Cluster Security, Working With ConfigMap & Limiting Resources With Resource Quota

Contents

1 INTRODUCTION

Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within your organization.
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 allows** you to decouple environment-specific configuration from your container images, so that your applications are easily portable.

Resource quotas

When several users or teams share a cluster with a fixed number of nodes, there is a concern that one team could use more than its fair share of resources.
Resource quotas are a tool for administrators to address this concern.
A resource quota, defined by a ResourceQuota object, provides constraints that limit aggregate resource consumption per namespace. It can limit the quantity of objects that can be created in a namespace by type, as well as the total amount of compute resources that may be consumed by resources in that project.

This guide Covers:

- Authentication and Authorisation using RBAC
- Configuring Network Policies for Applications
- Configuring Container Security Context
- Working With ConfigMap
- Limiting Resources With Resource Quota

2 DOCUMENTATION

2.1 Kubernetes Documentation

1. Using RBAC Authorization
   https://kubernetes.io/docs/reference/access-authn-authz/rbac/
2. Authorization Overview
   https://kubernetes.io/docs/reference/access-authn-authz/authorization/
3. Network Policies
   https://kubernetes.io/docs/concepts/services-networking/network-policies/
4. Configure a Security Context for a Pod or Container
   https://kubernetes.io/docs/tasks/configure-pod-container/security-context/
5. Resource Quotas
   https://kubernetes.io/docs/concepts/policy/resource-quotas/
6. Configure Memory and CPU Quotas for a Namespace
   https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/quota-memory-cpu-namespace/
7. ConfigMaps
   https://kubernetes.io/docs/concepts/configuration/configmap/

2.2 Linux Commands and VIM Commands

1. Basic Linux Commands

   https://maker.pro/linux/tutorial/basic-linux-commands-for-beginners

   https://www.hostinger.in/tutorials/linux-commands
2. Basic VIM Commands

   https://coderwall.com/p/adv71w/basic-vim-commands-for-getting-started
3. Popular VIM Commands

   https://www.keycdn.com/blog/vim-commands

3 PREVIOUS GUIDES

Ensure that you have completed following activity guides:

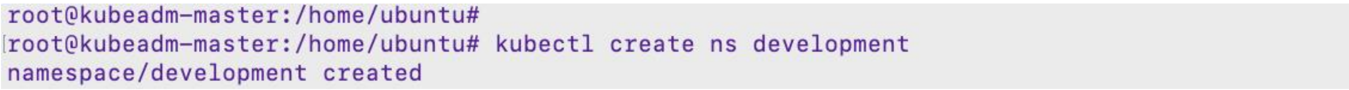
- **Note**: *Follow Activity Guide* ***AG_Bootstrap_Kubernetes_Cluster_Using_Kubeadm_Guide_ed***** from portal*
- **Note**: *Follow Activity Guide* ***AG_Deploy_App_On_Pod_&_Basic_Networking_ed***** from portal*
- **Note**: *Follow Activity Guide* ***AG_Deploying_Scalable_and_Configuring_Autoscaling_For_Stateless_Application_ed***** from portal*
- **Note**: *Follow Activity Guide* ***AG_Configuring_NFS_Storage_Persistence_Volume_ed***** from portal*
- **Note**: *Follow Activity Guide* ***AG_Constraint_Pod_and_Node_Selector_Node_Affinity_&_Anti_Affinity_ed***** from portal*
- **Note**: *Follow Activity Guide* ***AG_Cluster_Node_Maintenance_Debugging_Application_Failure_Troubleshooting_Cluster_ed***** from portal*

4 AUTHENTICATION AND AUTHORISATION USING RBAC

4.1 Creating Namespace, User & User Credentials

1. Create a new namespace named development

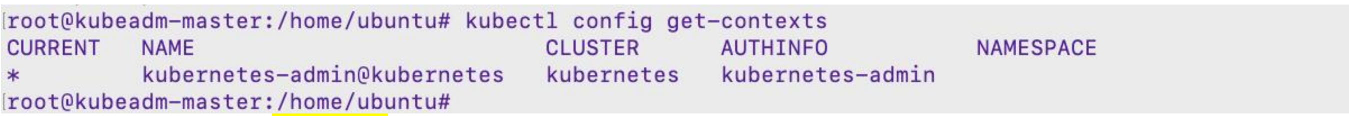
   $ kubectl create ns development



```
root@kubeadm-master:/home/ubuntu#
root@kubeadm-master:/home/ubuntu# kubectl create ns development
namespace/development created
```

2. View the current clusters and context available. The context allows us to configure the cluster to use, namespace and user for kubectl commands in an easy and consistent manner.

   $ kubectl config get-contexts

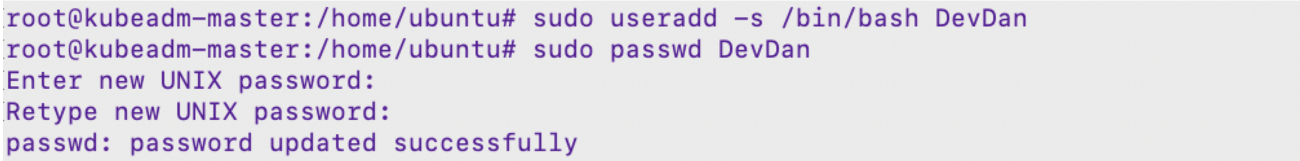


```
root@kubeadm-master:/home/ubuntu# kubectl config get-contexts
CURRENT   NAME                          CLUSTER      AUTHINFO           NAMESPACE
*         kubernetes-admin@kubernetes   kubernetes   kubernetes-admin
root@kubeadm-master:/home/ubuntu#
```

3. Create a new user *DevDan* and assign a password to him

   $ sudo useradd -s /bin/bash DevDan

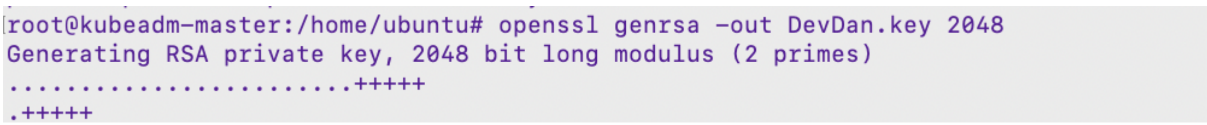
   $ sudo passwd DevDan



```
root@kubeadm-master:/home/ubuntu# sudo useradd -s /bin/bash DevDan
root@kubeadm-master:/home/ubuntu# sudo passwd DevDan
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

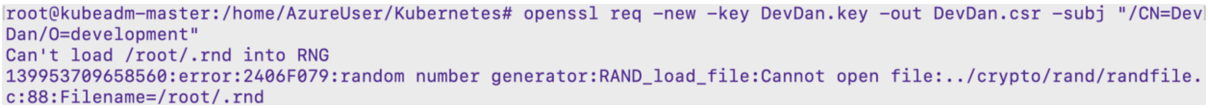
4. Generate a private key for DevDan and Certificate Signing Request (CSR) for DevDan

   $ openssl genrsa -out DevDan.key 2048



```
root@kubeadm-master:/home/ubuntu# openssl genrsa -out DevDan.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.......................+++++
.+++++
```

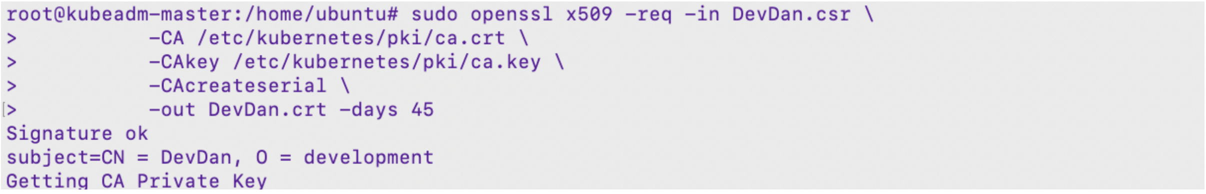
   $ openssl req -new -key DevDan.key \
   -out DevDan.csr -subj "/CN=DevDan/O=development"



```
root@kubeadm-master:/home/AzureUser/Kubernetes# openssl req -new -key DevDan.key -out DevDan.csr -subj "/CN=Dev
Dan/O=development"
Can't load /root/.rnd into RNG
139953709658560:error:2406F079:random number generator:RAND_load_file:Cannot open file:../crypto/rand/randfile.
c:88:Filename=/root/.rnd
```

5. Generate a self-signed certificate. Use the CA keys for the Kubernetes cluster and set the certificate expiration.

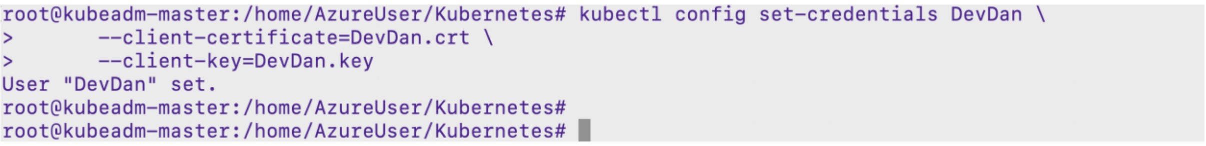
```
$ sudo openssl x509 -req -in DevDan.csr \

        -CA /etc/kubernetes/pki/ca.crt \

        -CAkey /etc/kubernetes/pki/ca.key \

        -CAcreateserial \

        -out DevDan.crt -days 45
```



```
root@kubeadm-master:/home/ubuntu# sudo openssl x509 -req -in DevDan.csr \
>           -CA /etc/kubernetes/pki/ca.crt \
>           -CAkey /etc/kubernetes/pki/ca.key \
>           -CAcreateserial \
>           -out DevDan.crt -days 45
Signature ok
subject=CN = DevDan, O = development
Getting CA Private Key
```

6. Update the access config file to reference the new key and certificate.

```
$ kubectl config set-credentials DevDan \

      --client-certificate=DevDan.crt \

      --client-key=DevDan.key
```

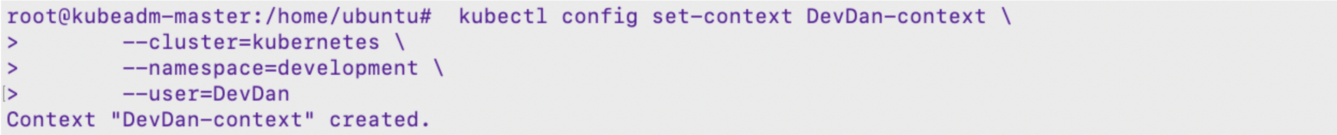


```
root@kubeadm-master:/home/AzureUser/Kubernetes# kubectl config set-credentials DevDan \
>       --client-certificate=DevDan.crt \
>       --client-key=DevDan.key
User "DevDan" set.
root@kubeadm-master:/home/AzureUser/Kubernetes#
root@kubeadm-master:/home/AzureUser/Kubernetes# 
```

4.2 Setting up Context for New User

1. Create context for DevDan user in the cluster and namespace

```
$ kubectl config set-context DevDan-context \

      --cluster=kubernetes \

      --namespace=development \

      --user=DevDan
```

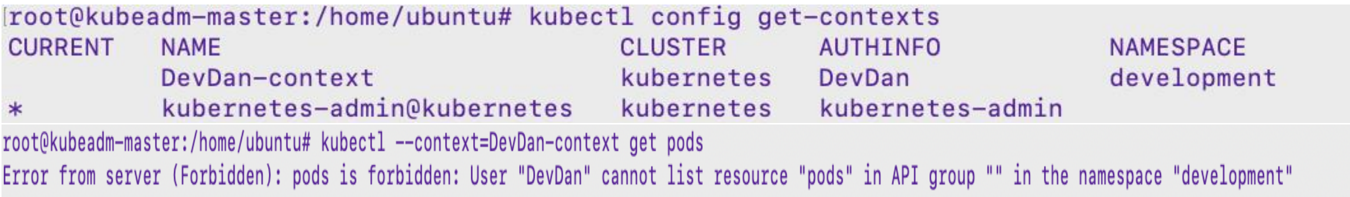


```
root@kubeadm-master:/home/ubuntu#  kubectl config set-context DevDan-context \
>        --cluster=kubernetes \
>        --namespace=development \
>        --user=DevDan
Context "DevDan-context" created.
```

2. Verify the context has been properly set. Attempt to view the Pods inside the DevDan-context. Be aware you will get an error.

   $ kubectl config get-contexts

   $ kubectl --context=DevDan-context get pods

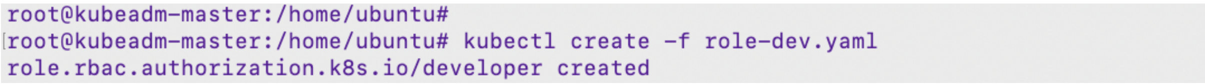


```
root@kubeadm-master:/home/ubuntu# kubectl config get-contexts
CURRENT   NAME                          CLUSTER      AUTHINFO           NAMESPACE
          DevDan-context                kubernetes   DevDan             development
*         kubernetes-admin@kubernetes   kubernetes   kubernetes-admin
root@kubeadm-master:/home/ubuntu# kubectl --context=DevDan-context get pods
Error from server (Forbidden): pods is forbidden: User "DevDan" cannot list resource "pods" in API group "" in the namespace "development"
```

4.3 Create RBAC Role and Rolebinding

1. Create a YAML file to associate RBAC rights to a particular namespace and Role. Create the object. Check white space and for typos if you encounter errors.

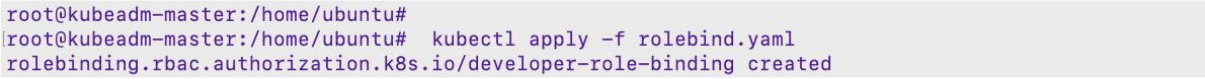
   $ kubectl create -f role-dev.yaml



```
root@kubeadm-master:/home/ubuntu#
root@kubeadm-master:/home/ubuntu# kubectl create -f role-dev.yaml
role.rbac.authorization.k8s.io/developer created
```

2. Then we create will a RoleBinding to associate the Role we just created with a user. Create the object from the rolebind.yaml file.

   $ kubectl create -f rolebind.yaml



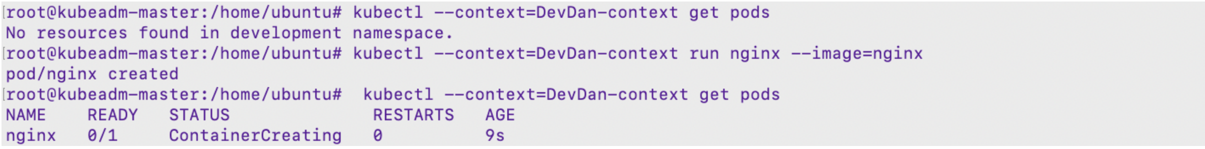
```
root@kubeadm-master:/home/ubuntu#
root@kubeadm-master:/home/ubuntu#  kubectl apply -f rolebind.yaml
rolebinding.rbac.authorization.k8s.io/developer-role-binding created
```

3. Now let’s try list pods and then creating a pod using DevDan-context

   $ kubectl --context=DevDan-context get pods

   $ kubectl --context=DevDan-context run nginx --image=nginx

   $ kubectl --context=DevDan-context get pods



```
root@kubeadm-master:/home/ubuntu# kubectl --context=DevDan-context get pods
No resources found in development namespace.
root@kubeadm-master:/home/ubuntu# kubectl --context=DevDan-context run nginx --image=nginx
pod/nginx created
root@kubeadm-master:/home/ubuntu#  kubectl --context=DevDan-context get pods
NAME    READY   STATUS              RESTARTS   AGE
nginx   0/1     ContainerCreating   0          9s
```

4.4 Clean-up Resources Created this Section

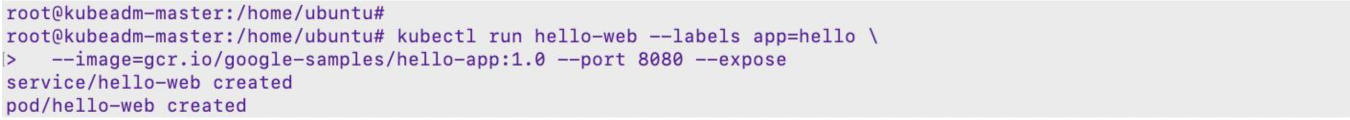
```
$ kubectl delete pod nginx -n development

$ kubectl delete -f rolebind.yaml

$ kubectl delete -f role-dev.yaml

$ kubectl delete ns development
```

5 CONFIGURING NETWORK POLICIES FOR APPLICATIONS

5.1 Restrict Incoming Traffic Pods

1. Run a simple web server application with label app=hello and expose it internally in the cluster

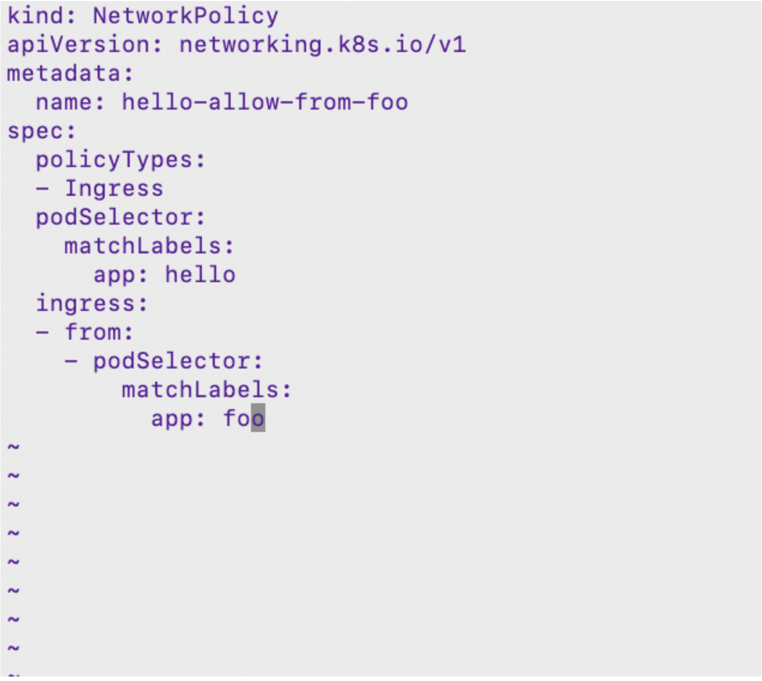
   $ kubectl run hello-web --labels app=hello --image=gcr.io/google-samples/hello-app:1.0 --port 8080 --expose



```
root@kubeadm-master:/home/ubuntu#
root@kubeadm-master:/home/ubuntu# kubectl run hello-web --labels app=hello \
[>   --image=gcr.io/google-samples/hello-app:1.0 --port 8080 --expose
service/hello-web created
pod/hello-web created
```

2. All inbound traffic by default is allowed. So let’s configure a NetworkPolicy to allow traffic to hello-web pods from only pods with label app=foo. All other incoming traffic will be blocked.

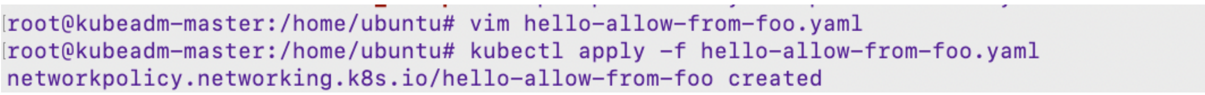
   $ vi hello-allow-from-foo.yaml



```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: hello-allow-from-foo
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: hello
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: foo
~
~
~
~
~
~
~
~
```

3. Apply this policy to the cluster with kubectl command

   $ kubectl apply -f hello-allow-from-foo.yaml

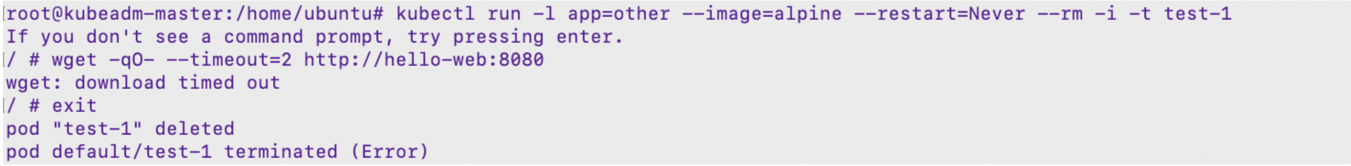


```
[root@kubeadm-master:/home/ubuntu# vim hello-allow-from-foo.yaml
[root@kubeadm-master:/home/ubuntu# kubectl apply -f hello-allow-from-foo.yaml
networkpolicy.networking.k8s.io/hello-allow-from-foo created
```

5.2 Validating Network Policy

1. Run a temporary Pod with a different label *(app=other)* and get a shell inside the Pod. Observe that the traffic is **not allowed** and therefore the request times out

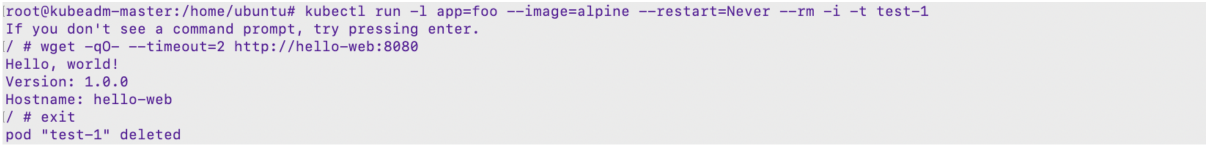
```
$ kubectl run -l app=other --image=alpine --restart=Never --rm -i -t test-1

# wget -qO- --timeout=2 http://hello-web:8080

# exit
```



```
root@kubeadm-master:/home/ubuntu# kubectl run -l app=other --image=alpine --restart=Never --rm -i -t test-1
If you don't see a command prompt, try pressing enter.
/ # wget -qO- --timeout=2 http://hello-web:8080
wget: download timed out
/ # exit
pod "test-1" deleted
pod default/test-1 terminated (Error)
```

2. Run a temporary Pod with a different label (app=foo) and get a shell inside the Pod. Observe that the traffic is **allowed**

```
$ kubectl run -l app=foo --image=alpine --restart=Never --rm -i -t test-1

# wget -qO- --timeout=2 http://hello-web:8080

# exit
```



```
root@kubeadm-master:/home/ubuntu# kubectl run -l app=foo --image=alpine --restart=Never --rm -i -t test-1
If you don't see a command prompt, try pressing enter.
/ # wget -qO- --timeout=2 http://hello-web:8080
Hello, world!
Version: 1.0.0
Hostname: hello-web
/ # exit
pod "test-1" deleted
```

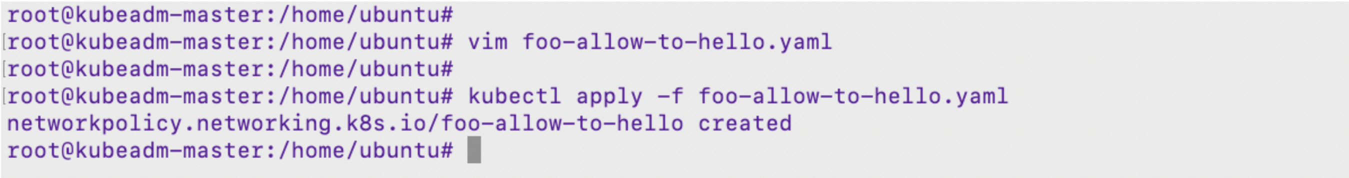
5.3 Restrict Outgoing Traffic from Pods

1. All outbound traffic by default is allowed. So let's configure a NetworkPolicy to allow traffic only from pods labelled as app=foo to send traffic only to pods with label app=hello. All other outgoing traffic from app=foo will be blocked.

```
$ vi foo-allow-to-hello.yaml
```

2. Apply this policy to the cluster with kubectl command

```
$ kubectl apply -f foo-allow-to-hello.yaml
```

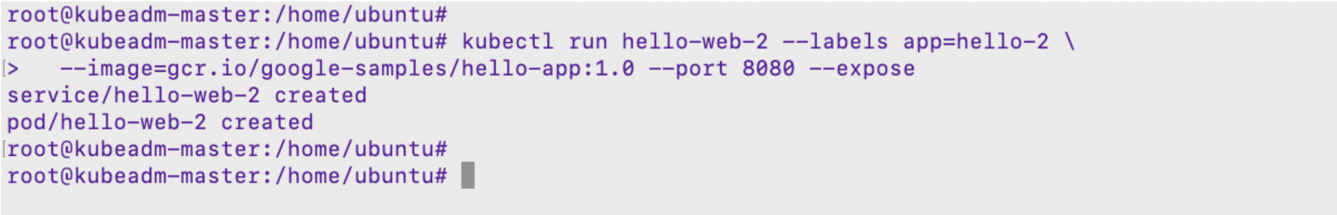


```
root@kubeadm-master:/home/ubuntu#
root@kubeadm-master:/home/ubuntu# vim foo-allow-to-hello.yaml
root@kubeadm-master:/home/ubuntu#
root@kubeadm-master:/home/ubuntu# kubectl apply -f foo-allow-to-hello.yaml
networkpolicy.networking.k8s.io/foo-allow-to-hello created
root@kubeadm-master:/home/ubuntu#
```

5.4 Validating Network Policy

1. Run a temporary Pod with a different label (app=hello-2)

```
$ kubectl run hello-web-2 --labels app=hello-2 \

  --image=gcr.io/google-samples/hello-app:1.0 --port 8080 --expose
```



```
root@kubeadm-master:/home/ubuntu#
root@kubeadm-master:/home/ubuntu# kubectl run hello-web-2 --labels app=hello-2 \
>   --image=gcr.io/google-samples/hello-app:1.0 --port 8080 --expose
service/hello-web-2 created
pod/hello-web-2 created
root@kubeadm-master:/home/ubuntu#
root@kubeadm-master:/home/ubuntu#
```

2. Next, run a temporary Pod with app=foo label and get a shell prompt inside the container:

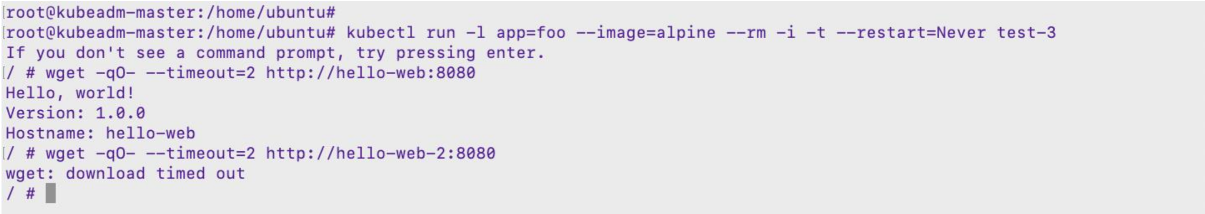
```
$kubectl run -l app=foo --image=alpine --rm -i -t --restart=Never test-3
```

3. Validate that the Pod can establish connections to hello-web:8080:

```
# wget -qO- --timeout=2 http://hello-web:8080
```

4. Validate that the Pod **cannot** establish connections to hello-web-2:8080:

```
# wget -qO- --timeout=2 http://hello-web-2:8080

# exit
```



```
root@kubeadm-master:/home/ubuntu#
root@kubeadm-master:/home/ubuntu# kubectl run -l app=foo --image=alpine --rm -i -t --restart=Never test-3
If you don't see a command prompt, try pressing enter.
/ # wget -qO- --timeout=2 http://hello-web:8080
Hello, world!
Version: 1.0.0
Hostname: hello-web
/ # wget -qO- --timeout=2 http://hello-web-2:8080
wget: download timed out
/ #
```

5.5 Clean-up the resources created in this Section

```
$ kubectl delete -f foo-allow-to-hello.yaml

$ kubectl delete -f hello-allow-from-foo.yaml
```

```
...
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          user: alice
      podSelector:
        matchLabels:
          role: client
  ...
```

This policy contains a single `from` element allowing connections from Pods with the label `role=client` in namespaces with the label `user=alice`.

```
...
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          user: alice
    - podSelector:
        matchLabels:
          role: client
  ...
```

Above -- It contains two elements in the `from` array, and allows connections from Pods in the local Namespace with the label `role=client`, *or* from any Pod in any namespace with the label `user=alice`.

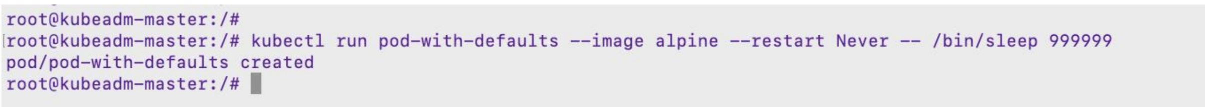
6 CONFIGURING CONTAINER SECURITY CONTEXT

6.1 Defining Security Contexts With Default User

Note: It allows you to lock down your containers, so that only certain processes can do certain things. This ensures the stability of your containers and allows you to give control or take it away. In this lesson, we'll go through how to set the security context at the container level and the pod level.

1. Run an alpine container with default security

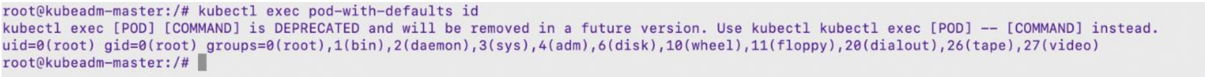
   $ kubectl run pod-with-defaults --image alpine --restart Never -- /bin/sleep 999999



```
root@kubeadm-master:/#
root@kubeadm-master:/# kubectl run pod-with-defaults --image alpine --restart Never -- /bin/sleep 999999
pod/pod-with-defaults created
root@kubeadm-master:/#
```

2. Check the ID on the container:

   $ kubectl exec pod-with-defaults id

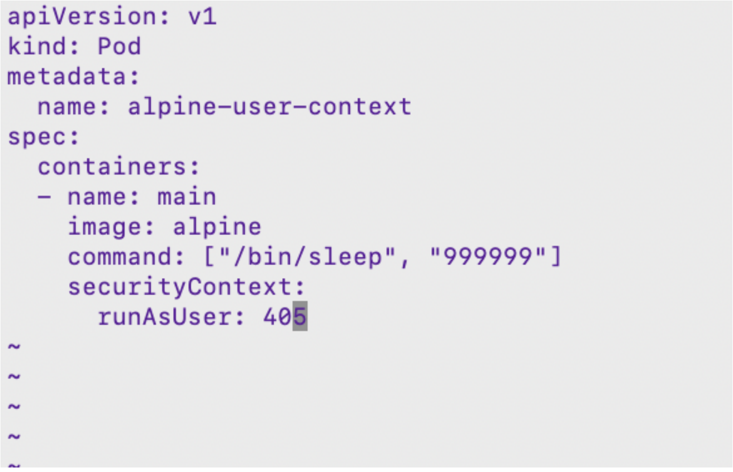


```
root@kubeadm-master:/# kubectl exec pod-with-defaults id
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl kubectl exec [POD] -- [COMMAND] instead.
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(dialout),26(tape),27(video)
root@kubeadm-master:/#
```

6.2 Defining Security Contexts With Specific User

1. The YAML for a container that runs as a user. View the file security-cxt.yaml

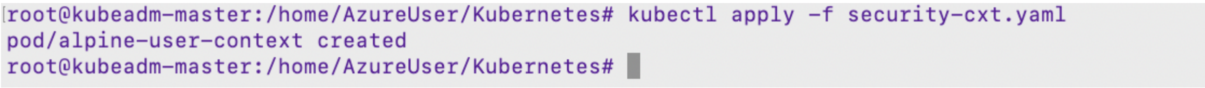
   $ vim security-cxt.yaml



```
apiVersion: v1
kind: Pod
metadata:
  name: alpine-user-context
spec:
  containers:
  - name: main
    image: alpine
    command: ["/bin/sleep", "999999"]
    securityContext:
      runAsUser: 405
~
~
~
~
~
```

2. Create the resource from above yaml file

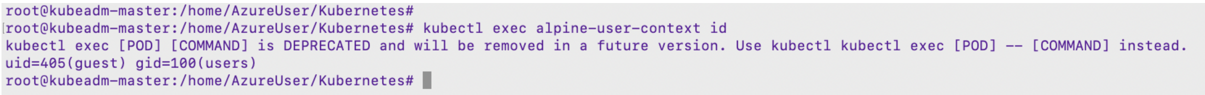
   $ kubectl apply -f security-cxt.yaml



```
root@kubeadm-master:/home/AzureUser/Kubernetes# kubectl apply -f security-cxt.yaml
pod/alpine-user-context created
root@kubeadm-master:/home/AzureUser/Kubernetes#
```

3. Check the user context

   $ kubectl exec alpine-user-context id

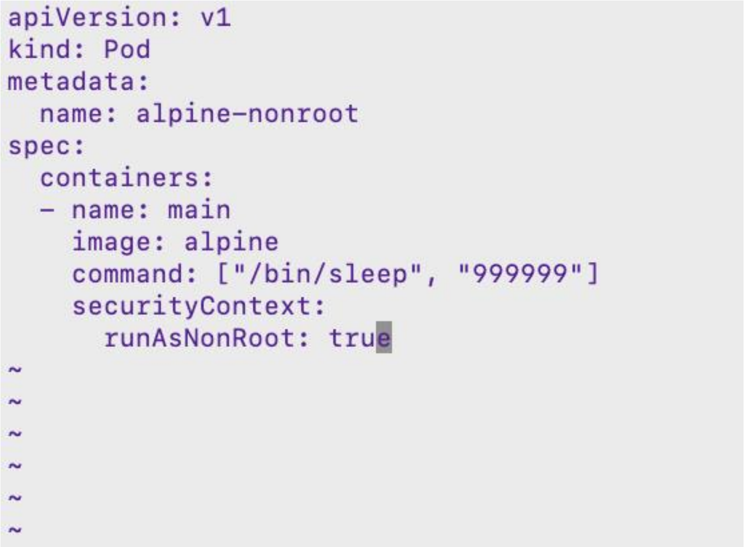


```
root@kubeadm-master:/home/AzureUser/Kubernetes#
root@kubeadm-master:/home/AzureUser/Kubernetes# kubectl exec alpine-user-context id
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl kubectl exec [POD] -- [COMMAND] instead.
uid=405(guest) gid=100(users)
root@kubeadm-master:/home/AzureUser/Kubernetes#
```

6.3 Defining Security Contexts With non-root User

1. The YAML for a pod that runs the container as non-root:

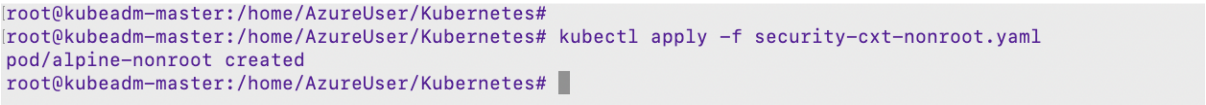
   $ vim security-cxt-nonroot.yaml



```
apiVersion: v1
kind: Pod
metadata:
  name: alpine-nonroot
spec:
  containers:
  - name: main
    image: alpine
    command: ["/bin/sleep", "999999"]
    securityContext:
      runAsNonRoot: true
~
~
~
~
~
~
```

2. Create a pod that runs the container as non-root:

   $ kubectl apply -f security-cxt-nonroot.yaml

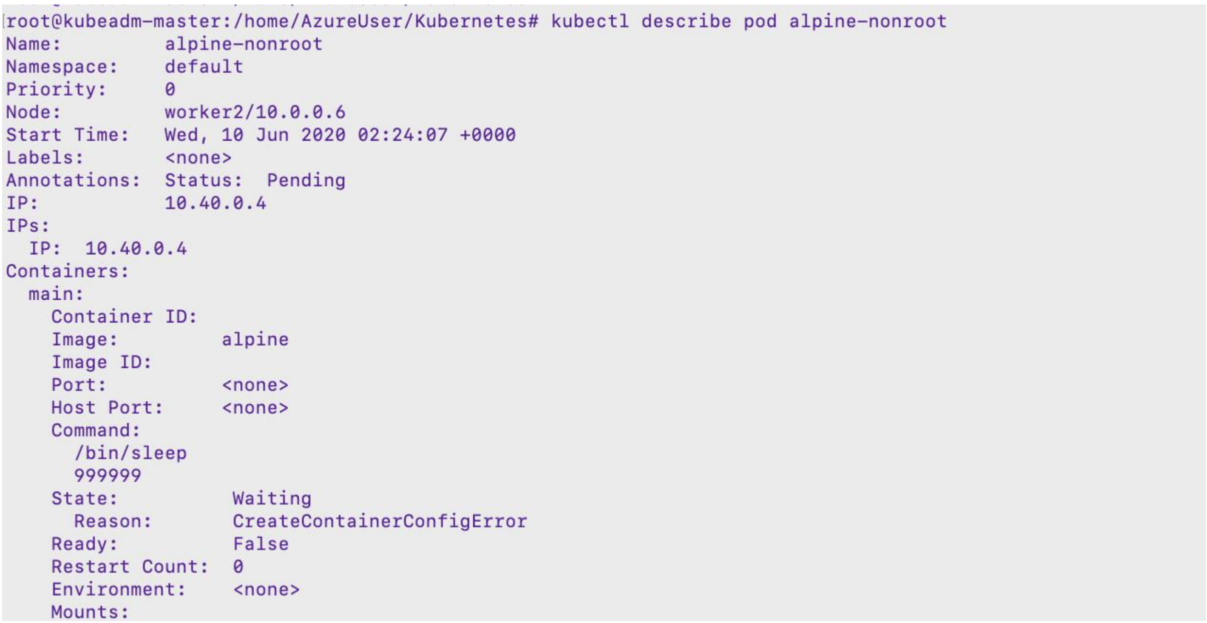


```
root@kubeadm-master:/home/AzureUser/Kubernetes#
root@kubeadm-master:/home/AzureUser/Kubernetes# kubectl apply -f security-cxt-nonroot.yaml
pod/alpine-nonroot created
root@kubeadm-master:/home/AzureUser/Kubernetes# 
```

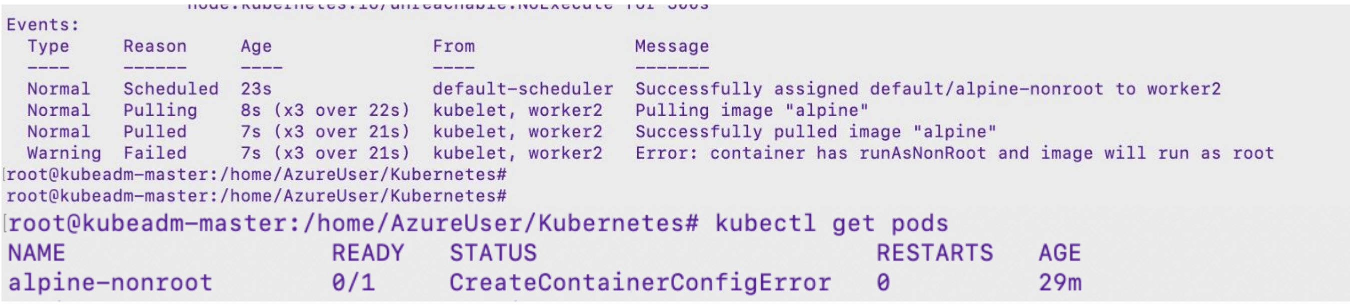
3. View more information about the pod error:

   $ kubectl describe pod alpine-nonroot

   $ kubectl get pods



```
root@kubeadm-master:/home/AzureUser/Kubernetes# kubectl describe pod alpine-nonroot
Name:         alpine-nonroot
Namespace:    default
Priority:     0
Node:         worker2/10.0.0.6
Start Time:   Wed, 10 Jun 2020 02:24:07 +0000
Labels:       <none>
Annotations:  Status:  Pending
IP:           10.40.0.4
IPs:
  IP:  10.40.0.4
Containers:
  main:
    Container ID:
    Image:         alpine
    Image ID:
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sleep
      999999
    State:          Waiting
      Reason:       CreateContainerConfigError
    Ready:          False
    Restart Count:  0
    Environment:    <none>
    Mounts:
```

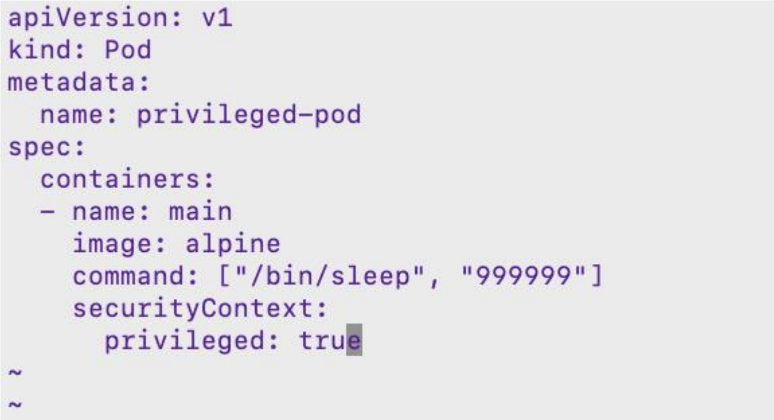


```
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type     Reason     Age                From               Message
  ----     ------     ----               ----               -------
  Normal   Scheduled  23s                default-scheduler  Successfully assigned default/alpine-nonroot to worker2
  Normal   Pulling    8s (x3 over 22s)   kubelet, worker2   Pulling image "alpine"
  Normal   Pulled     7s (x3 over 21s)   kubelet, worker2   Successfully pulled image "alpine"
  Warning  Failed     7s (x3 over 21s)   kubelet, worker2   Error: container has runAsNonRoot and image will run as root
root@kubeadm-master:/home/AzureUser/Kubernetes#
root@kubeadm-master:/home/AzureUser/Kubernetes#
root@kubeadm-master:/home/AzureUser/Kubernetes# kubectl get pods
NAME                READY   STATUS                       RESTARTS   AGE
alpine-nonroot      0/1     CreateContainerConfigError   0          29m
```

6.4 Defining Security Contexts With Privileged Container POD

1. The YAML for a privileged container pod:

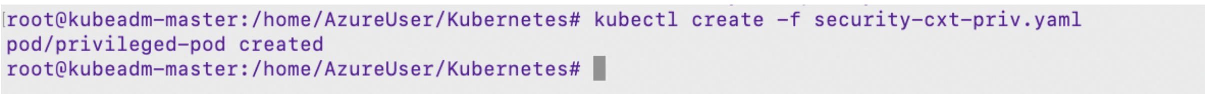
   $ vim security-cxt-priv.yaml



```
apiVersion: v1
kind: Pod
metadata:
  name: privileged-pod
spec:
  containers:
  - name: main
    image: alpine
    command: ["/bin/sleep", "999999"]
    securityContext:
      privileged: true
~
~
```

2. Create the privileged container pod:

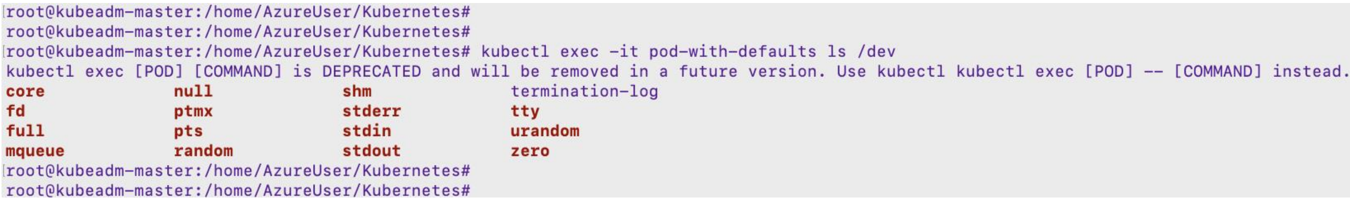
   kubectl apply -f security-cxt-priv.yaml



```
root@kubeadm-master:/home/AzureUser/Kubernetes# kubectl create -f security-cxt-priv.yaml
pod/privileged-pod created
root@kubeadm-master:/home/AzureUser/Kubernetes#
```

3. View the devices on the default container:

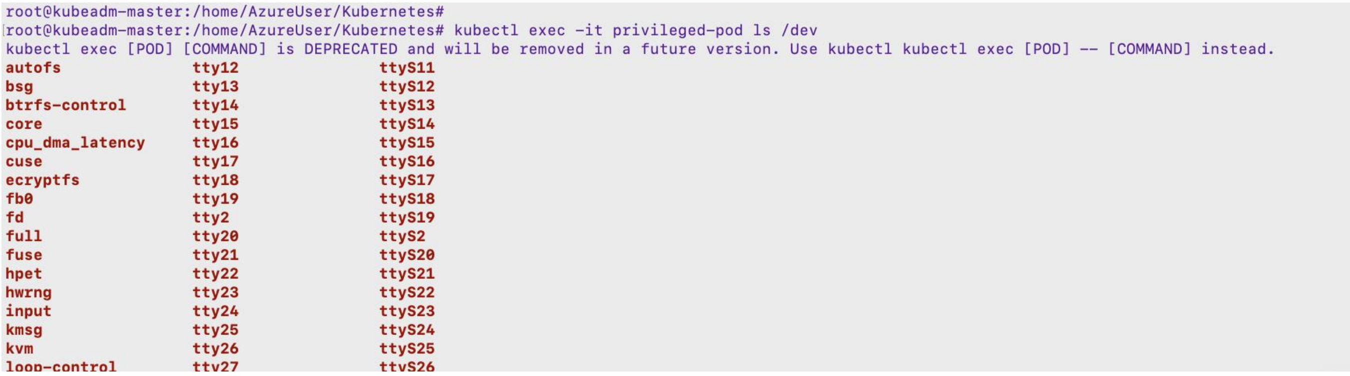
   $ kubectl exec -it pod-with-defaults ls /dev



```
root@kubeadm-master:/home/AzureUser/Kubernetes#
root@kubeadm-master:/home/AzureUser/Kubernetes#
root@kubeadm-master:/home/AzureUser/Kubernetes# kubectl exec -it pod-with-defaults ls /dev
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl kubectl exec [POD] -- [COMMAND] instead.
core             null             shm              termination-log
fd               ptmx             stderr           tty
full             pts              stdin            urandom
mqueue           random           stdout           zero
root@kubeadm-master:/home/AzureUser/Kubernetes#
root@kubeadm-master:/home/AzureUser/Kubernetes#
```

4. View the devices on the privileged pod container:

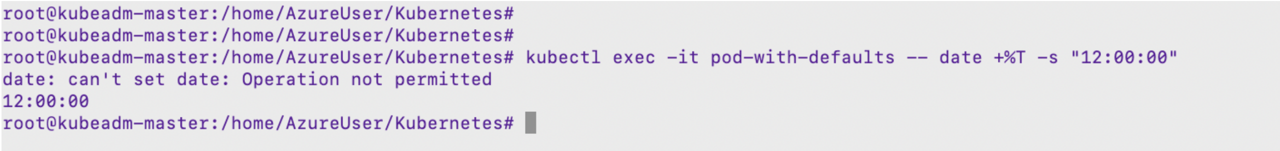
```
$ kubectl exec -it privileged-pod ls /dev
```



```
root@kubeadm-master:/home/AzureUser/Kubernetes#
root@kubeadm-master:/home/AzureUser/Kubernetes# kubectl exec -it privileged-pod ls /dev
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl kubectl exec [POD] -- [COMMAND] instead.
autofs              tty12               ttyS11
bsg                 tty13               ttyS12
btrfs-control       tty14               ttyS13
core                tty15               ttyS14
cpu_dma_latency     tty16               ttyS15
cuse                tty17               ttyS16
ecryptfs            tty18               ttyS17
fb0                 tty19               ttyS18
fd                  tty2                ttyS19
full                tty20               ttyS2
fuse                tty21               ttyS20
hpet                tty22               ttyS21
hwrng               tty23               ttyS22
input               tty24               ttyS23
kmsg                tty25               ttyS24
kvm                 tty26               ttyS25
loop-control        tty27               ttyS26
```

5. Try to change the time on a default container pod:

```
$ kubectl exec -it pod-with-defaults -- date +%T -s "12:00:00"
```

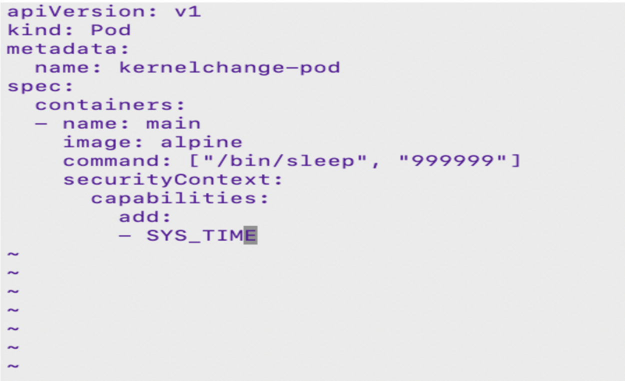


```
root@kubeadm-master:/home/AzureUser/Kubernetes#
root@kubeadm-master:/home/AzureUser/Kubernetes#
root@kubeadm-master:/home/AzureUser/Kubernetes# kubectl exec -it pod-with-defaults -- date +%T -s "12:00:00"
date: can't set date: Operation not permitted
12:00:00
root@kubeadm-master:/home/AzureUser/Kubernetes#
```

6.5 Defining Security Contexts With Privileged Container POD – add Capability

1. The YAML for a container that will allow you to change the time:

```
$ vim security-cxt-time.yaml
```



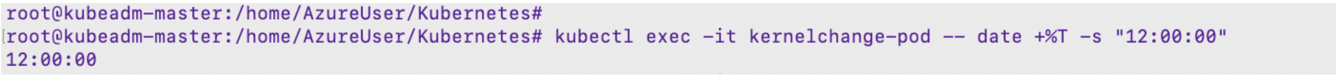
```
apiVersion: v1
kind: Pod
metadata:
  name: kernelchange-pod
spec:
  containers:
  - name: main
    image: alpine
    command: ["/bin/sleep", "999999"]
    securityContext:
      capabilities:
        add:
        - SYS_TIME
~
~
~
~
~
~
~
```

2. Create the pod that will allow you to change the container’s time:

$ kubectl create -f security-cxt-time.yaml

3. Change the time on a container:

$ kubectl exec -it kernelchange-pod -- date +%T -s "12:00:00"

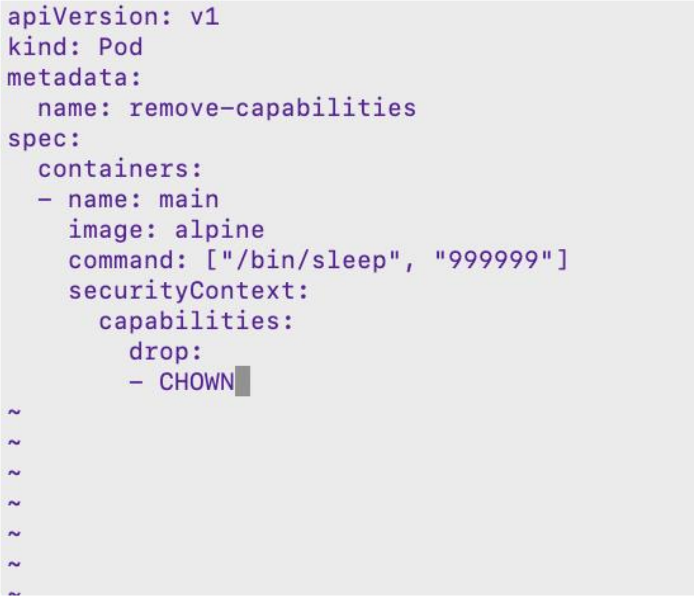


```
root@kubeadm-master:/home/AzureUser/Kubernetes#
root@kubeadm-master:/home/AzureUser/Kubernetes# kubectl exec -it kernelchange-pod -- date +%T -s "12:00:00"
12:00:00
```

6.6 Defining security contexts with privileged container pod – remove capability

1. The YAML for a container that removes capabilities:

$ vim security-cxt-rmcap.yaml



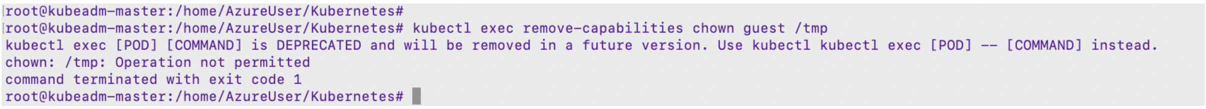
```
apiVersion: v1
kind: Pod
metadata:
  name: remove-capabilities
spec:
  containers:
  - name: main
    image: alpine
    command: ["/bin/sleep", "999999"]
    securityContext:
      capabilities:
        drop:
        - CHOWN
~
~
~
~
~
~
~
```

2. Create a pod that’s container has capabilities removed:

$ kubectl apply -f security-cxt-rmcap.yaml

3. Try to change the ownership of a container with removed capability:

$ kubectl exec remove-capabilities chown guest /tmp



```
root@kubeadm-master:/home/AzureUser/Kubernetes#
root@kubeadm-master:/home/AzureUser/Kubernetes# kubectl exec remove-capabilities chown guest /tmp
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl kubectl exec [POD] -- [COMMAND] instead.
chown: /tmp: Operation not permitted
command terminated with exit code 1
root@kubeadm-master:/home/AzureUser/Kubernetes#
```

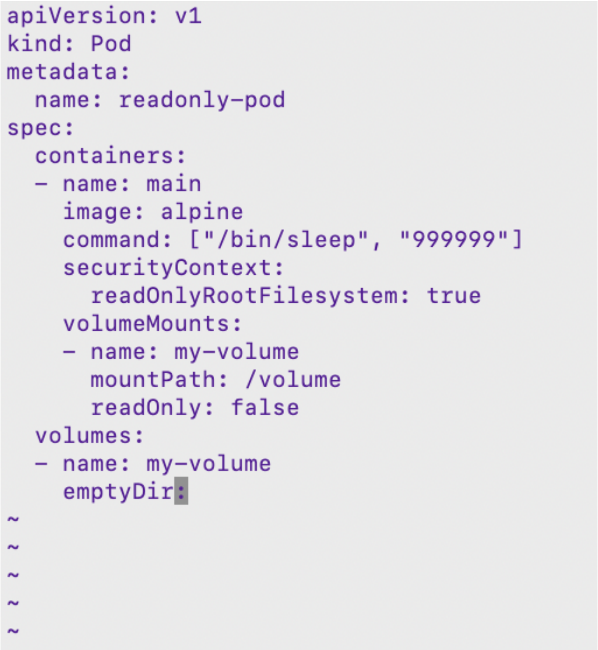
6.7 Defining security contexts with privileged container pod – ReadOnly

1. The YAML for a pod container that can’t write to the local filesystem:

   $ vim security-cxt-readonly.yaml

2. Create a pod that will not allow you to write to the local container filesystem:

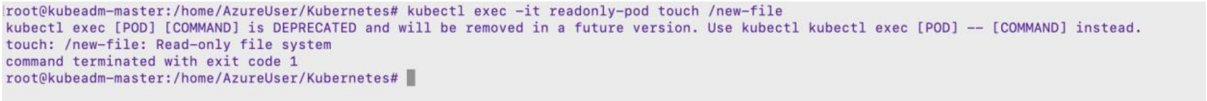
   $ kubectl apply -f security-cxt-readonly.yaml



```
apiVersion: v1
kind: Pod
metadata:
  name: readonly-pod
spec:
  containers:
  - name: main
    image: alpine
    command: ["/bin/sleep", "999999"]
    securityContext:
      readOnlyRootFilesystem: true
    volumeMounts:
    - name: my-volume
      mountPath: /volume
      readOnly: false
  volumes:
  - name: my-volume
    emptyDir:
~
~
~
~
~
```

3. Try to write to the container filesystem:

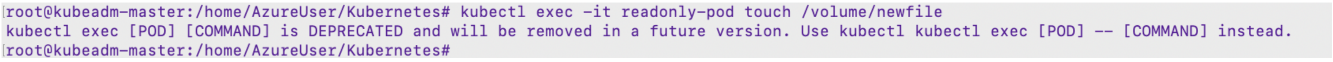
   $ kubectl exec -it readonly-pod touch /new-file



```
root@kubeadm-master:/home/AzureUser/Kubernetes# kubectl exec -it readonly-pod touch /new-file
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl kubectl exec [POD] -- [COMMAND] instead.
touch: /new-file: Read-only file system
command terminated with exit code 1
root@kubeadm-master:/home/AzureUser/Kubernetes#
```

4. Create a file on the volume mounted to the container:

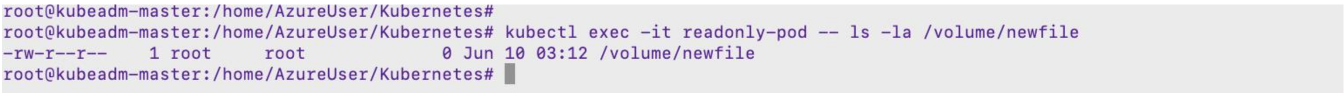
$ kubectl exec -it readonly-pod touch /volume/newfile



```
root@kubeadm-master:/home/AzureUser/Kubernetes# kubectl exec -it readonly-pod touch /volume/newfile
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl kubectl exec [POD] -- [COMMAND] instead.
root@kubeadm-master:/home/AzureUser/Kubernetes#
```

5. View the file on the volume that’s mounted:

$ kubectl exec -it readonly-pod -- ls -la /volume/newfile



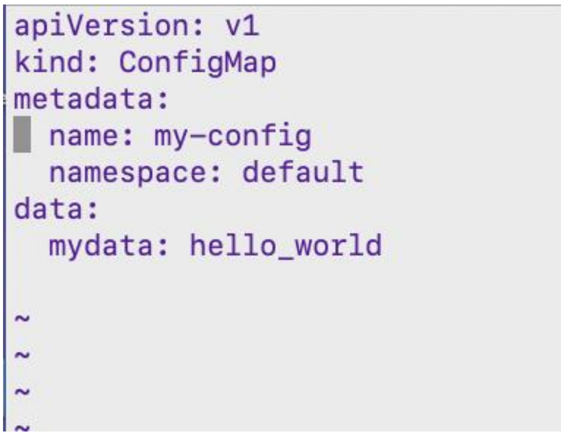
```
root@kubeadm-master:/home/AzureUser/Kubernetes#
root@kubeadm-master:/home/AzureUser/Kubernetes# kubectl exec -it readonly-pod -- ls -la /volume/newfile
-rw-r--r--    1 root     root             0 Jun 10 03:12 /volume/newfile
root@kubeadm-master:/home/AzureUser/Kubernetes#
```

7 WORKING WITH CONFIGMAP

7.1 Setting Container Environment Variables using ConfigMap

1. Create a ConfigMap from the yaml file and enter the contents given below

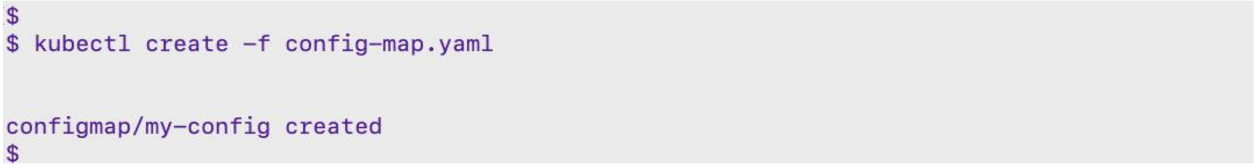
   $ vi config-map.yaml



```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
  namespace: default
data:
  mydata: hello_world

~
~
~
~
```

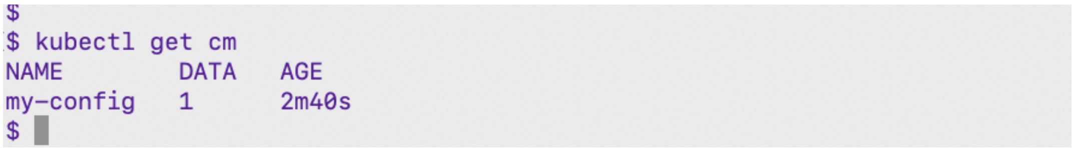
2. Create the ConfigMap using the yaml

   $ kubectl create -f config-map.yaml



```
$
$ kubectl create -f config-map.yaml


configmap/my-config created
$
```

$ kubectl get cm

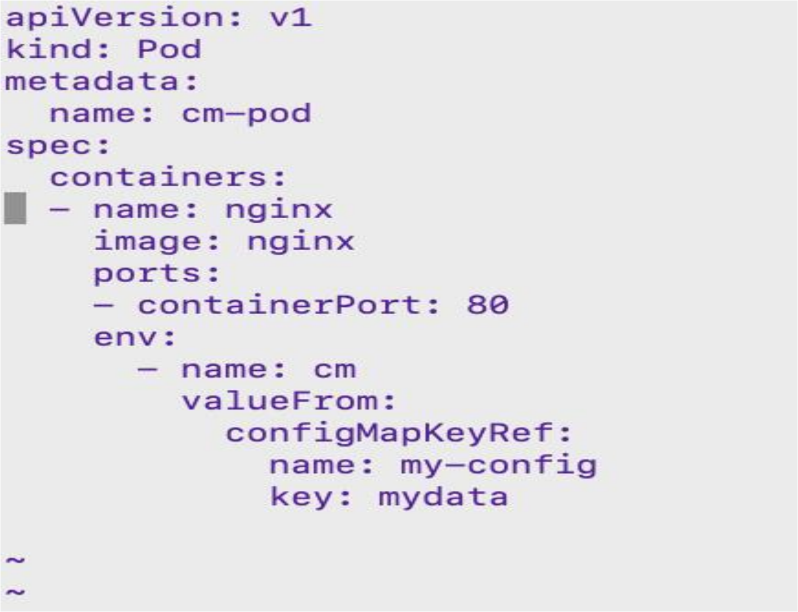


```
$
$ kubectl get cm
NAME          DATA    AGE
my-config     1       2m40s
$
```

7.2 Create Pod that Uses ConfigMap

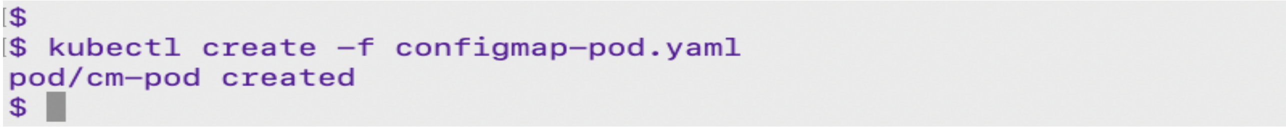
1. View and create the pod from configmap-pod.yaml file

   $ vi configmap-pod.yaml



```
apiVersion: v1
kind: Pod
metadata:
  name: cm-pod
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    env:
      - name: cm
        valueFrom:
          configMapKeyRef:
            name: my-config
            key: mydata

~
~
```

$ kubectl create -f configmap-pod.yaml

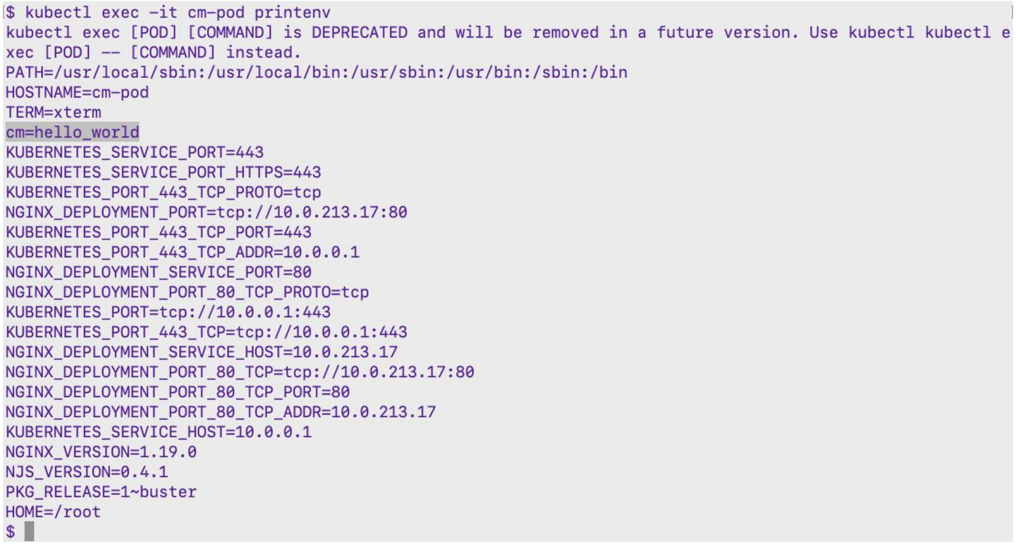


```
$
$ kubectl create -f configmap-pod.yaml
pod/cm-pod created
$
```

7.3 Verify Pod uses ConfigMap to set the Environmental Variable

1. Once the Pod is up, verify that the environment variable specified in the ConfigMap is set in the container

$ kubectl exec -it cm-pod printenv

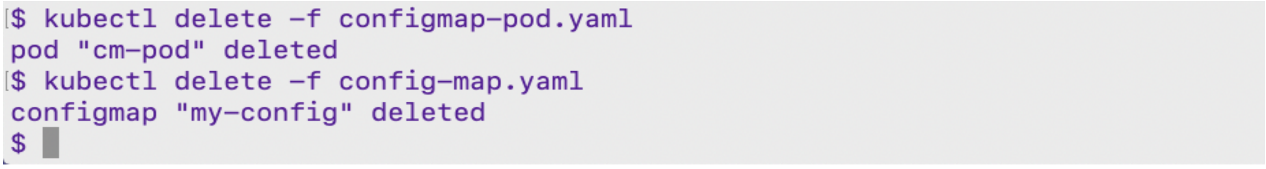


```
$ kubectl exec -it cm-pod printenv
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl kubectl e
xec [POD] -- [COMMAND] instead.
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=cm-pod
TERM=xterm
cm=hello_world
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT_443_TCP_PROTO=tcp
NGINX_DEPLOYMENT_PORT=tcp://10.0.213.17:80
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP_ADDR=10.0.0.1
NGINX_DEPLOYMENT_SERVICE_PORT=80
NGINX_DEPLOYMENT_PORT_80_TCP_PROTO=tcp
KUBERNETES_PORT=tcp://10.0.0.1:443
KUBERNETES_PORT_443_TCP=tcp://10.0.0.1:443
NGINX_DEPLOYMENT_SERVICE_HOST=10.0.213.17
NGINX_DEPLOYMENT_PORT_80_TCP=tcp://10.0.213.17:80
NGINX_DEPLOYMENT_PORT_80_TCP_PORT=80
NGINX_DEPLOYMENT_PORT_80_TCP_ADDR=10.0.213.17
KUBERNETES_SERVICE_HOST=10.0.0.1
NGINX_VERSION=1.19.0
NJS_VERSION=0.4.1
PKG_RELEASE=1~buster
HOME=/root
$
```

7.4 Clean-up the Resources

```
$ kubectl delete -f configmap-pod.yaml
```

```
$ kubectl delete -f config-map.yaml
```

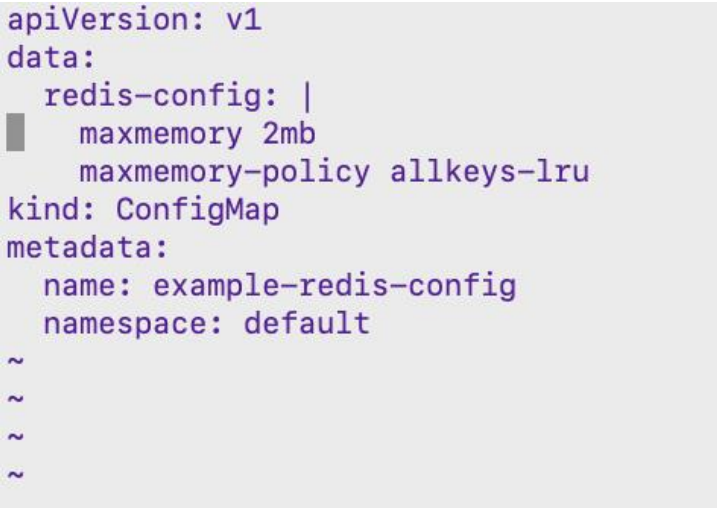


```
$ kubectl delete -f configmap-pod.yaml
pod "cm-pod" deleted
$ kubectl delete -f config-map.yaml
configmap "my-config" deleted
$
```

7.5 Setting Configuration File with Volume using ConfigMap

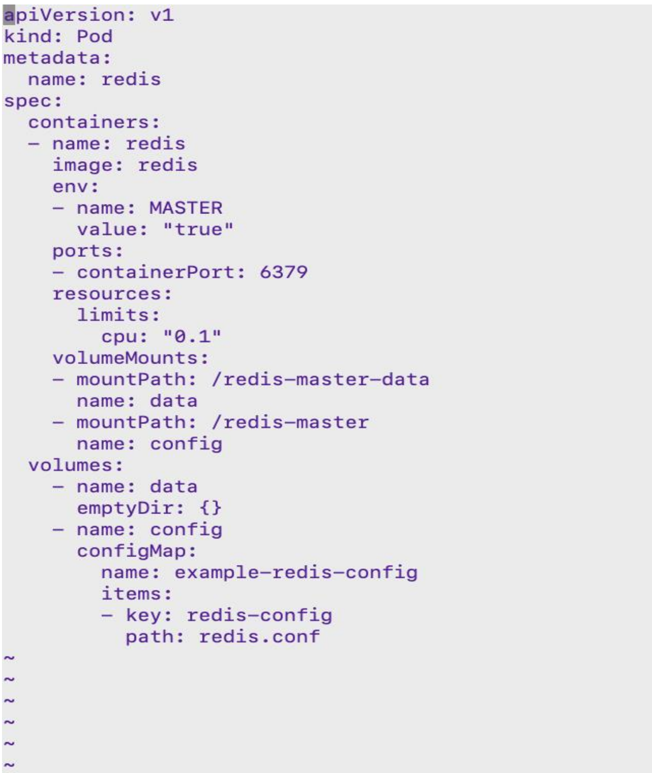
1. View the below ConfigMap and Pod yaml files and create the resources using them.

```
$ vim redis-cm.yaml
```



```
apiVersion: v1
data:
  redis-config: |
    maxmemory 2mb
    maxmemory-policy allkeys-lru
kind: ConfigMap
metadata:
  name: example-redis-config
  namespace: default
~
~
~
~
```

$ vim redis-pod.yaml

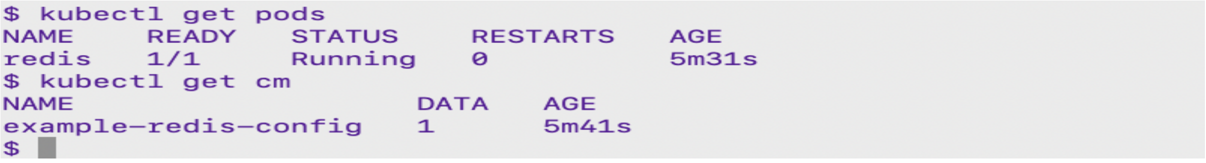


```
apiVersion: v1
kind: Pod
metadata:
  name: redis
spec:
  containers:
  - name: redis
    image: redis
    env:
    - name: MASTER
      value: "true"
    ports:
    - containerPort: 6379
    resources:
      limits:
        cpu: "0.1"
    volumeMounts:
    - mountPath: /redis-master-data
      name: data
    - mountPath: /redis-master
      name: config
  volumes:
    - name: data
      emptyDir: {}
    - name: config
      configMap:
        name: example-redis-config
        items:
        - key: redis-config
          path: redis.conf
~
~
~
~
~
~
```

2. Verify by listing the created resources

$ kubectl get pods

$ kubectl get cm

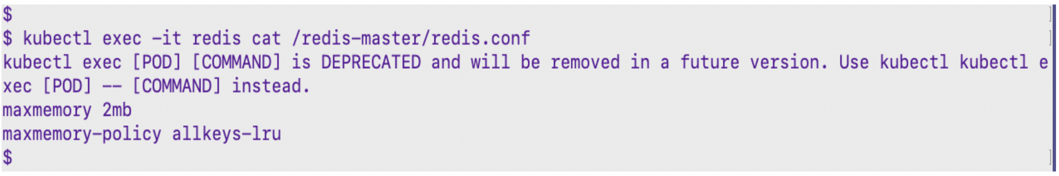


```
$ kubectl get pods
NAME     READY   STATUS    RESTARTS   AGE
redis    1/1     Running   0          5m31s
$ kubectl get cm
NAME                   DATA   AGE
example-redis-config   1      5m41s
$ 
```

7.6 Verify Mounting of ConfigMap as Volume

1. See that the config file redis.conf is present at /redis-master/ and is having the contents specified in the ConfigMap

   $ kubectl exec -it redis cat /redis-master/redis.conf



```
$
$ kubectl exec -it redis cat /redis-master/redis.conf
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl kubectl e
xec [POD] -- [COMMAND] instead.
maxmemory 2mb
maxmemory-policy allkeys-lru
$
```

7.7 Delete all the resources created in this task

$ kubectl delete -f redis-pod.yaml

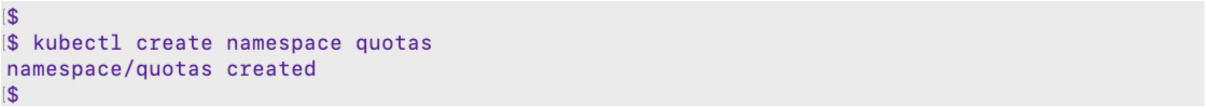
$ kubectl delete -f redis-cm.yaml

8 LIMITING RESOURCES WITH RESOURCE QUOTA

8.1 Create Namespace

1. Create a namespace called quotas

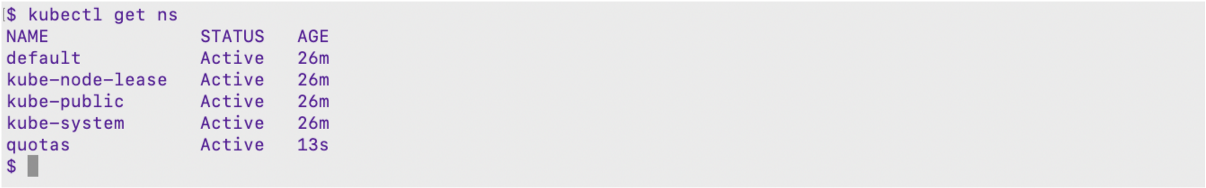
    $ kubectl create namespace quotas



```
$
$ kubectl create namespace quotas
namespace/quotas created
$
```

2. Verify namespace creation

    $ kubectl get ns

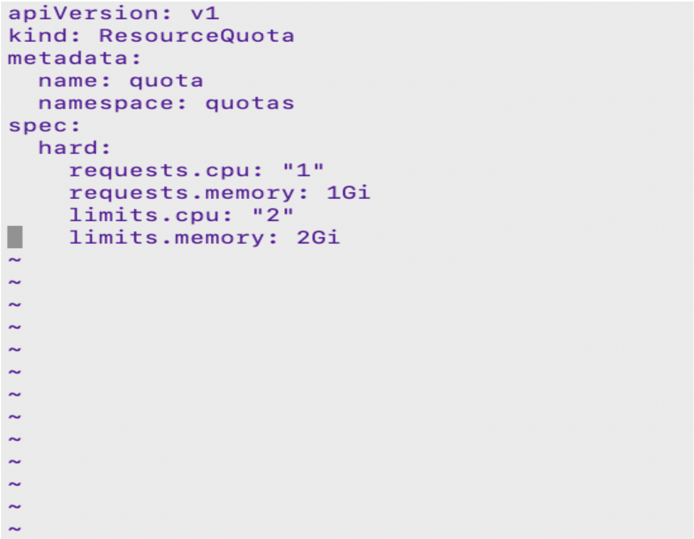


```
$ kubectl get ns
NAME              STATUS   AGE
default           Active   26m
kube-node-lease   Active   26m
kube-public       Active   26m
kube-system       Active   26m
quotas            Active   13s
$
```

8.2 Create ResourceQuota

1. Create a file quota.yaml

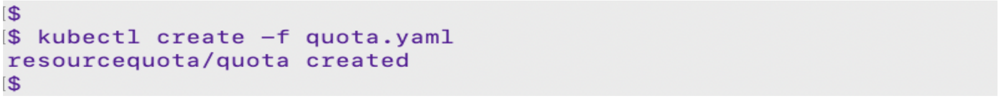
    $ vi quota.yaml



```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: quota
  namespace: quotas
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
~
~
~
~
~
~
~
~
~
~
~
~
~
```

2. Create the resourcequota from the yaml

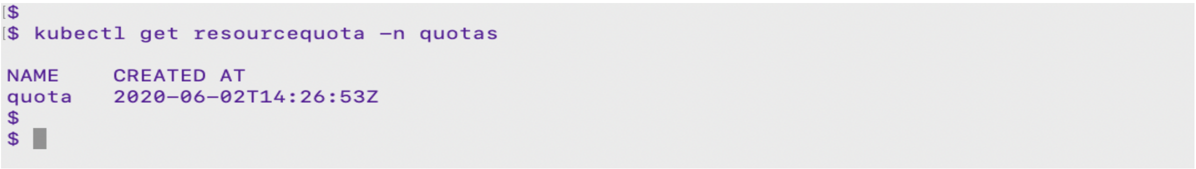
$ kubectl create -f quota.yaml



```
$
$ kubectl create -f quota.yaml
resourcequota/quota created
$
```

3. Verify resourcequota creation

$ kubectl get resourcequota -n quotas

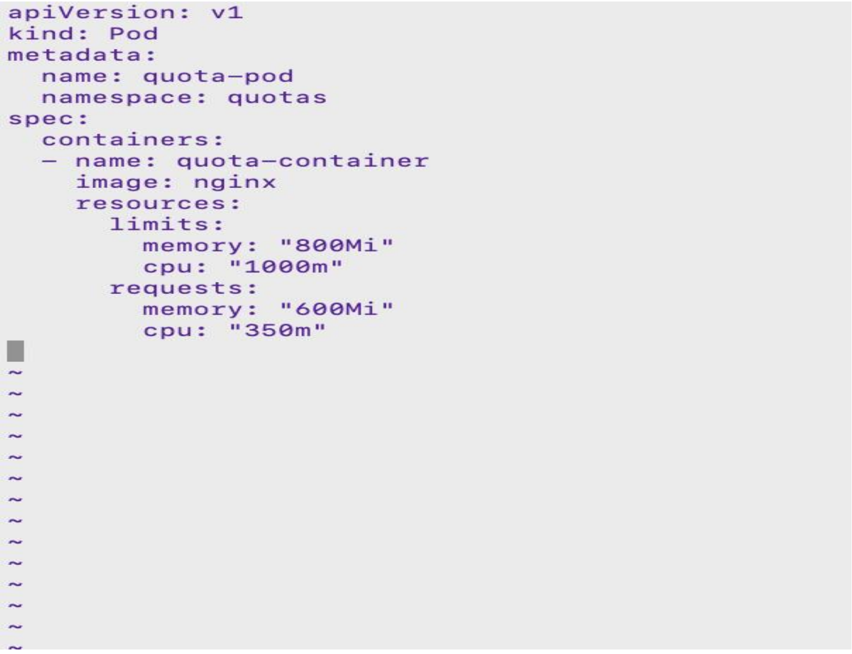


```
$
$ kubectl get resourcequota -n quotas

NAME     CREATED AT
quota    2020-06-02T14:26:53Z
$
$ █
```

8.3 Simulate Resource Creation Failure due to Resourcequota Limits

1. We will create two pods exceeding the request data to demonstrate the functionality of resourcequotas. Create the first pod from yaml file called quota-pod.yaml

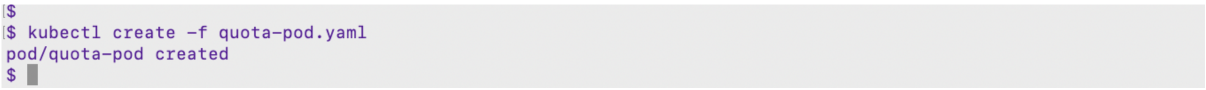
$ vi quota-pod.yaml



```
apiVersion: v1
kind: Pod
metadata:
  name: quota-pod
  namespace: quotas
spec:
  containers:
  - name: quota-container
    image: nginx
    resources:
      limits:
        memory: "800Mi"
        cpu: "1000m"
      requests:
        memory: "600Mi"
        cpu: "350m"

~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

Create the pod using the yaml created in above step

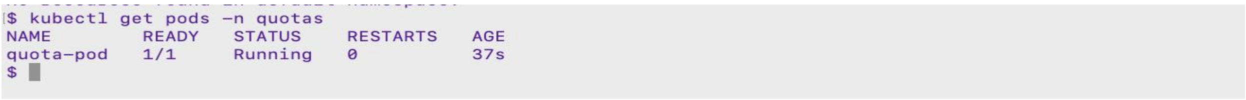
$ kubectl create -f quota-pod.yaml



```
$
$ kubectl create -f quota-pod.yaml
pod/quota-pod created
$
```

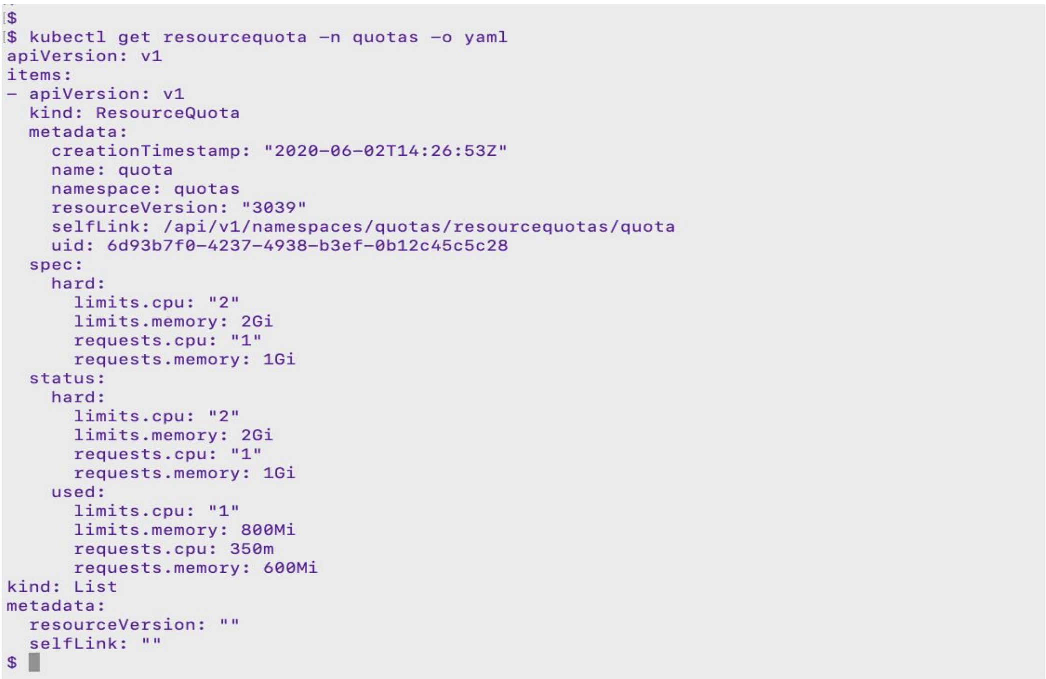
2. After creating the pod, check the pod and resources used by the pod

   $ kubectl get pods -n quotas



```
No resources found in default namespace.
$ kubectl get pods -n quotas
NAME        READY   STATUS    RESTARTS   AGE
quota-pod   1/1     Running   0          37s
$
```

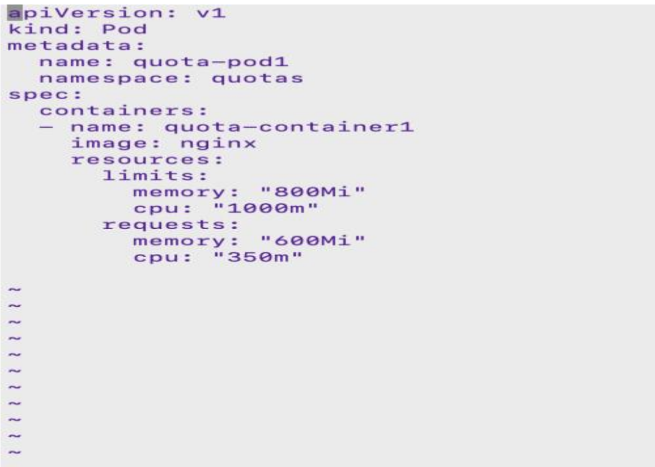
$ kubectl get resourcequota -n quotas -o yaml



```
$
$ kubectl get resourcequota -n quotas -o yaml
apiVersion: v1
items:
- apiVersion: v1
  kind: ResourceQuota
  metadata:
    creationTimestamp: "2020-06-02T14:26:53Z"
    name: quota
    namespace: quotas
    resourceVersion: "3039"
    selfLink: /api/v1/namespaces/quotas/resourcequotas/quota
    uid: 6d93b7f0-4237-4938-b3ef-0b12c45c5c28
  spec:
    hard:
      limits.cpu: "2"
      limits.memory: 2Gi
      requests.cpu: "1"
      requests.memory: 1Gi
  status:
    hard:
      limits.cpu: "2"
      limits.memory: 2Gi
      requests.cpu: "1"
      requests.memory: 1Gi
    used:
      limits.cpu: "1"
      limits.memory: 800Mi
      requests.cpu: 350m
      requests.memory: 600Mi
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
$
```

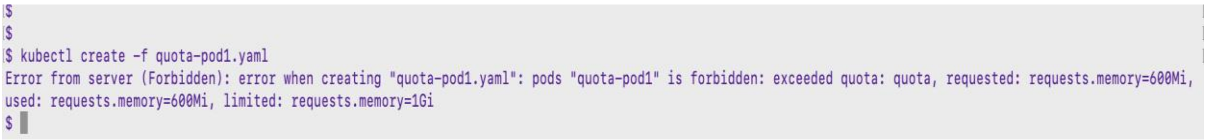
3. Create the similar pod again and see that it does not get created due to the set resourcequotas. Two pods together will request 1.2Gi of memory while quota is set at 1Gi

$ vim quota-pod1.yaml



```
apiVersion: v1
kind: Pod
metadata:
  name: quota-pod1
  namespace: quotas
spec:
  containers:
  - name: quota-container1
    image: nginx
    resources:
      limits:
        memory: "800Mi"
        cpu: "1000m"
      requests:
        memory: "600Mi"
        cpu: "350m"

~
~
~
~
~
~
~
~
~
~
~
```

$ kubectl create -f quota-pod1.yaml

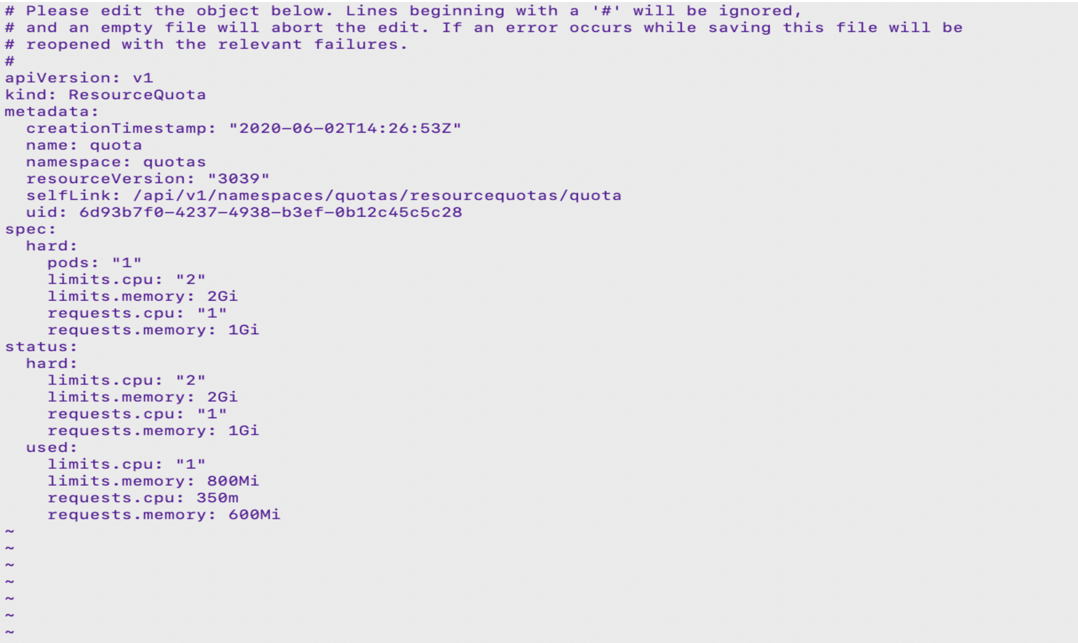


```
$
$
$ kubectl create -f quota-pod1.yaml
Error from server (Forbidden): error when creating "quota-pod1.yaml": pods "quota-pod1" is forbidden: exceeded quota: quota, requested: requests.memory=600Mi,
used: requests.memory=600Mi, limited: requests.memory=1Gi
$
```

8.4 Simulate Resource Creation Failure due to Resource Count Limits

1. Edit the existing resourcequota and include a limit for the number of pods. Add pods: “1” to the specification limits section in the file as shown below

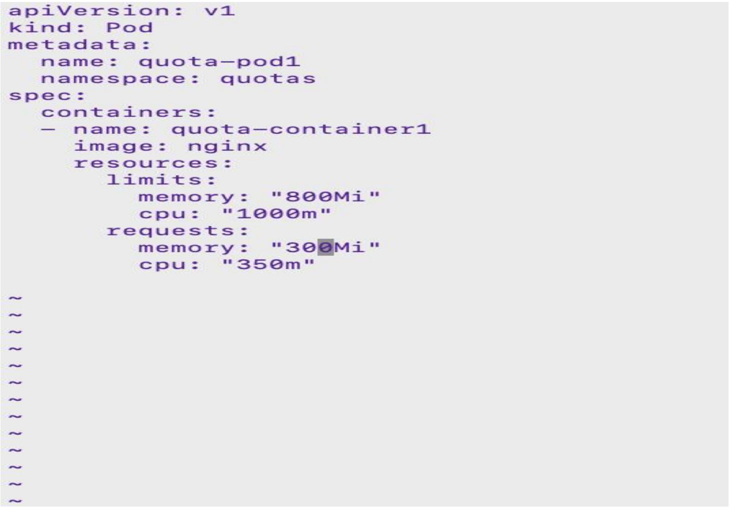
   $ kubectl edit resourcequotas quota -n quotas



```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: ResourceQuota
metadata:
  creationTimestamp: "2020-06-02T14:26:53Z"
  name: quota
  namespace: quotas
  resourceVersion: "3039"
  selfLink: /api/v1/namespaces/quotas/resourcequotas/quota
  uid: 6d93b7f0-4237-4938-b3ef-0b12c45c5c28
spec:
  hard:
    pods: "1"
    limits.cpu: "2"
    limits.memory: 2Gi
    requests.cpu: "1"
    requests.memory: 1Gi
status:
  hard:
    limits.cpu: "2"
    limits.memory: 2Gi
    requests.cpu: "1"
    requests.memory: 1Gi
  used:
    limits.cpu: "1"
    limits.memory: 800Mi
    requests.cpu: 350m
    requests.memory: 600Mi
~
~
~
~
~
~
~
```

2. Modify the pod memory limit of quota-pod1 pod to ensure that pod creation is not affected by the memory limit

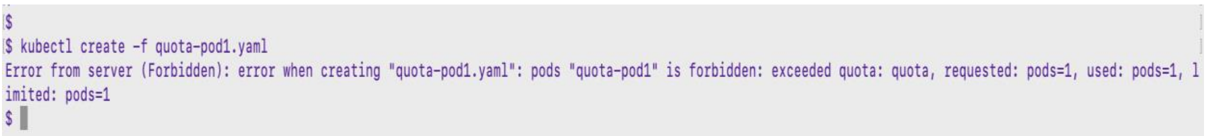
   $ vi quota-pod1.yaml



```
apiVersion: v1
kind: Pod
metadata:
  name: quota-pod1
  namespace: quotas
spec:
  containers:
  - name: quota-container1
    image: nginx
    resources:
      limits:
        memory: "800Mi"
        cpu: "1000m"
      requests:
        memory: "300Mi"
        cpu: "350m"

~
~
~
~
~
~
~
~
~
~
~
~
~
```

3. Now, try to create a pod and note that it will not be created due to the restriction on the number of pods

```
$ kubectl create -f quota-pod.yaml
```

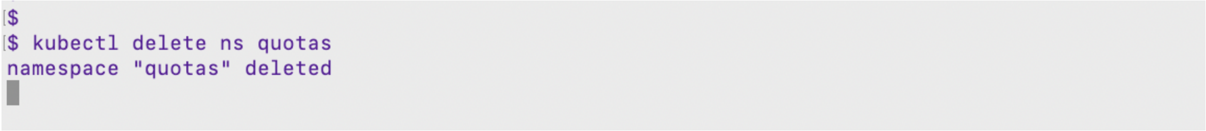


```
$
$ kubectl create -f quota-pod1.yaml
Error from server (Forbidden): error when creating "quota-pod1.yaml": pods "quota-pod1" is forbidden: exceeded quota: quota, requested: pods=1, used: pods=1, l
imited: pods=1
$
```

8.5 Clean-up of all resources created in this section

Delete the quota to clean up

```
$ kubectl delete ns quotas
```



```
$
$ kubectl delete ns quotas
namespace "quotas" deleted
```

9 SUMMARY

In this guide we Covered:

- Authentication and Authorisation using RBAC
- Configuring Network Policies for Applications
- Configuring Container Security Context
- Working With ConfigMap
- Limiting Resources With Resource Quota