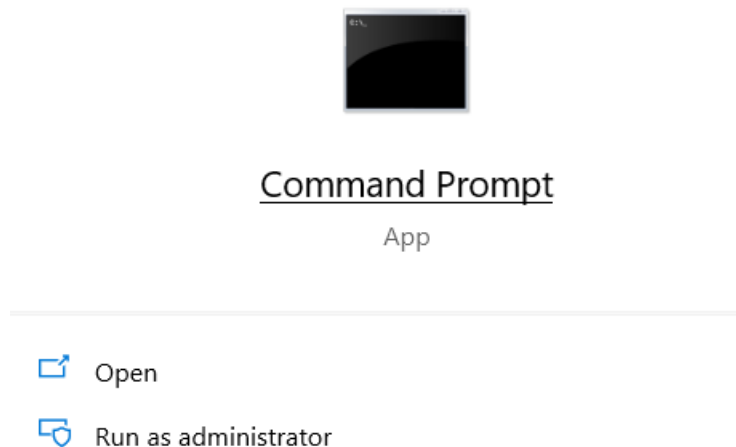


Experiment 06: Working with K3d and Local Persistent Volumes

Note: Refer to the Rancher LAB if you haven't already installed k3d for installation.

Make the local directory that we'll mount into our k3d containers.

Open a command prompt in "Run as administrator" mode

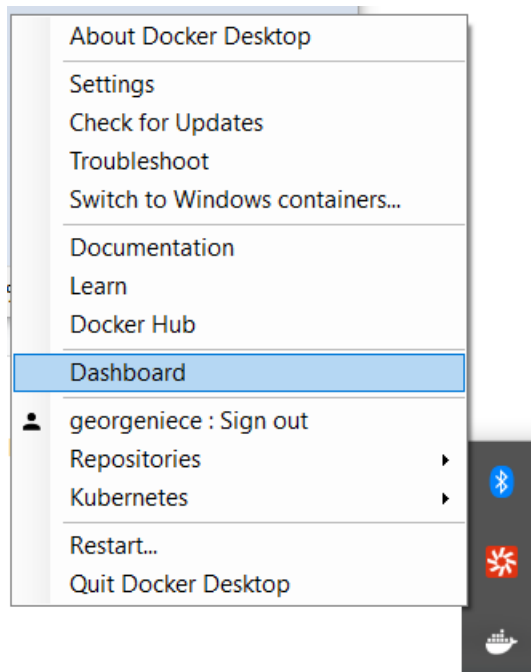


Make the folders that we'll mount into our containers for this experiment

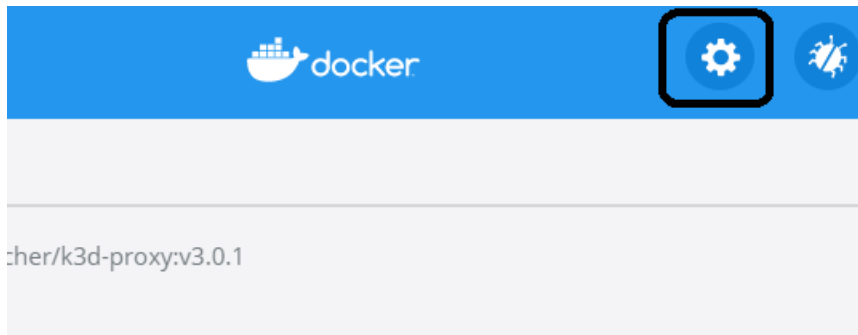
```
C:\k3d> mkdir c:\tmp
```

```
C:\k3d> mkdir c:\tmp\k3dvol
```

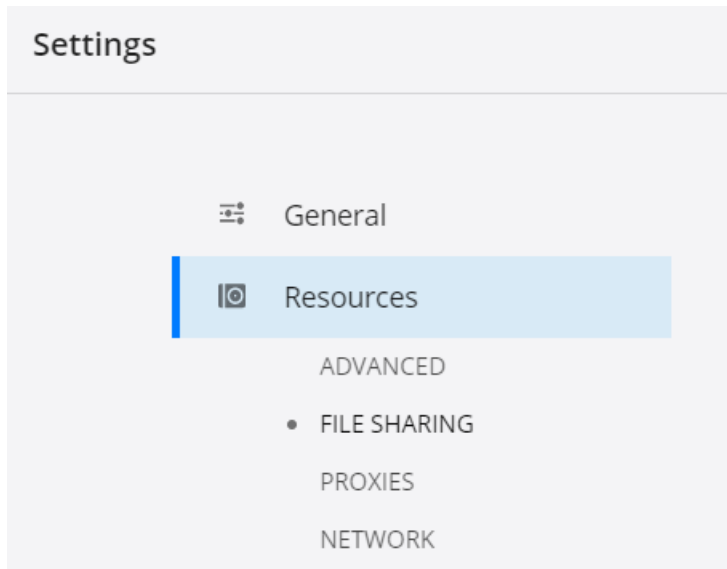
Open Docker for Desktop



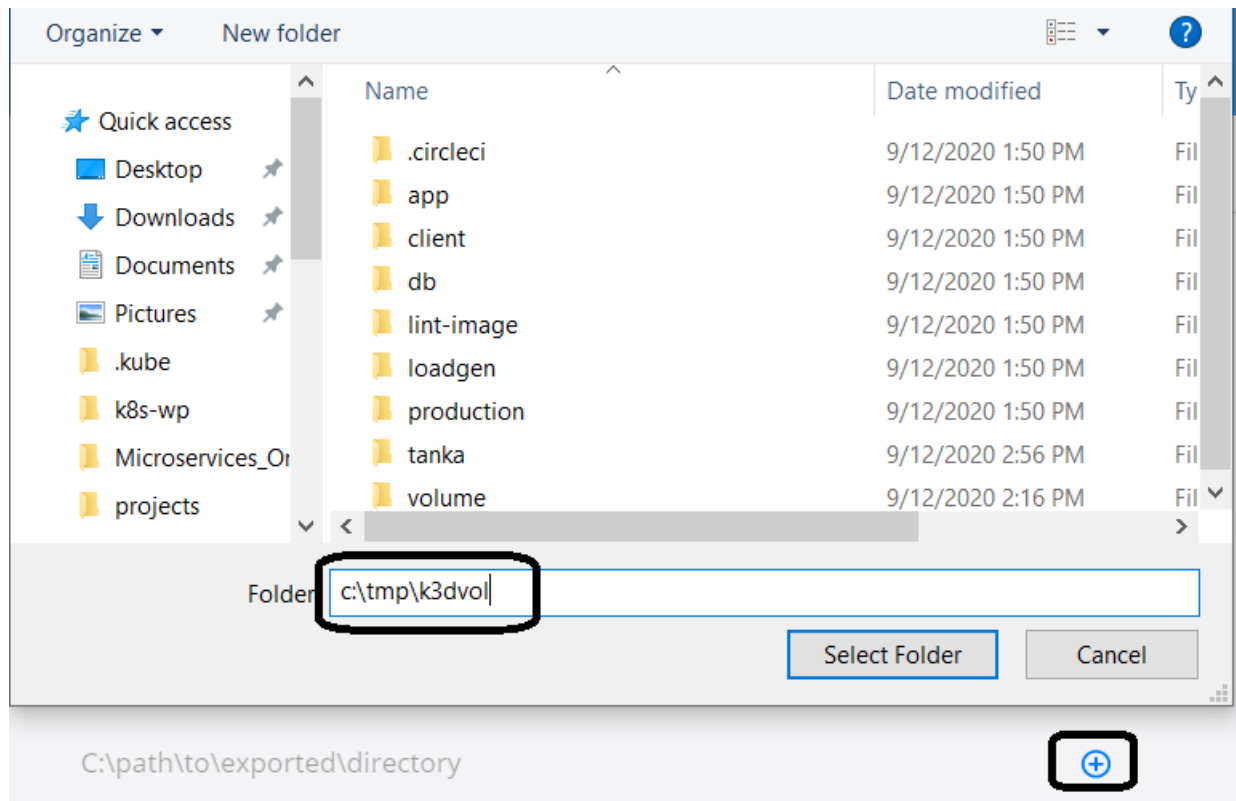
Select the settings icon



Under settings we'll expand the **Resources** and select **FILE SHARING**. From this Settings pane we can select folders that we want to expose as mount points within our Docker containers.

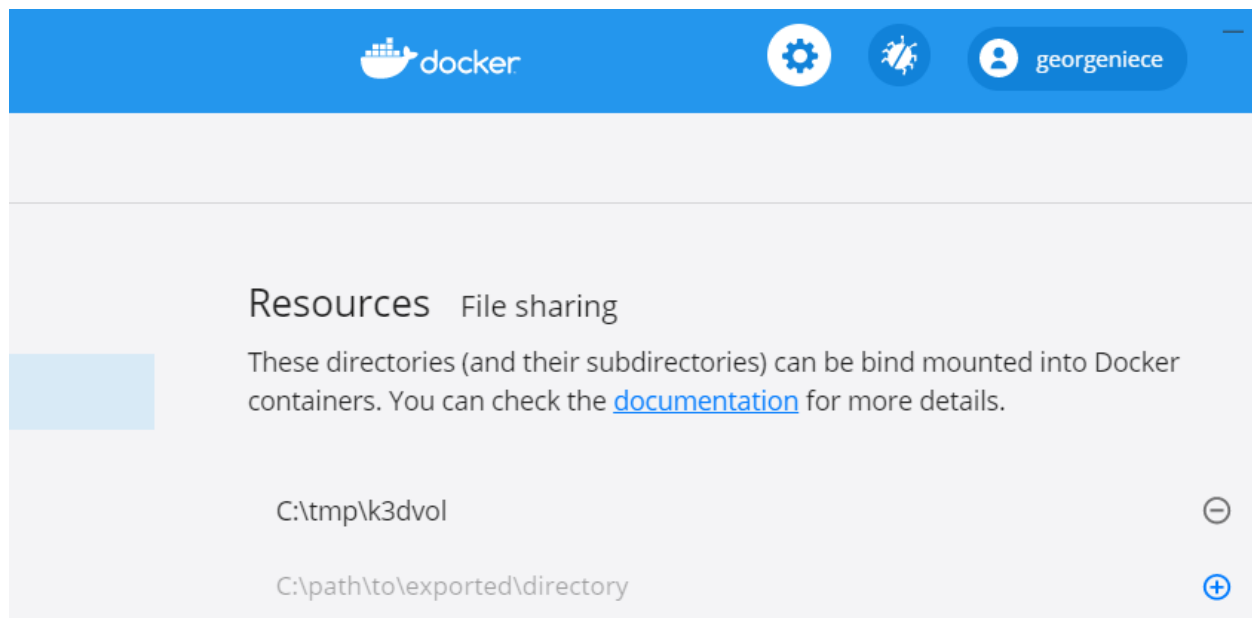


Select the + next to c:\path\to\exported\directory



Enter the folder that we created to be bind mounted into our Docker containers. For this experiment that should be **c:\tmp\k3dvol**, then select **Select Folder**

Close, or minimize, the Docker for Desktop dashboard, noting that our new folder is now available in Docker to be mounted for this experiment.



Create the cluster for this experiment, using the volume option to mount our local folder into each node in the cluster. For this cluster we'll expose port 80 against the load balancer and set our agents to 1. Servers option defaults to 1 when not specified.

```
C:\k3d> k3d cluster create "k3d-cluster" --volume /c/tmp/k3dvol:/tmp/k3dvol --port "80:80@loadbalancer" --agents 2
```

```
[33mWARN[0m[0000] Failed to stat file/directory/named volume that you're trying to mount:
'/c/tmp/k3dvol' in '/c/tmp/k3dvol:/tmp/k3dvol' -> Please make sure it exists
[33mWARN[0m[0000] No node filter specified
[36mINFO[0m[0000] Created network 'k3d-k3d-cluster'
[36mINFO[0m[0000] Created volume 'k3d-k3d-cluster-images'
[36mINFO[0m[0001] Creating node 'k3d-k3d-cluster-server-0'
[36mINFO[0m[0001] Creating node 'k3d-k3d-cluster-agent-0'
[36mINFO[0m[0001] Creating node 'k3d-k3d-cluster-agent-1'
[36mINFO[0m[0002] Creating LoadBalancer 'k3d-k3d-cluster-serverlb'
[36mINFO[0m[0008] Cluster 'k3d-cluster' created successfully!
[36mINFO[0m[0008] You can now use it like this:
kubectl cluster-info
```

Note: by using the 80:80@loadbalancer we're creating a passthrough for HTTP at the loadbalancer for our configuration.

Set our **KUBECONFIG_FILE** environment variable to the file we'll load our k8s configuration for kubectl usages

```
C:\k3d> set KUBECONFIG_FILE=.\kube\k3d-cluster
```

Put our cluster configuration for k3d-cluster into our file

```
C:\k3d> k3d kubeconfig get k3d-cluster > %KUBECONFIG_FILE%
```

```
C:\k3d> type %KUBECONFIG_FILE%
```

apiVersion: v1

clusters:

- cluster:

certificate-authority-data:

```
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB0tLS0tCk1JSUJWekNCL3FBREFnRUNBZ0VBTUFvR
0NDcUdTTTQ5QkFNQ01DTXhJVEFmQmdOVkJBTU1HR3N6Y3kxelpYSjIKWlhJdFkyRkFNVF
U1T1RZM01qUTRPEFIRncweU1EQTVNRGt4TnpJNE1EbGFGdzb6TURBNU1EY3hOekk0TU
RsYQpNQ014SVRBZklnTlZCQU1NR0dzemN5MXPawEoyWlhJdFkyRkFNVFU1T1RZM01qUT
RPVEJaTUJNR0J5cUdTTTQ5QkFNRUdDQ3FHU000OUF3RUhBMEIBQkdmRm53RUtycFVtbV
h3ckVFUFdaYSsxZWdYQWhPV2ZUZEorZU94UWo4U3kKUDgzSTJQbDYrTUQ4OUNMTIRtb
E1Ebk5pM3FvS1N0ZHdGZFRhOFRRHQUxTS2pJekFoTUE0R0ExVWREd0VCL3dRRQpBd0IDc
ERBUEJnTlZlUk1CQWY4RUJUQUJBUUgvdUFvR0NDcUdTTTQ5QkFNQ0EwZ0FNRVVSUF
```

```

VOGpaQ0RORkhMCkpDVkdOd2I2UXhxS0xPekp1NUtYV2JNdGZ0VVB4Ymc4QWIFQXNkQXF
JRm90R2JPcV4OUxudU45eStrTU44M1AKU1pPWWRGMElyNUV2dXgwPQotLS0tLUVORCB
DRVJUSUZJQ0FURS0tLS0tCg==
  server: https://0.0.0.0:6550
  name: k3d-k3d-cluster
contexts:
- context:
  cluster: k3d-k3d-cluster
  user: admin@k3d-k3d-cluster
  name: k3d-k3d-cluster
current-context: k3d-k3d-cluster
kind: Config
preferences: {}
users:
- name: admin@k3d-k3d-cluster
  user:
    password: dd79f910ebe64a30855bcd38b7425b98
    username: admin

```

C:\k3d> **set KUBECONFIG=%KUBECONFIG_FILE%**

List our clusters to view the

C:\k3d> **k3d cluster list k3d-cluster**

NAME	SERVICES	AGENTS	LOADBALANCER
k3d-cluster	1/1	2/2	true

C:\k3d> **kubectl cluster-info**

Kubernetes master is running at https://0.0.0.0:51472
 CoreDNS is running at https://0.0.0.0:51472/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
 Metrics-server is running at https://0.0.0.0:51472/api/v1/namespaces/kube-system/services/https:metrics-server:/proxy

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

C:\k3d> **kubectl cluster-info**

Unable to connect to the server: dial tcp [::1]:8080: connectex: No connection could be made because the target machine actively refused it.

Troubleshooting Note: If you have an error similar to above when executing **kubectl**, ensure you correctly set the **KUBECONFIG** in previous steps in this experiment.

Review the enhanced listing for the cluster-info

```
C:\k3d> kubectl cluster-info dump
```

View the information for the exposed traefik loadbalancer IP/hostname

```
C:\k3d> kubectl get svc traefik --namespace kube-system -w
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
traefik	LoadBalancer	10.43.245.42	172.18.0.2	80:32162/TCP,443:31433/TCP	42m

```
C:\k3d> kubectl describe svc traefik --namespace kube-system | grep Ingress
```

```
LoadBalancer Ingress: 172.18.0.2
```

Open an editor and paste the following yaml file that will create a **busybox** with a simple ping, as well as exposing our local **c:\tmp\k3dvol** as a mount on **/data** within our container.

```
C:\k3d> notepad app.yaml
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/tmp/k3dvol"
```

```
---
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

```
---
```

```
apiVersion: apps/v1
kind: Deployment
```

```

metadata:
  name: echo
spec:
  selector:
    matchLabels:
      app: echo
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: echo
    spec:
      volumes:
        - name: task-pv-storage
          persistentVolumeClaim:
            claimName: task-pv-claim
      containers:
        - image: busybox
          name: echo
          volumeMounts:
            - mountPath: "/data"
              name: task-pv-storage
          command: ["ping", "127.0.0.1"]

```

C:\k3d> **kubectl apply -f app.yaml**

persistentvolume/task-pv-volume created
 persistentvolumeclaim/task-pv-claim created
 deployment.apps/echo created

C:\k3d> **kubectl get pv**

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM
task-pv-volume	1Gi	RWO	Retain	Bound	default/task-pv-claim
				38s	manual

View our Persistent Volume Claim

C:\k3d> **kubectl get pvc**

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS
task-pv-claim	Bound	task-pv-volume	1Gi	RWO	manual
					73s

View our active pod

C:\k3d> **kubectl get pods**

NAME	READY	STATUS	RESTARTS	AGE
------	-------	--------	----------	-----

echo-859c44dcc6- pfc7m 1/1 Running 0 118s

Exec a shell into the container

C:\k3d> **kubectl exec -it echo-859c44dcc6-pfc7m -- sh**

Note: In the deprecated syntax we could have left off the double hyphen before the sh command, but with the pace of change in Kubernetes, and tools like k3d/kind, better to try to stay as current as possible

\$ echo \$(hostname)

\$ echo \$(hostname) > /data/hostname.txt

\$ cat /data/hostname.txt

\$ exit

C:\k3d> **kubectl get nodes -o wide**

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP
OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME				
k3d-k3d-cluster-agent-0	Ready	<none>	53m	v1.18.6+k3s1	172.18.0.3	<none>
Unknown	4.19.76-linuxkit	containerd://1.3.3-k3s2				
k3d-k3d-cluster-agent-1	Ready	<none>	53m	v1.18.6+k3s1	172.18.0.4	<none>
Unknown	4.19.76-linuxkit	containerd://1.3.3-k3s2				
k3d-k3d-cluster-server-0	Ready	master	53m	v1.18.6+k3s1	172.18.0.2	<none>
Unknown	4.19.76-linuxkit	containerd://1.3.3-k3s2				

Delete our identified pod, that we'd exec'd into and created our **hostname.txt** file

C:\k3d> **kubectl delete pod/echo-859c44dcc6-pfc7m**

pod "echo-859c44dcc6-pfc7m" deleted

c:\k3d> **type c:\tmp\k3dvol\hostname.txt**

echo-859c44dcc6-pfc7m

This is Kubernetes, and we're using a configuration requiring that pod for our application, so not surprisingly, we'll do another get pods with kubectl and see a new pod created

C:\k3d> **kubectl get pods -o wide**

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
NOMINATED NODE	READINESS	GATES				


```
echo-859c44dcc6-7mnnr 1/1 Running 0 98m 10.42.2.4 k3d-k3d-cluster-server-0
<none> <none>
```

Exec a sh into our new pod, but this time we'll leave off the double hyphen to see the deprecation warning.

```
C:\k3d> kubectl exec -it echo-859c44dcc6-7mnnr sh
```

```
/ # cat /data/hostname.txt
```

```
echo-859c44dcc6-pfc7m
```

```
/ # echo $(hostname)
```

```
echo-859c44dcc6-7mnnr
```

```
/ # exit
```

Delete our cluster for this experiment

```
C:\k3d> k3d cluster delete k3d-cluster
```

```
[36mINFO[0m[0000] Deleting cluster 'k3d-cluster'
[36mINFO[0m[0000] Deleted k3d-k3d-cluster-serverlb
[36mINFO[0m[0000] Deleted k3d-k3d-cluster-agent-1
[36mINFO[0m[0000] Deleted k3d-k3d-cluster-agent-0
[36mINFO[0m[0000] Deleted k3d-k3d-cluster-server-0
[36mINFO[0m[0000] Deleting cluster network
'f7f0376fbd55c7f4709ad960ad86c6501ed0a05a19a6d9757914370875a76600'
[36mINFO[0m[0001] Deleting image volume 'k3d-k3d-cluster-images'
[36mINFO[0m[0001] Removing cluster details from default kubeconfig...
[36mINFO[0m[0001] Removing standalone kubeconfig file (if there is one)...
[36mINFO[0m[0001] Successfully deleted cluster k3d-cluster!
```

```
C:\k3d> k3d cluster list
```

NAME	SERVICES	AGENTS	LOADBALANCER
------	----------	--------	--------------