CKAD Practice Exam: Multi-Container Pods Solution Guide

Intermediate 20m Bookmark

Check 1: Pod with Legacy Logs Potential Solution

```
# A Pod in the mcp namespace has a single container named random hat
writes its logs to the /var/log/random.log file. Add a second container
named second that uses the busybox image to allow the following command
to display the logs written to the random container's /var/log/random.log
file: kubectl -n mcp logs random second

# Start by saving the original Pod manifest so you don't lose any
information
kubectl -n mcp get pod random -o yaml > original.yaml

# Create a copy for you to modify
cp original.yaml second.yaml

# Edit the file to be equivalent to the following
cat << EOF > second.yaml
apiVersion: v1
kind: Pod
metadata:
  name: random
  namespace: mcp
spec:
  containers:
  - args:
    - /bin/sh
    - -c
    - while true; do shuf -i 0-1 -n 1 >> /var/log/random.log; sleep 1;
done
    image: busybox
    name: random
    volumeMounts:
    - mountPath: /var/log
      name: logs
  - name: second
    image: busybox
    args: [/bin/sh, -c, 'tail -n+1 -f /var/log/random.log']
    v
    -
  vo
```

```
# Delete the original Pod
kubectl delete -f second.yaml

# Create the modified version
kubectl create -f second.yaml
```

Commentary

The Pod needs to be modified to implement the sidecar pattern to stream the logs to standard output in the sidecar container. Knowing that the documentation has an example of adding a sidecar for streaming a logs can greatly reduce time spent modifying the Pod manifest. Taking time to familiarize with what is available in the documentation and strategically bookmarking will pay off in saved time otherwise spent searching during the exam.

Because you cannot add a second container to a running Pod, you will need to delete the Pod and replace it. That is why it is recommended to make a copy of the original manifest and also why you can't use `kubectl edit` on its own to accomplish the task. It is worth noting that `kubectl edit` does provide a cleaner manifest than the output of `kubectl get`. You could copy from the `edit` manifest and paste to a new file created by `vim` to avoid the extra fields produce by `get`.

Although the manifest produce by the `get` command is noisy, you can safely focus on modifying the spec and not waste time removing fields that are set when the Pod is created. The API server is able to create the Pod even with all of the additional fields you wouldn't normally include in a manifest file. Practicing with `vim` before writing the exam is highly recommended.

An `emptyDir` volume must be introduced to give the sidecar access to the random containers log file. The `emptyDir` volume type should be used when the only requirement is to share data between containers in the same Pod. The sidecar streaming log example and the emptyDir volume documentation both illustrate how to

using

The logging architecture documentation contains a more complex example that uses fluentd as a sidecar logging agent. Because the exam doesn't allow you to navigate to any websites outside of the Kubernetes documentation, it is likely that any exam question involving fluentd would be very similar to the example.

Content to review

- Introduction to Kubernetes - Volumes
- Kubernetes Patterns for Application Developers - Multi-container Patterns

Suggested documentation bookmark(s)

- Logging Architecture
- Volumes

Check 2: Multi Container Pod Networking Potential Solution

```
# Containers within the same Pod share the same Pod IP address, and can
communicate via loopback network interface – therefore any of the
following values can be used for the <REPLACE_HOST_HERE> placeholder

* localhost
* 127.0.0.1
* webpod
* <pod.assigned.ip.address>
* <pod-assigned-ip-address>.<namespace>.pod.cluster.local

# Update and deploy the provided manifest using localhost for
<REPLACE_HOST_HERE>
cat << EOF > kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
 name: webpod
 namespace: app1
spec:
 restartPolicy: Never
 volumes:
 - name: vol1
   emptyDir: {}
 con
 - na
   im
   volumeMounts:
```

```
        exec:
          command:
            - "bash"
            - "-c"
            - |
              date | sha256sum | tr -d " *-" >
/usr/share/nginx/html/index.html
 - name: c2
   image: appropriate/curl
   command: ["/bin/sh", "-c", "curl -s http://localhost && sleep 3600"]
EOF

# Confirm that the pod is up and running
kubectl get pod -n app1 webpod

# Confirm that container c2 is correctly working (able to communicate
with container c1)
kubectl logs -n app1 webpod -c c2

# Write out the log result for c2 to the /home/ubuntu/webpod-log.txt file
kubectl logs -n app1 webpod -c c2 > /home/ubuntu/webpod-log.txt
```

Commentary

Containers within the same Pod share the same Pod IP address and can communicate with each other through the local loopback network interface. Therefore, container c2 can send HTTP network traffic to container c1 using any of the following host values:

- localhost
- 127.0.0.1
- webpod (the pod's assigned name)
- <pod-assigned-ip-address> - for example: 10.0.0.11
- <pod-assigned-ip-address>.<namespace>.pod.cluster.local - for example: 10-0-0-11.app1.pod.cluster.local, where 10.0.0.11 is the assigned Pod IP address.

Content to review

- Kubernetes Patterns for Application Developers - Multi-container Patterns
- DNS for Services and Pods

Chec

```
metadata:
 name: md5er
 namespace: app2
spec:
 restartPolicy: Never
 volumes:
 - name: vol1
   emptyDir: {}
 containers:
 - name: c1
   image: bash
   env:
   - name: DATA
     valueFrom:
       configMapKeyRef:
         name: cm1
         key: data
   volumeMounts:
   - name: vol1
     mountPath: /data
   command: ["/usr/local/bin/bash", "-c", "echo $DATA > /data/file.txt"]
 - name: c2
   image: bash
   volumeMounts:
   - name: vol1
     mountPath: /data
     readOnly: true
   command:
     - "/usr/local/bin/bash"
     - "-c"
     - |
       for word in $(</data/file.txt)
       do
       echo $word | md5sum | awk '{print $1}'
       done

# Apply the update Pod manifest into the K8s cluster
kubectl apply -f/home/ubuntu/md5er-app.yaml

# Check that the new c2 container is correctly generating MD5s
kubectl logs -n app2 md5er c2

# Save the stdout output from the c2 container as per given instructions
kubectl logs -n app2 md5er c2 > /home/ubuntu/md5er-output.log
```

Commentary

The provided Pod manifest `/home/ubuntu/md5er-app.yaml` needs to be updated with an additional container named `c2`. The `c2` container uses the same b e volum

manifest is then deployed into the cluster using the `kubectl apply` command. Finally, the instructions require the user to capture the stdout output of the `c2` container and store it in the `/home/ubuntu/md5er-output.log` file - this can be achieved by using the `kubectl logs` command and specifing the namespace, pod name, and container name - those being `app2`, `md5er`, and `c2` respectively.

As an aside - the design and configuration of the `c1` container involves reading and mounting a string stored and managed within a `ConfigMap` named `cm1`. The `data` key value within this ConfigMap is read and assigned to the DATA container environment variable, which is then later echo'd out to the `/data/file.txt` file (see the `command` attribute for the `c1` container). The cm1 ConfigMap can be viewed using the `kubectl describe` command like so:

```
kubectl describe cm -n app2 cm1

Name: cm1
Namespace: app2
Labels: <none>
Annotations: <none>

Data
====
data:
----
no one has ever done anything like this
Events: <none>
```

Content to review

- Introduction to Kubernetes - Volumes
- Kubernetes Patterns for Application Developers - Multi-container Patterns
- ConfigMaps and Pods

Suggested documentation bookmark(s)

- Logging Architecture
- V

Mark it or miss it!

Make sure to mark this content as completed; otherwise, it will not be displayed as such.

Mark as completed

Did you like this Resource? **Report an issue**

About the author

Cloud Academy

Instructor

Students	Labs	Courses	Learning paths
2,617	83	9	18

Digital skills are built at the intersection of knowledge, experience, and context. The fundamental building blocks of the training templates in our Library meet teams wherever they are along the cloud maturity curve, imparting the knowledge and experience needed to take them to the next level. Our training platform is intuitive and scalable. Most importantly, all training is easily customizable, which enables organizations to provide context and guidance for teams of any size. Teams leveraging Cloud Academy hit the ground running.

Covered topics

Deployment Compute DevOps Kubernetes

About Circus Street

COMMUNITY

Join Discord Channel

HELP

Help Center