HCL SOFTWARE

HCL Digital Experience

Step-by-Step Guide

How to deploy DX CF_197 with DAM and CC on Azure AKS using HELM Chart

Author:

Fernanda de Sousa Gomes

HCL Digital Experience L2 Support

HCL Technologies

Contents

What you will find in this guide 3
Preparing your Azure Client Machine 4
Install Docker on Linux Fedora 4
Install Azure Client on Linux 4
Creating a Resource Group and Container Registry 6
Loading, Tagging and Pushing your DX images 8
Creating the Azure AKS Cluster 10
Set up the NFS server 11
Connecting AKS Cluster to NFS Server 13
Generate TLS Certificate 15
Install HELM on Linux Ubuntu 16
Deploy DX using HELM Chart 16
Update the hostname of your DX deploy 22
Testing your Portal deployment 23

What you will find in this guide

This guide will show you how to deploy HCL Digital Experience 9.5 CF197 + Digital Asset Manager + Content Composer on Azure AKS in NFS volumes using HELM Charts.

As part of the experience, we will show you how to install azure client, docker, kubectl, the NFS server and Helm.

Preparing your Azure Client Machine

In this guide, we have installed docker and the azure client on Linux running Fedora.

For the purpose of this guide, all commands are linux-based.

If you are **not** using Linux for your client machine, make sure you install the following software on your local machine and you may skip this section:

- Docker - https://docs.docker.com/engine/install/
- Azure client - https://docs.microsoft.com/en-us/cli/azure/install-azure-cli

Install Docker on Linux Fedora

The procedure below explains how to install Docker on your Fedora VM:

https://docs.docker.com/engine/install/fedora/#install-from-a-package

1. In summary, docker can be easily installed on Fedora with the following commands:

```
sudo systemctl enable sshd

sudo systemctl start sshd

curl -fsSL https://get.docker.com -o get-docker.sh

sudo sh get-docker.sh

sudo usermod -aG docker <your-user>

sudo dnf install grubby

sudo grubby --update-kernel=ALL --
args="systemd.unified_cgroup_hierarchy=0"

reboot
```

2. You can start docker using the command below:

```
sudo systemctl start docker
```

Install Azure Client on Linux

As a reference, we have used the procedure from this link:

https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-linux?pivots=dnf

1. Import the Microsoft repository key.

```
sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc
```

2. Create a file called "`azure-cli`" which will contain the following repository information:

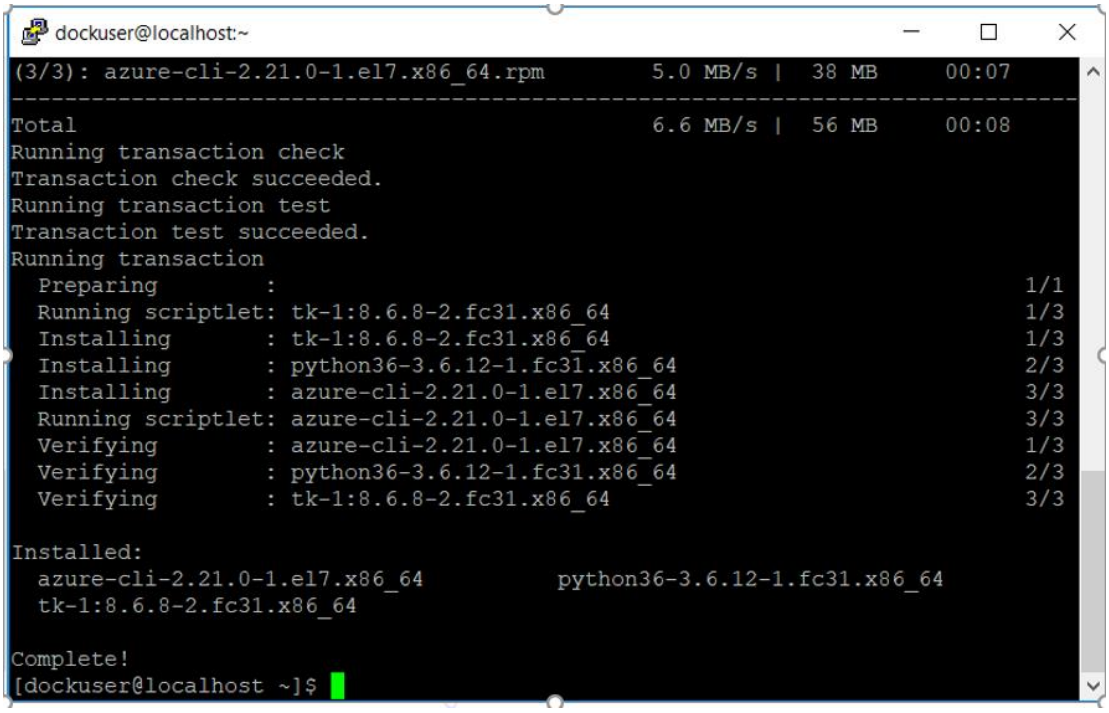
```
echo -e "[azure-cli]

name=Azure CLI

baseurl=https://packages.microsoft.com/yumrepos/azure-cli
```

```
enabled=1

gpgcheck=1

gpgkey=https://packages.microsoft.com/keys/microsoft.asc" | sudo tee
/etc/yum.repos.d/azure-cli.repo
```

3. Install with the `dnf install` command.

```
sudo dnf install azure-cli
```

The result should be:



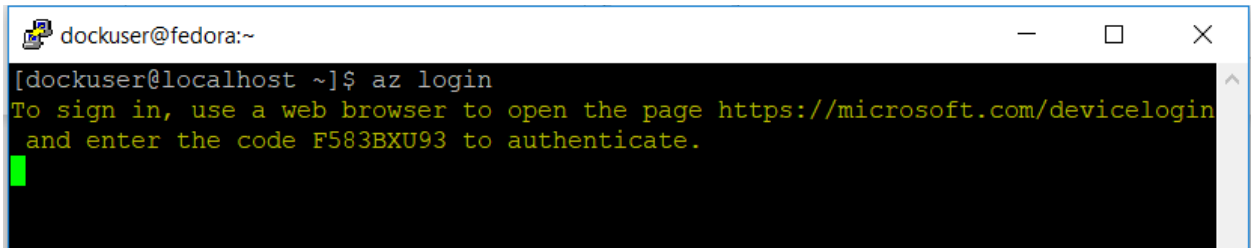
Creating a Resource Group and Container Registry

The steps below will show you how you can create your Resource Group and Container Registry in Azure:

1. Authenticate in our azure account

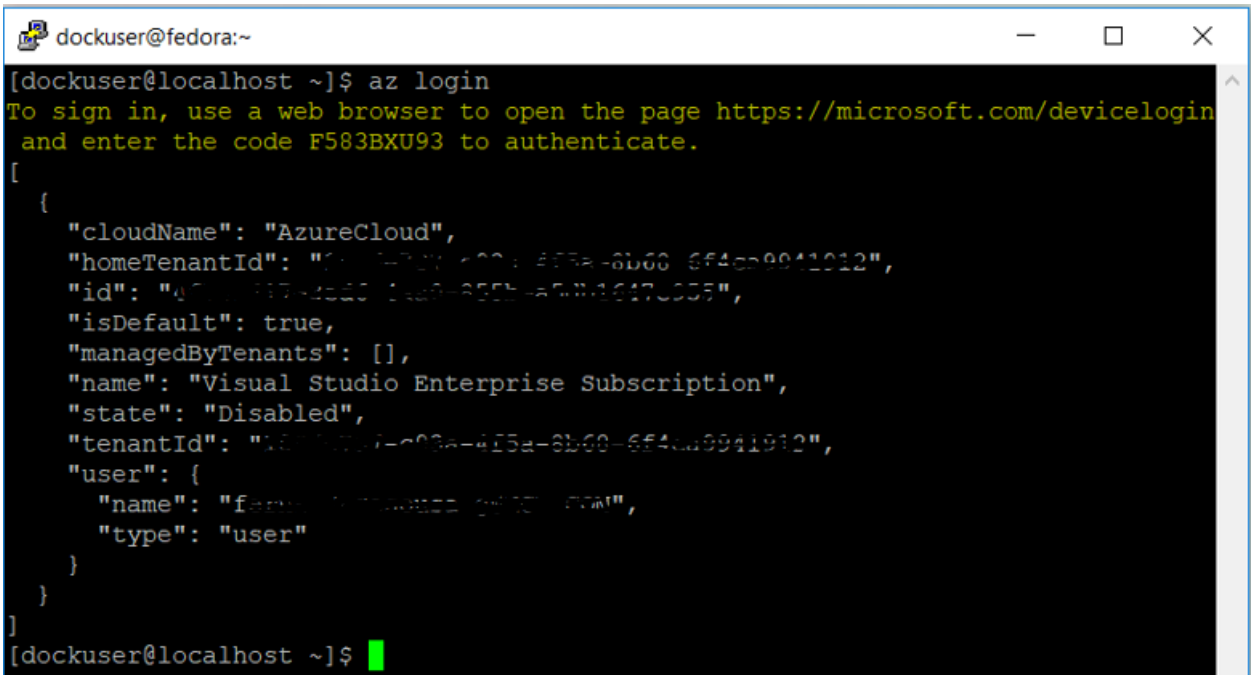
```
az login
```

If there is a browser available in your azure client, you will be redirected to a browser, so you can authenticate with your azure credentials. But, if you are using Putty, like me, you will see the following output:



Make sure you follow these directions and get yourself authenticated in azure.

Once you are authenticated, the client will capture the authentication and display the following output:



2. Create a Resource Group:

```
az group create --name <resourceGroupName> --location <region>
```

Example:

```
az group create --name aks-br-resource-grp --location brazilsouth
```

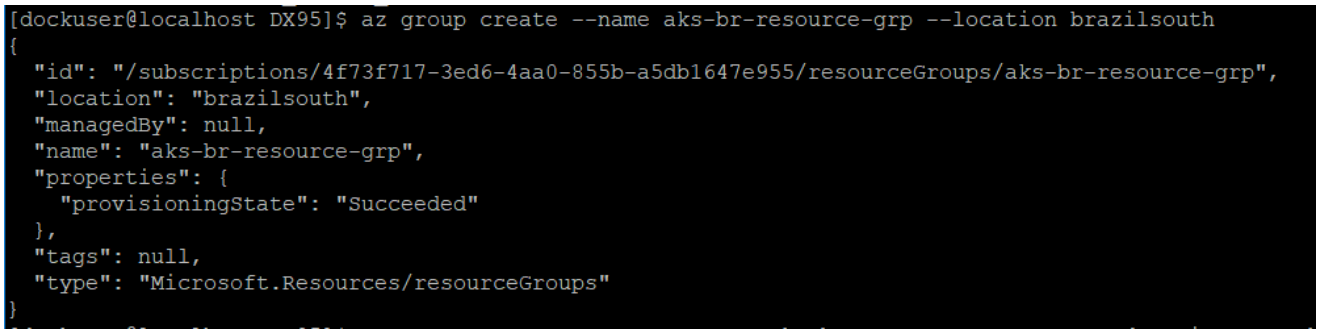
For US location:

```
az group create --name aks-resource-grp --location eastus
```

You can have access to all available locations with the following command:

```
az account list-locations
```

The result:



3. Create a container registry:

```
az acr create --resource-group <resourceGroupName> --name
<acr_registry_name> --sku Standard
```

Example:

```
az acr create --resource-group aks-br-resource-grp --name mydxregistry --
sku Standard
```

Now, you can start loading your images…

Loading, Tagging and Pushing your DX images

1- Login to the new container registry:

```
az acr login --name <containerRegistry>
```

Example:

```
az acr login --name mydxregistry
```

```
[dockuser@localhost DX95]$ az acr login --name mydxregistry
Login Succeeded
[dockuser@localhost DX95]$
```

2- Download from flexnet and copy the `hcl-dx-kubernetes-v95-CF197.zip` to you azure client machine

3- Unzip the file, in this guide, I have unzipped them under `/home/dockuser/DX95`.

4- Load all the images into docker using the command below:

```
ls -f | grep image | xargs -L 1 docker load -i
```

5- You can confirm all images have been loaded by running the "`docker images`" command:

```
[dockuser@localhost DX95]$ docker images
REPOSITORY                                  TAG                        IMAGE ID       CREATED         SIZE
hcl/dx/core                                 v95_CF197_20210806-1259    8a5abdd18684   3 days ago      6.28GB
hcl/dx/remote-search                        v95_CF197_20210806-1259    d6e161132286   4 days ago      2.26GB
hcl/dx/site-manager                         v0.3.0_20210806-1308       48cb6123441e   4 days ago      635MB
hcl/dx/cloud-operator                       v95_CF197_20210806-1310    249669b6457f   4 days ago      231MB
hcl/dx/ringapi                              v1.10.0_20210806-1311      c8341d8b4c81   4 days ago      433MB
hcl/dx/digital-asset-manager                v1.10.0_20210806-1302      5e94e5f67c4c   4 days ago      535MB
hcl/dx/postgres                             v1.10.0_20210806-1302      7565cdfbc8c6   4 days ago      526MB
hcl/dx/content-composer                     v1.10.0_20210806-1258      106ae3a1659e   4 days ago      464MB
hcl/dx/image-processor                      v1.10.0_20210806-1300      7eb5383d2f4c   4 days ago      540MB
hcl/dx/runtime-controller                   v95_CF197_20210806-1258    dbc992c4a3a1   4 days ago      503MB
hcl/dx/openldap                             v1.2.0_20210806-1258       1494f3961560   4 days ago      772MB
hcl/dx/digital-asset-management-operator    v95_CF197_20210806-1258    b6e03d5ea26b   4 days ago      229MB
hcl/dx/ambassador                           154                        c9fed6a373e5   13 months ago   355MB
hcl/dx/redis                                5.0.1                      c188f257942c   2 years ago     94.9MB
[dockuser@localhost DX95]$
```

6- To tag and push the images to your container registry (ACR), obtain the login server details:

```
az acr list --resource-group aks-br-resource-grp --query
"[].{acrLoginServer:loginServer}" --output table
```

This should be the output:

```
[dockuser@localhost DX95]$ az acr list --resource-group aks-br-resource-grp
 --query "[].{acrLoginServer:loginServer}" --output table
AcrLoginServer
----------------------
mydxregistry.azurecr.io
[dockuser@localhost DX95]$
```

7- Take note of your ACR Login server `mydxregistry.azurecr.io` you will need it.

8- Tag your images using the two commands below:

```
export REMOTE_REPO_PREFIX="<My ACR Login Server>"
```

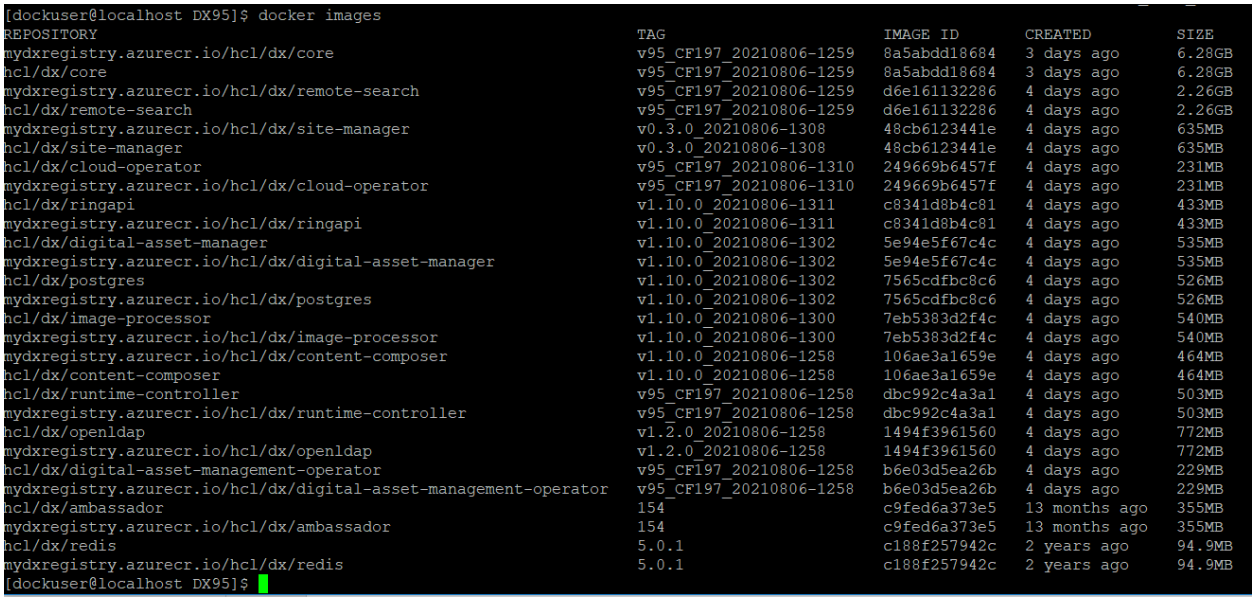
```
docker images "hcl/dx/*" | tail -n +2 | awk -F ' ' '{system("docker tag "
$1 ":" $2 " $REMOTE_REPO_PREFIX/" $1 ":" $2) }'
```

Where, *<My ACR Login Server>* is the ACR Login server hostname you have copied on step 7.

OR if you prefer, you can use the tag command as shown in the example below:

```
docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]
```

9- Once again, confirm that your images have been properly tagged with the "`docker images`" command:



```
[dockuser@localhost DX95]$ docker images
REPOSITORY                                                      TAG                      IMAGE ID       CREATED         SIZE
mydxregistry.azurecr.io/hcl/dx/core                             v95_CF197_20210806-1259  8a5abdd18684   3 days ago      6.28GB
hcl/dx/core                                                     v95_CF197_20210806-1259  8a5abdd18684   3 days ago      6.28GB
mydxregistry.azurecr.io/hcl/dx/remote-search                    v95_CF197_20210806-1259  d6e161132286   4 days ago      2.26GB
hcl/dx/remote-search                                            v95_CF197_20210806-1259  d6e161132286   4 days ago      2.26GB
mydxregistry.azurecr.io/hcl/dx/site-manager                     v0.3.0_20210806-1308     48cb6123441e   4 days ago      635MB
hcl/dx/site-manager                                             v0.3.0_20210806-1308     48cb6123441e   4 days ago      635MB
hcl/dx/cloud-operator                                           v95_CF197_20210806-1310  249669b6457f   4 days ago      231MB
mydxregistry.azurecr.io/hcl/dx/cloud-operator                   v95_CF197_20210806-1310  249669b6457f   4 days ago      231MB
hcl/dx/ringapi                                                  v1.10.0_20210806-1311    c8341d8b4c81   4 days ago      433MB
mydxregistry.azurecr.io/hcl/dx/ringapi                          v1.10.0_20210806-1311    c8341d8b4c81   4 days ago      433MB
hcl/dx/digital-asset-manager                                    v1.10.0_20210806-1302    5e94e5f67c4c   4 days ago      535MB
mydxregistry.azurecr.io/hcl/dx/digital-asset-manager            v1.10.0_20210806-1302    5e94e5f67c4c   4 days ago      535MB
hcl/dx/postgres                                                 v1.10.0_20210806-1302    7565cdfbc8c6   4 days ago      526MB
mydxregistry.azurecr.io/hcl/dx/postgres                         v1.10.0_20210806-1302    7565cdfbc8c6   4 days ago      526MB
hcl/dx/image-processor                                          v1.10.0_20210806-1300    7eb5383d2f4c   4 days ago      540MB
mydxregistry.azurecr.io/hcl/dx/image-processor                  v1.10.0_20210806-1300    7eb5383d2f4c   4 days ago      540MB
mydxregistry.azurecr.io/hcl/dx/content-composer                 v1.10.0_20210806-1258    106ae3a1659e   4 days ago      464MB
hcl/dx/content-composer                                         v1.10.0_20210806-1258    106ae3a1659e   4 days ago      464MB
hcl/dx/runtime-controller                                       v95_CF197_20210806-1258  dbc992c4a3a1   4 days ago      503MB
mydxregistry.azurecr.io/hcl/dx/runtime-controller               v95_CF197_20210806-1258  dbc992c4a3a1   4 days ago      503MB
hcl/dx/openldap                                                 v1.2.0_20210806-1258     1494f3961560   4 days ago      772MB
mydxregistry.azurecr.io/hcl/dx/openldap                         v1.2.0_20210806-1258     1494f3961560   4 days ago      772MB
hcl/dx/digital-asset-management-operator                        v95_CF197_20210806-1258  b6e03d5ea26b   4 days ago      229MB
mydxregistry.azurecr.io/hcl/dx/digital-asset-management-operator v95_CF197_20210806-1258 b6e03d5ea26b   4 days ago      229MB
hcl/dx/ambassador                                               154                      c9fed6a373e5   13 months ago   355MB
mydxregistry.azurecr.io/hcl/dx/ambassador                       154                      c9fed6a373e5   13 months ago   355MB
hcl/dx/redis                                                    5.0.1                    c188f257942c   2 years ago     94.9MB
mydxregistry.azurecr.io/hcl/dx/redis                            5.0.1                    c188f257942c   2 years ago     94.9MB
[dockuser@localhost DX95]$
```

10- Login to your container registry:

```
az acr login --name mydxregistry
```

11- Automatically push ALL your images to ACR using the following

```
docker images "$REMOTE_REPO_PREFIX/hcl/dx/*" | tail -n +2 | awk -F ' '
'{system("docker push " $1 ":" $2)}'
```

OR, if you prefer to push them manually, use the push command:

```
docker push [REMOTE_REPO_PREFIX/TAG]
```

12- Once the images are pushed, they can be listed using the commands below, or through use of the Microsoft Azure Kubernetes platform console.

Example:

```
az acr repository list --name mydxregistry --output table
```

Creating the Azure AKS Cluster

In this section, you will learn how to create a vnet, a subnet and finally the AKS Cluster.

1- Create the vnet and subnet

```
az network vnet create --resource-group <myResourceGroup> --name
<myAKSVnet> --address-prefixes 192.168.0.0/16 --subnet-name <myAKSSubnet>
--subnet-prefix 192.168.1.0/24
```

Example:

```
az network vnet create --resource-group aks-br-resource-grp --name
myAKSVnet --address-prefixes 192.168.0.0/16 --subnet-name myAKSSubnet --
subnet-prefix 192.168.1.0/24
```

2- View the vnet current configuration:

```
az network vnet list --resource-group <resourceGroupName>
```

Example:

```
az network vnet list --resource-group aks-br-resource-grp
```

3- Now, under `"subnets":` take note of the `"id"` value, in this case:

```
   "id": "/subscriptions/4f73f717-3ed6-4aa0-855b-
a5db1647e955/resourceGroups/aks-br-resource-
grp/providers/Microsoft.Network/virtualNetworks/myAKSVnet/subnets/myAKSSub
net",
```

4- Create the cluster using the copied subnet id in the `--vnet-subnet-id` parameter :

```
az aks create --resource-group aks-br-resource-grp --name myDXCluster --
node-count 2 --node-vm-size Standard_D8s_v3 --service-cidr 10.0.0.0/16 --
network-plugin kubenet --vnet-subnet-id /subscriptions/4f73f717-3ed6-4aa0-
855b-a5db1647e955/resourceGroups/aks-br-resource-
grp/providers/Microsoft.Network/virtualNetworks/myAKSVnet/subnets/myAKSSub
net --generate-ssh-keys --attach-acr mydxregistry
```

5- Install kubectl client:

```
sudo az aks install-cli
```

6- Configure `kubectl` to connect to your Kubernetes cluster using the command below. This command downloads credentials and configures the Kubernetes CLI to use them.

```
az aks get-credentials --resource-group aks-br-resource-grp --name
myDXCluster
```

7- You can see all your nodes by running this command:

```
kubectl get nodes
```

Set up the NFS server

As mentioned earlier, we will use NFS server for the container volumes, we have used these links as a reference:

https://docs.microsoft.com/en-us/azure/virtual-machines/linux/tutorial-manage-vm

https://docs.microsoft.com/en-us/azure/aks/azure-nfs-volume

1- Create a Ubuntu virtual machine on Azure using the same subnet as your AKS cluster:

```
az vm create --resource-group aks-br-resource-grp --name myNFSVM --image
UbuntuLTS --admin-username azureuser --generate-ssh-keys --vnet-name
myAKSVnet --subnet myAKSSubnet
```

It may take a few minutes to create the VM. Once the VM has been created, the Azure CLI outputs information about the VM.

Take note of the `publicIpAddress`, this address will be used to access the virtual machine:

2- Connect to your NFS VM using the default user “azureuser”:

```
ssh azureuser@<publicIpAddress>
```

Example:

```
ssh azureuser@191.233.143.113
```

3- Create a script called “`nfs-server-setup.sh`”

```
sudo vi nfs-server-setup.sh
```

4- Copy the following content to this file (this is the script to set up an NFS Server within your Ubuntu virtual machine):

```
#!/bin/bash

# This script should be executed on Linux Ubuntu Virtual Machine

DATA_DIRECTORY=${1:-/nfsshare}
```

```
AKS_SUBNET=${2:-*}

echo "Updating packages"
apt-get -y update

echo "Installing NFS kernel server"

apt-get -y install nfs-kernel-server

echo "Making data directory ${DATA_DIRECTORY}"
mkdir -p ${DATA_DIRECTORY}

echo "Giving 777 permissions to ${DATA_DIRECTORY} directory"
chmod 777 ${DATA_DIRECTORY}

echo "Appending localhost and Kubernetes subnet address ${AKS_SUBNET} to
exports configuration file"
echo "/nfsshare
${AKS_SUBNET}(rw,sync,no_root_squash,no_all_squash,no_wdelay,insecure)" >>
/etc/exports
echo "/nfsshare
localhost(rw,sync,no_root_squash,no_all_squash,no_wdelay,insecure)" >>
/etc/exports

nohup service nfs-kernel-server restart
```

5- Save the file and set execution permission via the command:

```
sudo chmod +x ~/nfs-server-setup.sh
```

6- You can ssh into the VM and execute it via the command:

```
sudo ./nfs-server-setup.sh
```

7- Check that the server is started:

```
sudo systemctl status nfs-server
```

8- Create the folders in your NFS server:

```
sudo mkdir /nfsshare/volumes_os
sudo mkdir /nfsshare/volumes_os/wp_profile
sudo mkdir /nfsshare/volumes_os/dam
sudo chmod 777 -R /nfsshare/
```

9- Disconnect from your NFS server:

```
exit
```

Connecting AKS Cluster to NFS Server

Connecting the two services in the same or peered virtual networks is necessary.

https://docs.microsoft.com/en-us/azure/aks/configure-kubenet#create-an-aks-cluster-in-the-virtual-network

1- On your azure client machine, create a file storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: dx-deploy-stg
provisioner: 192.168.1.6/nfs
```

PS: Where *privisioner* is the NFS´s private IP address/nfs

2- On your azure client machine, create a file `pv_wp_profile.yaml`

3- Add the following to this file:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: wp-profile
spec:
  capacity:
    storage: 100Gi
  accessModes:
  - ReadWriteMany
  nfs:
    path: /nfsshare/volumes_os/wp_profile
    server: 192.168.1.6
  persistentVolumeReclaimPolicy: Retain
  storageClassName: dx-deploy-stg
  mountOptions:
    - hard
    - nfsvers=4.1
    - rsize=10485760
    - wsize=10485760
```

```
    - timeo=600
    - retrans=2
    - noresvport
```

4- Create a pv for DAM: `pv_dam.yaml`

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: dam-pv
spec:
  capacity:
    storage: 100Gi
  accessModes:
  - ReadWriteMany
  nfs:
    path: /nfsshare/volumes_os/dam
    server: 192.168.1.6
  persistentVolumeReclaimPolicy: Retain
  storageClassName: dx-deploy-stg
  mountOptions:
    - hard
    - nfsvers=4.1
    - rsize=10485760
    - wsize=10485760
    - timeo=600
    - retrans=2
    - noresvport
```

5- Run the yaml file like this:

```
kubectl apply -f storageclass.yaml
kubectl apply -f pv_wp_profile.yaml
kubectl apply -f pv_dam.yaml
```

Generate TLS Certificate

Create a TLS certification to be used by the deployment.

1- Create your namespace using kubectl command:

```
kubectl create ns dxns
```

2- Install openssl if you haven't done it already:

```
sudo dnf install openssl-1:1.1.1k-1.fc33.x86_64
```

3- Prior to this step, create a Self-Signed Certificate to enable HTTPS using the following command:

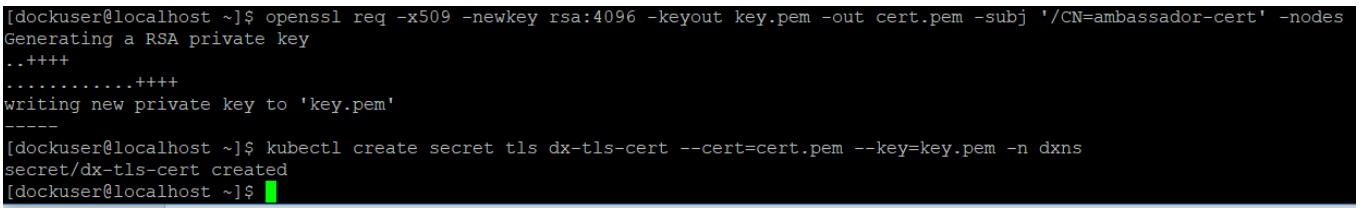
```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -subj
'/CN=ambassador-cert' -nodes
```

4- Then, store the Certificate and Key in a Kubernetes Secret using the following command:

```
kubectl create secret tls dx-tls-cert --cert=cert.pem --key=key.pem -n
<YourNamespace>
```

Example:

```
kubectl create secret tls dx-tls-cert --cert=cert.pem --key=key.pem -n dxns
```



Install HELM on Linux Ubuntu

1- Install HELM on your local machine

```
wget https://get.helm.sh/helm-v3.6.3-linux-amd64.tar.gz
```

2- The terminal prints out a confirmation message when the download completes.

```
[dockuser@localhost hcl-dx-deployment]$ wget https://get.helm.sh/helm-v3.6.3-lin
ux-amd64.tar.gz
--2021-07-21 16:11:34--  https://get.helm.sh/helm-v3.6.3-linux-amd64.tar.gz
Resolving get.helm.sh (get.helm.sh)... 152.195.19.97, 2606:2800:11f:1cb7:261b:1f
9c:2074:3c
Connecting to get.helm.sh (get.helm.sh)|152.195.19.97|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13702117 (13M) [application/x-tar]
Saving to: 'helm-v3.6.3-linux-amd64.tar.gz'

helm-v3.6.3-linux-a 100%[===================>]  13.07M  5.91MB/s    in 2.2s

2021-07-21 16:11:37 (5.91 MB/s) - 'helm-v3.6.3-linux-amd64.tar.gz' saved [137021
17/13702117]
```

3- Next, unpack the Helm file using the tar command:

```
tar -xvf helm-v3.6.3-linux-amd64.tar.gz
```

4- Move the linux-amd64/helm file to the /usr/local/bin directory:

```
sudo mv linux-amd64/helm /usr/local/bin
```

5- You may remove these file and folder to clean up space:

```
rm helm-v3.6.3-linux-amd64.tar.gz
```

```
rm -rf linux-amd64
```

6- Finally, verify you have successfully installed Helm by checking the version of the software:

```
helm version
```

The terminal prints out the version like this:

```
[dockuser@localhost DX95_CF196]$ helm version
version.BuildInfo{Version:"v3.6.3", GitCommit:"d506314abfb5d21419df8c7e7e6801237
9db2354", GitTreeState:"clean", GoVersion:"go1.16.5"}
```

Deploy DX using HELM Chart

1- If you haven't done it already, log in to the Cluster

```
az login
```

2- Go to the directory where you have downloaded all DX images, example `/home/dockuser/DX95`

3- Copy the name of your HCL DX 9.5 Container deployment file, in this case it is:

```
hcl-dx-deployment-v2.0.0_20210806-1300.tgz
```

4- Extract the default HCL DX 9.5 Container `values.yaml` file and name it “custom_values.yaml” using the following commands:

```
helm show values hcl-dx-deployment-v2.0.0_20210806-1300.tgz >
custom_values.yaml
```

5- Update the `custom_values.yaml` file with the following values, leave the rest unchanged:

```
images:
  repository: "mydxregistry.azurecr.io"
  names:
    contentComposer: "hcl/dx/content-composer"
    core: "hcl/dx/core"
    designStudio: "hcl/dx/site-manager"
    digitalAssetManagement: "hcl/dx/digital-asset-manager"
    imageProcessor: "hcl/dx/image-processor"
    openLdap: "hcl/dx/openldap"
    persistence: "hcl/dx/postgres"
    remoteSearch: "hcl/dx/remote-search"
    ringApi: "hcl/dx/ringapi"
    ambassadorIngress: "hcl/dx/ambassador"
    ambassadorRedis: "hcl/dx/redis"
    runtimeController: "hcl/dx/runtime-controller"
  tags:
    contentComposer: "v1.10.0_20210806-1258"
    core: "v95_CF197_20210806-1259"
    designStudio: "v95_CF197_20210806-1259"
    digitalAssetManagement: "v1.10.0_20210806-1302"
    imageProcessor: "v1.10.0_20210806-1300"
    openLdap: "v1.2.0_20210806-1258"
    persistence: "v1.10.0_20210806-1302"
    remoteSearch: "v95_CF197_20210806-1259"
    ringApi: "v1.10.0_20210806-1311"
```

```
    ambassadorIngress: "154"
    ambassadorRedis: "5.0.1"
    runtimeController: "v95_CF197_20210806-1258"
# Persistent Volume Setup
volumes:
  core:
    # Shared profile PVC shared by all Core pods - RWX
    profile:
      storageClassName: "dx-deploy-stg"
      requests:
        storage: "10Gi"
      # Optional volume name to specifically map to
      volumeName: "wp-profile"
    # Transaction Log PVC, one per Core pod - RWO
    tranlog:
      storageClassName: "default"
      requests:
        storage: "50Mi"
      # Optional volume name to specifically map to
      volumeName:
    # Application Log PVC, one per Core pod - RWO
    log:
      storageClassName: "default"
      requests:
        storage: "250Mi"
      # Optional volume name to specifically map to
      volumeName:
  # Persistent Volumes for Digital Asset Management
  digitalAssetManagement:
    binaries:
      storageClassName: "dx-deploy-stg"
      requests:
```

```
        storage: "2Gi"


      volumeName: "dam-pv"
# Persistent Volumes for Persistence
  persistence:
    # Database PVC, one per Persistence pod - RWO
    database:
      storageClassName: "default"
      requests:
        storage: "2Gi"
      # Optional volume name to specifically map to, RW and RO share the
same volume name, suffixed by -rw and -ro
      volumeName:
  # Persistent Volumes for Open LDAP
  openLdap:
    # slapd directory PVC, one per Open LDAP pod - RWO
    slapd:
      storageClassName: "default"
      requests:
        storage: "100Mi"
      # Optional volume name to specifically map to
      volumeName:
    # certificate directory, on per Open LDAP pod - RWO
    certificate:
      storageClassName: "default"
      requests:
        storage: "100Mi"
      # Optional volume name to specifically map to
      volumeName:
    # ldap directory PVC, one per Open LDAP pod - RWO
    ldap:
      storageClassName: "default"
```

```
      requests:
        storage: "10Gi"
        # Optional volume name to specifically map to
      volumeName:
  # Persistent Volumes for Remote Search
  remoteSearch:
    # Remote Search profile PVC, one per Remote Search pod - RWO
    prsprofile:
      storageClassName: "default"
      requests:
        storage: "10Gi"
      # Optional volume name to specifically map to
      volumeName:
```

6- Run Helm install command:

```
helm install -n <namespace> -f path/to/your/custom_values.yaml your-release-name path/to/hcl-dx-deployment-vX.X.X_XXXXXXXX-XXXX.tar.gz
```

Example:

```
helm install -n dxns -f custom_values.yaml dx hcl-dx-deployment-v2.0.0_20210806-1300.tgz
```

7- Validate the deployment:

```
kubectl get pods -n dxns
```

```
kubectl get pv -n dxns
```

```
[dockuser@localhost DX95_CF196]$ kubectl get pv -n dxns
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                                      STORAGECLASS    REASON   AGE
dam-pv                                     100Gi      RWX            Retain           Bound    dxns/dx-digital-asset-management           dx-deploy-stg            25h
pvc-055bcdb7-07cc-4a59-aed7-ee433e358435   1Gi        RWO            Delete           Bound    dxns/log-dx-core-0                         default                  7m28s
pvc-1685ee3b-c4a0-4d62-ba95-68d6c27ee5fa   1Gi        RWO            Delete           Bound    dxns/tranlog-dx-core-0                     default                  7m32s
pvc-2fbbd7ce-44de-41bf-908e-b0a26c1b62cf   10Gi       RWO            Delete           Bound    dxns/prsprofile-dx-remote-search-0         default                  7m32s
pvc-47031b5d-bb7b-4f96-8cd0-27ce506cc5c8   2Gi        RWO            Delete           Bound    dxns/database-dx-persistence-ro-0          default                  7m33s
pvc-48617539-d8b2-4b0f-bdb8-824d61b6d863   10Gi       RWO            Delete           Bound    dxns/ldap-dx-open-ldap-0                   default                  7m32s
pvc-56c30ee3-3494-4a47-a5fb-0f9d51bfc644   2Gi        RWO            Delete           Bound    dxns/database-dx-persistence-rw-0          default                  7m32s
pvc-640424cc-f1cb-488b-8a6a-efc5d004f4ba   1Gi        RWO            Delete           Bound    dxns/certificate-dx-open-ldap-0            default                  7m33s
pvc-6709d482-d614-42e6-a5ac-cb265ad3e840   1Gi        RWO            Delete           Bound    dxns/slapd-dx-open-ldap-0                  default                  7m27s
wp-profile                                 100Gi      RWX            Retain           Bound    dxns/dx-core-profile                       dx-deploy-stg            25h
[dockuser@localhost DX95_CF196]$
```

```
kubectl get pvc -n dxns
```

```
[dockuser@localhost DX95_CF196]$ kubectl get pvc -n dxns
NAME                                STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS    AGE
certificate-dx-open-ldap-0          Bound    pvc-640424cc-f1cb-488b-8a6a-efc5d004f4ba   1Gi        RWO            default         8m1s
database-dx-persistence-ro-0        Bound    pvc-47031b5d-bb7b-4f96-8cd0-27ce506cc5c8   2Gi        RWO            default         8m1s
database-dx-persistence-rw-0        Bound    pvc-56c30ee3-3494-4a47-a5fb-0f9d51bfc644   2Gi        RWO            default         8m1s
dx-core-profile                     Bound    wp-profile                                 100Gi      RWX            dx-deploy-stg   8m2s
dx-digital-asset-management         Bound    dam-pv                                     100Gi      RWX            dx-deploy-stg   8m2s
ldap-dx-open-ldap-0                 Bound    pvc-48617539-d8b2-4b0f-bdb8-824d61b6d863   10Gi       RWO            default         8m1s
log-dx-core-0                       Bound    pvc-055bcdb7-07cc-4a59-aed7-ee433e358435   1Gi        RWO            default         8m1s
prsprofile-dx-remote-search-0       Bound    pvc-2fbbd7ce-44de-41bf-908e-b0a26c1b62cf   10Gi       RWO            default         8m1s
slapd-dx-open-ldap-0                Bound    pvc-6709d482-d614-42e6-a5ac-cb265ad3e840   1Gi        RWO            default         8m1s
tranlog-dx-core-0                   Bound    pvc-1685ee3b-c4a0-4d62-ba95-68d6c27ee5fa   1Gi        RWO            default         8m1s
[dockuser@localhost DX95_CF196]$
```

PS: If the status of the pod is stuck in Pending, verify that the pv and pvc are correctly created/bounded.

8- Make sure all the pods are "Running" and in "Ready" state on your Microsoft Azure AKS platform, as shown in the example below:

```
kubectl get pod -n dxns
```

```
[dockuser@localhost DX95_CF196]$ kubectl get pods -n dxns
NAME                                    READY   STATUS    RESTARTS   AGE
dx-ambassador-5d9f5d7b5d-bm4fc          1/1     Running   0          16m
dx-ambassador-5d9f5d7b5d-j484m          1/1     Running   0          16m
dx-ambassador-5d9f5d7b5d-x6zf9          1/1     Running   0          16m
dx-ambassador-redis-8bd77f764-jnwst     1/1     Running   0          16m
dx-ambassador-redis-8bd77f764-tthtf     1/1     Running   0          16m
dx-ambassador-redis-8bd77f764-znntx     1/1     Running   0          16m
dx-content-composer-bfb495b77-4s2d7     1/1     Running   0          16m
dx-core-0                               1/1     Running   0          16m
dx-design-studio-b56d4dccc-756n6        1/1     Running   0          16m
dx-digital-asset-management-0           1/1     Running   3          16m
dx-image-processor-798444d79c-bccmw     1/1     Running   0          16m
dx-open-ldap-0                          1/1     Running   0          16m
dx-persistence-ro-0                     1/1     Running   0          16m
dx-persistence-rw-0                     1/1     Running   0          16m
dx-remote-search-0                      1/1     Running   0          16m
dx-ring-api-7bf486dd45-4vzr6            1/1     Running   0          16m
dx-runtime-controller-bc8c9f4cb-9b8dg   1/1     Running   0          16m
```

Update the hostname of your DX deploy

Since we didn´t know the Ambassador Ingress hostname beforehand, we will now run this additional step, which will retrieve the assigned hostname from the Ambassador Ingress and configure all applications accordingly:

1- Run the helm command to upgrade your deployment:

```
helm upgrade -n <namespace> -f path/to/your/custom_values.yaml your-
release-name path/to/hcl-dx-deployment-vX.X.X_XXXXXXXX-XXXX.tar.gz
```

Example:

```
helm upgrade -n dxns -f custom_values.yaml dx hcl-dx-deployment-
v2.0.0_20210806-1300.tgz
```

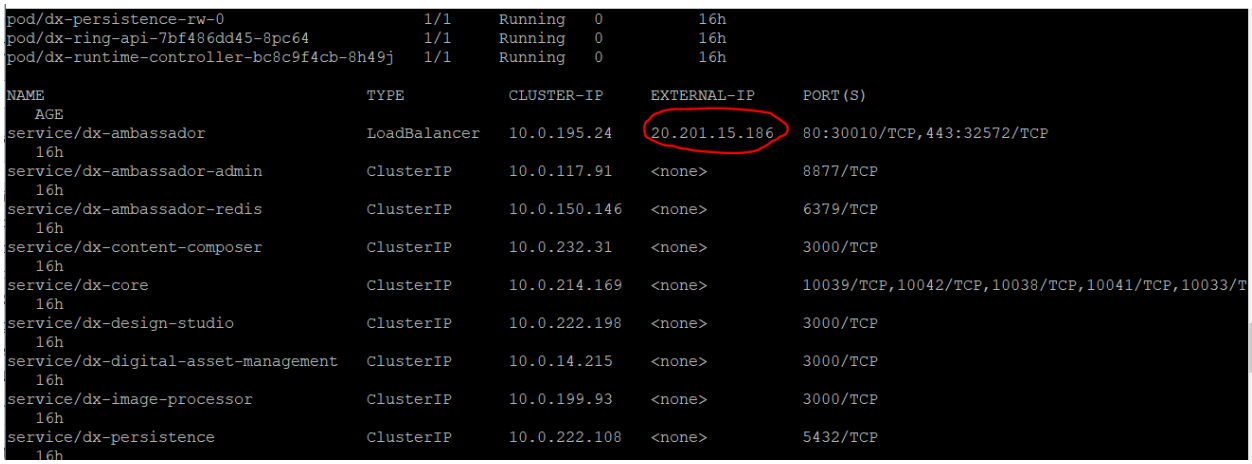
2- Validate the deployment once again, make sure all the pods are "Running" and in "Ready" state on your Microsoft Azure AKS platform:

```
kubectl get pods -n dxns

kubectl get pv -n dxns

kubectl get pvc -n dxns
```

Testing your Portal deployment

1. Afterwards, access the HCL DX 9.5 CF_197 container deployment. To do so, obtain the external IP from the container platform Load balancer to access the HCL DX 9.5 deployment, as shown in the example below:

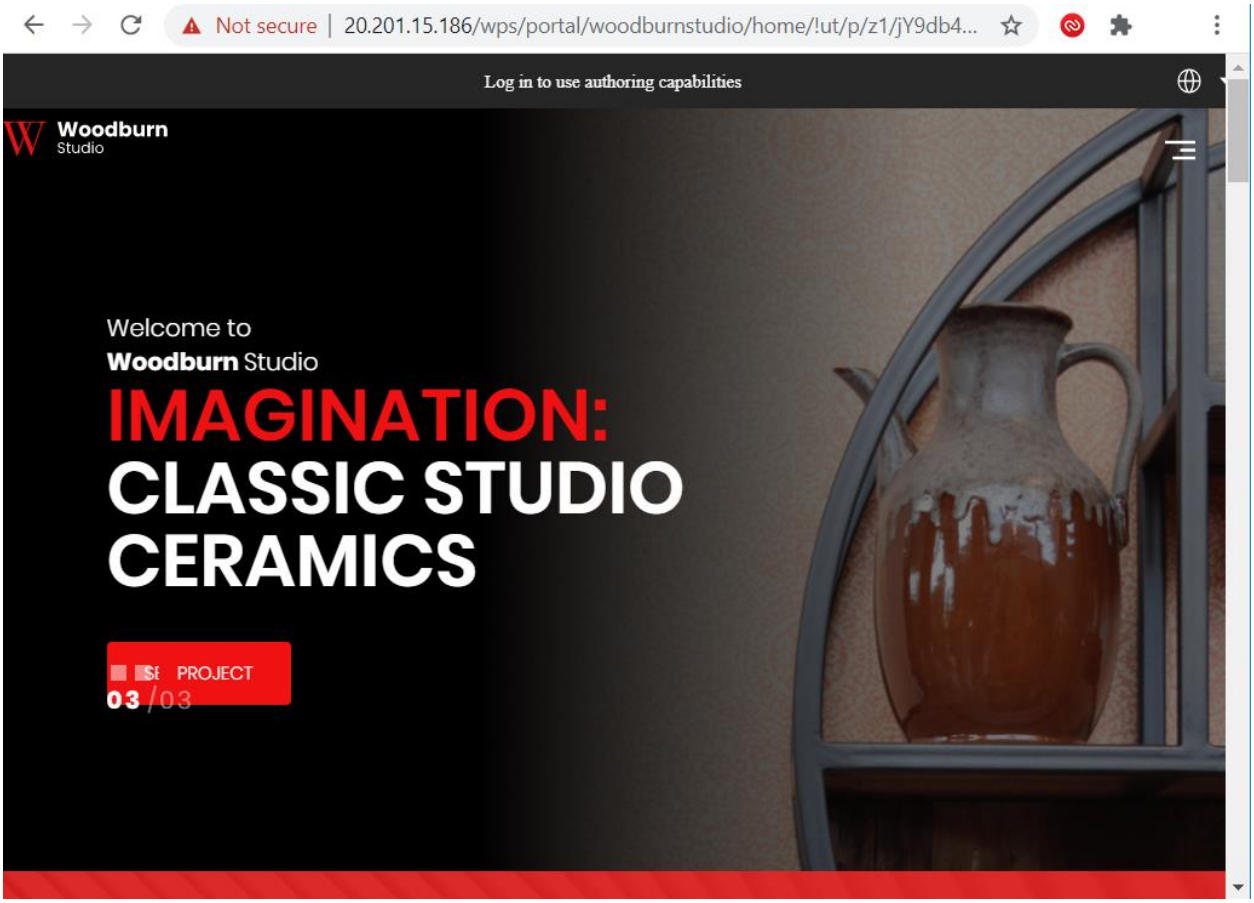
   ```
   kubectl get all -n dxns
   ```



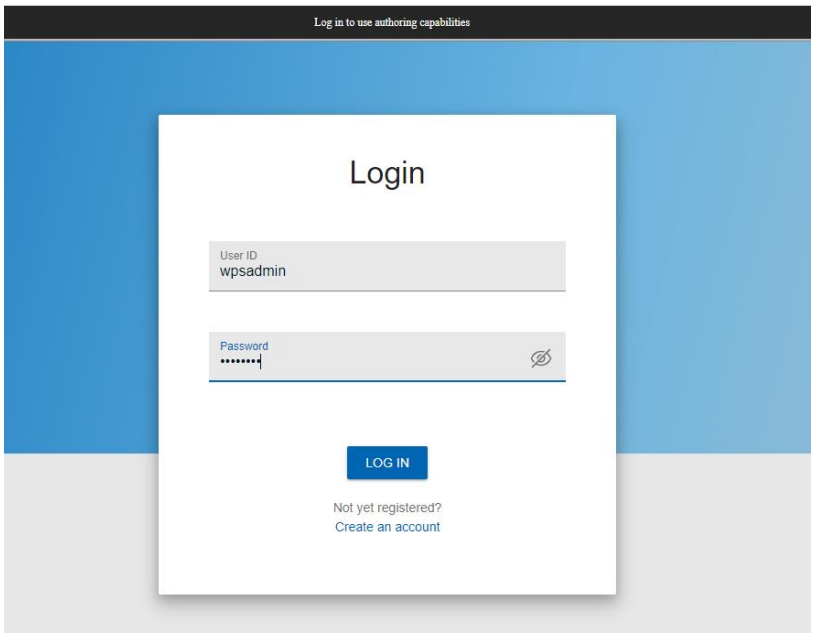
```
pod/dx-persistence-rw-0                    1/1     Running   0          16h
pod/dx-ring-api-7bf486dd45-8pc64           1/1     Running   0          16h
pod/dx-runtime-controller-bc8c9f4cb-8h49j  1/1     Running   0          16h

NAME                                TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)
   AGE
service/dx-ambassador               LoadBalancer   10.0.195.24    20.201.15.186   80:30010/TCP,443:32572/TCP
   16h
service/dx-ambassador-admin         ClusterIP      10.0.117.91    <none>          8877/TCP
   16h
service/dx-ambassador-redis         ClusterIP      10.0.150.146   <none>          6379/TCP
   16h
service/dx-content-composer         ClusterIP      10.0.232.31    <none>          3000/TCP
   16h
service/dx-core                     ClusterIP      10.0.214.169   <none>          10039/TCP,10042/TCP,10038/TCP,10041/TCP,10033/T
   16h
service/dx-design-studio            ClusterIP      10.0.222.198   <none>          3000/TCP
   16h
service/dx-digital-asset-management ClusterIP      10.0.14.215    <none>          3000/TCP
   16h
service/dx-image-processor          ClusterIP      10.0.199.93    <none>          3000/TCP
   16h
service/dx-persistence              ClusterIP      10.0.222.108   <none>          5432/TCP
   16h
```

2. Access your Portal using that URL:

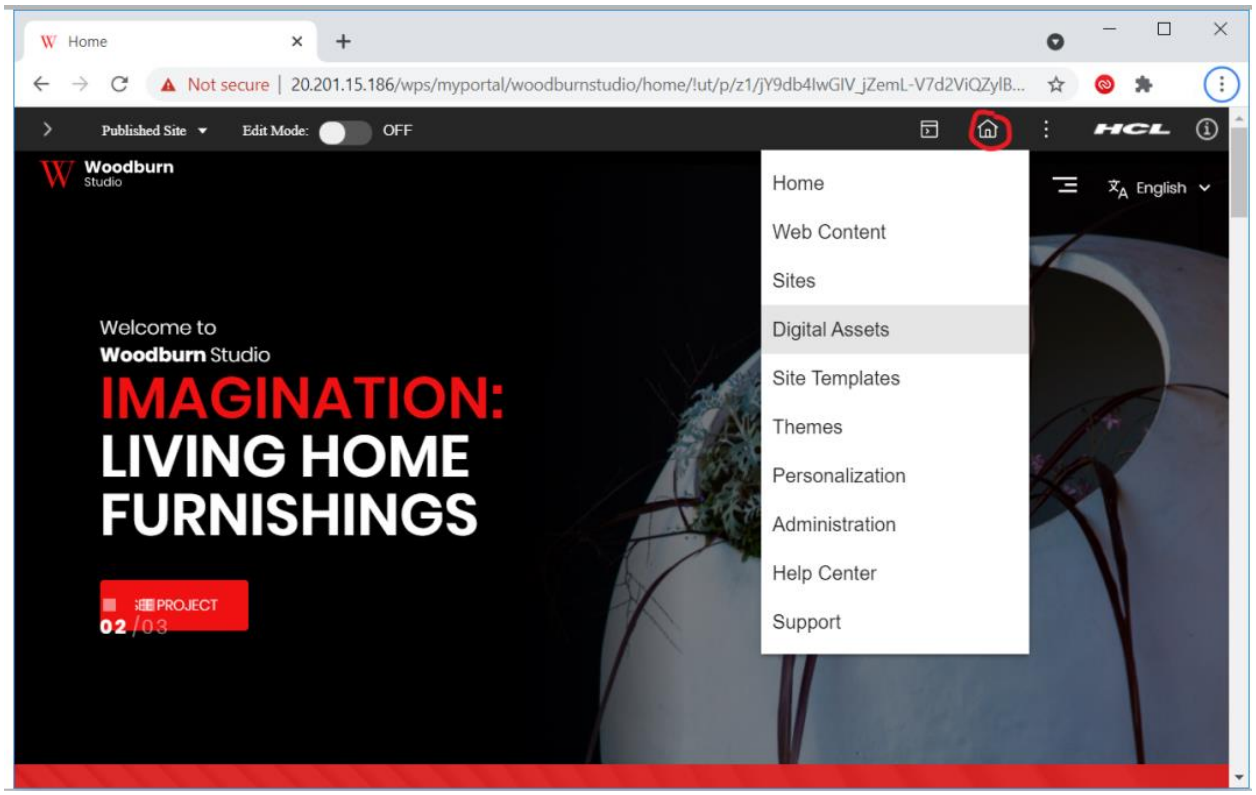
   https://<external-ip>/wps/myportal



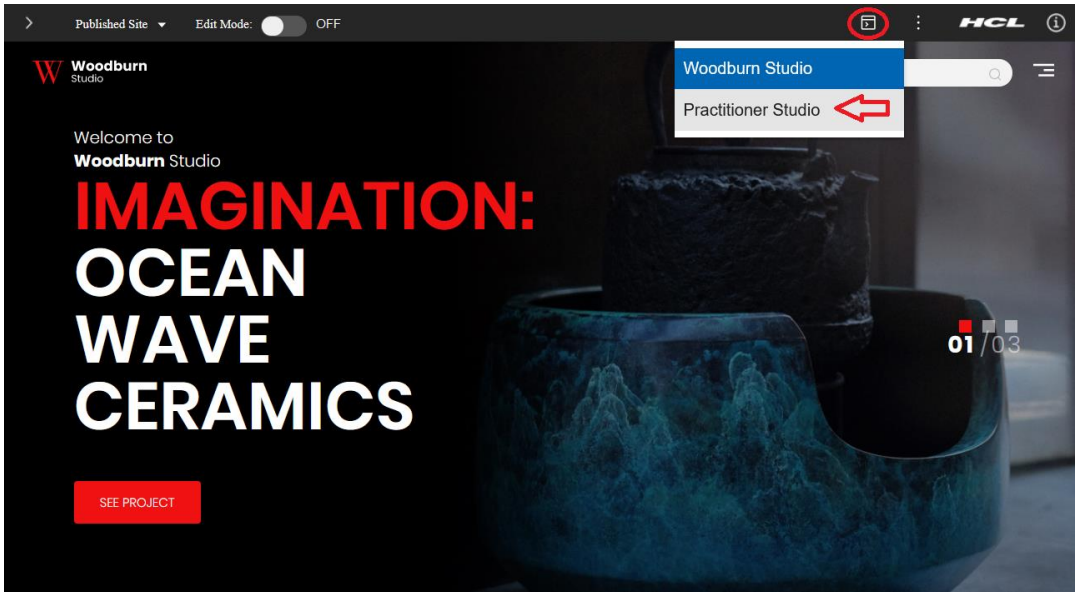
3. Authenticate as your Portal Administrator:



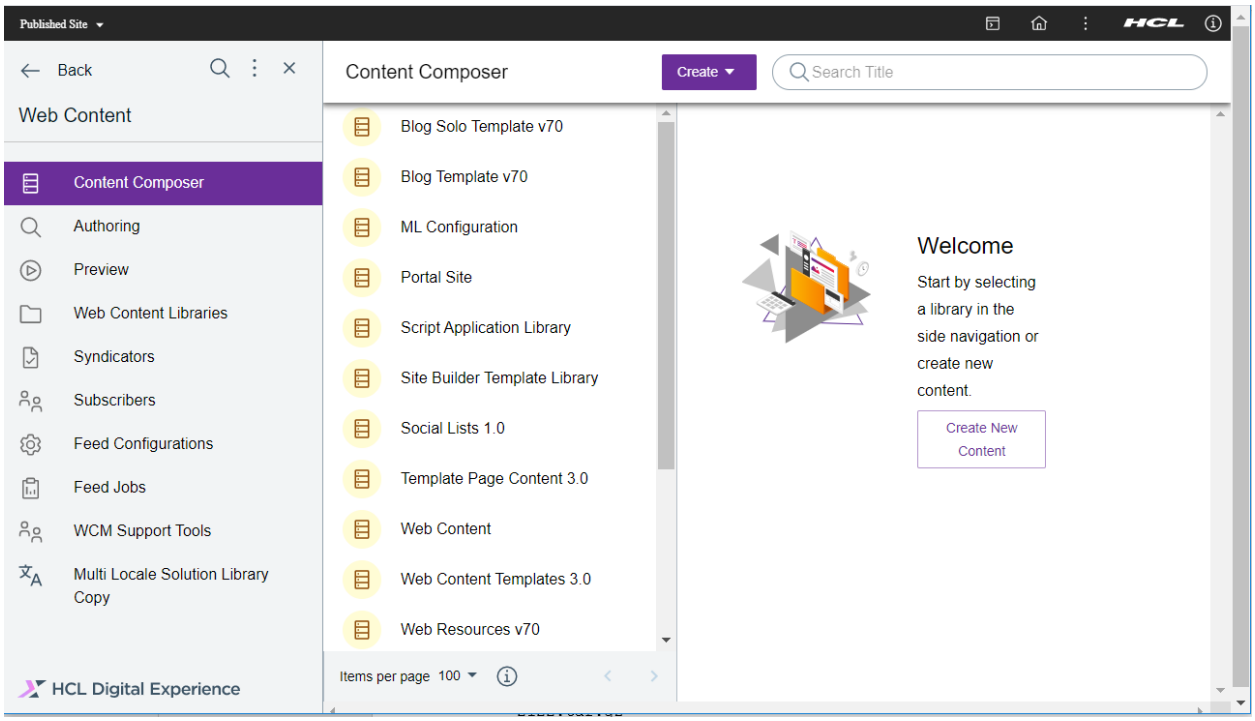
4. You can access DAM by clicking on “Open Applications Menu” and clicking on “Digital Assets”:



5. Content Composer will be available under **Practitioner Studio**



6. Select **Web Content >> Content Composer** page:



Congratulations! You have successfully deployed HCL Digital Experience 9.5 CF_196 + Digital Asset Manager + Content Composer on Microsoft Azure AKS using HELM.