Quick Start guide to IBM Cloud Pak for Integration on RHOS 3.11

Featuring

ACE

ACE+MQ

MQ

APIC

Event Streams

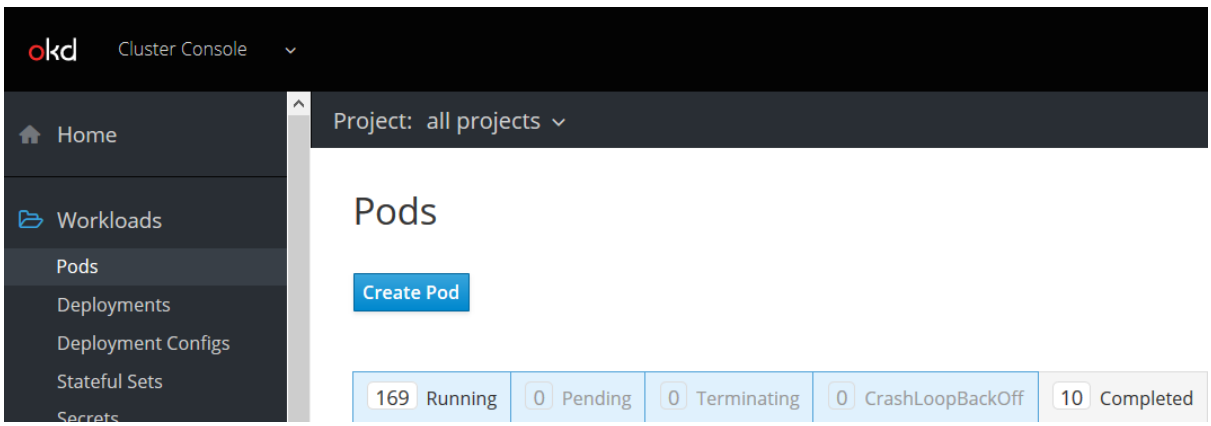
Contents

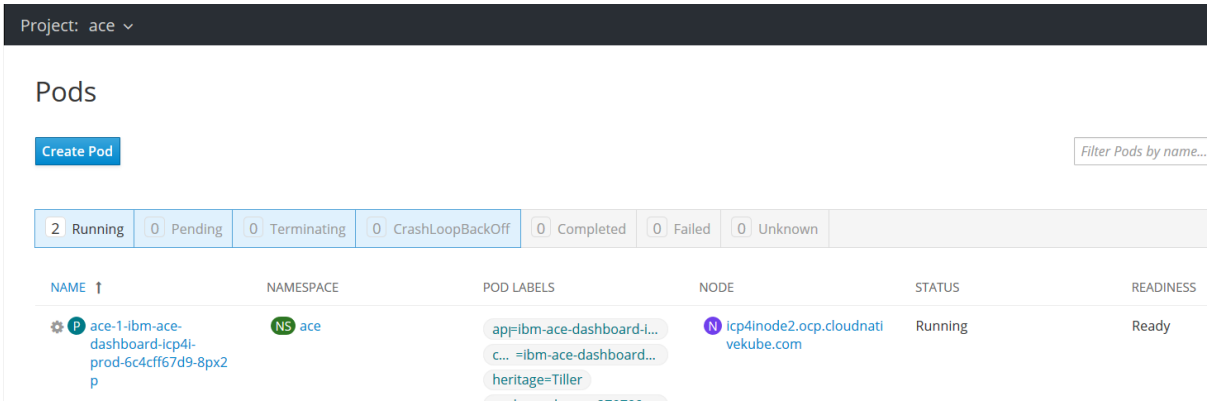
IBM Cloud Pak for Integration – Target Environment

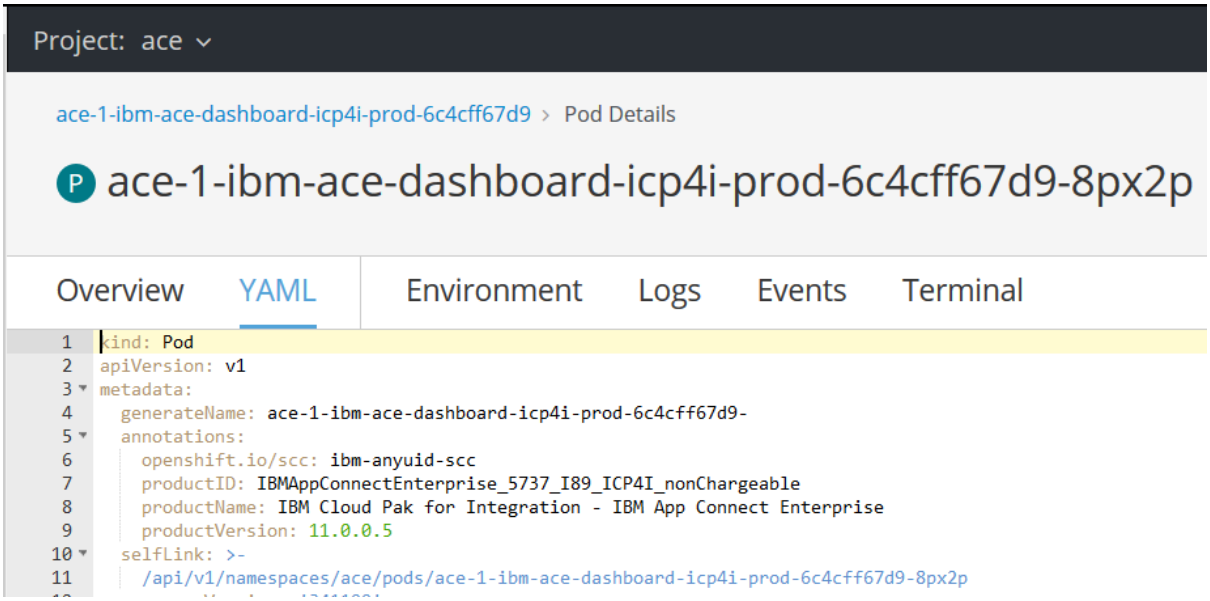
IBM Cloud – IBM ANZ Tech Sales instance – Public IPs

RHOS (OKD) Access point and navigation

Openshift: https://master.ocp.cloudnativekube.com:8443/ (admin/#######)







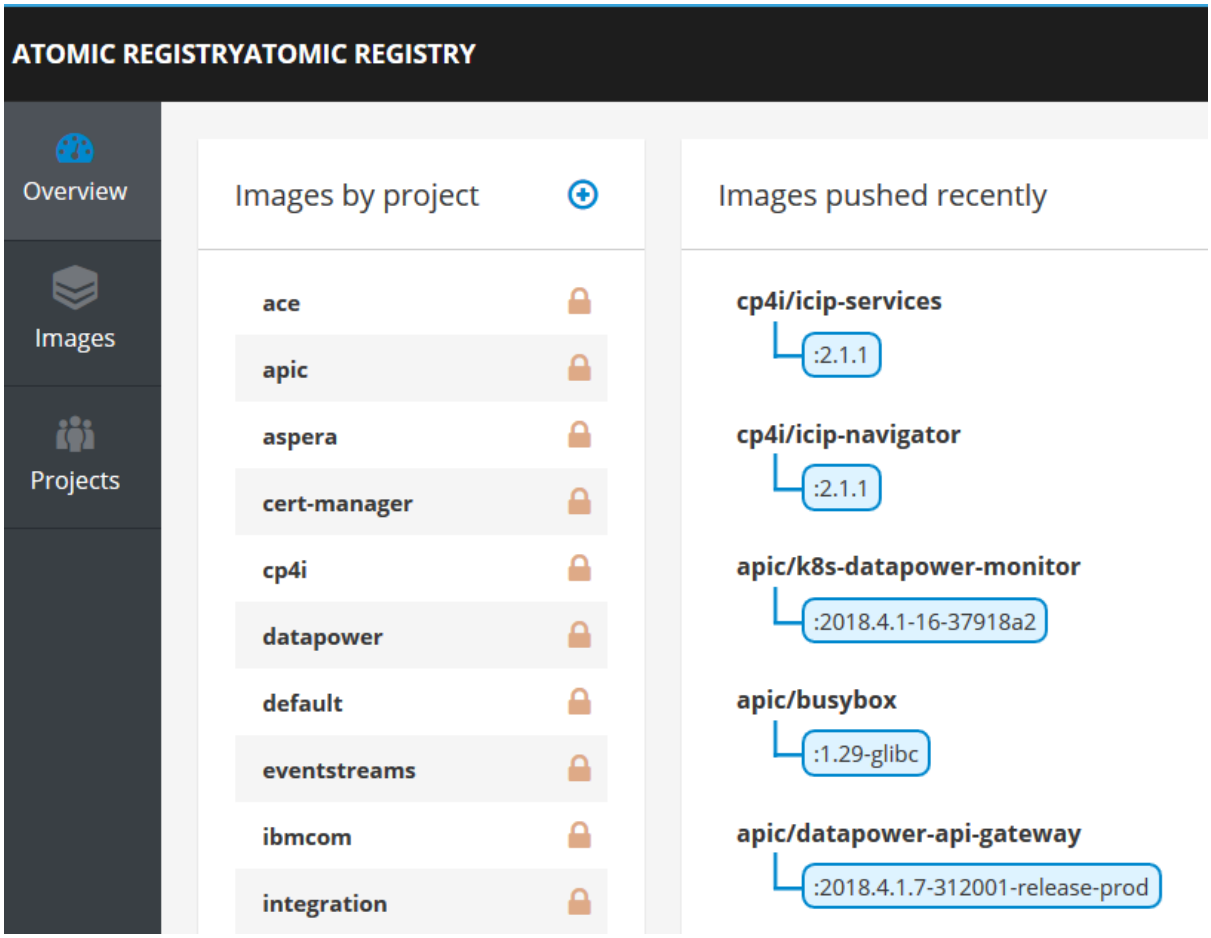
Getting key information from RHOS(OKD)

```
C:\openshift>oc login -u system:admin -n default
Authentication required for https://master.ocp.cloudnativekube.com:8443 (openshift)
Username: system:admin
Password:
```

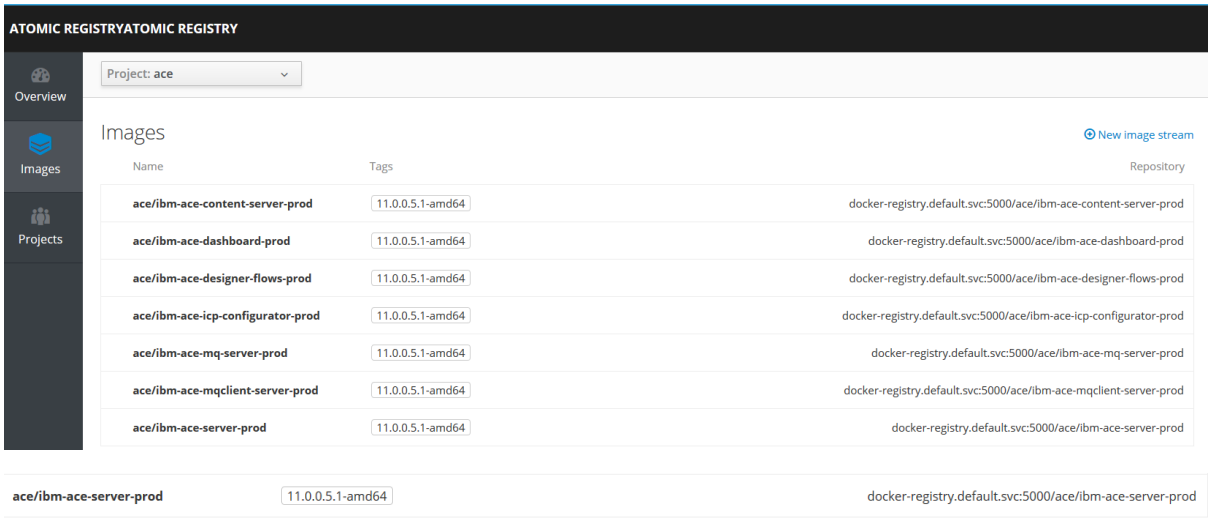
Get the onboard docker image registry

```
C:\openshift>oc get route -n default
NAME               HOST/PORT                                                PATH      SERVICES
docker-registry    docker-registry-default.apps.ocp.cloudnativekube.com               docker-registry
registry-console   registry-console-default.apps.ocp.cloudnativekube.com               registry-console
```

Goto the registry console URL
https://registry-console-default.apps.ocp.cloudnativekube.com/registry



Click on ace



For example this is the image in the onboard registry for an ace server (no MQ client no MQ server)

docker-registry.default.svc:5000/ace/ibm-ace-server-prod

Get the pull secret

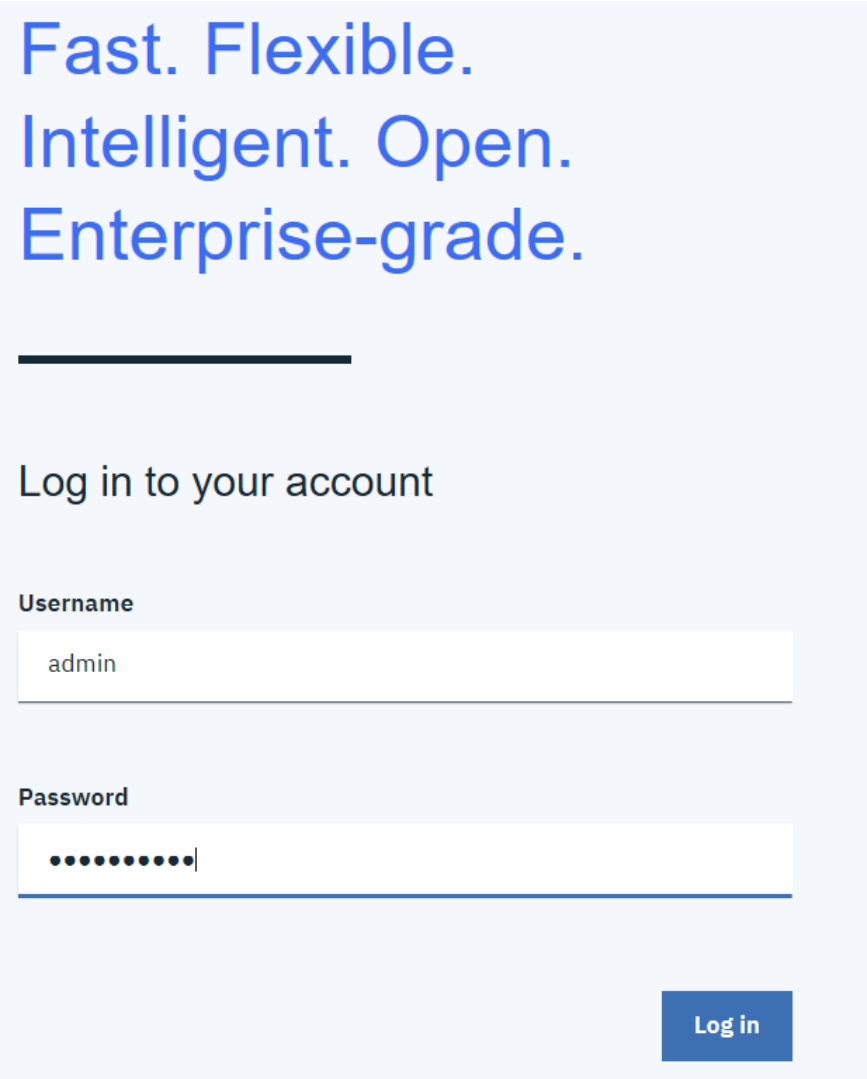
```
C:\openshift>oc get secrets
NAME                                                       TYPE                                  DATA      AGE
abace-ibm-ace-server-icp4i-prod-handle-384cc241-dockercfg-qxhzn   kubernetes.io/dockercfg               1         11m
abace-ibm-ace-server-icp4i-prod-handle-sec-384cc241-token-hrkbl   kubernetes.io/service-account-token   4         11m
abace-ibm-ace-server-icp4i-prod-handle-secret-serviceaccou4bx4f   kubernetes.io/service-account-token   4         11m
ace-1-ibm-ace-dashboard-icp4i-prod                         Opaque                                10        1d
ace-1-ibm-ace-dashboard-icp4i-prod-oidc                    Opaque                                2         1d
ace-1-ibm-ace-dashboard-icp4i-prod-ser-b89c56e3-dockercfg-5fpdj   kubernetes.io/dockercfg               1         1d
ace-1-ibm-ace-dashboard-icp4i-prod-serviceaccount-token-m6k9k   kubernetes.io/service-account-token   4         1d
ace-1-ibm-ace-dashboard-icp4i-prod-serviceaccount-token-sblgc   kubernetes.io/service-account-token   4         1d
builder-dockercfg-fz6xk                                    kubernetes.io/dockercfg               1         2d
builder-token-md65l                                        kubernetes.io/service-account-token   4         2d
builder-token-s61d9                                        kubernetes.io/service-account-token   4         2d
da-ace-echo-ibm-ace--623f-handle--6389-dockercfg-xbk6c     kubernetes.io/dockercfg               1         1h
da-ace-echo-ibm-ace--623f-handle--6389-token-4b9t6         kubernetes.io/service-account-token   4         1h
da-ace-echo-ibm-ace--623f-handle--6389-token-d9b6q         kubernetes.io/service-account-token   4         1h
daace-ibm-ace-server-icp4i-prod-handle-f5f55a05-dockercfg-9bfb4   kubernetes.io/dockercfg               1         56m
daace-ibm-ace-server-icp4i-prod-handle-sec-f5f55a05-token-jh7rn   kubernetes.io/service-account-token   4         56m
daace-ibm-ace-server-icp4i-prod-handle-secret-serviceaccouzb764   kubernetes.io/service-account-token   4         56m
default-dockercfg-8lm7v                                    kubernetes.io/dockercfg               1         2d
default-token-79zd2                                        kubernetes.io/service-account-token   4         2d
default-token-zlxdk                                        kubernetes.io/service-account-token   4         2d
deployer-dockercfg-k8tdp                                   kubernetes.io/dockercfg               1         2d
deployer-token-gcwbp                                       kubernetes.io/service-account-token   4         2d
deployer-token-zh6kj                                       kubernetes.io/service-account-token   4         2d
```

Get the Network file system

```
C:\SHARE\ProductsTechnology\IBM Cloud Pak for Integration\SetUp\yamls>\openshift\oc get sc
NAME                          PROVISIONER   AGE
managed-nfs-storage (default)   nfs-storage   2d
```

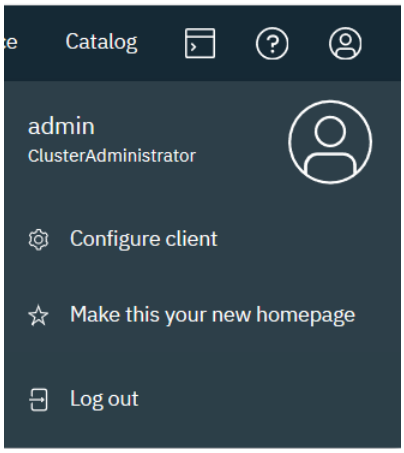
ICP4i via ICP Access point and navigation

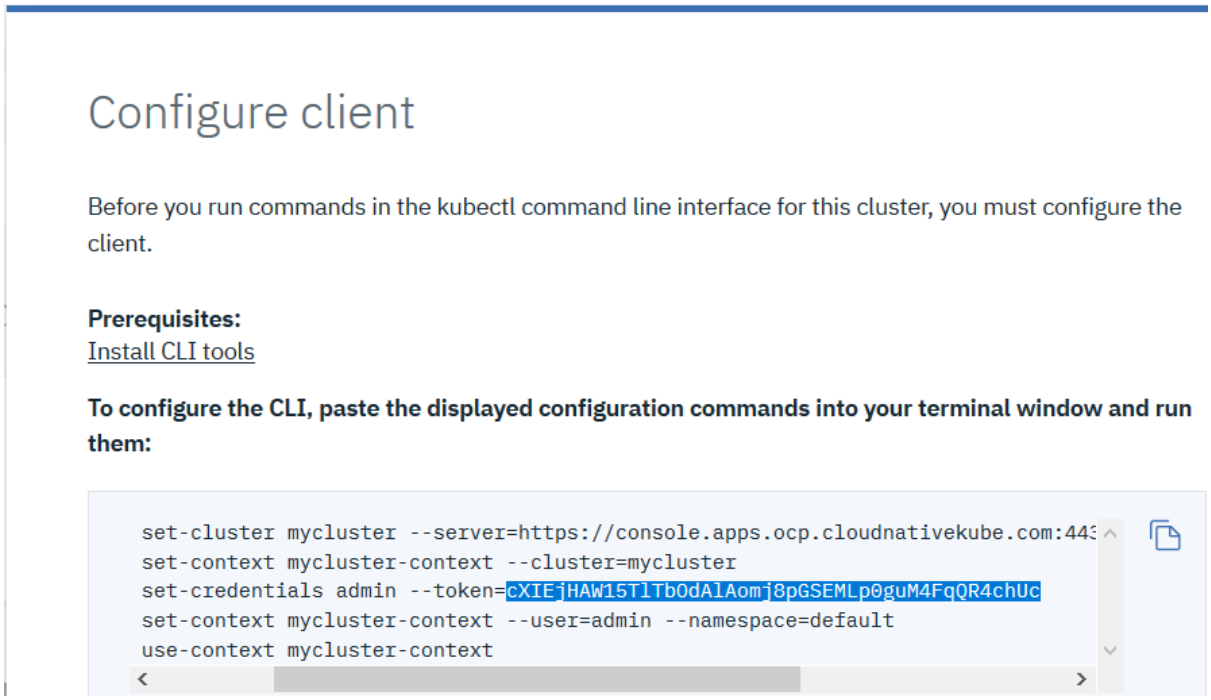
ICP on Openshift: https://icp-console.apps.ocp.cloudnativekube.com (admin/#######)



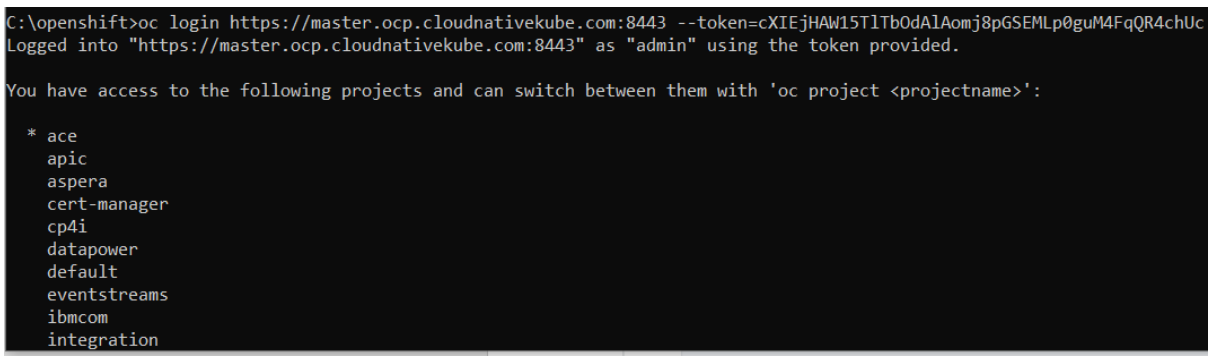
Key Information from ICP

ICP login information for kubectl and oc commands



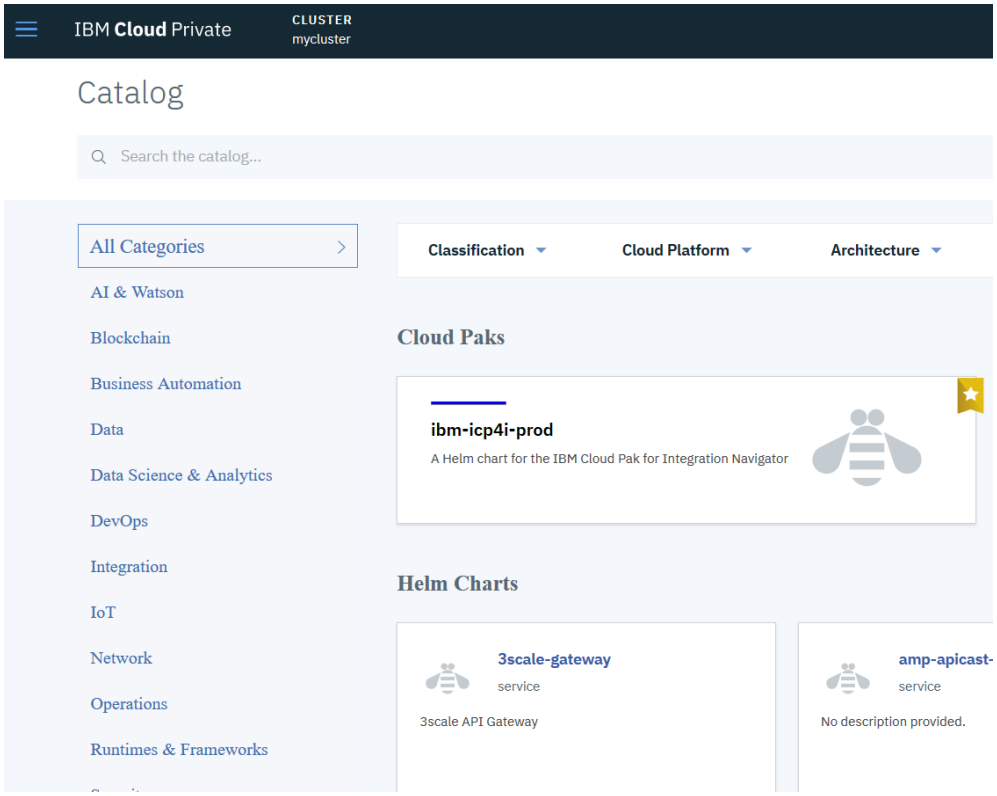


The token can be used on the oc command line interface

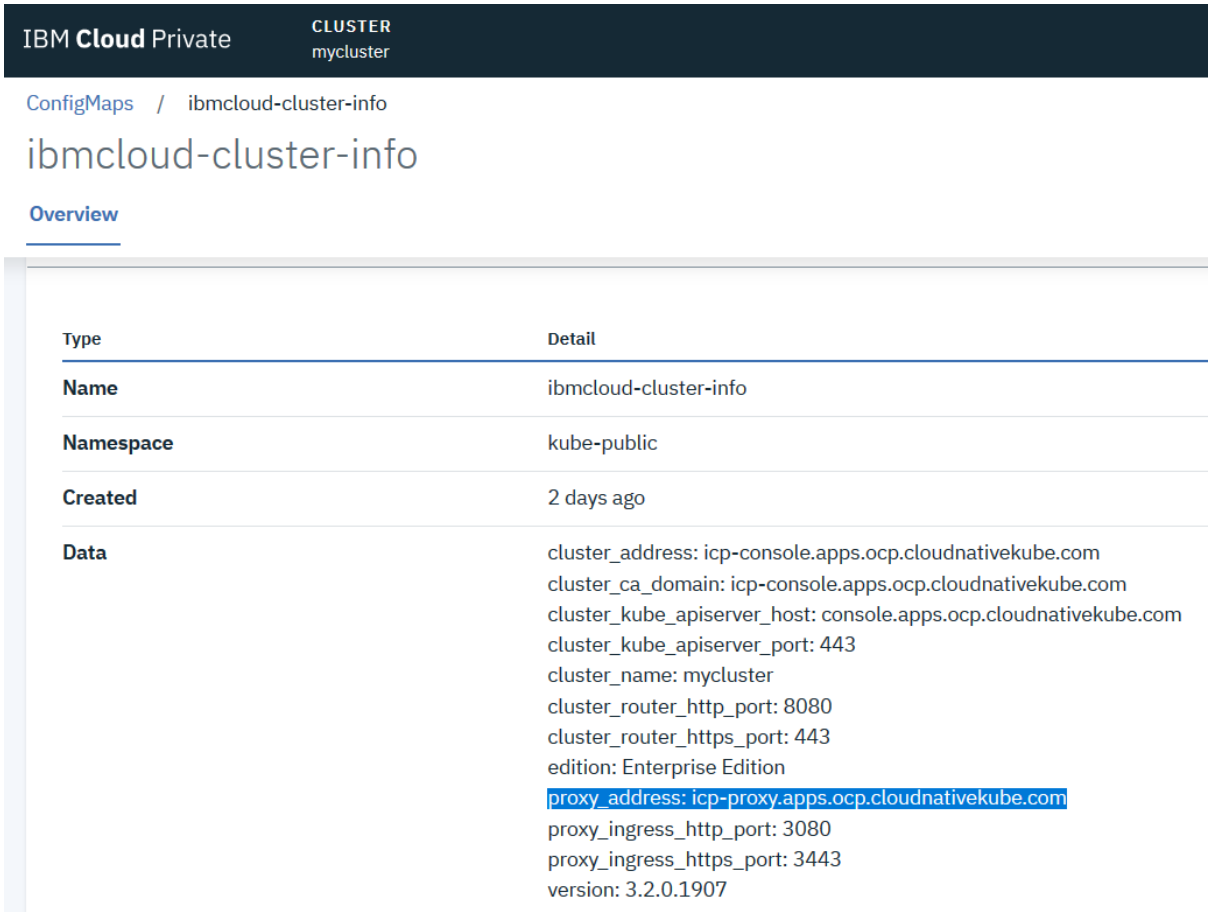


ICP Catalog

https://icp-console.apps.ocp.cloudnativekube.com/catalog/

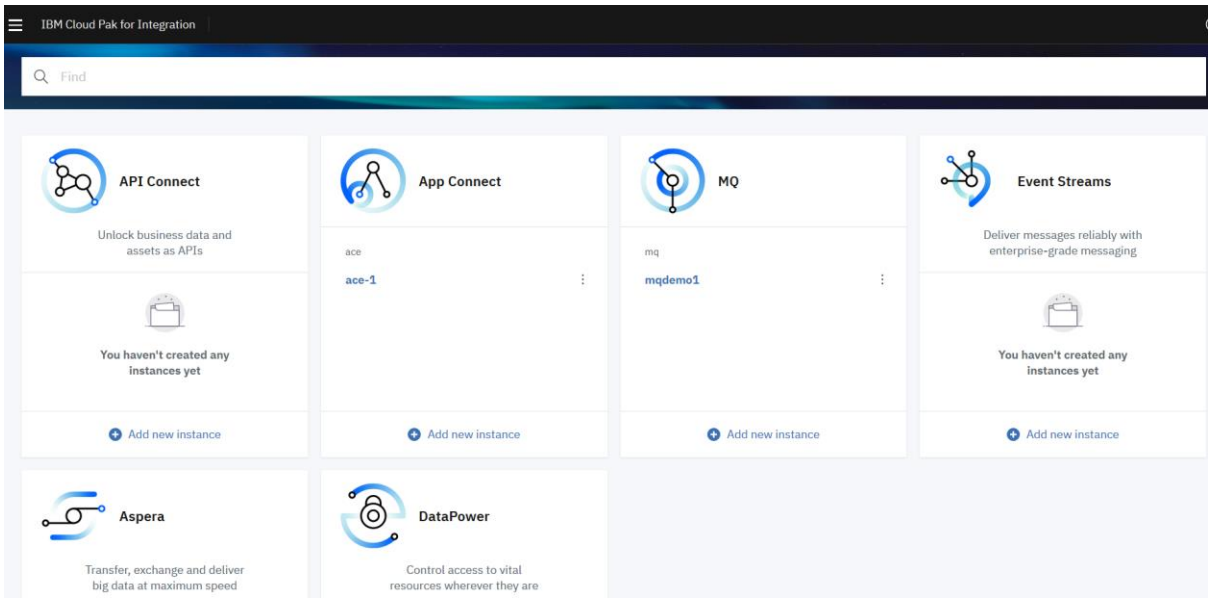


ICP4i proxy address



ICP4i console

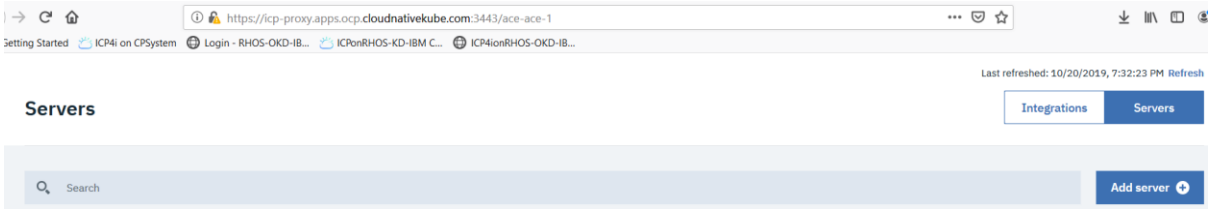
https://icp-proxy.apps.ocp.cloudnativekube.com:3443/integration



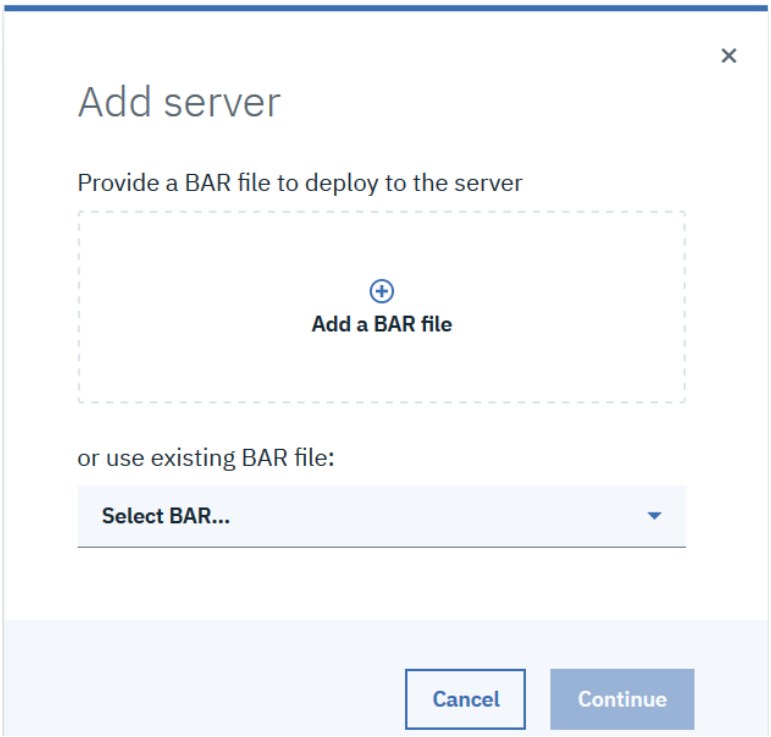
Click on ace-1 – for the ICP4i ACE dashboard

ACE on ICP4i on RHOS via the ICP4i Navigator

Add an ace server and associate a BAR

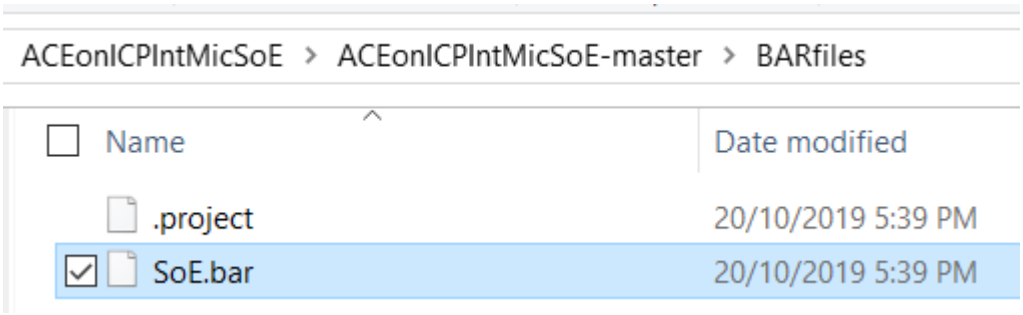


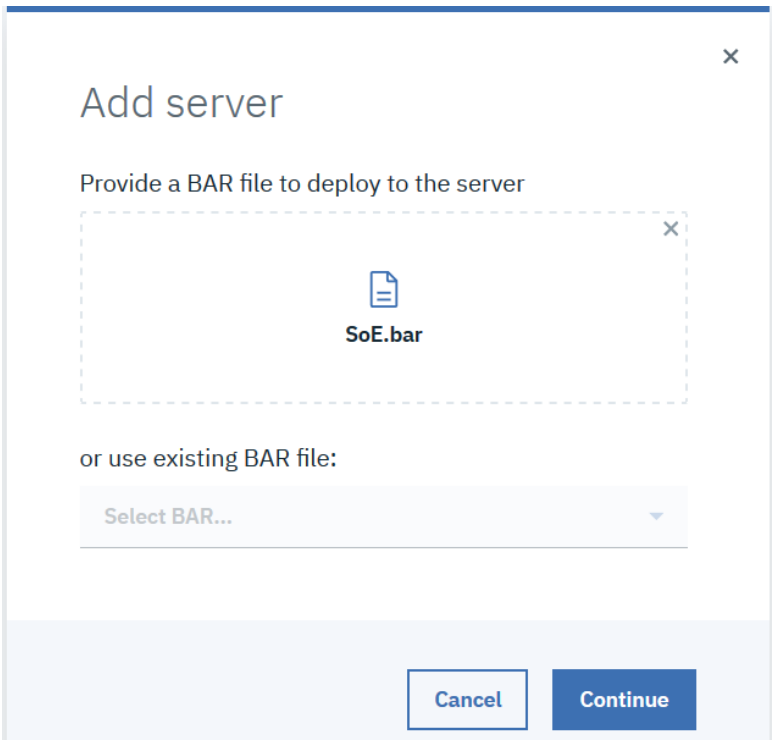
Add a server

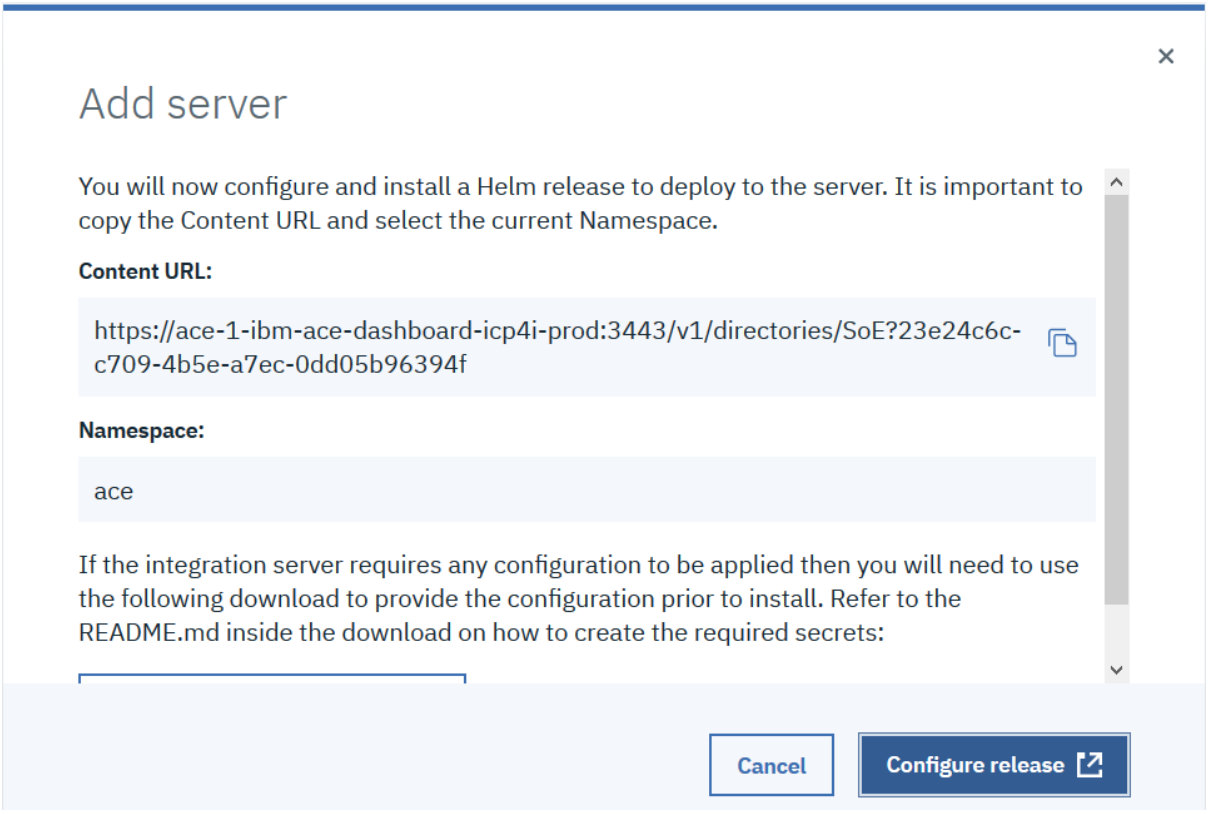


Navigate to a bar

https://github.com/DAVEXACOM/ACEonICPIntSupportingMaterial/tree/master/ace-livelinessProbe

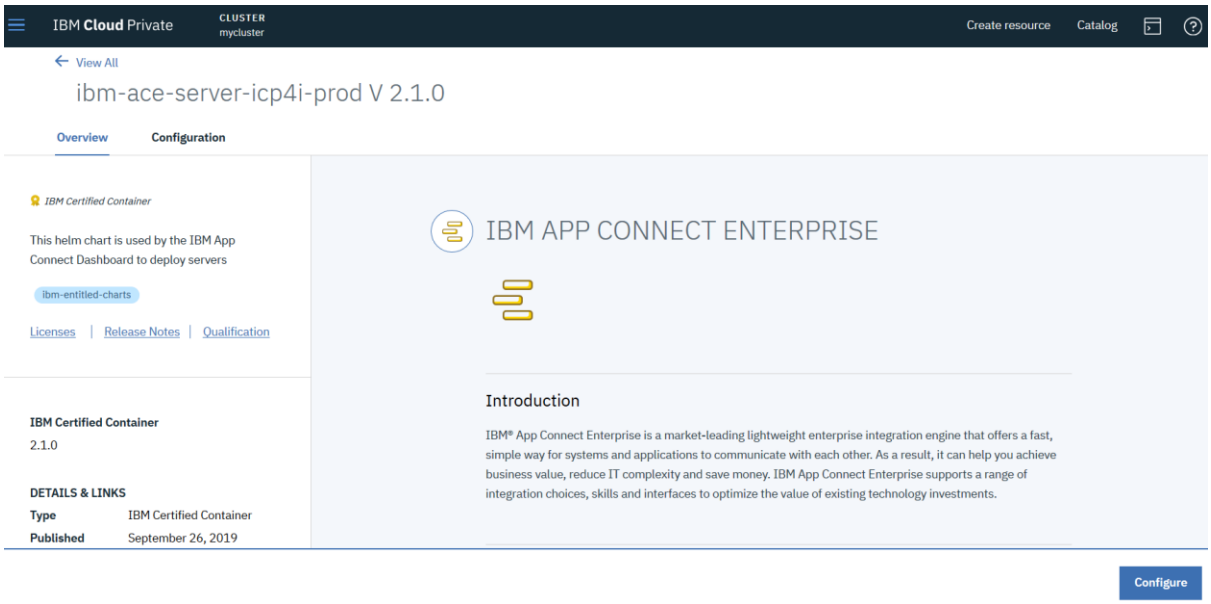






Take a copy of the url to the BAR (content) file

https://ace-1-ibm-ace-dashboard-icp4i-prod:3443/v1/directories/SoE?23e24c6c-c709-4b5e-a7ec-0dd05b96394f



Information you'll need to configure the release

Collected information you need to configure the release.

ICP4i Proxy address (from ICP configMaps): icp-proxy.apps.ocp.cloudnativekube.com

Content URL (from ICP4i): https://ace-1-ibm-ace-dashboard-icp4i-prod:3443/v1/directories/SoE?23e24c6c-c709-4b5e-a7ec-0dd05b96394f

onboard docker image registry (from oc command):

example for ace with MQ server

`docker-registry.default.svc:5000/ace/ibm-ace-mq-server-prod`

docker-registry.default.svc:5000/ace/ibm-ace-mq-server-prod:11.0.0.5.1

example for ace with no MQ

`docker-registry.default.svc:5000/ace/ibm-ace-server-prod`

docker-registry.default.svc:5000/ace/ibm-ace-server-prod:11.0.0.5.1

replace cp.icr.io with docker-registry.default.svc:5000/ace in the configuration

pull secret(from oc command):

deployer-dockercfg-k8tdp

Managed NFS name (from oc command):

managed-nfs-storage

Configuring the ACE release

Warnings

These are not a problem – see appendix on PSP

⚠ **Pod Security Conflict** This chart requires a namespace with a ibm-anyuid-psp pod security policy.

⚠ **Pod Security Warning** Your ICP cluster is running all namespaces Unrestricted (ibm-anyuid-hostpath-psp) by default. This could pose a security risk.

Configuration parameters

Configuration

This helm chart is used by the IBM App Connect Dashboard to deploy servers. Edit these parameters for configuration.

Helm release name *

my-ace-rel

Target namespace *

ace

Target Cluster *

local-cluster

License *

I have read and agreed to the License agreement

Parameters

To install this chart, additional configuration is needed in Quick start. To customize installation, view and edit All parameters.

Quick start

Required and recommended parameters to view and edit.

Content Server URL *

https://ace-1-ibm-ace-dashboard-icp4i-prod:3443/v1/directories/SoE?23e24c6c-c709-4b5e-a7ec-0dd05b96394f

Service

Service settings

Proxy Node IP or FQDN *

icp-proxy.apps.ocp.cloudnativekube.com

Production usage

Which type of image to run

App Connect Enterprise only

Architecture scheduling preference *

amd64

Enable IBM App Connect Designer flows

Images

Define images to be used

Docker image for App Connect Enterprise *

docker-registry.default.svc:5000/ace/ibm-ace-server-prod:11.0.0.5.1

Docker image for App Connect Enterprise with MQ client *

cp.icr.io/ibm-ace-mqclient-server-prod:11.0.0.5.1

Docker image for App Connect Enterprise with MQ server *

cp.icr.io/ibm-ace-mq-server-prod:11.0.0.5.1

Configurator Docker image *

cp.icr.io/ibm-ace-icp-configurator-prod:11.0.0.5.1

Image pull secret

deployer-dockercfg-k8tdp

Service

Service settings

Service type *

NodePort

Web UI Port *

7600

HTTP port *

7800

HTTPS port *

7843

Switch AgentC Port

Enter value

Switch AgentP Port

Enter value

Switch Admin Port

Enter value

Proxy Node IP or FQDN *

icp-proxy.apps.ocp.cloudnativekube.com

Integration Server

Define configuration for the Integration Server

Integration server name

myintsrv

Data persistent volume claims (PVCs)

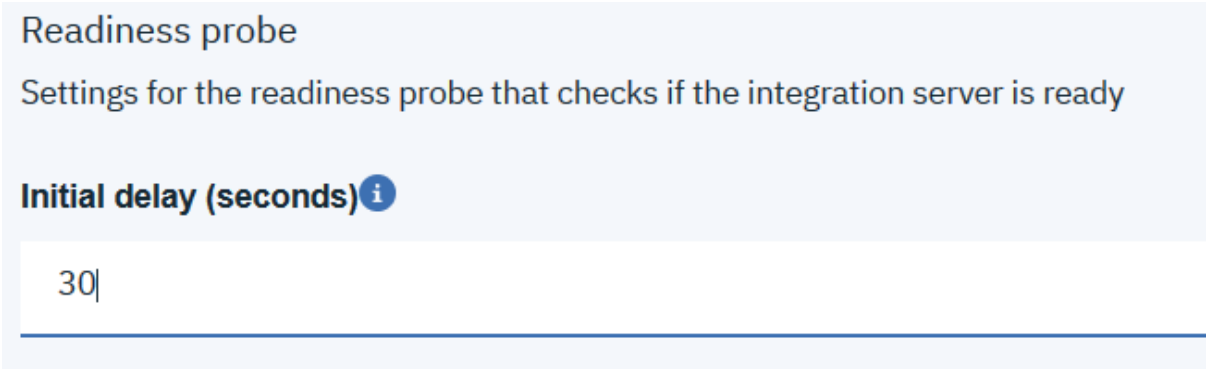
Settings for the PVCs (applicable only when running with a queue manager)

Name *

data

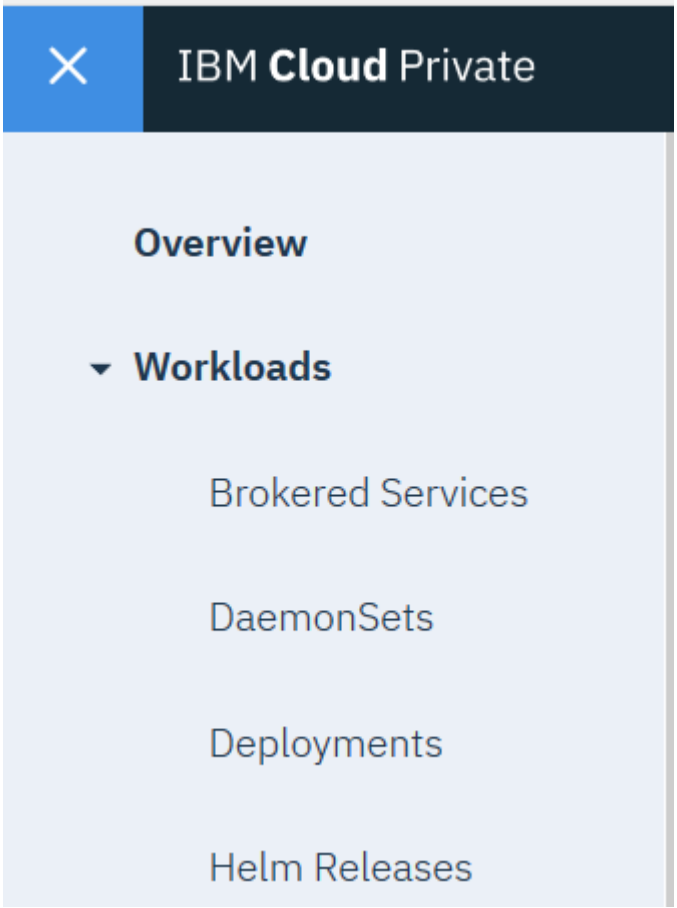
Storage class name

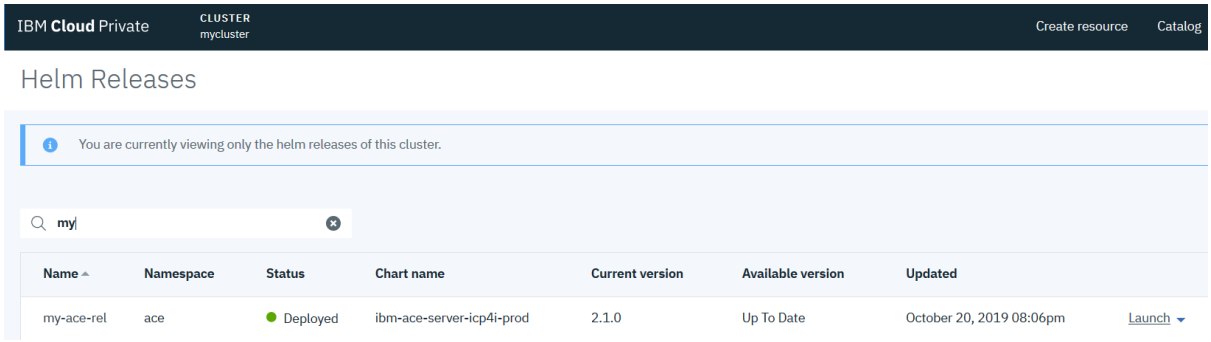
managed-nfs-storage

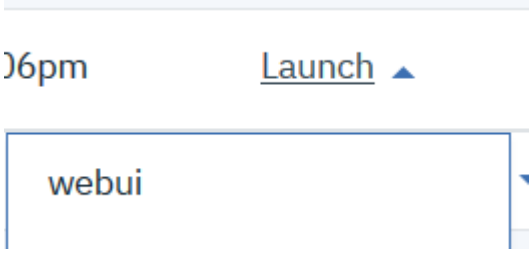


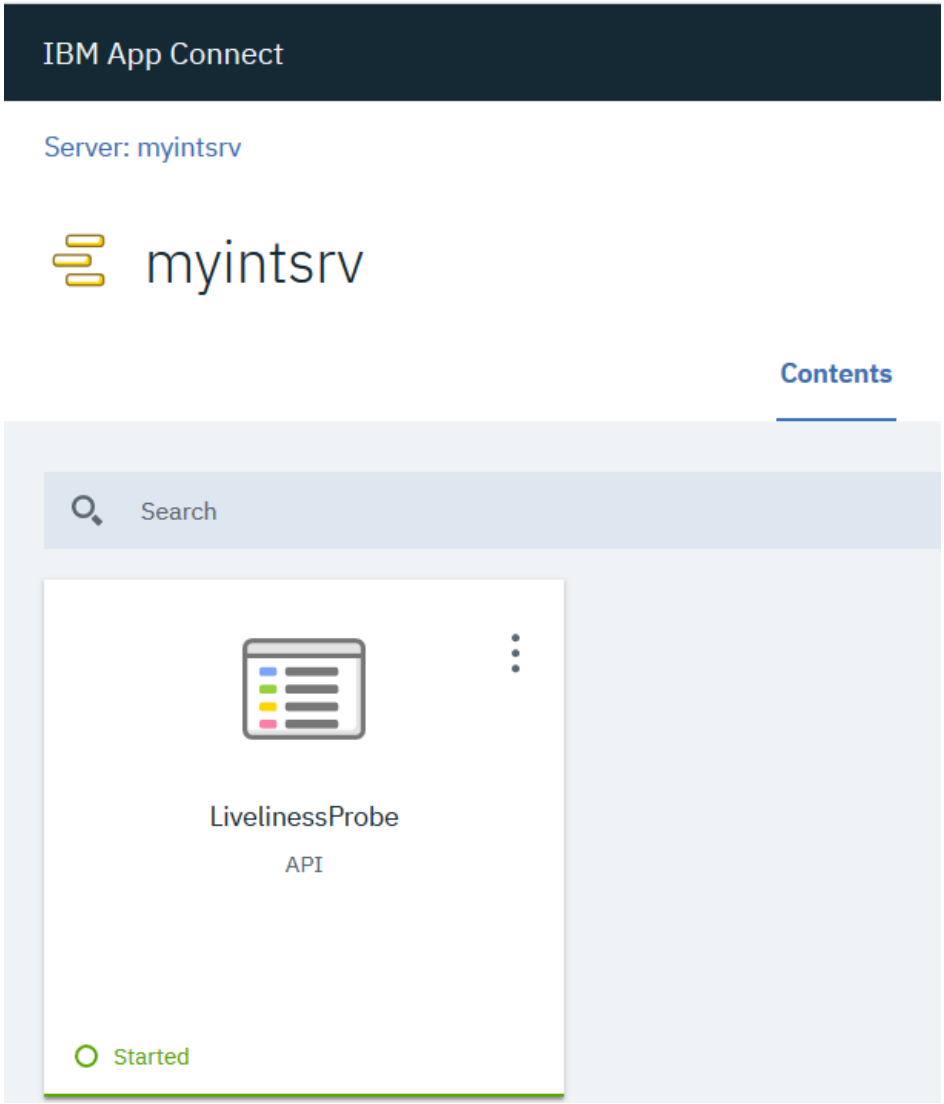
Might have to wait a little while – then goto helm releases

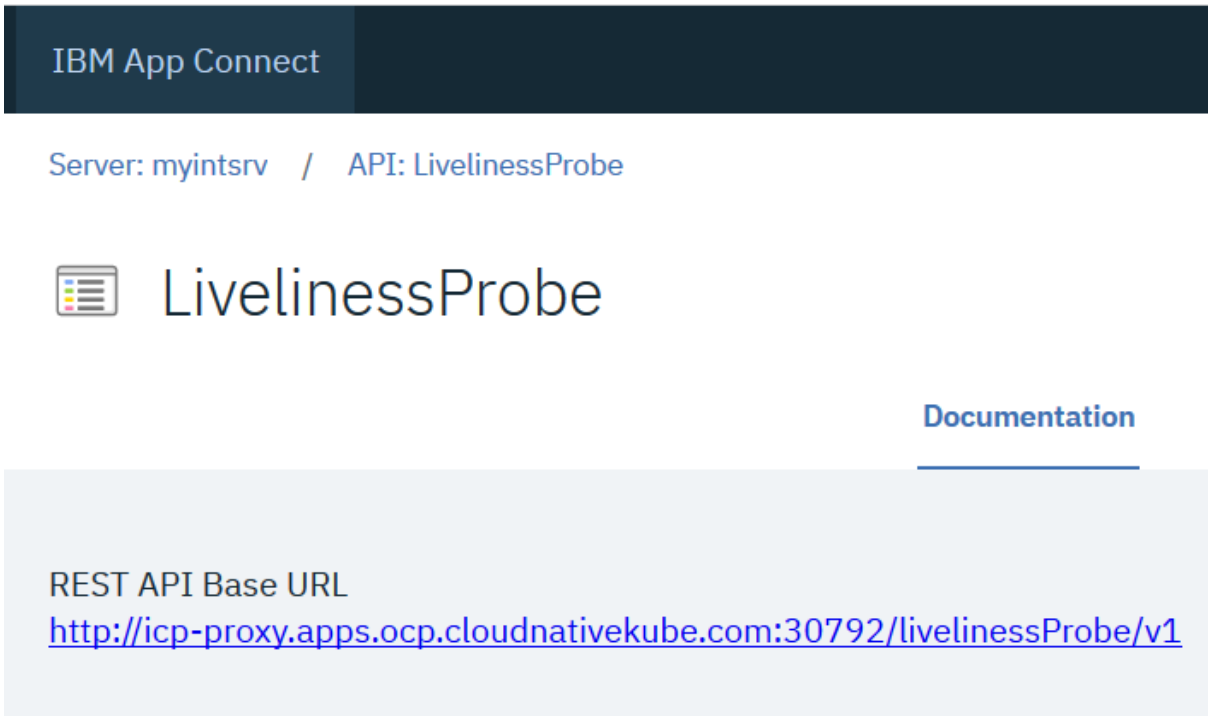
Observing the ACE deployment via ICP4i



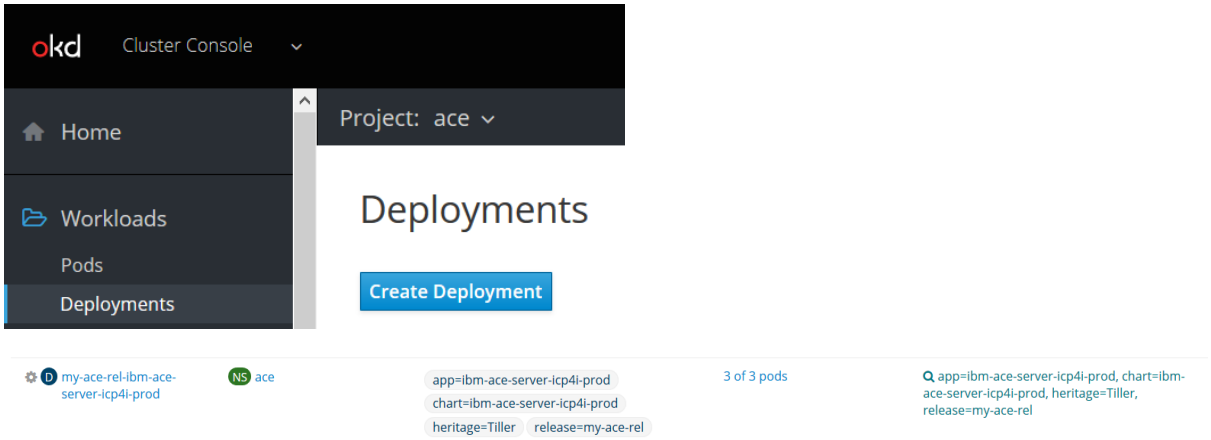

IBM Cloud Private
CLUSTER
mycluster
Create resource
Catalog
Helm Releases
You are currently viewing only the helm releases of this cluster.
my
Name
Namespace
Status
Chart name
Current version
Available version
Updated
my-ace-rel
ace
Deployed
ibm-ace-server-icp4i-prod
2.1.0
Up To Date
October 20, 2019 08:06pm
Launch

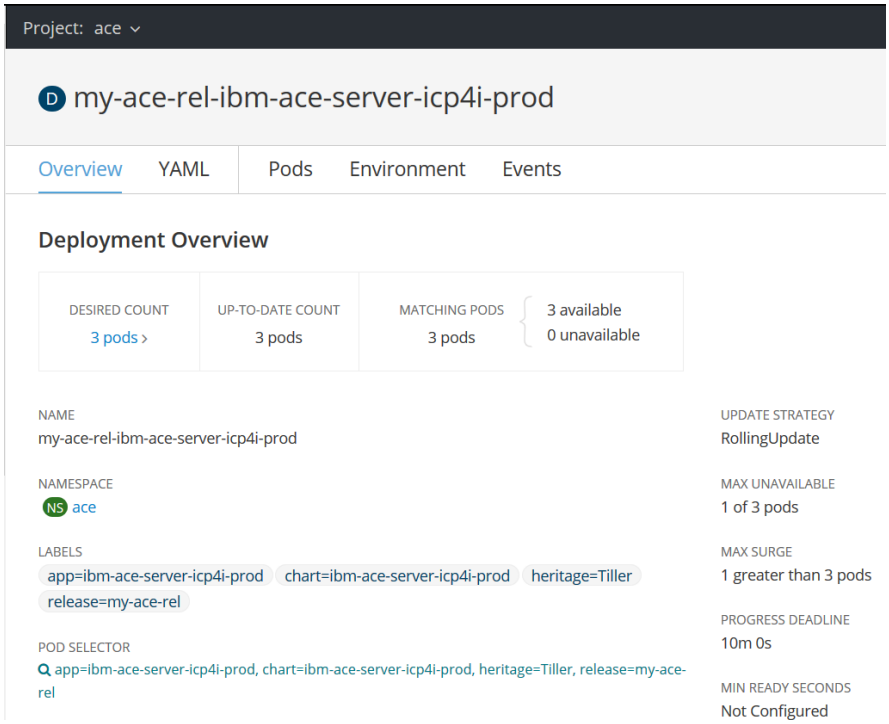

)6pm
Launch
webui

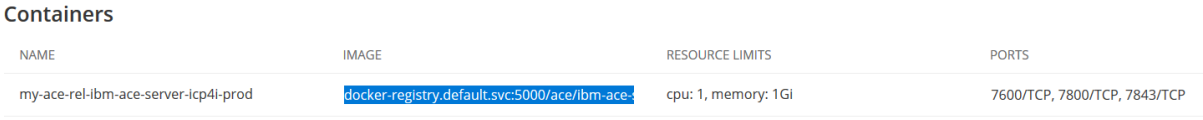

IBM App Connect
Server: myintsrv
myintsrv
Contents
Search
LivelinessProbe
API
Started

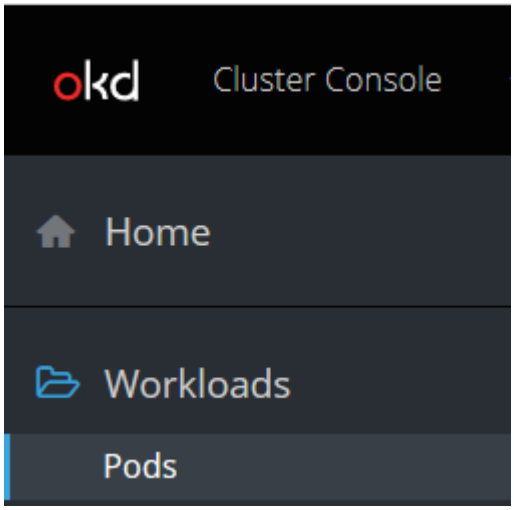


Observing the ACE deployment via RHOS




Project: ace
my-ace-rel-ibm-ace-server-icp4i-prod
Overview
YAML
Pods
Environment
Events
Deployment Overview
DESIRED COUNT
3 pods
UP-TO-DATE COUNT
3 pods
MATCHING PODS
3 pods
3 available
0 unavailable
NAME
my-ace-rel-ibm-ace-server-icp4i-prod
NAMESPACE
ace
LABELS
app=ibm-ace-server-icp4i-prod
chart=ibm-ace-server-icp4i-prod
heritage=Tiller
release=my-ace-rel
POD SELECTOR
app=ibm-ace-server-icp4i-prod, chart=ibm-ace-server-icp4i-prod, heritage=Tiller, release=my-ace-rel
UPDATE STRATEGY
RollingUpdate
MAX UNAVAILABLE
1 of 3 pods
MAX SURGE
1 greater than 3 pods
PROGRESS DEADLINE
10m 0s
MIN READY SECONDS
Not Configured


Containers
NAME
IMAGE
RESOURCE LIMITS
PORTS
my-ace-rel-ibm-ace-server-icp4i-prod
docker-registry.default.svc:5000/ace/ibm-ace-
cpu: 1, memory: 1Gi
7600/TCP, 7800/TCP, 7843/TCP


okd
Cluster Console
Home
Workloads
Pods

my-ace-rel-ibm-ace-server-icp4i-prod-66cf9c496c-gzwll	ace	a... =ibm-ace-server-icp4... ch... =ibm-ace-server-icp... heritage=Tiller pod-templat... =227957... release=my-ace-rel	icp4inode2.ocp.cloudnativekube.com	Running	Ready
my-ace-rel-ibm-ace-server-icp4i-prod-66cf9c496c-j4n4f	ace	a... =ibm-ace-server-icp4... ch... =ibm-ace-server-icp... heritage=Tiller pod-templat... =227957... release=my-ace-rel	icp4inode.ocp.cloudnativekube.com	Running	Ready
my-ace-rel-ibm-ace-server-icp4i-prod-66cf9c496c-znzkb	ace	a... =ibm-ace-server-icp4... ch... =ibm-ace-server-icp... heritage=Tiller pod-templat... =227957... release=my-ace-rel	icp4inode3.ocp.cloudnativekube.com	Running	Ready

Project: ace

my-ace-rel-ibm-ace-server-icp4i-prod-66cf9c496c › Pod Details

my-ace-rel-ibm-ace-server-icp4i-prod-66cf9c496c-gzwll

Overview | YAML | Environment | Logs | Events | Terminal

```
142    ports:
143      - name: webui
144        containerPort: 7600
145        protocol: TCP
146      - name: ace-http
147        containerPort: 7800
148        protocol: TCP
149      - name: ace-https
150        containerPort: 7843
151        protocol: TCP
152    imagePullPolicy: IfNotPresent
153    volumeMounts:
154      - name: webusers
155        mountPath: /home/aceuser/initial-config/webusers
156      - name: odbcini
157        mountPath: /home/aceuser/initial-config/odbcini
158      - name: policy
159        mountPath: /home/aceuser/initial-config/policy
160      - name: serverconf
161        mountPath: /home/aceuser/initial-config/serverconf
162      - name: setdbparms
163        mountPath: /home/aceuser/initial-config/setdbparms
```

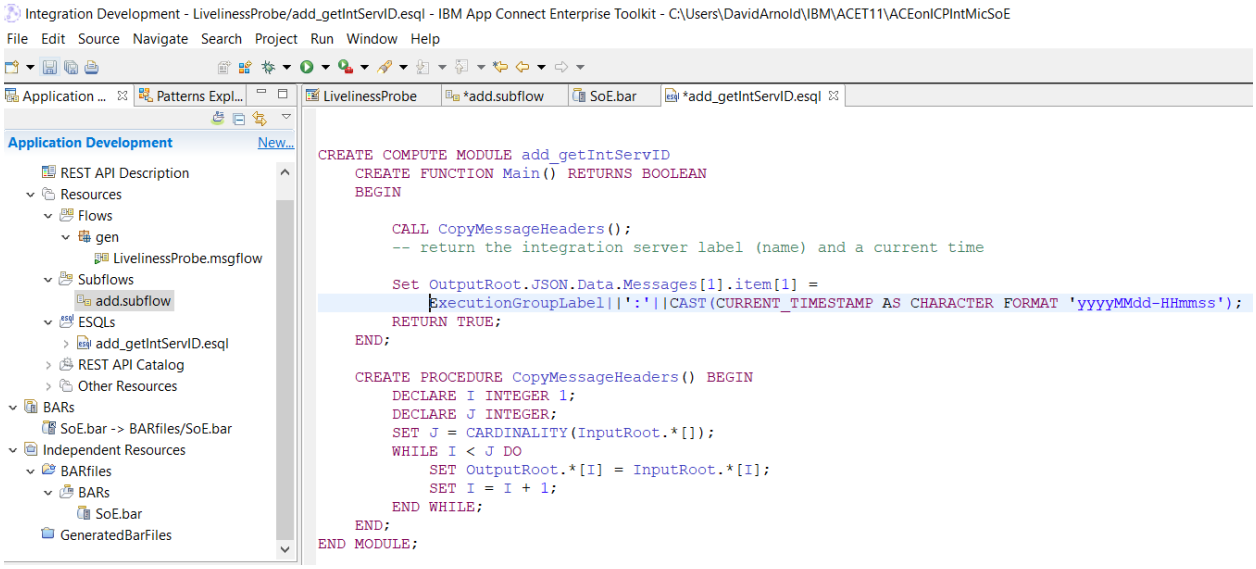
ACE Integration Liveliness Probe

Github Source Repos for ACE Liveliness Probe (The SoE ACE project)

https://github.com/DAVEXACOM/ACEonICPIntMicSoE

Description

The Liveliness Probe Service is an other restful service (this is not the service baked into the cloud pak images). Its just a user echo test service



Testing Liveliness Probe

Using RESTED rest client for firefox I this example

Base URI

http://icp-proxy.apps.ocp.cloudnativekube.com:30792/livelinessProbe/v1

input data:

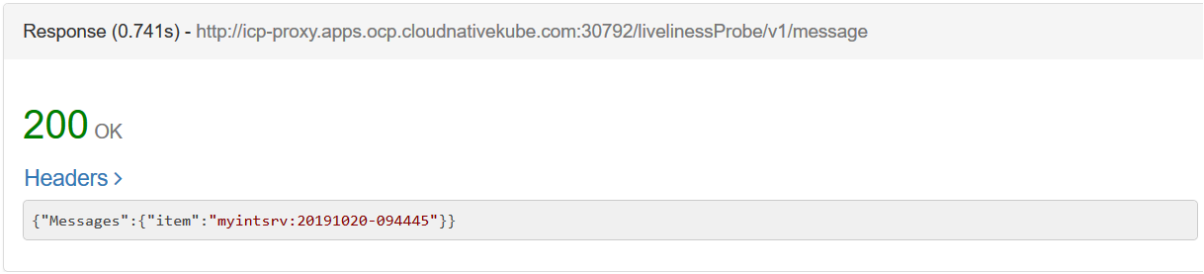
{“Messages”:[“test”]}

Return data

returns the integration server name plus a current timestamp for

Post

http://icp-proxy.apps.ocp.cloudnativekube.com:30792/livelinessProbe/v1/message



ACE and ACE/MQ Instances with additional config via secrets

Overview

It is possible to create additional configuration items and push these into a secret which the ACE deployment will introspect and use for configuration when the container is created. In this way MQSC scripts and many ACE configuration tasks can be executed automatically when the helm release is deployed.

List of configurable items

The items listed below are configurable via a secret:

File	Description
adminPassword.txt	MQ Developer defaults - administrator password
appPassword.txt	MQ Developer defaults - app password
mqsc.txt	An mqsc file to run against the Queue Manager
keystorePassword	A password to set for the Integration Server's keystore
keystore-{keyname}.pass	The passphrase for the private key being imported, if there is one
keystore-{keyname}.key	The private key in PEM format
keystore-{keyname}.crt	The certificate in PEM format
truststorePassword.txt	A password to set for the Integration Server's truststore
truststore-{certname}.crt	The trust certificate in PEM format
odbc.ini	An odbc.ini file for the Integration Server to define any ODBC data connections
policy.xml	Policies to apply
policyDescriptor.xml	The policy descriptor file
serverconf.yaml	The server.conf.yaml
setdbparms.txt	Multi-line file containing the {ResourceName} {UserId} {Password} to pass to mqsisetdbparms command
adminusers.txt	Multi-line value containing the {UserName} {Password} to pass to mqsiwebuseradmin command to create viewer users with READ, WRITE, EXECUTE access to the server

File	Description
viewerusers.txt	Multi-line value containing the {UserName} {Password} to pass to mqsiwebuseradmin command to create admin users with READ access to the server

Running the script to generate the secret

To run the script:

- The kubectl command must be installed and available
- The kubectl environment must be configured to point to the intended target cluster
- The name of the configuration secret to be created must be supplied as an argument on the command line
- The name of RELEASE may be provided as a second argument on the command line if admin or viewer users are required

The script will look for any files detailed in the Configuration section in the same directory and if found will add them into keys in the generated secret. The script will then use the kubectl command to generate a secret containing the specified configuration for the integration server and a second secret containing the user credentials if the second argument of RELEASE NAME is specified.

The script is shown below:

----------begin snippet----------

```
#!/bin/bash
#
# Licensed Materials - Property of IBM
#
# 5737-H33
#
# (C) Copyright IBM Corp. 2018  All Rights Reserved.
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.


TARGET_SECRET_NAME=$1
#########################################################
# Build up the arguments for the kubectl command
#########################################################
```

```
SECRET_ARGS=
if [ -s ./mqsc.txt ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=mqsc=./mqsc.txt"
else
  SECRET_ARGS="${SECRET_ARGS} --from-literal=mqsc="
fi
if [ -s ./adminPassword.txt ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=adminPassword=./adminPassword.txt"
else
  SECRET_ARGS="${SECRET_ARGS} --from-literal=adminPassword="
fi
if [ -s ./appPassword.txt ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=appPassword=./appPassword.txt"
else
  SECRET_ARGS="${SECRET_ARGS} --from-literal=appPassword="
fi
if [ -s ./keystorePassword.txt ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=keystorePassword=./keystorePassword.txt"
else
  SECRET_ARGS="${SECRET_ARGS} --from-literal=keystorePassword="
fi
if [ -s ./keystore-mykey.key ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=keystoreKey-mykey=./keystore-mykey.key"
else
  SECRET_ARGS="${SECRET_ARGS} --from-literal=keystoreKey-mykey="
fi
if [ -s ./keystore-mykey.crt ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=keystoreCert-mykey=./keystore-mykey.crt"
else
  SECRET_ARGS="${SECRET_ARGS} --from-literal=keystoreCert-mykey="
fi
```

```
if [ -s ./keystore-mykey.pass ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=keystorePass-mykey=./keystore-mykey.pass"
else
  SECRET_ARGS="${SECRET_ARGS} --from-literal=keystorePass-mykey="
fi

if [ -s ./truststorePassword.txt ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=truststorePassword=./truststorePassword.txt"
else
  SECRET_ARGS="${SECRET_ARGS} --from-literal=truststorePassword="
fi
if [ -s ./truststoreCert-mykey.crt ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=truststoreCert-mykey=./truststoreCert-mykey.crt"
else
  SECRET_ARGS="${SECRET_ARGS} --from-literal=truststoreCert-mykey="
fi
if [ -s ./odbc.ini ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=odbcini=./odbc.ini"
else
  SECRET_ARGS="${SECRET_ARGS} --from-literal=odbcini="
fi
if [ -s ./policy.xml ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=policy=./policy.xml "
else
  SECRET_ARGS="${SECRET_ARGS} --from-literal=policy="
fi
if [ -s ./policyDescriptor.xml ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=policyDescriptor=./policyDescriptor.xml "
else
  SECRET_ARGS="${SECRET_ARGS} --from-literal=policyDescriptor="
fi
```

```
if [ -s ./serverconf.yaml ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=serverconf=./serverconf.yaml "
else
  SECRET_ARGS="${SECRET_ARGS} --from-literal=serverconf="
fi

if [ -s ./setdbparms.txt ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=setdbparms=./setdbparms.txt "
else
  SECRET_ARGS="${SECRET_ARGS} --from-literal=setdbparms="
fi
if [ -s ./extensions.zip ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=extensions=./extensions.zip "
fi
if [ -s ./switch.json ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=switch=./switch.json "
fi
if [ -s ./agentx.json ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=agentx=./agentx.json "
fi
if [ -s ./agentp.json ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=agentp=./agentp.json "
fi
if [ -s ./agentc.json ]; then
  SECRET_ARGS="${SECRET_ARGS} --from-file=agentc=./agentc.json "
fi
#########################################################
# Create the Kubernetes secret resource
#########################################################
echo "Creating secret"
echo "kubectl create secret generic ${TARGET_SECRET_NAME}${SECRET_ARGS}"
```

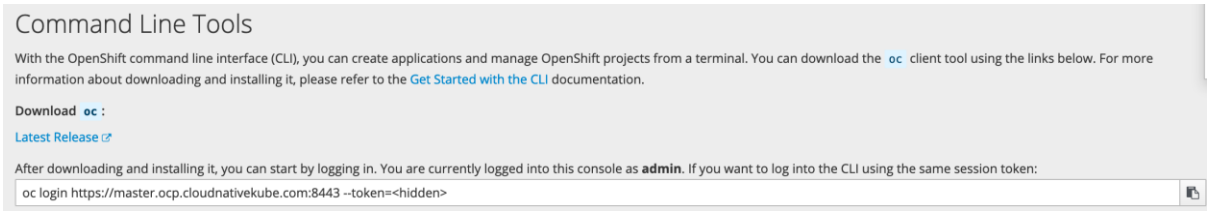
oc create secret generic ${TARGET_SECRET_NAME}${SECRET_ARGS} -n ace

----------end snippet----------

To run the script:

1. Connect to the RHOCP via oc login.

   Select Command line tools and follow the instruction here:



2. Create an empty directory and populate it with the generateSecrets.sh script
3. Create the files described above required for your configuration in the same directory.

   As an example, to set dbparms for external resource access to kafka for an integration server, a setdbparms file will contain something similar to:

   kafka::KAFKA token KsTEutxASFTZtvgqlqQZKRrkFqeU_QqhWPkN8GdJrz95

   !! NB: Take care not to include additional new lines at the end of the files (such as keystorePassword.txt) - some editors add these automatically on save and can result in the containers not starting due to password mismatch.

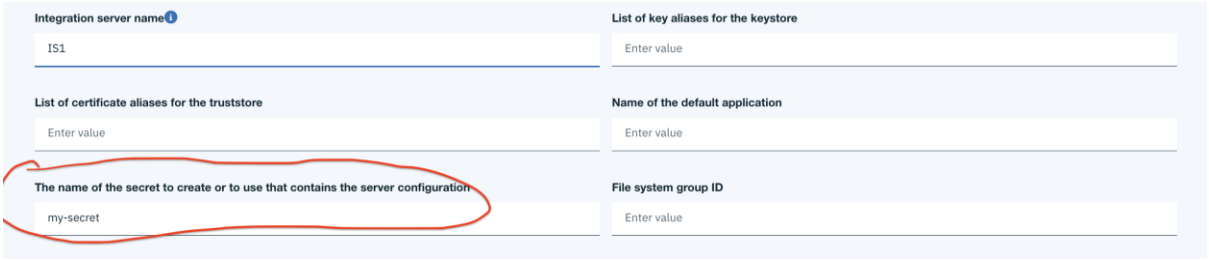
4. Make sure you're in the ACE namespace using the command : *oc project ace*
5. Run the command as shown, alternatively giving the second argument as the release name=, specifically for setting adminusers and viewusers for the webui (linux/mac syntax shown) :

   ./generateSecrets.sh my-secret [release1]

   You should see something similar to

./generateSecrets.sh ace-secret
Creating secret
kubectl create secret generic my-secret --from-literal=mqsc= --from-literal=adminPassword= --from-literal=appPassword= --from-literal=keystorePassword= --from-literal=keystoreKey-mykey= --from-literal=keystoreCert-mykey= --from-literal=keystorePass-mykey= --from-file=truststorePassword=./truststorePassword.txt --from-file=truststoreCert-mykey=./truststoreCert-mykey.crt --from-literal=odbcini= --from-literal=policy= --from-literal=policyDescriptor= --from-file=serverconf=./serverconf.yaml --from-file=setdbparms=./setdbparms.txt
secret/my-secret created

6. When deploying the ACE integration server, populate the following highlighted field with the secret name



Example:

Add an MQSC file to an ACE/MQ Deployment to pre-configure access to the ACE Queue Manager

- Do steps 1-4 above.
- Create an MQSC file called **mqsc.txt** in its own directory

DEFINE CHANNEL(IVT.SVRCONN) CHLTYPE(SVRCONN)

SET CHLAUTH(IVT.SVRCONN) TYPE(BLOCKUSER) USERLIST(nobody)

ALTER AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) AUTHTYPE(IDPWOS) CHCKCLNT(NONE) ADOPTCTX(NO)

SET CHLAUTH(IVT.SVRCONN) TYPE (ADDRESSMAP) ADDRESS(*) MCAUSER('mqm')

define qlocal (RTREP.Q)

define qlocal(RTREQ.Q)

REFRESH SECURITY TYPE(CONNAUTH)

- Run the ‘generateSecrets.sh ace-secret’

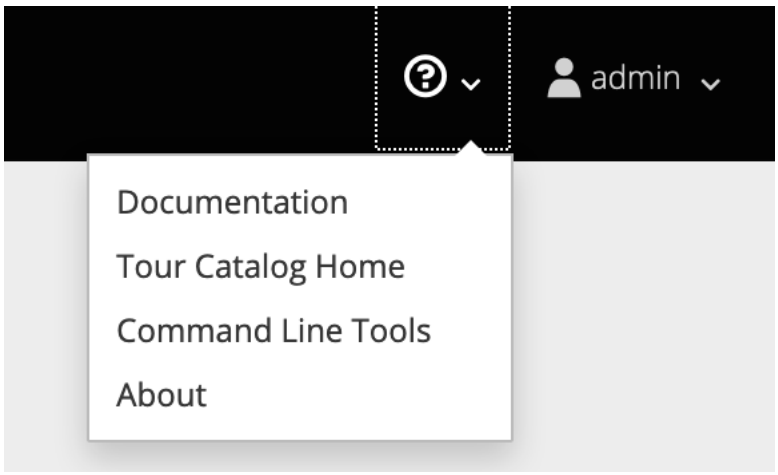
Do step 6 above.

Test access to the queue manager as in the section Testing the Queue manager

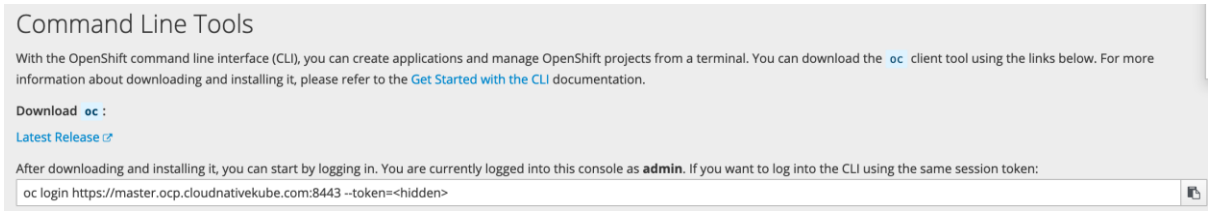
ACE on ICP4i on RHOS via Command Line Helm Release

Download and install command line tools

Go to the registry console URL
https://registry-console-default.apps.ocp.cloudnativekube.com/registry



Select Command line tools and follow the instructions here:



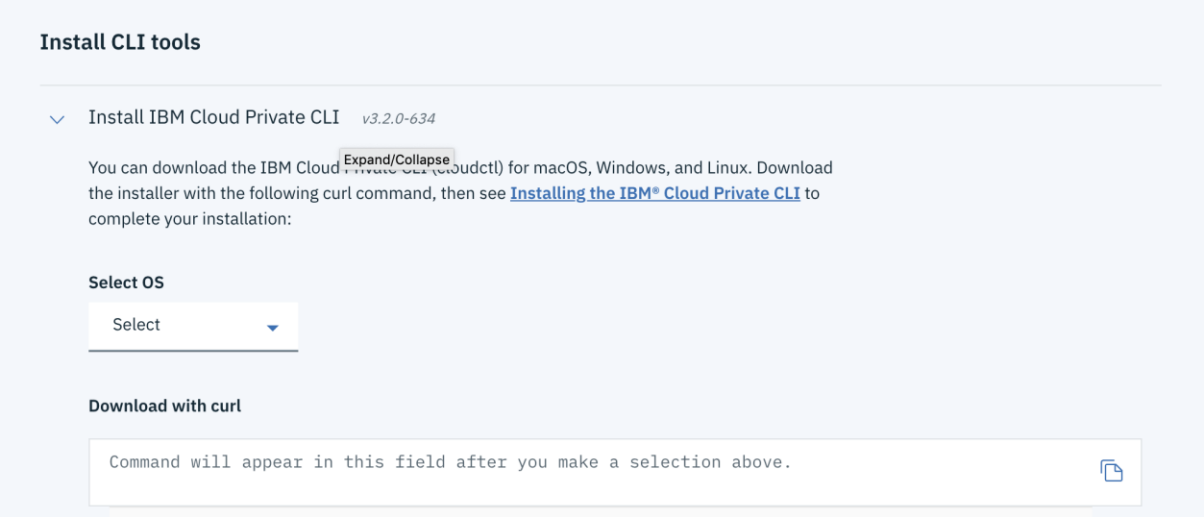
Once logged in:

Switch to the ACE Project:

oc project ace

Get the cloudctl version from ICP:

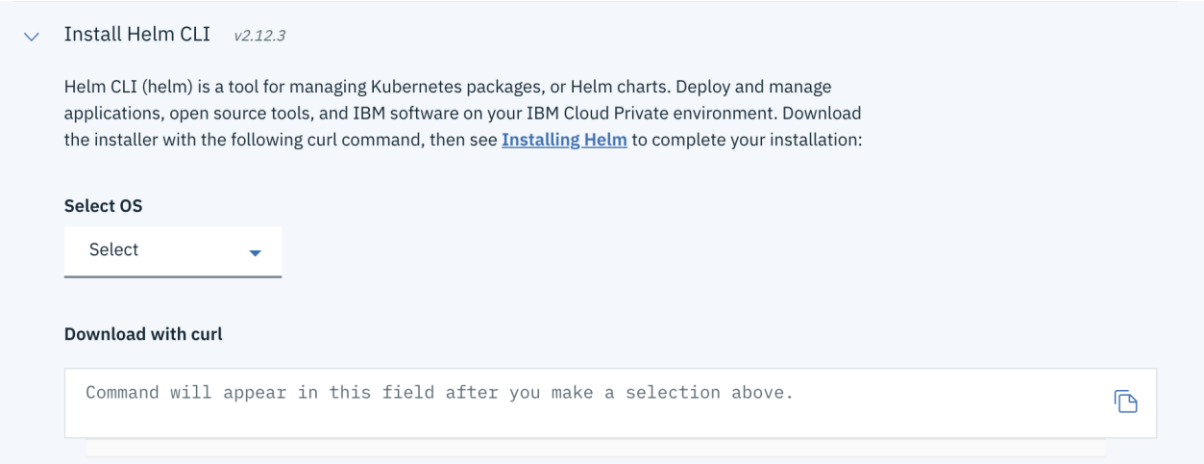
Goto https://icp-console.apps.ocp.cloudnativekube.com/console/tools/cli



Select your OS and download the file/tar or zip file.

Once the executable id identified copy it and rename it to cloudctl/cloudctl.exe

Follow the same process for the helm executable (the versions of the helm client and tiller on the server must match):



Login, setup and Helm install

Login to the cluster as follows:

sudo cloudctl login -a https://icp-console.apps.ocp.cloudnativekube.com -u admin -p ######## -n default

Run the following helm command:

sudo helm init --client-only --skip-refresh

You will see the following in the output of this command:

$HELM_HOME has been configured at <your_helm_home>

Run the flowing commands:

export HELM_HOME=<your_helm_home>

export ICP_URL=https://icp-console.apps.ocp.cloudnativekube.com

The expression below points to the location of the repository used for icp4i

sudo helm repo add local-charts $ICP_URL/helm-repo/charts --ca-file $HELM_HOME/ca.pem --cert-file $HELM_HOME/cert.pem --key-file $HELM_HOME/key.pem

This has prepared you for running helm install against a chart, image(s) and configuration.

The following script installs an ACE image with MQ, asking for command line parameters for helm release name, integration server name and an optional secret. It installs a pre-loaded bar file called SoE.bar provided by the context URL.

```
if [ "$2" == "" ]
then
echo "Please provide a integration server name as the second argument"
echo "Usage: installACE.sh <release_name> <Integration_server_name> <queue_manager_name> [<secret_name>]"
exit 1
fi
if [ "$3" == "" ]
then
echo "Please provide a queue manager name as the third argument"
echo "Usage: installACE.sh <release_name> <Integration_server_name> <queue_manager_name> [<secret_name>]"
exit 1
fi
if [ "$4" == "" ]
then
SECRET_ARG="nil"
else
SECRET_ARG=$4
fi
sudo helm install local-charts/ibm-ace-server-icp4i-prod
```

--name=$1

--set license=accept

--tls

--set integrationServer.configurationSecret=$SECRET_ARG

--set contentServerURL=https://ace-1-ibm-ace-dashboard-icp4i-prod:3443/v1/directories/SoE?23e24c6c-c709-4b5e-a7ec-0dd05b96394f

--set imageType=acemqserver

--set image.acemq=docker-registry.default.svc:5000/ace/ibm-ace-mq-server-prod:11.0.0.5.1

--set image.pullPolicy=IfNotPresent

--set image.configurator=docker-registry.default.svc:5000/ace/ibm-ace-icp-configurator-prod:11.0.0.5.1

--set acemq.qmname=$3

--set integrationServer.name=$2

--set service.iP=icp-proxy.apps.ocp.cloudnativekube.com

--set persistence.enabled=true

--set persistence.useDynamicProvisioning=true

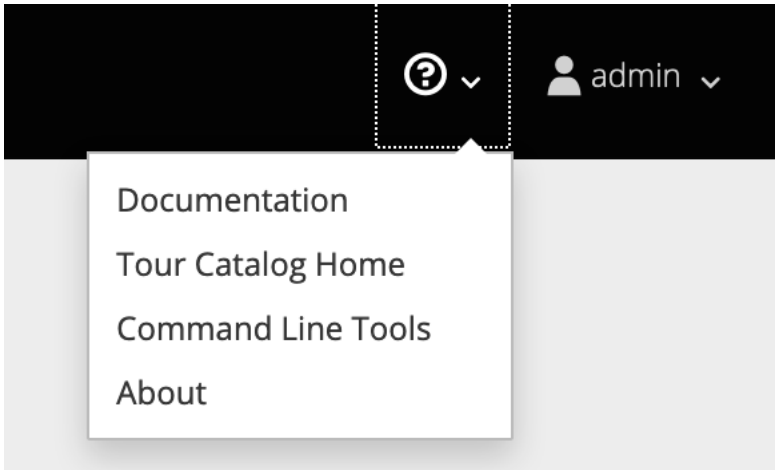
--set dataPVC.name=data --set dataPVC.storageClassName=managed-nfs-storage

--set productionDeployment=false

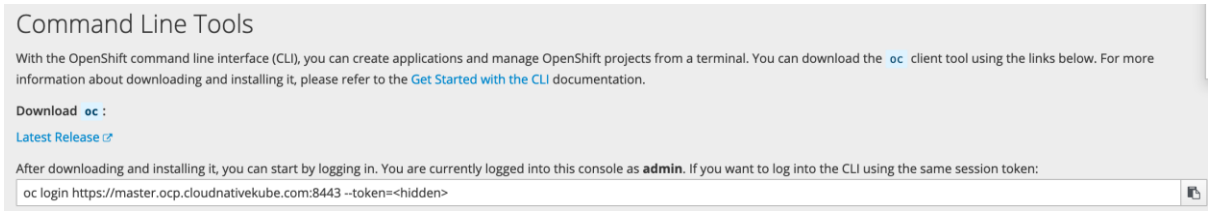
--set metrics.enabled=false

Configuring an MQ Queue Manager deployed with ACE

Go to the registry console URL
https://registry-console-default.apps.ocp.cloudnativekube.com/registry



Select Command line tools and follow the instruction here:



Once logged in:

Switch to the MQ Project

peters-mbp-4:ICP4i peterajessup$ oc project ace

Now using project "ace" on server "https://master.ocp.cloudnativekube.com:8443".

Get the ace running pods

peters-mbp-4:ICP4i peterajessup$ oc get pods

NAME READY STATUS RESTARTS AGE

ace-1-ibm-ace-dashboard-icp4i-prod-6c4cff67d9-8px2p 2/2 Running 0 6d

acemq-ibm-ace-server-icp4i-prod-0 1/1 Running 0 22h

daace1-ibm-ace-server-icp4i-prod-0 1/1 Running 0 5d

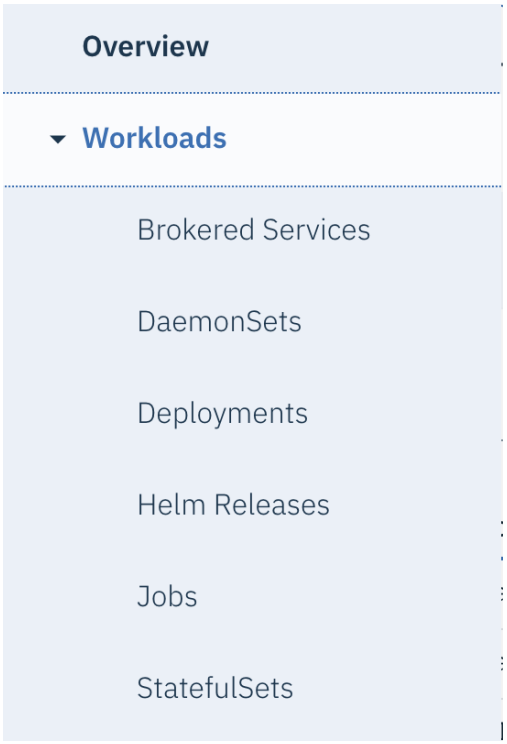
my-ace-rel-ibm-ace-server-icp4i-prod-66cf9c496c-gzwll 1/1 Running 0 2d

my-ace-rel-ibm-ace-server-icp4i-prod-66cf9c496c-j4n4f 1/1 Running 0 2d

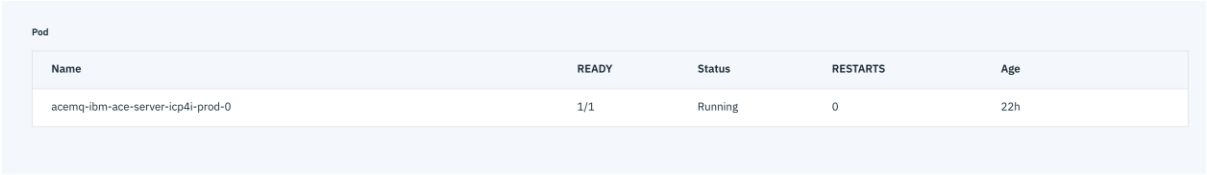
my-ace-rel-ibm-ace-server-icp4i-prod-66cf9c496c-znzkb 1/1 Running 0 2d

Identify your pod (assuming it's *acemq-ibm-ace-server-icp4i-prod-0* for the sake of this example)

If you're not sure of the ACE pod name navigate the Cloud Pak Foundation I/F under Helm Releases:



Find the release name you specified in the Chart before deploying and click on this to show the pods:



kubectl exec -it acemq-ibm-ace-server-icp4i-prod-0 -- /bin/bash

bash-4.2$ runmqsc

5724-H72 (C) Copyright IBM Corp. 1994, 2019.

Starting MQSC for queue manager QM2.

Copy and paste the following into the mqsc tool:

DEFINE CHANNEL(IVT.SVRCONN) CHLTYPE(SVRCONN)

AMQ8014I: IBM MQ channel created.

SET CHLAUTH(IVT.SVRCONN) TYPE(BLOCKUSER) USERLIST(nobody)

AMQ8877I: IBM MQ channel authentication record set.

ALTER AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) AUTHTYPE(IDPWOS) CHCKCLNT(NONE) ADOPTCTX(YES)

AMQ8567I: IBM MQ authentication information changed.

SET CHLAUTH(IVT.SVRCONN) TYPE (ADDRESSMAP) ADDRESS(*) MCAUSER('mqm')

AMQ8877I: IBM MQ channel authentication record set.

REFRESH SECURITY TYPE(CONNAUTH)

define qlocal (RTREP.Q)

AMQ8006I: IBM MQ queue created.

define qlocal(RTREQ.Q)

AMQ8006I: IBM MQ queue created.

Hi CTRL-C

5 MQSC commands read.

bash-4.2$ exit

exit

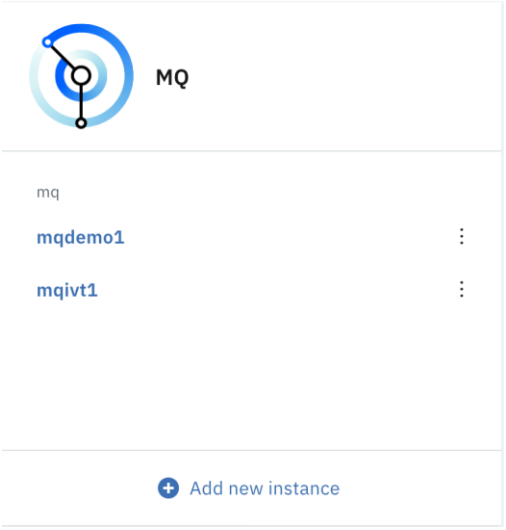
You're Done.

See the section on Testing the Queue Manager for how to test.

MQ on ICP4i on RHOS via the ICP4i Navigator

MQ using the ICP4i console

https://icp-proxy.apps.ocp.cloudnativekube.com:3443/integration



Click on add a new instance.

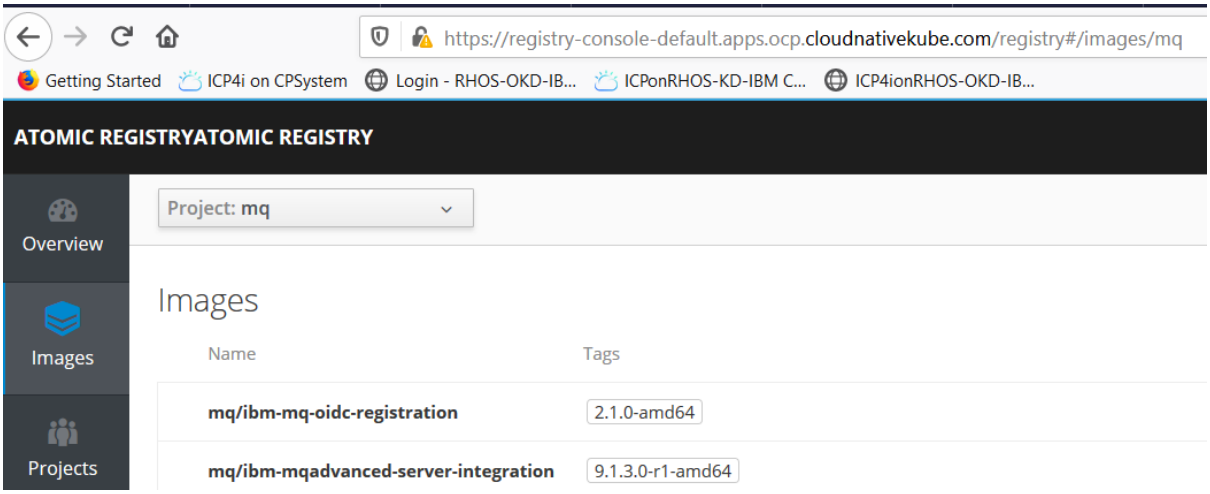
Information you'll need to configure the release

Collected information you need to configure the release.

ICP4i Proxy address (from ICP configMaps): icp-proxy.apps.ocp.cloudnativekube.com

onboard docker image registry (from oc command)

https://registry-console-default.apps.ocp.cloudnativekube.com/registry



MQ server image:

docker-registry.default.svc:5000/mq/ibm-mqadvanced-server-integration

MQ Server image Tag:

9.1.3.0-r1

MQ Server OIDC registration image

docker-registry.default.svc:5000/mq/ibm-mq-oidc-registration

MQ Server OIDC registration image tag:

2.1.0

Managed NFS name (from oc command):

managed-nfs-storage

Cluster Hostname (for deployment)

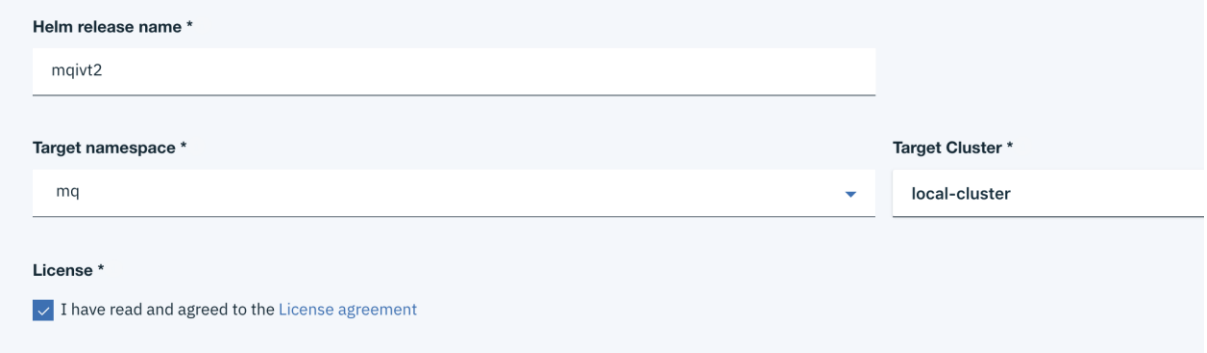
icp-proxy.apps.ocp.cloudnativekube.com

TLS Secret

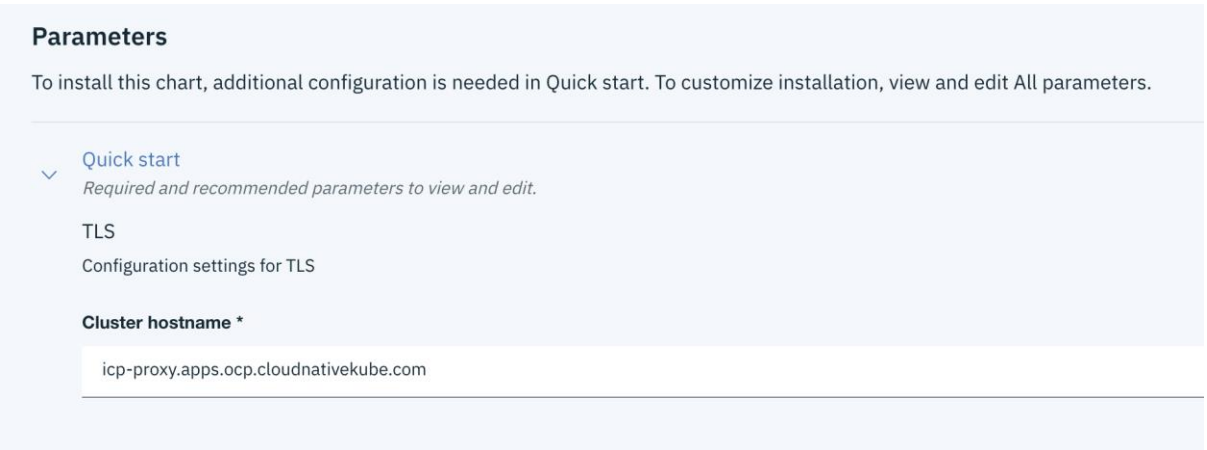
Ibm-mq-tls-secret

Configuring the MQ (helm) release

Give it a unique name and add it to the mq namespace



Cluster host name - icp-proxy.apps.ocp.cloudnativekube.com



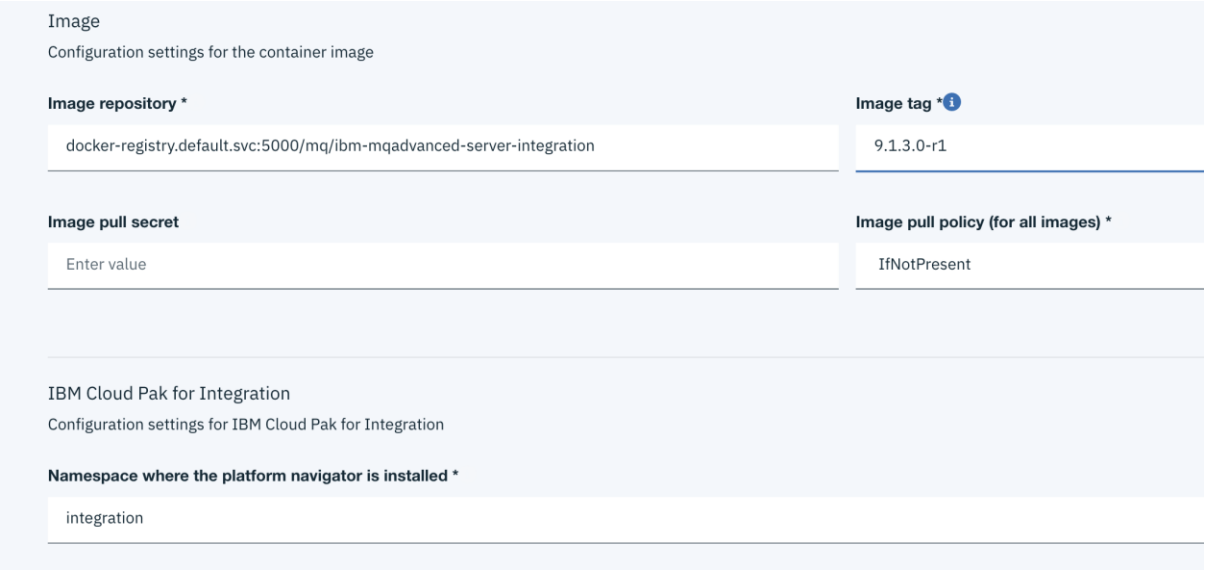
Enter the mq image container details

Image repo - docker-registry.default.svc:5000/mq/ibm-mqadvanced-server-integration

Image tag – 9.1.3.0-r1

Results in:

docker-registry.default.svc:5000/mq/ibm-mqadvanced-server-integration:9.1.3.0-r1-amd64



Enter the mq oidc rego container details

docker-registry.default.svc:5000/mq/ibm-mq-oidc-registration

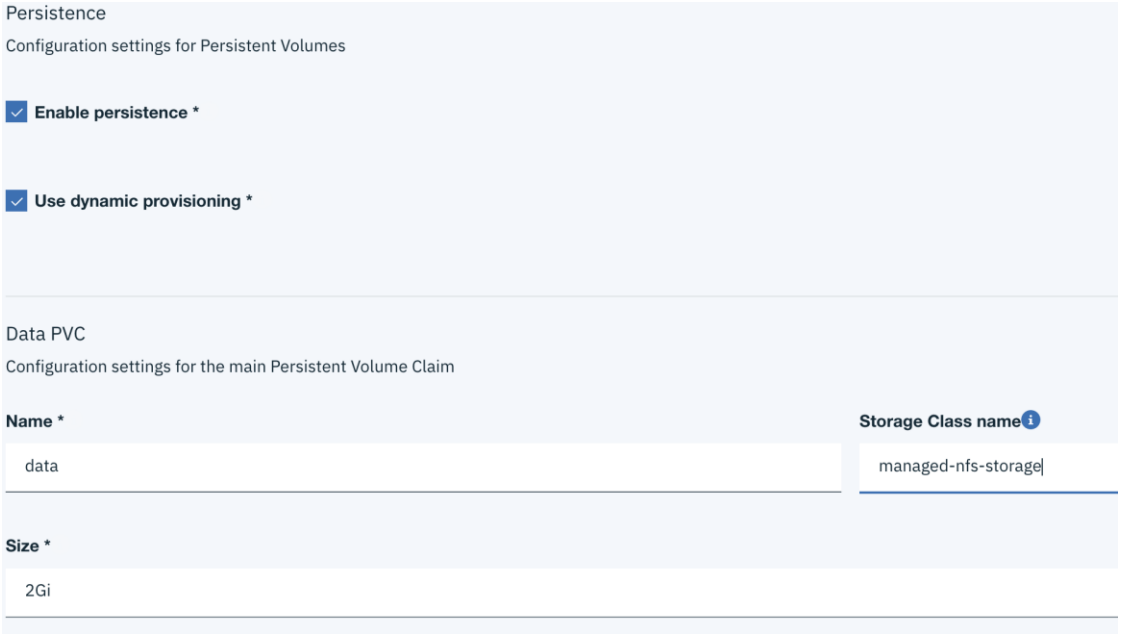
2.1.0

Enter the TLS secret and check the box.

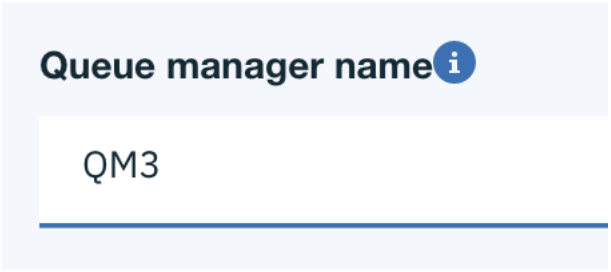
ibm-mq-tls-secret

Configure persistence

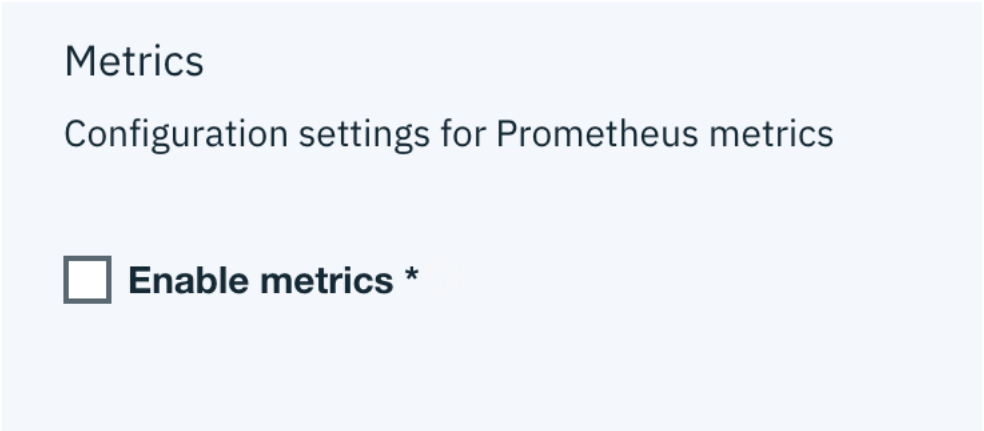
managed-nfs-storage



Give your queue manager a name else it defaults

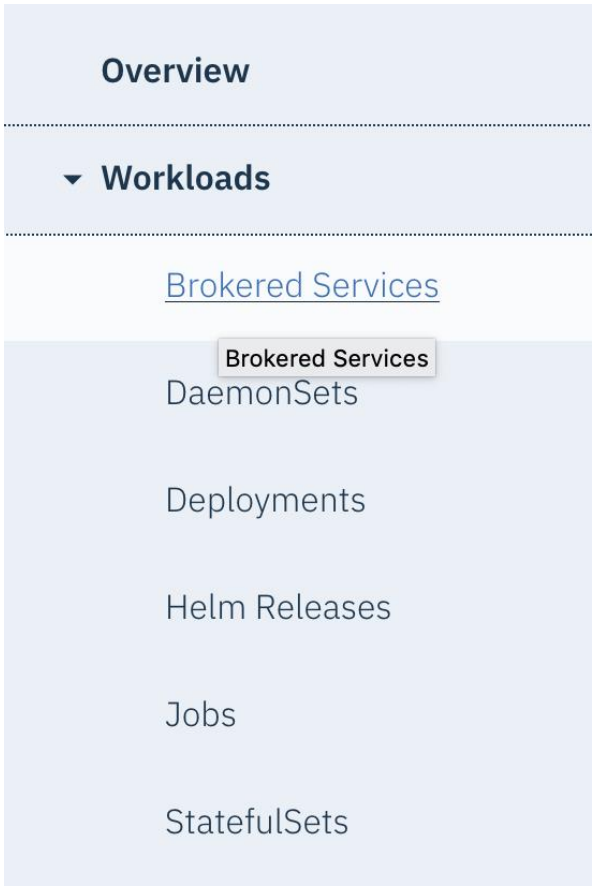


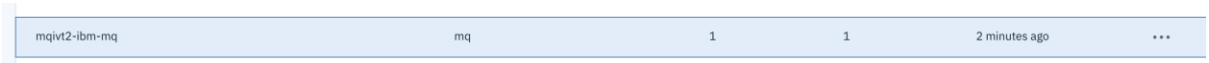
Uncheck ‘Enable Metrics’



Leave EVERYTHING else as default and click Install.

Might have to wait a little while – then goto Stateful Sets





Wait until you see the Pod ready 1/1

Pods

Search Pods

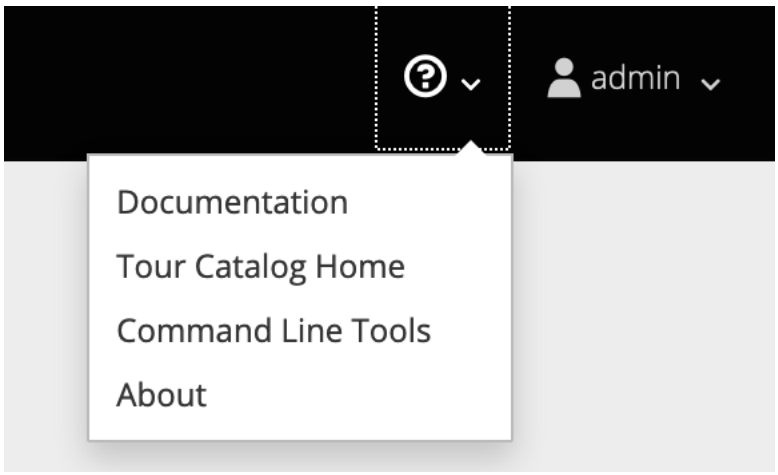
Name	Namespace	Status	Host IP	Pod IP	Ready	Start Time
mqivt2-ibm-mq-0	mq	Running	135.90.89.42	10.131.3.22	1/1	3 minutes ago

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

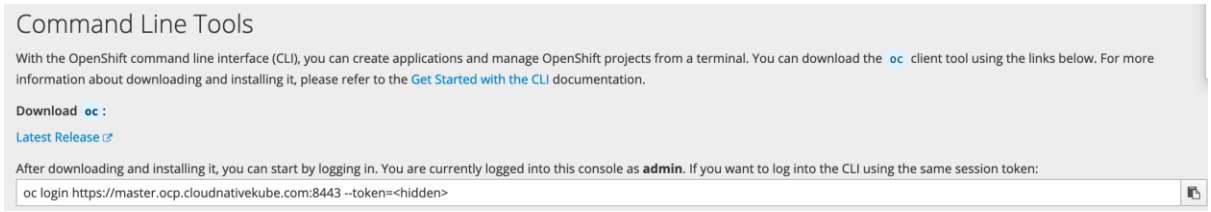
MQ on ICP4i via Command line Helm Release

Download command line tools

Go to the registry console URL
https://registry-console-default.apps.ocp.cloudnativekube.com/registry



Select Command line tools and follow the instructions here:



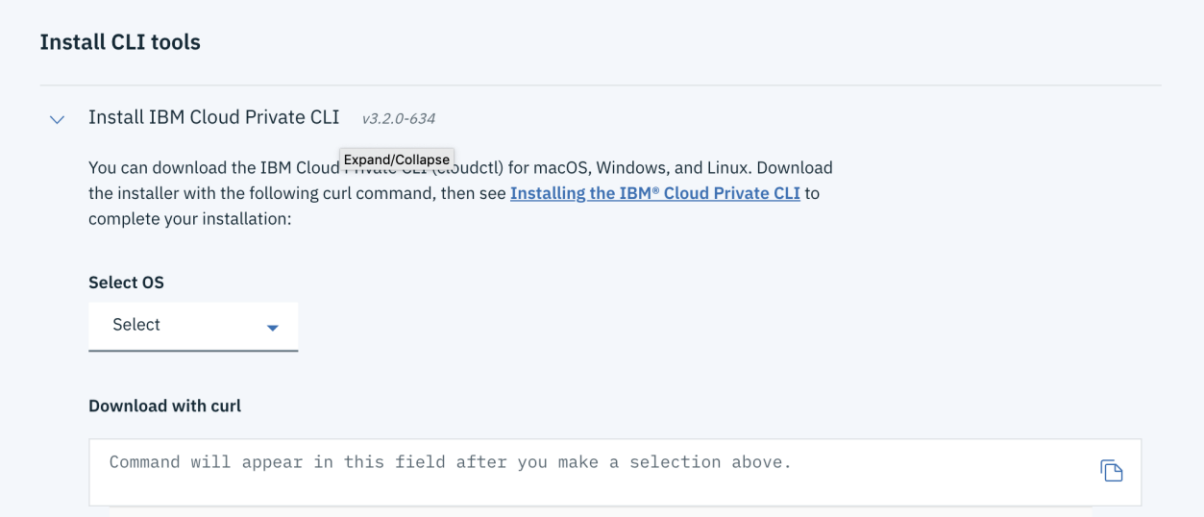
Once logged in:

Switch to the MQ Project:

oc project mq

Get the cloudctl version from ICP:

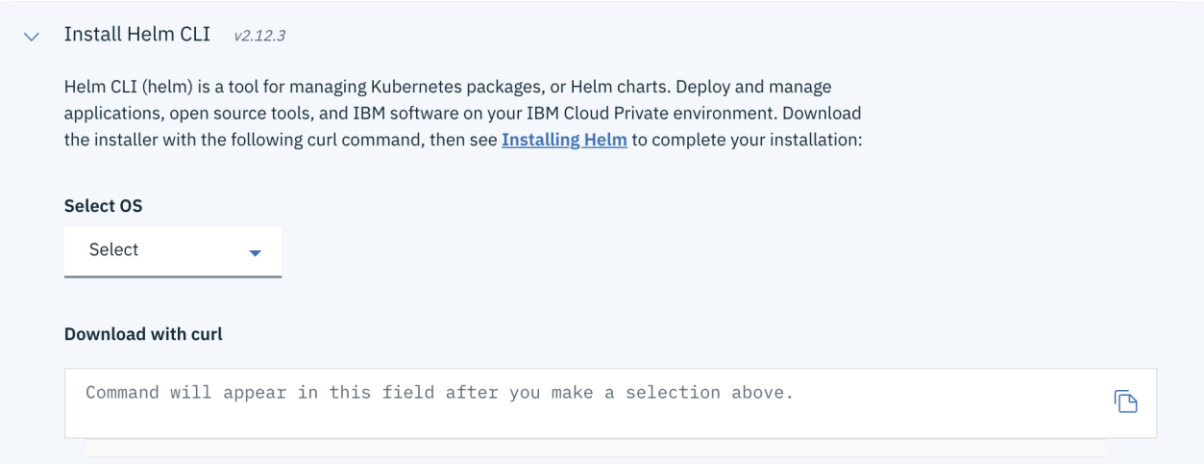
Goto https://icp-console.apps.ocp.cloudnativekube.com/console/tools/cli



Select your OS and download the file/tar or zip file.

Once the executable id identified copy it and rename it to cloudctl/cloudctl.exe

Follow the same process for the helm executable (the versions of the helm client and tiller on the server must match):



Log into the cluster

Login to the cluster as follows:

sudo cloudctl login -a https://icp-console.apps.ocp.cloudnativekube.com -u admin -p ####### -n default

Run the following helm command:

sudo helm init --client-only --skip-refresh

You will see the following in the output of this command:

$HELM_HOME has been configured at <your_helm_home>

Run the flowing commands:

export HELM_HOME=<your_helm_home>

export ICP_URL=https://icp-console.apps.ocp.cloudnativekube.com

The expression below points to the location of the repository used for icp4i

sudo helm repo add local-charts $ICP_URL/helm-repo/charts --ca-file $HELM_HOME/ca.pem --cert-file $HELM_HOME/cert.pem --key-file $HELM_HOME/key.pem

This has prepared you for running helm install against a chart, image(s) and configuration.

Command line Helm deployment

The MQ chart provides a sample install command and a list of parameters to pass to the installation.

The following script installs an MQ queue manager called ‘qmivt’ with a release name passed in as the first parameter and a queue manager name as the second parameter. The paramters are spaced out per line for clarity but the command should be executed as a single command with no new line characters.

sudo helm install ivt1 qmivt

--name $1 local-charts/ibm-mqadvanced-server-integration-prod

--set license=accept

--tls

--set productionDeployment=false

--set image.repository=docker-registry.default.svc:5000/mq/ibm-mqadvanced-server-integration

--set image.tag=9.1.3.0-r1

--set image.pullPolicy=IfNotPresent

--set image.pullSecret=nil

--set icp4i.namespace=integration

--set sso.registrationImage.repository=docker-registry.default.svc:5000/mq/ibm-mq-oidc-registration

--set sso.registrationImage.tag=2.1.0

--set sso.webAdminUsers="admin"

--set sso.uniqueUserIdentifier="sub"

--set tls.generate=true

--set tls.hostname=icp-proxy.apps.ocp.cloudnativekube.com

--set tls.secret=ibm-mq-tls-secret

--set persistence.enabled=true

--set persistence.useDynamicProvisioning=true

--set dataPVC.name="data"

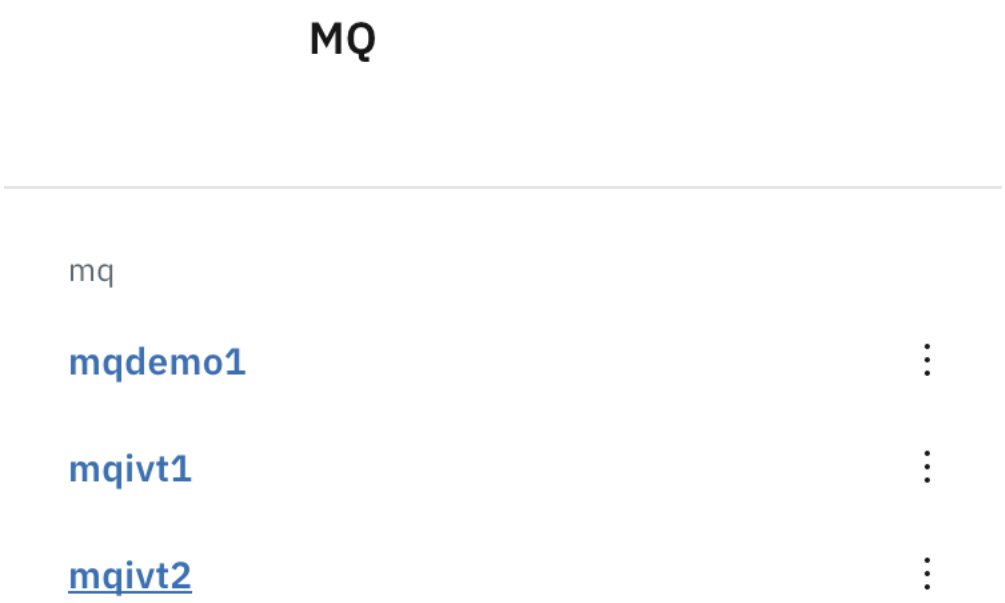
--set dataPVC.storageClassName="managed-nfs-storage"

--set metrics.enabled=false

--set queueManager.name=$2

Configuring your Queue Manager for client connection

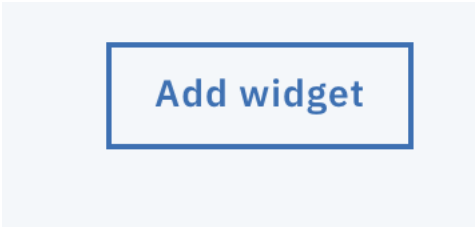
Go to Integration Navigator and click on your new MQ Service.



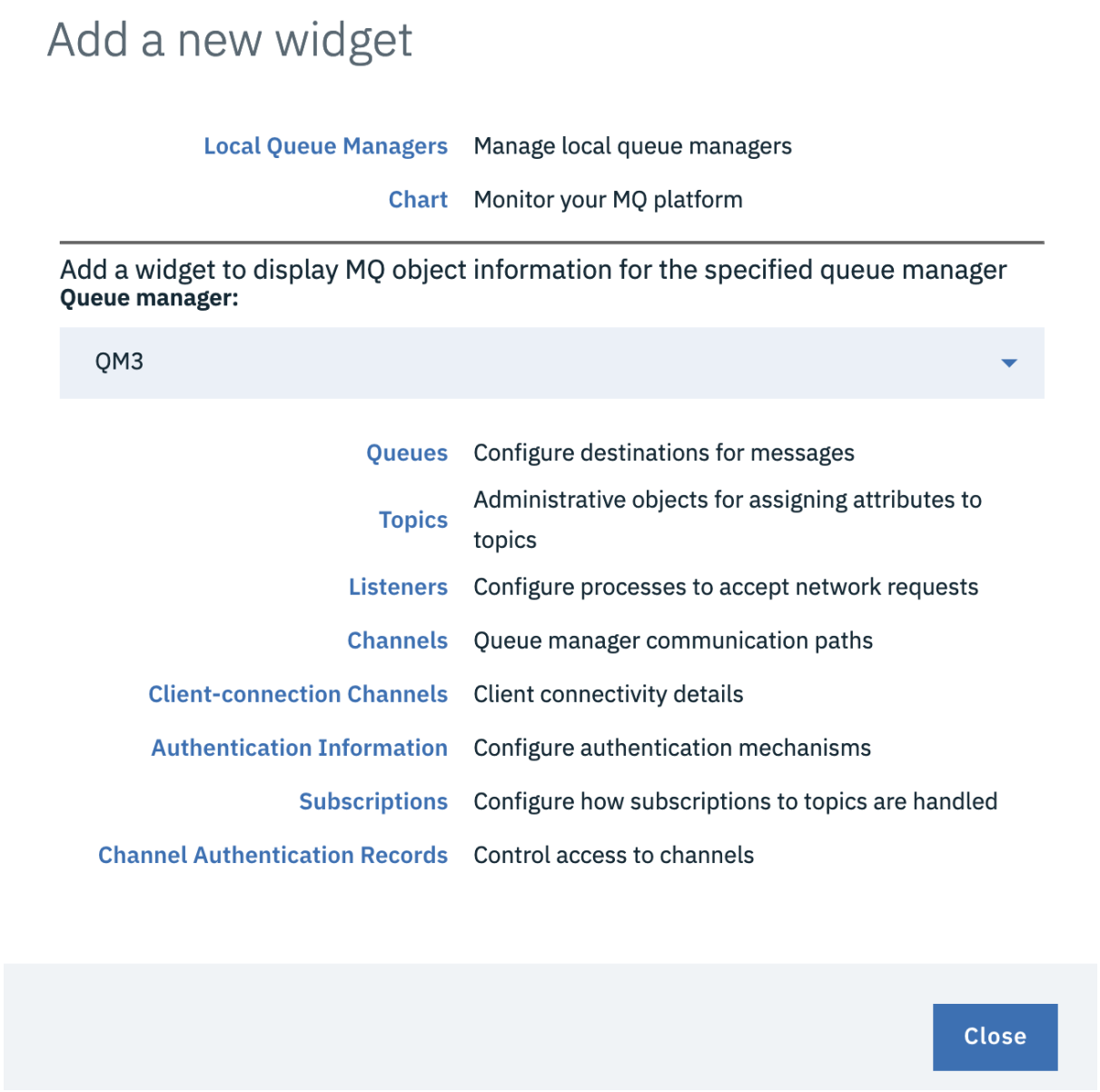
Connect via the MQ Web Console

Click through the security warnings to the MQ Console

Click Add Widget:

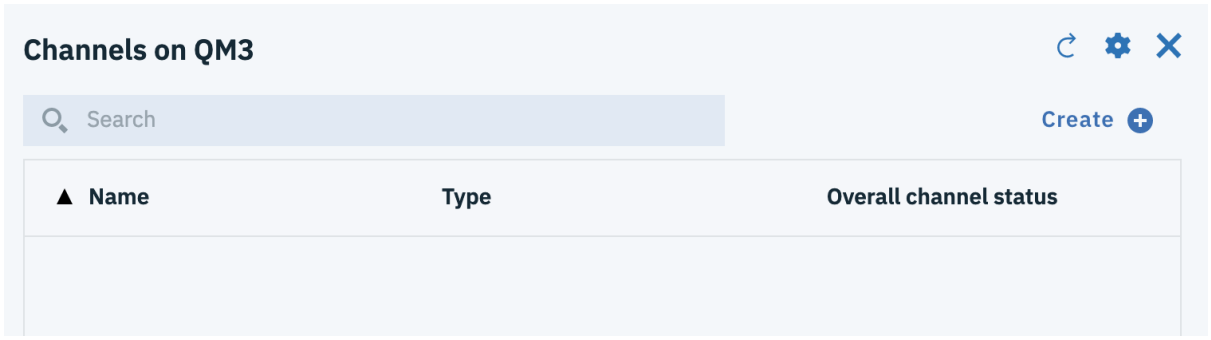


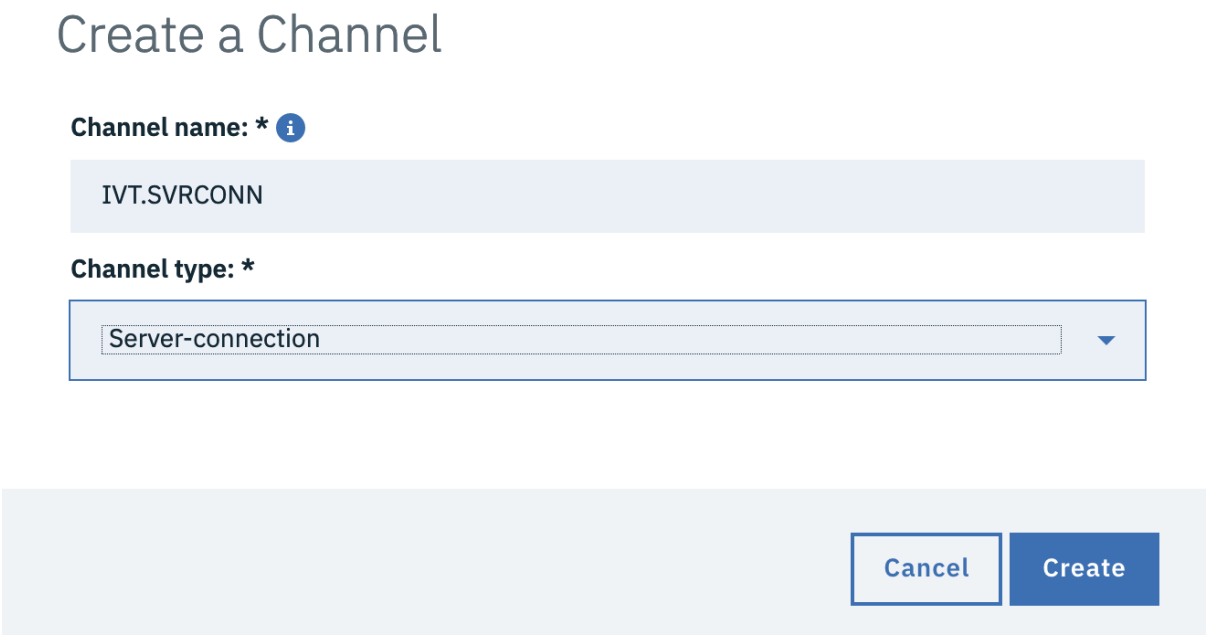
Select Queues:



Repeat this for Channels, Channel Authentication records and Authentication Information.

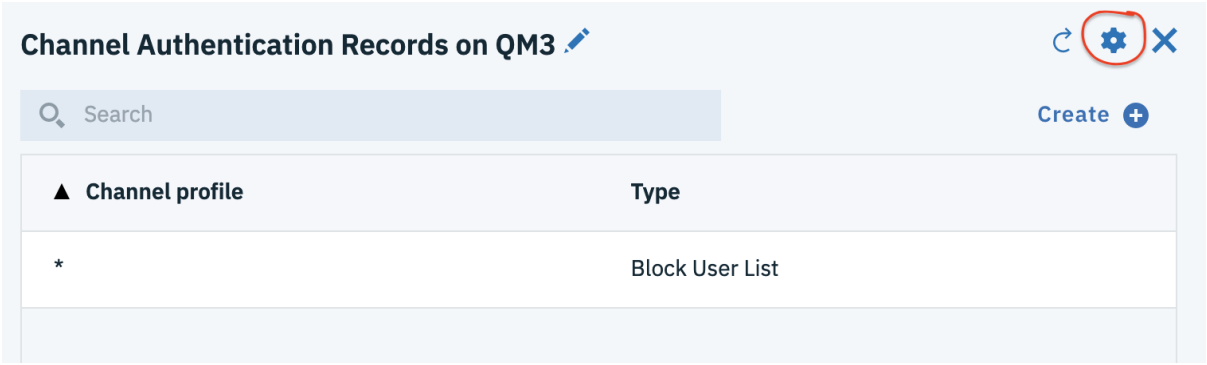
Click on Create Channel:





Click ‘Create’

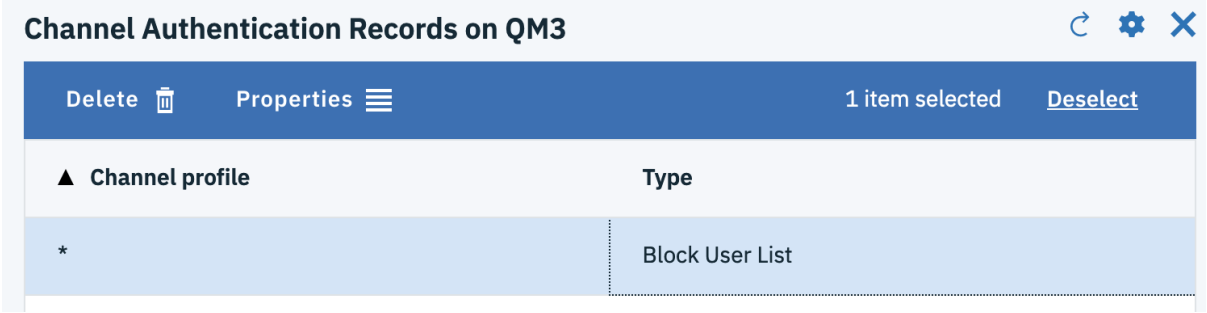
On Channel Authentication Records, click the Cog Icon



Select ‘System Objects’ -> Show, then click Save.

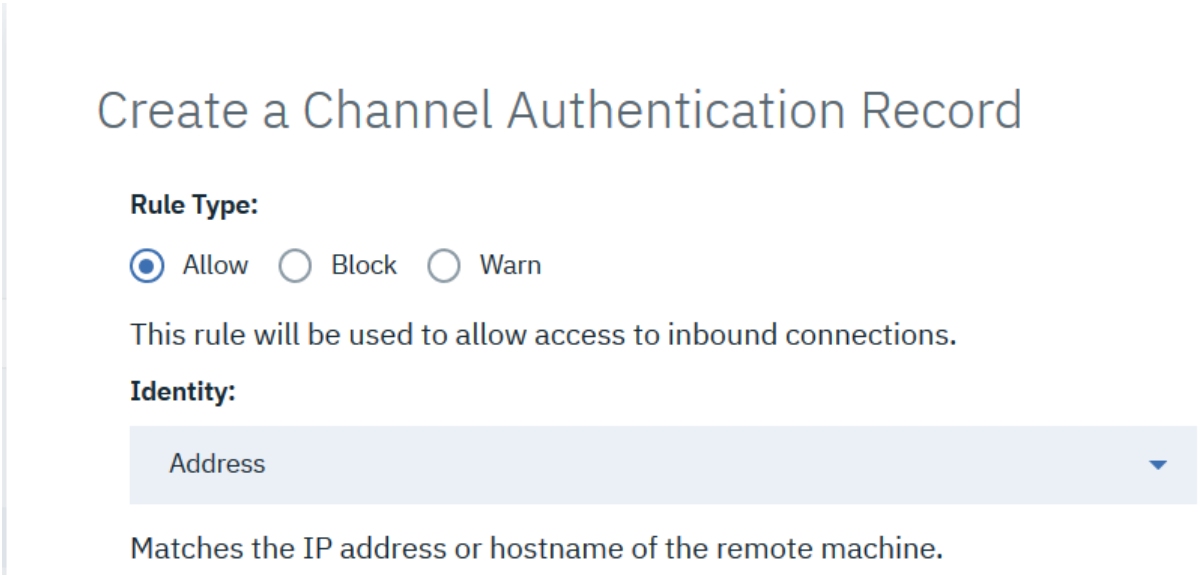
Repeat this step for Authentication Information as well.

Select the ‘Block User List on Channel Authentication Records and hit Delete then confirm delete.

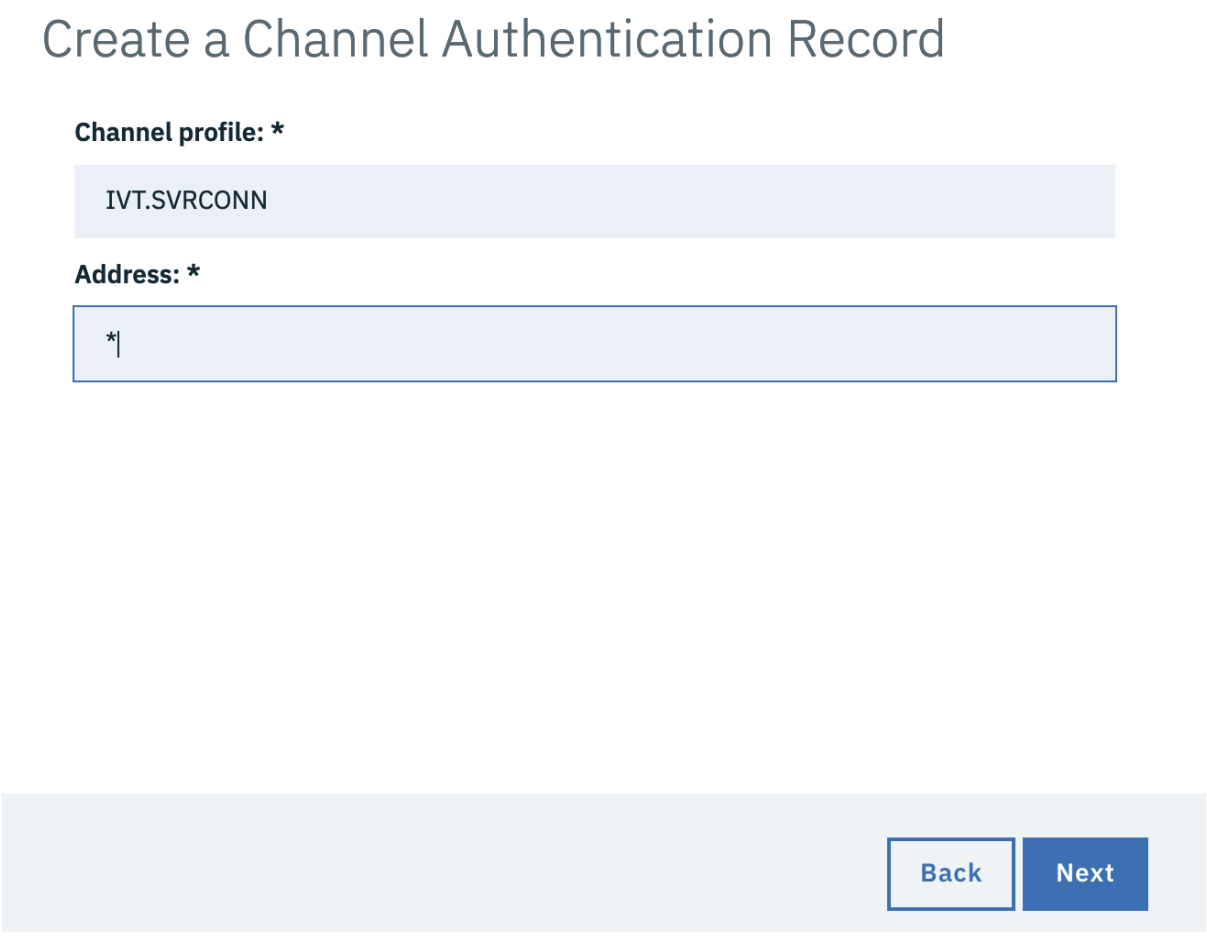


Click on ‘Create’

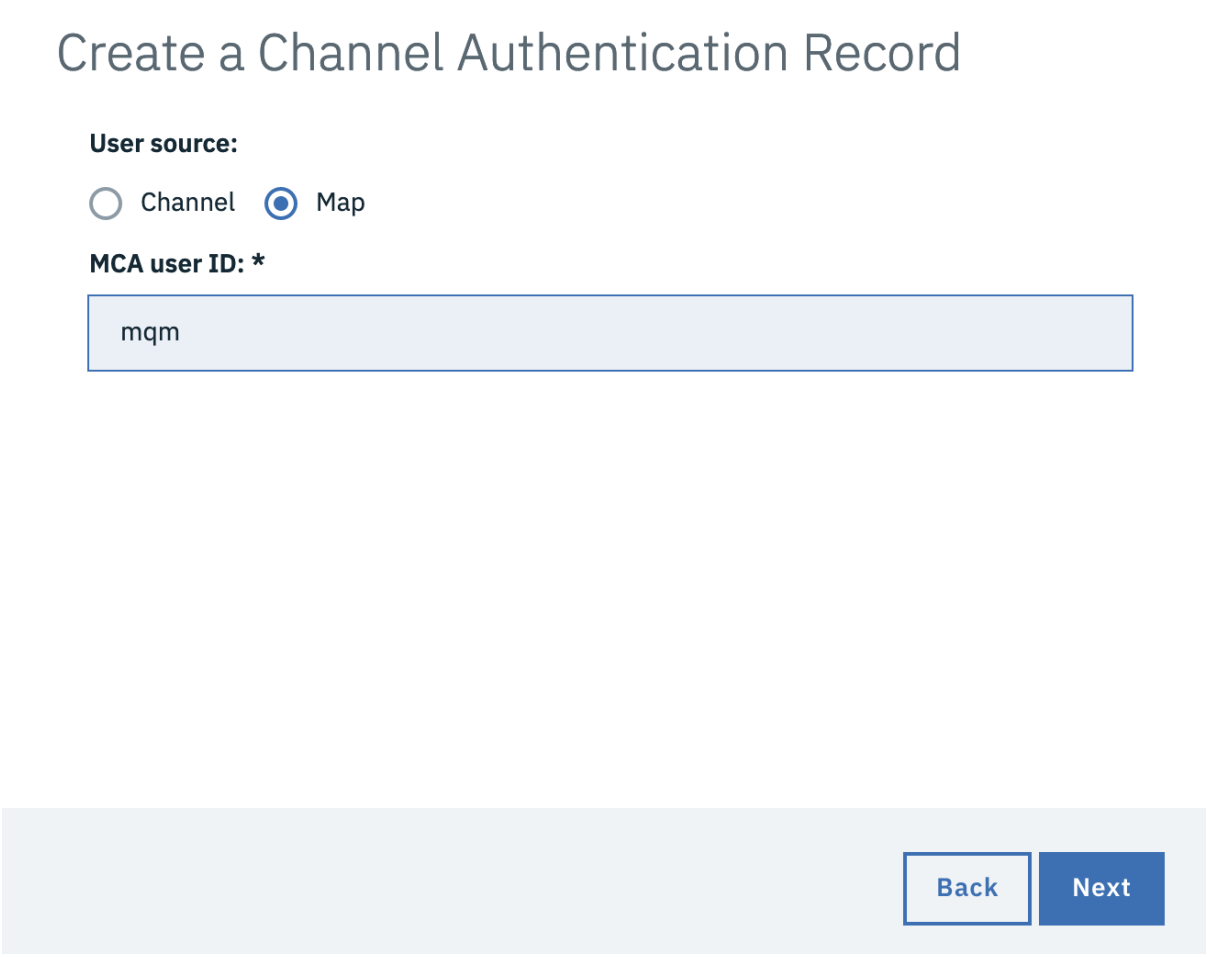
Select ‘Address’ from the drop down list, then click Next.



Enter the channel name created earlier, enter an Asterix in the Address box and click Next.

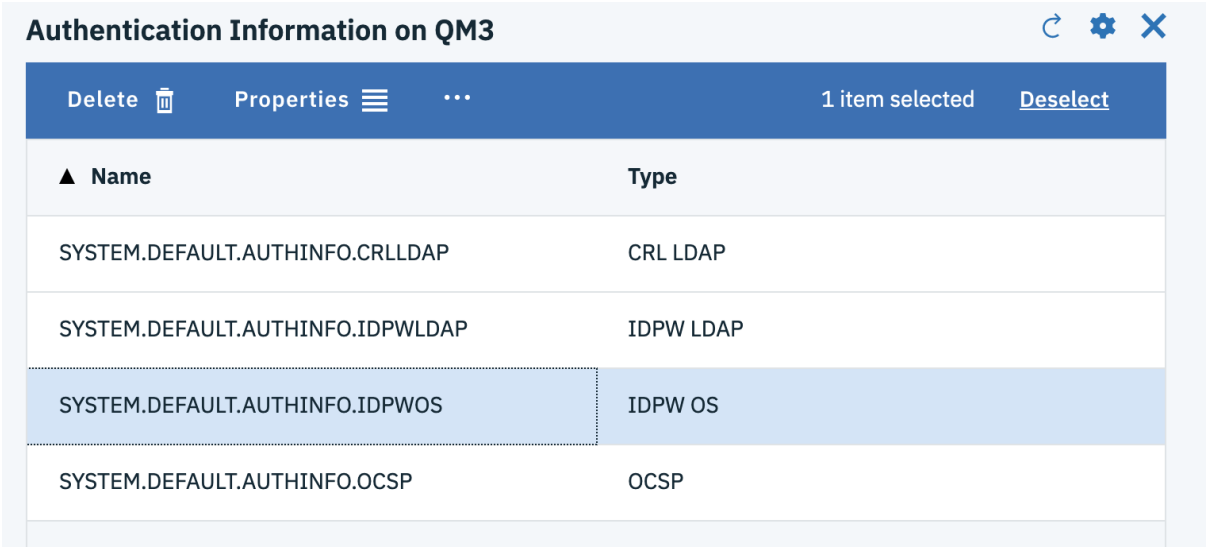


Click on ‘Map’ and enter ‘mqm’ as the MCA user ID*

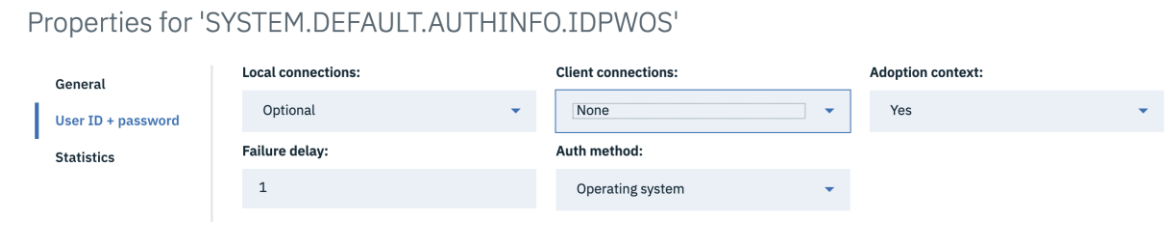


Click ‘Next’ then ‘Create’.

Select the ‘ID PWOS’ line on the ‘Authentication Information’ widget and click on ‘Properties’

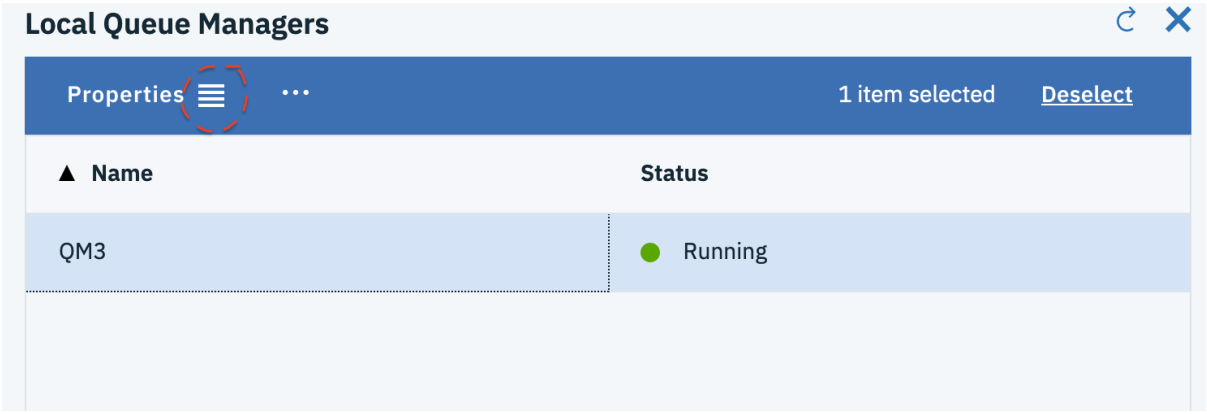


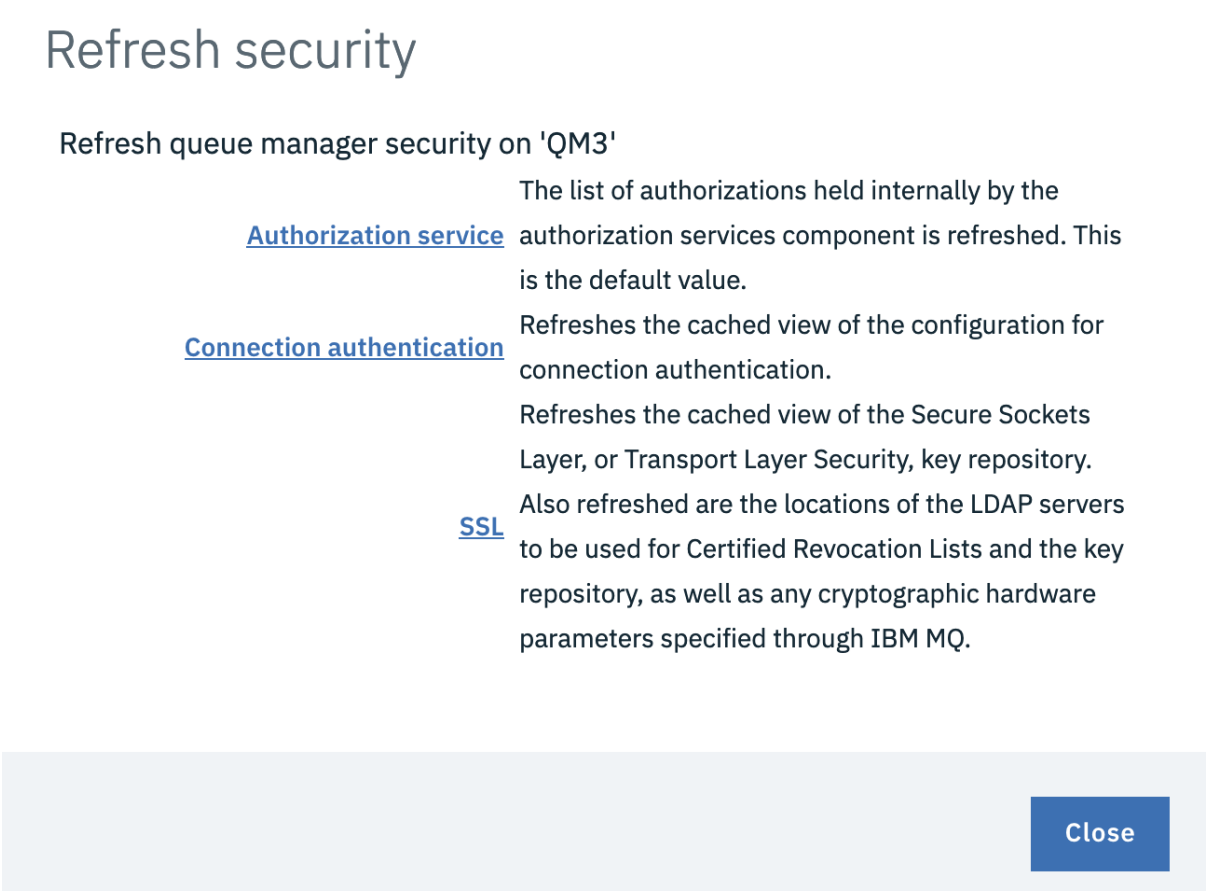
Change ‘Client Connections’ to ‘None’ in the dropdown list



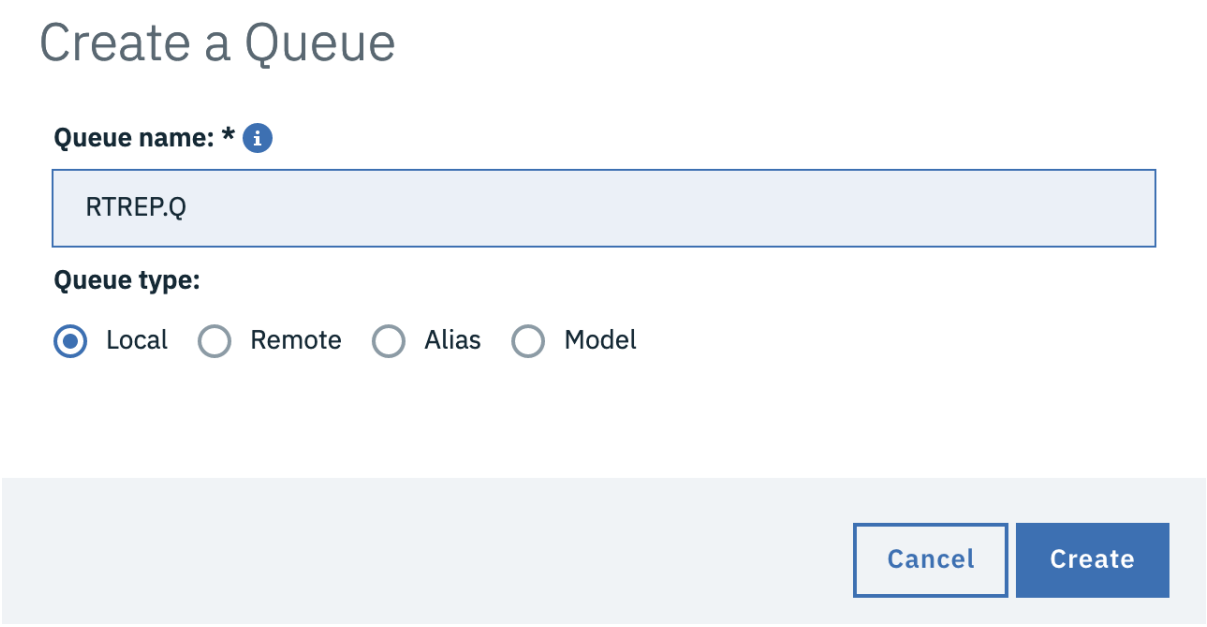
Click ‘Save’, then ‘Close’

On the Queue Manager Widget select the QM3 queue manager, click the’ …’ menu and select ‘Refresh Security’, then click on the ‘Connection authentication’ link.





On the Queues widget click ‘Create’.



Enter as shown and click ’ Create’.

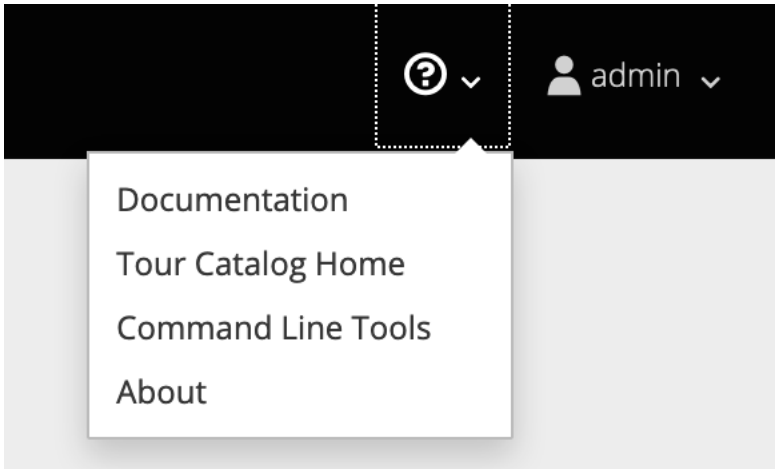
Repeat this for a queue called ‘RTREQ.Q’

Alternative Configuration procedure for a Queue manager

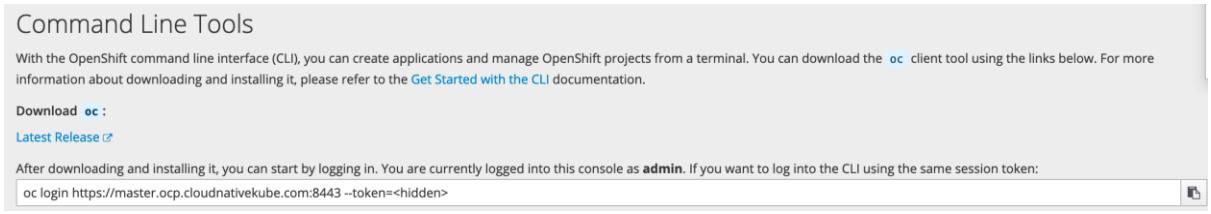
Bash into the container and runmqsc

This procedure uses the remote login facility of the container.

Go to the registry console URL
https://registry-console-default.apps.ocp.cloudnativekube.com/registry



Select Command line tools and follow the instruction here:



Once logged in:

Switch to the MQ Project

Oc project mq

peters-mbp-4:ICP4i peterajessup$ oc project mq

Now using project "mq" on server "https://master.ocp.cloudnativekube.com:8443".

Get the mq running pods

peters-mbp-4:ICP4i peterajessup$ oc get pods

NAME READY STATUS RESTARTS AGE

mqdemo1-ibm-mq-0 1/1 Running 0 6d

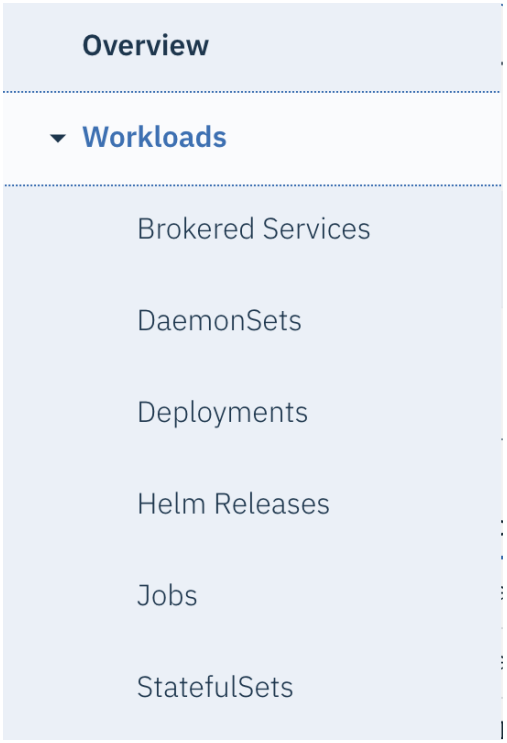
mqdemo1-ibm-mq-registration-6d64j 0/1 Completed 0 6d

mqivt1-ibm-mq-0 1/1 Running 0 1d

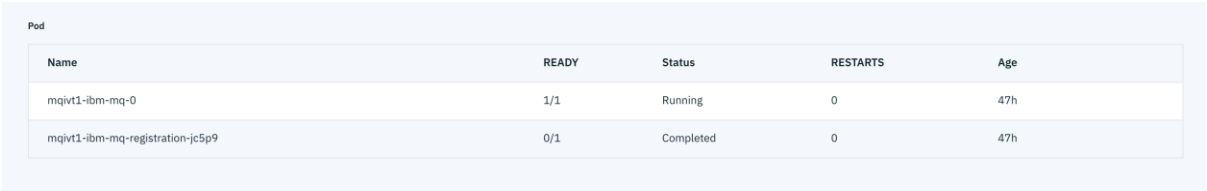
mqivt1-ibm-mq-registration-jc5p9 0/1 Completed 0 1d

Identify your pod (assuming it's mqivt-ibm-mq-0 for the sake of this example)

If you're not sure of the MQ pod name navigate the Cloud Pak Foundation I/F under Helm Releases:



Find the release name you specified in the Chart before deploying and click on this to show the pods:



Hint – the 'running pod' is the one you want……

kubectl exec -it mqivt1-ibm-mq-0 -- /bin/bash

bash-4.2$ runmqsc

5724-H72 (C) Copyright IBM Corp. 1994, 2019.

Starting MQSC for queue manager QM2.

Copy and paste the following into the mqsc tool:

DEFINE CHANNEL(IVT.SVRCONN) CHLTYPE(SVRCONN)

AMQ8014I: IBM MQ channel created.

SET CHLAUTH(IVT.SVRCONN) TYPE(BLOCKUSER) USERLIST(nobody)

AMQ8877I: IBM MQ channel authentication record set.

ALTER AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) AUTHTYPE(IDPWOS) CHCKCLNT(NONE) ADOPTCTX(YES)

AMQ8567I: IBM MQ authentication information changed.

SET CHLAUTH(IVT.SVRCONN) TYPE (ADDRESSMAP) ADDRESS(*) MCAUSER('mqm')

AMQ8877I: IBM MQ channel authentication record set.

REFRESH SECURITY TYPE(CONNAUTH)

define qlocal (RTREP.Q)

AMQ8006I: IBM MQ queue created.

define qlocal(RTREQ.Q)

AMQ8006I: IBM MQ queue created.

Hi CTRL-C

5 MQSC commands read.

bash-4.2$ exit

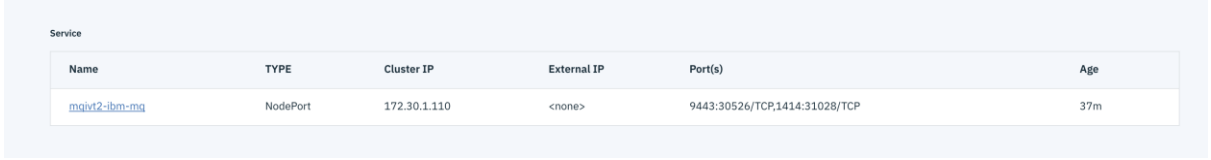
exit

Testing the Queue Manager.

Go to the Helm Releases view in the Cloud Pak Foundation interface

Locate the release name you chose and click on the list item.

Scroll down to the 'Service' and note the port mapping 1414:xxxxx/TCP



You will connect to your queue manager via the following settings:

Hostname: icp-proxy.apps.ocp.cloudnativekube.com

Port: xxxxx – in this case it's 31028

Channel Name: IVT.SVRCONN

Username – not required

Password – not required

Queue Manager - <the_helm_release_name> or whatever you chose for the queue manager name

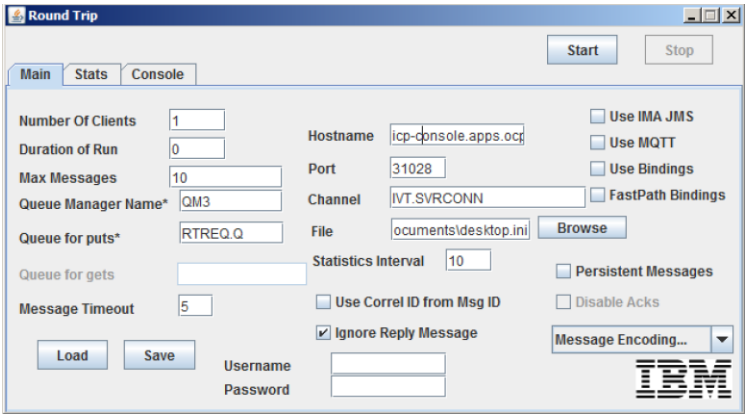
Queue Name - RTREQ.Q

Using the RoundTrip GUI

Unzip the RoundTrip.zip into it's own directory.

Run the rt.sh script. You may need to modify the script to suit your own path to the java executable. See the appendix for the equivalent rt.bat file for windows users .

The UI should appear :



Fill in the hostname, port, channel and Queue Manager Names outlined above with your values.

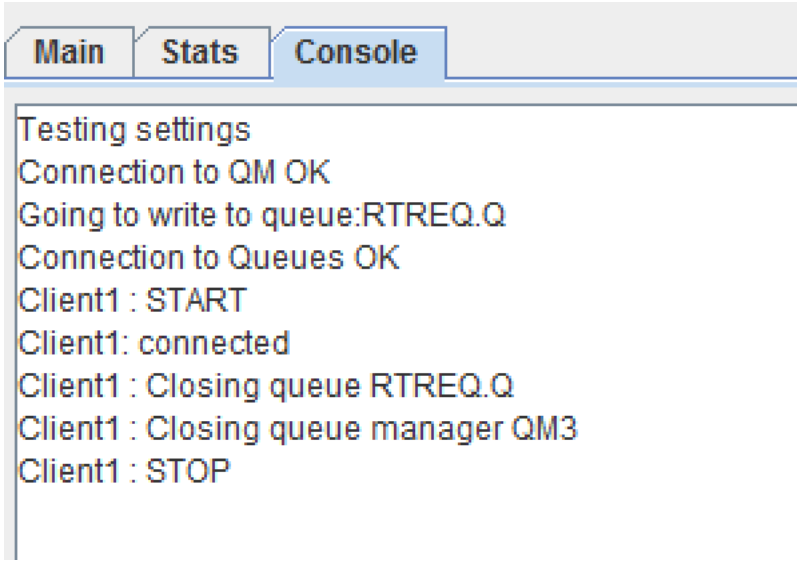
For a simple test, check 'Ignore Reply Message' and set the 'Queue for Puts' to 'RTREQ.Q'

Set Max Messages to 10.

Click on 'Browse' and select a message file to load for the payload data (small file is best)

Check the console tab and click 'Start'.

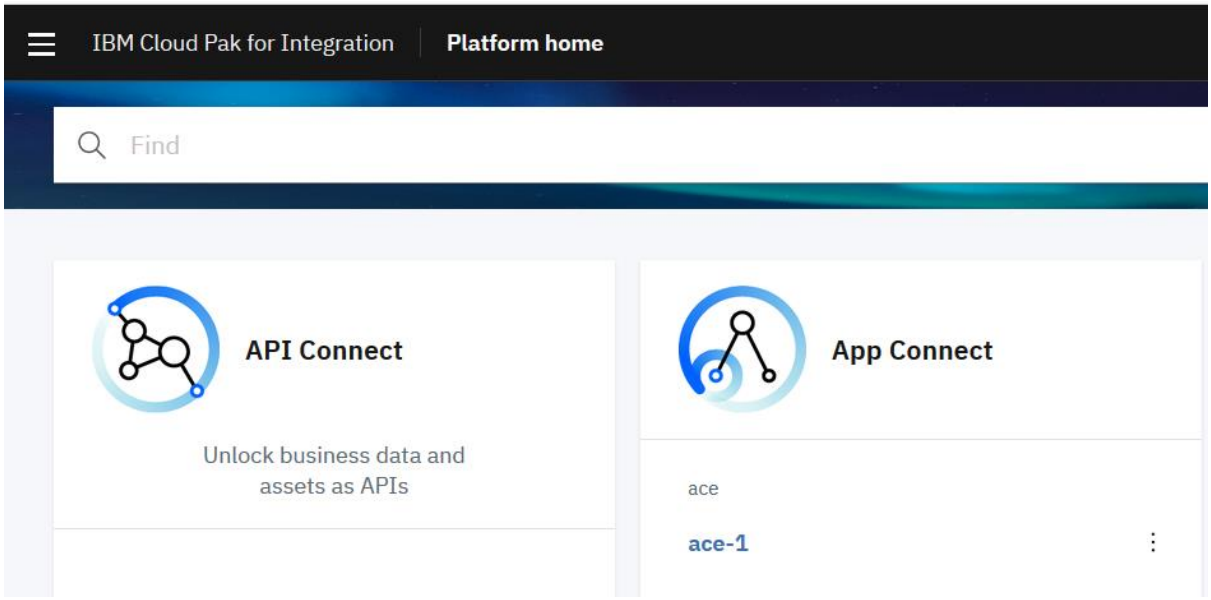
You should see something similar to:



Or you may see an error stack trace if something was not configured correctly.

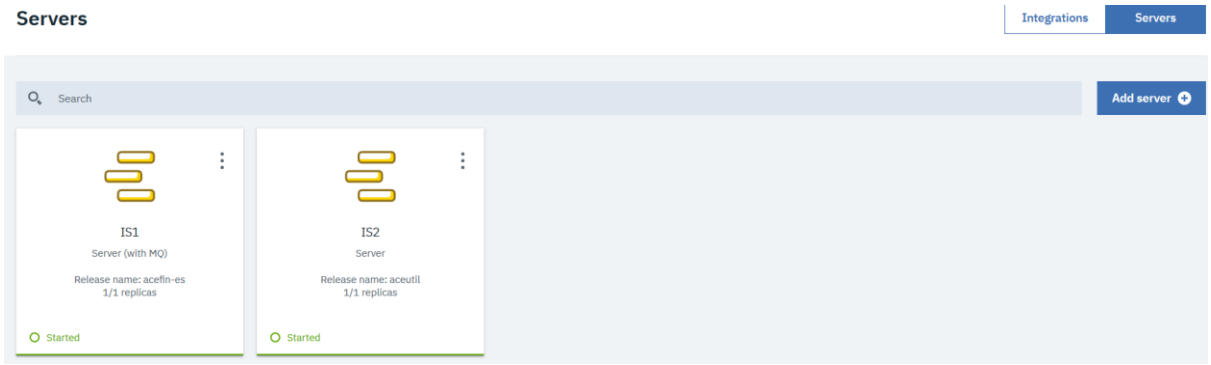
You can click ‘Save’ to save the configuration to file for quick loading (use ‘Load’) when re-testing.

ACE+MQ via the ICP4i GUI

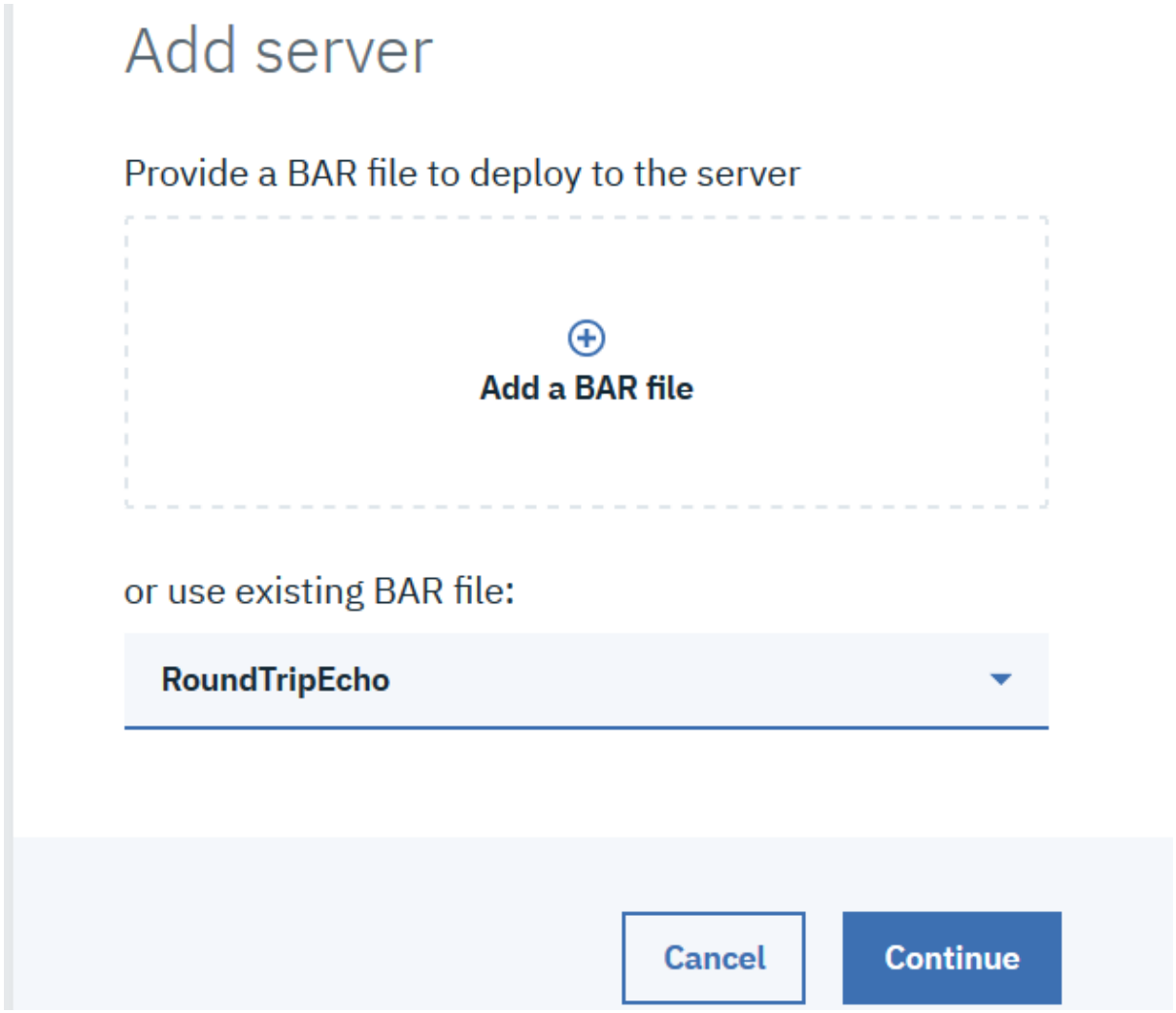


Select ace-1

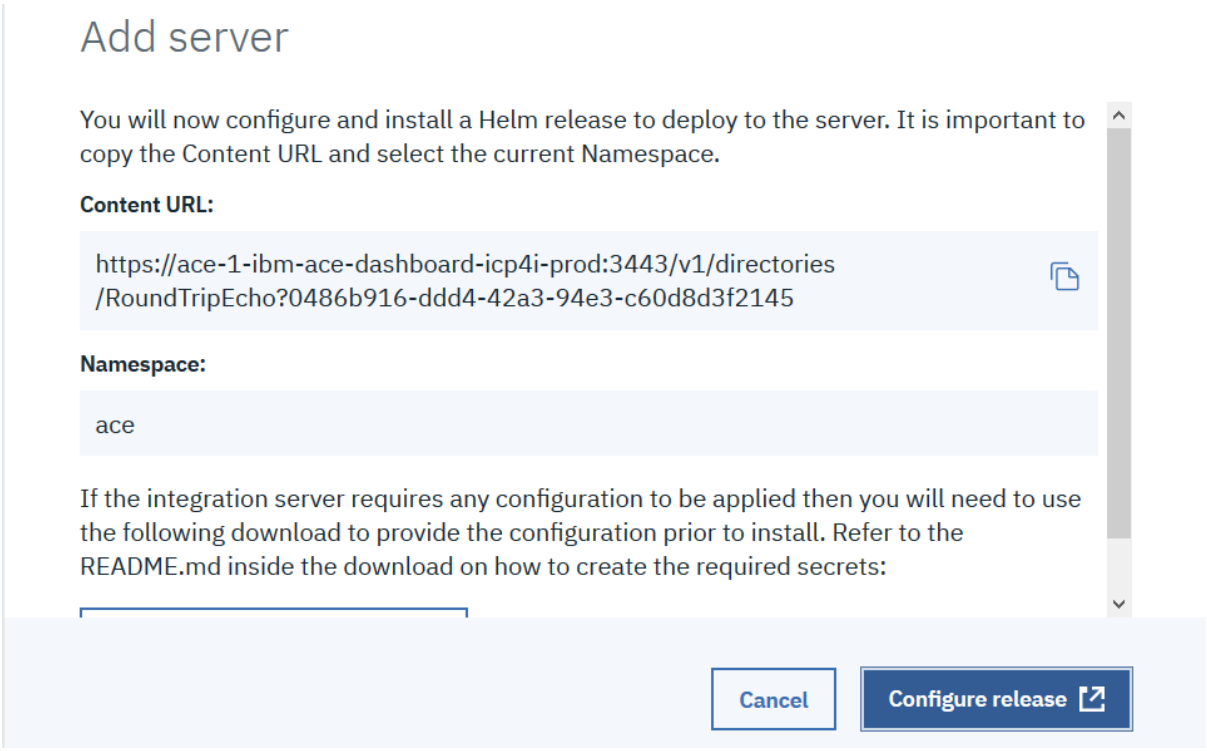
Add server



Select bar



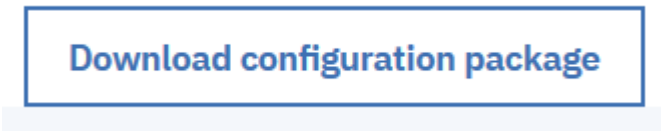
Copy (and paste) the content URL



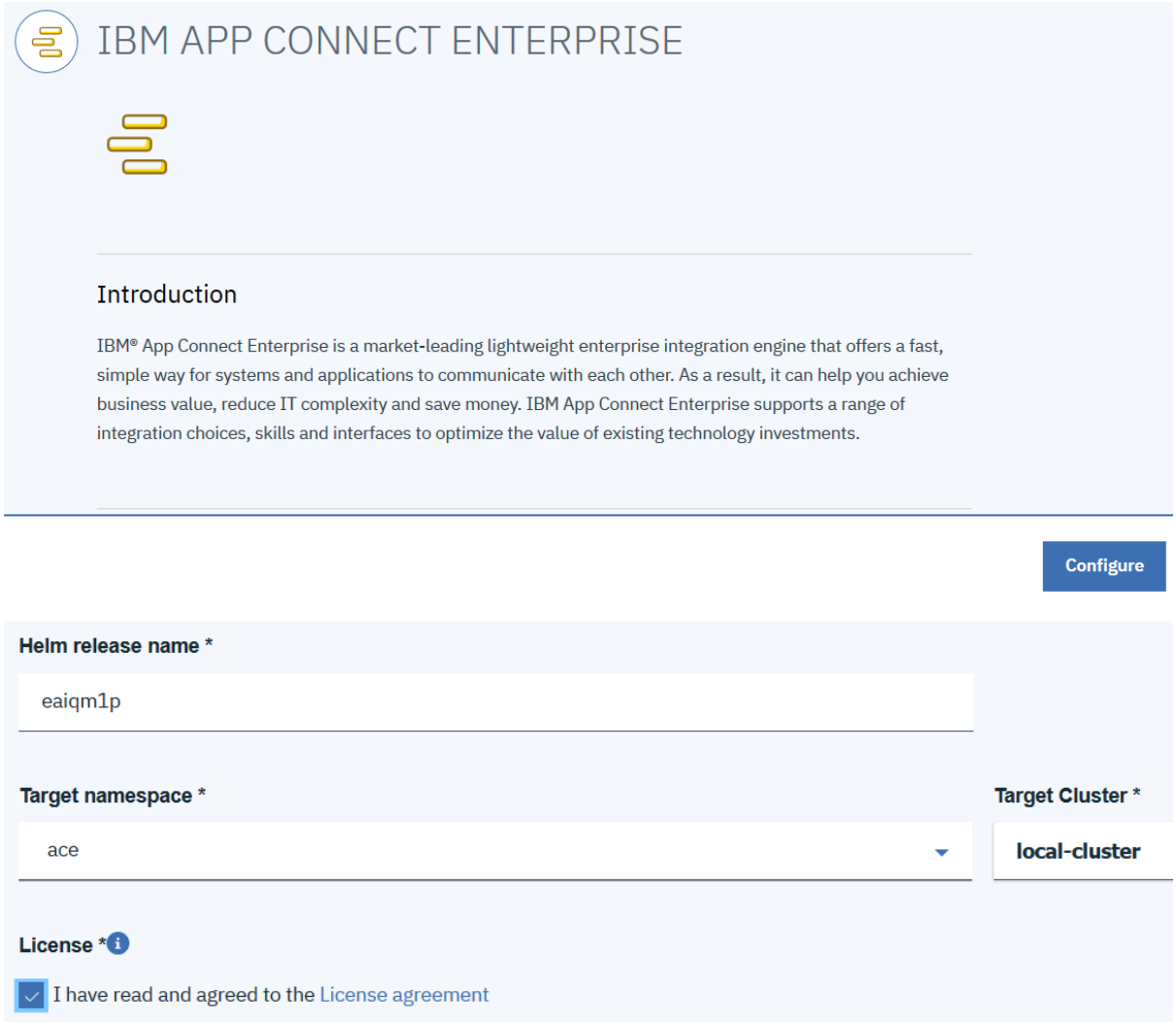
https://ace-1-ibm-ace-dashboard-icp4i-prod:3443/v1/directories/RoundTripEcho?0486b916-ddd4-42a3-94e3-c60d8d3f2145

optional step:

If the integration server requires any configuration to be applied then you will need to use the following download to provide the configuration prior to install. Refer to the README.md inside the download on how to create the required secrets:



Hit Configure release

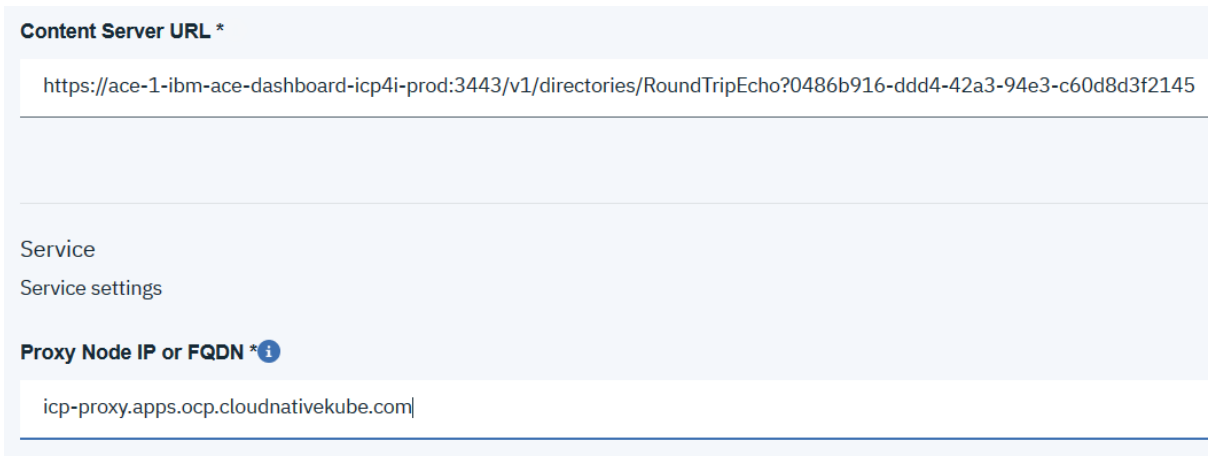


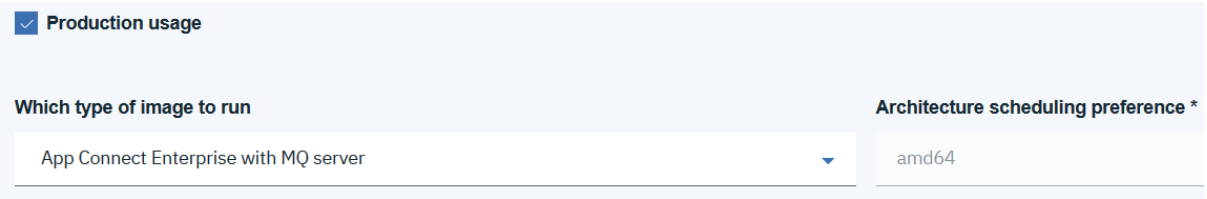
Paste in the content server URL

Proxy Node IP or FQDN

Get this from from the ICP main console –> configuration ->config-maps and filter on info. Find ibmcloud-cluster-info and it's the proxy address. i.e.=

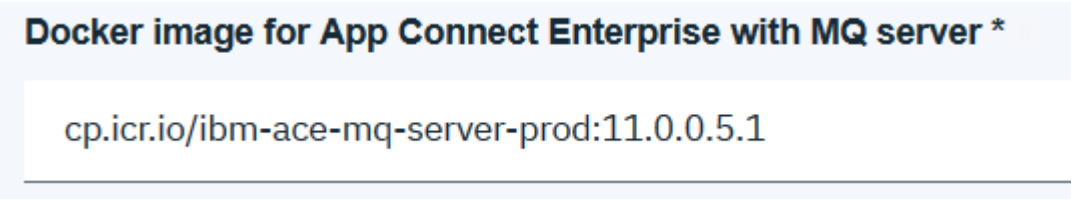
icp-proxy.apps.ocp.cloudnativekube.com



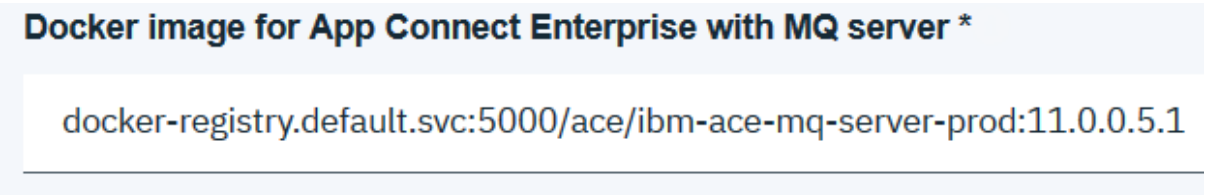


Configure the Image(s) to use the correct registry

Before:



After



You can find all the images and their details in the onboard registry via the OKD console

https://registry-console-default.apps.ocp.cloudnativekube.com/registry

https://registry-console-default.apps.ocp.cloudnativekube.com/registry#/images/ace/ibm-ace-mq-server-prod

on board registry for ace / mq images (rather that the cloud managed service)

replace cp.icr.io with docker-registry.default.svc:5000/ace

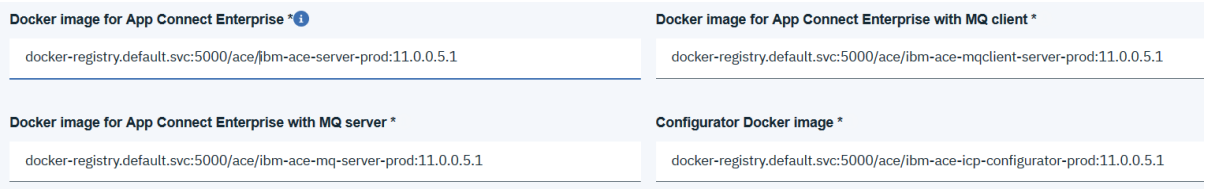
Before

`cp.icr.io/ibm-ace-mq-server-prod:11.0.0.5.1`

After

docker-registry.default.svc:5000/ace/ibm-ace-mq-server-prod:11.0.0.5.1

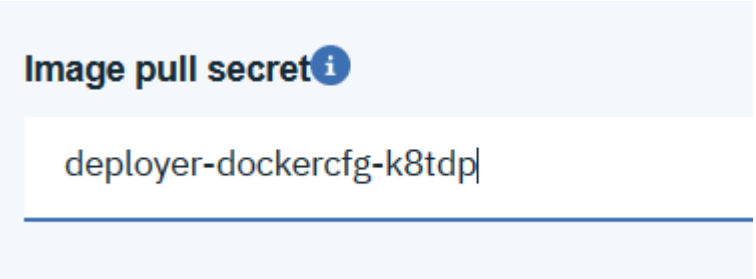
Here's an example of all the possible images changed

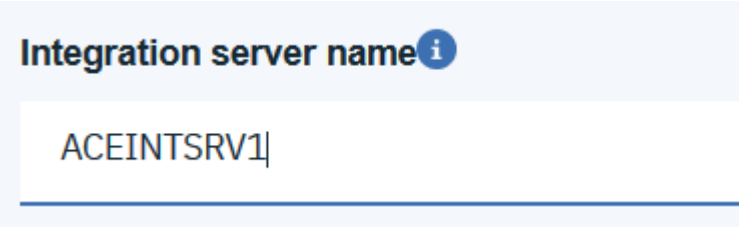


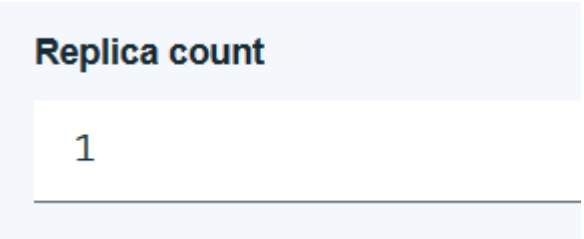
For the image pull secret

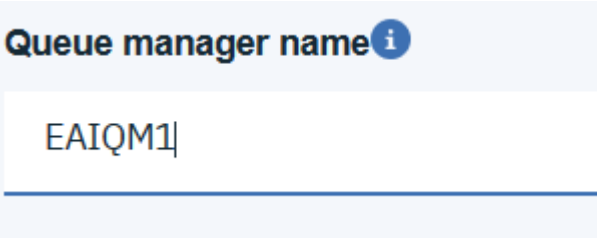
You can use the oc get secrets command

use deployer-dockercfg-k8tdp as image pull secret



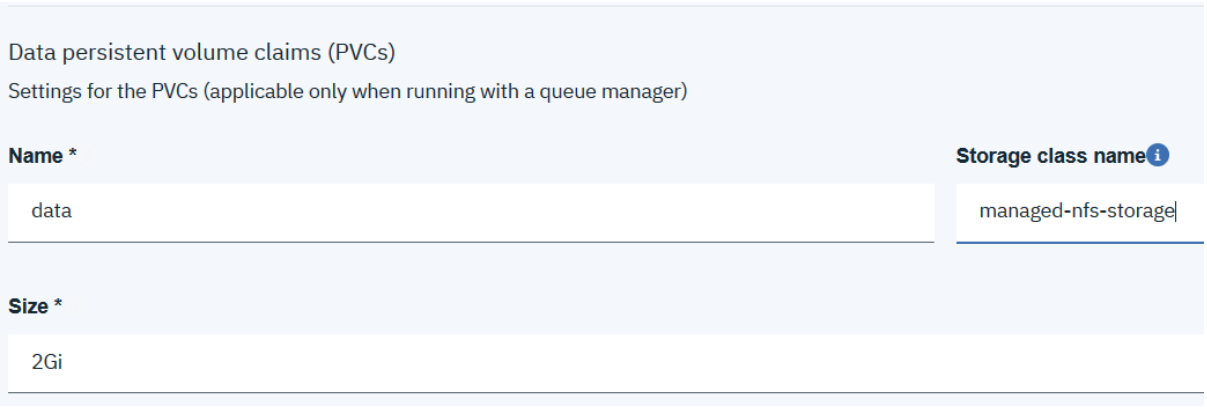




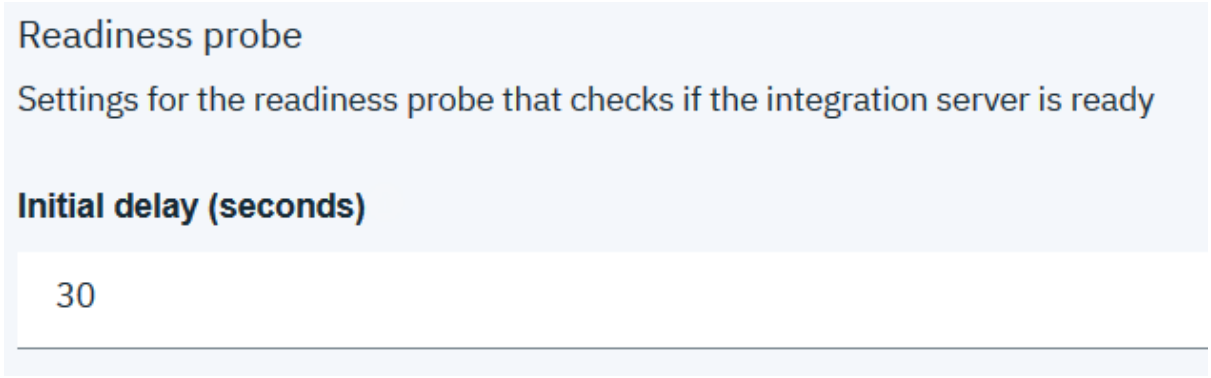


Use oc get storage command to find the storage classes available

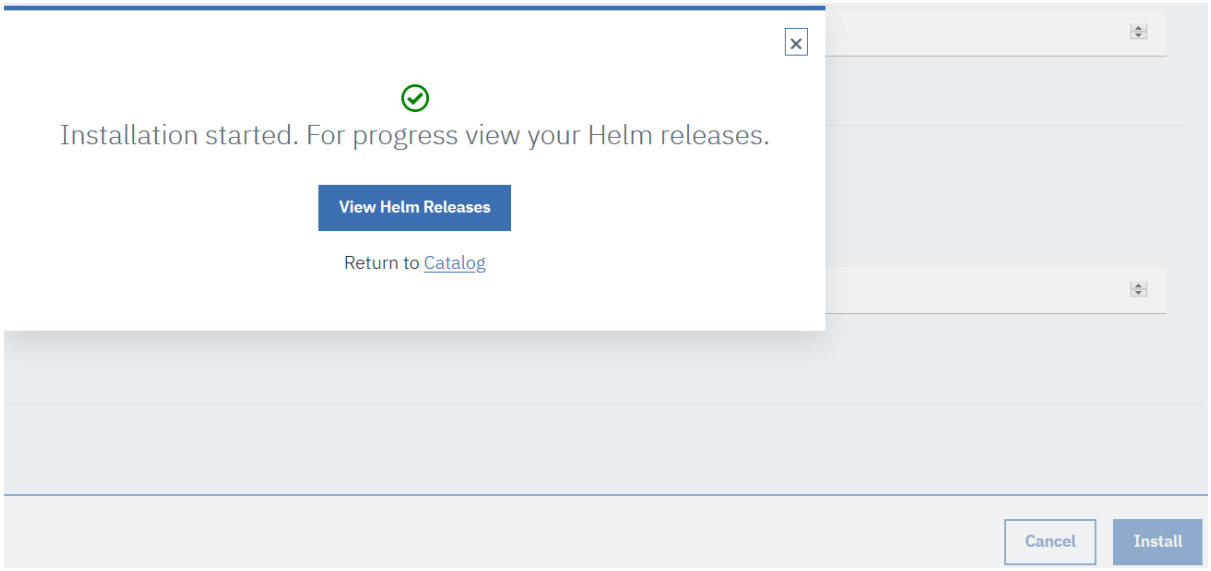
Storage class name = managed-nfs-storage



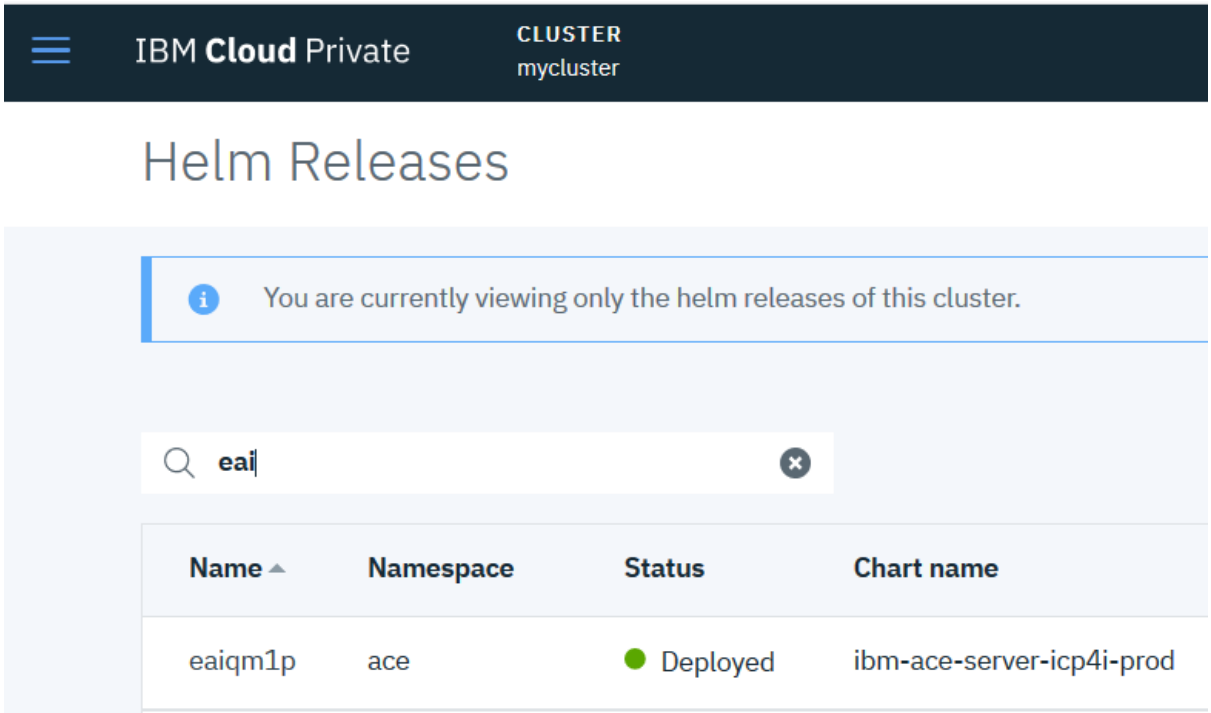
readiness probe. Change from 10 seconds to 30 seconds as things might take a bit longer to start with a queue manager in there too



Click install



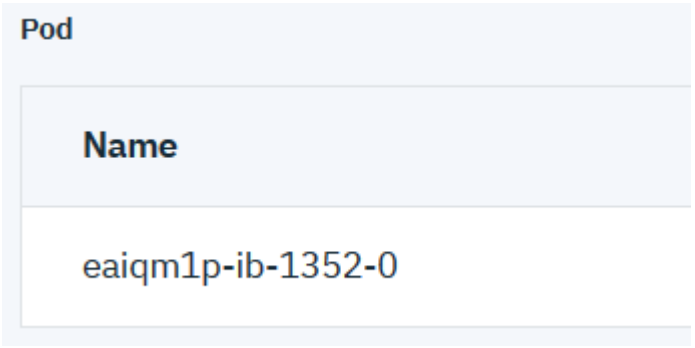
Wait a while and click view helm releases – filter via eai



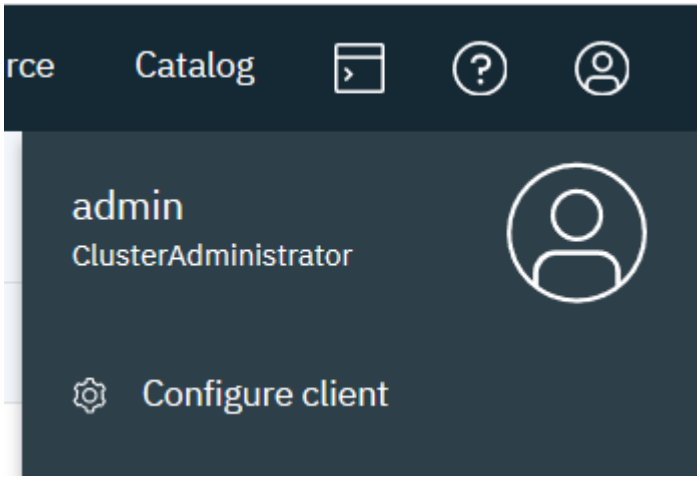
Modifying the queue manager connection security

You can BASH into the pod in order to execute a runmqsc command to enable remote admin because we did not use the configuration package to set a specific mqsc file.

Get the pod name from the helm release



You can also use oc get pods (or kubectl)



Configure client

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

Prerequisites:
Install CLI tools

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

```
kubectl config set-cluster mycluster --server=https://console.apps.ocp.cloudnati
kubectl config set-context mycluster-context --cluster=mycluster
kubectl config set-credentials admin --token=C99UTMlgDHdxoWS_cm0xGmFsundefined7[
kubectl config set-context mycluster-context --user=admin --namespace=default
kubectl config use-context mycluster-context
```

kubectl config set-cluster mycluster --
server=https://console.apps.ocp.cloudnativekube.com:443 --insecure-skip-tls-verify=true

kubectl config set-context mycluster-context --cluster=mycluster
kubectl config set-credentials admin --
token=C99UTMlgDHdxoWS_cm0xGmFsundefined7D_mxYfHYHOa758aPAO
kubectl config set-context mycluster-context --user=admin --namespace=default
kubectl config use-context mycluster-context

You can use the token in the oc login on the command line

**oc login https://master.ocp.cloudnativekube.com:8443 --
token=C99UTMlgDHdxoWS_cm0xGmFsundefined7D_mxYfHYHOa758aPAO**

switch projects

oc project ace

get the pods

oc get pods

```
TO MQ REPOS M NODE\ATO-MQREPOSM-NODE2-CHLMQSC>oc get pods
NAME                                              READY     STATUS    RESTARTS   AGE
ace-1-ibm-ace-dashboard-icp4i-prod-6c4cff67d9-8px2p   2/2       Running   0          20d
acefin-es-ib-1352-0                               1/1       Running   0          1d
aceutil-ibm-ace-server-icp4i-prod-6c96d85577-8mt6w    1/1       Running   0          4d
eaiqm1p-ib-1352-0                                 1/1       Running   0          33m
```

kubectl exec -it eaiqm1p-ib-1352-0 -- /bin/bash

runmqsc

```
TO MQ REPOS M NODE\ATO-MQREPOSM-NODE2-CHLMQSC>kubectl exec -it eaiqm1p-ib-1352-0 -- /bin/bash
bash-4.2$ runmqsc
5724-H72 (C) Copyright IBM Corp. 1994, 2019.
Starting MQSC for queue manager EAIQM1.
```

Cut and paste in the following five command one at a time

DEFINE CHANNEL(IVT.SVRCONN) CHLTYPE(SVRCONN)

SET CHLAUTH(IVT.SVRCONN) TYPE(BLOCKUSER) USERLIST(nobody)

ALTER AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) AUTHTYPE(IDPWOS) CHCKCLNT(NONE)
ADOPTCTX(YES)

SET CHLAUTH(IVT.SVRCONN) TYPE (ADDRESSMAP) ADDRESS(*) MCAUSER(‘mqm’)

REFRESH SECURITY TYPE(CONNAUTH)

What out for dodgy ‘ ‘ when cutting and pasting see below – you need to type over the ‘’ in the bash

Note might be required DEFINE CHANNEL(IVT.SVRCONN) CHLTYPE(SVRCONN) MCAUSER('mqm')

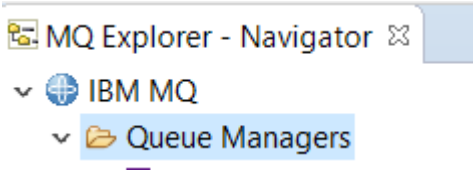
```
Starting MQSC for queue manager EAIQM1.


DEFINE CHANNEL(IVT.SVRCONN) CHLTYPE(SVRCONN)
     1 : DEFINE CHANNEL(IVT.SVRCONN) CHLTYPE(SVRCONN)
AMQ8014I: IBM MQ channel created.
SET CHLAUTH(IVT.SVRCONN) TYPE(BLOCKUSER) USERLIST(nobody)
     2 : SET CHLAUTH(IVT.SVRCONN) TYPE(BLOCKUSER) USERLIST(nobody)
AMQ8877I: IBM MQ channel authentication record set.
ALTER AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) AUTHTYPE(IDPWOS) CHCKCLNT(NONE) ADOPTCTX(YES)
     3 : ALTER AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) AUTHTYPE(IDPWOS) CHCKCLNT(NONE) ADOPTCTX(YES)
AMQ8567I: IBM MQ authentication information changed.
SET CHLAUTH(IVT.SVRCONN) TYPE (ADDRESSMAP) ADDRESS(*) MCAUSER('mqm')
     4 : SET CHLAUTH(IVT.SVRCONN) TYPE (ADDRESSMAP) ADDRESS(*) MCAUSER('mqm')
AMQ8405I: Syntax error detected at or near end of command segment below:-
SET CHLAUTH(IVT.SVRCONN) TYPE (ADDRESSMAP) ADDRESS(*) MCAUSER(

AMQ8427I: Valid syntax for the MQSC command:

  SET CHLAUTH( channel_profile )
    TYPE( ADDRESSMAP )
    ADDRESS( string )
  [ DESCR( string ) ]
  [ CHCKCLNT( ASQMGR | REQDADM | REQUIRED ) ]
  [ CUSTOM( string ) ]
  [ USERSRC( CHANNEL | MAP | NOACCESS ) ]
  [ MCAUSER( string ) ]                    [ WARN( NO | YES ) ]
  [ ACTION( ADD | REPLACE | REMOVE | REMOVEALL ) ]
SET CHLAUTH(IVT.SVRCONN) TYPE (ADDRESSMAP) ADDRESS(*) MCAUSER('mqm')
     5 : SET CHLAUTH(IVT.SVRCONN) TYPE (ADDRESSMAP) ADDRESS(*) MCAUSER('mqm')
AMQ8877I: IBM MQ channel authentication record set.
REFRESH SECURITY TYPE(CONNAUTH)
     6 : REFRESH SECURITY TYPE(CONNAUTH)
AMQ8560I: IBM MQ security cache refreshed.
```

Type end to end runmqsc

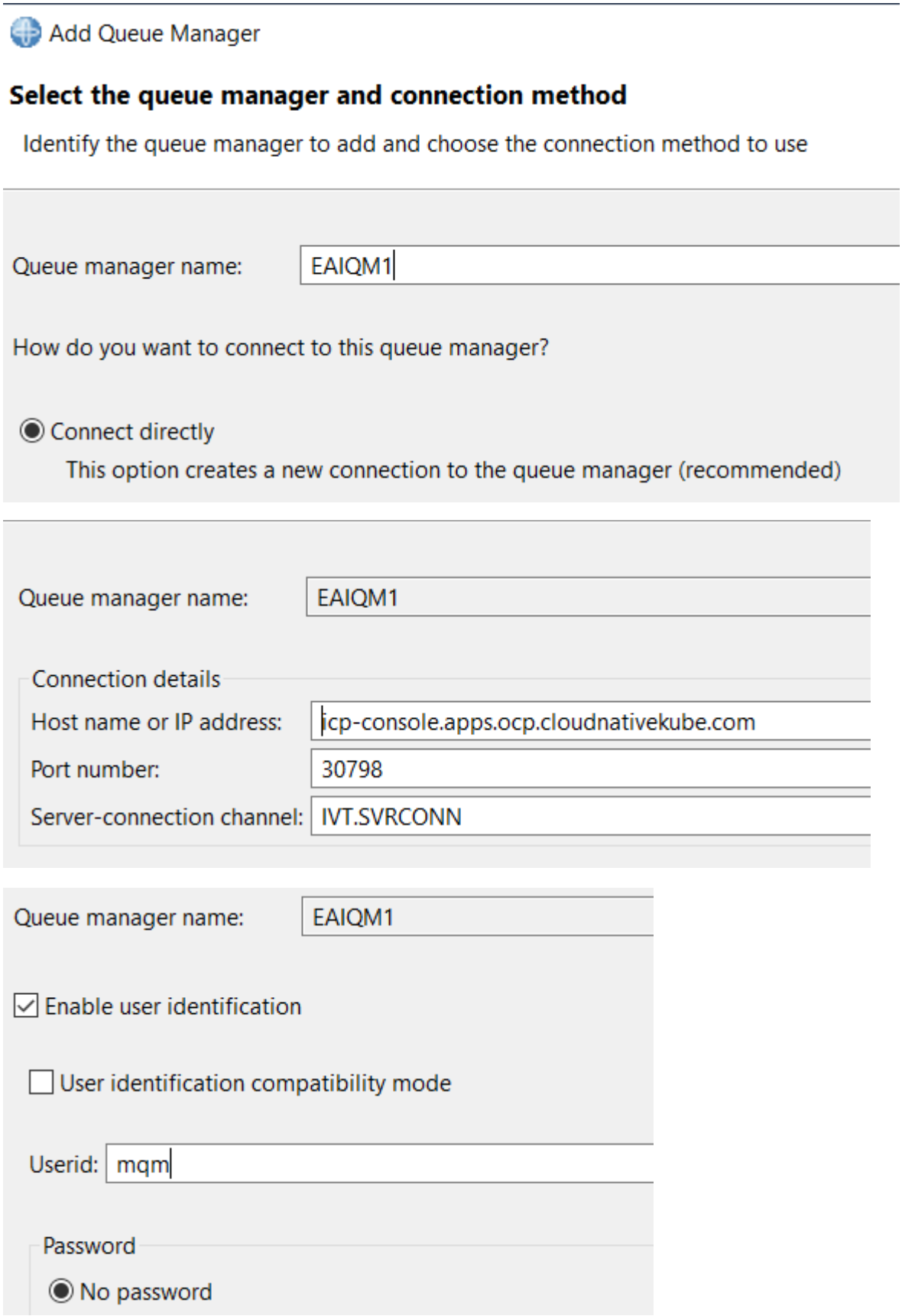
```
end
     7 : end
6 MQSC commands read.
One command has a syntax error.
All valid MQSC commands were processed.
bash-4.2$
```

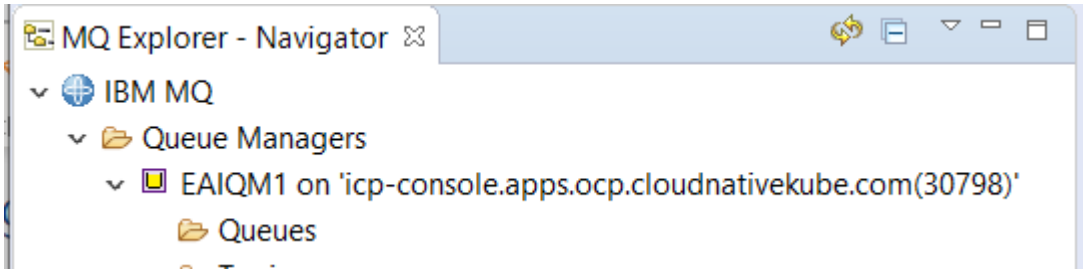
Exit to exit the bash

Try connecting MQ Explorer



Click on Queue Managers and select add remote queue manager


Add Queue Manager
Select the queue manager and connection method
Identify the queue manager to add and choose the connection method to use
Queue manager name:
EAIQM1
How do you want to connect to this queue manager?
Connect directly
This option creates a new connection to the queue manager (recommended)
Queue manager name:
EAIQM1
Connection details
Host name or IP address:
icp-console.apps.ocp.cloudnativekube.com
Port number:
30798
Server-connection channel:
IVT.SVRCONN
Queue manager name:
EAIQM1
Enable user identification
User identification compatibility mode
Userid:
mqm
Password
No password



Point of note

I could not get into the queue manager with

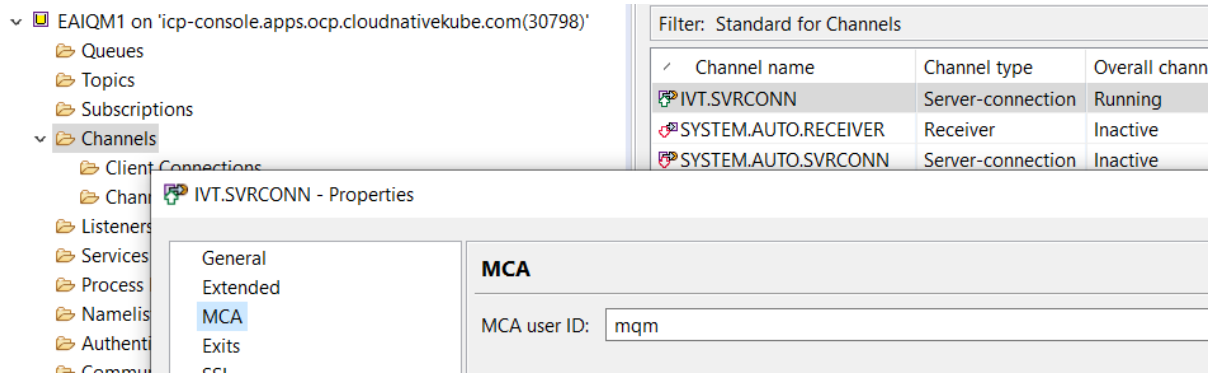
SET MQSERVER=IVT.SVRCONN/TCP/icp-console.apps.ocp.cloudnativekube.com(30798)

and

runmqsc -c EAIQM1

until I put mqm in the MCA user ID on the IVT.SVRCONN

(I also deleted the * and another channel auth record – first but that didn't help)



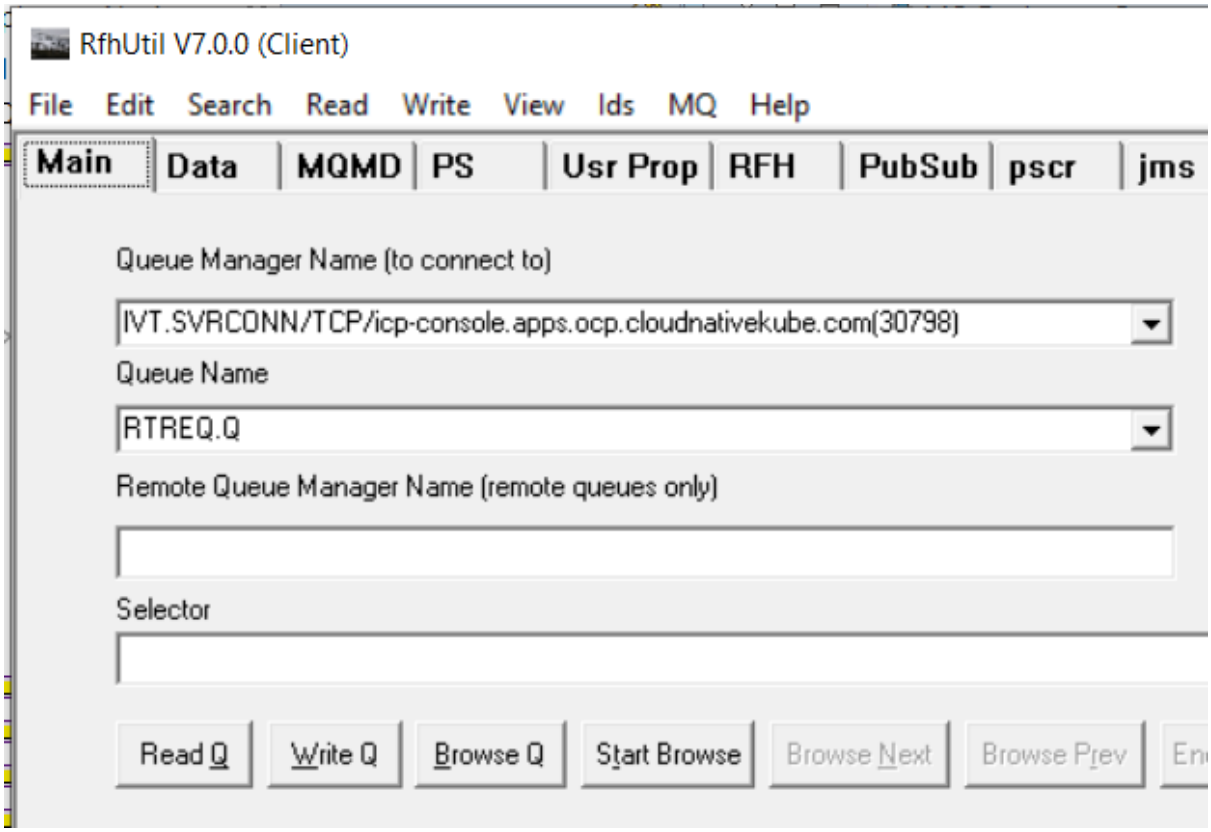
In a second attempt in the bash runmqsc I added the MCAUSER and that allows runmqsc -c to connect

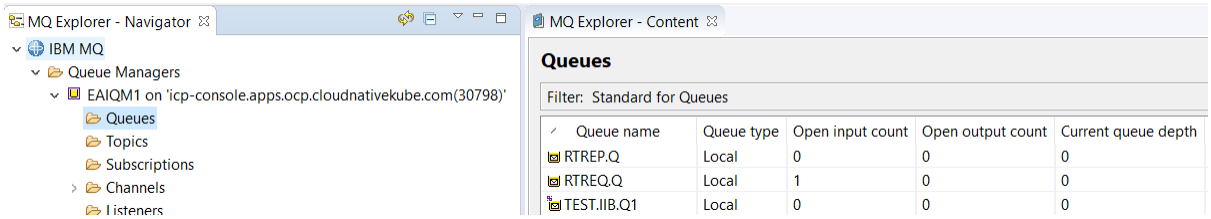
Note might be required DEFINE CHANNEL(IVT.SVRCONN) CHLTYPE(SVRCONN) MCAUSER('mqm')

Connecting RFHUTILC to test

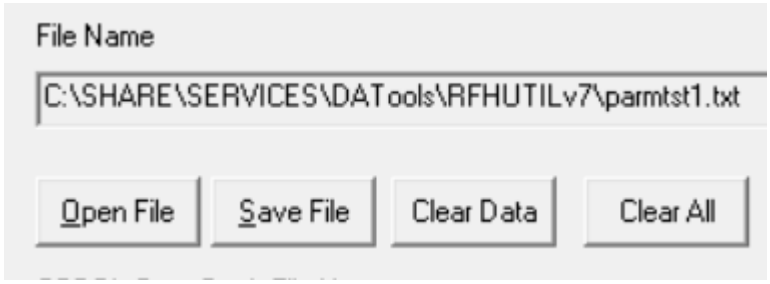
To connect to EAIQM1

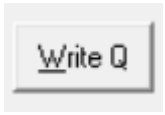
IVT.SVRCONN/TCP/icp-console.apps.ocp.cloudnativekube.com(30798)





Open a file to load some data





11.42.15 Message sent to RTREQ.Q length=3455

Queues

Filter: Standard for Queues

Queue name	Queue type	Open input count	Open output count	Current queue depth
RTREP.Q	Local	0	1	1
RTREQ.Q	Local	1	1	0

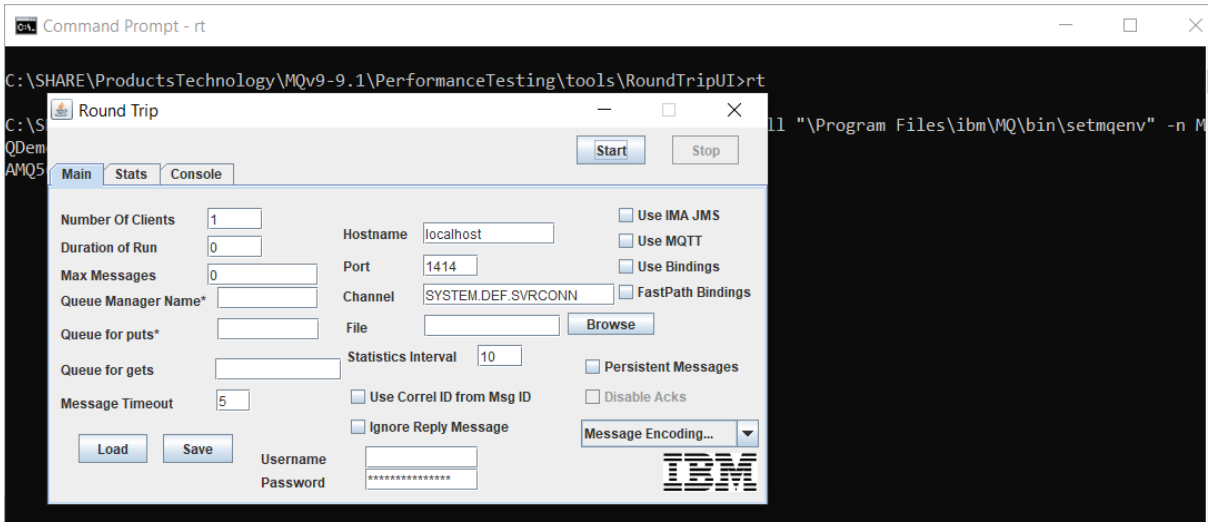
Using the RoundTrip GUI

Unzip the RoundTrip.zip into it's own directory.

Run the rt.sh script. You may need to modify the script to suit your own path to the java executable. See the appendix for the equivalent rt.bat file for windows users .

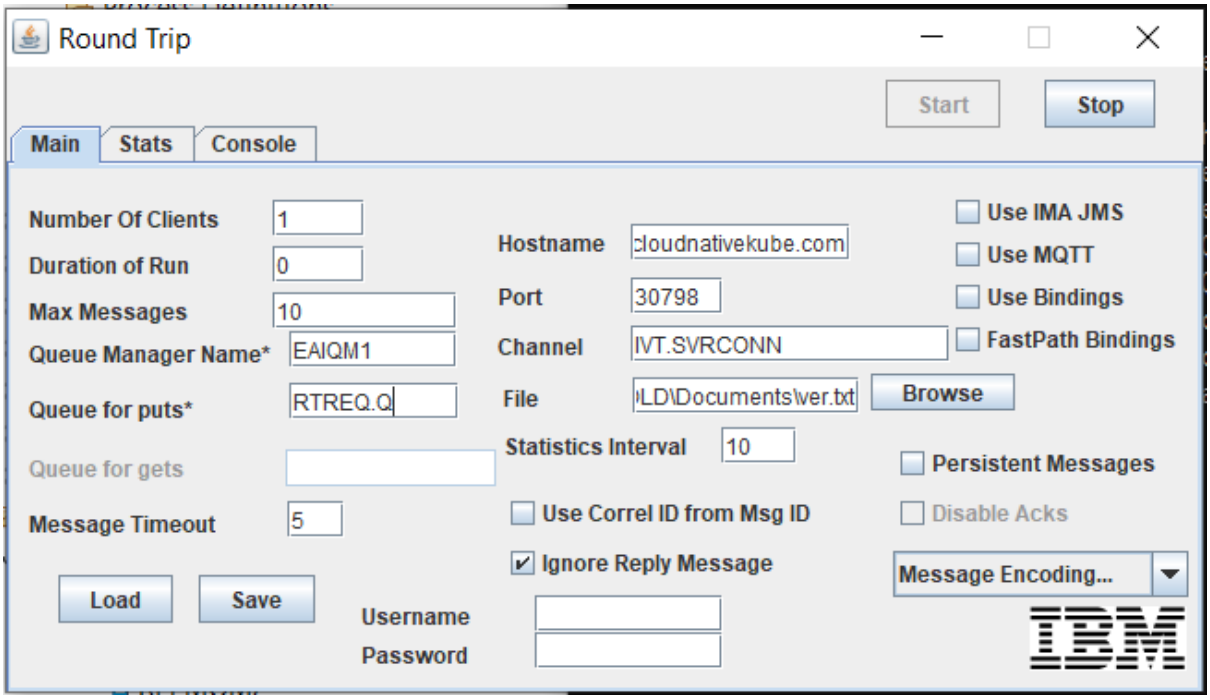
RT.BAT for windows

```
call "\Program Files\ibm\MQ\bin\setmqenv" -n MQDemo1 -x 64


rem "c:\Program Files\ibm\MQ\java\jre\bin\java" -classpath
.\connector.jar;.\fscontext.jar;.\com.ibm.mq.jar;.\com.ibm.mq.jmqi.jar;.\connector.jar;.\com.ibm.mq.headers.jar;.\com.ibm.mq.commonservices.jar;.\jms.jar;.\jndi.jar;.\imaclientjms.jar;.\jms.jar;.\GUIRoundTrip.jar;org.dyno.visual.swing.jar;org.dyno.visual.swing.layouts.jar
com.ibm.mq.testclient.RoundTripF

"c:\Program Files\ibm\MQ\java\jre\bin\java" -classpath
.\com.ibm.mq.allclient.jar;.\jms.jar;.\jndi.jar;.\imaclientjms.jar;.\jms.jar;.\GUIRoundTrip.jar;org.dyno.visual.swing.jar;org.dyno.visual.swing.layouts.jar com.ibm.mq.testclient.RoundTripF
```



The UI should appear :

Using RoundTrip for One-way testing



To connect to EAIQM1

Channel: IVT.SVRCONN

Hostname: icp-console.apps.ocp.cloudnativekube.com

Port: 30798

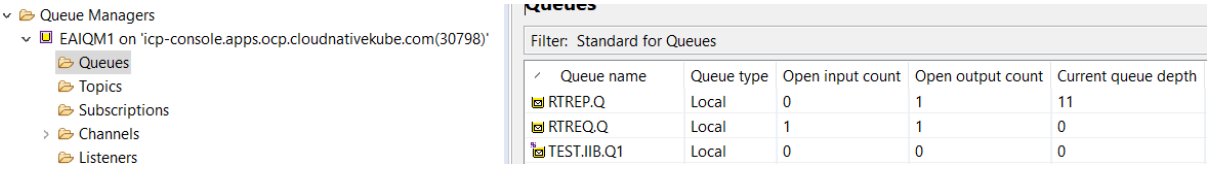
Fill in the hostname, port, channel and Queue Manager Names outlined above with your values.

For a simple test, check ‘Ignore Reply Message’ and set the ‘Queue for Puts’ to ‘RTREQ.Q’

Set Max Messages to 10.

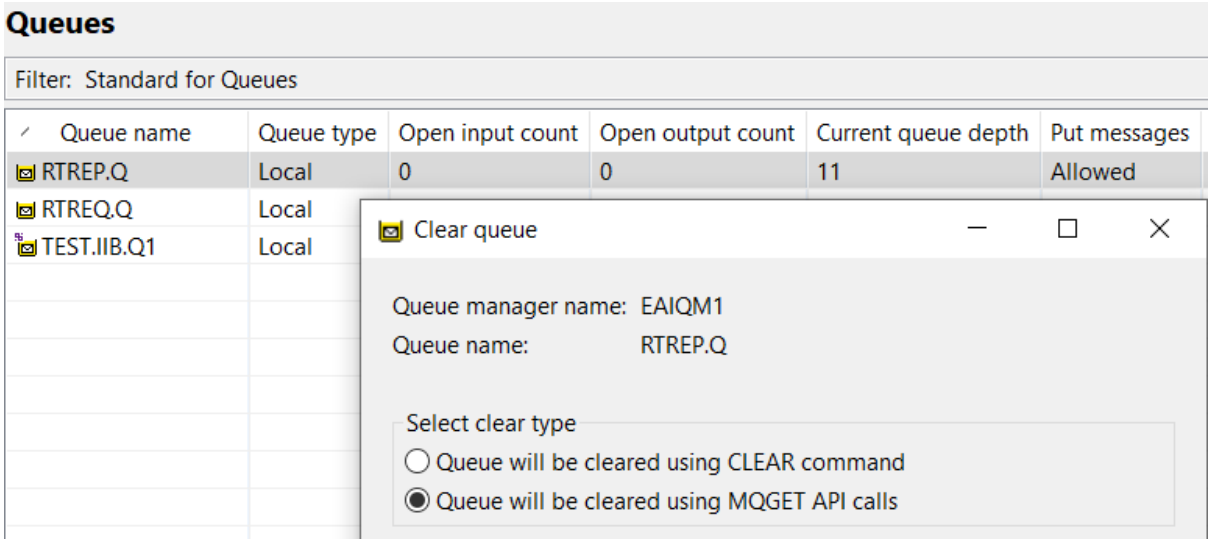
Click on ‘Browse’ and select a message file to load for the payload data (small file is best)

Check the console tab and click ‘Start’.

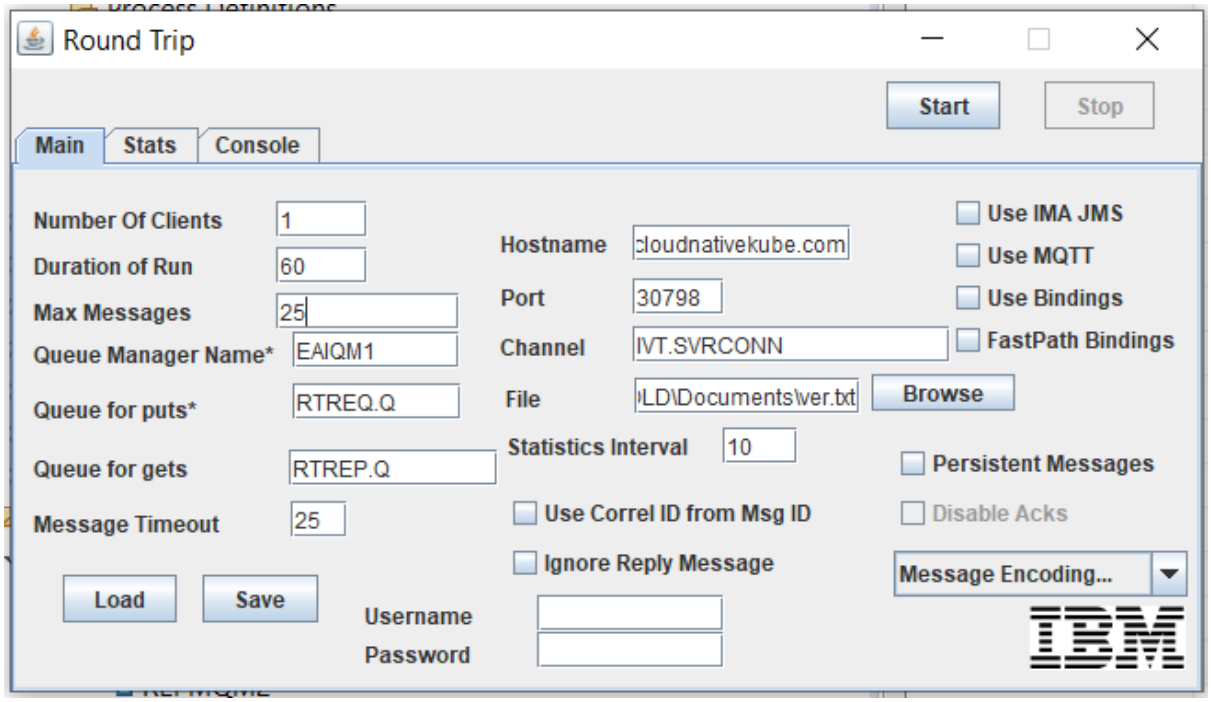


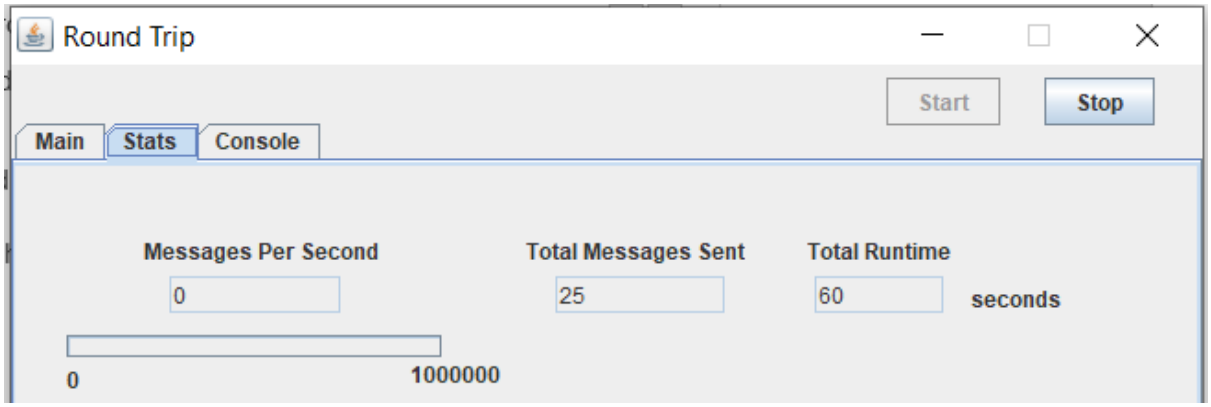
Result should be (1 message already on queue from RFHUTIL) 11 messages on the RTREP.Q as the ACEINTSRV1 RoundTrip echo flow is moving messages from RTREQ.Q to RTREP.Q

Clear the messages

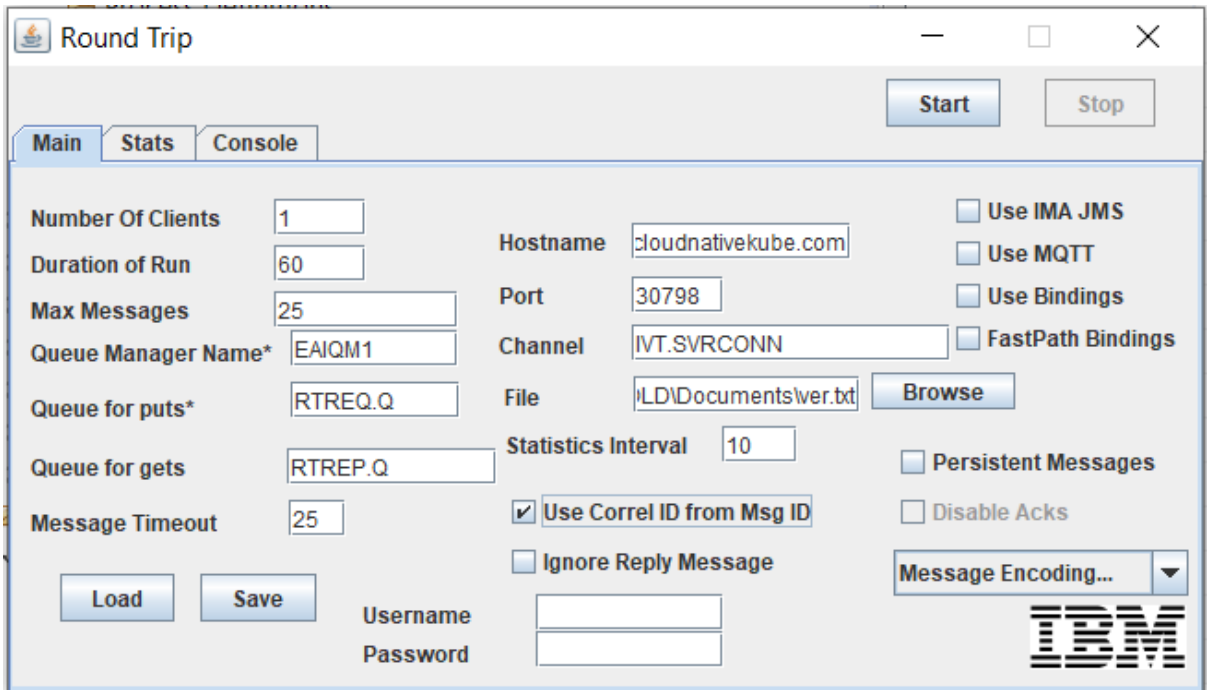


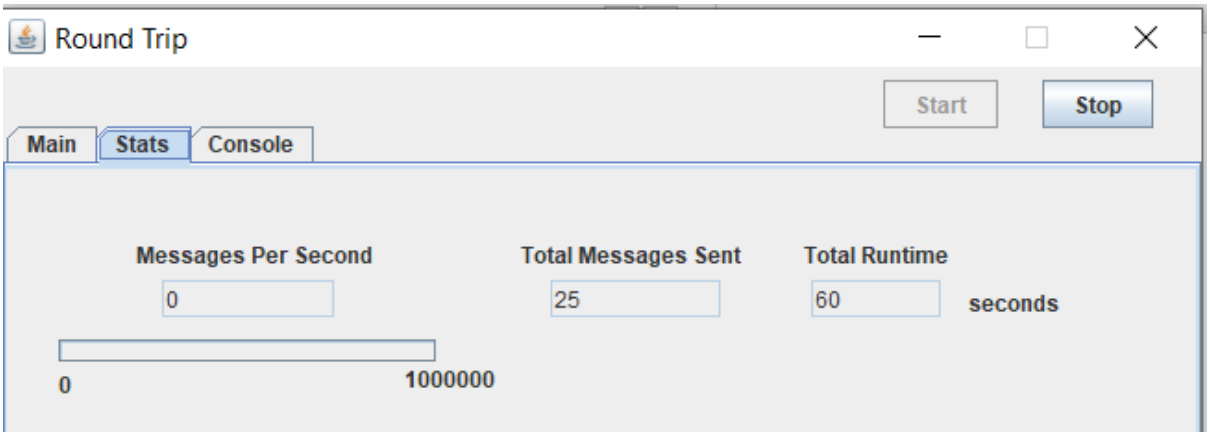
Using RoundTrip for RoundTrip testing – no correlID matching



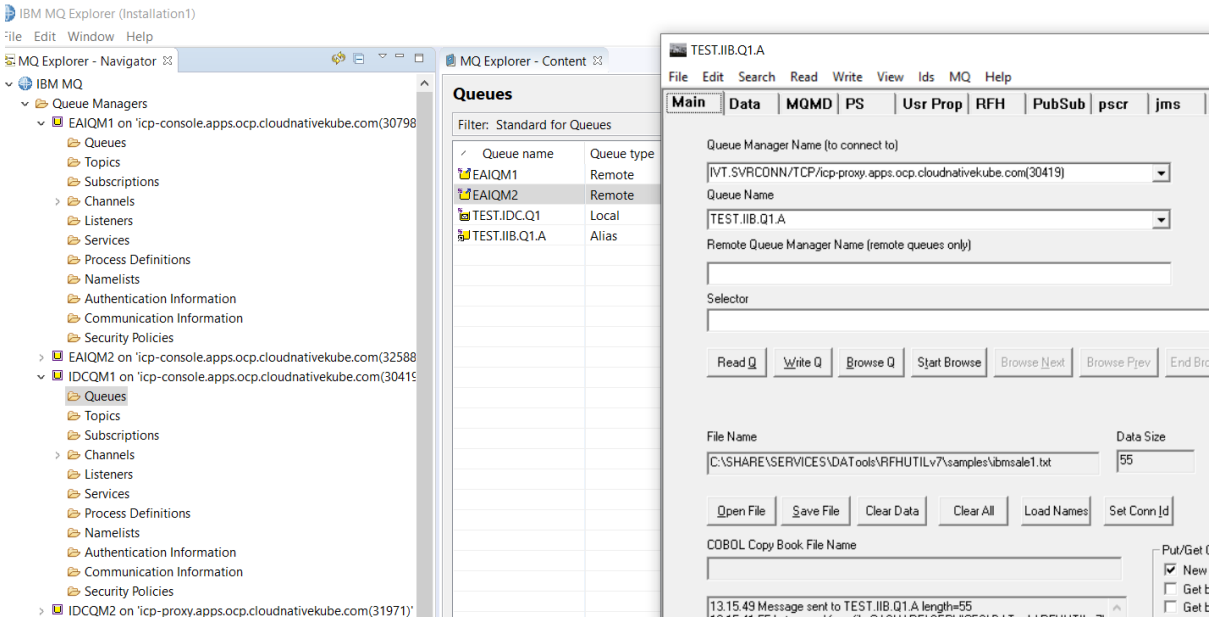


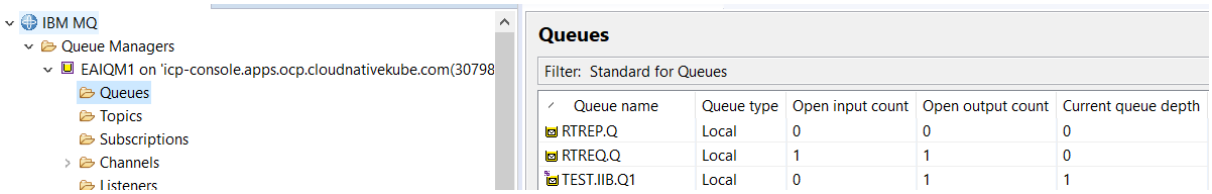
Using RoundTrip for RoundTrip testing – correlID matching





Routing Messages through IDC to EAI





Deploying API Connect for IBM Cloud Pak for Integration

The deployment information in this document is closely aligned with the Residency Playbook for CP4I which can be found here:

https://pages.github.ibm.com/CASE/cloudpak-onboard-residency/integration/deploy-api-mgmt/

Alternatively, there is a slightly older Recipe on DeveloperWorks on Deploying and Configuring IBM API Connect 2018 on ICP4I 2019.3.2.1 (on OCP 3.11)

https://developer.ibm.com/recipes/tutorials/deploying-and-configuring-ibm-api-connect-2018-on-icp4i-2019-3-2-1-on-ocp-3-11

Prepare Endpoints

We have to define the endpoint for each of the APIC subsystems. We can “construct” the endpoints by adding descriptive “prefixes” to the proxy URL

Management – all endpoints

The Management endpoints can all be the same like mgmt.apps.ocp.cloudnativekube.com, but we chose individual names for the endpoints.

Platform API Endpoint

platformapi.apps.ocp.cloudnativekube.com

Consumer API Endpoint

capi.apps.ocp.cloudnativekube.com

Cloud Admin UI

adminui.apps.ocp.cloudnativekube.com

API Manager UI

mgrui.apps.ocp.cloudnativekube.com

Portal endpoints

Portal – director

portaldirector.apps.ocp.cloudnativekube.com

Portal – web

portalweb.apps.ocp.cloudnativekube.com

Analytics endpoints

Analytics – ingestion

analingest.apps.ocp.cloudnativekube.com

Analytics - client

analclient.apps.ocp.cloudnativekube.com

Gateway endpoints

API Gateway

apigw.apps.ocp.cloudnativekube.com

Gateway Service

gwservice.apps.ocp.cloudnativekube.com

Obtain the pull secret

```
oc get secrets -n apic
```

The pull secret starts with deployer-dockercfg, in our case it was:

```
deployer-dockercfg-6mjp9
```

Create the TLS secret.

Setup the CLI environment, and make sure that helm command works correctly, for example run:

```
helm version --tls
```

and make sure that it has the connectivity to the server:

```
Client: &version.Version{SemVer:"v2.12.3",
GitCommit:"eecf22f77df5f65c823aacd2dbd30ae6c65f186e", GitTreeState:"clean"}

Server: &version.Version{SemVer:"v2.12.3+icp",
GitCommit:"34e12adfe271fd157db8f9745affe84c0f603809", GitTreeState:"clean"}
```

Run the following command to create the secret:

```
kubectl create secret generic apic-ent-helm-tls --from-
file=cert.pem=$HOME/.helm/cert.pem --from-file=ca.pem=$HOME/.helm/ca.pem --
from-file=key.pem=$HOME/.helm/key.pem -n apic
```

where apic-ent-helm-tls is the name of the secret.

Increase vm.max_map_count

To check and increase vm.max_map_count we would need an *ssh* access to each of the cluster nodes.

The alternative is to create a DaemonSet which will do that for us. Prepare the yaml file with the following content:

```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  labels:
    k8s-app: sysctl-conf
  name: sysctl-conf
  namespace: kube-system
spec:
  template:
    metadata:
      labels:
        k8s-app: sysctl-conf
    spec:
      containers:
      - command:
        - sh
        - -c
        - sysctl -w vm.max_map_count=262144 && while true; do sleep 86400;
done
        image: busybox:1.26.2
        name: sysctl-conf
```

```
        resources:
          limits:
            cpu: 10m
            memory: 50Mi
          requests:
            cpu: 10m
            memory: 50Mi
        securityContext:
          privileged: true
      terminationGracePeriodSeconds: 1
```

and run apply it with:

```
oc apply -f sysctl-conf.yaml
```

Storage Class

The block storage class is needed for APIC. You can obtain the class names with

```
oc get storageclass
```

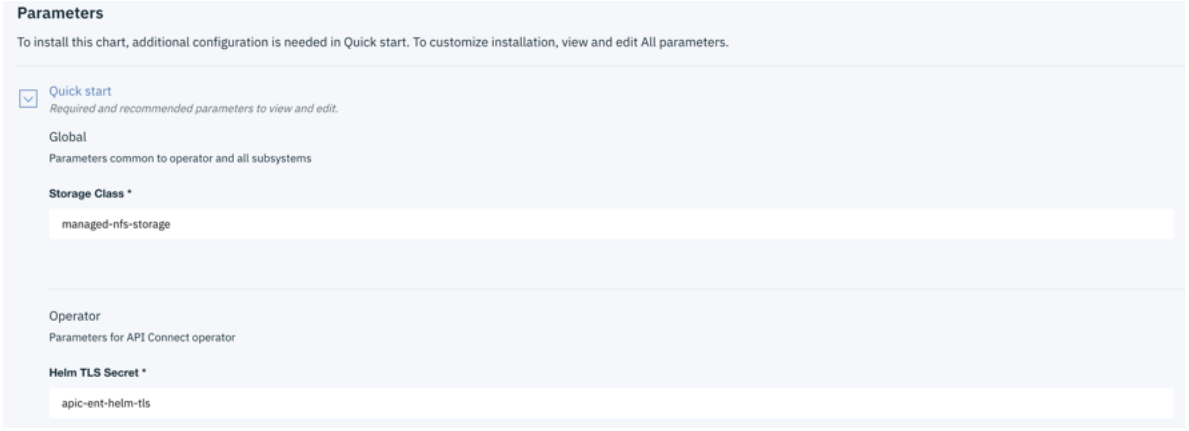
The following class was available on ROKS:

```
managed-nfs-storage (default)    nfs-storage    1d
```

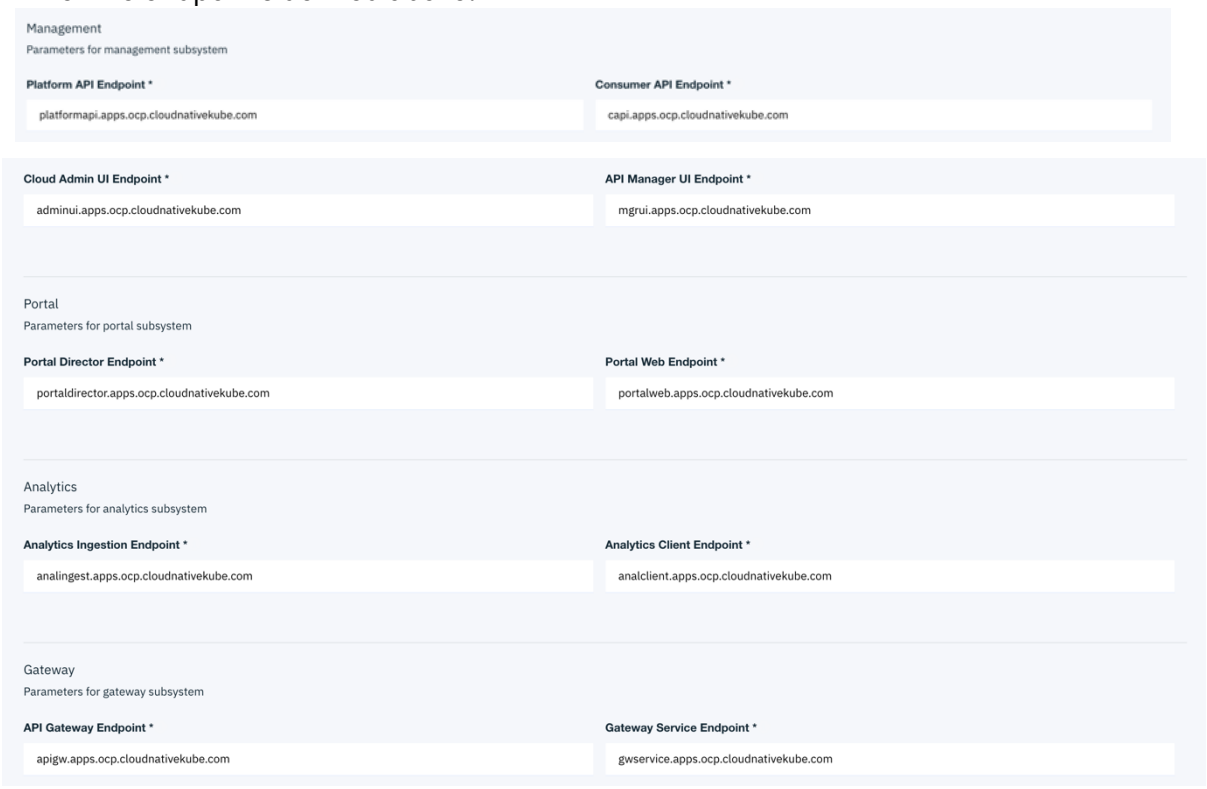
Create an Instance

- Open platform navigator and select API Connect / Add new instance
- Click *Continue*
- Define the helm release name, select apic namespace and the target cluster
    a. apicdemo
    b. apic
    c. local-cluster
- Reduce deployment footprint for test/dev install
    a. Uncheck Production Deployment
    b. Select Mode = dev
    c. Reduce Gateway Cluster Count to 1
    d. Optionally: Reduce the Cassandra Cluster Size to 1
- Uncheck V5 Compatibility Mode

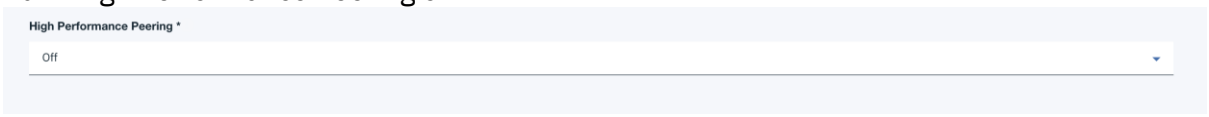
- Enter the helm TLS secret name and select storage class:



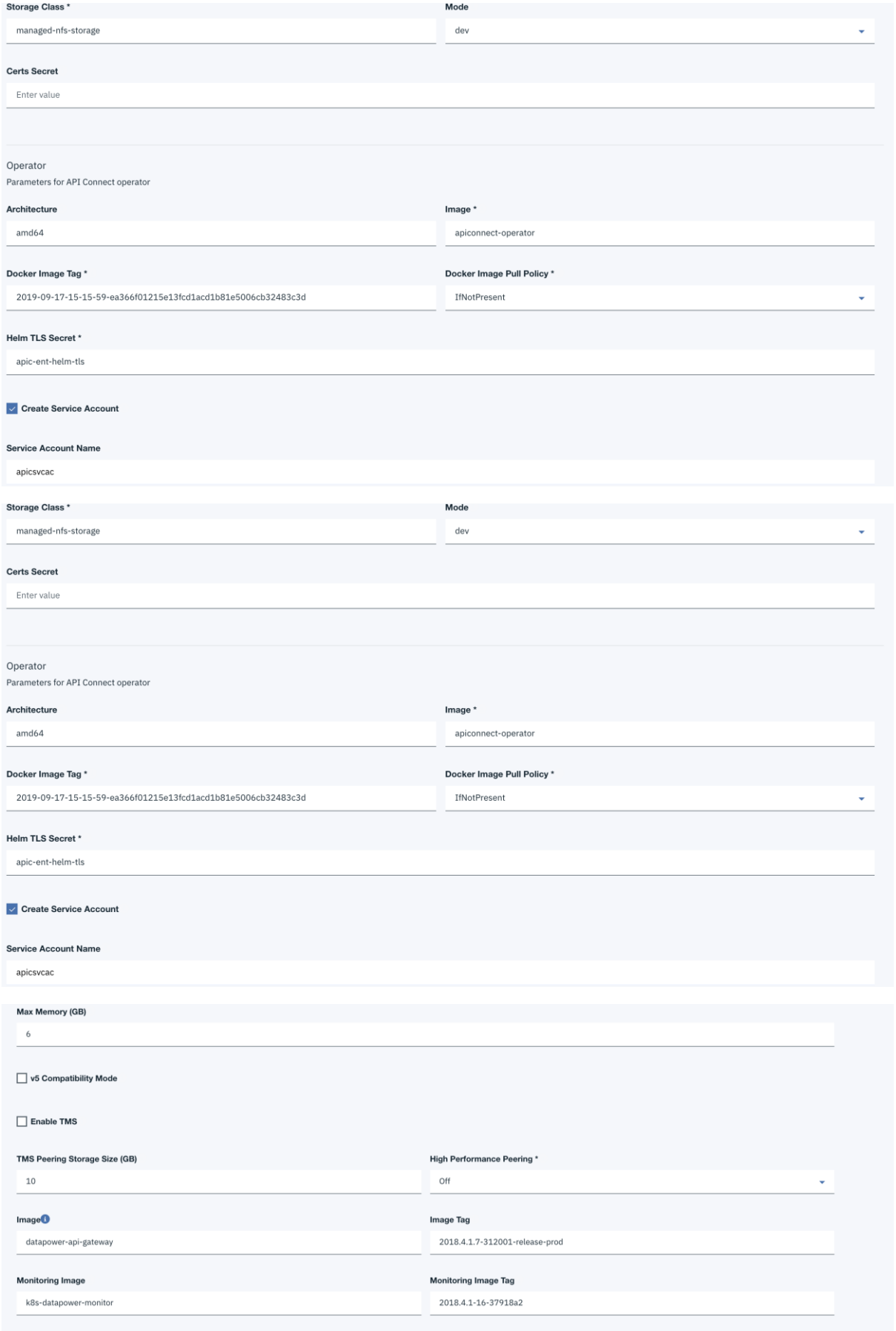
- Enter the endpoints defined above:



- Turn High Performance Peering off



- Lookup the Image name and Docker Image Tags according to the information provided in the Atomic Registry
    a. https://registry-console-default.apps.ocp.cloudnativekube.com/registry
    b. Select Project: apic
    c. Click on following images to get the details
        i. apic/analytics-operator
        ii. apic/apiconnect-operator
        iii. apic/cassandra-operator
    d. Note the Pull Repository Server and Tag
- Enter the information in the subsequent form sections



- Click on **Install**, the confirmation message will appear

- You can check the status of the pods also with the command:

```
oc get pods -n apic
```

- When deployment is completed, all pods must be in Running or Completed state.

SMTP Server

In order to configure the API Connect, we need a SMTP server. If we don't have one, we can run the Mailhog, a fake SMTP server ready for any Kubernetes environment.

- Make sure you are properly connected/logged into the server with oc, cloudctl and helm
- Install Mailhog with:

```
helm install --name mailhog stable/mailhog --tls
```

- The service is of ClusterIP type, so in order to read mails, we must change it to the NodePort, or create Route, or simply port-forward the pod. For example, if the pod name (obtained with `oc get pods -n apic`) is *mailhog-55c8c548dc-dphpj* we can run:

```
kubectl port-forward mailhog-55c8c548dc-dphpj 8025:8025 -n apic
```

and then access mails from the local browser on http://127.0.0.1:8025

Configuring API Connect

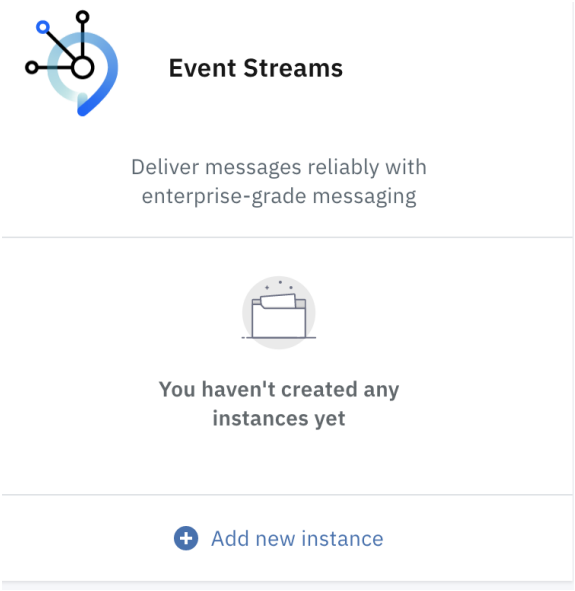
- Open the Cloud Management Console using the previously defined endpoint, in our case it was: https://adminui.apps.ocp.cloudnativekube.com/admin/
- Select IBM Common Services user registry, default username and password in this case are admin/ #######
- Under **Resources/Notifications** define the SMTP server
- For our Mailhog server enter ClusterIP address and port according to the installation response
  - IP: 172.30.233.85
  - Port: 1025
- And select the SMPT server defined under resources:
- Start with the Topology configuration
- Register service:
- Start with the Gateway, select the version that you defined under the Helm release properties when you started creating the instance. In our case it was DataPower API Gateway
- Give some name to the service (e.g. **gateway1**) enter the **endpoints** and click on **Save**
- The confirmation message should appear on the top right of the screen
- Click on *Register service* again and select Analytics
- Give some name to the service, enter Management endpoint (the one that you defined for **analytics client**) and click **Save**
- The confirmation message should appear on the top right of the screen
- Repeat the same with portal:
  - Name: portal1
  - Endpoint: portaldirector.apps.ocp.cloudnativekube.com
  - Website URL: portalweb.apps.ocp.cloudnativekube.com
- The confirmation appears again

- Click on Associate Analytics Service to associate analytics with the gateway
- Select the analytics service
- Click on Provider organizations and add new organization
- Give some name to the organization
    - Title: DemoOrg
- Define the owner
    - Select IBM Common Services user registry
    - Username: admin
- After you submit the organization will appear on the list
- Navigate to the API Manager, in our case the endpoint was: https://mgrui.apps.ocp.cloudnativekube.com/manager/demoorg/
- Login as the owner (defined in the previous step), the API Manager page should open
- Select Gateway Services and assign the gateway to the service
- Navigate to the catalog and then settings
    - Click Portal and click Create

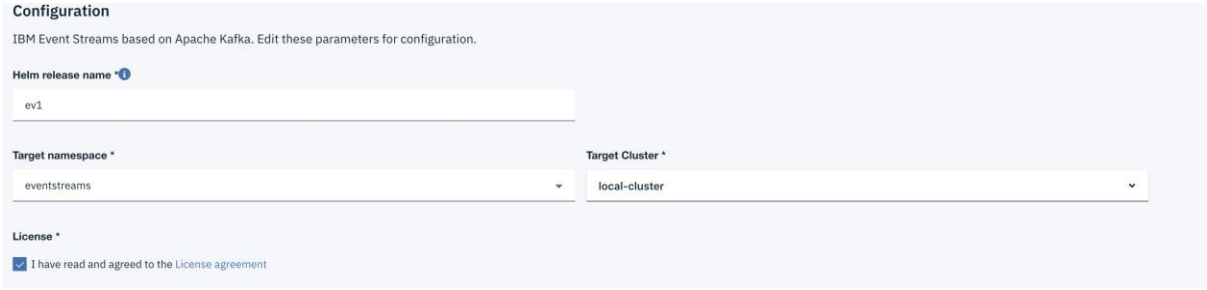
With that, your API Connect instance is ready for usage!

Creating an Event Streams Environment

Open the Integration Navigator and select Create a New Instance from the Event Streams tile

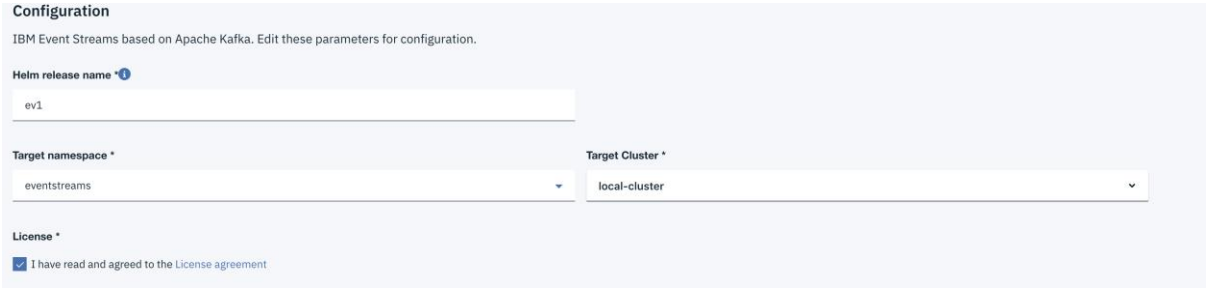


Give the new instance a name, select the EventStreams Namespace and select ‘Local Cluster’

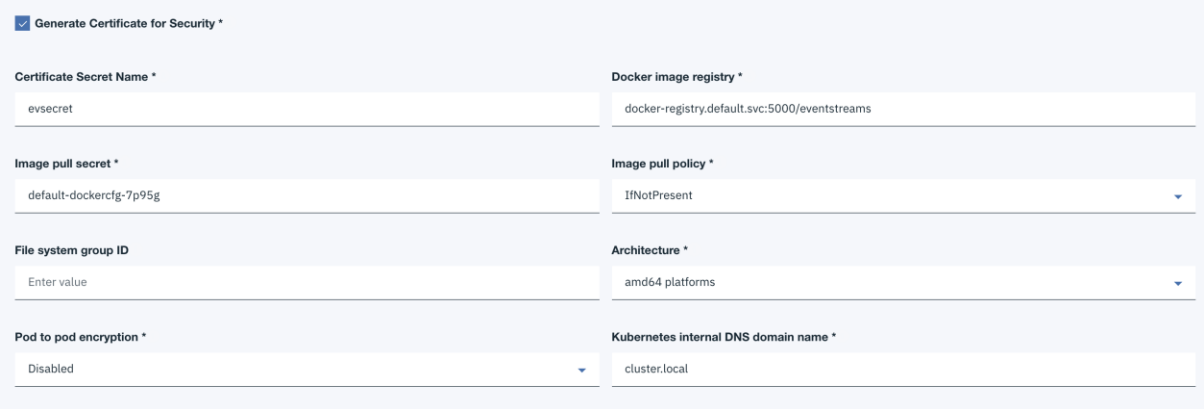


Create a new Cert secret name (Can be anything) and use the command ‘OC get secret’ from the EventStreams namespace or project tp determine the image pull secret.

Choose an external hostname for the instance (we’re using the default proxy entry point here)



Enter the Docker image registry and leave all other fields as default.



Leave all other fields at their default values and Click install.

Wait for all the pods to come up. Hint use ‘oc gety pods’ in the Eventstreams namespace.

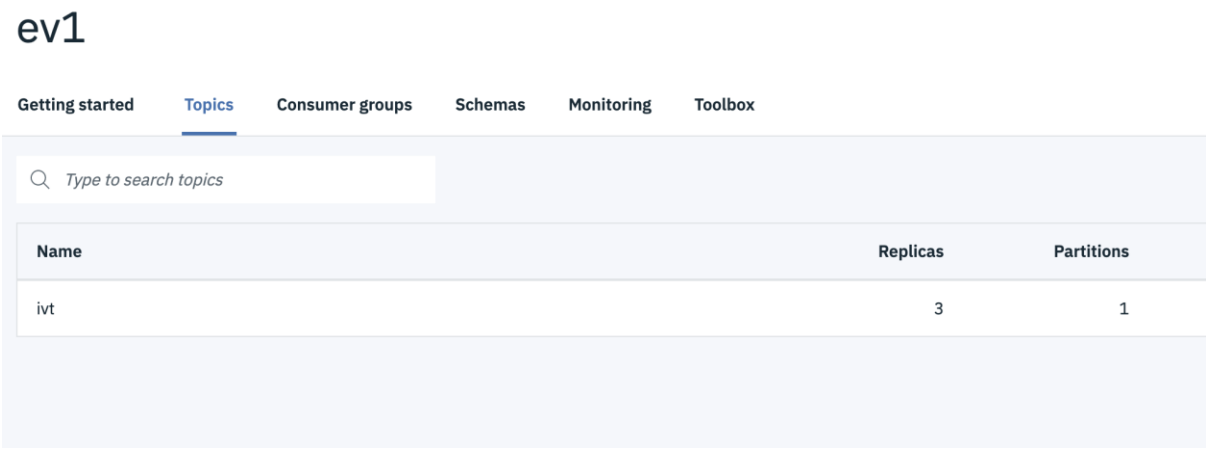
Peters-MacBook-Pro-4:ICP4i peterajessup$ oc get pods

NAME	READY	STATUS	RESTARTS	AGE
ev1-ibm-es-access-controller-deploy-5d5fd45477-6gmr5	2/2	Running	0	4m
ev1-ibm-es-access-controller-deploy-5d5fd45477-6nk4l	2/2	Running	0	4m
ev1-ibm-es-collector-deploy-8676b89c84-db4ww	2/2	Running	0	4m
ev1-ibm-es-elastic-sts-0	2/2	Running	0	4m
ev1-ibm-es-elastic-sts-1	2/2	Running	0	4m
ev1-ibm-es-indexmgr-deploy-8694764d4d-h8mj5	2/2	Running	0	4m
ev1-ibm-es-kafka-sts-0	5/5	Running	2	4m
ev1-ibm-es-kafka-sts-1	5/5	Running	1	4m
ev1-ibm-es-kafka-sts-2	5/5	Running	0	4m
ev1-ibm-es-proxy-deploy-79db8f6bf5-77bgs	1/1	Running	0	3m
ev1-ibm-es-proxy-deploy-79db8f6bf5-sfk87	1/1	Running	0	3m
ev1-ibm-es-proxy-deploy-79db8f6bf5-w8d6q	1/1	Running	0	4m
ev1-ibm-es-rest-deploy-7569668f8b-xjx4v	3/3	Running	0	4m
ev1-ibm-es-rest-producer-deploy-7dbd55b4f5-75n54	1/1	Running	0	4m
ev1-ibm-es-rest-proxy-deploy-54f9cc7f89-n8xv2	1/1	Running	0	4m
ev1-ibm-es-schemaregistry-sts-0	2/2	Running	0	2m
ev1-ibm-es-ui-deploy-c4b89946d-wrbt4	2/2	Running	0	4m
ev1-ibm-es-ui-oauth2-client-reg-52wpk	0/1	Completed	0	4m

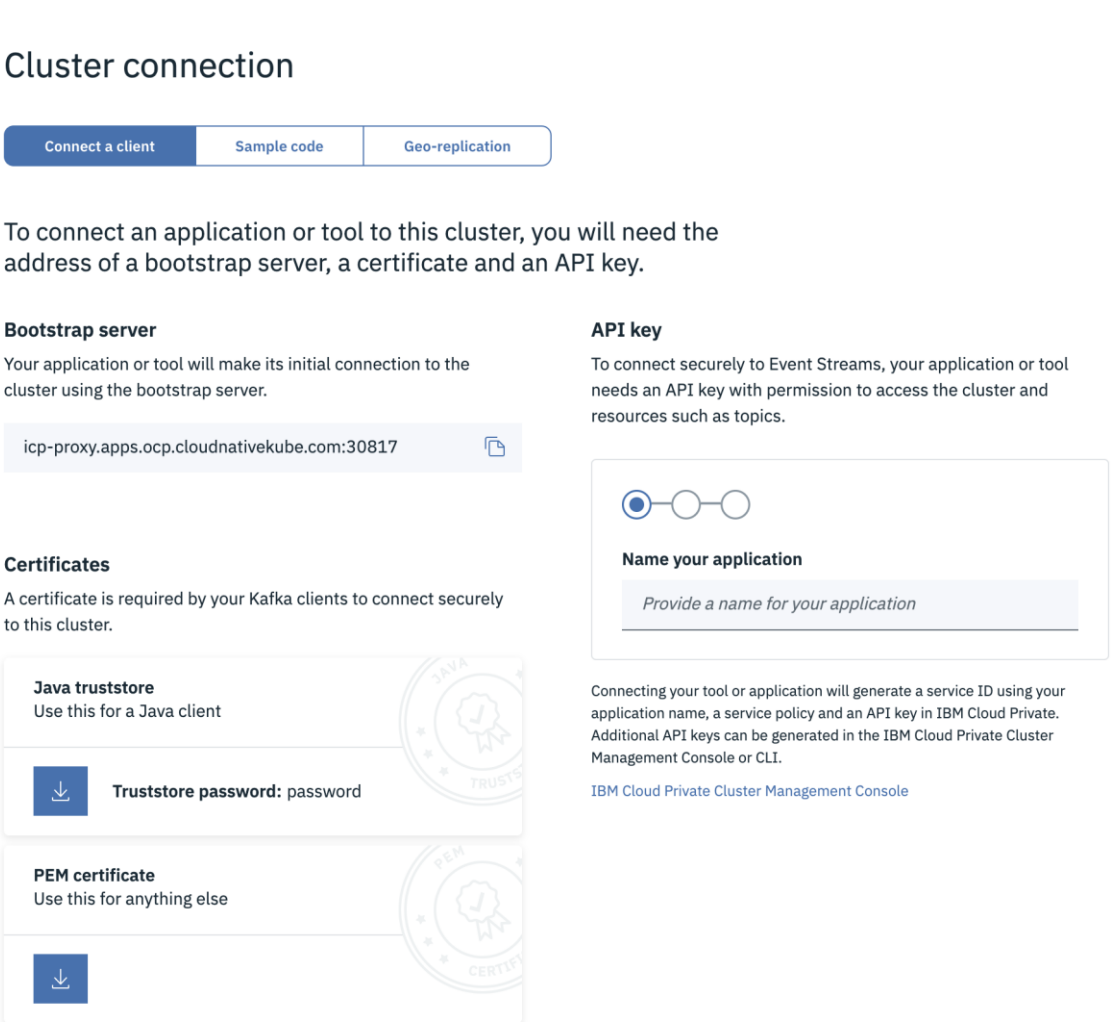
ev1-ibm-es-zookeeper-sts-0	2/2	Running	0	4m
ev1-ibm-es-zookeeper-sts-1	2/2	Running	0	4m
ev1-ibm-es-zookeeper-sts-2	2/2	Running	0	4m

Open the Event Streams Instance from the instance name link in the Event Streams Tile on the Integration Home page:

Configure a new topic called ivt using the ‘New Topic Button ;follow the wizard on the interface using all defaults.



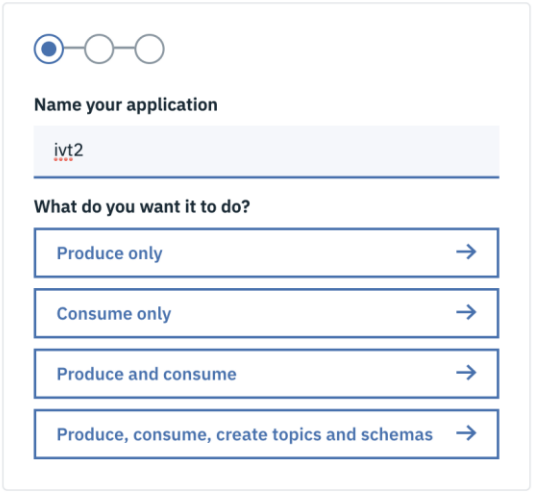
Click on ‘Connect to this cluster’ from the EventStreams UI Home Page and click on the Pem Certificate to download to your local machine. Save this away.



Take note of the bootstrap server host and port and use the API Key wizard to create a new application name and api key. Give the application a name and select ‘Produce and Consume’. Provide a topic name and leave the ‘Consumer Groups’ button set to ‘All’. Click ‘ Generate’ and copy the API Key and save away.

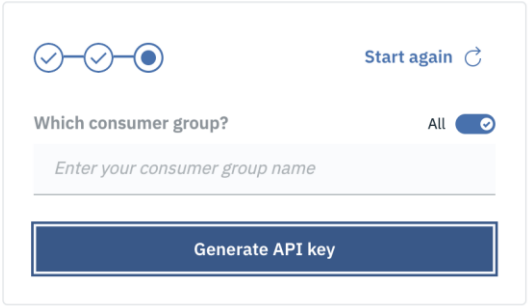
API key

To connect securely to Event Streams, your application or tool needs an API key with permission to access the cluster and resources such as topics.



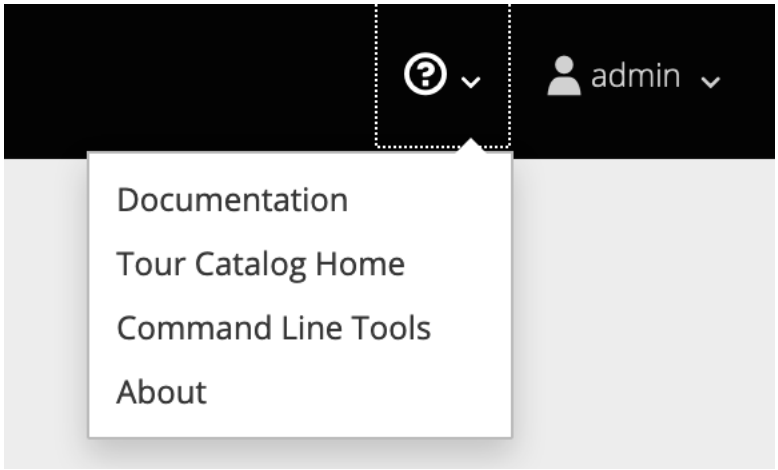
API key

To connect securely to Event Streams, your application or tool needs an API key with permission to access the cluster and resources such as topics.

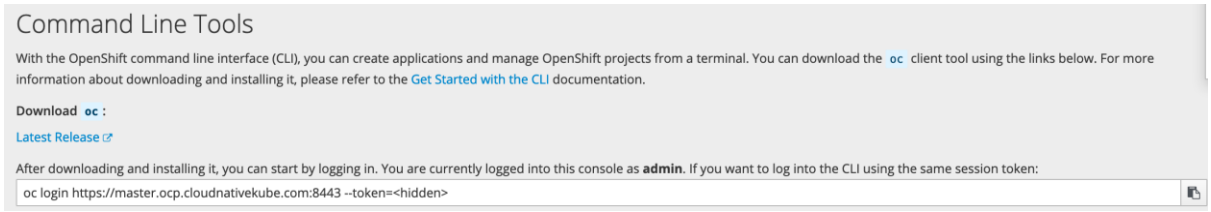


Perform the End to End IVT test

Log into openshift via the command line.
Go to the registry console URL
https://registry-console-default.apps.ocp.cloudnativekube.com/registry



Select Command line tools and follow the instruction here:



Login via cloudctl:

sudo cloudctl login -a https://icp-console.apps.ocp.cloudnativekube.com -u admin -p ##### -n default

Once logged in, copy the pem certificate saved from the Event Streams configuration into an empty directory. Stay in this directory for all further commands.

Copy the pem file to a file named “truststoreCert-mykey.crt”

Create a new file called serverconf.yaml and place the following snippet into it and save it.

-------Begin snippet----------

ResourceManagers:

JVM:

truststoreType: 'JKS'

truststoreFile: '/home/aceuser/ace-server/truststore.jks'

truststorePass: 'setdbparms::truststore'

Defaults:

policyProject: 'DefaultPolicies'

Policies:

HTTPSConnector: 'HTTPS'

----------End Snippet----------

Create a new file called setdbparms.txt and add the following snippet, replacing <api_key> with the api key you saved from the Event Streams instance.

-------Begin snippet----------

kafka::KAFKA token <api_key>
setdbparms::truststore dummy password

----------End Snippet----------

Create a new file called truststorePassword.txt and place the text 'password' on the first line of this file.

-------Begin snippet----------

password

----------End Snippet----------

Run the generateSecrets.sh defined earlier in this document by creating this file in the same directory as the files created earlier in this section:

Chmod a+x generateSecrets.sh

./generateSecrets.sh ivt-secret

This creates a secret called ivt-secret for use in the HELM chart.

Locate the ivt5.bar and ivt.zip (Project Interchange file) located in the IVT Artifacts directory under here: https://ibm.ent.box.com/folder/90729333577

We will use this bar file in the ACE deployment.

queue manager called 'QM1, HELM release name 'mqtest', with the following objects (as defined earlier in this document)

Channel: IVT.SVRCONN

Security: Disabled

Queues: REREQ.Q, RT.REP.Q

Queue Manager Name : QM1

Internal to Cluster Host Name: mqtest-ibm-mq.mq.svc

Port : 1414

This ACE deployment relies on an Event Streams deployment as defined in and earlier section, with a topic called IVT.

You can change these parameters by modifying the ivt.zip in the ACE toolkit and re-deploying the bar file after re-building.

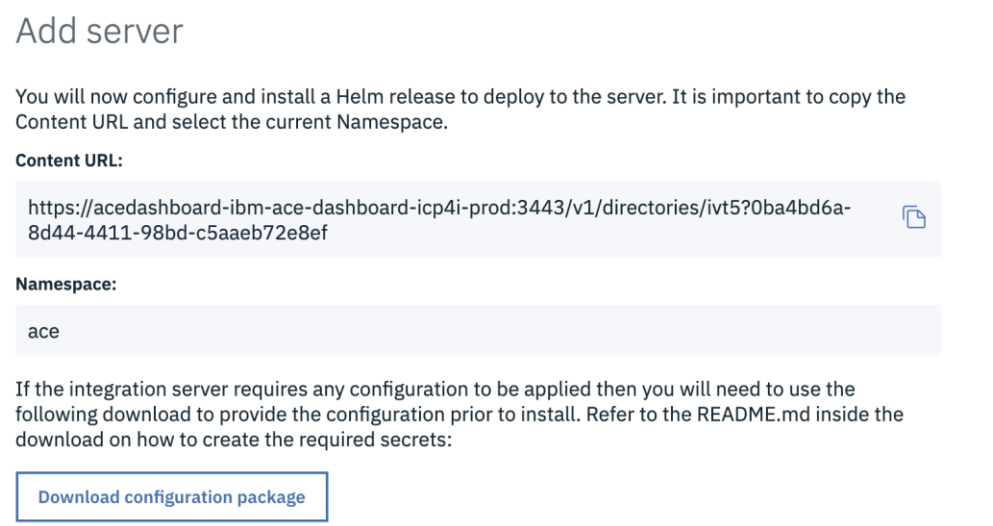
To deploy the ace server we will use the HEL command line and the following script called installACE-MQC.sh

-------Begin snippet----------

```
if [ "$2" == "" ]
then
echo "Please provide a integration server name as the second argument"
echo "Usage: installACE-MQC.sh <release_name> <Integration_server_name> <content Server URL> [<secret_name>]"
exit 1
fi
if [ "$4" == "" ]
then
SECRET_ARG="nil"
else
SECRET_ARG=$4
fi
if [ "$3" == "" ]
then
echo "Please provide the content server URL as the 3rd argument"
echo "Usage: installACE-MQC.sh <release_name> <Integration_server_name> <content Server URL> [<secret_name>]"
exit 1
fi
echo Secret=$SECRET_ARG
sudo helm install local-charts/ibm-ace-server-icp4i-prod --name=$1 --set license=accept --tls --set aceonly.replicaCount=1 --set integrationServer.configurationSecret=$SECRET_ARG --set integrationServer.truststoreCertNames=mykey --set contentServerURL=$3  --set imageType=acemqclient --set image.acemq=docker-registry.default.svc:5000/ace/ibm-ace-mqclient-server-prod:11.0.0.5.1 --set image.pullPolicy=IfNotPresent --set image.configurator=docker-registry.default.svc:5000/ace/ibm-ace-icp-configurator-prod:11.0.0.5.1 --set integrationServer.name=$2 --set integrationServer.configurationSecret=$SECRET_ARG --set service.iP=icp-proxy.apps.ocp.cloudnativekube.com --set persistence.enabled=true --set persistence.useDynamicProvisioning=true --set dataPVC.name=data --set dataPVC.storageClassName=managed-nfs-storage --set productionDeployment=false --set metrics.enabled=false
```

----------End Snippet----------

Note the arguments to this script, and ensure you have the correct content server URL as defined when deploying the bar in the GUI for a new ace instance – here's an example:



Create the script file on your local machine, and execute the script as follows:

----------Begin Snippet----------

Chmod a+x installACE-MQC.sh

./installACE-MQC.sh aceivt IS1 https://acedashboard-ibm-ace-dashboard-icp4i-prod:3443/v1/directories/ivt5?0ba4bd6a-8d44-4411-98bd-c5aaeb72e8ef ivt-secret

----------End Snippet----------

This will deploy an MQ Client only ACE instance with the supplied bar file and secret.

After deployment check the pods:

----------Begin Snippet----------

oc get pods

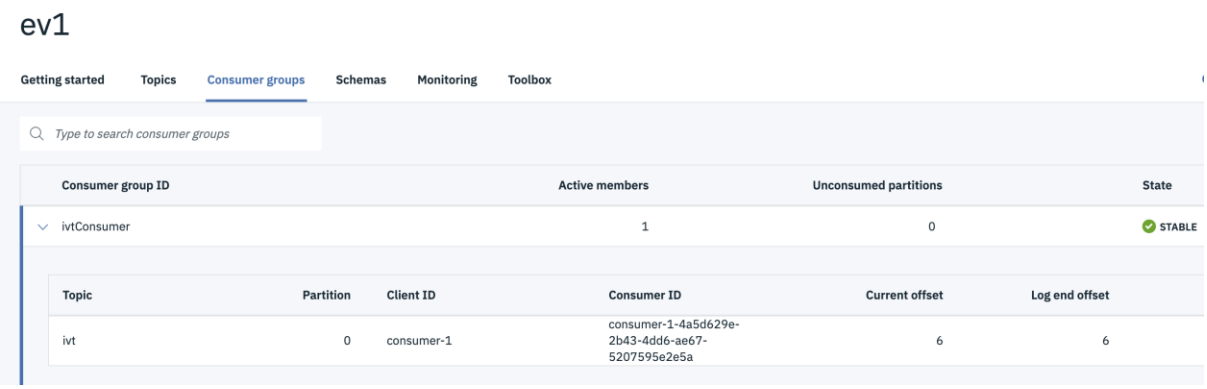
NAME READY STATUS RESTARTS AGE

acedashboard-ibm-ace-dashboard-icp4i-prod-546b78c6c6-xp5qr 2/2 Running 0 4d

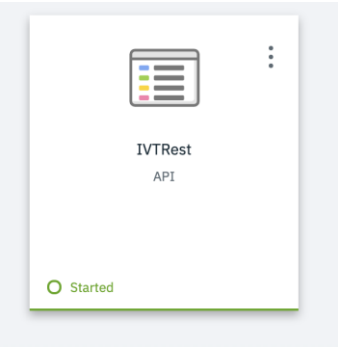
aceivt-ibm-ace-server-icp4i-prod-5f87ccfbf7-q29l9 1/1 Running 0 9m

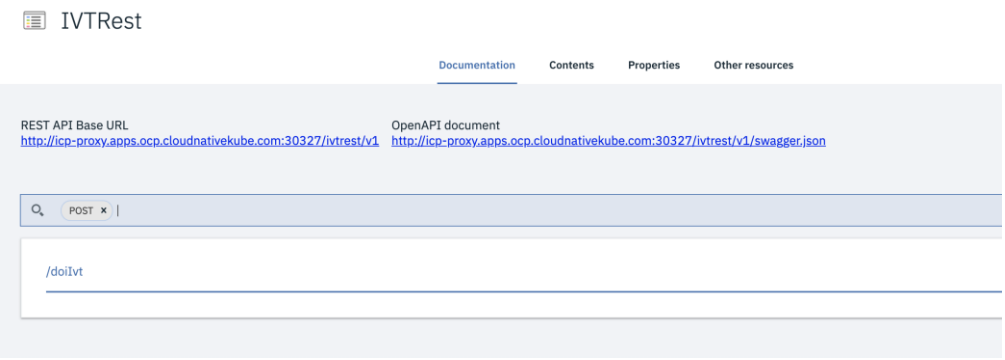
----------End Snippet----------

Check the Event Streams instance for a connected consumer group called ‘ivtConsumer’.



Test the ACE instance by clicking on the instance tile on the ACE dashboard and drilling down into the REST API which is deployed:





The REST API base URL *port number* will be different for each deployment.

Open up a REST testing tool and enter the following URL (from the example URL above) into the tool's URL:

http://icp-proxy.apps.ocp.cloudnativekube.com:30327/ivtrest/v1/doIvt

Use the following as the body for the POSt operation:

```
{
  "name":"anyname",
```

"phone":"123545454",

"email": "anyname@host.com"

}

Test the POST.

A json payload should be returned in ALL UpperCase characters.

{"name":"ANYNAME","phone":"123545454","email":"ANYNAME@HOST.COM"}

Check the console of the queue manager (or other tool) for the response message in the RTREQ.Q and the RTREP.Q

Check that the Consumer Group Offset in the Event Streams UI for the group ivtConsumer has increased by one.

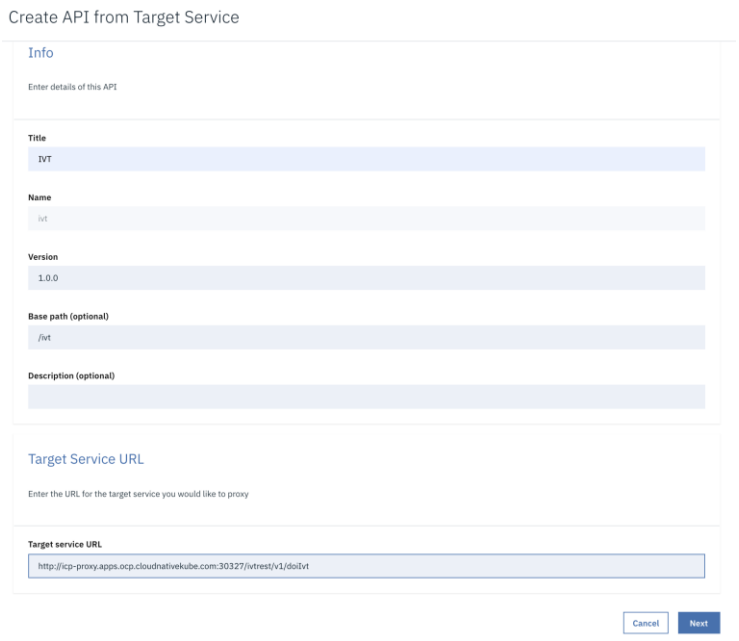
Now we'll create an API in API Connect to proxy this API.

Log into the designer UI of the API Connect Instance created by following the instructions in the document 'Deploying API Connect on ICP4i' located here:
https://ibm.ent.box.com/folder/90729333577

Click on Develop AIS and Products and click 'Add' and select 'API'

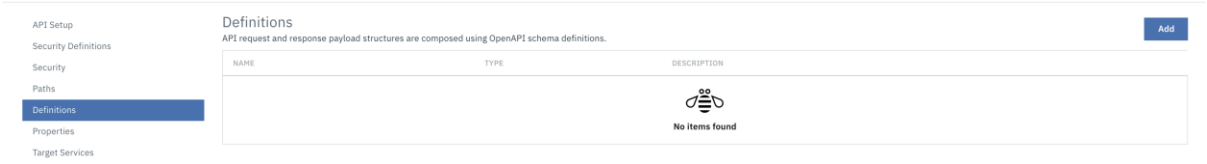
Click on 'From Target Service' and click 'Next'

Fill in the table as shown (note the target service URL port number for you will be different)

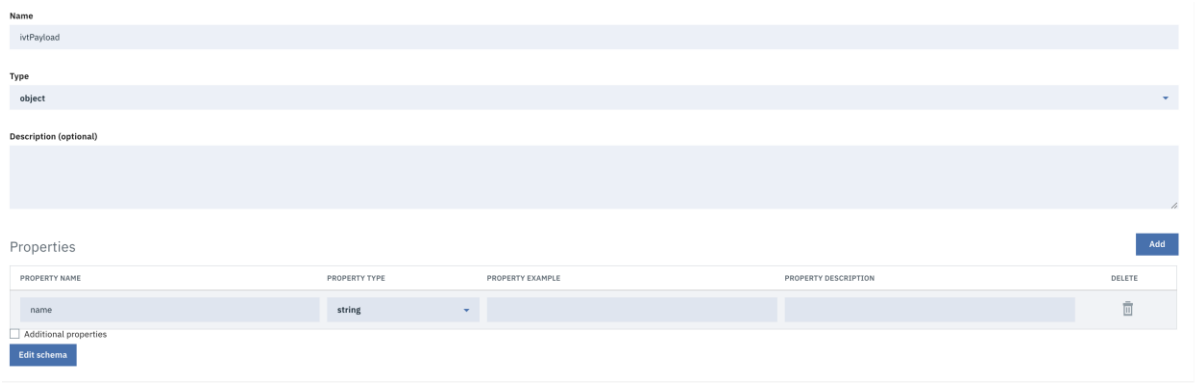


Click Next and accept the defaults, click 'Next' and then 'Edit API'

Click on ‘Definitions and click ‘Add’



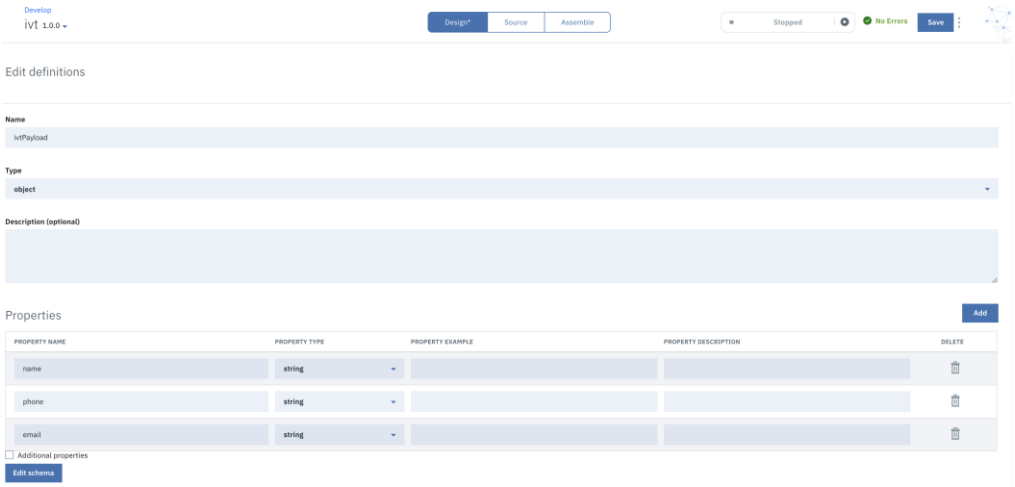
In the name field type ‘ivtPyload’ then click ‘Add’ adjacent to the properties section.



Enter the field name and select type of ‘String’.

Add two more properties called ‘phone’ and ‘email’ of type ‘String.

Click ‘Save’ at the top of the form.

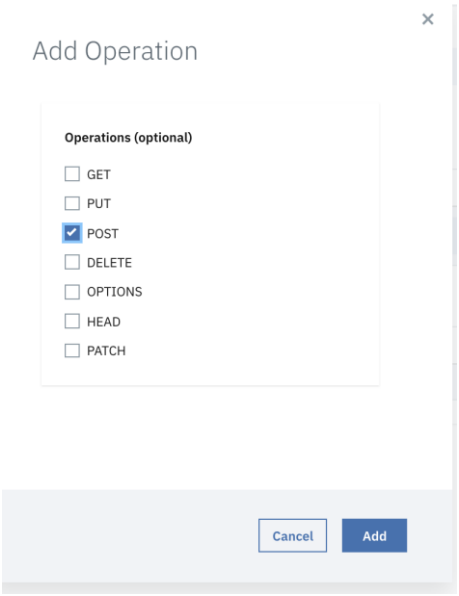


Select ‘Paths’ from the top level menu and delete the path labelled ‘/’ using the dot menu on the right hand side.

Click ‘Add’

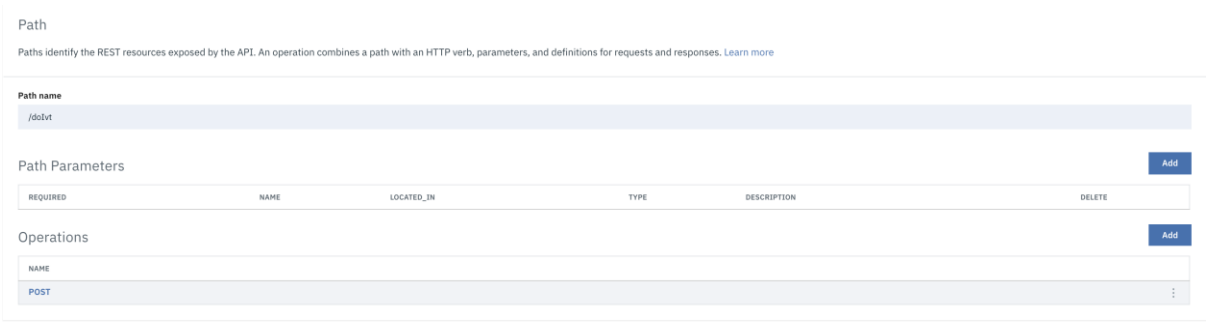
In the Path Name field type ‘/doIvt’. The forward slash is required.

Adjacent to the operations section click ‘Add’

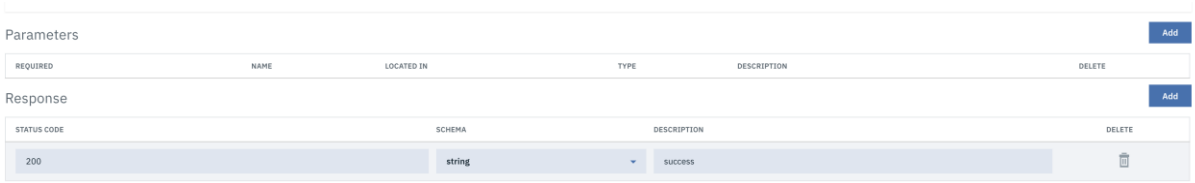


Click ‘Save’

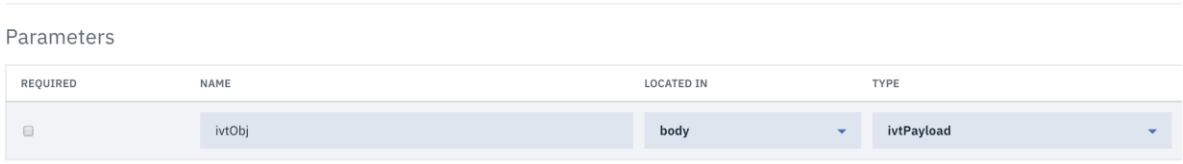
Click on the newly Created path, then on the Operation labelled ‘POST’



Click ‘Add’ Adjacent to the Parameters Section.

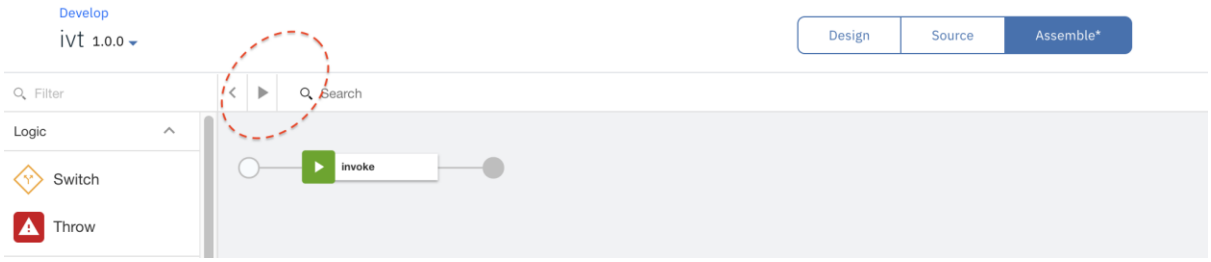


Fill out the form as follows:



Click ‘Save’ at the top of the page.

Click on ‘Assemble’ at the page top center to bring up the Assembly.



Click on the grey arrow highlighted above.

Click on ‘Activate API’ and wait for it to be published.

You’ll see a result similar to:

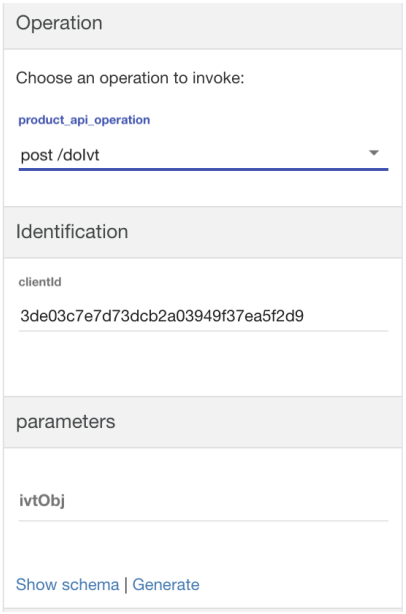
Catalog: sandbox

Product: ivt-auto-product

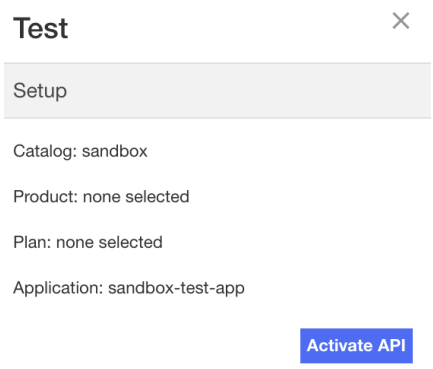
Plan: Default Plan

Application: sandbox-test-app

Click on the drop down showing ‘product_api_operation’. Select ‘post/doIvt’.

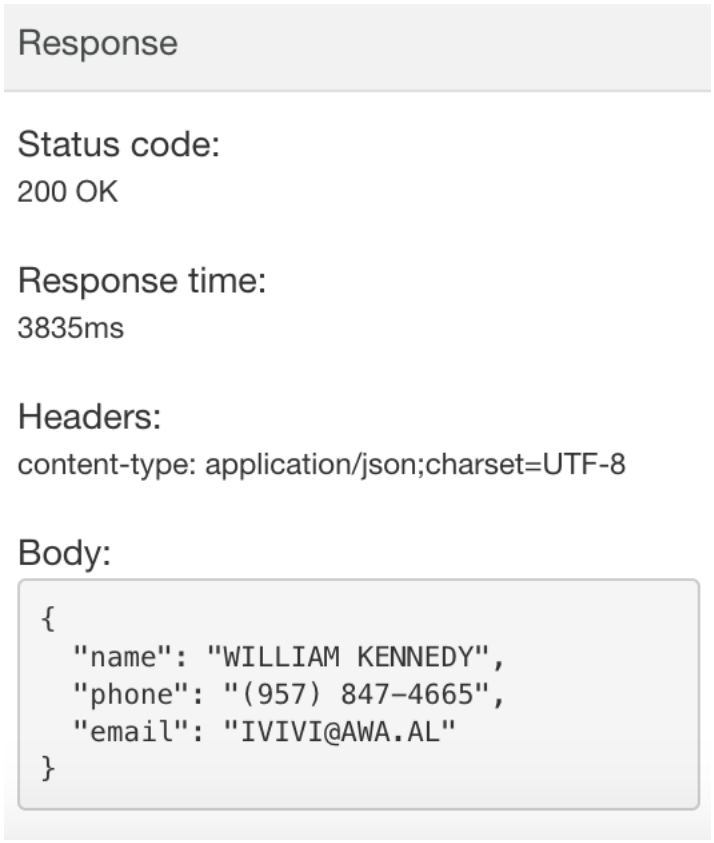


Click on ‘Generate’ to generate a test message.



Click on ‘Invoke’ at the base of the form.

You should see something similar to:



As before check the MQ queues and consumer group offset in Event Streams.

This concludes the end-to-end IVT test.

Appendix A – setting up the PSP

```
C:\SHARE\ProductsTechnology\IBM Cloud Pak for Integration\SetUp\yamls>dir
 Volume in drive C is Windows
 Volume Serial Number is 3CBA-3B12

 Directory of C:\SHARE\ProductsTechnology\IBM Cloud Pak for Integration\SetUp\yamls

18/10/2019  12:44 PM    <DIR>          .
18/10/2019  12:44 PM    <DIR>          ..
18/10/2019  12:44 PM               836 ibm-anyuid-psp.yaml
```

```
C:\SHARE\ProductsTechnology\IBM Cloud Pak for Integration\SetUp\yamls>\openshift\oc config current-context
ace/master-ocp-cloudnativekube-com:8443/admin
```

```
C:\SHARE\ProductsTechnology\IBM Cloud Pak for Integration\SetUp\yamls>\openshift\oc create -f ibm-anyuid-psp.yaml
podsecuritypolicy.policy/ibm-anyuid-psp created
```

```
C:\SHARE\ProductsTechnology\IBM Cloud Pak for Integration\SetUp\yamls>\openshift\oc get psp
NAME              PRIV      CAPS
SGROUP     SUPGROUP    READONLYROOTFS   VOLUMES
ibm-anyuid-psp     false     SETPCAP,AUDIT_WRITE,CHOWN,NET_RAW,DAC_OVERRIDE,FOWNER,FSETID,KILL,SETUID,SETGID,NET_BIND_SERVICE,SYS_CHROOT,SETFCAP
unAsAny    RunAsAny    false           configMap,emptyDir,projected,secret,downwardAPI,persistentVolumeClaim
tekton-pipelines   false
ustRunAs   MustRunAs   false           emptyDir,configMap,secret
```

Appendix B – RoundTrip Batch file

You may need to modify the path to your own java installation.

"c:\Program Files\ibm\MQ\java\jre\bin\java" -classpath .\com.ibm.mq.allclient.jar;.\jms.jar;.\jndi.jar;.\imaclientjms.jar;.\jms.jar;.\GUIRoundTrip.jar;org.dyno.visual.swing.jar;org.dyno.visual.swing.layouts.jar com.ibm.mq.testclient.RoundTripF

Appendix C – MQ and ACE DNS naming

For the ace/mq services it follows the pattern described here:

https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/

The dns name is <service_name>.<namespace>."svc"

So for the ace release "acefin-es" we lookup the service name as

"acefin-es-ibm-ace-server-icp4i-prod"

Using the same convention the dns name is

"acefin-es-ibm-ace-server-icp4i-prod.ace.svc"

For An MQ of Helm release name mqivt3 - has a QMGR=QM3

The MQ connection information for clients inside the cluster is as follows:
mqivt3-ibm-mq.mq.svc:1414

In order to connect to that queue manager. Created a SENDER channel on another queue manager in the cluster. The conname will be

CONNAME = mqivt3-ibm-mq.mq.svc(1414)

Therefore the naming convention is *helmRelease*-**ibm-mq.mq.svc**

DNS name for MQ only in mq namespace Example :Helm Release = repqm1p

DNS = repqm1p-ibm-mq.mq.svc

DNS name for ACE/MQ in ace namespace Example: Helm Release = eaiqm1

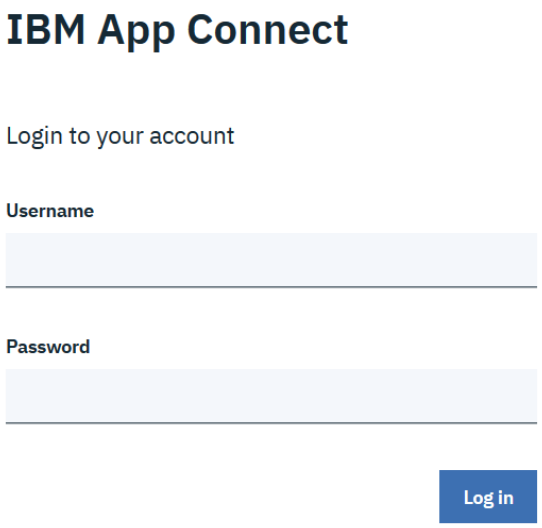
DNS = eaiqm1p-ibm-ace-server-icp4i-prod.ace.svc

Appendix D – ACE GUI – avoid login

Clicking on the webUI from Helm release

http://icp-console.apps.ocp.cloudnativekube.com:31867/login?redirectedFrom=%2F

results in a



To avoid this change the URL to use the **proxy** entry point not the console

http://icp-**proxy**.apps.ocp.cloudnativekube.com:31867/

Appendix E – Useful references

https://developer.ibm.com/messaging/2019/08/05/configure-and-install-ibm-mq-queue-managers-on-ibm-cloud-pak-for-integration/

Appendix F – connecting in an "Off RHOS" Queue manager

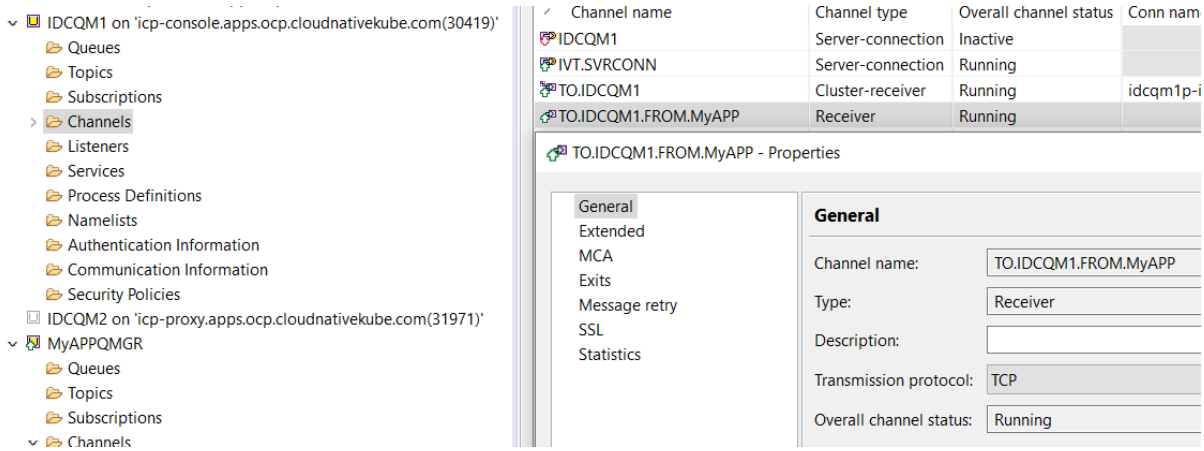
IDCQM1 – Definitions

```
*******
*** Place holder definitions for an OFF RHOS cluster queue manager connecting in via DQM
*** A queue manager called OFFRHOSQM with the opposite definitions will be required on-premises
********
* Receiver channel for OFF RHOS cluster queue manager connecting in via DQM
DEFINE CHANNEL('TO.QMGR_NAME.OFFRHOSQM') +
   CHLTYPE(RECEIVER) +
   TRPTYPE(TCP) +
   REPLACE

* Local transmission queue to OFFRHOSQM
DEFINE QLOCAL('OFFRHOSQM') +
  DEFPSIST(YES) +
  DEFBIND(NOTFIXED) +
  DISTL(NO) +
  MAXDEPTH(5000) +
  USAGE(XMITQ)+
  REPLACE

* Sender channel to OFF RHOS cluster queue manager connecting via DQM
DEFINE CHANNEL('TO.OFFRHOSQM.QMGR_NAME') +
   CHLTYPE(SENDER) +
   TRPTYPE(TCP) +
   CONNAME('ToBeDefined(9999)')+
   XMITQ('OFFRHOSQM')+
   REPLACE

DEFINE QREMOTE('TEST.IIB.Q1.REP') +
  RQMNAME('OFFRHOSQM) +
  RNAME('TEST.IIB.Q1.REP') +
  CLUSTER('EAICLUSTER.ENV_ID') +
  REPLACE
```

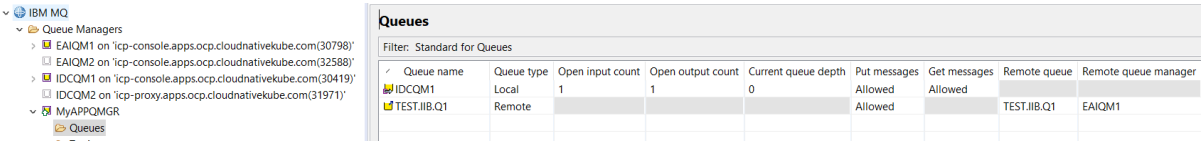
Note the following screen captures need to be updated to reflect the naming of objects above that are configured into the IDCQM1 queue manager on RHOS

Sending messages from Off RHOS Qmgr to EAIQM1 via IDCQM1(Gateway)

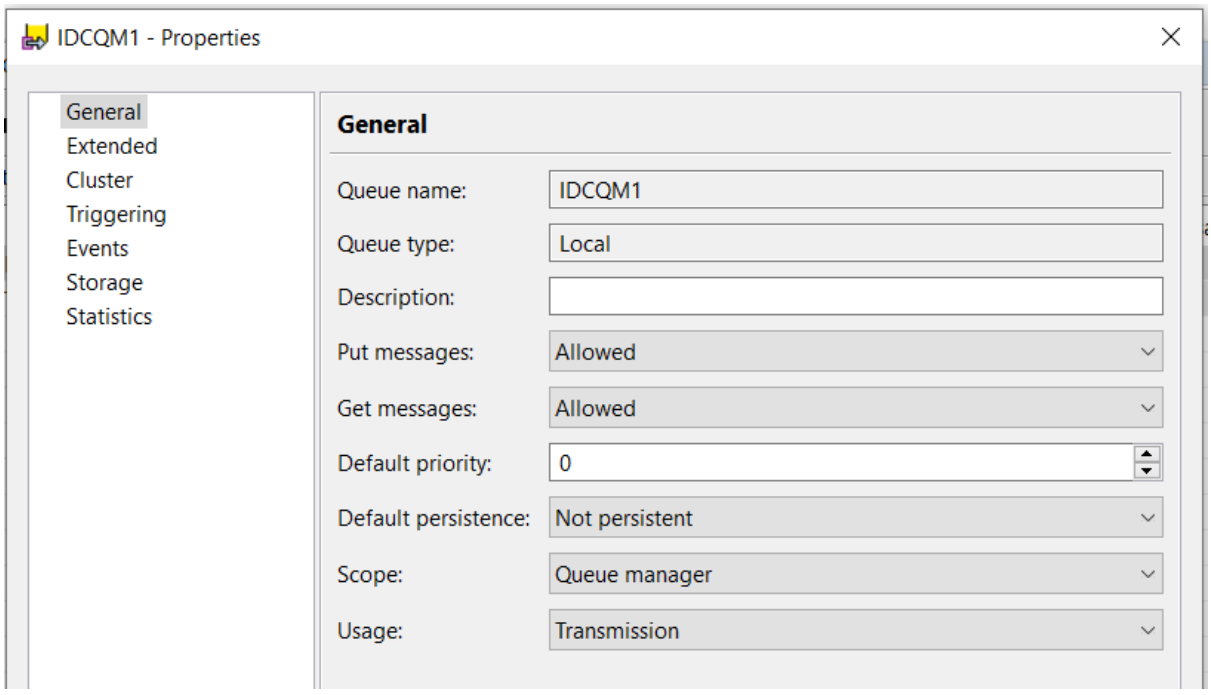
Channel receiver on IDCQM1



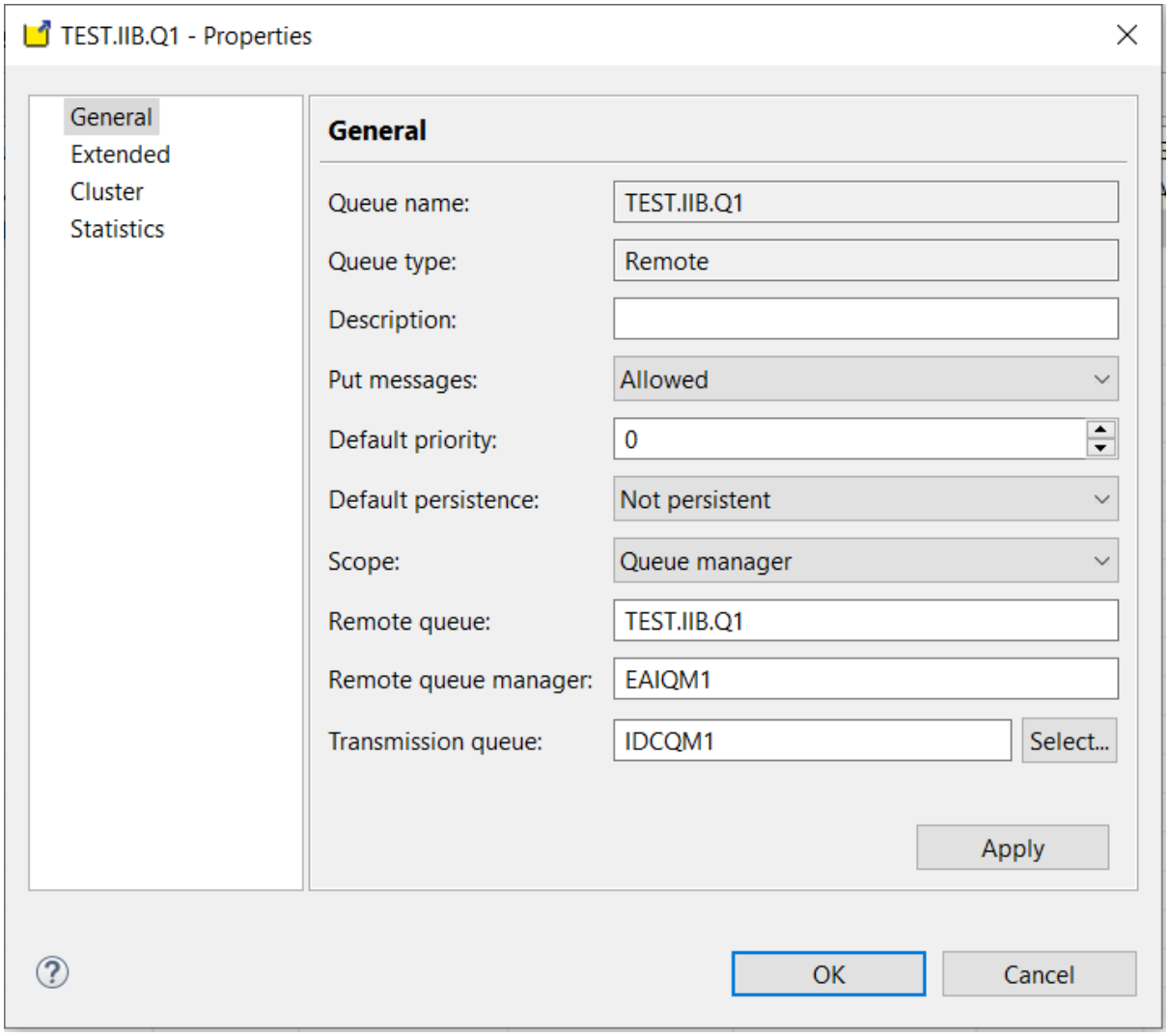
Create a local windows off rhos queue manager called MyAPPLQMGR with default settings



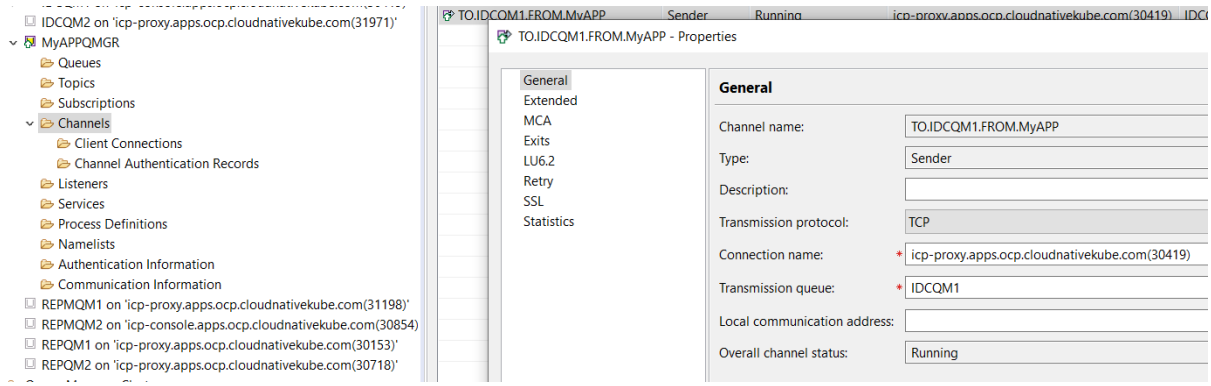
Create a transmission queue for IDCQM1 on OFFRHOS queue manager



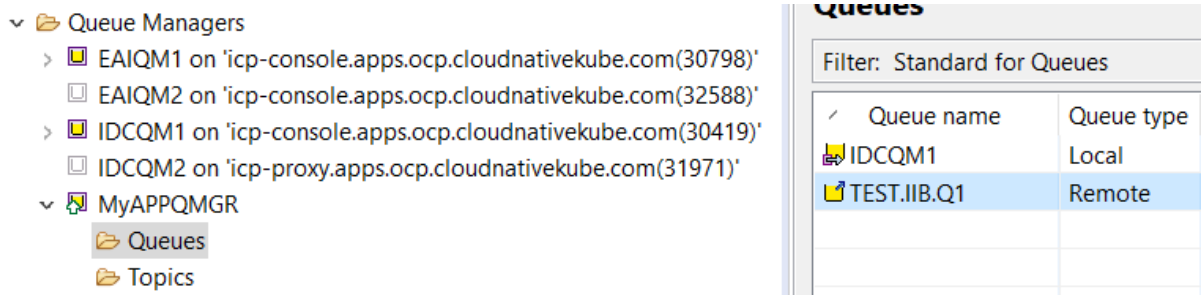
Create a remote queue definition on MyAPPLQMGR to send messages to EAIQM1 via IDCQM1

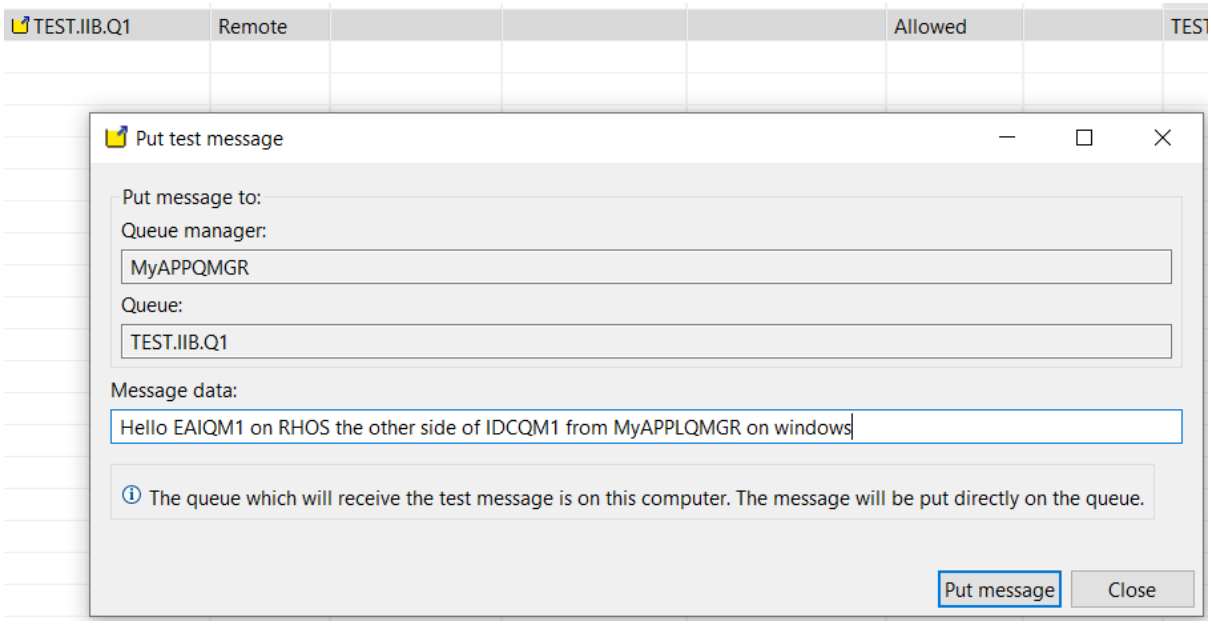


On off RHOS QM create a Sender channel to IDCQM1(matches the name of the receiver on IDCQM1)

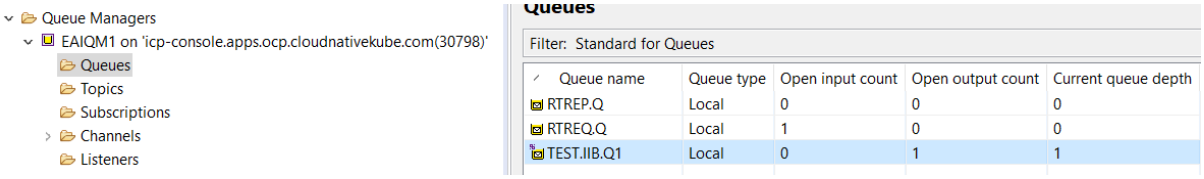


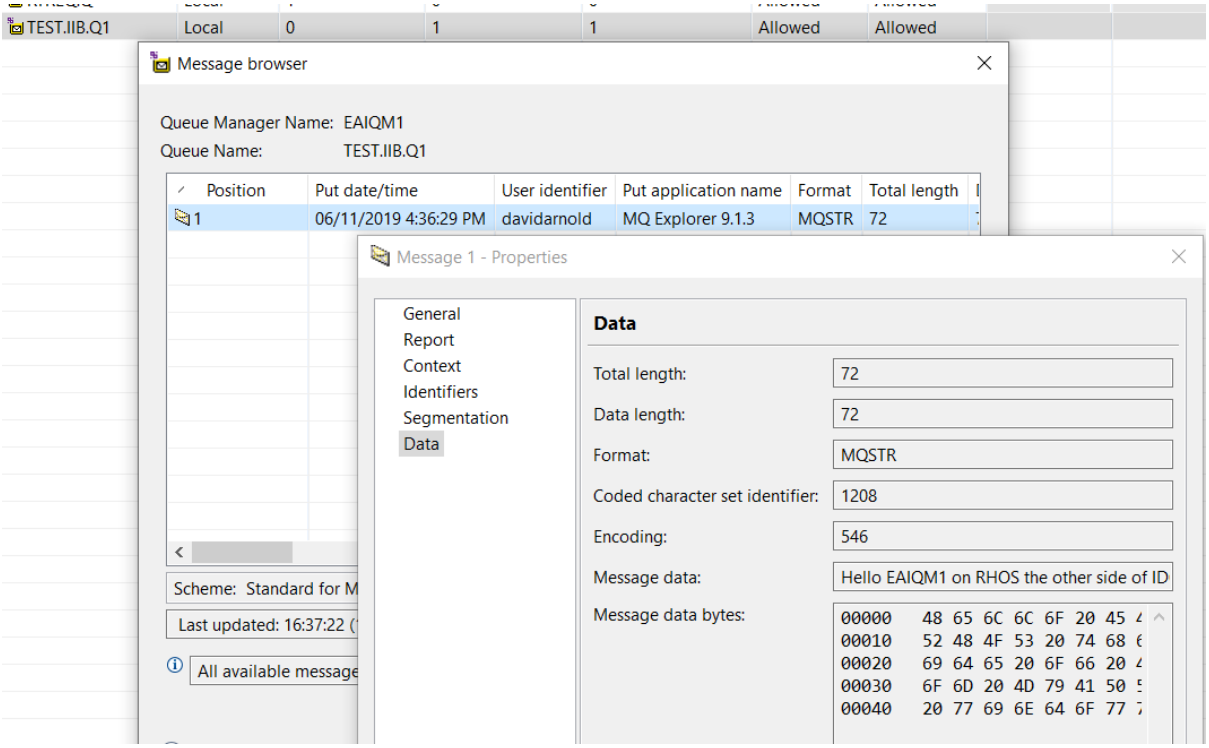
Put a test message from MyAPPLQMGR





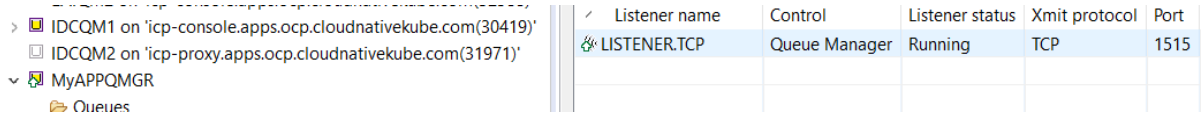
Check TEST.IIB.Q1 on EAIQM1



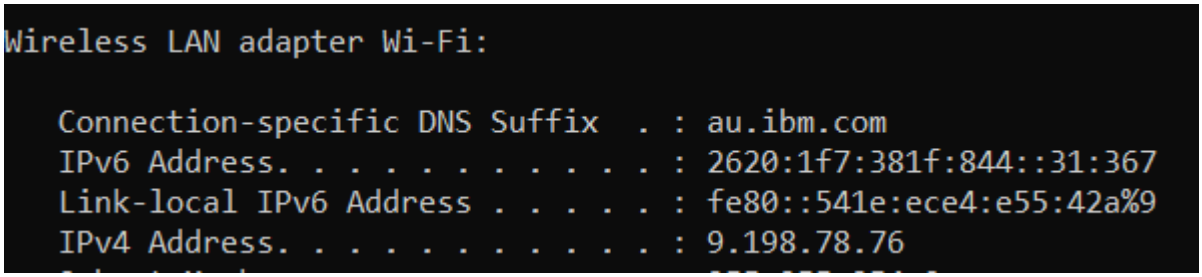

TEST.IIB.Q1
Local
0
1
1
Allowed
Allowed
Message browser
Queue Manager Name: EAIQM1
Queue Name: TEST.IIB.Q1
Position
Put date/time
User identifier
Put application name
Format
Total length
1
06/11/2019 4:36:29 PM
davidarnold
MQ Explorer 9.1.3
MQSTR
72
Scheme: Standard for M
Last updated: 16:37:22 (
All available message
Message 1 - Properties
General
Report
Context
Identifiers
Segmentation
Data
Data
Total length:
72
Data length:
72
Format:
MQSTR
Coded character set identifier:
1208
Encoding:
546
Message data:
Hello EAIQM1 on RHOS the other side of ID
Message data bytes:
00000 48 65 6C 6C 6F 20 45
00010 52 48 4F 53 20 74 68
00020 69 64 65 20 6F 66 20
00030 6F 6D 20 4D 79 41 50
00040 20 77 69 6E 64 6F 77

Receiving messages to Off RHOS Qmgr from EAIQM1 via IDCQM1(Gateway)

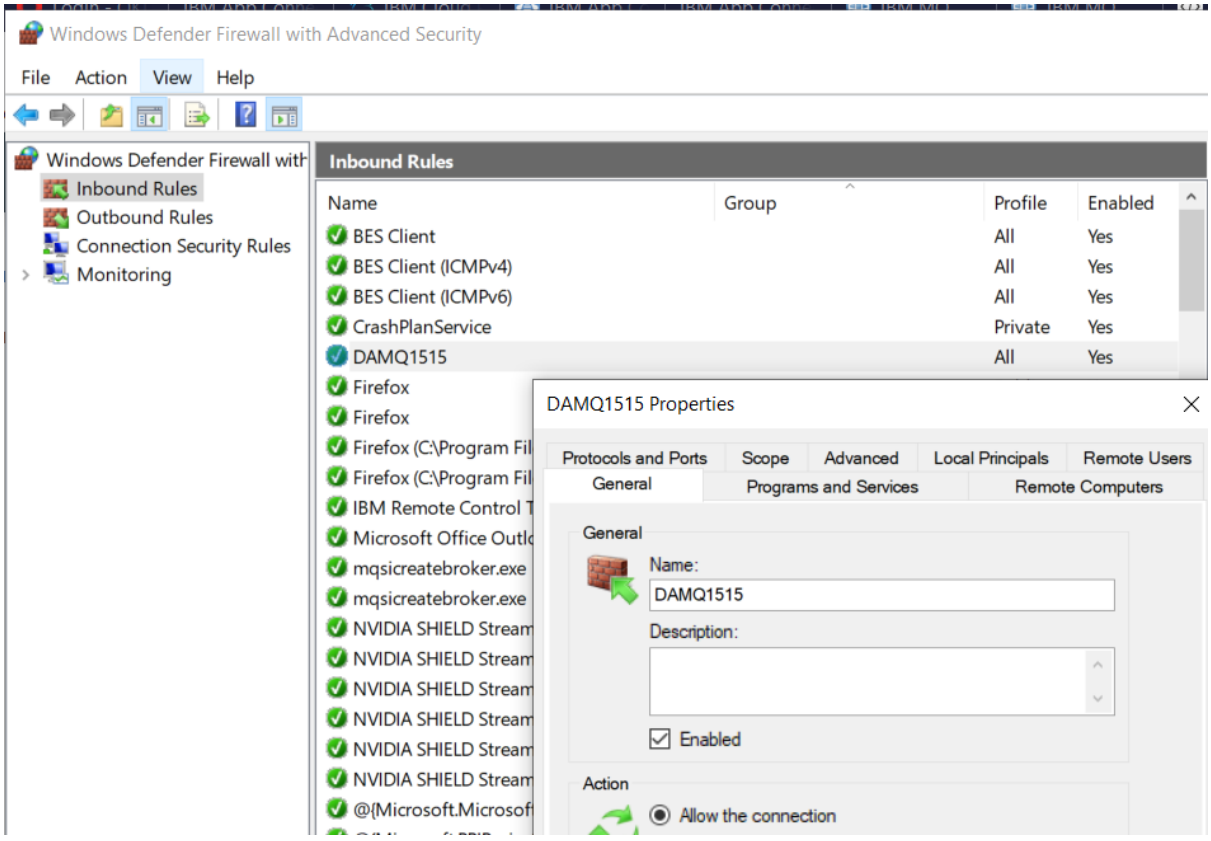
Listener on off RHOS system

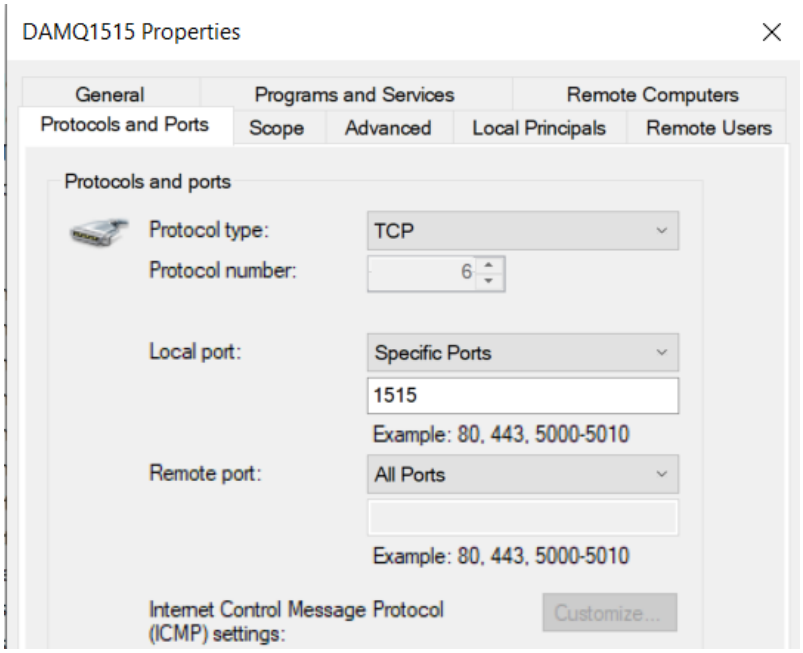


IPCONFIG on target off box system



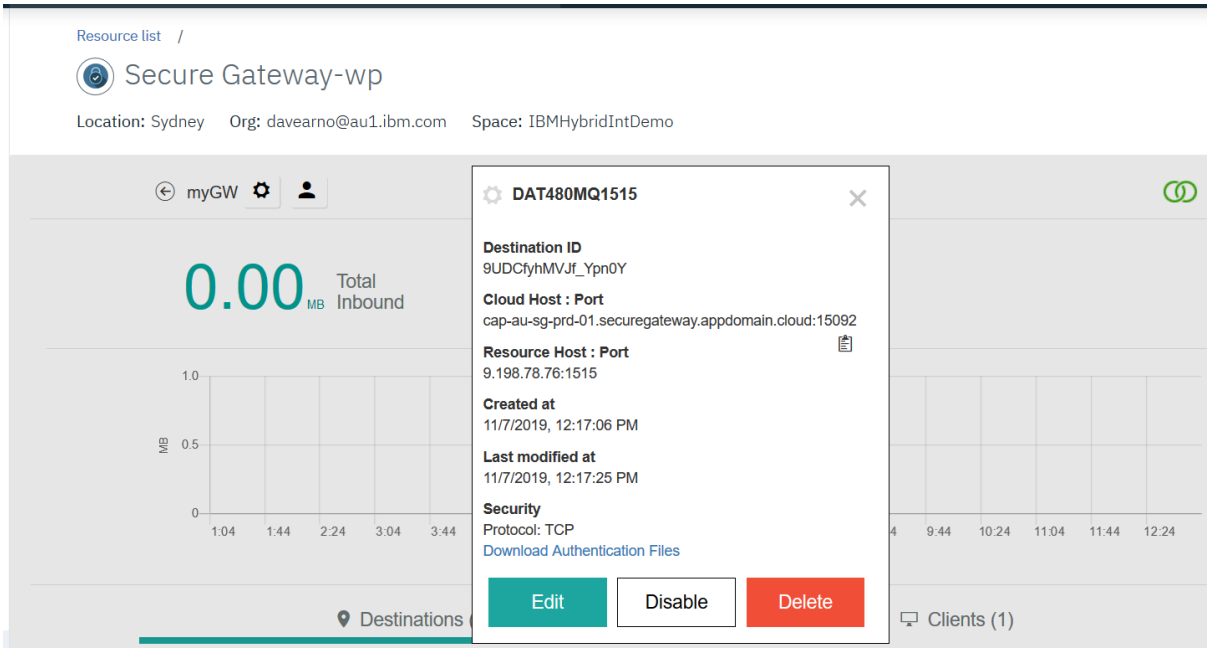
On premise Firewall rules



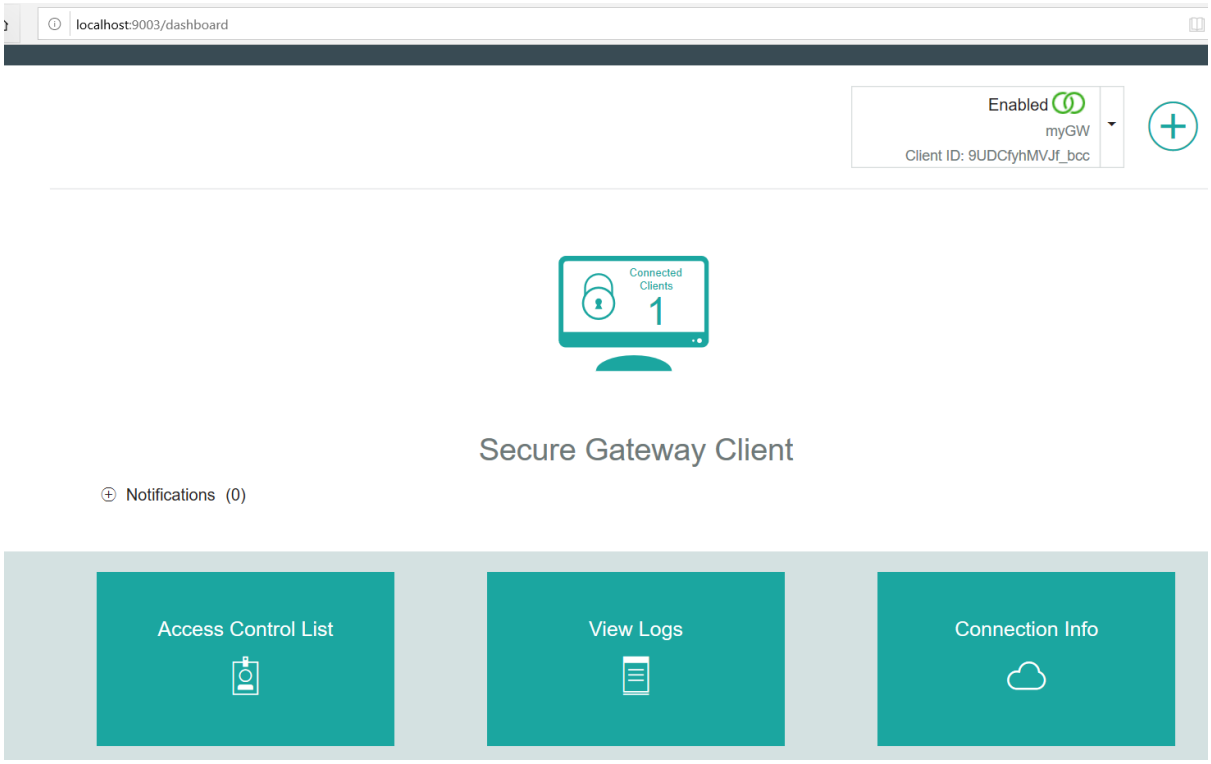


Secure Gateway on IBM cloud

You will need to use a service like the IBM Secure Gateway on IBM Public Cloud to provide a public IP address for your queue manager on premises.



Secure client on premises



localhost:9003/acl

Access Control List Ma

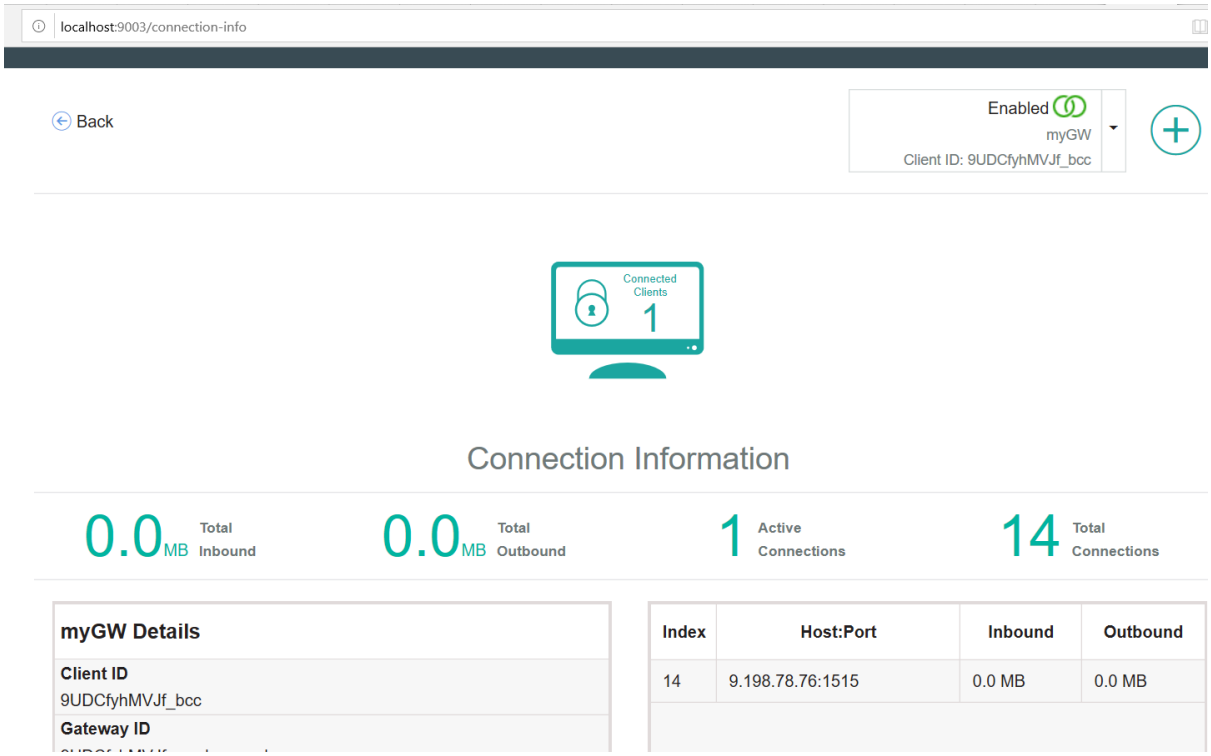
Access Control List entries determine what the client is a

Use the tables below to manually input the individual host:port entries. For bulk
page. For more information on the Access Control List, view the readme file in y

Allow access

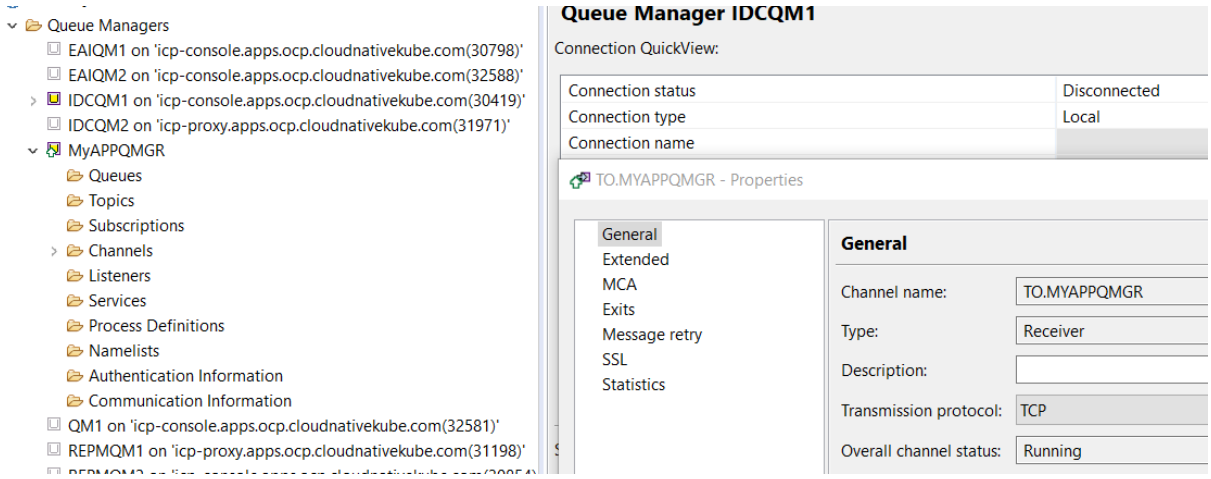
Resource Hostname : Port

cap-au-sg-prd-01.securegateway.appdomain.cloud:15092

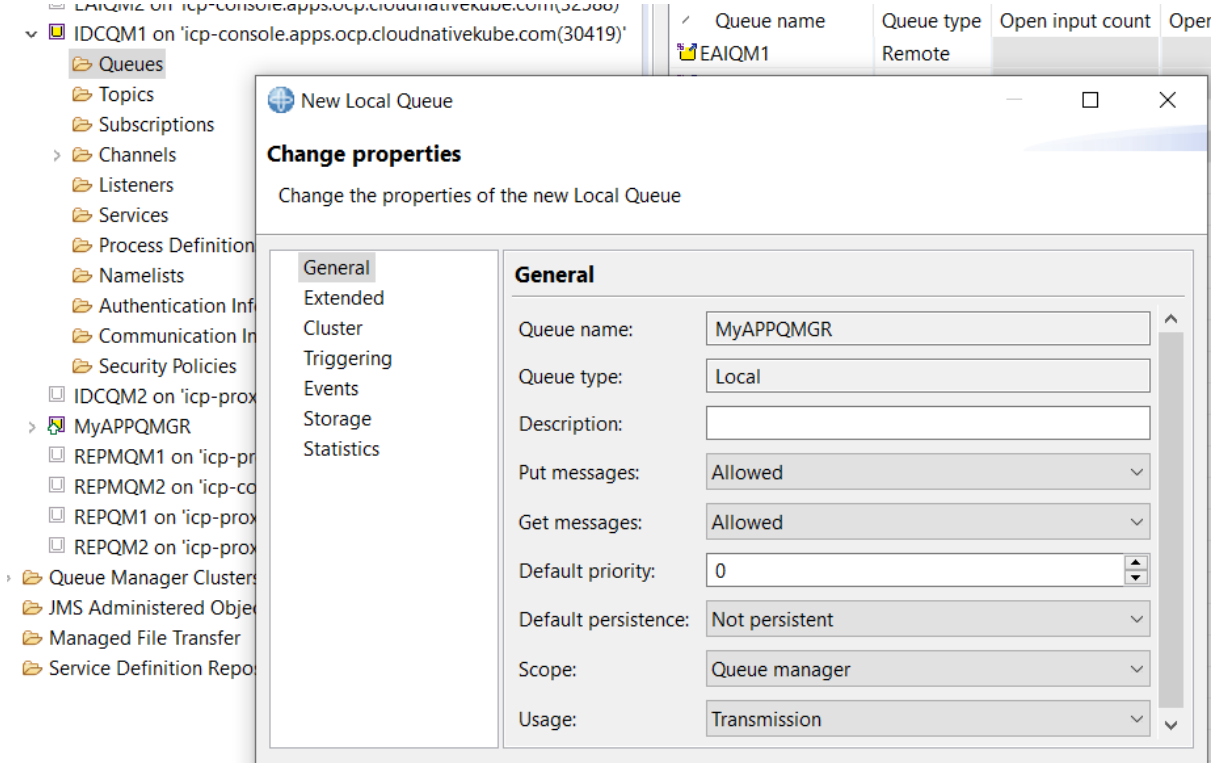


MQ configuration

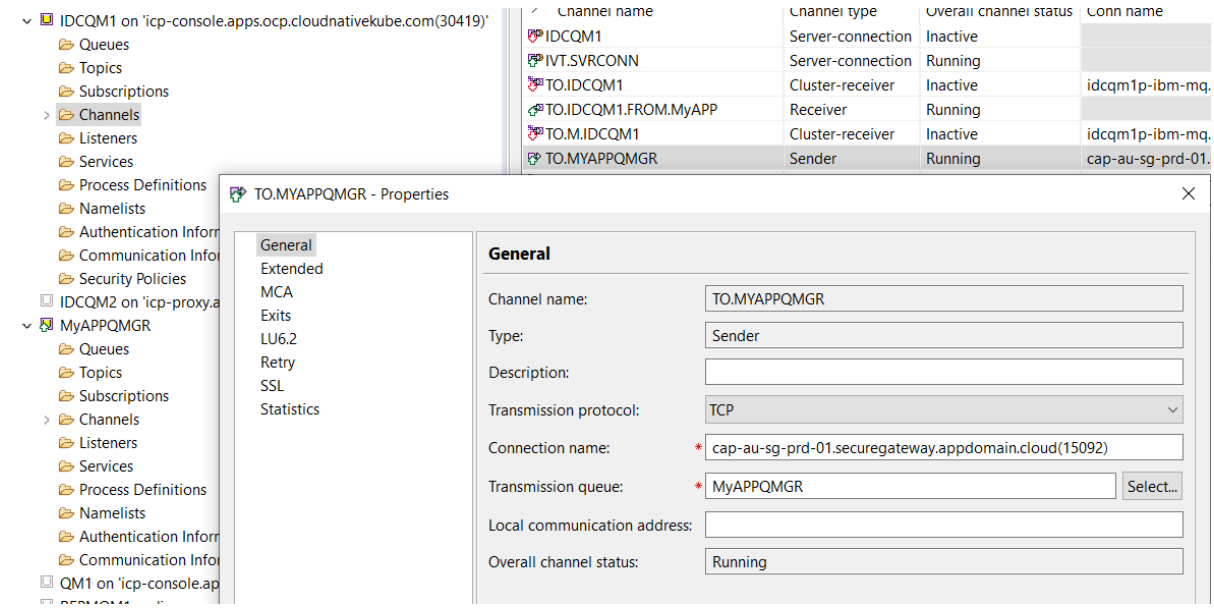
Receiver channel on MyAPPQMGR (the “off RHOS” queue manager)



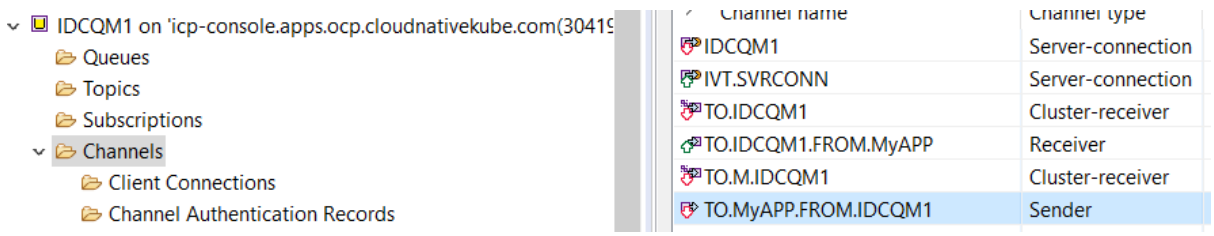
Transmission queue and sender channel on IDCQM1



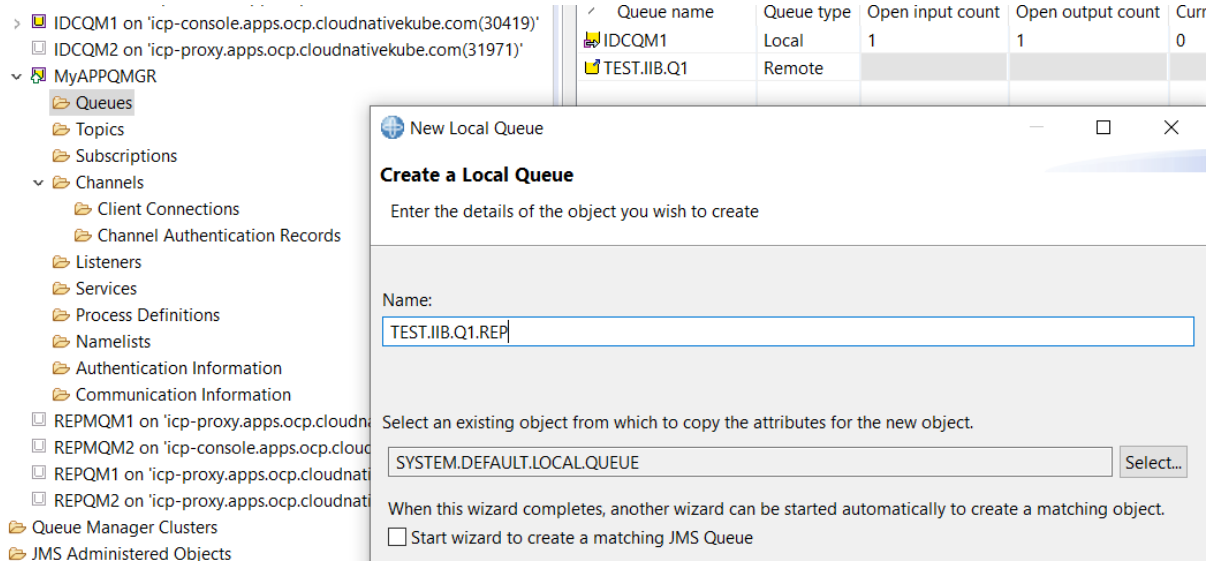
Sender channel



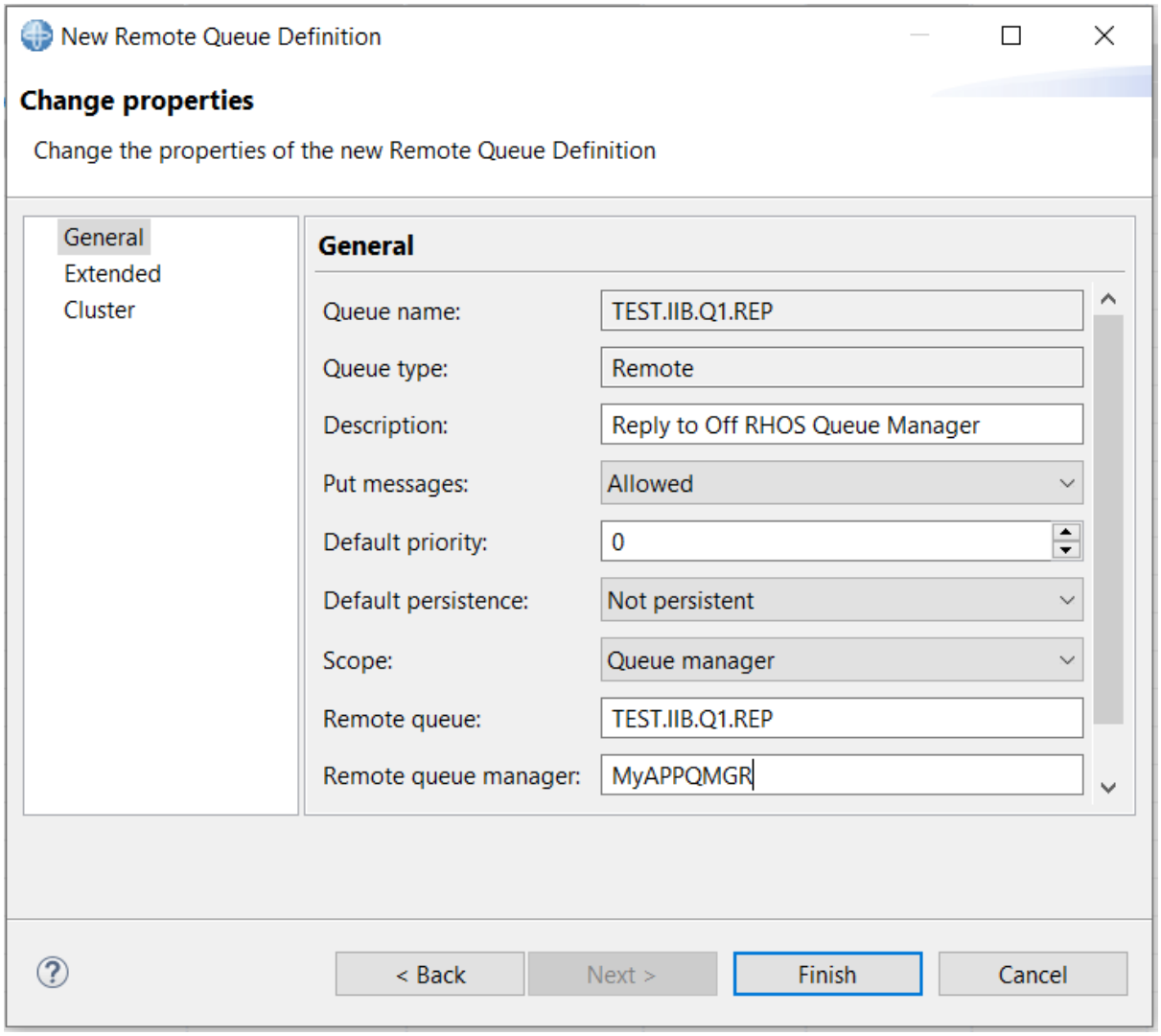
Start the channel

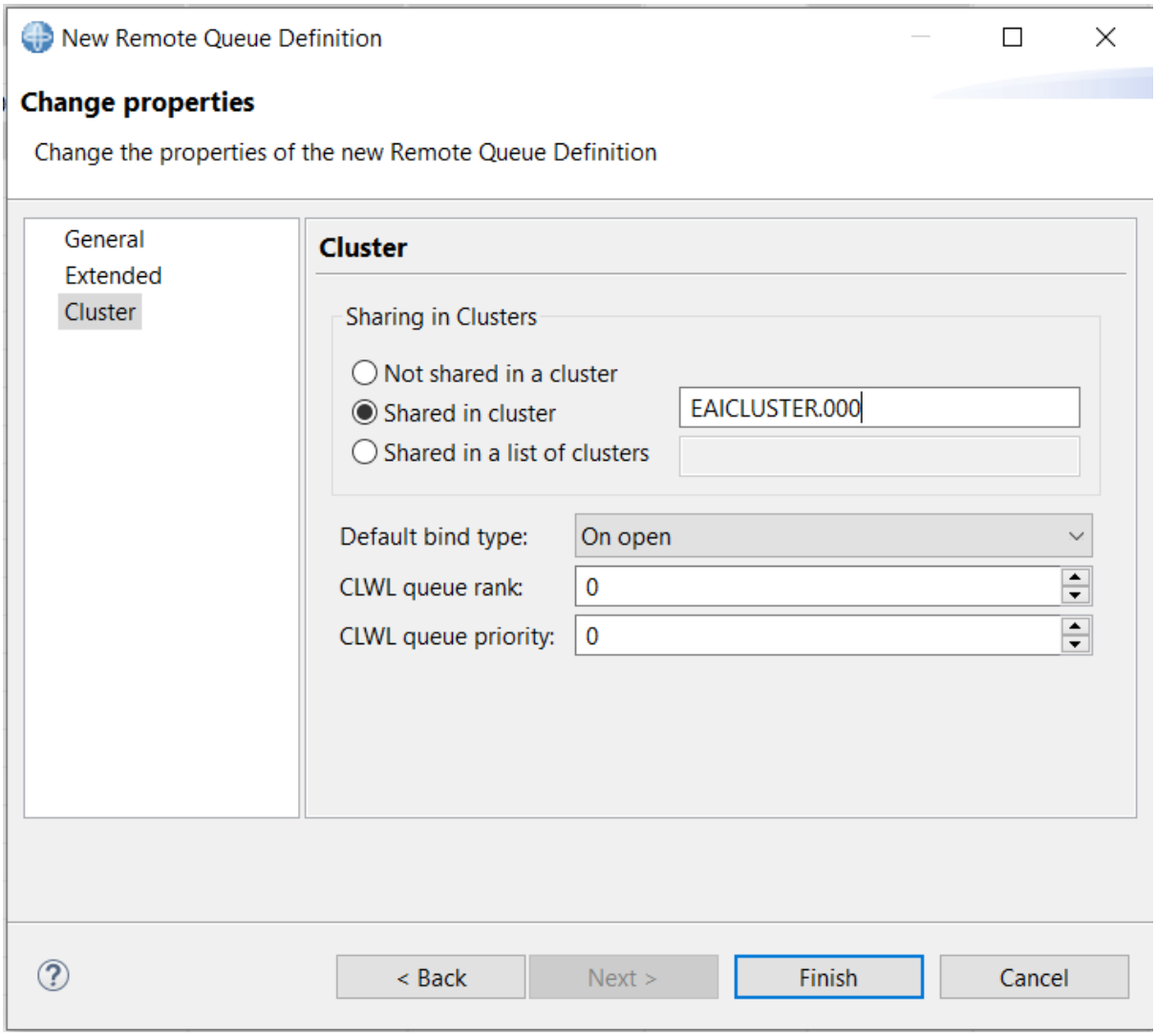


Create a local queue on “Off RHOS” Queue Manager



Create a Remote Queue Definition on IDCQM1 RHOS routing Queue Manager and share in the EAICLUSTER

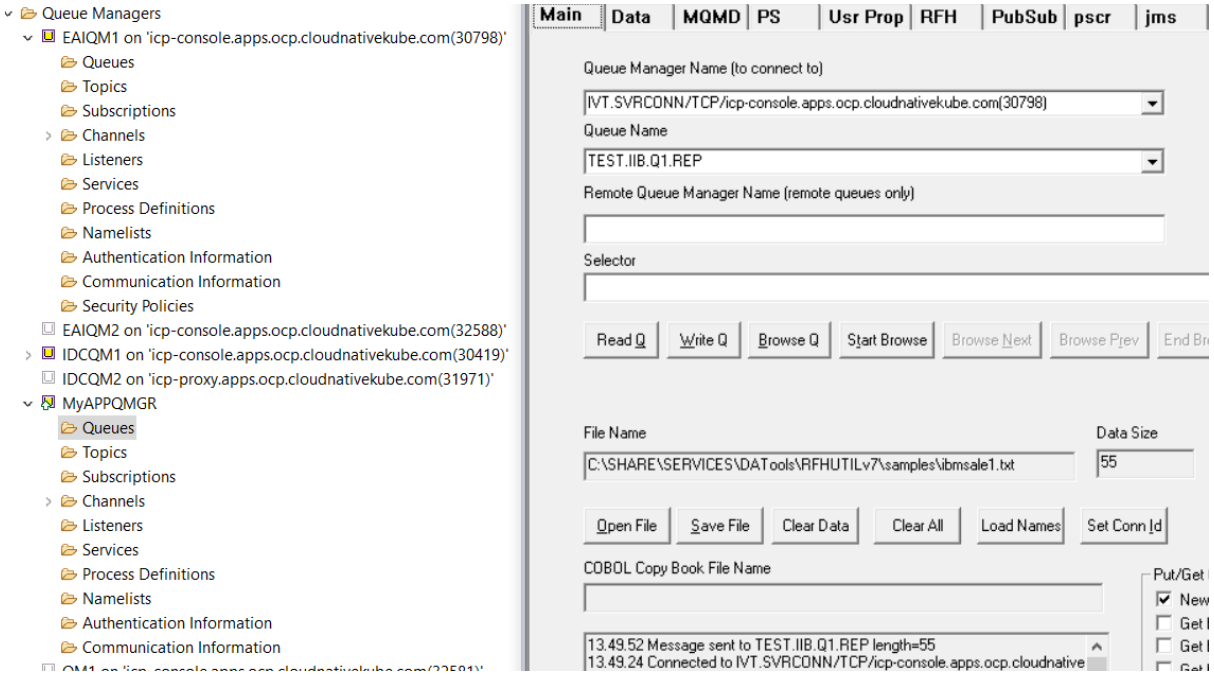

New Remote Queue Definition
Change properties
Change the properties of the new Remote Queue Definition
General
Extended
Cluster
General
Queue name:
TEST.IIB.Q1.REP
Queue type:
Remote
Description:
Reply to Off RHOS Queue Manager
Put messages:
Allowed
Default priority:
0
Default persistence:
Not persistent
Scope:
Queue manager
Remote queue:
TEST.IIB.Q1.REP
Remote queue manager:
MyAPPQMGR
< Back
Next >
Finish
Cancel

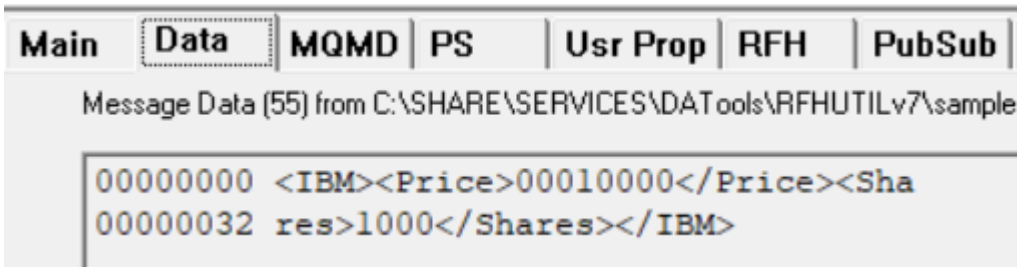


Sending a message from RHOS EAIQM1 to "Off RHOS" Queue Manager

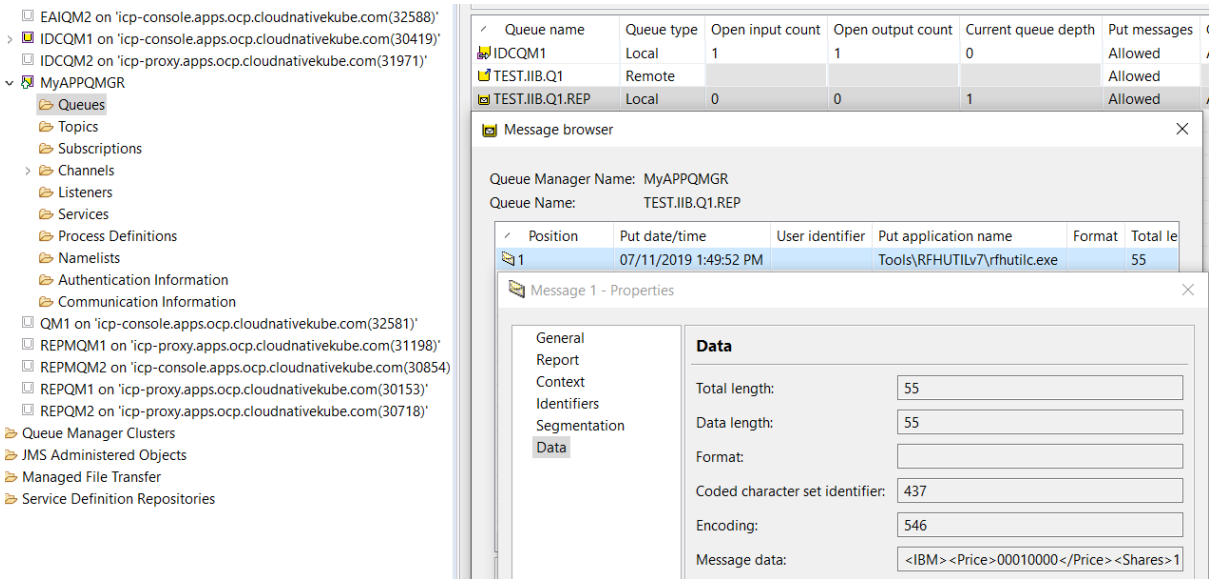
Use RFHUTILC to connect to EAIQM1

Put a message to TEST.IIB.Q1.REP





Browse the TEST.IIB.Q1.REP queue

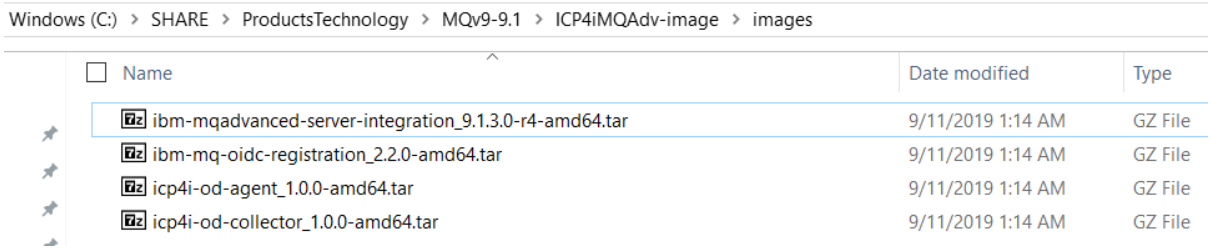


Deploy and Echo flow on EAIQM1 for TEST.IIB.Q1 and TEST.IIB.Q1.REP


```
DECLARE Delay EXTERNAL int 0;
DECLARE CopyMsgIDtoCorrID EXTERNAL boolean true;
DECLARE returnValue BOOLEAN;

CREATE COMPUTE MODULE RoundTripEcho_MsgIDCorIDandDelay
	CREATE FUNCTION Main() RETURNS BOOLEAN
	BEGIN
		SET returnValue = SLEEP(Delay);
		CALL CopyMessageHeaders();
		Set OutputRoot.BLOB.BLOB = x'4441313233'||InputRoot.BLOB.BLOB;

		If(CopyMsgIDtoCorrID)
		THEN Set OutputRoot.MQMD.CorrelId = InputRoot.MQMD.MsgId;
		END IF;


		RETURN TRUE;
	END;

	CREATE PROCEDURE CopyMessageHeaders() BEGIN
		DECLARE I INTEGER 1;
		DECLARE J INTEGER;
		SET J = CARDINALITY(InputRoot.*[]);
		WHILE I < J DO
			SET OutputRoot.*[I] = InputRoot.*[I];
			SET I = I + 1;
		END WHILE;
	END;
END MODULE;
```

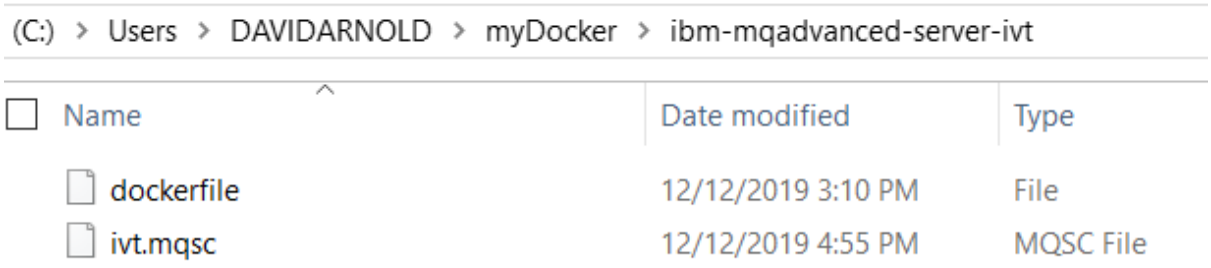
Appendix G – Building custom MQ image for client access

Load ICP4i MQAdvanced Server image to local docker repos



docker load < ibm-mqadvanced-server-integration_9.1.3.0-r4-amd64.tar.gz

```
 Directory of C:\SHARE\ProductsTechnology\MQv9-9.1\ICP4iMQAdv-image\images

13/12/2019  07:02 PM    <DIR>          .
13/12/2019  07:02 PM    <DIR>          ..
09/11/2019  01:14 AM       161,053,184 ibm-mq-oidc-registration_2.2.0-amd64.tar.gz
09/11/2019  01:14 AM       942,485,504 ibm-mqadvanced-server-integration_9.1.3.0-r4-amd64.tar.gz
09/11/2019  01:14 AM       310,161,408 icp4i-od-agent_1.0.0-amd64.tar.gz
09/11/2019  01:14 AM       321,895,424 icp4i-od-collector_1.0.0-amd64.tar.gz
               4 File(s)  1,735,595,520 bytes
               2 Dir(s)  217,878,286,336 bytes free

C:\SHARE\ProductsTechnology\MQv9-9.1\ICP4iMQAdv-image\images>docker load < ibm-mqadvanced-server-integration_9.1.3.0-r4-
amd64.tar.gz
cd28e8b3d42a: Loading layer [==================================================>]   7.68kB/7.68kB
84eaf14cef7b: Loading layer [==================================================>]  6.144kB/6.144kB
dc025946c48b: Loading layer [==================================================>]    806MB/806MB
084a5de0f8d3: Loading layer [==================================================>]   2.56kB/2.56kB
a0cf9052d6e2: Loading layer [==================================================>]  12.39MB/12.39MB
8c20517487b4: Loading layer [==================================================>]  5.876MB/5.876MB
0199fa08053a: Loading layer [==================================================>]  10.24kB/10.24kB
7f6264a97cf1: Loading layer [==================================================>]  11.78kB/11.78kB
1355ce09db14: Loading layer [==================================================>]  3.584kB/3.584kB
f9d44c67deb4: Loading layer [==================================================>]  18.28MB/18.28MB
c4b465bf4f6a: Loading layer [==================================================>]   7.68kB/7.68kB
79163b2a33c5: Loading layer [==================================================>]  2.146MB/2.146MB
2c903f0b7155: Loading layer [==================================================>]  3.072kB/3.072kB
87ba76dd2431: Loading layer [==================================================>]  3.584kB/3.584kB
1c321fdcb962: Loading layer [==================================================>]  7.168kB/7.168kB
f561b484f28b: Loading layer [==================================================>]  2.152MB/2.152MB
2b68e45b087e: Loading layer [==================================================>]  7.168kB/7.168kB
a10f6d83153f: Loading layer [==================================================>]   21.5kB/21.5kB
1cfd819e9f0e: Loading layer [==================================================>]  3.237MB/3.237MB
364112d1df0e: Loading layer [==================================================>]  261.6kB/261.6kB
9f4789d1e0ed: Loading layer [==================================================>]  3.072kB/3.072kB
07cdf8094239: Loading layer [==================================================>]  3.497MB/3.497MB
6eb6a773bf24: Loading layer [==================================================>]  3.738MB/3.738MB
ac7598f58291: Loading layer [==================================================>]  3.738MB/3.738MB
Loaded image: ibm-mqadvanced-server-integration:9.1.3.0-r4-amd64

C:\SHARE\ProductsTechnology\MQv9-9.1\ICP4iMQAdv-image\images>
```



Docker file

FROM ibm-mqadvanced-server-integration:9.1.3.0-r4-amd64

USER mqm

COPY ivt.mqsc /etc/mqm/

IVT.mqsc

DEFINE CHANNEL(IVT.SVRCONN) CHLTYPE(SVRCONN)

SET CHLAUTH(IVT.SVRCONN) TYPE(BLOCKUSER) USERLIST(nobody)

ALTER AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) AUTHTYPE(IDPWOS) CHCKCLNT(NONE) ADOPTCTX(YES)

SET CHLAUTH(IVT.SVRCONN) TYPE (ADDRESSMAP) ADDRESS(*) MCAUSER('mqm')

REFRESH SECURITY TYPE(CONNAUTH)

OR – if this is based off a non ICP4i mq image

* Connection authentication

DEFINE AUTHINFO('ORG.AUTHINFO') AUTHTYPE(IDPWOS) CHCKCLNT(REQDADM) CHCKLOCL(OPTIONAL) ADOPTCTX(YES) REPLACE

ALTER QMGR CONNAUTH('ORG.AUTHINFO')

REFRESH SECURITY(*) TYPE(CONNAUTH)

* Channels (Application + Admin)

DEFINE CHANNEL('IVT.SVRCONN') CHLTYPE(SVRCONN) REPLACE

DEFINE CHANNEL('ORG.APP.SVRCONN') CHLTYPE(SVRCONN) MCAUSER('app') REPLACE

* Channel authentication rules

SET CHLAUTH('*') TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(NOACCESS) DESCR('Back-stop rule - Blocks everyone') ACTION(REPLACE)

SET CHLAUTH('ORG.APP.SVRCONN') TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(CHANNEL) CHCKCLNT(REQUIRED) DESCR('Allows connection via APP channel') ACTION(REPLACE)

SET CHLAUTH('IVT.SVRCONN') TYPE(BLOCKUSER) USERLIST('nobody') DESCR('Allows admins on ADMIN channel') ACTION(REPLACE)

SET CHLAUTH('IVT.SVRCONN') TYPE(USERMAP) CLNTUSER('admin') USERSRC(CHANNEL) DESCR('Allows admin user to connect via ADMIN channel') ACTION(REPLACE)

* Authority records

SET AUTHREC GROUP('mqclient') OBJTYPE(QMGR) AUTHADD(CONNECT,INQ)

SET AUTHREC PROFILE('ORG.**') GROUP('mqclient') OBJTYPE(QUEUE) AUTHADD(BROWSE,GET,INQ,PUT)

SET AUTHREC PROFILE('ORG.**') GROUP('mqclient') OBJTYPE(TOPIC) AUTHADD(PUB,SUB)

Docker build

C:\Users\DAVIDARNOLD\myDocker\ibm-mqadvanced-server-ivt>docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ace11002mqc91intms1	1.0	ec39bb5df84f	2 days ago	1.59GB
ibmcom/mq	latest	268baf40303f	7 days ago	927MB
ibmcom/ace-mq	latest	0f5a883d8a95	4 weeks ago	2.51GB
ibmcom/ace-mqclient	latest	d71cee6dfd5f	4 weeks ago	1.94GB
ibmcom/ace	latest	d33c5a966d7b	4 weeks ago	1.63GB
ibm-mqadvanced-server-integration	9.1.3.0-r4-amd64	ee41039b9552	5 weeks ago	932MB

C:\Users\DAVIDARNOLD\myDocker\ibm-mqadvanced-server-ivt> docker build -t davexacom/ibm-mqadvanced-server-ivt:9.1.3.0-r4-amd64 .

```
C:\Users\DAVIDARNOLD\myDocker\mqsrvmqsc>docker build -t davexacom/ibm-mqadvanced-server-ivt:9.1.3.0-r4-amd64 .
Sending build context to Docker daemon  4.096kB
Step 1/3 : FROM ibmcom/mq
 ---> 268baf40303f
Step 2/3 : USER mqm
 ---> Using cache
 ---> f976d4712a79
Step 3/3 : COPY ivt.mqsc /etc/mqm/
 ---> Using cache
 ---> 0c686c880b92
Successfully built 0c686c880b92
Successfully tagged davexacom/ibm-mqadvanced-server-ivt:9.1.3.0-r4-amd64
```

Local test

C:\Users\DAVIDARNOLD\myDocker\ibm-mqadvanced-server-ivt>docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
davexacom/ibm-mqadvanced-server-ivt	9.1.3.0-r4-amd64	0c686c880b92	3 days ago	927MB

docker run -e LICENSE=accept -e MQ_QMGR_NAME=QM1 -P davexacom/ibm-mqadvanced-server-ivt:9.1.3.0-r4-amd64

C:\Users\DAVIDARNOLD\myDocker\ibm-mqadvanced-server-ivt>docker run -e LICENSE=accept -e MQ_QMGR_NAME=QM1 -P davexacom/ibm-mqadvanced-server-ivt:9.1.3.0-r4-amd64

2019-12-13T08:29:20.120Z CPU architecture: amd64

2019-12-13T08:29:20.121Z Linux kernel version: 4.9.184-linuxkit

2019-12-13T08:29:20.121Z Container runtime: docker

2019-12-13T08:29:20.121Z Base image: Red Hat Enterprise Linux Server 7.7 (Maipo)

Break

Starting MQSC for queue manager QM1.

1 : DEFINE CHANNEL(IVT.SVRCONN) CHLTYPE(SVRCONN)

AMQ8014I: IBM MQ channel created.

2 : SET CHLAUTH(IVT.SVRCONN) TYPE(BLOCKUSER) USERLIST(nobody)

AMQ8877I: IBM MQ channel authentication record set.

3 : ALTER AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) AUTHTYPE(IDPWOS) CHCKCLNT(NONE) ADOPTCTX(YES)

AMQ8567I: IBM MQ authentication information changed.

4 : SET CHLAUTH(IVT.SVRCONN) TYPE (ADDRESSMAP) ADDRESS(*) MCAUSER('mqm')

AMQ8877I: IBM MQ channel authentication record set.

5 : REFRESH SECURITY TYPE(CONNAUTH)

AMQ8560I: IBM MQ security cache refreshed.

5 MQSC commands read.

No commands have a syntax error.

All valid MQSC commands were processed.2019-12-13T08:29:22.125Z

Metrics are disabled

2019-12-13T09:09:17.684Z AMQ8024I: IBM MQ channel initiator started.

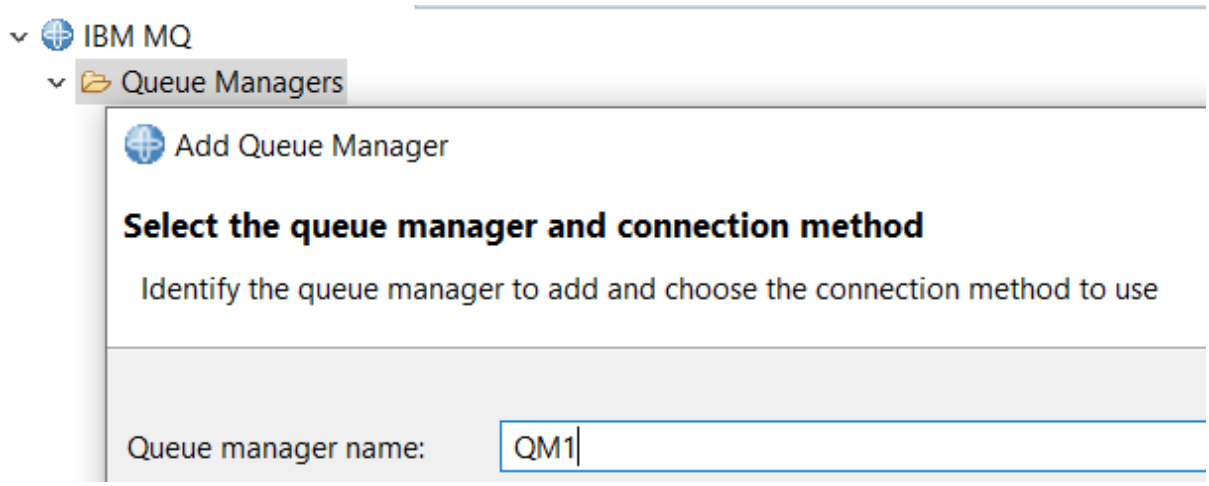
2019-12-13T09:09:17.706Z AMQ5806I: Queued Publish/Subscribe Daemon started for queue manager QM1.

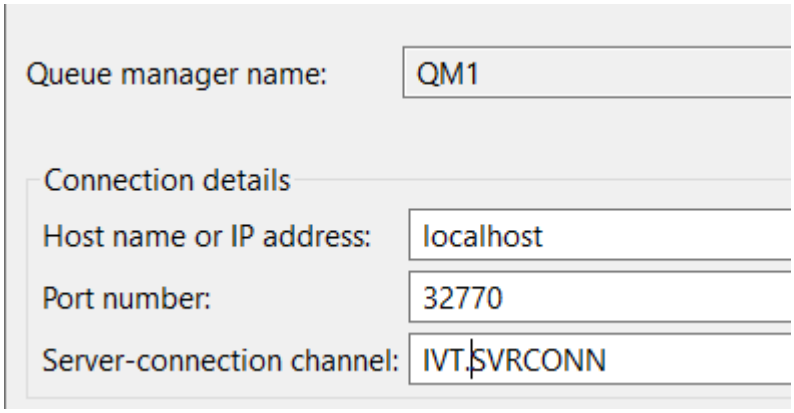
C:\Users\DAVIDARNOLD\myDocker\ibm-mqadvanced-server-ivt>
C:\Users\DAVIDARNOLD\myDocker\ibm-mqadvanced-server-ivt>docker ps

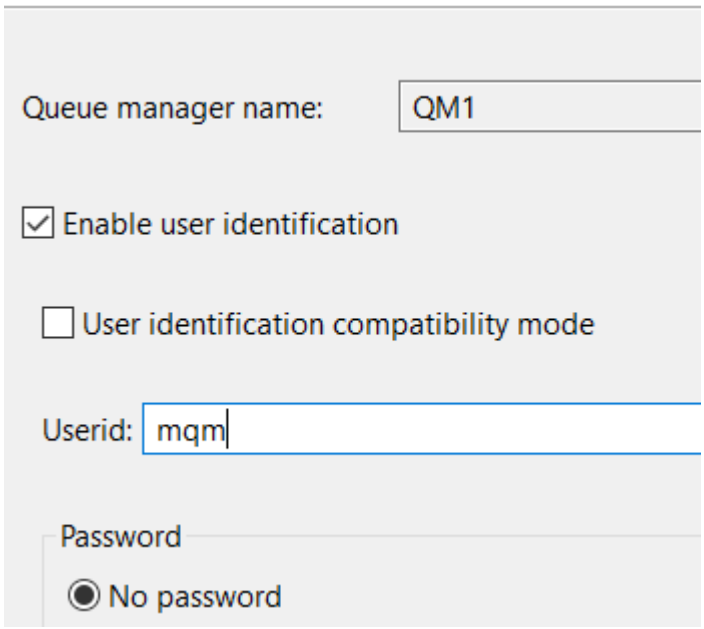
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

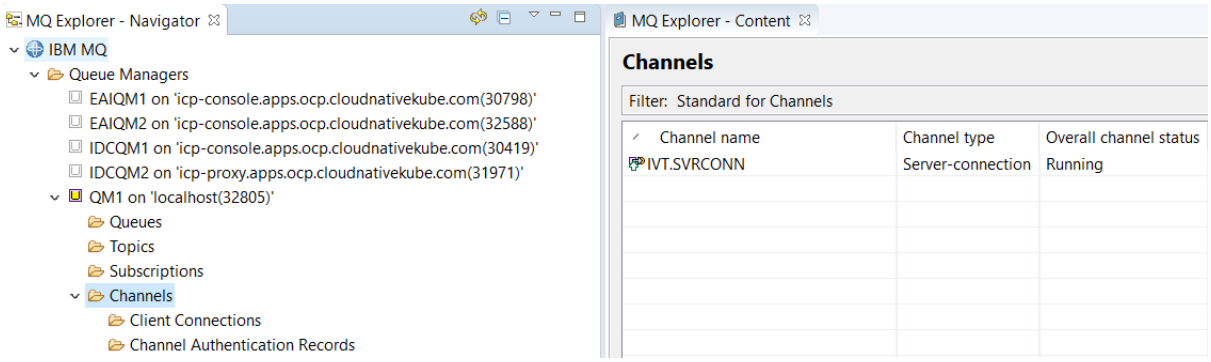
b5e2c8034d8c davexacom/ibm-mqadvanced-server-ivt:9.1.3.0-r4-amd64 "runmqintegrationser…" 26 seconds ago Up 25 seconds 0.0.0.0:32770->1414/tcp, 0.0.0.0:32769->9157/tcp, 0.0.0.0:32768->9443/tcp relaxed_lamport

Connect MQ Explorer









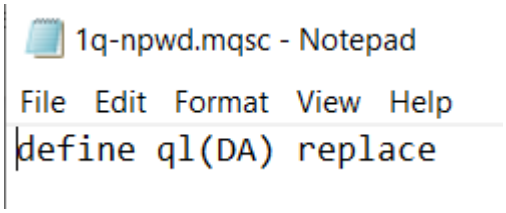
Access via RUNMQSC client

SET MQSERVER=IVT.SVRCONN/TCP/localhost(32770)

manual

```
C:\Users\DAVIDARNOLD\myDocker\ibm-mqadvanced-server-ivt>runmqsc -c QM1
5724-H72 (C) Copyright IBM Corp. 1994, 2019.
Starting MQSC for queue manager QM1.


def ql(DA.Q)
     1 : def ql(DA.Q)
AMQ8006I: IBM MQ queue created.
end
     2 : end
2 command responses received.
```

scripted



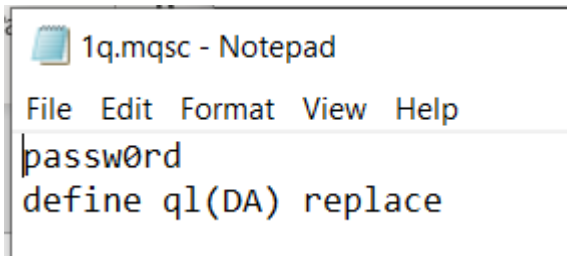
```
C:\temp>SET MQSERVER=IVT.SVRCONN/TCP/localhost(32805)

C:\temp>runmqsc -c QM1 < 1q-npwd.mqsc
5724-H72 (C) Copyright IBM Corp. 1994, 2019.
Starting MQSC for queue manager QM1.


     1 : define ql(DA) replace
AMQ8006I: IBM MQ queue created.
2 command responses received.
```

Note if you are using a queue manager where the username is password protected

Manual running of mqsc

```
C:\temp>runmqsc -c -u admin QM1
5724-H72 (C) Copyright IBM Corp. 1994, 2019.
Enter password:
********
Starting MQSC for queue manager QM1.


end
     1 : end
0 command responses received.
```

If -u is on the command and we are piping in, runmqsc assumes the first line is the password.




```
C:\temp>runmqsc -c -u admin QM1 < 1q.mqsc
5724-H72 (C) Copyright IBM Corp. 1994, 2019.
Enter password:
Starting MQSC for queue manager QM1.


     1 : define ql(DA) replace
AMQ8006I: IBM MQ queue created.
2 command responses received.
```

Push the image to Dockerhub

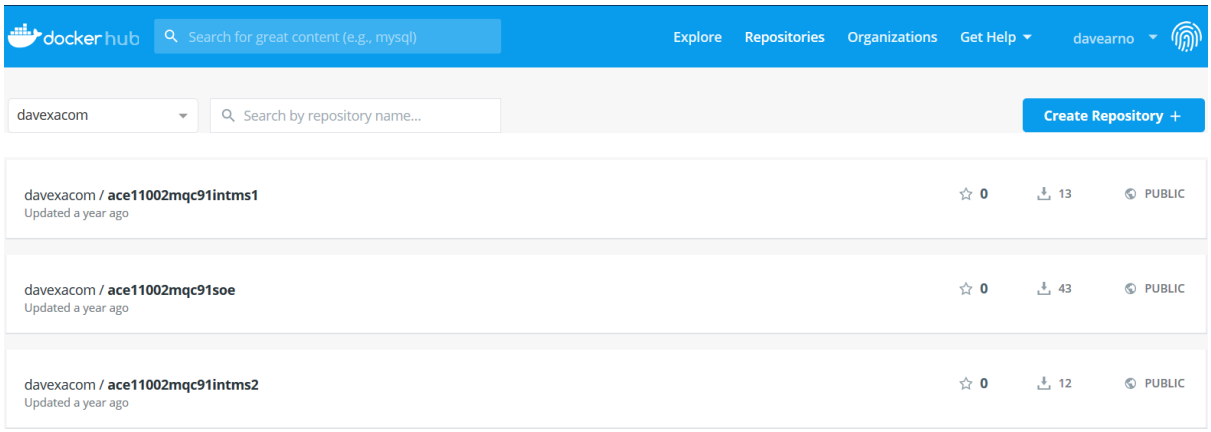
C:\Users\DAVIDARNOLD\myDocker\ibm-mqadvanced-server-ivt>docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

davexacom/ibm-mqadvanced-server-ivt 9.1.3.0-r4-amd64 2237fea38967 17 minutes ago 932MB

```
C:\Users\DAVIDARNOLD\myDocker\ibm-mqadvanced-server-ivt>docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
davexacom/ibm-mqadvanced-server-ivt   9.1.3.0-r4-amd64    e269ad126613        5 minutes ago       932MB
```

https://hub.docker.com/repositories



docker push davexacom/ibm-mqadvanced-server-ivt:9.1.3.0-r4-amd64

C:\Users\DAVIDARNOLD\myDocker\ibm-mqadvanced-server-ivt>docker push davexacom/ibm-mqadvanced-server-ivt:9.1.3.0-r4-amd64

The push refers to repository [docker.io/davexacom/ibm-mqadvanced-server-ivt]

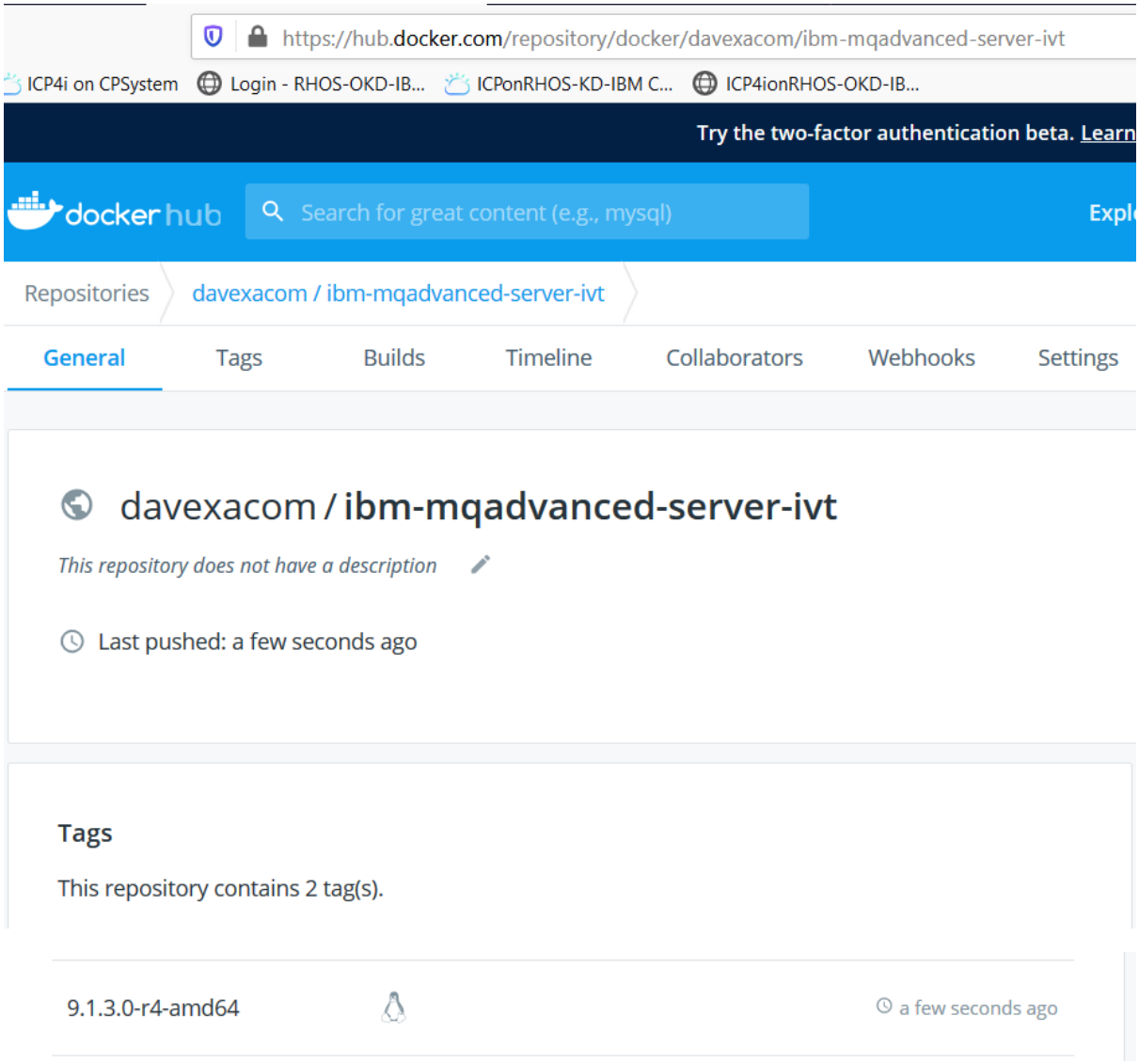
0d8459f27ecc: Pushing [==>] 3.072kB
ac7598f58291: Pushing [==================>] 1.378MB/3.735MB
6eb6a773bf24: Pushing [==================>] 1.378MB/3.735MB
07cdf8094239: Pushing [========>] 561.2kB/3.49MB
9f4789d1e0ed: Pushing [==>] 3.072kB
364112d1df0e: Waiting

cd28e8b3d42a: Pushed

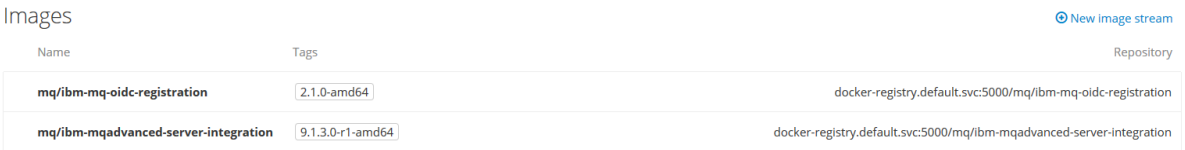
2705f79af6db: Layer already exists

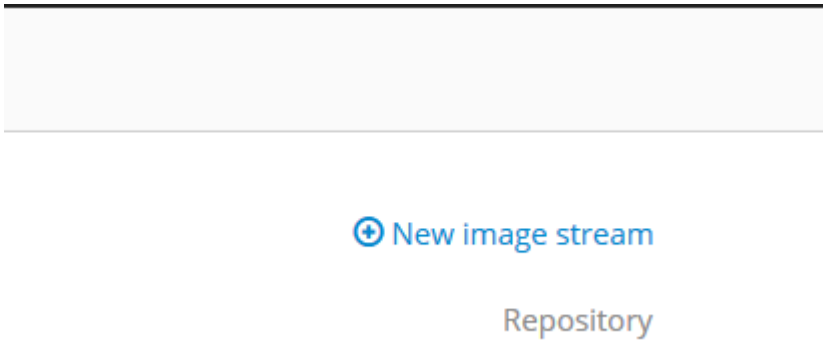
ce459cc53899: Layer already exists 9.1.3.0-r4-amd64: digest: sha256:040e8e0aca3307369b10cce8411f97e147a9a92bb7514619a16c2a78093c085f size: 5963

https://hub.docker.com/repository/docker/davexacom/ibm-mqadvanced-server-ivt



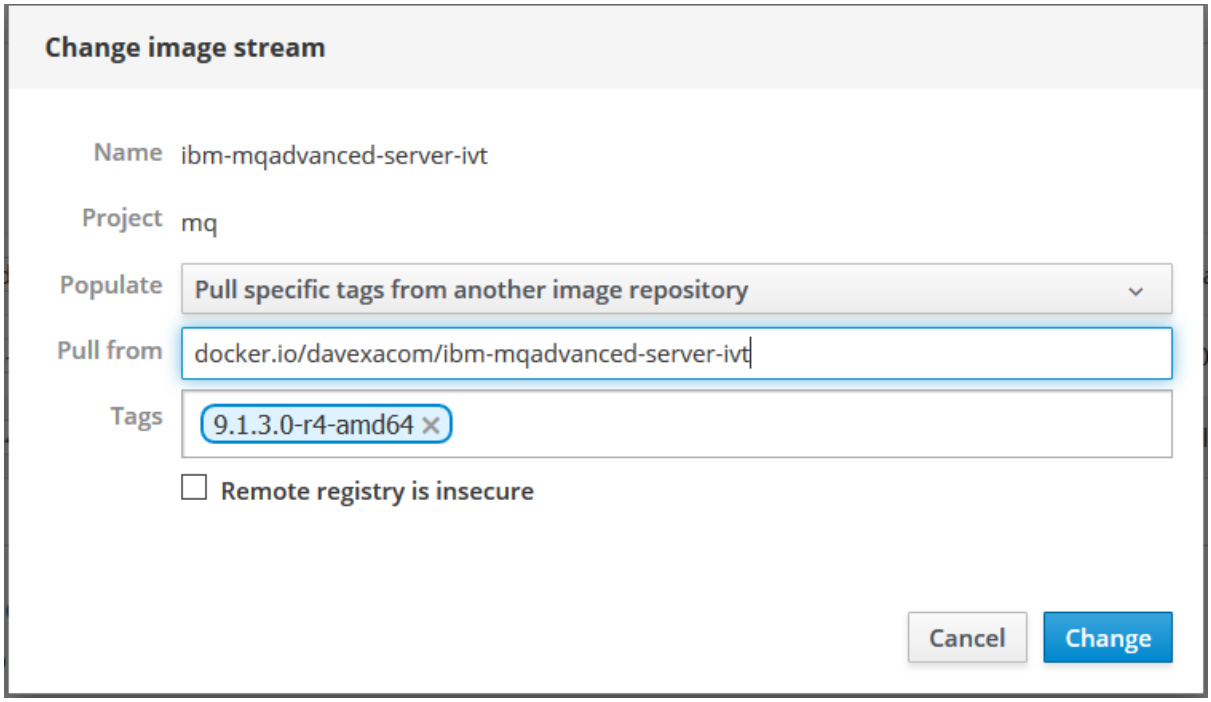
https://registry-console-default.apps.ocp.cloudnativekube.com/registry#/images/mq

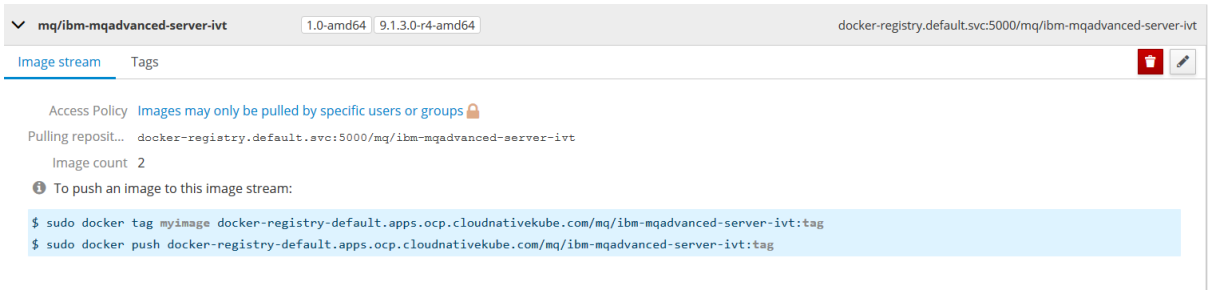




Docker.io/davexacom/ibm-mqadvanced-server-ivt

Create new image stream / or change image stream



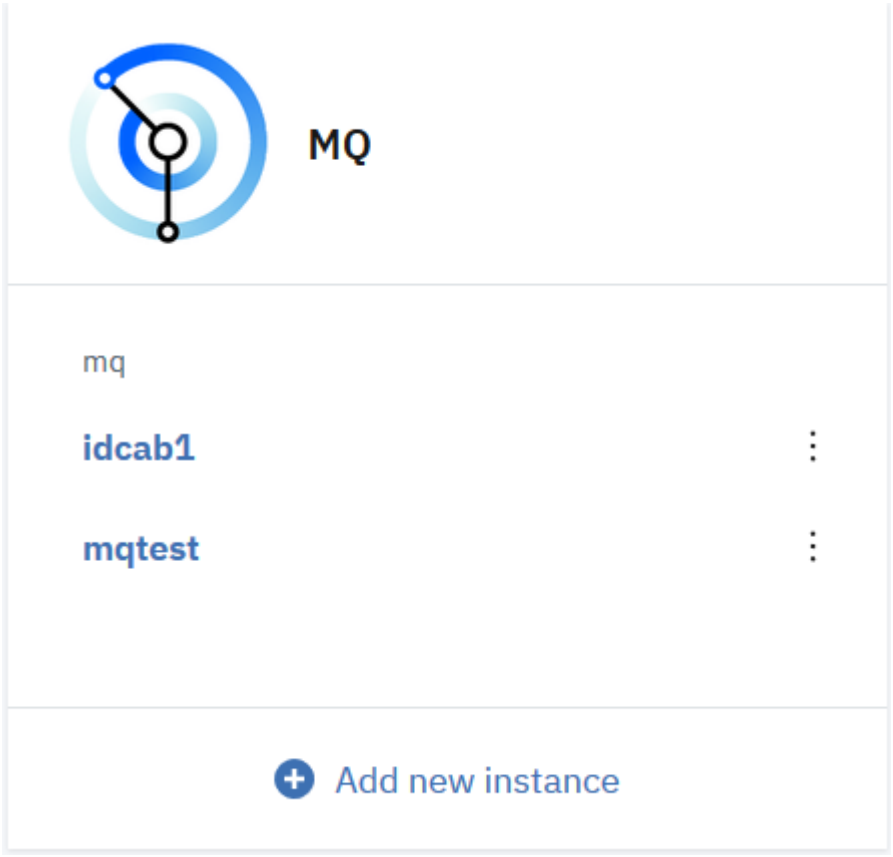


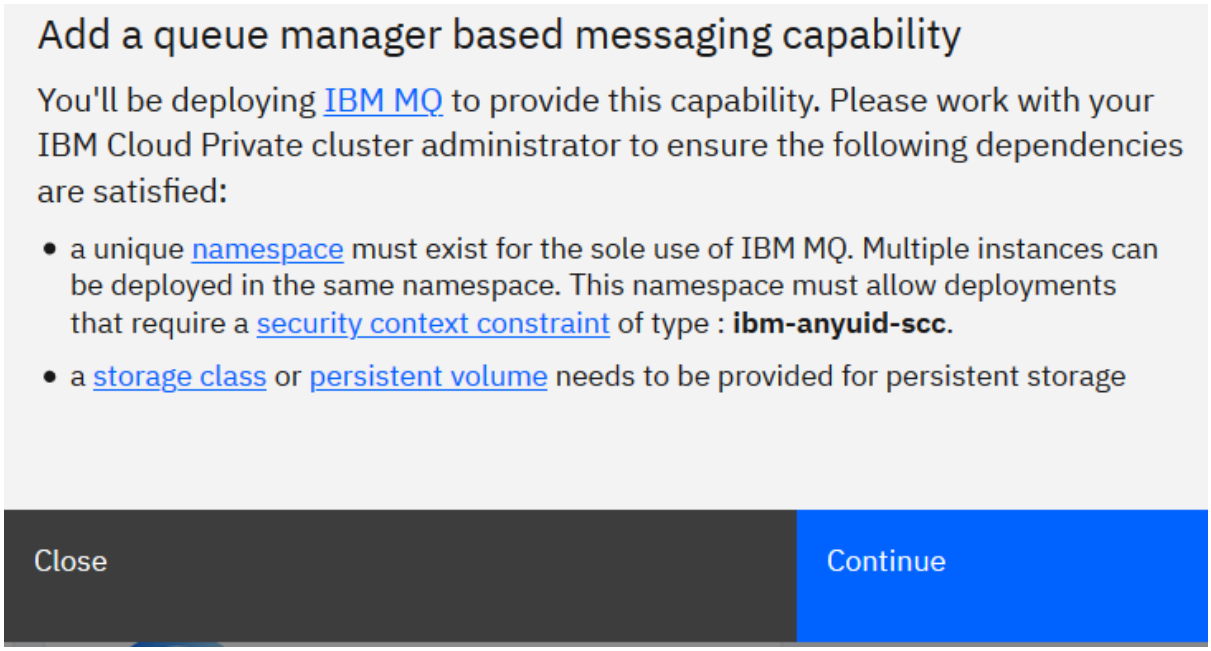
Pulling repository to use in helm charts

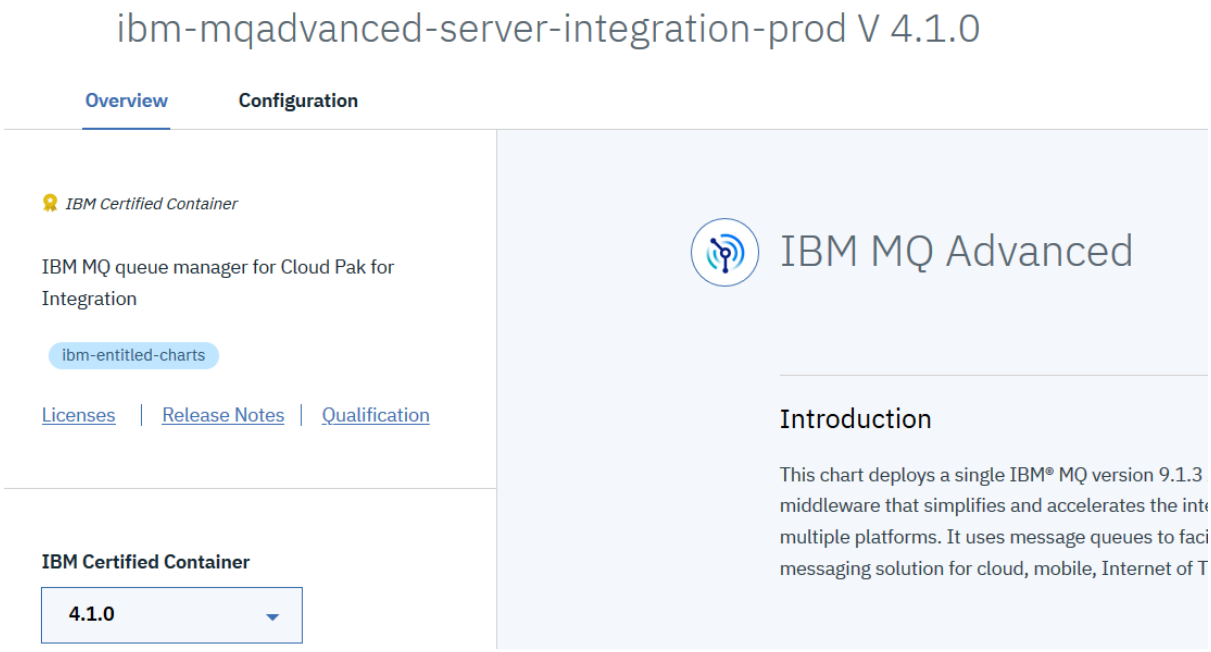
`docker-registry.default.svc:5000/mq/ibm-mqadvanced-server-ivt`

Add new ICP4i instance of MQ using custom image

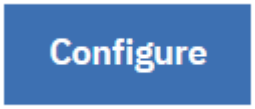
https://icp-proxy.apps.ocp.cloudnativekube.com/integration







Hit configure



Values for the charts

Cluster Hostname (for deployment)

icp-proxy.apps.ocp.cloudnativekube.com

MQ server image:

docker-registry.default.svc:5000/mq/ibm-mqadvanced-server-ivt

MQ Server image Tag:

9.1.3.0-r4

MQ Server OIDC registration image

docker-registry.default.svc:5000/mq/ibm-mq-oidc-registration

MQ Server OIDC registration image tag:

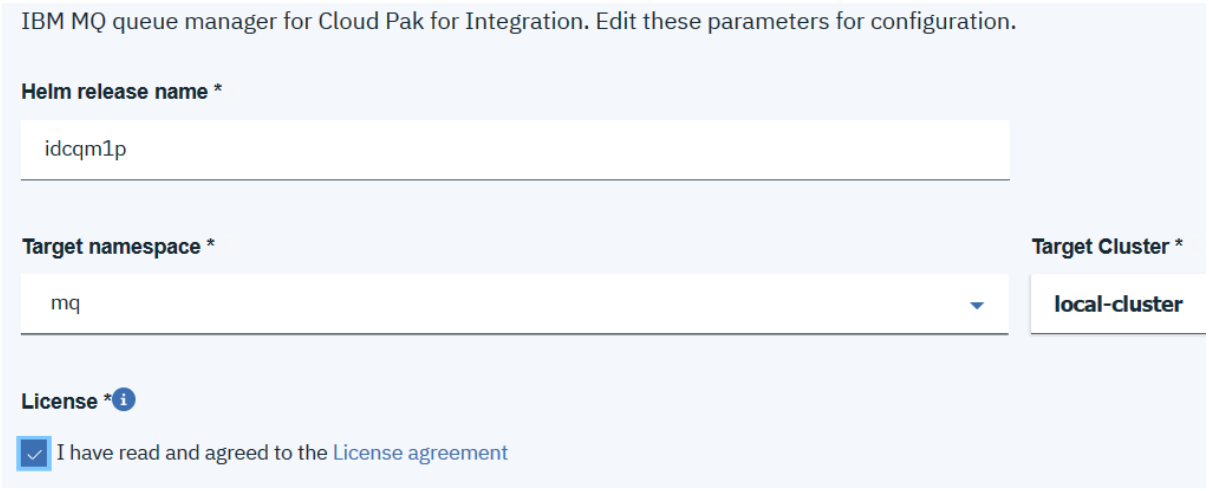
2.1.0

Managed NFS name (from oc command):

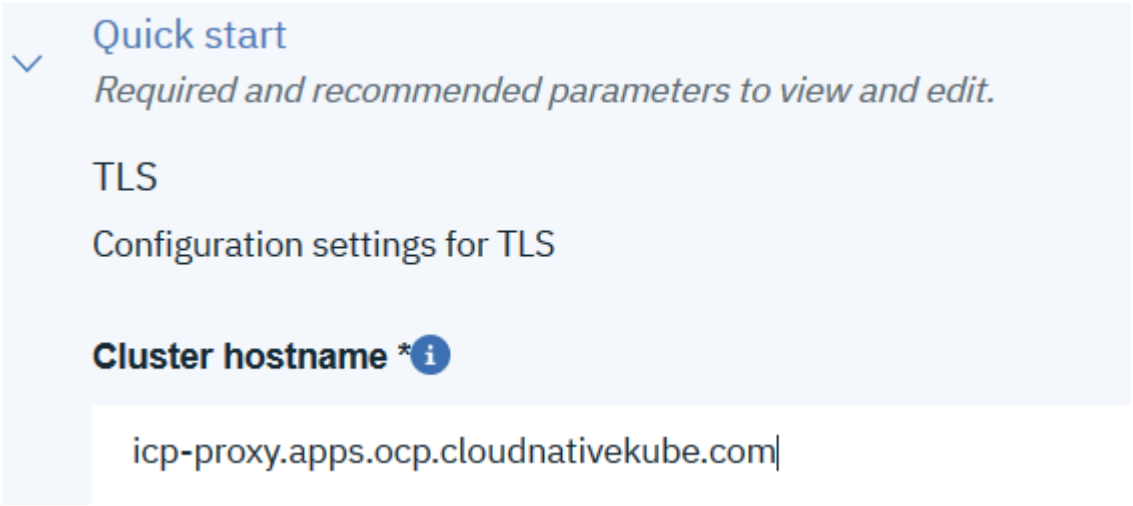
managed-nfs-storage

TLS Secret

Ibm-mq-tls-secret



icp-proxy.apps.ocp.cloudnativekube.com



MQ server image:

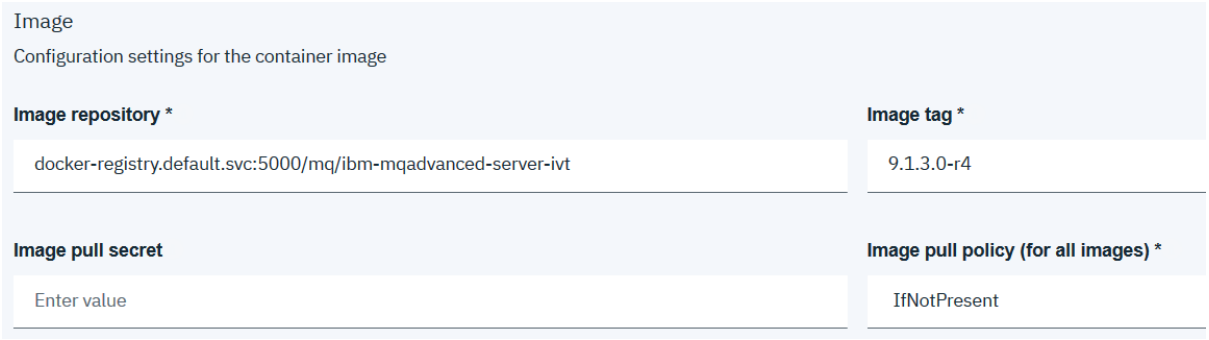
docker-registry.default.svc:5000/mq/ibm-mqadvanced-server-ivt

MQ Server image Tag:

9.1.3.0-r4

Results in the following image and tag being used

docker-registry.default.svc:5000/mq/ibm-mqadvanced-server-ict:9.1.3.0-r4-amd64

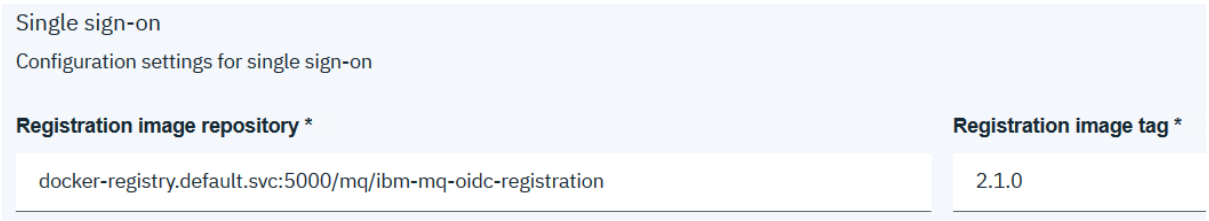


MQ Server OIDC registration image

docker-registry.default.svc:5000/mq/ibm-mq-oidc-registration

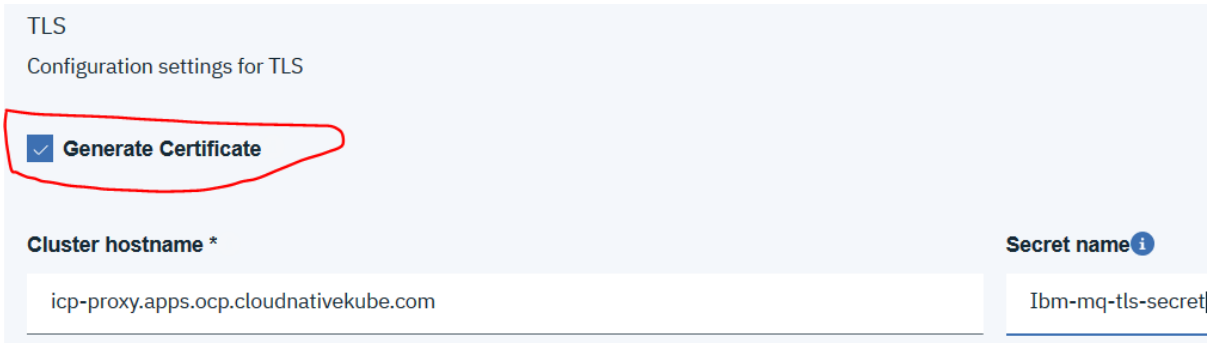
MQ Server OIDC registration image tag:

2.1.0



TLS Secret

Ibm-mq-tls-secret



Managed NFS name (from oc command):

managed-nfs-storage

Persistence

Configuration settings for Persistent Volumes

☑ **Enable persistence ***

☑ **Use dynamic provisioning ***

Data PVC

Configuration settings for the main Persistent Volume Claim

Name *

data

Storage Class name

managed-nfs-storage

Size *

2Gi

Queue manager

Configuration settings for the Queue Manager

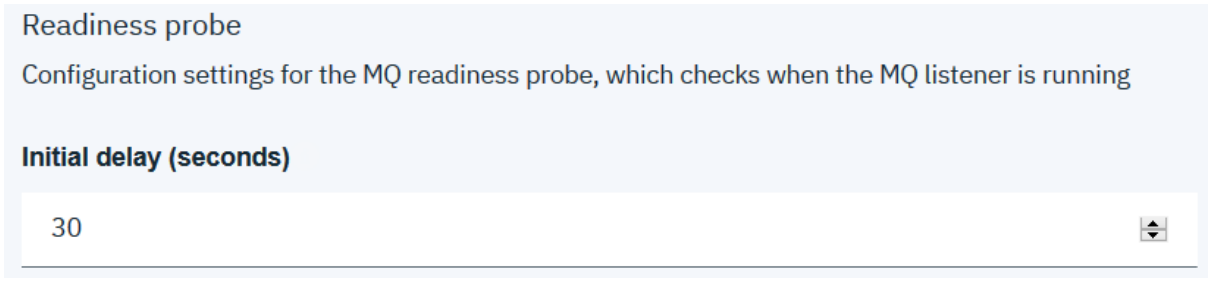
Queue manager name

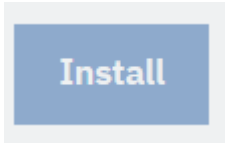
IDCQM1

Metrics

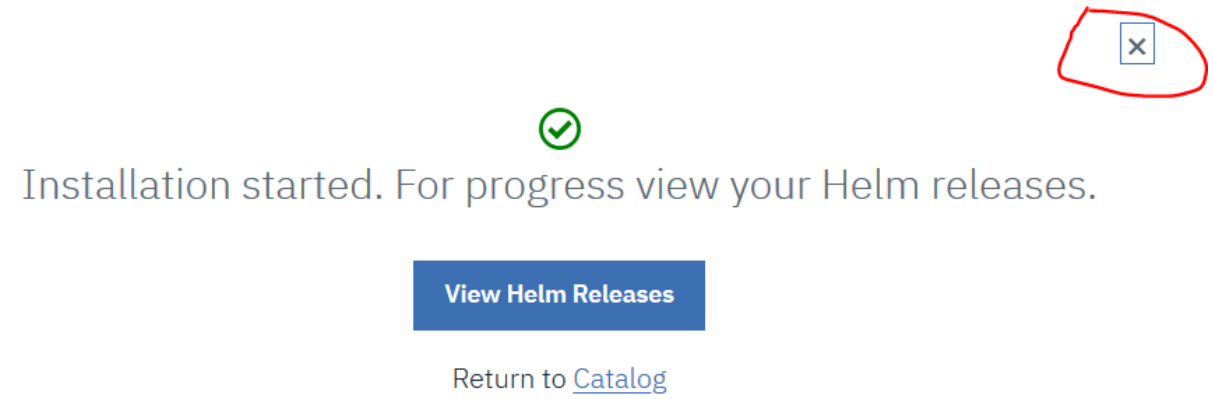
Configuration settings for Prometheus metrics

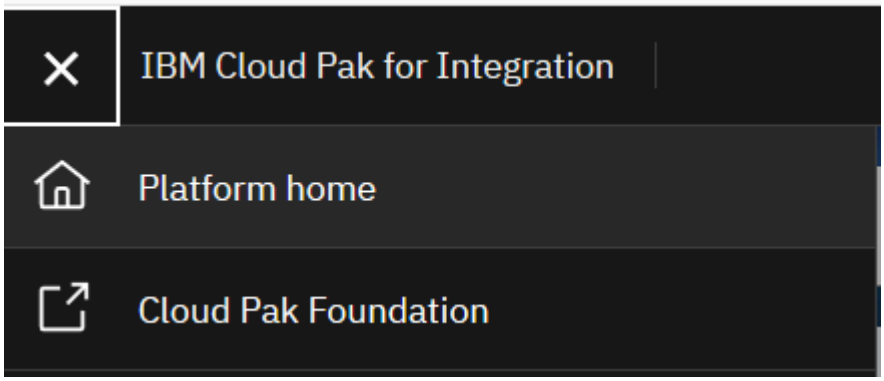
☐ **Enable metrics ***

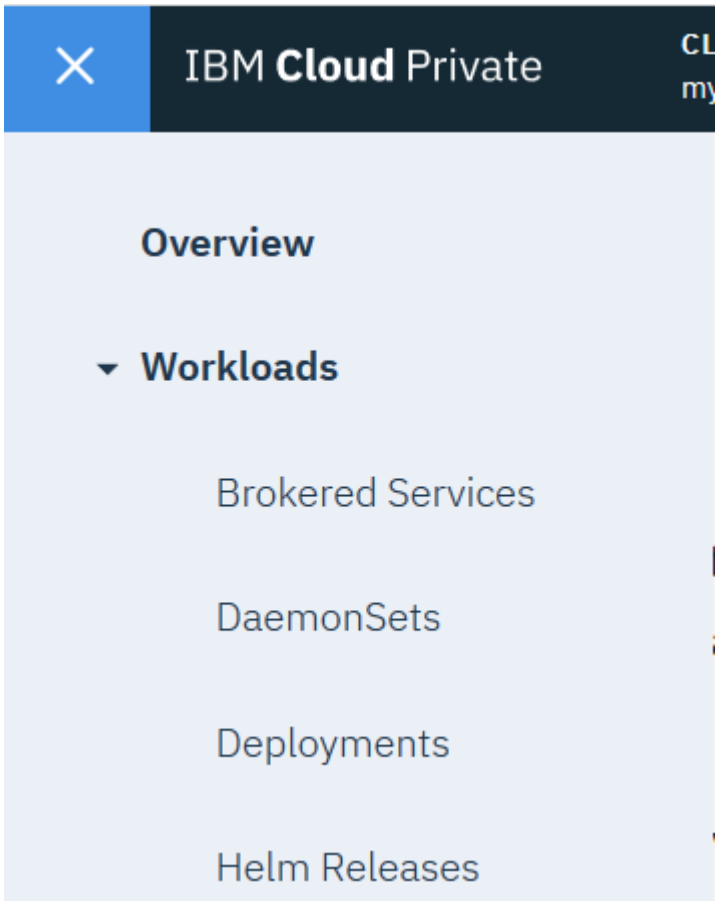




Use the little X to retain your populated chart for re-use if things go wrong.





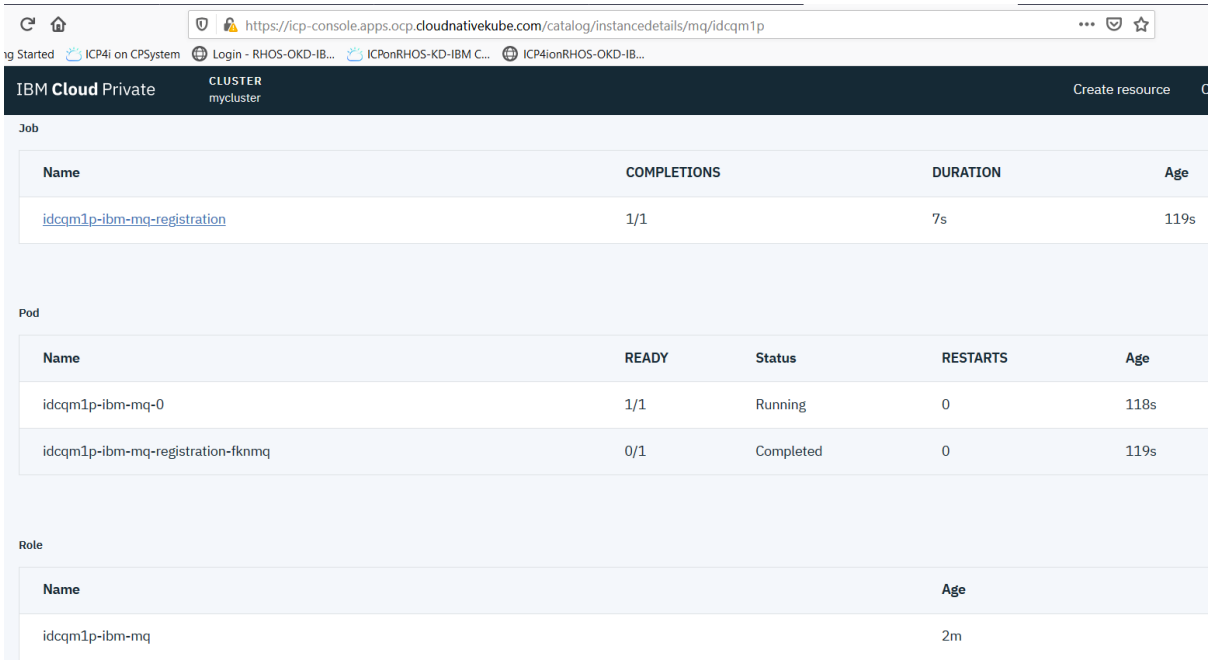


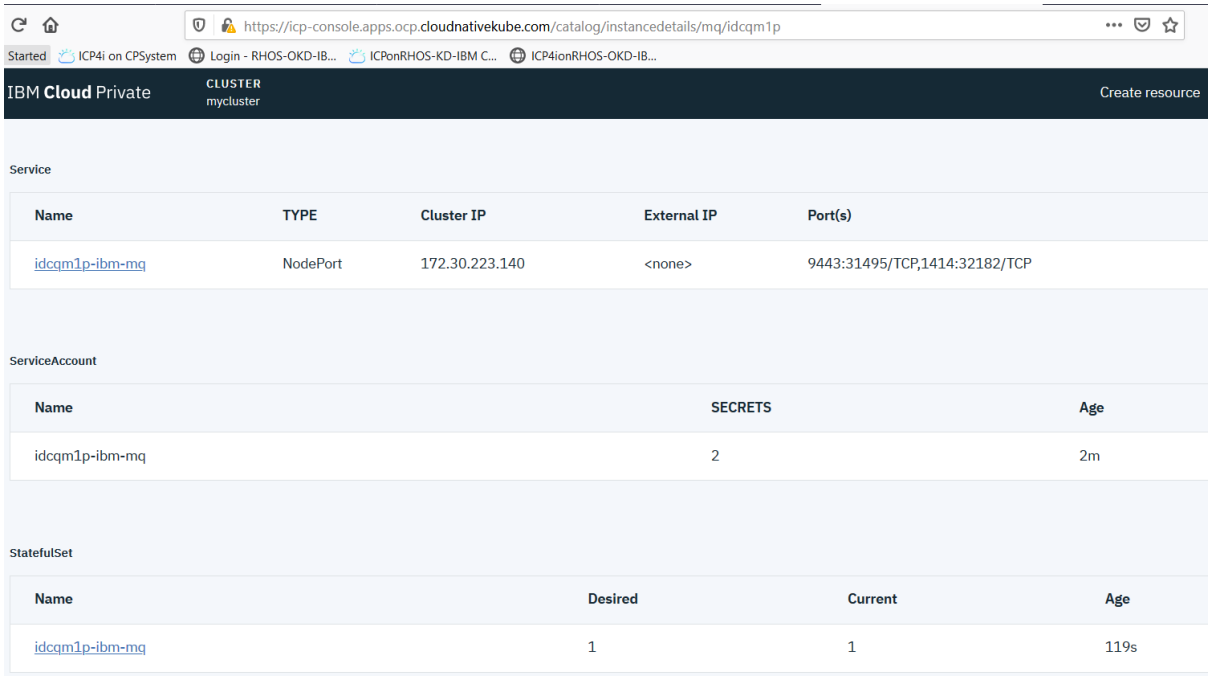
Helm Releases

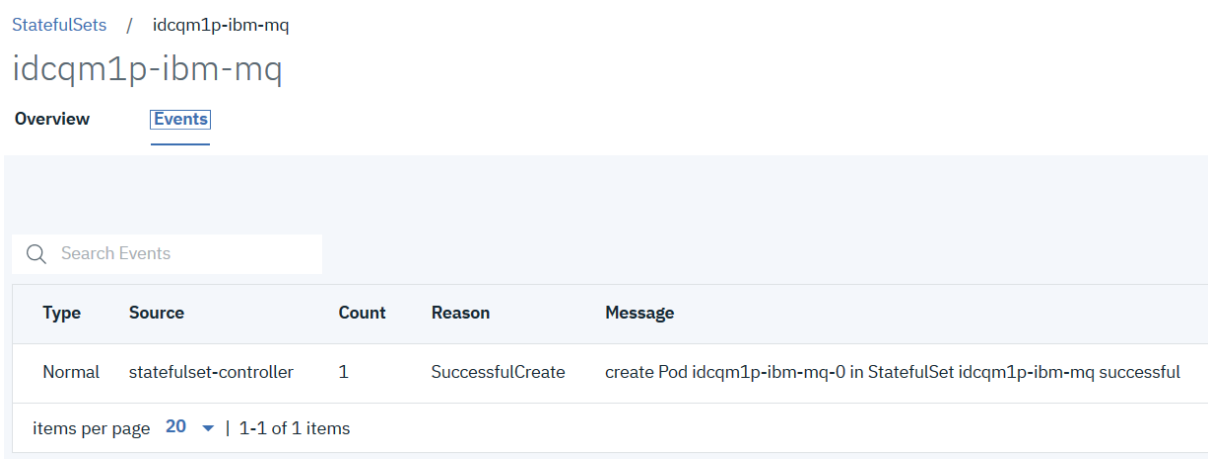
You are currently viewing only the helm releases of this cluster.

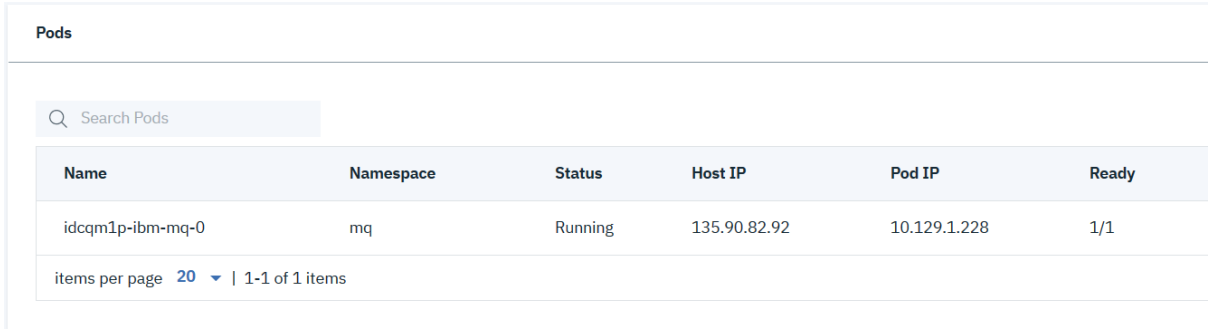
idcq

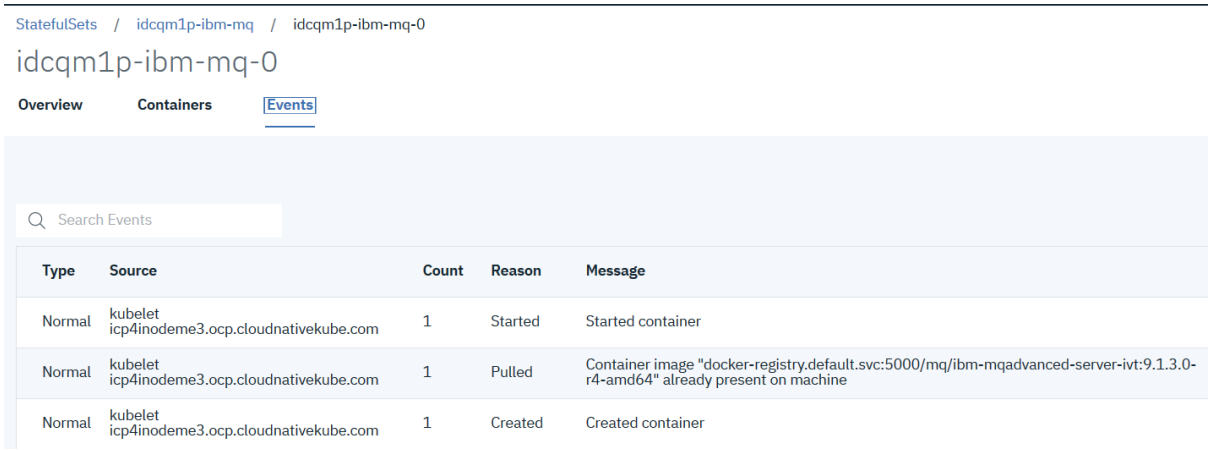
Name	Namespace	Status	Chart name
idcqm1p	mq	Deployed	ibm-mqadvanced-server-integration-prod


https://icp-console.apps.ocp.cloudnativekube.com/catalog/instancedetails/mq/idcqm1p
ICP4i on CPSystem
Login - RHOS-OKD-IB...
ICPonRHOS-KD-IBM C...
ICP4ionRHOS-OKD-IB...
IBM Cloud Private
CLUSTER
mycluster
Create resource
Job
Name
COMPLETIONS
DURATION
Age
idcqm1p-ibm-mq-registration
1/1
7s
119s
Pod
Name
READY
Status
RESTARTS
Age
idcqm1p-ibm-mq-0
1/1
Running
0
118s
idcqm1p-ibm-mq-registration-fknmq
0/1
Completed
0
119s
Role
Name
Age
idcqm1p-ibm-mq
2m

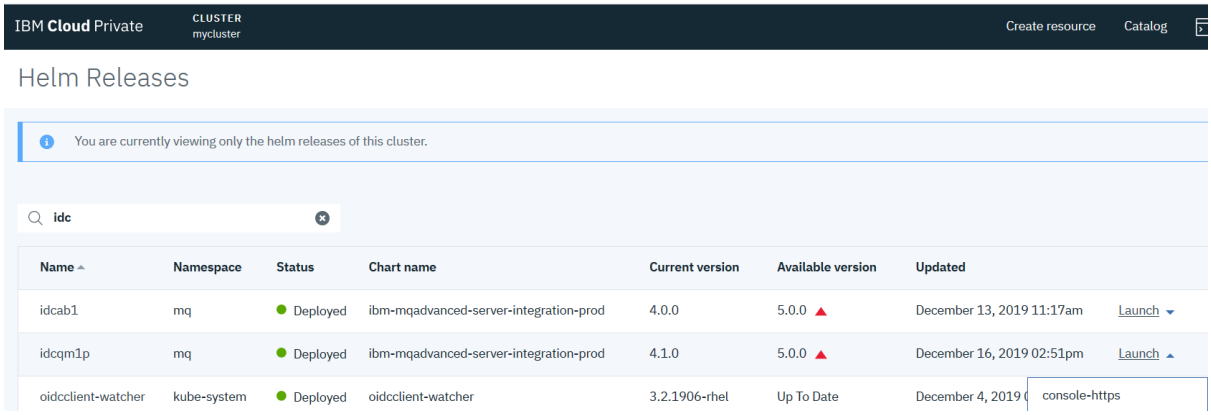

https://icp-console.apps.ocp.cloudnativekube.com/catalog/instancedetails/mq/idcqm1p
Started
ICP4i on CPSystem
Login - RHOS-OKD-IB...
ICPonRHOS-KD-IBM C...
ICP4ionRHOS-OKD-IB...
IBM Cloud Private
CLUSTER
mycluster
Create resource
Service
Name
TYPE
Cluster IP
External IP
Port(s)
idcqm1p-ibm-mq
NodePort
172.30.223.140
<none>
9443:31495/TCP,1414:32182/TCP
ServiceAccount
Name
SECRETS
Age
idcqm1p-ibm-mq
2
2m
StatefulSet
Name
Desired
Current
Age
idcqm1p-ibm-mq
1
1
119s

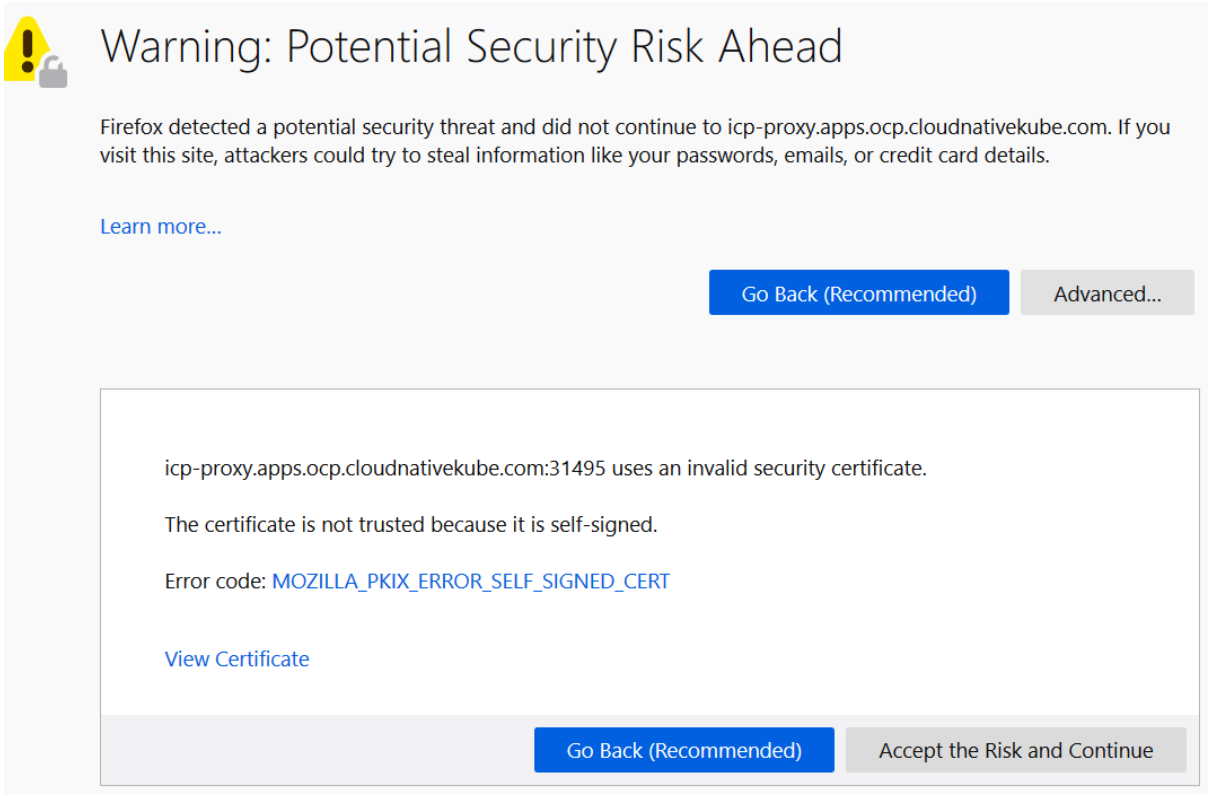

StatefulSets / idcqm1p-ibm-mq
idcqm1p-ibm-mq
Overview
Events
Search Events
Type
Source
Count
Reason
Message
Normal
statefulset-controller
1
SuccessfulCreate
create Pod idcqm1p-ibm-mq-0 in StatefulSet idcqm1p-ibm-mq successful
items per page 20 | 1-1 of 1 items



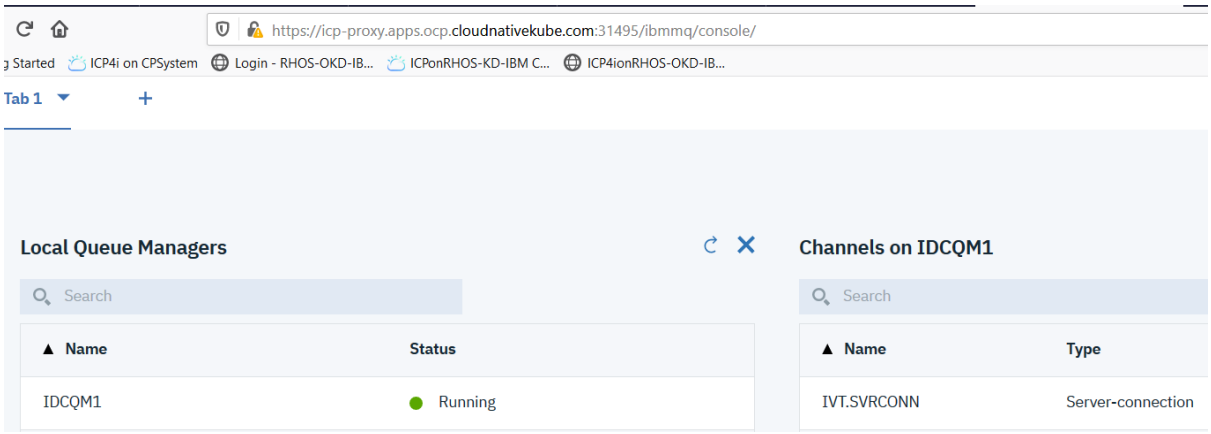


MQ Web Console



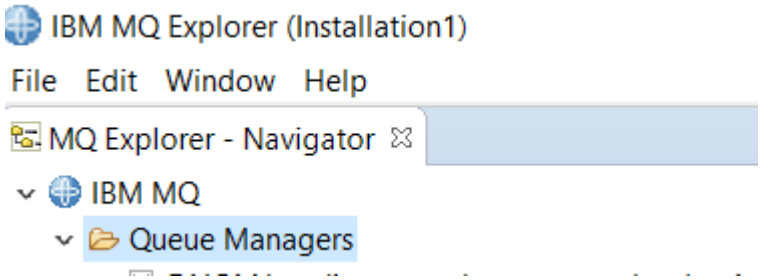


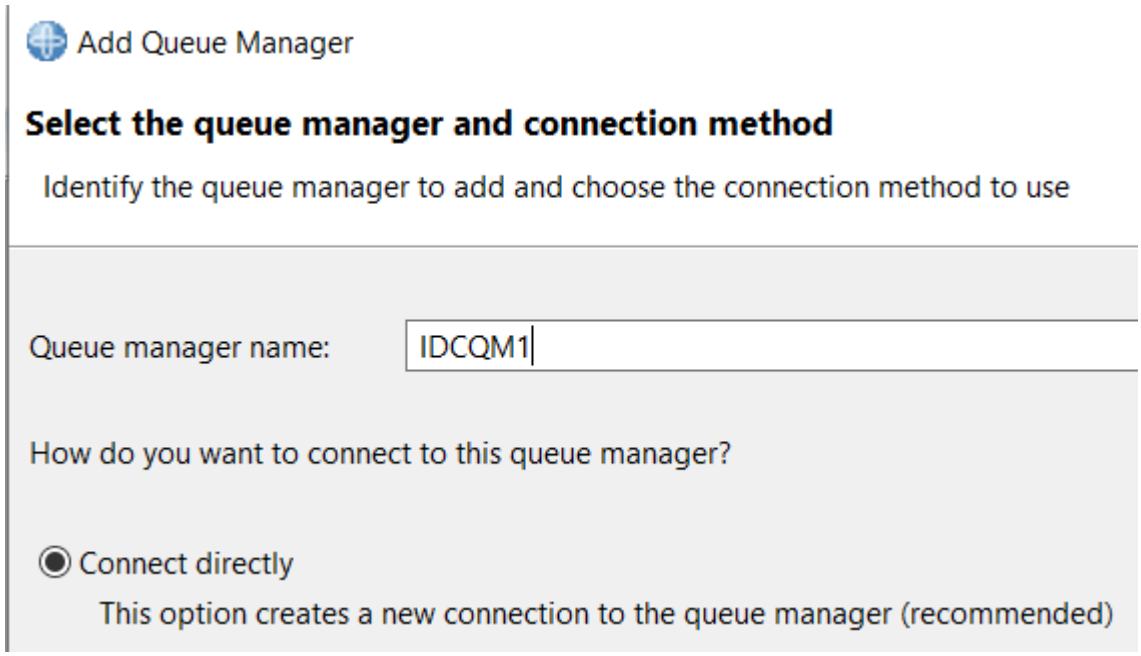
Add Widget - Channels

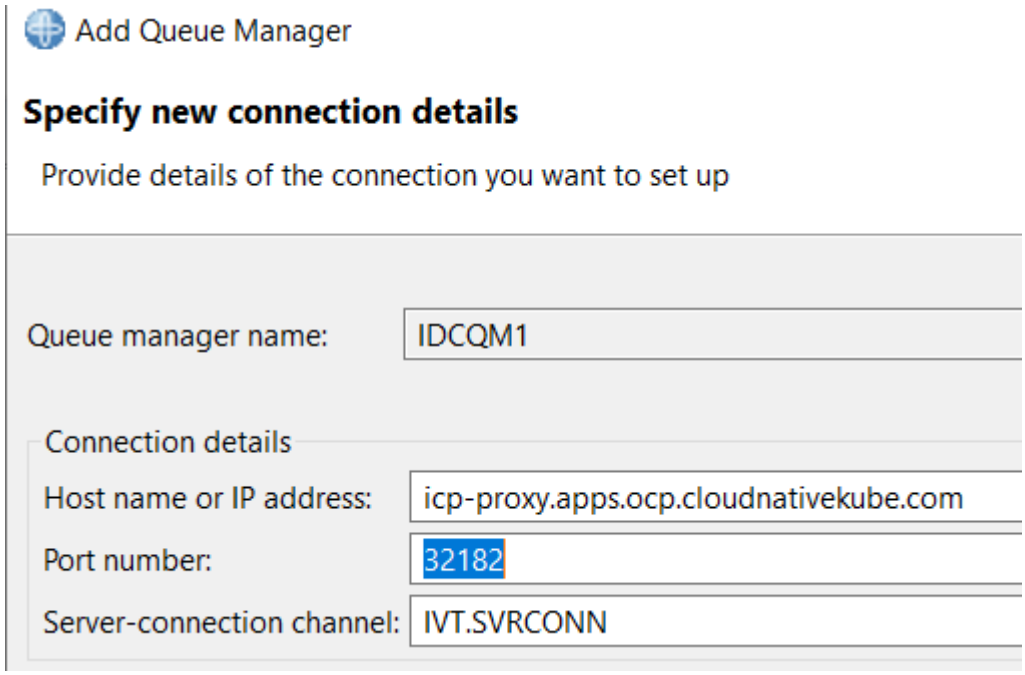


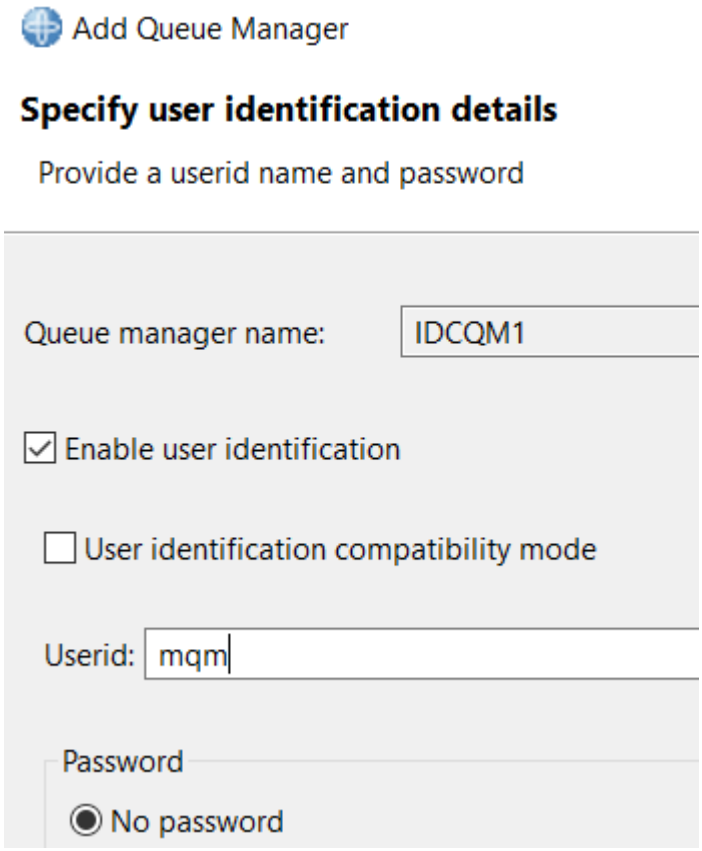
MQ Explorer

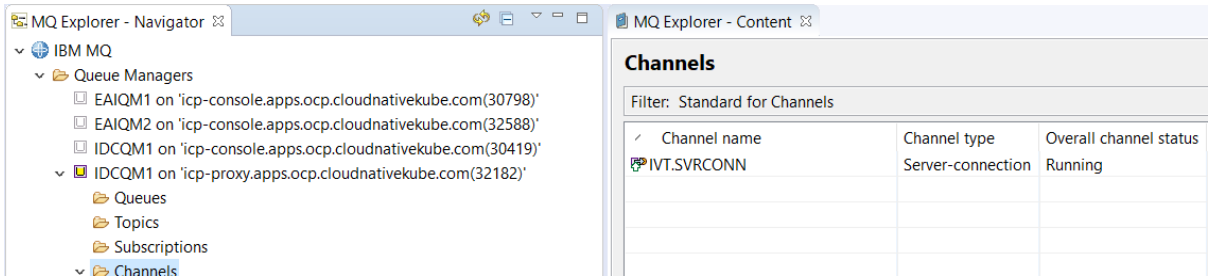
Add remote queue manager


IBM MQ Explorer (Installation1)
File Edit Window Help
MQ Explorer - Navigator
IBM MQ
Queue Managers


Add Queue Manager
Select the queue manager and connection method
Identify the queue manager to add and choose the connection method to use
Queue manager name:
IDCQM1
How do you want to connect to this queue manager?
Connect directly
This option creates a new connection to the queue manager (recommended)


Add Queue Manager
Specify new connection details
Provide details of the connection you want to set up
Queue manager name:
IDCQM1
Connection details
Host name or IP address:
icp-proxy.apps.ocp.cloudnativekube.com
Port number:
32182
Server-connection channel:
IVT.SVRCONN





Using Runmqsc in client mode

SET MQSERVER=IVT.SVRCONN/TCP/icp-proxy.apps.ocp.cloudnativekube.com(32182)

Runmqsc -c IDCQM1

```
C:\Users\DAVIDARNOLD\myDocker\ibm-mqadvanced-server-ivt>SET MQSERVER=IVT.SVRCONN/TCP/icp-proxy.apps.ocp.cloudnativekube.
com(32182)

C:\Users\DAVIDARNOLD\myDocker\ibm-mqadvanced-server-ivt>runmqsc -c IDCQM1
5724-H72 (C) Copyright IBM Corp. 1994, 2019.
Starting MQSC for queue manager IDCQM1.


def ql(manual.q)
     1 : def ql(manual.q)
AMQ8006I: IBM MQ queue created.
end
     2 : end
2 command responses received.
```

```
C:\temp>type 1q-npwd.mqsc
define ql(DA) replace

C:\temp>runmqsc -c IDCQM1 < 1q-npwd.mqsc
5724-H72 (C) Copyright IBM Corp. 1994, 2019.
Starting MQSC for queue manager IDCQM1.


     1 : define ql(DA) replace
AMQ8006I: IBM MQ queue created.
2 command responses received.
```

Appendix H – MQ client Security for non ICP4i containers

MQ containers with admin/passw0rd and app/passw0rd as default users

IBM MQ for developer container images have admin/passw0rd and app/passw0rd as default users

MQSC that sets up access for client connections.

* Connection authentication

DEFINE AUTHINFO('ORG.AUTHINFO') AUTHTYPE(IDPWOS) CHCKCLNT(REQDADM) CHCKLOCL(OPTIONAL) ADOPTCTX(YES) REPLACE

ALTER QMGR CONNAUTH('ORG.AUTHINFO')

REFRESH SECURITY(*) TYPE(CONNAUTH)

* Channels (Application + Admin)

DEFINE CHANNEL('IVT.SVRCONN') CHLTYPE(SVRCONN) REPLACE

DEFINE CHANNEL('ORG.APP.SVRCONN') CHLTYPE(SVRCONN) MCAUSER('app') REPLACE

* Channel authentication rules

SET CHLAUTH('*') TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(NOACCESS) DESCR('Back-stop rule - Blocks everyone') ACTION(REPLACE)

SET CHLAUTH('ORG.APP.SVRCONN') TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(CHANNEL) CHCKCLNT(REQUIRED) DESCR('Allows connection via APP channel') ACTION(REPLACE)

SET CHLAUTH('IVT.SVRCONN') TYPE(BLOCKUSER) USERLIST('nobody') DESCR('Allows admins on ADMIN channel') ACTION(REPLACE)

SET CHLAUTH('IVT.SVRCONN') TYPE(USERMAP) CLNTUSER('admin') USERSRC(CHANNEL) DESCR('Allows admin user to connect via ADMIN channel') ACTION(REPLACE)

* Authority records

SET AUTHREC GROUP('mqclient') OBJTYPE(QMGR) AUTHADD(CONNECT,INQ)

SET AUTHREC PROFILE('ORG.**') GROUP('mqclient') OBJTYPE(QUEUE) AUTHADD(BROWSE,GET,INQ,PUT)

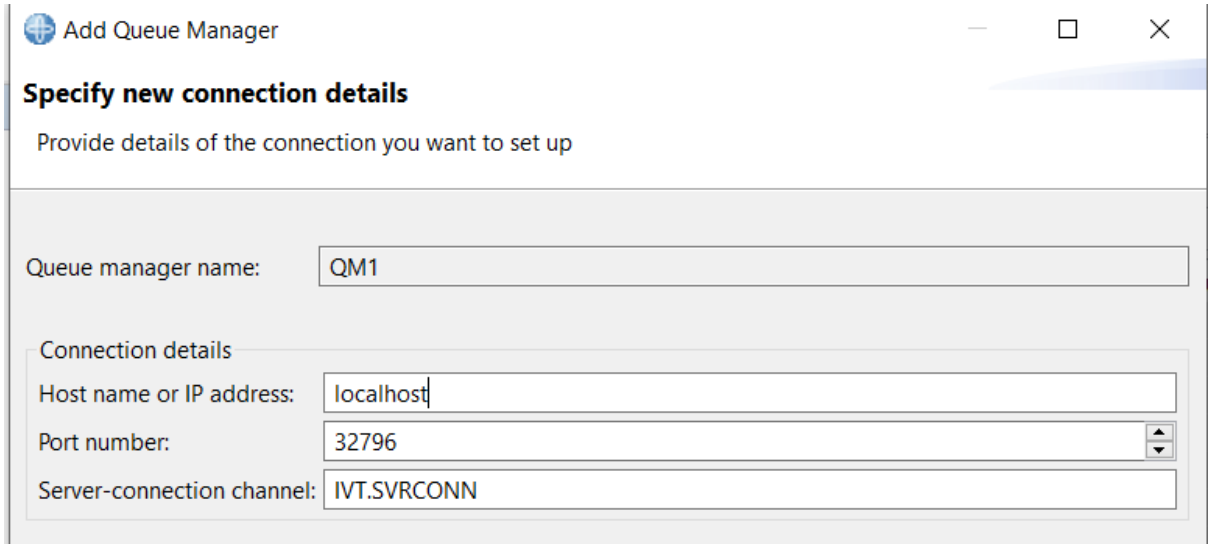
SET AUTHREC PROFILE('ORG.**') GROUP('mqclient') OBJTYPE(TOPIC) AUTHADD(PUB,SUB)

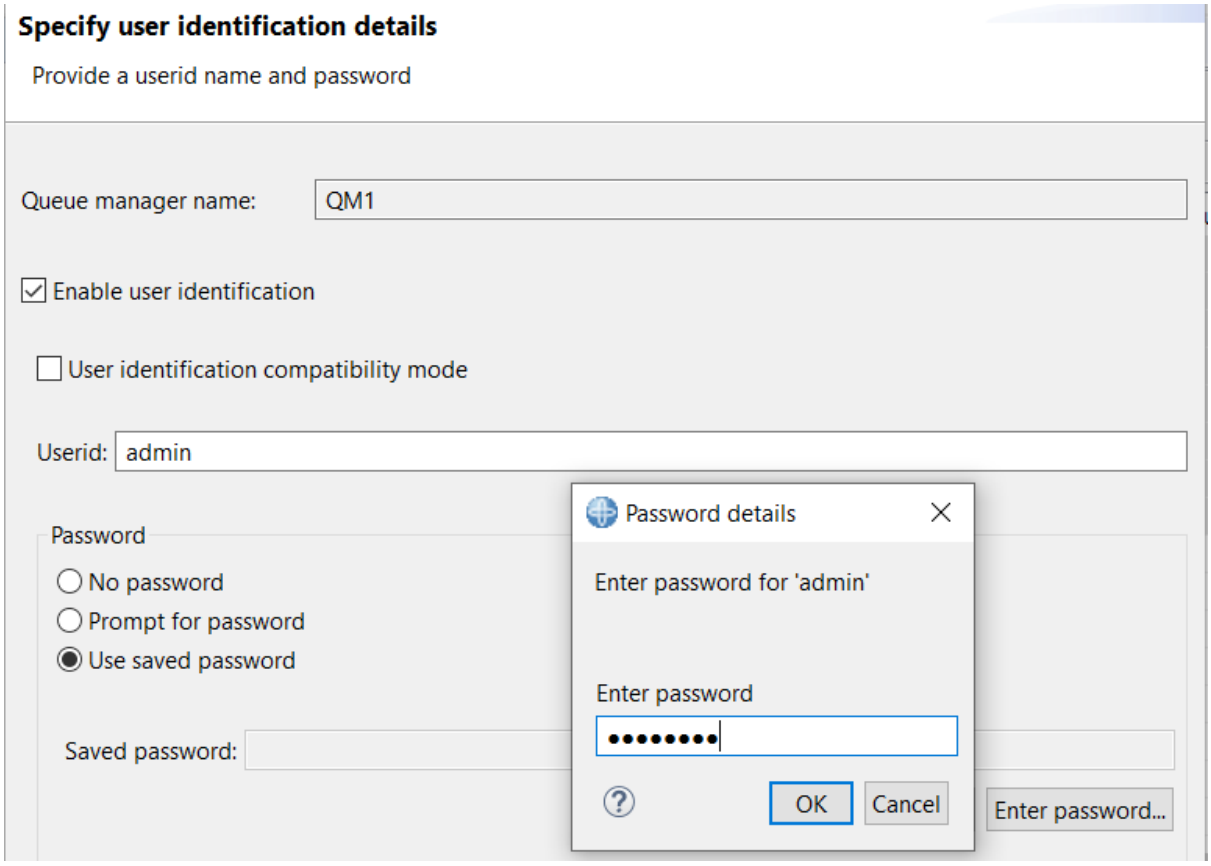
Access via MQ console

https://localhost:32795

Login in with admin/passw0rd

Access via MQ Explorer





Access via RUNMQSC client

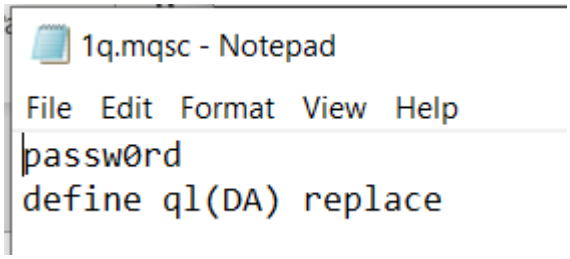
SET MQSERVER=IVT.SVRCONN/TCP/localhost(32796)

Runmqsc manually

```
C:\temp>runmqsc -c -u admin QM1
5724-H72 (C) Copyright IBM Corp. 1994, 2019.
Enter password:
********
Starting MQSC for queue manager QM1.


end
     1 : end
0 command responses received.
```

Runmqsc piping in a file

If -u is on the command and we are piping in, runmqsc assumes the first line is the password.




```
C:\temp>runmqsc -c -u admin QM1 < 1q.mqsc
5724-H72 (C) Copyright IBM Corp. 1994, 2019.
Enter password:
Starting MQSC for queue manager QM1.


     1 : define ql(DA) replace
AMQ8006I: IBM MQ queue created.
2 command responses received.
```