<https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/>

Here's an example of a yaml descriptor for a ConfigMap containing some data:

apiVersion: v1

kind: ConfigMap

metadata:

name: my-config-map

data:

myKey: myValue

anotherKey: anotherValue

Passing ConfigMap data to a container as an environment variable looks like this:

apiVersion: v1

kind: Pod

metadata:

name: my-configmap-pod

spec:

containers:

- name: myapp-container

image: busybox

command: ['sh', '-c', "echo $(MY\_VAR) && sleep 3600"]

env:

- name: MY\_VAR

valueFrom:

configMapKeyRef:

name: my-config-map

key: myKey

It's also possible to pass ConfigMap data to containers, in the form of file using a mounted volume, like so:

apiVersion: v1

kind: Pod

metadata:

name: my-configmap-volume-pod

spec:

containers:

- name: myapp-container

image: busybox

command: ['sh', '-c', "echo $(cat /etc/config/myKey) && sleep 3600"]

volumeMounts:

- name: config-volume

mountPath: /etc/config

volumes:

- name: config-volume

configMap:

name: my-config-map

In the lesson, we'll also use the following commands to explore how the ConfigMap data interacts with pods and containers:

kubectl logs my-configmap-pod

kubectl logs my-configmap-volume-pod

kubectl exec my-configmap-volume-pod -- ls /etc/config

kubectl exec my-configmap-volume-pod -- cat /etc/config/myKey

################## section 2 ####################

**securityContext**

Occasionally, it's necessary to customize how containers interact with the underlying security mechanisms present on the operating systems of Kubernetes nodes. The securityContext attribute in a pod specification allows for making these customizations. In this lesson, we will briefly discuss what the securityContext is, and demonstrate how to use it to implement some common functionality.

Relevant Documentation

* <https://kubernetes.io/docs/tasks/configure-pod-container/security-context/>

First, create some users, groups, and files on both worker nodes which we can use for testing.

sudo useradd -u 2000 container-user-0

sudo groupadd -g 3000 container-group-0

sudo useradd -u 2001 container-user-1

sudo groupadd -g 3001 container-group-1

sudo mkdir -p /etc/message/

echo "Hello, World!" | sudo tee -a /etc/message/message.txt

sudo chown 2000:3000 /etc/message/message.txt

sudo chmod 640 /etc/message/message.txt

On the controller, create a pod to read the message.txt file and print the message to the log.

vi my-securitycontext-pod.yml

Content of the YAML File

apiVersion: v1

kind: Pod

metadata:

name: my-securitycontext-pod

spec:

containers:

- name: myapp-container

image: busybox

command: ['sh', '-c', "cat /message/message.txt && sleep 3600"]

volumeMounts:

- name: message-volume

mountPath: /message

volumes:

- name: message-volume

hostPath:

path: /etc/message

Check the pod's log to see the message from the file:

kubectl logs my-securitycontext-pod

Delete the pod and re-create it, this time with a securityContext set to use a user and group that do not have access to the file.

kubectl delete pod my-securitycontext-pod --now

apiVersion: v1

kind: Pod

metadata:

name: my-securitycontext-pod

spec:

securityContext:

runAsUser: 2001

fsGroup: 3001

containers:

- name: myapp-container

image: busybox

command: ['sh', '-c', "cat /message/message.txt && sleep 3600"]

volumeMounts:

- name: message-volume

mountPath: /message

volumes:

- name: message-volume

hostPath:

path: /etc/message

Check the log again. You should see a "permission denied" message.

kubectl logs my-securitycontext-pod

Delete the pod and re-create it again, this time with a user and group that are able to access the file.

kubectl delete pod my-securitycontext-pod --now

apiVersion: v1

kind: Pod

metadata:

name: my-securitycontext-pod

spec:

securityContext:

runAsUser: 2000

fsGroup: 3000

containers:

- name: myapp-container

image: busybox

command: ['sh', '-c', "cat /message/message.txt && sleep 3600"]

volumeMounts:

- name: message-volume

mountPath: /message

volumes:

- name: message-volume

hostPath:

path: /etc/message

Check the log once more. You should see the message from the file.

kubectl logs my-securitycontext-pod

################## section 3 ###########################

## **Lecture: Resource Requirements**

Kubernetes is a powerful tool for managing and utilizing available resources to run containers. Resource requests and limits provide a great deal of control over how resources will be allocated. In this lesson, we will talk about what resource requests and limits do, and also demonstrate how to set resource requests and limits for a container.

### Relevant Documentation

* <https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/#resource-requests-and-limits-of-pod-and-container>

Specify resource requests and resource limits in the container spec like this:

apiVersion: v1

kind: Pod

metadata:

name: my-resource-pod

spec:

containers:

- name: myapp-container

image: busybox

command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']

resources:

requests:

memory: "64Mi"

cpu: "250m"

limits:

memory: "128Mi"

cpu: "500m"

################### section 4 ########################

## **Lecture: Secrets**

One of the challenges in managing a complex application infrastructure is ensuring that sensitive data remains secure. It is always important to store sensitive data, such as tokens, passwords, and keys, in a secure, encrypted form. In this lesson, we will talk about Kubernetes secrets, a way of securely storing data and providing it to containers. We will also walk through the process of creating a simple secret, and passing the sensitive data to a container as an environment variable.

### Relevant Documentation

* <https://kubernetes.io/docs/concepts/configuration/secret/>

Create a secret using a yaml definition like this. It is a good idea to delete the yaml file containing the sensitive data after the secret object has been created in the cluster.

apiVersion: v1

kind: Secret

metadata:

name: my-secret

stringData:

myKey: myPassword

Once a secret is created, pass the sensitive data to containers as an environment variable:

apiVersion: v1

kind: Pod

metadata:

name: my-secret-pod

spec:

containers:

- name: myapp-container

image: busybox

command: ['sh', '-c', "echo Hello, Kubernetes! && sleep 3600"]

env:

- name: MY\_PASSWORD

valueFrom:

secretKeyRef:

name: my-secret

key: myKey