HCL OneDB Containerized Deployment

Contents

Chapter 1. Deploying HCL OneDB using Helm charts

The OneDB Database server as well as OneDB Connect, OneDB Mongo Wire Listener, OneDB Rest Wire Listener and OneDB Explore have all been designed to work in a kubernetes environment.

This topic lists information about the enhanced/new features and a compatibility matrix for the available Helm Charts :

What's new in 0.4.27 chart version

This version includes the following enhancements:

- Removal of NFS dependency.
- Helm upgrade now supports "OnDelete" update strategy.

What's new in 0.4.16 Chart Version

This version includes the following enhancements:

- Ability to create custom Environment for Mongo container on page 104.
- Ability to create custom Environment for REST container on page 103.
- Ability to create custom Environment for Explore container on page 105.
- OneDB Product Configuration on page 106.
- Added a new External Connection Manager Service on page 115.

What's new in 0.4.12 Chart Version

This version includes the following enhancements:

- Connection Manager on page 172
- HA Scale out on page 192
- Automatic Failover on page 190
- Automatic Backups on page 183
- Support Multiple PVs for Storage on page 161
- Cloud Native method for Creating Spaces on page 165
- Cloud Native method for configuration/Users on page 166
- Support custom Init Container on page 167

Compatibility Matrix

Helm Chart Version	Product Versions Supported	Kubernetes Version Supported
0.4.27	onedb-server:2.0.1.2 onedb-operator:2.0.1.2 onedb-cm:2.0.1.2 onedb-mongo-connector:2.0.1.2 onedb-rest-connector:2.0.1.2 onedb-explore:2.0.1.2	1.18 - 1.22
0.4.16	onedb-server:2.0.1.0/2.0.1.1 onedb-operator:2.0.1.0 onedb-cm:2.0.1.0 onedb-mongo-connector:2.0.1.0/2.0.1.1 onedb-rest-connector:2.0.1.0/2.0.1.1 onedb-explore:2.0.1.0/2.0.1.1	1.18 - 1.21
0.4.12	onedb-server:2.0.1.0 onedb-operator:2.0.1.0 onedb-cm:2.0.1.0 onedb-mongo-connector:2.0.1.0 onedb-rest-connector:2.0.1.0 onedb-explore:2.0.1.0	1.18 - 1.21

Charts 0.4.27

This version includes the following enhancements:

- Removal of NFS dependency. on page 7
- Helm upgrade now supports “OnDelete” update strategy. on page 66

What’s New in this Helm Chart Version

This section includes information about the new, enhanced capabilities added in this version of the helm chart :

- Removal of NFS dependency. on page 7
- Helm upgrade now supports “OnDelete” update strategy. on page 66

Supported Platforms

The OneDB Helm charts have been tested on the following platforms:

- Google Kubernetes Engine (GKE) (https://cloud.google.com/kubernetes-engine)
- AWS Elastic Kubernetes Service (EKS) (https://aws.amazon.com/eks)
- Azure Kubernetes Service (AKS) (https://azure.microsoft.com/en-us/services/kubernetes-service)
- Redhat OpenShift Container Platform (OCP) (https://www.redhat.com/en/technologies/cloud-computing/openshift/container-platform)

Architectural Overview

Installing and deploying OneDB in a cloud-native environment is a new way of looking at things. An evolution of how OneDB is or can be deployed has occurred: starting with on-premises, to in the cloud in Virutal machines, to in the cloud in a highly scalable Kubernetes environment.

In the past, you would have acquired a physical machine, installed the OneDB database server on that machine and been responsible for the maintenance and upgrades on the machine as well as maintenance of the OneDB Database server.

There was then a move to the cloud and the use of Virtual machines in that cloud. Virtual machines made it possible to start up a machine and run a playbook that would install OneDB and configure accordingly. You might do this in your own cloud or a public cloud.

Then more recently, there is the move to a highly scaleable Kubernetes environment. This approach uses containerization of products and pieces of an entire solution. It allows for great flexibility with many benefits. You may use your own Kubernetes solution or a cloud provided Kubernetes from Google, Amazon or Microsoft for example.

General Terminology

To undestand how OneDB Database server is deployed in a kubernetes environment, it is important that you have a basic knowledge of certain terms:

Container

A container image is a lightweight standalone executable package of software that includes everything to run an application includeing system libraries, tools etc.

Docker

Docker is the leading technology for containizeration. When people think of containers they typcially think of docker. Although it is not the only container technology.

Microservices

A microservices architecture is a method of designing an overall solution to be broken up into smaller parts instead of a single monolithic application. Containers make this a natural path of software development as different pieces can be represented by a different container image.

OneDB HA Cluster

This use of the term cluster refers to the High availability nature of 2 or more OneDB Database servers working together. A 2 nodes OneDB HA cluster will consist of a OneDB HA Primary server and a OneDB HA Secondary server. More servers can be added into a OneDB HA Cluster, in this context the additional servers would be added as OneDB HA RSS nodes.

Kubernetes Terminology

Node

A node is a virtual machine or phyiscal machine with CPU/RAM resources. This is the hardware component that makes up a kubernetes cluster. Example nodes are worker nodes and master nodes.

Pod

A pod is the simplest unit that exists within kuberenetes. Typically this is 1 more more containers. It is pods that get scheduled to run on kubernetes nodes.

Cluster (kubernetes)

Is made up of 1 or more nodes. They provide a resource for a kubernetes solution to be deployed into and managed.

Service

An abstract API object that exposes an application’s network services.

Helm chart

A helm chart is a collection of files that describe a related set of kubernetes resources. A helm chart is typically a group of yaml files and other associated files that is used to deploy a solution into kubernetes.

Operator

A kubernetes operator is an application specific controller that extends the functionality of the kubernetes API.

LoadBalancer

A kubernetes object that allows you to expose an external IP address to outside the kubernetes cluster.

OneDB Deployment Resources

When deploying a OneDB helm chart a group of resources will be created. The resources created will depend on the specific OneDB Helm chart that is used.

The OneDB-sql helm chart will deploy the following resources:

- onedb-operator pod
- onedb-server-X pod
- onedbcm-X pod
- onedbcm-cm-service

The OneDB-mongo helm chart will deploy the following resources:

- odbp-mongo pod
- odbp-mongo service
- OneDB-sql chart

The OneDB-rest helm chart will deploy the following resources:

- odbp-rest pod
- odbp-rest service
- OneDB-sql chart

The OneDB-explore helm chart will deploy the following resources:

- odbp-explore pod
- odbp-explore service

The OneDB-product helm chart will deploy the following resources:

- OneDB-sql chart
- OneDB-mongo chart

- OneDB-rest chart
- OneDB-explore chart

Pods

onedb-operator

The purpose of the operator pod is to manager the OneDB HA cluster. By default, a OneDB HA Cluster is started with an HDR primary and secondary server, along with two connection managers.

onedb-server-x

This is the OneDB Database server pod. When deployed, a statefulset is used which will be assigned an ordinal index starting with 0. So, OneDB HA cluster with a primary secondary will have onedb-server-0 and onedb-server-1.

onedbcm-x

This is the OneDB Connection manager pod. It will be assigned an ordinal index starting with 0. By default, 2 connection managers are started. onedbcm-0 and onedbcm-1.

odbp-mongo

This is the OneDB Mongo Listener pod. It is started when the OneDB Mongo chart is deployed. It is used to connect to the OneDB Database server using the Mongo API.

odbp-rest

This is the OneDB REST Listener pod. It is started when the OneDB REST chart is deployed. It is used to connect to the OneDB Database esrver using RESTFUL services.

odbp-explore

This is the OneDB Explore pod. It will deploy the OneDB Explore administration and monitoring tool providing a web admin and monitoring GUI. It can be used to administer one or more OneDB Database servers.

Services

odbp-explore

This is the OneDB Explore service that can be used to access the OneDB Explore product.

odbp-mongo

This is the OneDB Mongo service that is used to access the OneDB Database server using the Mongo API.

odbp-rest

This is the OneDB REST service that is used to access the OneDB Database server using RESTFUL services.

onedbcm-cm-service

This is the OneDB Connection Manager service that is used to access the OneDB Database server using the SQLI + DRDA protocol. EX: JDBC, ODBC.

Prerequisites

To install OneDB into a kubernetes cluster, following prerequisites are needed:

- kubectl
- helm
- ReadWriteMany storage class

Note: To install HCL Sofy Solution into a kubernetes cluster, there may be additional requirements. For more information on the installation instructions for HCL Sofy, see (https://hclsofy.com/ua/guides#installing-solutions-step-by-step-instructions)

.

Kubectl

The kubernetes command line tool, kubectl, is used to run commands and interact with a kubernetes cluster. This is used for managing and interacting with OneDB in kubernetes.

Helm

The helm tool is used to install OneDB in a kubernetes cluster. Helm is a package manager for Kubernetes and is used to install a helm chart.

A helm chart is simply a set of kubernetes yaml manifests that are combined into a single package. This provide an easy method to install a group of kubernetes manifests as a single package.

For installations steps and more information on helm, see: https://helm.sh

RWM Storage

ReadWriteMany(RWM) PVC is an “optional” but recommended configuration to support database backup for OneDB server storage spaces and logical log files. RWM PVC mounted across all OneDB server pods and backup device available across all pods. Listed are some of the available options to install RWM storage, but not limited to these. Following options have been tested and verified to work with OneDB.

Important: Enable one and only one of the following options:

Cloud specific options that can be used for these specific cloud providers are:

- Google FireStore on page 8
- AWS Elastic filesystem on page 8
- Azure on page 9

Cloud generic options that can be installed into an existing kubernetes cluster are:

- nfs-server-provisionser on page 10
- rook-ceph on page 11
- rook-nfs on page 12

Google FileStore Configuration

1. See the Google filestore Instructions (https://cloud.google.com/filestore/docs/quickstart-console).
2. The following OneDB helm chart configuration values need to be set to use the Google filestore:

Parameter	***Description***	***Value***
nfsserver.enabled	Set this attribute to 'true' to mount ReadWriteMany PVC across all OneDB server pods. Required for OnBar storage space and logical log backups. If this attribute value is set to 'false' then rest of the "nfsserver" related helm attributes are ignored.	true
nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.googleFilestore.enable	Set to 'true' to enable Google Filestore	true
nfsserver.googleFilestore.filestoreIP	IP address of the Filestore instance	''
nfsserver.googleFilestore.filestoreShare	Name of the File share of the instance	''

AWS Elastic Filesystem Configuration

1. See AWS filesystem Instructions (https://docs.aws.amazon/eks/latest/userguide/efs-csi.html) .
2. The following OneDB helm chart configuration values need to be set to use the AWS Elastic filesystem.

Parameter	***Description***	***Value***

nfsserver.enabled	Set this attribute to 'true' to mount ReadWriteMany PVC across all OneDB server pods. Required for OnBar storage space and logical log backups. If this attribute value is set to 'false' then rest of the "nfsserver" related helm attributes are ignored.	true
nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.awsEFS.enable	Set to 'true' to enable AWS Elastic filestore	true
nfsserver.awsEFS.EFSServer	IP address of the Filestore instance	''
nfsserver.googleFilestore.filestoreShare	Name of the File share of the instance	''

Azure File share Configuration

1. See Azure File share instructions (https://docs.microsoft.com/en-us/azure/aks/azure-files-volume).
2. Following OneDB helm chart configuration values need to be set to use the Azure File share:

Parameter	***Description***	***Value***
nfsserver.enabled	Set this attribute to 'true' to mount ReadWriteMany PVC across all OneDB server pods. Required for OnBar storage space and logical log backups. If this attribute value is set to 'false' then rest of the "nfsserver" related helm attributes are ignored.	true
nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.azureFS.enable	Set to 'true' to enable Azure File share	true

nfsserver.azureFS.secretname	Kubernetes secret to use	”
nfsserver.azureFS.shareName	Azure file share name	”

Install and Configure nfs-server-provisioner

1. Add the nfs-server-provisioner helm repo.

```
helm repo add kvaps https://kvaps.github.io/charts
```

2. Install the helm chart for the nfs-server-provisioner. Specify the following parameters:

Parameter	***Description***	***Value***
persistence.size	Set this to a value of the NFS PV size	50Gi
persistence.enabled	Set to ‘true’ to enable NFS	true
persistence.storageClass	Set this to ‘standard’	standard
storageClass.create	Set to ‘true’	true
storageClass.name	Set thos to a unique Name	onedb-nfs-<namespace>
storageClass.mountOptions		{vers=4.1}

- **storageClass.name**: This is cluster wide so it is recommended to include the namespace in the name to provide uniqueness.
- **storageClass.mountOptions**: Onedb has been tested with NFS V4.1

```
helm install onedb-nfs-server-provisioner kvaps/nfs-server-provisioner   \
--version 1.3.1
--set persistence.enabled=true
--set persistence.storageClass="standard”
--set persistence.size=50Gi
--set storageClass.create=true
--set storageClass.name=-onedb-nfs-my-ns
 --set storageClass.mountOptions={vers=4.1}
```

3. The following OneDB helm chart configuration values need to be set to use the NFS server provisioner.

Parameter	*Description*	*Value*
nfsserver.enabled	Set this attribute to 'true' to mount ReadWriteMany PVC across all OneDB server pods. Required for OnBar storage space and logical log backups. If this attribute value is set to 'false' then rest of the "nfsserver" related helm attributes are ignored.	true
nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.other.enable	Set to 'true' to enable NFS	true
nfsserver.other.storageClass	Set this to the storage class of the NFS	onedb-nfs-<namespace>

- **nfsserver.other.storageClass:** This is set to the the storageClass name specified in the creation of the nfs server provisioner

Install and Configure rook-ceph

1. See the rook-ceph Prerequisites: (https://rook.io/docs/rook/v1.7/pre-reqs.html) .

   Note: Some environments you may need to provision and use *Ubuntu with containerd* node pool instead of the default *GKE container-Optimized OS* (COS).

   .
2. Follow the instructions for rook-ceph: (https://rook.io/docs/rook/v1.7/quickstart.html).
3. Configure a shared file system for rook: (https://rook.io/docs/rook/v1.7/ceph-filesystem.html).
4. Following OneDB helm chart configuration values need to be set to use rook-ceph:

Parameter	*Description*	*Value*
nfsserver.enabled	Set this attribute to 'true' to mount ReadWriteMany PVC across all OneDB server pods. Required for OnBar storage space and logical log backups. If this attribute value is set to 'false' then rest of the "nfsserver" related helm attributes are ignored.	true

nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.other.enable	Set to 'true' to enable NFS	true
nfsserver.other.storageClass	Set this to the storage class of the NFS	onedb-nfs-<namespace>

- **nfsserver.other.storageClass:** This is set to the the storageClass name specified in the creation of rook-ceph.

Installation of rook-nfs

Introduction to charts content to go here.

1. Follow the instructions for rook-nfs (https://github.com/rook/rook/blob/master/Documentation/nfs.md).
2. Before Installing OneDB modify the *template/nfs_other_pvc.yaml* file in the helm chart and change the **accessModes:** value from ***ReadWriteMany*** to ***ReadWriteOnce***.
3. After creating the Storage Class, refer to the *sc.yaml* file for rook-nfs. This will contain the storageclass name.
   - Default: rook-nfs-share1.
   - The storage name is needed when installing OneDB.

Parameter	***Description***	***Value***
nfsserver.enabled	Set this attribute to 'true' to mount ReadWriteMany PVC across all OneDB server pods. Required for OnBar storage space and logical log backups. If this attribute value is set to 'false' then rest of the "nfsserver" related helm attributes are ignored.	true
nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.other.enable	Set to 'true' to enable NFS	true
nfsserver.other.storageClass	Set this to the storage class of the NFS	rook-nfs

		-shar e1

- **nfsserver.other.storageClass:** This is set to the the storageClass name specified in the creation of rook-nfs.

OneDB Requirements and Recommendations

The OneDB database server is designed to be able to run on small devices like a Raspberry pi up to large Servers with 128 cores. The architecture of OneDB is flexible and allows you to run in these different environments with different configurations.

It is important to note that these are recommendations and not requirements. As one user may be able to run their workload on a small device like a Raspberry pi, but another user needs 32 CPUs and 100GB of memory.

When talking about recommendations, we typically refer to CPU, Memory and sometimes disk space.

OneDB Disk/Volume Recommendations

This depends on the amount of data and workload you will have in your OneDB Database server. So every database system will be different. But if High thoughput is needed then we recommend SSD drives to be used. And for your NFS shred drive, spinning disks are ok to use.

Below is a priority of spaces to be setup with SSD if possible. This is not required but the more spaces/volumes setup with SSD drives the better performance can be achieved with the OneDB Database server.

Space/Volume	Drive Type
Logical Log Dbspace	SSD drive(s)
High Volume space	SSD drive(s)
Temp Spaces	SSD drive(s)
Physical Log Dbspace	SSD drive(s)
Low volume space	SSD/Spinning drive(s)
RWM NFS	Spinning drive(s)

The amount of disk space allocated to each of these spaces and volumes is dependent on the size of your data and workload. It is recommended that the RWM NFS volume be approx 3-5 times the size of the total dbspaces if you plan to use the automated backups. We retain 3 archives of the OneDB database server.

Note: It is important to note that you can use all spinning disks and if needed you can put all spaces on a single volume, a separate volume is needed for the RWM NFS. These recommendations are given to provide the best performance possible for a production system.

OneDB Minimum CPU/Memory Recommendations

For a OneDB solution the following can be used as guidelines for the OneDB Server, the Connection Manager, the Mongo wire listener, and the REST wire listener. With OneDB Explore, the minimmum recommendation should be plenty.

Resource	Minimmum Recommendation	General Recommendatoin
CPU	1 core	2 cores
Memory	512 MB	8 GB

As with all systems the more resources, CPU and Memory that a system has the better performance can be achieved. If you find that your workload has a high number of quick connections using the REST or Mongo protocols you may want to increase resources in that area.

The more CPU that is provided to the OneDB system allows you to configure more CPUvps and the more Memory that is provided allows you to configure more memory for Buffers and other database operations.

Note: It is possible to run OneDB with less CPU and Memory. These recommendations are given to provide the best performance possible for a production system.

OneDB Minimum Kubernetes Recommendations

When a OneDB Helm chart is deployed you can specify the minimmum and maximum amount of resources that Kuberentes will use.

When scheduling a pod on a kubernetes node the pod specification can request minimmum resources required. If no node is available with those resources the pod will not be scheduled.

Example: If a pod has a resource.request.cpu of 1, kubernetes will attempt to schedule the pod on a node with >= 1 cpu. If not available, then the pod will not be scheduled.

The following are the current values set in the OneDB Helm Charts.

Pod	Resource	Request	Limit
onedb	CPU	.1 CPU	24 CPU

Pod	Resource	Request	Limit
	Memory	2GB	32GB
CM	CPU	.1 CPU	1 CPU
	Memory	100 MB	500 MB
Mongo/REST	CPU	.1 CPu	2.1 CPU
	Memory	128MB	1GB
Explore	CPU	.1 CPU	2 CPU
	Memory	64MB	512MB

The OneDB Helm charts are also configured by default to not allow two OneDB server pods to be scheduled on the same node. See onedb.nodeSelectorRequired configuration parameter.

The OneDB Helm chart is also configureed to not allow two OneDB Connection Manager pods to be scheduled on the same node. See onedbcm.nodeSelectorRequired configuration parameter.

This does not prevent a Connection manager pod from being scheduled on the same node as a OneDB server pod.

For best performance in a production system it is recommended to configure Affinity along with taints and tolerations to have full control of where the OneDB pods will be scheduled. This will allow you to control the resources available to the individual running pod.

Minimum Recommendation:

- 1 node per OneDB server pod
- 2 nodes for all other pods to be schedued on

General Recommendation: For best performance possible in a production system.

- Use affinity, taints and tolerations
- Configure 1 node per OneDB Server pod
- Configure 1 node per CM
- 1 node or Mongo wire listener
- 1 node for REST wire listener
- Configure 1-2 nodes for other pods

Note: It is possible to run a OneDB Helm chart with fewer nodes. These recommendations are given to provide the best performance possible for a production system.

Overview of Installation

OneDB is deployed into a kubernetes cluster using helm charts. A helm chart is a collection of files that describe a related set of kubernetes resources. A helm chart is typically a group of yaml files and other associated files that is used to deploy a solution into kubernetes.

Before installing a helm chart, you need access to the cluster. The Helm CLI is used to perform the install/uninstall and manage a helm release. Helm is commonly referred to as the package manager for kubernetes. For more information on helm and the installation instructions see: (https://helm.sh).

The kubectl CLI is a kubernetes command line tool to interact with and manage resources in a kubernetes cluster. You can use this tool to verify your installation. For more information on kubectl and installation, see: (https://kubernetes.io).

Previous install

If an helm install has been performed with a prior version, there may be a need to update the Custom Resource Definitions in the kubernetes cluster. OneDB's Custom Resource Definitions can change from release to release. You may see this with a helm upgrade as well when moving from one helm chart version to another, where the CRD has changed.

If you perform a helm install/upgrade and receive an error similar to the error below, update the CRDs associated with OneDB:

```
Error: INSTALLATION FAILED: unable to build kubernetes objects from release manifest: error validating "":
 error validating data: ValidationError(OneDB.spec): unknown field
```

Run kubectl get crds to find the name of the OneDB CRDs:

```
kubectl get crds |grep onedb

NAME CREATED AT
onedbcms.onedb.hcl 2022-03-08T17:02:44Z
onedbs.onedb.hcl 2022-03-08T17:02:44Z
```

Run a kubectl apply command to update the existing CRDs using the new CRDs in the new helm chart:

```

kubectl apply -f onedb-sql/chart/crds/onedb.hcl_onedbcms.yaml
kubectl apply -f onedb-sql/chart/crds/onedb.hcl_onedbs.yaml
```

OneDB Helm Charts

There are five helm charts for OneDB. These helm charts are listed below with a description of each:

- **OneDB SQL Data Store (onedb-sql):** This is the Helm chart that contains the OneDB Database server. It uses a kubernetes operator to deploy a statefulset in a kubernetes environment.
- **OneDB Rest Data Store (onedb-rest):** This is a helm chart that deploys the OneDB Database server with the REST listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Document Data Store (onedb-mongo):** This is a helm chart that deploys the OneDB SQL Data Store with the Mongo listener. The OneDB SQL Data Store is a subchart in this chart.

- **OneDB Explore (onedb-explore):** This is a Helm chart that contains only the OneDB Explore UI.
- **OneDB Product (onedb-product):** This is a Helm chart that contains OneDB SQL Data Store, OneDB Rest, OneDB Document and OneDB Explore all as subcharts.

Differences in Standalone and Solution Factory helm charts

Note: HCL does not currently provide standalone helm charts the only way to get a helm chart for OneDB is through the Solution Factory (Sofy).

The Solution Factory (Sofy), is an Enterprise Kubernetes Solution catalog. Sofy allows you to pick and choose various products to create an overall solution. You can choose one of the OneDB services or Products from the catalog and it will be included in an overall solution with the OneDB helm chart included as a subchart in the Sofy solution. With a Sofy solution helm chart other charts will be included as subcharts like prometheius and grafana and the Sofy UI and of course any product chosen in the catalog.

The main difference between a OneDB helm chart that is installed with a Sofy solution and installed on its own are:

1. Other products/subcharts are included in the Sofy chart.
2. Helm chart overrides at a different level.

helm install

Helm install is used to install a helm chart. This command can point to a path of a directory of an unpacked chart, or a packaged chart. Ex. (chart.tgz).

HCL does not currently provide standalone helm charts the only way to get a helm chart for OneDB is through the Solution Factory (Sofy).

```
helm install [NAME] [CHART] [flags]
```

Installing an unpacked directory chart:

```
 helm install onedb1 onedb-sql
```

Installing a packaged chart (tgz):

```
helm install onedb1 onedb-sql.tgz
```

helm overrides

To override default values in the helm chart you can use --set on the command line. Or you can specify a file with a list of overrides.

Installing with set overrides:

```
helm install onedb2 --set hclFlexnetURL=flex-net-xxxxx --set hclFlexnetID=xxxxxxx  onedb-sql
```

Installing with a overrides in a file:

```
helm install onedb2-f myvalues.yaml onedb-sql
```

File: myvalues.yaml

```
hclFlexnetURL:  flex-net-xxxxx
hclFlexnetID: xxxxxx
```

Verify Installation

After performing a helm install you can use the kubectl tool to verify the installation. The following resources are some of the items to verify within your kubernetes cluster.

- pods
- services
- deployments
- statefulsets

```
kubectl get pods
```

```
kubectl get services
```

```
kubectl get deployments
```

```
kubectl get statefulsets
```

These commands will show the status of each of these resources. For example, the pods need to be in a running state.
If any of these resources are not in a functioning running state you can use kubectl to diagnose. See the kubernetes documentation for more information on kubectl.

Install a Standalone helm chart

HCL does not currently provide standalone helm charts the only way to get a helm chart for OneDB is through the Solution Factory (Sofy).

Install a Solution Factory helm chart

When installing OneDB in a Sofy solution, it will be included as a subchart in the helm chart that is created from the Sofy catalog along with other Solution factory charts like Prometheus, grafana, Sofy console, etc. For more information about Solution Factory see: (https://hclsofy.com/ua/guides).

Before a Sofy helm chart can be installed, there are required steps to be taken. See the step by step instructions for installing a Sofy solution: (https://hclsofy.com/ua/guides#installing-solutions-step-by-step-instructions) .

License Requirements

The OneDB database server requires a license to be used. This is set using *hclFlexnetURL* and *hclFlexnetID* values in the helm chart. Below is a values override file that sets these parameter values. This values to be used for these parameters will be obtained from HCL.

File: myvalues.yaml

```
hclFlexnetURL:  flex-net-xxxxx
hclFlexnetID: xxxxxx
```

A Sofy solution uses a service named **anchor**. This service is used for license management. The OneDB helm charts use anchor but don't need the amount of resources set by default in a Sofy solution helm chart.

You can override the resources used by this **anchor** service by using the following values override file.

File: anchor.yaml

```
anchor:
    resources:
        cpu: 250m
```

Install OneDB SQL Data Store (onedb-sql)

The OneDB SQL Data Store helm chart will install the HCL OneDB database.

HCL OneDB is an enterprise grade database for storing and processing the following types of data:

- Relational (Table based)
- Document (Json Based)
- Timeseries (Time based)
- Spatial (Coordinate based)

Access to OneDB SQL Data Store is through the native SQLI protocol. HCL OneDB provides drivers for different programming languages to provide this connectivity. Ex. Java, Python, NodeJS, ODBC.

After all system requirements and prerequisites have been addressed you are ready to install the Sofy helm chart. The command below is a sample install that uses a file named myvales.yaml to provide any overrides to the helm chart values and anchor.yaml to set the anchor service's resources.

```
helm install sql-v1  -f myvalues.yaml -f anchor.yaml production-onedb-sql
```

Install OneDB RESTful Data Store (onedb-rest)

The OneDB RESTful Data Store helm chart will install the HCL OneDB database along with the OneDB REST listener.

HCL OneDB is an enterprise grade database for storing and processing the following types of data:

- Relational (Table based)
- Document (Json Based)

- Timeseries (Time based)
- Spatial (Coordinate based)

Access to OneDB RESTful Data store is through the REST API. This allows you to use language of choice that supports RESTful services.

After all system requirements and prerequisites have been addressed you are ready to install the Sofy helm chart. The command below is a sample install that uses a file named myvales.yaml to provide any overrides to the helm chart values and `anchor.yaml` to set the anchor service's resources.

```
helm install rest-v1  -f myvalues.yaml -f anchor.yaml production-onedb-rest
```

Install OneDB Document Data Store (onedb-mongo)

The OneDB Document Data Store helm chart will install the HCL OneDB database along with the OneDB Mongo Listener.

HCL OneDB is an enterprise grade database for storing and processing the following types of data:

- Relational (Table based)
- Document (Json Based)
- Timeseries (Time based)
- Spatial (Coordinate based)

Access to OneDB Document Data store is through the MongoDB protocol. This allows you to use any language that supports a MongoDB driver.

After all system requirements and prerequisites have been addressed you are ready to install the Sofy helm chart. The command below is a sample install that uses a file named myvales.yaml to provide any overrides to the helm chart values and anchor.yaml to set the anchor service's resources.

```
helm install mongo-v1  -f myvalues.yaml -f anchor.yaml production-onedb-mongo
```

Install OneDB Explore (onedb-explore)

The OneDB Explore helm chart will install the OneDB Explore web console. The web console is used for visualizing, monitoring, alerting and administering an HCL OneDB server instances.

HCL OneDB Explore features include:

- Purpose built for ease-of-use, scaling out, and optimizing DevOps needs.
- Provides critical performance management capabilities and monitoring of OneDB data store servers.
- The monitoring system feeds directly into a customizable alerting system so alerts can be immediately sent via email, Twilio, or PagerDuty.
- User and permission management for restricted access to dashboard of certain servers or group of servers.

The default login credentials for HCL OneDB Explore are:

- Username: admin
- Password: testPassw0rd

To override the admin password use a values override file and provide that at install time.

File: myvalues.yaml

```
onedb-explore:
    adminPassword: newPassw0rd
```

```
helm install expl-v1  -f myvalues.yaml  production-onedb-explore
```

Install OneDB Product (onedb-product)

The OneDB Product helm chart will install the HCL OneDB database. This helm chart will include as subcharts the other OneDB helm charts:

- OneDB SQL Data Store
- OneDB Mongo Data Store
- OneDB RESTful Data Store
- OneDB Explore

This chart is an all-inclusive chart that includes all the OneDB charts for full functionality.

HCL OneDB is an enterprise grade database for storing and processing the following types of data:

- Relational (Table based)
- Document (Json Based)
- Timeseries (Time based)
- Spatial (Coordinate based)

Access to OneDB Product is through:

- The native SQLI protocol. HCL OneDB provides drivers for different programming languages to provide this connectivity. Ex. Java, Python, NodeJS, ODBC
- The REST API. This allows you to use language of choice that supports RESTful services.
- The MongoDB protocol. This allows you to use any language that supports a MongoDB driver.

OneDB Explore is included as a UI to interact with and administer the OneDB Database server(s).

After all system requirements and prerequisites have been addressed you are ready to install the Sofy helm chart. The command below is a sample install that uses a file named `myvales.yaml` to provide any overrides to the helm chart values and `anchor.yaml` to set the anchor service's resources.

Following file overrides multiple parameters and is provided an installation time:

File: myvalues.yaml

```
hclFlexnetURL:  flex-net-xxxxx
hclFlexnetID: xxxxxx

anchor:
  resources:
    cpu: 250m

onedb-product:
  onedb-explore:
    adminPassword: newPassw0rd
```

```
helm install prod-v1  -f myvalues.yaml  production-onedb-product
```

Pod Scheduling

As a best practice when deploying OneDB SQL Data store into kubernetes in production isolate the OneDB Database server pods to a specific set of nodes. Also, make sure no two database server pods are scheduled on the save node.

OneDB Server pod scheduling is controlled with the helm chart parameters:

- onedb.nodeSelectorRequired
- onedb.nodeSelector
- onedb.tolerations

To understand how OneDB handles pod scheduling it is import to understand a few concepts.

- Assigning Pods to Nodes (Affinity / Anti-affinity)
- Taints and Tolerations

For more information on Pod scheduling, see kubernetes https://kubernetes.io/.

Assigning Pods to Nodes (Affinity/ Anti-Affinity)

Node Affinity allows you to constrain which nodes your pods are eligible to be scheduled on based on labels that have been defined for the nodes. There are two types of node affinity that are used with OneDB.

- requiredDuringSchedulingIgnoredDuringExecution
- preferredDuringSchedulingIgnoredDuringExecution

requiredDuringSchedulingIgnoredDuringExecution: This is a hard requirement in that the pod “must” be scheduled on the node with the defined set of rules.

preferredDuringSchedulingIgnonredDuringExecution: This is a soft requirement in that the pod will prefer or try to schedule on nodes with the defined rules but it is not guaranteed.

Note: OneDB uses these rules to force a pod to be scheduled on a specific set of nodes or prefer to be scheduled on a specific set of nodes.

Pod anti-affinity allows you to constrain which nodes your pod is eligible to be scheduled on based on pods that are already running on the node. As with node affinity OneDB uses two types of pod anti-affinity.

- requiredDuringSchedulingIgnoredDuringExecution
- preferredDuringSchedulingIgnoredDuringExecution

requiredDuringSchedulingIgnoredDuringExecution: This is a hard requirement in that the pod “must” be scheduled on the node with the defined set of rules.

preferredDuringSchedulingIgnonredDuringExecution: This is a soft requirement in that the pod will prefer or try to schedule on nodes with the defined rules but it is not guaranteed.

Note: OneDB uses these rules to force a OneDB pod to not schedule on nodes already running a OneDB pod or prefer to not be scheduled on that same node.

Labeling Nodes

Your kubernetes administrator will perform this task. They can label a single node or a group of nodes (node pool) with a specific designation with a key/value pair. This is needed to use affinity/anti-affinity capabilities with kubernetes.

To label a node the following command is used:

```
kubectl label nodes <nodename> key=value –overwrite
```

The key/value pair that is defined here is arbitrary. It is a key/value pair that would then be used with helm chart parameter overrides to specify the affinity/anti-affinity.

Example with an arbitrary key/value pair of type=database looks like this:

```
kubectl label nodes gke-worker4 type=database –overwrite
```

Configure OneDB Affinity/Anti-Affinity

We have two helm chart parameters that can be set with OneDB SQL Data store. The OneDB SQL Data store uses these helm chart parameters for both the onedb and onedbcm sections of the helm chart.

```
onedb:
    nodeSelectorRequired: true
    #nodeSelector:
        #type: database
. . .
onedbcm:
    nodeSelectorRequired: true
```

```
    #nodeSelector:
        #type: cm
```

The default values for onedb/onedbcm **nodeSelectorRequired** is **true.** When this is set to true the requiredDuringSchedulingIgnoredDuringExecution is used for Pod anti-affinity.

The effect of this is that, a OneDB Database server will not be scheduled on the same node where another OneDB Database server pod is running. And a OneDB Connection manager will not be scheduled on the same node where another OneDB Connection manager pod is running.

When we set the **nodeSelector** helm chart parameter for either onedb or onedbcm OneDB will use requiredDuringSchedulingIgnoredDuringExecution and Node affinity is enabled. This will require that all Pods be scheduled on nodes that have been labeled accordingly.

Example Labeling of Nodes:

```
kubectl label nodes gke-worker2 type=database –overwrite
kubectl label nodes gke-worker4 type=database –overwrite

kubectl label nodes gke-worker3 type=cm –overwrite
kubectl label nodes gke-worker5 type=cm –overwrite
```

With the above helm chart values set the OneDB Database server pods must run on a kubernetes nodes that are labled with **type:database**, and OneDB Connection manager pods must run on kubernetes nodes that are labeled with **type:cm**.

OneDB SQL Data store sets up an HA cluster with an HDR primary and HDR secondary. If nodeSelectorRequired is set to true, then we must have more than 1 node labeled when use nodeSelector. The same applies to the OneDB Connection manager based on how many replicas are running.

Note: When configuring pod scheduling it is important to have a good understanding of how this works or you may run into a situation where a pod is not able to be scheduled.

Taints and Toleration

While Node affinity is a property of a pod that attracts them to a set of nodes either as a preference or hard requirement. Taints are the opposite, in that the allow a node to repel a set of pods. A taint is defined on a pod.

```
kubectl taint nodes gke-worker2 type=onedb:NoSchedule
```

This uses a key/value pair in this example we used **type=onedb**, with the NoSchedule effect. This means that no pod will be able to schedule onto the node (gke-worker2) unless it has a matching toleration.

You would then need to use the **tolerations** helm chart parameter override and set the following:

```
tolerations:
-  key: “type”
   operator: “Exists”
   effect: “NoSchedule”
```

Note: Using a combination of Affinity/Anti-Affinity and taints and tolerations, you can control what nodes OneDB SQL Data store will be schedule on and dictate that those nodes are only used for OneDB.

OneDB Configuration

There are five helm charts for OneDB.

- **OneDB SQL Data Store (onedb-sql):** This is the Helm chart that contains the OneDB Database server. It uses a kubernetes operator to deploy a statefulset in a kubernetes environment.
- **OneDB Rest Data Store (onedb-rest):** This is a helm chart that deploys the OneDB Database server with the REST listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Document Data Store (onedb-mongo):** This is a helm chart that deploys the OneDB Database server with the Mongo listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Explore (onedb-explore):** This is a Helm chart that contains only the OneDB Explore UI.
- **OneDB Product (onedb-product):** This is a Helm chart that contains OneDB SQL Data Store, OneDB Rest, OneDB Document and OneDB Explore all as subcharts.

Each chart has a list of configuration options that can be set to specify how the OneDB Helm chart will be installed, setup and configured.

OneDB SQL Data Store Configuration

To customize the installation and configurations, see the list of configuration parameters available to the OneDB SQL Data Store.

List of OneDB SQL Data Store Configuration Parameters

Parameter	*Description*	*Value*
global.hclImagePullSecret	Your own secret with your credentials to HCL's Docker repository. Required when deploying solution in your own cluster.	''
global.isOpenShift	Set to true if using Openshift	false
hclFlexnetURL	Your HCL FlexNet license server URL for your HCL entitlements. Required when deploying in your own cluster	''
hclFlexnetID	Your HCL FlexNet license ID fo ryour HCL entitlements. Required when deploying solution in your own cluster.	''

Parameter	*Description*	*Value*
nfsserver.enabled	Set this attribute to ‘true’ to mount ReadWriteMany PVC across all OneDB server pods. Required for OnBar storage space and logical log backups. If this attribute value is set to ‘false’, then rest of the “nfsserver” related helm attributes are ignored.	false
nfsserver.volumeSize	Size of the Volume used for backups and other shared files.	50G
nfsserver.googleFilestore.enable	Set to true to enable Google Filestore	false
nfsserver.googleFilestore.filestoreIP	IP address of the Filestore instance	‘’
nfsserver.googleFilestore.filestoreShare	Name of the File share on the instance	‘’
nfsserver.awsEFS.enable	Set to true to enable AWS Elastic filestore	false
nfsserver.awsEFS.EFSServer	DNS name of the file system	fs-XXXXX.efs-us-west-2.amazonaws.com
nfsserver.awsEFS.EFSPath	NFS Mount path	/
nfsserver.azureFS.enable	Set to true to enable Azure File share	false
nfsserver.azureFS.secretName	Kubernetes secret name to use	‘’
nfsserver.azureFS.shareName	Azure file share name	‘’

Parameter	*Description*	*Value*
nfsserver.other.enable	Set to true to enable	true
nfsserver.other.storageClass	Set this to the storageClass of the NFS	nfs
tls.config	Set to true to enable TLS communication	false
tls.tlskey	Base64 value of server private key	tlskey: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS ...
tls.tlscert	Base64 value of signed server certificate	tlscert: LS0tLS1CRUdJTiBDRVJUSUEDFRQ0FURS ...
tls.tlscacert	Base64 value of certificate authority root certificate	tlscacert: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS ...
autoscaling.enabled	Set to true to enable auto scaling. To use auto scaling make sure to set onedb.serverReplicaCount, onedb.maxReplicaCount and onedb.resources appropriately.	false
autoscaling.targetCPUUtilizationPercentage	Set to the Percentage when autoscaling should occur.	70
onedb.serverReplicaCount	Set to the number of servers to start for an HA cluster	2
onedb.maxReplicaCount	Set to the max value of servers you would want to configure in the HA cluster	10
onedb.dbsapwd	OneDB Server DBSA (onedbsa) user password	onedb4ever

Parameter	*Description*	*Value*
onedb.backupTag	Set to unique value in case the backup device is shared with other OneDB HA Cluster	onedbbackup-myuniquetag
onedb.encryptionAtRest	Set to true to enable Encryption at rest	false
onedb.exploreAgent	Set to true to start the OneDB Explore Agent on each server	false
onedb.dataStorageClass	Set to the cloud vendor default storage class. Low latency disk I/O storage is recommended. For GKE “standard” is the default	""
onedb.dataStorageSize	Set the persistent Volume Size	10gi
onedb.dataStorageCount	Number of persistent volume’s to provision	2
onedb.restoreFromBackup	Set to true when a restore from last backup is needed	false
onedb.restoreTimestamp	set to specific point in time to perform a point in time restore. Ex 2021-05-11 11:35:00	‘’
onedb.nodeSelectorRequired	Set to true to enforce that no two OneDB Server pods are scheduled on the same K8s node	true
onedb.nodeSelector	Set to a node label to schedule OneDB server pods on preconfigured set of K8s nodes. Set your own keyvalue pair for the node selector	‘’
onedb.tolerations	Used to assist the K8s schedule	{}
onedb.resources	Resources like cpu and memory requested from the cluster.	{}
onedb.customServerEnv	Allows you to set additional environment variables to be used by the database server.	‘’

Parameter	*Description*	*Value*
onedb.customConfig	Allows you to set ONCONFIG parameters to be used by the Database server.	MULTIPROCESSOR: 1
onedb.customSpace	Allows you to create custom Dbspaces in the database server	''
onedb.appUsers	Allows you to create custom Users in the database server	''
onedb.customInitSQL	Allows you to create a script of SQL statements that is run after server initialization.	''
onedb.customInitImage	Allows you to create an Init Container image	''
onedb.customInitImageCmd	The command to run from the custom Init container	''
onedb.groupName	Unique name for each cluster if Setting up Enterprise Replication	g_cdr1
onedb.groupID	Unique ID for each cluster if setting up Enterprise Replication	1
onedb.updateStrategyType	This attribute controls OneDB server statefulset update strategy. Supported values are “RollingUpdate” and “OnDelete”. **OnDelete** With OnDelete update strategy, the StatefulSet controller will not automatically update the Pods in a StatefulSet. Users must manually delete Pods to cause the controller to create new Pods that reflect modifications made to a StatefulSet. **RollingUpdate** The RollingUpdate update strategy implements automated, rolling update for the Pods in a StatefulSet. This is the default update strategy. The RollingUpdate process will start from the highest pod ordinal index to the lowest pod ordinal index. For example, onedb-server-1 is updated before onedb-server-0.	RollingUpdate
onedbcmReplicaCount	Number of Connection Managers to start for the HA cluster	2
onedbcm.serviceType	Set the service type of the connection manager. (ClusterIP, LoadBalancer or NodePort)	ClusterIP

Parameter	*Description*	*Value*
onedbcm.sla_policy	Set the service level agreement of the connection manager. (ROUNDROBIN, WORKLOAD)	ROUNDROBIN
onedbcm.autofailover	If set to false then autofailover is disabled	true
onedbcm.nodeSelectorRequired	Set to true to enforce that no two OneDB CM pods are scheduled on the same K8s node.	true
onedbcm.nodeSelector	Set to a node label to schedule OneDB pods on preconfigured set of K8s nodes. Set your own keyvalue pair.	''
onedbcm.tolerations	Used to assist the K8s schedule	{}
onedbcm.resources	Resources like cpu and memory, requested from the cluster	{}
onedbcm.updateStrategyType	This attribute controls OneDB Connect (Connection Manager) statefulset update strategy. Supported values are “RollingUpdate” and “OnDelete”. **OnDelete** With OnDelete update strategy, the StatefulSet controller will not automatically update the Pods in a StatefulSet. Users must manually delete Pods to cause the controller to create new Pods that reflect modifications made to a StatefulSet. **RollingUpdate** The RollingUpdate update strategy implements automated, rolling update for the Pods in a StatefulSet. This is the default update strategy. The RollingUpdate process will start from the highest pod ordinal index to the lowest pod ordinal index. For example, onedbcm-1 is updated before onedbcm-0.	RollingUpdate

Customize Server configuration

The ONCONFIG file is used by the database server during initialization to setup the data store server. Use the ***customConfig*** helm chart configuration parameter to specify ONCONFIG parameters. It can be configured as follows. With parameters that are not unique specify a number after the parameter as seen below with BUFFERPOOL# .

```
onedb:
    customconfig:
          MULTIPROCESSOR: “1”
          BUFFERPOOL1: “size=8k,buffers=50000,lrus=8,lru_min_dirty=50,lru_max_dirty=60.5”
          BUFFERPOOL2: “size=2k,buffers=200000,lrus=8,lru_min_dirty=50,lru_max_dirty=60.5”
          LOGSIZE: “10000”
```

Create Initialization SQL script

The helm chart configuration parameter ***customInitSQL*** can be used to create an SQL scrpit that will run by the OneDB server after first initialization. This script can be used to perform needed setup tasks, creation of databases, etc.

```
onedb:
    customInitSQL:  |-
        database sysadmin;
        create database test with log;
        create table t1 (col1 int, col2 int);
```

Creating custom spaces

The helm chart configuration parameter ***customSpace*** can be used to create and setup spaces. Following table details the options available for the creation of spaces. When defining the customSpace parameter, you must create a well formed json document.

Parameter	***Description***	***Example Value***
name	The name of the space	my_data_dbspace
type	The type of space to create. Supported values are: dbspace: normal dbspace llog : logical log dbspace plog: physical log dbspace sbspace: smart blobspace tempdbspace: temporary dbspace tempsbspace: temporary smart blobspace	dbspace
pagesize	The size of the space, supported values are 2k,4k,6k,8k,16k	4k
size	Size of the space, supported values are GB, MB, KB	10GB
logging	Used for smart blobspaces to enable loging 1: enable logging 0: disable logging	1

```
onedb:
    customSpace: >-
        [
            {“name”:”datadbs”, “type”: “dbspace”, “pagesize”: “4k”, “size”: “4GB” },
            {“name”:”logdbs”, “type”: “llog”, “size”: “2GB” },
            {“name”:”plogdbs”, “type”: “plog”, “size”:  “4GB” },
            {“name”:”sbspace1”, “type”: “sbspace”, “size”: “1GB” , “logging”: 1},
```

```
            {“name”:”tmpdbspace1”, “type”: “tempdbspace”,”pagesize”: “4k”, “size”: “1GB” },
            {“name”:”tmpsbsp1”, “type”: “tempsbspace”, “size”: “500MB” }
        ]
```

Creating custom users

The helm chart configuration parameter ***appUsers*** can be used to create additional users. Following table details the options available for the creation of users. Currently, the only type of user support is an operating system user account. When using appUsers, you must create a well formed json document.

Parameter	***Description***	***Example Value***
user	The name of the user	appuser1
password	The password of the user	passw0rd
group	A group name to create for the user.	dev
uid	The user id number to use for the user	1003
gid	The group id number to to use for the group	2000
type	The type of user to create. Currently only osuser is supported	osuser

```
onedb:
    appUsers: >-
        [
            { “user”:”appuser1”, “password”: “passw0rd”, “group”:”dev”,
 “uid”:1003,”gid”:2000,”type”:”osuser” },
            { “user”:”appuser2”, “password”: “passw0rd”, “group”:”dev”,
 “uid”:1003,”gid”:2000,”type”:”osuser” }
        ]
```

Setting additional server Environment

The helm chart configuration parameter ***customServerEnv*** can be used to set additional server environment variables. This will be set in the environment script when initialization and starting the OneDB database server.

```
onedb:
    customServerEnv:
        DB_LOCALE: “en_us.utf8”
        DBTEMP: “/tmp”
```

Using an Init container

The helm chart configuration parameters ***customInitImage*** and ***customInitImageCmd*** can be used to create an Init container to perform setup steps prior to the startup of the OneDB server container image. The customInitimage parameter is used to specify an image to use and the customInitImageCmd is the command to run inside the image.

The Init container image can be a purposely built image with scripts built in. Or it can be a generic image with specific OS commands to run.

```
onedb:
    customInitImage: “gcr.io/<my-images>/busybox-custom:latest”
    customInitImageCmd: “/bin/initSetup.sh”
```

Scheduling of K8s pods

The helm chart configuration parameter ***nodeSelector*** for onedb and onedbcm are used to support Node affinity. It allows you to select a preconfigured set of K8s nodes to run on.

The following example will run the OneDB server on nodes labeled as onedb and the OneDB Connection manager on nodes labeled as onedbcm.

```
onedb:
    nodeSelector:
        database: onedb
onedbcm:
    nodeSelector:
        cm: onedbcm
```

The helm charts have an unconfigured parameter ***tolerations*** to allow for full configuration of taints and tolerations for K8s scheduling of pods. This can be used to specify a node taint, which means no pod can be scheduled on the node unless it has a matching toleration. Then a OneDB server is labeled with a toleration to allow it to run on the tainted nodes.

```
kubectl taint nodes node1 tainted4onedb=onedb-only:NoSchedule
```

```
onedb:
    tolerations:
    - key: “tainted4onedb”
      operator: “Exists”
      effect: “NoSchedule”
```

Sample helm override file

When specifying helm chart parameters, you can specify them on the command line. When specifying a number of parameters it is sometimes more convenient to create a file with the override parameters. The following example shows a single file that uses customServerEnv, appUsers and customInitSQL in a single file.

```
FILE:  onedb.override.yaml
onedb:
    customServerEnv:
        ONEDB_USER_MANAGEMENT: “true”
        ONEDB_USER: “user1”

    appUsers: >-
        {“user”:”user1”, “password”:”Passw0rd”, “group”:”dba”, “uid”: 1005, “gid”:2001,
```

```
        “type”:”osuser”}

   customInitSQL:  |-
       create database stores with log;
       create user dbauser with password ‘Passw0rd’ account unlock properties user ‘user1’
       authorization(dbsa);
```

The above yaml file sets two environment variable that are used in the container Image to enable database users. It then creates one os user to be used as the operating system user that the created database user will have permissions as.

OneDB REST Data Store Configuration

To customize the installation and configurations see the list of configuration parameters available to the OneDB REST Data Store.

List of OneDB REST Data Store Configuration Parameters

Parameter	***Description***	***Value***
global.hclImagePullSecret	Your own secret with your credentials to HCL’s Docker repository. Required when deploying solution in your own cluster.	‘’
hclFlexnetURL	Your HCL FlexNet license server URL for your HCL entitlements. Required when deploying in your own cluster	‘’
hclFlexnetID	Your HCL FlexNet license ID for your HCL entitlements. Required when deploying solution in your own cluster.	‘’
resources	Resources like cpu and memory, requested from the cluster.	requests.cpu: “100m”, requests.memory: “128mi”
config	Setting advanced options in the application’s yaml configuration file.	‘’
customEnv	Allows you to set additional environment variables to be used by the OneDB REST container image.	‘’
externalDBUrl	A custom external JDBC style URL if you want to connect to a OneDB server that is not part of the solution	‘’
databaseuser	The user the REST API will use to connect to the OneDB Database Server	onedbsa
databasePassword	The password the REST API will use to connect to the OneDB Database Server	onedb4ever

Custom REST Configuration

Additional configuration can be added to the REST service as follows. Review the product documentation for all available options for the REST configuration file.

```
onedb-rest:
    config: |-
        rest.session.timeout 600000
        security.csrf.token.enable: true
```

```
onedb-rest:
    customEnv:
        TZ: CST2
```

OneDB Document Data Store Configuration

To customize the installation and configurations see the list of configuration parameters available to the OneDB Document Data Store.

List of OneDB Document Data Store Configuration Parameters

Parameter	***Description***	***Value***
global.hclImagePullSecret	Your own secret with your credentials to HCL's Docker repository. Required when deploying solution in your own cluster.	''
hclFlexnetURL	Your HCL FlexNet license server URL for your HCL entitlements. Required when deploying in your own cluster	''
hclFlexnetID	Your HCL FlexNet license ID for your HCL entitlements. Required when deploying solution in your own cluster.	''
resources	Resources like cpu and memory, requested from the cluster	requests.cpu: "100m", requests.memory: "128mi"
config	Setting advanced options in the application's yaml configuration file	''
customEnv	Allows you to set additional environment variables to be used by the OneDB Mongo container image.	''
externalDBUrl	A custom external JDBC style URL if you want to connect to a OneDB server that is not part of the solution	''
mongoUser	The mongo username	mongo
mongoPassword	Password for the mongo user	mongoPassword
databaseuser	The user the MongoDB API will use to connect to the OneDB Database Server	onedbsa
databasePassword	The password the MongoDB API will use to connect to the OneDB Database Server	onedb4ever

Custom Mongo Configuration

Additional configuration can be added to the Document Data Store (Mongo) service as follows. Review the product documentation for all available options for the Mongo configuration file.

```
onedb-mongo:
    config:  |-
        security.sql.passthrough=true
```

```
onedb-mongo:
    customEnv:
        TZ: CST2
```

OneDB Explore Data Configuration

To customize the installation and configurations see the list of configuration parameters available to OneDB Explore Data.

List of OneDB Explore Configuration Parameters

Parameter	*Description*	*Value*
global.hclImagePullSecret	Your own secret with your credentials to HCL's Docker repository. Required when deploying solution in your own cluster.	''
resources	Resources like cpu and memory, requested from the cluster	requests.cpu: "100m", requests.memory: "128mi"
config	Setting advanced options in the application's yaml configuration file	''
customEnv	Allows you to set additional environment variables to be used by the OneDB Explore container image.	''
adminPassword	Initial admin password	testPassw0rd

Custom Explore Configuration

Additional configuration can be added to the Explore service as follows. Review the product documentation for all available options for the Explore configuration file.

```
onedb-explore:
    config:  |-
        key=value
```

```
onedb-explore:
    customEnv:
        TZ: CST2
```

OneDB Product Configuration

The majority of configuration will happen through the helm charts for OneDB SQL Data store, OneDB Document Data store, OneDB Rest Data Store or OneDB Explore. To customize the installation and configurations, following is the list of configuration parameters:

Table 1. List of OneDB Product Configuration Parameters

Parameter	Description	Value
enableChart.onedbMongo	Enable or disable the onedb-mongo chart for installation. Default: true	'true'
enableChart.onedbRest	Enable or disable the onedb-rest chart for installation. Default: true	'true'
enableChart.onedbExplore	Enable or disable the onedb-explore chart for installation. Default: true	'true'
enableChart.onedbSql	Enable or disable the onedb-product chart for installation. Default: true	'true'

Configuring TLS

Use transport layer security (TLS) to create secure connections from OneDB clients to the OneDB database server. By default, TLS is disabled. To enable TLS connections, set the **tls.tlsconfig** helm chart parameter value to **true**.

The following helm chart parameters also need to be set:

- **tlskey**: The base64 encoded value of the private key.
- **tlscert**: The base64 encoded value for the Public signed server certificate.
- **tlscacert**: The base64 encoded value of the certificate authority root certificate.

Example tls configuration:

```
tls:
    tlsconfig: true
    tlskey:  LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t....
    tlscert: LS0tLS1CRUdJTiBABEFUSUZJQ0FURS0tLS0t...
    tlscacert: LS0tLS1CRUdJTiBDRVJUSUEEFZFURS0tLS0t....
```

Create TLS Certificates

About this task

You can obtain your own certificates from a certificate authority or you can create your own with the following steps using openssl:

1. Generate root CA private key PEM file:

   ```
   openssl genrsa -out rootCA.key.pem
   ```

2. Create a self signed root CA certificate in PEM file:

```
openssl req -new -x509 -key rootCA.key.pem -subj "/C=US/ST=Kansas/L=Olathe/O=HCL/OU=OneDB" -days 3650
 -out
 rootCA.cert.pem
```

3. Generate server private key:

```
openssl genrsa -out server.key.pem
```

4. Generate a certificate signing request (CSR) for OneDB Server:

```
openssl req -new -key server.key.pem -subj
 "/C=US/ST=Kansas/L=Olathe/O=HCL/OU=OneDB/CN=Server/emailAddress=onedb@hcl.com" -out server.req.pem
```

5. Sign certificate with root CA:

```
openssl x509 -req -inform PEM -in server.req.pem -set_serial 1 -CA
rootCA.cert.pem -CAkey rootCA.key.pem -days 3650 -extensions usr_cert -outform PEM -out server.cert.pem
```

6. Convert rootCA.cert.pem to base64 -> tlscacert:

```
base64 rootCA.cert.pem -w 0 > tlscacert
```

7. Convert server.cert.pem to base64 -> tlscert:

```
base64 server.cert.pem -w 0 > tlscert
```

8. Convert server.key.pem to base64 -> tlskey:

```
base64 server.key.pem -w 0 > tlskey
```

Connect from Java client with TLS

About this task

To connect to the OneDB Databaser server with a Java client (JDBC) with TLS you must create a keystore for the client application to use. You need the root CA certificate and will use this file **rootCA.cert.pem** to generate the kesytore.

Create the keystore:

```
keytool -import -file rootCA.cert.pem -keystore ssl.keystore
```

Example

Example OneDB JDBC URL to connect to a OneDB Database server using TLS:

```
jdbc:onedb://XX.XXX.XXX.XX.nip.io:10001/sysmaster;user=onedbsa;password=xxxxxxx;ENCRYPT=true;TRUSTSTORE=./ssl.k
eystore;TRUSTSTOREPASSWORD=xxxxxxx;
CERTIFICATEVERIFICATION=false;loginTimeout=0
```

For more information on connecting JDBC applications with TLS, see HCL OneDB JDBC Driver Guide.

Accessing OneDB

Connecting to the OneDB database server is essential. OneDB allows for connections from Mongo Clients, REST Clients and Native SQLI clients. Ex. JDBC, ODBC, ESQL/C

Connectivity can occur from inside the cluster or from outside the cluster. By default, connections from outside the cluster are not enabled.

Connectivity will be different based on the installation. The two basic installations are:

1. Installation of the Standalone Helm Chart of OneDB
2. Installation of a Solution Factory Helm Chart of OneDB

The OneDB Connection Manager is used for Native SQLI Connections to the Database server. There is a Kubernetes service provided to handle this connectivity.

REST, Mongo and OneDB Explore will each have a Kubernetes service to handle their own connections.

```
   NAME                       TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)               AGE
helm-1-odbp-explore            ClusterIP      10.96.220.30       <none>       8080/TCP              20m
helm-1-odbp-mongo              ClusterIP      10.96.44.208       <none>       27017/TCP             20m
helm-1-odbp-rest               ClusterIP      10.96.90.21        <none>       8080/TCP              20m
onedbcm-cm-service             ClusterIP      10.96.159.103      <none>       10000/TCP,20000/TCP   19m
```

With the OneDB Connection Manager you will see a pattern emerge that will describe what type of connection will occur.

Port Number	Description
10XXX	Internal (Redirected) Connection
20XXX	External (Proxied) Connection
XXXX0	Connection to the HA Primary Server
XXXX1	Connection to the HDR Secondary Server
XXXX2	Connection to any server in the HA Cluster
XXXX3	Connection to the RS Secondary Server
XXX2X	Connection that uses SSL

Example: Port 10021 is an internal Connection (10XXX) to the HA Secondary server (XXXX1) using SSL (XXX2X).

Standalone OneDB Chart

A standalone OneDB helm chart does not include elements of a Solution Factory helm chart. There are mutliple different Standalone helm charts.

- **OneDB SQL Data Store (onedb-sql):** This is the Helm chart that contains the OneDB Database server. It uses a kubernetes operator to deploy a statefulset in a kubernetes environment.
- **OneDB Rest Data Store (onedb-rest):** This is a helm chart that deploys the OneDB Database server with the REST listener. The OneDB SQL Data Store is a subchart in this chart.

- **OneDB Document Data Store (onedb-mongo):** This is a helm chart that deploys the OneDB Database server with the Mongo listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Explore (onedb-explore):** This is a Helm chart that contains only the OneDB Explore UI.
- **OneDB Product (onedb-product):** This is a Helm chart that contains OneDB SQL Data Store, OneDB Rest, OneDB Document and OneDB Explore all as subcharts.

The OneDB helm(s) chart can be configured to only allow connections from within the cluster itself, or they can be configured to allow for connections from outside the cluster.

To allow for connections from outside the cluster the kubernetes service types should be configured as Loadbalancer, or an extra piece of software can be used to provide a single point of ingress into the cluster. Some commonly used ingress/loadbalancer's are Ambassodor, NGINX.

Setting up Ambassador or NGINX is outside the scope of this documentation, instead we will use a Loadbalancer service type.

Connecting from Inside the cluster

OneDB Native SQLI connection are accessible through the OneDB Connection Manager. A kubernetes service will be created with the deployment and that service name is used for connections. A kubernetes service is created for the non-SQLI types as well.

```
kubectl get services

    NAME                      TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)               AGE
helm-1-odbp-explore        ClusterIP         10.96.220.30      <none>       8080/TCP               20m
helm-1-odbp-mongo          ClusterIP         10.96.44.208      <none>       27017/TCP              20m
helm-1-odbp-rest           ClusterIP         10.96.90.21       <none>       8080/TCP               20m
onedbcm-cm-service         ClusterIP         10.96.159.103     <none>       10000/TCP,20000/TCP    19m
```

The OneDB Connection manager supports "Redirected" and "Proxied" connections. For internal connections it is recommended to use "Redirected" connections. The following table shows the Internal connection string to use for each driver type. From within the cluster the **.{namespace}.svc.cluster.local** may not be needed from the URL below.

Driver	URL	Example URL
OneDB driver (SQLI-Primary)	onedbcm-cm-service.{namespace}.svc.cluster.local:10000	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary)	onedbcm-cm-service.{namespace}.svc.cluster.local:10001	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any)	onedbcm-cm-service.{namespace}.svc.cluster.local:10002	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Primary-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10020	jdbc:onedb://{url}/sysmaster

OneDB driver (SQLI-Secondary-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10021	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10022	jdbc:onedb://{url}/sysmaster
Mongo combatible driver	<Release-Name>-odbp-mongo:27017	mongodb://<release-name>-odbp-mongo:27017
REST	<Release-Name>-odbp-rest:8080	http://<release-name>-odbp-rest:8080
Explore	<Release-Name>-odbp-explore:8080	http://<release-name>-odbp-explore:8080

Connecting from Outside the cluster

OneDB Native SQLI connection are accessible through the OneDB Connection Manager. A kubernetes service will be created with the deployment and that service name is used for connections. A kubernetes service is created for the non-SQLI types as well.

```
kubectl get services

NAME                      TYPE           CLUSTER-IP       EXTERNAL-IP            PORT(S)
 AGE
helm-1-odbp-explore       LoadBalancer   10.96.220.30     172.19.255.202         8080/TCP            20m
helm-1-odbp-mongo         LoadBalancer   10.96.44.208     172.19.255.201         27017/TCP           20m
helm-1-odbp-rest          LoadBalancer   10.96.90.21      172.19.255.200         8080/TCP            20m
onedbcm-cm-service        LoadBalancer   10.96.159.103    172.19.255.203         10000/TCP,20000/TCP     19m
```

The OneDB Connection manager supports “Redirected” and “Proxied” connections. For external connections you must use the “Proxied” connections. The following table shows the external connection string to use for each driver type.

Driver	URL	Example URL
OneDB driver (SQLI-Primary)	{LoadBalancer External IP address}:20000	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary)	{LoadBalancer External IP address}:20001	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any)	{LoadBalancer External IP address}:20002	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Primary-SSL)	{LoadBalancer External IP address}:20020	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary-SSL)	{LoadBalancer External IP address}:20021	jdbc:onedb://{url}/sysmaster

OneDB driver (SQLI-Any-SSL)	{LoadBalancer External IP address}:20022	jdbc:onedb://{url}/sysmaster
Mongo combatible driver	{LoadBalancer External IP address}:27017	mongodb://{url}:27017
REST	{LoadBalancer External IP address}:8080	http://{url}:8080
Explore	{LoadBalancer External IP address}:8080	http://{url}:8080

Setting LoadBalancer Type

The default setting for each service type is ClusterIP. This will only allow internal connections. The OneDB helm chart service types of interest are listed below. NOTE: The names of the services may be slightly different when installing onedb-mongo, onedb-rest, onedb-sql chart.

- onedbcm-cm-service
- <Release.Name>-odbp-explore
- <Release.Name>-odbp-mongo
- <Release.Name>-odbp-rest

Each of these service types can be configured as a LoadBalancer to provide external connectivity.

To set Loadbalancer for the Connection Manager:

```
onedb-sql:
    onedbcm:
        serviceType: LoadBalancer
```

To set LoadBalancer for Mongo

```
onedb-mongo:
    service:
        type: LoadBalancer
```

To set LoadBalancer for REST

```
onedb-rest:
   service:
      type: LoadBalancer
```

To set LoadBalancer for Explore

```
onedb-explore:
      service:
         type: LoadBalancer
```

Solution Factory OneDB Chart

A Solution factory OneDB helm chart contains elements of the Solution factory including things like a Console UI, grafana, prometheus. The OneDB helm chart is included as a subchart of the overall Helm chart. There are multiple different Solution Factory charts that include different aspects of OneDB.

- **OneDB SQL Data Store (onedb-sql):** This is the Helm chart that contains the OneDB Database server. It uses a kubernetes operator to deploy a statefulset in a kubernetes environment.
- **OneDB Rest Data Store (onedb-rest):** This is a helm chart that deploys the OneDB Database server with the REST listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Document Data Store (onedb-mongo):** This is a helm chart that deploys the OneDB Database server with the Mongo listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Explore (onedb-explore):** This is a Helm chart that contains only the OneDB Explore UI.
- **OneDB Product (onedb-product):** This is a Helm chart that contains OneDB SQL Data Store, OneDB Rest, OneDB Document and OneDB Explore all as subcharts.

The Solution Factory OneDB helm(s) chart can be configured to only allow connections from within the cluster itself, or they can be configured to allow for connections from outside the cluster.

To allow for connections from outside the cluster, a Solution factory OneDB helm chart includes and configures Ambassador. The ambassador LoadBalancer will handle connectivity into the kubernetes cluster for Mongo, REST and Explore. The OneDB Connection Manager will handle connections into the kubernetes cluster for Native SQLI clients (ex. JDBC, ODBC, ESQL/C)

By default, OneDB Connection Manager does not allow for external connections.

Connecting from Inside the cluster

OneDB Native SQLI connection are accessible through the OneDB Connection Manager. A kubernetes service will be created with the deployment and that service name is used for connections. The ambassador service is created for the non-SQLI connections. The ambassador service is setup as a Loadbalancer type where as the OneDB Connection Manager has a default setting of ClusterIP.

```
kubectl get services

  NAME                  TYPE               CLUSTER-IP         EXTERNAL-IP            PORT(S)
 AGE
sofy-1-ambassador       LoadBalancer         10.96.224.175     172.19.255.200         80:32653/TCP,44
   68m
sofy-1-odbp-mongo       ClusterIP            10.96.183.135     <none>                 27017/TCP
  69m
sofy-1-odbp-rest        ClusterIP            10.96.173.88      <none>                 8080/TCP
   69m
sofy-1-odbp-explore     ClusterIP            10.96.66.46       <none>                 8080/TCP
  71m
onedbcm-cm-service      ClusterIP            10.96.33.166      <none>                 10000:30248/TCP
  77m
```

The OneDB Connection manager supports “Redirected” and “Proxied” connections. For internal connections it is recommended to use “Redirected” connections. The following table shows the Internal connection string to use for each driver type. From within the cluster the **.{namespace}.svc.cluster.local** may not be needed from the URL below:

Driver	URL	Example URL
OneDB driver (SQLI-Primary)	onedbcm-cm-service.{namespace}.svc.cluster.local:10000	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary)	onedbcm-cm-service.{namespace}.svc.cluster.local:10001	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any)	onedbcm-cm-service.{namespace}.svc.cluster.local:10002	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Primary-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10020	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10021	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10022	jdbc:onedb://{url}/sysmaster
Mongo combatible driver	<Release-Name>-odbp-mongo:27017	mongodb://<release-name>-odbp-mongo:27017
REST	<Release-Name>-odbp-rest:8080	http://<release-name>-odbp-rest:8080
Explore	<Release-Name>-odbp-explore:8080	http://<release-name>-odbp-explore:8080

Connecting from Outside the cluster

OneDB Native SQLI connection are accessible through the OneDB Connection Manager. A kubernetes service will be created with the deployment and that service name is used for connections. The ambassador service is created for the non-SQLI connections. The ambassador service is setup as a Loadbalancer type where as the OneDB Connection Manager has a default setting of ClusterIP. To allow external SQLI connectivity you must set the service type to LoadBalancer.

```
kubectl get services

 NAME                  TYPE           CLUSTER-IP       EXTERNAL-IP          PORT(S)           AGE
sofy-1-ambassador     LoadBalancer   10.96.224.175    172.19.255.200       80:32653/TCP,44     68m
sofy-1-odbp-mongo     ClusterIP      10.96.183.135    <none>               27017/TCP           69m
sofy-1-odbp-rest      ClusterIP      10.96.173.88     <none>               8080/TCP            69m
sofy-1-odbp-explore   ClusterIP      10.96.66.46      <none>               8080/TCP            71m
onedbcm-cm-service    LoadBalancer   10.96.33.166     172.19.255.201       10000:30248/TCP     77m
```

The OneDB Connection manager supports “Redirected” and “Proxied” connections. For external connections you must use the “Proxied” connections. The following table shows the external connection string to use for each driver type.

Driver	URL	Example URL
OneDB driver (SQLI-Primary)	{LoadBalancer External IP address}:20000	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary)	{LoadBalancer External IP address}:20001	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any)	{LoadBalancer External IP address}:20002	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Primary-SSL)	{LoadBalancer External IP address}:20020	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary-SSL)	{LoadBalancer External IP address}:20021	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any-SSL)	{LoadBalancer External IP address}:20022	jdbc:onedb://{url}/sysmaster
Mongo combatible driver	{LoadBalancer External IP address}:27017	mongodb://{url}:27017
REST	{LoadBalancer External IP address}:8080	http://{url}:8080
Explore	{LoadBalancer External IP address}:8080	http://{url}:8080

Setting LoadBalancer Type

The default setting for the OneDB Connection Manager service type is ClusterIP. This will only allow internal connections. To allow for connectivity from outside the cluster you must set the service type for the OneDB Connection Manager to LoadBalancer.

To set Loadbalancer for the Connection Manager:

```
onedb-product:
    onedbcm:
        serviceType: Loadbalancer
```

Connection credentials

The following table shows the default connection credentials. This can be changed accordingly.

Product/Driver	User	Password
Explore	admin	testPassw0rd
REST	-	-

Mongo	mongo	mongoPassword
SQL Data Store	onedbsa	onedb4ever

The REST connection uses a database connection. You can connect with onedbsa or any other use that was created in the OneDB SQL Data Store.

To change the onedbsa password for the OneDB SQL Data Store use the following configuration override values. If you change this password it is important that you make changes to mongo, rest and explore.

```
onedb-sql:
    onedb:
        dbsapwd: one1dba4ever
```

If you change the passowrd for OneDB SQL Datastore (onedb-sql), you must tell the mongo,rest and explore charts how to connect to the OneDB SQL Datastore (onedb-sql). See the following configuration overrides.

```
onedb-mongo:
    databasePassword:  one1dba4ever

onedb-rest:
    databasePassword:  one1dba4ever

onedb-explore:
    serverConnection:
        password: one1dba4ever
```

To change the user and password for OneDB Mongo use the following configuration override values. If you change this password it is important that you make changes to mongo, rest and explore.

```
onedb-mongo:
    mongoUser: mymongo
    mongoPassword: mongoPassword
```

To change the admin password for OneDB Explore. Use the following configuration override values. If you change this password it is important that you make changes to mongo, rest and explore.

```
onedb-explore:
    adminPassword: newPassw0rd
```

OneDB Connection Manager External Service

Purpose of External CM Service

When you designate the default onedbcm-cm-service as a **LoadBalancer** you expose all ports outside the cluster. This may not be desirable because the non-SSL ports are also exposed outside the cluster.

```
onedb-sql:
    onedbcm:
        serviceType: Loadbalancer
```

Configuration of External CM Service

The following configuration options are available for the external CM service:

Parameter	Description	Value
onedbcm_ext.enabled	Set to true to enable the External CM service	'false'
onedbcm_ext.serviceType	Set the serviceType for this Service. LoadBalancer or NodePort	20000
onedbcm_ext.ports.olttp	Proxy mode Connection to Primary Server	20000
onedbcm_ext.ports.reportp	Proxy mode Connection to Secondary Server	20001
onedbcm_ext.ports.oltpanyp	Proxy mode Connection to ANY Server	20002
onedbcm_ext.ports.reportrssp	Proxy mode Connection to RSS Server	20003
onedbcm_ext.ports.oltpdrdap	Proxy mode Connection to Primary Server with DRDA protocol	20010
onedbcm_ext.ports.reportdrdap	Proxy mode Connection to Secondary Server with DRDA protocol	20011
onedbcm_ext.ports.oltpdrdaanyp	Proxy mode Connection to ANY Server with DRDA protocol	20012
onedbcm_ext.ports.reportrssdrdap	Proxy mode Connection to RSS Server with DRDA protocol	20013
onedbcm_ext.ports.oltpsslp	Proxy mode Connection to Primary Server using SSL	20020
onedbcm_ext.ports.reportsslp	Proxy mode Connection to Secondary Server using SSL	20021

Parameter	Description	Value
onedbcm_ext.ports.oltpanysslp	Proxy mode Connection to ANY Server using SSL	20022
onedbcm_ext.ports.reportrsssslp	Proxy mode Connection to RSS Server using SSL	20023
onedbcm_ext.ports.oltpdrdasslp	Proxy mode Connection to Primary Server with DRDA protocol using SSL	20030
onedbcm_ext.ports.reportdrdasslp	Proxy mode Connection to Secondary Server with DRDA protocol using SSL	20031
onedbcm_ext.ports.oltpdrdaanysslp	Proxy mode Connection to ANY Server with DRDA protocol using SSL	20032
onedbcm_ext.ports.reportrssdrdasslp	Proxy mode Connection to RSS Server with DRDA protocol using SSL	20033

Example Setup of External CM Service (LoadBalancer)

To setup the OneDB External CM service you would typically set the original onedbcm service so that it can only be accessed from within the cluster. This is done by setting the serviceType to ClusterIP.

Then configure the External CM serivce accordingly. The configuration below exposes only the SSL ports for the traditional SQLI protocol.

```
onedb-sql:
    onedbcm:
        serviceType: ClusterIP

    onedbcm_ext:
        enabled: true
        serviceType: LoadBalancer
        ports:
            oltpsslp:            20020
            reportsslp:          20021
            oltpanysslp:         20022
            reportrsssslp:       20023
```

Below is a list of services showing the onedbcm-cm-service defined as ClusterIP and the onedbcm-ext-service defined as LoadBalancer. You can then access the external IP address from outside the cluster for the configured ports.

```
kubectl get services

NAME                          TYPE                CLUSTER-IP       EXTERNAL-IP            PORT(S)
       AGE
helm-1-odbp-explore         ClusterIP             10.96.220.30      <none>                 8080/TCP
       20m
helm-1-odbp-mongo           ClusterIP             10.96.44.208      <none>                 27017/TCP
       20m
```

```
helm-1-odbp-rest            ClusterIP             10.96.90.21       <none>                 8080/TCP
       20m
onedbcm-cm-service          ClusterIP             10.96.159.103     <none>                10000/TCP,20000/TCP
       19m
onedbcm-ext-service         LoadBalancer          10.96.159.103     172.19.255.203        20020/TCP,20021/TCP
       19m
```

Example Setup of External CM Service (NodePort)

To setup the OneDB External CM service as a NodePort you would typically set the original onedbcm service so that it can only be accessed from within the cluster. This is done by setting the serviceType to ClusterIP.

Then, configure the External CM serivce accordingly. The configuration below exposes only the SSL ports for the traditional SQLI protocol. Depending on the kubernetes platform being used the port numbers from a NodePort service may need to be in a specific range. You can change the port numbers accordingly to fit with the required range.

```
onedb-sql:
    onedbcm:
        serviceType: ClusterIP

    onedbcm_ext:
        enabled: true
        serviceType: NodePort
        ports:
            oltpsslp:           30020
            reportsslp:         30021
            oltpanysslp:        30022
            reportrssslp:       30023
```

Below is the list of services showing the onedbcm-cm-service defined as ClusterIP and the onedbcm-ext-service defined as NodePort. You can access the configured ports through each Nodes IP address.

```
kubectl get services

NAME                          TYPE              CLUSTER-IP      EXTERNAL-IP                PORT(S)
      AGE
helm-1-odbp-explore        ClusterIP          10.96.220.30       <none>                  8080/TCP
      20m
helm-1-odbp-mongo          ClusterIP          10.96.44.208       <none>                 27017/TCP
      20m
helm-1-odbp-rest           ClusterIP          10.96.90.21        <none>                  8080/TCP
      20m
onedbcm-cm-service         ClusterIP          10.96.159.103      <none>                 10000/TCP,20000/TCP
      19m
onedbcm-ext-service        NodePort           10.96.159.103      <none>                 30020:30020/TCP
      19m
```

Administering OneDB

The following information describes the common tasks that an administrator may perform for a OneDB Database server in a kubernetes environment.

- Exec into OneDB pod.

  For some administration tasks, you may need to login to the OneDB pod and run commands. To do this, you must have authorization to run kubectl exec.
- Starting and Stopping OneDB.

  Starting and stopping the OneDB pod from kubernetes
- Viewing OneDB Log files.

  To view the OneDB logs in the different pods.
- Backup and Restore.

  To backup and restore the OneDB database server

Exec into OneDB Pod

There may be a need to login to the kuberneres pod to perform administration tasks. First identify the pod you need to login to.

```
kubectl get pods |grep onedb
onedb-operator-67f5d6cd9b-wxcg2 1/1 Running 0 42h
onedb-server-0                  1/1 Running 0 42h
onedb-server-1                  1/1 Running 0 42h
onedbcm-0                       1/1 Running 0 42h
onedbcm-1                       1/1 Running 0 42h
```

Run the kubectl command to login to the kubernetes pod:

```
kubectl exec --stdin --tty onedb-server-0 -- bash
```

You will be logged in as user “informix” and your environment will be set for the OneDB Database server. Which can be verified with the onstat - command:

```
onstat -
HCL OneDB Server Version 2.0.1.0 -- On-Line (Prim)
 -- Up 1 days 18:42:25 -- 793248 Kbytes
2021-12-01 17:37:54
```

Stop/Start OneDB Database Server

Use one of the following methods to stop and restart the OneDB database server:

1. Delete the kubernetes pod
2. Login to Pod and take the OneDB server offline

Delete Pod

About this task

Deleting a pod is a method that can be used to restart the pod. When a pod is deleted or dies, kubernetes will force the pod to be restarted.

1. Delete the pod of interest.

   ```
   kubectl delete pod onedb-server-0
   ```

2. Monitor the pod that was deleted, as it is restarted. After performing the **kubectl delete pod** the pod will terminate. As it restarts, it will go back into the initialization and eventually a Running state.

   ```
   onedb-server-0                                  0/1     Terminating    0          42h
   onedb-server-0                                  0/1     Init:0/1           0          1s
   onedb-server-0                                  1/1     Running          0          43s
   ```

Login Pod

1. First annotate the pod so kubernetes does not restart the pod while you are performing administration tasks:

   ```
   kubectl annotate pod onedb-server-0 livenessprobe=disabled --overwrite=true
   ```

2. Use the command to take the OneDB Database server offline:

   ```
   onmode -kuy
   ```

3. Perform any adminstration tasks needed with the server offline.
4. Run the command to bring the OneDB Database server back online.

   ```
   oninit
   ```

5. Re-enable the liveness probe:

   ```
   kubectl annotate pod onedb-server-0 livenessprobe=enabled --overwrite=true
   ```

   Note: If you do not disable the liveness probe, once you take the OneDB Databaser server offline. The kubernetes liveness probe will begin to fail. After 3 failures of the liveness probe, kubernetes will restart the pod on its own.

Viewing log files

About this task

Use one of the following methods to view the OneDB Database server logs:

1. Use kubectl to view the pod logs

   ```
   kubectl logs onedb-server-0
   ```

2. View the log files from inside the pod. The example below shows a tail of the online.log. With this method you can view any log file associated with the OneDB Database Server.

```
Exec into the pod
cd $ONEDB_DATA_DIR/logs
tail –f onedb*.logs
```

3. When a pod starts up it will sometimes use an init container to perform setup work prior to the main pod starting.

```
kubectl logs onedb-server-0 –c onedb-init
```

Backup and Restore

For zero data loss configuration, and to protect server from data corruption, OneDB storage spaces must be backed up on a nightly basis, and logical log files must be backed up continuously as and when they get full.

For OneDB backup functionality to work across all available pods, it is recommended to configure ReadWriteMany storage and specify storage configuration details using “nfsserver” helm attributes. If ReadWriteMany storage is not configured, then pod specific ReadWriteOnce storage is used for backup storage.

Backup Schedule

The default behavior is for a level 0 archive to occur every night at 2:30am. The last three backups are retained, and any prior archives are cleaned up and removed.

A restore should only be needed if both the OneDB HA primary and OneDB HA secondary servers are corrupted. If just one or the other is corrupted, then a failover scenario can occur to bring the two servers back into sync.

Change Backup Schedule

About this task

Backups are scheduled by the OneDB Scheduler. **Cloud Backup** scheduler task determines what days and what time the level 0 backups occur.

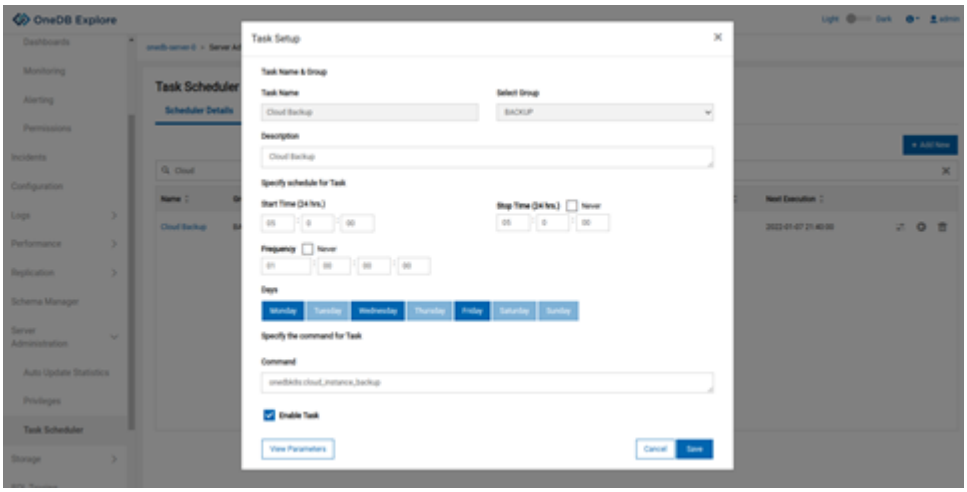
This can be changed by modifying the Scheduler task. You can modify the archive schedule from the command line or you can do this through OneDB Explore.

Using Command Line

1. Exec into onedb-server-0 pod.
2. Use dbaccess run an update statement against the sysadmin database.
    - update sysadmin:ph_task set …..... where tk_name=”Cloud Backup”;

Using OneDB Explore

1. Login to OneDB Explore.
2. Select onedb-server-0.
3. From the Left Panel choose **Server Administration -> Task Scheduler**.
4. Search for the **Cloud Backup** Task.
5. Select the **Cloud Backup** task and Edit accordingly.



In the above example, the archive time is changed to 05:00 and to occur on Monday, Wednesday and Friday.

Restore an archive

About this task

To restore from an archive or a backup:

1. Scale back the server to a single server pod;
    a. Set serverReplicaCount helm parameter to 1:

```
count.yaml

onedb:
    serverReplicaCount: 1
```

    b. Run a helm upgrade:

```
helm upgrade  <release.name> -f count.yaml -f <previous values>  onedb/onedb-production
```

2. Remove PVC's related to deleted server pods. onedb-server-1 and higher. This should be done by your **kubernetes administrator**:

```
kubectl get pvcs

onedb-onedb-server-1              Bound    pvc-bce5170   10Gi       RWO            standard       68m
onedb-onedb-server-2              Bound    pvc-ba34270   10Gi       RWO            standard       68m
```

```
kubectl delete pvc <pvc's for onedb-server-1 and higher>
```

3. Set the restoreFromBackup helm parameter and run a helm upgrade to initiate the database restore:
    a. Set restoreFromBackup to **true**:

    ```
    restore.yaml

    onedb:
        restoreFromBackup: true
    ```

    b. Run the helm upgrade:

    ```
    helm upgrade  <release.name> -f count.yaml -f <previous values>  onedb/onedb-production
    ```

4. After restore is complete set restoreFromBackup helm parameter back to false and update serverReplicaCount to appropriate values and run helm upgrade to scale out the HA cluster.
    a. Set restoreFromBackup to **false** and serverReplicaCount to desired value:

    ```
    after.yaml

    onedb:
        restoreFromBackup: false
        serverReplicaCount: 2
    ```

    b. Run helm upgrade:

    ```
    helm upgrade  <release.name> -f after.yaml -f <previous values>  onedb/onedb-production
    ```

Disable OneDB archives

About this task

You can disable the OneDB backups at deployment time by providing the following configuration override values:

```
onedb:
    customConfig:
        LOG_BACKUP_MODE: “NONE”

    customInitSQL: |-
        database sysadmin;
        update sysadmin:ph_task set tk_enable='f' where tk_name=”Cloud Backup”;
```

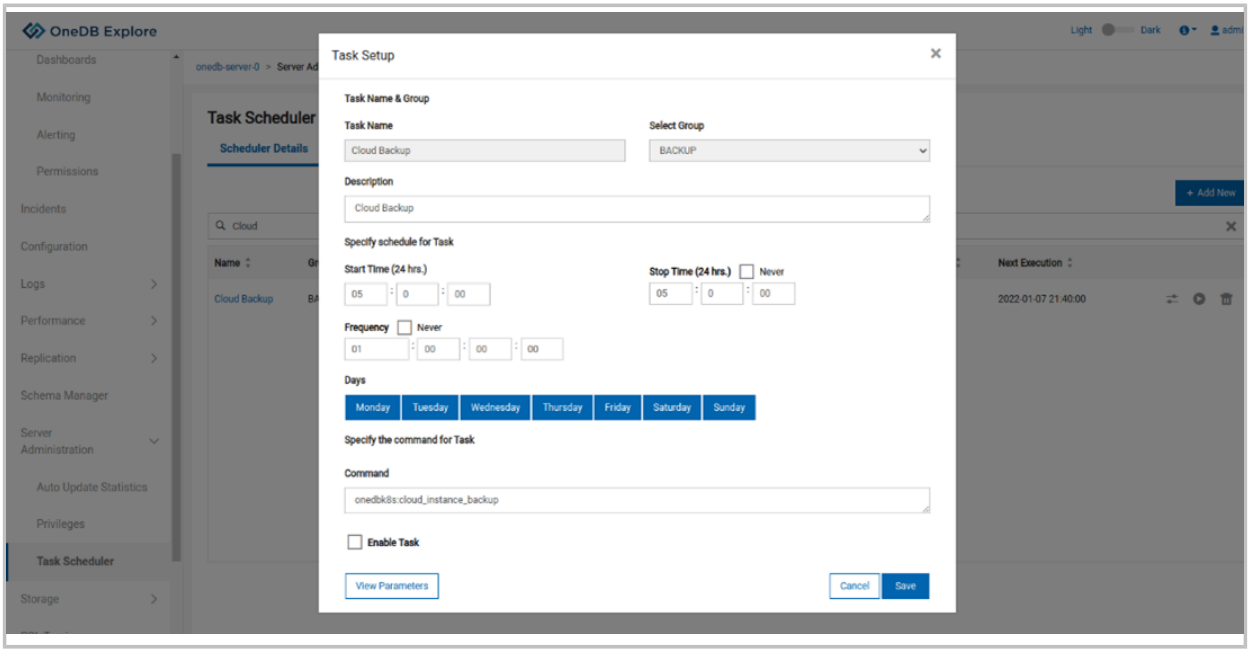
You can also disable the archives after OneDB helm charts have already been installed. You can do this from the command line or you can do this through OneDB Explore.

Using Command Line

1. Exec into onedb-server-0 pod.
2. vi $ONEDB_HOME/etc/$ONCONFIG.
3. Change LOG_BACKUP_MODE to NONE.
4. Use dbaccess run an update statement against the sysadmin database.
    - update sysadmin:ph_task set tk_enable='f' where tk_name=”Cloud Backup”;

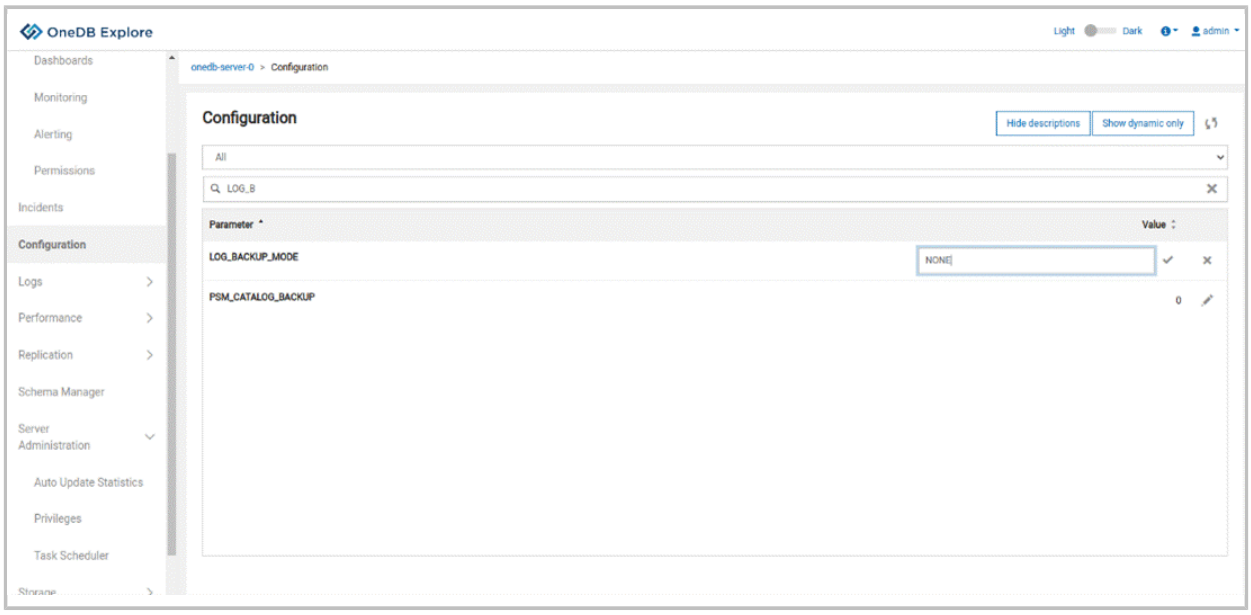
Using OneDB Explore

1. Login to OneDB Explore.
2. Select onedb-server-0.
3. From the Left Panel choose **Server Administration -> Task Scheduler**.
4. Search for the **Cloud Backup** Task.
5. Select the **Cloud Backup** task and Edit accordingly.
6. Uncheck the Enable Task button and Save.



7. Select **Configuration** from the Left Panel.
8. Search for the **LOG_BACKUP_MODE** parameter.

9. Edit this value and change to NONE.



Recover Failing pod

If you are running OneDB as an HA cluster. A primary and secondary and you have a failure of a single pod that doesn't recover, you don't need to perform a restore. Instead you can recover only the failing pod.

To force the pod to be recovered set an annotation to start the recovery:

```
kubectl annotate pod onedb-server-1 onedb_force_ifxclone=true -overwrite=true
```

Once the pod has successfully be cloned disable this annotation:

```
kubectl annotate pod onedb-server-1 onedb_force_ifxclone=false -overwrite=true
```

Note: This shows a recovery of pod onedb-server-0.

Archive with Kubernetes Solution

If you prefer to do your own backups with a kubernetes solution you can do this. First disable the OneDB backups. And when performing the non-OneDB backup it is important to flush all Database activity to disk. This can be performed using External backup and Restore (EBR).

For more information on performing a backup using EBR, see External backup and restore overview.

OneDB Explore

OneDB Explore as a graphical User Interface that can be deployed in the OneDB helm charts. It can be used to monitor and administer one or more OneDB Database servers.

The OneDB Explore helm chart can be deployed separately or as part of OneDB Product. When deploying the OneDB Explore helm chart on its own it is left up to the user to configure and setup. If you use the OneDB Explore with a OneDB Product deployment, then it will be configured and setup automatically for the OneDB HA cluster.

Below are the two OneDB helm charts that OneDB Explore is included with. The default **admin** user password is **testPassw0rd**. For information on changing this default, see Accessing OneDB on page 45.

- **OneDB Explore (onedb-explore):** This is a Helm chart that contains only the OneDB Explore UI.
- **OneDB Product (onedb-product):** This is a Helm chart that contains OneDB SQL Data Store, OneDB Rest, OneDB Document and OneDB Explore all as subcharts.

Configuration

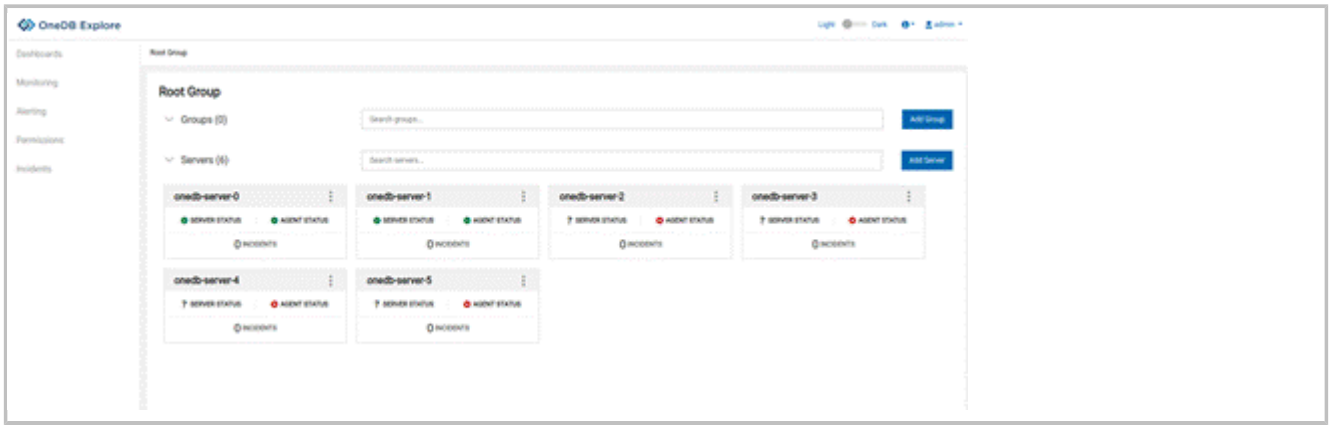
This topic explain the specifics of OneDB Explore in a kubernetes environment. For more information on OneDB Explore and its functionality and capabilities, see OneDB Explore guide.

In the image below, 6 servers have been pre-configured:

A default deployment of OneDB SQL Data Store will create an HA Primary/Secondary Cluster. The deployment will pre-configure 6 servers with OneDB Explore. If you need more than those configured you can add additional servers. If you want to remove any unused servers, you can delete them.

Monitoring

By default, the OneDB Explore agent is enabled on each OneDB Database server. In the following figure, you can see that the first two servers have a green Agent status. This indicates that monitoring is enabled.



To disable the Agent on the OneDB Database server set the configuration override values:

```
onedb-product:
    onedb:
        exploreAgent: false
```

High Availability

When deploying the OneDB SQL Data Store helm chart or one of the charts that includes this as a subchart the default behavior is that you will get an HA cluster with a primary and secondary server. The primary server will be running in onedb-server-0 pod and the updatable HDR secondary will be running in onedb-server-1 pod. HDR replication is configured to use NEAR_SYNC replication mode to avoid data loss.

The OneDB database server cluster is deployed using **onedb-server** statefulset. There will be a second statefulset **onedbcm** deployed using two connection manager pods, onedbcm-0 and onedbcm-1.

The connection manager SLA definitions are configured to use R**OUNDROBIN** policy. This can be changed to **WORKLOAD** by setting the **onedbcm.sla_policy** helm parameter.

Note: OneDB current primary server pod name details are saved using annotations on a **ConfigMap <Helm Release.name>-onedb-clusterinfo**. Pod restart logic looks up annotation details on this configmap to decide whether to start OneDB server as primary or HDR secondary server. Command helm uninstall does not delete this configmap as helm install command could re-use content from the existing configmap, if OneDB server pods are restarted using pre-existing PVCs.

Failover

The OneDB SQL Data store HA cluster supports automatic failover and manual failover. The default is set for automatic failover of the HA cluster. When the HDR primary server becomes non-responsive the HDR secondary needs to take over the primary responsibilities. This can happen automatically, or it can be configured to require manual intervention.

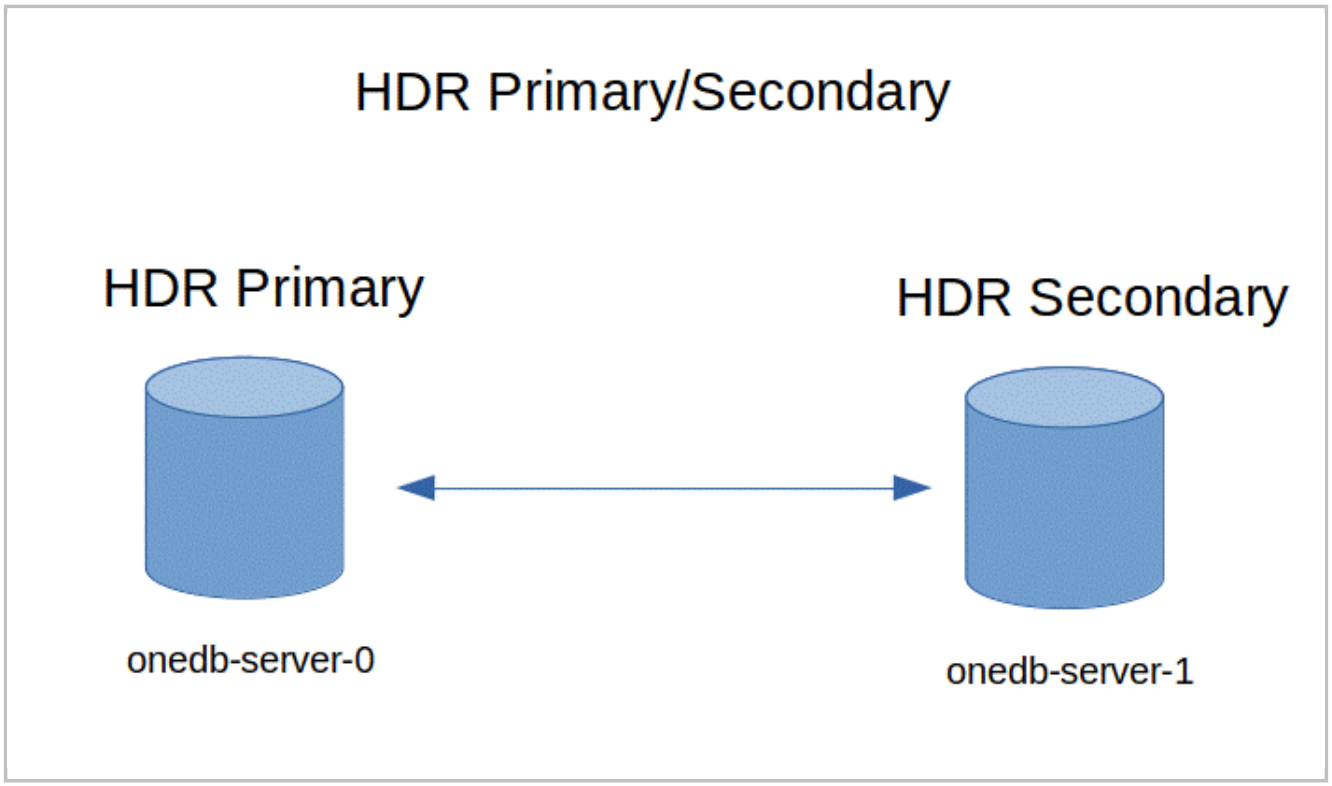
Auto failover functionality is designated by using the **onedbcm.autofailover** helm parameter in the OneDB SQL Data Store helm chart. By default, this value is set to **true**. If manual failover is preferred this can be set to **false**.

When failover is performed whether its automatic or manual the roles will toggle between pods onedb-server-0 and onedb-server-1. When a pod is restarted it will restart the OneDB database server as primary or secondary based on the peer pod and current state of that OneDB server.

Manual Failover

Manual Failover

When the HDR primary server is non-responsive the kubernetes health scripts will fail and the HDR primary server pod will restart. The pod will be restarted as an HDR primary. If the server comes back to a healthy state, then no failover is required.



If the restart of the HDR primary does not come back to a healthy state then manual intervention is needed. You must login to the HDR Secondary, onedb-server-1 pod, and switch it to the HDR primary. The onedb-server-0 pod will not become healthy and will restart as the HDR secondary.

```
onmode -d make primary onedb1
```

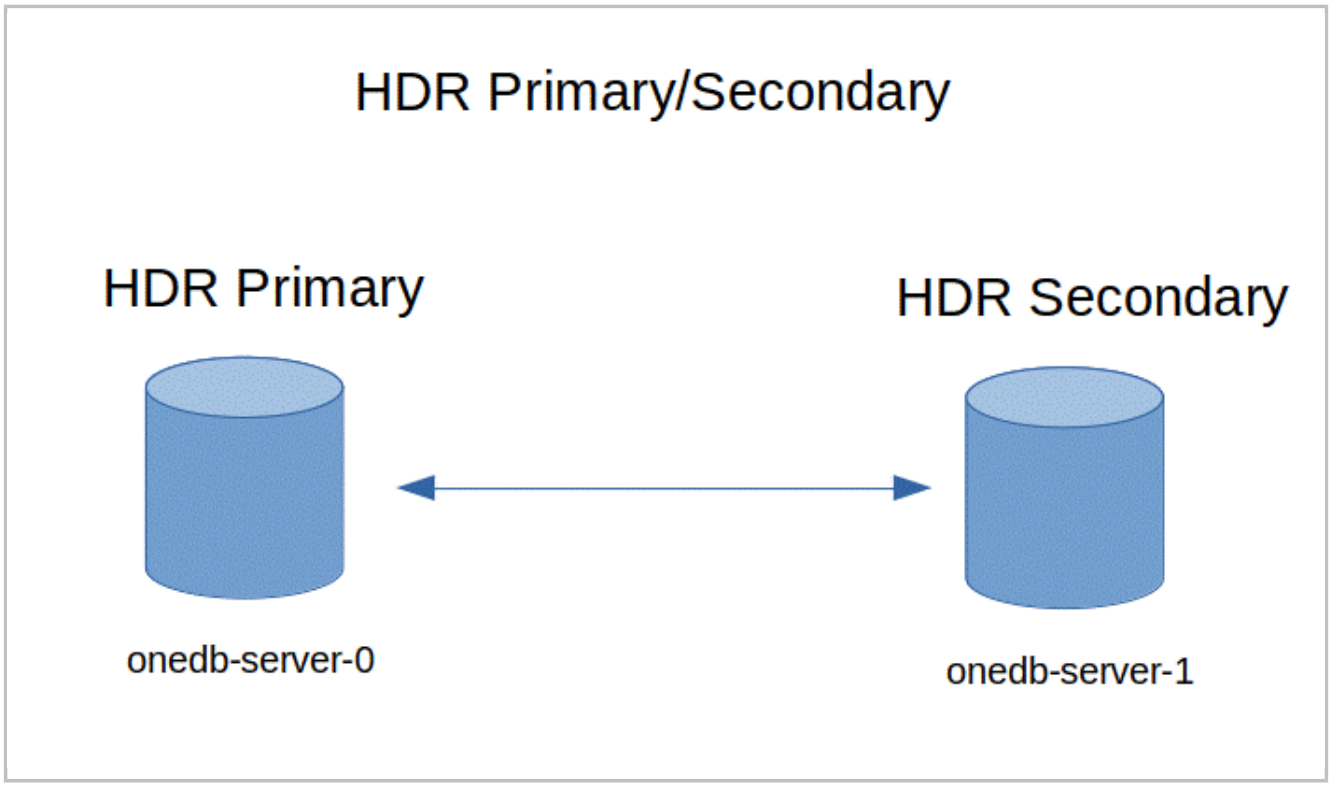
If TLS encryption is being used the name of the server is **onedb1_ssl**. So, the command would be:

```
onmode -d make primary onedb1_ssl
```

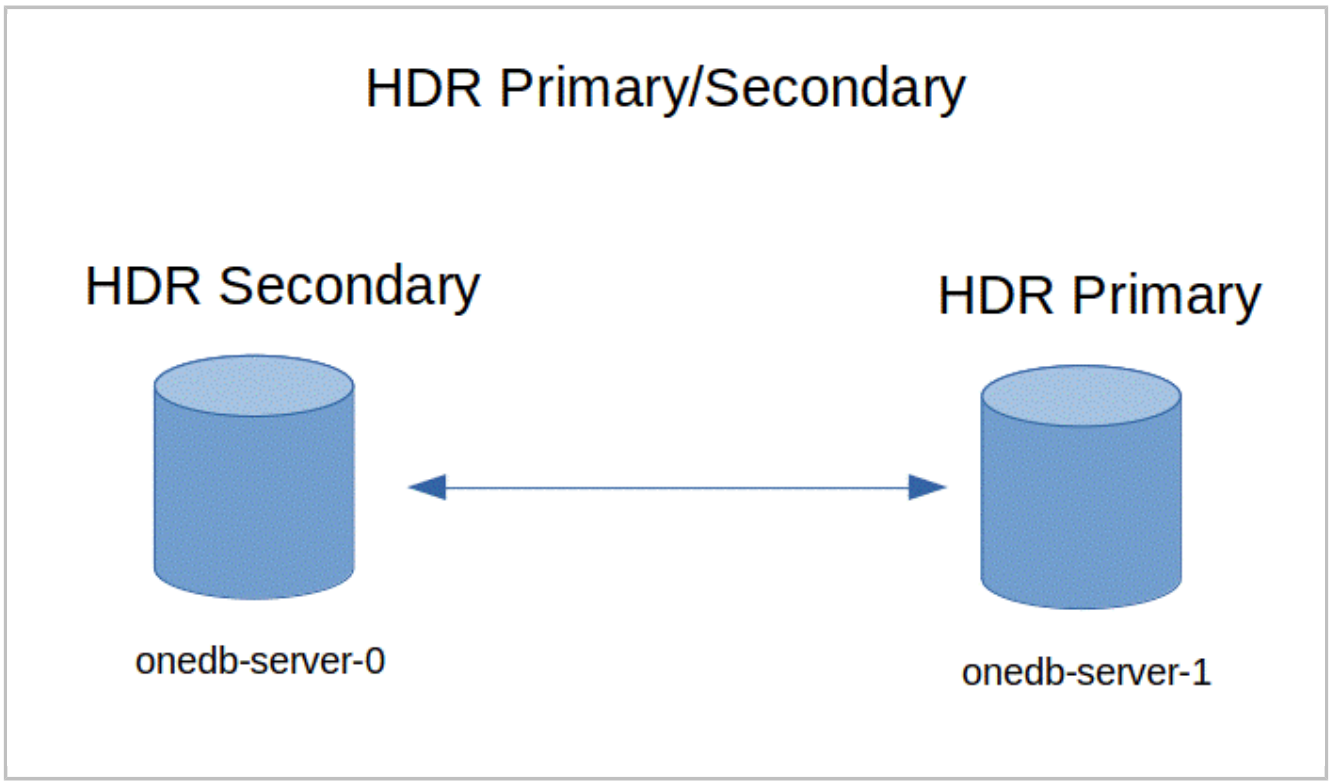
Automatic Failover

Automatic Failover

When the HDR primary server is non-responsive the connection manager will switch the HDR secondary to become the HDR primary. When the old primary server is restarted it will restart as the HDR secondary server.

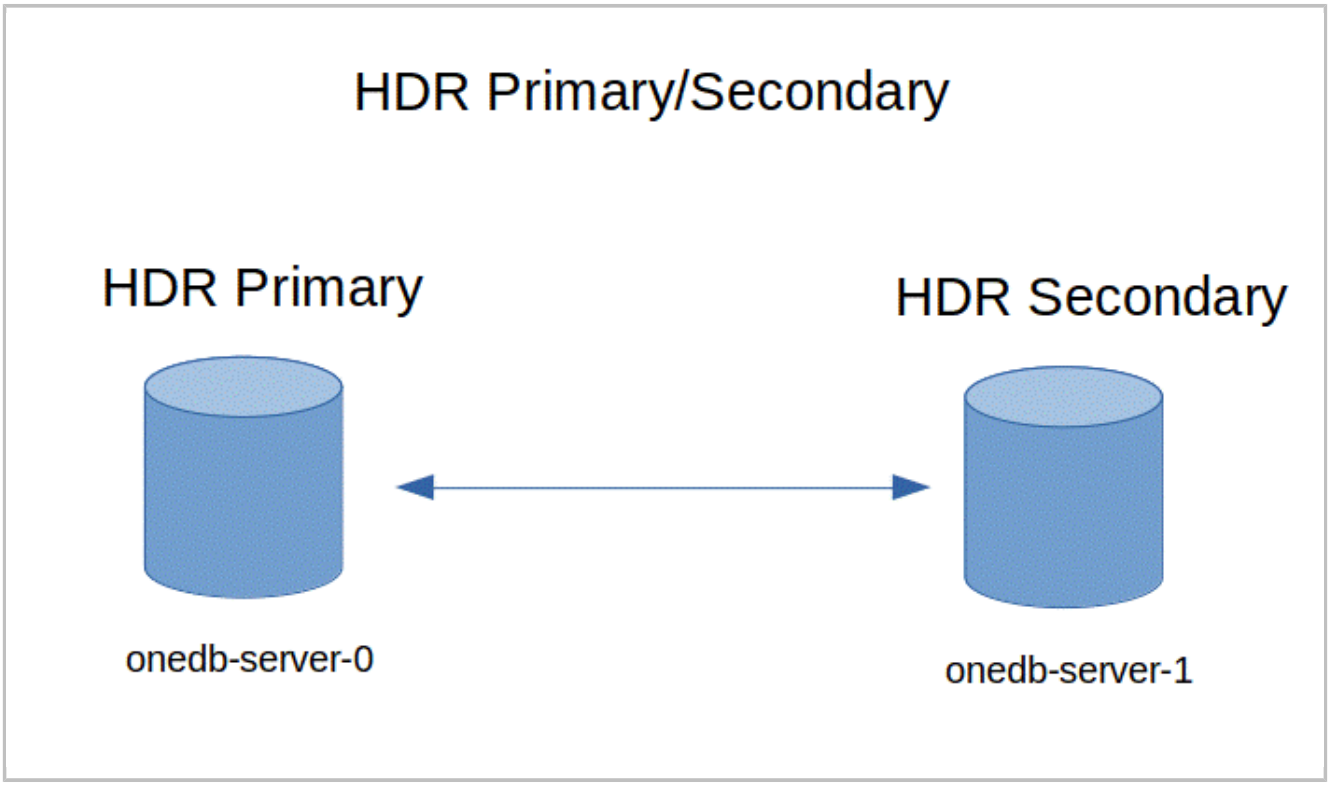


Automatic failure occurs and onedb-server-1 will be made the HDR Primary, and onedb-server-0 restarts as an HDR secondary.



Manual Failover

When the HDR primary server is non-responsive the kubernetes health scripts will fail and the HDR primary server pod will restart. The pod will be restarted as an HDR primary. If the server comes back to a healthy state, then no failover is required.



If the restart of the HDR primary does not come back to a healthy state then manual intervention is needed. You must login to the HDR Secondary, onedb-server-1 pod, and switch it to the HDR primary. The onedb-server-0 pod will not become healthy and will restart as the HDR secondary.

```
onmode –d make primary onedb1
```

If TLS encryption is being used the name of the server is **onedb1_ssl**. So, the command would be:

```
onmode –d make primary onedb1_ssl
```

Scale-Out

The OneDB SQL Data Store by default will start an HDR Primary + Secondary HA cluster. OneDB SQL Data Store allows the scaling of the OneDB Database server and the OneDB Connection manager. The default settings for both the **onedb.serverReplicaCount / onedbcm.cmReplicaCount** helm parameters are **2**.

Another helm chart parameter, **onedb.maxReplicacount,** controls the maximum number of servers that can be used. The default setting for this parameter is 10 and the max supported value is 10. This is an immutable value and once it is set its value cannot be changed.

The OneDB server pods are as follows:

- onedb-server-0: HDR Primary
- onedb-server-1: HDR Secondary
- onedb-server-[2-9]: HDR RSS

When an HDR secondary or RSS is created, **ifxclone** is used to clone the new server from the current HDR primary server. This applies to an initial setup or a scale out scenario.

The maximum number of replicas for the OneDB Connection manager (**onedbcm.cmReplicaCount**) is onedb.maxReplicaCount.

Manual Scale-Out

OneDB supports manual scale out for the OneDB server and the OneDB Connection Manager.

For the OneDB Server to scale out the number of servers manually set the his is accomplished by set the helm parameter **onedb.serverReplicaCount**.

For the OneDB Connection manager to scale out the number of connection managers manually set the his is accomplished by set the helm parameter **onedbcm.cmReplicaCount**.

Manual Scale out of Connection Manager

The onedbcm.cmReplicaCount helm chart parameter can be changed at any time with the helm upgrade command. You can increase or decrease the number of connection managers in the HA cluster by changing this value.

```
helm upgrade onedb-v1 –set onedbcm.cmReplicaCount=3 –f myvalues.yaml onedb-product
```

The pods for the connection manager are onedbcm-0, onedbcm-1, onedbcm-2, and so on.

Automatic Scale-out

The OneDB SQL Data store uses the kubernetes resource **Horizontal Pod Autoscaler (HPA)** to control how scaling will occur automatically. For more information on HPA refer to the kubernetes documentation here: (https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale).

Auto-scale is based on CPU usage and this is disabled by default. To enable auto-scaling set the **autoscale.enabled** helm chart parameter to **true** and set **autoscale.targetCPUUtilizationPercentage** to a percentage value where you want scaling to occur.

The minimum pods for auto scaling would be the helm chart parameter **onedb.serverReplicaCount** for the OneDB Server and **onedbcm.cmReplicaCount** for the Connection manager. The maximum pods for auto scaling would be the **onedb.maxReplicaCount** helm chart parameter.

Important: When enabling auto scaling OneDB Server and Connection Manager resources should be explicitly configured. See **onedb.resources** and o**nedbcm.resources** helm chart parameters.

Following table shows the HPA resource in kubernetes:

Table 2. $ kubectl get hpa Output

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
onedb-c5abcd-hpa	StatefulSet/onedb-server	8%/90%	2	10	2	3h34m
onedbcm-c5abcd-hpa	StatefulSet/onedbcm	5%/90%	2	10	2	3h34m

In the above output the **onedb-c5abcd-hpa** is for the OneDB server and **onedbcm-c5abcd-hpa** is for the Connection Manager. The CPU threshold is set to 90% for each and we see minimum pods is set to 2 with maximum set to 10. Currently there are 2 of each.

Automatic Scale-out

The OneDB SQL Data store uses the kubernetes resource **Horizontal Pod Autoscaler (HPA)** to control how scaling will occur automatically. For more information on HPA refer to the kubernetes documentation here: (https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale).

Auto-scale is based on CPU usage and this is disabled by default. To enable auto-scaling set the **autoscale.enabled** helm chart parameter to **true** and set **autoscale.targetCPUUtilizationPercentage** to a percentage value where you want scaling to occur.

The minimum pods for auto scaling would be the helm chart parameter **onedb.serverReplicaCount** for the OneDB Server and **onedbcm.cmReplicaCount** for the Connection manager. The maximum pods for auto scaling would be the **onedb.maxReplicaCount** helm chart parameter.

Important: When enabling auto scaling OneDB Server and Connection Manager resources should be explicitly configured. See **onedb.resources** and o**nedbcm.resources** helm chart parameters.

Following table shows the HPA resource in kubernetes:

Table 3. $ kubectl get hpa Output

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
onedb-c5abcd-hpa	StatefulSet/onedb-server	8%/90%	2	10	2	3h34m
onedbcm-c5abcd-hpa	StatefulSet/onedbcm	5%/90%	2	10	2	3h34m

In the above output the **onedb-c5abcd-hpa** is for the OneDB server and **onedbcm-c5abcd-hpa** is for the Connection Manager. The CPU threshold is set to 90% for each and we see minimum pods is set to 2 with maximum set to 10. Currently there are 2 of each.

Archive Restore Considerations

When an archive restore is initiated OneDB it will occur on onedb-server-0, pod #0. When a restore occurs OneDb will try to salvage the logical logs, backup up the current logical log.

If the primary is on onedb-server-1 and the onedb-server-0 pod is the secondary, then no salvaging of logical logs will happen. Which means the current logical log will not be archived and available for the restore.

To prevent this from occurring it is recommended to have onedb-server-0 be the Primary server in the HA cluster. If you are in a situation where the HDR primary is on onedb-server-1 you can force a switch over.

Manual Switch Over

If the onedb-server-1 server is the HDR primary and onedb-server-0 server is the HDR secondary you can switch these two roles when automatic failover is disabled by doing the following. Login to the onedb-server-0 server (current HDR Secondary) and perform the failover operation by running the following command:

```
onmode -d make primary onedb0
```

If TLS is enabled for the OneDB HA cluster use the following command:

```
onmode -d make primary onedb0_ssl
```

Configuring On Disk Encryption for database server

Setting helm attribute “onedb.encryptionAtRest” to true enables disk encryption for OneDB database server storage spaces. Encryption key and stash file content is stored in pod specific Kubernetes secret object. K8s Secret object naming convention: **<helm release.name>-ear-onedb0** for onedb-server-0 pod and **<helm release.name>-ear-onedb1** for onedb-server-1 pod.

Important: Make sure to backup content of these Kubernetes secret objects. Without data from these Kubernetes secret objects, OneDB storage space content cannot be decrypted, and data must be restored from database backup.

helm uninstall command do not delete these Kubernetes secret objects, and helm install command re-use content of these secret objects if OneDB server pods were started using pre-existing PVCs. These secret objects must be manually deleted if they are no longer needed.

```
Example:
$ kubectl get secrets
NAME                                                TYPE                                  DATA   AGE
onedb-ear-onedb0                    Opaque                2    3h40m
onedb-ear-onedb1                    Opaque                                  2      3h43m

$ kubectl describe secret onedb-ear-onedb0
Name:       onedb-ear-onedb0
```

```
Namespace:    onedb-nagaraju
Labels:       app=OneDB
              type=ear
Annotations:  <none>

Type:  Opaque

Data
====
p12:  5291 bytes
stl:  32 bytes
```

Automatic Switch Over

If the onedb-server-1 server is the HDR primary and onedb-server-0 server is the HDR secondary you can switch these two roles when automatic failover is enabled by doing the following.

During a planned downtime login to the onedb-server-1 (current HDR primary) and run the following commands:

```
onmode -c
# Wait for checkpoint to complete on primary & secondary
onmode -ky
```

After forcing a checkpoint (onmode –c) verify the checkpoint has completed on the primary and secondary servers by logging in to each and running onstat –m, looking for Checkpoint completed message. After verification of checkpoint, then run the onmode –ky command.

This will cause the HDR primary on onedb-server-1 to go offline. The onedb-server-0 will automatically failover from HDR secondary to HDR primary. And when onedb-server-1 comes back up it will restart as the HDR secondary.

Upgrading OneDB helm charts

When upgrading your helm chart, it is always recommended to take a database backup before upgrading the product. The helm upgrade command is used to upgrade the current release with new configuration. Or, it can be used to upgrade the current version of the chart to a new helm chart version.

OneDB SQL Data Store and the Connection Manager statefulsets support “RollingUpdate” and “OnDelete” update strategies. RollingUpdate is the default update strategy. The RollingUpdate process will start from the highest pod ordinal index to the lowest pod ordinal index. For example, onedb-server-1 is updated before onedb-server-0.

With “OnDelete” update strategy, after ‘helm update’ operation, pod definition will not be updated till user manually deletes the old pod. “OnDelete” update strategy is recommended when database conversion is required or in a situation where user would like to manually control the upgrade procedure.

Note: While upgrading 0.4.16 or earlier helm chart version to chart 0.4.27 or later version, make sure to set ‘nfsserver.enabled’ chart attribute value to ‘true’ before performing ‘helm upgrade’ operation. ‘nfsserver.enabled’

attribute must be set to 'true' to mount ReadWriteMany NFS PVC on all OneDB server pods, and perform necessary steps required to convert OneDB container pod on-disk storage to newer version format.

Important: OneDB Explore Container has upgraded its H2 (embedded database) version due to security vulnerabilities present in the older versions of H2. When upgrading your helm chart to 0.4.27 version for 2.0.1.2, H2 Database needs to be updated **manually**. For more information, see Upgrading OneDB Explore.

The goal of OneDB's upgrade process is to have as little interruption as possible. During an upgrade, kubernetes pods are restarted which will cause a slight interruption in write activity.

If the OneDB Database server does not need to perform a database conversion, then read activity can continue throughout the upgrade process. If a database conversion has to occur then there will be a slight interruption in read activity as well.

To maintain read activity during the upgrade process, your application must be designed with retry logic in it. When a pod is taken down so that it can be upgraded your application should retry its connection so it can connect to and use another server in the cluster.

Upgrading Current release

There may be times when you need to make changes to an existing running release of OneDB in kubernetes. This is performed using helm upgrade and providing the same installed chart with any new values. Parameter values you can change are:

- Set ReplicaCount
- Change container image
- Initiate onbar restore
- Change Connection Manager Service Type: Loadbalancer, ClusterIP, NodePort
- Enable/Disable Automatic failover using Connection Manager
- Change Connection Manager SLA policy: Workload, Round Robin

If the initial installation was performed with this:

```
helm install onedb-v1  -f myvalues.yaml production-onedb
```

The default installation of the helm chart will install an HA cluster with a primary and secondary OneDB server. If you wanted to manually scale the HA cluster to a 3rd server (RSS), you can use helm upgrade and specify a new serverReplicaCount value.

File: newvalues.yaml

```
onedb-product:
    onedb-sql:
        onedb:
            serverReplicaCount: 3
```

Issue the helm upgrade with the original and new values overrides.

```
helm upgrade onedb-v1  -f myvalues.yaml -f newvalues.yaml production-onedb
```

Upgrading 1.x.x.x to 2.x.x.x

A helm upgrade is not supported from OneDB 1.0.0.0 helm chart to a OneDB 2.x helm chart. There are major differences between these two helm chart versions that would prevent a helm upgrade.

When upgrading from a helm chart version using OneDB 1.x to 2.x then you must perform a data migration. It is recommended to use the dbexport.

Note: It is recommended to use the dbexport and dbimport utilities.

Upgrading from 2.0.0.0 version to current version

When upgrading to a new helm chart version you can use the helm upgrade command. This will also most likely be upgrading the version of OneDB database server. For example. Upgrading helm chart version 0.3.52 to 0.4.12 is an upgrade from OneDB 2.0.0.0 to OneDB 2.0.1.0.

When upgrading to a new helm chart version you can use the helm upgrade command. This will also most likely be upgrading the version of OneDB database server. For example. Upgrading helm chart version 0.3.52 to 0.4.12 is an upgrade from OneDB 2.0.0.0 to OneDB 2.0.1.0.

```
helm install onedb-v1  -f myvalues.yaml production-onedb-0.3.52
```

To upgrade to helm chart production-onedb-0.4.12, helm chart version 0.4.12 running OneDB 2.0.1.0, run the following helm upgrade command.

```
helm upgrade onedb-v1  -f myvalues.yaml production-onedb-0.4.12
```

In the above example, the helm chart production-onedb-0.3.52 is used for the OneDB 2.0.0.0 OneDB product. And the helm upgrade command upgrades the helm chart to production-onedb-0.4.12 which the helm chart running OneDB 2.0.1.0.

Rollback from 2.0.1.2 version to previous 2.0.x.x version

About this task

To roll back from OneDB version 2.0.1.2 (helm chart version 0.4.27) to previous 2.0.x.x version, helm rollback command is not supported . Follow these steps to rollback to previous release:

1. Identify current primary server pod name. For this, login to both onedb-server-0 and onedb-server-1 pods to check on current primary server pod name:

   ```
   Example:
   $ kubectl exec -it onedb-server-0  bash
   [informix@onedb-server-0 ~]$ onstat -
   ```

```
HCL OneDB Server Version 2.0.1.2 -- On-Line (Prim) -- Up 1 days 14:09:24 -- 793248 Kbytes
2022-03-25 19:37:34
```

If onstat banner show that it's a primary server(Prim), then note down the pod name.

Alternate method to get primary server pod name:

```
$ kubectl describe configmap <helm release name>-onedb-clusterinfo |grep holderIdentity

 {"holderIdentity":"onedb-server-0","leaseDurationSeconds":360000,"acquireTime":"2022-03-25T19:21:10Z","
renewTime":"2022-03-25T19:21:10Z","...
```

2. Create /opt/hcl/backup/${ONEDB_BACKUPTAG}/primaryhost file and add pod name to the primaryhost file:

```
Example:
$ kubectl exec -it onedb-server-0  bash
$ echo "onedb-server-0" > /opt/hcl/backup/${ONEDB_BACKUPTAG}/primaryhost
```

3. Uninstall 0.4.27 helm chart and wait for all pods to be terminated:

```
Example:
$ helm uninstall <helm release name>
```

4. Install previous release chart using command 'helm install <helm release name> …' and wait for the OneDB pods to be in "Running" state and verify OneDB server and replication state using onstat -g dri command.

Configuring On Disk Encryption for database server

Setting helm attribute "onedb.encryptionAtRest" to true enables disk encryption for OneDB database server storage spaces. Encryption key and stash file content is stored in pod specific Kubernetes secret object. K8s Secret object naming convention: **<helm release.name>-ear-onedb0** for onedb-server-0 pod and **<helm release.name>-ear-onedb1** for onedb-server-1 pod.

Important: Make sure to backup content of these Kubernetes secret objects. Without data from these Kubernetes secret objects, OneDB storage space content cannot be decrypted, and data must be restored from database backup.

helm uninstall command do not delete these Kubernetes secret objects, and helm install command re-use content of these secret objects if OneDB server pods were started using pre-existing PVCs. These secret objects must be manually deleted if they are no longer needed.

```
Example:
$ kubectl get secrets
NAME                                                  TYPE                                  DATA   AGE
onedb-ear-onedb0                    Opaque                2    3h40m
onedb-ear-onedb1                    Opaque                                2       3h43m

$ kubectl describe secret onedb-ear-onedb0
Name:        onedb-ear-onedb0
Namespace:   onedb-nagaraju
Labels:      app=OneDB
             type=ear
Annotations:  <none>
```

```
Type:  Opaque

Data
====
p12:  5291 bytes
stl:  32 bytes
```

Troubleshooting OneDB

The following documentation talks about some troubleshooting techniques that you might use with OneDB in a kubernetes environment.

From the viewing of log files, to enabling a higher level of logging. To disabling the liveness, probe for the OneDB server pod to prevent kubernetes from automatically restarting the OneDB server pods.

Contact OneDB Support with the diagnostic logs and data mentioned in this section as needed.

Troubleshooting Pods

Each pod that is started by kubernetes goes through a series of steps. Some of the common steps you might see are PodInitializing, Container Creating, Pending, Init, Running, ImagePullBackoff.

If a pod seems to be stuck in a state for a period, some of the following techniques can be used:

```
kubectl get pods


NAME                                      READY       STATUS          RESTARTS   AGE
my-nfs-server-provisioner-0                1/1       Running             0      103s
onedb-operator-86d899b5bf-hklq9            0/1       ImagePullBackOff    0      43s
sofy-1-grafana-b7b5f958d-lxcxf             0/2       PodInitializing     0      44s
sofy-1-ksmetrics-6d4677b7d5-zhtmh          1/1       Running             0      44s
sofy-1-odbp-explore-55c9db47c4-nqx8p       0/1       ErrImagePull        0      42s
sofy-1-odbp-mongo-6f6df887df-gn896         0/1       Init:0/1            0      43s
sofy-1-odbp-rest-64f94dfd98-bzj7x          0/1       Init:0/1            0      43s
```

Troubleshooting ImagePullBackoff, Pending pods

When a pod doesn't make it to the Init/Running state **kubectl describe pod** is commonly used to try to gather more information as to what the problem might be. In the above output we see a few pods in ErrImagePull/ImagePullbackOff.

```
kubectl describe pod onedb-operator-86d899b5bf-hklq9

Events:
  Type     Reason     Age                    From               Message
  ----     ------     ----                   ----               -------
  Normal   Scheduled  34m                    default-scheduler  Successfully assigned sofy-
testing/onedb-operator-86d899b5bf-hklq9 to kind-worker4
  Normal   Pulling    32m (x4 over 34m)      kubelet            Pulling image
"hclcr.io/sofy/services/onedb/onedb-operator:2.0.1.0"
  Warning  Failed     32m (x4 over 34m)      kubelet            Failed to pull image
"hclcr.io/sofy/services/onedb/onedb-operator:2.0.1.0": rpc error: code = Unknown desc = failed
to pull and unpack image "hclcr.io/sofy/services/onedb/onedb-operator:2.0.1.0": failed to
resolve reference "hclcr.io/sofy/services/onedb/onedb-operator:2.0.1.0": unexpected status code
[manifests 2.0.1.0]: 401 Unauthorized
  Warning  Failed     32m (x4 over 34m)      kubelet            Error: ErrImagePull
  Warning  Failed     32m (x6 over 34m)      kubelet            Error: ImagePullBackOff
  Normal   BackOff    4m31s (x128 over 34m)  kubelet            Back-off pulling image
"hclcr.io/sofy/services/onedb/onedb-operator:2.0.1.0"
```

You can see here that we failed to pull the image. This gives us some direction in trying to diagnose this issue.

Troubleshooting init pods

OneDB uses init containers to perform setup functions before a specific pod is fully functional. When a pod is in the init state you can run a kubectl logs command to get information about the pod. When running the kubectl logs command on an init-container you need to know the name of the init-container. This can be obtained from the kubectl describe command.

```
kubectl describe pod sofy-1-odbp-mongo-6f6df887df-gn896

Events:
 Type    Reason   Age              From    Message
 ----    ------   ----             ----    -------
 Normal  Pulled   22m (x6 over 63m)   kubelet  Container image "openjdk:8-alpine" already present
on machine
 Normal  Created  22m (x6 over 63m)   kubelet  Created container onedb-mongo-init
 Normal  Started  22m (x6 over 63m)   kubelet  Started container onedb-mongo-init
 Warning  BackOff  4m49s (x25 over 48m)  kubelet  Back-off restarting failed container
```

Once you find the name of the init container, you can run a kubectl logs command. You specify the pod and the name of the init container in the kubectl logs command.

```
kubectl logs sofy-1-odbp-mongo-6f6df887df-gn896 -c onedb-mongo-init
```

Below is a sample output and we can see there is a problem connecting to the OneDB Database server.

```
Running Main
```

```
SQL Service Test Unsuccessful. Server not ready
-908 : com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
```

Troubleshooting running pods

A pod's desired state is to get to a running state with a Ready of 1/1. If you see a Ready Status of 0/1 or see several Restarts for the pod, then you may need to investigate further. The **kubectl log** command can be used to get more information on what the container/pod is doing.

```
kubectl get pods

NAME                                   READY   STATUS    RESTARTS   AGE
helm-1-odbp-explore-686bf69c88-czntt   1/1     Running   0          8m2s
helm-1-odbp-mongo-5bc4d5bb94-54jh4     1/1     Running   0          8m2s
helm-1-odbp-rest-5bd6b956cc-8gqfw      1/1     Running   0          8m2s
my-nfs-server-provisioner-0            1/1     Running   0          8m23s
onedb-operator-6dfdd9bb7d-4phld        1/1     Running   0          8m2s
onedb-server-0                         0/1     Running   0          7m46s
```

Pod onedb-sever-0 hasn't moved into a running state yet and we want to dig deeper into what is going on with this specific pod.

```
kubectl logs onedb-server-0

20:59:00 Peer node onedb1 has version 131077
20:59:00 RSS Server onedb1 - state is now connected
20:59:00 setting version information for onedb1 131077
2022-01-08 20:59:03 LICENSING: <Information> Reacquire licenses: current allocation != expected count
2022-01-08 20:59:03 LICENSING: <Information> Processing current Capability
2022-01-08 20:59:13 LICENSING: <Information> Reacquire licenses: current allocation != expected count
2022-01-08 20:59:13 LICENSING: <Information> Processing current Capability
20:59:13 HDR TIMEOUT - log buffers being sent to onedb1

20:59:13 Error receiving a buffer from RSS onedb1 - shutting down
```

```
20:59:14 RSS Server onedb1 - state is now disconnected
20:59:14 RSS onedb1 deleted
2022-01-08 20:59:23 LICENSING: <Information> Reacquire licenses: current allocation != expected count
2022-01-08 20:59:23 LICENSING: <Information> Processing current Capability
```

Enable/Disable Liveness probe

When doing any type of diagnostic work on a container/pod, it is important that the liveness probe does not take effect and restart the pod. To prevent this from happening you can disable the liveness probe for the OneDB Database server.

To disable use the following kubectl annotation:

```
kubectl annotate pod onedb-server-0 livenessprobe=disabled –overwrite=true
```

Once you are done with your diagnostic work you should re-enable the liveness probe.

To enable, use the following kubectl annotation:

```
kubectl annotate pod onedb-server-0 livenessprobe=enabled –overwrite=true
```

Kubernetes events

Another log of events that can be reviewed/monitored is the Kubernetes events. Run the kubectl get events command and sort or filter this data accordingly.

```
kubectl get events
```

Log in to pod

There may be a need to login to a pod or the init container to obtain more diagnostic information than you get with kubernetes commands. First identify the pod you need to login.

```
kubectl get pods |grep onedb

onedb-operator-67f5d6cd9b-wxcg2 1/1 Running    0 4m
onedb-server-0                  1/1 Running    0 4m
onedb-server-1                  0/1 Init:0/1   0 2m
onedbcm-0                       1/1 Running    0 4m
```

Run the kubectl command to login to the kubernetes pod:

```
kubectl exec --stdin  --tty onedb-server-0 -- bash
```

Once you've logged in to the pod/container, you can move around and view log files as you would on any Linux system.

Log in to init container

To login to an init container, you must first find the name of the init container. This is done using the **kubectl describe pod** command.

```
kubectl describe pod onedb-server-1
```

```
Events:
 Type    Reason     Age   From               Message
 ----    ------     ----  ----               -------
 Normal  Scheduled  76s   default-scheduler  Successfully assigned sofy-testing/onedb-server-0 to
kind-worker
 Normal  Pulling    74s   kubelet            Pulling image "informix-docker-dev-local.us-
artifactory1.nonprod.hclpnp.com/releases/onedb-server:2.0.1.1"
 Normal  Pulled     74s   kubelet            Successfully pulled image "informix-docker-dev-local.us-
artifactory1.nonprod.hclpnp.com/releases/onedb-server:2.0.1.1" in 355.243101ms
 Normal  Created    74s   kubelet            Created container onedb-init
 Normal  Started    74s   kubelet            Started container onedb-init
```

Once you find the name of the init container, you can run the kubectl exec command and login to the init container.

```
Once you find the name of the init container you can run the kubectl exec command and login to the init
 container.
```

Once you've logged in to the pod/container, you can move around and view log files as you would on any Linux system.

Custom init container

Creating a custom init container is a more advanced topic for kubernetes users. An init container is designed to run before starting the main container.

Potential use for custom init container:

- Debug/patch container storage
- Custom container to load data spaces
- Perform any operation on the container/pod prior to starting the container

To use a customer init container, use the following helm parameter override values:

```
onedb:
    customInitImage: gcr.io/google-containers/busybox:latest
    customInitImageCmd: /bin/customization.sh
```

If you needed to login to this container, the name of the init container is **onedb-custom-init**, although we could use the kubectl describe pod command to determine this.

Charts 0.4.16

This version includes the following enhancements:

- Ability to create custom Environment for Mongo container on page 104.
- Ability to create custom Environment for REST container on page 103.
- Ability to create custom Environment for Explore container on page 105.
- Ability to disable/enable specific functionality in OneDB-Product chart on page 106.
- Added a new External Connection Manager Service on page 115.

What's New in this Helm Chart Version

This section includes information about the new, enhanced capabilities added in this version of the helm chart :

- Ability to create custom Environment for Mongo container on page 104.
- Ability to create custom Environment for REST container on page 103.
- Ability to create custom Environment for Explore container on page 105.
- Ability to disable/enable specific functionality in OneDB-Product chart on page 106.
- Added a new External Connection Manager Service on page 115.

Supported Platforms

The OneDB Helm charts have been tested on the following platforms:

- Google Kubernetes Engine (GKE) (https://cloud.google.com/kubernetes-engine)
- AWS Elastic Kubernetes Service (EKS) (https://aws.amazon.com/eks)
- Azure Kubernetes Service (AKS) (https://azure.microsoft.com/en-us/services/kubernetes-service)
- Redhat OpenShift Container Platform (OCP) (https://www.redhat.com/en/technologies/cloud-computing/openshift/container-platform)

Architectural Overview

Installing and deploying OneDB in a cloud-native environment is a new way of looking at things. An evolution of how OneDB is or can be deployed has occurred: starting with on-premises, to in the cloud in Virutal machines, to in the cloud in a highly scalable Kubernetes environment.

In the past, you would have acquired a physical machine, installed the OneDB database server on that machine and been responsible for the maintenance and upgrades on the machine as well as maintenance of the OneDB Database server.

There was then a move to the cloud and the use of Virtual machines in that cloud. Virtual machines made it possible to start up a machine and run a playbook that would install OneDB and configure accordingly. You might do this in your own cloud or a public cloud.

Then more recently, there is the move to a highly scaleable Kubernetes environment. This approach uses containerization of products and pieces of an entire solution. It allows for great flexibility with many benefits. You may use your own Kubernetes solution or a cloud provided Kubernetes from Google, Amazon or Microsoft for example.

General Terminology

To undestand how OneDB Database server is deployed in a kubernetes environment, it is important that you have a basic knowledge of certain terms:

- Container on page 76
- Docker on page 76
- Microservices on page 76
- OneDB HA Cluster on page 76

Container

A container image is a lightweight standalone executable package of software that includes everything to run an application includeing system libraries, tools etc.

Docker

Docker is the leading technology for containizeration. When people think of containers they typcially think of docker. Although it is not the only container technology.

Microservices

A microservices architecture is a method of designing an overall solution to be broken up into smaller parts instead of a single monolithic application. Containers make this a natural path of software development as different pieces can be represented by a different container image.

OneDB HA Cluster

This use of the term cluster refers to the High availability nature of 2 or more OneDB Database servers working together. A 2 nodes OneDB HA cluster will consist of a OneDB HA Primary server and a OneDB HA Secondary server. More servers can be added into a OneDB HA Cluster, in this context the additional servers would be added as OneDB HA RSS nodes.

Kubernetes Terminology

- Node on page 77
- Pod on page 77
- Cluster (kubernetes) on page 77

- Service on page 77
- Helm chart on page 77
- Operator on page 77
- LoadBalancer on page 77

Node

A node is a virtual machine or phyiscal machine with CPU/RAM resources. This is the hardware component that makes up a kubernetes cluster. Example nodes are worker nodes and master nodes.

Pod

A pod is the simplest unit that exists within kuberenetes. Typically this is 1 more more containers. It is pods that get scheduled to run on kubernetes nodes.

Cluster (kubernetes)

Is made up of 1 or more nodes. They provide a resource for a kubernetes solution to be deployed into and managed.

Service

An abstract API object that exposes an application's network services.

Helm chart

A helm chart is a collection of files that describe a related set of kubernetes resources. A helm chart is typically a group of yaml files and other associated files that is used to deploy a solution into kubernetes.

Operator

A kubernetes operator is an application specific controller that extends the functionality of the kubernetes API.

LoadBalancer

A kubernetes object that allows you to expose an external IP address to outside the kubernetes cluster.

OneDB Deployment Resources

When deploying a OneDB helm chart a group of resources will be created. The resources created will depend on the specific OneDB Helm chart that is used.

The OneDB-sql helm chart will deploy the following resources:

- onedb-operator pod
- onedb-server-X pod
- onedbcm-X pod
- onedbcm-cm-service

The OneDB-mongo helm chart will deploy the following resources:

- odbp-mongo pod
- odbp-mongo service
- OneDB-sql chart

The OneDB-rest helm chart will deploy the following resources:

- odbp-rest pod
- odbp-rest service
- OneDB-sql chart

The OneDB-explore helm chart will deploy the following resources:

- odbp-explore pod
- odbp-explore service

The OneDB-product helm chart will deploy the following resources:

- OneDB-sql chart
- OneDB-mongo chart
- OneDB-rest chart
- OneDB-explore chart

Pods

onedb-operator

The purpose of the operator pod is to manager the OneDB HA cluster. By default, a OneDB HA Cluster is started with an HDR primary and secondary server, along with two connection managers.

onedb-server-x

This is the OneDB Database server pod. When deployed, a statefulset is used which will be assigned an ordinal index starting with 0. So, OneDB HA cluster with a primary secondary will have onedb-server-0 and onedb-server-1.

onedbcm-x

This is the OneDB Connection manager pod. It will be assigned an ordinal index starting with 0. By default, 2 connection managers are started. onedbcm-0 and onedbcm-1.

odbp-mongo

This is the OneDB Mongo Listener pod. It is started when the OneDB Mongo chart is deployed. It is used to connect to the OneDB Database server using the Mongo API.

odbp-rest

This is the OneDB REST Listener pod. It is started when the OneDB REST chart is deployed. It is used to connect to the OneDB Database esrver using RESTFUL services.

odbp-explore

This is the OneDB Explore pod. It will deploy the OneDB Explore administration and monitoring tool providing a web admin and monitoring GUI. It can be used to administer one or more OneDB Database servers.

Services

odbp-explore

This is the OneDB Explore service that can be used to access the OneDB Explore product.

odbp-mongo

This is the OneDB Mongo service that is used to access the OneDB Database server using the Mongo API.

odbp-rest

This is the OneDB REST service that is used to access the OneDB Database server using RESTFUL services.

onedbcm-cm-service

This is the OneDB Connection Manager service that is used to access the OneDB Database server using the SQLI + DRDA protocol. EX: JDBC, ODBC.

Prerequisites

To install OneDB into a kubernetes cluster, following prerequisites are needed:

- kubectl
- helm
- ReadWriteMany storage class

Note: To install HCL Sofy Solution into a kubernetes cluster, there may be additional requirements. For more information on the installation instructions for HCL Sofy, see (https://hclsofy.com/ua/guides#installing-solutions-step-by-step-instructions)

.

Kubectl

The kubernetes command line tool, kubectl, is used to run commands and interact with a kubernetes cluster. This is used for managing and interacting with OneDB in kubernetes.

Helm

The helm tool is used to install OneDB in a kubernetes cluster. Helm is a package manager for Kubernetes and is used to install a helm chart.

A helm chart is simply a set of kubernetes yaml manifests that are combined into a single package. This provide an easy method to install a group of kubernetes manifests as a single package.

For installations steps and more information on helm, see: https://helm.sh

RWM Storage

RWM storage is needed to support High availability cluster options with OneDB. Listed are some available options to install RWM storage, but not limited to these. Following options have been tested and verified to work with OneDB.

Enable one and only one of the following options:

Cloud specific options that can be used for these specific cloud providers are:

- Google FireStore on page 80
- AWS Elastic filesystem on page 80
- Azure on page 81

Cloud generic options that can be installed into an existing kubernetes cluster are:

- nfs-server-provisionser on page 81
- rook-ceph on page 82
- rook-nfs on page 83

Google FileStore Configuration

1. See the Google filestore Instructions (https://cloud.google.com/filestore/docs/quickstart-console).
2. The following OneDB helm chart configuration values need to be set to use the Google filestore:

Parameter	***Description***	***Value***
nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.googleFilestore.enable	Set to 'true' to enable Google Filestore	true
nfsserver.googleFilestore.filestoreIP	IP address of the Filestore instance	''
nfsserver.googleFilestore.filestoreShare	Name of the File share of the instance	''

AWS Elastic Filesystem Configuration

1. See AWS filesystem Instructions (https://docs.aws.amazon/eks/latest/userguide/efs-csi.html) .
2. The following OneDB helm chart configuration values need to be set to use the AWS Elastic filesystem.

Parameter	***Description***	***Value***
nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.awsEFS.enable	Set to 'true' to enable AWS Elastic filestore	true
nfsserver.awsEFS.EFSServer	IP address of the Filestore instance	''
nfsserver.googleFilestore.filestoreShare	Name of the File share of the instance	''

Azure File share Configuration

1. See Azure File share instructions (https://docs.microsoft.com/en-us/azure/aks/azure-files-volume).
2. Following OneDB helm chart configuration values need to be set to use the Azure File share:

Parameter	***Description***	***Value***
nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.azureFS.enable	Set to 'true' to enable Azure File share	true
nfsserver.azureFS.secretname	Kubernetes secret to use	''
nfsserver.azureFS.shareName	Azure file share name	''

Install and Configure nfs-server-provisioner

1. Add the nfs-server-provisioner helm repo.

```
helm repo add kvaps https://kvaps.github.io/charts
```

2. Install the helm chart for the nfs-server-provisioner. Specify the following parameters:

Parameter	***Description***	***Value***

persistence.size	Set this to a value of the NFS PV size	50Gi
persistence.enabled	Set to ‘true’ to enable NFS	true
persistence.storageClass	Set this to ‘standard’	standard
storageClass.create	Set to ‘true’	true
storageClass.name	Set thos to a unique Name	onedb-nfs-<namespace>
storageClass.mountOptions		{vers=4.1}

- **storageClass.name**: This is cluster wide so it is recommended to include the namespace in the name to provide uniqueness.
- **storageClass.mountOptions**: Onedb has been tested with NFS V4.1

```
helm install onedb-nfs-server-provisioner kvaps/nfs-server-provisioner   \
--version 1.3.1
--set persistence.enabled=true
--set persistence.storageClass="standard”
--set persistence.size=50Gi
--set storageClass.create=true
--set storageClass.name=-onedb-nfs-my-ns
 --set storageClass.mountOptions={vers=4.1}
```

3. The following OneDB helm chart configuration values need to be set to use the NFS server provisioner.

Parameter	***Description***	***Value***
nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.other.enable	Set to ‘true’ to enable NFS	true
nfsserver.other.storageClass	Set this to the storage class of the NFS	onedb-nfs-<namespace>

- **nfsserver.other.storageClass:** This is set to the the storageClass name specified in the creation of the nfs server provisioner

Install and Configure rook-ceph

1. See the rook-ceph Prerequisites: (https://rook.io/docs/rook/v1.7/pre-reqs.html) .

   Note: Some environments you may need to provision and use *Ubuntu with containerd* node pool instead of the default *GKE container-Optimized OS* (COS).

   .
2. Follow the instructions for rook-ceph: (https://rook.io/docs/rook/v1.7/quickstart.html).
3. Configure a shared file system for rook: (https://rook.io/docs/rook/v1.7/ceph-filesystem.html).
4. Following OneDB helm chart configuration values need to be set to use rook-ceph:

Parameter	***Description***	***Value***
nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.other.enable	Set to 'true' to enable NFS	true
nfsserver.other.storageClass	Set this to the storage class of the NFS	onedb-nfs-<namespace>

   - **nfsserver.other.storageClass:** This is set to the the storageClass name specified in the creation of rook-ceph.

Installation of rook-nfs

Introduction to charts content to go here.

1. Follow the instructions for rook-nfs (https://github.com/rook/rook/blob/master/Documentation/nfs.md).
2. Before Installing OneDB modify the *template/nfs_other_pvc.yaml* file in the helm chart and change the **accessModes:** value from ***ReadWriteMany*** to ***ReadWriteOnce***.
3. After creating the Storage Class, refer to the *sc.yaml* file for rook-nfs. This will contain the storageclass name.
   - Default: rook-nfs-share1.
   - The storage name is needed when installing OneDB.

Parameter	***Description***	***Value***
nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.other.enable	Set to 'true' to enable NFS	true
nfsserver.other.storageClass	Set this to the storage class of the NFS	rook-nfs-share1

   - **nfsserver.other.storageClass:** This is set to the the storageClass name specified in the creation of rook-nfs.

OneDB Requirements and Recommendations

The OneDB database server is designed to be able to run on small devices like a Raspberry pi up to large Servers with 128 cores. The architecture of OneDB is flexible and allows you to run in these different environments with different configurations.

It is important to note that these are recommendations and not requirements. As one user may be able to run their workload on a small device like a Raspberry pi, but another user needs 32 CPUs and 100GB of memory.

When talking about recommendations, we typically refer to CPU, Memory and sometimes disk space.

OneDB Disk/Volume Recommendations

This depends on the amount of data and workload you will have in your OneDB Database server. So every database system will be different. But if High thoughput is needed then we recommend SSD drives to be used. And for your NFS shred drive, spinning disks are ok to use.

Below is a priority of spaces to be setup with SSD if possible. This is not required but the more spaces/volumes setup with SSD drives the better performance can be achieved with the OneDB Database server.

Space/Volume	Drive Type
Logical Log Dbspace	SSD drive(s)
High Volume space	SSD drive(s)
Temp Spaces	SSD drive(s)
Physical Log Dbspace	SSD drive(s)
Low volume space	SSD/Spinning drive(s)
RWM NFS	Spinning drive(s)

The amount of disk space allocated to each of these spaces and volumes is dependent on the size of your data and workload. It is recommended that the RWM NFS volume be approx 3-5 times the size of the total dbspaces if you plan to use the automated backups. We retain 3 archives of the OneDB database server.

Note: It is important to note that you can use all spinning disks and if needed you can put all spaces on a single volume, a separate volume is needed for the RWM NFS. These recommendations are given to provide the best performance possible for a production system.

OneDB Minimum CPU/Memory Recommendations

For a OneDB solution the following can be used as guidelines for the OneDB Server, the Connection Manager, the Mongo wire listener, and the REST wire listener. With OneDB Explore, the minimmum recommendation should be plenty.

Resource	Minimmum Recommendation	General Recommendatoin
CPU	1 core	2 cores
Memory	512 MB	8 GB

As with all systems the more resources, CPU and Memory that a system has the better performance can be achieved. If you find that your workload has a high number of quick connections using the REST or Mongo protocols you may want to increase resources in that area.

The more CPU that is provided to the OneDB system allows you to configure more CPUvps and the more Memory that is provided allows you to configure more memory for Buffers and other database operations.

Note: It is possible to run OneDB with less CPU and Memory. These recommendations are given to provide the best performance possible for a production system.

OneDB Minimum Kubernetes Recommendations

When a OneDB Helm chart is deployed you can specify the minimmum and maximum amount of resources that Kuberentes will use.

When scheduling a pod on a kubernetes node the pod specification can request minimmum resources required. If no node is available with those resources the pod will not be scheduled.

Example: If a pod has a resource.request.cpu of 1, kubernetes will attempt to schedule the pod on a node with >= 1 cpu. If not available, then the pod will not be scheduled.

The following are the current values set in the OneDB Helm Charts.

Pod	Resource	Request	Limit
onedb	CPU	.1 CPU	24 CPU
	Memory	2GB	32GB
CM	CPU	.1 CPU	1 CPU
	Memory	100 MB	500 MB
Mongo/REST	CPU	.1 CPu	2.1 CPU
	Memory	128MB	1GB

Pod	Resource	Request	Limit
Explore	CPU	.1 CPU	2 CPU
	Memory	64MB	512MB

The OneDB Helm charts are also configured by default to not allow two OneDB server pods to be scheduled on the same node. See onedb.nodeSelectorRequired configuration parameter.

The OneDB Helm chart is also configureed to not allow two OneDB Connection Manager pods to be scheduled on the same node. See onedbcm.nodeSelectorRequired configuration parameter.

This does not prevent a Connection manager pod from being scheduled on the same node as a OneDB server pod.

For best performance in a production system it is recommended to configure Affinity along with taints and tolerations to have full control of where the OneDB pods will be scheduled. This will allow you to control the resources available to the individual running pod.

Minimum Recommendation:

- 1 node per OneDB server pod
- 2 nodes for all other pods to be schedued on

General Recommendation: For best performance possible in a production system.

- Use affinity, taints and tolerations
- Configure 1 node per OneDB Server pod
- Configure 1 node per CM
- 1 node or Mongo wire listener
- 1 node for REST wire listener
- Configure 1-2 nodes for other pods

Note: It is possible to run a OneDB Helm chart with fewer nodes. These recommendations are given to provide the best performance possible for a production system.

Overview of Installation

OneDB is deployed into a kubernetes cluster using helm charts. A helm chart is a collection of files that describe a related set of kubernetes resources. A helm chart is typically a group of yaml files and other associated files that is used to deploy a solution into kubernetes.

Before installing a helm chart, you need access to the cluster. The Helm CLI is used to perform the install/uninstall and manage a helm release. Helm is commonly referred to as the package manager for kubernetes. For more information on helm and the installation instructions see: (https://helm.sh).

The kubectl CLI is a kubernetes command line tool to interact with and manage resources in a kubernetes cluster. You can use this tool to verify your installation. For more information on kubectl and installation, see: (https://kubernetes.io).

Previous install

If an helm install has been performed with a prior version, there may be a need to update the Custom Resource Definitions in the kubernetes cluster. OneDB's Custom Resource Definitions can change from release to release. You may see this with a helm upgrade as well when moving from one helm chart version to another, where the CRD has changed.

If you perform a helm install/upgrade and receive an error similar to the error below, update the CRDs associated with OneDB:

```
Error: INSTALLATION FAILED: unable to build kubernetes objects from release manifest: error validating "":
 error validating data: ValidationError(OneDB.spec): unknown field
```

Run kubectl get crds to find the name of the OneDB CRDs:

```
kubectl get crds |grep onedb

NAME CREATED AT
onedbcms.onedb.hcl 2022-03-08T17:02:44Z
onedbs.onedb.hcl 2022-03-08T17:02:44Z
```

Run a kubectl apply command to update the existing CRDs using the new CRDs in the new helm chart:

```

kubectl apply -f onedb-sql/chart/crds/onedb.hcl_onedbcms.yaml
kubectl apply -f onedb-sql/chart/crds/onedb.hcl_onedbs.yaml
```

OneDB Helm Charts

There are five helm charts for OneDB. These helm charts are listed below with a description of each:

- **OneDB SQL Data Store (onedb-sql):** This is the Helm chart that contains the OneDB Database server. It uses a kubernetes operator to deploy a statefulset in a kubernetes environment.
- **OneDB Rest Data Store (onedb-rest):** This is a helm chart that deploys the OneDB Database server with the REST listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Document Data Store (onedb-mongo):** This is a helm chart that deploys the OneDB SQL Data Store with the Mongo listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Explore (onedb-explore):** This is a Helm chart that contains only the OneDB Explore UI.
- **OneDB Product (onedb-product):** This is a Helm chart that contains OneDB SQL Data Store, OneDB Rest, OneDB Document and OneDB Explore all as subcharts.

Differences in Standalone and Solution Factory helm charts

Note: HCL does not currently provide standalone helm charts the only way to get a helm chart for OneDB is through the Solution Factory (Sofy).

The Solution Factory (Sofy), is an Enterprise Kubernetes Solution catalog. Sofy allows you to pick and choose various products to create an overall solution. You can choose one of the OneDB services or Products from the catalog and it will be included in an overall solution with the OneDB helm chart included as a subchart in the Sofy solution. With a Sofy solution helm chart other charts will be included as subcharts like prometheius and grafana and the Sofy UI and of course any product chosen in the catalog.

The main difference between a OneDB helm chart that is installed with a Sofy solution and installed on its own are:

1. Other products/subcharts are included in the Sofy chart.
2. Helm chart overrides at a different level.

helm install

Helm install is used to install a helm chart. This command can point to a path of a directory of an unpacked chart, or a packaged chart. Ex. (chart.tgz).

HCL does not currently provide standalone helm charts the only way to get a helm chart for OneDB is through the Solution Factory (Sofy).

```
helm install [NAME] [CHART] [flags]
```

Installing an unpacked directory chart:

```
 helm install onedb1 onedb-sql
```

Installing a packaged chart (tgz):

```
helm install onedb1 onedb-sql.tgz
```

helm overrides

To override default values in the helm chart you can use --set on the command line. Or you can specify a file with a list of overrides.

Installing with set overrides:

```
 helm install onedb2 --set hclFlexnetURL=flex-net-xxxxx --set hclFlexnetID=xxxxxxx  onedb-sql
```

Installing with a overrides in a file:

```
helm install onedb2-f myvalues.yaml onedb-sql
```

File: myvalues.yaml

```
hclFlexnetURL:  flex-net-xxxxx
hclFlexnetID: xxxxxx
```

Verify Installation

After performing a helm install you can use the kubectl tool to verify the installation. The following resources are some of the items to verify within your kubernetes cluster.

- pods
- services
- deployments
- statefulsets

```
kubectl get pods
```

```
kubectl get services
```

```
kubectl get deployments
```

```
kubectl get statefulsets
```

These commands will show the status of each of these resources. For example, the pods need to be in a running state. If any of these resources are not in a functioning running state you can use kubectl to diagnose. See the kubernetes documentation for more information on kubectl.

Install a Standalone helm chart

HCL does not currently provide standalone helm charts the only way to get a helm chart for OneDB is through the Solution Factory (Sofy).

Install a Solution Factory helm chart

When installing OneDB in a Sofy solution, it will be included as a subchart in the helm chart that is created from the Sofy catalog along with other Solution factory charts like Prometheus, grafana, Sofy console, etc. For more information about Solution Factory see: (https://hclsofy.com/ua/guides).

Before a Sofy helm chart can be installed, there are required steps to be taken. See the step by step instructions for installing a Sofy solution: (https://hclsofy.com/ua/guides#installing-solutions-step-by-step-instructions) .

License Requirements

The OneDB database server requires a license to be used. This is set using *hclFlexnetURL* and *hclFlexnetID* values in the helm chart. Below is a values override file that sets these parameter values. This values to be used for these parameters will be obtained from HCL.

File: myvalues.yaml

```
hclFlexnetURL:  flex-net-xxxxx
hclFlexnetID: xxxxxx
```

A Sofy solution uses a service named **anchor**. This service is used for license management. The OneDB helm charts use anchor but don't need the amount of resources set by default in a Sofy solution helm chart.

You can override the resources used by this **anchor** service by using the following values override file.

File: anchor.yaml

```
anchor:
    resources:
        cpu: 250m
```

Install OneDB SQL Data Store (onedb-sql)

The OneDB SQL Data Store helm chart will install the HCL OneDB database.

HCL OneDB is an enterprise grade database for storing and processing the following types of data:

- Relational (Table based)
- Document (Json Based)
- Timeseries (Time based)
- Spatial (Coordinate based)

Access to OneDB SQL Data Store is through the native SQLI protocol. HCL OneDB provides drivers for different programming languages to provide this connectivity. Ex. Java, Python, NodeJS, ODBC.

After all system requirements and prerequisites have been addressed you are ready to install the Sofy helm chart. The command below is a sample install that uses a file named myvales.yaml to provide any overrides to the helm chart values and anchor.yaml to set the anchor service's resources.

```
helm install sql-v1  -f myvalues.yaml -f anchor.yaml production-onedb-sql
```

Install OneDB RESTful Data Store (onedb-rest)

The OneDB RESTful Data Store helm chart will install the HCL OneDB database along with the OneDB REST listener.

HCL OneDB is an enterprise grade database for storing and processing the following types of data:

- Relational (Table based)
- Document (Json Based)
- Timeseries (Time based)
- Spatial (Coordinate based)

Access to OneDB RESTful Data store is through the REST API. This allows you to use language of choice that supports RESTful services.

After all system requirements and prerequisites have been addressed you are ready to install the Sofy helm chart. The command below is a sample install that uses a file named myvales.yaml to provide any overrides to the helm chart values and `anchor.yaml` to set the anchor service's resources.

```
helm install rest-v1  -f myvalues.yaml -f anchor.yaml production-onedb-rest
```

Install OneDB Document Data Store (onedb-mongo)

The OneDB Document Data Store helm chart will install the HCL OneDB database along with the OneDB Mongo Listener.

HCL OneDB is an enterprise grade database for storing and processing the following types of data:

- Relational (Table based)
- Document (Json Based)
- Timeseries (Time based)
- Spatial (Coordinate based)

Access to OneDB Document Data store is through the MongoDB protocol. This allows you to use any language that supports a MongoDB driver.

After all system requirements and prerequisites have been addressed you are ready to install the Sofy helm chart. The command below is a sample install that uses a file named myvales.yaml to provide any overrides to the helm chart values and anchor.yaml to set the anchor service's resources.

```
helm install mongo-v1  -f myvalues.yaml -f anchor.yaml production-onedb-mongo
```

Install OneDB Explore (onedb-explore)

The OneDB Explore helm chart will install the OneDB Explore web console. The web console is used for visualizing, monitoring, alerting and administering an HCL OneDB server instances.

HCL OneDB Explore features include:

- Purpose built for ease-of-use, scaling out, and optimizing DevOps needs.
- Provides critical performance management capabilities and monitoring of OneDB data store servers.
- The monitoring system feeds directly into a customizable alerting system so alerts can be immediately sent via email, Twilio, or PagerDuty.
- User and permission management for restricted access to dashboard of certain servers or group of servers.

The default login credentials for HCL OneDB Explore are:

- Username: admin
- Password: testPassw0rd

To override the admin password use a values override file and provide that at install time.

File: myvalues.yaml

```
onedb-explore:
    adminPassword: newPassw0rd
```

```
helm install expl-v1  -f myvalues.yaml  production-onedb-explore
```

Install OneDB Product (onedb-product)

The OneDB Product helm chart will install the HCL OneDB database. This helm chart will include as subcharts the other OneDB helm charts:

- OneDB SQL Data Store
- OneDB Mongo Data Store
- OneDB RESTful Data Store
- OneDB Explore

This chart is an all-inclusive chart that includes all the OneDB charts for full functionality.

HCL OneDB is an enterprise grade database for storing and processing the following types of data:

- Relational (Table based)
- Document (Json Based)
- Timeseries (Time based)
- Spatial (Coordinate based)

Access to OneDB Product is through:

- The native SQLI protocol. HCL OneDB provides drivers for different programming languages to provide this connectivity. Ex. Java, Python, NodeJS, ODBC
- The REST API. This allows you to use language of choice that supports RESTful services.
- The MongoDB protocol. This allows you to use any language that supports a MongoDB driver.

OneDB Explore is included as a UI to interact with and administer the OneDB Database server(s).

After all system requirements and prerequisites have been addressed you are ready to install the Sofy helm chart. The command below is a sample install that uses a file named `myvales.yaml` to provide any overrides to the helm chart values and `anchor.yaml` to set the anchor service's resources.

Following file overrides multiple parameters and is provided an installation time:

File: myvalues.yaml

```
hclFlexnetURL:  flex-net-xxxxx
hclFlexnetID: xxxxxx

anchor:
  resources:
    cpu: 250m
```

```
onedb-product:
  onedb-explore:
    adminPassword: newPassw0rd
```

```
helm install prod-v1  -f myvalues.yaml  production-onedb-product
```

Pod Scheduling

As a best practice when deploying OneDB SQL Data store into kubernetes in production isolate the OneDB Database server pods to a specific set of nodes. Also, make sure no two database server pods are scheduled on the save node.

OneDB Server pod scheduling is controlled with the helm chart parameters:

- onedb.nodeSelectorRequired
- onedb.nodeSelector
- onedb.tolerations

To understand how OneDB handles pod scheduling it is import to understand a few concepts.

- Assigning Pods to Nodes (Affinity / Anti-affinity)
- Taints and Tolerations

For more information on Pod scheduling, see kubernetes https://kubernetes.io/.

Assigning Pods to Nodes (Affinity/ Anti-Affinity)

Node Affinity allows you to constrain which nodes your pods are eligible to be scheduled on based on labels that have been defined for the nodes. There are two types of node affinity that are used with OneDB.

- requiredDuringSchedulingIgnoredDuringExecution
- preferredDuringSchedulingIgnoredDuringExecution

requiredDuringSchedulingIgnoredDuringExecution: This is a hard requirement in that the pod “must” be scheduled on the node with the defined set of rules.

preferredDuringSchedulingIgnonredDuringExecution: This is a soft requirement in that the pod will prefer or try to schedule on nodes with the defined rules but it is not guaranteed.

Note: OneDB uses these rules to force a pod to be scheduled on a specific set of nodes or prefer to be scheduled on a specific set of nodes.

Pod anti-affinity allows you to constrain which nodes your pod is eligible to be scheduled on based on pods that are already running on the node. As with node affinity OneDB uses two types of pod anti-affinity.

- requiredDuringSchedulingIgnoredDuringExecution
- preferredDuringSchedulingIgnoredDuringExecution

requiredDuringSchedulingIgnoredDuringExecution: This is a hard requirement in that the pod “must” be scheduled on the node with the defined set of rules.

preferredDuringSchedulingIgnonredDuringExecution: This is a soft requirement in that the pod will prefer or try to schedule on nodes with the defined rules but it is not guaranteed.

Note: OneDB uses these rules to force a OneDB pod to not schedule on nodes already running a OneDB pod or prefer to not be scheduled on that same node.

Labeling Nodes

Your kubernetes administrator will perform this task. They can label a single node or a group of nodes (node pool) with a specific designation with a key/value pair. This is needed to use affinity/anti-affinity capabilities with kubernetes.

To label a node the following command is used:

```
kubectl label nodes <nodename> key=value –overwrite
```

The key/value pair that is defined here is arbitrary. It is a key/value pair that would then be used with helm chart parameter overrides to specify the affinity/anti-affinity.

Example with an arbitrary key/value pair of type=database looks like this:

```
kubectl label nodes gke-worker4 type=database –overwrite
```

Configure OneDB Affinity/Anti-Affinity

We have two helm chart parameters that can be set with OneDB SQL Data store. The OneDB SQL Data store uses these helm chart parameters for both the onedb and onedbcm sections of the helm chart.

```
onedb:
    nodeSelectorRequired: true
    nodeSelector:
        type: database
. . .
onedbcm:
    nodeSelectorRequired: true
    nodeSelector:
        type: cm
```

The default values for onedb/onedbcm **nodeSelectorRequired** is **true.** When this is set to true the requiredDuringSchedulingIgnoredDuringExecution is used for Pod anti-affinity.

The effect of this is that, a OneDB Database server will not be scheduled on the same node where another OneDB Database server pod is running. And a OneDB Connection manager will not be scheduled on the same node where another OneDB Connection manager pod is running.

When we set the **nodeSelector** helm chart parameter for either onedb or onedbcm OneDB will use requiredDuringSchedulingIgnoredDuringExecution and Node affinity is enabled. This will require that all Pods be scheduled on nodes that have been labeled accordingly.

Example Labeling of Nodes:

```
kubectl label nodes gke-worker2 type=database -overwrite
kubectl label nodes gke-worker4 type=database -overwrite

kubectl label nodes gke-worker3 type=cm -overwrite
kubectl label nodes gke-worker5 type=cm -overwrite
```

With the above helm chart values set the OneDB Database server pods must run on a kubernetes nodes that are labled with **type:database**, and OneDB Connection manager pods must run on kubernetes nodes that are labeled with **type:cm**.

OneDB SQL Data store sets up an HA cluster with an HDR primary and HDR secondary. If nodeSelectorRequired is set to true, then we must have more than 1 node labeled when use nodeSelector. The same applies to the OneDB Connection manager based on how many replicas are running.

Note: When configuring pod scheduling it is important to have a good understanding of how this works or you may run into a situation where a pod is not able to be scheduled.

Taints and Toleration

While Node affinity is a property of a pod that attracts them to a set of nodes either as a preference or hard requirement. Taints are the opposite, in that the allow a node to repel a set of pods. A taint is defined on a pod.

```
kubectl taint nodes gke-worker2 type=onedb:NoSchedule
```

This uses a key/value pair in this example we used **type=onedb**, with the NoSchedule effect. This means that no pod will be able to schedule onto the node (gke-worker2) unless it has a matching toleration.

You would then need to use the **tolerations** helm chart parameter override and set the following:

```
tolerations:
-  key: “type”
   operator: “Exists”
   effect: “NoSchedule”
```

Note: Using a combination of Affinity/Anti-Affinity and taints and tolerations, you can control what nodes OneDB SQL Data store will be schedule on and dictate that those nodes are only used for OneDB.

OneDB Configuration

There are five helm charts for OneDB.

- **OneDB SQL Data Store (onedb-sql):** This is the Helm chart that contains the OneDB Database server. It uses a kubernetes operator to deploy a statefulset in a kubernetes environment.
- **OneDB Rest Data Store (onedb-rest):** This is a helm chart that deploys the OneDB Database server with the REST listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Document Data Store (onedb-mongo):** This is a helm chart that deploys the OneDB Database server with the Mongo listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Explore (onedb-explore):** This is a Helm chart that contains only the OneDB Explore UI.
- **OneDB Product (onedb-product):** This is a Helm chart that contains OneDB SQL Data Store, OneDB Rest, OneDB Document and OneDB Explore all as subcharts.

Each chart has a list of configuration options that can be set to specify how the OneDB Helm chart will be installed, setup and configured.

OneDB SQL Data Store Configuration

To customize the installation and configurations, see the list of configuration parameters available to the OneDB SQL Data Store.

List of OneDB SQL Data Store Configuration Parameters

Parameter	*Description*	*Value*
global.hclImagePullSecret	Your own secret with your credentials to HCL's Docker repository. Required when deploying solution in your own cluster.	''
hclFlexnetURL	Your HCL FlexNet license server URL for your HCL entitlements. Required when deploying in your own cluster	''
hclFlexnetID	Your HCL FlexNet license ID fo ryour HCL entitlements. Required when deploying solution in your own cluster.	''
isOpenShift	Set to true if using Openshift	false
nfsserver.volumeSize	Size of the Volume used for backups and other shared files.	50G
nfsserver.googleFilestore.enable	Set to true to enable Google Filestore	false
nfsserver.googleFilestore.filestoreIP	IP address of the Filestore instance	''
nfsserver.googleFilestore.filestoreShare	Name of the File share on the instance	''

Parameter	*Description*	*Value*
nfsserver.awsEFS.enable	Set to true to enable AWS Elastic filestore	false
nfsserver.awsEFS.EFSServer	DNS name of the file system	fs-XXXXX.efs-us-west-2.amazonaws.com
nfsserver.awsEFS.EFSPath	NFS Mount path	/
nfsserver.azureFS.enable	Set to true to enable Azure File share	false
nfsserver.azureFS.secretName	Kubernetes secret name to use	''
nfsserver.azureFS.shareName	Azure file share name	''
nfsserver.other.enable	Set to true to enable	true
nfsserver.other.storageClass	Set this to the storageClass of the NFS	nfs
tls.config	Set to true to enable TLS communication	false
tls.tlskey	Base64 value of server private key	tlskey: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS ...
tls.tlscert	Base64 value of signed server certificate	tlscert: LS0tLS1CRUdJTiBDRVJUSEDFRQ0FURS ...
tls.tlscacert	Base64 value of certificate authority root certificate	tlscacert: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS ...
autoscaling.enabled	Set to true to enable auto scaling. To use auto scaling make sure to set onedb.serverReplicaCount, onedb.maxReplicaCount and onedb.resources appropriately.	false
autoscaling.targetCPUUtilizationPercentage	Set to the Percentage when autoscaling should occur.	70
onedb.serverReplicaCount	Set to the number of servers to start for an HA cluster	2

Parameter	*Description*	*Value*
onedb.maxReplicaCount	Set to the max value of servers you would want to configure in the HA cluster	10
onedb.dbsapwd	OneDB Server DBSA (onedbsa) user password	onedb4ever
onedb.backupTag	Set to unique value in case the backup device is shared with other OneDB HA Cluster	onedbbackup-myuniquetag
onedb.encryptionAtRest	Set to true to enable Encryption at rest	false
onedb.exploreAgent	Set to true to start the OneDB Explore Agent on each server	false
onedb.dataStorageClass	Set to the cloud vendor default storage class. Low latency disk I/O storage is recommended. For GKE “standard” is the default	""
onedb.dataStorageSize	Set the persistent Volume Size	10gi
onedb.dataStorageCount	Number of persistent volume’s to provision	2
onedb.restoreFromBackup	Set to true when a restore from last backup is needed	false
onedb.restoreTimestamp	set to specific point in time to perform a point in time restore. Ex 2021-05-11 11:35:00	‘’
onedb.nodeSelectorRequired	Set to true to enforce that no two OneDB Server pods are scheduled on the same K8s node	true
onedb.nodeSelector	Set to a node label to schedule OneDB server pods on preconfigured set of K8s nodes. Set your own keyvalue pair for the node selector	‘’
onedb.tolerations	Used to assist the K8s schedule	{}
onedb.resources	Resources like cpu and memory requested from the cluster.	{}
onedb.customServerEnv	Allows you to set additional environment variables to be used by the database server.	‘’
onedb.customConfig	Allows you to set ONCONFIG parameters to be used by the Database server.	MULTIPROCESSOR: 1
onedb.customSpace	Allows you to create custom Dbspaces in the database server	‘’
onedb.appUsers	Allows you to create custom Users in the database server	‘’

Parameter	*Description*	*Value*
onedb.customInitSQL	Allows you to create a script of SQL statements that is run after server initialization.	''
onedb.customInitImage	Allows you to create an Init Container image	''
onedb.customInitImageCmd	The command to run from the custom Init container	''
onedb.groupName	Unique name for each cluster if Setting up Enterprise Replication	g_cdr1
onedb.groupID	Unique ID for each cluster if setting up Enterprise Replication	1
onedbcmReplicaCount	Number of Connection Managers to start for the HA cluster	2
onedbcm.serviceType	Set the service type of the connection manager. (ClusterIP, LoadBalancer or NodePort)	ClusterIP
onedbcm.sla_policy	Set the service level agreement of the connection manager. (ROUNDROBIN, WORKLOAD)	ROUNDROBIN
onedbcm.autofailover	If set to false then autofailover is disabled	true
onedbcm.nodeSelectorRequired	Set to true to enforce that no two OneDB CM pods are scheduled on the same K8s node.	true
onedbcm.nodeSelector	Set to a node label to schedule OneDB pods on preconfigured set of K8s nodes. Set your own keyvalue pair.	''
onedbcm.tolerations	Used to assist the K8s schedule	{}
onedbcm.resources	Resources like cpu and memory, requested from the cluster	{}

Customize Server configuration

The ONCONFIG file is used by the database server during initialization to setup the data store server. Use the ***customConfig*** helm chart configuration parameter to specify ONCONFIG parameters. It can be configured as follows. With parameters that are not unique specify a number after the parameter as seen below with BUFFERPOOL# .

```
onedb:
    customconfig:
          MULTIPROCESSOR: “1”
          BUFFERPOOL1: “size=8k,buffers=50000,lrus=8,lru_min_dirty=50,lru_max_dirty=60.5”
```

```
          BUFFERPOOL2: “size=2k,buffers=200000,lrus=8,lru_min_dirty=50,lru_max_dirty=60.5”
          LOGSIZE: “10000”
```

Create Initialization SQL script

The helm chart configuration parameter ***customInitSQL*** can be used to create an SQL scrpit that will run by the OneDB server after first initialization. This script can be used to perform needed setup tasks, creation of databases, etc.

```
onedb:
    customInitSQL:  |-
        database sysadmin;
        create database test with log;
        create table t1 (col1 int, col2 int);
```

Creating custom spaces

The helm chart configuration parameter ***customSpace*** can be used to create and setup spaces. Following table details the options available for the creation of spaces. When defining the customSpace parameter, you must create a well formed json document.

Parameter	***Description***	***Example Value***
name	The name of the space	my_data_dbspace
type	The type of space to create. Supported values are: dbspace: normal dbspace llog : logical log dbspace plog: physical log dbspace sbspace: smart blobspace tempdbspace: temporary dbspace tempsbspace: temporary smart blobspace	dbspace
pagesize	The size of the space, supported values are 2k,4k,6k,8k,16k	4k
size	Size of the space, supported values are GB, MB, KB	10GB
logging	Used for smart blobspaces to enable loging 1: enable logging 0: disable logging	1

```
onedb:
    customSpace: >-
        [
            {“name”:”datadbs”, “type”: “dbspace”, “pagesize”: “4k”, “size”: “4GB” },
```

```
            {“name”:”logdbs”, “type”: “llog”, “size”: “2GB” },
            {“name”:”plogdbs”, “type”: “plog”, “size”:  “4GB” },
            {“name”:”sbspace1”, “type”: “sbspace”, “size”: “1GB” , “logging”: 1},
            {“name”:”tmpdbspace1”, “type”: “tempdbspace”,”pagesize”: “4k”, “size”: “1GB” },
            {“name”:”tmpsbsp1”, “type”: “tempsbspace”, “size”: “500MB” }
        ]
```

Creating custom users

The helm chart configuration parameter ***appUsers*** can be used to create additional users. Following table details the options available for the creation of users. Currently, the only type of user support is an operating system user account. When using appUsers, you must create a well formed json document.

Parameter	***Description***	***Example Value***
user	The name of the user	appuser1
password	The password of the user	passw0rd
group	A group name to create for the user.	dev
uid	The user id number to use for the user	1003
gid	The group id number to to use for the group	2000
type	The type of user to create. Currently only osuser is supported	osuser

```
onedb:
    appUsers: >-
        [
            { “user”:”appuser1”, “password”: “passw0rd”, “group”:”dev”,
 “uid”:1003,”gid”:2000,”type”:”osuser” },
            { “user”:”appuser2”, “password”: “passw0rd”, “group”:”dev”,
 “uid”:1003,”gid”:2000,”type”:”osuser” }
        ]
```

Setting additional server Environment

The helm chart configuration parameter ***customServerEnv*** can be used to set additional server environment variables. This will be set in the environment script when initialization and starting the OneDB database server.

```
onedb:
    customServerEnv:
        DB_LOCALE: “en_us.utf8”
        DBTEMP: “/tmp”
```

Using an Init container

The helm chart configuration parameters ***customInitImage*** and ***customInitImageCmd*** can be used to create an Init container to perform setup steps prior to the startup of the OneDB server container image. The customInitimage parameter is used to specify an image to use and the customInitImageCmd is the command to run inside the image.

The Init container image can be a purposely built image with scripts built in. Or it can be a generic image with specific OS commands to run.

```
onedb:
    customInitImage: “gcr.io/<my-images>/busybox-custom:latest”
    customInitImageCmd: “/bin/initSetup.sh”
```

Scheduling of K8s pods

The helm chart configuration parameter ***nodeSelector*** for onedb and onedbcm are used to support Node affinity. It allows you to select a preconfigured set of K8s nodes to run on.

The following example will run the OneDB server on nodes labeled as onedb and the OneDB Connection manager on nodes labeled as onedbcm.

```
onedb:
    nodeSelector:
        database: onedb
onedbcm:
    nodeSelector:
        cm: onedbcm
```

The helm charts have an unconfigured parameter ***tolerations*** to allow for full configuration of taints and tolerations for K8s scheduling of pods. This can be used to specify a node taint, which means no pod can be scheduled on the node unless it has a matching toleration. Then a OneDB server is labeled with a toleration to allow it to run on the tainted nodes.

```
kubectl taint nodes node1 tainted4onedb=onedb-only:NoSchedule
```

```
onedb:
    tolerations:
    - key: “tainted4onedb”
      operator: “Exists”
      effect: “NoSchedule”
```

Sample helm override file

When specifying helm chart parameters, you can specify them on the command line. When specifying a number of parameters it is sometimes more convenient to create a file with the override parameters. The following example shows a single file that uses customServerEnv, appUsers and customInitSQL in a single file.

```
FILE:  onedb.override.yaml
onedb:
    customServerEnv:
        ONEDB_USER_MANAGEMENT: “true”
        ONEDB_USER: “user1”
```

```
appUsers: >-
    {“user”:”user1”, “password”:”Passw0rd”, “group”:”dba”, “uid”: 1005, “gid”:2001,
     “type”:”osuser”}

customInitSQL:  |-
    create database stores with log;
    create user dbauser with password ‘Passw0rd’ account unlock properties user ‘user1’
    authorization(dbsa);
```

The above yaml file sets two environment variable that are used in the container Image to enable database users. It then creates one os user to be used as the operating system user that the created database user will have permissions as.

OneDB REST Data Store Configuration

To customize the installation and configurations see the list of configuration parameters available to the OneDB REST Data Store.

List of OneDB REST Data Store Configuration Parameters

Parameter	***Description***	***Value***
global.hclImagePullSecret	Your own secret with your credentials to HCL’s Docker repository. Required when deploying solution in your own cluster.	‘’
hclFlexnetURL	Your HCL FlexNet license server URL for your HCL entitlements. Required when deploying in your own cluster	‘’
hclFlexnetID	Your HCL FlexNet license ID for your HCL entitlements. Required when deploying solution in your own cluster.	‘’
resources	Resources like cpu and memory, requested from the cluster.	requests.cpu: “100m”, requests.memory: “128mi”
config	Setting advanced options in the application’s yaml configuration file.	‘’
customEnv	Allows you to set additional environment variables to be used by the OneDB REST container image.	‘’
externalDBUrl	A custom external JDBC style URL if you want to connect to a OneDB server that is not part of the solution	‘’
databaseuser	The user the REST API will use to connect to the OneDB Database Server	onedbsa
databasePassword	The password the REST API will use to connect to the OneDB Database Server	onedb4ever

Custom REST Configuration

Additional configuration can be added to the REST service as follows. Review the product documentation for all available options for the REST configuration file.

```
onedb-rest:
    config: |-
        rest.session.timeout 600000
        security.csrf.token.enable: true
```

```
onedb-rest:
    customEnv:
        TZ: CST2
```

OneDB Document Data Store Configuration

To customize the installation and configurations see the list of configuration parameters available to the OneDB Document Data Store.

List of OneDB Document Data Store Configuration Parameters

Parameter	*Description*	*Value*
global.hclImagePullSecret	Your own secret with your credentials to HCL's Docker repository. Required when deploying solution in your own cluster.	''
hclFlexnetURL	Your HCL FlexNet license server URL for your HCL entitlements. Required when deploying in your own cluster	''
hclFlexnetID	Your HCL FlexNet license ID for your HCL entitlements. Required when deploying solution in your own cluster.	''
resources	Resources like cpu and memory, requested from the cluster	requests.cpu: "100m", requests.memory: "128mi"
config	Setting advanced options in the application's yaml configuration file	''
customEnv	Allows you to set additional environment variables to be used by the OneDB Mongo container image.	''
externalDBUrl	A custom external JDBC style URL if you want to connect to a OneDB server that is not part of the solution	''
mongoUser	The mongo username	mongo
mongoPassword	Password for the mongo user	mongoPassword
databaseuser	The user the MongoDB API will use to connect to the OneDB Database Server	onedbsa

databasePassword	The password the MongoDB API will use to connect to the OneDB Database Server	onedb4ever

Custom Mongo Configuration

Additional configuration can be added to the Document Data Store (Mongo) service as follows. Review the product documentation for all available options for the Mongo configuration file.

```
onedb-mongo:
    config:  |-
        security.sql.passthrough=true
```

```
onedb-mongo:
    customEnv:
        TZ: CST2
```

OneDB Explore Data Configuration

To customize the installation and configurations see the list of configuration parameters available to OneDB Explore Data.

List of OneDB Explore Configuration Parameters

Parameter	*Description*	*Value*
global.hclImagePullSecret	Your own secret with your credentials to HCL's Docker repository. Required when deploying solution in your own cluster.	''
resources	Resources like cpu and memory, requested from the cluster	requests.cpu: "100m", requests.memory: "128mi"
config	Setting advanced options in the application's yaml configuration file	''
customEnv	Allows you to set additional environment variables to be used by the OneDB Explore container image.	''
adminPassword	Initial admin password	testPassw0rd

Custom Explore Configuration

Additional configuration can be added to the Explore service as follows. Review the product documentation for all available options for the Explore configuration file.

```
onedb-explore:
    config:  |-
        key=value
```

```
onedb-explore:
    customEnv:
        TZ: CST2
```

OneDB Product Configuration

The majority of configuration will happen through the helm charts for OneDB SQL Data store, OneDB Document Data store, OneDB Rest Data Store or OneDB Explore. To customize the installation and configurations, following is the list of configuration parameters:

Table 4. List of OneDB Product Configuration Parameters

Parameter	Description	Value
enableChart.onedbMongo	Enable or disable the onedb-mongo chart for installation. Default: true	'true'
enableChart.onedbRest	Enable or disable the onedb-rest chart for installation. Default: true	'true'
enableChart.onedbExplore	Enable or disable the onedb-explore chart for installation. Default: true	'true'
enableChart.onedbSql	Enable or disable the onedb-product chart for installation. Default: true	'true'

Configuring TLS

Use transport layer security (TLS) to create secure connections from OneDB clients to the OneDB database server. By default, TLS is disabled. To enable TLS connections, set the **tls.tlsconfig** helm chart parameter value to **true**.

The following helm chart parameters also need to be set:

- **tlskey**: The base64 encoded value of the private key.
- **tlscert**: The base64 encoded value for the Public signed server certificate.
- **tlscacert**: The base64 encoded value of the certificate authority root certificate.

Example tls configuration:

```
tls:
    tlsconfig: true
    tlskey:  LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t....
    tlscert: LS0tLS1CRUdJTiBABEFUSUZJQ0FURS0tLS0t...
    tlscacert: LS0tLS1CRUdJTiBDRVJUSUEEFZFURS0tLS0t....
```

Create TLS Certificates

About this task

You can obtain your own certificates from a certificate authority or you can create your own with the following steps using openssl:

1. Generate root CA private key PEM file:

```
openssl genrsa -out rootCA.key.pem
```

2. Create a self signed root CA certificate in PEM file:

```
openssl req -new -x509 -key rootCA.key.pem -subj "/C=US/ST=Kansas/L=Olathe/O=HCL/OU=OneDB" -days 3650
 -out
 rootCA.cert.pem
```

3. Generate server private key:

```
openssl genrsa -out server.key.pem
```

4. Generate a certificate signing request (CSR) for OneDB Server:

```
openssl req -new -key server.key.pem -subj
 "/C=US/ST=Kansas/L=Olathe/O=HCL/OU=OneDB/CN=Server/emailAddress=onedb@hcl.com" -out server.req.pem
```

5. Sign certificate with root CA:

```
openssl x509 -req -inform PEM -in server.req.pem -set_serial 1 -CA
rootCA.cert.pem -CAkey rootCA.key.pem -days 3650 -extensions usr_cert -outform PEM -out server.cert.pem
```

6. Convert rootCA.cert.pem to base64 -> tlscacert:

```
base64 rootCA.cert.pem -w 0 > tlscacert
```

7. Convert server.cert.pem to base64 -> tlscert:

```
base64 server.cert.pem -w 0 > tlscert
```

8. Convert server.key.pem to base64 -> tlskey:

```
base64 server.key.pem -w 0 > tlskey
```

Connect from Java client with TLS

About this task

To connect to the OneDB Databaser server with a Java client (JDBC) with TLS you must create a keystore for the client application to use. You need the root CA certificate and will use this file **rootCA.cert.pem** to generate the kesytore.

Create the keystore:

```
keytool -import -file rootCA.cert.pem -keystore ssl.keystore
```

Example

Example OneDB JDBC URL to connect to a OneDB Database server using TLS:

```
jdbc:onedb://XX.XXX.XXX.XX.nip.io:10001/sysmaster;user=onedbsa;password=xxxxxxx;ENCRYPT=true;TRUSTSTORE=./ssl.k
eystore;TRUSTSTOREPASSWORD=xxxxxxx;
CERTIFICATEVERIFICATION=false;loginTimeout=0
```

For more information on connecting JDBC applications with TLS, see HCL OneDB JDBC Driver Guide.

Accessing OneDB

Connecting to the OneDB database server is essential. OneDB allows for connections from Mongo Clients, REST Clients and Native SQLI clients. Ex. JDBC, ODBC, ESQL/C

Connectivity can occur from inside the cluster or from outside the cluster. By default, connections from outside the cluster are not enabled.

Connectivity will be different based on the installation. The two basic installations are:

1. Installation of the Standalone Helm Chart of OneDB
2. Installation of a Solution Factory Helm Chart of OneDB

The OneDB Connection Manager is used for Native SQLI Connections to the Database server. There is a Kubernetes service provided to handle this connectivity.

REST, Mongo and OneDB Explore will each have a Kubernetes service to handle their own connections.

```
   NAME                      TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)               AGE
helm-1-odbp-explore           ClusterIP      10.96.220.30       <none>       8080/TCP              20m
helm-1-odbp-mongo             ClusterIP      10.96.44.208       <none>       27017/TCP             20m
helm-1-odbp-rest              ClusterIP      10.96.90.21        <none>       8080/TCP              20m
onedbcm-cm-service            ClusterIP      10.96.159.103      <none>       10000/TCP,20000/TCP   19m
```

With the OneDB Connection Manager you will see a pattern emerge that will describe what type of connection will occur.

Port Number	Description
10XXX	Internal (Redirected) Connection
20XXX	External (Proxied) Connection
XXXX0	Connection to the HA Primary Server
XXXX1	Connection to the HDR Secondary Server
XXXX2	Connection to any server in the HA Cluster
XXXX3	Connection to the RS Secondary Server
XXX2X	Connection that uses SSL

Example: Port 10021 is an internal Connection (10XXX) to the HA Secondary server (XXXX1) using SSL (XXX2X).

Standalone OneDB Chart

A standalone OneDB helm chart does not include elements of a Solution Factory helm chart. There are mutliple different Standalone helm charts.

- **OneDB SQL Data Store (onedb-sql):** This is the Helm chart that contains the OneDB Database server. It uses a kubernetes operator to deploy a statefulset in a kubernetes environment.
- **OneDB Rest Data Store (onedb-rest):** This is a helm chart that deploys the OneDB Database server with the REST listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Document Data Store (onedb-mongo):** This is a helm chart that deploys the OneDB Database server with the Mongo listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Explore (onedb-explore):** This is a Helm chart that contains only the OneDB Explore UI.
- **OneDB Product (onedb-product):** This is a Helm chart that contains OneDB SQL Data Store, OneDB Rest, OneDB Document and OneDB Explore all as subcharts.

The OneDB helm(s) chart can be configured to only allow connections from within the cluster itself, or they can be configured to allow for connections from outside the cluster.

To allow for connections from outside the cluster the kubernetes service types should be configured as Loadbalancer, or an extra piece of software can be used to provide a single point of ingress into the cluster. Some commonly used ingress/loadbalancer's are Ambassodor, NGINX.

Setting up Ambassador or NGINX is outside the scope of this documentation, instead we will use a Loadbalancer service type.

Connecting from Inside the cluster

OneDB Native SQLI connection are accessible through the OneDB Connection Manager. A kubernetes service will be created with the deployment and that service name is used for connections. A kubernetes service is created for the non-SQLI types as well.

```
kubectl get services

    NAME                       TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)               AGE
helm-1-odbp-explore        ClusterIP         10.96.220.30      <none>       8080/TCP              20m
helm-1-odbp-mongo          ClusterIP         10.96.44.208      <none>       27017/TCP             20m
helm-1-odbp-rest           ClusterIP         10.96.90.21       <none>       8080/TCP              20m
onedbcm-cm-service         ClusterIP         10.96.159.103     <none>       10000/TCP,20000/TCP   19m
```

The OneDB Connection manager supports "Redirected" and "Proxied" connections. For internal connections it is recommended to use "Redirected" connections. The following table shows the Internal connection string to use for each driver type. From within the cluster the **.{namespace}.svc.cluster.local** may not be needed from the URL below.

Driver	URL	Example URL
OneDB driver (SQLI-Primary)	onedbcm-cm-service.{namespace}.svc.cluster.local:10000	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary)	onedbcm-cm-service.{namespace}.svc.cluster.local:10001	jdbc:onedb://{url}/sysmaster

OneDB driver (SQLI-Any)	onedbcm-cm-service.{namespace}.svc.cluster.local:10002	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Primary-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10020	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10021	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10022	jdbc:onedb://{url}/sysmaster
Mongo combatible driver	<Release-Name>-odbp-mongo:27017	mongodb://<release-name>-odbp-mongo:27017
REST	<Release-Name>-odbp-rest:8080	http://<release-name>-odbp-rest:8080
Explore	<Release-Name>-odbp-explore:8080	http://<release-name>-odbp-explore:8080

Connecting from Outside the cluster

OneDB Native SQLI connection are accessible through the OneDB Connection Manager. A kubernetes service will be created with the deployment and that service name is used for connections. A kubernetes service is created for the non-SQLI types as well.

```
kubectl get services

NAME                       TYPE           CLUSTER-IP       EXTERNAL-IP            PORT(S)
 AGE
helm-1-odbp-explore        LoadBalancer   10.96.220.30     172.19.255.202         8080/TCP            20m
helm-1-odbp-mongo          LoadBalancer   10.96.44.208     172.19.255.201         27017/TCP           20m
helm-1-odbp-rest           LoadBalancer   10.96.90.21      172.19.255.200         8080/TCP            20m
onedbcm-cm-service         LoadBalancer   10.96.159.103    172.19.255.203         10000/TCP,20000/TCP     19m
```

The OneDB Connection manager supports “Redirected” and “Proxied” connections. For external connections you must use the “Proxied” connections. The following table shows the external connection string to use for each driver type.

Driver	URL	Example URL
OneDB driver (SQLI-Primary)	{LoadBalancer External IP address}:20000	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary)	{LoadBalancer External IP address}:20001	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any)	{LoadBalancer External IP address}:20002	jdbc:onedb://{url}/sysmaster

OneDB driver (SQLI-Primary-SSL)	{LoadBalancer External IP address}:20020	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary-SSL)	{LoadBalancer External IP address}:20021	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any-SSL)	{LoadBalancer External IP address}:20022	jdbc:onedb://{url}/sysmaster
Mongo combatible driver	{LoadBalancer External IP address}:27017	mongodb://{url}:27017
REST	{LoadBalancer External IP address}:8080	http://{url}:8080
Explore	{LoadBalancer External IP address}:8080	http://{url}:8080

Setting LoadBalancer Type

The default setting for each service type is ClusterIP. This will only allow internal connections. The OneDB helm chart service types of interest are listed below. NOTE: The names of the services may be slightly different when installing onedb-mongo, onedb-rest, onedb-sql chart.

- onedbcm-cm-service
- <Release.Name>-odbp-explore
- <Release.Name>-odbp-mongo
- <Release.Name>-odbp-rest

Each of these service types can be configured as a LoadBalancer to provide external connectivity.

To set Loadbalancer for the Connection Manager:

```
onedb-sql:
    onedbcm:
        serviceType: LoadBalancer
```

To set LoadBalancer for Mongo

```
onedb-mongo:
    service:
        type: LoadBalancer
```

To set LoadBalancer for REST

```
onedb-rest:
   service:
      type: LoadBalancer
```

To set LoadBalancer for Explore

```
```

```
onedb-explore:
      service:
         type: LoadBalancer
```

Solution Factory OneDB Chart

A Solution factory OneDB helm chart contains elements of the Solution factory including things like a Console UI, grafana, prometheus. The OneDB helm chart is included as a subchart of the overall Helm chart. There are multiple different Solution Factory charts that include different aspects of OneDB.

- **OneDB SQL Data Store (onedb-sql):** This is the Helm chart that contains the OneDB Database server. It uses a kubernetes operator to deploy a statefulset in a kubernetes environment.
- **OneDB Rest Data Store (onedb-rest):** This is a helm chart that deploys the OneDB Database server with the REST listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Document Data Store (onedb-mongo):** This is a helm chart that deploys the OneDB Database server with the Mongo listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Explore (onedb-explore):** This is a Helm chart that contains only the OneDB Explore UI.
- **OneDB Product (onedb-product):** This is a Helm chart that contains OneDB SQL Data Store, OneDB Rest, OneDB Document and OneDB Explore all as subcharts.

The Solution Factory OneDB helm(s) chart can be configured to only allow connections from within the cluster itself, or they can be configured to allow for connections from outside the cluster.

To allow for connections from outside the cluster, a Solution factory OneDB helm chart includes and configures Ambassador. The ambassador LoadBalancer will handle connectivity into the kubernetes cluster for Mongo, REST and Explore. The OneDB Connection Manager will handle connections into the kubernetes cluster for Native SQLI clients (ex. JDBC, ODBC, ESQL/C)

By default, OneDB Connection Manager does not allow for external connections.

Connecting from Inside the cluster

OneDB Native SQLI connection are accessible through the OneDB Connection Manager. A kubernetes service will be created with the deployment and that service name is used for connections. The ambassador service is created for the non-SQLI connections. The ambassador service is setup as a Loadbalancer type where as the OneDB Connection Manager has a default setting of ClusterIP.

```
kubectl get services

  NAME                  TYPE              CLUSTER-IP         EXTERNAL-IP            PORT(S)
 AGE
sofy-1-ambassador       LoadBalancer        10.96.224.175     172.19.255.200        80:32653/TCP,44
   68m
sofy-1-odbp-mongo       ClusterIP           10.96.183.135     <none>                27017/TCP
  69m
sofy-1-odbp-rest        ClusterIP           10.96.173.88      <none>                8080/TCP
   69m
```

```
sofy-1-odbp-explore      ClusterIP           10.96.66.46       <none>               8080/TCP
  71m
onedbcm-cm-service       ClusterIP           10.96.33.166      <none>               10000:30248/TCP
  77m
```

The OneDB Connection manager supports “Redirected” and “Proxied” connections. For internal connections it is recommended to use “Redirected” connections. The following table shows the Internal connection string to use for each driver type. From within the cluster the **.{namespace}.svc.cluster.local** may not be needed from the URL below:

Driver	URL	Example URL
OneDB driver (SQLI-Primary)	onedbcm-cm-service.{namespace}.svc.cluster.local:10000	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary)	onedbcm-cm-service.{namespace}.svc.cluster.local:10001	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any)	onedbcm-cm-service.{namespace}.svc.cluster.local:10002	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Primary-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10020	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10021	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10022	jdbc:onedb://{url}/sysmaster
Mongo combatible driver	<Release-Name>-odbp-mongo:27017	mongodb://<release-name>-odbp-mongo:27017
REST	<Release-Name>-odbp-rest:8080	http://<release-name>-odbp-rest:8080
Explore	<Release-Name>-odbp-explore:8080	http://<release-name>-odbp-explore:8080

Connecting from Outside the cluster

OneDB Native SQLI connection are accessible through the OneDB Connection Manager. A kubernetes service will be created with the deployment and that service name is used for connections. The ambassador service is created for the non-SQLI connections. The ambassador service is setup as a Loadbalancer type where as the OneDB Connection Manager has a default setting of ClusterIP. To allow external SQLI connectivity you must set the service type to LoadBalancer.

```
kubectl get services

 NAME                 TYPE           CLUSTER-IP       EXTERNAL-IP          PORT(S)           AGE
sofy-1-ambassador     LoadBalancer   10.96.224.175    172.19.255.200       80:32653/TCP,44   68m
sofy-1-odbp-mongo     ClusterIP      10.96.183.135    <none>               27017/TCP         69m
sofy-1-odbp-rest      ClusterIP      10.96.173.88     <none>               8080/TCP          69m
sofy-1-odbp-explore   ClusterIP      10.96.66.46      <none>               8080/TCP          71m
```

```
onedbcm-cm-service    LoadBalancer   10.96.33.166     172.19.255.201        10000:30248/TCP     77m
```

The OneDB Connection manager supports “Redirected” and “Proxied” connections. For external connections you must use the “Proxied” connections. The following table shows the external connection string to use for each driver type.

Driver	URL	Example URL
OneDB driver (SQLI-Primary)	{LoadBalancer External IP address}:20000	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary)	{LoadBalancer External IP address}:20001	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any)	{LoadBalancer External IP address}:20002	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Primary-SSL)	{LoadBalancer External IP address}:20020	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary-SSL)	{LoadBalancer External IP address}:20021	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any-SSL)	{LoadBalancer External IP address}:20022	jdbc:onedb://{url}/sysmaster
Mongo combatible driver	{LoadBalancer External IP address}:27017	mongodb://{url}:27017
REST	{LoadBalancer External IP address}:8080	http://{url}:8080
Explore	{LoadBalancer External IP address}:8080	http://{url}:8080

Setting LoadBalancer Type

The default setting for the OneDB Connection Manager service type is ClusterIP. This will only allow internal connections. To allow for connectivity from outside the cluster you must set the service type for the OneDB Connection Manager to LoadBalancer.

To set Loadbalancer for the Connection Manager:

```
onedb-product:
    onedbcm:
        serviceType: Loadbalancer
```

Connection credentials

The following table shows the default connection credentials. This can be changed accordingly.

Product/Driver	User	Password
Explore	admin	testPassw0rd
REST	-	-
Mongo	mongo	mongoPassword
SQL Data Store	onedbsa	onedb4ever

The REST connection uses a database connection. You can connect with onedbsa or any other use that was created in the OneDB SQL Data Store.

To change the onedbsa password for the OneDB SQL Data Store use the following configuration override values. If you change this password it is important that you make changes to mongo, rest and explore.

```
onedb-sql:
    onedb:
        dbsapwd: one1dba4ever
```

If you change the passowrd for OneDB SQL Datastore (onedb-sql), you must tell the mongo,rest and explore charts how to connect to the OneDB SQL Datastore (onedb-sql). See the following configuration overrides.

```
onedb-mongo:
    databasePassword:  one1dba4ever

onedb-rest:
    databasePassword:  one1dba4ever

onedb-explore:
    serverConnection:
        password: one1dba4ever
```

To change the user and password for OneDB Mongo use the following configuration override values. If you change this password it is important that you make changes to mongo, rest and explore.

```
onedb-mongo:
    mongoUser: mymongo
    mongoPassword: mongoPassword
```

To change the admin password for OneDB Explore. Use the following configuration override values. If you change this password it is important that you make changes to mongo, rest and explore.

```
onedb-explore:
    adminPassword: newPassw0rd
```

OneDB Connection Manager External Service

Purpose of External CM Service

When you designate the default onedbcm-cm-service as a **LoadBalancer** you expose all ports outside the cluster. This may not be desirable because the non-SSL ports are also exposed outside the cluster.

```
onedb-sql:
    onedbcm:
        serviceType: Loadbalancer
```

Configuration of External CM Service

The following configuration options are available for the external CM service:

Parameter	Description	Value
onedbcm_ext.enabled	Set to true to enable the External CM service	'false'
onedbcm_ext.serviceType	Set the serviceType for this Service. LoadBalancer or NodePort	20000
onedbcm_ext.ports.olttp	Proxy mode Connection to Primary Server	20000
onedbcm_ext.ports.reportp	Proxy mode Connection to Secondary Server	20001
onedbcm_ext.ports.oltpanyp	Proxy mode Connection to ANY Server	20002
onedbcm_ext.ports.reportrssp	Proxy mode Connection to RSS Server	20003
onedbcm_ext.ports.oltpdrdap	Proxy mode Connection to Primary Server with DRDA protocol	20010
onedbcm_ext.ports.reportdrdap	Proxy mode Connection to Secondary Server with DRDA protocol	20011
onedbcm_ext.ports.oltpdrdaanyp	Proxy mode Connection to ANY Server with DRDA protocol	20012
onedbcm_ext.ports.reportrssdrdap	Proxy mode Connection to RSS Server with DRDA protocol	20013
onedbcm_ext.ports.oltpsslp	Proxy mode Connection to Primary Server using SSL	20020

Parameter	Description	Value
onedbcm_ext.ports.reportsslp	Proxy mode Connection to Secondary Server using SSL	20021
onedbcm_ext.ports.oltpanysslp	Proxy mode Connection to ANY Server using SSL	20022
onedbcm_ext.ports.reportrsssslp	Proxy mode Connection to RSS Server using SSL	20023
onedbcm_ext.ports.oltpdrdasslp	Proxy mode Connection to Primary Server with DRDA protocol using SSL	20030
onedbcm_ext.ports.reportdrdasslp	Proxy mode Connection to Secondary Server with DRDA protocol using SSL	20031
onedbcm_ext.ports.oltpdrdaanysslp	Proxy mode Connection to ANY Server with DRDA protocol using SSL	20032
onedbcm_ext.ports.reportrssdrdasslp	Proxy mode Connection to RSS Server with DRDA protocol using SSL	20033

Example Setup of External CM Service (Load Balancer)

To setup the OneDB External CM service you would typically set the original onedbcm service so that it can only be accessed from within the cluster. This is done by setting the serviceType to ClusterIP.

Then configure the External CM serivce accordingly. The configuration below exposes only the SSL ports for the traditional SQLI protocol.

```
onedb-sql:
    onedbcm:
        serviceType: ClusterIP

    onedbcm_ext:
        enabled: true
        serviceType: LoadBalancer
        ports:
            oltpsslp:          20020
            reportsslp:        20021
            oltpanysslp:       20022
            reportrsssslp:     20023
```

Below is a list of services showing the onedbcm-cm-service defined as ClusterIP and the onedbcm-ext-service defined as LoadBalancer. You can then access the external IP address from outside the cluster for the configured ports.

```
kubectl get services

NAME                          TYPE                CLUSTER-IP       EXTERNAL-IP             PORT(S)
       AGE
helm-1-odbp-explore        ClusterIP              10.96.220.30      <none>                  8080/TCP
       20m
```

```
helm-1-odbp-mongo          ClusterIP              10.96.44.208      <none>                27017/TCP
       20m
helm-1-odbp-rest           ClusterIP              10.96.90.21       <none>                 8080/TCP
       20m
onedbcm-cm-service         ClusterIP              10.96.159.103     <none>                10000/TCP,20000/TCP
       19m
onedbcm-ext-service        LoadBalancer           10.96.159.103     172.19.255.203        20020/TCP,20021/TCP
       19m
```

Example Setup of External CM Service (NodePort)

To setup the OneDB External CM service as a NodePort you would typically set the original onedbcm service so that it can only be accessed from within the cluster. This is done by setting the serviceType to ClusterIP.

Then, configure the External CM serivce accordingly. The configuration below exposes only the SSL ports for the traditional SQLI protocol. Depending on the kubernetes platform being used the port numbers from a NodePort service may need to be in a specific range. You can change the port numbers accordingly to fit with the required range.

```
onedb-sql:
    onedbcm:
        serviceType: ClusterIP

    onedbcm_ext:
        enabled: true
        serviceType: NodePort
        ports:
            oltpsslp:           30020
            reportsslp:         30021
            oltpanysslp:        30022
            reportrssslp:       30023
```

Below is the list of services showing the onedbcm-cm-service defined as ClusterIP and the onedbcm-ext-service defined as NodePort. You can access the configured ports through each Nodes IP address.

```
kubectl get services

NAME                         TYPE             CLUSTER-IP      EXTERNAL-IP                PORT(S)
       AGE
helm-1-odbp-explore        ClusterIP          10.96.220.30       <none>                   8080/TCP
       20m
helm-1-odbp-mongo          ClusterIP          10.96.44.208       <none>                  27017/TC
       20m
helm-1-odbp-rest           ClusterIP          10.96.90.21        <none>                   8080/TCP
       20m
onedbcm-cm-service         ClusterIP          10.96.159.103      <none>                  10000/TCP,20000/TCP
       19m
onedbcm-ext-service        NodePort           10.96.159.103      <none>                  30020:30020/TCP
       19m
```

Administering OneDB

The following information describes the common tasks that an administrator may perform for a OneDB Database server in a kubernetes environment.

- Exec into OneDB pod.

  For some administration tasks, you may need to login to the OneDB pod and run commands. To do this, you must have authorization to run kubectl exec.
- Starting and Stopping OneDB.

  Starting and stopping the OneDB pod from kubernetes
- Viewing OneDB Log files.

  To view the OneDB logs in the different pods.
- Backup and Restore.

  To backup and restore the OneDB database server

Exec into OneDB Pod

There may be a need to login to the kuberneres pod to perform administration tasks. First identify the pod you need to login to.

```
kubectl get pods |grep onedb
onedb-operator-67f5d6cd9b-wxcg2 1/1 Running 0 42h
onedb-server-0                  1/1 Running 0 42h
onedb-server-1                  1/1 Running 0 42h
onedbcm-0                       1/1 Running 0 42h
onedbcm-1                       1/1 Running 0 42h
```

Run the kubectl command to login to the kubernetes pod:

```
kubectl exec --stdin --tty onedb-server-0 -- bash
```

You will be logged in as user “informix” and your environment will be set for the OneDB Database server. Which can be verified with the onstat - command:

```
onstat -
HCL OneDB Server Version 2.0.1.0 -- On-Line (Prim)
 -- Up 1 days 18:42:25 -- 793248 Kbytes
2021-12-01 17:37:54
```

Stop/Start OneDB Database Server

Use one of the following methods to stop and restart the OneDB database server:

1. Delete the kubernetes pod
2. Login to Pod and take the OneDB server offline

Delete Pod

About this task

Deleting a pod is a method that can be used to restart the pod. When a pod is deleted or dies, kubernetes will force the pod to be restarted.

1. Delete the pod of interest.

```
kubectl delete pod onedb-server-0
```

2. Monitor the pod that was deleted, as it is restarted. After performing the **kubectl delete pod** the pod will terminate. As it restarts, it will go back into the initialization and eventually a Running state.

```
onedb-server-0                                  0/1     Terminating    0          42h
onedb-server-0                                  0/1     Init:0/1           0          1s
onedb-server-0                                  1/1     Running          0          43s
```

Login Pod

1. First annotate the pod so kubernetes does not restart the pod while you are performing administration tasks:

```
kubectl annotate pod onedb-server-0 livenessprobe=disabled --overwrite=true
```

2. Use the command to take the OneDB Database server offline:

```
onmode -kuy
```

3. Perform any adminstration tasks needed with the server offline.
4. Run the command to bring the OneDB Database server back online.

```
oninit
```

5. Re-enable the liveness probe:

```
kubectl annotate pod onedb-server-0 livenessprobe=enabled --overwrite=true
```

Note: If you do not disable the liveness probe, once you take the OneDB Databaser server offline. The kubernetes liveness probe will begin to fail. After 3 failures of the liveness probe, kubernetes will restart the pod on its own.

Viewing log files

About this task

Use one of the following methods to view the OneDB Database server logs:

1. Use kubectl to view the pod logs

```
kubectl logs onedb-server-0
```

2. View the log files from inside the pod. The example below shows a tail of the online.log. With this method you can view any log file associated with the OneDB Database Server.

```
Exec into the pod
cd $ONEDB_DATA_DIR/logs
tail –f onedb*.logs
```

3. When a pod starts up it will sometimes use an init container to perform setup work prior to the main pod starting.

```
kubectl logs onedb-server-0 –c onedb-init
```

Backup and Restore

The OneDB SQL Datastore supports an HA cluster setup. To do this one of the requirements is that a shared RWM storage is available for use. When setting up HA with OneDB an archive is taken from the OneDB HA Primary Server and restored to a OneDB HA Secondary or OneDB HA RSS system.

The shared volume provides all pods access to the archive's. When deploying a OneDB helm chart one of the initial steps is to take an archive to be used to setup the Secondary and any RSS nodes.

The default behavior is for a level 0 archive to occur every night at 2:30am. The last three backups are retained and any prior archives are cleaned up and removed.

A restore should only be needed if both the OneDB HA primary and OneDB HA secondary servers are corrupted. If just one or ther other is corrupted then a failove scenario can occur to bring the two servers back into sync with one another.

Change Backup Schedule

About this task

Backups are scheduled by the OneDB Scheduler. **Cloud Backup** scheduler task determines what days and what time the level 0 backups occur.

This can be changed by modifying the Scheduler task. You can modify the archive schedule from the command line or you can do this through OneDB Explore.

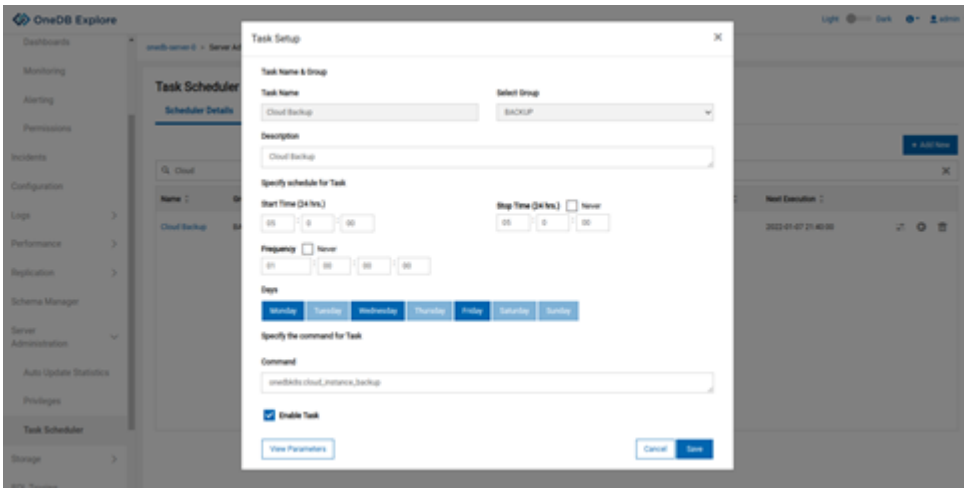
Using Command Line

1. Exec into onedb-server-0 pod.
2. Use dbaccess run an update statement against the sysadmin database.
    - update sysadmin:ph_task set …..... where tk_name=”Cloud Backup”;

Using OneDB Explore

1. Login to OneDB Explore.
2. Select onedb-server-0.

3. From the Left Panel choose **Server Administration -> Task Scheduler**.
4. Search for the **Cloud Backup** Task.
5. Select the **Cloud Backup** task and Edit accordingly.



In the above example, the archive time is changed to 05:00 and to occur on Monday, Wednesday and Friday.

Restore an archive

About this task

To restore from an archive or a backup:

1. Scale back the server to a single server pod;
   a. Set serverReplicaCount helm parameter to 1:

```
count.yaml

onedb:
    serverReplicaCount: 1
```

   b. Run a helm upgrade:

```
helm upgrade  <release.name> -f count.yaml -f <previous values>  onedb/onedb-production
```

2. Remove PVC's related to deleted server pods. onedb-server-1 and higher. This should be done by your **kubernetes administrator**:

```
kubectl get pvcs

onedb-onedb-server-1              Bound    pvc-bce5170   10Gi       RWO            standard       68m
onedb-onedb-server-2              Bound    pvc-ba34270   10Gi       RWO            standard       68m

kubectl delete pvc <pvc's for onedb-server-1 and higher>
```

3. Set the restoreFromBackup helm parameter and run a helm upgrade to initiate the database restore:

a. Set restoreFromBackup to **true**:

```
restore.yaml

onedb:
    restoreFromBackup: true
```

b. Run the helm upgrade:

```
helm upgrade  <release.name> -f count.yaml -f <previous values>  onedb/onedb-production
```

4. After restore is complete set restoreFromBackup helm parameter back to false and update serverReplicaCount to appropriate values and run helm upgrade to scale out the HA cluster.
    a. Set restoreFromBackup to **false** and serverReplicaCount to desired value:

```
after.yaml

onedb:
    restoreFromBackup: false
    serverReplicaCount: 2
```

b. Run helm upgrade:

```
helm upgrade  <release.name> -f after.yaml -f <previous values>  onedb/onedb-production
```

Disable OneDB archives

About this task

You can disable the OneDB backups at deployment time by providing the following configuration override values:

```
onedb:
    customConfig:
        LOG_BACKUP_MODE: “NONE”

    customInitSQL: |-
        database sysadmin;
        update sysadmin:ph_task set tk_enable=’f’ where tk_name=”Cloud Backup”;
```

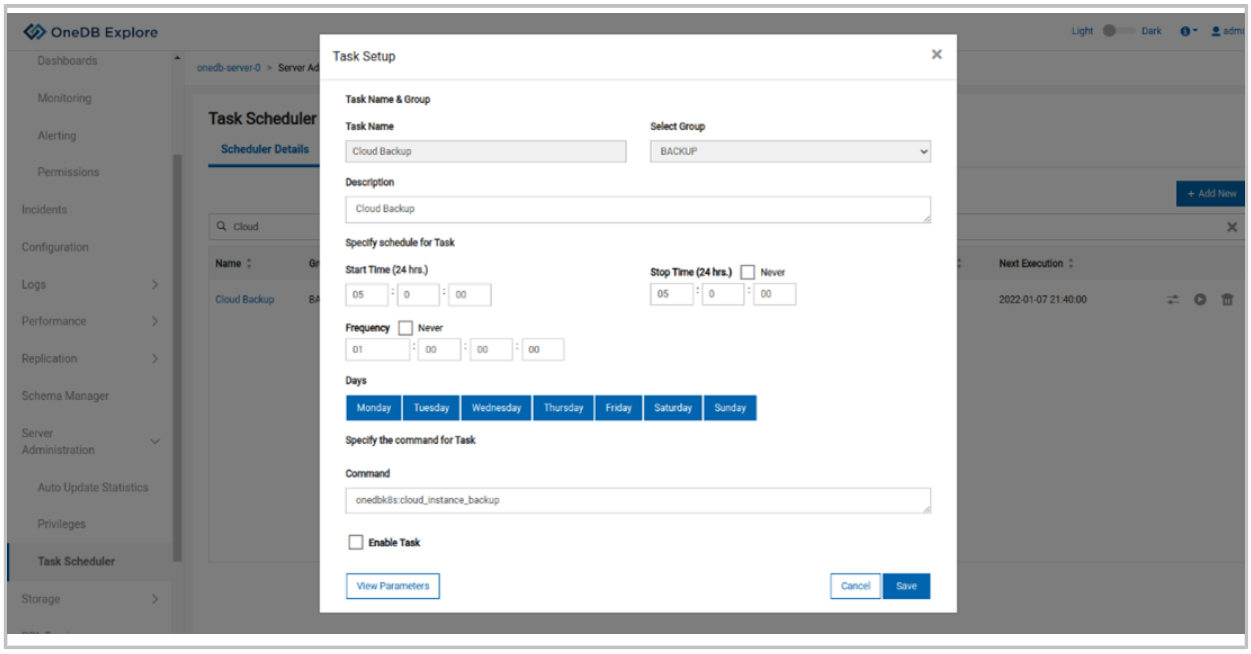
You can also disable the archives after OneDB helm charts have already been installed. You can do this from the command line or you can do this through OneDB Explore.

Using Command Line

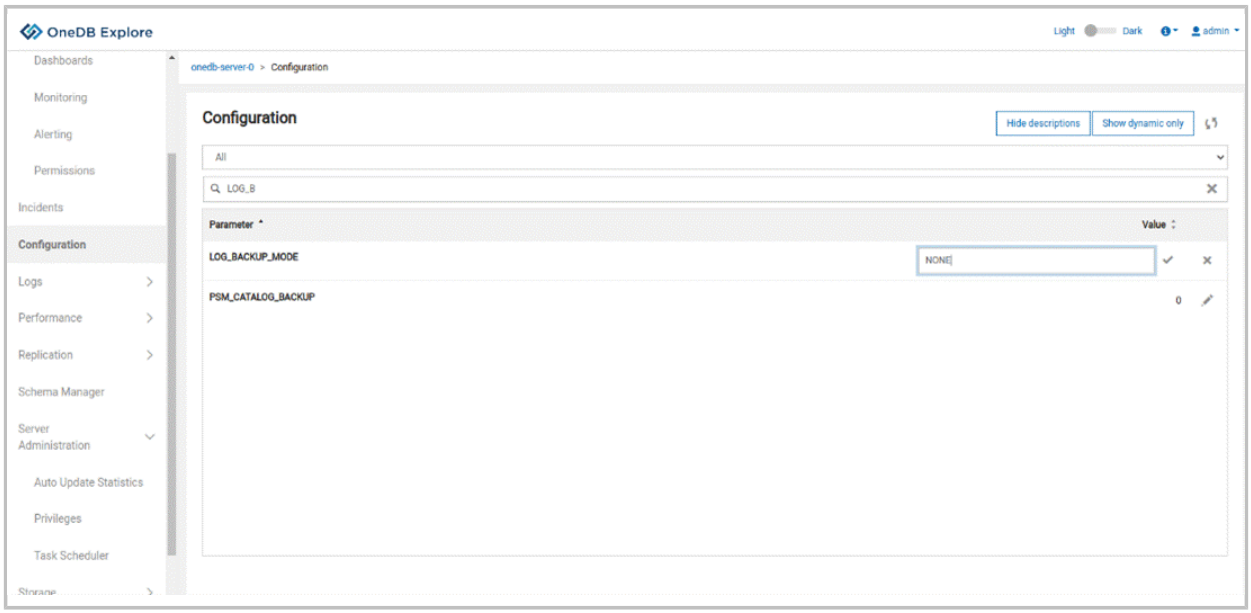
1. Exec into onedb-server-0 pod.
2. vi $ONEDB_HOME/etc/$ONCONFIG.
3. Change LOG_BACKUP_MODE to NONE.
4. Use dbaccess run an update statement against the sysadmin database.
    - update sysadmin:ph_task set tk_enable=’f’ where tk_name=”Cloud Backup”;

Using OneDB Explore

1. Login to OneDB Explore.
2. Select onedb-server-0.
3. From the Left Panel choose **Server Administration -> Task Scheduler**.
4. Search for the **Cloud Backup** Task.
5. Select the **Cloud Backup** task and Edit accordingly.
6. Uncheck the Enable Task button and Save.



7. Select **Configuration** from the Left Panel.
8. Search for the **LOG_BACKUP_MODE** parameter.
9. Edit this value and change to NONE.



Recover Failing pod

If you are running OneDB as an HA cluster. A primary and secondary and you have a failure of a single pod that doesn't recover, you don't need to perform a restore. Instead you can recover only the failing pod.

To force the pod to be recovered set an annotation to start the recovery:

```
kubectl annotate pod onedb-server-1 onedb_force_ifxclone=true -overwrite=true
```

Once the pod has successfully be cloned disable this annotation:

```
kubectl annotate pod onedb-server-1 onedb_force_ifxclone=false -overwrite=true
```

Note: This shows a recovery of pod onedb-server-0.

Archive with Kubernetes Solution

If you prefer to do your own backups with a kubernetes solution you can do this. First disable the OneDB backups. And when performing the non-OneDB backup it is important to flush all Database activity to disk. This can be performed using External backup and Restore (EBR).

For more information on performing a backup using EBR, see External backup and restore overview.

OneDB Explore

OneDB Explore as a graphical User Interface that can be deployed in the OneDB helm charts. It can be used to monitor and administer one or more OneDB Database servers.

The OneDB Explore helm chart can be deployed separately or as part of OneDB Product. When deploying the OneDB Explore helm chart on its own it is left up to the user to configure and setup. If you use the OneDB Explore with a OneDB Product deployment, then it will be configured and setup automatically for the OneDB HA cluster.

Below are the two OneDB helm charts that OneDB Explore is included with. The default **admin** user password is **testPassw0rd**. For information on changing this default, see Accessing OneDB on page 114.

- **OneDB Explore (onedb-explore):** This is a Helm chart that contains only the OneDB Explore UI.
- **OneDB Product (onedb-product):** This is a Helm chart that contains OneDB SQL Data Store, OneDB Rest, OneDB Document and OneDB Explore all as subcharts.

Configuration

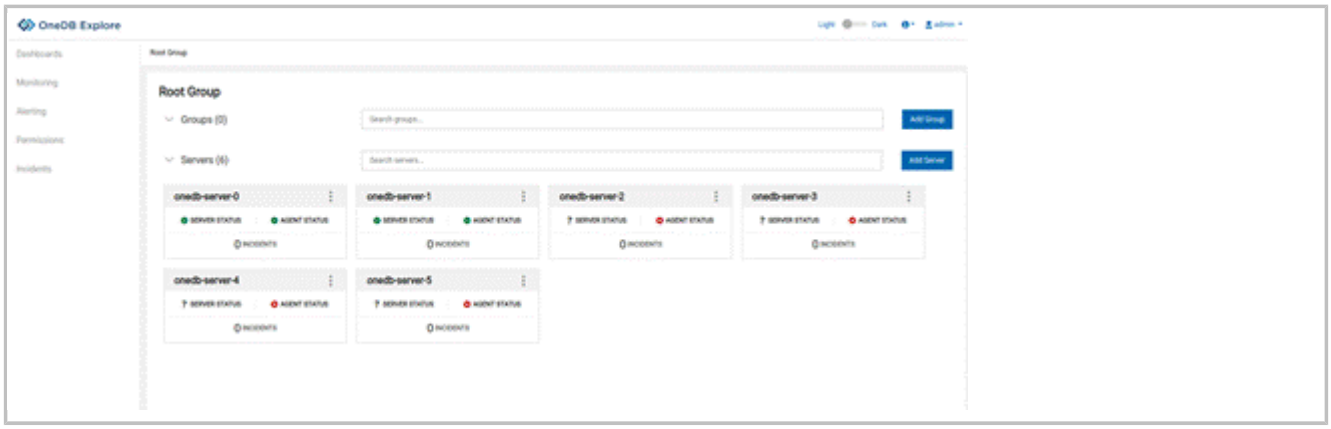
This topic explain the specifics of OneDB Explore in a kubernetes environment. For more information on OneDB Explore and its functionality and capabilities, see OneDB Explore guide.

In the image below, 6 servers have been pre-configured:

A default deployment of OneDB SQL Data Store will create an HA Primary/Secondary Cluster. The deployment will pre-configure 6 servers with OneDB Explore. If you need more than those configured you can add additional servers. If you want to remove any unused servers, you can delete them.

Monitoring

By default, the OneDB Explore agent is enabled on each OneDB Database server. In the following figure, you can see that the first two servers have a green Agent status. This indicates that monitoring is enabled.



To disable the Agent on the OneDB Database server set the configuration override values:

```
onedb-product:
    onedb:
        exploreAgent: false
```

High Availability

When deploying the OneDB SQL Data Store helm chart or one of the charts that includes this as a subchart the default behavior is that you will get an HA cluster with a primary and secondary server. The primary server will be running in onedb-server-0 pod and the updatable HDR secondary will be running in onedb-server-1 pod. HDR replication is configured to use NEAR_SYNC replication mode to avoid data loss.

The OneDB database server cluster is deployed using **onedb-server** statefulset. There will be a second statefulset **onedbcm** deployed using two connection manager pods, onedbcm-0 and onedbcm-1.

The connection manager SLA definitions are configured to use R**OUNDROBIN** policy. This can be changed to **WORKLOAD** by setting the **onedbcm.sla_policy** helm parameter.

Failover

The OneDB SQL Data store HA cluster supports automatic failover and manual failover. The default is set for automatic failover of the HA cluster. When the HDR primary server becomes non-responsive the HDR secondary needs to take over the primary responsibilities. This can happen automatically, or it can be configured to require manual intervention.

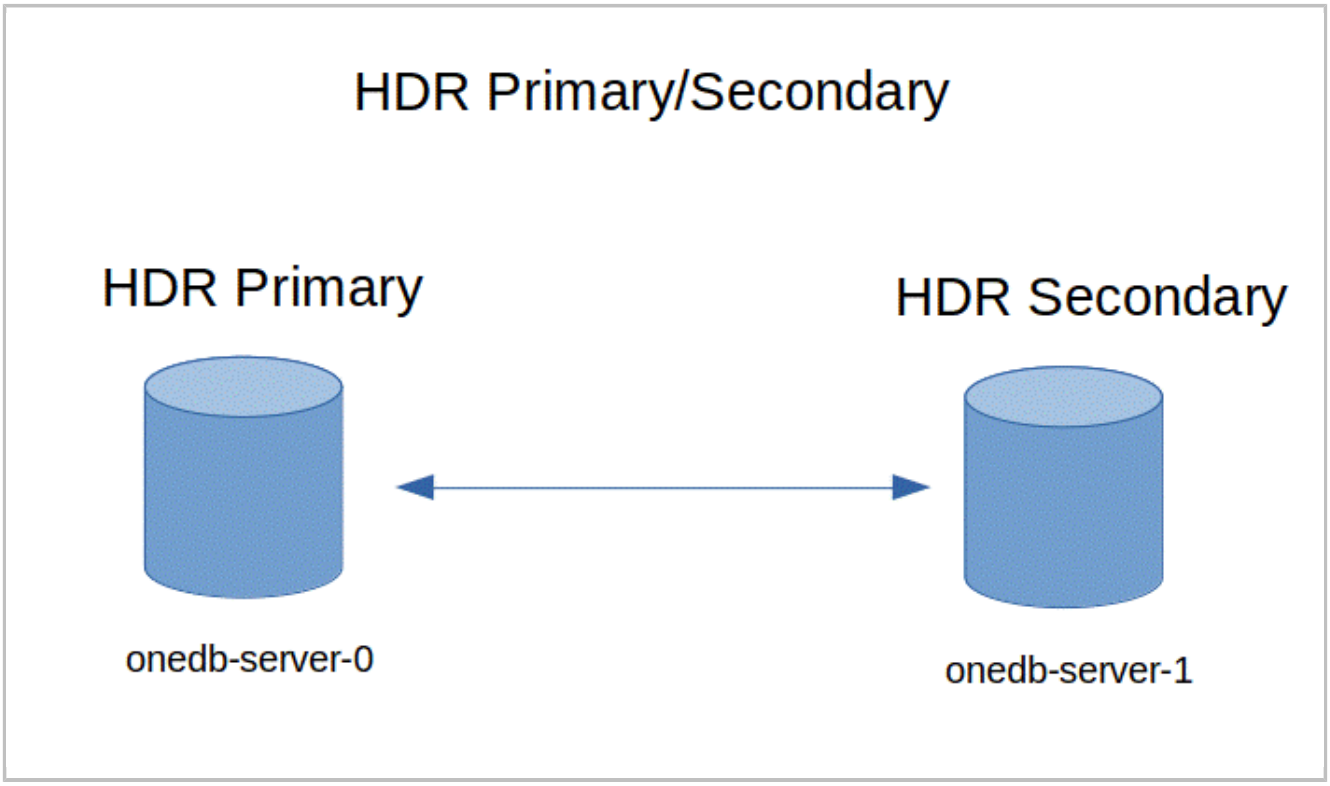
Auto failover functionality is designated by using the **onedbcm.autofailover** helm parameter in the OneDB SQL Data Store helm chart. By default, this value is set to **true**. If manual failover is preferred this can be set to **false**.

When failover is performed whether its automatic or manual the roles will toggle between pods onedb-server-0 and onedb-server-1. When a pod is restarted it will restart the OneDB database server as primary or secondary based on the peer pod and current state of that OneDB server.

Manual Failover

Manual Failover

When the HDR primary server is non-responsive the kubernetes health scripts will fail and the HDR primary server pod will restart. The pod will be restarted as an HDR primary. If the server comes back to a healthy state, then no failover is required.



If the restart of the HDR primary does not come back to a healthy state then manual intervention is needed. You must login to the HDR Secondary, onedb-server-1 pod, and switch it to the HDR primary. The onedb-server-0 pod will not become healthy and will restart as the HDR secondary.

```
onmode -d make primary onedb1
```

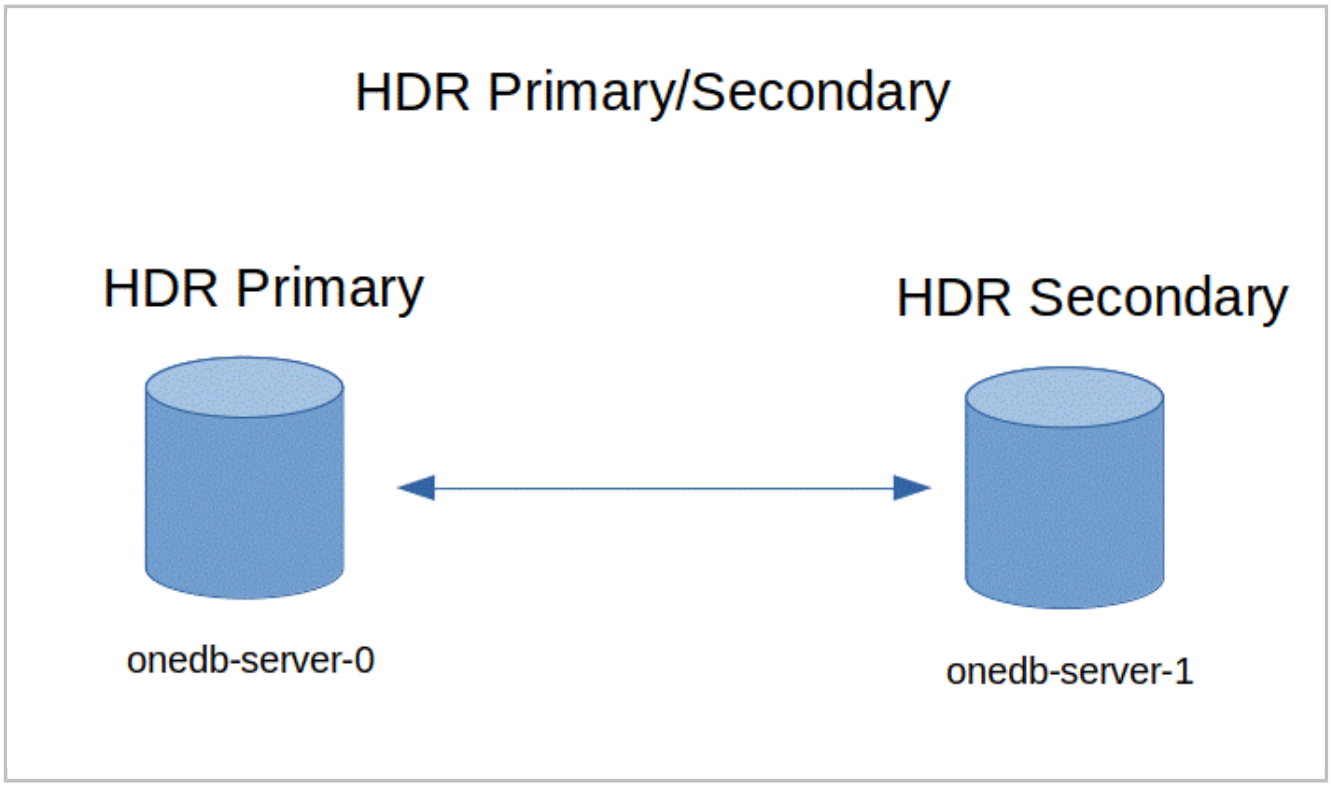
If TLS encryption is being used the name of the server is **onedb1_ssl**. So, the command would be:

```
onmode -d make primary onedb1_ssl
```

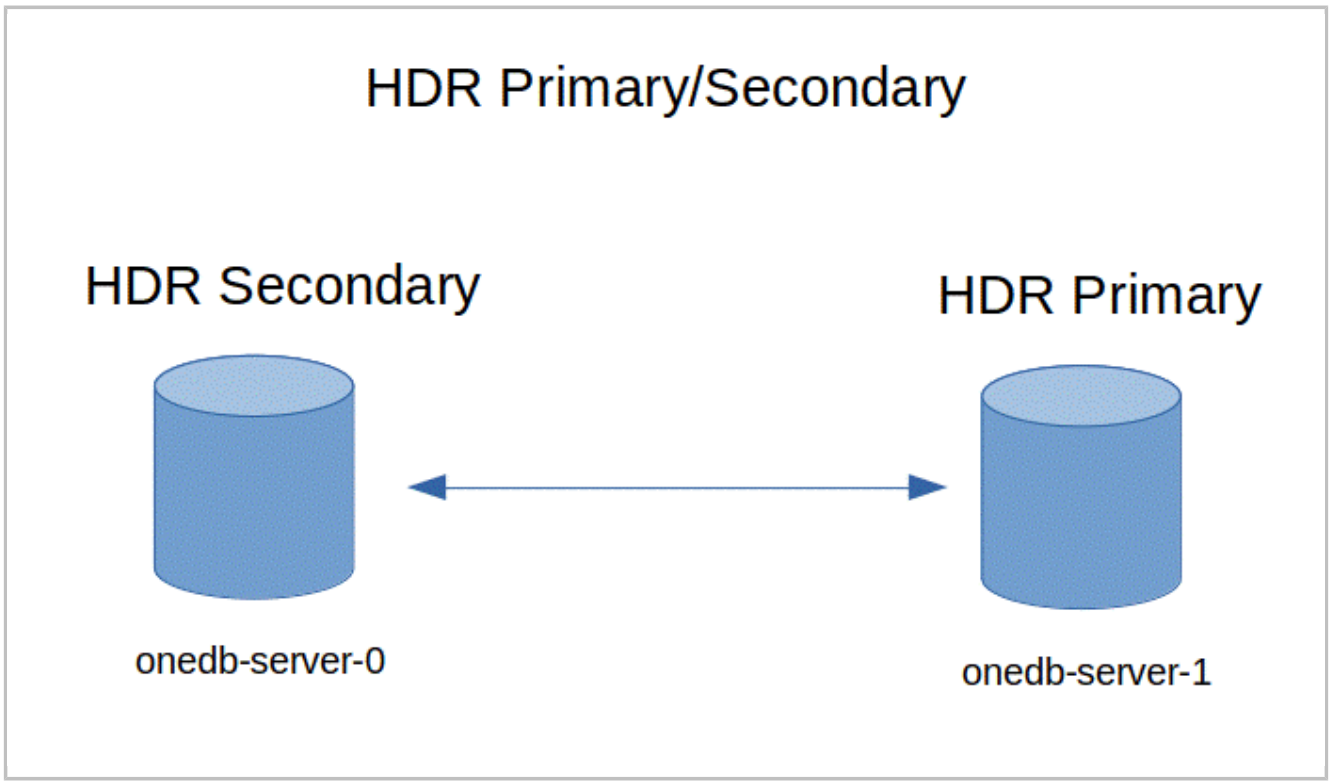
Automatic Failover

Automatic Failover

When the HDR primary server is non-responsive the connection manager will switch the HDR secondary to become the HDR primary. When the old primary server is restarted it will restart as the HDR secondary server.

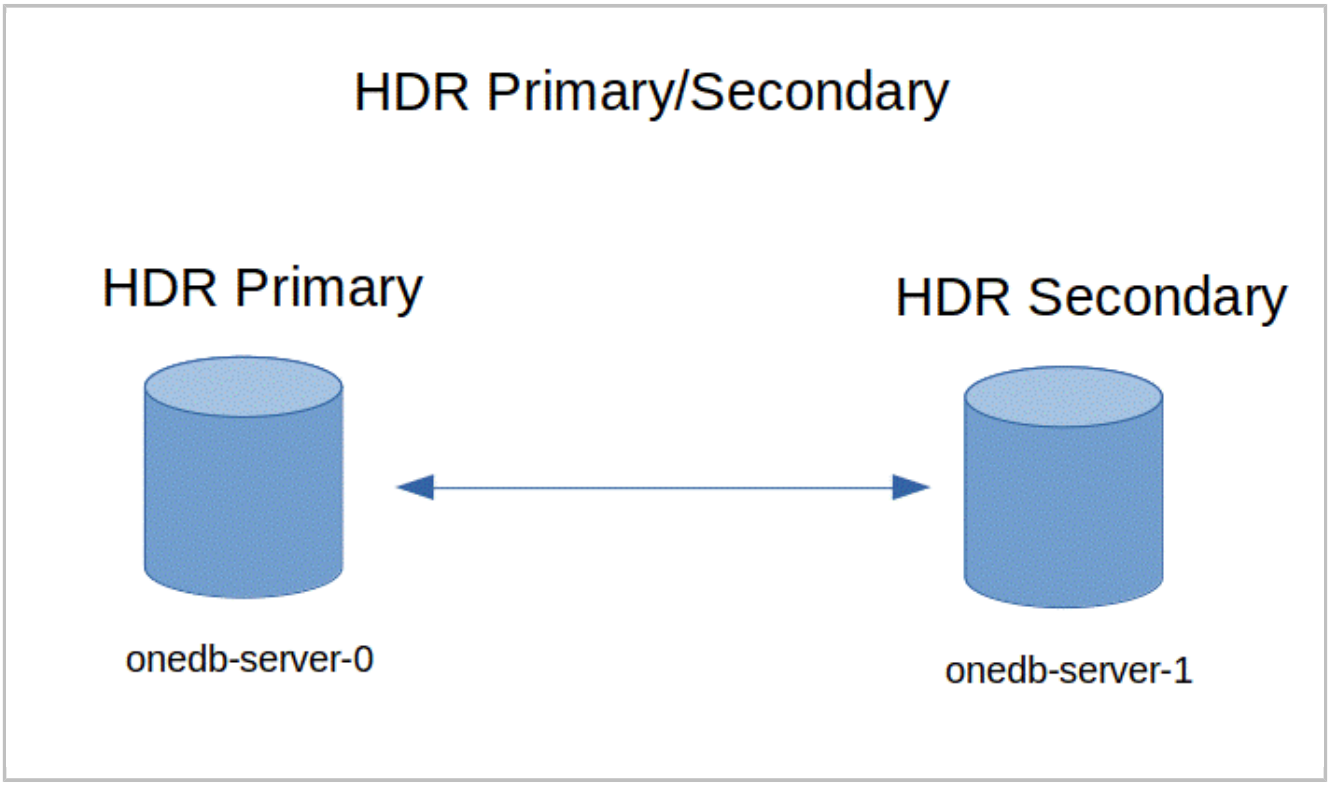


Automatic failure occurs and onedb-server-1 will be made the HDR Primary, and onedb-server-0 restarts as an HDR secondary.



Manual Failover

When the HDR primary server is non-responsive the kubernetes health scripts will fail and the HDR primary server pod will restart. The pod will be restarted as an HDR primary. If the server comes back to a healthy state, then no failover is required.



If the restart of the HDR primary does not come back to a healthy state then manual intervention is needed. You must login to the HDR Secondary, onedb-server-1 pod, and switch it to the HDR primary. The onedb-server-0 pod will not become healthy and will restart as the HDR secondary.

```
onmode –d make primary onedb1
```

If TLS encryption is being used the name of the server is **onedb1_ssl**. So, the command would be:

```
onmode –d make primary onedb1_ssl
```

Scale-Out

The OneDB SQL Data Store by default will start an HDR Primary + Secondary HA cluster. OneDB SQL Data Store allows the scaling of the OneDB Database server and the OneDB Connection manager. The default settings for both the **onedb.serverReplicaCount / onedbcm.cmReplicaCount** helm parameters are **2**.

Another helm chart parameter, **onedb.maxReplicacount,** controls the maximum number of servers that can be used. The default setting for this parameter is 10 and the max supported value is 10. This is an immutable value and once it is set its value cannot be changed.

The OneDB server pods are as follows:

- onedb-server-0: HDR Primary
- onedb-server-1: HDR Secondary
- onedb-server-[2-9]: HDR RSS

When an HDR secondary or RSS is created, **ifxclone** is used to clone the new server from the current HDR primary server. This applies to an initial setup or a scale out scenario.

The maximum number of replicas for the OneDB Connection manager (**onedbcm.cmReplicaCount**) is onedb.maxReplicaCount.

Manual Scale-Out

OneDB supports manual scale out for the OneDB server and the OneDB Connection Manager.

For the OneDB Server to scale out the number of servers manually set the his is accomplished by set the helm parameter **onedb.serverReplicaCount**.

For the OneDB Connection manager to scale out the number of connection managers manually set the his is accomplished by set the helm parameter **onedbcm.cmReplicaCount**.

Manual Scale out of Connection Manager

The onedbcm.cmReplicaCount helm chart parameter can be changed at any time with the helm upgrade command. You can increase or decrease the number of connection managers in the HA cluster by changing this value.

```
helm upgrade onedb-v1 -set onedbcm.cmReplicaCount=3 -f myvalues.yaml onedb-product
```

The pods for the connection manager are onedbcm-0, onedbcm-1, onedbcm-2, and so on.

Automatic Scale-out

The OneDB SQL Data store uses the kubernetes resource **Horizontal Pod Autoscaler (HPA)** to control how scaling will occur automatically. For more information on HPA refer to the kubernetes documentation here: (https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale).

Auto-scale is based on CPU usage and this is disabled by default. To enable auto-scaling set the **autoscale.enabled** helm chart parameter to **true** and set **autoscale.targetCPUUtilizationPercentage** to a percentage value where you want scaling to occur.

The minimum pods for auto scaling would be the helm chart parameter **onedb.serverReplicaCount** for the OneDB Server and **onedbcm.cmReplicaCount** for the Connection manager. The maximum pods for auto scaling would be the **onedb.maxReplicaCount** helm chart parameter.

Important: When enabling auto scaling OneDB Server and Connection Manager resources should be explicitly configured. See **onedb.resources** and o**nedbcm.resources** helm chart parameters.

Following table shows the HPA resource in kubernetes:

Table 5. $ kubectl get hpa Output

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
onedb-c5abcd-hpa	StatefulSet/onedb-server	8%/90%	2	10	2	3h34m
onedbcm-c5abcd-hpa	StatefulSet/onedbcm	5%/90%	2	10	2	3h34m

In the above output the **onedb-c5abcd-hpa** is for the OneDB server and **onedbcm-c5abcd-hpa** is for the Connection Manager. The CPU threshold is set to 90% for each and we see minimum pods is set to 2 with maximum set to 10. Currently there are 2 of each.

Automatic Scale-out

The OneDB SQL Data store uses the kubernetes resource **Horizontal Pod Autoscaler (HPA)** to control how scaling will occur automatically. For more information on HPA refer to the kubernetes documentation here: (https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale).

Auto-scale is based on CPU usage and this is disabled by default. To enable auto-scaling set the **autoscale.enabled** helm chart parameter to **true** and set **autoscale.targetCPUUtilizationPercentage** to a percentage value where you want scaling to occur.

The minimum pods for auto scaling would be the helm chart parameter **onedb.serverReplicaCount** for the OneDB Server and **onedbcm.cmReplicaCount** for the Connection manager. The maximum pods for auto scaling would be the **onedb.maxReplicaCount** helm chart parameter.

Important: When enabling auto scaling OneDB Server and Connection Manager resources should be explicitly configured. See **onedb.resources** and o**nedbcm.resources** helm chart parameters.

Following table shows the HPA resource in kubernetes:

Table 6. $ kubectl get hpa Output

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
onedb-c5abcd-hpa	StatefulSet/onedb-server	8%/90%	2	10	2	3h34m
onedbcm-c5abcd-hpa	StatefulSet/onedbcm	5%/90%	2	10	2	3h34m

In the above output the **onedb-c5abcd-hpa** is for the OneDB server and **onedbcm-c5abcd-hpa** is for the Connection Manager. The CPU threshold is set to 90% for each and we see minimum pods is set to 2 with maximum set to 10. Currently there are 2 of each.

Archive Restore Considerations

When an archive restore is initiated OneDB it will occur on onedb-server-0, pod #0. When a restore occurs OneDb will try to salvage the logical logs, backup up the current logical log.

If the primary is on onedb-server-1 and the onedb-server-0 pod is the secondary, then no salvaging of logical logs will happen. Which means the current logical log will not be archived and available for the restore.

To prevent this from occurring it is recommended to have onedb-server-0 be the Primary server in the HA cluster. If you are in a situation where the HDR primary is on onedb-server-1 you can force a switch over.

Manual Switch Over

If the onedb-server-1 server is the HDR primary and onedb-server-0 server is the HDR secondary you can switch these two roles when automatic failover is disabled by doing the following. Login to the onedb-server-0 server (current HDR Secondary) and perform the failover operation by running the following command:

```
onmode –d make primary onedb0
```

If TLS is enabled for the OneDB HA cluster use the following command:

```
onmode –d make primary onedb0_ssl
```

Automatic Switch Over

If the onedb-server-1 server is the HDR primary and onedb-server-0 server is the HDR secondary you can switch these two roles when automatic failover is enabled by doing the following.

During a planned downtime login to the onedb-server-1 (current HDR primary) and run the following commands:

```
onmode –c
# Wait for checkpoint to complete on primary & secondary
onmode –ky
```

After forcing a checkpoint (onmode –c) verify the checkpoint has completed on the primary and secondary servers by logging in to each and running onstat –m, looking for Checkpoint completed message. After verification of checkpoint, then run the onmode –ky command.

This will cause the HDR primary on onedb-server-1 to go offline. The onedb-server-0 will automatically failover from HDR secondary to HDR primary. And when onedb-server-1 comes back up it will restart as the HDR secondary.

Upgrading OneDB helm charts

When upgrading your helm chart, it is always recommended to take a database backup before upgrading the product. The helm upgrade command is used to upgrade the current release with new configuration. Or, it can be used to upgrade the current version of the chart to a new helm chart version.

OneDB SQL Data Store and the Connection Manager statefulsets support rolling upgrade. The upgrade process will start from the highest pod ordinal index to the lowest pod ordinal index. For example, onedb-server-1 is updated before onedb-server-0.

The goal of OneDB's upgrade process is to have as little interruption as possible. During an upgrade, kubernetes pods are restarted which will cause a slight interruption in write activity.

If the OneDB Database server does not need to perform a database conversion, then read activity can continue throughout the upgrade process. If a database conversion has to occur then there will be a slight interruption in read activity as well.

To maintain read activity during the upgrade process, your application must be designed with retry logic in it. When a pod is taken down so that it can be upgraded your application should retry its connection so it can connect to and use another server in the cluster.

Upgrading Current release

There may be times when you need to make changes to an existing running release of OneDB in kubernetes. This is performed using helm upgrade and providing the same installed chart with any new values. Parameter values you can change are:

- Set ReplicaCount
- Change container image
- Initiate onbar restore
- Change Connection Manager Service Type: Loadbalancer, ClusterIP, NodePort
- Enable/Disable Automatic failover using Connection Manager
- Change Connection Manager SLA policy: Workload, Round Robin

If the initial installation was performed with this:

```
helm install onedb-v1  -f myvalues.yaml production-onedb
```

The default installation of the helm chart will install an HA cluster with a primary and secondary OneDB server. If you wanted to manually scale the HA cluster to a 3rd server (RSS), you can use helm upgrade and specify a new serverReplicaCount value.

File: newvalues.yaml

```
onedb-product:
    onedb-sql:
        onedb:
            serverReplicaCount: 3
```

Issue the helm upgrade with the original and new values overrides.

```
helm upgrade onedb-v1  -f myvalues.yaml -f newvalues.yaml production-onedb
```

Upgrading 1.x.x.x to 2.x.x.x

A helm upgrade is not supported from OneDB 1.0.0.0 helm chart to a OneDB 2.x helm chart. There are major differences between these two helm chart versions that would prevent a helm upgrade.

When upgrading from a helm chart version using OneDB 1.x to 2.x then you must perform a data migration. It is recommended to use the dbexport.

Note: It is recommended to use the dbexport and dbimport utilities.

Upgrading from 2.0.0.0 version to current version

When upgrading to a new helm chart version you can use the helm upgrade command. This will also most likely be upgrading the version of OneDB database server. For example. Upgrading helm chart version 0.3.52 to 0.4.12 is an upgrade from OneDB 2.0.0.0 to OneDB 2.0.1.0.

When upgrading to a new helm chart version you can use the helm upgrade command. This will also most likely be upgrading the version of OneDB database server. For example. Upgrading helm chart version 0.3.52 to 0.4.12 is an upgrade from OneDB 2.0.0.0 to OneDB 2.0.1.0.

```
helm install onedb-v1  -f myvalues.yaml production-onedb-0.3.52
```

To upgrade to helm chart production-onedb-0.4.12, helm chart version 0.4.12 running OneDB 2.0.1.0, run the following helm upgrade command.

```
helm upgrade onedb-v1  -f myvalues.yaml production-onedb-0.4.12
```

In the above example, the helm chart production-onedb-0.3.52 is used for the OneDB 2.0.0.0 OneDB product. And the helm upgrade command upgrades the helm chart to production-onedb-0.4.12 which the helm chart running OneDB 2.0.1.0.

Upgrading from 2.0.0.0 version to current version

When upgrading to a new helm chart version you can use the helm upgrade command. This will also most likely be upgrading the version of OneDB database server. For example. Upgrading helm chart version 0.3.52 to 0.4.12 is an upgrade from OneDB 2.0.0.0 to OneDB 2.0.1.0.

When upgrading to a new helm chart version you can use the helm upgrade command. This will also most likely be upgrading the version of OneDB database server. For example. Upgrading helm chart version 0.3.52 to 0.4.12 is an upgrade from OneDB 2.0.0.0 to OneDB 2.0.1.0.

```
helm install onedb-v1  -f myvalues.yaml production-onedb-0.3.52
```

To upgrade to helm chart production-onedb-0.4.12, helm chart version 0.4.12 running OneDB 2.0.1.0, run the following helm upgrade command.

```
helm upgrade onedb-v1  -f myvalues.yaml production-onedb-0.4.12
```

In the above example, the helm chart production-onedb-0.3.52 is used for the OneDB 2.0.0.0 OneDB product. And the helm upgrade command upgrades the helm chart to production-onedb-0.4.12 which the helm chart running OneDB 2.0.1.0.

Troubleshooting OneDB

The following documentation talks about some troubleshooting techniques that you might use with OneDB in a kubernetes environment.

From the viewing of log files, to enabling a higher level of logging. To disabling the liveness, probe for the OneDB server pod to prevent kubernetes from automatically restarting the OneDB server pods.

Contact OneDB Support with the diagnostic logs and data mentioned in this section as needed.

Troubleshooting Pods

Each pod that is started by kubernetes goes through a series of steps. Some of the common steps you might see are PodInitializing, Container Creating, Pending, Init, Running, ImagePullBackoff.

If a pod seems to be stuck in a state for a period, some of the following techniques can be used:

```
kubectl get pods


NAME                                      READY       STATUS          RESTARTS   AGE
my-nfs-server-provisioner-0                1/1       Running            0      103s
onedb-operator-86d899b5bf-hklq9            0/1       ImagePullBackOff   0      43s
sofy-1-grafana-b7b5f958d-lxcxf             0/2       PodInitializing    0      44s
sofy-1-ksmetrics-6d4677b7d5-zhtmh          1/1       Running            0      44s
sofy-1-odbp-explore-55c9db47c4-nqx8p       0/1       ErrImagePull       0      42s
sofy-1-odbp-mongo-6f6df887df-gn896         0/1       Init:0/1           0      43s
sofy-1-odbp-rest-64f94dfd98-bzj7x          0/1       Init:0/1           0      43s
```

Troubleshooting ImagePullBackoff, Pending pods

When a pod doesn't make it to the Init/Running state **kubectl describe pod** is commonly used to try to gather more information as to what the problem might be. In the above output we see a few pods in ErrImagePull/ImagePullbackOff.

```
kubectl describe pod onedb-operator-86d899b5bf-hklq9

Events:
  Type     Reason     Age                    From               Message
  ----     ------     ----                   ----               -------
  Normal   Scheduled  34m                    default-scheduler  Successfully assigned sofy-
testing/onedb-operator-86d899b5bf-hklq9 to kind-worker4
  Normal   Pulling    32m (x4 over 34m)      kubelet            Pulling image
"hclcr.io/sofy/services/onedb/onedb-operator:2.0.1.0"
  Warning  Failed     32m (x4 over 34m)      kubelet            Failed to pull image
"hclcr.io/sofy/services/onedb/onedb-operator:2.0.1.0": rpc error: code = Unknown desc = failed
to pull and unpack image "hclcr.io/sofy/services/onedb/onedb-operator:2.0.1.0": failed to
resolve reference "hclcr.io/sofy/services/onedb/onedb-operator:2.0.1.0": unexpected status code
[manifests 2.0.1.0]: 401 Unauthorized
  Warning  Failed     32m (x4 over 34m)      kubelet            Error: ErrImagePull
  Warning  Failed     32m (x6 over 34m)      kubelet            Error: ImagePullBackOff
  Normal   BackOff    4m31s (x128 over 34m)  kubelet            Back-off pulling image
"hclcr.io/sofy/services/onedb/onedb-operator:2.0.1.0"
```

You can see here that we failed to pull the image. This gives us some direction in trying to diagnose this issue.

Troubleshooting init pods

OneDB uses init containers to perform setup functions before a specific pod is fully functional. When a pod is in the init state you can run a kubectl logs command to get information about the pod. When running the kubectl logs command on an init-container you need to know the name of the init-container. This can be obtained from the kubectl describe command.

```
kubectl describe pod sofy-1-odbp-mongo-6f6df887df-gn896

Events:
 Type    Reason   Age              From    Message
 ----    ------   ----             ----    -------
 Normal  Pulled   22m (x6 over 63m)    kubelet  Container image "openjdk:8-alpine" already present
on machine
 Normal  Created  22m (x6 over 63m)    kubelet  Created container onedb-mongo-init
 Normal  Started  22m (x6 over 63m)    kubelet  Started container onedb-mongo-init
 Warning  BackOff  4m49s (x25 over 48m)  kubelet  Back-off restarting failed container
```

Once you find the name of the init container, you can run a kubectl logs command. You specify the pod and the name of the init container in the kubectl logs command.

```
kubectl logs sofy-1-odbp-mongo-6f6df887df-gn896 -c onedb-mongo-init
```

Below is a sample output and we can see there is a problem connecting to the OneDB Database server.

```
Running Main
```

```
SQL Service Test Unsuccessful. Server not ready
-908 : com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
```

Troubleshooting running pods

A pod's desired state is to get to a running state with a Ready of 1/1. If you see a Ready Status of 0/1 or see several Restarts for the pod, then you may need to investigate further. The **kubectl log** command can be used to get more information on what the container/pod is doing.

```
kubectl get pods

NAME                                   READY   STATUS    RESTARTS   AGE
helm-1-odbp-explore-686bf69c88-czntt   1/1     Running   0          8m2s
helm-1-odbp-mongo-5bc4d5bb94-54jh4     1/1     Running   0          8m2s
helm-1-odbp-rest-5bd6b956cc-8gqfw      1/1     Running   0          8m2s
my-nfs-server-provisioner-0            1/1     Running   0          8m23s
onedb-operator-6dfdd9bb7d-4phld        1/1     Running   0          8m2s
onedb-server-0                         0/1     Running   0          7m46s
```

Pod onedb-sever-0 hasn't moved into a running state yet and we want to dig deeper into what is going on with this specific pod.

```
kubectl logs onedb-server-0

20:59:00 Peer node onedb1 has version 131077
20:59:00 RSS Server onedb1 - state is now connected
20:59:00 setting version information for onedb1 131077
2022-01-08 20:59:03 LICENSING: <Information> Reacquire licenses: current allocation != expected count
2022-01-08 20:59:03 LICENSING: <Information> Processing current Capability
2022-01-08 20:59:13 LICENSING: <Information> Reacquire licenses: current allocation != expected count
2022-01-08 20:59:13 LICENSING: <Information> Processing current Capability
20:59:13 HDR TIMEOUT - log buffers being sent to onedb1

20:59:13 Error receiving a buffer from RSS onedb1 - shutting down
```

```
20:59:14 RSS Server onedb1 - state is now disconnected
20:59:14 RSS onedb1 deleted
2022-01-08 20:59:23 LICENSING: <Information> Reacquire licenses: current allocation != expected count
2022-01-08 20:59:23 LICENSING: <Information> Processing current Capability
```

Enable/Disable Liveness probe

When doing any type of diagnostic work on a container/pod, it is important that the liveness probe does not take effect and restart the pod. To prevent this from happening you can disable the liveness probe for the OneDB Database server.

To disable use the following kubectl annotation:

```
kubectl annotate pod onedb-server-0 livenessprobe=disabled –overwrite=true
```

Once you are done with your diagnostic work you should re-enable the liveness probe.

To enable, use the following kubectl annotation:

```
kubectl annotate pod onedb-server-0 livenessprobe=enabled –overwrite=true
```

Kubernetes events

Another log of events that can be reviewed/monitored is the Kubernetes events. Run the kubectl get events command and sort or filter this data accordingly.

```
kubectl get events
```

Log in to pod

There may be a need to login to a pod or the init container to obtain more diagnostic information than you get with kubernetes commands. First identify the pod you need to login.

```
kubectl get pods |grep onedb

onedb-operator-67f5d6cd9b-wxcg2 1/1 Running    0 4m
onedb-server-0                  1/1 Running    0 4m
onedb-server-1                  0/1 Init:0/1   0 2m
onedbcm-0                       1/1 Running    0 4m
```

Run the kubectl command to login to the kubernetes pod:

```
kubectl exec --stdin  --tty onedb-server-0 -- bash
```

Once you've logged in to the pod/container, you can move around and view log files as you would on any Linux system.

Log in to init container

To login to an init container, you must first find the name of the init container. This is done using the **kubectl describe pod** command.

```
kubectl describe pod onedb-server-1
```

```
Events:
 Type    Reason     Age   From               Message
 ----    ------     ----  ----               -------
 Normal  Scheduled  76s   default-scheduler  Successfully assigned sofy-testing/onedb-server-0 to
kind-worker
 Normal  Pulling    74s   kubelet            Pulling image "informix-docker-dev-local.us-
artifactory1.nonprod.hclpnp.com/releases/onedb-server:2.0.1.1"
 Normal  Pulled     74s   kubelet            Successfully pulled image "informix-docker-dev-local.us-
artifactory1.nonprod.hclpnp.com/releases/onedb-server:2.0.1.1" in 355.243101ms
 Normal  Created    74s   kubelet            Created container onedb-init
 Normal  Started    74s   kubelet            Started container onedb-init
```

Once you find the name of the init container, you can run the kubectl exec command and login to the init container.

```
Once you find the name of the init container you can run the kubectl exec command and login to the init
 container.
```

Once you've logged in to the pod/container, you can move around and view log files as you would on any Linux system.

Custom init container

Creating a custom init container is a more advanced topic for kubernetes users. An init container is designed to run before starting the main container.

Potential use for custom init container:

- Debug/patch container storage
- Custom container to load data spaces
- Perform any operation on the container/pod prior to starting the container

To use a customer init container, use the following helm parameter override values:

```
onedb:
    customInitImage: gcr.io/google-containers/busybox:latest
    customInitImageCmd: /bin/customization.sh
```

If you needed to login to this container, the name of the init container is **onedb-custom-init**, although we could use the kubectl describe pod command to determine this.

Charts 0.4.12

This version includes the following enhancements:

- Connection Manager on page 172
- HA Scale out on page 192
- Automatic Failover on page 190
- Automatic Backups on page 183
- Support Multiple PVs for Storage on page 161
- Cloud Native method for Creating Spaces on page 165
- Cloud Native method for configuration/Users on page 166
- Support custom Init Container on page 167

What's New in this Helm Chart Version

This section includes information about the new, enhanced capabilities added in this version of the helm chart :

- Connection Manager on page 172
- HA Scale out on page 192
- Automatic Failover on page 190
- Automatic Backups on page 183
- Support Multiple PVs for Storage on page 161
- Cloud Native method for Creating Spaces on page 165
- Cloud Native method for configuration/Users on page 166
- Support custom Init Container on page 167

Supported Platforms

The OneDB Helm charts have been tested on the following platforms:

- Google Kubernetes Engine (GKE) (https://cloud.google.com/kubernetes-engine)
- AWS Elastic Kubernetes Service (EKS) (https://aws.amazon.com/eks)
- Azure Kubernetes Service (AKS) (https://azure.microsoft.com/en-us/services/kubernetes-service)
- Redhat OpenShift Container Platform (OCP) (https://www.redhat.com/en/technologies/cloud-computing/openshift/container-platform)

Architectural Overview

Installing and deploying OneDB in a cloud-native environment is a new way of looking at things. An evolution of how OneDB is or can be deployed has occurred: starting with on-premises, to in the cloud in Virutal machines, to in the cloud in a highly scalable Kubernetes environment.

In the past, you would have acquired a physical machine, installed the OneDB database server on that machine and been responsible for the maintenance and upgrades on the machine as well as maintenance of the OneDB Database server.

There was then a move to the cloud and the use of Virtual machines in that cloud. Virtual machines made it possible to start up a machine and run a playbook that would install OneDB and configure accordingly. You might do this in your own cloud or a public cloud.

Then more recently, there is the move to a highly scaleable Kubernetes environment. This approach uses containerization of products and pieces of an entire solution. It allows for great flexibility with many benefits. You may use your own Kubernetes solution or a cloud provided Kubernetes from Google, Amazon or Microsoft for example.

General Terminology

To undestand how OneDB Database server is deployed in a kubernetes environment, it is important that you have a basic knowledge of certain terms:

- Container on page 142
- Docker on page 142
- Microservices on page 142
- OneDB HA Cluster on page 142

Container

A container image is a lightweight standalone executable package of software that includes everything to run an application includeing system libraries, tools etc.

Docker

Docker is the leading technology for containizeration. When people think of containers they typcially think of docker. Although it is not the only container technology.

Microservices

A microservices architecture is a method of designing an overall solution to be broken up into smaller parts instead of a single monolithic application. Containers make this a natural path of software development as different pieces can be represented by a different container image.

OneDB HA Cluster

This use of the term cluster refers to the High availability nature of 2 or more OneDB Database servers working together. A 2 nodes OneDB HA cluster will consist of a OneDB HA Primary server and a OneDB HA Secondary server. More servers can be added into a OneDB HA Cluster, in this context the additional servers would be added as OneDB HA RSS nodes.

Kubernetes Terminology

Node

A node is a virtual machine or phyiscal machine with CPU/RAM resources. This is the hardware component that makes up a kubernetes cluster. Example nodes are worker nodes and master nodes.

Pod

A pod is the simplest unit that exists within kuberenetes. Typically this is 1 more more containers. It is pods that get scheduled to run on kubernetes nodes.

Cluster (kubernetes)

Is made up of 1 or more nodes. They provide a resource for a kubernetes solution to be deployed into and managed.

Service

An abstract API object that exposes an application's network services.

Helm chart

A helm chart is a collection of files that describe a related set of kubernetes resources. A helm chart is typically a group of yaml files and other associated files that is used to deploy a solution into kubernetes.

Operator

A kubernetes operator is an application specific controller that extends the functionality of the kubernetes API.

LoadBalancer

A kubernetes object that allows you to expose an external IP address to outside the kubernetes cluster.

OneDB Deployment Resources

When deploying a OneDB helm chart a group of resources will be created. The resources created will depend on the specific OneDB Helm chart that is used.

The OneDB-sql helm chart will deploy the following resources:

- onedb-operator pod
- onedb-server-X pod

- onedbcm-X pod
- onedbcm-cm-service

The OneDB-mongo helm chart will deploy the following resources:

- odbp-mongo pod
- odbp-mongo service
- OneDB-sql chart

The OneDB-rest helm chart will deploy the following resources:

- odbp-rest pod
- odbp-rest service
- OneDB-sql chart

The OneDB-explore helm chart will deploy the following resources:

- odbp-explore pod
- odbp-explore service

The OneDB-product helm chart will deploy the following resources:

- OneDB-sql chart
- OneDB-mongo chart
- OneDB-rest chart
- OneDB-explore chart

Pods

onedb-operator

The purpose of the operator pod is to manager the OneDB HA cluster. By default, a OneDB HA Cluster is started with an HDR primary and secondary server, along with two connection managers.

onedb-server-x

This is the OneDB Database server pod. When deployed, a statefulset is used which will be assigned an ordinal index starting with 0. So, OneDB HA cluster with a primary secondary will have onedb-server-0 and onedb-server-1.

onedbcm-x

This is the OneDB Connection manager pod. It will be assigned an ordinal index starting with 0. By default, 2 connection managers are started. onedbcm-0 and onedbcm-1.

odbp-mongo

This is the OneDB Mongo Listener pod. It is started when the OneDB Mongo chart is deployed. It is used to connect to the OneDB Database server using the Mongo API.

odbp-rest

This is the OneDB REST Listener pod. It is started when the OneDB REST chart is deployed. It is used to connect to the OneDB Database esrver using RESTFUL services.

odbp-explore

This is the OneDB Explore pod. It will deploy the OneDB Explore administration and monitoring tool providing a web admin and monitoring GUI. It can be used to administer one or more OneDB Database servers.

Services

odbp-explore

This is the OneDB Explore service that can be used to access the OneDB Explore product.

odbp-mongo

This is the OneDB Mongo service that is used to access the OneDB Database server using the Mongo API.

odbp-rest

This is the OneDB REST service that is used to access the OneDB Database server using RESTFUL services.

onedbcm-cm-service

This is the OneDB Connection Manager service that is used to access the OneDB Database server using the SQLI + DRDA protocol. EX: JDBC, ODBC.

Prerequisites

To install OneDB into a kubernetes cluster, following prerequisites are needed:

- kubectl
- helm
- ReadWriteMany storage class

Note: To install HCL Sofy Solution into a kubernetes cluster, there may be additional requirements. For more information on the installation instructions for HCL Sofy, see (https://hclsofy.com/ua/guides#installing-solutions-step-by-step-instructions)

.

Kubectl

The kubernetes command line tool, kubectl, is used to run commands and interact with a kubernetes cluster. This is used for managing and interacting with OneDB in kubernetes.

Helm

The helm tool is used to install OneDB in a kubernetes cluster. Helm is a package manager for Kubernetes and is used to install a helm chart.

A helm chart is simply a set of kubernetes yaml manifests that are combined into a single package. This provide an easy method to install a group of kubernetes manifests as a single package.

For installations steps and more information on helm, see: https://helm.sh

RWM Storage

RWM storage is needed to support High availability cluster options with OneDB. Listed are some available options to install RWM storage, but not limited to these. Following options have been tested and verified to work with OneDB.

Enable one and only one of the following options:

Cloud specific options that can be used for these specific cloud providers are:

- Google FireStore on page 146
- AWS Elastic filesystem on page 147
- Azure on page 147

Cloud generic options that can be installed into an existing kubernetes cluster are:

- nfs-server-provisionser on page 147
- rook-ceph on page 149
- rook-nfs on page 149

Google FileStore Configuration

1. See the Google filestore Instructions (https://cloud.google.com/filestore/docs/quickstart-console).
2. The following OneDB helm chart configuration values need to be set to use the Google filestore:

Parameter	***Description***	***Value***
nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.googleFilestore.enable	Set to 'true' to enable Google Filestore	true

nfsserver.googleFilestore.filestoreIP	IP address of the Filestore instance	''
nfsserver.googleFilestore.filestoreShare	Name of the File share of the instance	''

AWS Elastic Filesystem Configuration

1. See AWS filesystem Instructions (https://docs.aws.amazon/eks/latest/userguide/efs-csi.html) .
2. The following OneDB helm chart configuration values need to be set to use the AWS Elastic filesystem.

Parameter	***Description***	***Value***
nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.awsEFS.enable	Set to 'true' to enable AWS Elastic filestore	true
nfsserver.awsEFS.EFSServer	IP address of the Filestore instance	''
nfsserver.googleFilestore.filestoreShare	Name of the File share of the instance	''

Azure File share Configuration

1. See Azure File share instructions (https://docs.microsoft.com/en-us/azure/aks/azure-files-volume).
2. Following OneDB helm chart configuration values need to be set to use the Azure File share:

Parameter	***Description***	***Value***
nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.azureFS.enable	Set to 'true' to enable Azure File share	true
nfsserver.azureFS.secretname	Kubernetes secret to use	''
nfsserver.azureFS.shareName	Azure file share name	''

Install and Configure nfs-server-provisioner

1. Add the nfs-server-provisioner helm repo.

```
helm repo add kvaps https://kvaps.github.io/charts
```

2. Install the helm chart for the nfs-server-provisioner. Specify the following parameters:

Parameter	***Description***	***Value***
persistence.size	Set this to a value of the NFS PV size	50Gi
persistence.enabled	Set to ‘true’ to enable NFS	true
persistence.storageClass	Set this to ‘standard’	standard
storageClass.create	Set to ‘true’	true
storageClass.name	Set thos to a unique Name	onedb-nfs-<namespace>
storageClass.mountOptions		{vers=4.1}

- **storageClass.name**: This is cluster wide so it is recommended to include the namespace in the name to provide uniqueness.
- **storageClass.mountOptions**: Onedb has been tested with NFS V4.1

```
helm install onedb-nfs-server-provisioner kvaps/nfs-server-provisioner   \
--version 1.3.1
--set persistence.enabled=true
--set persistence.storageClass="standard”
--set persistence.size=50Gi
--set storageClass.create=true
--set storageClass.name=-onedb-nfs-my-ns
 --set storageClass.mountOptions={vers=4.1}
```

3. The following OneDB helm chart configuration values need to be set to use the NFS server provisioner.

Parameter	***Description***	***Value***
nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.other.enable	Set to ‘true’ to enable NFS	true
nfsserver.other.storageClass	Set this to the storage class of the NFS	onedb-nfs-<namespace>

- **nfsserver.other.storageClass:** This is set to the the storageClass name specified in the creation of the nfs server provisioner

Install and Configure rook-ceph

1. See the rook-ceph Prerequisites: (https://rook.io/docs/rook/v1.7/pre-reqs.html) .

   Note: Some environments you may need to provision and use *Ubuntu with containerd* node pool instead of the default *GKE container-Optimized OS* (COS).

   .
2. Follow the instructions for rook-ceph: (https://rook.io/docs/rook/v1.7/quickstart.html).
3. Configure a shared file system for rook: (https://rook.io/docs/rook/v1.7/ceph-filesystem.html).
4. Following OneDB helm chart configuration values need to be set to use rook-ceph:

Parameter	***Description***	***Value***
nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.other.enable	Set to 'true' to enable NFS	true
nfsserver.other.storageClass	Set this to the storage class of the NFS	onedb-nfs-<namespace>

   - **nfsserver.other.storageClass:** This is set to the the storageClass name specified in the creation of rook-ceph.

Installation of rook-nfs

1. Follow the instructions for rook-nfs (https://github.com/rook/rook/blob/master/Documentation/nfs.md).
2. Before Installing OneDB modify the *template/nfs_other_pvc.yaml* file in the helm chart and change the **accessModes:** value from ***ReadWriteMany*** to ***ReadWriteOnce***.
3. After creating the Storage Class, refer to the *sc.yaml* file for rook-nfs. This will contain the storageclass name.
   - Default: rook-nfs-share1.
   - The storage name is needed when installing OneDB.

Parameter	***Description***	***Value***
nfsserver.volumeSize	Set this to a value of the NFS PV size	50Gi
nfsserver.other.enable	Set to 'true' to enable NFS	true
nfsserver.other.storageClass	Set this to the storage class of the NFS	rook-nfs-share1

   - **nfsserver.other.storageClass:** This is set to the the storageClass name specified in the creation of rook-nfs.

OneDB Requirements and Recommendations

The OneDB database server is designed to be able to run on small devices like a Raspberry pi up to large Servers with 128 cores. The architecture of OneDB is flexible and allows you to run in these different environments with different configurations.

It is important to note that these are recommendations and not requirements. As one user may be able to run their workload on a small device like a Raspberry pi, but another user needs 32 CPUs and 100GB of memory.

When talking about recommendations, we typically refer to CPU, Memory and sometimes disk space.

OneDB Disk/Volume Recommendations

This depends on the amount of data and workload you will have in your OneDB Database server. So every database system will be different. But if High thoughput is needed then we recommend SSD drives to be used. And for your NFS shred drive, spinning disks are ok to use.

Below is a priority of spaces to be setup with SSD if possible. This is not required but the more spaces/volumes setup with SSD drives the better performance can be achieved with the OneDB Database server.

Space/Volume	Drive Type
Logical Log Dbspace	SSD drive(s)
High Volume space	SSD drive(s)
Temp Spaces	SSD drive(s)
Physical Log Dbspace	SSD drive(s)
Low volume space	SSD/Spinning drive(s)
RWM NFS	Spinning drive(s)

The amount of disk space allocated to each of these spaces and volumes is dependent on the size of your data and workload. It is recommended that the RWM NFS volume be approx 3-5 times the size of the total dbspaces if you plan to use the automated backups. We retain 3 archives of the OneDB database server.

Note: It is important to note that you can use all spinning disks and if needed you can put all spaces on a single volume, a separate volume is needed for the RWM NFS. These recommendations are given to provide the best performance possible for a production system.

OneDB Minimum CPU/Memory Recommendations

For a OneDB solution the following can be used as guidelines for the OneDB Server, the Connection Manager, the Mongo wire listener, and the REST wire listener. With OneDB Explore, the minimmum recommendation should be plenty.

Resource	Minimmum Recommendation	General Recommendatoin
CPU	1 core	2 cores
Memory	512 MB	8 GB

As with all systems the more resources, CPU and Memory that a system has the better performance can be achieved. If you find that your workload has a high number of quick connections using the REST or Mongo protocols you may want to increase resources in that area.

The more CPU that is provided to the OneDB system allows you to configure more CPUvps and the more Memory that is provided allows you to configure more memory for Buffers and other database operations.

Note: It is possible to run OneDB with less CPU and Memory. These recommendations are given to provide the best performance possible for a production system.

OneDB Minimum Kubernetes Recommendations

When a OneDB Helm chart is deployed you can specify the minimmum and maximum amount of resources that Kuberentes will use.

When scheduling a pod on a kubernetes node the pod specification can request minimmum resources required. If no node is available with those resources the pod will not be scheduled.

Example: If a pod has a resource.request.cpu of 1, kubernetes will attempt to schedule the pod on a node with >= 1 cpu. If not available, then the pod will not be scheduled.

The following are the current values set in the OneDB Helm Charts.

Pod	Resource	Request	Limit
onedb	CPU	.1 CPU	24 CPU
	Memory	2GB	32GB
CM	CPU	.1 CPU	1 CPU
	Memory	100 MB	500 MB
Mongo/REST	CPU	.1 CPu	2.1 CPU
	Memory	128MB	1GB

Pod	Resource	Request	Limit
Explore	CPU	.1 CPU	2 CPU
	Memory	64MB	512MB

The OneDB Helm charts are also configured by default to not allow two OneDB server pods to be scheduled on the same node. See onedb.nodeSelectorRequired configuration parameter.

The OneDB Helm chart is also configureed to not allow two OneDB Connection Manager pods to be scheduled on the same node. See onedbcm.nodeSelectorRequired configuration parameter.

This does not prevent a Connection manager pod from being scheduled on the same node as a OneDB server pod.

For best performance in a production system it is recommended to configure Affinity along with taints and tolerations to have full control of where the OneDB pods will be scheduled. This will allow you to control the resources available to the individual running pod.

Minimum Recommendation:

- 1 node per OneDB server pod
- 2 nodes for all other pods to be schedued on

General Recommendation: For best performance possible in a production system.

- Use affinity, taints and tolerations
- Configure 1 node per OneDB Server pod
- Configure 1 node per CM
- 1 node or Mongo wire listener
- 1 node for REST wire listener
- Configure 1-2 nodes for other pods

Note: It is possible to run a OneDB Helm chart with fewer nodes. These recommendations are given to provide the best performance possible for a production system.

Overview of Installation

OneDB is deployed into a kubernetes cluster using helm charts. A helm chart is a collection of files that describe a related set of kubernetes resources. A helm chart is typically a group of yaml files and other associated files that is used to deploy a solution into kubernetes.

Before installing a helm chart, you need access to the cluster. The Helm CLI is used to perform the install/uninstall and manage a helm release. Helm is commonly referred to as the package manager for kubernetes. For more information on helm and the installation instructions see: (https://helm.sh).

The kubectl CLI is a kubernetes command line tool to interact with and manage resources in a kubernetes cluster. You can use this tool to verify your installation. For more information on kubectl and installation, see: (https://kubernetes.io).

OneDB Helm Charts

There are five helm charts for OneDB. These helm charts are listed below with a description of each:

- **OneDB SQL Data Store (onedb-sql):** This is the Helm chart that contains the OneDB Database server. It uses a kubernetes operator to deploy a statefulset in a kubernetes environment.
- **OneDB Rest Data Store (onedb-rest):** This is a helm chart that deploys the OneDB Database server with the REST listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Document Data Store (onedb-mongo):** This is a helm chart that deploys the OneDB SQL Data Store with the Mongo listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Explore (onedb-explore):** This is a Helm chart that contains only the OneDB Explore UI.
- **OneDB Product (onedb-product):** This is a Helm chart that contains OneDB SQL Data Store, OneDB Rest, OneDB Document and OneDB Explore all as subcharts.

Differences in Standalone and Solution Factory helm charts

Note: HCL does not currently provide standalone helm charts the only way to get a helm chart for OneDB is through the Solution Factory (Sofy).

The Solution Factory (Sofy), is an Enterprise Kubernetes Solution catalog. Sofy allows you to pick and choose various products to create an overall solution. You can choose one of the OneDB services or Products from the catalog and it will be included in an overall solution with the OneDB helm chart included as a subchart in the Sofy solution. With a Sofy solution helm chart other charts will be included as subcharts like prometheius and grafana and the Sofy UI and of course any product chosen in the catalog.

The main difference between a OneDB helm chart that is installed with a Sofy solution and installed on its own are:

1. Other products/subcharts are included in the Sofy chart.
2. Helm chart overrides at a different level.

helm install

Helm install is used to install a helm chart. This command can point to a path of a directory of an unpacked chart, or a packaged chart. Ex. (chart.tgz).

HCL does not currently provide standalone helm charts the only way to get a helm chart for OneDB is through the Solution Factory (Sofy).

```
helm install [NAME] [CHART] [flags]
```

Installing an unpacked directory chart:

```
 helm install onedb1 onedb-sql
```

Installing a packaged chart (tgz):

```
helm install onedb1 onedb-sql.tgz
```

helm overrides

To override default values in the helm chart you can use --set on the command line. Or you can specify a file with a list of overrides.

Installing with set overrides:

```
 helm install onedb2 --set hclFlexnetURL=flex-net-xxxxx --set hclFlexnetID=xxxxxxx  onedb-sql
```

Installing with a overrides in a file:

```
helm install onedb2-f myvalues.yaml onedb-sql
```

File: myvalues.yaml

```
hclFlexnetURL:  flex-net-xxxxx
hclFlexnetID: xxxxxx
```

Verify Installation

After performing a helm install you can use the kubectl tool to verify the installation. The following resources are some of the items to verify within your kubernetes cluster.

- pods
- services
- deployments
- statefulsets

```
kubectl get pods
```

```
kubectl get services
```

```
kubectl get deployments
```

```
kubectl get statefulsets
```

These commands will show the status of each of these resources. For example, the pods need to be in a running state.
If any of these resources are not in a functioning running state you can use kubectl to diagnose. See the kubernetes documentation for more information on kubectl.

Install a Standalone helm chart

HCL does not currently provide standalone helm charts the only way to get a helm chart for OneDB is through the Solution Factory (Sofy).

Install a Solution Factory helm chart

When installing OneDB in a Sofy solution, it will be included as a subchart in the helm chart that is created from the Sofy catalog along with other Solution factory charts like Prometheus, grafana, Sofy console, etc. For more information about Solution Factory see: (https://hclsofy.com/ua/guides).

Before a Sofy helm chart can be installed, there are required steps to be taken. See the step by step instructions for installing a Sofy solution: (https://hclsofy.com/ua/guides#installing-solutions-step-by-step-instructions) .

License Requirements

The OneDB database server requires a license to be used. This is set using *hclFlexnetURL* and *hclFlexnetID* values in the helm chart. Below is a values override file that sets these parameter values. This values to be used for these parameters will be obtained from HCL.

File: myvalues.yaml

```
hclFlexnetURL:  flex-net-xxxxx
hclFlexnetID: xxxxxx
```

A Sofy solution uses a service named **anchor**. This service is used for license management. The OneDB helm charts use anchor but don't need the amount of resources set by default in a Sofy solution helm chart.

You can override the resources used by this **anchor** service by using the following values override file.

File: anchor.yaml

```
anchor:
    resources:
        cpu: 250m
```

Install OneDB SQL Data Store (onedb-sql)

The OneDB SQL Data Store helm chart will install the HCL OneDB database.

HCL OneDB is an enterprise grade database for storing and processing the following types of data:

- Relational (Table based)
- Document (Json Based)
- Timeseries (Time based)
- Spatial (Coordinate based)

Access to OneDB SQL Data Store is through the native SQLI protocol. HCL OneDB provides drivers for different programming languages to provide this connectivity. Ex. Java, Python, NodeJS, ODBC.

After all system requirements and prerequisites have been addressed you are ready to install the Sofy helm chart. The command below is a sample install that uses a file named myvales.yaml to provide any overrides to the helm chart values and anchor.yaml to set the anchor service's resources.

```
helm install sql-v1  -f myvalues.yaml -f anchor.yaml production-onedb-sql
```

Install OneDB RESTful Data Store (onedb-rest)

The OneDB RESTful Data Store helm chart will install the HCL OneDB database along with the OneDB REST listener.

HCL OneDB is an enterprise grade database for storing and processing the following types of data:

- Relational (Table based)
- Document (Json Based)
- Timeseries (Time based)
- Spatial (Coordinate based)

Access to OneDB RESTful Data store is through the REST API. This allows you to use language of choice that supports RESTful services.

After all system requirements and prerequisites have been addressed you are ready to install the Sofy helm chart. The command below is a sample install that uses a file named myvales.yaml to provide any overrides to the helm chart values and `anchor.yaml` to set the anchor service's resources.

```
helm install rest-v1  -f myvalues.yaml -f anchor.yaml production-onedb-rest
```

Install OneDB Document Data Store (onedb-mongo)

The OneDB Document Data Store helm chart will install the HCL OneDB database along with the OneDB Mongo Listener.

HCL OneDB is an enterprise grade database for storing and processing the following types of data:

- Relational (Table based)
- Document (Json Based)
- Timeseries (Time based)
- Spatial (Coordinate based)

Access to OneDB Document Data store is through the MongoDB protocol. This allows you to use any language that supports a MongoDB driver.

After all system requirements and prerequisites have been addressed you are ready to install the Sofy helm chart. The command below is a sample install that uses a file named myvales.yaml to provide any overrides to the helm chart values and anchor.yaml to set the anchor service's resources.

```
helm install mongo-v1  -f myvalues.yaml -f anchor.yaml production-onedb-mongo
```

Install OneDB Explore (onedb-explore)

The OneDB Explore helm chart will install the OneDB Explore web console. The web console is used for visualizing, monitoring, alerting and administering an HCL OneDB server instances.

HCL OneDB Explore features include:

- Purpose built for ease-of-use, scaling out, and optimizing DevOps needs.
- Provides critical performance management capabilities and monitoring of OneDB data store servers.
- The monitoring system feeds directly into a customizable alerting system so alerts can be immediately sent via email, Twilio, or PagerDuty.
- User and permission management for restricted access to dashboard of certain servers or group of servers.

The default login credentials for HCL OneDB Explore are:

- Username: admin
- Password: testPassw0rd

To override the admin password use a values override file and provide that at install time.

File: myvalues.yaml

```
onedb-explore:
    adminPassword: newPassw0rd
```

```
helm install expl-v1  -f myvalues.yaml  production-onedb-explore
```

Install OneDB Product (onedb-product)

The OneDB Product helm chart will install the HCL OneDB database. This helm chart will include as subcharts the other OneDB helm charts:

- OneDB SQL Data Store
- OneDB Mongo Data Store
- OneDB RESTful Data Store
- OneDB Explore

This chart is an all-inclusive chart that includes all the OneDB charts for full functionality.

HCL OneDB is an enterprise grade database for storing and processing the following types of data:

- Relational (Table based)
- Document (Json Based)

- Timeseries (Time based)
- Spatial (Coordinate based)

Access to OneDB Product is through:

- The native SQLI protocol. HCL OneDB provides drivers for different programming languages to provide this connectivity. Ex. Java, Python, NodeJS, ODBC
- The REST API. This allows you to use language of choice that supports RESTful services.
- The MongoDB protocol. This allows you to use any language that supports a MongoDB driver.

OneDB Explore is included as a UI to interact with and administer the OneDB Database server(s).

After all system requirements and prerequisites have been addressed you are ready to install the Sofy helm chart. The command below is a sample install that uses a file named `myvales.yaml` to provide any overrides to the helm chart values and `anchor.yaml` to set the anchor service's resources.

Following file overrides multiple parameters and is provided an installation time:

File: myvalues.yaml

```
hclFlexnetURL:  flex-net-xxxxx
hclFlexnetID: xxxxxx

anchor:
  resources:
    cpu: 250m

onedb-product:
  onedb-explore:
    adminPassword: newPassw0rd
```

```
helm install prod-v1  -f myvalues.yaml  production-onedb-product
```

Pod Scheduling

As a best practice when deploying OneDB SQL Data store into kubernetes in production isolate the OneDB Database server pods to a specific set of nodes. Also, make sure no two database server pods are scheduled on the save node.

OneDB Server pod scheduling is controlled with the helm chart parameters:

- onedb.nodeSelectorRequired
- onedb.nodeSelector
- onedb.tolerations

To understand how OneDB handles pod scheduling it is import to understand a few concepts.

- Assigning Pods to Nodes (Affinity / Anti-affinity)
- Taints and Tolerations

For more information on Pod scheduling, see kubernetes https://kubernetes.io/.

Assigning Pods to Nodes (Affinity/ Anti-Affinity)

Node Affinity allows you to constrain which nodes your pods are eligible to be scheduled on based on labels that have been defined for the nodes. There are two types of node affinity that are used with OneDB.

- requiredDuringSchedulingIgnoredDuringExecution
- preferredDuringSchedulingIgnoredDuringExecution

requiredDuringSchedulingIgnoredDuringExecution: This is a hard requirement in that the pod “must” be scheduled on the node with the defined set of rules.

preferredDuringSchedulingIgnonredDuringExecution: This is a soft requirement in that the pod will prefer or try to schedule on nodes with the defined rules but it is not guaranteed.

Note: OneDB uses these rules to force a pod to be scheduled on a specific set of nodes or prefer to be scheduled on a specific set of nodes.

Pod anti-affinity allows you to constrain which nodes your pod is eligible to be scheduled on based on pods that are already running on the node. As with node affinity OneDB uses two types of pod anti-affinity.

- requiredDuringSchedulingIgnoredDuringExecution
- preferredDuringSchedulingIgnoredDuringExecution

requiredDuringSchedulingIgnoredDuringExecution: This is a hard requirement in that the pod “must” be scheduled on the node with the defined set of rules.

preferredDuringSchedulingIgnonredDuringExecution: This is a soft requirement in that the pod will prefer or try to schedule on nodes with the defined rules but it is not guaranteed.

Note: OneDB uses these rules to force a OneDB pod to not schedule on nodes already running a OneDB pod or prefer to not be scheduled on that same node.

Labeling Nodes

Your kubernetes administrator will perform this task. They can label a single node or a group of nodes (node pool) with a specific designation with a key/value pair. This is needed to use affinity/anti-affinity capabilities with kubernetes.

To label a node the following command is used:

```
kubectl label nodes <nodename> key=value –overwrite
```

The key/value pair that is defined here is arbitrary. It is a key/value pair that would then be used with helm chart parameter overrides to specify the affinity/anti-affinity.

Example with an arbitrary key/value pair of type=database looks like this:

```
kubectl label nodes gke-worker4 type=database -overwrite
```

Configure OneDB Affinity/Anti-Affinity

We have two helm chart parameters that can be set with OneDB SQL Data store. The OneDB SQL Data store uses these helm chart parameters for both the onedb and onedbcm sections of the helm chart.

```
onedb:
    nodeSelectorRequired: true
    nodeSelector:
        type: database
. . .
onedbcm:
    nodeSelectorRequired: true
    nodeSelector:
        type: cm
```

The default values for onedb/onedbcm **nodeSelectorRequired** is **true.** When this is set to true the requiredDuringSchedulingIgnoredDuringExecution is used for Pod anti-affinity.

The effect of this is that, a OneDB Database server will not be scheduled on the same node where another OneDB Database server pod is running. And a OneDB Connection manager will not be scheduled on the same node where another OneDB Connection manager pod is running.

When we set the **nodeSelector** helm chart parameter for either onedb or onedbcm OneDB will use requiredDuringSchedulingIgnoredDuringExecution and Node affinity is enabled. This will require that all Pods be scheduled on nodes that have been labeled accordingly.

Example Labeling of Nodes:

```
kubectl label nodes gke-worker2 type=database -overwrite
kubectl label nodes gke-worker4 type=database -overwrite

kubectl label nodes gke-worker3 type=cm -overwrite
kubectl label nodes gke-worker5 type=cm -overwrite
```

With the above helm chart values set the OneDB Database server pods must run on a kubernetes nodes that are labled with **type:database**, and OneDB Connection manager pods must run on kubernetes nodes that are labeled with **type:cm**.

OneDB SQL Data store sets up an HA cluster with an HDR primary and HDR secondary. If nodeSelectorRequired is set to true, then we must have more than 1 node labeled when use nodeSelector. The same applies to the OneDB Connection manager based on how many replicas are running.

Note: When configuring pod scheduling it is important to have a good understanding of how this works or you may run into a situation where a pod is not able to be scheduled.

Taints and Toleration

While Node affinity is a property of a pod that attracts them to a set of nodes either as a preference or hard requirement. Taints are the opposite, in that the allow a node to repel a set of pods. A taint is defined on a pod.

```
kubectl taint nodes gke-worker2 type=onedb:NoSchedule
```

This uses a key/value pair in this example we used **type=onedb**, with the NoSchedule effect. This means that no pod will be able to schedule onto the node (gke-worker2) unless it has a matching toleration.

You would then need to use the **tolerations** helm chart parameter override and set the following:

```
tolerations:
-  key: “type”
   operator: “Exists”
   effect: “NoSchedule”
```

Note: Using a combination of Affinity/Anti-Affinity and taints and tolerations, you can control what nodes OneDB SQL Data store will be schedule on and dictate that those nodes are only used for OneDB.

OneDB Configuration

There are five helm charts for OneDB.

- **OneDB SQL Data Store (onedb-sql):** This is the Helm chart that contains the OneDB Database server. It uses a kubernetes operator to deploy a statefulset in a kubernetes environment.
- **OneDB Rest Data Store (onedb-rest):** This is a helm chart that deploys the OneDB Database server with the REST listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Document Data Store (onedb-mongo):** This is a helm chart that deploys the OneDB Database server with the Mongo listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Explore (onedb-explore):** This is a Helm chart that contains only the OneDB Explore UI.
- **OneDB Product (onedb-product):** This is a Helm chart that contains OneDB SQL Data Store, OneDB Rest, OneDB Document and OneDB Explore all as subcharts.

Each chart has a list of configuration options that can be set to specify how the OneDB Helm chart will be installed, setup and configured.

OneDB SQL Data Store Configuration

To customize the installation and configurations, see the list of configuration parameters available to the OneDB SQL Data Store.

List of OneDB SQL Data Store Configuration Parameters

Parameter	*Description*	*Value*
global.hclImagePullSecret	Your own secret with your credentials to HCL's Docker repository. Required when deploying solution in your own cluster.	''
hclFlexnetURL	Your HCL FlexNet license server URL for your HCL entitlements. Required when deploying in your own cluster	''
hclFlexnetID	Your HCL FlexNet license ID fo ryour HCL entitlements. Required when deploying solution in your own cluster.	''
isOpenShift	Set to true if using Openshift	false
nfsserver.volumeSize	Size of the Volume used for backups and other shared files.	50G
nfsserver.googleFilestore.enable	Set to true to enable Google Filestore	false
nfsserver.googleFilestore.filestoreIP	IP address of the Filestore instance	''
nfsserver.googleFilestore.filestoreShare	Name of the File share on the instance	''
nfsserver.awsEFS.enable	Set to true to enable AWS Elastic filestore	false
nfsserver.awsEFS.EFSServer	DNS name of the file system	fs-XXXXX.efs-us-west-2.amazonaws.com
nfsserver.awsEFS.EFSPath	NFS Mount path	/
nfsserver.azureFS.enable	Set to true to enable Azure File share	false
nfsserver.azureFS.secretName	Kubernetes secret name to use	''
nfsserver.azureFS.shareName	Azure file share name	''
nfsserver.other.enable	Set to true to enable	true
nfsserver.other.storageClass	Set this to the storageClass of the NFS	nfs

Parameter	*Description*	*Value*
tls.config	Set to true to enable TLS communication	false
tls.tlskey	Base64 value of server private key	tlskey: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS ...
tls.tlscert	Base64 value of signed server certificate	tlscert: LS0tLS1CRUdJTiBDRVJUSEDFRQ0FURS ...
tls.tlscacert	Base64 value of certificate authority root certificate	tlscacert: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS ...
autoscaling.enabled	Set to true to enable auto scaling. To use auto scaling make sure to set onedb.serverReplicaCount, onedb.maxReplicaCount and onedb.resources appropriately.	false
autoscaling.targetCPUUtilizationPercentage	Set to the Percentage when autoscaling should occur.	70
onedb.serverReplicaCount	Set to the number of servers to start for an HA cluster	2
onedb.maxReplicaCount	Set to the max value of servers you would want to configure in the HA cluster	10
onedb.dbsapwd	OneDB Server DBSA (onedbsa) user password	onedb4ever
onedb.backupTag	Set to unique value in case the backup device is shared with other OneDB HA Cluster	onedbbackup-myuniquetag
onedb.encryptionAtRest	Set to true to enable Encryption at rest	false
onedb.exploreAgent	Set to true to start the OneDB Explore Agent on each server	false
onedb.dataStorageClass	Set to the cloud vendor default storage class. Low latency disk I/O storage is recommended. For GKE “standard” is the default	""
onedb.dataStorageSize	Set the persistent Volume Size	10gi
onedb.dataStorageCount	Number of persistent volume’s to provision	2

Parameter	*Description*	*Value*
onedb.restoreFromBackup	Set to true when a restore from last backup is needed	false
onedb.restoreTimestamp	set to specific point in time to perform a point in time restore. Ex 2021-05-11 11:35:00	''
onedb.nodeSelectorRequired	Set to true to enforce that no two OneDB Server pods are scheduled on the same K8s node	true
onedb.nodeSelector	Set to a node label to schedule OneDB server pods on preconfigured set of K8s nodes. Set your own keyvalue pair for the node selector	''
onedb.tolerations	Used to assist the K8s schedule	{}
onedb.resources	Resources like cpu and memory requested from the cluster.	{}
onedb.customServerEnv	Allows you to set additional environment variables to be used by the database server.	''
onedb.customConfig	Allows you to set ONCONFIG parameters to be used by the Database server.	MULTIPROCESSOR: 1
onedb.customSpace	Allows you to create custom Dbspaces in the database server	''
onedb.appUsers	Allows you to create custom Users in the database server	''
onedb.customInitSQL	Allows you to create a script of SQL statements that is run after server initialization.	''
onedb.customInitImage	Allows you to create an Init Container image	''
onedb.customInitImageCmd	The command to run from the custom Init container	''
onedb.groupName	Unique name for each cluster if Setting up Enterprise Replication	g_cdr1
onedb.groupID	Unique ID for each cluster if setting up Enterprise Replication	1
onedbcmReplicaCount	Number of Connection Managers to start for the HA cluster	2
onedbcm.serviceType	Set the service type of the connection manager. (ClusterIP, LoadBalancer or NodePort)	ClusterIP
onedbcm.sla_policy	Set the service level agreement of the connection manager. (ROUNDROBIN, WORKLOAD)	ROUNDROBIN
onedbcm.autofailover	If set to false then autofailover is disabled	true

Parameter	*Description*	*Value*
onedbcm.nodeSelectorRequired	Set to true to enforce that no two OneDB CM pods are scheduled on the same K8s node.	true
onedbcm.nodeSelector	Set to a node label to schedule OneDB pods on preconfigured set of K8s nodes. Set your own keyvalue pair.	''
onedbcm.tolerations	Used to assist the K8s schedule	{}
onedbcm.resources	Resources like cpu and memory, requested from the cluster	{}

Customize Server configuration

The ONCONFIG file is used by the database server during initialization to setup the data store server. Use the ***customConfig*** helm chart configuration parameter to specify ONCONFIG parameters. It can be configured as follows. With parameters that are not unique specify a number after the parameter as seen below with BUFFERPOOL# .

```
onedb:
    customconfig:
          MULTIPROCESSOR: “1”
          BUFFERPOOL1: “size=8k,buffers=50000,lrus=8,lru_min_dirty=50,lru_max_dirty=60.5”
          BUFFERPOOL2: “size=2k,buffers=200000,lrus=8,lru_min_dirty=50,lru_max_dirty=60.5”
          LOGSIZE: “10000”
```

Create Initialization SQL script

The helm chart configuration parameter ***customInitSQL*** can be used to create an SQL scrpit that will run by the OneDB server after first initialization. This script can be used to perform needed setup tasks, creation of databases, etc.

```
onedb:
    customInitSQL:  |-
        database sysadmin;
        create database test with log;
        create table t1 (col1 int, col2 int);
```

Creating custom spaces

The helm chart configuration parameter ***customSpace*** can be used to create and setup spaces. Following table details the options available for the creation of spaces. When defining the customSpace parameter, you must create a well formed json document.

Parameter	***Description***	***Example Value***
name	The name of the space	my_data_dbspace
type	The type of space to create. Supported values are: dbspace: normal dbspace llog : logical log dbspace plog: physical log dbspace sbspace: smart blobspace tempdbspace: temporary dbspace tempsbspace: temporary smart blobspace	dbspace
pagesize	The size of the space, supported values are 2k,4k,6k,8k,16k	4k
size	Size of the space, supported values are GB, MB, KB	10GB
logging	Used for smart blobspaces to enable loging 1: enable logging 0: disable logging	1

```
onedb:
    customSpace: >-
        [
            {“name”:”datadbs”, “type”: “dbspace”, “pagesize”: “4k”, “size”: “4GB” },
            {“name”:”logdbs”, “type”: “llog”, “size”: “2GB” },
            {“name”:”plogdbs”, “type”: “plog”, “size”:  “4GB” },
            {“name”:”sbspace1”, “type”: “sbspace”, “size”: “1GB” , “logging”: 1},
            {“name”:”tmpdbspace1”, “type”: “tempdbspace”,”pagesize”: “4k”, “size”: “1GB” },
            {“name”:”tmpsbsp1”, “type”: “tempsbspace”, “size”: “500MB” }
        ]
```

Creating custom users

The helm chart configuration parameter ***appUsers*** can be used to create additional users. Following table details the options available for the creation of users. Currently, the only type of user support is an operating system user account. When using appUsers, you must create a well formed json document.

Parameter	***Description***	***Example Value***
user	The name of the user	appuser1
password	The password of the user	passw0rd
group	A group name to create for the user.	dev

Parameter	***Description***	***Example Value***
uid	The user id number to use for the user	1003
gid	The group id number to to use for the group	2000
type	The type of user to create. Currently only osuser is supported	osuser

```
onedb:
    appUsers: >-
        [
            { “user”:”appuser1”, “password”: “passw0rd”, “group”:”dev”,
 “uid”:1003,”gid”:2000,”type”:”osuser” },
            { “user”:”appuser2”, “password”: “passw0rd”, “group”:”dev”,
 “uid”:1003,”gid”:2000,”type”:”osuser” }
        ]
```

Setting additional server Environment

The helm chart configuration parameter ***customServerEnv*** can be used to set additional server environment variables. This will be set in the environment script when initialization and starting the OneDB database server.

```
onedb:
    customServerEnv:
        DB_LOCALE: “en_us.utf8”
        DBTEMP: “/tmp”
```

Using an Init container

The helm chart configuration parameters ***customInitImage*** and ***customInitImageCmd*** can be used to create an Init container to perform setup steps prior to the startup of the OneDB server container image. The customInitimage parameter is used to specify an image to use and the customInitImageCmd is the command to run inside the image.

The Init container image can be a purposely built image with scripts built in. Or it can be a generic image with specific OS commands to run.

```
onedb:
    customInitImage: “gcr.io/<my-images>/busybox-custom:latest”
    customInitImageCmd: “/bin/initSetup.sh”
```

Scheduling of K8s pods

The helm chart configuration parameter ***nodeSelector*** for onedb and onedbcm are used to support Node affinity. It allows you to select a preconfigured set of K8s nodes to run on.

The following example will run the OneDB server on nodes labeled as onedb and the OneDB Connection manager on nodes labeled as onedbcm.

```
onedb:
    nodeSelector:
        database: onedb
onedbcm:
    nodeSelector:
        cm: onedbcm
```

The helm charts have an unconfigured parameter ***tolerations*** to allow for full configuration of taints and tolerations for K8s scheduling of pods. This can be used to specify a node taint, which means no pod can be scheduled on the node unless it has a matching toleration. Then a OneDB server is labeled with a toleration to allow it to run on the tainted nodes.

```
kubectl taint nodes node1 tainted4onedb=onedb-only:NoSchedule
```

```
onedb:
    tolerations:
    - key: “tainted4onedb”
      operator: “Exists”
      effect: “NoSchedule”
```

Sample helm override file

When specifying helm chart parameters, you can specify them on the command line. When specifying a number of parameters it is sometimes more convenient to create a file with the override parameters. The following example shows a single file that uses customServerEnv, appUsers and customInitSQL in a single file.

```
FILE:  onedb.override.yaml
onedb:
    customServerEnv:
        ONEDB_USER_MANAGEMENT: “true”
        ONEDB_USER: “user1”

    appUsers: >-
        {“user”:”user1”, “password”:”Passw0rd”, “group”:”dba”, “uid”: 1005, “gid”:2001,
         “type”:”osuser”}

    customInitSQL:  |-
        create database stores with log;
        create user dbauser with password ‘Passw0rd’ account unlock properties user ‘user1’
        authorization(dbsa);
```

The above yaml file sets two environment variable that are used in the container Image to enable database users. It then creates one os user to be used as the operating system user that the created database user will have permissions as.

OneDB REST Data Store Configuration

To customize the installation and configurations see the list of configuration parameters available to the OneDB REST Data Store.

List of OneDB REST Data Store Configuration Parameters

Parameter	***Description***	***Value***
global.hclImagePullSecret	Your own secret with your credentials to HCL's Docker repository. Required when deploying solution in your own cluster.	''
hclFlexnetURL	Your HCL FlexNet license server URL for your HCL entitlements. Required when deploying in your own cluster	''
hclFlexnetID	Your HCL FlexNet license ID for your HCL entitlements. Required when deploying solution in your own cluster.	''
resources	Resources like cpu and memory, requested from the cluster	requests.cpu: "100m", requests.memory: "128mi"
config	Setting advanced options in the application's yaml configuration file	''
externalDBUrl	A custom external JDBC style URL if you want to connect to a OneDB server that is not part of the solution	''
databaseuser	The user the REST API will use to connect to the OneDB Database Server	onedbsa
databasePassword	The password the REST API will use to connect to the OneDB Database Server	onedb4ever

Custom REST Configuration

Additional configuration can be added to the REST service as follows. Review the product documentation for all available options for the REST configuration file.

```
onedb-rest:
    config: |-
        rest.session.timeout 600000
        security.csrf.token.enable: true
```

OneDB Document Data Store Configuration

To customize the installation and configurations see the list of configuration parameters available to the OneDB Document Data Store.

List of OneDB Document Data Store Configuration Parameters

Parameter	*Description*	*Value*
global.hclImagePullSecret	Your own secret with your credentials to HCL's Docker repository. Required when deploying solution in your own cluster.	''
hclFlexnetURL	Your HCL FlexNet license server URL for your HCL entitlements. Required when deploying in your own cluster	''
hclFlexnetID	Your HCL FlexNet license ID for your HCL entitlements. Required when deploying solution in your own cluster.	''
resources	Resources like cpu and memory, requested from the cluster	requests.cpu: "100m", requests.memory: "128mi"
config	Setting advanced options in the application's yaml configuration file	''
externalDBUrl	A custom external JDBC style URL if you want to connect to a OneDB server that is not part of the solution	''
mongoUser	The mongo username	mongo
mongoPassword	Password for the mongo user	mongoPassword
databaseuser	The user the MongoDB API will use to connect to the OneDB Database Server	onedbsa
databasePassword	The password the MongoDB API will use to connect to the OneDB Database Server	onedb4ever

Custom Mongo Configuration

Additional configuration can be added to the Document Data Store (Mongo) service as follows. Review the product documentation for all available options for the Mongo configuration file.

```
onedb-mongo:
    config:  |-
        security.sql.passthrough=true
```

OneDB Explore Data Configuration

To customize the installation and configurations see the list of configuration parameters available to OneDB Explore Data.

List of OneDB Explore Configuration Parameters

Parameter	*Description*	*Value*

global.hclImagePullSecret	Your own secret with your credentials to HCL's Docker repository. Required when deploying solution in your own cluster.	''
resources	Resources like cpu and memory, requested from the cluster	requests.cpu: "100m", requests.memory: "128mi"
config	Setting advanced options in the application's yaml configuration file	''
adminPassword	Initial admin password	testPassw0rd

Custom Explore Configuration

Additional configuration can be added to the Explore service as follows. Review the product documentation for all available options for the Explore configuration file.

```
onedb-explore:
    config:  |-
        key=value
```

Configuring TLS

Use transport layer security (TLS) to create secure connections from OneDB clients to the OneDB database server. By default, TLS is disabled. To enable TLS connections, set the **tls.tlsconfig** helm chart parameter value to **true**.

The following helm chart parameters also need to be set:

- **tlskey**: The base64 encoded value of the private key.
- **tlscert**: The base64 encoded value for the Public signed server certificate.
- **tlscacert**: The base64 encoded value of the certificate authority root certificate.

Example tls configuration:

```
tls:
    tlsconfig: true
    tlskey:  LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t....
    tlscert: LS0tLS1CRUdJTiBABEFUSUZJQ0FURS0tLS0t...
    tlscacert: LS0tLS1CRUdJTiBDRVJUSUEEFZFURS0tLS0t....
```

Create TLS Certificates

About this task

You can obtain your own certificates from a certificate authority or you can create your own with the following steps using openssl:

1. Generate root CA private key PEM file:

```
openssl genrsa -out rootCA.key.pem
```

2. Create a self signed root CA certificate in PEM file:

```
openssl req -new -x509 -key rootCA.key.pem -subj "/C=US/ST=Kansas/L=Olathe/O=HCL/OU=OneDB" -days 3650
 -out
 rootCA.cert.pem
```

3. Generate server private key:

```
openssl genrsa -out server.key.pem
```

4. Generate a certificate signing request (CSR) for OneDB Server:

```
openssl req -new -key server.key.pem -subj
 "/C=US/ST=Kansas/L=Olathe/O=HCL/OU=OneDB/CN=Server/emailAddress=onedb@hcl.com" -out server.req.pem
```

5. Sign certificate with root CA:

```
openssl x509 -req -inform PEM -in server.req.pem -set_serial 1 -CA
rootCA.cert.pem -CAkey rootCA.key.pem -days 3650 -extensions usr_cert -outform PEM -out server.cert.pem
```

6. Convert rootCA.cert.pem to base64 -> tlscacert:

```
base64 rootCA.cert.pem -w 0 > tlscacert
```

7. Convert server.cert.pem to base64 -> tlscert:

```
base64 server.cert.pem -w 0 > tlscert
```

8. Convert server.key.pem to base64 -> tlskey:

```
base64 server.key.pem -w 0 > tlskey
```

Connect from Java client with TLS

About this task

To connect to the OneDB Databaser server with a Java client (JDBC) with TLS you must create a keystore for the client application to use. You need the root CA certificate and will use this file **rootCA.cert.pem** to generate the kesytore.

Create the keystore:

```
keytool -import -file rootCA.cert.pem -keystore ssl.keystore
```

Example

Example OneDB JDBC URL to connect to a OneDB Database server using TLS:

```
jdbc:onedb://XX.XXX.XXX.XX.nip.io:10001/sysmaster;user=onedbsa;password=xxxxxxx;ENCRYPT=true;TRUSTSTORE=./ssl.k
eystore;TRUSTSTOREPASSWORD=xxxxxxx;
CERTIFICATEVERIFICATION=false;loginTimeout=0
```

For more information on connecting JDBC applications with TLS, see HCL OneDB JDBC Driver Guide.

Accessing OneDB

Connecting to the OneDB database server is essential. OneDB allows for connections from Mongo Clients, REST Clients and Native SQLI clients. Ex. JDBC, ODBC, ESQL/C

Connectivity can occur from inside the cluster or from outside the cluster. By default, connections from outside the cluster are not enabled.

Connectivity will be different based on the installation. The two basic installations are:

1. Installation of the Standalone Helm Chart of OneDB
2. Installation of a Solution Factory Helm Chart of OneDB

The OneDB Connection Manager is used for Native SQLI Connections to the Database server. There is a Kubernetes service provided to handle this connectivity.

REST, Mongo and OneDB Explore will each have a Kubernetes service to handle their own connections.

```
   NAME                      TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)               AGE
helm-1-odbp-explore           ClusterIP      10.96.220.30       <none>       8080/TCP              20m
helm-1-odbp-mongo             ClusterIP      10.96.44.208       <none>       27017/TCP             20m
helm-1-odbp-rest              ClusterIP      10.96.90.21        <none>       8080/TCP              20m
onedbcm-cm-service            ClusterIP      10.96.159.103      <none>       10000/TCP,20000/TCP   19m
```

With the OneDB Connection Manager you will see a pattern emerge that will describe what type of connection will occur.

Port Number	Description
10XXX	Internal (Redirected) Connection
20XXX	External (Proxied) Connection
XXXX1	Connection to the HA Primary Server
XXXX2	Connection to the HA Secondary Server
XXXX3	Connection to any server in the HA Cluster
XXX2X	Connection that uses SSL

Example: Port 10021 is an internal Connection (10XXX) to the HA Secondary server (XXXX1) using SSL (XXX2X).

Standalone OneDB Chart

A standalone OneDB helm chart does not include elements of a Solution Factory helm chart. There are mutliple different Standalone helm charts.

- **OneDB SQL Data Store (onedb-sql):** This is the Helm chart that contains the OneDB Database server. It uses a kubernetes operator to deploy a statefulset in a kubernetes environment.
- **OneDB Rest Data Store (onedb-rest):** This is a helm chart that deploys the OneDB Database server with the REST listener. The OneDB SQL Data Store is a subchart in this chart.

- **OneDB Document Data Store (onedb-mongo):** This is a helm chart that deploys the OneDB Database server with the Mongo listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Explore (onedb-explore):** This is a Helm chart that contains only the OneDB Explore UI.
- **OneDB Product (onedb-product):** This is a Helm chart that contains OneDB SQL Data Store, OneDB Rest, OneDB Document and OneDB Explore all as subcharts.

The OneDB helm(s) chart can be configured to only allow connections from within the cluster itself, or they can be configured to allow for connections from outside the cluster.

To allow for connections from outside the cluster the kubernetes service types should be configured as Loadbalancer, or an extra piece of software can be used to provide a single point of ingress into the cluster. Some commonly used ingress/loadbalancer's are Ambassodor, NGINX.

Setting up Ambassador or NGINX is outside the scope of this documentation, instead we will use a Loadbalancer service type.

Connecting from Inside the cluster

OneDB Native SQLI connection are accessible through the OneDB Connection Manager. A kubernetes service will be created with the deployment and that service name is used for connections. A kubernetes service is created for the non-SQLI types as well.

```
kubectl get services

    NAME                      TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)               AGE
helm-1-odbp-explore        ClusterIP         10.96.220.30      <none>       8080/TCP               20m
helm-1-odbp-mongo          ClusterIP         10.96.44.208      <none>       27017/TCP              20m
helm-1-odbp-rest           ClusterIP         10.96.90.21       <none>       8080/TCP               20m
onedbcm-cm-service         ClusterIP         10.96.159.103     <none>       10000/TCP,20000/TCP    19m
```

The OneDB Connection manager supports "Redirected" and "Proxied" connections. For internal connections it is recommended to use "Redirected" connections. The following table shows the Internal connection string to use for each driver type. From within the cluster the **.{namespace}.svc.cluster.local** may not be needed from the URL below.

Driver	URL	Example URL
OneDB driver (SQLI-Primary)	onedbcm-cm-service.{namespace}.svc.cluster.local:10000	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary)	onedbcm-cm-service.{namespace}.svc.cluster.local:10001	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any)	onedbcm-cm-service.{namespace}.svc.cluster.local:10002	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Primary-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10020	jdbc:onedb://{url}/sysmaster

OneDB driver (SQLI-Secondary-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10021	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10022	jdbc:onedb://{url}/sysmaster
Mongo combatible driver	<Release-Name>-odbp-mongo:27017	mongodb://<release-name>-odbp-mongo:27017
REST	<Release-Name>-odbp-rest:8080	http://<release-name>-odbp-rest:8080
Explore	<Release-Name>-odbp-explore:8080	http://<release-name>-odbp-explore:8080

Connecting from Outside the cluster

OneDB Native SQLI connection are accessible through the OneDB Connection Manager. A kubernetes service will be created with the deployment and that service name is used for connections. A kubernetes service is created for the non-SQLI types as well.

```
kubectl get services

NAME                       TYPE           CLUSTER-IP       EXTERNAL-IP            PORT(S)
 AGE
helm-1-odbp-explore        LoadBalancer   10.96.220.30     172.19.255.202         8080/TCP            20m
helm-1-odbp-mongo          LoadBalancer   10.96.44.208     172.19.255.201         27017/TCP           20m
helm-1-odbp-rest           LoadBalancer   10.96.90.21      172.19.255.200         8080/TCP            20m
onedbcm-cm-service         LoadBalancer   10.96.159.103    172.19.255.203         10000/TCP,20000/TCP     19m
```

The OneDB Connection manager supports “Redirected” and “Proxied” connections. For external connections you must use the “Proxied” connections. The following table shows the external connection string to use for each driver type.

Driver	URL	Example URL
OneDB driver (SQLI-Primary)	{LoadBalancer External IP address}:20000	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary)	{LoadBalancer External IP address}:20001	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any)	{LoadBalancer External IP address}:20002	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Primary-SSL)	{LoadBalancer External IP address}:20020	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary-SSL)	{LoadBalancer External IP address}:20021	jdbc:onedb://{url}/sysmaster

OneDB driver (SQLI-Any-SSL)	{LoadBalancer External IP address}:20022	jdbc:onedb://{url}/sysmaster
Mongo combatible driver	{LoadBalancer External IP address}:27017	mongodb://{url}:27017
REST	{LoadBalancer External IP address}:8080	http://{url}:8080
Explore	{LoadBalancer External IP address}:8080	http://{url}:8080

Setting LoadBalancer Type

The default setting for each service type is ClusterIP. This will only allow internal connections. The OneDB helm chart service types of interest are listed below. NOTE: The names of the services may be slightly different when installing onedb-mongo, onedb-rest, onedb-sql chart.

- onedbcm-cm-service
- <Release.Name>-odbp-explore
- <Release.Name>-odbp-mongo
- <Release.Name>-odbp-rest

Each of these service types can be configured as a LoadBalancer to provide external connectivity.

To set Loadbalancer for the Connection Manager:

```
onedb-sql:
    onedbcm:
        serviceType: LoadBalancer
```

To set LoadBalancer for Mongo

```
onedb-mongo:
    service:
        type: LoadBalancer
```

To set LoadBalancer for REST

```
onedb-rest:
   service:
      type: LoadBalancer
```

To set LoadBalancer for Explore

```
onedb-explore:
      service:
         type: LoadBalancer
```

Solution Factory OneDB Chart

A Solution factory OneDB helm chart contains elements of the Solution factory including things like a Console UI, grafana, prometheus. The OneDB helm chart is included as a subchart of the overall Helm chart. There are multiple different Solution Factory charts that include different aspects of OneDB.

- **OneDB SQL Data Store (onedb-sql):** This is the Helm chart that contains the OneDB Database server. It uses a kubernetes operator to deploy a statefulset in a kubernetes environment.
- **OneDB Rest Data Store (onedb-rest):** This is a helm chart that deploys the OneDB Database server with the REST listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Document Data Store (onedb-mongo):** This is a helm chart that deploys the OneDB Database server with the Mongo listener. The OneDB SQL Data Store is a subchart in this chart.
- **OneDB Explore (onedb-explore):** This is a Helm chart that contains only the OneDB Explore UI.
- **OneDB Product (onedb-product):** This is a Helm chart that contains OneDB SQL Data Store, OneDB Rest, OneDB Document and OneDB Explore all as subcharts.

The Solution Factory OneDB helm(s) chart can be configured to only allow connections from within the cluster itself, or they can be configured to allow for connections from outside the cluster.

To allow for connections from outside the cluster, a Solution factory OneDB helm chart includes and configures Ambassador. The ambassador LoadBalancer will handle connectivity into the kubernetes cluster for Mongo, REST and Explore. The OneDB Connection Manager will handle connections into the kubernetes cluster for Native SQLI clients (ex. JDBC, ODBC, ESQL/C)

By default, OneDB Connection Manager does not allow for external connections.

Connecting from Inside the cluster

OneDB Native SQLI connection are accessible through the OneDB Connection Manager. A kubernetes service will be created with the deployment and that service name is used for connections. The ambassador service is created for the non-SQLI connections. The ambassador service is setup as a Loadbalancer type where as the OneDB Connection Manager has a default setting of ClusterIP.

```
kubectl get services

  NAME                  TYPE               CLUSTER-IP         EXTERNAL-IP            PORT(S)
 AGE
sofy-1-ambassador       LoadBalancer         10.96.224.175     172.19.255.200         80:32653/TCP,44
   68m
sofy-1-odbp-mongo       ClusterIP            10.96.183.135     <none>                 27017/TCP
  69m
sofy-1-odbp-rest        ClusterIP            10.96.173.88      <none>                 8080/TCP
   69m
sofy-1-odbp-explore     ClusterIP            10.96.66.46       <none>                 8080/TCP
  71m
onedbcm-cm-service      ClusterIP            10.96.33.166      <none>                 10000:30248/TCP
  77m
```

The OneDB Connection manager supports “Redirected” and “Proxied” connections. For internal connections it is recommended to use “Redirected” connections. The following table shows the Internal connection string to use for each driver type. From within the cluster the **.{namespace}.svc.cluster.local** may not be needed from the URL below:

Driver	URL	Example URL
OneDB driver (SQLI-Primary)	onedbcm-cm-service.{namespace}.svc.cluster.local:10000	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary)	onedbcm-cm-service.{namespace}.svc.cluster.local:10001	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any)	onedbcm-cm-service.{namespace}.svc.cluster.local:10002	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Primary-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10020	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10021	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any-SSL)	onedbcm-cm-service.{namespace}.svc.cluster.local:10022	jdbc:onedb://{url}/sysmaster
Mongo combatible driver	<Release-Name>-odbp-mongo:27017	mongodb://<release-name>-odbp-mongo:27017
REST	<Release-Name>-odbp-rest:8080	http://<release-name>-odbp-rest:8080
Explore	<Release-Name>-odbp-explore:8080	http://<release-name>-odbp-explore:8080

Connecting from Outside the cluster

OneDB Native SQLI connection are accessible through the OneDB Connection Manager. A kubernetes service will be created with the deployment and that service name is used for connections. The ambassador service is created for the non-SQLI connections. The ambassador service is setup as a Loadbalancer type where as the OneDB Connection Manager has a default setting of ClusterIP. To allow external SQLI connectivity you must set the service type to LoadBalancer.

```
kubectl get services

 NAME                  TYPE           CLUSTER-IP       EXTERNAL-IP          PORT(S)           AGE
sofy-1-ambassador     LoadBalancer   10.96.224.175    172.19.255.200       80:32653/TCP,44     68m
sofy-1-odbp-mongo     ClusterIP      10.96.183.135    <none>               27017/TCP           69m
sofy-1-odbp-rest      ClusterIP      10.96.173.88     <none>               8080/TCP            69m
sofy-1-odbp-explore   ClusterIP      10.96.66.46      <none>               8080/TCP            71m
onedbcm-cm-service    LoadBalancer   10.96.33.166     172.19.255.201       10000:30248/TCP     77m
```

The OneDB Connection manager supports “Redirected” and “Proxied” connections. For external connections you must use the “Proxied” connections. The following table shows the external connection string to use for each driver type.

Driver	URL	Example URL
OneDB driver (SQLI-Primary)	{LoadBalancer External IP address}:20000	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary)	{LoadBalancer External IP address}:20001	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any)	{LoadBalancer External IP address}:20002	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Primary-SSL)	{LoadBalancer External IP address}:20020	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Secondary-SSL)	{LoadBalancer External IP address}:20021	jdbc:onedb://{url}/sysmaster
OneDB driver (SQLI-Any-SSL)	{LoadBalancer External IP address}:20022	jdbc:onedb://{url}/sysmaster
Mongo combatible driver	{LoadBalancer External IP address}:27017	mongodb://{url}:27017
REST	{LoadBalancer External IP address}:8080	http://{url}:8080
Explore	{LoadBalancer External IP address}:8080	http://{url}:8080

Setting LoadBalancer Type

The default setting for the OneDB Connection Manager service type is ClusterIP. This will only allow internal connections. To allow for connectivity from outside the cluster you must set the service type for the OneDB Connection Manager to LoadBalancer.

To set Loadbalancer for the Connection Manager:

```
onedb-sql:
    onedbcm:
        serviceType: Loadbalancer
```

Connection credentials

The following table shows the default connection credentials. This can be changed accordingly.

Product/Driver	User	Password
Explore	admin	testPassw0rd
REST	-	-

Mongo	mongo	mongoPassword
SQL Data Store	onedbsa	onedb4ever

The REST connection uses a database connection. You can connect with onedbsa or any other use that was created in the OneDB SQL Data Store.

To change the onedbsa password for the OneDB SQL Data Store use the following configuration override values. If you change this password it is important that you make changes to mongo, rest and explore.

```
onedb-sql:
    onedb:
        dbsapwd: one1dba4ever
```

If you change the passowrd for OneDB SQL Datastore (onedb-sql), you must tell the mongo,rest and explore charts how to connect to the OneDB SQL Datastore (onedb-sql). See the following configuration overrides.

```
onedb-mongo:
    databasePassword:  one1dba4ever

onedb-rest:
    databasePassword:  one1dba4ever

onedb-explore:
    serverConnection:
        password: one1dba4ever
```

To change the user and password for OneDB Mongo use the following configuration override values. If you change this password it is important that you make changes to mongo, rest and explore.

```
onedb-mongo:
    mongoUser: mymongo
    mongoPassword: mongoPassword
```

To change the admin password for OneDB Explore. Use the following configuration override values. If you change this password it is important that you make changes to mongo, rest and explore.

```
onedb-explore:
    adminPassword: newPassw0rd
```

Administering OneDB

The following information describes the common tasks that an administrator may perform for a OneDB Database server in a kubernetes environment.

- Exec into OneDB pod.

  For some administration tasks, you may need to login to the OneDB pod and run commands. To do this, you must have authorization to run kubectl exec.
- Starting and Stopping OneDB.

  Starting and stopping the OneDB pod from kubernetes
- Viewing OneDB Log files.

  To view the OneDB logs in the different pods.
- Backup and Restore.

  To backup and restore the OneDB database server

Exec into OneDB Pod

There may be a need to login to the kuberneres pod to perform administration tasks. First identify the pod you need to login to.

```
kubectl get pods |grep onedb
onedb-operator-67f5d6cd9b-wxcg2 1/1 Running 0 42h
onedb-server-0                  1/1 Running 0 42h
onedb-server-1                  1/1 Running 0 42h
onedbcm-0                       1/1 Running 0 42h
onedbcm-1                       1/1 Running 0 42h
```

Run the kubectl command to login to the kubernetes pod:

```
kubectl exec --stdin --tty onedb-server-0 -- bash
```

You will be logged in as user “informix” and your environment will be set for the OneDB Database server. Which can be verified with the onstat - command:

```
onstat -
HCL OneDB Server Version 2.0.1.0 -- On-Line (Prim)
 -- Up 1 days 18:42:25 -- 793248 Kbytes
2021-12-01 17:37:54
```

Stop/Start OneDB Database Server

Use one of the following methods to stop and restart the OneDB database server:

1. Delete the kubernetes pod
2. Login to Pod and take the OneDB server offline

Delete Pod

About this task

Deleting a pod is a method that can be used to restart the pod. When a pod is deleted or dies, kubernetes will force the pod to be restarted.

1. Delete the pod of interest.

```
kubectl delete pod onedb-server-0
```

2. Monitor the pod that was deleted, as it is restarted. After performing the **kubectl delete pod** the pod will terminate. As it restarts, it will go back into the initialization and eventually a Running state.

```
onedb-server-0                                  0/1     Terminating    0          42h
onedb-server-0                                  0/1     Init:0/1           0          1s
onedb-server-0                                  1/1     Running          0          43s
```

Login Pod

1. First annotate the pod so kubernetes does not restart the pod while you are performing administration tasks:

```
kubectl annotate pod onedb-server-0 livenessprobe=disabled --overwrite=true
```

2. Use the command to take the OneDB Database server offline:

```
onmode -kuy
```

3. Perform any adminstration tasks needed with the server offline.
4. Run the command to bring the OneDB Database server back online.

```
oninit
```

5. Re-enable the liveness probe:

```
kubectl annotate pod onedb-server-0 livenessprobe=enabled --overwrite=true
```

Note: If you do not disable the liveness probe, once you take the OneDB Databaser server offline. The kubernetes liveness probe will begin to fail. After 3 failures of the liveness probe, kubernetes will restart the pod on its own.

Viewing log files

About this task

Use one of the following methods to view the OneDB Database server logs:

1. Use kubectl to view the pod logs

```
kubectl logs onedb-server-0
```

2. View the log files from inside the pod. The example below shows a tail of the online.log. With this method you can view any log file associated with the OneDB Database Server.

```
Exec into the pod
cd $ONEDB_DATA_DIR/logs
tail –f onedb*.logs
```

3. When a pod starts up it will sometimes use an init container to perform setup work prior to the main pod starting.

```
kubectl logs onedb-server-0 –c onedb-init
```

Backup and Restore

The OneDB SQL Datastore supports an HA cluster setup. To do this one of the requirements is that a shared RWM storage is available for use. When setting up HA with OneDB an archive is taken from the OneDB HA Primary Server and restored to a OneDB HA Secondary or OneDB HA RSS system.

The shared volume provides all pods access to the archive's. When deploying a OneDB helm chart one of the initial steps is to take an archive to be used to setup the Secondary and any RSS nodes.

The default behavior is for a level 0 archive to occur every night at 2:30am. The last three backups are retained and any prior archives are cleaned up and removed.

A restore should only be needed if both the OneDB HA primary and OneDB HA secondary servers are corrupted. If just one or ther other is corrupted then a failove scenario can occur to bring the two servers back into sync with one another.

Change Backup Schedule

About this task

Backups are scheduled by the OneDB Scheduler. **Cloud Backup** scheduler task determines what days and what time the level 0 backups occur.

This can be changed by modifying the Scheduler task. You can modify the archive schedule from the command line or you can do this through OneDB Explore.

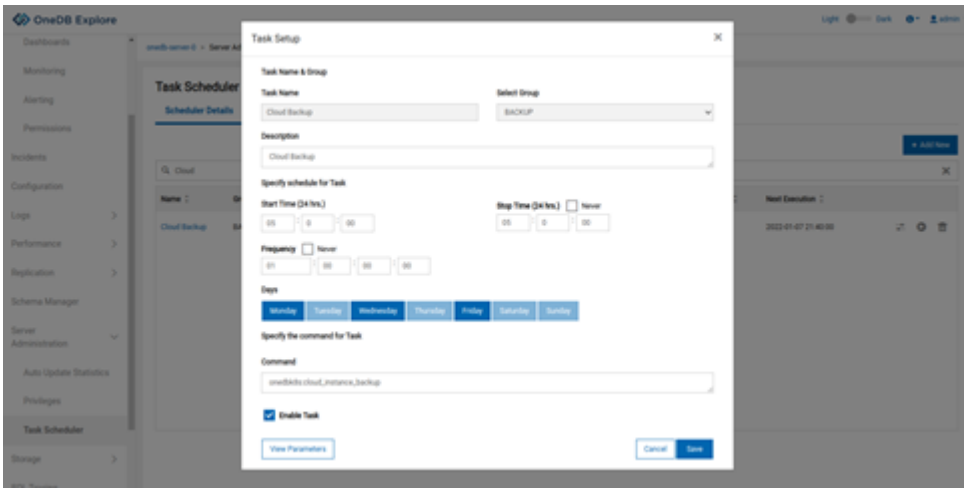
Using Command Line

1. Exec into onedb-server-0 pod.
2. Use dbaccess run an update statement against the sysadmin database.
    - update sysadmin:ph_task set …..... where tk_name=”Cloud Backup”;

Using OneDB Explore

1. Login to OneDB Explore.
2. Select onedb-server-0.

3. From the Left Panel choose **Server Administration -> Task Scheduler**.
4. Search for the **Cloud Backup** Task.
5. Select the **Cloud Backup** task and Edit accordingly.



In the above example, the archive time is changed to 05:00 and to occur on Monday, Wednesday and Friday.

Restore an archive

About this task

To restore from an archive or a backup:

1. Scale back the server to a single server pod;
   a. Set serverReplicaCount helm parameter to 1:

```
count.yaml

onedb:
    serverReplicaCount: 1
```

   b. Run a helm upgrade:

```
helm upgrade  <release.name> -f count.yaml -f <previous values>  onedb/onedb-production
```

2. Remove PVC's related to deleted server pods. onedb-server-1 and higher. This should be done by your **kubernetes administrator**:

```
kubectl get pvcs

onedb-onedb-server-1              Bound    pvc-bce5170   10Gi       RWO            standard       68m
onedb-onedb-server-2              Bound    pvc-ba34270   10Gi       RWO            standard       68m

kubectl delete pvc <pvc's for onedb-server-1 and higher>
```

3. Set the restoreFromBackup helm parameter and run a helm upgrade to initiate the database restore:

a. Set restoreFromBackup to **true**:

```
restore.yaml

onedb:
    restoreFromBackup: true
```

b. Run the helm upgrade:

```
helm upgrade  <release.name> -f count.yaml -f <previous values>  onedb/onedb-production
```

4. After restore is complete set restoreFromBackup helm parameter back to false and update serverReplicaCount to appropriate values and run helm upgrade to scale out the HA cluster.
    a. Set restoreFromBackup to **false** and serverReplicaCount to desired value:

```
after.yaml

onedb:
    restoreFromBackup: false
    serverReplicaCount: 2
```

b. Run helm upgrade:

```
helm upgrade  <release.name> -f after.yaml -f <previous values>  onedb/onedb-production
```

Disable OneDB archives

About this task

You can disable the OneDB backups at deployment time by providing the following configuration override values:

```
onedb:
    customConfig:
        LOG_BACKUP_MODE: “NONE”

    customInitSQL: |-
        database sysadmin;
        update sysadmin:ph_task set tk_enable=’f’ where tk_name=”Cloud Backup”;
```

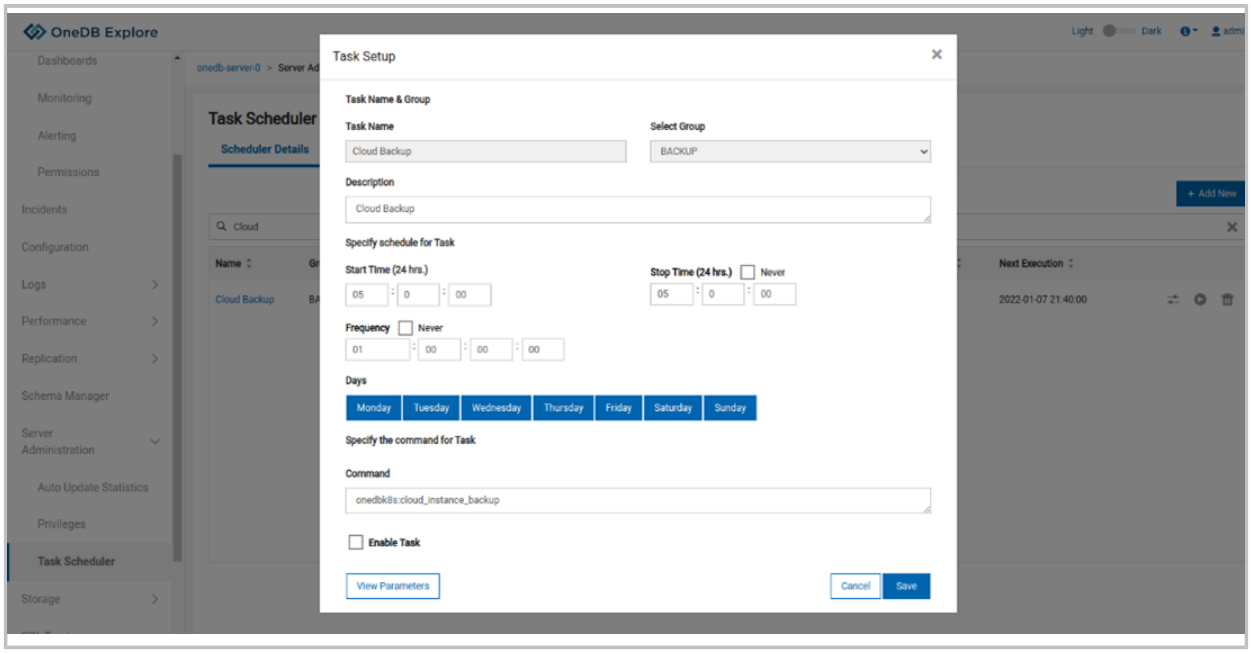
You can also disable the archives after OneDB helm charts have already been installed. You can do this from the command line or you can do this through OneDB Explore.

Using Command Line

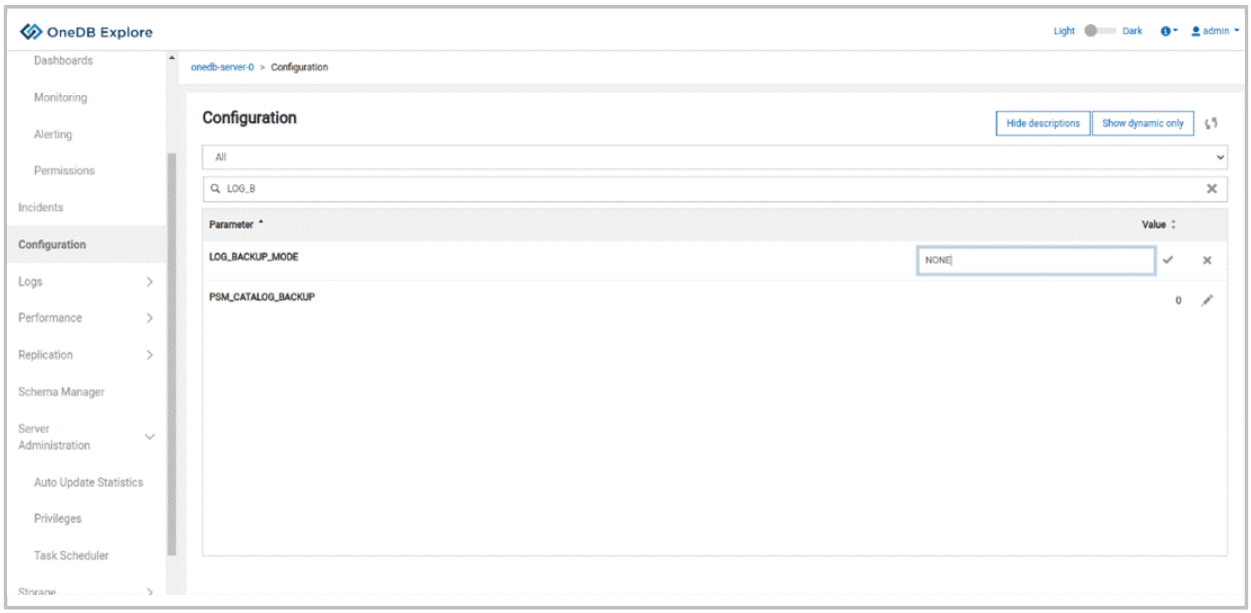
1. Exec into onedb-server-0 pod.
2. vi $ONEDB_HOME/etc/$ONCONFIG.
3. Change LOG_BACKUP_MODE to NONE.
4. Use dbaccess run an update statement against the sysadmin database.
    - update sysadmin:ph_task set tk_enable=’f’ where tk_name=”Cloud Backup”;

Using OneDB Explore

1. Login to OneDB Explore.
2. Select onedb-server-0.
3. From the Left Panel choose **Server Administration -> Task Scheduler**.
4. Search for the **Cloud Backup** Task.
5. Select the **Cloud Backup** task and Edit accordingly.
6. Uncheck the Enable Task button and Save.



7. Select **Configuration** from the Left Panel.
8. Search for the **LOG_BACKUP_MODE** parameter.
9. Edit this value and change to NONE.



Recover Failing pod

If you are running OneDB as an HA cluster. A primary and secondary and you have a failure of a single pod that doesn't recover, you don't need to perform a restore. Instead you can recover only the failing pod.

To force the pod to be recovered set an annotation to start the recovery:

```
kubectl annotate pod onedb-server-1 onedb_force_ifxclone=true -overwrite=true
```

Once the pod has successfully be cloned disable this annotation:

```
kubectl annotate pod onedb-server-1 onedb_force_ifxclone=false -overwrite=true
```

Note: This shows a recovery of pod onedb-server-0.

Archive with Kubernetes Solution

If you prefer to do your own backups with a kubernetes solution you can do this. First disable the OneDB backups. And when performing the non-OneDB backup it is important to flush all Database activity to disk. This can be performed using External backup and Restore (EBR).

For more information on performing a backup using EBR, see External backup and restore overview.

OneDB Explore

OneDB Explore as a graphical User Interface that can be deployed in the OneDB helm charts. It can be used to monitor and administer one or more OneDB Database servers.

The OneDB Explore helm chart can be deployed separately or as part of OneDB Product. When deploying the OneDB Explore helm chart on its own it is left up to the user to configure and setup. If you use the OneDB Explore with a OneDB Product deployment, then it will be configured and setup automatically for the OneDB HA cluster.

Below are the two OneDB helm charts that OneDB Explore is included with. The default **admin** user password is **testPassw0rd**. For information on changing this default, see Accessing OneDB on page 179.

- **OneDB Explore (onedb-explore):** This is a Helm chart that contains only the OneDB Explore UI.
- **OneDB Product (onedb-product):** This is a Helm chart that contains OneDB SQL Data Store, OneDB Rest, OneDB Document and OneDB Explore all as subcharts.

Configuration

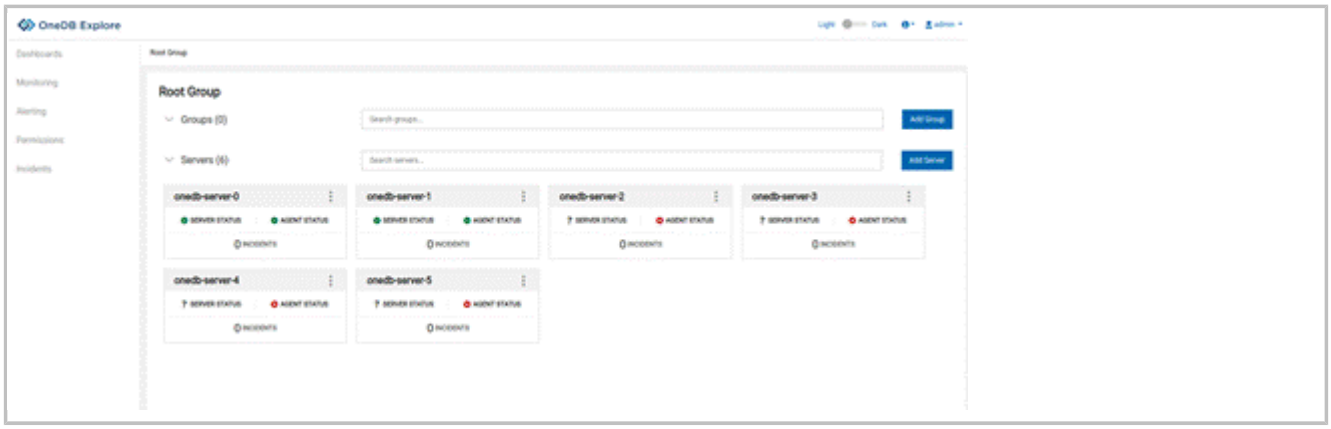
This topic explain the specifics of OneDB Explore in a kubernetes environment. For more information on OneDB Explore and its functionality and capabilities, see OneDB Explore guide.

In the image below, 6 servers have been pre-configured:

A default deployment of OneDB SQL Data Store will create an HA Primary/Secondary Cluster. The deployment will pre-configure 6 servers with OneDB Explore. If you need more than those configured you can add additional servers. If you want to remove any unused servers, you can delete them.

Monitoring

By default, the OneDB Explore agent is enabled on each OneDB Database server. In the following figure, you can see that the first two servers have a green Agent status. This indicates that monitoring is enabled.



To disable the Agent on the OneDB Database server set the configuration override values:

```
onedb-product:
    onedb:
        exploreAgent: false
```

High Availability

When deploying the OneDB SQL Data Store helm chart or one of the charts that includes this as a subchart the default behavior is that you will get an HA cluster with a primary and secondary server. The primary server will be running in onedb-server-0 pod and the updatable HDR secondary will be running in onedb-server-1 pod. HDR replication is configured to use NEAR_SYNC replication mode to avoid data loss.

The OneDB database server cluster is deployed using **onedb-server** statefulset. There will be a second statefulset **onedbcm** deployed using two connection manager pods, onedbcm-0 and onedbcm-1.

The connection manager SLA definitions are configured to use R**OUNDROBIN** policy. This can be changed to **WORKLOAD** by setting the **onedbcm.sla_policy** helm parameter.

Failover

The OneDB SQL Data store HA cluster supports automatic failover and manual failover. The default is set for automatic failover of the HA cluster. When the HDR primary server becomes non-responsive the HDR secondary needs to take over the primary responsibilities. This can happen automatically, or it can be configured to require manual intervention.

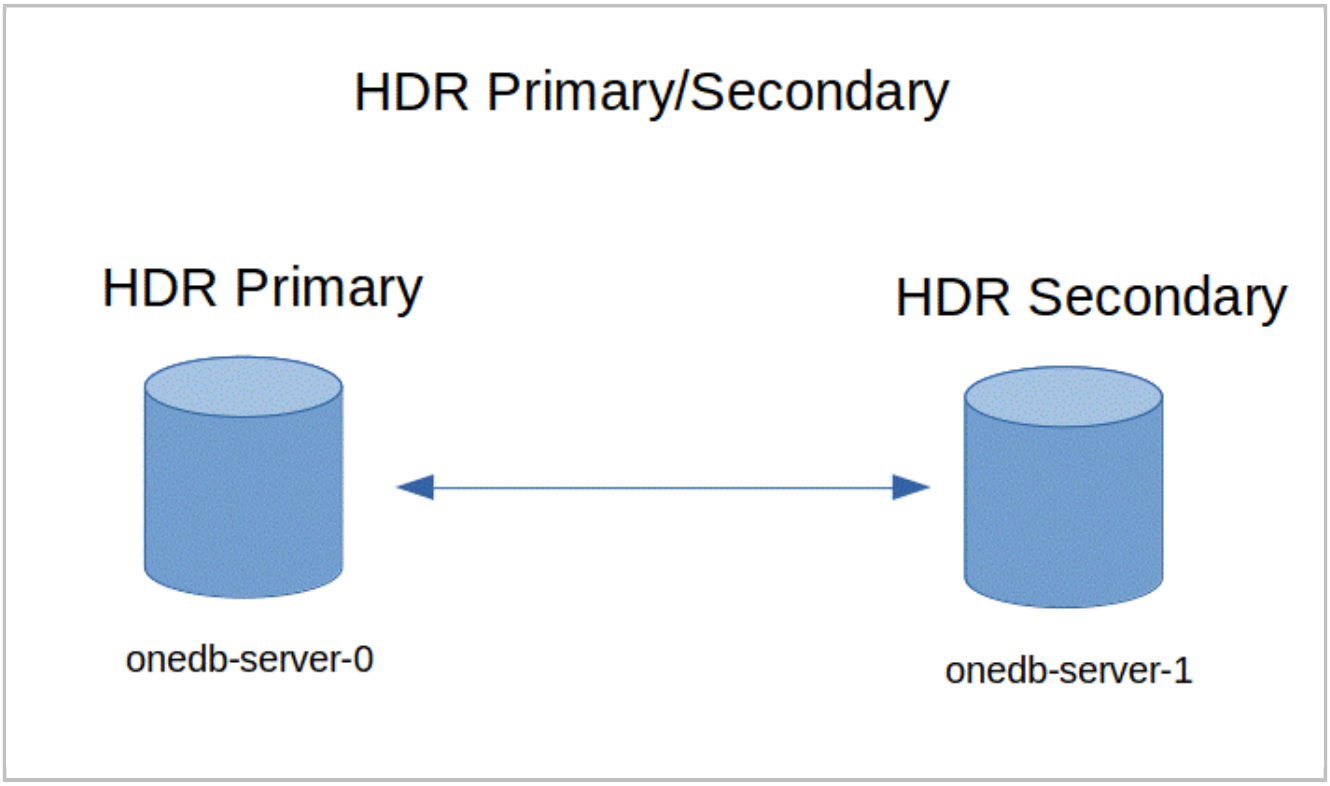
Auto failover functionality is designated by using the **onedbcm.autofailover** helm parameter in the OneDB SQL Data Store helm chart. By default, this value is set to **true**. If manual failover is preferred this can be set to **false**.

When failover is performed whether its automatic or manual the roles will toggle between pods onedb-server-0 and onedb-server-1. When a pod is restarted it will restart the OneDB database server as primary or secondary based on the peer pod and current state of that OneDB server.

Manual Failover

Manual Failover

When the HDR primary server is non-responsive the kubernetes health scripts will fail and the HDR primary server pod will restart. The pod will be restarted as an HDR primary. If the server comes back to a healthy state, then no failover is required.



If the restart of the HDR primary does not come back to a healthy state then manual intervention is needed. You must login to the HDR Secondary, onedb-server-1 pod, and switch it to the HDR primary. The onedb-server-0 pod will not become healthy and will restart as the HDR secondary.

```
onmode -d make primary onedb1
```

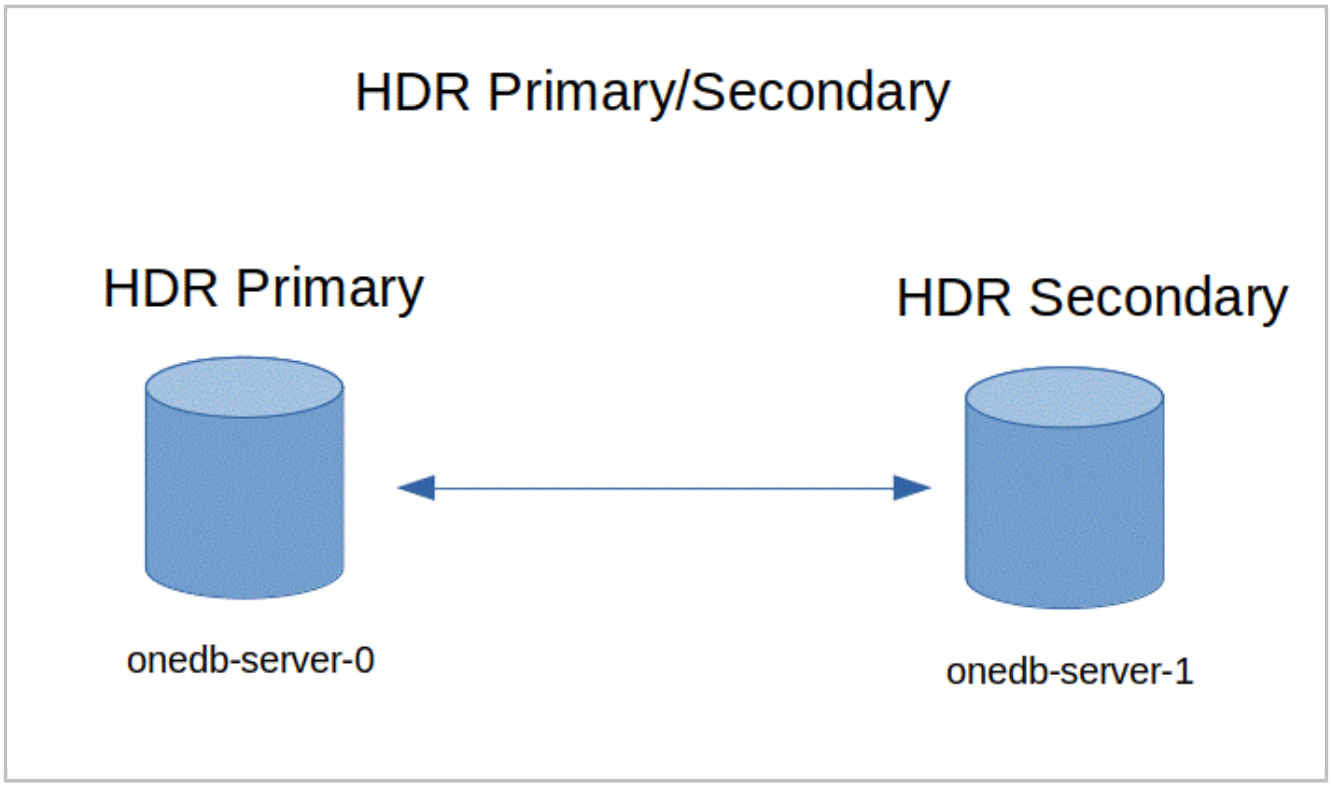
If TLS encryption is being used the name of the server is **onedb1_ssl**. So, the command would be:

```
onmode -d make primary onedb1_ssl
```

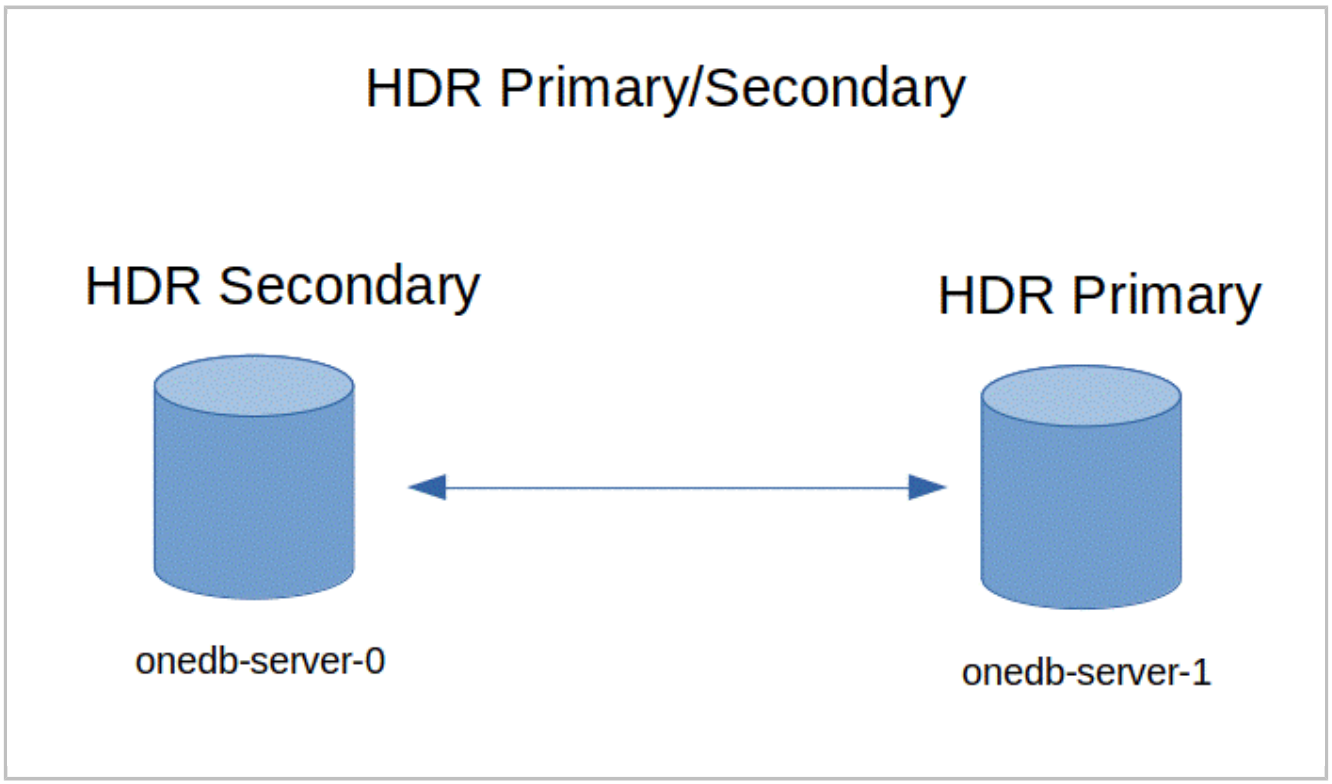
Automatic Failover

Automatic Failover

When the HDR primary server is non-responsive the connection manager will switch the HDR secondary to become the HDR primary. When the old primary server is restarted it will restart as the HDR secondary server.

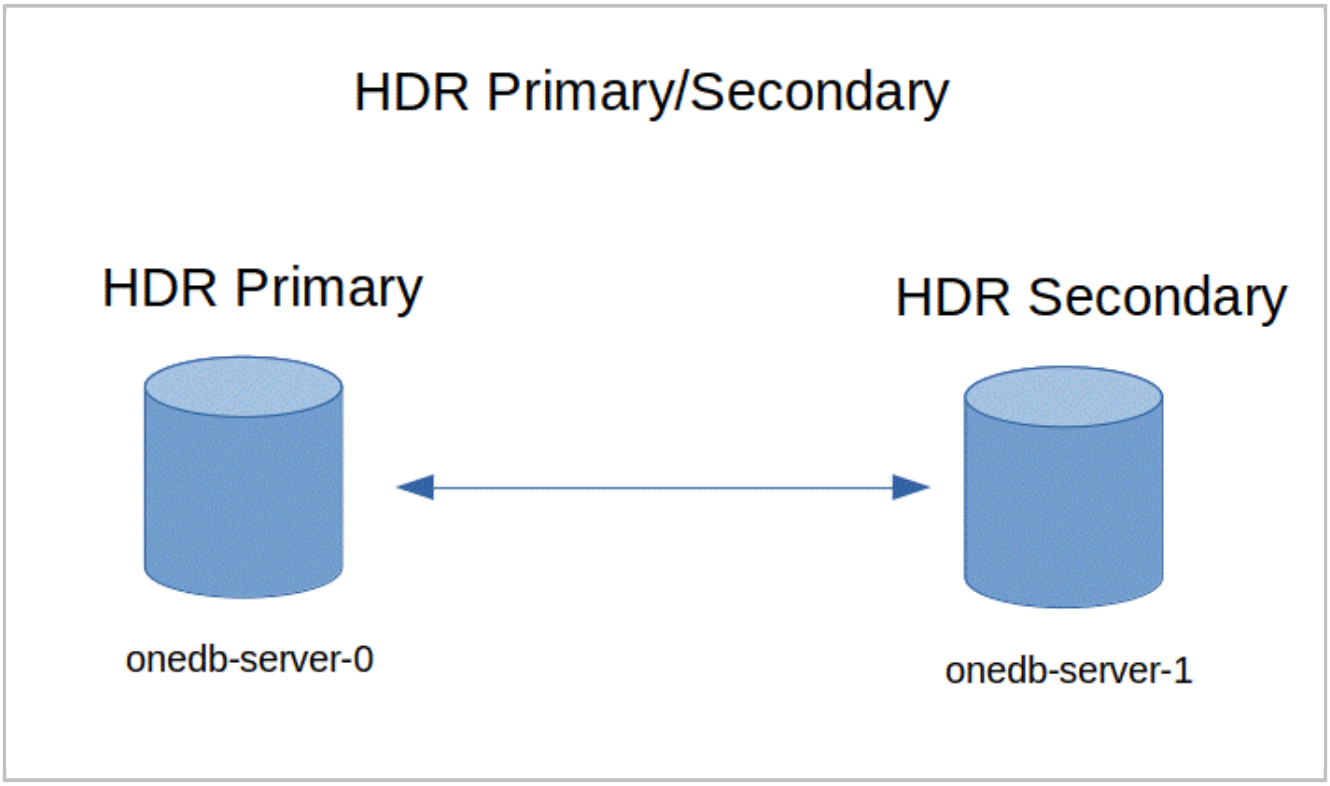


Automatic failure occurs and onedb-server-1 will be made the HDR Primary, and onedb-server-0 restarts as an HDR secondary.



Manual Failover

When the HDR primary server is non-responsive the kubernetes health scripts will fail and the HDR primary server pod will restart. The pod will be restarted as an HDR primary. If the server comes back to a healthy state, then no failover is required.



If the restart of the HDR primary does not come back to a healthy state then manual intervention is needed. You must login to the HDR Secondary, onedb-server-1 pod, and switch it to the HDR primary. The onedb-server-0 pod will not become healthy and will restart as the HDR secondary.

```
onmode –d make primary onedb1
```

If TLS encryption is being used the name of the server is **onedb1_ssl**. So, the command would be:

```
onmode –d make primary onedb1_ssl
```

Scale-Out

The OneDB SQL Data Store by default will start an HDR Primary + Secondary HA cluster. OneDB SQL Data Store allows the scaling of the OneDB Database server and the OneDB Connection manager. The default settings for both the **onedb.serverReplicaCount / onedbcm.cmReplicaCount** helm parameters are **2**.

Another helm chart parameter, **onedb.maxReplicacount,** controls the maximum number of servers that can be used. The default setting for this parameter is 10 and the max supported value is 10. This is an immutable value and once it is set its value cannot be changed.

The OneDB server pods are as follows:

- onedb-server-0: HDR Primary
- onedb-server-1: HDR Secondary
- onedb-server-[2-9]: HDR RSS

When an HDR secondary or RSS is created, **ifxclone** is used to clone the new server from the current HDR primary server. This applies to an initial setup or a scale out scenario.

The maximum number of replicas for the OneDB Connection manager (**onedbcm.cmReplicaCount**) is onedb.maxReplicaCount.

Manual Scale-Out

OneDB supports manual scale out for the OneDB server and the OneDB Connection Manager.

For the OneDB Server to scale out the number of servers manually set the his is accomplished by set the helm parameter **onedb.serverReplicaCount**.

For the OneDB Connection manager to scale out the number of connection managers manually set the his is accomplished by set the helm parameter **onedbcm.cmReplicaCount**.

Manual Scale out of Connection Manager

The onedbcm.cmReplicaCount helm chart parameter can be changed at any time with the helm upgrade command. You can increase or decrease the number of connection managers in the HA cluster by changing this value.

```
helm upgrade onedb-v1 -set onedbcm.cmReplicaCount=3 -f myvalues.yaml onedb-product
```

The pods for the connection manager are onedbcm-0, onedbcm-1, onedbcm-2, and so on.

Automatic Scale-out

The OneDB SQL Data store uses the kubernetes resource **Horizontal Pod Autoscaler (HPA)** to control how scaling will occur automatically. For more information on HPA refer to the kubernetes documentation here: (https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale).

Auto-scale is based on CPU usage and this is disabled by default. To enable auto-scaling set the **autoscale.enabled** helm chart parameter to **true** and set **autoscale.targetCPUUtilizationPercentage** to a percentage value where you want scaling to occur.

The minimum pods for auto scaling would be the helm chart parameter **onedb.serverReplicaCount** for the OneDB Server and **onedbcm.cmReplicaCount** for the Connection manager. The maximum pods for auto scaling would be the **onedb.maxReplicaCount** helm chart parameter.

Important: When enabling auto scaling OneDB Server and Connection Manager resources should be explicitly configured. See **onedb.resources** and o**nedbcm.resources** helm chart parameters.

Following table shows the HPA resource in kubernetes:

Table 7. $ kubectl get hpa Output

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
onedb-c5abcd-hpa	StatefulSet/onedb-server	8%/90%	2	10	2	3h34m
onedbcm-c5abcd-hpa	StatefulSet/onedbcm	5%/90%	2	10	2	3h34m

In the above output the **onedb-c5abcd-hpa** is for the OneDB server and **onedbcm-c5abcd-hpa** is for the Connection Manager. The CPU threshold is set to 90% for each and we see minimum pods is set to 2 with maximum set to 10. Currently there are 2 of each.

Automatic Scale-out

The OneDB SQL Data store uses the kubernetes resource **Horizontal Pod Autoscaler (HPA)** to control how scaling will occur automatically. For more information on HPA refer to the kubernetes documentation here: (https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale).

Auto-scale is based on CPU usage and this is disabled by default. To enable auto-scaling set the **autoscale.enabled** helm chart parameter to **true** and set **autoscale.targetCPUUtilizationPercentage** to a percentage value where you want scaling to occur.

The minimum pods for auto scaling would be the helm chart parameter **onedb.serverReplicaCount** for the OneDB Server and **onedbcm.cmReplicaCount** for the Connection manager. The maximum pods for auto scaling would be the **onedb.maxReplicaCount** helm chart parameter.

Important: When enabling auto scaling OneDB Server and Connection Manager resources should be explicitly configured. See **onedb.resources** and o**nedbcm.resources** helm chart parameters.

Following table shows the HPA resource in kubernetes:

Table 8. $ kubectl get hpa Output

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
onedb-c5abcd-hpa	StatefulSet/onedb-server	8%/90%	2	10	2	3h34m
onedbcm-c5abcd-hpa	StatefulSet/onedbcm	5%/90%	2	10	2	3h34m

In the above output the **onedb-c5abcd-hpa** is for the OneDB server and **onedbcm-c5abcd-hpa** is for the Connection Manager. The CPU threshold is set to 90% for each and we see minimum pods is set to 2 with maximum set to 10. Currently there are 2 of each.

Archive Restore Considerations

When an archive restore is initiated OneDB it will occur on onedb-server-0, pod #0. When a restore occurs OneDb will try to salvage the logical logs, backup up the current logical log.

If the primary is on onedb-server-1 and the onedb-server-0 pod is the secondary, then no salvaging of logical logs will happen. Which means the current logical log will not be archived and available for the restore.

To prevent this from occurring it is recommended to have onedb-server-0 be the Primary server in the HA cluster. If you are in a situation where the HDR primary is on onedb-server-1 you can force a switch over.

Manual Switch Over

If the onedb-server-1 server is the HDR primary and onedb-server-0 server is the HDR secondary you can switch these two roles when automatic failover is disabled by doing the following. Login to the onedb-server-0 server (current HDR Secondary) and perform the failover operation by running the following command:

```
onmode –d make primary onedb0
```

If TLS is enabled for the OneDB HA cluster use the following command:

```
onmode –d make primary onedb0_ssl
```

Automatic Switch Over

If the onedb-server-1 server is the HDR primary and onedb-server-0 server is the HDR secondary you can switch these two roles when automatic failover is enabled by doing the following.

During a planned downtime login to the onedb-server-1 (current HDR primary) and run the following commands:

```
onmode –c
# Wait for checkpoint to complete on primary & secondary
onmode –ky
```

After forcing a checkpoint (onmode –c) verify the checkpoint has completed on the primary and secondary servers by logging in to each and running onstat –m, looking for Checkpoint completed message. After verification of checkpoint, then run the onmode –ky command.

This will cause the HDR primary on onedb-server-1 to go offline. The onedb-server-0 will automatically failover from HDR secondary to HDR primary. And when onedb-server-1 comes back up it will restart as the HDR secondary.

Upgrading OneDB helm charts

When upgrading your helm chart, it is always recommended to take a database backup before upgrading the product. The helm upgrade command is used to upgrade the current release with new configuration. Or, it can be used to upgrade the current version of the chart to a new helm chart version.

OneDB SQL Data Store and the Connection Manager statefulsets support rolling upgrade. The upgrade process will start from the highest pod ordinal index to the lowest pod ordinal index. For example, onedb-server-1 is updated before onedb-server-0.

The goal of OneDB's upgrade process is to have as little interruption as possible. During an upgrade, kubernetes pods are restarted which will cause a slight interruption in write activity.

If the OneDB Database server does not need to perform a database conversion, then read activity can continue throughout the upgrade process. If a database conversion has to occur then there will be a slight interruption in read activity as well.

To maintain read activity during the upgrade process, your application must be designed with retry logic in it. When a pod is taken down so that it can be upgraded your application should retry its connection so it can connect to and use another server in the cluster.

Upgrading Current release

There may be times when you need to make changes to an existing running release of OneDB in kubernetes. This is performed using helm upgrade and providing the same installed chart with any new values. Parameter values you can change are:

- Set ReplicaCount
- Change container image
- Initiate onbar restore
- Change Connection Manager Service Type: Loadbalancer, ClusterIP, NodePort
- Enable/Disable Automatic failover using Connection Manager
- Change Connection Manager SLA policy: Workload, Round Robin

If the initial installation was performed with this:

```
helm install onedb-v1  -f myvalues.yaml production-onedb
```

The default installation of the helm chart will install an HA cluster with a primary and secondary OneDB server. If you wanted to manually scale the HA cluster to a 3rd server (RSS), you can use helm upgrade and specify a new serverReplicaCount value.

File: newvalues.yaml

```
onedb-product:
    onedb-sql:
        onedb:
            serverReplicaCount: 3
```

Issue the helm upgrade with the original and new values overrides.

```
helm upgrade onedb-v1  -f myvalues.yaml -f newvalues.yaml production-onedb
```

Upgrading 1.x.x.x to 2.x.x.x

A helm upgrade is not supported from OneDB 1.0.0.0 helm chart to a OneDB 2.x helm chart. There are major differences between these two helm chart versions that would prevent a helm upgrade.

When upgrading from a helm chart version using OneDB 1.x to 2.x then you must perform a data migration. It is recommended to use the dbexport.

Note: It is recommended to use the dbexport and dbimport utilities.

Upgrading from 2.0.0.0 version to current version

When upgrading to a new helm chart version you can use the helm upgrade command. This will also most likely be upgrading the version of OneDB database server. For example. Upgrading helm chart version 0.3.52 to 0.4.12 is an upgrade from OneDB 2.0.0.0 to OneDB 2.0.1.0.

When upgrading to a new helm chart version you can use the helm upgrade command. This will also most likely be upgrading the version of OneDB database server. For example. Upgrading helm chart version 0.3.52 to 0.4.12 is an upgrade from OneDB 2.0.0.0 to OneDB 2.0.1.0.

```
helm install onedb-v1  -f myvalues.yaml production-onedb-0.3.52
```

To upgrade to helm chart production-onedb-0.4.12, helm chart version 0.4.12 running OneDB 2.0.1.0, run the following helm upgrade command.

```
helm upgrade onedb-v1  -f myvalues.yaml production-onedb-0.4.12
```

In the above example, the helm chart production-onedb-0.3.52 is used for the OneDB 2.0.0.0 OneDB product. And the helm upgrade command upgrades the helm chart to production-onedb-0.4.12 which the helm chart running OneDB 2.0.1.0.

Upgrading from 2.0.0.0 version to current version

When upgrading to a new helm chart version you can use the helm upgrade command. This will also most likely be upgrading the version of OneDB database server. For example. Upgrading helm chart version 0.3.52 to 0.4.12 is an upgrade from OneDB 2.0.0.0 to OneDB 2.0.1.0.

When upgrading to a new helm chart version you can use the helm upgrade command. This will also most likely be upgrading the version of OneDB database server. For example. Upgrading helm chart version 0.3.52 to 0.4.12 is an upgrade from OneDB 2.0.0.0 to OneDB 2.0.1.0.

```
helm install onedb-v1  -f myvalues.yaml production-onedb-0.3.52
```

To upgrade to helm chart production-onedb-0.4.12, helm chart version 0.4.12 running OneDB 2.0.1.0, run the following helm upgrade command.

```
helm upgrade onedb-v1  -f myvalues.yaml production-onedb-0.4.12
```

In the above example, the helm chart production-onedb-0.3.52 is used for the OneDB 2.0.0.0 OneDB product. And the helm upgrade command upgrades the helm chart to production-onedb-0.4.12 which the helm chart running OneDB 2.0.1.0.

Troubleshooting OneDB

The following documentation talks about some troubleshooting techniques that you might use with OneDB in a kubernetes environment.

From the viewing of log files, to enabling a higher level of logging. To disabling the liveness, probe for the OneDB server pod to prevent kubernetes from automatically restarting the OneDB server pods.

Contact OneDB Support with the diagnostic logs and data mentioned in this section as needed.

Troubleshooting Pods

Each pod that is started by kubernetes goes through a series of steps. Some of the common steps you might see are PodInitializing, Container Creating, Pending, Init, Running, ImagePullBackoff.

If a pod seems to be stuck in a state for a period, some of the following techniques can be used:

```
kubectl get pods


NAME                                      READY       STATUS          RESTARTS   AGE
my-nfs-server-provisioner-0                1/1       Running            0      103s
onedb-operator-86d899b5bf-hklq9            0/1       ImagePullBackOff   0      43s
sofy-1-grafana-b7b5f958d-lxcxf             0/2       PodInitializing    0      44s
sofy-1-ksmetrics-6d4677b7d5-zhtmh          1/1       Running            0      44s
sofy-1-odbp-explore-55c9db47c4-nqx8p       0/1       ErrImagePull       0      42s
sofy-1-odbp-mongo-6f6df887df-gn896         0/1       Init:0/1           0      43s
sofy-1-odbp-rest-64f94dfd98-bzj7x          0/1       Init:0/1           0      43s
```

Troubleshooting ImagePullBackoff, Pending pods

When a pod doesn't make it to the Init/Running state **kubectl describe pod** is commonly used to try to gather more information as to what the problem might be. In the above output we see a few pods in ErrImagePull/ImagePullbackOff.

```
kubectl describe pod onedb-operator-86d899b5bf-hklq9

Events:
  Type     Reason     Age                    From               Message
  ----     ------     ----                   ----               -------
  Normal   Scheduled  34m                    default-scheduler  Successfully assigned sofy-
testing/onedb-operator-86d899b5bf-hklq9 to kind-worker4
  Normal   Pulling    32m (x4 over 34m)      kubelet            Pulling image
"hclcr.io/sofy/services/onedb/onedb-operator:2.0.1.0"
  Warning  Failed     32m (x4 over 34m)      kubelet            Failed to pull image
"hclcr.io/sofy/services/onedb/onedb-operator:2.0.1.0": rpc error: code = Unknown desc = failed
to pull and unpack image "hclcr.io/sofy/services/onedb/onedb-operator:2.0.1.0": failed to
resolve reference "hclcr.io/sofy/services/onedb/onedb-operator:2.0.1.0": unexpected status code
[manifests 2.0.1.0]: 401 Unauthorized
  Warning  Failed     32m (x4 over 34m)      kubelet            Error: ErrImagePull
  Warning  Failed     32m (x6 over 34m)      kubelet            Error: ImagePullBackOff
  Normal   BackOff    4m31s (x128 over 34m)  kubelet            Back-off pulling image
"hclcr.io/sofy/services/onedb/onedb-operator:2.0.1.0"
```

You can see here that we failed to pull the image. This gives us some direction in trying to diagnose this issue.

Troubleshooting init pods

OneDB uses init containers to perform setup functions before a specific pod is fully functional. When a pod is in the init state you can run a kubectl logs command to get information about the pod. When running the kubectl logs command on an init-container you need to know the name of the init-container. This can be obtained from the kubectl describe command.

```
kubectl describe pod sofy-1-odbp-mongo-6f6df887df-gn896

Events:
 Type    Reason   Age              From    Message
 ----    ------   ----             ----    -------
 Normal  Pulled   22m (x6 over 63m)    kubelet  Container image "openjdk:8-alpine" already present
on machine
 Normal  Created  22m (x6 over 63m)    kubelet  Created container onedb-mongo-init
 Normal  Started  22m (x6 over 63m)    kubelet  Started container onedb-mongo-init
 Warning  BackOff  4m49s (x25 over 48m)  kubelet  Back-off restarting failed container
```

Once you find the name of the init container, you can run a kubectl logs command. You specify the pod and the name of the init container in the kubectl logs command.

```
kubectl logs sofy-1-odbp-mongo-6f6df887df-gn896 -c onedb-mongo-init
```

Below is a sample output and we can see there is a problem connecting to the OneDB Database server.

```
Running Main
```

```
SQL Service Test Unsuccessful. Server not ready
-908 : com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
SQL Service Test Unsuccessful. Server not ready
-908: com.informix.asf.IfxASFException: Attempt to connect to database server (null) failed.
```

Troubleshooting running pods

A pod's desired state is to get to a running state with a Ready of 1/1. If you see a Ready Status of 0/1 or see several Restarts for the pod, then you may need to investigate further. The **kubectl log** command can be used to get more information on what the container/pod is doing.

```
kubectl get pods

NAME                                   READY   STATUS    RESTARTS   AGE
helm-1-odbp-explore-686bf69c88-czntt   1/1     Running   0          8m2s
helm-1-odbp-mongo-5bc4d5bb94-54jh4     1/1     Running   0          8m2s
helm-1-odbp-rest-5bd6b956cc-8gqfw      1/1     Running   0          8m2s
my-nfs-server-provisioner-0            1/1     Running   0          8m23s
onedb-operator-6dfdd9bb7d-4phld        1/1     Running   0          8m2s
onedb-server-0                         0/1     Running   0          7m46s
```

Pod onedb-sever-0 hasn't moved into a running state yet and we want to dig deeper into what is going on with this specific pod.

```
kubectl logs onedb-server-0

20:59:00 Peer node onedb1 has version 131077
20:59:00 RSS Server onedb1 - state is now connected
20:59:00 setting version information for onedb1 131077
2022-01-08 20:59:03 LICENSING: <Information> Reacquire licenses: current allocation != expected count
2022-01-08 20:59:03 LICENSING: <Information> Processing current Capability
2022-01-08 20:59:13 LICENSING: <Information> Reacquire licenses: current allocation != expected count
2022-01-08 20:59:13 LICENSING: <Information> Processing current Capability
20:59:13 HDR TIMEOUT - log buffers being sent to onedb1

20:59:13 Error receiving a buffer from RSS onedb1 - shutting down
```

```
20:59:14 RSS Server onedb1 - state is now disconnected
20:59:14 RSS onedb1 deleted
2022-01-08 20:59:23 LICENSING: <Information> Reacquire licenses: current allocation != expected count
2022-01-08 20:59:23 LICENSING: <Information> Processing current Capability
```

Enable/Disable Liveness probe

When doing any type of diagnostic work on a container/pod, it is important that the liveness probe does not take effect and restart the pod. To prevent this from happening you can disable the liveness probe for the OneDB Database server.

To disable use the following kubectl annotation:

```
kubectl annotate pod onedb-server-0 livenessprobe=disabled –overwrite=true
```

Once you are done with your diagnostic work you should re-enable the liveness probe.

To enable, use the following kubectl annotation:

```
kubectl annotate pod onedb-server-0 livenessprobe=enabled –overwrite=true
```

Kubernetes events

Another log of events that can be reviewed/monitored is the Kubernetes events. Run the kubectl get events command and sort or filter this data accordingly.

```
kubectl get events
```

Log in to pod

There may be a need to login to a pod or the init container to obtain more diagnostic information than you get with kubernetes commands. First identify the pod you need to login.

```
kubectl get pods |grep onedb

onedb-operator-67f5d6cd9b-wxcg2 1/1 Running    0 4m
onedb-server-0                  1/1 Running    0 4m
onedb-server-1                  0/1 Init:0/1   0 2m
onedbcm-0                       1/1 Running    0 4m
```

Run the kubectl command to login to the kubernetes pod:

```
kubectl exec --stdin  --tty onedb-server-0 -- bash
```

Once you've logged in to the pod/container, you can move around and view log files as you would on any Linux system.

Log in to init container

To login to an init container, you must first find the name of the init container. This is done using the **kubectl describe pod** command.

```
kubectl describe pod onedb-server-1
```

```
Events:
 Type    Reason     Age   From               Message
 ----    ------     ----  ----               -------
 Normal  Scheduled  76s   default-scheduler  Successfully assigned sofy-testing/onedb-server-0 to
kind-worker
 Normal  Pulling    74s   kubelet            Pulling image "informix-docker-dev-local.us-
artifactory1.nonprod.hclpnp.com/releases/onedb-server:2.0.1.1"
 Normal  Pulled     74s   kubelet            Successfully pulled image "informix-docker-dev-local.us-
artifactory1.nonprod.hclpnp.com/releases/onedb-server:2.0.1.1" in 355.243101ms
 Normal  Created    74s   kubelet            Created container onedb-init
 Normal  Started    74s   kubelet            Started container onedb-init
```

Once you find the name of the init container, you can run the kubectl exec command and login to the init container.

```
Once you find the name of the init container you can run the kubectl exec command and login to the init
 container.
```

Once you've logged in to the pod/container, you can move around and view log files as you would on any Linux system.

Custom init container

Creating a custom init container is a more advanced topic for kubernetes users. An init container is designed to run before starting the main container.

Potential use for custom init container:

- Debug/patch container storage
- Custom container to load data spaces
- Perform any operation on the container/pod prior to starting the container

To use a customer init container, use the following helm parameter override values:

```
onedb:
    customInitImage: gcr.io/google-containers/busybox:latest
    customInitImageCmd: /bin/customization.sh
```

If you needed to login to this container, the name of the init container is **onedb-custom-init**, although we could use the kubectl describe pod command to determine this.

Index

P

R

S

T

U

V