



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



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

The information in this publication is provided "as is." Dell Inc. makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose.

Use, copying, and distribution of any software described in this publication requires an applicable software license.

© 2021 Dell Inc. or its subsidiaries. All Rights Reserved. Dell, EMC, Dell Technologies and other trademarks are trademarks of Dell Inc. or its subsidiaries. Other trademarks may be trademarks of their respective owners.

Dell believes the information in this document is accurate as of its publication date. The information is subject to change without notice.

Student Lab Guide



Table of Contents

Lab 1: Docker	4
Module Objectives:.....	4
Lab Exercise: Build a docker image and run as a docker container.....	5
Retrospective: The results.....	6
Lab 2: Kubernetes	7
Module Objectives:.....	7
Lab Exercise: Create a Kubernetes deployment from a Docker image and expose a service.....	8
Retrospective: The results.....	11
Lab 3: Containerising an App	12
Module Objectives:.....	12
Lab Exercise Part 1: Package a 2-tier app and run locally using Docker Desktop.....	13
Lab Exercise Part-2: Deploy a Docker based 2-tier app to a Kubernetes cluster.....	16
Retrospective: The results.....	20
Lab 4: Dynamic Persistent Volumes with Isilon	23
Module Objectives:.....	23
Lab Exercise: Create a dynamic PV and PVC with an NFS mount to a K8s pod.....	24
Retrospective: The results.....	29
Lab 5: Data Protection for Kubernetes	30
Module Objectives:.....	30
Lab Exercise: Perform a Backup of a K8s Namespace and Restore to an Alternate Cluster	31
Retrospective: The results.....	39



Lab 1: Docker

Module Objectives:

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



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>>



Retrospective: The results

- ❑ Built a Docker image

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

```
D:\Piper-2021\Containers & Kubernetes\Lab 01 - Docker>docker build -t helloworld .
[+] Building 207.6s (9/9) FINISHED
--> [internal] load build definition from Dockerfile
--> => transferring dockerfile: 516B
--> [internal] load .dockerignore
--> => transferring context: 2B
--> [internal] load metadata for docker.io/library/python:latest
--> [1/4] FROM docker.io/library/python:latest@sha256:438cb846732e397ec5d94b1aea461fe26a51b901d510ca105eba5595621db3fe
--> => resolve docker.io/library/python:latest@sha256:438cb846732e397ec5d94b1aea461fe26a51b901d510ca105eba5595621db3fe
--> => sha256:3addbee1d2ce67402634a0133b126fd7f9b0b8fc1a0901fe 8.34kB / 8.34kB
--> => sha256:a3a3dbadd2d56d7418508211551c461f826d1cae349d39a355a1c22fe140b36 51.20MB / 51.20MB
--> => sha256:438cb846732e397ec5d94b1aea461fe26a51b901d510ca105eba5595621db3fe 2.36kB / 2.36kB
--> => sha256:c3aaeedffaf4f7327a2e251625c8eadf8b94079862503d02fc186a4b79c0B 2.22kB / 2.22kB
--> => sha256:d6970e2d41835ffea1a656cc3d1f8e17bc7b8539b2374540508fcfc25d10fd312 8.00MB / 8.00MB
--> => sha256:8c7a5e28b3a8e658d4f08c45428493e8e1cc74a8b83b3c4799 53.44MB / 53.44MB
--> => sha256:8c7a5e28b3a8e658d4f08c45428493e8e1cc74a8b83b3c4799 53.44MB / 53.44MB
--> => sha256:98e4b62c2fad846ba1bc17add18106ebdfbe771089574c1a033f9698ba0036 198.90MB / 198.90MB
--> => sha256:335761f9d9e5f0e2669cc89ee5b623626f822424bdc181ac5eab541a7a7b158ef0c01 6.49MB / 6.49MB
--> => => extracting sha256:a3a3dbadd2d56d7418508211551c461f826d1cae349d39a355a1c22fe140b36
--> => sha256:c21e6e52f5e0328a022cc7c9f0eafb5e0e69d3cd019da121ca14d0ad9f0 18.46MB / 18.46MB
--> => => extracting sha256:d6970e2d41835ffea1a656cc3d1f8e17bc7b8539b2374540508fcfc25d10fd312
--> => => extracting sha256:c27a2cf0e2e689cc89ee5b623626f822424bdc181ac5eab541a7a7b158ef0c01
--> => sha256:81b6e7b6d19cc5efd32b6f8778385d386824503ef1e9c1cdclc7b7cd1081982a3 233B / 233B
--> => => extracting sha256:8c7a5e28b3a8e658d4f08c45428493e8e1cc74a8b83b3c4799
--> => sha256:6fea567701f9dd6050ef10e369:9257094ea9:298ae67bb990c50d201c3e084e 2.16MB / 2.16MB
--> => => extracting sha256:98e4b62cf2ad846ba0be17add3810ebdfbe771089574c1a033f9698ba00e36
--> => => extracting sha256:355761f9d9e5f667e2b980cd1e12e202f5cd29176dd5e0dca45061fd7a499
--> => => extracting sha256:c21e6e52f5e0328a022cc7c9f0eafb5e0e69d3cd019da121ca14d0ad9f0
--> => => extracting sha256:81b6e7b6d19cc5efd32b6f8778385d386824503ef1e9c1cdclc7b7cd1081982a3
--> => => extracting sha256:6fea567701f9dd6050ef10e369:9257094ea9:298ae67bb990c50d201c3e084e
--> [internal] load build context
--> => transferring context: 1.36kB
--> [2/4] WORKDIR /usr/src/app
--> [3/4] COPY . ./
--> [4/4] RUN pip install --no-cache-dir -r requirements.txt
--> => exporting to image
--> => exporting layers
--> => writing image sha256:d34276607be85970d61a39a8df7b0bf230b92d4af9deff16a5eccc8a1dab52f4
--> => naming to docker.io/library/helloworld
```

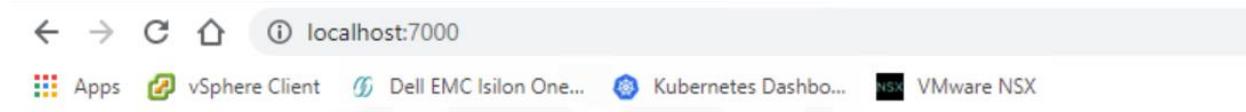
- ❑ Ran a container instance from a Docker image

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

```
D:\Piper-2021\Containers & Kubernetes\Lab 01 - Docker>docker run -p 7000:7000 helloworld
* 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://0.0.0.0:7000/ (Press CTRL+C to quit)
```

- ❑ Tested the web app

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



This is a basic web page written in Python and served by Flask



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



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

- Clone the following repository (if not already);

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

- 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

- 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
```

- 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
```

- 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
--> [internal] transfer dockerfile: 516B
--> [internal] load .dockerignore
--> [internal] load context: 2B
--> [internal] load metadata for docker.io/library/python:latest
--> [internal] load build context
--> [internal] transfer context: 1.34kB
--> [1/4] FROM docker.io/library/python:latest@sha256:438cb846732e397ec5d94b1aea461fe26a51b901d510ca105eba5595621db3fe
--> CACHED [2/4] WORKDIR /usr/src/app
--> [3/4] COPY . .
--> [4/4] RUN pip install --no-cache-dir -r requirements.txt
--> exporting to image
--> exporting layers
--> writing image sha256:3e104a8bf26a242733c89c81d344fed81bae6bd34fdd98a7273cf0524e3c9d3
--> naming to docker.io/theocrithary/helloworld
```

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>`
```



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

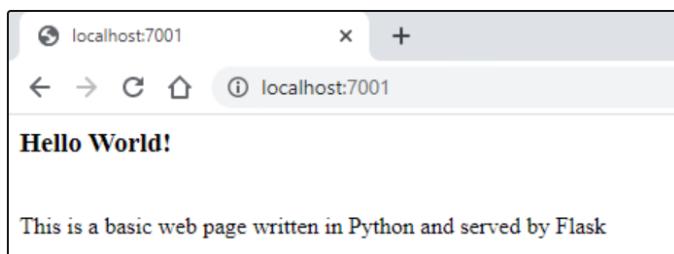
NAME	READY	STATUS	RESTARTS	AGE
helloworld	1/1	Running	0	53s
snapshot-controller-0	1/1	Running	2	149d

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



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



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



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;

- 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
49ccb10cca85: Pull complete
9729d7ec22de: Pull complete
7b7fd72268d8: Pull complete
5e2934dacaf5: Pull complete
bf9da24d4b2c: Pull complete
d2f8c3715616: Pull complete
e9f964aa45b0: Pull complete
bd66718f31e2: Pull complete
41ed4d1a1542: Pull complete
7336dfc228e2: Pull complete
Digest: sha256:67018ee2847d8c35e8c7aebea629795d091f93c93e23d3d60741fde74ed6858c4
Status: Downloaded newer image for mongo:latest
docker.io/library/mongo:latest
```

- Type this command to start the mongo DB container

```
> docker run -p 27018:27017 mongo
```

```
D:\Piper-2021\Containers & Kubernetes\Lab 03 - Part 1 - Containerising an App>docker run -p 27018:27017 mongo
{"t":{"$date":"2021-04-08T23:26:05.457+00:00"},"s":"I", "c":"CONTROL", "id":23285, "ctx":"main","msg":"Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2021-04-08T23:26:05.485+00:00"},"s":"W", "c":"ASIO", "id":22601, "ctx":"main","msg":"No TransportLayer configured during NetworkInterface startup"}
{"t":{"$date":"2021-04-08T23:26:05.495+00:00"},"s":"I", "c":"NETWORK", "id":4648601, "ctx":"main","msg":"Implicit TCP FastOpen unavailable. If TCP FastOpen is required, set tcpFastOpenServer, tcpFastOpenClient, and tcpFastOpenQueueSize."}
{"t":{"$date":"2021-04-08T23:26:05.499+00:00"},"s":"I", "c":"STORAGE", "id":4615611, "ctx":"initandlisten","msg":"MongoDB starting","attr":{"pid":1,"port":27017,"dbPath":"/data/db","architecture":"64-bit","host":"3de840abd713"}}
{"t":{"$date":"2021-04-08T23:26:05.499+00:00"},"s":"I", "c":"CONTROL", "id":23403, "ctx":"initandlisten","msg":"Build Info","attr":{"buildInfo":{"version":"4.4.5","gitVersion":"ff5cb77101b052fa02da43b8538093486cf9b3f7","openSSLVersion":"OpenSSL 1.1.1 1 Sep 2018","modules":[],"allocator":"tcmalloc","environment":{"distmod":"ubuntu1804","distarch":"x86_64","target_arch":"x86_64"}}}}
 {"t":{"$date":"2021-04-08T23:26:05.500+00:00"},"s":"I", "c":"CONTROL", "id":51765, "ctx":"initandlisten","msg":"Operating System","attr":{"os":{"name":"Ubuntu","version":"18.04"}}}
 {"t":{"$date":"2021-04-08T23:26:05.500+00:00"},"s":"I", "c":"CONTROL", "id":21951, "ctx":"initandlisten","msg":"Options set by command line","attr":{"options":{"net":{"bindIp":["*"]}}}}
 {"t":{"$date":"2021-04-08T23:26:05.507+00:00"},"s":"I", "c":"STORAGE", "id":22297, "ctx":"initandlisten","msg":"Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem","tags":["startupWarnings"]}
 {"t":{"$date":"2021-04-08T23:26:05.514+00:00"},"s":"I", "c":"STORAGE", "id":22315, "ctx":"initandlisten","msg":"Opening WiredTiger","attr":{"config":{"create,cache_size=2660M,session_max=3000,eviction=(threads_min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000,close_scan_interval=10,close_handle_minimum=250),statistics_log=(wait=0),verbose=[recovery_progress,checkpoint_progress,compact_progress]},"tags":["wiredTigerLogOpen"]}}
 {"t":{"$date":"2021-04-08T23:26:06.270+00:00"},"s":"I", "c":"STORAGE", "id":22430, "ctx":"initandlisten","msg":"WiredTiger message","attr":{"message":"[1617924366:270933][1:0x7f522d360a0c], txn-recover: [WT_VERB_RECOVERY | WT_VERB_RECOVERY_PROGRESS] Set global recovery timestamp: (0, 0)"}}
 {"t":{"$date":"2021-04-08T23:26:06.271+00:00"},"s":"I", "c":"STORAGE", "id":22438, "ctx":"initandlisten","msg":"WiredTiger message","attr":{"message":"[1617924366:271196][1:0x7f522d360a0c], txn-recover: [WT_VERB_RECOVERY | WT_VERB_RECOVERY_PROGRESS] Set global oldest timestamp: (0, 0)"}}
 {"t":{"$date":"2021-04-08T23:26:06.339+00:00"},"s":"I", "c":"STORAGE", "id":4795906, "ctx":"initandlisten","msg":"WiredTiger opened","attr":{"durationMillis":821}}
 {"t":{"$date":"2021-04-08T23:26:06.339+00:00"},"s":"I", "c":"RECOVERY", "id":23987, "ctx":"initandlisten","msg":"WiredTiger recoveryTimestamp","attr":{"recoveryTimestamp":{"$timestamp":{"t":0,"i":0}}}}
 {"t":{"$date":"2021-04-08T23:26:06.391+00:00"},"s":"I", "c":"STORAGE", "id":4366408, "ctx":"initandlisten","msg":"No table logging settings modifications are required for existing WiredTiger tables","attr":{"loggingEnabled":true}}
```

> 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
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
3de840abd713	mongo	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0:27018->27017/tcp	descending_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 descending_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",  
"SecondaryIPAddresses": null,
```

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
=> => transferring context: 2B
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 32B
=> [internal] load metadata for docker.io/library/python:latest
=> [1/4] FROM docker.io/library/python:latest@sha256:8a9325e9adb51dd65ec915036e270eae330807918fc1254f6a22e436741600b17
=> [internal] load build context
=> => transferring context: 4.60kB
=> CACHED [2/4] WORKDIR /usr/src/app
=> [3/4] COPY . .
=> [4/4] RUN pip install --no-cache-dir -r requirements.txt
=> exporting to image
=> => exporting layers
=> => writing image sha256:176af6ee0886e9270eeb51e5cef0445fd807a225f64ab706b7063227e2336cfc
=> => naming to docker.io/library/photo-album
```

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

> docker image ls

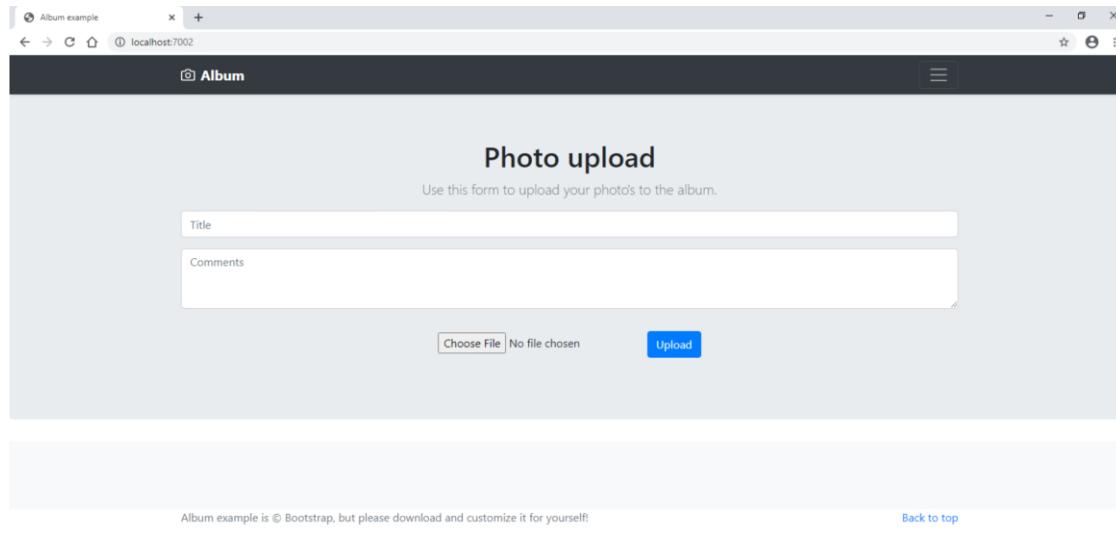
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
photo-album	latest	1764f6ee0886	19 seconds ago	946MB
<none>	<none>	a2cc4094c44d	7 minutes ago	946MB
theocrithary/helloworld	latest	4a1e123ea59b	7 hours ago	883MB
mongo	latest	ba0c2ff8d362	3 weeks ago	492MB

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>





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

- 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
```

- 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
```

- 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
```

- 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
```

- Clone the following repository (if not already);

<https://github.com/theocrithary/Piper-2021/tree/main/Containers%20%20Kubernetes/Lab%2003%20-%20Part%202%20-%20Deploying%20to%20Kubernetes>

- Copy all files from 'Lab 03 - Part 2 - Deploying to Kubernetes' to your working project directory (e.g. D:\MyProject)
- 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
=> => transferring context: 2B
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 32B
=> [internal] load metadata for docker.io/library/python:latest
=> [1/4] FROM docker.io/library/python:latest@sha256:0a9325e9adb51dd65ec915036e270eae330887918fc1254f6a22e436741600b17
=> [internal] load build context
=> => transferring context: 482B
=> CACHED [2/4] WORKDIR /usr/src/app
=> CACHED [3/4] COPY . .
=> CACHED [4/4] RUN pip install --no-cache-dir -r requirements.txt
=> exporting to image
=> => exporting layers
=> => writing image sha256:f877e9e4b4f89a3937f5eefc23dfa409261c5c32b931f06e9b913041955f270b
=> => naming to docker.io/theocrithary/photo-album-k8s
```

> 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
a134306bbbb39: 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

The screenshot shows a web browser window with the title "Album example". The address bar indicates "Not secure | 192.168.140.212:7003". The main content area displays a "Photo upload" form. The form includes fields for "Title" and "Comments", both of which are currently empty. Below the form is a file input field labeled "Choose File" with the placeholder "No file chosen" and a blue "Upload" button. At the bottom of the page, there is a preview of a uploaded image, which is a Dell EMC server component. The image has a caption "t2" below it. At the very bottom of the page, there is some small text about the source of the template and a "Back to top" link.



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
```

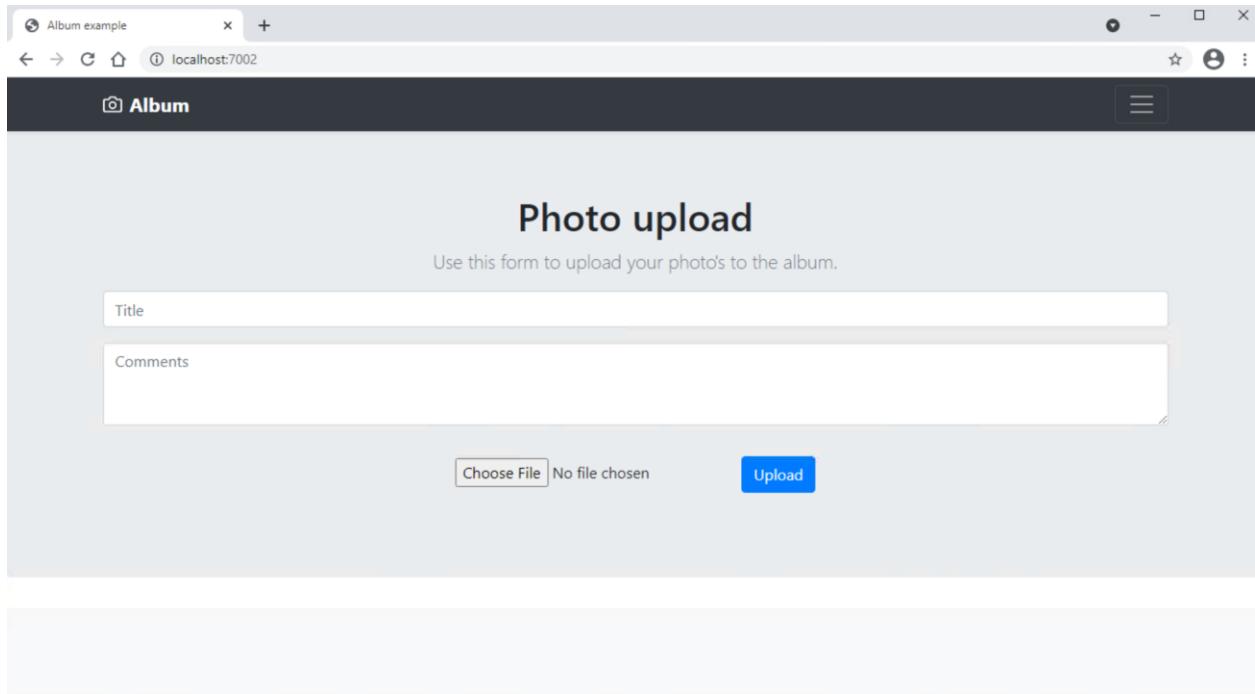
- Forward port to allow local testing

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

A screenshot of a web browser window titled "Album example". The address bar shows "localhost:7002". The main content area has a dark header with "Album" and a menu icon. Below it is a section titled "Photo upload" with the sub-instruction "Use this form to upload your photo's to the album.". There are two input fields: "Title" and "Comments", both currently empty. Below these is a file upload field labeled "Choose File" with the message "No file chosen" and a blue "Upload" button. At the bottom is a preview area showing a small image of a molecular structure and the word "test" below it.



- 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

The screenshot shows a web browser window titled "Album example". The address bar indicates the URL is "192.168.140.212:7003". The page content is as follows:

- Photo upload**: A form with fields for "Title" and "Comments".
- Choose File**: A button with the text "No file chosen".
- Upload**: A blue button.
- Thumbnail Preview**: A small image of a server rack with the Dell EMC logo, labeled "t2".
- Page Footer**: Text including "Album example is © Bootstrap, but please download and customize it for yourself!", "Back to top", and "New to Bootstrap? Visit the [homepage](#) or read our [getting started guide](#).



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



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.dell EMC.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
```



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
1  apiVersion: v1
2  kind: PersistentVolumeClaim
3  metadata:
4    name: pvol0
5  spec:
6    accessModes:
7      - ReadWriteOnce
8    volumeMode: Filesystem
9    resources:
10      requests:
11        storage: 8Gi
12    storageClassName: isilon
13
14
15
16
17
test_pod.yaml
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: isilon-test-pod1
5  spec:
6    containers:
7      - name: test
8        image: docker.io/centos:latest
9        command: [ "/bin/sleep", "3600" ]
10       volumeMounts:
11         - name: workdir
12           mountPath: "/data0"
13             name: pvol0
14       volumes:
15         - name: pvol0
16           persistentVolumeClaim:
17             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/isilon-test-pod1 created
```

11. Verify the PV was created successfully



> 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

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

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:   cn1.projectcalico.org/podIP: 192.0.2.229/32
                cn1.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://f36115f821b6082c3aab9a5e319a5a402c457545c2698c6ed4c3ae3b8f9b77a
    Image:         docker.io/centos:latest
    Image ID:     docker.io/library/centos@sha256:5528e8b1b1719d34604c87e11dc1c0a20bedf46e83b5632cdeac91b8c04efc1
    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 isilon-test-pod1 -- bash
```

```
D:\Piper-2021\Containers & Kubernetes\Lab 04 - PVs with Isilon>kubectl exec -it isilon-test-pod1 -- bash
[root@isilon-test-pod1 ~]#
```

16. Verify the NFS mount and confirm we can write to the file system

```
> cd /data0
```

```
> touch test
```

```
> ls -al
```

```
[root@isilon-test-pod1 ~]# cd /data0
[root@isilon-test-pod1 data0]# touch test
[root@isilon-test-pod1 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@isilon-test-pod1 data0]# df -h | grep data0
192.168.1.12:/ifs/data/csi/k8s-24d056fde3 19G 1.2G 18G 7% /data0
[root@isilon-test-pod1 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@isilon-test-pod1 data0]#
```

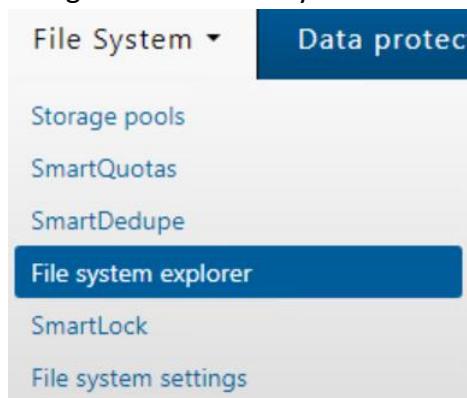
18. Open a chrome browser and login to the Isilon cluster using the bookmark link and the credentials: isilon / isilon

The screenshot shows a browser window with the following details:

- Title Bar:** Dell EMC Isilon OneFS
- Address Bar:** Not secure | 192.168.1.12:8080/#Login
- Navigation:** Back, Forward, Stop, Home
- Bookmarks:** Apps, Dell EMC Isilon One..., vSphere Client, Kubernetes Dashboard, Harbor
- Content Area:**
 - OneFS STORAGE ADMINISTRATION**
 - Links:**
 - Web: www.isilon.com
 - Support: 800 782 4362 | Worldwide +1 508 497 7901
<http://support.emc.com>
 - Sales: Isilon sales
866 GETEMC2 | 866 438 3622
info@isilon.com
 - Copyright Notice:** Copyright © 2001-2019 Dell Inc. or its subsidiaries. All Rights Reserved. This software is protected, without limitation, by copyright law and international treaties. Use of this software is subject to the license agreement contained therein. An express limitation to the terms and conditions of the License Agreement under which it is provided by or on behalf of Dell.
 - Login Form:**
 - Cluster name: Isilon
 - User name: Isilon
 - Password: *****
 - Log in button
 - DELL EMC Logo:**



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.

File system explorer

File system explorer																			
Path: /ifs/data/csi/k8s-334187667b		<input type="button" value="Browse..."/>	<input type="button" value="Search"/>																
<input type="button" value="Create directory"/>																			
<table border="1"> <thead> <tr> <th>Name</th> <th>Size</th> <th>Updated</th> <th>Actions</th> </tr> </thead> <tbody> <tr> <td> k8s-334187667b</td> <td>32 B</td> <td>2020-07-07 23:39</td> <td><input type="button" value="View details"/></td> </tr> <tr> <td> Parent directory</td> <td></td> <td></td> <td></td> </tr> <tr> <td> test</td> <td>0 B</td> <td>2020-07-07 23:50</td> <td><input type="button" value="View details"/></td> </tr> </tbody> </table>				Name	Size	Updated	Actions	k8s-334187667b	32 B	2020-07-07 23:39	<input type="button" value="View details"/>	Parent directory				test	0 B	2020-07-07 23:50	<input type="button" value="View details"/>
Name	Size	Updated	Actions																
k8s-334187667b	32 B	2020-07-07 23:39	<input type="button" value="View details"/>																
Parent directory																			
test	0 B	2020-07-07 23:50	<input type="button" value="View details"/>																

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

NFS exports

NFS exports			
<input type="button" value="Create export"/>			
<input type="button" value="Bulk actions"/> <input type="button" value="▼"/>			
Export ID	Paths	Description	Action
1	/ifs	Default export	<input type="button" value="View / Edit"/> <input type="button" value="Delete"/>
4	/ifs/data/csi/k8s-334187667b		<input type="button" value="View / Edit"/> <input type="button" value="Delete"/>



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



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



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!



User Name:

Password: [@](#)

[Forgot password?](#)

[Log In](#)

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

The screenshot shows the PowerProtect Data Manager interface. On the left, there is a navigation sidebar with the following structure:

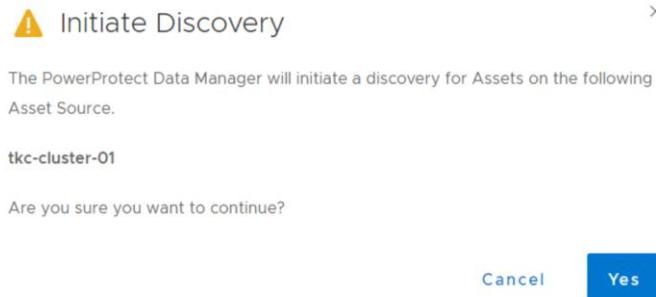
- Dashboard
- Infrastructure
 - Assets
 - Asset Sources** (highlighted with a red box)
 - Storage
 - Protection Engines
 - Application Agents
 - Search Engine
 - Networks
- Protection

The main content area is titled "Asset Sources". It has tabs for "vCenter" and "Kubernetes" (which is highlighted with a red box). Below the tabs are buttons for "Add", "Edit", "Discover", and "Delete", and a link "How to add a Kubernetes Cluster". A table lists two entries:

	Name	Address
<input type="checkbox"/>	centos-k8s	192.168.1.11
<input type="checkbox"/>	tkc-cluster-01 (highlighted with a red box)	192.168.140.210



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

Name	Asset Type	Asset Count	Protected Asset Size	Last Run Status
tkc-cluster-01-protection-policy	Kubernetes	1	13.96 GB	Completed
default-vm-protection-policy	Virtual Machine	1	21.47 GB	Completed
centos-k8s-protection-policy	Kubernetes	1	8.59 GB	Completed

6. Select the item named ‘tkc-cluster-01-protection-policy’ and click the “Edit” button

Name	Asset Type	Asset Count
tkc-cluster-01-protection-policy	Kubernetes	1
default-vm-protection-policy	Virtual Machine	1
centos-k8s-protection-policy	Kubernetes	1

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

The screenshot shows the left sidebar with 'Infrastructure' and 'Assets' selected. The main area is titled 'Assets' and shows a 'Virtual Machine' section with a 'Kubernetes' button highlighted by a red box. Below it is a 'Kubernetes' section also highlighted with a red box. Buttons for 'View Copies', 'Back Up Now', and 'Assign Network' are visible.

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

	Details	Namespace	Network	Status	Labels	Age	Protection Policy	Size	Cluster	PVCs Excluded
<input type="checkbox"/>	test			Available		3 weeks	tkc-cluster-01-protection-policy	14.0 GB	tkc-cluster-01	
<input type="checkbox"/>	test			Available		3 months	centos-k8s-protection-policy	8.6 GB	centos-k8s	0 of 1

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.

The dialog box contains the message: "The backup operation has been triggered successfully." with an information icon. It has 'Go to Jobs' and 'Ok' buttons at the bottom.

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

Details	Status	Description	Policy Name	Job Type
<input type="radio"/>		Manual Protecting Kubernetes - tkc-cluster-01-protection-policy - PROTECTION - Synthetic Full	tkc-cluster-01-protection-policy	Protect
Details	Status	Description	Policy Name	Job Type
<input type="radio"/>		Manual Protecting Kubernetes - tkc-cluster-01-protection-policy - PROTECTION - Synthetic Full	tkc-cluster-01-protection-policy	Protect
Details	Status	Description	Policy Name	Job Type
<input type="radio"/>		Manual Protecting Kubernetes - tkc-cluster-01-protection-policy - PROTECTION - Synthetic Full	tkc-cluster-01-protection-policy	Protect

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



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

The screenshot shows the Dell Protection UI. On the left, a navigation sidebar is open, showing categories like Dashboard, Infrastructure, Protection, Recovery, Assets, and Running Sessions. The 'Recovery' and 'Assets' buttons are highlighted with red boxes. On the right, the 'Recovery' screen is displayed for 'Kubernetes'. It has tabs for Virtual Machine, File System, SQL, and Kubernetes (which is highlighted with a red box). Below the tabs are buttons for View Copies, Restore, and Search. A table lists two items, both labeled 'test', each with a checkbox and a magnifying glass icon.

12. Select the ‘test’ item with the cluster name “tkc-cluster-01” and click on “Restore”

A screenshot of the 'Available' section of the Recovery screen. It shows a single item named 'test' with a checkbox and a magnifying glass icon. To the right, it displays the item's details: Original Namespace: test, Size: 8.6 GB, Backup: Dec 8, 2020, 11:19:47 PM.

13. Click “Next”

A screenshot of the 'Select Copy' step of the Recovery wizard. On the left, a vertical navigation bar lists steps 1 through 6. Step 1 is 'Select Copy' (highlighted with a blue bar), step 2 is 'Cluster', step 3 is 'Purpose', step 4 is 'Restore Type', step 5 is 'PVCs', and step 6 is 'Summary'. The main panel shows a 'Select Copy' section with the details: Original Namespace: test, Size: 8.6 GB, Backup: Dec 8, 2020, 11:19:47 PM. Below it is a 'Selected Copy' table with one row:

Create Time	Size	Retention	Consistency	Excluded PVCs
Dec 8, 2020, 11:19...	16.9 KB	Dec 9, 2020, 11:19:47 P	Crash Consistent	No

14. Select ‘Restore to an Alternate Cluster’ option and select the “centos-k8s” cluster from the dropdown menu, then click “Next”

A screenshot of the 'Cluster' selection step of the Recovery wizard. On the left, a vertical navigation bar lists steps 1 through 6. Step 2 is 'Cluster' (highlighted with a green checkmark). The main panel shows a 'Cluster' section with the details: Original Namespace: test, Size: 8.6 GB, Backup: Dec 8, 2020, 11:19:47 PM. Below it is a note: 'Select the cluster to which you would like to restore. The original is selected by default.' There are two radio buttons: 'Restore to Original Cluster' (unselected) and 'Restore to an Alternate Cluster' (selected). A dropdown menu shows the available clusters: 'centos-k8s' (highlighted with a red box) and 'tkc-cluster-01'.



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

Recovery

1 Select Copy	Purpose	Original Namespace: test Size: 8.6 GB Backup: Dec 8, 2020, 11:19:47 PM
2 Cluster		
3 Purpose	Select what you would like to restore.	
4 Restore Type	<input checked="" type="radio"/> Restore Namespace and Select PVCs <small>Restore the namespace and a subset of PVCs in the namespace.</small>	<small>i</small>
5 PVCs	<input type="checkbox"/> Include cluster scoped resources <small><small>i</small></small>	<small>i</small>
6 Summary	<input type="radio"/> Restore Only PVCs <small>Restore only selected PVCs.</small>	<small>i</small>

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

Recovery

1 Select Copy	Restore Type	Original Namespace: test Size: 8.6 GB Backup: Dec 9, 2020, 4:18:37 AM
2 Cluster		
3 Purpose	Select the namespace to which you would like to restore.	<small>i</small>
4 Restore Type	<input checked="" type="radio"/> Restore to a New Namespace <small>Restore the selected assets to a new namespace.</small>	<input type="text" value="test-restore"/>
5 PVCs	<input type="radio"/> Restore to an Existing Namespace <small>Restore the selected assets to an existing namespace.</small>	<small>i</small>
6 Summary		

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

Recovery

1 Select Copy	PVCs	Original Namespace: test Size: 8.6 GB Backup: Dec 8, 2020, 11:19:47 PM				
2 Cluster						
3 Purpose	Select PVCs to be restored to the new namespace.					
4 Restore Type	PVCs to Restore					
5 PVCs	<table border="1"> <thead> <tr> <th>PVC Name</th> <th>Size</th> </tr> </thead> <tbody> <tr> <td>pvol0</td> <td>8.6 GB</td> </tr> </tbody> </table>	PVC Name	Size	pvol0	8.6 GB	
PVC Name	Size					
pvol0	8.6 GB					
6 Summary						

18. Review the Summary page and click “Restore”



Recovery

1 Select Copy ✓

2 Cluster ✓

3 Purpose ✓

4 Restore Type ✓

5 PVCs ✓

6 Summary

Summary

Original Namespace: test | Size: 8.6 GB | Backup: Dec 9, 2020, 4:18:37 AM

Namespace and Restore Selections

Original Namespace	Backup to Be Restored	Size
test	Dec 9, 2020, 4:18:37 AM	8.6 GB

Cluster: Alternate Cluster - centos-k8s

[Edit](#)

Purpose: Restore Namespace and Select PVCs

[Edit](#)

Info Namespace scoped resources, including pods, services, secrets, and deployments, will not be overwritten during a restore. All resources that do not currently exist in the namespace will be restored.

Restore Type: New Namespace - test-restore

[Edit](#)

PVCs: Overwrite content of existing PVCs

[Edit](#)

PVC	Backup to Be Restored	Size

[Cancel](#)[Back](#)[Restore](#)

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

Info

Info The restore process was successfully initiated. Progress can be monitored via Jobs.

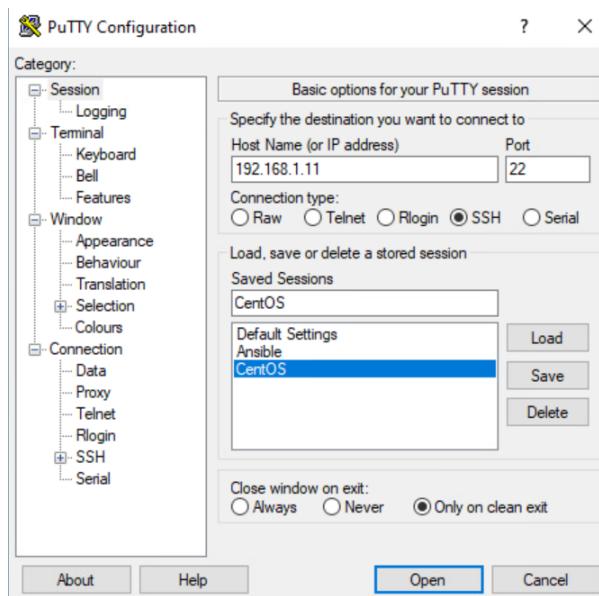
[Go to Jobs](#)[Ok](#)

Details	Status	Description	Policy Name	Job Type	Asset Type
View	Running	Restore namespace to New: Restoring copy to new namespace "test-restore"	tfc-cluster-01-protection-policy	Restore	Kubernetes
View	Running	Restore namespace to New: Restoring copy to new namespace "test-restore"	tfc-cluster-01-protection-policy	Restore	Kubernetes
View	Success	Restore namespace to New: Restoring copy to new namespace "test-restore"	tfc-cluster-01-protection-policy	Restore	Kubernetes

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!’

```
root@k8smaster:~  
└─ login as: root  
└─ root@192.168.1.11's password:  
Last login: Wed Dec  9 03:44:53 2020 from 192.168.1.10  
[root@k8smaster ~]#
```

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

> kubectl cluster-info

```
[root@k8smaster ~]# kubectl cluster-info  
Kubernetes master is running at https://192.168.1.11:6443  
KubeDNS is running at https://192.168.1.11:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy  
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.  
[root@k8smaster ~]#
```

23. Run the following to check that the pod was restored in the namespace ‘test-restore’

> kubectl get pods -n test-restore

```
[root@k8smaster ~]# kubectl get pods -n test-restore  
NAME          READY   STATUS    RESTARTS   AGE  
isilontestpod1  1/1     Running   0          35s
```

24. Run the following command to confirm the PVC was attached to the pod successfully

> kubectl describe pod isilontestpod1 -n test-restore



```
[root@k8smaster ~]# kubectl get pods -n test-restore
NAME        READY   STATUS    RESTARTS   AGE
isilonetestpod1  1/1    Running   0          35s
[root@k8smaster ~]# kubectl describe pod isilonetestpod1 -n test-restore
Name:           isilonetestpod1
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@sha256:5528e8b1b1719d34604c87e11dc1c0a20bedf46e83b5632cdeac91b8c04efc1
    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 pv0l0 (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-bth4p (ro)
Conditions:
Type        Status
Initialized  True
Ready       True
ContainersReady  True
PodScheduled  True
Volumes:
pv0l0:
  Type:     PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
  ClaimName:  pv0l0
  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/isilonetestpod1 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 isilonetestpod1 -n test-restore -- bash
```

```
[root@k8smaster ~]# kubectl exec -it isilonetestpod1 -n test-restore -- bash
[root@isilonetestpod1 ~]#
```

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@isilonetestpod1 ~]# cd /data0
[root@isilonetestpod1 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@isilonetestpod1 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
```



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