IBM® Content Manager V8.7

Running the resource manager application in containers

Special notice

The information contained in this publication was derived under specific operating and environmental conditions and is distributed on an ‘as is’ basis without any warranty either expressed or implied. IBM specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. IBM assumes no responsibility for any errors that may appear in this document. The use of this information or the implementation of any of these techniques are the responsibility of the user and depends on the user's ability to evaluate and integrate them into the user's operational environment. While each item might have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Users attempting to adapt these techniques to their own environments do so at their own risk.

The information contained in this document is subject to change without any notice. IBM reserves the right to make any such changes without obligation to notify any person of such revision or changes. IBM makes no commitment to keep the information contained herein up to date.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's program may be used. Any functionally equivalent program may be used instead. Equivalent product, program, or services that do not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service.

The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent application covering the subject matter in this document. The furnishing of this document does not give you license to these patents. Any information about non-IBM (‘vendor’) products in this document has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness.

IBM, the IBM logo, ibm.com, AIX, Db2, Tivoli, and WebSphere are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at ‘Copyright and trademark information‘ at www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Contents

1 Introduction

You can build and run a container image for the IBM Content Manager Enterprise Edition resource manager application component. You can run the image in a standalone container or a container cluster.

IBM WebSphere Liberty must be included in the image and used to run the resource manager application. You cannot use IBM WebSphere Application Server.

Building a resource manager application container image and running it in a container automates the process of deploying the resource manager application and running it in WebSphere Liberty. The basic procedure to deploy and run a resource manager application in WebSphere Liberty also applies when it is run in a container.

For more information about running the resource manager application in WebSphere Liberty, see Running the resource manager application in IBM WebSphere Liberty.

This feature is supported in IBM Content Manager Version 8.7 fix pack 4 and later.

About this document

The commands and examples in this document are based on Red Hat Enterprise Linux and use Docker as the container tool. Change the commands and examples to make them work in your environment.

1.1 Prerequisites

This document assumes that you know how to perform the following tasks:

- Install a container tool such as Docker.
- Pull an image from a container registry such as Docker hub.
- Build an image based on a Dockerfile.
- Run an image in a standalone container or in a container cluster.

For more information about the software that you need, see www.ibm.com.

2 Building a resource manager application container image

1 Install and configure a container tool such as Docker by following the manufacturer's instructions.

2 Build an image based on an existing container image, which includes IBM WebSphere Liberty and Java.

This document uses an example that builds the resource manager application image on top of the WebSphere Liberty multi-architecture image based on UBI 8, which is available on Docker hub (https://hub.docker.com/r/ibmcom/websphere-liberty). You can pull an image from Docker hub. For example:

```
docker pull ibmcom/websphere-liberty:full-java8-ibmjava-ubi
```

Support for running resource manager application on WebSphere Liberty with IBM Semeru java was added in Content Manager version 8.7 fix pack 5. Details: https://www.ibm.com/docs/en/content-manager/8.7.0?topic=rrmaiwl-running-resource-manager-application-in-websphere-liberty-semeru-java

2.1 Building an image with basic functions

To build an image for the resource manager application, you need a working directory where you can put all the files that are required for the image.

1 Create the working directory.

This document uses a working directory that is called `/root/rmappDocker`.

2 Add the following files to the working directory:

`icmrm-wlp.war`

Copy the `icmrm-wlp.war` file from the `<IBMCMROOT>/libertyApps/` directory.

This file is created when you configure the "Resource manager application for WebSphere Liberty" component in IBM Content Manager.

`icmrm.xml`

The main WebSphere Liberty configuration file for the resource manager application. For more information, see the IBM Content Manager documentation.

`authData.xml`

Contains information that is used to connect to the resource manager database and the library server database. For more information, see the IBM Content Manager documentation.

`bootstrap.properties`

Sets values that can be referenced from the `authData.xml` file and the `icmrm.xml` file. For more information, see the IBM Content Manager documentation.

`keystore.xml`

Configures a default keystore. This file is required only if you want to set a custom password to the default keystore. For more information, see the IBM Content Manager documentation.

`db2jcc4.jar` and `db2jcc_license_cisuz.jar`

Db2 JDBC driver jar files. These files are required only if your resource manager database and library server database are on Db2.

`ojdbc8.jar`

Oracle JDBC driver jar file. This file is required only if your resource manager database and library server database are on Oracle.

Dockerfile

A file with commands that need to be run to build the image. For information about the format of this file and how it is used, see the Docker documentation.

```
FROM ibmcom/websphere-liberty:full-java8-ibmjava-ubi
ENV LICENSE=accept
# Use the "default" user to run the commands
USER 1001
RUN mkdir -p /opt/ibm/wlp/usr/servers/defaultServer/lib
RUN mkdir -p /opt/ibm/wlp/usr/servers/defaultServer/configDropins
RUN mkdir -p
/opt/ibm/wlp/usr/servers/defaultServer/configDropins/defaults
RUN mkdir -p
/opt/ibm/wlp/usr/servers/defaultServer/configDropins/overrides
COPY --chown=1001:0 icmrm-wlp.war
/opt/ibm/wlp/usr/servers/defaultServer/apps/
# Copy the Oracle JDBC driver jar file. This is required only for Oracle
COPY --chown=1001:0 ojdbc8.jar
/opt/ibm/wlp/usr/servers/defaultServer/lib/
```

```
# Copy the Db2 JDBC driver jar file. This is required only for Db2
COPY --chown=1001:0 db2jcc4.jar
/opt/ibm/wlp/usr/servers/defaultServer/lib/
COPY --chown=1001:0 db2jcc_license_cisuz.jar
/opt/ibm/wlp/usr/servers/defaultServer/lib/
COPY --chown=1001:0 bootstrap.properties
/opt/ibm/wlp/usr/servers/defaultServer/
COPY --chown=1001:0 keystore.xml
/opt/ibm/wlp/usr/servers/defaultServer/configDropins/defaults/
COPY --chown=1001:0 icmrm.xml
/opt/ibm/wlp/usr/servers/defaultServer/configDropins/overrides/
COPY --chown=1001:0 authData.xml
/opt/ibm/wlp/usr/servers/defaultServer/configDropins/overrides/
```

3 To build the image, run the following command:

```
docker build -f <docker_file> -t <output_image> <working_directory>
```

where:

<docker_file> is the location of the Dockerfile.

<output_image> is the output image and tag.

<working_directory> is the working directory in which you put the files that Docker uses to build the image.

For example:

```
docker build -f /root/rmappDocker/Dockerfile -t icmrm-app:8.7.00.400
/root/rmappDocker
```

2.2 Installing and configuring the IBM Storage Protect client API

If your resource manager application needs to access IBM Storage Protect (formerly known as IBM Spectrum Protect or TSM), you must install the Storage Protect client API in your container image. Add configuration files to the image, and then add the Storage Protect server certificate to the Storage Protect client keystore.

The working directory must contain the following files:

`8.1.22.0-TIV-TSMBAC-LinuxX86.tar`

Storage Protect client API installation file. “8.1.22.0” is the version of client API; the file name might be different if you install a different version of the client API.

`cert256.arm`

Storage Protect server certificate file. You can get this file from the Storage Protect server.

`dsm.sys`

Storage Protect client configuration file.

`icmrm.opt`

Options file for Storage Protect. Set the path to this file when you create a server definition in the resource manager in IBM Content Manager system administration client.

Complete the following steps to install and configure the Storage Protect client API in the resource manager application image. Add the commands in the Dockerfile and specify to run them by using `root`:

1 Add the file to your image, and then run the installation commands by using `root`:

```
USER 0
ADD 8.1.22.0-TIV-TSMBAC-LinuxX86.tar /tmp/TIV-TSMBAC/
RUN rpm -U /tmp/TIV-TSMBAC/gskcrypt64-8.0.55.31.linux.x86_64.rpm
/tmp/TIV-TSMBAC/gskssl64-8.0.55.31.linux.x86_64.rpm
RUN rpm -i /tmp/TIV-TSMBAC/TIVsm-API64.x86_64.rpm
# install the BA client which is required to run the dsmcert command
RUN rpm -i /tmp/TIV-TSMBAC/TIVsm-BA.x86_64.rpm
```

Change the version information in the file names to match your version of Storage Protect client API:

`8.1.22.0-TIV-TSMBAC-LinuxX86.tar`
`gskcrypt64-8.0.55.31.linux.x86_64.rpm`
`gskssl64-8.0.55.31.linux.x86_64.rpm`

If you are building your image on top of a UBI minimal image with a tag that ends with "ubi-minimal", you must install the dependent library libxcrypt-compat before you install the Storage Protect client API rpms:

```
RUN microdnf install -y libxcrypt-compat
```

2 Copy the `dsm.sys` and `icmrm.opt` files to your image, for example:

```
COPY dsm.sys /opt/tivoli/tsm/client/api/bin64/
COPY icmrm.opt /opt/tivoli/tsm/client/api/bin64/
```

3 Copy the server certificate file to the image, for example:

```
COPY cert256.arm /tmp/
```

4 Add the certificate to your Storage Protect client keystore by running the `dsmcert` utility, for example:

```
RUN /opt/tivoli/tsm/client/ba/bin/dsmcert -add -server SPServer1 -file
/tmp/cert256.arm
```

2.3 Adding cloud object storage (COS) server certificates to the keystore

If you store objects on cloud object storage servers and use HTTPS to connect to those servers, you must add the CA certificates of the servers to the resource manager application keystore as trusted certificates. There are two ways to do this:

Using the default keystore generated by WebSphere Liberty server

If you use the default keystore generated by WebSphere Liberty server, add the CA certificates of the COS servers to your working directory before building your image. In your Dockerfile, create the keystore, and then add the COS server CA certificate as a trusted certificate.

The following example Dockerfile commands add `aws_s3.arm`, which is the CA certificate for Amazon S3, to the default keystore.

1 Copy the CA certificate file into the image:

```
COPY --chown=1001:0 aws_s3.arm /tmp/
```

2 Create the default keystore and self-signed certificates:

```
RUN /liberty/bin/securityUtility createSSLCertificate
--server=defaultServer --password=changeit --validity=3650
```

Use the same password as the one that is in the `keystore.xml` file.

3 Add the COS CA certificate to the keystore:

```
RUN /opt/ibm/java/jre/bin/keytool -importcert -trustcacerts -file
/tmp/aws_s3.arm -alias aws_s3 -keystore
/liberty/output/defaultServer/resources/security/key.p12 -storepass
changeit -noprompt
```

Using a custom keystore

If you use a custom keystore, complete the following steps:

1 Create the keystore outside of the container.

2 Add the COS server signer certificate into the keystore.

3 Copy the keystore to your image.

4 Build the image.

5 In the `icmrm.xml` file, you must specify the keystore file location in the container.

For more information, see Using a custom keystore and trust store for the resource manager application SSL communication.

3 Running the resource manager application image

You can run the resource manager application image in a standalone container or a container cluster, such as Red Hat OpenShift or any other Kubernetes clusters.

- To run the image in a *standalone container*, see the documentation for your container. If you use Docker, you can use the `docker run` command.
- To run the image in a *container cluster*, see the documentation for your cluster for information about how to deploy the image and run it.

The resource manager application that runs in the image must be at the same IBM Content Manager level as the resource manager database to which it connects.

To move a resource manager application from a WebSphere Application Server profile to a container, stop the resource manager application in WebSphere Application Server profile before you start the one in container. When you have validated the new resource manager application in container, you can delete the old one from WebSphere Application Server. For more information, see Removing the old resource manager application from WebSphere Application Server.

Do not run both of the applications at the same time which connect to a shared resource manager database.

3.1 Exposing ports

When the resource manager application image is run in a container or a container cluster, you must expose the HTTP and HTTPS ports that are configured for the WebSphere Liberty server.

You have to expose only the ports that are appropriate for your system. If you want to use HTTPS transportation only, expose only the HTTPS port; you do not have to expose the HTTP port.

If only the HTTPS port is exposed or your resource manager application is made accessible by HTTPS only in a load balancer, route or proxy, you must configure other IBM Content Manager components to access the resource manager only through HTTPS.

If you use ECM dashboard, expose the 32775 port.

Exposing the ports for a standalone container

If the image is in a standalone container, use the `docker run` command with the `-p` option, for example:

```
docker run -d -p 9080:9080 -p 9443:9443 -p 32775:32775 icmrm-app:8.7.00.400
```

Exposing the ports for a cluster container

If the image is in a Kubernetes cluster like OpenShift, you can expose the ports in the service and deployment definition. For more information, see the cluster documentation.

The following example can be part of a Kubernetes cluster service definition:

```
spec:
  ports:
    - name: http
      protocol: TCP
      port: 9080
      nodePort: 30080
      targetPort: 9080
    - name: https
      protocol: TCP
      port: 9443
      nodePort: 30443
      targetPort: 9443
    - name: listener
      protocol: TCP
      port: 32775
      nodePort: 32767
      targetPort: 32775
```

3.2 Using external storage to store persistent data

When the resource manager application runs in a container, data that is written to internal local file systems is not persisted; it is lost when the container is stopped. If you want to store persistent data, you must mount external file system volumes. The persisted data includes the following:

- user objects that are stored in file system volumes (lbosdata)
- cached objects that are stored in the staging area
- log files that are written by WebSphere Liberty and the resource manager application
- validation utility report files, which are in XML format.

Mounting external volumes for a standalone container

If the image is run in a standalone container, use the `docker run` command with the `-v` option. For example:

```
docker run -d -p 9080:9080 -p 9443:9443 -p 32775:32775 \
-v /mnt/RM_volume1:/mnt/RM_volume1 \
-v /mnt/RM_volume2:/mnt/RM_volume2 \
-v /mnt/RM_volume3/staging:/staging \
-v /mnt/RM_volume3/logs:/logs \
-v /mnt/RM_volume3/vuReport:/vuReport \
icmrm-app:8.7.00.400
```

Mounting external volumes for a container cluster

If the image is run in a container cluster, configure the external volumes in the environment, and then make them accessible to the resource manager application.

For example, if you use OpenShift Cloud Platform(OCP), you can create PersistentVolume and PersistentVolumeClaim, and then define volumeMounts and volumes in a Deployment.

Mounting directories as volumes

If you mount a directory as a volume in containers:

- You can use the directories as a staging area or to store log files and validation utility report files.
- Do not use the directories as “file system volumes” in resource manager storage systems. You must use whole physical file system volumes in resource manager storage systems.

Permissions for the user that starts WebSphere Liberty

Make sure that the user that is used to start WebSphere Liberty in the container has the following permissions:

- For files, the user must have permissions to read and write.
- For directories, the user must have read, write, *and* execute permissions, so that new files and subdirectories can be created in the directory.

In the example of the UBI image, the default user that is used to start WebSphere Liberty is “default”, and its group name is “root”.

The user that starts WebSphere Liberty in a container must be the same as the ICMRM_USER property value in the RMCONFIGURATION table in resource manager database. This ensures that resource manager application can access the files and directories. If you want to start the resource manager application by using a different user and ensure the permissions yourself, set RMCONFIGURATION property WLP_ENFORCE_ICMRM_USER to “false” by using the resource manager administration console.

Container cluster

If the image is run in a container cluster, the cluster members must share the same file systems for storage: file system volumes (lbosdata), staging area, log file location and validation utility report location. To avoid conflicts, specify a log file name that includes the hostname; for example:

```
icmrm.logfile@${sys:hostname}
```

The cluster members must have different hostnames (see the cluster documentation).

In a typical Kubernetes cluster, a new container comes with a new hostname. If you have the hostname in the log file name, a new resource manager application log file is generated. If you frequently delete and create containers in your cluster, monitor your log file storage, and then delete old log files so that you do not run out of disk space.

3.3 Synchronizing SMS and background service state changes

When the image is run in a container cluster, storage management system (SMS) changes and background service state changes can take up to 60 seconds to synchronize to all cluster members.

For example, when you change configuration of a volume by using the IBM Content Manager system administration client, the change takes effect immediately at one of the cluster members. The change is recorded in the resource manager database. All the cluster members periodically query the resource manager database for changes.

The default querying interval is 60 seconds, which is sufficient in most systems. Use a different value only if more frequent checks are required.

To change the querying interval

1 Set the RMCONFIGURATION property WLP_UUID_BACKGROUND_SERVICE_CYCLE by using the resource manager administration console.

2 Restart all cluster members.

3.4 Using passive mode for FTP server operations

If your application calls IBM Content Manager APIs to store or retrieve objects on FTP servers, you must use FTP passive mode.

To use passive mode, set the FTP_USE_PASSIVE_MODE property in RMCONFIGURATION table to `true` by using the resource manager administration console.

Make sure that your FTP server is configured to support passive mode.

If you do not use passive mode, the third party operations fail. The following error message is written to the resource manager log file:

```
Cannot bind data transfer to port 37441, return code =500
```

3.5 Requirement for request dispatching

A typical cluster environment uses a load balancer or route to dispatch client requests to endpoint cluster members. For example, in Red Hat OpenShift Container Platform (OCP), you can use Route or Ingress to dispatch client requests to pods in a cluster.

When the resource manager application is deployed in such clusters, subsequent requests must be routed to the same pod after a successful logon from the IBM Content Manager system administration client or the resource manager administration console. Otherwise, when using the IBM Content Manager system administration client and resource manager administration console, subsequent user operations fail.

In the load balancer or route configuration, do not redirect non-secure (HTTP) transports to secure (HTTPS) ones. Otherwise, data management operations like store, retrieve, and update through non-secure (HTTP) transports fail.

4 Updating the resource manager application image

There are various reasons why you might want to rebuild the resource manager application image and update it in your container or container cluster. These include:

- Upgrading the resource manager application to a higher level of IBM Content Manager.
- Upgrading the prerequisite software that is in the image: WebSphere Liberty, Java, JDBC driver library files, Storage Protect client API, and so on.
- Changing the WebSphere Liberty configuration.

Rebuilding the image

1 Add the new files to your working directory by replacing the old ones.

2 Re-run the build command to create a new image.

You can use a new tag for your image.

To update the image in your container or container cluster, refer to the container or container cluster documentation.

Updating the resource manager application

The resource manager application that is in the image must be at the same IBM Content Manager level as the resource manager database that it connects to. If you need to update your resource manager application in the container to a higher level of IBM Content Manager, complete the following steps:

1 Stop the container or container cluster.

2 Update the resource manager database to the target level.

3 Update the image of your container or container cluster to the target level

4 Start the container or container cluster.

For more information, see:
https://www.ibm.com/docs/en/content-manager/8.7.0?topic=rrmaiwl-upgrading-resource-manager-application-in-websphere-liberty-later-version-content-manager

5 Samples

Running the resource manager application in a standalone container

```
docker run -d -p 9080:9080 -p 9443:9443 -p 32775:32775 \
-v /mnt/RM_volume1:/mnt/RM_volume1 \
-v /mnt/RM_volume2:/mnt/RM_volume2 \
-v /mnt/RM_volume3/staging:/staging \
-v /mnt/RM_volume3/logs:/logs \
-v /mnt/RM_volume3/vuReport:/vuReport \
icmrm-app:8.7.00.400
```

Deploying the resource manager application in an OpenShift cluster

Sample YAML script to create persistent volumes

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: ecm-icmrm-vol1-pv
spec:
  capacity:
    storage: 4Gi
  accessModes:
  - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: ecm-icmrm-vol
  mountOptions:
    - nfsvers=3
  nfs:
    path: /mnt/RM_volume1
    server: sampleServer
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: ecm-icmrm-vol2-pv
spec:
  capacity:
    storage: 4Gi
  accessModes:
  - ReadWriteMany
```

```
  persistentVolumeReclaimPolicy: Retain
  storageClassName: ecm-icmrm-vol
  mountOptions:
    - nfsvers=3
  nfs:
    path: /mnt/RM_volume2
    server: sampleServer

---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: ecm-icmrm-files-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
  - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: ecm-icmrm-files
  mountOptions:
    - nfsvers=3
  nfs:
    path: /mnt/RM_volume3
server: sampleServer
```

Sample YAML script to create persistent volume claims

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ecm-icmrm-vol1-pvc
spec:
  storageClassName: ecm-icmrm-vol
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 4Gi
  volumeName: "ecm-icmrm-vol1-pv"
---
apiVersion: v1
kind: PersistentVolumeClaim
```

```
metadata:
  name: ecm-icmrm-vol2-pvc
spec:
  storageClassName: ecm-icmrm-vol
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 4Gi
  volumeName: "ecm-icmrm-vol2-pv"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ecm-icmrm-files-pvc
spec:
  storageClassName: ecm-icmrm-files
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  volumeName: "ecm-icmrm-files-pv"
```

Sample YAML script to create a service and deployment

```
apiVersion: v1
kind: Service
metadata:
  name: ecm-icmrm-svc
spec:
  ports:
    - name: http
      protocol: TCP
      port: 9080
      nodePort: 30080
      targetPort: 9080
    - name: https
      protocol: TCP
      port: 9443
      nodePort: 30443
      targetPort: 9443
    - name: listener
```

```
    protocol: TCP
    port: 32775
    nodePort: 32767
    targetPort: 32775
  selector:
    app: ecm-icmrm-svc
  #type: LoadBalancer
  type: NodePort
  sessionAffinity: ClientIP
  sessionAffinityConfig:
    clientIP:
      timeoutSeconds: 60
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ecm-icmrm-dep
spec:
  replicas: 2
  selector:
    matchLabels:
      app: ecm-icmrm-svc
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: ecm-icmrm-svc
    spec:
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: NotIn
                  values:
                  - ecm-icmrm-svc
              topologyKey: "kubernetes.io/hostname"
      containers:
```

```
     - image:
image-registry.openshift-image-registry.svc:5000/icmrm-user/icmrm-app:8.7.00
.400
       imagePullPolicy: Always
       name: ecm-icmrm
       ports:
       - containerPort: 9080
         name: http
       - containerPort: 9443
         name: https
       - containerPort: 32775
         name: listener
       volumeMounts:
       - name: ecm-icmrm-files
         mountPath: /staging
         subPath: staging
       - name: ecm-icmrm-files
         mountPath: /opt/ibm/wlp/output/defaultServer/logs
         subPath: output/defaultServer/logs
       # Define two file system volumes
       - name: ecm-icmrm-vol1
         mountPath: /ecm-icmrm-vol1
       - name: ecm-icmrm-vol2
         mountPath: /ecm-icmrm-vol2

     volumes:
     - name: ecm-icmrm-files
       persistentVolumeClaim:
         claimName: ecm-icmrm-files-pvc
     - name: ecm-icmrm-vol1
       persistentVolumeClaim:
         claimName: ecm-icmrm-vol1-pvc
     - name: ecm-icmrm-vol2
       persistentVolumeClaim:
         claimName: ecm-icmrm-vol2-pvc
```

Sample YAML script to create route

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: ecm-icmrm-route
  namespace: icmrm-user
```

```
  resourceVersion: '1492374'
  uid: 35e5b31a-8f96-11ea-923e-0800270ae510
spec:
  host:
icmrm-route-icmrm-user.apps.utp02-cm8oc-l01.development.unicom.software
  port:
    targetPort: https
  tls:
    termination: passthrough
  to:
    kind: Service
    name: ecm-icmrm-svc
    weight: 100
  wildcardPolicy: None
status:
  ingress:
    - conditions:
        - lastTransitionTime: '2020-05-06T12:36:30Z'
          status: 'True'
          type: Admitted
      host:
icmrm-route-icmrm-user.apps.utp02-cm8oc-l01.development.unicom.software
      routerName: router
      wildcardPolicy: None
```