MAAS CDS Installation Manual

MAAS CDS v2.5.0

generated on 2024-06-13

|  |  |  |  |
| --- | --- | --- | --- |
| **Written by** | **Responsibility-Office-Company** | **Date** | **Signature** |
| Antoine JAMMES | MAAS CDS Team |  |  |
| **Verified by** |  |  |  |
| Vincent Ravit | TPZ-F Quality assurance |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| **Approved** |  |  |  |
| Thomas TESTASECCA | MAAS CDS Product owner |  |  |

**Issuing entity** : DG Telespazio France

Table of Contents

[1 Omcs Deployment 3](#_Toc1046196911)

[1.1 Deployment process overview 4](#_Toc615925909)

[1.2 Prerequisites 5](#_Toc376819460)

[1.2.1 Nodes instanciation 6](#_Toc1258607272)

[1.2.2 Omcs-sftp deployment 6](#_Toc1258319183)

[1.2.3 Omcs-nfs deployment 7](#_Toc123181129)

[1.3 Deployment environment preparation 8](#_Toc1685795290)

[1.3.1 Deployment project sources 9](#_Toc1909849309)

[1.3.2 Namespace creation 9](#_Toc1201430848)

[1.3.3 Tag nodes 9](#_Toc1079794495)

[1.3.4 RabbitMQ operator deployment 10](#_Toc1226587716)

[1.3.5 NFS provisioner deployment 10](#_Toc1168263044)

[1.3.6 Container registry deployment 11](#_Toc1021273861)

[1.3.7 Container images upload 11](#_Toc1314878433)

[1.4 Cots deployment 13](#_Toc1786595227)

[1.4.1 Opensearch 14](#_Toc830890903)

[1.4.2 RabbitMQ 16](#_Toc352141722)

[1.5 Install MAAS 16](#_Toc1914267399)

[1.5.1 Create secrets 16](#_Toc893910749)

[1.5.2 Initialize the Opensearch database 18](#_Toc2035285003)

[1.5.3 MAAS nodes deployment 18](#_Toc356382645)

[1.5.4 First start 18](#_Toc2041800185)

[1.5.5 Deploy monitoring system 18](#_Toc257142095)

# Omcs Deployment

The purpose of this guide is to describes Omcs deployment.

Omcs application is based on severals item:

* Maas collectors, in charge of raw data collect.
* Maas cds engines, in charge of computation overs theses raw data.
* Grafana dashboards, in charge of displaying computation results in views.

It uses several cots:

* Opensearch database (Mandatory).
* Rabbitmq message broker (Mandatory).
* sftp server (Optional)
* nfs server (Optional)

It is based on containers and could be deployed on several orchestration systems.

* Docker-compose
* Swarm
* Kubernetes

For this documentation the “Kubernetes” deployment case is chosen.

## Deployment process overview

**Needed prerequisites**

A managed Kubernetes cluster for production environment the Kubernetes cluster it provides a set of nodes.

External service omcs-nfs provides a NFS server, optional, used by OpenSearch nodes to store snapshot of the database.

External service omcs-sftp provides a sftp server, optional, used by collectors, to store backups of all payload downloaded from external interfaces.

**Deployment environment preparation.**

The project sources, Clone omcs git repository.

Namespaces are used in the Kubernetes cluster to group deployed services into logical functions.

The used Tags in nodeSelector contexts to during deployment of the pods in Kubernetes cluster set somme affinities, they are based on nodeSeletors. Uses kubectl commands to set labels or tags for you own node affinity.

The RabbitMQ operator allow to deploy RabbitMQ cluster using [rabbitmq cluster operation project](https://github.com/rabbitmq/cluster-operator).

The NFS provisionner, to provide nfs persistent storages we uses a Nfs provisioner using [nfs-subdir-external-provisioner project](https://github.com/Kubernetes-sigs/nfs-subdir-external-provisioner) allowing to create PVC from a NFS server.

The container images repository used in Omcs is a Docker registry it uses [Docker Registry Helm Chart](https://github.com/twuni/docker-registry.helm). Binaries delivered are encapsulated in containers they need to be push (using Docker command) in the Kubernetes cluster Docker registry to be accessibles on pods instantiation.

**Cots deployment.**

Opensearch should be deployed it is the core database of the system.

Rabbitmq should be deployed it is the message brocker of the system.

**MAAS components deployment**

Store secrets, Mass components, to interact with others components, should share secrets as they are not provided the project values, theses secrets should be stored in the kubernetes cluster, whe sugest to store them in a password manager like Keepass.

The Opensearch database previously deployed should be Initialized before maas components deployment.

Then Maas collectors and maas-cds-engines should be deployed in the Kubernetes cluster using the provided helm templates and values.

A first start step should be performed to initialise dataflow configuration that act as a global configuration table for compute services or dashboards.

Then collectors and engines should be scaled.

All the items listed before are described below.

**⚠️️** At each **step** the **values** given in commands of file should be **checked** and **updated** accordingly to your **context**! **⚠️️**

**So lets start !**

## Prerequisites

1. A Kubernetes cluster
   * Kubernetes cluster the containers orchestrator.
2. External services
   * omcs-nfs provides a NFS server, use by OpenSearch master nodes, to store snapshot of the database.
   * omcs-sftp provides a sftp server, use by collectors, to store backups of all payloads downloaded from external interfaces.
3. Tools to interact with Kubernetes cluster
   * kubectl cli command is installed, and configured with the cluster config file (available in Keepass)
   * helm cli command is installed in order to provides backups storage spaces [helm install](https://helm.sh/docs/intro/install/).

### Nodes instanciation

The nodes (might be VM) should be created using your infrastructure management interface. Retrieve kubectl configuration file. Export env variable KUBECONFIG set to the path to this config file. This env variable is used by Helm and kubectl commands to connect to cluster.

export KUBECONFIG="/path/to/the/kubernetes/configuration/file"

Check your [context](#kubernetes) specificities!

### Omcs-sftp deployment

If you decide to not backup collected items on a sftp service, you can skip this step.

#### Attach sftp volume data disk

If not done previously the volume should be created, attached to the instance and mounted in it.

Check your [context](#attach-sftp-volume-data-disk) specificities!

Connect to sftp machine using ssh as root (or use sudo).

Create gpt partition using fdsik, mount the drive in fstab.

# init partition  
export SFTP\_DISK\_DEV="sdb" # The sftp disk dev name ex: sdb  
apt install -y parted  
parted /dev/${SFTP\_DISK\_DEV} --script mklabel gpt  
parted /dev/${SFTP\_DISK\_DEV} --script mkpart primary 0% 100%  
mkdir /data  
mkfs.ext4 /dev/${SFTP\_DISK\_DEV}1  
e2label /dev/${SFTP\_DISK\_DEV}1 DATA  
echo "LABEL=DATA /data ext4 discard,errors=remount-ro 0 1" >> /etc/fstab  
mount -a

#### Configure sftp

Configure sshd to provide sftp on the machine.

useradd -d /home/sftpmaas -s /bin/false sftpmaas  
passwd sftpmaas  
mkdir -p /data/sftp/files/MAAS/BACKUP/{ODATA,WEBDAV,SFTP,FTP,MON}  
mkdir -p /data/sftp/files/MAAS/{INGESTED,REJECTED,INBOX/REPRO}  
chown -R sftpmaas:sftpmaas /data/sftp/files  
  
cat <<'EOF' >> /etc/ssh/sshd\_config  
  
Match User sftpmaas  
 ChrootDirectory /data/sftp  
 ForceCommand internal-sftp  
 PasswordAuthentication yes  
 PermitTunnel no  
 AllowAgentForwarding no   
 X11Forwarding no  
  
EOF  
systemctl restart ssh  
systemctl restart sshd

Example of tree folders in results:

eouser@omcs-sftp:~$ find /data/sftp/files/MAAS/ -ls  
 9181456 4 drwxrwxr-x 5 sftpmaas sftpmaas 4096 Mar 11 15:09 /data/sftp/files/MAAS/  
 9181457 4 drwxrwxr-x 6 sftpmaas sftpmaas 4096 Mar 11 13:57 /data/sftp/files/MAAS/BACKUP  
 9181461 4 drwxrwxr-x 2 sftpmaas sftpmaas 4096 Mar 11 13:57 /data/sftp/files/MAAS/BACKUP/FTP  
 9181458 4 drwxrwxr-x 2 sftpmaas sftpmaas 4096 Mar 11 13:57 /data/sftp/files/MAAS/BACKUP/ODATA  
 9181459 4 drwxrwxr-x 2 sftpmaas sftpmaas 4096 Mar 11 13:57 /data/sftp/files/MAAS/BACKUP/WEBDAV  
 9181460 4 drwxrwxr-x 2 sftpmaas sftpmaas 4096 Mar 11 13:57 /data/sftp/files/MAAS/BACKUP/SFTP  
 9181460 4 drwxrwxr-x 2 sftpmaas sftpmaas 4096 Mar 11 13:57 /data/sftp/files/MAAS/BACKUP/MON  
 9181238 4 drwxrwxr-x 2 sftpmaas sftpmaas 4096 Mar 11 15:09 /data/sftp/files/MAAS/INGESTED  
 9181238 4 drwxrwxr-x 2 sftpmaas sftpmaas 4096 Mar 11 15:09 /data/sftp/files/MAAS/REJECTED  
 9181933 4 drwxrwxr-x 3 sftpmaas sftpmaas 4096 Mar 11 15:09 /data/sftp/files/MAAS/INBOX  
 9181941 4 drwxrwxr-x 2 sftpmaas sftpmaas 4096 Mar 11 15:09 /data/sftp/files/MAAS/INBOX/REPRO

Test sftp service.

sftp sftpmaas@localhost

#### Configure sftp backup script

Collected entries could be stored in sftp backup, this backup folder could be quickly heavy so it should be cleaned and or backup.

Check your [context](#configure-sftp-backup-script) specificities!

### Omcs-nfs deployment

If you decide to not snapshot opensearch database on a nfs service, you can skip this step.

#### Attach nfs volume data disk

If not done previously the volume should be created, attached to the instance and mounted in it.

Check your [context](#attach-nfs-volume-data-disk) specificities!

Connect to nfs machine using ssh as root (or uses sudo).

Create gpt partition using fdsik, mount the drive in fstab.

# init partition  
export NFS\_DISK\_DEV="sdb" # The sftp disk dev name ex: sdb  
apt install -y parted  
parted /dev/${NFS\_DISK\_DEV} --script mklabel gpt  
parted /dev/${NFS\_DISK\_DEV} --script mkpart primary 0% 100%  
mkdir /data  
mkfs.ext4 /dev/${NFS\_DISK\_DEV}1  
e2label /dev/${NFS\_DISK\_DEV}1 DATA  
echo "LABEL=DATA /data ext4 discard,errors=remount-ro 0 1" >> /etc/fstab  
mount -a

#### Configure nfs

Install nfs service on the machine (here ubuntu 22.04)

apt update  
apt upgrade  
apt install nfs-kernel-server  
mkdir -p /data/nfs  
chown -R nobody:nogroup /data/nfs  
chmod 777 /data/nfs  
mkdir -p /data/nfs/es-snapshots  
systemctl restart nfs-kernel-server  
# in case of ufw active  
ufw allow from 10.0.0.0/24 to any port nfs

Configure Nfs server create and modify exports.

export K8S\_NETWORK\_CIDR="192.168.1.0/24" # The K8s network cidr  
echo "/data/nfs ${K8S\_NETWORK\_CIDR}(rw,sync,no\_root\_squash,no\_all\_squash)" >> /etc/exports  
exit  
  
vim /etc/exports  
systemctl restart nfs-kernel-server

## Deployment environment preparation

This section describes the needed preparation before cots and application deployment.

* The deployment project sources
* The namespaces deployment used in the Kubernetes cluster.
* The tags used in nodeSelector contexts.
* The following providers/operator have been installed to ease infrastructure needs.
  + RabbitMQ
  + NfsProvider
  + Container registry
* The container images deployment.

### Deployment project sources

Clone omcs git repository.

All the scripts below needs path to access the values to use is based on env variable WORKING\_DIR and VALUES\_DIR path so they must be updated if needed and exported.

export WORKING\_DIR="." # The working dir path  
export VALUES\_DIR="${WORKING\_DIR}/deployment/values" # the helm values dir path

As the scripts below needs kubectl and helm commands path to access the connection configuration values to use is based on env variable KUBECONFIG so it has to be updated if needed and exported.

export KUBECONFIG="<./path/to/the/kube\_config.file>"

### Namespace creation

The following Kubernetes namespaces are defined to group deployed services into logical functions:

1. esa-csc-prd-cce-omcs-db is dedicated to the database provided by an Opensearch cluster.
2. esa-csc-prd-cce-omcs-etl is dedicated to collect and compute services.
3. esa-csc-prd-cce-omcs-front is dedicated to external interfaces with the dashboards.
4. storage is dedicated to an infrastructure function, providing a Kubernetes provisioner for the NFS storage.
5. monitoring is dedicated to an infrastructure function, providing the monitoring of the system.
6. container-registry is dedicated to an infrastructure function, providing a container registry used to store OMCS nonpublic containers

To create the namespaces, execute the following command on your terminal:

kubectl apply -f ${VALUES\_DIR}/omcs-namespace.yaml

The command should return:

namespace/esa-csc-prd-cce-omcs-db created  
namespace/esa-csc-prd-cce-omcs-etl created  
namespace/esa-csc-prd-cce-omcs-front created  
namespace/storage created  
namespace/monitoring created  
namespace/container-registry created

### Tag nodes

During deployment of the pods in Kubernetes cluster some affinities are defined, they are based on nodeSeletors.

Uses kubectl commands to set labels or tags for you own node affinity adjusted accordingly to your project values.

Check your [context](#tag-nodes) specificities!

### RabbitMQ operator deployment

The RabbitMQ operator allow to deploy RabbitMQ cluster using a Kubernetes manifest as specification.

This operator can be installed with [krew](https://krew.sigs.k8s.io/), a kubectl plugin manager.

This operator can be installed with [manifest](https://github.com/rabbitmq/cluster-operator/releases/latest/download/cluster-operator.yml) provided by rabbitmq.

Execute the following command from your terminal.

Using krew.

kubectl krew upgrade  
kubectl krew install rabbitmq  
kubectl rabbitmq install-cluster-operator

or with manifest

kubectl apply -f "https://github.com/rabbitmq/cluster-operator/releases/latest/download/cluster-operator.yml"

### NFS provisioner deployment

If you decide to not snapshot opensearch database on a nfs service, you can skip this step.

Persistent storages using nfs are claimed using a Nfs provisioner.

To configure this nfs-subdir-external-provisioner, retrieve nfs connection informations.

Check your [context](#nfs-provisioner-deployment) specificities!

The NFS provisioner allow to use the external storage on omcs-nfs as PersistentVolume in [ReadWriteMany](https://Kubernetes.io/fr/docs/concepts/storage/persistent-volumes/" \l "modes-d-acc%C3%A8s) mode.

As a prerequisite of the provisioner, the package nfs-common need to be installed on all nodes:

# Connect on each node as root user, and install nfs-common package:  
apt install -y nfs-common

Then install the provisioner from your local terminal with helm:

# Add nfs-subdir-external-provisioner helm repo  
helm repo add nfs-subdir-external-provisioner https://Kubernetes-sigs.github.io/nfs-subdir-external-provisioner/  
# Deploy the nfs provisioner  
helm install nfs-subdir-external-provisioner nfs-subdir-external-provisioner/nfs-subdir-external-provisioner --set nfs.server=<nfs-server-ip> --set nfs.path=/data/nfs

Ref: [nfs-subdir-external-provisioner](https://github.com/Kubernetes-sigs/nfs-subdir-external-provisioner) [nfs-provisioner.md](https://gitlab2.telespazio.fr/Maas/documentation/blob/develop/administration/k8s/nfs-provisioner.md)

### Container registry deployment

Container images repository used in Omcs is a Docker registry.

Check your [context](#container-registry-deployment) specificities!

The Helm package [Docker Registry Helm Chart](https://github.com/twuni/docker-registry.helm) have been used to install the container registry.

Create the target namespace for the registry:

kubectl create ns container-registry

Add helm repository to your local repository list:

helm repo add twuni https://helm.twun.io

Update the registry configuration if needed (ex: storage class): omcs-container-registry.yaml.

Deploy the registry with Helm:

helm install omcs-cr -n container-registry twuni/docker-registry -f ${VALUES\_DIR}/omcs-container-registry.yaml

If destroy needed:

helm delete omcs-cr -n container-registry

### Container images upload

Binaries delivered are containers and need to be push (using Docker command) in the cluster Docker registry.

To **Upload/Push** containers images needs to be present locally on your machine with the right tag, getting them from an external registry on building them on your local machine.

Containers are sent to the target environment with a port-forward on the service, throw a Kubernetes connection

Retrieve and tag containers:

**From external registry**

Update and export used variables.

export ORIGIN\_DOCKER\_REGISTRY="<http://your-docker-registry:port>" # the docker registry used to retrieve maas docker images empty if images are build locally  
export MAAS\_COLLECTOR\_DOCKER\_IMAGE\_PATH="<path>" # image path in the registry  
export MAAS\_CDS\_DOCKER\_IMAGE\_PATH="<path>" # image path in the registry  
export MAAS\_GRAFANA\_INIT\_IMAGE\_PATH="<path>" # image path in the registry  
export MAAS\_PSQL\_DOCKER\_IMAGE\_PATH="<path>" # image path in the registry  
export MAAS\_COLLECTOR\_DOCKER\_IMAGE\_VERSION="<release-X.Y.Z>" # image version to use  
export MAAS\_CDS\_DOCKER\_IMAGE\_VERSION="<release-X.Y.Z>" # image version to use  
export MAAS\_GRAFANA\_INIT\_DOCKER\_IMAGE\_VERSION="<release-X.Y.Z>" # image version to use  
export MAAS\_PSQL\_DOCKER\_IMAGE\_VERSION="<release-X.Y.Z>" # image version to use

Commands:

# pull images  
# maas-collector  
docker pull ${ORIGIN\_DOCKER\_REGISTRY}${MAAS\_COLLECTOR\_DOCKER\_IMAGE\_PATH}/maas-collector:${MAAS\_COLLECTOR\_DOCKER\_IMAGE\_VERSION}  
# maas-cds  
docker pull ${ORIGIN\_DOCKER\_REGISTRY}${MAAS\_CDS\_DOCKER\_IMAGE\_PATH}/maas-cds:${MAAS\_COLLECTOR\_DOCKER\_IMAGE\_VERSION}  
# init-grafana  
docker pull ${ORIGIN\_DOCKER\_REGISTRY}${MAAS\_GRAFANA\_INIT\_DOCKER\_IMAGE\_PATH}/init-grafana:${MAAS\_GRAFANA\_INIT\_DOCKER\_IMAGE\_VERSION}  
# psql-client-s3 (for grafana db backup)  
docker pull ${ORIGIN\_DOCKER\_REGISTRY}${MAAS\_PSQL\_DOCKER\_IMAGE\_PATH}/psql-client-s3:${MAAS\_PSQL\_DOCKER\_IMAGE\_VERSION}  
  
#Tag images  
docker tag ${ORIGIN\_DOCKER\_REGISTRY}${MAAS\_COLLECTOR\_DOCKER\_IMAGE\_PATH}/maas-collector:${MAAS\_COLLECTOR\_DOCKER\_IMAGE\_VERSION} localhost:5000/maas/maas-collector:${MAAS\_COLLECTOR\_DOCKER\_IMAGE\_VERSION}  
docker tag ${ORIGIN\_DOCKER\_REGISTRY}${MAAS\_CDS\_DOCKER\_IMAGE\_PATH}/maas-cds:${MAAS\_COLLECTOR\_DOCKER\_IMAGE\_VERSION} localhost:5000/maas/cds/maas-cds:${MAAS\_COLLECTOR\_DOCKER\_IMAGE\_VERSION}  
docker tag ${ORIGIN\_DOCKER\_REGISTRY}${MAAS\_GRAFANA\_INIT\_DOCKER\_IMAGE\_PATH}/init-grafana:${MAAS\_GRAFANA\_INIT\_DOCKER\_IMAGE\_VERSION} localhost:5000/maas/init-grafana:${MAAS\_GRAFANA\_INIT\_DOCKER\_IMAGE\_VERSION}  
docker tag ${ORIGIN\_DOCKER\_REGISTRY}${MAAS\_PSQL\_DOCKER\_IMAGE\_PATH}/psql-client-s3:${MAAS\_PSQL\_DOCKER\_IMAGE\_VERSION} localhost:5000/maas/psql-client-s3:${MAAS\_PSQL\_DOCKER\_IMAGE\_VERSION}

**From locals images**

If you build localy containers images you just have to tag them.

Update and export used variables.

export MAAS\_COLLECTOR\_DOCKER\_IMAGE\_VERSION="<release-X.Y.Z>" # image version to use  
export MAAS\_CDS\_DOCKER\_IMAGE\_VERSION="<release-X.Y.Z>" # image version to use  
export MAAS\_GRAFANA\_INIT\_DOCKER\_IMAGE\_VERSION="<release-X.Y.Z>" # image version to use  
export MAAS\_PSQL\_DOCKER\_IMAGE\_VERSION="<release-X.Y.Z>" # image version to use

Commands:

#Tag images  
docker tag maas-collector:${MAAS\_COLLECTOR\_DOCKER\_IMAGE\_VERSION} localhost:5000/maas/maas-collector:${MAAS\_COLLECTOR\_DOCKER\_IMAGE\_VERSION}  
docker tag maas-cds:${MAAS\_CDS\_DOCKER\_IMAGE\_VERSION} localhost:5000/maas/cds/maas-cds:${MAAS\_CDS\_DOCKER\_IMAGE\_VERSION}  
docker tag init-grafana:${MAAS\_GRAFANA\_INIT\_DOCKER\_IMAGE\_VERSION} localhost:5000/maas/init-grafana:${MAAS\_GRAFANA\_INIT\_DOCKER\_IMAGE\_VERSION}  
docker tag psql-client-s3:${MAAS\_PSQL\_DOCKER\_IMAGE\_VERSION} localhost:5000/maas/psql-client-s3:${MAAS\_PSQL\_DOCKER\_IMAGE\_VERSION}

Maybe you need to add the repository as insecure registry in daemon.json

In unix sys locate at: /etc/docker/daemon.json

{  
 "insecure-registries": ["${ORIGIN\_DOCKER\_REGISTRY}"]  
}

Send docker images to the Kubernetes cluster registry using port forward.

# Open port-forward to production registry  
kubectl -n container-registry port-forward svc/omcs-cr-docker-registry 5000 &  
# store pid to kill the port forward   
\_pid=$!  
echo "$\_pid" >.port\_forward.pid  
  
# Push images  
docker push localhost:5000/maas/cds/maas-cds:${MAAS\_CDS\_DOCKER\_IMAGE\_VERSION}  
docker push localhost:5000/maas/maas-collector:${MAAS\_COLLECTOR\_DOCKER\_IMAGE\_VERSION}  
docker push localhost:5000/maas/init-grafana:${MAAS\_GRAFANA\_INIT\_DOCKER\_IMAGE\_VERSION}  
docker push localhost:5000/maas/psql-client-s3:${MAAS\_PSQL\_DOCKER\_IMAGE\_VERSION}

Check images pushed :

curl http://localhost:5000/v2/\_catalog  
curl http://localhost:5000/v2/maas/maas-cds/tags/list  
curl http://localhost:5000/v2/maas/maas-collector/tags/list  
curl http://localhost:5000/v2/maas/init-grafana/tags/list

Kill the port forward process.

# kill port forward  
\_pid=$(cat .port\_forward.pid)  
kill -9 ${\_pid}  
rm .port\_forward.pid

## Cots deployment

Specific platform configuration could be update from the given sample if you copy them and store them in another place don’t forget to update you VALUES\_DIR env variable to the right path. On the following commands the following variables the current platform the sample values are used.

If you use an external helm repository yous have to set it in the variables below.

* From external repository

export HELM\_REPOSITORY="<Your external helm repository>" # the helm repository used to retrieve helm packages "." if heml gets localy  
export HELM\_VERSION="<the helm version to use>" # the helm package version  
export HELM\_VERSION\_ARG="--version ${HELM\_VERSION}" # the arg to give in command empty "" if no version given

* Or from local env

export HELM\_REPOSITORY="." # the helm repository used to retrieve helm packages "." if heml gets localy  
export HELM\_VERSION="" # the helm package version  
export HELM\_VERSION\_ARG="" # the arg to give in command empty "" if no version given

### Opensearch

#### Nfs pvc / pv

If you decide to not snapshot opensearch database on a nfs service, you can skip this step.

Some infrastructure prerequisites must be installed to provides snapshot storages using external NFS service and nfs provider pvc.

The following command will create disks where Opensearch snapshot will be stored. Enter following commands in your local terminal:

The spec/nfs/server value should match the ip of omcs-nfs service.

Create the persistent volume.

# Update tpz helm repo to add last release  
helm repo update  
# Create persistence volume for elasticsearch snapshots  
kubectl apply -f ${VALUES\_DIR}/es-snapshot-pvc.yaml

#### Certificates and Secrets

Create certificates and secrets used by Opensearch and other components for security, files and passwords to push in secret should be stored in Keepass.

##### Generate Opensearch certificates

Done one time and then be stored in a Keepass.

Check your [context](#generate_opensearch_certificates) specificities!

Exemple of certificates creation.

#!/bin/sh  
# Root CA  
openssl genrsa -out root-ca-key.pem 1024  
openssl req -new -x509 -sha128 -key root-ca-key.pem -subj "/C=AA/ST=BBBBBBB/L=CCCCCCC/O=DDDDDDD/OU=EEEE/CN=adminCN" -out root-ca.pem -days 2  
# Admin cert  
openssl genrsa -out admin-key-temp.pem 1024  
openssl pkcs8 -inform PEM -outform PEM -in admin-key-temp.pem -topk8 -nocrypt -v1 PBE-SHA1-3DES -out admin-key.pem  
openssl req -new -key admin-key.pem -subj "/C=AA/ST=BBBBBBB/L=CCCCCCC/O=DDDDDDD/OU=EEEE/CN=A" -out admin.csr  
openssl x509 -req -in admin.csr -CA root-ca.pem -CAkey root-ca-key.pem -CAcreateserial -sha128 -out admin.pem -days 2  
   
# Node cert  
openssl genrsa -out node1-key-temp.pem 1024  
openssl pkcs8 -inform PEM -outform PEM -in node1-key-temp.pem -topk8 -nocrypt -v1 PBE-SHA1-3DES -out node1-key.pem  
openssl req -new -key node1-key.pem -subj "/C=AA/ST=BBBBBBB/L=CCCCCCC/O=DDDDDDD/OU=EEEE/CN=node1CN" -out node1.csr  
#echo 'subjectAltName=DNS:node1.dns.a-record' > node1.ext  
# openssl x509 -req -in node1.csr -CA root-ca.pem -CAkey root-ca-key.pem -CAcreateserial -sha128 -out node1.pem -days 2 -extfile node1.ext  
openssl x509 -req -in node1.csr -CA root-ca.pem -CAkey root-ca-key.pem -CAcreateserial -sha128 -out node1.pem -days 2

##### Opensearch certificates as secrets

Store Opensearch certificates as secret (used for deployment). Certificates files are in the Keepass.

export NODE1\_PEM\_FILE="./node1.pem" # the node1.pem File path file in the keepass  
export NODE1\_KEY\_PEM\_FILE="./node1-key.pem" # the node1-key.pem File path file in the keepass  
export ROOT\_CA\_PEM\_FILE="./root-ca.pem" # the root-ca.pem File path file in the keepass

kubectl delete secret -n esa-csc-prd-cce-omcs-db opensearch-ssl  
kubectl create secret -n esa-csc-prd-cce-omcs-db generic opensearch-ssl --from-file=${NODE1\_PEM\_FILE} --from-file=${NODE1\_KEY\_PEM\_FILE} --from-file=${ROOT\_CA\_PEM\_FILE}

##### Opensearch user and roles as secrets

Store opensearch user and roles as secret (used for deployment). User and roles files are in the Keepass.

export INTERNAL\_USER\_YML\_FILE="" # the Internal\_user.yml File path file in the keepass  
export ROLES\_YML\_FILE="" # the roles.yml File path file in the keepass  
export ROLES\_MAPPING\_YML\_FILE="" # the roles\_mapping.yml File path file in the keepass

# deploy elastic user and roles as secrets  
kubectl delete secret -n esa-csc-prd-cce-omcs-db internalusers  
kubectl create secret -n esa-csc-prd-cce-omcs-db generic internalusers --from-file=${INTERNAL\_USER\_YML\_FILE}  
kubectl delete secret -n esa-csc-prd-cce-omcs-db roles  
kubectl create secret -n esa-csc-prd-cce-omcs-db generic roles --from-file=${ROLES\_YML\_FILE}  
kubectl delete secret -n esa-csc-prd-cce-omcs-db rolesmapping  
kubectl create secret -n esa-csc-prd-cce-omcs-db generic rolesmapping --from-file=${ROLES\_MAPPING\_YML\_FILE}

#### Opensearch database deployment

Deploy Opensearch pod in the Kubernetes cluster pods.

# Install database (opencsearch) service  
helm upgrade -n esa-csc-prd-cce-omcs-db --install cds-prd-db ${HELM\_REPOSITORY} ${HELM\_VERSION\_ARG} -f ${VALUES\_DIR}/values-prod-db.yaml

#### Database post deployment (Optional)

##### Opensearch s3 default client access

In the case of S3 snapshot usage access and key stored in key store could be defined in a secret (used at node deployment to be stored in opensearch key store). Because this definition could change in the project life we decide to store theses key after openseach deployment using commands.

On the db pods.

opensearch-keystore add s3.client.default.access\_key  
opensearch-keystore add s3.client.default.secret\_key

Or using a script:

# the list of db pod to renew access key and secret  
POD\_LIST="$(kubectl get pods -n esa-csc-prd-cce-omcs-db | grep "prod-db"|awk '{print$1}')"  
  
for POD in ${POD\_list}  
do echo "handling pod": ${POD}  
 kubectl exec -it -n esa-csc-prd-cce-omcs-db ${POD} -- /bin/bash -c "opensearch-keystore list ; opensearch-keystore remove s3.client.default.access\_key ;opensearch-keystore remove s3.client.default.secret\_key; echo ${S3\_ACCESS\_KEY} | opensearch-keystore add --stdin --force s3.client.default.access\_key && echo ${S3\_SECRET\_KEY} | opensearch-keystore add --stdin --force s3.client.default.secret\_key ; opensearch-keystore list"  
done

After changing theses keys security setting **should be reloaded**.

POST /\_nodes/reload\_secure\_settings

### RabbitMQ

Deploy Rabbitmq pod in the Kubernetes cluster pods.

# Install RabbitMq service  
helm upgrade -n esa-csc-prd-cce-omcs-etl --install cds-prd-rmq ${HELM\_REPOSITORY} ${HELM\_VERSION\_ARG} -f ${VALUES\_DIR}/values-prod-rmq.yaml

## Install MAAS

### Create secrets

The following commands will create all secrets needed in the cluster, as they are not provided in values stored in git. Instead, all secrets need to be retrieved from the password manager Keepass.

Check your [context](#Create_secrets) specificities!

Execute the commands from your local terminal:

The values below a specific forprod.

export BACKUP\_PORT="22" # the sftp port used  
export BACKUP\_USERNAME="" # the sftp user name in Keepass  
export BACKUP\_PASSWORD="" # the sftp user password in Keepass  
export COLLECTOR\_API\_CREDENTIALS\_FILE="" # the path tho the credecials file  
export ELASTIC\_URL="" # the K8s service url to the opensearch client  
export ELASTIC\_EDITOR\_USERNAME="" # the elastcic editor user name in Keepass  
export ELASTIC\_EDITOR\_PASSWORD="" # the elastcic editor user password in Keepass  
export ELASTIC\_READONLY\_USERNAME="readall" # the user readall name   
export ELASTIC\_READONLY\_PASSWORD="" # the user readall password in Keepass  
export GF\_SMTP\_FROM\_ADDRESS="" # the mail from address used for mailing alerts in Keepass  
export GF\_SMTP\_FROM\_NAME="" # the mail from user name in Keepass  
export GF\_SMTP\_HOST="" # the external smtp host in Keepass  
export GF\_SMTP\_USER="" # the external smtp user in Keepass  
export GF\_SMTP\_PASSWORD="" # the external smtp user password in Keepass  
export GRAFANA\_ADMIN\_PWD="" # the grafana admin user name in Keepass  
export GRAFANA\_DB\_ADMIN\_PASSWORD="" # the grafana admin user password in Keepass  
export S3\_ENDPOINT="" # the snapshot s3 end point in Keepass  
export S3\_ACCES\_KEY="" # the snapshot s3 access key in Keepass  
export S3\_SECRET\_KEY="" # the snapshot s3 secret key in Keepass

# Provide SFTP credentials to collector service  
echo  
echo "Provide SFTP credentials to collector service"  
kubectl delete secret -n esa-csc-prd-cce-omcs-etl etl-secrets  
kubectl create secret -n esa-csc-prd-cce-omcs-etl generic etl-secrets --from-literal=SFTP\_USERNAME="${BACKUP\_USERNAME}" --from-literal=SFTP\_PASSWORD="${BACKUP\_PASSWORD}" \  
--from-literal=BACKUP\_HOSTNAME="${BACKUP\_HOSTNAME}" \  
--from-literal=BACKUP\_PORT="${BACKUP\_PORT}" \  
--from-literal=BACKUP\_USERNAME="${BACKUP\_USERNAME}" \  
--from-literal=BACKUP\_PASSWORD="${BACKUP\_PASSWORD}"  
  
# Initiate the collector credentials file from a local file extracted from the Keepass  
echo   
echo "Initiate the collector credentials file from a local file extracted from the Keepass"  
kubectl delete secret -n esa-csc-prd-cce-omcs-etl collector-credentials  
kubectl create secret -n esa-csc-prd-cce-omcs-etl generic collector-credentials --from-file=${COLLECTOR\_API\_CREDENTIALS\_FILE}  
kubectl create secret -n esa-csc-prd-cce-omcs-front generic s3-secrets --from-literal=S3\_ACCESS\_KEY=${S3\_ACCES\_KEY} --from-literal=S3\_KEY\_ID=${S3\_KEY\_ID} --from-literal=S3\_ENDPOINT=${S3\_ENDPOINT}   
  
# Provide credentials to connect Opensearch from ETL  
echo  
echo "Provide credentials to connect Opensearch from ETL"  
kubectl delete secret -n esa-csc-prd-cce-omcs-etl elasticsearch-client-secret  
kubectl create secret -n esa-csc-prd-cce-omcs-etl generic elasticsearch-client-secret --from-literal=url=${ELASTIC\_URL} --from-literal=username=${ELASTIC\_EDITOR\_USERNAME} --from-literal=password=${ELASTIC\_EDITOR\_PASSWORD}  
  
# Copy credentials to connect rabbitmq from ETL (copy from generated rabbit secret)   
echo  
echo "Copy credentials to connect rabbitmq from ETL (copy from generated rabbit secret)"  
NS\_FROM="esa-csc-prd-cce-omcs-etl"  
NS\_TARGET="esa-csc-prd-cce-omcs-etl"  
DEPLOYMENT\_NAME="cds-prd-rmq"  
RABBIT\_USER=$(kubectl -n $NS\_FROM get secrets/${DEPLOYMENT\_NAME}-rabbitmq-default-user --template="{{.data.username | base64decode}}")  
RABBIT\_PASS=$(kubectl -n $NS\_FROM get secrets/${DEPLOYMENT\_NAME}-rabbitmq-default-user --template="{{.data.password | base64decode}}")  
RABBIT\_URL=$(kubectl -n $NS\_FROM get secrets/${DEPLOYMENT\_NAME}-rabbitmq-default-user --template="amqp://{{.data.host | base64decode}}:{{.data.port | base64decode}}")  
kubectl delete secret -n $NS\_TARGET rabbitmq-client-secret  
kubectl create secret -n $NS\_TARGET generic rabbitmq-client-secret --from-literal=username=$RABBIT\_USER --from-literal=password=$RABBIT\_PASS --from-literal=url=$RABBIT\_URL  
   
# Provide credentials to connect elastisearch from Grafana  
echo  
echo "Provide credentials to connect elastisearch from Grafana"  
kubectl delete secret -n esa-csc-prd-cce-omcs-front elasticsearch-client-secret  
kubectl create secret -n esa-csc-prd-cce-omcs-front generic elasticsearch-client-secret --from-literal=url=${ELASTIC\_URL} --from-literal=username=${ELASTIC\_READONLY\_USERNAME} --from-literal=password=${ELASTIC\_READONLY\_PASSWORD}  
#⚠️ No S3 In MAGNUM context   
#kubectl create secret -n esa-csc-prd-cce-omcs-front s3-secrets --from-literal=S3\_ACCESS\_KEY=<s3 access key> --from-literal=S3\_KEY\_ID=<s3 key id> --from-literal=S3\_ENDPOINT=<s3 endpoint>   
  
# Initialize Grafana smtp config user credentials  
echo  
echo "Initialize Grafana smtp config user credentials"  
kubectl delete secret -n esa-csc-prd-cce-omcs-front grafana-smtp-config  
kubectl create secret -n esa-csc-prd-cce-omcs-front generic grafana-smtp-config --from-literal=GF\_SMTP\_ENABLED=true --from-literal=GF\_SMTP\_FROM\_ADDRESS=${GF\_SMTP\_FROM\_ADDRESS} --from-literal=GF\_SMTP\_FROM\_NAME=${GF\_SMTP\_FROM\_NAME} --from-literal=GF\_SMTP\_HOST=${GF\_SMTP\_HOST} --from-literal=GF\_SMTP\_USER=${GF\_SMTP\_USER} --from-literal=GF\_SMTP\_PASSWORD=${GF\_SMTP\_PASSWORD}   
  
# Initialize Grafana admin user credentials  
echo  
echo "Initialize Grafana admin user credentials"  
kubectl delete secret -n esa-csc-prd-cce-omcs-front gf-admin-secret  
kubectl create secret -n esa-csc-prd-cce-omcs-front generic gf-admin-secret --from-literal=admin-user=admin --from-literal=admin-password=${GRAFANA\_ADMIN\_PWD}  
#⚠️ No S3 In MAGNUM context   
#kubectl create secret -n esa-csc-prd-cce-omcs-front s3-secrets --from-literal=S3\_ACCESS\_KEY=<s3 access key> --from-literal=S3\_KEY\_ID=<s3 key id> --from-literal=S3\_ENDPOINT=<s3 endpoint>   
  
# Provide to Grafana RDS credentials (Postgres database)  
echo  
echo "Provide to Grafana RDS credentials (Postgres database)"  
kubectl delete secret -n esa-csc-prd-cce-omcs-front maas-secret  
kubectl create secret -n esa-csc-prd-cce-omcs-front generic maas-secret --from-literal=GF\_DATABASE\_PASSWORD=${GRAFANA\_DB\_ADMIN\_PASSWORD}  
  
# rabbit copy  
# not useful because in same namespace

### Initialize the Opensearch database

Create a pod maas-engine-cli to run maas-cds engine, initializing Opensearch indices with templates stored in the maas-cds container.

# deploy db initaliser pod  
kubectl apply -f ${VALUES\_DIR}/omcs-db-init.yaml

# connect in pod using bash  
kubectl exec -it -n esa-csc-prd-cce-omcs-etl maas-engines-cli -- /bin/bash  
# Initialize templates and created indices for the current timeline  
maas\_migrate -v --install all --populate cds-s2-tilpar-tiles.bulk.xz

### MAAS nodes deployment

The following commands will install the core applications of MAAS CDS.

Execute the commands from your local terminal:

# Install db   
helm upgrade -n esa-csc-prd-cce-omcs-db --install cds-prd-db ${HELM\_REPOSITORY} ${HELM\_VERSION\_ARG} -f ${VALUES\_DIR}-prod-db.yaml  
  
# Install Frontend postgres database  
helm upgrade -n esa-csc-prd-cce-omcs-front --install cds-prd-front-db ${HELM\_REPOSITORY} ${HELM\_VERSION\_ARG} -f $VALUES\_DIR/values-prod-front-db.yaml  
  
# Install frontend apps  
helm upgrade -n esa-csc-prd-cce-omcs-front --install cds-prd-front ${HELM\_REPOSITORY} ${HELM\_VERSION\_ARG}-f $VALUES\_DIR/values-prod-front.yaml  
  
# Deploy collectors and engines services scaled to 0  
helm upgrade -n esa-csc-prd-cce-omcs-etl --install cds-prd-etl ${HELM\_REPOSITORY} ${HELM\_VERSION\_ARG} -f $VALUES\_DIR/values-prod-etl.yaml \  
 --set collector-odata.replicaCount=0 \  
 --set collector-ftp.replicaCount=0 \  
 --set collector-sftp.replicaCount=0 \  
 --set collector-webdav.replicaCount=0 \  
 --set collector-jira.replicaCount=0 \  
 --set collector-rosftp.replicaCount=0 \  
 --set collector-monitor.replicaCount=0 \  
 --set maas-engine-collect.replicaCount=0 \  
 --set maas-engine-cds-only-completeness-s1-s2.replicaCount=0 \  
 --set maas-engine-cds-only-completeness-s5.replicaCount=0 \  
 --set maas-engine-raw-only-dd.replicaCount=0 \  
 --set maas-engine-raw-only-other.replicaCount=0 \  
 --set maas-engine-cds-only-completeness-s3.replicaCount=0 \  
 --set maas-engine-cds-only-other.replicaCount=0 \  
 --set maas-engine-raw-only-lta.replicaCount=0 \  
 --set maas-engine-raw-only-prip.replicaCount=0

The application is now deployed, but without any collection or engine enabled.

### First start

Before collecting data, the dataflow configuration must be loaded.

Dataflow act as a global configuration table for compute services or dashboards.

From Filezilla or sftp command, push the dataflow configuration file in configuration/dataflow on input ftp in /files/MAAS/INBOX/DATAFLOW/ folder as dot prefixed file for transfert then remove dot prefix ti allow system to collect this data.

Once the file ready to be ingest on the sftp server, we can start the sftp collector:

kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=1 deployment/cds-prd-etl-collector-sftp

After changing the dataflow configuration, all engines need to be started:

# scale engines   
kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=2 deployment/cds-prd-etl-raw-only-prip   
kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=1 deployment/cds-prd-etl-raw-only-other   
kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=2 deployment/cds-prd-etl-raw-only-lta   
kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=1 deployment/cds-prd-etl-raw-only-dd   
kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=1 deployment/cds-prd-etl-cds-only-other   
kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=2 deployment/cds-prd-etl-cds-only-completeness-s5   
kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=1 deployment/cds-prd-etl-cds-only-completeness-s3   
kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=1 deployment/cds-prd-etl-cds-only-completeness-s1-s2

All compute-engine are now running, have initialized the RabbitMQ topology, and loaded the last dataflow configuration.

Then scale all collectors with default values in configuration to begin the collection:

# scale collectors  
kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=1 deployment/cds-prd-etl-collector-webdav   
kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=1 deployment/cds-prd-etl-collector-sftp   
kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=1 deployment/cds-prd-etl-collector-rosftp   
kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=2 deployment/cds-prd-etl-collector-odata   
kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=1 deployment/cds-prd-etl-collector-mpip   
kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=1 deployment/cds-prd-etl-collector-monitor   
kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=1 deployment/cds-prd-etl-collector-loki   
kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=1 deployment/cds-prd-etl-collector-jira   
kubectl scale -n esa-csc-prd-cce-omcs-etl --replicas=1 deployment/cds-prd-etl-collector-ftp

### Deploy monitoring system

A pre-configured kube-prometheus-stack and loki-stack is available in the maas-cds helm chart.

This allow to monitor application log and metrics to ease system administration.

Check your [context](#Deploy_monitoring_system) specificities!

To deploy theses stacks, run from your local terminal the following commands:

# Create a new namespace for the monitoring  
kubectl create ns monitoring  
  
# Provide credentials to connect Opensearch from monitoring  
kubectl create secret -n monitoring generic elasticsearch-client-secret --from-literal=url=<Elastic url> --from-literal=username=<Elastic admin username> --from-literal=password=<Elastic admin password>   
  
# Deploy public dashboard  
helm upgrade -n monitoring cds-prd-mon tpzfr/maas-cds --version 1.10.0 -f values/prod/values-prod-monitoring.yaml  
kubectl -n monitoring apply --filename manifests/prometheus/monitors/  
kubectl -n monitoring apply --recursive --filename manifests/prometheus/rules/  
kubectl -n monitoring apply --filename manifests/grafana/dashboards/