Lab 7: Pod and Container level security using Context

A security context defines permission to access an object, on pod and container level for security based on user ID (UID) and group ID (GID).

In this lab, we are going to create main application container which writes the current date to log file every five seconds and we are going to create sidecar container as nginx server which serves that log file.

Task 1: Set the security context for a pod

1. Create file sc-pod.yml to apply security context on pod level. These security settings will be applied on all the containers in the pod

# vi sc-pod.yml

:

apiVersion: v1

kind: Pod

metadata:

name: sc-pod

spec:

securityContext:

runAsUser: 1000

runAsGroup: 3000

fsGroup: 2000

volumes:

- name: sc-vol

emptyDir: {}

containers:

- name: sc-ctr

image: busybox

command: [ "sh", "-c", "sleep 1h" ]

volumeMounts:

- name: sc-vol

mountPath: /data/demo

securityContext:

allowPrivilegeEscalation: false

~

~

~

~

~

~

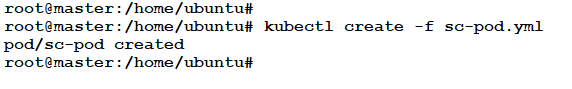
~

The yml file should look similar to the below image:



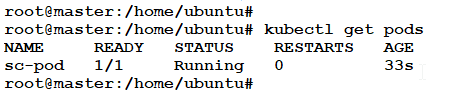
1. Create the pod

# kubectl create -f sc-pod.yml



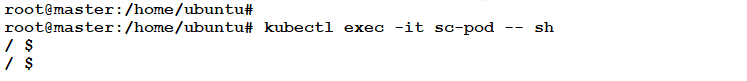
1. Verify that the pod’s containers are running

# kubectl get pods



1. Now SSH into it

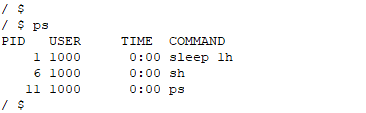
# kubectl exec -it pod-sc -- sh



When we go into the container, usually we will be the root user. We can identify that by looking into the shell prompt, “#”. But here the shell prompt is “$”, showing that we were not into the container as a root user.

1. List the running processes in the container. The output will show that all the processes are running as user 1000, i.e. **runAsUser**

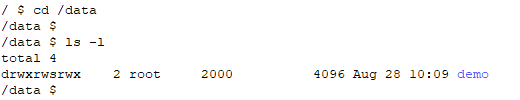
$ ps



1. Navigate to /data and list the directories

$ cd /data

$ ls -l



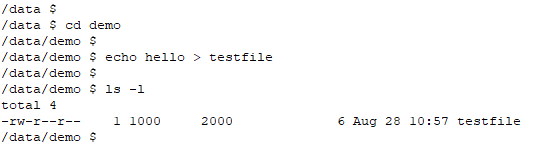
The output shows the directory, demo with the group id 2000 which is the value of **fsGroup.** The fsGroup owns the pod’s volumes, hence the directory belongs to the fsGroup

1. Go into the demo directory and create a file. Then list the directory and notice the group ID that the file belongs to is 2000, i.e. **fsGroup**

$ cd demo

$ echo hello > testfile

$ ls -l



1. Run the following command

$ id



Exit the shell

Task 2: Set the security context for a container

The security settings that we specify for a container apply only to the container, and they override settings made at the Pod level when there is overlap. Containers setting do not affect the pod’s volumes

1. Create file sc-ctr.yml to apply security context on container level

# vi sc-ctr.yml

apiVersion: v1

kind: Pod

metadata:

name: sc-pod2

spec:

securityContext:

runAsUser: 1000

containers:

- name: sc-ctr2

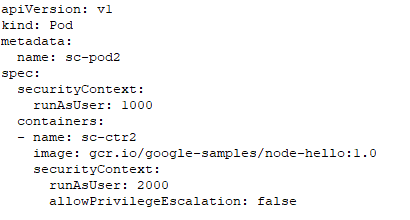
image: gcr.io/google-samples/node-hello:1.0

securityContext:

runAsUser: 2000

allowPrivilegeEscalation: false

The yml file should look similar to the below image:



1. Create the pod

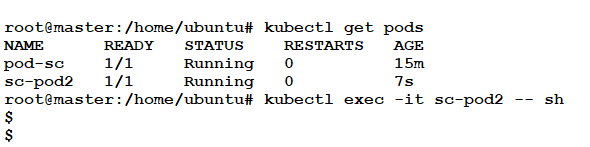
# kubectl create -f sc-ctr.yml



1. Verify that the pod’s containers are running. Then SSH into it

# kubectl get pods

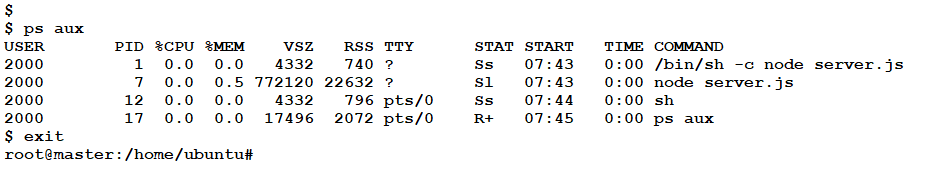
# kubectl exec -it sc-pod2 -- sh



1. List the running processes and **exit**

$ ps aux

$ exit



The output will show that all the processes are running as user 2000, i.e. **runAsUser.** Notice that the value we specified for the pod was 1000. Hence, they override settings made at the pod level.

1. Delete the resources that we created in this lab.

# kubectl delete -f sc-pod.yml

# kubectl delete -f sc-ctr.yml

