opentext™

OpenText™ Documentum™ Platform and Platform Extensions

Cloud Deployment Guide

This guide contains information about deploying and configuring cloud-native Documentum Platform and Platform Extensions products on different Cloud platforms.

EDCSYCD230200-IGD-EN-02

OpenText™ Documentum™ Platform and Platform Extensions
Cloud Deployment Guide
EDCSYCD230200-IGD-EN-02
Rev.: 2023-May-12

This documentation has been created for OpenText™ Documentum™ Platform and Platform Extensions CE 23.2.
It is also valid for subsequent software releases unless OpenText has made newer documentation available with the product, on an OpenText website, or by any other means.

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1

Tel: +1-519-888-7111
Toll Free Canada/USA: 1-800-499-6544 International: +800-4996-5440
Fax: +1-519-888-0677
Support: https://support.opentext.com
For more information, visit https://www.opentext.com

Table of Contents

Chapter 1

Overview

As OpenText Documentum advances towards the cloud world, it is also designed to guide you to advance your journey towards the cloud world by providing you with roadmaps to adopt new cloud deployments, while continuing to support legacy environments and have hybrid implementations. The move to the cloud world is driven by the following key capabilities for you to:

- reduce the high operating costs to develop, manage and maintain on-premises applications
- avoid end user adoption issues caused by slow performance and lengthy deployment timelines
- gain access to extensive resources to support EIM applications
- deploy EIM applications and grow as needed to scale to your business needs

With evidence that the cloud is the future for data, and that it is imminent that enterprise workloads will run in the cloud, OpenText encourages you to choose the cloud over an on-premises solution.

Documentum offers a single All-In-One Helm chart for supported Documentum Platform and its components' containers. This approach defines a common layout within the Documentum CE solutions and incudes all containers with the ability to deploy and upgrade using a single Helm package. You can open the `values.yaml` file of the single Helm package, enable or disable Platform and its components (for example, Documentum Administrator, Documentum REST Services, and so on in the respective categories within a single Helm chart. After providing the appropriate values as per your business requirement, you can choose to deploy multiple containers along with Documentum Server with one `helm install` command. Similarly, you can use one `helm upgrade` command to upgrade multiple containers.

Chapter 2

Documentum Platform and Platform Extensions applications on Docker

You can deploy and configure Documentum Platform and Platform Extensions applications on the supported Docker containers.

Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating system level virtualization. Docker uses resource isolation features of the Linux kernel such as cgroups and kernel namespaces to allow independent containers to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines. Docker is a tool that can package an application and its dependencies in a virtual container that can run on any Linux server.

The product *Release Notes* document contains detailed information about the list of applications and its supported versions.

Note: Docker images for Documentum Server with the Oracle Linux/ PostgreSQL configuration is provided.

2.1 Installing Docker

1. Log in with root account and install the Docker Engine.
2. Install the Docker Compose.
3. Start the Docker daemon service.

Docker Documentation contains detailed information.

2.2 Using container image tags

Container images in the OpenText Container Registry are identified by the base container image name and a tag (version). You can obtain container images using two types of tags:

- `YY.Q.#`

  In a tag of this style, the dot-separated numbers represent the year, quarter, and point release. This type of image tag identifies a specific point-in-time release with the patches and configurations that were available at the time that the image was built.

Use a `<YY.Q.#>` tag to specify the exact version number of the image you want to download. For example, if you download a container image of `dctm-server:<YY.Q.#>` using the `docker pull registry.opentext.com/dctm-server:<YY.Q.#>` command, you obtain the specific version of the `Documentum Server` container image that you specify, even if more recent images are available.

- `YY.Q`

  In a tag of this style, the dot-separated numbers represent the year and quarter. This type of tag identifies the latest version of a container image available in the OpenText Container Registry.

  Use a `<YY.Q>` tag to obtain the latest image available that has the most recent patches and configurations. For example, to download the latest available version of the `Documentum Server` container image, use a `docker pull registry.opentext.com/dctm-server:<YY.Q>` command.

Notes

- OpenText recommends that you use the `<YY.Q>` tag in your `docker pull` command to make sure that you have the latest available image including all patches available at the time the image was built.
- In special cases, OpenText may advise you to use an image with a `<YY.Q.#>` tag for testing or other purposes.

2.3 Deploying and configuring Documentum Server on Docker

2.3.1 Prerequisites

1. Check your kernel version. You must have the supported version of Red Hat Enterprise Linux for the Docker support. It requires a 64-bit installation regardless of your Linux version and you should have the supported version of kernel. To check your current kernel version, open a terminal and run the following command:

   ```
   uname -r
   ```

2. (Only for Oracle Linux) Check the file system type. If the Docker mount point is an xfs file system, then set *`d_type`* to true. *Docker Documentation* contains detailed information.

3. Download the Docker image from OpenText Container Registry. Perform the following tasks:

   a. Log in to OpenText Container Registry using the following command format:

      ```
      docker login registry.opentext.com
      ```

      When prompted, provide your OpenText My Support login credentials.

b. Download the Docker image using the following command format:

```
docker pull registry.opentext.com/<image_name>:<image_tag>
```

The following Docker images are available for download:

Image name	Image tag
dctm-server	23.2.0 or 23.2

4. Download the Helm charts from OpenText My Support.

5. Download the Docker Compose file from OpenText My Support.

6. For Docker Documentum Server image, the location of *`INSTALL_HOME`* is `/opt` for the Oracle Linux image. Make sure that in the Docker Compose file, you use `/opt` as your Documentum home directory.

2.3.2 Common notes

- For internal applications, use the internal connection broker (for example, running on 1489). Do not have any translations. Point `dfc.properties` of internal clients to use internal Docker IP. For external applications, use the external connection broker (for example, running on 1689). Translations are done by the Docker scripts automatically. The translation is from internal Docker IP to external IP. Point `dfc.properties` to external IP of the connection broker.

- If you use remote file system with netshare plug-in, then make sure to install the respective NFS or CIFS RPMs in Docker host machine. For example: `yum install nfs*` (for NFS) and `yum install cifs*` (for CIFS).

- For stateless configuration on CIFS, you must create the volumes manually and then use the GID and UID identifiers for netshare plug-in. *Docker Documentation* contains the workaround details.

- UID should be synchronized or same when deploying Documentum Server between different host systems for seamless upgrade or sharing of data between two systems.

- If you use netshare plug-in for remote data and for any reason if the service is restarted, you must run the following command:

  ```
  docker-compose -f <compose file name> up -d
  ```

  You can use the configured YML file names in your `up` command to start the container.

  For example, Documentum provides the following compose YML files:

  – `CS-Docker-Compose_Stateless.yml`

  – `CS-Docker-Compose_Ha.yml`

  – `CS-Docker-Compose_Seamless.yml`

- In stateless and HA configurations, to make sure that the Docker volumes are synchronized, you must use the project name or place the configuration files in the same location.

  For example, you can use the following command:

  ```
  docker_compose -f CS-Docker-Compose_Stateless.yml -p <project name> up -d
  ```

- If your database is PostgreSQL, perform the following tasks depending on your operating system:

  – Linux: Log in as a postgres user and create a folder called `db_<RepositoryName>_dat.dat` in `/var/lib/pgsql/<supported PostgreSQL version number>/data/`.

  – Windows: Log in as a postgres user and create a folder called `db_<RepositoryName>_dat.dat` in `C:\Program Files\PostgreSQL\<supported PostgreSQL version number>\data\`.

  Note: During the deployment, you can create a folder called `db_<RepositoryName>_dat.dat` in `/var/lib/pgsql/<supported PostgreSQL version number>/data/` and then select the **Use Particular tablespace** option. In addition, you can proceed with the deployment without creating a folder in `/var/lib/pgsql/<supported PostgreSQL version number>/data/` and then select the **Use default Tablespace** option.

- For asynchronous write and precaching operations in Docker, perform the following tasks:

  – Create the DMS configuration having message_post_url, message_consume_url with the internal IP (for example, http://172.17.0.1:8489/).

  – Change the following in `dms.properties`:

    ○ Provide the external IP for dms.webservice.update.url (for example, dms.webservice.update.url = http://10.31.86.166:8489).

    ○ Provide the internal IP for dms.jmx.host (for example, dms.jmx.host = 172.17.0.1).

2.3.3 Creating the Documentum Server Red Hat Enterprise Linux/Oracle Docker image

2.3.3.1 Prerequisites

You must have working knowledge of Docker, Red Hat Enterprise Linux and Oracle. You must have administrative privileges on the machine where you are deploying Documentum Server and also have database administrator account for the Oracle server. You must make sure to update the Docker file with the supported version of JDK and Tomcat versions. The *OpenText Documentum Server Release Notes* contains detailed information about the supported version of JDK and Tomcat. You need two machines with the following configuration:

2.3.3.1.1 Hardware requirements

Item	Requirement
Free disk space	80 GB
RAM	8 GB
Swap space	8 GB
Free space in temporary directory	2 GB

2.3.3.1.2 Software requirements

The product *Release Notes* document contains detailed information for your product.

2.3.3.2 Configuring the Red Hat Enterprise Linux/Oracle Docker image

1. Do one of the following tasks:

   - Download the RHEL image offered from the RHEL hub.
   - Download the RHEL universal base image from the RHEL hub and rebuild the RHEL image on the RHEL server obtained from the subscription.

   RHEL Documentation contains detailed information.

2. Create a container to install all the required RPMs and the respective Oracle client.

   ```
   docker run  -ti  --name <RHEL container name> <RHEL image id/name with tag> /bin/
   bash
   ```

3. Install the required packages. Do one of the following tasks:

   a. Install the required packages from the RHEL repository as the image you downloaded is subscription-enabled. Use the `yum install` command to install the required packages.

   b. Copy the required packages to the container. Use the `RPM` command to install the required packages.

For example, to copy RPMs to the container:

```
docker cp <packages or RPMs> <RHEL container name>:/
```

4. Export the proxy, if required, to download the packages from the RHEL hub.

5. Install `gnome-packagekit` and `xeyes` RPMs to support GUI installation of Documentum Server in the container.

   For example:

   ```
   $ yum install gnome-packagekit*
   $ yum install xeyes*
   ```

6. Install `rng-tools` to support random number generation on Linux.

   For example:

   ```
   $ yum install rng-tools
   ```

7. Install all the required RPMs.

   For example, run the following commands in the given sequence:

   ```
   $ yum install ksh
   $ yum install binutils*
   $ yum install elfutils-libelf-0.*
   $ yum install glibc-2.*
   $ yum install glibc-common-2.*
   $ yum install libaio-0.*
   $ yum install libgcc-4.*
   $ yum install libstdc++-4.*
   $ yum install make-3.*
   $ yum install compat-libcap1*
   $ yum install gcc-4.*
   $ yum install gcc-c++-4.*
   $ yum install libaio-devel-0.*
   $ yum install libstdc++-devel-4.*
   $ yum install unixODBC-2.*
   $ yum install unixODBC-devel-2.*
   $ yum install libXtst
   $ yum install sysstat*
   $ yum install csh*
   $ yum install hostname wget iputils
   $ yum install -y expect
   $ yum install -y tcl
   $ yum install -y unzip
   ```

   Note: Make sure that you also install all the dependent RPMs.

8. To make sure that dmdbtest does not fail during the loading of the shared libraries of libsasl2.so.2, while configuring the repository, perform the following steps in RHEL Docker container:

   ```
   root:~ # cd /usr/lib64
   root :/usr/lib64# ln -s libsasl2.so.3 libsasl2.so.2
   ```

9. Optional If you copied the packages or RPMs manually as described in Step 3, then delete the folder that contains the packages or RPMs in the container.

10. Install the Oracle client on the container to connect the Oracle database server. *Oracle Documentation* contains detailed information.

Note: Make sure that you perform all the additional steps described in *Oracle Documentation*.

11. After installing and configuring the Oracle client, remove the packages that is not required and save the changes to the image.

    For example:

    ```
    docker commit -m <base configuration of RHEL with Oracle client>
    <RHEL container name> documentumserver/rhelora/stateless/cs:base
    ```

2.3.3.3 Configuring Documentum Server in Red Hat Enterprise Linux/ Oracle Docker image

1. Download the Documentum Docker Compose including the `Dockerfile-RhelOra_Statelesscs` Docker template file from OpenText My Support.

2. Download the Documentum Server Docker related binaries from the FTP server and place them in a temporary local file system.

   Example for FTP server details:

   ```
   wget -nd -r --ftp-user=<user name> --ftp-password=<password>
   ftp://<IP address of the FTP server>/Builds/Documentum_Server/
   ${PRODUCT_MAJOR_VERSION}/${BUILD_NUMBER}
   /Server/linux_ora/*
   ```

   Note: Contact OpenText Global Technical Services to obtain the FTP server details.

3. Contact OpenText Global Technical Services to obtain the packages. Make sure to modify the path of downloaded directories or files in the `Dockerfile-RhelOra_Statelesscs` Docker file according to your platform.

   For example:

   ```
   COPY ./common/dctm_docker $DM_DOCKER_HOME
   COPY ./common/java.security_anon $JAVA_HOME/conf/security/java.security
   COPY configure-thumbnail-server.sh ${DM_DOCKER_HOME}/thumbserverbuild
   ```

4. Make sure that you modify the `Dockerfile-RhelOra_Statelesscs` Docker template file with proper entries.

5. Run the following command to configure the Documentum Server in Red Hat Enterprise Linux/Oracle Docker image.

   For example:

   ```
   docker build --build-arg PRODUCT_MAJOR_VERSION=
   <Documentum Server release version number>
   --build-arg BUILD_NUMBER=<Build number>
   --build-arg INSTALL_OWNER_USER=dmadmin
   --build-arg THUMNAIL_BUILD_NUMBER=<Thumbnail build number>
   -f Dockerfile-RhelOraCS_Statelesscs
   -t documentumserver/rhelora/stateless/cs:<base image name>
   ```

2.3.4 Exporting environment variables for storing passwords to Docker

Passwords are provided as environment variables. Make sure that you provide valid password set on the environment variables before creating or restarting the containers. Otherwise, the deployment may fail.

Export the following environment variables on the shell manually before running the following Docker Compose command while creating or recreating the container:

```
docker-compose -f <compose file name> up -d
```

```
export APP_SERVER_PASSWORD=<web application server administrator password>
export INSTALL_OWNER_PASSWORD=<installation owner password>
export ROOT_PASSWORD=<root user password>
export DOCBASE_PASSWORD=<repository password> (only for stateless configuration)
export DATABASE_PASSWORD=<external database server administrator password>
export GLOBAL_REGISTRY_PASSWORD=<global registry password>
export AEK_PASSPHRASE=<aek passphrase>
```

Note: Make sure that follow password complexity rules for *APP_SERVER_PASSWORD*, *DOCBASE_PASSWORD*, *GLOBAL_REGISTRY_PASSWORD*, and *AEK_PASSPHRASE*. The *Documentum Server* chapter in *OpenText Platform and Platform Extensions Installation Guide* contains detailed information. If you want to change the default value of minimum length of password, you can change it using the *DM_CRYPTO_MIN_PASSWORD_LENGTH* environment variable in the *environment* category.

2.3.5 Deploying and configuring Documentum Server on Stateless images

1. Perform the following steps on the Docker base machine containing either Oracle Linux or Ubuntu operating system:

   a. Docker requires a 64-bit installation regardless of your Linux version and you should have the supported version of kernel.

      For example:

      ```
      $uname -r
      ```

   b. Install the Docker engine.

   c. Start the Docker daemon service.

2. Download the Docker image from OpenText Container Registry and copy to the Docker base machine. The Docker image contains the PostgreSQL client, Documentum Server, two connection brokers, and a repository in a single container.

3. Run the `docker images` command to verify if the Docker image is downloaded successfully and listed.

4. Provide all the required details in the `CS-Docker-Compose_Stateless.yml` compose file. Read the `Readme.txt` file in the directory of the compose file for description of fields and provide valid values for each variable.

5. Export the environment variables as described in "Exporting environment variables for storing passwords to Docker" on page 20.

6. Log in as a PostgreSQL user and start the PostgreSQL server.

   Use the following command for Red Hat Enterprise Linux:

```
bash:#/usr/pgsql-<supported PostgreSQL version number>
/bin/pg_ctl -D /var/lib/pgsql/<supported PostgreSQL version number>
/data/ start
```

   Use the following command for Oracle Linux:

```
/usr/pgsql-<supported PostgreSQL version number>
/bin/pg_ctl -D /var/lib/pgsql/<supported PostgreSQL version number>
/data/ start
```

   Use the following command for Ubuntu:

```
/usr/lib/postgresql/<supported PostgreSQL version number>
/bin/pg_ctl -D /var/lib/postgresql/<supported PostgreSQL version number>
/data/ start
```

7. Create the Documentum Server Docker container.

   For example:

```
docker-compose -f <compose file name> up -d
```

8. To support random number generator and to avoid issues while renaming the log files, run the following command with the root account in the container:

```
root:#rngd -b -r /dev/urandom -o /dev/random
```

9. To verify the deployment, check the logs at `/opt/dctm_docker/logs/`*`hostname`*`.log` inside the container.

2.3.6 Deploying and configuring of Documentum Server HA on Docker

1. Install the supported version of Docker, Docker Compose file, and netshare plug-in in your host machine.

2. Start the Docker process from the service.

3. Start the Docker netshare plug-in if data or share are in external file system.

   For example:

```
./docker-volume-netshare --basedir=/var/lib/docker/volumes --verbose=true nfs
```

4. Share the `$DOCUMENTUM/data` and `$DOCUMENTUM/share` if the existing Documentum Server does not use the remote file system for data.

5. Provide all the required details in the `CS-Docker-Compose_Ha.yml` compose file. Read the `Readme.txt` file in the directory of the compose file for description of fields and provide valid values for each variable.

6. Export the environment variables as described in "Exporting environment variables for storing passwords to Docker" on page 20.

7. Create the Documentum Server HA Docker container.

   For example:

   ```
   docker-compose -f <HA compose file name> up -d
   ```

8. To verify the deployment, check the logs at `/opt/dctm_Docker/logs/`*`hostname`*`.log` inside the container.

2.3.7 Limitations

Installation owner is predefined and cannot be changed. The value is `dmadmin`.

2.3.8 Troubleshooting

There are no troubleshooting information for this release.

2.4 Deploying and configuring Independent Java Method Server on Docker

2.4.1 Prerequisites

1. Make sure that the Oracle Linux Docker image is deployed on the xfs file system and `ftype=1`.

2. Install the Docker Engine and Docker Compose file on your host machine.

   Docker Documentation contains detailed information.

3. Download the Docker image from OpenText Container Registry. Perform the following tasks:

   a. Log in to OpenText Container Registry using the following command format:

      ```
      docker login registry.opentext.com
      ```

      When prompted, provide your OpenText My Support login credentials.

   b. Download the Docker image using the following command format:

      ```
      docker pull registry.opentext.com/<image_name>:<image_tag>
      ```

   The following Docker image is available for download:

Image name	Image tag
dctm-ijms	23.2.0 or 23.2

   Note: Download the Docker Compose file from OpenText My Support.

4. Run the `docker images` command to verify if the Docker image is downloaded successfully and listed.

2.4.2 Deploying IJMS on Docker

You can run the `docker-compose -f docker-compose.yml up -d` command to start the container. Before running the `docker-compose` command, you must provide the details (IP address of the connection broker and other mandatory details) in `docker-compose.yml`.

1. Navigate to the location of the extracted `docker-compose.yml` file.

   For example, `/root/ijms`.

2. Run the `vi docker-compose.yml` command to open the `docker-compose.yml` file in the edit mode and review all the entries are correct.

3. Run the `docker-compose -f docker-compose.yml up -d` command to start the container and create volumes.

   Note: You must provide the correct location of the `docker-compose.yml` file in the `docker-compose -f <location of the docker-compose.yml file> up -d` command.

   For example, the location specified in the command in Step 3 assumes that the `docker-compose.yml` file is available in the current working directory.

4. Run the `docker volume ls` command to list the Docker volumes.

5. Run the `docker ps` command to view all the active Docker containers.

6. To verify the deployment, access the URL of IJMS container.

   For example, `http://<Docker_host>:<JMS_PORT>/DmMethods/servlet/DoMethod`.

7. Verify the logs inside the container at `/opt/dctm_Docker/logs/<hostname>.log` for the deployment details.

8. Restart the repositories.

Example 2-1: docker-compose.yml to create container and volumes

```
version: '2'
services:
ijms:
image: <ijms_image_name>:<tag>
network_mode: <Docker network name where Documentum Server container is deployed> (refer
the Network details section)
environment:
- GLOBAL_REGISTRY_DOCBASE=<Global repository name as in dfc.globalregistry.repository>
- GLOBAL_REGISTRY_USER==<Global repository user name as in dfc.globalregistry.username>
- GLOBAL_REGISTRY_PASSWORD==<Global repository password as in
dfc.globalregistry.password>
- DOCBROKER_HOST=<IP address of the Documentum Server connection broker>
- DOCBROKER_PORT=<Port number of the connection broker>
- INSTALL_OWNER_USER=<Repository installation owner username to which this IJMS is
configured>
- INSTALL_OWNER_PASSWORD=<Repository installation owner password to which this IJMS is
configured>
- APP_SERVER_PASSWORD=<JBoss application server password>
- DOCKER_HOST= <IP address of the base machine or IP address of Docker host>
- DOCBASE_NAME=<Name of the repository to which this IJMS is configured>
- PRIMARY_LOG_LOCATION=<Repository result of file_system_path from dm_location where
object_name="log" query>
- JMS_PORT=<Port number on which this IJMS instance JBoss runs on>
- CONTENT_SERVER_NAME=ALL
hostname:
"<Unique hostname>"
container_name:
"<Unique container name
for example, ijmscontainer<JMS_PORT> ex ijmscontainer9180)>"
ports: (refer the Port details section)
- "<JMS_PORT>:9180"
- "<APP_SERVER_MGMNT_PORT>=9184"
- "<APP_SERVER_MGMNT_CONSOLE_PORT>=9185"
volumes: (refer the Volume details section)
-<container_name>_log:/opt/dctm_Docker/jms/
<tomcatversion>/server/DctmServer_MethodServerHA1/log
-<container_name>_mdserver_conf:/opt/dctm_Docker/mdserver_conf
volumes:
<container_name>_log:
<container_name>_mdserver_conf:
```

2.4.2.1 Network details

The *network_mode* variable is used to deploy the IJMS instance on the same network where your Documentum Server is deployed. This is for limiting the accessibility of other resources to connect to the Documentum Server container.

To obtain the Docker network name, inspect the Documentum Server container using the following command format:

```
docker inspect <Documentum Server container name>
```

In the result of the command, the Docker network name is specified in the *Networks* section.

2.4.2.2 Port details

The IJMS instance is accessed or available through the *JMS_PORT* and you must provide an unique port number for each deployment.

For example, if a repository (for example, `docbase1`) is already configured with IJMS (for example, `IJMS1`) on default port 9180, then the next IJMS configuration (for example, `IJMS2`) must not use the same port. You must change the *JMS_PORT* value to any other available port other than the default port 9180.

If you have given an unique port number for *JMS_PORT* as 9280, then it is resolved as `9280:9180` meaning that the container is running with port number `9180` is externally exposed as `9280`. You can use `docker-compose.yml` as a template to expose the port, as needed.

For example, if xCP requires *APP_SERVER_MGMNT_PORT*, it is exposed as *<APP_SERVER_MGMNT_PORT>*`=9184`.

Docker Documentation contains detailed information.

2.4.2.3 Volume details

Deployed host paths or named volumes are specified as sub-options to a service. The *<container_name>_log* volume is available at `/opt/dctm_Docker/jms/<Tomcat_Version>/server/DctmServer_MethodServerHA1/log`. Similarly, *<container_name>_mdserver_conf* is available at `/opt/dctm_Docker/mdserver_conf`.

2.4.3 Limitations

Installation owner is predefined and cannot be changed. The value is `dmadmin`.

2.4.4 Troubleshooting

There are no troubleshooting information for this release.

2.5 Deploying and configuring Documentum Administrator on Docker

2.5.1 Prerequisites

1. Install the Docker Engine and Docker Compose file on your host machine.

   Docker Documentation contains detailed information.

2. Download the Docker image from OpenText Container Registry. Perform the following tasks:

   a. Log in to OpenText Container Registry using the following command format:

      ```
      docker login registry.opentext.com
      ```

      When prompted, provide your OpenText My Support login credentials.

   b. Download the Docker image using the following command format:

      ```
      docker pull registry.opentext.com/<image_name>:<image_tag>
      ```

   The following Docker image is available for download:

Image name	Image tag
dctm-admin	23.2.0 or 23.2

   Note: Download the Docker Compose file from OpenText My Support.

3. Run the `docker images` command to verify if the Docker image is downloaded successfully and listed.

2.5.2 Deploying Documentum Administrator on Docker

You can run the `docker-compose -f ./statelessda_compose.yml up -d` command to start the container. Before running the `docker-compose` command, you must provide the details (IP address of the connection broker and other mandatory details) in `statelessda_compose.yml`.

1. Navigate to the location of the extracted `statelessda_compose.yml` file.

   For example, `/home/da`.

2. Run the `vi statelessda_compose.yml` command to open the `statelessda_compose.yml` file in the edit mode and perform the following tasks:

   a. Provide the image details.

   b. Provide the environment details such as *DA_EXT_CONF*, *PREFERPASS*, *PRESETPASS*, *OTDS_PROPERTIES*, *APP_PROPERTIES*, *DFC_PROPERTIES*, name of the container, ports, volumes, and so on.

      Note: Perform the following tasks in `dfc.properties` and save the file.

      - Add *`dfc.session.allow_trusted_login`* in `dfc.properties` and set the value to `false`, if it does not exist.
      - Add *`dfc.security.ssl.use_anonymous_cipher`* in `dfc.properties` and set the value to `true`, if it does not exist.

   c. Provide the location of the `dalogs` volume (for example, `/opt/tomcat/logs`) and the location of the `dacustom` volume (for example, `/opt/tomcat/webapps/da/custom`).

   The `docker images` command lists the repository and tag along with other details.

   Note: Documentum Administrator supports two locales. “Deploying and configuring Documentum Administrator on private cloud” on page 104 contains detailed information.

3. Run the `docker-compose -f ./statelessda_compose.yml up -d` command to start the container and create volumes.

   Note: You must provide the correct location of the `statelessda_compose.yml` file in the `docker-compose -f <location of the statelessda_compose.yml file> up` command.

   For example, the location specified in the command in Step 3 assumes that the `statelessda_compose.yml` file is available in the current working directory.

4. Run the `docker volume ls` command to list Docker volumes.

   Example output:

```
[root@oraclelinux ~]# docker volume ls
DRIVER  VOLUME NAME
local   dalogs
local   dacustom
```

5. Run the `docker ps` command to view all the active Docker containers.

6. To verify the deployment, access the URL of Documentum Administrator container.

   For example, `http://<HostIP>:8080/da`.

2.6 Deploying and configuring Branch Office Caching Services on Docker

1. Install the supported version of Docker and Docker Compose file in your host machine.

2. Download the Docker image from OpenText Container Registry. Perform the following tasks:

   a. Log in to OpenText Container Registry using the following command format:

      ```
      docker login registry.opentext.com
      ```

      When prompted, provide your OpenText My Support login credentials.

   b. Download the Docker image using the following command format:

      ```
      docker pull registry.opentext.com/<image_name>:<image_tag>
      ```

   The following Docker image is available for download:

Image name	Image tag
dctm-bocs	23.2.0 or 23.2

3. Provide all the required details in the `bocs_compose.yml` file. Read the description of every field and provide valid values for each variable.

4. Run the Docker Compose command.

   For example:

   ```
   docker-compose -f bocs_compose.yml up -d
   ```

5. To verify the installation, check `http://<dockerbaseip>:8086/bocs/servlet/ACS`. In addition, check the web application server log files.

   For example, the logs at `/opt/tomcat/logs/<container_name>.log` or the Tomcat application server log files or the Docker log files.

Note: Ignore the Protocol family unavailable error in the `install.log` file located at `/opt/bocs-docker/logs`.

2.7 Deploying and configuring Documentum Foundation Services on Docker

1. Install the supported version of Docker on your host machine.
2. Prepare the Documentum Foundation Services WAR file with the appropriate configuration files (`dfc.properties` and `log4j2.properties`) and place the WAR file in a temporary location of your local machine.

   Set the value of `dfc.data.dir` to `/opt/tomcat/work`.
3. Download the base Tomcat image from OpenText Container Registry. Perform the following tasks:

   a. Log in to OpenText Container Registry using the following command format:

   ```
   docker login registry.opentext.com
   ```

   When prompted, provide your OpenText My Support login credentials.

   b. Download the Docker image(s) using the following command format:

   ```
   docker pull registry.opentext.com/<image_name>:<image_tag>
   ```

   The following Docker image is available for download:

Image name	Image tag
dctm-tomcat	23.2.0 or 23.2

4. Optional Upload the Docker image to your container registry using the following command format:

   ```
   docker push <image>
   ```
5. Configure the `dfs_config.conf` file with base Tomcat image information.
6. Start the container using the `dfs.sh` script.
7. Copy the DFS WAR file prepared in Step 2 to the `/opt/tomcat/webapps` path inside the container using the following command format:

   ```
   docker cp <path to the folder that contains the DFS WAR file>/<name of DFS WAR
   file> <docker-container-id>:</opt/tomcat/webapps>
   ```
8. To verify the deployment, access the URL of the Documentum Foundation Services container. For example, `http://<Docker base IP address>:8080/services/core/SchemaService?wsdl`.

Note: If you restart the container, you must perform Step 6 and Step 8 to start and verify the Documentum Foundation Services container.

The DFS WAR file is persisted even though the container is restarted multiple times. This is achieved by mounting the volume for the path `/opt/tomcat/webapps` in the DFS compose file.

2.8 Deploying and configuring Documentum Records Client on Docker

2.8.1 Prerequisites

1. Install the Docker Engine and Docker Compose file on your host machine.

   Docker Documentation contains detailed information.

2. Download the Docker image from OpenText Container Registry. Perform the following tasks:

   a. Log in to OpenText Container Registry using the following command format:

   ```
   docker login registry.opentext.com
   ```

   When prompted, provide your OpenText My Support login credentials.

   b. Download the Docker image using the following command format:

   ```
   docker pull registry.opentext.com/<image_name>:<image_tag>
   ```

   The following Docker images are available for download:

Image name	Image tag
dctm-records	23.2.0 or 23.2
dctm-records-darinstallation	23.2.0 or 23.2
dctm-rqm	23.2.0 or 23.2

   Note: Download the Docker Compose file from OpenText My Support.

3. Run the `docker images` command to verify if the Docker image is downloaded successfully and listed.

2.8.2 Deploying Documentum Records Client DAR files on Docker

You can run the `docker-compose up` command to start the container. If you want to use `docker_compose.yml`, copy the extracted Docker Compose file to a specific location on your host machine (for example, `/home/darinstallation`). Run the Docker Compose file related commands from this location only.

1. Navigate to the location of the extracted `docker-compose.yml` file. For example, `/home/darinstallation`.

2. Run the `vi docker-compose.yml` command to open the `docker-compose.yml` file in the edit mode.

3. Run the `docker-compose up` command to start the container and create volumes (see Example 2-4, "docker-compose.yml to create single container and volumes" on page 33).

   Note: If there is a delay in starting the Tomcat web application sever when you run the docker-compose up command, then log in as a root user and run the `rngd -b -r /dev/urandom -o /dev/random` command to perform seeding for random number generator.

4. Run the `docker volume ls` command to list docker volumes. An example result of the command:

```
[root@oraclelinux ~]# docker volume ls DRIVER VOLUME NAME
Local darinstallation_darinstaller_logs
local darinstallation_ext-conf
```

   Note: Since the Docker Compose file is available in the `/home/darinstallation` directory, the `docker-compose up` command creates two volumes (`darinstallation_darinstaller_logs` and `darinstallation_ext-conf`).

5. Run the `docker volume inspect <volume name of Docker>` command. An example result of the command:

```
[root@oraclelinux ~]# docker volume inspect ext-conf [
{
"Name": " ext-conf", "Driver": "local",
"Mountpoint": /var/lib/docker/volumes/darinstallation_ext-conf/_data", "Labels":
null,
"Scope": "local"
}
]
[root@oraclelinux ~]# docker volume inspect darinstaller_logs [
{
"Name": " darinstallation_darinstaller_logs", "Driver": "local",
"Mountpoint": "/var/lib/docker/volumes/darinstallation_darinstaller_logs/_data",
"Labels": null,
"Scope": "local"
}
]
```

6. Navigate to the volume using this command:

```
cd /var/lib/docker/volumes/darinstallation_ext-conf/_data
```

   and copy `dfc.properties` and `dfc.keystore` (if Documentum Server is SSL-enabled) to this location.

7. Run the `docker ps` command to view all the active Docker containers.

8. To verify the deployment, access the URL of Documentum Records Client container. Example URL: `http://<HostIP>:8030/records`

The DARs that will be installed are: `rps.dar`, `prm.dar`, `rm.dar`, `Forms.dar`. `RM-Default.dar`, `RM-Forms-Adaptor.dar`, `Rich_Media_Services.dar`, and `Transformation.dar`.

In the following example, the `/opt/external-configurations` folder is externalized and the volume name is **darinstaller_logs**.

Example 2-2: docker-compose.yml to create single container and volumes

```
version: '2'

services:
  darinstallation:
    image: artifactory.otxlab.net/bpdockerhub/dctm-records-darinstallation:<release-
version>
    environment:
      - DOCBASE_NAME=testenv
      - INSTALL_OWNER=Administrator
      - INSTALL_OWNER_PASSWORD=Password@123
      - DOCBROKERHOSTNAME=10.194.54.184
      - DARINSTALLATION_EXT_CONF=/opt/dctm
      - JAR_INSTALL_TYPE=update
      - SINGLEHELM=false
    volumes:
        - darinstaller_logs:/records_install/dar_logs
        - ext-conf:/opt/external-configurations

volumes:
  darinstaller_logs:
  ext-conf:
```

In the following example, you can use `docker-compose.yml` file to create two containers and volumes. This file can be modified to create as many containers as required.

Example 2-3: docker-compose.yml to create two containers and volumes

```
version: '2'

services:
  darinstallation:
    image: artifactory.otxlab.net/bpdockerhub/dctm-records-darinstallation:<release-
version>
    environment:
      - DOCBASE_NAME=testenv
      - INSTALL_OWNER=Administrator
      - INSTALL_OWNER_PASSWORD=Password@123
      - DOCBROKERHOSTNAME=10.194.54.184
      - DARINSTALLATION_EXT_CONF=/opt/dctm
      - JAR_INSTALL_TYPE=update
      - SINGLEHELM=false
    volumes:
        - darinstaller_logs:/records_install/dar_logs
        - ext-conf:/opt/external-configurations

volumes:
  darinstaller_logs:
  ext-conf:

version: '2'

services:
  darinstallation:
    image: artifactory.otxlab.net/bpdockerhub/dctm-records-darinstallation:<release-
version>
    environment:
      - DOCBASE_NAME=testenv
      - INSTALL_OWNER=Administrator
      - INSTALL_OWNER_PASSWORD=Password@123
      - DOCBROKERHOSTNAME=10.194.54.184
      - DARINSTALLATION_EXT_CONF=/opt/dctm
      - JAR_INSTALL_TYPE=update
      - SINGLEHELM=false

    volumes:
        - darinstaller_logs:/records_install/dar_logs
        - ext-conf:/opt/external-configurations

volumes:
  darinstaller_logs:
  ext-conf:
```

Note: Perform the following tasks in `dfc.properties` and save the file.

- Add *`dfc.session.allow_trusted_login`* in `dfc.properties` and set the value to `false`, if it does not exist.
- Add *`dfc.security.ssl.use_anonymous_cipher`* in `dfc.properties` and set the value to `true`, if it does not exist.

2.8.3 Deploying Documentum Records Client on Docker

You can run the `docker-compose up` command to start the container. If you want to use `statelessrecords_compose.yml`, copy the extracted Docker Compose file to a specific location on your host machine (for example, `/home/records`). Run the Docker Compose file related commands from this location only.

1. Navigate to the location of the extracted `docker-compose.yml` file. For example, `/home/records`.
2. Run the `vi docker-compose.yml` command to open the `docker-compose.yml` file in the edit mode.
3. Run the `docker-compose up` command to start the container and create volumes (see Example 2-4, “docker-compose.yml to create single container and volumes” on page 33).

   Note: If there is a delay in starting the Tomcat web application sever when you run the docker-compose up command, then log in as a root user and run the `rngd -b -r /dev/urandom -o /dev/random` command to perform seeding for random number generator.

4. Run the `docker volume ls` command to list docker volumes. An example result of the command:

```
[root@oraclelinux ~]# docker volume ls
DRIVER VOLUME NAME
local recordscustom
local recordslogs
```

   Note: Since the Docker Compose file is available in the `/home/records` directory, the `docker-compose up` command creates two volumes (`records_ext-conf` and `records_custom`).

5. Run the `docker volume inspect <volume name of Docker>` command. An example result of the command:

   ```
   [root@oraclelinux ~]# docker volume inspect records_ext-conf
   [
   {
   "Name": "recordslogs",
   "Driver": "local",
   "Mountpoint": "/var/lib/docker/volumes/recordslogs/_data",
   "Labels": null,
   "Scope": "local"
   }
   ]
   [root@oraclelinux ~]# docker volume inspect records_custom
   [
   {
   "Name": "recordscustom",
   "Driver": "local",
   "Mountpoint": "/var/lib/docker/volumes/recordscustom/_data",
   "Labels": null,
   "Scope": "local"
   }
   ]
   ```

6. Navigate to the volume using this command:

   ```
   cd /var/lib/docker/volumes/records_ext-conf/_data
   ```

   and copy `dfc.properties`, `dfc.keystore` (if Documentum Server is SSL-enabled), and `app.properties` to this location.

7. Run the `docker ps` command to view all the active Docker containers.

8. To verify the deployment, access the URL of Documentum Records Client container. Example URL: http://<HostIP>:8030/records

In the following example, the `external-configurations` folder available in the `/opt/tomcat/logs` directory is externalized and the volume name is `recordslogs`. Similarly, the `custom` folder available in the `opt/tomcat/records/custom` directory is externalized and the volume name is `recordscustom`.

Example 2-4: docker-compose.yml to create single container and volumes

```
version: '2'
services:
    recordsstateless:
      user: 1001:2001
      image: artifactory.otxlab.net/bpdockerhub/dctm-records:<release-version>
    environment:
      - INIT_RECORDS_WITH_DEFAULTS=false
      - RECORDS_EXT_CONF=/opt/tomcat/webapps/records/external-configurations
      - PREFERPASS=webtopUser@12345
      - PRESETPASS=webtopUser@12345
      - LOG4J_FORMAT_MSG_NO_LOOKUPS=true
      #- OTDSPROPERTIES=<OTDSPROPERTIES ::separated>
      #-
APPPROPERTIES=application.language.supported_locales.locale=[en_US,es_ES,ko_KR,de_DE,it_I
T,ar_EG]::application.language.default_locale=en_US
      # -
application.language.supported_locales.locale=[en_US,fr_FR,nl_NL,ja_JP,zh_CN,ru_RU,pt_BR,
es_ES,ko_KR,sv_SE,de_DE,it_IT,ar_EG;righttoleft|true]
      - application.language.supported_locales.locale=[en_US]
      #- DFCPROPERTIES=<DFCPROPERTIES ::separated>
    container_name:
      "records212container"
```

```
    ports:
     - "<APPSERVER_PORT>:8080"
    volumes:
      - ext-conf:/opt/tomcat/webapps/records/external-configurations
      - custom:/opt/tomcat/webapps/records/custom
      - wtp-tomcat-logs:/opt/tomcat/logs
      - wtp-documentum-logs:/home/recordsadmin/documentum
    privileged: true
volumes:
  ext-conf:
  custom:
  wtp-tomcat-logs:
  wtp-documentum-logs:
```

In the following example, you can use `docker-compose.yml` file to create two containers and volumes. This file can be modified to create as many containers as required.

Example 2-5: docker-compose.yml to create two containers and volumes

```
version: '2'
services:
  recordsstateless:
    user: 1001:2001
    image: artifactory.otxlab.net/bpdockerhub/dctm-records:<release-version>
    environment:
      - INIT_RECORDS_WITH_DEFAULTS=false
      - RECORDS_EXT_CONF=/opt/tomcat/webapps/records/external-configurations
      - PREFERPASS=webtopUser@12345
      - PRESETPASS=webtopUser@12345
      - LOG4J_FORMAT_MSG_NO_LOOKUPS=true
      #- OTDSPROPERTIES=<OTDSPROPERTIES ::separated>
      #-
APPPROPERTIES=application.language.supported_locales.locale=[en_US,es_ES,ko_KR,de_DE,it_I
T,ar_EG]::application.language.default_locale=en_US
      # -
application.language.supported_locales.locale=[en_US,fr_FR,nl_NL,ja_JP,zh_CN,ru_RU,pt_BR,
es_ES,ko_KR,sv_SE,de_DE,it_IT,ar_EG;righttoleft|true]
      - application.language.supported_locales.locale=[en_US]
      #- DFCPROPERTIES=<DFCPROPERTIES ::separated>
    container_name:
      "records212container"
    ports:
     - "<APPSERVER_PORT>:8080"
    volumes:
      - ext-conf:/opt/tomcat/webapps/records/external-configurations
      - custom:/opt/tomcat/webapps/records/custom
      - wtp-tomcat-logs:/opt/tomcat/logs
      - wtp-documentum-logs:/home/recordsadmin/documentum
    privileged: true
volumes:
  ext-conf:
  custom:
  wtp-tomcat-logs:
  wtp-documentum-logs:

version: '2'
services:
  recordsstateless:
    user: 1001:2001
    image: artifactory.otxlab.net/bpdockerhub/dctm-records:<release-version>
    environment:
      - INIT_RECORDS_WITH_DEFAULTS=false
      - RECORDS_EXT_CONF=/opt/tomcat/webapps/records/external-configurations
      - PREFERPASS=webtopUser@12345
      - PRESETPASS=webtopUser@12345
      - LOG4J_FORMAT_MSG_NO_LOOKUPS=true
      #- OTDSPROPERTIES=<OTDSPROPERTIES ::separated>
      #-
APPPROPERTIES=application.language.supported_locales.locale=[en_US,es_ES,ko_KR,de_DE,it_I
T,ar_EG]::application.language.default_locale=en_US
      # -
application.language.supported_locales.locale=[en_US,fr_FR,nl_NL,ja_JP,zh_CN,ru_RU,pt_BR,
es_ES,ko_KR,sv_SE,de_DE,it_IT,ar_EG;righttoleft|true]
      - application.language.supported_locales.locale=[en_US]
      #- DFCPROPERTIES=<DFCPROPERTIES ::separated>
    container_name:
      "records212container"
    ports:
     - "<APPSERVER_PORT>:8080"
    volumes:
      - ext-conf:/opt/tomcat/webapps/records/external-configurations
      - custom:/opt/tomcat/webapps/records/custom
      - wtp-tomcat-logs:/opt/tomcat/logs
      - wtp-documentum-logs:/home/recordsadmin/documentum
    privileged: true
volumes:
  ext-conf:
  custom:
  wtp-tomcat-logs:
  wtp-documentum-logs:
```

Note: Perform the following tasks in `dfc.properties` and save the file.

- Add *`dfc.session.allow_trusted_login`* in `dfc.properties` and set the value to `false`, if it does not exist.
- Add *`dfc.security.ssl.use_anonymous_cipher`* in `dfc.properties` and set the value to `true`, if it does not exist.

2.8.4 Using Up, Down, Start, and Stop options in Docker commands

- Run the `docker-compose up` command to create containers, volumes, and start containers. Use the up option only for the first time or if you want to initialize containers.
- Run the `docker-compose down` command to stop and remove the containers. Use the down option to terminate the containers.
- Run the `docker-compose start` command only if you want to gracefully start the containers. Also, run the `docker-compose stop` command only if you want to gracefully stop the containers. This ensures container ID and state do not change.

2.8.5 Deploying Records Queue Manager on Docker

Prerequisites

1. Install the Docker Engine and Docker Compose file on your host machine.

   Docker Documentation contains detailed information.

2. Download the Documentum Records Queue Manager (RQM) Docker image from OpenText Container Registry.

   a. Log in to OpenText Container Registry using the following command format:

   ```
   docker login registry.opentext.com
   ```

   When prompted, provide your OpenText My Support login credentials.

   b. Download the Docker image using the following command format:

   ```
   docker pull registry.opentext.com/<image_name>:<image_tag>
   ```

   The following Docker image is available for download:

Image name	Image tag
dctm-rqm	23.2.0 or 23.2

   Note: Download the Docker Compose file from OpenText My Support.

3. Run the `docker images` command to verify if the Docker image is downloaded successfully and listed.

You can run the `docker-compose up` command to start the container. If you want to use `docker-compose.yml`, copy the extracted Docker Compose file to a specific location on your host machine (for example, `/home/RQM`). Run the Docker Compose file related commands from this location only.

To deploy Records Queue Manager on Docker

1. Navigate to the location of the extracted `docker-compose.yml` file. For example, `/home/RQM`.

2. Run the `vi docker-compose.yml` command to open the `docker-compose.yml` file in the edit mode.

3. Run the `docker-compose up` command to start the container and create volumes (see Example 2-6, “docker-compose.yml to create single container and volumes” on page 37).

   Note: If there is a delay in starting the Tomcat web application sever when you run the `docker-compose up` command, then log in as a root user and run the `rngd -b -r /dev/urandom -o /dev/random` command to perform seeding for random number generator.

4. Run the `docker volume ls` command to list docker volumes. An example result of the command:

```
[root@oraclelinux ~]# docker volume ls
DRIVER VOLUME NAME
local RQMCTSConfig
local RQMJettyConfig
local RQMlogs
```

5. Run the `docker volume inspect <volume name of Docker>` command. An example result of the command:

```
[root@oraclelinux ~]# docker volume inspect RQMCTSConfig
[
{
"Name": "RQMCTSConfig",
"Driver": "local",
"Mountpoint": "/var/lib/docker/volumes/RQMCTSConfig/_data",
"Labels": null,
"Scope": "local"
}
]
[root@oraclelinux ~]# docker volume inspect RQMlogs
[
{
"Name": "RQMlogs",
"Driver": "local",
"Mountpoint": "/var/lib/docker/volumes/RQMlogs/_data",
"Labels": null,
"Scope": "local"
}
}
```

6. Navigate to the volume using this command:

```
cd /var/lib/docker/volumes/RQM_ext-conf/_data
```

   and copy `dfc.properties` and `dfc.keystore` (if Documentum Server is SSL-enabled) to this location.

7. Run the `docker ps` command to view all the active Docker containers.

8. To verify the configuration run the following DQL statement against the repository to verify the cts_instance_info:
   `select websrv_url, hostname from cts_instance_info`
   It should return the websrv_url and list `http://[host]:9096/cts/`.

In the following example, the RQM CTS configuration folder available in the `/root/dctm/CTS/config` directory is externalized and the volume name is `RQMCTSConfig`. Similarly, configuration logs available in the `/root/dctm/CTS/logs` directory is externalized and the volume name is `RQMlogs`. Jetty configuration folder available in the `/root/dctm/CTS/jetty/etc` directory is externalized and the volume name is `RQMJettyConfig`.

Example 2-6: docker-compose.yml to create single container and volumes

```
version: '2'
services:
  recordsqueuemanager:
    image: artifactory.otxlab.net/bpdockerhub/dctm-rqm:<release-version>
    environment:
      - RQM_INSTALLATION_DIR=/home/recordsadmin/dctm
      - RQM_HOSTNAME=10.194.54.19
      - RQM_ADMIN_PORT=9095
      - RQM_JETTY_PORT=9096
      - RQM_EXT_CONF=/opt/external-configurations
      - INSTALL_FILES=false
      - DOCBASE_NAME=testenv
      - INSTALL_OWNER_USER=Administrator
      - INSTALL_OWNER_PASSWORD=Password@123
      - RQM_DOMAIN=
      - RQM_DOCBASE_USER=Administrator
      - RQM_ADDITIONAL_DOMAINS=false
      - RQM_IS_PERFORMANCE_LOG_REPO=false
      - RQM_SYS_ADMIN_NAME=recordsadmin
      - RQM_SYS_ADMIN_PASS=password
      - GLOBAL_REGISTRY_REPOSITORY=testenv
      - BOF_REGISTRY_USER=dm_bof_registry
      - DOCBROKER_HOSTNAME=10.194.45.213
      - DOCBROKER_PORT=1489
      - BOF_REGISTRY_PASSWORD=Password@123
      - USE_CERTIFICATES=true
      - DFC_SSL_TRUSTSTORE=/opt/external-configurations/dfc.keystore
      - DFC_SSL_TRUSTSTORE_PASSWORD=AAAAEGZADhrsRexHk2iEo+4KCToM5MIhPCFkmU5kRQxi4lTV
      - DFC_SSL_USE_EXISTING_TRUSTSTORE=false
      - RQM_BOCS_SELECTED=
      - RQM_ALLOW_BOCS_TRANSFER=
      - RQM_PREFER_ACS_TRANSFER=
      - RQM_ALLOW_SURROGATE_TRANSFER=
      - RQM_DOMAIN_USER_LIST=
      - RQM_PROCESS_LOCAL_CONTENT_ONLY=
      - SINGLEHELM=false
    container_name:
      "RQMContainer"
    ports:
      - "9096:9096"
      - "9095:9095"
    volumes:
      - RQMCTSConfig:/home/recordsadmin/dctm/CTS/config
      - RQMlogs:/home/recordsadmin/dctm/CTS/logs
      - RQMJettyConfig:/home/recordsadmin/dctm/CTS/jetty/etc
      - RQM-ext-conf:/opt/external-configurations

volumes:
    RQMCTSConfig:
    RQMlogs:
    RQMJettyConfig:
    RQM-ext-conf:
```

Note: Perform the following tasks in `dfc.properties` and save the file.

- Add *`dfc.session.allow_trusted_login`* in `dfc.properties` and set the value to `false`, if it does not exist.
- Add *`dfc.security.ssl.use_anonymous_cipher`* in `dfc.properties` and set the value to `true`, if it does not exist.

2.9 Deploying and configuring REST Services on Docker

To deploy Documentum REST Services with non-SSL communication:

1. Install the supported version of Docker on your host machine.
2. Prepare the Documentum REST Services WAR file with the appropriate configuration files (`dfc.properties`, `log4j2.properties`, and `rest-api-runtime.properties`) and place the WAR file in a temporary location of your local machine.
3. Download the base Tomcat image from OpenText Container Registry. Perform the following tasks:

   a. Log in to OpenText Container Registry using the following command format:

   ```
   docker login registry.opentext.com
   ```

   When prompted, provide your OpenText My Support login credentials.

   b. Download the Docker image(s) using the following command format:

   ```
   docker pull registry.opentext.com/<image_name>:<image_tag>
   ```

   The following Docker image is available for download:

Image name	Image tag
dctm-tomcat	23.2.0 or 23.2

4. Optional Upload the Docker image to your container registry using the following command format:

   ```
   docker push <image>
   ```

5. Create a Docker volume using the following command format:

   ```
   docker volume create <name of volume>
   ```

6. Start the container using the following command format:

   ```
   docker run -ti -p 8080:8080 -d -v <name of volume>:/opt/tomcat/webapps
   <base_Tomcat_image_name>:<image_tag>
   ```

7. Copy the `dctm-rest.war` file prepared in Step 2 to the `/opt/tomcat/webapps` path inside the container using the following command format:

   ```
   docker cp <path to the folder that contains the dctm-rest.war file>/dctm-rest.war
   <docker-container-id>:</opt/tomcat/webapps>
   ```

8. To verify the deployment, access the URL of the Documentum REST Services container. For example, `http://localhost:8080/dctm-rest/services`.

Note: If you restart the container, you must perform Step 6 and Step 8 to start and verify the Documentum REST Services container.

To deploy Documentum REST Services with SSL communication:

1. Install the supported version of Docker on your host machine.

2. Prepare the Documentum REST Services WAR file with the appropriate configuration files (`dfc.properties`, `log4j2.properties`, and `rest-api-runtime.properties`) and place the WAR file in a temporary location of your local machine.

3. Download the base Tomcat image from OpenText Container Registry. Perform the following tasks:

   a. Log in to OpenText Container Registry using the following command format:

      ```
      docker login registry.opentext.com
      ```

      When prompted, provide your OpenText My Support login credentials.

   b. Download the Docker image(s) using the following command format:

      ```
      docker pull registry.opentext.com/<image_name>:<image_tag>
      ```

   The following Docker image is available for download:

Image name	Image tag
dctm-tomcat	23.2.0 or 23.2

4. Optional Upload the Docker image to your container registry using the following command format:

   ```
   docker push <image>
   ```

5. Create a Docker volume using the following command format:

   ```
   docker volume create <name of volume>
   ```

6. Start the container using the following command format:

   ```
   docker run -ti -p 8080:8080 -d -v <name of volume>:/opt/tomcat/webapps --entrypoint
   sh <base_Tomcat_image_name>:<image_tag>
   ```

7. Log in to the container created in Step 6 and update the `server.xml` configuration file of Tomcat with the SSL certificate information.

8. Start the Tomcat web application server.

9. To verify the deployment, access the URL of the Documentum REST Services container. For example, `https://localhost:8080/dctm-rest/services`.

Note: If you restart the container, you must perform the steps from Step 6 to Step 9 to start and verify the Documentum REST Services container.

2.10 Deploying and configuring Thumbnail Server on Docker

1. Install the supported version of Docker and Docker Compose file in your host machine.
2. Set up the external database server and remote file system.
3. Installation and configuration of Thumbnail Server is bundled with Documentum Server image. So, Thumbnail Server configuration settings needs to be done along with Documentum Server configuration. By default, Thumbnail Server configuration is set to `NO` with default port numbers. Change the value to `YES` to enable the Thumbnail Server configuration. For example:

```
CONFIGURE_THUMBNAIL_SERVER = YES
THUMBNAIL_SERVER_PORT = 8081
THUMBNAIL_SERVER_SSL_PORT = 8443
```

4. To verify the configuration, perform the following tasks:

   - Check the Thumbnail Server startup log at `$DOCUMENTUM/product/<product_version_folder>/thumbsrv/container/logs/catalina.out` inside the container.
   - Check the Thumbnail Server URL (`http://:8081/thumbsrv/getThumbnail?`) availability from any browser.

     By default, Thumbnail Server runs in HTTP mode on port 8081.

2.11 Deploying and configuring Content Connect on Docker

2.11.1 Prerequisites

1. Obtain and deploy the certificate authority (CA) certificates to deploy Content Connect and DCTM-REST on the HTTPS mode.
2. You must have a valid instance of PostgreSQL or Microsoft SQL Server database service.
3. To deploy Content Connect, the `contentconnect_<operating-system>_<release-version>.tar` and the `contentconnectdbinit_<operating-system>_<release-version>.tar` files are required. Each of the Docker `.tar` file consists of the following files when extracted:

   - `.env`: Environment configuration file to provide the database entries (Microsoft SQL or PostgreSQL).
   - Docker-compose`.yaml` file.

4. Install the Docker Engine and the Docker Compose file on your host machine.

The *Docker documentation* contains more information.

5. Download the Content Connect Docker image (Oracle Linux only) from OpenText Container Registry.

   a. Log in to OpenText Container Registry using the following command format:

      `docker login registry.opentext.com`

      When prompted, provide your OpenText My Support login credentials.

   b. Download the Docker image using the following command format:

      `docker pull registry.opentext.com/<image_name>:<image_tag>`

   The following Docker images are available for download:

Image name	Image tag
dctm-content-connect	23.2.0 or 23.2
dctm-content-connect-dbinit	23.2.0 or 23.2

6. Download the Docker Compose files and .env file for the Docker environment from OpenText My Support.

7. Run the `tar -xvf <packaged file>` command to extract the packaged file.

   For example:

```
tar -xvf contentconnect_<operating-system>_<release-version>.tar
```

   This extracts the Docker Compose (`docker-compose.yml`) and the `.env` file.

2.11.2 Deploying Content Connect on Docker

1. Download the `dctm-content-connect-dbinit` (for Oracle Linux) image from OpenText Container Registry.

2. Update the `.env` file with database endpoint details. Currently, Microsoft SQL and PostgreSQL servers are supported.

   For example:

```
db_host=NNN.NNN.NNN.NNN
db_port=NNNN
db_username=SA
db_password=SAPassword
db_dbname=CCDB
db_dbserver=mssql
```

   Where,

   - <db_host> is the IP address of the machine where the database is installed.
   - <db_port> is the port for database. The default port for Microsoft SQL is 1433 and PostgreSQL is 5432. However, the port can vary in different environments.

- <db_username> is the database user having privileges to create new database.
- <db_password> is the password for the database user.
- <db_dbserver> is the database server type. Set as `postgres` for PostgreSQL or `mssql` for Microsoft SQL.
- <db_name> is the database name to be created.

3. Use the Docker Compose file to initialize the container for Content Connect database, which is extracted from the `contentconnectdbinit_<operating-system>_<release-version>.tar` file.

4. Run the `docker-compose up` command to start the `contentconnectdbinit container`. This creates the database, table, and the Admin users in the provided database resource.

   Note: Admin can get the admin user details from the `contentconnectdbinit container` logs. These details are needed for login to Content Connect Admin console.

   Example 2-7: docker-compose.yml to create single container and initialize the database configurations

```
version: '2'

services:
    ccdbinitilize:
        image: "registry.opentext.com/dctm-content-connect-dbinit:<image tag>"
        container_name: "ccdbinitializecontainermssql2"
        env_file: .env
        command: sh -c "node initialize.js dbconfig ${db_host} ${db_port} $
{db_dbname} ${db_username} ${db_password} ${db_dbserver} && node initialize.js
database && node initialize.js adminuser"
```

5. To deploy Content Connect, download the `dctm-content-connect` (for Oracle Linux) file.

6. Download the `contentconnect` image from OpenText Container Registry.

7. Update the `.env` file with the relevant details, such as the port number, certificates path, database endpoint details, database username, password and database server type set to “postgres” for PostgreSQL or “mssql” for Microsoft SQL.

   For example:

```
        server_port=1607
        server_ssl_key_path=/usr/src/app/Content_Connect/sslkey.pem
        server_ssl_cert_path=/usr/src/app/Content_Connect/sslkey-cert.pem
        server_Allowed-Origins=*
        server_purge_old_logs=true
        db_host=NNN.NNN.NNN.NNN
        db_port=NNNN
        db_username=SA
        db_password=SAPassword
        db_dbname=CCDB
        db_dbserver=mssql
        logger_file_level=warn
        logger_file_filename=error.log
        logger_db_enabled=false
        logger_db_filename=db.log
        cc_port=8443
```

8. Run the `Docker Compose` file related commands from this location along with the `.env` file.

9. Update the `.env` file and the `Docker Compose` file related commands:

```
version: '2.0'

services:

       cc:

        image: "registry.opentext.com/dctm-content-connect:23.2.0000.0220"

        container_name: "cccontainer"

        env_file: .env

        ports:

            - "${cc_port}:${cc_port}"

            - "${server_port}:${server_port}"

        command: "/bin/sh","-c","node initialize.js dbconfig ${db_host} ${db_port} $
{db_dbname} ${db_username} ${db_password} ${db_dbserver} && node initialize.js
docker true && node initialize.js graphconfig ${clientId} ${clientSecret} $
{tenantId} && node initialize.js ccextension ${server_extension} && node
initialize.js updateprotocol ${server_protocol} && node UsertableUpgrade.js && node
CloudDbMigration.js &&  npm run-script gulp ContentConnect && node
postInstallScript.js && chmod -R 775 /usr/src/app/Content_Connect/dist &&  npm run-
script gulp cc_webserver"]

        volumes:

            -  c:/New_folder/certs:/usr/src/app/Content_Connect/certs
```

10. Use the `Docker Compose` file to initialize the container for Content Connect, which is extracted from the `contentconnect_<operating-system>_<release-version>.tar` file.

11. Run the `docker -compose up` command to start the `contentconnect` container. This step completes the deployment of Content Connect.

Example 2-8: docker-compose.yml to create single container and initialize the content connect node server

```
version: '2'

services:
    cc:
        image: "registry.opentext.com/dctm-content-connect:<image tag>"
        container_name: "contentconnectappcontainermssql2"
        env_file: .env
        ports:
            - "${cc_port}:${cc_port}"
            - "${server_port}:${server_port}"
        command: ["/bin/sh","-c","node initialize.js dbconfig ${db_host} ${db_port}
${db_dbname} ${db_username} ${db_password} ${db_dbserver} && node initialize.js
docker true && node CloudDbMigration.js && gulp ContentConnect && chmod -R 775 /usr/
src/app/Content_Connect/dist && gulp cc_webserver"]
        volumes:
            - C:/helm/certs:/usr/src/app/Content_Connect/certs
```

Note: You can keep the valid certificates in any location and modify the location details in the `docker-compose.yml` (Modify the `C:/helm/certs` path with the certificate location.)

12. Run the `docker ps` command to view all the active Docker containers.

13. To verify the deployment, access the URL of Content Connect container.

    For example:

    URL: `https://<content connect host>:<port>/<ccextension>/admin`

2.11.3 Deploying Content Connect on the client machine

Note: To access Content Connect Admin Console, you can retrieve the Admin credentials from the `contentconnectdbinit` deployment logs.

1. After the Content Connect Admin Console URL is up, add the endpoint details, and download the manifest file.

2. Use the manifest file to add the Content Connect add-in to the Office or Outlook application. For more information, see *OpenText Content Connect Installation and Administration Guide*.

Chapter 3

Documentum Platform and Platform Extensions applications on private cloud platform

Kubernetes is a portable, extensible open-source platform orchestration engine for automating deployment, scaling, and management of containerized applications. Kubernetes can be considered as a container platform, microservices platform, and portable cloud platform. It provides a container-centric management environment.

OpenShift is an enterprise-ready, subscription-based platform orchestration engine for automating deployment, scaling, and management of containerized applications. OpenShift is a container platform that provides capabilities to manage advanced clusters.

You can use the containerized deployment using the Documentum Docker images and Documentum Helm charts that are packaged with the release.

The product *Release Notes* document contains detailed information about the list of applications and its supported versions for private cloud platform.

3.1 Deploying and configuring Documentum Server on private cloud

3.1.1 Prerequisites

1. Download and configure the Docker application from the Docker website.

   Docker Documentation contains detailed information.

2. Download and configure the supported version of Helm package from the Helm website.

   The product *Release Notes* document contains detailed information about the supported versions.

   Helm Documentation contains detailed information.

3. Deploy a Docker registry using the following command format:

   ```
   docker run -d -p 5000:5000 --name <name of the registry> registry:2
   ```

   Docker Documentation contains detailed information.

4. Download and configure the Kubernetes application from the Kubernetes website.

   Kubernetes Documentation contains detailed information.

5. Download and configure the PostgreSQL database (server) from the PostgreSQL website.

> **Note:** The PostgreSQL database client is packaged with Documentum Server Docker image.

PostgreSQL Documentation contains detailed information.

6. Download the sample `values.yaml` file for PostgreSQL from the GitHub website. Open and provide the appropriate values for all the required variables.

7. Download the Documentum Server and the required Documentum application Docker images (Oracle Linux only) from OpenText Container Registry. Perform the following tasks:

   a. Log in to OpenText Container Registry using the following command format:

   ```
   docker login registry.opentext.com
   ```

   When prompted, provide your OpenText My Support login credentials.

   b. Download the Docker image(s) using the following command format:

   ```
   docker pull registry.opentext.com/<image_name>:<image_tag>
   ```

   The following Docker images are available for download:

Image name	Image tag
dctm-server	23.2.0 or 23.2
dctm-admin	23.2.0 or 23.2
dctm-dfs	23.2.0 or 23.2
dctm-records	23.2.0 or 23.2
dctm-records-darinstallation	23.2.0 or 23.2
dctm-rqm	23.2.0 or 23.2
dctm-rest	23.2.0 or 23.2
dctm-tomcat	23.2.0 or 23.2

   > **Note:** The `dctm-tomcat` image is required only for Documentum REST Services and Documentum Foundation Services. "Decoupling Documentum product image from base Tomcat image" on page 262 contains detailed information.

8. Upload the Docker image(s) to your container registry using the following command format:

   ```
   docker push <image>
   ```

9. Download the Helm charts available in the `documentum_server_<release-version>_kubernetes_helm_<operating-system>.tar` file from the OpenText My Support.

   > **Note:** The TAR file also contains Docker Compose scripts for reference along with Helm charts for the Oracle Linux operating system.

3.1.2 Deploying Documentum Server

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.
2. Download the Graylog Docker image from the Docker Hub website.

   Graylog Docker Documentation contains detailed information.
3. Upload the Graylog Docker image.

   Make sure that you configure the Graylog Docker image as per your requirement.
4. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and perform the following tasks:

   a. Provide the appropriate values for the common variables as described in “common-variables” on page 50.
   b. Provide the appropriate values for the variables in the `cs-secrets` category to pass them to your templates as described in “cs-secrets” on page 55.
   c. Provide the appropriate values for the variables in the `db` category to pass them to your templates as described in “db” on page 61.
   d. Provide the appropriate values for the variables in the `docbroker` category to pass them to your templates as described in “docbroker” on page 62.
   e. Provide the appropriate values for the variables in the `content-server` category to pass them to your templates as described in “content-server” on page 66.
   f. Provide the appropriate values for the variables in the `cs-logging-configMap` category to pass them to your templates as described in “cs-logging-configMap” on page 80.
   g. Make sure that the ingress controller is deployed in the cluster. Provide the appropriate values for the variables in the `dctm-ingress` category to pass them to your templates as described in “dctm-ingress” on page 84.
   h. Provide the appropriate values for the variables in the `cs-dfc-properties` category to pass them to your templates as described in “cs-dfc-properties” on page 85.
   i. Provide the appropriate values for the variables in the `otds` category to pass them to your templates as described in “otds” on page 89.
5. Optional Note that there are other variables that have their own default values specified in the following individual Helm charts:

   - `dctm-server/charts/cs-secrets/values.yaml`
   - `dctm-server/charts/db/values.yaml`
   - `dctm-server/charts/docbroker/values.yaml`
   - `dctm-server/charts/content-server/values.yaml`

- `dctm-server/charts/cs-logging-configMap/values.yaml`
- `dctm-server/charts/cs-dfc-properties/values.yaml`
- `dctm-server/charts/dctm-ingress/values.yaml`
- `dctm-server/charts/otds/values.yaml`

You can modify the default values in the individual Helm charts for deployment, if required.

Note: For the variables that exist both in the individual Helm charts and `dctm-server/values.yaml`, the value provided in `dctm-server/values.yaml` is used for deployment.

6. Deploy the Documentum Server Helm using the following command format:

```
helm install <release_name> <location where Helm charts are extracted>/dctm-server -
f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
platform>.yaml --namespace <name of namespace>
```

For example:

```
helm install dctm-server /opt/temp/Helm-charts/dctm-server -f /opt/temp/Helm-charts/
dctm-server/platforms/cfcr.yaml --namespace docu
```

Note: Make sure that you deploy Documentum Server in the same namespace where the PostgreSQL database is installed.

7. Verify the status of the deployment of Documentum Server Helm using the following command format:

```
helm status <release_name>
```

8. Verify the status of the deployment of the Documentum Server pod using the following command format:

```
kubectl describe pods <name of the pod>
```

Important

In a case, you have set the value of *`enable`* for any of the categories in `dctm-server/values.yaml` to `true` and completed the initial deployment. After this deployment, if you want to perform Helm upgrade (because of any changes to the values in the initial deployment), make sure that you retain the value to `true` during the upgrade process. Otherwise, the existing deployment of enabled categories gets deleted if set to `false` during the upgrade process.

Table 3-1: common-variables

Category	Name	Description
rwoStorage	*`rwo_storage_class`*	Name of the storage class with persistent volume claim (PVC) access mode as `ReadWriteOnce`. The default value is `trident-nfs`.

Category	Name	Description
rwmstorage	*rwm_storage_class*	Name of the storage class with PVC access mode as `ReadWriteMany`. The default value is `trident-nfs`.
env	*env_domain*	Name of the cluster space. The default value is `docu.svc.cluster.local`.
docbase	*docbase_name*	Name of the repository. The default value is `docbase1`.
csSecrets	*cs_secret_name*	Name of the secret configuration file. The format is `<sname>cs-secret-config`.
dctmotds	*otds_enabled*	Indicates if OTDS is enabled. The default value is `true`.
grayLog	*graylog_enabled*	Indicates if Graylog is enabled for use. The default value is `true`.
docbroker RepCount	*docbroker_replica _count*	Number of replica pods. The default value is `2`. **Note:** The value of this variable must not be changed after the successful deployment of Helm.
dbrService Name	*dbr_service_name*	Service name of connection broker. You must provide the fully qualified domain name (FQDN) and it must not be greater than 59 characters. The format is `<sname>dbr`.
openshift Enable	*openshift_enable*	Indicates if deployment in Red Hat OpenShift is enabled. The default value is `false`.
openshiftTls	*openshifttls _enable*	Indicates if deployment in Red Hat OpenShift is enabled with HTTPS configuration. The default value is `false`.
pullPolicy Type	*pull_policy_type*	Policy type to fetch the image. The default value is `IfNotPresent`.
pullSecret Name	*pull_secret_name*	Fetch the secret configuration file. The format is `<pullsecretsname>`.
liveness Enable	*liveness_enabled*	Indicates if the liveness probe is enabled. The default value is `true`.
image Repository	*image_repository*	Path of the repository. **Note:** Make sure that the same path of the repository is specified in `dctm-server/Chart.yaml` also.
imageTag	*image_tag*	Tag as a version-specific number.

Category	Name	Description
grayLogImage	*graylog_image*	Image details as per your Graylog server configuration.
graylogServer	*graylog_server*	Server details as per your Graylog server configuration.
graylogPort	*graylog_port*	Port reserved as per your Graylog server configuration.
useCertificates	*use_certificates*	Indicates if certificate-based communication is enabled. The default value is `false`.
aekLocation	*aek_location*	Path where AEK is available. The default value is `Local`. **Note:** If you want to use OpenText Key Mediation Service (OT KMS), make sure that you set the value to `Remote`. For more information about OT KMS, see “Configuring OT KMS with Documentum Server” on page 258.
kmsUrl	*kms_url*	URL to connect to OT KMS. The format is `https://<IP address or host of OT KMS server>:port`.
newrelicEnable	*newrelic_enabled*	Indicates if New Relic is enabled. **Note:** If you set the value of this variable to `true`, by default, the New Relic feature is enabled for the connection broker and Documentum Server pods. The default value is `false`.
newrelicProxy Host	*newrelic_proxy _host*	IP address of the proxy server.
newrelicProxy Port	*newrelic_proxy _port*	Port reserved for the New Relic server.
newrelicProxy Protocol	*newrelic_proxy _protocol*	Protocol to connect to the pod. The default value is `http`.
connectMode	*connect_mode*	Type of the connection mode. The default value is `dual`.
externalAccess Enaled	*external_access _enable*	Indicates if the external access of the connection broker is enabled. The default value is `false`.
fluentd	*fluentd_enabled*	Indicates if Fluentd is enabled for use. The default value is `false`.

Category	Name	Description
fluentd TcpPort	*fluentd_tcp_port*	Port on which Fluentd is listening for TCP connection. **Note:** This port is used for communication between DFC-based applications and Fluentd.
fluentd RestPort	*fluentd_rest_port*	Port on which Fluentd is listening for REST connection.
fluentd UdpPort	*fluentd_udp_port*	Port on which Fluentd is listening for UDP connection. **Note:** Documentum Server uses the UDP connection.
fluentd Image	*fluentd_image*	Name of Fluentd image.
eventLog Level	*event_log_level*	Log level for the DFC events. Set any value from `0` to `5` where, • 0 is for NO LOG • 1 is for ERROR • 2 is for WARN • 3 is for INFO • 4 is for DEBUG • 5 is for TRACE The default value is `0`.
dfcRPC tracing	*dfc_rpc_log_ enable*	Indicates if DFC RPC tracing log is enabled. The default value is `false`.
docbroker Port	*dbrport*	Port reserved for the connection broker.
database Host	*database_host*	Host name of the database. The format is `<sname>db-pg-0.<sname>db-pg.<namespace>.svc.cluster.local`.
database Port	*database_port*	Port reserved for the database.
dbrConfigmap Name	*dbr_configmap_name*	ConfigMap name of the connection broker. The format is `<sname>.dbr.configmap`.
kafka	*kafka_enabled*	Indicates if the Apache Kafka cluster deployment is enabled. The default value is `false`.
kafka _replicas	*kafka_replica_count*	Replica count of Kafkabroker and ZooKeeper services. The default value is `3`.

Category	Name	Description
namespace	*`namespace`*	Name of namespace used for deployment. The default value is `docu`.
zookeeper _port	*`zookeeper_port`*	Port reserved for the ZooKeeper service.
kafka_admin _user_name	*`kafka_admin _username`*	Administrator user name used to communicate among the replicas in the Apache Kafka cluster. The default value is `kafka-user`.
kafka_admin _user_passwor d	*`kafka_admin _userpassword`*	Administrator password used to communicate among the replicas in the Apache Kafka cluster. The default value is `kafka-password`.
kafka_topic _name	*`kafka_topic _name`*	Topic name provided while deploying the Apache Kafka cluster for storing the events.
kafka_image	*`kafka_image`*	Image name of Apache Kafka.
kafka_tag	*`kafka_tag`*	Tag of Apache Kafka as a version-specific number.
kafka_port	*`kafka_broker_port`*	Port reserved for the Kafka broker.
xcpRepository	*`xcp_repository`*	Path of xCP repository.
ccExtension	*`ccExtension`*	Extention name for Content Connect to be used in accessing Admin Console. `https://<content connect host>/ <ccextension>/admin`
cslogrotate	*`cslogrotate`*	Indicates if the usage of the logrotate tool is enabled. The default value is `false`.
tomcatbase image_name	*`tomcatbase image_name`*	Name of the Tomcat base image with the repository location.
tomcatbase image_tag	*`tomcatbase image_tag`*	Tag as a version-specific number.
tomcatbase _usecommonp vc	*`tomcatbase _usecommonpvc`*	Indicates to use the common PVC if the PVC already exists and need to be shared across Documentum products.
tomcatbase _commonpvcn ame	*`tomcatbase _commonpvcname`*	Name of the common PVC.
secrets_ Change	*`secrets_Change`*	Indicates if any changes to variables are made in the `cs-secrets` category. To reflect the changes, you must change the value to a different numeric value than the one provided for the previous deployment. When you change the value and then run the `helm upgrade` command, the Documentum Server and connection broker pods are recreated automatically.

Category	Name	Description
installOwner	*`installOwner`*	User name of the installation owner. The default value is `dmadmin`.

Note: The TCP, REST, and UDP ports are for internal use only.

After providing the appropriate values for the variables in `common-variables`, perform the next task in Step 4.b in "Deploying Documentum Server" on page 49.

Table 3-2: cs-secrets

Category	Name	Description
enable	*`enabled`*	Indicates if the secret configuration is enabled. The default value is `true`.
secret	*`name`*	Name of the secret configuration file. Specify the value provided for *`cs_secret_name`* in the `common-variables` category in Step 4.a.
docbroker	*`certificate`*	Certificate information of the connection broker. • *`password`*: Keystore password. • *`aekpassphrase`*: Passphrase to protect the AEK file. Specify the value provided for *`contentserver.aek.passphrase`*. • *`trustpassword`*: Trust store password. • *`pemCertPrivKey`*: Private key in the `PEM` format. Make sure that you change the default value and is consistent. • *`pemCertificate`*: Certificate in the `PEM` format. Make sure that you change the default value and is consistent. **Note:** If you have set the value of *`use_certificate`* to `true` in the `docbroker` category in Step 4.d to enable the certificate-based communication, a valid certificate and the private key in the `PEM` format must be provided for all inter-service communications. If you do not provide a valid certificate and the private key information, then the self-signed certificates are used.

Category	Name	Description
docbase	*password*	Password to connect to the repository. Make sure that you follow the password complexity rules. The *Documentum Server* chapter in *OpenText Platform and Platform Extensions Installation Guide* contains detailed information about password complexity rules. The default value is `Password@1234567890`. If you want to change the default value of minimum length of password, you can change it using the *DM_CRYPTO_MIN_PASSWORD_LENGTH* environment variable in the *extraEnv* category. If you change the repository owner password in the database, change the value of this variable to the new password during the Helm upgrade. Make sure that you set the value of *docbaseOwnerPasswordChange* to `true` in the `content-server` category.
	licenses	License details for the following optional modules: • Records Manager • Physical Records Manager • Federation Records Services • Retention Policy Services • Content Services for SnapLock • Storage aware devices • Trusted Content Services • High-Volume Server **Notes** • When you provide licenses for the Storage aware devices and Trusted Content Services modules, record the following capabilities: – New deployment: Encryption, compression, and de-duplication are enabled in your default filestore. – Upgrade: New default filestore with encryption, compression, and de-duplication is created. You can migrate the data from previous filestore to the new default filestore, if required. • When you provide licenses for the Storage aware devices and Trusted Content Services modules, you must enable the High-Volume Server license.

Category	Name	Description
contentserver	*`installOwner`*	• *`userName`*: User name of the installation owner. Specify the value provided for *`installOwner`* in the `common-variables` category in Step 4.a. **Note:** From the 20.4 release, when you log in to the pods, the installation owner is `dmadmin` instead of `root` (used in pre-20.4 version of pods). From the 23.2 release, when you log in to the pods, the default user can be any installation owner including `dmadmin`. • *`password`*: Password of the installation owner. The default value is `password`.
	`globalRegistry.password`	Password of the global registry. Make sure that you follow the password complexity rules. The *Documentum Server* chapter in *OpenText Platform and Platform Extensions Installation Guide* contains detailed information about password complexity rules. The default value is `Password@1234567890`. If you want to change the default value of minimum length of password, you can change it using the *`DM_CRYPTO_MIN_PASSWORD_LENGTH`* environment variable in the *`extraEnv`* category.
	`aek`	• *`algorithm`*: Algorithm for the encryption. Valid values are: – AES_128_CBC – AES_192_CBC – AES_256_CBC • *`oldPassphrase`*: Value of the older AEK passphrase (*`aek.passphrase`*) must be provided if you are upgrading the AEK key. The default value is null. • *`passphrase`*: Passphrase to protect the AEK file. Make sure that you follow the password complexity rules. The *Documentum Server* chapter in *OpenText Platform and Platform Extensions Installation Guide* contains detailed information about password complexity rules. The default value is `Password@1234567890`. If you want to change the default value of minimum length of password, you can change it using the *`DM_CRYPTO_MIN_PASSWORD_LENGTH`* environment variable in the *`extraEnv`* category.

Category	Name	Description
	certificate	Certificate information of the Documentum Server. • *password*: Keystore password. • *trustpassword*: Trust store password. • *pemCertPrivKey*: Private key in the `PEM` format. Make sure that you change the default value and is consistent. • *pemCertificate*: Certificate in the `PEM` format. Make sure that you change the default value and is consistent. **Note:** If you have set the value of *use_certificate* to `true` in the `docbroker` category in Step 4.d to enable the certificate-based communication, a valid certificate and the private key in the `PEM` format must be provided for all inter-service communications. If you do not provide a valid certificate and the private key information, then the self-signed certificates are used.
	install	• *appserver.admin.password*: Administrator password of the application server. Make sure that you follow the password complexity rules. The *Documentum Server* chapter in *OpenText Platform and Platform Extensions Installation Guide* contains detailed information about password complexity rules. The default value is `Password@1234567890`. If you want to change the default value of minimum length of password, you can change it using the *DM_CRYPTO_MIN_PASSWORD_LENGTH* environment variable in the *extraEnv* category. • *root.password*: Password of the installation root directory. The default value is `password`.
kms	*apiKey*	API key of OT KMS Server provided during the OT KMS configuration. **Note:** The OT KMS cloud service is available only for the new deployment.
database	*userName*	User name to access the database. The default value is `postgres`. For information about deploying Documentum Server with Microsoft Azure PaaS, see "Deploying Documentum Server with Azure PaaS" on page 182.
	password	Password to access the database. The default value is `password`.

Category	Name	Description
	`certificate`	Certificate information of the database. Make sure that you change the default value as per your requirement.
thumbnail Server	`appServer Password`	Password to access Thumbnail Server. The default value is `password@123`.
email	`smtpUser`	User name to access the SMTP server.
	`smtpPass`	Password of the user to access the SMTP server.
	`smtpSSL Certificate`	Certificate information of the SMTP server.
s3Store	`s3Store BaseUrl`	URL that Documentum Server uses to communicate with the Amazon S3 store. The URL format is `http://X.X.X.X/<BUCKET>`. The *OpenText Documentum Server Administration and Configuration Guide* contains detailed information.
	`s3Store CredentialID`	Name of the user accessing the S3 store. Use the S3 Tenant Owner. The default value is `nocredentials` if you want to use S3 role-based access. The *OpenText Documentum Server Administration and Configuration Guide* contains detailed information.
	`s3Store CredentialKEY`	Password of the user accessing the S3 store. Use the Object Access Key. The default value is `nocredentials` if you want to use S3 role-based access. The *OpenText Documentum Server Administration and Configuration Guide* contains detailed information.
	`certificate`	SSL certificate of S3 store. Make sure that you change the default value as per your requirement.
restStore	`restStore BaseUrl`	URL that Documentum Server uses to communicate with the REST store.
	`restStore CredentialID`	Name of the user accessing the REST store. Use the REST Tenant Owner. **Note:** The value for this variable is not required for Google Cloud store.
	`restStore CredentialKEY`	Password of the user accessing the REST store. Use the Object Access Key. **Note:** The value for this variable is not required for Google Cloud store.

Category	Name	Description
	azure certificate	SSL certificate of Azure Blob store. Make sure that you change the default value as per your requirement.
	gcpcertificate	SSL certificate of Google Cloud store. Make sure that you change the default value as per your requirement.
newrelic	*license_key*	License key information of New Relic.
gcpStore	*gcpStore_ credentials*	Contents of the JSON file generated on Google Cloud store.
graylog	*token*	API token of Graylog Sidecar.
ingress	*tlscrt*	Base64-encoded TLS certificate in one line without any carriage returns or line wrap.
	tlskey	Base64-encoded TLS key in one line without any carriage returns or line wrap.
openshift Tls	*enable*	Indicates if deployment in Red Hat OpenShift is enabled with HTTPS configuration. Specify the value provided for *`openshifttls_enable`* in the `common-variables` category in Step 4.a. • Kubernetes: Set the value of *`openshifttls_enable`* to `false`. • Red Hat OpenShift: Set the value of *`openshifttls_enable`* to `true` only if you want to configure the HTTPS mode for communication. This creates secrets of TLS type with the `<secret-name>-tls` name. Then, use this secret for the HTTPS configuration.

Note: Use OpenSSL or any other utilities to generate key and certificate.

For example, use the following command format to generate a private key file (`KEY_FILE.crt`) and a certificate (`CERT_FILE.crt`) for a given domain:

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout KEY_FILE.crt
-out CERT_FILE.crt -subj "/CN=<domain name>/O=<domain name>"
```

Then, convert the key and certificate to the Base64-encoded format.

If the host name (CN name) specified during the certificate request process does not match the host defined in your ingress route, your ingress controller displays a `Kubernetes Ingress Controller Fake Certificate` warning. Make sure that your certificate and ingress route host names are same.

After providing the appropriate values for the variables in `cs-secrets`, perform the next task in Step 4.c in “Deploying Documentum Server” on page 49.

Table 3-3: db

Category	Name	Description
database	*enabled*	Indicates if the database pod deployment is enabled. The default value is `true`. **Note:** To use an external database of your choice such as VM or database service to ignore the deployment of `dctm-server/charts/db/values.yaml`, set the value to `false`.
User Name	*serviceName*	Unique name for *sname*. For example, `dbforcluster1`db-pg.
images	*repository*	Path of the repository. Specify the value provided for *image_repository* in the `common-variables` category in Step 4.a.
	db.tag	Tag as a version-specific number.
	pullPolicy	Policy type to fetch the image. Specify the value provided for *pull_policy_type* in the `common-variables` category in Step 4.a.
	pullSecrets	Secret name to pull the secret configuration file. Specify the value provided for *pull_secret_name* in the `common-variables` category in Step 4.a.
secret	*name*	Name of the secret configuration file. Specify the value provided for *cs_secret_name* in the `common-variables` category in Step 4.a.
persistent Volume	*storageClass*	Storage class of the persistent volume (PV). Specify the value provided for *rwo_storage_class* in the `common-variables` category in Step 4.a.
configMap	*configName*	Unique name for *sname*. For example, `configforcluster1`-alter.
	volumeName	Unique name for *sname*. For example, `volumeforcluster1`-alter-volume.
alterParam	*max_connections*	Maximum number of concurrent connections to the database server.
	default_statistics_target	Sets the default statistics target for table columns. It is set without a column-specific target set through ALTER TABLE SET STATISTICS.
	maintenance_work_mem	Maximum amount of memory that must be used by maintenance operations.
	effective_cache_size	Effective size of the disk cache that is available to a single query.

Category	Name	Description
	wal_buffers	Amount of shared memory used for Write-Ahead Log (WAL) data that is not written to disk.
	work_mem	Base maximum amount of memory that needs to be used by a query operation before writing to temporary disk files.
rootsquash	*enable*	Indicates if the deployment in the root squash environment is enabled. The default value is `false`.
	ispg12orgreater	PostgreSQL database version (12 or later) in the root squash environment. The default value is `false`.
openshift	*enable*	Indicates if deployment in Red Hat OpenShift is enabled. Specify the value provided for *openshift_enable* in the `common-variables` category in Step 4.a. • Kubernetes: Set the value of *openshift_enable* to `false`. • Red Hat OpenShift: Set the value of *openshift_enable* to `true`.

After providing the appropriate values for the variables in `db`, perform the next task in Step 4.d in "Deploying Documentum Server" on page 49.

Table 3-4: docbroker

Category	Name	Description
enable	*enabled*	Indicates if the connection broker pod deployment is enabled. The default value is `true`.
User Name	*serviceName*	Service name of connection broker. Specify the value provided for *dbr_service_name* in the `common-variables` category in Step 4.a.
images	*repository*	Path of the repository. Specify the value provided for *imageRepository* in the `common-variables` category in Step 4.a.
	contentserver.tag	Tag as a version-specific number. Specify the value provided for *image_tag* in the `common-variables` category in Step 4.a.
	pullPolicy	Policy type to fetch the image. Specify the value provided for *pull_policy_type* in the `common-variables` category in Step 4.a.
	pullSecrets	Secret name to fetch the secret configuration file. Specify the value provided for *pull_secret_name* in the `common-variables` category in Step 4.a.

Category	Name	Description
openshift	*enable*	Indicates if deployment in Red Hat OpenShift is enabled. Specify the value provided for *openshift_enable* in the `common-variables` category in Step 4.a. • Kubernetes: Set the value of *openshift_enable* to `false`. • Red Hat OpenShift: Set the value of *openshift_enable* to `true`.
secret	*name*	Name of the secret configuration file. Specify the value provided for *cs_secret_name* in the `common-variables` category in Step 4.a.
docbroker	*replicaCount*	Number of replica pods. Specify the value provided for *docbroker_replica_count* in the `common-variables` category in Step 4.a. **Important** OpenText recommends two replica pods.
	connectMode	Type of the connection mode. Specify the value provided for *connect_mode* in the `common-variables` category in Step 4.a.
	liveness.enable	Indicates if the liveness probe is enabled. The default value is `true`. “Checking liveness of connection broker” on page 238 contains detailed information.
	secretsChange	Indicates if any changes to variables are made in the `cs-secrets` category. Specify the value provided for *secrets_Change* in the `common-variables` category in Step 4.a. See Step 4.a for the detailed description.
persistent Volume	*vctName*	Name of the volume claim template (VCT). For example, `dbr-vct`.
	dbrdataPVCName	Name of the PVC. For example, `certdbr-data-pvc`.
	storageClass	Storage class of the PV. Specify the value provided for *rwo_storage_class* in the `common-variables` category in Step 4.a.
	logVct StorageClass	Storage class of the VCT log. Specify the value provided for *rwo_storage_class* in the `common-variables` category in Step 4.a.
	pvcStorageClass	Indicates if a different storage class is used for PVC and VCT. The default value is null.

Category	Name	Description
	pvcStorageSize	Size of the PVC. The default value is `1Gi`. For Red Hat OpenShift, you must set the value to `256Gi`.
	existVolumePv	Details of existing volume, if any. Read the `Readme.txt` file for more information.
	awsEFS	Indicates if Amazon Elastic File System (EFS) is enabled. The default value is `false`.
	awsEFSCSI Driver	External storage provisioner (Amazon Elastic File System Container Storage Interface (CSI) Driver). The default value is `efs.csi.aws.com`.
	awsEFSCSI Handle	Volume ID of the EFS volume.
	awsEFSCSI ClaimPolicy	Reclaim policy of the EFS static provisioning persistent volumes (*persistentVolumeReclaimPolicy*). The default value is `Retain`.
graylog	*enable*	Indicates if Graylog is enabled for use. The default value is `true`. Specify the value provided for *graylog_enabled* in the `common-variables` category in Step 4.a.
	image	Image details as per your Graylog server configuration. Specify the value provided for *graylog_image* in the `common-variables` category in Step 4.a.
	server	Server details as per your Graylog server configuration. Specify the value provided for *graylog_server* in the `common-variables` category in Step 4.a.
	port	Port reserved as per your Graylog server configuration. Specify the value provided for *graylog_port* in the `common-variables` category in Step 4.a.
ExtDocbroker	*enable*	Indicates if the external connection broker is enabled for use. The default value is `false`. Specify the value provided for *external_access_enable* in the `common-variables` category in Step 4.a.
	extNativeNode Port	Make sure that there is no value specified for this variable.
	extSSLNode Port	Make sure that there is no value specified for this variable.

Category	Name	Description
	createExtService	External service for connection broker. The default value is `true`. **Note:** To externalize the connection broker in Red Hat OpenShift, set the value to `false`.
	useELB	Indicates if externalizing Documentum Server deployment in Amazon Web Services is enabled. The default value is `false`.
	enableLiveness	Indicates if liveness is enabled for the external connection broker.
	useLBAnnotations	Indicates if load balancer annotations is enabled for load balancer services used for externalizing Documentum Server. The default value is `false`.

	LBAnnotations	If the value of *useLBAnnotations* is set to `true`, then provide the required annotations to be enabled in load balancer services used for externalizing connection broker.
		
		For example:
		
		`useLBAnnotations: true`
		`LBAnnotations:`
		` networking.gke.io/load-balancer-type: "Internal"`

certificate	*use_certificate*	Indicates to enable certificate-based communication. Specify the value provided for *use_certificates* in the `common-variables` category in Step 4.a.
	aekname	Name of AEK key. The default value is `aek_name`.
	aeklocation	Path where AEK key is available. Specify the value provided for *aek_location* in the `common-variables` category in Step 4.a.
kms	*url*	URL to connect to OT KMS Server. Specify the value provided for *kms_url* in the `common-variables` category in Step 4.a.
	masterkey_id	OT KMS master key ID generated during the OT KMS configuration.
newrelic	• *enable* • *proxy_host* • *proxy_port* • *proxy_protocol* • *c_app_name*	"Integrating New Relic with Documentum Server" on page 229 contains detailed information.

After providing the appropriate values for the variables in `docbroker`, perform the next task in Step 4.e in "Deploying Documentum Server" on page 49.

Table 3-5: content-server

Category	Name	Description
enable	*enabled*	Indicates if the Documentum Server pod deployment is enabled. The default value is `true`.
User Name	*serviceName*	Unique name for *sname*. For example, `csforcluster1`dcs-pg.
images	*repository*	Path of the repository. The value for this variable is obtained from the value that you provided for *imageRepository* in the `common-variables` category in Step 4.a.
	contentserver.tag	Tag as a version-specific number. Specify the value provided for *image_tag* in the `common-variables` category in Step 4.a.
	pullPolicy	Policy type to fetch the image. Specify the value provided for *pull_policy_type* in the `common-variables` category in Step 4.a.
	pullSecrets	Secret name to fetch the secret configuration file. Specify the value provided for *pull_secret_name* in the `common-variables` category in Step 4.a.
secret	*name*	Name of the secret configuration file. Specify the value provided for *cs_secret_name* in the `common-variables` category in Step 4.a.
docbroker	*serviceName*	Service name of connection broker. Specify the value provided for *dbr_service_name* in the `common-variables` category in Step 4.a.
	clusterSpace	Name of the cluster space. Specify the value provided for *env_domain* in the `common-variables` category in Step 4.a.
docbase	*name*	Name of the repository. Specify the value provided for *docbase_name* in the `common-variables` category in Step 4.a.
	id	Unique ID of the repository. Make sure that the unique ID is any number from 1 to 16777215 and must not start with a zero (0).
	owner	Name of the repository owner. Specify the value provided for *docbase_name* in the `common-variables` category in Step 4.a. For information about deploying Documentum Server with Microsoft Azure PaaS, see “Deploying Documentum Server with Azure PaaS” on page 182.
	existing	Indicates to use an existing repository. The default value is `false`.

Category	Name	Description
	index	Name of the repository index if you want to use the existing repository. The format is `dm_<docbase name>_docbase`.
contentserver	*replicaCount*	Number of replica pods. The default value is `2`. ! **Important** OpenText recommends two replica pods.
	docbrokersCount	Number of connection brokers deployed. Specify the value provided for *docbroker_replica_count* in the `common-variables` category in Step 4.a.
	connectMode	Indicates to use the default table space of the database. Specify the value provided for *connect_mode* in the `common-variables` category in Step 4.a. Do NOT change this value.
	aek	• *name*: Name of AEK key. The default value is `aek_name`. • *location*: Path where AEK key is available. Specify the value provided for *aek_location* in the `common-variables` category in Step 4.a. Valid values are: – `Local` for password-based keys. – `Remote` for OT KMS-based keys.
	kms	• *url*: URL to connect to OT KMS. Specify the value provided for *ksm_url* in the `common-variables` category in Step 4.a. • *masterkey_id*: OT KMS master key ID generated during the OT KMS configuration.
	jmsProtocol	Protocol to connect to JMS.
	jmsVersion	Supported Tomcat version.

Category	Name	Description
	localeName	Locale of Documentum Server. The default value is `en`. Supported locales are: • `en`: English • `ar`: Arabic • `de`: German • `es`: Spanish • `fr`: French • `it`: Italian • `ja`: Japanese • `ko`: Korean • `nl`: Dutch • `pt`: Brazilian Portuguese • `ru`: Russian • `sv`: Swedish • `zh`: Simplified Chinese • `iw`: Hebrew **Note:** From the 22.1 release, you can configure multiple languages for Documentum Server. To select multiple languages, use language codes separated by comma. For example: `localeName = en, de, es` *OpenText Documentum Server Administration and Configuration Guide* contains detailed information about populating and publishing the data dictionary.
	fulltextEngine SSVCEnable	Indicates to change from Documentum Search query plug-in to Solr query plug-in in the Documentum Server setup. The default value is `false`.
	jmsMemSetting	Memory setting of JMS.
	secretsChange	Indicates if any changes to variables are made in the `cs-secrets` category. Specify the value provided for *`secrets_Change`* in the `common-variables` category in Step 4.a. See Step 4.a for the detailed description.

Category	Name	Description
	readiness	• *considerBPMForReadiness*: Indicates to consider BPM for the readiness probe. The default value is `false`. • *periodSeconds*: Frequency to perform the probe. The default value is `60` seconds.
	liveness	Indicates if the liveness probe is enabled. The default value is `true`. • *enable*: Indicates if the liveness probe is enabled. Specify the value provided for *liveness_enable* in the `common-variables` category in Step 4.a. • *initialDelaySeconds*: Time in seconds after the container or pod has started before the liveness probe is initiated. • *considerJMSForLiveness*: Indicates to consider JMS for the liveness probe. The default value is `true`. • *considerBPMForLiveness*: Indicates to consider BPM for the liveness probe. The default value is `false`. "Checking liveness of Documentum Server" on page 239 contains detailed information.
ExtCS	*enable*	Indicates if the access to external Documentum Server is enabled. The default value is `false`.
	tcp_route	IP address of the node. • Kubernetes: Use the `kubectl get node -o wide` command to obtain the IP of the node. The default value is `10.0.0.0`. • Red Hat OpenShift: Use the value specified in "Deploying Documentum Server" on page 169.
	nativeExtPort	Port reserved for externalizing the Documentum Server pod. • Kubernetes: Specify the reserved port value as per your requirement. • Red Hat OpenShift: Set the value to `80`.
	sslExtPort	Port reserved for externalizing the Documentum Server pod. • Kubernetes: Specify the reserved port value as per your requirement. • Red Hat OpenShift: Set the value to `81`.
	extDbrPort	Port reserved for the external connection broker. The default value is `1491`.

Category	Name	Description
	createExtService	Indicates if creating external service for Documentum Server is enabled. The default value is `true`. **Note:** To externalize Documentum Server in Red Hat OpenShift, set the value to `false`.
	useELB	Indicates if externalizing Documentum Server deployment in Amazon Web Services is enabled. The default value is `false`.
	useLBAnnotations	Indicates if load balancer annotations is enabled for load balancer services used for externalizing Documentum Server. The default value is `false`.

	LBAnnotations	If the value of *useLBAnnotations* is set to `true`, then provide the required annotations to be enabled in load balancer services used for externalizing Documentum Server.
		
		For example:
		
		`useLBAnnotations: true`
		`LBAnnotations:`
		` networking.gke.io/load-balancer-type: "Internal"`

database	*host*	Information you provided while configuring the database. See “Prerequisites” on page 47. **Note:** For the support of PostgreSQL installed on a VM: • *host*: IP address or host name of PostgreSQL on a VM. • *databaseService Name*: The default value is `MyPostgres`. You can change to any other user-defined name but only at the time of new deployment. • *port*: Port reserved for connecting to PostgreSQL on a VM.
	database ServiceName	
	port	
	sslEnabled	Indicates if SSL is enabled. The default value is `false`.
	paasEnv	Indicates if the Documentum Server pod deployment is enabled in PaaS. The default value is `false`. For information about deploying Documentum Server with Microsoft Azure PaaS, see “Deploying Documentum Server with Azure PaaS” on page 182.
	docbaseOwner PasswordChange	Indicates if repository owner password is changed in the database. The default value is `false`.

Category	Name	Description
thumbnail Server	*configure*	Indicates if Thumbnail Server pod configuration is enabled. The default value is `true`. Do NOT change this value.
	userMemArgs	User memory arguments. The default value is null.
	tnsProtocol	Protocol to connect to Thumbnail Server. Only HTTP protocol is supported. Do NOT change this value.
otds	*configureOTDS*	Configuration of OTDS. Specify the value provided for *otds_enabled* in the `common-variables` category in Step 4.a. Do NOT change this value.
	configNameOption	Type of configuration. The default value is HA. Do NOT change this value.
	otdsAPIsvc	Location where OTDS APIs are deployed. The default value is `otdsapi-highland.dev.bp-paas.otxlab.net`.
	ssl	Indicates if you want the OTDS URL in the HTTPS mode. The default value is `false`. For OTDS URL to work in the HTTPS mode, set the value to `true`. To verify if the URL works, validate *otds_rest_credential_url* in `otdsauth.properties` located at `$DM_JMS_HOME/webapps/OTDSAuthentication/WEB-INF/classes`.
	clientCapability	Expertise level of the user. The default value is `0`.
	userPrivileges	Privileges assigned to the user. The default value is `0`.
	userXPrivileges	Extended privileges assigned to the user. The default value is `0`.
	userRename Enabled	User name of dm_user is updated during OTDS synchronization. The default value is `F`.
	userRename UnlockLocked Object	Indicates to unlock all the objects checked out by the user to be renamed. The default value is `T`. This is applicable only if *userRenameEnabled* is set to `T`.
	groupRename Enabled	Indicates if the group name of dm_group can be updated during OTDS synchronization. The default value is `F`.

Category	Name	Description
	updateOTDScert onrestart	Indicates to update the OTDS certificate details on a restart. The default value is `false`.
	passauth_use _oauth2_token	Indicates to use OTDS token-based password authentication. The *OpenText Directory Services Integration with Documentum Server* chapter in the *OpenText Documentum Server Administration and Configuration Guide* contains detailed information.
	client_id	
	client_secret	
	otds_rest_oauth2 _url	
	synced_user_login _name	
	auto_cert_refresh	
	cert_jwks_url	
	is_hybrid	User partition in user name. The default value is `false` which means user name has partition. **Notes** • If you set the value to `true`, you must set the value of *synced_user_login_name* to `sAMAccountName`. • If you want to update the value of *synced_user_login_name* (`sAMAccountName`) in `otdsauth.properties`, you must enable *passauth_use_oauth2_token*.
custom	*scriptExecute*	Indicates if custom scripts execution is enabled. The default value is `false`. Set the value to `true` to install the DAR files or custom scripts. In addition, if you build as a compose image on Documentum Server image to install DAR files, custom scripts and so on, then place the script as `script_from_external.sh` in `${DM_DOCKER_HOME}/custom_script/` and make sure to start the composite image. During the installation process, `script_from_external.sh` is called and executed.
	scriptinPVC	Custom scripts in PVC.
	enableBPMPVC	Indicates if BPM PVC is enabled.
	scriptPVCname	Name of the custom script in PVC.
	PVCSubPath	Subpath of PVC.
	versions	Details of the versions.

Category	Name	Description
	markerFiles	Marker file names of client products. Provide them in any order separated by comma. Marker files availability are verified before Documentum Server copies the custom scripts.
	useInit Containers	Indicates to use one or more init containers, which are run before the application containers are started. The default value is `false`.
graylog	*enable*	Indicates if Graylog is enabled. Specify the value provided for *graylog_enabled* in the `common-variables` category in Step 4.a.
	image	Image details as per your Graylog server configuration. Specify the value provided for *graylog_image* in the `common-variables` category in Step 4.a.
	server	Server details as per your Graylog server configuration. Specify the value provided for *graylog_server* in the `common-variables` category in Step 4.a.
	port	Port reserved as per your Graylog server configuration. Specify the value provided for *graylog_port* in the `common-variables` category in Step 4.a.
persistent Volume	*storageClass*	Storage class for the PV. Specify the value provided for *rwm_storage_class* in the `common-variables` category in Step 4.a.
	size	Size of the PVC. The default value is `1Gi`. For Red Hat OpenShift, you must set the value to `256Gi`.
	awsEFS	Indicates if Amazon Elastic File System (EFS) is enabled. The default value is `false`.
	awsEFSCSI Driver	External storage provisioner (Amazon Elastic File System Container Storage Interface (CSI) Driver). The default value is `efs.csi.aws.com`.
	awsEFSCSI Handle	Volume ID of the EFS volume.
	awsEFSCSI ClaimPolicy	Reclaim policy of the EFS static provisioning persistent volumes (*persistentVolumeReclaimPolicy*). The default value is `Retain`.
	csdataPVCName	Name of the PVC.
	createPVC	Indicates to create PVC.
	existVolumePv	Details of existing volume, if any. Read the `Readme.txt` file for more information.

Category	Name	Description
volumeClaim Template	*vctName*	Name of the VCT. For example, `dcs-vct`.
	storageClass	Storage class of the VCT. Specify the value provided for *rwo_storage_class* in the `common-variables` category in Step 4.a.
	logVct StorageClass	Storage class of the VCT log. Specify the value provided for *rwo_storage_class* in the `common-variables` category in Step 4.a.
certificate	*use_certificate*	Indicates to use certificate-based communication. Specify the value provided for *use_certificates* in the `common-variables` category in Step 4.a. The default value is `false`.
	dbrserviceName	Service name of connection broker. Specify the value provided for *dbr_service_name* in the `common-variables` category in Step 4.a.
	customUpgrade	Indicates to use certificate-based communication in an upgraded environment. • *enable*: "Enabling certificate-based communication in upgraded 23.2 enviroment" on page 270 contains detailed information. • *nativeFirst*: Native mode of connection. The default value is `true`. Do NOT change this value.
dbrpersistent Volume	*dbrdataPVCName*	Name of the PVC.
	storageClass	Storage class for the PVC. Specify the value provided for *rwo_storage_class* in the `common-variables` category in Step 4.a.
	size	Size for the PVC. The default value is `1Gi`. For Red Hat OpenShift, you must set the value to `256Gi`.
email	*configure*	Indicates if email server configuration is enabled. The default value is `false`. This allows to externalize the setting of email server in dm_server_config in the Documentum Server pod.
	smtpServer	SMTP host name or IP address accessible to the Documentum Server pod.
	smtpPort	SMTP port accessible to the Documentum Server pod.
	smtpAuth	Indicates if SMTP authentication is enabled. The default value is `false`. If you set the value to true, provide the value for *email.smtpUser* and *email.smtpPass* in the `cs-secrets` category.

Category	Name	Description
	smtpSSL	Indicates if SMTP SSL communication is enabled. The default value is `false`. If you set the value to true, provide the value for *email.smtpSSLCertificate* in the `cs-secrets` category.
	emailAddress	Installation owner's email address.
	notification	Indicates if email notification is enabled. The default value is `false`.
s3Store	*enable*	Indicates if S3 store is enabled. The default value is `false`. To enable the S3 store, set the value to `true`.
	default	Indicates if S3 store is the default object store. The default value is `false`. To set S3 store as the default object store, set the value to `true`. **Notes** • If you want to set the S3 store as the default store, you must deploy the Documentum Ingress Helm before deploying the Documentum Server Helm. • If you set S3 store as the default object store, compression and de-duplication are not supported.
	• *name* • *proxyHost* • *proxyPort* • *proxy Protocal* • *noProxy* • *multiPartEnable* • *numThreads* • *isworm* • *vendor* • *region* • *enable_md5* • *enable _v4signing*	S3 store-related variables. **Note:** You must set the value for *proxyHost* and *proxyPort* if the Documentum Server host is behind the proxy server and S3 object store in the public cloud. The default value for both the variables is `noproxy`. *OpenText Documentum Server Administration and Configuration Guide* contains detailed information.

Category	Name	Description
	updateExisting Store	Indicates if you want to update the values for proxy host and proxy port to the existing public S3-compatible stores during an upgrade process. The default value is `false`. If you want to update the values for proxy host and proxy port to the existing public S3-compatible stores while upgrading from versions earlier to 21.3, you must set the value to `true` and provide the information for *storeListUpdate*, *proxyHostUpdate*, and *proxyPortUpdate*.
	storeListUpdate	List of existing public S3-compatible stores separated by comma. For example, `store1, store2`.
	proxyHostUpdate	Value of the proxy host.
	proxyPortUpdate	Value of the proxy port.
gcpStore	*enable*	Indicates if Google Cloud store is enabled. The default value is `false`. **Note:** If you want to enable Google Cloud store, you must set the value of both *gcpStore.enable* and *restStore.enable* to `true`.
restStore	• *enable* • *default* • *name* • *restStoreType* • *proxyHost* • *proxyPort* • *proxyProtocal* • *noProxy*	REST store-related variables. **Notes** • Only HTTP is supported. Do NOT change this value. • Creation of both the Azure Blob and Google Cloud stores at the same time is not possible in a cloud platform. *OpenText Documentum Server Administration and Configuration Guide* contains detailed information.
newrelic	• *enable* • *app_name* • *proxy_host* • *proxy_port* • *proxy_protocol* • *c_app_name*	"Integrating New Relic with Documentum Server" on page 229 contains detailed information.
fluentd_servi ce	*enable*	Indicates if Fluentd is enabled. The default value is `false`. Specify the value provided for *fluentd_enabled* in the `common-variables` category in Step 4.a.

Category	Name	Description
	image	Name of the Fluentd image. Specify the value provided for *fluentd_image* in the `common-variables` category in Step 4.a.
	imagePullPolicy	Policy type to fetch the Fluentd image. Specify the value provided for *pull_policy_type* in the `common-variables` category in Step 4.a.
	restartPolicy	Policy type to restart the pods. The default value is `Always`.
	TCPPort	Port on which Fluentd is listening for TCP connection. Specify the value provided for *fluentd_tcp_port* in the `common-variables` category in Step 4.a.
	RESTPort	Port on which Fluentd is listening for REST connection. Specify the value provided for *fluentd_rest_port* in the `common-variables` category in Step 4.a.
	UDPPort	Port on which Fluentd is listening for UDP connection. Specify the value provided for *fluentd_udp_port* in the `common-variables` category in Step 4.a.
	liveness.enable	Indicates if the liveness probe is enabled. The default value is `true`.

Category	Name	Description

ingress	*host*	Ingress controller domain name service value to update the ACS URL in the ACS configuration object and Thumbnail server URL. This value must be in the following format: `<INGRESS-HOSTNAME>.<CLUSTER-DOMAIN-NAME>`. The *host* and *clusterDomainName* values must be same as the one specified in the `dctm-ingress` category in the single All-In-One `dctm-server/values.yaml` Helm chart file. The application URLs must be in the following format: `<PROTOCOL>://<INGRESS-HOSTNAME>.<CLUSTER-DOMAIN-NAME>/<APPLICATION PATH>`.
		
		If you do not have the ingress controller domain name value at the time of Documentum Server pod deployment, leave the value blank and continue with the Documentum Server pod deployment. After you obtain the ingress controller domain name value, update the details for *ingress.host* in the `content-server` category. Then, run the Helm upgrade command to update the ingress host values in the ACS and Thumbnail Server URLs:
		
		Note: If you do not want to run the Helm upgrade command to update the ACS and Thumbnail Server URL values with the ingress host details, log in to the Documentum Server pod and run a script manually as follows:
		1. Connect to any of the Documentum Server pod.
		For example:
		`kubectl exec -it testlacdcs-pg-1 -c testlacdcs-pg bash`
		2. Navigate to the scripts folder using the following command format:
		`cd /opt/dctm_docker/scripts/`
		3. Run the command in the following command format:
		`./update_acs_tns_urls_with_ingress_host.sh <INGRESS-HOSTNAME>.<CLUSTER-DOMAIN-NAME> <INGRESS_PROTOCOL>`
		If the `<INGRESS_PROTOCOL>` value is not provided, then by default, the value used is `http`.

Category	Name	Description
	protocol	Type of communication mode. The default value is `http`. If you want to access the ACS Ingress URL in the HTTPS mode, set the value to `https`.
logrotate	*enable*	Indicates if the usage of the logrotate tool is enabled.
	interval	Frequency of the log rotation in minutes. The default value is `1440`.
logging	*cs*	Information provided for all these variables must be same as specified in the `cs-logging-configMap` category.
	jms	
	bpm	
	fluentdConf	
	logrotate_ configMap_name	
	d2	
extraEnv	*extraEnv*	Additional environment variables. You can use this category and its variables as per your requirement.
extraInit Containers	*extraInit Containers*	Additional init containers.

Note: To configure the log levels of the applications that are not part of the default JMS, perform the following tasks:

1. Provide the appropriate values for the *extraVolumes* and *extraVolumeMounts* variables to map the ConfigMap of the applications with the volume.

2. Remove the comment tag from the following sections:

```
extraVolumes:

- name: <application_name>-log4j2-properties-volume
  configMap:
    name: <APPLICATION_LOGGING_CONFIGMAP_NAME>
    items:
      - key: log4j
        path: log4j2.properties

extraVolumeMounts:

- mountPath: <location of log4j2.properties>
  name: <application_name>-log4j2-properties-volume
  subPath: log4j2.properties
```

   Do not change any other values.

After providing the appropriate values for the variables in `content-server`, perform the next task in Step 4.f in "Deploying Documentum Server" on page 49.

Table 3-6: cs-logging-configMap

Category	Name	Description
enabled	*enabled*	Indicates if the ConfigMap deployment is enabled. The default value is `true`.
User Name	*serviceName*	Name of the service. Specify the value provided for *sname* in the `content-server` category in “content-server” on page 66.
configMap	*cs_configMap_name*	ConfigMap for both the Documentum Server and Documentum Foundation Classes pods. The format is `<sname>cs-logging-configmap`.
	acs_configMap_name	ACS application ConfigMap of Java Method Server. The format is `<sname>acs-logging-configmap`.
	serverApps_configMap_name	ServerApps application ConfigMap of Java Method Server. The format is `<sname>serverapps-logging-configmap`.
	bpm_configMap_name	BPM application ConfigMap of Java Method Server. The format is `<sname>bpm-logging-configmap`.
	otdsauth_configMap_name	OTDS authentication ConfigMap of Java Method Server. The format is `<sname>otdsauth-logging-configmap`.
	dmotdsrest_configMap_name	OTDS REST ConfigMap of Java Method Server. The format is `<sname>dmotdsrest-logging-configmap`.
	saml_configMap_name	SAML ConfigMap of Java Method Server. The format is `<sname>saml-logging-configmap`.
	oauth_configMap_name	OAuth ConfigMap of Java Method Server. The format is `<sname>oauth-logging-configmap`.
	fluentd_configMap_name	ConfigMap name of Fluentd. The format is `<sname>fluentd-configmap`.
	fluentd_configMap_name	ConfigMap name of Fluentd. The format is `<sname>fluentd-configmap`.
	logrotate_configMap_name	ConfigMap name of logrotate. The format is `<sname>logrotate-configmap`.

Category	Name	Description
logging Configurat ion	*containerName*	Name of the Documentum Server container. The name can be either the name of complete deployment or specific containers.
dfcEvent HubTracin g	*enable*	Indicates if DFC RPC tracing log is enabled.. The default value is `false`.
	level	Log level for the DFC events. Set any value from `0` to `5` where 0 is for NO LOG, 1 is for ERROR, 2 is for WARN, 3 is for INFO, 4 is for DEBUG, and 5 is for TRACE.
dfcTracein g	*enable*	Indicates if the DFC tracing is enabled.. The default value is `false`.
docbase Trace	*enable*	Indicates if the Documentum Server tracing is enabled. The default value is `false`.
	options	All tracing options. The default values are `rpctrace` and `trace_authentication`. All the tracing options are supported. You can add multiple tracing options using comma as a separator. The generated trace information is recorded in the Documentum Server log file.
server Apps	*log4j*	Log levels for the ServerApps application. The valid values are `DEBUG`, `INFO`, `WARN`, `ERROR`, and `FATAL`. You can configure the required log4j log levels for the JMS deployed applications. • *rootLogLevel*: Log level for the `log4j2.properties` file. The default value is `WARN`. • *layout_type*: Layout type for the `log4j2.properties`. The default value is `PatternLayout`. • *layout_pattern*: Layout pattern for the `log4j2.properties` file. The format is `"%d{ABSOLUTE} %5p [%t] %c - %m%n"`. • *monitorInterval*: Monitor interval for the `log4j2.properties` file. The default value is `30` seconds.
otdsauth	*log4j*	Log levels for the OTDS authentication. The valid values are `DEBUG`, `INFO`, `WARN`, `ERROR`, and `FATAL`. You can configure the required log4j log levels for the JMS deployed applications. *rootLogLevel*: Log level for the `log4j2.properties` file. The default value is `INFO`.

Category	Name	Description
dmotdsrest	*log4j*	Log levels for the OTDS REST application. The valid values are DEBUG, INFO, WARN, ERROR, and FATAL. You can configure the required log4j log levels for the JMS deployed applications. • *rootLogLevel*: Log level for the log4j2.properties file. The default value is INFO. • *dmotdsrestLogLevel*: Log level for the OTDS REST application. The default value is INFO.
saml	*log4j*	Log levels for the SAML application. The valid values are DEBUG, INFO, WARN, ERROR, and FATAL. You can configure the required log4j log levels for the JMS deployed applications. *rootLogLevel*: Log level for the log4j2.properties file. The default value is INFO.
oauth	*log4j*	Log levels for the OAuth application. The valid values are DEBUG, INFO, WARN, ERROR, and FATAL. You can configure the required log4j log levels for the JMS deployed applications. *rootLogLevel*: Log level for the log4j2.properties file. The default value is INFO.
acs	*log4j*	Log levels for the ACS application. The valid values are DEBUG, INFO, WARN, ERROR, and FATAL. You can configure the required log4j log levels for the JMS deployed applications. • *rootLogLevel*: Log level for the log4j2.properties file. The default value is WARN. • *acsLogLevel*: Log level for the ACS application. The default value is WARN. • *atmosLogLevel*: Log level for the ATMOS store. The default value is WARN.

Category	Name	Description
bpm	*log4j*	Log levels for the BPM application. The valid values are `DEBUG`, `INFO`, `WARN`, `ERROR`, and `FATAL`. You can configure the required log4j log levels for the JMS deployed applications. • *rootLogLevel*: Log level for the `log4j2.properties` file. The default value is `WARN`. • *bpmLogLevel*: Log level for the BPM application. The default value is `WARN`. • *bpsLogLevel*: Log level for the BPS application. The default value is `WARN`. • *traceLogLevel*: Log level for the tracing options. The default value is `DEBUG`.
logrotate	*enable*	Indicates if the usage of the logrotate tool is enabled.
	configmap	Indicates the configuration that is used in the `logrotate.conf` file in the pod. The default values mentioned in the Helm can be used as is or modified as per your requirement with the same format.
fluentdCon f	*enable*	Indicates if Fluentd is enabled. The default value is `false`. Specify the value provided for *fluentd_enabled* in the `common-variables` category in Step 4.a.
	TCPPort	Port on which Fluentd is listening for TCP connection. Specify the value provided for *fluentd_tcp_port* in the `common-variables` category in Step 4.a.
	RESTPort	Port on which Fluentd is listening for REST connection. Specify the value provided for *fluentd_rest_port* in the `common-variables` category in Step 4.a.
	UDPPort	Port on which Fluentd is listening for UDP connection. Specify the value provided for *fluentd_udp_port* in the `common-variables` category in Step 4.a.
	kafkabroker	• *port*: Port reserved for the Kafka broker. Specify the value provided for *kafka_broker_port* in the `common-variables` category in Step 4.a. • *name*: Name of the Kafka broker instance. • *domain*: Domain where Apache Kafka cluster is deployed. Specify the value provided for *env_domain* in the `common-variables` category in Step 4.a. • *replica*: Number of replica pods. Specify the value provided for *kafka_replica_count* in the `common-variables` category in Step 4.a.
	kafkaTopic	Topic name provided while deploying the Apache Kafka cluster for storing the events.

Category	Name	Description
	kafkaUser	Administrator user name used to communicate among the replicas in the Apache Kafka cluster. Specify the value provided for *kafka_admin_name* in the `common-variables` category in Step 4.a.
	kafkaUsrPasswd	Administrator password used to communicate among the replicas in the Apache Kafka cluster. Specify the value provided for *kafka_admin_password* in the `common-variables` category in Step 4.a.
	compressionMode	Type of the compression mode. The default value is `gzip`.
	bufferingMode	Type of the buffering mode. The default value is `FILEBUFFER`. The two types of buffering mode supported are file buffer and memory buffer. If the value is set to `FILEBUFFER`, then file buffer is used. Otherwise, memory buffer is used.
	flushInterval	Time taken to flush the delayed events. The default value is `3` seconds. The value is considered only for the file buffer mode. For the memory buffer mode, the value is ignored.

Note: When you change the default value of any of the variables in `cs-logging-configMap` each time after the initial deployment of Documentum Server logging ConfigMap Helm, you must run the Helm upgrade command in the following command format:

```
helm upgrade <release_name> ./dctm-server -f <location where Helm charts are extracted>/
dctm-server/platforms/<cloud platform>.yaml --namespace <name of namespace>
```

After providing the appropriate values for the variables in `cs-logging-configMap`, perform the next task in Step 4.g in "Deploying Documentum Server" on page 49.

Table 3-7: dctm-ingress

Category	Name	Description
enable	*enabled*	Indicates if the ingress pod deployment is enabled. The default value is `false`.
Ingress Prefix	*ingressPrefix*	Unique ingress name.
ingress	*configureHost*	Indicates if the configuration of the host and cluster domain name is enabled. The default value is `true`. **Note:** If you set the value to `false`, you need not provide the value for `host` and `clusterDomainName`.
	host	Name of the ingress host.
	clusterDomainName	DNS name of the cluster.

Category	Name	Description
	class	Name of the ingress controller.
	annotations	Details of the annotations and its default values. You can modify the default values as per your requirement in the `dctm-server/platforms/<cloud platform>.yaml` file. Ingress Resource determines the controller that is utilized to serve traffic. Set the Ingress annotation to select the ingress controller. For example, this is set with an annotation, `kubernetes.io/ingress.class`, in the metadata section of the Ingress Resource.
services	*jmsService*, *jmsBase*, *acsService*, and so on	Details of the services and its default values. **Note:** Make sure that *jmsBase* is disabled (value is set to `false`) if you are using AWS Application Load Balancer (ALB) as the ingress controller.
openshift Tls	*enable*	Indicates if deployment in the Red Hat OpenShift platform is enabled with HTTPS configuration. Specify the value provided for *openshifttls_enable* in the `common-variables` category in Step 4.a. • Kubernetes: Set the value of *openshifttls_enable* to `false`. • Red Hat OpenShift: Set the value of *openshifttls_enable* to `true` only if you want to configure the HTTPS mode for communication. This creates secrets of TLS type with the `<secret-name>-tls` name. Then, use this secret for the HTTPS configuration.
tls	*enable*	Indicates if TLS is enabled. The default value is `false`.
	secretName	Name of the secret configuration file. Specify the value provided for *cs_secret_name* in the `common-variables` category in Step 4.a.

After providing the appropriate values for the variables in `dctm-ingress`, perform the next task in Step 4.h in "Deploying Documentum Server" on page 49.

Table 3-8: cs-dfc-properties

Category	Name	Description
enable	*enabled*	Indicates if `cs-dfc-properties` is enabled. The default value is `false`.
env	*domain*	Domain name of the environment. Specify the value provided for *env_domain* in the `common-variables` category in Step 4.a.

Category	Name	Description
User Name	*serviceName*	Service name of connection broker. Specify the value provided for *dbr_service_name* in the common-variables category in Step 4.a.
configMap	*namespace*	Domain name of the environment.
globalregistry	*repository*	Repository defined as a global registry. Specify the value provided for *docbase_name* in the common-variables category in Step 4.a.
	username	User name of the global registry. The default value is dm_bof_registry. Do NOT change this value.

Table 3-9: kafka-secrets

Category	Name	Description
enable	*enabled*	Indicates if the Apache Kafka cluster deployment is enabled. Specify the value provided for *kafka_enabled* in the common-variables category in Step 4.a.
secret	*name*	Name of the Kafka secret configuration file.
kafka_broker	*broker_admin_username*	Administrator user name used to communicate among the replicas in the Apache Kafka cluster. Specify the value provided for *kafka_admin_username* in the common-variables category in Step 4.a.
	broker_admin_password	Administrator password used to communicate among the replicas in the Apache Kafka cluster. Specify the value provided for *kafka_admin_password* in the common-variables category in Step 4.a.

Table 3-10: kafka-configMap

Category	Name	Description
enable	*enabled*	Indicates if the Apache Kafka ConfigMap deployment is enabled. Specify the value provided for *kafka_enabled* in the common-variables category in Step 4.a.
serviceName	*serviceName*	Service name of the Apache Kafka cluster.
configMap	*zookeeper_configMap_name*	ConfigMap name of the ZooKeeper service.
	kafka_broker_configMap_name	ConfigMap name of the Kafka broker service.

Table 3-11: zookeeper-statefulset

Category	Name	Description
enable	*enabled*	Indicates if ZooKeeper statefulset is enabled. Specify the value provided for *kafka_enabled* in the `common-variables` category in Step 4.a.
serviceName	*serviceName*	Service name of the ZooKeeper service.
configMaps	*zookeeper_configMap*	ConfigMap name of the ZooKeeper service.
tag	*tag*	Tag of Apache Kafka as a version-specific number. Specify the value provided for *kafka_tag* in the `common-variables` category in Step 4.a.
image	*image*	Image name of Apache Kafka. Specify the value provided for *kafka_image* in the `common-variables` category in Step 4.a.
replicas	*replicas*	Replica count of Kafkabroker and ZooKeeper services. Specify the value provided for *kafka_replica_count* in the `common-variables` category in Step 4.a.
zookeeper	*ports*	• *clientPort*: Port reserved for the ZooKeeper service. Specify the value provided for *zookeeper_port* in the `common-variables` category in Step 4.a. • *serverPort*: Internal use only. • *electionPort*: Internal use only.
	persistentVolume	• *servicedataPVCName*: Name of the PVC. • *namespace*: Name of the namespace used for deployment. Specify the value provided for *namespace* in the `common-variables` category in Step 4.a.
livenessprobe	*enable*	Indicates if the liveness probe is enabled. The default value is `true`.
volumeClaimTemplate	*vctname*	Name of the VCT.

Table 3-12: kafka-statefulset

Category	Name	Description
enable	*enabled*	Indicates if ZooKeeper statefulset is enabled. The default value is `false`. Specify the value provided for *kafka_enabled* in the `common-variables` category in Step 4.a.
serviceName	*serviceName*	Service name of the ZooKeeper service.

Category	Name	Description
tag	*tag*	Tag of Apache Kafka as a version-specific number. Specify the value provided for *kafka_tag* in the `common-variables` category in Step 4.a.
images	*pullsecrets*	Secret name to pull the secret configuration file. Specify the value provided for *pull_secret_name* in the `common-variables` category in Step 4.a.
replicas	*replicas*	Replica count of Kafkabroker and ZooKeeper services. Specify the value provided for *kafka_replica_count* in the `common-variables` category in Step 4.a.
secret	*name*	Name of the Kafka secret configuration file.
configMaps	*kafka_ broker_configMap*	ConfigMap name of the Kafka broker service.
zookeeper	*ports*	• *name*: Name of the ZooKeeper service. • *domain*: Domain where ZooKeeper is deployed. Specify the value provided for *env_domain* in the `common-variables` category in Step 4.a. • *replica*: Number of replica pods. Specify the value provided for *kafka_replica_count* in the `common-variables` category in Step 4.a. • *port*: Port reserved for the ZooKeeper service. Specify the value provided for *zookeeper_port* in the `common-variables` category in Step 4.a.
kafkabroker	*ports*	*kafkabrkrport*: Port reserved for the Kafka broker. Specify the value provided for *kafka_broker_port* in the `common-variables` category in Step 4.a.
	livenessprobe. enable	Indicates if the liveness probe is enabled. The default value is `true`. Specify the value provided for *liveness_enabled* in the `common-variables` category in Step 4.a.
	topic	• *create*: Indicates if new topics creation is enabled. The default value is `true`. • *topicname*: Name of the topic. • *partitions*: Number of partitions in a given topic.
	consumer	• *username*: Name of the user created with the consumer role to retrieve the logged messages. • *password*: Password of the user created with the consumer role to retrieve the logged messages.

Category	Name	Description
	persistent Volume	• *servicedataPVCName*: Name of the PVC. • *namespace*: Name of the namespace used for deployment. Specify the value provided for *namespace* in the `common-variables` category in Step 4.a.
ExtAccess	*enable*	Indicates if externalization of the Kafka broker service is enabled. The default value is `false`. If you want to consume the messages using the receiver SDK code snippet in "Retrieving messages" on page 247 from outside the cluster, set the value to `true`.
	external NodeIp	External IP address of any of the nodes which can be obtained using the `kubectl get nodes -o wide` command.
	external Port	External port of any of the nodes. **Note:** The port that you provide must not be available in the output when you run the following command: `kubectl get svc --all-namespaces -o go-template='{{range .items}}{{range.spec.ports}}{{if .nodePort}}{{.nodePort}}{{"\n"}}{{end}}{{end}}{{end}}'`
volumeClaim Template	*vctname*	Name of the VCT.

Table 3-13: otds

Category	Name	Description
enable	*enabled*	Indicates if OTDS authentication is enabled. The default value is `false`.
global	*otdsUse ReleaseName*	Indicates if the release name is used in the names of the objects. The default value is `false`. **Important** The value of this variable must not be changed for an upgrade process.
namespace	*namespace*	Name of the namespace used for deployment. Specify the value provided for *namespace* in the `common-variables` category in Step 4.a.
ingress	*enabled*	Indicates if the Kubernetes ingress is enabled. The default value is `true`.

Category	Name	Description
	secret	Name of the TLS/SSL Kubernetes secret for HTTPS connections. If you do not provide a value, then SSL is not activated in the ingress definition. You can create a secret in Kubernetes using the following command format: `kubectl create secret tls <secret name> --cert fullchain.pem --key privkey.pem`
	class	Ingress class used for annotations on different cloud platforms.
	annotations	Cloud platform-specific annotations of the ingress object.
	expose Individual Endpoints	Indicates if exposing the list of five endpoints is enabled. The default value is `false`.
	paths	Provides the list of endpoints that are associated with OTDS backend. Both the host and path must match to route the traffic to the desired service properly. The paths are exposed only if *exposeIndividualEndpoints* is `true`. The `"/otds-admin/"`, `"/otdstenant/"`, `"/otdsws/"`, `"/ot-authws/"`, and `"/otds-v2/"` endpoints are supported in OTDS. Otherwise, by default, only root `'/'` is exposed.

otdsws	*podLabels*	Lists additional pod labels to be applied.
		
		For example:
		
		`app.kubernetes.io/app_name: otdsws`
		`app.kubernetes.io/app_version: 22.2.0`

	enabled	Indicates if OTDS server deployment as a container is enabled in Kubernetes. The default value is `true`.
	serviceAccount Name	Name of the service account where pods are running. The default value is `default`.
	statefulSet	Controls if the OTDS server gets deployed as a statefulSet Kubernetes resource. The default value is `false`. This is helpful especially when you want OTDS to have static pod name in some on-premises scenarios.

Category	Name	Description
	ingress	• *enabled*: Indicates if the Kubernetes ingress is used. Specify the value provided for *otds_ingress_enabled* in *ingress.enabled*. • *secret*: Name of the TLS/SSL Kubernetes secret for HTTPS connections. • *prependPath*: Prepend a value to the standard OTDS server path when multiple services share a host name. Adding a value of `otds` results in the login URL in the following format: `<publicHostname>/otds/otdsws/login`
	serviceName	Name of the service for OTDS server.
	serviceType	Overrides the specification type for the otdsws service. If you do not set any value, Specify the value provided for *otds_ingress_enabled* in *ingress.enabled*. If ingress is enabled, the type is set to `ClusterIP`. If ingress is disabled, the type is set to `LoadBalancer`.
	carrierGradeNAT	Configure Tomcat to treat 100.64.0.0/10 addresses as internal for compatibility with environments that use Carrier-grade NAT.
	customSecretName	Name of an already existing secret object. This is useful when OTDS charts work as subcharts and parent chart create a single secret.
	replicas	Replica count for OTDS server. The default value is `1`.
	port	External port reserved for the OTDS Kubernetes service. The default value is `80`.
	publicHostname	FQDN or IP address of OTDS Kubernetes service. If the value is empty (`""`), then the host name is dynamically determined using the Kubernetes API.
	timeZone	Time zone of the Linux operating system within the container. The default value is `Etc/UTC`.
	allow DuplicateUsers	Indicates if OTDS allows the same user accounts to be created in multiple partitions. This applies to the users synchronized through eDirSync or SCIM.
	cryptKey	Indicates to use secure synchronized access to OpenDJ from the frontend instances. The value is a 16 character ASCII string in the Base64-encoded format.

Category	Name	Description
	additional JavaOpts	Additional Java parameters for OTDS which should be separated by space. For example, `-Dotds.repo.allowduplicateusers=true` enables the duplicated user in different partitions.
	enableBootstrap Config	Indicates to use `config.yml` in the otdsws chart directory to apply a specific set of configuration on the initial run when the database is populated. The default value is `false`.
	existingBootstrap Config	Content of the OTDS bootstrap configuration file. This overrides the configuration files in the chart directory.
	otadmin Password	Password of the `otadmin@otds.admin user` for OTDS. The default value is `otds`.

Category	Name	Description
	• *migration.enabled* • *migration.usingLegacyImage* • *migration.legacyImagePVC* • *migration.servicename* • *migration.servicePort* • *migration.opendjUri* • *migration.password*	From the OTDS 22.1.0 release, OpenDJ is not supported as a directory server. Instead, the existing or new PostgreSQL or SQL Server or Oracle or SAP database can be used. Indicates if the migration from OpenDJ must be attempted and defines how the migration must be performed. The three different types of migration are: • From a legacy OTDS deployment (for example, otds:21.3.0 images): Requires the value of *usingLegacyImage* set to `true` and that the information for *legacyImagePVC* is specified. In addition, the minimum memory requirements are increased such as the value of *resource.requests.memory* is set to `2Gi` and the value of *resource.limits.memory* is set to `3Gi`. • From an existing OTDS deployment in the same cluster: Requires the existing *servicename*, *servicePort*, and *password* information of OpenDJ is specified. • From an external OpenDJ such as a VM deployment: Requires the *opendjUri* (for example, `ldap://otds.domain.local:1389`) and *password* information is specified. **Notes** • If you set the value of *enabled* to `true`, then the default values are migrated from the previous default configuration. • If you set the value of *usingLegacyImage* to `true`, then the other non-relevant values are ignored. In addition, if *opendjUri* is specified, then the *servicename* and *servicePort* are ignored.
	migration.preUpgradeJob.enabled	Indicates if the one-step migration or deployment is enabled. The default value is `false`.

Category	Name	Description
	migration. preUpgradeJob. resources	• *requests.cpu*: Maximum number of CPUs for transaction. • *requests.memory*: Maximum usage of memory for transaction. • *limits.cpu*: Maximum number of allocated CPUs. • *limits.memory*: Maximum usage of allocated memory. Memory amounts in M, Mi, G or Gi are supported.
	migration. preUpgradeJob. jvmMemory	Sets the maximum amount of memory used by the OpenDJ migrate JVM. If a value is not set, the JVM memory sets to 75% of *preUpgradeJob*. Set the value to at least 2Gi as it splits the memory between OpenDJ and Tomcat. For other JVM memory settings, refer to the comments added for this variable.
	migration.image	Image parameters for the preupgrade job are *source*, *name*, and *tag*.

Category	Name	Description
	otdsdb	• *url*: URL to connect to the required database for OTDS. The default value is `jdbc:postgresql://postgres.domain.local:5432/otdsdb`. • *username*: Account for the database connection. • *password*: Password for the database account. • *enableCustomizedTruststore*: Allows to use customized trust store. For example, to validate a database server CA certificate for Oracle or Microsoft SQL. The certificate is imported into a user writable copy of the trust store which is used instead of the default Java trust store. **Note:** If you have any concerns due to the trust store being user writable and for a SSL connection with public signed root CA certificate, then do not set any value. • *sslDBRootCert*: Accepts the content of the database server custom root CA certificate by the `set-file` option. The certificate is stored in the `otds-certs` secret and mounted into pod at the mount point. The database root CA certificate file named as `dbRootCA.crt` is located at `/opt/config/certificates`. If *enableCustomizedTruststore* is enabled, the certificate is imported into trust store which is used to validate server for encrypted database connection. **Note:** For SSL connection with public signed root CA certificate, do not set any value. • *useDefaultSchema*: By default, OTDS uses a database schema named `otds`. If you set this option to `true`, the default schema assigned to the database user account is used instead. This is required on a shared Oracle database. • *automaticDatabaseCreation*: – *enabled*: Creates the required database or user and grant privileges automatically without any manual intervention. – *dbType*: Type of the database. The default value is `postgres`.

Category	Name	Description
		Note: Only the initialization of PostgreSQL database is supported. – *dbHost*: Host of the database server. – *dbAdmin*: Administrator account of the database. The default value is `postgres` for the PostgreSQL database. – *dbAdminPassword*: Password for the administrator account. – *dbName*: Database to be created for OTDS. The value provided for *otdsdb.username* and *otdsdb.password* is used for database user and password respectively. – *dbExtensions*: Extensions to be created for the OTDS database. For OTDS, the extension of `pg_trgm` is required. – *dbImage*: Database image and variables such as *source*, *name*, *tag*, and *pullPolicy* for running the database image. You must provide the value for *tag* that is used in the production environment.
	image	• *source*: Path of the repository. • *tag*: Tag as a version-specific number. • *pullPolicy*: Policy type to fetch the image. • *pullSecret*: Secret name to fetch the secret configuration file.
resources	*enabled*	Indicates if the resource requirements for OTDS is enabled. The default value is `true`.
	requests	• *cpu*: Maximum number of CPUs for transaction. • *memory*: Maximum usage of memory for transaction.
	limits	• *cpu*: Maximum number of allocated CPUs. • *memory*: Maximum usage of allocated memory.
newrelic	*NEW_RELIC_ LICENSE_KEY*	License key information of New Relic.
	NEW_RELIC_ APP_NAME	Application name of OTDS.
	NEW_RELIC_ LOG_FILE_NAME	OTDS log file name.
	NEW_RELIC_ LOG_LEVEL	Log level for OTDS.
	JAVA_OPTS	Location of the `newrelic.jar` file.

Perform the following additional tasks, as required:

1. The default ingress services are defined. By default, the ingress services is disabled. To enable the required ingress service(s), perform the following tasks:

   a. Set the value of *`enable`* of the required ingress service(s) to `true`.

   b. Provide the service name for *`serviceName`* of the required ingress service(s).

   c. Retain the default port value for *`servicePort`*.

2. To enable the HTTPS and TLS support in ingress, perform the following tasks:

   a. Set the value of *`enable`* of *`TLS`* to `true`.

   b. Specify the same secret name for *`secretName`* of *`TLS`* provided in the `cs-secrets` category in the single All-In-One `dctm-server/values.yaml` Helm chart file.

   c. Specify the appropriate values for *`tlscrt`* and *`tlskey`* in the single All-In-One `dctm-server/values.yaml` Helm chart file.

3. To verify and access the application using the ingress service, the application URL must be in the following format: `<PROTOCOL>://<INGRESS-HOSTNAME>.<CLUSTER-DOMAIN-NAME>/<APPLICATION PATH>`.

   For example, if the value of *`host`* is `prod` and the value of *`clusterDomainName`* is `docu.cfcr-lab.bp-paas.otxlab.net`, then use the `http://prod.docu.cfcr-lab.bp-paas.otxlab.net/ACS/servlet/ACS` URL to access the ACS application.

4. To verify the HTTPS and TLS support in ingress, perform the following tasks:

   a. Set the value of the ingress protocol to HTTPS in the single All-In-One `dctm-server/values.yaml` Helm chart file.

   b. Make sure that you have enabled the HTTPS and TLS support as described in Step 2.

   c. Deploy the Documentum Server Helm.

   d. After the Documentum Server Helm is deployed successfully, deploy the Documentum Ingress Helm, and verify if the ingress URL is accessible in the HTTPS mode.

After providing the appropriate values for the variables in `cs-dfc-properties` and completing the additional tasks as required, perform the tasks from Step 6 to Step 8 in “Deploying Documentum Server” on page 49 to complete the deployment and verification of the Documentum Server pod.

3.1.3 Limitations

- Installation path is predefined and cannot be changed. The value is `/opt/dctm/product/<product version>`.
- Upgrading of schema is not supported.
- When you run Tomcat on Linux, the console output is redirected to the `catalina.out` file. This is a Tomcat limitation.

3.1.4 Troubleshooting

Symptom	Cause	Fix
When you check the status of available pods using the `kubectl get pods` command, the READY value of one or more pod(s) reads as `1/2`.	One of the two containers in the specified pod(s) is down or unavailable.	Delete the pod using the `kubectl delete pods <name of the pod>` command. The pod is recreated automatically.
When you check the status of available deployed image, it results in the `Error: ImagePullBackOff` error.	Incorrect Helm deployment.	Delete the Helm deployment using the following command: `helm uninstall <release_name> --namespace <name of namespace>` Then, provide the correct image path and redeploy the Helm chart.

When you check the status of the upgrade process, it results in an error.	Unsuccessful upgrade.	Use the `describe` command to find the cause.
		For example:
		`kubectl describe pod <name of the pod>`
		Or
		`kubectl describe statefulset <name of the statefulset>`
		If any pod is down or not available, then the upgrade or rollback is not started at the pod level. Recreate the pod for the upgrade or rollback to start.

Symptom	Cause	Fix
All the Documentum Server pods gets recreated and the upgrade or rollback process is stuck for a long period than the usual time.	Kubernetes platform issue.	Log in to the primary Documentum Server pod and delete the `UpgradeInitiated<version>` file located in the `/opt/dctm/data` folder.
Upgrade process is stuck because of change in the installation owner.	Unsuccessful upgrade.	Go to the `/opt/dctm/kube/` folder, change the installation owner name that belongs to the previous deployment, and recreate the pod.

3.2 Deploying and configuring Independent Java Method Server on private cloud

3.2.1 Prerequisites

1. Perform the steps from Step 1 to Step 4 in "Prerequisites" on page 47 in "Deploying and configuring Documentum Server on private cloud" on page 47.

2. Deploy the Documentum Server pod as described in "Deploying and configuring Documentum Server on private cloud" on page 47.

3. Download the Docker image (Oracle Linux only) from OpenText Container Registry. Perform the following tasks:

   a. Log in to OpenText Container Registry using the following command format:

   ```
   docker login registry.opentext.com
   ```

   When prompted, provide your OpenText My Support login credentials.

   b. Download the Docker image using the following command format:

   ```
   docker pull registry.opentext.com/<image_name>:<image_tag>
   ```

   The following Docker image is available for download:

Image name	Image tag
dctm-ijms	23.2.0 or 23.2

4. Download the Helm charts available in the `ijms_<release-version>_kubernetes_helm_<operating-system>.tar` file from OpenText My Support.

   Note: The TAR file also contains Docker Compose scripts for reference along with Helm charts for the Oracle Linux operating system.

3.2.2 Deploying Independent Java Method Server

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.

2. Open the `ijms/values.yaml` file and provide the appropriate values for the variables to pass them to your templates as described in the following table:

Category	Name	Description
User Name	*`serviceName`*	Unique name for *`sname`*.
ijms	*`replicaCount`*	Number of replica pods. The default value is `2`. ! **Important** OpenText recommends two replica pods.
	`host`	Host of IJMS. The format is `<sname>ijms.<ingress domain name>`. For example, `test1ijms.docu.cfcr-lab.bp-paas.otxlab.net`.
	`ijmsConfiguringContentServer`	Name of a specific IJMS configuring Documentum Server or `ALL` to configure all Documentum Servers.
	`liveness`	• *`enable`*: Indicates if the liveness probe is enabled. The default value is `true`. • *`livenessScript`*: Path of the liveness probe script. • *`initialDelaySeconds`*: Time in seconds after the container or pod has started before the liveness probe is initiated. The default value is `960` seconds. • *`periodSeconds`*: Frequency to perform the probe. The default value is `120` seconds. • *`failureThreshold`*: Number of times the probe is allowed to fail. The default value is `3`. For example, if the value is set to `3`, then the pod is restarted after the probe fails for three times consecutively. "Checking liveness of Java Method Server" on page 240 contains detailed information.

Category	Name	Description
	readiness	• *readinessScript*: Path of the readiness probe script. • *initialDelaySeconds*: Time in seconds after the container or pod has started before the readiness probe is initiated. The default value is `600` seconds. • *periodSeconds*: Frequency to perform the probe. The default value is `120` seconds. • *failureThreshold*: Number of times the probe is allowed to fail. The default value is `3`. For example, if the value is set to `3`, then the readiness of the container is marked as `not ready` after the probe fails for three times consecutively.
images	*repository*	Registry host and port information of Docker images.
	ijms	• *name*: Name of the IJMS image. For example, `dctm-ijms`. • *tag*: Tag as a version-specific number.
docbase	*name*	Name of the repository.
secret	*name*	Specify the value provided for *name* in the `cs-secrets` category in the single All-In-One `dctm-server/values.yaml` Helm chart file.
docbroker	*serviceName*	Specify the value provided for *serviceName* in the `docbroker` category in the single All-In-One `dctm-server/values.yaml` Helm chart file.
	port	Port reserved for the connection broker. The default value is `1489`.
	clusterSpace	Name of the cluster space. The format is `<name of namespace>.svc.cluster.local`. For example, `docu.svc.cluster.local`.
	docbrokersCount	Number of connection brokers deployed. The default value is `2`.
global Repository	*globalRepositoryName*	Name of the global repository which has to be used as the global repository for the current repository. If it is set with some value then that is used as the global repository for this repository, otherwise the current repository is configured as the global repository.

Category	Name	Description
	globalRepositoryUser	User of the global repository. The default value is `dm_bof_registry`.
ports	*jmsport*	Port reserved for Java Method Server. The default value is `9180`.
	protocol	Protocol to connect to IJMS.
ingress	*enable*	Indicates if ingress is enabled. The default value is `false`. If you set the value to `true`, then one JMS configuration ID is created for `jms_config`.
	host	Ingress controller domain name service. This format is `<sname><INGRESS-HOSTNAME>.<CLUSTER-DOMAIN-NAME>`.
	port	Reserved for future use.
	protocol	Reserved for future use.
persistent Volume	*ijmsdataPVCName*	Name of the PVC. For example, `ijms-data-pvc`.
	pvcAccessModes	Access modes of the PVC. The default value is `ReadWriteMany`. Do NOT change this value.
	size	Storage size of the PV. The default value is `3Gi`.
	storageClass	AWS Storage class (RWX) for the PV.
	awsEFS	Indicates if Amazon Elastic File System (EFS) is enabled. The default value is `false`. If you want to deploy in AWS, set the value to `true`.
	awsEFSCSIDriver	External storage provisioner (Amazon Elastic File System Container Storage Interface (CSI) Driver). The default value is `efs.csi.aws.com`.
	awsEFSCSIHandle	Volume ID of the EFS volume. The format is `EFS file system ID::EFS access point ID1`. For example,`fs-1fc8901b::fsap-0e45ed0c4555fcd34`.
	existVolumePv	Details of existing volume, if any. Provide a unique user-defined name for PV. For example,`dctmijmspv`.
volume Claim Template	*vctName*	Name of the VCT. For example, `ijms-vct`.
	vctAccessModes	Access modes of the VCT. The default value is `ReadWriteOnce`.

Category	Name	Description
	size	Size of the VCT. The default value is `1Gi`.
	storageClass	Storage class of the VCT.
	logVctAccess Modes	Access modes of the VCT log. The default value is `ReadWriteOnce`.
	logVctSize	Size of the VCT log. The default value is `2Gi`.
	logVctStorage Class	Storage class of the VCT log.
service	*service*	• *type*: Type of the service. The default value is `ClusterIP`. • *port*: Port reserved for the service. The default value is `80`.
custom	*useInitContainers*	Indicates to use init containers. The default value is `false`.
extraInit Containers	*extraInit Containers*	Additional init containers.
extraEnv	*extraEnv*	Additional environment variables. You can use this category and its variables as per your requirement.

3. Deploy the Independent Java Method Server Helm using the following command format:

```
helm upgrade <release_name> <location where Helm charts are extracted>/ijms --
namespace <name of namespace>
```

For example:

```
helm upgrade ijms /opt/temp/Helm-charts/ijms --namespace docu
```

Note: Deploy IJMS in the same namespace where Documentum Server is deployed.

4. Verify the status of the deployment of Independent Java Method Server Helm using the following command format:

```
helm status <release_name>
```

5. Verify the status of the deployment of Independent Java Method Server pod using the following command format:

```
kubectl describe pods <name of the pod>
```

3.2.3 Limitations

There are no limitations for this release.

3.2.4 Troubleshooting

There are no troubleshooting information for this release.

3.3 Deploying and configuring Documentum Administrator on private cloud

3.3.1 Prerequisites

Perform all the steps as described in "Prerequisites" on page 47 in "Deploying and configuring Documentum Server on private cloud" on page 47 except for those steps related to the PostgreSQL database.

3.3.2 Deploying Documentum Administrator

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.

2. Perform the tasks mentioned from Step 2 to Step 5 in "Deploying Documentum Server" on page 49 in "Deploying and configuring Documentum Server on private cloud" on page 47.

3. Navigate to the `da` category in the single All-In-One `dctm-server/values.yaml` Helm chart file and provide the appropriate values for the variables to pass them to your templates as described in the following table:

Category	Name	Description
enabled	*`enabled`*	Indicates if Documentum Administrator deployment is enabled.
persistent VolumeClaim	*`storageClass`*	Storage class for the PVC. Specify the value provided for *`rwo_storage_class`* in the `common-variables` category in Step 4.a.
	`awsEFSCSIDriver`	External storage provisioner (Amazon Elastic File System Container Storage Interface (CSI) Driver). The default value is `efs.csi.aws.com`.
	`awsEFSCSIHandle`	Volume ID of the EFS volume.
service	*`name`*	Name of the service.
	`port`	Available port reserved for the service.

Category	Name	Description
ingress	*enable*	Indicates if the ingress service is enabled. This variable controls the creation of ingress service for your deployment. If `dctm-ingress` is used, then it is recommended to keep it disabled. Valid values are `true` and `false` and it is case-sensitive.
	clusterDomainName	DNS name of the cluster.
cs	*useCSDfcConfigMap*	Indicates to use the ConfigMap name used in Documentum Server. Valid values are `true` and `false`. • `true`: Use DFC properties from Documentum Server logging ConfigMap. • `false`: Use DFC properties from the *dfcProperties* environment variable. The default value is `true` and it is case-sensitive.
	configMapName	ConfigMap name used in Documentum Server. The format is `<sname>dbr.configmap`.
	csSecretConfigName	Name of the secret configuration file that was created while deploying the Documentum Server pod. Specify the value provided for *cs_secret_name* in the `common-variables` category in Step 4.a.
certificate	*useCertificate*	Indicates to connect to connection broker with SSL certificate. Specify the value provided for *use_certificates* in the `common-variables` category in Step 4.a.
	dbrServiceName	Service name of connection broker. Specify the value provided for *dbr_service_name* in the `common-variables` category in Step 4.a.
deployment	*name*	Indicates the deployment name.
	appName	Application name for Documentum Administrator.
	replicaCount	Number of replica pods. The default value is `1`. The maximum value is `10`.

Category	Name	Description
images	*da*	• *repository*: Path of the repository. Specify the value provided for *image_repository* in the `common-variables` category in Step 4.a. • *name*: Name of the Documentum Administrator image. • *tag*: Tag as a version-specific number. • *pullPolicy*: Policy type to fetch the image. • *imagepullSecrets*: Secret name to fetch the secret configuration file. The default value is null.
	graylog	• *enable*: Specify the value provided for *graylog_enabled* in the `common-variables` category in Step 4.a. Only if the value is set to `true`, the graylog side container is created. • *image*: Graylog image repository location. Specify the value provided for *graylog_image* in the `common-variables` category in Step 4.a. • *pullPolicy*: Policy type to fetch the image. Specify the value provided for *pull_policy_type* in the `common-variables` category in Step 4.a. • *name*: Name of the container. • *server*: Details of the Graylog server. Specify the value provided for *graylog_server* in the `common-variables` category in Step 4.a. • *port*: Port reserved for the Graylog server. Specify the value provided for *graylog_port* in the `common-variables` category in Step 4.a.

Category	Name	Description
containers.da	*probing*	• *failureThreshold*: Number of times the probe is allowed to fail. The default value is `2`. For example, if the value is set to `2`, then the readiness of the container is marked as `not ready` after the probe fails for two times consecutively. For the same example, for liveness, the pod is restarted after the probe fails for two times consecutively. • *successThreshold*: Minimum consecutive successes for the probe to be considered successful after having failed. The default value is `1`. • *timeoutSeconds*: Number of seconds after which the probe times out. The default value is `120` seconds. • *readinessProbe*: – *initialDelaySeconds*: Number of seconds after the Documentum Administrator pod has started before the probe is initiated. The default value is `600` seconds. – *periodSeconds*: Frequency to perform the probe. The default value is `300` seconds. • *livenessProbe*: "Checking liveness of Documentum Administrator" on page 240 contains detailed information.
otds	*enable*	Indicates if OTDS is enabled. Specify the value provided for *otds_enabled* in the `common-variables` category in Step 4.a.
	url	URL of OTDS.
	clientID	Details of client ID.
lmsgcustom Log	*enable*	Indicates if the custom log is enabled. The default value is `false`.
	loglocation	Path of the log file. The default value is `/opt/tomcat/logs`.

Category	Name	Description
wdkAppXml Config	*tagsnvalues*	Syntax for both English and Japanese language pack application. Documentum Administrator supports two locales. The *application.language.supported_locales.locale* property is used for the supported locales. The supported locales are en_US for U.S English and ja_JP for Japanese. By default, the *application.language.supported_locales.locale* property is set to [en_US] within the square brackets. If you want to add Japanese as a supported locale, add ja_JP separated by a comma within the square brackets. For example: *application.language.supported_locales.locale=[en_US, ja_JP]*. The *application.language.default_locale* property is used for the default locale. The default locale is en_US. If you want to change the default locale to Japanese, set the value of *application.language.default_locale* to ja_JP.
logging	*rootLoggerLevel*	Root log level for Documentum Administrator. The default value is WARN.
	consoleThresholdLevel	Console threshold level for Documentum Administrator. The default value is WARN.
	filename	Name of the log file.
	logFileSize	Maximum size of the log file.
	maxLogFiles	Maximum number of the log files.
newrelic	*newrelic*	"Integrating New Relic with Documentum Administrator" on page 230 contains detailed information.

4. Deploy the Documentum Server and Documentum Administrator Helm charts using the following command format:

```
helm install <release_name> <location where Helm charts are extracted>/dctm-server -
f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
platform>.yaml --namespace <name of namespace>
```

For example:

```
helm install dctm-server /opt/temp/Helm-charts/dctm-server -f /opt/temp/Helm-charts/
dctm-server/platforms/cfcr.yaml --namespace docu
```

5. Verify the status of the Documentum Server and Documentum Administrator Helm charts deployment using the following command format:

```
helm status <release_name>
```

6. Verify the status of the deployment of Documentum Server and Documentum Administrator pods using the following command format:

```
kubectl describe pods <name of the pod>
```

3.3.3 Limitations

Installation path is predefined and cannot be changed. The value is `/opt/tomcat/webapps/da`.

3.3.4 Troubleshooting

Symptom	Cause	Fix
When you check the status of available pods using the `kubectl get pods` command, the READY value of one or more pod(s) reads as 1/2.	One of the two containers in the specified pod(s) is down or unavailable.	Delete the pod using the following command: `kubectl delete pods <name of the pod command>.` After you delete, the pod is recreated automatically.
When you check the status of available deployed image, it results in the `Error: ImagePullBackOff` error.	Incorrect Helm deployment.	Delete the Helm deployment using the following command: `helm uninstall <release_name> --namespace <name of namespace>` Then, provide the correct image path and redeploy the Helm chart.

3.4 Deploying and configuring Documentum Foundation Services on private cloud

3.4.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 47 in “Deploying and configuring Documentum Server on private cloud” on page 47.

3.4.2 Deploying Documentum Foundation Services

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.

2. Perform the tasks mentioned from Step 2 to Step 5 in "Deploying Documentum Server" on page 49 in "Deploying and configuring Documentum Server on private cloud" on page 47.

3. Navigate to the `dfs` category in the single All-In-One `dctm-server/values.yaml` Helm chart file and provide the appropriate values for the variables to pass them to your templates as described in the following table:

Category	Name	Description
enabled	*`enabled`*	Indicates if Documentum Foundation Services deployment is enabled.
serviceName	*`serviceName`*	Name of the service. Specify the value provided for *`sname`* in the `content-server` category in "content-server" on page 66.
replicaCount	*`replicaCount`*	Number of replica pods. The default value is `1`. The maximum value is `10`. **Important** OpenText recommends two replica pods.
installOwner	*`installOwner`*	User name of the installation owner. Specify the value provided for *`installOwner`* in the `common-variables` category in Step 4.a.
imagePull Secrets	*`imagePullSecrets`*	Secret name to fetch the secret configuration file. Specify the value provided for *`pull_secret_name`* in the `common-variables` category in Step 4.a.
cs	*`useCSDfcConfigMap`*	Indicates to use DFC properties from ConfigMap. • `true`: Uses DFC properties from Documentum Server ConfigMap. • `false`: Uses DFC properties from ConfigMap created through the Documentum Foundation Services Helm. You must provide appropriate values for all the variables specified in `docbaseConnection` and `dfc` in the `dfs` category.
	`configMapName`	ConfigMap name used in Documentum Server. Specify the value provided for *`dbr_configmap_name`* in the `common-variables` category in Step 4.a.

Category	Name	Description
	csSecretConfigName	Name of the secret configuration file. Specify the value provided for *cs_secret_name* in the `common-variables` category in Step 4.a.
containers .initcontainer	*name*	Name of the Documentum Foundation Services init container. The default value is `dfsinit`.
	image. repository	Image repository information. Specify the value provided for *image_repository* in the `common-variables` category in Step 4.a.
	image.path	Name of the Documentum Foundation Services image.
	image.tag	Tag as a version-specific number.
	image. PullPolicy	Policy type to fetch the image. Specify the value provided for *pull_policy_type* in the `common-variables` category in Step 4.a.
containers .dfs	*name*	Name of the container. For example, `dfs`.
	kubernetes	Indicates if Kubernetes is enabled. The default value is `true`.
	image	• *repository*: Name of the repository. Specify the value provided for *image_repository* in the `common-variables` category in Step 4.a. • *path*: Name of the Tomcat base image with the repository location. Specify the value provided for *tomcatbaseimage_name* in the `common-variables` category in Step 4.a. • Tag as a version-specific number. Specify the value provided for *tomcatbaseimage_tag* in the `common-variables` category in Step 4.a. • *pullPolicy*: Policy type to fetch the image. Specify the value provided for *pull_policy_type* in the `common-variables` category in Step 4.a.
	containerHttpPort	HTTP port reserved for the Documentum Foundation Services container. The default value is `8080`.
	containerSslPort	SSL port reserved for the Documentum Foundation Services container. The default value is `8443`.

Category	Name	Description
	probing	• *url*: DFS URL that must be probed for readiness and liveness. • *port*: DFS port on which DFS server is running. • *failureThreshold*: Number of times the probe is allowed to fail. The default value is `2`. For example, if the value is set to `2`, then the readiness of the container is marked as `not ready` after the probe fails for two times consecutively. For the same example, for liveness, the pod is restarted after the probe fails for two times consecutively. • *successThreshold*: Minimum consecutive successes for the probe to be considered successful after having failed. The default value is `1`. • *timeoutSeconds*: Number of seconds after which the probe times out. The default value is `120` seconds. • *readinessProbe*: – *enable*: Indicates if the readiness probe is enabled. The default value is `true`. – *initialDelaySeconds*: Number of seconds after the Documentum Foundation Services pod has started before the probe is initiated. The default value is `60` seconds. – *periodSeconds*: Frequency to perform the probe. The default value is `60` seconds. • *livenessProbe*: "Checking liveness of Documentum Foundation Services" on page 240 contains detailed information.
fluentd	*enable*	Indicates if Fluentd is enabled. The default value is `false`.
	TCPPort	Port on which Fluentd is listening for TCP connection. Specify the value provided for *fluentd_tcp_port* in the `common-variables` category in Step 4.a.
	RESTPort	Port on which Fluentd is listening for REST connection. Specify the value provided for *fluentd_rest_port* in the `common-variables` category in Step 4.a.
	compression Mode	Indicates the compression mode. The default value is `gzip`.

Category	Name	Description
	bufferingMode	Indicates the buffering mode. The default value is `FILEBUFFER`.
	flushInterval	Time taken to flush the delayed events. The default value is `3` seconds.
	name	Name of Fluentd. For example, `fluentd`.
	pullPolicy	Policy type to fetch the image. Specify the value provided for *pull_policy_type* in the `common-variables` category in Step 4.a.
	restartPolicy	Policy type to restart the pods. For example, `Always`.
	fluentdLogFolder	Log folder of Fluentd. For example, `fluentd-logging`.
	fluentdConfigFolder	Configuration folder of Fluentd. For example, `fluentd-config-map`.
	servicedataPVCName	Name of the PVC. For example, `fluentd-data-pvc`.
	image	Path of the Fluentd image. Specify the value provided for *fluentd_image* in the `common-variables` category in Step 4.a.
	readiness.enable	Indicates if the readiness probe is enabled. The default value is `false`.

Category	Name	Description
	kafkabroker	• *port*: Port reserved for the Kafka broker. Specify the value provided for *kafka_broker_port* in the `common-variables` category in Step 4.a. • *name*: Name of the Kafka broker instance. • *domain*: Domain where Apache Kafka cluster is deployed. Specify the value provided for *env_domain* in the `common-variables` category in Step 4.a. • *replica*: Number of replica pods. Specify the value provided for *kafka_replica_count* in the `common-variables` category in Step 4.a. • *kafkaTopic*: Topic name provided while deploying the Apache Kafka cluster for storing the events. Specify the value provided for *kafka_topic* in the `common-variables` category in Step 4.a. • *kafkaUser*: Administrator user name used to communicate among the replicas in the Apache Kafka cluster. Specify the value provided for *kafka_admin_username* in the `common-variables` category in Step 4.a. • *kafkaUsr Passwd*: Administrator password used to communicate among the replicas in the Apache Kafka cluster. Specify the value provided for *kafka_admin_password* in the `common-variables` category in Step 4.a.
graylog	*enabled*	Indicates if Graylog is enabled for use. The default value is `false`. Specify the value provided for *graylog_enabled* in the `common-variables` category in Step 4.a. Only if the value is set to `true`, the graylog side container is created.
	name	Name of the Graylog server.
	image	Graylog image repository location. Specify the value provided for *graylog_image* in the `common-variables` category in Step 4.a.
	imagePullPolicy	Policy type to fetch the image. Specify the value provided for *pull_policy_type* in the `common-variables` category in Step 4.a.
	server	Details of Graylog server. Specify the value provided for *graylog_server* in the `common-variables` category in Step 4.a.

Category	Name	Description
	port	Available port reserved for the Graylog server. Specify the value provided for *graylog_port* in the `common-variables` category in Step 4.a.
	tags	Tags in brackets and as a comma-separated list to associate filebeat with. For example, `"[\"linux\",\"apache\"]"`.
	serviceToken	Service token of Graylog Sidecar.
tomcat	*username*	User name of Tomcat manager. The default value is `admin`.
	password	Password of Tomcat manager. The default value is `password`.
certificate	*use_certificate*	Indicates to enable certificate-based communication. The default value is `false`.
	dbrserviceName	Service name of connection broker. The format is `<sname>dbr`.
	dfcTrustStore Password	Trust store password of Documentum Foundation Classes. The default value is `password`.
dbrpersistent Volume	*dbrdataPVCName*	Name of the PVC. For example, `certdbr-data-pvc`.
dfc	**Note:** You must provide the appropriate values for this category only if you do not want to use the DFC properties of Documentum Server ConfigMap. If you want the DFS pod to use the values provided in this category for the `dfc.properties` file, then you must set the value of *cs.useCSDfcConfigMap* in the `dfs` category to `false`.	
	docbroker	Name of connection broker used in Documentum Server. The format is `<sname>.dbr`.
	port	Port reserved for the connection broker. Specify the value provided for *dbrport* in the `common-variables` category in Step 4.a.
	globalRegistry Repository	Repository defined as a global registry. Specify the value provided for *docbase_name* in the `common-variables` category in Step 4.a.
	globalRegistry Username	User name for the global registry user. Specify the value provided for *globalUsername* in the `cs-dfc-properties` category in Step 4.h.
	globalRegistry Password	Password for the global registry user. Specify the value provided for *globalRegistryPassword* in the `cs-secrets` category in Step 4.b.

Category	Name	Description
	connectionMode	Type of the connection mode. The default value is `try_native_first`.
	cryptoRepository	Repository defined as a global registry. Specify the value provided for *docbase_name* in the `common-variables` category in Step 4.a.
	dataDir	Data directory path of Documentum Foundation Classes. The default value is `/var/documentum`. **Note:** The data directory path need not be changed.
	client	• *should_use_enduserinfo*: Indicates to enable the end user tracking feature. The default value is `false`. • *should_use_eventhub*: Indicates to enable the Event Hub feature. The default value is `false`. • *eventhub*: – *log_level*: Event log levels. – *queue_size*: Size of the events in kilobytes buffered in Documentum Foundation Classes.

	additional Properties	Additional properties for use in the `dfc.properties` file. By default, the additional properties are commented. To use the following existing additional DFC properties in the `dfc.properties` file, uncomment them as follows:
		`- dfc.security.ssl.truststore=/opt/dctm/certificate/dfc.keystore`
		`- dfc.tokenstorage.enable=false`
		`- dfc.security.ssl.use_anonymous_cipher=true`
		`- dfc.security.ssl.use_existing_truststore=false`
		You can also add new additional DFC properties you want to use in the `dfc.properties` file.

log4j	*rootLogLevel*	Root log level for the Documentum Foundation Services packages. The default value is `WARN`.
	rtLogLevel	Log level for the Documentum Foundation Services run time module. The default value is `WARN`.
	datamodelLogLevel	Log level for the Documentum Foundation Services data model module. The default value is `WARN`.
	servicesLogLevel	Log level for the Documentum Foundation Services services module. The default value is `WARN`.

Category	Name	Description
	toolsLogLevel	Log level for the Documentum Foundation Services tools module. The default value is `WARN`.
	traceLogLevel	Log level for the tracing options. The default value is `WARN`.
service	*ports*	• *httpPort*: HTTP port reserved for the Documentum Foundation Services service. The default value is `8080`. • *sslPort*: SSL port reserved for the Documentum Foundation Services service. The default value is `8443`.
resources	*limits*	• *cpu*: Maximum number of allocated CPUs. • *memory*: Maximum usage of allocated memory.
	requests	• *cpu*: Maximum number of CPUs for transaction. • *memory*: Maximum usage of memory for transaction.
newrelic	• *enable* • *licenseKeyName* • *dfs_application_name* • *proxy_host* • *proxy_port*	"Integrating New Relic with Documentum Foundation Services" on page 231 contains detailed information.
configMap	*fluentd_configMap_name*	ConfigMap name of Fluentd. The format is `<sname>fluentd-configmap`.
volumeClaim Template	*logVctAccessModes*	Access modes of the VCT log. The default value is `ReadWriteOnce`.
	size	Size of the VCT. The default value is `1Gi`.
	logVctStorageClass	Storage class of the VCT log. Specify the value provided for *rwo_storage_class* in the `common-variables` category in Step 4.a.
	logVctSize	Size of the VCT log. The default value is `2Gi`.
extension PVC	*createPVC*	Indicates to create PVC. The default value is `true`.
	PVCAccessMode	Access mode of PVC. The default value is `ReadWriteOnce`.
	PVCStorageClass	Storage class of PVC. The default value is `trident-nfs`.
	PVCSize	Size of the PVC. The default value is `2Gi`.

Category	Name	Description
	useCommon PVC	Indicates to use the common PVC if the PVC already exists and need to be shared across Documentum products. Specify the value provided for *tomcatbase_usecommonpvc* in the common-variables category in Step 4.a.
	useCommon PVC	Name of the common PVC. Specify the value provided for *tomcatbase_commonpvcname* in the common-variables category in Step 4.a.

4. Deploy the Documentum Server and Documentum Foundation Services Helm charts using the following command format:

```
helm install <release_name> <location where Helm charts are extracted>/dctm-server -
f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
platform>.yaml --namespace <name of namespace>
```

   For example:

```
helm install dctm-server /opt/temp/Helm-charts/dctm-server -f /opt/temp/Helm-charts/
dctm-server/platforms/cfcr.yaml --namespace docu
```

5. Verify the status of the deployment of Documentum Server and Documentum Foundation Services Helm charts using the following command format:

```
helm status <release_name>
```

6. Verify the status of the deployment of Documentum Server and Documentum Foundation Services pods using the following command format:

```
kubectl describe pods <name of the pod>
```

Notes

- After setting the values for the variables in the *log4j* category or to upgrade the Documentum Foundation Services pod, run the helm upgrade command in the following command format:

```
helm upgrade <release_name> ./dctm-server -f <location where Helm charts are
extracted>/dctm-server/platforms/<cloud platform>.yaml --namespace <name of
namespace>
```

- After the successful upgrade of the Documentum Foundation Services pod, log in to the pod using the following command format:

```
kubectl exec -it <name of the pod> -c <name of the container> bash
```

- Navigate to /opt/tomcat/CustomConf/, open log4j2.properties, and then verify if the values provided for the variables in the *log4j* category are available.
- If you want to change the log level values of the Documentum Foundation Services pod momentarily, you can navigate to /opt/tomcat/CustomConf/, open log4j2.properties and change the values. The values that you change are effective immediately without using the helm upgrade command and/or restarting the pod. However, whenever you run the helm upgrade command and/or restart the pod at a later time, the values that you changed are lost.

3.4.3 Limitations

Installation path is predefined and cannot be changed. The value is `/opt/tomcat/webapps/dfs`.

3.4.4 Troubleshooting

Symptom	Cause	Fix
When you check the status of available deployed image, it results in the `Error: ImagePullBackOff` error.	Incorrect Helm deployment.	Delete the Helm deployment using the following command: `helm uninstall <release_name> --namespace <name of namespace>` Then, provide the correct image path and redeploy the Helm chart.

3.5 Deploying and configuring Documentum Records Client on private cloud

3.5.1 Prerequisites

1. Perform the steps from Step 1 to Step 4 in "Deploying and configuring Documentum Server on private cloud" on page 47.

2. Deploy the Documentum Server pod as described in "Deploying and configuring Documentum Server on private cloud" on page 47.

3. Download the Docker image (Oracle Linux only) from OpenText Container Registry. Perform the following tasks:

   a. Log in to OpenText Container Registry using the following command format:

   ```
   docker login registry.opentext.com
   ```

   When prompted, provide your OpenText My Support login credentials.

   b. Download the Docker image using the following command format:

   ```
   docker pull registry.opentext.com/<image_name>:<image_tag>
   ```

   The following Docker images are available for download:

Image name	Image tag
dctm-records	23.2.0 or 23.2
dctm-records-darinstallation	23.2.0 or 23.2
dctm-rqm	23.2.0 or 23.2

4. Download the Helm charts from OpenText My Support.

5. Deploy the Records Client Helm charts in the following order:

   a. Deploy the Records Client DAR files. See "Deploying Documentum Records Client DAR files" on page 120.

   b. Deploy the Records Client. See "Deploying Documentum Records Client" on page 122.

   c. Deploy the Records Queue Manager. See "Deploying Records Queue Manager" on page 128.

3.5.2 Deploying Documentum Records Client DAR files

This deployment is for installing the following DAR files in Documentum Server:

- `rps.dar`
- `prm.dar`
- `rm.dar`
- `RM-DoD5015v3-Standard-Record.dar`
- `RM-DoD5015v3-Classified-Record.dar`
- `rmce.dar`
- `Forms.dar`
- `RM-Default.dar`
- `RM-Forms-Adaptor.dar`
- `Rich_Media_Services.dar`
- `Transformation.dar`

To deploy the Records client DAR files:

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.

2. If there are any DARs that were previously installed, you must flush the Documentum Server cache:

   a. To flush the cache, run the following commands through IAPI on Documentum Server:

   ```
   API>flush,c,ddcache,dm_type
   API>flush,c,ddcache,dmi_type_info
   API>flush,c,ddcache,dm_aggr_domain
   API>flush,c,ddcache,dm_domain
   API>flush,c,ddcache,dm_dd_info
   API>flush,c,ddcache,dm_nls_dd_info
   API>flush,c,ddcache,dm_foreign_key
   ```

   b. Clear the persistence cache:

   ```
   API>flush,c,persistentcache
   ```

   c. Publish the data dictionary:

      ```
      API>publish_dd,c
      API>reinit,c
      ```

   d. Restart the repository.

3. To deploy the Records Client DAR files using init container, open the single All-In-One `dctm-server/values.yaml` Helm chart file.

4. Under `contentserver/custom`, set the value of the **useInitContainers** parameter to **true**. Additionally, under `recordsdarinstallation`, set the value of **enabled** to **false**.

5. Open the `dctm-server/init-containers/recordsdarinstaller-init.yaml` file, provide the appropriate values in `content-server` for the variables as described in the following table:

Category	Name	Description
extraInitContainers	*name*	Name of the additional init containers.
	image	recordsdarinstaller image repository location.
	imagePullPolicy	Policy type to fetch the image. The default value is `Always`.
	volumeMounts	• *name*: Name of the volume mount for the init container. • *mountPath*: Path of the volume mount for the init container. • *subPath*: Subpath of the volume mount for the init container.

6. Deploy the Documentum Records DAR Installation Helm using one of the following command format:

   - For new deployment, use the following command format:

     ```
     helm install <release_name> <location where Helm charts are extracted>/dctm-
     server -f <location where Helm charts are  extracted>/dctm-server/init-
     containers/recordsdarinstaller-init.yaml
     ```

   - For upgrade, use the following command format:

     ```
     helm upgrade <release_name> <location where Helm charts are extracted>/dctm-
     server -f <location where Helm charts are extracted>/dctm-server/init-
     containers/recordsdarinstaller-init.yaml
     ```

7. Verify the status of the Helm deployment using the following command format:

   ```
   helm status <release name>
   ```

   The DARs will be installed in the Documentum Server pod after Documentum Server installation is completed.

   You can access all the logs from the Documentum Server pod location, `/opt/dctm_docker/customscriptpvc/records_dar_installer`. If the markerfile is created properly, it indicates that the DARs are successfully installed in the Documentum Server pod.

3.5.3 Deploying Documentum Records Client

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.
2. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and in the `records` category, provide the appropriate values for the variables to pass them to your templates as described in the following table:

Category	Name	Description
enabled	*`enabled`*	Indicates if Records Client pod deployment is enabled. The default value is `false`.
namespace	*`namespace`*	Name of the Kubernetes platform. Specify the value provided for *`namespace`* in the `common-variables` category in Step 4.a.
userName	*`userName`*	User name of the installation owner. Specify the value provided for *`installOwner`* in the `common-variables` category in Step 4.a. The default value is dmadmin.
records	*`replicaCount`*	Number of replica pods. The default value is 2. **Important** OpenText recommends two replica pods.
custom	*`scriptinPVC`*	Custom scripts in PVC.
	`scriptPVCname`	Name of the custom script in PVC.
	`PVCSubPath`	Path of PVC.
persistent VolumeClaim	*`pvcName`*	Name of the PVC.
	`accessModes`	Type of access modes. Valid values are: • `ReadWriteOnce`: Volume can be mounted as read-write by a single node. • `ReadWriteMany`: Volume can be mounted as read-write by many nodes.
	`size`	Storage size of the PV. For example, `1Gi`.
	`storageClass`	Storage class for the PV. Specify the value provided for *`rwo_storage_class`* in the `common-variables` category in Step 4.a.
	`awsEFSCSIDriver`	External storage provisioner (Amazon Elastic File System Container Storage Interface (CSI) Driver). The default value is `efs.csi.aws.com`.
	`awsEFSCSIHandle`	Volume ID of the EFS volume.

Category	Name	Description
service	*service*	• *name*: Name of the service. The default value is `records-svc`. • *port*: Port reserved for the service. The default value is `8080`.
ingress	*name*	Name of the ingress service.
	enable	Indicates if ingress is enabled. If dctm-ingress is used, it is recommended to keep it disabled. Applicable values are either `true` or `false` and case sensitive. When dctm-ingress is used, update the recordsService in dctm-ingress with the service name specified in Records.
	ingressHostName	Ingress host name.
	clusterDomainName	Domain name of the cluster.
	annotations	Metadata attachment to ingress objects. • `nginx.ingress.kubernetes.io/proxy-body-size: 5g` • `nginx.ingress.kubernetes.io/proxy-connect-timeout: 30m`
docbroker	*replicaCount*	Number of replica pods. Specify the value provided for *docbroker_replica_count* in the `common-variables` category in Step 4.a. **Important** OpenText recommends two replica pods.
	serviceName	Service name of connection broker. Specify the value provided for *dbr_service_name* in the `common-variables` category in Step 4.a.
	port	Port for the connection broker. Specify the value provided for *dbr_port* in the `common-variables` category in Step 4.a.
env	*domain*	Name of the cluster space. Specify the value provided for *env_domain* in the `common-variables` category in Step 4.a.

Category	Name	Description
cs	*`useCSDfcConfigMap`*	Indicates to use DFC properties from ConfigMap. • `true`: Uses DFC properties from Documentum Server ConfigMap. • `false`: Uses DFC properties from ConfigMap created through the Records Helm. **Note:** If Documentum Server is SSL enabled, set this value as false.
	`configMapName`	ConfigMap name used in Documentum Server. Specify the value provided for *`dbr_configmap_name`* in the `common-variables` category in Step 4.a. The format is `<sname>.configmap` where `<sname>` is the same provided in the single All-In-One `dctm-server/values.yaml` Helm chart file.
	`csSecretConfigName`	Name of the secret configuration file. Specify the value provided for *`cs_secret_name`* in the `common-variables` category in Step 4.a.
	`allowTrustedLogin`	Indicates to use the *`dfc.session.allow_trusted_login`* in the `dfc.properties` file. The default value is `false`.
certificate	*`useCertificate`*	Indicates to connect to connection broker with SSL certificate. Specify the value provided for *`use_certificates`* in the `common-variables` category in Step 4.a.
	`dbrServiceName`	Service name of connection broker. Specify the value provided for *`dbr_service_name`* in the `common-variables` category in Step 4.a.
	`dbrDataPVCName`	Subpath of certificate inside the connection broker PVC.
deployment	*`name`*	Name of the Documentum Records Client deployment.

Category	Name	Description
images	*records*	• *repository*: Path of the repository. Specify the value provided for *image_repository* in the `common-variables` category in Step 4.a. • *name*: Name of the Docker image. • *tag*: Tag of the Docker image. • *pullPolicy*: Policy type to fetch the Docker image. Specify the value provided for *pull_policy_type* in the `common-variables` category in Step 4.a. • *pullSecrets*: Secret name to fetch the secret configuration file.
	graylog	• *enable*: Indicates if the use of Graylog Sidecar is enabled. Specify the value provided for *graylog_enabled* in the `common-variables` category in Step 4.a. • *image*: Image name of the Graylog Sidecar. Specify the value provided for *graylog_image* in the `common-variables` category in Step 4.a. • *server*: Server details as per your Graylog server configuration. Specify the value provided for *graylog_server* in the `common-variables` category in Step 4.a. • *port*: Port reserved for the Graylog server. Specify the value provided for *graylog_port* in the `common-variables` category in Step 4.a.

Category	Name	Description
containers	*records*	• *name*: Name of the application. • *recordsSingleHelm*: Indicates if Records Client pod deployment is using single Helm chart. • *docbaseName*: Name of the repository. Specify the value provided for *docbase_name* in the `common-variables` category in Step 4.a. • *container Port*: Port reserved within the container to access the application. • *probing*: – *healthPort*: Port reserved for the Documentum Records Client pod. – *failureThreshold*: Number of times the probe is allowed to fail. For example, if the value is set to `2`, then the readiness of the container is marked as `not ready` after the probe fails for two times consecutively. For the same example, for liveness, the pod is restarted after the probe fails for two times consecutively. – *successThreshold*: Minimum consecutive successes for the probe to be considered successful after having failed. – *timeoutSeconds*: Number of seconds after which the probe times out. – *readinessProbe*: ○ *initialDelay Seconds*: Number of seconds after the Documentum Records Client pod has started before the readiness probe is initiated. ○ *periodSeconds*: Frequency to perform the probe. – *livenessProbe*: ○ *initialDelay Seconds*: Number of seconds after the Documentum Records Client pod has started before the liveness probe is initiated. ○ *periodSeconds*: Frequency to perform the probe.

Category	Name	Description
		"Checking liveness of Documentum Records Client" on page 241 contains detailed information.
otds	*enable*	Indicates if OTDS is enabled or disabled. Specify the value provided for *otds_enabled* in the common-variables category in Step 4.a.
	url	OTDS URL.
	clientID	ID of the Records OAuth client.
	scheme	Type of scheme. For example, HTTPS.
	authentication	• *useDefaultDocbase*: Indicates to use the default repository. The default value is `true`. • *repoSelectionPageRequired*: Indicates if the repository page must be displayed on login. The default value is `true`. • *loginTicketTimeout*: Life span of the login ticket. The default value is 250. • *renewtokenafterlogout*: Indicates if the token must be renewed on logout.
wdkAppXml Config	*tagsnvalues*	Syntax for specifying both English and Japanese language pack: For example, `application.language.supported_locales.locale=[en_US, ja_JP]`, `application.language.default_locale=en_US`
logging	*rootLoggerLevel*	Root log level for Documentum Records Client. The default value is `WARN`.
	consoleThresholdLevel	Console threshold level for Documentum Records Client. The default value is `WARN`.
	filename	Name of the log file.
	logFileSize	Maximum size of the log file.
	maxLogFiles	Location of the log file.
userProvided Services	*newrelic*	"Integrating New Relic with Documentum Records Client" on page 232 contains detailed information.
dfcTracing	*enable*	Indicates if DFC tracing is enabled. The default value is `false`.
	configMapName	DFC tracing ConfigMap name for Documentum Records Client.

Category	Name	Description
	filePrefix	File prefix. For example, `dfcTrace`.
	logLevel	Log level for the DFC tracing. The default value is `DEBUG`.
replica count	*replicaCount*	Number of replica pods. The default value is `1`. **Note:** If the value of *replicaCount* is more than `1`, then you must set the value of *pvcAccessModes* to `ReadWriteMany`.

3. Deploy the Documentum Records Client Helm using the following command format:

```
helm upgrade <release_name> <location where Helm charts are extracted>/dctm-server -
f --values resources-values<config>.yaml --namespace <name of namespace>
```

   where `<config>` is: **-large**, **-medium**, or **-small** resource value YAML file that has been provided. These resource files contain pod sizing values like cpu, memory, and so on.

   For example:

```
helm upgrade records /opt/temp/Helm-charts/dctm-server --values resources-values-
small.yaml --namespace docu
```

   Note: Records Client Privilege will be approved automatically.

4. Verify the status of the Helm deployment using the following command format:

```
helm status <release name>
```

5. Verify the status of the deployment of Documentum Records Client pod using the following command format:

```
kubectl describe pods <name of the pod>
```

3.5.4 Deploying Records Queue Manager

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.

2. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and in the `rqm` category, provide the appropriate values for the variables to pass them to your templates as described in the following table:

Category	Name	Description
enabled	*enabled*	Indicates if Records Queue Manager pod deployment is enabled. The default value is `false`.

Category	Name	Description
namespace	*namespace*	Name of the Kubernetes platform. Specify the value provided for *namespace* in the `common-variables` category in Step 4.a.
serviceName	*serviceName*	Name of the service. Specify the value provided for *rqmName* in the `containers` category in *rqm*.
userName	*userName*	User name of the installation owner. Specify the value provided for *installOwner* in the `common-variables` category in Step 4.a. The default value is *dmadmin*.
custom	*createPVC*	Indicates to create the PVC.
	scriptPVCname	Name of the custom script in PVC.
	PVCSubPath	Path of PVC.
persistent VolumeClaim	*pvcName*	Name of the PVC.
	accessModes	Type of access modes. Valid values are: • `ReadWriteOnce`: Volume can be mounted as read-write by a single node. • `ReadWriteMany`: Volume can be mounted as read-write by many nodes. The default value is `ReadWriteOnce`.
	size	Storage size of the PV.
	storageClass	Storage class for the PV. Specify the value provided for *rwo_storage_class* in the `common-variables` category in Step 4.a.
	awsEFSCSIDriver	External storage provisioner (Amazon Elastic File System Container Storage Interface (CSI) Driver). The default value is `efs.csi.aws.com`.
	awsEFSCSIHandle	Volume ID of the EFS volume.
cs	*useCSDfcConfigMap*	Indicates to use DFC properties from ConfigMap. • `true`: Uses DFC properties from Documentum Server ConfigMap. • `false`: Uses DFC properties from ConfigMap created through the Records Helm. **Note:** If Documentum Server is SSL enabled, set this value as false.

Category	Name	Description
	configMapName	ConfigMap name used in Documentum Server. Specify the value provided for *dbr_configmap_name* in the `common-variables` category in Step 4.a. The format is `<sname>.configmap` where `<sname>` is the same provided in the single All-In-One `dctm-server/values.yaml` Helm chart file.
	csSecretConfigName	Name of the secret configuration file. Specify the value provided for *cs_secret_name* in the `common-variables` category in Step 4.a.
	allowTrustedLogin	Indicates to use the *dfc.session.allow_trusted_login* in the `dfc.properties` file. The default value is `false`.
certificate	*useCertificate*	Indicates to connect to connection broker with SSL certificate. Specify the value provided for *use_certificates* in the `common-variables` category in Step 4.a.
	dbrServiceName	Service name of connection broker. Specify the value provided for *dbr_service_name* in the `common-variables` category in Step 4.a.
	dbrDataPVCName	Subpath of certificate inside the connection broker PVC.
images	*repository*	Path of the repository. Specify the value provided for *image_repository* in the `common-variables` category in Step 4.a.
	rqm	• *name*: Name of the Docker image. • *tag*: Tag of the Docker image. • *pullPolicy*: Policy type to fetch the Docker image. Specify the value provided for *pull_policy_type* in the `common-variables` category in Step 4.a. • *pullSecrets*: Secret name to fetch the secret configuration file.

Category	Name	Description
	graylog	• *enable*: Indicates to use the Graylog Sidecar. Specify the value provided for *graylog_enabled* in the `common-variables` category in Step 4.a. • *image*: Image name of the Graylog Sidecar. Specify the value provided for *graylog_image* in the `common-variables` category in Step 4.a. • *server*: Server details as per your Graylog server configuration. Specify the value provided for *graylog_server* in the `common-variables` category in Step 4.a. • *port*: Port reserved for the Graylog server. Specify the value provided for *graylog_port* in the `common-variables` category in Step 4.a.

Category	Name	Description
containers	*rqm*	• *name*: Name of the application. Specify the value provided for *serviceName* in the rqm category. • *containerName*: Name of the container. For example, RQMContainer. • *rqmSingleHelm*: Indicates if RQM pod deployment is using single Helm chart. The default value is true. • *kubernetes*: Indicates if Kubernetes is enabled. The default value is **true**. • *replicaCount*: Number of replica pods. The default value is 1. • *installFiles*: Indicates whether to retain the Records Queue Manager installation files. The default value is false. • *docbaseName*: Name of the repository. Specify the value provided for *docbase_name* in the common-variables category in Step 4.a. • *rqmDocbaseUser*: User name of the Records Queue Manager repository user. The default value is dmadmin. • *rqmSysAdminName*: User name of the Records Queue Manager system administrator. • *rqmSysAdminPass*: User name of the Records Queue Manager system administrator. • *globalRegistry Repository*: Global repository name as in dfc.globalregistry.repository. Specify the value provided for *docbase_name* in the common-variables category in Step 4.a. • *bofRegistryUser*: User name of the global registry. Specify the value provided for *globalUsername* in the common-variables category in Step 4.a. The default value is dm_bof_registry. Do NOT change this value. • *rqmadminport*: Port reserved for the Records Queue Manager Admin. • *rqmjettyport*: Port reserved for the Records Queue Manager Jetty.

Category	Name	Description
		• *docbrokerport*: Port reserved for the connection broker. Specify the value provided for *dbrport* in the `common-variables` category in Step 4.a. • *recordsaws*: Indicates whether to enable Records on AWS.
docbroker	*replicaCount*	Number of replica pods. Specify the value provided for *docbroker_replica_count* in the `common-variables` category in Step 4.a. ! **Important** OpenText recommends two replica pods.
	serviceName	Service name of connection broker. Specify the value provided for *dbr_service_name* in the `common-variables` category in Step 4.a.
	port	Port for the connection broker. Specify the value provided for *dbrport* in the `common-variables` category in Step 4.a.
env	*domain*	Name of the cluster space. Specify the value provided for *env_domain* in the `common-variables` category in Step 4.a.
userProvided Services	*newrelic*	"Integrating New Relic with Documentum Records Client" on page 232 contains detailed information.

3. Deploy the Documentum Records Queue Manager Helm using the following command format:

```
helm upgrade <release_name> <location where Helm charts are extracted>/dctm-server
--values resources-values<config>.yaml --namespace <name of namespace>
```

where `<config>` is: **-large**, **-medium**, or **-small** resource value YAML file that has been provided. These resource files contain pod sizing values like cpu, memory, and so on.

For example:

```
helm upgrade records /opt/temp/Helm-charts/dctm-server --values resources-values-
small.yaml --namespace docu
```

4. Verify the status of the Helm deployment using the following command format:

```
helm status <release name>
```

5. Verify the status of the deployment of Documentum Records Queue Manager pod using the following command format:

```
kubectl describe pods <name of the pod>
```

3.5.5 Limitations

The installation path is predefined and cannot be changed. The value is `/opt/tomcat/records`.

3.5.6 Troubleshooting

Symptom	Cause	Fix
When you check the status of available pods using the `kubectl get pods` command, the READY value of one or more pod(s) reads as `1/2`.	One of the two containers in the specified pod(s) is down or unavailable.	Delete the pod using the `kubectl delete pods <name of the pod>` command. The pod is recreated automatically.
When you check the status of available deployed image, it results in the `Error: ImagePullBackOff` error.	Incorrect Helm deployment.	Delete the Helm deployment using the `helm delete --purge <name of the pod> --namespace <name of namespace>` command, provide the correct image path and redeploy the Helm chart.

3.6 Deploying and configuring Documentum REST Services on private cloud

3.6.1 Prerequisites

1. Perform the tasks mentioned from Step 1 to Step 4 in "Deploying and configuring Documentum Server on private cloud" on page 47.

2. Download the Docker image (Oracle Linux only) from OpenText Container Registry. Perform the following tasks:

   a. Log in to OpenText Container Registry using the following command format:

   ```
   docker login registry.opentext.com
   ```

   When prompted, provide your OpenText My Support login credentials.

   b. Download the Docker image using the following command format:

   ```
   docker pull registry.opentext.com/<image_name>:<image_tag>
   ```

   The following Docker image is available for download:

Image name	Image tag
dctm-rest	23.2.0 or 23.2

3. Download the Helm charts from OpenText My Support.

3.6.2 Deploying Documentum REST Services

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.

2. Perform the tasks mentioned from Step 2 to Step 5 in "Deploying Documentum Server" on page 49.

3. Make sure that the repositories and connection brokers are ready in the cluster.

4. Deploy xPlore and CTS in the cluster if you want the full-text search and CTS capabilities.

5. Navigate to the `dctm-rest` category in the single All-In-One `dctm-server/values.yaml` Helm chart file and provide the appropriate values for the variables to pass them to your templates as described in the following table:

Category	Name	Description
enabled	*`enabled`*	Indicates if Documentum REST Services deployment is enabled.
service name	*`serviceName`*	Name of the service. Specify the value provided for *`sname`* in the `content-server` category in Step 4.e.
namespace	*`namespace`*	Name of the namespace. Specify the value provided for *`namespace`* in the `common-variables` category in Step 4.a.
custom labels	*`customLabels.app`*	Label for the deployment. Specify the value provided for *`sname`* in the `content-server` category in Step 4.e.
container name	*`containerName`*	Name of the container. The default value is `rest-container`.
deployment	*`replicaCount`*	Number of replica pods. The default value is `1`.
image	*`image`*	Image repository information. Specify the value provided for *`image_repository`* in the `common-variables` category in Step 4.a.
	`path`	Name of the Tomcat base image with the repository location. Specify the value provided for *`tomcatbaseimage_name`* in the `common-variables` category in Step 4.a.
	`imageTag`	Tag as a version-specific number. Specify the value provided for *`tomcatbaseimage_tag`* in the `common-variables` category in Step 4.a.
	`imagePullPolicy`	Policy type to fetch the image. Specify the value provided for *`pull_policy_type`* in the `common-variables` category in Step 4.a.

Category	Name	Description
	imagePullSecrets	Secret name to fetch the secret configuration file. The default value is null. **Note:** This is applicable only for deployment in Red Hat OpenShift.
restInit Containers	*name*	Name of the Documentum REST Services init container. The default value is `rest-init-container`.
	image	Name of the init container image with the repository location.
	imageTag	Tag as a version-specific number for the init container image.
	imagePullPolicy	Policy type to fetch the image. The default value is `Always`.
extraInit Containers	*extraInit Containers*	• *name*: Name used for deployment. • *image*: Name of the additional init container image with the repository location. • *imageTag*: Tag as a version-specific number for the additional init container image. • *imagePullPolicy*: Policy type to fetch the image. • *volumeMounts*: – *name*: Volume name to be mounted. The default value is `custom-script-pvc`. Do NOT change this value. – *mountPath*: Path where the volume is mounted. The default value is `/opt/customscriptpvc`. The path is created in the init container Docker file. The init container startup script copies the customization files from the init container image to this mount path. – *subPath*: Subpath inside the referenced volume. The default value is `initcontainercustomscripts/dctm-rest-custom`. Do NOT change this value.
rest	*useCommon PVC*	Indicates to use the common PVC if the PVC already exists and need to be shared across Documentum products. Specify the value provided for *tomcatbase_usecommonpvc* in the `common-variables` category in Step 4.a.
	commonPVC name	Name of the common PVC. Specify the value provided for *tomcatbase_commonpvcname* in the `common-variables` category in Step 4.a.

Category	Name	Description
service	*httpPort*	HTTP port reserved for the Documentum REST Services service. The default value is `8080`.
	httpsPort	HTTPS port reserved for the Documentum REST Services service. The default value is `8443`.
ConfigMap	*existingConfigMap*	Name of the existing *ConfigMap*. Specify the value provided for *dbr_configmap_name* in the `common-variables` category in Step 4.a. If you do not provide any value, installing Documentum REST Services Helm creates a new *ConfigMap* with configuration files specified by *configurationFiles*. You can use an existing *ConfigMap* created using *configurationFiles*.
configuration files	*configurationFiles*	Details of the configuration files. For example, `rest-api-runtime.properties` and `dfc.properties`. The files should be in the root directory of the Helm chart. **Note:** If you want to use the BOF global repository session, do not provide any value for *existingConfigMap* and in addition, update the `dfc.properties` file located at `dctm-server/charts/dctm-rest/` with the appropriate values for the following variables: • *dfc.docbroker.host[0]* • *dfc.docbroker.port[0]* • *dfc.globalregistry.repository* • *dfc.globalregistry.username* • *dfc.globalregistry.password*
extra configuration mount path	*extraConfigMountPath*	Volume mount path for additional configurations. The default value is `/home/dmadmin/ext-conf`.
Java	*javaOptions*	Used to provide the Java options required during the startup of the Tomcat server. The default value `-Djava.security.egd=file:/dev/./urandom` is provided to improve the application performance associated with the random number generation. For more information about random number generation involved with */dev/random* and */dev/urandom*, see *Oracle Linux* documentation.

Category	Name	Description
Documentum Server	*content_server.secretName*	Name of the secret configuration file. Specify the value provided for *cs_secret_name* in the `common-variables` category in Step 4.a.
docbroker	*useCertificate*	Indicates to connect to connection broker with SSL certificate. Specify the value provided for *use_certificates* in the `common-variables` category in Step 4.a.
	dbrServiceName	Service name of connection broker.
	pvcCertSubPath	Subpath of certificate inside the connection broker PVC.
log4j	*rootLogLevel*	Root log level of logging. The default value is `INFO`.
	restLogLevel	Log level of the Documentum REST Services container. The default value is `INFO`.
	dfcLogLevel	Log level of DFC. The default value is `INFO`
graylog	*enabled*	Indicates to use the Graylog Sidecar. Specify the value provided for *graylog_enabled* in the `common-variables` category in Step 4.a.
	image	Image name of Graylog Sidecar. Specify the value provided for *graylog_image* in the `common-variables` category in Step 4.a.
	imagePullPolicy	Policy type to fetch the image. Specify the value provided for *pull_policy_type* in the `common-variables` category in Step 4.a.
	server	Server details as per your Graylog server configuration. Specify the value provided for *graylog_server* in the `common-variables` category in Step 4.a.
	port	Port reserved for the Graylog server. Specify the value provided for *graylog_port* in the `common-variables` category in Step 4.a.
	serviceToken	Service token of the Graylog Sidecar.
newrelic	• *enabled* • *configurationFile* • *addNodeNamePrefix* • *proxy_host* • *proxy_protocol* • *app_name*	"Integrating New Relic with Documentum REST Services" on page 232 contains detailed information.

Category	Name	Description
livenessProbe	• *enabled* • *scheme* • *initialDelay Seconds* • *periodSeconds*	"Checking liveness of Documentum REST Services" on page 242 contains detailed information.
fluentd _service	*enable*	Indicates if Fluentd is enabled. The default value is `false`.
	image	Name of the Fluentd image. Specify the value provided for *fluentd_image* in the `common-variables` category in Step 4.a.
	imagePullPolicy	Policy type to fetch the image. Specify the value provided for *pull_policy_type* in the `common-variables` category in Step 4.a.
	restartPolicy	Policy type to restart the pods. The default value is `Always`.
	TCPPort	Port on which Fluentd is listening for TCP connection. Specify the value provided for *fluentd_tcp_port* in the `common-variables` category in Step 4.a.
	RESTPort	Port on which Fluentd is listening for REST connection. Specify the value provided for *fluentd_rest_port* in the `common-variables` category in Step 4.a.
	UDPPort	Port on which Fluentd is listening for UDP connection. Specify the value provided for *fluentd_udp_port* in the `common-variables` category in Step 4.a.
	servicedata PVCName	Name of the PVC. For example, `fluentd-data-pvc`.
	readiness. enable	Indicates if the readiness probe is enabled. The default value is `false`.
fluentdConf	*enable*	Indicates if Fluentd is enabled. Specify the value provided for *drestfluentd_enabled* in the `fluentd_service` category.
	TCPPort	Port on which Fluentd is listening for TCP connection. Specify the value provided for *fluentd_tcp_port* in the `common-variables` category in Step 4.a.
	RESTPort	Port on which Fluentd is listening for REST connection. Specify the value provided for *fluentd_rest_port* in the `common-variables` category in Step 4.a.

Category	Name	Description
	UDPPort	Port on which Fluentd is listening for UDP connection. Specify the value provided for *fluentd_udp_port* in the common-variables category in Step 4.a.
	kafkabroker	• *port*: Port reserved for the Kafka broker. Specify the value provided for *kafka_broker_port* in the common-variables category in Step 4.a. • *name*: Name of the Kafka broker instance. • *domain*: Domain where Apache Kafka cluster is deployed. Specify the value provided for *env_domain* in the common-variables category in Step 4.a. • *replica*: Number of replica pods. Specify the value provided for *kafka_replica_count* in the common-variables category in Step 4.a.
	kafkaTopic	Same topic name provided while deploying the Apache Kafka cluster.
	kafkaUser	Administrator user name used to communicate among the replicas in the Apache Kafka cluster. Specify the value provided for *kafka_admin_name* in the common-variables category in Step 4.a.
	kafkaUsr Passwd	Administrator password used to communicate among the replicas in the Apache Kafka cluster. Specify the value provided for *kafka_admin_password* in the common-variables category in Step 4.a.
	compression Mode	Type of the compression mode. The default value is gzip.
	buffering Mode	Type of the buffering mode. The two types of buffering mode supported are file buffer and memory buffer. If the value is set to FILEBUFFER, then file buffer is used. Otherwise, memory buffer is used. The default value is FILEBUFFER.
	flushInterval	Time taken to flush the delayed events. The value is considered only for the file buffer mode. For the memory buffer mode, the value is ignored. The default value is 3 seconds.
	flushThread Count	Thread count to flush output in parallel and to increase throughput. The default value is 8.
acsService	*serviceName*	Service name for the ACS ingress resource.
	servicePort	Port reserved for the ACS ingress resource.

Category	Name	Description
otds	*enable*	Indicates if OTDS is enabled. The `rest-api-runtime.properties` file is populated with the value provided for this variable.
	url	OTDS service URL. The format is `https://<OTDS server url>:<port>/otdsws`. The `rest-api-runtime.properties` file is populated with the value provided for this variable.
	clientID	Details of client ID. The `rest-api-runtime.properties` file is populated with the value provided for this variable.
single Helm	*single_helm.enable*	Indicates if Documentum REST Services deployment using single Helm is enabled. The default value is `false`.
custom	*useInitContainers*	Indicates to use init containers. The default value is `false`.
	scriptPVCname	Name of the custom script in PVC. Specify the value provided for *volumeMounts.name*.
	PVCSubPath	Subpath inside the referenced volume. Specify the value provided for *volumeMounts.subPath*.
	PVCSize	Size of the PVC.
	pvcAccessModes	Access modes of the PVC. The default value is `ReadWriteMany`.
	storageClass	Storage class of the PVC. Specify the value provided for *rwm_storage_class* in the `common-variables` category in Step 4.a.
	existVolumePv	Details of existing volume, if any.

6. Deploy the Documentum Server and Documentum REST Services Helm charts using the following command format:

```
helm upgrade <release_name> <location where Helm charts are extracted>/dctm-server -
f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
platform>.yaml --namespace <name of namespace>
```

For example:

```
helm upgrade dctm-server /opt/temp/Helm-charts/dctm-server -f /opt/temp/Helm-charts/
dctm-serverplatforms/cfcr.yaml --namespace docu
```

7. Verify the status of the Documentum Server and Documentum REST Services Helm charts deployment using the following command format:

```
helm status <release name>
```

8. Verify the status of the deployment of Documentum Server and Documentum REST Services pods using the following command format:

```
kubectl describe pods <name of the pod>
```

3.6.2.1 Integrating Graylog

Documentum REST Services Helm chart supports the integration with Graylog using Sidecar. When you set the value of *`graylog.enabled`* to `true`, a Sidecar container is created in the Documentum REST Services pod.

Example for Graylog integration:

```
graylog:
enabled:true
image: gcr.io/documentum-search-product/graylog-sidecar
imagePullPolicy: Always
server: rest-graylog-headless.dctm-rest.svc.cluster.local
port: 9000
serviceToken: 87ckh5e9aammi6rd6g75ceuibce4ot8icb3itpeq4bibea25ge0
logsDir: /home/dmadmin/logs
```

By default, the value of *`graylog.logsDir`* is `/home/dmadmin/logs`. Make sure that the value of *`graylog.logsDir`* aligns with the REST log path specified in `log4j2.properties` in `dctm-rest/template/configMap-log.yaml`.

Sample Graylog Sidecar configuration that needs to be configured in Graylog server to be pushed to Sidecar:

```
#required for Graylog
fields_under_root: true
fields.collector_node_id: ${sidecar.nodeName}
fields.gl2_source_collector    : ${sidecar.nodeId}
fields.source: ${sidecar.nodeName}

filebeat.inputs:
- input_type: log
paths:
- /pod-data*.log
type: log
output.logstash:
hosts: ["rest-graylog-headless.dctm-rest.svc.cluster.local:5044"]
path:
data: /var/lib/graylog-sidecar/collectors/filebeat/data
logs: /var/lib/graylog-sidecar/collectors/filebeat/log
```

3.6.3 Upgrading Documentum REST Services

Helm has built-in upgrade support. Modify the variables in `values.yaml`, including configuration and image version.

```
helm upgrade <release_name> ./dctm-rest
```

When *`ConfigMap`* is updated, an additional argument, the *`--recreate-pods`* argument, is required for the upgrade command as shown in the following command format:

```
helm upgrade --recreate-pods <release_name> ./dctm-server
```

You can rollback the upgrade using the following command format:

```
helm rollback <release_name> <revision>
```

3.6.4 Rolling back the upgrade process

Rolling back the upgrade process (rolling back to the previous image) is recommended when the upgrade process fails or when you encounter errors using the new image.

```
helm rollback <release_name> <revision>
```

3.6.5 Extensibility

Documentum REST Services supports extensibility for you to customize the resources.

If you want to deploy extended Documentum REST Services and/or customization in Kubernetes, perform the following tasks:

1. Create the `1deploy.sh` file to apply the customization-related steps.
2. Build the customization- and configurations-related image that includes the `1deploy.sh` file.
3. Update *`extraInitContainers`* with the appropriate values as described in Step 5.
4. Run the `helm upgrade` command.

3.6.6 Limitations

There are no limitations for this release.

3.6.7 Troubleshooting

There are no troubleshooting information for this release.

3.7 Deploying and configuring Content Connect on private cloud

3.7.1 Prerequisites

Make sure that you complete the following activities before you deploy Content Connect on Kubernetes environment:

1. Obtain and deploy the certificate authority (CA) certificates to deploy Content Connect and DCTM-REST on the HTTPS mode.
2. Download and configure the Docker application from the Docker website. The *Docker documentation* contains detailed information.
3. Download and configure Helm.
4. Download and configure the Kubernetes application from the Kubernetes website.
5. Download the Content Connect Docker images from OpenText Container Registry.

   The Step 5 in "Prerequisites" on page 41 contains detailed information.
6. Run the Content Connect `docker images` command to verify if the Docker images are downloaded successfully and listed.
7. Download the Documentum Server Helm chart available in the `documentum_server_<release-version>_kubernetes_helm_<operating-system>.tar` file from the *OpenText My Support* > *Documentum Server* > *Software Download* page.
8. Integrate Content Connect with Documentum REST Services.

   - If Documentum REST Services is not deployed, then perform the following steps:

     1. Navigate to the `<dctm-server>\charts\dctm-rest\templates`, open the `configMap-rest-api-runtime-properties.yaml` file.
     2. Add the following entries:

        ```
        rest.cors.enabled=true
        ```

```
rest.cors.allowed.origins=<Content Connect url>
rest.cors.allowed.methods=GET, POST, PUT, DELETE, OPTIONS, HEAD
rest.cors.allowed.headers=Access-Control-Allow-Origin, DOCUMENTUM-CUSTOM-
UNAUTH-SCHEME, Authorization, Content-Type, Accept, X-CLIENT-LOCATION, X-
CLIENT-APPLICATION-NAME rest.cors.exposed.headers=Accept-Ranges, Content-
Encoding,Content-Length, Content-Range,Authorization, Content-
Disposition
rest.should.parse.emails=true
```

3. Save the changes.

- If Documentum REST Services is already deployed, perform the following steps:

1. Run the following command to edit the configmap of `rest-api-runtime-properties`.

```
kubectl edit cm rest-api-runtime-properties
```

2. Add the following entries:

```
rest.cors.enabled=true
rest.cors.allowed.origins=<Content Connect url>
rest.cors.allowed.methods=GET, POST, PUT, DELETE, OPTIONS, HEAD
rest.cors.allowed.headers=Access-Control-Allow-Origin, DOCUMENTUM-
CUSTOMUNAUTH- SCHEME, Authorization, Content-Type, Accept, X-CLIENT-
LOCATION, XCLIENT- APPLICATION-NAME
rest.cors.exposed.headers=Accept-Ranges, Content- Encoding,Content-
Length, Content-Range,Authorization, Content- Disposition
rest.should.parse.emails=true
```

3. Save the changes.
4. Restart the dctm-rest pod.

3.7.2 Deploying Content Connect

1. Extract the Documentum Server Helm charts downloaded from OpenText My Support.

2. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and perform the following:

   - Set the value of *`ccExtension`* as per your requirement in `common-variables` (“common-variables” on page 50)
   - Provide the appropriate values accordingly in the `contentconnect` section for the mandatory variables depending on your environment to pass them to your templates as described in the following table:

Table 3-14: contentconnect

Category	Name	Description
contentconnect	*`enabled`*	Set the value to `true` to deploy the Content Connect. By default, the value is set to `false`.
secret	*`DB_PASSWORD`*	Password of the database to be connected for Content Connect.

Category	Name	Description
namespace	*namespace*	Namespace in the Kubernetes environment to deploy Content Connect.
configmap	*DB_IP*	Database IP to be connected.
	DB_PORT	Database port to be connected.
	DB_USERNAME	Database user to be connected.
	DB_DB	Database name to be connected.
	DB_TYPE	Database type to be connected. Valid values are: – `postgres` for PostgreSQL – `mssql` for Microsoft SQL
	DB_TABLESPACE_NAME	Table space name to create the database. Example: `contentconnect_tablespace` To use the default tablespace, this value must be empty. This parameter is not supported for the Microsoft SQL database. Hence, specify an invalid tablespace name.
	authType	Authentication type for Content Connect. By default the value is set to `otds`. Available modes are: – otds – Basic
	otdsUrl	OTDS server URL. Example: `https://<url>otdsws`. For Basic authentication, this value must be empty.
	otdsClientID	Client ID created for Content Connect in OTDS admin. Example: `cc-client` For Basic authentication, this value must be empty.
	clientId	Client ID of the application registered in Microsoft Azure. This value is mandatory. For more information, see *OpenText Content Connect Installation and Administration Guide*.
	clientSecret	Client Secret of the application registered in Microsoft Azure. This value is mandatory. For more information, see *OpenText Content Connect Installation and Administration Guide*.

Category	Name	Description
	tenantId	Tenant ID of the application registered in Microsoft Azure. This value is mandatory. For more information, see *OpenText Content Connect Installation and Administration Guide*.
	protocol	By default, Content Connect runs on the IPv6 protocol. When only IPv4 protocol is installed on the server machine, ensure to update the protocol value to *ipv4*.
newrelic	*newrelic*	"Integrating New Relic with Content Connect" on page 233 contains detailed information.
ingress	*enabled*	By default its set to *false*. Enable this value to use the cc ingress.
	configureHost	Set the value to true to configure the host and cluster domain name. **Note:** To use ALB as the ingress controller, set the value to false. In addition, you must update the appropriate value for annotations.
	tls.enable	Set the value to false to deploy ingress on HTTP. Default value: false. Set this value to true to deploy ingress on HTTPS. It is mandatory to provide the *crt* and *key* values to deploy ingress on HTTPS.
	tls.hosts	Name of the ingress domain URL. The format is *<ingress-domain-url>*.
	rules.host	Name of the ingress host. The host appended to ingressHostName during the creation of Ingress.
graylog	*graylog.enable*	Set the value to true to enable application log files in Graylog. Whether Graylog is enabled or disabled, Content Connect default logging is retained.
images	**cc:**	
	repository	The path of the repository. Specify the value provided for *image_repository* in the common-variables category in Step 4.a.
	name	The name for Docker Image.
	tag	The tag as a version-specific number.

Category	Name	Description
	pullPolicy	The policy type to fetch the image. Specify the value provided for *pull_policy_type* in the `common-variables` category in Step 4.a.
	ccdb:	
	repository	The path of the repository.
	name	The name for Docker Image.
	tag	The tag as a version-specific number.
	pullPolicy	The policy type to fetch the image.
	pullSecrets	Secret name to fetch the secret configuration file. Specify the value provided for *pull_secret_name* in the `common-variables` category in Step 4.a.
contentconnectdb	*contentconnectdb.value*	Set the value to `true` to create the Content Connect database. **Note:** This value must be set to `false` while performing an upgrade.
certificatesecret	*crt*	Certificate value.
	key	Certificate key value.

3. To enable ingress resources, make sure that you set the value of enabled to *true* in the `dctm-ingress` category in the single All-In-One `dctm-server/values.yaml` Helm chart file for the following parameters:

```
ccService:
   enabled: true
   serviceName: cc
   servicePort: 8080
   extension: *ccExtension

  ccadminService:
   enabled: true
   serviceName: cc-admin
   servicePort: 80
   extension: *ccExtension
```

4. Optional Perform the following steps to install the `Email_Attachments_Relation.dar` file.

   Note: The DAR file installation is required to establish a relation between:

   - The email and its attachments.
   - The attachments when the parent email is not selected. In such instances, the first attachment is considered as the parent entity.

   For more information, see *OpenText Content Connect Installation and Administration Guide*.

a. In the single All-In-One `dctm-server/values.yaml` Helm chart file, modify `init-containers` as per the following:

   1. Set the value for `custom: useInitContainers` to *`true`*

      ```
      custom:
      useInitContainers: true
      ```

   2. Verify the Content Connect image name in `dctm-server\init-containers\cc-init.yaml`.

   3. Update the `<sname>` in the subpath in `cc-init.yaml`.

b. Run the following Helm command to install dctm-server helm chart with CC dar enabled:

   ```
   helm install <release-name> . -f .\platforms\cfcr.yaml -f .\init-containers\cc-
   init.yaml --values .\resources-values-test.yaml -n <namespace>
   ```

The logs of the dar installation are available in the CS pod logs. Detailed logs of CC dars are located at `/opt/dctm/dba/config/<docbase_name>/ccdars.log`.

5. Navigate to the location where Helm Chart TAR files are extracted and deploy the Documentum Server Helm in your Kubernetes environment using the following command format:

   - For Documentum Server single Helm, run the following command:

     ```
     helm install <release name> . -f ./platforms/<current platform>.yaml --
     values ./<resource>.yaml -n <namespace>
     ```

     For example:

     ```
     helm install cctest . -f ./platforms/aws.yaml --values ./resources-values-
     test.yaml -n test
     ```

6. Verify the status of the Helm deployment using the following command format:

   ```
   helm status <release name>
   ```

   Note: If Helm deployment is unsuccessful, the Administrator must run the following command to remove the created services, ConfigMap, secrets and so on and then redeploy Helm:

   `helm delete <releasename>`

7. Verify the status of the Kubectl pod deployment using the following command:

   ```
   kubectl get pods
   ```

   The following is an example of the output:

   ```
   NAME                                 READY        Status         RESTARTS        AGE
   cc-7bf9889df8-5gx7c          1/1        Running        0                 52s
   ```

3.7.3 Deploying Content Connect on the client machine

1. After the Admin URL is up, add the endpoint details and download the manifest file.
2. Use the manifest file to add the Content Connect add-in to the Microsoft Word or Outlook application. For more information, see *OpenText Content Connect Installation and Administration Guide*.

3.7.4 Deploying only Content Connect using single Helm

3.7.4.1 Documentum Server single Helm

To deploy only Content Connect for Documentum Server in single Helm, perform the following steps in the dctm-server/values.yaml file:

1. Except for the following categories, disable all the product deployment categories:
   - `cs-secrets` and `contentconnect`
   - `ccService` and `ccadminService` (`dctm-ingress`)
2. Update the values for `database.userName` and `database.password`.
3. In the `contentconnect` category, provide all the required values.

   The “contentconnect” on page 145 table contains detailed information about the variables.
4. Deploy the Content Connect Helm chart and pod using the same command as Documentum Server deployment.

3.7.5 Limitations

The installation path is predefined and cannot be changed. The value is */usr/src/app/Content_Connect*.

3.7.6 Troubleshooting

Symptom	Cause	Fix
When you check the status of available pods using the `kubectl get pods` command, the READY value of one or more pod(s) reads as `1/2`.	One of the two containers in the specified pod(s) is down or unavailable.	Delete the pod using the following command: `kubectl delete pod`. After you delete, the pod is recreated automatically.

Symptom	Cause	Fix
When you check the status of available deployed image, it results in the **Error: ImagePullBackOff** error.	Incorrect Helm deployment.	Delete the Helm deployment using the following command: `helm delete <deployment name> -n <namespace>`. Then, provide the correct image path and redeploy the Helm Chart.
When using Content Connect, Cross-Origin Resource Sharing (CORS) related errors occur even after all configurations are set.	CORS related erros occur.	• Increase the load balancer response timeout. • Add the Content Connect Admin Console URL to the `rest.cors.allowed.origins` tag in the `rest-api-runtime.properties` file.

3.7.7 Upgrading Content Connect

Helm has built-in upgrade support. Modify the variables in `values.yaml`, including configuration and image version.

```
helm upgrade <release name> . -f ./platforms/<current platform>.yaml --values ./
<resource>.yaml -n <namespace>
```

Note: You must use `cc-ingress` to upgrade Content Connect.

You can rollback the upgrade using the following command format:

```
helm rollback <release_name> <revision>
```

3.8 Deploying and configuring Documentum Workflow Designer on private cloud

3.8.1 Prerequisites

1. Perform the steps from Step 1 to Step 9 in “Deploying and configuring Documentum Server on private cloud” on page 47.
2. Deploy the Documentum Server pod as described from Step 1 to Step 5 in “Deploying and configuring Documentum Server on private cloud” on page 47.
3. Download the Documentum Workflow Designer Docker image (Oracle Linux only) from OpenText Container Registry. Perform the following tasks:

    a. Log in to OpenText Container Registry using the following command format:

    ```
    docker login registry.opentext.com
    ```

When prompted, provide your OpenText My Support login credentials.

b. Download the Docker image using the following command format:

```
docker pull registry.opentext.com/<image_name>:<image_tag>
```

The following Docker image is available for download:

Image name	Image tag
dctm-workflow-designer	23.2.0 or 23.2

4. Enable *`pe-silentinstaller`* and *`dctm-workflow-designer`* in the single All-In-One `dctm-server/values.yaml` Helm chart file.

5. Provide the marker file names of client products in the `content-server` category in the single All-In-One `dctm-server/values.yaml` Helm chart file.

   For example:

```
markerFiles: pe-copy-succeeded-<release-version>
```

3.8.2 Deploying Documentum Workflow Designer

1. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and in the `pe-silentinstaller` category, provide the appropriate values for the variables to pass them to your templates as described in the following table:

Category	Name	Description
enabled	*`enabled`*	Indicates if Process Engine pod deployment is enabled. The default value is `false`. Make sure that this attribute is enabled before you deploy Documentum Workflow Designer.
image	*`repository`*	Path of the repository.
	`name`	Name of the Documentum xCP installer image.
	`tag`	Tag as a version-specific number.
	`pullPolicy`	Policy type to fetch the image. Specify the value provided for *`pull_policy_type`* in the `common-variables` category in Step 4.a.
prefix	*`prefix`*	Prefix for the Process Engine pod name or ID. Use a suitable prefix to distinguish Process Engine pods in the cluster environment.
docbase Connection	*`username`*	Name of the repository. Specify the value provided for *`installOwner.userName`* in the `contentserver` category in `cs-secrets` in Step 4.b.
	`password`	Password of the repository user. Specify the value provided for *`installOwner.password`* in the `contentserver` category in Step 4.b.

Category	Name	Description
	globalRegistry Username	User name for the global registry user. Specify the value provided for *globalUsername* in the `cs-dfc-properties` category in Step 4.h.
	globalRegistry Password	Password for the global registry user. Specify the value provided for *globalRegistryPassword* in the `cs-secrets` category in Step 4.b.
appServer	*httpport*	Service HTTP port reserved for Process Engine. The default value is `9080`.
	password	Password of the application server. Specify the value provided for *contentserver.install.appserver.admin.password* in the `cs-secrets` category in Step 4.b.
	servicename	JMS service name.
persistent Volume	*createPVC*	Set to `true` if Process Engine is the first client to be installed on Documentum Server. Enabling this as `true` creates a PVC shared by other clients to copy their install scripts to Documentum Server container. If this is not the first client, set the variable as `false` as the PVC is created by another client. The default value is `false`.
	scriptPVCname	When Process Engine is the first client and createPVC variable is set as `true`, specify the name for the PVC, which is created by the Process Engine Helm package. Alternatively, specify the name of PVC created by another client. For example, `<sname>`customscript-pvc.
	PVCSubPath	When Process Engine is the first client and *createPVC* is set as `true`, specify the name of the subpath which is created by the Process Engine Helm package. Alternatively, specify the subpath provided by another client.
	size	Size of the PVC.
	storageClass ReadWriteMany	Storage class for a PV, with ReadWriteMany access mode, which can be accessible by many nodes. Specify the value provided for *rwm_storage_class* in the `common-variables` category in Step 4.a.

2. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and in the `dctm-workflow-designer` category, provide the appropriate values for the variables to pass them to your templates as described in the following table:

Category	Name	Description
enable	*enabled*	Indicates if Documentum Workflow Designer pod deployment is enabled. The default value is `false`. Make sure that this attribute is enabled.
prefix	*prefix*	Prefix for Documentum Workflow Designer pod name or ID. Use a suitable prefix to distinguish Documentum Workflow Designer pod in the cluster environment.
initImage	*repository*	Path of the repository. Specify the value provided for *xcp_repository* in the `pe-silentinstaller` category.
	name	Any image with the wget tool such as BusyBox. For example, `dctm-xcp-apphost`.
	tag	Tag as a version-specific number.
	pullPolicy	Policy type to fetch the image. Specify the value provided for *pull_policy_type* in the `common-variables` category in Step 4.a.
image	*repository*	Path of the repository. Specify the value provided for *xcp_repository* in the `pe-silentinstaller` category.
	name	Name of Documentum Workflow Designer image. For example, `dctm-workflow-designer`.
	tag	Tag as a version-specific number.
	pullPolicy	Policy type to fetch the image. Specify the value provided for *pull_policy_type* in the `common-variables` category in Step 4.a.
contextPath	*contextPath*	Context path of the Documentum Workflow Designer application. By default, the path for Documentum Workflow Designer application is `DocumentumWorkflowDesigner`. For example, `http://<ingress-host>/DocumentumWorkflowDesigner`.
persistent Volume	*storageClass ReadWriteOnce*	Storage class for a PV, with ReadWriteOnce access mode, which can be accessible by many nodes. Specify the value provided for *rwo_storage_class* in the `common-variables` category in Step 4.a. **Note:** Change the variable name from *rwm_storage_class* to *rwo_storage_class* to mount the volume to the required storage class.
	size	PV size to accommodate all the log files.

Category	Name	Description
docbase Connection	*use_certificate*	Indicates if certificate-based communication with Documentum Server is enabled. Specify the value provided for *use_certificate* in the `common-variables` category in Step 4.a.
	truststore Password	Password of the DFC truststore that contains Documentum Server and connection broker certificates. Specify the value provided for *docbroker.certificate.truststorepassword* in the `cs-secrets` category in Step 4.b.
	docbroker	Name of the service for the connection broker.
	port	Port for the connection broker. Specify the value provided for *dbrport* in the `common-variables` category in Step 4.a.
	jmshttpport	JMS port on which Documentum Server JMS service is running. Default is 9080.
	jmsservicename	JMS service name created by Documentum Server.
	docbase	Name of the repository. Specify the value provided for *docbase_name* in the `common-variables` category in Step 4.a.
	superUser	User name to connect to the repository. Specify the value provided for *installOwner* in the `cs-secrets` category in Step 4.b.
	superUserPassword	Password to connect to the repository. Specify the value provided for *installOwnerPassword* in the `cs-secrets` category in Step 4.b.
	globalRegistry Repository	Name of the global registry repository. Specify the value provided for *docbase_name* in the `cs-secrets` category in Step 4.b.
	globalRegistry Username	User name to access the global registry repository. Specify the value provided for *globalUsername* in the `cs-dfc-properties` category in Step 4.h.
	globalRegistry Password	Password to access the global registry repository. Specify the value provided for *globalRegistryPassword* in the `cs-secrets` category in Step 4.b.
dbrpersistentVolume	*dbrdataPVC Name*	Name of the PVC. Specify the value provided for *persistentVolume.dbrdataPVCName* in the `docbroker` category in Step 4.d.
tomcat	*javaOptions*	JMS memory settings for Tomcat server.

Category	Name	Description
secret	*name*	Name of the Documentum Server secret resource. Specify the value provided for *cs_secret_name* in the `common-variables` category in Step 4.a.
newrelic	*enable*	Indicates if the New Relic Java agent is enabled. Set the element to `true` to enable the New Relic Java agent. Specify the value provided for *newrelic_enabled* in the `common-variables` category in Step 4.a.
	licenseKeyName	License key corresponding to the name of the New Relic license key. Specify the value provided for *newrelic.license_key* in the `cs-secrets` category in Step 4.b.
	app_name	Descriptive application name for the Documentum Workflow Designer. This attribute is mandatory if New Relic is enabled. The format is `DCTM_WFD-PROD-OT2_CFCR_LI3-EIM-<sname>`.
	proxy_host	IP address of the proxy server. Specify the value provided for *newrelic_proxy_host* in the `common-variables` category in Step 4.a.
	proxy_port	Port reserved for the New Relic server. The default port reserved for New Relic server is `3128`. Specify the value provided for *newrelic_proxy_port* in the `common-variables` category in Step 4.a.
liveness	*initialDelay*	Time in seconds after the container or pod has started before the liveness probe is initiated.
	timeout	Number of seconds after which the probe times out.
	period	Frequency to perform the probe.
	failure	Number of times the probe is allowed to fail.
readiness	*initialDelay*	Time in seconds after the container or pod has started before the readiness probe is initiated.
	timeout	Number of seconds after which the probe times out.
	period	Frequency to perform the probe.
	failure	Time when a pod starts and the probe fails, Kubernetes attempts based on the failure threshold time before stopping.

3. Deploy the Documentum Server and Documentum Workflow Designer Helm charts using the following command format:

```
helm install <release_name> <location where Helm charts are extracted>/dctm-server -
f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
platform>.yaml --namespace <name of namespace>
```

For example:

```
helm install dctm-workflow-designer ./dctm-server -f <location where Helm charts
are extracted>/dctm-server/platforms/cfcr.yaml --values ./dctm-server/values.yaml --
namespace docu
```

Note: Make sure that you deploy Documentum Workflow Designer in the same namespace where the PostgreSQL database is installed.

4. Verify the status of the deployment of Documentum Workflow Designer Helm using the following command format:

```
helm status <release_name>
```

5. Verify the status of the deployment of the Documentum Workflow Designer pod using the following command format:

```
kubectl describe pods <name of the pod>
```

6. Enable the `workflowDesignerService` and `bpm` services to access Documentum Workflow Designer from a web browser.

   a. Set the value of *`enable`* to `true` for *`bpm`* and *`workflowDesignerService`* in the `dctm-ingress` category in the single All-In-One `dctm-server/values.yaml` Helm chart file.

   b. Configure *`serviceName`* for *`bpm`* and *`workflowDesignerService`* services.

   Notes

   - To access Documentum Workflow Designer from web browser, use `http://<ingress-host>/DocumentumWorkflowDesigner`.
   - Do not modify the default port value for *`servicePort`*.

3.8.3 Importing batch processes using WFD cloud utility

Starting with WorkFlow Designer 22.2, you can import workflow packages into Workflow Designer using the WorkFlow Designer cloud utility. The utility scans the specified shared directory path for packages and uploads the available packages to the WorkFlow Designer. You must copy the extracted packages into the shared directory and run the utility to upload the packages. A new docker image is created and corresponding Helm chart are created to configure workflow designer environment details and the shared directory path from shared PV.

Notes

- To avoid any process failure, you must import one process at a time.
- If you are importing multiple packages and one of the process fails during installation, rest of the processes in the package are ignored for installation and the next package is installed.

3.8.3.1 Prerequisites

- Processes from previous versions should be validated using Workflow Designer before exporting with Workflow Designer cloud utility.
- The shared PV and corresponding shared directory path for importing process packages should be valid.
- The user should be able to login using inline or basic authentication into Workflow Designer.

3.8.3.2 Deploying WorkFlow Designer Cloud utility

1. Open the `dctm-workflow-designer-cli/values.yaml` file and provide the appropriate values for the variables to pass them to your templates as described in the following table:

Category	Name	Description
prefix	*prefix*	Prefix for name or ID of the utility pod. Use a suitable prefix to distinguish WFD client utility pod in the cluster environment.
image	*repository*	Path of the repository.
	name	Name of the image.
	tag	Tag as a version-specific number.
	pullPolicy	Policy type to fetch the image. By default, the value is `IfNotPresent`, which skips fetching a Docker image if the same image already exists. Otherwise, use `Always` to force to fetch the Docker image.
	pullSecrets	Secret for fetching images.
workflow Designer	*url*	Workflow Designer URL where packages needs to be imported.
	username	Workflow Designer username to connect to the repository.
	password	Workflow Designer password to connect to the repository.
	repositoryName	Name of Workflow Designer repository.
trustStore	*enable*	Indicates if certificate-based communication with Workflow Designer is enabled. Specify the value either as `true` to enable the truststore or as `false` to disable the truststore for Workflow Designer.
	fileName	Truststore filename. Truststore file should be saved in secrets folder available in Helm chart.

Category	Name	Description
	password	Truststore password for the Workflow Designer.
workflow Designer CliParam	*force*	Indicates if the existing processes override is enabled. Specify `true` to enable overriding the existing processes or as `false` to disable overriding existing processes.
persistent Volume	*storageClassReadWriteOnce*	Storage class for a PV, with `ReadWriteOnce` access mode, which can be accessed by a single node.
	size	Size of the PV.
processPackage PersistentVolume	*sharedPVCName*	PV name shared by customer for importing the process package files.
	sharedPVCFolder	Shared subpath in the PV for importing the process package files.

2. Deploy the Documentum Workflow Designer Cloud utility Helm charts using the following command format:

```
helm install <release_name> <location where Helm charts are extracted>/ --namespace
<name of namespace>
```

For example:

```
helm install wfd-cli ./dctm-workflow-designer-cli --namespace docu
```

Note: After completion, the pod status changes to **Complete** and the imported processes are installed into WorkFlow Designer.

3. Verify the status of the deployment of Documentum Workflow Designer Cloud utility Helm using the following command format:

```
helm status <release_name>
```

4. Verify the status of the deployment of the Documentum Workflow Designer Cloud utility pod using the following command format:

```
kubectl describe pods <name of the pod>
```

3.8.4 Limitations

There are no limitations for this release.

3.8.5 Troubleshooting

There are no troubleshooting information for this release.

Chapter 4

Documentum Platform and Platform Extensions applications on Microsoft Azure cloud platform

The product *Release Notes* document contains detailed information about the list of applications and its supported versions for Microsoft Azure cloud platform.

4.1 Deploying and configuring Documentum Server on Microsoft Azure cloud platform

4.1.1 Kubernetes platform

4.1.1.1 Prerequisites

1. Download and configure the Docker application from the Docker website.

   Docker Documentation contains detailed information.

2. Download the supported version of Helm package from the Helm website.

   The product *Release Notes* document contains detailed information about the supported versions.

   Helm Documentation contains detailed information.

3. Download and configure the Kubernetes application from the Kubernetes website.

   Kubernetes Documentation contains detailed information.

4. Download and configure the PostgreSQL database (server) from the PostgreSQL website.

   Note: The PostgreSQL database client is packaged with Documentum Server Docker image.

   PostgreSQL Documentation contains detailed information.

5. Set up Azure.

   a. Create a resource group. A resource group in Azure is a folder to keep your collection. It does not serve any other purpose.
   b. Create a container registry. Container registry is used to store the Docker images in Azure. A standard container registry can store up to 100 GB of images.
   c. Create an Azure Kubernetes Service (AKS). AKS is a managed container orchestration service, based on the open source Kubernetes system, which is available on the Azure public cloud.

d. Create an Azure database for the PostgreSQL server to enable Postgres as a service.

Microsoft Azure Documentation contains detailed information.

6. Connect to Azure cluster using the Azure CLI command.

7. Create a storage account using the following command format:

```
az storage account create --resource-group <name of the resource group>
--name <name of the storage account>
--sku <SKU of the storage account>
```

Example output:

```
[root@skvoraclelinux ~]# az storage account create
--resource-group MC_dctm_dctmaks_eastus
name dctmstorageacc --sku Standard_LRS
{
"accessTier": null,
"creationTime": "2018-11-26T06:11:58.380457+00:00",
"customDomain": null,
"enableHttpsTrafficOnly": false,
"encryption": {
"keySource": "Microsoft.Storage",
"keyVaultProperties": null,
"services": {
"blob": {
...
...
[root@skvoraclelinux ~]#
```

8. Create a storage class, a YAML file (for example, `azstorageclass.yaml` with appropriate values for the parameters), and apply the configuration.

A storage class is used to define how an Azure file share is created. A storage account can be specified in the class.

Different types of storage are:

- Locally-redundant storage (LRS): A simple, low-cost replication strategy. Data is replicated within a single storage scale unit.
- Zone-redundant storage (ZRS): Replication for high availability and durability. Data is replicated synchronously across three availability zones.
- Geo-redundant storage (GRS): Cross-regional replication to protect against region-wide unavailability.
- Read-access geo-redundant storage (RA-GRS): Cross-regional replication with read access to the replica.

Perform the following tasks:

a. Create a storage class using the following command format:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
name: azurefile
provisioner: kubernetes.io/azure-file
mountOptions:
- dir_mode=0777 (refers to permission mode)
- file_mode=0777
- uid=1000 (refers to the user id of the Documentum installation owner)
- gid=1000
parameters:
skuName: Standard_LRS
storageAccount: <name of the created storage account>
```

b. Apply the configuration to a resource using the name of the YAML file using the following command format:

```
kubectl apply -f <name of the YAML file>.yaml
```

Example output:

```
[root@skvoraclelinux ~]# kubectl apply -f azstorageclass.yaml
storageclass.storage.k8s.io/azurefile created
```

Resource is created if it does not exist yet. Make sure that you specify the resource name.

9. Create and verify a successful sample PVC creation with the newly created storage class (azurefile) in RWX access mode.

10. Download and upload the Docker image(s) into Azure. Perform the following tasks:

a. Download the Documentum Server and the required Documentum application Docker images (Oracle Linux only) from OpenText Container Registry. Perform the following tasks:

i. Log in to OpenText Container Registry using the following command format:

```
docker login registry.opentext.com
```

When prompted, provide your OpenText My Support login credentials.

ii. Download the Docker image using the following command format:

```
docker pull registry.opentext.com/<image_name>:<image_tag>
```

The following Docker images are available for download:

Image name	Image tag
dctm-server	23.2.0 or 23.2
dctm-admin	23.2.0 or 23.2
dctm-dfs	23.2.0 or 23.2
dctm-records	23.2.0 or 23.2
dctm-records-darinstallation	23.2.0 or 23.2
dctm-rqm	23.2.0 or 23.2
dctm-rest	23.2.0 or 23.2

b. Tag the Docker image(s) to the Azure registry specific image using the following command format:

```
docker tag <source image> <destination image>
```

c. Log in to the Azure container registries if not already logged in using the following command format:

```
az acr login --name <azure container registry>
```

d. Upload the Docker image(s) to the Azure Container Registry (ACR) using the following command format:

```
docker push <image>
```

11. Install the NGINX controller.

    Azure Documentation contains detailed information.

12. Download the Helm charts available in the `documentum_server_<release-version>_kubernetes_helm_<operating-system>.tar` file from OpenText My Support.

    Note: The TAR file also contains Docker Compose scripts for reference along with Helm charts for the Oracle Linux operating system.

4.1.1.2 Deploying Documentum Server

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.

2. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and provide the appropriate values for the variables to pass them to your templates.

   "Deploying and configuring Documentum Server on private cloud" on page 47 contains detailed information about the variables.

3. Optional To enable Documentum Server/connection broker external access, update the values for the following variables:

   a. Set the value of *`externalAccessEnaled`* to `true` in the `common-variables` category.

   b. Retain all the default values for *`ExtDocbroker`* in the `docbroker` category.

   c. Set the value of *`ExtCS.nativeExtPort`* to `80` and *`sslExtPort`* to `81` in the `content-server` category. Retain the default values for all other variables.

4. To enable ingress resources, make sure that you set the value of *`enabled`* to `true` in the `dctm-ingress` category in the single All-In-One `dctm-server/values.yaml` Helm chart file.

5. Set the value of *`configureHost`* in the `dctm-ingress` category in the single All-In-One `dctm-server/values.yaml` Helm chart file to `false`.

   Note: When you set the value of *`configureHost`* to `true`, you must provide the appropriate values for *`host`* and *`clusterDomainName`*.

6. Enable and update the service names for the required ingress resource in the `dctm-ingress` category in the single All-In-One `dctm-server/values.yaml` Helm chart file.

   Sample with example values:

```
jmsService:
  enable: true
  serviceName: dctmdcs-pg-jms-service
  servicePort: 9080

jmsBase:
    enable: false
    serviceName: <jms-service-name>
    servicePort: 9080

acsService:
  enable: true
  serviceName: dctmdcs-pg-jms-service
  servicePort: 9080

tnsService:
  enable: true
  serviceName: dctmdcs-pg-tns-service
  servicePort: 8081
```

7. Deploy the Documentum Server Helm using the following command format:

```
helm install <release_name> <location where Helm charts are extracted>/dctm-server -
f <location where Helm charts are extracted>/dctm-server/platforms/
<cloudplatform>.yaml --namespace <name of namespace>
```

   For example:

```
helm install dctm-server /opt/temp/Helm-charts/dctm-server -f /opt/temp/Helm-charts/
dctm-server/platforms/azure.yaml --namespace docu
```

   Note: Make sure that you deploy Documentum Server in the same namespace where the PostgreSQL database is installed.

8. Verify the status of the deployment of Documentum Server Helm using the following command format:

```
helm status <release_name>
```

9. Verify the status of the deployment of Documentum Server pod using the following command format:

```
kubectl describe pods <name of the pod>
```

10. Obtain the ingress deployment address using the following command format:

```
kubectl get ingress
```

   Example output with address in bold:

```
NAME                                    CLASS        HOSTS
ADDRESS                                PORTS   AGE
azurenginx-ingress   nginx   *             20.236.198.164   80      4d1h
```

11. To access any Documentum ingress resources in a browser, use the following URL format:

```
http://<ingress deployment external IP>/<Documentum ingress resource>
```

For example:

If you are trying to access the ACS in a browser, use the following URL format:

```
http://20.236.198.164/ACS/servlet/ACS
```

12. Optional If you performed the tasks in Step 3, obtain the external IP address of the *csext<sname>* load balancer service using the `kubectl get svc` command, and then change the default value of *ExtCS.tcp_route* to the new external IP.

13. Run the Helm upgrade command using the following command format:

```
helm upgrade <release_name> ./dctm-server -f ./dctm-server/platforms/azure.yaml
```

14. To access Documentum Server by external clients (outside the cluster), Documentum Server and connection broker must be deployed with externalization configuration as mentioned in Step 3, Step 12, and Step 13. After the successful deployment, two load balancer services are created, one each for *csext-<sname>* and *dbrext-<sname>* having their own external IP.

    Copy the `dfc.properties` file from `/opt/dctm/config/` to your client machine from the primary Documentum Server pod. The original `dfc.properties` file contains two sets of host and port pair. Remove one set of host and port pair and specify the following for the other set:

```
dfc.docbroker.host[0]=<External IP of dbrext load balancer service>
dfc.docbroker.port[0]=<ExtCS.nativeExtPort>
```

    For example:

```
dfc.docbroker.host[0]=10.70.62.110
dfc.docbroker.port[0]=80
```

4.1.1.3 Limitations

- Host name must have the FQDN and it must not be greater than 59 characters.
- You must change the storage class according to the Azure Kubernetes service offering. The *default* storage class provisions a standard Azure disk while the *managed-premium* storage class provisions a premium Azure disk.
- Only HTTP configuration is supported for *jmsProtocol* and *tnsProtocol*.

4.1.1.4 Troubleshooting

Symptom	Cause	Fix
When you configure Azure on a Linux VM, it results in the `[root@skvoraclelinux ~]# az aks browse --resource-group dctm --name dctmaks Merged "dctmaks" as current context in /tmp/tmpdr1aC2 Unable to connect to the server: proxyconnect tcp: tls: oversized record received with length 20527` error.	Failure to connect to the server.	Use the export command as follows: `export https_proxy =<proxy_value>:<port>`

4.1.2 Red Hat OpenShift platform

4.1.2.1 Prerequisites

1. Download and configure the Docker application from the Docker website.

   Docker Documentation contains detailed information.

2. Download and configure the OpenShift CLI (OC) from the Red Hat website.

   Red Hat OpenShift Documentation contains detailed information.

3. Download the supported version of Helm package from the Helm website.

   The product *Release Notes* document contains detailed information about the supported versions.

   Helm Documentation contains detailed information.

4. Download and configure the Red Hat OpenShift platform from the Red Hat website.

   Red Hat OpenShift Documentation contains detailed information.

5. Download and configure the PostgreSQL database (server) from the PostgreSQL website.

   Note: The PostgreSQL database client is packaged with Documentum Server Docker image.

   PostgreSQL Documentation contains detailed information.

6. Create a Red Hat OpenShift cluster.

   a. Configure an Azure account.

   b. Create a resource group. A resource group in Azure is a folder to keep your collection. It does not serve any other purpose.

   c. Create a DNS zone and configure a domain name.
   d. Create and configure a service principal.

   Red Hat OpenShift Documentation contains detailed information.

7. Deploy the Red Hat OpenShift cluster.

   Red Hat OpenShift Documentation contains detailed information.

8. Create an Azure database for the PostgreSQL server to enable Postgres as a service.

9. Download and upload the Docker image into Azure. Perform the following tasks:

   a. Verify if Docker daemon or server is running on the local system or server using the following command format:

   ```
   sudo /usr/bin/dockerd --insecure-registry
   <registry_ip : registry_port>
   --insecure-registry <registry_ip : registry_port>
   -H unix:///var/run/docker.sock --init 2>&1 &
   ```

   b. Create a container registry. Container registry is used to store the Docker images in Azure. A standard container registry can store up to 100 GB of images.

   c. Obtain the login server details of the container registry from Azure using the following command format:

   ```
   docker login <login server> --username <service principal id> --password
   <service principal password>
   ```

   After the Docker login is successful, a `config.json` configuration file is created in the `.docker` folder.

   d. Use the configuration file to fetch Docker images inside the OpenShift cluster. Use the following command format:

   ```
   oc create secret generic <pull_secret_name> \ --fromfile=.
   dockerconfigjson=<path/to/.docker/config.json> \ --
   type=kubernetes.io/dockerconfigjson
   ```

   If you have not logged in, use the following command format:

   ```
   oc create secret docker-registry <pull_secret_name> \ --dockerserver=<
   registry_server> \ --docker-username=<user_name> \ --docker
   password=<password> \ --docker-email=<email>
   ```

   e. Link *`pull_secret_name`* to the service account and add it as a default service account of OpenShift cluster using the following command format:

   ```
   oc secrets link default <pull_secret_name> --for=pull
   ```

   f. Download the Docker image (Oracle Linux only) from OpenText Container Registry. Perform the following tasks:

      i. Log in to OpenText Container Registry using the following command format:

      ```
      docker login registry.opentext.com
      ```

When prompted, provide your OpenText My Support login credentials.

ii. Download the Docker image using the following command format:

```
docker pull registry.opentext.com/<image_name>:<image_tag>
```

The following Docker image is available for download:

Image name	Image tag
dctm-server	23.2.0 or 23.2

g. Tag the Docker image to the Azure registry specific image using the following command format:

```
docker tag <source image> <destination image>
```

h. Upload the Docker image to the Azure Container Registry (ACR) using the following command format:

```
docker push <image>
```

10. Download the Helm charts available in the `documentum_server_<release-version>_kubernetes_helm_<operating-system>.tar` file from OpenText My Support.

Note: The TAR file also contains Docker Compose scripts for reference along with Helm charts for the Oracle Linux operating system.

4.1.2.2 Deploying Documentum Server

1. Create a project using the following command format:

```
oc new-project <project-name>
```

2. Create a storage class.

A storage class is used to define how an Azure file share is created. A storage account can be specified in the class.

Only the `Premium_LRS` storage type is supported for shared type SSD. You must create a storage class with the *`maxshares: "2"`* option to support `ReadWriteMany` PVC required for Documentum Server.

Perform the following tasks:

a. Create a storage class using the following sample contents as a YAML file (for example, `azopenshiftstorageclassreadwritemany.yaml` with appropriate values for the variables):

```
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"allowVolumeExpansion":true,"apiVersion":"storage.k8s.io/
v1beta1","kind":"StorageClass","metadata":{"annotations":
{"storageclass.beta.kubernetes.io/is-default-class":"true"},"labels":
```

```
{"kubernete    s.io/cluster-service":"true"},"name":"default"},"parameters":
{"cachingmode":"ReadOnly","kind":"Managed","storageaccounttype":"StandardSSD_LR
S"},"provisioner":"kubernetes.io/azure-disk"}
    storageclass.beta.kubernetes.io/is-default-class: "true"
  labels:
    kubernetes.io/cluster-service: "true"
  name: <storage-class-name>
parameters:
  maxShares: "2"
  cachingmode: None
  kind: Managed
  storageaccounttype: Premium_LRS
provisioner: kubernetes.io/azure-disk
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
```

b. Create a storage class using the following sample contents for all other PVCs that require `ReadWriteOnce` access mode as a YAML file (for example, `azopenshiftstorageclassreadwriteonce.yaml` with appropriate values for the variables):

```
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"allowVolumeExpansion":true,"apiVersion":"storage.k8s.io/
v1beta1","kind":"StorageClass","metadata":{"annotations":
{"storageclass.beta.kubernetes.io/is-default-class":"true"},"labels":
{"kubernetes.io/cluster-service":"true"},"name":"default"},"parameters":
{"cachingmode":"ReadOnly","kind":"Managed","storageaccounttype":"StandardSSD_LR
S"},"provisioner":"kubernetes.io/azure-disk"}
    storageclass.beta.kubernetes.io/is-default-class: "true"
  labels:
    kubernetes.io/cluster-service: "true"
  name: <storage-class-name>
parameters:
  cachingmode: None
  kind: Managed
  storageaccounttype: Premium_LRS
provisioner: kubernetes.io/azure-disk
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
```

c. Create the storage class using the following command format:

```
oc create -f <name of the YAML file>.yaml
```

d. Set the value of *`size`* to `256Gi` and specify the name of the storage class that has the *`maxshares = “2”`* option for *`storageClass`* specified in Step 2.a in the single All-In-One `dctm-server/values.yaml` Helm chart file.

Note: For all other PVCs, provide the name of the storage class specified in Step 2.b in the single All-In-One `dctm-server/values.yaml` Helm chart file.

3. Optional If you want to externalize Documentum Server outside of a cluster, create two load balancers one each for connection broker and Documentum Server.

a. Externalize the connection broker.

i. To externalize the connection broker, save the following sample contents as a `<name of YAML file>.yaml` file and then run the `oc create` command:

```
apiVersion: v1
kind: Service
metadata:
labels:
app: <sname>dbr-nginx-ingress
name: <sname>dbr-nginx-ingress
namespace: default
spec:
ports:
- name: dbrnative
nodePort: 30523
port: 80
protocol: TCP
targetPort: 1491
- name: dbrssl
nodePort: 30524
port: 81
protocol: TCP
targetPort: 1492
selector:
app: <sname>
sessionAffinity: None
type: LoadBalancer
```

where *`<sname>`* is the name of the connection broker application name.

ii. Obtain the external IP address of the connection broker using the following command format:

```
oc get svc <sname>-nginx-ingress
```

iii. Access the connection broker using the external IP address of the connection broker for the load balancer with port 80.

b. Externalize the Documentum Server.

i. To externalize the Documentum Server, save the following sample contents as a `<name of YAML file>.yaml` file and then run the `oc create` command:

```
apiVersion: v1
kind: Service
metadata:
labels:
app: <sname>cs-nginx-ingress
name: <sname>cs-nginx-ingress
namespace: default
spec:
ports:
- name: csnative
nodePort: 30525
port: 80
protocol: TCP
targetPort: 50000
- name: csssl
nodePort: 30526
port: 81
protocol: TCP
targetPort: 50001
selector:
app: <sname>dcs
sessionAffinity: ClientIP
type: LoadBalancer
```

where *<sname>* is the name of the Documentum Server application name.

ii. Obtain the external IP address using the following command format:

```
oc get svc <sname>-nginx-ingress
```

iii. Apply the external IP address for *tcp_route* in the single All-In-One `dctm-server/values.yaml` Helm chart file.

4. When a user logs in to Red Hat OpenShift, by default, the user is added to the restricted security context constraints (SCC). For some privilege operation such as `chmod`, you must provide specific privileges to the pods. Do one of the following tasks:

   a. (Recommended) Add the default service account deployer to `anyuid` using the following command format:

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

   b. Add all authenticated users to `anyuid scc` instead of `restricted` using the following command format:

   ```
   oc adm policy add-scc-to-group anyuid system:authenticated
   ```

5. Extract the Helm charts downloaded from OpenText My Support to a temporary location.

6. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and provide the appropriate values for the variables to pass them to your templates.

   "Deploying Documentum Server" on page 49 contains detailed information about the variables.

7. Perform all the steps from Step 4 to Step 8 in "Deploying Documentum Server" on page 49 to complete the deployment and verification of the Documentum Server pod.

4.1.2.3 Limitations

There are no limitations for this release.

4.1.2.4 Troubleshooting

Symptom	Cause	Fix
When you try to describe pod, it results in the "`{chmod: changing permissions of '/var/lib/postgresql/data': Operation not permitted}`" error.	The pod does not have sufficient permissions to run privileged commands.	Provide the proper SCC to the user.

Symptom	Cause	Fix
When you run the `VolumeBinding prebind` plug-in for the pod, it results in the `Warning FailedScheduling <invalid> default-scheduler, Failed to bind volumes: provisioning failed for PVC "dbr-vct-testdocbroker-0"` error.	PVC unable to bind.	Verify if you are able to create PVC using `assign storage class` or set the value for *`pvcAccessModes`* to `ReadWriteMany` and the value for PVC *`size`* to `256Gi`.

When you try to deploy the connection broker or Documentum Server pod, it results in the `Warning FailedScheduling 4m26s default-scheduler 0/5 nodes are available: 2 node(s) exceed max volume count, 3 node(s) had taint {node-role. kubernetes.io/master: }, that the pod didn't tolerate.` error.	Node does not have sufficient volume disk resource. By default, OpenShift cluster creates node with size as Standard_D2s_v3 for worker node that has a maximum four disk volume capacity.	Do one of the following tasks:
		• Edit the `machinset` YAML files and deploy the size as `Standard_ D32s_v 3` using the following command format:
		`oc scale -- replicas=0 machineset <machinesetname>`
		`oc edit machineset <machinesetname>`
		`oc scale -- replicas=1 machineset <machinesetname>`
		• Increase the node resource.

The `oc get` command fails.	Pending certificate signing request (CSR) for approval.	Verify if any CSRs are pending using the following command foramt:
		`oc get csr`
		Approve the pending CSR (if any) using the following command format:
		`oc get csr -o name | xargs`
		`oc adm certificate approve`

When you try to perform install cluster, it results in the `DEBUG Still waiting for the Kubernetes API: Get "https:// api.<clustername>.<domain name>: 6443/ version?timeout=32s": Service Unavailable` error.	Firewall blocks the communication.	Provide access for both the port and host in the HTTPS URL.

4.2 Deploying and configuring Documentum Administrator on Azure cloud platform

4.2.1 Kubernetes platform

4.2.1.1 Prerequisites

1. Perform all the steps as described in "Prerequisites" on page 161 in "Deploying and configuring Documentum Server on Microsoft Azure cloud platform" on page 161 except for those steps related to the PostgreSQL database.
2. Download the Docker image from OpenText Container Registry. Perform the following tasks:
   a. Log in to OpenText Container Registry using the following command format:
   ```
   docker login registry.opentext.com
   ```
   When prompted, provide your OpenText My Support login credentials.
   b. Download the Docker image using the following command format:
   ```
   docker pull registry.opentext.com/<image_name>:<image_tag>
   ```
   The following Docker image is available for download:

Image name	Image tag
dctm-admin	23.2.0 or 23.2

   Note: Download the Docker Compose file from OpenText My Support.
3. Upload the Docker image into the Kubernetes cluster.

4.2.1.2 Deploying Documentum Administrator

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.
2. Perform the tasks mentioned from Step 2 to Step 5 in "Deploying Documentum Server" on page 49 in "Deploying and configuring Documentum Server on private cloud" on page 47.
3. Perform the task mentioned in Step 3 in "Deploying Documentum Administrator" on page 104 in "Deploying and configuring Documentum Administrator on private cloud" on page 104.
4. Update the ingress resource rule for Documentum Administrator in the `dctm-ingress` category.

   For example:

```
daService:
    enable: true
    serviceName: <da-service-name>
    servicePort: 8080
```

5. Deploy the Documentum Server and Documentum Administrator Helm charts using the following command format:

   ```
   helm install <release_name> <location where Helm charts are extracted>/dctm-server -
   f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
   platform>.yaml --namespace <name of namespace>
   ```

   For example:

   ```
   helm install dctm-server /opt/temp/Helm-charts/dctm-server -f /opt/temp/Helm-charts/
   dctm-server/platforms/azure.yaml --namespace docu
   ```

6. Verify the status of the deployment of Documentum Server and Documentum Administrator Helm charts using the following command format:

   ```
   helm status <release_name>
   ```

7. Verify the status of the deployment of the Documentum Server and Documentum Administrator pods using the following command format:

   ```
   kubectl describe pods <name of the pod>
   ```

8. Obtain the external IP address of the load balancer service of the NGINX Ingress controller using the following command format:

   ```
   kubectl get svc | grep <name of the ingress controller>
   ```

9. Access the Documentum Administrator application using the external IP address of the NGINX Ingress controller load balancer service in a browser using the following URL format:

   ```
   http://<external IP address of NGINX Ingress controller load balancer service>/
   <Documentum ingress resource>
   ```

4.2.1.3 Limitations

There are no limitations for this release.

4.2.1.4 Troubleshooting

There are no troubleshooting information for this release.

4.3 Deploying and configuring Documentum Records Client on Azure cloud platform

4.3.1 Kubernetes platform

4.3.1.1 Prerequisites

1. Perform all the steps as described in “Prerequisites” on page 161 in “Deploying and configuring Documentum Server on Microsoft Azure cloud platform” on page 161.
2. Deploy the Documentum Server pod as described in “Deploying and configuring Documentum Server on private cloud” on page 47.
3. Download the Docker image (Oracle Linux only) from OpenText Container Registry. Perform the following tasks:

   a. Log in to OpenText Container Registry using the following command format:

   ```
   docker login registry.opentext.com
   ```

   When prompted, provide your OpenText My Support login credentials.

   b. Download the Docker image using the following command format:

   ```
   docker pull registry.opentext.com/<image_name>:<image_tag>
   ```

   The following Docker images are available for download:

Image name	Image tag
dctm-records	23.2.0 or 23.2
dctm-records-darinstallation	23.2.0 or 23.2
dctm-rqm	23.2.0 or 23.2

4. Download the Helm charts from OpenText My Support.

4.3.1.2 Deploying Documentum Records Client

The information for deploying and configuring Documentum Records Client on Azure cloud platform is same as described in “Deploying and configuring Documentum Server on Microsoft Azure cloud platform” on page 161.

Make sure that you deploy the Documentum Records Client Docker image (Oracle Linux only) and Helm charts.

4.3.1.3 Limitations

- Host name must have the fully qualified domain name (FQDN) and must not be greater than 59.
- You must change the storage class according to the Azure Kubernetes service offering. The *default* storage class provisions a standard Azure disk while the *managed-premium* storage class provisions a premium Azure disk.

4.3.1.4 Troubleshooting

Symptom	Cause	Fix
When you configure Azure on a Linux VM, it results in the `[root@skvoraclelinux ~]# az aks browse --resource-group recordsrg --name recordsaks Merged "recordsaks" as current context in /tmp/tmpdr1aC2 Unable to connect to the server: proxyconnect tcp: tls: oversized record received with length 20527` error.	Failure to connect to the server.	Use the export command as follows: `export https_proxy=<proxy_value>:<port>`

4.4 Deploying and configuring Documentum REST Services on Azure cloud platform

4.4.1 Kubernetes platform

4.4.1.1 Prerequisites

Perform all the steps as described in "Prerequisites" on page 161 in "Deploying and configuring Documentum Server on Microsoft Azure cloud platform" on page 161.

4.4.1.2 Deploying Documentum REST Services

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.
2. Perform the tasks mentioned from Step 2 to Step 5 in "Deploying Documentum Server" on page 49 in "Deploying and configuring Documentum Server on private cloud" on page 47.
3. Perform the task mentioned in Step 5 in "Deploying Documentum REST Services" on page 135 in "Deploying and configuring Documentum REST Services on private cloud" on page 134.
4. Update the ingress resource rule for Documentum REST Services in the `dctm-ingress` category.

   For example:

   ```
   restService:
       enable: true
       serviceName: <sname>-rest
       servicePort: 8080
   ```
5. Deploy the Documentum Server and Documentum REST Services Helm charts using the following command format:

   ```
   helm install <release_name> <location where Helm charts are extracted>/dctm-server -
   f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
   platform>.yaml --namespace <name of namespace>
   ```

   For example:

   ```
   helm install dctm-server /opt/temp/Helm-charts/dctm-server -f /opt/temp/Helm-charts/
   dctm-server/platforms/azure.yaml --namespace docu
   ```
6. Verify the status of the deployment of Documentum Server and Documentum REST Services Helm charts using the following command format:

   ```
   helm status <release_name>
   ```
7. Verify the status of the deployment of the Documentum Server and Documentum REST Services pods using the following command format:

   ```
   kubectl describe pods <name of the pod>
   ```
8. Obtain the external IP address of the load balancer service of the NGINX Ingress controller using the following command format:

   ```
   kubectl get svc | grep <name of the ingress controller>
   ```
9. Access the Documentum REST Services application using the external IP address of the NGINX Ingress controller load balancer service in a browser using the following URL format:

   ```
   http://<external IP address of NGINX Ingress controller load balancer service>/
   <Documentum ingress resource>
   ```

4.4.1.3 Configuring external IP address

The information for configuring external IP address on the Azure cloud platform is same as described in “Deploying Documentum Server” on page 164.

4.4.1.4 Limitations

There are no limitations for this release.

4.4.1.5 Troubleshooting

There are no troubleshooting information for this release.

4.4.2 Red Hat OpenShift platform

4.4.2.1 Prerequisites

The information for prerequisites is same as described in “Prerequisites” on page 177 in “Kubernetes platform” on page 177.

4.4.2.2 Deploying Documentum REST Services

The information for deploying and configuring Documentum REST Services in Red Hat OpenShift cloud platform is same as described in “Deploying Documentum REST Services” on page 178.

4.4.2.3 Configuring external IP address

The information for configuring external IP address on the Azure cloud platform is same as described in “Deploying Documentum Server” on page 164.

4.4.2.4 Limitations

There are no limitations for this release.

4.4.2.5 Troubleshooting

There are no troubleshooting information for this release.

4.5 Deploying and configuring Content Connect on Azure cloud platform

4.5.1 Prerequisites

1. Perform all the steps as described in the *Deploying and configuring Documentum Server on Microsoft Azure cloud platform* > *Prerequisites* section.
2. Download the Docker image (Oracle Linux only) from OpenText Container Registry. Perform the following tasks:
   a. Log in to OpenText Container Registry using the following command format:
   ```
   docker login registry.opentext.com
   ```
   When prompted, provide your OpenText My Support login credentials.

   b. Download the Docker image using the following command format:
   ```
   docker pull registry.opentext.com/<image_name>:<image_tag>
   ```
   The following Docker images are available for download:

Image name	Image tag
dctm-content-connect	23.2.0 or 23.2
dctm-content-connect-dbinit	23.2.0 or 23.2

3. Download the Helm charts:
   - For Documentum Server, the Helm charts are available in the `documentum_server_<releaseversion>_ kubernetes_helm_<operating-system>.tar` file from the *OpenText My Support* > *Documentum Server* > *Software Download* page.

4.5.2 Deploying Content Connect

To deploy Content Connect on Azure cloud platform, perform the following steps:

1. The information for deploying and configuring Content Connect on Azure cloud platform is same as described in *Deploying and configuring Documentum Server on Microsoft Azure cloud platform*.

   Make sure that you deploy the Content Connect Docker image (Oracle Linux only).
2. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and in the contentconnect category, provide the appropriate values for the variables to pass them to your templates.

   The "contentconnect" on page 145 table contains detailed information about the variables.

4.5.2.1 Deploying Content Connect on the client machine

To deploy Content Connect to the client machine, perform the following steps:

1. After the Admin Console URL is up, add the endpoint details and download the manifest file.
2. Use the manifest file to add the Content Connect add-in to the Word or Outlook application. For more information, see *OpenText Content Connect Installation and Administration Guide*.

4.5.2.2 Deploying only Content Connect using single Helm

4.5.2.2.1 Documentum Server single Helm

To deploy only Content Connect for Documentum Server in single Helm, perform the following steps in the dctm-server/values.yaml file:

1. Except for `cs-secrets` and `contentconnect` categories, disable all the product deployment categories.
2. Update the `database: userName` and `password`.
3. In the `contentconnect` category, provide all the required values.

   The “contentconnect” on page 145 table contains detailed information about the variables.
4. Deploy Content Connect using the same command as Documentum Server deployment.

4.5.2.2.2 Limitations

There are no limitations for this release.

4.5.2.2.3 Troubleshooting

Symptom	Cause	Fix
When you configure Azure on a Linux VM, it results in the [root@skvoraclelinux ~]# az aks browse --resource-group >See_genenties_File!<rg --name >See_genenties_File!<aks Merged “>See_genenties_File!<aks” as current context in /tmp/tmpdr1aC2 Unable to connect to the server: proxyconnect tcp: tls: oversized record received with length 20527 error.	Failure to connect to the server.	Use the export command as follows: `export https_proxy=<proxy_value>:<port>`

Symptom	Cause	Fix
When using Content Connect, Cross-Origin Resource Sharing (CORS) related errors occur even after all configurations are set.	CORS related erros occur.	• Increase the load balancer response timeout. • Add the Content Connect Admin Console URL to the `rest.cors.allowed.origins` tag in the `rest-api-runtime.properties` file.

4.5.2.2.4 Upgrading

Upgrade the Content Connect pod using the `helm upgrade` command:

```
Helm upgrade <release name> . --values <resource file name> -n <namespace>
```

4.6 Deploying Documentum Server with Azure PaaS

You can deploy Documentum Server with both `Azure Database for PostgreSQL - Single server` and `Azure Database for PostgreSQL - Flexible server`.

4.6.1 Deploying Documentum Server with Azure Database for PostgreSQL - Single server

You must update the values of all the required variables in the single All-In-One `dctm-server/values.yaml` Helm chart file to deploy the Documentum Server pod. In addition, perform the following tasks:

1. Specify the value of *`database.userName`* in the `<database admin user>@<host name created for your database>` format in the `cs-secrets` category (“cs-secrets” on page 55).

2. If you want to use the existing repository, specify the value of *`database.userName`* in the `<repository owner name>@<host name created for your database>` format in the `cs-secrets` category (“cs-secrets” on page 55).

   In addition, make sure that you set the value of *`docbase.existing`* to `true` in the `content-server` category (“content-server” on page 66).

3. If you want certificate-based communication in the database, set the value of *`database.sslEnabled`* to `true` in the `content-server` category (“content-server” on page 66).

4. Set the value of *`database.paasEnv`* to `true` in the `content-server` category (“content-server” on page 66).

5. Do not specify the value of *`docbase.owner`* in the `@<host name created for the database>` format in the `content-server` category (“content-server” on page 66).

4.6.2 Deploying Documentum Server with Azure Database for PostgreSQL - Flexible server

You must update the values of all the required variables in the single All-In-One `dctm-server/values.yaml` Helm chart file to deploy the Documentum Server pod. In addition, perform the following tasks:

1. Specify the value of *`database.userName`* in the `<database administrator user>` format in the `cs-secrets` category (“cs-secrets” on page 55).
2. If you want to use the existing repository, specify the value of *`database.userName`* in the `<repository owner name>` format in the `cs-secrets` category (“cs-secrets” on page 55).

   In addition, make sure that you set the value of *`docbase.existing`* to `true` in the `content-server` category (“content-server” on page 66).
3. Copy the Flexible server certificate from the Azure website and provide it as a value for *`database.certificate`* in the `cs-secrets` category (“cs-secrets” on page 55).
4. Set the value of *`database.sslEnabled`* to `true` in the `content-server` category (“content-server” on page 66).
5. Set the value of *`database.paasEnv`* to `false` in the `content-server` category (“content-server” on page 66).

4.7 Configuring Documentum Server for Azure OAuth authentication

Documentum Server supports Azure OAuth 2.0 authentication. Perform the following tasks:

1. In Documentum Server deployed on Azure, run the following IAPI commands:

```
API> retrieve,c,dm_server_config
...
3d0004d280000102
API> append,c,l,app_server_name
SET> oAuthAuthentication
...
OK
API> append,c,l,app_server_uri
SET> http://localhost:9080/oAuthAuthentication/servlet/authenticate
...
OK
API> save,c,l
```

2. Open the `oauth.properties` file located at `$DM_JMS_HOME/webapps/OTDSAuthentication/WEB-INF/classes`.
3. Retain the default values of the following parameters:

   - *`azure_enable`*=`true`

- *azure_openid_url*=`https://login.microsoftonline.com/common/.well-known/openid-configuration`

4. The *azure_dctm_server_app_id* parameter in `oauth.properties` is the Documentum OAuth client ID configured in Azure. The default value is blank and you can keep the value as blank. However, OpenText recommends that you provide the configured Documentum OAuth client ID as the value for this parameter for enhanced security.

5. Save the `oauth.properties` file.

Note: Documentum Server supports authentication only using the OAuth token and does not support the authorization. In addition, the Client Credentials grant type is not supported.

Chapter 5

Documentum Platform and Platform Extensions applications on Google cloud platform

The product *Release Notes* document contains detailed information about the list of applications and its supported versions for Google Cloud Platform (GCP).

5.1 Deploying and configuring Documentum Server on GCP

5.1.1 Kubernetes platform

5.1.1.1 Prerequisites

1. Download and configure the Docker application from the Docker website.

   Docker Documentation contains detailed information.

2. Download the supported version of Helm package from the Helm website.

   The product *Release Notes* document contains detailed information about the supported versions.

   Helm Documentation contains detailed information.

3. Download and configure the Kubernetes application from the Kubernetes website.

   Kubernetes Documentation contains detailed information.

4. Download and configure the PostgreSQL database (server) from the PostgreSQL website.

   Note: The PostgreSQL database client is packaged with Documentum Server Docker image.

   PostgreSQL Documentation contains detailed information.

5. Download and install the latest version of Google Cloud SDK from the Google Cloud Platform (GCP) website on your machine which includes gcloud command line utility.

6. Create a Google Kubernetes Engine (GKE) and a namespace within the cluster.

   a. From the GCP website, select the GCP project linked with your corporate billing account.
   b. Create a cluster. Navigate to **Kubernetes Engine > Clusters**, click **CREATE CLUSTER**, and perform the following tasks:

- Select a standard cluster template.
- Provide a name to your cluster.

  For example, demo-cluster.
- Select Zonal for the location type. You can use Regional for *production grade - high available* clusters.
- Select a Zone closer to your location. The GCP website contains the list of zones.
- Review the number of nodes for your cluster and the machine type for each of your nodes.

  For example, three nodes with 2vCPUs/7.5 GB memory for each node.

Click **Create**.

c. Click **Connect** next to the cluster you created and then click **Run in Cloud Shell**.

A Google Cloud shell (a VM created by Google with pre-installed kubectl and gcloud SDK) is created.

d. Press Enter at the command that shows up.

Cluster credentials are fetched and creates a *`kubeconfig`* entry (the `kubectl` configuration file).

e. Create a Kubernetes cluster namespace on the Cloud shell using the following command format:

```
kubectl create namespace <name of namespace>
```

f. Verify if the namespace is created using the following command format:

```
kubectl get namespace
```

7. Download the Documentum Server and the required Documentum application Docker images (Oracle Linux only) from OpenText Container Registry. Perform the following tasks:

a. Log in to OpenText Container Registry using the following command format:

```
docker login registry.opentext.com
```

When prompted, provide your OpenText My Support login credentials.

b. Download the Docker image using the following command format:

```
docker pull registry.opentext.com/<image_name>:<image_tag>
```

The following Docker images are available for download:

Image name	Image tag
dctm-server	23.2.0 or 23.2
dctm-admin	23.2.0 or 23.2

Image name	Image tag
dctm-dfs	23.2.0 or 23.2
dctm-records	23.2.0 or 23.2
dctm-records-darinstallation	23.2.0 or 23.2
dctm-rqm	23.2.0 or 23.2
dctm-rest	23.2.0 or 23.2

8. Upload the Docker image(s) to Google Container Registry (GCR). Perform the following tasks:

   a. Configure Docker to use gcloud as a credential helper using the following command format:

   ```
   gcloud auth configure-docker
   ```

   b. Tag your Docker image(s) using the following command format:

   ```
   [HOSTNAME]/[PROJECT-ID]/[IMAGE]
   ```

   For example:

   ```
   gcr.io/documentum-d2-product/dctm-server:<version>
   ```

   c. Upload the tagged Docker image(s) to GCR. For example:

   ```
   docker push gcr.io/documentum-d2-product/dctm-server:<version>
   ```

9. Extract the Helm package you downloaded in Step 2 to a temporary location.

10. Verify the version of downloaded Helm package using the following command format:

    ```
    abc@cloudshell:~ (dctm-server)$ ./helm version
    ```

11. Create a Google Cloud Filestore instance.

    In **Filestore** of GCP dashboard, click **CREATE INSTANCE**, and perform the following tasks:

    - Provide a name to the Google Cloud Filestore instance.

      For example, demo-gcfss.
    - Select a standard cluster template.
    - Select the default authorized network.
    - Select your region and zone for the **Location** type for better performance.
    - Select the NFS mount point for the **Fileshare name**.

      For example, demogcfs.
    - Provide a minimum of 1 TB for **Fileshare Capacity**.

    Google Cloud Filestore need not be created every time. The Google Cloud Filestore instance can be shared by multiple clusters.

After the Google Cloud Filestore instance is created, click the instance and note the IP address and path of the instance.

Note: The instance tier of the Google Cloud Filestore instance cannot be modified after it is created. However, the **Fileshare Capacity** can be modified.

12. Deploy an external storage provisioner (nfs-client-provisioner) for dynamic provisioning of ReadWriteMany (RWX) Persistent Volumes (PVs) on Google Cloud Filestore instance.

    a. Download the external nfs-client-provisioner Helm chart from the Github website to your Cloud Shell machine.

    b. Install the nfs-client-provisioner Helm chart: Provide the `nfs.server` value as the IP address of the Google Cloud Filestore instance, the `nfs.path` value as the path given in the Google Cloud Filestore instance and `storageClass.name` value.

       For example, gcp-rwx.ReadWriteMany PVCs should be created with this storageClassName for ReadWriteMany Persistent Volumes.

       Example output:

```
abc@cloudshell:~ (dctm-d2)$ ./helm install
demo-nfs-client-provisioner ./nfs-client-provisioner/
--namespace demo
--set nfs.server="10.91.118.66"
--set nfs.path=/demogcfs
--set storageClass.name=gcp-rwx
NAME:   demo-nfs-client-provisioner
LAST DEPLOYED: Sun Jun  9 11:52:05 2019
NAMESPACE: demo
STATUS: DEPLOYED

RESOURCES:
==> v1/Deployment
NAME
demo-nfs-client-provisioner
AGE
1s
==> v1/Pod(related)
NAME
demo-nfs-client-provisioner-76986844 0/1...
READY...
0/1...
==> v1/StorageClass

NAME
gcp-rwx
AGE
1s
==> v1/ServiceAccount
demo-nfs-client-provisioner 1s

==> v1/ClusterRole
demo-nfs-client-provisioner-runner 1s

==> v1/ClusterRoleBinding
run-demo-nfs-client-provisioner  1s

==> v1/Role
leader-locking-demo-nfs-client-provisioner 1s
```

```
==> v1/RoleBinding
leader-locking-demo-nfs-client-provisioner 1s
```

13. Create a sample PVC (for example, `testPVC.yaml`) to test the dynamic provisioning of ReadWriteMany (RWX) PVs using the following command format:

    ```
    kind: PersistentVolumeClaim
    apiversion: v1
    metadata:
    name: test-claim
    spec:
    accessModes:
    - ReadWriteMany
    storageClassName: gcp-rwx
    resources:
    requests:
    storage: 1Mi
    ```

14. Verify if the corresponding PV is created using the following command format:

    ```
    kubectl get pv
    ```

    Make sure that the status of PV is `Bound`.

15. Delete the PVC and make sure that the PV is also deleted.

16. Download the Helm charts available in the `documentum_server_<release-version>_kubernetes_helm_<operating-system>.tar` file from OpenText My Support.

    Note: The TAR file also contains Docker Compose scripts for reference along with Helm charts for the Oracle Linux operating system.

5.1.1.2 Deploying Documentum Server

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.

2. Update the image and tag fields in the single All-In-One `dctm-server/values.yaml` Helm chart file to point to the GCR images.

3. Update the *`storageClass`* fields in the single All-In-One `dctm-server/values.yaml` Helm chart file of PV with `ReadWriteMany` access mode and VCT with `ReadWriteOnce` access mode.

4. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and provide the appropriate values for the variables to pass them to your templates.

   "Deploying and configuring Documentum Server on private cloud" on page 47 contains detailed information about the variables.

5. Perform all the steps from Step 4 to Step 8 in "Deploying Documentum Server" on page 49 to complete the deployment and verification of the Documentum Server pod.

6. Download the NGINX (Ingress Controller) Helm from the Github website.

   Ingress consists of two components:

- Ingress Resource: Collection of rules for the inbound traffic to reach Services. These are Layer 7 (L7) rules that allow host names (and optionally paths) to be directed to specific Services in Kubernetes.
- Ingress Controller: Acts upon the rules set by the Ingress Resource, typically via an HTTP or L7 load balancer.

7. Deploy the NGINX Ingress Controller Helm using the following command format:

```
helm install <name of namespace>-nginx-ingress
<location where Helm charts are extracted>
--set rbac.create=true
--namespace <name of namespace>
```

   For example:

```
helm install demo-nginx-ingress stable/nginx-ingress
--set rbac.create=true
--namespace demo

NAME: demo-nginx-ingress
LAST DEPLOYED: Sun Jun 9 15:48:55 2019
NAMESPACE: demo
STATUS: DEPLOYED

RESOURCES:
==> v1beta1/ClusterRoleBinding
NAME                AGE
demo-nginx-ingress  0s

==> v1beta1/Role
demo-nginx-ingress  0s

==> v1/Service
demo-nginx-ingress-controller       0s
demo-nginx-ingress-default-backend  0s

==> v1/ConfigMap
demo-nginx-ingress-controller  0s

==> v1/ServiceAccount
demo-nginx-ingress  0s

==> v1beta1/ClusterRole
demo-nginx-ingress  0s

==> v1beta1/RoleBinding
demo-nginx-ingress  0s

==> v1beta1/Deployment
demo-nginx-ingress-controller       0s
demo-nginx-ingress-default-backend  0s

==> v1/Pod(related)
...
NAME                                                READY...
demo-nginx-ingress-controller-78cd47cf46-cw9q2      0/1...
demo-nginx-ingress-default-backend-5d47879fb7-5lptf  0/1...
```

   The nginx-ingress controller is installed. It may take a few minutes for the load balancer IP to be available.

8. Verify the status of the NGINX Ingress Controller Helm deployment using the following command format:

```
kubectl --namespace <name of namespace> get services -o wide -w demo-nginx-ingress-
controller
```

Example ingress that takes control of the controller:

```
apiVersion: extensions/vibeta1
kind: Ingress
metadata:
annotations:
kubernetes.io/ingress.class: nginx
name: example
namespace: test
spec:
rules:
- host: www.example.com
http:
paths:
- backend:
serviceName: exampleService
serviceport: 80
path: /

The following code is required only if TLS is enabled for the Ingress:
tls:
- hosts:
- www.example.com
secretName: example-tls
```

9. Optional If TLS is enabled for the Ingress, you must provide a Secret containing the certificate and key using the following command format:

```
apiversion: v1
kind: Secret
metadata:
name: example-tls
namespace: test
data:
tls.cert: <Base64 encoded certificate>
tls.key: <Base64 encoded key>
type: kubernetes.io/tls
```

10. Install a single host simple fan-out Ingress resource Helm to route the traffic to the cluster-internal services using the following command format:

```
helm install <name of Ingress resource> ./helm-charts-demo/documentum-ingress --
namespace <name of namespace>
```

For example:

```
NAME: dctm-common-ingress
LAST DEPLOYED: Sun Jun 9 15:55:47 2019
NAMESPACE: demo
STATUS: DEPLOYED

RESOURCES:
==> v1beta1/Ingress
NAME                 AGE
dctm-common-ingress  0s
```

11. The Ingress Resource determines the controller that is utilized to serve traffic. Set the Ingress annotation to select the NGINX ingress controller. This is set with an annotation, `kubernetes.io/ingress.class`, in the metadata section of the Ingress Resource.

For example:

```
annotations:
kubernetes.io/ingress.class nginx
```

12. Update the `dctm-ingress/templates/ingress.yaml` file.

    Sample `ingress.yaml` file with example values:

```
ingressPrefix: <prefix>
ingress:
  host: <ingress-host>
  clusterDomainName: <DNS name of the cluster>
  annotations:
    nginx.ingress.kubernetes.io/proxy-body-size: "0"
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/affinity: cookie
    nginx.ingress.kubernetes.io/session-cookie-hash: sha1
    nginx.ingress.kubernetes.io/proxy-connect-timeout: "60"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "60"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "60"
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    nginx.org/hsts: "false"


jmsService:
  enable: false
  serviceName: <jms-service-name>
  servicePort: 9080

jmsBase:
  enable: false
  serviceName: <jms-service-name>
  servicePort: 9080

acsService:
  enable: false
  serviceName: <jms-service-name>
  servicePort: 9080

tnsService:
  enable: false
  serviceName: <tns-service-name>
  servicePort: 8081

d2clientService:
  enable: false
  serviceName: <d2client-service-name>
  servicePort: 8080

d2configService:
  enable: false
  serviceName: <d2config-service-name>
  servicePort: 8080

daService:
  enable: false
  serviceName: <da-service-name>
  servicePort: 8080

restService:
  enable: false
  serviceName: <dctm-rest-service-name>
  servicePort: 8080

appHostService:
  enable: false
  serviceName: <apphost-service-name>
  servicePort: 8080

bamService:
  enable: false
  serviceName: <bam-service-name>
  servicePort: 8080

bpsService:
  enable: false
  serviceName: <bps-service-name>
  servicePort: 8080

xdaService:
  enable: false
  serviceName: <xda-service-name>
  servicePort: 7000

dsearchadminService:
  enable: false
  serviceName: <dsearchadmin-service-name>
  servicePort: 9300

indexagentService:
  enable: false
  serviceName: <indexagent-service-name>
  servicePort: 9200

dfsService:
  enable: false
  serviceName: <dfs-service-name>
  servicePort: 8080

webtopService:
  enable: false
  serviceName: <webtop-service-name>
  servicePort: 8080

dmotdsrestService:
  enable: false
  serviceName: <dmotdsrest-service-name>
  servicePort: 9080

otdsadmin:
  enable: false
  serviceName: <otdsServiceName>
  servicePort: 80

otdsws:
  enable: false
  serviceName: <otdsServiceName>
  servicePort: 80

otdstenant:
  enable: false
  serviceName: <otdsServiceName>
  servicePort: 80

tls:
  enable: false
  secretName: cs-secret-config
```

13. Perform the following to access the Documentum applications outside of a cluster:

    a. Obtain the external IP address of the load balancer service of the NGINX Ingress controller using the following command format:

    ```
    kubectl get svc | grep <name of the ingress controller>
    ```

    b. Access the Documentum application using the external IP address of the NGINX Ingress controller load balancer service in a browser using the following URL format:

```
http://<external IP address of NGINX Ingress controller load balancer service>/
<Documentum application>
```

If you are trying to access the Documentum Webtop application in a browser, use the following URL format:

```
http://<external IP address of NGINX Ingress controller load balancer service>/
webtop
```

Depending on the Ingress resource rule you have set in Step 12 for accessing Documentum Webtop, the URL redirects you to the Documentum Webtop login page.

5.1.1.3 Limitations

- External storage provisioners limitation: For deploying Documentum products, you need the capability to dynamically provision both ReadWriteOnce (RWO) Persistent Volumes (PVs) and ReadWriteMany (RWX) Persistent Volumes (PVs). Cloud providers provide the built-in provisioner to dynamically provision the ReadWriteOnce Persistent Volumes.

  For example, in GCP, if you specify the storageclass as standard in the PVC, then GCP automatically creates a PV of requested size using Google Compute Engine Persistent Disk. However, the Google Compute Engine Persistent Disk does not support ReadWriteMany operation, and hence you have to provision a Google Cloud Filestore instance and external provisioner to dynamically manage the Persistent Volumes for ReadWriteMany PVCs.
- In order to achieve ingress in GCP or GKE Kubernetes cluster, GCP Google Cloud Load Balancer (GCLB) L7 Load balancer is not used as this GCP GCBL L7 load balancer does not communicate to services of the ClusterIP type on the backend. Use NGINX Ingress controller to achieve Ingress in a GKE cluster. *Google Documentation* contains detailed information about Ingress with NGINX controller on GKE.

5.1.1.4 Troubleshooting

There are no troubleshooting information for this release.

5.2 Deploying and configuring Documentum Administrator on GCP

5.2.1 Kubernetes platform

5.2.1.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 185 in “Deploying and configuring Documentum Server on GCP” on page 185 except for those steps related to the PostgreSQL database.

5.2.1.2 Deploying Documentum Administrator

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.

2. Perform the tasks mentioned from Step 2 to Step 5 in “Deploying Documentum Server” on page 49 in “Deploying and configuring Documentum Server on private cloud” on page 47.

3. Perform the task mentioned in Step 3 in “Deploying Documentum Administrator” on page 104 in “Deploying and configuring Documentum Administrator on private cloud” on page 104.

4. Update the ingress resource rule for Documentum Administrator in the `dctm-ingress` category.

   For example:

   ```
   daService:
       enable: true
       serviceName: <da-service-name>
       servicePort: 8080
   ```

5. Deploy the Documentum Server and Documentum Administrator Helm charts using the following command format:

   ```
   helm install <release_name> <location where Helm charts are extracted>/dctm-server -
   f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
   platform>.yaml --namespace <name of namespace>
   ```

   For example:

   ```
   helm install dctm-server /opt/temp/Helm-charts/dctm-server -f /opt/temp/Helm-charts/
   dctm-server/platforms/gcp.yaml --namespace docu
   ```

6. Verify the status of the deployment of Documentum Server and Documentum Administrator Helm charts using the following command format:

   ```
   helm status <release_name>
   ```

7. Verify the status of the deployment of the Documentum Server and Documentum Administrator pods using the following command format:

   ```
   kubectl describe pods <name of the pod>
   ```

8. Obtain the external IP address of the load balancer service of the NGINX Ingress controller using the following command format:

   ```
   kubectl get svc | grep <name of the ingress controller>
   ```

9. Access the Documentum Administrator application using the external IP address of the NGINX Ingress controller load balancer service in a browser using the following URL format:

   ```
   http://<external IP address of NGINX Ingress controller load balancer service>/
   <Documentum ingress resource>
   ```

5.2.1.3 Limitations

There are no limitations for this release.

5.2.1.4 Troubleshooting

There are no troubleshooting information for this release.

5.3 Deploying and configuring Documentum Records Client on GCP

5.3.1 Kubernetes platform

5.3.1.1 Prerequisites

1. Perform all the steps as described in "Prerequisites" on page 185 in "Deploying and configuring Documentum Server on GCP" on page 185.

2. Deploy the Documentum Server pod as described in "Deploying and configuring Documentum Server on GCP" on page 185.

3. Download the Docker image (Oracle Linux only) from OpenText Container Registry. Perform the following tasks:

   a. Log in to OpenText Container Registry using the following command format:

      ```
      docker login registry.opentext.com
      ```

      When prompted, provide your OpenText My Support login credentials.

   b. Download the Docker image using the following command format:

      ```
      docker pull registry.opentext.com/<image_name>:<image_tag>
      ```

   The following Docker images are available for download:

Image name	Image tag
dctm-records	23.2.0 or 23.2
dctm-records-darinstallation	23.2.0 or 23.2

Image name	Image tag
dctm-rqm	23.2.0 or 23.2

4. Download the Helm charts from OpenText My Support.

5.3.1.2 Deploying Documentum Records Client

1. Upload the Documentum Records Client Docker image to Google Container Registry. Perform the following tasks:

   a. Tag your Docker image using the following command format:

   ```
   [HOSTNAME]/[PROJECT-ID]/[IMAGE]
   ```

   For example:

   ```
   gcr.io/documentum-d2-product/records:<release-version>
   ```

   b. Upload the tagged Docker image to GCR.

   For example:

   ```
   docker push gcr.io/documentum-d2-product/records:<release-version>
   ```

2. Extract the Helm charts downloaded from OpenText My Support to a temporary location.

3. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and in the `records` category, provide the appropriate values for the variables to pass them to your templates.

   "Deploying and configuring Documentum Records Client on private cloud" on page 119 contains detailed information about the variables.

   For example:

   a. Update the image and tag fields in the `values.yaml` files of your Helm charts to point to the GCR images.

   ```
   images:
   repository: gcr.io
   records:
   name: <name of Documentum product>/records
   tag: <build number/version of Documentum product>
   ```

   b. Update the ***`storageClass`*** fields in the `values.yaml` files of PV with `ReadWriteMany` access mode and VCT with `ReadWriteOnce` access mode using the following command format:

   ```
   persistentVolume:
   wtdataPVCName: records-data-pvc
   pvcAccessModes: ReadWriteMany
   size: 3 Gi
   volumeClaimTemplate:
   vctName: documentum-vct
   vctAccessModes: ReadWriteOnce
   size: 1 Gi
   storageclass: standard
   ```

4. Deploy the Documentum Records Client Helm using the following command format:

```
helm install --name <release name> <location where Helm charts are extracted>/dctm-
server -f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
platform>.yaml --namespace <name of namespace>
```

For example:

```
helm install --name records /opt/temp/Helm-charts/dctm-server -f <location where
Helm charts are extracted>/dctm-server/platforms/gcp.yaml --namespace docu
```

Note: Deploy the Documentum Records Client in the same namespace where the PostgreSQL database and Documentum Server are installed.

5. Verify the status of the deployment of Documentum Records Client Helm using the following command format:

```
helm status <release name>
```

6. Verify the status of the deployment of Documentum Records Client pod using the following command format:

```
kubectl describe pods <name of the pod>
```

7. Enable the ingress resource rule for Documentum Records Client using the following command format:

```
kubectl apply -f ingress.yaml
```

8. Obtain the external IP address of the load balancer service of the NGINX Ingress controller using the following command format:

```
kubectl get svc | grep <name of the ingress controller>
```

9. Access the Documentum Records Client application using the external IP address of the NGINX Ingress controller load balancer service in a browser using the following URL format:

```
http://<external IP address of NGINX Ingress controller loadbalancer service>/
records
```

The URL redirects you to the Documentum Records Client login page.

5.3.1.3 Limitations

- External storage provisioners limitation: For deploying Documentum products, you need the capability to dynamically provision both ReadWriteOnce (RWO) Persistent Volumes (PVs) and ReadWriteMany (RWX) Persistent Volumes (PVs). Cloud providers provide the built-in provisioner to dynamically provision the ReadWriteOnce PVs.

  For example, on Google Cloud Platform, if you specify the storageclass as standard in the PVC, then Google Cloud Platform automatically creates a PV of requested size using Google Compute Engine Persistent Disk. However, the Google Compute Engine Persistent Disk does not support ReadWriteMany operation, and therefore, you have to provision a Google Cloud Filestore instance and external provisioner to dynamically manage the PVs for ReadWriteMany PVC.

- To achieve ingress on Google Cloud Platform or GKE Kubernetes cluster, Google Cloud Platform Google Cloud Load Balancer (GCLB) L7 is not used as this load balancer does not communicate to the services of the ClusterIP type. Use the NGINX Ingress controller to achieve Ingress in a GKE cluster. *Google Documentation* contains detailed information.

5.3.1.4 Troubleshooting

There are no troubleshooting information for this release.

5.4 Deploying and configuring Documentum REST Services on GCP

5.4.1 Kubernetes platform

5.4.1.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 211 in “Deploying and configuring Documentum Server on GCP” on page 185.

5.4.1.2 Deploying Documentum REST Services

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.

2. Perform the tasks mentioned from Step 2 to Step 5 in “Deploying Documentum Server” on page 49 in “Deploying and configuring Documentum Server on private cloud” on page 47.

3. Perform the task mentioned in Step 5 in “Deploying Documentum REST Services” on page 135 in “Deploying and configuring Documentum REST Services on private cloud” on page 134.

4. Update the ingress resource rule for Documentum REST Services in the `dctm-ingress` category.

   For example:

   ```
   restService:
       enable: true
       serviceName: <sname>-rest
       servicePort: 8080
   ```

5. Deploy the Documentum Server and Documentum REST Services Helm charts using the following command format:

   ```
   helm install <release_name> <location where Helm charts are extracted>/dctm-server -
   f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
   platform>.yaml --namespace <name of namespace>
   ```

   For example:

   ```
   helm install dctm-server /opt/temp/Helm-charts/dctm-server -f /opt/temp/Helm-charts/
   dctm-server/platforms/gcp.yaml --namespace docu
   ```

6. Verify the status of the deployment of Documentum Server and Documentum REST Services Helm charts using the following command format:

```
helm status <release_name>
```

7. Verify the status of the deployment of the Documentum Server and Documentum REST Services pods using the following command format:

```
kubectl describe pods <name of the pod>
```

8. Obtain the external IP address of the load balancer service of the NGINX Ingress controller using the following command format:

```
kubectl get svc | grep <name of the ingress controller>
```

9. Access the Documentum REST Services application using the external IP address of the NGINX Ingress controller load balancer service in a browser using the following URL format:

```
http://<external IP address of NGINX Ingress controller load balancer service>/
<Documentum ingress resource>
```

5.4.1.3 Limitations

There are no limitations for this release.

5.4.1.4 Troubleshooting

There are no troubleshooting information for this release.

5.5 Deploying and configuring Documentum Workflow Designer on GCP

5.5.1 Kubernetes platform

5.5.1.1 Prerequisites

1. Perform all the steps from Step 1 to Step 16 in "Deploying and configuring Documentum Server on GCP" on page 185.

2. Deploy the NGINX Ingress Controller Helm using the following command format:

```
helm install <name of namespace>-nginx-ingress
<location where Helm charts are extracted>
--set rbac.create=true
--namespace <name of namespace>
```

For example:

```
helm install demo-nginx-ingress stable/nginx-ingress
--set rbac.create=true
--namespace demo

NAME: demo-nginx-ingress
LAST DEPLOYED: Sun Jun 9 15:48:55 2019
NAMESPACE: demo
```

```
STATUS: DEPLOYED

RESOURCES:
==> v1beta1/ClusterRoleBinding
NAME                AGE
demo-nginx-ingress  0s

==> v1beta1/Role
demo-nginx-ingress  0s

==> v1/Service
demo-nginx-ingress-controller       0s
demo-nginx-ingress-default-backend  0s

==> v1/ConfigMap
demo-nginx-ingress-controller  0s

==> v1/ServiceAccount
demo-nginx-ingress  0s

==> v1beta1/ClusterRole
demo-nginx-ingress  0s

==> v1beta1/RoleBinding
demo-nginx-ingress  0s

==> v1beta1/Deployment
demo-nginx-ingress-controller       0s
demo-nginx-ingress-default-backend  0s

==> v1/Pod(related)
...
NAME                                                READY...
demo-nginx-ingress-controller-78cd47cf46-cw9q2      0/1...
demo-nginx-ingress-default-backend-5d47879fb7-5lptf  0/1...
```

The nginx-ingress controller is installed. It may take a few minutes for the load balancer IP to be available.

3. Verify the status of the NGINX Ingress Controller Helm deployment using the following command format:

```
kubectl --namespace <name of namespace> get services -o wide -w demo-nginx-ingress-
controller
```

Example ingress that takes control of the controller:

```
apiVersion: extensions/vibeta1
kind: Ingress
metadata:
annotations:
kubernetes.io/ingress.class: nginx
name: example
namespace: test
spec:
rules:
- host: www.example.com
http:
paths:
- backend:
serviceName: exampleService
serviceport: 80
path: /

The following code is required only if TLS is enabled for the Ingress:
tls:
- hosts:
- www.example.com
secretName: example-tls
```

4. Perform the steps from Step 2 to Step 5 in "Deploying and configuring Documentum Workflow Designer on private cloud" on page 151.

5.5.2 Deploying Documentum Workflow Designer

1. Perform all the steps as described in "Deploying Documentum Workflow Designer" on page 152 in "Deploying and configuring Documentum Workflow Designer on private cloud" on page 151.

2. To access the Documentum Workflow Designer outside of a cluster:

   a. Obtain the external IP address of the load balancer service of the NGINX Ingress controller using the following command format:

   ```
   kubectl get svc | grep <name of the ingress controller>
   ```

   b. Access the Documentum Workflow Designer using the external IP address of the NGINX Ingress controller load balancer service in a browser using the following URL format:

   ```
   http://<external IP address of NGINX Ingress controller load balancer service>/
   <Documentum workflow designer>
   ```

5.5.3 Importing batch processes using WFD cloud utility

Starting with WorkFlow Designer 22.2, you can import workflow packages into Workflow Designer using the WorkFlow Designer cloud utility. The cloud utility scans the specified shared directory path for packages and uploads the available packages to the WorkFlow Designer. You must copy the extracted packages into the shared directory and run the utility to upload the packages. A new docker image is created and corresponding Helm chart are created to configure workflow designer environment details and the shared directory path from shared PV.

Notes

- To avoid any process failure, you must import one process at a time.
- If you are importing multiple packages and one of the process fails during installation, rest of the processes in the package are ignored for installation and the next package is installed.

5.5.3.1 Prerequisites

- Processes from previous versions should be validated using Workflow Designer before exporting with Workflow Designer cloud utility.
- The shared PV and corresponding shared directory path for importing process packages should be valid.
- The user should be able to login using inline or basic authentication into Workflow Designer.

5.5.3.2 Deploying WorkFlow Designer Cloud utility

1. Open the `dctm-workflow-designer-cli/values.yaml` file and provide the appropriate values for the variables to pass them to your templates as described in the following table:

Category	Name	Description
prefix	*prefix*	Prefix for name or ID of the utility pod. Use a suitable prefix to distinguish WFD client utility pod in the cluster environment.
image	*repository*	Path of the repository.
	name	Name of the image.
	tag	Tag as a version-specific number.
	pullPolicy	Policy type to fetch the image. By default, the value is `IfNotPresent`, which skips fetching a Docker image if the same image already exists. Otherwise, use `Always` to force to fetch the Docker image.
	pullSecrets	Secret for fetching images.
workflow Designer	*url*	Workflow Designer URL where packages needs to be imported.
	username	Workflow Designer username to connect to the repository.
	password	Workflow Designer password to connect to the repository.
	repositoryName	Name of Workflow Designer repository.
trustStore	*enable*	If the certificate-based communication with Workflow Designer is enabled. Specify the value either as `true` to enable the TrustStore or as `false` to disable the TrustStore for Workflow Designer.
	fileName	Truststore filename. Truststore file should be saved in secrets folder available in Helm chart.
	password	Truststore password for the Workflow Designer.
workflow Designer CliParam	*force*	If the existing processes override is enabled. Specify `true` to enable overriding the existing processes or as `false` to disable overriding existing processes.
persistent Volume	*storageClassReadWriteOnce*	Storage class for a PV, with `ReadWriteOnce` access mode, which can be accessed by a single node.
	size	Size of the PV.

Category	Name	Description
processPackage PersistentVolume	*sharedPVCName*	PV name shared by customer for importing the process package files.
	sharedPVCFolder	Shared subpath in the PV for importing the process package files.

2. Deploy the Documentum Workflow Designer Cloud utility Helm charts using the following command format:

```
helm install <release_name> <location where Helm charts are extracted>/ --namespace
<name of namespace>
```

   For example:

```
helm install wfd-cli ./dctm-workflow-designer-cli  --namespace docu
```

   Note: After completion, the pod status changes to **Complete** and the imported processes are installed into WorkFlow Designer.

3. Verify the status of the deployment of Documentum Workflow Designer Cloud utility Helm using the following command format:

```
helm status <release_name>
```

4. Verify the status of the deployment of the Documentum Workflow Designer Cloud utility pod using the following command format:

```
kubectl describe pods <name of the pod>
```

5.5.4 Limitations

- External storage provisioners limitation: For deploying Documentum products, you need the capability to dynamically provision both ReadWriteOnce (RWO) Persistent Volumes (PVs) and ReadWriteMany (RWX) Persistent Volumes (PVs). Cloud providers provide the built-in provisioner to dynamically provision the ReadWriteOnce Persistent Volumes.

  For example, in GCP, if you specify the storageclass as standard in the PVC, then GCP automatically creates a PV of requested size using Google Compute Engine Persistent Disk. However, the Google Compute Engine Persistent Disk does not support ReadWriteMany operation, and hence you have to provision a Google Cloud Filestore instance and external provisioner to dynamically manage the Persistent Volumes for ReadWriteMany PVCs.

- In order to achieve ingress in GCP or GKE Kubernetes cluster, GCP Google Cloud Load Balancer (GCLB) L7 Load balancer is not used as this GCP GCBL L7 load balancer does not communicate to services of the ClusterIP type on the backend. Use NGINX Ingress controller to achieve Ingress in a GKE cluster. *Google Documentation* contains detailed information about Ingress with NGINX controller on GKE.

5.5.5 Troubleshooting

There are no troubleshooting information for this release.

5.6 Deploying and configuring Content Connect on GCP

5.6.1 Prerequisites

1. Perform all the steps as described in the *Deploying and configuring Documentum Server on GCP* > "Prerequisites" on page 185.
2. Download the Docker image (Oracle Linux only) from OpenText Container Registry. Perform the following tasks:

    a. Log in to OpenText Container Registry using the following command format:

    ```
    docker login registry.opentext.com
    ```

    When prompted, provide your OpenText My Support login credentials.

    b. Download the Docker image using the following command format:

    ```
    docker pull registry.opentext.com/<image_name>:<image_tag>
    ```

    The following Docker images are available for download:

Image name	Image tag
dctm-content-connect	23.2.0 or 23.2
dctm-content-connect-dbinit	23.2.0 or 23.2

3. Download the Helm charts:

    - For Documentum Server, the Helm charts are available in the `documentum_server_<releaseversion>_ kubernetes_helm_<operating-system>.tar` file from OpenText My Support.

5.6.2 Deploying Content Connect

To deploy Content Connect on GCP, perform the following steps:

1. The information for deploying and configuring Content Connect on GCP is same as described in "Deploying and configuring Documentum Server on GCP" on page 185.

    Make sure that you deploy the Content Connect Docker image (Oracle Linux only).

2. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and in the contentconnect category, provide the appropriate values for the variables to pass them to your templates.

The "contentconnect" on page 145 table contains detailed information about the variables.

5.6.2.1 Deploying Content Connect on the client machine

To deploy Content Connect to the client machine, perform the following steps:

1. After the Admin Console URL is up, add the endpoint details and download the manifest file.
2. Use the manifest file to add the Content Connect add-in to the Word or Outlook application. For more information, see *OpenText Content Connect Installation and Administration Guide*.

5.6.2.2 Deploying only Content Connect using single Helm

5.6.2.2.1 Documentum Server single Helm

To deploy only Content Connect for Documentum Server in single Helm, perform the following steps in the dctm-server/values.yaml file:

1. Except for `cs-secrets` and `contentconnect` categories, disable all the product deployment categories.
2. Update the `database: userName` and `password`.
3. In the `contentconnect` category, provide all the required values.

   The "contentconnect" on page 145 table contains detailed information about the variables.
4. Deploy Content Connect using the same command as Documentum Server deployment.

5.6.2.2.2 Limitations

There are no limitations for this release.

5.6.2.2.3 Troubleshooting

There are no troubleshooting information for this release.

5.6.2.2.4 Upgrading

Upgrade the Content Connect pod using the `helm upgrade` command:

```
Helm upgrade <release name> . --values <resource file name> -n <namespace>
```

5.7 Creating PostgreSQL database instance on GCP

1. In the GCP console, navigate to the **SQL** tab.
2. Click **CREATE INSTANCE** to create a PostgreSQL database instance.
3. Select **Postgres** for the database instance.
4. In the next tab, provide the following information:

   - Instance name (for example, `mypgdb`)
   - Password
   - Region (for example, `asia-south1`)
   - Zone (for example, `asia-south1–a`)
   - PostgreSQL version

   The database instance is created in approximately 10 minutes.

5. Click on the instance you created to view the details of the database. Note down the **Public IP address** (for example, `34.93.115.240`) and **Instance Connection name** (for exmaple, `documentum-da-product:asia-south1:mypgdb`).
6. Navigate to the **cluster** tab and note down the **Endpoint** (for example, `35.200.232.48`) details. In addition, run the following command inside the container and note down the external IP addresses of all nodes:

   ```
   kubectl get nodes -o wide
   ```

7. Navigate to the database instance, click the **Connections** tab, and then click **Add network** in **Public IP**. Add the cluster endpoint details and external IP addresses of all nodes obtained from Step 6 and save.
8. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and perform the following tasks:

   a. Provide all the details obtained from Step 5 in the *`database`* category.
   b. Provide the details for *`userName`* (the value must be `postgres` only if you are using PostgreSQL database irrespective of instance or host name) and *`password`* in *`database`* in the `content-server` category.

9. Deploy the Documentum Server Helm using the following command format:

   ```
   helm install <release_name> <location where Helm charts are extracted>/dctm-server -
   f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
   platform>.yaml --namespace <name of namespace>
   ```

   For example:

```
helm install content-server /opt/temp/Helm-charts/dctm-server -f <location where
Helm charts are extracted>/dctm-server/platforms/gcp.yaml --namespace docu
```

10. Verify the status of the deployment of Documentum Server Helm using the following command format:

```
helm status <release_name>
```

11. Verify the status of the deployment of the Documentum Server pod using the following command format:

```
kubectl describe pods <name of the pod>
```

5.8 Configuring Elastifile file system on GCP

Elastifile Cloud File System (ECFS) is a file system that can be scaled to a large number of users. ECFS is POSIX-compliant NFS file services.

ECFS supports all the enterprise-class storage features. In addition, it supports the following features:

- De-duplication
- Compression
- Snapshots
- Cross-region replication
- Quotas

Using ECFS, you can leverage GCP to shift, burst, accelerate, and/or scale file system applications and workflows without refactoring the application.

ECFS is used in two modes:

- Fully managed ECFS instances
- Self managed ECFS instance and provisioner

To integrate, use the dynamic storage provisioner to define *`StorageClass`* and *`PVC`*.

Example of the single All-In-One `dctm-server/values.yaml` Helm chart file for Documentum Platform and Platform Extensions applications:

```
storageClass: ecfs-rwx
awsEFS: false
awsEFSCSIDriver: efs.csi.aws.com
awsEFSCSIHandle: fs-82630bfa
csdataPVCName: dcs-data-pvc
createPVC: true
existVolumePv:

volumeClaimTemplate:
vctName: documentum-vct
storageClass: ecfs-rwx
logvctStorageClass: ecfs-rwx
```

5.8.1 Configuring ECFS in fully managed mode

1. Make sure that you have the ECFS access permission to enable the services. *Google Documentation* contains detailed information.
2. Search and navigate to the **Elastfile File Service** page and click **CREATE** to create an instance. Provide all the mandatory details and click **CREATE**.
3. After the instance is created, record the IP address of the NFS server and the root path that must be configured as NFS service in **Share Name**. Use this to configure PV or PVC. Example NFS server configuration:

   ```
   nfs:
   path: /ecfs-poc-flymngdsvc/root/test-dbr-vct-sjtestdbr-1-pvc-
   db0a5969-1fcd-11ea-8805-42010a8001a0
   server: 192.168.0.1
   ```

4. Optional Download the nfs-client-provisioner Helm chart from the Github website and install the chart using the following command format:

   ```
   helm install stable/nfs-client-provisioner
   --set nfs.server=<x.x.x.x
   --set nfs.path=</exported/path>
   --set nfs.name=<customStorageClassName>
   ```

   For example:

   ```
   helm install stable/nfs-client-provisioner
   --set nfs.server=<192.168.0.1
   --set nfs.path=/ecfs-poc-flymngdsvc/root/
   --set nfs.name=ecfs-fully-managed-rwx
   ```

5.8.2 Configuring ECFS in self managed mode

1. Perform all the steps to install ECFS using Google Marketplace. In addition, configure and deploy ECFS. *Google Documentation* contains detailed information.
2. To avoid the networking issues, make sure that the value provided for **ZONE** and **Network Name** is same as mentioned in the GKE cluster where your application must be deployed.
3. Retain all the other default values as described in *Google Documentation* except for the values related to storage.
4. Set up Elastifile storage provisioner. *Google Documentation* contains detailed information about:

   - Deploying Elastifile Kubernetes provisioner
   - Deploying Kubernetes provisioner in GCP for use with an ECFS

Notes

- If your Kubernetes provisioner requires network access to your ECFS cluster, it is recommended that Elastifile share the same VPC and subnet.

- For Management console URL, user name and password, use the values from Step 1.
- The default value of *`storageClass`* is `elastifile` and cannot be changed.
- Create a PVC with annotation as `volume.beta.kubernetes.io/storage-class: "elastifile"` to validate the configuration. If required, modify the size of *`storage`* and/or the mode of *`accessModes`*.

Chapter 6

Documentum Platform and Platform Extensions applications on Amazon Web Services cloud platform

The product *Release Notes* document contains detailed information about the list of applications and its supported versions for Amazon Web Services (AWS) cloud platform.

6.1 Deploying and configuring Documentum Server on AWS cloud platform

6.1.1 Kubernetes platform

6.1.1.1 Prerequisites

1. Download and configure the Docker application from the Docker website.

   Docker Documentation contains detailed information.

2. Download the supported version of Helm package from the Helm website.

   The product *Release Notes* document contains detailed information about the supported versions.

   Helm Documentation contains detailed information.

3. Download and configure the Kubernetes application from the Kubernetes website.

   Kubernetes Documentation contains detailed information.

4. Download and configure the PostgreSQL database (server) from the PostgreSQL website.

   Note: The PostgreSQL database client is packaged with Documentum Server Docker image.

   PostgreSQL Documentation contains detailed information.

5. You must be an Identity and Access Management (IAM) user with the following roles and permissions to create resources on AWS.

   - Roles: EKS to allow EKS to manage clusters and Elastic Compute Cloud (EC2) to allow instances to call AWS services.
   - Permissions: *elasticfilesystem:CreateFileSystem* and *elasticfilesystem:CreateMountTarget*.

6. Create an EKS cluster and a namespace within the cluster.

    a. Install the AWS CLI in your windows machine. *AWS Documentation* contains detailed information.

    b. Configure your AWS CLI credentials using the following command format:

    ```
    C:\ aws configure
    AWS Access Key ID [None]: <your AWS key ID>
    AWS Secret Access Key [None]: <your AWS secret key>
    Default region name [None]: <your AWS region>
    Default output format [None]:
    ```

    c. Install `eksctl` and `aws-iam-authenticator` using the following command format:

    ```
    C:\ chocolatey install -y eksctl aws-iam-authenticator
    ```

    If it is already installed, use the following command format to upgrade:

    ```
    C:\ chocolatey upgrade -y eksctl aws-iam-authenticator
    ```

    d. Verify the `eksctl` installation using the following command format:

    ```
    eksctl version
    ```

    e. Install the `kubectl` binary in your Windows CLI machine and add the binary to your `PATH` variable.

    AWS Documentation contains detailed information.

    f. Verify the `kubectl` installation using the following command format:

    ```
    kubectl version --short --client
    ```

    Note: You must use a kubectl version that is within one minor version difference of your Amazon EKS cluster control plane.

    For example, a 1.20 kubectl client must work with Kubernetes 1.19, 1.20, and 1.21 clusters.

    OpenText Documentum Server Release Notes contains the supported versions of Kubernetes cluster.

    g. Create an EKS Cluster mentioning the required cluster name, number of nodes, and your region using the following command format:

    ```
    eksctl create cluster --name myekscluster --nodes 4 --region us-east-2
    ```

    This command may take few minutes. Successful execution of this command creates an EKS cluster, nodes, default policies, and sets your `kubectl` configuration.

    h. Create a Kubernetes cluster namespace on the Cloud shell using the following command format:

    ```
    kubectl create namespace <name of namespace>
    ```

    i. Verify if the namespace is created using the following command format:

    ```
    kubectl get namespace
    ```

7. Download the Documentum Server and the required Documentum application Docker images (Oracle Linux only) from OpenText Container Registry. Perform the following tasks:

a. Log in to OpenText Container Registry using the following command format:

```
docker login registry.opentext.com
```

When prompted, provide your OpenText My Support login credentials.

b. Download the Docker image using the following command format:

```
docker pull registry.opentext.com/<image_name>:<image_tag>
```

The following Docker images are available for download:

Image name	Image tag
dctm-server	23.2.0 or 23.2
dctm-admin	23.2.0 or 23.2
dctm-dfs	23.2.0 or 23.2
dctm-records	23.2.0 or 23.2
dctm-records-darinstallation	23.2.0 or 23.2
dctm-rqm	23.2.0 or 23.2
dctm-rest	23.2.0 or 23.2

8. Upload the Docker image(s) to Amazon Elastic Container Registry (ECR). Perform the following tasks:

   a. Authenticate ECR from CLI using the following command format:

   ```
   aws ecr get-login-password --region <region> | docker login --username AWS --
   password-stdin <AWS account ID>.dkr.ecr.<region>.amazonaws.com
   ```

   For example:

   ```
   aws ecr get-login-password --region us-west-2 | docker login --username AWS --
   password-stdin 777777777.dkr.ecr.us-west-2.amazonaws.com
   ```

   b. Create a repository in ECR to load your Docker images.

      i. Open the Amazon ECR console.
      ii. From the navigation bar, choose the region where you want to create your repository.
      iii. Click **Repositories**.
      iv. In the Repositories page, click **Create repository**.
      v. Enter an unique name for your repository to configure your repository.
      vi. For the image tag mutability, choose the tag mutability setting for the repository. Repositories configured with immutable tags prevents image tags from being overwritten.

         AWS Documentation contains detailed information.
      vii. For scanning the image, choose the image scanning setting for the repository. Repositories configured to scan on load, starts an image scan whenever an image is loaded. Otherwise, the image scans must be started manually.

AWS Documentation contains detailed information.

viii. Click **Create repository**.

c. Tag your Docker image(s) using the following command format:

```
[aws_account_id.dkr.ecr.region.amazonaws.com]/[name of repository]/[image tag]
```

d. Upload the tagged Docker image(s) to ECR. For example:

```
docker push 777777777.dkr.ecr.region.amazonaws.com/dctmrepository:<image_tag>
```

9. Extract the Helm package you downloaded in Step 2 to a temporary location.

10. Install and verify the version of the Helm package.

    a. Install the Helm package in your Windows cloud shell machine using the following command format:

    ```
    choco install kubernetes-helm
    ```

    b. Verify the version of the Helm package using the following command format:

    ```
    helm version
    ```

11. Create an Amazon Elastic File System (EFS) file system.

    a. Obtain the cluster information: Before you create all of the required resources to use Amazon EFS with your Amazon ECS cluster, obtain the basic information about the cluster, such as the VPC hosted inside, and the security group that it uses.

       To obtain the VPC and security group IDs for a cluster: Open the Amazon EC2 console, select one of the container instances from your cluster and view the **Description** tab of the instance. Record the VPC ID value for your container instance. Then, create a security group and an Amazon EFS file system in this VPC. Open the security group to view its details and then record the Group ID. Then, allow the inbound traffic from this security group to your Amazon EFS file system.

    b. Create a security group for an Amazon EFS file system: Create a security group for an Amazon EFS file system to allow inbound access from your container instances.

       i. Open the Amazon EC2 console.
       ii. Click **Security Groups > Create Security Group**.
       iii. Enter an unique name for your security group.

            For example, `EFS-access-for-sg-dc025fa2`.
       iv. Enter a description for your security group.
       v. Choose a VPC that you identified earlier for your cluster.
       vi. Click **Inbound > Add rule**.
       vii. Choose **NFS** for type.
       viii. Choose **Custom** for source.
       ix. Enter the security group ID that you identified earlier for your cluster.

x. Click **Create**.

c. Create an Amazon EFS file system for Amazon ECS container instances.

i. Open the Amazon EFS console.

ii. Click **Create file system**.

iii. In the Configure file system access page, choose the VPC where your container instances are hosted. By default, each subnet in the specified VPC receives a mount target that uses the default security group for that VPC.

Note: Your Amazon EFS file system and your container instances must be in the same VPC.

iv. In **Create mount targets**, for **Security groups**, add the security group that you created. Click **Next step**.

v. Choose a throughput mode for your file system.

Note: By default, **Bursting** is the throughput mode and recommended for most file systems.

vi. Choose a performance mode for your file system.

Note: By default, **General Purpose** is the performance mode and recommended for most file systems.

vii. Review your file system options and click **Create File System**.

12. Deploy the Amazon EFS Container Storage Interface (CSI) driver.

AWS Documentation contains detailed information.

Notes

- Amazon EFS CSI driver dynamic provisioning is supported from the 23.2 release.
- If you are deploying Documentum Server 23.2 (new deployment), OpenText recommends you to use dynamic provisioning.
- If you are upgrading from a previous deployment of Documentum Server versions (for example, 22.2 or 22.4) that used static provisioning, then make sure that you use the same static provisioning storage class with the same Helm inputs (for example, *`existVolumePv`*, *`awsEFS`*, *`awsEFSCSIDriver`*, and *`awsEFSCSIHandle`*) used in the previous deployment (for example, 22.2 or 22.4) while upgrading to 23.2.

13. Create a storage class (for example, `efs-sc`) using CSI driver and test the sample PV and PVC bindings for both the Read Write Once and Read Write Many access modes.

AWS Documentation contains detailed information.

Notes

- When you are installing the EFS CSI driver with dynamic provisioning using the *AWS Documentation*, make sure to change the following variables in addition to *`fileSystemId`* in the `storageclass.yaml` file before creating the storage class as follows:

  From:

  ```
  gidRangeStart: "1000" # optional
  gidRangeEnd: "2000" # optional
  ```

  To:

  ```
  uid: "1000" # optional
  gid: "1000" # optional
  ```

- If you are using the Postgres container for database which uses the Postgres image internally, then make sure to create another storage class (for example, `efs-dbpg`) with *`uid`* and *`gid`* values as follows:

  ```
  uid: "999" # optional
  gid: "999" # optional
  ```

  Use this (`efs-dbpg`) storage class in *`db.persistentVolume.storageClass`* in the single All-In-One `dctm-server/values.yaml` Helm chart file.

  Skip this note about creating additional storage class if you are using Amazon Aurora or PostgreSQL database service.

14. Delete both the sample PV and PVC resources. Make sure that you do not delete the storage class (for example, `efs-sc`).

15. Optional Amazon Aurora or PostgreSQL database service are supported with Documentum Server. To create the database instance, see *AWS Documentation*. Make sure that you set the required security settings (for example, Virtual Private Cloud security groups) in the database instance to access it from the EKS cluster deployments.

    a. When the database instance is created successfully and is up and running, perform the following tasks:

       i. Navigate to the AWS RDS console.

       ii. Select the database instance you created.

       iii. Record the name provided for **Database instance identifier** and the **Endpoint** information.

    b. To use Amazon Aurora or PostgreSQL database instance with Documentum Server Helm deployment, open the single All-In-One `dctm-server/values.yaml` Helm chart file and perform the following tasks:

       i. Provide the appropriate value for the *`database_host`* and *`database_port`* variables in the `common-variables` category as follows:

```
databaseHost: &database_host <endpoint information>
databasePort: &database_port 5432
```

ii. Provide the appropriate value for the *`database.databaseServiceName`* variable in the `content-server` category as follows:

```
databaseServiceName: <name provided for database instance identifier>
```

iii. Provide the appropriate values for the *`database.username`* and *`database.password`* variables in the `cs-secrets` category as follows:

```
username: <database user name>
password: <database password>
```

AWS Documentation contains detailed information to set up Amazon RDS, create Amazon Aurora or PostgreSQL database instance, and to connect to the database.

16. Deploy the AWS Application Load Balancer (ALB) ingress controller add-on Helm to your EKS cluster.

    AWS Documentation contains detailed information.

17. Verify the status of the ALB ingress controller Helm deployment using the following command format:

```
kubectl get deployment -n kube-system aws-load-balancer-controller
```

18. Download the Helm charts available in the `documentum_server_<release-version>_kubernetes_helm_<operating-system>.tar` file from OpenText My Support.

    Note: The TAR file also contains Docker Compose scripts for reference along with Helm charts for the Oracle Linux operating system.

6.1.1.2 Deploying Documentum Server

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.

2. Update the image and tag fields in the single All-In-One `dctm-server/values.yaml` Helm chart file to point to the ECR images.

3. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and provide the appropriate values for the variables to pass them to your templates.

   "Deploying and configuring Documentum Server on private cloud" on page 47 contains detailed information about the variables.

   Note: When you change the default value of any of the variables in the `cs-logging-configMap` category in the single All-In-One `dctm-server/values.yaml` Helm chart file each time after the initial deployment, you must run the Helm upgrade command.

4. Set the value of *`rwo_storage_class`* and *`rwm_storage_class`* to the storage class name created in Step 13 using dynamic provisioning (for example, `efs-sc`) in “common-variables” on page 50.

   For example:

   ```
   rwoStorage: &rwo_storage_class efs-sc
   rwmstorage: &rwm_storage_class efs-sc
   ```

5. Optional To enable Documentum Server/connection broker external access, update the values for the following variables:

   a. Set the value of *`externalAccessEnaled`* to `true` in the `common-variables` category.

   b. Retain all the default values for *`ExtDocbroker`* in the `docbroker` category and make sure to change the value of *`ExtDocbroker.useELB`* to `true`.

   c. Set the value of *`ExtCS.nativeExtPort`* to `80`, *`sslExtPort`* to `81`, and *`ExtCS.useELB`* to `true` in the `content-server` category. Retain the default values for all other variables.

6. To enable ingress resources, make sure that you set the value of *`enabled`* to `true` in the `dctm-ingress` category in the single All-In-One `dctm-server/values.yaml` Helm chart file.

7. Set the value of *`configureHost`* in the `dctm-ingress` category in the single All-In-One `dctm-server/values.yaml` Helm chart file to `false`.

   Note: When you set the value of *`configureHost`* to `true`, you must provide the appropriate values for *`host`* and *`clusterDomainName`*.

8. Enable and update the service names for the required ingress resource in the `dctm-ingress` category in the single All-In-One `dctm-server/values.yaml` Helm chart file.

   Sample with example values:

   ```
   jmsService:
     enable: true
     serviceName: dctmdcs-pg-jms-service
     servicePort: 9080

   jmsBase:
       enable: false
       serviceName: <jms-service-name>
       servicePort: 9080

   acsService:
     enable: true
     serviceName: dctmdcs-pg-jms-service
     servicePort: 9080

   tnsService:
     enable: true
     serviceName: dctmdcs-pg-tns-service
     servicePort: 8081
   ```

9. Open the `aws.yaml` file in the extracted `platforms` folder and perform the following task:

In the *alb.ingress.kubernetes.io/subnets* annotation, provide the clusters' public subnet IDs separated by comma.

For example:

```
alb.ingress.kubernetes.io/subnets: subnet-0c4a1017a4ffb7962,
subnet-0bbd3ec1319a30772, subnet-0a6ab032a5e03be49
```

(Optional) If you want multiple Documentum Server deployments with different ingress addresses, you must modify the default value of the *alb.ingress.kubernetes.io/group.name* annotation to any user-defined group name for each deployment.

10. Optional If you want to use ingress in secure connection mode, perform the following tasks:

    a. Generate certificate and key from a Certificate Authority and convert them to the `PEM` format.

       For example, you can use OpenSSL. The *OpenSSL documentation* contains detailed information.

    b. To generate the AWS certificate Amazon Resource Name (ARN), upload the PEM files to AWS.

       Example command:

       ```
       C:\Users\>aws iam upload-server-certificate --server-certificate-name <name of
       certificate> --certificate-body file://<certificate in PEM format> --private-
       key file://<key in PEM format>
       ```

       When you run the preceding example command, record the ARN value in the following output:

       ```
       {
       "ServerCertificateMetadata": {
       "Path": "/",
       "ServerCertificateName": "testcertificate",
       "ServerCertificateId": "ASCA36A5J5Z4HDTEGMMXH",
       "Arn": "arn:aws:iam::<AWS account ID>:server-certificate/testcertificate",
       "UploadDate": "2021-11-30T10:56:25+00:00",
       "Expiration": "2022-11-30T10:54:31+00:00"
       }
       }
       ```

    c. Open the `aws.yaml` file in the extracted `platforms` folder and perform the following steps:

       i. In the *alb.ingress.kubernetes.io/subnets* annotation, provide the clusters' public subnet IDs separated by comma.

          For example:

          ```
          alb.ingress.kubernetes.io/subnets: subnet-0c4a1017a4ffb7962,
          subnet-0bbd3ec1319a30772, subnet-0a6ab032a5e03be49
          ```

       ii. Comment the following annotation which is provided for using ingress in the non-secure connection mode:

          ```
          alb.ingress.kubernetes.io/listen-ports: '[{"HTTP":80}]'
          ```

       iii. Uncomment the following annotations to use ingress in the secure connection mode:

```
alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS":443}]'
alb.ingress.kubernetes.io/certificate-arn: arn:aws:iam::<AWS account
ID>:server-certificate/albselfsigned
```

iv. Update the value of ARN obtained from Step 10.b in the uncommented annotation.

For example:

```
alb.ingress.kubernetes.io/certificate-arn: arn:aws:iam::<AWS account
ID>:server-certificate/testcertificate
```

11. Perform all the steps from Step 4 to Step 8 in "Deploying Documentum Server" on page 49 to complete the deployment and verification of the Documentum Server pod.

12. Obtain the ingress deployment address using the following command format:

    ```
    kubectl get ingress
    ```

    Example output with address in bold:

    ```
    NAME                                 CLASS        HOSTS
    ADDRESS                                                        PORTS
    AGE
    albingress-ingress   <none>   *       k8s-dctmgroup-2edba3b1a3-591770489.us-
    west-2.elb.amazonaws.com   80      154m
    ```

13. After you obtain the address from the ingress deployment, update the value for *host* of *ingress* in the content-server category in the single All-In-One dctm-server/values.yaml Helm chart file with the value of *ADDRESS* from Step 12.

14. Optional If you performed the tasks in Step 5, obtain the external IP address of the *csext<sname>* load balancer service using the kubectl get svc command, and then change the default value of *ExtCS.tcp_route* to the new external IP address.

15. Run the Helm upgrade command using the following command format:

    ```
    helm upgrade <release_name> ./dctm-server -f ./dctm-server/platforms/aws.yaml
    ```

16. To access any Documentum ingress resources in a browser, use the following URL format:

    ```
    http://<ingress deployment ADDRESS>/<Documentum ingress resource>
    ```

    For example:

    If you are trying to access the ACS in a browser, use the following URL format:

    ```
    http://k8s-dctmgroup-2edba3b1a3-591770489.us-west-2.elb.amazonaws.com/ACS/servlet/
    ACS
    ```

17. To access Documentum Server by external clients (outside the cluster), Documentum Server and connection broker must be deployed with externalization configuration as mentioned in Step 5, Step 14, and Step 15. After the successful deployment, two load balancer services are created, one each for *csext-<sname>* and *dbrext-<sname>* having their own external IP addresses.

Copy the `dfc.properties` file from `/opt/dctm/config/` to your client machine from the primary Documentum Server pod. The original `dfc.properties` file contains two sets of host and port pair. Remove one set of host and port pair and specify the following for the other set:

```
dfc.docbroker.host[0]=<External IP address of dbrext load balancer service>
dfc.docbroker.port[0]=<ExtCS.nativeExtPort>
```

For example:

```
dfc.docbroker.host[0]=a5e68491d32b14de4a479e41f5e3ba11-2004703172.us-
east-1.elb.amazonaws.com
dfc.docbroker.port[0]=80
```

6.1.1.3 Limitations

There are no limitations for this release.

6.1.1.4 Troubleshooting

There are no troubleshooting information for this release.

6.2 Deploying and configuring Documentum Administrator on AWS cloud platform

6.2.1 Kubernetes platform

6.2.1.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 211 in “Deploying and configuring Documentum Server on AWS cloud platform” on page 211 except for those steps related to the PostgreSQL database.

6.2.1.2 Deploying Documentum Administrator

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.

2. Perform the tasks mentioned from Step 2 to Step 5 in “Deploying Documentum Server” on page 49 in “Deploying and configuring Documentum Server on private cloud” on page 47.

3. Perform the task mentioned in Step 3 in “Deploying Documentum Administrator” on page 104 in “Deploying and configuring Documentum Administrator on private cloud” on page 104.

4. Update the ingress resource rule for Documentum Administrator in the `dctm-ingress` category.

   For example:

   ```
   daService:
       enable: true
   ```

```
    serviceName: <da-service-name>
    servicePort: 8080
```

5. Deploy the Documentum Server and Documentum Administrator Helm charts using the following command format:

   ```
   helm install <release_name> <location where Helm charts are extracted>/dctm-server -
   f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
   platform>.yaml --namespace <name of namespace>
   ```

   For example:

   ```
   helm install dctm-server /opt/temp/Helm-charts/dctm-server -f /opt/temp/Helm-charts/
   dctm-server/platforms/aws.yaml --namespace docu
   ```

6. Verify the status of the deployment of Documentum Server and Documentum Administrator Helm charts using the following command format:

   ```
   helm status <release_name>
   ```

7. Verify the status of the deployment of the Documentum Server and Documentum Administrator pods using the following command format:

   ```
   kubectl describe pods <name of the pod>
   ```

8. Perform the tasks mentioned in Step 12 to Step 15 in "Deploying Documentum Server" on page 217 in "Deploying and configuring Documentum Server on AWS cloud platform" on page 211.

9. Access the Documentum Administrator application using the ALB ingress controller service in a browser using the following URL format:

   ```
   http://<ingress deployment ADDRESS>/<Documentum ingress resource>
   ```

6.2.1.3 Limitations

There are no limitations for this release.

6.2.1.4 Troubleshooting

There are no troubleshooting information for this release.

6.3 Deploying and configuring Documentum Records Client on AWS cloud platform

6.3.1 Kubernetes platform

6.3.1.1 Prerequisites

- Perform all the steps as described in "Prerequisites" on page 211 in "Deploying and configuring Documentum Server on AWS cloud platform" on page 211 except for those steps related to the PostgreSQL database.

6.3.1.2 Deploying Documentum Records Client

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.
2. Perform the tasks mentioned from Step 2 to Step 5 in "Deploying Documentum Server" on page 49 in "Deploying and configuring Documentum Server on private cloud" on page 47.
3. Perform the task mentioned in Step 2 in "Deploying Documentum Records Client" on page 122 in "Deploying and configuring Documentum Records Client on private cloud" on page 119.
4. Update the ingress resource rule for Documentum Records Client in the `dctm-ingress` category.

   For example:

   ```
   recordsService:
       enable: true
       serviceName: <records-service-name>
       servicePort: 8080
   ```

5. Deploy the Documentum Server and Documentum Records Client Helm charts using the following command format:

   ```
   helm install <release_name> <location where Helm charts are extracted>/dctm-server -
   f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
   platform>.yaml --namespace <name of namespace>
   ```

   For example:

   ```
   helm install dctm-server /opt/temp/Helm-charts/dctm-server -f /opt/temp/Helm-charts/
   dctm-server/platforms/aws.yaml --namespace docu
   ```

6. Verify the status of the deployment of Documentum Server and Documentum Records Client Helm charts using the following command format:

   ```
   helm status <release_name>
   ```

7. Verify the status of the deployment of the Documentum Server and Documentum Records Client pods using the following command format:

   ```
   kubectl describe pods <name of the pod>
   ```

8. Perform the tasks mentioned in Step 12 to Step 15 in "Deploying Documentum Server" on page 217 in "Deploying and configuring Documentum Server on AWS cloud platform" on page 211.

9. Access the Documentum Records Client application using the ALB ingress controller service in a browser using the following URL format:

```
http://<ingress deployment ADDRESS>/<Documentum ingress resource>
```

6.3.1.3 Limitations

External storage provisioners limitation: For deploying Documentum products, you need the capability to dynamically provision both ReadWriteOnce (RWO) Persistent Volumes (PVs) and ReadWriteMany (RWX) Persistent Volumes (PVs). Cloud providers provide the built-in provisioner to dynamically provision the ReadWriteOnce Persistent Volumes.

For example, in AWS, if you specify the storageclass as `gp2` in the PVC, then AWS automatically creates a PV of requested size using Amazon Elastic Book Store (EBS). However, the default (`gp2`) does not support ReadWriteMany operation, and hence you have to provision a Amazon EFS file system and external provisioner to dynamically manage the Persistent Volumes for ReadWriteMany PVCs.

6.3.1.4 Troubleshooting

There are no troubleshooting information for this release.

6.4 Deploying and configuring Documentum REST Services on AWS cloud platform

6.4.1 Kubernetes platform

6.4.1.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 211 in “Deploying and configuring Documentum Server on AWS cloud platform” on page 211.

6.4.1.2 Deploying Documentum REST Services

1. Extract the Helm charts downloaded from OpenText My Support to a temporary location.

2. Perform the tasks mentioned from Step 2 to Step 5 in “Deploying Documentum Server” on page 49 in “Deploying and configuring Documentum Server on private cloud” on page 47.

3. Perform the task mentioned in Step 5 in “Deploying Documentum REST Services” on page 135 in “Deploying and configuring Documentum REST Services on private cloud” on page 134.

4. Update the ingress resource rule for Documentum REST Services in the `dctm-ingress` category.

   For example:

```
restService:
    enable: true
    serviceName: <sname>-rest
    servicePort: 8080
```

5. Deploy the Documentum Server and Documentum REST Services Helm charts using the following command format:

   ```
   helm install <release_name> <location where Helm charts are extracted>/dctm-server -
   f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
   platform>.yaml --namespace <name of namespace>
   ```

   For example:

   ```
   helm install dctm-server /opt/temp/Helm-charts/dctm-server -f /opt/temp/Helm-charts/
   dctm-server/platforms/aws.yaml --namespace docu
   ```

6. Verify the status of the deployment of Documentum Server and Documentum REST Services Helm charts using the following command format:

   ```
   helm status <release_name>
   ```

7. Verify the status of the deployment of the Documentum Server and Documentum REST Services pods using the following command format:

   ```
   kubectl describe pods <name of the pod>
   ```

8. Perform the tasks mentioned in Step 12 to Step 15 in “Deploying Documentum Server” on page 217 in “Deploying and configuring Documentum Server on AWS cloud platform” on page 211.

9. Access the Documentum REST Services application using the ALB ingress controller service in a browser using the following URL format:

   ```
   http://<ingress deployment ADDRESS>/<Documentum ingress resource>
   ```

6.4.1.3 Limitations

There are no limitations for this release.

6.4.1.4 Troubleshooting

There are no troubleshooting information for this release.

6.5 Deploying and configuring Content Connect on AWS cloud platform

6.5.1 Prerequisites

1. Perform all the steps as described in the "Prerequisites" on page 161 section of *Deploying and configuring Documentum Server on Amazon Web Services cloud platform*.
2. Download the Docker image (Oracle Linux only) from OpenText Container Registry. Perform the following tasks:

    a. Log in to OpenText Container Registry using the following command format:

    ```
    docker login registry.opentext.com
    ```

    When prompted, provide your OpenText My Support login credentials.

    b. Download the Docker image using the following command format:

    ```
    docker pull registry.opentext.com/<image_name>:<image_tag>
    ```

    The following Docker image is available for download:

Image name	Image tag
dctm-content-connect	23.2.0 or 23.2
dctm-content-connect-dbinit	23.2.0 or 23.2

3. Download the Documentum Helm chart in the `documentum_ server_<releaseversion>_ kubernetes_helm_<operating-system>.tar` file from the *OpenText My Support* > *Documentum Server* > *Software Download* page.

6.5.2 Deploying Content Connect

1. Upload the Content Connect image to Amazon Elastic Container Registry (ECR). Perform the following tasks:

    a. Tag your Docker image using the following command format:

    ```
    [aws_account_id.dkr.ecr.region.amazonaws.com]/[name of repository]/[image tag]
    ```

    For example:

    ```
    777777777.dkr.ecr.region.amazonaws.com/dctm-content-connect:23.2
    ```

    ```
    777777777.dkr.ecr.region.amazonaws.com/dctm-content-connect-dbinit:23.2
    ```

    b. Upload the tagged Docker image to Amazon ECR.

    For example:

    ```
    docker push 777777777.dkr.ecr.region.amazonaws.com/dctm-content-connect:23.2
    ```

```
docker push 777777777.dkr.ecr.region.amazonaws.com/dctm-content-connect-dbinit:
23.2
```

2. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and in the contentconnect category, provide the appropriate values for the variables to pass them to your templates.

   The "contentconnect" on page 145 table contains detailed information about the variables.

6.5.2.1 Deploying only Content Connect using single Helm

6.5.2.1.1 Documentum Server single Helm

To deploy only Content Connect for Documentum Server in single Helm, perform the following steps in the dctm-server/values.yaml file:

1. Except for `cs-secrets` and `contentconnect` categories, disable all the product deployment categories.
2. Update the `database: userName` and `password`.
3. In the `contentconnect` category, provide all the required values.

   The "contentconnect" on page 145 table contains detailed information about the variables.
4. Deploy Content Connect using the same command as Documentum Server deployment.

6.5.3 Limitations

There are no limitations for this release.

6.5.4 Troubleshooting

Symptom	Cause	Fix
When using Content Connect, Cross-Origin Resource Sharing (CORS) related errors occur even after all configurations are set.	CORS related erros occur.	• Increase the Load balancer response timeout. • Add the Content Connect Admin Console URL to the `rest.cors.allowed.origins` tag in the `rest-api-runtime.properties` file.

Chapter 7

Features for all cloud platforms

7.1 Integrating New Relic

New Relic is an application performance monitoring tool. This tool gives information that helps to analyze the application behavior, create real-time dashboards, and customization charts that are useful for application metric.

7.1.1 Integrating New Relic with Documentum Server

Documentum Server is integrated with New Relic using new library, `dmMonitorLib`. Documentum Server communicates with the New Relic C agent using the new library. For New Relic monitoring of Documentum Server, you must have a daemon agent running so that it communicates with the New Relic server. Documentum Server integrated with New Relic C-SDK provides the RPC-wide information for every session in Documentum. The supported version of C-SDK is 1.2.0. The integration helps to analyze Documentum Server application data for performance monitoring on New Relic.

The integration is valid ONLY for the Linux/PostgreSQL combination of Documentum Server deployed on a cloud platform.

In the single All-In-One `dctm-server/values.yaml` Helm chart file, provide the appropriate values for the variables in the `cs-secrets` and `docbroker` categories to pass them to your templates as described in the following table:

Category	Name	Description
newrelic	*enable*	Indicates if New Relic is enabled. The default value is `false`. Specify the value provided for *newrelic_enabled* in the `common-variables` category in Step 4.a. If you set the value to `true`, then the values for the remaining variables are used to identify the process in the dashboard.
	app_name	Application name of Java Method Server. The format is `<application name>-<sname>`.
	proxy_host	IP address of the proxy server. Specify the value provided for *newrelic_proxy_host* in the `common-variables` category in Step 4.a.

Category	Name	Description
	proxy_port	Available port reserved for the New Relic server. Specify the value provided for *newrelic_proxy_port* in the common-variables category in Step 4.a.
	proxy_protocol	Protocol to connect to the pod. Specify the value provided for *newrelic_proxy_protocol* in the common-variables category in Step 4.a.
	c_app_name	Name of the Documentum Server pod to be integrated with New Relic. The format is <application name>-<sname>.

7.1.2 Integrating New Relic with JMS

In the single All-In-One dctm-server/values.yaml Helm chart file, provide the appropriate value for the variable to pass them to your templates as described in the following table:

Category	Name	Description
newrelic	*license_key*	License key information of New Relic.
	app_name	Application name of Java Method Server. The format is <application name>-<sname>.

Note: If you want to use the New Relic Java agent for Tomcat or JMS independent of Documentum Server, you must provide the appropriate values as described in "Integrating New Relic with Documentum Server" on page 229. You must not provide any value for *c_app_name* as it belongs to Documentum Server.

7.1.3 Integrating New Relic with Documentum Administrator

In the da category in the single All-In-One dctm-server/values.yaml Helm chart file, provide the appropriate values for the variables to pass them to your templates as described in the following table:

Category	Name	Description
newrelic	*enable*	Indicates if New Relic is enabled. The default value is false. Specify the value provided for *newrelic_enabled* in the common-variables category in Step 4.a. If you set the value to true, then the New Relic APM agent is configured and started during the startup of the container.

Category	Name	Description
	appName	Application name (descriptive) of Documentum Administrator. This is mandatory if New Relic is enabled.
	proxyHost	IP address of the proxy server, if any. Specify the value provided for *newrelic_proxy_host* in the common-variables category in Step 4.a.
	proxyPort	Available port reserved for the New Relic server, if any. Specify the value provided for *newrelic_proxy_port* in the common-variables category in Step 4.a.
	proxyScheme	Indicates to allow the agent to connect through proxy using the HTTP or HTTPS protocol. Specify the value provided for *newrelic_proxy_protocol* in the common-variables category in Step 4.a.

7.1.4 Integrating New Relic with Documentum Foundation Services

In the dfs category in the single All-In-One dctm-server/values.yaml Helm chart file, provide the appropriate values for the variables to pass them to your templates as described in the following table:

Category	Name	Description
newrelic	*enable*	Indicates if New Relic is enabled. The default value is false. If you set the value to true, then the values for the remaining variables are used to identify the process in the dashboard.
	licenseKeyName	License key information of New Relic.
	dfs_application_name	Application name of Documentum Foundation Services. The format is <MacroService*>_<MicroService>-<ENV>-<Platform_DC>-<BU>-<Customer*>.
	proxy_host	IP address of the proxy server.
	proxy_port	Port reserved for the New Relic server.

7.1.5 Integrating New Relic with Documentum Records Client

In the `records` category in the single All-In-One `dctm-server/values.yaml` Helm chart file, provide the appropriate values for the variables to pass them to your templates as described in the following table:

Category	Name	Description
newrelic	*`enable`*	Indicates if New Relic is enabled. The default value is `false`. If you set the value to `true`, then the values for the remaining variables are used to identify the process in the dashboard.
	`appName`	Application name of the Records Client.
	`proxyHost`	IP address of the proxy server.
	`proxyPort`	Port reserved for the New Relic server.
	`proxyScheme`	Proxy scheme such as HTTP. This allows the agent to connect through proxies.

7.1.6 Integrating New Relic with Documentum REST Services

In the `dctm-rest` category in the single All-In-One `dctm-server/values.yaml` Helm chart file, provide the appropriate values for the variables to pass them to your templates as described in the following table:

Category	Name	Description
newrelic	*`enabled`*	Indicates if New Relic is enabled. The default value is `false`. Specify the value provided for *`newrelic_enabled`* in the `common-variables` category in Step 4.a.
	`configurationFile`	Name of the configuration file. The default value is `newrelic.yml`.
	`addNodeNamePrefix`	Indicates to add the node name as a prefix for the application name that is displayed in the New Relic dashboard.
	`proxy_host`	IP address of the proxy server. Specify the value provided for *`newrelic_proxy_host`* in the `common-variables` category in Step 4.a.
	`proxy_port`	Port reserved for the New Relic server. Specify the value provided for *`newrelic_proxy_port`* in the `common-variables` category in Step 4.a.
	`proxy_protocol`	Protocol to connect to the pod. Specify the value provided for *`newrelic_proxy_protocol`* in the `common-variables` category in Step 4.a.

Category	Name	Description
	app_name	Application name of Documentum REST Services. The format is `<application name>-<sname>`.

7.1.7 Integrating New Relic with Content Connect

The New Relic APM has been integrated with Content Connect for application monitoring. For more information about New Relic, visit the New Relic website.

Open the single All-In-One `dctm-server/values.yaml` Helm chart file and provide the appropriate value for the variables depending on your environment to pass them to your templates as described in the following table:

Category	Name	Description
newrelic	*new_relic_enabled*	Set `true|false` to enable New Relic in the application. The default value is `false`. When set to `true`, provide the values for *new_relic_app_name* and *new_relic_license_key*.
	new_relic_ app_name	A unique application name. Content Connect constructs the application name as *NODE_NAME*, *appNameSuffix* in `newrelic.yaml`.
	new_relic_ license_key	License key information of New Relic.
	new_relic_ proxy_protocol	Protocol to connect to the pod. Specify the value provided for *newrelic_proxy_protocol* in the `common-variables` category in Step 4.a.
	new_relic_ proxy_host	IP address of the proxy server. Specify the value provided for *newrelic_proxy_host* in the `common-variables` category in Step 4.a.
	new_relic_ proxy_port	Port reserved for the New Relic server. Specify the value provided for *newrelic_proxy_port* in the `common-variables` category in Step 4.a.

7.1.7.1 Troubleshooting New Relic integration

Unable to check the New Relic logs

Run the command `docker exec -it <container name> /bin/bash` to access the container. After the New Relic parameters are passed successfully, you can check the New Relic logs in the `newrelic_agent.log` file generated and available at the `/contentconnect` source folder. This log file helps in monitoring the application connectivity with the New Relic portal.

Cannot find the applications in the New Relic website after starting the container

- The New Relic SDK sends the data to the New Relic server. Verify if there are any network issues.
- Verify the proxy settings.

7.2 Deploying Graylog Collector Sidecar and Graylog Sidecar using Documentum Helm charts

Graylog Collector Sidecar (legacy) and Graylog Sidecar are lightweight configuration management systems to collect the logs.

7.2.1 Prerequisites

Deploy the Graylog server in your cloud platform and record the service name and port number in which the services are listening to.

7.2.2 Deploying Graylog Collector Sidecar (legacy)

1. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and provide the values for *`graylogServer`* and *`graylogPort`* in the `common–variables` category with the Graylog headless service name and the port information. Provide values for all other variables in the single All-In-One `dctm-server/values.yaml` Helm chart file.
2. Deploy the Documentum Server Helm using the following command format:
   ```
   helm install <release_name> <location where Helm charts are extracted>/dctm-server -
   f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
   platform>.yaml --namespace <name of namespace>
   ```
   For example:
   ```
   helm install dctm-server /opt/temp/Helm-charts/dctm-server -f <location where Helm
   charts are extracted>/dctm-server/platforms/<cloud platform>.yaml --namespace docu
   ```
3. Verify the status of the deployment of Documentum Server Helm using the following command format:
   ```
   helm status <release_name>
   ```
4. Verify the status of the deployment of the Documentum Server pod using the following command format:

```
kubectl describe pods <name of the pod>
```

5. After deploying the Documentum Server Helm chart, navigate to the Graylog server dashboard and click **System / Collectors (legacy)**.

   The list of deployments is displayed.

6. Perform the following configuration to get the application logs in the Graylog server dashboard:

   a. Click **System > Inputs** and create a Beats input (Global) as follows:

      i. **Title**: Provide a name for your new input.

         For example, `beats-input`.

      ii. **Bind address**: Provide the binding address to listen on.

         For example, `0.0.0.0`.

      iii. **Port**: Provide the port information to listen on.

         For example, `5044`.

      iv. Retain all the other default values.

      v. Click **Save**.

   b. Navigate to **System > Collectors > Manage configurations** in the Graylog web interface and click **Create Configuration**.

   c. In the **Create Configuration** page, type a configuration name (for example, `apache`) and click **Save**.

   d. Create a Beats output in the new configuration page created in Step 6.c as follows:

      i. **Name**: Provide an unique name for the output.

      ii. **Type**: Provide the output type that you want to configure.

      iii. **Hosts**: Provide the Graylog headless service name and port information.

      iv. Retain all the other default values.

      v. Click **Save**.

      For example:

      ```
      Name: beats-output
      Type: [Filebeat]Beats output
      Hosts:['graylog-headless.demo.svc.cluster.local:5044']
      ```

   e. Create Beats input in the new configuration page created in Step 6.c as follows:

      i. **Name**: Provide an unique name for the input.

      ii. **Forward to (Required)**: Provide the collector output to forward messages from the input.

      iii. **Type**: Provide the input type that you want to configure.

      iv. **Path to Logfile**: Provide the location of the log files that must be used.

v. Retain all the other default values.

vi. Click **Save**.

For example:

```
Name: dba-logs
Forward to (Required): beats-output[filebeat]
Type: [Filebeat]file input
Path to Logfile: ['/pod-data/dba/*.log']
```

f. In the **Collector apache Configuration > Configuration tags** page, tag the configuration with the `apache` and `Linux` tags as follows:

i. **Tags**: Provide the name of the tags.

For example, `apache` and `Linux`.

ii. Press Enter.

iii. Click **Update tags**.

Graylog Documentation contains detailed information.

The Graylog Collector Sidecar (legacy) set up is complete.

The Graylog Collector Sidecar (legacy) deployment in your cluster automatically detects all the configuration information and then restarts. You can view the connection broker and the Documentum Server logs in the Graylog dashboard search page.

7.2.3 Deploying Graylog Sidecar

1. Create an API token for Graylog Sidecar. Navigate to **System > Authentication** in the Graylog web interface. For Sidecar System User Built-In User, click **More Actions > Edit tokens**. Provide an unique name for the new token and click **Create Token**.

2. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and perform the following tasks:

   a. Provide the appropriate values for the variables in the `cs-secrets` category to pass them to your templates as described in Step 4.b in "Deploying Documentum Server" on page 49. Make sure that you provide the Graylog token value created in Step 1 in "Deploying Graylog Sidecar" on page 236.

   b. Provide the values for ***graylogServer*** and ***graylogPort*** in the `common-variables` category with the Graylog headless service name and the port information. Provide values for all other variables in the single All-In-One `dctm-server/values.yaml` Helm chart file.

3. Deploy the Documentum Server Helm using the following command format:

```
helm install <release_name> <location where Helm charts are extracted>/dctm-server -
f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
platform>.yaml --namespace <name of namespace>
```

For example:

```
helm install dctm-server /opt/temp/Helm-charts/dctm-server -f <location where Helm
charts are extracted>/dctm-server/platforms/<cloud platform>.yaml --namespace docu
```

4. Verify the status of the deployment of Documentum Server Helm using the following command format:

   ```
   helm status <release_name>
   ```

5. Verify the status of the deployment of the Documentum Server pod using the following command format:

   ```
   kubectl describe pods <name of the pod>
   ```

6. After deploying the Documentum Server Helm, navigate to the Graylog server dashboard and click **System / Sidecars**.

   The list of deployments is displayed.

7. Perform the following configuration to get the application logs in the Graylog server dashboard:

   a. Navigate to **System > Inputs** in the Graylog web interface, select **Beats** and click **Launch new Beats Input**.

   b. In the **Launch new Beats input** page, create the Filebeat input as follows:

      i. **Title**: Provide a name for your new input.

         For example, `demo cluster filebeat input`.

      ii. **Bind address**: Provide the binding address to listen on.

         For example, `0.0.0.0`.

      iii. **Port**: Provide the port information to listen on.

         For example, `5044`.

      iv. Retain all the other default values.

      v. Click **Save**.

   c. Start the new Filebeat input.

   d. Navigate to **System / Sidecars** in the Graylog web interface, click **Configuration** and then click **Create Configuration**.

   e. In the new configuration page created in Step 7.d, provide all the required information. Make sure that you provide the information for input paths and Graylog headless service and port information.

      For example:

      ```
      Name: dctm-dbr-config
      Collector : filebeat on Linux
      .
      Configuration:
      fields_under_root: true
      fields.collector_node_id: ${sidecar.nodeName}
      fileds.gl2_source_collector: {sidecar.nodeId}
      .
      filebeat.inputs:
      - input_type: log
      paths:
      - /pod-data/dba/*.log
      ```

```
type: log
ouput.logstash:
hosts: ["graylog-headless.demo.svc.cluster.local:5044"]
path:
data: /var/lib/graylog-sidecar/collectors/filebeat/data
logs: /var/lib/graylog-sidecar/collectors/filebeat/log
```

f. Navigate to **System / Sidecars** in the Graylog web interface and click **Administration**.

g. On the **Collectors Administration** page, select the **filebeat** option in the Graylog Sidecar to which you want to apply the Filebeat configuration created in Step 7.d.

h. Click the **Configure** list and select the Filebeat configuration created in Step 7.e.

   After the Filebeat configuration is applied, the Filebeat daemon in the Graylog Sidecar displays the applied configuration and the running status of the **filebeat** configuration.

Graylog Documentation contains detailed information.

The Graylog Sidecar setup is complete.

After the Filebeat configuration is applied, the configuration is passed on to the Graylog Sidecar. The Filebeat daemon running inside the Graylog Sidecar detects the changes in the configuration and restarts. Then, the Graylog Sidecar sends the logs mentioned in the Filebeat configuration to the Beats input configured in the Graylog server. The Filebeat configuration can be applied to multiple Graylog Sidecars (for example, to all the connection brokers running in the clusters as the paths of the logs for all the connection brokers remain the same).

7.3 Configuring containers and pods for Kubernetes liveness probe

The liveness probe feature of Kubernetes is used to check if the container or pod, where applications are running, is running or stuck. In the case of the liveness probe failure, the container or pod is ready for a restart.

7.3.1 Checking liveness of connection broker

The liveness probe of the connection broker checks for the ports based on the mode in which the connection broker is configured. The liveness probe is based on the following table:

Connection mode	External connection broker enabled	Default port(s) to probe
native	false	1489
secure	false	1490
dual	false	1489, 1490

Connection mode	External connection broker enabled	Default port(s) to probe
native	true	1489, 1491
secure	true	1490, 1492
dual	true	1489, 1490, 1491, 1492

By default, the liveness probe starts to check if the connection broker container or pod is running or stuck, 10 minutes after the connection broker container or pod is deployed. However, you can modify the value of *`initialDelaySeconds`* in *`docbroker.liveness`* in the single All-In-One `dctm-server/values.yaml` Helm chart file to set a different initial delay time. After the initial delay time that you have set, the liveness probe of the connection broker container or pod is done once in three minutes. If the liveness probe fails consecutively for five times, then the container or pod is ready for a restart.

By default, the liveness probe is enabled. If you want to disable the liveness probe of the connection broker container or pod, set the value of *`enable`* to `false` in *`docbroker.liveness`* in the single All-In-One `dctm-server/values.yaml` Helm chart file.

7.3.2 Checking liveness of Documentum Server

The liveness probe of Documentum Server checks the `50000` and `50001` ports.

By default, the liveness probe starts to check if the Documentum Server container or pod is running or stuck, 10 minutes after the Documentum Server container or pod is deployed. However, you can modify the value of *`initialDelaySeconds`* in *`contentserver.liveness`* in the single All-In-One `dctm-server/values.yaml` Helm chart file to set a different initial delay time.

After the initial delay time that you have set, the liveness probe of Documentum Server is done once in three minutes. If the liveness probe fails consecutively for five times, then the container or pod is ready for a restart.

By default, the liveness probe is enabled. If you want to disable the liveness probe of Documentum Server, set the value of *`enable`* to `false` in *`contentserver.liveness`* in the single All-In-One `dctm-server/values.yaml` Helm chart file.

If the Documentum repository is unavailable because of the unavailability of the dependent database, then the Documentum Server container or pod does not get restarted automatically.

7.3.3 Checking liveness of Java Method Server

To enable liveness probe of JMS, the liveness probe in Documentum Server container verifies the following ports and URLs:

- Documentum Server ports: `50000` and `50001`
- JMS port: `9080`
- URL of JMS: `<JMS_PROTOCOL>://localhost:9080/DmMethods/servlet/DoMethod`
- URL of ACS: `<JMS_PROTOCOL>://localhost:9080/ACS/servlet/ACS`

By default, the liveness probe is enabled. If you want to disable the liveness probe of JMS, set the value of *`considerJMSForLiveness`* to `false` in *`contentserver.liveness`* in the single All-In-One `dctm-server/values.yaml` Helm chart file.

7.3.4 Checking liveness of Documentum Administrator

The liveness probe of Documentum Administrator sends the HTTP/HTTPS requests to the Documentum Administrator pod to check if the Documentum Administrator container or pod is running or stuck. By default, the liveness probe is enabled.

In the `da` category in the single All-In-One `dctm-server/values.yaml` Helm chart file, provide the appropriate values for the variables as described in the following table:

Category	Name	Description
liveness probe	*`initialDelaySeconds`*	Number of seconds after the pod has started before the probe is initiated. The default value is `600` seconds.
	`periodSeconds`	Frequency to perform the probe. The default value is `120` seconds.

7.3.5 Checking liveness of Documentum Foundation Services

The liveness probe of Documentum Foundation Services sends the HTTP/HTTPS requests to the Documentum Foundation Services pod to check if the Documentum Foundation Services container or pod is running or stuck. By default, the liveness probe is enabled.

In the `dfs` category in the single All-In-One `dctm-server/values.yaml` Helm chart file, provide the appropriate values for the variables as described in the following table:

Category	Name	Description
liveness probe	*initialDelay Seconds*	Number of seconds after the pod has started before the probe is initiated. The default value is 180 seconds.
	period Seconds	Frequency to perform the probe. The default value is 120 seconds.

7.3.6 Checking liveness of Documentum Records Client

The liveness probe of Documentum Records Client sends the HTTP/HTTPS requests to the Documentum Records Client pod to check if the Documentum Records Client container or pod is running or stuck. By default, the liveness probe is enabled.

In the `records` category in the single All-In-One `dctm-server/values.yaml` Helm chart file, provide the appropriate values for the variables as described in the following table:

Category	Name	Description
liveness Probe	*healthPath*	Path used for checking the health of the Documentum Records Client pod.
	healthPort	Port reserved for the Documentum Records Client pod.
	initialDelay Seconds	Number of seconds after the Documentum Records Client pod has started before the probe is initiated. The default value is 180 seconds.
	period Seconds	Frequency to perform the probe. The default value is 300 seconds.
	failure Threshold	Time when a pod starts and the probe fails, Kubernetes attempts based on the failure threshold time before stopping. The default value is 2 seconds.
	success Threshold	Minimum consecutive successes for the probe to be considered successful after having failed. The default value is 1 second.
	timeout Seconds	Number of seconds after which the probe times out. The default value is 120 seconds.

7.3.7 Checking liveness of Documentum REST Services

The liveness probe of Documentum REST Services sends the HTTP/HTTPS requests to the REST pod to check if the Documentum REST Services container or pod is running or stuck. By default, the liveness probe is enabled.

In the `dctm-rest` category in the single All-In-One `dctm-server/values.yaml` Helm chart file, provide the appropriate values for the variables as described in the following table:

Category	Name	Description
liveness Probe	*`enabled`*	Indicates if the liveness probe is enabled. The default value is `true`.
	`scheme`	Scheme (HTTP/HTTPS) for the probe URL. The default value is `HTTP`.
	`initialDelay Seconds`	Time in seconds after the container or pod has started before performing the first probe. The default value is `40`.
	`periodSeconds`	Frequency to perform the probe. The default value is `5`.

The port number in which the requests are sent to, is the same as defined in *`httpPort`* or *`httpsPort`* in the `dctm-rest` category in the single All-In-One `dctm-server/values.yaml` Helm chart file depending on scheme utilized.

7.4 Integrating OTDS

7.4.1 Integrating OTDS with Documentum Server

Documentum Server is integrated with OTDS. The *OpenText Documentum Server Administration and Configuration Guide* contains detailed information.

Open the single All-In-One `dctm-server/values.yaml` Helm chart file and provide the appropriate values for the variables in the `content-server` category to pass them to your templates as described in “content-server” on page 66.

7.4.2 Integrating OTDS with Documentum Foundation Services

Documentum Foundation Services is integrated with OTDS. The *Documentum Foundation Services* chapter in *OpenText Documentum Platform and Platform Extensions Installation Guide* contains detailed information.

By default, OTDS (token- and password-based) authentication is enabled for both SSO- (`SsoIdentity`) and repository-identity (`RepositoryIdentity`) in Documentum Foundation Services. *OpenText Documentum Foundation Services Development Guide* contains detailed information.

7.4.3 Configuring OTDS roles for Documentum Records Client

If OTDS is enabled for Documentum Records Client, follow these steps to add OTDS to the required roles:

1. Log in to Documentum Administrator.
2. Navigate to users and search for the OTDS user.
3. Right-click the user, select **dmc_rps_retentionmanager**, and click the right arrow.
4. Select **dmc_rm_recordsmanager** and click the right arrow.
5. If Physical Records is used, select **dmc_prm_physical_records_manager** and click the right arrow.
6. Click **OK**.

7.4.4 Integrating OTDS with Documentum REST Services

Documentum REST Services is integrated with OTDS.

If the client wants to use OTDS, open the single All-In-One `dctm-server/values.yaml` Helm chart file and in the `dctm-rest` category, set the value of *`enable`* of *`otds`* to `true`. Provide the appropriate values for the mandatory variables such as *`url`* and *`clientID`* to pass them to your templates as described in Step 5 in “Deploying Documentum REST Services” on page 135.

7.5 Integrating Event Hub

Documentum supports the Event Hub feature using the Fluentd and Apache Kafka services. Fluentd is an open source data collection tool using the Apache 2.0 license. Apache Kafka cluster is an open source real-time streaming messaging system and protocol built around the publish-subscribe system. In the publish-subscribe system, the producers publish data to feeds for the subscribed users.

Fluentd acts as an event aggregator for the Documentum applications and producer for Kafka using `@forward` and `@Kafka2` as input and output plug-ins respectively. Kafka is used to store the events accumulated at Fluentd.

Audit trail, custom, and RPC events are logged in Kafka.

Mandatory fields for the events logged in Event Hub:

- EVENT_VERSION
- EVENT_DESCRIPTION
- EVENT_DATETIME
- EVENT_NAME
- EVENT_CATEGORY

7.5.1 Configuring Apache Kafka cluster

To configure Apache Kafka cluster, you must configure both the `Kafkabroker` and `ZooKeeper` services. *Apache Kafka Documentation* contains detailed information.

In addition, perform the following tasks:

1. Download and extract the Helm charts for Kafka from OpenText My Support.
2. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and provide the appropriate values for the Kafka variables in common variables in "common-variables" on page 50.

   Note: OpenText recommends you to use *Apache Kafka documentation* for providing values for *`kafka_replica_count`*. If you want to change the value of replica count for Kafka broker and ZooKeeper, you must manually modify the value in common variables in Step 4.a.

3. Optional If you want to add new Kafka broker and ZooKeeper variables, then add the variables prefixed with `KAFKA_` and `ZOOKEEPER_` respectively in a new `extraEnv` category in the `kafka-statefulset` (in `kafkabroker`) and `zookeeper-statefulset` (in `zookeeper`) categories.

   For example, if you want to add a configuration variable such as *`compression.type`* for Kafkabroker, then you must add as follows:

```
extraEnv:
  - name: KAFKA_compression_type
    value: "delete"
```

4. Optional If you want to externalize the Kafka broker service, set the value of *`enable`* of *`ExtAccess`* to `true` in the single All-In-One `dctm-server/values.yaml` Helm chart file.

5. Optional If you want to change the default values for the following actions, then naviage to `charts/kafka-configMap/templates/kafka-configMap.yaml` provide the appropriate values for the required variables as per your requirement:

   - To enable the debug traces in Kafka or ZooKeeper, set the value of *`logging_level`* to `DEBUG`. Valid values are `INFO` and `DEBUG`.
   - To enable dynamic topic creation in Kafka, set the value of *`auto_topic_create`* to `true`.
   - To set the default number of topics for dynamic topic, provide the value for *`num_partitions`* as per your requirement.
   - To modify the default retention time, provide the value in hours for *`log_retention_hours`* as per your requirement.

     Note: If you want to modify the value after creating the cluster, then the value of *`log_retention_hours`* must be set to addition of previous value and required time.
   - To modify the default log rolling time, provide the value in hours of *`log_roll_hours`* as per your requirement.
   - To increase the performance during a higher load, set the value of *`background_threads`* to any value greater than 10.

     Note: OpenText recommends that you do not change the value of this variable frequently.

   Note: If you change values for any of these variables after the deploying the cluster, you must run the `helm upgrade` command.

After the cluster is configured, the cluster uses SASL_PLAINTEXT as the security protocol for listeners, and uses the SCRAM mechanism using SHA-512 as crypto-algorithm during the user authentication operations.

7.5.2 Integrating Event Hub with Documentum Server

Documentum Server supports the Event Hub feature.

To use the feature, you must set the value of *fluentd_enabled* and *kafka_enabled* to `true` in common variables in Step 4.a in the single All-In-One `dctm-server/values.yaml` Helm chart file. Make sure that you update the values of other *fluentd* and *kafka* related variables in common variables in Step 4.a in the single All-In-One `dctm-server/values.yaml` Helm chart file.

For example, *fluentd_tcp_port*, *fluentd_image*, *kafka_replica_count*, *kafka_image*, and so on.

Fluentd pushes the events that are generated by Documentum Server to the Apache Kafka cluster.

By default, the `Eventhub.dar` file is deployed when you deploy the Documentum Server pod. The `Eventhub.dar` file establishes the connection with Fluentd and works as a BOF module.

The `dfc.properties` file of Documentum Server contains the following variables for Event Hub:

- *dfc.client.should_use_eventhub*: Used to enable Event Hub at DFC. You must manually set the value to `true` in all the pods.
- *dfc.client.eventhub.log_level*: Used to define the log level for the DFC events. Specify the value provided for *event_log_level* in the `common-variables` category in Step 4.a. The values may range from `0` to `5` where 0 is for NO LOG, 1 is for ERROR, 2 is for WARN, 3 is for INFO, 4 is for DEBUG, and 5 is for TRACE.

7.5.2.1 Fluentd and Kafka with Documentum Server

Fluentd is packaged with Documentum Server. When you deploy the Documentum Server pod, Fluentd is deployed as a sidecar container.

You must deploy the Apache Kafka cluster as described in “Configuring Apache Kafka cluster” on page 244.

Make sure that you provide the appropriate values for the Fluentd and Kafka variables in the single All-In-One `dctm-server/values.yaml` Helm chart file. Some of the important variables are *enable*, *TCPPort*, *RESTPort*, *UDPPort*, *compressionMode*, *bufferingMode*, *flushInterval*, *image*, *kafkaTopic*, *kafkaUser*, *kafkaUsrPasswd*, *kafkabroker*, and *readiness*.

“Deploying Documentum Server” on page 49 contains detailed information about the variables.

7.5.3 Retrieving messages

You can retrieve the messages in the following ways:

- Using the Kafka built-in utility: Log in to the Kafka broker pod, naviage to `/kafka-setup/kafka_<version>/bin`, and then run the following command format:

```
./kafka-console-consumer.sh --topic <topic name> --from-beginning --
consumer.config ../config/consumer.properties --bootstrap-server <kafka broker IP
address or host>:<kafka broker port>
```

- Using the sample program: To access events from Kafka using the sample program, see *Apache Kafka documentation*.

Note: You may encounter any additional messages that are not related to custom, DFC, or Audit events while retrieving the logged event messages. You can ignore these additional messages.

7.5.4 Integrating Event Hub with Documentum Foundation Services

Documentum Foundation Services supports the Event Hub feature.

To use the feature, you must perform the following tasks:

- Set the value of *`dfc.client.should_use_eventhub`* to `true` in `dfc.properties` of Documentum Foundation Services.
- Set the value of *`enable`* in the `fluendConf` category in the single All-In-One `dctm-server/values.yaml` Helm chart file.

Fluentd pushes the events that are generated by Documentum Foundation Services to the Apache Kafka cluster.

By default, the `Eventhub.dar` file is deployed when you deploy the Documentum Server pod. The `Eventhub.dar` file establishes the connection with Fluentd and works as a BOF module.

The `dfc.properties` of Documentum Foundation Services and the single All-In-One `dctm-server/values.yaml` Helm chart files contain the following variables for Event Hub:

- *`dfc.client.should_use_eventhub`*: Used to enable Event Hub. Set the value to `true` to enable Event Hub. The default value is `false`.
- *`dfc.client.eventhub.log_level`*: Used to define the event log levels. Set any value from `0` to `5` where 0 is for NO LOG, 1 is for ERROR, 2 is for WARN, 3 is for INFO, 4 is for DEBUG, and 5 is for TRACE. The default value is `4`.
- *`dfc.client.eventhub.queue_size`*: Used to define the number of events in the queue buffered in Documentum Foundation Classes. The default value is `1000`.

Note: Changes to the `dfs-configmap.yaml` file are reflected in the Documentum Foundation Services pod within a few seconds.

7.5.4.1 Fluentd and Kafka with Documentum Foundation Services

Fluentd is packaged with Documentum Foundation Services. When you deploy the Documentum Foundation Services pod, Fluentd is deployed as a sidecar container.

You must deploy the Apache Kafka cluster as described in "Configuring Apache Kafka cluster" on page 244.

Make sure that you provide the appropriate values for the Fluentd and Kafka variables in the single All-In-One `dctm-server/values.yaml` Helm chart file. Some of the important variables are *`enable`*, *`TCPPort`*, *`RESTPort`*, *`UDPPort`*, *`compressionMode`*, *`bufferingMode`*, *`flushInterval`*, *`image`*, *`kafkaTopic`*, *`kafkaUser`*, *`kafkaUsrPasswd`*, *`kafkabroker`*, and *`readiness`*.

"Deploying Documentum Foundation Services" on page 110 contains detailed information about the variables.

7.5.4.2 Documentum Foundation Services specific event names

Service name or Category	Event name
REPOSITORY INQUIRY	GetRepoListOperation
	ListContentReposAction
	SetContentRepoList
	GetServers
	CheckMinServerVersionCompatibility
	GetRepoNameByObjIdOperation
	RepoNameById
QUERY	ExecuteOperation
	QueryAction
	GetActionResult
	LogQuery
	SetQueryResult
QUERY STORE	ListSavedQueriesOperation
	ListSavedQueriesAction
	ListSavedSearches
	LoadSavedQueryOperation
	LoadSavedQueryAction
	LoadSavedQuery

Service name or Category	Event name
SCHEMA	GetRepoInfoOperation
	GetSchemaInfoOperation
	GetTypeInfosOperation
	GetTypeInfoOperation
	GetPropertyInfoOperation
	GetDynamicAssistValuesOperation
	GetValueAssistSnapshotOperation
OBJECT CREATE	CreateOperation
	CreateAction
	CreateObject
	Validate
	ValidateContent
	CreateContentfulObject
	CreateContentlessObject
	UpdateLightweigthObject
	Checkout
	SaveObject
OBJECT CREATE PATH	CreatePathOperation
OBEJCT SERVICE GET	GetOperation
	DeepGetAction
	ContentLoaderObjectCreate
	PropertiesLoaderObejctCreate
	PermissionLoaderObjectCreate
	Walk
	AddCreatedObject
	GatherRelatedDataObjects
	LoadRestOfRelations
OBJECT MOVE	MoveOperation
	MoveAction
	MoveProcess
OBJECT UPDATE	UpdateOperation
	UpdateAction
	UpdateObject

Service name or Category	Event name
OBJECT COPY	CopyOperation
	CopyAction
	CopyProcess
OBJECT SAVE AS NEW	SaveAsNewOperation
	SaveAsNewAction
OBEJCT SERVICE REGISTER EVENT	RegisterEventOperation
	RegisterEventAction
OBJECT DELETE	DeleteOperation
	DeleteAction
	DeleteSysObject
OBJECT UNREGISTER	UnRegisterEventOperation
	UnRegisterEventAction
OBJECT VALIDATE	ValidateOperation
	ValidateAction
OBJECT GET PERMISSION	GetPermissionOperation
	GetPermissionAction
OBJECT GET LINKED REPEATING STRING	GetLinkedRepeatingStringOperation
	GetLinkedRepeatingStringAction
OBJECT GET CONTENT URLS	GetContentUrlsOperation
	GetContentUrlsProcess
OBJECT MARK VERSION	MarkVersionOperation
	MarkVersionAction
OBJECT UPDATE NON CURRENT VERSION	UpdateNonCurrentVerionOperation
	UpdateNonCurrentVerionAction
OBJECT HAS ATTRIBUTES	HasAttributesOperation
	HasAttributesAction
OBJECT REMOVE RENDITION	RemoveRenditionOperation
	RemoveRenditionAction
USER AUTHENTICATION	UserAuthentication
RELATE ACTION	RelateAction
	ProcessLighObjRel
GENERIC	GetSession
	ReleaseSession

Service name or Category	Event name
	NewSessionCreate
	FindDataObjectById
	GetChronicalId
	GetCurrentVersion
	GetCurrentVersionIdentity
	CurrentVersionCheck
	ImmutableCheck
	GetPersistentObj
	GetRawId
	GetIdByQualification
	GetMultiIdsByQualification
	GetTargetObjIdentity
DOCUMENTUM UTIL	DateTimeToIDfTimeOperation
	DateTimeToIDfTimeAction
LOGIN TICKET	GetLoginTicketOperation
	GetDCTMLoginTicketOperation
	GetLoginTicketAction
	GetDCTMLoginTicketAction
SEARCH	AsyncSearchAction
	StopSearchOperation
	StopSearchAction
	ListSearchSourcesAction
	GetClustersOperation
	GetClusterAction
	GetSubclustersOperation
	GetResultsPropsOperation
	GetFacetsOperation
	GetFacetsAction
COMMENT	CreateCommentOperation
	EnumCommentOperation
	GetCommentOperation
	MarkReadOperation
	MarkUnreadOperation

Service name or Category	Event name
	UpdateCommentOperation
TASK MANAGEMENT	ClaimOperation
	WorkQueueTaskAcquire
	WorkflowTaskAcquire
	ReleaseOperation
	SuspendOperation
	SuspendUntilOperation
	WorkQueueTaskSuspend
	ResumeOperation
	WorkflowResume
	CompleteOperation
	WorkflowTaskComplete
	SetNextActivities
	SetRerunMethod
	RemoveOperation
	SetPriorityOperation
	AddAttachmentOperation
	GetAttachmentInfosOperation
	GetAttachmentsOperation
	DeleteAttachmentsOperation
	RemoveWorkflowAttachment
	AddCommentOperation
	AddComment
	GetCommentsOperation
	ForwardOperation
	DelegateOperation
	SetDelegateUser
	GetTaskInfoOperation
	GetTaskDescriptionOperation
	SetOutputOperation
	GetInputOperation
	GetOutputOperation
	NominateOperation

Service name or Category	Event name
	GetMyTaskAbstractsOperation
	GetMyTasksOperation
	QueryOperation
WORKFLOW	StartProcessOperation
	StartedWorkflowService
	SetPackageInfo
	SetAttachmentInfo
	SetAliasInfo
	SetPerformerInfo
	GetProcessInfoOperation
	RetrievePackageInfo
	RetrieveAliasInfo
	RetrievePerformerInfo
	GetProcessTempsOperation
	GetProcessDQL
	TerminateWorkflowOperation
	WorkflowHalt
	WorkflowAbort
	WorkflowDestroy
	CompleteWorkItemOperation
	WorkItemAcquire
	GetForwardActivities
	SetOutputActivities
	WorkItemComplete
LIFECYCLE	AttachOperation
	AttachAction
	ExecuteLifecycleAction
	GetLifecycleOperation
	GetLifecycleAction
	DetachOperation
	DetachAction
VIRTUAL DOCUMENT	UpdateDocumentOperation
	RetrieveSnapshotOperation

Service name or Category	Event name
	CreateSnapshotOperation
	CreateSnapshotAction
	RemoveSnapshotOperation
	DisassemblyAction
	PrepareVDocUpdateAction
USER MANAGEMENT	GetUserOperation
	GetUserAction
VIRTUAL DEPLOYMENT	GetDormacyStatusOperation
	GetDormacyStatusAction
	MakeActiveOperation
	MakeDormantOperation
	ProjectDormant
	ChangeDormancyStatusAction
	Events Related to Agent Service Call
	GetHttpSessionIdOperation
	GetSessionIdFromReqOperation
	GetStatusOperation
	GetDeploymentIdOperation
	LookupOperation
CONTEXT REGISTRY	RegisterOperation
	UnRegisterOperation
	Events Related to Analytic Service Call
	AnalyzeOperation
	Events Related to License Service Call
	RequestLicenseOperation
	CheckinOperation
VERSION CONTROL	CheckinAction
	CheckoutOperation
	CheckoutAction
	CancelCheckoutOperation
	CancelCheckoutAction
	GetCheckOutInfoOperation
	GetCheckoutInfoAction

Service name or Category	Event name
	GetVersionInfoOperation
	GetVersionInfoAction
	DeleteVersionOperation
	DeleteAllVersionsOperation
	GetCurrentOperation
	ACLCreateOperation
ACCESS CONTROL	CreateACLAction
	ACLUpdateOperation
	UpdateACLAction
	ACLGetOperation
	GetACLAction
	ACLDeleteOperation
	DeleteACLAction

7.5.5 Integrating Event Hub with Documentum REST Services

Documentum REST Services supports the Event Hub feature.

To use the feature, you must perform the following tasks:

- Set the value of *`dfc.client.should_use_eventhub`* to `true` in `dfc.properties` of Documentum REST Services.
- Set the value of *`enable`* in the `fluentd_service` and `fluendConf` categories to `true` in the `dctm-rest` category in the single All-In-One `dctm-server/values.yaml` Helm chart file.

Fluentd pushes the events that are generated by Documentum REST Services to the Apache Kafka cluster.

By default, the `Eventhub.dar` file is deployed when you deploy the Documentum Server pod. The `Eventhub.dar` file establishes the connection with Fluentd and works as a BOF module.

The `dfc.properties` of Documentum REST Services file contains the following variables for Event Hub:

- *`dfc.client.should_use_eventhub`*: Used to enable Event Hub. Set the value to `true` to enable Event Hub. The default value is `false`.
- *`dfc.client.eventhub.log_level`*: Used to define the log levels for the DFC events. Set any value from `0` to `5` where 0 is for NO LOG, 1 is for ERROR, 2 is for WARN, 3 is for INFO, 4 is for DEBUG, and 5 is for TRACE. The default value is `4`.

- *`dfc.client.eventhub.queue_size`*: Used to define the number of events in the queue buffered in Documentum Foundation Classes. The default value is `1000`.

7.5.5.1 Fluentd and Kafka with Documentum REST Services

Fluentd is packaged with Documentum Foundation Services. When you deploy the Documentum Foundation Services pod, Fluentd is deployed as a sidecar container.

You must deploy the Apache Kafka cluster as described in "Configuring Apache Kafka cluster" on page 244.

Make sure that you provide the appropriate values for the Fluentd and Kafka variables in the `dctm-rest` category in the single All-In-One `dctm-server/values.yaml` Helm chart file. Some of the important variables are *`enable`*, *`TCPPort`*, *`RESTPort`*, *`UDPPort`*, *`compressionMode`*, *`bufferingMode`*, *`flushInterval`*, *`image`*, *`kafkaTopic`*, *`kafkaUser`*, *`kafkaUsrPasswd`*, *`kafkabroker`*, and *`readiness`*.

"Deploying Documentum REST Services" on page 135 contains detailed information about the variables.

7.5.5.2 Documentum REST Services specific event names

Service name or Category	Event name
REST_URI_ACCESS	REQUEST_BEGIN
	REQUEST_END
REST_BATCH_URI_ACCESS	REQUEST_BEGIN
	REQUEST_END
REST_LOGIN	LOGIN_SUCCESS
	LOGIN_FAILED

7.5.5.3 REST URI access

For Documentum REST Services environments enabled with Event Hub captures all the API requests including request and response and then publishes them to Event Hub.

By default, Documentum REST Services captures the following information for a request:

- `REQUEST_PROTOCOL`: Protocol of the request.

  For example, HTTP or HTTPS.
- `REQUEST_URI`: Relative path of URI.

  For example, /dctm-rest/repositories/testenv.
- `REQUEST_PARAMS`: Parameter to the API endpoints.
- `REQUEST_BODY`: Payload data of the request.

- `REQUEST_METHOD`: Type of the request.

  For example, GET, POST, and so on.
- `REQUEST_HEADERS`: Request headers in the raw format.

By default, Documentum REST Services captures the following information for a response:

- `RESPONSE_STATUS_CODE`: Status code of the request.
- `RESPONSE_BODY`: Response body of the request.

Documentum REST Services adds a unique ID as `REQUEST_UNIQUE_ID` for all the requests to track both the request and response.

The `REQUEST_BODY` and `RESPONSE_BODY` information are captured depending on the value of `rest.eventhub.httpbody.enabled` in `rest-api.runtime.properties`. By default, the value of `rest.eventhub.httpbody.enabled` is `false` and so the `REQUEST_BODY` and `RESPONSE_BODY` information are not captured.

By defalut, the size of the request body and response body capture is limited to 102400 bytes (which means 100 KB). This can be configured in `rest.eventhub.httpbody.logging.limit` in `rest-api.runtime.properties`.

7.5.5.4 REST batch URI access

The REST URI Access feature is also implemented in the Batch request API. For all the batch requests, Documentum REST Services publishes individual API requests and responses.

7.5.5.5 Authentication

Documentum REST Services sends all the authentication events to Event Hub. For Event Hub, Documentum REST Services support the authentication mode as described in the following table:

Authentication type mentioned in rest-api.runtime.properties	AUTHENTICATION_TYPE in Event Hub events
basic/basic-ct	PASSWORD
otds_password/ct-otds_password	OTDS_PASSWORD
otds_token/ct-otds_token/otds_token-basic/ct-otds_token-basic	OTDS_TOKEN

The event category for authentication events is `EVENT_CATEGORY` and the category for event names are `LOGIN_SUCCESS` and `LOGIN_FAILED`. If the authentication fails for OTDS token-based login, then the `OTDS_TOKEN` field records the login-based information. All the authentication events are synchronous.

7.5.5.6 Publising event from external client

From the 22.1 release, an external client can publish an event to Event Hub using Documentum REST Services endpoint. The following API is used to call with the POST data:

```
End Point: /repositories/<REPOSITORY_NAME>/event-hub-event
```

For example:

```
{
    "event-category": "<CLIENT_EVENT_CATEGORY>", //MANDATORY FIELD
    "event-name": "<CLIENT_EVENT_NAME>", //MANDATORY FIELD
    "event-level": "2", //MANDATORY FIELD
    "event-data": "<CLIENT_EVENT_DATA>", //MANDATORY FIELD
    "username": "<username@org.com>",
      "object-id": "",
    "duration": "22",
    "application-name": "X Client",
    "client-ip": "<IP address of client>",
    "client-location": "",
    "synchronous": "true",
    "event-fields": {
        "EXTRA_EVENT_1": "Sample1",
        "EXTRA_EVENT_2": "Sample2"
    }
}
```

The *synchronous* field indicates if Event Hub is sent synchronously (`true`) or asynchronously (`false`). The clients can customize the fields in the *event-fields* field. The API returns the `201 CREATED` response.

7.6 Configuring OT KMS with Documentum Server

From the 21.4 release, Documentum Server supports to protect AEK keys using both the passphrase and OT KMS. Documentum Server protects AEK using a key created in the OpenText Key Mediation Service (OT KMS) Server. OT KMS is a solution that offers generic REST API for protecting application data.

For information about deploying OT KMS and integrating GCP or AWS KMS with OT KMS, see *OpenText Key Mediation Service Documentation*. OT KMS supports the GCP and AWS KMS/HSM service.

1. Deploy and integrate GCP or AWS KMS with OT KMS.

   For information about deploying OT KMS and integrating GCP or AWS KMS with OT KMS, see *OpenText Key Mediation Service Documentation*.

2. Configure GCP KMS on Google Cloud Platform or AWS KMS on Amazon Web Services cloud platform.

   For information about configuring GCP KMS on Google Cloud Platform, see *Google Cloud Platform Documentation*.

   For information about configuring AWS KMS on Amazon Web Services cloud platform, see *AWS Documentation*.

Important

- Make sure that the keys created on GCP/AWS KMS have required permissions for encryption and decryption.
- Make sure that the keys that are created on the GCP/AWS KMS/HSM service do not have any associated expiry date. This is a OT KMS limitation.

3. Provide the appropriate values for all the OT KMS-related variables in the following categories in the single All-In-One `dctm-server/values.yaml` Helm chart file as follows:

   a. `common-variables`:

      - Set the value of *`aek_location`* to `Remote`. The default value is `Local` which means the AEK key is protected using a passphrase.
      - Set the value of *`kms_url`* in the `https://<IP address or host of OT KMS server>:port` format.

   b. `cs-secrets`: Set the value of *`apiKey`* in the `kms` category to API key of OT KMS Server generated during the OT KMS configuration.

   c. `docbroker`: Set the value of *`masterkey_id`* in the `kms` category to the OT KMS master key ID generated during the OT KMS configuration. This is required only for certficate-based Documentum Server deployment.

   d. `content-server`: Set the value of *`masterkey_id`* in the `kms` category to the OT KMS master key ID generated during the OT KMS configuration.

Note: For more information about generating master key ID, see *OpenText Key Mediation Service Documentation*.

7.7 Scaling and descaling of Documentum Server

You can increase (scale) and decrease (descale) the number of replica pods to be spawned for the Documentum Server image.

Note: The scaling and descaling features are not supported for the connection broker.

7.7.1 Scaling Documentum Server

1. To increase the number of replica pods to be spawned for the Documentum Server image, open the single All-In-One `dctm-server/values.yaml` Helm chart file and increase the value of *`replicaCount`* in the `content-server` category to a required value. The default value is `2`.

2. After increasing the value of *`replicaCount`*, you must run the following command:

```
helm upgrade <release_name> ./dctm-server -f <location where Helm charts are
extracted>/dctm-server/platforms/<cloud platform>.yaml --namespace <name of
namespace>
```

7.7.2 Descaling Documentum Server

1. To decrease the number of replica pods to be spawned for Documentum Server image, open the single All-In-One `dctm-server/values.yaml` Helm chart file and decrease the value of *`replicaCount`* in the `content-server` category to a required value. The default value is `2`.

2. After decreasing the value of *`replicaCount`*, you must run the following command:

```
helm upgrade <release_name> ./dctm-server -f <location where Helm charts are
extracted>/dctm-server/platforms/<cloud platform>.yaml --namespace <name of
namespace>
```

Note: By default, the resources are not removed after the descaling process. OpenText does not recommend you to remove the resources.

7.8 Rotating AEK key in Documentum Server

From the 22.2 release, you can rotate AEK key within the same Documentum Server version.

Important

- Do not change the value of any other variables in the single All-In-One `dctm-server/values.yaml` Helm chart file other than the variables mentioned in this section.
- If the value of *`aek.name`* is same as the existing AEK key name, then the existing AEK key is reused.
- Upgrading of AEK key is not supported in certificate-based communication.
- AEK key rotation is supported only within the same Documentum Server version.

1. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and modify the values of the variables depending on the following rotation requirement:

- From local to local AEK key:
  - (Optional) *`aek.algorithm`* in the `cs-secrets` category. Valid values are mentioned in "cs-secrets" on page 55.
  - *`aek.name`* in the `content-server` category. See "content-server" on page 66.
  - *`aek.oldPassphrase`* in the `cs-secrets` category. See "cs-secrets" on page 55.
  - (Optional) *`aek.passphrase`* in the `cs-secrets` category. Make sure that you follow the password complexity rules. See "cs-secrets" on page 55.
- From local to remote AEK key:
  - *`aek_location`* in the `common-variables` category. You must set the value to `Remote`. See "common-variables" on page 50.
  - *`kms_url`* in the `common-variables` category. See "common-variables" on page 50.
  - (Optional) *`aek.algorithm`* in the `cs-secrets` category. Valid values are mentioned in "cs-secrets" on page 55.
  - *`aek.oldPassphrase`* in the `cs-secrets` category. See "cs-secrets" on page 55.
  - (Optional) *`aek.passphrase`* in the `cs-secrets` category. Make sure that you follow the password complexity rules. See "cs-secrets" on page 55.
  - *`kms.apiKey`* in the `cs-secrets` category. See "cs-secrets" on page 55.
  - *`aek.name`* in the `content-server` category. See "content-server" on page 66.
  - *`kms.masterkey_id`* in the `content-server` category. See "content-server" on page 66.
- From remote to remote AEK key:
  - (Optional) *`aek.algorithm`* in the `cs-secrets` category. Valid values are mentioned in "cs-secrets" on page 55.
  - *`aek.name`* in the `content-server` category. See "content-server" on page 66.
  - *`kms.masterkey_id`* in the `content-server` category. See "content-server" on page 66.
- From remote to local AEK key:
  - (Optional) *`aek.algorithm`* in the `cs-secrets` category. Valid values are mentioned in "cs-secrets" on page 55.
  - *`aek.passphrase`* in the `cs-secrets` category. Make sure that you follow the password complexity rules. See "cs-secrets" on page 55.

   - *`aek.name`* in the `content-server` category. See "content-server" on page 66.
   - *`aek_location`* in the `common-variables` category. You must set the value to `Local`. See "common-variables" on page 50.

2. Upgrade the Documentum Server Helm deployment using the following command format:

```
helm upgrade <release_name> ./dctm-server -f <location where Helm charts are
extracted>/dctm-server/platforms/<cloud platform>.yaml --namespace <name of
namespace>
```

7.9 Initializing DFC without dfc.properties

From the 22.4 release, you can initialize DFC without the `dfc.properties` file. To configure DFC at runtime without the use of the `dfc.properties` file, perform the following steps:

1. Add the required DFC properties as environment variables in the Docker Compose file in the `services` category and change the values as per your requirement:

```
dfc.data.dir=/opt/dctm
dfc.docbroker.host[0]=<Connection broker IP address for Documentum Server>
dfc.docbroker.port[0]=<Connection broker port>
dfc.globalregistry.repository=<Global registry repository name>
dfc.globalregistry.username=dm_bof_registry
dfc.globalregistry.password=<Global registry repository password>
dfc.security.ssl.truststore=<Trust store file path>
dfc.security.ssl.truststore_password=<Trust store password>
dfc.security.ssl.use_existing_truststore=false
```

2. Call `DfPreferencesLoader.load()` before DFC initialization (that is, `getLocalClient()`).

   This initializes DFC properties entries from the environment variables instead of the `dfc.properties` file.

7.10 Decoupling Documentum product image from base Tomcat image

From the 23.2 release, decoupling of product Docker image from the base Tomcat image is supported only for the Documentum REST Services and Documentum Foundation Services products.

Documentum product Docker image is decoupled from the base Tomcat image which is built on the operating system and JDK image resulting into the following two images:

- Base Tomcat image
- Documentum product image

The base Tomcat image, is common for all the Documentum products, and is used to deploy the Documentum product artifacts. The Documentum product image that contains the product artifacts is implemented as an init container image.

The base Tomcat image is maintained and consumed across the Documentum products and avoids the effort to rebuild product image every time in case of any vulnerabilities reported in the base Tomcat image and the underlying operating system and JDK image. This was not possible in earlier releases until version 22.4. With the decoupling of product image from the base Tomcat image, you just need to maintain only the impacted images to optimize the time cycle to replace the impacted image.

The execution and completion of the Documentum product init container image takes precedence before the execution of the base Tomcat image in the pod.

As part of the deployment of the Documentum product Helm, the init container is created and the Documentum product artifacts are pushed to a PVC. The PVC is created if there is no common PVC configuration enabled and the common PVC name is not provided in the `dctm-rest` or `dfs` category in the single All-In-One `dctm-server/values.yaml` Helm chart file. The Documentum product artifacts are updated in the PVC only if the existing artifacts in the PVC are older than the new artifacts that needs to be updated. This helps to avoid copying artifacts when pod is restarted.

As part of the deployment of the Documentum product Helm, in addition to the init container creation as previously described, the base Tomcat image container is created and all the artifacts are obtained from the PVC (including customizations, if any, is applicable only for Documentum REST Services), to the base Tomcat image container. This container helps to service the Documentum product functionality.

For more information about the Helm chart variables, see “common-variables” on page 50 (common variables), “Deploying and configuring Documentum REST Services on private cloud” on page 134 (for Documentum REST Services), and “Deploying and configuring Documentum Foundation Services on private cloud” on page 109 (for Documentum Foundation Services).

7.11 Using logrotate in Documentum Server

The logrotate tool is used to implement the log rotation in the Documentum Server pods. By default, the log rotation frequency (*`logrotate.interval`*) is 24 hours and this can be customized. When the size of a log file is greater than the value provided for size in *`logrotate.configmap`*, a backup log file is created when the log rotation frequency (*`logrotate.interval`*) reaches 24 hours. As and when the backup log files reaches greater than the value provided for rotate in *`logrotate.configmap`*, then the first backup log file gets deleted for optimization of space in the pod. This ensures that the total number of log files maintained at any time is same as the number provided for rotate in *`logrotate.configmap`*.

Chapter 8

Upgrading and Migrating Documentum Platform and Platform extensions applications

8.1 Upgrading Documentum Server

8.1.1 Upgrading Documentum Server to 23.2

8.1.1.1 Important tasks and notes to upgrade to 23.2

This section provides information about the important tasks that you must perform and notes that you must consider before upgrading Documentum Server to 23.2.

8.1.1.1.1 Using the Helm charts of previous deployment

Before upgrading the Documentum Server pod, make sure that you have the Helm charts used for the previous deployment ready for your reference. The configuration values defined in the Helm charts used for the previous deployment need to be used while upgrading the 22.2 or later Helm charts in the upgrade process.

Caution

- The configuration value of *`install.appserver.admin.password`* defined in the 22.2 Helm charts used in the previous deployment must be changed to a value that comply with the password complexity rules. The *Documentum Server* chapter in *OpenText Platform and Platform Extensions Installation Guide* contains detailed information. Retain the older deployment passwords as is for *`docbase.password`*, *`contentserver.globalRegistry.password`*, and *`contentserver.aek.passphrase`* in the `cs-secrets` category.
- If a mismatch exist in the configuration values, it impacts the deployment of 23.2 Helm chart and the upgrade process may fail.

8.1.1.1.2 Using the correct release name

Make sure that you use the same release name used in the previous deployment of Documentum Server for upgrading the previous deployment of Documentum Server to 23.2.

8.1.1.1.3 Using the previous release ingress annotations

If you have modified any default ingress annotations in the previous release, make sure that you copy all those ingress annotations from the `dctm-ingress` category in the single All-In-One `dctm-server/values.yaml` Helm chart file of previous release to the `dctm-server/platforms/<cloud platform>.yaml` file of 23.2.

8.1.1.1.4 Updating new migrated database host details

If you have migrated the database to a new database host, then provide the new database login credentials in the `cs-secrets` category in the single All-In-One `dctm-server/values.yaml` Helm chart file. In addition, provide the new host value of the migrated database for the *`databaseHost`* and *`databasePort`* in common variables in the single All-In-One `dctm-server/values.yaml` Helm chart file.

8.1.1.1.5 Changing the default value of AEK key algorithm

If you want to upgrade Documentum Server from certificate-based communication mode to certificate- or non-certificate based communication mode, you must change the value of the AEK key algorithm in the `docbase` category in `dctm-server/charts/cs-secrets/values.yaml` to `AES_128_CBC`. The default value is `AES_256_CBC`.

To change the AEK algorithm, perform the following tasks before the upgrade process:

1. Upgrade the Documentum Server 22.2 or 22.4 pod to 23.2 using the steps described in “Upgrading from 22.2 and 22.4 to 23.2” on page 267.

2. After the successful upgrade, you can upgrade the AEK key as described in “Rotating AEK key in Documentum Server” on page 260.

8.1.1.1.6 Upgrading Documentum Server with OT KMS

If OT KMS is not configured in your previous releases, then you must not enable OT KMS configuration during the upgrade process. For more information about OT KMS, see “Configuring OT KMS with Documentum Server” on page 258.

8.1.1.2 Upgrading from 22.2 and 22.4 to 23.2

1. Copy the value of *`sname`* provided in the previous deployment and apply the same value for all instances of *`sname`* in the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2.

2. Update the new image location details for *`image_repository`* and *`image_tag`* in the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2.

3. Copy all the required configuration values from the `common-variables` category of the previous deployment to the `common-variables` category of the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2. In addition, provide the appropriate values for the new variables (if any and required as per your requirement) in 23.2.

4. Copy all the required configuration values from the `db` category of the previous deployment to the `db` category of the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2. In addition, provide the appropriate values for the new variables (if any and required as per your requirement) in 23.2.

   Important
   Make sure that the database version is same for both the existing and new deployment.

   For example, if your 22.2 deployment is running with PostgreSQL 13.6 database container, then in your 23.2 upgrade, make sure that you change the database version to 13.6 as follows:

   - Value of *`tag`* in the `db` category in the single All-In-One `dctm-server/values.yaml` Helm chart file.
   - Value of *`version`* in the `db` category in `dctm-server/Chart.yaml`.
   - Value of *`tag`* in the `db` category in `dctm-server/charts/db/values.yaml`.

5. Copy all the required configuration values from the `docbroker` category of the previous deployment to the `docbroker` category of the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2. In addition, provide the appropriate values for the new variables (if any and required as per your requirement) in 23.2.

   Notes

   - Upgrade process is in descending order. The upgrade process starts from the second connection broker (for example, `docbroker-1`) followed by the first connection broker (for example, `docbroker-0`).
   - If you encounter any problems during the upgrade process with the new image, then the upgrade process stops automatically. In addition, you can roll back to the previous image. "Rolling back the upgrade process" on page 271 contains detailed information.

- The time for the upgrade process is approximately four minutes for each pod. If the replica count of the connection broker pod is more than one, then there is no downtime. It is because, when one pod is in the process of upgrade, the other pod serves the requests. However, if the client is connected through external connection broker, a downtime occurs for approximately four minutes.

6. Copy all the required configuration values from the `content-server` category of the previous deployment to the `content-server` category of the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2. In addition, provide the appropriate values for the new variables (if any and required as per your requirement) in 23.2.

   Notes

   - Replica count should not be modified during the initial upgrade. To modify replica count, perform another upgrade exclusively for replica count change.
   - Upgrade process is in descending order. The upgrade process starts from the second Documentum Server (for example, `documentumserver-1`) followed by the first Documentum Server (for example, `documentumserver-0`).
   - While upgrading the Documentum Server pod, the existing Documentum Server pod is deleted and new Documentum Server pod is created. VCTs and PVCs remain as is and the new pods continue to mount the old VCTs and PVCs.
   - If you encounter any problems during the upgrade process with the new image, then the upgrade process stops automatically. In addition, you can roll back to the previous image. “Rolling back the upgrade process” on page 271 contains detailed information.
   - If the replica count of the Documentum Server pod is more than one, then there is no downtime. It is because, when one pod is being upgraded, the other pod serves the requests. If the client chooses to connect to a specific Documentum Server pod (server config object) and if that Documentum Server pod is in the process of upgrade, then a downtime occurs until the pod is upgraded.

7. Copy all the required configuration values from the `cs-logging-configMap` category of the previous deployment to the `cs-logging-configMap` category of the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2. In addition, provide the appropriate values for the new variables (if any and required as per your requirement) in 23.2.

8. Copy all the required configuration values from the `dctm-ingress` category of the previous deployment to the `dctm-ingress` category of the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2. In addition, provide the appropriate values for the new variables (if any and required as per your requirement) in 23.2.

9. Copy all the required configuration values from the `cs-dfc-properties` category of the previous deployment to the `cs-dfc-properties` category of the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2. In addition, provide the appropriate values for the new variables (if any and required as per your requirement) in 23.2.

10. Copy all the required configuration values from the `otds` category of the previous deployment to the `otds` category of the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2. In addition, provide the appropriate values for the new variables (if any and required as per your requirement) in 23.2.

11. Disable the value of *`enabled`* (if enabled) for all Documentum client applications such as Documentum Administrator, Documentum REST Services, and so on. If you want to enable the Event Hub feature during the upgrade process, make sure that you follow the information provided in "Integrating Event Hub" on page 244.

12. Upgrade the 22.2 or 22.4 Documentum Server pod using the following command format:

    ```
    helm upgrade <release_name> ./dctm-server -f ./dctm-server/platforms/<cloud
    platform>.yaml
    ```

13. Verify the status of the successful upgrade using the following steps:

    a. Verify if all the pods are recreated successfully after the upgrade using the following example command:

       ```
       kubectl get pods
       ```

       If the pods are recreated correctly after the upgrade process, then the READY state of each pod is displayed as `3/3` (one for Documentum Server, one for Graylog, and one for Event Hub (if enabled)) and the STATUS state is shown as `Running`.

    b. Verify the installation log files using the following example command:

       ```
       kubectl logs dctmcs-0 -c dctmcs
       ```

       If the upgrade is successful, no errors are reported in the log files.

    c. Verify if the repository is updated successfully. Log in to the pod and run a command in the following command format:

       ```
       kubecl exec -ti <name of pod> -c <name of container> bash
       ```

       Verify the `<repository_name>.log` file located at `/opt/dctm/dba/logs`. If the upgrade is successful, no errors are reported in the log file.

    d. Verify the Documentum Server version. Log in to the pod and run a command in the following command format:

       ```
       kubectl exec -ti <name of pod> -c <name of container> bash
       ```

       Run the following command:

       ```
       documentum -version
       ```

If the upgrade is successful, the upgraded version of Documentum Server is displayed.

e. Verify if all the existing ingress URLs are working correctly.

f. Verify if you can create a sample document and check the document successfully into the upgraded repository using the IAPI commands.

Note: Verification can also be done for the secondary repository. Make sure that you specify the name of the repository in the following format in the IAPI command:

```
<name of repository>.HA1<name of repository>
```

g. Verify if New Relic monitors are created for Documentum Server and Java Method Server and that you are able to generate the metrics.

h. Verify if the logs are generated on the Graylog server console for Documentum Server and connection broker.

i. Verify the status of the pod. If the upgrade is successful, the status of the pod is active.

Note: The `DM_DOCBROKER_E_NETWORK_ERROR` and `DM_DOCBROKER_E_CONNECT_FAILED_EX` errors are reported in the repository log files of the secondary Documentum Server pod after an upgrade process. You can ignore these errors.

8.1.1.3 Enabling certificate-based communication in upgraded 23.2 enviroment

If you want to upgrade from a non-certificate 22.2 or 22.4 environment to a non-certificate 23.2 environment where both the environments were deployed using single Helm and then you want to enable certificate-based communication in the upgraded 23.2 environment, you must perform the following tasks:

1. Upgrade the non-certificate 22.2 or 22.4 environment to a non-certificate 23.2 environment.

   "Upgrading from 22.2 and 22.4 to 23.2" on page 267 contains the instructions.

2. Set the value of enable of *`customUpgrade`* to true in `content-server` ("content-server" on page 66) in the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2. In addition, provide the appropriate values for the new variables (if any and required as per your requirement) in 23.2. Then, upgrade the updated 23.2 environment using the `helm upgrade` command.

3. Set the value of *`use_certificates`* to `false` in `common-variables` ("common-variables" on page 50) in the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2. In addition, provide the appropriate values for the new variables (if any and required as per your requirement) in 23.2.

4. Set the value of *`use_certificate`* to `true` in `docbroker` in `content-server` ("content-server" on page 66) in the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2. In addition, provide the appropriate values for the new variables (if any and required as per your requirement) in 23.2.

5. Specify all the certificate-related configuration values for *`docbroker`* in `cssecrets` ("cs-secrets" on page 55) in the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2. In addition, provide the appropriate values for the new variables (if any and required as per your requirement) in 23.2.
6. Upgrade the updated 23.2 environment using the `helm upgrade` command.
7. Modify the value of *`use_certificate`* to `*use_certificates` in `docbroker` in `content-server` ("content-server" on page 66) in the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2.
8. Modify the value of *`use_certificates`* to `true` in `common-variables` ("cs-secrets" on page 55) in the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2.
9. Modify the value of *`enable`* of *`customUpgrade`* to `false` in `content-server` ("content-server" on page 66) in the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2.
10. Specify all the certificate-related configuration values for `docbase` in "cs-secrets" on page 55 in the single All-In-One `dctm-server/values.yaml` Helm chart file of 23.2.
11. Upgrade the updated 23.2 environment using the helm upgrade command.
12. Verify if the upgraded environment uses certificate-based communication.

8.1.2 Rolling back the upgrade process

Rolling back the upgrade process (rolling back to the previous image) is recommended when the upgrade process fails or when you encounter errors using the new image. Perform the following steps:

To roll back to the 22.2 or 22.4 image if AEK key is not upgraded:

1. Retrieve the details of history using the following command format:
   ```
   helm history <release_name>
   ```
2. Roll back to the previous image using the following command format:
   ```
   helm rollback <release_name> <revision>
   ```

To roll back to the 22.2 or 22.4 image if AEK key is upgraded:

1. Retrieve the details of history using the following command format:
   ```
   helm history <release_name>
   ```
2. Upgrade the AEK key.

   a. Change the value of *`aek.name`* in the `content-server` category to the same key name mentioned in the 22.2 image.

   b. Change the value of *`aek.location`* in the `content-server` category to the same location mentioned in the 22.2 image.

   c. Change the value of *`aek.algorithm`* in the `cs-secrets` category to the same key algorithm mentioned in the 22.2 image.

   d. Do not change the value of *`aek.passphrase`* in the `cs-secrets` category. It should be same for the upgraded image and the previous image.

   e. Upgrade the Documentum Server 23.2 Helm deployment using the following command format:

      ```
      helm upgrade <release_name> ./dctm-server -f ./dctm-server/platforms/<cloud
      platform>.yaml
      ```

   Make sure that the Helm upgrade is successful and the Documentum Server pod is up and running.

3. Roll back to the previous image using the following command format:

   ```
   helm rollback <release_name> <revision>
   ```

To roll back to the 22.2 or 22.4 image if installOwner value is changed and upgraded:

1. Retrieve the details of history using the following command format:

   ```
   helm history <release_name>
   ```

2. Upgrade the installation owner (*`installOwner`*).

   a. Change the value of *`installOwner`* in 23.2 in the `common-variables` category to the same value mentioned in the `cs-secrets` category in the single All-In-One `dctm-server/values.yaml` Helm chart file) of 22.2 or 22.4.

   b. Upgrade the Documentum Server 23.2 Helm deployment using the following command format:

      ```
      helm upgrade <release_name> ./dctm-server -f ./dctm-server/platforms/<cloud
      platform>.yaml
      ```

   Make sure that the Helm upgrade is successful and the Documentum Server pod is up and running.

3. Roll back to the previous image using the following command format:

   ```
   helm rollback <release_name> <revision>
   ```

Notes

- Database schema changes are not reverted when you roll back.
- Rolling back within the same Documentum Server version is not supported if you have upgraded the AEK key during the upgrade process.

8.2 Migrating on-premises Documentum applications to cloud platform

8.2.1 Migrating Documentum Server, connection broker, and database to cloud platform

The migration of Documentum Server, connection broker, and database from on-premises to the cloud platforms is supported only for:

- Documentum Server on Windows/SQL Server or Oracle Linux/PostgreSQL
- Documentum client applications

8.2.1.1 Prerequisites

1. Stop the repository.
2. Take a backup of the database. You can use any third-party tools of your choice.
3. Take a backup of the contents of the `dba`, `config` and `data` folders in the existing Documentum Server.

Note: If any of your objects in your on-premises environment contains FQDN, then you must run the migration utility to change the host name (without FQDN).

8.2.1.2 Creating secrets

1. Open the single All-In-One `dctm-server/values.yaml` Helm chart file and provide the same values that you have used on the on-premises environment.

   For example, the "Deploying and configuring Documentum Server on private cloud" on page 47 section contains detailed information.

2. Set the value of *`enabled`* in the `cs-secrets` category in the single All-In-One `dctm-server/values.yaml` Helm chart file to `true`.

3. Set the value of *`enabled`* of all other categories (such as `cs-logging-configMap`, `docbroker`, `content-server`, and so on other than `cs-secrets`) to `false`.

4. Store the secret values using the following command format:

```
helm install <release_name> <location where Helm charts are extracted>/dctm-server -
f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
platform>.yaml --namespace <name of namespace>
```

8.2.1.3 Creating database

The migration of the PostgreSQL database only is supported.

1. Set the value of *`enabled`* in the `db` category in the single All-In-One `dctm-server/values.yaml` Helm chart file to `true`.
2. Set the value of *`enabled`* of all other categories (such as `cs-logging-configMap`, `docbroker`, `content-server`, and so on other than `db`) to `false`.
3. Create or use the existing database. Do one of the following tasks:
   a. Use the existing database installed in your cloud platform as follows:
      i. Log in as a `postgres` user.

         For example:
         ```
         #su postgres
         ```
      ii. Create new user with an encrypted password using the following command format:
         ```
         # psql
         # create role <name of user> noinherit login password '<password to
         authenticate user>';
         ```
         For example:
         ```
         # psql
         # create role testenv noinherit login password '024$2356*651';
         ```
      iii. Create new database and grant the permissions using the following command format:
         ```
         # psql
         # create database dm_<docbasename>_docbase with encoding='UTF8'
         LC_COLLATE='C' LC_CTYPE='C' CONNECTION LIMIT=-1 owner <name of owner>
         TEMPLATE template0;
         # ALTER DATABASE dm_<docbasename>_docbase SET SEARCH_PATH to <docbasename>;
         # GRANT ALL ON database dm_<docbasename>_docbase to <name of owner>;
         ```
         For example:
         ```
         # psql
         # CREATE DATABASE dm_testenv_docbase WITH ENCODING='UTF8' LC_COLLATE='C'
         LC_CTYPE='C' CONNECTION LIMIT=-1 OWNER testenv TEMPLATE template0;
         # ALTER DATABASE dm_testenv_docbase SET SEARCH_PATH to testenv;
         # GRANT ALL ON DATABASE dm_testenv_docbase to testenv;
         ```
   b. Create new database in your cloud platform as follows:
      i. Create the database pod using the following command format:
         ```
         helm install <release_name>
         ```
         For example:
         ```
         helm install db
         ```
      ii. Log in to the database pod using the following command format:
         ```
         kubecl exec -ti <name of new database pod> bash
         ```
         For example:

```
kubecl exec -ti mdb-0 bash
```

iii. Log in as a `postgres` user.

For example:

```
#su postgres
```

iv. Create new user with an encrypted password using the following command format:

```
# psql
# create role <name of user> noinherit login password '<password to
authenticate user>';
```

For example:

```
# psql
# create role testenv noinherit login password '024$2356*651';
```

v. Create new database and grant the permissions using the following command format:

```
# psql
# create database dm_<docbasename>_docbase with encoding='UTF8'
LC_COLLATE='C' LC_CTYPE='C' CONNECTION LIMIT=-1 owner <name of owner>
TEMPLATE template0;
# ALTER DATABASE dm_<docbasename>_docbase SET SEARCH_PATH to <docbasename>;
# GRANT ALL ON database dm_<docbasename>_docbase to <name of owner>;
```

For example:

```
# psql
# CREATE DATABASE dm_testenv_docbase WITH ENCODING='UTF8' LC_COLLATE='C'
LC_CTYPE='C' CONNECTION LIMIT=-1 OWNER testenv TEMPLATE template0;
# ALTER DATABASE dm_testenv_docbase SET SEARCH_PATH to testenv;
# GRANT ALL ON DATABASE dm_testenv_docbase to testenv;
```

4. Import the database to your cloud platform created in Step 3. You can use any third-party tools of your choice to import the database. Make sure that all the tables are migrated successfully.

8.2.1.4 Creating connection broker

To create a connection broker in your cloud platform, perform the steps from Step 4.d to Step 8 in “Deploying Documentum Server” on page 49. You must perform the following tasks:

1. Set the value of *`enabled`* in the `docbroker` category to `true`.

2. Set the value of *`enabled`* of all other categories (such as `cs-secrets`, `cs-logging-configMap`, `content-server`, and so on other than `docbroker`) to `false`.

3. Deploy the connection broker pod using the following command format:

```
helm install <release_name> <location where Helm charts are extracted>/dctm-server -
f <location where Helm charts are extracted>/dctm-server/platforms/<cloud
platform>.yaml --namespace <name of namespace>
```

8.2.1.5 Migrating Documentum Server

1. Migrate the NFS store and configurations to your cloud platform as follows:

   a. Create PVC for the `data` and `config` folders with the following sample YAML content:

      Note: Replace *`csServiceName`* with the service name of Documentum Server used while creating the Documentum Server pod.

```
##migration of PVC
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: csServiceName-pvc
spec:
accessmodes:
- ReadWriteMany
storageClassName: trident-NFS
resources:
requests:
storage: 3Gi
```

```
##dummy pod to create configs and data
apiVersion: v1
kind: Pod
metadata:
name: migrationcs
spec:
containers:
- name: migrationcs
image: alpine:latest
command:
- sleep
- "999999"
volumeMounts:
- mountPath: /opt/dbaconfig
name: migpvc
subpath: dba_mig/csServiceName
mountpath: /opt/migdata
name: migpvc
subpath: data/csServiceName
volumes:
- name: migpvc
persistentVolumeClaim:
claimName: csServiceName-pvc
```

   b. Copy the `dba` folder to the `data_pvc/opt/dbaconfig` location.

   c. Copy the `config` folder to the `data_pvc/opt/dbaconfig` location.

   d. Copy the contents of the `data` folder to the `data_pvc/opt/migdata` location. Make sure that the `data` folder owner is set to `installowner`.

2. Create the Documentum Server pod.

   a. Update the following in the single All-In-One `dctm-server/values.yaml` Helm chart file:

      - Set the value of *`enabled`* in the `content-server` category in the single All-In-One `dctm-server/values.yaml` Helm chart file to `true`.

- Set the value of *`enabled`* of all other categories (such as `cs-logging-configMap`, `docbroker`, and so on other than `content-server`) to `false`.
- Set the value of *`docbase.existing`* to `true`.
- Set the value of *`migratecs`* to `true`.
- (Only if you want to migrate from Windows) Set the value of *`migfromwindows`* to `true`.
- Make sure that you provide the same values for *`secret`* you created "Creating secrets" on page 273, same values for *`database`* you created in "Creating database" on page 274, and same values for *`docbroker`* created in "Creating connection broker" on page 275. In addition, make sure all the other values used in on-premises Documentum Server is also updated in the single All-In-One `dctm-server/values.yaml` Helm chart file.

b. Set the value of *`oldDocumentumHome`* in `.\dctm-server\charts\content-server\values.yaml` to previous deployment of Documentum Server (on-premises) path.

c. To access S3 store, perform the following tasks in the `content-server` category in `dctm-server\values.yaml`:

   i. Set the value of *`updateExistingStore`* to `true`.

   ii. Provide the value for *`storeListUpdate`* with the S3 store name created in the on-premises environment.

   iii. Provide the values for *`ProxyHostUpdate`* and *`ProxyPortUpdate`*.

d. To create a Documentum Server pod in your cloud platform, perform the steps from Step 4.e to Step 8 in "Deploying Documentum Server" on page 49.

e. Connect to the repository using IAPI.

f. Update the custom location objects.

g. After the migration, you must run the migration utility.

Note: After the migration process, if any of your objects in your on-premises environment contains FQDN, then you must run the migration utility to change the host name.

8.2.2 Migrating Documentum Server only to cloud platform

Use the database and data (file system) available in your on-premises environment. Perform the following tasks:

1. Create secrets. "Creating secrets" on page 273 contains detailed information.
2. Create the connection broker pod. "Creating connection broker" on page 275 contains detailed information.
3. Create the Documentum Server pod. "Migrating Documentum Server" on page 276 contains detailed information. Instead of copying the `data` folder as mentioned in Step 1.d, you must mount the `data` folder to the `data_pvc/opt/migdata` location. Make sure that the `data` folder owner is set to `installowner`.

   Note: If you want to use Documentum Server installed in an on-premises environment, make sure that your cloud platform has all the permission to access the Documentum Server installed in the on-premises environment.

8.2.3 Migrating Documentum client applications to cloud platform

Migrating Documentum client applications is same as installing or deploying the Documentum client applications in your cloud platform. Install the Documentum client application in your cloud platform. The *Installation Guide* or *Deployment Guide* contains detailed information for your client product. In addition, you must provide the values for Documentum Server (either the values of Documentum Server installed in an on-premises environment or cloud platform).

Note: If you want to use Documentum Server installed in an on-premises environment, make sure that your cloud platform has all the permission to access the Documentum Server installed in the on-premises environment.