**Working with Configmaps and Secrets**

* Define a Command and Arguments for a container.
* Creating ConfigMap
* Using ConfigMap as Environment Variables
* CofigMap from file
* Kubernetes Secrets
* Using Private Repository from Docker Hub using Secret

# **Define a Command and Arguments for a Container**

Below example shows how to define commands and arguments when you run a container in a [Pod](https://kubernetes.io/docs/concepts/workloads/pods/)

apiVersion: v1

kind: Pod

metadata:

  name: command-demo

  labels:

    purpose: demonstrate-command

spec:

  containers:

  - name: command-demo-container

    image: nginx

   command: ["printenv"]

   args: ["HOSTNAME", "KUBERNETES\_PORT", "K1", "K2"]

    env:

    - name: K1

      value: "V1"

    - name: K2

      value: "V2"

**ConfigMap and Environment Variables**

ConfigMaps is a collection of Key/Value pairs to be used for configurating various pods in the complete cluster

The **ConfigMap** resource provides a way to **inject** **configuration** **data** into Pods irrespective of Node on which the pods are running.

ConfigMaps allow you to decouple configuration artifacts from image content to keep containerized applications portable.

Data is read-only – pod can’t alter

Every Environment can have a different ConfigMap YAML.

ConfigMap is an object which when created can be used by other objects on all nodes in the cluster.

Use the **kubectl create configmap** command to create ConfigMaps from **directories, files, or literal values**.

## Example: Define container environment variables using ConfigMap data

**Options1:** Define an environment variable as a key-value pair in a ConfigMap (--from-literal)

kubectl create configmap **mysettings-config1** --from-literal **name**=SONI --from-literal **location=INDIA -n Development**

kubectl create configmap **mysettings-config1** --from-literal **name**=SONI --from-literal **location=USA -n Production**

kubectl get cm mysettings-config1 -o yaml -n Development

apiVersion: v1

kind: ConfigMap

metadata:

  name: mysettings-config1

namespace: **Development**

data:

  name: SONI

  location: INDIA

kubectl apply -f configmap.yaml ~~-n Development~~

apiVersion: v1

kind: ConfigMap

metadata:

  name: mysettings-config1

namespace: **Production**

data:

  name: SONI

  location: USA

kubectl apply -f configmap.yaml ~~-n Production~~

**To View the ConfigMap details**

kubectl **describe** **configmap** mysettings-config1 -n Development

kubectl **describe** **cm** mysettings-config1 -n Development

**Accessing ConfigMap data in a Pod.**

Assign the **mysettings.name=SANDEEP** value defined in the ConfigMap to the **MYSETTINGS.NAME** environment variable in the Pod specification.

**File: test-pod.yaml**

apiVersion: v1

kind: Pod

metadata:

  name: test-pod1

spec:

  containers:

    - name: test-container

      image: nginx

      env:

      - name: MYSETTINGS\_NAME

        valueFrom:

            configMapKeyRef:

              name: mysettings-config1

              key: name

      - name: MYSETTINGS\_LOCATION

        valueFrom:

            configMapKeyRef:

              name: mysettings-config1

              key: location

kubectl apply -f test-pod.yaml -n Development

kubectl exec -it test-pod1 -n Development -- env

Note: The ENVIRONMENT variable MYSETTINGS\_NAME=SANDEEP1 and MYSETTINGS\_LOCATION=INDIA1

kubectl apply -f test-pod.yaml -n Production

kubectl exec -it test-pod1 -n Production -- env

Note: The ENVIRONMENT variable MYSETTINGS\_NAME=SANDEEP2 and MYSETTINGS\_LOCATION=INDIA2

Note: ConfigMaps resides in a specific Namespace. A ConfigMap can only be referenced by pods residing in the same namespace.

**envFrom** can be used to **load ALL ConfigMaps** k/v into environment variables:

**test-pod2.yaml**

apiVersion: v1

kind: Pod

metadata:

  name: test-pod2

spec:

  containers:

    - name: test-container

      image: nginx

      envFrom:

      - configMapRef:

         name: mysettings-config1

kubectl apply -n Development -f test-pod2.yaml

kubectl exec -it test-pod2 -n Development -- printenv

**Note:**

* Change in value of ConfigMap Keys will not be reflected in the already created Pod/Containers if ConfigMap was used for setting Environment Variables.
* Once the container is created, environment variables cannot be changed unless explicity set using OS commands or You can obviously delete and recreate the Deployment.

**Option2: ConfigMaps can be created from Env File (defining Key.Value Pairs)**

**mysettings.env**

name=sandeep

location=India

**Command: --from-env-file**

kubectl create configmap mysettings-config2 **--from-env-file**=mysettings.env

kubectl get configmap mysettings-config2 -o yaml

**Note that the filename is not included as a Key.**

**Option3: ConfigMaps can be created from File:**

Key is a filename, value is the file content (can be JSON, XML, CSV, keys/values, etc…). The application in the container will have to parse the content of the file.

**demo1.txt**

This is content of the file1

**demo2.txt**

This is content of the file2

**Command: --from-file**

kubectl create configmap mysettings-config3 **--from-file**=demo1.txt **--from-file**=demo2.txt

kubectl get configmap mysettings-config3 -o yaml

Note: The Key=filename (mysettings.json) and Value is the entire content of the file.

kubectl create configmap mysettings-config4 **--from-file**=**d1**=demo1.txt **--from-file**=**d2**=demo2.txt

kubectl get configmap mysettings-config4 -o yaml

In the above command Key=d1 and d2 and not demo1.txt and demo2.txt

**Note: If --from-file is set to directory, for every file in the directory a key is added to configmap with file content as its value.**

**Note: ConfigMaps can be accessed from a Pod using ConfigMap Volumes. We will cover this later in volumes chapter.**

**Kubernetes Secrets**

* How do you store sensitive information? Should you include it in Docker image? How about in a pod spec? NEVER?
* Kubernetes Secrets let you store and manage sensitive information that your pods can access at runtime. Think passwords, OAuth tokens, and ssh keys.
* When using kubectl get, you wont see the contents of a secret. But they are accessible to those with access directly to the cluster.
* Its best to have secrets managed by a limited set of people who know how to keep them safe. And don’t just check them into source control alongside your resources.

kubectl create **secret** generic dbsecrets --**from-literal** user=admin **--from-literal** password=tiger1234

kubectl describe secret dbsecret #Note that we can’t see the values of keys

kubectl get secret dbsecret -o yaml

**Secret.yaml**

apiVersion: v1

kind: Secret

metadata:

  name: dbsecrets

type: Opaque

data:

  user: YWRtaW4= #base64 value of admin

  password: dGlnZXIxMjM0 #base64 value of tiger1234

**Note: In YAML value of secret keys must be base64 encoded.**

**Referencing a secret:**

**pod.yaml**

apiVersion: v1

kind: Pod

metadata:

  name: test-pod

spec:

  containers:

    - name: test-container

      image: nginx

      env:

        - name: USERENV

          valueFrom:

**secretKeyRef**:

              name: dbsecrets

              key: user

        - name: MYPASSWORDENV

          valueFrom:

**secretKeyRef**:

              name: dbsecrets

              key: password

**Execute the following commands:**

1. Kubectl apply secret.yaml
2. Kubectl apply pod.yaml
3. Kubectl get secrets dbsecrets -o yaml

**Secrets from file:**

**Credentials.txt**

username=admin

password=tiger1234

**Command:**

kubectl create secret generic mysecrets **--from-env-file** credentials.txt

**OR**

kubectl create secret generic mysecrets **--from-file**=ssh-privatekey=~/.ssh/id\_rsa **--from-file**=ssh-publickey=~/.ssh/id\_rsa.pub

**Using the Private Repository from Docker Hub using Secret**

**Step0: Push an image to Registry**

docker login

docker tag nginx sandeepsoni/mynginx

docker push sandeepsoni/mynginx

Go to docker.io and make the image repository as PRIVATE.

**Step1: Create a Secret:**

If DockerHub is used:

**kubectl create secret** **docker-registry** **mydockersecret** **--docker-username**="sandeepsoni" **--docker-password** "XXXXXXXX" **--docker-server**=docker.io

**Step2: Update the YAML file**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: myhelloapp-deployment

spec:

  replicas: 2

  selector:

    matchLabels:

      app: myhelloapp

  template:

    metadata:

      labels:

        app: myhelloapp

    spec:

      containers:

      - name: myhelloapp-container

        image: sandeepsoni/mynginx

        ports:

        - containerPort: 80

        imagePullPolicy: Always

      imagePullSecrets:

      - name: mydockersecret

**Step3: Deploy**

kubectl apply -f deployment.yaml

**Secrets YAML:**

**kubectl get secrets mydockersecret -o yaml > secrets.yaml**

apiVersion: v1

data:

  .dockerconfigjson: eyJhdXRocyI6eyJodHRwczovL2luZGV4Lm...RPT0ifX19

kind: Secret

metadata:

  creationTimestamp: "2020-07-01T12:38:09Z"

  name: mysecret

  namespace: default

  resourceVersion: "369988"

  selfLink: /api/v1/namespaces/default/secrets/docker-registry-secret

  uid: cf146992-8896-48e7-b9df-80a9051036b2

type: kubernetes.io/dockerconfigjson

The value of the .dockerconfigjson field is a base64 representation of your Docker credentials

You can visit <https://www.base64decode.org/> and decode the Base64 value.

**More about Pulling an Image from Private Registry:**

<https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/>