



## OpenText™ Documentum™ Platform and Platform Extensions

### **Cloud Deployment Guide**

Deploy and configure cloud-native products on different cloud platforms.

EDCSYCD230400-IGD-EN-02

---

## **OpenText™ Documentum™ Platform and Platform Extensions**

### **Cloud Deployment Guide**

EDCSYCD230400-IGD-EN-02

Rev.: 2024-Jan-16

This documentation has been created for OpenText™ Documentum™ Platform and Platform Extensions CE 23.4.

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>

#### **© 2024 Open Text**

Patents may cover this product, see <https://www.opentext.com/patents>.

#### **Disclaimer**

##### **No Warranties and Limitation of Liability**

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, Open Text Corporation and its affiliates accept no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication.

---

# Table of Contents

<b>1</b>	<b>Overview .....</b>	<b>17</b>
<b>2</b>	<b>Documentum Platform and Platform Extensions applications on Docker .....</b>	<b>19</b>
2.1	Installing Docker .....	19
2.2	Using container image tags .....	19
2.3	Deploying and configuring Documentum Server on Docker .....	20
2.3.1	Prerequisites .....	20
2.3.2	Common notes .....	21
2.3.3	Creating the Documentum Server Red Hat Enterprise Linux/Oracle Docker image .....	22
2.3.3.1	Prerequisites .....	22
2.3.3.1.1	Hardware requirements .....	23
2.3.3.1.2	Software requirements .....	23
2.3.3.2	Configuring the Red Hat Enterprise Linux/Oracle Docker image .....	23
2.3.3.3	Configuring Documentum Server in Red Hat Enterprise Linux/Oracle Docker image .....	25
2.3.4	Exporting environment variables for storing passwords to Docker .....	26
2.3.5	Deploying and configuring Documentum Server on Stateless images .....	26
2.3.6	Deploying and configuring of Documentum Server HA on Docker .....	27
2.3.7	Limitations .....	28
2.3.8	Troubleshooting .....	28
2.4	Deploying and configuring Independent Java Method Server on Docker .....	28
2.4.1	Prerequisites .....	28
2.4.2	Deploying IJMS on Docker .....	29
2.4.2.1	Network details .....	30
2.4.2.2	Port details .....	30
2.4.2.3	Volume details .....	31
2.4.3	Limitations .....	31
2.4.4	Troubleshooting .....	31
2.5	Deploying and configuring Documentum Administrator on Docker .....	31
2.5.1	Prerequisites .....	31
2.5.2	Deploying Documentum Administrator on Docker .....	32
2.6	Deploying and configuring Branch Office Caching Services on Docker .....	33
2.7	Deploying and configuring Documentum Foundation Services on Docker .....	34
2.8	Deploying and configuring Documentum Records Client on Docker ....	35
2.8.1	Prerequisites .....	35
2.8.2	Deploying Documentum Records Client DAR files on Docker .....	36

2.8.3	Deploying Documentum Records Client on Docker .....	39
2.8.4	Using Up, Down, Start, and Stop options in Docker commands .....	42
2.8.5	Deploying Records Queue Manager on Docker .....	42
2.9	Deploying and configuring REST Services on Docker .....	45
2.10	Deploying and configuring Thumbnail Server on Docker .....	47
2.11	Deploying and configuring Content Connect on Docker .....	48
2.11.1	Prerequisites .....	48
2.11.2	Deploying Content Connect on Docker .....	49
2.11.3	Deploying Content Connect on the client machine .....	52
<b>3</b>	<b>Documentum Platform and Platform Extensions applications on private cloud platform .....</b>	<b>53</b>
3.1	Defining common variables .....	53
3.2	Deploying and configuring Documentum Server on private cloud .....	56
3.2.1	Prerequisites .....	56
3.2.2	Deploying Documentum Server .....	58
3.2.3	Limitations .....	105
3.2.4	Troubleshooting .....	105
3.3	Deploying and configuring Independent Java Method Server on private cloud .....	106
3.3.1	Prerequisites .....	106
3.3.2	Deploying Independent Java Method Server .....	107
3.3.3	Limitations .....	111
3.3.4	Troubleshooting .....	111
3.4	Deploying and configuring Documentum Administrator on private cloud .....	111
3.4.1	Prerequisites .....	111
3.4.2	Deploying Documentum Administrator .....	111
3.4.3	Limitations .....	116
3.4.4	Troubleshooting .....	116
3.5	Deploying and configuring D2 components on private cloud .....	117
3.5.1	Prerequisites .....	117
3.5.2	Container image tags .....	120
3.5.3	Deploying D2 Components on private cloud .....	121
3.5.3.1	Troubleshooting .....	132
3.5.4	Setting up OpenText Directory Services (OTDS) .....	132
3.5.4.1	Automatic setup using config.yml .....	132
3.5.4.2	Manual setup .....	133
3.5.5	Deploying a custom D2 configuration during deployment .....	134
3.5.5.1	Custom D2 configuration .....	134
3.5.5.2	Custom Dar(s) deployment .....	136
3.5.5.3	Custom Dar and D2 configuration .....	137
3.5.6	Deploying customizations .....	138

3.5.6.1	Customizing D2 Webapp containers using single init container .....	138
3.5.6.2	Customizing D2 Webapp containers using hook scripts .....	141
3.5.6.3	Customizing D2 Webapps containers using single init container including hook scripts (hybrid approach) .....	143
3.5.6.4	Language Pack support .....	144
3.5.7	Set the acs_base_url to localhost and acs_supported_protocol to s3 for S3 store/Google Cloud Store .....	145
3.5.8	Documentum Reports (DTR) .....	145
3.5.8.1	Enabling/Disabling Reports with D2 .....	145
3.5.8.2	Post-deployment steps .....	147
3.5.8.2.1	Configuring OTDS for OTIV .....	148
3.5.9	Deploying optional components for Documentum .....	150
3.5.9.1	Deploying Documentum Connector for Core (DCC) .....	150
3.5.9.2	Documentum Workflow Designer .....	153
3.5.9.2.1	Deploying the Documentum Workflow Designer .....	153
3.5.9.2.2	Post-deployment instructions .....	154
3.5.9.2.3	Enabling OTDS for Documentum Workflow Designer .....	154
3.5.9.3	Deploying Documentum Content Connect .....	154
3.5.9.3.1	Enabling OTDS for Content Connect .....	156
3.5.9.3.2	Content Connect Admin Console Configurations .....	156
3.5.9.4	Enabling Documentum Records Client/Records Queue Manager ....	157
3.5.10	Checking deployments .....	158
3.5.11	Upgrading or patching D2 CE single Helm deployment .....	159
3.5.11.1	Section 1: From 20.3 to 20.4 .....	161
3.5.11.2	Section 2: From 20.4 to 21.2 or a newer version .....	161
3.5.11.3	Section 3: From 21.2 to 21.4 .....	162
3.5.11.4	Section 4: Upgrade from 22.1 or higher .....	162
3.5.12	Rollback .....	164
3.5.13	Cleaning up the deployment .....	165
3.5.14	SMTP Configuration .....	166
3.5.15	Brava certificate configuration .....	167
3.5.16	Graylog logging .....	167
3.5.17	Deploying D2 in Independent Java Method Server (IJMS) on private cloud .....	167
3.5.18	Troubleshooting .....	168
3.5.19	Documentum 23.4 Multi-Repository support .....	169
3.5.19.1	Overview of high-level steps to support Multi-Repository for cloud deployments .....	169
3.5.19.2	Pre-requisites .....	170
3.5.19.3	Additional pre-requisites for deploying an additional repository on AWS .....	172
3.5.19.4	Additional pre-requisites for deploying additional repository on Azure .....	172
3.5.19.4.1	Create a public IP address on Azure .....	172

3.5.19.5	Updates to be done in Documentum single Helm chart .....	173
3.5.19.5.1	Documentum/Documentum-components.yaml file changes .....	173
3.5.19.5.2	Documentum/values.yaml file changes to support additional repositories .....	173
3.5.19.5.3	Changes to dockerimages-values.yaml .....	183
3.5.19.5.4	Changes to platforms/<platform>.yaml .....	183
3.5.19.6	Deploy additional repository .....	185
3.5.19.7	Post-deployment steps .....	185
3.5.19.7.1	Additional post-deployment steps when additional repository is deployed on AWS .....	187
3.5.19.7.2	OTDS configuration post-deployment .....	189
3.5.19.8	Known issues .....	194
3.5.19.9	Post-deployment validation .....	194
3.6	Deploying and configuring Documentum Records Client on private cloud .....	194
3.6.1	Prerequisites .....	194
3.6.2	Deploying Documentum Records Client DAR files .....	195
3.6.3	Deploying Documentum Records Client .....	198
3.6.4	Deploying Records Queue Manager .....	203
3.6.5	Limitations .....	207
3.6.6	Troubleshooting .....	207
3.7	Deploying and configuring Documentum Foundation Services on private cloud .....	208
3.7.1	Prerequisites .....	208
3.7.2	Deploying Documentum Foundation Services .....	208
3.7.3	Limitations .....	217
3.7.4	Troubleshooting .....	217
3.8	Deploying and configuring Documentum REST Services on private cloud .....	217
3.8.1	Prerequisites .....	217
3.8.2	Deploying Documentum REST Services .....	217
3.8.3	Extensibility .....	224
3.8.4	Limitations .....	225
3.8.5	Troubleshooting .....	225
3.9	Deploying and configuring Documentum Content Management Interoperability Services on private cloud .....	225
3.9.1	Prerequisites .....	225
3.9.2	Deploying Documentum Content Management Interoperability Services .....	225
3.9.3	Limitations .....	229
3.9.4	Troubleshooting .....	230
3.10	Deploying and configuring Documentum Reports on private cloud ....	230
3.10.1	Prerequisites .....	230

3.10.2	Deploying Documentum Reports (dtrinstaller and dtrbase) .....	230
3.10.2.1	dtrinstaller .....	230
3.10.2.2	dtrbase .....	230
3.10.3	Limitations .....	236
3.10.4	Troubleshooting .....	236
3.11	Deploying and configuring Documentum Workflow Designer on private cloud .....	236
3.11.1	Prerequisites .....	236
3.11.2	Deploying Documentum Workflow Designer .....	237
3.11.3	Importing batch processes using WFD cloud utility .....	243
3.11.3.1	Prerequisites .....	243
3.11.3.2	Deploying WorkFlow Designer Cloud utility .....	243
3.11.4	Limitations .....	245
3.11.5	Troubleshooting .....	245
3.12	Deploying and configuring Content Connect on private cloud .....	245
3.12.1	Prerequisites .....	245
3.12.2	Deploying Content Connect .....	246
3.12.3	Deploying Content Connect on the client machine .....	250
3.12.4	Limitations .....	250
3.12.5	Troubleshooting .....	250
3.12.6	Upgrading Content Connect .....	251
3.13	Deploying and configuring Extended ECM Documentum for Microsoft 365 on private cloud .....	251
3.13.1	Prerequisites .....	251
3.13.2	Deploying Extended ECM Documentum for Microsoft 365 .....	252
3.13.3	Configuring Extended ECM Documentum for Microsoft 365 .....	257
3.13.4	Deploying the language pack .....	258
3.13.4.1	Deploying the language pack using init container .....	259
<b>4</b>	<b>Documentum Platform and Platform Extensions applications on Microsoft Azure cloud platform .....</b>	<b>261</b>
4.1	Deploying and configuring Documentum Server on Microsoft Azure cloud platform .....	261
4.1.1	Kubernetes platform .....	261
4.1.1.1	Prerequisites .....	261
4.1.1.2	Deploying Documentum Server .....	264
4.1.1.3	Limitations .....	267
4.1.1.4	Troubleshooting .....	267
4.1.2	Red Hat OpenShift platform .....	267
4.1.2.1	Azure Red Hat OpenShift platform cluster .....	268
4.1.2.1.1	Prerequisites .....	268
4.1.2.1.2	Deploying Documentum Server .....	269
4.1.2.2	Red Hat OpenShift cluster on bare metal .....	270

4.1.2.2.1	Prerequisites .....	271
4.1.2.2.2	Deploying Documentum Server .....	272
4.1.2.3	Limitations .....	273
4.1.2.4	Troubleshooting .....	273
4.2	Deploying and configuring Documentum Administrator on Microsoft Azure cloud platform .....	274
4.2.1	Kubernetes platform .....	274
4.2.1.1	Prerequisites .....	274
4.2.1.2	Deploying Documentum Administrator .....	274
4.2.1.3	Limitations .....	276
4.2.1.4	Troubleshooting .....	276
4.3	Deploying D2 components on Microsoft Azure .....	276
4.3.1	Kubernetes platform .....	276
4.3.1.1	Overview .....	276
4.3.1.2	Prerequisites .....	276
4.3.1.3	Deploying D2 Components .....	284
4.3.1.4	Limitations .....	284
4.3.1.5	Troubleshooting .....	284
4.3.2	Red Hat OpenShift platform .....	284
4.3.2.1	Prerequisites .....	284
4.3.2.2	Deploying Documentum D2 .....	286
4.3.2.3	Limitations .....	290
4.3.2.4	Troubleshooting .....	290
4.4	Deploying and configuring Documentum Records Client on Microsoft Azure cloud platform .....	291
4.4.1	Kubernetes platform .....	291
4.4.1.1	Prerequisites .....	291
4.4.1.2	Deploying Documentum Records Client .....	292
4.4.1.3	Limitations .....	292
4.4.1.4	Troubleshooting .....	292
4.5	Deploying and configuring Documentum Foundation Services on Microsoft Azure cloud platform .....	293
4.5.1	Kubernetes platform .....	293
4.5.1.1	Prerequisites .....	293
4.5.1.2	Deploying Documentum Foundation Services .....	293
4.5.1.3	Limitations .....	294
4.5.1.4	Troubleshooting .....	294
4.6	Deploying and configuring Documentum REST Services on Microsoft Azure cloud platform .....	294
4.6.1	Kubernetes platform .....	294
4.6.1.1	Prerequisites .....	294
4.6.1.2	Deploying Documentum REST Services .....	295
4.6.1.3	Configuring external IP address .....	296

4.6.1.4	Limitations .....	296
4.6.1.5	Troubleshooting .....	296
4.6.2	Red Hat OpenShift platform .....	296
4.6.2.1	Prerequisites .....	296
4.6.2.2	Deploying Documentum REST Services .....	296
4.6.2.3	Configuring external IP address .....	296
4.6.2.4	Limitations .....	296
4.6.2.5	Troubleshooting .....	296
4.7	Deploying and configuring Documentum Content Management Interoperability Services on Microsoft Azure cloud platform .....	297
4.7.1	Kubernetes platform .....	297
4.7.1.1	Prerequisites .....	297
4.7.1.2	Deploying Documentum Content Management Interoperability Services .....	297
4.7.1.3	Configuring external IP address .....	298
4.7.1.4	Limitations .....	298
4.7.1.5	Troubleshooting .....	298
4.8	Deploying and configuring Documentum Reports on Microsoft Azure cloud platform .....	298
4.8.1	Kubernetes platform .....	299
4.8.1.1	Prerequisites .....	299
4.8.1.2	Deploying Documentum Reports .....	299
4.8.1.3	Limitations .....	300
4.9	Deploying and configuring Content Connect on Microsoft Azure cloud platform .....	300
4.9.1	Prerequisites .....	300
4.9.2	Deploying Content Connect .....	301
4.9.2.1	Deploying Content Connect on the client machine .....	301
4.9.3	Limitations .....	301
4.9.4	Troubleshooting .....	301
4.9.5	Upgrading .....	302
4.10	Deploying and configuring Extended ECM Documentum for Microsoft 365 on Microsoft Azure cloud platform .....	302
4.10.1	Prerequisites .....	302
4.10.2	Deploying Extended ECM Documentum for Microsoft 365 .....	303
4.10.3	Limitations .....	303
4.10.4	Troubleshooting .....	303
4.11	Deploying Documentum Server with Azure PaaS .....	304
4.11.1	Deploying Documentum Server with Azure Database for PostgreSQL - Single server .....	304
4.11.2	Deploying Documentum Server with Azure Database for PostgreSQL - Flexible server .....	304
4.12	Configuring Documentum Server for Azure OAuth authentication .....	305

<b>5</b>	<b>Documentum Platform and Platform Extensions applications on Google cloud platform .....</b>	<b>307</b>
5.1	Deploying and configuring Documentum Server on GCP .....	307
5.1.1	Kubernetes platform .....	307
5.1.1.1	Prerequisites .....	307
5.1.1.2	Deploying Documentum Server .....	309
5.1.1.3	Limitations .....	310
5.1.1.4	Troubleshooting .....	310
5.2	Deploying and configuring Documentum Administrator on GCP .....	310
5.2.1	Kubernetes platform .....	310
5.2.1.1	Prerequisites .....	310
5.2.1.2	Deploying Documentum Administrator .....	311
5.2.1.3	Limitations .....	312
5.2.1.4	Troubleshooting .....	312
5.3	Deploying D2 components on Google Cloud Platform .....	312
5.3.1	Overview .....	312
5.3.2	Prerequisites .....	312
5.3.3	Accessing the Webapp from Outside the Cluster .....	317
5.3.4	Deploying D2 Components .....	317
5.3.5	Setting up HTTP(S) Load Balancing with Ingress .....	318
5.3.6	Limitations .....	318
5.4	Deploying and configuring Documentum Records Client on GCP .....	319
5.4.1	Kubernetes platform .....	319
5.4.1.1	Prerequisites .....	319
5.4.1.2	Deploying Documentum Records Client .....	319
5.4.1.3	Limitations .....	321
5.4.1.4	Troubleshooting .....	321
5.5	Deploying and configuring Documentum Foundation Services on GCP .....	322
5.5.1	Kubernetes platform .....	322
5.5.1.1	Prerequisites .....	322
5.5.1.2	Deploying Documentum Foundation Services .....	322
5.5.1.3	Limitations .....	323
5.5.1.4	Troubleshooting .....	323
5.6	Deploying and configuring Documentum REST Services on GCP .....	323
5.6.1	Kubernetes platform .....	323
5.6.1.1	Prerequisites .....	323
5.6.1.2	Deploying Documentum REST Services .....	324
5.6.1.3	Configuring external IP address .....	325
5.6.1.4	Limitations .....	325
5.6.1.5	Troubleshooting .....	325

5.7	Deploying and configuring Documentum Content Management Interoperability Services on GCP .....	325
5.7.1	Kubernetes platform .....	325
5.7.1.1	Prerequisites .....	325
5.7.1.2	Deploying Documentum Content Management Interoperability Services .....	325
5.7.1.3	Limitations .....	327
5.7.1.4	Troubleshooting .....	327
5.8	Deploying and configuring Documentum Reports on GCP .....	327
5.8.1	Kubernetes platform .....	327
5.8.1.1	Prerequisites .....	327
5.8.1.2	Deploying Documentum Reports .....	328
5.8.1.3	Limitations .....	328
5.8.1.4	Troubleshooting .....	329
5.9	Deploying and configuring Documentum Workflow Designer on GCP .....	329
5.9.1	Kubernetes platform .....	329
5.9.1.1	Prerequisites .....	329
5.9.2	Deploying Documentum Workflow Designer .....	330
5.9.3	Importing batch processes using WFD cloud utility .....	331
5.9.3.1	Prerequisites .....	331
5.9.3.2	Deploying WorkFlow Designer Cloud utility .....	331
5.9.4	Limitations .....	333
5.9.5	Troubleshooting .....	333
5.10	Deploying and configuring Content Connect on GCP .....	334
5.10.1	Prerequisites .....	334
5.10.2	Deploying Content Connect .....	334
5.10.2.1	Deploying Content Connect on the client machine .....	335
5.10.3	Limitations .....	335
5.10.4	Troubleshooting .....	335
5.10.5	Upgrading .....	335
5.11	Deploying and configuring Extended ECM Documentum for Microsoft 365 on GCP .....	335
5.11.1	Prerequisites .....	335
5.11.2	Deploying Extended ECM Documentum for Microsoft 365 .....	336
5.11.3	Limitations .....	336
5.11.4	Troubleshooting .....	336
5.12	Creating PostgreSQL database instance on GCP .....	336
5.13	Configuring Elastifile file system on GCP .....	338
5.13.1	Configuring ECFS in fully managed mode .....	339
5.13.2	Configuring ECFS in self managed mode .....	339
<b>6</b>	<b>Documentum Platform and Platform Extensions applications on Amazon Web Services cloud platform .....</b>	<b>341</b>

6.1	Deploying and configuring Documentum Server on AWS cloud platform .....	341
6.1.1	Kubernetes platform .....	341
6.1.1.1	Prerequisites .....	341
6.1.1.1.1	Prerequisites for deployment in AWS EKS EC2 environment .....	341
6.1.1.1.2	Prerequisites for deployment in AWS EKS Fargate environment .....	346
6.1.1.2	Deploying Documentum Server .....	349
6.1.1.3	Limitations .....	353
6.1.1.4	Troubleshooting .....	353
6.2	Deploying and configuring Documentum Administrator on AWS cloud platform .....	353
6.2.1	Kubernetes platform .....	353
6.2.1.1	Prerequisites .....	353
6.2.1.2	Deploying Documentum Administrator .....	353
6.2.1.3	Limitations .....	354
6.2.1.4	Troubleshooting .....	354
6.3	Deploying D2 components on Amazon Web Services (AWS) .....	354
6.3.1	Overview .....	355
6.3.2	Prerequisites .....	355
6.3.3	Deploying D2 on AWS .....	362
6.3.4	Limitations .....	369
6.4	Deploying and configuring Documentum Records Client on AWS cloud platform .....	370
6.4.1	Kubernetes platform .....	370
6.4.1.1	Prerequisites .....	370
6.4.1.2	Deploying Documentum Records Client .....	370
6.4.1.3	Limitations .....	371
6.4.1.4	Troubleshooting .....	371
6.5	Deploying and configuring Documentum Foundation Services on AWS cloud platform .....	371
6.5.1	Kubernetes platform .....	371
6.5.1.1	Prerequisites .....	371
6.5.1.2	Deploying Documentum Foundation Services .....	372
6.5.1.3	Limitations .....	373
6.5.1.4	Troubleshooting .....	373
6.6	Deploying and configuring Documentum REST Services on AWS cloud platform .....	373
6.6.1	Kubernetes platform .....	373
6.6.1.1	Prerequisites .....	373
6.6.1.2	Deploying Documentum REST Services .....	373
6.6.1.3	Limitations .....	374
6.6.1.4	Troubleshooting .....	374

6.7	Deploying and configuring Documentum Content Management Interoperability Services on AWS cloud platform .....	375
6.7.1	Kubernetes platform .....	375
6.7.1.1	Prerequisites .....	375
6.7.1.2	Deploying Documentum Content Management Interoperability Services .....	375
6.7.1.3	Limitations .....	376
6.7.1.4	Troubleshooting .....	376
6.8	Deploying and configuring Documentum Reports on AWS cloud platform .....	376
6.8.1	Kubernetes platform .....	376
6.8.1.1	Prerequisites .....	376
6.8.1.2	Deploying Documentum Reports .....	377
6.8.1.3	Limitations .....	378
6.8.1.4	Troubleshooting .....	378
6.9	Deploying and configuring Content Connect on AWS cloud platform ..	378
6.9.1	Prerequisites .....	378
6.9.2	Deploying Content Connect .....	379
6.9.3	Limitations .....	379
6.9.4	Troubleshooting .....	379
6.10	Deploying and configuring Extended ECM Documentum for Microsoft 365 on AWS cloud platform .....	380
6.10.1	Prerequisites .....	380
6.10.2	Deploying Extended ECM Documentum for Microsoft 365 .....	380
6.10.3	Limitations .....	381
6.10.4	Troubleshooting .....	381
<b>7</b>	<b>Features for all cloud platforms .....</b>	<b>383</b>
7.1	Integrating New Relic .....	383
7.1.1	Integrating New Relic with connection broker, Documentum Server, and Java Method Server .....	383
7.1.2	Integrating New Relic with Documentum Administrator .....	385
7.1.3	Integrating New Relic with Documentum Records Client .....	385
7.1.4	Integrating New Relic with Documentum Foundation Services .....	386
7.1.5	Integrating New Relic with Documentum REST Services .....	387
7.1.6	Integrating New Relic with Documentum Content Management Interoperability Services .....	388
7.1.7	Integrating New Relic with Documentum Reports .....	389
7.1.8	Integrating New Relic with Content Connect .....	390
7.1.8.1	Troubleshooting New Relic integration .....	390
7.1.9	Integrating New Relic with Extended ECM Documentum for Microsoft 365 .....	391
7.2	Integrating Graylog .....	391
7.2.1	Integrating Graylog with Documentum Server .....	391

7.2.1.1	Prerequisites .....	391
7.2.1.2	Deploying Graylog Collector Sidecar and Graylog Sidecar using Documentum Helm charts .....	392
7.2.1.2.1	Deploying Graylog Collector Sidecar (legacy) .....	392
7.2.1.2.2	Deploying Graylog Sidecar .....	394
7.2.2	Integrating Graylog with Documentum REST Services .....	396
7.2.3	Integrating Graylog with Content Management Interoperability Services .....	397
7.3	Configuring containers and pods for Kubernetes liveness probe .....	398
7.3.1	Checking liveness of connection broker .....	398
7.3.2	Checking liveness of Documentum Server .....	399
7.3.3	Checking liveness of Java Method Server .....	399
7.3.4	Checking liveness of Documentum Administrator .....	400
7.3.5	Checking liveness of Documentum Foundation Services .....	400
7.3.6	Checking liveness of Documentum Records Client .....	400
7.3.7	Checking liveness of Documentum REST Services .....	401
7.3.8	Checking liveness of Documentum Content Management Interoperability Services .....	401
7.4	Integrating OTDS .....	402
7.4.1	Integrating OTDS with Documentum Server .....	402
7.4.2	Configuring OTDS roles for Documentum Records Client .....	402
7.4.3	Integrating OTDS with Documentum Foundation Services .....	403
7.4.4	Integrating OTDS with Documentum REST Services .....	403
7.4.5	Integrating OTDS with Documentum Content Management Interoperability Services .....	403
7.5	Integrating Event Hub .....	403
7.5.1	Integrating Event Hub with Documentum Server .....	404
7.5.1.1	Fluentd and Kafka with Documentum Server .....	404
7.5.2	Retrieving messages .....	405
7.5.3	Integrating Event Hub with Documentum Foundation Services .....	405
7.5.3.1	Fluentd and Kafka with Documentum Foundation Services .....	406
7.5.3.2	Documentum Foundation Services specific event names .....	406
7.5.4	Integrating Event Hub with Documentum REST Services .....	413
7.5.4.1	Fluentd and Kafka with Documentum REST Services .....	414
7.5.4.2	Documentum REST Services specific event names .....	415
7.5.4.3	REST URI access .....	415
7.5.4.4	REST batch URI access .....	416
7.5.4.5	Authentication .....	416
7.5.4.6	Publishing event from external client .....	416
7.6	Scaling and descaling of Documentum Server .....	417
7.6.1	Scaling Documentum Server .....	417
7.6.2	Descaling Documentum Server .....	418
7.7	Supporting multiple repositories in one namespace .....	418

7.8	Rotating AEK in Documentum Server .....	419
7.9	Initializing DFC without dfc.properties .....	420
7.10	Decoupling Documentum product image from base Tomcat image ...	420
7.11	Using logrotate in Documentum Server .....	422
7.12	Using logcleanup in Documentum Server .....	422
7.13	Editing configmap details for D2 webapps .....	423
7.14	Manually deploying D2 artifacts .....	430
7.14.1	Extracting D2 artifacts .....	430
7.14.2	Adding D2.war and D2-Config.war to the App server .....	432
7.14.3	Deploying plugins .....	432
7.14.4	Installing DARS .....	433
7.14.5	Adding D2-Smartview.war to the App server .....	434
7.14.6	Installing RPS components .....	434
7.15	Intelligent Viewing configuration .....	435
7.15.1	Enabling OTIV in D2 single Helm .....	435
7.15.2	Configuring OTDS for IV .....	435
7.15.2.1	Syncing OTIV users to docbase .....	435
7.15.2.2	Allocating the license to users .....	435
7.15.2.3	Adding/Updating oAuth clients and system attributes .....	436
7.15.3	Configuring Intelligent Viewing .....	436
<b>8</b>	<b>Upgrading and Migrating Documentum applications .....</b>	<b>437</b>
8.1	Upgrading Documentum Server .....	437
8.1.1	Upgrading Documentum Server to 23.4 .....	437
8.1.1.1	Important tasks and notes to upgrade to 23.4 .....	437
8.1.1.1.1	Using the Helm charts of previous deployment .....	437
8.1.1.1.2	Using the correct release name .....	437
8.1.1.1.3	Using the previous release ingress annotations .....	438
8.1.1.1.4	Updating new migrated database host details .....	438
8.1.1.1.5	Modifying service account name .....	438
8.1.1.2	Upgrading from 22.4 and 23.2 to 23.4 .....	439
8.1.1.3	Enabling certificate-based communication in upgraded 23.4 environment .....	442
8.1.2	Rolling back the upgrade process .....	443
8.2	Upgrading Documentum Administrator .....	446
8.2.1	Rolling back the upgrade process .....	447
8.3	Upgrading Documentum REST Services .....	447
8.3.1	Rolling back the upgrade process .....	448
8.4	Upgrading Documentum Foundation Services .....	448
8.4.1	Rolling back the upgrade process .....	449
8.5	Migrating on-premises Documentum applications to cloud platform ..	449
8.5.1	Migrating Documentum Server, connection broker, and database to cloud platform .....	449

## Table of Contents

---

8.5.1.1	Prerequisites .....	449
8.5.1.2	Creating secrets .....	449
8.5.1.3	Creating database .....	450
8.5.1.4	Creating connection broker .....	452
8.5.1.5	Migrating Documentum Server .....	452
8.5.2	Migrating Documentum Server only to cloud platform .....	454
8.5.3	Migrating Documentum client applications to cloud platform .....	455

# 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 containers. This approach defines a common layout within the Documentum CE solutions and includes 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 products and components (for example, Documentum Server, Documentum client, Documentum Administrator, Documentum REST Services, and so on) in the respective sections within a single Helm chart. After providing the appropriate values according to 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 (RHEL) 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	23.4.0 or 23.4

4. Download the documentum\_server\_<release-version>\_docker\_compose\_scripts.tar Docker Compose file from My Support.
5. For Docker Documentum Server image, the location of <Installdir> 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 iutils
$ 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 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=dadmin
--build-arg THUMBNAIL_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 Documentum 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 DM\_CRYPTO\_MIN\_PASSWORD\_LENGTH in environment.

### 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 26.

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 26.

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 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	23.4.0 or 23.4



**Note:** Download the Docker Compose file from 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_MGMT_PORT>=9184"
- "<APP_SERVER_MGMT_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_MGMT_PORT`, it is exposed as `<APP_SERVER_MGMT_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 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	23.4.0 or 23.4



**Note:** Download the Docker Compose file from 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 111 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 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	23.4.0 or 23.4

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 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	23.4.0 or 23.4

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 Documentum Foundation Services 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/dfs/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 Documentum Foundation Services 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 Documentum Foundation Services 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 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	23.4.0 or 23.4
dctm-records-darinstallation	23.4.0 or 23.4
dctm-rqm	23.4.0 or 23.4



**Note:** Download the Docker Compose file from 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/darinstalation). 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/darinstalation.
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 40](#)).



**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 darinstalation_darinstaller_logs
local darinstalation_ext-conf
```



**Note:** Since the Docker Compose file is available in the /home/darinstalation directory, the docker-compose up command creates two volumes (darinstalation\_darinstaller\_logs and darinstalation\_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/darinstalation_ext-conf/_data", "Labels": null,
  "Scope": "local"
}
]
[root@oraclelinux ~]# docker volume inspect darinstaller_logs [
{
  "Name": " darinstalation_darinstaller_logs", "Driver": "local",
  "Mountpoint": "/var/lib/docker/volumes/darinstalation_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`
- `Transformation.dar`
- `RM-DoD51015v3-Classified-Record.dar`
- `RM-DoD51015v3-Standard-Record.dar`
- `rmce.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>
    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>
    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 40](#)).



**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: <USER_NAME>
    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_IT,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:
      "recordscontainer"
    ports:
      - "<APP SERVER 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: <USER_NAME>
    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_IT,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=<DFCOPROPERTIES ::separated>
    container_name:
      "recordscontainer"
    ports:
      - "<APPERVER_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: <USER_NAME>
    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_IT,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=<DFCOPROPERTIES ::separated>
    container_name:
      "recordscontainer"
    ports:
      - "<APPERVER_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 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	23.4.0 or 23.4



**Note:** Download the Docker Compose file from 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 44](#)).



**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=AAAAEKGZADhrsRexHk2iEo+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 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	23.4.0 or 23.4

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<br/><base_Tomcat_image_name>:<image_tag> tail -f /dev/null
```

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<br/><docker-container-id>:</opt/tomcat/webapps>
```

8. Log in to the container created in **step 6** and start the Tomcat web application server.

9. 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 9** 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 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	23.4.0 or 23.4

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> tail -f /dev/null
```

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. Log in to the container created in [step 6](#) and update the `server.xml` configuration file of Tomcat with the SSL certificate information.
9. Start the Tomcat web application server.
10. 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 10](#) 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 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	23.4.0 or 23.4
dctm-content-connect-dbinit	23.4.0 or 23.4

6. Download the Docker Compose files and .env file for the Docker environment from 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=mssql1
```

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:
  ccdbinitialize:
    image: "registry.opentext.com/dctm-content-connect-dbinit:<image tag>"
    container_name: "ccdbinitializecontainermssql12"
    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:23.4.0000.0220"
    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 Defining common variables

This section describes the list of common variables for all products in the single All-In-One documentum/values.yaml Helm chart file. Provide the appropriate values as described in the following table:

**Table 3-1: common-variables**

Name	Description
rwoStorage	Name of the storage class with persistent volume claim (PVC) access mode as ReadWriteOnce.
rwmstorage	Name of the storage class with PVC access mode as ReadWriteMany.
env	Name of the cluster space.
namespace	Name of namespace used for deployment.
docbase	Name of the repository.
csSecrets	Name of the secret configuration file.
installOwnerUsername	User name of the installation owner.
installOwnerPassword	Password of the installation owner.
globalRegistryUsername	User name of the global registry user.

Name	Description
globalRegistryPassword	Password of the global registry user.
trustStorePassword	Password of the trust store.
otdsEnabled	Indicates if OTDS is enabled.
otdsadminPassword	Password of the OTDS administrator.
otdsUserName	Database user name to connect to OTDS.
otdsUserPassword	Database password to connect to OTDS.
documentumserviceaccount	Service account information.
tomcatbase_usecommonpvc	Indicates to use the common PVC if the PVC already exists.
tomcatbase_commonpvcname	Name of the common PVC.
ingressUrl	Ingress URL.
aekLocation	Path where Application Encryption Key (AEK) is available.
connectMode	Type of the connection mode.
oauthClient	OAuth client information.
ingressDomain	Domain name of the ingress controller in the cluster namespace.
ingressProtocol	Type of the communication mode.
webappServiceType	Service type of the web application.
grayLogEnable	Indicates if Graylog is enabled for use.
graylogServer	Server details based on your Graylog server configuration.
graylogPort	Port reserved based on your Graylog server configuration.
persistLogs	Indicates if Graylog is disabled for xPlore.
newrelic	Indicates if New Relic is enabled.   <b>Note:</b> 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.
newrelicProxyHost	IP address of the proxy server.
newrelicProxyPort	Port reserved for the New Relic server.
newrelicProxyProtocol	Protocol to connect to the pod.
licenseKey	License key of New Relic.

Name	Description
useCertificate	Indicates if certificate-based communication is enabled.
dbrServiceName	Service name of connection broker. You must provide the fully qualified domain name (FQDN) and it must not be greater than 59 characters.
dbrConfigmapName	ConfigMap name of the connection broker.
openshiftEnable	Indicates if deployment in Red Hat OpenShift is enabled.
openshiftTls	Indicates if deployment in Red Hat OpenShift is enabled with HTTPS configuration.
externalAccessEnabled	Indicates if the external access of the connection broker is enabled.
kafka_admin_user_name	Administrator user name used to communicate among the replicas in the Apache Kafka cluster.
kafka_admin_user_password	Administrator password used to communicate among the replicas in the Apache Kafka cluster.
kafka_topic_name	Topic name provided while deploying the Apache Kafka cluster for storing the events.
kafkaBrokerList	Broker list information of Apache Kafka.
fluentd	Indicates if Fluentd is enabled for use.
fluentdTcpPort	Port on which Fluentd is listening for TCP connection.   <b>Note:</b> This port is used for communication between DFC-based applications and Fluentd.
fluentdRestPort	Port on which Fluentd is listening for REST connection.
fluentdUdpPort	Port on which Fluentd is listening for UDP connection.   <b>Note:</b> Documentum Server uses the UDP connection.
eventLogLevel	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.
dfcRPCTracing	Indicates if DFC RPC tracing log is enabled.

Name	Description
ccExtension	Extension name for Content Connect to be used in accessing Admin Console.
cslogrotate	Indicates if the usage of the logrotate tool is enabled.
jmsServiceName	JMS service name created by Documentum Server.
secrets_Change	Indicates if any changes to variables are made in <code>dctm-server.cs-secrets</code> . 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.
msClientId	Microsoft Azure app registration client ID.



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

## 3.2 Deploying and configuring Documentum Server on private cloud

### 3.2.1 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 for all products except for Documentum REST Services, Documentum Foundation Services, and Documentum Content Management Interoperability Services) from OpenText Container Registry. Use Alpine Linux Docker images for Documentum REST Services, Documentum Foundation Services, and Documentum Content Management Interoperability Services. 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 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	23.4.0 or 23.4
dctm-admin	23.4.0 or 23.4
dctm-dfs	23.4.0 or 23.4
dctm-records	23.4.0 or 23.4
dctm-records-darinstallation	23.4.0 or 23.4
dctm-rqm	23.4.0 or 23.4
dctm-reports-installer	22.4.2
dctm-reports-base	22.4.2
dctm-rest	23.4.0 or 23.4
dctm-cmis	23.4.0 or 23.4
dctm-tomcat	23.4.0 or 23.4



**Note:** The `dctm-tomcat` image (Alpine Linux) is required only for Documentum REST Services, Documentum Foundation Services, and Documentum Content Management Interoperability Services. “Decoupling Documentum product image from base Tomcat image” on page 420 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-<release-version>.tar file from My Support.

### 3.2.2 Deploying Documentum Server

1. Extract the Helm chart downloaded from 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 according to your requirement.
4. Make sure that you have provided values for the required common variables in “[Defining common variables](#)” on page 53.
5. Make sure that `dctm-server.docbroker.enabled`, `dctm-server.content-server.enabled`, `dctm-server.cs-secrets.enabled`, `dctm-server.cs-logging-configMap.enabled`, `dctm-server.cs-dfc-properties.enabled`, `dctm-server.dctm-ingress.enabled`, and `dctm-server.db.enabled` are all set to true in the <location where Helm charts are extracted>/documentum-components.yaml file.
6. Make sure that all the required Docker image details are updated in `dctm-server` in the <location where Helm charts are extracted>/dockerimages-values.yaml file.
7. **Optional** You can create a configuration file for each environment that you deploy. The values in a configuration file override the values of `documentum/values.yaml`. You can find a basic template of the `config` file in `documentum/config/configuration.yml`. If you are using this configuration file, make sure that you pass this `configuration.yml` in the Helm deployment commands.
8. Open the single All-In-One `documentum/values.yaml` Helm chart file and perform the following tasks:
  - a. Provide the appropriate values for the variables in `dctm-server.cs-secrets` to pass them to your templates as described in “[cs-secrets](#)” on page 60.
  - b. Provide the appropriate values for the variables in `dctm-server.db` to pass them to your templates as described in “[db](#)” on page 67.
  - c. Provide the appropriate values for the variables in `dctm-server.docbroker` to pass them to your templates as described in “[docbroker](#)” on page 68.
  - d. Provide the appropriate values for the variables in `dctm-server.content-server` to pass them to your templates as described in “[content-server](#)” on page 72.

- e. Provide the appropriate values for the variables in `dctm-server.cs-logging-configMap` to pass them to your templates as described in “[cs-logging-configMap](#)” on page 90.
  - f. Make sure that the ingress controller is deployed in the cluster. Provide the appropriate values for the variables in `dctm-server.dctm-ingress` to pass them to your templates as described in “[dctm-ingress](#)” on page 95.
  - g. Provide the appropriate values for the variables in `dctm-server.cs-dfc-properties` to pass them to your templates as described in “[cs-dfc-properties](#)” on page 96.
  - h. Provide the appropriate values for the variables in `otds` to pass them to your templates as described in “[otds](#)” on page 97.
9. **Optional** Note that there are other variables that have their own default values specified in the following individual Helm charts:
- `documentum/charts/dctm-server/charts/cs-secrets/values.yaml`
  - `documentum/charts/dctm-server/charts/db/values.yaml`
  - `documentum/charts/dctm-server/charts/docbroker/values.yaml`
  - `documentum/charts/dctm-server/charts/content-server/values.yaml`
  - `documentum/charts/dctm-server/charts/cs-logging-configMap/values.yaml`
  - `documentum/charts/dctm-server/charts/cs-dfc-properties/values.yaml`
  - `documentum/charts/dctm-server/charts/dctm-ingress/values.yaml`
  - `documentum/charts/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 `documentum/values.yaml`, the value provided in `documentum/values.yaml` is used for deployment.

10. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where
Helm charts are extracted>/documentum-components.yaml --namespace <name of
namespace>
```

where `<config>` is `-extra-large-enhanced`, `-extra-large-standard`, `-large-enhanced`, `-large-standard`, `-medium-enhanced`, `-medium-large-enhanced`, `-medium-large-standard`, `-medium-standard`, `-small-enhanced`, `-small-medium-enhanced`, `-small-medium-standard`, `-small-standard`, or `-test-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 install dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/platforms/cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /opt/temp/documentum/documentum-components.yaml --namespace onedctmns
```



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

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

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

12. Verify the status of the deployment of the 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 sections in `documentum-components/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 3-2: cs-secrets**

Name	Description
enabled	Indicates if the secret configuration is enabled. The default value is <code>true</code> .
secret.name	Name of the secret configuration file. This variable uses the value that you specified for <code>csSecrets</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
docbroker.certificate	Certificate information of the connection broker.  <b>Note:</b> If you have set the value of <code>use_certificate</code> to <code>true</code> in <code>dctm-server.docbroker</code> in <a href="#">step 8.c</a> 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.
docbroker.certificate.password	Keystore password.

Name	Description
docbroker.certificate.aekpassphrase	Passphrase to protect the AEK file. Specify the value provided for <code>contentserver.aek.passphrase</code> .
docbroker.certificate.trustpassword	Trust store password.
docbroker.certificate.pemCertPrivKey	Private key in the PEM format. Make sure that you change the default value and is consistent.
docbroker.certificate.pemCertificate	Certificate in the PEM format. Make sure that you change the default value and is consistent.
docbase.password	<p>Password to connect to the repository. Make sure that you follow the password complexity rules. The <i>Documentum Server</i> chapter in <i>OpenText Documentum Platform and Platform Extensions Installation Guide</i> contains detailed information about password complexity rules. The default value is <code>Password@1234567890</code>.</p> <p>If you want to change the default value of minimum length of password, you can change it using the <code>DM_CRYPTO_MIN_PASSWORD_LENGTH</code> environment variable in <code>extraEnv</code>.</p> <p>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 <code>docbaseOwnerPasswordChange</code> to <code>true</code> in <code>dctm-server.content-server</code>.</p>
docbase.licenses.recordManager	License details for the Records Manager optional module.
docbase.licenses.prm	License details for the Physical Records Manager optional module.
docbase.licenses.fedRecdService	License details for the Federation Records Services optional module.
docbase.licenses.rps	License details for the Retention Policy Services optional module.
docbase.licenses.snaplockstore	License details for the Content Services for SnapLock optional module.

Name	Description
docbase.licenses.storageaware	<p>License details for the Storage aware devices optional module.</p> <p> <b>Notes</b></p> <ul style="list-style-type: none"> <li>When you provide the license details for the Storage aware devices module, record the following capabilities: <ul style="list-style-type: none"> <li>New deployment: Encryption, compression, and de-duplication are enabled in your default filestore.</li> <li>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.</li> </ul> </li> <li>When you provide the license details for the Storage aware devices module, you must enable the High-Volume Server license.</li> </ul>
docbase.licenses.trusted	<p>License details for the Trusted Content Services optional module.</p> <p> <b>Notes</b></p> <ul style="list-style-type: none"> <li>When you provide the license details for the Trusted Content Services module, record the following capabilities: <ul style="list-style-type: none"> <li>New deployment: Encryption, compression, and de-duplication are enabled in your default filestore.</li> <li>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.</li> </ul> </li> <li>When you provide the license details for the Trusted Content Services module, you must enable the High-Volume Server license.</li> </ul>
docbase.licenses.highVolumeServer	License details for the High-Volume Server optional module.

Name	Description
contentserver.installOwner.userName	<p>User name of the installation owner. This variable uses the value that you specified for <code>installOwner</code> in <code>common-variables</code> in “<a href="#">Defining common variables</a>” on page 53.</p> <p> <b>Note:</b> From the 20.4 release, when you log in to the pods, the installation owner is <code>dmadmin</code> instead of <code>root</code> (used in pods prior to the 20.4 version). From the 23.2 release, when you log in to the pods, the default user can be any installation owner including <code>dmadmin</code>.</p>
contentserver.installOwner.password	Password of the installation owner. The default value is <code>password</code> .
contentserver.globalRegistry.password	<p>Password of the global registry. Make sure that you follow the password complexity rules. The <i>Documentum Server</i> chapter in <i>OpenText Documentum Platform and Platform Extensions Installation Guide</i> contains detailed information about password complexity rules. The default value is <code>Password@1234567890</code>.</p> <p>If you want to change the default value of minimum length of password, you can change it using <code>DM_CRYPTO_MIN_PASSWORD_LENGTH</code> in <code>extraEnv</code>.</p>
contentserver.aek.algorithm	<p>Algorithm for the encryption. Valid values are:</p> <ul style="list-style-type: none"> <li>• <code>AES_128_CBC</code></li> <li>• <code>AES_192_CBC</code></li> <li>• <code>AES_256_CBC</code></li> </ul>
contentserver.aek.oldPassphrase	Value of the older AEK passphrase ( <code>aek.passphrase</code> ) must be provided if you are upgrading AEK. The default value is null.
contentserver.aek.passphrase	<p>Passphrase to protect the AEK file. Make sure that you follow the password complexity rules. The <i>Documentum Server</i> chapter in <i>OpenText Documentum Platform and Platform Extensions Installation Guide</i> contains detailed information about password complexity rules. The default value is <code>Password@1234567890</code>.</p> <p>If you want to change the default value of minimum length of password, you can change it using <code>DM_CRYPTO_MIN_PASSWORD_LENGTH</code> in <code>extraEnv</code>.</p>

Name	Description
contentserver.certificate	Certificate information of the Documentum Server.   <b>Note:</b> If you have set the value of <code>docbroker.certificate.use_certificate</code> to true in step 8.c 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.
contentserver.certificate.password	Keystore password.
contentserver.certificate.trustpassword	Trust store password.
contentserver.certificate.pemCertPrivKey	Private key in the PEM format. Make sure that you change the default value and is consistent.
contentserver.certificate.pemCertificate	Certificate in the PEM format. Make sure that you change the default value and is consistent.
contentserver.install.appserver.admin.password	Administrator password of the application server. Make sure that you follow the password complexity rules. The <i>Documentum Server</i> chapter in <i>OpenText Documentum Platform and Platform Extensions Installation Guide</i> 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 <code>DM_CRYPTO_MIN_PASSWORD_LENGTH</code> in <code>extraEnv</code> .
contentserver.install.root.password	Password of the installation root directory. The default value is <code>password</code> .
database.userName	User name to access the database. The default value is <code>postgres</code> .  For information about deploying Documentum Server with Microsoft Azure PaaS, see “ <a href="#">Deploying Documentum Server with Azure PaaS</a> ” on page 304.
database.password	Password to access the database. The default value is <code>password</code> .

Name	Description
database.certificate	Certificate information of the database. Make sure that you change the default value according to your requirement.
thumbnailServer.appServerPassword	Password to access Thumbnail Server. The default value is password@123.
email.smtpUser	User name to access the SMTP server.
email.smtpPass	Password of the user to access the SMTP server.
email.smtpSSLCertificate	Certificate information of the SMTP server.
s3Store.s3StoreBaseUrl	URL that Documentum Server uses to communicate with the Amazon S3 store. The URL format is <code>http://X.X.X.X/&lt;BUCKET&gt;</code> .  The <i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.
s3Store.s3StoreCredentialID	Name of the user accessing the S3 store. Use the S3 Tenant Owner. The default value is <code>nocredentials</code> if you want to use S3 role-based access.  The <i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.
s3Store.s3StoreCredentialKEY	Password of the user accessing the S3 store. Use the Object Access Key. The default value is <code>nocredentials</code> if you want to use S3 role-based access.  The <i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.
s3Store.certificate	SSL certificate of S3 store. Make sure that you change the default value according to your requirement.
restStore.restStoreBaseUrl	URL that Documentum Server uses to communicate with the REST store.
restStore.restStoreCredentialID	Name of the user accessing the REST store. Use the REST Tenant Owner.   <b>Note:</b> The value for this variable is not required for Google Cloud store.

Name	Description
restStore.restStoreCredentialKEY	Password of the user accessing the REST store. Use the Object Access Key.   <b>Note:</b> The value for this variable is not required for Google Cloud store.
restStore.azurecertificate	SSL certificate of Azure Blob store. Make sure that you change the default value according to your requirement.
restStore.gcpcertificate	SSL certificate of Google Cloud store. Make sure that you change the default value according to your requirement.
newrelic.license_key	License key information of New Relic.
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.
ingress.tlskey	Base64-encoded TLS key in one line without any carriage returns or line wrap.
openshifttls.enable	Indicates if deployment in Red Hat OpenShift is enabled with HTTPS configuration. This variable uses the value that you specified for openshiftTls in common-variables in “Defining common variables” on page 53. <ul style="list-style-type: none"> <li>• Kubernetes: Set the value of openshiftTls to false.</li> <li>• Red Hat OpenShift: Set the value of openshiftTls to true only if you want to configure the HTTPS mode for communication. This creates secrets of TLS type with the &lt;secret-name&gt;-tls name. Then, use this secret for the HTTPS configuration.</li> </ul>
clients.drServiceAccountUser	Service account user name to connect to Documentum Reports. The default value is dcmreports.
clients.drServiceAccountPassword	Service account password to connect to Documentum Reports. The default value is Password@1234567890.
clients.preferencePassword	Preference password information. The default value is webtopUser@12345.
clients.presetPassword	Preset password information. The default value is webtopUser@12345.



**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 `dctm-server.cs-secrets`, perform the next task in [step 8.b](#) in “[Deploying Documentum Server](#)” on page 58.

**Table 3-3: db**

Name	Description
enabled	Indicates if the database pod deployment is enabled. The default value is <code>true</code> .  <b>Note:</b> To use an external database of your choice such as Virtual Machine (VM) or database service to ignore the deployment of <code>documentum/values.yaml</code> , set the value to <code>false</code> .
serviceName	Unique name.
secret.name	Name of the secret configuration file. This variable uses the value that you specified for <code>csSecrets</code> in <code>common-variables</code> in “ <a href="#">Defining common variables</a> ” on page 53.
persistentVolume.storageClass	Storage class of the persistent volume (PV). This variable uses the value that you specified for <code>rwoStorage</code> in <code>common-variables</code> in “ <a href="#">Defining common variables</a> ” on page 53.
configMap.configName	Name for the configuration.
configMap.volumeName	Name for the volume.
alterParam.max_connections	Maximum number of concurrent connections to the database server.
alterParam.default_statistics_target	Sets the default statistics target for table columns. It is set without a column-specific target set through <code>ALTER TABLE SET STATISTICS</code> .

Name	Description
alterParam.maintenance_work_mem	Maximum amount of memory that must be used by maintenance operations.
alterParam.effective_cache_size	Effective size of the disk cache that is available to a single query.
alterParam.wal_buffers	Amount of shared memory used for Write-Ahead Log (WAL) data that is not written to disk.
alterParam.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 <code>false</code> .
rootsquash.ispg12orgreater	PostgreSQL database version in the root squash environment. The default value is <code>false</code> .
openshift.enable	Indicates if deployment in Red Hat OpenShift is enabled. This variable uses the value that you specified for <code>openshiftEnable</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> . <ul style="list-style-type: none"> <li>• Kubernetes: Set the value of <code>openshiftEnable</code> to <code>false</code>.</li> <li>• Red Hat OpenShift: Set the value of <code>openshiftEnable</code> to <code>true</code>.</li> </ul>

After providing the appropriate values for the variables in `dctm-server.db`, perform the next task in [step 8.c](#) in [“Deploying Documentum Server” on page 58](#).

**Table 3-4: docbroker**

Name	Description
enabled	Indicates if the connection broker pod deployment is enabled. The default value is <code>true</code> .
serviceName	Service name of the connection broker. This variable uses the value that you specified for <code>dbrServiceName</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .

Name	Description
serviceAccount.createServiceAccount	Indicates to create a new service account. The default value is false.  ! <b>Important</b> If you set the value to true, you must provide a unique service account name manually for serviceAccount.serviceAccountName.
serviceAccount.serviceAccountName	Name of the service account. This variable uses the value that you specified for documentumServiceAccount in common-variables in “Defining common variables” on page 53.
openshift.enable	Indicates if deployment in Red Hat OpenShift is enabled. This variable uses the value that you specified for openshiftEnable in common-variables in “Defining common variables” on page 53. <ul style="list-style-type: none"> <li>Kubernetes: Set the value of openshiftEnable to false.</li> <li>Red Hat OpenShift: Set the value of openshiftEnable to true.</li> </ul>
secret.name	Name of the secret configuration file. This variable uses the value that you specified for csSecrets in common-variables in “Defining common variables” on page 53.
docbroker.connectMode	Type of the connection mode. This variable uses the value that you specified for connectMode in common-variables in “Defining common variables” on page 53.
docbroker.liveness.enable	Indicates if the liveness probe is enabled. The default value is true.  “Checking liveness of connection broker” on page 398 contains detailed information.
docbroker.secretsChange	Indicates if any changes to variables are made in dcm-server.cs-secrets. This variable uses the value that you specified for secrets_Change in common-variables in “Defining common variables” on page 53. See “Defining common variables” on page 53 for the detailed description.
persistentVolume.storageClass	Storage class of the PV. This variable uses the value that you specified for rwoStorage in common-variables in “Defining common variables” on page 53.

Name	Description
persistentVolume.logVctStorageClass	Storage class of the VCT log. This variable uses the value that you specified for rwoStorage in common-variables in “Defining common variables” on page 53.
persistentVolume.pvcStorageClass	Indicates if a different storage class is used for PVC and VCT. The default value is null.
persistentVolume.pvcStorageSize	Size of the PVC. The default value is 1Gi. For Red Hat OpenShift, you must set the value to 256Gi.
persistentVolume.existVolumePv	Details of existing volume, if any. Read the Readme.txt file for more information.
persistentVolume.awsEFS	Indicates if Amazon Elastic File System (EFS) is enabled. The default value is false.
persistentVolume.awsEFSCSIDriver	External storage provisioner (Amazon Elastic File System Container Storage Interface (CSI) Driver). The default value is efs.csi.aws.com.
persistentVolume.awsEFSCSIHandle	Volume ID of the EFS volume.
graylog.enable	Indicates if Graylog is enabled for use. The default value is true. This variable uses the value that you specified for grayLogEnable in common-variables in “Defining common variables” on page 53.
graylog.server	Server details based on your Graylog server configuration. This variable uses the value that you specified for graylogServer in common-variables in “Defining common variables” on page 53.
graylog.port	Port reserved based on your Graylog server configuration. This variable uses the value that you specified for graylogPort in common-variables in “Defining common variables” on page 53.
ExtDocbroker.enable	Indicates if the external connection broker is enabled for use. The default value is false. This variable uses the value that you specified for externalAccessEnabled in common-variables in “Defining common variables” on page 53.
ExtDocbroker.extNativeNodePort	Make sure that there is no value specified for this variable.
ExtDocbroker.extSSLNode	Make sure that there is no value specified for this variable.

Name	Description
ExtDocbroker.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.
ExtDocbroker.useELB	Indicates if externalizing Documentum Server deployment in Amazon Web Services is enabled. The default value is false.
ExtDocbroker.enableLiveness	Indicates if liveness is enabled for the external connection broker.
ExtDocbroker.useLBAnnotations	Indicates if load balancer annotations is enabled for load balancer services used for externalizing Documentum Server. The default value is false.
ExtDocbroker.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. This variable uses the value that you specified for useCertificate in common-variables in <a href="#">"Defining common variables" on page 53</a> .
certificate.aekname	Name of AEK. The default value is aek_name.
certificate.aeklocation	Path where AEK is available. This variable uses the value that you specified for aekLocation in common-variables in <a href="#">"Defining common variables" on page 53</a> .
newrelic	<a href="#">"Integrating New Relic with connection broker, Documentum Server, and Java Method Server" on page 383</a> contains detailed information.

After providing the appropriate values for the variables in docbroker, perform the next task in [step 8.d](#) in ["Deploying Documentum Server" on page 58](#).

**Table 3-5: content-server**

Name	Description
enabled	Indicates if the Documentum Server pod deployment is enabled. The default value is true.
serviceName	Unique name.
serviceAccount.createServiceAccount	Indicates to create a service account. The default value is false.  Note: If the value of <code>dctm-server.docbroker.createServiceAccount</code> is set to true, then the value of <code>dctm-server.content-server.createServiceAccount</code> must be set to false as the service account is already created.
serviceAccount.serviceAccountName	Name of the service account. This variable uses the value that you specified for <code>documentumServiceAccount</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
secret.name	Name of the secret configuration file. This variable uses the value that you specified for <code>csSecrets</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
docbroker.serviceName	Service name of connection broker. This variable uses the value that you specified for <code>dbrServiceName</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
docbroker.clusterSpace	Name of the cluster space. This variable uses the value that you specified for <code>env</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
docbase.name	Name of the repository. This variable uses the value that you specified for <code>docbase</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
docbase.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).

Name	Description
docbase.owner	<p>Name of the repository owner. This variable uses the value that you specified for docbase in common-variables in “Defining common variables” on page 53.</p> <p>For information about deploying Documentum Server with Microsoft Azure PaaS, see “Deploying Documentum Server with Azure PaaS” on page 304.</p>
docbase.existing	Indicates to use an existing repository. The default value is false.
docbase.index	<p>Name of the repository index if you want to use the existing repository.</p> <p>The format is DM_&lt;docbase name&gt;_docbase.</p>
contentserver.connectMode	Type of the connection mode. This variable uses the value that you specified for connectMode in common-variables in “Defining common variables” on page 53.
contentserver.externalUserEnable	<p>Indicates if the tracking of external user transactions is enabled. The default value is false.</p> <p>Set the value to true if you want to track external user transactions. In addition, make sure that the value is true during the upgrade process.</p> <p>“Upgrading Documentum Server to 23.4” on page 437 contains detailed information to upgrade Documentum Server.</p> <p>For more information about tracking external user transactions, see <i>OpenText Documentum Server Administration and Configuration Guide</i>.</p>
contentserver.aek.name	Name of AEK. The default value is aek_name.
contentserver.aek.location	<p>Path where AEK is available. This variable uses the value that you specified for aekLocation in common-variables in “Defining common variables” on page 53.</p> <p>Valid value is Local for password-based keys.</p>

Name	Description
contentserver.aek.shareKey	New AEK is created.  If you set the value to <code>true</code> in your first statefulset deployment, new PVC is created with name in the <code>&lt;sname&gt;-sharedkey-pvc</code> format and the AEK is copied after the deployment of the Documentum Server pod.  To reuse the existing AEK, set the value to <code>false</code> during the second and further statefulset deployments and then provide the appropriate value for the <code>shareKeyPVCName</code> variable.  For more information, see <a href="#">“Supporting multiple repositories in one namespace” on page 418</a> .
contentserver.jmsProtocol	Protocol to connect to JMS.
contentserver.jmsVersion	Supported Tomcat version.

Name	Description
contentserver.localeName	<p>Locale of Documentum Server. The default value is en.</p> <p>Supported locales are:</p> <ul style="list-style-type: none"> <li>• en: English</li> <li>• ar: Arabic</li> <li>• de: German</li> <li>• es: Spanish</li> <li>• fr: French</li> <li>• it: Italian</li> <li>• ja: Japanese</li> <li>• ko: Korean</li> <li>• nl: Dutch</li> <li>• pt: Brazilian Portuguese</li> <li>• ru: Russian</li> <li>• sv: Swedish</li> <li>• zh: Simplified Chinese</li> <li>• iw: Hebrew</li> </ul> <p> <b>Note:</b> From the 22.1 release, you can configure multiple languages for Documentum Server. To select multiple languages, use language codes separated by comma.</p> <p>For example:</p> <pre>localeName = en, de, es</pre> <p><i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information about populating and publishing the data dictionary.</p>
contentserver.fulltextEngineSSVCEnable	Indicates to change from Documentum Search query plug-in to Solr query plug-in in the Documentum Server setup. The default value is false.
contentserver.jmsMemSetting	Memory setting of JMS.
contentserver.secretsChange	<p>Indicates if any changes to variables are made in <code>dctm-server.cs-secrets</code>. This variable uses the value that you specified for <code>secrets_Change</code> in <code>common-variables</code> in “Defining common variables” on page 53.</p> <p>See “Defining common variables” on page 53 for the detailed description.</p>

Name	Description
contentserver.readiness.considerBPMForReadiness	Indicates to consider BPM for the readiness probe. The default value is <code>false</code> .
contentserver.readiness.periodSeconds	Frequency to perform the probe. The default value is 60 seconds.
contentserver.liveness.enable	Indicates if the liveness probe is enabled. The default value is <code>true</code> .  “ <a href="#">Checking liveness of Documentum Server on page 399</a> contains detailed information.
contentserver.liveness.initialDelaySeconds	Time in seconds after the container or pod has started before the liveness probe is initiated.
contentserver.liveness.considerJMSForLiveness	Indicates to consider JMS for the liveness probe. The default value is <code>true</code> .
contentserver.liveness.considerBPMForLiveness	Indicates to consider BPM for the liveness probe. The default value is <code>false</code> .
globalRepository.globalRepositoryName	Name of the global repository which has to be used as the global repository for the new repository. If you do not set any value, then the new repository is configured as the global repository.
globalRepository.globalRepositoryDocbrokerHost	Global repository connection broker host information. If you do not set any value, then the value of the new connection broker host is used.
globalRepository.globalRepositoryDocbrokerPort	Global repository connection broker port information. If you do not set any value, then the value of the new connection broker port is used.
ExtCS.enable	Indicates if the access to external Documentum Server is enabled. The default value is <code>false</code> .
ExtCS.tcp_route	IP address of the node. <ul style="list-style-type: none"> <li>Kubernetes: Use the <code>kubectl get node -o wide</code> command to obtain the IP of the node. The default value is <code>10.0.0.0</code>.</li> <li>Red Hat OpenShift: Use the value specified in “<a href="#">Deploying Documentum Server</a>” on page 269.</li> </ul>
ExtCS.nativeExtPort	Port reserved for externalizing the Documentum Server pod. <ul style="list-style-type: none"> <li>Kubernetes: Specify the reserved port value according to your requirement.</li> <li>Red Hat OpenShift: Set the value to 80.</li> </ul>

Name	Description
ExtCS.sslExtPort	Port reserved for externalizing the Documentum Server pod. <ul style="list-style-type: none"> <li>Kubernetes: Specify the reserved port value according to your requirement.</li> <li>Red Hat OpenShift: Set the value to 81.</li> </ul>
ExtCS.extDbrPort	Port reserved for the external connection broker. The default value is 1491.
ExtCS.createExtService	Indicates if creating external service for Documentum Server is enabled. The default value is true. <p> <b>Note:</b> To externalize Documentum Server in Red Hat OpenShift, set the value to false.</p>
ExtCS.useELB	Indicates if externalizing Documentum Server deployment in Amazon Web Services is enabled. The default value is false.
ExtCS.useLBAnnotations	Indicates if load balancer annotations is enabled for load balancer services used for externalizing Documentum Server. The default value is false.
ExtCS.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:  <code>useLBAnnotations: true LBAnnotations:   networking.gke.io/load-balancer-type:   "Internal"</code>
database.host	Information you provided while configuring the database. See “ <a href="#">Prerequisites</a> ” on page 56. <p> <b>Note:</b> For the support of PostgreSQL installed on a VM, the value is the IP address or host name of PostgreSQL on a VM.</p>
database.databaseServiceName	Information you provided while configuring the database. See “ <a href="#">Prerequisites</a> ” on page 56. <p> <b>Note:</b> For the support of PostgreSQL installed on a VM, the default value is MyPostgres. You can change to any other user-defined name but only at the time of new deployment.</p>

Name	Description
database.port	Information you provided while configuring the database. See “ <a href="#">Prerequisites</a> ” on page 56.  Note: For the support of PostgreSQL installed on a VM, the value is the port reserved for connecting to PostgreSQL on a VM.
database.sslEnabled	Indicates if SSL is enabled. The default value is false.
database.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 “ <a href="#">Deploying Documentum Server with Azure PaaS</a> ” on page 304.
database.docbaseOwnerPasswordChange	Indicates if repository owner password is changed in the database. The default value is false.
thumbnailServer.configure	Indicates if Thumbnail Server pod configuration is enabled. The default value is true.  Do not change this value.
thumbnailServer.userMemArgs	User memory arguments. The default value is null.
thumbnailServer.tnsProtocol	Protocol to connect to Thumbnail Server. Only HTTP protocol is supported.  Do not change this value.
otds.configureOTDS	Configuration of OTDS. This variable uses the value that you specified for <code>dctmotds</code> in <code>common-variables</code> in “ <a href="#">Defining common variables</a> ” on page 53.  Do not change this value.
otds.configNameOption	Type of configuration. Valid values are HA and HOSTNAME.
otds.otdsAPISvc	Location where OTDS APIs are deployed.

Name	Description
otds.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 <code>otds_rest_credential_url</code> in <code>otdsauth.properties</code> located at <code>\$DM_JMS_HOME/webapps/OTDSAAuthentication/WEB-INF/classes</code> .
otds.clientCapability	Expertise level of the user. The default value is 0.
otds.userPrivileges	Privileges assigned to the user. The default value is 0.
otds.userXPrivileges	Extended privileges assigned to the user. The default value is 0.
otds.userRenameEnabled	User name of <code>dm_user</code> is updated during OTDS synchronization. The default value is F.
otds.userRenameUnlockLockedObject	Indicates to unlock all the objects checked out by the user to be renamed. The default value is T.  This is applicable only if <code>userRenameEnabled</code> is set to T.
otds.groupRenameEnabled	Indicates if the group name of <code>dm_group</code> can be updated during OTDS synchronization. The default value is F.
otds.updateOTDScertonrestart	Indicates to update the OTDS certificate details on a restart. The default value is false.
otds.passauth_use_oauth2_token	Indicates to use oAuth token for password authentication. The default value is false.
otds.client_id	Client ID information.
otds.client_secret	Client secret information.
otds.otds_rest_oauth2_url	OTDS REST oAuth2 URL information.
otds.synced_user_login_name	Synchronized user login name.
otds.auto_cert_refresh	Indicates to automatically refresh certificate information. The default value is true.
otds.cert_jwks_url	JSON Web Key Sets (JWKS) URL information for certificates.

Name	Description
otds.is_hybrid	<p>User partition in user name. The default value is <code>false</code> which means user name has partition.</p> <p> <b>Notes</b></p> <ul style="list-style-type: none"> <li>If you set the value to <code>true</code>, you must set the value of <code>synced_user_login_name</code> to <code>sAMAccountName</code>.</li> <li>If you want to update the value of <code>synced_user_login_name</code> (<code>sAMAccountName</code>) in <code>otdsauth</code> properties, you must set the value of <code>passauth_use_oauth2_token</code> to <code>true</code>.</li> </ul>
custom.scriptExecute	<p>Indicates if custom scripts execution is enabled. The default value is <code>false</code>.</p> <p>Set the value to <code>true</code> 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 <code>script_from_external.sh</code> in <code> \${DM_DOCKER_HOME}/custom_script/</code> and make sure to start the composite image. During the installation process, <code>script_from_external.sh</code> is called and executed.</p>
custom.scriptinPVC	Custom scripts in PVC.
custom.enableBPMPVC	Indicates if BPM PVC is enabled.
custom.scriptPVCname	Name of the custom script in PVC.
custom.PVCSubPath	Subpath of PVC.
custom.useInitContainers	Indicates to use one or more init containers, which are run before the application containers are started. The default value is <code>false</code> .
graylog.enable	Indicates if Graylog is enabled for use. This variable uses the value that you specified for <code>grayLogEnable</code> in <code>common-variables</code> in “Defining common variables” on page 53.
graylog.server	Server details based on your Graylog server configuration. This variable uses the value that you specified for <code>graylogServer</code> in <code>common-variables</code> in “Defining common variables” on page 53.

Name	Description
graylog.port	Port reserved based on your Graylog server configuration. This variable uses the value that you specified for <code>graylogPort</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
<code>persistentVolume.storageClass</code>	Storage class for the PV. This variable uses the value that you specified for <code>rwmStorage</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
<code>persistentVolume.awsEFS</code>	Indicates if Amazon Elastic File System (EFS) is enabled. The default value is <code>false</code> .
<code>persistentVolume.awsEFSCSIDriver</code>	External storage provisioner (Amazon Elastic File System Container Storage Interface (CSI) Driver). The default value is <code>efs.csi.aws.com</code> .
<code>persistentVolume.awsEFSCSIHandle</code>	Volume ID of the EFS volume.
<code>persistentVolume.awsEFSCSIClaimPolicy</code>	Reclaim policy of the EFS static provisioning persistent volumes. The default value is <code>Retain</code> .
<code>persistentVolume.shareKeyPVCName</code>	Existing PVC name to reuse AEK from any other statefulset deployment in a multi-repository environment. The default value is empty (" ") which means each statefulset deployment creates its own AEK.  For more information, see <a href="#">“Supporting multiple repositories in one namespace” on page 418</a> .
<code>persistentVolume.existVolumePv</code>	Details of existing volume, if any. Read the <code>Readme.txt</code> file for more information.
<code>volumeClaimTemplate.storageClass</code>	Storage class of the VCT. This variable uses the value that you specified for <code>rwoStorage</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
<code>volumeClaimTemplate.logVctStorageClass</code>	Storage class of the VCT log. This variable uses the value that you specified for <code>rwoStorage</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
<code>certificate.use_certificate</code>	Indicates to use certificate-based communication. This variable uses the value that you specified for <code>useCertificate</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> . The default value is <code>false</code> .

Name	Description
certificate.dbrserviceName	Service name of connection broker. This variable uses the value that you specified for dbrServiceName in common-variables in “Defining common variables” on page 53.
certificate.customUpgrade.enable	Indicates to use certificate-based communication in an upgraded environment.  “Enabling certificate-based communication in upgraded 23.4 environment” on page 442 contains detailed information.
certificate.customUpgrade.nativeFirst	Native mode of connection. The default value is true. Do not change this value.
dbrpersistentVolume.dbrdataPVCName	Name of the PVC.
dbrpersistentVolume.storageClass	Storage class for the PVC. This variable uses the value that you specified for rwoStorage in common-variables in “Defining common variables” on page 53.
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.
email.smtpServer	SMTP host name or IP address accessible to the Documentum Server pod.
email.smtpPort	SMTP port accessible to the Documentum Server pod.
email.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 dctm-server.cs-secrets.
email.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 dctm-server.cs-secrets.
email.emailAddress	Email address of the installation owner.
email.notification	Indicates if the email notification is enabled. The default value is true.
s3Store.enable	Indicates if S3 store is enabled. The default value is false.  To enable the S3 store, set the value to true.

Name	Description
s3Store.default	<p>Indicates if S3 store is the default object store. The default value is <code>false</code>.</p> <p>To set S3 store as the default object store, set the value to <code>true</code>.</p> <p> <b>Notes</b></p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• If you set S3 store as the default object store, compression and deduplication are not supported.</li> </ul>
s3Store.name	<p>S3 store-related variable.</p> <p><i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.</p>
s3Store.proxyHost	<p>S3 store-related variable.</p> <p> <b>Note:</b> You must set the value for <code>proxyHost</code> if the Documentum Server host is behind the proxy server and S3 object store in the public cloud. The default value is <code>noproxy</code>.</p> <p><i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.</p>
s3Store.proxyPort	<p>S3 store-related variable.</p> <p> <b>Note:</b> You must set the value for <code>proxyPort</code> if the Documentum Server host is behind the proxy server and S3 object store in the public cloud. The default value is <code>noproxy</code>.</p> <p><i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.</p>
s3Store.proxyProtocol	<p>S3 store-related variable.</p> <p><i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.</p>
s3Store.noProxy	S3 store-related variable.

Name	Description
s3Store.isworm	S3 store-related variable.  <i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.
s3Store.vendor	S3 store-related variable.  <i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.
s3Store.region	S3 store-related variable.  <i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.
s3Store.enable_md5	S3 store-related variable.  <i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.
s3Store.enable_v4signing	S3 store-related variable.  <i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.
s3Store.updateExistingStore	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.
s3Store.storeListUpdate	List of existing public S3-compatible stores separated by comma.  For example, store1, store2.
s3Store.proxyHostUpdate	Value of the proxy host.
s3Store.proxyPortUpdate	Value of the proxy port.
s3Store.multiPartEnable	S3 store-related variable.  <i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.

Name	Description
s3Store.numThreads	S3 store-related variable.
gcpStore.enable	Indicates if Google Cloud store is enabled. The default value is <code>false</code> .   <b>Note:</b> If you want to enable Google Cloud store, you must set the value of both <code>gcpStore.enable</code> and <code>restStore.enable</code> to <code>true</code> .
restStore.enable	REST store-related variable.   <b>Notes</b> <ul style="list-style-type: none"> <li>Only HTTP is supported. Do not change this value.</li> <li>Creation of both the Azure Blob and Google Cloud stores at the same time is not possible in a cloud platform.</li> </ul> <p><i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.</p>
restStore.default	REST store-related variable.  <i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.
restStore.name	REST store-related variable.  <i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.
restStore.restStoreType	REST store-related variable.  <i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.
restStore.proxyHost	REST store-related variable.  <i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.
restStore.proxyPort	REST store-related variable.  <i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.

Name	Description
restStore.proxyProtocol	REST store-related variable.  <i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.
restStore.noProxy	REST store-related variable.  <i>OpenText Documentum Server Administration and Configuration Guide</i> contains detailed information.
newrelic	<a href="#">“Integrating New Relic with connection broker, Documentum Server, and Java Method Server” on page 383</a> contains detailed information.
fluentd_service.enable	Indicates if Fluentd is enabled. The default value is <code>false</code> . This variable uses the value that you specified for <code>fluentd</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
fluentd_service.TCPPort	Port on which Fluentd is listening for TCP connection. This variable uses the value that you specified for <code>fluentdTcpPort</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
fluentd_service.RESTPort	Port on which Fluentd is listening for REST connection. This variable uses the value that you specified for <code>fluentdRestPort</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
fluentd_service.UDPPort	Port on which Fluentd is listening for UDP connection. This variable uses the value that you specified for <code>fluentdUdpPort</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
fluentd_service.DFCLogLevel	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.
fluentd_service.liveness.enable	Indicates if the liveness probe is enabled. The default value is <code>true</code> .

Name	Description
ingress.host	<p>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: &lt;INGRESS-HOSTNAME&gt;.&lt;CLUSTER-DOMAIN-NAME&gt;. The host and clusterDomainName values must be same as the one specified in <code>dctm-server.dctm-ingress</code> in the single All-In-One <code>documentum/values.yaml</code> Helm chart file. The application URLs must be in the following format: &lt;PROTOCOL&gt;://&lt;INGRESS-HOSTNAME&gt;.&lt;CLUSTER-DOMAIN-NAME&gt;/&lt;APPLICATION PATH&gt;.</p> <p>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 <code>ingress.host</code> in <code>dctm-server.content-server</code>. Then, run the Helm upgrade command to update the ingress host values in the ACS and Thumbnail Server URLs:</p> <p> <b>Note:</b> 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:</p> <ol style="list-style-type: none"> <li>1. Connect to any of the Documentum Server pod. For example: <pre>kubectl exec -it testlacdcs-pg-1 -c testlacdcs-pg bash</pre></li> <li>2. Navigate to the scripts folder using the following command format: <pre>cd /opt/dctm_docker/scripts/</pre></li> <li>3. Run the command in the following command format: <pre>./update_acs_tns_urls_with_ingress_host.sh &lt;INGRESS-HOSTNAME&gt;.&lt;CLUSTER-DOMAIN-NAME&gt;&lt;INGRESS_PROTOCOL&gt;</pre></li> </ol>

Name	Description
	If the <INGRESS_PROTOCOL> value is not provided, then by default, the value used is http.
ingress.protocol	Type of the communication mode. This variable uses the value that you specified for ingressProtocol in common-variables in <a href="#">“Defining common variables” on page 53</a> .
ingress.disableUpdateAcsUrl	Indicates to disable the ACS URL update.
logrotate.enable	Indicates if the usage of the logrotate tool is enabled. This variable uses the value that you specified for cslogrotate in common-variables in <a href="#">“Defining common variables” on page 53</a> .
logrotate.interval	Frequency of the log rotation in minutes. The default value is 1440.
logcleanup.enable	Indicates if the log cleanup feature is enabled. The default value is true.
logcleanup.interval	Frequency to remove all the files present in a specific folder. The default value is 1440 minutes. <a href="#">“Using logcleanup in Documentum Server” on page 422</a> contains detailed information.
logcleanup.sessionlogcleanupinterval	Interval to remove the files from the specified folders. The default value is 10080 minutes. <a href="#">“Using logcleanup in Documentum Server” on page 422</a>
logging.cs.configMap_name	Information provided for this variable must be same as specified in dctm-server.cs-logging-configMap.
logging.jms.acs_configMap_name	Information provided for this variable must be same as specified in dctm-server.cs-logging-configMap.
logging.jms.serverApps_configMap_name	Information provided for this variable must be same as specified in dctm-server.cs-logging-configMap.
logging.jms.dmotdsrest_configMap_name	Information provided for this variable must be same as specified in dctm-server.cs-logging-configMap.
logging.jms.oauth_configMap_name	Information provided for this variable must be same as specified in dctm-server.cs-logging-configMap.
logging.jms.otdsauth_configMap_name	Information provided for this variable must be same as specified in dctm-server.cs-logging-configMap.

Name	Description
logging.bpm.enable	Information provided for this variable must be same as specified in <code>dctm-server.cs-logging-configMap</code> .
logging.bpm.bpm_configMap_name	Information provided for this variable must be same as specified in <code>dctm-server.cs-logging-configMap</code> .
logging.fluentdConf.enable	Indicates if Fluentd is enabled. This variable uses the value that you specified for fluentd in common-variables in “Defining common variables” on page 53.
logging.fluentdConf.fluentd_configMap_name	ConfigMap name of Fluentd. The default value is <code>fluentd-configmap</code> .
logging.logrotate_configMap_name	Information provided for this variable must be same as specified in <code>dctm-server.cs-logging-configMap</code> .
logging.d2.logback.enable	Indicates if the D2 logback is enabled. The default value is true.
logging.d2.logback.d2_configMap_name	Information provided for this variable must be same as specified in <code>dctm-server.cs-logging-configMap</code> .
extraEnv	Additional environment variables. You can use this section and its variables based on your requirement.



#### To configure the log levels of the applications that are not part of the default JMS:

- Provide the appropriate values for the `extraVolumes` and `extraVolumeMounts` variables to map the ConfigMap of the applications with the volume.
- Remove the comment tag in the `documentum/charts/dctm-server/charts/content-server/values.yaml` file 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 `dctm-server.content-server`, perform the next task in [step 8.e in “Deploying Documentum Server” on page 58](#).

**Table 3-6: cs-logging-configMap**

Name	Description
<code>enabled</code>	Indicates if the ConfigMap deployment is enabled. The default value is <code>true</code> .
<code>serviceName</code>	Name of the service.
<code>configMap.cs_configMap_name</code>	ConfigMap for both the Documentum Server and Documentum Foundation Classes pods. The default value is <code>cs-logging-configmap</code> .
<code>configMap.acs_configMap_name</code>	ACS application ConfigMap of Java Method Server. The default value is <code>acs-logging-configmap</code> .
<code>configMap.serverApps_configMap_name</code>	ServerApps application ConfigMap of Java Method Server. The default value is <code>serverapps-logging-configmap</code> .
<code>configMap.bpm_configMap_name</code>	BPM application ConfigMap of Java Method Server. The default value is <code>bpm-logging-configmap</code> .
<code>configMap.otdsauth_configMap_name</code>	OTDS authentication ConfigMap of Java Method Server. The default value is <code>otdsauth-logging-configmap</code> .
<code>configMap.dmotdsrest_configMap_name</code>	OTDS REST ConfigMap of Java Method Server. The default value is <code>dmotdsrest-logging-configmap</code> .
<code>configMap.oauth_configMap_name</code>	OAuth ConfigMap of Java Method Server. The default value is <code>oauth-logging-configmap</code> .
<code>configMap.fluentd_configMap_name</code>	ConfigMap name of Fluentd. The default value is <code>fluentd-configmap</code> .
<code>configMap.logrotate_configMap_name</code>	ConfigMap name of logrotate. The default value is <code>logrotate-configmap</code> .
<code>configMap.d2_configMap_name</code>	ConfigMap name of logrotate. The default value is <code>d2-logback-configmap</code> .
<code>loggingConfiguration.containerName</code>	Name of the Documentum Server container. The name can be either the name of complete deployment or specific containers.

Name	Description
dfcEventHubTracing.enable	Indicates if Documentum Foundation Classes RPC tracing log is enabled. This variable uses the value that you specified for dfcRPCTracing in common-variables in “Defining common variables” on page 53.
dfcEventHubTracing.level	Log level for the Documentum Foundation Classes events. This variable uses the value that you specified for eventLogLevel in common-variables in “Defining common variables” on page 53.
dfcTraceing.enable	Indicates if the Documentum Foundation Classes tracing is enabled. The default value is false.
docbaseTrace.enable	Indicates if the Documentum Server tracing is enabled. The default value is false.
docbaseTrace.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.
serverApps.log4j.rootLogLevel	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.  Log level for the log4j2.properties file. The default value is WARN.
serverApps.log4j.layout_type	Layout type for the log4j2.properties. The default value is PatternLayout.
serverApps.log4j.layout_pattern	Layout pattern for the log4j2.properties file.  The format is "%d{ABSOLUTE} %5p [%t] %c - %m%n".
serverApps.log4j.monitorInterval	Monitor interval for the log4j2.properties file. The default value is 30 seconds.

Name	Description
otdsauth.log4j.rootLogLevel	<p>Log level for the OTDS authentication. The valid values are DEBUG, INFO, WARN, ERROR, and FATAL.</p> <p>You can configure the required log4j log levels for the JMS deployed applications.</p> <p>Log level for the <code>log4j2.properties</code> file. The default value is INFO.</p>
dmotdsrest.log4j.rootLogLevel	<p>Log level for the OTDS REST application. The valid values are DEBUG, INFO, WARN, ERROR, and FATAL.</p> <p>You can configure the required log4j log levels for the JMS deployed applications.</p> <p>Log level for the <code>log4j2.properties</code> file. The default value is INFO.</p>
dmotdsrest.log4j.dmotdsrestLogLevel	<p>Log levels for the OTDS REST application. The valid values are DEBUG, INFO, WARN, ERROR, and FATAL.</p> <p>You can configure the required log4j log levels for the JMS deployed applications.</p> <p>Log level for the <code>log4j2.properties</code> file. The default value is INFO.</p>
oauth.log4j.rootLogLevel	<p>Log level for the OAuth application. The valid values are DEBUG, INFO, WARN, ERROR, and FATAL.</p> <p>You can configure the required log4j log levels for the JMS deployed applications.</p> <p>Log level for the <code>log4j2.properties</code> file. The default value is INFO.</p>
acs.log4j.rootLogLevel	<p>Log level for the ACS application. The valid values are DEBUG, INFO, WARN, ERROR, and FATAL.</p> <p>You can configure the required log4j log levels for the JMS deployed applications.</p> <p>Log level for the <code>log4j2.properties</code> file. The default value is WARN.</p>
acs.log4j.acsLogLevel	Log level for the ACS application. The default value is WARN.
acs.log4j.atmosLogLevel	Log level for the ATMOS store. The default value is WARN.

Name	Description
bpm.log4j.rootLogLevel	<p>Log levels for the BPM application. The valid values are DEBUG, INFO, WARN, ERROR, and FATAL.</p> <p>You can configure the required log4j log levels for the JMS deployed applications.</p> <p>Log level for the log4j2.properties file. The default value is WARN.</p>
bpm.log4j.bpmLogLevel	Log level for the BPM application. The default value is WARN.
bpm.log4j.bpsLogLevel	Log level for the BPS application. The default value is WARN.
bpm.log4j.traceLogLevel	Log level for the tracing options. The default value is WARN.
logrotate.enable	Indicates if the usage of the logrotate tool is enabled. This variable uses the value that you specified for cslogrotate in common-variables in <a href="#">"Defining common variables" on page 53</a> .
logrotate.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 based on your requirement with the same format.
d2.logback.enable	Indicates if the D2 logback is enabled.
d2.logback.file	Path of the D2 logback file. The default value is /tmp/D2-JMS.log.
d2.logback.fileNamePattern	<p>File name of the D2 logback file.</p> <p>The format is /tmp/D2-JMS-%d{yyyy-MM-dd}.log.zip.</p>
d2.logback.logLevel	Log level of the logback file. The default value is info.
fluentdConf.enable	Indicates if Fluentd is enabled. The default value is false. This variable uses the value that you specified for fluentd in common-variables in <a href="#">"Defining common variables" on page 53</a> .
fluentdConf.TCPPort	Port on which Fluentd is listening for TCP connection. This variable uses the value that you specified for fluentdTcpPort in common-variables in <a href="#">"Defining common variables" on page 53</a> .

Name	Description
fluentdConf.RESTPort	Port on which Fluentd is listening for REST connection. This variable uses the value that you specified for fluentdRestPort in common-variables in “Defining common variables” on page 53.
fluentdConf.UDPPort	Port on which Fluentd is listening for UDP connection. This variable uses the value that you specified for fluentdUdpPort in common-variables in “Defining common variables” on page 53.
fluentdConf.kafkaTopic	Topic name provided while deploying the Apache Kafka cluster for storing the events. This variable uses the value that you specified for kafka_topic_name in common-variables in “Defining common variables” on page 53.
fluentdConf.kafkaUser	Administrator user name used to communicate among the replicas in the Apache Kafka cluster. This variable uses the value that you specified for kafka_admin_user_name in common-variables in “Defining common variables” on page 53.
fluentdConf.kafkaUsrPasswd	Administrator password used to communicate among the replicas in the Apache Kafka cluster. This variable uses the value that you specified for kafka_admin_user_password in common-variables in “Defining common variables” on page 53.
fluentdConf.compressionMode	Type of the compression mode. The default value is gzip.
fluentdConf.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.
fluentdConf.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.
fluentdConf.kafkaBrokerList	Broker list information of Apache Kafka. This variable uses the value that you specified for kafkaBrokerList in common-variables in “Defining common variables” on page 53.



**Note:** When you change the default value of any of the variables in `dctm-server.cs-logging-configMap` each time after the initial deployment of Documentum Server logging ConfigMap Helm, you must upgrade the Helm deployment to take effect of the changes.

After providing the appropriate values for the variables in `dctm-server.cs-logging-configMap`, perform the next task in [step 8.f](#) in “[Deploying Documentum Server](#)” on page 58.

**Table 3-7: dctm-ingress**

Name	Description
<code>enabled</code>	Indicates if the ingress pod deployment is enabled. The default value is <code>false</code> .
<code>IngressPrefix</code>	Unique ingress name.
<code>ingress.configureHost</code>	Indicates if the ingress host is configured. The default value is <code>false</code> .
<code>ingress.host</code>	Name of the ingress host.
<code>ingress.clusterDomainName</code>	Domain name of the ingress controller in the cluster namespace. This variable uses the value that you specified for <code>ingressDomain</code> in <a href="#">common-variables</a> in “ <a href="#">Defining common variables</a> ” on page 53.
<code>ingress.class</code>	Name of the ingress controller.
<code>ingress.annotations</code>	<p>Details of the annotations and its default values. You can modify the default values based on your requirement in the <code>dctm-server/platforms/&lt;cloud platform&gt;.yaml</code> file.</p> <p>Ingress Resource determines the controller that is utilized to serve traffic. Set the Ingress annotation to select the ingress controller.</p> <p>For example, this is set with an annotation, <code>kubernetes.io/ingress.class</code>, in the metadata section of the Ingress Resource.</p>
<code>jmsService,</code> <code>jmsBase,</code> <code>acsService,</code> and so on	<p>Details of the services and its default values.</p> <p> <b>Note:</b> Make sure that <code>jmsBase</code> is disabled (value is set to <code>false</code>) if you are using Amazon Web Services (AWS) Application Load Balancer (ALB) as the ingress controller.</p>

Name	Description
openshiftTls.enable	<p>Indicates if deployment in the Red Hat OpenShift platform is enabled with the HTTPS configuration. This variable uses the value that you specified for <code>openshiftTls</code> in <code>common-variables</code> in “Defining common variables” on page 53.</p> <ul style="list-style-type: none"> <li>• Kubernetes: Set the value of <code>openshiftTls</code> to <code>false</code>.</li> <li>• Red Hat OpenShift: Set the value of <code>openshiftTls</code> to <code>true</code> only if you want to configure the HTTPS mode for communication. This creates secrets of TLS type with the <code>&lt;secret-name&gt;-tls</code> name. Then, use this secret for the HTTPS configuration.</li> </ul>
tls.enable	Indicates if TLS is enabled. The default value is <code>false</code> .
tls.secretName	Name of the secret configuration file. This variable uses the value that you specified for <code>csSecrets</code> in <code>common-variables</code> in “Defining common variables” on page 53.

After providing the appropriate values for the variables in `dctm-server.dctm-ingress`, perform the next task in step 8.g in “Deploying Documentum Server” on page 58.

**Table 3-8: cs-dfc-properties**

Name	Description
enabled	Indicates if <code>cs-dfc-properties</code> is enabled. The default value is <code>false</code> .
env.domain	Domain name of the environment. This variable uses the value that you specified for <code>env</code> in <code>common-variables</code> in “Defining common variables” on page 53.
serviceName	Service name of connection broker. This variable uses the value that you specified for <code>dbrServiceName</code> in <code>common-variables</code> in “Defining common variables” on page 53.
configMap.namespace	Domain name of the environment.
globalregistry.repository	Repository defined as a global registry. This variable uses the value that you specified for <code>docbase</code> in <code>common-variables</code> in “Defining common variables” on page 53.

Name	Description
globalregistry.username	User name of the global registry. This variable uses the value that you specified for <code>globalRegistryUsername</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .

**Table 3-9: otds**

Name	Description
global.otdsUseReleaseName	<p>Indicates if the release name is used in the names of the objects. The default value is <code>false</code>.</p> <p><b>! Important</b> The value of this variable must not be changed for an upgrade process.</p> <p> <b>Note:</b> Global default values for all <code>global.</code> variables apply to all subcharts when defined as global by a parent chart. These values are overwritten by values provided for all the <code>otdsws.</code> variables.</p>
global.namespace	Name of the namespace used for deployment. This variable uses the value that you specified for <code>namespace</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .
ingress.enabled	Indicates if the Kubernetes ingress is enabled. The default value is <code>false</code> .
ingress.secret	Name of the TLS/SSL Kubernetes secret for the HTTPS connections.
ingress.class	Ingress class used for annotations on different cloud platforms.
ingress.annotations	Cloud platform-specific annotations of the ingress object.
ingress.exposeIndividualEndpoints	Indicates if exposing the list of five endpoints is enabled. The default value is <code>false</code> .

Name	Description
ingress.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 available only if exposeIndividualEndpoints is true. The "/otds-admin/", "/otdstenant/", "/otdsaws/", "/ot-authws/", and "/otds-v2/" endpoints are supported in OTDS. Otherwise, by default, only root '/' is available.
otdsaws.securityContext.fsGroup	Security context configuration for a pod. <ul style="list-style-type: none"> <li>Kubernetes: The value for fsgroup is 1000.</li> <li>Red Hat OpenShift: The value for fsgroup is 1000620000.</li> </ul>
otdsaws.securityContext.runAsGroup	Security context configuration for a pod. <ul style="list-style-type: none"> <li>Kubernetes: The value for runAsGroup is 1000.</li> <li>Red Hat OpenShift: The value for runAsGroup is 1000620000.</li> </ul>
otdsaws.securityContext.runAsUser	Security context configuration for a pod. <ul style="list-style-type: none"> <li>Kubernetes: The value for runAsUser is 1000.</li> <li>Red Hat OpenShift: The value for runAsUser is 0.</li> </ul>
otdsaws.podLabels	Lists the additional pod labels.  For example:  app.kubernetes.io/app_name: otdsaws app.kubernetes.io/app_version: 22.2.0
otdsaws.podAnnotations	Lists the additional pod annotations.
otdsaws.serviceAccountName	Name of the service account where pods are running. The default value is default.
otdsaws.enabled	Indicates if OTDS server deployment as a container is enabled in Kubernetes. This variable uses the value that you specified for ingress.enabled.
otdsaws.statefulSet	Controls if the OTDS server gets deployed as a statefulSet Kubernetes resource. The default value is false. This is helpful when you want OTDS to have a static pod name in some on-premises scenarios.
otdsaws.serviceName	Name of the service for OTDS server.

Name	Description
otdsws.serviceType	Overrides the specification type for the otds service.  If you do not set any value, this variable uses the value that you specified for ingress.enabled. If ingress is enabled, the type is set to ClusterIP. If ingress is disabled, the type is set to LoadBalancer.
otdsws.serviceAnnotations	Define custom annotations that need to be assigned to the OTDS service.  For example: <pre>cloud.google.com/backend-config: '{"ports": {"80": "abcd-otds-backend"}}' cloud.google.com/neg: '{"ingress": true}'</pre>
otdsws.carrierGradeNAT	Configure Tomcat to treat 100.64.0.0/10 addresses as internal for compatibility with environments that use Carrier-grade NAT. The value is always true. This variable is deprecated.
otdsws.customSecretName	Name of an existing secret object. This is useful when OTDS charts work as subcharts and parent chart create a single secret.
otdsws.replicas	Replica count for OTDS server. The default value is 1.
otdsws.port	External port reserved for the OTDS Kubernetes service. The default value is 80.
otdsws.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.
otdsws.timeZone	Time zone of the Linux operating system within the container. The default value is Etc/UTC.
otdsws.allowDuplicateUsers	Indicates if OTDS allows the same user accounts to be created in multiple partitions. This applies to the users synchronized through eDirSync or SCIM.
otdsws.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.

Name	Description
otdsws.additionalJavaOpts	Additional Java parameters for OTDS which should be separated by space. For example, -Dotds.repo.allowduplicateusers=true enables the duplicated user in different partitions.
otdsws.enableBootstrapConfig	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.
otdsws.existingBootstrapConfig	Content of the OTDS bootstrap configuration file. This overrides the configuration files in the chart directory.

Name	Description
otdsws.migration.enabled	<p>Indicates if the migration from OpenDJ must be attempted and defines how the migration must be performed.</p> <p>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.</p> <p>The three different types of migration are:</p> <ul style="list-style-type: none"> <li>From a legacy OTDS deployment (for example, otds:21.3.0 images): Requires the value of <code>usingLegacyImage</code> set to <code>true</code> and that the information for <code>legacyImagePVC</code> is specified. In addition, the minimum memory requirements are increased such as the value of <code>resource.requests.memory</code> is set to <code>2Gi</code> and the value of <code>resource.limits.memory</code> is set to <code>3Gi</code>.</li> <li>From an existing OTDS deployment in the same cluster: Requires the existing <code>servicename</code>, <code>servicePort</code>, and <code>password</code> information of OpenDJ is specified.</li> <li>From an external OpenDJ such as a VM deployment: Requires the <code>opendjUri</code> (for example, <code>ldap://otds.domain.local:1389</code>) and <code>password</code> information is specified.</li> </ul> <p> <b>Notes</b></p> <ul style="list-style-type: none"> <li>If you set the value of <code>otdsws.migration.enabled</code> to <code>true</code>, then the default values are migrated from the previous default configuration.</li> <li>If you set the value of <code>otdsws.migration.usingLegacyImage</code> to <code>true</code>, then the other non-relevant values are ignored. In addition, if <code>opendjUri</code> is specified, then the <code>servicename</code> and <code>servicePort</code> are ignored.</li> </ul>
otdsws.migration.usingLegacyImage	Migration-related variable. The description for <code>otdsws.migration.enabled</code> provides more information.
otdsws.migration.legacyImagePVC	Migration-related variable. The description for <code>otdsws.migration.enabled</code> provides more information.

Name	Description
otdsws.migration.servicename	Migration-related variable. The description for otdsbs.migration.enabled provides more information.
otdsbs.migration.servicePort	Migration-related variable. The description for otdsbs.migration.enabled provides more information.
otdsbs.migration.deploymentName	Deployment name for the migration.
otdsbs.migration.opendjUri	Migration-related variable. The description for otdsbs.migration.enabled provides more information.
otdsbs.migration.password	Migration-related variable. The description for otdsbs.migration.enabled provides more information.
otdsbs.migration.preUpgradeJob.enabled	Indicates if the one-step migration or deployment is enabled. The default value is false.
otdsbs.migration.preUpgradeJob.timeout	Indicates the timeout value for the migrate job to complete in seconds or minutes or hours. The default value is 100h.
otdsbs.otadminPassword	Password of the otadmin@otds.admin user for OTDS. The default value is otds.
otdsbs.enableCustomizedTruststore	Allows the import of custom certificates into the JVM truststore. This variable is used to import self-signed or private CA certificates.  The certificates that you want to import must be placed in the /certificates/ directory of the Helm chart or specified with the --set-file option.
otdsbs.singleCaCert	singleCaCert is used to provide a single certificate file using --set-file. This method is used instead of placing the certificates in the certificates directory of the Helm chart.
otdsbs.otdsdb	Connection information of the database for OTDS.
otdsbs.otdsdb.username	Account for the database connection. This variable uses the value that you specified for otdsUserName in common-variables in <a href="#">“Defining common variables” on page 53</a> .
otdsbs.otdsdb.password	Password for the database account. This variable uses the value that you specified for otdsUserPassword in common-variables in <a href="#">“Defining common variables” on page 53</a> .

Name	Description
otdsws.otdsdb.url	URL to connect to the required database for OTDS. The default value is <code>jdbc:postgresql://postgres.domain.local:5432/otdsdb</code> .
otdsws.otdsdb.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.   <b>Note:</b> 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.
otdsws.otdsdb.sslDBRootCert	Accepts the content of the database server custom root CA certificate by the <code>-file</code> option. The certificate is stored in the <code>otds-certs</code> secret and mounted into pod at the mount point. The database root CA certificate file named as <code>dbRootCA.crt</code> is located at <code>/opt/config/certificates</code> .
otdsws.otdsdb.useDefaultSchema	By default, OTDS uses a database schema named <code>otds</code> . If you set this option to <code>true</code> , the default schema assigned to the database user account is used instead. This is required on a shared Oracle database.

**Table 3-10: dtrinstaller**

Category	Name	Description
enable	<i>enabled</i>	Indicates if Documentum Reports license is enabled.
images	<i>repository</i>	Path of the repository. Specify the value provided for <code>imageRepository</code> in the common-variables category in “ <a href="#">Defining common variables</a> ” on page 53.

Category	Name	Description
installer image	<i>dtrinstaller image</i>	<ul style="list-style-type: none"> <li><i>name</i>: Name of the Documentum Reports installer image.</li> <li><i>tag</i>: Tag as a version-specific number.</li> <li><i>pullPolicy</i>: Policy type to fetch the image. Specify the value provided for <i>pullPolicyType</i> in the common-variables category in “<a href="#">Defining common variables</a>” on page 53.</li> <li><i>pullSecrets</i>: Secret name to fetch the secret configuration file. Specify the value provided for <i>pullSecretName</i> in the common-variables category in “<a href="#">Defining common variables</a>” on page 53.</li> </ul>
custom	<i>createPVC</i>	Indicates to create the PVC.
	<i>scriptPVCname</i>	Name of the custom script in PVC.
	<i>PVCSubPath</i>	Path of PVC.
persistent Volume	<i>pvcAccessModes</i>	Access mode of PVC.
	<i>size</i>	Storage size of PV.
	<i>storageClass</i>	Storage class for the PV.
ingress host	<i>ingressHost</i>	Ingress host of the Documentum Reports base component.

Perform the following additional tasks, as required:

- The default ingress services are defined. By default, the ingress services is disabled. To enable the required ingress service(s), perform the following tasks:
  - Set the value of `enable` of the required ingress service(s) to true.
  - Provide the service name for `serviceName` of the required ingress service(s).
  - Retain the default port value for `servicePort`.
- To enable the HTTPS and TLS support in ingress, perform the following tasks:
  - Set the value of `TLS.enable` to true.
  - Specify the same secret name for `TLS.secretName` provided in `dctm-server.cs-secrets` in the single All-In-One documentum/values.yaml Helm chart file.
  - Specify the appropriate values for `tlscrt` and `tlskey` in the single All-In-One documentum/values.yaml Helm chart file.
- 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 documentum/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 `dctm-server.cs-dfc.properties` and completing the additional tasks as required, perform the tasks from [step 10](#) to [step 12](#) in “[Deploying Documentum Server](#)” on page 58 to complete the deployment and verification of the Documentum Server pod.

### 3.2.3 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.

### 3.2.4 Troubleshooting

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

Symptom	Cause	Fix
When you check the status of the upgrade process, it results in an error.	Unsuccessful upgrade.	<p>Use the <code>describe</code> command to find the cause.</p> <p>For example:</p> <pre>kubectl describe pod &lt;name of the pod&gt;</pre> <p>Or</p> <pre>kubectl describe statefulset &lt;name of the statefulset&gt;</pre> <p>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.</p>
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.	<p>Log in to the primary Documentum Server pod and delete the <code>UpgradeInitiated&lt;version&gt;</code> file located in the <code>/opt/dctm/data</code> folder.</p>
Upgrade process is stuck because of change in the installation owner.	Unsuccessful upgrade.	<p>Go to the <code>/opt/dctm/kube/</code> folder, change the installation owner name that belongs to the previous deployment, and recreate the pod.</p>

### 3.3 Deploying and configuring Independent Java Method Server on private cloud

#### 3.3.1 Prerequisites

1. Perform the steps from [step 1 to step 4](#) in “Prerequisites” on page 56 in “Deploying and configuring Documentum Server on private cloud” on page 56.
2. Deploy the Documentum Server pod as described in “Deploying and configuring Documentum Server on private cloud” on page 56.
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 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	23.4.0 or 23.4

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



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

### 3.3.2 Deploying Independent Java Method Server

1. Extract the Helm charts downloaded from 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:

**Table 3-11: ijms**

Name	Description
serviceName	Unique name for sname.
ijms.replicaCount	Number of replica pods. The default value is 2.  ! <b>Important</b> OpenText recommends two replica pods.
ijms.host	Host of IJMS. The format is <sname>ijms.<ingress domain name>. For example, test1ijms.docu.cfcr-lab.bp-paas.oxlab.net.
ijms.ijmsConfiguringContentServer	Name of a specific IJMS configuring Documentum Server or ALL to configure all Documentum Servers.
ijms.proxyHost	REST store-related variable. You must set the value for proxyHost if the Documentum Server host is behind the proxy server and REST object store in the public cloud. The default value is <code>noproxy</code> .

Name	Description
ijms.proxyPort	REST store-related variable. You must set the value for proxyPort if the Documentum Server host is behind the proxy server and REST object store in the public cloud.
ijms.liveness.enable	Indicates if the liveness probe is enabled. The default value is true. <i>"Checking liveness of Java Method Server" on page 399</i> contains detailed information.
ijms.liveness.livenessScript	Path of the liveness probe script.
ijms.liveness.initialDelaySeconds	Time in seconds after the container or pod has started before the liveness probe is initiated. The default value is 960 seconds.
ijms.liveness.periodSeconds	Frequency to perform the probe. The default value is 120 seconds.
ijms.liveness.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.
ijms.readiness.readinessScript	Path of the readiness probe script.
ijms.readiness.initialDelaySeconds	Time in seconds after the container or pod has started before the readiness probe is initiated. The default value is 180 seconds.
ijms.readiness.periodSeconds	Frequency to perform the probe. The default value is 120 seconds.
ijms.readiness.failureThreshold	Number of times the probe is allowed to fail. The default value is 6. For example, if the value is set to 6, then the readiness of the container is marked as not ready after the probe fails for six times consecutively.
images.repository	Registry host and port information of Docker images.
images.ijms.name	Name of the IJMS image. The default value is dctm-ijms.
images.ijms.tag	Tag as a version-specific number.
images.pullPolicy	Policy type to fetch the image. The default value is IfNotPresent.
images.pullSecrets	Secret name to pull the secret configuration file.

Name	Description
docbase.name	Name of the repository.
secret.name	Name of the secret configuration file. The default value is cs-secret-config.
docbroker.serviceName	Service name of connection broker.
docbroker.port	Port reserved for the connection broker. The default value is 1489.
docbroker.clusterSpace	Name of the cluster space. The format is <name of namespace>.svc.cluster.local. For example, docu.svc.cluster.local
docbroker.docbrokersCount	Number of connection brokers deployed. The default value is 2.
globalRepository.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.
globalRepository.globalRepositoryUser	User of the global repository.
ports.jmsport	Port reserved for Java Method Server. The default value is 9180.
ports.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.
ingress.host	Ingress controller domain name service. The format is <sname><INGRESS-HOSTNAME>.<CLUSTER-DOMAIN-NAME>.
ingress.port	Reserved for future use.
ingress.protocol	Reserved for future use.
persistentVolume.ijmsdataPVCName	PVC name. For example, ijms-data-pvc.
persistentVolume.pvcAccessModes	PVC access mode. The default value is ReadWriteMany. Do not change this value.
persistentVolume.size	PV storage size. The default value is 3Gi.

Name	Description
persistentVolume.storageClass	AWS Storage class (RWX) for the PV.
persistentVolume.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.
persistentVolume.awsEFSCSIDriver	External storage provisioner (Amazon Elastic File System Container Storage Interface (CSI) Driver). The default value is efs.csi.aws.com.
persistentVolume.awsEFSCSIHandle	Volume ID of the EFS volume.
persistentVolume.existVolumePv	Details of existing volume, if any. Provide a unique user-defined name for PV. For example, dctmijmspv.
volumeClaimTemplate.vctName	VCT name. The default value is ijms-vct.
volumeClaimTemplate.vctAccessModes	VCT access mode. The default value is ReadWriteOnce.
volumeClaimTemplate.size	VCT size. The default value is 1Gi.
volumeClaimTemplate.storageClass	VCT storage class. The default value is trident-nfs.
volumeClaimTemplate.logVctAccessModes	VCT log access mode. The default value is ReadWriteOnce.
volumeClaimTemplate.logVctSize	VCT log size. The default value is 2Gi.
volumeClaimTemplate.logVctStorageClass	VCT log storage class. The default value is trident-nfs.
service.type	Service type. The default value is ClusterIP.
service.port	Port reserved for the service. The default value is 80.
custom.useInitContainers	Indicates to use init containers. The default value is false.
extraInitContainers	Additional init containers.
extraEnv	Additional environment variables. You can use this section and its variables based on your requirement.

3. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted>/ijms -f
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --
namespce <name of namespace>
```

For example:

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



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

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

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

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

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

### 3.3.3 Limitations

There are no limitations for this release.

### 3.3.4 Troubleshooting

There are no troubleshooting information for this release.

## 3.4 Deploying and configuring Documentum Administrator on private cloud

### 3.4.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 56 in “Deploying and configuring Documentum Server on private cloud” on page 56 except for those steps related to the PostgreSQL database.

### 3.4.2 Deploying Documentum Administrator

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from step 2 to step 9 in “Deploying Documentum Server” on page 58.
3. Make sure that da.enabled is set to true in the <location where Helm charts are extracted>/documentum-components.yaml file.
4. Make sure that all the required Docker image details are updated in the da section in the <location where Helm charts are extracted>/dockerimages-values.yaml file.

5. **Optional** You can create a configuration file for each environment that you deploy. The values in a configuration file override the values of documentum/values.yaml. You can find a basic template of the config file in documentum/config/configuration.yml. If you are using this configuration file, make sure that you pass this configuration.yml in the Helm deployment commands.
6. Go to the da section in the single All-In-One documentum/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:

**Table 3-12: da**

Name	Description
enabled	Indicates if Documentum Administrator deployment is enabled.
serviceAccount.createServiceaccount	Indicates to create a new service account. The default value is false.  ! <b>Important</b> If you set the value to true, you must provide a unique service account name manually for serviceAccount.serviceAccountName.
serviceAccount.serviceAccountName	Name of the service account. This variable uses the value that you specified for documentumServiceaccount in common-variables in <a href="#">“Defining common variables” on page 53</a> .
persistentVolumeClaim.pvcName	Name of the PVC.  For example, da-pvc.
persistentVolumeClaim.storageClass	Storage class for the PVC. This variable uses the value that you specified for rwoStorage in common-variables in <a href="#">“Defining common variables” on page 53</a> .
persistentVolumeClaim.awsEFSCSIDriver	External storage provisioner (Amazon Elastic File System Container Storage Interface (CSI) Driver). The default value is efs.csi.aws.com.
persistentVolumeClaim.awsEFSCSIHandle	Volume ID of the EFS volume.
service.name	Name of the service.
service.type	Type of the service. This variable uses the value that you specified for webappServiceType in common-variables in <a href="#">“Defining common variables” on page 53</a> .

Name	Description
ingress.enable	Indicates if the ingress service is enabled. This variable controls the creation of ingress service for your deployment. If <code>dctm-server.dctm-ingress</code> is used, then it is recommended to keep it disabled. Valid values are <code>true</code> and <code>false</code> and it is case-sensitive.
cs.useCSDfcConfigMap	Indicates to use the ConfigMap name used in Documentum Server. Valid values are <code>true</code> and <code>false</code> . <ul style="list-style-type: none"> <li>• <code>true</code>: Use DFC properties from Documentum Server logging ConfigMap.</li> <li>• <code>false</code>: Use DFC properties from the <code>dfcProperties</code> environment variable.</li> </ul> The default value is <code>true</code> and it is case-sensitive.
cs.configMapName	ConfigMap name used in Documentum Server. This variable uses the value that you specified for <code>dbrConfigmapName</code> in <a href="#">common-variables in “Defining common variables” on page 53</a> .
cs.csSecretConfigName	Name of the secret configuration file that was created while deploying the Documentum Server pod. This variable uses the value that you specified for <code>csSecrets</code> in <a href="#">common-variables in “Defining common variables” on page 53</a> .
cs.allowTrustedLogin	Indicates to use the <code>dfc.session.allow_trusted_login</code> in the <code>dfc.properties</code> file. The default value is <code>false</code> .
certificate.useCertificate	Indicates to connect to connection broker with SSL certificate. This variable uses the value that you specified for <code>useCertificate</code> in <a href="#">common-variables in “Defining common variables” on page 53</a> .
certificate.dbrServiceName	Service name of connection broker. This variable uses the value that you specified for <code>dbrServiceName</code> in <a href="#">common-variables in “Defining common variables” on page 53</a> .
deployment.name	Indicates the deployment name.
deployment.appName	Application name for Documentum Administrator.

Name	Description
images.graylog.enable	Indicates if Graylog is enabled for use. This variable uses the value that you specified for grayLogEnable in common-variables in “Defining common variables” on page 53. If the value is set to true, the graylog side container is created.
images.graylog.server	Details of the Graylog server. This variable uses the value that you specified for graylogServer in common-variables in “Defining common variables” on page 53.
images.graylog.port	Port reserved for the Graylog server. This variable uses the value that you specified for graylogPort in common-variables in “Defining common variables” on page 53.
containers.da.name	Name of the container. For example, da.
containers.da.docbaseName	Name of the repository. This variable uses the value that you specified for docbase in common-variables in “Defining common variables” on page 53.
containers.da.probing.readinessProbe.initialDelaySeconds	Number of seconds after the Documentum Administrator pod has started before the probe is initiated.
containers.da.probing.readinessProbe.periodSeconds	Frequency to perform the probe.
containers.da.probing.livenessProbe.initialDelaySeconds	“Checking liveness of Documentum Administrator” on page 400 contains detailed information.
containers.da.probing.livenessProbe.periodSeconds	“Checking liveness of Documentum Administrator” on page 400 contains detailed information.
otds.enable	Indicates if OTDS is enabled. This variable uses the value that you specified for dctmots in common-variables in “Defining common variables” on page 53.
otds.url	URL of OTDS.
otds.clientID	Details of client ID.
otds.authentication.repoSelectionPageRequired	Indicates if the repository page must be displayed on login. The default value is true.
otds.authentication.loginTicketTimeout	Life span of the login ticket.

Name	Description
dctmreports.enable	Indicates if Documentum Reports pod deployment is enabled. The default value is <code>false</code> .
dctmreports.URL	URL to connect to Documentum Reports. The format is <code>https://&lt;fully qualified ingress host&gt;/dtr</code> .
wdkAppXmlConfig.tagsnvalues	Syntax for both English and Japanese language pack application.  Documentum Administrator supports two locales. The <code>application.language.supported_locales.locale</code> property is used for the supported locales. The supported locales are <code>en_US</code> for U.S English and <code>ja_JP</code> for Japanese.  By default, the <code>application.language.supported_locales.locale</code> property is set to <code>[en_US]</code> within the square brackets. If you want to add Japanese as a supported locale, add <code>ja_JP</code> separated by a comma within the square brackets.  For example, <code>application.language.supported_locales.locale=[en_US, ja_JP]</code> .  The <code>application.language.default_locale</code> property is used for the default locale. The default locale is <code>en_US</code> . If you want to change the default locale to Japanese, set the value of <code>application.language.default_locale</code> to <code>ja_JP</code> .
logging.rootLoggerLevel	Root log level for Documentum Administrator. The default value is <code>WARN</code> .
logging.consoleThresholdLevel	Console threshold level for Documentum Administrator. The default value is <code>WARN</code> .
logging.filename	Name of the log file.
logging.logFileSize	Maximum size of the log file.
logging.maxLogFiles	Maximum number of the log files.
newrelic	<a href="#">“Integrating New Relic with Documentum Administrator” on page 385</a> contains detailed information.

7. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where
```

```
Helm charts are extracted>/documentum-components.yaml --namespace <name of namespace>
```

where <config> is -extra-large-enhanced, -extra-large-standard, -large-enhanced, -large-standard, -medium-enhanced, -medium-large-enhanced, -medium-large-standard, -medium-standard, -small-enhanced, -small-medium-enhanced, -small-medium-standard, -small-standard, or -test-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 install dcmdeployment /opt/temp/documentum --values /opt/temp/documentum/platforms/fcfc.yaml --values /opt/temp/documentum/dockerimages-values.yaml --values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /opt/temp/documentum/documentum-components.yaml --namespace onedctmns
```

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

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

9. Verify the status of the deployment of the pod using the following command format:

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

### 3.4.3 Limitations

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

### 3.4.4 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.

## 3.5 Deploying and configuring D2 components on private cloud

### 3.5.1 Prerequisites

In this section you will find a description of the basic requirements for a Kubernetes environment, plus detailed information on how to deploy D2 once the Kubernetes environment is set up. Refer to the documentation of your chosen cloud platform for details on how to set up prerequisites in a Kubernetes environment.

1. Select the cloud environment of your choice that features Kubernetes support. For example: Cloud Foundry, Google Cloud Platform, Microsoft Azure, or AWS.
2. Set up a cluster.
3. Set up a namespace.
4. Follow the cloud environment's documentation to ensure the Kubernetes engine is up and running.
5. Install Docker and kubectl cli on your machine and ensure you can use the kubectl command to access the Kubernetes environment. For reference purposes, in a Microsoft Windows environment, see *Install Docker Desktop on Windows* on the *Docker Docs* web site, and *Install kubectl on Windows* on the *Kubernetes* web site.
6. Setup Helm. Documentum 20.4 and later supports Helm v3 which removes the Tiller component. To learn more about Helm, read "Using Helm" on the Helm Documentation website.
7. Set up storage class with `ReadWriteMany` access.
8. Decide whether to install D2 Corepack or Pluspack, then download the D2 Docker image by executing a `docker pull registry.opentext.com/<image_name>:<image_tag>` command.

The following D2 images are available for download:

#### List of D2 images:

Sl. No.	Image	Tag
1	dctm-d2cp-classic	23.4
2	dctm-d2cp-config	23.4
3	dctm-d2cp-installer	23.4
4	dctm-d2cp-rest	23.4
5	dctm-d2cp-smartview	23.4
6	dctm-d2pp-classic	23.4

Sl. No.	Image	Tag
7	dctm-d2pp-config	23.4
8	dctm-d2pp-installer	23.2
9	dctm-d2pp-rest	23.4
10	dctm-d2pp-smartview	23.4
11	dctm-d2-mobile-init-container	23.4
12	dctm-d2cp-ijms	23.4
13	dctm-d2pp-ijms	23.4
14	dctm-tomcat	23.4

**List of additional images:**

Sl. No.	Image	Tag
1	dctm-server	23.4
2	dctm-admin	23.4
3	dctm-xcp-installer	23.4
4	dctm-xcp-apphost	23.4
5	dctm-reports-d2	23.4
6	dctm-reports-base	23.4
7	dctm-reports-installer	23.4
8	dctm-workflow-designer	23.4
9	dctm-content-connect	23.4
10	dctm-content-connect-dbinit	23.4
11	dctm-xplore-indexserver	22.1.4
12	dctm-xplore-indexagent	22.1.4
13	dctm-xplore-cps	22.1.4
14	dctm-dfs	23.2
15	dctm-rest	23.4
16	dctm-records-darinstallation	23.4
17	dctm-records	23.4
18	dctm-rqm	23.4
19	dctm-dcc-consul	23.4
20	dctm-dcc-metadataservice	23.4
21	dctm-dcc-syncagent	23.4
22	dctm-dcc-dbschema	23.4

Sl. No.	Image	Tag
23	dctm-dcc-syncnshare-manual	23.4
24	dcc-dar-installer	23.4
25	dctm-dcc-corenotificationservice	23.4
26	dctm-dcc-mailservice	23.4
27	otds-server	23.3.0
28	otawg	23.4.0
29	otawg-init	23.4.0
30	otawg-pg-init	23.4.0
31	fluentd-4.4.2-1	23.4
32	otiv-init-otds	23.4
33	otiv-amqp	23.4
34	otiv-viewer	23.4
35	otiv-markup	23.4
36	otiv-config	23.4
37	otiv-highlight	23.4
38	otiv-publication	23.4
39	otiv-publisher	23.4
40	otiv-asset	23.4
41	dctm-smartviewm365customjar	23.4
42	dctm-smartviewm365	23.4
43	dctm-cmis	23.4
44	dctm-xcp-bps	23.4
45	dctm-xcp-xda	23.4
46	zookeeper	3.6.3

For information on which image tag to use, see “[Container image tags](#)” on page 120.

For example to download the D2 23.4 corepack client container image, execute:

```
docker pull registry.opentext.com/dctm-d2cp-classic:23.4
```

9. Download the documentum-<version>.tgz Helm chart from OpenText MySupport and unzip the helm package.

### 3.5.2 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 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 `dctm-d2pp-smartview-ol` container image, the following command:

```
docker pull registry.opentext.com/dctm-d2pp-smartview:23.4
```

#### 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 want to download the 23.4.0 version of the `dctm-d2pp-smartview` container image, use the following command:

```
docker pull registry.opentext.com/dctm-d2pp-smartview:23.4.0
```

This command would pull the 23.4.0 version of `dctm-d2pp-smartview`, even if a newer version is available.



#### 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.

### 3.5.3 Deploying D2 Components on private cloud

1. Update documentum/Chart.yaml with the helm chart versions you want.

2. Use the following command to update the subcharts:

```
cd documentum
helm dependency update .
```

3. Modify documentum/dockerimages-values.yaml. Search for all instances of registry.opentext.com and replace with the Docker repository you are using.

 **Note:** Update image pull secrets per the environment in dockerimages-values.yaml.

4. If you do not want to install PE, you must remove the pe init container entry (shown below) from the list in the dockerimage values yaml file:

```
dctm-server:
  content-server:
    extraInitContainers:
      - name: "peinstaller-init"
        image: "registry.opentext.com/dctm-xcp-installer:23.4"
        imagePullPolicy: Always
        command: ['/bin/sh', '-c', 'yes | sudo cp -Rf /pescritps/* /opt/dctm_docker/customscriptpvc/']
        volumeMounts:
          - name: dcs-data-pvc
            mountPath: /opt/dctm_docker/customscriptpvc
            subPath: initcontainercustomscripts/dcs-pg
```

5. Follow the steps below:

- a. From 23.2 onwards, D2 webapps, dctm-rest, and dfs images are decoupled from their base image. Now all product images become init containers to their respective main containers (i.e., app server image) as shown below.

```
d2config:
  images:
    repository: *docker_repo
    name : appserverImageName
    tag : *appserverImageTag
    pullPolicy: *pull_policy_type
    pullSecrets: *pull_secret_name
  extensionImage:
    repository: *docker_repo
    name : dctm-d2pp-config
    tag : 23.4
    pullPolicy: *pull_policy_type
  extraInitContainers: []
```

- b. Update the appserverImageName and appserverImageTag in the dockerimages values yaml file per your requirements.

```
appserverImageTag: &appserverImageTag '23.4'
appserverImageName: appserverImageName dctm-tomcat
```

- c. Make sure createPVC is set to true in extension section for one of the webapps and false for the remaining webapps in documentum/values.yaml.

```
d2config:
  extension:
    createPVC: true
```

```
PVCStorageClass: *rwm_storage_class  
pvcName: d2-extension-pvc
```

 **Note:** By default, `createPVC` is set to true for `d2config`.

6. Modify the `documentum/values.yaml`.

a. Update anchors tags as described:

- By default, `dmadmin` and `password` are the username and password for the install owner. Update the below place holders based on your requirements.

```
installOwnerUsername: &installOwner_username <username>  
installOwnerPassword: &installOwner_password <password>
```

- Default service account is used out of the box. If you want to use any non-default service account, then update the anchor tag `documentumserviceaccount: &documentum_service_account` to the common service account name you want to use for the deployment.

```
documentumserviceaccount: &documentum_service_account <service-account-name>
```

You can create the non-default service account as part of the Helm deployment by setting the parameter `createserviceaccount: true` for any one of the components.

 **Note:** By default, `createserviceaccount` is set to false for all the pods since the default service account is used out of the box. otds and otiv pods use their own service accounts since they have different permissions configured.

- Update the following security tags:

```
sessionAllowTrustedLogin: &allow_trusted_login false  
###--Certificate based SSL Communication section--###  
useCertificate: &use_certificate false  
dbrServiceName: &dbr_service_name dbr
```

- If you want to enable Documentum Server and DocBroker access from outside the cluster, then set the following parameter to true:

```
externalAccessEnabled: &external_access_enable true
```

- Update the `rwo_storage_class` and `rwm_storage_class` to fit your environment. You can replace any instance of the `*rwo_storage_class` and `*rwm_storage_class` anchor references with a different storage class name to fit your environment.

- Search for all instances of `<namespace>` and replace it with `namesapce`.
- Search for all instances (not only the anchor tag) of `d2.cfcr-lab.bp-paas.otxlab.net` and replace with ingress domain. For Google Cloud Platform, GKE will automatically assign an external IP address after the Kubernetes Ingress object is created. You can then map the assigned external ip to a fully qualified domain name. You should replace

anywhere that specifies ingress domain or ingress host to match the fully qualified domain name that you will use to map the assigned external IP.

- Search for all instances (not only the anchor tag) of <docbase\_name> and replace with the docbase name you want.
- For webapp\_service\_type, it should be ClusterIP for CFCR, Azure and AWS. It should be NodePort for GCP.
- Update otdsServiceName:&otds\_db\_service to postgres while using the default postgres service with postgres db/container or <postgres-service-name> if not using the default postgres service.
- Update otdsUserName anchor tag to postgres if using default user with postgres database or <postgres-db-user> if not using the default user.
- Update otdsUserPassword:&otds\_db\_password with the database user's (mentioned in above step) password.

```
otdsServiceName:&otds_db_service postgres
otdsUserName:&otds_db_username postgres
otdsUserPassword:&otds_db_password password
```

- If you want to customize the D2 webapp names, update the following anchor tags with the custom webapp name you want to use for the different webapps:

```
d2classicWebappName: &d2classic_webapp_name D2
d2configWebappName: &d2config_webapp_name D2-Config
d2restWebappName: &d2rest_webapp_name d2-rest
d2smartviewWebappName: &d2smartview_webapp_name D2-Smartview
```



**Note:** Please don't use conflicting names among the custom names. Each custom webapp name should be unique, for example:

```
d2classicWebappName: &d2classic_webapp_name D2Custom
d2configWebappName: &d2config_webapp_name D2-ConfigCustom
d2restWebappName: &d2rest_webapp_name d2-restCustom
d2smartviewWebappName: &d2smartview_webapp_name D2-SmartviewCustom
```

- Update the anchor tag ingressUrl with the complete ingress hostname used for the D2webapps as follows:

```
ingressUrl: &ingressUrl <ingress-hostname>.<domain-name>/
```

For example:

```
ingressUrl: &ingressUrl bhat1dctm-ingress.d2.cfcr-lab.bp-paas.otxlab.net/
```

- Update all other anchors to fit your environment.

- b. By default, PE is installed. If you don't want to install PE, update to the following values:

```
content-server:
  logging:
    bpm:
      enable: false
  cs-logging-configMap:
    bpm:
```

```
log4j:
  enable: false
```

c. Enable or disable OTDS authentication.

- If you want to use OTDS authentication, specify the OTDS admin user's password:

```
otds:
  otdsaws:
    ## otadminPassword is the password of the otadmin@otds.admin user for
    OTDS
    ## Note that this password option will only have an impact when initially
    ## populating the backend DB.
    otadminPassword: otds
    ## otdsdb.* defines the connection information for the DB which is the
    ## backend data storage location for OTDS
```

- If you do not want to use OTDS authentication, disable OTDS by updating the following value to false:

```
otds: &otds_enabled false
```

- Under the otdsdb section, update the URL with the URL used to connect to the database for OTDS in case Postgres DB is used to update the <namespace> place holder. Otherwise, if Oracle DB VM is used, update it with jdbc:oracle:thin:@<host>:

```
1521:<databaseServiceName>
```

```
otdsdb:
  ## url specified the url used to connect to the DB for OTDS
  ## The following is a list of sample values.
  ## jdbc:postgresql://postgres.domain.local:5432/otdsdb
  ## jdbc:sqlserver://ms-sql.domain.local:1433;databaseName=otdsdb
  ## jdbc:oracle:thin:@oracle.domain.local:1521:otdsdb
  ## jdbc:sap://hana.domain.local:30015/?databaseName=otdsdb
  url:jdbc:postgresql://db-pg-0.db-pg.<namespace>.svc.cluster.local:5432/
  posgres
```

d. Under the cs-secrets —> database section, update username and password with the database username and password for the database being used

```
database:
  userName:postgres
  password:password
```

e. In the database sub-section under the content-server section, mention the databaseServiceName of the Postgres VM/container. Update the host with host details of the Postgres DB VM/container. Update the port with 5432 for Postgres DB.

```
content-server:
  database:
    host:db-pg-0.db-pg.<namespace>.svc.cluster.local
    databaseServiceName:MyPostgres
    port: 5432
    docbaseOwnerPasswordChange:false
```

New relic monitoring is disabled by default. To enable newrelic then update the anchor tag &newrelic\_enabled mentioned in the code block below to true. Update the following anchor tags with your environment details:

```
###--Newrelic section--###
newrelic: &newrelic_enabled false
```

```

newrelicProxyHost: &newrelic_proxy_host
newrelicProxyPort: &newrelic_proxy_port
newrelicProxyProtocol: &newrelic_proxy_protocol http
licenseKey: &newrelic_license_key <newrelic_license_key>

```

- f. If you are using S3 store or Google cloud store, set disableUpdateAcsUrl: true for content server in documentum/values.yaml or documentum/config/configuration.yaml.

- g. If mail notification is enabled, add the below parameter in the extraEnv section of content-server as shown below:

```

extraEnv:
  - name: addWebServerPort
    value: "true"

```

- h. For D2-Config, set the following value to true if you are using a pluspack Docker image, false if you are using corepack docker image:

```

d2config:
  pluspack: true

```

- i. To modify the time zone setting, you must modify the *extraEnv* in the values.yaml file. For example:

```

### extra Environment Variables ###
extraEnv:
  - name: TZ
    value: "America/Los_Angeles"

```

- j. To set the Crypto salt for client token encryption and decryption, update cryptoKeySalt under restApiRuntime of the d2smartview section in the documentum/values.yaml value to the desired value.

```

d2smartview:
  restApiRuntime:
    cryptoKeySalt:

```

Example:

```

d2smartview:
  restApiRuntime:
    cryptoKeySalt: a$g7*hjkfs

```

- k. To enable the D2 Mobile, make the following changes.

- If you are using postgres VM as a service, update the postgres server host. If not, update the host with the db pod host.
- Update username and password based on the DB being used.

```

appworks-gateway:
  enabled: true
  database:
    server:
      host: db-pg-0.db-pg.<namespace>.svc.cluster.local

```

- Update the following ingress URLs to fit your environment:

```

otds:
  server:
    url: "http://otdsws:80"
awg:
  externalurl: "http://appworks-gateway.d2.cfcrlab.bppaas.oxlab.net"
ingress:
  hosts:
    - host: appworks-gateway.d2.cfcrlab.bppaas.oxlab.net

```

- l. For ingress related entries, the Ingress Controller is cluster based. You need to make your ingress URL unique among other possible deployments in the cluster. Update the following values to fit your environment (dctm-ingress: --> platform: is available in documentum/platforms/<platform>.yaml file):

```
dctm-ingress:
  ingressPrefix: dctm-ingress
  ingress:
    host: dctm-ingress

  # List of valid values is CFCR, Azure, AWS, GCP
  platform: CFCR
```

- m. To deploy webapps to a node, update the node labels in documentum/values.yaml under the respective webapps section, as follows:

```
nodeSelector:
  <label key>: <label value>
```

For example:

```
nodeSelector:
  disktype: ssd
```

To add labels to a node:

- Run `kubectl get nodes` to get the names of your cluster's nodes.
- Find the relevant node and then run `kubectl label nodes <node-name> <label-key>=<label-value>` to add a label to that node.

For example, if the node name is `kubernetes-foo-node-1.c.a-robinson.internal` and the label is `disktype=ssd`, then add the label by using the following command: `kubectl label nodes kubernetes-foo-node-1.c.a-robinson.internal disktype=ssd`.

- n. To disable dfs update, set `dfs.enable` and `dctm-ingress.dfsService.enable` to false as shown below.

```
dfs:
  enable: false

dctm-ingress:
  dfsService:
    enable: false
```

- o. If you want to pass command line arguments to Tomcat at run time, update the `tomcatJVMArgs` value for all web applications with the required parameters.

For example:

```
d2classic:
  env:
    tomcatJVMArgs: "-Xms4096m -Xmx8192m -Xss4m -XX:+DisableExplicitGC"
```

7. Update `appTitle` value to a desired value under `env` section of desired D2 webapp (D2Classic/D2Config).

```
env:
  # By Default the appTitle is "D2" that gets displayed on the Browser tab.
  # Update to a custom appTitle below when required
  appTitle:
```

8. Update *logfilerotation*, *logfilesize*, *logrotateFrequency*, and *connectionTimeoutInMilliseconds*:

```
tomcat:
  connectionTimeoutInMilliseconds: 60000
  logfilerotation: 7 #Log files are rotated "logfilerotation" times before
  being removed
  logfilesize: 5M #Log files are rotated only if they grow bigger than
  "logfilesize" bytes
  logrotateFrequency: 1d #Log files are rotated every "logrotateFrequency".
  Ex., 10m, 10h, 10d
```

9. By default installation of Swagger documentation for D2 SmartView and D2-REST is disabled. This is achieved by setting `remove_documentation` to true.

If documentation is needed, set `remove_documentation` to false.

```
removeDocumentation: &remove_documentation true
```

10. To enable OTIV:



**Note:** OpenText Intelligent Viewer 23.4 supports PostgreSQL database versions 15.x and below only. Use Azure Disk as the storage class name for Intelligent Viewer when Documentum is deployed in AKS.

- a. Set `otiv.enabled` to true in `.\documentum\documentum-components.yaml`. It is enabled by default.

```
otiv:
  enabled: true
```

- b. If ingress is an https end-point, create a secret with certificate and private key and set the following property with the name of the secret in `.\documentum\values.yaml`.

```
otiv:
  global:
    ingressSSLSecret:
```

- c. Update `trustedSourceOrigins` with the ingress domain URL through which the SmartView service is accessible in `.\documentum\values.yaml`.

```
otiv:
  global:
    trustedSourceOrigins: https://dctm-ingress.d2.cfcr-lab.bp-paas.otlab.net
```

- d. Update host aliases for `d2smartview` and `otiv-publisher` in `.\documentum\values.yaml`. IP Address must be the external IP address of nginx service.



**Note:** This is applicable only if nginx is used in any hyperscalers.

For `d2smartview` and `otiv-publisher`, hostnames of `otiv-publication` and public ingress must be provided.

```
- ip: "127.0.0.1"
  hostnames:
    - "foo.local"
    - "bar.local"
```

11. To automatically configure OTDS, use the following steps. Otherwise, skip to the next step.

- a. Under the **otdsws** section of the documentum/values yaml file, set *enableBootstrapConfig* to true.

For example:

```
otds:  
  otdsws:  
    enableBootstrapConfig: true
```

- b. Update documentum/charts/otds/charts/otdsws/config/config.yaml as follows:

- Update the partition name and description. This creates a non-sync partition in OTDS. For example:

```
partitions:  
  - name: <partition_name>  
    description: <partition_description>
```



**Note:** Creating a sync partition is not supported in a cloud deployment. It is disabled by default.

- Update the users and respective partition IDs. For example:

```
users:  
  - name: <user_name>  
    userPartitionID: <partition_name>  
    values:  
      - name: userPassword  
        values:  
          - <user_password>
```

- Add the client capabilities and privileges as shown below:

```
roles:  
  - name: privilege_none  
    userPartitionID: d2partition  
    description: No privilege  
  - name: privilege_createtype  
    userPartitionID: d2partition  
    description: Create privilege  
  - name: privilege_createcabinet  
    userPartitionID: d2partition  
    description: Create Cabinet privilege  
  - name: privilege_createtypeandcabinet  
    userPartitionID: d2partition  
    description: Create type and cabinet privilege  
  - name: privilege_creategroup  
    userPartitionID: d2partition  
    description: Create group privilege  
  - name: privilege_creategroupandtype  
    userPartitionID: d2partition  
    description: Create group and type privilege  
  - name: privilege_creategroupandcabinet  
    userPartitionID: d2partition  
    description: Create group and cabinet privilege  
  - name: privilege_creategroupcabinetandtype  
    userPartitionID: d2partition  
    description: Create group, cabinet and type privilege  
  - name: privilege_sysadmin  
    userPartitionID: d2partition  
    description: Sysadmin privilege  
  - name: privilege_superuser  
    userPartitionID: d2partition  
    description: Superuser privilege  
  - name: Client_Consumer  
    userPartitionID: d2partition
```

```

description: Superuser privilege
- name: Client_Contributor
userPartitionID: d2partition
description: Superuser privilege
- name: Client_Coordinator
userPartitionID: d2partition
description: Superuser privilege
- name: Client_System_Administrator
userPartitionID: d2partition
description: Superuser privilege

```

- Update the resources section with the details shown below:

```

resources:
- resourceName: <resource_name>
  connectorName: REST (Generic)
  connectorid: rest
  userSynchronizationState: true
  pccreatePermissionAllowed: true
  pcmModifyPermissionAllowed: true
  userAttributeMapping:
  userAttributeMapping:
    - sourceAttr:
        - cn
        destAttr: default_folder
        mappingFormat: "/%s"
    - sourceAttr:
        -
        destAttr: client_capability
        mappingFormat: "2"
    - sourceAttr:
        - oExternalID3
        destAttr: user_login_name
        mappingFormat: "%s"
    - sourceAttr:
        -
        destAttr: user_type
        mappingFormat: "dm_user"
    - sourceAttr:
        - cn
        destAttr: user_name
        mappingFormat: "%s"
    - sourceAttr:
        - mail
        destAttr: user_address
        mappingFormat: "%s"
    - sourceAttr:
        -
        destAttr: create_default_cabinet
        mappingFormat: "F"
    - sourceAttr:
        -
        destAttr: user_rename_enabled
        mappingFormat: "F"
    - sourceAttr:
        - cn
        destAttr: __NAME__
        mappingFormat: "%s"
    - sourceAttr:
        -
        destAttr: user_privileges
        mappingFormat: "0"
    - sourceAttr:
        -
        destAttr: user_rename_unlock_locked_obj
        mappingFormat: "T"
    - sourceAttr:
        -
        destAttr: user_xprivileges
        mappingFormat: "0"

```

```

        - oTObjectGUID
destAttr: user_global_unique_id
mappingFormat: "%s"

groupAttributeMapping:
- sourceAttr:
  - mail
  destAttr: group_address
  mappingFormat: "%s"
- sourceAttr:
  - oTObjectGUID
  destAttr: group_global_unique_id
  mappingFormat: "%s"
- sourceAttr:
  -
  destAttr: group_rename_enabled
  mappingFormat: "F"
- sourceAttr:
  - cn
  destAttr: group_name
  mappingFormat: "%s"
- sourceAttr:
  - cn
  destAttr: __NAME__
  mappingFormat: "%s"
connectionParamInfo:
- name: fBaseUrl
  value: http://<value specified for serviceName in content
server helm chart's values.yaml>-jms-service:<value specified for
ports.jmsport in content server helm chart's values.yaml>/dmotdsrest
- name: fUsername
  value: <docbase_name>\<install_owner_name>
- name: fPassword
  value: <install_owner_password>

```

- Update the access roles section with the existing partition and resource details. For example:

```

accessRoles:
- name: <accessrole_name>
  description: <accessrole_description>
  accessRoleMembers:
    userPartitions:
      - userPartition: <partition_name>
  resources:
    - resourceName: <resource_name>
  attributeList:
    - name: pushAllGroups
      values:
        - "True"

```

- Update the Oauth clients section with the redirect URLs. For example:

```

oauthClients:
- name: <oauth_client_id>
  description: <oauth_client_description>
  redirectURLs:
    - "<Ingress URL>/D2/d2_otds.html"
    - "<Ingress URL>/D2/OTDSLogoutResponse.html"
    - "<Ingress URL>/D2-Smartview/ui"

```

If the webapp names are updated, these URLs must be updated accordingly.

- Ensure the http.negotiate auth handler is disabled. For example:

```

authHandlers:
- _name: http.negotiate

```

```

_id: http.negotiate
_description: Handles "Negotiate" authentication with the browser.
_class:
com.opentext.otds.as.drivers.http.NegotiateHttpAuthenticationHandler
_enabled: false
_priority: 20
_scope:
_properties:
  - _key: com.opentext.otds.as.drivers.http.negotiate.enablemobile
    _value: 'false'
_authPrincipalAttrNames:
  - oExternalID4

```

12. You can create a configuration file for each environment that you deploy. The values in a configuration file override the values of documentum/values.yaml. You can find a basic template of the config file under documentum/config/configuration.yaml.
13. From 22.2 onwards, field validation is provided for few attributes as described below in the D2 helm charts.
  - Datatype of Repository, Image, Tag fields in dockerimages-values.yaml should be Strings for all webapps Classic, Config, Smart View, REST.
  - Datatype of serviceType attribute should be string for all webapps and the only allowed values are ClusterIP, LoadBalancer, NodePort

If any of the above requirements does not match, D2 deployment will throw a schema validation error.

14. If required, update the ingress annotation values in the documentum/platforms/<platform>.yaml file based on your platform's ingress and then deploy the D2 helm chart using the following command:

```
helm install <d2deployment> . --values config/configuration.yaml --values dockerimages-values.yaml --values documentum-resources-values-<config>.yaml --values platforms/<platform>.yaml --values .\documentum-components.yaml
```

Use the following to complete the command:

- <config> can be:
  - *small-standard, small-enhanced, small-medium-standard, small-medium-enhanced*
  - *medium-standard, medium-enhanced, medium-large-standard, medium-large-enhanced*
  - *large-standard, large-enhanced, extra-large-standard, and extra-large-enhanced*
- <platform> can be *cfcr, aws, gcp, azure, openshift*.



**Note:** Use the argument --values config/configuration.yaml only if you use the configuration file present in documentum/config/configuration.yaml

From 23.4 onwards, the different components should be enabled or disabled in a single yaml file `documentum-components.yaml` and should be passed as part of the Helm install command at the end of the Helm install command. The values set in this file will override whatever has been set in `values.yaml` or `configuration.yaml`.

### 3.5.3.1 Troubleshooting

For NGINX, observe the following settings and revise as necessary:

- If you use the NGINX controller for the ingress layer and also use Rest Security Authentication mode with `otds_ticket-otds-token` or `ct-otds_ticket-otds_token` in your `d2smartview-rest-api-runtime-properties.yaml` file, then you must update `enable-underscores-in-headers` to "true" in the NGINX configMap. By default, the header underscore option is disabled.

For more information, see: <https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/configmap/#enable-underscores-in-headers>

## 3.5.4 Setting up OpenText Directory Services (OTDS)

There are two methods to set up OTDS, using `config.yml` and manually. If you set up `config.yml`, see “Automatic setup using `config.yml`” on page 132. To set up manually, see “Manual setup” on page 133.



**Note:** In older k8s clusters (<=1.17), if you are using 23.1.0 OTDS, `otds-admin` portal redirects to the http version by default. Append the forward slash (/) at the end of the URL to redirect to the https version.

### 3.5.4.1 Automatic setup using `config.yml`

To use these instructions, you must have configured `config.yml` to set up OTDS. To do so, see step 11. If you have not configured `config.yml`, see “Manual setup” on page 133.

1. Access the OTDS admin website <*Ingress URL*>/`otds-admin`.

For credentials, use `admin` as the user name, and then the password you specified in the OTDS helm chart's `values.yaml`.

2. Sync members to your D2 Documentum Server repository:
  - a. Click **Resources**.
  - b. Next to your resource, click **Actions**.
  - c. Select **Consolidate**.

### 3.5.4.2 Manual setup

1. Access the OTDS admin website <Ingress URL>/otds-admin with the admin as the username and the value of the password you specified in otds helm chart's values.yaml.
2. Click **Partitions** on the left side, create a non-synchronized user partition. Add some users to this partition. After D2 deployment is complete, you can use the user account you created to log into D2 webapps.
3. Click **Resources** on the left tab, then click **Add**.
  - a. Provide a unique **Resource name** and **Description**.
  - b. On the **Synchronization** tab, select **User and group synchronization**, **Create users and groups** and **Modify users and groups**. For **Synchronization connector**, select **REST (Generic)**.
  - c. For the **Connection Information** tab, the **Base URL** should be `http://<value specified for serviceName in content server helm chart's values.yaml>-jms-service:<value specified for ports.jmssport in content server helm chart's values.yaml>/dmotdsrest`. For example, `http://dcs-pg-jms-service:9080/dmotdsrest`. Username should be `<docbase name>\<install owner name>` and password is `<install owner password>`, then click **Test connection**. The test should succeed.
  - d. For **User Attribute Mappings**, add `client_capability` attribute with Format value of 2. Add `default_folder` attribute with OTDS Attribute value of `cn` and Format value of `/%`
  - e. Take the default for everything else and click **Save**.
4. Click **Access Roles** on the left tab, find **Access to <your newly create resource>**.
  - a. Click **Actions**, select **Include Groups**.
  - b. Click **Actions** again, select **View Access Role Details**.
  - c. Under the **User Partitions** tab, add the Partition you just created.
  - d. Under **Users** tab, remove `otadmin@otds.admin` user if there is one.
  - e. Under **Groups** tab, remove `otdsadmins@otds.admin` group if there is one.
  - f. Click **Save**.

 **Note:** The OTDS partition needs to be added to the app works gateway access role to enable access for the D2 partition users to D2 Mobile.
5. Click **Resources** on the left tab, click **Actions** next to your resource, select **Consolidate** to sync the members down to your D2 Documentum Server repository.
6. Click **OAuth clients** on the left tab, create an OAuth client.
  - a. For **Redirect URLs**, add `<Ingress URL>/D2/d2_otds.html`, `<Ingress URL>/D2/OTDSLogoutResponse.html` and `<Ingress URL>/D2-Smartview/ui` to the **Redirect URLs** list.

- b. Click **Save**.
7. Click **Auth Handlers** on the left tab, select **http.negotiate** on the right side and click **Action**, select **Disable**.
8. Assigning individual users with different privileges & client capability is done through roles in OTDS. Make sure `da_privilege_enabled=T` and `lss_cc_enabled=T` are added in `<jms-home>/webapps/dmotdsrest/WEB-INF/classes/dmotds.properties` in the cs pods.
9. For the user privileges, the following roles are created in OTDS via the auto configuration: `privilege_none`, `privilege_createtype`, `privilege_createcabinet`, `privilege_createtypeandcabinet`, `privilege_creategroup`, `privilege_creategroupandtype`, `privilege_creategroupandcabinet`, `privilege_creategroupcabinetandtype`, `privilege_sysadmin`, `privilege_superuser` and assign the users to any one of these roles according to the required privilege by clicking on **Actions > Edit Application Roles** for the user in the **Users** tab in the partition.
10. For the client capability, the following roles are created in OTDS via the auto configuration: `Client_Consumer`, `Client_Contributor`, `Client_Coordinator`, `Client_System_Administrator`. Assign the users to any one of these roles according to the required client capability by clicking on **Actions > Edit Application Roles** for the user in the **Users** tab in the partition so all capabilities and privileges can be handled.



**Note:** When users are synced through OTDS, it is not recommended to modify it through docbase. Any change should be done through OTDS only.

### 3.5.5 Deploying a custom D2 configuration during deployment

#### 3.5.5.1 Custom D2 configuration

To deploy a custom D2 configuration during deployment, you must create a Docker image with the custom configuration included as a zip or xml file. You can download the busybox image from the hub.

##### To create and deploy a custom D2 configuration docker image:

1. Copy the following to a Docker file:

```
FROM <repository IP address>:5000/busybox:1.35
ARG CUSTOM_FILE_NAME
RUN adduser -D -H dmadmin &&
    mkdir -p /opt/D2-install/custom && \
    chown -R dmadmin:dmadmin /opt/D2-install/custom
COPY --chown=dmadmin:dmadmin $CUSTOM_FILE_NAME /opt/D2-install/custom/
CMD sh
```

2. Modify busybox image (`<repository IP address>:5000/busybox:1.35`) to the downloaded busybox image in the local environment.

3. Run the following command to build the Docker image:

```
docker build -f Dockerfile -t <repository IP address>/D2ConfigCustom:latest --build-arg CUSTOM_FILE_NAME=<d2 customer config xml or zip file> --no-cache .
```

For example:

```
docker build -f Dockerfile -t 10.192.02.01:5000/D2ConfigCustom:latest --build-arg CUSTOM_FILE_NAME=docbase1-2020-07-16-Export-Config.xml --no-cache .
```

4. Once the Docker image is built, add the respective values to the documentum/values.yaml file, under the d2config section:

In a multi-repo deployment, when d2configs need to be imported in multi-repo, update the filename as below:

```
d2config:
  customConfigurations:
    custom: true
    filename:
      docbase1:config1.zip,config2.xml,config3.zip;docbase2:config1.zip,config2.xml,config3.zip
```

If there is a need for multiple config import in the single-repo deployment, update the filename parameter as below with comma-separated format:

```
d2config:
  customConfigurations:
    custom: true
    filename: docbase1:config1.zip,config2.xml,config3.zip
```

 **Note:** If you are only deploying d2config, then update the d2config/values.yaml file.

1. Config import only supports XML/ZIP formats.
2. Monitor the d2 config pod logs for the successful completion of multiple config imports (or for any errors) before accessing D2-Config.
3. Make sure config file names do not have any spaces in them.
4. Make sure to not use repeating config file names across docbases.
5. Add the image name and other values to the documentum/dockerimages-values.yaml file, under the d2config section:

```
1 d2config:
2   extraInitContainers: |
3     - name: init
4       image: <image:tag>
5       imagePullPolicy: Always
6       command: ['/bin/sh', '-c', 'yes | cp -rf /opt/D2-install/custom/* /customdir/']
7       volumeMounts:
8         - name: customconfig
9           mountPath: /customdir
```

 **Note:** If only deploying d2config, then update d2config section in the values.yaml file.

### 3.5.5.2 Custom Dar(s) deployment

To deploy a custom Dar during D2 deployment, you must create a Docker image with a custom Dar file. You can download the busybox image from the hub.



**Note:** If you want to install multiple custom dars in a sequence/order, create a text file with the name darOrder.txt and update the file with the docbase name followed by a colon and then the dar file names in the order they should be deployed in a comma-separated manner. Repeat for the required docbases if using multi-repo configuration.

For example:

```
docbase1:dar1.dar,dar2.dar,dar3.dar  
docbase2:dar4.dar,dar5.dar  
docbase3:dar6.dar,dar7.dar
```

Place the darOrder.txt file in the same location as other custom dar files and create the Docker image.

1. Make sure to use dar file names without a space.
2. Make sure to have the darOrder.txt file in UNIX format before creating the Docker image.
3. Make sure to add an EOL character even in the last line of darOrder.txt.

#### To deploy a custom dar/multiple dars during D2 deployment:

1. Copy the following to a Dockerfile:

```
FROM <repository IP address>:5000/busybox:1.35  
RUN adduser -D -H dmadmin &&\  
    mkdir -p /opt/D2-install/custom && \  
    chown -R dmadmin:dmadmin /opt/D2-install/custom  
  
COPY --chown=dmadmin:dmadmin ./*.dar /opt/D2-install/custom/  
CMD sh
```

2. Modify busybox image (<repository IP address>:5000/busybox:1.28) to the downloaded busybox image in the local environment.
3. Run the following command to build the Docker image:

```
docker build -f Dockerfile -t <repository IP address>/D2CustomDar:latest --no-cache .
```

For example:

```
docker build -f Dockerfile -t 10.192.02.01:5000/D2CustomDar:latest --no-cache .
```

4. Add the image details to the documentum/dockerimages-values.yaml file under the extraInitContainers section of content-server

```
dctm-server:  
  content-server:  
    extraInitContainers:  
      - name: d2customdarinit  
        image: <custom_dar_image_name_with_tag>  
        command: ['/bin/sh', '-c', 'yes | sudo cp -rf /opt/D2-install/custom/* /opt/
```

```
dctm_docker/customscriptspvc/d2' ]
volumeMounts:
- name: dcs-data-pvc
  mountPath: /opt/dctm_docker/customscriptpvc
  subPath: initcontainercustomscripts/dcs-pg
```

### 3.5.5.3 Custom Dar and D2 configuration

You can deploy both a D2 custom configuration and custom dar in a single docker file.

 **Note:** If you want to install multiple custom dars in a sequence/order, create a text file with the name darOrder.txt and provide the names of the custom dars each in a new line with .dar extension. Place the darOrder.txt file in the same location as other custom dar files and create the docker image.

**To create and deploy a custom Dar and D2 configuration with a single docker image:**

1. Copy the following to a Docker file:

```
FROM <repository IP address>:5000/busybox:1.35
ARG CUSTOM_FILE_NAME
ARG CUSTOM_DAR_FILE
RUN adduser -D -H dmadmin &&
    mkdir -p /opt/D2-install/custom && \
    chown -R dmadmin:dmadmin /opt/D2-install/custom
COPY --chown=dmadmin:dmadmin $CUSTOM_FILE_NAME /opt/D2-install/custom/
COPY --chown=dmadmin:dmadmin $CUSTOM_DAR_FILE /opt/D2-install/custom/
CMD sh
```

2. Modify busybox image (<repository IP address>:5000/busybox:1.35) to the downloaded busybox image in the local environment.
3. Run the following command to build the Docker image:

```
docker build -f Dockerfile -t <repository IP address>/D2CustomDar:latest --build-arg CUSTOM_FILE_NAME=<d2 customer config xml or zip file> --build-arg CUSTOM_DAR_FILE=<d2 custom dar file> --no-cache .
```

For example:

```
docker build -f Dockerfile -t 10.192.02.01:5000/D2ConfigCustom:latest --build-arg CUSTOM_FILE_NAME=docbase1-2020-07-16-Export-Config.xml --build-arg CUSTOM_DAR_FILE=1.D2-Base.dar --no-cache .
```

4. Once the docker image is built, add the respective values under the extraInitContainers section of content-server followed by existing init containers present in the documentum/dockerimages-values.yaml file.

```
dctm-server:
  content-server:
    extraInitContainers:
      - name: d2customdarinit
        image: <custom_dar_image_name_with_tag>
        command: ['bin/sh', '-c', 'yes | cp -rf /opt/D2-install/custom/* /opt/dctm_docker/customscriptpvc/d2']
        volumeMounts:
          - name: dcs-data-pvc
            mountPath: /opt/dctm_docker/customscriptpvc
            subPath: initcontainercustomscripts/dcs-pg
```

```

d2config:
  extraInitContainers:
    - name: init
      image: <image:tag>
      imagePullPolicy: Always
      command: ['#!/bin/sh', '-c', 'yes | cp -rf /opt/D2-install/custom/* /customdir/']
      volumeMounts:
        - name: customconfig
          mountPath: /customdir

```

- Add the respective values to the documentum/values.yaml file, under the d2config section:

```

d2config:
  ### Custom Configuration ###
  customConfigurations:
    custom: True
    filename: <filename of the file with the d2 config>

```

## 3.5.6 Deploying customizations

### 3.5.6.1 Customizing D2 Webapp containers using single init container

For customizing Webapps, you must build a Docker image where you can add jar files, custom war files, plug-ins, and the property files. The following is a common format for all D2 Webapps where you can place all the custom files in their respective locations.

For example, all jar/plugin files should be placed in the D2Customizations/<webapp\_name>/D2Custom/WEB-INF/lib location and all JavaScript files should go in D2Customizations/<webapp\_name>/D2Custom/js and so on.

```

D2Customizations/
  D2Classic/
    D2Custom/
      js/           --> All D2Classic javascript files
      css/          --> All D2Classic css files
      wars/         --> External WAR files
      WEB-INF/lib   --> All D2Classic Plugins and Lib files
      WEB-INF/classes --> All D2Classic properties files
      LP-Package/   --> All D2Classic language pack files
      D2-Ext/
        Widget1/    --> External Widget files
  D2Smartview/
    D2Custom/
      wars/          --> External WAR files
      WEB-INF/lib    --> All D2Smartview Plugins and Lib files
      WEB-INF/classes --> All D2Smartview properties files
      LP-Package/   --> All D2Smartview language pack files
  D2Config/
    D2Custom/
      wars/          --> External WAR files
      WEB-INF/lib    --> All D2Config Plugins and Lib files
      LP-Package/   --> All D2Config language pack files
        sampleconfig1.zip --> Custom import zip/xml file1
        sampleconfig2.zip --> Custom import zip/xml file2
  D2Rest/
    D2Custom/
      wars/          --> External WAR files
      WEB-INF/lib    --> All D2Rest Plugins and Lib files
      LP-Package/   --> All D2Rest language pack files

```



**Note:** D2 Smart View does not support css/js/external widget customizations. If new wars are added to existing D2-based PODs, resources must be updated accordingly.

The following is an example of the Docker file you can use to build the image. Download the latest Alpine image from the repository.

```
FROM <repository ip>:<port>/alpine:<image tag>
ARG D2CUSTOM
ENV D2_CUSTOM_PATH /customdir
RUN adduser -D -H dmadmin && \
mkdir -p $D2_CUSTOM_PATH && \
chown -R dmadmin:dmadmin $D2_CUSTOM_PATH
COPY --chown=dmadmin:dmadmin $D2CUSTOM/ $D2_CUSTOM_PATH/
CMD sh
```

1. Copy the above content to a file and name it `Dockerfile`. Modify the alpine image path based on your repo location to download the image to the local environment.
2. Run the following command in the location where `Dockerfile` is created to build the Docker image:

```
docker build -f Dockerfile -t <repository>/<custom_image_name>:<custom_image_tag> --build-arg D2CUSTOM=<D2customizations Dir> --no-cache .
```

For example:

```
docker build -f Dockerfile -t repo.opentext.com/d2webappcustomimage:23.2 --build-arg D2CUSTOM=D2Customizations --no-cache .
```

3. Once the Docker image is generated, make the following changes in the Helm charts when deploying D2:

- In the `dockerimages-values.yaml` file, add or update the `extraInitContainer` sections of the Webapps as follows:

```
extraInitContainers: |
  - name: <init_container_name>
    image: <repository_path>/<init_image_name>:<image_tag>
    imagePullPolicy: Always
    command: ['/bin/sh', '-c', 'yes|cp -rf /customdir/<webapp_name>/* /opt/D2-install/custom']
    volumeMounts:
      - name: customconfig
        mountPath: /opt/D2-install/custom
```

- Update image details with the Docker image created previously and replace `<webapp_name>` (in the command above) with `D2Classic/D2Config/D2Smartview/D2Rest` based on the webapp that you are going to place in the `extraInitContainer` section.

For example, shown below are the init container details for `D2Classic`:

```
extraInitContainers: |
  - name: customwebappinit
    image: repo.opentext.com/d2webappcustomimage:23.2
    imagePullPolicy: Always
    command: ['/bin/sh', '-c', 'yes | cp -rf /customdir/D2Classic/* /opt/D2-install/custom']
    volumeMounts:
```

```
- name: customconfig
  mountPath: /opt/D2-install/custom
```

- In the values yaml file (configuration yaml if used), set custom to true under the customConfigurations section for the respective Webapps:

```
customConfigurations:
  custom: true
```

4. In D2Config webapp:

- If any plug-in .jar files are added to the customization, you must add the corresponding plug-in .jar file names under the d2config plugins section of the documentum/values.yaml or documentum/config/configuration.yaml file as shown below.

```
d2config:
  customConfigurations:
    custom: true
    - Plugin1
    - Plugin2
```

- A properties file with customized attributes can be added directly to the d2config subchart under documentum/charts/d2config/properties before the deployment. Provide the file name in the customConfigurations section of d2config in the documentum/values.yaml or documentum/config/configuration.yaml file as shown below.

```
d2config:
  customConfigurations:
    custom: true
    settings: properties/D2Config.properties
```



**Note:** If both custom plugin jar files and custom D2Config properties are used in the single deployment, custom D2Config properties will override the plugin jar names mentioned in point a above.

- To import multiple custom D2 configuration files, place the custom files in the D2Customizations/D2Config/<custom\_files> location as shown in the folder structure above and update the below values in values.yaml or configuration.yaml.

```
d2config:
  customConfigurations:
    custom: true
    filename: <custom_configuration_file_names_with_extension_comma_separated
e.g., D2_Test_Config1.zip,D2_Test_Config2.xml>
```



**Note:** Custom import only supports XML/ZIP formats.

Monitor the d2 config pod logs for successful completion of multiple config imports (or for any errors) before accessing D2-Config.

- If war file customization is used then uncomment the following section in documentum/values.yaml or documentum/config/configuration.yaml and update serviceName with the custom webapp name.

For example:

```
#extraPaths is used to define additional ingress resources for custom wars.
#For aws path should be as follows path: /<path>*extraPaths:
extraPaths:
  - backend:
      serviceName: <servicename>
      servicePort: 8080
      path: /<path>
```

### 3.5.6.2 Customizing D2 Webapp containers using hook scripts

To customize Webapps using the hook script, we need to create a Docker image with scripts and the necessary files in it. Place the hook script files according to the structure shown below.

```
D2Customizations/
D2Classic/
  hook_scripts(*deploy.sh)  --> hook scripts of classic webapp
  applicable_files          --> Necessary files/dirs/sub-dirs referred in the hook
                                script
D2Smartview/
  hook_scripts(*deploy.sh)  --> hook scripts of d2 smartview and necessary files
  applicable_files          --> Necessary files/dirs/sub-dirs referred in the hook
                                script
D2Config/
  hook_scripts(*deploy.sh)  --> hook scripts of d2 config and necessary files
  applicable_files          --> Necessary files/dirs/sub-dirs referred in the hook
                                script
D2Rest/
  hook_scripts(*deploy.sh)  --> hook scripts of d2 rest and necessary files
  applicable_files          --> Necessary files/dirs/sub-dirs referred in the hook
                                script
```



**Note:** The name of the hook scripts can be preceded by the execution order number, e.g., 10deploy.sh runs first before 20deploy.sh and then 30deploy.sh etc.

The following is an example of the Docker file you can use to build the image. Download the Alpine image from the repository.

```
FROM <repository_ip>:<port>/alpine:<image_tag>
ARG D2CUSTOM

ENV D2_CUSTOM_PATH /customdir

RUN adduser -D -H dmadmin && \
    mkdir -p $D2_CUSTOM_PATH && \
    chown -R dmadmin:dmadmin $D2_CUSTOM_PATH

COPY --chown=dmadmin:dmadmin $D2CUSTOM/ $D2_CUSTOM_PATH/
CMD sh
```

1. Copy the above content to a file and name it Dockerfile. Modify the Alpine image path based on your repo location to download the image to the local environment.
2. Run the following command in the location where Dockerfile is created and the necessary script files are present to build the Docker image.

```
docker build -f Dockerfile -t <repository>/custom_image_name:<custom_image_tag> --
build-arg D2CUSTOM=<D2Scripts Dir> -- no cache .
```

For example:

```
docker build -f Dockerfile -t repo.opentext.com/d2webapphook:23.2 --build-arg D2CUSTOM=D2Customizations --no-cache .
```

3. Once the Docker image is generated, make the following change in the Helm charts when deploying D2.

- In the dockerimages-values.yaml file, add or update the extraInitContainers sections of the respective Webapp as follows:

```
extraInitContainers: |
  - name: <init_container_name>
    image: <repository_path>/<init_image_name>:<image_tag>
    imagePullPolicy: Always
    command: ['/bin/sh', '-c', 'yes | cp -rf /customdir/<webapp_name>/* /opt/D2-install/custom/']
    volumeMounts:
      - name: customconfig
        mountPath: /opt/D2-install/custom
        subPath: <webapp_service_name>
```

For example, update image details with the Docker image created previously and replace <webapp\_name> (in the command above) with D2Classic/D2Config/D2Smartview/D2Rest and <webapp\_service\_name> (in the command above) with d2classic/d2config/d2smartview/d2rest based on the Webapp you are going to place in this extraInitContainers section.

```
extraInitContainers: |
  - name: customwebappinit
    image: repo.opentext.com/d2webapphook:23.2
    imagePullPolicy: Always
    command: ['/bin/sh', '-c', 'yes | cp -rf /customdir/D2Classic/* /opt/D2-install/custom/']
    volumeMounts:
      - name: customconfig
        mountPath: /opt/D2-install/custom
        subPath: d2classic
```

- In the values.yaml file (or configuration.yaml if used) set custom, hook\_approach, and createPVC to true under the customConfigurations section for the respective Webapps.

```
customConfigurations:
  custom: true
  hook_approach: true
  createPVC: true
```



## Notes

- If hook approach customization is used for multiple Webapps, make sure createPVC is set to true only for one of the Webapps and false for the remaining Webapps.
- /opt/D2CS-install is the PVC mount location of customscriptpvc under the Documentum Content Server pod.

### 3.5.6.3 Customizing D2 Webapps containers using single init container including hook scripts (hybrid approach)

Follow the process as shown in [Section 3.5.6.1](#) with a minor change in the folder structure to build the Docker image as shown below.

```
D2Customizations/
D2Classic/
    hook_scripts(*deploy.sh)      --> hook scripts of d2 classic and necessary files
    applicable_files               --> Necessary files/dirs/sub-dirs referred in the hook
                                         script
D2Custom/
    js/                           --> All D2Classic javascript files
    css/                          --> All D2Classic css files
    wars/                         --> External WAR files
    WEB-INF/lib                   --> All D2Classic Plugins and Lib files
    WEB-INF/classes               --> All D2Classic properties files
    LP-Package/                  --> All D2Classic language pack files
    D2-Ext/
        Widget1/                 --> External Widget files
D2Smartview/
    hook_scripts(*deploy.sh)      --> hook scripts of d2 smartview and necessary files
    applicable_files               --> Necessary files/dirs/sub-dirs referred in the hook
                                         script
D2Custom/
    wars/                         --> External WAR files
    WEB-INF/lib                   --> All D2Smartview Plugins and Lib files
    WEB-INF/classes               --> All D2Smartview properties files
    LP-Package/                  --> All D2Smartview language pack files
D2Config/
    hook_scripts(*deploy.sh)      --> hook scripts of d2 config and necessary files
    applicable_files               --> Necessary files/dirs/sub-dirs referred in the hook
                                         script
    D2Custom/
        wars/                     --> External WAR files
        WEB-INF/lib               --> All D2Config Plugins and Lib files
        LP-Package/              --> All D2Config language pack files
        sampleconfig1.zip         --> Custom import zip/xml file1
        sampleconfig2.zip         --> Custom import zip/xml file2
D2Rest/
    hook_scripts(*deploy.sh)      --> hook scripts of d2 rest and necessary files
    applicable_files               --> Necessary files/dirs/sub-dirs referred in the hook
                                         script
    D2Custom/
        wars/                     --> External WAR files
        WEB-INF/lib               --> All D2Rest Plugins and Lib files
        LP-Package/              --> All D2Rest language pack files
```

Follow the steps shown in [Section 3.5.6.1](#) and [Section 3.5.6.2](#) to update values yaml or configuration yaml files with respect to the Webapp used.



**Note:** Make sure subPath: is added for the init container of the respective webapp while using the hybrid approach.

### 3.5.6.4 Language Pack support

For a list of available language support packs, see the Documentum D2 CE Release Notes. This list also includes the language codes required for deployment.

To implement a language pack, build a Docker image with the structure as shown in [Section 3.5.6.1](#) and place all of the required language pack files in the **LP-Package** folder of each Webapp. The following are the required files:

- AdvancedPublishing-LanguagePack\_xx.zip
- C2-LanguagePack\_xx.zip
- D2-Bin-LanguagePack\_xx.zip
- D2-LanguagePack\_xx.zip
- D2-Mobile\_LanguagePack\_xx.zip
- D2-Smartview\_LanguagePack\_xx.zip
- D2-Config-LanguagePack\_xx.zip

Where xx is the language code. For example, the French D2 language pack name would be `D2-LanguagePack_fr.zip`.

1. Once the image is built, update the `extraInitContainers` section of each Webapp in the `dockerimages-values.yaml` file as shown in [Section 3.5.6.1](#).
2. In the `documentum/values.yaml` file, add `localeName` under the `contentserver` section of the `dctm-server` and `customConfigurations` sections of the respective Webapp with the required language code.

For example, to use the French language add `fr` in the `localeName`.

```
content-server:  
  contentserver:  
    localename: fr  
  
d2classic:  
  customConfigurations:  
    custom: true  
    locale: fr
```



**Note:** D2 versions 22.1 and newer support multiple language packs in a single deployment. To deploy multiple language packs, make sure each language pack jar you want to deploy is present in the Docker image while building the image and update the locale names with comma-separated values, e.g., en,ar,de,es,fr,it,iw,ja,ko,nl,ru,sv,zh,pt.

3. Make the below change in the browser to view the D2 clients in their respective languages (steps given with regard to the Google Chrome browser):
  - a. Go to **Settings > Advanced > Languages > Add Language**.
  - b. Search for the required language and click **Add**.
  - c. Click the ... menu of the added language and select **Move to the top**.

- d. Relaunch the browser and access the D2 clients.

### 3.5.7 Set the `acs_base_url` to `localhost` and `acs_supported_protocol` to `s3` for S3 store/Google Cloud Store

1. Make sure `disableUpdateAcsUrl: true` for content server in `documentum/values.yaml` or `documentum/config/configuration.yml`.
2. Open the iapi in cs pod and execute the following commands:

```

API>?,c,select object_name,r_object_id from dm_acs_config
#Note down the acs_base_url by dumping any one of the ACS r_object_id & make sure
you update
all the returned ACS r_object_id using following commands
API> fetch,c,<r_object_id>
API> set,c,1,acs_base_url[0]
SET> http://localhost:9080/ACS/servlet/ACS
API> set,c,1,acs_supported_protocol
SET> s3
API> append,c,1,acs_base_url
<original ingress url>
API> append,c,1,acs_supported_protocol
https
API> save,c,1
...
OK
#To check if the parameter is updated, open a new iapi session and execute the
following command:
API> ?,c,select object_name,r_object_id from dm_acs_config
#Dump each <r_object_id> and check the acs_base_url and acs_supported_protocol
parameter
dump,c,<r_object_id>
# The 0th index of acs_base_url and acs_supported_protocol has to be localhost and
s3 correspondingly

```

### 3.5.8 Documentum Reports (DTR)

#### 3.5.8.1 Enabling/Disabling Reports with D2

By default, Documentum Reports are enabled in Documentum single Helm. To disable reports, make sure the values below are updated per the deployment environment.

1. If Documentum Reports is disabled in `documentum-components.yaml`, comment out the below shown. If Documentum Reports is enabled, make sure the `dctm-reports-installer` and `dctm-reports-d2init` container details are uncommented.

```

dctm-server:
  content-server:
    extraInitContainers:
      - name: dctm-reports-d2
        image: registry.opentext.com/dctm-reports-d2:22.4
        imagePullPolicy: *pull_policy_type
        command: [ '/bin/sh', '-c', 'yes | sudo cp -rf /opt/D2-install/custom/dtr_dar/
* /opt/dctm_docker/customscriptpvc/d2' ]
        volumeMounts:
          - name: dcs-data-pvc
            mountPath: /opt/dctm_docker/customscriptpvc

```

```

        subPath: initcontainercustomscripts/dcs-pg
      - name: dctm-reports-installer
        image: registry.opentext.com/dctm-reports-installer:22.4
        imagePullPolicy: *pull_policy_type
        command: ['/bin/sh', '-c', 'yes | sudo cp -rf /opt/dtr_build/* /opt/
dctm_docker/customscriptpvc/']
        volumeMounts:
          - name: dcs-data-pvc
            mountPath: /opt/dctm_docker/customscriptpvc

```

2. If Documentum Reports is disabled in documentum-components.yaml, comment out the below shown. If Documentum Reports is enabled, then make sure dctm-reports-d2init container details are uncommented

```

d2config:
  extraInitContainers:
    - name: dctm-reports-d2
      image: registry.opentext.com/dctm-reports-d2:22.4
      imagePullPolicy: *pull_policy_type
      command: ['/bin/sh', '-c', 'yes | cp -rf /opt/D2-install/custom/dtr_config/* /
customdir/']
      volumeMounts:
        - name: customconfig
          mountPath: /customdir

```

3. If Documentum Reports is disabled in documentum-components.yaml, comment out the below shown. If Documentum Reports is enabled, make sure dctm-reports-d2init container details are uncommented.

```

d2classic:
  extraInitContainers:
    - name: dctm-reports-d2
      image: registry.opentext.com/dctm-reports-d2:22.4
      imagePullPolicy: *pull_policy_type
      command: ['/bin/sh', '-c', 'yes | cp -rf /opt/D2-install/custom/dtr_build/* /
customdir/']
      volumeMounts:
        - name: d2report
          mountPath: /customdir

```

4. If Documentum Reports is enabled, make sure the image name and tags are updated as per the environment under the dctm-server section of documentum/dockerimages-values.yaml file.

```

dtrbase:
  imagePullSecrets: *pull_secret_name
  images:
    dtrbase:
      repository: *docker_repo
      name: dctm-reports-base
      tag: 22.4.2
      pullPolicy: *pull_policy_type
    graylog:
      image: *graylog_image
      pullPolicy: *pull_policy_type

```

5. Ensure that dtrbase is enabled in the dctm-server section of documentum/values.yaml or documentum/config/configuration.yaml.

```

dctm-server:
  dtrbase:
    enabled: true

```

6. Ensure that dtrbaseService is enabled in the dctm-ingress section of documentum/values.yaml.

```
dctm-server:
  dctm-ingress:
    dtrbaseService:
      enable: true
```

7. Ensure that `d2report` is enabled in the `d2classic` section of `documentum/values.yaml` and update the D2, DTR host, and port details.

```
d2classic:
  d2report:
    enable: true
    dctm_reports_iis_host: <fully qualified dctm ingress host e.g., dctm-ingress.d2.cfcr-lab.bp-paas.otxlab.net>
    d2_host: <fully qualified ingress host configured with d2classic service e.g., dctm-ingress.d2.cfcr-lab.bp-paas.otxlab.net>
    dctm_reports_port: "443"
    d2_port: "443"
    d2_scheme: "https"
    dctm_reports_scheme: "https"
```



**Note:** if `d2_scheme` or `dctm_reports_scheme` uses `http`, change the port number of `d2_port` or `dctm_reports_port` to 80 accordingly.

8. Enable `custom` to `true` under `customConfigurations` for `d2config` and update/uncomment `fileformat` and `filename` as shown below when Documentum Reports is enabled..

```
d2config:
  customConfigurations:
    custom: true
    filename: DCTM-Reports-Application-22.4.0-Export-Config.zip
```

### 3.5.8.2 Post-deployment steps

After successful deployment of D2 reports, follow the steps below to access reports.

#### To view reports:

1. Log in to DA.
2. Open **Administration > User Management > Groups > `dctm-reports-users`**.
3. Go to **File > Add members**.
4. Add the newly created/existing user.
5. Log in to D2 with the above user and access the reports from the Documentum Reports workspace.

#### To create/edit report templates:

1. Follow steps 1 to 4 above.
2. Open **Administration > User Management > Users**.
3. Search the user, right-click the user, and select **Assign Group Membership**.
4. Search for **`dctm-reports-designer`** and add it to the selected list by clicking the right arrow.

5. Search for **d2-admins**, add it to the selected list, and click **OK**.
6. Log in to D2 with the above user and access the reports from the Documentum Reports workspace.

### 3.5.8.2.1 Configuring OTDS for OTIV

#### To sync OTIV users to docbase

1. Go to **Access Roles**.
2. Click on **Actions** of the D2 access role and select **View access role details**.
3. Add the OAuthClients partition that was created by OTIV and save the access role.
4. Go to **Resources** and select D2 resource, **Actions > Consolidate**.

#### To allocate the license to users

1. Go to **Users and Groups**.
2. Select the required user and click on **Actions > Allocate to License**.
3. From the **License** dropdown, select **Viewing-iv** and from the **Counter** dropdown select **INTELLIGENT\_VIEWING.FULLTIME\_USERS\_BASIC** or **INTELLIGENT\_VIEWING.FULLTIME\_USERS\_REGULAR**, as per the need.
4. Consolidate the users by following the steps in the previous section.

#### To add/update oAuth clients and system attributes

1. Go to **Oauth Clients** and select the required client (where Smart View is added as Redirect URLs).
2. Click on **Actions > Properties > Advanced**.
3. Check the box for **Use session lifetime as Refresh token lifetime** and provide **Access Token life time** as **1000**.
4. Add the below values in **Permissible** scopes, Default scopes and save the Oauth client:
  - `create_publications`
  - `view_publications`
  - `view_any_publication`
  - `delete_any_publication`
  - `delete_publications`
  - `write_any_markups`
  - `read_any_markups`

5. Add the below attributes in **System Config** under **System Attributes** tab. Click on **Add Attribute** button.
  - Set the **Name** as **otds.as.SameSiteCookieVal** and **Value** to **None**.
  - Click on **Add Attribute** button again.
  - Set the **Name** as **directory.auth.EnforceSSL** and **Value** to **False**.

#### To configure Intelligent Viewing

1. In D2-Config, click on **Tools > OpenText Viewing...**
2. Fill all the fields with relevant details. All the URLs can be obtained from otiv ingress.



**Note:** Mention only the protocol and the ingress host name .

3. OTDS Client ID and Secret must be obtained from OTDS.
  - Note:** If OTDS Client secret is available already, use the same and following steps can be ignored.
    - a. Login into OTDS and get the Client ID and Secret.
    - b. Edit the OAuth client used under **OAuth Clients**, change it to a public client by clearing the **Confidential** check box, and then save the OAuth client.
    - c. Next, edit the OAuth client again, select the **Confidential** check box, and then save the OAuth client to generate OTDS Client secret.
  - 4. Create a new widget by name "IntelligentViewer" and set widget type to **OpenTextIntelligentViewerWidget**.
  - 5. Upload json configurations to dynamically add/remove the pane/panels/components based on customer requirement on the viewer and upload the below JSON in the corresponding tabs.

We can also disable annotations directly by checking the checkbox available on the widget for certain groups of users.

JSON Configuration file can be obtained from D2 smartview pod. Use the command below to copy json files to the local system.

- opentext\_intelligent\_viewing\_config\_template.json
- opentext\_iv\_text\_compare\_config\_template.json
- opentext\_iv\_graphical\_compare\_config\_template.json

```
kubectl cp d2smartview-0:/opt/tomcat/webapps/<d2smartview_webapp_name>/WEB-INF/
classes/opentext_intelligent_viewing_config_template.json -n <namespace>
<local_system_path>/opentext_intelligent_viewing_config_template.json
```

6. In D2-Config > Widget View > Smart View Landing Page... download SmartView landing page XML and add entry "<otiv-widget>IntelligentViewer</

otiv-widget>" to the SmartView landing page XML under the parent tag <default-widgets>.

 **Note:** Whatever the widget name given here must be updated in landing page XML.

7. Map the newly created widget against the context in D2 Config matrix.
8. In D2-Config > Tools > Refresh Cache.
9. Verify if the following user is present in DA after consolidation in OTDS or else create the user in DA.

```
iv-publisher    iv-publisher@OAuthClients
```

### 3.5.9 Deploying optional components for Documentum

#### 3.5.9.1 Deploying Documentum Connector for Core (DCC)

**To deploy DCC:**

1. To enable DCC, enable `dctm-ingress.dcc.enable` and `dctm-ingress.syncagent.enable` to true, as shown below:

```
dctm-ingress:  
  dcc:  
    enable: true  
  
  syncagent:  
    enable: true
```

2. Add/Uncomment the below init container section under `dctm-server` in `documentum/dockerimages-values.yaml`.

```
dctm-server:  
  content-server:  
    extraInitContainers:  
      - name: dcc-dar-installer  
        image: registry.opentext.com/dctm-dcc-darinitcontainer:22.4  
        imagePullPolicy: Always  
        command: ['/bin/sh', '-c', 'yes |cp -rf /opt/customscripts/* /opt/  
dctm_docker/customscriptpvc/]'  
        volumeMounts:  
          - name: dcs-data-pvc  
            mountPath: /opt/dctm_docker/customscriptpvc  
            subPath: initcontainercustomscripts/dcs-pg
```

3. Under the `dctm-dcc` section:

- set `dctm-dcc.enable` to true
- update `dctm-prefix` to the desired prefix value
- update all occurrences of place holders `<db_host>`, `<db_port>`, and `<db_name>`
- update `database.username`, `database.password`, `database.dbunencryptedpassword`, `database.db_host`, and `db_port` per your environment

- update all proxy.enabled to true for proxy.proxyHost and proxy.proxyPort configurations in order to connect to the internet
- set pvc.createPV to false & ingress.enableDCCIngress to true

For example:

```

dctm-dcc:
  enable: true

  prefix: dcc

  ## Database configuration ##
  database:
    username: postgres
    password: AAAAEPEPDoF6jSARBH0pnSPzSTplGjBo6ZoyTveVa80egQZPyrRSLsaurkhQY=
    dbunencryptedpassword: password
    db_host: db-pg-0.db-pg.dasaris2.svc.cluster.local
    db_port: 5432

  ## meta data service ##
  metadata:
    enable: true
    dbname: postgres
    configMap:
      database:
        url: jdbc:postgresql://db-pg-0.db-pg.dasaris2.svc.cluster.local:5432/
postgres?escapeSyntaxCallMode=callIfNoReturn

    ## Syncagent ##
    syncagent:
      prefix: syncagent
      enable: true
      dbSchemaInit:
        enable: true
        dbname: postgres
      configMap:
        database:
          ## url specified the url used to connect to the DB for syncagent
          service.
          url: jdbc:postgresql://db-pg-0.db-pg.dasaris2.svc.cluster.local:5432/
postgres?escapeSyntaxCallMode=callIfNoReturn

    proxy:
      enable: true
      proxyHost: bp2-prox01-1001.otxlab.net
      proxyPort: 3128
      noProxyHosts: "localhost|127.0.0.1|*.otxlab.net|10.100.*|10.200.*|
10.194.42.237|10.194.54.43|serviceregistry|10.194.*|rmsync-consul-service"

    ## Syncnshare manaul ##
    syncnshareManual:
      prefix: syncagent-manual
      enable: true
      dbSchemaInit:
        enable: true
        dbname: postgres
      configMap:
        database:
          ## url specified the url used to connect to the DB for syncshare manual

```

```

service.
    url: jdbc:postgresql://db-pg-0.db-pg.dasaris2.svc.cluster.local:5432/
postgres?escapeSyntaxCallMode=callIfNoReturn

## Core Notification ##
coreNotification:
    prefix: core-notification
    enable: true
    dbSchemaInit:
        enable: true
        dbname: postgres
    configMap:
        database:
            ## url specified the url used to connect to the DB for core notification
    service.
        url: jdbc:postgresql://db-pg-0.db-pg.dasaris2.svc.cluster.local:5432/
postgres?escapeSyntaxCallMode=callIfNoReturn

proxy:
    enable: true
    proxyHost: bp2-prox01-1001.otxlab.net
    proxyPort: 3128
    noProxyHosts: "localhost|127.0.0.1|*.otxlab.net|10.100.*|10.200.*|
10.194.42.237|10.194.54.43|serviceregistry|10.194.*|rmsync-consul-service"

## Persistent volume claim ##
pvc:
    createPV: false

ingress:
    ##enable below if we need to use dcc ingress
    enableDCCIngress: false

```



## Notes

- For dcc to work, we need dcm-rest to be deployed with otds disabled.

```

dctm-ingress:
    restService:
        enable: true

dctm-rest:
    enable: true
    otds:
        enable: false

```

- Make sure dctm-dcc.prefix and dctm-ingress.dcc.prefix are the same.

## Post-deployment steps:

1. Log in to syncagent on URL <dctm-ingress>/syncagent/ with credentials username: Administrator, password: Administrator.
2. Under global settings, under Source repository, click on edit under the Repository superuser account.
3. Enter login credentials and DCTM-rest URL in DCTM url section, select desired Repository from the drop down, and click Save.
4. Log in to Core Share with you credentials.

5. Under the security section, find OAuth Confidential Clients, enter a Client Description, and enter <dctm-ingress>/syncagent/oauthcallback under the Redirect URLs field.
6. Copy the client secret to Notepad.
7. Under existing oauth clients, find the newly created client and copy the client ID.
8. Go back to <dctm-ingress>/syncagent/.
9. Under global settings, under Target Repository, click on edit.
10. Add your Core Share URL in Opentext Core URL (Client ID and secret will be the ID and secret copied in steps 6 and 7) and click on Save.
11. Log in to DA, click on User management under Administration, click on groups and search for dmc\_sync\_users, and add Users that can use dcc.

### 3.5.9.2 Documentum Workflow Designer

#### 3.5.9.2.1 Deploying the Documentum Workflow Designer

To enable the Documentum Workflow Designer, you must enable it and provide a truststore password and port. You must also update the new relic app name if new relic is enabled in documentum/values.yaml.

For example:

```
dctm-server:
  dctm-workflow-designer:
    enabled: true
    docbaseConnection:
      truststorePassword: password
      port: 1489
    #Tomcat Admin credentials:
    tomcat:
      javaOptions: <javaOptions>

  newrelic:
    app_name: <newrelic_app_name>
```



**Note:** To enable OTDS for WFD, update the below values as shown:

Key	Value
url	<dctm-ingress> # if d2ingress is enabled then <d2-ingress>
client_id	OTDS client id which has Documentum Workflow Designer's redirect URL
reverseproxy_url	<vpn-ingress>
logon_appname	DocumentumWorkflowDesigner

```
dctm-workflow-designer:
  otds:
```

```
enable: *otds_enabled
url:
reverseproxy_url:
client_id:
client_secret:
logon_appname:
```

### 3.5.9.2.2 Post-deployment instructions

After deploying Documentum Workflow Designer, you must grant users access to it.

1. Open DA.
2. Open **Administration > User Management > Roles > documentum\_workflow\_designer**.
3. Click **Add members**.
4. Add the users to the group to grant access to the Documentum Workflow Designer.

### 3.5.9.2.3 Enabling OTDS for Documentum Workflow Designer

1. Open <http://<dctm-ingress>/otds-admin>.
2. Open OAuth Clients and click the OAuth Client used for D2 redirect URLs.
3. If OTDS is enabled, ensure Confidential is checked for the client id in which WFD is added.
4. Update the client secret as shown and upgrade the deployment.

```
dctm-workflow-designer:
  otds:
    client_secret:<client secret>
```

### 3.5.9.3 Deploying Documentum Content Connect

1. To deploy Documentum Content Connect, set contentconnect to true in the `documentum/values.yaml` file and update DB\_IP, DB\_PORT, DB\_DB and DB\_TABLESPACE\_NAME with the database details.

Update the anchor tag ccExtension to any value other than the default value 'cc' if needed:

```
ccExtension: &ccExtension cc
```

 **Note:** From release 23.2 onwards, Documentum Content Connect will use the unified dctm-ingress by default and not the separate cc-ingress available. If you want to explicitly use cc-ingress, enable it and disable the cc services in dctm-ingress as follows:

```
contentconnect:
#Ingress
  ingress:
    enabled: true
    name: cc-ingress
    configureHost: true
```

```

    tls:
      enable: false
      hosts: *ingress_domain
    rules:
      host: <ingress prefix> # cc-ingress prefix value should be given here

dctm-ingress:
  ccService:
    enabled: false
    serviceName: cc
    servicePort: 8080
    extension: *ccExtension
  ccadminService:
    enabled: false
    serviceName: cc-admin
    servicePort: 80
    extension: *ccExtension

```

2. Update the `clientId`, `clientSecret` and `tenantId` based on the Content Connect app registration details in Azure portal for using Microsoft Graph API, This is a pre-requisite to deploy Content Connect pod.

 **Note:** For more details on Microsoft Graph configuration, see the Content Connect Administration and Installation Guide.

```

contentconnect:
  configmap:
    #CC app registration details in azure portal for using Microsoft MS Graph
    API, This is a Pre-requisite to deploy CC pod
    clientId: <CLIENT-ID>
    clientSecret: <CLIENT-SECRET>
    tenantId: <TENANT-ID>

```

3. Update protocol to IPV4 for Kubernetes version =<1.20 and IPV6 for Kubernetes version > 1.20.

```

contentconnect:
  configmap:
    protocol: IPV6

```

4. Update the `newrelic_app_name`.

For example:

```

dctm-server:
  contentconnect:
    newrelic:
      new_relic_app_name: <NEWRELIC-APPNAME>

```

5. As shown, update the following D2 webapps section when ContentConnect is enabled in the documentum/values.yaml file:

```

d2classic:
  settings:
    allowedFrameOrigins: https://<host>.<dctm-ingress-url> https://
    *.sharepoint.com https://*.officeapps.live.com/ https://outlook.office.com
    appsforoffice.microsoft.com outlook.office365.com https://inc-word-
    edit.officeapps
  esapi:
    forcedSecureCookies: true
d2smartview:
  restApiRuntime:
    CookieConfiguration:
      Enable: true
    ContentConnect:

```

```
        Enable:true
        restAllowedOrigins: https://<host>.<dctm-ingress-url> https://
*.sharepoint.com https://*.officeapps.live.com/ https://outlook.office.com
appsforoffice.microsoft.com outlook.office365.com https://inc-word-edit.officeapps
d2rest:
    restApiRuntime:
        ContentConnect:
            AllowCors:
                Enable: true
#The restAllowedOrigins value should be in the format- https://<host>.<dctm-ingress-
url>:<port>
    restAllowedOrigins: https://<host>.<dctm-ingress-url>:<port>
```

### 3.5.9.3.1 Enabling OTDS for Content Connect

1. Open <http://<dctm-ingress>/otds-admin>, and go to the **OAuth Clients** section and click the OAuth client used for D2 redirect URLs.
2. Click **Next** and navigate to the **Redirect URLs** section.
3. Click **Add** and specify the Content Connect URL.  
For example: `https://<ContentConnectServer>:<port>`
4. Click **Save**.
5. Add the following sites to the **Trusted Sites** section:  
`https://<host>.<dctm-ingress-url>
https://*.sharepoint.com
https://outlook.office.com
https://*.officeapps.live.com/
https://outlook.office365.com`
6. Save the configurations.

### 3.5.9.3.2 Content Connect Admin Console Configurations

Log in to the application using the URL of the format `https://<host>.<dctm-ingress url>/<ccExtension>/admin`. `<dctm-ingress-url>` is the `dctm-ingress` host given in the `dctm-ingress` section of `documentum/values.yaml` and `<ccExtension>` is the value given for the anchor tag `ccExtension`. For example: <https://bhats1dctm-ingress.d2.cfcr-lab.bp-paas.otxlab.net/cc/admin>.

1. Provide the default user credentials as **SystemAdmin/SystemAdmin@123**. When you first log in to the Content Connect Admin Console, it is mandatory to change the default password for the **SystemAdmin** user. Also, change the password for the **BusinessAdmin** user. The default password for this user is **BusinessAdmin@123**.
2. Log in to the Admin Console with the new password.
3. Set *Authentication type* to OTDS, then provide the OTDS Server info (for example: `https://<host>.<otds-ingress>/otdsws`) and the OTDS Client ID created in the previous step.
4. Select endpoints as **Documentum-D2-REST-Server** and then provide the d2-rest endpoint (for example: `https://<host>.<ingress-url>/d2-rest`

5. Test the connection by clicking the **Test** button. If the test is successful, you see a **Success** message.

You can either point to D2 Classic or D2 Smart View.

- To point to D2 Classic, provide the D2 client URL (for example, `https://<host>.<ingress-url>/D2`).
  - To point to D2 Smart View, select the **D2 Smart View** check box and provide the D2 Smart View URL (for example, `https://<host>.<ingress-url>/D2-Smartview`)
6. Provide value for Outlook REST URL, for example: For Office 365, value will be `https://outlook.office365.com`.
  7. Enable “Enable Content Connect Add-in for Office Online” under Office settings section. (This is optional, need to be enabled for Word, Excel, PowerPoint online application to work). Click Save to save all configurations.
  8. Check the option “Enable proxy server” and provide the proxy server URL according to your environment.
  9. Click the **Download manifest** button to download the Content Connect manifest files for Outlook and Word from the Admin Console and Microsoft Office from the Content Connect Admin Console.

#### 3.5.9.4 Enabling Documentum Records Client/Records Queue Manager

Provide the following product license keys for the Documentum Records Client/Records Queue Manager product components to be deployed under the **cs-secrets** section of the `documentum/values.yaml` file:

```
dctm-server:
  cs-secrets:
    docbase:
      licenses:
        recordManager: <license-key>
        prm: <license-key>
        fedRecdService: <license-key>
        rps: <license-key>
```

To enable Documentum Records Client/Records Queue Manager, set the following components to *true*:

```
dctm-server:
  recordsdarinstallation:
    enabled: true

  records:
    enabled: true
  rqm:
    enabled: true
```

Update the ingress and OTDS details in the records components to match your environment. For example:

```
dctm-server:
  records:
    ingress:
```

```

    name: records-ingress-test
    enable: true
    ingressHostName: records-ingress-host
    clusterDomainName: d2.cfcr-lab.bp-paas.otxlab.net
    otds:
      url: <OTDS_URL>
      clientID: <CLIENT_ID>
      scheme: https
  
```

Update the following parameters in the RQM component to match your environment. For example:

```

dctm-server:
  rqm:
    userName: dmadmin
    containers:
      rqm:
        rqmDocbaseUser: dmadmin
        rqmSysAdminName: recordsadmin
        rqmSysAdminPass: password
        docbrokerhostname: dbr-0.dbr.<namespace>.svc.cluster.local
        docbrokerport: 1489
  
```

### 3.5.10 Checking deployments

You can use `kubectl logs <pod name>` to check all the pods.

Users should be able to access the D2 services and OTDS service through ingress with the following URLs:

Use any browser with `{{.Values.ingressProtocol}}://{{.Values.dctm-ingress.ingress.host}}.{{.Values.dctm-ingress.ingress.clusterDomainName}}/D2-Config` to access the D2 Config webapp

Use any browser with `{{.Values.ingressProtocol}}://{{.Values.dctm-ingress.ingress.host}}.{{.Values.dctm-ingress.ingress.clusterDomainName}}/D2` to access the D2 client webapp

Use any browser with `{{.Values.ingressProtocol}}://{{.Values.dctm-ingress.ingress.host}}.{{.Values.dctm-ingress.ingress.clusterDomainName}}/D2-Smartview` to access the D2 smartview webapp

Use any browser with `{{.Values.ingressProtocol}}://{{.Values.dctm-ingress.ingress.host}}.{{.Values.dctm-ingress.ingress.clusterDomainName}}/d2-rest` to access D2 rest.

Use any browser with `{{.Values.ingressProtocol}}://{{.Values.dctm-ingress.ingress.host}}.{{.Values.dctm-ingress.ingress.clusterDomainName}}/otds-admin` to access OTDS admin site.



**Note:** If you are using custom D2 webapp names, use the following URL to access the D2 webapps:

```

{{.Values.ingressProtocol}}://{{.Values.dctmingress.ingress.host}}.{{.Values.dctm-ingress.ingress.clusterDomainName}}/{{.Values.d2classicWebappName}}
{{.Values.ingressProtocol}}://{{.Values.dctmingress.ingress.host}}.{{.Values.dctm-ingress.ingress.clusterDomainName}}/{{.Values.d2configWebappName}}
{{.Values.ingressProtocol}}://{{.Values.dctmingress.ingress.host}}.{{.Values.dctm-ingress.ingress.clusterDomainName}}/{{.Values.d2restWebappName}}
  
```

```
ingress.ingress.clusterDomainName}}/{{.Values.d2smartviewWebappName}}
{{.Values.ingressProtocol}}://{{.Values.dctmingress.ingress.host}}.{{.Values.dctm-
ingress.ingress.clusterDomainName}}/{{.Values.d2restWebappName}}
```

### 3.5.11 Upgrading or patching D2 CE single Helm deployment

This section includes instructions about how to upgrade from one version of D2 to a newer version.

- Get the desired version of D2 charts.
- In Documentum 23.4, DFS, DCTM-REST, DCTM Reports, DCTM-CMIS, OTIV, BPS, xDA, DCTM Records/RQM, xECM TTeam Integration (smartviewm365), and Content Connect components are enabled by default. Enable/Disable the components as per your requirement during upgrade in `documentum-components.yaml`.
- Update `d2/dockerimages-values.yaml` and `d2/Chart.yaml` with the newer versions.
- Modify the `values.yaml` according to your old deployment.
- Ensure to back up the database before proceeding to the upgrade. Before the database backup, set the docbase to Dormant state using the steps provided in DCTM Server Cloud Deployment Guide.



**Note:** Once the database backup is done, revert the Dormant state to Active state, log in to DA with backupadmin user and navigate to **Administration>Basic Configuration>Repository**, select the docbase name and right-click, then choose **Make Active**.

#### Upgrade notes:

- Please verify Kubernetes version support for desired D2 version before upgrade, check release notes for details.
- If you used the `d2-resources-values-<config>.yaml` file during deployment, before upgrading, ensure that the replica counts for all the pods are the same as the existing deployment's `d2-resources-values-.yaml` file.
- Make sure `configNameOption` in `d2/values.yaml` file is the same as the previous deployment, i.e., if it's HOSTNAME retain it as HOSTNAME and if it's HA retain it as HA.

```
dctm-server:
  content-server:
    otds:
      configNameOption: HOSTNAME
```

- Ensure that the PVC sizes for all components are equal or greater than the PVC sizes in the existing deployment's `d2-resources-values-.yaml` file.
- If DA is part of the older deployment, do the following steps to clear the existing DA deployment before proceeding with the update:

```
kubectl delete deployment.apps/da -n <namespace>
kubectl delete pvc da-pvc -n <namespace>
```

- From 23.4 release onwards, d2 master helm chart has been renamed to documentum. So if you are upgrading to version 23.4 and above, read d2 as documentum for the helm chart name and folder structure in any of the upgrade steps.

From OTDS 22.1.0 onwards, its architecture has been changed. OpenDJ will no longer be used as a directory server. Instead, the existing/new Postgres/SQL Server/Oracle/SAP database can be used. While directly upgrading OTDS from 21.3.x or older versions to 22.1.0 or higher, data will be lost. Data migration is needed before upgrading from the previous release.

#### Migrating data from a running OpenDJ container:

- When migrating data from a running OpenDJ container, an OTDS 22.1.0 temporary instance is needed to connect to the previous OpenDJ service and migrate the data. To do that, scale down the existing otdswn replicas to 0 and scale the OpenDJ replicas to 1 using the following commands:

```
kubectl scale deployment otdswn --replicas=0 kubectl scale statefulset opendj --replicas=1
```
- Download the 22.1.0 OTDS Helm chart or copy the otds folder from 22.2 d2/charts/otds to a temporary location and update the values yaml file with the below values:

- Set otdsUseReleaseName to true
- Set otdswn.publicHostname to otds-migrate
- Set otdswn.migration.enabled to true
- Provide existing database details as shown below:

```
otdsdb.url = jdbc:postgresql://<db_host>:<db_port>/postgres  
otdsdb.username = <db_username>  
otdsdb.password = <db_password>
```



**Note:** otdsdb.url should be given per the database being used. Sample values are given below:

- jdbc:postgresql://postgres.domain.local:5432/otdsdb
- jdbc:sqlserver://ms-sql.domain.local:1433;databaseName=otdsdb
- jdbc:oracle:thin:@oracle.domain.local:1521:otdsdb
- jdbc:sap://hana.domain.local:30015/?databaseName=otdsdb

- Provide docker image and tag details:

```
image.source = <artifactory_location>  
image.name = <otds-server_image_name>  
image.tag = <otds-server_image_version>
```

- Deploy the modified OTDS Helm chart using the command below:

```
helm install otds-migrate <path_to_the_values_yaml_directory_of_otdschart>
```

- If upgrading an existing Helm v2 deployment, migrate the existing deployment to helm v3 using these instructions: <https://helm.sh/blog/migrate-from-helm-v2-tohelm-v3/>
- Verify the otdsws container logs (of the temporary deployment), access the OTDS front-end using the newly created ingress, and check if the data/functionality is intact. Once everything is working fine, uninstall the temporary deployment using the below command and proceed with the D2 upgrade, making sure migration is set to false in the D2 Helm charts while upgrading.

```
helm uninstall otds-migrate
```

Refer to the below table for additional sections to be followed for upgrade when upgrading from specific versions.

From D2 version / To D2 version	20.4	21.1	21.2	21.3	21.4	>=22.1
20.3	Section 1	Section 2				
20.4	N/A	Section 2				
21.1	N/A	Section 2		Section 3	Section 4	
21.2	N/A			Section 3	Section 4	
21.3	N/A			Section 3	Section 4	
21.4	N/A				Section 4	

For example, if I am upgrading from 21.2 to 22.2, the sections I need to follow are Section-3 and Section-4.

### 3.5.11.1 Section 1: From 20.3 to 20.4

If upgrading an existing helm v2 deployment, then migrate the existing deployment to helm v3 using these instructions: Helm | How to migrate from Helm v2 to Helm v3 (<https://helm.sh/blog/migrate-from-helm-v2-to-helm-v3/>).

### 3.5.11.2 Section 2: From 20.4 to 21.2 or a newer version

When upgrading D2 to version 21.2 or newer, you must also upgrade the Documentum Server to version 21.2 or newer.

D2 image versions 21.2 and newer work with Documentum Server version 21.2 and newer which uses tomcat as the Java Method server. Documentum Server version 21.1 and older uses wildfly as the Java Method server.

### 3.5.11.3 Section 3: From 21.2 to 21.4

1. Delete the otdsws service with the following command:

```
kubectl delete svc otdsws
```

2. Delete the d2config, d2rest, and d2smartview statefulsets as follows:

```
kubectl delete statefulset --cascade=orphan d2config
kubectl delete statefulset --cascade=orphan d2rest
kubectl delete statefulset --cascade=orphan d2smartview
```

3. Edit the PVC size of the following PVCs to 1Gi:

```
d2config-vct- <pod-name>
d2rest-vct- <pod-name>
d2smartview-vct- <pod-name>
```

For example:

```
kubectl edit pvc d2config-vct-d2config-0
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 107374182400m
  storageClassName: trident-nfs
  volumeMode: Filesystem
  volumeName: pvc-a32685ea-4b50-400d-b416-93e43f79acd0
status:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 103Mi
  phase: Bound
```

4. Under resources.requests, change storage from 107374182400m to 1Gi, and then save the file.

### 3.5.11.4 Section 4: Upgrade from 22.1 or higher

Please follow the steps below after a successful upgrade:

1. If existing deployment is with Google object store or S3 store, set the default file store to filestore\_01 using below iapi commands by logging into CS pod.

```
? ,c,alter type dm_sysobject set DEFAULT STORAGE = 'filestore_01'
? ,c,alter type dm_document set DEFAULT STORAGE = "filestore_01"
```

2. In 23.4, newrelic is disabled by default, so if you are upgrading to 23.4 and if newrelic is enabled in existing deployment, then enable newrelic in 23.4 documentum/values.yaml by updating the below mentioned anchor tag to true.

```
newrelic: &newrelic_enabled true
```

3. If Content Connect is already enabled in the existing deployment, set the below value to false in documentum/values.yaml so that post upgrade it'll use the same database instead of creating a new one.

```
dctm-server:
  content-server:
    contentconnect:
```

```
contentconnectdb:
  value: false
```

4. Uncomment and update the values in dctm-workflow-designer config map template yaml with existing deployment details (make sure to use correct ingress w.r.t. 22.2) under d2\charts\dctm-server\charts\dctm-workflow-designer\templates\dctm-workflow-designer-configmap.yaml.

```
# OTDS SSO authentication settings
rest.security.auth.mode-otds-basic
rest.security.otds.login.url={otds-idp}/otdsws/login?
response_type=token&client_id={client_id}&client_secret={client_secret}&logon_appname={app name to display in log on screen}
rest.signon.logout.url=/otds-signin.jsp?logout=yes
rest.security.otds.reverseproxy.url=<WFD URL/base URL of the reverse proxy/load balancer>
```

5. OTAG-11262: awg upgrade from 21.3.0 to 21.3.1 fails when upgrading D2 from 21.3 to 21.4. Due to this known issue in AWG, the upgrade is not supported for D2 21.4 release, and AWG needs to be deleted before doing the upgrade.
6. If OTDS auto-configuration is not used in the previous deployment, then update the Redirect URLs under the **Oauth** clients section from d2ingress to the dctm ingress host.
7. Update the trusted sites to dctm ingress in the OTDS admin portal if any.
8. If CC is enabled in the previous deployment, the URLs in the CC admin portal should be updated to the dctm-ingress host and new manifest files should be used in Office365 and Outlook to work with the content connect plugin.
9. If you are upgrading D2 to 22.1 or higher, then update the ingress annotation values present in the d2/platforms/<platform>.yaml file according to your platform's ingress if necessary, and then run the following command:

```
helm dependency update .
helm upgrade <d2deployment> . --values config\configuration.yml --values
dockerimages-values.yaml --values d2-resources-values-<config>.yaml --values
platforms/<platform>.yaml -n <namespace>
```

where:

*config* can be:

```
- small-standard, small-enhanced, small-medium-standard, small-mediumenhanced
- medium-standard, medium-enhanced, medium-large-standard, medium-large-enhanced
- large-standard, large-enhanced, extra-largestandard,extra-large-enhanced
```

*platform* can be:

```
cfcf, awt, gcp, azure, openshift
```



**Note:** If you are upgrading to version 23.4 and above, documentum-components.yaml should be used for enabling or disabling the components. Also the resource yaml names now begin with documentum and not d2. So the helm upgrade command will be as follows:

```
helm upgrade <d2deployment> . --values config\configuration.yml --values
dockerimages-values.yaml --values documentum-resources-values-<config>.yaml --values
platforms/<platform>.yaml --values documentum-components.yaml -n <namespace>
```

10. If client URLs were added in D2Config (**Tools>Options>Client URLs**), update them according to their respective ingresses and perform **Tools>Reload Options, Tools>Refresh Cache**.
11. Once the deployment is complete and configured with native filestore and the functional testing is completed, make the gcp store as default store using the below iapi command in CS pod.

```
kubectl exec -it cedcs-pg-0 -nxcpc  
iapi <docbase_name>  
API> ?,c,alter type dm_sysobject set DEFAULT STORAGE = '<gcp/s3-store>'  
API> ?,c,alter type dm_document set DEFAULT STORAGE = 'gcp/s3-store'  
API> reinit,c
```

12. Steps to check the functionality of GCP-object store with default as true:

- Try to push/pull the content through the iapi session by following the below steps (while pushing the content, do not set a\_storage\_type as content will go to the default store by default):

```
API> create,c,dm_document  
...  
0901e24080002dee  
API> setfile,c,1,/opt/dctm/share/sample.txt,crtext  
...  
OK  
API> save,c,1  
...  
OK  
API> getpath,c,1  
...  
https://storage.googleapis.com/testidlit/01e240/80/00/11/ce.dat  
API> getfile,c,1
```



**Note:** The content is now stored in gcp store <https://storage.googleapis.com/testidlit/01e240/80/00/11/ce.dat> and the content will be in encrypted format. Anytime before dar installation, the default store should be switched to the filestore.



**Note:** Rollback is not supported in 22.2 to previous versions since the d2installer is moved as an init container and there is no need for shared-pvc.

### 3.5.12 Rollback

#### To roll back to any previous revision of the release:

1. If for some reason you want to roll back to any previous revision of the release, run the following command:

```
helm rollback <release> [REVISION]
```

The first argument of the rollback command is the name of a release, and the second is a revision (version) number. If this argument is omitted, it will roll back to the previous release. To see revision numbers, run 'helm history release name'.



**Note:** DAR rollback is not supported yet and is tracked with JIRA DTWO=69317.

2. After the helm rollback is successful, restore the database which was backed up prior to the upgrade process.



### Caution

Do not restore the database without bringing the docbase to Dormant state. Uninstall DA and stop the CIS and CTS services. Then restore the Dctm server database which was backed up prior to the upgrade process. Let the docbase be in Dormant state until all the components are rolled back.

3. If you are rolling back from D2 21.3 or higher to 21.2, then follow these steps before roll back:

- a. If DA is part of the deployment, then do the following steps to clear the existing DA deployment before proceeding with the roll back.

```
kubectl delete deployment.apps/da -n <namespace>
kubectl delete pvc da-pvc -n <namespace>
```

- b. Delete the otdsws service by running the following command:

```
kubectl delete svc otdsws -n <namespace>
```

- c. Delete the d2config,d2rest and d2smartview stateful sets as follows:

```
kubectl delete statefulset --cascade=false d2config - n <namespace>
kubectl delete statefulset --cascade=false d2rest - n <namespace>
kubectl delete statefulset --cascade=false d2smartview - n <namespace>
```



### Notes

- If we get the below warning then we could use orphan value in place of false.

*warning: --cascade=false is deprecated (boolean value) and can be replaced*

- Appworks-gateway rollback/downgrade is not supported.
- Rollback is not supported from 22.2 to previous versions since d2installer is moved as an init container and there is no need for shared-pvc.

## 3.5.13 Cleaning up the deployment

### To clean up the deployment:

1. Run the following script:

```
helm del <d2deployment> --namespace <your namespace> (for helm v3)
kubectl delete pvc --all --namespace <your namespace>
```

2. Delete all of the PVs being used by the deployment if any of them are not cleared after clearing the pvc's.

```
# get the names of the pv's
$kubectl get pv
NAME          CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS
```

```

CLAIM                                STORAGECLASS  REASON   AGE
dasaris2cspv      10Gi       RWX        Retain    Bound  dasaris2/dcs-pg-
pvc          efs-sc           2d22h
dasaris2custompv  2Gi        RWX        Retain    Bound  dasaris2/
d2custom-shared-pvc efs-sc           2d22h

# delete the pv's
$kubectl delete pv <pv-name>
persistentvolume "<pv-name>" deleted

#verify all pv's have been deleted
$kubectl get pv
No resources found.

```

### 3.5.14 SMTP Configuration

Before configuring properties in the `smtp_c6.properties` file, you must configure the mail server in D2 Config. To configure the SMTP properties, you must modify the `smtp_c6.properties` file using the `d2classic-smtpc6-properties.yaml` file.

Property	Description
<code>Hostname</code>	The hostname of the SMTP mail server. This value supercedes the server specified in <b>D2-Config &gt; Email Configuration</b> .
<code>Port</code>	The port number of the SMTP mail server. This value supercedes the port specified in <b>D2-Config &gt; Email Configuration</b> .
<code>from</code>	Specifies an email address for <code>mail.smtp.from</code> property in <code>smtp_c6.properties</code> . This value supercedes the D2 end-user email address listed in the <code>dm_user</code> table.
<code>Starttls.enable</code>	If enabled and is supported by the server, allows use of the STARTTLS command to switch the connection to a TLS-protected connection before issuing any login commands. Defaults to <code>false</code> .
<code>auth</code>	If <code>true</code> , attempts to authenticate a user using the AUTH command. Defaults to <code>false</code> .
<code>socketFactory.port</code>	Specifies the port to connect to when using the specified socket factory. If not set, the default port is used.
<code>socketFactory.class</code>	If set, specifies the name of a class that implements the javax.net.SocketFactory interface. This class is used to create SMTP sockets.
<code>socketFactory.fallback</code>	If set to <code>true</code> , failure to create a socket using the specified socket factory class causes the socket to be created using the java.net.Socket class. Defaults to <code>true</code> .

### 3.5.15 Brava certificate configuration

You need to place the Brava certificate (self-signed or CA certificate provided by Brava for communication) in the following directories: `d2classic/certificates/` and `d2smartview/certificates/`. Then, update the parameter in `values.yaml` or `documentum/values.yaml` for single helm deployment. Then, set the `bravatls.enable` value to true and provide the certificate name at `bravatls.certName`.

For example:

```
bravatls:
  enable: true
  certName: <Certificate placed in d2classic/certificates/ and d2smartview/
certificates/>
```

### 3.5.16 Graylog logging

Graylog is enabled by default for D2 web application containers.

If you do not want Graylog enabled and want to store application logs in the persistent volume, then disable it by setting the anchor tag parameter `graylog_enable` to false in `d2/values.yaml`.

All log files of D2 web applications (d2config, d2classic, d2smartview, and d2rest) are then stored in the persistent volumes.

The `resources-values.yaml` files contain the log vct size configuration, which you can use to configure the volume size for storing log files.

### 3.5.17 Deploying D2 in Independent Java Method Server (IJMS) on private cloud

If you want to install D2 in an already deployed IJMS container, use the following steps but run the helm upgrade command instead of helm install command.

**To deploy D2 in IJMS on a private cloud:**

- Before you deploy IJMS, enable the init container in the IJMS helm chart (`ijms/values.yaml`) using the following:

```
custom:
  useInitContainers: true
```

- Add the following entries in the `extraInitContainers` section of `ijms/values.yaml`:

```
extraInitContainers:
- name: d2ijmsinit
  image: <d2-ijms-init-container-image-name>
  command: ['/bin/sh', '-c', 'yes | cp -rf /customscripts/* /opt/dctm_docker/
customscriptpvc/']
  volumeMounts:
  - name: ijms-data-pvc
```

```
mountPath: /opt/dctm_docker/customscriptpvc  
subPath: ijmscustomscripts/<serviceName>
```

Where *<serviceName>* is the IJMS service name.

For the image, provide the D2 IJMS init container core pack or the plus pack image based on your requirements. Refer to [step 8](#) in ["Prerequisites"](#) on page 117 for the image name.

3. Deploy IJMS using helm install command. For complete steps, refer to the platform Cloud Deployment Guide.
4. Verify the IJMS container logs using the following command:

```
kubectl logs <ijms-container-name>
```

The logs include the following entries which indicate successful deployment of D2 in IJMS:

```
custom_script_execution  
2d2deploy.sh  
Create the target directory for the D2 installation.  
-----  
---  
Install/Configure D2  
The D2 install log name is 20d2deploy_install.log  
Installing D2 IJMS now  
D2 install completed successfully  
custom script succeeded: /opt/dctm_docker/custom_script_pvc/20d2deploy.sh  
All custom scripts passed  
Successfully configured IJMS in the container. Please check /opt/dctm_docker/logs/  
<logfilename>.log for more details  
ijms_startup.sh completed successfully
```

### 3.5.18 Troubleshooting

Problem	Cause	Solution
You receive intermittent <b>Permission denied</b> errors for shared RWM pvc with trident storage class which causes deployment to fail:  <pre>cp: cannot create regular file '/customscripts./d2/ dfs-sdk-21.4/samples/ security/webseal/ CookieSettingWcfBehavior.cs' : Permission denied cp: cannot create regular file '/customscripts./d2/ dfs-sdk-21.4/samples/ security/webseal/ WebsealIVUserHandler.java': Permission denied cp: cannot create regular file '/customscripts./d2/ dfs-sdk-21.4.tar.gz': Permission denied cp: cannot create regular file '/customscripts./d2/ D2AutoInstall.xml': Permission denied cp: cannot create regular file '/customscripts./d2/ configure-d2.sh': Permission denied</pre>	Pods start as dmadmin with UID 1000 (non-root user). You must set the UID for the PVC to 1000 to avoid permission denied issues.	Configure the security context for the pod which is creating errors, so that the UID in <i>runAsUser</i> and <i>gid</i> in <i>fsGroup</i> are 1000 in the statefulset/pod/deployment template yaml file under the templates directory of the respective sub-chart as follows:  <pre>spec:   securityContext:     runAsUser: 1000     fsGroup: 1000</pre>

### 3.5.19 Documentum 23.4 Multi-Repository support

#### 3.5.19.1 Overview of high-level steps to support Multi-Repository for cloud deployments

For multi-repository setup, the deployment steps are as follows:

1. Deploy Documentum single Helm chart with required clients and other components by following the 23.2 build book on your chosen hyperscaler. The docbase configured in this deployment is considered as a global repository and the deployment is considered as a global repository deployment.
2. For each additional repository, deploy Documentum single Helm with only Content-Server and related components (cs-secrets, cs-logging-configMap, content-server, cs-dfc-properties, dctm-ingress, xda, bps, and Xplore) enabled. Ensure that all additional repositories are configured to the single docbroker that was deployed in Step 1.

And as a best practice, deploy additional repositories in separate namespaces under the same cluster.

**!** **Important**

All the below steps, including the pre-requisites, have to be repeated for each additional repository deployment.

### 3.5.19.2 Pre-requisites

1. Documentum single Helm deployment with all required clients and components enabled on the chosen hyperscaler are performed by following the 23.4 build book. Ensure all replicas of Content-Server and docbroker pods are in running state. The docbase configured in this deployment is considered as global a repository and the deployment is considered as a global repository deployment.
2. The global repository deployed cluster will be considered for all additional repository deployments.
3. The install owner username and password should be the same as the global repository for all the additional repositories. This should be the same for all docbases.

```
installOwnerUsername: &installOwner_username dmadmin
installOwnerPassword: &installOwner_password password
```

4. Create a new namespace for each additional repository deployment under the same cluster (where the global repository is deployed).
5. Copy the AEK key (aekey\_name) from the Content-Server pod (dcs-pg) of the global repository from the location /opt/dctm/dba/secure/aekey\_name. This can be done by replacing the <global repository namespace> with the namespace of the global repository deployment and the <aekey.name provided in the global repository deployment> with the dctm-server.content-server contentserver.aekey.name provided in the global repository deployment in the below command and then running the command.
 

```
kubectl cp <global repository namespace>/dcs-pg-0:/opt/dctm/dba/secure/<aekey.name provided in global repository deployment> aekey_name -c dcs-pg
```
6. Create a PVC in the new namespace where the additional repository will be deployed by following the steps below:
  - a. Create a yaml file with the below details. Replace <serviceName of the content server pod in the global repository deployment> with the serviceName of the Content-Server pod in the global repository deployment:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: <serviceName of the content server pod in global repository deployment>-sharedkey-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 20Mi
  storageClassName: trident-nfs
  volumeMode: Filesystem
status:
  accessModes:
    - ReadWriteMany
  capacity:
```

```
storage: 20Mi
phase: Bound
```

- b. Create the PVC in the new namespace by running the below command. Replace <additional repository namespace> with the additional repository namespace.

```
kubectl apply -f ./pvc.yaml -n <additional repository namespace>
```

7. To copy the AEK copied from the global repository deployment to the newly created PVC, follow the below steps:

- a. Create a Docker file with below details and update <install owner> with the install owner of the global repository deployment:

```
FROM <alpine image>
ARG AEKPATH
ENV AEK_KEY_PATH /aekeypath
RUN set -ex && apk --no-cache add sudo
RUN adduser -D -H <install owner> &&\ 
    mkdir -p $AEK_KEY_PATH && \
    chown -R <install owner>:<install owner> $AEK_KEY_PATH
COPY --chown=<install owner>:<install owner> $AEKPATH$AEK_KEY_PATH
CMD sh
```

- b. Create a Docker image using the Docker file created in the previous step by running the command below:

```
docker build -f <path to docker file created in previous step> -t <docker image name> --build-arg AEKPATH=<relative path to folder that contains aek key copied from global repo> --no-cache .
```

Example:

```
docker build -f Dockerfile -t artifactory.oxlab.net/ot2-paas-dev/copyaekkey:23.4 --build-arg AEKPATH=./folder --no-cache .
```

- c. Push the created Docker image to the preferred registry using the below command:

```
docker push <docker image name>
```

8. The docbroker replicaCount mentioned under the content-server section of the additional repository deployment's resource yaml file should be the same as that of the docbroker replica count mentioned under the content-server section of the global repository deployment's resource yaml file.

**Chosen resource yaml file:**

```
dctm-server:
  content-server:
    contentserver:
      replicaCount: 1
      docbrokersCount: 1
```

### 3.5.19.3 Additional pre-requisites for deploying an additional repository on AWS

- If you are using static provisioning, create two access points with default values in the EFS for the following PVCs. You can ignore this step in the case of dynamic provisioning. AWS documentation contains detailed information.

 **Note:** This EFS can be the same as the EFS used/created for global repository deployment.

- PVC for the Documentum Server in the EFS for Dctm-server and xplore.

### 3.5.19.4 Additional pre-requisites for deploying additional repository on Azure

Configure additional nginx-ingress so the applications are accessible externally.

#### 3.5.19.4.1 Create a public IP address on Azure

As a best practice, create a static IP address.

Run the following command to create a static public IP address on Azure:

```
az network public-ip create --resource-group <resource group> --name <name of public IP>
--sku Standard --allocation-method static --query publicIp.ipAddress -o tsv
```

Use the following command to deploy the additional ingress controller:

 **Important**

Ensure that the ingress class name used for creating the additional ingress controller is unique in relation to the existing ingress class names.

```
helm install <name of ingress resource> ingress-nginx/ingress-nginx --namespace
<namespace>
--set controller.replicaCount=1
--set controller.ingressClass=<ingress class name>
--set controller.ingressClassResource.name=<ingress class name>
--set controller.ingressClassResource.controllerValue="k8s.io/<ingress class name>"
--set controller.ingressClassResource.enabled=true
--set controller.IngressClassName=true
--set controller.service.loadBalancerIP="
```

Verify that the nginx-ingress has been deployed successfully and the service has been created.

Configure a fully-qualified domain name (FQDN) for the ingress controller IP address. You can run the following Bash script to map the external IP to the DNS name or run the command line-by-line in the Windows environment.



**Note:** Provide the IP and the DNS name details. You can find the public IP address from kubectl get svc when the Helm install ingress command is run. DNSNAME should be unique in relation to the existing DNS names.

```

#!/bin/bash
# Public IP address of your additional ingress controller created in previous step
IP="52.188.221.96"
# Name to associate with public IP address
DNSNAME="documentumdeployment"
# Get the resource-id of the public ip
PUBLICIPID=$(az network public-ip list --query "[?ipAddress!=null] | [?
contains(ipAddress, '$IP')].[id]" --output tsv)
# Update public ip address with DNS name
az network public-ip update --ids $PUBLICIPID --dns-name $DNSNAME

```

### 3.5.19.5 Updates to be done in Documentum single Helm chart

#### 3.5.19.5.1 Documentum/Documentum-components.yaml file changes

- Disable all components except cs-secrets, cs-logging-configMap, content-server, cs-dfc-properties, dctm-ingress, xda, bps, and Xplore.

#### 3.5.19.5.2 Documentum/values.yaml file changes to support additional repositories

##### General updates

1. Replace all instances of <namespace> with the namespace where the additional deployment is going to be deployed.
2. Replace all instances of <docbase\_name> with the additional docabse name.

##### Updates to anchor tag section

1. Update rwo/rwmStorage class as per the storage class being used.
2. Update install owner username and install owner password with install owner username and install owner password of the global repository deployment.
3. Update ingressUrl: &ingressUrl with the FQDN of the ingress.
4. Update ingressDomain: &ingress\_domain with the ingress domain name used in the deployment.
5. Update otdsAuthSvc: &otds\_auth\_svc with the FQDN of ingress>/otdsaws.
6. Update globalRegistryPassword: &global\_registry\_password with the global registry password.
7. Update otdsEnabled: &otds\_enabled to true if OTDS is enabled in the global repository deployment.
8. To enable SSL mode in the additional deployment, update useCertificate: &use\_certificate to true in the anchor tag section.



**Note:** Both the global repository deployment and additional repository deployments should be SSL-enabled or disabled as mixed mode is not supported.

For example:

```
rwoStorage: &rwo_storage_class azurefile
rwmStorage: &rwm_storage_class azurefile
ingressUrl: &ingressUrl namespace2.australiaeast.cloudapp.azure.com/
otdsAuthSvc: &otds_auth_svc namespace2.australiaeast.cloudapp.azure.com/otdsaws
ingressDomain: &ingress_domain australiaeast.cloudapp.azure.com
useCertificate: &use_certificate false
globalRegistryPassword: &global_registry_password Password@1234567890
otdsEnabled: &otds_enabled true
```

### Updates to cs-secrets section

1. The secrets file contains sensitive information like license keys and passwords. Follow the below steps to configure the Documentum Server in SSL mode. In the docbroker certificate section, give the password, aekpassphrase and trustpassword mentioned in the global repository deployment. Provide valid certificate key and certificate in Pem format for pemCertPrivKey and pemCertificate parameters. pemCertPrivKey and pemCertificate parameters can be left blank for the self-signed key.

```
dctm-server:
  cs-secrets:
    docbroker:
      certificate:
        password:
        aekpassphrase:
        trustpassword:
        pemCertPrivKey:
        pemCertificate:
```

2. Similarly in the content-server certificate section, update the password and trust password and provide the certificate and certificate key in Pem format. pemCertPrivKey and pemCertificate parameters can be left blank for the self-signed key.

```
dctm-server:
  cs-secrets:
    contentserver:
      certificate:
        password:
        trustpassword:
        pemCertPrivKey:
        pemCertificate:
```

3. Update the database section with the database username and password and certificate.



**Note:** The database for the additional repository deployment doesn't have to be the same as the global repository deployment.

For example:

Database section under cs-secrets:

```
dctm-server:
  cs-secrets:
    database:
      userName: database-username
      password: database-password
      certificate: |
      -----BEGIN CERTIFICATE-----
```

```
database-certificate
-----END CERTIFICATE-----
```



**Note:** If the provided GCP certificate is valid, the storage bucket with the given name will be created as part of the CS scripts. There's no need to manually create the bucket in the Google cloud store.

4. If TLS for dcm-ingress needs to be enabled for additional repository deployment, provide the base64-encoded TLS certificate and the TLS key in the cs-secrets section as follows:

```
dctm-server:
  cs-secrets:
    ingress:
      tlscrt: <base64-encoded TLS certificate> as one line i.e. without any
              carriage returns or wrapped lines.
      tlskey: <base64-encoded TLS key> as one line i.e. without any carriage
              returns or wrapped lines.
```

#### Updates to cs-dfc-properties parameters under dctm-server section

- Update the env:domain to point to the env\_domain of the global repository deployment. For example:

Global repository values:

```
env: &env_domain namespace1.svc.cluster.local
dctm-server:
  cs-dfc-properties:
    enabled: true
    ### Environment ####
  env:
    domain: *env_domain
```

Additional repository values:

```
nv: &env_domain namespace2.svc.cluster.local
dctm-server:
  cs-dfc-properties:
    enabled: true
    ### Environment ####
  env:
    domain: namespace1.svc.cluster.local
```

#### Updates to content-server parameters under dctm-server section

1. Update the docbroker cluster space under the content-server section to point to the global repository deployment's docbroker cluster space, as shown in the code block example below:

Global repository values:

```
env: &env_domain namespace1.svc.cluster.local
dctm-server:
  content-server:
    enabled: true
    docbroker:
      serviceName: *dbr_service_name
      clusterSpace: *env_domain
```

Additional repository values:

```
env: &env_domain namespace2.svc.cluster.local
dctm-server:
```

```

content-server:
  enabled: true
  docbroker:
    serviceName: *dbr_service_name
    clusterSpace: namespace1.svc.cluster.local #env_domain of existing
deployment

```

2. Update docbase parameters under the content-server section, as shown in the code block example below.

 **Note:** The id used for the additional docbase should be unique in relation to the existing docbases.

Global repository values:

```

content-server:
  enabled: true
  docbase:
    id: 123456
    index: DM_docbase1_DOCBASE

```

Additional repository values:

```

content-server:
  enabled: true
  docbase:
    id: 123457
    index: DM_docbase2_DOCBASE

```

3. Update the database parameters under the content-server section to point to the preferred database that includes updates to host, databaseServiceName, port, sslEnabled, paasEnv, and docbaseOwnerPasswordChange. In the illustrated example, the database of the additional repository deployment is pointing to the database of the global repository deployment.

Global repository values:

```

content-server:
  ### Database ###
  database:
    host: db-pg-0.db-pg.namespace1.svc.cluster.local
    databaseServiceName: MyPostgres
    port: 5432
    sslEnabled: false
    paasEnv: false
    docbaseOwnerPasswordChange: false

```

Additional repository values:

```

content-server:
  ### Database ###
  database:
    host: db-pg-0.db-pg.namespace1.svc.cluster.local
    databaseServiceName: MyPostgres
    port: 5432
    sslEnabled: false
    paasEnv: false
    docbaseOwnerPasswordChange: false

```

4. Update the globalRepositoryName under the content-server section with the global repository deployment's docbase name.

 **Note:** Make sure the global repository mentioned is up and running during the additional docbase deployment.

Global repository values:

```
content-server:
  globalRepository:
    globalRepositoryName: ""
```

Additional repository values:

```
content-server:
  globalRepository:
    globalRepositoryName: docbase1
```

- Set disableUpdateAcsUrl to true under the ingress sub-section of the content-server section, and update the ingress:host as per your environment.

```
dctm-server:
  content-server:
    ingress:
      host: dctmint-ingress.d2.cfcr-lab.bp-paas.otxlab.net
      disableUpdateAcsUrl: false
```



**Note:** The above parameter is set to true to avoid overwriting of acs\_base\_url when the pod restarts so that the user can manually update the required acs\_base\_url and preserve it.

- Update the persistentVolume.shareKeyPVCName under the content-server section to point to <serviceName of the content server pod in global repository deployment>-sharedkey-pvc.

Global repository values:

```
dctm-server:
  content-server:
    persistentVolume:
      shareKeyPVCName: ""
```

Additional repository values:

```
dctm-server:
  content-server:
    persistentVolume:
      shareKeyPVCName: "dcs-pg-sharedkey-pvc"
```

- Make sure the extraEnv section has the following values set to T. Also update the <docbase\_name> placeholder to the docbase name.

```
extraEnv:
  - name: LSS_CC_ENABLED
    value: "T"
  - name: DA_PRIVILEGE_ENABLED
    value: "T"
  - name: <docbase_name>_resource_id
    value: ""
  - name: <docbase_name>_secretKey
    value: ""
  - name: <docbase_name>_MIGRATE_LDAP_CONFIGS
    value: ""
  - name: MIGRATE_LDAP_DOCBASES
    value: "<docbase_name>"
```

8. Update `otds.otdsAPISvc` under the `content-server` section to point to `<service name of otds in global repository deployment>. <namespace of global repository namespace>:<otds service port>/otdsws` and update `cert_jwks_url` in the `content-server` section to point to `http://<service name of otds in global repository deployment>. <namespace of global repository namespace>:<otds service port>/otdsws/oauth2/jwks`.



**Note:** The service name of OTDS in the global repository deployment can be found in `documentum/values.yaml` of the global repository deployment in key: `otds.otdsws.serviceName`, and `<otds service port>` can be found in key: `otds.otdsws.port`.

(Optional) Update `updateOTDScertonrestart` to `true` to update OTDS secret on Content-Server pod restart.

Global repository values:

```
content-server:  
  otds:  
    otdsAPISvc: otdsaws:80/otdsaws  
    updateOTDScertonrestart: false  
    auto_cert_refresh: true  
    cert_jwks_url: http://otdsaws:80/otdsaws/oauth2/jwks
```

Additional repository values:

```
content-server:  
  otds:  
    otdsAPISvc: otdsaws.namespace1:80/otdsaws  
    updateOTDScertonrestart: true  
    auto_cert_refresh: true  
    cert_jwks_url: http://otdsaws.namespace1:80/otdsaws/oauth2/jwks
```

9. If additional repository is deployed on AWS, provide the following information if you are using static provisioning (leave these values to default in case of dynamic provisioning):

- `existVolumePv`: Provide a unique user-defined name for PV. For example: `dctmcspv`.
- `awsEFS`: Set the value to `true`.
- `awsEFSCSIDriver`: Retain the default value (`efs.csi.aws.com`).
- `awsEFSCSIHandle`: The format is EFS file system ID1::EFS access point ID2. For example: `fs-1fc8901b::fsap-0e45ed0c8888fcd34`.

### Externalizing content server

- The below configuration section will enable VM-based deployment outside the cluster to connect to Content-Server. Make sure `externalAccessEnabled` is set to `true` in the anchor tags section of `documentum/values.yaml`.

```
externalAccessEnabled: &external_access_enable true
```

- a. For GCP deployment, in the `ExtCS` section, set `nativeExtPort` as `80` and `sslExtPort` as `81`. Keep the other default values as is.

```
ExtCS:
  tcp_route: 10.0.0.0
  nativeExtPort: 80
  sslExtPort: 81
  extDbrPort: 1491
```

If the GCP cluster is internal or if you want to use only the internal Loadbalancer for ExtCS/ExtDocbroker, the Loadbalancer should be created internally. Please set the below values in both the Docbroker and Content-Server section to create the internal Load Balancer service.

```
useLBAnnotations: true
LBAnnotations:
  networking.gke.io/load-balancer-type: "Internal"
```

- b. Once the Content-Server is deployed, it can be accessed externally as below.

### Updates to dctm-ingress

1. Disable all services except jmsService, acsService, bpm, dsearchadminService, and indexagentService.
2. Under `dctm-ingress`, update `ingressPrefix` (prefix for the ingress name) and `ingress:host`.

```
dctm-server:
  dctm-ingress:
    #prefix for the ingress name
    ingressPrefix: dctm
    ingress:
      host: dctm-ingress
```

3. To enable TLS for `dctm-ingress`, update `tls.enabled` to `true`.

```
dctm-server:
  dctm-ingress:
    tls:
      enable: true
      secretName: *cs_secret_name
```



**Note:** While deploying additional repositories on Azure, make sure the ingress class is updated to the desired value and different from the global repository deployment in `documentum/platforms/azure.yaml`.

`d2/platforms/azure.yaml`

```
dctm-server:
  dctm-ingress:
    ingress:
      class: nginx2
```

### Configuring object or content store for hyperscalers

1. For AWS:

1. If S3 store is enabled, provide the object storage details under `s3Store` under the `cs-secrets` section.

```
dctm-server:
  cs-secrets:
```

```
s3Store:
  s3StoreBaseUrl: Base_URL/bucket_name
  s3StoreCredentialID: <credentialid>
  s3StoreCredentialKEY: <credentialkey>
```

2. To enable S3 store, update the `s3Store` section under the `content-server` section with `enable` as `true`. Also, set `default` as `false` to make S3 store as the non-default store. In `name`, provide the name of the store which you want to create in the S3 store. This is a user-defined field, which means the user can give any name which is not already in the S3 store. If the store configured is a public store such as Amazon S3, it is mandatory to update `proxyHost` and `proxyPort` details corresponding to the underlying S3 store. However, if the store configured is private such as NetApp, do not pass any value for these fields. Please keep the default values for `proxyProtocol` and `noProxy`. `isworm` is applicable only for IBM store, therefore while configuring other stores keep the default values starting from `isworm` as given below:

```
dctm-server:
  content-server:
    s3Store:
      enable: true
      default: false
      name: <s3storename>
      proxyHost: gcp-prox01-1001.oxlab.net
      proxyPort: 3128
      proxyProtocol: http
      noProxy: localhost,127.0.0.1,*.oxlab.net
      isworm: false
      vendor:
      region:
      enable_md5: true
      enable_v4signing: true
```



**Note:** If you are using a private s3store created in AWS, set `proxyHost` and `proxyPort` as follows:

```
proxyHost: noproxy
proxyPort: noproxy
```

Once the deployment is done and all the client products are configured successfully, only then the `s3store` has to be set as the default store using the steps mentioned in the post deployment steps.

## 2. For Azure:

1. If Azure BLOB store is enabled, provide the object storage details under `restStore` in the `cs-secrets` section.

```
### Rest Store ###
restStore:
  restStoreBaseUrl: https://dctmreststoredev.blob.core.windows.net/autotest1
  restStoreCredentialID:
  restStoreCredentialKEY:
```



**Note:** `restStoreBaseUrl: https://dctmreststoredev.blob.core.windows.net/autotest1` is the BLOB storage name.

2. To enable Azure BLOB store, update the restStore section with enable to true as shown below and provide name, proxyHost and proxyPort as per Azure cluster under the content-server section.

```
dctm-server:
  content-server:
    ### Rest store enable ####
    restStore:
      enable: true
      name: autotest1
      restStoreType: 0
      proxyHost: <proxy host>
      proxyPort: <proxy port>
      proxyProtocol: http
      noProxy: localhost,127.0.0.1,*.otxlab.net
```

3. For GCP:

1. If GCP store is enabled, get the GCP Store credentials from GCP Console as follows:
  - a. Click on **IAM & Admin** from the left side options, select **service accounts**, click the three dots of a service account and select **Manage keys**, then create a new key by selecting **Add key option in JSON format**.
  - b. Copy the GCP Store credentials from the downloaded key and populate under the gcpStore\_credentials section.

```
### GCP Store credentials ####
gcpStore_credentials: |
{
  "type": "service_account",
  "project_id": "ot1-eng-ecd-dctm-CS",
  "private_key_id": "c8c5a09edac691d3d115b5e773ba77e99b87ac17",
  "private_key": "-----BEGIN PRIVATE KEY-----\nMIIEvQIBADANBgkqhkiG9wOBAQEFAASCBKcgwgSjAgEAAoIBAQCVp4B4AiWahuVW\nnsMF82V8WF6VLiCdHSv5TDJrY71MaDYKb/b6qELfLcLvWtNudn7yE4NLL5g1ili\nnxRqPAeyAkRQqUhwoDEV7bHUq7qb8+fKVRYZT2Nax/\nFzuVtaUEMDy69HYwSGKRdd7\nn3wt51wJWIBoNn1jpEGiwTU5IPKi6P00BIqkbPaMvK3233Q6Me0\n40hwS1lpQDyvZd\noFLqXr0Ac7dZ0cYkQeJw\n+gLIiwtsBNDnXqoQNbjXBzhAs9f7WuwTw8+m9MJN3\nnyyvgjJLPE3Z7w9VWmUvZLOHVL5vcN\nZC1ClvikXhlfMNfzmaNdZ4DTPcc0OfidxCm\n\nnHSN7JiJrAgMBAAECggEARI3dpRga2ubqWbvBU/qg10eIHnz9KIkCyImDK710E1K\nnt2T1czaB0lbnas8m8w5NqJrzgGT08PTWrzqH81L4BtgvRFzzELIp3iM+Sd0kX+j\nn7atN2dz1zKbHoc/15oaMK0jYpZkDXg2zFE7zwUx98kFCBEusbVmjkfJ2md/\n6T7/nlcQFAbj8fjeYwbC1g0MSMveoa08P4Y7FiGcyno3/PoY/4iR6ewuBnBeB+6T1f/1Q\nnePvg511iXegEhZhuEc0lRmBSGoWb3wAB1vzyY/\nK6kGpvGBtlLiT3FAAnTgfrpDu975\nndq8inPpVn\n+Z14CxnbH105AU33T5sSHN1ZZJH2T8poQKBgQDmE00V1Czr2HTgpGML\nnMy\n+a7t4TeVhznbGcp0gHcArLw445jmh/x/UoWpmEyjdJUNRQR8H1tYFsgkadgsx\nnRtz7Pyv0rpMREz3a1+zBooZ1FKCAr6uJWcfbPdlfnU7Zqv0XfTy6pXyObGouBJrc\nnGYi6xKck6m/yU9fL08CbaGZNewKbgQDC/\nrIu9eUqvWm6oScruPyvqR0L0ots0L9/ntbP6uSkpHXE5sGgZrg07bBrnD0AZG3TQ2qbhZEgm5\nHoq9+Cbau5hbYxuOPPYaa\n\nnrYyRo8u6fJGKdPymfcx0XoAsiaLmVFBuYKotmDhyONLjQ1MSxuWQshwdLRx0dZdh\n\nkyqZ60ZT00KBgCnSbKGfCr9gXHaNSze4+T1XnGS71R1HHAJc\n+BnqAvxOSz8pJN09/nfFH2qHFfn++S0tU6ucI+Z\n+8UMy1tMcGmV8yVg0mZkEA1WDQuTNPVfstRFi+H302b\nnVv0EFFxx\n+WhVzaXcbRKcjFSzXmW5DpFdzt3sa1mph+nri1b1GZ6LeYgJLaoGAGxER\n\nmlLUn1SNfYtrDWiHgfrzLKBwGHHDcKQFghnrz5X/iL/gDkJuyrZXSnfYvxnTapc\nnnG9g3yLvDzp0Pv2fd41c9d/rkwx/7i6S6ZBr8hnide6hN1cU7z5c2mvrlhG6Wp3z\nnCVss0qYS19sX/u4/zF18y4v8du0Y1ccAzbZnS1EcgYEAp20g5NKJDYwfa92pUanK\nnebmR4v1Dt0CV4BCs45D+d1kYpb9BoP9wDQpxNw7p7vCSrFhSOpCCn4faPd27k3W0\nnX9Bq8aV4+TQgZ8s68DN/HP93en2mYu07NBeGfWNex51iP4BLdOaIEpnJINxYVi0V/n8VfjA//\nhGz3dCT2B8N/6r8s=\n-----END PRIVATE KEY-----\n",
```

```

"client_email": "138701173802-compute@developer.gserviceaccount.com",
"client_id": "117247930302497494342",
"auth_uri": "https://accounts.google.com/o/oauth2/auth",
"token_uri": "https://oauth2.googleapis.com/token",
"auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
"client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/138701173802-compute%40developer.gserviceaccount.com"
}

```

2. If GCP store is enabled, update the object storage details under restStore in the cs-secrets section.

```

### Rest Store ###
restStore:
  restStoreBaseUrl: https://storage.googleapis.com/autotest1
  restStoreCredentialID:
  restStoreCredentialKEY:
## rest Store ssl certificate, below value is for (gcp blob store) change
accordnigly ###
gpcertificate:

```

3. restStoreBaseUrl: https://storage.googleapis.com/autotest1. Here autotest1 is assumed to be the cloud storage bucket name. Update autotest1 with the bucket name. Whatever bucket name is given will be created as part of the Content-Server scripts. No need to manually create the bucket in the Google cloud store.
4. Provide the valid gpcertificate. It is as per GCP design that the public certificate expires every 90 days. Once the GCP team releases a new certificate, the old certificate stops working even though its validity has still not expired. So we have to fetch a new one using the Java keytool utility as shown below and updating the gpcertificate parameter.

```
keytool -printcert -J-Dhttps.proxyHost=<proxy-host> -J-Dhttps.proxyPort=<proxy-port> -rfc -sslServer storage.googleapis.com:443
```



**Note:** If the provided GCP certificate is valid, the storage bucket with the given name will be created as part of the Content-Server scripts. No need to manually create the bucket in the Google cloud store.

5. To enable GCP store, set enable to true as shown below and provide name, proxyHost, and proxyPort as per GCP cluster and set restStoreType to 1 under the content-server section.

```

dctm-server:
  content-server:
    ### Rest store enable ###
    restStore:
      enable: true
      name: autotest1
      restStoreType: 1
      proxyHost: gcp-prox01-1001.otxlab.net
      proxyPort: 3128
      proxyProtocol: http
      noProxy: localhost,127.0.0.1,*.otxlab.net

```

6. To enable GCP store, set enable to true as shown below.

```

dctm-server:
  content-server:
    ### GCP store enable ###

```

```
gcpStore:
  enable: true
```

### 3.5.19.5.3 Changes to dockerimages-values.yaml

1. Copy anchor-tag section, content-server section, xplore section, xda section and bps section from global repository deployment to additional repository deployment section.
2. Append the below details under the init container section of content-server under dctm-server for additional repository deployment. Replace <docker image created in step 7 of prerequisites> with the Docker image created in step 7 of Prerequisites:

```
- name: aekkeycopy
  image: <docker image created in step 7 of prerequisites>
  imagePullPolicy: *pull_policy_type
  command: ['/bin/sh', '-c', 'yes |sudo cp -rf /aekeypath/* /opt/dctm/shared_key']
  volumeMounts:
    - name: dcs-pg-sharedkey-pvc
      mountPath: /opt/dctm/shared_key
```

Example:

```
dctm-server:
  content-server:
    extraInitContainers:
      - name: aekkeycopy
        image: artifactory.oxlab.net/dockerhub/copyaekey:23.4
        imagePullPolicy: *pull_policy_type
        command: ['/bin/sh', '-c', 'yes |sudo cp -rf /aekeypath/* /opt/dctm/shared_key']
        volumeMounts:
          - name: dcs-pg-sharedkey-pvc
            mountPath: /opt/dctm/shared_key
```

### 3.5.19.5.4 Changes to platforms/<platform>.yaml

1. If you are on GCP or Azure, update ingress.class with the ingress class name deployed in Section 3.5.19.3.1. Create a public IP address on Azure for dctm-ingress in platforms/<platform>.yaml.
2. Applicable for AWS only: When deploying additional repository on AWS, 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 a comma. For example:

```
alb.ingress.kubernetes.io/subnets: subnet-0c4a1017a4ffb7962,
subnet-0bbd3ec1319a30772, subnet-0a6ab032a5e03be49
```

- (Optional) If you want to use ingress in secure connection mode, perform the following steps:
  - i. 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.
  - ii. 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> --
privatekey
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"
}}
```

- iii. Open the aws.yaml file in the extracted platforms folder and perform the following steps for the ingress category dctm-ingress:

1. In the alb.intress.kubernetes.io/subnets annotation, provide the clusters' public subnet IDs separated by a comma. For example:

```
alb.ingress.kubernetes.io/subnets: subnet-0c4a1017a4ffb7962,
subnet-0bbd3ec1319a30772, subnet-0a6ab032a5e03be49
```

2. Comment the following annotation that is provided for using ingress in the non-secure connection mode:

```
alb.ingress.kubernetes.io/listen-ports: '[{"HTTP":80}]'
```



**Note:** For the dctm-ingress category, listen ports will be both http and https as follows. This is a requirement for xPlore:

```
alb.ingress.kubernetes.io/listen-ports: '[{"HTTP":80}, {"HTTPS":443}]'
```

3. 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/asbselfsigned
```

4. Update the value of alb.ingress.kubernetes.io/group.name to a desired value. For example:

```
alb.ingress.kubernetes.io/group.name: dctm-internal-group-2
```



**Note:** The value chosen as group.name should be unique in relation to the group.name used for global repository deployment and additional repository deployments.

5. Update the value of ARN obtained from Step 9.b in the uncommented annotation. For example:

```
alb.ingress.kubernetes.io/certificate-arn: arn:aws:iam::<AWS account
ID>:server-certificate/testcertificate
```



**Note:** If necessary, update the AWS ingress annotation values in the documentum/platforms/aws.yaml file according to your platform's ingress.

### 3.5.19.6 Deploy additional repository

- Run the following command before deployment to validate the yaml files.

```
cd <documentum master chart directory>
helm install <documentumdeployment_name> . --values .\platforms\<platform_yaml> --values .\dockerimages-values.yaml --values .\documentum-components.yaml --values <resources_yaml_file>.yaml -n <namespace> --debug --dry-run
```



**Note:** The above command should not return any error.

- Run the following Helm install command to deploy the Helm charts.



**Note:** Based on customer deployment size (small , medium , large , or extra), select the resource values file from the Documentum master directory and use it in the deployment command.

```
helm install <documentumdeployment_name> . --values .\platforms\<platform_yaml> --values .\dockerimages-values.yaml --values .\documentum-components.yaml --values <resources_yaml_file>.yaml -n <namespace>
```

### 3.5.19.7 Post-deployment steps

- In global repository deployment, we need to register all D2 webapps and their corresponding replicas (d2config, d2classic, d2smartview and d2rest ) as privileged clients in all the docbases in order to use OTDS authentication. This can be done by following the below steps.

- Execute into D2 webapp using the kubectl exec command.

- Run the privileged client utility for all the docbases by updating the <DOCBASE\_NAME> placeholder. Steps to perform:

```
kubectl exec -ti <d2 webapp>-0 -n <namespace> -- bash
```

```
java -cp $CATALINA_HOME/CustomConf:$CATALINA_HOME/webapps/<webapp folder>/WEB-INF/lib/*:$CATALINA_HOME/webapps/<webapp folder>/utils/d2privilegedclient/* com.opentext.d2.util.D2PrivilegedClientUtil -d <DOCBASE_NAME> -u $INSTALL_OWNER -p $INSTALL_OWNER_PASSWORD
```

- Repeat the same for all webapps and docbases.

- Follow the steps below to export the login ticket from global repository deployment to additional repository deployments.

- Firstly exec into the content-server pod of global repository deployment using the below command and replace <global repository deployment namespace> with global repository deployment.

```
kubectl exec -ti dcs-pg-0 -c dcs-pg -n <global repository deployment namespace> -- bash
```

- Start an iapi session by running the command. Replace <global docbase name> with the global docbase name:

```
iapi < global docbase name>
```

- Execute the below commands in iapi.

```

- API>
apply,c,NULL,EXPORT_TICKET_KEY,PASSWORD,S,<password>
...
q0
API> ?,c,q0
result
<key_string>:
```

- d. Copy the ticket starting from DM\_ENCR\_TEXT\_V2= to end of ticket string.
3. Import login ticket from global repository to additional repository.
  - a. Execute into the Content-Server pod of global repository deployment using the below command. Replace <additional repository deployment namespace> with additional repository deployment.
 

```
kubectl exec -ti dcs-pg-0 -c dcs-pg -n <additional repository deployment namespace> -- bash
```
  - b. Start an iapi session by running the command. Replace <additional docbase name> with additional docbase name:
 

```
iapi <additional docbase name>
```
  - c. Execute the below commands in iapi:
 

```

- API>
apply,c,NULL,IMPORT_TICKET_KEY,KEY_STRING,S,<ticket copied in
previous step>,PASSWORD,S,password
...
q0
API> ?,c,q0
result
-----
T
(1 row affected)
```
4. Restart the additional docbase by running the below commands in order. Replace <additional docbase name> with the additional docbase name:
 

```
/opt/dctm/dba/dm_shutdown_<additional docbase name>
/opt/dctm/dba/dm_start_<additional docbase name>
```



**Note:** Repeat points 2, 3, and 4 under Section 3.5.19.7 in order for all additional repositories if login ticket is changed in global repository deployment post performing post deployment steps on additional repository.

5. Once Content-Server is deployed for the first time, Content-Server can be accessed by external clients (outside the cluster) as below:
  - a. Content-Server should be deployed with externalization configuration as mentioned above. After the initial deployment, there will be one load balancer service created for csext with an External IP.
  - b. Copy the external IP and update tcp\_route with the copied external IP and do not give any values for nativeExtPort or sslExtPort.

```

ExtCS:
  tcp_route: 10.9.56.136
  nativeExtPort:
```

```
sslExtPort:  
extDbrPort: 1491
```

- c. Upgrade using the below mentioned Helm command.

```
cd <Documentum master chart directory>  
helm upgrade <documentumdeployment_name> . --values .\platforms  
\<platform_yaml> --values ./dockerimages-values.yaml --values documentum-  
components.yaml --values <resources_yaml_file>.yaml -n <namespace>
```

 **Note:** If AEK key of global repository is changed post additional repository deployment, repeat steps 5 and 7 under Section 3.5.19.2 in order and then step 2 under Section 3.5.19.5.3.

Make sure to use a different name for the Docker image created in step 7.b if pullpolicy for content-server images is IfNotPresent.

Perform Helm upgrade on additional repository deployment with the below command:

```
cd <Documentum master chart directory>  
helm upgrade <documentumdeployment_name> . --values .\platforms  
\<platform_yaml> --values ./dockerimages-values.yaml --values documentum-  
components.yaml --values <resources_yaml_file>.yaml -n <namespace>
```

Repeat these steps for all additional repository deployments.

6. Please refer to point 4 under Section 3.5.5.1. If this step is done for current additional repository, restart D2 Config pod in global repository deployment post additional repository deployment. This will do custom config import for other docbases. Otherwise edit the global repository deployment helm charts as per point 4 under Section 3.5.5.1 and perform Helm upgrade on global repository deployment post helm upgrade. Restart D2 Config pod in global repository deployment.

#### 3.5.19.7.1 Additional post-deployment steps when additional repository is deployed on AWS

1. Obtain the ingress deployment addresses using the following command format:

```
$ kubectl get ingress  
  
Example output with address:  
  
NAME          CLASS  
HOSTS  
ADDRESS  
dctm-internal-ingress    <none>  
*           k8s-  
d2newdctmgroup-440a21863a-1248970307.us-east-1.elb.amazonaws.com
```

2. Obtain the External IP of the csext LoadBalancer service using the following command format:

```
kubectl get svc -n <namespace>  
  
NAME      TYPE      CLUSTER-IP      EXTERNAL-  
IP  
PORT(S)      AGE  
csext-dcs-pg  LoadBalancer  10.100.197.39  
a4652670631244318837d475c5b7e8c5-30432636.us-east-1.elb.amazonaws.com  80:30650/  
TCP,81:32322/TCP      10d
```

```
dbrex-dbr LoadBalancer 10.100.137.236
aaf2a87c480164c9cb3dd2f3167a23ff-835178087.us-east-1.elb.amazonaws.com 80:32478/
TCP,81:30552/TCP 10d
```

- After you obtain the addresses from the ingress deployment, update the dctm-ingress host values wherever it is present in documentum/values.yaml with the value of corresponding ADDRESS from Step 1 of this section. Make sure you have updated the following anchor tags in documentum/values.yaml accordingly. For example:

```
otdsAuthSvc: &otds_auth_svc k8s-d2newdctmgroup-440a21863a-1248970307.us-
east-1.elb.amazonaws.com/otds
ingressDomain: &ingress_domain us-east-1.elb.amazonaws.com
```

- Update the tcp\_route parameter under the ExtCS subsection in the content-server section in documentum/values.yaml with the value of the external IP of the csext LoadBalancer service.

```
Extcs:
  tcp_route: a4652670631244318837d475c5b7e8c5-30432636.us-east-1.elb.amazonaws.com
  nativeExtPort: 80
  sslExtPort: 81
  extDbrPort: 1491
```

- Run the Helm upgrade command using the following command format:

```
helm upgrade <documentumdeployment_name> . --values ./platforms/aws.yaml --values .
\dockerimages-values.yaml --values <resources_yaml_file>.yaml -n <namespace>
```

Now, run the kubectl get ingress command again and you should see the host value updated with the corresponding address as follows:

```
$ kubectl get ingress
Example output with address:
NAME          HOSTS           ADDRESS
dctm-internal-ingress   east-1.elb.amazonaws.com   k8s-d2newdctmgroup-440a21863a-1248970307.us-
                                         k8s-d2newdctmgroup-440a21863a-1248970307.us-
                                         east-1.elb.amazonaws.com
```

## ! Important

- If you are planning to use user-friendly names for the ingress host, you can map the user-friendly host names provided in the Helm charts to the ingress address in the DNS server, or you can map the user-friendly host names provided in the Helm charts to the ingress address and the host IP address in the hosts file of your client machine (see screenshot attached below ) as applicable and access the applications using the user-friendly host names.

To get the host IP address, use the command:

```
ping <ingress-address>
```

```
# localhost name resolution is handled within DNS itself.
# 127.0.0.1      localhost
# ::1            localhost

3.214.181.176  ccgeingress.us-east-1.elb.amazonaws.com  k8s-d2214ccgroup-16dc54713d-1384428887.us-east-1.elb.amazonaws.com
52.203.102.250  d2geingress.us-east-1.elb.amazonaws.com  k8s-d2214publicdctmgr-6dd80014ce-1833497987.us-east-1.elb.amazonaws.com
52.44.163.213  awggeingress.us-east-1.elb.amazonaws.com  k8s-d2214awgrygroup-3a7f007d52-1550993048.us-east-1.elb.amazonaws.com
54.209.294.219  dctmeingress.us-east-1.elb.amazonaws.com  k8s-d2214dctmgrou-p-a2ebac36c1-2090327313.us-east-1.elb.amazonaws.com
```

- Once the content server is deployed for first time and upgraded as mentioned above, Content-Server can be accessed by external clients (outside the cluster) as below:

Client connection:

- Content-Server should be deployed and configured with external access enabled as mentioned earlier. After the initial deployment, there will be one load balancer services created for csext with an external IP.
- Copy dfc.properties from /opt/dctm/config/dfc.properties to your client machine from the primary Content-Server pod. The original dfc.properties will have two host and port pairs. Remove one pair of host and port and keep one updated as follows:

```
dfc.docbroker.host[0]=<External IP of dbrext LoadBalancer 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
```

- In the previous deployment, if node port was used for externalizing docbroker/content server, if you want to use load balancer (ELB) URL, set useELB to true in documentum/values.yaml for the content-server section and upgrade the deployment. Follow step 5 of this section to upgrade the deployment with the load balancer URL.

#### 3.5.19.7.2 OTDS configuration post-deployment

- Access the OTDS admin website <Ingress URL>/otds-admin with admin as the username and the value of the password you specified in OTDS Helm chart's values.yaml file.
- Click on **Partitions** on the left side and create a non-synchronized user partition. Add some users to this partition. After D2 webapp Helm charts are deployed, you can use the user account you created to log into D2 webapps.
- Click on "Resources" on the left tab, then click on "Add" on the top right.
  - Give a unique "Resource name" and "Description".
  - On the "Synchronization" tab, check "User and group synchronization", "Create users and groups" and "Modify users and groups." For "Synchronization connector," select "REST (Generic)."
  - For "Connection Information" tab, base URL should be http://<value specified for serviceName in the Content-Server Helm chart's values.yaml>-jms-service:<value specified for ports.jmsport in Content-Server Helm chart's values.yaml>/dmotdsrest. For example "http://dcs-pg-jms-service:9080/dmotdsrest." Username should be "<docbase name>\<install owner name>" and password should be "<install owner password>." Click on "Test connection."
  - The test should succeed.

- e. For "User Attribute Mappings," add "client\_capability" attribute with "Format" value of "2." Add "default\_folder" attribute with "OTDS Attribute" value as "cn" and "Format" value as "%s."
  - f. Take the default for everything else and click on "Save."
  - g. Repeat above steps for all additional repositories.
4. Click on "Access Roles" on the left tab. You'll find "Access to <your newly created resource>."
    - a. Click on "Actions" and select "Include Groups."
    - b. Click on "Actions" again and select "View Access Role Details."
    - c. Under "Users" tab, remove "otadmin@otds.admin" user if there is one. Add users for different access roles for multiple repositories.
    - d. Under "Groups" tab, remove "otdsadmins@otds.admin" group if there is one.
    - e. Click on "Save."
    - f. Repeat above steps for all additional repositories.
  5. Click on "Access Roles" on the left tab. You'll find "Access to <global repository deployment resource>." Click on "Actions" again, select "View Access Role Details" under "Users" tab, and remove "otadmin@otds.admin" user if there is one. Add Partitions created in step 2 under "Groups" tab and remove "otdsadmins@otds.admin" group if there is one. Click on "Save." Repeat the above steps for all additional repositories.



**Note:** This step is done in order to have all the additional repository users be in global repository as well. This is necessary for D2 otds login in additional repositories.

6. Click on "Resources" on the left tab again. Click on "Actions" next to your resource then select "Consolidate" to synchronize the members down to your D2 Content-Server repository.
7. Remove/Comment the line X3-OTDS.defaultRepository=<docbase name> in d2classic-shiro-ini configmap and restart the D2 Classic pod in global repository deployment.

#### **Set the acs\_base\_url to localhost and acs\_supported\_protocol to s3 for S3 store/Google Cloud Store/ Azure BLOB store**

1. Make sure disableUpdateAcsUrl is true for the content-server section in documentum/values.yaml.
2. Open iapi in the Content-Server pod and execute the following commands:

```
API>?,c,select object_name,r_object_id from dm_acs_config  
#Note down the acs_base_url by dumping any one of the ACS r_object_id
```

3. Update <content-server-pod-name> with content-server pod name and update <cluster space> with env: &env\_domain mentioned in the anchor tag section of documentum/values.yaml and update <original ingress url> with acs\_base\_url copied in the above step.

```

API> fetch,c,<id1 returned above>

API> set,c,1,acs_base_url[0]

SET> http://<content-server-pod-name>.dcs-pg.<cluster space>:9080/ACS/servlet/ACS

API> set,c,1,acs_supported_protocol

SET> s3

API> append,c,1,acs_base_url

<original ingress url>

API> append,c,1,acs_supported_protocol

https

API> save,c,1

...

OK

#To check if the parameter is updated, open a new iapi session and execute the
following command:

API> ?,c,select object_name,r_object_id from dm_acs_config

Dump each <r_object_id> and check the acs_base_url and acs_supported_protocol
parameter

dump,c,<r_object_id>

Note: 0th index of acs_base_url and acs_supported_protocol has to be localhost and
s3 correspondingly

```

4. In case there are multiple replicas of the Content-Server pod, the command API> ?,c,select object\_name,r\_object\_id from dm\_acs\_config returns multiple values in order of the replicas. Repeat the above steps for all the r\_object\_ids returned by updating the Content-Server pod names. For example:

```

API> fetch,c,<id1 returned above>

API> set,c,1,acs_base_url[0]

SET> http://dcs-pg-0.dcs-pg.namespace2.svc.cluster.local:9080/ACS/servlet/ACS

API> set,c,1,acs_supported_protocol

SET> s3

API> append,c,1,acs_base_url

<original ingress url>

API> append,c,1,acs_supported_protocol

https

API> save,c,1

...

```

```

OK
API> fetch,c,<id2 returned above>
API> set,c,l,acs_base_url[0]
SET> http://dcs-p-1.dcs-pg.namespace2.svc.cluster.local:9080/ACS/servlet/ACS
API> set,c,l,acs_supported_protocol
SET> s3
API> append,c,l,acs_base_url
<original ingress url>
API> append,c,l,acs_supported_protocol
https
API> save,c,l
...
OK

```

### Making GCP store as the default store in Documentum GCP deployment

- Once the deployment is complete and configured with native filestore and the functional testing is completed, make the GCP store as the default store using the below iapi command in the Content-Server pod.

```

kubectl exec -it dcs-pg-0 -c dcs-pg bash -n <namespace>
iapi <docbase_name>
API> ?,c,alter type dm_sysobject set DEFAULT STORAGE = '<gcpstore>'
API> ?,c,alter type dm_document set DEFAULT STORAGE = '<gcpstore>'
API> reinit,c

```

- Steps to check the functionality of GCP-object store set as the default store: Try to push/pull the content through the iapi session by following the below steps (while pushing the content, don't set a\_storage\_type as the content will by default go to the default GCP store):

```

API> create,c,dm_document
...
0901e24080002dee
API> setfile,c,l,/opt/dctm/share/sample.txt,crtext
...
OK
API> save,c,l
...
OK
API> getpath,c,l
...
https://storage.googleapis.com/testidlit/01e240/80/00/11/ce.dat
API> getfile,c,l
/opt/dctm/local/process8118925741892856614.tmp/0/0601e24080009c7a.txt

```



**Note:** The content is now stored in the GCP store <https://storage.googleapis.com/testidlit/01e240/80/00/11/ce.dat> and the content is in encrypted format. Any time before the DAR installation, the default store should be switched to the filestore.

### Making BLOB store as the default store in Documentum Azure deployment

- Once the deployment is complete and configured with native filestore and the functional testing is completed, make the BLOB store as the default store using the below iapi command in the Content-Server pod.

```
kubectl exec -it dcs-pg-0 -c dcs-pg bash -n <namespace>
iapi <docbase_name>
API> ?,c,alter type dm_sysobject set DEFAULT STORAGE = '<Azure blob store>'
API> ?,c,alter type dm_document set DEFAULT STORAGE = '<Azure blob store>'
API> reinit,c
```

- Steps to check the functionality of BLOB store set as the default store: Try to push/pull the content through the iapi session by following the below steps (while pushing the content, don't set a\_storage\_type as the content will by default go to the default BLOB store):

```
API> create,c,dm_document
...
0901e24080002dee
API> setfile,c,1,/opt/dctm/share/sample.txt,crtext
...
OK
API> save,c,1
...
OK
API> getpath,c,1
https://dctmreststoredev.blob.core.windows.net/testkube/01e240/80/00/15/11.dat
API> getfile,c,1 API> getfile,c,1
/opt/dctm/local/process8118925741892856614.tmp/0/0601e24080009c7a.txt
```



**Note:** The content is now stored in the BLOB store <https://dctmreststoredev.blob.core.windows.net/testkube/01e240/80/00/15/11.dat> and the content is in encrypted format. Any time before DAR installation, the default store should be switched to the filestore.

### Making S3 store as the default store in Documentum AWS deployment

- Once the deployment is complete and configured with native filestore and the functional testing is completed, make the S3 store as the default store using the below iapi command in the Content-Server pod.

```
kubectl exec -it cedcs-pg-0 bash -n <namespace>
iapi <docbase_name>
API> ?,c,alter type dm_sysobject set DEFAULT STORAGE = '<s3store>'
API> ?,c,alter type dm_document set DEFAULT STORAGE = '<s3store>'
API> reinit,c
```

- Steps to check the functionality of S3-object store set as the default store: Try to push/pull the content through the iapi session by following the below steps (while pushing the content, don't set a\_storage\_type as the content will by default go to the default S3 store):

```
API> create,c,dm_document
...
0901e24080002dee
API> setfile,c,1,/opt/dctm/share/sample.txt,crtext
...
OK
API> save,c,1
...
```

```
OK
API> getpath,c,1
...
http://s3.ap-south-1.amazonaws.com/dctm-s3-bucket-1/01e240/80/00/9c/7a.dat
API> getfile,c,1
/opt/dctm/local/process8118925741892856614.tmp/0/0601e24080009c7a.txt
```



**Note:** The content is now stored in S3 store <http://s3.ap-south-1.amazonaws.com/dctm-s3-bucket-1/01e240/80/00/9c/7a.dat> and the content is in encrypted format. Any time before DAR installation, the default store should be switched to the filestore.

#### Steps related to CTS configuration

- After CTS is configured with additional repository, the DA pod should be restarted otherwise the transformation feature will be disabled in DA until it gets restarted. This can be done by following the below steps:

```
kubectl get pod -n namespace1 | grep da
da-68f76df88b-7mxmn #da pod name

kubectl delete pod <da pod name> -n namespace1
```

#### 3.5.19.8 Known issues

- DTR-1935: Clicking on the Reports widgets multiple times is throwing an error

#### 3.5.19.9 Post-deployment validation

Once the repositories are successfully deployed, each deployed repository should be listed in the repository dropdown list on the login page of each webapp, i.e., d2classic, d2config, and d2smartview.

## 3.6 Deploying and configuring Documentum Records Client on private cloud

### 3.6.1 Prerequisites

- Perform the steps from [step 1 to step 4](#) in “Deploying and configuring Documentum Server on private cloud” on page 56.
- Deploy the Documentum Server pod as described in “[Deploying and configuring Documentum Server on private cloud](#)” on page 56.
- Download the Docker image (Oracle Linux only) from OpenText Container Registry. Perform the following tasks:
  - Log in to OpenText Container Registry using the following command format:

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

When prompted, provide your My Support login credentials.

- 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	23.4.0 or 23.4
dctm-records-darinstallation	23.4.0 or 23.4
dctm-rqm	23.4.0 or 23.4

4. Download the Helm charts from 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 195.
  - b. Deploy the Records Client. See “[Deploying Documentum Records Client](#)” on page 198.
  - c. Deploy the Records Queue Manager. See “[Deploying Records Queue Manager](#)” on page 203.

### 3.6.2 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 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,dm1_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 documentum/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 `<location where Helm charts are extracted>/init-containers/recordsdarinstaller-init.yaml` file, provide the appropriate values in `dctm-server.content-server` for the variables as described in the following table:
- | Name  | Description  |
|---|--|
| <code>extraInitContainers.name</code>                   | Name of the additional init containers.                      |
| <code>extraInitContainers.image</code>                  | recordsdarinstaller image repository location.               |
| <code>extraInitContainers.imagePullPolicy</code>        | Policy type to fetch the image. The default value is Always. |
| <code>extraInitContainers.volumeMounts.name</code>      | Name of the volume mount for the init container.             |
| <code>extraInitContainers.volumeMounts.mountPath</code> | Path of the volume mount for the init container.             |
| <code>extraInitContainers.volumeMounts.subPath</code>   | 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> --values <location where Helm charts are extracted>/config/configuration.yml --values <location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values <location where Helm charts are extracted>/dockerimages-values.yaml --values <location where Helm charts are extracted>/documentum-resources-values-<config>.yaml --values <location where Helm charts are extracted>/documentum-components.yaml --values <location where Helm charts are extracted>/initcontainers/recordsdarinstaller-init.yaml --values <location where Helm charts are extracted>/documentum/values.yaml --namespace <name of namespace>
```

where <config> is: **-extra-large-enhanced, -extra-large-standard, -large-enhanced, -large, -large-standard, -medium-enhanced, -medium-standard, -medium-large-enhanced, -medium-large-standard, -small-medium-enhanced, -small-medium-standard, -small-enhanced, -small-standard, or -test-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 install dcmdeployment /opt/temp/documentum --values /opt/temp/documentum/config/configuration.yml --values /opt/temp/documentum/platforms/cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /opt/temp/documentum/documentum-components.yaml --values /opt/temp/documentum/initcontainers/recordsdarinstaller-init.yaml --values /opt/temp/documentum/values.yaml --namespace onedctmns
```

- For upgrade, use the following command format:

```
helm upgrade <release name> <location where Helm charts are extracted> --values <location where Helm charts are extracted>/config/configuration.yml --values <location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values <location where Helm charts are extracted>/dockerimages-values.yaml --values <location where Helm charts are extracted>/documentum-resources-values-<config>.yaml --values <location where Helm charts are extracted>/documentum-components.yaml --values <location where Helm charts are extracted>/initcontainers/recordsdarinstaller-init.yaml --values <location where Helm charts are extracted>/documentum/values.yaml --namespace <name of namespace>
```

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

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.

### 3.6.3 Deploying Documentum Records Client

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

Name	Description
enabled	Indicates if Records Client pod deployment is enabled. The default value is <code>false</code> .
namespace	Name of the Kubernetes platform. This variable uses the value that you specified for namespace in common-variables in “Defining common variables” on page 53.
userName	User name of the installation owner. This variable uses the value that you specified for installOwnerUsername in common-variables in “Defining common variables” on page 53. The default value is <code>dmadmin</code> .
serviceAccount.createServiceaccount	Indicates to create a new service account. The default value is <code>false</code> .  ! <b>Important</b> If you set the value to <code>true</code> , you must provide a unique service account name manually for <code>serviceAccount.serviceAccountName</code> .
serviceAccount.serviceAccountName	Name of the service account. This variable uses the value that you specified for documentumServiceaccount in common-variables in “Defining common variables” on page 53.
custom.scriptInPVC	Custom scripts in PVC.
custom.scriptPVCname	Name of the custom script in PVC.
custom.PVCSubPath	Path of PVC.
persistentVolumeClaim.storageClass	Storage class for the PV. This variable uses the value that you specified for rwoStorage in common-variables in “Defining common variables” on page 53.
persistentVolumeClaim.awsEFSCSIDriver	External storage provisioner (Amazon Elastic File System Container Storage Interface (CSI) Driver). The default value is <code>efs.csi.aws.com</code> .
persistentVolumeClaim.awsEFSCSIHandle	Volume ID of the EFS volume.
service.name	Name of the service. The default value is <code>records-svc</code> .
service.port	Port reserved for the service. The default value is 8080.
ingress.name	Name of the ingress service.

Name	Description
ingress.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.
ingress.ingressHostName	Ingress host name.
ingress.clusterDomainName	Domain name of the cluster. This variable uses the value that you specified for ingressDomain in common-variables in “Defining common variables” on page 53.
ingress.annotations	Metadata attachment to ingress objects. <ul style="list-style-type: none"> <li>• nginx.ingress.kubernetes.io/proxy-body-size: 5g</li> <li>• nginx.ingress.kubernetes.io/proxy-connect-timeout: 30m</li> </ul>
docbroker.serviceName	Service name of connection broker. This variable uses the value that you specified for dbrServiceName in common-variables in “Defining common variables” on page 53.
docbroker.port	Port for the connection broker.
env.domain	Name of the cluster space. This variable uses the value that you specified for env in common-variables in “Defining common variables” on page 53.
cs.useCSDfcConfigMap	Indicates to use DFC properties from ConfigMap. <ul style="list-style-type: none"> <li>• true: Uses DFC properties from Documentum Server ConfigMap.</li> <li>• false: Uses DFC properties from ConfigMap created through the Records Helm.</li> </ul> <p> <b>Note:</b> If Documentum Server is SSL enabled, set this value as false.</p>
cs.configMapName	ConfigMap name used in Documentum Server. This variable uses the value that you specified for dbrConfigmapName in common-variables in “Defining common variables” on page 53. The format is <sname>.configmap where <sname> is the same provided in the single All-In-One documentum/values.yaml Helm chart file.
cs.csSecretConfigName	Name of the secret configuration file. This variable uses the value that you specified for csSecrets in common-variables in “Defining common variables” on page 53.
cs.allowTrustedLogin	Indicates to use the <i>dfc.session.allow_trusted_login</i> in the <i>dfc.properties</i> file. The default value is false.

Name	Description
certificate.useCertificate	Indicates to connect to connection broker with SSL certificate. This variable uses the value that you specified for useCertificate in common-variables in “Defining common variables” on page 53.
certificate.dbrServiceName	Service name of connection broker. This variable uses the value that you specified for dbrServiceName in common-variables in “Defining common variables” on page 53.
certificate.dbrDataPVCName	Subpath of certificate inside the connection broker PVC.
deployment.name	Name of the Documentum Records Client deployment.
images.graylog.enable	Indicates if the use of Graylog Sidecar is enabled. This variable uses the value that you specified for grayLogEnable in common-variables in “Defining common variables” on page 53.
images.graylog.server	Server details based on your Graylog server configuration. This variable uses the value that you specified for graylogServer in common-variables in “Defining common variables” on page 53.
images.graylog.port	Port reserved for the Graylog server. This variable uses the value that you specified for graylogPort in common-variables in “Defining common variables” on page 53.
containers.records.name	Name of the application.
containers.records.docbaseName	Name of the repository. This variable uses the value that you specified for docbase in common-variables in “Defining common variables” on page 53.
containers.records.containerPort	Port reserved within the container to access the application.
containers.records.probing.healthPort	Port reserved for the Documentum Records Client pod.
containers.records.probing.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.
containers.records.probing.successThreshold	Minimum consecutive successes for the probe to be considered successful after having failed.
containers.records.probing.timeoutSeconds	Number of seconds after which the probe times out.

Name	Description
containers.records.probing.readinessProbe.initialDelaySeconds	Number of seconds after the Documentum Records Client pod has started before the readiness probe is initiated.
containers.records.probing.readinessProbe.periodSeconds	Frequency to perform the probe.
containers.records.probing.livenessProbe.initialDelaySeconds	Number of seconds after the Documentum Records Client pod has started before the liveness probe is initiated. <a href="#">“Checking liveness of Documentum Records Client” on page 400</a> contains detailed information.
containers.records.probing.livenessProbe.periodSeconds	Frequency to perform the probe.
otds.enable	Indicates if OTDS is enabled. This variable uses the value that you specified for otdsEnabled in common-variables in <a href="#">“Defining common variables” on page 53</a> .
otds.url	OTDS URL.
otds.clientID	ID of the Records OAuth client. This variable uses the value that you specified for oauthClient in common-variables in <a href="#">“Defining common variables” on page 53</a> .
otds.scheme	Type of scheme. This variable uses the value that you specified for ingressProtocol in common-variables in <a href="#">“Defining common variables” on page 53</a> .
otds.authentication.repoSelectionPageRequired	Indicates if the repository page must be displayed on login. The default value is true.
otds.authentication.loginTicketTimeout	Life span of the login ticket. The default value is 250.
otds.authentication.renewtokenafterlogout	Indicates if the token must be renewed on logout. This value must be set to true if the Records deployment requires OTDS authentication.
wdkAppXmlConfig.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

Name	Description
logging.rootLoggerLevel	Root log level for Documentum Records Client. The default value is WARN.
logging.consoleThresholdLevel	Console threshold level for Documentum Records Client. The default value is WARN.
logging.filename	Name of the log file.
logging.logFileSize	Maximum size of the log file.
logging.maxLogFiles	Location of the log file.
userProvidedServices.newrelic	<a href="#">“Integrating New Relic with Documentum Records Client” on page 385</a> contains detailed information.
dfcTracing.enable	Indicates if DFC tracing is enabled. The default value is false.
dfcTracing.configMapName	DFC tracing ConfigMap name for Documentum Records Client.
dfcTracing.filePrefix	File prefix. For example, dfcTrace.
dfcTracing.logLevel	Log level for the DFC tracing. The default value is DEBUG.

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

- For new deployment, use the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values <location where Helm charts are extracted>/config/configuration.yaml --values <location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values <location where Helm charts are extracted>/dockerimages-values.yaml --values <location where Helm charts are extracted>/documentum-resources-values-<config>.yaml --values <location where Helm charts are extracted>/documentum-components.yaml --values <location where Helm charts are extracted>/documentum/values.yaml --namespace <name of namespace>
```

where <config> is: **-extra-large-enhanced, -extra-large-standard, -large-enhanced, -large, -large-standard, -medium-enhanced, -medium-standard, -medium-large-enhanced, -medium-large-standard, -small-medium-enhanced, -small-medium-standard, -small-enhanced, -small-standard, or -test-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 install dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/config/configuration.yaml --values /opt/temp/documentum/platforms/cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /opt/temp/documentum/documentum-components.yaml --values /opt/temp/documentum/values.yaml --namespace onedctmns
```

- For upgrade, use the following command format:

```
helm upgrade <release name> <location where Helm charts are extracted> --values <location where Helm charts are extracted>/config/configuration.yaml --values <location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values <location where Helm charts are extracted>/dockeringimages-values.yaml --values <location where Helm charts are extracted>/documentum-resources-values-<config>.yaml --values <location where Helm charts are extracted>/documentum-components.yaml --values <location where Helm charts are extracted>/documentum/values.yaml --namespace <name of namespace>
```

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



**Note:** Records Client Privilege will be approved automatically.

- Verify the status of the Helm deployment using the following command format:

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

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

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

### 3.6.4 Deploying Records Queue Manager

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

Name	Description
enabled	Indicates if Records Queue Manager pod deployment is enabled. The default value is <code>false</code> .
namespace	Name of the Kubernetes platform. This variable uses the value that you specified for <code>namespace</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .
serviceName	Name of the service.
userName	User name of the installation owner. This variable uses the value that you specified for <code>installOwnerUsername</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> . The default value is <code>dmaadmin</code> .

Name	Description
serviceAccount.createServiceAccount	Indicates to create a new service account. The default value is false.  ! <b>Important</b> If you set the value to true, you must provide a unique service account name manually for serviceAccount.serviceAccountName.
serviceAccount.serviceAccountName	Name of the service account. This variable uses the value that you specified for documentumServiceaccount in common-variables in “Defining common variables” on page 53.
custom.scriptInPVC	Indicates to create the PVC.
custom.scriptPVCName	Name of the custom script in PVC.
custom.PVCSubPath	Path of PVC.
persistentVolumeClaim.pvcName	Name of the PVC.
persistentVolumeClaim.storageClass	Storage class for the PV. This variable uses the value that you specified for rwoStorage in common-variables in “Defining common variables” on page 53.
persistentVolumeClaim.awsEFSCSIDriver	External storage provisioner (Amazon Elastic File System Container Storage Interface (CSI) Driver). The default value is efs.csi.aws.com.
persistentVolumeClaim.awsEFSCSIHandle	Volume ID of the EFS volume.
cs.useCSDfcConfigMap	Indicates to use DFC properties from ConfigMap. <ul style="list-style-type: none"> <li>• true: Uses DFC properties from Documentum Server ConfigMap.</li> <li>• false: Uses DFC properties from ConfigMap created through the Records Helm.</li> </ul>  <b>Note:</b> If Documentum Server is SSL enabled, set this value as false.
cs.configMapName	ConfigMap name used in Documentum Server. This variable uses the value that you specified for dbrConfigmapName in common-variables in “Defining common variables” on page 53.  The format is <sname>.configmap where <sname> is the same provided in the single All-In-One documentum/values.yaml Helm chart file.
cs.csSecretConfigName	Name of the secret configuration file. This variable uses the value that you specified for csSecrets in common-variables in “Defining common variables” on page 53.

Name	Description
cs. allowTrustedLogin	Indicates to use the <code>dfc.session.allow_trusted_login</code> in the <code>dfc.properties</code> file. The default value is <code>false</code> .
certificate. useCertificate	Indicates to connect to connection broker with SSL certificate. This variable uses the value that you specified for <code>useCertificate</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .
certificate. dbrServiceName	Service name of connection broker. This variable uses the value that you specified for <code>dbrServiceName</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .
images.graylog. enable	Indicates to use the Graylog Sidecar. This variable uses the value that you specified for <code>grayLogEnable</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .
images.graylog. server	Server details based on your Graylog server configuration. This variable uses the value that you specified for <code>graylogServer</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .
images.graylog. port	Port reserved for the Graylog server. This variable uses the value that you specified for <code>graylogPort</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .
containers.rqm. name	Name of the application. Specify the value provided for <code>serviceName</code> in the <code>rqm</code> section.
containers.rqm. containerName	Name of the container. For example, <code>RQMContainer</code> .
containers.rqm. rqmSingleHelm	Indicates if RQM pod deployment is using single Helm chart. The default value is <code>true</code> .
containers.rqm. replicaCount	Number of replica pods. The default value is 1.
containers.rqm. installFiles	Indicates whether to retain the Records Queue Manager installation files. The default value is <code>false</code> .
containers.rqm. docbaseName	Name of the repository. This variable uses the value that you specified for <code>docbase</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .
containers.rqm. rqmDocbaseUser	User name of the Records Queue Manager repository user. This variable uses the value that you specified for <code>installOwnerUsername</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .
containers.rqm. rqmSysAdminName	User name of the Records Queue Manager system administrator.
containers.rqm. rqmSysAdminPass	User name of the Records Queue Manager system administrator.
containers.rqm. globalRegistryRep ository	Global repository name as in <code>dfc.globalregistry.repository</code> . This variable uses the value that you specified for <code>docbase</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .

Name	Description
containers.rqm.bofRegistryUser	User name of the global registry. This variable uses the value that you specified for globalRegistryUsername in common-variables in <a href="#">"Defining common variables" on page 53</a> . The default value is dm_bof_registry. Do NOT change this value.
containers.rqm.docbrokerhostname	Name of the connection broker used in Records Queue Manager.
containers.rqm.rqmadminport	Port reserved for the Records Queue Manager Admin.
containers.rqm.rqmjettyport	Port reserved for the Records Queue Manager Jetty.
containers.rqm.docbrokerport	Port reserved for the connection broker.
containers.rqm.recordsaws	Indicates whether to enable Records on AWS.
docbroker.serviceName	Service name of connection broker. This variable uses the value that you specified for dbrServiceName in common-variables in <a href="#">"Defining common variables" on page 53</a> .
docbroker.port	Port for the connection broker.
env.domain	Name of the cluster space. This variable uses the value that you specified for env in common-variables in <a href="#">"Defining common variables" on page 53</a> .
userProvidedServices.newrelic	<a href="#">"Integrating New Relic with Documentum Records Client" on page 385</a> contains detailed information.

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

- For new deployment, use the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values <location where Helm charts are extracted>/config/configuration.yml --values <location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values <location where Helm charts are extracted>/dockerimages-values.yaml --values <location where Helm charts are extracted>/documentum-resources-values-<config>.yaml --values <location where Helm charts are extracted>/documentum-components.yaml --values <location where Helm charts are extracted>/documentum/values.yaml --namespace <name of namespace>
```

where <config> is: **-extra-large-enhanced, -extra-large-standard, -large-enhanced, -large, -large-standard, -medium-enhanced, -medium-standard, -medium-large-enhanced, -medium-large-standard, -small-medium-enhanced, -small-medium-standard, -small-enhanced, -small-standard, or -test-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 install dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/config/configuration.yml --values /opt/temp/documentum/platforms/
```

```
cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /opt/temp/documentum/documentum-components.yaml --values /opt/temp/documentum/values.yaml --namespace onedctmns
```

- For upgrade, use the following command format:

```
helm upgrade <release name> <location where Helm charts are extracted> --values <location where Helm charts are extracted>/config/configuration.yaml --values <location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values <location where Helm charts are extracted>/dockerimages-values.yaml --values <location where Helm charts are extracted>/documentum-resources-values-<config>.yaml --values <location where Helm charts are extracted>/documentum-components.yaml --values <location where Helm charts are extracted>/documentum/values.yaml --namespace <name of namespace>
```

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

- Verify the status of the Helm deployment using the following command format:

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

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

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

### 3.6.5 Limitations

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

### 3.6.6 Troubleshooting

Symptom	Cause	Fix
When you check the status of available pods using the <code>kubectl get pods</code> 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 <code>kubectl delete pods &lt;name of the pod&gt;</code> command. The pod is recreated automatically.
When you check the status of available deployed image, it results in the Error: <code>ImagePullBackOff</code> error.	Incorrect Helm deployment.	Delete the Helm deployment using the <code>helm delete --purge &lt;name of the pod&gt; --namespace &lt;name of namespace&gt;</code> command, provide the correct image path and redeploy the Helm chart.

## 3.7 Deploying and configuring Documentum Foundation Services on private cloud

### 3.7.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 56 in “Deploying and configuring Documentum Server on private cloud” on page 56 except for those steps related to the PostgreSQL database.

### 3.7.2 Deploying Documentum Foundation Services

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from [step 2 to step 9](#) in “Deploying Documentum Server” on page 58.
3. Make sure that `dfs.enabled` is set to `true` in the `<location where Helm charts are extracted>/documentum-components.yaml` file.
4. Make sure that all the required Docker image details are updated in the `dfs` section in the `<location where Helm charts are extracted>/dockerimages-values.yaml` file.
5. **Optional** You can create a configuration file for each environment that you deploy. The values in a configuration file override the values of `documentum/values.yaml`. You can find a basic template of the config file in `documentum/config/configuration.yaml`. If you are using this configuration file, make sure that you pass this `configuration.yaml` in the Helm deployment commands.
6. Go to the `dfs` section in the single All-In-One `documentum/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:

**Table 3-13: dfs**

Name	Description
<code>enabled</code>	Indicates if Documentum Foundation Services deployment is enabled.
<code>serviceName</code>	Name of the service.
<code>installOwner</code>	User name of the installation owner. This variable uses the value that you specified for <code>installOwnerUsername</code> in common-variables in “Defining common variables” on page 53.

Name	Description
secretsChange	Indicates if any changes to variables are made in <code>dctm-server.cs-secrets</code> . This variable uses the value that you specified for <code>secrets_Change</code> in <code>common-variables</code> in “Defining common variables” on page 53. See “Defining common variables” on page 53 for the detailed description.
serviceAccount.createServiceaccount	Indicates to create a new service account. The default value is <code>false</code> . <p><b>! Important</b> If you set the value to <code>true</code>, you must provide a unique service account name manually for <code>serviceAccount.serviceAccountName</code>.</p>
serviceAccount.serviceAccountName	Name of the service account. This variable uses the value that you specified for <code>documentumserviceaccount</code> in <code>common-variables</code> in “Defining common variables” on page 53.
cs.useCSDfcConfigMap	Indicates to use DFC properties from ConfigMap. <ul style="list-style-type: none"> <li><code>true</code>: Uses DFC properties from Documentum Server ConfigMap.</li> <li><code>false</code>: Uses DFC properties from ConfigMap created through the Documentum Foundation Services Helm.</li> </ul>
cs.configMapName	ConfigMap name used in Documentum Server. This variable uses the value that you specified for <code>dbrConfigmapName</code> in <code>common-variables</code> in “Defining common variables” on page 53.
cs.csSecretConfigName	Name of the secret configuration file. This variable uses the value that you specified for <code>csSecrets</code> in <code>common-variables</code> in “Defining common variables” on page 53.
containers.dfs.name	Name of the container.
containers.dfs.containerHttpPort	HTTP port reserved for the Documentum Foundation Services container. The default value is 8080.
containers.dfs.containerSslPort	SSL port reserved for the Documentum Foundation Services container. The default value is 8443.

Name	Description
containers.dfs.probing.url	Documentum Foundation Services URL that must be probed for readiness and liveness.
containers.dfs.probing.port	Documentum Foundation Services port on which Documentum Foundation Services server is running.
containers.dfs.probing.failureThreshold	<p>Number of times the probe is allowed to fail.</p> <p>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.</p>
containers.dfs.probing.successThreshold	Minimum consecutive successes for the probe to be considered successful after having failed.
containers.dfs.probing.timeoutSeconds	Number of seconds after which the probe times out.
containers.dfs.probing.readinessProbe.initialDelaySeconds	Number of seconds after the Documentum Foundation Services pod has started before the probe is initiated.
containers.dfs.readinessProbe.periodSeconds	Frequency to perform the probe.
containers.dfs.probing.livenessProbe.initialDelaySeconds	<p><a href="#">“Checking liveness of Documentum Foundation Services” on page 400</a> contains detailed information.</p>
containers.dfs.probing.livenessProbe.periodSeconds	<p><a href="#">“Checking liveness of Documentum Foundation Services” on page 400</a> contains detailed information.</p>
containers.dfs.fluentd.enable	Indicates if Fluentd is enabled. The default value is <code>false</code> .
containers.dfs.fluentd.TCPPort	Port on which Fluentd is listening for TCP connection. This variable uses the value that you specified for <code>fluentdTcpPort</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
containers.dfs.fluentd.RESTPort	Port on which Fluentd is listening for REST connection. This variable uses the value that you specified for <code>fluentdRestPort</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .

Name	Description
containers.dfs.fluentd.compressionMode	Indicates the compression mode. The default value is gzip.
containers.dfs.fluentd.bufferingMode	Indicates the buffering mode. The default value is FILEBUFFER.
containers.dfs.fluentd.flushInterval	Time taken to flush the delayed events.
containers.dfs.fluentd.name	Name of the Fluentd image.
containers.dfs.fluentd.pullPolicy	Policy type to fetch the Fluentd image. The default value is Always.
containers.dfs.fluentd.restartPolicy	Policy type to restart the pods. The default value is Always.
containers.dfs.fluentd.fluentdLogFolder	Log folder name for the Fluentd image. The default value is fluentd-logging.
containers.dfs.fluentd.fluentdConfigFolder	Configuration folder name for the Fluentd image. The default value is fluentd-config-map.
containers.dfs.fluentd.servicedataPVCName	Name of the PVC. The default value is fluentd-data-pvc.
containers.dfs.fluentd.kafkaBrokerList	Broker list information of Apache Kafka. This variable uses the value that you specified for kafkaBrokerList in common-variables in “Defining common variables” on page 53.
containers.dfs.fluentd.readiness.enable	Indicates if the readiness probe is enabled. The default value is false.
containers.dfs.fluentd.readiness.kafkaTopic	Topic name provided while deploying the Apache Kafka cluster for storing the events. This variable uses the value that you specified for kafka_topic_name in common-variables in “Defining common variables” on page 53.
containers.dfs.fluentd.readiness.kafkaUser	Administrator user name used to communicate among the replicas in the Apache Kafka cluster. This variable uses the value that you specified for kafka_admin_user_name in common-variables in “Defining common variables” on page 53.

Name	Description
containers.dfs.fluentd.readiness.kafkaUsrPasswd	Administrator password used to communicate among the replicas in the Apache Kafka cluster. This variable uses the value that you specified for <code>kafka_admin_user_password</code> in <code>common-variables</code> in “ <a href="#">Defining common variables</a> ” on page 53.
containers.dfs.graylog.enabled	Indicates if Graylog is enabled for use. The default value is <code>false</code> . This variable uses the value that you specified for <code>grayLogEnable</code> in <code>common-variables</code> in “ <a href="#">Defining common variables</a> ” on page 53. Only if the value is set to <code>true</code> , the graylog side container is created.
containers.dfs.graylog.name	Name of the Graylog server.
containers.dfs.graylog.server	Details of Graylog server. This variable uses the value that you specified for <code>graylogServer</code> in <code>common-variables</code> in “ <a href="#">Defining common variables</a> ” on page 53.
containers.dfs.graylog.port	Available port reserved for the Graylog server. This variable uses the value that you specified for <code>graylogPort</code> in <code>common-variables</code> in “ <a href="#">Defining common variables</a> ” on page 53.
containers.dfs.graylog.serviceToken	Service token of Graylog Sidecar.
tomcat.username	User name of Tomcat manager. The default value is <code>admin</code> .
tomcat.password	Password of Tomcat manager. The default value is <code>password</code> .
tomcat.tomcatClusterEnabled	Indicates to enable Tomcat clustering when you have multiple Documentum Foundation Services pods. The default value is <code>true</code> .
certificate.use_certificate	Indicates to enable certificate-based communication. This variable uses the value that you specified for <code>useCertificate</code> in <code>common-variables</code> in “ <a href="#">Defining common variables</a> ” on page 53.
certificate.dbrserviceName	Service name of connection broker. This variable uses the value that you specified for <code>dbrServiceName</code> in <code>common-variables</code> in “ <a href="#">Defining common variables</a> ” on page 53.

Name	Description
certificate.dfcTrustStorePassword	Trust store password of Documentum Foundation Classes. The default value is password.
dfc.docbroker	<p> <b>Note:</b> You must provide the appropriate values for this section only if you do not want to use the DFC properties of Documentum Server ConfigMap. If you want the Documentum Foundation Services pod to use the values provided in this section for the dfc.properties file, then you must set the value of cs.useCSDfcConfigMap in the dfs section to false.</p>
dfc.port	Port reserved for the connection broker. The default value is 1489.
dfc.globalRegistryRepository	Repository defined as a global registry. This variable uses the value that you specified for docbase in common-variables in <a href="#">“Defining common variables” on page 53</a> .
dfc.globalRegistryUsername	User name of the global registry user. This variable uses the value that you specified for globalRegistryUsername in common-variables in <a href="#">“Defining common variables” on page 53</a> .
dfc.globalRegistryPassword	Password for the global registry user. This variable uses the value that you specified for globalRegistryPassword in common-variables in <a href="#">“Defining common variables” on page 53</a> .
dfc.connectionMode	Type of the connection mode. The default value is try_native_first.
dfc.cryptoRepository	Repository defined as a global registry. This variable uses the value that you specified for docbase in common-variables in <a href="#">“Defining common variables” on page 53</a> .
dfc.dataDir	Data directory path of Documentum Foundation Classes. The default value is /var/documentum.
	 <b>Note:</b> The data directory path need not be changed.

Name	Description
dfc.client.should_use_enduserinfo	Indicates to enable the end user tracking feature. The default value is false.
dfc.client.should_use_eventhub	Indicates to enable the Event Hub feature. The default value is false.
dfc.client.eventhub.log_level	Event log levels.
dfc.client.eventhub.queue_size	Size of the events in kilobytes buffered in Documentum Foundation Classes.
dfc.additionalProperties	<p>Additional properties for use in the <code>dfc.properties</code> file. By default, the additional properties are commented. To use the following existing additional DFC properties in the <code>dfc.properties</code> file, uncomment them as follows:</p> <pre>- 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</pre> <p>You can also add new additional DFC properties you want to use in the <code>dfc.properties</code> file.</p>
log4j.rootLogLevel	Root log level for the Documentum Foundation Services packages. The default value is WARN.
log4j.rtLogLevel	Log level for the Documentum Foundation Services run time module. The default value is WARN.
log4j.datamodelLogLevel	Log level for the Documentum Foundation Services data model module. The default value is WARN.
log4j.servicesLogLevel	Log level for the Documentum Foundation Services services module. The default value is WARN.
log4j.toolsLogLevel	Log level for the Documentum Foundation Services tools module. The default value is WARN.
log4j.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.

Name	Description
service.ports.sslPort	SSL port reserved for the Documentum Foundation Services service. The default value is 8443.
newrelic.enable	"Integrating New Relic with Documentum Foundation Services" on page 386 contains detailed information.
newrelic.dfs_application_name	"Integrating New Relic with Documentum Foundation Services" on page 386 contains detailed information.
newrelic.proxy_host	"Integrating New Relic with Documentum Foundation Services" on page 386 contains detailed information.
newrelic.proxy_port	"Integrating New Relic with Documentum Foundation Services" on page 386 contains detailed information.
configMap.fluentd_configMap_name	ConfigMap name of the Fluentd image. The default value is dfsfluentd-configmap.
volumeClaimTemplate.logVctAccessModes	Access modes of the VCT log. The default value is ReadWriteOnce.
volumeClaimTemplate.logVctStorageClass	Storage class of the VCT log. This variable uses the value that you specified for rwoStorage in common-variables in "Defining common variables" on page 53.
volumeClaimTemplate.logVctSize	Size of the VCT log. The default value is 2Gi.
extensionPVC.createPVC	Indicates to create PVC. The default value is true.
extensionPVC.PVCAccessMode	Access mode of PVC. The default value is ReadWriteOnce.
extensionPVC.PVCStorageClass	Storage class of PVC. This variable uses the value that you specified for rwoStorage in common-variables in "Defining common variables" on page 53.
extensionPVC.PVCSIZE	Size of the PVC. The default value is 2Gi.
extensionPVC.useCommonPVC	Indicates to use the common PVC if the PVC already exists and need to be shared across Documentum products. This variable uses the value that you specified for tomcatbase_usecommonpvc in common-variables in "Defining common variables" on page 53.

Name	Description
extensionPVC.commonPVCName	Name of the common PVC. This variable uses the value that you specified for tomcatbase_commonpvcname in common-variables in “Defining common variables” on page 53.

7. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where
Helm charts are extracted>/documentum-components.yaml --namespace <name of
namespace>
```

where <config> is -extra-large-enhanced, -extra-large-standard, -large-enhanced, -large-standard, -medium-enhanced, -medium-large-enhanced, -medium-large-standard, -medium-standard, -small-enhanced, -small-medium-enhanced, -small-medium-standard, -small-standard, or -test-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 install dcmdeployment /opt/temp/documentum --values /opt/temp/documentum/
platforms/cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --
values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /
opt/temp/documentum/documentum-components.yaml --namespace onedctmns
```

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

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

9. Verify the status of the deployment of the pod using the following command format:

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



## Notes

- After setting the values for the variables in log4j or to upgrade the Documentum Foundation Services pod, run the Helm upgrade command.
- 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 log4j 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.

### 3.7.3 Limitations

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

### 3.7.4 Troubleshooting

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

## 3.8 Deploying and configuring Documentum REST Services on private cloud

### 3.8.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 56 in “Deploying and configuring Documentum Server on private cloud” on page 56 except for those steps related to the PostgreSQL database.

### 3.8.2 Deploying Documentum REST Services

1. Extract the Helm chart downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from step 2 to step 9 in “Deploying Documentum Server” on page 58.
3. **Optional** If you want the full-text search and CTS capabilities, then deploy xPlore and Content Transformation Services in the cluster.
4. Make sure that dctm-rest.enabled is set to true in the <location where Helm charts are extracted>/documentum-components.yaml file.

5. Make sure that all the required Docker image details are updated in the `dctm-rest` section in the <location where Helm charts are extracted>/`dockerimages-values.yaml` file.
6. **Optional** You can create a configuration file for each environment that you deploy. The values in a configuration file override the values of `documentum/values.yaml`. You can find a basic template of the `config` file in `documentum/config/configuration.yaml`. If you are using this configuration file, make sure that you pass this `configuration.yaml` in the Helm deployment commands.
7. Go to the `dctm-rest` section in the single All-In-One `documentum/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:

**Table 3-14: dctm-rest**

Name	Description
<code>enabled</code>	Indicates if Documentum REST Services deployment is enabled.
<code>serviceName</code>	Name of the service. The default value is <code>dctm-rest</code> .
<code>namespace</code>	Name of the namespace. This variable uses the value that you specified for <code>namespace</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
<code>serviceAccount.createServiceaccount</code>	Indicates to create a new service account. The default value is <code>false</code> .  <b>Important</b> If you set the value to <code>true</code> , you must provide a unique service account name manually for <code>serviceAccount.serviceAccountName</code> .
<code>serviceAccount.serviceAccountName</code>	Name of the service account. By default, this variable uses the value that you specified for <code>documentumserviceaccount</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
<code>customLabels.app</code>	Label for the deployment. The default value is <code>dctm-rest</code> .
<code>containerName</code>	Name of the container. The default value is <code>rest-container</code> .

Name	Description
rest.useCommonPVC	Indicates to use the common PVC if the PVC already exists and need to be shared across Documentum products. This variable uses the value that you specified for <code>tomcatbase_usecommonpvc</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .
rest.commonPVCname	Name of the common PVC. This variable uses the value that you specified for <code>tomcatbase_commonpvcname</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .
serviceType	Type of the service. This variable uses the value that you specified for <code>webappServiceType</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .
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.
existingConfigMap	Name of the existing ConfigMap. This variable uses the value that you specified for <code>dbrConfigmapName</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> . If you do not provide any value, installing Documentum REST Services Helm creates a new ConfigMap with configuration files specified by <code>configurationFiles</code> .
configurationFiles	Details of the configuration files. For example, <code>rest-api-runtime.properties</code> and <code>dfc.properties</code> . The files should be in the root directory of the Helm chart.
extraConfigMountPath	Volume mount path for additional configurations. The default value is <code>/home/dmadmin/ext-conf</code> .

Name	Description
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 <i>Oracle Linux</i> documentation.
content_server.secretName	Name of the secret configuration file. This variable uses the value that you specified for csSecrets in common-variables in <a href="#">“Defining common variables” on page 53</a> .
docbroker.useCertificate	Indicates to connect to connection broker with SSL certificate. This variable uses the value that you specified for useCertificate in common-variables in <a href="#">“Defining common variables” on page 53</a> .
docbroker.dbrServiceName	Service name of connection broker.
log4j.rootLogLevel	Root log level of logging. The default value is INFO.
log4j.restLogLevel	Log level of the Documentum REST Services container. The default value is INFO.
log4j.dfcLogLevel	Log level of DFC. The default value is INFO
formatMsgNoLookups	Indicates to modify the logging configuration to enable or disable message lookups. The default value is true.
graylog.enabled	Indicates to use the Graylog Sidecar. This variable uses the value that you specified for grayLogEnable in common-variables in <a href="#">“Defining common variables” on page 53</a> .
graylog.server	Server details based on your Graylog server configuration. This variable uses the value that you specified for graylogServer in common-variables in <a href="#">“Defining common variables” on page 53</a> .
graylog.port	Port reserved for the Graylog server. This variable uses the value that you specified for graylogPort in common-variables in <a href="#">“Defining common variables” on page 53</a> .

Name	Description
graylog.serviceToken	Service token of the Graylog Sidecar.
newrelic	"Integrating New Relic with Documentum REST Services" on page 387 contains detailed information.
livenessprobe	"Checking liveness of Documentum REST Services" on page 401 contains detailed information.
fluentd_service.enable	Indicates if Fluentd is enabled. This variable uses the value that you specified for fluentd in common-variables in "Defining common variables" on page 53.
fluentd_service.TCPPort	Port on which Fluentd is listening for TCP connection. This variable uses the value that you specified for fluentdTcpPort in common-variables in "Defining common variables" on page 53.
fluentd_service.RESTPort	Port on which Fluentd is listening for REST connection. This variable uses the value that you specified for fluentdRestPort in common-variables in "Defining common variables" on page 53.
fluentd_service.servicedataPVCName	Name of the PVC. For example, fluentd-data-pvc.
fluentd_service.readiness.enable	Indicates if the readiness probe is enabled. The default value is false.
fluentdConf.enable	Indicates if Fluentd is enabled. This variable uses the value that you specified for fluentd in common-variables in "Defining common variables" on page 53.
fluentdConf.TCPPort	Port on which Fluentd is listening for TCP connection. This variable uses the value that you specified for fluentdTcpPort in common-variables in "Defining common variables" on page 53.
fluentdConf.RESTPort	Port on which Fluentd is listening for REST connection. This variable uses the value that you specified for fluentdRestPort in common-variables in "Defining common variables" on page 53.

Name	Description
fluentdConf.UDPPort	Port on which Fluentd is listening for UDP connection. This variable uses the value that you specified for fluentdUdpPort in common-variables in “Defining common variables” on page 53.
fluentdConf.kafkaTopic	Topic name you provided while deploying the Apache Kafka cluster.
fluentdConf.kafkaUser	Administrator user name used to communicate among the replicas in the Apache Kafka cluster.
fluentdConf.kafkaUsrPasswd	Administrator password used to communicate among the replicas in the Apache Kafka cluster.
fluentdConf.compressionMode	Type of the compression mode. The default value is gzip.
fluentdConf.bufferingMode	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.
fluentdConf.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.
fluentdConf.kafkaBrokerList	Broker list information of Apache Kafka. This variable uses the value that you specified for kafkaBrokerList in common-variables in “Defining common variables” on page 53.
acsService.serviceName	Service name for the ACS ingress resource. This variable uses the value that you specified for jmsServiceName in common-variables in “Defining common variables” on page 53.
acsService.servicePort	Port reserved for the ACS ingress resource. The default value is 9080.
otds.enable	Indicates if OTDS is enabled. This variable uses the value that you specified for otdsEnabled in common-variables in “Defining common variables” on page 53. The rest-api-runtime.properties file is populated with the value provided for this variable.

Name	Description
otds.url	OTDS service URL. The format is <code>https://&lt;OTDS server url&gt;:&lt;port&gt;/otdsws</code> . The <code>rest-api-runtime.properties</code> file is populated with the value provided for this variable.
otds.clientID	Details of client ID. This variable uses the value that you specified for <code>oauthClient</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> . The <code>rest-api-runtime.properties</code> file is populated with the value provided for this variable.
single_helm.enable	Indicates if Documentum REST Services deployment using single Helm is enabled. The default value is <code>true</code> .
custom.useInitContainers	Indicates to use init containers. The default value is <code>false</code> .
custom.scriptPVCname	Name of the custom script in PVC. The default value is <code>custom-script-pvc</code>
custom.PVCSubPath	Subpath inside the referenced volume. The default value is <code>dctm-rest-custom</code>
custom.PVCSIZE	Size of the PVC. The default value is <code>1Gi</code> .
custom.pvcAccessModes	Access modes of the PVC. The default value is <code>ReadWriteMany</code> .
custom.storageClass	Storage class of the PVC. This variable uses the value that you specified for <code>rwmStorage</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
custom.existVolumePv	Details of existing volume, if any.
timeZone.enable	Indicates if you want to deploy the Documentum REST Services pod in a specific time zone. The default value is <code>false</code> .
timeZone.value	Any valid time zone. The default value is <code>Etc/UTC</code> .

8. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where
Helm charts are extracted>/documentum-components.yaml --namespace <name of
namespace>
```

where <config> is -extra-large-enhanced, -extra-large-standard, -large-enhanced, -large-standard, -medium-enhanced, -medium-large-enhanced, -medium-large-standard, -medium-standard, -small-enhanced, -small-medium-enhanced, -small-medium-standard, -small-standard, or -test-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 install dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/platforms/cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /opt/temp/documentum/documentum-components.yaml --namespace onedctmns
```

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

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

10. Verify the status of the deployment of the pod using the following command format:

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

### 3.8.3 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 7](#).
4. Run the Helm upgrade command.

### 3.8.4 Limitations

There are no limitations for this release.

### 3.8.5 Troubleshooting

There are no troubleshooting information for this release.

## 3.9 Deploying and configuring Documentum Content Management Interoperability Services on private cloud

### 3.9.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 56 in “Deploying and configuring Documentum Server on private cloud” on page 56 except for those steps related to the PostgreSQL database.

### 3.9.2 Deploying Documentum Content Management Interoperability Services

1. Extract the Helm chart downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from [step 2 to step 9](#) in “Deploying Documentum Server” on page 58.
3. Make sure that `dctm-cmis.enabled` is set to `true` in the `<location where Helm charts are extracted>/documentum-components.yaml` file.
4. Make sure that all the required Docker image details are updated in the `dctm-cmis` section in the `<location where Helm charts are extracted>/dockerimages-values.yaml` file.
5. **Optional** You can create a configuration file for each environment that you deploy. The values in a configuration file override the values of `documentum/values.yaml`. You can find a basic template of the `config` file in `documentum/config/configuration.yaml`. If you are using this configuration file, make sure that you pass this `configuration.yaml` in the Helm deployment commands.
6. Go to the `dctm-cmis` section in the single All-In-One `documentum/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:

**Table 3-15: dctm-cmis**

Name	Description
enabled	Indicates if Documentum Content Management Interoperability Services deployment is enabled.
serviceName	Name of the service. The default value is <code>dctm-cmis</code> .
namespace	Name of the namespace. This variable uses the value that you specified for <code>namespace</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
<code>serviceAccount.createServiceaccount</code>	Indicates to create a new service account. The default value is <code>false</code> .  <b>Important</b> If you set the value to <code>true</code> , you must provide a unique service account name manually for <code>serviceAccount.serviceAccountName</code> .
<code>serviceAccount.serviceAccountName</code>	Name of the service account. This variable uses the value that you specified for <code>documentumserviceaccount</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
<code>customLabels.app</code>	Label for the deployment. The default value is <code>dctm-cmis</code> .
<code>containerName</code>	Name of the container. The default value is <code>cmis-container</code> .
<code>cmis.pvcName</code>	Name of the PVC.
<code>cmis.pvcSize</code>	Size of the PVC. The default value is <code>2Gi</code> .
<code>cmis.pvcAccessModes</code>	Access mode of the PVC. The default value is <code>ReadWriteMany</code> .
<code>cmis.storageClass</code>	Storage class of the PVC. This variable uses the value that you specified for <code>rwmStorage</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .
<code>cmis.existVolumePv</code>	Details of existing volume, if any.
<code>cmis.useCommonPVC</code>	Indicates to use the common PVC if the PVC already exists and need to be shared across Documentum products. This variable uses the value that you specified for <code>tomcatbase_usecommonpvc</code> in <code>common-variables</code> in <a href="#">“Defining common variables” on page 53</a> .

Name	Description
cmis.commonPVCname	Name of the common PVC. This variable uses the value that you specified for <code>tomcatbase_commonpvcname</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .
httpPort	HTTP port reserved for the Documentum Content Management Interoperability Services service. The default value is 8080.
httpsPort	HTTPS port reserved for the Documentum Content Management Interoperability Services service. The default value is 8443.
existingConfigMap	Name of the existing ConfigMap. This variable uses the value that you specified for <code>dbrConfigmapName</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .  If you do not provide any value, installing Documentum REST Services Helm creates a new ConfigMap with configuration files specified by <code>configurationFiles</code> .
configurationFiles	Details of the configuration files.  The files should be in the root directory of the Helm chart.
extraConfigMountPath	Volume mount path for additional configurations. The default value is <code>/home/dmadmin/ext-conf</code> .
java.javaOptions	Used to provide the Java options required during the startup of the Tomcat server.  The default value <code>-Djava.security.egd=file:/dev/.urandom</code> is provided to improve the application performance associated with the random number generation. For more information about random number generation involved with <code>/dev/random</code> and <code>/dev/urandom</code> , see <i>Oracle Linux</i> documentation.
content_server.secretName	Name of the secret configuration file. This variable uses the value that you specified for <code>csSecrets</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .
docbroker.useCertificate	Indicates to connect to connection broker with SSL certificate. This variable uses the value that you specified for <code>useCertificate</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> .

Name	Description
docbroker.dbrServiceName	Service name of connection broker.
docbroker.pvcCertSubPath	Subpath of the PVC certificate in connection broker.
log4j.rootLogLevel	Root log level of logging. The default value is INFO.
log4j.cmisLogLevel	Log level of the Documentum Content Management Interoperability Services container. The default value is INFO.
log4j.dfcLogLevel	Log level of DFC. The default value is INFO
formatMsgNoLookups	Modify the logging configuration to enable or disable message lookups. The default value is true.
graylog.enabled	Indicates to use the Graylog Sidecar. This variable uses the value that you specified for grayLogEnable in common-variables in <a href="#">"Defining common variables" on page 53</a> .
graylog.server	Server details based on your Graylog server configuration. This variable uses the value that you specified for graylogServer in common-variables in <a href="#">"Defining common variables" on page 53</a> .
graylog.port	Port reserved for the Graylog server. This variable uses the value that you specified for graylogPort in common-variables in <a href="#">"Defining common variables" on page 53</a> .
graylog.serviceToken	Service token of the Graylog Sidecar.
newrelic	<a href="#">"Integrating New Relic with Documentum Content Management Interoperability Services" on page 388</a> contains detailed information.
livenessprobe	<a href="#">"Checking liveness of Documentum Content Management Interoperability Services" on page 401</a> contains detailed information.
acsService.serviceName	Service name for the ACS ingress resource. This variable uses the value that you specified for jmsServiceName in common-variables in <a href="#">"Defining common variables" on page 53</a> .
acsService.servicePort	Port reserved for the ACS ingress resource. The default value is 9080.

Name	Description
single_helm.enable	Indicates if Documentum Content Management Interoperability Services deployment using single Helm is enabled. The default value is true.

7. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where
Helm charts are extracted>/documentum-components.yaml --namespace <name of
namespace>
```

where <config> is -extra-large-enhanced, -extra-large-standard, -large-enhanced, -large-standard, -medium-enhanced, -medium-large-enhanced, -medium-large-standard, -medium-standard, -small-enhanced, -small-medium-enhanced, -small-medium-standard, -small-standard, or -test-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 install dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/
platforms/cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --
values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /
opt/temp/documentum/documentum-components.yaml --namespace onedctmns
```

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

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

9. Verify the status of the deployment of the pod using the following command format:

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

### 3.9.3 Limitations

There are no limitations for this release.

### 3.9.4 Troubleshooting

There are no troubleshooting information for this release.

## 3.10 Deploying and configuring Documentum Reports on private cloud

### 3.10.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 56 in “Deploying and configuring Documentum Server on private cloud” on page 56.

### 3.10.2 Deploying Documentum Reports (dtrinstaller and dtrbase)

#### 3.10.2.1 dtrinstaller

This deployment is for installing the DCTM\_Reports\_Base.dar and DCTM\_Reports\_GBL.dar files in Documentum Server and custom JMS Method to Documentum Server.



**Note:** The Documentum Reports DAR files are prerequisites for Client applications Helm deployment.

#### 3.10.2.2 dtrbase

1. Perform the tasks mentioned from [step 2](#) to [step 9](#) in “Deploying Documentum Server” on page 58 in “Deploying and configuring Documentum Server on private cloud” on page 56.
2. Navigate to the dtrbase category in the single All-In-One documentum/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	<i>enabled</i>	Indicates if Documentum Reports pod deployment is enabled. The default value is <code>false</code> .
service	<i>name</i>	Name of the service. The default value is <code>dtrbase</code> .
persistent VolumeClaim	<i>pvcName</i>	Name of the Persistent Volume Claim (PVC). The default value is <code>dtr-pvc</code> .
	<i>accessMode</i>	Access mode of the PVC. The default value is <code>ReadWriteOnce</code> .
	<i>size</i>	Storage size of the PVC. The default value is <code>1Gi</code> .

Category	Name	Description
	<i>storageClass</i>	Storage class for the PVC. Specify the value provided for <i>rwoStorage</i> in the common-variables category in “Defining common variables” on page 53.
	<i>PVCSubPath</i>	Path to copy custom scripts. The default value is <i>dtr-customscript</i> .
ingress	<i>name</i>	Name of the ingress service. The default value is <i>dtr-ingress</i> .
	<i>enable</i>	Indicates if the ingress service is enabled. This variable controls the creation of ingress service for your deployment. If <i>dctm-ingress</i> is used, then it is recommended to keep it disabled. Valid values are <i>true</i> and <i>false</i> and it is case-sensitive. The default value is <i>false</i> .
	<i>ingressHostName</i>	Name of the ingress host. The cluster domain name appended to <i>ingressHostName</i> during the creation of Ingress. The default value is <i>dtr-ingress-host</i> .
	<i>clusterDomainName</i>	DNS name of the cluster.
	<i>annotations</i>	Annotations to attach metadata to ingress object. <ul style="list-style-type: none"> <li>• <i>nginx.ingress.kubernetes.io/proxy-body-size</i> : The default value is <i>5g</i>.</li> <li>• <i>nginx.ingress.kubernetes.io/proxy-connect-timeout</i> : The default value is <i>30m</i>.</li> </ul>
cs	<i>useCSDfcConfigMap</i>	Indicates to use the ConfigMap name used in Documentum Server. Valid values are <i>true</i> and <i>false</i> . <ul style="list-style-type: none"> <li>• <i>true</i>: Use DFC properties from Documentum Server logging ConfigMap.</li> <li>• <i>false</i>: Use DFC properties from the <i>dfcProperties</i> environment variable.</li> </ul> The default value is <i>true</i> and it is case-sensitive.
	<i>configMapName</i>	ConfigMap name used in Documentum Server. The format is <i>&lt;sname&gt;.configmap</i> where <i>&lt;sname&gt;</i> is the same provided in the single All-In-One documentum/values.yaml Helm chart file. The default value is <i>dbr.configmap</i> .
	<i>csSecretConfigName</i>	Name of the secret configuration file that was created while deploying the Documentum Server pod. Specify the value provided for <i>csSecrets</i> in the common-variables category in “Defining common variables” on page 53.

Category	Name	Description
	<i>allowTrustedLogin</i>	Indicates if trusted login is enabled. The default value is <code>false</code> .
docbroker	<i>port</i>	Indicates the port for the connection broker. Specify the value provided for <code>docbrokerPort</code> in the common-variables category in “Defining common variables” on page 53.
	<i>replicaCount</i>	Number of replica pods. Specify the value provided for <code>docbrokerRepcount</code> in the common-variables category in “Defining common variables” on page 53.
	<i>serviceName</i>	Service name of connection broker. Specify the value provided for <code>dbrServiceName</code> in the common-variables category in “Defining common variables” on page 53.
env	<i>domain</i>	Name of cluster space. Specify the value provided for <code>env</code> in the common-variables category in “Defining common variables” on page 53.
methodsvr	<i>containerName</i>	Name of the container for the methodsvr. The default value is <code>dcs-pg</code> .
deployment	<i>name</i>	Indicates the deployment name. Specify the value provided for <code>dtrbase</code> .
	<i>appName</i>	Application name for Documentum Reports. Specify the value provided for <code>dtrbase</code> .
	<i>replicaCount</i>	Number of replica pods. The default value is 1. The maximum value is 10.
images	<i>dtrbase</i>	<ul style="list-style-type: none"> <li><i>repository</i>: Path of the repository. Specify the value provided for <code>imageRepository</code> in the common-variables category in “Defining common variables” on page 53.</li> <li><i>name</i>: Name of the Documentum Reports image. The default value is <code>dctm-reports-base</code>.</li> <li><i>tag</i>: Tag as a version-specific number.</li> <li><i>pullPolicy</i>: Policy type to fetch the image. Specify the value provided for <code>pullPolicyType</code> in the common-variables category in “Defining common variables” on page 53.</li> </ul>

Category	Name	Description
	<i>graylog</i>	<ul style="list-style-type: none"><li>• <i>enable</i>: Specify the value provided for <i>grayLog</i> in the common-variables category in “Defining common variables” on page 53. Only if the value is set to true, the graylog side container is created.</li><li>• <i>image</i>: Graylog image repository location. Specify the value provided for <i>grayLogImage</i> in the common-variables category in “Defining common variables” on page 53.</li><li>• <i>pullPolicy</i>: Policy type to fetch the image. Specify the value provided for <i>pullPolicyType</i> in the common-variables category in “Defining common variables” on page 53.</li><li>• <i>server</i>: Details of the Graylog server. Specify the value provided for <i>graylogServer</i> in the common-variables category in “Defining common variables” on page 53.</li><li>• <i>port</i>: Port reserved for the Graylog server. Specify the value provided for <i>graylogPort</i> in the common-variables category in “Defining common variables” on page 53.</li></ul>

Category	Name	Description
containers	<i>dtrbase</i>	<ul style="list-style-type: none"> <li>• <i>name</i>: Name of the container. Specify the value provided for <i>dtrbase</i>.</li> <li>• <i>darInstallRepo</i>: Name of the DAR installation repository. Specify the value provided for <i>docbase</i> in the common-variables category in “<a href="#">Defining common variables</a>” on page 53.</li> <li>• <i>drServiceUserID</i>: The email ID for the user created through DA. The default value is <i>dctmreports@ot.com</i>.</li> <li>• <i>ingressHost</i>: Ingress controller domain name service value. The format is &lt;INGRESS-HOSTNAME&gt;. &lt;CLUSTER-DOMAIN-NAME&gt;. <ul style="list-style-type: none"> <li>– <i>drCoreServerTimeout</i>: Time for which the server is being timed out. The default value is 00:30:00.</li> <li>– <i>drCoreSmtpPort</i>: Port number for the mail server. The default value is 25.</li> </ul> </li> <li>• <i>drCoreSmtpAuthRequired</i>: Indicates if authentication is required. The default value is true.</li> <li>• <i>drCoreMailSubject</i>: Subject of the mail. The default value is test eMail from DR Job.</li> <li>• <i>drCoreMaxAttachmentSize</i>: Maximum attachment size in a mail. The default value is 0.</li> <li>• <i>drCoreAttachAsZip</i>: Indicates if attaching Zip files is enabled. The default value is false.</li> <li>• <i>drCoreSmtpUser</i>: Name of the SMTP user. The default value is <i>username</i>.</li> <li>• <i>drCoreUrlTimeout</i>: Time for the URL time out. The default value is 500000.</li> <li>• <i>drCoreSmtpStarttlsEnable</i>: Indicates if SMTP TLS is enabled. The default value is true.</li> <li>• <i>drCoreSmtpHost</i>: Name of the SMTP host. The default value is <i>smtp.org.net</i>.</li> <li>• <i>drCoreFromAddress</i>: Email ID of the sender. The default value is <i>user@opentext.com</i>.</li> <li>• <i>drCoreSmtpPassword</i>: Password of the sender. The default value is <i>pwd</i>.</li> <li>• <i>drCoreReportServlet</i>: URL for <i>dctmreports</i> servlet.</li> <li>• <i>drCoreReportCacheTimeout</i>: Time for the cache time out. The default value is 00:30:00.</li> </ul>

Category	Name	Description
logging	<i>rootLoggerLevel</i>	Root log level for Documentum Reports. The default value is WARN.
	<i>consoleThresholdLevel</i>	Console threshold level for Documentum Reports. The default value is WARN.
	<i>logFileSize</i>	Maximum size of the log file. The default value is 100MB.
	<i>maxLogFiles</i>	Maximum number of log files. The default value is 5.
newrelic	<i>newrelic</i>	"Integrating New Relic with Documentum Reports" on page 389 contains detailed information.
certificate	<i>useCertificate</i>	Indicates to connect to connection broker with SSL certificate. Specify the value provided for <i>useCertificates</i> in the common-variables category in "Defining common variables" on page 53.
	<i>dbrServiceName</i>	Service name of connection broker. Specify the value provided for <i>dbrServiceName</i> in the common-variables category in "Defining common variables" on page 53.
	<i>dbrDataPVCName</i>	Subpath of certificate inside the connection broker PVC. The default value is certdbr-data-pvc.

3. Deploy the Documentum Chart with Documentum Reports Helm charts using the following command format:

```
helm install <release name> <location where Helm charts are extracted>/documentum -f <location where Helm charts are extracted>/documentum/platforms/<cloud platform>.yaml <location where Helm charts are extracted>/documentum/init-containers/dtr-init.yaml --namespace <name of namespace>
```

For example:

```
helm install dctrmdeployment /opt/temp/Helm-charts/documentum -f /opt/temp/Helm-charts/documentum/platforms/cfcr.yaml /opt/temp/Helm-charts/documentum/init-containers/dtr-init.yaml --namespace docu
```

4. Verify the status of the Documentum Chart with Documentum Reports Helm charts deployment using the following command format:

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

5. Verify the status of the deployment of Documentum Chart with Documentum Reports pods using the following command format:

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

### 3.10.3 Limitations

There are no limitations for this release.

### 3.10.4 Troubleshooting

There are no troubleshooting information for this release.

## 3.11 Deploying and configuring Documentum Workflow Designer on private cloud

### 3.11.1 Prerequisites

1. Perform the steps from [step 1](#) to [step 9](#) in “Deploying and configuring Documentum Server on private cloud” on page 56.
2. Deploy the Documentum Server pod as described from [step 1](#) to [step 9](#) in “Deploying and configuring Documentum Server on private cloud” on page 56.
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 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	23.4.0 or 23.4

4. Enable `pe-silentinstaller` and `dctm-workflow-designer` in the single All-In-One documentum/values.yaml Helm chart file.
5. Provide the marker file names of client products in the `dctm-server.content-server` section in the single All-In-One documentum/values.yaml Helm chart file.

For example:

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

### 3.11.2 Deploying Documentum Workflow Designer

1. Open the single All-In-One documentum/values.yaml Helm chart file and in the dcm-workflow-designer section, provide the appropriate values for the variables to pass them to your templates as described in the following table:

Name	Description
enabled	Indicates if Process Engine pod deployment is enabled. The default value is <code>false</code> . Make sure that this attribute is enabled before you deploy Documentum Workflow Designer.
image.repository	Path of the repository.
image.name	Name of the Documentum xCP installer image.
image.tag	Tag as a version-specific number.
image.pullPolicy	Policy type to fetch the image. This variable uses the value that you specified for <code>pullPolicyType</code> in the common-variables section in “ <a href="#">Defining common variables</a> ” on page 53.
prefix	Prefix for the Process Engine pod name or ID. Use a suitable prefix to distinguish Process Engine pods in the cluster environment.
docbaseConnection.username	Name of the repository. This variable uses the value that you specified for <code>installOwner.userName</code> in the contentserver section in <code>dctm-server.cs-secrets</code> in <a href="#">step 8.a</a> .
docbaseConnection.password	Password of the repository user. This variable uses the value that you specified for <code>installOwner.password</code> in the contentserver section in <a href="#">step 8.a</a> .
docbaseConnection.globalRegistryUsername	User name of the global registry user. This variable uses the value that you specified for <code>globalUsername</code> in the cs-dfc-properties section in <a href="#">step 8.g</a> .
docbaseConnection.globalRegistryPassword	Password for the global registry user. This variable uses the value that you specified for <code>globalRegistryPassword</code> in the <code>dctm-server.cs-secrets</code> section in <a href="#">step 8.a</a> .
appServer.httpport	Service HTTP port reserved for Process Engine. The default value is 9080.
appServer.password	Password of the application server. This variable uses the value that you specified for <code>contentserver.install.appserver.admin.password</code> in the <code>dctm-server.cs-secrets</code> section in <a href="#">step 8.a</a> .

Name	Description
appServer.servicename	JMS service name.
persistentVolume.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.
persistentVolume.scriptPVCname	When Process Engine is the first client and <i>createPVC</i> 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.
persistentVolume.PVCSubPath	When Process Engine is the first client and <i>createPVC</i> 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.
persistentVolume.size	Size of the PVC.
persistentVolume.storageClassReadWriteMany	Storage class for a PV, with ReadWriteMany access mode, which can be accessible by many nodes. Specify the value provided for <i>rwmStorage</i> in the common-variables section in “Defining common variables” on page 53.

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

Name	Description
enable	Indicates if Documentum Workflow Designer pod deployment is enabled. The default value is false. Make sure that this attribute is enabled.
serviceAccount.createServiceAccount	Specifies whether to create a new service account or not. The default value is false. Set this attribute to true to create a new service account.
serviceAccount.serviceAccountName	Specifies the service account name. This variable uses the value that you specified for documentumServiceAccount in common-variables in “Defining common variables” on page 53.

Name	Description
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. This variable uses the value that you specified for <i>xcpRepository</i> in the <i>pe-silentinstaller</i> section.
initImage.name	Any image with the wget tool such as BusyBox. For example, <i>dctm-xcp-apphost</i> .
initImage.tag	Tag as a version-specific number.
initImage.pullPolicy	Policy type to fetch the image. This variable uses the value that you specified for <i>pullPolicyType</i> in the <i>common-variables</i> section in “ <a href="#">Defining common variables</a> ” on page 53.
image.repository	Path of the repository. This variable uses the value that you specified for <i>xcpRepository</i> in the <i>pe-silentinstaller</i> section.
image.name	Name of Documentum Workflow Designer image. For example, <i>dctm-workflow-designer</i> .
image.tag	Tag as a version-specific number.
image.pullPolicy	Policy type to fetch the image. This variable uses the value that you specified for <i>pullPolicyType</i> in the <i>common-variables</i> section in “ <a href="#">Defining common variables</a> ” on page 53.
contextPath	Context path of the Documentum Workflow Designer application. By default, the path for Documentum Workflow Designer application is <i>DocumentumWorkflowDesigner</i> . For example, <i>http://&lt;ingress-host&gt;/DocumentumWorkflowDesigner</i> .
persistentVolume. storageClassReadWriteOnce	Storage class for a PV, with ReadWriteOnce access mode, which can be accessible by many nodes. This variable uses the value that you specified for <i>rwoStorage</i> in the <i>common-variables</i> section in “ <a href="#">Defining common variables</a> ” on page 53.
<p> <b>Note:</b> Change the variable name from <i>rwmStorage</i> to <i>rwoStorage</i> to mount the volume to the required storage class.</p>	
persistentVolume.size	PV size to accommodate all the log files.

Name	Description
docbaseConnection.use_certificate	Indicates if certificate-based communication with Documentum Server is enabled. This variable uses the value that you specified for <i>use_certificate</i> in the common-variables section in “ <a href="#">Defining common variables</a> ” on page 53.
docbaseConnection.truststorePassword	Password of the DFC truststore that contains Documentum Server and connection broker certificates. This variable uses the value that you specified for <i>docbroker.certificate.trustpassword</i> in the <i>dctm-server.cs-secrets</i> section in <a href="#">step 8.a.</a>
docbaseConnection.docbroker	Name of the service for the connection broker.
docbaseConnection.port	Port for the connection broker. This variable uses the value that you specified for <i>docbrokerPort</i> in the common-variables section in “ <a href="#">Defining common variables</a> ” on page 53.
docbaseConnection.jmshttpport	JMS port on which Documentum Server JMS service is running. Default is 9080.
docbaseConnection.jmservicename	JMS service name created by Documentum Server.
docbaseConnection.docbase	Name of the repository. This variable uses the value that you specified for <i>docbase</i> in the common-variables section in “ <a href="#">Defining common variables</a> ” on page 53.
docbaseConnection.superUser	User name to connect to the repository. This variable uses the value that you specified for <i>installOwner</i> in the <i>dctm-server.cs-secrets</i> section in <a href="#">step 8.a.</a>
docbaseConnection.superUserPassword	Password to connect to the repository. This variable uses the value that you specified for <i>installOwnerPassword</i> in the <i>dctm-server.cs-secrets</i> section in <a href="#">step 8.a.</a>
docbaseConnection.globalRegistryRepository	Name of the global registry repository. This variable uses the value that you specified for <i>docbase</i> in the <i>dctm-server.cs-secrets</i> section in <a href="#">step 8.a.</a>
docbaseConnection.globalRegistryUsername	User name to access the global registry repository. This variable uses the value that you specified for <i>globalUsername</i> in the <i>cs-dfc-properties</i> section in <a href="#">step 8.g.</a>

Name	Description
docbaseConnection.globalRegistryPassword	Password to access the global registry repository. This variable uses the value that you specified for <i>globalRegistryPassword</i> in the <i>dctm-server.cs-secrets</i> section in <a href="#">step 8.a.</a>
dbrpersistentVolume.dbrdataPVCName	Name of the PVC. This variable uses the value that you specified for <i>persistentVolume.dbrdataPVCName</i> in the <i>dctm-server.docbroker</i> section in <a href="#">step 8.c.</a>
tomcat.javaOptions	JMS memory settings for Tomcat server.
secret.name	Name of the Documentum Server secret resource. This variable uses the value that you specified for <i>csSecrets</i> in the <i>common-variables</i> section in <a href="#">“Defining common variables” on page 53.</a>
newrelic.enable	Indicates if the New Relic Java agent is enabled. Set the element to true to enable the New Relic Java agent. This variable uses the value that you specified for <i>newrelicEnable</i> in the <i>common-variables</i> section in <a href="#">“Defining common variables” on page 53.</a>
newrelic.licenseKeyName	License key corresponding to the name of the New Relic license key. This variable uses the value that you specified for <i>newrelic.license_key</i> in the <i>dctm-server.cs-secrets</i> section in <a href="#">step 8.a.</a>
newrelic.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>.
newrelic.proxy_host	IP address of the proxy server. This variable uses the value that you specified for <i>newrelicProxyHost</i> in the <i>common-variables</i> section in <a href="#">“Defining common variables” on page 53.</a>
newrelic.proxy_port	Port reserved for the New Relic server. The default port reserved for New Relic server is 3128. This variable uses the value that you specified for <i>newrelicProxyPort</i> in the <i>common-variables</i> section in <a href="#">“Defining common variables” on page 53.</a>
liveness.initialDelay	Time in seconds after the container or pod has started before the liveness probe is initiated.
liveness.timeout	Number of seconds after which the probe times out.
liveness.period	Frequency to perform the probe.

Name	Description
liveness.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.
readiness.timeout	Number of seconds after which the probe times out.
readiness.period	Frequency to perform the probe.
readiness.failure	Time when a pod starts and the probe fails, Kubernetes attempts based on the failure threshold time before stopping.

3. Deploy the Helm chart using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/config/configuration.yaml --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/documentum-resources-values-test-
small.yaml --values <location where Helm charts are extracted>/documentum-
components.yaml --namespace <name of namespace>
```

For example:

```
helm install dcmdeployment /opt/temp/documentum --values /opt/temp/documentum/
config/configuration.yaml --values /opt/temp/documentum/platforms/cfcr.yaml --
values /opt/temp/documentum/dockerimages-values.yaml --values /opt/temp/documentum/
documentum-resources-values-test-small.yaml --values /opt/temp/documentum/
documentum-components.yaml --namespace onedctmns
```



**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.

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

### 3.11.3 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.

#### 3.11.3.1 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.

#### 3.11.3.2 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:

Name	Description
<code>prefix</code>	Prefix for name or ID of the utility pod. Use a suitable prefix to distinguish WFD client utility pod in the cluster environment.
<code>image.repository</code>	Path of the repository.
<code>image.name</code>	Name of the image.
<code>image.tag</code>	Tag as a version-specific number.
<code>image.pullPolicy</code>	Policy type to fetch the image. By default, the value is <code>IfNotPresent</code> , which skips fetching a Docker image if the same image already exists. Otherwise, use <code>Always</code> to force to fetch the Docker image.

Name	Description
image.pullSecrets	Secret for fetching images.
workflowDesigner.url	Workflow Designer URL where packages needs to be imported.
workflowDesigner.username	Workflow Designer user name to connect to the repository.
workflowDesigner.password	Workflow Designer password to connect to the repository.
workflowDesigner.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.
trustStore.fileName	Truststore filename. Truststore file should be saved in secrets folder available in Helm chart.
trustStore.password	Truststore password for the Workflow Designer.
workflowDesignerCliParam.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.
persistentVolume.storageClassReadWriteOnce	Storage class for a PV, with ReadWriteOnce access mode, which can be accessed by a single node.
persistentVolume.size	Size of the PV.
processPackagePersistentVolume.sharedPVCName	PV name shared by customer for importing the process package files.
processPackagePersistentVolume.sharedPVCFolder	Shared subpath in the PV for importing the process package files.

2. Deploy the Helm chart 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>
```

### 3.11.4 Limitations

There are no limitations for this release.

### 3.11.5 Troubleshooting

There are no troubleshooting information for this release.

## 3.12 Deploying and configuring Content Connect on private cloud

### 3.12.1 Prerequisites

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 48 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-<release-version>.tar file from the My Support.
8. Integrate Content Connect with Documentum REST Services.

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

1. Go 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
rest.security.client.token.cookie.samesite=none
```

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-
CUSTOMNAUTH- 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
rest.security.client.token.cookie.samesite=none
```
  3. Save the changes.
  4. Restart the `dctm-rest` pod.

### 3.12.2 Deploying Content Connect

1. Extract the Documentum Server Helm charts downloaded from My Support.
2. Open the single All-In-One `documentum/values.yaml` Helm chart file and perform the following:
  - In `common-variables`, set the value of `ccExtension` according to your requirement. For more information, see “[common-variables](#)” on page 53.
  - Provide the appropriate values in the `contentconnect` section to pass them to your templates as described in the following table:

**Table 3-16: contentconnect**

Name	Description
<code>enabled</code>	Indicates if Content Connect pod is deployed. The default value is set to <code>false</code> .

Name	Description
serviceAccount.createServiceAccount	Indicates if a new service account is created. The default value is false.  ! <b>Important</b> If you set the value to true, you must provide a unique service account name manually for serviceAccount.serviceAccountName.
serviceAccount.serviceAccountName	Name of the service account. This variable uses the value that you specified for documentumServiceaccount in common-variables in <a href="#">Section 3.1 on page 53</a> .
secret.DB_PASSWORD	Password of the database to be connected for Content Connect.
namespace	Namespace in the Kubernetes environment to deploy Content Connect. This variable uses the value that you specified for namespace in common-variables in <a href="#">Section 3.1 on page 53</a> .
configmap.extension	Extension name for Content Connect to be used to access Admin Console. This variable uses the value that you specified for ccExtension in common-variables in <a href="#">Section 3.1 on page 53</a> .
configmap.DB_IP	Database IP to be connected.
configmap.DB_PORT	Database port to be connected.
configmap.DB_PORT.DB_USERNAME	Database user to be connected.
configmap.DB_DB	Database name to be connected.
configmap.DB_TYPE	Database type to be connected. Valid values are: – postgres for PostgreSQL – mssql for Microsoft SQL
configmap.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.
configmap.authType	Authentication type for Content Connect. By default the value is set to otds. Available modes are: – otds – Basic
configmap.otdsUrl	OTDS server URL. Example: <a href="https://&lt;url&gt;otdsws">https://&lt;url&gt;otdsws</a> . For Basic authentication, this value must be empty.

Name	Description
configmap. otdsClientID	Client ID created for Content Connect in OTDS admin. This variable uses the value that you specified for <code>oauthclient</code> in <code>common-variables</code> in <a href="#">Section 3.1 on page 53</a> .
configmap. clientId	Client ID of the application registered in Microsoft Azure. This value is mandatory. For more information, see <i>OpenText Content Connect Installation and Administration Guide</i> .
configmap. clientSecret	Client Secret of the application registered in Microsoft Azure. This value is mandatory. For more information, see <i>OpenText Content Connect Installation and Administration Guide</i> .
configmap. tenantId	Tenant ID of the application registered in Microsoft Azure. This value is mandatory. For more information, see <i>OpenText Content Connect Installation and Administration Guide</i> .
configmap. 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 <code>ipv4</code> .
newrelic	<a href="#">“Integrating New Relic with Content Connect” on page 390</a> contains detailed information.
ingress.enabled	Enable this value to use the cc ingress. The default value is <code>false</code> .
ingress.name	Name of the ingress variable. The default value is <code>cc-ingress</code> .
ingress. configureHost	Set the value to <code>true</code> to configure the host and cluster domain name.   <b>Note:</b> To use ALB as the ingress controller, set the value to <code>false</code> . In addition, you must update the appropriate value for annotations.
ingress.tls. enable	Set the value to <code>false</code> to deploy ingress on HTTP. Default value: <code>false</code> . Set this value to <code>true</code> to deploy ingress on HTTPS. It is mandatory to provide <code>theCRT</code> and <code>key</code> values to deploy ingress on HTTPS.
ingress.tls. hosts	Name of the ingress domain URL. The format is <code>&lt;ingress-domain-url&gt;</code> . This variable uses the value that you specified for <code>ingressdomain</code> in <code>common-variables</code> in <a href="#">Section 3.1 on page 53</a> .
ingress.tls. hosts.secretName	Name of the secret configuration file. The default value is <code>cctlsecret</code> .
ingress.rules. host	Name of the ingress host. The default value is <code>ccchost</code> . The host appended to <code>ingressHostName</code> during the creation of Ingress.

Name	Description
graylog.enable	Indicates if application log files in Graylog are enabled. This variable uses the value that you specified for grayLogEnable in common-variables in <a href="#">Section 3.1 on page 53</a> . Whether Graylog is enabled or disabled, Content Connect default logging is retained.
contentconnectdb.value	Set the value to true to create the Content Connect database.  <b>Note:</b> This value must be set to false while performing an upgrade.
certificatesecret.crt	Certificate value.
certificatesecret.key	Certificate key value.

3. To enable ingress resources, make sure that you set the value of enabled to *true* in the dctm-ingress section in the single All-In-One documentum/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. Make sure that contentconnect.enabled is set to true in the <location where Helm charts are extracted>/documentum-components.yaml file.
5. Make sure that all the required Docker image details are updated in the dctm-server.contentconnect section in the <location where Helm charts are extracted>/dockerimages-values.yaml file.
6. Go 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:

```
helm install <release name> . --values .\dctm-components.yaml --values .
\dockerimages-values.yaml --values .\platforms\<current platform>.yaml --values .
\<resources>.yaml -n <namespace>
```

For example:

```
helm install cctest . --values .\dctm-components.yaml --values .\dockerimages-
values.yaml --values .\platforms\aws.yaml --values .\resources-values-test.yaml -n
test
```

7. 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>
```

8. 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

### 3.12.3 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*.

### 3.12.4 Limitations

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

### 3.12.5 Troubleshooting

Symptom	Cause	Fix
When you check the status of available pods using the <code>kubectl get pods</code> 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: <code>kubectl delete pod</code> . After you delete, the pod is recreated automatically.
When you check the status of available deployed image, it results in the <b>Error: ImagePullBackOff</b> error.	Incorrect Helm deployment.	Delete the Helm deployment using the following command: <code>helm delete &lt;deployment name&gt; -n &lt;namespace&gt;</code> . Then, provide the correct image path and redeploy the Helm Chart.

Symptom	Cause	Fix
When using Content Connect, Cross-Origin Resource Sharing (CORS) related errors occur even after all configurations are set.	CORS related errors occur.	<ul style="list-style-type: none"> <li>• Increase the load balancer response timeout.</li> <li>• Add the Content Connect Admin Console URL to the rest.cors.allowed.origins tag in the rest-api-runtime.properties file.</li> </ul>

### 3.12.6 Upgrading Content Connect

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

```
helm upgrade <release name> . --values .\dctm-components.yaml --values .\dockerimages-values.yaml --values .\platforms\<current platform>.yaml --values .\<resources>.yaml -n <namespace>
```

You can rollback the upgrade using the following command format:

```
helm rollback <release name> <revision>
```

## 3.13 Deploying and configuring Extended ECM Documentum for Microsoft 365 on private cloud

### 3.13.1 Prerequisites

1. Download and configure the Docker application from the Docker website. The *Docker documentation* contains detailed information.
2. Download and configure Helm.
3. Download and configure the Kubernetes application from the Kubernetes website.
4. Download the Extended ECM Documentum for Microsoft 365 Docker image (Alpine 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 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-smartviewm365	23.4.0 or 23.4
dctm-smartviewm365customjar	23.4.0 or 23.4

5. Run the `docker images` command to verify if the Docker images are downloaded successfully and listed.
6. Download the Documentum Helm chart available in the `documentum.tar` file from My Support.
7. Download and extract the Extended ECM Documentum for Microsoft 365 from My Support.

### 3.13.2 Deploying Extended ECM Documentum for Microsoft 365

1. Extract the Documentum Helm charts downloaded from My Support.
2. Open the single All-In-One `documentum/values.yaml` Helm chart file and perform the following:
  - In `common-variables`, set the value of `msClientId` with Microsoft Azure app registration client ID. For more information, see “[common-variables](#)” on page 53.
  - Provide the appropriate values in the `smartviewm365` section to pass them to your templates as described in the following table:

**Table 3-17: smartviewm365**

Name	Description
<code>enabled</code>	Indicates if Extended ECM Documentum for Microsoft 365 pod deployment is enabled.
<code>serviceAnnotation</code>	Defines custom annotations that must be assigned to the <code>smartviewm365</code> service.
<code>env.tomcatJVMArgs</code>	JVM arguments in Tomcat.
<code>tomcat.connectionTimeoutInMilliseconds</code>	Tomcat connection timeout in milliseconds. The default value is 60000.
<code>tomcat.maxHttpHeaderSize</code>	Maximum HTTP header size in Tomcat.
<code>tomcat.logfilerotation</code>	Maximum number of the log files. The default value is 7.
<code>tomcat.logfilesize</code>	Maximum size of the log file.
<code>readinessProbe.initialDelaySeconds</code>	Time in seconds after the container or pod has started before the readiness probe is initiated.

Name	Description
livenessProbe.initialDelaySeconds	Time in seconds after the container or pod has started before the liveness probe is initiated.
startupProbe.initialDelaySeconds	Time in seconds after the container or pod has started before the startup probe is initiated. The default value is 180.
persistentVolume.storageClass	Storage class of the persistent volume (PV). This variable uses the value that you specified for rwoStorage in common-variables in <a href="#">"Defining common variables" on page 53</a> .
graylog.enable	Indicates if Graylog is enabled for use. Graylog enabled by default. Set to false if logs mount to PVC. This variable uses the value that you specified for grayLogEnable in common-variables in <a href="#">"Defining common variables" on page 53</a> .
graylog.pvcName	Graylog persistent volume name. The default value is smartviewm365-logs-pvc.
graylog.PVCAccessMode	PVC access mode for Graylog.
graylog.PVCStorageClass	Graylog persistent volume storage class name. This variable uses the value that you specified for rwmStorage in common-variables in <a href="#">"Defining common variables" on page 53</a> .
graylog.PVCSIZE	Persistent volume storage size for Graylog.
serviceType	Service type of the Extended ECM Documentum for Microsoft 365 app. This variable uses the value that you specified for webappServiceType in common-variables in <a href="#">"Defining common variables" on page 53</a> .
extension.PVCStorageClass	Persistent volume storage class name for the Extended ECM Documentum for Microsoft 365 extension image. This variable uses the value that you specified for rwmStorage in common-variables in <a href="#">"Defining common variables" on page 53</a> .
extension.createPVC	Indicates if persistent volume must be created for the Extended ECM Documentum for Microsoft 365 extension image. The default value is false.
extension.pvcName	Persistent volume name for the Extended ECM Documentum for Microsoft 365 extension image. This variable uses the value that you specified for tomcatbase_commonpvcname in common-variables in <a href="#">"Defining common variables" on page 53</a> .

Name	Description
settings.connectionRemoteUrlProtocol	Protocol of the Ingress resource URL that a user outside the cluster uses to access the webapp. This variable uses the value that you specified for <code>ingressProtocol</code> in <code>common-variables</code> in “ <a href="#">Defining common variables</a> ” on page 53.
settings.connectionRemoteUrl	Ingress resource URL that a user outside the cluster uses to access the webapp. This variable uses the value that you specified for <code>ingressUrl</code> in <code>common-variables</code> in “ <a href="#">Defining common variables</a> ” on page 53.
newrelic	<a href="#">“Integrating New Relic with Extended ECM Documentum for Microsoft 365”</a> on page 391 contains detailed information.
nodeSelector	Node labels that is used to constrain the pods.
ccsv.appname	Application name for Extended ECM Documentum for Microsoft 365.
ccsv.teamsrootpath	Microsoft Teams home page or mapped folder. Make sure that this parameter is not blank. Make sure the <code>dm_world</code> group have the <code>WRITE</code> permission on this folder. By default, the <code>teamsrootpath</code> is set to <code>/TeamsM365</code> .
ccsv.loglevel	The log level of the Tomcat server. The recommended value is <code>ERROR</code> .
ccsv.window	Indicates if the logs will open in a new window. The default value is <code>false</code> .
ccsv.consoleRe	Indicates if the logs will open in a Console window. The default value is <code>false</code> .
ccsv.performancetimestamp	Indicates if the logs will provide the detailed time stamp <code>hh:mm:ss</code> . The default value is <code>false</code> .
ccsv.timing	Indicates if the logs will provide the time stamp in <code>hr:mm</code> format. The default value is <code>false</code> .
ccsv.clientid	Microsoft Azure app registration client ID. This variable uses the value that you specified for <code>msClientId</code> in <code>common-variables</code> in “ <a href="#">Defining common variables</a> ” on page 53.
ccsv.d2rest_url	D2 REST URL. Set the D2 REST URL in the following format <code>https://&lt;ingress&gt;/d2-rest</code> .
secret.csConfigName	Name of the secret configuration file. This variable uses the value that you specified for <code>csSecrets</code> in <code>common-variables</code> in “ <a href="#">Defining common variables</a> ” on page 53.

Name	Description
serviceAccount.createServiceaccount	Indicates to create a new service account. The default value is false.  ! <b>Important</b> If you set the value to true, you must provide a unique service account name manually for serviceAccount.serviceAccountName.
serviceAccount.serviceAccountName	Service account name for Extended ECM Documentum for Microsoft 365. This variable uses the value that you specified for documentumServiceaccount in common-variables in “Defining common variables” on page 53.

- Make sure that you provide the appropriate values in the d2rest section to configure D2 REST for Extended ECM Documentum for Microsoft 365:

Name	Description
d2rest.restApiRuntime.AllowCors.enable	Indicates if cross domain communication is allowed.
d2rest.restApiRuntime.AllowCors.restAllowedOrigins	Extended ECM Documentum for Microsoft 365 ingress URL.
d2rest.msgraphConfig.enable	Indicates if the Microsoft Azure app registration is enabled.
d2rest.msgraphConfig.clientid	Microsoft Azure app registration client ID. This variable uses the value that you specified for msClientId in common-variables in “Defining common variables” on page 53.
d2rest.msgraphConfig.clientSecret	Client Secret of the application registered in Microsoft Azure.
d2rest.msgraphConfig.tenantId	Tenant ID of the application registered in Microsoft Azure.
d2rest.msgraphConfig.scope	Microsoft Graph permissions.
d2rest.msgraphConfig.proxy_host	IP address of the proxy server.
d2rest.msgraphConfig.proxy_port	Port number of the proxy server.

- Make sure that you provide the appropriate values in the dctm-ingress section to enable ingress resources for Extended ECM Documentum for Microsoft 365:

Name	Description
dctm-ingress.smartviewm365.enabled	Indicates if ingress resource is enabled for Extended ECM Documentum for Microsoft 365. Specify this value as true.
dctm-ingress.smartviewm365.serviceName	Ingress service name for Extended ECM Documentum for Microsoft 365. Retain the service name as smartviewm365.
dctm-ingress.smartviewm365.servicePort	Ingress service port for Extended ECM Documentum for Microsoft 365. Retain the service port as 8080.

3. Make sure that `smartviewm365.enabled` is set to true in the `<location where Helm charts are extracted>/documentum-components.yaml` file.
4. Make sure that all the required Docker image details are updated in the `smartviewm365` section in the `<location where Helm charts are extracted>/dockerimages-values.yaml` file.
5. Go to the location where Helm Chart TAR files are extracted and deploy the Documentum Helm in your Kubernetes environment using the following command format:

```
helm install <release name> . --values .\documentum-components.yaml --values .\dockerimages-values.yaml --values .\platforms\<cloud platform>.yaml --values .\<resources>.yaml -n <namespace>
```

For example:

```
helm install onedcc . --values .\documentum-components.yaml --values .\dockerimages-values.yaml --values .\platforms\cfcr.yaml --values .\documentum-resources-values-test-small.yaml -n xecmtest
```

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 <release name>
```

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
smartviewm365-7bf9889df8-5gx7c	1/1	Running	0	52s

### 3.13.3 Configuring Extended ECM Documentum for Microsoft 365

1. Deploy D2 REST with Documentum Server.
2. Extract the `D2-config.zip` file. The extracted folder contains the `Teams-config.zip` file.

 **Note:** Use the `D2-config.zip` file extracted in [step 7](#).
3. In D2-Config, import the `Teams-config.zip` file and select the following options:
  - **FOLDER** and `xecmdms_teams_Acl` for the security configuration element
  - `xecmdms_channel` and `xecmdms_teams` for the context
4. Configure the following security configuration elements:
  - For the `xecmdms_channel` context, enable the **FOLDER** security configuration element.
  - For the `xecmdms_teams` context, enable the `xecmdms_teams_Acl` security configuration element.
5. Refresh the D2-Config cache.
6. Deploy and configure OTDS service for Documentum Server and D2 REST.
7. Complete the instructions in [“Manual setup” on page 133](#).
8. Configure OAuth Clients on OTDS for SmartviewM365.
  - a. Create the OTDS users with the Microsoft Teams user IDs mapped as part of the email configuration.
  - b. Open `http://<dctm-ingress>/otds-admin`, and go to the OAuth Clients section and click the OAuth client used for D2 redirect URLs.
  - c. Click **Next** and go to the Redirect URLs section.
  - d. Click **Add** and specify the Ingress URL.
  - e. Click **Save**.
  - f. Add the following sites to the Trusted Sites section:
    - `https://<dctm-ingress-url>`
    - `https://<host>.<dctm-ingress-url>/d2-rest`
    - `https://teams.microsoft.com`
  - g. Save the configurations.

### 3.13.4 Deploying the language pack

Extended ECM Documentum for Microsoft 365 supports the following languages: Arabic, German, Spanish, French, Italian, Japanese, Korean, Dutch, Swedish, Brazilian Portuguese, Hebrew, and Simplified Chinese.

To implement a language pack, build a Docker image with the Docker file as described in “[Deploying the language pack using init container](#)” on page 259 and update the language pack details in the init container in the dockerimages-values.yaml in documentum chart.

1. Extract the D2-Smartview\_LanguagePack\_xx.zip file and use the required language ZIP file to create the docker image:

 **Note:** Use the D2-Smartview\_LanguagePack\_xx.zip file extracted in [step 7 on page 252](#).

The following list includes the language codes required for deployment along with supported languages:

- Arabic: ar
- German: de
- Spanish: es
- French: fr
- Italian: it
- Japanese: ja
- Korean: ko
- Dutch: nl
- Swedish: sv
- Simplified Chinese: zh
- Brazilian Portuguese: pt\_BR
- Hebrew: he

2. After you build the image, update smartviewm365.extraInitContainers in the dockerimages-values.yaml file as shown in “[Deploying the language pack using init container](#)” on page 259.
3. In the documentum/values.yaml file, update the smartviewm365.customconfiguration.locale variable with the required language code. For example, to use the French language add fr.

 **Note:** To deploy multiple language packs, make sure the locale ZIP file for each language pack you want to deploy is available in the Docker image. Update the locale names with comma-separated values. For example, en, ar, de, es, fr, it.

4. Microsoft Teams user must select the preferred language in Microsoft Teams Settings. You should also select the same preferred language for Browser language settings.

### 3.13.4.1 Deploying the language pack using init container

1. Download the latest Alpine image from the repository.
2. Copy the following content to a file and name it dockerfile. Modify the Alpine image path according to your repository location to download the image to the local environment:

```
FROM <repository ip>:<port>/alpine:<image tag>
ARG SVMCUSTOM
ENV SVM_CUSTOM_PATH /customdir/LP-Package
RUN adduser -D -H dmadmin && \
mkdir -p $SVM_CUSTOM_PATH && \
chown -R dmadmin:dmadmin $SVM_CUSTOM_PATH
COPY --chown=dmadmin:dmadmin $SVMCUSTOM/ $SVM_CUSTOM_PATH/
USER dmadmin
CMD sh
```

3. Run the following command in the location where dockerfile is created to build the Docker image:

```
docker build -t <repository>/<custom_image_name>:<custom_image_tag> --build-arg
SVMCUSTOM=/D2-
Smartview_LanguagePack_<i>.zip -no-cache .
```

For example:

```
docker build -t repo.opentext.com/smartviewm365customimage:23.4 --build-arg
SVMCUSTOM=/D2-
Smartview_LanguagePack_de.zip -no-cache .
```

4. After you generate the Docker image, make the following changes in the Helm charts when deploying Extended ECM Documentum for Microsoft 365:

- In the dockerimages-values.yaml file, add or update smartviewm365.extraInitContainer as follows:

```
extraInitContainers:
- name: <init_container_name>
  image: <repository_path>/<init_image_name>:<image_tag>
  imagePullPolicy: Always
  command: ['/bin/sh', '-c', 'yes | cp -rf /customdir/* /opt/CC-install/custom']
  volumeMounts:
  - name: customconfig
    mountPath: /opt/CC-install/custom
```

- Update image details with the Docker image created in step 3.

The following example contains the init container details for Extended ECM Documentum for Microsoft 365:

```
extraInitContainers:
- name: teamscustomlp
  image: artifactory.oxlab.net/bpdockerhub/dctm-smartviewm365custom-
languagepack-de:23.4
  imagePullPolicy: *pull_policy_type
  command: ['/bin/sh', '-c', 'yes | cp -rf /customdir/* /opt/CC-install/
custom']
  volumeMounts:
```

```
- name: customconfig  
  mountPath: /opt/CC-install/custom
```

- In the documentum/values.yaml file, set smartviewm365.

customConfigurations.custom to true:

```
customConfigurations:  
  custom: true  
  locale: de
```



**Note:** Make sure the same language pack is deployed with D2 REST also.  
For more information about language pack deployment on D2 REST, see  
[“Deploying customizations” on page 138](#).

## 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 for all products except for Documentum REST Services, Documentum Foundation Services, and Documentum Content Management Interoperability Services) from OpenText Container Registry. Use Alpine Linux Docker images for Documentum REST Services, Documentum Foundation Services, and Documentum Content Management Interoperability Services. 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 My Support login credentials.
    - ii. 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:

<b>Image name</b>	<b>Image tag</b>
dctm-server	23.4.0 or 23.4
dctm-admin	23.4.0 or 23.4
dctm-dfs	23.4.0 or 23.4
dctm-records	23.4.0 or 23.4
dctm-records-darinstallation	23.4.0 or 23.4
dctm-rqm	23.4.0 or 23.4

Image name	Image tag
dctm-reports-installer	22.4.2
dctm-reports-base	22.4.2
dctm-rest	23.4.0 or 23.4
dctm-cmis	23.4.0 or 23.4
dctm-tomcat	23.4.0 or 23.4



**Note:** The `dctm-tomcat` image (Alpine Linux) is required only for Documentum REST Services, Documentum Foundation Services, and Documentum Content Management Interoperability Services. “Decoupling Documentum product image from base Tomcat image” on page 420 contains detailed information.

- 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-<release-version>.tar` file from My Support.

#### 4.1.1.2 Deploying Documentum Server

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Open the single All-In-One `documentum/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 56 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 `externalAccessEnabled` to `true` in `common-variables`.
  - b. Retain all the default values for `ExtDocbroker` in `dctm-server.docbroker`.
  - c. Set the value of `ExtCS.nativeExtPort` to 80 and `sslExtPort` to 81 in `dctm-server.content-server`. Retain the default values for all other variables.

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



**Note:** When you set the value of `dctm-server.dctm-ingress.ingress.configureHost` to true, you must provide the appropriate values for `dctm-server.dctm-ingress.ingress.host` and `dctm-server.dctm-ingress.ingress.clusterDomainName`.

6. Enable and update the service names for the required ingress resource in `dctm-server.dctm-ingress` in the single All-In-One documentum/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 Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where
Helm charts are extracted>/documentum-components.yaml --namespace <name of
namespace>
```

where <config> is -extra-large-enhanced, -extra-large-standard, -large-enhanced, -large-standard, -medium-enhanced, -medium-large-enhanced, -medium-large-standard, -medium-standard, -small-enhanced, -small-medium-enhanced, -small-medium-standard, -small-standard, or -test-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 install dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/
platforms/cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --
values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /
opt/temp/documentum/documentum-components.yaml --namespace onedctmns
```



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

- Verify the status of the deployment of Helm using the following command format:

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

- Verify the status of the deployment of the pod using the following command format:

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

- 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
azureenginx-ingress	nginx * 20.236.198.164	80 4d1h

- 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
```

- 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.
- Run the Helm upgrade command using the following command format:

```
helm upgrade <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where
Helm charts are extracted>/documentum-components.yaml --namespace <name of
namespace>
```

where <config> is -extra-large-enhanced, -extra-large-standard, -large-enhanced, -large-standard, -medium-enhanced, -medium-large-enhanced, -medium-large-standard, -medium-standard, -small-enhanced, -small-medium-enhanced, -small-medium-standard, -small-standard, or -test-small resource value YAML file that has been provided. These resource files contain pod sizing values like CPU, memory, and so on.

- 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 dbrexst-<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 dbrex 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 <code>[root@skvoraclelinux ~]# az aks browse --resource-group dctm --name dctmaks Merged "dctmaks" as current context in /tmp/tmpdr1aC2</code> 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:  <code>export https_proxy=&lt;proxy_value&gt;:&lt;port&gt;</code>

#### 4.1.2 Red Hat OpenShift platform

### 4.1.2.1 Azure Red Hat OpenShift platform cluster

#### 4.1.2.1.1 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 an Azure 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.

*Microsoft Azure/Red Hat OpenShift* documentation contains detailed information.

  7. Create an Azure database for the PostgreSQL server to enable Postgres as a service.
  8. Download and upload the Docker image into Azure. Perform the following tasks:
    - a. Create a container registry in Azure.
    - b. 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 My Support login credentials.

- 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	23.4.0 or 23.4

- Tag the Docker image to the Azure registry specific image using the following command format:  

```
docker tag <source image> <destination image>
```
  - Upload the Docker image to the Azure Container Registry (ACR) using the following command format:  

```
docker push <image>
```
- Download the Helm charts available in the documentum-<release-version>.tar file from My Support.

#### 4.1.2.1.2 Deploying Documentum Server

- Create a project using the following command format:

```
oc new-project <project-name>
```

- Create storage class to support the ReadWriteMany (RWX) access mode. To create a storage class to support the RWX access mode, refer to the *Red Hat Documentation*.

An example YAML to create a RWX storage class:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azure-file
provisioner: kubernetes.io/azure-file
mountOptions:
  - dir_mode=0777
  - file_mode=0777
  - uid=1000
  - gid=1000
  - mfsymlinks
  - actimeo=30
  - nosharesock
parameters:
  location: centralus
  secretNamespace: kube-system
  skuName: Standard_LRS
  storageAccount: dctmazurstorageaccount
  resourceGroup: azurefile_rg
  reclaimPolicy: Delete
  allowVolumeExpansion: true
  volumeBindingMode: Immediate
```

- Create the storage class using the following command format:

```
oc create -f <name of the YAML file>.yaml
```

- b. Make sure that there is a storage class to support both the ReadWriteOnce (RWO) and ReadWriteMany (RWX) access modes before you proceed with the deployment.
3. 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
```
4. Extract the Helm charts downloaded from My Support to a temporary location.
5. Open the single All-In-One documentum/values.yaml Helm chart file and provide the appropriate values for the variables to pass them to your templates. “[Deploying Documentum Server](#)” on page 58 contains detailed information about the variables.



### Notes

- Make sure to update the value for pullSecretName with the secret name to access the registry from Red Hat OpenShift.
  - Make sure that you set the value of class to openshift-default in dctm-server.dctm-ingress.ingress to use the openshift-default ingress controller
6. Perform all the steps from [step 8](#) to [step 12](#) in “[Deploying Documentum Server](#)” on page 58 to complete the deployment and verification of the Documentum Server pod.

#### 4.1.2.2 Red Hat OpenShift cluster on bare metal

#### 4.1.2.2.1 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. Make sure that you have a Linux virtual machine (with CentOS 8 operating system) with hard disk more than 100 GB, RAM 32 GB and 8 CPUs and add a user who is part of wheel group.
6. Download CodeReady Containers (CRC) from the Red Hat website.
7. Install Network Manager.
8. Include the CRC path in the PATH environment variable.
9. Fetch the `pull secret` configuration file from the Red Hat account.
10. Run the following commands:

```
crc setup  
crc start -p <pull_secret_file>
```

11. After the cluster is ready, log in as `kubeadmin` using the following command format:

```
oc login -u kubeadmin -p <password>
```

12. Configure Security Context Constraints (SCCs) using the following command format:

```
oc adm policy add-scc-to-user anyuid -z default  
oc adm policy add-scc-to-group anyuid system:authenticated
```

13. 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 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-server	23.4.0 or 23.4

14. Upload the Docker image to the bare metal OpenShift container registry.
15. Download the Helm charts available in the documentum-<release-version>.tar file from My Support.

#### 4.1.2.2.2 Deploying Documentum Server

1. Create a project using the following command format:

```
oc adm new-project <project_name>
```

2. Switch to the new project using the following command format:

```
oc project <project_name>
```

3. Make sure that there is a storage class to support both the ReadWriteOnce (RWO) and ReadWriteMany (RWX) access modes before you proceed with the deployment.
4. Extract the Helm charts downloaded from My Support to a temporary location.
5. Open the single All-In-One documentum/values.yaml Helm chart file and provide the appropriate values for the variables to pass them to your templates.

[“Deploying Documentum Server” on page 58](#) contains detailed information about the variables.



#### Notes

- Make sure to update the value for pullSecretName with the secret name to access the registry from Red Hat OpenShift.
  - Make sure that you set the value of class to openshift-default in the dctm-server.dctm-ingress.ingress section to use the openshift-default ingress controller
6. Perform all the steps from [step 8 to step 12 in “Deploying Documentum Server” on page 58](#) 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.
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.	<p>Do one of the following tasks:</p> <ul style="list-style-type: none"> <li>• Edit the machinset YAML files and deploy the size as Standard_D32s_v3 using the following command format:</li> </ul> <pre>oc scale -- replicas=0 machineset &lt;machinesetname&gt; oc edit machineset &lt;machinesetname&gt; oc scale -- replicas=1 machineset &lt;machinesetname&gt;</pre> <ul style="list-style-type: none"> <li>• Increase the node resource.</li> </ul>

Symptom	Cause	Fix
The <code>oc get</code> command fails.	Pending certificate signing request (CSR) for approval.	<p>Verify if any CSRs are pending using the following command format:</p> <pre>oc get csr</pre> <p>Approve the pending CSR (if any) using the following command format:</p> <pre>oc get csr -o name   xargs oc adm certificate approve</pre>
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 Microsoft Azure cloud platform

### 4.2.1 Kubernetes platform

#### 4.2.1.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 261 in “Deploying and configuring Documentum Server on Microsoft Azure cloud platform” on page 261 except for those steps related to the PostgreSQL database.

#### 4.2.1.2 Deploying Documentum Administrator

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from [step 2 to step 9](#) in “Deploying Documentum Server” on page 58 in “Deploying and configuring Documentum Server on private cloud” on page 56.
3. Perform the tasks mentioned from [step 3 to step 6](#) in “Deploying Documentum Administrator” on page 111 in “Deploying and configuring Documentum Administrator on private cloud” on page 111.
4. Update the ingress resource rule for Documentum Administrator in the `dctm-server.dctm-ingress` section in the single All-In-One `documentum/values.yaml` Helm chart file.

For example:

```
daService:
  enable: true
  serviceName: da-svc
  servicePort: 8080
```

5. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where
Helm charts are extracted>/documentum-components.yaml --namespace <name of
namespace>
```

where <config> is -extra-large-enhanced, -extra-large-standard, -large-enhanced, -large-standard, -medium-enhanced, -medium-large-enhanced, -medium-large-standard, -medium-standard, -small-enhanced, -small-medium-enhanced, -small-medium-standard, -small-standard, or -test-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 install dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/
platforms/cfcf.yaml --values /opt/temp/documentum/dockerimages-values.yaml --
values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /
opt/temp/documentum/documentum-components.yaml --namespace onedctmns
```

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

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

7. Verify the status of the deployment of the pod 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 D2 components on Microsoft Azure**

#### **4.3.1 Kubernetes platform**

##### **4.3.1.1 Overview**

Microsoft Azure is a Cloud computing service for building, testing, deploying, and managing applications and services through Microsoft-managed data centers.

##### **4.3.1.2 Prerequisites**

Ensure that you complete the following activities before you deploy D2 on the Azure environment:

1. Visit the Azure Portal web site to create an Azure Account.
2. 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.

Navigate to **Home > Resource groups > Resource group** and provide the following information:

- **Resource group name**
- **Subscription**
- **Resource group location**

Click **Create**.

- 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.

Navigate to **Home > Container registries > Create container registry** and provide the following information:

- **Registry name**
- **Subscription**
- **Resource group**
- **Location**

- Admin user
- SKU

Click **Create**.

- c. Create an Azure Kubernetes Service (AKS) cluster. AKS is a managed container orchestration service, based on the open source Kubernetes system, which is available on the Azure public cloud.

Navigate to **Home > Kubernetes services > Create Kubernetes cluster** and provide the information in the following tabs:

- **Basics:** Provide valid values for all the mandatory fields such as **PROJECT DETAILS**, **CLUSTER DETAILS**, and so on. Select the **E4s\_V3** with the family as Memory Optimized for the virtual machine size.

Click **Next: Authentication >**.

- **Authentication:** Provide valid values for all the mandatory fields such as **CLUSTER INFRASTRUCTURE** and **KUBERNETES AUTHENTICATION AND AUTHORIZATION**. Enable Role-based access control (RBAC).

Click **Next: Networking >**.

- **Networking:** Disable **HTTP application routing** and set the proper ingress controller. Also, select **Basic** for **Network configuration**.

Click **Next: Monitoring >**.

- **Monitoring:** Enable the container monitoring. Also, select the log analytics workspace.

Click **Next: Tags >**.

- **Tags:** (Optional) Name or value pairs that enable you to categorize resources.

Click **Next: Review + create >**.

- **Review + create:** Review the summary of information and click **Create** to create an AKS.

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

Navigate to **Home > Azure Database for PostgreSQL servers > PostgreSQL server > Pricing tier**. Provide valid values for all the mandatory fields.

Click **Create**.



**Note:** Microsoft Azure documentation contains detailed information.

3. Choose to configure Azure on Microsoft Windows or Linux VM:
4. Configure Azure on Linux VM:

- a. Configure Azure using the following commands:

```
az aks install-cli  
sudo yum update azure-cli  
sudo sh -c 'echo -e "[azure-cli]\nname=Azure CLI\nbaseurl=  
https://packages.microsoft.com/yumrepos/azure-\nbaseurl=  
1\\ngpgcheck=1\\ ngpkey=https://packages.microsoft.com/keys/microsoft.asc">  
/etc/yum.repos.d/azure-cli.repo' sudo yum install azure-cli  
az aks get-credentials --resource-group d2rg --name d2aks az login  
az login -u <id>@opentext.com
```

After the configuration, view the configuration of Azure using the following command format:

```
kubectl config view
```

Example output:

```
[root@skvcentos ~]# kubectl config view  
apiVersion: v1  
clusters:  
- cluster:  
  certificate-authority-data: DATA+OMITTED  
  server: https://d2aks-64dadd07.hcp.centralus.azmk8s.io:443  
  name: d2aks  
contexts:  
- context:  
  cluster: d2aks  
  user: clusterUser_d2rg_d2aks  
  name: d2aks  
  current-context: d2aks  
  kind: Config  
  preferences: {}  
  users:  
  - name: clusterUser_d2rg_d2aks  
  user:  
    client-certificate-data: REDACTED  
    client-key-data: REDACTED  
    token: f6d9d3ffefb2e07e23924933731cde6
```

- b. Install Helm on Linux VM.
- c. Download the Helm release from the GitHub website.
- d. Extract the Helm release TAR file to a temporary location using the following command format:

```
tar -zxf helm-xxxx-amd64.tgz
```

Example output:

```
[root@skvcentos ~]# tar -zxf helm-v3.11.0-rc.3-linux-386.tar.gz  
linux-386/  
linux-386/README.md  
linux-386/helm  
linux-386/LICENSE  
[root@skvcentos ~]#
```

- e. Find the Helm binary and move it to the /usr/local/bin/helm folder using the following command format:

```
mv linux-amd64/helm /usr/local/bin/helm
```

5. Configure Azure on Microsoft Windows:

- a. Install Azure client on Windows. Visit the Microsoft support site for Azure client installation instructions.

- b. Run the following command to update C:\Users\<username>\.kube\config:

```
az aks get-credentials --resource-group <resource group name> --name <AKS name>
```

Open the C:\Users\<username>\.kube\config file, check the **contexts**: section. You can set current-context to be the Azure context and you should also specify namespace: that you want to deploy D2 in for the Azure context after you create the namespace.

- c. Install Docker on Windows. Follow the instructions on the Docker web site.
- d. Install Kubernetes Client on Windows. Follow the instructions on the Kubernetes web site. If you already had Kubernetes client installed, check for updates. You might need to update Kubernetes client in order to work with the newer Azure Kubernetes servers.
- e. Download the helm release for Windows from <https://github.com/helm/helm/releases> and move the helm.exe to the location that is on your Windows environment variable Path.

6. You can use the default namespace or create a namespace for D2 deployment. To create a D2 namespace, create a namespace-d2.json with the following content:

```
{
  "apiVersion": "v1",
  "kind": "Namespace",
  "metadata": {
    "name": "d2",
    "labels": {
      "name": "d2"
    }
  }
}
```

Then run the following command:

```
kubectl create -f namespace-d2.json
```

7. Fetch the details of the resource using the following command format:

```
az aks show --resource-group <resource group name>
--name <name of the cluster>
--query nodeResourceGroup -o tsv
```

Example output:

```
[root@skvcentos ~]# az aks show --resource-group d2rg
--name d2aks
--query nodeResourceGroup -o tsv
MC_d2rg_d2aks_centralus
[root@skvcentos ~]#
```

8. 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@skvcentos ~]# az storage account create
--resource-group MC_d2rg_d2aks_centralus
```

```

name d2storageacc --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@skvcentos ~]#

```

9. 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.

- a. Sample storage class yaml is shown below:

```

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

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

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

11. Update the azurefile storage class in the anchor tags section of documentum/values.yaml as follows:

```
rwoStorage: &rwo_storage_class azurefile
rwmStorage: &rwm_storage_class azurefile
```

12. Load the image into Azure.

- 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 &
```

Example output:

```
sudo /usr/bin/dockerd --insecure-registry 10.194.42.173:5000
--insecure-registry 10.8.176.180:5000
--insecure-registry 10.8.146.181:80
--insecure-registry 10.8.176.180:5000
--insecure-registry 10.9.56.50:80
--insecure-registry docker.mcr.azurecr.io
--insecure-registry 10.9.57.7 -H unix:///var/run/docker.sock
--init 2>&1 &
```

- b. Download or pull all the CS/D2/PE/OTDS images into the local registry using the following command format. Please follow “[Deploying and configuring D2 components on private cloud](#)” on page 117 to understand the list of Docker images that you will need for your deployment. For example, you might not need PE or OTDS Docker images if you don’t need process engine or OTDS for authentication.

```
docker pull registry.opentext.com/<image-name>:<image-tag>
```

- c. Tag the image to the Azure registry specific image using the following command format:

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

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

```
az acr login --name <azure container registry>
Example output:
[root@csazure ~]# az acr login --name d2cr
Login Succeeded
WARNING! Your password will be stored unencrypted in
/root/.docker/config.json.
Configure a credential helper to remove this warning.
[root@csazure ~]#
```

- e. Push the image to the Azure Container Registry (ACR) using the following command format:

```
docker push <image>
```

13. Configure Nginx-Ingress so the applications are accessible externally. Use the following command to deploy the Ingress controller:

```
helm install <name of ingress resource> stable/nginx-ingress --namespace
<namespace> --set controller.replicaCount=1
```

Verify that the Ngnix-Ingress has been deployed successfully and service has been created.

- Configure an FQDN for the ingress controller IP address. You can run the following Bash script to map the external IP to the DNS name or run the command line-by-line in the Windows environment.



**Note:** Provide the IP and the DNS name details. You can find the Public IP address from `kubectl get svc` when the `helm install ingress` command is run.

```
#!/bin/bash
# Public IP address of your ingress controller
IP="52.188.221.96"
# Name to associate with public IP address
DNSNAME="d2deployment"
# Get the resource-id of the public ip
PUBLICIPID=$(az network public-ip list --query "[?ipAddress!=null] | [?contains(ipAddress, '$IP')].[id]" --output tsv)
# Update public ip address with DNS name
az network public-ip update --ids $PUBLICIPID --dns-name $DNSNAME
```

#### 14. Install cert-manager controller:

```
consoleCopy
# Install the CustomResourceDefinition resources separately
kubectl apply --validate=false -f https://raw.githubusercontent.com/jetstack/cert-
manager/release-0.13/deploy/manifests/00-crds.yaml

# Label the ingress-basic namespace to disable resource validation
kubectl label namespace d2 cert-manager.io/disable-validation=true

# Add the Jetstack Helm repository
helm repo add jetstack https://charts.jetstack.io

# Update your local Helm chart repository cache
helm repo update

# Install the cert-manager Helm chart
helm install --namespace <name of the namespace> --generate-name
```

#### 15. Create a CA cluster issuer.

Before certificates can be issued, cert-manager requires an Issuer or ClusterIssuer resource. These Kubernetes resources are identical in functionality, however Issuer works in a single namespace, and ClusterIssuer works across all namespaces. For more information, see the cert-manager issuer documentation.

Create a cluster issuer, such as `cluster-issuer.yaml`, using the following example manifest. Update the email address with a valid address from your organization:

```
apiVersion: cert-manager.io/v1alpha2
kind: ClusterIssuer
metadata:
  name: letsencrypt
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: MY_EMAIL_ADDRESS
    privateKeySecretRef:
      name: letsencrypt
    solvers:
```

```
- http01:
  ingress:
    class: nginx
```

To create the issuer, use the `kubectl apply` command:

```
consoleCopy
kubectl apply -f cluster-issuer.yaml --namespace d2
```

16. Update the ingress helm chart by modifying the `ingress/templates/ingress.yaml` to add the following lines:

Add `cert-manager.io/cluster-issuer: letsencrypt` under annotations.

Add the `tls` section under `spec` section at the same level as `rules` section.

```
tls:
- hosts:
  - {{ .Values.ingress.hostShortName }}.{{ .Values.ingress.hostDomainName }}
  secretName: tls-secret
```

17. To enable ingress resources, ensure that you set the value of `enabled` to true in the `dctm-ingress` and `appworks-gateway` ingress category in `documentum/values.yaml` and update the DNS names for the ingress host values as mapped in the previous step.
18. Enable and update the service names for the required ingress resources in the `dctm-ingress` and `appworks-gateway` ingress category in `ce-d2/values.yaml`. 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
```



**Note:** Make sure that `jmsBase` is disabled (value is set to false).

19. If necessary, update the AKS ingress annotation values in the `documentum/platforms/azure.yaml` file according to your platform's ingress.

### 4.3.1.3 Deploying D2 Components

Follow the steps documented in “[Deploying D2 Components on private cloud](#)” on page 121.

### 4.3.1.4 Limitations

- Host name must have the fully qualified domain name (FQDN) and must not be greater than 59.
- You must change the storage class based on 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.
- Microsoft imposes a 2GB file size limit on content that passes through their web application firewall.

### 4.3.1.5 Troubleshooting

Symptom	Cause	Fix
When you configure Azure on a Linux VM, it results in the [root@skvcentos ~]# az aks browse --resource-group d2rg --name d2aks Merged "d2aks" 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:  1 export https_proxy=<proxy_value>:<port>

## 4.3.2 Red Hat OpenShift platform

### 4.3.2.1 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 create pull secret in order to pull 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 the pull secret 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 images from OpenText Container Registry. See [step 8 in “Deploying and configuring D2 components on private cloud” on page 117](#) for more information.

- 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 d2-<version>.tgz Helm chart from OpenText My Support and unzip the Helm package.

### 4.3.2.2 Deploying Documentum D2

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":
{"kubernetes   s.io/cluster-service":"true"},"name":"default"},"parameters":
```

```
{"cachingmode": "ReadOnly", "kind": "Managed", "storageaccounttype": "StandardSSD_LRS"}, "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_LRS"}, "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. Ensure that you set the value of `pvc size` to `256Gi` for the `ReadWriteMany storageClass`.



**Note:** This is not needed for the `ReadWriteOnce` storage class specified in Step 2b.

You can do this by setting the `pvc size` for all the components in the `d2-resources-values<scale>.yaml` to `256Gi` before deploying in Azure Redhat OpenShift cluster as follows:

```
persistentVolume:  
  size: 256Gi  
  
persistentVolumeClaim:  
  size: 256Gi
```

For all non-D2 components, the *PersistentVolume Size* is added into the resources-values<scale>.yaml. You can modify the size in the resources-values file according to your deployment.

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. Update the `tcp_route` entry of `ExtCS` section in `documentum/values.yaml` with the external IP address.

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. Configure the security context for all D2 pods by setting `uid` in `runAsUser` and `gid` in `fsGroup` to 1000 in the `statefulset` template `yaml` files of the D2 sub-charts as follows:

```

spec:
  securityContext:
    runAsUser: 1000
    fsGroup: 1000

```

7. Set the following anchor tags to `true` in `d2\values.yaml`:

```

#####-Open Shift section--#####
openshiftEnable: &openshift_enable true
openshiftTls: &openshifttls_enable true

```

8. Follow the instructions in [step 8](#) from “Deploying D2 Components in a Private Cloud” to deploy components.

### 4.3.2.3 Limitations

There are no limitations for this release.

### 4.3.2.4 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.
When you run the VolumeBinding prebind plugin 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 <i>pvcAccessModes</i> to ReadWriteMany and the value for PVC <i>size</i> to 256Gi.
When you try to deploy D2, you receive the following 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.	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.	<p>Do one of the following tasks:</p> <ul style="list-style-type: none"> <li>• Edit the machineSet YAML files and deploy the size as Standard_D32s_v3 using the following command format:</li> </ul> <pre>oc scale -- replicas=0 machineset &lt;machinesetname&gt; oc edit machineset &lt;machinesetname&gt; oc scale -- replicas=1 machineset &lt;machinesetname&gt;</pre> <ul style="list-style-type: none"> <li>• Increase the node resource.</li> </ul>

Symptom	Cause	Fix
The oc get command fails.	Pending certificate signing request (CSR) for approval.	<p>Verify if any CSRs are pending using the following command format:</p> <pre>oc get csr</pre> <p>Approve the pending CSR (if any) using the following command format:</p> <pre>oc get csr -o name   xargs oc adm certificate approve</pre>
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.

## 4.4 Deploying and configuring Documentum Records Client on Microsoft Azure cloud platform

### 4.4.1 Kubernetes platform

#### 4.4.1.1 Prerequisites

1. Perform all the steps as described in “Prerequisites” on page 261 in “Deploying and configuring Documentum Server on Microsoft Azure cloud platform” on page 261.
2. Deploy the Documentum Server pod as described in “Deploying and configuring Documentum Server on private cloud” on page 56.
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 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	23.4.0 or 23.4
dctm-records-darinstalation	23.4.0 or 23.4
dctm-rqm	23.4.0 or 23.4

4. Download the Helm charts from My Support.

#### 4.4.1.2 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 261.

Make sure that you deploy the Documentum Records Client Docker image (Oracle Linux only) and Helm charts.

#### 4.4.1.3 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.

#### 4.4.1.4 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>

## 4.5 Deploying and configuring Documentum Foundation Services on Microsoft Azure cloud platform

### 4.5.1 Kubernetes platform

#### 4.5.1.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 261 in “Deploying and configuring Documentum Server on Microsoft Azure cloud platform” on page 261 except for those steps related to the PostgreSQL database.

#### 4.5.1.2 Deploying Documentum Foundation Services

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from [step 2 to step 9](#) in “Deploying Documentum Server” on page 58 in “Deploying and configuring Documentum Server on private cloud” on page 56.
3. Perform the tasks mentioned from [step 3 to step 6](#) in “Deploying Documentum Foundation Services” on page 208 in “Deploying and configuring Documentum Foundation Services on private cloud” on page 208.
4. Update the ingress resource rule for Documentum Foundation Services in the `dctm-server.dctm-ingress` section in the single All-In-One documentum/values.yaml Helm chart file.

For example:

```
dfsService:  
  enable: true  
  serviceName: dfs-server  
  servicePort: 8080
```

5. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values  
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values  
<location where Helm charts are extracted>/dockerimages-values.yaml --values  
<location where Helm charts are extracted>/<config>.yaml --values <location where  
Helm charts are extracted>/documentum-components.yaml --namespace <name of  
namespace>
```

where `<config>` is `-extra-large-enhanced`, `-extra-large-standard`, `-large-enhanced`, `-large-standard`, `-medium-enhanced`, `-medium-large-enhanced`, `-medium-large-standard`, `-medium-standard`, `-small-enhanced`, `-small-medium-enhanced`, `-small-medium-standard`, `-small-standard`, or `-test-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 install dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/  
platforms/cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --
```

```
values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /  
opt/temp/documentum/documentum-components.yaml --namespace onedctmns
```

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

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

7. Verify the status of the deployment of the pod 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 Foundation 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>
```

#### **4.5.1.3 Limitations**

There are no limitations for this release.

#### **4.5.1.4 Troubleshooting**

There are no troubleshooting information for this release.

## **4.6 Deploying and configuring Documentum REST Services on Microsoft Azure cloud platform**

### **4.6.1 Kubernetes platform**

#### **4.6.1.1 Prerequisites**

Perform all the steps as described in “Prerequisites” on page 261 in “Deploying and configuring Documentum Server on Microsoft Azure cloud platform” on page 261 except for those steps related to the PostgreSQL database.

### 4.6.1.2 Deploying Documentum REST Services

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from **step 2** to **step 9** in “Deploying Documentum Server” on page 58 in “Deploying and configuring Documentum Server on private cloud” on page 56.
3. Perform the tasks mentioned from **step 4** to **step 7** in “Deploying Documentum REST Services” on page 217 in “Deploying and configuring Documentum REST Services on private cloud” on page 217.
4. Update the ingress resource rule for Documentum REST Services in the `dctm-server.dctm-ingress` section in the single All-In-One `documentum/values.yaml` Helm chart file.

For example:

```
restService:
  enable: true
  serviceName: dctm-rest
  servicePort: 8080
```

5. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where
Helm charts are extracted>/documentum-components.yaml --namespace <name of
namespace>
```

where `<config>` is `-extra-large-enhanced`, `-extra-large-standard`, `-large-enhanced`, `-large-standard`, `-medium-enhanced`, `-medium-large-enhanced`, `-medium-large-standard`, `-medium-standard`, `-small-enhanced`, `-small-medium-enhanced`, `-small-medium-standard`, `-small-standard`, or `-test-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 install dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/
platforms/cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --
values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /
opt/temp/documentum/documentum-components.yaml --namespace onedctmns
```

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

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

7. Verify the status of the deployment of the pod 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>
```

#### **4.6.1.3 Configuring external IP address**

Perform all the steps as described in “Deploying Documentum Server” on page 264.

#### **4.6.1.4 Limitations**

There are no limitations for this release.

#### **4.6.1.5 Troubleshooting**

There are no troubleshooting information for this release.

### **4.6.2 Red Hat OpenShift platform**

#### **4.6.2.1 Prerequisites**

Perform all the steps as described in “Prerequisites” on page 294 in “Kubernetes platform” on page 294.

#### **4.6.2.2 Deploying Documentum REST Services**

Perform all the steps as described in “Deploying Documentum REST Services” on page 295.

#### **4.6.2.3 Configuring external IP address**

Perform all the steps as described in “Deploying Documentum Server” on page 264.

#### **4.6.2.4 Limitations**

There are no limitations for this release.

#### **4.6.2.5 Troubleshooting**

There are no troubleshooting information for this release.

## 4.7 Deploying and configuring Documentum Content Management Interoperability Services on Microsoft Azure cloud platform

### 4.7.1 Kubernetes platform

#### 4.7.1.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 261 in “Deploying and configuring Documentum Server on Microsoft Azure cloud platform” on page 261 except for those steps related to the PostgreSQL database.

#### 4.7.1.2 Deploying Documentum Content Management Interoperability Services

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from step 2 to step 9 in “Deploying Documentum Server” on page 58 in “Deploying and configuring Documentum Server on private cloud” on page 56.
3. Perform the tasks mentioned from step 3 to step 6 in “Deploying Documentum Content Management Interoperability Services” on page 225 in “Deploying and configuring Documentum REST Services on private cloud” on page 217.
4. Update the ingress resource rule for Documentum Content Management Interoperability Services in the `dctm-server.dctm-ingress` section in the single All-In-One documentum/values.yaml Helm chart file.

For example:

```
cmisService:  
  enable: true  
  serviceName: dctm-cmis  
  servicePort: 8080
```

5. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values  
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values  
<location where Helm charts are extracted>/dockerimages-values.yaml --values  
<location where Helm charts are extracted>/<config>.yaml --values <location where  
Helm charts are extracted>/documentum-components.yaml --namespace <name of  
namespace>
```

where `<config>` is `-extra-large-enhanced`, `-extra-large-standard`, `-large-enhanced`, `-large-standard`, `-medium-enhanced`, `-medium-large-enhanced`, `-medium-large-standard`, `-medium-standard`, `-small-enhanced`, `-small-medium-enhanced`, `-small-medium-standard`, `-small-standard`, or `-test-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 install dcmdeployment /opt/temp/documentum --values /opt/temp/documentum/platforms/cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /opt/temp/documentum/documentum-components.yaml --namespace onedctmns
```

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

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

7. Verify the status of the deployment of the pod 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 Content Management Interoperability 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>
```

#### **4.7.1.3 Configuring external IP address**

Perform all the steps as described in “[Deploying Documentum Server](#)” on page 264.

#### **4.7.1.4 Limitations**

There are no limitations for this release.

#### **4.7.1.5 Troubleshooting**

There are no troubleshooting information for this release.

### **4.8 Deploying and configuring Documentum Reports on Microsoft Azure cloud platform**

## 4.8.1 Kubernetes platform

### 4.8.1.1 Prerequisites

1. Perform all the steps as described in “[Prerequisites](#)” on page 261 in “[Deploying and configuring Documentum Server on Microsoft Azure cloud platform](#)” on page 261.
2. Deploy the Documentum Server pod as described in “[Deploying and configuring Documentum Server on private cloud](#)” on page 56.
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 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-reports-installer	22.4.2
dctm-reports-base	22.4.2

4. Download the Helm charts from My Support.

### 4.8.1.2 Deploying Documentum Reports

1. The information for deploying and configuring Documentum Reports on Azure cloud platform is same as described in “[Deploying and configuring Documentum Server on Microsoft Azure cloud platform](#)” on page 261.

Make sure that you deploy the Documentum Reports Docker image (Oracle Linux only).

2. Open the single All-In-One documentum/values.yaml Helm chart file and in the ingress and dtrbase categories, provide the appropriate values for the variables to pass them to your templates.

“[Deploying Documentum Reports \(dtrinstaller and dtrbase\)](#)” on page 230 contains detailed information about the variables.

#### 4.8.1.3 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.

### 4.9 Deploying and configuring Content Connect on Microsoft Azure cloud platform

#### 4.9.1 Prerequisites

1. Perform all the steps as described in the “[Prerequisites](#)” on page 261 section of “[Deploying and configuring Documentum Server on Microsoft Azure cloud platform](#)” on page 261.
2. Download the Docker image (Alpine 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 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	23.4.0 or 23.4
dctm-content-connect-dbinit	23.4.0 or 23.4

3. Download the Documentum Server Helm charts available in the `documentum_<release-version>.tar` file from My Support page.

## 4.9.2 Deploying Content Connect

**To deploy Content Connect on Azure cloud platform, perform the following steps:**

1. Deploy and configure Content Connect on Azure cloud platform as described in “[Deploying Documentum Server](#)” on page 264.

Make sure that you deploy the Content Connect Docker image (Alpine Linux only).

2. Open the single All-In-One documentum/values.yaml Helm chart file and in the contentconnect section, provide the appropriate values for the variables to pass them to your templates.

The “[contentconnect](#)” on page 246 table contains detailed information about the variables for Content Connect.

### 4.9.2.1 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*.

## 4.9.3 Limitations

There are no limitations for this release.

## 4.9.4 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 http s_proxy=<proxy_value>:<port>
When using Content Connect, Cross-Origin Resource Sharing (CORS) related errors occur even after all configurations are set.	CORS related errors occur.	<ul style="list-style-type: none"> <li>• Increase the load balancer response timeout.</li> <li>• Add the Content Connect Admin Console URL to the rest.cors.allowed.origins tag in the rest-api-runtime.properties file.</li> </ul>

## 4.9.5 Upgrading

Upgrade the Content Connect pod using the helm upgrade command:

```
Helm upgrade <release name> . --values <resource file name> -n <namespace>
```

# 4.10 Deploying and configuring Extended ECM Documentum for Microsoft 365 on Microsoft Azure cloud platform

## 4.10.1 Prerequisites

1. Perform all the steps as described in the “Prerequisites” on page 261 section of “Deploying and configuring Documentum Server on Microsoft Azure cloud platform” on page 261.
2. Download the Docker image (Alpine 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 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-smartviewm365	23.4.0 or 23.4
dctm-smartviewm365customjar	23.4.0 or 23.4

3. Download the Documentum Server Helm charts available in the documentum-<release-version>.tar file from My Support page.

## 4.10.2 Deploying Extended ECM Documentum for Microsoft 365

**To deploy Extended ECM Documentum for Microsoft 365 on Azure cloud platform, perform the following steps:**

1. Deploy and configure Extended ECM Documentum for Microsoft 365 on Azure cloud platform as described in [“Deploying Extended ECM Documentum for Microsoft 365” on page 252](#).  
Make sure that you deploy the Extended ECM Documentum for Microsoft 365 Docker image (Alpine Linux only).
2. Open the single All-In-One documentum/values.yaml Helm chart file and in the smartviewm365 section, provide the appropriate values for the variables to pass them to your templates.

The [“smartviewm365” on page 252](#) table contains detailed information about the variables for Extended ECM Documentum for Microsoft 365.

## 4.10.3 Limitations

There are no limitations for this release.

## 4.10.4 Troubleshooting

There are no troubleshooting information for this release.

## 4.11 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.

### 4.11.1 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 documentum/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 dctm-server.cs-secrets section ([“cs-secrets” on page 60](#)).
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 dctm-server.cs-secrets ([“cs-secrets” on page 60](#)).  
In addition, make sure that you set the value of docbase.existing to true in dctm-server.content-server ([“content-server” on page 72](#)).
3. If you want certificate-based communication in the database, set the value of database.sslEnabled to true in dctm-server.content-server ([“content-server” on page 72](#)).
4. Set the value of database.paasEnv to true in dctm-server.content-server ([“content-server” on page 72](#)).
5. Do not specify the value of docbase.owner in the @<host name created for the database> format in dctm-server.content-server ([“content-server” on page 72](#)).

### 4.11.2 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 documentum/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 dctm-server.cs-secrets ([“cs-secrets” on page 60](#)).
2. If you want to use the existing repository, specify the value of database.userName in the <repository owner name> format in dctm-server.cs-secrets ([“cs-secrets” on page 60](#)).  
In addition, make sure that you set the value of docbase.existing to true in dctm-server.content-server ([“content-server” on page 72](#)).

3. Copy the Flexible server certificate from the Azure website and provide it as a value for database.certificate in `dctm-server.cs-secrets` ("cs-secrets" on page 60).
4. Set the value of database.sslEnabled to true in `dctm-server.content-server` ("content-server" on page 72).
5. Set the value of database.paasEnv to false in `dctm-server.content-server` ("content-server" on page 72).

## 4.12 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,1,app_server_name
SET> oAuthAuthentication
...
OK
API> append,c,1,app_server_uri
SET> http://localhost:9080/oAuthAuthentication/servlet/authenticate
...
OK
API> save,c,1
```

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. Create a GKE cluster.
  - b. Create namespace within the cluster.
  - c. Create the NGNIX ingress controller in the GKE cluster.

7. Download the Documentum Server and the required Documentum application Docker images (Oracle Linux only for all products except for Documentum REST Services, Documentum Foundation Services, and Documentum Content Management Interoperability Services) from OpenText Container Registry. Use Alpine Linux Docker images for Documentum REST Services, Documentum Foundation Services, and Documentum Content Management Interoperability Services. 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 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	23.4.0 or 23.4
dctm-admin	23.4.0 or 23.4
dctm-dfs	23.4.0 or 23.4
dctm-records	23.4.0 or 23.4
dctm-records-darinstallation	23.4.0 or 23.4
dctm-rqm	23.4.0 or 23.4
dctm-reports-installer	22.4.2
dctm-reports-base	22.4.2
dctm-rest	23.4.0 or 23.4
dctm-cmis	23.4.0 or 23.4
dctm-tomcat	23.4.0 or 23.4



**Note:** The `dctm-tomcat` image (Alpine Linux) is required only for Documentum REST Services, Documentum Foundation Services, and Documentum Content Management Interoperability Services.

*“Decoupling Documentum product image from base Tomcat image” on page 420 contains detailed information.*

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. Make sure that there is a storage class to support both the ReadWriteOnce (RWO) and ReadWriteMany (RWX) access modes before you proceed with the deployment.
11. Download the Helm charts available in the documentum-<release-version>.tar file from My Support.

### 5.1.1.2 Deploying Documentum Server

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Open the single All-In-One documentum/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 56](#) 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 externalAccessEnabled to true in common-variables.
  - b. Retain all the default values for ExtDocbroker in dctm-server.docbroker.
  - c. Set the value of ExtCS.nativeExtPort to 80 and sslExtPort to 81 in dctm-server.content-server. Retain the default values for all other variables.
4. Perform all the steps from [step 8](#) to [step 12](#) in [“Deploying Documentum Server” on page 58](#) to complete the deployment and verification of the Documentum Server pod.
5. **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.
6. Run the Helm upgrade command using the following command format:

```
helm upgrade <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where
Helm charts are extracted>/documentum-components.yaml --namespace <name of
namespace>
```

where <config> is -extra-large-enhanced, -extra-large-standard, -large-enhanced, -large-standard, -medium-enhanced, -medium-large-enhanced, -medium-large-standard, -medium-standard, -small-enhanced, -small-medium-enhanced, -small-medium-standard, -small-standard, or -test-small resource value YAML file that has been provided. These resource files contain pod sizing values like CPU, memory, and so on.

7. 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
```

### 5.1.1.3 Limitations

There are no limitations for this release.

### 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 307 in “Deploying and configuring Documentum Server on GCP” on page 307 except for those steps related to the PostgreSQL database.

### 5.2.1.2 Deploying Documentum Administrator

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from **step 2** to **step 9** in “Deploying Documentum Server” on page 58 in “Deploying and configuring Documentum Server on private cloud” on page 56.
3. Perform the tasks mentioned from **step 3** to **step 6** in “Deploying Documentum Administrator” on page 111 in “Deploying and configuring Documentum Administrator on private cloud” on page 111.
4. Update the ingress resource rule for Documentum Administrator in the `dctm-server.dctm-ingress` section in the single All-In-One `documentum/values.yaml` Helm chart file.

For example:

```
daService:
  enable: true
  serviceName: da-svc
  servicePort: 8080
```

5. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where
Helm charts are extracted>/documentum-components.yaml --namespace <name of
namespace>
```

where `<config>` is `-extra-large-enhanced`, `-extra-large-standard`, `-large-enhanced`, `-large-standard`, `-medium-enhanced`, `-medium-large-enhanced`, `-medium-large-standard`, `-medium-standard`, `-small-enhanced`, `-small-medium-enhanced`, `-small-medium-standard`, `-small-standard`, or `-test-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 install dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/
platforms/cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --
values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /
opt/temp/documentum/documentum-components.yaml --namespace onedctmns
```

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

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

7. Verify the status of the deployment of the pod 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 D2 components on Google Cloud Platform

### 5.3.1 Overview

Google Cloud Platform (GCP), is a suite of cloud computing services that runs on the Google infrastructure. Alongside a set of management tools, it provides a series of modular cloud services including computing, data storage, data analytics and machine learning.

### 5.3.2 Prerequisites

Ensure that you complete the following activities before you deploy D2 on Google Cloud Platform (GCP) environment:

1. Download and configure the Docker application from the Docker website.  
The *Docker Documentation* contains the instructions.
2. Download and configure the Helm (client) application from the Helm website in the cluster namespace (your Cloud Shell machine).  
The *Helm Documentation* contains detailed information.
3. Download and configure the Kubernetes application from the Kubernetes website.  
The *Kubernetes Documentation* contains detailed information.
4. Download the latest version of Google Cloud SDK from the Google Cloud Platform website and install it on the machine that includes the gcloud command line utility.
5. Create a Google Kubernetes Engine (GKE) cluster 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 steps:
    - i. Select a standard cluster template.
    - ii. Provide a name to your cluster. For example, demo-cluster.
    - iii. Select **Zonal** for the location type. You can use Regional for production grade-high available clusters.
    - iv. Select a Zone closer to your location. The Google Cloud Platform website contains the list of zones.
    - v. 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.
    - vi. 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. Click the **ENTER** key at the command prompt that is displayed. 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
```
6. Upload the Docker images to Google Container Registry (GCR).
- a. Use the following command format to configure Docker to use gcloud as a credential helper:
 

```
gcloud auth configure-docker
```
  - b. Tag your Docker images using the following command format:
 

```
[HOSTNAME] / [PROJECT-ID] / [IMAGE]
```

For example:

```
gcr.io/documentum-d2-product/contentserver/centos/stateless/cs:20.2.0152
```
  - c. Upload the tagged image to GCR. For example:
 

```
docker push gcr.io/documentum-d2-product/contentserver/centos/stateless/cs:20.2.0152
```
7. Verify the version of Helm using the following command format:
- ```
abc@cloudshell:~ (dctm-d2)$ ./helm version
```
- Example output:

```
version.BuildInfo{Version:"v3.2.1",
GitCommit:"fe51cd1e31e6a202cba7dead9552a6d418ded79a", GitTreeState:"clean",
GoVersion:"go1.13.10"}
```

8. Extract the Helm package you downloaded to a temporary location.
9. Build the NGINX Plus Ingress image:
  - a. Download the Kubernetes NGINX Plus Ingress code:
    - git clone: <https://github.com/nginxinc/kubernetes-ingress/>
    - cd kubernetes-ingress/
    - git checkout v1.11.1
  - b. Download the NGINX Plus certificate (`nginx-repo.crt`) and public key (`nginx-repo.key`) from the email from F5 Inc..
  - c. Place `nginx-repo.crt` and `nginx-repo.key` under the `kubernetes-ingress` directory.
  - d. Copy the `nginx-repo.crt` and `nginx-repo.key` files to the `/etc/ssl/nginx/` directory:
 

```
sudo cp nginx-repo.crt /etc/ssl/nginx/
sudo cp nginx-repo.key /etc/ssl/nginx/
```
  - e. Run the following command to build the NGINX Plus Ingress image:
 

```
make debian-image-plus PREFIX=hydarprd01.lab.opentext.com/ecm-devdocker/nginx/
nginx-ingress-plus TARGET=container****
```
10. Deploy NGINX Plus Ingress:
  - a. Charts are available in the same repo that was downloaded under `kubernetes-ingress/deployments/helm-chart`

```
[root@csazure helm-chart]# ls -lrt
total 64
rw-rr- 1 root root 493 May 13 16:27 Chart.yaml
rw-rr- 1 root root 8165 May 13 16:27 chart-icon.png
rw-r-r- 1 root root 16117 May 13 16:27 values.yaml
rw-rr- 1 root root 92 May 13 16:27 values-plus.yaml
rw-rr- 1 root root 393 May 13 16:27 values-icp.yaml
drwxr-xr-x 2 root root 4096 May 13 16:27 templates
rw-rr- 1 root root 18534 May 13 16:28 README.md
drwxr-xr-x 2 root root 4096 May 13 16:28 crds
[root@csazure helm-chart]#
```
  - b. Under the `helm-chart` directory, run the following command to install NGINX Plus Ingress:
 

```
helm install <nginx-plus> . -n <namespace> --set controller.replicaCount=2 --
set controller.image.repository=<nginx/nginx-ingress-plus> --set
controller.nginxplus=true
```

For example:

```
helm install <nginx-plus> . -n d2poc --set controller.replicaCount=2 --set
controller.image.repository=<nginx/nginx-ingress-plus> --set
controller.nginxplus=true
```

Once the NGINX Plus Ingress pods are running, you can deploy dctm-ingress, which is part of D2 helm charts.

- Under annotation of dctm-ingress add the following line:

```
nginx.com/sticky-cookie-services: "serviceName={{ .Values.userName }}d2classic
srv_id expires=1h path=/D2"
```

For example:

```
nginx.com/sticky-cookie-services:"serviceName=d2d2classic srv_id expires=1h
path=/D2;serviceName=d2d2smartview srv_id expires=1h path=/D2-Smartview"
```

- Save the Ingress.

## 11. Create a Google Cloud Filestore instance.

In the Filestore of the Google Cloud Platform dashboard, click **CREATE INSTANCE** and perform the following:

- Provide a name to the Google Cloud Filestore instance. For example, `demogcfs`
- 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 once 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.

- Download the external nfs-client-provisioner Helm chart from the Github website to your Cloud Shell machine.
- Install the nfs-client-provisioner Helm chart by passing 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 Persistent Volume Claims must 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 AGE
demo-nfs-client-provisioner 1s
==> v1/Pod(related)
NAME READY...
demo-nfs-client-provisioner-76986844 0/1...
==> v1/StorageClass
NAME AGE
gcp-rwx 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 Persistent Volume is created using the following command format:

```
$ kubectl get pv
```

Ensure the status of PV is Bound.

15. Delete the PVC and check if the PV is also deleted.
16. Download the required D2 Images from the OpenText registry and push them to the Google container registry. Download the Helm Chart TAR file from OpenText MySupport.

### 5.3.3 Accessing the Webapp from Outside the Cluster

Now that the ingress controller is deployed and the ingress resources defined, you must locate the nginx-ingress load balancer service's external IP address and map it to the host name that you specified as the *host* property in the *ingress.yaml* file in C:\Windows\System32\drivers\etc\hosts.

```
hding@cloudshell:~ )documentum-d2-product)$ kubectl get svc | grep nginx-ingress
nginx-ingress           LoadBalancer   10.40.18.126   34.73.83.119   80:32474/TCP,
443:1034/TCP    7d20h
```

```
c:\Windows\System32\drivers\etc\hosts
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97      rhino.acme.com          # source server
#      38.25.63.10      x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
#      127.0.0.1      localhost
#      ::1            localhost
34.73.83.119      hdingttest.com
```

You can now access the webapp at <http://<hostname>/<webapp path>>. For example: <http://hdingttest.com/D2>.

### 5.3.4 Deploying D2 Components

Follow the steps documented in “[Deploying D2 Components on private cloud](#)” on page 121.



#### Notes

- If you are enabling root squash for the volumes in your D2 deployment in GCP, remove sudo from the command entries in all the extraInitContainers sections of dockerimages-values.yaml. For example, remove sudo from the command section shown below:

```
extraInitContainers:
  - name: d2installerinit
    image: artifactory.oxylab.net/bpdockerhub/dctm-d2pp-installer-ol:23.2.0000.0143
    imagePullPolicy: *pull_policy_type
    command: ['/bin/sh', '-c', 'yes |sudo cp -rf /customscripts/* /opt/dctm_docker/customscriptpvc/']
```

- If you are using Google Cloud SQL Postgres instance for the appworks DB, For the database host, provide the Google Cloud SQL instance Private/Public IP Address obtained from the Google Cloud SQL console. Update

appworksdb user as default admin user (postgres) and password respectively. This will create db and awg user automatically. If you don't want to use default admin user, create a new user manually and grant admin privileges using below commands.

```
CREATE ROLE <new_user> WITH LOGIN PASSWORD '<new_password>' CREATEDB;
GRANT <new_user> to postgres;

database:
  vendor: PostgreSQL
  server:
    host: <Private/Public IP Address>
    port: 5432

appworksdb:
  user: gateway201user
  password: password
  database: gateway201db
```

### 5.3.5 Setting up HTTP(S) Load Balancing with Ingress

When we create an Ingress Kubernetes object in GKE, we instruct GKE to create an Ingress resource. GKE makes appropriate Google Cloud API calls to create an external HTTP(S) load balancer. The load balancer's URL map's host rules and path matchers reference one or more backend services. Each backend service corresponds to a GKE Service of type NodePort, as referenced in the Ingress.

After D2 is deployed, you can log into **Google Cloud Platform Console**, under **Networking category >Network Services > Load Balancing**, and click your load balancer instance.

Your load balancer should have Protocol set to HTTP, name should contain your ingress object name in it. You can edit the backend configuration and frontend configuration for your HTTP(S) load balancer instance to enable session stickiness and HTTPS. For more detailed information on how to configure the HTTP(s) load balancer, please refer to Google documentation.

### 5.3.6 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 Persistent Volume Claim (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.

## 5.4 Deploying and configuring Documentum Records Client on GCP

### 5.4.1 Kubernetes platform

#### 5.4.1.1 Prerequisites

1. Perform all the steps as described in “Prerequisites” on page 307 in “Deploying and configuring Documentum Server on GCP” on page 307.
2. Deploy the Documentum Server pod as described in “Deploying and configuring Documentum Server on GCP” on page 307.
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 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                 | 23.4.0 or 23.4 |
| dctm-records-darinstallation | 23.4.0 or 23.4 |
| dctm-rqm                     | 23.4.0 or 23.4 |

4. Download the Helm charts from My Support.

#### 5.4.1.2 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 My Support to a temporary location.
3. Open the single All-In-One documentum/values.yaml Helm chart file and in the records section, provide the appropriate values for the variables to pass them to your templates.

*“Deploying and configuring Documentum Records Client on private cloud”* on page 194 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>/  
documentum -f <location where Helm charts are extracted>/documentum/platforms/  
<cloud platform>.yaml --namespace <name of namespace>
```

For example:

```
helm install --name records /opt/temp/Helm-charts/documentum -f <location where  
Helm charts are extracted>/documentum/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.

#### 5.4.1.3 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.

#### 5.4.1.4 Troubleshooting

There are no troubleshooting information for this release.

## 5.5 Deploying and configuring Documentum Foundation Services on GCP

### 5.5.1 Kubernetes platform

#### 5.5.1.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 307 in “Deploying and configuring Documentum Server on GCP” on page 307 except for those steps related to the PostgreSQL database.

#### 5.5.1.2 Deploying Documentum Foundation Services

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from step 2 to step 9 in “Deploying Documentum Server” on page 58 in “Deploying and configuring Documentum Server on private cloud” on page 56.
3. Perform the tasks mentioned from step 3 to step 6 in “Deploying Documentum Foundation Services” on page 208 in “Deploying and configuring Documentum Foundation Services on private cloud” on page 208.
4. Update the ingress resource rule for Documentum Foundation Services in the dctm-server.dctm-ingress section in the single All-In-One documentum/values.yaml Helm chart file.

For example:

```
dfsService:  
  enable: true  
  serviceName: dfc-server  
  servicePort: 8080
```

5. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values  
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values  
<location where Helm charts are extracted>/dockeimages-values.yaml --values  
<location where Helm charts are extracted>/<config>.yaml --values <location where  
Helm charts are extracted>/documentum-components.yaml --namespace <name of  
namespace>
```

where <config> is -extra-large-enhanced, -extra-large-standard, -large-enhanced, -large-standard, -medium-enhanced, -medium-large-enhanced, -medium-large-standard, -medium-standard, -small-enhanced, -small-medium-enhanced, -small-medium-standard, -small-standard, or -test-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 install dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/  
platforms/ofcr.yaml --values /opt/temp/documentum/dockeimages-values.yaml --  
values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /  
opt/temp/documentum/documentum-components.yaml --namespace onedctmns
```

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

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

7. Verify the status of the deployment of the pod 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 Foundation 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>
```

#### 5.5.1.3 Limitations

There are no limitations for this release.

#### 5.5.1.4 Troubleshooting

There are no troubleshooting information for this release.

## 5.6 Deploying and configuring Documentum REST Services on GCP

### 5.6.1 Kubernetes platform

#### 5.6.1.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 307 in “Deploying and configuring Documentum Server on GCP” on page 307 except for those steps related to the PostgreSQL database.

### 5.6.1.2 Deploying Documentum REST Services

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from [step 2](#) to [step 9](#) in “[Deploying Documentum Server](#)” on page 58 in “[Deploying and configuring Documentum Server on private cloud](#)” on page 56.
3. Perform the tasks mentioned from [step 4](#) to [step 7](#) in “[Deploying Documentum REST Services](#)” on page 217 in “[Deploying and configuring Documentum REST Services on private cloud](#)” on page 217.
4. Update the ingress resource rule for Documentum REST Services in the `dctm-server.dctm-ingress` section in the single All-In-One `documentum/values.yaml` Helm chart file.

For example:

```
restService:  
  enable: true  
  serviceName: dctm-rest  
  servicePort: 8080
```

5. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values  
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values  
<location where Helm charts are extracted>/dockerimages-values.yaml --values  
<location where Helm charts are extracted>/<config>.yaml --values <location where  
Helm charts are extracted>/documentum-components.yaml --namespace <name of  
namespace>
```

where `<config>` is `-extra-large-enhanced`, `-extra-large-standard`, `-large-  
enhanced`, `-large-standard`, `-medium-enhanced`, `-medium-large-enhanced`,  
`-medium-large-standard`, `-medium-standard`, `-small-enhanced`, `-small-medium-  
enhanced`, `-small-medium-standard`, `-small-standard`, or `-test-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 install dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/  
platforms/cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --  
values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /  
opt/temp/documentum/documentum-components.yaml --namespace onedctmns
```

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

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

7. Verify the status of the deployment of the pod 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>
```

### 5.6.1.3 Configuring external IP address

Perform all the steps as described in “Deploying Documentum Server” on page 264.

### 5.6.1.4 Limitations

There are no limitations for this release.

### 5.6.1.5 Troubleshooting

There are no troubleshooting information for this release.

## 5.7 Deploying and configuring Documentum Content Management Interoperability Services on GCP

### 5.7.1 Kubernetes platform

#### 5.7.1.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 307 in “Deploying and configuring Documentum Server on GCP” on page 307 except for those steps related to the PostgreSQL database.

#### 5.7.1.2 Deploying Documentum Content Management Interoperability Services

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from step 2 to step 9 in “Deploying Documentum Server” on page 58 in “Deploying and configuring Documentum Server on private cloud” on page 56.
3. Perform the task mentioned in step 6 in “Deploying Documentum Content Management Interoperability Services” on page 225 in “Deploying and configuring Documentum Content Management Interoperability Services on private cloud” on page 225.
4. Update the ingress resource rule for Documentum Content Management Interoperability Services in the `dctm-server.dctm-ingress` section in the single All-In-One `documentum/values.yaml` Helm chart file.

For example:

```
cmisService:  
  enable: true
```

```
serviceName: dcm-cmis
servicePort: 8080
```

5. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where
Helm charts are extracted>/documentum-components.yaml --namespace <name of
namespace>
```

where `<config>` is `-extra-large-enhanced`, `-extra-large-standard`, `-large-enhanced`, `-large-standard`, `-medium-enhanced`, `-medium-large-enhanced`, `-medium-large-standard`, `-medium-standard`, `-small-enhanced`, `-small-medium-enhanced`, `-small-medium-standard`, `-small-standard`, or `-test-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 install dcmdeployment /opt/temp/documentum --values /opt/temp/documentum/
platforms/cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --
values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /
opt/temp/documentum/documentum-components.yaml --namespace onedctmn
```

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

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

7. Verify the status of the deployment of the pod 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 Content Management Interoperability 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>
```

### 5.7.1.3 Limitations

There are no limitations for this release.

### 5.7.1.4 Troubleshooting

There are no troubleshooting information for this release.

## 5.8 Deploying and configuring Documentum Reports on GCP

### 5.8.1 Kubernetes platform

#### 5.8.1.1 Prerequisites

1. Perform all the steps as described in “Prerequisites” on page 307 in “Deploying and configuring Documentum Server on GCP” on page 307.
2. Deploy the Documentum Server pod as described in “Deploying and configuring Documentum Server on GCP” on page 307.
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 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-reports-installer | 22.4.2    |
| dctm-reports-base      | 22.4.2    |

4. Download the Helm charts from My Support.

### 5.8.1.2 Deploying Documentum Reports

1. Upload the Docker image to Google Container Registry (GCR). 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/reports:<release-version>
```

- b. Upload the tagged Docker image to GCR.

For example:

```
docker push gcr.io/documentum-d2-product/reports:<release-version>
```

2. Open the single All-In-One documentum/values.yaml Helm chart file and in the ingress and dtrbase categories, provide the appropriate values for the variables to pass them to your templates.

[“Deploying Documentum Reports \(dtrinstaller and dtrbase\)” on page 230](#) contains detailed information about the variables.

### 5.8.1.3 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.

#### 5.8.1.4 Troubleshooting

There are no troubleshooting information for this release.

## 5.9 Deploying and configuring Documentum Workflow Designer on GCP

### 5.9.1 Kubernetes platform

#### 5.9.1.1 Prerequisites

1. Perform all the steps from [step 1](#) to [step 11](#) in “[Deploying and configuring Documentum Server on GCP](#)” on page 307.
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/v1beta1
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 236.

## 5.9.2 Deploying Documentum Workflow Designer

1. Perform all the steps as described in “Deploying Documentum Workflow Designer” on page 237 in “Deploying and configuring Documentum Workflow Designer on private cloud” on page 236.

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>
```

### 5.9.3 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.

#### 5.9.3.1 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.

#### 5.9.3.2 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:

| Name                                             | Description                                                                                                                                                                                                                |
|--------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>serviceAccount.createServiceaccount</code> | Specifies whether to create a new service account or not. The default value is <code>false</code> . Set this attribute to <code>true</code> to create a new service account.                                               |
| <code>serviceAccount.serviceAccountName</code>   | Specifies the service account name. This variable uses the value that you specified for <code>documentumServiceAccount</code> in <code>common-variables</code> in <a href="#">"Defining common variables" on page 53</a> . |
| <code>prefix</code>                              | Prefix for name or ID of the utility pod. Use a suitable prefix to distinguish WFD client utility pod in the cluster environment.                                                                                          |
| <code>image.repository</code>                    | Path of the repository.                                                                                                                                                                                                    |

| Name                                               | Description                                                                                                                                                                                            |
|----------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| image.name                                         | Name of the image.                                                                                                                                                                                     |
| image.tag                                          | Tag as a version-specific number.                                                                                                                                                                      |
| image.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. |
| image.pullSecrets                                  | Secret for fetching images.                                                                                                                                                                            |
| workflowDesigner.url                               | Workflow Designer URL where packages needs to be imported.                                                                                                                                             |
| workflowDesigner.username                          | Workflow Designer username to connect to the repository.                                                                                                                                               |
| workflowDesigner.password                          | Workflow Designer password to connect to the repository.                                                                                                                                               |
| workflowDesigner.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.       |
| trustStore.fileName                                | Truststore filename. Truststore file should be saved in secrets folder available in Helm chart.                                                                                                        |
| trustStore.password                                | Truststore password for the Workflow Designer.                                                                                                                                                         |
| workflowDesignerCliParam.force                     | If the existing processes override is enabled. Specify true to enable overriding the existing processes or as false to disable overriding existing processes.                                          |
| persistentVolume.<br>storageClassReadWriteOnce     | Storage class for a PV, with ReadWriteOnce access mode, which can be accessed by a single node.                                                                                                        |
| persistentVolume.size                              | Size of the PV.                                                                                                                                                                                        |
| processPackagePersistentVolume.<br>sharedPVCName   | PV name shared by customer for importing the process package files.                                                                                                                                    |
| processPackagePersistentVolume.<br>sharedPVCFolder | Shared subpath in the PV for importing the process package files.                                                                                                                                      |

2. Deploy the Helm chart 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>
```

#### 5.9.4 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.

#### 5.9.5 Troubleshooting

There are no troubleshooting information for this release.

## 5.10 Deploying and configuring Content Connect on GCP

### 5.10.1 Prerequisites

1. Perform all the steps as described in the “[Prerequisites](#)” on page 307 section of “[Deploying and configuring Documentum Server on GCP](#)” on page 307.
2. Download the Docker image (Alpine 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 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        | 23.4.0 or 23.4 |
| dctm-content-connect-dbinit | 23.4.0 or 23.4 |

3. Download the Documentum Server Helm charts available in the documentum-<release-version>.tar file from My Support.

### 5.10.2 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 307.  
Make sure that you deploy the Content Connect Docker image (Alpine Linux only).
2. Open the single All-In-One documentum/values.yaml Helm chart file. In the contentconnect section, provide the appropriate values for the variables to pass them to your templates.

The “[contentconnect](#)” on page 246 table contains detailed information about the variables.

### 5.10.2.1 Deploying Content Connect on the client machine

**To deploy Content Connect on the client machine, perform the following steps:**

1. After the Admin Console URL is accessible, 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*.

### 5.10.3 Limitations

There are no limitations for this release.

### 5.10.4 Troubleshooting

There are no troubleshooting information for this release.

### 5.10.5 Upgrading

Upgrade the Content Connect pod using the `helm upgrade` command:

```
Helm upgrade <release name> . --values <resource file name> -n <namespace>
```

## 5.11 Deploying and configuring Extended ECM Documentum for Microsoft 365 on GCP

### 5.11.1 Prerequisites

1. Perform all the steps as described in the “[Prerequisites](#)” on page 307 section of “[Deploying and configuring Documentum Server on GCP](#)” on page 307.
2. Download the Docker image (Alpine 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 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-smartviewm365          | 23.4.0 or 23.4 |
| dctm-smartviewm365customjar | 23.4.0 or 23.4 |

3. Download the Documentum Server Helm charts available in the `documentum-<release-version>.tar` file from My Support.

## 5.11.2 Deploying Extended ECM Documentum for Microsoft 365

To deploy Extended ECM Documentum for Microsoft 365 on GCP, perform the following steps:

1. The information for deploying and configuring Extended ECM Documentum for Microsoft 365 on GCP is same as described in “[Deploying Extended ECM Documentum for Microsoft 365](#)” on page 252.

Make sure that you deploy the Extended ECM Documentum for Microsoft 365 Docker image (Alpine Linux only).

2. Open the single All-In-One `documentum/values.yaml` Helm chart file. In the `smartviewm365` section, provide the appropriate values for the variables to pass them to your templates.

The “[smartviewm365](#)” on page 252 table contains detailed information about the variables.

## 5.11.3 Limitations

There are no limitations for this release.

## 5.11.4 Troubleshooting

There are no troubleshooting information for this release.

## 5.12 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.

- 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 example, documentum-da-product:asia-south1:mypgdb).

- 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
```

- 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.

- Open the single All-In-One documentum/values.yaml Helm chart file and perform the following tasks:

- Provide all the details obtained from [step 5](#) in the database section.
- 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 `dctm-server.content-server`.

- Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/config/configuration.yaml --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where
Helm charts are extracted>/documentum-components.yaml --namespace <name of
namespace>
```

where `<config>` is `-extra-large-enhanced`, `-extra-large-standard`, `-large-enhanced`, `-large-standard`, `-medium-enhanced`, `-medium-large-enhanced`, `-medium-large-standard`, `-medium-standard`, `-small-enhanced`, `-small-medium-enhanced`, `-small-medium-standard`, `-small-standard`, or `-test-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 install dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/
config/configuration.yaml --values /opt/temp/documentum/platforms/gcp.yaml --values /
opt/temp/documentum/dockerimages-values.yaml --values /opt/temp/documentum/
documentum-resources-values-test-small.yaml --values /opt/temp/documentum/
documentum-components.yaml --namespace onedctms
```

- Verify the status of the deployment of Helm using the following command format:

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

- Verify the status of the deployment of the pod using the following command format:

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

## 5.13 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 documentum/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
```

### 5.13.1 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 **Elastifile 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-
db0a5969-1fcf-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
```

### 5.13.2 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

##### 6.1.1.1.1 Prerequisites for deployment in AWS EKS EC2 environment

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 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.
  - e. Create an EKS Cluster.  
AWS documentation contains detailed information.
  - f. Create a Kubernetes cluster namespace on the Cloud shell using the following command format:

```
kubectl create namespace <name of namespace>
```
  - g. 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 for all products except for Documentum REST Services, Documentum Foundation Services, and Documentum Content Management Interoperability Services) from OpenText Container Registry. Use Alpine Linux Docker images for Documentum REST Services, Documentum Foundation Services, and Documentum Content Management Interoperability Services. 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 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                  | 23.4.0 or 23.4 |
| dctm-admin                   | 23.4.0 or 23.4 |
| dctm-dfs                     | 23.4.0 or 23.4 |
| dctm-records                 | 23.4.0 or 23.4 |
| dctm-records-darinstallation | 23.4.0 or 23.4 |
| dctm-rqm                     | 23.4.0 or 23.4 |
| dctm-reports-installer       | 22.4.2         |
| dctm-reports-base            | 22.4.2         |
| dctm-rest                    | 23.4.0 or 23.4 |
| dctm-cmis                    | 23.4.0 or 23.4 |
| dctm-tomcat                  | 23.4.0 or 23.4 |



**Note:** The `dctm-tomcat` image (Alpine Linux) is required only for Documentum REST Services, Documentum Foundation Services, and Documentum Content Management Interoperability Services.

[“Decoupling Documentum product image from base Tomcat image” on page 420](#) contains detailed information.

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 77777777.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 77777777.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.

AWS documentation contains detailed information.

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, 22.4 or 23.2) 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, 22.4 or 23.2) while upgrading to 23.4.

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 `documentum/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. For information 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 `documentum/values.yaml` Helm chart file and perform the following tasks:

- i. Provide the appropriate value for the databaseHost and databasePort variables in common-variables as follows:

```
databaseHost: &database_host <endpoint information>
databasePort: &database_port 5432
```

- ii. Provide the appropriate value for the database.databaseServiceName variable in dcm-server.content-server as follows:

```
databaseServiceName: <name provided for database instance identifier>
```

- iii. Provide the appropriate values for the database.username and database.password variables in dcm-server.cs-secrets 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.



**Note:** If the database is SSL enabled, make sure that you update the following parameters in the Helm charts as follows:

- Set the value of database.sslEnabled to true in dcm-server.content-server.
- Provide the value for database.certificate in dcm-server.cs-secrets.

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-<release-version>.tar file from My Support.

#### 6.1.1.1.2 Prerequisites for deployment in AWS EKS Fargate environment

1. Set up the AWS EKS Fargate environment.

*AWS Documentation* contains detailed information.

2. Create a dedicated namespace in the AWS EKS Fargate environment. Make sure that any pod that is deployed in this environment are served by Fargate nodes or profiles.

*AWS Documentation* contains detailed information.

3. Perform all the relevant steps described in “[Prerequisites for deployment in AWS EKS EC2 environment](#)” on page 341.

4. Create the required PV and PVCs manually for the Documentum Server pod deployment as AWS cloud platform supports only AWS EFS CSI static provisioning for the AWS EKS Fargate environment.

Make sure that unique access points are created for each PV as follows:

- **User ID, Group ID, Owner user ID, and Owner group ID** set to 1000.
- Permission set to 777.
- Unique root directory path for each access point.

An example YAML file to create PV and PVC is as follows:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: fargatepv1
spec:
  capacity:
    storage: 1Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  storageClassName: <storage class name>
  csi:
    driver: efs.csi.aws.com
    volumeHandle: <file system ID>::<access point ID>
    persistentVolumeReclaimPolicy: Delete
  volumeMode: Filesystem
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  labels:
    app: dbr
  name: dbr-vct-dbr-0
  namespace: fargate
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: <storage class name>
  volumeMode: Filesystem
  volumeName: fargatepv1
  resources:
    requests:
      storage: 1Gi
```

5. To deploy Documentum Server in non-SSL mode with two replica each for the connection broker and Documentum Server pods, you must create the following PVCs as described in the table:

| PVC name         | PV name    | App    | Access mode   | Storage | PV reclaim policy |
|------------------|------------|--------|---------------|---------|-------------------|
| dbr-vct-dbr-0    | fargatepv1 | dbr    | ReadWriteOnce | 1Gi     | Delete            |
| dbr-vct-dbr-1    | fargatepv2 | dbr    | ReadWriteOnce | 1Gi     | Delete            |
| dcs-vct-dcs-pg-0 | fargatepv3 | dcs-pg | ReadWriteOnce | 1Gi     | Delete            |

| PVC name             | PV name    | App    | Access mode   | Storage | PV reclaim policy |
|----------------------|------------|--------|---------------|---------|-------------------|
| dcs-vct-dcs-pg-1     | fargatepv4 | dcs-pg | ReadWriteOnce | 1Gi     | Delete            |
| shared-logs-dbr-0    | fargatepv5 | dbr    | ReadWriteOnce | 2Gi     | Delete            |
| shared-logs-dbr-1    | fargatepv6 | dbr    | ReadWriteOnce | 2Gi     | Delete            |
| shared-logs-dcs-pg-0 | fargatepv7 | dcs-pg | ReadWriteOnce | 2Gi     | Delete            |
| shared-logs-dcs-pg-1 | fargatepv8 | dcs-pg | ReadWriteOnce | 2Gi     | Delete            |
| dcs-pg-pvc           | fargatepv9 | dcs-pg | ReadWriteMany | 1Gi     | Retain            |

 **Notes**

- You can modify the PVC storage size as per your requirement.
  - If you want to increase the replica count of Documentum Server pods after the successful deployment of pods, make sure that you create PV and PVCs appropriately as per your requirement before scaling up the deployment of Documentum Server pod.
6. After you create the PV and PVCs, make sure that the STATUS of all PVCs are Bound.
  7. Make sure that the values of resources.limits and resources.requests in dctm-server.content-server in <location where Helm charts are extracted>/<config>.yaml are set as per your sizing and Fargate environment requirement where <config> is -extra-large-enhanced, -extra-large-standard, -large-enhanced, -large-standard, -medium-enhanced, -medium-large-enhanced, -medium-large-standard, -medium-standard, -small-enhanced, -small-medium-enhanced, -small-medium-standard, -small-standard, or -test-small resource value YAML file that has been provided. These resource files contain pod sizing values like CPU, memory, and so on.
  8. Make sure that the values of resources.limits and resources.requests in dctm-server.content-server are set as per your sizing or environment requirement.

### 6.1.1.2 Deploying Documentum Server

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Update the image and tag fields in the single All-In-One documentum/values.yaml Helm chart file to point to the ECR images.
3. Open the single All-In-One documentum/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 56 contains detailed information about the variables.



**Note:** When you change the default value of any of the variables in dcm-server.cs-logging-configMap in the single All-In-One documentum/values.yaml Helm chart file each time after the initial deployment, you must run the Helm upgrade command.

4. Set the value of rwoStorage and rwmStorage to the storage class name created in step 13 using dynamic provisioning (for example, efs-sc) in “common-variables” on page 53.

For example:

```
rwoStorage: &rwoStorage efs-sc
rwmstorage: &rwmStorage efs-sc
```



**Note:** If you are deploying Documentum Server in the AWS EKS Fargate environment with EFS CSI static provisioning, make sure that you set the value of persistentVolume.createPVC to false in dcm-server.content-server because PV and PVCs are already created manually as described in step 4.

5. **Optional** To enable Documentum Server/connection broker external access, update the values for the following variables:
  - a. Set the value of externalAccessEnabled to true in common-variables.
  - b. Retain all the default values for ExtDocbroker in dcm-server.docbroker 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 dcm-server.content-server. Retain the default values for all other variables.
6. To enable ingress resources, make sure that you set the value of dcm-server.dcm-ingress.enabled to true in the single All-In-One documentum/values.yaml Helm chart file.
7. Set the value of dcm-server.dcm-ingress.ingress.configureHost to false in the single All-In-One documentum/values.yaml Helm chart file.



**Note:** When you set the value of dcm-server.dcm-ingress.ingress.configureHost to true, you must provide the appropriate values for dcm-

server.dctm-ingress.ingress.host and dctm-server.dctm-ingress.ingress.clusterDomainName.

8. Enable and update the service names for the required ingress resource in dctm-server.dctm-ingress in the single All-In-One documentum/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:

```
1 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"
  }
}
```

- Open the `aws.yaml` file in the extracted `platforms` folder and perform the following steps:

- 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
```

- Comment the following annotation which is provided for using ingress in the non-secure connection mode:

```
alb.ingress.kubernetes.io/listen-ports: '[{"HTTP":80}]'
```

- 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
```

- 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
```

- Perform all the steps from [step 8](#) to [step 12](#) in ["Deploying Documentum Server" on page 58](#) to complete the deployment and verification of the Documentum Server pod.

- Obtain the ingress deployment address using the following command format:

```
kubectl get ingress
```

Example output with address in bold:

| NAME                        | CLASS | HOSTS                                                                 | PORTS   |
|-----------------------------|-------|-----------------------------------------------------------------------|---------|
| ADDRESS                     |       |                                                                       |         |
| AGE                         |       |                                                                       |         |
| albingress-ingress <none> * |       | <b>k8s-dctmgroup-2edba3b1a3-591770489.us-west-2.elb.amazonaws.com</b> | 80 154m |

- After you obtain the address from the ingress deployment, update the value for `ingress.host` in `dctm-server.content-server` in the single All-In-One documentum/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> <location where Helm charts are extracted> --values  
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values  
<location where Helm charts are extracted>/dockerimages-values.yaml --values  
<location where Helm charts are extracted>/<config>.yaml --values <location where  
Helm charts are extracted>/documentum-components.yaml --namespace <name of  
namespace>
```

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

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 `dbrex-<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 dbrex 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 341 in “Deploying and configuring Documentum Server on AWS cloud platform” on page 341 except for those steps related to the PostgreSQL database.

#### 6.2.1.2 Deploying Documentum Administrator

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from [step 2 to step 9](#) in “Deploying Documentum Server” on page 58 in “Deploying and configuring Documentum Server on private cloud” on page 56.
3. Perform the tasks mentioned from [step 3 to step 6](#) in “Deploying Documentum Administrator” on page 111 in “Deploying and configuring Documentum Administrator on private cloud” on page 111.
4. Update the ingress resource rule for Documentum Administrator in the `dctm-server.dctm-ingress` section in the single All-In-One `documentum/values.yaml` Helm chart file.

For example:

```
daService:  
  enable: true  
  serviceName: da-svc  
  servicePort: 8080
```

5. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values  
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values  
<location where Helm charts are extracted>/dockerimages-values.yaml --values  
<location where Helm charts are extracted>/<config>.yaml --values <location where  
Helm charts are extracted>/documentum-components.yaml --namespace <name of  
namespace>
```

where <config> is -extra-large-enhanced, -extra-large-standard, -large-enhanced, -large-standard, -medium-enhanced, -medium-large-enhanced, -medium-large-standard, -medium-standard, -small-enhanced, -small-medium-enhanced, -small-medium-standard, -small-standard, or -test-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 install dcmdeployment /opt/temp/documentum --values /opt/temp/documentum/platforms/cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /opt/temp/documentum/documentum-components.yaml --namespace onedctmns
```

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

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

7. Verify the status of the deployment of the pod 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 349 in “Deploying and configuring Documentum Server on AWS cloud platform” on page 341.

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 D2 components on Amazon Web Services (AWS)

### 6.3.1 Overview

Amazon Web Services (AWS) provides on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered pay-as-you-go basis.

### 6.3.2 Prerequisites

Ensure that you complete the following activities before you deploy D2 on Amazon Web Services (AWS) environment:

1. Download and configure the Docker application from the Docker website.  
*Docker Documentation* contains detailed information.
2. Download and configure the Kubernetes application from the Kubernetes website.  
*Kubernetes Documentation* contains detailed information.
3. Download and configure the PostgreSQL database (server) from the PostgreSQL website.



**Note:** The PostgreSQL database client is packaged with Documentum Server Image.

*PostgreSQL Documentation* contains more information.

4. 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*.
5. Create an EKS cluster and a namespace within the cluster.
  - a. Install the AWS CLI on your Windows machine. *AWS Documentation* contains the instructions.
  - 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 more 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 1.12 kubectl client must work with Kubernetes 1.11, 1.12, and 1.13 clusters.

- g. Create an EKS Cluster mentioning the required cluster name, number of nodes, and your region using the following command format:

```
C:\ eksctl create cluster --name myekscluster --nodes 4 --region us-east-2
```

This command may take a 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
```

6. Upload the Docker images to Amazon Elastic Container Registry (ECR).

- a. Authenticate ECR from CLI using the following command format:

```
aws ecr get-login-password --region < your aws region> | docker login --username AWS --password-stdin aws_account_id.dkr.ecr.<aws_region>.amazonaws.com
```

Copy and run the output of this command in CLI. Successful execution of this command gives you authentication to ECR.

- b. Create a repository in ECR to load your 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 a 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 more information about the image tag mutability.
- vii. For scanning the image, choose the image scanning setting for the repository. Repositories configured to scan on load, start an image scan whenever an image is loaded. Otherwise, the image scans must be started manually.
- AWS Documentation contains more information about the scanning of image.
- viii. Click **Create repository**.
- c. Tag your Docker images using the following command format:
- ```
[aws_account_id.dkr.ecr.region.amazonaws.com]/[name of repository]/[image tag]
```
- d. Load the tagged image to ECR. For example:
- ```
docker push 77777777.dkr.ecr.region.amazonaws.com/dctmrepository:  
16.7.2000.0029
```
7. Install Helm and verify the installed version.
- a. Install Helm in your Windows cloud shell machine using the following command format:
- ```
C:\ choco install kubernetes-helm
```
- b. Verify the version of Helm using the following command format:
- ```
C:\ helm version
```
- Example output:
- ```
version.BuildInfo{Version:"v3.2.1",  
GitCommit:"fe51cdie31e6a202cba7dead9552a6d418ded79a", GitTreeState:"clean",  
GoVersion:"go1.13.10"}
```
8. 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. If you are using static provisioning of volumes, create two Amazon EPS file systems (one for Dctm-server and another for xPlore) 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.
  - iv. Under **Create mount targets**, for **Security groups**, add the security group that you created. Also add other security groups related to your cluster as shown below:

Availability zone	Subnet ID	IP address	Security groups
us-east-1b	subnet-036cf60827ae7aa6	192.168.112.252	<input type="button" value="Choose security groups ▾"/> Remove
			<div style="border: 1px solid #ccc; padding: 5px; display: inline-block;">           sg-0ea9b4132155e3a92            eks-cluster-sg-d2qclustereks4-1139805727         </div> <div style="border: 1px solid #ccc; padding: 5px; display: inline-block;">           sg-08276097498bec8d7            default         </div> <div style="border: 1px solid #ccc; padding: 5px; display: inline-block;">           sg-0194fd589c40974f8            eksctl-d2qclustereks4-cluster-            ControlPlaneSecurityGroup-K8TOK3E2M3KI         </div> <div style="border: 1px solid #ccc; padding: 5px; display: inline-block;">           sg-06b3e279e6340d5f1            EFS-access-for-D2-sg         </div> <div style="border: 1px solid #ccc; padding: 5px; display: inline-block;">           sg-05266365ee359130            eksctl-d2qclustereks4-cluster-            ClusterSharedNodeSecurityGroup-1X9X586IS0GFW         </div>

- v. Click **Next step**.
- vi. Choose a throughput mode for your file system.

 **Note:** By default, **Bursting** is the throughput mode and recommended for most file systems.

- vii. Choose a performance mode for your file system.



**Note:** By default, **General Purpose** is the performance mode and recommended for most file systems.

viii. Review your file system options and click **Create File System**.



**Note:** Create only one EFS file system if you are using dynamic provisioning.

9. Deploy the Amazon EFS Container Storage Interface (CSI) driver.  
*AWS Documentation* contains detailed information.
10. Create a storage class (for example, *efs-sc*) using CSI driver and test the sample PV and PVC bindings.  
*AWS Documentation* contains detailed information.
11. Delete both the sample PV and PVC resources. Ensure that you do not delete the storage class (for example, *efs-sc*).
12. If you are using static provisioning of volumes, create four EFS access points with default values for the following PVCs. Skip this point if you are using dynamic provisioning.  
*AWS Documentation* contains detailed information.
  - a. pvc for docbroker (used only in SSL mode ) in the EFS for Dctm-server.
  - b. pvc for the Documentum Server in the EFS for Dctm-server.
  - c. pvc for the DA in the EFS for Dctm-server.
  - d. pvc for xPlore in the EFS for xPlore.
13. If you are creating an EKS cluster of version 1.24 or upgrading an existing cluster to 1.24, you must install the Amazon EBS driver before upgrading the cluster or within the cluster if it is newly created. Please refer to the AWS documentation for more details on how to install the EBS driver.
14. If you want to use gp3 storage class for RWO volumes, follow the steps below:
  - a. Once the EKS cluster is created, execute the below command to fetch the existing storage classes present inside the cluster. If your cluster returns the below output, it already has the gp2 storage class. You can define the gp3 storage class using the yaml template shown in the next step.

```
> kubectl get storageclass
Output:
NAME      PROVISIONER          RECLAIMPOLICY  VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION  AGE
gp2 (default)  kubernetes.io/aws-eds  Delete      WaitForFirstConsumer
false           34m
```

- b. Create an AWS storage class manifest file for your storage class. The following `gp3-storage-class.yaml` example defines a storage class called `gp3` that uses the Amazon EBS `gp3` volume type and sets it as default storage class.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gp3
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp3
  fsType: ext4
```

- c. Use the `kubectl` command to create the storage class from the manifest file.

```
kubectl create -f gp3-storage-class.yaml
```

- d. Execute the below command to unset/remove `gp2` storageclass as the default sc.

```
kubectl annotate storageclass gp2 storageclass.kubernetes.io/is-default-class-
```

- e. Check if the `gp3` storageclass is successfully created and made as the default using the below command.

```
> kubectl get storageclass

Example Output:
gp2           kubernetes.io/aws-ebs   Delete
WaitForFirstConsumer   false          1d
gp3 (default)   kubernetes.io/aws-ebs   Delete
Immediate      false          10m
```

15. If you want to use EFS storage class for RWO volumes, update the `rwo_storage_class` anchor tag in the values yaml file.



**Note:** Currently, using EFS storage class for two volumes is supported only in dynamic provisioning.

16. **Optional** If you want to use Amazon PostgreSQL engine to create a database instance, perform the following tasks:

- a. Navigate to the AWS Relational Database Services (RDS) console and select **Databases**.
- b. Select **Create database**.
- c. Select a database creation method, for example, **Standard Create** or **Easy Create**.
- d. Select the **PostgreSQL** engine.
- e. Select the version of the engine from the **Version** list.
- f. Select a template according to your requirement, for example, **Production** or **Dev/Test** or **Free tier**.
- g. In **Settings**, provide the following tasks:
  - i. Enter a name for **DB instance identifier**.
  - ii. Enter the root user name for **Master username**.

- iii. Enter the password for **Master password** and confirm.
- h. In **DB instance size**, select a database instance for the **DB instance class** according to your requirement.
- i. In **Connectivity**, select the **Virtual Private Cloud (VPC)** where your cluster is available. For **subnet group**, select **public subnet group** and set **Public access to Yes**.
- j. Provide any additional configuration according to your requirement, for example, database options, encryption, maintenance, and so on.
- k. Click **Create database**.
- l. 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**.
- m. To use AWS PostgreSQL instance with Documentum D2 single Helm deployment, open `documentum/values.yaml` and perform the following tasks:
  - i. Provide the appropriate values for the variables in the content-server category as follows:

```
database:
  host: <Endpoint information>
  databaseServiceName: <name provided for database instance identifier>
  port: 5432
```

  - ii. Provide the appropriate values for the variables in the cs-secrets category as follows:

```
database:
  username: <name provided for master user name>
  password: <password provided for master password>
```

*AWS Documentation* contains detailed information to set up Amazon RDS, create Amazon Aurora or PostgreSQL database instance, and to connect to the database.



**Note:** For any other additional Dctm server-specific configuration for AWS, please refer to the DCTM server Cloud Deployment Guide.

17. Download the required D2 Images from the OpenText registry and push them to the Amazon container registry. Download the Helm Chart TAR file from OpenText MySupport.

### 6.3.3 Deploying D2 on AWS

1. Load the D2 Images to Amazon Elastic Container Registry and update the exact D2 Image names.
2. Upload the Docker images to Amazon Elastic Container Registry (ECR):
  - a. Tag your Docker images using the following command format:  
`[aws_account_id.dkr.ecr.region.amazonaws.com]/[name of repository]/[image tag]`

For example:  
`77777777.dkr.ecr.region.amazonaws.com/d2:16.7.2000.0029`
3. Extract the Helm Chart TAR file to a temporary location.
4. Open the documentum/values.yaml file and provide the appropriate values for the variables depending on your environment to pass them to your templates. For your reference, see the “[Deploying D2 Components on private cloud](#)” on page 121 section for the descriptions of the parameters.

For example:

- a. Update the image and tag fields in the values.yaml file of your Helm Chart to point to the ECR images.

```
images:
  Application:
    repository : 77777777.dkr.ecr.us-east-2.amazonaws.com
    name : d2
    tag : 16.7.2000.0029
```

- b. In the anchor tags section of documentum/values.yaml, set the value of rwo\_storage\_class to gp2 (or gp3 if configured as mentioned in Point 13 in Prerequisites) and the value of rwm\_storage\_class to the name created in Step 10, such as efs-sc.

For example:

```
rwoStorage: &rwo_storage_class gp2 or gp3
rwmstorage: &rwm_storage_class efs-sc
```

- c. Update the values for the following variables as follows (skip this step if you are using dynamic provisioning):

- In the docbroker category in documentum/values.yaml, provide the following information:
  - *existVolumePv*: provide a unique user-defined name for the PV (for example, dctmdbrpv)
  - *awsEFS*: Set the value to true
  - *awsEFSCSIDriver*: Retain the default value (efs.csi.aws.com)

- *awsEFSCSIHandle*: The format is EFS file system ID1::EFS access point ID1 (for example, fs-1fc8901b::fsap-0e45ed0c4555fcd34)



**Note:** The values provided in the docbroker category are used only when the Documentum Server is deployed in SSL mode.

- In the content-server category in `documentum/values.yaml`, provide the following information:
  - *existVolumePv*: Provide a unique user-defined name for PV (for example, dctmcspv)
  - *awsEFS*: Set the value to true
  - *awsEFSCSIDriver*: Retain the default value (efs,csi.aws.com)
  - *awsEFSCSIHandle*: The format is the EFS file system ID1::EFS access point ID2 (for example, fs-1fc8901b::fsap-0e45ed0c8888fcd34)
- In the da category in `documentum/values.yaml`, provide the following information:
  - *pvcName*: Provide a unique user-defined name for PVC (for example, dtcmdapcv)
  - *awsEFSCSIDriver*: Retain the default value (efs.csi.aws.com)
  - *awsEFSCSIHandle*: The format is EFS file system ID1::EFS access point ID2 (for example: fs-1fc8901b::fsap-0e45ed0c8888fcd34)
- In the persistentVolume category in `documentum/values.yaml`, provide the following information:
  - *awsEFSCSIPvName*: Provide a unique user-defined name for PV. For example: xplorepv
  - *awsEFS*: Set the value to true
  - *awsEFSCSIDriver*: Retain the default value (efs.csi.aws.com)
  - *awsEFSCSIHandle*: The format is EFS file system ID2::EFS access point ID2. For example: fs-1fc8901b::fsap-0e45ed0c8888fcd34.



**Note:** Use the first file system for docbroker/content-server and another file system for xPlore, i.e., Dctm-server and xPlore should use different EFS file systems.

5. Deploy the AWS Application Load Balancer (ALB) ingress controller Helm. Perform the following steps:
  - a. Perform all the steps provided in *AWS Documentation* to create an OpenID Connect (OIDC) provider and IAM role for the ALB ingress controller
  - b. Install the ALB ingress controller using Helm 3.x. Perform the following tasks:
    - i. Install the `TargetGroupBinding` custom resource definitions.

- ii. Add the eks-charts repository.
- iii. Install the ALB ingress controller that corresponds to the AWS region for your cluster using the following command format:

```
helm upgrade -i aws-load-balancer-controller eks/aws-load-
balancercontroller --set clusterName=<cluster-name> --set
serviceAccount.create=false --set serviceAccount.name=aws-load-balancer-
controller -n kube-system
```

*AWS Documentation* contains detailed information.

6. 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
```

7. To enable ingress resources, ensure that you set the value of enabled to true in the *dctm-ingress* category in *documentum/values.yaml*.
8. Set the value of *configureHost* in the *dctm-ingress* category in *documentum/values.yaml* to false.



**Note:** When you set the value of *configureHost* to true, you must provide the appropriate values for host and *clusterDomainName*.

9. Enable and update the service names for the required ingress resource in the *dctm-ingress* category in *documentum/values.yaml*.



**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.

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
```

10. 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 a comma.

For example:

```
1 alb.ingress.kubernetes.io/subnets: subnet-0c4a1017a4ffb7962,
subnet-0bbd3ec1319a30772, subnet-0a6ab032a5e03be49
```

11. **Optional** If you want to use ingress in secure connection mode, perform the following steps:

- 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.
- 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:

- In the alb.ingress.kubernetes.io/subnets annotation, provide the clusters' public subnet IDs separated by a comma.

For example:

```
alb.ingress.kubernetes.io/subnets: subnet-0c4a1017a4ffb7962,
subnet-0bbd3ec1319a30772, subnet-0a6ab032a5e03be49
```

- Comment the following annotation which is provided for using ingress in the non-secure connection mode:

```
alb.ingress.kubernetes.io/listen-ports: '[{"HTTP":80}]'
```

- 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/asbselfsigned
```

- Update the value of ARN obtained from Step 11.b in the uncommented annotation.

For example:

```
alb.ingress.kubernetes.io/certificate-arn: arn:aws:iam::<AWS account
ID>:server-certificate/testcertificate
```

12. If necessary, update the AWS ingress annotation values in the documentum/platforms/aws.yaml file according to your platform's ingress.
13. In appworks-gateway section of configuration.yml, add an **additional \* to ingress path** as follows. Set **useSSL to true** and copy the SSL certificates for AWG under the directory ce-d2\charts\appworks-gateway\trustCertificates. For the database **host**, provide the AWS PostgreSQL instance **endpoint** obtained from the AWS RDS console.

```

database:
  vendor: PostgreSQL
  server:
    host: <Endpoint information>
    port: 5432

appworksdb:
  user: gateway201user
  password: password
  database: gateway201db

useSSL: true

service:
  type: ClusterIP
  port: 80

ingress:
  hosts:
    - host: <ingress-host>
      paths:
        - path: /*
          pathType: ImplementationSpecific

```



**Note:** Update appworksdb user as default admin user (postgres) and password respectively. This will create db and awg user automatically in Amazon RDS.

If you don't want to use default admin user, create a new user manually and grant admin privileges using below commands.

```
CREATE ROLE <new_user> WITH LOGIN PASSWORD '<new_password>' CREATEDB;
GRANT <new_user> to postgres;
```

Setting useSSL to true and copying the certificates is not needed if you are not using SSL mode.

14. ExtDocbroker should be enabled for accessing content-server outside of the AWS cluster. Make sure externalAccessEnabled is set to true in the anchor tags section of documentum/values.yaml. Leave the default values in ExtDocbroker section of documentum/values.yaml as it is.

Set useELB to true if you want to use external load balancer instead of node port. If you want to provide any annotations for the ingress set useLBAnnotations to true and provide annotations under LBAnnotations.

```

ExtDocbroker:
  extNativeNodePort:
  extSSLNodePort:
  createExtService: true
  useELB: true
  useLBAnnotations: true
  LBAnnotations:

```

```
service.beta.kubernetes.io/aws-load-balancer-internal: "true"
service.beta.kubernetes.io/aws-load-balancer-scheme: internal
```

15. ExtCS should be enabled for accessing content-server outside of the AWS cluster. In the ExtCS section of documentum/values.yaml , mention nativeExtPort as 80 and sslExtPort as 81 and keep the other default values as it is.

Set useELB to true if you want to use external load balancer instead of node port. If you want to provide any annotations for the ingress set useLBAnotations to true and provide annotations under LBAnnotations.

```
ExtCS:
  tcp_route: 10.0.0.0
  nativeExtPort: 80
  sslExtPort: 81
  extDbrPort: 1491
  useELB: true
  useLBAnotations: true
  LBAnnotations:
    service.beta.kubernetes.io/aws-load-balancer-internal: "true"
    service.beta.kubernetes.io/aws-load-balancer-scheme: internal
```

16. For D2 AWS deployment:

- a. In D2 Config, click **Tools>Options**, set the **Redirect URL** as follows: https://<d2-ingress-host>/D2/. Click the **Save** icon and click **Tools/Reload D2 Options** and then **Tools>Refresh Cache**.
- b. If AWS RDS PostGreSQL instance is used, update the otdsdb URL in documentum/values.yaml as follows:  
url: jdbc:postgresql://<Database Endpoint>:5432/<otdsdb>
- c. Due to known issue XPORE-19947, disable xPlore in D2 single Helm deployment by setting xplore.enabled to false in documentum/values.yaml and deploy xPlore separately post D2 deployment.

17. For all other values, please refer to “Prerequisites” on page 117 in “Deploying and configuring D2 components on private cloud” on page 117.

18. Deploy the D2 Helm in your AWS EKS environment using the following command:

```
helm install <release name> <location where Helm ChartTAR files are extracted>/d2 --values platforms/aws.yaml --values d2-resources-values-<config>.yaml --namespace <name of namespace>
```

For example:

```
helm install d2 /opt/temp/Helm-charts/d2 --values platforms/aws.yaml --values d2-resources-values-<config>.yaml --namespace docu
```

19. Verify the status of the deployment of D2 Helm using the following command format:

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

20. Verify the status of the deployment of D2 pods using the following command format:

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

21. Obtain the ingress deployment address using the following command format:

```
kubectl get ingress
```

Example output with address:

NAME	CLASS	HOSTS	ADDRESS
cc-ingress	<none>	*	k8s-d2newccgroup-e43deaba18-870087646.us-east-1.elb.amazonaws.com
appworks-gateway-ingress	<none>	*	k8s-d2newawggroup-7c13480ced-2039655174.us-east-1.elb.amazonaws.com
dctm-ingress	<none>	*	k8s-d2newdctmgroup-440a21863a-1248970307.us-east-1.elb.amazonaws.com

22. After you obtain the addresses from the ingress deployment, update the dctm-ingress, appworks-gateway ingress and host values wherever it is present in documentum/values.yaml with the value of corresponding ADDRESS from Step 21. Make sure you have updated the following anchor tags in documentum/values.yaml accordingly. For example:

- otdsAuthSvc: &otds\_auth\_svc k8s-d2newdctmgroup-440a21863a-1248970307.us-east-1.elb.amazonaws.com/otdsws
- ingressDomain: &ingress\_domain us-east-1.elb.amazonaws.com

23. Obtain the External IP of csext LoadBalancer service using the following command format:

```
kubectl get svc -n <namespace>
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP
csext-dcs-pg	LoadBalancer	10.100.197.39	
a4652670631244318837d475c5b7e8c5-30432636.us-east-1.elb.amazonaws.com			80:30650/TCP, 81:32322/TCP
dbrextdbr	LoadBalancer	10.100.137.236	
aaf2a87c480164c9cb3dd2f3167a23ff-835178087.us-east-1.elb.amazonaws.com			80:32478/TCP, 81:30552/TCP

24. Update that tcp\_route parameter under ExtCS subsection in the content-server category in documentum/values.yaml with the value of the External IP of csext LoadBalancer service.

```
ExtCS:
  tcp_route: a4652670631244318837d475c5b7e8c5-30432636.us-east-1.elb.amazonaws.com
  nativeExtPort: 80
  sslExtPort: 81
  extDbrPort: 1491
```

25. In previous deployment if node port was used for externalizing docbroker/content server, now if you want to use load balancer (ELB) URL set useELB to true in ce-d2/config/configuration.yaml for CS and docbroker and upgrade the deployment. Follow the steps 23, 24 and 25 above to upgrade the deployment with the load balancer URL.

26. Run the Helm upgrade command using the following command format:

```
helm upgrade <d2deployment_name> . --values .\config\configuration.yaml --values .\platforms\aws.yaml --values .\dockerimages-values.yaml --values <resources_yaml_file>.yaml -n <namespace>
```

Now, run `kubectl get ingress` command again and you should see the host value is updated with the corresponding address as follows:

```
$ kubectl get ingress
```

Example output with address:

NAME	HOSTS	ADDRESS
cc-ingress	east-1.elb.amazonaws.com	k8s-d2newccgroup-e43deaba18-870087646.us-
	east-1.elb.amazonaws.com	k8s-d2newccgroup-e43deaba18-870087646.us-
appworks-gateway-ingress	east-1.elb.amazonaws.com	k8s-d2newawggroup-7c13480ced-2039655174.us-
	east-1.elb.amazonaws.com	k8s-d2newawggroup-7c13480ced-2039655174.us-
dctm-ingress	east-1.elb.amazonaws.com	k8s-d2newdctmggroup-440a21863a-1248970307.us-
	east-1.elb.amazonaws.com	k8s-d2newdctmggroup-440a21863a-1248970307.us-



**Note:** If you are planning to use user-friendly names for the ingress host, then you can map the user-friendly host names provided in the Helm charts to the ingress address in the DNS server or the hosts file of your client machine as applicable and access the applications using the user-friendly host names.

In that case, steps 21 and 22 are not needed.

27. To access any Documentum ingress resources in a browser, use the following URL format:

```
http://<ingress-protocol
```

For example, if you are trying to access D2 Config in a browser, use the following URL format:

```
https://k8s-dctmggroup-2edba3b1a3-591770489.us-west-2.elb.amazonaws.com/D2-Config
```



### Notes

- If the application ingress URL is not accessible after the Helm upgrade, restart the respective pods and try again.
- For D2 AWS deployment, add an extra / at the end for the D2 Classic URL. For example: <https://k8s-dctmggroup-2edba3b1a3-591770489.uswest-2.elb.amazonaws.com/D2/>.

### 6.3.4 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 AWS, if you specify the storageclass as gp2 in the Persistent volume Claim (PVC), then AWS automatically creates a Persistent Volume of requested size using Amazon Elastic Block 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 Persistent Volume Claims.

## 6.4 Deploying and configuring Documentum Records Client on AWS cloud platform

### 6.4.1 Kubernetes platform

#### 6.4.1.1 Prerequisites

- Perform all the steps as described in “Prerequisites” on page 341 in “Deploying and configuring Documentum Server on AWS cloud platform” on page 341 except for those steps related to the PostgreSQL database.

#### 6.4.1.2 Deploying Documentum Records Client

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from step 2 to step 9 in “Deploying Documentum Server” on page 58 in “Deploying and configuring Documentum Server on private cloud” on page 56.
3. Perform the task mentioned in step 2 in “Deploying Documentum Records Client” on page 198 in “Deploying and configuring Documentum Records Client on private cloud” on page 194.
4. Update the ingress resource rule for Documentum Records Client in the `dctm-server.dctm-ingress` section.

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>/documentum -  
f <location where Helm charts are extracted>/documentum/platforms/<cloud  
platform>.yaml --namespace <name of namespace>
```

For example:

```
helm install documentum /opt/temp/Helm-charts/documentum -f /opt/temp/Helm-charts/  
documentum/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 349 in “Deploying and configuring Documentum Server on AWS cloud platform” on page 341.
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>
```

#### 6.4.1.3 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.

#### 6.4.1.4 Troubleshooting

There are no troubleshooting information for this release.

### 6.5 Deploying and configuring Documentum Foundation Services on AWS cloud platform

#### 6.5.1 Kubernetes platform

##### 6.5.1.1 Prerequisites

Perform all the steps as described in “[Prerequisites](#)” on page 341 in “Deploying and configuring Documentum Server on AWS cloud platform” on page 341 except for those steps related to the PostgreSQL database.

### 6.5.1.2 Deploying Documentum Foundation Services

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from [step 2](#) to [step 9](#) in “Deploying Documentum Server” on page 58 in “Deploying and configuring Documentum Server on private cloud” on page 56.
3. Perform the tasks mentioned from [step 3](#) to [step 6](#) in “Deploying Documentum Foundation Services” on page 208 in “Deploying and configuring Documentum Foundation Services on private cloud” on page 208.
4. Update the ingress resource rule for Documentum Foundation Services in the `dctm-server.dctm-ingress` section in the single All-In-One documentum/`values.yaml` Helm chart file.

For example:

```
dfsService:
  enable: true
  serviceName: dfs-server
  servicePort: 8080
```

5. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where
Helm charts are extracted>/documentum-components.yaml --namespace <name of
namespace>
```

where `<config>` is `-extra-large-enhanced`, `-extra-large-standard`, `-large-enhanced`, `-large-standard`, `-medium-enhanced`, `-medium-large-enhanced`, `-medium-large-standard`, `-medium-standard`, `-small-enhanced`, `-small-medium-enhanced`, `-small-medium-standard`, `-small-standard`, or `-test-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 install dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/
platforms/cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --
values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /
opt/temp/documentum/documentum-components.yaml --namespace onedctmns
```

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

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

7. Verify the status of the deployment of the pod 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 349 in “Deploying and configuring Documentum Server on AWS cloud platform” on page 341.

9. Access the Documentum Foundation Services application using the ALB ingress controller service in a browser using the following URL format:

```
http://<ingress deployment ADDRESS>/<Documentum ingress resource>
```

### 6.5.1.3 Limitations

There are no limitations for this release.

### 6.5.1.4 Troubleshooting

There are no troubleshooting information for this release.

## 6.6 Deploying and configuring Documentum REST Services on AWS cloud platform

### 6.6.1 Kubernetes platform

#### 6.6.1.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 341 in “Deploying and configuring Documentum Server on AWS cloud platform” on page 341 except for those steps related to the PostgreSQL database.

#### 6.6.1.2 Deploying Documentum REST Services

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from **step 2** to **step 9** in “Deploying Documentum Server” on page 58 in “Deploying and configuring Documentum Server on private cloud” on page 56.
3. Perform the tasks mentioned from **step 4** to **step 7** in “Deploying Documentum REST Services” on page 217 in “Deploying and configuring Documentum REST Services on private cloud” on page 217.
4. Update the ingress resource rule for Documentum REST Services in the `dctm-server.dctm-ingress` section in the single All-In-One `documentum/values.yaml` Helm chart file.

For example:

```
restService:  
  enable: true  
  serviceName: dctm-rest  
  servicePort: 8080
```

5. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values  
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values  
<location where Helm charts are extracted>/dockerimages-values.yaml --values  
<location where Helm charts are extracted>/<config>.yaml --values <location where
```

```
Helm charts are extracted>/documentum-components.yaml --namespace <name of namespace>
```

where <config> is -extra-large-enhanced, -extra-large-standard, -large-enhanced, -large-standard, -medium-enhanced, -medium-large-enhanced, -medium-large-standard, -medium-standard, -small-enhanced, -small-medium-enhanced, -small-medium-standard, -small-standard, or -test-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 install dcmdeployment /opt/temp/documentum --values /opt/temp/documentum/platforms/cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /opt/temp/documentum/documentum-components.yaml --namespace onedctmns
```

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

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

7. Verify the status of the deployment of the pod 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 349 in “[Deploying and configuring Documentum Server on AWS cloud platform](#)” on page 341.
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>
```

### 6.6.1.3 Limitations

There are no limitations for this release.

### 6.6.1.4 Troubleshooting

There are no troubleshooting information for this release.

## 6.7 Deploying and configuring Documentum Content Management Interoperability Services on AWS cloud platform

### 6.7.1 Kubernetes platform

#### 6.7.1.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 341 in “Deploying and configuring Documentum Server on AWS cloud platform” on page 341.

#### 6.7.1.2 Deploying Documentum Content Management Interoperability Services

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from step 2 to step 9 in “Deploying Documentum Server” on page 58 in “Deploying and configuring Documentum Server on private cloud” on page 56.
3. Perform the task mentioned in step 6 in “Deploying Documentum Content Management Interoperability Services” on page 225 in “Deploying and configuring Documentum Content Management Interoperability Services on private cloud” on page 225.
4. Update the ingress resource rule for Documentum Content Management Interoperability Services in the `dctm-server.dctm-ingress` section in the single All-In-One `documentum/values.yaml` Helm chart file.

For example:

```
cmisService:  
  enable: true  
  serviceName: dctm-cmis  
  servicePort: 8080
```

5. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values  
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values  
<location where Helm charts are extracted>/dockerimages-values.yaml --values  
<location where Helm charts are extracted>/<config>.yaml --values <location where  
Helm charts are extracted>/documentum-components.yaml --namespace <name of  
namespace>
```

where `<config>` is `-extra-large-enhanced`, `-extra-large-standard`, `-large-enhanced`, `-large-standard`, `-medium-enhanced`, `-medium-large-enhanced`, `-medium-large-standard`, `-medium-standard`, `-small-enhanced`, `-small-medium-enhanced`, `-small-medium-standard`, `-small-standard`, or `-test-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 install dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/platforms/cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --values /opt/temp/documentum/documentum-resources-values-test-small.yaml --values /opt/temp/documentum/documentum-components.yaml --namespace onedctmns
```

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

```
helm status <release name>
```
7. Verify the status of the deployment of the pod 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 349 in “Deploying and configuring Documentum Server on AWS cloud platform” on page 341.
9. Access the Documentum Content Management Interoperability Services application using the ALB ingress controller service in a browser using the following URL format:  

```
http://<ingress deployment ADDRESS>/<Documentum ingress resource>
```

#### 6.7.1.3 Limitations

There are no limitations for this release.

#### 6.7.1.4 Troubleshooting

There are no troubleshooting information for this release.

### 6.8 Deploying and configuring Documentum Reports on AWS cloud platform

#### 6.8.1 Kubernetes platform

##### 6.8.1.1 Prerequisites

Perform all the steps as described in “Prerequisites” on page 341 in “Deploying and configuring Documentum Server on AWS cloud platform” on page 341.

### 6.8.1.2 Deploying Documentum Reports

1. Extract the Helm charts downloaded from My Support to a temporary location.
2. Perform the tasks mentioned from **step 2** to **step 9** in “Deploying Documentum Server” on page 58 in “Deploying and configuring Documentum Server on private cloud” on page 56.
3. Perform the task mentioned in **step 2** in “dtrbase” on page 230 in “Deploying Documentum Reports (dtrinstaller and dtrbase)” on page 230 in “Deploying and configuring Documentum Reports on private cloud” on page 230.
4. Update the ingress resource rule for Documentum Reports in the `dctm-ingress` category.

For example:

```
dtrbaseService:
  enable: false
  serviceName: <service name of Documentum Report Base>
  servicePortCore: 5001
  servicePortServlet: 8080
```

5. Deploy the Documentum Chart with Documentum Reports Helm charts using the following command format:

```
helm install <release name> <location where Helm charts are extracted>/documentum -f <location where Helm charts are extracted>/documentum/platforms/<cloud platform>.yaml <location where Helm charts are extracted>/documentum/init-containers/dtr-init.yaml --namespace <name of namespace>
```

For example:

```
helm install dctmdeployment /opt/temp/Helm-charts/documentum -f /opt/temp/Helm-charts/documentum/platforms/aws.yaml /opt/temp/Helm-charts/documentum/init-containers/dtr-init.yaml --namespace docu
```

6. Verify the status of the deployment of Documentum Chart with Documentum Reports Helm charts using the following command format:

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

7. Verify the status of the deployment of the Documentum Chart with Documentum Reports 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 349 in “Deploying and configuring Documentum Server on AWS cloud platform” on page 341.

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>
```

### 6.8.1.3 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.

### 6.8.1.4 Troubleshooting

There are no troubleshooting information for this release.

## 6.9 Deploying and configuring Content Connect on AWS cloud platform

### 6.9.1 Prerequisites

1. Perform all the steps as described in the “Prerequisites” on page 341 section of “Deploying and configuring Documentum Server on AWS cloud platform” on page 341.
2. Download the Docker image (Alpine 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 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	23.4.0 or 23.4
dctm-content-connect-dbinit	23.4.0 or 23.4

3. Download the Documentum Helm chart in the `documentum-<release-version>.tar` file from the *My Support > Documentum Server > Software Download* page.

## 6.9.2 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:

```
77777777.dkr.ecr.region.amazonaws.com/dctm-content-connect:23.4
```

```
77777777.dkr.ecr.region.amazonaws.com/dctm-content-connect-dbinit:23.4
```

- b. Upload the tagged Docker image to Amazon ECR.

For example:

```
docker push 77777777.dkr.ecr.region.amazonaws.com/dctm-content-connect:23.4
```

```
docker push 77777777.dkr.ecr.region.amazonaws.com/dctm-content-connect-dbinit:23.4
```

2. Open the single All-In-One documentum/values.yaml Helm chart file and in the contentconnect section, provide the appropriate values for the variables to pass them to your templates.

The “contentconnect” on page 246 table contains detailed information about the variables.

## 6.9.3 Limitations

There are no limitations for this release.

## 6.9.4 Troubleshooting

Symptom	Cause	Fix
When using Content Connect, Cross-Origin Resource Sharing (CORS) related errors occur even after all configurations are set.	CORS related errors occur.	<ul style="list-style-type: none"> <li>• Increase the Load balancer response timeout.</li> <li>• Add the Content Connect Admin Console URL to the rest.cors.allowed.origins tag in the rest-api-runtime.properties file.</li> </ul>

## 6.10 Deploying and configuring Extended ECM Documentum for Microsoft 365 on AWS cloud platform

### 6.10.1 Prerequisites

1. Perform all the steps as described in the “Prerequisites” on page 341 section of “Deploying and configuring Documentum Server on AWS cloud platform” on page 341.
2. Download the Docker image (Alpine 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 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-smartviewm365	23.4.0 or 23.4
dctm-smartviewm365customjar	23.4.0 or 23.4

3. Download the Documentum Helm chart in the documentum-<release-version>.tar file from My Support.

### 6.10.2 Deploying Extended ECM Documentum for Microsoft 365

1. The information for deploying and configuring Extended ECM Documentum for Microsoft 365 on AWS is same as described “Deploying Extended ECM Documentum for Microsoft 365” on page 252. Make sure that you deploy the Extended ECM Documentum for Microsoft 365 Docker image (Alpine Linux only).
2. Open the single All-In-One documentum/values.yaml Helm chart file and in the smartviewm365 section, provide the appropriate values for the variables to pass them to your templates.

The “smartviewm365” on page 252 table contains detailed information about the variables.

### **6.10.3 Limitations**

There are no limitations for this release.

### **6.10.4 Troubleshooting**

There is no troubleshooting information for this release.



## 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 connection broker, Documentum Server, and Java Method 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 `documentum/values.yaml` Helm chart file, provide the appropriate values for the variables to pass them to your templates as described in the following table:

Name	Description
<code>dctm-server.cs-secrets.newrelic.license_key</code>	License key information of New Relic. This variable uses the value that you specified for <code>licenseKey</code> in <code>common-variables</code> in “ <a href="#">Defining common variables</a> ” on page 53.
<code>dctm-server.docbroker.newrelic.enable</code>	Indicates if New Relic is enabled for the connection broker. The default value is <code>false</code> . This variable uses the value that you specified for <code>newrelic</code> in <code>common-variables</code> in “ <a href="#">Defining common variables</a> ” on page 53.  If you set the value to <code>true</code> , then the values for the remaining variables are used to identify the process in the dashboard.

Name	Description
dctm-server.docbroker.newrelic.proxy_host	IP address of the proxy server. This variable uses the value that you specified for newrelicProxyHost in common-variables in “Defining common variables” on page 53.
dctm-server.docbroker.newrelic.proxy_port	Available port reserved for the New Relic server. This variable uses the value you specified for newrelicProxyPort in common-variables in “Defining common variables” on page 53.
dctm-server.docbroker.newrelic.proxy_protocol	Protocol to connect to the pod. This variable uses the value you specified for newrelicProxyProtocol in common-variables in “Defining common variables” on page 53.
dctm-server.docbroker.newrelic.c_app_name	Unique application name of the connection broker.
dctm-server.content-server.newrelic.enable	Indicates if New Relic is enabled for Documentum Server. The default value is false. This variable uses the value that you specified for newrelic in common-variables in “Defining common variables” on page 53.  If you set the value to true, then the values for the remaining variables are used to identify the process in the dashboard.
dctm-server.content-server.newrelic.app_name	Unique application name of the Java Method Server.
dctm-server.content-server.newrelic.proxy_host	IP address of the proxy server. This variable uses the value that you specified for newrelicProxyHost in common-variables in “Defining common variables” on page 53.
dctm-server.content-server.newrelic.proxy_port	Available port reserved for the New Relic server. This variable uses the value you specified for newrelicProxyPort in common-variables in “Defining common variables” on page 53.
dctm-server.content-server.newrelic.proxy_protocol	Protocol to connect to the pod. This variable uses the value you specified for newrelicProxyProtocol in common-variables in “Defining common variables” on page 53.
dctm-server.content-server.newrelic.c_app_name	Unique application name of Documentum Server.

## 7.1.2 Integrating New Relic with Documentum Administrator

In the single All-In-One documentum/values.yaml Helm chart file, provide the appropriate values for the variables to pass them to your templates as described in the following table:

Name	Description
dctm-server.da.newrelic.enable	Indicates if New Relic is enabled. The default value is <code>false</code> . This variable uses the value you specified for <code>newrelicEnable</code> in <a href="#">common-variables</a> in <a href="#">“Defining common variables” on page 53</a> .  If you set the value to <code>true</code> , then the New Relic APM agent is configured and starts during the startup of the container.
dctm-server.da.newrelic.appName	Application name (descriptive) of Documentum Administrator. It is mandatory to provide the application name if New Relic is enabled.
dctm-server.da.newrelic.proxyHost	IP address of the proxy server. This variable uses the value that you specified for <code>newrelicProxyHost</code> in <a href="#">common-variables</a> in <a href="#">“Defining common variables” on page 53</a> .
dctm-server.da.newrelic.proxyPort	Available port reserved for the New Relic server. This variable uses the value you specified for <code>newrelicProxyPort</code> in <a href="#">common-variables</a> in <a href="#">“Defining common variables” on page 53</a> .
dctm-server.da.newrelic.proxyScheme	Indicates to allow the agent to connect through proxy using the HTTP or HTTPS protocol. This variable uses the value you specified for <code>newrelicProxyProtocol</code> in <a href="#">common-variables</a> in <a href="#">“Defining common variables” on page 53</a> .

## 7.1.3 Integrating New Relic with Documentum Records Client

In the records section in the single All-In-One documentum/values.yaml Helm chart file, provide the appropriate values for the variables to pass them to your templates as described in the following table:

Name	Description
newrelic.enable	Indicates if New Relic is enabled. The default value is <code>false</code> .  If you set the value to <code>true</code> , then the values for the remaining variables are used to identify the process in the dashboard.  This variable uses the value that you specified for <code>newrelic</code> in the <code>common-variables</code> section in <a href="#">“Defining common variables” on page 53</a> .
newrelic.appName	Application name of the Records Client.
newrelic.licenseKey	License key corresponding to the name of the New Relic license key. This variable uses the value that you specified for <code>licenseKey</code> in the <code>common-variables</code> section in <a href="#">“Defining common variables” on page 53</a> .
newrelic.proxyHost	IP address of the proxy server. This variable uses the value that you specified for <code>newrelicProxyHost</code> in the <code>common-variables</code> section in <a href="#">“Defining common variables” on page 53</a> .
newrelic.proxyPort	Port reserved for the New Relic server. This variable uses the value that you specified for <code>newrelicProxyPort</code> in the <code>common-variables</code> section in <a href="#">“Defining common variables” on page 53</a> .
newrelic.proxyScheme	Proxy scheme such as HTTP. This allows the agent to connect through proxies. This variable uses the value that you specified for <code>newrelicProxyProtocol</code> in the <code>common-variables</code> section in <a href="#">“Defining common variables” on page 53</a> .

#### 7.1.4 Integrating New Relic with Documentum Foundation Services

In the single All-In-One `documentum/values.yaml` Helm chart file, provide the appropriate values for the variables to pass them to your templates as described in the following table:

Name	Description
dctm-server.dfs.newrelic.enable	Indicates if New Relic is enabled. The default value is <code>false</code> . This variable uses the value you specified for <code>newrelicEnable</code> in <a href="#">common-variables</a> in “Defining common variables” on page 53.  If you set the value to <code>true</code> , then the New Relic APM agent is configured and starts during the startup of the container.
dctm-server.dfs.newrelic.dfs_application_name	Application name of Documentum Foundation Services.
dctm-server.dfs.newrelic.proxyHost	IP address of the proxy server. This variable uses the value that you specified for <code>newrelicProxyHost</code> in <a href="#">common-variables</a> in “Defining common variables” on page 53.
dctm-server.dfs.newrelic.proxyPort	Available port reserved for the New Relic server. This variable use the value you specified for <code>newrelicProxyPort</code> in <a href="#">common-variables</a> in “Defining common variables” on page 53.

## 7.1.5 Integrating New Relic with Documentum REST Services

In the single All-In-One `documentum/values.yaml` Helm chart file, provide the appropriate values for the variables to pass them to your templates as described in the following table:

Name	Description
dctm-server.dctm-rest.newrelic.enabled	Indicates if New Relic is enabled. The default value is <code>false</code> . This variable uses the value you specified for <code>newrelicEnable</code> in <a href="#">common-variables</a> in “Defining common variables” on page 53.  If you set the value to <code>true</code> , then the New Relic APM agent is configured and starts during the startup of the container.
dctm-server.dctm-rest.newrelic.configurationFile	Name of the configuration file.
dctm-server.dctm-rest.newrelic.addNodeNamePrefix	Indicates to add the node name as a prefix for the application name that is displayed in the New Relic dashboard.

Name	Description
dctm-server.dctm-rest.newrelic.proxy_host	IP address of the proxy server. This variable uses the value that you specified for newrelicProxyHost in common-variables in “Defining common variables” on page 53.
dctm-server.dctm-rest.newrelic.proxy_port	Available port reserved for the New Relic server. This variable use the value you specified for newrelicProxyPort in common-variables in “Defining common variables” on page 53.
dctm-server.dctm-rest.newrelic.proxy_protocol	Protocol to connect to the pod. This variable use the value you specified for newrelicProxyPort in common-variables in “Defining common variables” on page 53.
dctm-server.dctm-rest.newrelic.app_name	Application name of Documentum REST Services.

### 7.1.6 Integrating New Relic with Documentum Content Management Interoperability Services

In the single All-In-One documentum/values.yaml Helm chart file, provide the appropriate values for the variables to pass them to your templates as described in the following table:

Name	Description
dctm-server.dctm-cmis.newrelic.enabled	Indicates if New Relic is enabled. The default value is false. This variable uses the value you specified for newrelicEnable in common-variables in “Defining common variables” on page 53.  If you set the value to true, then the New Relic APM agent is configured and starts during the startup of the container.
dctm-server.dctm-cmis.newrelic.configurationFile	Name of the configuration file.
dctm-server.dctm-cmis.newrelic.addNodeNamePrefix	Indicates to add the node name as a prefix for the application name that is displayed in the New Relic dashboard.
dctm-server.dctm-cmis.newrelic.proxy_host	IP address of the proxy server. This variable uses the value that you specified for newrelicProxyHost in common-variables in “Defining common variables” on page 53.

Name	Description
dctm-server.dctm-cmis.newrelic.proxy_port	Available port reserved for the New Relic server. This variable use the value you specified for <code>newrelicProxyPort</code> in <code>common-variables</code> in “Defining common variables” on page 53.
dctm-server.dctm-cmis.newrelic.proxy_protocol	Protocol to connect to the pod. This variable use the value you specified for <code>newrelicProxyPort</code> in <code>common-variables</code> in “Defining common variables” on page 53.
dctm-server.dctm-cmis.newrelic.app_name	Application name of Documentum Content Management Interoperability Services.

### 7.1.7 Integrating New Relic with Documentum Reports

In the `dtrbase` category in the single All-In-One `documentum/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	<code>enable</code>	Indicates if New Relic is enabled. The default value is <code>false</code> . Specify the value provided for <code>newrelicEnable</code> in the <code>common-variables</code> category in “Defining common variables” on page 53.  If you set the value to <code>true</code> , then the New Relic APM agent is configured and started during the startup of the container.
	<code>appName</code>	Application name (descriptive) of Documentum Reports. This is mandatory if New Relic is enabled.
	<code>proxyHost</code>	IP address of the proxy server, if any. Specify the value provided for <code>newrelicProxyHost</code> in the <code>common-variables</code> category in “Defining common variables” on page 53.
	<code>proxyPort</code>	Available port reserved for the New Relic server, if any. Specify the value provided for <code>newrelicProxyPort</code> in the <code>common-variables</code> category in “Defining common variables” on page 53.
	<code>proxyScheme</code>	Indicates to allow the agent to connect through proxy using the HTTP or HTTPS protocol. Specify the value provided for <code>newrelic_proxy_protocol</code> in the <code>common-variables</code> category in “Defining common variables” on page 53.

## 7.1.8 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 documentum/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:

Name	Description
newrelic.new_relic_enabled	Indicates if New Relic is enabled for Extended ECM Documentum for Microsoft 365. Specify the value provided for newrelic in the common-variables section in <a href="#">“Defining common variables” on page 53</a> .  When set to true, provide the values for new_relic_app_name.
newrelic.new_relic_app_name	A unique application name. Content Connect constructs the application name as NODE_NAME, appNameSuffix in newrelic.yaml.
newrelic.new_relic_license_key	New Relic license key. Specify the value provided for licenseKey in the common-variables section in <a href="#">“Defining common variables” on page 53</a> .
newrelic.new_relic_proxy_protocol	Protocol to connect to the pod. Specify the value provided for newrelic_proxy_protocol in the common-variables section in <a href="#">“Defining common variables” on page 53</a> .
newrelic.new_relic_proxy_host	IP address of the proxy server. Specify the value provided for newrelicProxyHost in the common-variables section in <a href="#">“Defining common variables” on page 53</a> .
newrelic.new_relic_proxy_port	Port reserved for the New Relic server. Specify the value provided for newrelicProxyPort in the common-variables section in <a href="#">“Defining common variables” on page 53</a> .

### 7.1.8.1 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.

### 7.1.9 Integrating New Relic with Extended ECM Documentum for Microsoft 365

The New Relic APM has been integrated with Extended ECM Documentum for Microsoft 365 for application monitoring. For more information about New Relic, visit the New Relic website.

Open the single All-In-One documentum/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:

Name	Description
newrelic.enable	Indicates if New Relic is enabled for Extended ECM Documentum for Microsoft 365. Specify the value provided for newrelic in the common-variables section in <a href="#">“Defining common variables” on page 53</a> .
newrelic.app_name	Application name for Extended ECM Documentum for Microsoft 365.
newrelic.proxy_host	IP address of the proxy server. Specify the value provided for newrelicProxyHost in the common-variables section in <a href="#">“Defining common variables” on page 53</a> .
newrelic.proxy_port	Port reserved for the New Relic server. Specify the value provided for newrelicProxyPort in the common-variables section in <a href="#">“Defining common variables” on page 53</a> .

## 7.2 Integrating Graylog

Graylog Collector Sidecar (legacy) and Graylog Sidecar are lightweight configuration management systems to collect the logs.

### 7.2.1 Integrating Graylog with Documentum Server

#### 7.2.1.1 Prerequisites

Deploy the Graylog server in your cloud platform and record the service name and port number in which the services are listening to.

### 7.2.1.2 Deploying Graylog Collector Sidecar and Graylog Sidecar using Documentum Helm charts

#### 7.2.1.2.1 Deploying Graylog Collector Sidecar (legacy)

1. Open the single All-In-One documentum/values.yaml Helm chart file and provide the values for graylogServer and graylogPort in the common-variables section with the Graylog headless service name and the port information. Provide values for all other variables in the single All-In-One documentum/values.yaml Helm chart file.
2. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values  
<location where Helm charts are extracted>/config/configuration.yml --values  
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values  
<location where Helm charts are extracted>/dockerimages-values.yaml --values  
<location where Helm charts are extracted>/<config>.yaml --values <location where  
Helm charts are extracted>/documentum-components.yaml --namespace <name of  
namespace>
```

where <config> is -extra-large-enhanced, -extra-large-standard, -large-enhanced, -large-standard, -medium-enhanced, -medium-large-enhanced, -medium-large-standard, -medium-standard, -small-enhanced, -small-medium-enhanced, -small-medium-standard, -small-standard, or -test-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 install dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/  
config/configuration.yml --values /opt/temp/documentum/platforms/aws.yaml --values /  
opt/temp/documentum/dockerimages-values.yaml --values /opt/temp/documentum/  
documentum-resources-values-test-small.yaml --values /opt/temp/documentum/  
documentum-components.yaml --namespace onedctmns
```

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

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

4. Verify the status of the deployment of the 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.

#### 7.2.1.2.2 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 documentum/values.yaml Helm chart file and perform the following tasks:
  - a. Provide the appropriate values for the variables in the dcm-server.cs-secrets section to pass them to your templates as described in [step 8.a](#) in “Deploying Documentum Server” on page 58. Make sure that you provide the Graylog token value created in step 1 in “Deploying Graylog Sidecar” on page 394.
  - b. Provide the values for graylogServer and graylogPort in common-variables with the Graylog headless service name and the port information. Provide values for all other variables in the single All-In-One documentum/values.yaml Helm chart file.
3. Deploy the Helm using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values  
<location where Helm charts are extracted>/config/configuration.yaml --values  
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values  
<location where Helm charts are extracted>/dockerimages-values.yaml --values  
<location where Helm charts are extracted>/<config>.yaml --values <location where  
Helm charts are extracted>/documentum-components.yaml --namespace <name of  
namespace>
```

where **<config>** is -extra-large-enhanced, -extra-large-standard, -large-enhanced, -large-standard, -medium-enhanced, -medium-large-enhanced, -medium-large-standard, -medium-standard, -small-enhanced, -small-medium-enhanced, -small-medium-standard, -small-standard, or -test-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 install dcmdeployment /opt/temp/documentum --values /opt/temp/documentum/  
config/configuration.yaml --values /opt/temp/documentum/platforms/aws.yaml --values /  
opt/temp/documentum/dockerimages-values.yaml --values /opt/temp/documentum/  
documentum-resources-values-test-small.yaml --values /opt/temp/documentum/  
documentum-components.yaml --namespace onedctmns
```

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

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

5. Verify the status of the deployment of the 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}
fields.gl2_source_collector: {sidecar.nodeId}
.
filebeat.inputs:
- input_type: log
paths:
- /pod-data/dba/*.log
type: log
output.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).

## 7.2.2 Integrating Graylog with Documentum REST Services

Documentum REST Services 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-rest-product/graylog-sidecar
  imagePullPolicy: Always
  server: rest-graylog-headless.dctm-rest.svc.cluster.local
  port: 9000
  serviceToken: 87ckh5e9aammi6rd6g75ceuibce4ot8icb3itpeq4bibe25ge0
  logsDir: /home/dmadmin/logs
```

By default, the value of `graylog.logsDir` in `documentum/charts/dctm-server/charts/dctm-rest/values.yaml` 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.g12_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

```

### 7.2.3 Integrating Graylog with Content Management Interoperability Services

Documentum Content Management Interoperability Services 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 Content Management Interoperability Services pod.

Example for Graylog integration:

```

graylog:
enabled:true
image: gcr.io/documentum-cmis-product/graylog-sidecar
imagePullPolicy: Always
server: rest-graylog-headless.dctm-rest.svc.cluster.local
port: 9000
serviceToken: 87ckh5e9ammi6rd6g75ceuibce4ot8icb3itpeq4bibe25ge0
logsDir: /home/dmadmin/logs

```

By default, the value of `graylog.logsDir` in `documentum/charts/dctm-server/charts/dctm-cmis/values.yaml` is `/home/dmadmin/logs`. Make sure that the value of `graylog.logsDir` aligns with the REST log path specified in `log4j2.properties` in `dctm-cmis/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.g12_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

```

## 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
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 `docbroker.liveness.initialDelaySeconds` in `documentum/charts/dctm-server/charts/docbroker/values.yaml` 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 `docbroker.liveness.enable` to `false` in the single All-In-One `documentum/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 `contentserver.liveness.initialDelaySeconds` in `documentum/charts/dctm-server/charts/content-server/values.yaml` 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 three 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 `dctm-server.content-server.contentserver.liveness.enable` to `false` in the single All-In-One documentum/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 `dctm-server.content-server.contentserver.liveness.considerJMSForLiveness` to `false` in the single All-In-One documentum/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 section in the single All-In-One documentum/values.yaml Helm chart file, provide the appropriate values for the variables as described in the following table:

Name	Description
containers.da.probing.livenessProbe.initialDelaySeconds	Number of seconds after the pod has started before the probe is initiated.
containers.da.probing.livenessProbe.periodSeconds	Frequency to perform the probe.

### 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 dctm-server.dfs section in the single All-In-One documentum/values.yaml Helm chart file, provide the appropriate values for the variables as described in the following table:

Name	Description
containers.dfs.probing.livenessProbe.initialDelaySeconds	Number of seconds after the pod has started before the probe is initiated.
containers.da.probing.livenessProbe.periodSeconds	Frequency to perform the probe.

### 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 dctm-server.records section in the single All-In-One documentum/values.yaml Helm chart file, provide the appropriate values for the variables as described in the following table:

Name	Description
containers.da.probing.livenessProbe.initialDelaySeconds	Number of seconds after the Documentum Records Client pod has started before the probe is initiated. The default value is 180 seconds.

Name	Description
containers.da.probing.livenessProbe.periodSeconds	Frequency to perform the probe. The default value is 300 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-server.dctm-rest` section in the single All-In-One `documentum/values.yaml` Helm chart file, provide the appropriate values for the variables as described in the following table:

Name	Description
<code>livenessProbe.enabled</code>	Indicates if the liveness probe is enabled. The default value is true.
<code>livenessProbe.scheme</code>	Scheme (HTTP/HTTPS) for the probe URL. The default value is HTTP.
<code>livenessProbe.initialDelaySeconds</code>	Time in seconds after the container or pod has started before performing the first probe. The default value is 200 seconds.
<code>livenessProbe.periodSeconds</code>	Frequency to perform the probe. The default value is 25 seconds.

The port number in which the requests are sent to, is the same as defined in `httpPort` or `httpsPort` in the `dctm-server.dctm-rest` section in the single All-In-One `documentum/values.yaml` Helm chart file depending on scheme utilized.

### 7.3.8 Checking liveness of Documentum Content Management Interoperability Services

The liveness probe of Documentum Content Management Interoperability Services sends the HTTP/HTTPS requests to the Content Management Interoperability Services pod to check if the Documentum Content Management Interoperability Services container or pod is running or stuck. By default, the liveness probe is enabled.

In the `dctm-server.dctm-cmis` section in the single All-In-One `documentum/values.yaml` Helm chart file, provide the appropriate values for the variables as described in the following table:

Name	Description
<code>livenessProbe.enabled</code>	Indicates if the liveness probe is enabled. The default value is true.

Name	Description
livenessProbe.scheme	Scheme (HTTP/HTTPS) for the probe URL. The default value is HTTP.
livenessProbe.initialDelaySeconds	Time in seconds after the container or pod has started before performing the first probe. The default value is 40 seconds.
livenessProbe.periodSeconds	Frequency to perform the probe. The default value is 5 seconds.

The port number in which the requests are sent to, is the same as defined in `httpPort` or `httpsPort` in the `dctm-server.dctm-cmis` section in the single All-In-One documentum/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. *OpenText Documentum Server Administration and Configuration Guide* contains detailed information.

Open the single All-In-One documentum/values.yaml Helm chart file and provide the appropriate values for the variables in `dctm-server.content-server` to pass them to your templates as described in “[content-server](#)” on page 72.

### 7.4.2 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**.

### 7.4.3 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- (`$soIdentity`) and repository-identity (`RepositoryIdentity`) in Documentum Foundation Services. *OpenText Documentum Foundation Services Development Guide* contains detailed information.

### 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 `documentum/values.yaml` Helm chart file and in common variables in “[Defining common variables](#)” [on page 53](#), make sure that the value of `otdsEnabled` is set to true. Provide the appropriate values for the mandatory variables such as `dctm-rest.otds.url` and `dctm-rest.otds.clientID` to pass them to your templates as described in [step 7](#) in “[Deploying Documentum REST Services](#)” [on page 217](#).

### 7.4.5 Integrating OTDS with Documentum Content Management Interoperability Services

Documentum Content Management Interoperability Services is integrated with OTDS. For more information, see the *Documentum Content Management Interoperability Services* chapter in *OpenText Documentum Platform and Platform Extensions Installation Guide*.

## 7.5 Integrating Event Hub

Documentum supports the Event Hub feature using the Fluentd and Apache Kafka services.

You must configure the Apache Kafka services externally. You must configure the Salted Challenge Response Authentication Mechanism (SCRAM) plain authentication mechanism in your Apache Kafka deployment. In addition, you must create a user and a topic for producing the messages which is pushed to Kafka.

Fluentd is an open source data collection tool using the Apache 2.0 license. 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 Integrating Event Hub with Documentum Server

Documentum Server supports the Event Hub feature.

To use the feature, you must set the value of fluentd to true in common variables in “Defining common variables” on page 53 in the single All-In-One documentum/values.yaml Helm chart file. Make sure that you update the values of other fluentd and kafka related variables in common variables in “Defining common variables” on page 53 in the single All-In-One documentum/values.yaml Helm chart file.

For example, fluentdTcpPort, fluentdImage, and so on.

Fluentd pushes the events that are generated by Documentum Server to the external 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 eventLogLevel in the common-variables section in “Defining common variables” on page 53. 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.

#### 7.5.1.1 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.

Make sure that you provide the appropriate values for the Fluentd and Kafka variables in the single All-In-One documentum/values.yaml Helm chart file. Some of the important variables are dctm-server.cs-logging-configMap.fluentdConf.enable, dctm-server.cs-logging-configMap.fluentdConf.TCPPort, dctm-server.cs-logging-configMap.fluentdConf.RESTPort, dctm-server.cs-logging-configMap.fluentdConf.UDPPort, dctm-server.cs-logging-configMap.fluentdConf.compressionMode, dctm-server.cs-logging-configMap.fluentdConf.bufferingMode, dctm-server.cs-logging-configMap.fluentdConf.kafkaTopic, dctm-server.cs-logging-configMap.fluentdConf.kafkaUser, dctm-server.cs-logging-configMap.fluentdConf.kafkaUsrPasswd, and so on.

[“Deploying Documentum Server” on page 58](#) contains detailed information about the variables.

## 7.5.2 Retrieving messages

You can retrieve the messages in the following ways:

- Using the Kafka built-in utility, log in to the external Kafka deployment, then navigate 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 external Kafka deployment 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.

## 7.5.3 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 dctm-server.cs-logging-configMap.fluentdConf.enable in the single All-In-One documentum/values.yaml Helm chart file.

Fluentd pushes the events that are generated by Documentum Foundation Services to the external 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 `documentum/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.

### 7.5.3.1 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.

Make sure that you provide the appropriate values for the Fluentd and Kafka variables in the single All-In-One `documentum/values.yaml` Helm chart file. Some of the important variables are `dctm-server.cs-logging-configMap.fluentdConf.enable`, `dctm-server.cs-logging-configMap.fluentdConf.TCPPort`, `dctm-server.cs-logging-configMap.fluentdConf.RESTPort`, `dctm-server.cs-logging-configMap.fluentdConf.UDPPort`, `dctm-server.cs-logging-configMap.fluentdConf.compressionMode`, `dctm-server.cs-logging-configMap.fluentdConf.bufferingMode`, `dctm-server.cs-logging-configMap.fluentdConf.kafkaTopic`, `dctm-server.cs-logging-configMap.fluentdConf.kafkaUser`, `dctm-server.cs-logging-configMap.fluentdConf.kafkaUsrPasswd`, and so on.

[“Deploying Documentum Foundation Services” on page 208](#) contains detailed information about the variables.

### 7.5.3.2 Documentum Foundation Services specific event names

Service name or Category	Event name
REPOSITORY INQUIRY	GetRepoListOperation
	ListContentReposAction
	SetContentRepoList
	GetServers
	CheckMinServerVersionCompatibility
	GetRepoNameByObjIdOperation
	RepoNameById
QUERY	ExecuteOperation

<b>Service name or Category</b>	<b>Event name</b>
	QueryAction
	GetActionResult
	LogQuery
	SetQueryResult
QUERY STORE	ListSavedQueriesOperation
	ListSavedQueriesAction
	ListSavedSearches
	LoadSavedQueryOperation
	LoadSavedQueryAction
	LoadSavedQuery
SCHEMA	GetRepoInfoOperation
	GetSchemaInfoOperation
	GetTypeInfosOperation
	GetTypeInfoOperation
	GetPropertyInfoOperation
	GetDynamicAssistValuesOperation
	GetValueAssistSnapshotOperation
OBJECT CREATE	CreateOperation
	CreateAction
	CreateObject
	Validate
	ValidateContent
	CreateContentfulObject
	CreateContentlessObject
	UpdateLightweighObject
	Checkout
	SaveObject
OBJECT CREATE PATH	CreatePathOperation
OBEJCT SERVICE GET	GetOperation
	DeepGetAction
	ContentLoaderObjectCreate
	PropertiesLoaderObejctCreate
	PermissionLoaderObjectCreate

<b>Service name or Category</b>	<b>Event name</b>
	Walk
	AddCreatedObject
	GatherRelatedDataObjects
	LoadRestOfRelations
OBJECT MOVE	MoveOperation
	MoveAction
	MoveProcess
OBJECT UPDATE	UpdateOperation
	UpdateAction
	UpdateObject
OBJECT COPY	CopyOperation
	CopyAction
	CopyProcess
OBJECT SAVE AS NEW	SaveAsNewOperation
	SaveAsNewAction
OBJECT 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

<b>Service name or Category</b>	<b>Event name</b>
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
	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

<b>Service name or Category</b>	<b>Event name</b>
	GetClustersOperation
	GetClusterAction
	GetSubclustersOperation
	GetResultsPropsOperation
	GetFacetsOperation
	GetFacetsAction
COMMENT	CreateCommentOperation
	EnumCommentOperation
	GetCommentOperation
	MarkReadOperation
	MarkUnreadOperation
	UpdateCommentOperation
TASK MANAGEMENT	ClaimOperation
	WorkQueueTaskAcquire
	WorkflowTaskAcquire
	ReleaseOperation
	SuspendOperation
	SuspendUntilOperation
	WorkQueueTaskSuspend
	ResumeOperation
	WorkflowResume
	CompleteOperation
	WorkflowTaskComplete
	SetNextActivities
	SetRerunMethod
	RemoveOperation
	SetPriorityOperation
	AddAttachmentOperation
	GetAttachmentInfosOperation
	GetAttachmentsOperation
	DeleteAttachmentsOperation
	RemoveWorkflowAttachment
	AddCommentOperation

<b>Service name or Category</b>	<b>Event name</b>
	AddComment
	GetCommentsOperation
	ForwardOperation
	DelegateOperation
	SetDelegateUser
	GetTaskInfoOperation
	GetTaskDescriptionOperation
	SetOutputOperation
	GetInputOperation
	GetOutputOperation
	NominateOperation
	GetMyTaskAbstractsOperation
	GetMyTasksOperation
	QueryOperation
WORKFLOW	StartProcessOperation
	StartedWorkflowService
	SetPackageInfo
	SetAttachmentInfo
	SetAliasInfo
	SetPerformerInfo
	GetProcessInfoOperation
	RetrievePackageInfo
	RetrieveAliasInfo
	RetrievePerformerInfo
	GetProcessTempsOperation
	GetProcessDQL
	TerminateWorkflowOperation
	WorkflowHalt
	WorkflowAbort
	WorkflowDestroy
	CompleteWorkItemOperation
	WorkItemAcquire
	GetForwardActivities

<b>Service name or Category</b>	<b>Event name</b>
	SetOutputActivities
	WorkItemComplete
LIFECYCLE	AttachOperation
	AttachAction
	ExecuteLifecycleAction
	GetLifecycleOperation
	GetLifecycleAction
	DetachOperation
	DetachAction
VIRTUAL DOCUMENT	UpdateDocumentOperation
	RetrieveSnapshotOperation
	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

Service name or Category	Event name
	AnalyzeOperation
	Events Related to License Service Call
	RequestLicenseOperation
	CheckinOperation
VERSION CONTROL	CheckinAction
	CheckoutOperation
	CheckoutAction
	CancelCheckoutOperation
	CancelCheckoutAction
	GetCheckOutInfoOperation
	GetCheckoutInfoAction
	GetVersionInfoOperation
	GetVersionInfoAction
	DeleteVersionOperation
	DeleteAllVersionsOperation
	GetCurrentOperation
	ACLCREATEOperation
ACCESS CONTROL	CreateACLACTION
	ACLUpdateOperation
	UpdateACLACTION
	ACLGetOperation
	GetACLACTION
	ACLDelteOperation
	DeleteACLACTION

#### 7.5.4 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 `dctm-rest.fluentd_service.enable` and `dctm-rest.fluentdConf.enable` to `true` in the single All-In-One documentum/values.yaml Helm chart file.

Fluentd pushes the events that are generated by Documentum REST Services to the external 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.

#### 7.5.4.1 Fluentd and Kafka with Documentum REST Services

Fluentd is packaged with Documentum REST Services. When you deploy the Documentum REST Services pod, Fluentd is deployed as a sidecar container.

Make sure that you provide the appropriate values for the Fluentd and Kafka variables in the single All-In-One documentum/values.yaml Helm chart file. Some of the important variables are dctm-server.cs-logging-configMap.fluentdConf.enable, dctm-server.cs-logging-configMap.fluentdConf.TCPPort, dctm-server.cs-logging-configMap.fluentdConf.RESTPort, dctm-server.cs-logging-configMap.fluentdConf.UDPPort, dctm-server.cs-logging-configMap.fluentdConf.compressionMode, dctm-server.cs-logging-configMap.fluentdConf.bufferingMode, dctm-server.cs-logging-configMap.fluentdConf.kafkaTopic, dctm-server.cs-logging-configMap.fluentdConf.kafkaUser, dctm-server.cs-logging-configMap.fluentdConf.kafkaUsrPasswd, and so on.

[“Deploying Documentum REST Services” on page 217](#) contains detailed information about the variables.

#### 7.5.4.2 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

#### 7.5.4.3 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 default, 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`.

#### 7.5.4.4 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.

#### 7.5.4.5 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 <code>rest-api.runtime.properties</code>	<code>AUTHENTICATION_TYPE</code> in Event Hub events
<code>basic/basic-ct</code>	PASSWORD
<code>otds_password/ct-otds_password</code>	OTDS_PASSWORD
<code>otds_token/ct-otds_token/otds_token-basic/ct-otds_token-basic</code>	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.

#### 7.5.4.6 Publishing 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 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.

### 7.6.1 Scaling Documentum Server

1. To increase the number of replica pods to be spawned for the Documentum Server image, update `content-server.contentserver.replicaCount` in `<location where Helm charts are extracted>/<config>.yaml` to a required value where `<config>` is `-extra-large-enhanced`, `-extra-large-standard`, `-large-enhanced`, `-large-standard`, `-medium-enhanced`, `-medium-large-enhanced`, `-medium-large-standard`, `-medium-standard`, `-small-enhanced`, `-small-medium-enhanced`, `-small-medium-standard`, `-small-standard`, or `-test-small` resource value YAML file that has been provided. These resource files contain pod sizing values like CPU, memory, and so on.
2. After increasing the value of `replicaCount`, you must run the Helm upgrade command using the following command format:

```
helm upgrade <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where
Helm charts are extracted>/documentum-components.yaml --namespace <name of
namespace>
```

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

## 7.6.2 Descaling Documentum Server

1. To decrease the number of replica pods to be spawned for the Documentum Server image, update `content-server.contentserver.replicaCount` in `<location where Helm charts are extracted>/<config>.yaml` to a required value where where `<config>` is `-extra-large-enhanced`, `-extra-large-standard`, `-large-enhanced`, `-large-standard`, `-medium-enhanced`, `-medium-large-enhanced`, `-medium-large-standard`, `-medium-standard`, `-small-enhanced`, `-small-medium-enhanced`, `-small-medium-standard`, `-small-standard`, or `-test-small` resource value YAML file that has been provided. These resource files contain pod sizing values like CPU, memory, and so on.
2. After decreasing the value of `replicaCount`, you must the Helm upgrade using the following command format:

```
helm upgrade <release name> <location where Helm charts are extracted> --values  
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values  
<location where Helm charts are extracted>/dockerimages-values.yaml --values  
<location where Helm charts are extracted>/<config>.yaml --values <location where  
Helm charts are extracted>/documentum-components.yaml --namespace <name of  
namespace>
```

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



**Note:** By default, the resources are not removed after the descaling process. OpenText does not recommend you to remove the resources.

## 7.7 Supporting multiple repositories in one namespace

In a cloud platform, one statefulset of Documentum Server node serves only one repository. To deploy multiple repositories in one namespace, you must deploy new statefulset.

The new statefulset (new repository) can use the connection broker of the first repository or any of the existing connection broker or new connection broker. When you are deploying a new repository, it can either use the current repository as the global repository or another repository as the global repository. If you want the new repository to use another repository as the global repository, then you must configure the values for all the variables in `dctm-server.content-server.globalRepository` in the single All-In-One `documentum/values.yaml` Helm chart file.



**Note:** Make sure that the global repository and the new repository are projected to the same connection broker.

The new repository can either use the database of the previous repository or a separate database. If you want the new repository to use the database of the previous repository, then you must provide unique values for `dctm-server.content-server.docbase.name`, `dctm-server.content-server.docbase.id`, `dctm-server.content-server.docbase.owner`, and `dctm-server.content-server.docbase.index` in the single All-In-One documentum/values.yaml Helm chart file.

To share AEK across multiple repositories, make sure you set the value of `dctm-server.content-server.contentserver.aek.shareKey` and `dctm-server.content-server.contentserver.aek.persistentVolume.shareKeyPVCName` in the single All-In-One documentum/values.yaml Helm chart file.

## 7.8 Rotating AEK in Documentum Server

From the 22.2 release, you can rotate AEK within the same Documentum Server version.

**!** **Important**

- Do not change the value of any other variables in the single All-In-One documentum/values.yaml Helm chart file other than the variables mentioned in this section.
  - If the value of `dctm-server.content-server.contentserver.aek.name` is same as the existing AEK name, then the existing AEK is reused.
  - Upgrading of AEK is not supported in certificate-based communication.
  - AEK rotation is supported only within the same Documentum Server version.
1. Open the single All-In-One documentum/values.yaml Helm chart file and modify the values of the variables depending on the following rotation requirement:  
From local to local AEK:
    - (Optional) Value for `dctm-server.cs-secrets.contentserver.aek.algorithm`. See “[cs-secrets](#)” on page 60.
    - Value for `dctm-server.cs-secrets.contentserver.aek.oldPassphrase`. See “[cs-secrets](#)” on page 60.
    - (Optional) Value for `dctm-server.cs-secrets.contentserver.aek.passphrase`. Make sure that you follow the password complexity rules. See “[cs-secrets](#)” on page 60.
    - Value for `dctm-server.content-server.contentserver.aek.name`. See “[content-server](#)” on page 72.
  2. Run the Helm upgrade command using the following command format:

```
helm upgrade <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
```

```
<location where Helm charts are extracted>/<config>.yaml --values <location where  
Helm charts are extracted>/documentum-components.yaml --namespace <name of  
namespace>
```

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

## 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 section and change the values according to 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 for the Documentum REST Services and Documentum Foundation Services products. From the 23.4 release, decoupling of product Docker image from the base Tomcat image is supported for the Documentum Content Management Interoperability Services.

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 (for Documentum REST Services and Documentum Foundation Services) and until version 23.2 (for Documentum Content Management Interoperability Services). 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-server.dctm-rest` or `dctm-server.dfs` or `dctm-server.dctm-cmis` section in the single All-In-One `documentum/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 53](#) (common variables), “[Deploying and configuring Documentum REST Services on private cloud](#)” on page 217 (for Documentum REST Services), “[Deploying and configuring Documentum Foundation Services on private cloud](#)” on page 208 (for Documentum Foundation Services), and “[Deploying and configuring Documentum Content Management Interoperability Services on private cloud](#)” on page 225 (for Documentum Content Management Interoperability Services).

## 7.11 Using logrotate in Documentum Server

The logrotate utility 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`.

## 7.12 Using logcleanup in Documentum Server

The logcleanup utility is used to remove all the files from a specific folder in the Documentum Server pod. By default, the session logs generated in the `/opt/dctm/dba/log/<hexa_id of docbase id>/<installowner>` location are cleaned up. The cleanup interval for session logs (`logcleanup.sessionlogcleanupinterval`) is set to seven days. You can modify the cleanup interval value as per your requirement.

The log files generated in the following locations can also be cleaned up:

- `/opt/dctm/tomcat<version>/logs`
- `/opt/dctm/product/<version>/install/logs`
- `/opt/dctm/product/<version>/thumbsrv/container/logs`
- `/opt/dctm/logs`
- `/opt/dctm_docker/logs`

To cleanup the preceding list of folders, the folder and the age of files to be removed must be mentioned in `extraEnv`.

For example, you can specify name and value for log cleanup in `extraEnv` as follows where 1 is the first folder and 2 is the age in minutes when the file is deleted:

```
- name: LOG_CLEANUP_FOLDER_1
value: "/opt/dctm/logs"
- name: LOG_CLEANUP_FOLDER_PERIOD_1
value: "2"
```

The cleanup interval (`logcleanup.interval`) for the additional folders is set to 24 hours. You can modify the cleanup interval value as per your requirement.



**Note:** The location provided for the logrotate and logcleanup utilities must be different.

## 7.13 Editing configmap details for D2 webapps

Configmap objects are available for each D2 webapp configuration file. More details regarding the configuration file can be found in the *Documentum D2 and Documentum D2 Mobile Administration Guide*.

1. Run the following command to see a list of configmap objects:

```
kubectl get cm
```

2. Modify the configuration of a particular configmap using the following command.

```
kubectl get cm config_map_name -o jsonpath=".data.config_file_name" > config_file_name
```

3. Edit the config\_file\_name. Note that the file must be encoded in UTF-8.

```
kubectl create cm config_map_name --from-file config_file_name -o yaml --dry-run=client | kubectl apply -f -
```

For example:

```
kubectl get cm d2classic-logback-xml -o jsonpath=".data.logback\\.xml" > logback.xml  
notepad logback.xml
```

Use **File > Save As** in Notepad, and select **UTF-8** in the **Encoding** field in the Save As dialog box when saving changes to the file.

```
kubectl create cm d2classic-logback-xml --from-file logback.xml -o yaml --dry-run=client | kubectl apply -f -
```

4. You can also edit the config map directly using `kubectl edit cm <configmap-name>` command.
5. After you update the configmap, wait several minutes, then check the configuration file inside the webapp container. The file content should be automatically updated by Kubernetes. Currently D2 webapps will automatically pick up the changes from `dfc.properties` and `logback.xml` after a few minutes. For other configuration files, a restart of D2 webapp might be required to pick up the changes.

Below is the list of the config maps available in a typical D2 CE deployment for the different components enabled and their purpose:

Config map name	Component	Purpose
appworks-gateway-config-map	Appworks Gateway	Configuration of the different parameters for Appworks Gateway container
awg-init-container-config-map	Appworks Gateway	Configuration of the different parameters for Appworks Gateway init container

<b>Config map name</b>	<b>Component</b>	<b>Purpose</b>
awg-pg-init-container-configmap	Appworks Gateway	Configuration of the different parameters for Appworks Gateway Postgres init container
acs-logging-configmap	Dctm server	Configuration of the properties file log4j.properties for acs.log, AcsServer.log & acs_trace.log
bpm-logging-configmap	Dctm server	Configuration of the properties file log4j.properties for bpm.log, bpm-runtime.log & bpm_trace.log
cs-logging-configmap	Dctm server	Configuration of dfc trace, docbase trace & CS container logs
d2-logback-configmap	Dctm server	Configuration of the properties file logback.xml for D2-JMS.log
d2classic-adminmessages-properties	D2 Classic	Configuration of the properties file adminMessages.properties
d2classic-antisamy-xml	D2 Classic	Configuration of the properties file antisamy.xml
d2classic-applicationcontext-xml	D2 Classic	Configuration of the properties file applicationContext.xml
d2classic-bravaparameters-properties	D2 Classic	Configuration of the properties file brava_parameters.properties
d2classic-ctfconfig-json	D2 Classic	Configuration of the properties file brava_parameters.properties
d2classic-d2-cache-xml	D2 Classic	Configuration of the properties file d2-cache.xml
d2classic-d2fs-properties	D2 Classic	Configuration of the properties file D2FS.properties
d2classic-dfc-properties	D2 Classic	Configuration of the properties file dfc.properties
d2classic-esapi-properties	D2 Classic	Configuration of the properties file ESAPI.properties

<b>Config map name</b>	<b>Component</b>	<b>Purpose</b>
d2classic-log4j2-properties	D2 Classic	Configuration of the properties file log4j2.properties for D2-log4j.log
d2classic-logback-xml	D2 Classic	Configuration of the properties file logback.xml for D2.log
d2classic-settings-properties	D2 Classic	Configuration of the properties file settings.properties
d2classic-shiro-ini	D2 Classic	Configuration of the properties file shiro.ini
d2classic-smtpc6-properties	D2 Classic	Configuration of the properties file smtp_c6.properties
d2classic-validation-properties	D2 Classic	Configuration of the properties file validation.properties
d2classic-server-xml	D2 Classic	Configuration of the Tomcat server.xml file
d2classic-web-xml	D2 Classic	Configuration of the Tomcat web.xml file
d2config-d2-config-properties	D2 Config	Configuration of the properties file D2-Config.properties
d2config-dfc-properties	D2 Config	Configuration of the properties file dfc.properties
d2config-esapi-properties	D2 Config	Configuration of the properties file ESAPI.properties
d2config-log4j2-properties	D2 Config	Configuration of the properties file log4j2.properties for D2-Config-log4j.log
d2config-logback-xml	D2 Config	Configuration of the properties file logback.xml for D2-Config.log
d2config-validation-properties	D2 Config	Configuration of the properties file validation.properties
d2config-server-xml	D2 Config	Configuration of the Tomcat server.xml file

<b>Config map name</b>	<b>Component</b>	<b>Purpose</b>
d2config-web-xml	D2 Config	Configuration of the Tomcat web.xml file
d2rest-antisamy-xml	D2 REST	Configuration of the properties file antisamy.xml
d2rest-d2fs-properties	D2 REST	Configuration of the properties file D2FS.properties
d2rest-dfc-properties	D2 REST	Configuration of the properties file dfc.properties
d2rest-esapi-properties	D2 REST	Configuration of the properties file ESAPI.properties
d2rest-log4j2-properties	D2 REST	Configuration of the properties file log4j2.properties for rest-api-log4j.log
d2rest-logback-xml	D2 REST	Configuration of the properties file logback.xml for rest-api-logback.log
d2rest-rest-api-common-ehcache-xml	D2 REST	Configuration of the properties file rest-api-common-ehcache.xml
d2rest-rest-api-runtime-properties	D2 REST	Configuration of the properties file rest-api-runtime.properties
d2rest-settings-properties	D2 REST	Configuration of the properties file settings.properties
d2rest-trust-properties	D2 REST	Configuration of the properties file trust.properties
d2rest-validation-properties	D2 REST	Configuration of the properties file validation.properties
d2rest-server-xml	D2 REST	Configuration of the Tomcat server.xml file
d2rest-web-xml	D2 REST	Configuration of the Tomcat web.xml file
d2smartview-antisamy-xml	D2 Smartview	Configuration of the properties file antisamy.xml
d2smartview-bravaparameters-properties	D2 Smartview	Configuration of the properties file brava_parameters.properties

<b>Config map name</b>	<b>Component</b>	<b>Purpose</b>
d2smartview-d2fs-properties	D2 Smartview	Configuration of the properties file D2FS.properties
d2smartview-dfc-properties	D2 Smartview	Configuration of the properties file dfc.properties
d2smartview-esapi-properties	D2 Smartview	Configuration of the properties file ESAPI.properties
d2smartview-log4j2-properties	D2 Smartview	Configuration of the properties file log4j2.properties for rest-api-log4j.log
d2smartview-logback-xml	D2 Smartview	Configuration of the properties file logback.xml for D2-Smartview.log
d2smartview-rest-api-runtime-properties	D2 Smartview	Configuration of the properties file rest-api-runtime.properties
d2smartview-settings-properties	D2 Smartview	Configuration of the properties file settings.properties
d2smartview-sw-config-json	D2 Smartview	Configuration of the properties file sw-config.json
d2smartview-validation-properties	D2 Smartview	Configuration of the properties file validation.properties
d2smartview-server-xml	D2 Smartview	Configuration of the Tomcat server.xml file
d2smartview-web-xml	D2 Smartview	Configuration of the Tomcat web.xml file
da-appproperties-configmap	Documentum Administrator (DA)	Configuration of the properties file app.properties
da-logging-configmap	Documentum Administrator (DA)	Configuration of the properties file log4j2.properties for documentum.log
dbr.configmap	Documentum Server	Configuration of the properties file dfc.properties
dctm-rest-configmap	Documentum REST	Configuration of the properties file rest-api-runtime.properties
dctm-rest-configmap-ext	Documentum REST	Configuration of the properties file newrelic.yml

<b>Config map name</b>	<b>Component</b>	<b>Purpose</b>
dctm-rest-logging-configmap	Documentum REST	Configuration of the properties file log4j2.properties for dctm-rest-services.log
dctm-rest-rest-api-runtime-properties	Documentum REST	Configuration of the properties file rest-api-runtime.properties
dctm-workflow-designer-dctm-workflow-designer-config	Documentum Workflow Designer	Configuration of the different parameters for Documentum Workflow Designer
dctm-workflow-designer-dctm-workflow-designer-log-config	Documentum Workflow Designer	Configuration of the properties file log4j2.properties for WFDesigner Log, WFMigration log & WFImport log
dfs-server-dfs-config	Documentum Foundation Services	Configuration of the properties files dfc.properties & log4j.properties
dmotdsrest-logging-configmap	Documentum Server	Configuration of the properties file log4j.properties for dmotdsrest.log
logrotate-configmap	Documentum Server	Configuration of log rotation for catalina.out & server config log
new-relic.configmap	Documentum Server	Configuration of the properties file newrelic.yml
oauth-logging-configmap	Documentum Server	Configuration of the properties file log4j.properties for oauth.log
otdsauth-logging-configmap	Documentum Server	Configuration of the properties file log4j.properties for otdsauth.log
otdsws-configmap	OTDS	Configuration of the different parameters for the OTDS server
initdb-script	OTDS	Configuration of init-db.sh for the OTDS server
saml-logging-configmap	Documentum Server	Configuration of the properties file log4j.properties for SAMLAAuthentication.log

<b>Config map name</b>	<b>Component</b>	<b>Purpose</b>
serverapps-logging-configmap	Documentum Server	Configuration of the properties file log4j.properties for DmMethods.log
records-appproperties-configmap	Documentum Records	Configuration of the properties file app.properties
records-logging-configmap	Documentum Records	Configuration of the properties file log4j2.properties for documentum.log
records-otdsproperties-configmap	Documentum Records	Configuration of the properties file otdsoauth.properties
smartviewm365-server-xml	smartviewm365	Configuration of the Tomcat server.xml file
smartviewm365-web-xml	smartviewm365	Configuration of the Tomcat web.xml file
smartviewm365-launcher-json	smartviewm365	Configuration of the properties file launcher.json
d2rest-msgraph-properties	D2 REST	Configuration of the properties file msgraph.properties
bps--log4j-properties-config	BPS	Configuration of the properties file log4j2.properties for bps-all.log and bps.log
bps--bps-dfc-config	BPS	Configuration of the properties file dfc.properties
xda--xda-config	XDA	Configuration of the properties files dfc.properties and log4j.properties for xda.log
dctm-cmis-configmap	CMIS	Configuration of the properties file dfc.properties for cmis
dctm-cmis-configmap-ext	CMIS	Configuration of newrelic for cmis
dctm-cmis-logging-configmap	CMIS	Configuration of the properties file log4j2.properties for dctm-cmis-services.log

## 7.14 Manually deploying D2 artifacts

As an alternative to using the D2 installer, OpenText also provides access to D2 artifacts that you can use to manually deploy D2. Additionally, you can integrate these D2 artifacts into automated CI/CD pipelines or use them to create your own custom-built containers. This chapter includes instructions on how to manually deploy D2 using these artifacts.

### 7.14.1 Extracting D2 artifacts

1. Extract the content of D2 unpackaged folders from the support site:
  - d2\_core\_unpackaged\_23.4.0.zip
  - d2\_pluspack\_unpackaged\_23.4.0.zip
2. Copy all .jar files in the JMS FOLDER (\unpacked\d2\_core\_unpackaged\_23.4.0\JMS) to (<JMS\_HOME>/webapps/DmMethods/WEB-INF/lib).
3. If you are using BPM, copy all .jar files in the BPM folder to (<BPM\_HOME>/webapps/BPM/WEB-INF/lib).
4. Deploy D2.war and D2-Config.war from \unpacked\d2\_core\_unpackaged\_23.4.0 to the App server machine.
5. Deploy plugins:
  - a. Open \D2-Config\WEB-INF\classes and create a folder named plugins.
  - b. From ..\unpacked\d2\_core\_unpackaged\_.zip and ..\unpacked\d2\_pluspack\_unpackaged\_.zip, copy all the necessary plugins to the new plugins folder. The following plug-ins are required:
    - C2-Plugin-.jar
    - D2-Bin-Plugin-.jar
    - D2-Specifications-Plugin-.jar
    - D2-Widget-Plugin-.jar
    - O2-Plugin-.jar
    - P2-Plugin-.jar
6. From ..\unpacked\d2\_core\_unpackaged\_.zip and ..\unpacked\d2\_pluspack\_unpackaged\_.zip, copy the following plug-in API .jar files to D2-Config\WEB-INF\lib.
  - C2-API-.jar
  - D2-Bin-API-.jar
  - D2-Specifications-API-.jar
  - D2-Widget-API-.jar

- O2-API.jar
  - P2-API.jar
7. Add the following plug-ins entry to D2-Config\WEB-INF\classes\D2-Config.properties:
- ```
##### Plugins #####
plugin_1=plugins/C2-Plugin-23.4.0.jar
plugin_2=plugins/D2-Bin-Plugin-23.4.0.jar
plugin_3=plugins/D2-Specifications-Plugin-23.4.0.jar
plugin_4=plugins/D2-Widget-Plugin-23.4.0.jar
plugin_5=plugins/O2-Plugin-23.4.0.jar
plugin_6=plugins/P2-Plugin-23.4.0.jar
```
8. Install the DARS manually in the following order:
- D2-DAR.dar
  - D2Widget-DAR.dar
  - D2Widget-DAR\_DB2.dar (only required when using a DB2 database)
  - Collaboration\_Services.dar
  - brava\_formats.dar (only required when using Brava)
  - BravaPublication.dar (only required when using Brava)
  - C2-DAR.dar
  - O2-DAR.dar
  - D2-BIN.dar
9. Delete jars in D2-Config(\webapps\D2-Config\WEB-INF\lib):
- D2-Core-Signature-Plugin-23.4.0.jar
  - D2-Specifications-Plugin-23.4.0.jar
  - D2-Widget-Plugin-23.4.0.jar
  - D2-xECM-Plugin-23.4.0.jar
10. Delete jars from (<JMS\_HOME>/DmMethods/WEB-INF/lib):
- D2-Widget-Install-23.4.0.jar
  - D2-Specifications-Install-23.4.0.jar
11. Add the below jars to D2 (\webapps\D2\WEB-INF\lib):
- C2-API-23.4.0.jar
  - C2-Plugin-23.4.0.jar
  - D2-Bin-API-23.4.0.jar
  - D2-Bin-Plugin-23.4.0.jar
  - O2-API-23.4.0.jar

- O2-Plugin-23.4.0.jar
  - D2-Specifications-API-23.4.0.jar
  - D2-Specifications-Plugin-23.4.0.jar
  - D2-Widget-API-23.4.0.jar
  - D2-Widget-Plugin-23.4.0.jar
  - P2-API-23.4.0.jar
  - P2-Plugin-23.4.0.jar
12. Add the below jars to DmMethod ((<JMS\_HOME>/webapps/DmMethods/WEB-INF/lib):
- C2-API-23.4.0.jar
  - C2-Plugin-23.4.0.jar
  - D2-Bin-API-23.4.0.jar
  - D2-Bin-Plugin-23.4.0.jar
  - O2-API-23.4.0.jar
  - O2-Plugin-23.4.0.jar
  - stax-ex.jar
  - commons-compress-1.21.jar
  - commons-io-2.10.0.jar

## 7.14.2 Adding D2.war and D2–Config.war to the App server

Copy D2.war and D2–Config.war to the App server.

## 7.14.3 Deploying plugins

For this task you need the following plug-in packages:

- d2\_core\_unpackaged\_23.4.zip
  - d2\_pluspack\_unpackaged\_23.4.zip
1. Open <App server>\D2–Config\WEB-INF\classes and create a folder name plugins.
  2. From “..\\unpackaged\\d2\_core\_unpackaged\_<version>.zip” and “..\\unpackaged\\d2\_pluspack\_unpackaged\_<version>.zip”, copy all of the necessary plug-ins to the new plugins folder. The following plug-ins are required:
    - C2-Plugin-<version>.jar
    - D2-Bin-Plugin-<version>.jar

- D2-Specifications-Plugin-<version>.jar
  - D2-Widget-Plugin-<version>.jar
  - O2-Plugin-<version>.jar
  - P2-Plugin-<version>.jar
3. From “..\\unpackaged\\d2\_core\_unpackaged\_<version>.zip” and “..\\unpackaged\\d2\_pluspack\_unpackaged\_<version>.zip”, copy the following plug-in API.jar files to D2-Config\\WEB-INF\\lib:
- C2-API-<version>.jar
  - D2-Bin-API-<version>.jar
  - D2-Specifications-API-<version>.jar
  - D2-Widget-API-<version>.jar
  - O2-API-<version>.jar
  - P2-API-<version>.jar
4. Add the following plug-ins entry to D2-Config\\WEB-INF\\classes\\D2-Config.properties:

```
#####
Plugins
#####
plugin_1=plugins/C2-Plugin-23.4.0.jar
plugin_2=plugins/D2-Bin-Plugin-23.4.0.jar
plugin_3=plugins/D2-Specifications-Plugin-23.4.0.jar
plugin_4=plugins/D2-Widget-Plugin-23.4.0.jar
plugin_5=plugins/O2-Plugin-23.4.0.jar
plugin_6=plugins/P2-Plugin-23.4.0.jar
```

#### 7.14.4 Installing DARS

Manually install the DARS in the following order:

1. D2-DAR.dar
2. D2Widget-DAR.dar
3. D2Widget-DAR\_DB2.dar (Only required when using a DB2 database.)
4. Collaboration\_Services.dar
5. brava\_formats.dar (Only required when using Brava.)
6. BravaPublication.dar (Only required when using Brava.)

### 7.14.5 Adding D2-Smartview.war to the App server

Copy D2-Smartview.war to the App server.

### 7.14.6 Installing RPS components

The following steps are only required if you are using RPS.

1. Use Composer to install rps.dar, rm.dar, and prm.dar on the Content Server repository.
2. Deploy the records.war file on Tomcat. You should be able to run the records webapp.
3. Open the lib directory of the records webapp (records\WEB-INF\lib) that you deployed on Tomcat.
4. Copy the following .jar files to the lib directory of JMS (C:\Documentum\tomcat9.0.41\shared\lib):
  - DmcPolicyEngine.jar
  - DmcRecords.jar
  - DmcRps.jar
  - IDmcPolicyEngine.jar
  - IDmcRps.jar
  - IDmcRpsModules.jar
5. From the Knowledge Center records location, download rps-webservices-23.4.zip and extract the contents.
6. Copy the following .jar files to the lib directory of JMS (C:\Documentum\<version>\shared\lib) and to the records webapp:
  - emc-policy-services.jar
  - emc-retentionmarkup-services.jars
7. Add the Records client as a privileged client and approve the privileges in the Client Rights Management node in Documentum Administrator.

## 7.15 Intelligent Viewing configuration

### 7.15.1 Enabling OTIV in D2 single Helm



**Note:** OpenText Intelligent Viewer 23.2 supports PostgreSQL DB versions 14.x and below only.

#### To enable OTIV:

- Update trustedSourceOrigins with the ingress domain URL through which the Smart View service is accessible.

```
otiv:  
  global:  
    trustedSourceOrigins: https://dctm-ingress.d2.cfcr-lab.bp-paas.otlab.net
```

### 7.15.2 Configuring OTDS for IV

#### 7.15.2.1 Syncing OTIV users to docbase

##### To sync OTIV users to docbase:

- Go to **Access Roles**.
- Click on **Actions** of the D2 access role and select **View access role details**.
- Add the OAuthClients partition that was created by OTIV and save the access role.
- Go to **Resources** and select **D2 resource**, **Actions --> Consolidate**.

#### 7.15.2.2 Allocating the license to users

##### To allocate the license to users:

- Go to **Users and Groups**.
- Select the required user and click on **Actions --> Allocate to License**.
- From the **License** dropdown, select **Viewing-iv** and from the **Counter** dropdown select **INTELLIGENT\_VIEWING.FULLTIME\_USERS\_BASIC** or **INTELLIGENT\_VIEWING.FULLTIME\_USERS\_REGULAR**, according to the need.
- Consolidate the users by following the steps in the previous section.

### 7.15.2.3 Adding/Updating oAuth clients and system attributes

**To add/update oAuth clients and system attributes:**

1. Go to **Oauth Clients** and select the required client (where Smart View is added as Redirect URLs).
2. Click on **Actions --> Properties --> Advanced**.
3. Check the box for **Use session lifetime as Refresh token lifetime** and provide **Access Token life time as 1000**.
4. Add the below values in Permissible scopes, Default scopes and Save the Oauth client:
  - create\_publications
  - view\_publications
  - view\_any\_publication
  - delete\_any\_publication
  - delete\_publications
  - write\_any\_markups
  - read\_any\_markups
5. Add the below attributes in **System Config**.
  - otds.as.SameSiteCookieVal - none
  - directory.auth.EnforceSSL - false

### 7.15.3 Configuring Intelligent Viewing

After installing Intelligent Viewing and applying OTDS licenses, you must configure Intelligent Viewing using D2-Config. For details on configuring Intelligent Viewing, see the *OpenText Documentum D2 and OpenText Documentum D2 Mobile - Administration Guide*.

## Chapter 8

# Upgrading and Migrating Documentum applications

## 8.1 Upgrading Documentum Server

### 8.1.1 Upgrading Documentum Server to 23.4

#### 8.1.1.1 Important tasks and notes to upgrade to 23.4

This section provides information about the important tasks that you must perform and notes that you must consider before upgrading Documentum Server to 23.4.

##### 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 in the upgrade process.



#### Caution

- The configuration value of `install.appserver.admin.password` must comply with the password complexity rules. The *Documentum Server* chapter in *OpenText Documentum 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 `dctm-server.cs-secrets`.
- If a mismatch exist in the configuration values, it impacts the deployment of 23.4 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 23.4.

#### 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 previous release to the documentum/charts/dctm-server/platforms/<cloud platform>.yaml file of 23.4.

#### 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 dctm-server.cs-secrets section in the single All-In-One documentum/values.yaml Helm chart file. In addition, provide the new host value of the migrated database for the dctm-server.content-server.database.host and dctm-server.content-server.database.port in common variables in “[Defining common variables](#)” on page 53 in the single All-In-One documentum/values.yaml Helm chart file.

#### 8.1.1.1.5 Modifying service account name

1. Upgrade the Documentum Server 22.4 or 23.2 pod to 23.4 using the steps described in “[Upgrading from 22.4 and 23.2 to 23.4](#)” on page 439 with the same service account name that was used in the previous deployment.
2. Set the value of dctm-server.docbroker.createserviceaccount to true.
3. Provide a new service account name for documentumserviceaccount in common-variables in “[common-variables](#)” on page 53.
4. Run the Helm upgrade command using the following command format:

```
helm upgrade <release name> <location where Helm charts are extracted> --values  
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values  
<location where Helm charts are extracted>/dockerimages-values.yaml --values  
<location where Helm charts are extracted>/<config>.yaml --values <location where  
Helm charts are extracted>/documentum-components.yaml --namespace <name of  
namespace>
```

where <config> is -extra-large-enhanced, -extra-large-standard, -large-enhanced, -large-standard, -medium-enhanced, -medium-large-enhanced, -medium-large-standard, -medium-standard, -small-enhanced, -small-medium-enhanced, -small-medium-standard, -small-standard, or -test-small resource value YAML file that has been provided. These resource files contain pod sizing values like CPU, memory, and so on.

### 8.1.1.2 Upgrading from 22.4 and 23.2 to 23.4

1. Update the new image location details for `imageRepository` and `imageTag` in the single All-In-One `documentum/dockerimage-values.yaml` Helm chart file of 23.4.
2. Copy all the required configuration values from the `common-variables` section of the previous deployment to the `common-variables` section of the single All-In-One `documentum/values.yaml` Helm chart file of 23.4. In addition, provide the appropriate values for the new variables (if any and required according to your requirement) in 23.4.
3. Copy all the required configuration values from the `db` section of the previous deployment to the `dctm-server.db` section of the single All-In-One `documentum/values.yaml` Helm chart file of 23.4. In addition, provide the appropriate values for the new variables (if any and required according to your requirement) in 23.4.

**!** **Important**

Make sure that the database version is same for both the existing and new deployment.

For example, if your 22.4 deployment is running with PostgreSQL 14.4 database container, then in your 23.4 upgrade, make sure that you change the database version to 14.4 as follows:

- Value of `db.version` in `documentum/charts/dctm-server/Chart.yaml`.
  - Value of `dctm-server.db.tag` in `documentum/dockerimages-values.yaml`.
4. Copy all the required configuration values from `docbroker` of the previous deployment to `dctm-server.docbroker` of the single All-In-One `documentum/values.yaml` Helm chart file of 23.4. In addition, provide the appropriate values for the new variables (if any and required according to your requirement) in 23.4.



#### 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 443](#) 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.

5. Copy all the required configuration values from `content-server` of the previous deployment to `dctm-server.content-server` of the single All-In-One `documentum/values.yaml` Helm chart file of 23.4. In addition, provide the appropriate values for the new variables (if any and required according to your requirement) in 23.4.

### 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 443](#) contains detailed information.
6. Copy all the required configuration values from `cs-logging-configMap` of the previous deployment to `dctm-server.cs-logging-configMap` of the single All-In-One `documentum/values.yaml` Helm chart file of 23.4. In addition, provide the appropriate values for the new variables (if any and required according to your requirement) in 23.4.
  7. Copy all the required configuration values from `dctm-ingress` of the previous deployment to `dctm-server.dctm-ingress` of the single All-In-One `documentum/values.yaml` Helm chart file of 23.4. In addition, provide the appropriate values for the new variables (if any and required according to your requirement) in 23.4.
  8. Copy all the required configuration values from `cs-dfc-properties` of the previous deployment to `dctm-server.cs-dfc-properties` of the single All-In-One `documentum/values.yaml` Helm chart file of 23.4. In addition, provide the appropriate values for the new variables (if any and required according to your requirement) in 23.4.
  9. Copy all the required configuration values from the `otds` section of the previous deployment to the `otds` section of the single All-In-One `documentum/values.yaml` Helm chart file of 23.4. In addition, provide the appropriate values for the new variables (if any and required according to your requirement) in 23.4.

10. **Optional** 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 403.
11. Upgrade the 22.4 or 23.2 Documentum Server pod using the following command format:

```
helm upgrade <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where
Helm charts are extracted>/documentum-components.yaml --namespace <name of
namespace>
```

where <config> is -extra-large-enhanced, -extra-large-standard, -large-enhanced, -large-standard, -medium-enhanced, -medium-large-enhanced, -medium-large-standard, -medium-standard, -small-enhanced, -small-medium-enhanced, -small-medium-standard, -small-standard, or -test-small resource value YAML file that has been provided. These resource files contain pod sizing values like CPU, memory, and so on.

12. 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:

```
kubectl 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>.<server_config_name>
```

- 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.
- j. After the successful upgrade, you can upgrade AEK as described in [“Rotating AEK in Documentum Server” on page 419](#).



**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 23.4 environment

If you want to upgrade from a non-certificate 22.4 or 23.2 environment to a non-certificate 23.4 environment where both the environments were deployed and then you want to enable certificate-based communication in the upgraded 23.4 environment, you must perform the following tasks:

1. Upgrade the non-certificate 22.4 or 23.2 environment to a non-certificate 23.4 environment.  
[“Upgrading from 22.4 and 23.2 to 23.4” on page 439](#) contains the instructions.
2. Set the value of enable of dctm-server.content-server.customUpgrade to true ([“content-server” on page 72](#)) in the single All-In-One documentum/values.yaml Helm chart file of 23.4. In addition, provide the appropriate values for the new variables (if any and required according to your requirement) in 23.4. Then, upgrade the updated 23.4 environment using the Helm upgrade command.
3. Set the value of use\_certificate to true in common variables ([“common-variables” on page 53](#)) in the single All-In-One documentum/values.yaml Helm chart file of 23.4. In addition, provide the appropriate values for the new variables (if any and required according to your requirement) in 23.4.
4. Specify all the certificate-related configuration values for docbroker and content-server in dctm-server.cs-secrets ([“cs-secrets” on page 60](#)) in the single All-In-One documentum/values.yaml Helm chart file of 23.4. In addition, provide the appropriate values for the new variables (if any and required according to your requirement) in 23.4.

5. Modify the value of `customUpgrade.enable` to `false` in `dctm-server.content-server` ("content-server" on page 72) in the single All-In-One documentum/values.yaml Helm chart file of 23.4.
6. Upgrade the updated 23.4 environment using the following command format:

```
helm upgrade <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where
Helm charts are extracted>/documentum-components.yaml --namespace <name of
namespace>
```

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

7. 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 22.4 or 23.2 image if AEK 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 22.4 or 23.2 image if AEK is upgraded:

1. Retrieve the details of history using the following command format:

```
helm history <release name>
```

2. Upgrade AEK.

- a. Change the value of `aeK.name` in `dctm-server.content-server.contentserver` to the same key name mentioned in the 22.4 or 23.2 image.
- b. Change the value of `aeK.location` in `dctm-server.content-server` to the same location mentioned in the 22.4 or 23.2 image.
- c. Change the value of `aeK.algorithm` in `dctm-server.cs-secrets` to the same key algorithm mentioned in the 22.4 or 23.2 image.
- d. Do not change the value of `aeK.passphrase` in `dctm-server.cs-secrets`. It should be same for the upgraded image and the previous image.
- e. Run the Helm upgrade command using the following command format:

```
helm upgrade <release name> <location where Helm charts are extracted> --values <location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values <location where Helm charts are extracted>/dockermimages-values.yaml --values <location where Helm charts are extracted>/<config>.yaml --values <location where Helm charts are extracted>/documentum-components.yaml --namespace <name of namespace>
```

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

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 22.4 or 23.2 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 23.4 in `common-variables` to the same value mentioned in `dctm-server.cs-secrets` in the single All-In-One `documentum/values.yaml` Helm chart file) of 22.4 or 23.2.

- b. Run the Helm upgrade command using the following command format:

```
helm upgrade <release name> <location where Helm charts are extracted> --values <location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values <location where Helm charts are extracted>/dockermimages-values.yaml --values <location where Helm charts are extracted>/<config>.yaml --values <location where Helm charts are extracted>/documentum-components.yaml --namespace <name of namespace>
```

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

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 22.4 or 23.2 image if service account name is modified and upgraded:**

If you have modified the service account name during the upgrade as described in “[Modifying service account name](#)” on page 438, then to roll back the previous image release version, perform the following steps:

1. Retrieve the details of history using the following command format:

```
helm history <release name>
```

For example:

| REVISION | UPDATED                 | STATUS     | CHART             | APP VERSION | DESCRIPTION                                               |
|----------|-------------------------|------------|-------------------|-------------|-----------------------------------------------------------|
| 1        | Wed Oct 4 13:19:54 2023 | superseded | documentum-m-23.2 | 23.2        | Install complete                                          |
| 2        | Wed Oct 4 14:10:40 2023 | superseded | documentum-m-23.4 | 23.4        | Upgrade complete //major upgrade                          |
| 3        | Wed Oct 4 19:02:54 2023 | deployed   | documentum-m-23.4 | 23.4        | Upgrade complete // service account name modified version |

2. Roll back to the previous deployment release version with previous service account name (revision 2).

For example:

```
helm rollback <release name> 2
```

3. Roll back to the previous release image version (revision 1).

For example:

```
helm rollback <release name> 1
```



## 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 AEK during the upgrade process.

## 8.2 Upgrading Documentum Administrator

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

Run the Helm upgrade command using the following command format:

```
helm upgrade <release name> <location where Helm charts are extracted> --values  
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values  
<location where Helm charts are extracted>/dockerimages-values.yaml --values <location  
where Helm charts are extracted>/<config>.yaml --values <location where Helm charts are  
extracted>/documentum-components.yaml --namespace <name of namespace>
```

where `<config>` is `-extra-large-enhanced`, `-extra-large-standard`, `-large-enhanced`, `-large-standard`, `-medium-enhanced`, `-medium-large-enhanced`, `-medium-large-standard`, `-medium-standard`, `-small-enhanced`, `-small-medium-enhanced`, `-small-medium-standard`, `-small-standard`, or `-test-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 dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/platforms/  
cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --values /opt/temp/  
documentum/<config>.yaml --values /opt/temp/documentum/documentum-components.yaml --  
namespace onedctmns
```



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

```
helm upgrade --recreate-pods <release name> <location where Helm charts are extracted> --  
values <location where Helm charts are extracted>/platforms/<cloud platform>.yaml --  
values <location where Helm charts are extracted>/dockerimages-values.yaml --values  
<location where Helm charts are extracted>/<config>.yaml --values <location where Helm  
charts are extracted>/documentum-components.yaml --namespace <name of namespace>
```

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

### 8.2.1 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>
```

## 8.3 Upgrading Documentum REST Services

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

Run the Helm upgrade command using the following command format:

```
helm upgrade <release name> <location where Helm charts are extracted> --values
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values
<location where Helm charts are extracted>/dockerimages-values.yaml --values <location
where Helm charts are extracted>/<config>.yaml --values <location where Helm charts are
extracted>/documentum-components.yaml --namespace <name of namespace>
```

where `<config>` is `-extra-large-enhanced`, `-extra-large-standard`, `-large-enhanced`, `-large-standard`, `-medium-enhanced`, `-medium-large-enhanced`, `-medium-large-standard`, `-medium-standard`, `-small-enhanced`, `-small-medium-enhanced`, `-small-medium-standard`, `-small-standard`, or `-test-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 dcmdeployment /opt/temp/documentum --values /opt/temp/documentum/platforms/
cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --values /opt/temp/
documentum/<config>.yaml --values /opt/temp/documentum/documentum-components.yaml --
namespace onedctmns
```



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

```
helm upgrade --recreate-pods <release name> <location where Helm charts are extracted> --
values <location where Helm charts are extracted>/platforms/<cloud platform>.yaml --
values <location where Helm charts are extracted>/dockerimages-values.yaml --values
<location where Helm charts are extracted>/<config>.yaml --values <location where Helm
charts are extracted>/documentum-components.yaml --namespace <name of namespace>
```

### 8.3.1 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>
```

## 8.4 Upgrading Documentum Foundation Services

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

Run the Helm upgrade command using the following command format:

```
helm upgrade <release name> <location where Helm charts are extracted> --values  
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values  
<location where Helm charts are extracted>/dockerimages-values.yaml --values <location  
where Helm charts are extracted>/<config>.yaml --values <location where Helm charts are  
extracted>/documentum-components.yaml --namespace <name of namespace>
```

where `<config>` is `-extra-large-enhanced`, `-extra-large-standard`, `-large-enhanced`, `-large-standard`, `-medium-enhanced`, `-medium-large-enhanced`, `-medium-large-standard`, `-medium-standard`, `-small-enhanced`, `-small-medium-enhanced`, `-small-medium-standard`, `-small-standard`, or `-test-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 dctmdeployment /opt/temp/documentum --values /opt/temp/documentum/platforms/  
cfcr.yaml --values /opt/temp/documentum/dockerimages-values.yaml --values /opt/temp/  
documentum/<config>.yaml --values /opt/temp/documentum/documentum-components.yaml --  
namespace onedctmns
```



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

```
helm upgrade --recreate-pods <release name> <location where Helm charts are extracted> --  
values <location where Helm charts are extracted>/platforms/<cloud platform>.yaml --  
values <location where Helm charts are extracted>/dockerimages-values.yaml --values  
<location where Helm charts are extracted>/<config>.yaml --values <location where Helm  
charts are extracted>/documentum-components.yaml --namespace <name of namespace>
```

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

### 8.4.1 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>
```

## 8.5 Migrating on-premises Documentum applications to cloud platform

### 8.5.1 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

#### 8.5.1.1 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).

#### 8.5.1.2 Creating secrets

1. Open the single All-In-One documentum/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 56 section contains detailed information.
2. Set the value of enabled in dcm-server.cs-secrets in the documentum/documentum-components.yaml Helm chart file to true.
3. Set the value of enabled of all other categories in the documentum/documentum-components.yaml Helm chart file to false.
4. Store the secret values using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values <location where Helm charts are extracted>/config/configuration.yml --values
```

```
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values  
<location where Helm charts are extracted>/dockerimages-values.yaml --values  
<location where Helm charts are extracted>/<config>.yaml --values <location where  
Helm charts are extracted>/documentum-components.yaml --namespace <name of  
namespace>
```

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

### 8.5.1.3 Creating database

The migration of the PostgreSQL database only is supported.

1. Set the value of `enabled` in `dctm-server.db` in the `documentum/documentum-components.yaml` Helm chart file to `true`.
2. Set the value of `enabled` of all other categories in the `documentum/documentum-components.yaml` Helm chart file 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> <location where Helm charts are extracted> --values <location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values <location where Helm charts are extracted>/dockerimages-values.yaml --values <location where Helm charts are extracted>/<config>.yaml --values <location where Helm charts are extracted>/documentum-components.yaml --namespace <name of namespace>
```

where <config> is -extra-large-enhanced, -extra-large-standard, -large-enhanced, -large-standard, -medium-enhanced, -medium-large-enhanced, -medium-large-standard, -medium-standard, -small-enhanced, -small-medium-enhanced, -small-medium-standard, -small-standard, or -test-small resource value YAML file that has been provided. These resource files contain pod sizing values like CPU, memory, and so on.

- ii. Log in to the database pod using the following command format:

```
kubectl exec -ti <name of new database pod> bash
```

For example:

```
kubectl 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.

### 8.5.1.4 Creating connection broker

To create a connection broker in your cloud platform, perform the steps from [step 8.c](#) to [step 12](#) in “[Deploying Documentum Server](#)” on page 58. You must perform the following tasks:

1. Set the value of `enabled` in `docbroker` in the `documentum/documentum-components.yaml` Helm chart file to `true`.
2. Set the value of `enabled` of all other categories in the `documentum/documentum-components.yaml` Helm chart file to `false`.
3. Deploy the connection broker pod using the following command format:

```
helm install <release name> <location where Helm charts are extracted> --values  
<location where Helm charts are extracted>/config/configuration.yml --values  
<location where Helm charts are extracted>/platforms/<cloud platform>.yaml --values  
<location where Helm charts are extracted>/dockerimages-values.yaml --values  
<location where Helm charts are extracted>/documentum-resources-values-test-  
small.yaml --values <location where Helm charts are extracted>/documentum-  
components.yaml --namespace <name of namespace>
```

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

### 8.5.1.5 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: migrationoncs  
spec:  
containers:
```

```

- name: migrationscs
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 variables:
    - Set the value of dcm-server.content-server.enabled to true in the documentum/documentum-components.yaml Helm chart file.
    - Set the value of enabled of all other categories to false in the documentum/documentum-components.yaml Helm chart file.
    - Set the value of dtm-server.content-server.docbase.existing to true in the single All-In-One documentum/values.yaml Helm chart file.
    - Set the value of migration.migratecs to true in the documentum/charts/dcm-server/charts/content-server/values.yaml Helm chart file.
    - (Only if you want to migrate from Windows) Set the value of migration.migfromwindows to true in the documentum/charts/dcm-server/charts/content-server/values.yaml Helm chart file.
    - Make sure that you provide the same values for secret you created in “Creating secrets” on page 449, same values for database you created in “Creating database” on page 450, and same values for docbroker created in “Creating connection broker” on page 452. In addition, make sure all the other values used in on-premises Documentum Server is also updated in the single All-In-One documentum/values.yaml Helm chart file.
    - Make sure that retain the same values for docbase.name, docbase.id, docbase.owner, and docbase.index in the cloud environment similar to on-premises environment.
  - b. Set the value of oldDocumentumHome in documentum/charts/dcm-server/charts/values.yaml to previous deployment of Documentum Server (on-premises) path.

- c. To access S3 store, perform the following tasks in `dctm-server.content-server` in `documentum/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 8.d](#) to [step 12](#) in “[Deploying Documentum Server](#)” [on page 58](#).
- 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.

### 8.5.2 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 449](#) contains detailed information.
2. Create the connection broker pod. “[Creating connection broker](#)” [on page 452](#) contains detailed information.
3. Create the Documentum Server pod. “[Migrating Documentum Server](#)” [on page 452](#) 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.

### 8.5.3 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.

