DSPACE 7 POC

Written by Charlotte Lim

Quick Setup

API (Backend)

```
cd D4L-Dspace

docker-compose -p d7 up -d
```

Frontend

```
cd Dspace-UI

docker-compose -p d7 -f docker/docker-compose.yml up -d
```

Quick Close

API (Backend)

```
# Ensure that Front end is not running

docker-compose -p d7 down
```

Quick Build

API (Backend)

```
# Ensure that Front end is not running

docker-compose -f docker-compose.yml -f docker-compose-cli.yml
build
```

1. Installations and Setup

DSPACE Frontend and Backend

USING DOCKER - UI and REST API

This is to load up the already pre-configured Dspace's UI and REST API.

This is assuming that you have docker installed

1. Download the repository

```
# Download the UI codebase
git clone https://github.com/DSpace/dspace-angular.git
# Move into the created codebase directory
cd dspace-angular
```

2. Install using docker
   a. Pull images

   ```
   docker-compose -f docker/docker-compose.yml pull
   ```

   b. Start up the docker images

   ```
   docker-compose -p d7 -f docker/docker-compose.yml -f
   docker/docker-compose-rest.yml up -d
   ```

   c. Check logs

   ```
   docker-compose -p d7 -f docker/docker-compose.yml -f
   docker/docker-compose-rest.yml logs -f
   ```

3. Create Admin User for Dspace using another console

```
docker-compose -p d7 -f docker/cli.yml run --rm dspace-cli
create-administrator -e test@test.edu -f admin -l user -p admin -c en
```

This would create a user:
Username: test@test.edu
Password: admin

4. Load up sample Data

This second command will import a batch of test/sample AIPs (see "cli.ingest.yml" for more info)

```
docker-compose -p d7 -f docker/cli.yml -f ./docker/cli.ingest.yml run
--rm dspace-cli
```

Try out on browser:

Frontend: http://localhost:4000/
Backend: http://localhost:8080/server/

Shutting down:

```
# Shut down everything
docker-compose -p d7 -f docker/docker-compose.yml -f
docker/docker-compose-rest.yml down

# Restart everything
docker-compose -p d7 -f docker/docker-compose.yml -f
docker/docker-compose-rest.yml up -d
```

USING DOCKER - Local UI and Local Rest API

1. Install both front-end and backend repositories in a folder

```
mkdir dspace

git clone https://github.com/DSpace/dspace-angular.git


git clone https://github.com/DSpace/DSpace.git
```

This would create a directory dspace and download both front-end (Angular) and backend (Dspace)

2. Load up local REST API (Backend)

```
cd DSpace

docker-compose -p d7 up -d
```

Additional:

Build Docker image in Dspace

```
docker build -t dspace/dspace:dspace-7_x -f Dockerfile .
```

3. Load up sample data into your back end first

   a. Create Admin User for Dspace using another console

```
cd dspace-angular

docker-compose -p d7 -f docker/cli.yml run --rm dspace-cli
create-administrator -e test@test.edu -f admin -l user -p admin -c en
```

This would create a user:

Username: test@test.edu

Password: admin

   b. Load up sample Data

This second command will import a batch of test/sample AIPs (see "cli.ingest.yml" for more info)

```
docker-compose -p d7 -f docker/cli.yml -f ./docker/cli.ingest.yml run
--rm dspace-cli
```

4. Setup front-end to use local

Change the following settings under `environment.common.ts` in dspace-angular

```
// The REST API server settings.
// NOTE: these must be "synced" with the 'dspace.server.url' setting in your backend's local.cfg.
rest: {
  ssl: false,      You, seconds ago • Uncommitted changes
  host: 'localhost:8080',
  port: 443,
  // NOTE: Space is capitalized because 'namespace' is a reserved string in TypeScript
  nameSpace: '/server',
},
```

Location: dspace-angular > src > environments

5. Start the front end

OPTION 1) Local front end testing

```
cd dspace-angular

Yarn install
Yarn build
Yarn start
```

OPTION 2) Running using docker-compose

```
cd dspace-angular

# Build Angular Docker Image (FrontEnd)
docker build . -t dspace/dspace-angular:latest

# Up the frontend
docker-compose -p d7 -f docker/docker-compose.yml up -d

# Read frontend logs
docker-compose -p d7 -f docker/docker-compose.yml -f
docker/docker-compose-rest.yml logs -f
```

Shutting down:

```
# Shut down everything
docker-compose -p d7 -f docker/docker-compose.yml -f
docker/docker-compose-rest.yml down

# Restart everything
docker-compose -p d7 -f docker/docker-compose.yml -f
docker/docker-compose-rest.yml up -d
```

Try out on browser:

Frontend: http://localhost:4000/
Backend: http://localhost:8080/server/

USING DOCKER - Remote VM Access

This configuration is to configure a running setup on a vm. Both Backend and Frontend would be hosted on the same vm.

VM Requirements

- Ubuntu 18 LTS OS
- VM set up and docker/docker-compose installed (See bottom)
- Public Ip address

1. Download files required

```
mkdir dspace
```

```
git clone https://github.com/DSpace/dspace-angular.git

git clone https://github.com/DSpace/DSpace.git
```

2. Configuration
   a. DSpace (Backend/REST)

   Take note of the underlined Bold configurations

   Make the following changes the local.cfg as following, other configuration should be left as it is:

   Location: Dspace > dspace > src > main > docker > local.cfg

```
dspace.server.url=http://<vm_public_ip_address>:8080/server

dspace.ui.url=http://<vm_public_ip_address>:4000

rest.cors.allowed-origins = ${dspace.ui.url},
http://<vm_public_ip_address>:4000, http://localhost:4000

proxies.trusted.ipranges = 172.23.0, <vm_public_ip_address>
```

   Location: Dspace > dspace > src > main > docker-compose > local.cfg

```
dspace.server.url=http://<vm_public_ip_address>:8080/server

dspace.ui.url=http://<vm_public_ip_address>:4000

# NOTE: This setting is required for a REST API running in Docker to
trust requests from the host machine.
# This IP range MUST correspond to the 'dspacenet' subnet defined in
our 'docker-compose.yml'.
proxies.trusted.ipranges = 172.23.0, <vm_public_ip_address>

rest.cors.allowed-origins = ${dspace.ui.url},
http://<vm_public_ip_address>:4000, http://localhost:4000
```

   b. dspace-angular

   Location: dspace-angular > dockerfile

   Change the following line `CMD yarn run start:dev` to this:

```
CMD yarn run start:prod
```

Location: dspace-angular > docker > local.cfg

```
dspace.server.url=http://<vm_public_ip_address>:8080/server

dspace.ui.url=http://<vm_public_ip_address>:4000

rest.cors.allowed-origins = ${dspace.ui.url},
http://<vm_public_ip_address>:4000, http://localhost:4000

proxies.trusted.ipranges = 172.23.0, <vm_public_ip_address>
```

Location: dspace-angular > src > environments

Create a new file with the following name > environment.prod.ts

Paste the following into the file and save

```
export const environment = {
    ui: {
        ssl: false,
        host: '<vm_public_ip_address>',
        port: 4000,
        nameSpace: '/'
    },
    rest: {
        ssl: false,
        host: '<vm_public_ip_address>',
        port: 8080,
        nameSpace: '/server'
    }
};
```

3. Start up back-end first by building the image

Build BackEnd Docker image in Dspace

```
cd Dspace

docker build -t dspace/dspace:dspace-7_x -f Dockerfile .
```

Run Docker-compose for Backend

```
docker-compose -p d7 up -d
```

4. Populate the backend
    a. Create Admin User for Dspace using another console

```
cd dspace-angular

docker-compose -p d7 -f docker/cli.yml run --rm dspace-cli
create-administrator -e test@test.edu -f admin -l user -p admin -c en
```

This would create a user:

Username: test@test.edu
Password: admin

b. Load up sample Data

This second command will import a batch of test/sample AIPs (see "cli.ingest.yml" for more info)

```
docker-compose -p d7 -f docker/cli.yml -f ./docker/cli.ingest.yml run
--rm dspace-cli
```

5. Start front-end

Build Front-End Docker image in Dspace

```
cd dspace-angular

docker build . -t dspace/dspace-angular:latest
```

Start the front-end using docker-compose

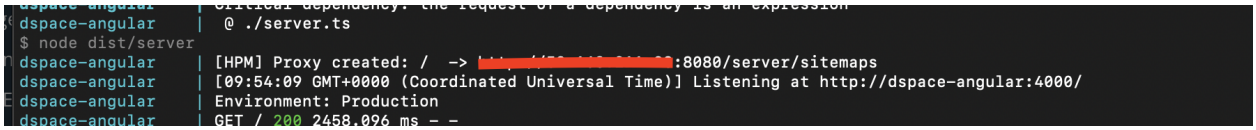
```
docker-compose -p d7 -f docker/docker-compose.yml up -d
```

Open the logs

```
docker-compose -p d7 -f docker/docker-compose.yml -f
docker/docker-compose-rest.yml logs -f
```

6. Check from your browser

When the front-end is read, you should see something like this in the logs (Do note that you have to wait for around a few minutes 5-10 for it to fully load)



Access it from your browser using the following:

On VM:

Frontend: http://localhost:4000/
Backend: http://localhost:8080/server/

On your own browser:

Frontend: http://<VM IP ADDRESS>:4000/
Backend: http://<VM IP ADDRESS>:8080/server/

Shutting down:

```
# Shut down everything
docker-compose -p d7 -f docker/docker-compose.yml -f
docker/docker-compose-rest.yml down

# Restart everything
docker-compose -p d7 -f docker/docker-compose.yml -f
docker/docker-compose-rest.yml up -d
```

API (BACKEND) HTTPS Setup [INCOMPLETE]

https://letsencrypt.org/docs/certificates-for-localhost/
Need some docker scripts (Current Dspace is not production ready) Link
This section would show you how to setup https on your local dspace.

1. Create a certificate for your localhost. If you have previously worked on research-insights, you might have the same certificate before.
2. Change the following settings in environment.dev.ts (dspace > src > main > docker-compose > environment.dev.ts)

```
/**
 * The contents of this file are subject to the license and copyright
 * detailed in the LICENSE and NOTICE files at the root of the source
 * tree and available online at
 *
 * http://www.dspace.org/license/
 */
// This file is based on environment.template.ts provided by Angular UI
export const environment = {
  // Default to using the local REST API (running in Docker)
  rest: {
    ssl: true,      You, seconds ago • Uncommitted changes
    host: 'localhost',
    port: 8080,
    // NOTE: Space is capitalized because 'namespace' is a reserved string in TypeScript
    nameSpace: '/server'
  }
};
```

3.

Kubernetes: Azure (AKS)

1. Connect to the cluster **(Do this once)**

```
az aks get-credentials --name <Cluster_name> --resource-group <resource_group>
```

After running this command, it would show up in the lens. You can check the name under AKS. In our case:
<Cluster_name> : aks-dspace
<resource_group> : dspacetest

2. Change the context (IF previously on minikube)

```
# View all current context
kubectl config get-contexts

# Switch context
kubectl config use-context <CONTEXT_NAME>
```

3. Deploy the pods

```
kubectl apply -f k8s/AKS
```

Running ingress AKS

https://docs.nginx.com/nginx-ingress-controller/installation/installation-with-helm/

Free no cost

Applying Ingress controllers

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.0.4/deploy/static/provider/cloud/deploy.yaml
```

Checking the pods

```
kubectl get pods -n ingress-nginx \
  -l app.kubernetes.io/name=ingress-nginx --watch
```

Applying the manifest

```
kubectl apply -f k8s/AKS/dspace-ingress.yaml

# Check the load balancer
kubectl get services -n=ingress-nginx
```

Example output:

```
NAME                                 TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)                      AGE
ingress-nginx-controller             LoadBalancer   10.0.199.13    40.119.235.43   80:30324/TCP,443:30235/TCP   2d23h
ingress-nginx-controller-admission   ClusterIP      10.0.217.240   <none>          443/TCP                      2d23h
```

Access your web at <EXTERNAL_IP>/server/

http://40.119.235.43/server/api/authn/token?redirectUrl=http://40.119.235.43/home&token=<INSERT TOKEN>

Static IP

Helm Has to be working for this.

Move the relevant images for ingress into our ACR

```
REGISTRY_NAME=<REGISTRY_NAME>
CONTROLLER_REGISTRY=k8s.gcr.io
CONTROLLER_IMAGE=ingress-nginx/controller
CONTROLLER_TAG=v0.48.1
PATCH_REGISTRY=docker.io
PATCH_IMAGE=jettech/kube-webhook-certgen
PATCH_TAG=v1.5.1
DEFAULTBACKEND_REGISTRY=k8s.gcr.io
DEFAULTBACKEND_IMAGE=defaultbackend-amd64
DEFAULTBACKEND_TAG=1.5
CERT_MANAGER_REGISTRY=quay.io
CERT_MANAGER_TAG=v1.3.1
CERT_MANAGER_IMAGE_CONTROLLER=jetstack/cert-manager-controller
```

```
CERT_MANAGER_IMAGE_WEBHOOK=jetstack/cert-manager-webhook
CERT_MANAGER_IMAGE_CAINJECTOR=jetstack/cert-manager-cainjector

az acr import --name $REGISTRY_NAME --source
$CONTROLLER_REGISTRY/$CONTROLLER_IMAGE:$CONTROLLER_TAG --image
$CONTROLLER_IMAGE:$CONTROLLER_TAG
az acr import --name $REGISTRY_NAME --source
$PATCH_REGISTRY/$PATCH_IMAGE:$PATCH_TAG --image
$PATCH_IMAGE:$PATCH_TAG
az acr import --name $REGISTRY_NAME --source
$DEFAULTBACKEND_REGISTRY/$DEFAULTBACKEND_IMAGE:$DEFAULTBACKEND_TAG
--image $DEFAULTBACKEND_IMAGE:$DEFAULTBACKEND_TAG
az acr import --name $REGISTRY_NAME --source
$CERT_MANAGER_REGISTRY/$CERT_MANAGER_IMAGE_CONTROLLER:$CERT_MANAGER
_TAG --image $CERT_MANAGER_IMAGE_CONTROLLER:$CERT_MANAGER_TAG
az acr import --name $REGISTRY_NAME --source
$CERT_MANAGER_REGISTRY/$CERT_MANAGER_IMAGE_WEBHOOK:$CERT_MANAGER_TA
G --image $CERT_MANAGER_IMAGE_WEBHOOK:$CERT_MANAGER_TAG
az acr import --name $REGISTRY_NAME --source
$CERT_MANAGER_REGISTRY/$CERT_MANAGER_IMAGE_CAINJECTOR:$CERT_MANAGER
_TAG --image $CERT_MANAGER_IMAGE_CAINJECTOR:$CERT_MANAGER_TAG
```

e.g **<AZURE_REGISTRY_NAME>:** `ALPDspace`

Get Resource name of cluster

az aks show --resource-group **<RESOURCE_GROUP>** --name **<CLUSTER_NAME>** --query nodeResourceGroup -o tsv

e.g
<RESOURCE_GROUP>: Dspacetest
<CLUSTER_NAME>: aks-dspace

Example Response:

```
charlotte.lim@Charlottes-MBP D4L-Dspace % az aks show --resource-group Dspacetest --name aks-dspace  --query nodeR
esourceGroup -o tsv

MC_dspacetest_aks-dspace_southeastasia
```

Create a public static Ip from this resource (Need Permission Level)

az network public-ip create --resource-group **<Resource_name_from_previous>** --name **<CUSTOM_IP>** --sku Standard --allocation-method static --query publicIp.ipAddress -o tsv

<Resource_name_from_previous>: MC_dspacetest_aks-dspace_southeastasia
<CUSTOM_IP>: alpDspacePublicIP

Example output:

```
az network public-ip create --resource-group MC_dspacetest_aks-dspace_southeastasia -
-name alpDspacePublicIP --sku Standard --allocation-method static --query
publicIp.ipAddress -o tsv
[Coming breaking change] In the coming release, the default behavior will be changed
as follows when sku is Standard and zone is not provided: For zonal regions, you will
get a zone-redundant IP indicated by zones:["1","2","3"]; For non-zonal regions, you
will get a non zone-redundant IP indicated by zones:null.
52.253.88.76
```

Login into helm

```
# Show Credentials
az acr credential show --name <REGISTRY_NAME>

# Set Environment
export HELM_EXPERIMENTAL_OCI=1

# Login
echo $spPassword | helm registry login <REGISTRY_NAME>.azurecr.io \
  --username ALPDspace \
  --password=<PASSWORD>
```

<PASSWORD>: T
aken from show credentials
<REGISTRY_NAME>: alpdspace

Install Helm Chart For Ingress
See [Img pull Error]

```
# Add the ingress-nginx repository
helm repo add ingress-nginx
https://kubernetes.github.io/ingress-nginx

# Set variable for ACR location to use for pulling images
ACR_URL=<REGISTRY_URL>
STATIC_IP=<STATIC_IP>
DNS_LABEL=<DNS_LABEL>

# Use Helm to deploy an NGINX ingress controller
helm install nginx-ingress ingress-nginx/ingress-nginx \
    --version 3.36.0 \
    --set controller.replicaCount=2 \
    --set controller.nodeSelector."kubernetes\.io/os"=linux \
    --set controller.image.registry=$ACR_URL \
    --set controller.image.image=$CONTROLLER_IMAGE \
    --set controller.image.tag=$CONTROLLER_TAG \
    --set controller.image.digest="" \
    --set
```

```
controller.admissionWebhooks.patch.nodeSelector."kubernetes\.io/os"
=linux \
    --set
controller.admissionWebhooks.patch.image.registry=$ACR_URL \
    --set
controller.admissionWebhooks.patch.image.image=$PATCH_IMAGE \
    --set controller.admissionWebhooks.patch.image.tag=$PATCH_TAG \
    --set defaultBackend.nodeSelector."kubernetes\.io/os"=linux \
    --set defaultBackend.image.registry=$ACR_URL \
    --set defaultBackend.image.image=$DEFAULTBACKEND_IMAGE \
    --set defaultBackend.image.tag=$DEFAULTBACKEND_TAG \
    --set controller.service.loadBalancerIP=$STATIC_IP \
    --set
controller.service.annotations."service\.beta\.kubernetes\.io/azure
-dns-label-name"=$DNS_LABEL
```

<REGISTRY_URL>: alpdspace.azurecr.io
<STATIC_IP>: 52.253.88.76
<DNS_LABEL>: (Create your own) alp-dspace-demo

This label will create a DNS name of the form
<DNS_LABEL>.<AZURE REGION NAME>.*cloudapp.azure.com*

In this case would be alp-dspace-demo.southeastasia.cloudapp.azure.com

Check your config

```
# Check LoadBalancer
kubectl get services -o wide

# Check DNS (You might need permission level)
az network public-ip list --resource-group
<Resource_name_from_previous> --query
"[?name==<CUSTOM_IP>'].[dnsSettings.fqdn]" -o tsv
```

<Resource_name_from_previous>: MC_dspacetest_aks-dspace_southeastasia
<CUSTOM_IP>: alpDspacePublicIP

Enabling https On static IP using LetsEncrypt Certificates

Resource: Link

Install Cert-manager

```
kubectl label namespace default
cert-manager.io/disable-validation=true

# Add the Jetstack Helm repository
```

```
helm repo add jetstack https://charts.jetstack.io

# Update your local Helm chart repository cache
helm repo update

# Install the cert-manager Helm chart
helm install cert-manager jetstack/cert-manager \
  --version $CERT_MANAGER_TAG \
  --set installCRDs=true \
  --set nodeSelector."kubernetes\.io/os"=linux \
  --set image.repository=$ACR_URL/$CERT_MANAGER_IMAGE_CONTROLLER \
  --set image.tag=$CERT_MANAGER_TAG \
  --set
webhook.image.repository=$ACR_URL/$CERT_MANAGER_IMAGE_WEBHOOK \
  --set webhook.image.tag=$CERT_MANAGER_TAG \
  --set
cainjector.image.repository=$ACR_URL/$CERT_MANAGER_IMAGE_CAINJECTOR
\
  --set cainjector.image.tag=$CERT_MANAGER_TAG
```

Resource: [Cert-manager]

Apply the cluster-issuer

```
kubectl apply -f k8s/AKS/cluster-issuer.yaml
```

Apply Deployments and Ingress

```
kubectl apply -f k8s/AKS
```

Check Certificate

```
kubectl get certificates

kubectl describe certificate alp-dspace-demo-secret
```

Example output:

```
Events:
  Type    Reason     Age   From          Message
  ----    ------     ----  ----          -------
  Normal  Issuing    77s   cert-manager  Issuing certificate as Secret does not exist
  Normal  Generated  76s   cert-manager  Stored new private key in temporary Secret resource "alp-dspace-demo-secr
et-kljzr"
  Normal  Requested  76s   cert-manager  Created new CertificateRequest resource "alp-dspace-demo-secret-qwjmm"
  Normal  Issuing    72s   cert-manager  The certificate has been successfully issued
```

Note: After creating this, DO NOT delete the secret that is generated unless you are using staging certificate

Kubernetes: Local Minikube

Installation

```
# Install minikube
Brew install minikube
# Install kubectl
brew install kubectl

# Check versions
Kubectl version
minikube version
```

Start Minikube

```
minikube start --nodes=2
```

Connecting with Azure Container Registry via Secrets

1. Create a container registry in azure
2. Login in local

```
# Login into azure
Az login
```

Replace **<REGISTRY_NAME>** with the name of the registry created. In our case we would use **ALPDspace**.

Output:

```
charlotte.lim@Charlottes-MBP D4L-Dspace % az acr login --name ALPDspace
Uppercase characters are detected in the registry name. When using its server url in docker commands, to avoid authentic
ation errors, use all lowercase.
Login Succeeded
```

3. Get the acr repository location for images using the following command

```
az acr list --resource-group <RESOURCE_GROUP_NAME>  --query
"[].{acrLoginServer:loginServer}" --output table
```

Replace **<RESOURCE_GROUP_NAME>** with the resource group that was configured for your docker registry. In this case we will use **Dspacetest**

Output:

```
charlotte.lim@Charlottes-MBP D4L-Dspace % az acr list --resource-group dspacetest --query "[].{acrLoginServer:loginServe
r}" --output table
AcrLoginServer
--------------------
alpdspace.azurecr.io
```

4. Change the docker-compose files image line to the following

In docker-compose-cli.yml

```
image: "alpdspace.azurecr.io/dspace-cli:${DSPACE_VER:-dspace-7_x}"
```

In docker-composei.yml

```
image: "alpdspace.azurecr.io/dspace:${DSPACE_VER:-dspace-7_x-test}"
```

```
You, an hour ago | 3 authors (Tim Donohue and others)
1  version: '3.7'
2  networks:
3    dspacenet:
4      ipam:
5        config:
6          # Define a custom subnet for our DSpace network, so that we can easily trust requests
7          # If you customize this value, be sure to customize the 'proxies.trusted.ipranges' in
8          - subnet: 172.23.0.0/16
9  services:
10   # DSpace (backend) webapp container
11   dspace:
12     container_name: dspace
13     # image: "alpdspace.azurecr.io/dspace:${DSPACE_VER:-dspace-7_x-test}"
14     image: "alpdspace.azurecr.io/dspace:${DSPACE_VER:-dspace-7_x-test}"      You, an hour ago
15     build:
16       context: .
17       dockerfile: Dockerfile.test
18     depends_on:
19     - dspacedb
20     networks:
21       dspacenet:
```

Pushing images to ACR

Building images

```
# Build dspace image
docker-compose -f docker-compose.yml build

# Build cli image
docker-compose -f docker-compose-cli.yml build

# Build solr image
docker build -t alpdspace.azurecr.io/dspace-solr -f
dspace/src/main/docker/solr/Dockerfile .

# Build FE Image
docker build -t alpdspace.azurecr.io/dspace-angular:dev -f .

# Build FE Image using separate dockerfile
```

```
docker build -t alpdspace.azurecr.io/dspace-angular:dev-v0.1
-f Dockerfile.dev .
```

Pushing the image

```
# Login to ACR Repository
az acr login --name <REPOSITORY_NAME>

# Push dspace image
docker-compose -f docker-compose.yml push

# Push cli image
docker-compose -f docker-compose-cli.yml push

# Push solr
docker image push alpdspace.azurecr.io/dspace-solr:latest
```

In this case our **<REPOSITORY_NAME>** is **ALPDspace**

Getting Azure Secret for Image Pulling

Get Pull Secret Password

```
# Connect to the registry
az acr login --name <REGISTRY_NAME>

# Set admin (only do this once)
az acr update -n <REGISTRY_NAME> --admin-enabled true

# Show Credentials
az acr credential show --name <REGISTRY_NAME>
```

Example output:

```
charlotte.lim@Charlottes-MacBook-Pro D4L-Dspace % az acr credential show --name ALPDspace
{
  "passwords": [
    {
      "name": "password",
      "value": "[redacted]"
    },
    {
      "name": "password2",
      "value": "[redacted]"
    }
  ],
  "username": "ALPDspace"
}
```

Run the following command to generate the secret

```
kubectl create secret docker-registry <SECRETNAME>
--docker-server=<REGISTRY_NAME>.azurecr.io --docker-username=<USERNAME>
--docker-password=<PASSWORD_VALUE>
```

In our example,

<SECRETNAME> : **dspace-secret**

<REGISTRY_NAME> : **ALPDspace**

<USERNAME> : **ALPDspace** (From the picture above)

<PASSWORD_VALUE> : ”value” content (From the picture above)

In the sense, the following command is used to create secret (You need recreate it every time you restart minikube)

```
kubectl create secret docker-registry dspace-secret
--docker-server=alpdspace.azurecr.io
--docker-username=ALPDspace --docker-password=<password value
(Pick any)>
```

If you mess up, you can delete the secret with the following command

```
kubectl delete secret dspace-secret
```

Copy it to the yaml files for both cli-deployment and dspace-deployment
Under containers:
Do this for dspace-cli-deployment.yml and dspace-deployment.yml

```yaml
  spec:
    containers:
      - args:
          - help
        command:
          - /dspace/bin/dspace
        image: alpdspace.azurecr.io/dspace-cli:dspace-7_x
        name: dspace-cli
        resources: {}
        stdin: true
        tty: true
        volumeMounts:
          - mountPath: /dspace/config/local.cfg
            name: dspace-cli-claim0
          - mountPath: /dspace/assetstore
```

```
            name: assetstore
    imagePullSecrets:
      - name: "dspace-secret"
```

Running Deployment on Minikube (Local)

1. Apply the deployment files

```
kubectl apply -f k8s
```

2. Check the pod if its running

```
kubectl get pods
```

The pods are only completed if they have the 1/1 status as shown below

```
NAME                                    READY   STATUS    RESTARTS   AGE
dspace-angular-deploy-646d7d69b8-b8tvm  1/1     Running   0          48m
dspace-deploy-9698468f-4nsp6            1/1     Running   0          48m
dspace-solr-deploy-9df54c55c-s2qs8      1/1     Running   0          48m
dspacedb-deploy-58fb76cf8d-hdlvr        1/1     Running   0          48m
```

3. Create Admin Account and Import sample data (DEPRECATED)
   <u>Note: You can only do this if all pods are ready</u>

Create admin Account

```
kubectl apply -f k8s/cli/dspace-createAdmin.yaml

# Check status
kubectl get pods

Kubectl logs <DSPACE-CREATE-ADMIN_POD_NAME>
```

Import Sample Data

```
kubectl apply -f k8s/cli/dspace.job.ingest.v7.yaml

# Check status
kubectl get pods

kubectl logs <DSPACE-INGEST_POD_NAME>
```

If its finish, you would be able to see the following after running the following command

```
kubectl get jobs
```

```
NAME                    COMPLETIONS   DURATION   AGE
dspace-create-admin     1/1           22s        40m
dspace-ingest           1/1           118s       37m
```

4. Port-forward the ports in order to access it in your local browser

```
# Backend
kubectl port-forward service/dspace-service 8080:8080

# Solr
kubectl port-forward service/dspace-solr-service 8983:8983

# Front-End
kubectl port-forward service/dspace-solr-service 4000:4000
```

You can access it at your local browser at:
Backend: http://localhost:8080/server
Solr: http://localhost:8983/
Frontend: http://localhost:4000

Populate Current Dspace

In order to do that, we have to enter into Dspace's deployment pod and ensure that the database is working properly. Use the following to check the database connection.
Ensure that Dspace has been fully initialized before attempting this.

1. Ensure that the `db.password` and `db.user` information has been filled under `local.cfg`
2. Enter the Dspace pod shell:

```
kubectl exec -it <POD_NAME> -c <CONTAINER_NAME> -- //bin/bash
```

Where
- **<POD_NAME>** refers to the pod name
- **<CONTAINER_NAME>** is `dspace`

3. Migrate Database

```
/dspace/bin/dspace database migrate
```

Expected output:

```
root@dspace-deploy-8569ff4f95-45nhj:/usr/local/tomcat# /dspace/bin/dspace database migrate

Database URL: jdbc:postgresql://alp-pg-dev.postgres.database.azure.com:5432/dspace
Migrating database to latest version... (Check dspace logs for details)
Done.
root@dspace-deploy-8569ff4f95-45nhj:/usr/local/tomcat#
```

4. Create Administrator

```
/dspace/bin/dspace create-administrator -e <ADMIN_EMAIL> -f <FIRST_NAME>
-l <LAST_NAME> -p <ADMIN_PASS> -c en
```

- <ADMIN_EMAIL>: The admin account used to populate data
- <ADMIN_PASS>: Any password

e.g `/dspace/bin/dspace create-administrator -e charlotte.lim@data4life-asia.care -f charlotte -l lim -p adminpassword -c en`

5. Populate data

Paste and run the following script

SETUP Environment Variables

```
JAVA_MEM=${JAVA_MEM:--Xmx2500m}
export JAVA_OPTS="${JAVA_OPTS} ${JAVA_MEM}
-Dupload.temp.dir=/dspace/upload -Djava.io.tmpdir=/tmp"

AIPZIP=${AIPZIP:-https://github.com/DSpace-Labs/AIP-Files/raw/master/dog
AndReport.zip}
```

Extract data using Admin Email, Replace email with admin email created before

```
ADMIN_EMAIL=${ADMIN_EMAIL:-<ADMIN_EMAIL>}
AIPDIR=/tmp/aip-dir
rm -rf ${AIPDIR}
mkdir ${AIPDIR} /dspace/upload
cd ${AIPDIR}
pwd
curl ${AIPZIP} -L -s --output aip.zip
unzip aip.zip
cd ${AIPDIR}
```

Run Dspace Packager config

```
/dspace/bin/dspace packager -r -a -t AIP -e ${ADMIN_EMAIL} -f -u
SITE*.zip

/dspace/bin/dspace database update-sequences

/dspace/bin/dspace index-discovery -b
```

Expected output:

1) Packager

```
root@dspace-deploy-8569ff4f95-45nhj:/tmp/aip-dir# /dspace/bin/dspace packager -r -a -t AIP -e ${ADMIN_EMAIL} -f -u SITE*.zip
Beginning replacement process...

Replacing DSpace object(s) with package located at SITE.zip

REPLACED a total of 57 DSpace Objects.


REPLACED DSpace SITE [ hdl=123456789/0 ]
REPLACED DSpace COMMUNITY [ hdl=123456789/2 ]
REPLACED DSpace COLLECTION [ hdl=123456789/20 ]
REPLACED DSpace ITEM [ hdl=123456789/21 ]
REPLACED DSpace ITEM [ hdl=123456789/22 ]
REPLACED DSpace ITEM [ hdl=123456789/23 ]
REPLACED DSpace ITEM [ hdl=123456789/24 ]
REPLACED DSpace COLLECTION [ hdl=123456789/16 ]
REPLACED DSpace ITEM [ hdl=123456789/17 ]
```

2) Update-sequences

```
root@dspace-deploy-8569ff4f95-45nhj:/tmp/aip-dir# /dspace/bin/dspace database update-sequences
Running org/dspace/storage/rdbms/sqlmigration/postgres/update-sequences.sql
update-sequences complete
```

3) Index discovery

```
root@dspace-deploy-8569ff4f95-45nhj:/tmp/aip-dir# /dspace/bin/dspace index-discovery -b
The script has started
(Re)building index from scratch.
Done with indexing
The script has completed
```

Note that Index Discovery command is the command that repopulates the current api with data from postgres

Running Ingress minikube

Note that Ingress on minikube does not work well if the driver used is a docker driver. Only Hyperkit can be used when testing ingress according to here. If you were using docker previously, run the following command to start hyperkit

To use hyperkit instead

```
# Install if first time
brew install hyperkit

# Stop previous
minikube stop
```

```
# Delete
minikube delete

# Start minikube
minikube start minikube start
--cpus 4 --memory 8192
--driver=hyperkit
```

Remember to re-set up the secrets after deleting the previous minikube

1. Enable ingress

```
minikube addons enable ingress
```

2. Check that ingress is running

```
# Check pod name
kubectl get pods -n ingress-nginx

# Get logs
kubectl logs <INGRESS_POD_NAME> -n ingress-nginx
```

3. Visit the dspace service via nodePod, we can check if the app is running by using this command.

```
minikube service dspace-service --url
```

This command checks if your service is working even without the ingress.
Ctrl-C this once finish testing

Example output:

```
charlotte.lim@Charlottes-MBP D4L-Dspace % minikube service dspace-service --url
http://192.168.64.2:31299
http://192.168.64.2:30122
http://192.168.64.2:32420
http://192.168.64.2:31894
http://192.168.64.2:30777
```

4. Apply the ingress router to our service

```
kubectl apply -f k8s/Local/ingress-controller.yaml
```

5. Verify that the ingress works.

Note: This might take a while for the address to popup

```
kubectl get ingress
```

Example output:

```
NAME             CLASS   HOSTS         ADDRESS        PORTS   AGE
dspace-ingress   nginx   dspace.info   192.168.64.2   80      58s
```

6. Edit the /etc/hosts file

```
# Edit the file with info
echo '192.168.64.2   dspace.info' | sudo tee -a /etc/hosts

# Confirm that it works
cat /etc/hosts
```

```
charlotte.lim@Charlottes-MBP D4L-Dspace % cat /etc/hosts
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost
# Added by Docker Desktop
# To allow the same kube context to work on the host and the container:
127.0.0.1 kubernetes.docker.internal
# End of section
192.168.64.2   dspace.info
```

7. Check the endpoint

```
curl dspace.info/server/api
```

You can access it at the browser at

Backend: dspace.info/server/
Solr: dspace.info/solr/

Additional Notes:
In order for the frontend to work, some changes were made in the front end

Added the following flag under scripts/serve.ts

```
ng serve --disable-host-check
```

Reasoning: [StackOverFlow]

Useful Kubectl Commands

Apply

```
kubectl apply -f k8s
```

Restart pods

```
kubectl rollout restart deploy
```

Get Pods

```
Kubectl get pods
```

Get More pod info

```
# Single container
kubectl describe pod <POD_NAME>
```

Get Pod Service info

```
kubectl get svc
```

Example output

```yaml
apiVersion: v1
kind: Service
metadata:
  name: dspace-service
  labels:
    app: dspace-app
spec:
  type: NodePort
  selector:
    app: dspace-app
  ports:
  - protocol: T
    port: 8080
    targetPort: 8080
    name: http
---
```

```
NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                    AGE
dspace-angular-service  NodePort    10.107.216.89   <none>        3000:30873/TCP,9876:31656/TCP   22m
dspace-service          NodePort    10.100.245.94   <none>        8080:3                     22m
dspace-solr-service     NodePort    10.104.125.46   <none>        8983:301/TCP               22m
dspacedb-service        NodePort    10.111.234.58   <none>        5432:31908/TCP             22m
kubernetes              ClusterIP   10.96.0.1       <none>        443/TCP                    6d2h
charlotte.lim@Charlottes-MBP D4L-Dspace %
```

Exposed Port

Get Pod Logs

```
# Single Container
kubectl logs -f <POD_NAME>

# Multiple containers
kubectl logs -f <POD_NAME> -c <CONTAINER_NAME>
```

Entering pod shell

```
# Single container
kubectl exec -it <POD NAME> -- //bin/bash

# Multiple containers
kubectl exec -it <POD NAME> -c <CONTAINER_NAME>-- //bin/bash

# Only frontend (It only has shell)
kubectl exec -it <POD NAME> -c dspace-angular -- sh
```

Delete all pods

```
kubectl delete -f <File name where all the yaml are>
```

Delete Everything

```
kubectl delete all --all
```

Running curl from one pod to another

```
# Check available pods
kubectl get po -o wide


# Run command (Single container)
kubectl exec -t <one_pod>  -- curl -I <another pod's service
endpoint>

# Run command (Multiple container)
kubectl exec -t <one_pod> -c=<container_name>  -- curl -I <another
pod's service endpoint>
```

E.g *kubectl exec -t dspace-deploy-9698468f-sg8dp -- curl -I 10.101.56.155:8080*

E.g *kubectl exec -t dspace-deploy-7fc789d78-w68tn -c=dspace -- curl -I http://dspace-service:8983*

Switching context

```
# View all current context
kubectl config get-contexts
```

```
# Switch context
kubectl config use-context <CONTEXT_NAME>
```

E.g `kubectl config use-context minikube`

g

kubectl delete -f **<File name where all the yaml are>**

g

kubectl delete -f **<File name where all the yaml are>**

Linux / Ubuntu (VM SetUp)

SSH into ubuntu vm

```
Sudo ssh -i /Users/charlotte.lim/Desktop/DSpace-7.0-keypair.pem
azureuser@52.163.211.23
```

Set up Remote Desktop

1. Install xfce4

```
sudo apt-get update

sudo apt-get -y install xfce4
```

2. Configure Remote Desktop
    a. Install xrdp

```
sudo apt-get -y install xrdp

sudo systemctl enable xrdp
```

b. Tell xrdp what desktop environment to use when you start your session. Configure xrdp to use xfce as your desktop environment as follows:

```
echo xfce4-session >~/.xsession
```

c. Restart the service

```
sudo service xrdp restart
```

3. Set up local account
   a. Create account to access the vm

   ```
   sudo passwd <password>
   ```

   This sets up password for the user azureuser

   b. Follow the next step to access via remote desktop

Install Remote Desktop viewer for Mac application :

Download: [Link]

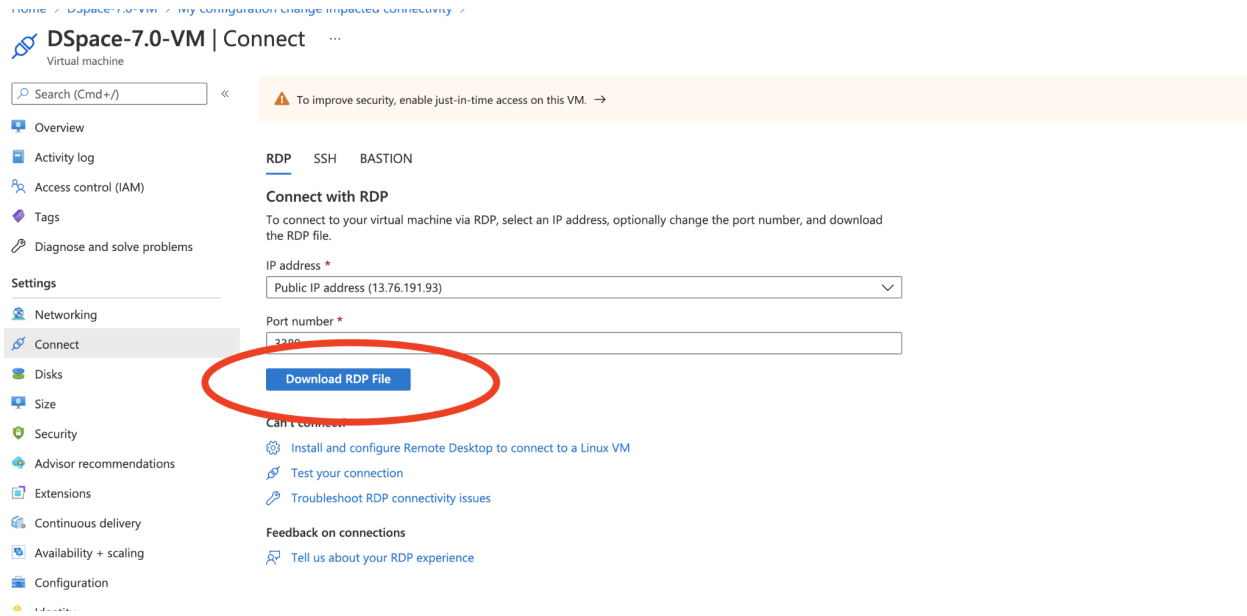
Connect to the remote desktop

Add the following settings in networks

Settings > InBound port rule (link)
This is to allow RDP connections

310	RDP	3389	TCP	Any	Any	Allow	•••

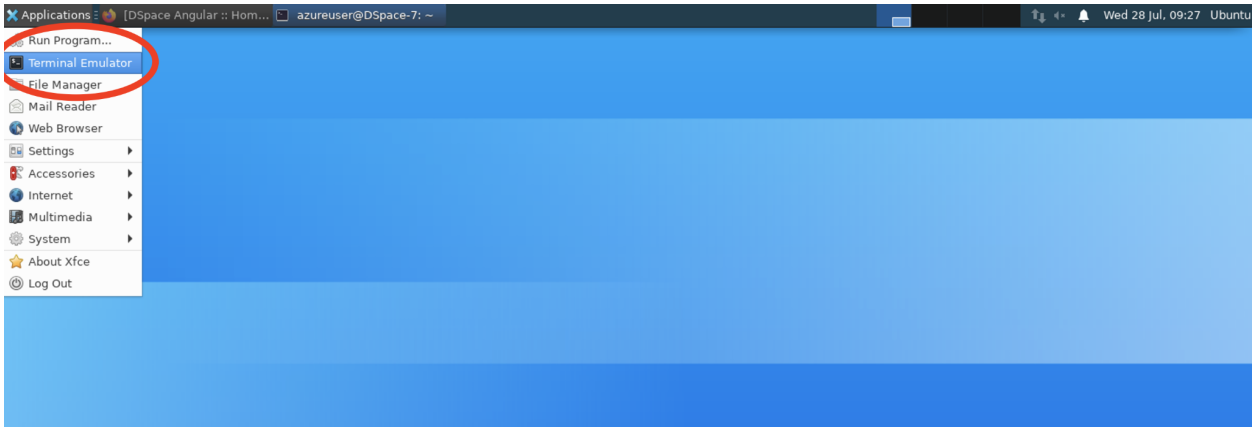
Connect to the VM using RDP



Open the RDP and log in using the username and password that you created in step 1

Install A browser within the desktop

Open cmd prompt



Install FireFox

```
sudo apt-get install firefox
```

Install Docker

```
sudo apt-get install docker.io

# Run this only when you cannot access the docker command even
with sudo
sudo chmod 666 /var/run/docker.sock
```

Install docker-compose

```
# Grab it from latest
sudo curl -L
"https://github.com/docker/compose/releases/download/1.24.0/docker-compo
se-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

# Chmod it
sudo chmod +x /usr/local/bin/docker-compose

# Check version
docker-compose -v
```

Install Yarn

```
curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo
apt-key add -

echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo
tee /etc/apt/sources.list.d/yarn.list

sudo apt-get update
sudo apt-get install yarn -y
```

Install NodeJs

```
curl -fsSL https://deb.nodesource.com/setup_current.x | sudo -E
bash -
sudo apt-get install -y nodejs
```

2. DSPACE OBJECTS

Dspace mainly comprises 3 objects.

- Community
- Collection
- Items

For each of these items, they can have multiple sub items of the same kind with different kinds of metadata. The following table highlights the types of relation the items have.

Item name	Possible Parent	Number of siblings	Number of childrens

Community	- Community	Multiple	Multiple
Collection	- Community - Collection	Multiple	Multiple
Items	- Collection	Multiple	None

For the sake of mapping the Dspace Objects to the current existing portal study. The following shows the plan.

Object	Mapped Item in Portal	Possible contained objects
Community	Tenant	- Tenant Name
Collection	Fixed: “Studies”	Multiple items
Item	Study	- Study dashboard details - Study metadata - Study tags

Each Community has a fixed Collection titled Studies. This collection would hold multiple items which are studies

Note: That Dspace would automatically create additional metadata values even without specifying it during creation. Metadata fields such as `dc.title` are required to have its value set in order for the `name` value to be set

The next section would show how the creation is done. A new schema is required in order to define new metadata fields.
For each step in creating a tenant and study, the main flow goes like this.

1. Create the schema for that particular object
2. Create metadata fields for the newly created schema
3. Create the object itself using the metadata fields created for the schema

Schema Creation

The following shows how to create schemas using Dspace API. Take note that for each schema creation, authorisation with admin user and the CSRF Token is required.

Schema creation is required if the user wants to make use of custom metadata. For example, currently Dspace uses the default `DC` schema. This schema holds multiple fields such as the

`Title` field. The metadata would then be referenced by the object as shown in the image below.

```
"metadata": {
    "dc.description": [
        {
            "value": "An example description",
            "language": "en",
            "authority": null,
            "confidence": -1,
            "place": 0
        }
    ],
    "dc.title": [
        {
            "value": "test new title",
            "language": null,
            "authority": null,
            "confidence": -1,
            "place": 0
        }
    ]
```

POST

```
http://localhost:8080/server/api/core/metadataschemas
```

Body Data

```
{
  "prefix": "tenant",
  "namespace": "http://data4life-tea.com",
  "type": "metadataschema"
}
```

Field Info

Fields	Meaning	Example	Notes
Prefix	The prefix used when referenced by objects	dc, tenant, study	Needs to be unique (Cant be created before)

namespace	Can be any url	http://datalife.com	Needs to be unique
type	Type of schema	metadataschema	

Take note of the id generated after creation. This ID is required for creating metadata in the next step.

Checking of existing schema

GET

```
http://localhost:8080/server/api/core/metadataschemas
```

Creation of metadata fields using schema created

Dspace makes use of the schemaID query parameters to know which schema the user is trying to create the field for. The ID can be retrieved by making a GET request to retrieve the available schemas. Metadata fields can have at most up to 2 types of nesting. The following shows possible metadata field names

- study.metadata
- study.metadata.token-gen
- dc.title
- study.dashboard.name

POST

```
http://localhost:8080/server/api/core/metadatafields?schemaId=<id>
```

Body Data

```
{
  "element": "metadata",
  "qualifier": "token-gen",
  "scopeNote": "The token use for study",
  "type": "metadatafield"
}
```

Field Info

Fields	Meaning	Example	Notes
element	The first parent field	title, metadata, detail, dashboard	

qualifier	The child field	token-gen, name, desc	Can be left as NULL
scopeNote	A string field which give a brief description of the field	“The token use for study”	
type	Type of field	metadatafield	

Creating the object with newly Created Metadata field

Depending on the object, there are different endpoints that are allocated to each object. You can defined the metadata created in the previous step under the metadata tag as shown in the example below. Note that even though unspecified, Dspace would automatically create extra fields using `DC` schema by default (Regardless of POST command)

Community

POST

```
http://localhost:8080/server/api/core/communities
```

Data

```
{
  "name": "Test-1",
  "metadata": {
    "tenant.details.name": [
      {
        "value": "Test-1",
        "language": null,
        "authority": null,
        "confidence": -1
      }
    ]
  }
}
```

Collection

POST

```
http://localhost:8080/server/api/core/collections?parent=<CommunityID>
```

Data

```
{
```

```
  "name": "Study 1",
  "metadata": {
    "study.details.email": [
      {
        "value": "charlotte.lim@data4life-asia.care",
        "language": null,
        "authority": null,
        "confidence": -1
      }
    ]
  }
}
```


Item

POST

```
http://localhost:8080/server/api/core/items?owningCollection=<CollectionID>
```

Data


```
{
  "name": "Study tag",
  "metadata": {
    "study.study_tag.name": [
      {
        "value": "diabetes, cancer, tuberculosis, heheXDdiseases",
        "language": "en",
        "authority": null,
        "confidence": -1
      }
    ]
  },
  "inArchive": true,
  "discoverable": false,
  "withdrawn": false,
  "type": "item"
}
```

Appendix

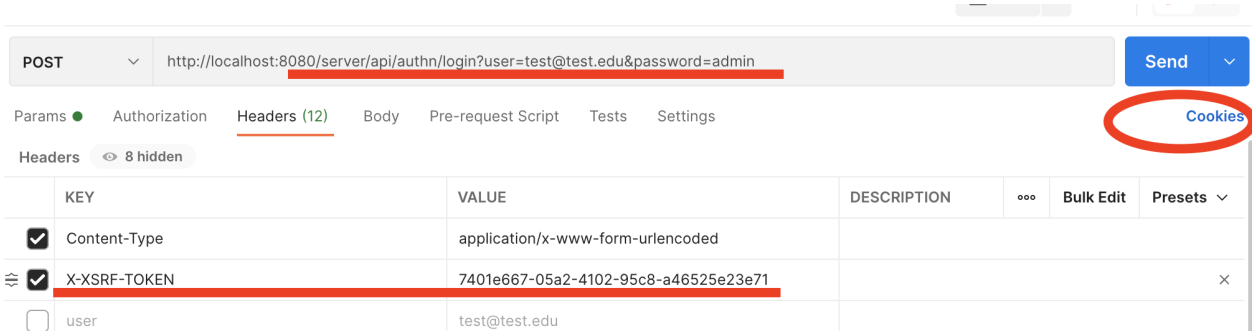
Metadata field	Schema	Used in	Corresponding Alp database	Usage
dc.title	dc	Item, Community, Collection	Study and Tenant titles	To store the name of the object
dc.description	dc	Item, Community, Collection	Study and Tenant Details	To store object description
study.study_tag.name	study	item	Study Tag	To store all the tags in string format

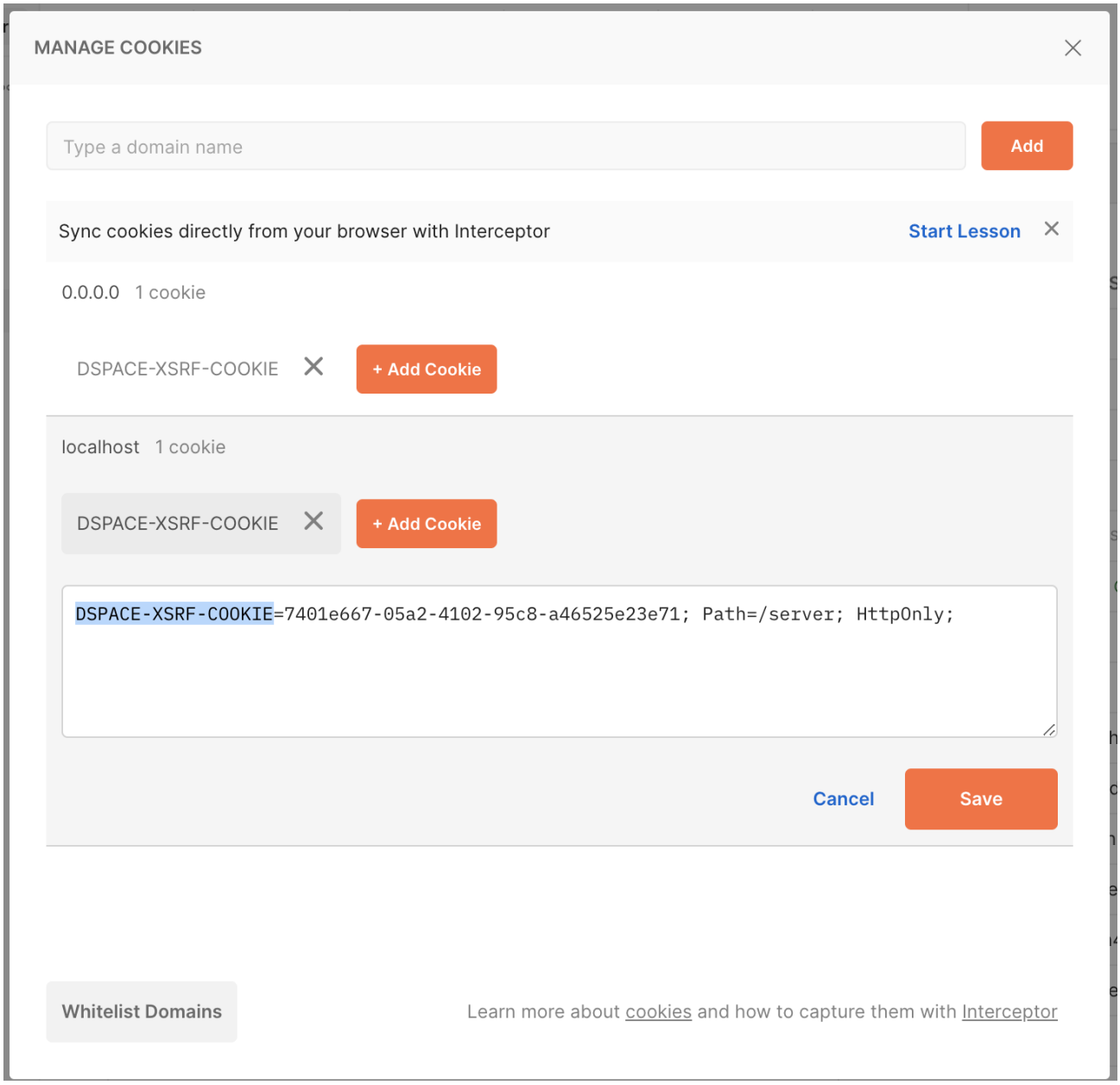
3. Postman: Connecting with REST API

1. Login/ Authentication

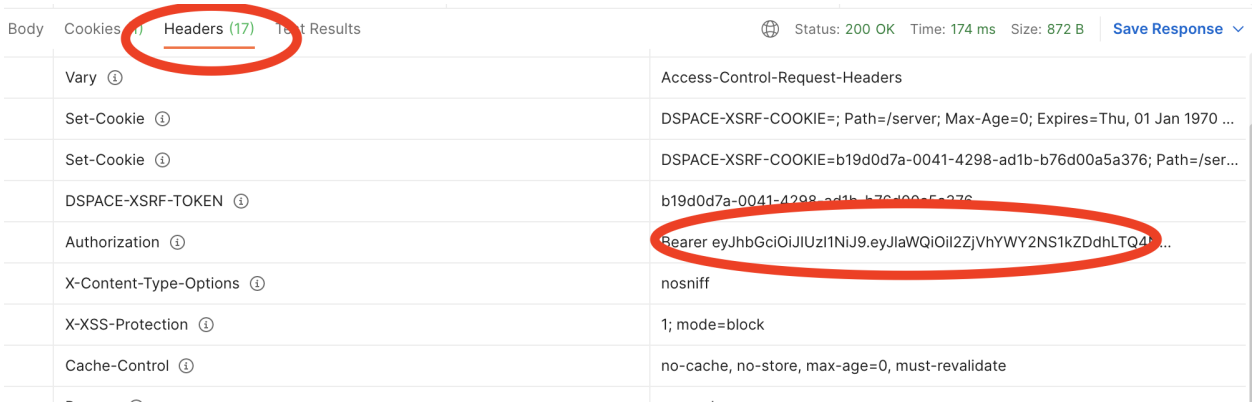
The end point for authentication is POST at
`http://localhost:8080/server/api/authn/login`
In order to login and bypass csrf, we need to ensure that `X-XSRF-TOKEN` exists in the header and `DSPACE-XSRF-COOKIE` exists in the cookies setting. The two values must be the same.

Take note of the 3 things needed to log in





Once POST is completed, check the headers returned. The authentication needed is in the bearer column. Use this as authentication in the future.



4. Logging

Rest API Logging

To see the logs for incoming rest api calls to the server.
The following containers must be up

- Solr
- Dspace-test
- Dspace db

Check current docker container's id

```
docker ps
```

Enter Dspace's main docker container using bash

```
docker exec -it <container id> /bin/bash
```

Check the log file

```
cd logs

#Check available logs
ls

# Tail the logs to see
tail -f <filename>
```

DSpace Logging

Enter Dspace's main docker container using bash

```
docker exec -it <container id> /bin/bash
```

Check Dspace.log

```
tail -f /dspace/log/dspace.log
```

6. Others: Useful Commands

Dspace-CLI (Docker-compose)

All commands interacting with Dspace must be run in the front-end repository

Creating a user

```
docker-compose -p d7 -f docker/cli.yml run --rm dspace-cli user
--add --email user@test.com -g John -s User --password userpass
```

Create admin default user

```
docker-compose -p d7 -f docker/cli.yml run --rm dspace-cli
create-administrator -e test@test.edu -f admin -l user -p adminpass -c
en
```

user: test@test.edu
password: adminpass

View all user

```
docker-compose -p d7 -f docker/cli.yml run --rm dspace-cli user
--list
```

Docker Commands

To clear containers

```
docker rm -f $(docker ps -a -q)
```

To clear images

```
docker rmi -f $(docker images -a -q)
```

To clear volumes

```
docker volume rm $(docker volume ls -q)
```

To clear networks

```
docker network rm $(docker network ls | tail -n+2 | awk '{if($2 !~ /bridge|none|host/){ print $1 }}')
```

Clear Cache

```
docker builder prune
```

Remove all images/ volume/ networks

```
docker system prune -a
```

Check Docker Status

```
docker ps
```

Check containers

```
docker container ps
```

Access docker container

```
docker exec -it <container id> /bin/bash
```

Check Logs

```
docker logs -f dspace
```

CMD

Check process

```
lsof -i :<Port>
```