DELL Technologies

Piper Technical Workshop

Containers and Kubernetes

Abstract

This document is provided to assist attendees with completing the appropriate labs to apply the concepts and knowledge learnt throughout the technical workshop program. It is not intended to be used or distributed in isolation and may not contain all required information.

April 2021

Student Lab Guide

DELL Technologies

Revisions

Version	Date	Description
0.1	May 2020	Initial draft
0.2	June 2020	Updates- Labs documented with detailed steps and additional screenshots
0.3	July 2020	Added Lab 4 – PV's with Isilon
0.4	July 2020	Added Lab 5 – TKGi
0.5	August 2020	Moved K8s Cluster to CentOS VM (https://192.168.1.11:6443)
0.6	September 2020	Replaced TKGi with vSphere 7 Kubernetes
0.7	November 2020	All K8s labs now using vSphere with Tanzu (removed lab 5 – TKGi)
0.8	December 2020	Added lab 5 – PowerProtect Data Manager with Data Domain
1.0	April 2021	Isolated labs for 2021 program and on-demand delivery

DELL Technologies

Table of Contents

DELL Technologies

Lab 1: Docker

Module Objectives:

- ❑ Learn about Docker and Docker Desktop
- ❑ Build a Docker image
- ❑ Run a Docker container
- ❑ Test its functionality

DELL Technologies

Lab Exercise: Build a docker image and run as a docker container

1. Clone the following repository (if not already);

   https://github.com/theocrithary/Piper-2021/tree/main/Containers%20%26%20Kubernetes/Lab%2001%20-%20Docker

2. Review the files contained in the “Containers & Kubernetes/Lab 01 - Docker/” directory

   - app.py
   - Dockerfile
   - requirements.txt

3. Open a command prompt and cd into the local directory containing your files

   > cd "d:\Piper-2021\Containers & Kubernetes\Lab 01 - Docker"

4. Type the following command and observe the output {NOTE- don’t miss the dot (.) at the end of the command below}

   > docker build -t helloworld .

5. Type this command to check that the container image was built successfully

   > docker image ls

6. Type this command to start the container and run the Python application

   > docker run -p 7000:7000 helloworld

7. Open a chrome browser and go to the following URL: http://localhost:7000

8. Press CTRL +C on command prompt to break the running webapp connection. The container will still continue running and the webapp is still accessible. If you want to stop and delete the container, please run the following;

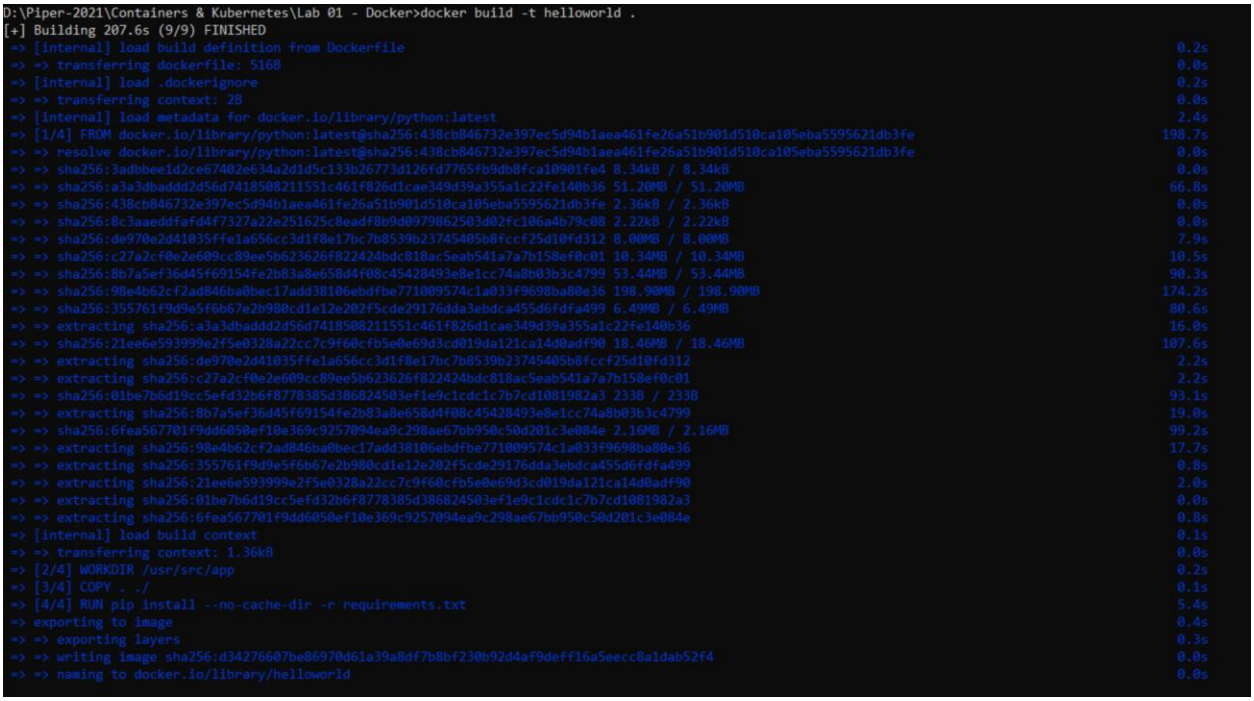
   > docker ps -q -f "ancestor=helloworld" ***(this command will output the container id)***
   > docker kill <<replace with container id>> && docker rm <<replace with container id>>

DELL Technologies

Retrospective: The results

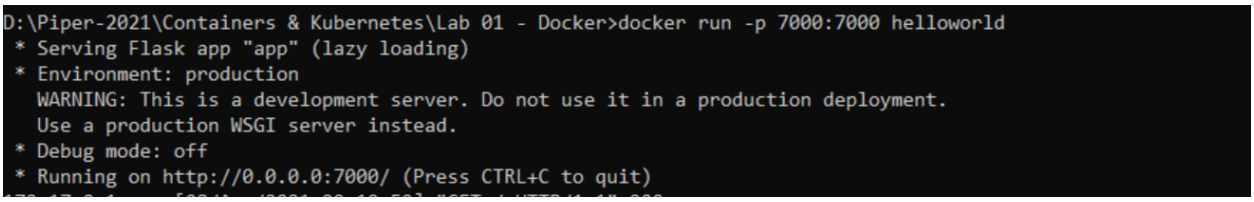
- ❑ Built a Docker image

If you successfully built a docker image, you should see the below result in the command prompt



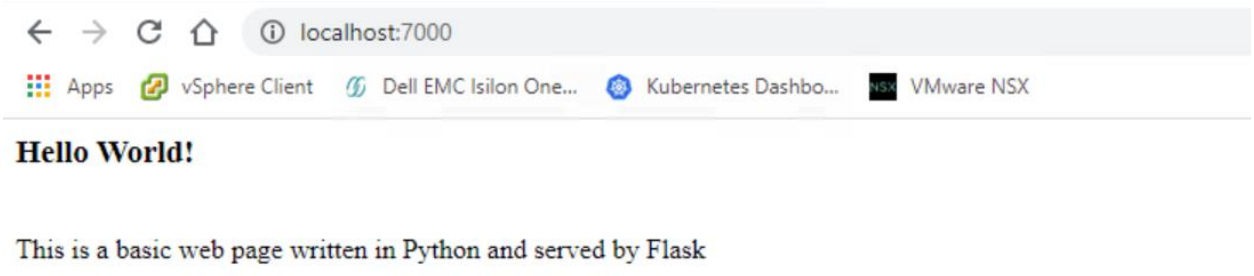
- ❑ Ran a container instance from a Docker image

If you successfully ran the docker container, you should see the below results on the command prompt



- ❑ Tested the web app

If you successfully ran the docker container, you should see the below results on the web browser or CLI



DELL Technologies

Lab 2: Kubernetes

Module Objectives:

- ❑ Learn about Kubernetes and kubectl command line tool
- ❑ Create a Kubernetes pod from a Docker image published on DockerHub
- ❑ Create a service to expose the port for the application
- ❑ Test its functionality

DELL Technologies

Lab Exercise: Create a Kubernetes deployment from a Docker image and expose a service

1. Clone the following repository (if not already);

   https://github.com/theocrithary/Piper-2021/tree/main/Containers%20%26%20Kubernetes/Lab%2002%20-%20Kubernetes

2. Review the files contained in the “Containers & Kubernetes/Lab 02 - Kubernetes" directory

   - app.py
   - Dockerfile
   - requirements.txt

   Note: This version has changed port to 7001 in app.py and DockerFile

3. Open a command prompt and cd into the local directory containing your files (note that it’s changed to “Lab 02 - Kubernetes” folder)

   > d:

   > cd "D:\Piper-2021\Containers & Kubernetes\Lab 02 - Kubernetes"

```
D:\Piper-2021\Containers & Kubernetes\Lab 02 - Kubernetes>dir
 Volume in drive D is Data
 Volume Serial Number is F6EC-39A7

 Directory of D:\Piper-2021\Containers & Kubernetes\Lab 02 - Kubernetes

04/08/2021  10:07 PM    <DIR>          .
04/08/2021  10:07 PM    <DIR>          ..
04/08/2021  10:07 PM               738 app.py
04/08/2021  10:07 PM               477 Dockerfile
04/08/2021  10:07 PM                 6 requirements.txt
               3 File(s)          1,221 bytes
               2 Dir(s)  11,840,118,784 bytes free
```

4. Login to docker with your DockerHub credentials

   > docker login

```
D:\Piper-2021\Containers & Kubernetes\Lab 02 - Kubernetes>docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: theocrithary
Password:
Login Succeeded
```

5. Build docker container with your DockerHub credentials

   Note: don’t miss the dot (.) at the end of the command below

   > docker build -t <YourDockerHubID>/helloworld .

```
D:\Piper-2021\Containers & Kubernetes\Lab 02 - Kubernetes>docker build -t theocrithary/helloworld .
[+] Building 8.9s (9/9) FINISHED
 => [internal] load build definition from Dockerfile                                                0.1s
 => => transferring dockerfile: 516B                                                                0.0s
 => [internal] load .dockerignore                                                                   0.1s
 => => transferring context: 2B                                                                     0.0s
 => [internal] load metadata for docker.io/library/python:latest                                    1.3s
 => [internal] load build context                                                                   0.1s
 => => transferring context: 1.34kB                                                                 0.0s
 => [1/4] FROM docker.io/library/python:latest@sha256:438cb846732e397ec5d94b1aea461fe26a51b901d510ca105eba5595621db3fe  0.0s
 => CACHED [2/4] WORKDIR /usr/src/app                                                               0.0s
 => [3/4] COPY . ./                                                                                 0.0s
 => [4/4] RUN pip install --no-cache-dir -r requirements.txt                                        6.6s
 => exporting to image                                                                              0.6s
 => => exporting layers                                                                             0.5s
 => => writing image sha256:3e104a8bf26a242733c89c81d344fed81bae6bd34fddf98a7273cf0524e3c9d3        0.0s
 => => naming to docker.io/theocrithary/helloworld                                                  0.0s
```

6. Push the docker image to repository on Docker Hub

   > docker push <YourDockerHubID>/helloworld

```
D:\Piper-2021\Containers & Kubernetes\Lab 02 - Kubernetes>docker push theocrithary/helloworld
The push refers to repository [docker.io/theocrithary/helloworld]
63203e5902c8: Pushed
f901874bd8d8: Pushed
e359db8b3902: Pushed
74f4273bc66a: Mounted from library/python
1a49173d2880: Mounted from library/python
2e5df26b58f4: Mounted from library/python
cddf7c29c7e8: Mounted from library/python
b11edd36d511: Mounted from library/python
fd713e46419c: Mounted from library/python
d381f471a064: Mounted from library/python
ee77c16e08cd: Mounted from library/python
da0bfb47817e: Mounted from library/python
latest: digest: sha256:42ba7e7a24df08ab94ec550d9a7ae6ba5aa4dd4a53342a27a27e40c9b58477cc size: 2843
```

7. Login to the Tanzu Kubernetes Cluster provided for this lab

   > kubectl vsphere login --server=192.168.1.120 --tanzu-kubernetes-cluster-name tkc-cluster-01 --tanzu-kubernetes-cluster-namespace piper --vsphere-username administrator@vsphere.local --insecure-skip-tls-verify

   Note: Enter the password when prompted: Password123!

```
D:\Piper-2021\Containers & Kubernetes\Lab 02 - Kubernetes>kubectl vsphere login --server=192.168.1.120 --tanzu-kubernetes-cluster-name tkc-cluster-01 --tanzu
-kubernetes-cluster-namespace piper --vsphere-username administrator@vsphere.local --insecure-skip-tls-verify

Password:
Logged in successfully.

You have access to the following contexts:
   192.168.1.120
   kubernetes-admin@kubernetes
   piper
   tkc-cluster-01

If the context you wish to use is not in this list, you may need to try
logging in again later, or contact your cluster administrator.

To change context, use `kubectl config use-context <workload name>`
```

DELL Technologies

8. Change context to the tkc-cluster-01 cluster we will be using for the labs

   > kubectl config use-context tkc-cluster-01

```
D:\Piper-2021\Containers & Kubernetes\Lab 02 - Kubernetes>kubectl config use-context tkc-cluster-01
Switched to context "tkc-cluster-01".
```

9. Check that the Kubernetes cluster is running and available

   > kubectl cluster-info

```
D:\Piper-2021\Containers & Kubernetes\Lab 02 - Kubernetes>kubectl cluster-info
Kubernetes master is running at https://192.168.140.210:6443
KubeDNS is running at https://192.168.140.210:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

10. Run the container as a K8s pod using the docker image previously created

    > kubectl run helloworld --image=docker.io/<YourDockerHubID>/helloworld

```
D:\Piper-2021\Containers & Kubernetes\Lab 02 - Kubernetes>kubectl run helloworld --image=docker.io/theocrithary/helloworld
pod/helloworld created
```

11. Check that a pod was built and is running successfully

    > kubectl get pods

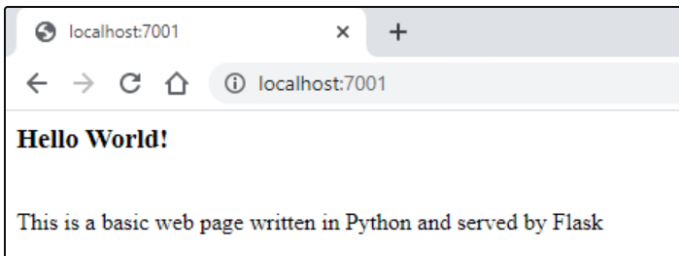
```
D:\Piper-2021\Containers & Kubernetes\Lab 02 - Kubernetes>kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
helloworld            1/1     Running   0          53s
snapshot-controller-0   1/1     Running   2          149d
```

12. Temporarily expose port 7001 and allow external access into the Kubernetes cluster via a proxy

    > kubectl port-forward helloworld 7001:7001

```
D:\Piper-2021\Containers & Kubernetes\Lab 02 - Kubernetes>kubectl port-forward helloworld 7001:7001
Forwarding from 127.0.0.1:7001 -> 7001
Forwarding from [::1]:7001 -> 7001
```

13. Open a chrome browser and go to the following URL: http://localhost:7001



14. Press CTRL +C on command prompt to break the running webapp port-forward proxy.

15. If you want to delete the pod, please run the following;

    > kubectl delete pod helloworld

DELL Technologies

Retrospective: The results

- ❑ Learnt about Kubernetes and used Tanzu Kubernetes via the kubectl cli
- ❑ Created a Kubernetes pod from a Docker Hub image
- ❑ Created a temporary port-forward to expose port 7001
- ❑ Tested the web app

DELL Technologies

Lab 3: Containerising an App

Module Objectives:

- ❑ Apply knowledge we learnt in previous labs
- ❑ Deploy a 2-tier app to Docker
- ❑ Forward port to allow local testing
- ❑ Deploy a 2-tier app to a Kubernetes cluster
- ❑ Expose the service port
- ❑ Test its functionality

DELL Technologies

Lab Exercise Part 1: Package a 2-tier app and run locally using Docker Desktop

Complete the below steps to demonstrate your understanding of the tools and concepts required for the remaining lab exercises;

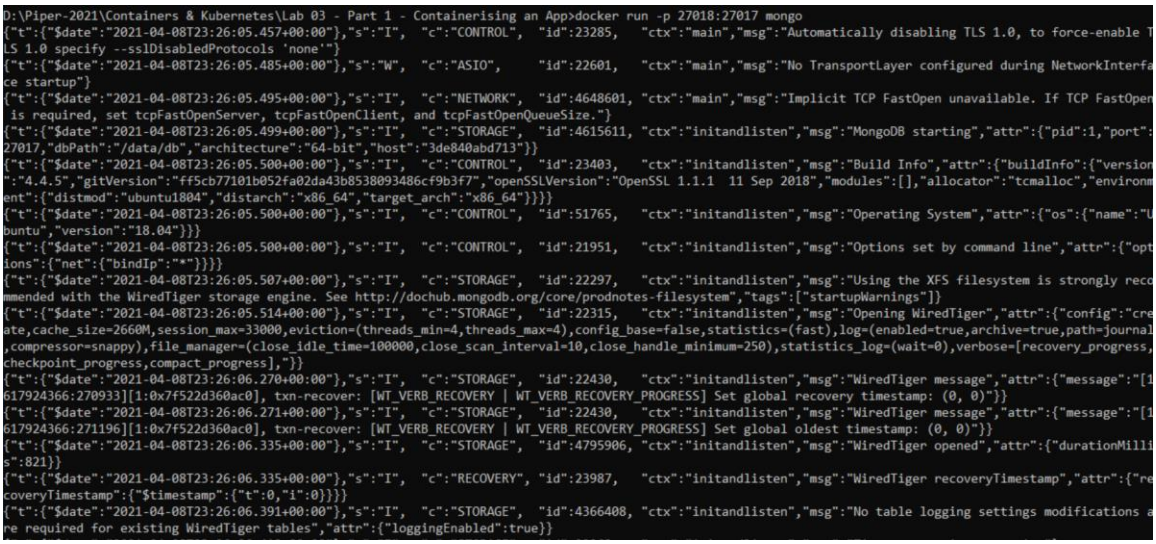
1. Get the mongodb image from Docker Hub

   > docker pull mongo

```
D:\Piper-2021\Containers & Kubernetes\Lab 03 - Part 1 - Containerising an App>docker pull mongo
Using default tag: latest
latest: Pulling from library/mongo
6e0aa5e7af40: Pull complete
d47239a868b3: Pull complete
49cbb10cca85: Pull complete
9729d7ec22de: Pull complete
7b7fd72268d8: Pull complete
5e2934dacaf5: Pull complete
bf9da24d4b2c: Pull complete
d2f8c3715616: Pull complete
e9f96a4a45b0: Pull complete
bd66718f31e2: Pull complete
41ed4d1a1542: Pull complete
7336dfc228e2: Pull complete
Digest: sha256:67018ee2847d8c35e8c7aeba629795d091f93c93e23d3d60741fde74ed6858c4
Status: Downloaded newer image for mongo:latest
docker.io/library/mongo:latest
```

2. Type this command to start the mongo DB container

   > docker run -p 27018:27017 mongo



   > Ctrl + C to break from command and run in the background

   Note: The change in exposed port to 27018. This is to avoid conflicts with the existing MongoDB server running locally at 27017

3. Type these commands to check that the container was deployed to Docker

   > docker container ls

```
D:\Piper-2021\Containers & Kubernetes\Lab 03 - Part 1 - Containerising an App>docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                     NAMES
3de840abd713   mongo     "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:27018->27017/tcp   condescending_carson
```

4. Type this command to get the IP address of the docker container running mongodb

   > docker container inspect {container name} | findstr IPAddress

```
D:\Piper-2021\Containers & Kubernetes\Lab 03 - Part 1 - Containerising an App>docker container inspect condescending_carson | findstr IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "172.17.0.2",
                    "IPAddress": "172.17.0.2",
```

   NOTE- Take note of this IP address, we will use it later in our code to create a database connection

```
"IPAddress": "172.17.0.2",
```

5. Clone the following repository (if not already); https://github.com/theocrithary/Piper-2021/tree/main/Containers%20%26%20Kubernetes/Lab%2003%20-%20Part%201%20-%20Containerising%20an%20App

6. Copy all files from 'Lab 03 - Part 1 - Containerising an App' to your working project directory (e.g. D:\MyProject)

7. Rename the **config-example.py** file to **config.py**, then edit the file as below

   - Replace with your ECS account credentials obtained from ecstestdrive.com

```
1  ecs_test_drive = {
2      'ecs_endpoint_url' : 'https://object.ecstestdrive.com',
3      'ecs_access_key_id' : '1234-your-unique-number-5678@ecstestdrive.emc.com',
4      'ecs_secret_key' : 'your-long-secret-key-from-ECS-testdrive-portal',
5      'ecs_bucket_name' : 'photo-album'
6  }
```

8. Edit the models.py file on line 29 to reflect the IP address of your mongodb container

```
29      client = MongoClient('172.17.0.2:27017')
```

   Notice the port is 27017? This is because the Docker container has an internal network that uses port 27017 on the 172.17.0.0 network. We will be connecting our application server directly to the MongoDB container via this 'internal' network.

9. Type the following commands and observe the outputs {NOTE- don't miss the dot (.) at the end of the command below}

   > docker build -t photo-album .

```
D:\MyProject>docker build -t photo-album .
[+] Building 71.7s (9/9) FINISHED
 => [internal] load .dockerignore                                                                                0.1s
 => => transferring context: 2B                                                                                  0.0s
 => [internal] load build definition from Dockerfile                                                             0.1s
 => => transferring dockerfile: 32B                                                                              0.0s
 => [internal] load metadata for docker.io/library/python:latest                                                 1.3s
 => [1/4] FROM docker.io/library/python:latest@sha256:0a9325e9adb51dd65ec915036e270eae33087918fc1254f6a22a436741600b17   0.0s
 => [internal] load build context                                                                                0.1s
 => => transferring context: 4.60kB                                                                              0.0s
 => CACHED [2/4] WORKDIR /usr/src/app                                                                            0.0s
 => [3/4] COPY . ./                                                                                              0.0s
 => [4/4] RUN pip install --no-cache-dir -r requirements.txt                                                    68.5s
 => exporting to image                                                                                           1.6s
 => => exporting layers                                                                                          1.6s
 => => writing image sha256:1764f6ee0886a9270aeb31e5cef0445fd807a225f64ab706b7063227e2336cfc                     0.0s
 => => naming to docker.io/library/photo-album                                                                   0.0s
```

10. Type this command to check that the container image photo-album was built and mongo was downloaded from docker hub

    > docker image ls

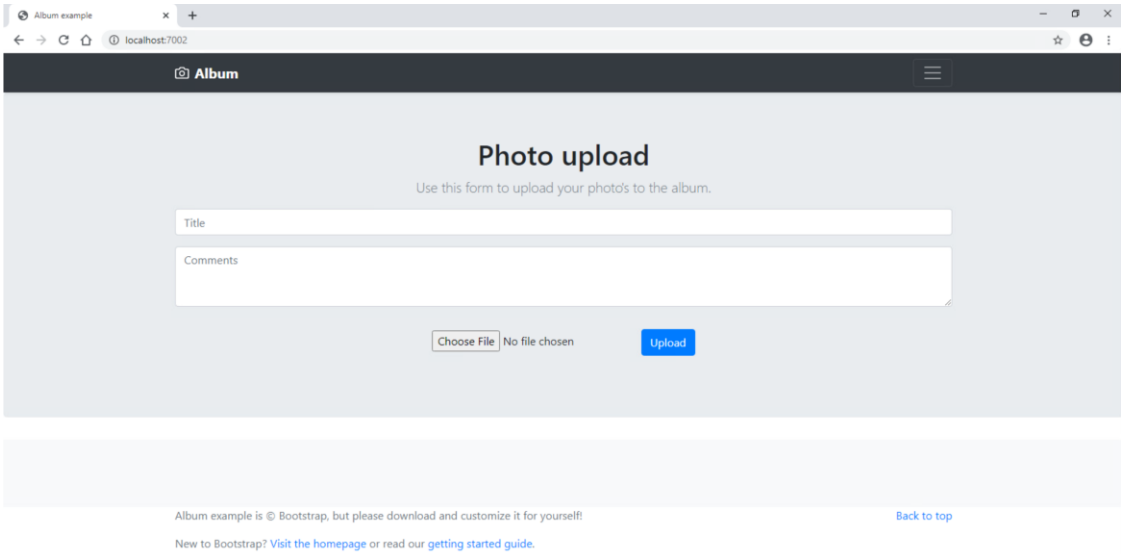
```
D:\MyProject>docker image ls
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
photo-album               latest              1764f6ee0886        19 seconds ago      946MB
<none>                    <none>              a2cc4094c44d        7 minutes ago       946MB
theocrithary/helloworld   latest              4a1e123ea59b        7 hours ago         883MB
mongo                     latest              ba0c2ff8d362        3 weeks ago         492MB
```

11. Type this command to start the mongo DB container

    > docker run -p 7002:7002 photo-album

```
D:\MyProject>docker run -p 7002:7002 photo-album
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:7002/ (Press CTRL+C to quit)
```

12. Open a browser and go to the following URL: http://localhost:7002



DELL Technologies

Lab Exercise Part-2: Deploy a Docker based 2-tier app to a Kubernetes cluster

Clean up and delete any Docker container instances you deployed in the previous labs

1. Login to the Tanzu Kubernetes Cluster provided for this lab

   > kubectl vsphere login --server=192.168.1.120 --tanzu-kubernetes-cluster-name tkc-cluster-01 --tanzu-kubernetes-cluster-namespace piper --vsphere-username administrator@vsphere.local --insecure-skip-tls-verify

   Note: Enter the password when prompted: Password123!

```
C:\Users\demouser>kubectl vsphere login --server=192.168.1.120 --tanzu-kubernetes-cluster-name tkc-cluster-01 --tanzu-kubernetes-cluster-namespace piper --vsphere-username
 administrator@vsphere.local --insecure-skip-tls-verify

Password:
Logged in successfully.

You have access to the following contexts:
   192.168.1.120
   piper
   tkc-cluster-01

If the context you wish to use is not in this list, you may need to try
logging in again later, or contact your cluster administrator.

To change context, use `kubectl config use-context <workload name>`
```

2. Change context to the tkc-cluster-01 cluster we will be using for the labs

   > kubectl config use-context tkc-cluster-01

```
C:\Users\demouser>kubectl config use-context tkc-cluster-01
Switched to context "tkc-cluster-01".
```

3. Check that the Kubernetes cluster is running and if there are any existing deployments

   > kubectl cluster-info

```
D:\MyProject>kubectl cluster-info
Kubernetes master is running at https://192.168.140.210:6443
KubeDNS is running at https://192.168.140.210:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

4. Check if there are any existing deployments that need to be deleted

   > kubectl get deployment

5. If this is a new TKG cluster, we need to allow permissions for the administrator@vsphere.local account to use the service accounts and create deployments. We do this through a rolebinding to leverage existing groups and default pod security policies.

   First check if the rolebinding already exists;

   > kubectl get rolebinding

```
D:\MyProject>kubectl get rolebinding
NAME                                            AGE
rolebinding-default-privileged-sa-ns_default    7h3m
```

   If it does not, then create the rolebinding as per below;

> kubectl create rolebinding rolebinding-default-privileged-sa-ns_default --namespace=default --clusterrole=psp:vmware-system-privileged --group=system:serviceaccounts

6. Now, let's create a new deployment for our Mongo DB database instance

   > kubectl create deployment mongo --image=mongo

```
D:\MyProject>kubectl create deployment mongo --image=mongo
deployment.apps/mongo created
```

7. Type these commands to check that the container was deployed to the Kubernetes cluster

   > kubectl get deployment (this may take several minutes to complete)

```
D:\MyProject>kubectl get deployment
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
mongo   1/1     1            1           75s
```

   > kubectl get pods

```
D:\MyProject>kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
mongo-79d55cddb5-27z5g   1/1     Running   0          82s
```

8. Type this command to expose port 27017 to allow our application to connect to the database port

   > kubectl expose deployment mongo --type=LoadBalancer --port=27017

```
D:\MyProject>kubectl expose deployment mongo --type=LoadBalancer --port=27017
service/mongo exposed
```

9. Type this command to retrieve the cluster IP address of the mongo deployment

   > kubectl get svc

```
D:\MyProject>kubectl get svc
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)           AGE
kubernetes   ClusterIP      198.51.100.1    <none>           443/TCP           4d
mongo        LoadBalancer   198.51.100.18   192.168.140.211  27017:31714/TCP   7s
supervisor   ClusterIP      None            <none>           6443/TCP          4d
```

10. Clone the following repository (if not already); https://github.com/theocrithary/Piper-2021/tree/main/Containers%20%26%20Kubernetes/Lab%2003%20-%20Part%202%20-%20Deploying%20to%20Kubernetes

11. Copy all files from 'Lab 03 - Part 2 - Deploying to Kubernetes' to your working project directory (e.g. D:\MyProject)

12. Ensure you still have the config.py file from Part 1 of this lab. If not, copy it from the Part 1 folder or follow step 7 in Part 1 to recreate the file.

13. Edit the models.py file on line 29 and add the IP address you obtained from the mongo svc

```
28      # Set the database target to your local MongoDB instance
29      client = MongoClient('198.51.100.18:27017')
```

14. Go back to your command prompt and ensure you are in the directory of your project

    > cd D:\MyProject

15. Build the docker image and push it to Docker Hub

    NOTE: don't miss the dot (.) at the end of the command below

    > docker build -t <dockerhubID>/photo-album-k8s .

```
D:\MyProject>docker build -t theocrithary/photo-album-k8s .
[+] Building 1.5s (9/9) FINISHED
 => [internal] load .dockerignore                                                                                0.1s
 => => transferring context: 2B                                                                                  0.0s
 => [internal] load build definition from Dockerfile                                                             0.1s
 => => transferring dockerfile: 32B                                                                              0.0s
 => [internal] load metadata for docker.io/library/python:latest                                                 1.2s
 => [1/4] FROM docker.io/library/python:latest@sha256:0a9325e9adb51dd65ec915036e270eae33087918fc1254f6a22e436741600b17  0.0s
 => [internal] load build context                                                                                0.0s
 => => transferring context: 482B                                                                                0.0s
 => CACHED [2/4] WORKDIR /usr/src/app                                                                            0.0s
 => CACHED [3/4] COPY . ./                                                                                       0.0s
 => CACHED [4/4] RUN pip install --no-cache-dir -r requirements.txt                                              0.0s
 => exporting to image                                                                                           0.1s
 => => exporting layers                                                                                          0.0s
 => => writing image sha256:f877e9e4b4f89a3937f5eefc23dfa409261c5c32b931f06e9b913041955f270b                     0.0s
 => => naming to docker.io/theocrithary/photo-album-k8s                                                          0.0s
```

    > docker images

```
D:\MyProject>docker images
REPOSITORY                    TAG              IMAGE ID       CREATED          SIZE
theocrithary/photo-album-k8s  latest           f877e9e4b4f8   31 seconds ago   946MB
photo-album                   latest           1764f6ee0886   2 hours ago      946MB
<none>                        <none>           a2cc4094c44d   2 hours ago      946MB
theocrithary/helloworld       latest           4a1e123ea59b   9 hours ago      883MB
mongo                         latest           ba0c2ff8d362   3 weeks ago      492MB
```

    > docker push <dockerhubID>/photo-album-k8s

```
D:\MyProject>docker push theocrithary/photo-album-k8s
The push refers to repository [docker.io/theocrithary/photo-album-k8s]
234314d70de5: Pushed
188c889ea78d: Pushed
a134306bbb39: Mounted from theocrithary/helloworld
3c12301cd590: Mounted from theocrithary/helloworld
56cede1eca1a: Mounted from theocrithary/helloworld
3bda9ad494e0: Mounted from theocrithary/helloworld
a90437ea99dc: Mounted from theocrithary/helloworld
9fa091ee2596: Mounted from theocrithary/helloworld
91736fdb1e15: Mounted from theocrithary/helloworld
dcc5e9acdfb7: Mounted from theocrithary/helloworld
064c3b400650: Mounted from theocrithary/helloworld
bd986cba86c6: Mounted from theocrithary/helloworld
latest: digest: sha256:f727c5cb652cf9c158bc736484ef224a769ead431f42c41ff157c478178e879d size: 2845
```

16. Deploy your image to the Kubernetes cluster

    > kubectl create deployment photo-album-k8s --image=<dockerhubID>/photo-album-k8s

```
D:\MyProject>kubectl create deployment photo-album-k8s --image=theocrithary/photo-album-k8s
deployment.apps/photo-album-k8s created
```

>kubectl get deployment

```
D:\MyProject>kubectl get deployment
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
mongo             1/1     1            1           23m
photo-album-k8s   1/1     1            1           21s
```

17. Expose port 7003 for local access to your application

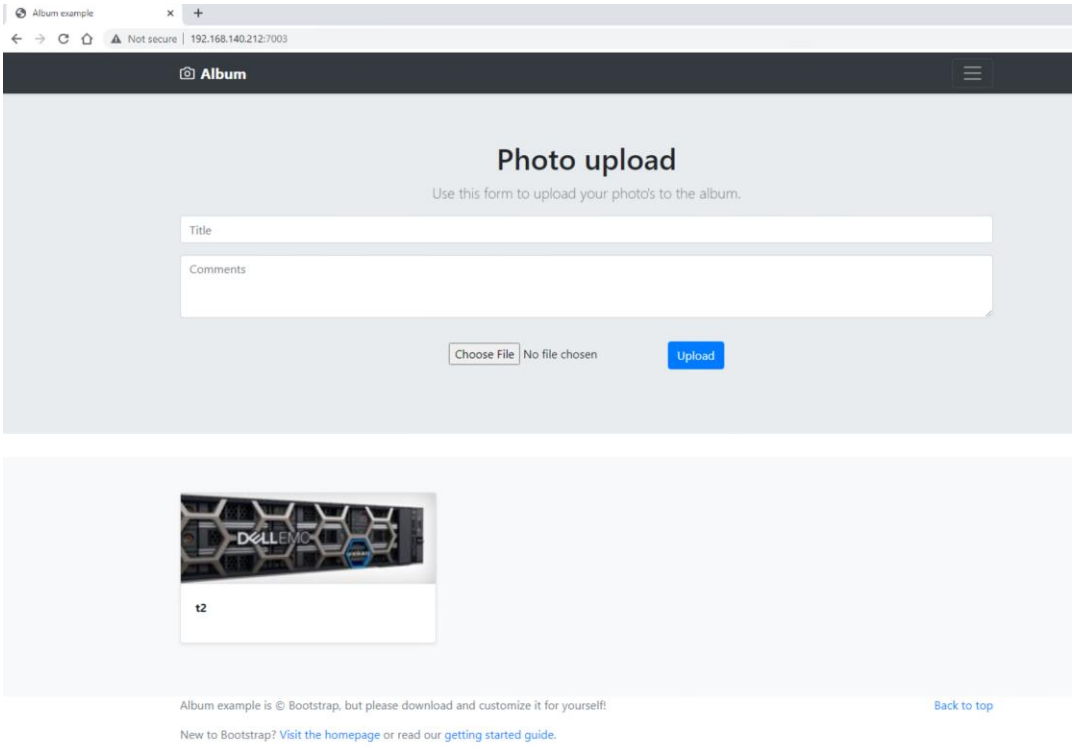
    > kubectl expose deployment photo-album-k8s --type=LoadBalancer --port=7003

```
D:\MyProject>kubectl expose deployment photo-album-k8s --type=LoadBalancer --port=7003
service/photo-album-k8s exposed
```

    > kubectl get svc

```
D:\MyProject>kubectl get svc
NAME              TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
kubernetes        ClusterIP      198.51.100.1    <none>           443/TCP          4d
mongo             LoadBalancer   198.51.100.18   192.168.140.211  27017:31714/TCP  15m
photo-album-k8s   LoadBalancer   198.51.100.48   192.168.140.212  7003:31921/TCP   26s
supervisor        ClusterIP      None            <none>           6443/TCP         4d
```

18. Open a web browser and go to the following URL: http://192.168.140.212:7003
    Note: your external IP may be different



DELL Technologies

Retrospective: The results

- ❑ Deployed a 2-tier app to both Docker

If you successfully ran docker containers for mongodb and photo-album, you should see the below result in the command prompt when you run; docker container ls

```
D:\MyProject>docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                     NAMES
6f43800994cc        photo-album         "python app.py"          10 minutes ago      Up 10 minutes       0.0.0.0:7002->7002/tcp    gifted_edis
on
94f03d643bf9        mongo               "docker-entrypoint.s…"   7 hours ago         Up 7 hours          0.0.0.0:27018->27017/tcp  epic_matsum
oto
```

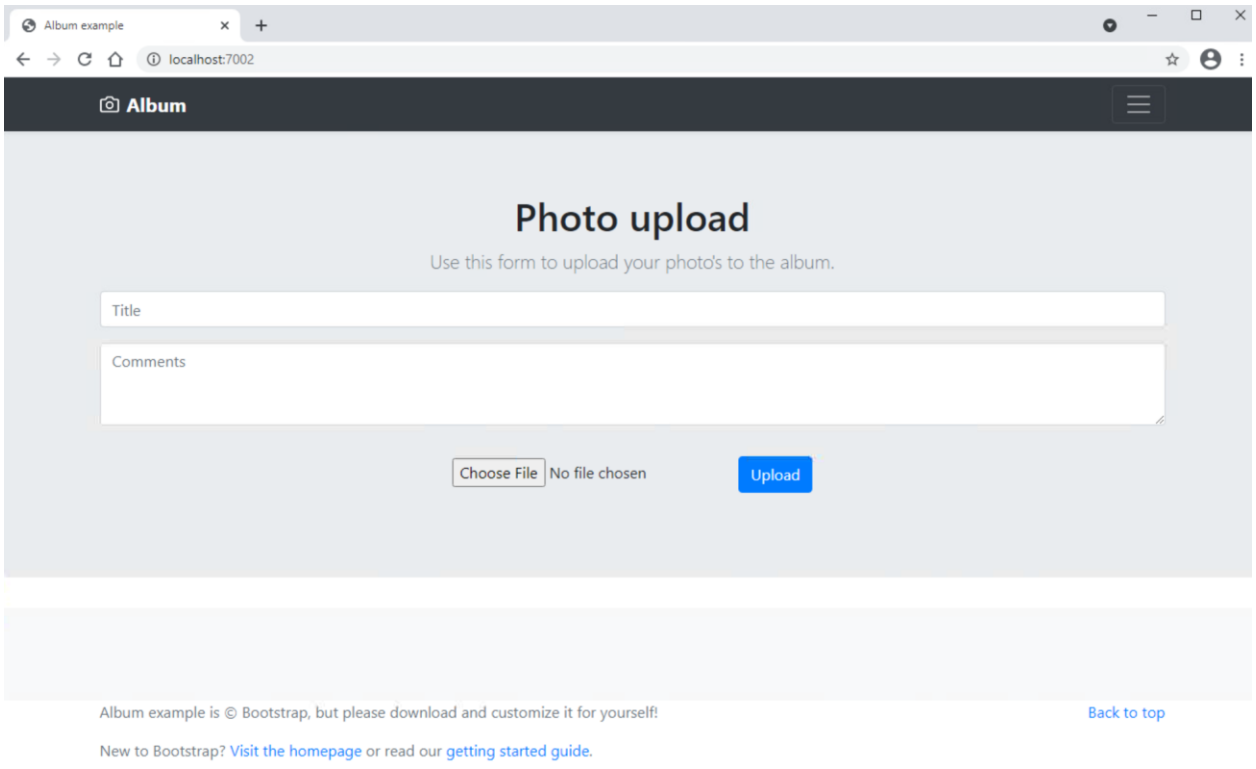
- ❑ Forward port to allow local testing

If you successfully ran docker for photo-album on stated port, you should see the below result in the command prompt when the page is loaded in a web browser and a response is sent back to the user

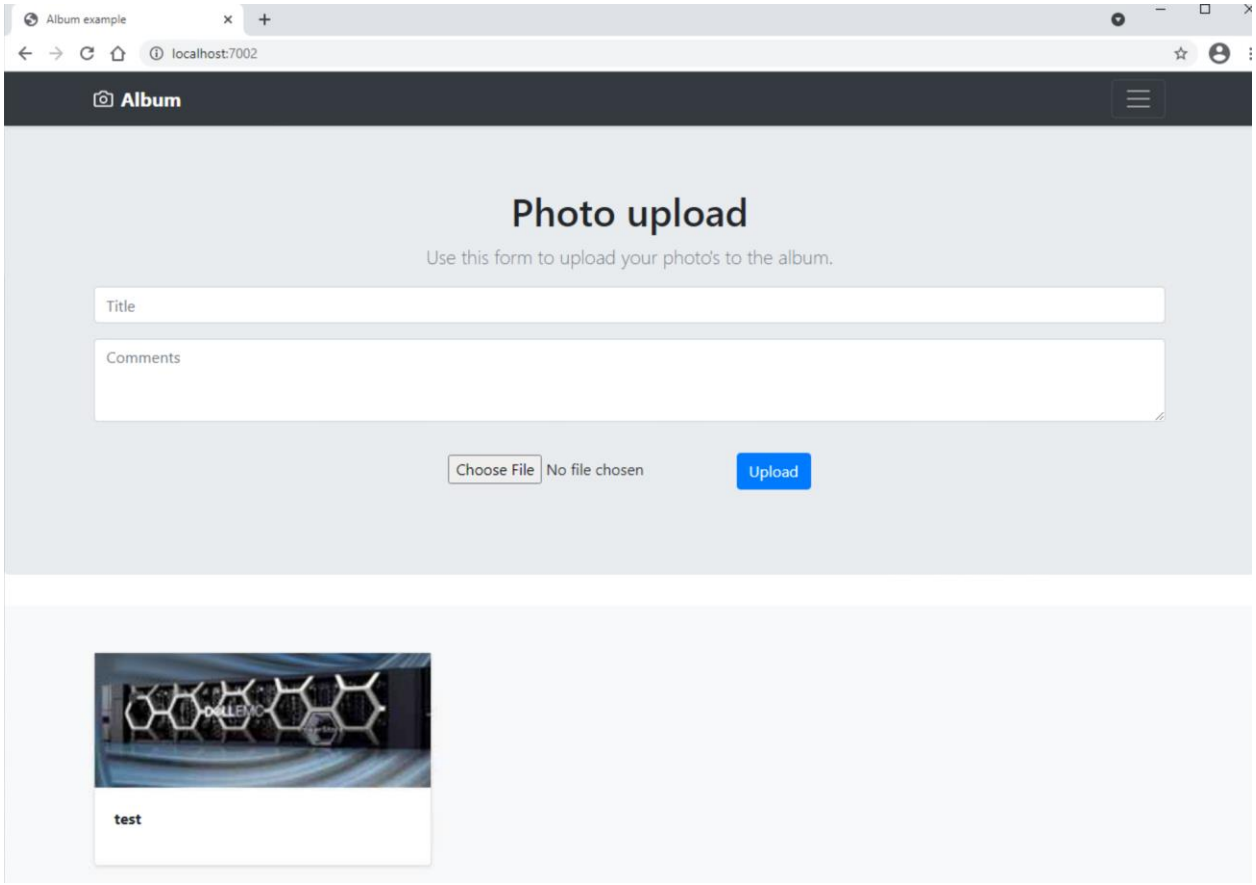
```
D:\MyProject>docker run -p 7002:7002 photo-album
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:7002/ (Press CTRL+C to quit)
172.17.0.1 - - [23/Oct/2020 06:55:41] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [23/Oct/2020 06:55:42] "GET /favicon.ico HTTP/1.1" 404 -
```

- ❑ Tested the web app

If you are running photo-album container on stated port successfully, you should see the below results on the web browser or CLI



If you upload a photo, you should see similar results as below on the web browser or CLI



- ❑ Deployed a 2-tier app to Kubernetes

If you successfully deployed mongodb and photo-album deployments, you should see the below result in the command prompt

```
D:\MyProject>kubectl get deployment
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
mongo             1/1     1            1           23m
photo-album-k8s   1/1     1            1           21s
```

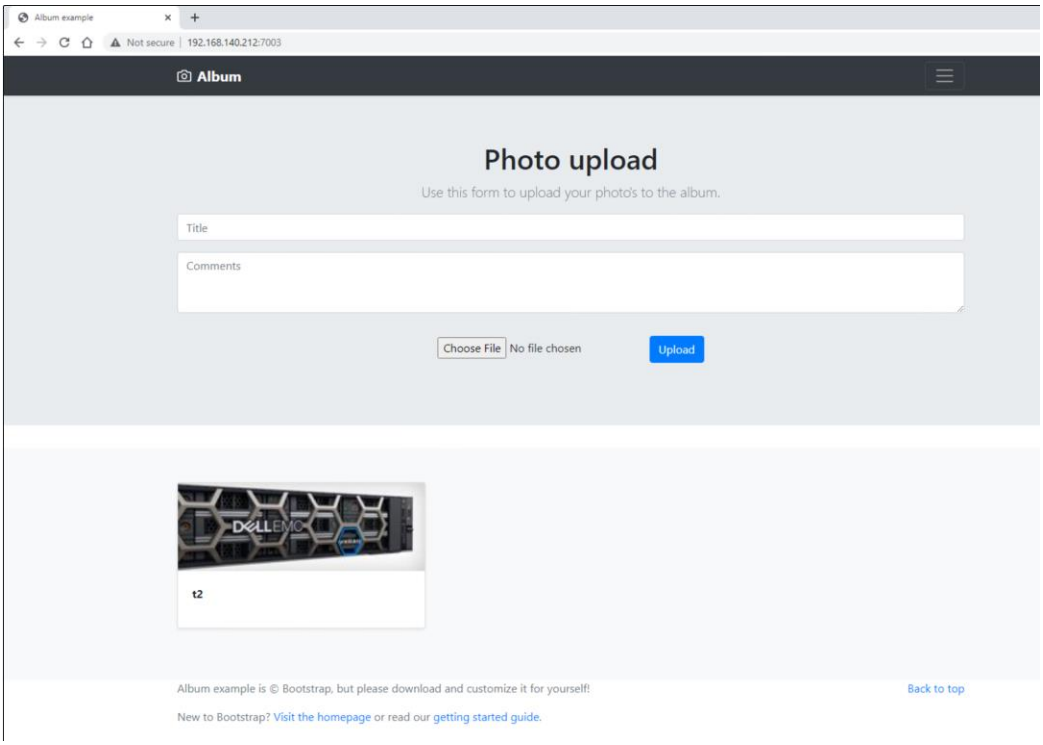
- ❑ Exposed the service port to access application

If you successfully exposed the service port to access application photo-album-k8s on stated port, you should see the similar result in the command prompt

```
D:\MyProject>kubectl get svc
NAME              TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
kubernetes        ClusterIP      198.51.100.1    <none>           443/TCP          4d
mongo             LoadBalancer   198.51.100.18   192.168.140.211  27017:31714/TCP  15m
photo-album-k8s   LoadBalancer   198.51.100.48   192.168.140.212  7003:31921/TCP   26s
supervisor        ClusterIP      None            <none>           6443/TCP         4d
```

- ❑ Tested the web app

If you are running photo-album-k8s deployment on stated port successfully, you should see the below results on the web browser



DELL Technologies

Lab 4: Dynamic Persistent Volumes with Isilon

Module Objectives:

- ❑ Learn about K8s persistent volumes and claims
- ❑ Learn about the CSI plugin for Isilon
- ❑ Use the CSI driver to dynamically provision NFS storage and mount to a pod
- ❑ Test its functionality

DELL Technologies

Lab Exercise: Create a dynamic PV and PVC with an NFS mount to a K8s pod

Complete the below steps to demonstrate your understanding of the tools and concepts required for the remaining lab exercises;

1. Login to the Tanzu Kubernetes Cluster provided for this lab

   > kubectl vsphere login --server=192.168.1.120 --tanzu-kubernetes-cluster-name tkc-cluster-01 --tanzu-kubernetes-cluster-namespace piper --vsphere-username administrator@vsphere.local --insecure-skip-tls-verify

   Note: Enter the password when prompted: Password123!

```
C:\Users\demouser>kubectl vsphere login --server=192.168.1.120 --tanzu-kubernetes-cluster-name tkc-cluster-01 --tanzu-kubernetes-cluster-namespace piper --vsphere-username
 administrator@vsphere.local --insecure-skip-tls-verify

Password:
Logged in successfully.

You have access to the following contexts:
   192.168.1.120
   piper
   tkc-cluster-01

If the context you wish to use is not in this list, you may need to try
logging in again later, or contact your cluster administrator.

To change context, use `kubectl config use-context <workload name>`
```

2. Change context to the tkc-cluster-01 cluster we will be using for the labs

   > kubectl config use-context tkc-cluster-01

```
C:\Users\demouser>kubectl config use-context tkc-cluster-01
Switched to context "tkc-cluster-01".
```

3. Check that the Isilon storage class is setup correctly
   > kubectl describe sc isilon

```
C:\Users\demouser>kubectl describe sc isilon
Name:                  isilon
IsDefaultClass:        Yes
Annotations:           meta.helm.sh/release-name=isilon,meta.helm.sh/release-namespace=isilon,storageclass.beta.kubernetes.io/is-default-class=true
Provisioner:           csi-isilon.dellemc.com
Parameters:            AccessZone=System,AzServiceIP=192.168.1.12,IsiPath=/ifs/data/csi,RootClientEnabled=false
AllowVolumeExpansion:  True
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     Immediate
Events:                <none>
```

   Note: Notice the provisioner is csi-isilon and the parameters that provide the Isilon cluster IP and storage path. These have been pre-configured for the lab environment along with the installation of the CSI driver for Isilon.

4. Check that we have a new namespace for our test pod
   > kubectl get namespace test

```
D:\MyProject>kubectl get namespace test
NAME   STATUS   AGE
test   Active   148d
```

DELL Technologies

5. Set the context to our new namespace
   > kubectl config set-context --current --namespace=test

```
D:\MyProject>kubectl config set-context --current --namespace=test
Context "tkc-cluster-01" modified.
```

6. Clone the following repository (if not already);
   https://github.com/theocrithary/Piper-2021/tree/main/Containers%20%26%20Kubernetes/Lab%2004%20-%20PVs%20with%20Isilon

7. Review the yaml files 'pvc.yaml' and 'test_pod.yaml'

pvc.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvol0
spec:
  accessModes:
  - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
  storageClassName: isilon
```

test_pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: isilontestpod1
spec:
  containers:
  - name: test
    image: docker.io/centos:latest
    command: [ "/bin/sleep", "3600" ]
    volumeMounts:
    - name: workdir
      mountPath: "/data0"
      name: pvol0
  volumes:
    - name: pvol0
      persistentVolumeClaim:
        claimName: pvol0
```

8. Open your command prompt and navigate to the above folder
   > cd "D:\Piper-2021\Containers & Kubernetes\Lab 04 - PVs with Isilon"

9. Create the dynamic PV and PVC with the above yaml file
   > kubectl create -f pvc.yaml

```
D:\Piper-2021\Containers & Kubernetes\Lab 04 - PVs with Isilon>kubectl create -f pvc.yaml
persistentvolumeclaim/pvol0 created
```

10. Create the test pod with a centos image and NFS mount the PVC with the above yaml file
    > kubectl create -f test_pod.yaml

```
D:\Piper-2021\Containers & Kubernetes\Lab 04 - PVs with Isilon>kubectl create -f test_pod.yaml
pod/isilontestpod1 created
```

11. Verify the PV was created successfully

> kubectl get persistentvolumes

```
D:\Piper-2021\Containers & Kubernetes\Lab 04 - PVs with Isilon>kubectl get persistentvolumes
NAME             CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM        STORAGECLASS   REASON   AGE
k8s-b9802c859e   8Gi        RWO            Delete           Bound    test/pvol0   isilon                  116s
```

12. Verify the PVC was created successfully

> kubectl get pvc

```
D:\Piper-2021\Containers & Kubernetes\Lab 04 - PVs with Isilon>kubectl get pvc
NAME     STATUS   VOLUME           CAPACITY   ACCESS MODES   STORAGECLASS   AGE
pvol0    Bound    k8s-b9802c859e   8Gi        RWO            isilon         66s
```

13. Verify the pod was created and is running successfully

> kubectl get pods

```
D:\Piper-2021\Containers & Kubernetes\Lab 04 - PVs with Isilon>kubectl get pods
NAME             READY   STATUS    RESTARTS   AGE
isilontestpod1   1/1     Running   0          53s
```

14. Verify the PVC was mounted successfully to the pod

> kubectl describe pod isilontestpod1

```
D:\Piper-2021\Containers & Kubernetes\Lab 04 - PVs with Isilon>kubectl describe pod isilontestpod1
Name:         isilontestpod1
Namespace:    test
Priority:     0
Node:         tkc-cluster-01-workers-55fb9-74df9dc5b-wqzmf/192.168.140.107
Start Time:   Wed, 14 Apr 2021 04:36:55 +0000
Labels:       <none>
Annotations:  cni.projectcalico.org/podIP: 192.0.2.229/32
              cni.projectcalico.org/podIPs: 192.0.2.229/32
              kubernetes.io/psp: vmware-system-privileged
Status:       Running
IP:           192.0.2.229
IPs:
  IP:  192.0.2.229
Containers:
  test:
    Container ID:  containerd://f36115f821b6082c3aab9a55e319a5a402c457545c2698c6ed4c3ae3b8f9b77a
    Image:         docker.io/centos:latest
    Image ID:      docker.io/library/centos@sha256:5528e8b1b1719d34604c87e11dcd1c0a20bedf46e83b5632cdeac91b8c04efc1
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sleep
      3600
    State:          Running
      Started:      Wed, 14 Apr 2021 04:37:01 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /data0 from pvol0 (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-w9442 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  pvol0:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  pvol0
    ReadOnly:   false
  default-token-w9442:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-w9442
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                 node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason                  Age    From                     Message
  ----    ------                  ----   ----                     -------
  Normal  Scheduled               3m17s  default-scheduler        Successfully assigned test/isilontestpod1 to tkc-cluster-01-workers-55fb9-74df9dc5b-wqzmf
  Normal  SuccessfulAttachVolume  3m17s  attachdetach-controller  AttachVolume.Attach succeeded for volume "k8s-b9802c859e"
  Normal  Pulling                 3m15s  kubelet                  Pulling image "docker.io/centos:latest"
  Normal  Pulled                  3m13s  kubelet                  Successfully pulled image "docker.io/centos:latest"
  Normal  Created                 3m13s  kubelet                  Created container test
  Normal  Started                 3m13s  kubelet                  Started container test
```

15. Connect to a bash terminal of the test pod we just created
    > kubectl exec -it isilontestpod1 -- bash

```
D:\Piper-2021\Containers & Kubernetes\Lab 04 - PVs with Isilon>kubectl exec -it isilontestpod1 -- bash
[root@isilontestpod1 /]#
```

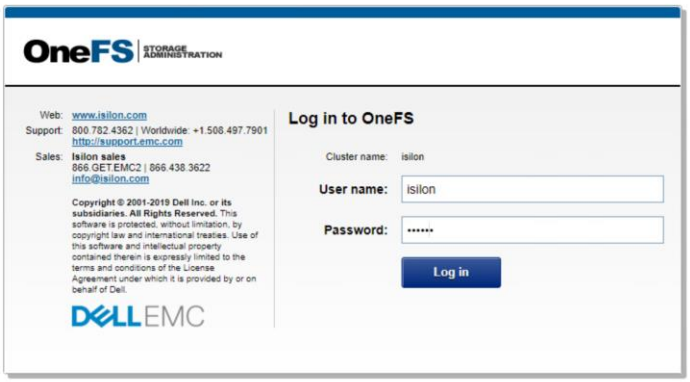
16. Verify the NFS mount and confirm we can write to the file system
    > cd /data0
    > touch test
    > ls -al

```
[root@isilontestpod1 /]# cd /data0
[root@isilontestpod1 data0]# touch test
[root@isilontestpod1 data0]# ls -al
total 7
drwxrwxrwx 2   2001   1544   22 Nov 16 23:51 .
drwxr-xr-x 1 root   root   4096 Nov 16 23:53 ..
-rw-r--r-- 1 nobody nobody    0 Nov 16 23:51 test
```

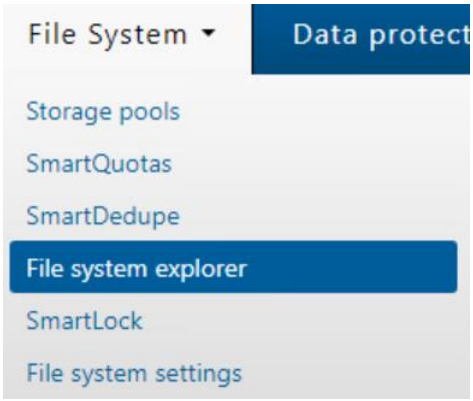
17. Verify the mounted path and NFS options by running the following linux commands
    > df -h | grep data0
    > mount | grep data0

```
[root@isilontestpod1 data0]# df -h | grep data0
192.168.1.12:/ifs/data/csi/k8s-24d056fde3   19G  1.2G   18G   7% /data0
[root@isilontestpod1 data0]# mount | grep data0
192.168.1.12:/ifs/data/csi/k8s-24d056fde3 on /data0 type nfs4 (rw,relatime,vers=4.0,rsize=1048576,wsize=1048576,namlen=255,hard,proto=tcp,timeo=600,retrans=2,sec=sys,client
addr=192.168.140.107,local_lock=none,addr=192.168.1.12)
[root@isilontestpod1 data0]#
```

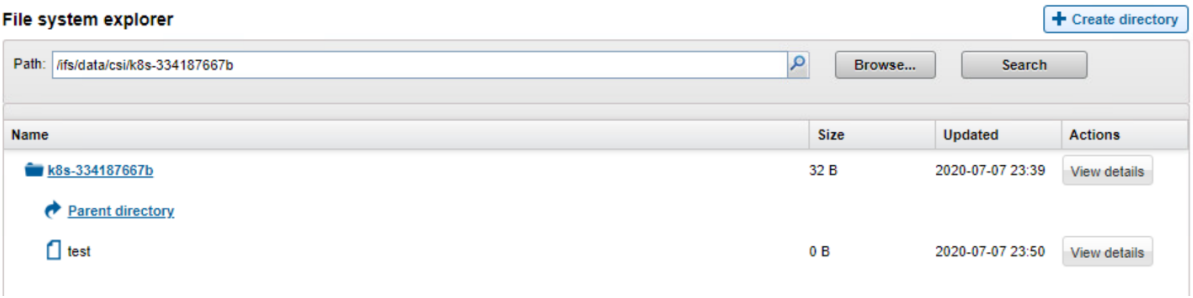
18. Open a chrome browser and login to the Isilon cluster using the bookmark link and the credentials: isilon / isilon



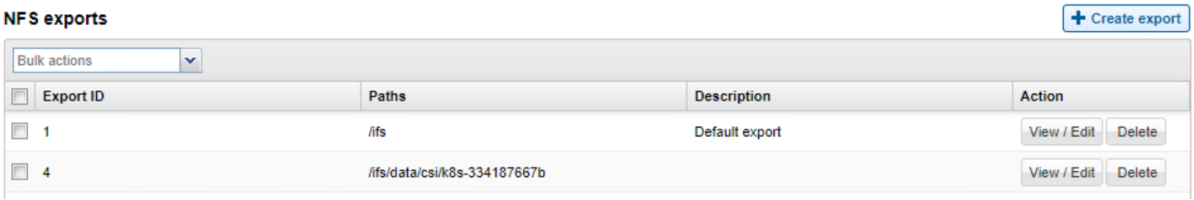
19. Navigate to the “File System” -> “File system explorer” menu



20. Click through the folder structure to get to the ‘/ifs/data/csi’ path and check that the NFS share was created with a k8s dynamically provisioned share name and that the test file was created as tested in step 14.



21. Navigate to the “Protocols” -> “Unix sharing (NFS)” menu and confirm the K8s dynamically provisioned NFS export



DELL Technologies

Retrospective: The results

- ❑ Created a K8s dynamic persistent volume and persistent volume claim using the Isilon CSI driver and provisioner to create NFS shares on an Isilon cluster
- ❑ Created a test pod based on a CentOS image and mounted the PVC to the pod at a new mount point /data0
- ❑ Verified the NFS mount and performed a test write to the path using the touch command
- ❑ Verified the test file was created on the Isilon cluster and confirmed the NFS dynamic share export

DELL Technologies

Lab 5: Data Protection for Kubernetes

Module Objectives:

- ❑ Learn about PowerProtect Data Manager
- ❑ Learn about Kubernetes crash consistent backup and restore functions
- ❑ Learn about PVC's and their application for Data Protection
- ❑ Use PPDM to perform a crash consistent backup of a namespace
- ❑ Restore the namespace to an alternate cluster and validate

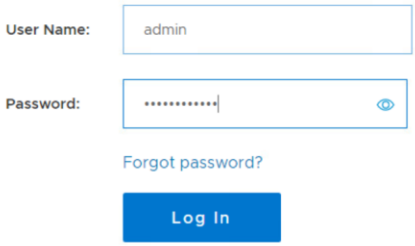
DELL Technologies

Lab Exercise: Perform a Backup of a K8s Namespace and Restore to an Alternate Cluster

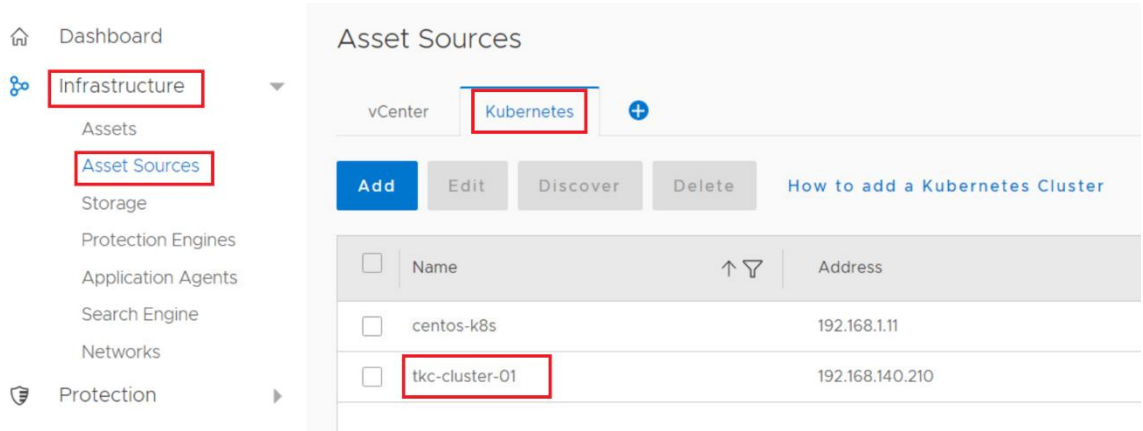
1. Open a chrome browser session
2. Login to the Power Protect Data Manager interface using the bookmark provided in the lab

   https://ppdm.demo.local

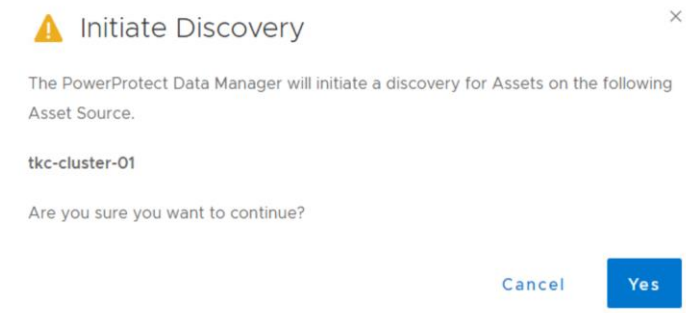
   Note: Enter the username and password when prompted: admin / Password123!



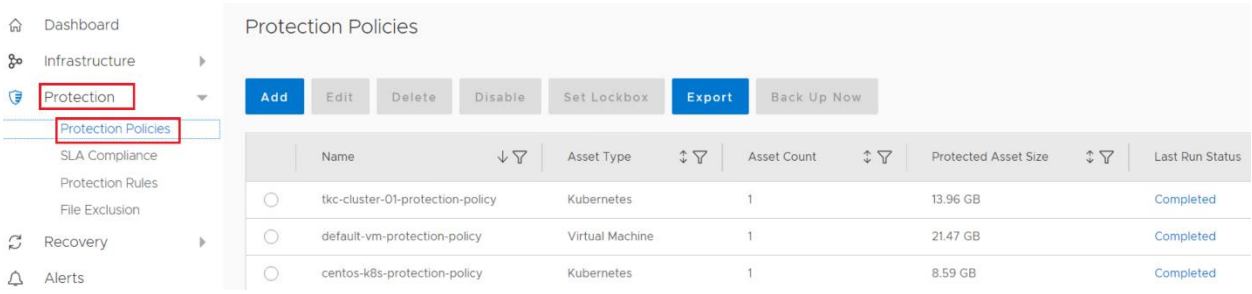
3. Navigate to Infrastructure --> Asset Sources --> Kubernetes using the side menu



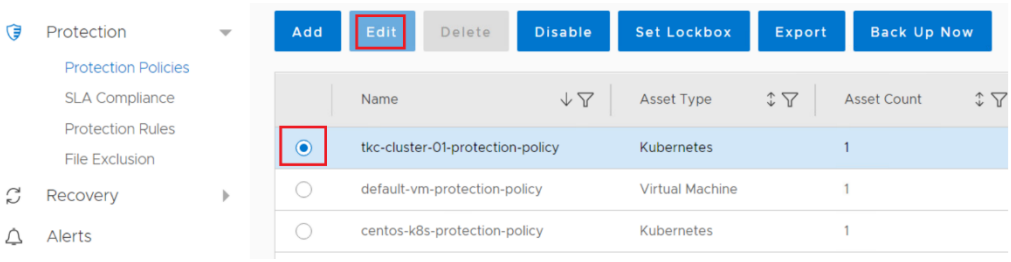
4. Select the 'tkc-cluster-01' item and click on the "Discover" button. Click "Yes", when you are prompted with the below message.



5. Navigate to Protection --> Protection Policies using the side menu

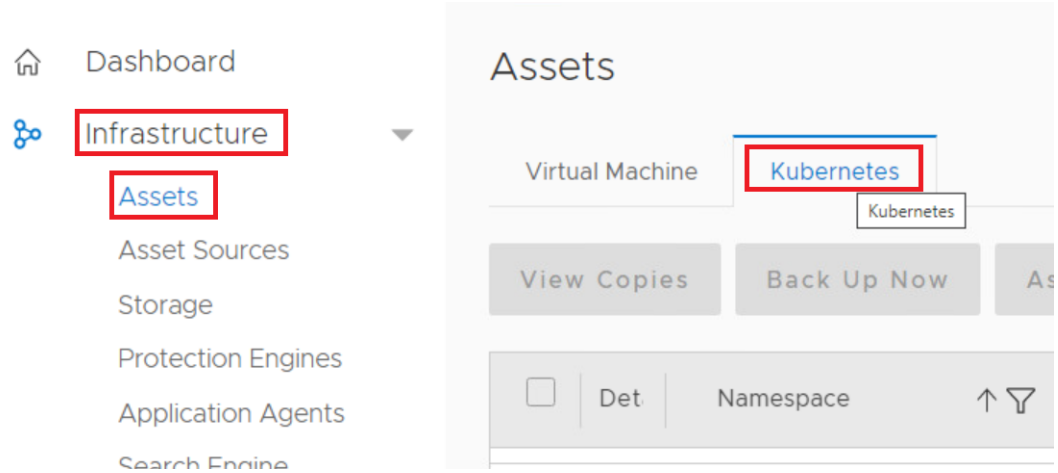


6. Select the item named 'tkc-cluster-01-protection-policy' and click the "Edit" button

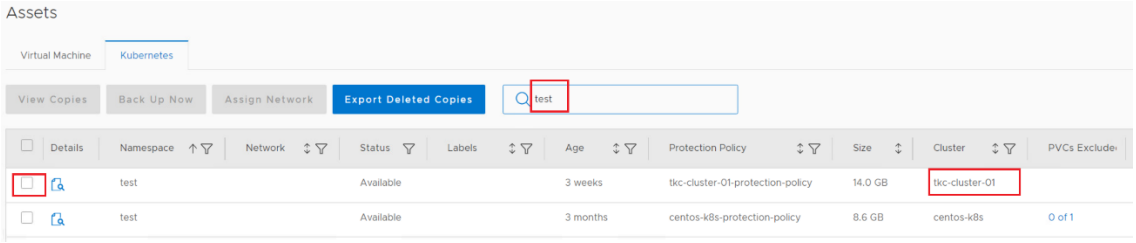


7. Review the policy by using the back and next buttons to navigate between the pages of settings. Click cancel or finish when you have reviewed the policy. Do not save the policy if you have made changes.

8. Navigate to Infrastructure --> Assets --> Kubernetes using the side menu

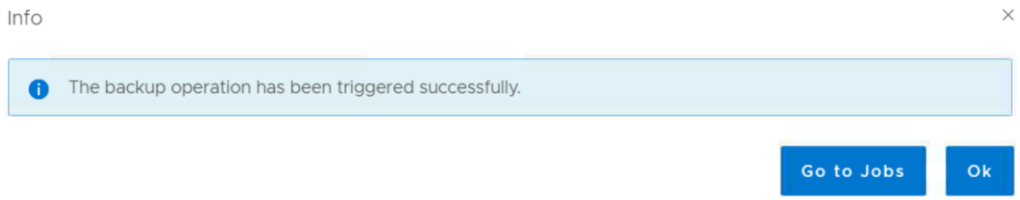


9. Type in 'test' into the search field to filter for namespaces that are named test

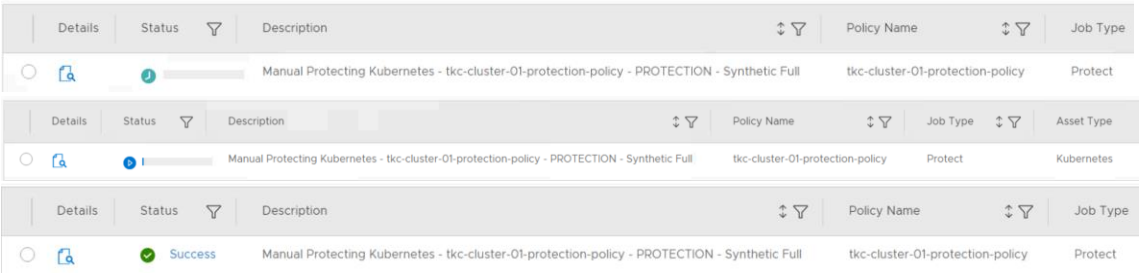


Note: We have 2 clusters currently configured that both have a 'test' namespace. We will use the TKC cluster for this example lab.

10. Select the item with the cluster name 'tkc-cluster-01' and click the "Back Up Now" button. You will then be presented the below screen.

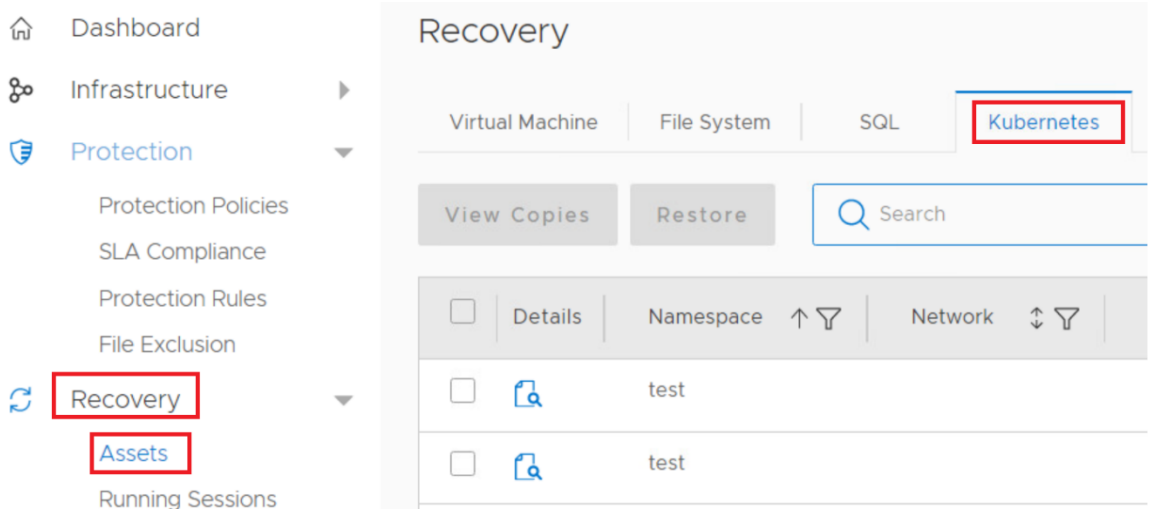


11. Click on "Go to Jobs" and you should see the progress status of the backup job change status



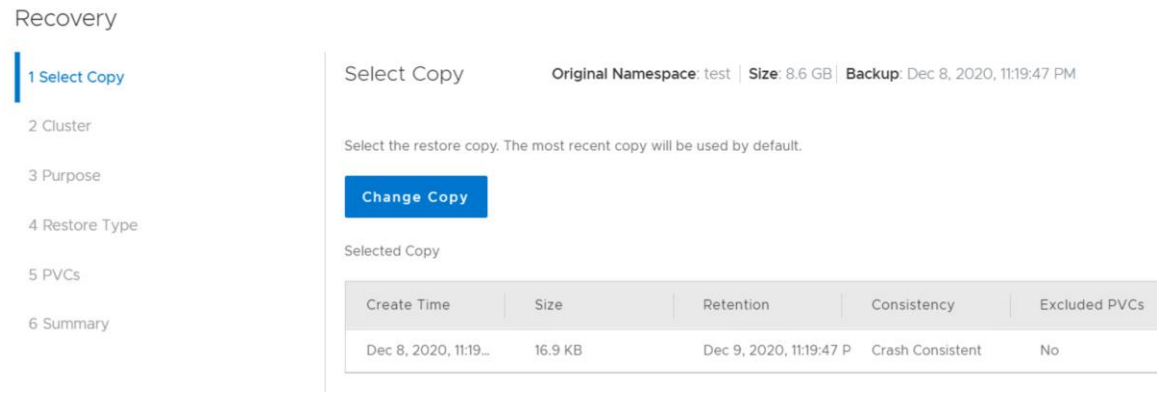
Note: The backup job will take around 3mins to complete

11. Navigate to Recovery --> Assets --> Kubernetes using the side menu

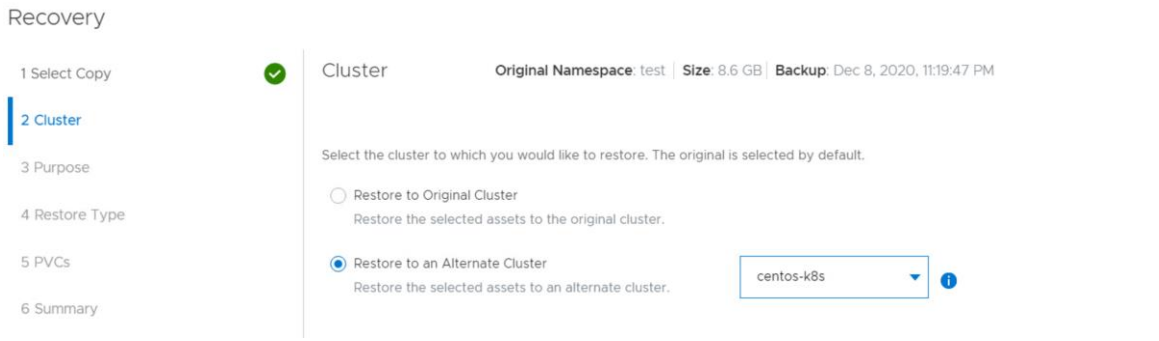


12. Select the 'test' item with the cluster name "tkc-cluster-01" and click on "Restore"

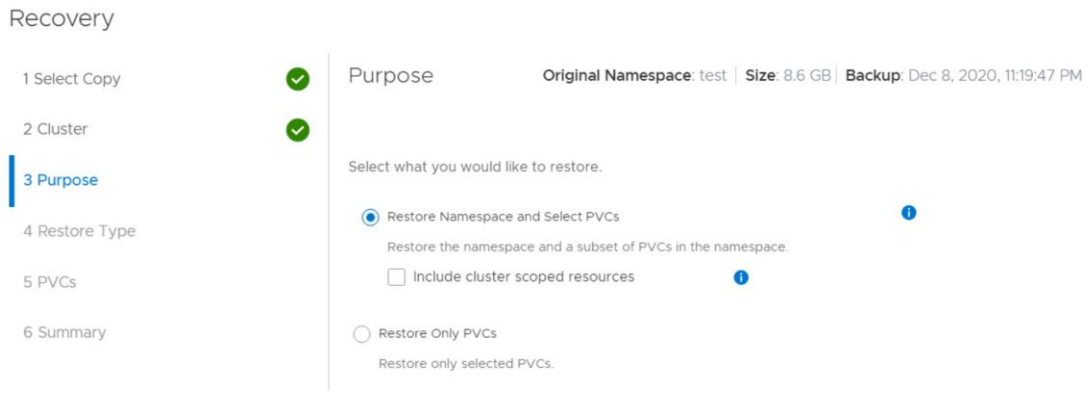
13. Click "Next"



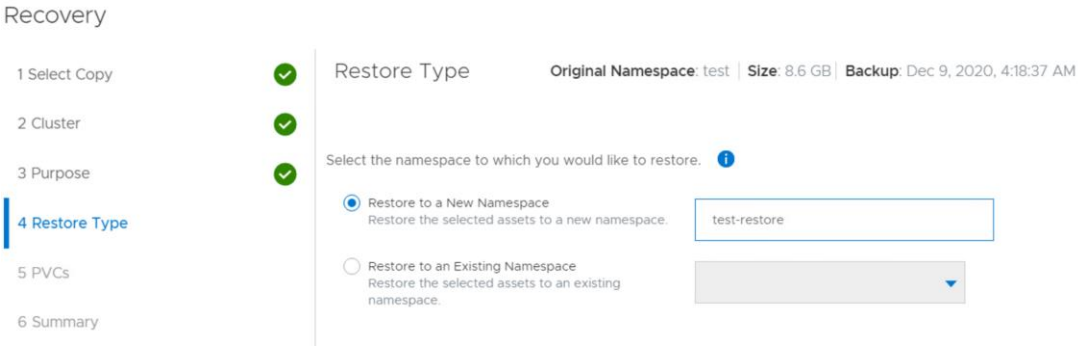
14. Select 'Restore to an Alternate Cluster' option and select the "centos-k8s" cluster from the dropdown menu, then click "Next"



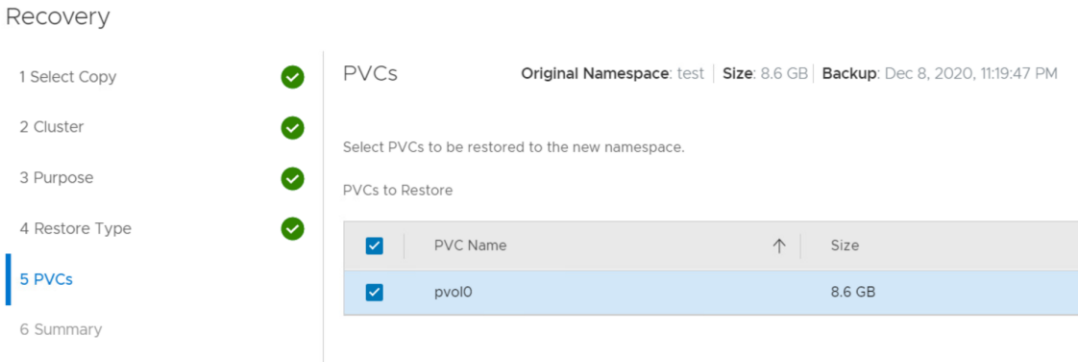
15. Select ‘Restore Namespaces and Select PVCs’ and click “Next”



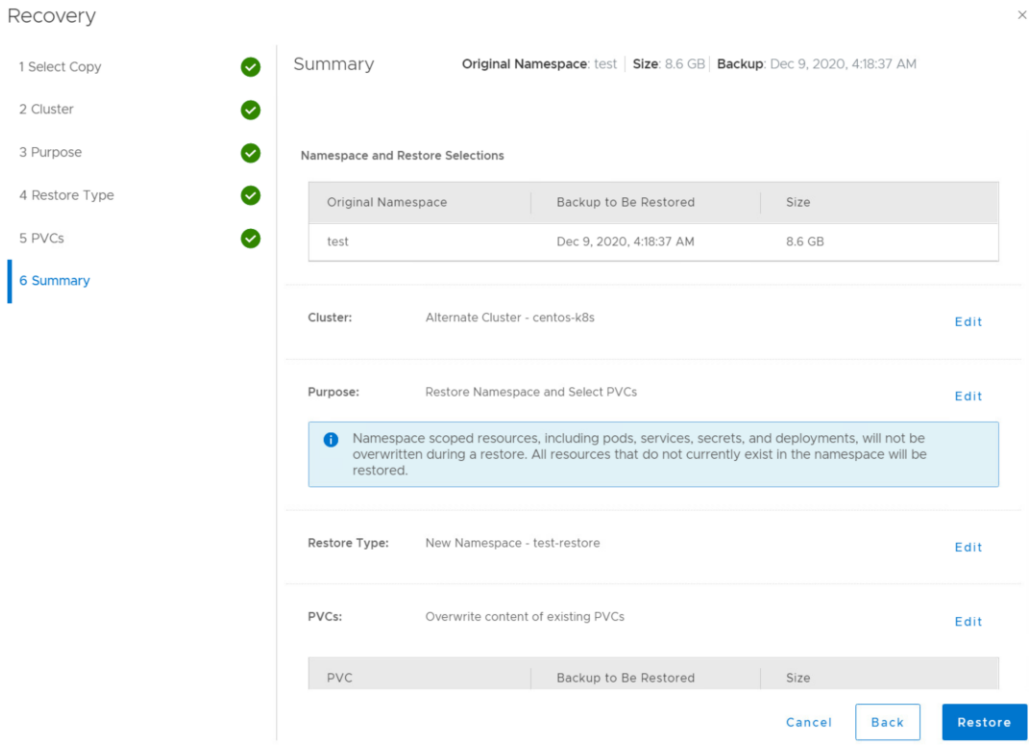
16. Select ‘Restore to a New Namespace’ and enter “test-restore” in the text field before clicking “Next”



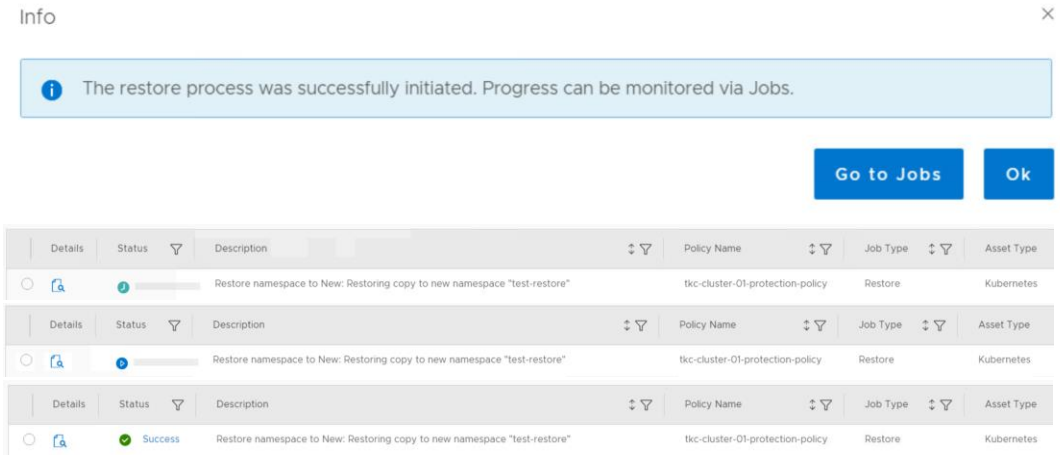
17. Ensure that the PVC “pvol0” is selected and click “Next”



18. Review the Summary page and click “Restore”

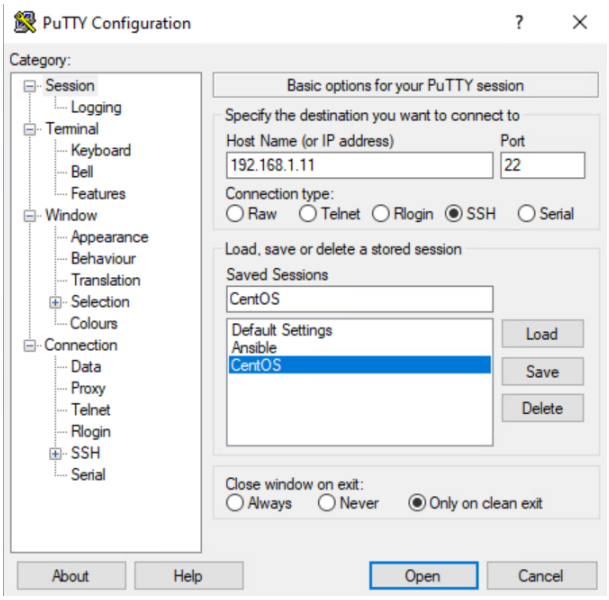


19. Click on “Go to Jobs” and monitor the status as it completes the restore process

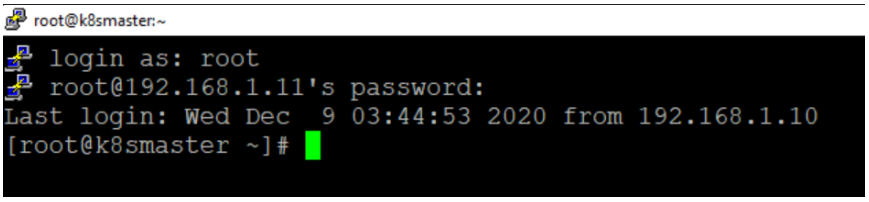


Note: The restore job will take around 4mins to complete

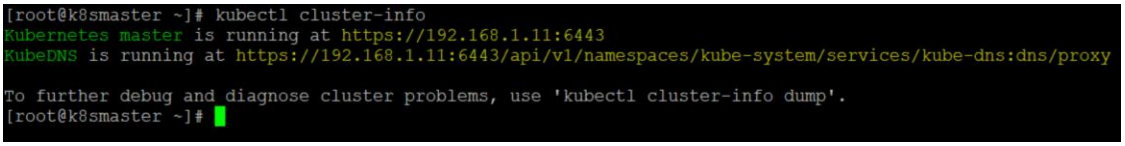
20. Open a putty session using the desktop shortcut and connect to the CentOS saved session



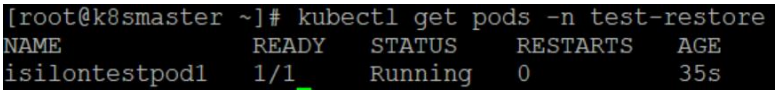
21. Login with the username ‘root’ and password ‘Password123!’



22. Confirm you are logged in and connected to the K8s cluster by running the following command;
    > kubectl cluster-info



23. Run the following to check that the pod was restored in the namespace ‘test-restore’
    > kubectl get pods -n test-restore



24. Run the following command to confirm the PVC was attached to the pod successfully
    > kubectl describe pod isilontestpod1 -n test-restore

DELL Technologies

```
[root@k8smaster ~]# kubectl get pods -n test-restore
NAME            READY   STATUS    RESTARTS   AGE
isilontestpod1  1/1     Running   0          35s
[root@k8smaster ~]# kubectl describe pod isilontestpod1 -n test-restore
Name:         isilontestpod1
Namespace:    test-restore
Priority:     0
Node:         k8smaster/192.168.1.11
Start Time:   Wed, 14 Apr 2021 06:34:20 +0100
Labels:       velero.io/backup-name=test-2021-04-14-04-59-16-test
              velero.io/restore-name=064acc6b-c39f-5bd5-ae2d-6703f016e406-2021-04-14-05-32-43-923fd6
Annotations:  cni.projectcalico.org/podIP: 172.16.16.160/32
              cni.projectcalico.org/podIPs: 172.16.16.160/32
              kubernetes.io/psp: vmware-system-privileged
Status:       Running
IP:           172.16.16.160
IPs:
  IP:  172.16.16.160
Containers:
  test:
    Container ID:  docker://182960edc8093224507c82fc239b41679f053c2d2210e87f3c2faea903ce9245
    Image:         docker.io/centos:latest
    Image ID:      docker-pullable://docker.io/centos@sha256:5528e8b1b1719d34604c87e11dcd1c0a20bedf46e83b5632cdeac91b8c04efc1
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sleep
      3600
    State:          Running
      Started:      Wed, 14 Apr 2021 06:34:40 +0100
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /data0 from pvol0 (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-bth4p (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  pvol0:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  pvol0
    ReadOnly:   false
  default-token-bth4p:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-bth4p
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                 node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason                  Age   From                     Message
  ----    ------                  ----  ----                     -------
  Normal  Scheduled               109s  default-scheduler        Successfully assigned test-restore/isilontestpod1 to k8smaster
  Normal  SuccessfulAttachVolume  108s  attachdetach-controller  AttachVolume.Attach succeeded for volume "k8s-d3af83a500"
  Normal  Pulling                 91s   kubelet                  Pulling image "docker.io/centos:latest"
  Normal  Pulled                  89s   kubelet                  Successfully pulled image "docker.io/centos:latest"
  Normal  Created                 89s   kubelet                  Created container test
  Normal  Started                 88s   kubelet                  Started container test
```

25. Execute a bash tunnel into the pod to verify the contents of the persistent volume
    > kubectl exec -it isilontestpod1 -n test-restore -- bash

```
[root@k8smaster ~]# kubectl exec -it isilontestpod1 -n test-restore -- bash
[root@isilontestpod1 /]#
```

26. You can now use the bash terminal to navigate to the data0 mount point and verify the file created in the previous “Lab 4 – Dynamic Persistent Volumes” was restored
    # cd /data0
    # df -h .
    # ls -al

```
[root@isilontestpod1 /]# cd /data0
[root@isilontestpod1 data0]# df -h .
Filesystem                                Size  Used Avail Use% Mounted on
192.168.1.12:/ifs/data/csi/k8s-d3af83a500   19G  3.0G   16G  16% /data0
[root@isilontestpod1 data0]# ls -al
total 3
drwxrwxrwx. 2    2001    1544 22 Apr 14 05:23 .
drwxr-xr-x. 1 root    root    30 Apr 14 05:34 ..
-rw-r--r--. 1 nobody nobody  0 Apr 14 05:23 test
```

DELL Technologies

Retrospective: The results

- ❑ Reviewed the backup protection policy settings and configuration
- ❑ Created a backup of a namespace consisting of a pod with an attached persistent volume provided by an Isilon storage provider
- ❑ Confirmed the backup was completed successfully
- ❑ Restored the backup to an alternate k8s cluster
- ❑ Verified the namespace, pod and pvc were created on the destination cluster
- ❑ Verified the test file was created on the restored pod with mounted persistent volume