HCL SOFTWARE

HCL Digital Experience

Step-by-Step Guide

How to deploy DX CF_194 with DAM and CC on Azure AKS

Author:

Fernanda de Sousa Gomes

HCL Digital Experience L2 Support

HCL Technologies

Contents

What you will find in this guide

This guide will show you how to deploy HCL Digital Experience 9.5 CF_194 + Digital Asset Manager + Content Composer on Azure AKS using dxctl in NFS volumes.

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

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:

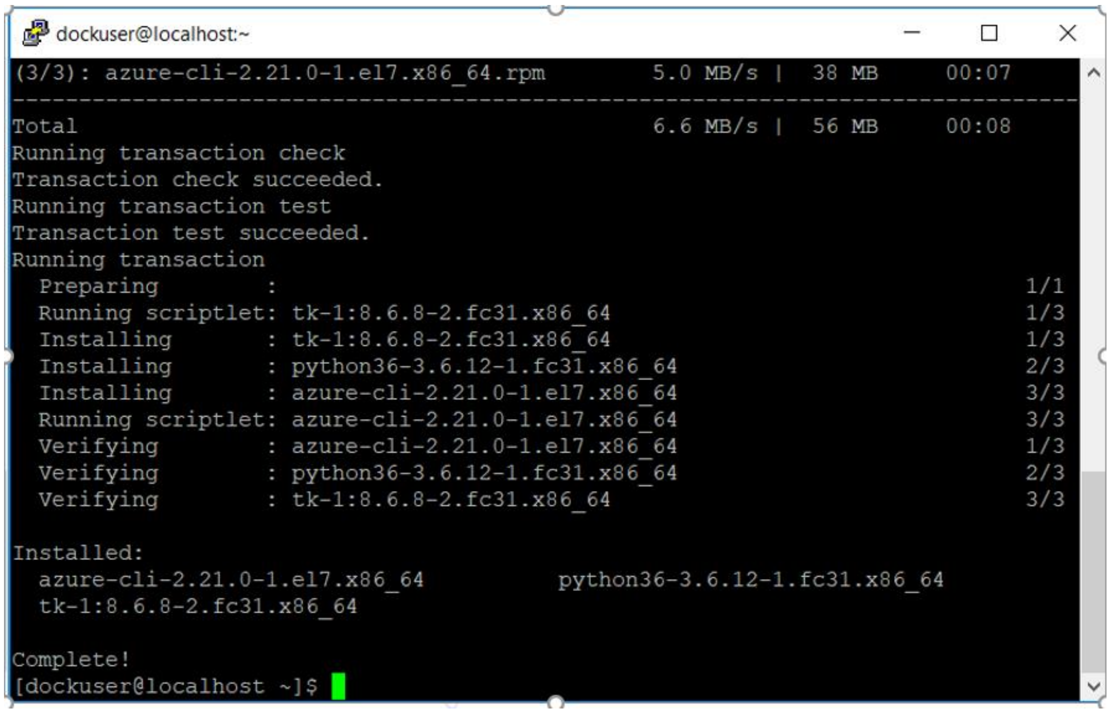
```
echo -e "[azure-cli]

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:



```
(3/3): azure-cli-2.21.0-1.el7.x86_64.rpm        5.0 MB/s |  38 MB     00:07
--------------------------------------------------------------------------------
Total                                           6.6 MB/s |  56 MB     00:08
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Running scriptlet: tk-1:8.6.8-2.fc31.x86_64                               1/3
  Installing       : tk-1:8.6.8-2.fc31.x86_64                               1/3
  Installing       : python36-3.6.12-1.fc31.x86_64                          2/3
  Installing       : azure-cli-2.21.0-1.el7.x86_64                          3/3
  Running scriptlet: azure-cli-2.21.0-1.el7.x86_64                          3/3
  Verifying        : azure-cli-2.21.0-1.el7.x86_64                          1/3
  Verifying        : python36-3.6.12-1.fc31.x86_64                          2/3
  Verifying        : tk-1:8.6.8-2.fc31.x86_64                               3/3

Installed:
  azure-cli-2.21.0-1.el7.x86_64          python36-3.6.12-1.fc31.x86_64
  tk-1:8.6.8-2.fc31.x86_64

Complete!
[dockuser@localhost ~]$
```

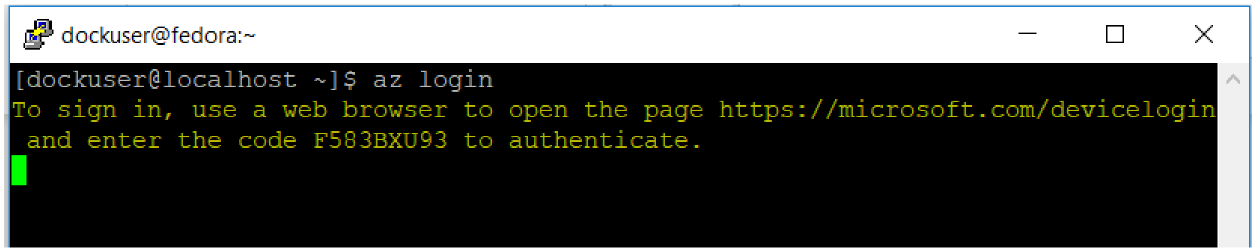
Creating a Resource Group and Container Registry

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:

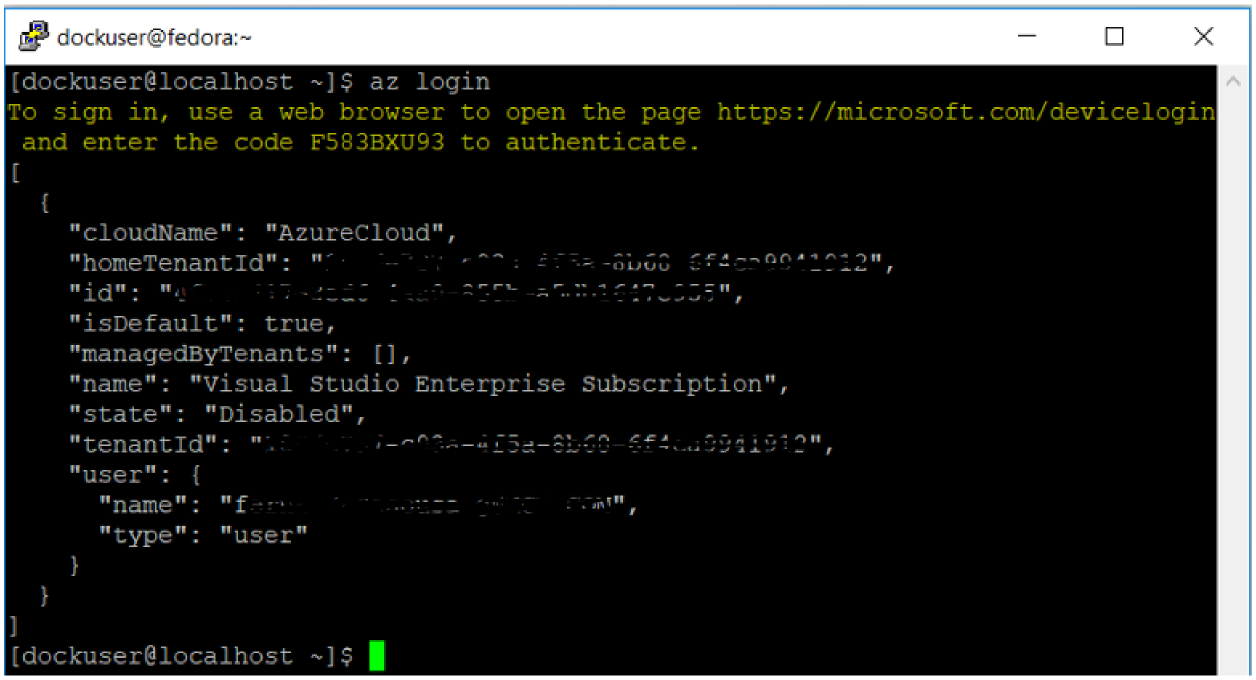


dockuser@fedora:~

```
[dockuser@localhost ~]$ az login
To sign in, use a web browser to open the page https://microsoft.com/devicelogin
 and enter the code F583BXU93 to authenticate.
```

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:



dockuser@fedora:~

```
[dockuser@localhost ~]$ az login
To sign in, use a web browser to open the page https://microsoft.com/devicelogin
 and enter the code F583BXU93 to authenticate.
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "[redacted]",
    "id": "[redacted]",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Visual Studio Enterprise Subscription",
    "state": "Disabled",
    "tenantId": "[redacted]",
    "user": {
      "name": "[redacted]",
      "type": "user"
    }
  }
]
[dockuser@localhost ~]$
```

2. Create a Resource Group:

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

Example:

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

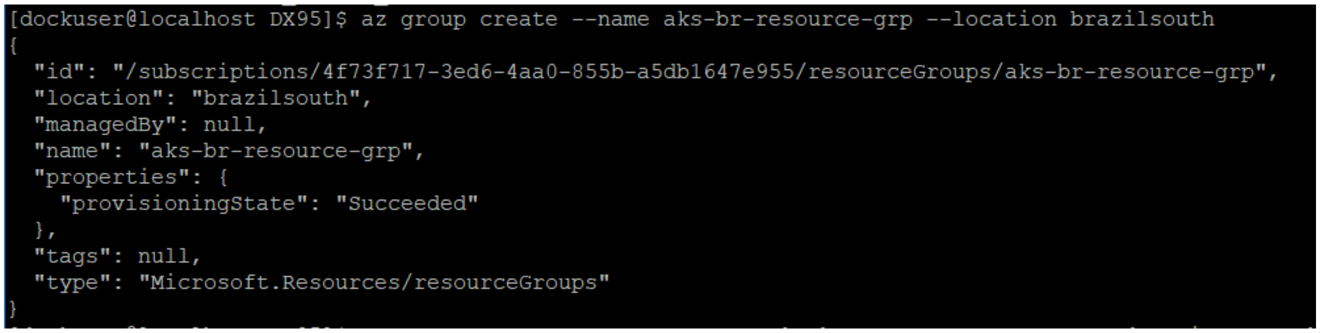
For US location:

```
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:



```
[dockuser@localhost DX95]$ az group create --name aks-br-resource-grp --location brazilsouth
{
  "id": "/subscriptions/4f73f717-3ed6-4aa0-855b-a5db1647e955/resourceGroups/aks-br-resource-grp",
  "location": "brazilsouth",
  "managedBy": null,
  "name": "aks-br-resource-grp",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
}
```

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…

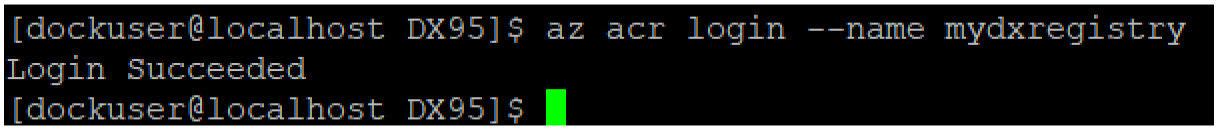
Loading, Tagging and Pushing your DX images

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-CF194.zip` to you azure client machine

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

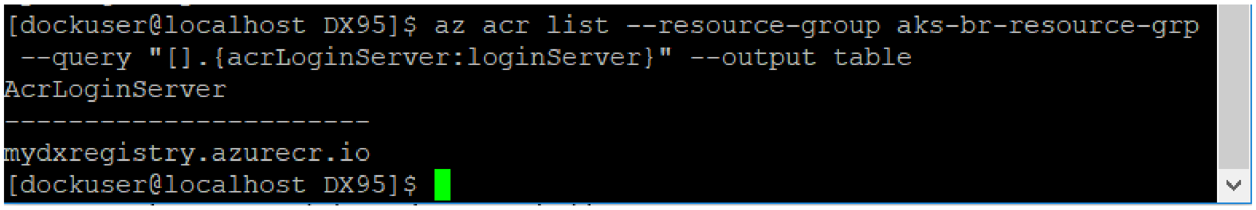
4- Load the images into docker, for now, we are only loading these 4 images:

```
docker load -i hcl-dx-core-image-v95_CF194_20210415-2120.tar.gz

docker load -i hcl-dx-ambassador-image-154.tar.gz

docker load -i hcl-dx-cloud-operator-image-v95_CF194_20210416-0233.tar.gz

docker load -i hcl-dx-redis-image-5.0.1.tar.gz

docker load -i hcl-dx-content-composer-image-v1.7.0_20210415-2121.tar.gz

docker load -i hcl-dx-digital-asset-management-operator-image-
v95_CF194_20210415-2127.tar.gz

docker load -i hcl-dx-digital-asset-manager-image-v1.7.0_20210415-
2122.tar.gz

docker load -i hcl-dx-image-processor-image-v1.7.0_20210415-2120.tar.gz

docker load -i hcl-dx-postgres-image-v1.7.0_20210415-2120.tar.gz

docker load -i hcl-dx-ringapi-image-v1.7.0_20210415-2122.tar.gz
```

5- 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:



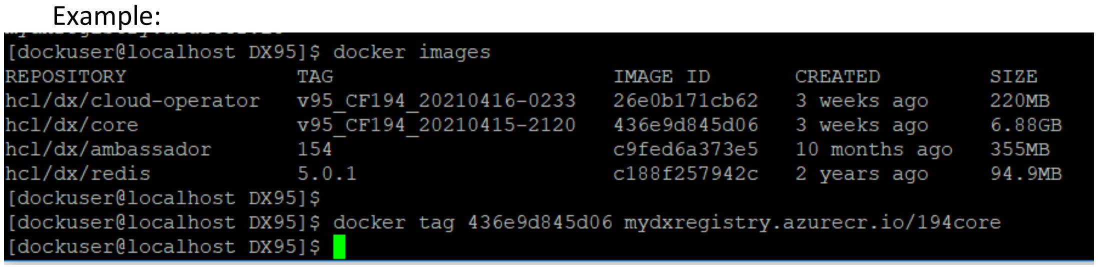
```
[dockuser@localhost DX95]$ az acr list --resource-group aks-br-resource-grp
 --query "[].{acrLoginServer:loginServer}" --output table
AcrLoginServer
-----------------------
mydxregistry.azurecr.io
[dockuser@localhost DX95]$
```

6- Take note of your ACR Login server `mydxregistry.azurecr.io` you will need it.
7- Tag your images using the `tag` command as shown in the examples below:

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

Tip: You can collect your IMAGE ID using the "docker images" command, and then, use the id in your tag command.

Example:



```
[dockuser@localhost DX95]$ docker images
REPOSITORY               TAG                        IMAGE ID       CREATED         SIZE
hcl/dx/cloud-operator    v95_CF194_20210416-0233    26e0b171cb62   3 weeks ago     220MB
hcl/dx/core              v95_CF194_20210415-2120    436e9d845d06   3 weeks ago     6.88GB
hcl/dx/ambassador        154                        c9fed6a373e5   10 months ago   355MB
hcl/dx/redis             5.0.1                      c188f257942c   2 years ago     94.9MB
[dockuser@localhost DX95]$
[dockuser@localhost DX95]$ docker tag 436e9d845d06 mydxregistry.azurecr.io/194core
[dockuser@localhost DX95]$
```

```
docker tag 436e9d845d06 mydxregistry.azurecr.io/194core

docker tag 26e0b171cb62 mydxregistry.azurecr.io/194cloud-operator

docker tag c9fed6a373e5 mydxregistry.azurecr.io/194ambassador

docker tag c188f257942c mydxregistry.azurecr.io/194redis

docker tag f05255d7567b mydxregistry.azurecr.io/194digital-asset-manager

docker tag 813392c3eef8 mydxregistry.azurecr.io/194content-composer

docker tag 3a364ef2256d mydxregistry.azurecr.io/194postgres

docker tag 733d97c8f283 mydxregistry.azurecr.io/194image-processor

docker tag 633dda6727c6 mydxregistry.azurecr.io/194digital-asset-
management-operator

docker tag cde0ad39a09b mydxregistry.azurecr.io/194ringapi
```

8- Login to your container registry:

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

9- Push the images to ACR using the following `push` commands:

```
docker push mydxregistry.azurecr.io/194core

docker push mydxregistry.azurecr.io/194cloud-operator

docker push mydxregistry.azurecr.io/194ambassador

docker push mydxregistry.azurecr.io/194redis

docker push mydxregistry.azurecr.io/194digital-asset-manager

docker push mydxregistry.azurecr.io/194content-composer

docker push mydxregistry.azurecr.io/194postgres
docker push mydxregistry.azurecr.io/194image-processor

docker push mydxregistry.azurecr.io/194digital-asset-management-operator

docker push mydxregistry.azurecr.io/194ringapi
```

10- 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:

```
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 az aks get-credentials command. 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

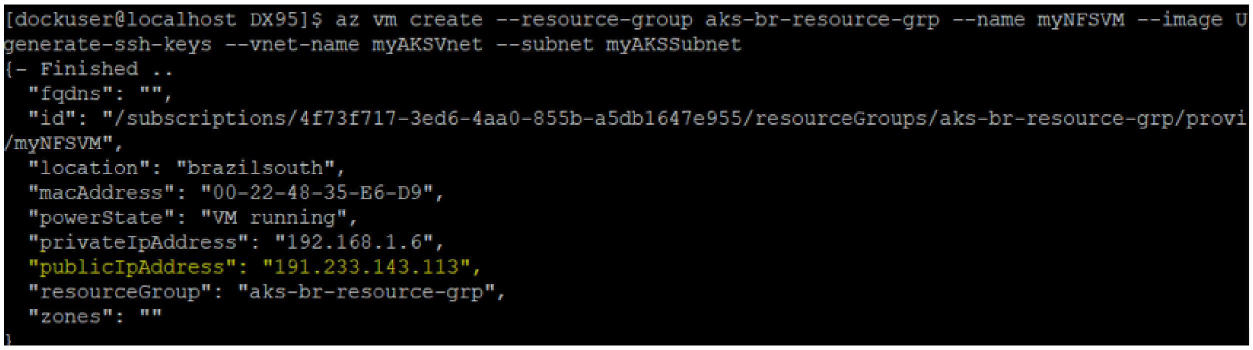
https://docs.microsoft.com/en-us/azure/aks/azure-nfs-volume

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:



```
[dockuser@localhost DX95]$ az vm create --resource-group aks-br-resource-grp --name myNFSVM --image U
generate-ssh-keys --vnet-name myAKSVnet --subnet myAKSSubnet
{- Finished ..
  "fqdns": "",
  "id": "/subscriptions/4f73f717-3ed6-4aa0-855b-a5db1647e955/resourceGroups/aks-br-resource-grp/provi
/myNFSVM",
  "location": "brazilsouth",
  "macAddress": "00-22-48-35-E6-D9",
  "powerState": "VM running",
  "privateIpAddress": "192.168.1.6",
  "publicIpAddress": "191.233.143.113",
  "resourceGroup": "aks-br-resource-grp",
  "zones": ""
}
```

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
```

Deploy DX using dxctl tool

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

```
az login
```

2- In your azure client machine, unzip the file that contains the dxctl tool:

```
unzip hcl-dx-cloud-scripts-v95_CF194_20210416-0233.zip
```

3- Deploy the the DxDeployment custom resource definition:

```
cd hcl-dx-cloud-scripts

./scripts/deployCrd.sh
```

Result:

customresourcedefinition.apiextensions.k8s.io/dxdeployments.git.cwp.pnp-hcl.com created

4- Run the commands below on your client machine:

```
mkdir -p /home/$USER/deployments/

cp dxctl/properties/full-deployment.properties
/home/$USER/deployments/myfirst_deployment.properties

vi /home/$USER/deployments/myfirst_deployment.properties
```

5- Then, update the `dxctl` properties file values below, leave the rest unchanged:

```
dx.namespace: dxns

default.repository: mydxregistry.azurecr.io

dx.name: dx-deployment

dx.image: 194core

dx.tag: latest

dx.storageclass: dx-deploy-stg

dx.volume: wp-profile

dx.volume.size: 100

dx.splitlogging: true

dx.logging.stgclass: default

remote.search.enabled: false

api.enabled: true

api.image: 194ringapi

api.tag: latest

composer.enabled: true
composer.image: 194content-composer

composer.tag: latest

dam.enabled: true

dam.image: 194digital-asset-manager

dam.tag: latest

dam.volume: dam-pv

dam.stgclass: dx-deploy-stg

persist.image: 194postgres

persist.tag: latest

persist.force-read: false

imgproc.image: 194image-processor

imgproc.tag: latest

ingress.image: 194ambassador

ingress.tag: latest

ingress.redis.image: 194redis

ingress.redis.tag: latest

dx.operator.name: dx-deployment-operator

dx.operator.image: 194cloud-operator

dx.operator.tag: latest

dam.operator.image: 194digital-asset-management-operator

dam.operator.tag: latest
```

IMPORTANT: IF you are deploying CF195 or higher, make sure you have the following properties added as well, as they are not required on CF194:

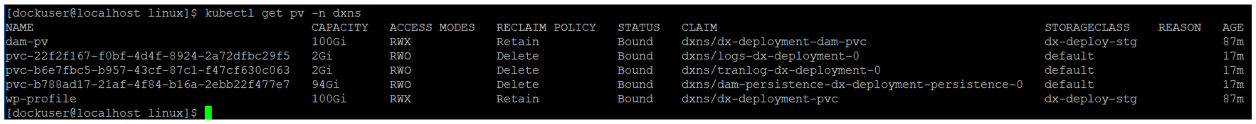
```
dx.tranlogging: true

dx.tranlogging.reclaim:Delete

dx.tranlogging.stgclass: default

dx.tranlogging.size:1G
```

6- Deploy DX using the command below:

```
cd /home/dockuser/DX95/hcl-dx-cloud-scripts/dxctl/linux

./dxctl --deploy -p /home/$USER/deployments/myfirst_deployment.properties
```

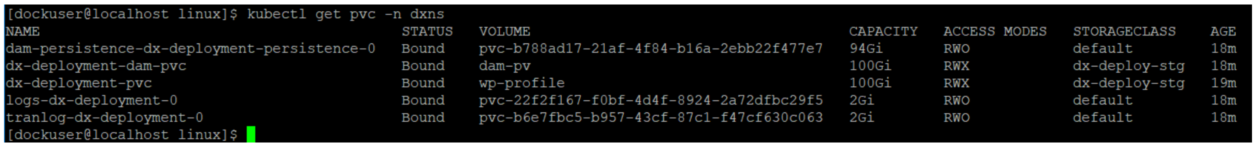
7- Validate the deployment.

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



```
[dockuser@localhost linux]$ kubectl get pv -n dxns
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                                             STORAGECLASS    REASON   AGE
dam-pv                                     100Gi      RWX            Retain           Bound    dxns/dx-deployment-dam-pvc                        dx-deploy-stg            87m
pvc-22f2f167-f0bf-4d4f-8924-2a72dfbc29f5   2Gi        RWO            Delete           Bound    dxns/logs-dx-deployment-0                         default                  17m
pvc-b6e7fbc5-b957-43cf-87c1-f47cf630c063   2Gi        RWO            Delete           Bound    dxns/tranlog-dx-deployment-0                      default                  17m
pvc-b788ad17-21af-4f84-b16a-2ebb22f477e7   94Gi       RWO            Delete           Bound    dxns/dam-persistence-dx-deployment-persistence-0   default                  17m
wp-profile                                 100Gi      RWX            Retain           Bound    dxns/dx-deployment-pvc                            dx-deploy-stg            87m
[dockuser@localhost linux]$
```

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

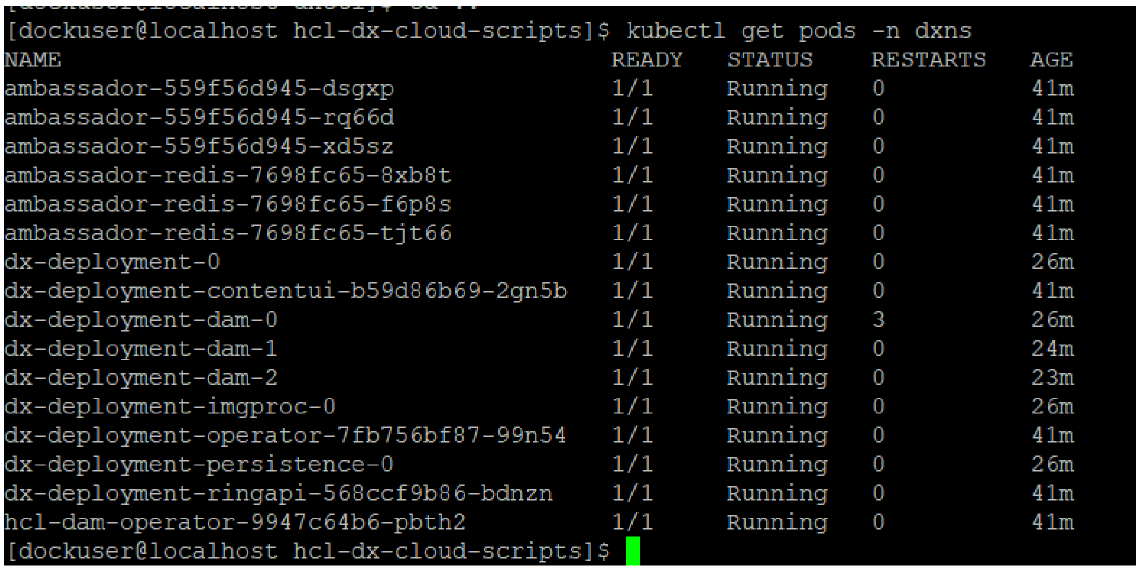


```
[dockuser@localhost linux]$ kubectl get pvc -n dxns
NAME                                          STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS    AGE
dam-persistence-dx-deployment-persistence-0   Bound    pvc-b788ad17-21af-4f84-b16a-2ebb22f477e7   94Gi       RWO            default         18m
dx-deployment-dam-pvc                         Bound    dam-pv                                     100Gi      RWX            dx-deploy-stg   18m
dx-deployment-pvc                             Bound    wp-profile                                 100Gi      RWX            dx-deploy-stg   19m
logs-dx-deployment-0                          Bound    pvc-22f2f167-f0bf-4d4f-8924-2a72dfbc29f5   2Gi        RWO            default         18m
tranlog-dx-deployment-0                       Bound    pvc-b6e7fbc5-b957-43cf-87c1-f47cf630c063   2Gi        RWO            default         18m
[dockuser@localhost linux]$
```

PS: The deployment of the core pod can take a while, on our system, it took around 20 minutes, if the status of the core is stuck in Pending, verify that the pv and pvc are correctly created/bounded. Once the status reaches Init, the transfer of the files will start, so you can take a look inside the wp-profile pv in your NFS server.

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 hcl-dx-cloud-scripts]$ kubectl get pods -n dxns
NAME                                        READY   STATUS    RESTARTS   AGE
ambassador-559f56d945-dsgxp                 1/1     Running   0          41m
ambassador-559f56d945-rq66d                 1/1     Running   0          41m
ambassador-559f56d945-xd5sz                 1/1     Running   0          41m
ambassador-redis-7698fc65-8xb8t             1/1     Running   0          41m
ambassador-redis-7698fc65-f6p8s             1/1     Running   0          41m
ambassador-redis-7698fc65-tjt66             1/1     Running   0          41m
dx-deployment-0                             1/1     Running   0          26m
dx-deployment-contentui-b59d86b69-2gn5b     1/1     Running   0          41m
dx-deployment-dam-0                         1/1     Running   3          26m
dx-deployment-dam-1                         1/1     Running   0          24m
dx-deployment-dam-2                         1/1     Running   0          23m
dx-deployment-imgproc-0                     1/1     Running   0          26m
dx-deployment-operator-7fb756bf87-99n54     1/1     Running   0          41m
dx-deployment-persistence-0                 1/1     Running   0          26m
dx-deployment-ringapi-568ccf9b86-bdnzn      1/1     Running   0          41m
hcl-dam-operator-9947c64b6-pbth2            1/1     Running   0          41m
[dockuser@localhost hcl-dx-cloud-scripts]$
```

Generate TLS Certificate

Create a TLS certification to be used by the deployment.

First, install openssl if you haven't done it already:

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

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

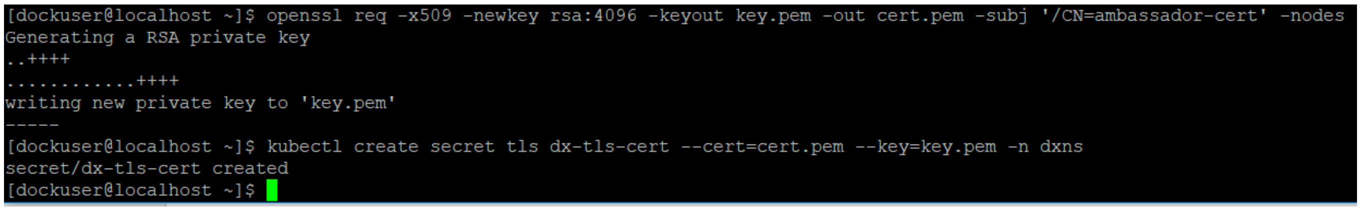
```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -subj
'/CN=ambassador-cert' -nodes
```

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
```

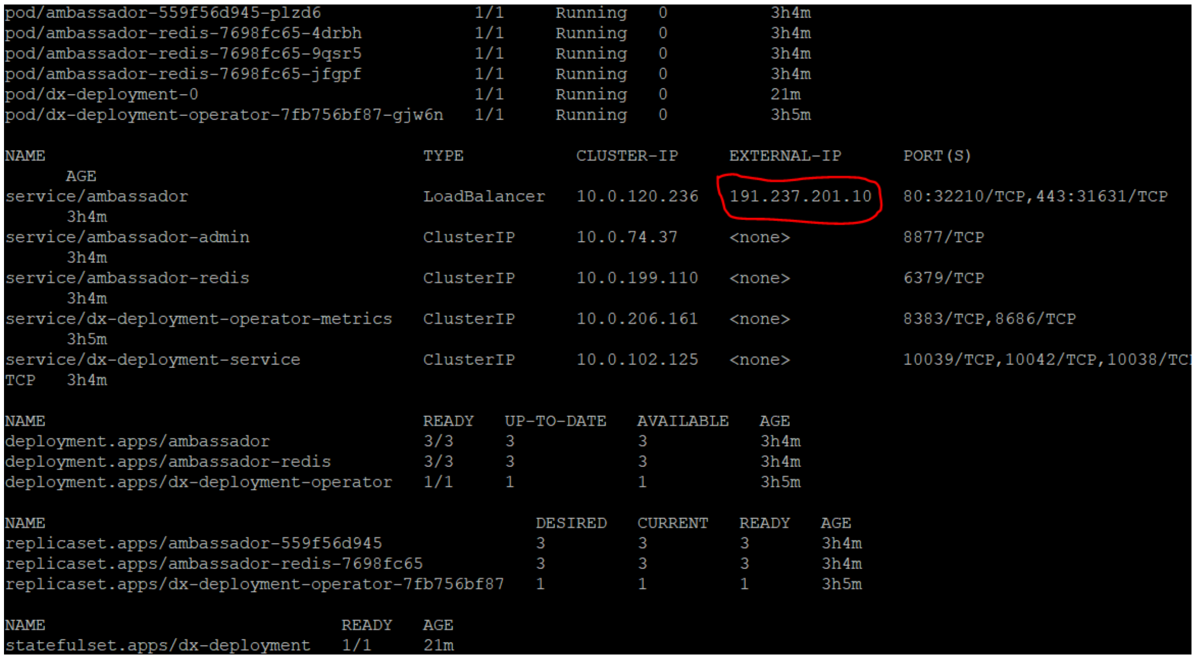


```
[dockuser@localhost ~]$ openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -subj '/CN=ambassador-cert' -nodes
Generating a RSA private key
..++++
...........++++
writing new private key to 'key.pem'
-----
[dockuser@localhost ~]$ kubectl create secret tls dx-tls-cert --cert=cert.pem --key=key.pem -n dxns
secret/dx-tls-cert created
[dockuser@localhost ~]$
```

Testing your Portal deployment

1. Afterwards, access the HCL DX 9.5 CF_194 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:

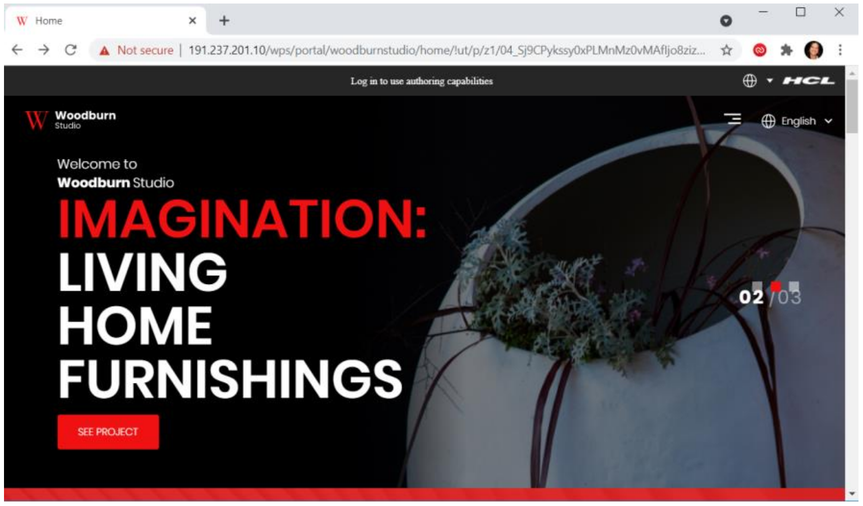
   ```
   kubectl get all -n dxns
   ```



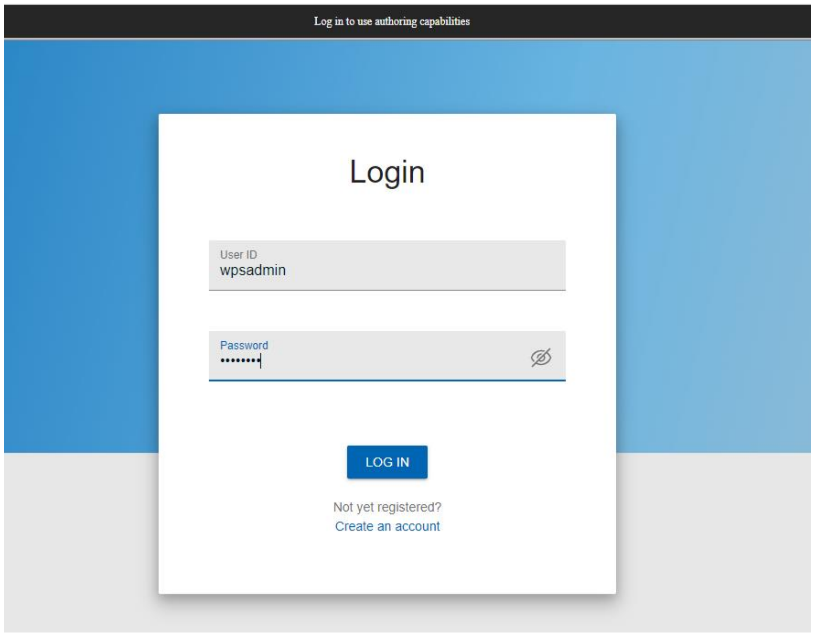
```
pod/ambassador-559f56d945-plzd6                1/1     Running   0          3h4m
pod/ambassador-redis-7698fc65-4drbh            1/1     Running   0          3h4m
pod/ambassador-redis-7698fc65-9qsr5            1/1     Running   0          3h4m
pod/ambassador-redis-7698fc65-jfgpf            1/1     Running   0          3h4m
pod/dx-deployment-0                            1/1     Running   0          21m
pod/dx-deployment-operator-7fb756bf87-gjw6n    1/1     Running   0          3h5m

NAME                                     TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)
      AGE
service/ambassador                       LoadBalancer   10.0.120.236   191.237.201.10   80:32210/TCP,443:31631/TCP
      3h4m
service/ambassador-admin                 ClusterIP      10.0.74.37     <none>           8877/TCP
      3h4m
service/ambassador-redis                 ClusterIP      10.0.199.110   <none>           6379/TCP
      3h4m
service/dx-deployment-operator-metrics   ClusterIP      10.0.206.161   <none>           8383/TCP,8686/TCP
      3h5m
service/dx-deployment-service            ClusterIP      10.0.102.125   <none>           10039/TCP,10042/TCP,10038/TC
TCP   3h4m

NAME                                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ambassador               3/3     3            3           3h4m
deployment.apps/ambassador-redis         3/3     3            3           3h4m
deployment.apps/dx-deployment-operator   1/1     1            1           3h5m

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/ambassador-559f56d945               3         3         3       3h4m
replicaset.apps/ambassador-redis-7698fc65           3         3         3       3h4m
replicaset.apps/dx-deployment-operator-7fb756bf87   1         1         1       3h5m

NAME                               READY   AGE
statefulset.apps/dx-deployment     1/1     21m
```

2. Access your Portal using that URL:

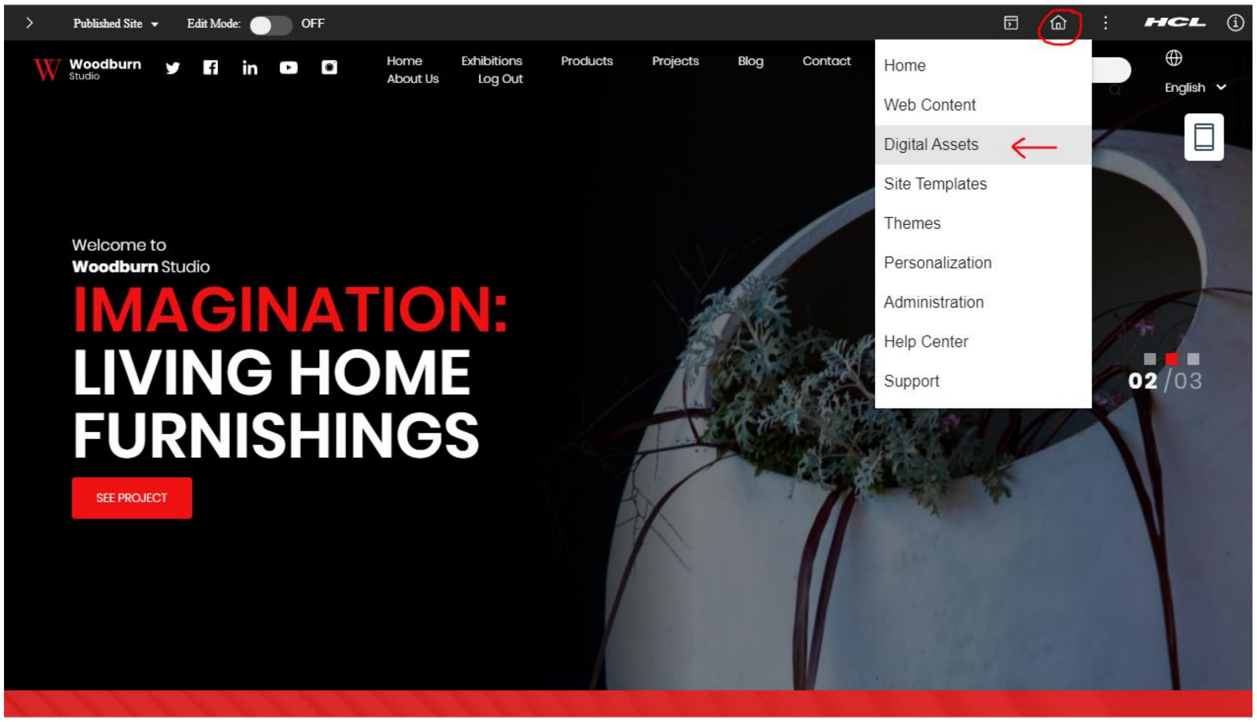
   https://<external-ip>/wps/myportal


Home
Not secure | 191.237.201.10/wps/portal/woodburnstudio/home/!ut/p/z1/04_Sj9CPykssy0xPLMnMz0vMAfljo8ziz...
Log in to use authoring capabilities
HCL
Woodburn Studio
English
Welcome to Woodburn Studio
IMAGINATION:
LIVING
HOME
FURNISHINGS
SEE PROJECT
02 / 03

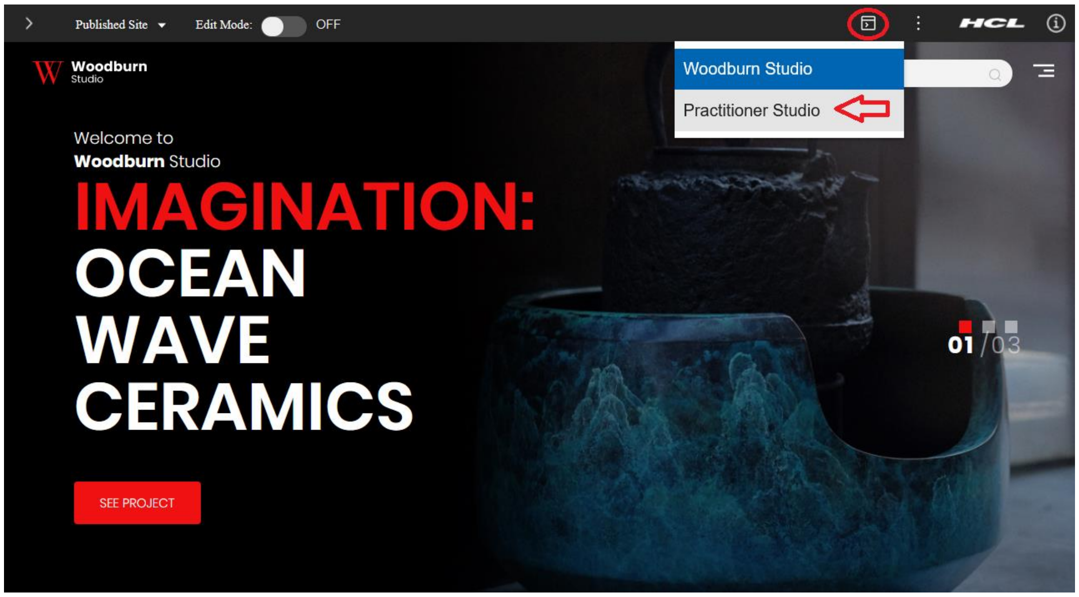
3. Authenticate as your Portal Administrator:



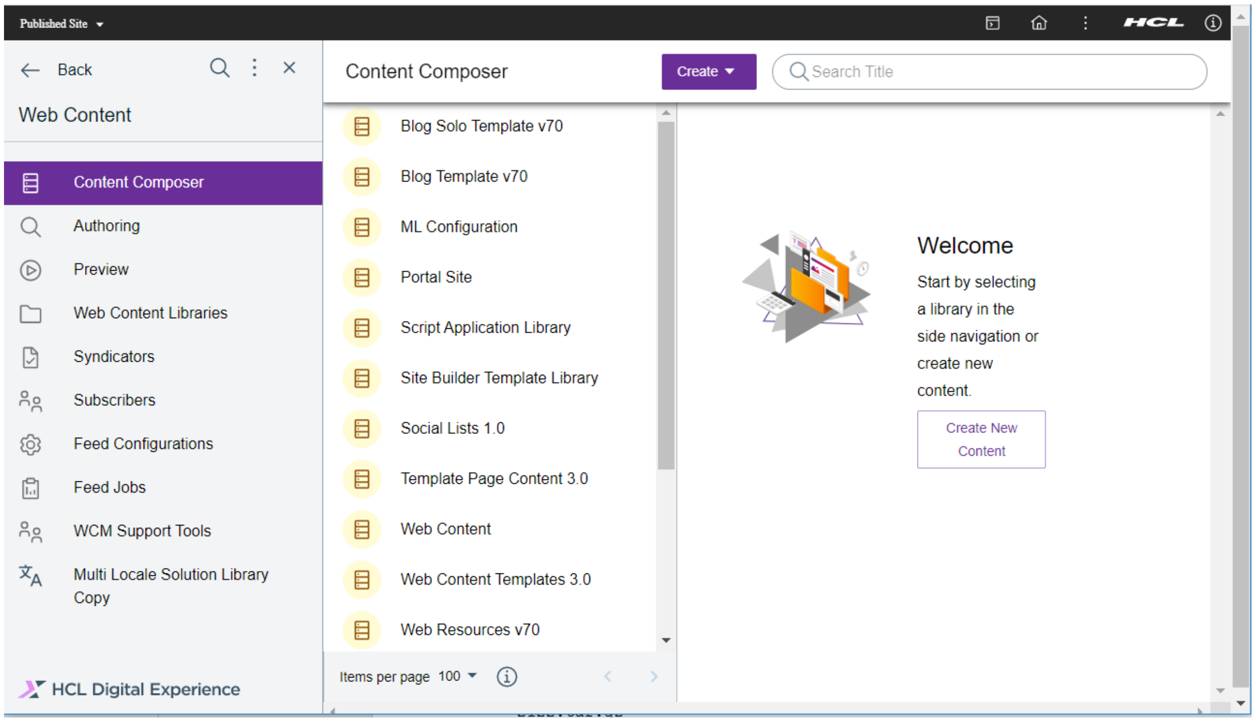
4. You can access DAM by clicking on “Open Applications Menu” and clicking on “Digital Assets”:



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_194 + Digital Asset Manager + Content Composer on Microsoft Azure AKS using dxctl.