RX-M Cloud Native Consulting

Kubernetes

Lab 7 – Namespaces and Patching

In Kubernetes, Namespaces are a mechanism to partition resources created by users into a logically named group. 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.

Another useful feature of Kubernetes is its support for cluster information gathering. Kubernetes Dashboard can be used to “introspect” the cluster, allowing you to see what resources are deployed, monitor node utilization, look for error messages and more. Kubernetes Dashboard is available here:

- Dashboard - https://github.com/kubernetes/dashboard

Kubernetes Dashboard is a general purpose, web-based UI for Kubernetes clusters. It allows users to manage applications running in the cluster and troubleshoot them, as well as manage the cluster itself.

1. Working with Namespaces

In Kubernetes almost all operating state can be created using a configuration file. Namespaces are, for many purposes, just like other objects. We can *“get”* a list of them, *“describe”* one in detail and *“create”* new ones.

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

```
ubuntu@ip-10-0-2-200:~/vol$ cd ~
```

```
ubuntu@ip-10-0-2-200:~$ kubectl get namespaces

NAME          STATUS    AGE
default       Active    8h
kube-public   Active    8h
kube-system   Active    8h
ubuntu@ip-10-0-2-200:~$
```

```
ubuntu@ip-10-0-2-200:~$ kubectl describe namespace default

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

No resource quota.

No resource limits.
ubuntu@ip-10-0-2-200:~$
```

Our system has two namespaces, the *default* namespace and the *kube-system* namespace. Even very basic deployments of Kubernetes usually make use of a system namespace (called kube-system) to run cluster centric pods. The default namespace is free of quotas and limits.

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.

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 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.

Try creating a namespace with the following config:

```
ubuntu@ip-10-0-2-200:~$ mkdir ns

ubuntu@ip-10-0-2-200:~$ cd ns

ubuntu@ip-10-0-2-200:~/ns$ vim ns.yaml

ubuntu@ip-10-0-2-200:~/ns$ cat ns.yaml

apiVersion: v1
kind: Namespace
metadata:
  name: marketing
ubuntu@ip-10-0-2-200:~/ns$
```

Try creating the namespace and listing the results:

```
ubuntu@ip-10-0-2-200:~/ns$ kubectl create -f ns.yaml

namespace/marketing created
ubuntu@ip-10-0-2-200:~/ns$
```

```
ubuntu@ip-10-0-2-200:~/ns$ kubectl get ns

NAME          STATUS    AGE
default       Active    8h
kube-public   Active    8h
kube-system   Active    8h
marketing     Active    5s
ubuntu@ip-10-0-2-200:~/ns$
```

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

```
ubuntu@ip-10-0-2-200:~/ns$ kubectl run --generator=run-pod/v1 myweb --image=nginx

pod/myweb created
ubuntu@ip-10-0-2-200:~/ns$
```

```
ubuntu@ip-10-0-2-200:~/ns$ kubectl get pod --namespace=kube-system

NAMESPACE     NAME                                    READY   STATUS    RESTARTS
AGE
kube-system   coredns-576cbf47c7-7qlht                1/1     Running   0
14h
kube-system   coredns-576cbf47c7-qp7qk                1/1     Running   0
14h
kube-system   etcd-ip-10-0-2-200                      1/1     Running   0
14h
kube-system   kube-apiserver-ip-10-0-2-200            1/1     Running   0
14h
kube-system   kube-controller-manager-ip-10-0-2-200   1/1     Running   0
14h
kube-system   kube-proxy-rmxrk                        1/1     Running   0
14h
kube-system   kube-scheduler-ip-10-0-2-200            1/1     Running   0
14h
kube-system   weave-net-4xsgf                         2/2     Running   0
12h
ubuntu@ip-10-0-2-200:~/ns$
```

```
ubuntu@ip-10-0-2-200:~/ns$ kubectl get pod --namespace=default

NAME    READY   STATUS    RESTARTS   AGE
myweb   1/1     Running   0          6s
ubuntu@ip-10-0-2-200:~/ns$
```

```
ubuntu@ip-10-0-2-200:~/ns$ kubectl get pod --all-namespaces

NAMESPACE     NAME                                    READY   STATUS    RESTARTS
```

```
AGE
default       myweb                                   1/1     Running   0
36s
kube-system   coredns-576cbf47c7-7qlht                1/1     Running   0
14h
kube-system   coredns-576cbf47c7-qp7qk                1/1     Running   0
14h
kube-system   etcd-ip-10-0-2-200                      1/1     Running   0
14h
kube-system   kube-apiserver-ip-10-0-2-200            1/1     Running   0
14h
kube-system   kube-controller-manager-ip-10-0-2-200   1/1     Running   0
14h
kube-system   kube-proxy-rmxrk                        1/1     Running   0
14h
kube-system   kube-scheduler-ip-10-0-2-200            1/1     Running   0
14h
kube-system   weave-net-4xsgf                         2/2     Running   0
12h
ubuntu@ip-10-0-2-200:~/ns$
```

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.

```
ubuntu@ip-10-0-2-200:~/ns$ kubectl run --generator=run-pod/v1 myweb --image=nginx
--namespace=marketing

pod/myweb created
ubuntu@ip-10-0-2-200:~/ns$
```

```
ubuntu@ip-10-0-2-200:~/ns$ kubectl get pod --namespace=marketing

NAME    READY   STATUS    RESTARTS   AGE
myweb   1/1     Running   0          17s
ubuntu@ip-10-0-2-200:~/ns$
```

- 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.

```
ubuntu@ip-10-0-2-200:~/ns$ kubectl config view
```

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://10.0.2.200:6443
  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
ubuntu@ip-10-0-2-200:~/ns$
```

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

Try it:

```
ubuntu@ip-10-0-2-200:~/ns$ kubectl config set-context kubernetes-admin@kubernetes
--namespace=marketing

Context "kubernetes-admin@kubernetes" modified.
ubuntu@ip-10-0-2-200:~/ns$
```

```
ubuntu@ip-10-0-2-200:~/ns$ kubectl config view

apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://10.0.2.200:6443
  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
```

```
ubuntu@ip-10-0-2-200:~/ns$
```

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

```
ubuntu@ip-10-0-2-200:~/ns$ kubectl config use-context kubernetes-admin@kubernetes

Switched to context "kubernetes-admin@kubernetes".
ubuntu@ip-10-0-2-200:~/ns$
```

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

```
ubuntu@ip-10-0-2-200:~/ns$ kubectl get pod

NAME    READY   STATUS    RESTARTS   AGE
myweb   1/1     Running   0          66s

ubuntu@ip-10-0-2-200:~/ns$ kubectl get pod --namespace=marketing

NAME    READY   STATUS    RESTARTS   AGE
myweb   1/1     Running   0          66s

ubuntu@ip-10-0-2-200:~/ns$ kubectl get pod --namespace=default

NAME    READY   STATUS    RESTARTS   AGE
myweb   1/1     Running   0          3m25s
ubuntu@ip-10-0-2-200:~/ns$
```

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

```
ubuntu@ip-10-0-2-200:~/ns$ kubectl get events --namespace=marketing | tail

'LAST SEEN   TYPE     REASON      KIND   MESSAGE
5m57s       Normal   Scheduled   Pod    Successfully assigned marketing/myweb to
ip-10-0-2-200
5m56s       Normal   Pulling     Pod    pulling image "nginx"
5m54s       Normal   Pulled      Pod    Successfully pulled image "nginx"
5m54s       Normal   Created     Pod    Created container
5m54s       Normal   Started     Pod    Started container
ubuntu@ip-10-0-2-200:~/ns$
```

```
ubuntu@ip-10-0-2-200:~/ns$ kubectl get events --namespace=default | tail

4m23s       Normal   Created          Pod    Created container
4m23s       Normal   Started          Pod    Started container
7m48s       Normal   Killing          Pod    Killing container with id
docker://redis:Need to kill Pod
7m18s       Normal   Killing          Pod    Killing container with id
docker://shell:Need to kill Pod
55m         Normal   Scheduled        Pod    Successfully assigned default/prod-
```

```
db-client-pod to ip-10-0-2-200
55m         Normal   Pulling          Pod    pulling image "nginx"
55m         Normal   Pulled           Pod    Successfully pulled image "nginx"
55m         Normal   Created          Pod    Created container
55m         Normal   Started          Pod    Started container
4m42s       Normal   Killing          Pod    Killing container with id
docker://db-client-container:Need to kill Pod
ubuntu@ip-10-0-2-200:~/ns$
```

Reset your config to use the default namespace:

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

2. Patching

Using the `mydep.yaml` Deployment spec from lab 5, create the Deployment once more using `kubectl create -f mydep.yaml`. As a reminder the `mydep` Deployment looks like this:

```
ubuntu@ip-10-0-2-200:~/ns$ cd ../dep/

ubuntu@ip-10-0-2-200:~/dep$

ubuntu@ip-10-0-2-200:~/dep$ cat 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
ubuntu@ip-10-0-2-200:~/dep$
```

Imagine we have special concerns about one of our pods, perhaps it is producing intermittent errors. We can filter out the other pods by giving the problem pod a special label. The patch command comes in handy here. Use the `patch` subcommand to give one of your pods a new label and then verify that it was set correctly:

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get pods

NAME                        READY   STATUS    RESTARTS   AGE
myweb                       1/1     Running   0          6m
website-6dc99878b-8xklc     1/1     Running   0          8s
website-6dc99878b-bkhb6     1/1     Running   0          8s
website-6dc99878b-vfc9z     1/1     Running   0          8s
ubuntu@ip-10-0-2-200:~/dep$
```

Save one of the pod names to a variable:

```
ubuntu@ip-10-0-2-200:~/dep$ POD=$(kubectl get pod -o name |tail -1) && echo $POD

website-6dc99878b-vfc9z

ubuntu@ip-10-0-2-200:~/dep$ kubectl patch $POD \
-p '{"metadata": {"labels": {"monitor": "problem", "appname": "webserver",
"targetenv":"demo" } } }'

pod/website-6dc99878b-vfc9z patched
ubuntu@ip-10-0-2-200:~/dep$
```

Now display the pods labels:

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get $POD -o json | jq .metadata.labels

{
  "appname": "webserver",
  "monitor": "problem",
  "pod-template-hash": "6dc99878b",
  "targetenv": "demo"
}
ubuntu@ip-10-0-2-200:~/dep$
```

Our new label: `monitor=problem` is in place. Now we can quickly view any of our problem pods using the new label:

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get pod -l "monitor in (problem, error)"

NAME                        READY   STATUS    RESTARTS   AGE
website-6dc99878b-vfc9z     1/1     Running   0          3m41s
ubuntu@ip-10-0-2-200:~/dep$
```

Try viewing just the pods with webserver as the value for the appname key:

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get pod -l "appname in (webserver)"

NAME                        READY   STATUS    RESTARTS   AGE
```

```
website-6dc99878b-8xklc   1/1     Running   0          3m59s
website-6dc99878b-bkhb6   1/1     Running   0          3m59s
website-6dc99878b-vfc9z   1/1     Running   0          3m59s
ubuntu@ip-10-0-2-200:~/dep$
```

Now imagine we have decided that the problem in this pod is nginx.

- `curl` the pod IP (e.g. `curl http://10.32.0.4`), you should get the nginx response
- Use the `patch` subcommand to change the image for the container in the problem pod to “httpd” and verify your work.
- `curl` the pod IP again, verify that you get the (“it works”) apache response now

Hint: You can use the *-o json* switch to display the pod spec in JSON if you need a template to start from (e.g. `kubectl get pod website-bbcdd544d-bhj6f -o json`.)

3. Modifying Node properties

Imagine we would like to do some work on this node and do not want users to schedule pods on it for a while. You can make a node unschedulable by setting the unschedulable property to true.

Try making the node unschedulable:

```
ubuntu@ip-10-0-2-200:~/dep$ cd

ubuntu@ip-10-0-2-200:~$ kubectl patch $(kubectl get node -o name) -p '{"spec":
{"unschedulable":true}}'

node/ip-10-0-2-200 patched
ubuntu@ip-10-0-2-200:~$
```

Now try to run a pod, or reusing `limits.yaml` aka frontend.

```
ubuntu@ip-10-0-2-200:~$ kubectl create -f pods/limit.yaml

pod/frontend created
ubuntu@ip-10-0-2-200:~$
```

The pod appears to have been created but in reality only the target state has been created. The scheduler will assess the config asynchronously. When it does it will have a problem. Display the pod and node status:

```
ubuntu@ip-10-0-2-200:~$ kubectl get pods

NAME                      READY   STATUS    RESTARTS   AGE
frontend                  0/2     Pending   0          16s
ubuntu@ip-10-0-2-200:~$
```

```
ubuntu@ip-10-0-2-200:~$ kubectl describe pod frontend |tail -1

  Warning  FailedScheduling  7s (x7 over 55s)  default-scheduler  0/1 nodes are
available: 1 node(s) were unschedulable.
ubuntu@ip-10-0-2-200:~$
```

As you can see there are no nodes available.

Repatch your node to make it schedulable again.

```
ubuntu@ip-10-0-2-200:~$ kubectl get $(kubectl get node -o name) -o json  | jq
.spec.unschedulable

true
ubuntu@ip-10-0-2-200:~$
```

```
ubuntu@ip-10-0-2-200:~$ kubectl patch $(kubectl get node -o name) -p '{"spec":
{"unschedulable":false}}'

node/ip-10-0-2-200 patched
ubuntu@ip-10-0-2-200:~$
```

```
ubuntu@ip-10-0-2-200:~$ kubectl get $(kubectl get node -o name) -o json  | jq
.spec.unschedulable

null
ubuntu@ip-10-0-2-200:~$
```

Test it by deleting and running a pod, or wait for the previous pod to be redeployed.

When you are finished experimenting, delete all of the deployments and pods on your system.

```
ubuntu@ip-10-0-2-200:~$ kubectl get pods

NAME                      READY   STATUS    RESTARTS   AGE
frontend                  1/2     Running   0          103s
ubuntu@ip-10-0-2-200:~$
```

```
ubuntu@ip-10-0-2-200:~$ kubectl delete pod frontend

pod "frontend" deleted
ubuntu@ip-10-0-2-200:~$
```

```
ubuntu@ip-10-0-2-200:~$ kubectl get pods

No resources found.
ubuntu@ip-10-0-2-200:~$
```

Congratulations you have completed the lab!

Selected Solutions

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl patch $(kubectl get pod -o name |tail -1) \
-p '{"spec": {"containers": [ {"name": "podweb", "image": "httpd"} ] } }'

pod/website-6dc99878b-vfc9z patched

ubuntu@ip-10-0-2-200:~/dep$ kubectl get pod -o wide

NAME                       READY   STATUS    RESTARTS   AGE     IP          NODE
NOMINATED NODE
myweb                      1/1     Running   0          13m     10.32.0.4   ip-10-
0-2-200   <none>
website-6dc99878b-8xklc    1/1     Running   0          7m11s   10.32.0.8   ip-10-
0-2-200   <none>
website-6dc99878b-bkhb6    1/1     Running   0          7m11s   10.32.0.7   ip-10-
0-2-200   <none>
website-6dc99878b-vfc9z    1/1     Running   1          7m11s   10.32.0.6   ip-10-
0-2-200   <none>

ubuntu@ip-10-0-2-200:~/dep$ curl 10.32.0.6

<html><body><h1>It works!</h1></body></html>
ubuntu@ip-10-0-2-200:~/dep$
```