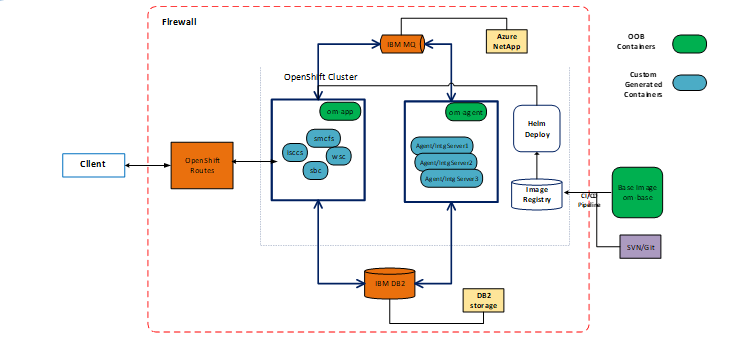
## **Introduction**

IBM Sterling Order Management v10 provides IBM-certified containers that can be deployed on RedHat OpenShift platform. Also, the containers can be deployed on Kubernetes cluster on the customer’s cloud environment.

As a part of one engagement for Infosys, the IBM GSI labs worked hand-in-glove with the Infosys pre-sales and delivery teams, to help with the OMS v10 deployment on Azure.

IBM Sterling OMS comes with 3 images - om-base , om-app and om-agent, which can be pulled from the IBM cloud registry.

Here’s a high level architecture used for the deployment :



## IBM DB2 and IBM MQ are deployed outside of cluster, on Azure Virtual machines.

## The NFS share is used for Persistent Volume storage for deploying the pods - Azure NetApp is used as Network File Storage (NFS),mount points to be created for MQ.

* Latest Code should be pulled for repository (Git/Bitbucket) and custom images should be built, which should be pushed to Image Registry as part of build process.
* Custom images should be deployed to Cluster using Helm Chart provided by IBM for OpenShift version and using Helm deployment process.
* Customized Application, agent and integration servers will be deployed as pods in Open Shift cluster. Client will access these pods through open shift routes.

## **High level Components**

**Container Images:**

* om-app — Order management application server image handling synchronous traffic patterns embedded with IBM WebSphere® Liberty application server
* om-agent — Order management workflow agent and integration server container to handle asynchronous traffic patterns
* om-base — Image provisioned for adding your extensions/customizations to the base product and enabled to rebuild the preceding images

**Helm Charts**

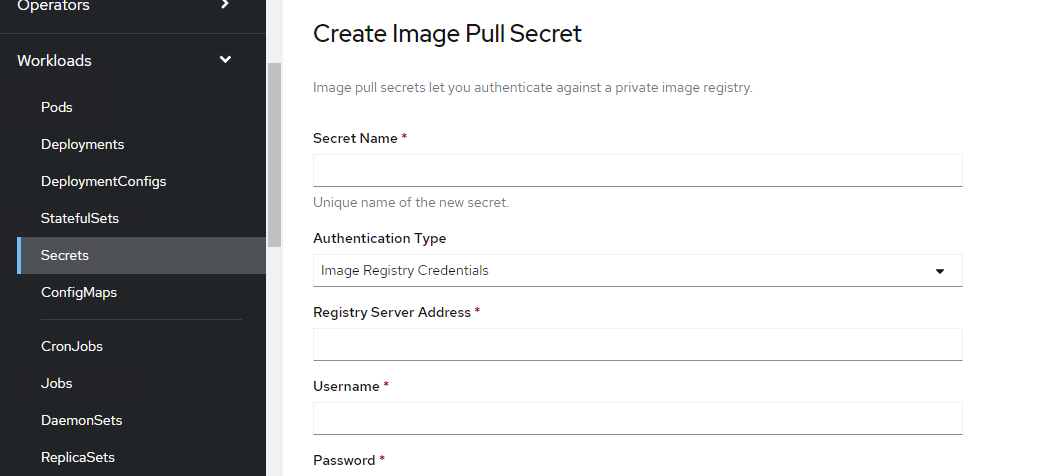
IBM provides Helm Chart for Open shift that could be downloaded from IBM Knowledge center using the below link.

<https://www.ibm.com/docs/en/order-management-sw/10.0?topic=artifacts-downloading-helm-charts>

## **Setting up Environment**

**Prerequisites:**

* Create ARO Cluster.
* Procure Azure NetApp and create NFS mount.
* Install DB2 on VM server outside of Open shift cluster.
* Install MQ on VM server outside of Open shift cluster.
* Install Docker on VM build server.
* Copy the Helm binary to build server.
* Copy the Helm chart downloaded from IBM knowledge center to the build server.
* Download the latest images from IBM repository with IBM entitlement key using these [instructions](https://www.ibm.com/docs/en/order-management-sw/10.0?topic=prerequisites-obtaining-container-images).
* Create image-pull secret - This is required for connecting to Azure Container Registry to pull the images as part of Helm Deployment.

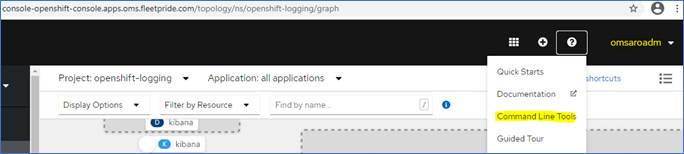


Give the Secret name, Image Registry url , userid and password.

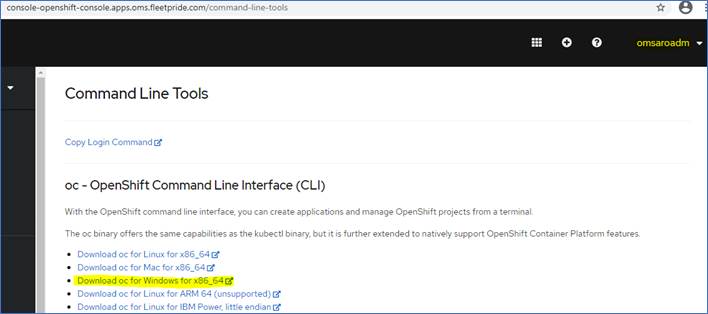
Execute the below command for linking the secret with service account.

|  |
| --- |
| *oc secrets link default <secret-name>  --for=pull* |

* Setup the oc command utility on the build server
  + Once ARO cluster setup is complete , login to ARO console



* Download “oc client” for windows/linux by clicking the respective link as shown in the below screenhsot



* Unzip the archive and we should be able to see “oc client”executable

**High-level flow:**



|  |
| --- |
|  |

|  |
| --- |
|  |

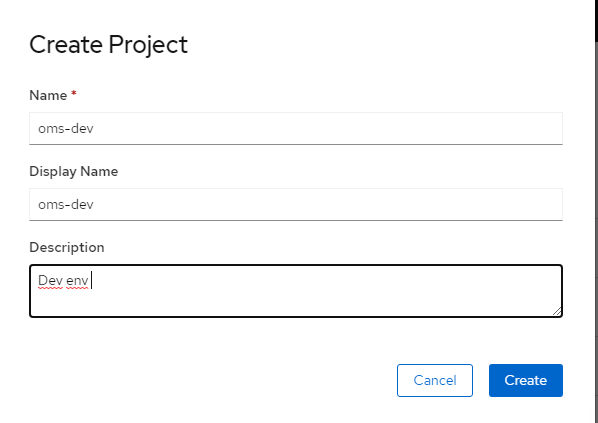
**Build Process:**

* **Download the om-base image from IBM Registry though entitlement key.**
* **Explode the container from base image.**
* **Update the sandbox.cfg file with Database Host, port , database name and schema within the container**
* **Run setup files command.**
* **Copy the customization and create custom jar for java code.**
* **Build the resources JAR and entities JAR**
* **Generate new images**
* **Load the images and create tag for new images.**
* **Push the custom images to Azure Container Registry.**

|  |
| --- |
| *docker run -e LICENSE=accept --privileged -v <shared file   system directory path>:/opt/ssfs/shared -it --name  <container name> <image>*  *docker exec -it <containerid> bash*  Update the sandbox.cfg files under /opt/ssfs/runtime/properties  Execute *./setupfiles.sh*  Copy the required custom xsl and xmls  Build resource jar  *./deployer.sh -t resourcejar*  Copy all the required java classes.  *./install3rdParty.sh <classes> 1 -j /opt/ssfs/shared/<classes.jar> -targetJVM EVERY*  Copy the Extensions.xml  Build entities.jar  *./deployer.sh -t entitydeployer*  Generate app, agent image  */opt/ssfs/runtime/container-scripts/imagebuild*  *./generateImages.sh --MODE=app,agent --DEV\_MODE=true*  Load Images, Tag Images and Push images to Registry.  *docker load –i om-app\_10.0.tar.gz*  *docker load –i om-agent\_10.0.tar.gz*  *docker tag <imageid> <registryname>: <tagname>*  *docker push > <registryname>: <tagname>* |

**Deployment Process:**

1. From Open Shift Console, create a project



1. Add the project to any userid system access.

*oc adm policy add-scc-to-user anyuid system:serviceaccount:<namespace>:default*

1. Create Global secret**,** secret for data source connectivity details.

**oms\_secret.yaml**

|  |
| --- |
| apiVersion: v1  kind: Secret  metadata:  name: 'oms-qa-secret'  type: Opaque  stringData:  consoleadminpassword: 'wasadmin'  consolenonadminpassword: 'wasadmin'  dbpassword: '—‘ |

1. Create Role and Role Bindings. Refer to the [Readme](https://github.com/IBM/charts/blob/master/repo/ibm-helm/ibm-oms-pro-prod.md" \l "creating-a-role-based-access-control-RBAC)
   1. Role and RoleBinding are used to create Role Based Access Control for the default service account with the namespace.

**Role yaml**

|  |
| --- |
| kind: Role  apiVersion: rbac.authorization.k8s.io/v1  metadata:  name: oms-role-oms-qa  namespace: oms-qa  rules:  - apiGroups: ['']  resources: ['secrets']  verbs: ['get', 'watch', 'list', 'create', 'delete', 'patch', 'update'] |

**RoleBinding yaml**

|  |
| --- |
| kind: RoleBinding  apiVersion: rbac.authorization.k8s.io/v1  metadata:  name: oms-rolebinding-oms-qa  namespace: oms-qa  subjects:  - kind: ServiceAccount  name: default  namespace: oms-qa  roleRef:  kind: Role  name: oms-role-oms-qa  apiGroup: rbac.authorization.k8s.io |

1. Create PV and PVC

PV is the storage that has been provisioned for the application.PVC is the claim for storage that has been provisioned for the application.

We faced issues with Azure File storage. As a work around, we used the NFS file storage and created the PV and PVC.

**PV Yaml**

|  |
| --- |
| kind: PersistentVolume  apiVersion: v1  metadata:  name: oms-qa-pv  spec:  capacity:  storage: 10Gi  nfs:  server: <IP address>  path: <Path to NFS>  accessModes:  - ReadWriteMany  persistentVolumeReclaimPolicy: Retain  volumeMode: Filesystem |

**PVC Yaml**

|  |
| --- |
| kind: PersistentVolumeClaim  apiVersion: v1  metadata:  name: oms-qa-ibm-oms-pro-prod-oms-common  spec:  accessModes:  - ReadWriteMany  resources:  requests:  storage: 10Gi  volumeName: oms-qa-pv  storageClassName: ''  volumeMode: Filesystem |

1. Create Azure Container Registry Secret, as mentioned in the pre-requisite section.
2. Edit values.yaml file with appsecret, db properties, customer overrides properties, agent, app tags and image registry properties.
3. Helm install – Use this command to deploy the pods to cluster.
   1. helm install --debug <namespace> -f <path to values.yaml> <release-name>
4. Helm Upgrade – Use this command to update the pods with new changes
   1. helm upgrade <namespace> -f <path to values.yaml> <release-name>

**Sample Yaml file**

|  |
| --- |
| global:  license: true  license\_store\_call\_center: true  image:  repository: 'nprdsomsconreg.azurecr.io'  appSecret: oms-qa-secret  database:  serverName: DBIP  port: DBport  dbname: dbname  user: user  dbvendor: DB2  datasourceName: jdbc/OMDS  systemPool: true  schema: schemaname  serviceAccountName: default  customerOverrides:  envs: []  persistence:  claims:  name: oms-common  accessMode: ReadWriteMany  capacity: 100  capacityUnit: Gi  storageClassName: ''  securityContext:  fsGroup: 0  supplementalGroup: 0  mq:  bindingConfigName: qabindings  bindingMountPath: /opt/ssfs/.bindings  arch:  amd64: 2 - No preference  ppc64le: 2 - No preference  log:  format: json  customConfigMaps: []  customSecrets: []  appserver:  deploymentStrategy: {}  exposeRestService: False  replicaCount: 1  image:  tag: 10.0.0.21  pullPolicy: IfNotPresent  names:  - name: om-app  tag: 10.0.021  routePrefix: qa  config:  vendor: websphere  vendorFile: servers.properties  serverName: DefaultAppServer  jvm:  xms: 1024m  xmx: 2048m  params: []  database:  maxPoolSize: 50  minPoolSize: 10  corethreads: 20  maxthreads: 100  libertyServerXml: ''  libertyStartupWrapper: /opt/ibm/helpers/runtime/docker-server.sh  livenessCheckBeginAfterSeconds: 900  livenessFailRestartAfterMinutes: 10  service:  http:  port: 9080  https:  port: 9443  annotations: {}  labels: {}  resources:  requests:  memory: 2560Mi  cpu: 1  limits:  memory: 3840Mi  cpu: 2  ingress:  host: <domain name of the cluster’s proxy node>  ssl:  enabled: false  secretname: ''  controller: nginx  contextRoots:  - smcfs  - sbc  - sma  - isccs  - wsc  - adminCenter  annotations: {}  labels: {}  podLabels: {}  tolerations: []  nodeAffinity:  requiredDuringSchedulingIgnoredDuringExecution: {}  preferredDuringSchedulingIgnoredDuringExecution: []  podAffinity:  requiredDuringSchedulingIgnoredDuringExecution: []  preferredDuringSchedulingIgnoredDuringExecution: []  podAntiAffinity:  requiredDuringSchedulingIgnoredDuringExecution: []  preferredDuringSchedulingIgnoredDuringExecution: []  replicaNotOnSameNode:  mode: prefer  weightForPreference: 100  omserver:  deploymentStrategy: {}  image:  name: om-agent  tag: qaagent0104  pullPolicy: IfNotPresent  common:  jvmArgs: '-Xms512m -Xmx1024m'  replicaCount: 1  resources:  requests:  memory: 1024Mi  cpu: 0.5  limits:  memory: 2048Mi  cpu: 1  readinessFailRestartAfterMinutes: 10  podLabels: {}  tolerations: []  nodeAffinity:  requiredDuringSchedulingIgnoredDuringExecution: {}  preferredDuringSchedulingIgnoredDuringExecution: []  podAffinity:  requiredDuringSchedulingIgnoredDuringExecution: []  preferredDuringSchedulingIgnoredDuringExecution: []  podAntiAffinity:  requiredDuringSchedulingIgnoredDuringExecution: []  preferredDuringSchedulingIgnoredDuringExecution: []  replicaNotOnSameNode:  mode: prefer  weightForPreference: 100  deployHealthMonitor: true  servers:  - group: "Integration"  name:  - IntServer1  replicaCount: 1  resources:  requests:  memory: 1024Mi  cpu: 0.5  - group: "Agent"  name:  - AgtServer1  replicaCount: 1  resources:  requests:  memory: 1024Mi  cpu: 0.5  datasetup:  loadFactoryData: install  mode: create  fixPack:  loadFPFactoryData: install  installedFPNo: 0 |

Set datasetup.loadFactoryData to install for the first time to run the datasetup job. Once helm install is executed and data setup pod is complete, set it to donotinstall or blank, so that the datasetup job isn’t invoked.

Set datasetup.fixPack.loadFPFactoryData to install and datasetup.fixPack.installedFPNo to 0 for initial installation only

**Other Infra activities**

**SSL Certificates:**

Below are the steps to be followed for any outbound external system integration from OMS

* Copy certificate to build server.
* Execute rsync to copy the certificate from build server to the appserver pod

|  |
| --- |
| oc rsync <sourcedir> <podname>: <sharedpath> |

* Connect to pod through terminal session
* Go to NFS mount shared path.
* openssl command is used to convert certificatetype to pem from cer.

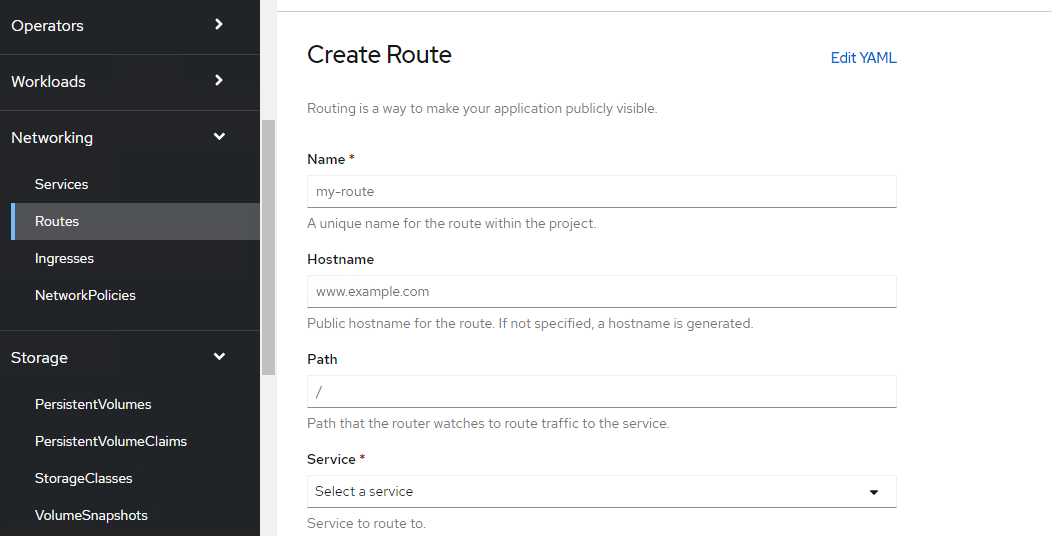
|  |
| --- |
| openssl x509 -in <cert>.cer -outform PEM -out <cert>.pem |

* Copy the .pem file to shared path.
* Set permissions to the .pem file and restart appserver pod

**External Domain:**

External Domain should be created to expose applications to external systems. Open shift routes should be created with external domain, will be used by all inbound interfaces that will access OMS applications. The same route should be created for Production and DR so that in case of DR all external systems will access the same url without any change.

Only change will be DNS switch from PROD to DR IP.



**Azure Load Balancer configuration for DB2 clustering**

In Azure, for IBM DB2 - HA Cluster Setup VIP will not work ,as Virtual IPs are not accessible over network.

Azure Load Balancer should be created and configurations to be made. All traffic from OMS application to DB will be through Azure Load Balancer. Do ensure that the LB IP should be same as DB VIP.

[https://docs.microsoft.com/en-us/azure/virtual-machines/workloads/sap/high-availability-guide-rhel-ibm-db2-luw](https://apc01.safelinks.protection.outlook.com/?url=https%3A%2F%2Fdocs.microsoft.com%2Fen-us%2Fazure%2Fvirtual-machines%2Fworkloads%2Fsap%2Fhigh-availability-guide-rhel-ibm-db2-luw&data=04%7C01%7CHaritha_Tirumuru%40infosys.com%7Ca0ac124f0f5d4df48cc908d955d82add%7C63ce7d592f3e42cda8ccbe764cff5eb6%7C0%7C0%7C637635210447448213%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=4lePI79C4FwH9Dz8RWKzbvuMaeGh%2BI5JXaOODGUEz%2Fw%3D&reserved=0)

Azure LB works in Active Active mode, where as we intend IBM DB2 to work in Active-Passive mode. Now, Azure Load balancer could redirect traffic to any DB node, as ports are up in both DB nodes.

To ensure that DB2 remains passive on one node, configure a dummy port on both the nodes from OS front. Meanwhile backend script brings up the port on PRIMARY node and downs the port on STANDBY node. Health probe port on LB is set to dummy port, so that LB can route the traffic to only one node on which the port is up.