RX-M Cloud Native Consulting

Kubernetes

Lab 7 – Namespaces & Admission Control

Namespaces

Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces. Namespaces are the Kubernetes multi-tenant mechanism and allow different teams to create resources independently of each other. Namespaces can be given quotas and individual user can be allowed in some namespaces and excluded from others. Using Namespaces, a single cluster can satisfy the needs of multiple user communities. Each user community can have their own namespace allowing them to work in (virtual) isolation from other communities.

Each namespace has its own:

- **resources** - pods, services, replica sets, etc.
- **policies** - who can or cannot perform actions in their community
- **constraints** - this community is allowed to run this many pods, etc.

Cluster operators can delegate namespace authority to trusted users in those communities.

1. Working with Namespaces

Try listing the namespaces available and looking at the details of the current namespace:

```
user@ubuntu:~/configmaps$ cd ~
```

```
user@ubuntu:~$ kubectl get namespaces

NAME              STATUS   AGE
default           Active   3h12m
kube-node-lease   Active   3h12m
kube-public       Active   3h12m
kube-system       Active   3h12m

user@ubuntu:~$
```

Kubernetes starts with three initial namespaces:

- `default` - the default namespace for objects with no other namespace
- `kube-system` - the namespace for objects created by the Kubernetes system (houses control plane components)
- `kube-public` - this namespace is readable by all users (including those not authenticated)
  - Reserved for cluster usage, in case that some resources should be visible and readable publicly throughout the whole cluster
  - Houses a single ConfigMap called cluster-info which houses the CA cert for the cluster (useful in some security bootstrapping scenarios).
- `kube-node-lease` - namespace stores a Lease object that is renewed by the node periodically which act as lightweight heartbeats for those nodes.

Try creating a namespace and listing the results:

```
user@ubuntu:~$ kubectl create namespace marketing

namespace/marketing created

user@ubuntu:~$ kubectl get ns

NAME              STATUS   AGE
```

```
default           Active   3h12m
kube-node-lease   Active   3h12m
kube-public       Active   3h12m
kube-system       Active   3h12m
marketing         Active   5s

user@ubuntu:~$
```

Try running a new pod and then display the pods in various namespaces:

```
user@ubuntu:~$ kubectl run --generator=run-pod/v1 myweb --image=nginx

pod/myweb created

user@ubuntu:~$
```

```
user@ubuntu:~$ kubectl get pod --namespace=kube-system

NAME                              READY   STATUS    RESTARTS   AGE
coredns-5644d7b6d9-b4rnz          1/1     Running   2          3h13m
coredns-5644d7b6d9-lxdqv          1/1     Running   2          3h13m
etcd-ubuntu                       1/1     Running   2          3h12m
kube-apiserver-ubuntu             1/1     Running   2          3h12m
kube-controller-manager-ubuntu    1/1     Running   2          3h12m
kube-proxy-npxks                  1/1     Running   2          3h13m
kube-scheduler-ubuntu             1/1     Running   2          3h12m
weave-net-rvhvk                   2/2     Running   6          177m

user@ubuntu:~$
```

```
user@ubuntu:~$ kubectl get pod --namespace=default

NAME    READY   STATUS    RESTARTS   AGE
myweb   1/1     Running   0          45s

user@ubuntu:~$
```

```
user@ubuntu:~$ kubectl get pod --all-namespaces

NAMESPACE     NAME                              READY   STATUS    RESTARTS   AGE
default       myweb                             1/1     Running   0          57s
kube-system   coredns-5644d7b6d9-b4rnz          1/1     Running   2          3h14m
kube-system   coredns-5644d7b6d9-lxdqv          1/1     Running   2          3h14m
kube-system   etcd-ubuntu                       1/1     Running   2          3h13m
kube-system   kube-apiserver-ubuntu             1/1     Running   2          3h13m
kube-system   kube-controller-manager-ubuntu    1/1     Running   2          3h12m
kube-system   kube-proxy-npxks                  1/1     Running   2          3h14m
kube-system   kube-scheduler-ubuntu             1/1     Running   2          3h13m
kube-system   weave-net-rvhvk                   2/2     Running   6          178m

user@ubuntu:~$
```

In the example we use the *--namespace* switch to display pods in namespaces "kube-system" and "default". We also used the *--all-namespaces* option to display all pods in the cluster.

You can issue any command in a particular namespace assuming you have access. Try creating the same pod in the new marketing namespace.

```
user@ubuntu:~$ kubectl run --generator=run-pod/v1 myweb --image=nginx --namespace=marketing

pod/myweb created
```

```
user@ubuntu:~$
```

```
user@ubuntu:~$ kubectl get pod --namespace=marketing

NAME    READY   STATUS    RESTARTS   AGE
myweb   1/1     Running   0          7s

user@ubuntu:~$
```

- How many pods are there in the marketing namespace?
- How many pods are there on the cluster?
- What are the names of all of the pods?
- Can multiple pods have the same name?
- What happens when you don't specify a namespace?

You can use `kubectl` to set your current namespace. Unless specified, default is always the current namespace. Display the current context with config view.

```
user@ubuntu:~$ kubectl config view

apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://192.168.228.157:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED

user@ubuntu:~$
```

Our context has no namespace set, making our current context "default". We can use *set-context* to change our active namespace.

Try it:

```
user@ubuntu:~$ kubectl config set-context kubernetes-admin@kubernetes --namespace=marketing

Context "kubernetes-admin@kubernetes" modified.

user@ubuntu:~$
```

```
user@ubuntu:~$ kubectl config view
```

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://192.168.228.157:6443
  name: kubernetes
contexts:
- context:
```

```
    cluster: kubernetes
    namespace: marketing
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

```
user@ubuntu:~$
```

Now to activate the context use the "use-context" command:

```
user@ubuntu:~$ kubectl config use-context kubernetes-admin@kubernetes

Switched to context "kubernetes-admin@kubernetes".

user@ubuntu:~$
```

Display your pods to verify that the marketing namespace is active.

```
user@ubuntu:~$ kubectl get pod

NAME    READY   STATUS    RESTARTS   AGE
myweb   1/1     Running   0          74s

user@ubuntu:~$ kubectl get pod --namespace=marketing

NAME    READY   STATUS    RESTARTS   AGE
myweb   1/1     Running   0          78s

user@ubuntu:~$ kubectl get pod --namespace=default

NAME    READY   STATUS    RESTARTS   AGE
myweb   1/1     Running   0          2m7s

user@ubuntu:~$
```

Note that events like other objects are partitioned by namespace. You can view events in the namespace you desire.

```
user@ubuntu:~$ kubectl get events --namespace=marketing | tail

LAST SEEN   TYPE     REASON      OBJECT      MESSAGE
104s        Normal   Scheduled   pod/myweb   Successfully assigned marketing/myweb to ubuntu
103s        Normal   Pulling     pod/myweb   Pulling image "nginx"
102s        Normal   Pulled      pod/myweb   Successfully pulled image "nginx"
102s        Normal   Created     pod/myweb   Created container myweb
101s        Normal   Started     pod/myweb   Started container myweb

user@ubuntu:~$
```

```
user@ubuntu:~$ kubectl get events --namespace=default | tail

116m        Normal    ScalingReplicaSet         deployment/website                  Scaled up
replica set website-769bf6f999 to 2
120m        Normal    ScalingReplicaSet         deployment/website                  Scaled
down replica set website-5577f87457 to 1
116m        Normal    ScalingReplicaSet         deployment/website                  Scaled up
```

```
replica set website-769bf6f999 to 3
120m        Normal    ScalingReplicaSet         deployment/website                    Scaled
down replica set website-5577f87457 to 0
119m        Normal    ScalingReplicaSet         deployment/website                    Scaled up
replica set website-5577f87457 to 1
119m        Normal    ScalingReplicaSet         deployment/website                    Scaled
down replica set website-769bf6f999 to 2
114m        Normal    ScalingReplicaSet         deployment/website                    (combined
from similar events): Scaled up replica set website-5577f87457 to 3
114m        Normal    ScalingReplicaSet         deployment/website                    Scaled up
replica set website-5577f87457 to 2
114m        Normal    ScalingReplicaSet         deployment/website                    Scaled
down replica set website-769bf6f999 to 1
114m        Normal    ScalingReplicaSet         deployment/website                    Scaled
down replica set website-769bf6f999 to 0

user@ubuntu:~$
```

2. Resource Quotas

A resource quota provides constraints that limit aggregate resource consumption per namespace. 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. Quotas 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.

Describe your new marketing namespace:

```
user@ubuntu:~$ kubectl describe namespace marketing

Name:         marketing
Labels:       <none>
Annotations:  <none>
Status:       Active

No resource quota.

No resource limits.

user@ubuntu:~$
```

Currently the marketing namespace is free of quotas and limits. Let's change that!

First, delete your pod(s) in the marketing namespace (the `-n` flag is shorthand for `--namespace`):

```
user@ubuntu:~$ kubectl delete pod myweb -n marketing

pod "myweb" deleted

user@ubuntu:~$
```

Even though your current context directs your requests to the marketing namespace it never hurts to be explicit!

Quotas can limit the sum of resources such as CPU, memory, and persistent and ephemeral storage; quotas can also limit counts of standard namespaced resource types in the format: `count/<resource>.<api-group>`. Some examples:

- `count/persistentvolumeclaims`
- `count/services`
- `count/secrets`
- `count/configmaps`
- `count/deployments.apps`
- `count/replicasets.apps`
- `count/statefulsets.apps`
- `count/jobs.batch`
- `count/cronjobs.batch`

Counts of objects are charged against a given quota when the object exists in etcd (whether or not is is actually deployed). Larg(er) objects such as secrets and configmaps can prevent controllers from spawning pods in large clusters, so limiting the numbers of them is a good idea.

Let's create a basic count quota which limits the number of pods in our new namespace to 2:

```
user@ubuntu:~$ mkdir ns && cd ns

user@ubuntu:~/ns$ nano pod-quota.yaml && cat pod-quota.yaml
```

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: pod-count
spec:
  hard:
    pods: "2"
```

```
user@ubuntu:~/ns$ kubectl apply -f pod-quota.yaml -n marketing

resourcequota/pod-count created

user@ubuntu:~/ns$
```

Describe your resource quota:

```
user@ubuntu:~/ns$ kubectl describe resourcequota pod-count

Name:       pod-count
Namespace:  marketing
Resource    Used  Hard
--------    ----  ----
pods        0     2

user@ubuntu:~/ns$
```

Our resource quota is in place; describe the marketing namespace once more:

```
user@ubuntu:~/ns$ kubectl describe ns marketing

Name:         marketing
Labels:       <none>
Annotations:  <none>
Status:       Active

Resource Quotas
 Name:     pod-count
 Resource  Used  Hard
 --------  ---   ---
 pods      0     2

No resource limits.

user@ubuntu:~/ns$
```

To test our quota, use the mydep deployment which has a replication factor of 3. As a reminder mydep looks like this:

```
user@ubuntu:~/ns$ cat ../dep/mydep.yaml
```

```
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: website
  labels:
    bu: sales
spec:
  replicas: 3
  selector:
    matchLabels:
      appname: webserver
      targetenv: demo
  template:
    metadata:
      labels:
        appname: webserver
        targetenv: demo
    spec:
      containers:
      - name: podweb
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

```
user@ubuntu:~/ns$
```

Create the deployment:

```
user@ubuntu:~/ns$ kubectl apply -f ../dep/mydep.yaml

deployment.apps/website created

user@ubuntu:~/ns$
```

What happened? Our deployment was successful, but did it deploy all the desired replicas?

Describe your namespace:

```
user@ubuntu:~/ns$ kubectl describe ns marketing

Name:         marketing
Labels:       <none>
Annotations:  <none>
Status:       Active

Resource Quotas
 Name:     pod-count
 Resource  Used  Hard
 --------  ---   ---
 pods      2     2

No resource limits.
```

List the objects in the marketing namespace:

```
user@ubuntu:~/ns$ kubectl get all -n marketing

NAME                           READY   STATUS    RESTARTS   AGE
pod/website-5577f87457-j6h87   1/1     Running   0          21s
pod/website-5577f87457-pllq8   1/1     Running   0          21s

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/website   2/3     2            2           21s

NAME                                 DESIRED   CURRENT   READY   AGE
```

```
replicaset.apps/website-5577f87457   3         2         2       21s

user@ubuntu:~/ns$
```

Examine the events for the marketing namespace:

```
user@ubuntu:~/ns$ kubectl get events -n marketing | tail

43s         Warning   FailedCreate        replicaset/website-5577f87457   Error creating: pods
"website-5577f87457-zm9xx" is forbidden: exceeded quota: pod-count, requested: pods=1, used:
pods=2, limited: pods=2
43s         Warning   FailedCreate        replicaset/website-5577f87457   Error creating: pods
"website-5577f87457-xc96h" is forbidden: exceeded quota: pod-count, requested: pods=1, used:
pods=2, limited: pods=2
43s         Warning   FailedCreate        replicaset/website-5577f87457   Error creating: pods
"website-5577f87457-n7psq" is forbidden: exceeded quota: pod-count, requested: pods=1, used:
pods=2, limited: pods=2
43s         Warning   FailedCreate        replicaset/website-5577f87457   Error creating: pods
"website-5577f87457-58ngt" is forbidden: exceeded quota: pod-count, requested: pods=1, used:
pods=2, limited: pods=2
43s         Warning   FailedCreate        replicaset/website-5577f87457   Error creating: pods
"website-5577f87457-k77rz" is forbidden: exceeded quota: pod-count, requested: pods=1, used:
pods=2, limited: pods=2
43s         Warning   FailedCreate        replicaset/website-5577f87457   Error creating: pods
"website-5577f87457-8cnl2" is forbidden: exceeded quota: pod-count, requested: pods=1, used:
pods=2, limited: pods=2
43s         Warning   FailedCreate        replicaset/website-5577f87457   Error creating: pods
"website-5577f87457-zdbxn" is forbidden: exceeded quota: pod-count, requested: pods=1, used:
pods=2, limited: pods=2
42s         Warning   FailedCreate        replicaset/website-5577f87457   Error creating: pods
"website-5577f87457-hf9xb" is forbidden: exceeded quota: pod-count, requested: pods=1, used:
pods=2, limited: pods=2
20s         Warning   FailedCreate        replicaset/website-5577f87457   (combined from similar
events): Error creating: pods "website-5577f87457-j5z89" is forbidden: exceeded quota: pod-
count, requested: pods=1, used: pods=2, limited: pods=2
43s         Normal    ScalingReplicaSet   deployment/website              Scaled up replica set
website-5577f87457 to 3

user@ubuntu:~/ns$
```

Our quota is working!

Remove the "website" deployment before moving on: `kubectl delete deploy website`.

3. Limit Ranges

If a namespace has a resource quota, it is helpful to have a default value in place for a limit. Here are two of the restrictions that a resource quota imposes on a namespace:

- Every container that runs in the namespace must have its own resource limits
- The total amount of resources used by all containers in the namespace must not exceed a specified limit

For example, if a container does not specify its own memory limit, it is given the default limit, and then it can be allowed to run in a namespace that is restricted by a quota.

Let's update our quota to allow more pods and add resource requests and limits:

```
user@ubuntu:~/ns$ cp pod-quota.yaml res-quota.yaml

user@ubuntu:~/ns$ nano res-quota.yaml

user@ubuntu:~/ns$ cat res-quota.yaml
```

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: pod-count
```

```
spec:
  hard:
    pods: "5"
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "1.5"
    limits.memory: 2Gi
```

```
user@ubuntu:~/ns$ kubectl apply -f res-quota.yaml

resourcequota/pod-count configured

user@ubuntu:~/ns$ kubectl describe namespace marketing

Name:         marketing
Labels:       <none>
Annotations:  <none>
Status:       Active

Resource Quotas
 Name:            pod-count
 Resource         Used  Hard
 --------         ---   ---
 limits.cpu       0     1500m
 limits.memory    0     2Gi
 pods             0     5
 requests.cpu     0     1
 requests.memory  0     1Gi

No resource limits.

user@ubuntu:~/ns$
```

Try creating a pod:

```
user@ubuntu:~/ns$ kubectl run --generator=run-pod/v1 myweb --image=nginx --namespace=marketing

Error from server (Forbidden): pods "myweb" is forbidden: failed quota: pod-count: must specify
limits.cpu,limits.memory,requests.cpu,requests.memory

user@ubuntu:~/ns$
```

The quota is working; pods have to specify requests and limits or the Kubernetes API rejects them.

Now we can create a LimitRange that provides default values for cpu and memory for all pods in the namespace:

```
user@ubuntu:~/ns$ nano limit-range.yaml && cat limit-range.yaml
```

```
apiVersion: v1
kind: LimitRange
metadata:
  name: marketing-limit
spec:
  limits:
  - default:
      cpu: .5
      memory: 256Mi
    defaultRequest:
      cpu: .25
      memory: 128Mi
    type: Container
```

```
user@ubuntu:~/ns$
```

Submit the limit to the Kubernetes API:

```
user@ubuntu:~/ns$ kubectl apply -f limit-range.yaml -n marketing

limitrange/marketing-limit created

user@ubuntu:~/ns$
```

Check that it was successful and describe your namespace to see how it has been affected:

```
user@ubuntu:~/ns$ kubectl get limitranges

NAME              CREATED AT
marketing-limit   2020-01-08T23:47:47Z

user@ubuntu:~/ns$ kubectl describe ns marketing

Name:         marketing
Labels:       <none>
Annotations:  <none>
Status:       Active

Resource Quotas
 Name:            pod-count
 Resource         Used  Hard
 --------         ---   ---
 limits.cpu       0     1500m
 limits.memory    0     2Gi
 pods             0     5
 requests.cpu     0     1
 requests.memory  0     1Gi

Resource Limits
 Type       Resource  Min  Max  Default Request  Default Limit  Max Limit/Request Ratio
 ----       --------  ---  ---  ---------------  -------------  -----------------------
 Container  cpu       -    -    250m             500m           -
 Container  memory    -    -    128Mi            256Mi          -

user@ubuntu:~/ns$
```

To test it out we can re-run our pod without values for memory requests/limits:

```
user@ubuntu:~/ns$ kubectl run --generator=run-pod/v1 myweb --image=nginx --namespace=marketing

pod/myweb created

user@ubuntu:~/ns$ kubectl describe pod myweb | grep -A5 Limits

    Limits:
      cpu:     500m
      memory:  256Mi
    Requests:
      cpu:        250m
      memory:     128Mi

user@ubuntu:~/ns$
```

Success! Now any pods made in the marketing namespace without resource requests/limits will receive the defaults.

Now create a pod that specifies requests/limits; we can use the `frontend` pod defined in limit.yaml. As a reminder, it looks like this:

```
user@ubuntu:~/ns$ cat ../pods/limit.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    resources:
      requests:
        memory: "64Mi"
        cpu: ".25"
      limits:
        memory: "128Mi"
        cpu: ".5"
  - name: wp
    image: wordpress
    resources:
      requests:
        memory: "64Mi"
        cpu: ".25"
      limits:
        memory: "128Mi"
        cpu: ".5"
```

```
user@ubuntu:~/ns$ kubectl apply -f ../pods/limit.yaml

pod/frontend created

user@ubuntu:~/ns$ kubectl describe pod frontend | grep -A5 Limits

    Limits:
      cpu:     500m
      memory:  128Mi
    Requests:
      cpu:        250m
      memory:     64Mi
--
    Limits:
      cpu:     500m
      memory:  128Mi
    Requests:
      cpu:        250m
      memory:     64Mi

user@ubuntu:~/ns$
```

Because the frontend pod specifies its own requests and limits, they are used instead of the defaults.

Before moving on, delete your resources, including the marketing namespace, and reset your config to use the default namespace:

`kubectl config set-context kubernetes-admin@kubernetes --namespace=default`

Admission Control

Admission controllers intercept authorized requests to the Kubernetes API server and then decide whether the request should be allowed, modified and then allowed or rejected. The built-in Kubernetes admission controllers include:

- AlwaysPullImages
- DefaultStorageClass
- DefaultTolerationSeconds
- EventRateLimit
- ExtendedResourceToleration
- ImagePolicyWebhook
- LimitPodHardAntiAffinityTopology

- LimitRanger
- MutatingAdmissionWebhook
- NamespaceAutoProvision
- NamespaceExists
- NamespaceLifecycle
- NodeRestriction
- OwnerReferencesPermissionEnforcement
- PodNodeSelector
- Configuration File Format
- PodPreset
- PodSecurityPolicy
- PodTolerationRestriction
- Priority
- ResourceQuota
- SecurityContextDeny
- ServiceAccount
- ValidatingAdmissionWebhook

Admission controllers are compiled into the kube-apiserver binary, and may only be configured by the cluster administrator. Admission controllers may be “validating”, “mutating”, or both. Mutating controllers may modify the objects they admit; validating controllers may not. If any of the controllers reject the request, the entire request is rejected immediately and an error is returned to the end-user.

4. Create secpol namespace

We'll create a new namespace for the remaining steps of this lab. Create a new namespace called "secpol":

```
user@ubuntu:~/ns$ kubectl create namespace secpol

namespace/secpol created

user@ubuntu:~/ns$ kubectl get ns

NAME              STATUS   AGE
default           Active   3h42m
kube-node-lease   Active   3h42m
kube-public       Active   3h42m
kube-system       Active   3h42m
secpol            Active   7s

user@ubuntu:~/ns$ kubectl describe ns secpol

Name:         secpol
Labels:       <none>
Annotations:  <none>
Status:       Active

No resource quota.

No resource limits.

user@ubuntu:~$
```

5. Create a Service Account

Permissions of any sort are generally defined in roles and imparted upon some security principal through a RoleBinding in Kubernetes. Create a service account to use with our upcoming security experiments:

```
user@ubuntu:~/ns$ kubectl create serviceaccount -n secpol poduser

serviceaccount/poduser created
```

```
uuser@ubuntu:~/ns$ kubectl get sa -n secpol

NAME      SECRETS   AGE
default   1         26s
poduser   1         5s

user@ubuntu:~/ns$ kubectl describe sa -n secpol

Name:                default
Namespace:           secpol
Labels:              <none>
Annotations:         <none>
Image pull secrets:  <none>
Mountable secrets:   default-token-248lq
Tokens:              default-token-248lq
Events:              <none>


Name:                poduser
Namespace:           secpol
Labels:              <none>
Annotations:         <none>
Image pull secrets:  <none>
Mountable secrets:   poduser-token-vphcd
Tokens:              poduser-token-vphcd
Events:              <none>

user@ubuntu:~/ns$
```

As you can see, creating a service account also creates a secret token which can be used to authenticate as that service account. Examine the secret:

```
user@ubuntu:~/ns$ kubectl get secret poduser-token-vphcd -n secpol

NAME                  TYPE                                  DATA   AGE
poduser-token-vphcd   kubernetes.io/service-account-token   3      73s

user@ubuntu:~/ns$ kubectl get secret poduser-token-vphcd -n secpol -o yaml

apiVersion: v1
data:
  ca.crt:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUN5RENDQWJDZ0F3SUJBZ0lCQURBTkJna3Foa2lHOXcwQkFRc0ZBREFW
TVJNd0VRWURWUVFERXdwcmRXSmwKY201bGRHVnpNQjRYRFRJd01ERXdPREl3TVRRMU0xb1hEVE13TURFd05USXdNVFExTTFv
d0ZURVRNQkVHQTFVRQpBeE1LYTNWaVpYSnVaWFJsY3pDQ0FTSXdEUVlKS29aSWh2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2dn
RUJBT3d4ClF2RkJUS2VMWjErNEdJUXF6VUF5NGtoY2EvZExUdkkvUG5lL1pNTCtVNS9LeGgwNEg2K3Zua0JtVXh6cW4vMVEK
T0hyaFBoY0dIek45VHhuWHNWOHU4anJUUFFMU1lkNVBGc05yRTBaakJHNzd2RlVNTEIyMjd6TjgzdkdORTB3YgpZdE05U05V
aEVkWkp5WHRxbnNPK1FoYTl2aDhYK016MkRsM1BkQ3pmeS9SY2ViM1dHeFk0bnpsejNvYVhrc3JsCkMyd0JzemdhdmJuZllY
cGppSFd0WXhWVC91RVdZUU9oUGQreFFDU0Vtcm5MS080Ti82bHpLa0VuLzJaWDJLS2sKNURjYUh3WlFYMmFpY3MvcHg2T0ky
NllrWWlWNXlBUU1pcmxsNXlTSDZXTzU5bTdtazRaOWJLZEZuR05MRk8rVApIK3E3djdYZG5CUjR0UFhCUXprQ0F3RUFBYU1q
TUNFd0RnWURWUjBQQVFIL0JBUURBZ0trTUE4R0ExVWRFd0VCCi93UUZNQU1CQWY4d0RRWUpLb1pJaHZjTkFRUxCUUFEZ2dF
QkFIQ1FoR0dPY2Urd1JRYVQ0N0RpTE90NStyNnUKa2ZsSTN6eWs1K3JvZng4YTBmRnZRZlJEcXEwc2JmTURLdTNhYnkyRmN1
Sk1QMElHOEdQWU40UHRPeXdKSGxiNgpTTWVQRytYdVJhSFFXb3ZSU3VGQ1I0RTlKR3lSTnZ4R0xxblErQ2FWdEhRK3FSYWh4
UnJWT2g2RmlvSDVqbFJLCk1EaWdnNWplYlVENVVDb0JJY0luVmFHMTcxM0NWTkpzd0FPQkRvbGNJeXpxeDdBTWxhS2hEdG9Q
N3lnMXozaTMKUFdWNnhLcUNTaTNXbzhFUHk1TkUvUFpYQXl1bXhnck5oT013R210L3VPZ2p0WVVrdUJwMWEzMUNad2RaYnFr
WgpJSDNxSEJodStBTFp6YmVab25WbUt6Z1VoMUdRT0hYRTFkRzI0YzUxaG1VN2x2WWpHL1Fkb0dwVk0zcz0KLS0tLS1FTkQg
Q0VSVElGSUNBVEUtLS0tLQo=
  namespace: c2VjcG9s
  token:
ZXlKaGJHY2lPaUpTVXpJMU5pSXNJbXRwWkNJNklqTTBiR3BQWTBoSVozaDNTM0pEV1hSS2R6VllUbWhPWVUwNE56WmtVV1V4
YVRGNVZIbFphbWRKVFZVaWZRLmV5SnBjM01pT2lKcmRXSmxjbTVsZEdWekwzTmxjblpwWTJWaFkyTnZkVzUwSWl3aWEzVmla
WEp1WlhSbGN5NXBieTl6WlhKMmFXTmxZV05qYjNWdWRDOXVZVzFsYzNCaFkyVWlPaUp6WldOd2Iyd2lMQ0pyZFdKbGNtNWxk
R1Z6TG1sdkwzTmxjblpwWTJWaFkyTnZkVzUwTDNObFkzSmxkQzV1WVcxbElqb2ljRzlrZFhObGNpMTBiMnRsYmkxMmNHaGpa
Q0lzSW10MVltVnlibVYwWlhNdWFXOHZjMlZ5ZG1salpXRmpZMjkxYm5RdmMyVnlkbWxqWlMxaFkyTnZkVzUwTG01aGJXVWlP
aUp3YjJSMWMyVnlJaXdpYTNWaVpYSnVaWFJsY3k1cGJ5OXpaWEoyYVdObFlXTmpiM1Z1ZEM5elpYSjJhV05sTFdGalkyOTFi
blF1ZFdsa0lqb2lOV0pqTmpObFltTXRNRFJoTlMwMFl6TXdMV0pqTmpjdFlqVTFNREJoTlRVMU1HVm1JaXdpYzNWaUlqb2lj
M2x6ZEdWdE9uTmxjblpwWTJWaFkyTnZkVzUwT25ObFkzQnZiRHB3YjJSMWMyVnlJbjAuWllTMHlyUWdhYTlpMVdXSFNnYXBC
UjZmN0RyOHN5WVZNVlNCZ2NIN0N3eTM5Y19KQVZOSzFCNFYtc2lDR2c0eVhHd1FoVnI3T0ZQNk5pZFBsSUd5OHg0amJvYnlQ
clJqUXp3cU5mYkkxamtzY1R2RjQ3eUowLWxXMi0xTGxMOGIyTEFjWlNEVFp5QTEtcjk4MXhWalpmdHNpeC01M05IUHo4ZkQw
```

```
R2M5TmprMldwZ2V6NTFBcUVpYXVReWc1UmhNemhNVFpRZnlxMkFERS1TWU15Wk0zdU05RkFzUmxUOXpmdkJkakdYaFRoM1Jl
LXplRm5VV2d2ekRBN1lGMU1sNmlmVzNRSnFwRmJBU3h0Yjg0QldTYmZWZHNxWU1IQ3RFUkFWbzcwQ2JhWFdwcndBdndiVnF4
dWc4U2NmblZnU2F3dnhfSFZNeW1qVUJERGhOZHlmVVNn
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: poduser
    kubernetes.io/service-account.uid: 5bc63ebc-04a5-4c30-bc67-b5500a5550ef
  creationTimestamp: "2020-01-08T23:58:32Z"
  name: poduser-token-vphcd
  namespace: secpol
  resourceVersion: "14270"
  selfLink: /api/v1/namespaces/secpol/secrets/poduser-token-vphcd
  uid: eb731b41-67f1-4f6a-a9f7-9333b78c1d53
type: kubernetes.io/service-account-token

user@ubuntu:~/ns$
```

6. Working with the PodSecurityPolicy Admission Controller

Pod security policy control is implemented through the optional (but recommended) admission controller PodSecurityPolicy. Policies are created as regular Kubernetes resources and enforced by enabling the admission controller. PodSecurityPolicy is a white list style controller, so if enabled without any policies, it will prevent *any* pods from being created in the cluster.

Policies are associated with Kubernetes security principals, such as service accounts. This allows administrators to create different policies for different users, groups, pods, etc.

Most Kubernetes pods are not created directly by users. Instead, they are typically created indirectly as part of a Deployment, ReplicaSet, or other templated controller via the controller manager. Granting the controller access to a policy would grant access for all pods created by that the controller, so the preferred method for authorizing policies is to configure a service account for the pod and to grant access to the service account.

List the parameters used to start your api-server:

```
user@ubuntu:~/ns$ ps -ef -ww | grep kube-apiserver | sed  "s/--/\n--/g"

root       3823   3736  1 14:45 ?        00:01:35 kube-apiserver
--advertise-address=192.168.228.157
--allow-privileged=true
--authorization-mode=Node,RBAC
--client-ca-file=/etc/kubernetes/pki/ca.crt
--enable-admission-plugins=NodeRestriction
--enable-bootstrap-token-auth=true
--etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
--etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
--etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
--etcd-servers=https://127.0.0.1:2379
--insecure-port=0
--kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
--kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
--kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
--proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
--proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key
--requestheader-allowed-names=front-proxy-client
--requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt
--requestheader-extra-headers-prefix=X-Remote-Extra-
--requestheader-group-headers=X-Remote-Group
--requestheader-username-headers=X-Remote-User
--secure-port=6443
--service-account-key-file=/etc/kubernetes/pki/sa.pub
--service-cluster-ip-range=10.96.0.0/12
--tls-cert-file=/etc/kubernetes/pki/apiserver.crt
--tls-private-key-file=/etc/kubernetes/pki/apiserver.key
user      45335   2377  0 16:05 pts/4    00:00:00 grep
--color=auto kube-apiserver

user@ubuntu:~/ns$
```

The relevant line above is: `--enable-admission-plugins=NodeRestriction`

This enables the NodeRestriction admission controller only. The NodeRestriction admission controller limits the Node and Pod objects a kubelet can modify. In order to be limited by this admission controller, kubelets use credentials in the system:nodes group, with a username in the form system:node:. Such kubelets will only be allowed to modify their own Node API object, and only modify Pod API objects that are bound to their node.

Display the manifest that the kubelet uses to create the api-server:

```
user@ubuntu:~/ns$ sudo cat /etc/kubernetes/manifests/kube-apiserver.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.228.157
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
    - --insecure-port=0
    - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
    - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
    - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
    - --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
    - --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key
    - --requestheader-allowed-names=front-proxy-client
    - --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt
    - --requestheader-extra-headers-prefix=X-Remote-Extra-
    - --requestheader-group-headers=X-Remote-Group
    - --requestheader-username-headers=X-Remote-User
    - --secure-port=6443
    - --service-account-key-file=/etc/kubernetes/pki/sa.pub
    - --service-cluster-ip-range=10.96.0.0/12
    - --tls-cert-file=/etc/kubernetes/pki/apiserver.crt
    - --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
    image: k8s.gcr.io/kube-apiserver:v1.16.4

...
```

```
user@ubuntu:~/ns$
```

The listing shows the `spec.containers.command` field setting the `--enable-admission-plugins=NodeRestriction` parameter. To enable the PodSecurityPolicy admission controller [AC] we will need to append it to the list. Edit the manifest so that the PodSecurityPolicy AC is enabled. *Do not* try to edit the file in place as kubelet has a bug that will deploy the temp file; copy the file to your home directory, edit it and copy the edited file back into the `/etc/kubernetes/manifests` path:

```
user@ubuntu:~/ns$ mkdir ~/secpol && cd ~/secpol
```

```
user@ubuntu:~/secpol$ sudo cp /etc/kubernetes/manifests/kube-apiserver.yaml .

user@ubuntu:~/secpol$ sudo nano kube-apiserver.yaml

user@ubuntu:~/secpol$ sudo cat kube-apiserver.yaml | head -18
```

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.228.157
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction,PodSecurityPolicy

...
```

```
user@ubuntu:~/secpol$ sudo cp kube-apiserver.yaml /etc/kubernetes/manifests/

user@ubuntu:~/secpol$
```

The kubelet will see the change in its next update cycle and replace the api-server pod as specified.

Run the command below until you see the new api-server running with the additional AC:

```
user@ubuntu:~/secpol$ ps -ef -ww | grep kube-apiserver | sed  "s/--/\n--/g" | grep admission

--enable-admission-plugins=NodeRestriction,PodSecurityPolicy

user@ubuntu:~/secpol$
```

7. Creating a pod security policy

Now that we have the admission controller configured let's test the pod security policy feature.

To begin create a simple pod security policy:

```
user@ubuntu:~/secpol$ nano podsec.yaml

user@ubuntu:~/secpol$ cat podsec.yaml
```

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: example
spec:
  privileged: false  # Don't allow privileged pods
  # Set required fields with defaults
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  runAsUser:
```

```
  rule: RunAsAny
fsGroup:
  rule: RunAsAny
volumes:
- '*'
```

```
user@ubuntu:~$
```

This policy allows anything except privileged pods (a good policy to consider in your own cluster!).

Create the policy:

```
user@ubuntu:~/secpol$ kubectl create -n secpol -f podsec.yaml

podsecuritypolicy.policy/example created

user@ubuntu:~/secpol$
```

8. Using a pod security policy

To begin we will give our service account the ability to create resources of all types by binding it to the predefined clusterrole "edit". Display the capabilities of the edit role:

```
user@ubuntu:~/secpol$ kubectl get clusterrole edit

NAME   AGE
edit   3h55m

user@ubuntu:~/secpol$ kubectl describe clusterrole edit

Name:         edit
Labels:       kubernetes.io/bootstrapping=rbac-defaults
              rbac.authorization.k8s.io/aggregate-to-admin=true
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
  Resources                                  Non-Resource URLs  Resource Names  Verbs
  ---------                                  -----------------  --------------  -----
  configmaps                                 []                 []              [create delete
deletecollection patch update get list watch]
  endpoints                                  []                 []              [create delete
deletecollection patch update get list watch]
  persistentvolumeclaims                     []                 []              [create delete
deletecollection patch update get list watch]
  pods                                       []                 []

...
```

Because this is a clusterrole it applies to all namespaces.

Now bind the edit role to the poduser service account:

```
user@ubuntu:~/secpol$ kubectl create rolebinding -n secpol cledit \
--clusterrole=edit --serviceaccount=secpol:poduser

rolebinding.rbac.authorization.k8s.io/cledit created

user@ubuntu:~/secpol$
```

Now create a simple test pod manifest:

```
user@ubuntu:~/secpol$ nano pod.yaml
```

```
user@ubuntu:~/secpol$ cat pod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: secpol
spec:
  containers:
    - name:  secpol
      image: nginx
```

```
user@ubuntu:~$
```

Next see if you can create the pod in the secpol namespace using the service account identity:

```
user@ubuntu:~/secpol$ kubectl --as=system:serviceaccount:secpol:poduser -n secpol apply -f
pod.yaml

Error from server (Forbidden): error when creating "pod.yaml": pods "secpol" is forbidden:
unable to validate against any pod security policy: []

user@ubuntu:~/secpol$
```

As you can see we are not authorized on any policy that allows the creation of this pod. Even though we have RBAC permission to create the pod, the admission controller overrides RBAC.

Check to see if you have access to the example policy created above:

```
user@ubuntu:~/secpol$ kubectl --as=system:serviceaccount:secpol:poduser \
-n secpol auth can-i use podsecuritypolicy/example

Warning: resource 'podsecuritypolicies' is not namespace scoped in group 'policy'
no

user@ubuntu:~/secpol$
```

We need to attach the policy to our SA. First create a role with "use" access to the example policy. First create a yaml file for the role with "list" access. Then modify the role's yaml file from "list" access to "use" access.

```
user@ubuntu:~/secpol$ kubectl create role psp:unprivileged -n secpol \
--verb=list --resource=podsecuritypolicy --resource-name=example -o yaml --dry-run >> psp.yaml

user@ubuntu:~/secpol$ nano psp.yaml && cat psp.yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  creationTimestamp: null
  name: psp:unprivileged
rules:
- apiGroups:
  - policy
  resourceNames:
  - example
  resources:
  - podsecuritypolicies
  verbs:
  - use
```

```
user@ubuntu:~/secpol$ kubectl apply -f psp.yaml -n secpol

role.rbac.authorization.k8s.io/psp:unprivileged created

user@ubuntu:~/secpol$
```

Now bind the role to the SA:

```
user@ubuntu:~/secpol$ kubectl create rolebinding poduserpol -n secpol \
--role=psp:unprivileged --serviceaccount=secpol:poduser

rolebinding.rbac.authorization.k8s.io/poduserpol created

user@ubuntu:~/secpol$
```

Now retry checking your policy permissions:

```
user@ubuntu:~/secpol$ kubectl --as=system:serviceaccount:secpol:poduser \
-n secpol auth can-i use podsecuritypolicy/example

Warning: resource 'podsecuritypolicies' is not namespace scoped in group 'policy'
yes

user@ubuntu:~/secpol$
```

Great, now try to create the pod again:

```
user@ubuntu:~/secpol$ kubectl --as=system:serviceaccount:secpol:poduser \
-n secpol apply -f pod.yaml

pod/secpol created

user@ubuntu:~/secpol$ kubectl get po -n secpol

NAME     READY   STATUS    RESTARTS   AGE
secpol   1/1     Running   0          27s

user@ubuntu:~/secpol$
```

Perfect!

9. Policies in action

Now we'll try to create a pod that violates the policy, a pod that requests privileged execution.

```
user@ubuntu:~/secpol$ nano priv.yaml && cat priv.yaml
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: privileged
spec:
  containers:
    - name: priv
      image: nginx
      securityContext:
        privileged: true
```

```
user@ubuntu:~/secpol$ kubectl --as=system:serviceaccount:secpol:poduser -n secpol apply -f
priv.yaml
```

```
Error from server (Forbidden): error when creating "priv.yaml": pods "privileged" is forbidden:
unable to validate against any pod security policy:
[spec.containers[0].securityContext.privileged: Invalid value: true: Privileged containers are
not allowed]

user@ubuntu:~/secpol$
```

As expected the admission controller denies us the ability to create a privileged container.

10. Cleanup

To revert your cluster back to the base state without PodSecurityPolicy, edit the `kube-apiserver.yaml` manifest and revert the change made to `--enable-admission-plugins`:

```
user@ubuntu:~/secpol$ sudo nano kube-apiserver.yaml

user@ubuntu:~/secpol$ sudo cat kube-apiserver.yaml | head -18
```

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.228.157
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
```

```
user@ubuntu:~/secpol$ sudo cp kube-apiserver.yaml /etc/kubernetes/manifests/

user@ubuntu:~/secpol$
```

Reverting this change is absolutely important! Keeping the PodSecurityPolicy plugin in place may prevent you from proceeding with the rest of the labs!

Then, delete the secpol namespace, which will remove all other resources deployed within it:

```
user@ubuntu:~/secpol$ kubectl delete ns secpol

namespace "secpol" deleted

user@ubuntu:~/secpol$ cd ~

user@ubuntu:~$
```

Congratulations, you have completed the lab!