Lab 4 : Kubernetes DaemonSets

Nom du module : PaaS Environment **Année Universitaire** : 2021/2022 **Public cible** : 4 ArcTIC	**Enseignants** : ➢ Soumaya Mbarek ➢ Hamdi Gabsi

Goals :

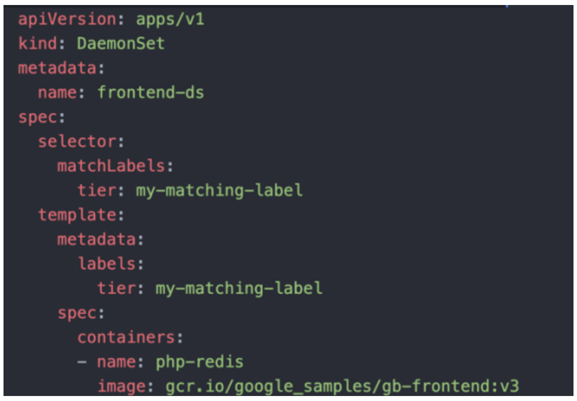
In this lab, we will have a closer look to Kubernetes DaemonSets. Daemonsets are similar to Kubernetes Deployments, but they will deploy one and only one single POD per (non-master) Kubernetes node. We will have a closer look at the two possible update strategies of DaemonSets: RollingUpdate (default) and OnDelete.

I. Step 0 : Access the Kubernetes Playground

We start by accessing the Katacode Kubernetes Playground.

II. Step 1: Create a DaemonSet

Let us try to create a Daemonset using the following YAML file :

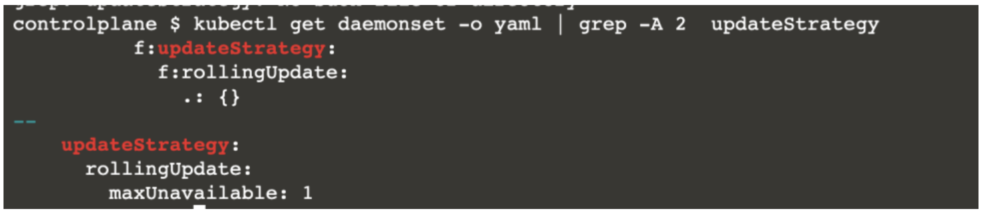


```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: frontend-ds
spec:
  selector:
    matchLabels:
      tier: my-matching-label
  template:
    metadata:
      labels:
        tier: my-matching-label
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

III. Step 2: Change the Update Strategy to OnDelete

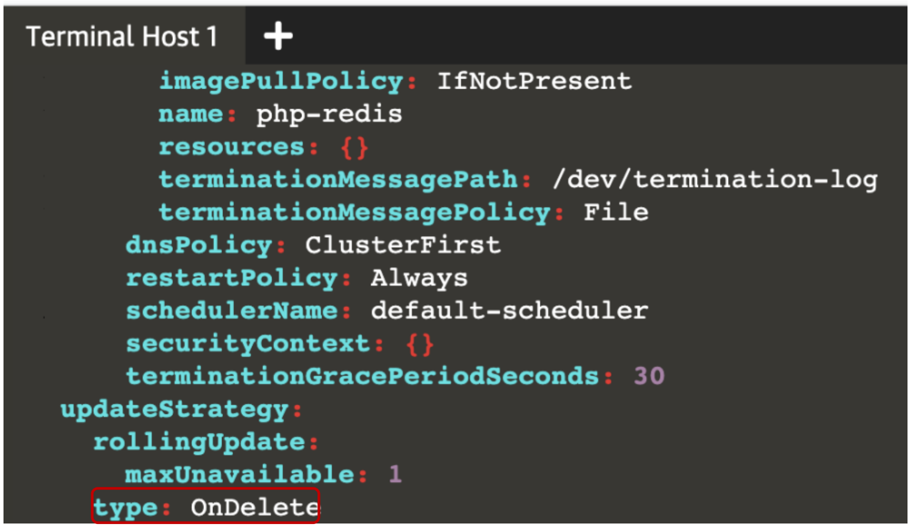
Let us review the update strategy of the DaemonSet as follows:

Lab 4 : Kubernetes DaemonSets



```
controlplane $ kubectl get daemonset -o yaml | grep -A 2  updateStrategy
          f:updateStrategy:
            f:rollingUpdate:
              .: {}
--
    updateStrategy:
      rollingUpdate:
        maxUnavailable: 1
```

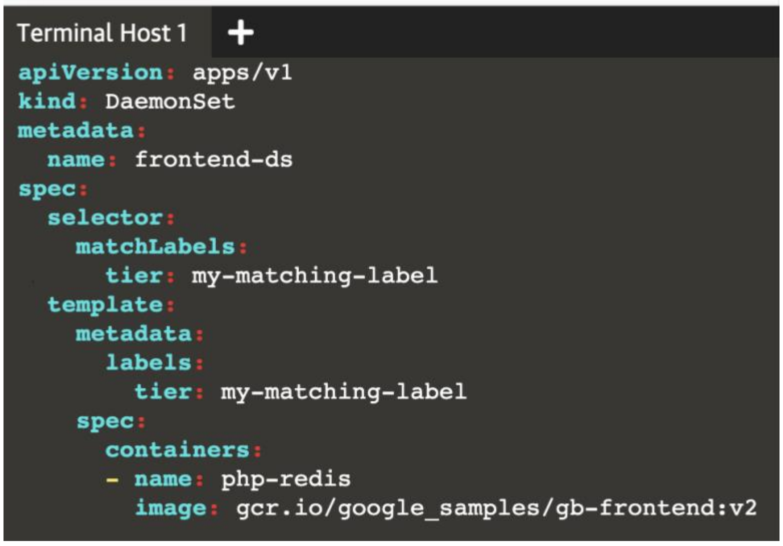
We can see, that the default update strategy of a DeamonSet is a RollingUpdate. Let us change that to OnDelete for now.



```
Terminal Host 1  +
          imagePullPolicy: IfNotPresent
          name: php-redis
          resources: {}
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
        dnsPolicy: ClusterFirst
        restartPolicy: Always
        schedulerName: default-scheduler
        securityContext: {}
        terminationGracePeriodSeconds: 30
    updateStrategy:
      rollingUpdate:
        maxUnavailable: 1
      type: OnDelete
```

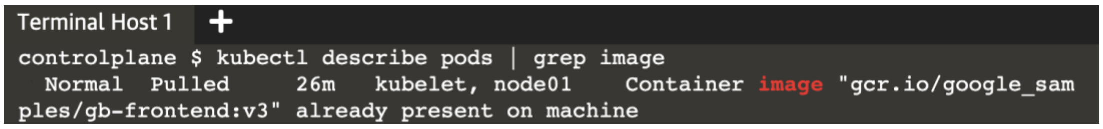
IV. Step 3: Change POD Template Image Version

We now edit the file again and change the POD template image version from v3 to v2:



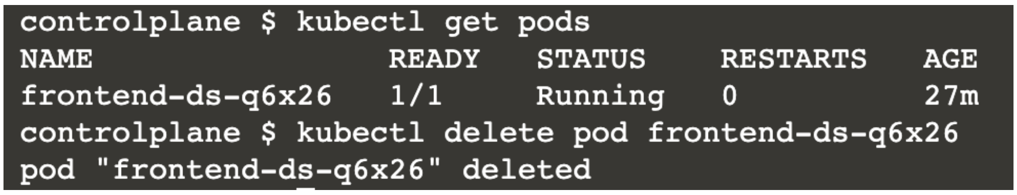
```
Terminal Host 1  +
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: frontend-ds
spec:
  selector:
    matchLabels:
      tier: my-matching-label
  template:
    metadata:
      labels:
        tier: my-matching-label
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v2
```

We check the POD Image version(s):



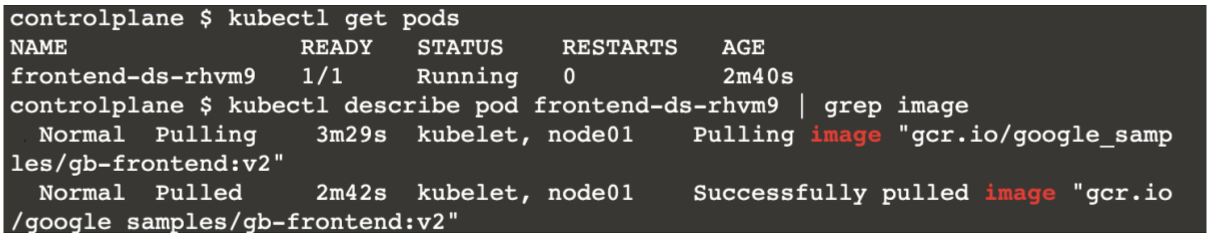
```
Terminal Host 1  +
controlplane $ kubectl describe pods | grep image
  Normal  Pulled     26m   kubelet, node01    Container image "gcr.io/google_sam
ples/gb-frontend:v3" already present on machine
```

We can see, that the image version has not changed. To trigger a replacement, we need to delete the pod:



```
controlplane $ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
frontend-ds-q6x26   1/1     Running   0          27m
controlplane $ kubectl delete pod frontend-ds-q6x26
pod "frontend-ds-q6x26" deleted
```

Now, we expect a new POD to be created with the new Docker image:

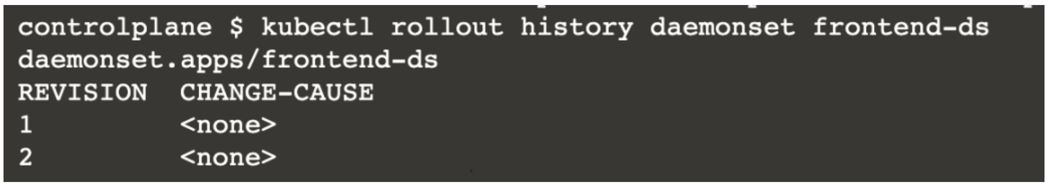


```
controlplane $ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
frontend-ds-rhvm9   1/1     Running   0          2m40s
controlplane $ kubectl describe pod frontend-ds-rhvm9 | grep image
  Normal  Pulling    3m29s  kubelet, node01    Pulling image "gcr.io/google_samp
les/gb-frontend:v2"
  Normal  Pulled     2m42s  kubelet, node01    Successfully pulled image "gcr.io
/google_samples/gb-frontend:v2"
```

V. Step 4: Rollout and Rollback

1. Step 4.1: Rollback History

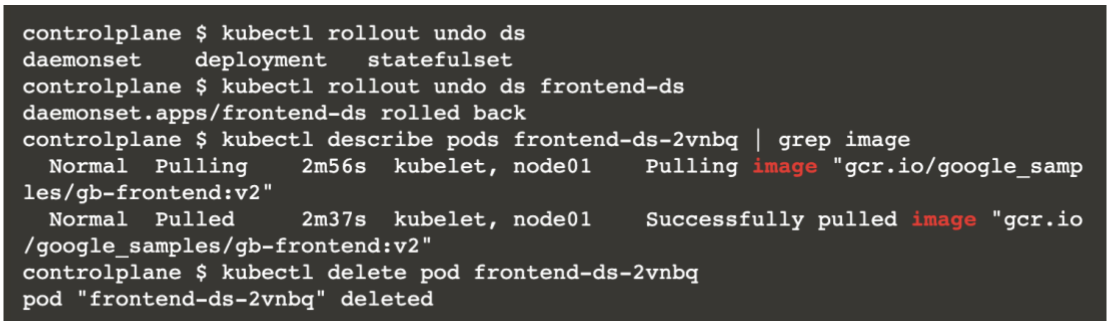
We can see that there is a rollout history available similar to Deployments:



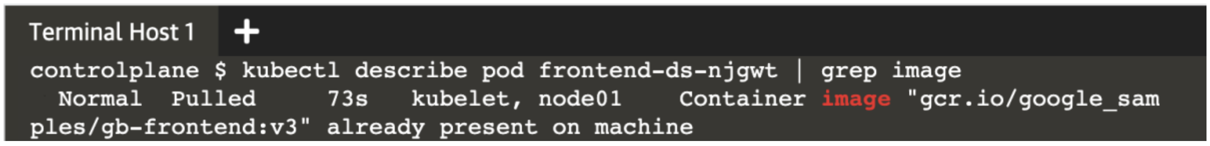
```
controlplane $ kubectl rollout history daemonset frontend-ds
daemonset.apps/frontend-ds
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
```

2. Step 4.2: Rollback

We can rollback to Image version 3 by undoing the previous rollout. This will roll back the image version of the POD template to v3. However, the PODs will not be updated automatically, and we need to delete the POD to force the update.



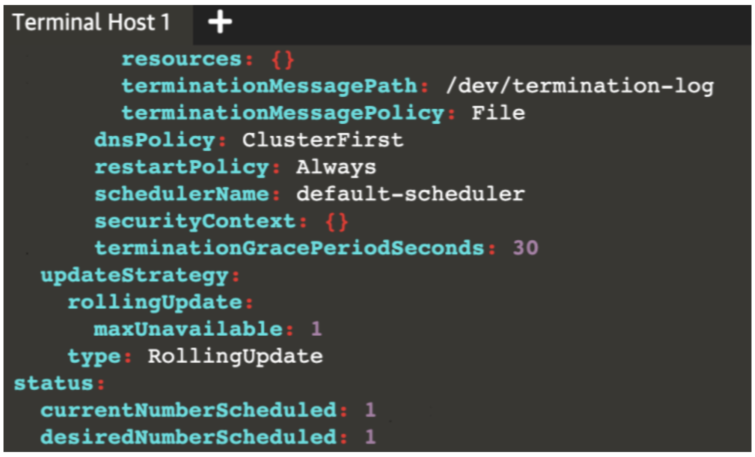
```
controlplane $ kubectl rollout undo ds
daemonset    deployment   statefulset
controlplane $ kubectl rollout undo ds frontend-ds
daemonset.apps/frontend-ds rolled back
controlplane $ kubectl describe pods frontend-ds-2vnbq | grep image
  Normal  Pulling    2m56s  kubelet, node01    Pulling image "gcr.io/google_samp
les/gb-frontend:v2"
  Normal  Pulled     2m37s  kubelet, node01    Successfully pulled image "gcr.io
/google_samples/gb-frontend:v2"
controlplane $ kubectl delete pod frontend-ds-2vnbq
pod "frontend-ds-2vnbq" deleted
```



```
controlplane $ kubectl describe pod frontend-ds-njgwt | grep image
  Normal  Pulled     73s   kubelet, node01    Container image "gcr.io/google_sam
ples/gb-frontend:v3" already present on machine
```

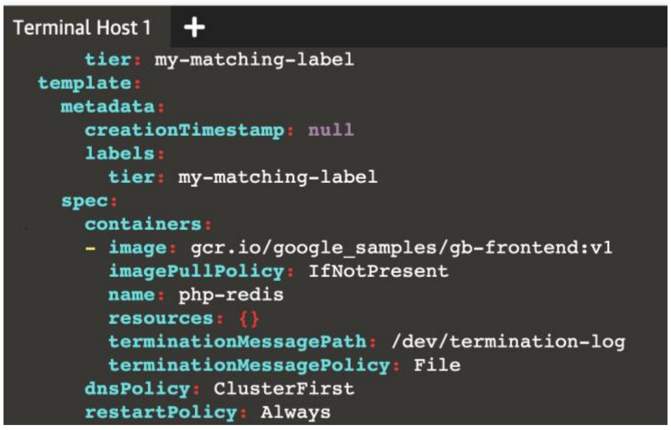
VI. Step 5: Rolling Update

Now let us change the Update Strategy to RollingUpdate:



```
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
  updateStrategy:
    rollingUpdate:
      maxUnavailable: 1
    type: RollingUpdate
status:
  currentNumberScheduled: 1
  desiredNumberScheduled: 1
```

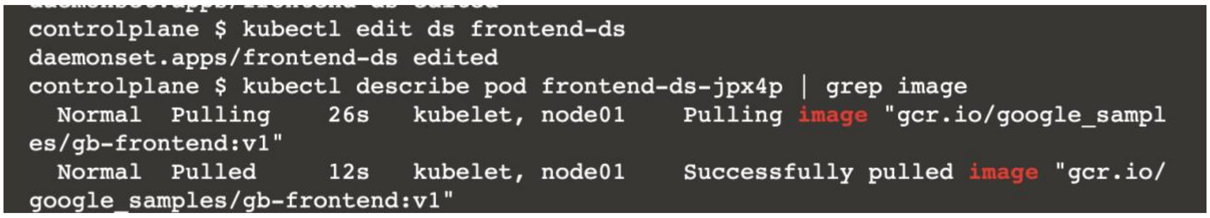
Now let us change the version again. This time, we choose v1:



```
      tier: my-matching-label
  template:
    metadata:
      creationTimestamp: null
      labels:
        tier: my-matching-label
    spec:
      containers:
      - image: gcr.io/google_samples/gb-frontend:v1
        imagePullPolicy: IfNotPresent
        name: php-redis
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
```

Lab 4 : Kubernetes DaemonSets

Since the update strategy is RollingUpdate now, the POD(s) are replaced immediately:



```
controlplane $ kubectl edit ds frontend-ds
daemonset.apps/frontend-ds edited
controlplane $ kubectl describe pod frontend-ds-jpx4p | grep image
  Normal  Pulling    26s   kubelet, node01    Pulling image "gcr.io/google_sampl
es/gb-frontend:v1"
  Normal  Pulled     12s   kubelet, node01    Successfully pulled image "gcr.io/
google_samples/gb-frontend:v1"
```

Note : We have created a minimal Kubernetes DaemonSet. First, we have observed that and POD template changes are not propagated to existing PODs if we choose the OnDelete update strategy. In that case, PODs need to be deleted, to trigger the update of the PODs. However, if we choose the RollingUpdate strategy, POD renewal is triggered with any update of the DaemonSet's POD template.