Quick guide

RH OpenShift on IBM Cloud

And

Tekton Pipelines

Running up a minimal Openshift 4.3.5 beta on IBM Cloud and installing and testing Tekton.

Contents

[IBM Cloud – provision a Red Hat OpenShift on IBM Cloud service 4](#_Toc39046260)

[Accessing the Red Hat OpenShift service 7](#_Toc39046261)

[Installing Tekton 8](#_Toc39046262)

[C:\openshift>oc new-project tekton-pipelines 8](#_Toc39046263)

[C:\openshift>oc adm policy add-scc-to-user anyuid -z tekton-pipelines-controller 8](#_Toc39046264)

[C:\openshift>oc apply --filename https://storage.googleapis.com/tekton-releases/latest/release.yaml 8](#_Toc39046265)

[C:\openshift>oc get pods --namespace tekton-pipelines 9](#_Toc39046266)

[Correcting the paths to the image locations 10](#_Toc39046267)

[C:\openshift>oc get pods --namespace tekton-pipelines 12](#_Toc39046268)

[Test Tekton with an Echo example 12](#_Toc39046269)

[C:\openshift>oc apply -f Tekton-ivt\echo-hello-world-task.yaml 13](#_Toc39046270)

[C:\openshift>oc apply -f Tekton-ivt\echo-hello-world-task-run.yaml 13](#_Toc39046271)

[C:\openshift>oc get taskruns/echo-hello-world-task-run -o yaml 13](#_Toc39046272)

[OpenShift Console Pipelines 15](#_Toc39046273)

[Tekton Pipeline for IBM MQ (tls enabled) 17](#_Toc39046274)

[Create the Tekton artifacts in RH Openshift 17](#_Toc39046275)

[Updating the Tekton artifacts in RH Openshift 18](#_Toc39046276)

[Deleting the Tekton artifacts in RH Openshift 18](#_Toc39046277)

[Testing - Pipelinerun 19](#_Toc39046278)

[Jenkins 20](#_Toc39046279)

[Install the operator from the operator hub 20](#_Toc39046280)

[Create a Jenkins from the operator on your cluster 21](#_Toc39046281)

[Create the Jenkins Application Instance 22](#_Toc39046282)

[C:\openshift\oc new-app Jenkins 22](#_Toc39046283)

[C:\openshift\oc expose svc/jenkins 23](#_Toc39046284)

[Access the Jenkins Console 23](#_Toc39046285)

[Create Docker build configuration 25](#_Toc39046286)

[Create Jenkins Pipeline 28](#_Toc39046287)

[Explore the Jenkinsfile section of the mqtls-jen-pipeline BuildConfig; 30](#_Toc39046288)

[Stage 0: Declare Jenkins Agent 32](#_Toc39046289)

[Stage 1: Build 32](#_Toc39046290)

[Stage 2: Deploy 33](#_Toc39046291)

[Test the Pipeline 34](#_Toc39046292)

[Appendix A 36](#_Toc39046293)

[Changing to a later version of Tekton 36](#_Toc39046294)

[Tekton Controller 36](#_Toc39046295)

[Tekton Webhook 40](#_Toc39046296)

[NOTE: Leave the Webhook deployment alone i.e. v0.10.1 . I could not get later versions of the image to load. Instruction here just for completeness. 40](#_Toc39046297)

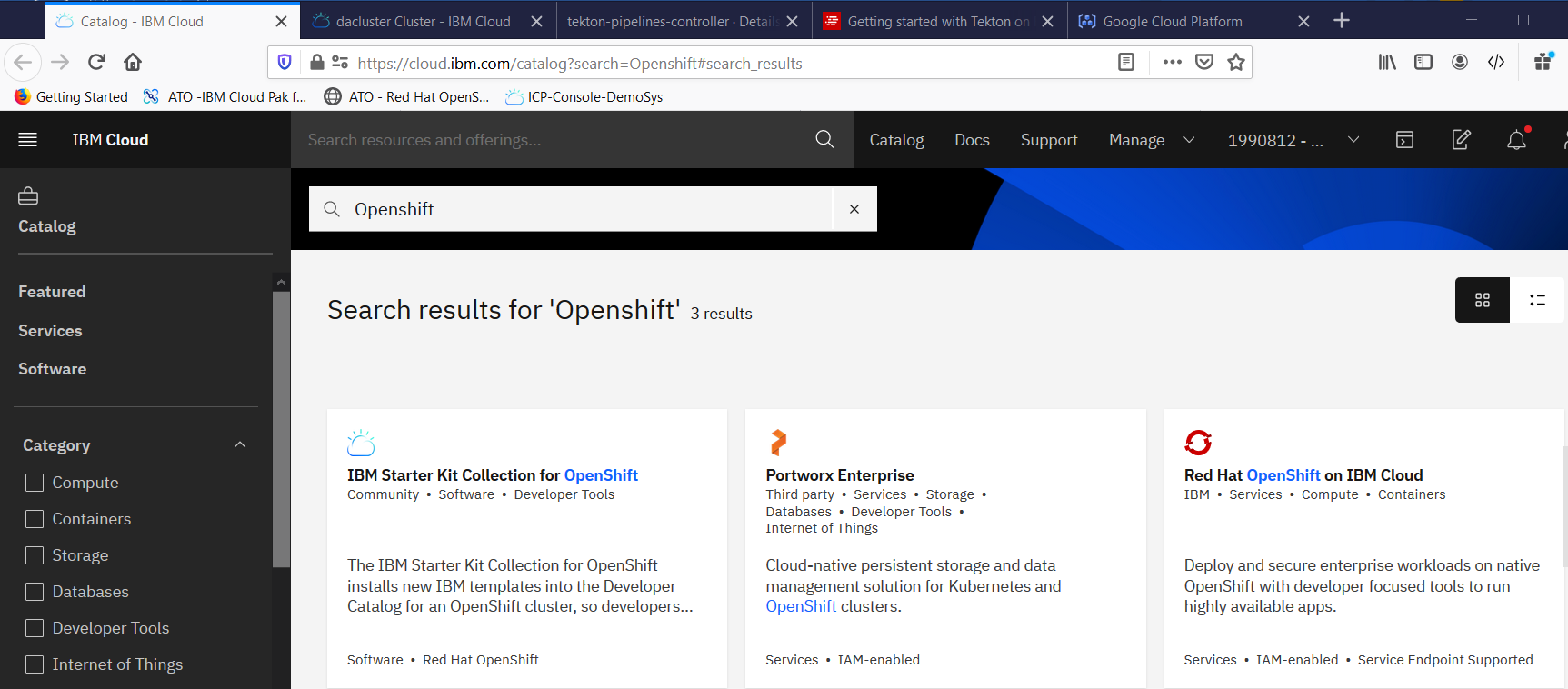
[Test the Tekton Update 40](#_Toc39046298)

## IBM Cloud – provision a Red Hat OpenShift on IBM Cloud service

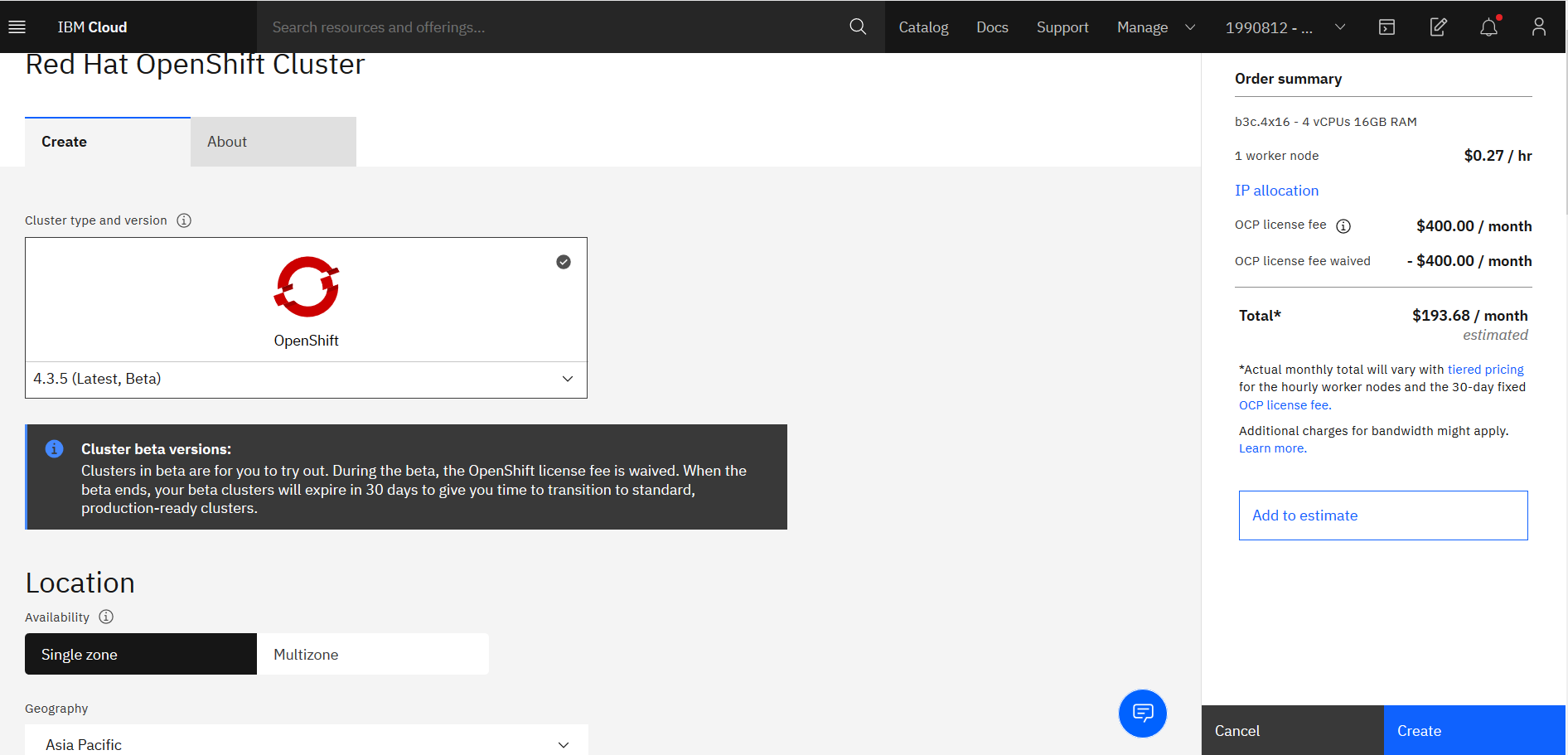
Log into IBM cloud

Goto the catalog and type openshift in the search

Select RedHat Openshift on IBM cloud



Select 4.3.5 Beta as there is no RHOCP license

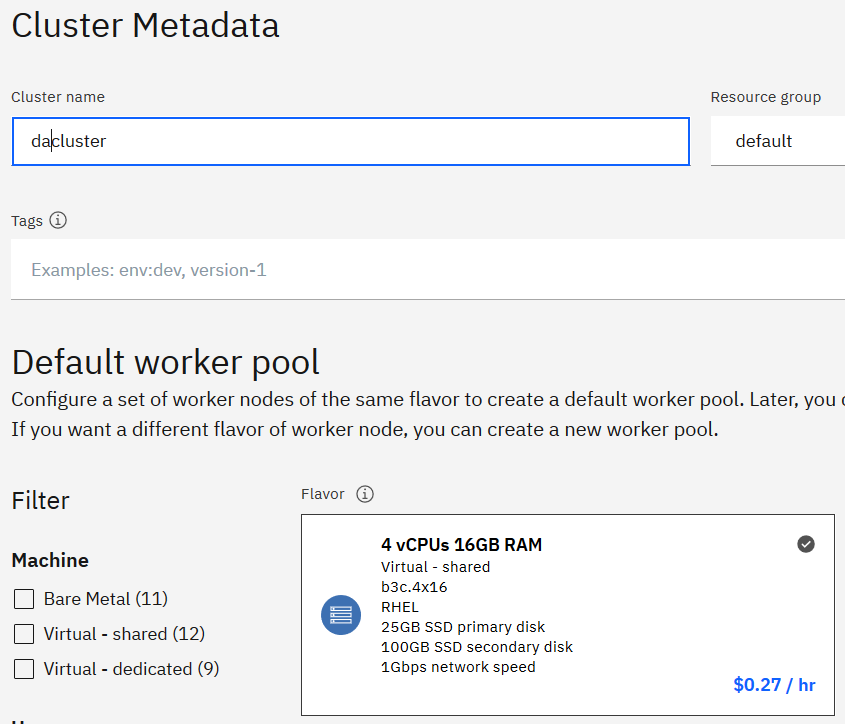


Switch to Single Zone

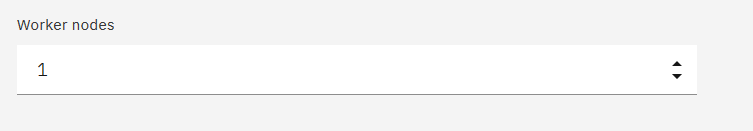
Asia Pacific

Sydney 01

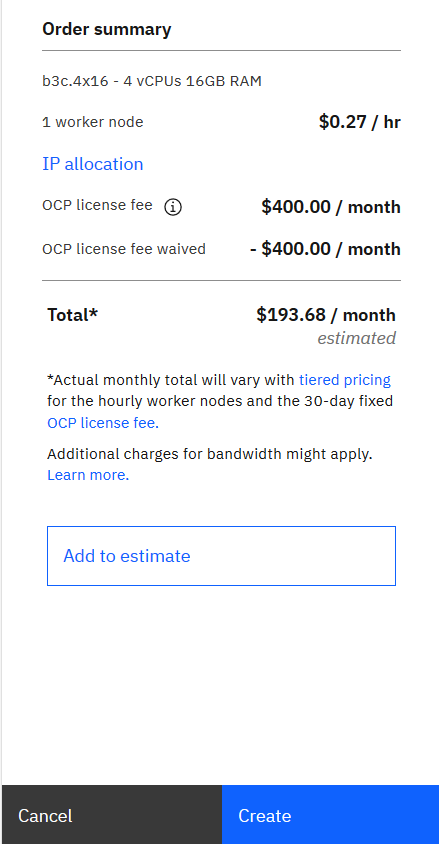
Name your cluster and select the small worker node size



Drop the worker node count from 3 to 1

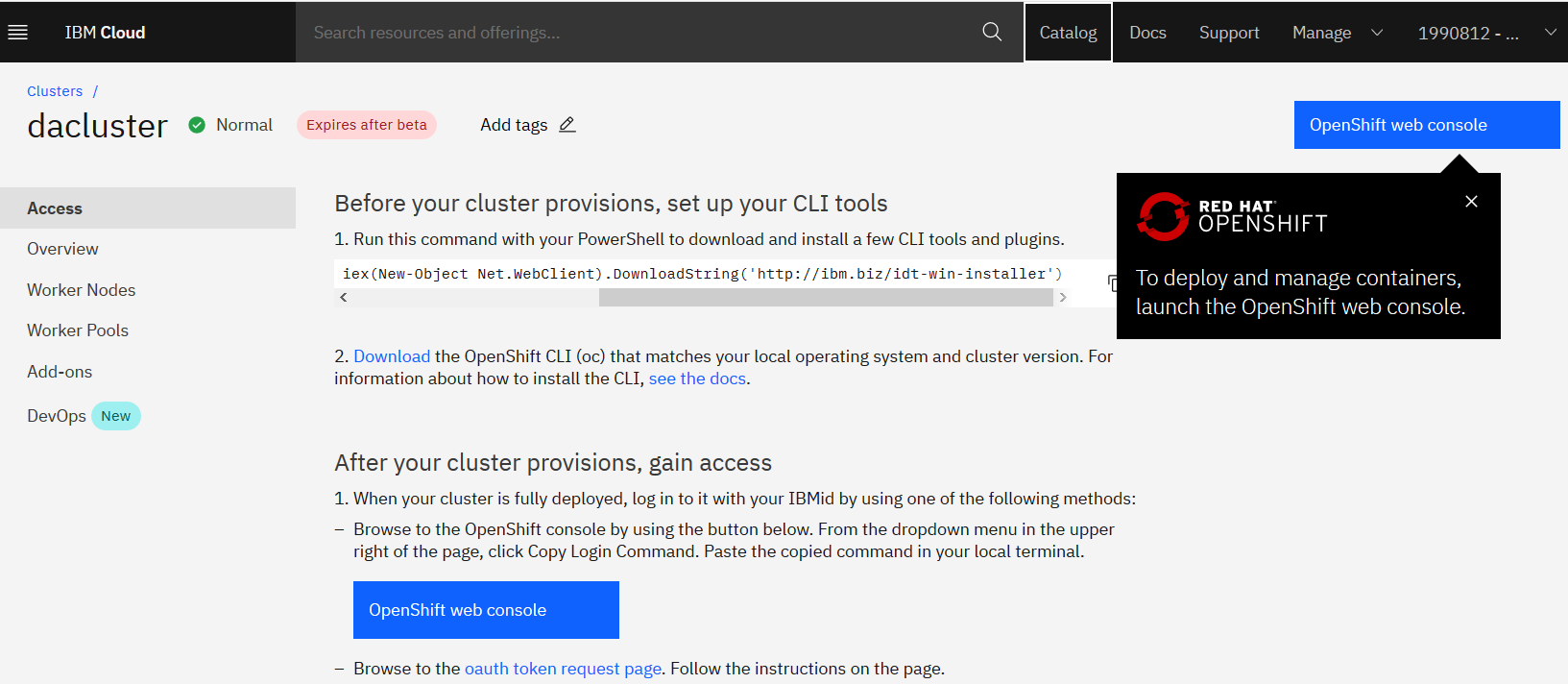


Price for 30 days is low as there is no Openshift license for the Beta version service. **This is point in time.**

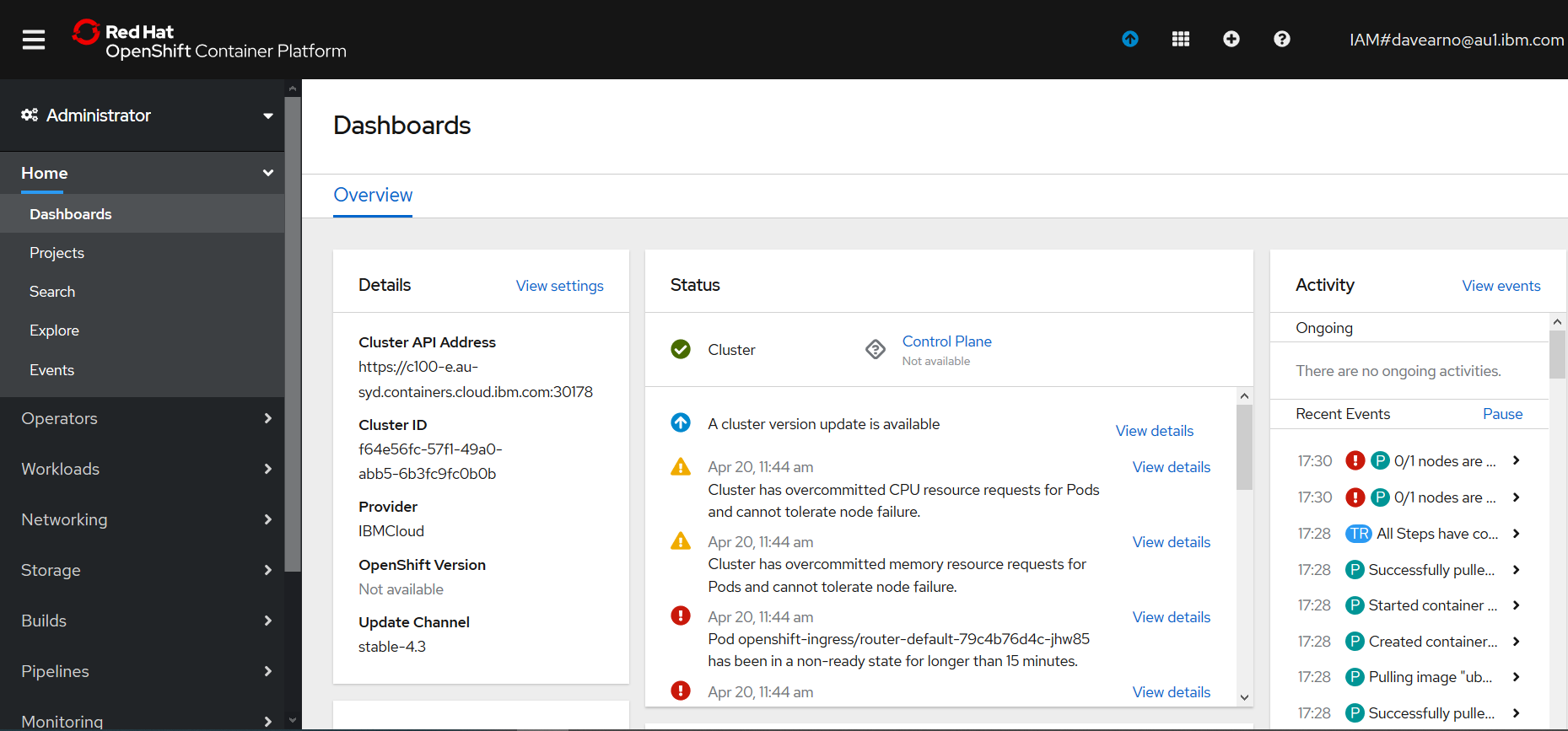


Click create and wait for the service to be provisioned

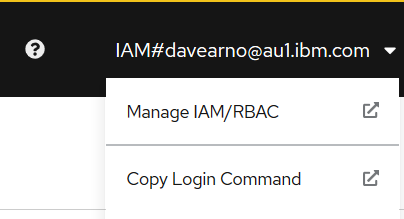
## Accessing the Red Hat OpenShift service

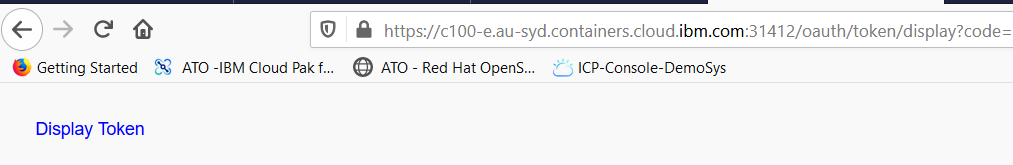


Open the web console



Log in via the command line



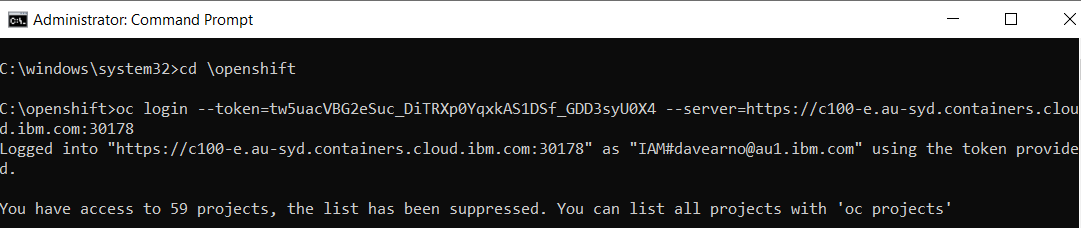


Click on display token



Copy the oc login command

(you need the RH Openshift client installed (the oc.exe))



## Installing Tekton

Follow the instructions in this article

https://developers.redhat.com/blog/2019/07/19/getting-started-with-tekton-on-red-hat-openshift/

### C:\openshift>oc new-project tekton-pipelines

Now using project "tekton-pipelines" on server "https://c100-e.au-syd.containers.cloud.ibm.com:30178".

You can add applications to this project with the 'new-app' command. For example, try:

oc new-app centos/ruby-25-centos7~https://github.com/sclorg/ruby-ex.git

to build a new example application in Ruby.

### C:\openshift>oc adm policy add-scc-to-user anyuid -z tekton-pipelines-controller

scc "anyuid" added to: ["system:serviceaccount:tekton-pipelines:tekton-pipelines-controller"]

### C:\openshift>oc apply --filename https://storage.googleapis.com/tekton-releases/latest/release.yaml

Warning: oc apply should be used on resource created by either oc create --save-config or oc apply

namespace/tekton-pipelines configured

podsecuritypolicy.policy/tekton-pipelines created

clusterrole.rbac.authorization.k8s.io/tekton-pipelines-admin created

serviceaccount/tekton-pipelines-controller created

clusterrolebinding.rbac.authorization.k8s.io/tekton-pipelines-controller-admin created

customresourcedefinition.apiextensions.k8s.io/clustertasks.tekton.dev created

customresourcedefinition.apiextensions.k8s.io/conditions.tekton.dev created

customresourcedefinition.apiextensions.k8s.io/images.caching.internal.knative.dev created

customresourcedefinition.apiextensions.k8s.io/pipelines.tekton.dev created

customresourcedefinition.apiextensions.k8s.io/pipelineruns.tekton.dev created

customresourcedefinition.apiextensions.k8s.io/pipelineresources.tekton.dev created

customresourcedefinition.apiextensions.k8s.io/tasks.tekton.dev created

customresourcedefinition.apiextensions.k8s.io/taskruns.tekton.dev created

service/tekton-pipelines-controller created

service/tekton-pipelines-webhook created

clusterrole.rbac.authorization.k8s.io/tekton-aggregate-edit created

clusterrole.rbac.authorization.k8s.io/tekton-aggregate-view created

configmap/config-artifact-bucket created

configmap/config-artifact-pvc created

configmap/config-defaults created

configmap/config-logging created

configmap/config-observability created

deployment.apps/tekton-pipelines-controller created

deployment.apps/tekton-pipelines-webhook created

### C:\openshift>oc get pods --namespace tekton-pipelines

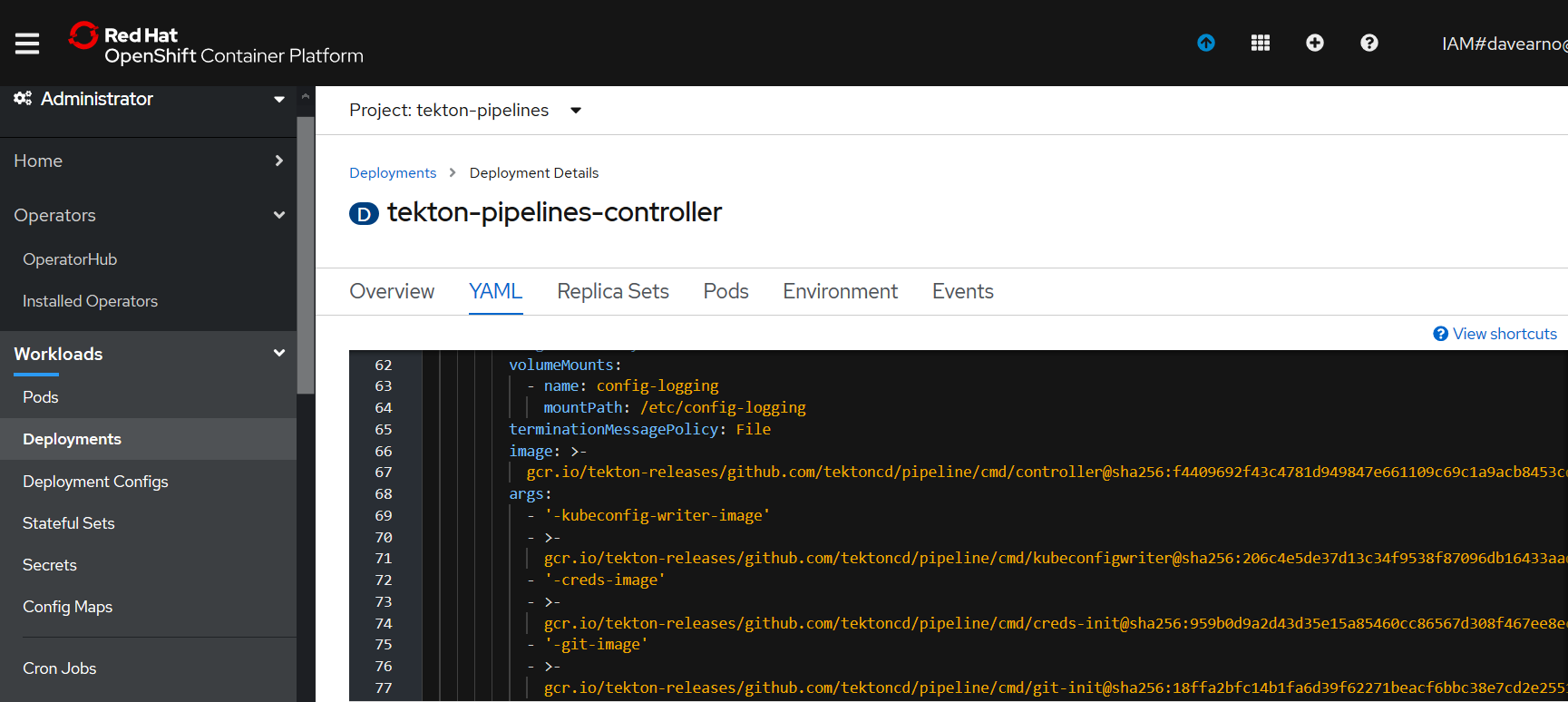
NAME READY STATUS RESTARTS AGE

tekton-pipelines-controller-5b75cdfb95-25gfg 0/1 ImagePullBackOff 0 11m

tekton-pipelines-webhook-b848dcd97-bdhnz 0/1 ImagePullBackOff 0 11m

### Correcting the paths to the image locations

Navigate to the RH Openshift Console->Workloads->Deploymenets->tekton-pipelines-controller



Edit the YAML

The deployment YAML seems to have a mismatch for the actual location of the images on google cloud. The YAML includes **:v0.10.1** this need to be removed as per the example below.

<https://console.cloud.google.com/gcr/images/tekton-releases/GLOBAL/github.com/tektoncd/pipeline/cmd>

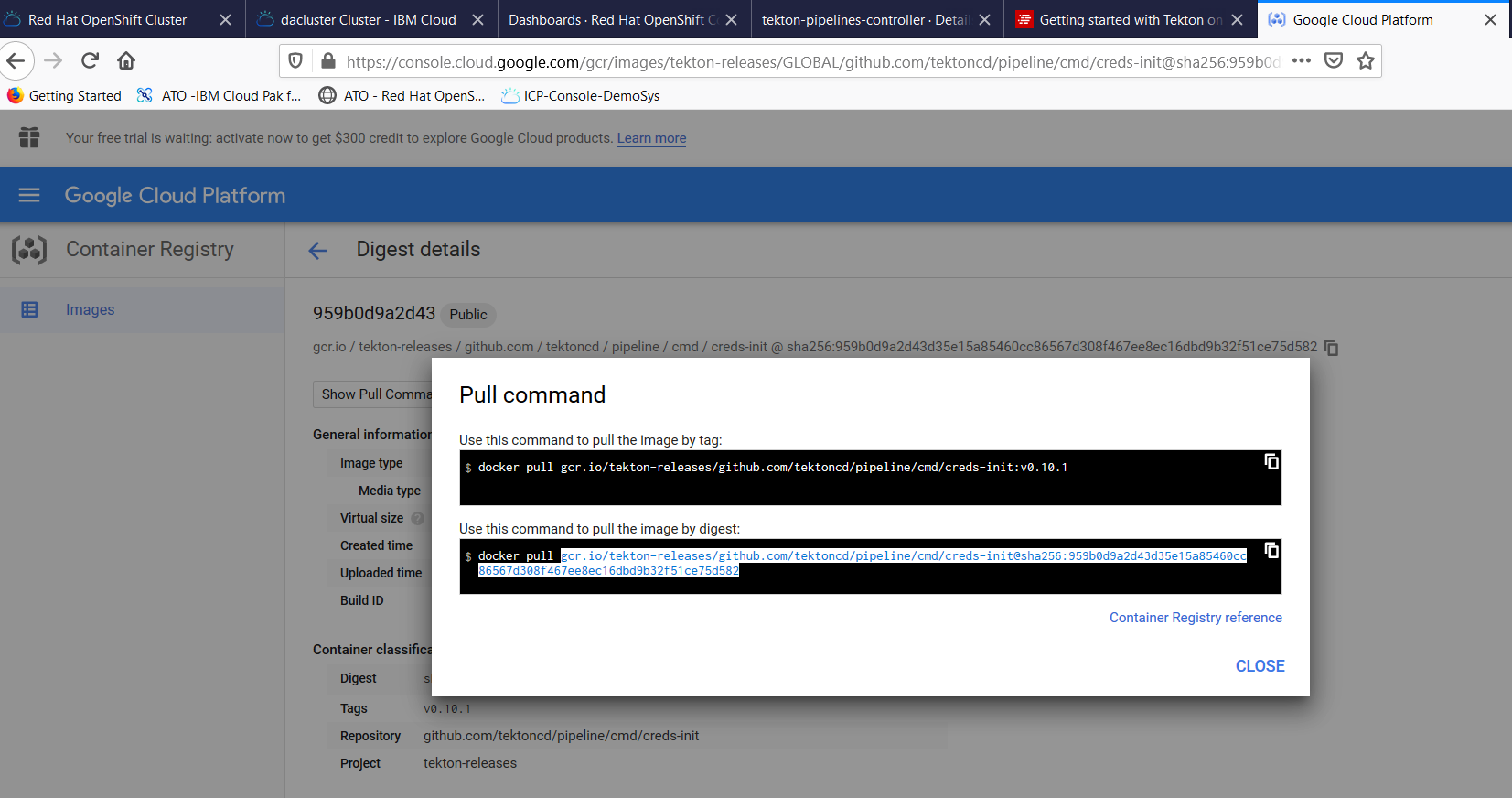
The YAML:

gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/creds-init**:v0.10.1**@sha256:959b0d9a2d43d35e15a85460cc86567d308f467ee8ec16dbd9b32f51ce75d582

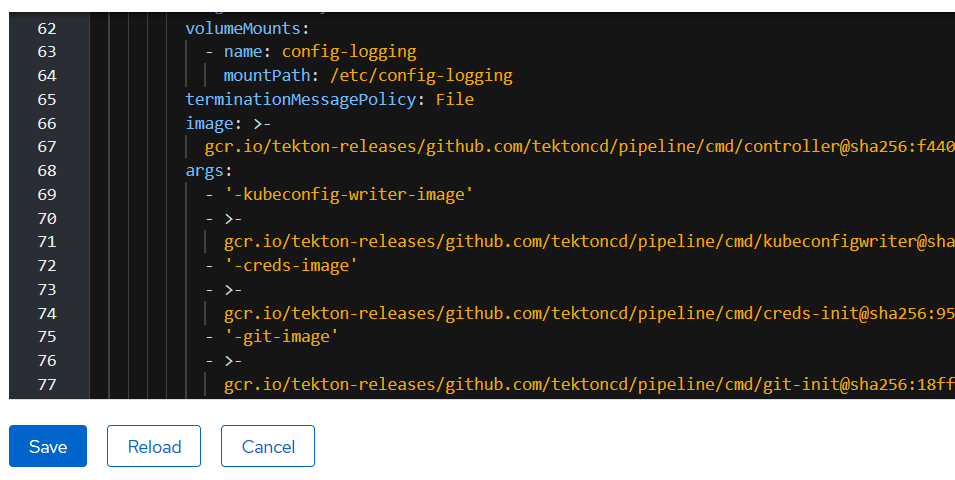
The location of the images on google:

[gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/creds-init@sha256:959b0d9a2d43d35e15a85460cc86567d308f467ee8ec16dbd9b32f51ce75d582](mailto:gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/creds-init@sha256:959b0d9a2d43d35e15a85460cc86567d308f467ee8ec16dbd9b32f51ce75d582)

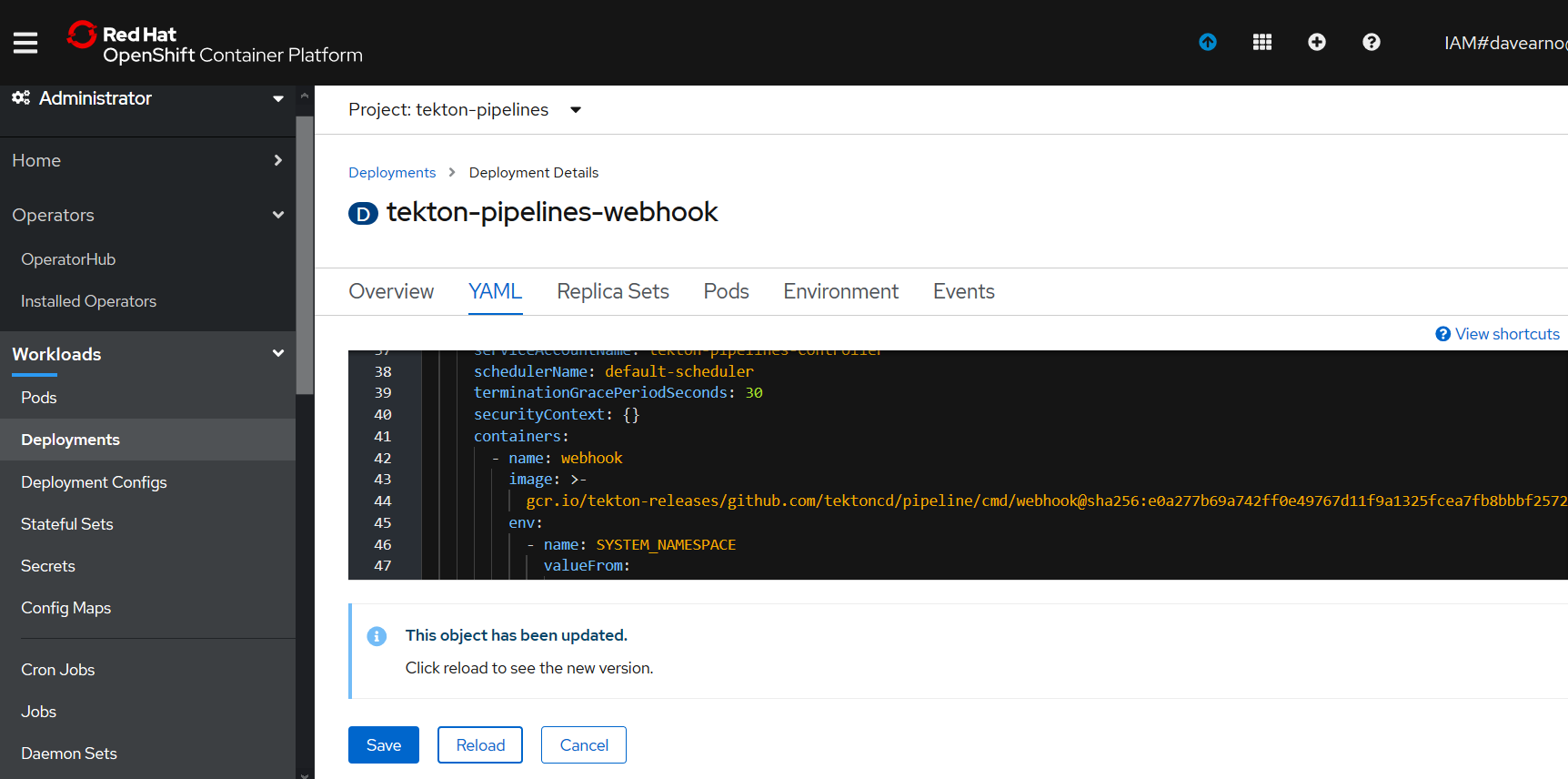
you can navigate to the image and copy and paste from the pull command.



Correct the YAML and hit Save and Reload.

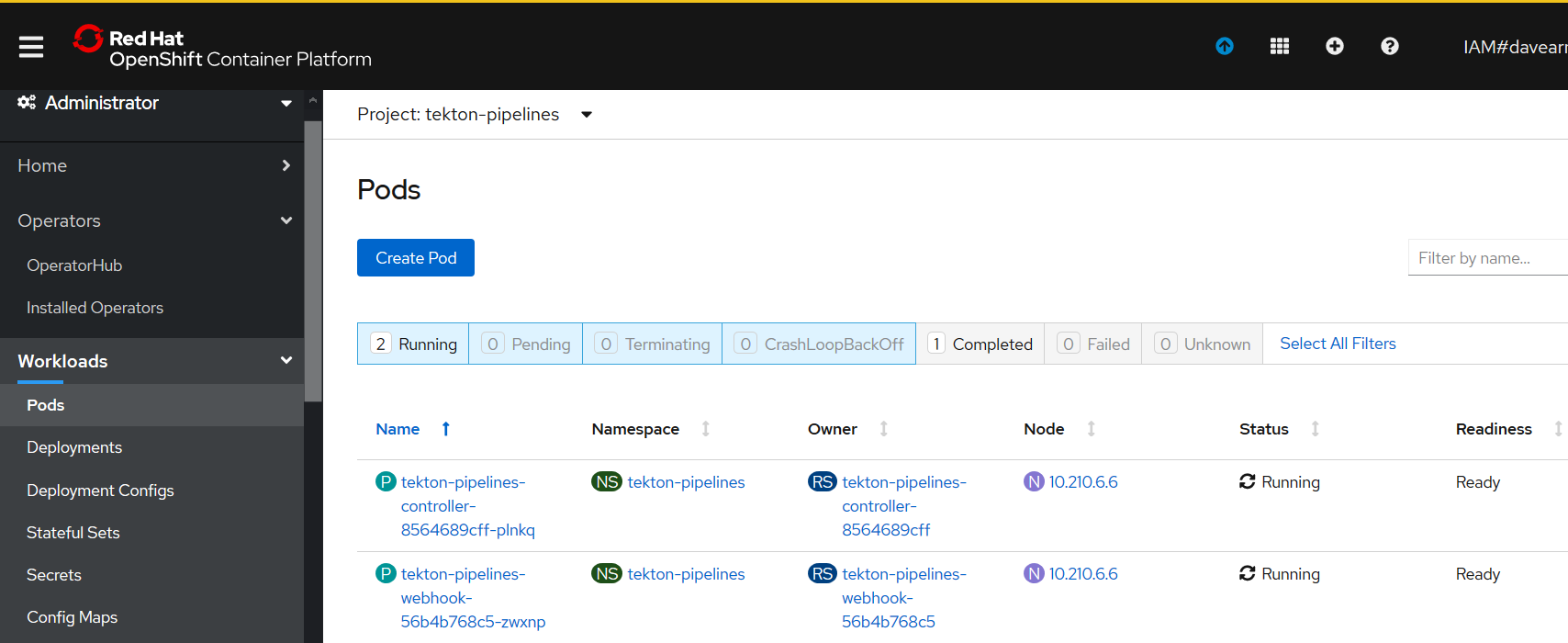


Repeat for the WebHook deployment YAML



[gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/webhook@sha256:e0a277b69a742ff0e49767d11f9a1325fcea7fb8bbbf2572af9d49116cbb2385](mailto:gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/webhook@sha256:e0a277b69a742ff0e49767d11f9a1325fcea7fb8bbbf2572af9d49116cbb2385)

The Pods should now start up correctly



### C:\openshift>oc get pods --namespace tekton-pipelines

NAME READY STATUS RESTARTS AGE

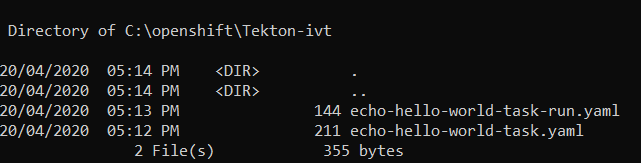
tekton-pipelines-controller-8564689cff-plnkq 1/1 Running 0 50m

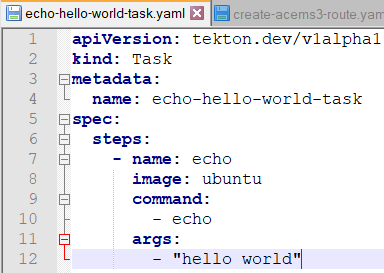
tekton-pipelines-webhook-56b4b768c5-zwxnp 1/1 Running 0 3m12s

## Test Tekton with an Echo example

As we are operating in the tekton-pipelines namespace (where tekton was installed) the default service account will be used as such you don’t need to set up a service account for pipelines at this stage.

Create two YAML files





apiVersion: tekton.dev/v1alpha1

kind: Task

metadata:

name: echo-hello-world-task

spec:

steps:

- name: echo

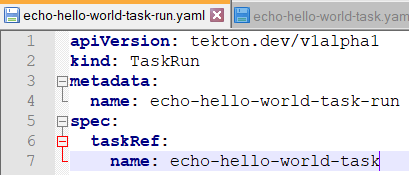
image: ubuntu

command:

- echo

args:

- "hello world"



apiVersion: tekton.dev/v1alpha1

kind: TaskRun

metadata:

name: echo-hello-world-task-run

spec:

taskRef:

name: echo-hello-world-task

### C:\openshift>oc apply -f Tekton-ivt\echo-hello-world-task.yaml

task.tekton.dev/echo-hello-world-task created

### C:\openshift>oc apply -f Tekton-ivt\echo-hello-world-task-run.yaml

taskrun.tekton.dev/echo-hello-world-task-run created

### C:\openshift>oc get taskruns/echo-hello-world-task-run -o yaml

Check the results.

apiVersion: tekton.dev/v1alpha1

kind: TaskRun

metadata:

annotations:

kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"tekton.dev/v1alpha1","kind":"Task","metadata":{"annotations":{},"name":"echo-hello-world-task","namespace":"tekton-pipelines"},"spec":{"steps":[{"args":["hello world"],"command":["echo"],"image":"ubuntu","name":"echo"}]}}

tekton.dev/release: devel

creationTimestamp: 2020-04-20T07:14:58Z

generation: 1

labels:

app.kubernetes.io/managed-by: tekton-pipelines

tekton.dev/task: echo-hello-world-task

name: echo-hello-world-task-run

namespace: tekton-pipelines

resourceVersion: "108232"

selfLink: /apis/tekton.dev/v1alpha1/namespaces/tekton-pipelines/taskruns/echo-hello-world-task-run

uid: 54bf7e0d-b1de-4989-914e-06444e067242

spec:

inputs: {}

outputs: {}

serviceAccountName: ""

taskRef:

kind: Task

name: echo-hello-world-task

timeout: 1h0m0s

status:

completionTime: 2020-04-20T07:28:25Z

conditions:

- lastTransitionTime: 2020-04-20T07:28:25Z

message: All Steps have completed executing

reason: Succeeded

status: "True"

type: Succeeded

podName: echo-hello-world-task-run-pod-pz6vx

startTime: 2020-04-20T07:14:58Z

steps:

- container: step-echo

imageID: docker.io/library/ubuntu@sha256:bec5a2727be7fff3d308193cfde3491f8fba1a2ba392b7546b43a051853a341d

name: echo

terminated:

containerID: cri-o://f04dd508f82092cb3978f14cfc0bedc64aa110082fc557790790577cf265d025

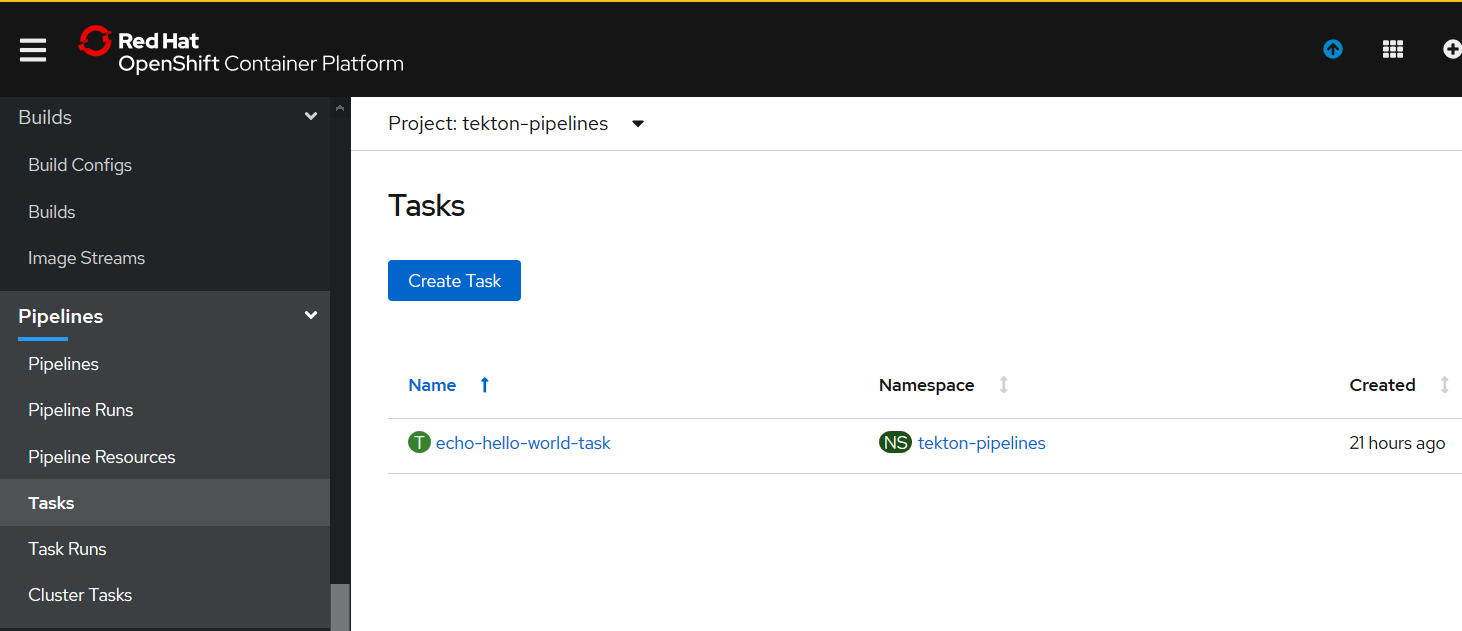
exitCode: 0

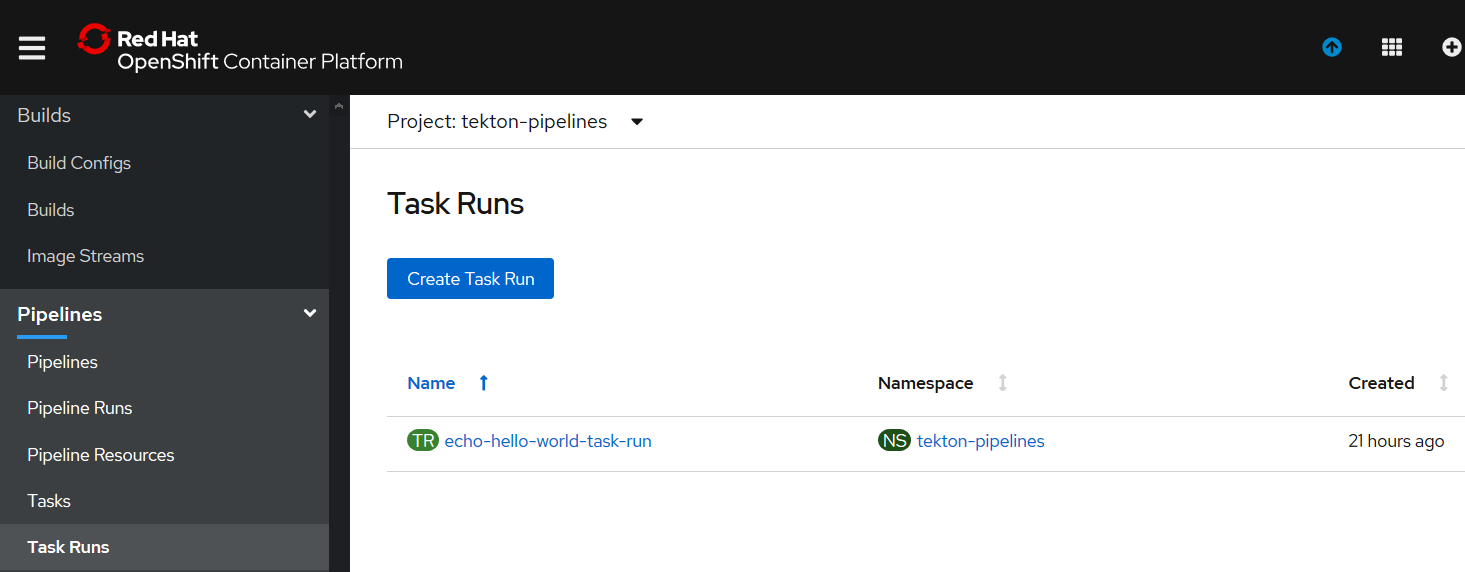
finishedAt: 2020-04-20T07:28:24Z

reason: Completed

startedAt: 2020-04-20T07:28:24Z

### OpenShift Console Pipelines



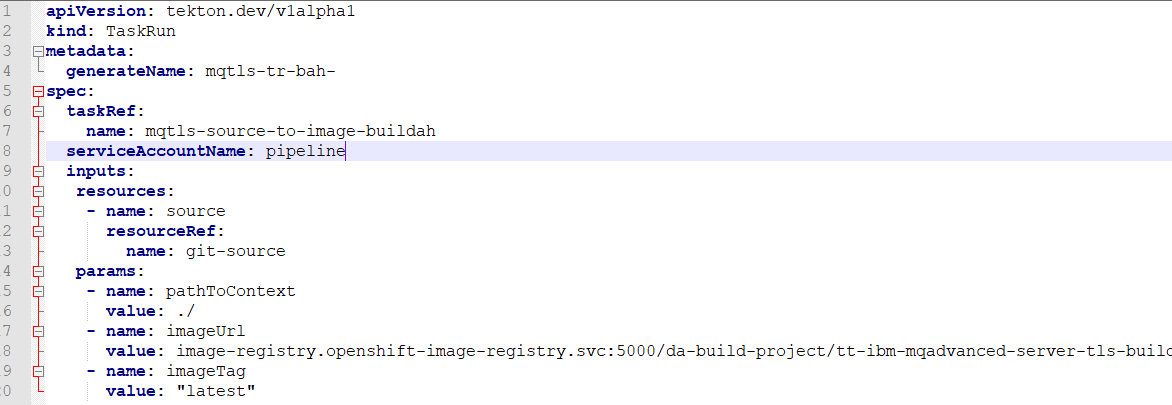


## Tekton Pipeline for IBM MQ (tls enabled)

### Create the Pipeline Service Account

We are now going to create and use tekton artifacts in the da-build-project namespace. The default service account for this namespace will not have access to run tekton. So we will create a service account called “pipeline” and this service account name will be used in the “serviceaccountname” parameter in the Spec: for Taskruns and Pipelineruns

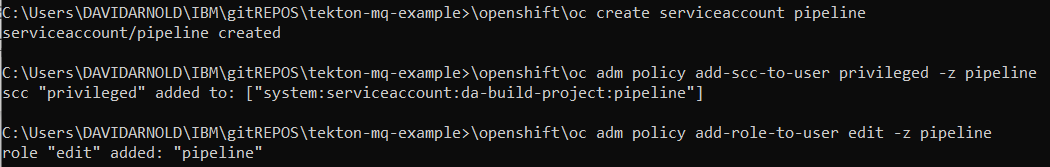
For example:



c:\openshift\oc create serviceaccount pipeline

c:\openshift\oc adm policy add-scc-to-user privileged -z pipeline

c:\openshift\oc adm policy add-role-to-user edit -z pipeline



### Create the (non executing)Tekton artifacts in RH Openshift

#### Resources

c:\openshift\oc apply -f c:\users\DAVIDARNOLD\IBM\gitREPOS\tekton-mq-example\tekton\resources\mqtls-git.yaml

#### Tasks

c:\openshift\oc apply -f c:\users\DAVIDARNOLD\IBM\gitREPOS\tekton-mq-example\tekton\tasks\mqtls-source-to-image-buildah.yaml

c:\openshift\oc apply -f c:\users\DAVIDARNOLD\IBM\gitREPOS\tekton-mq-example\tekton\tasks\mqtls-deploy-using-kubectl.yaml

#### Pipeline

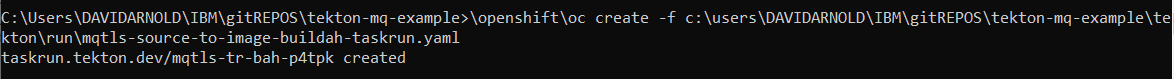
c:\openshift\oc apply -f c:\users\DAVIDARNOLD\IBM\gitREPOS\tekton-mq-example\tekton\pipeline\mqtls-build-and-deploy-pipeline-buildah.yaml

### Create and test the Tekton Taskrun artifacts in RH Openshift

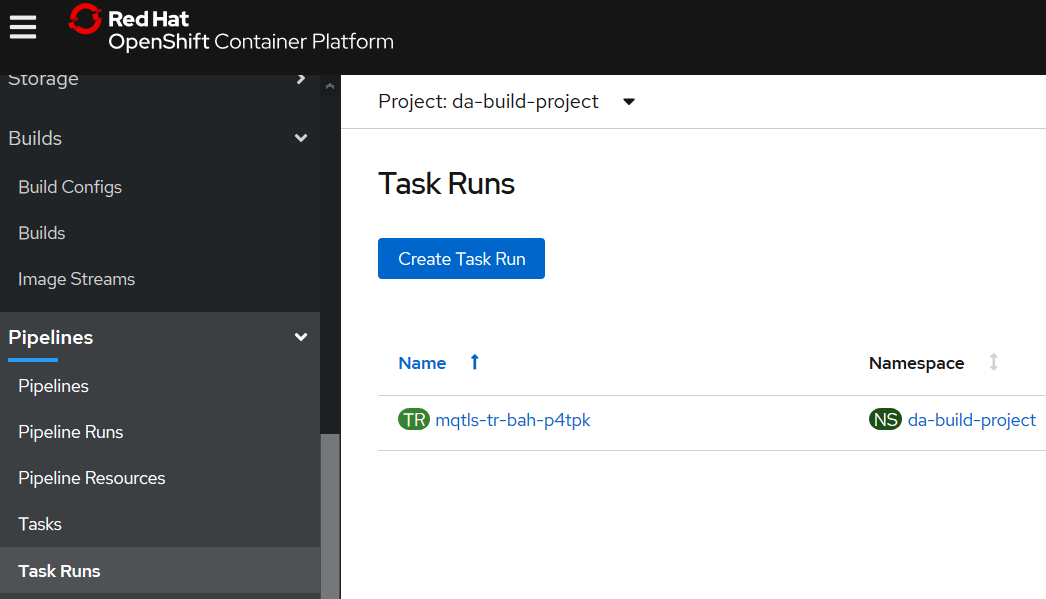
#### Taskrun for Buildah Build task

c:\openshift\oc create -f c:\users\DAVIDARNOLD\IBM\gitREPOS\tekton-mq-example\tekton\run\mqtls-source-to-image-buildah-taskrun.yaml

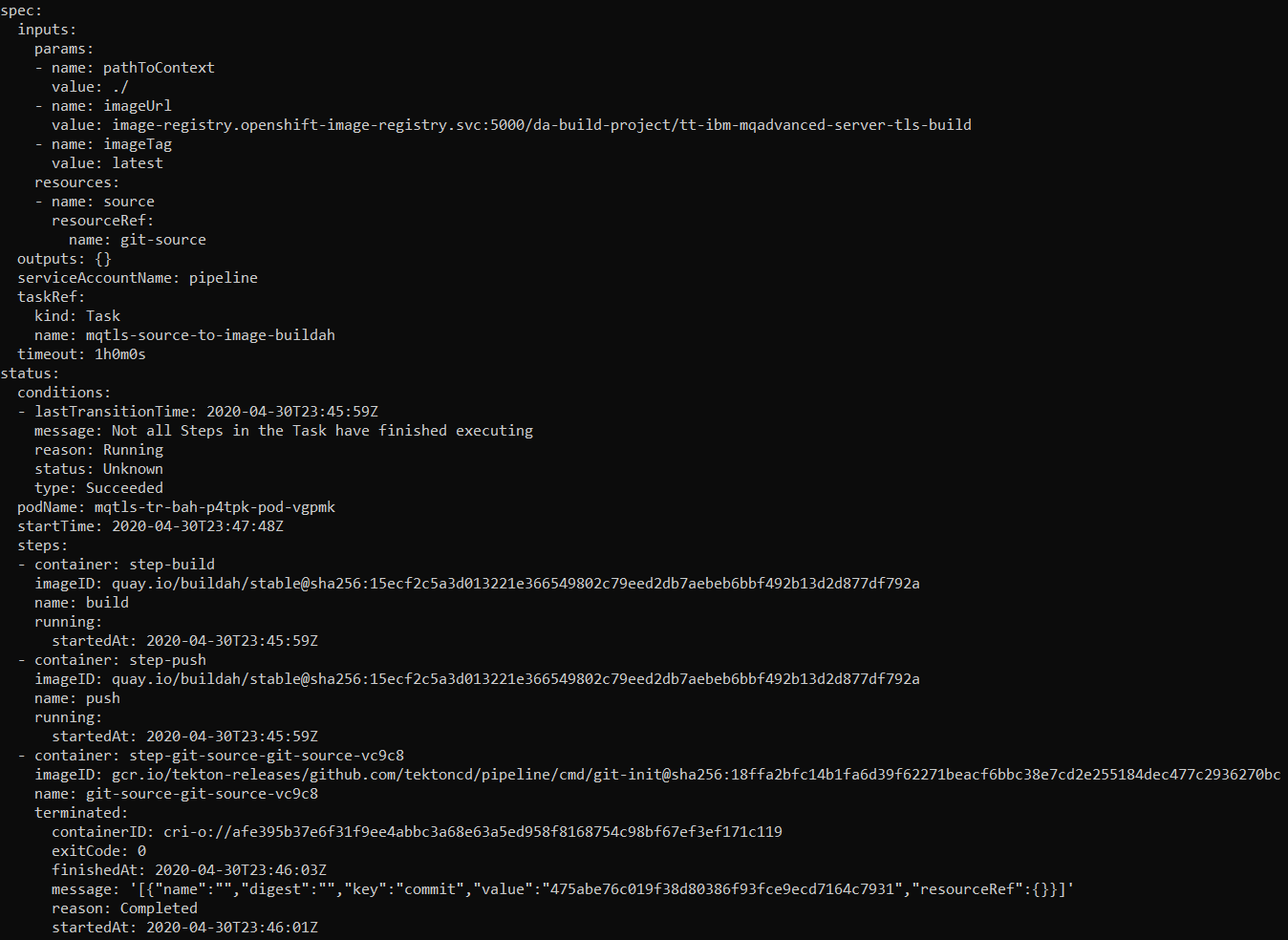
creation of the taskrun will trigger the task.



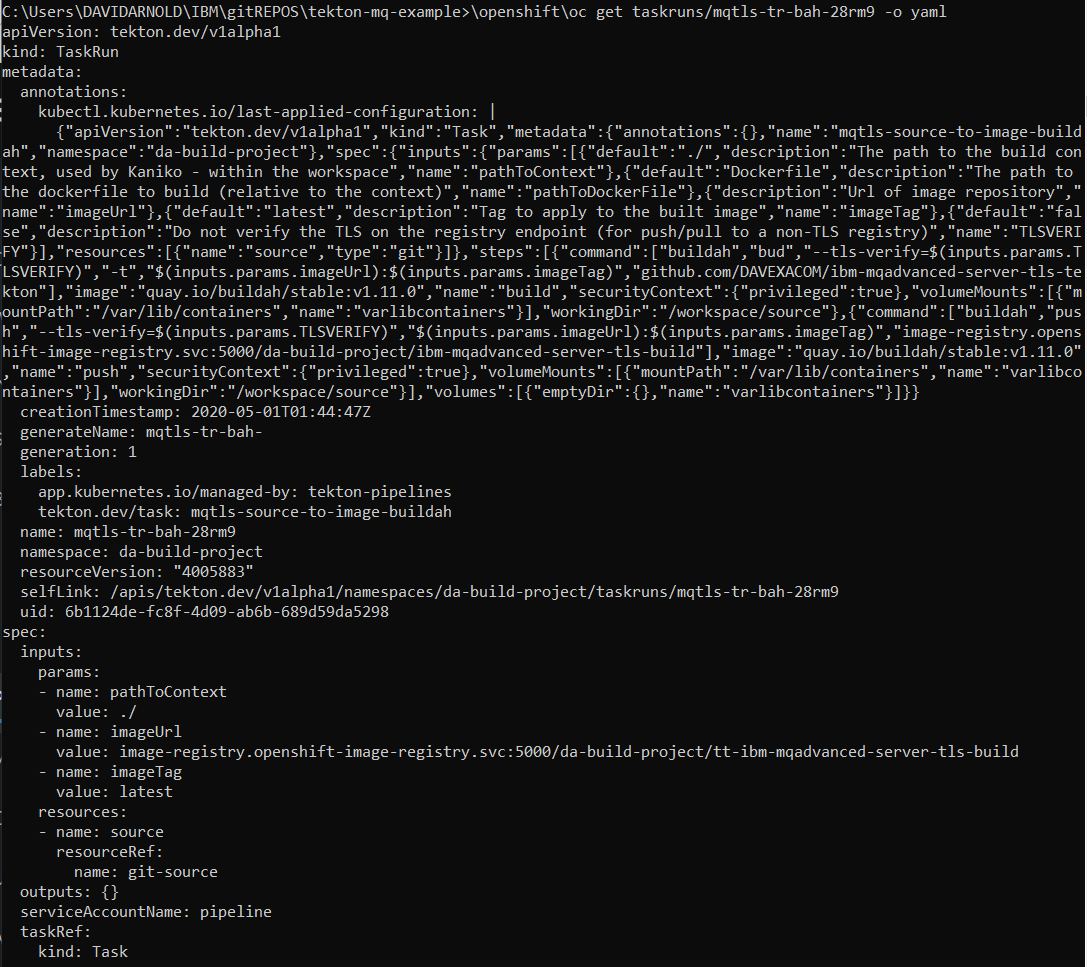
Check the RH Openshift console->pipelines->Task runs

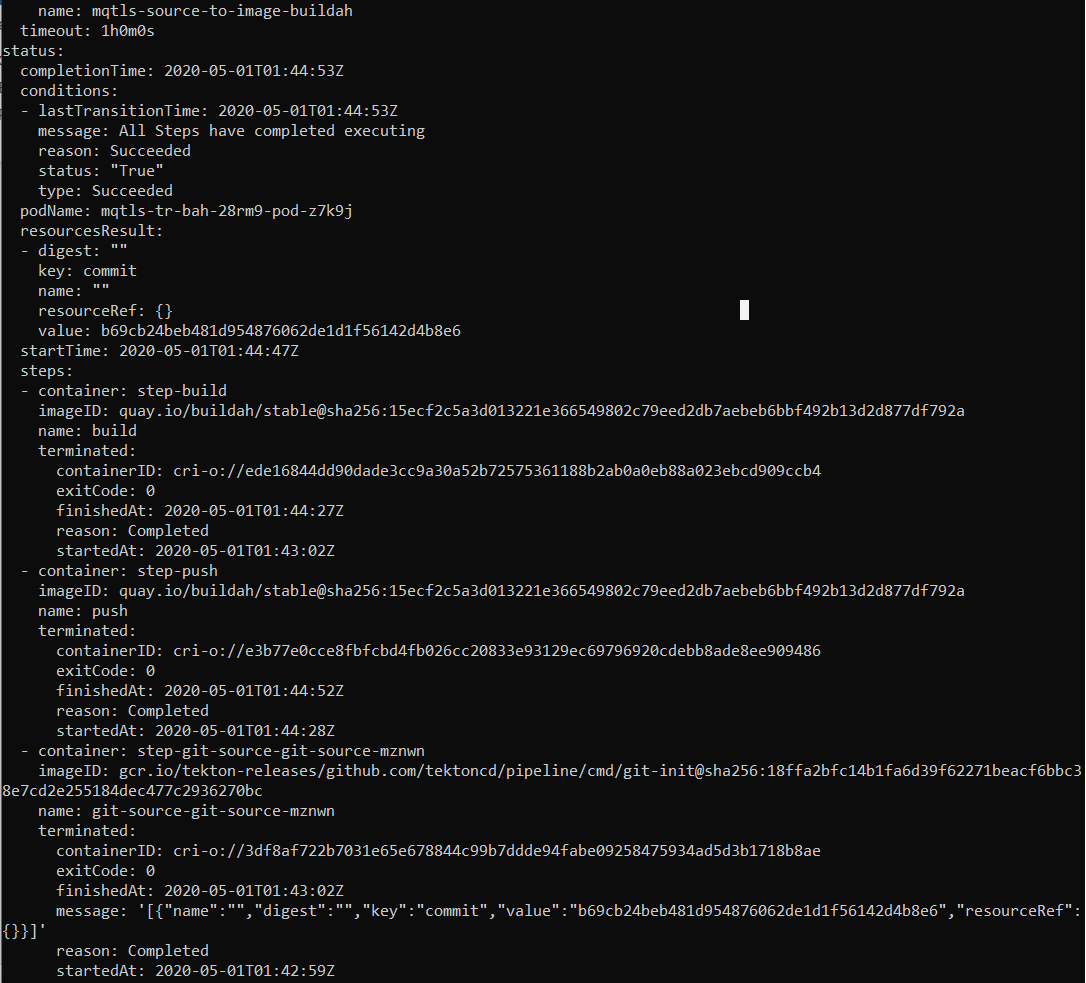


C:\openshift\oc get taskruns/mqtls-tr-bah-p4tpk -o yaml



You may have to run the C:\openshift\oc get taskruns/mqtls-tr-bah-p4tpk -o yaml command a number of times as the task is in flight.



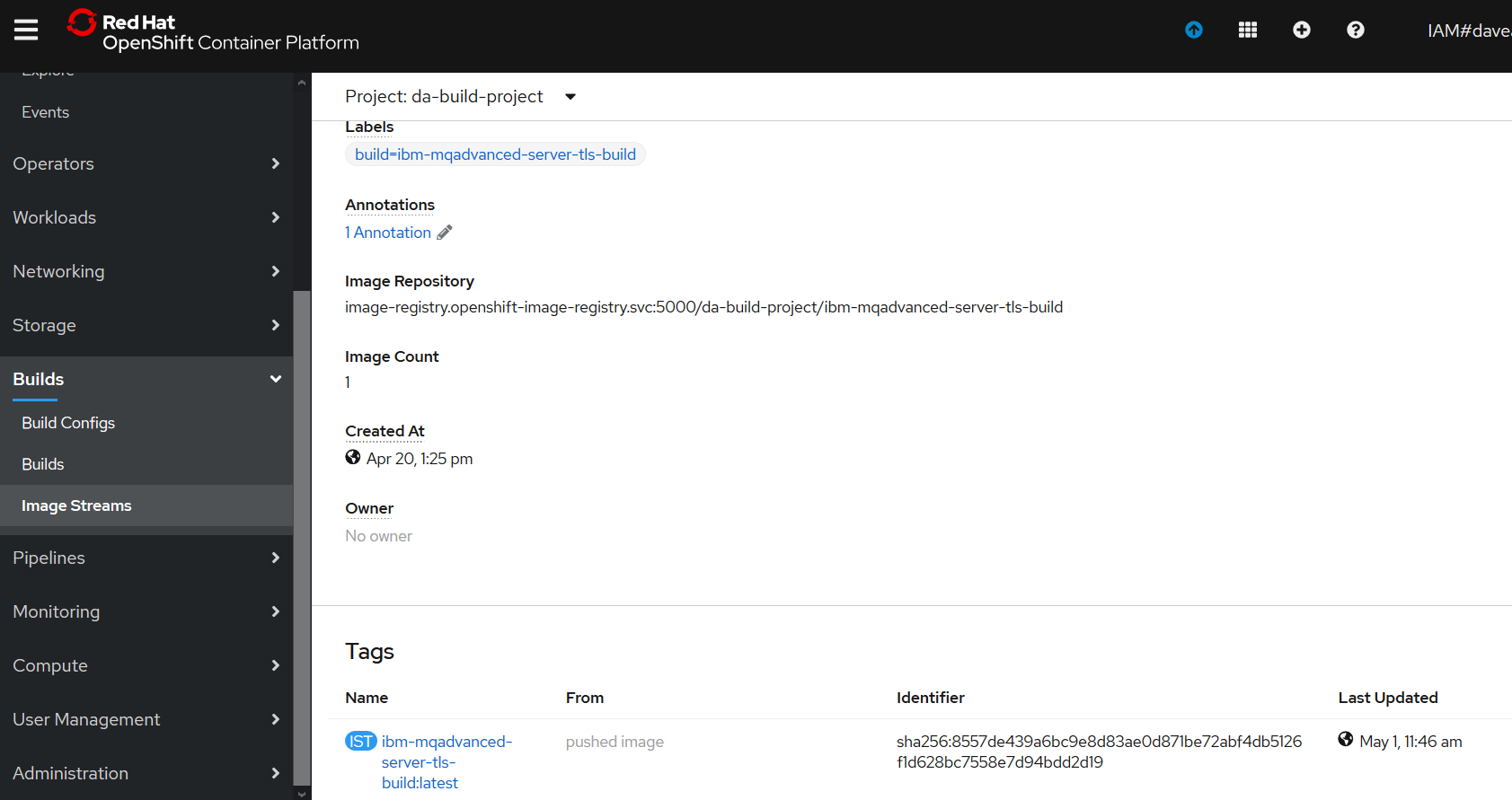


If you get errors jump to the [Debugging Appendix](#_Debugging_Appendix)

If it worked check the results in the RH Openshift console

RH Openshift Console->Builds->ImageStreams->ibm-mqadvanced-server-tls-build

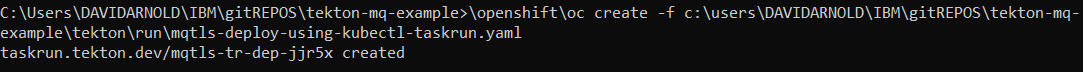
Note last update date and time



#### Taskrun for Kubectl Deploy task

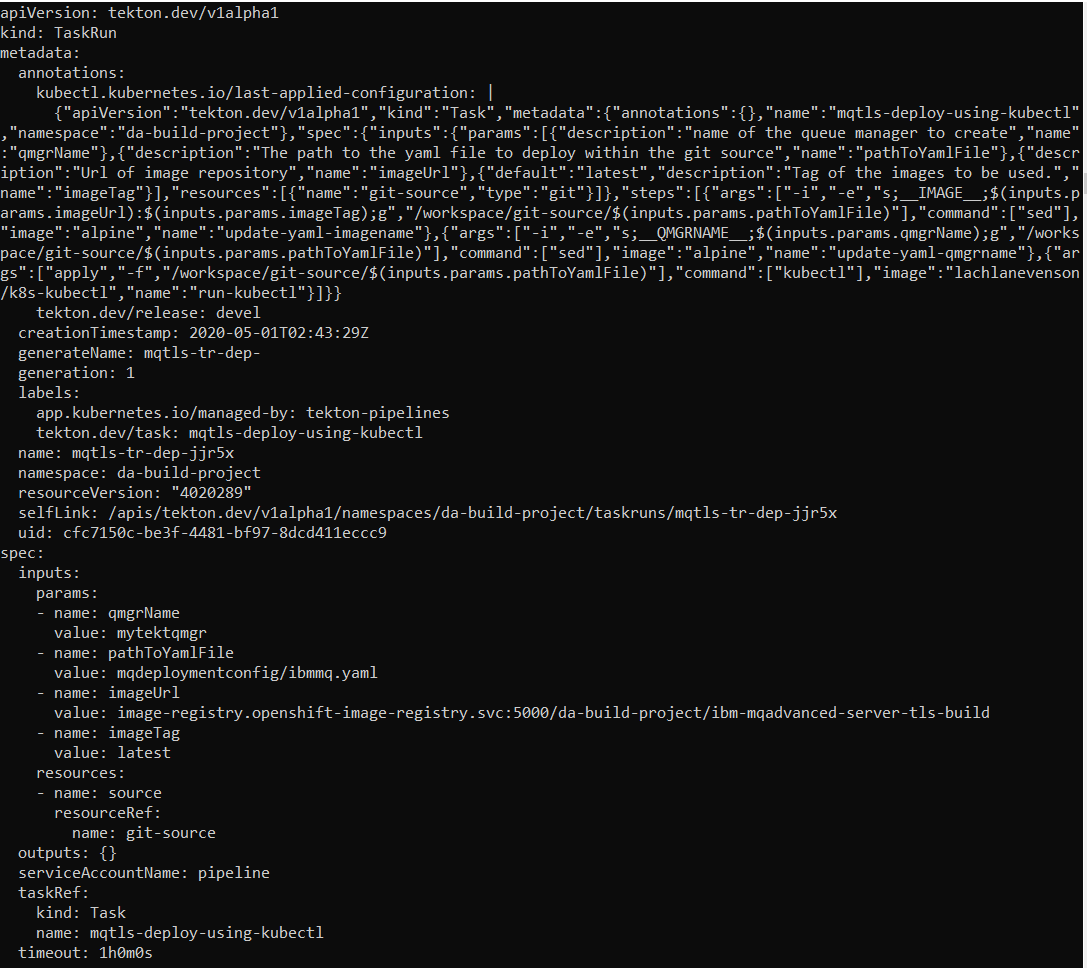
creation of the taskrun will trigger the task.

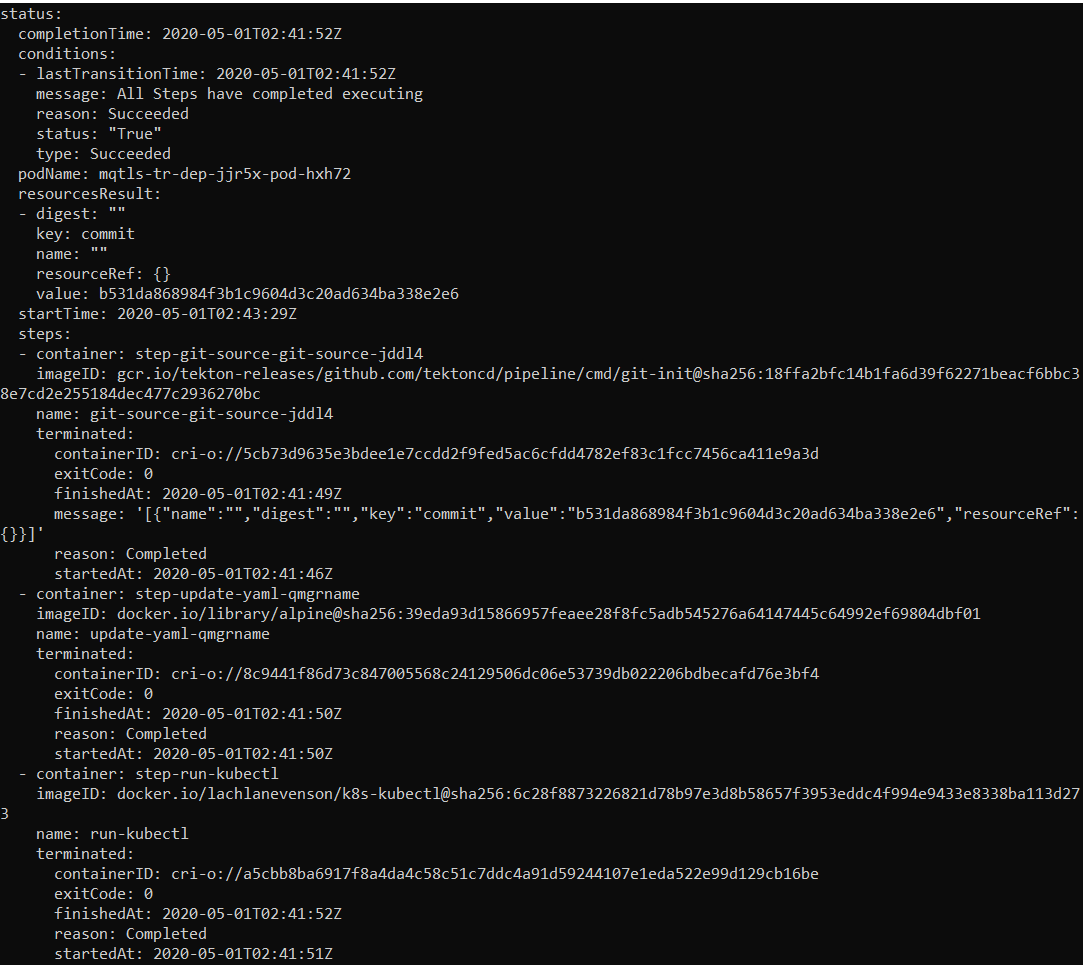
C:\openshift\oc create -f c:\users\DAVIDARNOLD\IBM\gitREPOS\tekton-mq-example\tekton\run\mqtls-deploy-using-kubectl-taskrun.yaml

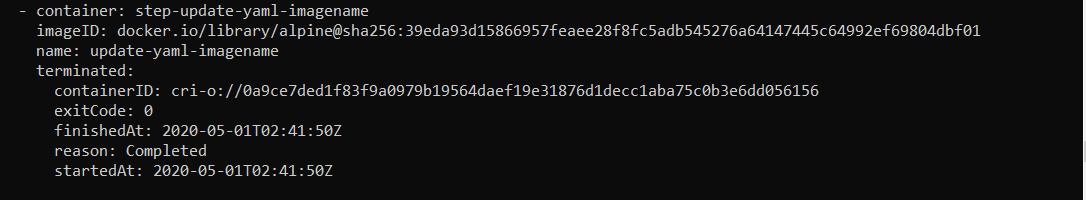


C:\openshift\oc get taskruns/mqtls-tr-dep-jjr5x -o yaml

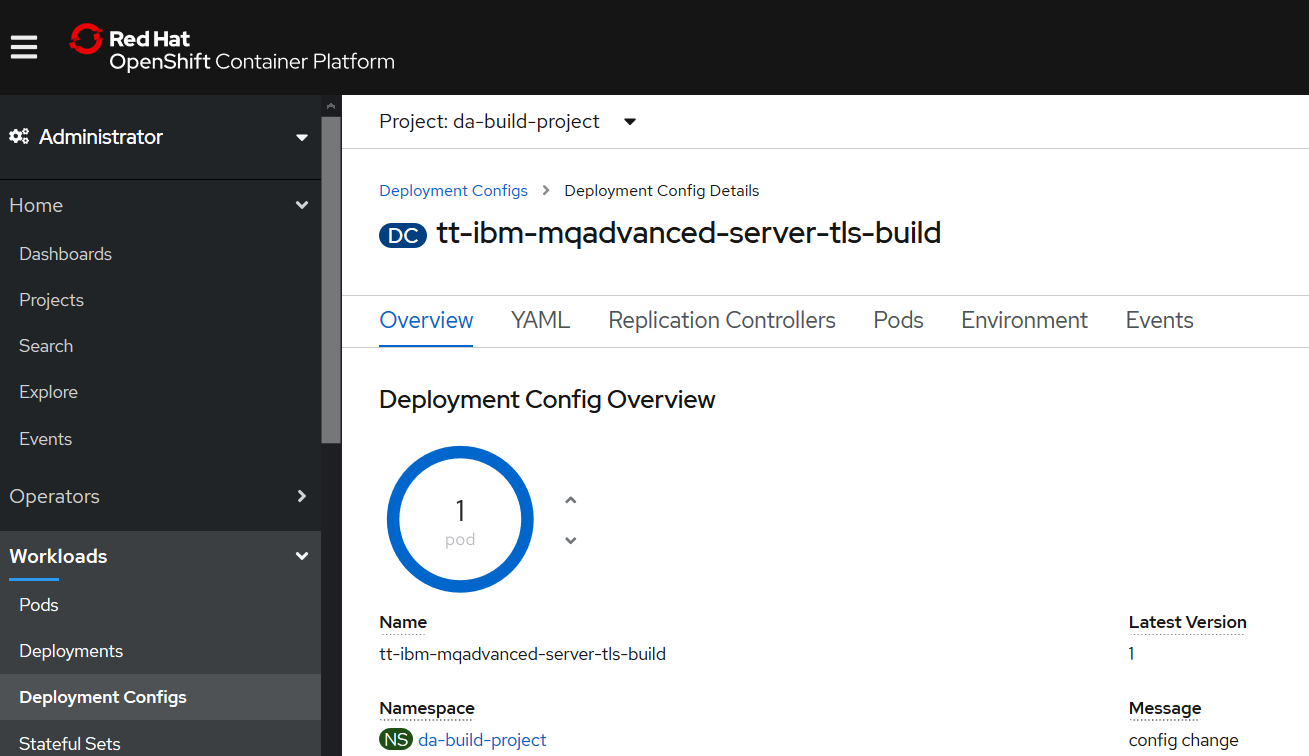
If you get errors jump to the [debug Appendix](#_Taskrun_for_deploy)

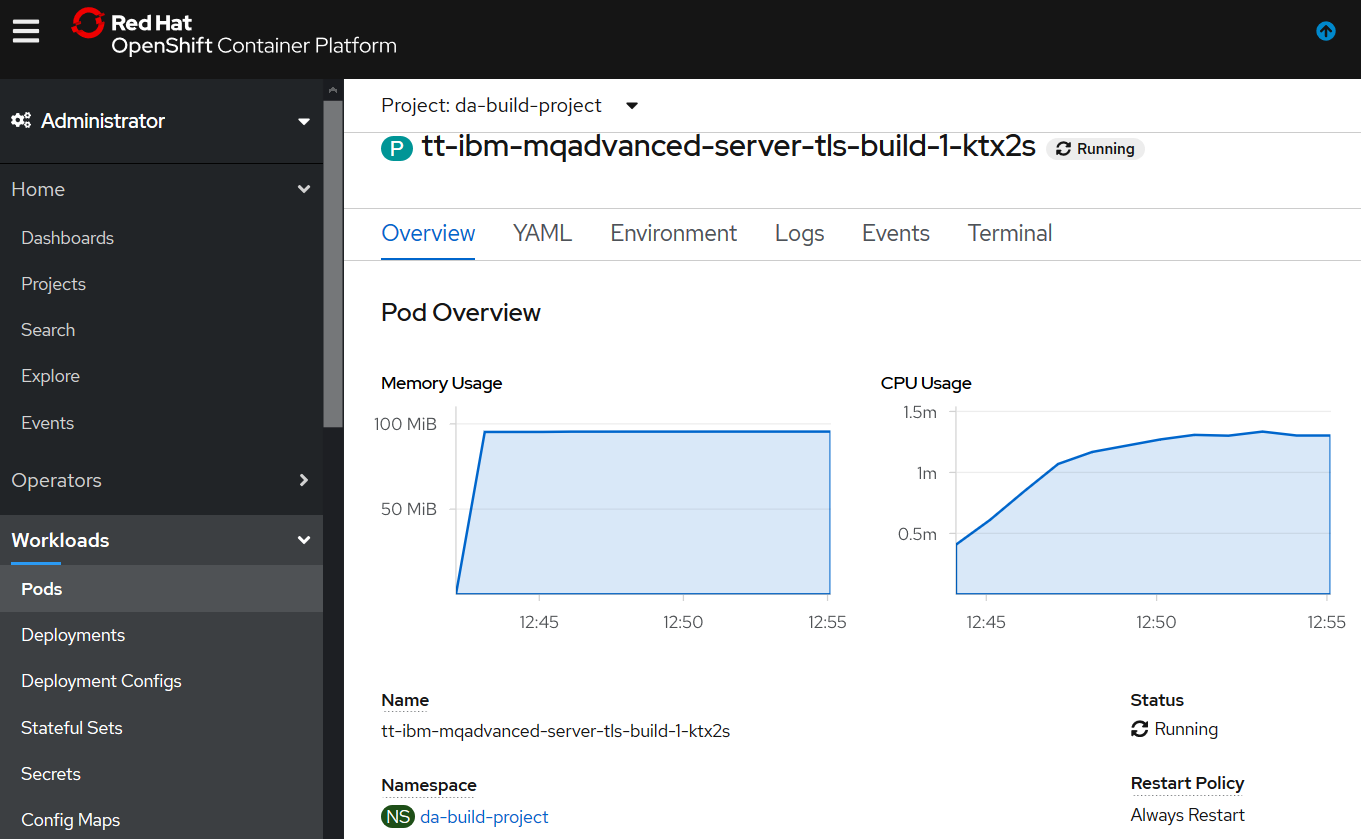


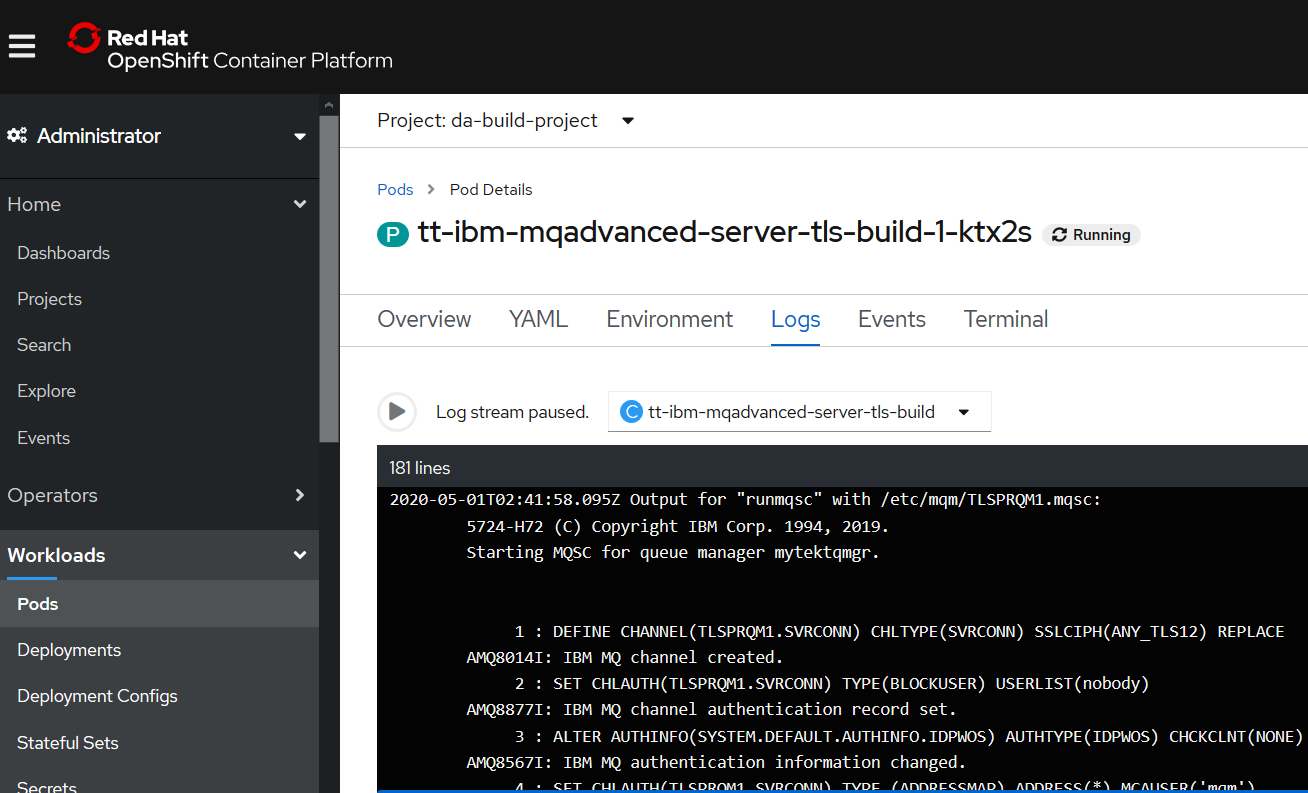




Check the results in the RH Openshift Console->Deployment Config->tt-ibm-mqadvanced-server-tls-build







#### Pipelinerun

### Updating the Tekton artifacts in RH Openshift

USE OC REPLACE if you change and files

oc replace -f c:\users\DAVIDARNOLD\IBM\gitREPOS\tekton-ace-example\tekton\tasks\source-to-image-buildah.yaml

oc apply -f Tekton-ivt\echo-hello-world-task.yaml

### Deleting the Tekton artifacts in RH Openshift

USE OC DELETE -f c:\users\davidarnold\ibm\github\tekton-mq-example\tekton\ other\deleteartifacts.yaml to remove any artifacts from RH OpenShift - use a yaml snippet in a file to identify kind: , name: and namespace: see example deleteartifacts.yaml below

apiVersion: tekton.dev/v1alpha1

kind: Task

metadata:

creationTimestamp: '2020-03-03T00:27:03Z'

generation: 3

name: mqtls-source-to-image-buildah

namespace: da-build-project

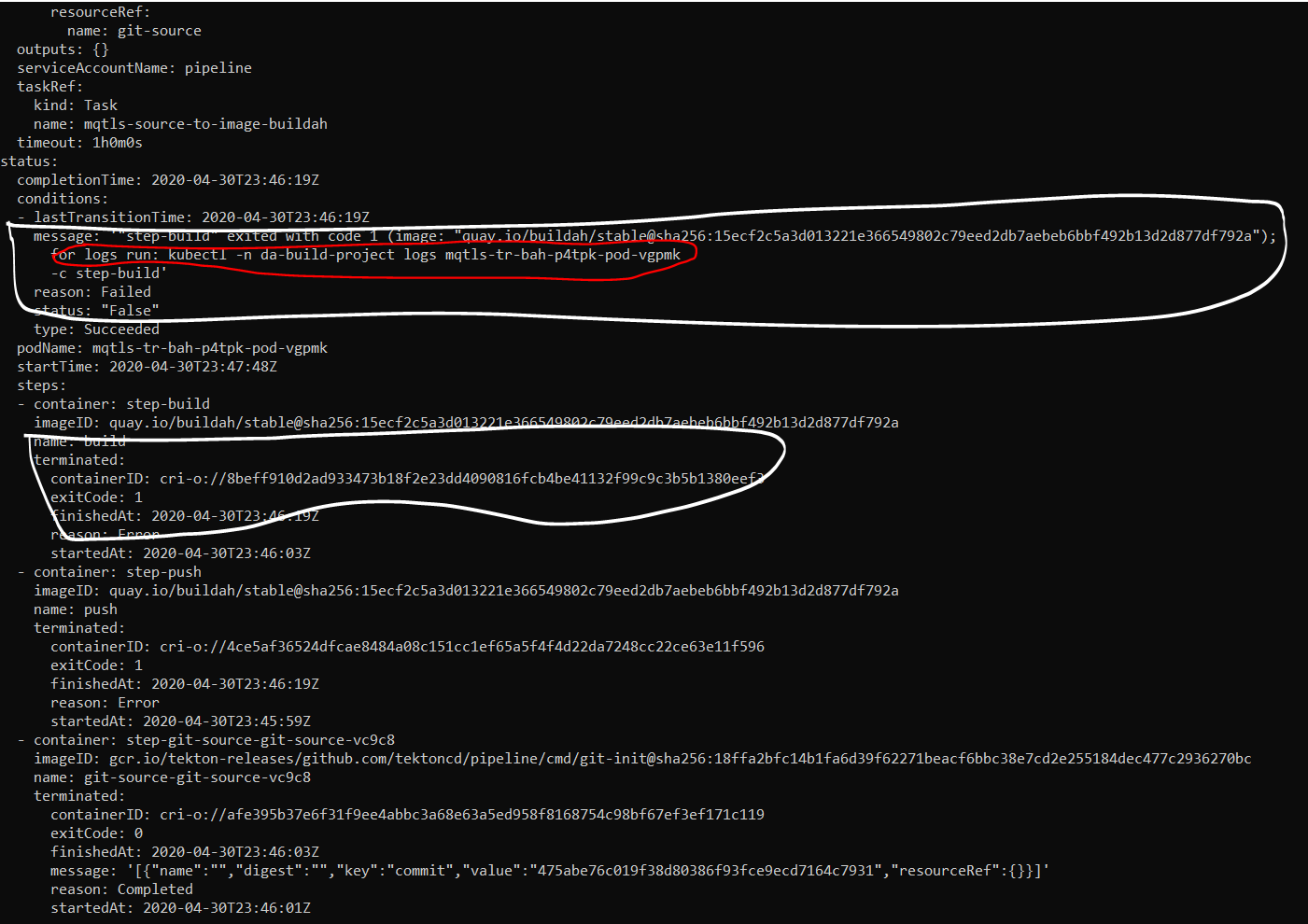
### Testing - Pipelinerun

# Debugging Appendix

## Taskrun for build fails

When the task runs completes you need to check its logs for success for failure.

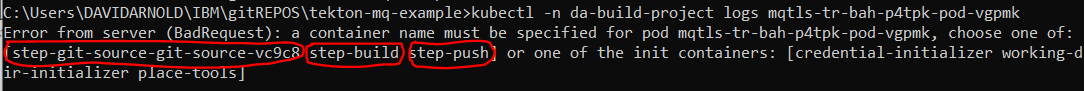
C:\Users\DAVIDARNOLD\IBM\gitREPOS\tekton-mq-example>\openshift\oc get taskruns/mqtls-tr-bah-p4tpk -o yaml



### Inspect the logs

C:\Users\DAVIDARNOLD\IBM\gitREPOS\tekton-mq-example>kubectl -n da-build-project logs mqtls-tr-bah-p4tpk-pod-vgpmk

There are 3 logs to inspect



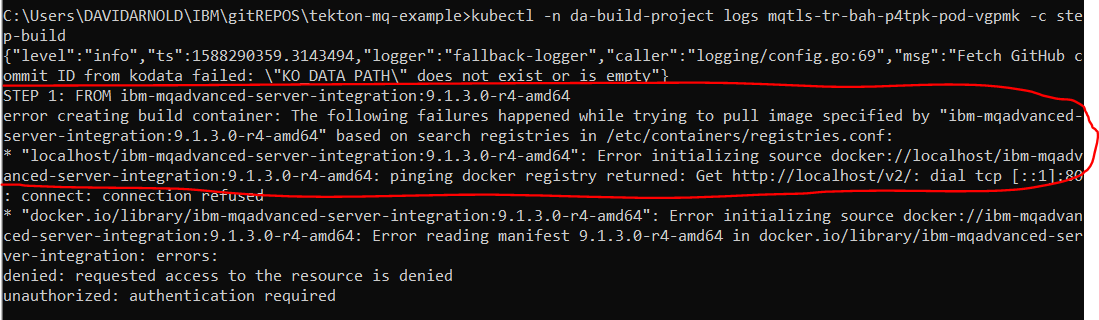
kubectl -n da-build-project logs mqtls-tr-bah-p4tpk-pod-vgpmk -c **step-git-source-git-source-vc9c8**



Looks like we successfully cloned the github source repository into **workspace/source**

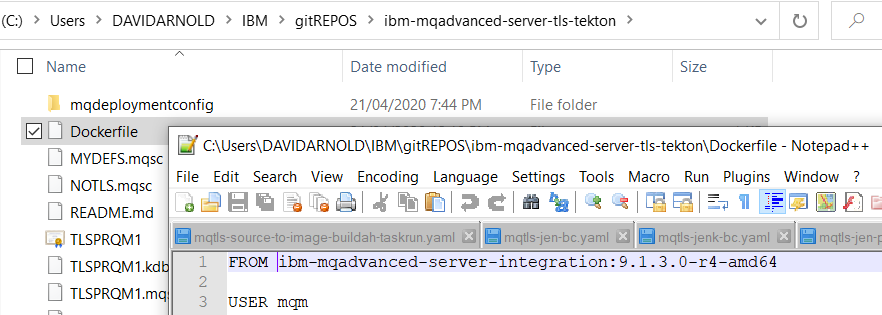
note: potentially subsequent steps and tasks will need to reference **workspace/source**.

kubectl -n da-build-project logs mqtls-tr-bah-p4tpk-pod-vgpmk -c **step-build**



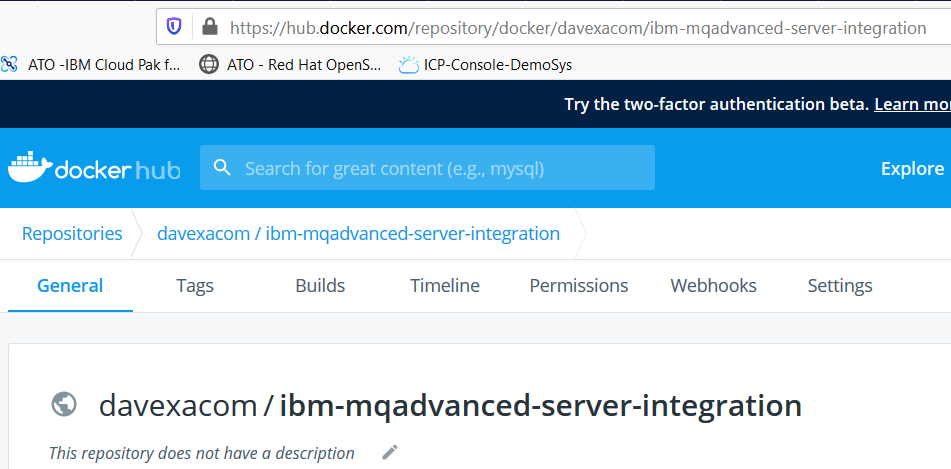
Looks like a problem finding the MQ TLs base image to build FROM in step one of the docker file in the GitHub repos we cloned

Check the docker file



Change the dockerfile to match the image location on dockerhub or other registry

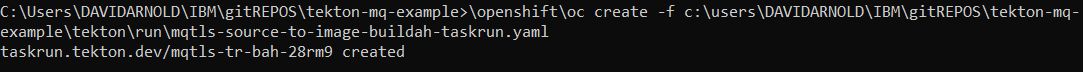




Push your changes to Github

Create a new instance of the taskrun

C:\openshift\oc create -f c:\users\DAVIDARNOLD\IBM\gitREPOS\tekton-mq-example\tekton\run\mqtls-source-to-image-buildah-taskrun.yaml



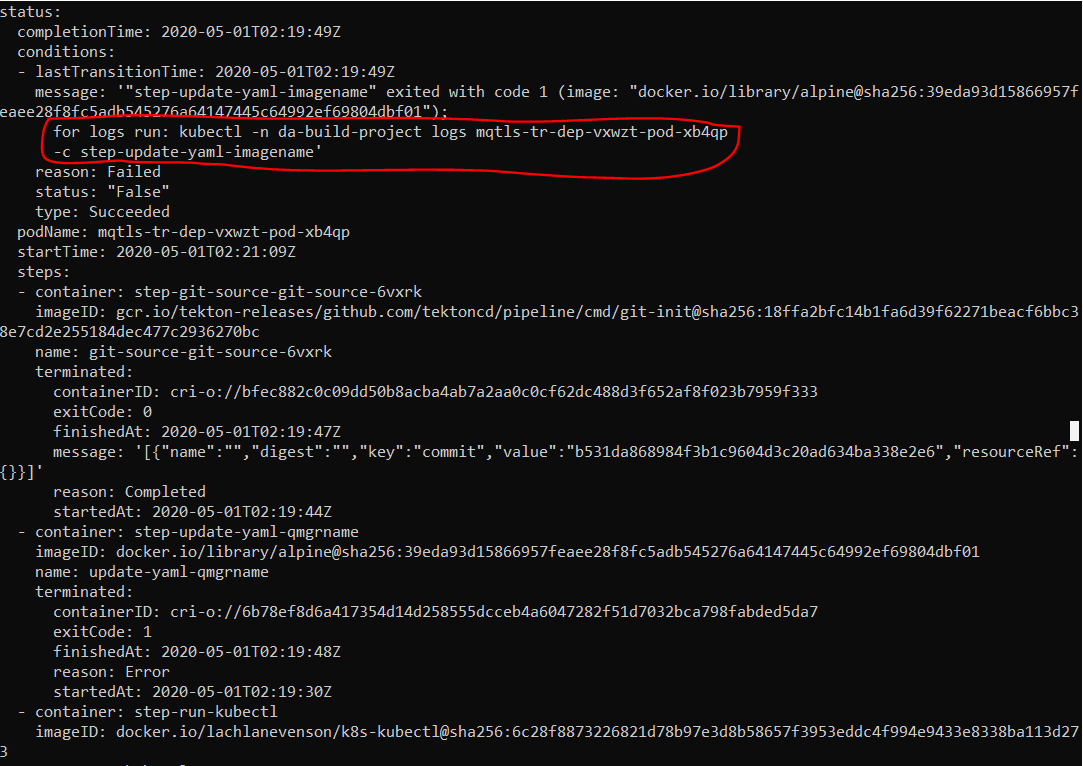
Get the output from run

C:\openshift\oc get taskruns/mqtls-tr-bah-28rm9 -o yaml

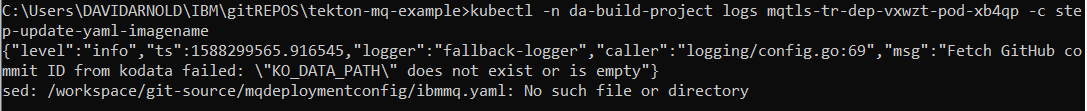
## Taskrun for deploy fails

When the task runs completes you need to check its logs for success for failure.

C:\Users\DAVIDARNOLD\IBM\gitREPOS\tekton-mq-example>\openshift\ oc get taskruns/mqtls-tr-dep-vxwzt -o yaml



kubectl -n da-build-project logs mqtls-tr-dep-vxwzt-pod-xb4qp -c step-update-yaml-imagename



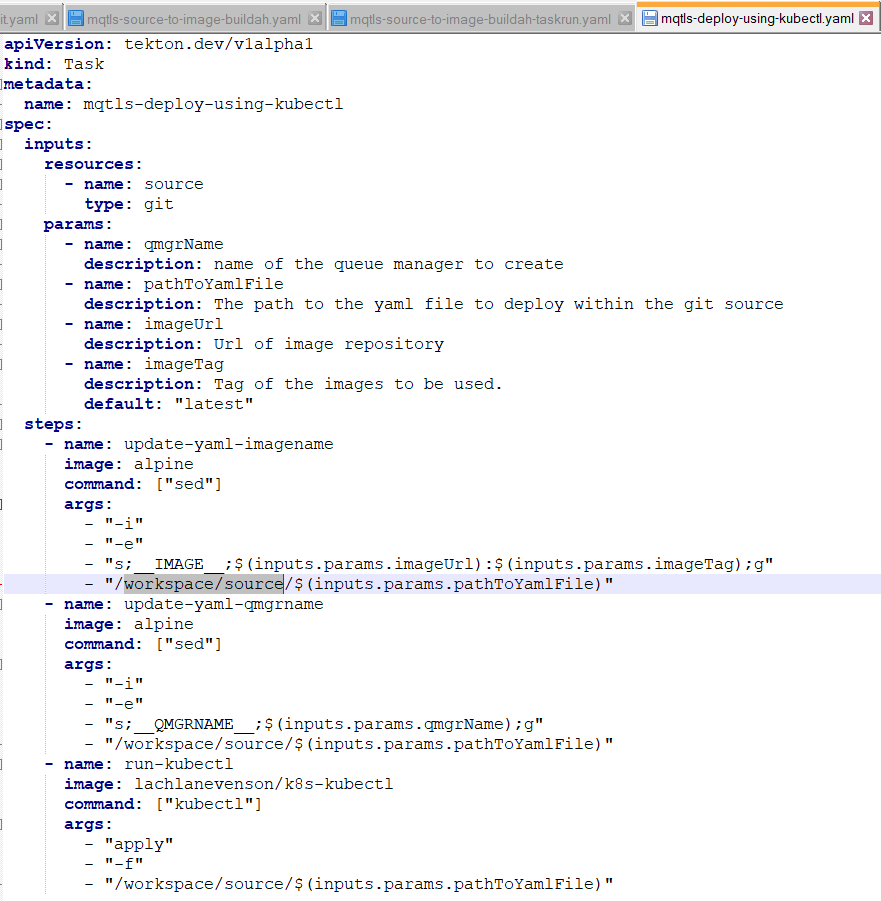
Looks like we have a mismatch between where the Buildah step cloned the git repos to and where this deploy step is looking for the ibmmq.yaml file

Buildah task cloned to: workspace/source

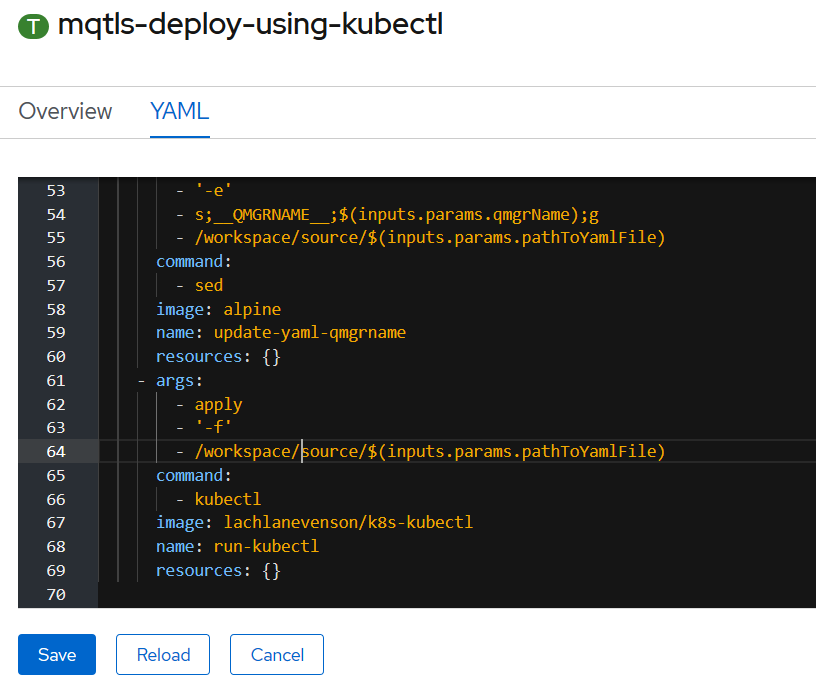
Deploy task is looking in: workspace/git-source

Correct the deploy task YAML.

You can do this in the source and oc apply to update the task



Or edit in the RH Openshift console

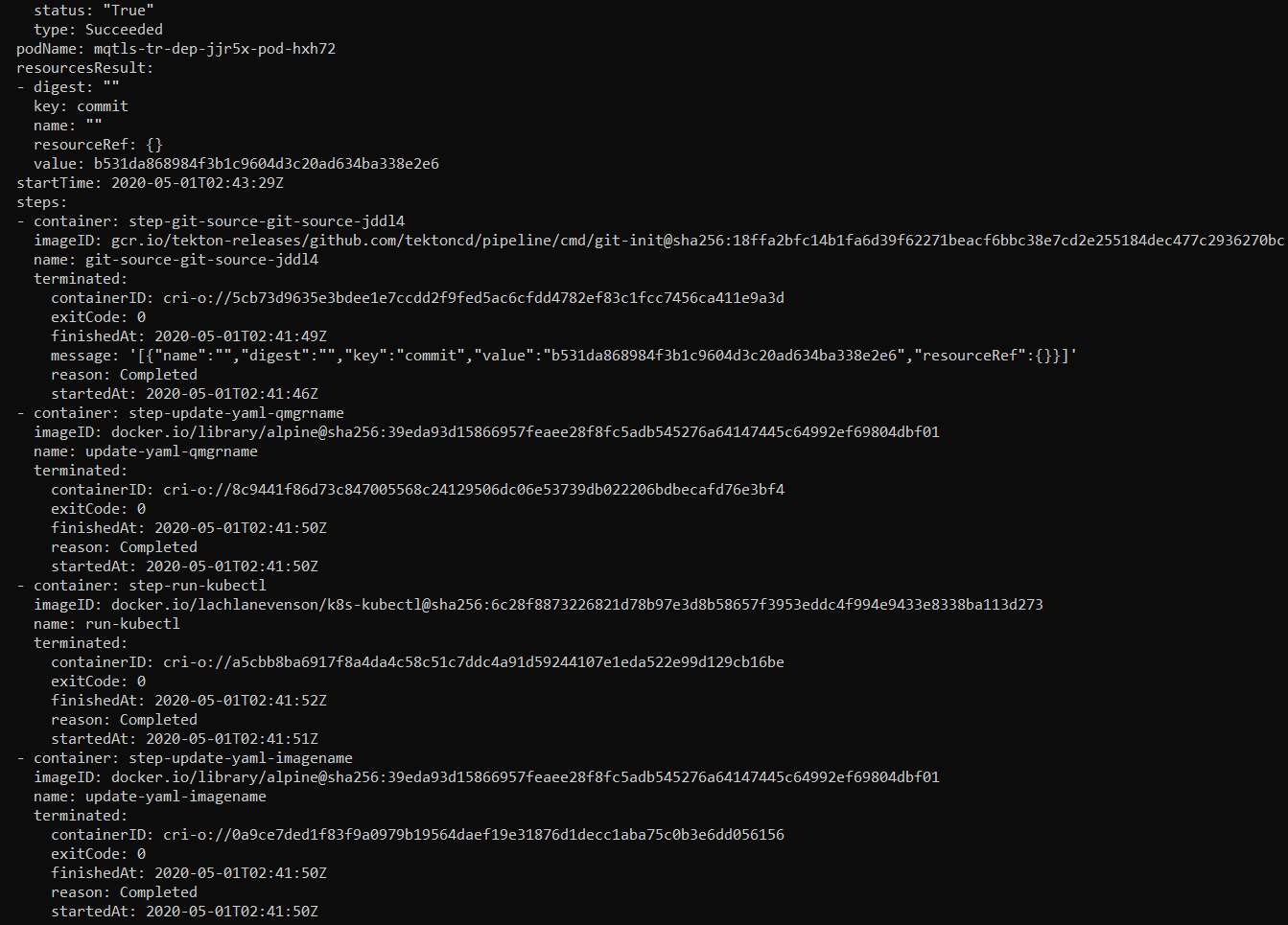


Retry the deploy Taskrun

C:\openshift\oc create -f c:\users\DAVIDARNOLD\IBM\gitREPOS\tekton-mq-example\tekton\run\mqtls-deploy-using-kubectl-taskrun.yaml

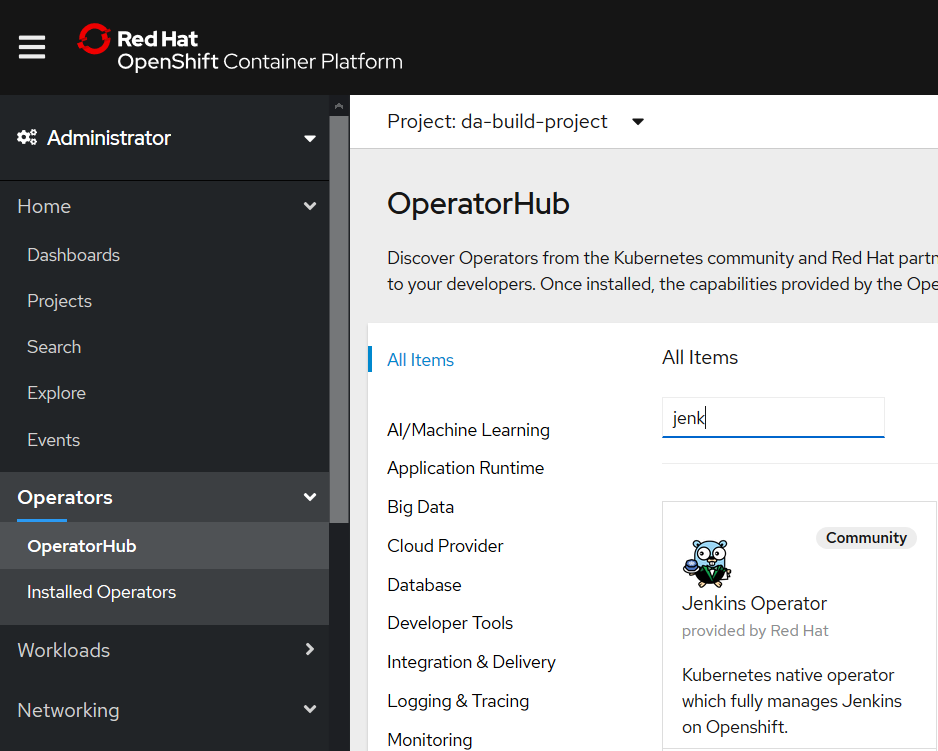


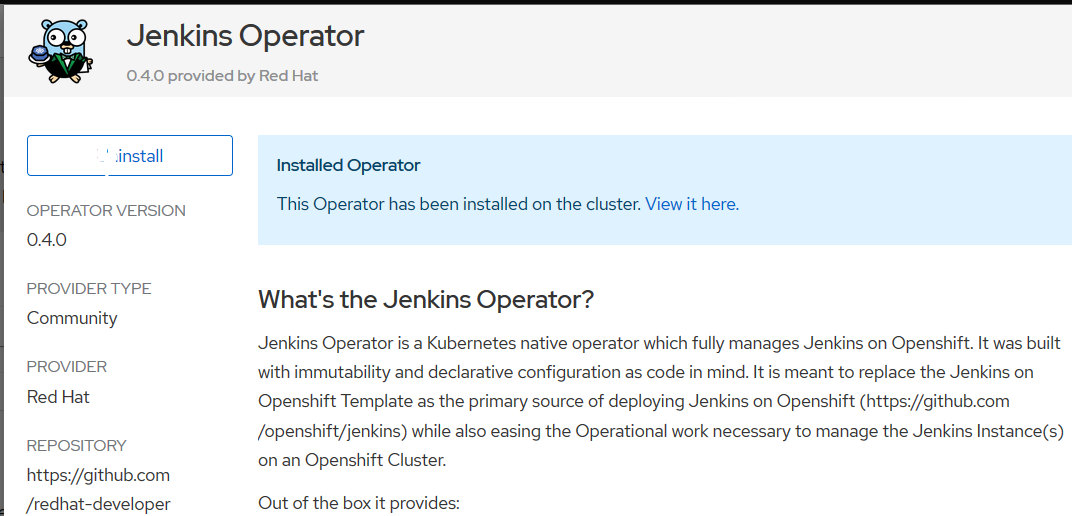
C:\openshift\oc get taskruns/mqtls-tr-dep-jjr5x -o yaml



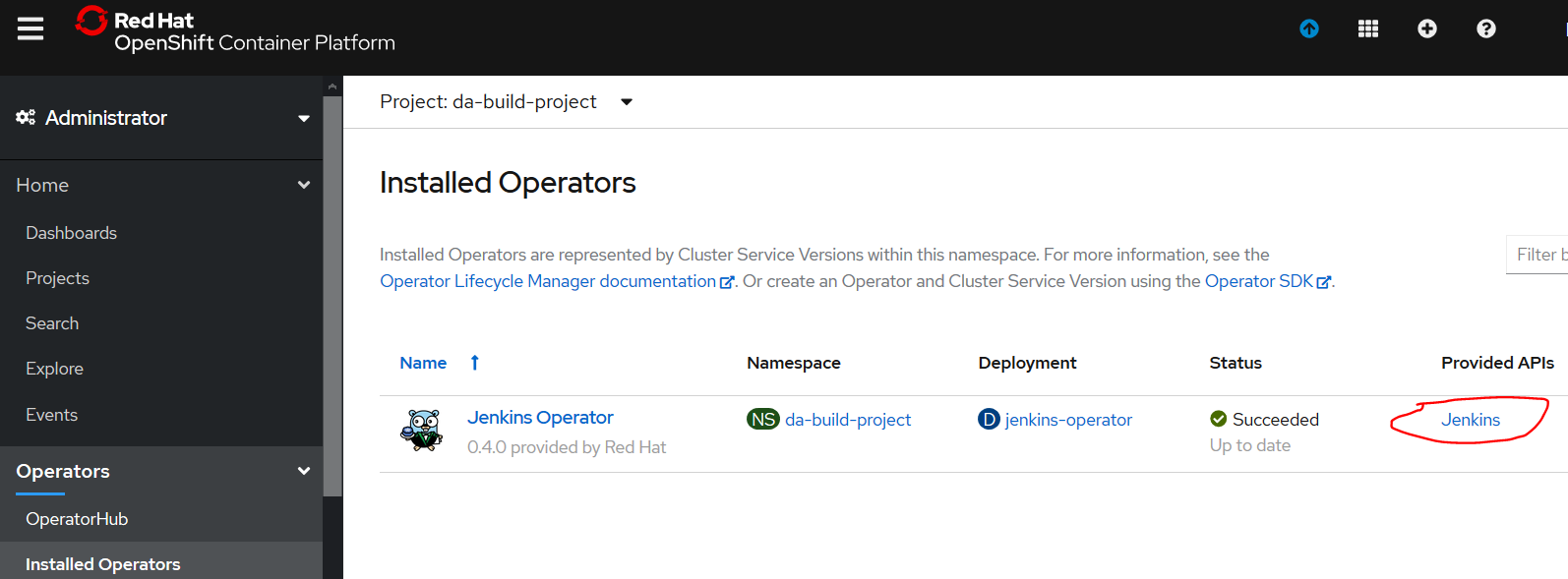
# Jenkins

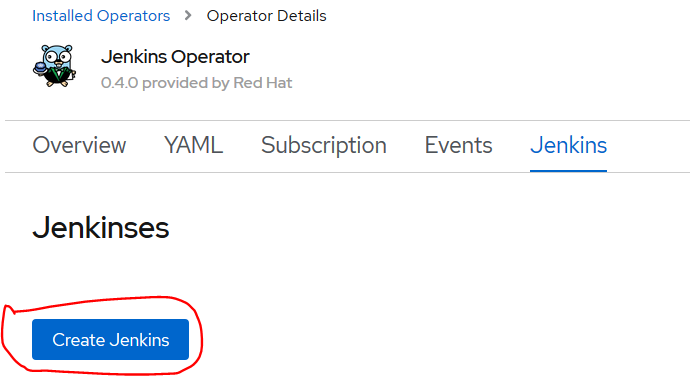
## Install the operator from the operator hub

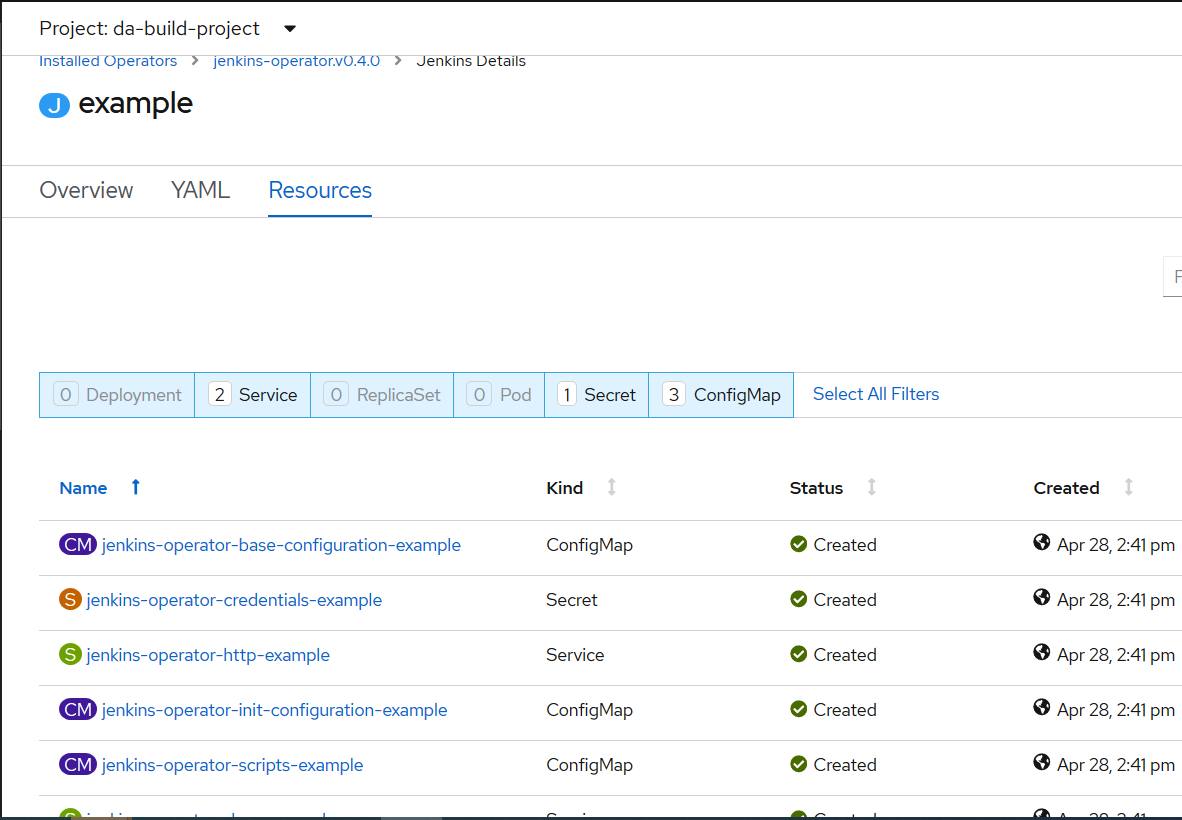




## Create a Jenkins from the operator on your cluster





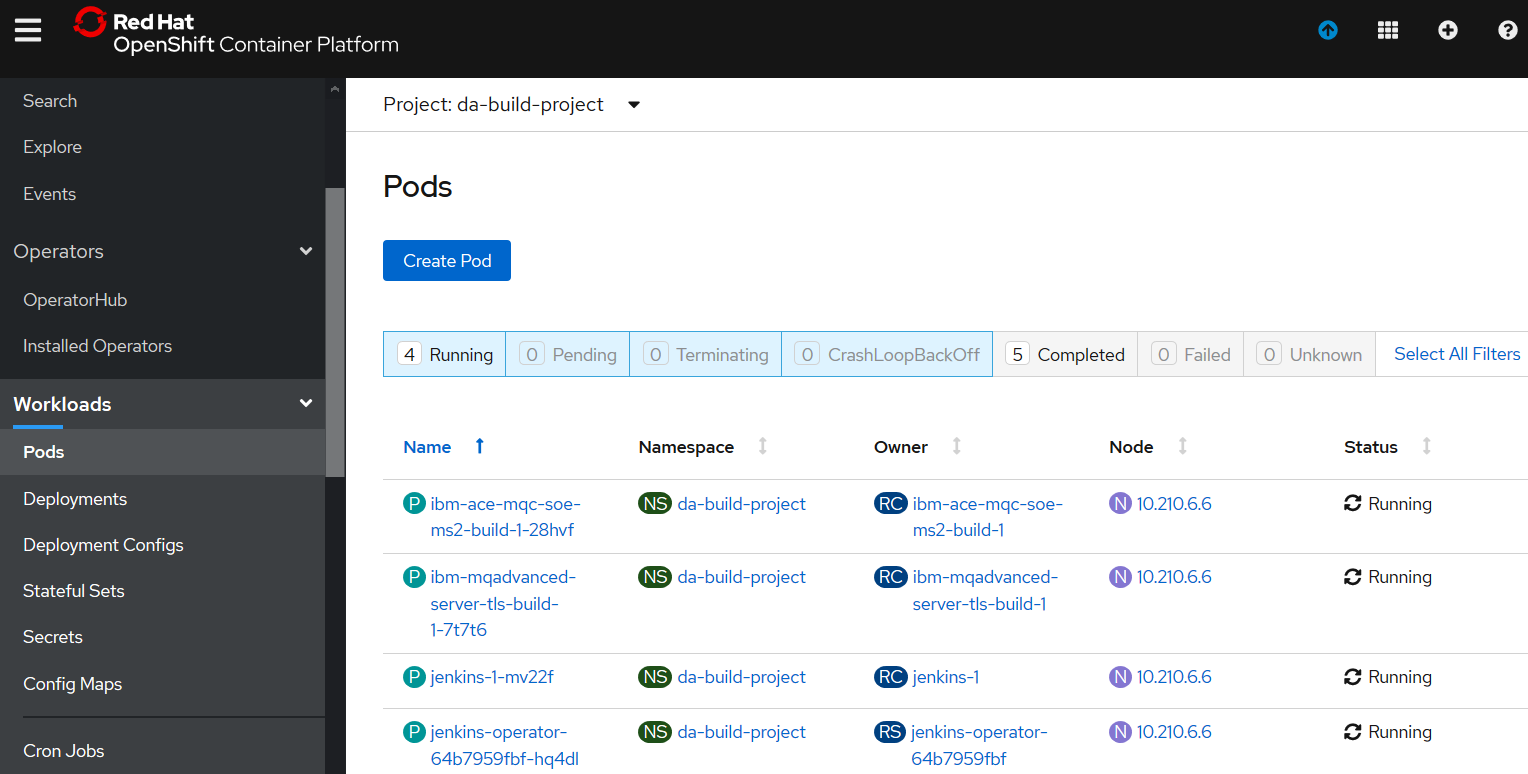


## Create the Jenkins Application Instance

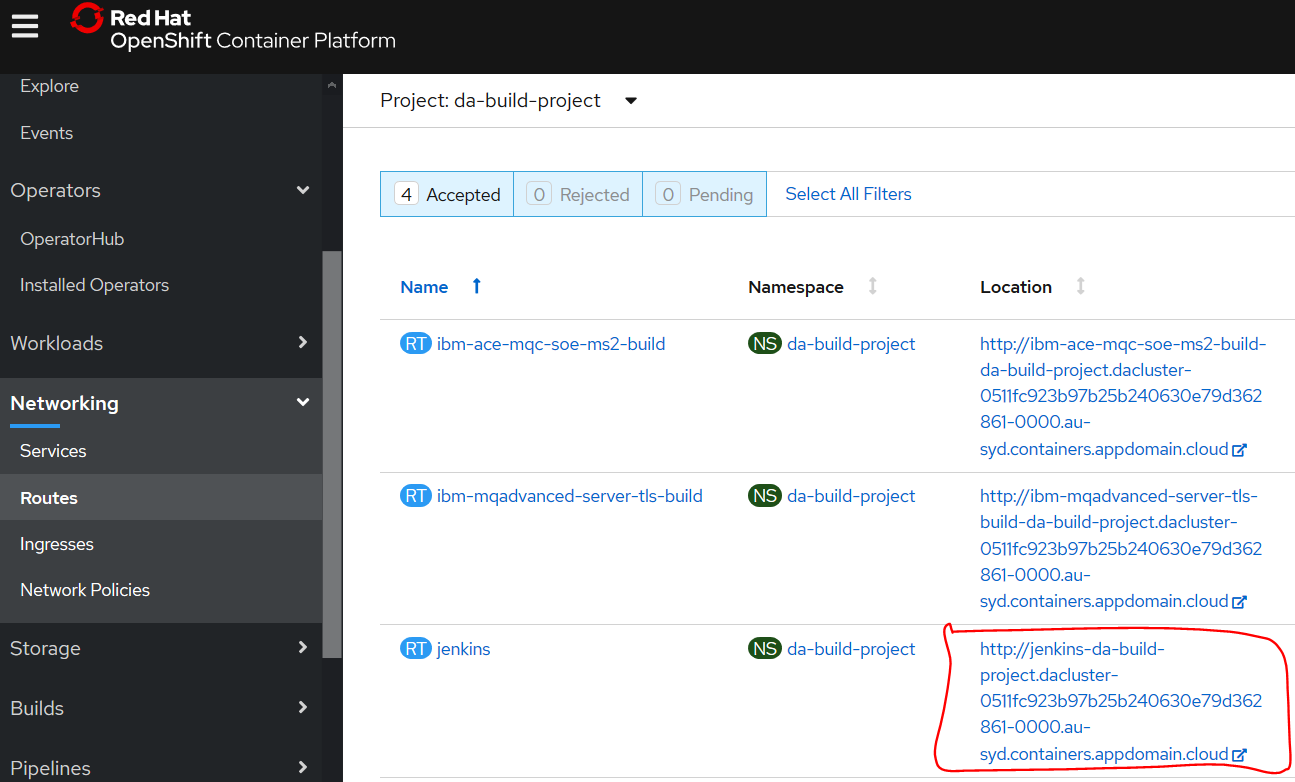
### C:\openshift\oc new-app Jenkins



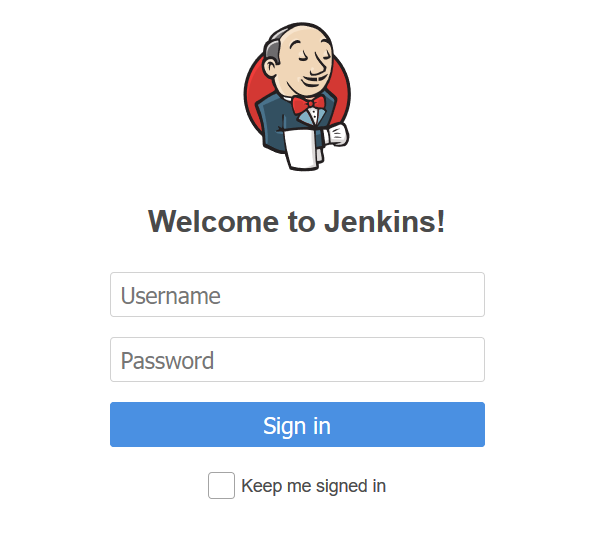
### C:\openshift\oc expose svc/jenkins



## Access the Jenkins Console

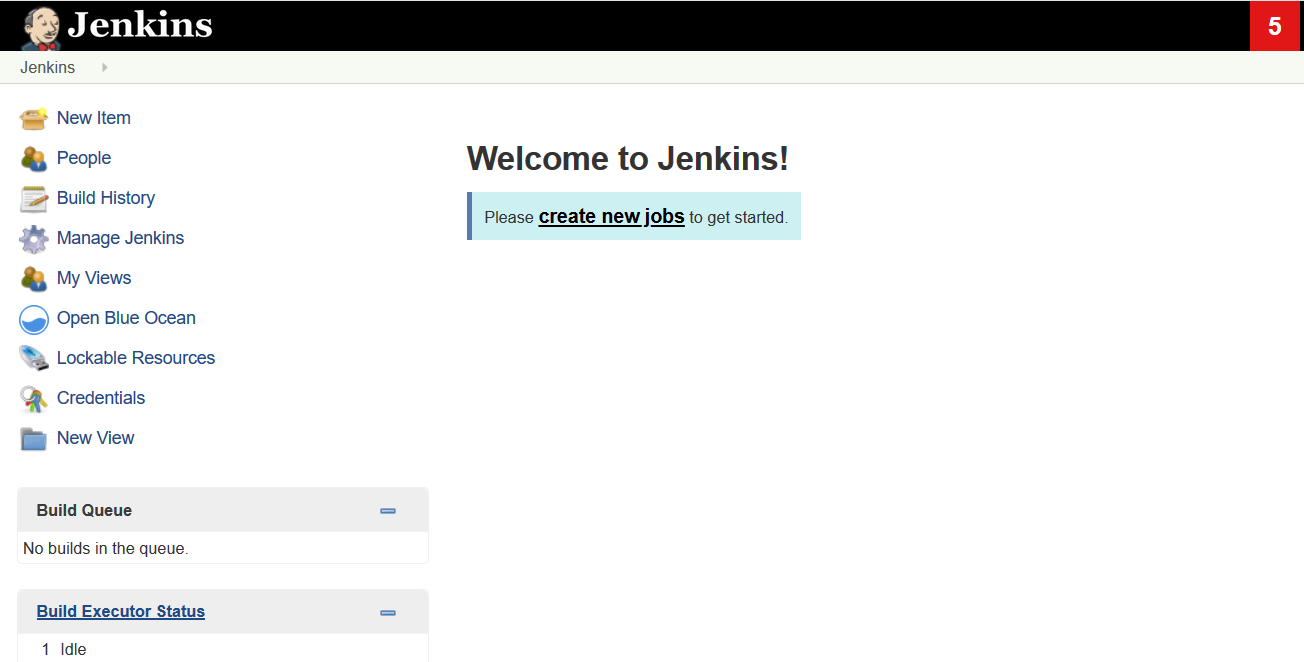


<http://jenkins-da-build-project.dacluster-0511fc923b97b25b240630e79d362861-0000.au-syd.containers.appdomain.cloud/login?from=%2F>



Username:Admin

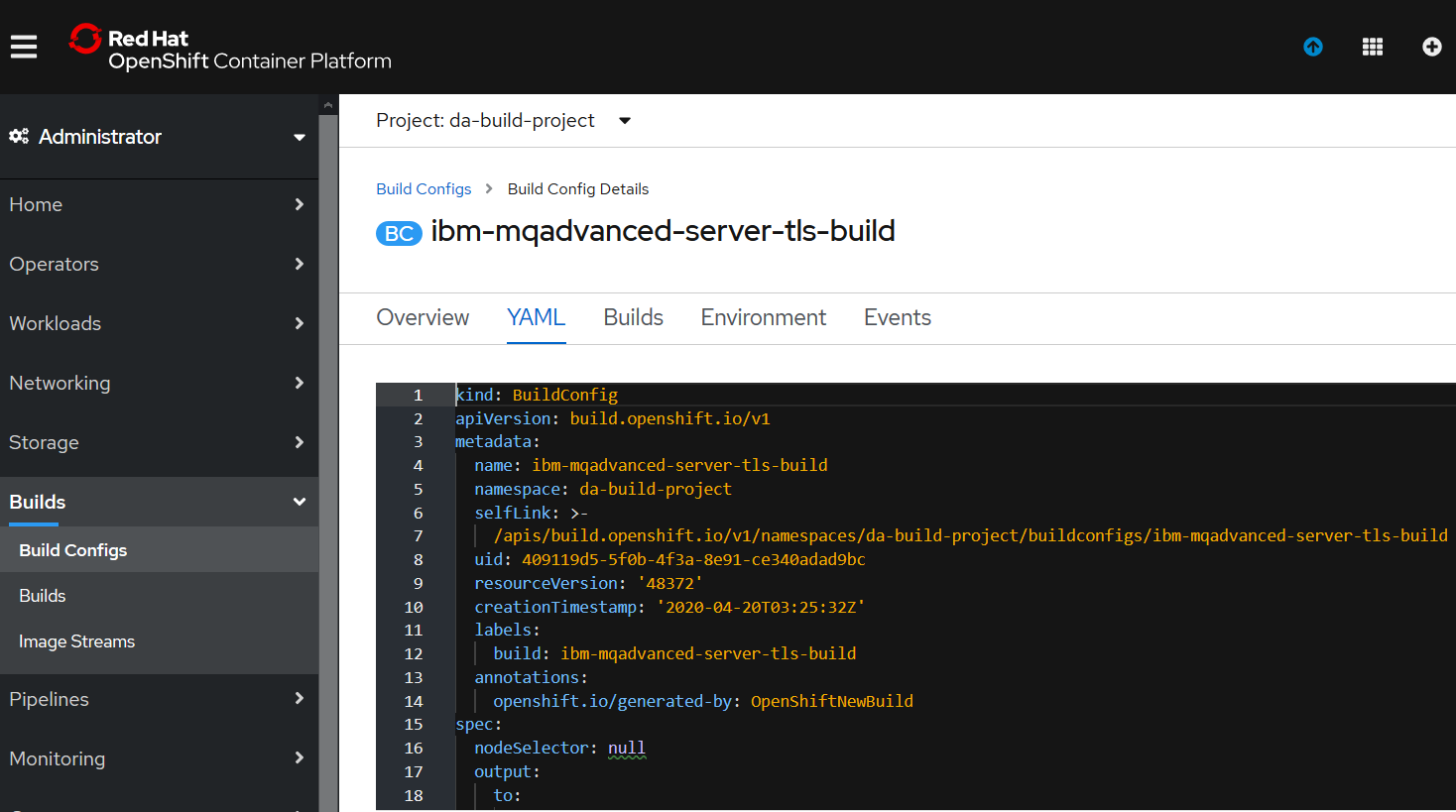
Password:password



## Create Docker build configuration

Earlier in order to build a docker image on OpenShift platform we created a BuildConfig.

RH Openshift Console->Builds->Build Configs->ibm-mqadvanced-server-tls-build



We’ll use this as a basis to create a new Build config specifically for Jenkins with MQ

Create a new file named mqtls-jen-bc.yaml with the following content:

kind: BuildConfig

apiVersion: build.openshift.io/v1

metadata:

name: mqtls-jen-bc

namespace: da-build-project

labels:

build: mqtls-jen-bc

annotations:

description: Defines how to build the messaging application

spec:

output:

to:

kind: ImageStreamTag

name: 'ibm-mqadvanced-server-tls-build:latest'

strategy:

type: Docker

dockerStrategy:

from:

kind: ImageStreamTag

namespace: da-build-project

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

source:

type: Git

dockerfile: |-

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

USER mqm

COPY TLSPRQM1.mqsc /etc/mqm/

COPY NOTLS.mqsc /etc/mqm/

COPY MYDEFS.mqsc /etc/mqm

COPY TLSPRQM1.kdb /etc/mqm/

COPY TLSPRQM1.crt /etc/mqm/

COPY TLSPRQM1.rdb /etc/mqm/

COPY TLSPRQM1.sth /etc/mqm/

git:

uri: 'https://github.com/DAVEXACOM/ibm-mqadvanced-server-tls-build.git'

resources: {}

successfulBuildsHistoryLimit: 5

failedBuildsHistoryLimit: 5

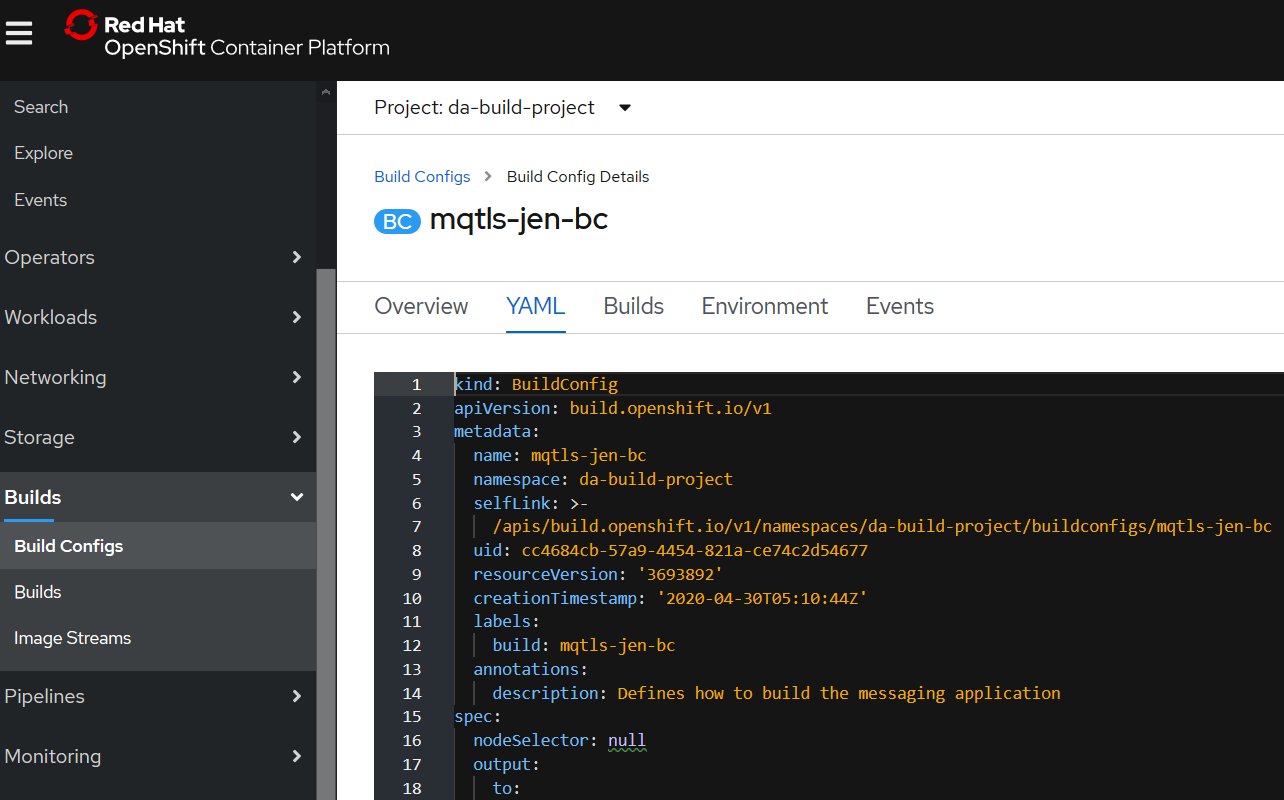
postCommit: {}

triggers: []

### c:\openshift\oc create -f mqtls-jen-bc.yaml



### Review RH OpenShift Console-> Builds-> Build Configs->mqtls-jen-bc



## Create Jenkins Pipeline

Create a basicpipeline workflow that will first build an MQ image and then deploy it.

So, the pipeline consists of just two steps.

We will consume mqtls-jen-bc BuildConfig to trigger the MQ image build.

Create a build configuration file named mqtls-jen-pipeline.yaml with the following content:

apiVersion: build.openshift.io/v1

kind: BuildConfig

metadata:

labels:

app: mqtls-jen

name: mqtls-jen-pipeline

spec:

failedBuildsHistoryLimit: 5

nodeSelector: {}

output: {}

postCommit: {}

resources: {}

runPolicy: Serial

source:

type: None

strategy:

jenkinsPipelineStrategy:

jenkinsfile: |-

pipeline {

agent {

node {

label 'nodejs'

}

}

options {

timeout(time: 20, unit: 'MINUTES')

}

stages {

stage('build') {

steps {

script {

openshift.withCluster() {

openshift.withProject() {

echo "Using project: ${openshift.project()}"

openshift.selector("bc", "mqtls-jen-bc").startBuild("--wait")

}

}

}

}

}

stage('deploy') {

when {

expression {

openshift.withCluster() {

return !openshift.selector('dc', 'mqtls-jen').exists()

}

}

}

steps {

script {

openshift.withCluster() {

openshift.withProject() {

def created = openshift.newApp( " ibm-mqadvanced-server-tls-build:latest", "--name=mqtls-jen", "--env=LICENSE=accept", "--env=MQ\_QMGR\_NAME=myjenqmgr")

echo "new-app created ${created.count()} objects named: ${created.names()}"

created.narrow('svc').expose("--name=mqtls-listener","--port=1414")

}

}

}

}

}

}

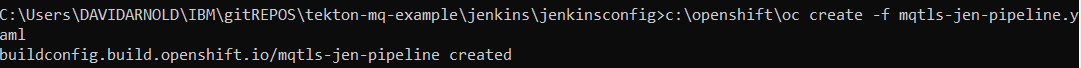
}

type: JenkinsPipeline

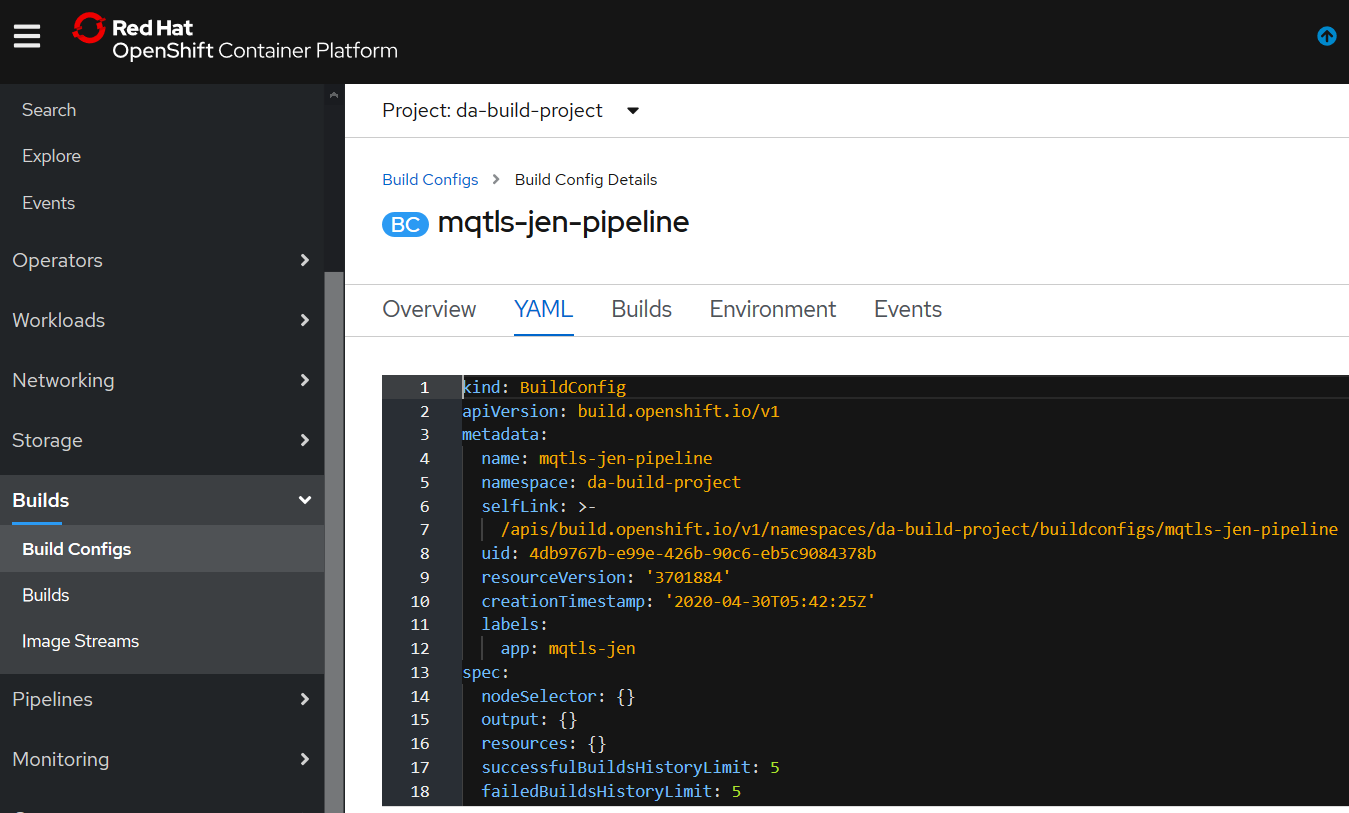
successfulBuildsHistoryLimit: 5

triggers: []

### c:\openshift\oc create -f mqtls-jen-pipeline.yaml



### Review RH OpenShift->Console->Builds->Build Configs->mqtls-jen-pipeline



## Explore the Jenkinsfile section of the mqtls-jen-pipeline BuildConfig;

### Stage 0: Declare Jenkins Agent

We will declare Jenkins agent (also known as slave) in which the pipeline will be executing. OpenShift provides three images that are suitable for use as Jenkins agents: **Base**, **Maven**, and **Node.js** images. We will be using Node.js agent to execute our pipeline.

agent {

**node** {

label 'nodejs'

}

}

### Stage 1: Build

At this stage, openshift.selector selects the build config named mqtls-jen-bc which we defined in the previous step. Using that build config, it initiates a build. After the build is complete, you will see an image named ace-app tagged as latest.

stage('build') {

steps {

script {

openshift.withCluster() {

openshift.withProject() {

**echo** "Using project: ${openshift.project()}"

openshift.selector("bc", "mqtls-jen").startBuild("--wait")

}

}

}

}

}

### Stage 2: Deploy

At this stage, openshift.newApp creates a new application named ace-app using the image produced at build stage. Note that this step is executed only in first time execution of pipeline. In next pipeline builds, this stage is skipped if the ace-app already exists in the namespace. But still, a deployment will be automatically done after each build is completed successfully due to the Image Change Trigger capability of Deployment Config. Meaning, after each new image build, an automated deployment will initiate. We will talk about this in the later posts.

stage('deploy') {

when {

expression {

openshift.withCluster() {

return !openshift.selector('dc', 'mqtls-jen').exists()

}

}

}

steps {

script {

openshift.withCluster() {

openshift.withProject() {

**def** created = openshift.newApp("ace-app:latest", "--name=ace-app", "--env=LICENSE=accept")

**echo** "new-app created ${created.count()} objects named: ${created.names()}"

created.narrow('svc').expose("--name=ace-console", "--port=7600")

created.narrow('svc').expose("--name=ace-traffic", "--port=7800")

}

}

}

}

}

## Test the Pipeline

Start the pipeline with the OpenShift Web Console. Navigate to the Builds → Pipeline section and click **Start Pipeline**.

Alternatively, you can start the pipeline using CLI:

**oc start-build** mqtls-jen-pipeline

Once the pipeline is started, you should see the following actions performed within your project:

* A job instance is created on the Jenkins server.
* A Node.js pod is launched, as our pipeline requires Node.js as Jenkins agent.
  + “Build” stage of the pipeline launches a build and upon completion a new image is pushed to imagestream.
  + “Deploy” stage of pipeline creates and exposes an application using the image produced in the previous stage. If the application already exists, the steps are skipped.
* Node.js Jenkins agent pod is deleted by default after the pipeline execution completes or is stopped.

You can visualize the pipeline execution by viewing it directly in OpenShift Web Console, choose Builds -> Builds. (or from Jenkins Web Console.)

A screenshot of a cell phone

Description automatically generated

Outcome of a successful pipeline execution is a new deployment configuration and a new pod and related service and the routes for ACE dashboard and integration service. You can access the routes using the OpenShift Web Console, choose Networking -> Routes.

A screenshot of a cell phone

Description automatically generated

Just go ahead and test your API exposed by your containerized ACE application.



# Appendix A

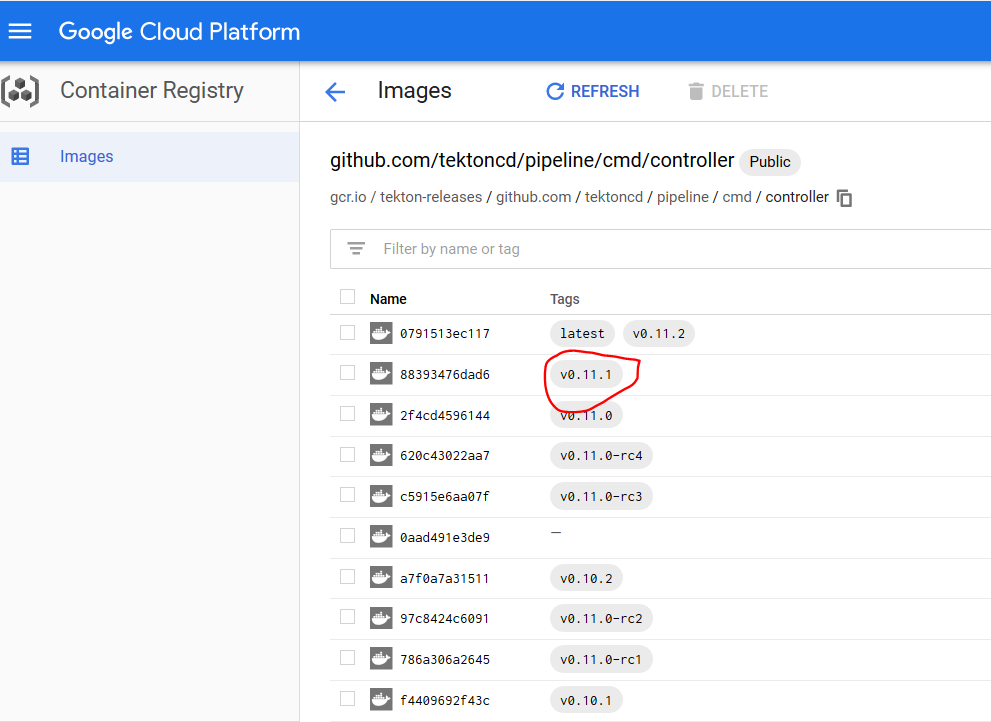
## Changing to a later version of Tekton

### Tekton Controller

Changing from v0.10.1 to v0.11.1 example

Make a back up of the YAML file in case you mess up the cut and pasting.

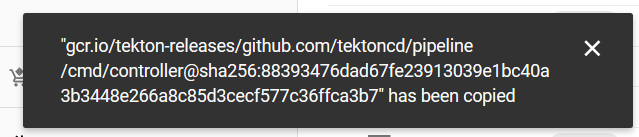
<https://console.cloud.google.com/gcr/images/tekton-releases/GLOBAL/github.com/tektoncd/pipeline/cmd/controller?gcrImageListsize=30>



Hover mouse over the gap



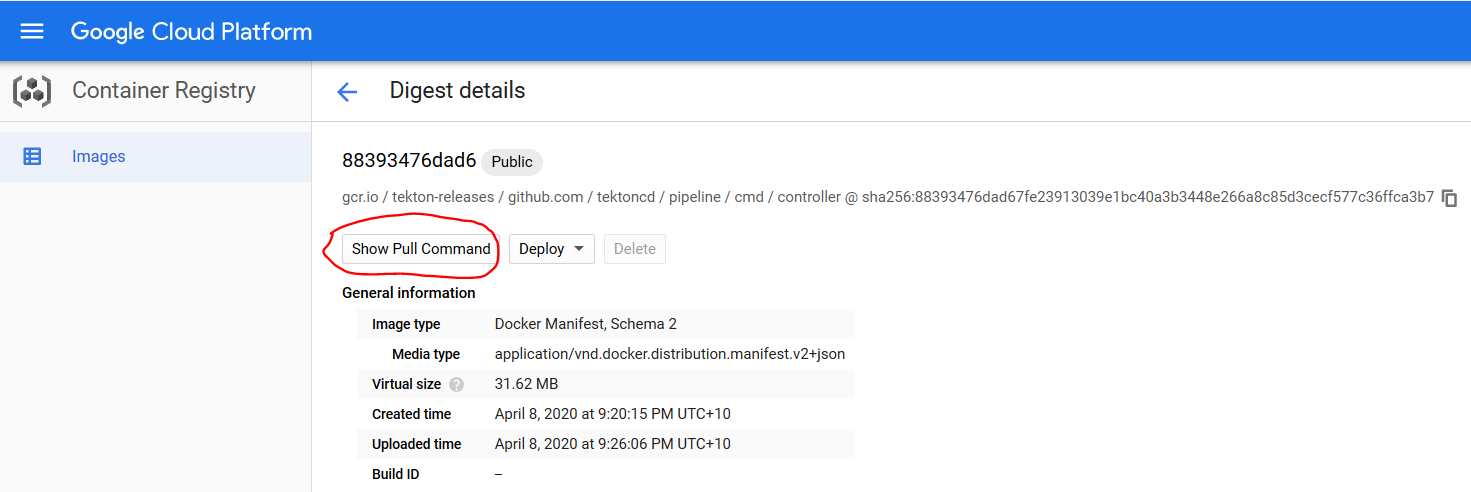
And you will be offered the option to copy the full image name



Or you can click on the



Then Click on show pull command

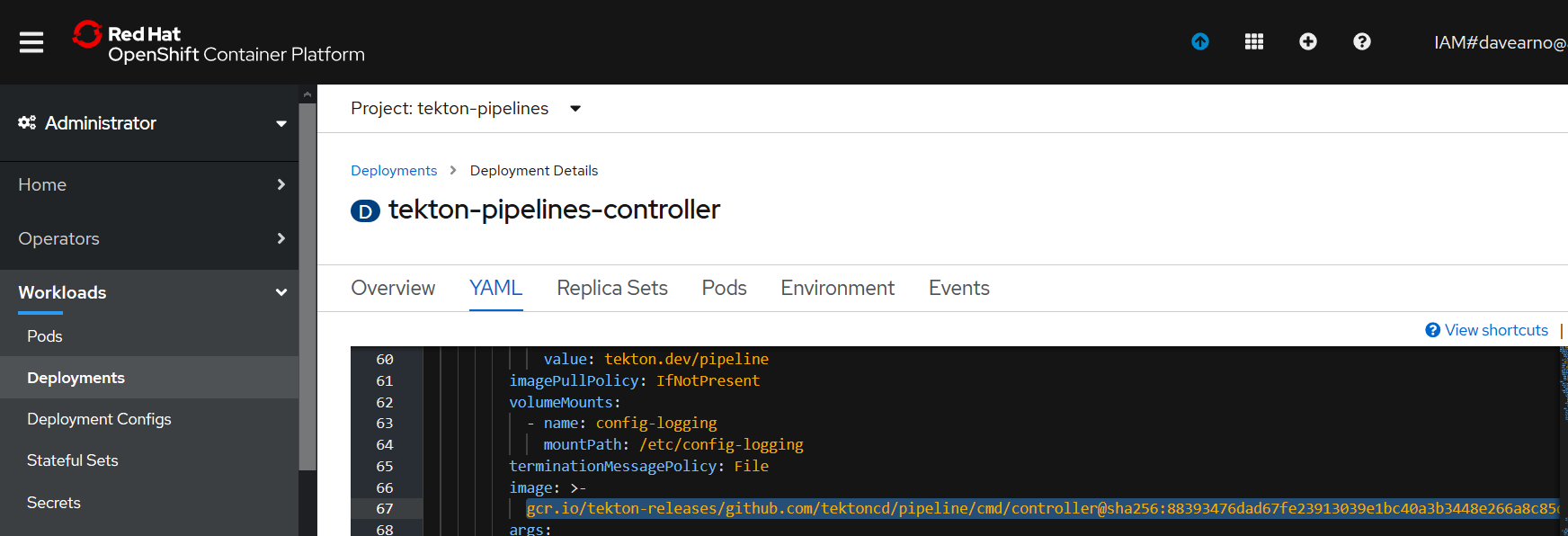


And copy the path to the image digest from there



Paste over the image name in the RH openshift Console->Workloads->deployments->tekton-pipelines-controller YAML

gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/controller@sha256:88393476dad67fe23913039e1bc40a3b3448e266a8c85d3cecf577c36ffca3b7



Repeat the process for the other tekton images referenced in the YAML file

**kubeconfig-writer-image**

gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/kubeconfigwriter@sha256:206c4e5de37d13c34f9538f87096db16433aadba264f24e9995cbb6b66fb67de

**creds-image**

gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/creds-init@sha256:959b0d9a2d43d35e15a85460cc86567d308f467ee8ec16dbd9b32f51ce75d582

**git-image**

gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/git-init@sha256:18ffa2bfc14b1fa6d39f62271beacf6bbc38e7cd2e255184dec477c2936270bc

**entrypoint-image**

gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/entrypoint@sha256:cf0e81477c45dca0df6253e3239f6f0603700641292bf207503a7b267dc4c916

imagedigest-exporter-image

gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/imagedigestexporter@sha256:1cf3f27f3ff7c73782d8a65853e8fc7f0d4aafc6443893e0150bdbe614a9169d

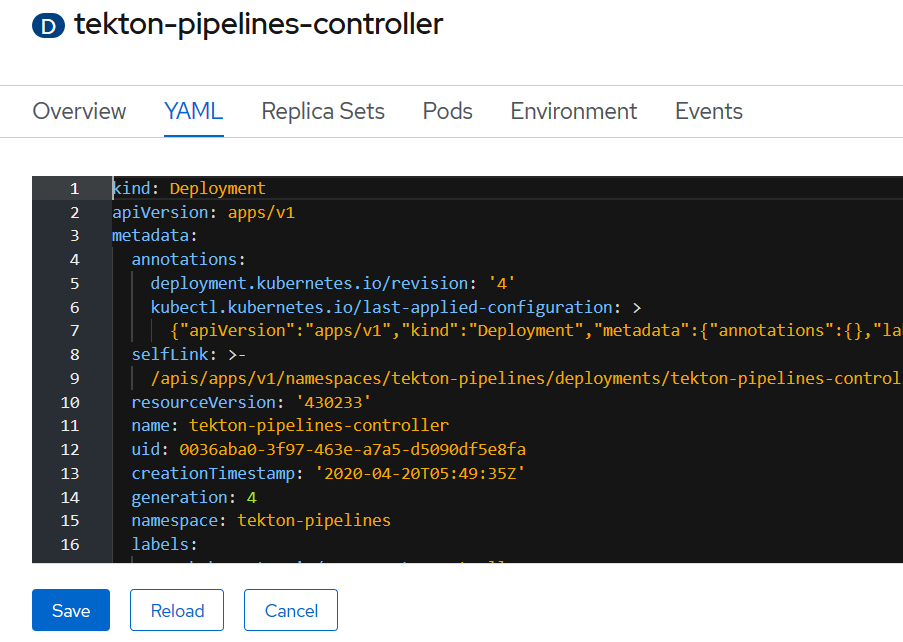
**pr-image**

gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/pullrequest-init@sha256:3783254c379b286dd0987674160d3e19a95be1ccf0985788d6dcc0f159199095

**build-gcs-fetcher-image**

gcr.io/tekton-releases/github.com/tektoncd/pipeline/vendor/github.com/googlecloudplatform/cloud-builders/gcs-fetcher/cmd/gcs-fetcher@sha256:70f8d32a572496169d451130541541cbc99434932fd28beea486189af8a2995a

Save and reload



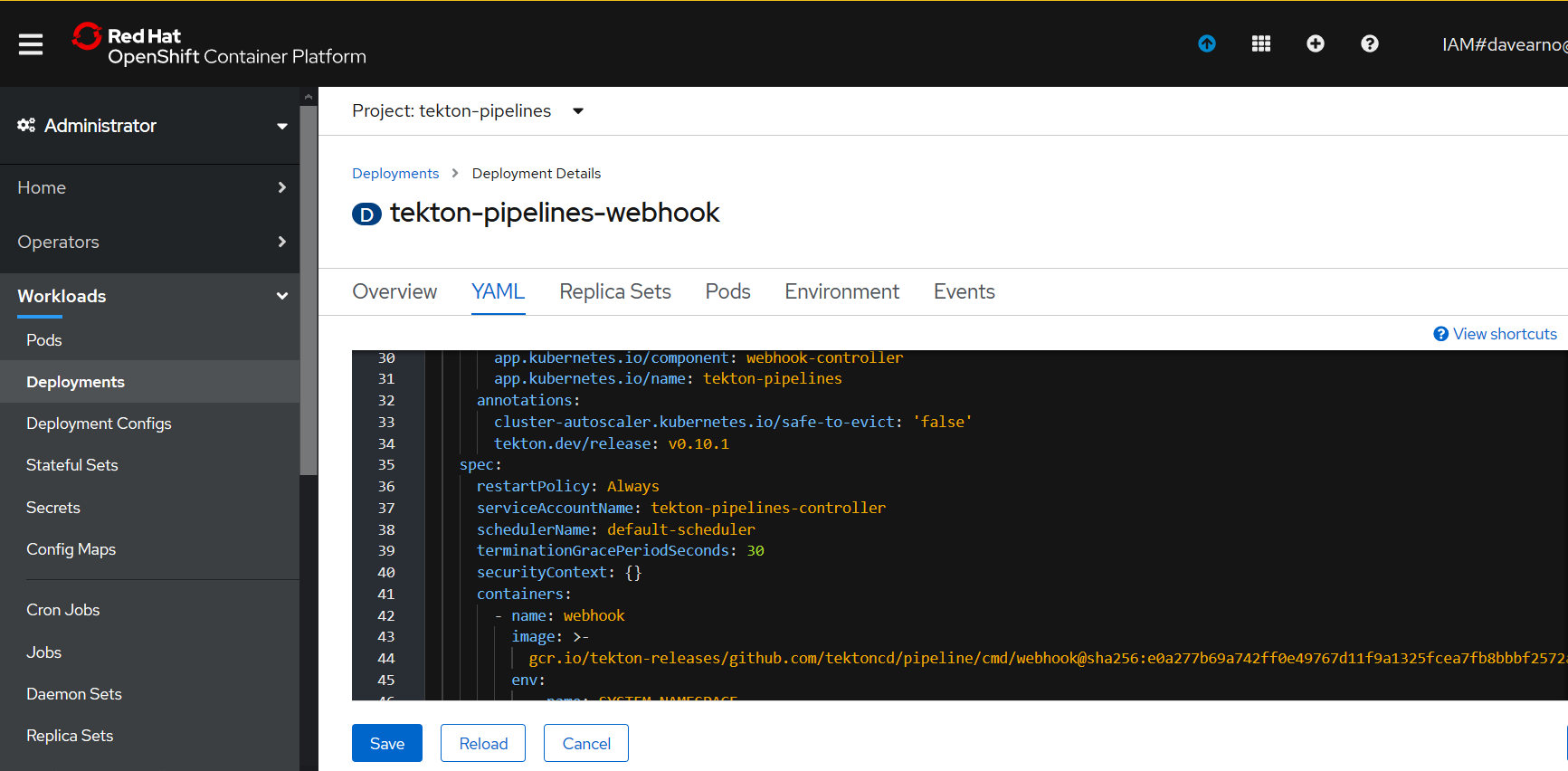
Check the controller pod terminates and a new pod starts.

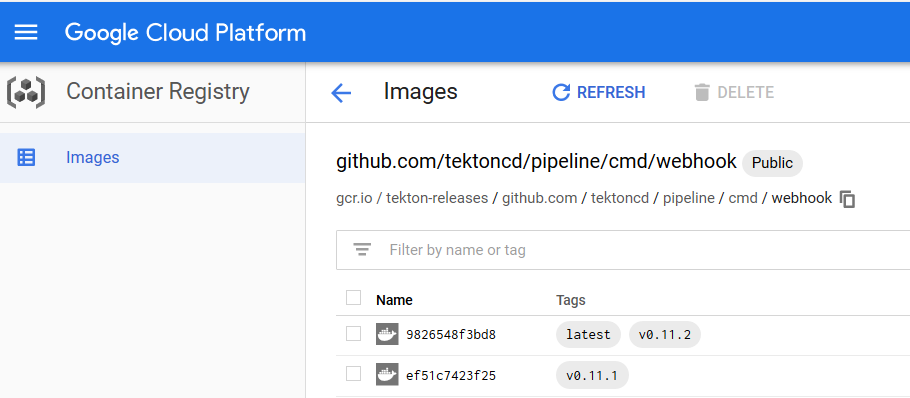


### Tekton Webhook

### NOTE: Leave the Webhook deployment alone i.e. v0.10.1 . I could not get later versions of the image to load. Instruction here just for completeness.

Repeat for tekton-pipelines-webhook



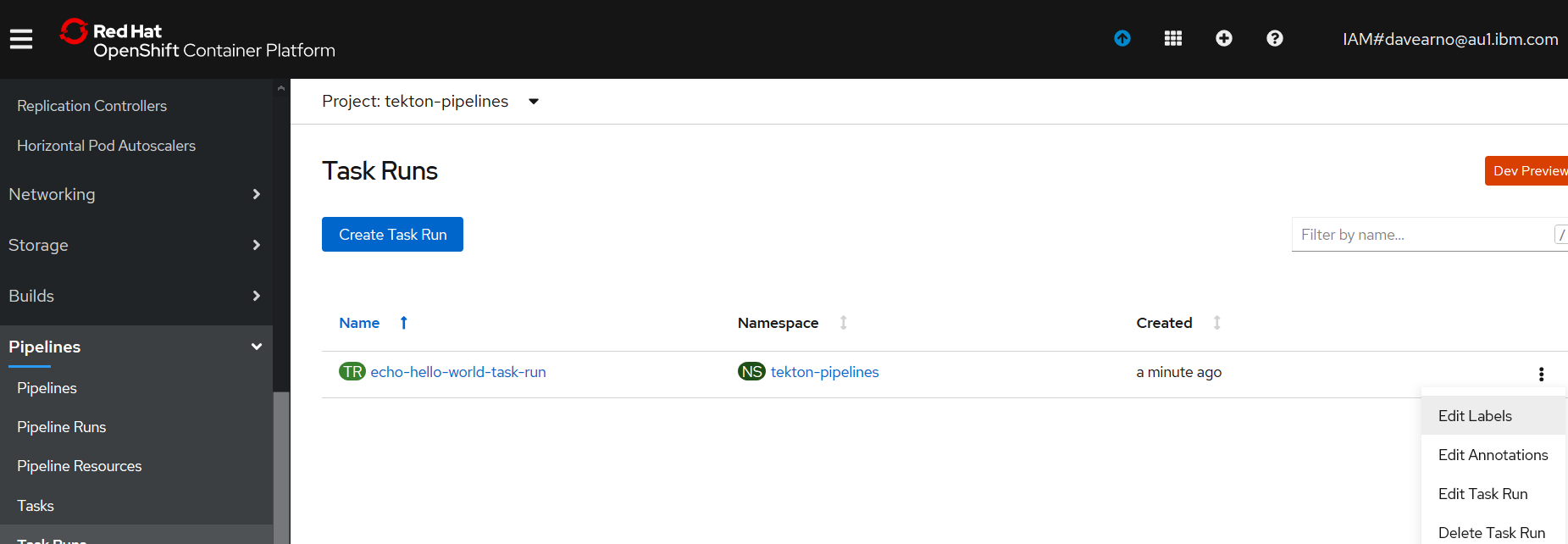


[gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/webhook@sha256:ef51c7423f25b257ee63a4a5fb62c9df4721cabb5e34a516e03fb3f3b74f8c8f](mailto:gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/webhook@sha256:ef51c7423f25b257ee63a4a5fb62c9df4721cabb5e34a516e03fb3f3b74f8c8f)

Save and reload.

### Test the Tekton Update

delete the Task run



**C:\openshift>oc apply -f Tekton-ivt\echo-hello-world-task-run.yaml**

taskrun.tekton.dev/echo-hello-world-task-run created

**C:\openshift>oc get taskruns/echo-hello-world-task-run -o yaml**