Init Containers

Introduction:

These are the specialized containers that run before app containers in a Pod. Init containers can contain utilities or setup scripts not present in an app image.

A Pod can have multiple containers running apps within it, but it can also have one or more init containers, which are run before the app containers are started.

Init containers are exactly like regular containers, except:

- Init containers always run to completion.
- Each init container must complete successfully before the next one starts.

Objectives:

1. Create a Pod with Init Containers

1. Create a Pod with Init Containers

Use the below Yaml file to create a pod with Init containers.

```
vi init.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app.kubernetes.io/name: MyApp
spec:
  containers:
  - name: myapp-container
    image: busybox:1.28
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice
    image: busybox:1.28
    command: ['sh', '-c', "until nslookup myservice.$(cat
/var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.loca
l; do echo waiting for myservice; sleep 2; done"]
  - name: init-mydb
    image: busybox:1.28
    command: ['sh', '-c', "until nslookup mydb.$(cat
/var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.local; do
echo waiting for mydb; sleep 2; done"]
```

Above given file, we are having **InitContainers** field and we have created two Init containers under that.
init-myservice: This container will look for a service named myservice
init-mydb: This container will look for the service named mydb.

Currently we do not have any service so let's see how this pod will react. Let's apply this definition file.

```
Kubectl apply -f init.yaml
```

```
root@master:~#
root@master:~# kubectl apply -f init.yaml
pod/myapp-pod created
root@master:~#
root@master:~# kubectl get pods
NAME        READY   STATUS     RESTARTS   AGE
myapp-pod   0/1     Init:0/2   0          7s
root@master:~#
```

Under status we can see that out of 2 Init container, 0 completed because both the containers are looking for the service they are configured for. Let's find out more under description.

kubectl describe pod myapp-pod

```
Init Containers:
  init-myservice:
    Container ID:  docker://f3719c831e2d6e7c0c4199ac133a1d245698e40f5081334b8fb255d86deadf19
    Image:         busybox:1.28
    Image ID:      docker-pullable://busybox@sha256:141c253bc4c3fd0a201d32dc1f493bcf3fff003b6df416dea4f41046e0f37d47
    Port:          <none>
    Host Port:     <none>
    Command:
      sh
      -c
      until nslookup myservice.$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.local; do echo waiting for myservice; sleep 2; done
    State:          Running
      Started:      Mon, 16 Jan 2023 10:31:25 +0000
    Ready:          False
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-7pbhh (ro)
  init-mydb:
    Container ID:
    Image:         busybox:1.28
    Image ID:
    Port:          <none>
    Host Port:     <none>
    Command:
      sh
      -c
      until nslookup mydb.$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.local; do echo waiting for mydb; sleep 2; done
    State:          Waiting
      Reason:       PodInitializing
    Ready:          False
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-7pbhh (ro)
```

So, our main container is in pending state as the Init containers are yet to be completed. From above output, **init-myservice** container is running and looking for **myservice** service whereas **init-mydb** is still in waiting condition. Unless first init container's job is completed, the second **init-mydb** will be in waiting condition only.

Let's create a service with name myservice.

vi myservice.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: myservice
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

kubectl apply myservice.yaml

```
root@master:~# kubectl apply -f myservice.yaml
service/myservice created
root@master:~#
root@master:~# kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP   8d
myservice    ClusterIP   10.98.40.84    <none>        80/TCP    7s
root@master:~#
root@master:~#
root@master:~# kubectl get pods
NAME        READY   STATUS     RESTARTS   AGE
myapp-pod   0/1     Init:1/2   0          17m
root@master:~#
```

The above output shows that the service has been created. As soon as **myservice** is created we can see the status of **myapp-pod**, one of the init container is completed.

Below output shows us that **init-myservice** has been terminated because its job is done as they run for completion. Now, job has been transferred to second init container which is **init-mydb** which is now running and looking for a service named **mydb**.

```
Init Containers:
  init-myservice:
    Container ID:  docker://f3719c831e2d6e7c0c4199ac133a1d245698e40f5081334b8fb255d86deadf19
    Image:         busybox:1.28
    Image ID:      docker-pullable://busybox@sha256:141c253bc4c3fd0a201d32dc1f493bcf3fff003b6df416dea4f41046e0f37d47
    Port:          <none>
    Host Port:     <none>
    Command:
      sh
      -c
      until nslookup myservice.$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.local; do echo waiting for myservice; sleep 2; done
    State:          Terminated
      Reason:       Completed
      Exit Code:    0
      Started:      Mon, 16 Jan 2023 10:31:25 +0000
      Finished:     Mon, 16 Jan 2023 10:48:28 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-7pbhh (ro)
  init-mydb:
    Container ID:  docker://c2fea6fe87930ca689100121c1b3d1e3f7cf5794ec12136a70a5f130d6fe7a3a
    Image:         busybox:1.28
    Image ID:      docker-pullable://busybox@sha256:141c253bc4c3fd0a201d32dc1f493bcf3fff003b6df416dea4f41046e0f37d47
    Port:          <none>
    Host Port:     <none>
    Command:
      sh
      -c
      until nslookup mydb.$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.local; do echo waiting for mydb; sleep 2; done
    State:          Running
      Started:      Mon, 16 Jan 2023 10:48:29 +0000
    Ready:          False
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-7pbhh (ro)
```

Let’s create another service named mydb.

```
vi mydb.yaml
```

```
apiVersion: v1
kind: Service
metadata:
  name: mydb
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9377
```

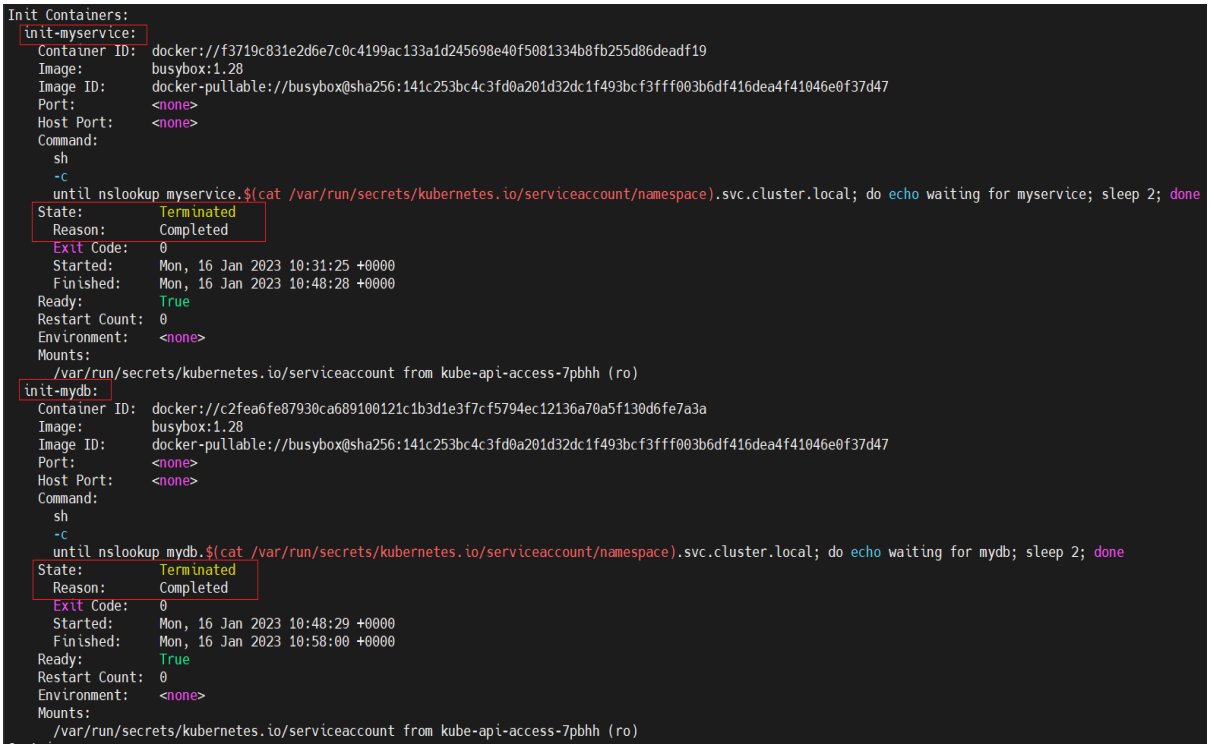
Now apply this definition file.

```
kubectl apply -f mydb.yaml
```

```
root@master:~#
root@master:~# kubectl apply -f mydb.yaml
service/mydb created
root@master:~#
root@master:~#
root@master:~# kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP   8d
mydb         ClusterIP   10.99.175.60   <none>        80/TCP    8s
myservice    ClusterIP   10.98.40.84    <none>        80/TCP    9m22s
root@master:~#
```

We have created mydb service, now let's see the status of our pod.

```
root@master:~#
root@master:~# kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
myapp-pod   1/1     Running   0          27m
root@master:~#
```



Now our pod is running as both the init containers are terminated and our main application container will start running.