Kubernetes - CKAD (Pod basics & design)

MSc. Carlos Avendaño Arango

Resources

Github Repository

https://github.com/carloselpapa10/kubernetes.git

Roadmap

Namespaces

Pods

Label Selectors

Annotations

Exercises

Namespaces

Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called *namespaces*.

Namespaces are intended for use in environments with many users spread across multiple teams, or projects. For cluster with a few to tens of users, you SHOULD NOT need to create or think about namespaces.

Namespaces are a way to divide cluster resources between multiple users.

Namespaces

Create a new namespace called ‘myspace’.
$ **kubectl create namespace myspace**

Show all namespaces
$ **kubectl get namespaces**

Create a Pod that belongs to the namespace ‘myspace’
$ **kubectl run nginx --image=nginx --restart=Never --namespace=myspace**

Get pods into the namespace ‘myspace’
$ **kubectl get pods -n myspace**

Trick: To avoid passing the target namespace in your kubectl commands, change your context with the following command.
$ **kubectl config set-context --current --namespace=myspace**

Pods

Pods represent a *Logical Application*.

“Generally, if you have multiple containers with a hard dependency on each other, they would be packaged together inside of a single pod”.

Pods provide *two* kinds of shared resources for their constituent containers: **Networking** and **Storages**.

Pods - Networking

Each Pod is assigned a unique IP address. Every container in a Pod shares the network namespace, including the API address and network ports. Containers inside a Pod can communicate with another using *localhost*. When containers in a Pod communicate with entities outside the Pod, they must coordinate how they use the shared network resources (such as ports).

Pods - Storage

A Pod can specify a set of shared storage volumes. All containers in the Pod can access the shared volumes, allowing those containers to share data. Volumes also allow persistent data in a Pod to survive in case one of the containers within a Pod needs to be restarted.

Note: *Restarting a container in a Pod should not be confused with restarting the Pod. The Pod itself does not run, but is an environment the containers run in and persist until it is deleted.*

Pod lifetime

In general, Pods do not disappear until someone destroys them. This may be a human or a Controller. The only exception to this rule is that Pods with a phase of *Succeeded* or *Failed* for more than some duration will expire and be automatically destroyed.

Pod phases: *Pending, Running, Succeeded, Failure, Unknown.*

Container states: *Waiting, Running, Terminating.*

Restart Policy: *Always, OnFailure, Never.*

Pods - Example states

Issue	Event	Restart Policy	Activity / Pod phase
Pod is Running and has one container exits with success.	Log completion event	Always	Restart container / Running
		OnFailure	Nothing / Succeeded
		Never	Nothing / Succeeded
Pod is Running and has one container. Container exits with failure.	Log failure event	Always	Restart container / Running
		OnFailure	Restart container / Running
		Never	Nothing / Failed

Pods - Example states

Issue	Event	Restart Policy	Activity / Pod phase
Pod is Running and has 2 containers. One exits with failure.	Log failure event	Always	Restart container / Running
		OnFailure	Restart container / Running
		Never	Nothing / Running
Pod is Running and has 2 containers. If one container is not running and container 2 exits (with failure)	Log failure event	Always	Restart container / Running
		OnFailure	Restart container / Running
		Never	Nothing / Failed

Pods - Example states

Issue	Event	Restart Policy	Activity / Pod phase
Pod is Running and has one container. Container runs out of memory.	Container terminates in failure / Log OOM event	Always	Restart container / Running
		OnFailure	Restart container / Running
		Never	Nothing / Failed

Label selectors

The API currently supports 2 types of selectors: *equality-based* and *set-based*.

A label selector can be made of multiple requirements which are comma-separated. In the case of multiple requirements, all must be satisfied so the comma separator acts as a logical AND (&&) operator.

Label selectors - Equality-based requirements

3 kinds of operator are admitted **‘=’, ‘==’, ‘!=’** (i.e. environment=production; tier != frontend)

The sample Pod below selects nodes with the label “accelerator=nvidia-tesla-p100”

```
apiVersion: v1
kind: Pod
metadata:
  name: cuda-test
spec:
  containers:
    -    name: cuda-test
         image: demo
         nodeSelector:

           accelerator: nvidia-tesla-p100
```

Label selectors - Set-based requirements

Set-based requirements allow filtering keys according to a set of values. 3 kinds of operators are supported (i.e. **in, notin and exists**)

Samples:

- environment in (production, qa)
- tier notin (frontend, backend)
- partition
- ! partition

Annotations

You can use either *labels* or *annotations* to attach metadata to kubernetes objects. Clients such as tools and libraries can retrieve this metadata.

Labels can be used to select objects and to find collections of objects that satisfy certain conditions. In contrast, *annotation* can be small or large, structured or unstructured, and can include characters not permitted by labels.

Annotations - Use cases

- Fields managed by a declarative configuration layer.
- Build, release or image information like timestamps, release IDs, git branch, PR numbers, image hashes, and registry address.
- Pointers to logging, monitoring, analytics, ot audit repositories.
- Client library or tool information that can be used for debugging purposes: for example: name, version, and build information.

Annotations - Example

```
apiVersion: v1
kind: Pod
metadata:
  name: demo
  annotations:
    imageRegistry: “https://hub.docker.com/”
spec:
  containers:
    -       name: nginx
            image: nginx
            port:
              - containerPort: 80
```

Exercise 1

Create a namespace called ‘mynamespace’ and a pod with image nginx called nginx on this namespace.

Solution

$ **kubectl create namespace mynamespace**
$ **kubectl run nginx --image=nginx --restart=Never -n mynamespace**

Exercise 2

Create the Pod that was just described using YAML.

Solution

$ **kubectl run nginx --image=nginx --restart=Never --dry-run --output yaml > lab-pod-basics/exercise-2.yaml**
$ **kubectl apply -f lab-pod-basics/exercise-2.yaml -n mynamespace**

Alternatively we can run in one line:
$ **kubectl run nginx --image=nginx --restart=Never --dry-run --output yaml | kubectl create -n mynamespace -f -**

Exercise 3

Create a busybox pod (using kubectl command) that runs the command ‘env’. Run it and see the output.

Solution

(option 1) $ **kubectl run busybox --image=busybox --restart=Never -it --command -- env**
(option 2) $ **kubectl run busybox --image=busybox --restart=Never -it -- /bin/sh -c "env"**

Exercise 4

Create a busybox pod (using YAML) that runs the command ‘env’. Run it and see the output.

Solution

$ **kubectl run busybox --image=busybox --restart=Never --dry-run -o yaml --command -- env | kubectl apply -f -**

$ **kubectl logs busybox**

```
nycmbw-nwx0087:kubernetes cavendanoa$ kubectl run busybox --image=busybox --restart=Never --dry-run -o yaml --command -- env | kubectl apply -f -
pod/busybox created
nycmbw-nwx0087:kubernetes cavendanoa$ kubectl logs busybox
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=busybox
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_SERVICE_PORT=443
HOME=/root
```

Exercise 5

Get the YAML for a new namespace called ‘myns’ without creating it.

Solution

$ **kubectl create namespace myns --dry-run --output yaml**

Exercise 6

Get the YAML for a new resource quota called ‘myrq’ without creating it.

Solution

$ **kubectl create quota test --hard=count/pods=4,count/configmaps=4,count/deployments.extensions=10 --dry-run -o yaml**

```
nycmbw-nwx0087:kubernetes cavendanoa$ kubectl create quota test --hard=count/pods=4,count/configmaps=4,count/deployments.extensions=10
--dry-run -o yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  creationTimestamp: null
  name: test
spec:
  hard:
    count/configmaps: "4"
    count/deployments.extensions: "10"
    count/pods: "4"
status: {}
```

Exercise 7

Get pods on all namespaces.

Solution

$ **kubectl get pods --all-namespaces**

Exercise 8

Create a Pod with image nginx called nginx and allow traffic on port 80

Solution

$ **kubectl run nginx --image=nginx --restart=Never --port=80**

Exercise 9

Change pod's image to nginx:1.7.1. Observe that the pod will be killed and recreated as soon as the image gets pulled

Solution

kubectl set image POD/POD_NAME CONTAINER_NAME=IMAGE_NAME:TAG
(Option 1) $ **kubectl set image pod/nginx nginx=nginx:1.7.1**
(Option 2) $ **kubectl edit pod nginx**
Change the version of the container's image.
$ **kubectl describe pod nginx**

Exercise 10

Get the pod's ip, use a temp busybox image to wget its '/'

Solution

(Option 1) $ **kubectl describe pod nginx**
(Option 2) $ **kubectl get pod -o wide**
You will find something like **IP: 172.17.0.6**

$ **kubectl run busybox --image=busybox --restart=Never --rm -it -- /bin/sh -c "wget -O- http://172.17.0.6:80"**

Exercise 11

Get this pod's YAML without cluster specific information

Solution

$ **kubectl get pod nginx -o yaml --export**

Exercise 12

Get information about the pod, including details about potential issues (e.g. pod hasn't started)

Solution

$ **kubectl describe pod nginx**

Exercise 13

Get pod logs

Solution

$ **kubectl logs nginx**

Exercise 14

If pod crashed and restarted, get logs about the previous instance

Solution

$ **kubectl logs nginx -p**

Exercise 16

Create a busybox pod that echoes 'hello world' and then exits

Solution

$ **kubectl run busybox --image=busybox --restart=Never -it -- bin/sh -c "echo hello world"**

Exercise 17

Do the same, but have the pod deleted automatically when it's completed

Solution

$ **kubectl run busybox --image=busybox --restart=Never -it --rm -- bin/sh -c "echo hello world"**

Exercise 18

Create an nginx pod and set an env value as 'var1=val1'. Check the env value existence within the pod

Solution

$ **kubectl run nginx --image=nginx --restart=Never -it --env=var1=val1 -- bin/sh -c "echo var1"**

Exercise 19

Create a Pod with two containers, both with image busybox and command "echo hello; sleep 3600". Connect to the second container and run 'ls'

Solution

$ **kubectl run busybox --image=busybox --restart=Never --dry-run -o yaml -- bin/sh -c "echo hello world;sleep 3600"**
Then, I edit the output to add a second container in the pod (exercise-19.yaml).
$ **kubectl apply -f lab-pod-basics/exercise-19.yaml**
$ **kubectl exec -it busybox -c busybox2 -- sh**
  # ls
  # exit

Exercise 20

Create 3 pods with names nginx1, nginx2, nginx3. All of them should have the label app=v1

Solution

$ **kubectl run nginx1 --image=nginx --restart=Never --dry-run -o yaml**
Then, I edit the output to add a two more pods in a kubernetes List object (exercise-20.yaml).
$ **kubectl apply -f lab-pod-basics/exercise-20.yaml**

Alternatively, we can run these commands:
$ **kubectl run nginx1 --image=nginx --restart=Never --labels=app=v1**
$ **kubectl run nginx2 --image=nginx --restart=Never --labels=app=v1**
$ **kubectl run nginx3 --image=nginx --restart=Never --labels=app=v1**

Exercise 21

Show all labels of the pods

Solution

$ **kubectl get pods --show-labels**

```
[nycmbw-nwx0087:kubernetes cavendanoa$ kubectl get pods --show-labels
NAME      READY   STATUS    RESTARTS   AGE   LABELS
nginx1    1/1     Running   0          67s   app=v1
nginx2    1/1     Running   0          49s   app=v1
nginx3    1/1     Running   0          42s   app=v1
```

Exercise 22

Change the labels of pod 'nginx2' to be app=v2

Solution

$ **kubectl label --overwrite pods nginx2 app=v2**

```
nycmbw-nwx0087:kubernetes cavendanoa$ kubectl get po --show-labels
NAME      READY   STATUS    RESTARTS   AGE     LABELS
nginx1    1/1     Running   0          4m57s   app=v1
nginx2    1/1     Running   0          4m39s   app=v2
nginx3    1/1     Running   0          4m32s   app=v1
```

Exercise 23

Get the label 'app' for the pods

Solution

$ **kubectl get pods -l "app"**

```
[nycmbw-nwx0087:kubernetes cavendanoa$ kubectl get pods -l "app"
NAME     READY   STATUS    RESTARTS   AGE
nginx1   1/1     Running   0          7m14s
nginx2   1/1     Running   0          6m56s
nginx3   1/1     Running   0          6m49s
```

Exercise 24

Get only the 'app=v2' pods

Solution

$ **kubectl get pods -l "app=v2"**

```
[nycmbw-nwx0087:kubernetes cavendanoa$ kubectl get pods -l "app=v2"
NAME      READY   STATUS    RESTARTS   AGE
nginx2    1/1     Running   0          8m17s
```

Exercise 25

Remove the 'app' label from the pods we created before

Solution

$ **kubectl label pods nginx1 nginx2 nginx3 app-**

```
nycmbw-nwx0087:kubernetes cavendanoa$ kubectl get pods --show-labels
NAME      READY   STATUS    RESTARTS   AGE   LABELS
nginx1    1/1     Running   0          13m   <none>
nginx2    1/1     Running   0          13m   <none>
nginx3    1/1     Running   0          13m   <none>
```

Exercise 26

Create a pod that will be deployed to a Node that has the label 'accelerator=nvidia-tesla-p100'

Solution

$ **kubectl apply -f lab-pod-basics/exercise-26.yaml**

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: busybox
  name: busybox
spec:
  containers:
  - image: busybox
    name: busybox
  nodeSelector:
    accelerator: nvidia-tesla-p100
```

Exercise 27

Annotate pods nginx1, nginx2, nginx3 with "description='my description'" value.

Solution

$ **kubectl annotate pods nginx1 nginx2 nginx3 "description=my description"**

Exercise 28

Check the annotations for pod nginx1.

Solution

$ **kubectl describe pod nginx1**

```
nycmbw-nwx0087:kubernetes cavendanoa$ kubectl describe pod nginx1
Name:         nginx1
Namespace:    default
Priority:     0
Node:         minikube/192.168.64.6
Start Time:   Sat, 03 Aug 2019 02:57:26 -0500
Labels:       app=v1
Annotations:  description: my description
              kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":{},"labels":{"app":"v1"},"name":"nginx1","namespace":"default"},"spec":{"contain...
Status:       Running
IP:           172.17.0.6
Containers:
```

Exercise 29

Remove the annotations for these three pods.

Solution

$ **kubectl annotate pods nginx1 nginx2 nginx3 description-**

Exercise 30

Remove these pods to have a clean state in your cluster.

Solution

$ **kubectl delete pods nginx{1..3}**