KUBERNETES EPHEMERAL STORAGE

When a container is created, it starts with the read-only layers of its image. As the application runs and generates data, all changes are written to a writable layer on top of those read-only layers.

Docker manages the data written to the writable layer using storage drivers throughout the container's lifecycle. However, once the container is deleted, the data in this writable layer is lost.

Kubernetes has similar mechanism for containers inside the pod where data on writable layers is managed by ephemeral storage through out the pod life.

Types of ephemeral storage are:

1) EmptyDir Volume:

When this type of storage is assigned to a Pod, it can be shared among all containers within that Pod, as long as the volume is mounted on to each container. It's a scratch space used for caching, temporary computations available through out the lifetime of pod and isn't persistent in nature.

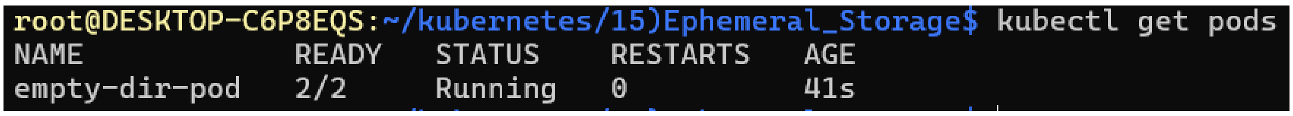
Lets Explore it practically:

EmptyDir.yaml:

```
apiVersion: v1
kind: Pod
metadata:
  name: empty-dir-pod
  labels:
    app: my-pod
spec:
  volumes:
    - name: temp-storage
      emptyDir: {}
  containers:
    - name: busybox-1
      image: busybox
      command: ["/bin/sh", "-c", "sleep 3600"]
      volumeMounts:
        - name: temp-storage
          mountPath: /data
    - name: busybox-2
      image: busybox
      command: ["/bin/sh", "-c", "sleep 3600"]
      volumeMounts:
        - name: temp-storage
          mountPath: /data
```

Apply;

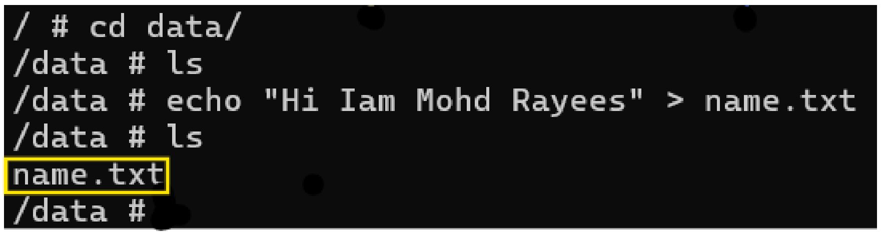
```
kubectl apply -f emptyDir-practice.yaml
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/15)Ephemeral_Storage$ kubectl get pods
NAME            READY   STATUS    RESTARTS   AGE
empty-dir-pod   2/2     Running   0          41s
```

Lets exec into container-1 of pod and create some data inside /data location of container.

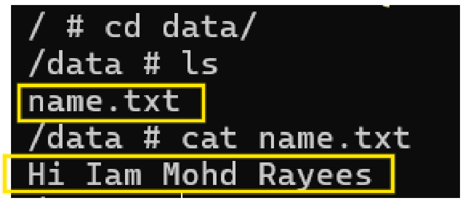
```
kubectl exec -it empty-dir-pod -c busybox-1 --
/bin/sh
```



```
/ # cd data/
/data # ls
/data # echo "Hi Iam Mohd Rayees" > name.txt
/data # ls
name.txt
/data #
```

Noe exec into container-2 and check if this file is accessible:

```
kubectl exec -it empty-dir-pod -c busybox-2 --
/bin/sh
```



```
/ # cd data/
/data # ls
name.txt
/data # cat name.txt
Hi Iam Mohd Rayees
```

File name.txt is accessible from /data directory of container-2. It was possible because of mounting same emptyDir volume on both containers. Hence writable layers are shared between containers inside a pod.

2) Downward Api:

1. Downward API helps containers to know the metadata such as Pod name, Namespace, Resources, labels, annotation etc of their pod.
2. This is very useful in case of sidecar containers used for logging and monitoring of main application container. So that sidecar container is aware of details of the pod , for which it is working.
3. It is injected into containers in a Pod either as environment variables or through mounted files via volumes.
4. This allows Pods to retrieve runtime-specific details dynamically, without hardcoding or manual intervention.

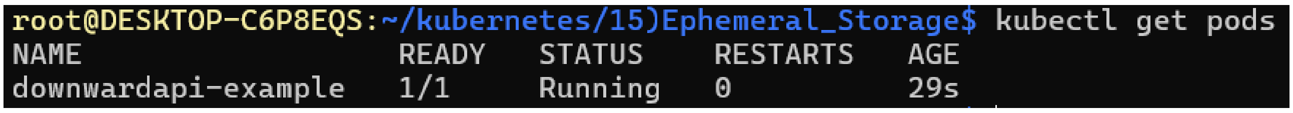
Lets Explore it Practically:

downwardApi.yaml:

```
apiVersion: v1
kind: Pod
metadata:
  name: downwardapi-example
  labels:
    app: myapp
spec:
  containers:
    - name: metadata-container
      image: busybox
      command: ["/bin/sh", "-c", "env && sleep 
3600"]
      env:
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
```

Apply;

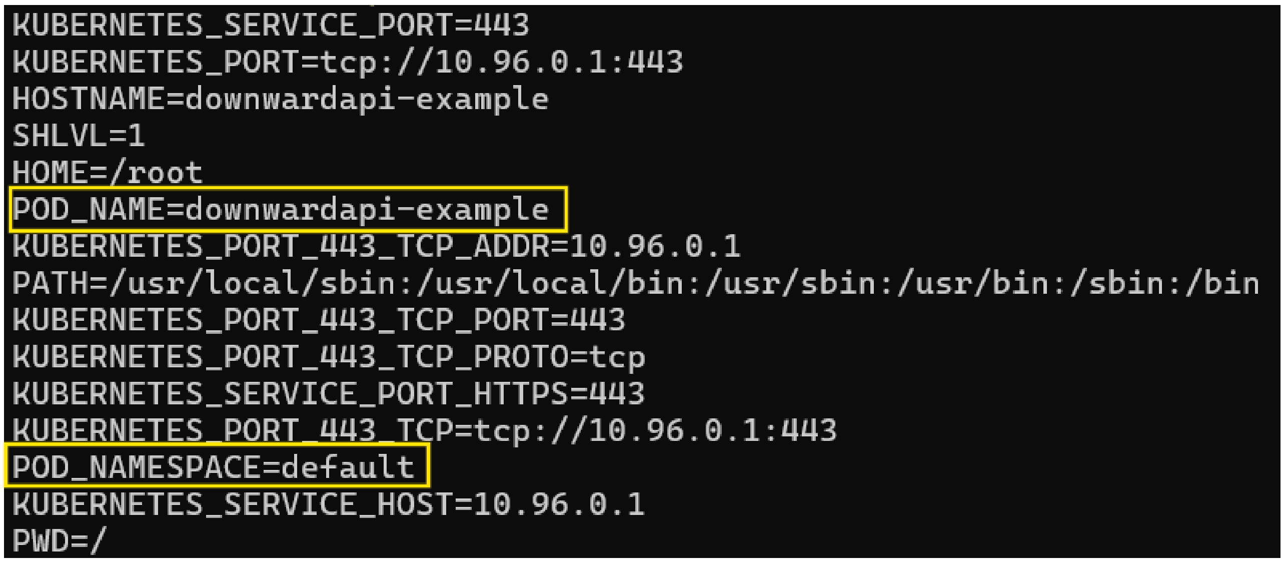
```
kubectl apply -f downwardApi.yaml
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/15)Ephemeral_Storage$ kubectl get pods
NAME                 READY   STATUS    RESTARTS   AGE
downwardapi-example  1/1     Running   0          29s
```

Lets check the logs of this pod

```
kubectl logs downwardapi-example
```



```
KUBERNETES_SERVICE_PORT=443
KUBERNETES_PORT=tcp://10.96.0.1:443
HOSTNAME=downwardapi-example
SHLVL=1
HOME=/root
POD_NAME=downwardapi-example
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
POD_NAMESPACE=default
KUBERNETES_SERVICE_HOST=10.96.0.1
PWD=/
```

Hence Pod's name and namespace details are injected into the container as Environment Variables.

Lets inject Pod's metadata using files as volumes:

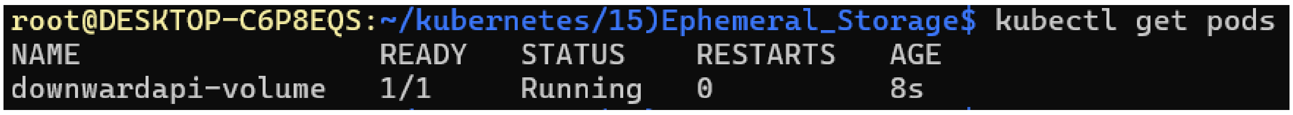
downwardApi-2.yaml:

```
apiVersion: v1
kind: Pod
metadata:
  name: downwardapi-volume
  labels:
    app: good_app
    owner: hr
  annotations:
    version: "good_version"
spec:
  containers:
    - name: metadata-container
```

```
      image: busybox
      command: ["/bin/sh", "-c", "cat
/etc/podinfo/* && sleep 3600"]
      # The container will display the contents
of all files under /etc/podinfo (i.e.,
metadata)
      # and then sleep for an hour, keeping the
pod alive for verification.
      volumeMounts:
        - name: downwardapi-volume
          mountPath: /etc/podinfo
          # Mounts the downward API volume at
/etc/podinfo inside the container.
  volumes:
    - name: downwardapi-volume
      downwardAPI:
        items:
          - path: "labels"
            fieldRef:
              fieldPath: metadata.labels
            # Writes the Pod's labels to a file
named 'labels' under /etc/podinfo.
          - path: "annotations"
            fieldRef:
              fieldPath: metadata.annotations
            # Writes the Pod's annotations to a
file named 'annotations' under /etc/podinfo.
```

Apply;

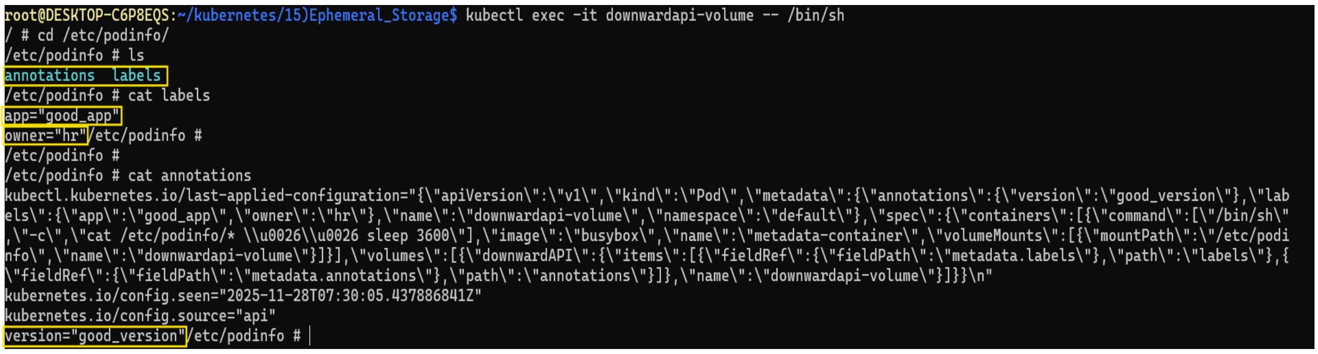
```
kubectl apply -f downwardApi-2.yaml
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/15)Ephemeral_Storage$ kubectl get pods
NAME                 READY   STATUS    RESTARTS   AGE
downwardapi-volume   1/1     Running   0          8s
```

Exec into the container and check /etc/podinfo location:

```
kubectl exec -it downwardapi-volume -- /bin/sh
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/15)Ephemeral_Storage$ kubectl exec -it downwardapi-volume -- /bin/sh
/ # cd /etc/podinfo/
/etc/podinfo # ls
annotations  labels
/etc/podinfo # cat labels
app="good_app"
owner="hr"/etc/podinfo #
/etc/podinfo #
/etc/podinfo # cat annotations
kubectl.kubernetes.io/last-applied-configuration="{\"apiVersion\":\"v1\",\"kind\":\"Pod\",\"metadata\":{\"annotations\":{\"version\":\"good_version\"},\"lab
els\":{\"app\":\"good_app\",\"owner\":\"hr\"},\"name\":\"downwardapi-volume\",\"namespace\":\"default\"},\"spec\":{\"containers\":[{\"command\":[\"/bin/sh\"
,\"-c\",\"cat /etc/podinfo/* \\u0026\\u0026 sleep 3600\"],\"image\":\"busybox\",\"name\":\"metadata-container\",\"volumeMounts\":[{\"mountPath\":\"/etc/podi
nfo\",\"name\":\"downwardapi-volume\"}]}],\"volumes\":[{\"downwardAPI\":{\"items\":[{\"fieldRef\":{\"fieldPath\":\"metadata.labels\"},\"path\":\"labels\"},{
\"fieldRef\":{\"fieldPath\":\"metadata.annotations\"},\"path\":\"annotations\"}]},\"name\":\"downwardapi-volume\"}]}}\n"
kubernetes.io/config.seen="2025-11-28T07:30:05.437886841Z"
kubernetes.io/config.source="api"
version="good_version"/etc/podinfo # 
```

Hence able to inject Pod’s Metadata using files as volumes.