# How to Set Up Stock Trader

Current as of May 30, 2019

Deployed onto ICP 3.1.2

Github location: <https://github.com/IBMStockTrader>

**Look at the comments in RED pen, and those labeled with KLP: These indicate notes that I made when I followed this doc to get the Stock Trader app up and running successfully.**

This is a description of how to set up Stock Trader, including all the required services.

INTERNAL USE ONLY because it has some of our actual API keys and such which we do not want public.

[How to Set Up Stock Trader 1](#_Toc7100567)

[Access the PYRK8s IBM Stock Trader Project Skytap environment. 2](#_Toc7100568)

[Clone the Stock Trader repos 3](#_Toc7100569)

[Cluster Setup 3](#_Toc7100570)

[NFS Server 3](#_Toc7100571)

[Preparing Cluster for Stock Trader 4](#_Toc7100572)

[Create Namespace “stock-trader” 4](#_Toc7100573)

[Create Image Policy 4](#_Toc7100574)

[Installing Services Needed by Stock Trader 5](#_Toc7100575)

[Install DB2 5](#_Toc7100576)

[Install DB2 Helm Chart 7](#_Toc7100577)

[Create a secret so that the Stock Trader app will be able to use DB2 8](#_Toc7100578)

[Deploy MQ 8](#_Toc7100579)

[Create queue: NotificationQ 10](#_Toc7100580)

[Create a secret so that the Stock Trader app will be able to use MQ 12](#_Toc7100581)

[Deploy Redis 12](#_Toc7100582)

[Create a secret so that the Stock Trader app will be able to use Redis 13](#_Toc7100583)

[Deploy ODM 14](#_Toc7100584)

[Obtain Stock Trader’s Loyalty decision service 17](#_Toc7100585)

[Import the Loyalty decision service into ODM 18](#_Toc7100586)

[Create a secret so that the Stock Trader app will be able to use ODM 20](#_Toc7100587)

[Deploy Event Streams 20](#_Toc7100588)

[Create Secrets 20](#_Toc7100589)

[Deploy Stock Trader Pods 23](#_Toc7100590)

[Prepare: 23](#_Toc7100591)

[Deploy the pods 25](#_Toc7100592)

[Find Stock Trader URLs 26](#_Toc7100593)

[Validate the solution. 27](#_Toc7100594)

ORDER YOU MUST FOLLOW:

1. Access the environment
2. Clone the StockTrader git repos
3. Set up Cluster
4. Install services
5. Create Secrets
6. Install Stock Trader app
7. Validate and troubleshoot the solution as needed to get it functioning properly

## Access the PYRK8s IBM Stock Trader Project Skytap environment.

If using the provided Skytap environment, Use the **Workstation** VM for your work.

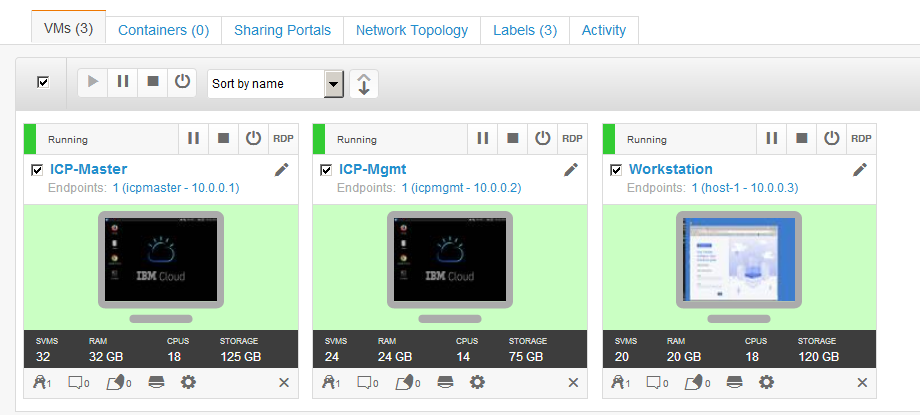
VM credentials are: ibmdemo / passw0rd

ICP Admin Console URL: <https://mycluster.icp:8443>

ICP credentials are: admin / admin

ICP cloudctl login API endpoint: https://10.0.0.1:8443

ICP is installed all-in-one VM on IP 10.0.0.1 (ICP-Master)



## Locate the Stock Trader projects on the Workstation VM

KLP: These have already been cloned on the Workstation VM.

Location: **/home/ibmdemo/IBMStockTrader**

**Note:** Only clone the repos if you are **NOT** using the Skytap environment provided

KLP: I have already cloned the github repositories on the Workstation VM. They are located in **/home/ibmdemo/IBMStockTrader**

Here are the git clone commands I used to clone the repos locally.

git clone <https://github.com/IBMStockTrader/trader>

git clone <https://github.com/IBMStockTrader/portfolio>

git clone <https://github.com/IBMStockTrader/messaging>

git clone <https://github.com/IBMStockTrader/notification-twitter>

git clone <https://github.com/IBMStockTrader/stock-quote>

git clone <https://github.com/IBMStockTrader/loyalty-level>

git clone <https://github.com/IBMStockTrader/stocktrader>

## Cluster Setup

**KLP: The cluster is already setup for you if using the ICP Project Skytap environment**

We won’t detail the instructions but general rules:

* We have 1 master, 1 management, and at least 4 workers.
* We have a dedicated NFS server per cluster (many times NFS servers have crapped out on us
* Sometimes we use Gluster but clients don’t due to issues so we try to mimic client environments
* Unique cluster name: We have found more problems with ICP assuming the default name that we have to name it uniquely…just like clients would

## NFS Server

**KLP: The NFS server is already setup for you**

We create a new NFS server for each cluster so that when an NFS cluster has problems it only affects a single cluster.

**To create new NFS server:**

**KLP: I did all of the NFS server on 10.0.0.3 – (Workstation Node)**

**KLP: The following steps have been completed on the Workstation VM, so you can assume NFS server is already running.**

The NFS shared folder is **/shared** (on the workstation VM, IP: 10.0.0.3)

apt-get install nfs-kernel-server

mkdir -p **/shared**

chown nobody:nogroup /shared

vi /etc/exports

added line--> /shared \*(rw,no\_subtree\_check,async,insecure,no\_root\_squash)

systemctl restart nfs-kernel-server

**To create a folder for use by PV: (From Workstation VM)**

**FYI ONLY….. See Details below in specific installation sections**

cd /shared

mkdir -p <foldername>

chmod 777 <foldername>

When creating the PV through the ICP UI, use these parameters:   
server = 10.0.0.3 (workstation VM)  
path = /shared/<foldername>

# Preparing Cluster for Stock Trader

## Create Namespace “stock-trader”

Create Namespace with “**ibm-anyuid-psp**” policy. This will create one with anyuid and restricted.

Name = **stock-trader**

## Create Image Policy

This is a change to ICP 3.1.x. Each image repo needs to be “allowed” through an image policy. Select the “**ibmcloud-default-cluster-image-policy**

* Verify that **docker.io/store/ibmcorp/\*** is in the list.
* Verify that **mycluster.icp:8500/\*** is in the list.

If these do not exist, add them, or create a new image pull policy with these listed.



If not, add it.

KLP: This is already in the default image policy. But you can create a new policy if you like, which only includes the required policy info.

# Installing Services Needed by Stock Trader

## Install DB2

KLP: I used DB2 helm release **3.2.0**, which was latest at time of this writing

Generally, we install Db2 in the `**default**` namespace since additional security needs to be configured if it’s installed into `stock-trader` or any other namespace.

The following deployment considerations are used:

* + namespace=**default**
  + plain DB2 (not HADR)
  + database=trader for Stock Trader
  + using this chart: ibm-db2oltp-dev

**Prereq #1:**  Docker container – generate a key

* Follow instructions in chart’s readme as to visiting docker store and subscribe to Db2.
* You need to copy your generate key’s value, because you will use it when you create a secret.
* NOTE: Docker Cloud changed, so instead of the API key in password, just use your dockerHub userID and password in the secret.

Make secret. I used <secretname>=**db2dockerregistry**

Secret using your Docker Hub credentials:

**KLP, I used this. And specified my public docker credentials**

kubectl create secret docker-registry db2dockerregistry --docker-username=<YOUR DOCKER ID> --docker-password=<YOUR DOCKER PASSWORD> --docker-email=userh@gmail.com -n default

**Prereq #2:** pod security KLP, I used **default** Namespace

Need to run a setup script, be sure you can kubectl to the cluster

Decide the namespace to run Db2 in, then:

* Clone <https://github.com/IBM/charts>
* CD to charts/stable/ibm-db2oltp-dev/ibm\_cloud\_pak/pak\_extensions/prereqs
* Run the following: ./createNSandSecurity.sh --namespace <NAMESPACE>

**Prereq #3:** Storage – create the persistent volume

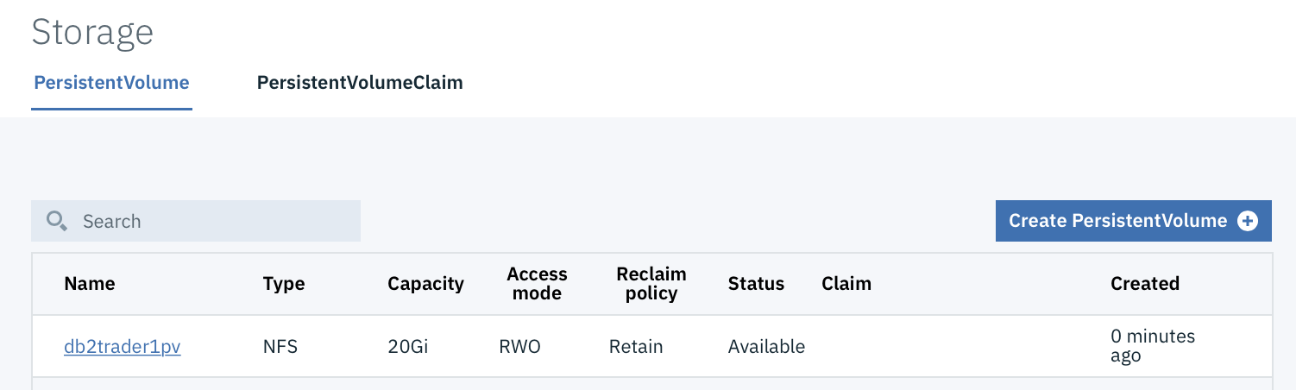
For plain DB2, we need only 1 PV with Storage. Here we will use **NFS**.

1. On Workstation VM, go to /shared and create the /shared/db2trader1 folder used for NFS server. Then chmod 777

(b) go to ICP dashboard and create Persistent Volume

capacity=20Gi (KLP: I used 10 Gi)  
RWO  
type=NFS

server = 10.0.0.3 (KLP: 10.0.0.3 – the Workstation node)  
path = /shared/db2trader1



## 

## Install DB2 Helm Chart

ICP Dashboard > Catalog > start to type in DB2 > select **ibm-db2oldp-dev**

KLP: At the time of this writing, I installed the latest version, which is **v3.2.0**

Review the readme then click **Configure**  
  
Note that in 3.1, the ICP dashboard has some of the parameters in collapsible sections. Expand “All parameters”.

Provide or verify the following input values:

* + - * Release name = db2trader1
      * namespace = default
      * agree to license
      * secret name = db2dockerregistry
      * db2 instance name = db2inst1
      * size of volume claim = 2 Gi (KLP)
      * pw for db2 instance name = inst1pw (KLP: used **passw0rd**)
      * database name = trader
      * OPTIONAL: For Resource configuration, you can make pods smaller
        + the memory request to 1Gi
        + the memory limit to 4Gi
        + CPU request = 1000m (KLP: I set to 25m)
        + CPU limit = 2000m
      * Install

Go to the Helm Release, then StatefulSet and look at the Pod. Look at the Pod’s Events to ensure that the Events are progressing, and are not stuck on some error. NOTE: It may show “unhealthy” for a bit of time, but give it time. As long as it binds to the PV, and pulls the image it should build OK.

Then create the tables. Note that the createTables.ddl file is included in the “portfolio” obtained from a repo

Get into the Db2 pod:

kubectl exec -it db2trader1-ibm-db2oltp-dev-0 -n default bash

sudo su db2inst1

db2 connect to trader

Create the tables needed by StockTrader:

db2 CREATE TABLE Portfolio\(owner VARCHAR\(32\) NOT NULL, total DOUBLE, loyalty VARCHAR\(8\), balance DOUBLE, commissions DOUBLE, free INTEGER, sentiment VARCHAR\(16\), PRIMARY KEY\(owner\)\)

db2 CREATE TABLE Stock\(owner VARCHAR\(32\) NOT NULL, symbol VARCHAR\(8\) NOT NULL, shares INTEGER, price DOUBLE, total DOUBLE, dateQuoted DATE, commission DOUBLE, FOREIGN KEY \(owner\) REFERENCES Portfolio\(owner\) ON DELETE CASCADE, PRIMARY KEY\(owner, symbol\)\)

db2 list tables

## 

## Create a secret so that the Stock Trader app will be able to use DB2

The db2 is in the ‘**default’** namespace.

The client using it will be in the ‘**stock-trader**’ namespace.

First, get the data for the service with the Node Port. You’ll see 2 Db2 services. Pick the one that ends with -db2.



ICP Dashboard > Network Access > Services > \*.ibm-db2oltp-dev-db2

Look for the Cluster IP (example 10.0.0.101) (KLP, may be different on your machine)

kubectl create secret generic db2 --from-literal=id=<**your Db2 instance name**> --from-literal=pwd=<**DB2 Password you specified**> --from-literal=host=<**Your DB2 Host IP**> --from-literal=port=50000 --from-literal=db=<**Your DB2 database name**> -n stock-trader

## Deploy MQ

KLP: I deployed MQ helm release **3.0.1**, which was latest at time of this writing

In this environment, the following deployment considerations will be used:

* namespace=stock-trader
* plain MQ (not some multi-queue manager fancy thing)
* queue manager name = stocktrader
* queue name = NotificationQ

**Prereq #1:** Storage – create the persistent volume

MQ needs only 1 PV with Storage. Here we will use NFS. If you use Gluster, you can skip this step.

1. From Workstation VM, create the NFS directory and chmod to 777

Go to the /shared directory on Workstation VM

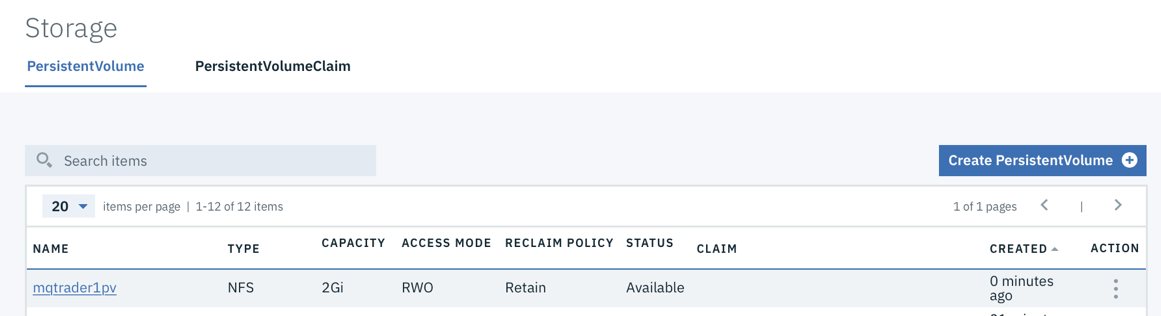
mkdir -p mqtrader1

chmod 777 mqtrader1

1. go to ICP dashboard and create Persistent Volume

capacity=2Gi  
RWO  
type=NFS

server = 10.0.0.3 (KLP: 10.0.0.3 – Workstation node)  
path = /shared/mqtrader1



**Install MQ Helm Chart**

ICP Dashboard > Catalog > start to type in MQ > select **ibm-mqadvanced-server-dev**

KLP: I Used version **3.0.1** helm chart, latest version at time of writing

Review the readme then click **Configure**

Provide or verify the following input values:

release name = mq-trader1  
target namespace = stock-trader   
accept license   
Service type = NodePort  
Queue manager name = stocktrader  
Admin pw = mq1pw **(KLP: I used passw0rd)**  
App pasword = LEAVE BLANK

**CPU request: 50m (KLP added)**

**Uncheck Dynamic PVC**

Click INSTALL.

Go to the StatefulSet and look at the Pod.

Look at the Pod’s Events to ensure that the Events are progressing, and are not stuck on some error

Look at the Pod’s Logs to ensure that the MQ is started and that the queue manager “stocktrader” has been created

**Log in to MQ Dashboard**

When MQ is completely up, figure out how to log in to the MQ console:

* Helm Releases > mq-trader1
* View services
* View the port for the “mq-trader1-ibm-mq”



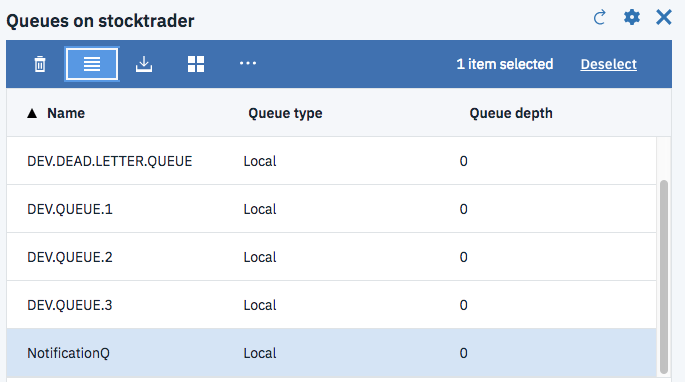
Look at your proxy node IP address (in menu Platform > Nodes), and add it to the beginning of the URL so it should look something like:

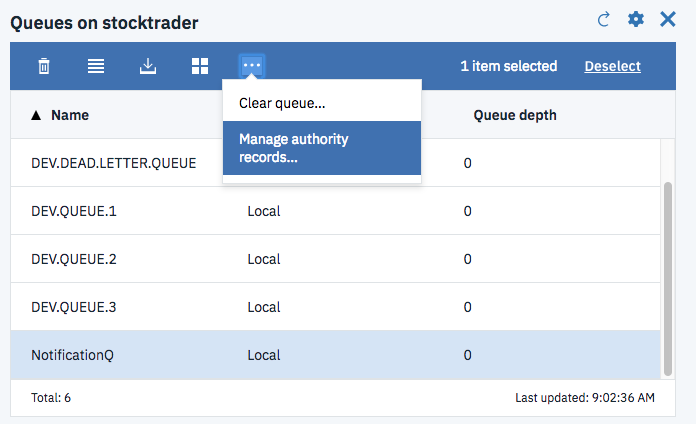
**KLP: Get the Nodeport for the MQ Console, not the QMGR, from the service)**

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

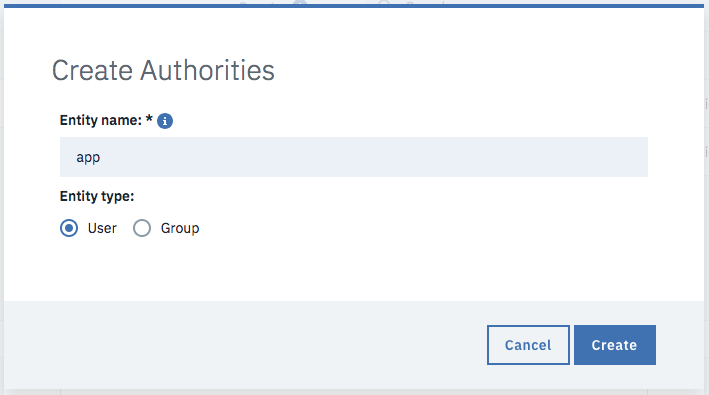
credentials: (admin / <Your MQ Password>

## Create queue: NotificationQ

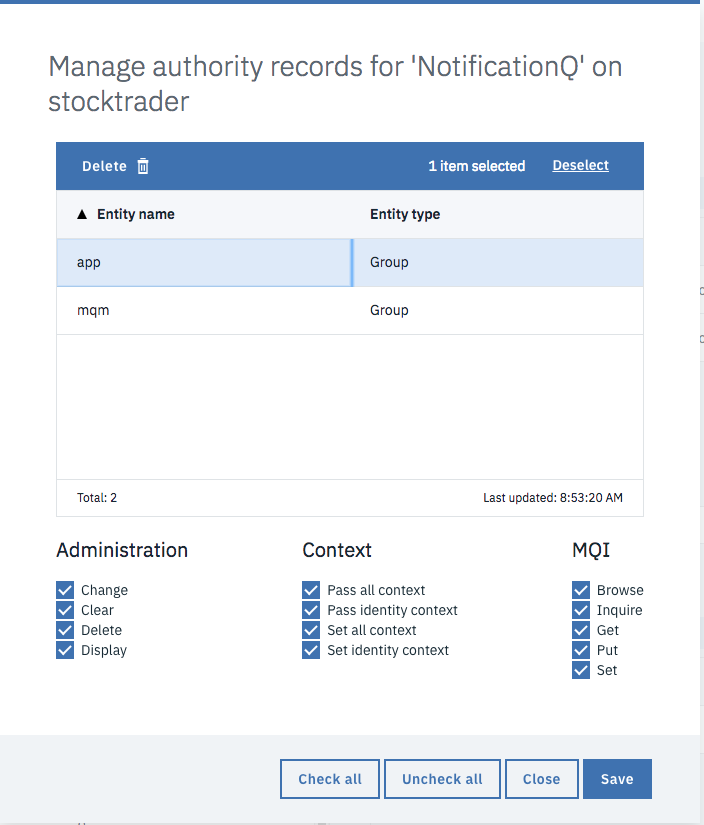
* Log in to the MQ Dashboard **(KLP…. admin / passw0rd ). You may have setup a different password for your MQ.**
* In the section for “Queues on stocktrader”, click on Create
* Give the queue the name NotificationQ. It is of type **Local** queue. Click Create.
* Select the NotificationQ, and then click on Properties
* 
* Set the default persistence property to be **Persistent** and then Save
* Set the authority records:
  + select the NotificationQ then click on More... then Manage Authority Records...



* Click on the “Create +” and add Entity name = app (“app” is a User)

  
this shows up as app (entity type Group, even though User was selected)  
Select app  
select boxes for **Browse, Inquire, Get, and Put** then remember to Save

* NOTE, lately I’ve had to select all of them as below.



## 

## Create a secret so that the Stock Trader app will be able to use MQ

This secret must exist in the “stock-trader” namespace.

To know the IP for the secret, go in the ICP dashboard > Network Access > Services

Click on NameOfMQService, then look for the IP = 10.0.0.x --> 10.0.0.132 (KLP, your IP will be different)

Then create the MQ secret in the stock-trader namespace, using the following literal values, similar to how you created the Db2 secret.

Secret name: **mq**

Namespace: **stock-trader**

id=app

pwd=

host==<**your MQ service name**>

port=1414

channel=DEV.APP.SVRCONN

queue-manager==<**your MQ Qmgr Name**>

queue==<**Your Notification Queue name**>

*// Leave “pwd=” blank. Doesn’t seem right but doesn’t work if you add an App password.*

## Deploy Redis

Redis is available in IBM Charts, but it uses token-based authentication, whereas Stock Trader was developed using password-based authentication.

Here are instructions on deploying Bitnami password-based Redis:

Add Bitnami charts:

* Add the helm repo via “Manage > Helm Repositories”
  + https://charts.bitnami.com/bitnami
* Refresh repos

You’ll see this:



**Install Redis Helm Chart**

**KLP: Deploy the Bitnami Redis chart, not the IBM Redis chart….**

ICP Dashboard > Catalog > start to type in Redis > select the **Bitnami redis chart**

Review the readme then click Configure

Provide or verify the following input values:

Release name = redistrader1

namespace = stock-trader

accept license

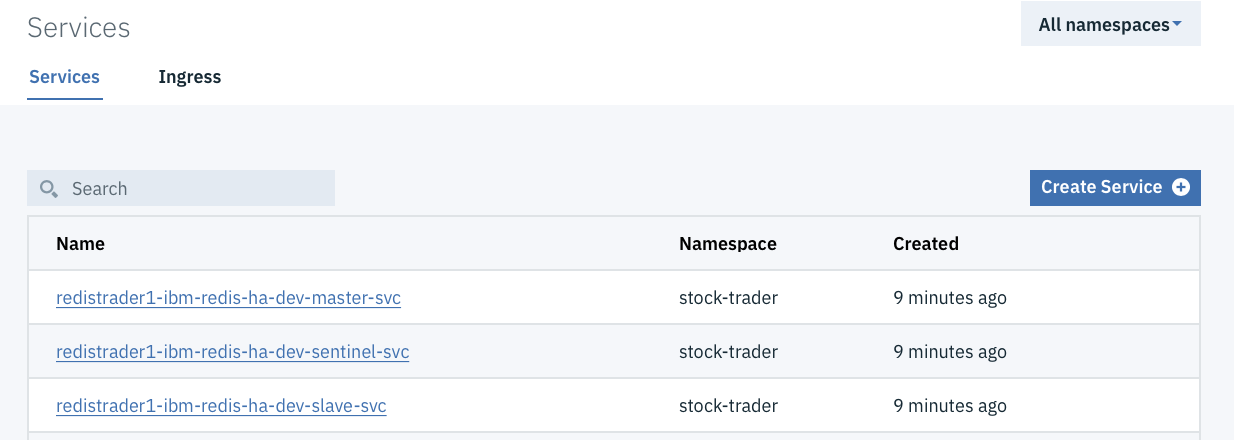
password=passw0rd (**KLP, password field only on the Binami Redis chart, not in the IBM redis chart.**

Install

## Create a secret so that the Stock Trader app will be able to use Redis

This secret must exist in the “stock-trader” namespace.

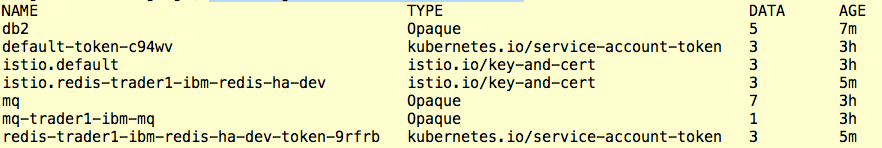
To know the IP for the secret, go in the ICP dashboard > Network Access > Services > Click on master-svc



Then create the secret:

First, find the redis password secret:

kubectl get secrets -n stock-trader



**KLP, my secret is redistrader1, which got created when I installed the Btnami redis chart**

Then copy the redis secret into the following command and run it:

export REDIS\_PASSWORD=$(kubectl get secret -n stock-trader <**Your redis service name**> -o jsonpath="{.data.redis-password}" | base64 --decode)

**KLP: This resulted in passw0rd that we put in the password field in the chart deployment**

*// Depending on version of Redis, the password may be defaulted so need to run this export to get it into a variable so that the secret can be correct*

kubectl create secret generic redis --from-literal=url=redis://x:$REDIS\_PASSWORD@<**Your redis service name**>:6379 -n stock-trader

*// “url=” will be different for your setup. Find the Redis service name and put after the @ and before the :6379*

## Deploy ODM

KLP: I deployed ODM helm release **2.1.0**, which was latest at time of this writing

In this environment, the following deployment considerations will be used:

namespace=stock-trader

use the dev ODM chart

**Prereq #1:** Storage – create the persistent volume

ODM dev needs 1 PV. Here we will use NFS.

From Workstation VM, create the NFS directory listed below, and chmod 777

KLP: On the 10.0.0.3 Workstation VM

cd /shared

mkdir -p odmtrader1

chmod 777 odmtrader1

go to ICP dashboard and create Persistent Volume

name= odmtrader1pv

capacity=2Gi  
 RWO  
 type=NFS

server = 10.0.0.3 (KLP: 10.0.0.3)  
 path = /shared/odmtrader1

**Install ODM Helm Chart**

ICP Dashboard > Catalog > start to type in ODM > select ibm-odm-dev

Review the readme then click Configure

Provide or verify the following input values:

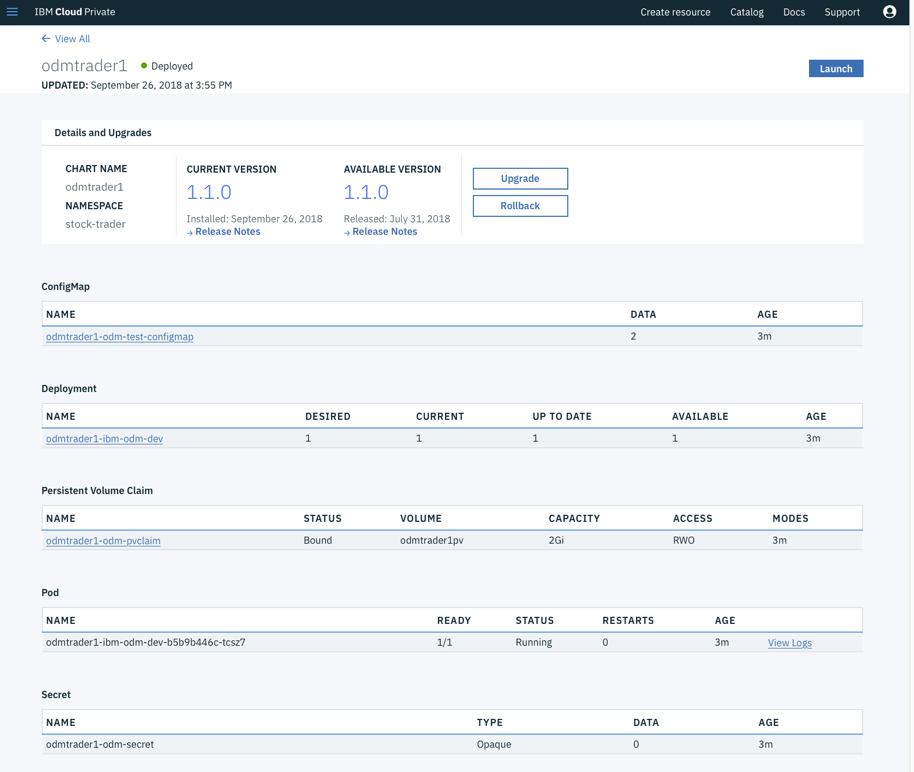
Release name = odm-trader1

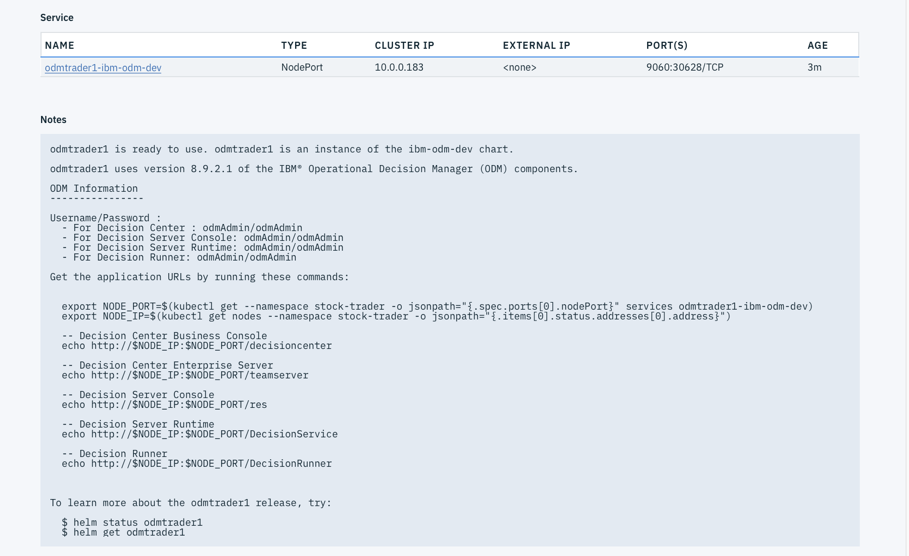
namespace = stock-trader

agree to license

KLP reduce **cpu request** to **50m** from 1. Kept **limit** at default of **2**. Otherwise you may get a failure on CPU unavailable during the deployment.

Click Install





You’ll notice in the “Notes” section a bunch of tips on how to find the URL for the Decision Center, and what the user ID and passwords are. Here’s what it listed out:

odmtrader1 is ready to use. odmtrader1 is an instance of the ibm-odm-dev chart.

odmtrader1 uses version 8.9.2.1 of the IBM® Operational Decision Manager (ODM) components.

ODM Information

----------------

Username/Password :

- For Decision Center : odmAdmin/odmAdmin

- For Decision Server Console: odmAdmin/odmAdmin

- For Decision Server Runtime: odmAdmin/odmAdmin

- For Decision Runner: odmAdmin/odmAdmin

Get the application URLs by running these commands:

export NODE\_PORT=$(kubectl get --namespace stock-trader -o jsonpath="{.spec.ports[0].nodePort}" services odmtrader1-ibm-odm-dev)

export NODE\_IP=$(kubectl get nodes --namespace stock-trader -o jsonpath="{.items[0].status.addresses[0].address}")

-- Decision Center Business Console

echo http://$NODE\_IP:$NODE\_PORT/decisioncenter

-- Decision Center Enterprise Server

echo http://$NODE\_IP:$NODE\_PORT/teamserver

-- Decision Server Console

echo http://$NODE\_IP:$NODE\_PORT/res

-- Decision Server Runtime

echo http://$NODE\_IP:$NODE\_PORT/DecisionService

-- Decision Runner

echo http://$NODE\_IP:$NODE\_PORT/DecisionRunner

To learn more about the odmtrader1 release, try:

$ helm status odmtrader1

$ helm get odmtrader1

All you need to get Stock Trader running is the Decision Center:

export NODE\_PORT=$(kubectl get --namespace stock-trader -o jsonpath="{.spec.ports[0].nodePort}" services odm-trader1-ibm-odm-dev)

export NODE\_IP=$(kubectl get nodes --namespace stock-trader -o jsonpath="{.items[0].status.addresses[0].address}")

echo http://$NODE\_IP:$NODE\_PORT/decisioncenter

http://10.0.0.1:<NodePort>/decisioncenter

## Obtain Stock Trader’s Loyalty decision service

From terminal window, cd to your repository directory. On the Workstation VM, it is **/home/ibmdemo/IBMStockTrader/portfolio**

cd into the portfolio directory and find the file that contains the Loyalty decision service: stock-trader-loyalty-decision-service.zip

## Import the Loyalty decision service into ODM

Log in to the Decision Center console: [http://10.0.0.1:<NodePort>/decisioncenter](http://10.0.0.1:%3cNodePort%3e/decisioncenter)

Login Credentials: odmAdmin /odmAdmin

Click on LIBRARY  
 Click on the weird Up Arrow

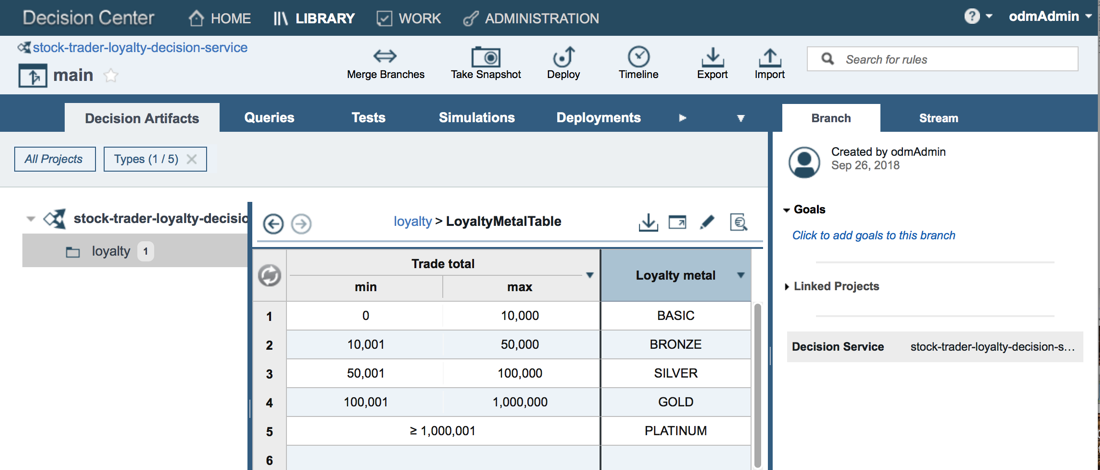
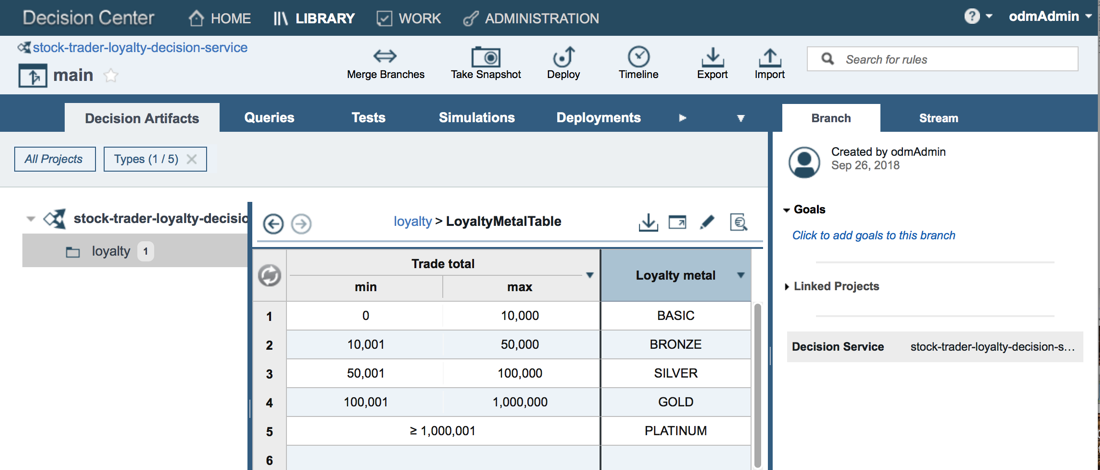
Click on Choose...

browse to and select the .zip

Import

After the import, click on the Branches tab and then click on “main”

On the “Decision Artifacts” tab, click on “loyalty” and then click on “LoyaltyMetalTable”

Here is the decision table in which you can alter the values  
 

**IMPORTANT:** You also need to **deploy** the rule app

* Click on the curved up arrow for **Deploy**



* This says it will deploy from the Decision Center to a target Decision Server.
* Click on Deploy then OK

KLP: Note that this step of loading the rules and deploying is required after each restart of the ICP environment, as we have not setup persistence for ODM.

### Create the odm secret so that the Stock Trader app will be able to use ODM

Then create the ODM secret in the stock-trader namespace, using the following literal values, similar to how you created the Db2 secret.

Secret name: **odm**

Namespace: **stock-trader**

url=http://odm-trader1-ibm-odm-dev:9060/DecisionService/rest/ICP\_Trader\_Dev\_1/determineLoyalty

id=<**Your odmAdmin ID**>

pwd==<**Your odmAdmin Password**>

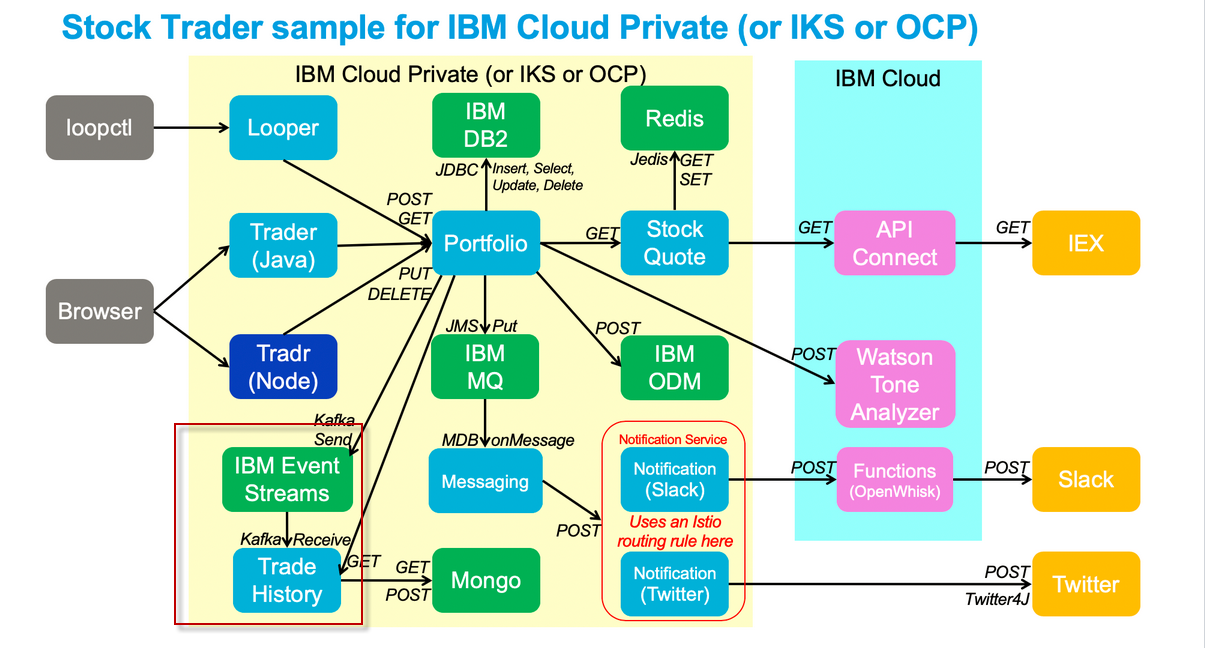
*// “url=” may be different for your setup. Find the ODM service name and add to URL before the port. Also add the ID and PWD you added in the helm chart.*

## Deploy Event Streams

**KLP: I did not do the event streams and Kafka. Too heavy weight, and need more cpu, RAM on VMs to install it**

**KLP: The event streams is used in a scenario for getting your trade history. So we will not setup that capability in this exercise.**

**KLP: Because we are not setting up event streams and its associated Kubernetes secrets, I had to modify the deploy.yaml file for the portfolio service, to remove the variables that were being pulled from the Kafka secret.**



## Create Secrets

KLP: TIP: Want to see the text of the secrets that I created when I got the solution to run? On the Workstation VM, look in folder **/home/ibmdemo/KLP-IBMStockTrader-secrets**

**#Db2**

KLP: You should have already created this. Verify it exists

kubectl create secret generic db2.....

**#MQ**

KLP: You should have already created this. Verify it exists

kubectl create secret generic mq ...

**#Redis**

KLP: You should have already created this. Verify it exists

**#ODM**

KLP: You should have already created this. Verify it exists

kubectl create secret generic odm...

*the helm chart.*

**#Twitter**

KLP: If you setup the Twitter notification scenario, you need this secret, using the tokens from **your personal twitter app.**

kubectl create secret generic twitter --from-literal=consumerKey=<**your twitter app consumer key**> --from-literal=consumerSecret=<**Your twitter app consumer secret**> --from-literal=accessToken=<**Your twitter app access token**> --from-literal=accessTokenSecret=<**Your twitter app token secret**> -n stock-trader

*// This lets use post to your IBM Twitter account.*

*KLP: I created a twitter app in twitter UI. The app was just created so that I could add my own twitter consumer key, consumer secret, access token, and acess token secret, which is all provided in the twitter app. Then I used that info in the twitter secret that I created.*

**#JWT**

KLP: run this command as-is…. We need the secret since the values are referenced in the deployment yaml files. but we will not actually use the tradr service that requires it.

kubectl create secret generic jwt --from-literal=audience=stock-trader --from-literal=issuer=http://stock-trader.ibm.com -n stock-trader

**#Ingress Host**

KLP: run this command as-is, if using the environment we provided. Otherwise modify the host= to match your proxy server IP address

kubectl create secret generic ingress-host --from-literal=host=10.0.0.1:443 -n stock-trader

***// “host=” will be different for your setup. Must use your proxy node IP address.***

**#Watson**

KLP: run this command as-is…. We need the secret since the values are referenced in the deployment yaml files, but we will not actually use this functionality in the app requires it.

kubectl create secret generic watson "--from-literal=url=https://gateway.watsonplatform.net/tone-analyzer/api/v3/tone?version=2017-09-21&sentences=false" --from-literal=id=apikey --from-literal=pwd=4S9tAi1qrNQcIj2\_-PECqK9VaGKqoMIaUGe-gqn5DV8n -n stock-trader

*// This uses the CTO Office IBM Cloud account ‘Standard’ instance. Just run the command above as-is for this environment*

***KLP: I did not do this next step, as I did not care about setting up the Watson Tone analyzer***

*// (how to test Watson connection from Portfolio pod:*

curl 'https://gateway.watsonplatform.net/tone-analyzer/api/v3/tone?version=2017-09-21&sentences=false' -X POST -d '{"text": "Great tool!"}' -u 08af66f3-74fd-43bf-b300-5702a22a27bb:7zgrUP1PbhNZ -H 'Content-Type: application/json'

{"document\_tone":{"tones":[{"score":0.88932,"tone\_id":"joy","tone\_name":"Joy"}]}}

**#OIDC**

KLP: run this command as-is…. We need the secret since the values are referenced in the deployment yaml files, but we will not actually use the tradr service that requires it.

kubectl create secret generic oidc --from-literal=name=IBMid --from-literal=issuer=https://idaas.iam.ibm.com --from-literal=auth=https://idaas.iam.ibm.com/idaas/oidc/endpoint/default/authorize --from-literal=token=https://idaas.iam.ibm.com/idaas/oidc/endpoint/default/token --from-literal=id=ODllNjBlMDgtYzM5NS00 --from-literal=secret=MzhlZTY1ZjItM2IwNC00 --from-literal=key=blueidprod --from-literal=nodeport=https://9.42.103.236:32389 -n stock-trader

// Generally, create this OIDC secret as-is. It’s not actually used if you deploy the default Trader container, which you will do in this exercise. However, the trader deploy.yaml still pulls fields from the secret during the deployment. So these need be available, or you need to remove the fields from the trader deploy.yaml file. **Just easy enough to create this dummy secret.**

# Deploy Stock Trader Pods

KLP: I have pulled the docker images from Dockerhub, and they are located on the VM Workstation (10.0.0.3) machine.

docker images | grep trader

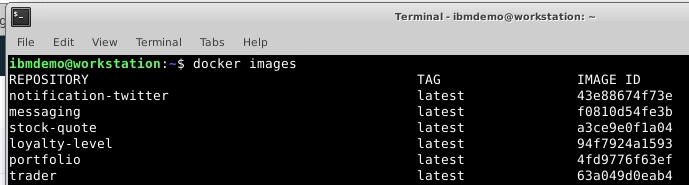
docker images | grep portfolio

docker images | grep stock-quote

docker images | grep messaging

docker images | grep loyalty-level

docker images | grep notification-twitter

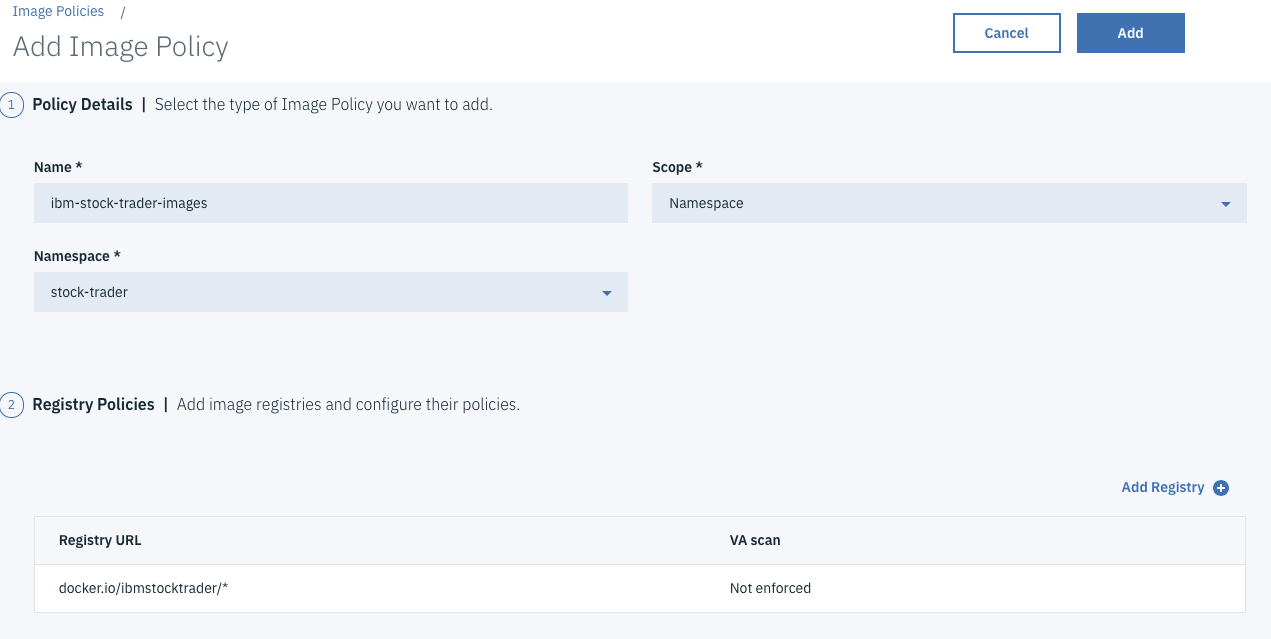


## Prepare:

Add an image policy so that we can pull from IBMStockTrader:

KLP: Next step is not needed if using the yaml provided, as we are pulling the images from ICP repo, rather than dockerhub

Note: I created a new one with the “**docker.io/ibmstocktrader/\***” in it, but I’ve had some problems so you may need to add that URL into the current existing default policy



KLP: You need to tag the images and push images to your ICP image repo, using the following commands. I have already done the maven build of the binaries, and built the docker images.

**Trader service**

~~cd /home/ibmdemo/IBMStockTrader/trader~~

~~mvn package~~

~~docker build -t trader .~~

docker tag trader:latest mycluster.icp:8500/default/trader:latest

docker push mycluster.icp:8500/default/trader:latest

**Portfolio service**

~~cd /home/ibmdemo/IBMStockTrader/portfolio~~

~~mvn package~~

~~docker build -t portfolio .~~

**Repeat the image tag and push in similar way that you did for trader service**

Tag the portfolio:latest for ICP

Push the tagged portfolio image to ICP repo

**Loyalty-Level service**

~~cd /home/ibmdemo/IBMStockTrader/loyalty-level~~

~~mvn package~~

~~docker build -t loyalty-level .~~

**Repeat the image tag and push in similar way that you did for trader service**

Tag the loyalty-level:latest for ICP

Push the tagged loyalty-level image to ICP repo

**Stock-quote service**

~~cd /home/ibmdemo/IBMStockTrader/stock-quote~~

~~mvn package~~

~~docker build -t stock-quote .~~

**Repeat the image tag and push in similar way that you did for trader service**

Tag the stock-quote:latest for ICP

Push the tagged stock-quote image to ICP repo

**messaging service**

~~cd /home/ibmdemo/IBMStockTrader/messaging~~

~~mvn package~~

~~docker build -t messaging .~~

**Repeat the image tag and push in similar way that you did for trader service**

Tag the messaging:latest for ICP

Push the tagged messaging image to ICP repo

**notification-twitter service**

~~cd /home/ibmdemo/IBMStockTrader/notification-twitter~~

~~mvn package~~

~~docker build -t notification-twitter .~~

**Repeat the image tag and push in similar way that you did for trader service**

Tag the notification-twitter:latest for ICP

Push the tagged notification-twitter image to ICP repo

KLP: Change the scope of the images from **namespace** to **global**, for all of the images in ICP.

## Deploy the pods

KLP. On the Workstation VM, go to **/home/ibmdemo/IBMStockTrader**

Deploy ‘trader’:

cd ~home/IBMStockTrader/trader

Modify the .manifests/deploy.yaml to pull the ICP tagged image from the ICP repo

kubectl -n stock-trader create -f ./manifests/deploy.yaml

Deploy ‘portfolio’:

cd ~home/IBMStockTrader/portfolio

Modify the .manifests/deploy.yaml to pull the ICP tagged image from the ICP repo

**KLP: for Portfolio, modify the ./manifests/deploy.yaml, commenting out or removing the kafka volume bits from the yaml. We will not use kafka in this exercise.**

Run the kubectl command to deploy the **portfolio** service resources, in similar way that you did for the trader service

Deploy ‘messaging’:

cd ~home/IBMStockTrader/messaging

Modify the .manifests/deploy.yaml to pull the ICP tagged image from the ICP repo

Run the kubectl command to deploy the **messaging** service resources, in similar way that you did for the trader service

Deploy ‘stock-quote’:

cd ~home/IBMStockTrader/stock-quote

Modify the .manifests/deploy.yaml to pull the ICP tagged image from the ICP repo

Run the kubectl command to deploy the **stock-quote** service resources, in similar way that you did for the trader service

Deploy ‘loyalty-level’:

cd ~home/IBMStockTrader/loyalty-level

Modify the .manifests/deploy.yaml to pull the ICP tagged image from the ICP repo

Run the kubectl command to deploy the **loyalty-level** service resources, in similar way that you did for the trader service

Deploy ‘twitter-notification’:

cd ~home/IBMStockTrader/notification-twitter

Modify the .manifests/deploy.yaml to pull the ICP tagged image from the ICP repo

Run the kubectl command to deploy the **notification-twitter** service resources, in similar way that you did for the trader service

## Find Stock Trader URLs

The “Old” UI is servlet-based, and is helpful to show the ‘before/after’ when showing multiple languages. The “New” UI is node.js based.

### Old “Trader” UI:

The old UI URL can be found by going to “Deployments”, finding “trader” deployment and clicking “Launch”. Depending on what version you use, the URL may be:

* Using NodePort: https://10.0.0.1:<NodePort>/trader

(For you, replace the IP address with your proxy node IP address, and NodePort with the service NodePort)

Login to the Stock Trader app using the following credentials:

**Username:** stock

**Password:** trader

## Validate the solution.

Now, test the application core functionality for adding, updating and deleting portfolio.

Ensure the data is represented, which indicates the database is functional.

Ensure the loyalty-level changes, when portfolio is greater than $50,000, indicating the ODM rules is working.

**Then work out the Twitter scenario.**

I created a Twitter App (no code or functionality) in my twitter acct. This gave me the data needed to create the twitter secret for my acct.

See: <https://github.com/IBMStockTrader/stocktrader>

## Configure Twitter (optional)

If you want to install the stocktrader Twitter notification project, follow these steps.

1. Create a [Twitter app](https://apps.twitter.com/).
2. Open a command window on your workstation. Run the following command.

kubectl create secret generic twitter --from-literal=consumerKey=<CONSUMER\_KEY> --from-literal=consumerSecret=<CONSUMER\_SECRET> --from-literal=accessToken=<ACCESS\_TOKEN> --from-literal=accessTokenSecret=<ACCESS\_TOKEN\_SECRET> -n stocktrader

where <CONSUMER\_KEY>, <CONSUMER\_SECRET>, <ACCESS\_TOKEN> and <ACCESS\_TOKEN\_SECRET> are obtained from Twitter following successful creation of your Twitter app.

When buying a stock results in a loyalty-level change. You will see a message in the logs in the notification-twitter pod, indicating a tweet was successfully sent. Go to your twitter acct, and you will see the new tweet arrive.

