# CodeReady on Windows

## Install Codeready

[Chapter 2. Installation Red Hat CodeReady Containers 1.19 | Red Hat Customer Portal](https://access.redhat.com/documentation/en-us/red_hat_codeready_containers/1.19/html/getting_started_guide/installation_gsg#installing-codeready-containers_gsg)

You’ll need the pull image secret from your RedHat accouint

[Install OpenShift 4 | Red Hat OpenShift Cluster Manager | CRC](https://cloud.redhat.com/openshift/install/crc/installer-provisioned?intcmp=7013a000002CtetAAC)

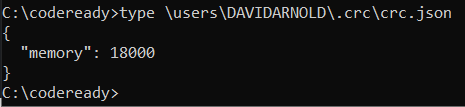
## Create Codeready image

Crc setup

Crc start -n 1.1.1.1 –disk-space 43

## Configure codeready

## 



## Start and log into codeready & create da-mq project

crc start -n 1.1.1.1 –disk-size 43

crc oc-env

@FOR /f "tokens=\*" %i IN ('crc oc-env') DO @call %i

crc console

oc login -u kubeadmin -p HqC3I-wgtiB-q7qCf-KEsuK <https://api.crc.testing:6443>

oc new project da-mq OR oc project da-mq

# MQ on CodeReady on Windows

## Build and deploy MQ

### Input Image stream

I have a copy of the MQ for Developers image on my docker repos but the actual image can be used.

kind: ImageStream

apiVersion: image.openshift.io/v1

metadata:

  name: ibm-mqadvanced-server-integration

  namespace: da-mq

spec:

  lookupPolicy:

    local: false

  tags:

    - name: 9.1.3.0-r4-amd64

      annotations: null

      from:

        kind: DockerImage

        name: 'davexacom/ibm-mqadvanced-server-integration:9.1.3.0-r4-amd64'

      generation: 2

      importPolicy: {}

      referencePolicy:

        type: Source

status:

  dockerImageRepository: >-

    image-registry.openshift-image-registry.svc:5000/da-mq/ibm-mqadvanced-server-integration

  publicDockerImageRepository: >-

    default-route-openshift-image-registry.apps-crc.testing/da-mq/ibm-mqadvanced-server-integration

  tags:

    - tag: 9.1.3.0-r4-amd64

      items:

        - created: '2021-01-08T06:00:40Z'

          dockerImageReference: >-

            davexacom/ibm-mqadvanced-server-integration@sha256:0cd8142e68ae76a84b06540fd93caf9f964045dbd35c6d0c7efb309e29f2bbb0

          image: >-

            sha256:0cd8142e68ae76a84b06540fd93caf9f964045dbd35c6d0c7efb309e29f2bbb0

          generation: 2

create the input file and use oc create to create it.

apiVersion: v1

kind: ImageStream

metadata:

name: ibm-mqadvanced-server-integration

spec:

tags:

- name: "9.1.3.0-r4-amd64"

from:

kind: DockerImage

name: davexacom/ibm-mqadvanced-server-integration:9.1.3.0-r4-amd64

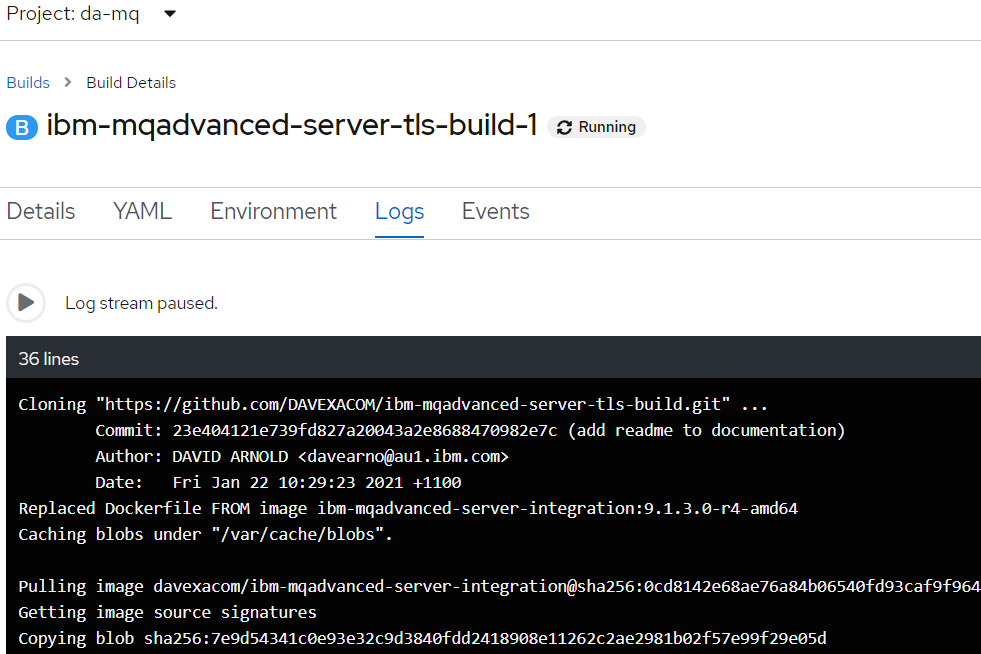
**oc create -f c:\openshift\data\build-mq-custom.yaml**

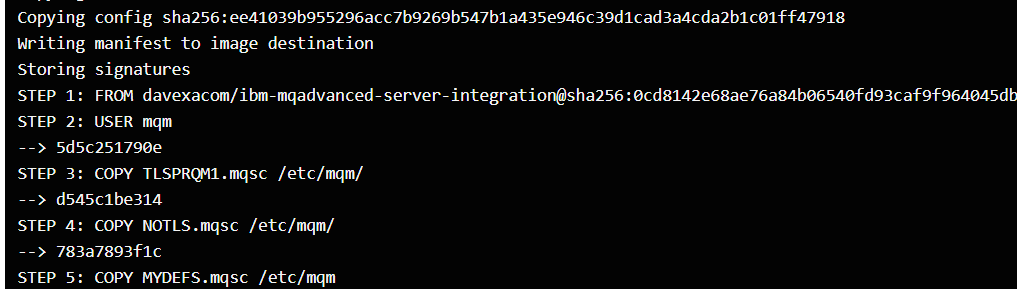
**oc adm policy add-scc-to-user anyuid -z default**

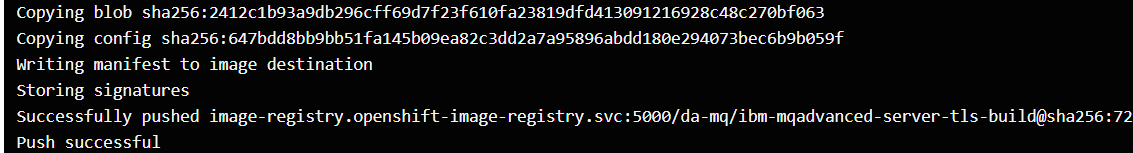
scc "anyuid" added to: ["system:serviceaccount:da-mq:default"]

### Build MQ Custom container via the image stream & github

**oc new-build** [**https://github.com/DAVEXACOM/ibm-mqadvanced-server-tls-build.git**](https://github.com/DAVEXACOM/ibm-mqadvanced-server-tls-build.git)



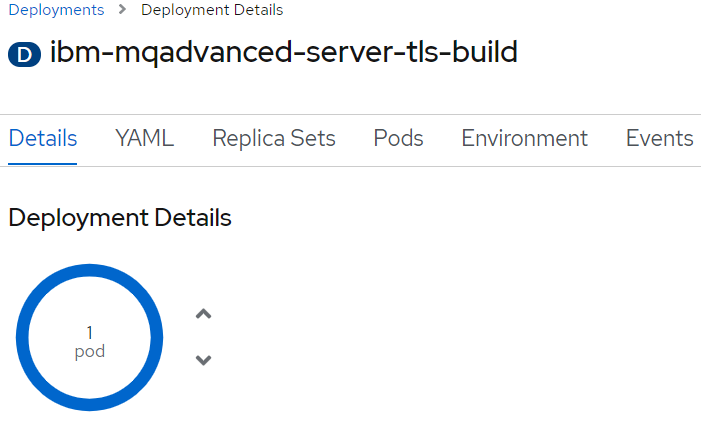


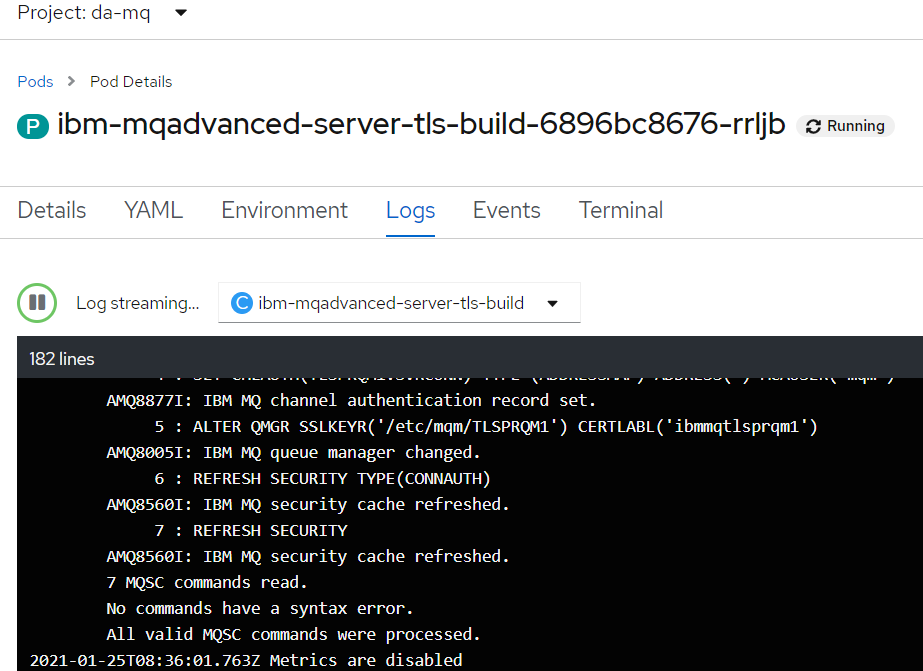


### Deploy the MQ custom container

Accept the license and set a queue manager name

oc new-app ibm-mqadvanced-server-tls-build --env LICENSE=accept --env MQ\_QMGR\_NAME=TLSPRQM1

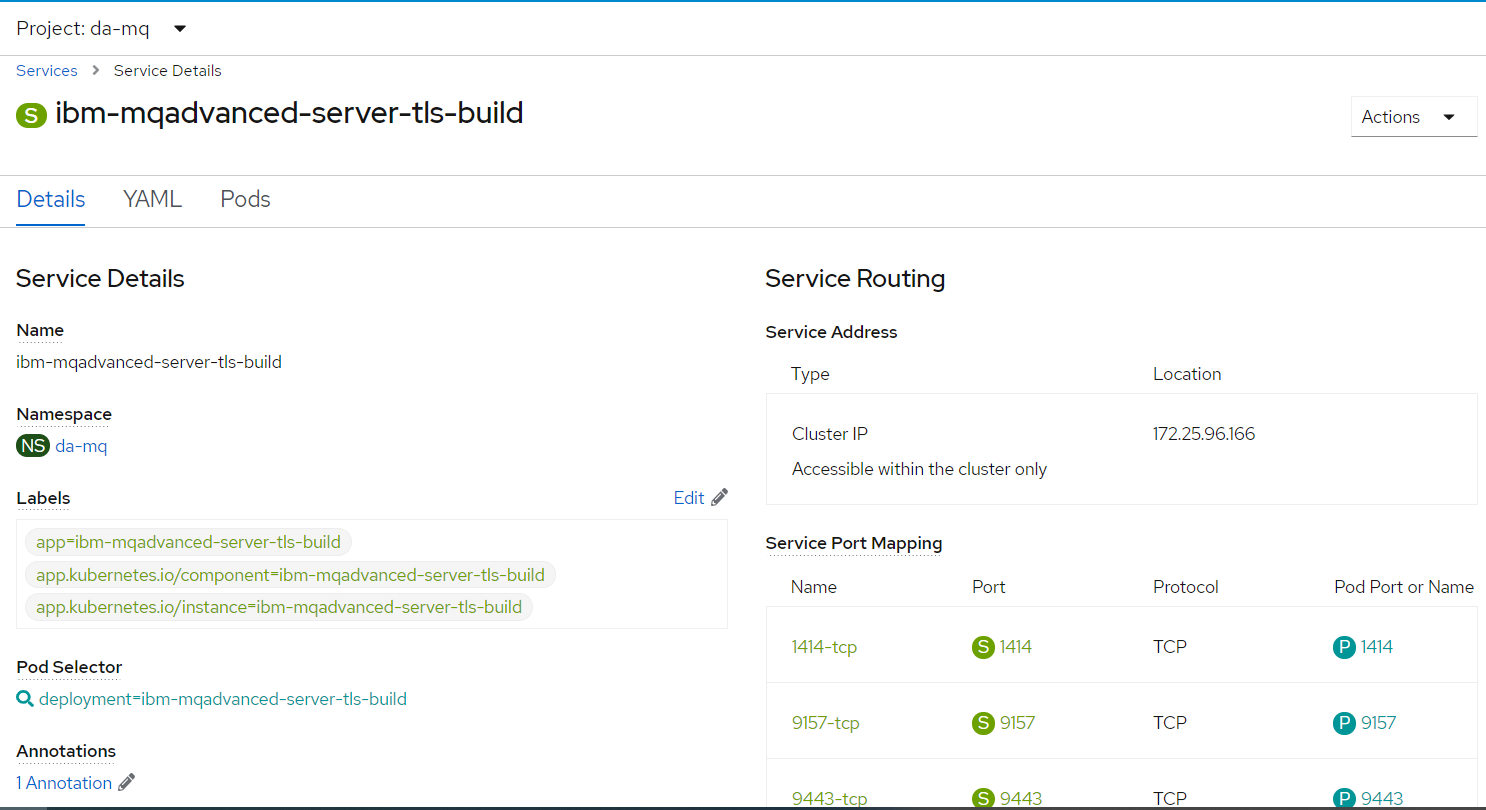




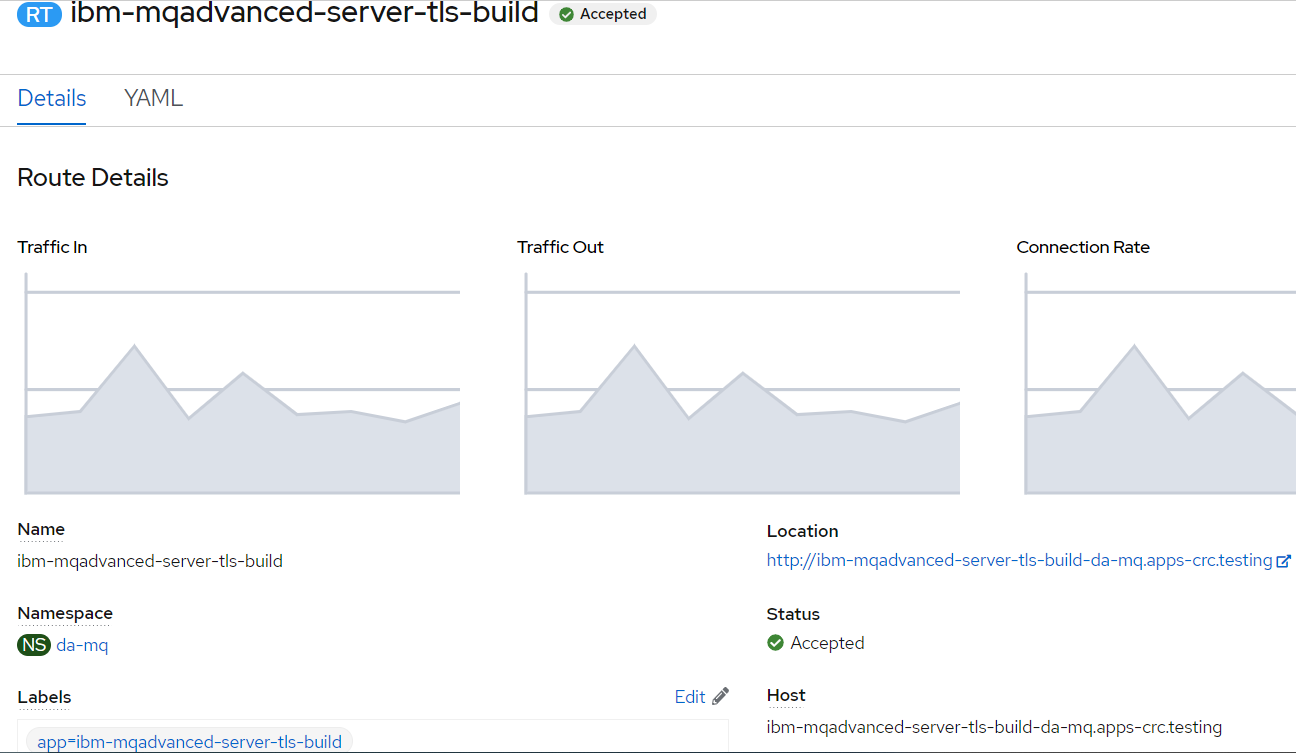
oc status

### Create/Expose the service as a route

This is the “base” route, we’ll need to create the TLS route as well that maps the SNI to the TLS channel.



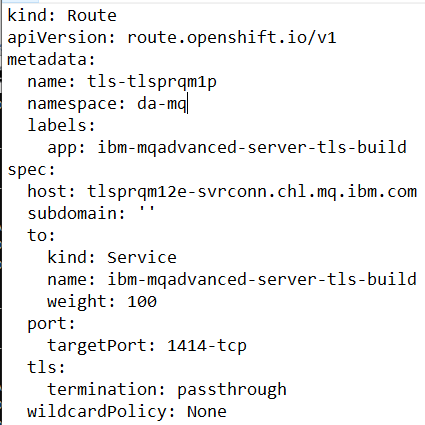
oc expose service/ibm-mqadvanced-server-tls-build



### Create the TLS route

This is the route that maps the SNI to the channel name

This host - tlsprqm12e-svrconn.chl.mq.ibm.com is the mapping for SVRCONN channel name TLSPRQM1.SVRCONN



kind: Route

apiVersion: route.openshift.io/v1

metadata:

name: tls-tlsprqm1p

namespace: da-mq

labels:

app: ibm-mqadvanced-server-tls-build

spec:

host: tlsprqm12e-svrconn.chl.mq.ibm.com

to:

kind: Service

name: ibm-mqadvanced-server-tls-build

weight: 100

port:

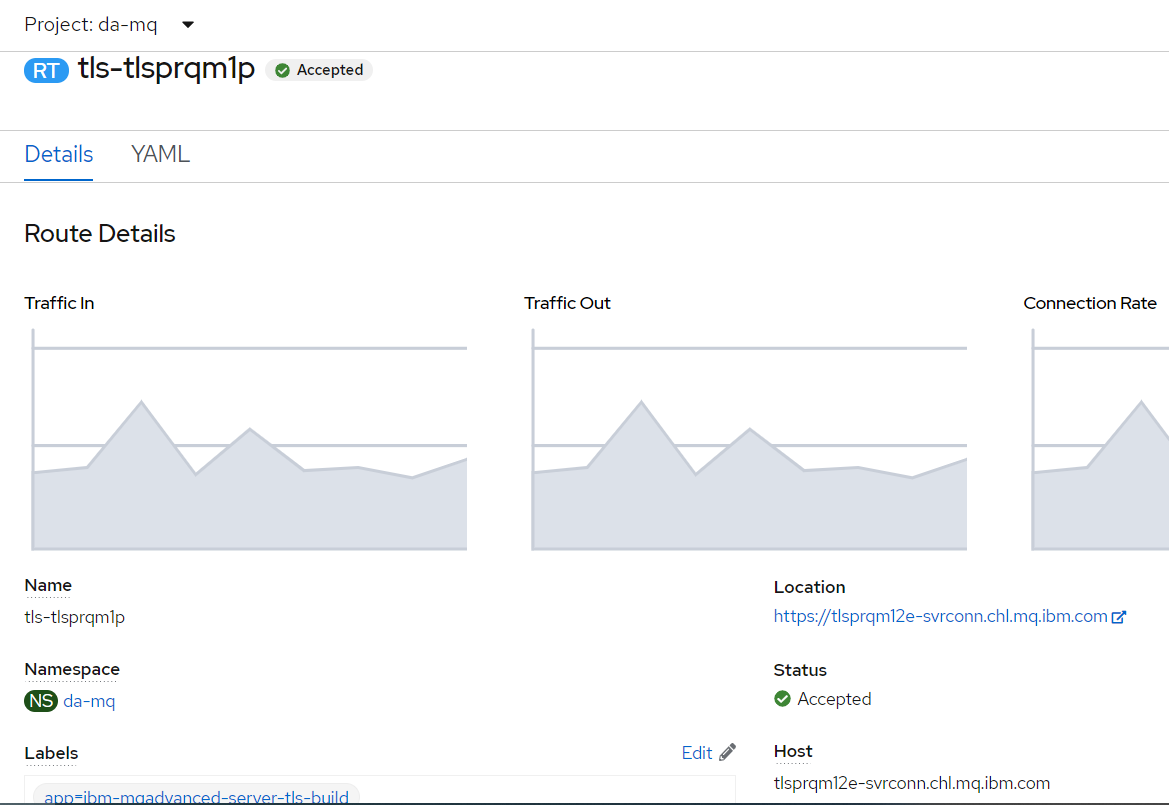
targetPort: 1414-tcp

tls:

termination: passthrough

wildcardPolicy: None

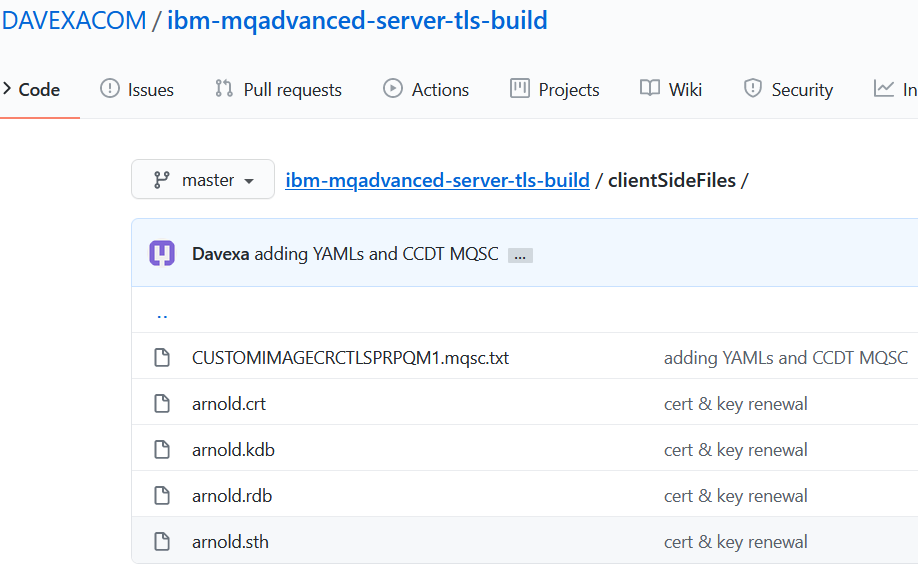
**oc create -f route-tls-tlsprqm1p.yaml**



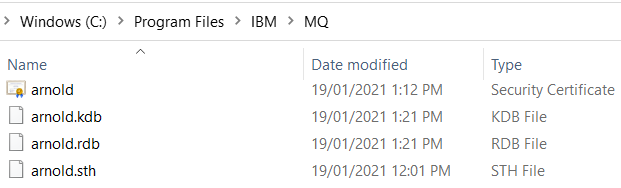
## Connect and test

### Client side keys and cert files

Download the following files from <https://github.com/DAVEXACOM/ibm-mqadvanced-server-tls-build/tree/master/ClientSideFiles>



Copy to (or linux equivalent) default location c:\Program Files\IBM\MQ



### Create Client Connection channel in the CCDT

### Edit the MQSC file

Download the MQSC file from <https://github.com/DAVEXACOM/ibm-mqadvanced-server-tls-build/blob/master/ClientSideFiles/CUSTOMIMAGECRCTLSPRQM1.mqsc>

You get the CONNAME from the “base” route yaml created by the oc expose command, so Get the hostname from the route generated by the oc expose service and update the mqsc file

|  |
| --- |
| DEFINE CHANNEL(TLSPRQM1.SVRCONN) + |
|  |

|  |
| --- |
| CHLTYPE(CLNTCONN) + |
|  |

|  |
| --- |
| TRPTYPE(TCP) + |
|  |

|  |
| --- |
| CONNAME('ibm-mqadvanced-server-tls-build-da-mq.apps-crc.testing(443)') + |
|  |

|  |
| --- |
| CERTLABL('ibmmqarnold') + |
|  |

|  |
| --- |
| QMNAME('TLSPRQM1') + |
|  |

|  |
| --- |
| SSLCIPH(ANY\_TLS12) + |
|  |

REPLACE

### Set up the Environment in the Command Line window

Start a command prompt as administrator

Clear the MQSERVER variable

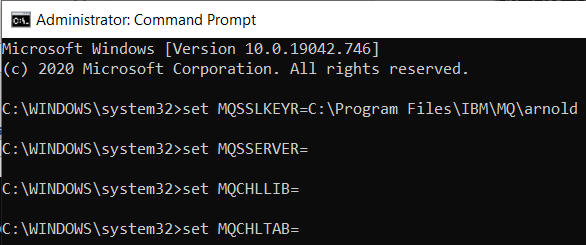
SET MQSERVER=

set MQSSLKEYR=C:\Program Files\IBM\MQ\arnold

Clear the MQCHLLIB and MQCHLTAB variables

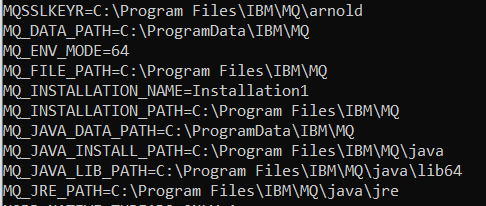
SET MQCHLLIB=

SET MQCHLTAB=



Check the MQClient.ini file as well to ensure there is no SSL stanza set.

Run SET on the command line and double check the MQ environment variables



### Runmqsc with -n flag to create CCDT

use runmqsc with -n flag and pipe in the MQSC file to create the MQ CCDT Table

DEFINE CHANNEL(TLSPRQM1.SVRCONN) +

CHLTYPE(CLNTCONN) +

TRPTYPE(TCP) +

CONNAME('ibm-mqadvanced-server-tls-build-da-mq.apps-crc.testing(443)') +

CERTLABL('ibmmqarnold') +

QMNAME('TLSPRQM1') +

SSLCIPH(ANY\_TLS12) +

REPLACE

runmqsc -n < CUSTOMIMAGECRCTLSPRPQM1.mqsc

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

Starting local MQSC for 'AMQCLCHL.TAB'.

1 : DEFINE CHANNEL(TLSPRQM1.SVRCONN) +

: CHLTYPE(CLNTCONN) +

: TRPTYPE(TCP) +

: CONNAME('ibm-mqadvanced-server-tls-build-da-mq.apps-crc.testing(443)') +

: CERTLABL('ibmmqarnold') +

: QMNAME('TLSPRQM1') +

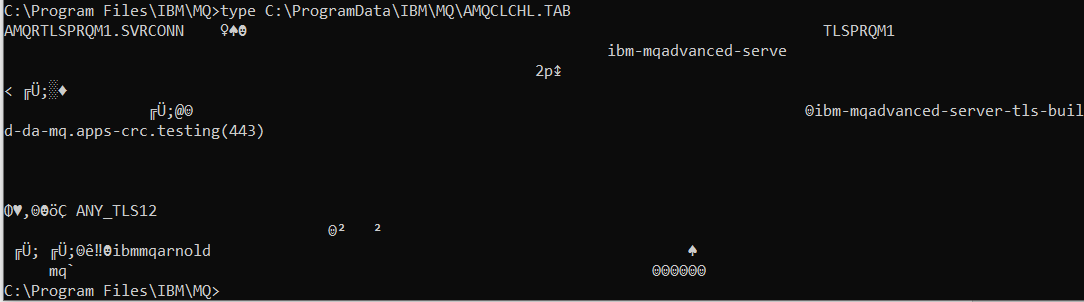
: SSLCIPH(ANY\_TLS12) +

: REPLACE

AMQ8014I: IBM MQ channel created.

No commands have a syntax error.

### Check the CCDT Table and copy to default location

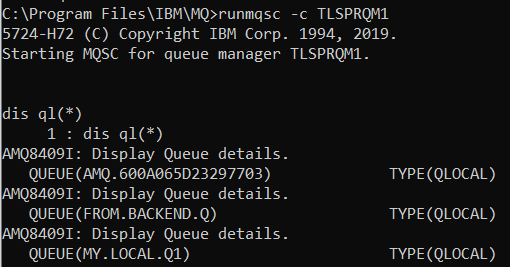
type C:\ProgramData\IBM\MQ\AMQCLCHL.TAB 

copy the AMQCLCHL.TAB file to the default location

C:\WINDOWS\system32>copy C:\ProgramData\IBM\MQ\AMQCLCHL.TAB "C:\Program Files\IBM\MQ\AMQCLCHL.TAB"

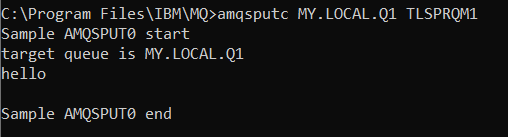
### RUNMQSC with the client connect flag -c

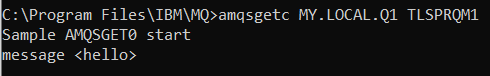
Runmqsc -c TLSPRQM1



End

## Test with AMQSPUTC & AMQSGETQ





# ACE on CodeReady on Windows

## Build and run ACE Microservice 2 from ACE SoE on Codeready

### Github/dockerhub

<https://github.com/DAVEXACOM/ibm-ace-mqc-soe-ms2-build>

<https://hub.docker.com/repository/docker/davexacom/ace11002mqc91soe>

Build from <https://hub.docker.com/repository/docker/davexacom/ace11002mqc91soe>

Using a version of <https://github.com/DAVEXACOM/ACEonICPIntMicSrv2Img>

Which is at <https://github.com/DAVEXACOM/ibm-ace-mqc-soe-ms2-build>

### Code for ACE Microservice2

[https://github.com/DAVEXACOM/ACEonICPIntMicSrv2](https://github.com/DAVEXACOM/ACEonICPIntMicSrv21)

### Code for ACE Liveliness probe

<https://github.com/DAVEXACOM/ACEonICPIntMicSoE>

## Start and prepare Codeready

C:\codeready>crc start -n 1.1.1.1 --disk-size 43

C:\codeready>crc oc-env

C:\codeready>@FOR /f "tokens=\*" %i IN ('crc oc-env') DO @call %i

C:\codeready>crc console

C:\codeready>oc login -u kubeadmin -p HqC3I-wgtiB-q7qCf-KEsuK <https://api.crc.testing:6443>

C:\codeready>oc new-project da-mq OR

C:\codeready>oc project da-mq OR

## Set the admin policy

**C:\codeready>oc adm policy add-scc-to-user anyuid -z default**

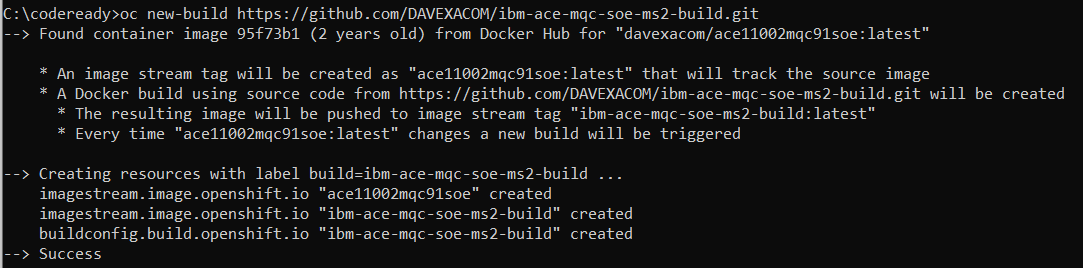
## Deploying to Codeready

@DA2@ I don’t think you need the oc create of the image stream for the input.

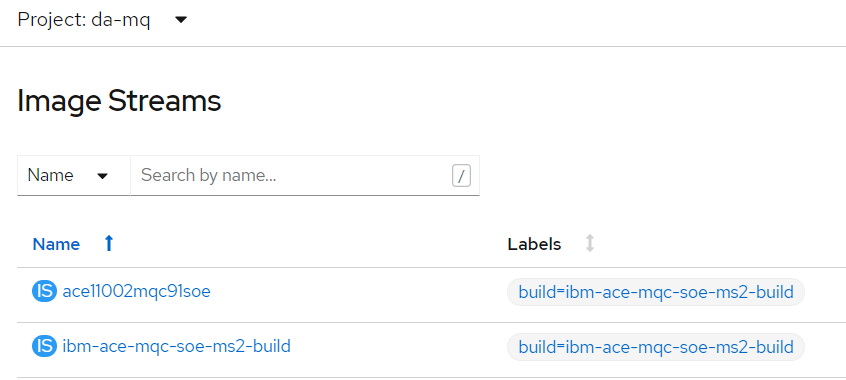
The oc new-build seem to take care of that via the FROM

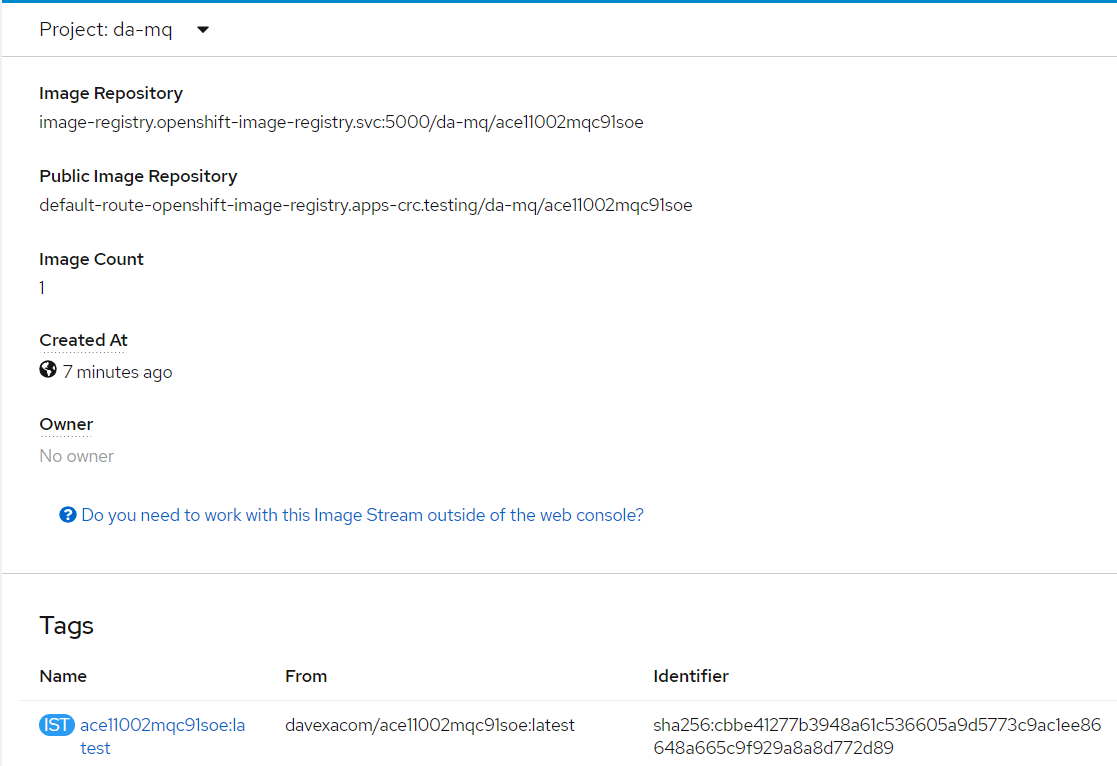
~~oc create -f c:\openshift\data\build-ace-custom.yaml~~

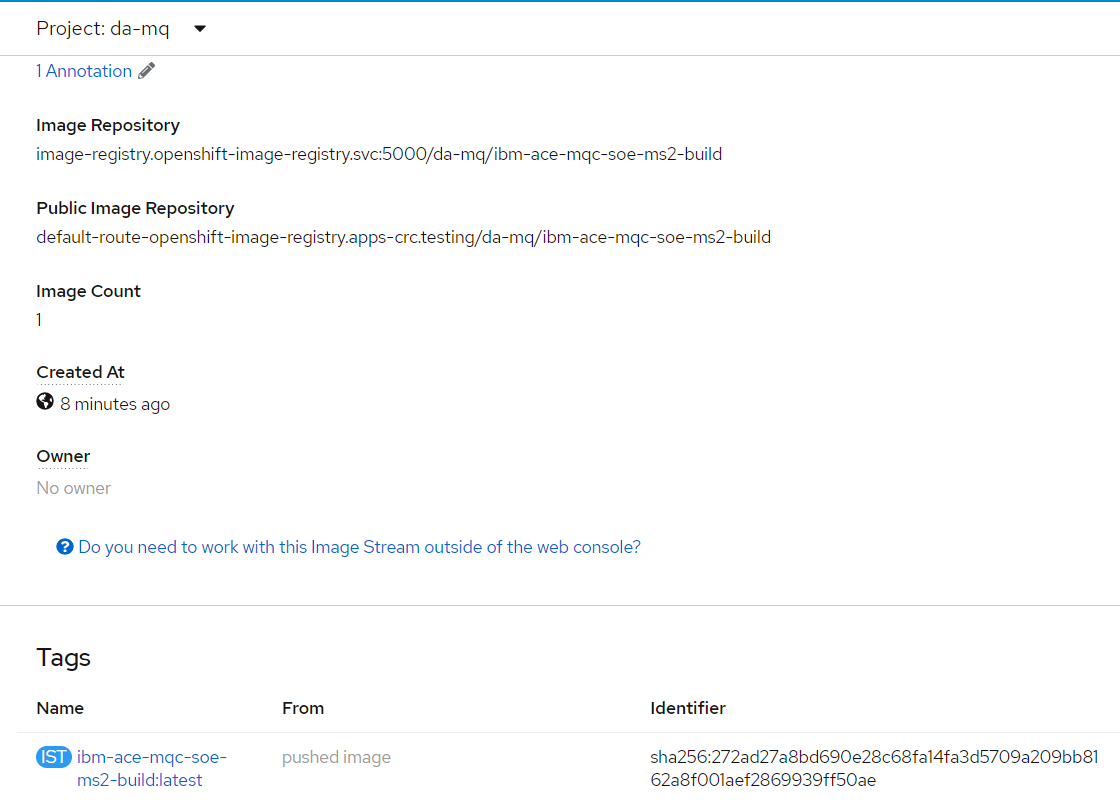
#### oc new-build <https://github.com/DAVEXACOM/ibm-ace-mqc-soe-ms2-build.git>



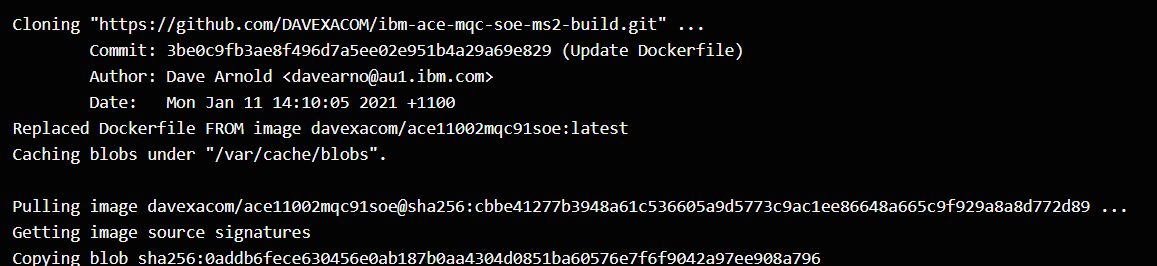
### Review image streams



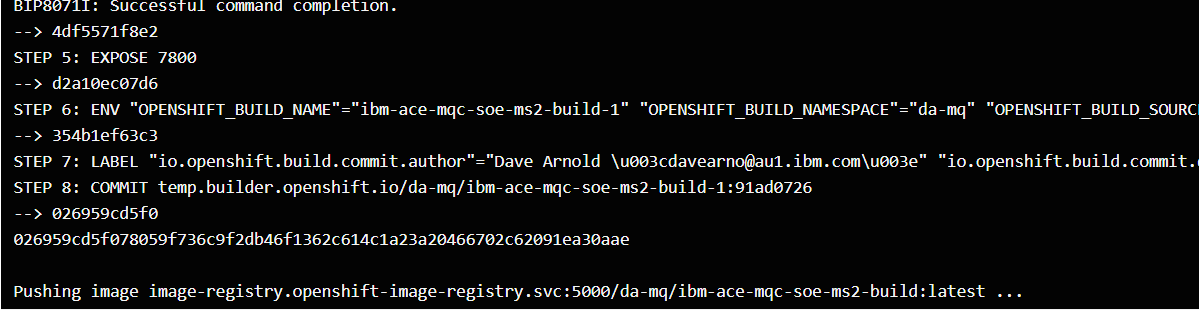




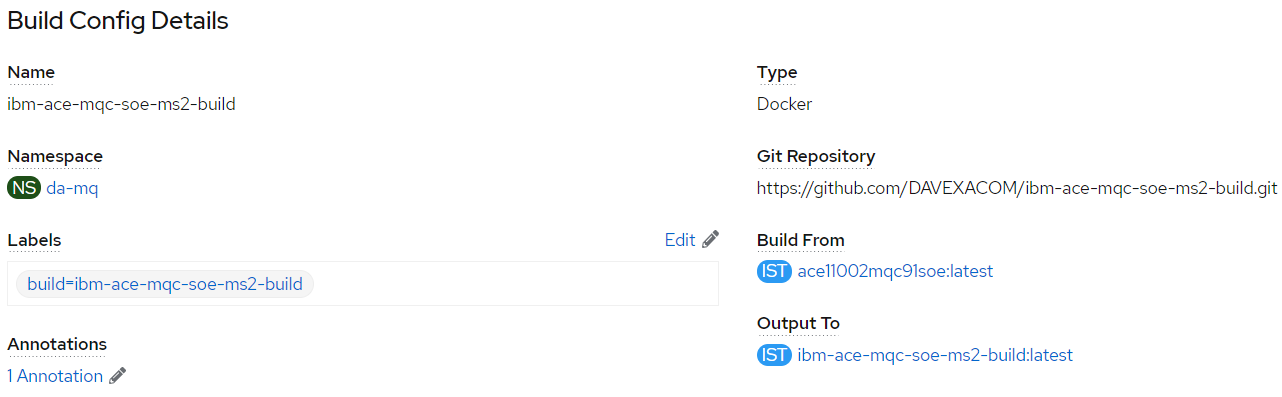
### Review build





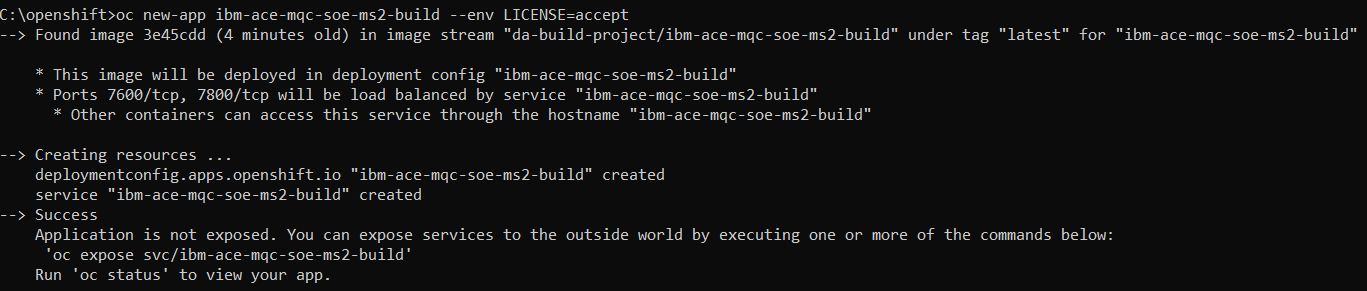


### Review build config

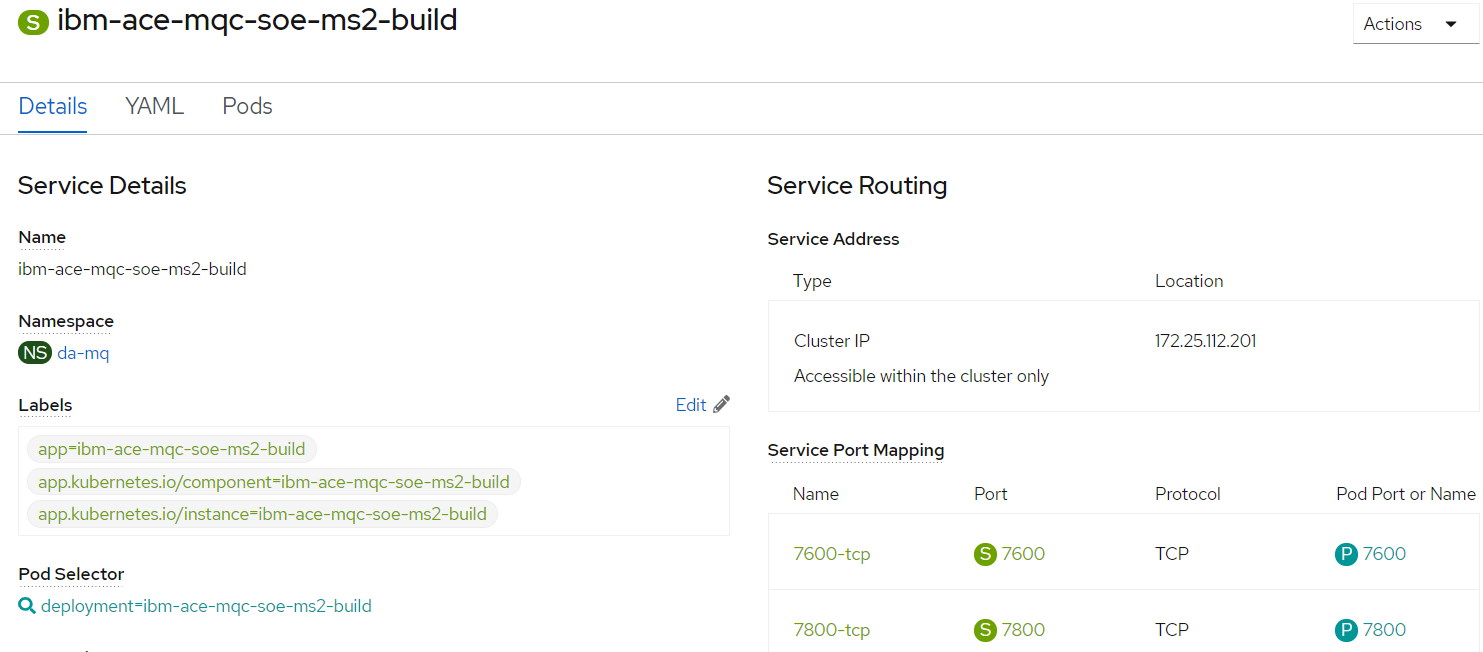


### Create app from build

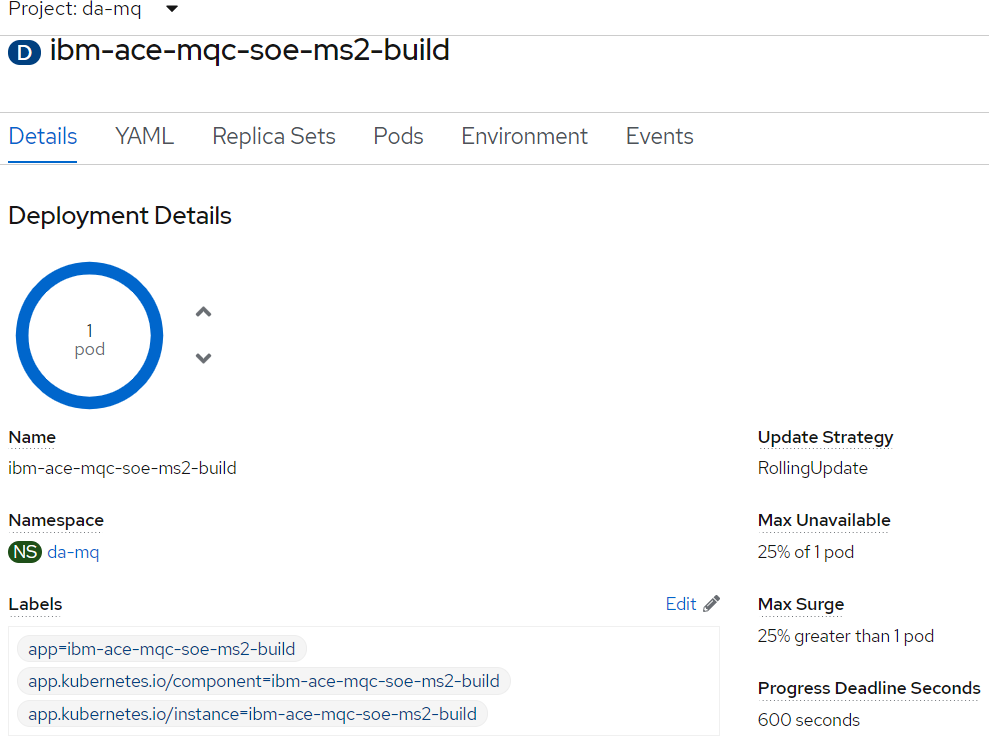
#### oc new-app ibm-ace-mqc-soe-ms2-build --env LICENSE=accept

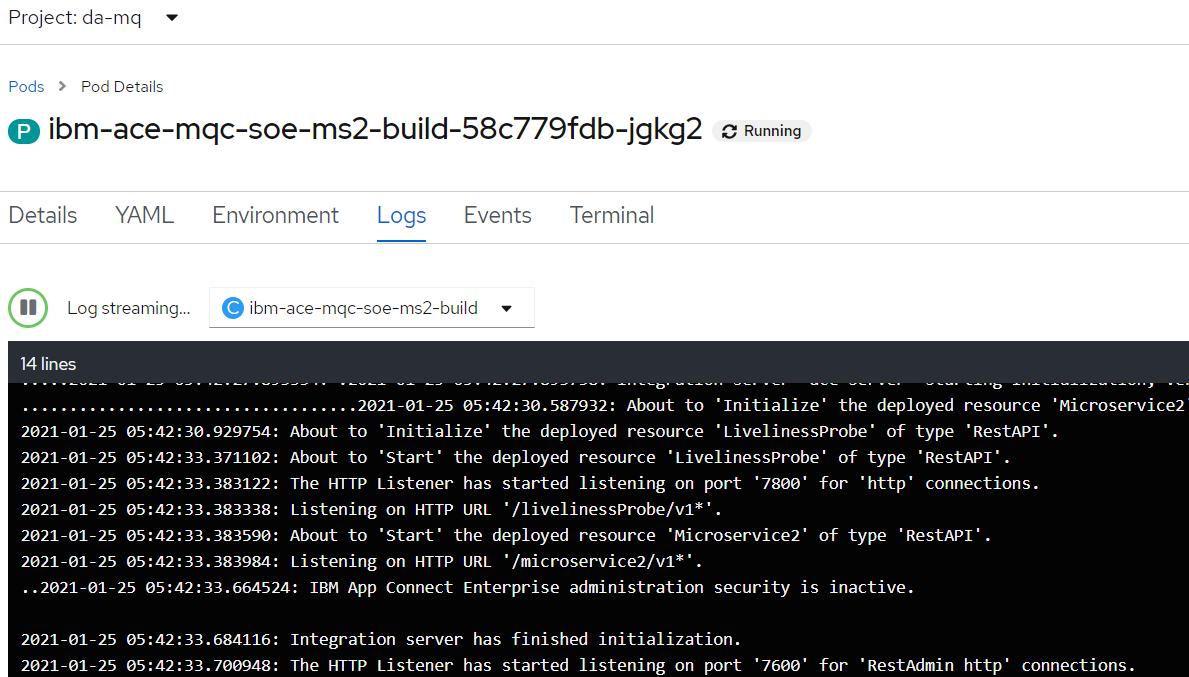


### Review service



### Review deployment



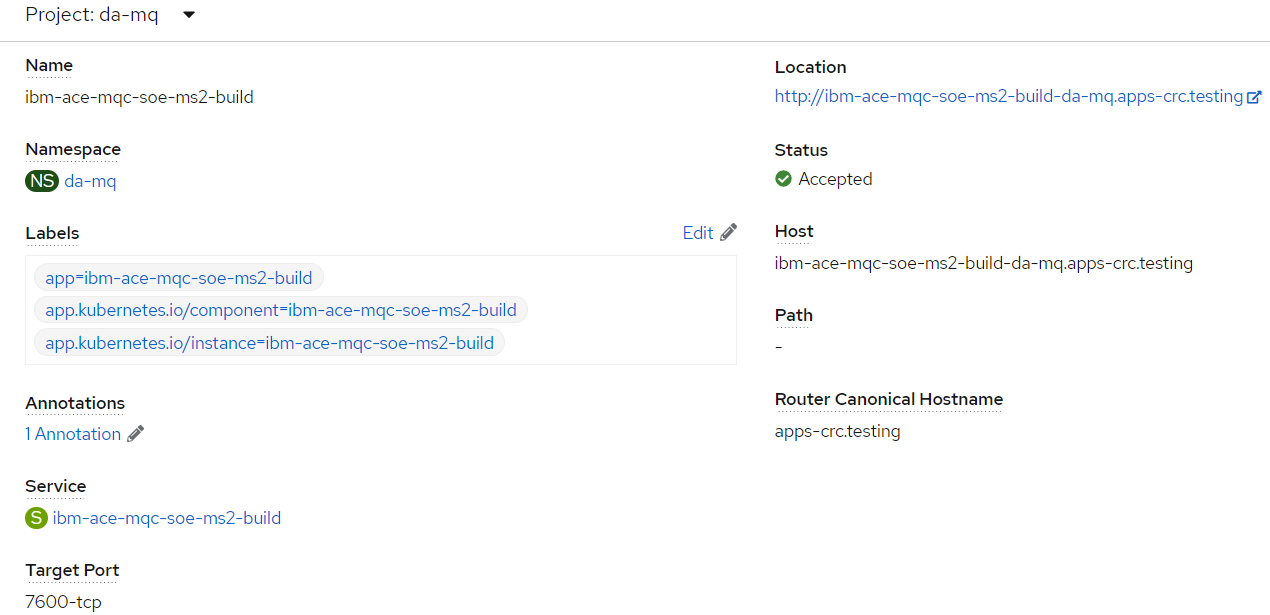


### Expose the service with a route

oc expose service/ibm-ace-mqc-soe-ms2-build

### Review the route

Port 7600 was exposed by default



Either edit the route to expose 7800 or create a second route.

### Create the route

kind: Route

apiVersion: route.openshift.io/v1

metadata:

name: ibm-ace-mqc-soe-ms2-build

namespace: da-mq

labels:

app: ibm-ace-mqc-soe-ms2-build

spec:

host: >-

ibm-ace-mqc-soe-ms2-build-da-mq.apps-crc.testing

subdomain: ''

to:

kind: Service

name: ibm-ace-mqc-soe-ms2-build

weight: 100

port:

targetPort: 7800-tcp

wildcardPolicy: None

#### oc create -f c:\openshift\data\create-acems2-route.yaml



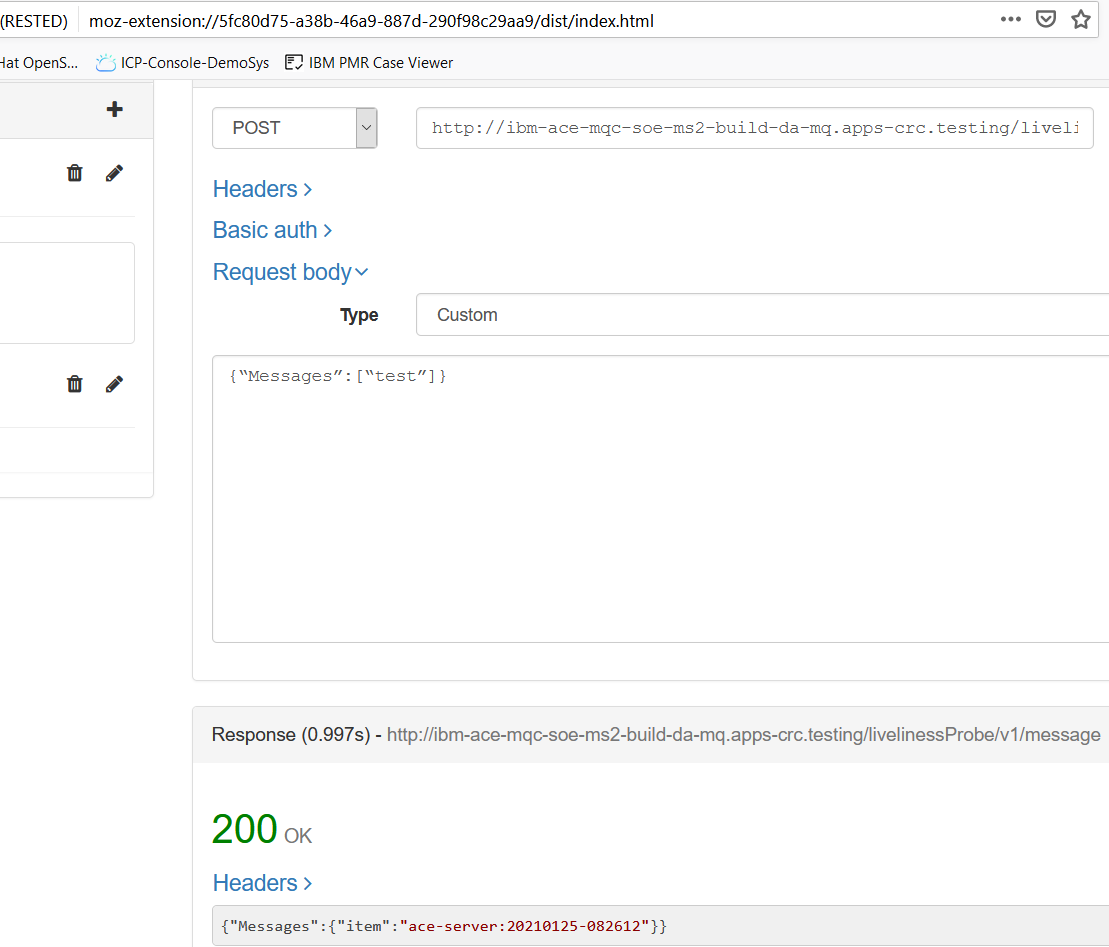
#### oc get routes

## Test the services

### Test the liveliness probe in the MS2 container

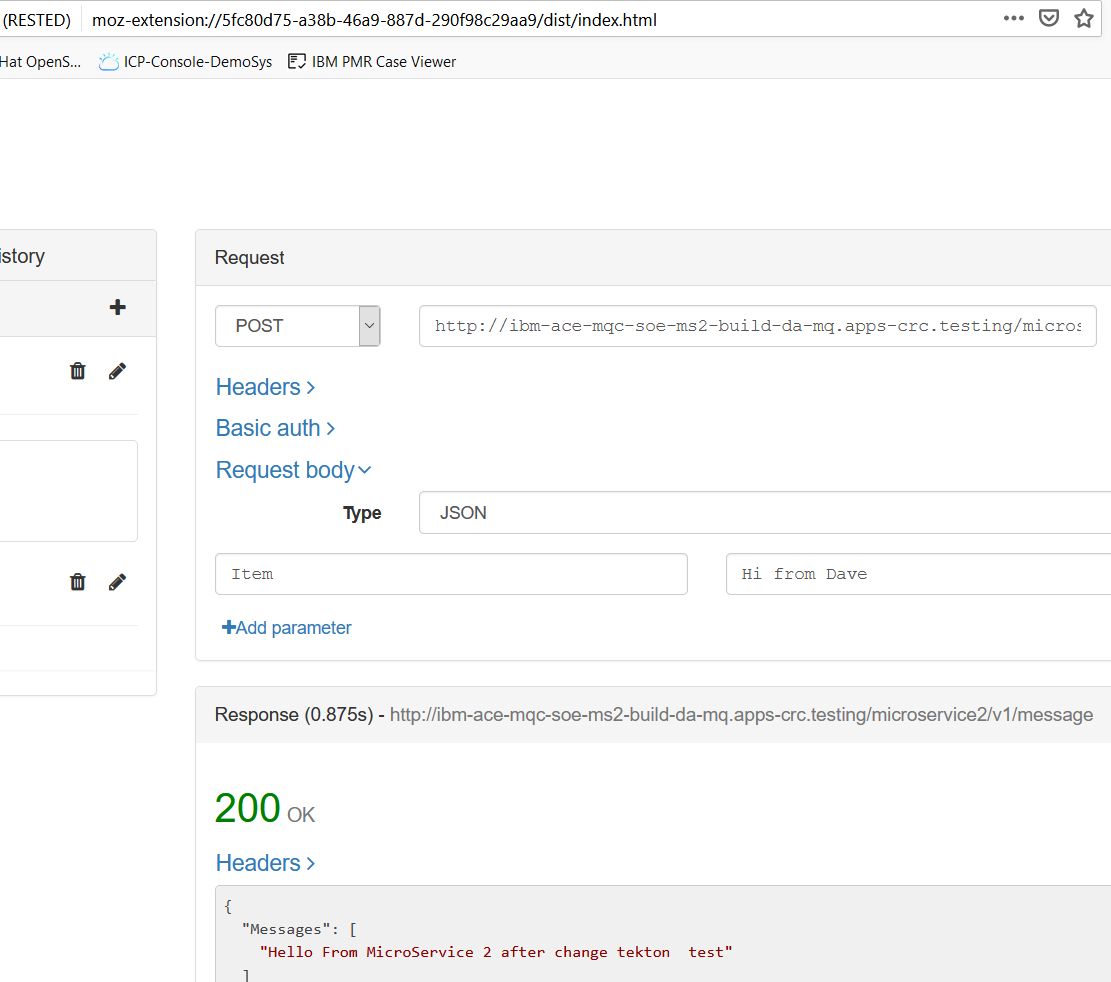
http:// ibm-ace-mqc-soe-ms2-build-da-mq.apps-crc.testing/livelinessProbe/v1/message

{“Messages”:[“test”]}



### Test microservice 2 in the MS2 container

### <http:// http:// ibm-ace-mqc-soe-ms2-build-da-mq.apps-crc.testing/microservice2/v1/message>



# Testing Ansible against Openshift in CRC on Windows

## Using raw

### Log into the CRC cluster

Davearno@DESKTOP-2MFRK17:~$ ansible localhost -m raw -a "oc login -u kubeadmin -p HqC3I-wgtiB-q7qCf-KEsuK https://api.crc.testing:6443 --insecure-skip-tls-verify=true"

### Change project

davearno@DESKTOP-2MFRK17:~$ ansible localhost -m raw -a "oc project da-mq"

### Get pods

davearno@DESKTOP-2MFRK17:~$ ansible localhost -m raw -a "oc get pods"

### describe pods

davearno@DESKTOP-2MFRK17:~$ ansible localhost -m raw -a "oc describe pod ibm-ace-mqc-soe-ms2-build-4-build"

### get deployments

davearno@DESKTOP-2MFRK17:~$ ansible localhost -m raw -a "oc get deployments"

### describe deployment

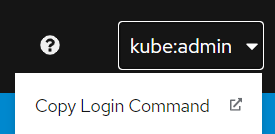
davearno@DESKTOP-2MFRK17:~$ ansible localhost -m raw -a "oc describe deployment ibm-ace-mqc-soe-ms2-build-4-build"

### scale deployment

davearno@DESKTOP-2MFRK17:~$ ansible localhost -m raw -a "oc scale deployment ace11002mqc91soe --replicas=3"

## K8s plug in

### Configuration – connection details





### Configuration – k8s.yml file

davearno@DESKTOP-2MFRK17:~$ tail k8s.yml

plugin: k8s

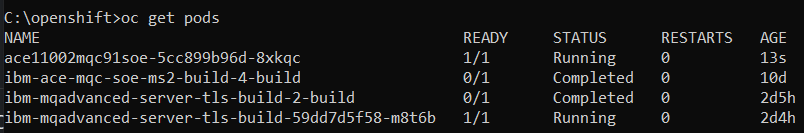
connections:

- host: https://api.crc.testing:6443

token: sha256~K9TnZbvgdmQaT3G1k9CJv6B0PHxx2oiKgi-9KD6X66c

validate\_certs: false

### Create ansible playbook – psefacepod.yml file



Nano psefacepod.yml

- hosts: localhost

tasks:

- set\_fact:

# this is \*just an example for brevity\*

# in reality you would use `k8s:` or `kubectl get -o name pods -l my-selector=my-value` to get the pod names

pod\_names:

- ibm-mqadvanced-server-tls-build-59dd7d5f58-m8t6b

- ace11002mqc91soe-5cc899b96d-8xkqc

- add\_host:

name: '{{ item }}'

groups:

- my-pods

with\_items: '{{ pod\_names }}'

- hosts: my-pods

gather\_facts: False

connection: kubectl

tasks:

# and now you are off to the races

# - command: ls

# watch out if the Pod doesn't have a working python installed

# as you will have to use raw: instead

# (and, of course, disable "gather\_facts: no")

- raw: ps -ef

### Running the playbook

davearno@DESKTOP-2MFRK17:~$ ansible-playbook psefacepod.yml

