Certified Kubernetes Administrator (CKA) — Tips and Tricks — Part 5

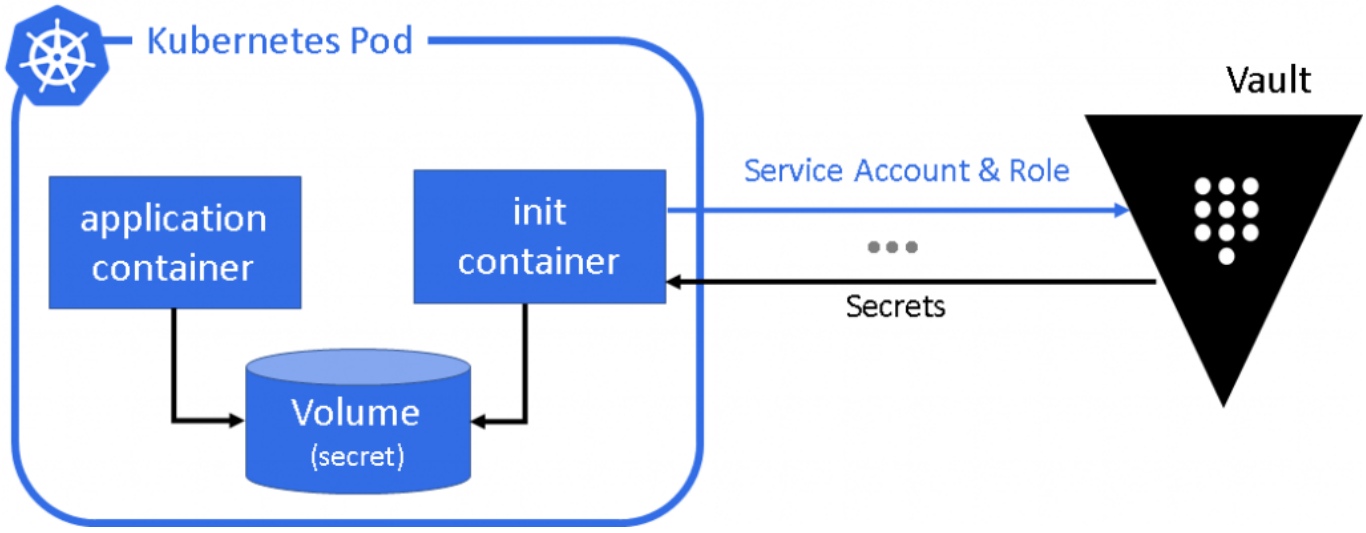
Init Container is the way to do some setup task before the actual container starts

Arun Ramakani Follow
Dec 19 · 5 min read

Init container is an important concept for the exam. There is a very high chance that this is one of your 24 questions. This blog will attempt to make you aware of the traps that you may get into and answer any form of init container question.

Init container containers are specialized containers that run before the normal containers in a Pod. Init containers generally contain setup scripts, that we are not able to make it as a part of our standard container. So when do we use init container? let's look at a real-time example.

Init containers can be used to delay app container startup until a set of preconditions are met. Say we have to download a security key from the key vault, which we do not wish to make it as the past of our regular container for security reasons, then init container is the best choice. The below picture represents the above-mentioned scenarios.



The secret is downloaded and made available in a volume, for wich the main container will have access.

Some Facts

Below are some of the facts that we should know about init container.

- We can have more than one init container in a pod
- Init Containers always run to completion
- Init Container executes in the order they are specified within the YAML definition
- Each Init Container must complete successfully before the next one starts
- If an init container fails, Kubernetes repeatedly restarts the Pod until the init container succeeds
- Init containers do not support readiness probes because they must run to completion before the Pod can be ready.

Init Container YAML

Init Container YAML config looks very similar to standard container. I will have a name, image name with optional command and volume mount details. Init container documentations which can be accessed during the exam are available here

```yaml
initContainers:
- name: init-myservice
  image: busybox:1.28
  command: ['sh', '-c', 'sleep 100']
  volumeMounts:
  - name: mypvc
    mountPath: /workdir
```

initContainers.yaml hosted with ♡ by GitHub view raw

Init Container Image

The best way to use init containers is, creating an image and adding the necessary script to the docker file to do your task. But you don't have an exam environment to do this task during the exam.

The way we should do in the exam is to use a dummy image name and then add a command attribute with all the necessary script representing the task that we wish to achieve. So which dummy image I use 🤔 "busybox:1.28" come to the rescue. The above code snippet represents a simple task of init container, enabling sleep for a specified time, blocking the main container to start.

Shared Volume

For most of the requirements with Init Container, a shared volume between init container and application container is a key. In the exam, you may get a similar question involving shared volume. Let's take a look at an example

> *Example: We should use Init Container to create a file named "sharedfile.txt" under the "work" directory and the application container should check if the file exists and sleep for a while. If the file does not exist the application container should exit.*

Let have a full YAML view first and

```yaml
apiVersion: v1
```

```
kind: Pod
metadata:
  name: init-container-test
spec:
  containers:
  - name: application-container
    image: alpine
    command: ['sh', '-c', 'if [ -f /work/sharedfile.txt ]; then sleep 99999; else exit; fi']
    volumeMounts:
    - name: workdir-volume
      mountPath: /work
  initContainers:
  - name: init-container
    image: busybox:1.28
    command: ['sh', '-c', 'mkdir /work; echo>/work/sharedfile.txt']
    volumeMounts:
    - name: workdir-volume
      mountPath: /work
  volumes:
  - name: workdir-volume
    emptyDir: {}
```

InitContainerExample.yaml hosted with ♡ by GitHub view raw

emptyDir: {} — For Shared Volume

In the above example we are using "*emptyDir: {}*" for sharing volume between init container and application container. What is that ? Documentation for the same is *here at the Kubernetes documentation page which* you can use during the exam. As mentioned in the Kubernetes documentation

"An `emptyDir` volume is first created when a Pod is assigned to a Node, and exists as long as that Pod is running on that node. As the name says, it is initially empty. Containers in the Pod can all read and write the same files in the `emptyDir` volume, though that volume can be mounted at the same or different paths in each Container. When a Pod is removed from a node for any reason, the data in the `emptyDir` is deleted forever."

Validation

Once you can create the Pod, we should validate if the pod is running and init container completed its task

> *Run "kubectl apply -f initpod.yaml"*
>
> *then "kubectl describe pod init-container-test"*

You can see the status of init container, you will see "Terminated" and reason as "Completed". This shows that the init container completed its job successfully. You will also be able to see that the volume mount is created.

```
Init Containers:
  init-container:
    Container ID:  docker://68671c0299c7792e747e94bad4acd31ecf4a6c44ff64ab2b3fdf80cd21807867
    Image:         busybox:1.28
    Image ID:      docker-pullable://busybox@sha256:141c253bc4c3fd0a201d32dc1f493bcf3fff003b
6df416dea4f41046e0f37d47
    Port:          <none>
    Host Port:     <none>
    Command:
      sh
      -c
      mkdir /work; echo>/work/sharedfile.txt
    State:          Terminated
      Reason:       Completed
      Exit Code:    0
      Started:      Tue, 17 Dec 2019 06:01:19 +0000
      Finished:     Tue, 17 Dec 2019 06:01:19 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-bhns8 (ro)
      /work from workdir-volume (rw)
```

Debugging Errors

You may end up in errors, let's see how to debug that. Update the in YAML @ init container section with the below line

```
command: ['sh', '-c', 'mkdir1 /work; echo>/work/sharedfile1.txt']
```

This will fail the init container as we trying to use "mkdir1" a command which does not exist. Note the restart count indication.

```
master $ kubectl get po
```

```
NAME                READY   STATUS                  RESTARTS   AGE
init-container-test   0/1     Init:CrashLoopBackOff   1          10s
master $ kubectl get po
NAME                READY   STATUS       RESTARTS   AGE
init-container-test   0/1     Init:Error   2          22s
```

Effectively use logs to identify the error. Look at init container error with

> *kubectl logs init-container-test init-container*

```
master $ kubectl get po
NAME                READY   STATUS   RESTARTS   AGE
init-container-test   0/1     Error    2          25s
master $ kubectl logs init-container-test init-container
sh: mkdir1: not found
```

That is all you need to know about init-container, we are done 🚀. Also, visit other tips and tricks for *Certified Kubernetes Administrator (CKA)*

Certified Kubernetes Administrator (CKA) — Tips and Tricks — Part 1

Certified Kubernetes Administrator is a challenging exam by CNCF. Unlike many other certifications, it's a practical...

medium.com

Certified Kubernetes Administrator (CKA) — Tips and Tricks — Part 2

Today let's see about a static pod question, that I faced on both of the attempts. Of course, I got it correct only in...

medium.com

Certified Kubernetes Administrator (CKA) — Tips and Tricks — Part 3

Today let's look into ETDC backup. If you get this question it's a jackpot. You can score the full mark in less than a