

Developing and Deploying Microservices into IBM Cloud Private

John Alcorn (jalcorn@us.ibm.com)
Greg Hintermeister (gregh@us.ibm.com)

Find the latest version here:
<http://ibm.biz/StockTraderLab>



1. Introduction.....	3
Prepare: Log Into Skytap	6
2. Introduction to the StockTrader sample.....	7
3. Introduction to IBM Cloud Private User Interface.....	14
4. Using the Kubernetes CLI	33
5. Edit and Deploy a New Version of a StockTrader Micro-Service.....	41
Update Code.....	41
Build the project	44
Deploy to IBM Cloud Private	45
Run the code.....	46
Visualize the interactions	47
6. Scaling StockTrader.....	50
7. Download, Build, and Deploy a Node.js microservice	55
Download code from GitHub.....	55
Build the project	56
Deploy to IBM Cloud Private	56
Run the code.....	58
8. Optional: Deploying a Helm Chart from the Catalog	62
Create Persistent Volume	62
Get to the MQ Helm Chart	64
Deploy the MQ Helm Chart.....	68
Start Using MQ	70
Configuring MQ	72
Create Secrets.....	74
Switch to use YOUR MQ service	74
9. Conclusions.....	76
Contacts	76
Acknowledgements and Disclaimers.....	77



1. Introduction

In this lab, you will use and update an application in an IBM Cloud Private environment, built on Kubernetes. The lab will use a Skytap-hosted virtual machine (VM), which, for the purpose of the lab can be considered to be your on-premises environment running IBM Cloud Private that hosts your application and its required services.

The Skytap VM provided to you contains all necessary artifacts for the lab. The application artifacts are also available for review at <https://github.com/IBMStockTrader/> You can learn more about this application and how to deploy it in your own IBM Cloud Private environment via a series of blog posts from the IBM Hybrid Cloud CTO Office:

- Introduction to the StockTrader microservices application -
https://www.ibm.com/developerworks/community/blogs/5092bd93-e659-4f89-8de2-a7ac980487f0/entry/Building_Stock_Trader_in_IBM_Cloud_Private_2_1_using_Production_Services?lang=en
- Developing Microservices for IBM Cloud private -
https://www.ibm.com/developerworks/community/blogs/5092bd93-e659-4f89-8de2-a7ac980487f0/entry/Developing_microservices_for_IBM_Cloud_private?lang=en

The lab is divided into five main parts:

1. First, you will familiarize yourself with the StockTrader application and the Kubernetes cloud it's running in. StockTrader consists of multiple, small, specialized services (microservices) which work together as a stock portfolio-management system.
2. Second, you will learn how management of applications and services occurs within IBM Cloud Private and Kubernetes. You will learn how to scale up and down containers based upon workload requirements.
3. Third, you will edit Java code in one of the StockTrader microservices and deploy this new version into your IBM Cloud Private environment. This simulates the actual operations an administrator may perform when deploying and updating workloads. You will also use a Kubernetes tool to visualize how the various microservices interact at runtime.
4. Fourth, you will experience pulling a new repository from GitHub, for a Node.js-based client to StockTrader, and building and deploying and using it.
5. Lastly, you will experience deploying a helm chart from the catalog, to deploy your own instance of IBM MQ.



The Skytap VM features Ubuntu 16.04.3 LTS (xenial) as the operating system and has the following software installed:

- Docker 17.03.1-ce
- IBM Cloud Private - 2.1.0.1 (with Kubernetes 1.8.3)
- kubectl CLI version 1.9.1
- git CLI version 2.7.4
- Java 8
- Apache Maven version 3.5.0
- VS Code version 1.20

This particular Skytap VM is configured to host all portions of IBM Cloud Private: a master node for managing the environment, a proxy node for servicing network requests, and a worker node for hosting workloads. A more realistic environment would have separate virtual machines for each configuration, but for the purpose of this lab we've combined them into a single VM for convenience.

In addition to the above software, the Skytap VM also contains the artifacts needed to complete the lab. The following table points out the most important artifacts.

Directory	Description	Notes
/StockTrader/trader	Front-end User Interface	Java servlets
/StockTrader/portfolio	Portfolio management API	Uses JDBC to persist portfolios to DB2
/StockTrader/stock-quote	Stock prices via API Connect/Quandl	Uses Redis for caching
/StockTrader/loyalty-level	Determines a portfolio's level (Silver, Gold, etc.), and sends a message when it changes	Uses JMS to deliver messages to MQ
/StockTrader/messaging	Receives messages about loyalty level changes	Message Driven Bean (EJB)
/StockTrader/notification-twitter	Sends tweets	Uses Twitter4J
/StockTrader/notification-slack	Posts to a Slack channel Not used in this lab	Uses IBM Cloud Functions

You will download and use one more microservice, tradr, in a later section of this lab.

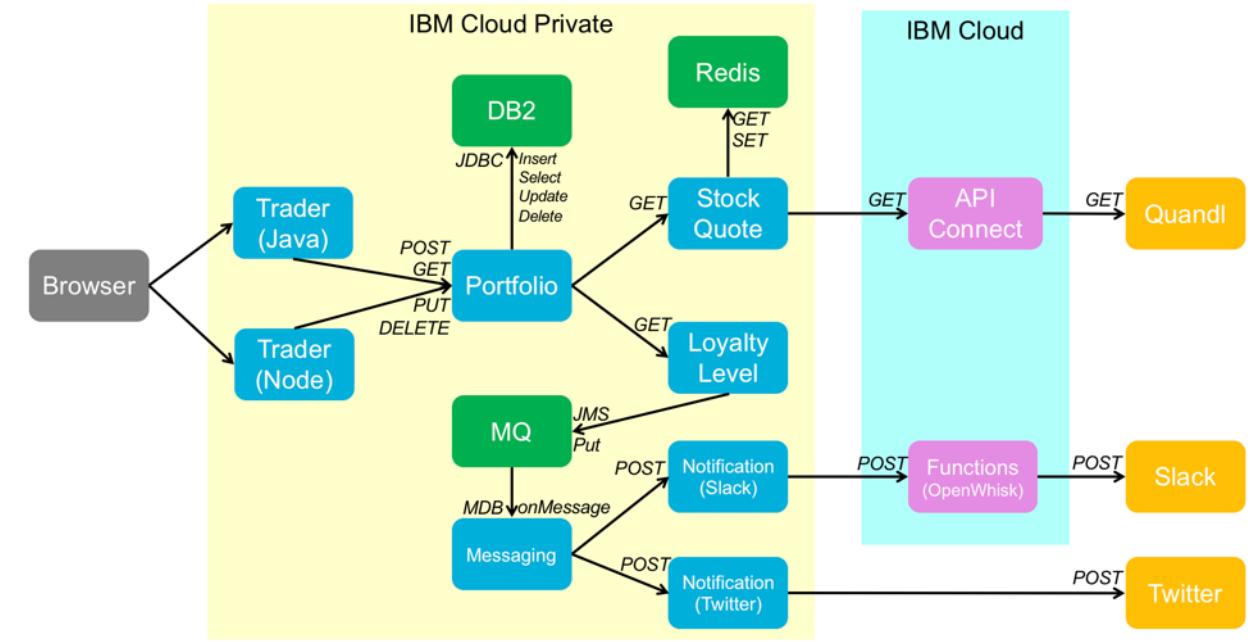


The application in this lab, **StockTrader**, is a microservice-based application utilizing various features of IBM WebSphere Liberty runtime (JAX-RS, JDBC, JMS, etc.) to interact with a set of database tables in DB2 via JDBC and to send notifications over MQ via JMS to Twitter. It also uses API Connect in the IBM public cloud to get stock quotes, and uses Redis for caching of those quotes.

As you can see in the following diagram (also shown at <https://github.com/IBMStockTrader/trader>, in its README.md), most microservice-to-microservice communication occurs via REST calls. There are two variants of the front-end web user interface – initially we'll use the Java Servlet based client for this lab, then we'll switch to the Node.js based client in a later section of the lab. The *portfolio* microservice uses JDBC to talk to DB2 for persistence. Two microservices, *loyalty-level* and *messaging*, use JMS to talk to MQ for their communications. Everything in the yellow box is running in IBM Cloud Private.

Microservices are colored in blue, their prerequisites are green. They reach out to services in the IBM public cloud (cyan), shown as purple boxes (they also authenticate against IBMid, an OpenID Connect provider in the IBM public cloud). And some services out on the internet are used as well, shown in orange. You can see how the microservices work together in the following diagram (note that the Slack notification is not enabled in this lab, as the Slack server to which it posts requires an internal IBM intranet id/pwd – so all notifications will be via Twitter in this lab).

Stock Portfolio sample for IBM Cloud Private



The end-to-end flow of the application is as follows: A user opens their browser and navigates to *trader's* web page where they can interact with various stock portfolios. When the user takes an action, such as buying stock, *trader* makes a REST call to *portfolio* to perform the requested action. As part of the buy stock action *portfolio* calls the *stock-quote* service to get the current stock price. *stock-quote* first checks a **Redis** cache for the stock price and if the price is not found in the cache *stock-quote* calls an **API Connect** service that fronts a **Quandl** Stock Data API for the price and stores the value in **Redis**.



before returning the value back to *portfolio*. Then *portfolio* calculates the total purchase price (number of shares * stock price) before storing that data in the user's portfolio within DB2. *portfolio* next calls *loyalty-level* to see if the new total portfolio value has changed the customer's loyalty level. If such a change occurs a message is put on an MQ queue and the new loyalty level is returned back to *portfolio* for storage in DB2. Finally, *portfolio* returns all this data as a JSON object back to *trader* for rendering in the user's browser. Asynchronously, an MDB in the *messaging* microservice receives the MQ message and calls either *notification-twitter* to send a tweet, or *notification-slack* (not used in this lab) to call IBM Cloud Functions to post to a Slack channel.

Finally, we will use the Consolas font for any commands you should enter in a terminal window. For example:

```
kubectl get deployments
```

should be entered in a terminal window (but not yet). Any actions you should take or forms to fill in will be **bolded**.

Prepare: Log Into Skytap

We are using Skytap, and each of you will have your own VM environment. Note that normally you would run IBM Cloud Private on your own on-premises hardware, but for the purposes of this lab, we have installed it into this cloud-based VM-hosting environment. Note the password is **A1rb0rn3**.

You can also click the keys icon  in the Skytap menu bar and click the **Insert** button next to the password for user **skytap**.



Skytap Tips:

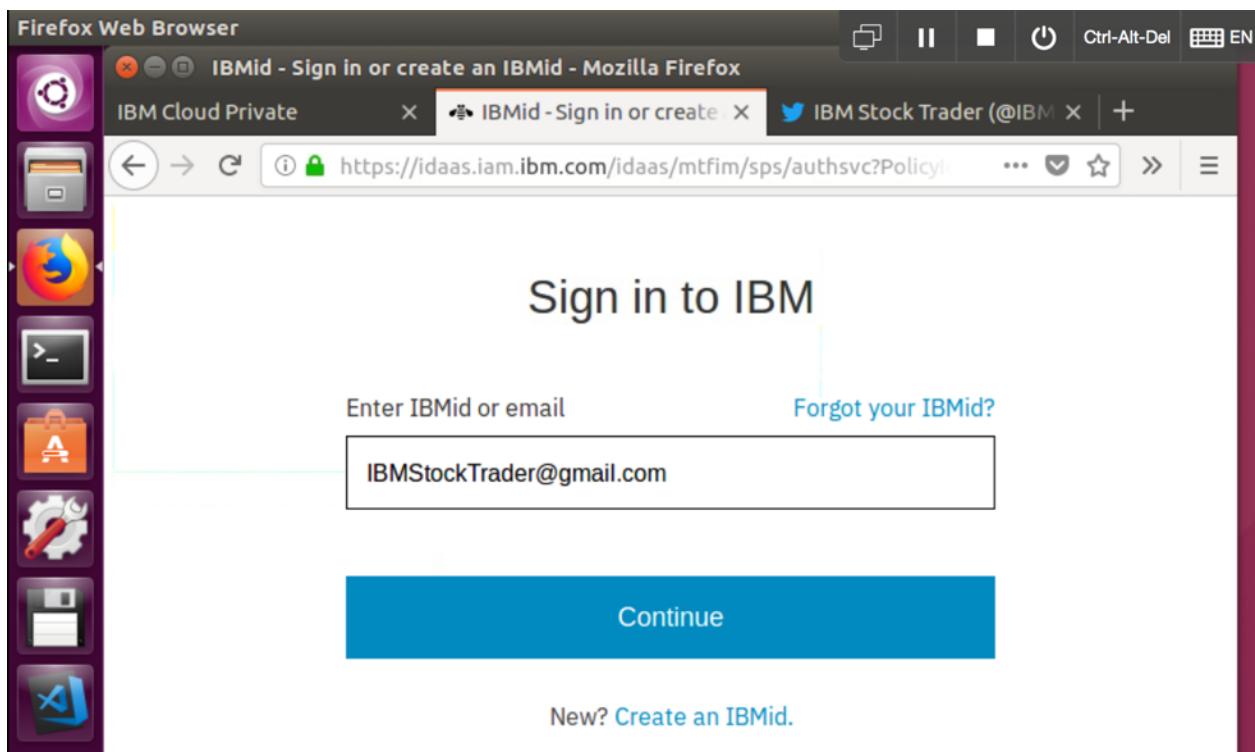
- All commands will be performed from a Terminal window within the Skytap VM in the browser (not directly from your laptop, which will not have connectivity to the Skytap VM other than working with its desktop via the web browser)
- Click out of the browser before performing any laptop keyboard shortcuts, otherwise the keyboard shortcuts will be done in the Skytap VM Linux OS
- Use the right click copy/paste, don't use keyboard shortcuts...they often don't work.
- To copy from the Skytap VM into your laptop, right click in the VM to copy the content, go to the top icon  and then copy the content again...this will add the content into your laptops cut/paste buffer.



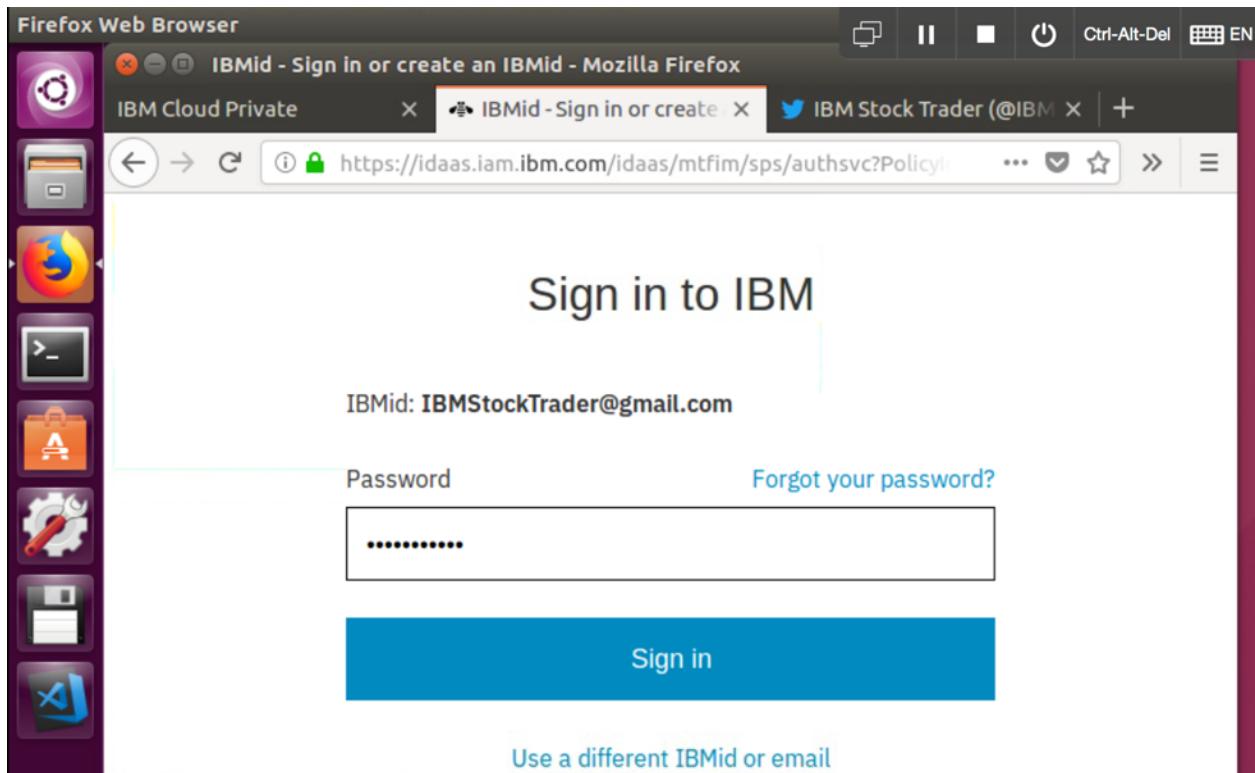
2. Introduction to the StockTrader sample

As noted in the introduction, StockTrader is a microservice-based stock portfolio management application which utilizes IBM WebSphere Liberty, IBM DB2, IBM MQ, Redis, and IBM API Connect as the underlying runtime and storage mechanisms. All services, except API Connect (which runs in the IBM public cloud), are configured to run in the IBM Cloud Private instance within your Skytap VM.

1. Open the Firefox browser and navigate to <https://10.0.0.1:32389/trader/summary>
2. You will be rerouted to an OpenID Connect authentication page. To log in, enter your **IBM ID**. Your IBM ID is the same email you used to register for this conference. If you don't have an IBM ID, click the "Create an IBMid" link and follow the instructions (an email will be sent with a confirmation code once you submit your info, and the account won't be activated until you enter that code).



3. Enter your password for your IBM ID. Note that IBM employees using their work e-mail addresses will be routed to a different page to do a w3id login with their intranet ID and password.

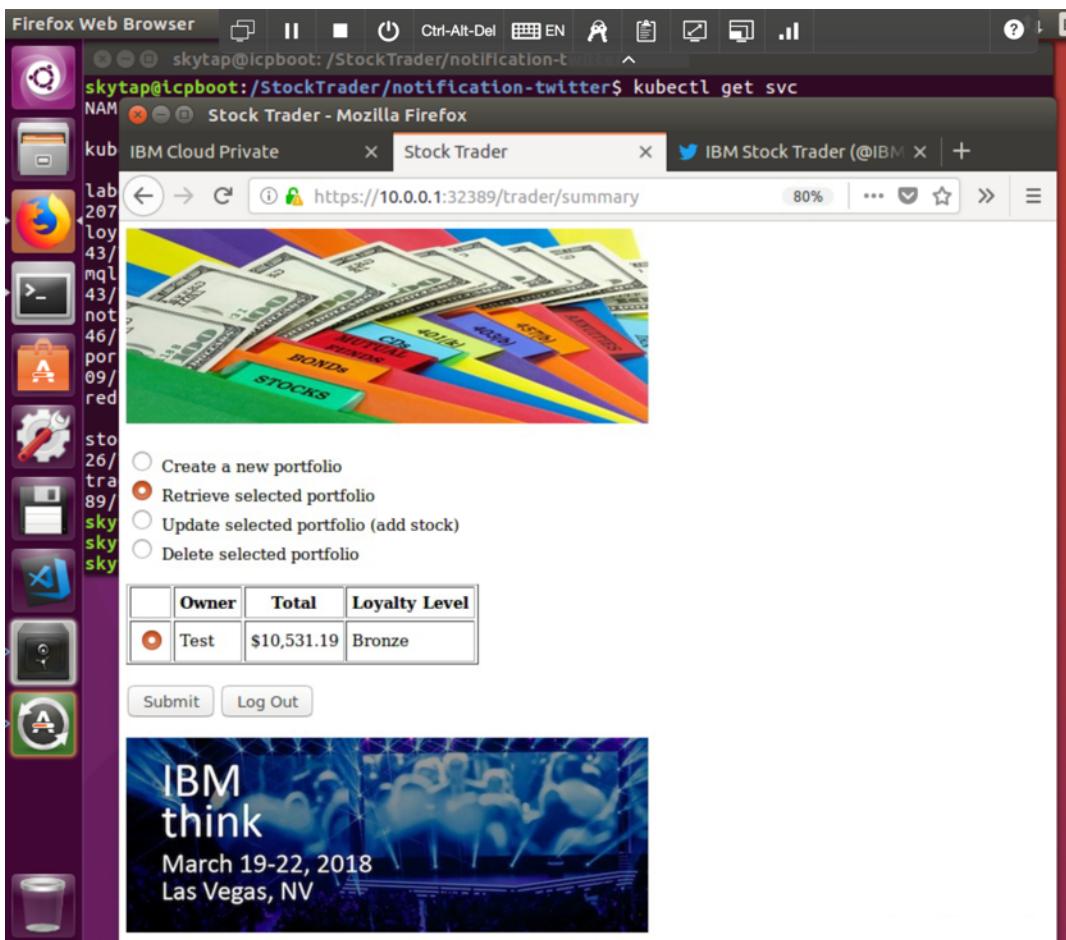


You may get told that you need to perform two-factor authentication, due to this being the first time your IBM ID has logged in from this location (in SkyTap). If so, it will send you a confirmation code, as a text to your mobile device or to your e-mail, and you will need to enter that code to complete your login.

4. Upon a successful login, you will see the StockTrader landing page, which provides a list of the stock portfolios you manage along with a set of actions you can take for each portfolio. The list should be pre-populated with a portfolio named Test, which contains a wide variety of stocks.

Note that there is a daily limit on how many stock quotes can be served up from **Quandl** to our **API Connect** service, and with many lab participants all running the sample simultaneously, that rate limit may be hit. In that case, the sample will revert to using values already cached in Redis, and the price for any new stock symbols not already in Redis will not be available, resulting in the UI showing “Error” for that stock. This will also be shown if an invalid stock ticker symbol is entered.





5. Select the **Create a new portfolio** radio button and click **Submit** to create a new portfolio, as shown here.

The image shows a close-up of the dashboard's portfolio creation section. The 'Create a new portfolio' radio button is selected (indicated by a red circle). A blue arrow points from the 'Submit' button at the bottom to the selected radio button.

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

	Owner	Total	Loyalty Level
<input checked="" type="radio"/>	EM	\$5,688.93	Basic
<input type="radio"/>	EM1	\$2,158,753.68	Platinum

Submit

6. Provide a name for the new portfolio owner and click the **Submit** button to create the new portfolio. Make sure not to enter any spaces in the name, or other values that would be illegal as part of a URL.



Owner:

When you click Submit, the *trader* service (our user interface) makes a REST call to *portfolio* service which inserts a new row in DB2's portfolio table. Now that you've created a portfolio you need to buy some stocks for that person.

7. Click the **Update selected portfolio (add stock)** radio button and the radio button next to your newly created portfolio before clicking **Submit**.

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

	Owner	Total	Loyalty Level
<input type="radio"/>	EM	\$5,688.93	Basic
<input type="radio"/>	EM1	\$2,158,753.68	Platinum
<input checked="" type="radio"/>	Fred	\$0.00	Basic

8. On the Add Stock page you enter a **Stock Symbol** and a **Number of Shares** you want to buy. You can enter any actual NYSE stock symbol and any number of shares before clicking the **Submit** button (as shown on the next diagram).

Owner: Fred

Stock Symbol:

Number of Shares:

9. When you click Submit *trader* calls *portfolio* which then calls *stock-quote* to retrieve the current stock price for that stock. There are some smarts in *stock-quote*: A Redis database caches previously retrieved stock prices to reduce load against the Quandl back end stock quote service (and to improve performance). If there is no cached value in Redis, *stock-quote* calls an API Connect service, which calls Quandl for the most recent market price, stores the value in Redis, and returns the quote back to *portfolio*. The *portfolio* service updates DB2 with the new stocks and portfolio contents which you'll see when you're returned back to the main landing page.





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

	Owner	Total	Loyalty Level
<input checked="" type="radio"/>	EM	\$5,688.93	Basic
<input type="radio"/>	EM1	\$2,158,753.68	Platinum
<input type="radio"/>	Fred	\$7,328.00	Basic

10. Feel free to add more stocks to your new portfolio by repeating step 8. If you want to *sell* a stock enter a **negative** number for the number of shares you'd like to buy. You may notice that the loyalty level changes from Basic when a certain portfolio value threshold is reached. Any time a stock transaction takes place *portfolio* calls the *loyalty-level* service to compute the current loyalty level for the portfolio. When a threshold is reached, the database is updated with the new loyalty value and a JMS message is put onto an MQ queue. The *messaging* service contains a Message Driven Bean which reads from this queue and calls a *notification-twitter* service, which sends a message to Twitter about that change in level.
11. The various loyalty levels are defined as:
 - Basic: < \$10,000
 - Bronze: < \$50,000
 - Silver: < \$100,000
 - Gold: < \$1,000,000
 - Platinum: > \$1,000,000
12. Feel free to buy or sell more stocks to change the different loyalty levels. You can open a new tab in your browser and go to <https://twitter.com/IBMStockTrader> to see the tweets:



skytap@lcpboot:/StockTrader/notification-l

IBM Stock Trader (@IBMS stockTrader) | Twitter - Mozilla Firefox

Twitter, Inc. (US) https://twitter.com/IBMS stockTrader

Search Twitter Have an account? Log In

IBM Stock Trader (@IBMS stockTrader)

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

United States Joined October 2017

New to Twitter? Sign up now to get your own personalized timeline!

Tweets 16 Following 2 Followers 4 Follow

Tweets **Tweets & replies**

- IBM Stock Trader @IBMS stockTrader · 40m On Friday, February 16, 2018 at 9:06 AM UTC, Test changed status from Basic to Bronze. #IBMS stockTrader
- IBM Stock Trader @IBMS stockTrader · 46m On Friday, February 16, 2018 at 9:00 AM UTC, John changed status from Bronze to Basic. #IBMS stockTrader
- IBM Stock Trader @IBMS stockTrader · 17h On Thursday, February 15, 2018 at 4:37 PM UTC, John changed status from Silver to Bronze. #IBMS stockTrader

13. You can also view the full details for your selected portfolio by selecting the **Retrieve selected portfolio** radio button and the portfolio you'd like to see before clicking the **Submit** button.

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

	Owner	Total	Loyalty Level
<input type="radio"/>	EM	\$5,688.93	Basic
<input type="radio"/>	EM1	\$2,158,753.68	Platinum
<input checked="" type="radio"/>	Fred	\$8,218.56	Basic

Submit



Stock Portfolio for Fred:

Symbol	Shares	Price	Date Quoted	Total
IBM	50	\$146.56	2017-09-26	\$7,328.00
T	23	\$38.72	2017-09-26	\$890.56

Total Portfolio Value: \$8,218.56

Loyalty Level: Basic

OK

Here, the *trader* service has again called *portfolio* to gather all the details for this particular portfolio from DB2. Click the **OK** button to return to the portfolio list.

14. You can also delete a portfolio (and all of its stocks) by clicking the **Delete selected portfolio** radio button, clicking on a portfolio, and then clicking **Submit**. This is shown in the following diagram.

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

	Owner	Total	Loyalty Level
<input type="radio"/>	EM	\$5,688.93	Basic
<input checked="" type="radio"/>	EM1	\$2,158,753.68	Platinum
<input type="radio"/>	Fred	\$8,218.56	Basic

Submit

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

	Owner	Total	Loyalty Level
<input checked="" type="radio"/>	EM	\$5,688.93	Basic
<input type="radio"/>	Fred	\$8,218.56	Basic

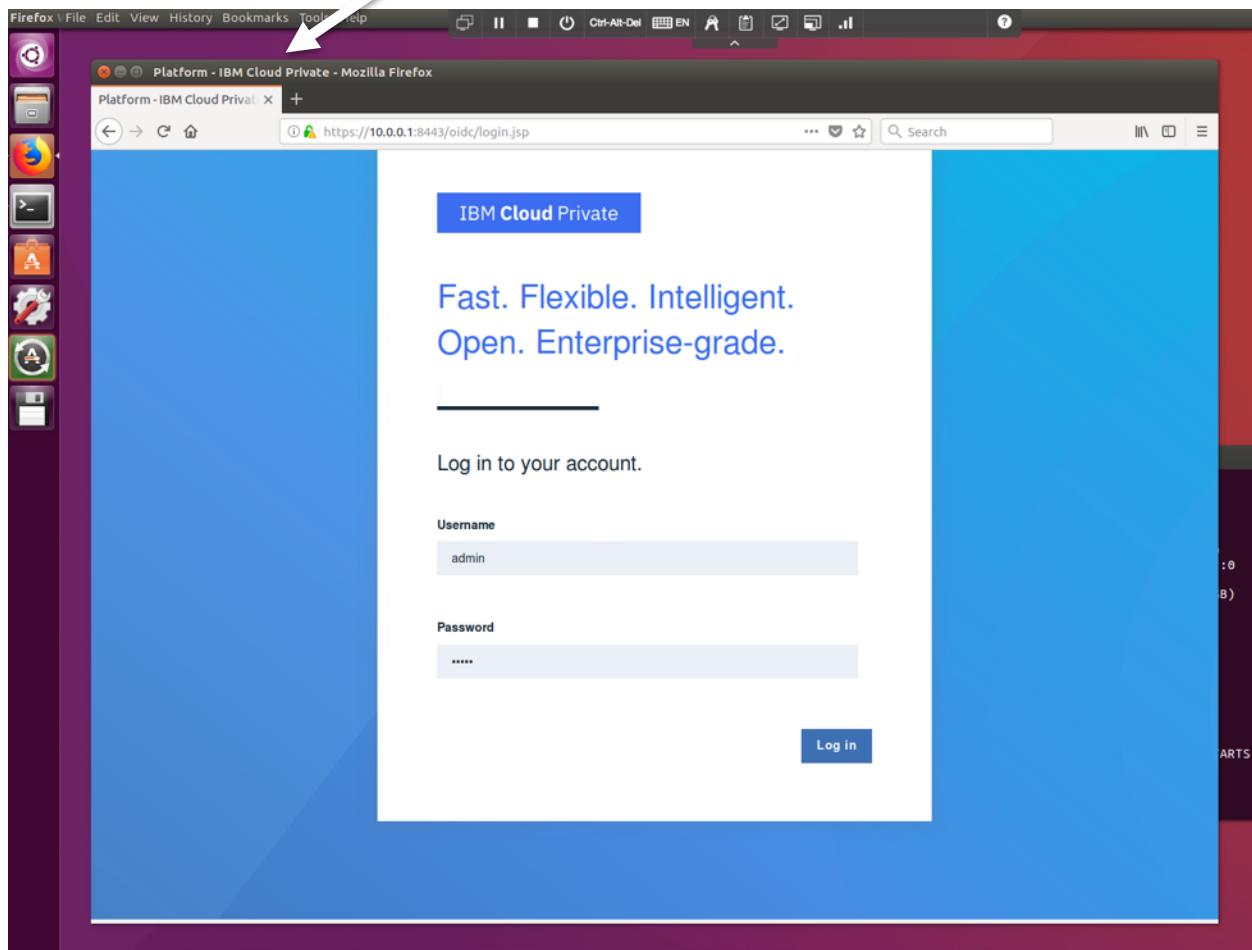
Submit



3. Introduction to IBM Cloud Private User Interface

In this section, you will utilize the IBM Cloud Private web console to view how StockTrader is deployed within IBM Cloud Private.

1. Within Firefox you should see a tab called **IBM Cloud private**. Click on this tab to continue.



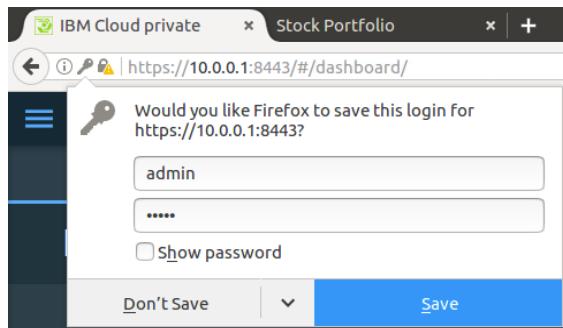
If there is no IBM Cloud Private tab in Firefox please navigate to <https://10.0.0.1:8443>

2. Enter the following usernames and passwords before clicking the “**Log in**” button:

User ID	admin
Password	admin



If prompted by Firefox to save this login select **Don't save** to dismiss the dialog.



You've arrived at the IBM Cloud Private Dashboard landing page which provides a brief overview of your environment.

The **System Overview** section provides a high-level health about the worker nodes where workloads actually run (titled **Nodes**, first widget), the Shared Storage utilization for persistent workloads (titled **Shared Storage**, second widget), and the status of all deployed applications (titled **Applications**, final widget).

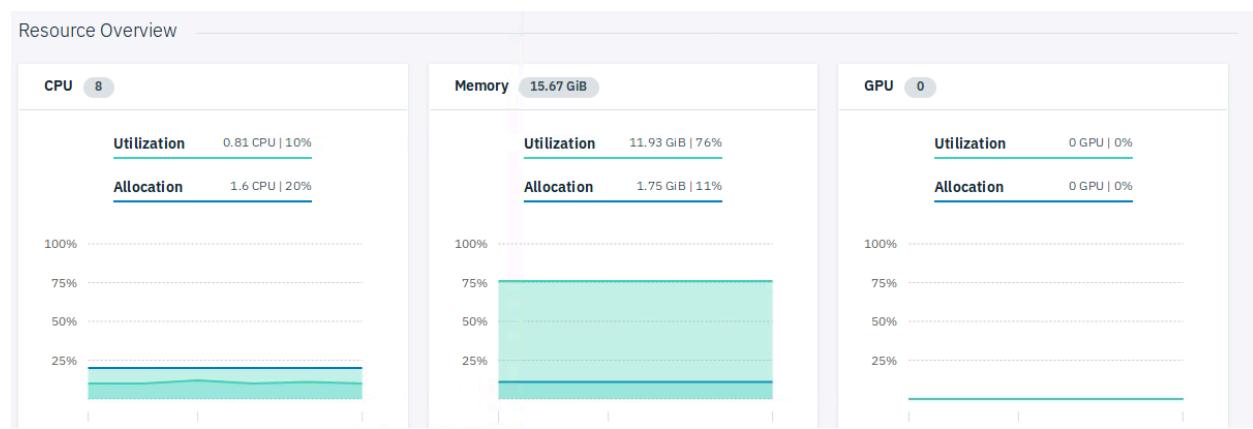
In the **Nodes** widget, we see there is one active worker node. This worker node is where our workloads, or applications and services, are deployed. These applications and services are Dockerized versions of IBM WebSphere Liberty, IBM DB2, IBM MQ, Redis, along with other Kubernetes internal services.

The **Shared Storage** widget shows that we have 49GB of shared storage utilized by workloads to store persistent data. This does not mean all 49GB have been utilized, only claimed. Since our workloads and services are Dockerized it is important to remember that Docker containers should be considered ephemeral and any data stored within a container will disappear if that container is deleted. Shared storage allows data persistence outside a Docker container and outside the Docker lifecycle.



The **Applications** widget shows that there are 17 applications/services deployed and started in our environment. This includes not only the various microservices for StockTrader but also the dependent services, like IBM DB2, StockTrader depends on along with other internal Kubernetes services.

If you scroll down the landing page you'll find the **Resources Overview** section. This includes current **CPU**, **Memory**, and **GPU** utilization and allocation.



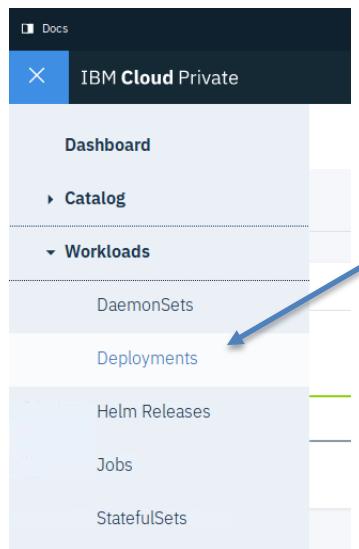
The **CPU** widget shows the total number of CPUs available across all the worker nodes, how much of the CPU is utilized and how much is allocated. Here we have 8 CPUs available across all workers (this VM has eight CPUs), the various workloads are using a total of 2.86 CPUs while only 0.4 CPUs have been directly assigned to a workload (dedicated compute resource)

Likewise, the **Memory** widget shows our workers have a total of 7.8 GB of RAM (this VM has 8GB of RAM), and that RAM is 90% utilized but only 7% has been directly requested or dedicated to a workload.

Finally, the **GPU** widget shows how many GPU resources are configured and available. If our workers had access to NVIDIA GPUs they would appear in this list and become available for analytics workloads to exploit.

3. Scroll back to the top of the browser window and select the menu icon in the upper left corner . This opens the Menu sidebar.





Click on **Workloads** and then **Deployments** in the menu.

This opens the list of applications and microservices (called *deployments* in Kubernetes) currently running in this IBM Cloud Private environment.



NAME	NAMESPACE	DESIRED	CURRENT	READY	AVAILABLE	CREATION TIME	ACTION
calico-policy-controller	kube-system	1	1	1	1	Jan 4th 2018 at 9:52 AM	⋮
catalog-catalog-controller-manager	kube-system	1	1	1	1	Jan 4th 2018 at 9:58 AM	⋮
default-http-backend	kube-system	1	1	1	1	Jan 4th 2018 at 9:57 AM	⋮
elasticsearch-client	kube-system	1	1	1	1	Jan 4th 2018 at 9:57 AM	⋮
elasticsearch-master	kube-system	1	1	1	1	Jan 4th 2018 at 9:57 AM	⋮
heapster	kube-system	1	1	1	1	Jan 4th 2018 at 9:57 AM	⋮
helm-api	kube-system	1	1	1	1	Jan 4th 2018 at 9:59 AM	⋮
helmrepo	kube-system	1	1	1	1	Jan 4th 2018 at 9:59 AM	⋮
kube-dns	kube-system	1	1	1	1	Jan 4th 2018 at 9:53 AM	⋮
labdb2-ibm-db2oltp-dev	default	1	1	1	1	Feb 6th 2018 at 10:00 AM	⋮

Items running in the *kube-system* namespace are internal IBM Cloud Private/Kubernetes services and can usually be ignored. StockTrader microservices and dependencies are running in the *default* namespace. Applications can also create their own custom namespaces; we will see an example of this later on, when we install Weave.

You will notice that there is a DB2 deployment named **labdb2-ibm-db2oltp-dev**; this hosts the DB2 database StockTrader uses for storing portfolio information. Endpoint and credential info for this DB2 database are stored in a Kubernetes secret (named *db2* in this case) used by *portfolio*, which allows such info to be configured at deployment time, rather than being hard-coded in the microservice's source code. Values in the secret are exposed to the microservice as environment variables, and can be updated (like if a password gets updated) without having to modify and redeploy the microservice.



db2

100 items per page | 1-1 of 1 items

Create Deployment +

NAME	NAMESPACE	DESIRED	CURRENT	READY	AVAILABLE	CREATION TIME	ACTION
labdb2-ibm-db2oltp-dev	default	1	1	1	1	Feb 6th 2018 at 10:00 AM	⋮

The data in this table indicate that for this DB2 deployment, there is one desired container instance, one currently running instance, one instance which has the currently desired configuration, and one instance that is up and running (corresponding to columns *Desired*, *Current*, *Ready*, and *Available*, respectively).

If you click on the namespace dropdown at the top right and choose *default*, it will filter the view down to just the StockTrader microservices: *trader*, *portfolio*, *stock-quote*, *loyalty-level*, *messaging*, and *notification-twitter* (and the *db2* and *redis* dependencies).

Firefox Web Browser

IBM Cloud Private - Mozilla Firefox

IBM Cloud Private

Docs admin

IBM Cloud Private Create resource Support

Deployments default

Search items Create Deployment +

20 items per page | 1-8 of 8 items 1 of 1 pages < 1 >

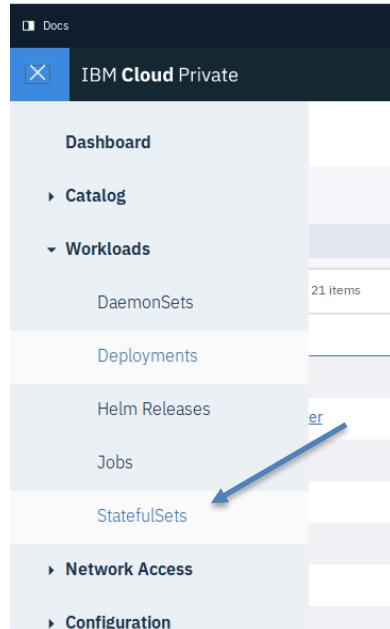
NAME	NAMESPACE	DESIRED	CURRENT	READY	AVAILABLE	CREATION TIME	ACTION
labdb2-ibm-db2oltp-dev	default	1	1	1	1	Feb 6th 2018 at 10:00 AM	⋮
loyalty-level	default	1	1	1	1	Feb 12th 2018 at 12:34 PM	⋮
messaging	default	1	1	1	1	Feb 15th 2018 at 11:57 PM	⋮
notification-twitter	default	1	1	1	1	Feb 16th 2018 at 12:59 AM	⋮
portfolio	default	1	1	1	1	Feb 12th 2018 at 12:30 PM	⋮
redislabs-redis	default	1	1	1	1	Feb 6th 2018 at 11:30 AM	⋮
stock-quote	default	1	1	1	1	Feb 12th 2018 at 12:33 PM	⋮
trader	default	1	1	1	1	Feb 12th 2018 at 12:29 PM	⋮



Later in this lab you will scale up and scale down one of our microservices. You can perform scaling operations on this page via the action icon (the three vertical dots) in the right-most column.

4. While most applications deploy as “Deployments”, a few services deploy as “Stateful Sets”, including MQ, used by Stock Trader. In fact, soon you will deploy your own instance of MQ.

Open the menu again and click on **Workloads** then **StatefulSets**.



The screenshot shows the 'StatefulSets' list view. At the top, there is a search bar and a 'Create StatefulSet' button. Below is a table with the following data:

NAME	NAMESPACE	DESIRED	CURRENT	CREATION TIME	ACTION
elasticsearch-data	kube-system	1	1	Jan 4th 2018 at 9:57 AM	⋮
icp-ds	kube-system	1	1	Jan 4th 2018 at 9:53 AM	⋮
image-manager	kube-system	1	1	Jan 4th 2018 at 9:57 AM	⋮
mqlab-ibm-mq	default	1	1	Feb 6th 2018 at 1:12 PM	⋮

Here you can see there are four StatefulSets: one for IBM MQ (*mqlab-ibm-mq*) and three used by IBM Cloud Private itself. Remember, the *kube-system* namespace is used by IBM to run essential services like image management and elastic search. A StatefulSet allows you to maintain unique information



across pod reschedules or restarts. This is important for IBM MQ since it relies on unique server information to maintain internal state. If a container should fail this unique data is persisted and reused as part of recovery.



5. Back in the Menu click on **Network Access** then **Services**. This page lists how our applications are exposed to the outside world and to each other.

The screenshot shows the IBM Cloud Private interface. At the top, there is a dark header bar with a 'Docs' link and an 'X' button. Below this is a blue navigation bar with the text 'IBM Cloud Private'. On the left, a sidebar menu is visible with the following items:

- Dashboard
- Catalog
- Workloads
- Network Access (with a blue arrow pointing to the 'Services' option)
- Configuration
- Platform
- Manage
- Command Line Tools

On the right side of the screen, there is a table titled 'Services' with one item listed:

ITEMS	NAMESPACE
1	kube-system



IBM Cloud Private - Mozilla Firefox

IBM Cloud Private

https://10.0.0.1:8443/console/access/services

admin

IBM Cloud Private

Services

All namespaces

NAME	NAMESPACE	CREATION TIME	ACTION
elasticsearch	kube-system	Jan 4th 2018 at 9:57 AM	⋮
elasticsearch-data	kube-system	Jan 4th 2018 at 9:57 AM	⋮
elasticsearch-discovery	kube-system	Jan 4th 2018 at 9:57 AM	⋮
heapster	kube-system	Jan 4th 2018 at 9:57 AM	⋮
helm-api	kube-system	Jan 4th 2018 at 9:59 AM	⋮
helmrepo	kube-system	Jan 4th 2018 at 9:59 AM	⋮
image-manager	kube-system	Jan 4th 2018 at 9:57 AM	⋮
kube-dns	kube-system	Jan 4th 2018 at 9:53 AM	⋮
kubernetes	default	Jan 4th 2018 at 9:52 AM	⋮
labdb2-ibm-db2oltp-dev	default	Feb 6th 2018 at 10:00 AM	⋮
logging-kibana	kube-system	Feb 16th 2018 at 12:41 AM	⋮
logstash	kube-system	Jan 4th 2018 at 9:57 AM	⋮
loyalty-level-service	default	Feb 12th 2018 at 12:34 PM	⋮
metering-dm	kube-system	Jan 4th 2018 at 9:57 AM	⋮

You'll notice **labdb2-ibm-db2oltp-dev** is listed again. Click on the **labdb2-ibm-db2oltp-dev** link.

O: db2

Create Service +

NAME	NAMESPACE	CREATION TIME	ACTION
labdb2-ibm-db2oltp-dev	default	Feb 6th 2018 at 10:00 AM	⋮



labdb2-ibm-db2oltp-dev

Overview

Service details	
Type	Detail
Name	labdb2-ibm-db2oltp-dev
Namespace	default
Creation time	Feb 6th 2018 at 10:00 AM
Type	NodePort
Labels	app=labdb2-ibm-db2oltp-dev,chart=ibm-db2oltp-dev-1.1.1,heritage=Tiller,release=labdb2
Selector	app=labdb2-ibm-db2oltp-dev
IP	10.0.0.14
Port	ibm-db2oltp-dev 50000/TCP; ibm-db2oltp-dev-text 55000/TCP
Node port	<u>ibm-db2oltp-dev 30042/TCP</u> ← <u>ibm-db2oltp-dev-text 32070/TCP</u>
Session affinity	None

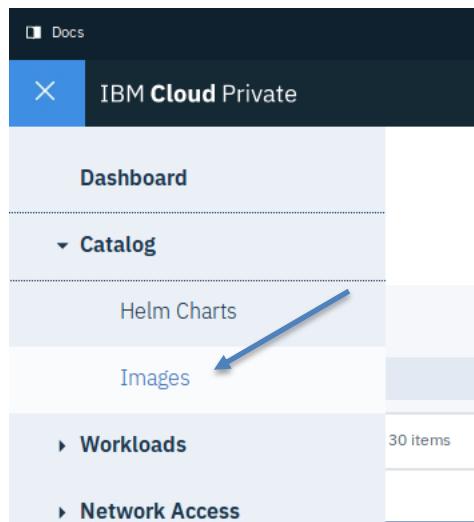
This data indicates that DB2 is exposed within IBM Cloud Private and Kubernetes to other services over port 50000 (the **Port** value) and that it is exposed *externally* at port 30042 (the **Node Port** value). By default, applications and services are not exposed outside of IBM Cloud Private or Kubernetes. They can be exposed to other services internally by specifying a port and externally by exposing a NodePort. IBM Cloud Private and Kubernetes then knows how to route any traffic received on port 30042 to the DB2 container's port 50000.

Back in the Service list you should also see entries for *loyalty-level-service*, *notification-service*, *portfolio-service*, *stock-quote-service*, and *trader-service*.



6. Go back to the Menu , expand **Catalog**, and select **Images**.



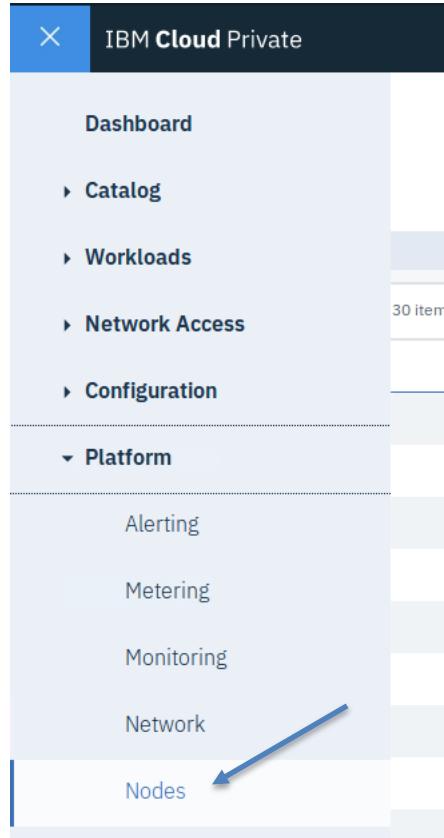


The screenshot shows a Firefox browser window with the URL <https://10.0.0.1:8443/console/images>. The page displays the 'Images' section of the IBM Cloud Private Docker Registry. It includes a search bar, a table of images with columns for NAME, OWNER, and SCOPE, and a sidebar with various icons.

NAME	OWNER	SCOPE
default/loyalty-level	default	namespace
default/messaging	default	namespace
default/notification-twitter	default	namespace
default/portfolio	default	namespace
default/stock-quote	default	namespace
default/trader	default	namespace

IBM Cloud Private ships with its own Docker Registry. This panel lets you view and interact with the Docker images already in the registry. Here you can see there are six images corresponding to the six microservices which comprise the StockTrader application. Clicking on an image name lets you see the different versions available for a given image along with some metadata about that image.

7. Go back to the Menu  expand **Platform**, and select **Nodes**.



The Nodes page shows all nodes in this Kubernetes cluster, the role they play, their architecture (ICP runs on x86, Power Linux, and zLinux), as well as its status to receive additional deployments.

Nodes

NAME	ROLE	ARCHITECTURE	STATUS	SCHEDULABLE	CREATION TIME
10.0.0.1	proxy, management, master	amd64	Active	Schedulable	Jan 4th 2018 at 9:52 AM

Since this VM contains all the IBM Cloud Private/Kubernetes roles you see only one node listed at the bottom of the page. If we separated each role into its own virtual machine you would see multiple nodes listed in the table of nodes. If you click on **10.0.0.1** you are taken to the live statistics and currently running workloads on that particular node.



The screenshot shows the IBM Cloud Private console interface running in a Firefox browser window. The left sidebar contains various icons for managing resources like nodes, clusters, and databases. The main content area has two tabs: "Nodes" and "Pods".

Nodes Tab:

- Overview:** Shows node details for node 10.0.0.1. The table includes columns for Type, Detail, Hostname, Unschedulable, Status, Address, Labels, CPU, Memory, CPU Request, CPU Limits, Memory Request, Memory Limits, and GPU Limits.

Pods Tab:

- Search:** A search bar at the top of the list.
- Table Headers:** NAME, NAME SPACE, STATUS, PORTS, POD IP, READY, START TIME, ACTION.
- Data:** A list of 78 items across 8 pages. Some entries are failed, while others are running or succeeded. Examples include auth-pki-xyz, auth-idp-xyz, auth-psp-xyz, auth-pdo-xyz, calico-node-xyz, calico-policy-controller-xyz, calico-catalog-server-xyz, calico-catalog-controller-xyz, calico-catalog-xyz, calico-ui-xyz, cloudbeaver-db-xyz, configure-calico-mesh-xyz, deflty-lb-xyz, deflty-lb-xyz, elasticsearch-client-xyz, elasticsearch-client-xyz, elasticsearch-data-0, and elasticsearch-master-xyz.

By default, 20 pods are shown per page; you can use the arrows at the top of the Pods section to see the later pages. If you click on a pod name, **portfolio-647fcc8fcd-2l7pv** for example (the generated suffix after **portfolio** may be different in your environment), you can see the details for that pod.



The screenshot shows a Firefox browser window titled "IBM Cloud Private - Mozilla Firefox". The URL is <https://10.0.0.1:8443/console/platform/nodes/10>. The page displays the "IBM Cloud Private" interface with a sidebar containing icons for Docs, Create resource, and Support. The main content area shows the "Nodes / 10.0.0.1 / portfolio-647fcc8fcd-2l7pv /" path. Below this, there are four tabs: Overview (selected), Containers, Events, and Logs. The "Overview" section is titled "Pod details" and contains a table with the following data:

Type	Detail
Name	portfolio-647fcc8fcd-2l7pv
Namespace	default
Start time	Feb 12th 2018 at 12:30 PM
Labels	app=portfolio,pod-template-hash=2039774978,solution=stock-trader
PodSecurityPolicy applied	default
Node	10.0.0.1
IP	10.1.64.126
Status	Running

At the top of the pod details screen you will see four tabs with extra information: Overview, Containers, Events, and Logs. The **Overview** tab is shown above.

The **Containers** tab shows all the running containers in this pod. For the portfolio pod, there is only one container running.

The screenshot shows the "Containers" tab for the "portfolio-647fcc8fcd-2l7pv" pod. The table lists one container:

NAME	IMAGE	PORT	STATE	RESTART COUNT
portfolio	mycluster.icp:8500/default/portfolio:latest	9080/TCP,9443/TCP	Running	1



Clicking on the **portfolio** link shows details about that Docker container.

Nodes / 10.0.0.1 / portfolio-647fcc8fcfd-2l7pv / Containers / portfolio / portfolio

Overview

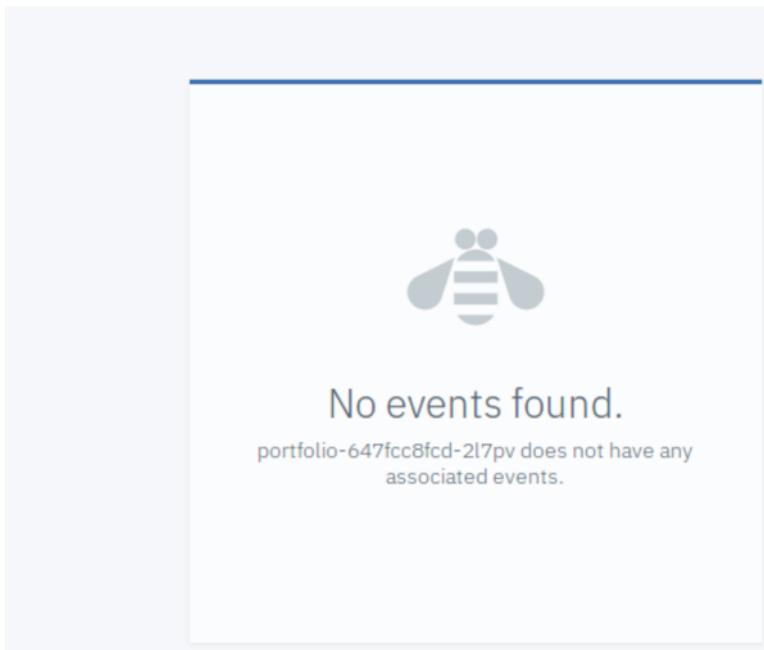
Container details	
Type	Detail
Container ID	docker://9a5195b0d12f28161afb88ffba187b4dc7e3449fc86ac483be9632e6136d98e5
Image	mycluster.icp:8500/default/portfolio:latest
Image ID	docker-pullable://mycluster.icp:8500/default/portfolio@sha256:5c9ca879cc338e731d590135317f638c6cf68f4e41cbbe7c1ffbc02c2ee687a6
Port	9080/TCP,9443/TCP
Command	-
Environment variables	JDBC_HOST={secretKeyRef:{name=db2,key=host}},JDBC_PORT={secretKeyRef:{name=db2,key=port}},JDBC_DB={secretKeyRef:{name=db2,key=db}},JDBC_ID={secretKeyRef:{name=db2,key=id}},JDBC_PASSWORD={secretKeyRef:{name=db2,key=pwd}},JWT_AUDIENCE={secretKeyRef:{name=jwt,key=audience}},JWT_ISSUER={secretKeyRef:{name=jwt,key=issuer}}
Resource	-

Back on the pod page, the **Events** tab is useful when debugging deployment problems for a given pod. In our case, there were no problems, so it says “No events found”.



Nodes / 10.0.0.1 / portfolio-647fcc8fcd-2l7pv /
portfolio-647fcc8fcd-2l7pv

Overview Containers Events **Logs**



Finally, the **Logs** tab shows you the logs emitted by the container. For Liberty containers, you'll see messages written via Java's `System.out.println()` or methods of `java.util.logging.Logger` (depending on the `consoleLogLevel` set in the `logging` stanza in your microservice's `server.xml`). This is useful for debugging any application, middleware, or container configuration problems.



Nodes / 10.0.0.1 / portfolio-647fcc8fcfcd-2l7pv /

portfolio-647fcc8fcfcd-2l7pv

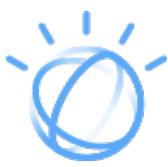
Overview Containers Events Logs

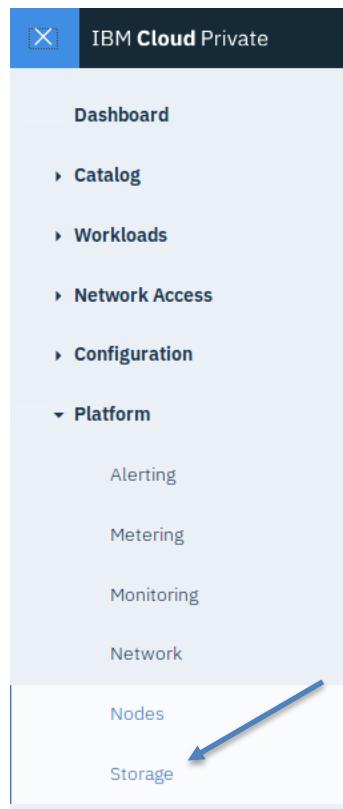
Search

portfolio ▾

```
[INFO ] SRVE9103I: A configuration file for a web server plugin was aut
[INFO ] Setting the server's publish address to be /
[INFO ] SRVE0242I: [Portfolio] [/portfolio] [com.ibm.hybrid.cloud.sample]
[INFO ] J2CA8050I: An authentication alias should be used instead of de
[INFO ] CWRLS0010I: Performing recovery processing for local WebSphere
[INFO ] CWRLS0007I: No existing recovery log files found in /opt/ibm/wl
[INFO ] CWRLS0006I: Creating new recovery log file /opt/ibm/wlp/output/
[INFO ] CWRLS0006I: Creating new recovery log file /opt/ibm/wlp/output/
[INFO ] CWRLS0007I: No existing recovery log files found in /opt/ibm/wl
[INFO ] CWRLS0006I: Creating new recovery log file /opt/ibm/wlp/output/
[INFO ] CWRLS0006I: Creating new recovery log file /opt/ibm/wlp/output/
[INFO ] CWRLS0012I: All persistent services have been directed to perf
[INFO ] WTRN0135I: Transaction service recovering no transactions.
[INFO ] DSRA8203I: Database product name : DB2/LINUXX8664
[INFO ] DSRA8205I: JDBC driver name : IBM Data Server Driver for JDBC
[INFO ] DSRA8204I: Database product version : SQL11012
[INFO ] DSRA8206I: JDBC driver version : 4.22.29
```

8. Go back to the Menu  expand **Platform**, and select **Storage**.





Storage								Create resource	Support	
PersistentVolume		PersistentVolumeClaim								
Name	Type	Capacity	Access Mode	Reclaim Policy	Status	Claim	Creation Time	Action		
cloudant-10.0.0.1		2Gi	RWO	Delete	Bound	kube-system/icp-ds-icp-ds-0	Jan 4th 2018 at 9:53 AM	⋮		
db2vol01	Hostpath	5Gi	RWO	Retain	Bound	default/labdb2-ibm-db2oltp-dev-data-stor	Feb 6th 2018 at 9:59 AM	⋮		
image-manager-10.0.0.1		20Gi	RWO	Retain	Bound	kube-system/image-manager-image-manager-0	Jan 4th 2018 at 9:57 AM	⋮		
logging-es-pv-10.0.0.1		20Gi	RWO	Delete	Bound	kube-system/elasticsearch-data-elasticsearch-data-0	Jan 4th 2018 at 9:57 AM	⋮		
mqvol1	Hostpath	2Gi	RWO	Retain	Bound	default/mqlab-ibm-mq-data-mqlab-ibm-mq-0	Feb 6th 2018 at 1:12 PM	⋮		
redisvol01	Hostpath	8Gi	RWO	Retain	Released	default/redislab-redis	Feb 6th 2018 at 10:38 AM	⋮		

On this page, you can review the existing PersistentVolumes used by pods. These volumes allow data to exist throughout pod deletion and creation. You can also create new PersistentVolumes on this page. **Note:** PersistentVolumes should be created with a highly available file system. For this lab, we are using an NFS server but you could use iSCSI, GlusterFS, or a cloud-provider storage solution.



db2vol01

Overview

PersistentVolume details	
Type	Detail
Name	db2vol01
Type	Hostpath
Labels	-
Status	Bound
Capacity	5Gi
Access modes	RWO
Claim	default/labdb2-ibm-db2oltp-dev-data-stor
Reclaim policy	Retain
Creation time	Feb 6th 2018 at 9:59 AM

A PersistentVolume is claimed by a pod via a **PersistentVolumeClaim**. A PersistentVolumeClaim is an implementation of claim check pattern for storage. When a pod requires persistent storage it creates a claim against a PersistentVolume to hold that volume exclusively. When the pod finishes using a PersistentVolume (when a pod is deleted for example), the PersistentVolumeClaim is released and the PersistentVolume is made available. You can view the existing claims by clicking the **PersistentVolumeClaim** tab.



The screenshot shows the Kubernetes UI for managing PersistentVolumeClaims. A blue arrow points from the text above to the 'PersistentVolumeClaim' tab in the navigation bar. The table below lists five PersistentVolumeClaims:

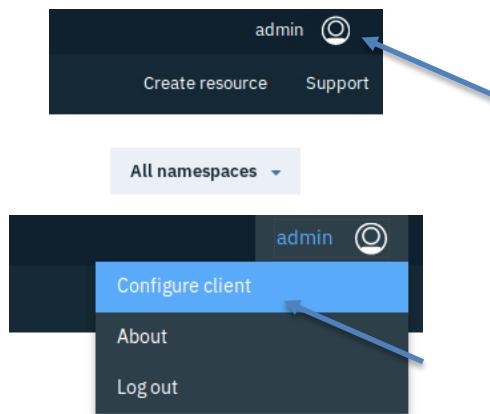
NAME	NAMESPACE	STATUS	PERSISTENTVOLUME	REQUESTS	ACCESS MODE	CREATION TIME	ACTION
elasticsearch-data-elasticsearch-data-0	kube-system	Bound	logging-es-pv-10.0.0.1	20Gi	RWO	Jan 4th 2018 at 9:57 AM	⋮
icp-ds-icp-ds-0	kube-system	Bound	cloudant-10.0.0.1	2Gi	RWO	Jan 4th 2018 at 9:53 AM	⋮
image-manager-image-manager-0	kube-system	Bound	image-manager-10.0.0.1	20Gi	RWO	Jan 4th 2018 at 9:57 AM	⋮
labdb2-ibm-db2oltp-dev-data-stor	default	Bound	db2vol01	5Gi	RWO	Feb 6th 2018 at 10:00 AM	⋮
mqlab-ibm-mq-data-mqlab-ibm-mq-0	default	Bound	mqvol1	2Gi	RWO	Feb 6th 2018 at 1:12 PM	⋮



4. Using the Kubernetes CLI

In this section, you will utilize the Kubernetes command line interface, `kubectl`, to view information about the IBM Cloud Private environment, and view information about StockTrader.

1. Before you can use the CLI you need to configure the command client with some connection information and credentials. You do this by clicking the username, **admin**, in the upper right corner of the web console and selecting **Configure Client**.



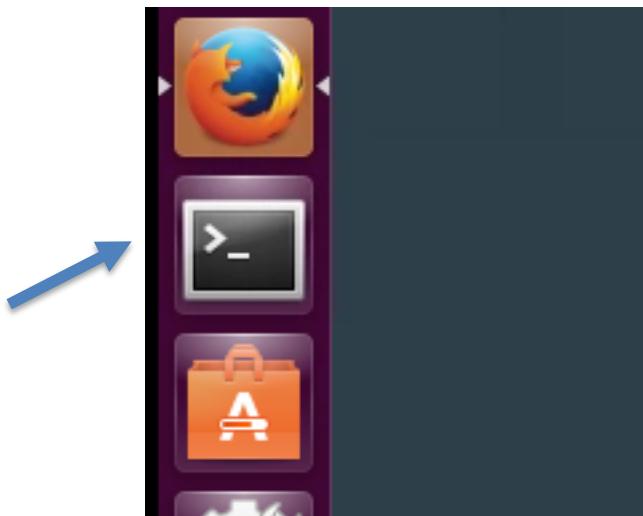
2. In the pop up box that appears, click on the blue icon to the right of the commands to copy them to the clipboard.

A screenshot of a modal dialog titled 'Configure kubectl'. The dialog provides instructions: 'Before you run commands in the kubectl command line interface for this cluster, you must configure the client.' It lists 'Prerequisites:' as 'Install the kubectl CLI: kubectl'. It then instructs the user to 'To configure the CLI, paste the displayed configuration commands into your terminal window and run them.:'. A code block shows the commands:

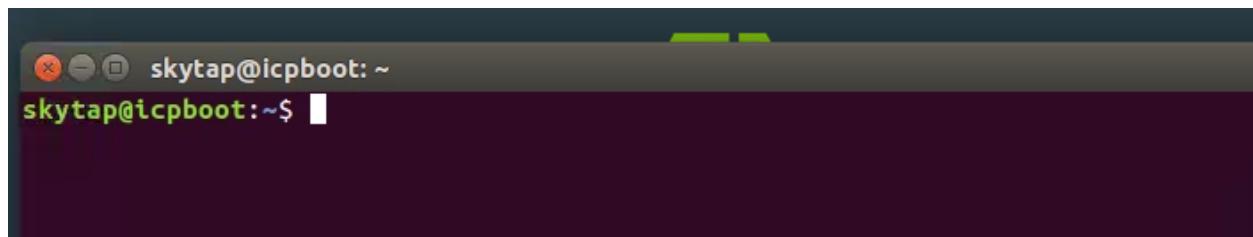
```
kubectl config set-cluster mycluster.icp --server=https://10.0.0.1:8001 --inse
kubectl config set-context mycluster.icp-context --cluster=mycluster.icp
kubectl config set-credentials admin --token=eyJhbGciOiJSUzI1NiJ9.eyJzdW
kubectl config set-context mycluster.icp-context --user=admin --namespace=
kubectl config use-context mycluster.icp-context
```

A blue arrow points from the text 'Copy' to the copy icon (a clipboard with a plus sign) located to the right of the command list.

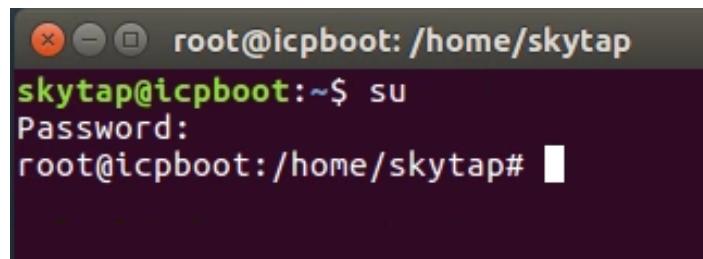
- Now you need to open a terminal by clicking the Terminal icon in the sidebar on the left side of the screen.



You should now see the terminal appear in front of Firefox:



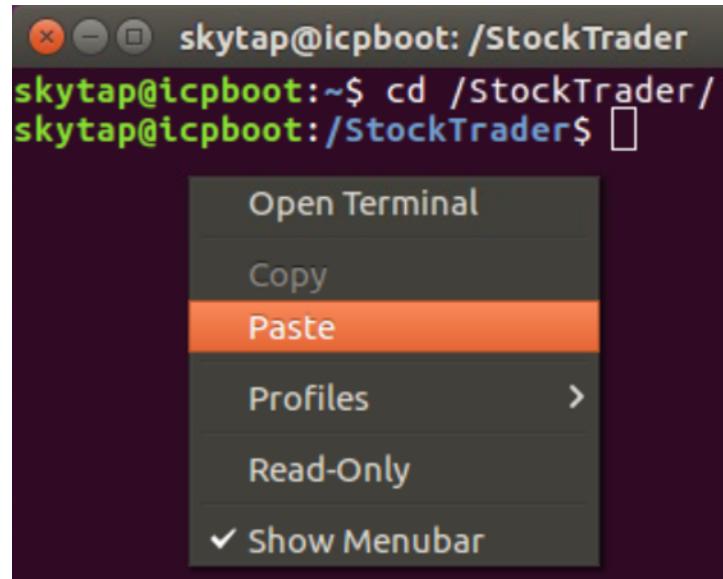
We need to switch to the root user and set up our command line. Type **su** and enter **A1rb0rn3** as the password.



Now enter the StockTrader directory by entering **cd /StockTrader**

- Right click in the terminal window and select **Paste**. Note that in this Skytap VM, choosing Paste from the right-click menu works better than keyboard shortcuts like Ctrl-V.





This configures `kubectl` to talk to our IBM Cloud Private instance. The last line in the console should read ‘Switched to context “mycluster.icp-context”’.

```
Switched to context "mycluster.icp-context".
skytap@icpboot:/StockTrader$ █
```

5. First, find all the deployments in IBM Cloud Private. A deployment is a high-level abstraction which manages pods. As mentioned earlier, deployment is equivalent to the Workload/Deployments in the IBM Cloud Private web interface. You’ll use deployments later in the lab. You can get the list of current deployments and their status by running `kubectl get deployments`

```
skytap@icpboot:/StockTrader$ kubectl get deployments
NAME          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
labdb2-ibm-db2oltp-dev   1         1         1           1          9d
loyalty-level      1         1         1           1          3d
messaging          1         1         1           1          8h
notification-twitter  1         1         1           1          7h
portfolio          1         1         1           1          3d
redislab2-redis     1         1         1           1          9d
stock-quote         1         1         1           1          3d
trader              1         1         1           1          3d
skytap@icpboot:/StockTrader$ █
```

You will recognize these deployments as the StockTrader application. If you want more information about these deployments, such as what containers and Docker images are associated with a given deployment you can run `kubectl get deployments -o wide`



(Remember, if you copy from this document, make sure you select the Skytap clipboard menu item  and paste the command so the VM adds that text to its clipboard)

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE	CONTAINERS	IMAGES	SELECTOR
labdb2-lbm-db2oltp-dev	1	1	1	1	10d	labdb2-lbm-db2oltp-dev	na.cumulusrepo.com/hclcp_dev/db2server_dcc:11.1.2.2b mycluster.icp:8500/default/loyalty-level:latest	app=labdb2-lbm-db2oltp-dev app=loyalty-level,solution=social
loyalty-level						loyalty-level		
stock-trader,version=v1								
messaging	1	1	1	1	1d	messaging	mycluster.icp:8500/default/messaging:latest	app=messaging,solution=stock-trader
k-trader								
notification-twittter	1	1	1	1	1d	notification-twittter	mycluster.icp:8500/default/notification-twittter:latest	app=notification,solution=stock-trader
stock-trader,version=twittter								
portfolio	1	1	1	1	4d	portfolio	mycluster.icp:8500/default/portfolio:latest	app=portfolio,solution=stock-trader
x-trader								
redisLab2-redis	1	1	1	1	10d	redisLab2-redis	bitnami/redis:4.0.7-r0 mycluster.icp:8500/default/stock-quote:latest	app=redisLab2-redis app=stock-quote,solution=stock-trader
stock-quote	1	1	1	1	4d	stock-quote		
stock-trader								
trader	1	1	1	1	4d	trader	mycluster.icp:8500/default/trader:latest	app=trader,solution=stock-trader
trader,version=v1								

- For even more information about a deployment you can run `kubectl describe deployment <deploymentName>`. Try it for the `stock-quote` deployment by entering `kubectl describe deployment stock-quote`

In the returned data, you should recognize some of the same data from IBM Cloud Private's Workloads > Deployments web interface.

```
skytap@icpboot:/StockTrader
skytap@icpboot:/StockTrader$ kubectl describe deployment stock-quote
Name:           stock-quote
Namespace:      default
CreationTimestamp: Mon, 12 Feb 2018 12:33:05 -0800
Labels:          app=stock-quote
                 solution=stock-trader
Annotations:    deployment.kubernetes.io/revision=1
Selector:       app=stock-quote,solution=stock-trader
Replicas:       1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:   RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 1 max unavailable, 1 max surge
Pod Template:
  Labels:  app=stock-quote
           solution=stock-trader
  Containers:
    stock-quote:
      Image:  mycluster.icp:8500/default/stock-quote:latest
      Ports:   9080/TCP, 9443/TCP
      Environment:
        REDIS_URL:      <set to the key 'url' in secret 'redis'>      Optional: false
        QUANDL_KEY:     <set to the key 'quandl-key' in secret 'redis'>  Optional: false
        JWT_AUDIENCE:   <set to the key 'audience' in secret 'jwt'>    Optional: false
        JWT_ISSUER:     <set to the key 'issuer' in secret 'jwt'>      Optional: false
      Mounts:          <none>
      Volumes:         <none>
  Conditions:
    Type      Status  Reason
    ----      ----   -----
    Available  True    MinimumReplicasAvailable
OldReplicaSets: <none>
NewReplicaSet:  stock-quote-d7d8c4cf5 (1/1 replicas created)
Events:         <none>
skytap@icpboot:/StockTrader$
```

- To get the list of pods in IBM Cloud Private type: `kubectl get pods`



NAME	READY	STATUS	RESTARTS	AGE
labdb2-ibm-db2oltp-dev-6dfcb56b8d-r6dg9	1/1	Running	1	9d
loyalty-level-789465954c-gcmkq	1/1	Running	1	3d
messaging-64cf9848ff-wvpg8	1/1	Running	0	9h
mqlab-ibm-mq-0	1/1	Running	1	9d
notification-twitter-79f4b9f465-h5g6x	1/1	Running	0	8h
portfolio-647fcc8fcfd-2l7pv	1/1	Running	1	3d
redislab2-redis-c874b98d9-5q6t6	1/1	Running	1	9d
stock-quote-d7d8c4cf5-5zdwn	1/1	Running	1	3d
trader-5c75688d9-92brf	1/1	Running	1	3d

You will notice that this list of pods is shorter than the list reported in the web interface. This is because `kubectl` filters out the Kubernetes system pods by default (you can see all the pods across all namespaces, like the web interface shows by default, by adding the `--all-namespaces` flag, as in `kubectl get pods --all-namespaces`). You will also recognize some of the same pods for StockTrader as from the IBM Cloud Private web interface. If you want the location where the pods are running you can run `kubectl get pods -o wide`

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
labdb2-ibm-db2oltp-dev-6dfcb56b8d-r6dg9	1/1	Running	1	9d	10.1.64.104	10.0.0.1
loyalty-level-789465954c-gcmkq	1/1	Running	1	3d	10.1.64.72	10.0.0.1
messaging-64cf9848ff-wvpg8	1/1	Running	0	9h	10.1.64.99	10.0.0.1
mqlab-ibm-mq-0	1/1	Running	1	9d	10.1.64.124	10.0.0.1
notification-twitter-79f4b9f465-h5g6x	1/1	Running	0	8h	10.1.64.111	10.0.0.1
portfolio-647fcc8fcfd-2l7pv	1/1	Running	1	3d	10.1.64.126	10.0.0.1
redislab2-redis-c874b98d9-5q6t6	1/1	Running	1	9d	10.1.64.67	10.0.0.1
stock-quote-d7d8c4cf5-5zdwn	1/1	Running	1	3d	10.1.64.113	10.0.0.1
trader-5c75688d9-92brf	1/1	Running	1	3d	10.1.64.115	10.0.0.1

- If you need more detail about a pod you can run `kubectl describe pod <podName>` Try it for the `stock-quote` pod by running the following command (note the generated suffix after `stock-quote` might be different in your environment):

```
kubectl describe pod stock-quote-d7d8c4cf5-5zdwn
```



```

skytap@icpboot:/StockTrader$ kubectl describe pod stock-quote-d7d8c4cf5-5zdw
Name:           stock-quote-d7d8c4cf5-5zdw
Namespace:      default
Node:          10.0.0.1/10.0.0.1
Start Time:    Mon, 12 Feb 2018 12:33:05 -0800
Labels:         app=stock-quote
                pod-template-hash=838470791
                solution=stock-trader
Annotations:   kubernetes.io/created-by={"kind":"SerializedReference","apiVersion":"v1","reference":{"kind":"ReplicaSet","namespace":"default","name":"stock-quote-d7d8c4cf5","uid":"edcf47cc-1033-11e8-9bc1-005056379c..."}},kubernetes.io/psp=default
Status:        Running
IP:            10.1.64.113
Controlled By: ReplicaSet/stock-quote-d7d8c4cf5
Containers:
  stock-quote:
    Container ID:  docker://234f662014b8e7bcd8a300f5900d9137cb19a45933d868f424f2ebb142c19371
    Image:         mycluster.icp:8500/default/stock-quote:latest
    Image ID:     docker-pullable://mycluster.icp:8500/default/stock-quote@sha256:cf3e59af45a065e7cb89a52ac6123bc3d00b7ce3c6e08c3ccfff1dc5aa304f92
    Ports:        9080/TCP, 9443/TCP
    State:        Running
      Started:   Thu, 15 Feb 2018 23:31:55 -0800
    Last State:  Terminated
      Reason:    Error
      Exit Code: 137
      Started:   Mon, 12 Feb 2018 12:33:07 -0800
      Finished:  Tue, 13 Feb 2018 14:07:18 -0800
    Ready:       True
    Restart Count: 1
    Environment:
      REDIS_URL: <set to the key 'url' in secret 'redis'>          Optional: false
      QUANDL_KEY: <set to the key 'quandl-key' in secret 'redis'>    Optional: false
      JWT_AUDIENCE: <set to the key 'audience' in secret 'jwt'>      Optional: false
      JWT_ISSUER:  <set to the key 'issuer' in secret 'jwt'>        Optional: false

```

This provides a lot of container and runtime data for this particular pod. This is useful when debugging problems with pod deployments. Some of this data should look familiar from the deployment information gathered earlier. This pod utilized that data when instantiating itself.

9. You can also gather the node information by running `kubectl get nodes` and `kubectl describe node <nodeName>`

```

skytap@icpboot:/StockTrader$ kubectl get nodes
NAME      STATUS    ROLES   AGE      VERSION
10.0.0.1  Ready     <none>  42d      v1.8.3+icp+ee
skytap@icpboot:/StockTrader$ █

```

The `kubectl describe node` command can be rather verbose so you must scroll up in the terminal to see everything. The data returned is very similar to the graphs you saw in IBM Cloud Private's Node page in the web interface.



```

skytap@icpboot:/StockTrader$ kubectl describe node 10.0.0.1
Name:           10.0.0.1
Roles:          <none>
Labels:         beta.kubernetes.io/arch=amd64
                beta.kubernetes.io/os=linux
                gpu/nvidia=NA
                kubernetes.io/hostname=10.0.0.1
                management=true
                proxy=true
                role=master
Annotations:   alpha.kubernetes.io/provided-node-ip=10.0.0.1
                node.alpha.kubernetes.io/ttl=0
                volumes.kubernetes.io/controller-managed-attach-detach=true
Taints:         <none>
CreationTimestamp: Thu, 04 Jan 2018 09:52:21 -0800
Conditions:
  Type      Status  LastHeartbeatTime          LastTransitionTime
  Reason    Message
  ----     -----
  OutOfDisk False   Fri, 16 Feb 2018 09:38:37 -0800  Thu, 04 Jan 2018 09:52:21
  -0800   KubeletHasSufficientDisk  kubelet has sufficient disk space available
  MemoryPressure False   Fri, 16 Feb 2018 09:38:37 -0800  Fri, 16 Feb 2018 06:58:43
  -0800   KubeletHasSufficientMemory kubelet has sufficient memory available
  DiskPressure False   Fri, 16 Feb 2018 09:38:37 -0800  Fri, 16 Feb 2018 06:58:43
  -0800   KubeletHasNoDiskPressure kubelet has no disk pressure
  Ready      True    Fri, 16 Feb 2018 09:38:37 -0800  Fri, 16 Feb 2018 06:58:43
  -0800   KubeletReady            kubelet is posting ready status. AppArmor enabled
Addresses:
  InternalIP: 10.0.0.1
  Hostname:   10.0.0.1
Capacity:
  cpu:        8
  memory:    16431908Ki
  pods:       110

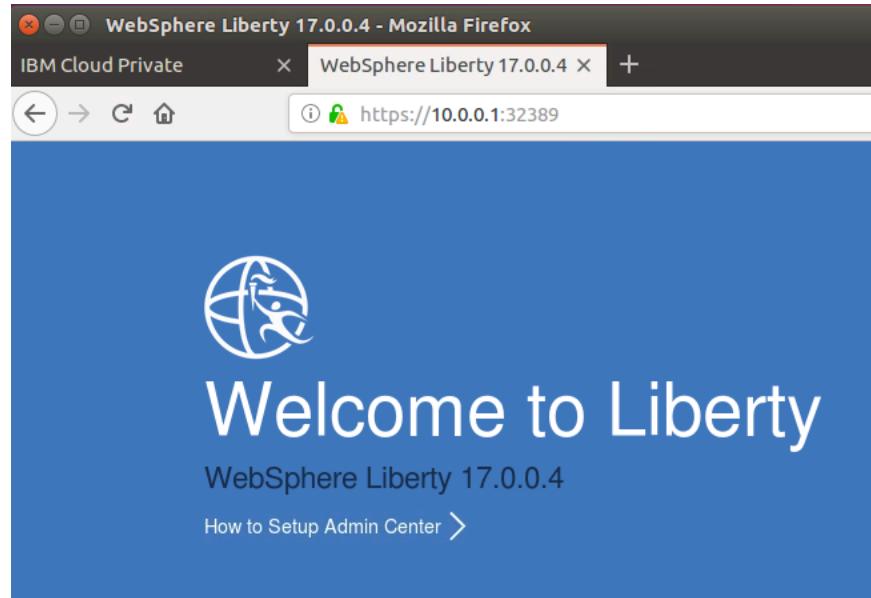
```

10. What if you've just deployed StockTrader to IBM Cloud Private and need to know which external port you should use when accessing StockTrader's web interface? You can determine the port by running `kubectl get services`

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.0.0.1	<none>	443/TCP	42d
labdb2-ibm-db2oltp-dev	NodePort	10.0.0.14	<none>	50000:30042/TCP,55000:32070/TCP	9d
loyalty-level-service	NodePort	10.0.0.95	<none>	9080:31181/TCP,9443:32143/TCP	3d
mqlab-ibm-mq	NodePort	10.0.0.205	<none>	1414:30016/TCP,9443:32343/TCP	9d
notification-service	NodePort	10.0.0.231	<none>	9080:30043/TCP,9443:32046/TCP	8h
portfolio-service	NodePort	10.0.0.186	<none>	9080:31066/TCP,9443:32709/TCP	3d
redislab2-redis	NodePort	10.0.0.151	<none>	6379:31803/TCP	9d
stock-quote-service	NodePort	10.0.0.169	<none>	9080:30302/TCP,9443:30826/TCP	3d
trader-service	NodePort	10.0.0.152	<none>	9080:32388/TCP,9443:32389/TCP	3d

We know that the *trader* microservice provides the user interface so we check the table for *trader-service*. Here you see that *trader-service*'s port 9443 (Liberty's default HTTPS port) is exposed to the outside world at port **32389**. If you go back to Firefox and enter <https://10.0.0.1:32389> you should see the "Welcome to Liberty" splash page.





If you add **/trader** to the end of the URL you should be redirected to the StockTrader user interface.

A screenshot of a Mozilla Firefox browser window titled "Stock Portfolio - Mozilla Firefox". The address bar shows "Stock Portfolio" and "10.0.0.1:31110/trader/summary". The main content area features a colorful background image of various financial instruments like stocks, bonds, and mutual funds. Below the image is a list of options:

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

A table displays portfolio data:

	Owner	Total	Loyalty Level
<input checked="" type="radio"/>	EM	\$5,688.93	Basic
<input type="radio"/>	EM1	\$2,158,753.68	Platinum



5. Edit and Deploy a New Version of a StockTrader Micro-Service

Earlier you saw how a tweet gets sent out when a portfolio's loyalty level changes. As a reminder, here's an example tweet, showing the message that the version of *notification-twitter* in GitHub (and on the lab VM) produces:



Update Code

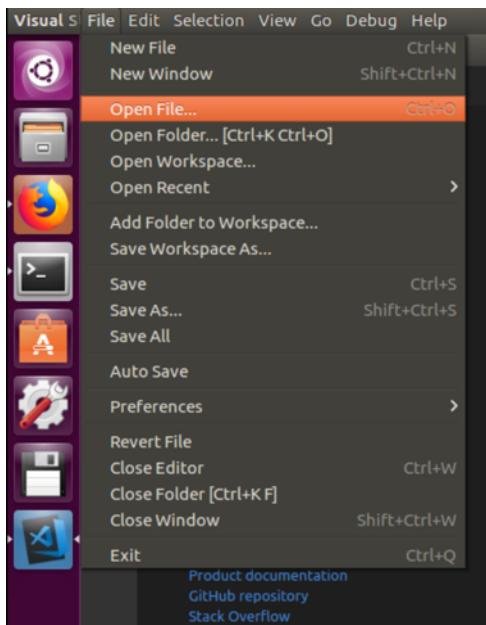
Let's go update that message, so we can see the process for how to build and deploy a new version of a microservice. First, open up the **VS Code** editor (<https://code.visualstudio.com>), which is pre-installed in the lab VM image, via the blue icon on the left palette:



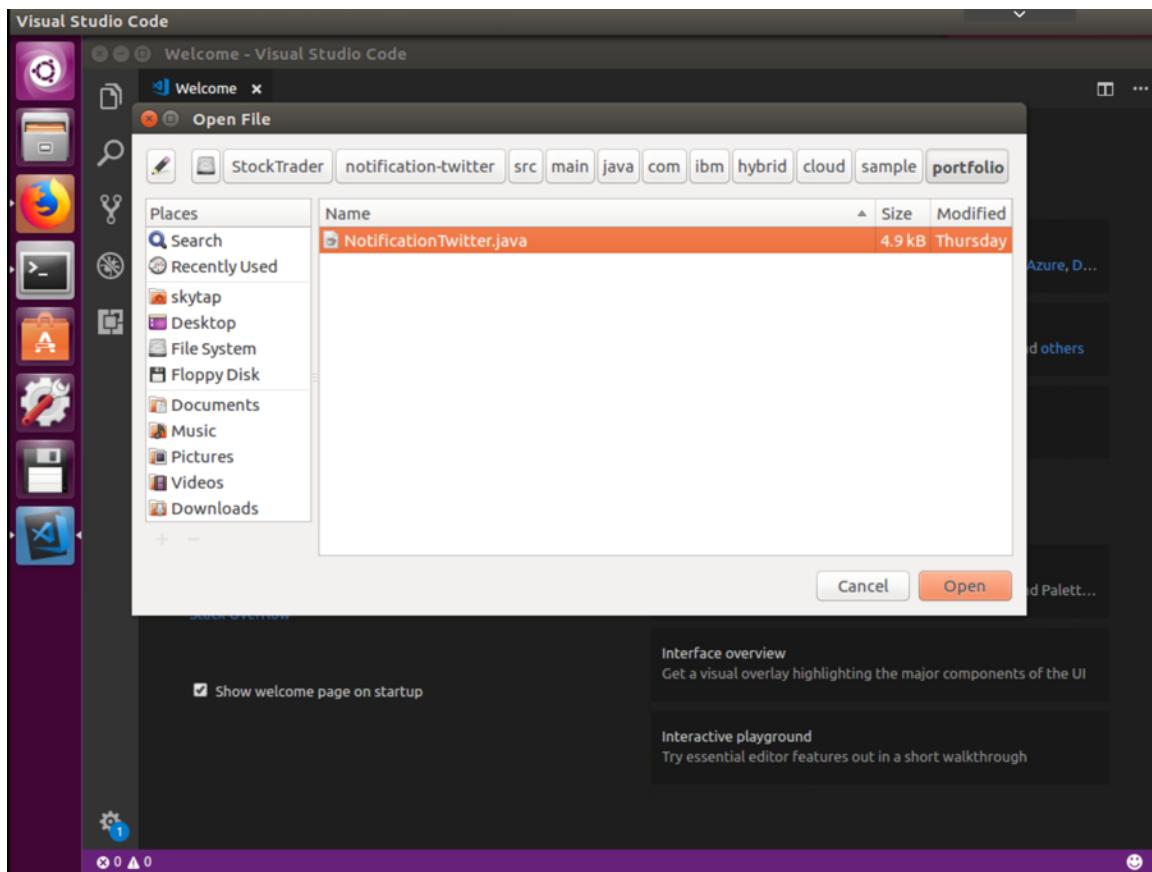
You could use another editor if you prefer, like **nano** or **vi**, but I recommend **VS Code** (with its Java language extension installed) since it understands Java and will provide code highlighting and instant reporting of syntax errors.

Choose **File->Open File...** from the menu bar at the top of the screen:





Navigate to /StockTrader/notification-twitter/src/main/java/com/ibm/hybrid/cloud/sample/portfolio/NotificationTwitter.java



Notice how the time stamp in the tweet is in the *UTC* (London) time zone? Since this conference takes place in the *US Pacific Standard Time* (PST) time zone, let's edit the code to send the tweet in PST instead. Go to line 106 (in the `initialize()` method) and let's make two changes. First, replace the "UTC" with "PST" in the `SimpleDateFormat` constructor. Then add a line right under that one which says:

```
format.setTimeZone(java.util.TimeZone.getTimeZone("PST"));
```

(you could use the more formal "America/Los_Angeles" in place of "PST" if you desire). The `initialize()` method should now look like this:

```
NotificationTwitter.java ●

92     private void initialize() {
93         logger.fine("Initializing Twitter API");
94
95         ConfigurationBuilder builder = new ConfigurationBuilder();
96         builder.setDebugEnabled(true);
97         builder.setOAuthConsumerKey(System.getenv("TWITTER_CONSUMER_KEY"));
98         builder.setOAuthConsumerSecret(System.getenv("TWITTER_CONSUMER_SECRET"));
99         builder.setOAuthAccessToken(System.getenv("TWITTER_ACCESS_TOKEN"));
100        builder.setOAuthAccessTokenSecret(System.getenv("TWITTER_ACCESS_TOKEN_SECRET"));
101
102        TwitterFactory factory = new TwitterFactory(builder.build());
103        twitter = factory.getInstance(); //initialize twitter4j
104
105        //Example: Monday, October 16, 2017 at 3:45 PM UTC
106        format = new SimpleDateFormat("EEEE, MMMM d, yyyy 'at' h:mm a 'PST'");
107        format.setTimeZone(java.util.TimeZone.getTimeZone("PST"));
108
109        logger.fine("Initialization completed successfully!");
110        initialized = true;
111    }
```

Let's also include the hashtag for the conference. The Think conference is known by #Think2018, so let's go add that to the tweet message, in line 120 (I split the line for readability). Then save the file. Here's what the new `tweet()` method looks like after that:



```
NotificationTwitter.java ●

113     /** Tweet a message to our @IBMStockTrader account.
114      * @throws TwitterException
115      */
116     private String tweet(String owner, String oldLoyalty, String loyalty) throws TwitterException {
117         if (!initialized) initialize();
118
119         Date now = new Date();
120         String message = "On "+format.format(now)+" , "+owner+" changed status from "+oldLoyalty+
121             " to "+loyalty+". #IBMStockTrader #Think2018";
122
123         logger.fine("Sending following tweet: "+message);
124         twitter.updateStatus(message);
125
126         logger.info("Message tweeted successfully!");
127         return message;
128     }
```

Build the project

Now return to the Terminal window (where “su” has been run), switch to the /StockTrader/notification-twitter directory, and run “mvn package”.

```
skytap@icpboot:~$ cd /StockTrader/notification-twitter/
skytap@icpboot:/StockTrader/notification-twitter$ su
Password:
root@icpboot:/StockTrader/notification-twitter# mvn package
```

That command produces several screens of output. Once it completes, you should see output like the following. If so, then run “docker build -t notification-twitter .” (note the trailing dot, meaning current directory), as shown below:

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 43.497 s
[INFO] Finished at: 2018-02-18T04:53:48-08:00
[INFO] Final Memory: 16M/144M
[INFO] -----
root@icpboot:/StockTrader/notification-twitter# docker build -t notification-twitter .
```

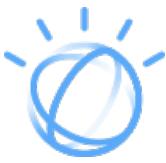
This make take a couple of minutes. When it completes, run “docker tag notification-twitter:latest mycluster.icp:8500/default/notification-twitter:latest”, followed by “docker push mycluster.icp:8500/default/notification-twitter:latest”.



```
root@icpboot:/StockTrader/notification-twitter
Successfully tagged notification-twitter:latest
root@icpboot:/StockTrader/notification-twitter# docker tag notification-twitter:latest
mycluster.icp:8500/default/notification-twitter:latest
root@icpboot:/StockTrader/notification-twitter# docker push mycluster.icp:8500/default/
notification-twitter:latest
The push refers to repository [mycluster.icp:8500/default/notification-twitter]
311480de49d0: Pushed
05ab2361008b: Pushed
7a42ed25cdab: Pushed
1fa497110f5d: Layer already exists
70ff20a97f51: Layer already exists
a0c9e4c8bfb4: Layer already exists
0165e4961a77: Layer already exists
a049d653e9fe: Layer already exists
49cc0168f595: Layer already exists
58860ebfe087: Layer already exists
2cdf1100a9ba: Layer already exists
babe8441459b: Layer already exists
c302aac9a989: Layer already exists
eb2b8fa110d3: Layer already exists
6f4ce6b88849: Layer already exists
92914665e7f6: Layer already exists
c98ef191df4b: Layer already exists
9c7183e0ea88: Layer already exists
ff986b10a018: Layer already exists
latest: digest: sha256:b6a9419935c34fe5cc4aae1c2dde4fe33daf2acf7723c7e18441e9dc6c7abaf0
size: 4288
root@icpboot:/StockTrader/notification-twitter# kubectl apply -f manifests/deploy.yaml
```

Deploy to IBM Cloud Private

We now have the updated microservice in the Docker image registry. So we're ready to deploy the updates to IBM Cloud Private. First, let's undeploy the old version, via "kubectl delete -f manifests/deploy.yaml". Then do a "kubectl get pods" to confirm it's gone. Then you can do a "kubectl create -f manifests/deploy.yaml", and another "kubectl get pods" to confirm the new version is deployed and running.



```

root@icpboot:/StockTrader/notification-twitter
root@icpboot:/StockTrader/notification-twitter# kubectl delete -f manifests/deploy.yaml

deployment "notification-twitter" deleted
service "notification-service" deleted
ingress "notification-ingress" deleted
root@icpboot:/StockTrader/notification-twitter# kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
labdb2-ibm-db2oltp-dev-6dfcb56b8d-r6dg9  1/1     Running   2          11d
loyalty-level-789465954c-gcmkq            1/1     Running   2          5d
messaging-64cf9848ff-wvpg8               1/1     Running   1          2d
mqlab-ibm-mq-0                          1/1     Running   2          11d
portfolio-647fcc8fcfd-sqxtm             1/1     Running   1          14h
redislab3-redis-7887cb7c46-qqqkp        1/1     Running   7          1d
stock-quote-d7d8c4cf5-vw2cl              1/1     Running   1          1d
trader-5c75688d9-8mjpd                  1/1     Running   1          14h
root@icpboot:/StockTrader/notification-twitter# kubectl create -f manifests/deploy.yaml
deployment "notification-twitter" created
service "notification-service" created
ingress "notification-ingress" created
root@icpboot:/StockTrader/notification-twitter# kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
labdb2-ibm-db2oltp-dev-6dfcb56b8d-r6dg9  1/1     Running   2          11d
loyalty-level-789465954c-gcmkq            1/1     Running   2          5d
messaging-64cf9848ff-wvpg8               1/1     Running   1          2d
mqlab-ibm-mq-0                          1/1     Running   2          11d
notification-twitter-79f4b9f465-nhzcl      1/1     Running   0          13s
portfolio-647fcc8fcfd-sqxtm             1/1     Running   1          14h
redislab3-redis-7887cb7c46-qqqkp        1/1     Running   7          1d
stock-quote-d7d8c4cf5-vw2cl              1/1     Running   1          1d
trader-5c75688d9-8mjpd                  1/1     Running   1          14h
root@icpboot:/StockTrader/notification-twitter#

```

Run the code

Now go add some stock to cause a change in loyalty level, and then take a look at the new tweet.



IBM Stock Trader @IBMStockTrader · 7m

On Tuesday, March 6, 2018 at 6:20 AM PST, Think2018 changed status from Basic to Bronze. #IBMStockTrader #Think2018



Congratulations, you've now seen the full edit->compile->deploy->run cycle for a microservice in IBM Cloud Private. But before we wrap up this section, let's do one more thing. Let's deploy a tool that will show the various microservice interactions that led from you clicking in the browser to the tweet getting sent, which is a fairly long call path.



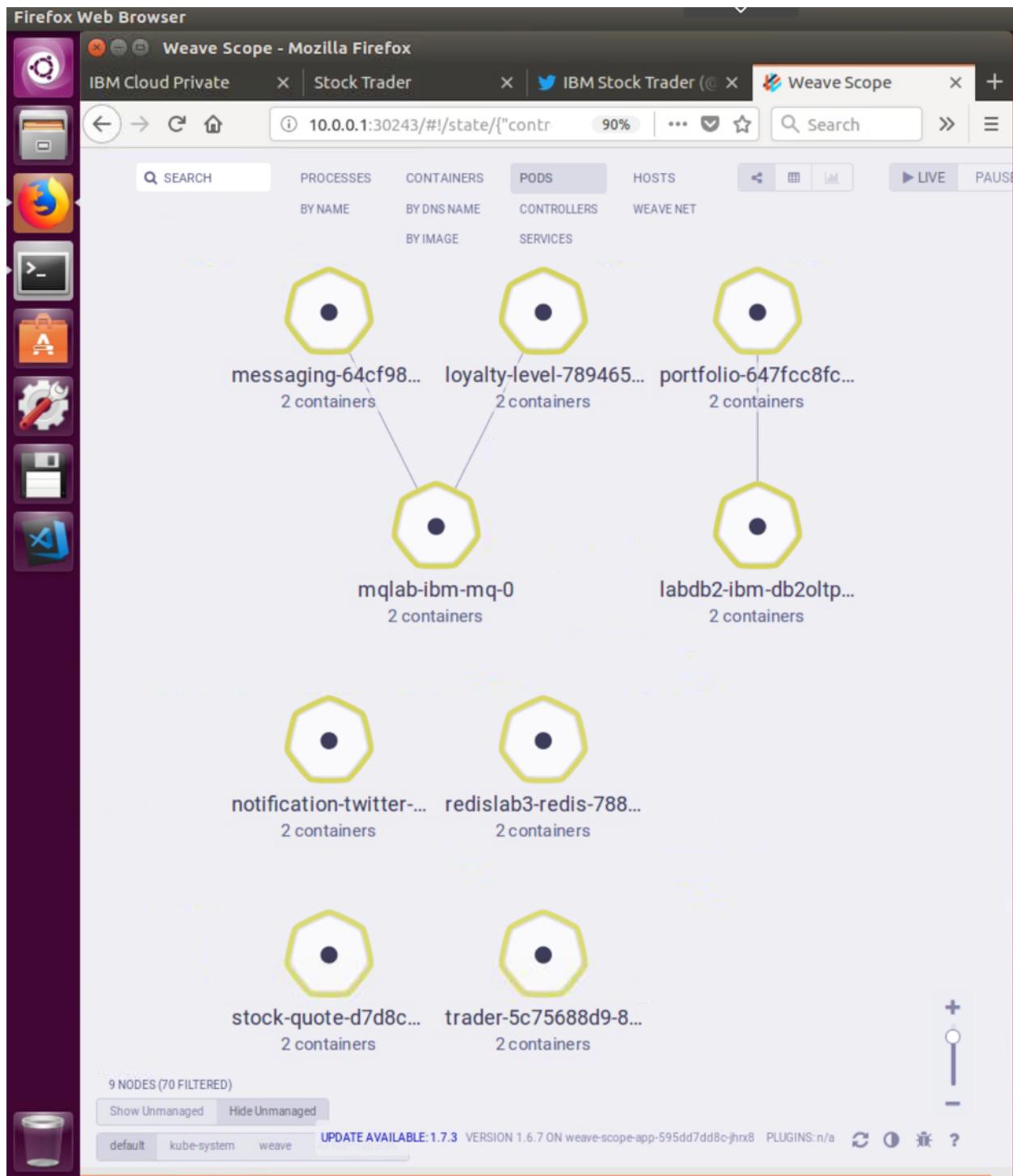
Visualize the interactions

WeaveWorks has a cool tool (<https://www.weave.works/docs/net/latest/kubernetes/kube-addon/>) that plugs in to Kubernetes, that among other things provides dynamic discovery and visualization of interaction between the pods. For convenience, the lab VM has its deployment yaml already downloaded and in the `/StockTrader` directory. Just go there and run `"kubectl create -f weave.yaml"`. This will create a new "weave" namespace (rather than dumping everything in the "default" namespace, like where StockTrader is) and creates several resources in that namespace. We'll have to use the `"-n"` flag to kubectl to tell it to target that new namespace. For example, you can run `"kubectl get pods -n weave"` to see the new pods it made available, and `"kubectl get services -n weave"` to see its service, which will tell you the node port that you need to enter in the browser to access its UI.

```
skytap@icpboot:/StockTrader$ ls
loyalty-level notification-twitter stock-quote weave.yaml
messaging portfolio trader
skytap@icpboot:/StockTrader$ kubectl create -f weave.yaml
namespace "weave" created
serviceaccount "weave-scope" created
clusterrole "weave-scope" created
clusterrolebinding "weave-scope" created
deployment "weave-scope-app" created
service "weave-scope-app" created
daemonset "weave-scope-agent" created
skytap@icpboot:/StockTrader$ kubectl get pods -n weave
NAME                  READY   STATUS    RESTARTS   AGE
weave-scope-agent-zrzlb   1/1    Running   0          30s
weave-scope-app-595dd7dd8c-jhrx8   1/1    Running   0          30s
skytap@icpboot:/StockTrader$ kubectl get services -n weave
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
weave-scope-app   NodePort    10.0.0.186    <none>        80:30243/TCP   40s
```

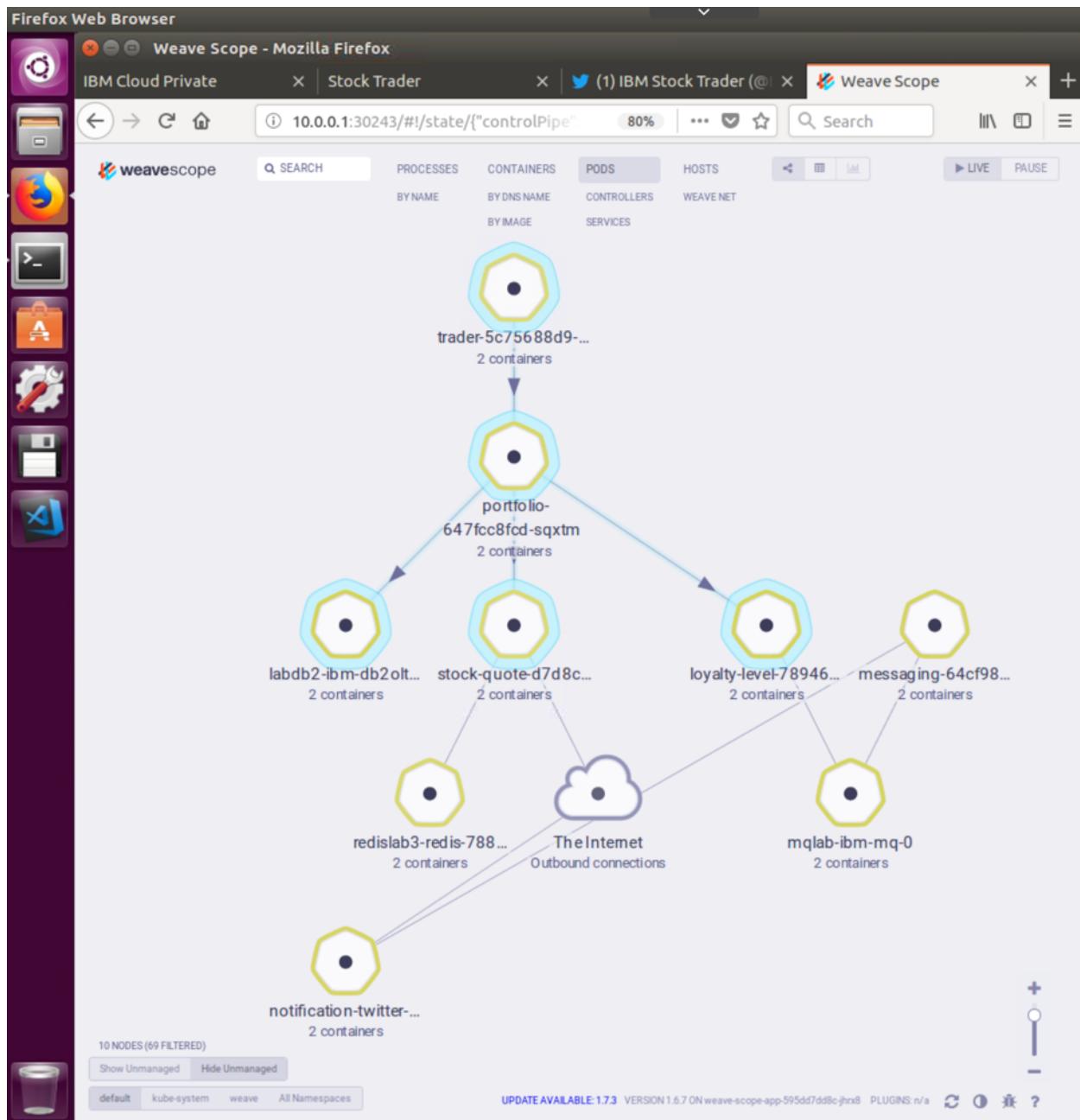
As you can see in this example, it got assigned a node port of 30243 (you may get a different value when you run this – if so, use that instead). So let's go back to Firefox and hit <http://10.0.0.1:30243> and give it a try. At first it shows way too much, so let's filter it down to just what we want, by clicking **Pods** in the top middle, and clicking on the **default** namespace at the bottom left:



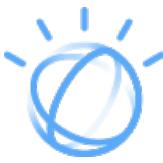


So we can see our pods, but with the app “at rest”, the only connections it could discover were the ones to DB2 and MQ (the non-http connections). So now let’s go use the Stock Trader app, making sure to cause a change in loyalty level. Then quickly return to this tab, since some of the http links between pods disappear after only a few seconds. If you hover over a pod, it will highlight and the links to and from it become directional arrows, such as when I hovered over the portfolio pod.





So other than being oriented top down rather than left to right, this dynamically discovered diagram (*note: yours may layout differently, but should have the same connections between nodes*) does exactly match my PowerPoint slide shown at the very beginning of the lab, showing who calls whom, who uses the prereqs like DB2, MQ and Redis, and who reaches out to services on the internet (note the link from messaging to notification-twitter only appears to go through the internet – really, that's just a sloppy layout thing). Plugins to Kubernetes, like this Weave tool, can help you view and manage things at an application level, rather than dealing with machines, memory, networks, etc.



6. Scaling StockTrader

Scaling deployments in IBM Cloud Private can occur in a few different places: the IBM Cloud Private web interface, the `kubectl` command line, or the Kubernetes management console (not shown in this lab). Within those places there are two different methods of scaling up and down. First, you can directly tell IBM Cloud Private and Kubernetes to scale up or down the deployment. Alternatively, if you find that you always have to change the number of replicas for a given service you can provide an updated deployment YAML file with a new `replicas` value to IBM Cloud Private and Kubernetes. This allows you to version control your configuration in an external source code repository and track changes which occur over time. We will focus on scaling directly via the command line in this lab.

1. Now let's scale up the number of instances for the *trader* microservice. Back in the terminal enter the command `kubectl get deployments` again.

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
labdb2-ibm-db2oltp-dev	1	1	1	1	11d
loyalty-level	1	1	1	1	5d
messaging	1	1	1	1	2d
notification-twitter	1	1	1	1	1d
portfolio	1	1	1	1	9h
redislab3-redis	1	1	1	1	1d
stock-quote	1	1	1	1	5d
trader	1	1	1	1	9h

Notice that the *trader* service currently expects and has a single container supporting it. Let's try scaling up *trader* by running the command

```
kubectl scale --replicas=3 deployment trader
```

```
skytap@icpboot:/StockTrader$ kubectl scale --replicas=3 deployment trader
deployment "trader" scaled
```

If you re-run `kubectl get deployments` you should now see there are three containers as part of the *trader* deployment.

```
skytap@icpboot:/StockTrader$ kubectl get deployments
NAME            DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
labdb2-ibm-db2oltp-dev   1         1         1         1         11d
loyalty-level      1         1         1         1         5d
messaging          1         1         1         1         2d
notification-twitter 1         1         1         1         1d
portfolio          1         1         1         1         9h
redislab3-redis    1         1         1         1         1d
stock-quote         1         1         1         1         5d
trader              3         3         3         3         9h
```



Also, if you run `kubectl get pods` you should now see three separate *trader* pods.

NAME	READY	STATUS	RESTARTS	AGE
labdb2-ibm-db2oltp-dev-6dfcb56b8d-r6dg9	1/1	Running	1	11d
loyalty-level-789465954c-gcmkq	1/1	Running	1	5d
messaging-64cf9848ff-wvpg8	1/1	Running	0	2d
mqlab-ibm-mq-0	1/1	Running	1	11d
notification-twitter-79f4b9f465-h5g6x	1/1	Running	0	1d
portfolio-647fcc8fcfd-sqxtm	1/1	Running	0	9h
redislab3-redis-7887cb7c46-qqqkp	1/1	Running	6	1d
stock-quote-d7d8c4cf5-vw2cl	1/1	Running	0	1d
trader-5c75688d9-8mjpd	1/1	Running	0	9h
trader-5c75688d9-dpvqd	1/1	Running	0	3m
trader-5c75688d9-tbnx9	1/1	Running	0	3m

2. To scale down the number of trader pods you can run

```
kubectl scale --replicas=1 deployment/trader
```

```
skytap@icpboot:/StockTrader$ kubectl scale --replicas=1 deployment trader
deployment "trader" scaled
```

And if you re-run `kubectl get deployments` and `kubectl get pods` you see the new values for *trader*.

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
labdb2-ibm-db2oltp-dev	1	1	1	1	11d
loyalty-level	1	1	1	1	5d
messaging	1	1	1	1	2d
notification-twitter	1	1	1	1	1d
portfolio	1	1	1	1	9h
redislab3-redis	1	1	1	1	1d
stock-quote	1	1	1	1	5d
trader	1	1	1	1	9h

NAME	READY	STATUS	RESTARTS	AGE
labdb2-ibm-db2oltp-dev-6dfcb56b8d-r6dg9	1/1	Running	1	11d
loyalty-level-789465954c-gcmkq	1/1	Running	1	5d
messaging-64cf9848ff-wvpg8	1/1	Running	0	2d
mqlab-ibm-mq-0	1/1	Running	1	11d
notification-twitter-79f4b9f465-h5g6x	1/1	Running	0	1d
portfolio-647fcc8fcfd-sqxtm	1/1	Running	0	9h
redislab3-redis-7887cb7c46-qqqkp	1/1	Running	6	1d
stock-quote-d7d8c4cf5-vw2cl	1/1	Running	0	1d
trader-5c75688d9-8mjpd	1/1	Running	0	9h



3. You can also scale up and down by editing the deployment YAML file for a given deployment and sending the updated value to IBM Cloud Private. To scale up the *stock-quote* service, in the terminal enter `nano stock-quote/manifests/deploy.yaml`

```
skytap@icpboot:/StockTrader$ nano stock-quote/manifests/deploy.yaml
```

In the editor look for the line `replicas: 1` under the `spec` header.

```
#Deploy the pod
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: stock-quote
# namespace: stock-trader
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: stock-quote
        solution: stock-trader
```

Change the `1` to `3` then save and exit the file by pressing and releasing `Ctrl+O`, then press `Enter`. Finally press `Ctrl+X` to exit the editor. Note you must have done the “`su`” command described earlier, or you won’t have authority to save the file.

```
#Deploy the pod
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: stock-quote
# namespace: stock-trader
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: stock-quote
        solution: stock-trader
```



4. Now scale out stock quote by typing

```
kubectl apply -f stock-quote/manifests/deploy.yaml --record
```

```
skytap@icpboot:/StockTrader$ kubectl apply -f stock-quote/manifests/deploy.yaml --record
Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
deployment "stock-quote" configured
Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
service "stock-quote-service" configured
Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
ingress "stock-quote-ingress" configured
```

You can ignore the warnings. Any time you take an action against a deployment that action is saved as part of the deployment's history. The `--record` flag saves the triggering command into that history for future reference.

If you run `kubectl get deployments` you'll see that stock-quote has scaled up to three.

```
skytap@icpboot:/StockTrader$ kubectl get deployments
NAME        DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
labdb2-ibm-db2oltp-dev   1         1         1           1          11d
loyalty-level      1         1         1           1          5d
messaging          1         1         1           1          2d
notification-twitter 1         1         1           1          2d
portfolio          1         1         1           1          10h
redislab3-redis    1         1         1           1          1d
stock-quote         3         3         3           3          5d
trader              1         1         1           1          10h
```

5. Now let's review recorded history. Run the command

```
kubectl rollout history deployment/stock-quote
```

```
skytap@icpboot:/StockTrader$ kubectl rollout history deployment/stock-quote
deployments "stock-quote"
REVISION  CHANGE-CAUSE
1          kubectl apply --filename=stock-quote/manifests/deploy.yaml --record=true
```

Notice there is only one change to this deployment, the scale out we just performed. If we performed another scale out we could tell IBM Cloud Private to revert the deployment to an earlier state in case an operation fails. This roll back is controlled by the `kubectl rollout undo` command.



Of course, we can also see the result of the scaling in the ICP UI, at *Deployments->stock-quote*:

The screenshot shows the IBM Cloud Private console interface in Mozilla Firefox. The URL is <https://10.0.0.1:8443/console/workloads/deployments/stock-quote>. The sidebar on the left has icons for Docs, IBM Cloud Private, Create resource, and Support. The main area shows the 'stock-quote' deployment under 'Deployments'. The 'Overview' tab is selected, showing a 'ReplicaSets' table and a 'Pods' table.

Type	desired	Current
stock-quote-d7d8c4cf5 Feb 12th 2018 at 12:33 PM	3	3

NAME	NAMESPACE	STATUS	HOST IP	POD IP	READY	START TIME	ACTION
stock-quote-d7d8c4cf5-7qdxf	default	Running	10.0.0.1	10.1.64.113	1/1	Mar 6th 2018 at 6:44 AM	⋮
stock-quote-d7d8c4cf5-vv2cl	default	Running	10.0.0.1	10.1.64.103	1/1	Feb 16th 2018 at 12:22 PM	⋮
stock-quote-d7d8c4cf5-wxphm	default	Running	10.0.0.1	10.1.64.107	1/1	Mar 6th 2018 at 6:44 AM	⋮

If you want to return to one replica, repeat Steps 3 and 4 above replacing the replica value of 3 with **1**.

The deploy.yaml file allows you to version control your deployment configuration. You could make this part of a DevOps pipeline so any changes, once proven correct through a pipeline deployment, would eventually reach production. It lets you treat your deployments as another governable artifact.

You've now seen how to adjust the scaling of a microservice in a Kubernetes environment such as IBM Cloud Private. For high availability reasons, in production environments, it is recommended to have at least two replicas running of each microservice, so that if one goes down for any reason, the other can continue processing requests until Kubernetes automatically restarts the failed pod (which it does quite quickly, once the pod's process (such as a Liberty JVM) has exited/abended, or a health check has failed). You can also use this scaling feature to increase the number of concurrent requests that your microservice can handle before slowing down to an unacceptable level of performance.



7. Download, Build, and Deploy a Node.js microservice

So far, you have worked with all Java-based microservices. But since IBM Cloud Private is polyglot, you can deploy microservices written in any language to run in its Kubernetes-based environment. The user interface microservice, named *tradr*, is a deliberately simple sample implemented using old-fashioned Java servlets. It is pure server-side rendering; you don't even need JavaScript enabled in your browser to run it. But nowadays, people expect more professional looking web-based user interfaces, so we have an alternate, more sophisticated one written in Node.js and using the Vue.js (<https://vuejs.org>) UI framework. Let's go pull that from GitHub and deploy it to our environment and use it.

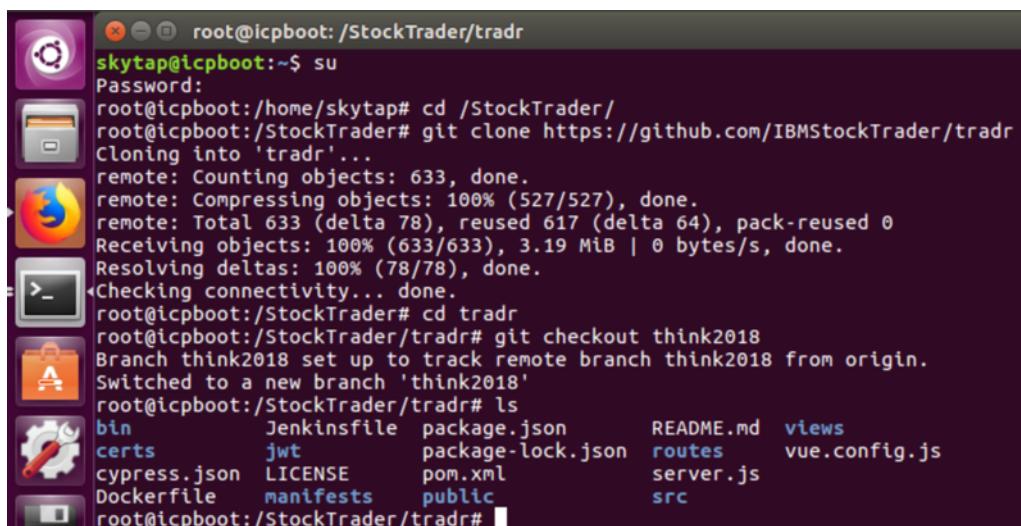
Download code from GitHub

This alternate client is not preloaded on your VM image. It is called *tradr* (note no "e" in the name), and we'll need to clone its repository from GitHub. Go to your Terminal window, make sure you've done an "su" so you have full authority, return to the /StockTrader directory, and run the following command:

```
git clone https://github.com/IBMStockTrader/tradr
```

This will make a new *tradr* directory. Change into that directory via "cd *tradr*". Note that, due to an Ingress issue in the old build of IBM Cloud Private that we are using in this lab (a newer 2.1.0.2 version should have been released just before the conference), we need to grab a workaround to make this client work with a Node Port instead. We can get that workaround by running the following command:

```
git checkout think2018
```



The screenshot shows a terminal window with the following session:

```
root@icpboot:/StockTrader/tradr
skytap@icpboot:~$ su
Password:
root@icpboot:/home/skytap# cd /StockTrader/
root@icpboot:/StockTrader# git clone https://github.com/IBMStockTrader/tradr
Cloning into 'tradr'...
remote: Counting objects: 633, done.
remote: Compressing objects: 100% (527/527), done.
remote: Total 633 (delta 78), reused 617 (delta 64), pack-reused 0
Receiving objects: 100% (633/633), 3.19 MiB | 0 bytes/s, done.
Resolving deltas: 100% (78/78), done.
Checking connectivity... done.
root@icpboot:/StockTrader# cd tradr
root@icpboot:/StockTrader/tradr# git checkout think2018
Branch think2018 set up to track remote branch think2018 from origin.
Switched to a new branch 'think2018'
root@icpboot:/StockTrader/tradr# ls
bin           Jenkinsfile  package.json      README.md  views
certs         jwt          package-lock.json  routes     vue.config.js
cypress.json  LICENSE      pom.xml          server.js
Dockerfile    manifests   public           src
root@icpboot:/StockTrader/tradr#
```



Build the project

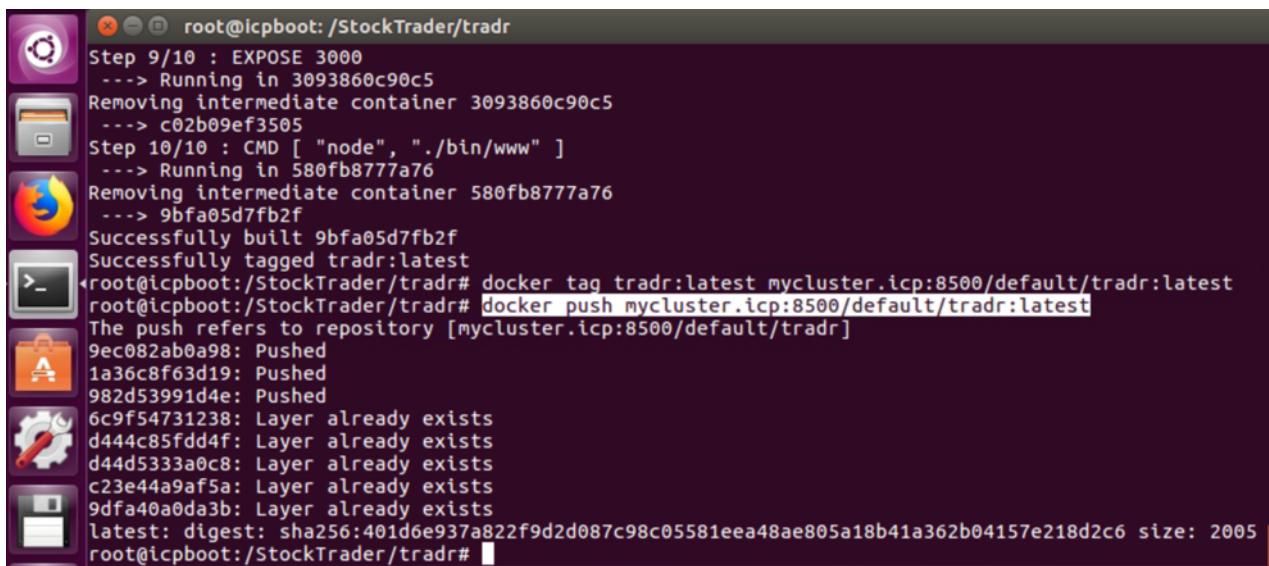
Unlike the Java-based project we worked with earlier, there is no need to use Maven to compile and package anything with this Node.js-based project. So we can jump straight to building its Docker container, via the following command (note the trailing dot, which means “current directory”):

```
docker build -t tradr .
```

Note that the build will likely take several minutes, as it has to download and unzip a lot into the container. If the build is taking a long time, you may wish to skip ahead to section 8, then return here once it completes. Next, tag and push this Docker image, via the following commands:

```
docker tag tradr:latest mycluster.icp:8500/default/tradr:latest
```

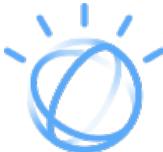
```
docker push mycluster.icp:8500/default/tradr:latest
```

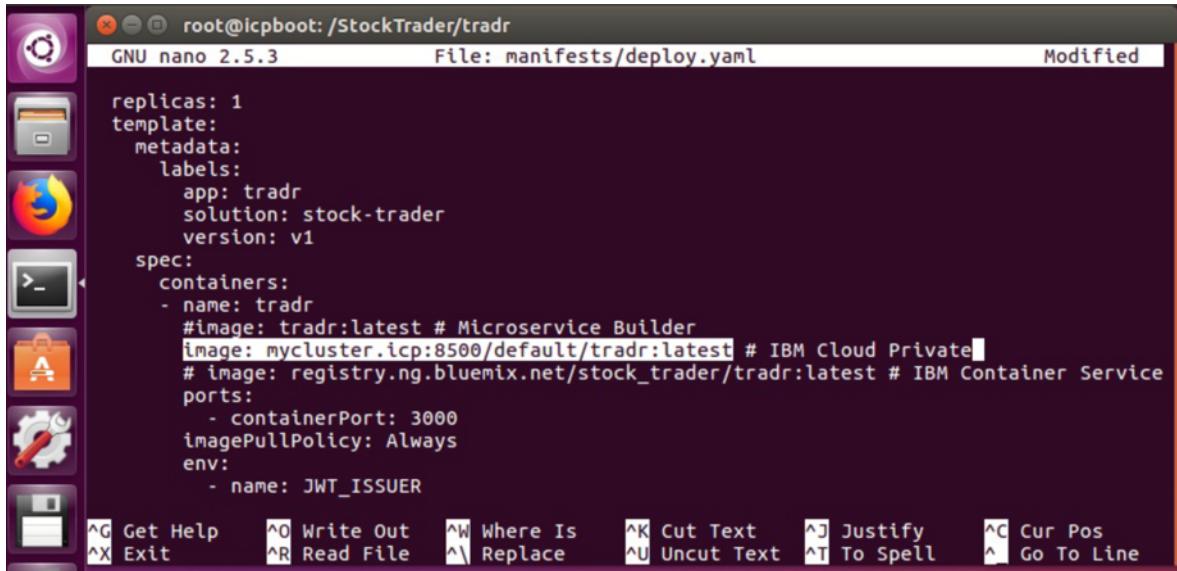


```
root@icpboot:/StockTrader/tradr
Step 9/10 : EXPOSE 3000
--> Running in 3093860c90c5
Removing intermediate container 3093860c90c5
--> c02b09ef3505
Step 10/10 : CMD [ "node", "./bin/www" ]
--> Running in 580fb8777a76
Removing intermediate container 580fb8777a76
--> 9bfa05d7fb2f
Successfully built 9bfa05d7fb2f
Successfully tagged tradr:latest
root@icpboot:/StockTrader/tradr# docker tag tradr:latest mycluster.icp:8500/default/tradr:latest
root@icpboot:/StockTrader/tradr# docker push mycluster.icp:8500/default/tradr:latest
The push refers to repository [mycluster.icp:8500/default/tradr]
9ec082ab0a98: Pushed
1a36c8f63d19: Pushed
982d53991d4e: Pushed
6c9f54731238: Layer already exists
d444c85fdd4f: Layer already exists
d44d5333a0c8: Layer already exists
c23e44a9af5a: Layer already exists
9dfa40a0da3b: Layer already exists
latest: digest: sha256:401d6e937a822f9d2d087c98c05581eea48ae805a18b41a362b04157e218d2c6 size: 2005
root@icpboot:/StockTrader/tradr#
```

Deploy to IBM Cloud Private

You will need to edit the `deploy.yaml`, in the `manifests` directory, to specify an image tag to match what you just pushed into the docker image registry. That is, it needs to be set to `mycluster.icp:8500/default/tradr:latest` (you’ll need to comment out the line setting it to just `tradr:latest` and uncomment the line for IBM Cloud Private). Run the command `nano manifests/deploy.yaml`, and scroll down to the image tag and set it as shown below:





```

root@icpboot: /StockTrader/tradr
GNU nano 2.5.3          File: manifests/deploy.yaml      Modified

replicas: 1
template:
  metadata:
    labels:
      app: tradr
      solution: stock-trader
      version: v1
  spec:
    containers:
      - name: tradr
        #image: tradr:latest # Microservice Builder
        image: mycluster.icp:8500/default/tradr:latest # IBM Cloud Private
        # image: registry.ng.bluemix.net/stock_trader/tradr:latest # IBM Container Service
        ports:
          - containerPort: 3000
        imagePullPolicy: Always
    env:
      - name: JWT_ISSUER

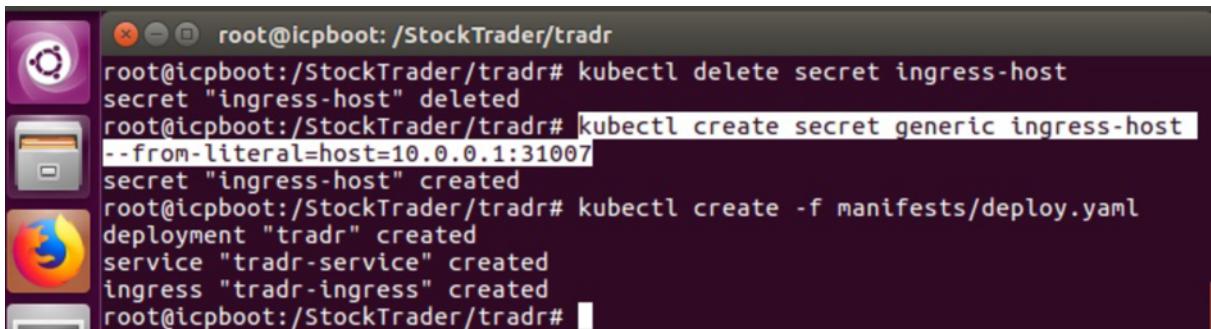
```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
 ^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line

Then press Ctrl-X to exit, hit “y” to save, and press Enter to accept the original file name. Of course, if you prefer using a different editor, such as **VS Code**, you are welcome to do so. Once you have saved the deploy.yaml file, we’ll need to do one more step before feeding this deploy.yaml to IBM Cloud Private.

Due to the Ingress bug mentioned earlier, we need to change the *ingress-host* secret to point to the Node Port for the *tradr* service, which the deploy.yaml defines to be port 31007. So first, let’s delete any existing *ingress-host* secret, via “`kubectl delete secret ingress-host`”. Then we will recreate it, passing `10.0.0.1:31007` as the value for the “host” field. Run the command “`kubectl create secret generic ingress-host --from-literal=host=10.0.0.1:31007`”.

With the updated *ingress-host* secret in place, we can now feed the deploy.yaml to IBM Cloud Private, via “`kubectl create -f manifests/deploy.yaml`”, as shown below:



```

root@icpboot:/StockTrader/tradr# kubectl delete secret ingress-host
secret "ingress-host" deleted
root@icpboot:/StockTrader/tradr# kubectl create secret generic ingress-host
--from-literal=host=10.0.0.1:31007
secret "ingress-host" created
root@icpboot:/StockTrader/tradr# kubectl create -f manifests/deploy.yaml
deployment "tradr" created
service "tradr-service" created
ingress "tradr-ingress" created
root@icpboot:/StockTrader/tradr#

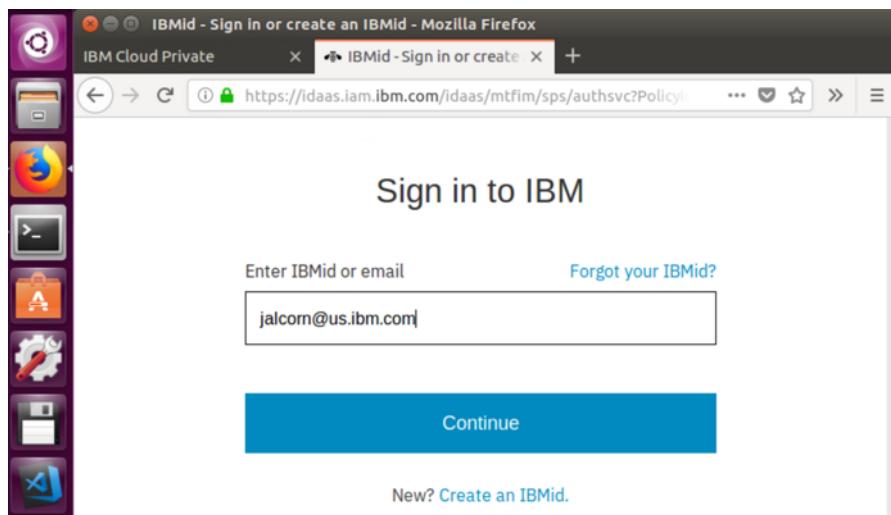
```

Congratulations, you have just deployed your first Node.js-based microservice!

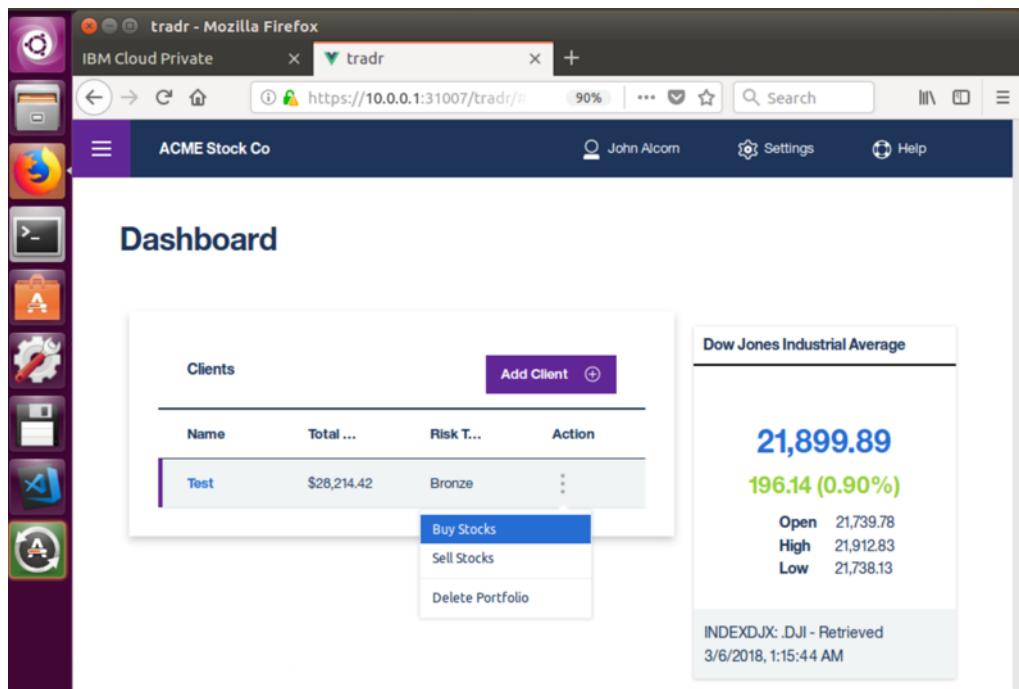


Run the code

Now that we've deployed our new Node.js/Vue.js based client for Stock Trader, let's give it a try. In a new tab in your browser, enter <https://10.0.0.1:31007/tradr> in the address bar. You will be redirected to do an IBMid login, just like with the Java-based client:

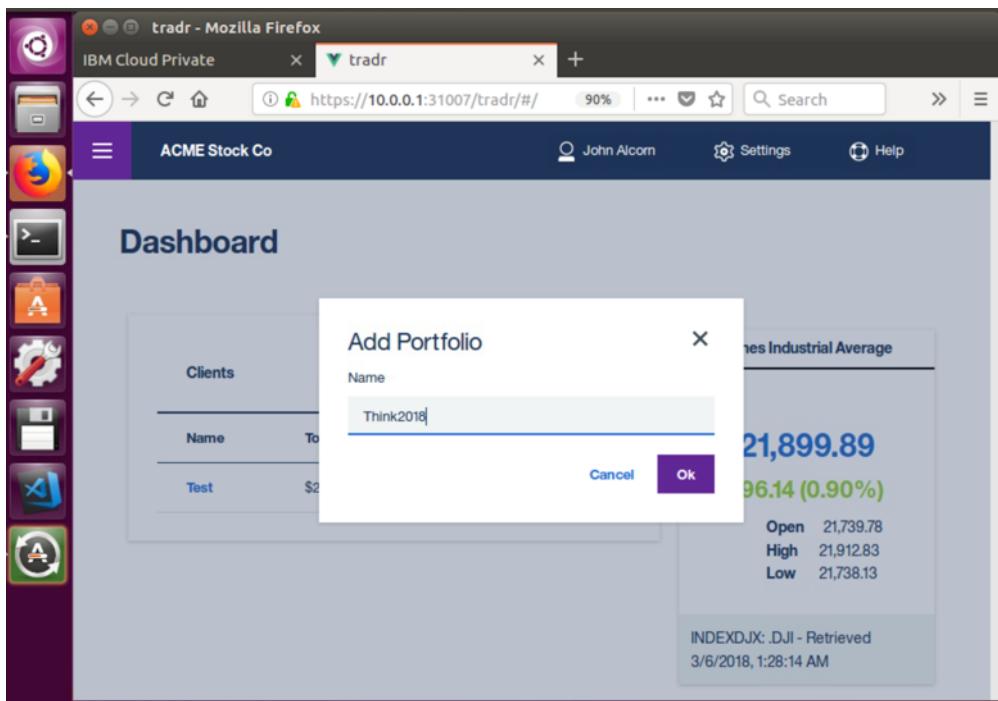


Once you complete the IBMid login, you will be redirected to the Node.js client's landing page:



You will notice that you see a list of all of your portfolios, which *tradr* calls “Clients”. It also calls the loyalty level “Risk Tolerance”. And at the top, you can see the name associated with the IBMid that you used to log in. You can click on the hyperlink name of a “client”, such as “Test”, to see the stocks in that portfolio. You can also click the vertical ellipsis under the Action column to access a menu that will let you buy or sell stocks, or delete a portfolio. All of these actions are calling the exact same REST services on the *portfolio* microservice that the Java-based client did; this client is simply rendering the results that come back in a different, more visually appealing way. This client also shows how the markets are doing on the far right, such as the latest value of the Dow Jones Industrial Average.

Let’s run through creating a new portfolio (what *tradr* calls a “client”), and adding a stock to it. First, click on the purple `Add Client` button. Then fill in a name, such as “Think2018” (as before, avoid names with spaces or any other characters invalid in a URL), and click the purple `OK` button.



Now you will see your new “Think2018” client/portfolio on the main landing page. Click on the vertical ellipsis under Action and choose `Add Stock`. Then enter your stock ticker symbol and the number of shares to buy, such as “IBM” and “42”, and click on the purple `OK` button.



The screenshot shows a Mozilla Firefox browser window with the URL <https://10.0.0.1:31007/tradr/#/>. The main page is titled "ACME Stock Co" and displays a "Dashboard". A modal dialog box is open, titled "Buy stocks for Think2018". It contains fields for "Stock Name" (set to "IBM") and "Shares" (set to "42"). Below the dialog, a table shows client information:

Name	Total
Test	\$26,500.00
Think2018	\$0.00

On the right side of the dashboard, there is a summary for the "Dow Jones Industrial Average" with the value **21,899.89** and a change of **196.14 (0.90%)**. It also shows historical data: Open 21,739.78, High 21,912.83, Low 21,738.13. A message at the bottom indicates the data was retrieved from INDEXDJX.DJI on 3/6/2018 at 1:28:14 AM.

Now you will see the updated values for the Think2018 portfolio on the main landing page. Add another stock, such as 7 shares of AAPL stock, so we'll have more than one stock in our portfolio table. Then click on the Think2018 hyperlink to view that portfolio.

The screenshot shows the same Mozilla Firefox browser window. The main page now displays the "Client Overview" for the "Think2018" portfolio. The "PORTFOLIO VALUE" is listed as **\$7,791.70** and the "YEAR TO DATE" performance is **1.576%**. Below this, the "Stock Portfolio" table shows the following data:

Symbol	Shares	Price	Date	Value
AAPL	7	\$178.12	2018-02-28	\$1,246.84
IBM	42	\$155.83	2018-02-28	\$6,544.86

At the bottom of the portfolio section are two buttons: "Sell Stock" and "Buy Stock".



You can click the left-pointing arrow to the left of the portfolio name to go back to the main landing page.

The screenshot shows a Mozilla Firefox browser window with the title bar "tradr - Mozilla Firefox". The address bar shows the URL "https://10.0.0.1:31007/tradr/#/". The main content area is titled "Dashboard". On the left, there is a vertical sidebar with several icons. The main content area has two main sections: "Clients" (a table with two rows) and "Dow Jones Industrial Average" (a card with current price, percentage change, and historical data). A small note at the bottom of the card states "INDEXDJX:.DJI - Retrieved 3/6/2018, 1:47:10 AM".

Name	Total A...	Risk T...	Action
Test	\$26,572.82	Bronze	⋮ ⋮
Think2018	\$7,791.70	Basic	⋮ ⋮

Dow Jones Industrial Average

21,899.89
196.14 (0.90%)

Open 21,739.78
High 21,912.83
Low 21,738.13

INDEXDJX:.DJI - Retrieved 3/6/2018, 1:47:10 AM

Feel free to play with this alternate client to the Stock Trader application to create or delete more portfolios, or to buy or sell more stocks.

Congratulations, in addition to working with a bunch of Java-based microservices, you have now used your first Node.js-based microservice. Due to the polyglot nature of Kubernetes and IBM Cloud Private, you can write your microservices in whatever languages you prefer, and in fact mix and match them, such as we just did, with a microservice written in Node.js calling one written in Java.



8. Optional: Deploying a Helm Chart from the Catalog

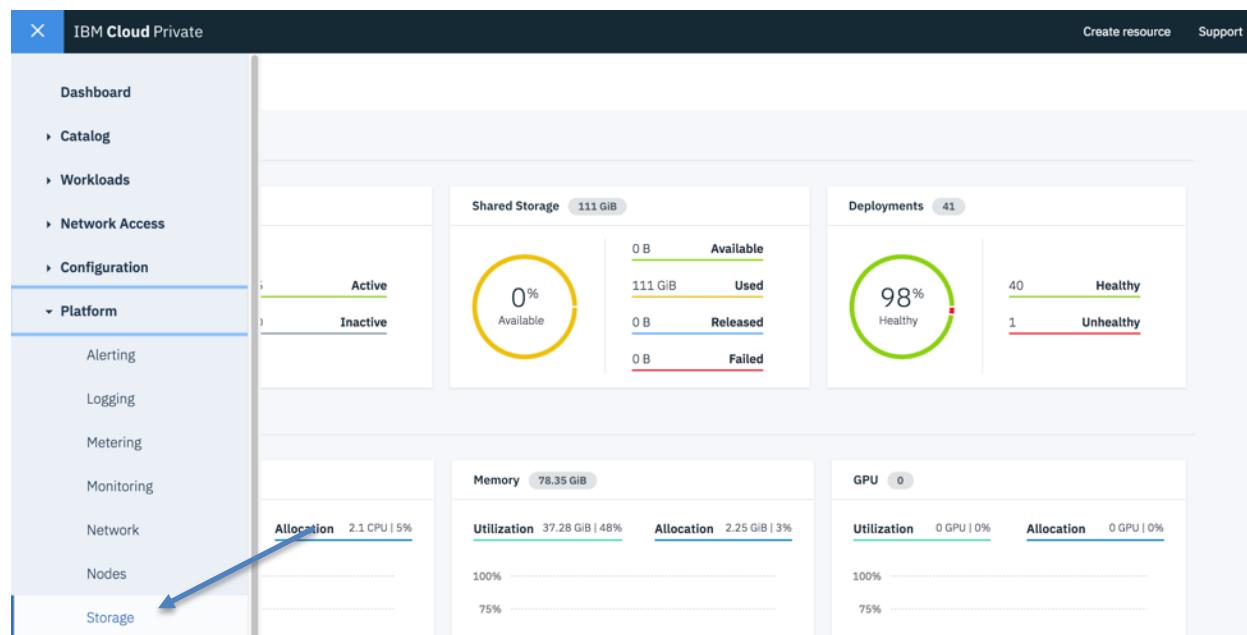
NOTE: This is an optional section. If you are short on time, feel free to skip ahead to the final section.

To have you experience the value of Helm Charts, this section will involve you installing and configuring **MQ**. While this traditionally took hours to install, with a containerized service deployed through Helm Charts, it really only takes minutes.

Create Persistent Volume

To deploy IBM MQ-Dev, you need to create a persistent volume that uses the “ReadWriteOnce” (RWO) access mode. You can create the storage volume from the UI, and should ideally be created before the deployment. However, you could create it after the deployment as well...Kubernetes will pause the deployment until it can find a matching persistent volume.

To create the volume, log into IBM Cloud Private, and use the upper left menu to navigate to the Storage page by selecting **Platform > Storage**



The screenshot shows the IBM Cloud Private interface. The top navigation bar includes 'IBM Cloud Private', 'Create resource', and 'Support'. The left sidebar has a tree view with 'Dashboard', 'Catalog', 'Workloads', 'Network Access', 'Configuration' (selected), and 'Platform' (selected). Under 'Platform', there are links for 'Alerting', 'Logging', 'Metering', 'Monitoring', 'Network', 'Nodes', and 'Storage'. A blue arrow points to the 'Storage' link. The main content area displays several cards: 'Shared Storage' (111 GiB, 0% Available, 111 GiB Used, 0 B Released, 0 B Failed), 'Deployments' (41, 98% Healthy, 40 Healthy, 1 Unhealthy), 'Memory' (78.35 GiB, Utilization 37.28 GiB | 48%, Allocation 2.25 GiB | 3%), and 'GPU' (0, Utilization 0 GPU | 0%, Allocation 0 GPU | 0%).

Once on the storage page, click **Create PersistentVolume**



PERSISTENTVOLUME

JSON mode On

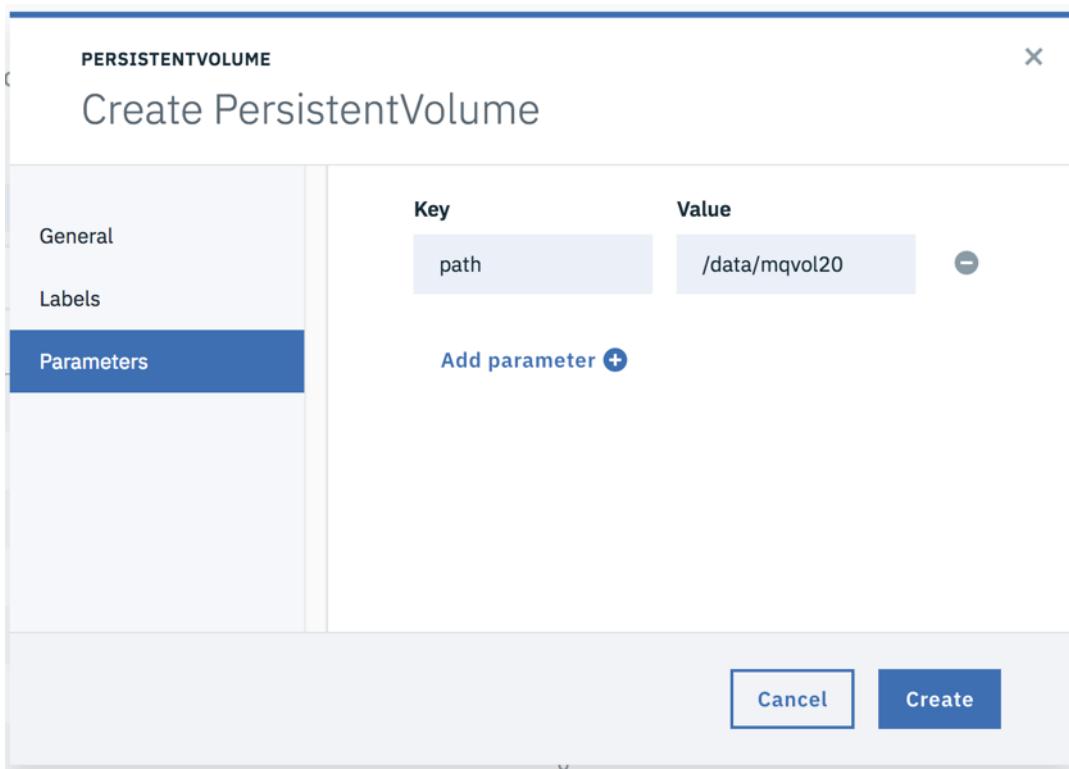
Create PersistentVolume

General	Name mqvol20
Labels	Storage class name
Parameters	Capacity Unit 2 Gi
	Access mode Read write once
	Reclaim policy Retain
	Storage type Host path
	<input type="button" value="Cancel"/> <input type="button" value="Create"/>



In this example, name: **mqvol20**, I created a Capacity: **2GB** Storage Type: **Host Path** with “**Read Write Once**” access mode.

Click the **Parameters** tab, and then enter **path**, and **/data/mqvol20**.



Click **Create**.

The value of using a persistent volume is that even though this recipe only deploys a single MQ queue manager, you can still achieve a level of availability in that if the queue manager pod or the node itself were to fail, IBM Cloud Private automatically re-schedules the queue manager to another node.

Get to the MQ Helm Chart

Next, install MQ itself.

For this example, we will use the catalog. However, if you want to use the command line, you can directly access the helm charts here: <https://github.com/IBM/charts>. Either path is fine, and as we both know, if you love kubectl and helm commands, then using the helm charts directly is a great option (you can [get helm command line here](#))

For the UI path, open IBM Cloud Private, and navigate to the catalog.



The screenshot shows the IBM Cloud Private dashboard. On the left, there's a sidebar with a tree view:

- Catalog** (selected)
- Helm Charts** (highlighted with a pink arrow)
- Images
- Workloads (Active, Inactive)
- Network Access
- Configuration
- Platform
- Manage
- Command Line Tools

The main area has several cards:

- Shared Storage**: 57 GiB
 - Available: 0 B
 - Used: 49 GiB
 - Released: 8 GiB
 - Failed: 0 B
- Deployments**: 21
 - Healthy: 100%
 - Unhealthy: 0
- Memory**: 15.67 GiB
 - Utilization: 12.26 GiB | 78%
 - Allocation: 1.75 GiB | 11%
- GPU**: 0
 - Utilization: 0 GPU | 0%
 - Allocation: 0 GPU | 0%

At the bottom left is the URL: <https://10.0.0.1:8443/catalog>

The catalog lists all the content you can install into IBM Cloud Private. Notice some are marked **community** and some are marked **ibm-charts**

Enter **mq** in the search to find MQ.

The screenshot shows the Catalog search results for 'mq':

Search Bar: Q **mq** Filter

Helm charts: Deploy your applications and install software packages

Chart Name	Description	Type
rabbitmq	Open source message broker software that implements the Advanced Message Queuing	Community
rabbitmq-ha	Highly available RabbitMQ cluster, the open source message broker software that	Community
ibm-rabbitmq-dev	Open source message broker software that implements the Advanced Message Queuing	ibm-charts
ibm-mqadvanced-server-dev	IBM MQ queue manager	ibm-charts



Click on **ibm-mqadvanced-server-dev**

From this details view, select the “Configure” button at the bottom, then go to the next step.

The screenshot shows the IBM Cloud Private interface with the following details:

- IBM MQ queue manager**: The chart name.
- ibm-charts**: The category.
- View Licenses**: License information.
- VERSION**: 1.1.0
- PUBLISHED**: Jan 2nd 2018
- TYPE**: Helm Chart

IBM MQ

IBM® MQ is messaging middleware that simplifies and accelerates the integration of diverse applications and business data across multiple platforms. It uses message queues to facilitate the exchanges of information and offers a single messaging solution for cloud, mobile, Internet of Things (IoT) and on-premises environments.

Introduction

This chart deploys a single IBM MQ Advanced for Developers server (queue manager) into an IBM Cloud private or other Kubernetes environment.

Prerequisites

Kubernetes 1.6 or greater, with beta APIs enabled

If persistence is enabled (see [configuration](#)), then you either need to create a PersistentVolume, or specify a Storage Class if classes are defined in your cluster.,

Installing the Chart

To install the chart with the release name `foo`:

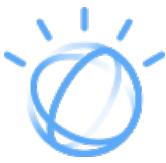
```
helm install --name foo stable/ibm-mqadvanced-server-dev --set license=accept
```

This command accepts the [IBM MQ Advanced for Developers license](#) and deploys an MQ Advanced for Developers server on the Kubernetes cluster. The [configuration](#) section lists the parameters that can be configured during installation.

Tip: See all the resources deployed by the chart using `kubectl get all -l release=foo`

Configure

There are quite a few settings in the MQ install, but I'll just highlight the ones you need to worry about.



Install Options 1 of 2

Configuration
IBM MQ queue manager Edit these parameters for configuration

Release name i
mq-greg-lab

Target namespace i
default

I have read and agreed to the [license agreements](#)

In this first section you need to give your deployment a name **mq-myname-lab**, select **default** namespace, then accept the license, otherwise the deployment will fail.

Install Options 2 of 2:

Service

Service name i
qmgr

Service type i
NodePort

Resources

CPU limit i
500m

Memory limit i
512Mi

CPU request i
500m

Memory request i
512Mi

Queue manager

Queue manager name i
trader

Admin password i

App password i
Enter value

Cancel Install



There are 4 fields you should consider in this last section.

dataPVC.size

Enter 2Gi. This is where you can customize how large you want your storage to be, but remember, it can't be bigger than the persistent volume you created earlier.

service.type

Select **NodePort**. If you keep it ‘ClusterIP’, then only apps running inside the IBM Cloud Private cluster will be able to access the MQ service. However, if you change it to ‘NodePort’, then you will be able to call the MQ service from the web browser.

queueManager.name

Enter **trader**, which is the name of the queue manager that your app will use.

queueManager.dev.adminPassword

Enter **admin** for the admin password.

Deploy the MQ Helm Chart

Once you are ready (you have accepted the license, created your persistent volume, and filled in all the advanced installation details) click **Install** and you are installing!

Depending on compute and storage speeds, it may take a few minutes to install since it needs to create default queues, and start the management UI.

To monitor the progress, select **View Helm Releases**, then scroll to **StatefulSet**. You'll see the pod running, select it and you can view logs.



The screenshot shows the Kubernetes UI for a StatefulSet named "mq-greg-lab-ibm-mq". On the left, the "Overview" tab is selected, displaying "StatefulSet details". It lists the following information:

Type	Detail
Name	mq-greg-lab-ibm-mq
Namespace	default
Images	ibmcom/mq:9;
Selector	app= mq-greg-lab-ibm-mq
Labels	app= mq-greg-lab-ibm-mq, chart= ibm-mqadvanced-server-dev-1.1.0, heritage= Tiller, release= mq-greg-lab
Service	qmgr
Desired replicas	1
Current replicas	1

On the right, the "Pods" section shows a table with one item:

NAME	NAMESPACE	STATUS	HOST IP	POD IP	READY	START TIME	ACTION
mq-greg-lab-ibm-mq-0	default	Running	10.0.0.1	10.1.64.94	1/1	Feb 17th 2018 at 8:08 AM	⋮

From the UI, click on “Workloads”, select the MQ Stateful Set, locate the pod and select it, then select the Logs tab.

The screenshot shows the "Logs" tab for the pod "mq-greg-lab-ibm-mq-0". The logs output is as follows:

```

: * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
: * See the License for the specific language governing permissions and
: * limitations under the License.
:
: * Enable and start a TCP/IP listener on port 1414
1 : ALTER LISTENER('SYSTEM.DEFAULT.LISTENER.TCP') TRPTYPE(TCP) PORT(1414) CONTROL(QMGR)
AMQ8021I: Request to start IBM MQ listener accepted.
AMQ8623I: IBM MQ listener changed.
2 : START LISTENER('SYSTEM.DEFAULT.LISTENER.TCP')
2 MQSC commands read.
All valid MQSC commands were processed.
No commands have a syntax error.
-----
Monitoring Queue Manager trader
QMNAME(trader)                                     STATUS(Running)
IBM MQ Queue Manager trader is now fully running
Server mqweb started with process ID 367.

```

From the command line: find the pod name and monitor its log:

```
kubectl get pods
```

```
kubectl logs <podname> -f
```



Start Using MQ

To open the MQ UI, you need to find what service MQ is listening through. To find which URLs to use, run the following to get the list of services:

```
kubectl get svc
```

Then once you identify the service name, run:

```
kubectl describe service <service Name>
```

Here is an example of what you will see:

```
Name: mqlab-ibm-mq
Namespace: default
Labels: app=mqlab-ibm-mq
chart=ibm-mqadvanced-server-dev-1.1.0
heritage=Tiller
release=mqlab
Annotations:
Selector: app=mqlab-ibm-mq
Type: NodePort
IP: 10.0.0.205
Port: qmgr-server 1414/TCP
TargetPort: 1414/TCP
NodePort: qmgr-server 30016/TCP
Endpoints: 10.1.64.117:1414
Port: qmgr-web 9443/TCP
TargetPort: 9443/TCP
NodePort: qmgr-web 32343/TCP
Endpoints: 10.1.64.117:9443
Session Affinity: None
External Traffic Policy: Cluster
Events: <none>
```

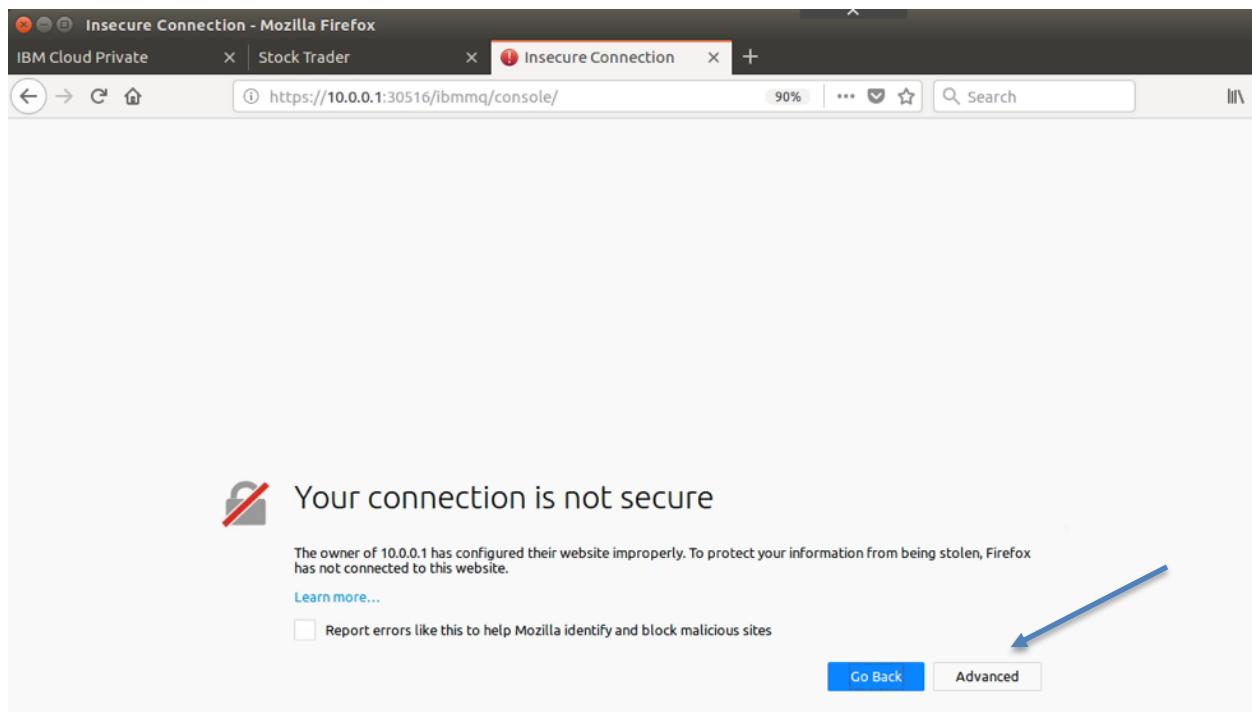
Notice there are 2 sets of ports. MQ uses 1414 for the API, and 9443 for the web UI. Make sure you use the correct NodePort when accessing the UI (in the above example, **32343**).

Paste this (with the correct node port) into a new browser tab:

<https://10.0.0.1:NodePort/ibmmq/console/>

You should get a “not secure” page, click **Advanced, Add Exception**, then **Confirm Security Exception**





Once you see the login on the console, you can start with the default userID and password you entered earlier: **admin/admin**

Queue Manager

Name	Status
trader	Running

Total: 1 Selected: 0 Updated: 8:20:36 AM

Channels on trader

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

Total: 2 Selected: 0 Updated: 8:18:21 AM

Queues on trader

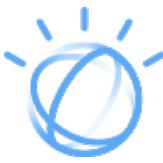
Name	Queue type	Queue depth
DEV.DEAD.LETTER.QUEUE	Local	0
DEV.QUEUE.1	Local	0
DEV.QUEUE.2	Local	0
DEV.QUEUE.3	Local	0

Total: 4 Selected: 0 Updated: 8:18:22 AM

Topics on trader

Name	Topic String
DEV.BASE.TOPIC	dev/

Total: 1 Selected: 0 Updated: 8:18:20 AM

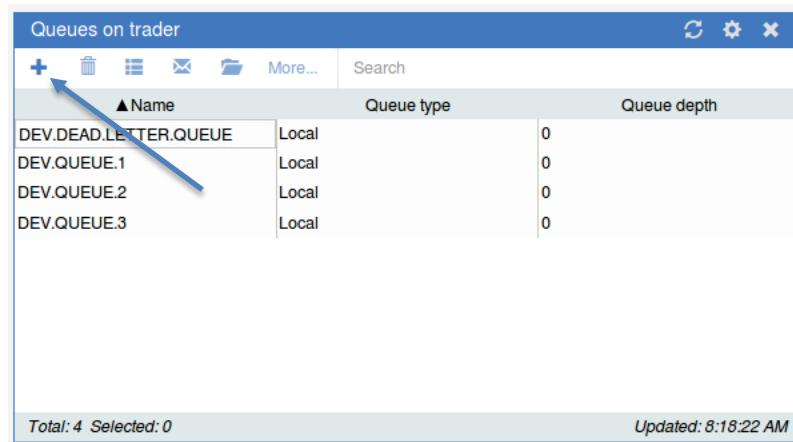


To start using MQ from an application, you will want to use the application user ID, not the admin user ID. The application user ID is “app” (with no default password). [Click here to learn more and see examples.](#)

Configuring MQ

Once MQ is running, we need to create the message queue that the app expected.

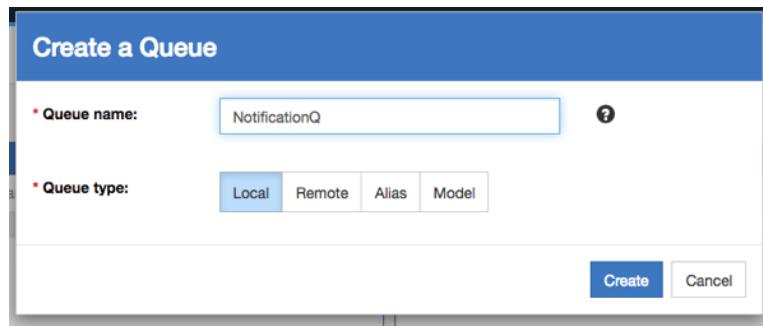
Select the + icon



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

Total: 4 Selected: 0 Updated: 8:18:22 AM

Create a queue called, **NotificationQ**



Create a Queue

* Queue name:

* Queue type: Local Remote Alias Model

Create **Cancel**

Next, we authorize the app user ID ‘app’ to the queue. The MQ instructions says NOT to use the default admin ID, but to use the default application ID, ‘app’. To do that, select **NotificationQ**, the **More...** menu, then **Manage Authority Records**.



Queues on trader		
	+ More...	Search
▲ Name		Queue depth
DEV.DEAD.LETTER.QUEUE	Clear queue...	0
DEV.QUEUE.1	Manage authority records...	0
DEV.QUEUE.2		0
DEV.QUEUE.3		0
NotificationQ		0

Total: 5 Selected: 1 Updated: 8:22:16 AM

Click the plus button, and enter **app**

Create Authorities

* Entity name: ?

* Entity type: User Group

Create Cancel

You will then see the group mqclient in the list. Select **mqclient** and select all the check boxes on the right.



Authority records for 'NotificationQ' on stocktrader

Entity name	Entity type
mqclient	Group
mqm	Group

Total: 2 Selected: 1 Updated: 12:34:14 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

You have unsaved changes Check all Uncheck all Save Close

Finally, click **Save** then **Close**.

Create Secrets

We have a recipe describing why secrets are so useful, and you can read it here (in a nutshell they make your application portable by letting you define custom, cloud-specific information regarding how the app can use each service):

<https://developer.ibm.com/recipes/tutorials/use-kubernetes-secrets-to-make-your-app-portable-across-clouds/>

Switch to use YOUR MQ service

In this section you will use Kubernetes Secrets to switch from using the MQ service we deployed to the MQ service you just deployed. As we mentioned earlier, our Stock Trader microservices use Kubernetes secrets to “learn” how to connect to services. This allows the cloud admin to “pivot” from using different versions of the services, even to pivot to use services not even running in IBM Cloud Private.

In this case, you’ll want to delete the existing Kubernetes secret for MQ:

```
kubectl delete secret mq
```



Then, find your MQ service name

```
kubectl get svc
```

Then viewed the properties

```
kubectl describe service your-mq-service-name
```

Now **create** the secret that lists your MQ instance:

```
kubectl create secret generic mq --from-literal=id=app  
--from-literal=pwd= --from-literal=host=<your-mq-service-name>  
--from-literal=port=1414 --from-literal=channel=DEV.APP.SVRCONN  
--from-literal=queue-manager=trader --from-literal=queue=NotificationQ
```

Finally, for the Stock Trader application to use your new MQ service, you should delete the **loyalty-level** and **messaging** pods (via `kubectl delete pod <pod name>`). Kubernetes will immediately replace them (since the associated deployment has a Desired pod count of 1, and you just reduced the actual number to 0) and as a result the new pods will pick up the new secret.

NOTE: If you want to go back to the MQ service we created earlier, just delete your secret and run this command:

```
kubectl create secret generic mq --from-literal=id=app  
--from-literal=pwd= --from-literal=host=mqlab-ibm-mq  
--from-literal=port=1414 --from-literal=channel=DEV.APP.SVRCONN  
--from-literal=queue-manager=stocktrader  
--from-literal=queue=NotificationQ
```



9. Conclusions

Congratulations! You have successfully managed and updated a microservice-based application running in IBM Cloud Private. You now know how to package and build a Docker image containing WebSphere Liberty and your application; and how to update existing application instances running in IBM Cloud Private. You have also deployed a Node.js based microservice. You have explored the scaling and auto-recoverability features of IBM Cloud Private. Finally, you have deployed a helm chart from the catalog.

We hope you have enjoyed exploring IBM Cloud Private, and this little sample meant to showcase it. Please share any feedback you might have, on the lab instructions, the Stock Trader sample, Kubernetes, IBM Cloud Private, or anything else regarding your experience in this lab.

Note that IBM also has an IBM Container Service, which is an IBM-hosted Kubernetes environment, and Stock Trader runs great there too. If you wish, you can try it out at <https://myibmcloudcluster.us-south.containers.mybluemix.net:32389/trader>. This is the exact same microservice code, just running in the IBM public cloud (though note it isn't integrated with IBMID, so you just login as stock/trader).

If you wish to play with the Stock Trader sample on your own later, all of its code is in public GitHub, at <https://github.com/IBMStockTrader>. Built versions of the Docker container for each microservice are also in public DockerHub, at <https://hub.docker.com/r/ibmstocktrader>.

Thank you for attending, and for your interest in microservices, Kubernetes, and IBM Cloud Private!

Contacts

John Alcorn (jalcorn@us.ibm.com) Hybrid Cloud CTO Office

Greg Hintermeister (greh@us.ibm.com) Hybrid Cloud CTO Office



Acknowledgements and Disclaimers

Copyright © 2017 by International Business Machines Corporation (IBM). No part of this document may be reproduced or transmitted in any form without written permission from IBM.

U.S. Government Users Restricted Rights — use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either express or implied. In no event shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted according to the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts.

In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply."

Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.

Performance data contained herein was generally obtained in a controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer is in compliance with any law.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a particular, purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

IBM, the IBM logo, ibm.com, Aspera®, Bluemix, Blueworks Live, CICS, Clearcase, Cognos®, DOORS®, Emtoris®, Enterprise Document Management System™, FASP®, FileNet®, Global Business Services®, Global Technology Services®, IBM ExperienceOne™, IBM SmartCloud®, IBM Social Business®, Information on Demand, ILOG, Maximo®, MQIntegrator®, MQSeries®, Netcool®, OMEGAMON, OpenPower, PureAnalytics™, PureApplication®, pureCluster™, PureCoverage®, PureData®, PureExperience®, PureFlex®, pureQuery®, pureScale®, PureSystems®, QRadar®, Rational®, Rhapsody®, Smarter Commerce®, SoDA, SPSS, Sterling Commerce®, StoredIQ, Tealeaf®, Tivoli® Trusteer®, Unica®, urban{code}®, Watson, WebSphere®, Worklight®, X-Force® and System z® Z/OS, are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: www.ibm.com/legal/copytrade.shtml.

