ConfigMaps

Introduction:

A ConfigMap is an API object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume.

A ConfigMap is not designed to hold large chunks of data. The data stored in a ConfigMap cannot exceed 1 MiB. If you need to store settings that are larger than this limit, you may want to consider mounting a volume or use a separate database or file service.

When a ConfigMap currently consumed in a volume is updated, projected keys are eventually updated as well. ConfigMaps consumed as environment variables are not updated automatically and require a pod restart.

Objectives:

1. Consume ConfigMaps using volume
2. Consume ConfigMaps using environment

1. Consume ConfigMaps using volume:

Step1:
First of all, we need to create ConfigMap which will be used via volume. Let’s create a ConfigMaps from t html file.

```
echo "hello from the production department" > prod.html
```

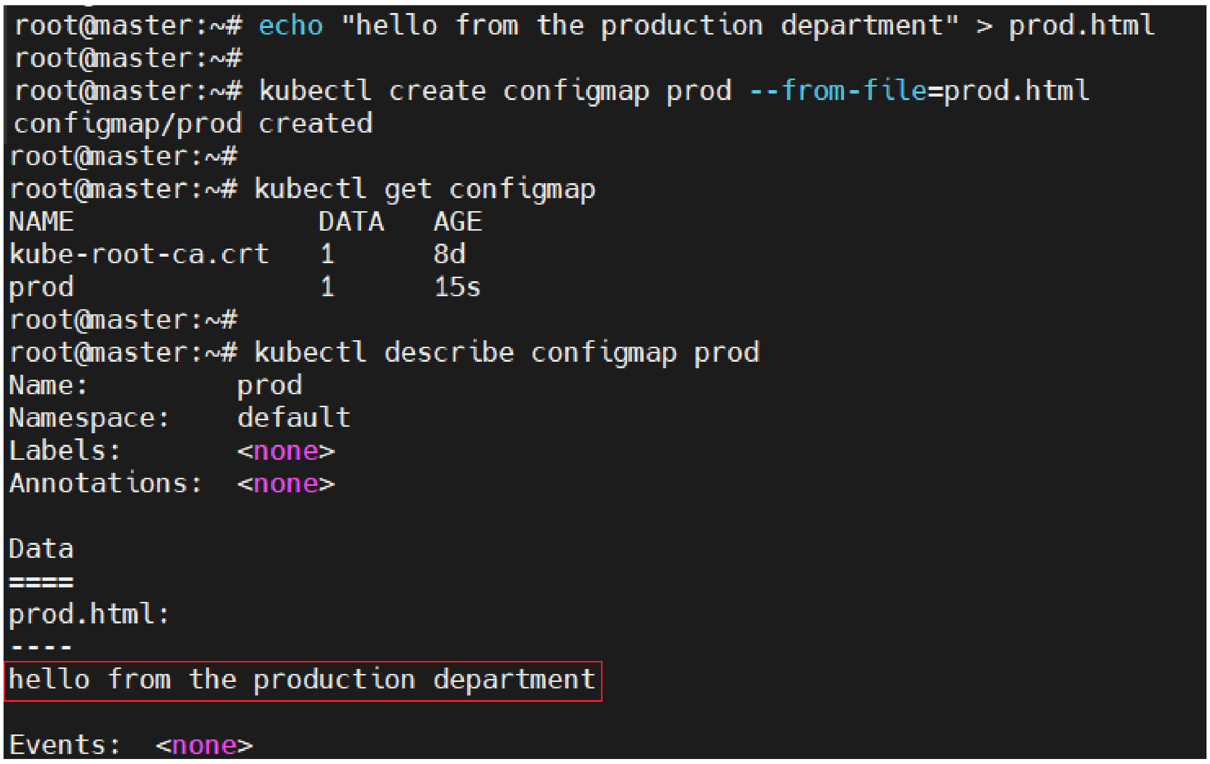
We have created a file named prod.html and now we will be storing this information in a config map.

```
kubectl create configmap prod --from-file=prod.html
```

Our config map is ready, we can get more details using the description command.

```
kubectl get configmap
```

```
kubectl describe configmap prod
```



```
root@master:~# echo "hello from the production department" > prod.html
root@master:~#
root@master:~# kubectl create configmap prod --from-file=prod.html
configmap/prod created
root@master:~#
root@master:~# kubectl get configmap
NAME               DATA   AGE
kube-root-ca.crt   1      8d
prod               1      15s
root@master:~#
root@master:~# kubectl describe configmap prod
Name:         prod
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
prod.html:
----
hello from the production department

Events:  <none>
```

In above output, we can see that the information has been stored inside a config map.

Step 2:

Now we will create a pod and we will be using the above config map via volume.
Let's create a pod using the below definition file.

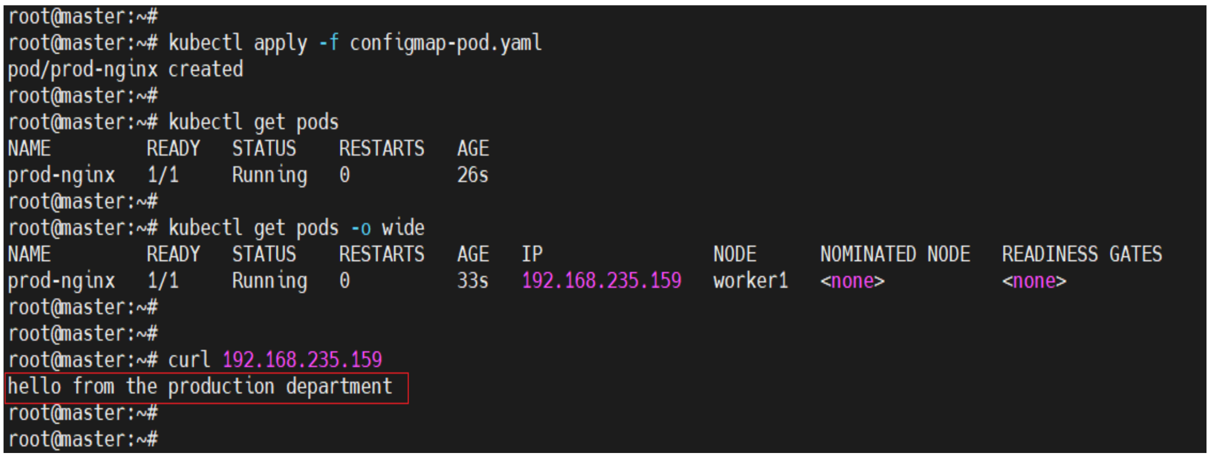
```
vi configmap-pod.yaml
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: prod-nginx
  labels:
    app: prod-nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
    volumeMounts:
    - name: config-volume
      mountPath: /usr/share/nginx/html
  volumes:
  - name: config-volume
    configMap:
      name: prod
      items:
      - key: prod.html
        path: index.html
```

In above pod definition file, we are storing the config map information in container's mountpath = **/usr/share/nginx/html**

Now apply the definition file and see for the result.

```
kubectl apply -f configmap-pod.yaml
```



```
root@master:~#
root@master:~# kubectl apply -f configmap-pod.yaml
pod/prod-nginx created
root@master:~#
root@master:~# kubectl get pods
NAME         READY   STATUS    RESTARTS   AGE
prod-nginx   1/1     Running   0          26s
root@master:~#
root@master:~# kubectl get pods -o wide
NAME         READY   STATUS    RESTARTS   AGE   IP                NODE      NOMINATED NODE   READINESS GATES
prod-nginx   1/1     Running   0          33s   192.168.235.159   worker1   <none>           <none>
root@master:~#
root@master:~#
root@master:~# curl 192.168.235.159
hello from the production department
root@master:~#
root@master:~#
```

From above output, we can see that pod has consumed the information stored in the config map.

2. **Consume ConfigMaps using environment:**

In this exercise, we will be consuming the config map information via environment.

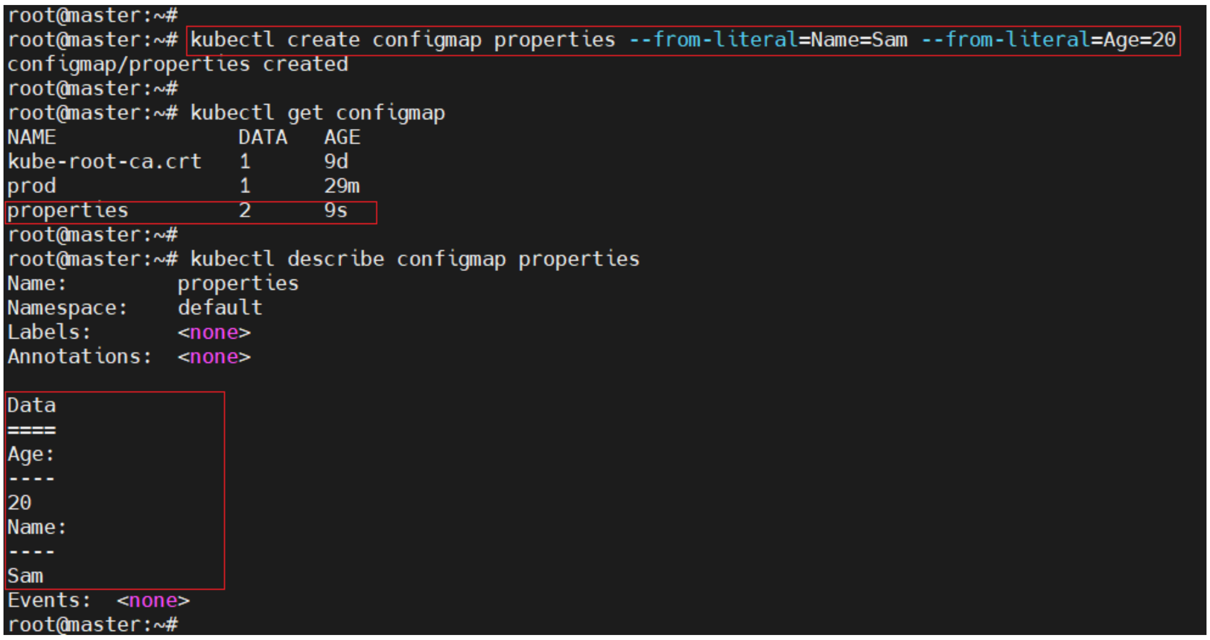
Step 1:

Let’s create a config map first using the below command.

```
kubectl create configmap properties --from-literal=Name=Sam --from-literal=Age=20
```

We have created a config map named properties and stored two information (Name and Age) inside it. Let’s describe the config map to get more details about it.

```
kubectl describe configmap properties
```



```
root@master:~#
root@master:~# kubectl create configmap properties --from-literal=Name=Sam --from-literal=Age=20
configmap/properties created
root@master:~#
root@master:~# kubectl get configmap
NAME               DATA   AGE
kube-root-ca.crt   1      9d
prod               1      29m
properties         2      9s
root@master:~#
root@master:~# kubectl describe configmap properties
Name:         properties
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
Age:
----
20
Name:
----
Sam
Events:  <none>
root@master:~#
```

In above output, we can see that the information has been stored inside the config map.

Step 2:

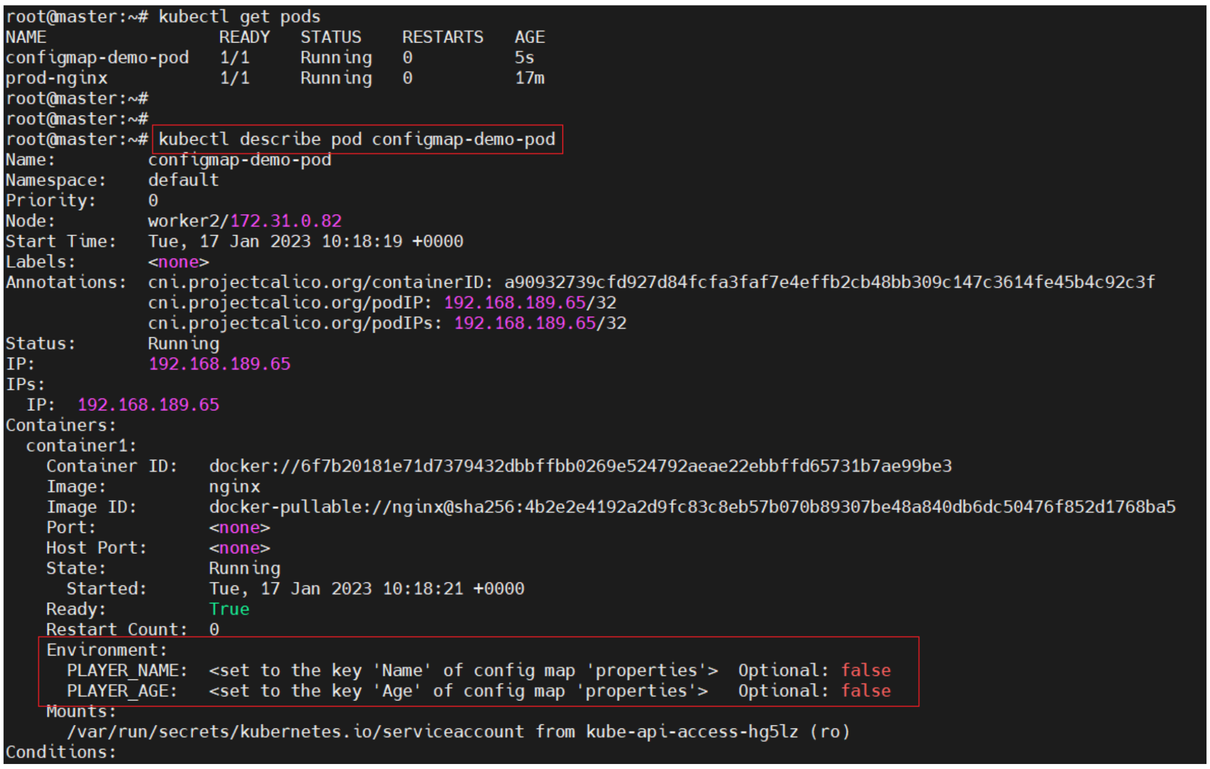
Now let’s create a pod which will be using the config map information as environment variables.

```
vi configmap-env-pod.yaml
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: configmap-demo-pod
spec:
  containers:
  - name: container1
    image: nginx
    env:
    - name: PLAYER_NAME
      valueFrom:
        configMapKeyRef:
          name: properties # The ConfigMap this value comes from.
          key: Name # The key to fetch.
    - name: PLAYER_AGE
      valueFrom:
        configMapKeyRef:
          name: properties
          key: Age
```

Now apply the above definition file and let’s see the result.

```
kubectl apply -f configmap-env-pod.yaml
```

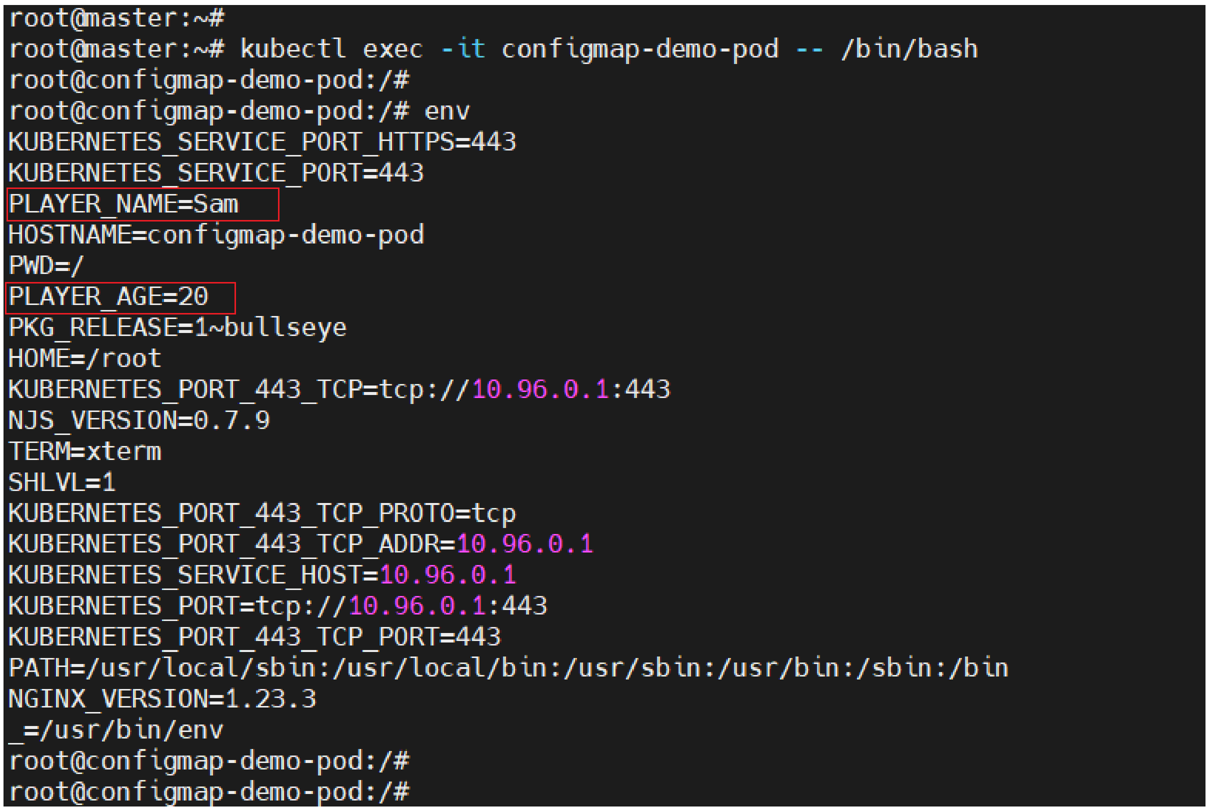


```
root@master:~# kubectl get pods
NAME                 READY   STATUS    RESTARTS   AGE
configmap-demo-pod   1/1     Running   0          5s
prod-nginx           1/1     Running   0          17m
root@master:~#
root@master:~#
root@master:~# kubectl describe pod configmap-demo-pod
Name:         configmap-demo-pod
Namespace:    default
Priority:     0
Node:         worker2/172.31.0.82
Start Time:   Tue, 17 Jan 2023 10:18:19 +0000
Labels:       <none>
Annotations:  cni.projectcalico.org/containerID: a90932739cfd927d84fcfa3faf7e4effb2cb48bb309c147c3614fe45b4c92c3f
              cni.projectcalico.org/podIP: 192.168.189.65/32
              cni.projectcalico.org/podIPs: 192.168.189.65/32
Status:       Running
IP:           192.168.189.65
IPs:
  IP:  192.168.189.65
Containers:
  container1:
    Container ID:   docker://6f7b20181e71d7379432dbbffbb0269e524792aeae22ebbffd65731b7ae99be3
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:4b2e2e4192a2d9fc83c8eb57b070b89307be48a840db6dc50476f852d1768ba5
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Tue, 17 Jan 2023 10:18:21 +0000
    Ready:          True
    Restart Count:  0
    Environment:
      PLAYER_NAME:  <set to the key 'Name' of config map 'properties'>  Optional: false
      PLAYER_AGE:   <set to the key 'Age' of config map 'properties'>   Optional: false
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-hg5lz (ro)
Conditions:
```

In above output, two environment variables have been created and values have been stored from config map file.

Let’s get inside the pod and check for the environment variables.

In below output, we can clearly witness that the environment variables **PLAYER_NAME** and **PLAYER_AGE** has received the information from **Name** and **Age** keys present in config map properties.



```
root@master:~#
root@master:~# kubectl exec -it configmap-demo-pod -- /bin/bash
root@configmap-demo-pod:/#
root@configmap-demo-pod:/# env
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_SERVICE_PORT=443
PLAYER_NAME=Sam
HOSTNAME=configmap-demo-pod
PWD=/
PLAYER_AGE=20
PKG_RELEASE=1~bullseye
HOME=/root
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
NJS_VERSION=0.7.9
TERM=xterm
SHLVL=1
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PORT=443
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
NGINX_VERSION=1.23.3
_=/usr/bin/env
root@configmap-demo-pod:/#
root@configmap-demo-pod:/#
```