

opentext™

Learning to deploy Documentum on Kubernetes

A tutorial using Docker Desktop Kubernetes



José María Sotomayor

Presentation

With this tutorial we are providing you with:

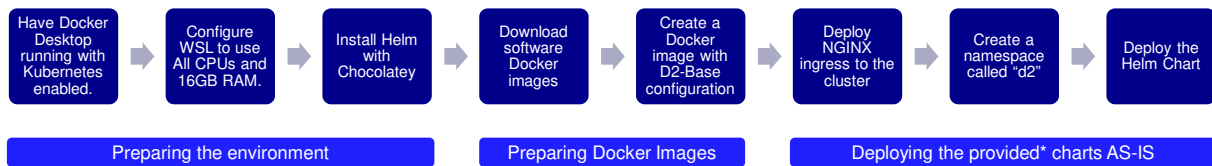
- Instructions on how to have your own Kubernetes cluster in your laptop.
- A D2 Helm Chart already configured. You can deploy a complete Documentum stack with a single command.
- As much practical information as possible about how this is done, so you can use it as foundations for further learning.

Slides go straight to the point, but I've added extensive footnotes with additional information when possible.

Benefits:

- Quickly deploy the 23.2 Documentum stack with a single command.
- Learning how you can do it yourself in the future, if needed.
- Familiarize yourself with Kubernetes technology.
- Not having to pay for a cloud Kubernetes service just for learning purposes.
- Quickly deploy new versions of the Documentum stack as soon as they are released.

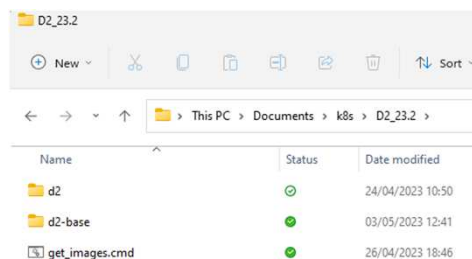
Steps



** In this tutorial we will refer to the Helm charts already configured and ready to be deployed as "the provided" charts, as opposed to "the original" charts, this is, the charts as you would download from Opentext Support.*

Getting the tutorial package

- Download the file named “Tutorial K8s D2 23_2.7z” accompanying this tutorial and extract it.
- The package contains three elements:
 - **“d2” folder** – Is the 23.2 Helm chart already configured to be deployed in your Kubernetes cluster with a single command. We will refer to them as “the provided” charts, for the purpose of this tutorial.
 - **“d2-base” folder** – contains some artifacts required to deploy the D2 Base configuration as part of your deployment.
 - **get_images.cmd** – it’s a very simple script that will pull the required Docker images from Opentext’s Docker registry.



Preparing the environment

Setting up your laptop to run Kubernetes using Docker Desktop.

Installing Docker Desktop with Kubernetes Support (1)

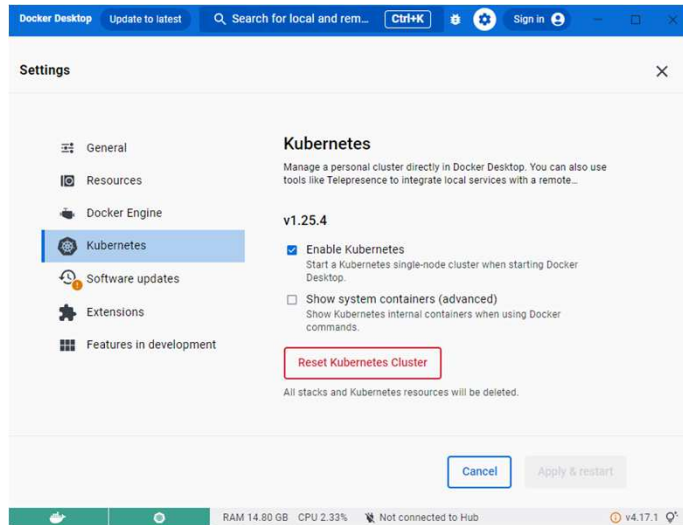
- Docker Desktop has an option to enable Kubernetes. It deploys a complete k8s cluster in your laptop.
- A prerequisite to use this kubernetes flavour is to have WSL2 enabled. Complete instructions can be found here: <https://learn.microsoft.com/en-us/windows/wsl/install>
- In most cases, opening an elevated Powershell and issuing the command `wsl --install` will suffice (you may need to enable it before under “Add Windows Features”).
- Instructions to install Docker Desktop can be found here: <https://docs.docker.com/desktop/install/windows-install/>
- No need to install additional Linux distros. Docker Desktop will install the required ones.

```
PS C:\WINDOWS\system32> wsl --list
Windows Subsystem for Linux Distributions:
docker-desktop-data (Default)
docker-desktop
PS C:\WINDOWS\system32>
```

Installing Docker Desktop with Kubernetes Support (2)

- Once it's installed and running, click settings, select "Kubernetes" in the left pane and click on "Enable Kubernetes". You should see both Docker and k8s in green after a while.
- You can double check your cluster is running with `kubectl get nodes` command:

```
PS C:\WINDOWS\system32> kubectl get nodes
NAME                STATUS    ROLES    AGE   VERSION
docker-desktop      Ready    control-plane   8d   v1.25.4
PS C:\WINDOWS\system32>
```



Configuring WSL for additional CPU&RAM

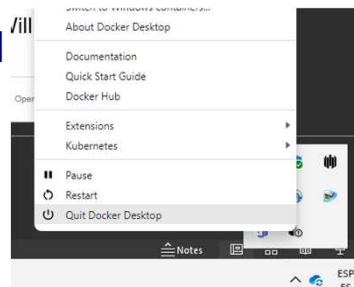
- We recommend host system to feature 32GB RAM, so at least 16GB can be easily assigned to WSL2's VM.
- The CPU and RAM resources of WSL2 by default, are not enough for deploying the Documentum platform.
- Shutdown Docker desktop.
- Shutdown WSL2 by issuing `wsl --shutdown`
- Create a new file under `C:\Users\<your_user_name>` called `.wslconfig` (dot wslconfig) with the following content (copy and paste from this slide notes):

```
# Settings apply across all Linux distros running on WSL 2
[wsl2]

# Limits VM memory to use no more than 16 GB, this can be set as whole numbers using GB or MB
memory=16GB

#Comment (or do not include) the following line so VM uses all available Logical Processors
#processors=6
```

- Next time you launch Docker Desktop, the new values (all vCPUs & 16GB RAM) will be picked up.



-----Source for .wslconfig-----

```
# Settings apply across all Linux distros running on WSL 2
[wsl2]
```

```
# Limits VM memory to use no more than 16 GB, this can be set as whole numbers
using GB or MB
memory=16GB
```

```
#Comment (or do not include) the following line so VM uses all available Logical
Processors
#processors=6
```


Installing Helm with Chocolatey

- Helm will be used to automatically deploy our workload to the k8s cluster using Helm charts.
- Easiest way to install Helm is via Chocolatey package manager (on Windows)
- Chocolatey can be downloaded from: <https://chocolatey.org/install>
- Once Chocolatey is installed, issue the following command from an elevated Powershell:
 - `choco install Kubernetes-Helm`
- Deploy mysql to your cluster to test Helm is working as expected:
 - `helm repo add bitnami https://charts.bitnami.com/bitnami`
 - `helm repo update`
 - `helm install bitnami/mysql --generate-name`
- Test mysql is running by issuing:
 - `kubectl get pods`
- Once is working, uninstall it to save resources:

```
PS C:\WINDOWS\system32> helm uninstall mysql-1683196330
release "mysql-1683196330" uninstalled
```

```
PS C:\WINDOWS\system32> helm install bitnami/mysql --generate-name
NAME: mysql-1683196330
LAST DEPLOYED: Thu May 4 12:32:10 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: mysql
CHART VERSION: 0.9.0
APP VERSION: 8.0.33
```

```
PS C:\WINDOWS\system32> kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
mysql-1683196330-0  1/1     Running   0           53s
```

Preparing Docker images

Downloading Documentum binary images and creating an image with the D2 Base configuration

Downloading software images

- Binaries are downloaded as Docker images from Opentext's Docker registry.
- Login into Opentext's Docker registry with your Opentext username and password:

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

- Pull images from the registry. For your convenience, use a cmd script such as:

```
k8s > D2_23.2 > get_images.cmd
1 docker pull postgres:15.1
2 docker tag postgres:15.1 registry.opentext.com/cs/pg:15.1
3 docker pull registry.opentext.com/dctm-d2pp-classic-ol:23.2
4 docker pull registry.opentext.com/dctm-d2pp-config-ol:23.2
5 docker pull registry.opentext.com/dctm-d2pp-installer-ol:23.2
6 docker pull registry.opentext.com/dctm-d2pp-rest-ol:23.2
7 docker pull registry.opentext.com/dctm-d2pp-smartview-ol:23.2
8 docker pull registry.opentext.com/dctm-d2pp-ijms-ol:23.2
9 docker pull registry.opentext.com/dctm-tomcat:23.2
10 docker pull registry.opentext.com/dctm-server:23.2
```

- This operation may take quite some time (only once) depending on your internet connection.
- A sample script is provided, valid for 23.2. (get_images.cmd). Not all images are used.

opentext

OpenText ©2023 All rights reserved 11

The downloading (pulling) process will take some hours, but you can leave it totally unattended. Compare this to downloading every single binary installer one by one from Opentext Support.

Postgres database is pulled directly from Docker's hub (no need to be logged in). Is tagged as registry.opentext.com/cs/pg for convenience (there's a variable in Helm chart for the repository name –registry.opentext.com and it has been historically named as cs/pg by Engineering)

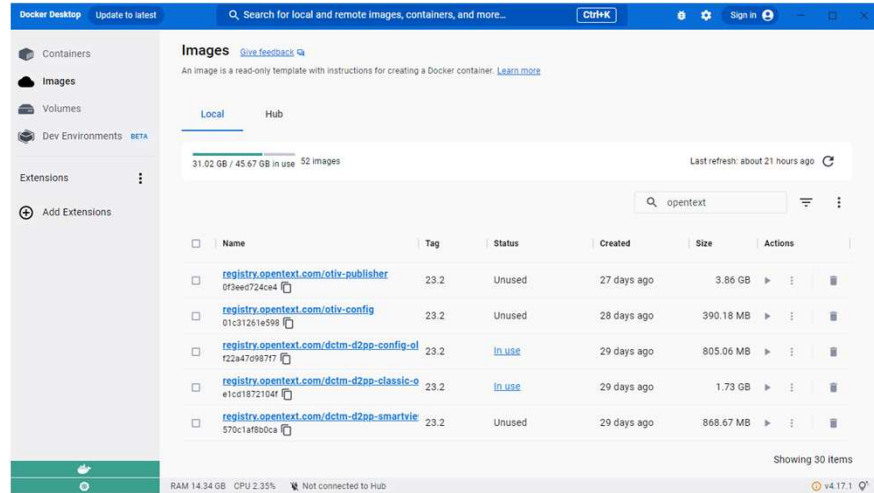
-----Source for get_images.cmd:-----

```
docker pull postgres:15.1
docker tag postgres:15.1 registry.opentext.com/cs/pg:15.1
docker pull registry.opentext.com/dctm-d2pp-classic-ol:23.2
docker pull registry.opentext.com/dctm-d2pp-config-ol:23.2
docker pull registry.opentext.com/dctm-d2pp-installer-ol:23.2
docker pull registry.opentext.com/dctm-d2pp-rest-ol:23.2
docker pull registry.opentext.com/dctm-d2pp-smartview-ol:23.2
docker pull registry.opentext.com/dctm-d2pp-ijms-ol:23.2
docker pull registry.opentext.com/dctm-tomcat:23.2
docker pull registry.opentext.com/dctm-server:23.2
```

```
docker pull registry.opentext.com/dctm-d2pp-ijms-ol:23.2
docker pull registry.opentext.com/dctm-xcp-installer:23.2
docker pull registry.opentext.com/dctm-xcp-apphost:23.2
docker pull registry.opentext.com/dctm-workflow-designer:23.2
docker pull registry.opentext.com/otds-server:23.1.1
docker pull registry.opentext.com/dctm-rest:23.2
docker pull registry.opentext.com/dctm-xplore-indexserver:22.1.2
docker pull registry.opentext.com/dctm-xplore-indexagent:22.1.2
docker pull registry.opentext.com/dctm-xplore-cps:22.1.2
docker pull registry.opentext.com/dctm-admin:23.2
docker pull registry.opentext.com/dctm-content-connect:23.2
docker pull registry.opentext.com/dctm-content-connect-dbinit:23.2
docker logout
```

Downloaded images are in your local Docker registry

- Docker keeps all downloaded images in a local registry.
- Select “Images” on the Docker Desktop app to see them.
- Filter by “opentext” to see all the images you’ve downloaded.



Creating a Docker image with D2-Base configuration

- You need to create a Docker image containing the D2-Base application DAR and ZIP files.
- The provided Helm charts are already configured to use it.
- In powershell, cd to the directory named “d2-base”. Inside there’s a Dockerfile that will create the image for you.
- Issue the following command to create the Docker image (one line, the dot at the end must remain)

```
docker build -f Dockerfile -t d2customdar:latest --build-arg  
CUSTOM_FILE_NAME=D2-Base-Export-Config.zip --build-arg CUSTOM_DAR_FILE=D2-  
Base.dar --no-cache .
```

```
PS C:\Users\jsotomay\OneDrive - OpenText\Documents\k8s\02_23_2\d2-base> docker build -f Dockerfile -t d2customdar:latest --build-arg CUSTOM_FILE_NAME=D2-Base-Export-Config.zip --build-arg CUSTOM_DAR_FILE=D2-Base.dar --no-cache .  
[*] Building 4.5s (9/9) FINISHED  
-> [internal] load build definition from Dockerfile 0.0s  
-> > transferring dockerfile: 376B 0.0s  
-> [internal] load .dockerignore 0.0s  
-> > transferring context: 2B 0.0s  
-> [internal] load metadata for docker.io/library/busybox:1.28 2.2s  
-> [internal] load build context 0.0s  
-> > transferring context: 66B.68kB 0.0s  
-> CACHED [1/4] FROM docker.io/library/busybox:1.28@sha256:141c253bc4c3fd8a281d32dc1f493bcf3fff003b6df416dea4f41046e0f37d47 0.0s  
-> [2/4] RUN adduser -D -H dmadin && mkdir -p /opt/D2-install/custom && chown -R dmadin:dmadin /opt/D2-install/custom 2.0s  
-> [3/4] COPY --chown=dmadin:dmadin D2-Base-Export-Config.zip /opt/D2-install/custom/ 0.0s  
-> [4/4] COPY --chown=dmadin:dmadin D2-Base.dar /opt/D2-install/custom/ 0.0s  
-> exporting to image 0.1s  
-> > exporting layers 0.1s  
-> > writing image sha256:465bbab005c3c27e5f642b0f467c487532f01a98c57b5a78926f319bee22c31e 0.0s  
-> naming to docker.io/library/d2customdar:latest 0.0s
```

Why is this step needed?

When CS and D2-Config pods do boot up, several initialization and installation scripts are run.

Some of these scripts are in charge of deploying DAR files and D2 configurations. These scripts expect to find DAR and config files at some preconfigured directories. But obviously, Opentext is shipping the Docker images without our configurations. How can we include them?

We are creating this small Docker image just containing (mostly) our DAR and ZIP files inside it.

During deployment, we are creating a container from our image, and we make it part of the CS and D2 Config pods.

This way, we are mounting the directories containing our files inside the original pod’s filesystem, so the scripts can pick them up for installation.

-----Source for Dockerfile -----

```
FROM busybox:1.28  
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
```

Deploying the provided chart AS-IS

All you need is ~~love~~ Helm

Deploying NGINX ingress to the cluster

- Docker Desktop's Kubernetes don't provide an ingress service by default.
- For POC/Demo purposes, a regular NGINX service will suffice (no need for NGINX+)
- To deploy the NGINX service, issue this command (one line):

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.7.0/deploy/static/provider/cloud/deploy.yaml
```

- This version is proved to work with the 23.2 stack and k8s version 1.25.4 (as installed by Docker Desktop v. 4.17.1). In the future you may need to find a more recent NGINX version.
- You can test your ingress controller is working with:

```
PS C:\WINDOWS\system32> kubectl get pods -n ingress-nginx
NAME                                READY   STATUS    RESTARTS   AGE
ingress-nginx-admission-create-rgc7l 0/1     Completed 0           7d
ingress-nginx-admission-patch-p9rlc   0/1     Completed 1           7d
ingress-nginx-controller-7d9674b7cf-xffzz 1/1     Running   4 (168m ago) 7d
```

- **Important:** see slide notes below.

opentext™

OpenText ©2023 All rights reserved 15

By default nginx is configured to accept request body sizes of maximum 1MB. This means that if you try to upload to D2 a document bigger than this size, an error will occur.

Please edit values.yaml and modify dctm-ingress definition to include the highlighted annotations (in bold), before deploying D2's Helm chart. Line is around 1092.

```
dctm-ingress:
  enabled: true
  #prefix for the ingress name
  ingressPrefix: dctm
  ingress:
    #No need to configure host: and clustrDomainName: if
configureHost is false.
    configureHost: true
    #Domain name of the ingress controller in the cluster
namespace
    host: dctm-ingress
    clusterDomainName: *ingress_domain
```

```
#To accomodate cluster 1.22
class: nginx
#annotations for the ingress object
#annotations: {}
annotations:
  nginx.ingress.kubernetes.io/proxy-body-size: 50m
```

Creating a namespace for D2

- In k8s, a namespace acts as a sort of logical cluster. We will create one for our D2 deployment.
- Issue the following command:

```
kubectl create namespace d2
```

- Check that your new namespace has been created with:

```
kubectl get namespaces
```

```
PS C:\WINDOWS\system32> kubectl get namespaces
NAME      STATUS   AGE
d2         Active   8d
default    Active   8d
ingress-nginx  Active   7d2h
```

- Note that the provided Helm charts do assume that you have created the “d2” namespace.
- From now on, when you issue kubectl and Helm commands, you’ll have to append the `-n d2` switch, to specify you are working inside the “d2” namespace i.e. :

```
PS C:\WINDOWS\system32> kubectl get services -n d2
NAME      TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
cps       ClusterIP  None          <none>         9680/TCP          3d2h
d2classic ClusterIP  10.96.102.11  <none>         8080/TCP          7d20h
d2config  ClusterIP  10.108.178.138 <none>         8080/TCP          7d20h
d2rest    ClusterIP  10.101.60.246  <none>         8080/TCP          7d20h
```

Deploying the Helm chart

- If you have performed the following prerequisites, you can deploy the provided Helm chart without any modification:

- Have Docker Desktop running with Kubernetes enabled.
- Configured WSL to use 16GB RAM and all vCPUs.
- Installed Helm
- Downloaded software Docker images
- Created a Docker image with D2-Base configuration
- Deployed NGINX
- Created the "d2" namespace.

```
PS C:\Users\jsotomay\OneDrive - OpenText\Documents\k8s\02_23.2\d2> helm install d2
NAME: d2
LAST DEPLOYED: Sat May 6 13:46:34 2023
NAMESPACE: d2
STATUS: deployed
REVISION: 1
TEST SUITE: None
PS C:\Users\jsotomay\OneDrive - OpenText\Documents\k8s\02_23.2\d2>
```

- In an elevated Powershell, cd to the directory called "d2" (it's the directory containing the file "values.yaml").

- Issue the following command (single line):

```
helm install d2 . --values=dockerimages-values.yaml --values=d2-resources-
values-test-small.yaml --namespace d2
```

- This command will deploy a complete Documentum 23.2 stack automatically.

opentext™

OpenText ©2023 All rights reserved 17

`helm install d2 .` : install the chart found in the current directory (.) and deploy it with the name "d2"

`--values=dockerimages-values.yaml --values=d2-resources-values-test-small.yaml` they do provide

information about what docker images to use and sizing information. Note that `values.yaml` is also picked by default. You have more information about these files in the following slides.

`--namespace d2` specifies that elements must be deployed into the d2 namespace.

Monitoring the Helm chart deployment (1) – Tips

- The first time you deploy this chart it can take between one to two hours to deploy, depending on your laptop specs, specially for the content server (but once it's installed, subsequent restarts will take no more than 10 minutes)
- As a rule of thumb, pods are synchronized during deployment, and they patiently wait for their dependencies. i.e. Content Server will wait for the docbroker to become alive, D2 will wait for the content server, etc.
- As soon as you launch the Helm chart, start issuing `kubectl get pods -n d2` commands every 10 seconds or so. You should see their status going into "Running". Most common error you may find is "ImagePullBackOff" meaning that a Docker image can't be found. Double check that you've included all the correct names in `dockerimages-values.yaml`. In theory, if you used all the provided files without changes, no errors of this type are expected.
- If you experience these errors, it's better to cancel the deployment, then correct the issue in the yaml files, and redeploy again until you see all pods going happily into "Running" status.
- To cancel and erase the deployment (everything inside the d2 namespace):
 - `helm uninstall d2`
 - `kubectl delete pvc --all -n d2` (wait a little after this command so all bound pvs are deleted)

Monitoring the Helm chart deployment (2) - Tips

- Even if pods are in “Running” status, they won’t become available until they show “1/1” under the “Ready” column.
- Each pod will periodically test itself until its liveness and readiness probe are positive. At this point it will appear as really available for the rest of the pods via its exposed services.
- Be patient, as some of these probes are tested with several minutes in between, especially during first installation.
- Especially during first installation, you may see pods restarting themselves after several minutes. This is expected behaviour. If their dependencies take time to be ready, they restart hoping for these dependencies to be available next time. Eventually they will.
- See next slide for several commands that you may use to obtain more details during deployment.
- Be aware that, if you use “describe” commands, you will be able to read errors at the end of the output. Note that the problem may have dissapeared even if there’s not a “Problem fixed” message.

Monitoring the Helm chart deployment (3) - Commands

You can use the following commands to monitor how things are going with your deployment.

- `kubectl get pods -n d2`
Provides information about pods' status. You can “get” anything else (services, nodes, ingress...)
- `kubectl logs <pod name> -n d2`
Provides the log for a given pod
- `kubectl describe pod <pod name> -n d2`
Provides detailed information about the pod configuration, detailed readiness and liveness information, etc. You can “describe” anything else (services, nodes, ingress...)
- `kubectl exec --stdin --tty -n d2 <pod name> -- /bin/bash`
Opens a linux shell into a pod so you can browse for additional information. Most of the platform files are located under `/opt/dctm` and `/opt/dctm_docker` typically. From here you can cat the docbase log or any other file of your interest.

With “kubectl get” and “kubectl describe” you can obtain information about any component. For instance:

```
kubectl get services -n d2      : list all deployed services
kubectl get ingress -n d2      : list all deployed ingresses
kubectl get pvc -n d2          : list all persistent volumen claims
```

```
kubectl describe service <service name> -n d2 : describe the service called
<service name>
```

```
kubectl describe ingress <ingress name> -n d2 : describe the ingress called
<ingress name>
```

Monitoring the Helm chart deployment (4) - Commands

- `kubectl logs <pod name> -n d2`
Prints the log of the specified pod. Add a `-f` switch to tail the log.
- `kubectl get deployments -n d2`
Shows all deployments in the specified namespace (d2).
- `kubectl get statefulsets -n d2`
Shows all statefulsets in the specified namespace (d2).

With “kubectl get” and “kubectl describe” you can obtain information about any component. For instance:

<code>kubectl get services -n d2</code>	: list all deployed services
<code>kubectl get ingress -n d2</code>	: list all deployed ingresses
<code>kubectl get pvc -n d2</code>	: list all persistent volumen claims

`kubectl describe service <service name> -n d2` : describe the service called
<service name>

`kubectl describe ingress <ingress name> -n d2` : describe the ingress called
<ingress name>

Checking all pods are launched correctly

```
PS C:\WINDOWS\system32> kubectl get pods -n d2
NAME          READY   STATUS    RESTARTS   AGE
cpe-0         0/1     Init:0/1   0           64s
d2classic-0   0/1     Init:0/1   0           64s
d2config-0    0/1     Init:0/2   0           63s
d2rest-0      0/1     Init:0/1   0           64s
d2smartview-0 0/1     Init:0/1   0           64s
da-74f664dd47-8czdf 0/1     Running   0           64s
db-pg-0       1/1     Running   0           63s
dbr-0         0/1     Running   0           64s
dcs-pg-0      0/1     Init:0/3   0           64s
dctm-rest-59d56fdbc9-rdjdf 0/1     Running   0           64s
indexagent-0  0/1     Running   0           64s
indexserver-0 0/1     Running   0           64s
otdsws-7b9ddb8c4c-t6nf5 0/1     Running   1 (38s ago) 64s
PS C:\WINDOWS\system32>
```

Some pods are running, some initializing, only postgres is ready

```
PS C:\WINDOWS\system32> kubectl get pods -n d2
NAME          READY   STATUS    RESTARTS   AGE
cpe-0         0/1     Running   0           112s
d2classic-0   0/1     Init:0/1   0           112s
d2config-0    0/1     Init:1/2   0           111s
d2rest-0      0/1     Running   0           112s
d2smartview-0 0/1     Running   0           111s
da-74f664dd47-8czdf 0/1     Running   0           111s
db-pg-0       1/1     Running   0           111s
dbr-0         0/1     Running   0           112s
dcs-pg-0      0/1     Running   0           112s
dctm-rest-59d56fdbc9-rdjdf 0/1     Running   0           112s
indexagent-0  0/1     Running   0           112s
indexserver-0 0/1     Running   0           112s
otdsws-7b9ddb8c4c-t6nf5 0/1     Running   2 (60s ago) 112s
```

More pods are reaching the Running state...

```
PS C:\WINDOWS\system32> kubectl get pods -n d2
NAME          READY   STATUS    RESTARTS   AGE
cpe-0         0/1     Running   0           2m26s
d2classic-0   0/1     Running   0           2m26s
d2config-0    0/1     Running   0           2m25s
d2rest-0      0/1     Running   0           2m26s
d2smartview-0 0/1     Running   0           2m26s
da-74f664dd47-8czdf 0/1     Running   0           2m26s
db-pg-0       1/1     Running   0           2m25s
dbr-0         0/1     Running   0           2m26s
dcs-pg-0      0/1     Running   0           2m26s
dctm-rest-59d56fdbc9-rdjdf 0/1     Running   0           2m26s
indexagent-0  0/1     Running   0           2m26s
indexserver-0 0/1     Running   0           2m26s
otdsws-7b9ddb8c4c-t6nf5 1/1     Running   2 (94s ago) 2m26s
```

Finally everyone is up and running.

```
PS C:\WINDOWS\system32> kubectl get pods -n d2
NAME          READY   STATUS    RESTARTS   AGE
cpe-0         1/1     Running   1 (64m ago) 71m
d2classic-0   0/1     Running   5 (4409s ago) 71m
d2config-0    0/1     Running   8 (5m5s ago) 71m
d2rest-0      0/1     Running   8 (6m22s ago) 71m
d2smartview-0 0/1     Running   8 (6m12s ago) 71m
da-74f664dd47-8czdf 1/1     Running   0           71m
db-pg-0       1/1     Running   0           71m
dbr-0         1/1     Running   0           71m
dcs-pg-0      0/1     Running   0           71m
dctm-rest-59d56fdbc9-rdjdf 0/1     Running   13 (58s ago) 71m
indexagent-0  0/1     Running   12 (86s ago) 71m
indexserver-0 1/1     Running   0           71m
otdsws-7b9ddb8c4c-t6nf5 1/1     Running   2 (70m ago) 71m
```

Until the docbase initialization process is complete, CS (dcs-pg-0) won't be ready and most of the pods will restart from time to time, until CS is ready. This may take more than 90 minutes to finish.

```
PS C:\Users\jsotomay\OneDrive - openText\Documents\k8s\02_23_21\d2> k
NAME          READY   STATUS    RESTARTS   AGE
cpe-0         1/1     Running   1 (7h23m ago) 7h30m
d2classic-0   1/1     Running   5 (6h49m ago) 7h30m
d2config-0    1/1     Running   5 (6h49m ago) 7h30m
d2rest-0      1/1     Running   5 (6h49m ago) 7h30m
d2smartview-0 1/1     Running   5 (6h49m ago) 7h30m
da-74f664dd47-trgj2 1/1     Running   0           7h30m
db-pg-0       1/1     Running   0           7h30m
dbr-0         1/1     Running   0           7h30m
dcs-pg-0      1/1     Running   1 (6h56m ago) 7h30m
dctm-rest-59d56fdbc9-448p7 1/1     Running   8 (6h50m ago) 7h30m
indexagent-0  1/1     Running   1 (7h18m ago) 7h30m
indexserver-0 1/1     Running   0           7h30m
otdsws-7b9ddb8c4c-9dbfm 1/1     Running   1 (7h30m ago) 7h30m
```

When all pods are 1/1 Ready and in running status, the deployment has been completed and we can access our apps.

* These screenshots have been taken in different sessions.

Inspecting correct docbroker initialization

```
PS C:\WINDOWS\system32> kubectll logs dbr-0 -n d2
sat May 6 11:47:02 AM UTC 2023 In cs_startup.sh
sat May 6 11:47:02 AM UTC 2023 Install owner is already dmadin
sat May 6 11:47:02 AM UTC 2023 Documentum configuration/startup with dmadin
sat May 6 11:47:02 AM UTC 2023 -----
opt/dctm/docker/kube.cs.sh: line 2973: $logfile: ambiguous redirect
sat May 6 11:47:02 AM UTC 2023 initialize
opt/dctm/docker/kube.cs.sh: line 2973: $logfile: ambiguous redirect
sat May 6 11:47:02 AM UTC 2023 -----
sat May 6 11:47:02 AM UTC 2023 Setting mounted folders owner as install owner
sat May 6 11:47:03 AM UTC 2023 -----
sat May 6 11:47:03 AM UTC 2023 handle_password_change
changing password for user dmadin.
passwd: all authentication tokens updated successfully.
changing password for user root.
passwd: all authentication tokens updated successfully.
sat May 6 11:47:03 AM UTC 2023 -----
sat May 6 11:47:04 AM UTC 2023 handle_start_docbroker
sat May 6 11:47:04 AM UTC 2023 Docbroker Installation Started....
Picked up JAVA_TOOL_OPTIONS: -Djava.locale.providers=COMPAT,SPI --add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.lang.invoke=ALL-UNNAMED --add-exports=java.base/sun.security.provider=ALL-UNNAMED --add-exports=java.b
sat May 6 11:48:39 AM UTC 2023 Docbroker Installation Completed
sat May 6 11:48:39 AM UTC 2023 -----
sat May 6 11:48:39 AM UTC 2023 handle_start_ext_docbroker
sat May 6 11:48:39 AM UTC 2023 Docbroker Installation Started....
Picked up JAVA_TOOL_OPTIONS: -Djava.locale.providers=COMPAT,SPI --add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.lang.invoke=ALL-UNNAMED --add-exports=java.base/sun.security.provider=ALL-UNNAMED --add-exports=java.b
sat May 6 11:50:03 AM UTC 2023 External Docbroker Installation Completed
sat May 6 11:50:03 AM UTC 2023 -----
sat May 6 11:50:03 AM UTC 2023 prepare_readlines
sat May 6 11:50:03 AM UTC 2023 Docbroker-Installed file created
sat May 6 11:50:03 AM UTC 2023 -----
sat May 6 11:50:03 AM UTC 2023 handle_logs_cleanup
sat May 6 11:50:03 AM UTC 2023 Documentum Content Server Installation Completed
sat May 6 11:50:03 AM UTC 2023 -----
sat May 6 11:50:03 AM UTC 2023 cs_startup.sh completed successfully
PS C:\WINDOWS\system32>
```

dbr-0 1/1 Running 0 8m43s

Inspecting correct Content Server Initialization

[illegible]

Inspecting docbase initialization

CS pod will show as “Ready 0/1” for a while because the docbase installation keeps going.
This is where most of the time goes for the deployment (+1 hour).

```
% C:\WINDOWS\system32\kubectl exec --stdin --tty -n d2 dcs-pg-0 -- /bin/bash
%faulted container "dcs-pg-0" out of: dcs-pg, d2installerinit (init), cc-dar-installer (init), d2customdarinit (init)
[dmadmin@dcs-pg-0 /]$ cd /opt/dcm/product/23.2/install/logs
[dmadmin@dcs-pg-0 logs]$ cat install.log
11:54:34,510 INFO [main] com.documentum.install.shared.installanywhere.actions.InitializeSharedLibrary - Log is ready and is set to the level - INFO
11:54:34,512 INFO [main] com.documentum.install.shared.installanywhere.actions.InitializeSharedLibrary - The product name is: UniversalServerConfigurator
11:54:34,512 INFO [main] com.documentum.install.shared.installanywhere.actions.InitializeSharedLibrary - The product version is: 23.2.0000.0165
11:54:34,513 INFO [main] -
11:54:34,533 INFO [main] com.documentum.install.shared.installanywhere.actions.InitializeSharedLibrary - Done InitializeSharedLibrary ...
11:54:34,576 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerInformation - Setting CONFIGURE DOCBROKER value to TRUE for SERVER
11:54:34,576 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerInformation - Setting CONFIGURE DOCBASE value to TRUE for SERVER
11:54:35,580 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerCheckEnvironmentVariable - The installer was started using the da_launch_server.config_program.sh script.
11:54:35,580 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerCheckEnvironmentVariable - The installer will determine the value of environment variable DOCUMENTUM.
11:54:35,580 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerCheckEnvironmentVariable - existingVersion : 23.2servermajorversion : 23.2
11:54:38,580 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerCheckEnvironmentVariable - The installer will determine the value of environment variable PATH.
11:54:38,580 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerCheckEnvironmentVariable - existingVersion : 23.2servermajorversion : 23.2

...

11:59:34,520 INFO [main] - Waiting for repository docbase1.docbase1 to start up.
11:59:35,306 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerDataIniGenerator - The installer will create data_dictionary.ini for the repository.
11:59:35,312 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerAddDocbaseEntryToBPMN - BPM webapp does not exist.
11:59:35,399 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerVerifyDataIni - The installer will verify data_dictionary.ini for the repository.
11:59:35,420 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerProcessingScripts - The installer will execute the : Repository HeadStart script.
11:59:35,436 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerProcessingScripts - The installer will execute the : Updating format objects script.
12:04:44,487 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerProcessingScripts - The installer will execute the : Creating Electronic Signature Objects script.
12:05:00,380 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerProcessingScripts - The installer will execute the : Creating CSEC Plugin Object script.
12:07:23,181 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerProcessingScripts - The installer will execute the : Configuring templates script.
12:08:17,146 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerProcessingScripts - The installer will execute the : Replication configuration script.

12:08:17,146 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerProcessingScripts - The installer will execute the : Replication configuration script.
12:13:24,162 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerProcessingScripts - The installer will execute the Documentum Desktop Client script.
12:16:42,080 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerProcessingScripts - The installer will execute the OFC Support script.

12:20:17,602 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerProcessingScripts - The installer will execute the OFC JavaDocbase implementation script.
12:28:47,164 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerProcessingScripts - The installer will execute the Installs BPM Modules script.
12:21:02,783 INFO [main] com.documentum.install.server.installanywhere.actions.DIMAServerProcessingScripts - The installer will execute the Creates method object for CTS script.
```

Accessing the applications

Updating hosts file

- To simulate a proper domain name without a DNS, we will include our domain in the hosts file.
- Open values.yaml and find the line:

```
23 ingressUrl: &ingressUrl dcm-ingress.d2.jm.net/
```

- Copy this value (dcm-ingress.d2.jm.net)
- Edit C:\Windows\System32\drivers\etc\hosts and add an entry with your current ip and the domain name you've copied:

```
# localhost name resolution is handled within DNS itself.
# 127.0.0.1 localhost
# ::1 localhost
10.179.144.31 dcm-ingress.d2.jm.net
# Added by Docker Desktop
10.179.144.31 host.docker.internal
10.179.144.31 gateway.docker.internal
# To allow the same kube context to work on the host and the container:
127.0.0.1 kubernetes.docker.internal
# End of section
```

- You must update the file again everytime your ip changes.

Application endpoints

- Applications and services are exposed to the world outside the cluster via an ingress configuration. Issue this command to know how are they exposed:

```
kubectl describe ingress dctm-ingress -n d2
```

```
Rules:
Host      Path  Backends
----
dctm-ingress.d2.jm.net
  /DmMethods/servlet/DoMethod  dcs-pg-jms-service:9080 (10.1.1.124:9080)
  /ACS/servlet/ACS             dcs-pg-jms-service:9080 (10.1.1.124:9080)
  /thumbsrv/getThumbnail       dcs-pg-tns-service:8081 (10.1.1.124:8081)
  /D2                          d2classic:8080 (10.1.1.126:8080)
  /D2-Config                   d2config:8080 (10.1.1.122:8080)
  /da                          da-svc:8080 (10.1.1.113:8080)
  /dsearchadmin                indexserver:9300 (10.1.1.125:9300)
  /IndexAgent                  indexagent:9200 (10.1.1.121:9200)
  /otds-admin/                 otds:80 (10.1.1.114:8080)
```

- For instance, if you want to access Documentum Administrator, you just have to point your browser to: <https://dctm-ingress.d2.jm.net/da>

Accessing our applications

- Browse to the application endpoint i.e. <https://dctm-ingress.d2.jm.net/da/>
- You will get a warning about the security certificate not being valid. Click on "Advanced" and proceed to site anyway. This is due to the certificate being self signed (normal for demo envs).



Your connection isn't private

Attackers might be trying to steal your information from **dctm-ingress.d2.jm.net** (for example, passwords, messages, or credit cards).

NET::ERR_CERT_AUTHORITY_INVALID

Hide advanced

Go back

This server couldn't prove that it's **dctm-ingress.d2.jm.net**; its security certificate is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection.

Continue to **dctm-ingress.d2.jm.net** (unsafe)



Documentum Administrator

https://dctm-ingress.d2.jm.net/da/component/main

Please Install Webtop Content Transfer Extension

System Overview

User: dmadm@dmadm

Report Date: Thu May 04 15:07:31 UTC 2023

Docbase Name: docbase1

Docbase Version: 21.1.0000.0165 Linux64/Posix

Projected Doclinkers: db-d2-inc,cluster-local:1489

State of the Docbase:

Administration - Server Information

Content Storage Services: DISABLED

Trusted Node: DISABLED

List Of Platforms Exhibited:

BOCS Servers: 0

ACS Servers: 1

Basic Configuration - Product Licensing

Collaborative Edition: ENABLED

Physical Records Management: DISABLED

Retention Policy Services: DISABLED

Administration - Distributed Content Configuration

BOCS Servers Configurations: NOT INSTALLED

Administration - Storage

STORAGE Snapshot Connector: DISABLED

Administration - Content Delivery

ES Configurations: 0

Administration - Content Transformation Services

Listing File Store Details

Total Number of File Stores: 1

File Store Name	Media Type	Last Stored Time	Store Type	Active File Count	Orphaned File Count
Repository_01	REGULAR CONTENT	5/4/2023 1:05:17 PM	1	2284	0
Overhead_Store_01	Overheading CONTENT	NA	1	0	0
Streaming_Store_01	STREAMING CONTENT	NA	1	0	0
Repository_Pending_Store	REGULAR CONTENT	NA	1	0	0
Repository_Retention_01	REGULAR CONTENT	NA	1	0	0

Summary of Users

Total Number of Users	Active Users	De-active Users
75	75	0

opentext™

OpenText ©2023 All rights reserved 29

Manual steps – OTDS Configuration

- DA will work with a simple inline user `dadmin:password`. But to Access D2, Smartview, etc, you will need to configure OTDS.
- OTDS must be configured following the steps described in page 24 of “OpenText Documentum D2 CE 23.2 - Cloud Deployment Guide”
(https://support.opentext.com/csm?id=kb_article_view&sys_kb_id=9030388947de6510f3f9da7a436d431f)
- The URL suggested in step 3c is valid unless you’ve modified service names in `values.yaml`:
<http://dcs-pg-jms-service:9080/dmotsrest>
- Step 6, create an OAuth client, use “`d2_oauth_client`” as its name, unless you modified it in `values.yaml`
- Step 6 a, you can find your ingress URL around line 23 of `values.yaml`, or by issuing a `kubectl describe ingress dctm-ingress -n d2` command. If you have not modified the provided charts, it will be: <https://dctm-ingress.d2.jm.net>

- These steps could also be automated in the Helm charts, but I did not have the time to try it out yet.
- These steps must be done only once, after first installation is complete.

Modifying the configuration

Overview of the changes made to the original Helm charts

Configuring Docker images

- In `dockerimages-values.yaml`, provide the Docker repository name

```
1 repository : &docker_repo registry.opentext.com
```

- The following variables may be left as default:

```
2 image: &graylog_image registry.opentext.com/graylog-sidecar:v1.8
3 appserverImageTag: &appserverImageTag '23.2'
4 appserverImageName: &appserverImageName dctm-tomcat
5 pullPolicyType: &pull_policy_type IfNotPresent
6 pullSecretName: &pull_secret_name []
```

- For each component, ensure the Docker image name and tag matches the one you've downloaded:

```
190 da:
191   images:
192     da:
193       repository: *docker_repo
194       name: dctm-admin
195       tag: 23.2
196       pullPolicy: *pull_policy_type
197       imagePullSecrets: *pull_secret_name
```

opentext™

OpenText ©2023 All rights reserved 32

A complete list of the image names and tags can be obtained from the Cloud Deployment Guide.

For those components you won't be using (i.e. Graylog, fluentd, etc) you can leave them as they are. If the component is disabled, the image won't be used.

I've commented some sections as, for instance, I'm not going to use Process Engine:

```
# JM COMMENTED
#- name: "peinstaller-init"
# image: "registry.opentext.com/dctm-xcp-
installer:23.2"
# imagePullPolicy: *pull_policy_type
# command: ['/bin/sh', '-c', 'yes |sudo cp -Rf
/pescripts/* /opt/dctm_docker/customscriptpvc/']
# volumeMounts:
# - name: dcs-data-pvc
#   mountPath: /opt/dctm_docker/customscriptpvc
#   subPath: initcontainercustomscripts/dcs-pg
```

Also, you're responsible to add some extra init containers with the images you've

created with D2 configuration DAR and ZIP files, i.e. under d2config:

```
- name: init
  image: d2customdar:latest
  imagePullPolicy: *pull_policy_type
  command: ['/bin/sh', '-c', 'yes | cp -rf /opt/D2-
install/custom/* /customdir/']
  volumeMounts:
    - name: customconfig
      mountPath: /customdir
```

Modifying variables to fit your environment

Deployment is configured in the file values.yaml. You don't have to configure everything. Just some global values and the elements you want to deploy.

- rwoStorage & rwmStorage: change it from trident-nfs to hostpath
- Find and replace all occurrences of <namespace> by your namespace (i.e. d2)
- Find and replace all occurrences of <docbase_name> by your docbase (i.e. docbase1)

```
3- rwoStorage: &rwo_storage_class trident-nfs
4- rwmStorage: &rwm_storage_class trident-nfs
5- env: &env_domain <namespace>.svc.cluster.local
6- namespace: &namespace <namespace>
7- docbase: &docbase_name <docbase_name>
```

```
→ 3+ rwoStorage: &rwo_storage_class hostpath
4+ rwmStorage: &rwm_storage_class hostpath
5+ env: &env_domain d2.svc.cluster.local
6+ namespace: &namespace d2
7+ docbase: &docbase_name docbase1
```

- Find and replace all occurrences of cfcrlab.bp-paas.otxlab.net by your own domain (i.e. jm.net)

```
22 # Provide the Ingress url used for D2,D2-Smartview and d2-rest
23- ingressUrl: &ingressUrl d2m-ingress.d2.cfcrlab.bp-paas.otxlab.net/
```

```
→ 22 # Provide the Ingress url used for D2,D2-Smartview and d2-rest
23+ ingressUrl: &ingressUrl d2m-ingress.d2.jm.net/
```

- You will find that these replacements will indirectly configure several values across the file. (i.e)

```
1363 #the ods service should be in the format https://<OTDS server url>:<port>/otdsws
1364- url: https://d2m-ingress.d2.cfcrlab.bp-paas.otxlab.net/otdsws
```

```
→ 1363 #the ods service should be in the format https://<OTDS server url>:<port>/otdsws
1364+ url: https://d2m-ingress.d2.jm.net/otdsws
```

opentext™

OpenText ©2023 All rights reserved 33

values.yaml is not shipping empty by default, but contains most of the values engineering uses to deploy and test the charts.

It's a matter of time and practice to develop your eye to identify the values you must change, the ones that you can leave as they are, the ones that you don't need etc.

In my experience, once you have managed to configure and deploy a Helm chart once and become familiar with the key values, the next one will be significantly easier. You can just compare your latest working values.yaml file (i.e 23.2) with the new one you have downloaded (i.e. 23.4) and transpose the correct values to the new one. Always keep an eye on new sections or configurations that may have been added.

A good exercise you could do is to compare my provided values.yaml with the original one (see specific slide for comparing files). That will provide you with an idea of the changes I've made to values.yaml to fit my needs.

Also, there are no shortcuts here: we must know Documentum and learn Kubernetes as much as possible.

- trident-nfs is the storage class used by engineering in their own environments. With Docker Desktop we only have hostpath available by default. All persistent information will be stored in our laptop filesystem. This is not a good practice if we had more than one node in our cluster, but for a single node demo environment

is good enough.

- cfcrlab.bp-paas.otxlab.net is the domain name used by engineering in their own environments. We can choose to have any domain we want. By including this domain in our hosts file in Windows, we can create the illusion of using a real domain name exposed in the Internet.

Enabling or disabling features

- Deploying Documentum features is just a matter of activating (true) or disabling (false) them, and in some cases providing some additional configuration parameters.
- Some of them are enabled at variable level. Let's say you don't want to use Graylog, turn it to false:

```
33 #Gray log server details
34- graylogEnable: &graylog_enable true      → 34+ graylogEnable: &graylog_enable false
```

- Some others are enabled at the beginning of their own section. There's always a "enabled" attribute that you can set to true or false. Let's say you want to enable Documentum REST services. Just turn it to true:

```
1264- dcm-rest:
1265-   enabled: false
1266-   serviceName: dcm-rest
1267-   namespace: "namespace"
1268-   customLabels:
1269-     app: dcm-rest
1270-     containerName: rest-container

1264+ dcm-rest:
1265+   enabled: true
1266+   serviceName: dcm-rest
1267+   namespace: "namespace"
1268+   customLabels:
1269+     app: dcm-rest
1270+     containerName: rest-container
```

- The provided sample Helm chart deploys postgres, content server, docbroker, administrator, REST, xplore, otds, d2 classic, smartview and config.

If instead of using the provided Helm chart you want to learn to configure it yourself, I highly recommend to start it simple: postgres, docbroker, server and da. Also disable graylog, Kafka and OTDS components. Once you have this process mastered and you are able to log in to Documentum Administrator, you can start enabling other features one at a time.

If you think you've broken your environment, don't hesitate to reset your Kubernetes cluster using the option in Docker Desktop settings and start again. The less components you try to deploy until you get the hang of it, the faster you will progress.

Setting up resources

- In the original Helm charts, there are several files named as “d2-resources-values-[sample_description].yaml”.
- These files basically describe:
 - Sizes of volume claims
 - CPU requests and limits
 - Memory requests and limits
 - Starting number of Pods per element (i.e. 2x Content Server)
- Chances are that even the smallest of the environments (test_small) are too much for your laptop.
- In the provided “d2-resources-values-test-small.yaml” we have stripped out the requests and limits sections, and downsized all deployments to just one pod each.
- By doing this, k8s will use CPU and memory as specified in the original charts. This will be enough for most demo / learning use cases.

```
1 dcm-server:
2   db:
3     database:
4       replicaCount: 1
5     alterParam:
6       max_connections: 1000
7       shared_buffers: 2GB
8     persistentVolume:
9       size: 100Gi
10  docbroker:
11    docbroker:
12      replicaCount: 1
13    persistentVolume:
14      size: 1Gi
15  content-server:
16    contentserver:
17      replicaCount: 1
18    docbrokersCount: 1
```

This approach is acceptable for POC/Demo purposes, not for production. But in a laptop we have limited resources (my laptop uses 11GB just to show the Desktop). By default, our Helm charts deploy the components as deployments or statefulsets with at least 2 pods per service, which is great to provide HA out of the box. But in our laptop we don't have enough memory to have a minimum of “two of everything”, so this is why we have downsized all components to just one pod.

Setting up resources – downscaling and upscaling

- If you don't want a given component to be deployed, just set `enabled=false` in `values.yaml`.
- For temporary adjustments, you can downscale or upscale the number of pods.
- Documentum components are deployed in the form of statefulsets or deployments:

- You can downscale them to 0 if you want to temporarily “shut down” a component and save memory / cpu: i.e:

```
PS C:\> kubectl scale deployment dctm-rest --replicas 0 -n d2
deployment.apps/dctm-rest scaled
```

```
PS C:\> kubectl get deployments -n d2
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
da	1/1	1	1	3d22h
dctm-rest	0/0	0	0	3d22h
otdsws	1/1	1	1	3d22h

- Conversely, you can scale to 1 again:

```
PS C:\> kubectl scale deployment dctm-rest --replicas 1 -n d2
deployment.apps/dctm-rest scaled
```

```
PS C:\> kubectl get deployments -n d2
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
da	1/1	1	1	3d22h
dctm-rest	1/1	1	1	3d22h
otdsws	1/1	1	1	3d22h

opentext™

```
PS C:\Users\jsotomay\OneDrive - OpenText\Documents\k8s> kubectl get statefulsets -n d2
```

NAME	READY	AGE
cps	1/1	3d22h
d2classic	1/1	3d22h
d2config	1/1	3d22h
d2rest	1/1	3d22h
d2smartview	1/1	3d22h
db-pg	1/1	3d22h
dbr	1/1	3d22h
dcs-pg	1/1	3d22h
indexagent	1/1	3d22h
indexserver	1/1	3d22h
kfk	1/1	3d22h
zkpr	1/1	3d22h

```
PS C:\Users\jsotomay\OneDrive - OpenText\Documents\k8s> kubectl get deployments -n d2
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
da	1/1	1	1	3d22h
dctm-rest	1/1	1	1	3d22h
otdsws	1/1	1	1	3d22h

Comparing documents with VS Code

- Right click on any file and select “Select for Compare”
- Right click on a different file and select “Compare with Selected”.
- This is a very powerful feature for upgrades, as we can quickly transpose our values from our current chart to a new version.

```

1 ## Update the anchor tags below and any other values can be customized/updated using the
2
3+ rwfstorage: &rwfstorage.class trident-nfs
4- rwfstorage: &rwfstorage.class trident-nfs
5+ env: &env.domain &env.namespace.svc.cluster.local
6- env: &env.domain &env.namespace.svc.cluster.local
7+ dochase: &dochase.name &dochase.name
8- c5secrets: &c5secrets.name c5-secret-config
9- installOwnerPassword: &installOwner.username default
10- installOwnerPassword: &installOwner.password password
11- globalRegistryPassword: &global_registry_password Password@1234567890
12- otdsEnabled: &otds.enabled true
13- otdsAdminPassword: &otds.admin.password otds
14- otdsUsername: &otds.db.username postgres
15- otdsUserPassword: &otds.db.password password
16- d2ClassicWebappName: &d2ClassicWebappName D2
17- d2ConfigWebappName: &d2ConfigWebappName D2-Config
18- d2RestWebappName: &d2RestWebappName D2-rest
19- d2SmartViewWebappName: &d2SmartViewWebappName D2-SmartView
20- tomcatbase_usecommonpcv: &tomcatbase_usecommonpcv true
21- tomcatbase_commpcName: &tomcatbase_commpcName D2-extension-pvc
22- # Provide the ingress url used for D2, D2-SmartView and D2-rest
23- ingressUrl: &ingressUrl d2c-ingress.d2.cfr-lab.hp-paa-otclab.net/
24- askLocation: &ask_location local
25- askUrl: &ask_url https://askurl:port
26- # Only need to update the below three anchor values if otds is enabled above
27- oauthClient: &otds.oauth_client d2_oauth_client
28- otdsAuthSvcProtocol: &otds_auth_svc_protocol https
29- otdsAuthSvc: &otds_auth_svc.d2c-ingress.d2.cfr-lab.hp-paa-otclab.net/otdus
30- ingressDomain: &ingress_domain d2.cfr-lab.hp-paa-otclab.net
31- ingressProtocol: &ingress_protocol https
32- subdomain: &subdomain &subdomain.service.type ClusterIP
  
```

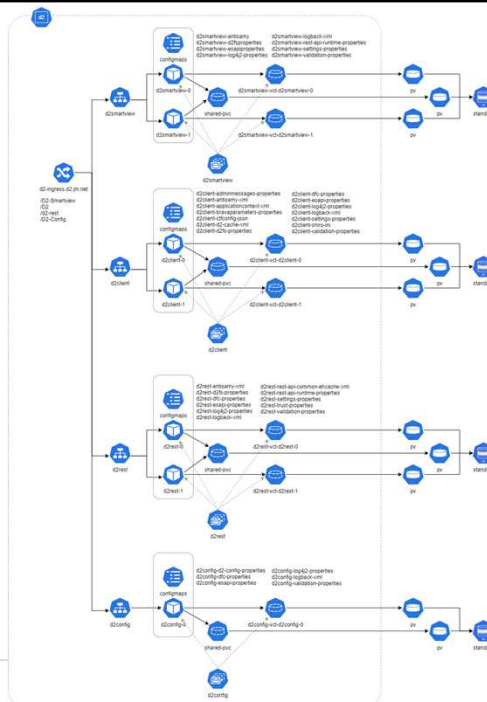
Original Helm chart VS modified Helm chart (as provided for this tutorial)

WIP Slides

Following procedures have not been fully tested yet,
but you may find them helpful.

opentext™

Text ©2023 All rights reserved 39



Deploy Kubernetes Dashboard

- Deploy application by issuing (single line):

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.
7.0/aio/deploy/recommended.yaml
```

- Create a file named dashboard-admin.yaml with the contents shown on the right side.

- Create the user issuing:

```
kubectl apply -f .\dashboard-admin.yaml
```

- Create a token issuing:

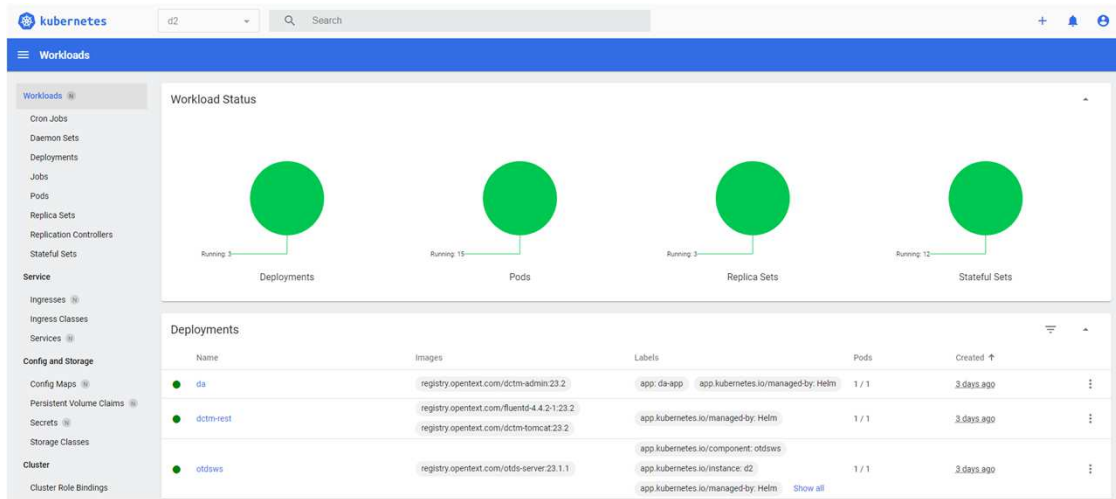
```
kubectl -n kubernetes-dashboard create token admin-user
```

- Issue `kubectl proxy` and navigate to the URL shown below. Paste the token from previous command.

<http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/>

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
```

Kubernetes Dashboard showing “d2” namespace



Enable metrics server

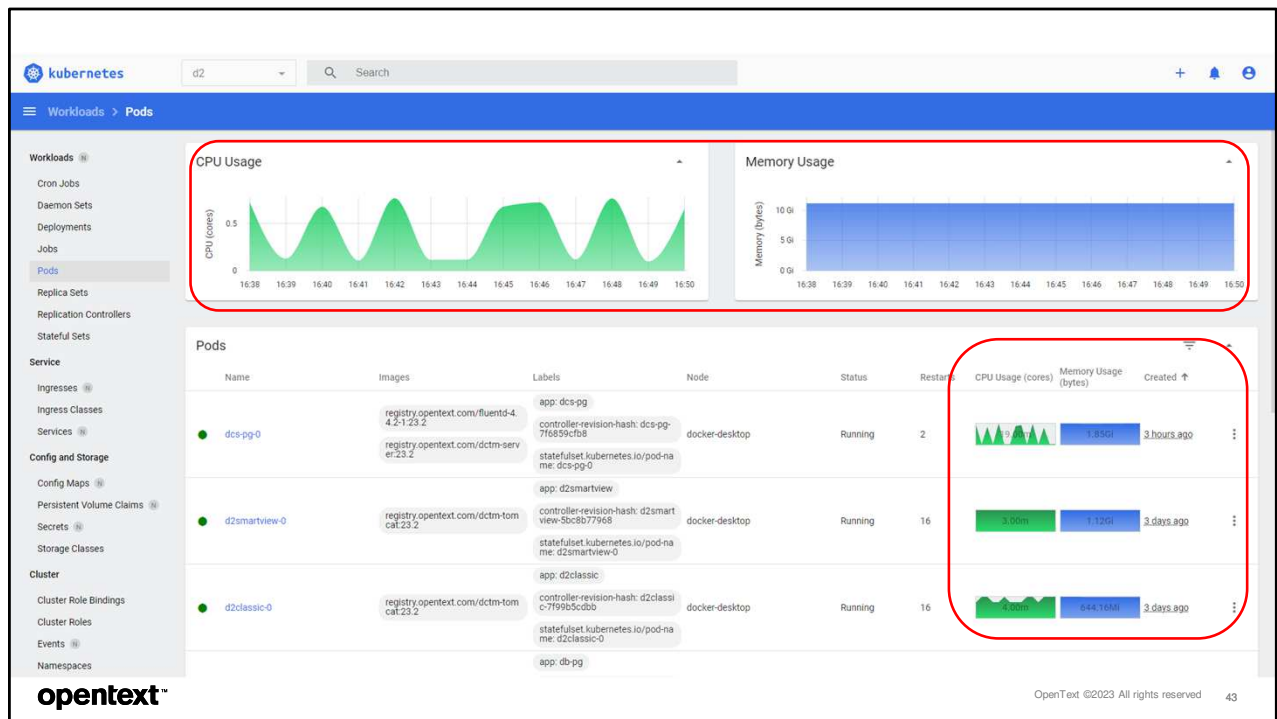
Workloads > Deployments > metrics-server

- Metrics server allows to have detailed graphical information of CPU and Memory usage.
- Enable metrics server by issuing:

`kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml`




- It won't launch correctly because of the TLS certs. You must patch the metrics server deployment. To do so:
- In Kubernetes Dashboard, select "Deployments". In the top namespace selector, select "All". In the list, click "metrics-server".
- Edit the deployment by clicking the pencil icon in the right of the blue header.
- Edit the highlighted line and click "Update"

```
176 - name: tmp-dir
177   emptyDir: {}
178 containers:
179   - name: metrics-server
180     image: registry.k8s.io/metrics-server/metrics-server:v0.6.3
181     args:
182       - '--cert-dir=/tmp'
183       - '--secure-port=4443'
184       - '--kubelet-insecure-tls'
185       - '--kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname'
186       - '--kubelet-use-node-status-port'
187       - '--metric-resolution=15s'
188     ports:
189       - name: https
```



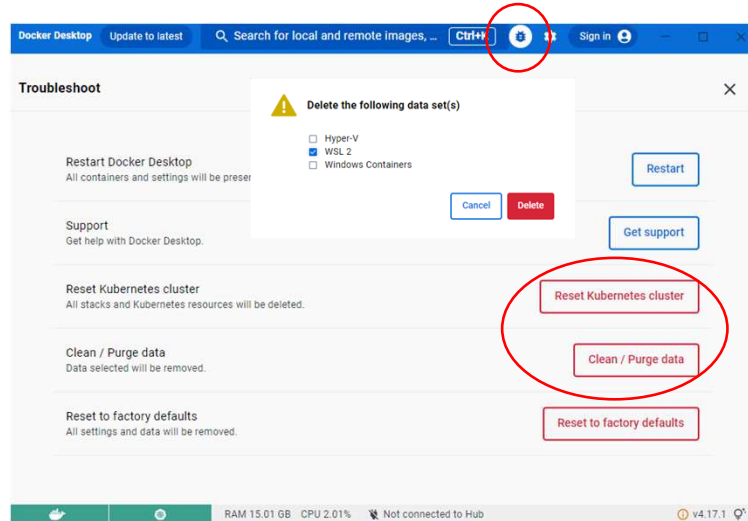
Enable metrics server (2)

- The information shown by the metrics server is very useful to know real memory and CPU consumption.
- You can adjust the “resources” section of your yaml files to adapt to your environment, as a small demo environment with one user (hey, that’s you!) will consume even less than the smallest of the environments included with the charts.

● dcs-pg-0	registry.opentext.com/fluentsd-4.2.1:23.2 registry.opentext.com/dctm-server:23.2	app: dcs-pg controller-revision-hash: dcs-pg-7f6899cfb8 statefulset.kubernetes.io/pod-name: dcs-pg-0	docker-desktop	Running	2	 3.9%	1.95Gi	3 hours ago	⋮
● d2smartview-0	registry.opentext.com/dctm-tomcat:23.2	app: d2smartview controller-revision-hash: d2smartview-5bc8b77968 statefulset.kubernetes.io/pod-name: d2smartview-0	docker-desktop	Running	16	 3.00m	1.12Gi	3 days ago	⋮
● d2classic-0	registry.opentext.com/dctm-tomcat:23.2	app: d2classic controller-revision-hash: d2classic-7f99b5cddb statefulset.kubernetes.io/pod-name: d2classic-0	docker-desktop	Running	16	 4.00m	644.16Mi	3 days ago	⋮

Resetting the cluster

- If you want to start fresh, you can use “Reset Kubernetes cluster”. This will erase all deployments but will keep your Docker images.
- Sometimes you may need to delete all data in the WSL VM to retrieve space and performance. Using “Clean/Purge Data” will do it. **You will loose your downloaded images. See next slide before doing this.**



Keeping your Docker images

- Docker images are downloaded (pulled) to your local Docker registry. It takes a long time to complete. This registry, the images and its contents will disappear if you purge Docker's data.
- For testing purposes sometimes purging this data is necessary (several install / uninstall operations of the Helm chart may have a severe impact on performance when using hostpath storageclass).
- Once you have downloaded the images at least once and they are in your Docker registry, you can save them as TAR files by issuing this command:

```
docker save --output dctm-rest_23.2.tar registry.opentext.com/dctm-rest:23.2
```

- Conversely, you can load the images from the TAR files way quicker than downloading them again:

```
docker load --input dctm-rest_23.2.tar
```

- "save_images.cmd" and "load_images.cmd" scripts are provided for your convenience.
- Execute "save_images" from a directory not synchronized with Core Share or Onedrive.
- By using this procedure you can re-push the images to your Docker registry in about 20 minutes instead of 2 hours

Proper way to do this is by keeping our own Docker image registry, but for testing purposes this is cumbersome and requires more administration.

Deploying Eventhub (1)

- Pull required images:

```
docker pull registry.opentext.com/fluentsd-4.4.2-1:23.2
```

```
docker pull registry.opentext.com/kafka-2.13-3.4.0:23.2
```

- Add them to dockerimages-values.yaml (already correct in 23.2 helm chart)
- Enable kafka & fluentsd in values.yaml

```
57 # attributes for kafka deployment
58 kafka: &kafka_enabled true
59 kafka_replicas: &kafka_replica_count 1
60 zookeeper_port: &zookeeper_port 2181
61 zookeeper_connect: &zookeeper_connect_tag d2.jm.net:2181
62 kafka_admin_user_name: &kafka_admin_username kafka-user
63 kafka_admin_user_password: &kafka_admin_password kafka-password
64 kafka_topic_name: &kafka_topic Cs-Audit-Topic
65 kafka_port: &kafka_broker_port 9092
66
67 ### Global variables for fluentsd side car ###
68 fluentsd: &fluentsd_enabled true
69 fluentsdTcpPort: &fluentsd_tcp_port 24224
70 fluentsdRestPort: &fluentsd_rest_port 63000
71 #fluentsdRestPort: &fluentsd_rest_port 88886
72 fluentsdUdpPort: &fluentsd_udp_port 20001
73 eventLogLevel: &event_log_level 3
74 dfcRPCTracing: &dfc_rpc_log_enable true
75
```

Deploying Eventhub (2)

- In 23.2 Helm chart Kafka deployment is configured to use trident-nfs storageclass. Add storageclass information to values.yaml so Kafka is deployed using hostpah storage.

```
2041 kafka:
2042   ports:
2043     kafka: 9092
2044
2045   livenessProbe:
2046     enable: true
2047
2048   topic:
2049     create: true
2050     topicname: *kafka_topic
2051     partitions: 3
2052
2053   consumer:
2054     username: kafka-consumer
2055     password: kafka-consumer
2056
2057   persistentVolume:
2058     servicedataPVCName: kfk-data-pvc
2059     namespace: *namespace
2060     storageClass: *rwo_storage_class
2061
2062   volumeClaimTemplate:
2063     vctname: kfk-vct-data
2064     storageClass: *rwo_storage_class
```

```
2007 zookeeper:
2008   ports:
2009     clientPort: *zookeeper_port
2010     serverPort: 2888
2011     electionPort: 3888
2012
2013   persistentVolume:
2014     servicedataPVCName: zkpr-data-pvc
2015     namespace: *namespace
2016     storageClass: *rwo_storage_class
2017
2018   volumeClaimTemplate:
2019     vctname: zkpr-vct-data
2020     storageClass: *rwo_storage_class
```

When eventhub is deployed, fluentd is added as a container to dcs-pg pod, dcm-rest, etc. To ssh into these pods now, you need to specify the container you want to connect to, i.e:

```
kubect1 exec --stdin --container dcs-pg --tty -n d2 dcs-pg-0 -- /bin/bash
```

Otherwise you will connect to fluentd's container by default.

Deploying a browser for Kafka data (1)

- Download Helm Chart from: <https://github.com/obsidiandynamics/kafdrop>
- Encode the following kafka properties as base 64 (<https://www.base64encode.org/>):

```
security.protocol=SASL_PLAINTEXT
sasl.mechanism=SCRAM-SHA-512
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule required username="kafka-user"
password="kafka-password";
```

- This string gets encoded as:

```
c2VjdXJpdHkucHJvdG9jb2w9U0FTTF9QTEFJTlRFWFQKc2FzbC5tZWNoYW5pc209U0NSQU0tU0hBLTUxMgpzYXNsLmphYXMuY29uZmlnPW9yZy5hcGFjaGUua2Fma
2EuY29tbW9uLnNlY3VyaxR5LnNjcmFtLlNjcmFtTG9naW5Nb2R1bGUgcmlwZmlyZWQgdXNlcm5hbWU9ImthZmthLXVzZXIiIHBhc3N3b3JkPSJrYWZrYS1wYXNzd2
9yZCI7
```

- Include this encoded string as value for "properties" in values.yaml.
- Include
kfk-0.kfk.d2.svc.cluster.local:9092
as brokerConnect



```
1 replicaCount: 1
2
3 image:
4   repository: obsidiandynamics/kafdrop
5   tag: latest
6   pullPolicy: Always
7
8 kafka:
9   brokerConnect: kfk-0.kfk.d2.svc.cluster.local:9092
10  properties: "c2VjdXJpdHkucHJvdG9jb2w9U0FTTF9QTEFJTlRFWFQKc2FzbC5tZWNoYW5pc209U0NSQU0tU0hBLTUxMgpzYXNsLmphYXMuY29uZmlnPW9yZy5hcGFjaGUua2Fma
11  2EuY29tbW9uLnNlY3VyaxR5LnNjcmFtLlNjcmFtTG9naW5Nb2R1bGUgcmlwZmlyZWQgdXNlcm5hbWU9ImthZmthLXVzZXIiIHBhc3N3b3JkPSJrYWZrYS1wYXNzd2
12  9yZCI7"
13  truststore: ""
14  keystore: ""
15  sasl.jaas.config: "kafka.properties"
16  truststoreUrl: "kafka.truststore.jks"
17  keystoreUrl: "kafka.keystore.jks"
18
19 test:
```

Deploying a browser for Kafka data (2)

- Create a namespace for Kafdrop

```
kubectl create namespace kafdrop
```

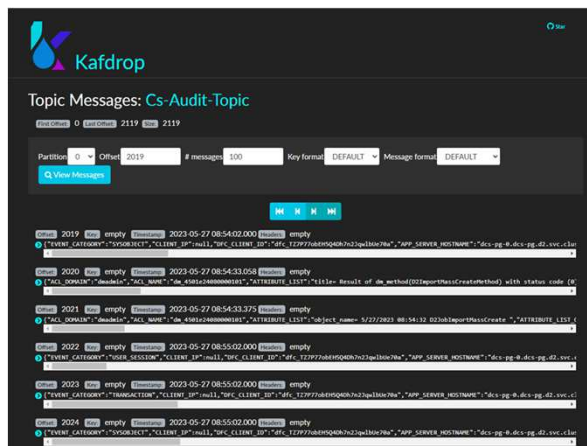
- From the directory containing the Helm Chart Deploy Kafkadrop with

```
helm install kafdrop . --namespace kafdrop
```

- To access Kafkadrop, issue:

```
kubectl proxy
```

Browse to <http://localhost:8001/api/v1/namespaces/kafdrop/services/http:kafdrop:9000/proxy>



Deploying RabbitMQ

- RabbitMQ is useful for managing messaging queues. It can be installed quickly into your cluster.
- Deploy RabbitMQ by issuing:

```
kubectl create namespace rmq
```

```
helm install rabbitmq oci://registry-1.docker.io/bitnamicharts/rabbitmq --set  
persistence.enabled=false -n rmq
```

- No PVCs configured. Data won't be persisted.
- Port forward ports for AMQP API and Management console

```
kubectl port-forward --namespace rmq svc/rabbitmq 5672:5672
```

```
kubectl port-forward --namespace rmq svc/rabbitmq 15672:15672
```

- By default decoded password: eVnaXY67NEZLEu7f
- By default decoded cookie: kzEfLHmq58FplhktDslWUCMt1yAkch4H



Accessing RabbitMQ

- Once the port is forwarded, browse to <http://127.0.0.1:15672/>

The screenshot shows the RabbitMQ management interface in a web browser. The address bar displays `127.0.0.1:15672/`. The page title is "RabbitMQ" with a version of 3.11.16 and Erlang 25.3.2. The navigation menu includes Overview, Connections, Channels, Exchanges, Queues, and Admin. The Overview page shows a "Totals" section with metrics for Queued messages, Current idle, Message rates, and Global counts. Below this, there are buttons for Connections, Channels, Exchanges, Queues, and Consumers. The "Nodes" section displays a table with columns for Name, File descriptors, Socket descriptors, Erlang processes, Memory, Disk space, Uptime, Info, and Reset stats. The table shows one node: `rabbit@rabbitmq-0.rabbitmq-headless.mq.svc.cluster.local`. The bottom of the page features a footer with links to HTTP API, Documentation, Tutorials, New releases, Commercial edition, Commercial support, Google Group, Discord, Slack, Plugins, and Github.

Name	File descriptors	Socket descriptors	Erlang processes	Memory	Disk space	Uptime	Info	Reset stats
<code>rabbit@rabbitmq-0.rabbitmq-headless.mq.svc.cluster.local</code>	54	0	490	148 MB	176 GiB	4m 13s	basic disc 5 r16	This node All nodes

RabbitMQ: Sending and receiving messages

Send.js

```
1 var amqp = require('amqplib/callback_api');
2
3 amqp.connect('amqp://user:eVnaXY67NEZLEu7f@localhost', function(error0, connection) {
4   if (error0) {
5     throw error0;
6   }
7   connection.createChannel(function(error1, channel) {
8     if (error1) {
9       throw error1;
10    }
11
12    var queue = 'test_queue';
13    var msg = '{"event":"DOCUMENT_REQUEST","r_object_id":"0901e240800001ed"}';
14
15    channel.assertQueue(queue, {
16      durable: false
17    });
18    channel.sendToQueue(queue, Buffer.from(msg));
19
20    console.log(" [x] Sent %s", msg);
21  });
22  setTimeout(function() {
23    connection.close();
24    process.exit(0);
25  }, 500);
26 });
```

```
C:\Users\jsotomay\OneDrive - OpenText\Documents\k8s\rabbitmq_tests>node send.js
[x] Sent {"event":"DOCUMENT_REQUEST","r_object_id":"0901e240800001ed"}
```

```
C:\Users\jsotomay\OneDrive - OpenText\Documents\k8s\rabbitmq_tests>
```

Receive.js

```
1 var amqp = require('amqplib/callback_api');
2
3 amqp.connect('amqp://user:eVnaXY67NEZLEu7f@localhost', function(error0, connection) {
4   if (error0) {
5     throw error0;
6   }
7   connection.createChannel(function(error1, channel) {
8     if (error1) {
9       throw error1;
10    }
11
12    var queue = 'test_queue';
13
14    channel.assertQueue(queue, {
15      durable: false
16    });
17
18    console.log(" [*] Waiting for messages in %s. To exit press CTRL+C", queue);
19
20    channel.consume(queue, function(msg) {
21      console.log(" [x] Received %s", msg.content.toString());
22    }, {
23      noAck: true
24    });
25  });
26 });
```

```
C:\Users\jsotomay\OneDrive - OpenText\Documents\k8s\rabbitmq_tests>node receive.js
[*] Waiting for messages in test_queue. To exit press CTRL+C
[x] Received {"event":"DOCUMENT_REQUEST","r_object_id":"0901e240800001ed"}
```



opentext™

Thank you



twitter.com/opentext



linkedin.com/company/opentext

opentext.com