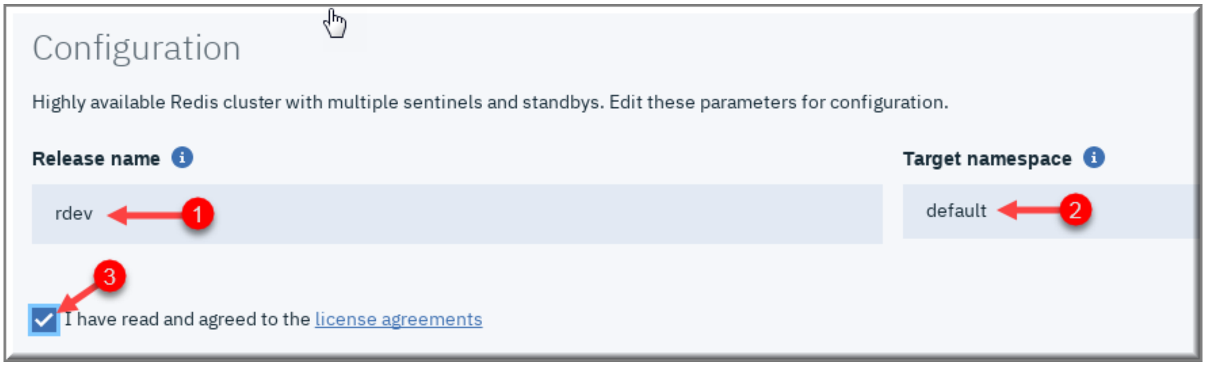
Step	Action
	 Create a Queue * Queue name: NotificationQ * Queue type: Local
	__aa. Select NotificationQ, click More... and select Manage authority records.
	 Queues on qmgr More... Search Clear queue... Manage authority records... 1 2 3 ▲Name | Queue depth DEV.DEAD.LETTER.QUE... | Lo DEV.QUEUE.1 | Lo DEV.QUEUE.2 | Local | 0 DEV.QUEUE.3 | Local | 0 NotificationQ | Local | 0
	__bb. Click +
	 Authority records for 'NotificationQ' on qmgr
	__cc. Type Entity name: app and click Create.
	 Create Authorities * Entity name: app * Entity type: User

Step	Action
	__dd. Click mqclient. Check Browse, Get and Put and click Save.  Authority records for 'NotificationQ' on qmgr Search ▲Entity name — Entity type mqclient — Group mqm — Group Total: 2 Selected: 1 — Updated: 6:01:45 PM Administration: □ Change □ Clear □ Delete □ Display Context: □ Pass all context □ Pass identity context □ Set all context □ Set identity context MQI: ☑ Browse □ Inquire ☑ Get ☑ Put □ Set 1 2 3 4 5 Check all Uncheck all Save Close __ee. Click Close. Click x to close the browser tab for IBM MQ dashboard.
4	**<u>Deploy Redis- In Memory Cache</u>** __ff. Click Hamburger ☰ ⇨ Catalog ⇨ Helm Charts __gg. Type Redis in the search box. Click ibm-redis-ha-dev for my-local-charts.  redis Deploy your applications and install software packages ibm-redis-ha-dev — Highly available Redis cluster with multiple sentinels and standbys. — ibm-charts redis-cache — A pure in-memory redis cache, using statefulset and redis-sentinel-micro — stable __hh. Review the ibm-redis-ha-dev readme file.

Step	Action

__ii. Click Configure.

__jj. Type *Release name* rdev, select *Target namespace* default and check I have read and agreed to the license agreement.



Configuration

Highly available Redis cluster with multiple sentinels and standbys. Edit these parameters for configuration.

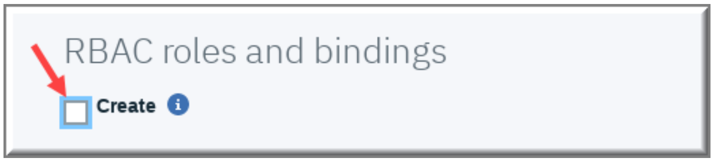
Release name: rdev

Target namespace: default

☑ I have read and agreed to the license agreements

__kk. Scroll to *RBAC roles and bindings*.

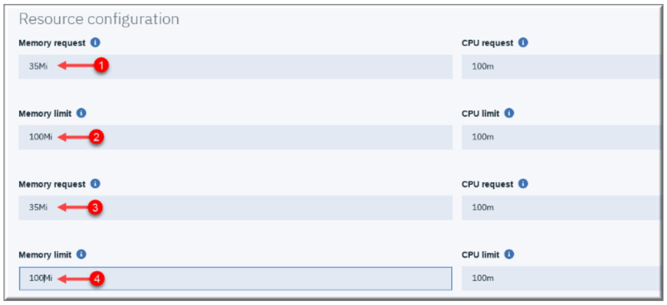
__ll. Uncheck Create.



RBAC roles and bindings

☐ Create

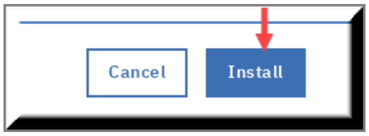
__mm. Scroll to *Resource configuration* and change Memory request to 35Mi and Memory limit to 100Mi.



Resource configuration

Memory request: 35Mi	CPU request: 100m
Memory limit: 100Mi	CPU limit: 100m
Memory request: 35Mi	CPU request: 100m
Memory limit: 100Mi	CPU limit: 100m

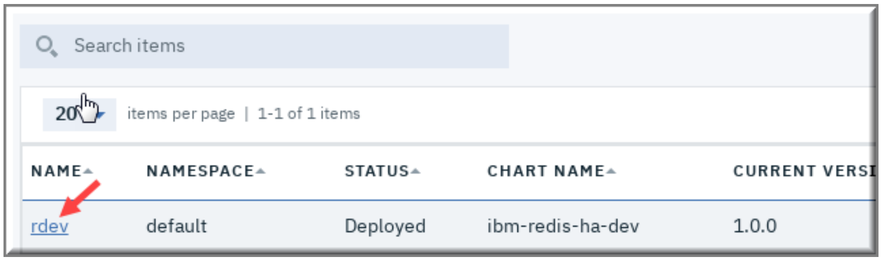
__nn. Click Install.



Step	Action

__oo. Click View Helm Release.

__pp. Click rdev.



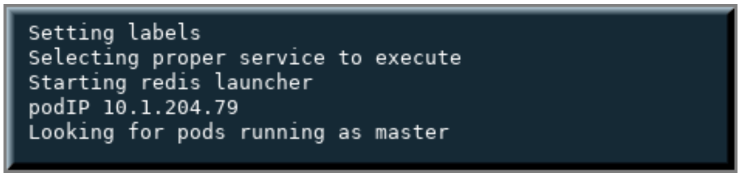
20 items per page | 1-1 of 1 items

NAME	NAMESPACE	STATUS	CHART NAME	CURRENT VERSI
rdev	default	Deployed	ibm-redis-ha-dev	1.0.0

__qq. Please notice the artifacts *Service Account*, *Service*, *Deployment* and *Pods* created for Redis in memory cache database.

__rr. Scroll to Deployment and click rdev-ibm-redis-ha-dev-server

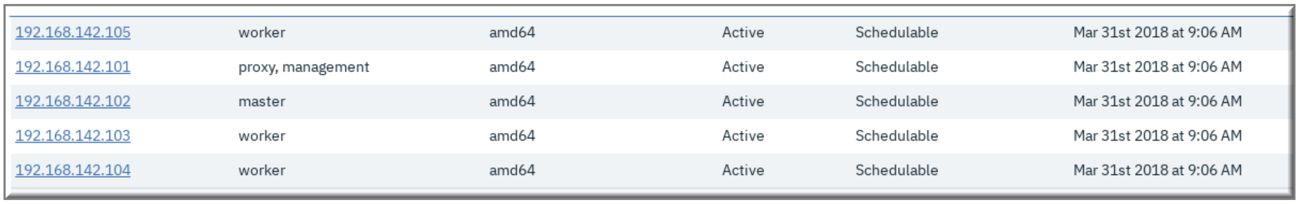
__ss. Click Logs.



```
Setting labels
Selecting proper service to execute
Starting redis launcher
podIP 10.1.204.79
Looking for pods running as master
```

Note: We have deployed Db2, MQ and Redis using IBM Cloud Private Catalog. We connect these deployments together in later labs to demonstrate a microservices application that uses IBM MQ, In-memory cache Redis and backend as Db2.

__tt. Navigate to Hamburger ⇨ Platform ⇨ Nodes. Click each worker node to find the host in which Db2, MQ and Redis are running.



192.168.142.105	worker	amd64	Active	Schedulable	Mar 31st 2018 at 9:06 AM
192.168.142.101	proxy, management	amd64	Active	Schedulable	Mar 31st 2018 at 9:06 AM
192.168.142.102	master	amd64	Active	Schedulable	Mar 31st 2018 at 9:06 AM
192.168.142.103	worker	amd64	Active	Schedulable	Mar 31st 2018 at 9:06 AM
192.168.142.104	worker	amd64	Active	Schedulable	Mar 31st 2018 at 9:06 AM

__uu. Notice that Kubernetes scheduler has spread them across all three worker nodes.

__vv. If any of the pods/workers is lost, Kubernetes starts the pod on other available worker nodes.

Step	Action
	__ww. We have only one master node running, which is a single point of failure. The Enterprise edition allows the creation of multiple master nodes for high availability. Due to resource constraints, we created only one master node.

Troubleshooting

__a. If your chart (db2, MQ or Liberty) did not install properly or it appears to be in perpetual pending mode, take the following steps.

- ✓ Be sure you specified the image repository name correctly, as given in the lab.
- ✓ Be sure you specified the tag properly, as given in the lab.
- ✓ Be sure you specified the Storage Class Name correctly – `glusterfs-storage`
- ✓ Not specifying Storage Class properly is the most common error.
- ✓ If the pod is still in pending state, delete the deployment.
  - __i. Go to **Hamburger** ⇨ **Workload** ⇨ **Helm release**.
  - __ii. Select the **release**, click the **three vertical dots** and click **delete**.
  - __iii. Go to **Hamburger** ⇨ **Platform** ⇨ **Storage** ⇨ **Persistent Volume Claim** and delete it.
- ✓ Repeat the exercise.

__b. If you forgot to specify MQ password, you can obtain the password from the command line.

__c. Type command:

```
# kubectl get secret qdev-ibm-mq -o json | jq -r .data.adminPassword | base64 -d
```

Section 6: Lab Summary

In this section, you learned how to deploy services from Helm charts using the ICP Catalog feature.

Section 7: 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

- In previous lab exercises, we saw the build process for Db2, MQ and Redis.

- 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.

- 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

Step	Action
1	**Building the Docker Containers for Each Microservice**

__a. Switch to the GNOME Terminal command line.

__b. Type cd8 to switch the lab directory to 08ms (microservices).

```
[root@node01 07ta]# cd8
[root@node01 08ms]#
```

__c. 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
```

__d. The above script runs the 01-builddocker script from each of the subdirectories. This script creates the Docker container for each microservice.

__e. Run ls -l

```
[root@node01 08ms]# ls -l
total 32
-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
```

Step	Action

```
-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
```

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

__g. Run `cd 03-portfolio`

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

__h. 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
```

Build Containers

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

Step	Action

```
[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 .
```

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

__k. 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
```

__l. 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.

__m. 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.

Step	Action

__n. 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.

__o. Run `tree config`.

```
[root@node01 03-portfolio]# tree config/
config/
├── apps
│   └── Portfolio.war
├── configDropins
│   └── defaults
│       └── keystore.xml
├── resources
│   └── security
│       └── key.jks
└── server.xml
```

__p. 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.

__q. 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.

__r. Note that we have `Portfolio.war` in `apps` directory. The security files `key.jks` and `keystore.xml` in their respective directories.

__s. 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>
-  - - -

    <connectionManager id="DB2-Connections"
         minPoolSize="5" maxPoolSize="50"/>
```

Step	Action

```
    <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>
```

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

__u. 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.

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

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

Step	Action

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

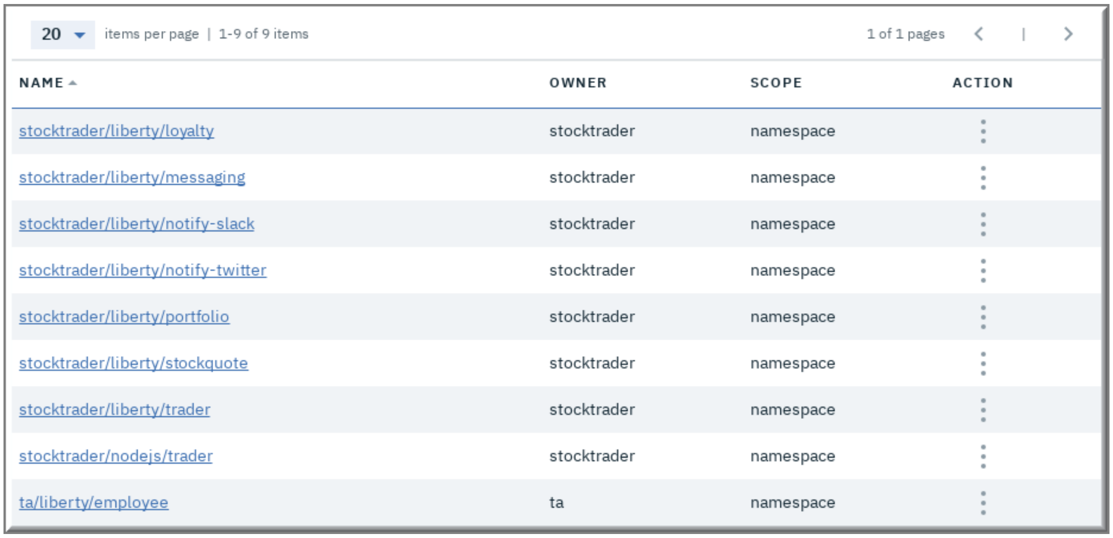
__g. Run `01-push-image-to-local-registry`

```
[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`.



20 ▾ items per page | 1-9 of 9 items — 1 of 1 pages

NAME ▴	OWNER	SCOPE	ACTION
stocktrader/liberty/loyalty	stocktrader	namespace	⋮
stocktrader/liberty/messaging	stocktrader	namespace	⋮
stocktrader/liberty/notify-slack	stocktrader	namespace	⋮
stocktrader/liberty/notify-twitter	stocktrader	namespace	⋮
stocktrader/liberty/portfolio	stocktrader	namespace	⋮
stocktrader/liberty/stockquote	stocktrader	namespace	⋮
stocktrader/liberty/trader	stocktrader	namespace	⋮
stocktrader/nodejs/trader	stocktrader	namespace	⋮
ta/liberty/employee	ta	namespace	⋮

__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.

Step	Action
	__l. Switch back to the command line.

3

Create Database Tables

__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
DB20000I  The SQL command completed successfully.
```

Step	Action
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.

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

```
[root@node01 08ms]# kubectl -n stocktrader get secret db2 -o yaml
apiVersion: v1
data:
```

Step	Action

```
  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
        image: poticpcluster.icp:8500/stocktrader/liberty/portfolio:1.0.1
        env:
          - name: JDBC_HOST
            valueFrom:
              secretKeyRef:
                name: db2
```

Step	Action

```
                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
      protocol: TCP
      port: 9443
      targetPort: 9443
  selector:
    app: portfolio
---
```

Step	Action

```
#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.

✓ 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.`

Step	Action

✓ 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
==========================================================

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

Step	Action

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

secret "ingress-host" created
==========================================================
Deploy Liberty Docker container for loyalty
```

Step	Action

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

Step	Action

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.

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
```

Step	Action

```
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`

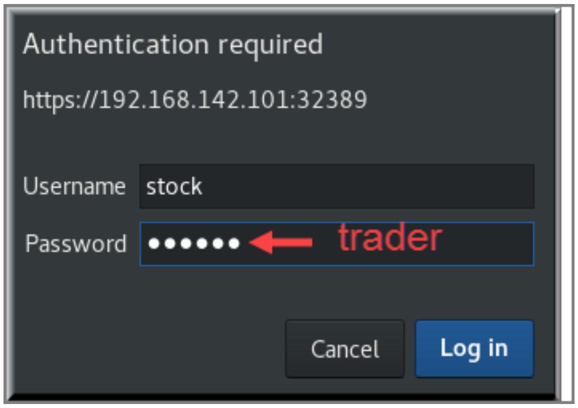
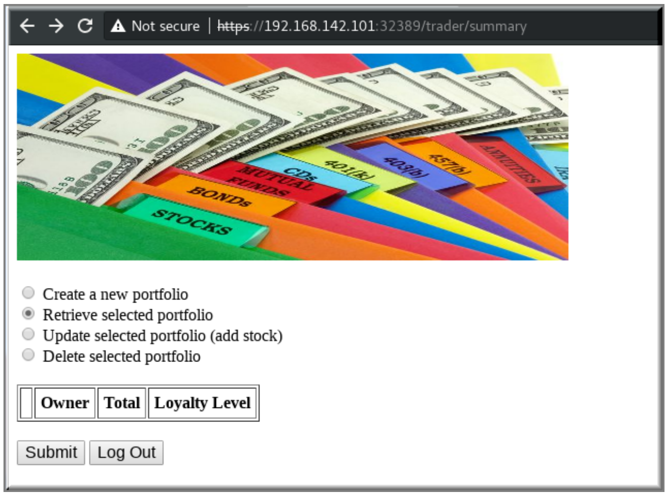
__n. Note that we are using the node port when accessing the application through proxy server.

__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.

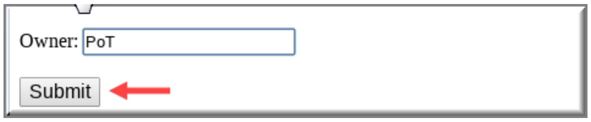
> **Note:** The internal routing is managed by iptables rules defined by the Kubernetes when a network service is defined.

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 collection of routing rules that govern how external users access services running in a Kubernetes cluster.

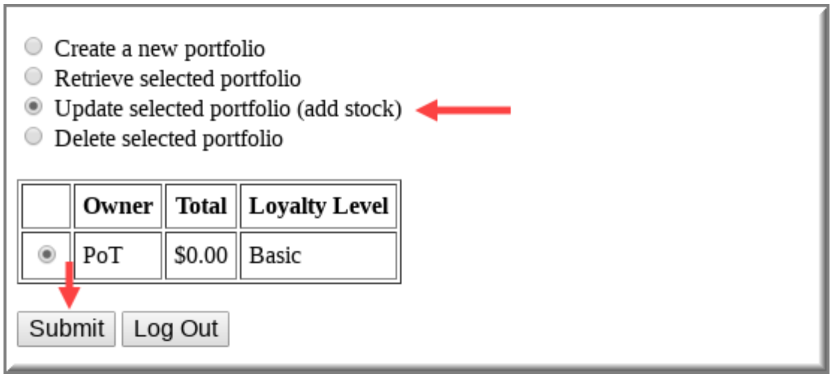
Step	Action
6	**Run the Microservice Application** __a. Open a new browser tab to run the application. __b. Type URL: `https://192.168.142.101:32389/trader/summary` __c. Type Username `stock` and Password `trader`. Click `Log in`.  Authentication required https://192.168.142.101:32389 Username stock Password •••••• trader Cancel 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.  Not secure | ~~https~~://192.168.142.101:32389/trader/summary ○ Create a new portfolio ● Retrieve selected portfolio ○ Update selected portfolio (add stock) ○ Delete selected portfolio Owner Total Loyalty Level Submit Log Out __d. Tick `Create a new portfolio`. Click `Submit`. __e. Type Owner `PoT` and click `submit`.

Step	Action



Owner: PoT

__f. Tick `Update selected portfolio (add stock).` Click `Submit.`

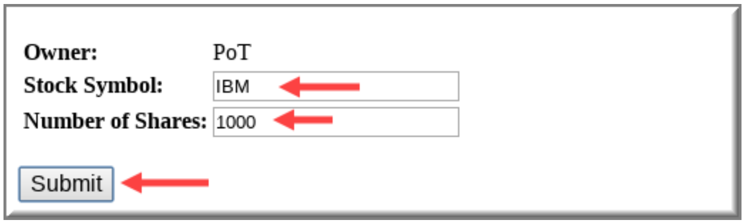


○ Create a new portfolio
○ Retrieve selected portfolio
● Update selected portfolio (add stock)
○ Delete selected portfolio

	Owner	Total	Loyalty Level
●	PoT	$0.00	Basic

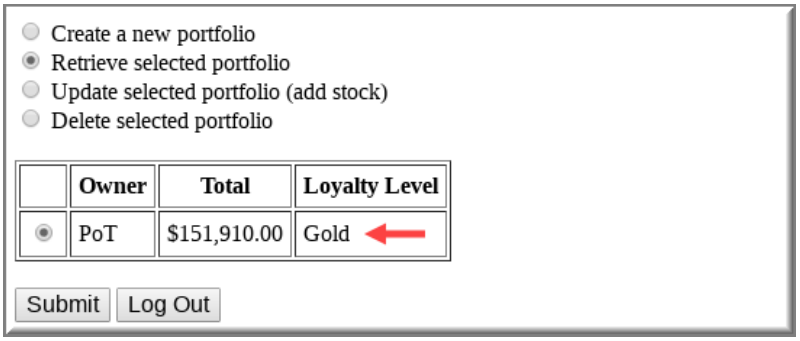
__g. Note: You need an Internet connection as this request routes to `https://www.quandl.com/` to retrieve the stock quote at the end of the previous day.

__h. Type Stock Symbol `IBM` and specify `1000` stocks. Click `Submit`.



Owner: PoT
Stock Symbol: IBM
Number of Shares: 1000

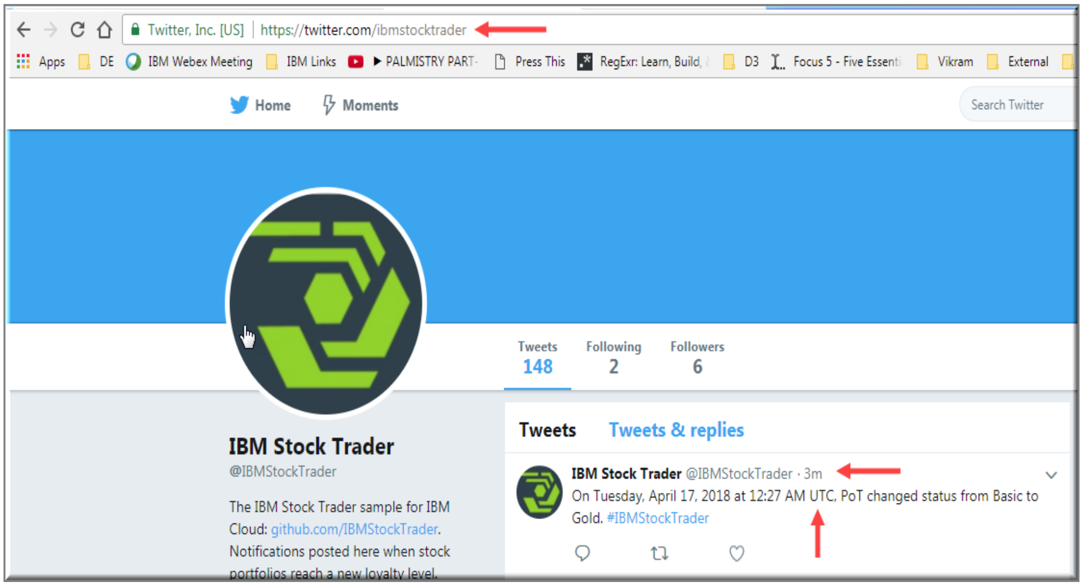
__i. The `Loyalty Level` changes to `Gold` with the following screen.



○ Create a new portfolio
● Retrieve selected portfolio
○ Update selected portfolio (add stock)
○ Delete selected portfolio

	Owner	Total	Loyalty Level
●	PoT	$151,910.00	Gold

__j. 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.



IBM Stock Trader
@IBMStockTrader

The IBM Stock Trader sample for IBM Cloud: github.com/IBMStockTrader. Notifications posted here when stock portfolios reach a new loyalty level.

IBM Stock Trader @IBMStockTrader · 3m
On Tuesday, April 17, 2018 at 12:27 AM UTC, PoT changed status from Basic to Gold. #IBMStockTrader

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

__l. Run `kubectl -n stocktrader get pods`

```
[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
```

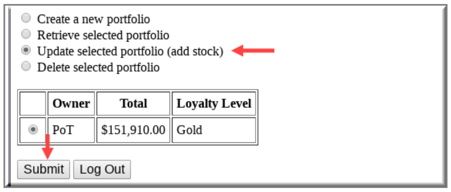
__m. Note the messaging pod name and get logs.

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

Step	Action

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

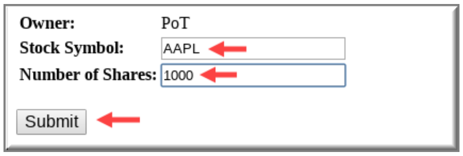
__p. Select Update selected portfolio (add stock)



○ Create a new portfolio
○ Retrieve selected portfolio
● Update selected portfolio (add stock)
○ Delete selected portfolio

	Owner	Total	Loyalty Level
●	PoT	$151,910.00	Gold

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



Owner: PoT
Stock Symbol: AAPL
Number of Shares: 1000

__r. Switch to the command line.

7

Explore Db2 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   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.

Step Action

```
[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.
```

__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
```

Step	Action

```
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#
```

__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>
```

Step	Action

__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.