HCLSoftware

Software Containerization Lesson 5

l.ziosi@vu.nl

Copyright © 2022 HCL Software Limited | Confidential

Agenda

- Rolling Updates
- Canary Deployments

- Helm Charts

HCLSoftware

Rolling updates

How to update a Deployment ensuring that the application remains accessible.

Deployment updates

If (and only if) you change the Deployment's Pod template (.spec.template), a deployment rollout is triggered. This happens for example if the labels or container images of the template are updated.

If you make other updates, such as scaling the Deployment, no rollout is triggered.

For example, you can change the version of the image to simulate the situation where you release a new version of the application.

Start with a simple deployment of 3 replicas of nginx as shown.

Execute the deployment. You can add `--record` so it will record the cause of the change in the annotation **kubernetes.io/change-cause**:

```
kubectl apply -f nginx-deployment.yaml --record
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx-container
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Observe deployment rollout

Recall that in a previous lesson, we had created a horizontal pod autoscaler for nginx-deployment:

```
lara@kube-master-gui:~/k8s_services$ kubectl get hpa
NAME               REFERENCE                     TARGETS          MINPODS   MAXPODS   REPLICAS   AGE
nginx-deployment   Deployment/nginx-deployment   <unknown>/60%    7         8         7          13d
```

You can now describe the deployment and it will show that it's attempting to create 3 replicas.

But you can also use:

```
kubectl rollout status deployment/nginx-deployment
```

You will see that eventually it creates 7 replicas due to the minimum required by the autoscaler.

```
lara@kube-master-gui:~/k8s_services$ kubectl apply -f nginx-deployment.yaml --record
deployment.apps/nginx-deployment created
lara@kube-master-gui:~/k8s_services$ kubectl get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   0/3     3            0           14s
lara@kube-master-gui:~/k8s_services$ kubectl rollout status deployment/nginx-deployment
Waiting for deployment "nginx-deployment" rollout to finish: 0 of 7 updated replicas are available...
Waiting for deployment "nginx-deployment" rollout to finish: 1 of 7 updated replicas are available...
Waiting for deployment "nginx-deployment" rollout to finish: 2 of 7 updated replicas are available...
Waiting for deployment "nginx-deployment" rollout to finish: 3 of 7 updated replicas are available...
Waiting for deployment "nginx-deployment" rollout to finish: 4 of 7 updated replicas are available...
Waiting for deployment "nginx-deployment" rollout to finish: 5 of 7 updated replicas are available...
Waiting for deployment "nginx-deployment" rollout to finish: 6 of 7 updated replicas are available...
deployment "nginx-deployment" successfully rolled out
```

pod-template-hash label

Each pod and the ReplicaSet created by the Deployment controller get the same **pod-template-hash** label automatically added.

This label is generated by hashing the **PodTemplate** of the **ReplicaSet** and using the resulting hash as the label value that is added to the ReplicaSet selector, Pod template labels, and in any existing Pods that the ReplicaSet might have.

The purpose of this label is to ensure that ReplicaSets created from a Deployment do not overlap. If you later change something in the PodTemplate, this will generate a new ReplicaSet which will get a new hash.

```
lara@kube-master-gui:~/k8s_services$ kubectl get deployment -o wide
NAME               READY   UP-TO-DATE   AVAILABLE   AGE     CONTAINERS        IMAGES          SELECTOR
nginx-deployment   7/7     7            7           8m36s   nginx-container   nginx:1.14.2    app=nginx
lara@kube-master-gui:~/k8s_services$ kubectl get rs -o wide
NAME                          DESIRED   CURRENT   READY   AGE     CONTAINERS        IMAGES          SELECTOR
nginx-deployment-69b6b46f5    7         7         7       8m43s   nginx-container   nginx:1.14.2    app=nginx,pod-template-hash=69b6b46f5
lara@kube-master-gui:~/k8s_services$ kubectl get pods --show-labels
NAME                                READY   STATUS    RESTARTS   AGE     LABELS
nginx-deployment-69b6b46f5-m922q    1/1     Running   0          8m46s   app=nginx,pod-template-hash=69b6b46f5
nginx-deployment-69b6b46f5-xrbbg    1/1     Running   0          8m47s   app=nginx,pod-template-hash=69b6b46f5
nginx-deployment-69b6b46f5-qvpxs    1/1     Running   0          8m46s   app=nginx,pod-template-hash=69b6b46f5
nginx-deployment-69b6b46f5-5sgvd    1/1     Running   0          8m32s   app=nginx,pod-template-hash=69b6b46f5
nginx-deployment-69b6b46f5-bngst    1/1     Running   0          8m32s   app=nginx,pod-template-hash=69b6b46f5
nginx-deployment-69b6b46f5-qtqn5    1/1     Running   0          8m32s   app=nginx,pod-template-hash=69b6b46f5
nginx-deployment-69b6b46f5-gckl6    1/1     Running   0          8m32s   app=nginx,pod-template-hash=69b6b46f5
```

Updating the container image of a deployment

Let’s update the container image in the Deployment with the command:

```
kubectl --record deployment.apps/nginx-deployment set image deployment.v1.apps/nginx-deployment nginx-container=nginx:1.16.1
```

Then we can monitor the status of the update with the command:

```
kubectl rollout status deployment/nginx-deployment
```

```
lara@kube-master-gui:~/k8s_services$ kubectl --record deployment.apps/nginx-deployment set image deployment.v1.apps/nginx-deployment nginx-
container=nginx:1.16.1
deployment.apps/nginx-deployment image updated
deployment.apps/nginx-deployment image updated
lara@kube-master-gui:~/k8s_services$ kubectl rollout status deployment/nginx-deployment
Waiting for deployment "nginx-deployment" rollout to finish: 3 out of 7 new replicas have been updated...
```

If we now look at the pods with –show-labels we can see that there are only 6 pods with the old hash label and 3 pods are being created with the new label.

```
lara@kube-master-gui:~/helm-charts/my-first-chart/templates$ kubectl get pods --show-labels
NAME                                READY   STATUS              RESTARTS   AGE     LABELS
nginx-deployment-69b6b46f5-m922q    1/1     Running             0          22m     app=nginx,pod-template-hash=69b6b46f5
nginx-deployment-69b6b46f5-xrbbg    1/1     Running             0          22m     app=nginx,pod-template-hash=69b6b46f5
nginx-deployment-69b6b46f5-qvpxs    1/1     Running             0          22m     app=nginx,pod-template-hash=69b6b46f5
nginx-deployment-69b6b46f5-5sgvd    1/1     Running             0          22m     app=nginx,pod-template-hash=69b6b46f5
nginx-deployment-69b6b46f5-bngst    1/1     Running             0          22m     app=nginx,pod-template-hash=69b6b46f5
nginx-deployment-69b6b46f5-qtqn5    1/1     Running             0          22m     app=nginx,pod-template-hash=69b6b46f5
nginx-deployment-67585b6796-58cph   0/1     ContainerCreating   0          3m52s   app=nginx,pod-template-hash=67585b6796
nginx-deployment-67585b6796-lx29n   0/1     ContainerCreating   0          3m52s   app=nginx,pod-template-hash=67585b6796
nginx-deployment-67585b6796-j8v84   0/1     ImagePullBackOff    0          3m53s   app=nginx,pod-template-hash=67585b6796
```

Max unavailable and max surge

```
lara@kube-master-gui:~/k8s_services$ kubectl describe deployment nginx-deployment
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Sun, 17 Jan 2021 17:29:53 +0100
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision: 2
                        kubernetes.io/change-cause:
                          kubectl deployment.apps/nginx-deployment set image deployment.v1.apps/nginx-deployment nginx-container=nginx:1.16.1 --record=true
Selector:               app=nginx
Replicas:               3 desired | 1 updated | 4 total | 3 available | 1 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx-container:
    Image:        nginx:1.16.1
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    ReplicaSetUpdated
OldReplicaSets:  nginx-deployment-69b6b46f5 (3/3 replicas created)
NewReplicaSet:   nginx-deployment-67585b6796 (1/1 replicas created)
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  2m50s  deployment-controller  Scaled up replica set nginx-deployment-69b6b46f5 to 3
  Normal  ScalingReplicaSet  118s   deployment-controller  Scaled up replica set nginx-deployment-67585b6796 to 1
```

How deployments ensure the application remains available

Deployment ensures that only a certain number of Pods are down while they are being updated.

By default, it ensures that at least 75% of the desired number of Pods are up (**25% max unavailable**).

Deployment also ensures that only a certain number of Pods are created above the desired number of Pods.

By default, it ensures that at most 125% of the desired number of Pods are up (**25% max surge**).

For example, if I remove the hpa and the deployment, and re-create the Deployment with replicas=3, it starts by adding one pod without removing any of the existing 3 pods, because removing one would violate the max unavailable rule (we would have 33% of pods unavailable):

```
lara@kube-master-gui:~/k8s_services$ kubectl get pods --show-labels
NAME                                READY   STATUS              RESTARTS   AGE   LABELS
nginx-deployment-69b6b46f5-7vbgm    1/1     Running             0          67s   app=nginx,pod-template-hash=69b6b46f5
nginx-deployment-69b6b46f5-xf7z4    1/1     Running             0          67s   app=nginx,pod-template-hash=69b6b46f5
nginx-deployment-69b6b46f5-4fgtd    1/1     Running             0          67s   app=nginx,pod-template-hash=69b6b46f5
nginx-deployment-67585b6796-2wm2k   0/1     ContainerCreating   0          15s   app=nginx,pod-template-hash=67585b6796
```

Inspecting the History of a Deployment rollout

My update got stuck on pulling the new image from DockerHub, which I could see by inspecting the Pod and the rollout status:

```
lara@kube-master-gui:~/k8s_services$ kubectl describe pod nginx-deployment-67585b6796-2wm2k
Name:           nginx-deployment-67585b6796-2wm2k
```

```
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  4m59s  default-scheduler  Successfully assigned default/nginx-deployment-67585b6796-2wm2k to kube-master-gui
  Normal  Pulling    4m58s  kubelet            Pulling image "nginx:1.16.1"
lara@kube-master-gui:~/k8s_services$ kubectl rollout status deployment/nginx-deployment
Waiting for deployment "nginx-deployment" rollout to finish: 1 out of 3 new replicas have been updated...
error: deployment "nginx-deployment" exceeded its progress deadline
```

Look at the deployment history with the command:

```
kubectl rollout history deployment.v1.apps/nginx-deployment
```

```
lara@kube-master-gui:~/k8s_services$ kubectl rollout history deployment.v1.apps/nginx-deployment
deployment.apps/nginx-deployment
REVISION  CHANGE-CAUSE
1         kubectl apply --filename=nginx-deployment.yaml --record=true
2         kubectl deployment.apps/nginx-deployment set image deployment.v1.apps/nginx-deployment nginx-container=nginx:1.16.1 --record=true
```

CHANGE-CAUSE was copied from the Deployment annotation **kubernetes.io/change-cause** which was generated because of the `--record` command line parameter we used.

Seeing details of each Revision

You can see the details of each revision by appending `--revision=n` to the rollout history command:

```
lara@kube-master-gui:~/k8s_services$ kubectl rollout history deployment.v1.apps/nginx-deployment --revision=1
deployment.apps/nginx-deployment with revision #1
Pod Template:
  Labels:       app=nginx
        pod-template-hash=69b6b46f5
  Annotations:  kubernetes.io/change-cause: kubectl apply --filename=nginx-deployment.yaml --record=true
  Containers:
   nginx-container:
    Image:      nginx:1.14.2
    Port:       80/TCP
    Host Port:  0/TCP
    Environment:        <none>
    Mounts:     <none>
  Volumes:      <none>

lara@kube-master-gui:~/k8s_services$ kubectl rollout history deployment.v1.apps/nginx-deployment --revision=2
deployment.apps/nginx-deployment with revision #2
Pod Template:
  Labels:       app=nginx
        pod-template-hash=67585b6796
  Annotations:  kubernetes.io/change-cause:
          kubectl deployment.apps/nginx-deployment set image deployment.v1.apps/nginx-deployment nginx-container=nginx:1.16.1 --record=true
  Containers:
   nginx-container:
    Image:      nginx:1.16.1
    Port:       80/TCP
    Host Port:  0/TCP
    Environment:        <none>
    Mounts:     <none>
  Volumes:      <none>
```

Rolling back

You can roll back to a specific revision or to the previous revision if you leave the parameter `–to-revision=n` unspecified. The command is:

```
kubectl rollout undo deployment.v1.apps/nginx-deployment –to-revision=1
```

```
lara@kube-master-gui:~/k8s_services$ kubectl rollout undo deployment.v1.apps/nginx-deployment --to-revision=1
deployment.apps/nginx-deployment rolled back
lara@kube-master-gui:~/k8s_services$ kubectl rollout history deployment.v1.apps/nginx-deployment
deployment.apps/nginx-deployment
REVISION  CHANGE-CAUSE
2         kubectl deployment.apps/nginx-deployment set image deployment.v1.apps/nginx-deployment nginx-container=nginx:1.16.1 --record=true
3         kubectl apply --filename=nginx-deployment.yaml --record=true
```

Note that the rollback actually created the 3rd revision, equal to the 1st one.
If you execute:
`kubectl describe deployment nginx-deployment`
You can see that:

- the first replica set was scaled up to 3 (first revision).
- the second replica set was scaled up to 1 (second revision; this could not go further as it could not pull the image).
- the second replica set was scaled back down to zero (third revision).

```
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  21m    deployment-controller  Scaled up replica set nginx-deployment-69b6b46f5 to 3
  Normal  ScalingReplicaSet  20m    deployment-controller  Scaled up replica set nginx-deployment-67585b6796 to 1
  Normal  ScalingReplicaSet  2m39s  deployment-controller  Scaled down replica set nginx-deployment-67585b6796 to 0
```

Successful deployment update

I was able to make a successful update to the image tagged as nginx:latest which was present in my local registry.

To see the images in the local microk8s registry add-on, use the command:

```
microk8s ctr images ls
```

```
lara@kube-master-gui:~/k8s_services$ microk8s ctr images ls |grep nginx
docker.io/library/nginx:1.14.2                                                                application/vnd.docker
.distribution.manifest.list.v2+json sha256:f7988fb6c02e0ce69257d9bd9cf37ae20a60f1df7563c3a2a6abe24160306b8d 42.6 MiB  linux/386,linux/amd6
4,linux/arm/v7,linux/arm64/v8,linux/ppc64le,linux/s390x                                       io.cri-containerd.image=managed
docker.io/library/nginx:latest                                                                application/vnd.docker
.distribution.manifest.list.v2+json sha256:10b8cc432d56da8b61b070f4c7d2543a9ed17c2b23010b43af434fd40e2ca4aa 51.1 MiB  linux/386,linux/amd6
4,linux/arm/v5,linux/arm/v7,linux/arm64/v8,linux/mips64le,linux/ppc64le,linux/s390x io.cri-containerd.image=managed
```

After deleting the previous deployment, re-create the deployment and update it with:

```
kubectl --record deployment.apps/nginx-deployment set image
deployment.v1.apps/nginx-deployment nginx-container=nginx:latest
```

```
lara@kube-master-gui:~/k8s_services$ kubectl --record deployment.apps/nginx-deployment set image deployment.v1.apps/nginx-deployment nginx
-container=nginx:latest
deployment.apps/nginx-deployment image updated
deployment.apps/nginx-deployment image updated
lara@kube-master-gui:~/k8s_services$ kubectl get deployment
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     2            3           26s
```

Events in a successful deployment update

```
lara@kube-master-gui:~/k8s_services$ kubectl describe deployment nginx-deployment
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Sun, 17 Jan 2021 18:14:55 +0100
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision: 2
                        kubernetes.io/change-cause:
                          kubectl deployment.apps/nginx-deployment set image deployment.v1.apps/nginx-deployment nginx-container=nginx:latest --record=true
Selector:               app=nginx
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx-container:
    Image:        nginx:latest
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-deployment-65c9c688d6 (3/3 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  81s   deployment-controller  Scaled up replica set nginx-deployment-69b6b46f5 to 3
  Normal  ScalingReplicaSet  58s   deployment-controller  Scaled up replica set nginx-deployment-65c9c688d6 to 1
  Normal  ScalingReplicaSet  56s   deployment-controller  Scaled down replica set nginx-deployment-69b6b46f5 to 2
  Normal  ScalingReplicaSet  56s   deployment-controller  Scaled up replica set nginx-deployment-65c9c688d6 to 2
  Normal  ScalingReplicaSet  54s   deployment-controller  Scaled down replica set nginx-deployment-69b6b46f5 to 1
  Normal  ScalingReplicaSet  54s   deployment-controller  Scaled up replica set nginx-deployment-65c9c688d6 to 3
  Normal  ScalingReplicaSet  52s   deployment-controller  Scaled down replica set nginx-deployment-69b6b46f5 to 0
```

HCLSoftware

Canary Deployments

How to test a new release for a subset of users

Canary Deployments

Canary Deployments are used to test a new release with a subset of users, before propagating the changes to all users, so in case something is wrong with the new codebase, you can roll back without affecting the large user community.

“Canaries were iconically used in coal mines to detect the presence of carbon monoxide. The bird's rapid breathing rate, small size, and high metabolism, compared to the miners, led birds in dangerous mines to succumb before the miners, thereby giving them time to take action. “ (https://en.wikipedia.org/wiki/Sentinel_species#Toxic_gases)

Running a Canary Deployment in Kubernetes involves using at least one Service that can direct the traffic to pods that run the old code or pods that run the new code.

To achieve this, you typically add one more label to the Pods. The value of this new Label indicates whether the pod is of the original type or the canary type.

If the Service selects only the pods with the old label, all the traffic goes to those and all users connect to the old application.

If the Service does not discriminate based on this label (it does not use it in the selector) then both types of pods get traffic directed to them. Some users connect to the old application and some to the new one.

If the Service only selects the canary type, then the application is completely upgraded to the new codebase

Example of Canary Deployment

Container Solutions published an interesting Blog post that explains many deployment strategies, including Rolling Updates, Blue-Green and Canary:

https://blog.container-solutions.com/kubernetes-deployment-strategies

For each type, the blob provides an example in GitHub. The following example is simplified from the Canary example with native functionality:

https://github.com/ContainerSolutions/k8s-deployment-strategies/tree/master/canary/native

Current Deployment (v1.0.0)

Create a Deployment with 10 replicas of the first version of the application.

You signal this by adding a version label besides the normal application selector label app: my-app.

Note that you can also inject the value into the container using env.

You can then get the list of pods with `--show-labels` parameter to see the results:

```
lara@kube-master-gui:~/canary$ kubectl get pods --show-labels
NAME                          READY   STATUS    RESTARTS   AGE     LABELS
my-app-v1-f6bfd9988-jp69n     1/1     Running   0          6m28s   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-5fsqn     1/1     Running   0          6m28s   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-5vmgz     1/1     Running   0          6m28s   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-jx429     1/1     Running   0          6m28s   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-s67ds     1/1     Running   0          6m28s   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-gdg26     1/1     Running   0          6m28s   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-jjn9k     1/1     Running   0          6m28s   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-d778k     1/1     Running   0          6m28s   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-mb5gl     1/1     Running   0          6m28s   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-6qfvh     1/1     Running   0          6m28s   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app-v1
  labels:
    app: my-app
spec:
  replicas: 10
  selector:
    matchLabels:
      app: my-app
      version: v1.0.0
  template:
    metadata:
      labels:
        app: my-app
        version: v1.0.0
    spec:
      containers:
      - name: my-app-container
        imagePullPolicy: IfNotPresent
        image: nginx:latest
        ports:
        - name: http
          containerPort: 80
        env:
        - name: VERSION
          value: v1.0.0
```

Canary Deployment (v2.0.0)

For the canary Deployment, create just one replica (it is frequent for a canary to run on 10% of the total number of replicas).

Add the same label name, version, with value v2.0.0.

Also add this as a environment variable for the container.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app-v2
  labels:
    app: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
      version: v2.0.0
  template:
    metadata:
      labels:
        app: my-app
        version: v2.0.0
    spec:
      containers:
      - name: my-app
        imagePullPolicy: IfNotPresent
        image: nginx:latest
        ports:
        - name: http
          containerPort: 80
        env:
        - name: VERSION
          value: v2.0.0
```

Service Definition for canary Deployment

Create a Service of type NodePort that selects all the pods that have the label app set to the value my-app.

Note that the Service does not distinguish which pods to select based on the version label.

Check the Service and note the nodePort: 31414 in this example.

```
lara@kube-master-gui:~/canary$ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.152.183.1     <none>        443/TCP          17d
my-app       NodePort    10.152.183.137   <none>        8080:31414/TCP   40m
```

```
apiVersion: v1
kind: Service
metadata:
  name: my-app
  labels:
    app: my-app
spec:
  type: NodePort
  ports:
  - name: http
    port: 8080
    targetPort: 80
  selector:
    app: my-app
```

Perform the Canary Deployment

Create the canary deployment and then scale down the current deployment from 10 to 9 replicas:

```
lara@kube-master-gui:~/canary$ kubectl apply -f app-v2.yaml
deployment.apps/my-app-v2 created
lara@kube-master-gui:~/canary$ kubectl scale --replicas=9 deploy my-app-v1
deployment.apps/my-app-v1 scaled
```

Check the labels of the pods: one pod with v.1.0.0 is Terminating and one Pod with v2.0.0 is running while 9 Pods with v1.0.0 are Running:

```
lara@kube-master-gui:~/canary$ kubectl get pods --show-labels
NAME                         READY   STATUS        RESTARTS   AGE   LABELS
my-app-v1-f6bfd9988-jp69n    1/1     Running       0          19m   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-5fsqn    1/1     Running       0          19m   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-5vmgz    1/1     Running       0          19m   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-jx429    1/1     Running       0          19m   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-s67ds    1/1     Running       0          19m   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-gdg26    1/1     Running       0          19m   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-jjn9k    1/1     Running       0          19m   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-d778k    1/1     Running       0          19m   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-6qfvh    1/1     Running       0          19m   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v2-59b4f44888-b7s6x   1/1     Running       0          27s   app=my-app,pod-template-hash=59b4f44888,version=v2.0.0
my-app-v1-f6bfd9988-mb5gl    0/1     Terminating   0          19m   app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
```

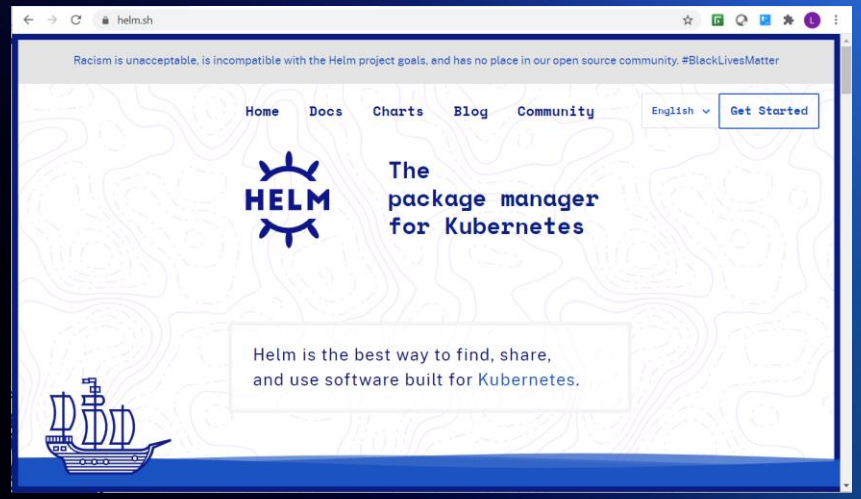
Complete the update of the application

Now users can continue to access the application via the Service.
One time out of 10 the users will be sent to the pod with the new code.
If there are no problems with v2.0.0, you can scale that Deployment up to 10 replicas.
Then you can delete the Deployment of v1.0.0.
Compared to a rolling update, this gives you the ability to test the behavior of the new version before affecting all users.

```
lara@kube-master-gui:~/canary$ kubectl scale --replicas=10 deploy my-app-v2
deployment.apps/my-app-v2 scaled
lara@kube-master-gui:~/canary$ kubectl delete deploy my-app-v1
deployment.apps "my-app-v1" deleted
lara@kube-master-gui:~/canary$ kubectl get pods --show-labels
NAME                         READY   STATUS        RESTARTS   AGE     LABELS
my-app-v2-59b4f44888-b7s6x   1/1     Running       0          9m12s   app=my-app,pod-template-hash=59b4f44888,version=v2.0.0
my-app-v2-59b4f44888-56rwv   1/1     Running       0          16s     app=my-app,pod-template-hash=59b4f44888,version=v2.0.0
my-app-v2-59b4f44888-vfq9h   1/1     Running       0          16s     app=my-app,pod-template-hash=59b4f44888,version=v2.0.0
my-app-v2-59b4f44888-vv62b   1/1     Running       0          16s     app=my-app,pod-template-hash=59b4f44888,version=v2.0.0
my-app-v2-59b4f44888-6wc5p   1/1     Running       0          16s     app=my-app,pod-template-hash=59b4f44888,version=v2.0.0
my-app-v2-59b4f44888-qdbdm   1/1     Running       0          15s     app=my-app,pod-template-hash=59b4f44888,version=v2.0.0
my-app-v2-59b4f44888-55vc6   1/1     Running       0          15s     app=my-app,pod-template-hash=59b4f44888,version=v2.0.0
my-app-v2-59b4f44888-7brfg   1/1     Running       0          16s     app=my-app,pod-template-hash=59b4f44888,version=v2.0.0
my-app-v2-59b4f44888-bntsd   1/1     Running       0          16s     app=my-app,pod-template-hash=59b4f44888,version=v2.0.0
my-app-v2-59b4f44888-r2jgk   1/1     Running       0          16s     app=my-app,pod-template-hash=59b4f44888,version=v2.0.0
my-app-v1-f6bfd9988-s67ds    0/1     Terminating   0          27m     app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-jx429    0/1     Terminating   0          27m     app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-gdg26    0/1     Terminating   0          27m     app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-d778k    0/1     Terminating   0          27m     app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-6qfvh    0/1     Terminating   0          27m     app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-5fsqn    0/1     Terminating   0          27m     app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-jjn9k    0/1     Terminating   0          27m     app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-jp69n    0/1     Terminating   0          27m     app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
my-app-v1-f6bfd9988-5vmgz    0/1     Terminating   0          27m     app=my-app,pod-template-hash=f6bfd9988,version=v1.0.0
```

HCLSoftware

Helm



Simplifying application deployment

Motivation for Helm

Helm is a Kubernetes Package manager.

So far, you created a lot of Kubernetes objects one by one to support a single application (Deployment/StatefulSet, ConfigMap, Secret, Persistent Volume, Persistent Volume Claim).

When you then want to uninstall the application you need to remember to delete all of these objects.

Helm introduces an artifact called a **Helm chart** that describes the contents of all these obejcts in a single package.

A running instance of such a set of Kubernetes objects is called a **release.**

Helm has a command line interface to install, delete, list releases.

Helm charts are stored in **repositories**, from which you can directly install them.

Helm from the end-user point of view

Helm needs to be installed separately from Kubernetes.

On microk8s you can do:

```
microk8s enable helm3
```

Unless you create an alias, you need to launch it with the following command instead of `helm`:

```
microk8s helm3
```

Helm 3 is a major change because it no longer needs the installation of the Tiller component required until version 2.

The Helm version needs to be compatible with the Kubernetes version.

Helm Version	Supported Kubernetes Versions
3.11.x	1.26.x - 1.23.x
3.10.x	1.25.x - 1.22.x
3.9.x	1.24.x - 1.21.x
3.8.x	1.23.x - 1.20.x
3.7.x	1.22.x - 1.19.x
3.6.x	1.21.x - 1.18.x
3.5.x	1.20.x - 1.17.x
3.4.x	1.19.x - 1.16.x
3.3.x	1.18.x - 1.15.x
3.2.x	1.18.x - 1.15.x
3.1.x	1.17.x - 1.14.x
3.0.x	1.16.x - 1.13.x
2.16.x	1.16.x - 1.15.x

https://helm.sh/docs/topics/version_skew/

Helm Chart Repositories

Helm *charts* are found in repositories. The default repository is:

https://artifacthub.io/packages/search?kind=0

Here you can search for charts, for example, for Postgresql:

https://artifacthub.io/packages/helm/bitnami/postgresql

To configure the repository and install the chart, do:

```
microk8s helm3 repo add my-bitnami-repo https://charts.bitnami.com/bitnami
microk8s helm3 install my-release my-bitnami-repo/postgresql
```

After configuring the repository, you can search it with:

```
microk8s helm3 search repo my-bitnami-repo
```

```
root@BLMYCLDDL31451:/home/hcluser# microk8s helm3 search repo my-bitnami-repo
NAME                                    CHART VERSION   APP VERSION     DESCRIPTION
my-bitnami-repo/airflow                 14.0.7          2.5.0           Apache Airflow is a tool to express and execute...
my-bitnami-repo/apache                  9.2.11          2.4.55          Apache HTTP Server is an open-source HTTP serve...
my-bitnami-repo/appsmith                0.1.9           1.9.2           Appsmith is an open source platform for buildin...
my-bitnami-repo/argo-cd                 4.4.2           2.5.7           Argo CD is a continuous delivery tool for Kuber...
my-bitnami-repo/argo-workflows          5.1.2           3.4.4           Argo Workflows is meant to orchestrate Kubernet...
my-bitnami-repo/aspnet-core             4.0.2           7.0.2           ASP.NET Core is an open-source framework for we...
my-bitnami-repo/cassandra               10.0.0          4.1.0           Apache Cassandra is an open source distributed ...
my-bitnami-repo/cert-manager            0.8.12          1.11.0          cert-manager is a Kubernetes add-on to automate...
my-bitnami-repo/clickhouse              2.3.1           22.12.3         ClickHouse is an open-source column-oriented OL...
my-bitnami-repo/common                  2.2.2           2.2.2           A Library Helm Chart for grouping common logic ...
my-bitnami-repo/concourse               2.0.1           7.8.3           Concourse is an automation system written in Go...
my-bitnami-repo/consul                  10.9.9          1.14.3          HashiCorp Consul is a tool for discovering and ...
my-bitnami-repo/contour                 10.1.4          1.23.2          Contour is an open source Kubernetes ingress co...
my-bitnami-repo/contour-operator        3.0.2           1.23.0          The Contour Operator extends the Kubernetes API...
```

Installing the Keycloak Helm chart

Example Helm chart installation: Keycloak, Java based server to manage user identity

https://artifacthub.io/packages/helm/bitnami/keycloak

```
lara@kube-master-gui:~$ microk8s helm3 repo add bitnami https://charts.bitnami.com/bitnami
"bitnami" has been added to your repositories
lara@kube-master-gui:~$ microk8s helm3 install my-keycloak-release bitnami/keycloak
NAME: my-keycloak-release
LAST DEPLOYED: Sat Jan 16 14:57:29 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
** Please be patient while the chart is being deployed **

Keycloak can be accessed through the following DNS name from within your cluster:

    my-keycloak-release.default.svc.cluster.local (port 80)

To access Keycloak from outside the cluster execute the following commands:

1. Get the Keycloak URL by running these commands:

  NOTE: It may take a few minutes for the LoadBalancer IP to be available.
        You can watch its status by running 'kubectl get --namespace default svc -w my-keycloak-release'

    export SERVICE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].port}" services my-keycloak-release)
    export SERVICE_IP=$(kubectl get svc --namespace default my-keycloak-release -o jsonpath='{.status.loadBalancer.ingress[0].ip}')
    echo "http://${SERVICE_IP}:${SERVICE_PORT}/auth"

2. Access Keycloak using the obtained URL.
3. Access the Administration Console using the following credentials:

  echo Username: user
  echo Password: $(kubectl get secret --namespace default my-keycloak-release-env-vars -o jsonpath="{.data.KEYCLOAK_ADMIN_PASSWORD}" | base64 --decode)
```

Inspecting the Keycloak Helm chart installation

The installation output explains how to get the IP address and port of the service to access the URL:

```
lara@kube-master-gui:~$ export SERVICE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].port}" services my-keycloak-release)
lara@kube-master-gui:~$ export SERVICE_IP=$(kubectl get svc --namespace default my-keycloak-release -o jsonpath='{.status.loadBalancer.ingress[0].ip}')
lara@kube-master-gui:~$ echo "http://${SERVICE_IP}:${SERVICE_PORT}/auth"
http://10.50.100.5:80/auth
```

There are 2 Headless Services (ClusterIP=None), one CusterIP for Postgresql, one LoadBalancer for Keycloak, one PVC/PV for Postgresql and 2 StefulSets. Note that you can get the release listed through Helm:

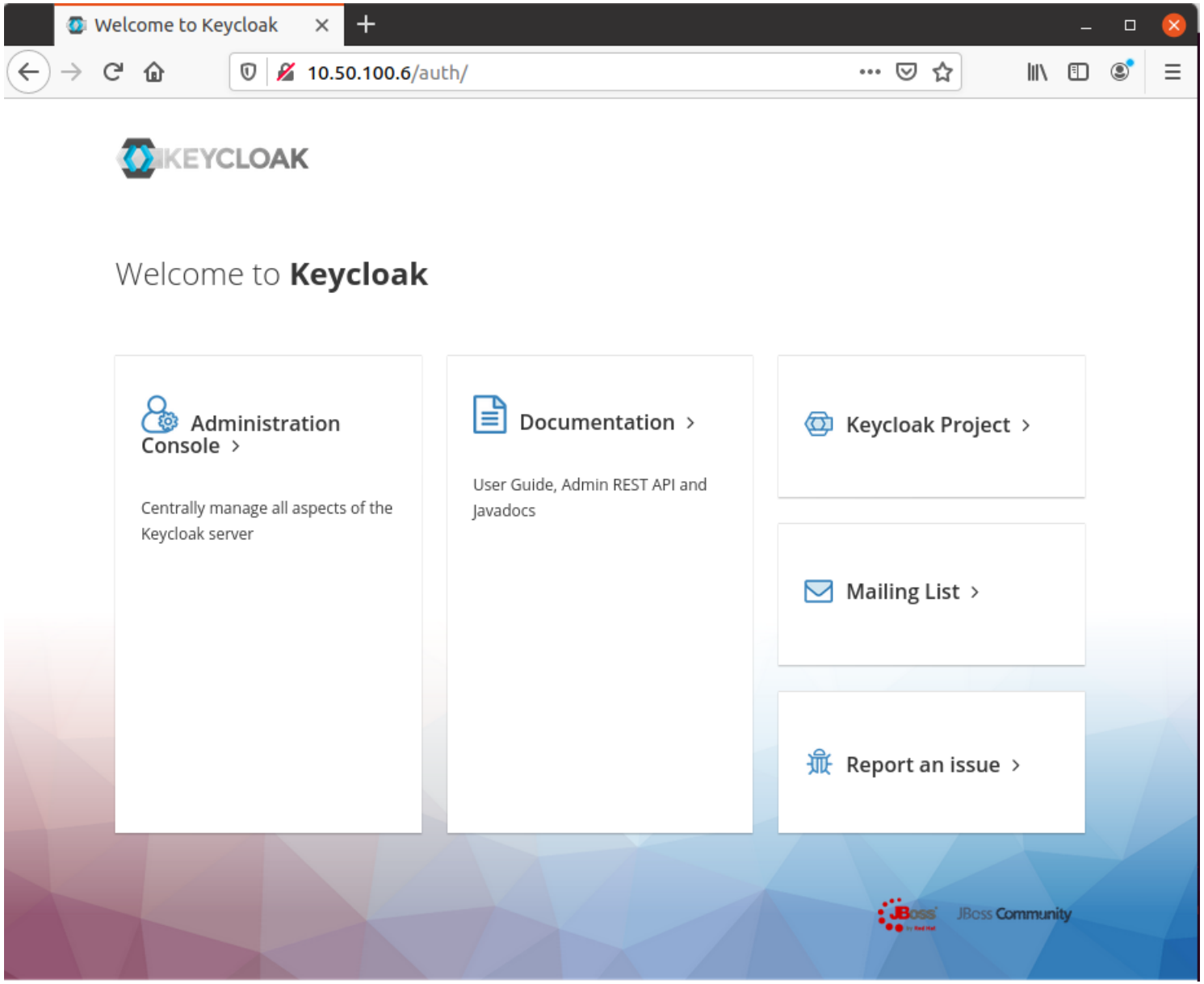
```
lara@kube-master-gui:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
my-keycloak-release-postgresql-0    1/1     Running   0          25m
my-keycloak-release-0               1/1     Running   0          25m
lara@kube-master-gui:~$ kubectl get svc
NAME                                  TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
kubernetes                            ClusterIP      10.152.183.1     <none>        443/TCP                      16d
my-keycloak-release-headless          ClusterIP      None             <none>        80/TCP                       25m
my-keycloak-release-postgresql-headless   ClusterIP      None             <none>        5432/TCP                     25m
my-keycloak-release-postgresql        ClusterIP      10.152.183.26    <none>        5432/TCP                     25m
my-keycloak-release                   LoadBalancer   10.152.183.139   10.50.100.5   80:32297/TCP,443:30191/TCP   25m
lara@kube-master-gui:~$ kubectl get pvc
NAME                                STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS        AGE
data-my-keycloak-release-postgresql-0   Bound    pvc-e98418b9-aadf-4858-a2f8-eccc91540f97   8Gi        RWO            microk8s-hostpath   25m
lara@kube-master-gui:~$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                                          STORAGECLASS        REASON   AGE
pvc-33b7eeaa-f523-4445-9c1d-e404559e1308   20Gi       RWX            Delete           Bound    container-registry/registry-claim              microk8s-hostpath            6d1h
pvc-e98418b9-aadf-4858-a2f8-eccc91540f97   8Gi        RWO            Delete           Bound    default/data-my-keycloak-release-postgresql-0   microk8s-hostpath            25m
lara@kube-master-gui:~$ microk8s helm3 list
NAME                    NAMESPACE       REVISION        UPDATED                                 STATUS          CHART           APP VERSION
my-keycloak-release     default         1               2021-01-16 15:49:38.52087127 +0100 CET  deployed        keycloak-1.2.0  11.0.3
```

```
lara@kube-master-gui:~$ kubectl get statefulset
NAME                                READY   AGE
my-keycloak-release-postgresql      1/1     33m
my-keycloak-release                 1/1     33m
```

Accessing Keycloak

To access Keycloak, you can use the LoadBalancer supplied IP address.

From here you select:
Administration Console where you need to supply username and password.



Keycloak login credentials

```
lara@kube-master-gui:~$ kubectl get secrets
NAME                                  TYPE                                  DATA   AGE
default-token-tnd79                   kubernetes.io/service-account-token   3      16d
my-keycloak-release-postgresql        Opaque                                2      37m
my-keycloak-release-env-vars          Opaque                                3      37m
my-keycloak-release-token-64fsk       kubernetes.io/service-account-token   3      37m
sh.helm.release.v1.my-keycloak-release.v1   helm.sh/release.v1              1      37m
lara@kube-master-gui:~$ kubectl get configmaps
NAME                         DATA   AGE
kube-root-ca.crt             1      16d
my-keycloak-release-env-vars   13     37m
lara@kube-master-gui:~$ kubectl describe secret my-keycloak-release-env-vars
Name:         my-keycloak-release-env-vars
Namespace:    default
Labels:       app.kubernetes.io/component=keycloak
              app.kubernetes.io/instance=my-keycloak-release
              app.kubernetes.io/managed-by=Helm
              app.kubernetes.io/name=keycloak
              helm.sh/chart=keycloak-1.2.0
Annotations:  <none>

Type:  Opaque

Data
====
KEYCLOAK_ADMIN_PASSWORD:       10 bytes
KEYCLOAK_DATABASE_PASSWORD:    13 bytes
KEYCLOAK_MANAGEMENT_PASSWORD:  10 bytes
```

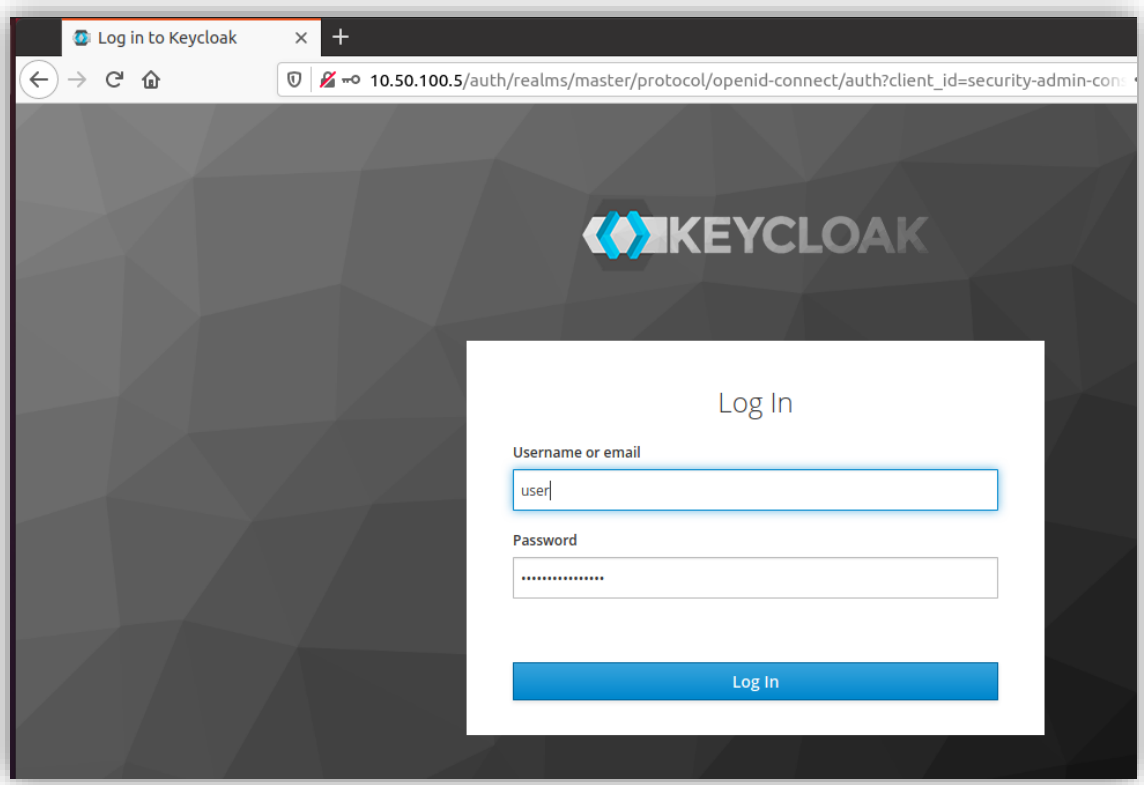
To get the login credentials, you can inspect the Secret called: my-keycloak-release-env-vars

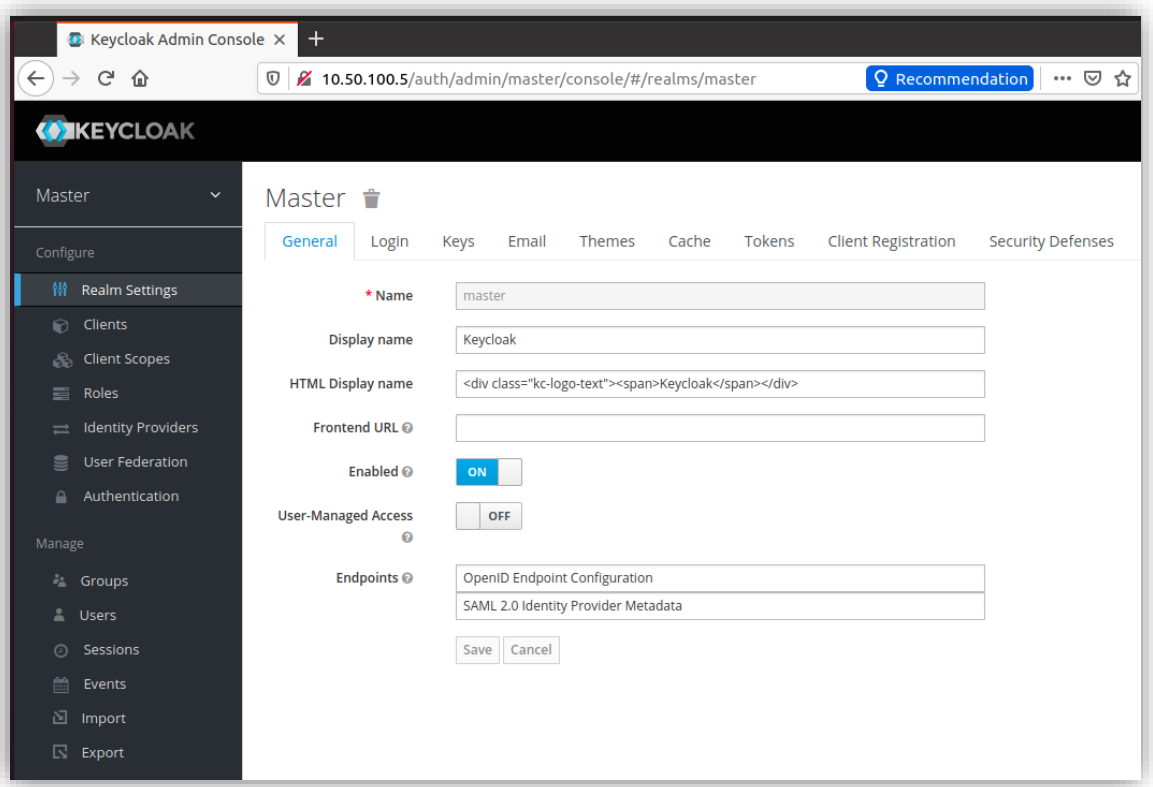
Note that there is also a ConfigMap with the same name, that contains many environment variables including default user names.

To actually be able to run this, a VM with 4GB RAM and 2 CPUs is not enough.
It runs well on a VM with 8GB RAM and 4 CPUs.

```
lara@kube-master-gui:~$ echo Password: $(kubectl get secret --namespace default my-keycloak-release-env-vars -o jsonpath="{.data.KEYCLOAK_ADMIN_PASSWORD}" | base64 --decode)
Password: 9oDFsyiTUW
lara@kube-master-gui:~$ echo Username: user
Username: user
```

Keycloak Administration Console





You might be able to see the previous page, but not to actually login to the Administration Console if your VM does not have enough resources. If you get the password by getting the value of KEYCLOAK_ADMIN_PASSWORD from the secret, remember to base64 decode it.

Helm command line interface (major commands only) 1/2

Command	Behavior
helm search hub [name]	Search for charts in Artifact Hub, possibly search for name.
helm repo add [repo-address]	Add a repository for future searches.
helm repo list	List all the available repositories
helm repo update	Update the local client with latest repo contents
helm repo remove	Remove a repository from the local client
helm search repo [name]	Search a previously added repository, possibly for name.
helm pull [chart-repo/chart-name]	Download a Helm chart package.
helm install [release-name] [chart-name]	Install the chart chart-name under the name of release-name. Note that this command may exit before all containers are initialized.
helm list	Show the installed releases
helm uninstall [release-name]	Uninstall a release.

Helm command line interface (major commands only) 2/2

Command	Behavior
helm status [release-name]	Show the status of an installed release.
helm show values [chart-name]	Show the values that can be customized in the chart.
helm get values [release-name]	Show the actual values in a running release.
helm upgrade –f [config-file] [release-name] [chart-name]	Upgrade a release based on information in a configuration file.
helm rollback [release-name] [revision-number]	Rollback a release to a previous revision number.
helm create [chart-name]	Create a new chart.
helm lint [chart-name]	Perform static code analysis on the chart.
helm package [chart-name]	Package the chart for distribution creating a tgz archive.

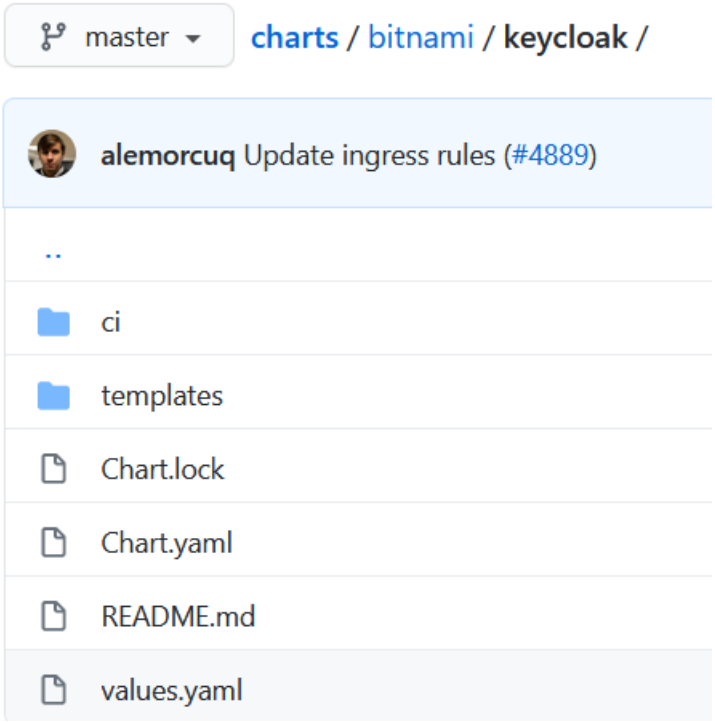
Keycloak chart Source code

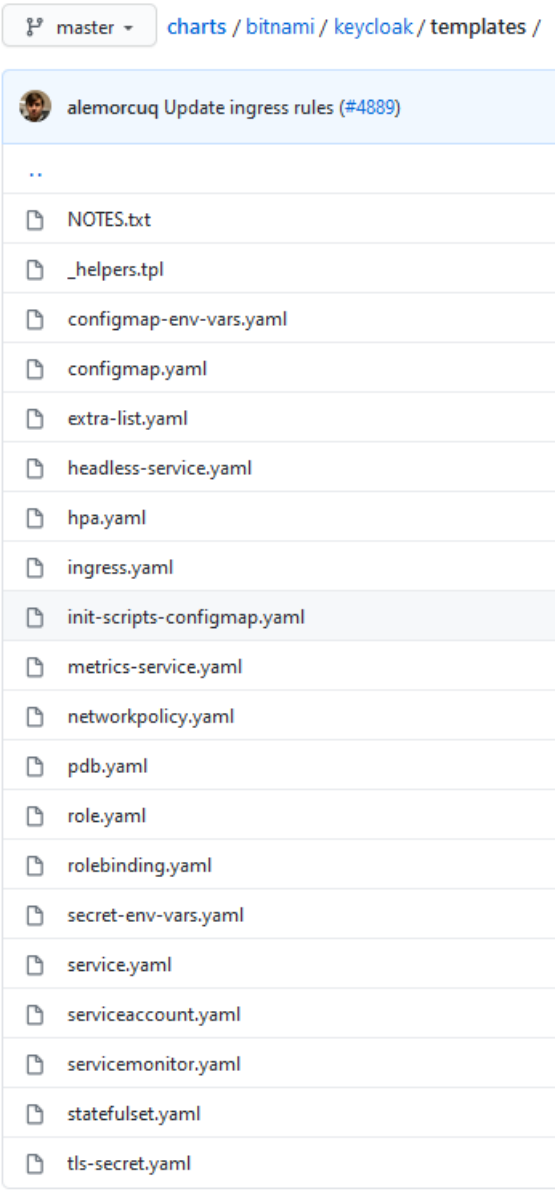
The source code of the Bitnami Keycloak Helm Chart can be found at:

https://github.com/bitnami/charts/tree/master/bitnami/keycloak

You can also download the source code as a .tgz file using the command:
`helm pull repository/chart`

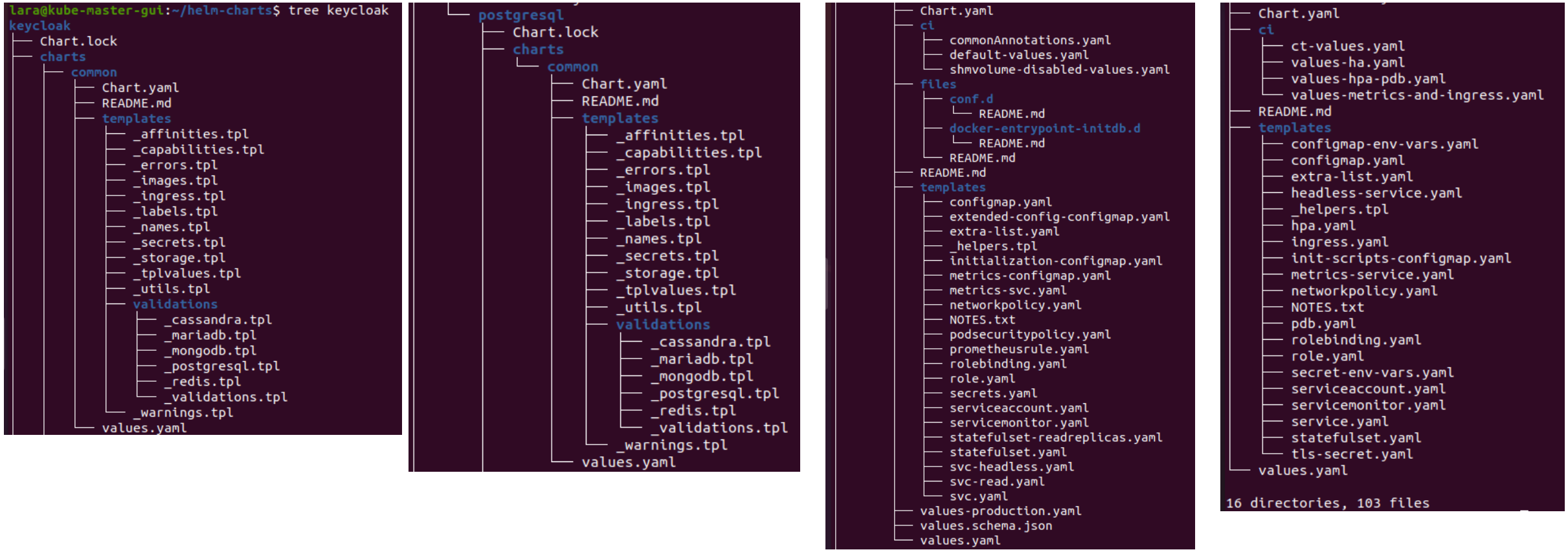
To know the name of the repository and chart, you can use the command:
`helm search repo name`






```
lara@kube-master-gui:~/helm-charts$ microk8s helm3 search repo keycloak
NAME                    CHART VERSION   APP VERSION     DESCRIPTION
bitnami/keycloak        1.2.0           11.0.3          Keycloak is a high performance Java-based ident...
stable/keycloak         4.10.1          5.0.0           DEPRECATED - Open Source Identity and Access Ma...
lara@kube-master-gui:~/helm-charts$ microk8s helm3 pull bitnami/keycloak
lara@kube-master-gui:~/helm-charts$ ls
keycloak-1.2.0.tgz
```

Chart package structure



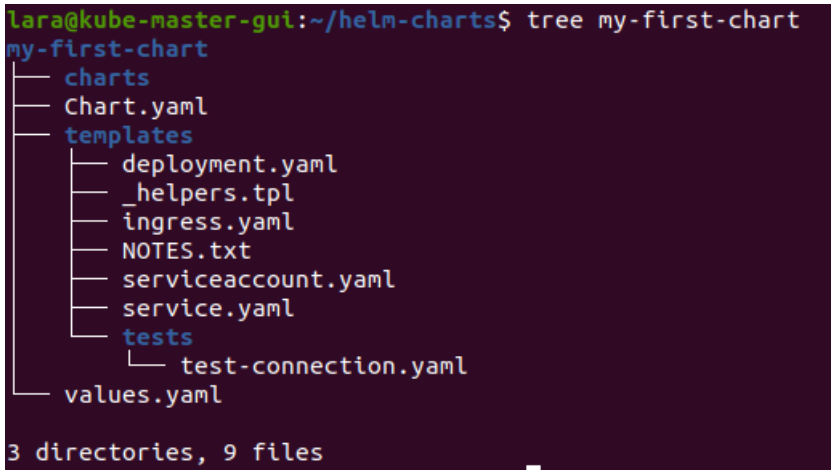
A chart can include other charts.
A chart has a file Chart.yaml and a file values.yaml that separate the definition from the configuration.
A chart has a templates section

Creating Helm Charts

To create a new Helm chart, you can start from the template generated by the command:

```
helm create [chart-name]
```

This creates 9 files in 3 directories as shown below:



Installing the newly created Helm chart

The newly created Helm chart can be installed with the command:

```
helm install [RELEASE-NAME] [CHART-DIRECTORY]
```

Then you can list the release with:

```
helm ls
```

```
lara@kube-master-gui:~/helm-charts$ microk8s helm3 install my-first-chart-release-1 ./my-first-chart
NAME: my-first-chart-release-1
LAST DEPLOYED: Sun Jan 17 21:42:54 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=my-first-chart,app.kubernetes.io/instance=my-first-chart-release-1" -o jsonpath="{.items[0].metadata.name}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:80
lara@kube-master-gui:~/helm-charts$ microk8s helm3 ls
NAME                    NAMESPACE       REVISION        UPDATED                                 STATUS          CHART                   APP VERSION
my-first-chart-release-1        default         1               2021-01-17 21:42:54.86879575 +0100 CET  deployed        my-first-chart-0.1.0    1.16.0
```

Chart.yaml

Chart.yaml is mandatory. It gets created as the following skeleton.

The **type** can be either **application** or **library**. Applications can be deployed standalone, while libraries can only be deployed as dependencies of applications.

In the Chart.yaml file, you can declare dependencies on other Helm charts.

```yaml
apiVersion: v2
name: my-first-chart
description: A Helm chart for Kubernetes

# A chart can be either an 'application' or a 'library' chart.
#
# Application charts are a collection of templates that can be packaged into versioned archives
# to be deployed.
#
# Library charts provide useful utilities or functions for the chart developer. They're included as
# a dependency of application charts to inject those utilities and functions into the rendering
# pipeline. Library charts do not define any templates and therefore cannot be deployed.
type: application

# This is the chart version. This version number should be incremented each time you make changes
# to the chart and its templates, including the app version.
version: 0.1.0

# This is the version number of the application being deployed. This version number should be
# incremented each time you make changes to the application.
appVersion: 1.16.0
```

Chart dependencies

This is the text of the **Chart.yaml** of the Keycloak chart.

In the dependencies field it lists two charts, **common** and **postgresql** that are required by this chart.

Dependencies may include **tags** and **conditions**.

The **tags** field contains a list of labels. They can be enabled or disabled in the values.yaml of the top parent chart.

The **condition** field contains one or more YAML paths (delimited by commas). If this path exists in the values.yaml of the top parent chart and resolves to a boolean value, the chart will be enabled or disabled based on that boolean value.
In this case the top-chart values.yaml contains:

```
postgresql:
enabled: true
```

```
annotations:
  category: DeveloperTools
apiVersion: v2
appVersion: 11.0.3
dependencies:
- name: common
  repository: https://charts.bitnami.com/bitnami
  tags:
  - bitnami-common
  version: 1.x.x
- condition: postgresql.enabled
  name: postgresql
  repository: https://charts.bitnami.com/bitnami
  version: 10.x.x
description: Keycloak is ….
home: https://www.keycloak.org
icon: https://..../keycloak-stack-110x117.png
keywords:
- keycloak
- access-management
maintainers:
- email: containers@bitnami.com
  name: Bitnami
name: keycloak
sources:
- https://github.com/bitnami/bitnami-docker-keycloak
- https://github.com/keycloak/keycloak
version: 1.2.0
```

Helm chart templates

In the templates folder you find template files that are defined following the syntax of Go templates.

Look at the **service.yaml**. It has the typical structure of a Kubernetes Service yaml file, but the values are expressed as **actions**, which are strings enclosed in {{ }}. Action may define data evaluations or control structures.
All the text outside of actions is copied to the output unchanged.
The **dot** in these actions represents the current location in the structure as execution proceeds.
To see what this template will generate, you can run this command from the parent directory of the chart:
`helm install --dry-run --debug ./my-first-chart  --generate-name`
It generates a single file, of which the service definition part is shown:

```
apiVersion: v1
kind: Service
metadata:
  name: {{ include "my-first-chart.fullname" . }}
  labels:
    {{- include "my-first-chart.labels" . | nindent 4 }}
spec:
  type: {{ .Values.service.type }}
  ports:
    - port: {{ .Values.service.port }}
      targetPort: http
      protocol: TCP
      name: http
  selector:
    {{- include "my-first-chart.selectorLabels" . | nindent 4 }}
```

```
# Source: my-first-chart/templates/service.yaml
apiVersion: v1
kind: Service
metadata:
  name: my-first-chart-1610825031
  labels:
    helm.sh/chart: my-first-chart-0.1.0
    app.kubernetes.io/name: my-first-chart
    app.kubernetes.io/instance: my-first-chart-1610825031
    app.kubernetes.io/version: "1.16.0"
    app.kubernetes.io/managed-by: Helm
spec:
  type: ClusterIP
  ports:
    - port: 80
      targetPort: http
      protocol: TCP
      name: http
  selector:
    app.kubernetes.io/name: my-first-chart
    app.kubernetes.io/instance: my-first-chart-1610825031
---
```

.Chart and .Values objects

The template in service.yaml makes use of the Helm-specific objects **.Chart** and **.Values**.

The **.Chart** object provides metadata such as name and version.

The **.Values** object exposes configuration that can be set at deployment time.

The default values for this object are defined in the **values.yaml** file.

If a user wanted to change the default configuration, they could override it on the command-line.

For example, if the template contains the line: `type: {{ .Values.service.type }}`
the user can override the value by referring to service.type in the command line below:

```
helm install --dry-run --debug ./mychart --set service.type=NodePort –generate-name
```

You can also specify a YAML file containing overrides with the --values option.

Customizing the default values in a Helm chart

The actual values in the **.Values** object may come form the following sources:

1. The **values.yaml** file in the chart

2. If this is a subchart, the **values.yaml** file of a parent chart

3. A values file supplied with the -f flag to the following commands:

```
helm install -f myvals.yaml ./mychart
helm upgrade -f myvals.yaml ./mychart
```

4. Individual parameters passed with `--set`:

```
helm install --set foo=bar ./mychart
```

The options with higher order number override values in the options with lower order number.

If you need to delete a key from the default values, you may override the value of the key to be null, in which case Helm will remove the key from the overridden values merge. Example:

```
--set livenessProbe.httpGet=null
```

Packaging a Helm release

Before upgrading a Helm release, let's package our first version of the Helm chart with:

`helm package [CHART_DIRECTORY]`

```
lara@kube-master-gui:~/helm-charts$ microk8s helm3 package ./my-first-chart/
Successfully packaged chart and saved it to: /home/lara/helm-charts/my-first-chart-0.1.0.tgz
lara@kube-master-gui:~/helm-charts$ ls
keycloak  keycloak-1.2.0.tgz  my-first-chart  my-first-chart-0.1.0.tgz
```

Now edit the Chart.yaml and increase the version from 0.1.0 to 0.1.1 and appVersion from 1.1.4.2 to latest

```
# This is the chart version. This version number should be incremented each time you make changes
# to the chart and its templates, including the app version.
version: 0.1.1

# This is the version number of the application being deployed. This version number should be
# incremented each time you make changes to the application.
appVersion: latest
```

appVersion is used in templates/deployment.yaml as the version of the image to pull from the registry:

```
    containers:
      - name: {{ .Chart.Name }}
        securityContext:
          {{- toYaml .Values.securityContext | nindent 12 }}
        image: "{{ .Values.image.repository }}:{{ .Chart.AppVersion }}"
        imagePullPolicy: {{ .Values.image.pullPolicy }}
```

If you now repeat the package command it creates my-first-chart.0.1.1.tgz that refers to the nginx:latest image.

Upgrading a Release

You can install the packaged Helm chart by supplying the name of the .tgz file instead of the directory name.
You can then upgrade the release to the second package with:

```
helm upgrade [RELEASE-NAME] [CHART-PACKAGE-TGZ]
```

Note that RELEASE-NAME must be the same name you used in the first installation.
Helm will increment the Revision number as you can see with `helm ls`.

```
lara@kube-master-gui:~/helm-charts$ microk8s helm3 install my-first-chart-release my-first-chart-0.1.0.tgz
NAME: my-first-chart-release
LAST DEPLOYED: Sun Jan 17 22:02:48 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=my-first-chart,app.kubernetes.io/instance=my-first-chart-release" -o jsonpath="{.items[0].metadata.name}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:80
lara@kube-master-gui:~/helm-charts$ microk8s helm3 ls
NAME                    NAMESPACE       REVISION        UPDATED                                 STATUS          CHART                   APP VERSION
my-first-chart-release  default         1               2021-01-17 22:02:48.803749963 +0100 CET deployed        my-first-chart-0.1.0    1.14.2
lara@kube-master-gui:~/helm-charts$ microk8s helm3 upgrade my-first-chart-release my-first-chart-0.1.1.tgz
Release "my-first-chart-release" has been upgraded. Happy Helming!
NAME: my-first-chart-release
LAST DEPLOYED: Sun Jan 17 22:03:15 2021
NAMESPACE: default
STATUS: deployed
REVISION: 2
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=my-first-chart,app.kubernetes.io/instance=my-first-chart-release" -o jsonpath="{.items[0].metadata.name}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:80
lara@kube-master-gui:~/helm-charts$ microk8s helm3 ls
NAME                    NAMESPACE       REVISION        UPDATED                                 STATUS          CHART                   APP VERSION
my-first-chart-release  default         2               2021-01-17 22:03:15.601929247 +0100 CET deployed        my-first-chart-0.1.1    latest
```

Rolling back a Helm Release

If you describe the deployment, you can see that a rolling update happened as follows:

```
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  9m56s  deployment-controller  Scaled up replica set my-first-chart-release-6cc79b9bb8 to 1
  Normal  ScalingReplicaSet  9m30s  deployment-controller  Scaled up replica set my-first-chart-release-67d797c7ff to 1
  Normal  ScalingReplicaSet  9m19s  deployment-controller  Scaled down replica set my-first-chart-release-6cc79b9bb8 to 0
```

You can rollback using the command:

```
helm rollback [RELEASE-NAME] [REVISION]
```

```
lara@kube-master-gui:~/helm-charts$ microk8s helm3 rollback my-first-chart-release 1
Rollback was a success! Happy Helming!
```

If you describe the deployment again you can see the additional events:

```
Events:
  Type    Reason             Age                From                   Message
  ----    ------             ----               ----                   -------
  Normal  ScalingReplicaSet  13m                deployment-controller  Scaled up replica set my-first-chart-release-67d797c7ff to 1
  Normal  ScalingReplicaSet  12m                deployment-controller  Scaled down replica set my-first-chart-release-6cc79b9bb8 to 0
  Normal  ScalingReplicaSet  76s (x2 over 13m)  deployment-controller  Scaled up replica set my-first-chart-release-6cc79b9bb8 to 1
  Normal  ScalingReplicaSet  65s                deployment-controller  Scaled down replica set my-first-chart-release-67d797c7ff to 0
```

Note that the REVISION was increased from 2 to 3 and the version of the chart and the app were reverted:

```
lara@kube-master-gui:~/helm-charts$ microk8s helm3 ls
NAME                    NAMESPACE       REVISION        UPDATED                                 STATUS          CHART                   APP VERSION
my-first-chart-release  default         3               2021-01-17 22:15:06.60469694 +0100 CET  deployed        my-first-chart-0.1.0    1.14.2
```

Helm Chart Hooks

Hooks represent insertion points in the lifecycle of a Helm Chart, where you execute custom code.

You insert the code by creating .yaml files with special annotations that indicate at which point in the lifecycle they should be executed (pre-install, post-install etc).

A typical use case for a hook is to run a Kubernetes Job or Pod.

For example, the hook could take a database backup, trigger a migration to upgrade a database schema etc.

The execution of a hook of type Job or Pod is *blocking:* the release pauses until that Job or Pod completes.

If the hook creates a ConfigMap or Secret, then the hook is considered *ready* when the resource has been created or updated.

Hooks can have weights, that determine in which order they should be executed within the same phase.

Available Helm Chart Hooks

Annotation Value	Description
pre-install	Executes after templates are rendered, but before any resources are created in Kubernetes
post-install	Executes after all resources are loaded into Kubernetes
pre-delete	Executes on a deletion request before any resources are deleted from Kubernetes
post-delete	Executes on a deletion request after all of the release's resources have been deleted
pre-upgrade	Executes on an upgrade request after templates are rendered, but before any resources are updated
post-upgrade	Executes on an upgrade request after all resources have been upgraded
pre-rollback	Executes on a rollback request after templates are rendered, but before any resources are rolled back
post-rollback	Executes on a rollback request after all resources have been modified
test	Executes when the Helm test subcommand is invoked (`view test docs`)

https://helm.sh/docs/topics/charts_hooks/

Example hook creation

```
hcluser@BLMYCLDDL31451:~/charts$ microk8s helm3 create sample-hooks
Creating sample-hooks
hcluser@BLMYCLDDL31451:~/charts$ cd sample-hooks
hcluser@BLMYCLDDL31451:~/charts/sample-hooks$ cd templates
hcluser@BLMYCLDDL31451:~/charts/sample-hooks/templates$ ls
deployment.yaml  _helpers.tpl  hpa.yaml  ingress.yaml  NOTES.txt  serviceaccount.yaml  service.yaml  tests
hcluser@BLMYCLDDL31451:~/charts/sample-hooks/templates$ nano pre-install.yaml
hcluser@BLMYCLDDL31451:~/charts/sample-hooks/templates$ cp pre-install.yaml post-install.yaml
hcluser@BLMYCLDDL31451:~/charts/sample-hooks/templates$ nano post-install.yaml
```

1. Create a chart
2. Create hook files in the templates folder

3. The hook file can be a Pod yaml file, or a Job or some other resource.

4. It must have the annotation: **`helm.sh/hook`** with one of the available hook values.

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-postinstall-hook
  annotations:
    "helm.sh/hook": "post-install"
spec:
  containers:
  - name: sample-postinstall-hook--container
    image: busybox
    imagePullPolicy: IfNotPresent
    command: ['sh', '-c', 'echo The sample post-install hook Pod is running']
  restartPolicy: Never
  terminationGracePeriodSeconds: 0
```

Verification of the hook execution

To verify that hook worked, install the Hem chart.
Since these hooks were supposed to create pods, check what pods exist and note that the two hook pods still exist but are in completed state.
After you uninstall the chart, the two hook pods remain there.
Since these hooks were just printing a message, you can view the message in the hook pod logs.

```
hcluser@BLMYCLDDL31451:~/charts$ sudo microk8s helm3 install my-sample-hooks ./sample-hooks
NAME: my-sample-hooks
LAST DEPLOYED: Fri Jan 20 09:39:17 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=sample-hooks,app.kubernetes.io/instance=my-sample-hooks" -o jsonpath="{.items[0].metadata.name}")
  export CONTAINER_PORT=$(kubectl get pod --namespace default $POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:$CONTAINER_PORT
hcluser@BLMYCLDDL31451:~/charts$ sudo microk8s kubectl get pods
NAME                          READY   STATUS      RESTARTS   AGE
sample-preinstall-hook        0/1     Completed   0          15s
my-sample-hooks-557b8bd4fc-6q9nk   1/1     Running     0          12s
sample-postinstall-hook       0/1     Completed   0          12s
hcluser@BLMYCLDDL31451:~/charts$ sudo microk8s helm3 uninstall my-sample-hooks
release "my-sample-hooks" uninstalled
hcluser@BLMYCLDDL31451:~/charts$ sudo microk8s kubectl get pods
NAME                    READY   STATUS      RESTARTS   AGE
sample-preinstall-hook    0/1     Completed   0          40s
sample-postinstall-hook   0/1     Completed   0          37s
hcluser@BLMYCLDDL31451:~/charts$ sudo microk8s kubectl logs sample-preinstall-hook
The sample pre-install hook Pod is running
```

Cleaning up resources created by hooks

Hooks are managed separately from the other resources created by a Helm Chart.

If you create resources in a hook, `helm uninstall` does not *currently* remove these resources.

Any hook resources that must never be deleted should be annotated with
`helm.sh/resource-policy: keep`
(in case later helm implements garbage collection).

The annotation:
`helm.sh/hook-delete-policy`
can be used to change the behavior of Helm:

Annotation Value	Description
before-hook-creation	Delete the previous resource before a new hook is launched (default)
hook-succeeded	Delete the resource after the hook is successfully executed
hook-failed	Delete the resource if the hook failed during execution

Example of hook resource cleaning

If you specify the annotation:
`helm.sh/hook-delete-policy`
with the value:
`Hook-succeeded`
as soon as the hook pod completed, it is deleted automatically (no need to uninstall the Helm chart).

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-postinstall-hook
  annotations:
    "helm.sh/hook": "post-install"
    "helm.sh/hook-delete-policy": "hook-succeeded"
spec:
  containers:
  - name: sample-postinstall-hook--container
    image: busybox
    imagePullPolicy: IfNotPresent
    command: ['sh', '-c', 'echo The sample post-install
hook Pod is running']
  restartPolicy: Never
  terminationGracePeriodSeconds: 0
```

```
hcluser@BLMYCLDDL31451:~/charts$ sudo microk8s helm3 install my-sample-hooks ./sample-hooks
NAME: my-sample-hooks
LAST DEPLOYED: Fri Jan 20 10:00:23 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=sample-hooks,app.kubernetes.io/instance=my-sample-hooks" -o jsonpath="{.items[0].metadata.name}")
  export CONTAINER_PORT=$(kubectl get pod --namespace default $POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:$CONTAINER_PORT
hcluser@BLMYCLDDL31451:~/charts$ sudo microk8s kubectl get pods
NAME                              READY   STATUS      RESTARTS   AGE
sample-preinstall-hook            0/1     Completed   0          13s
my-sample-hooks-557b8bd4fc-nm5s8  1/1     Running     0          9s
```

Case study: database migrations

Database migration is a term that refers to the database schema changes required when an application gets installed for the first time or upgraded/downgraded.

When using Python, database migrations:

- are included in Django: https://docs.djangoproject.com/en/4.1/topics/migrations/
- can be implemented with Alembic: https://alembic.sqlalchemy.org/en/latest/

In Kubernetes, it is possible to use Helm Chart hooks that contain Kubernets jobs and implement database migrations after installing a database.

An alternative approach could be to execute a Job that implements the migration and create an application deployment with an init container that sleeps until the completion of the job. This way, the deployment main container will only start executing after the job completed.

https://andrewlock.net/deploying-asp-net-core-applications-to-kubernetes-part-7-running-database-migrations/

Example: database migration with Postgresql and post-install hook 1/8

Ensure that the hostpath-storage is enables and verify that the default storage class exists.

```
hcluser@BLMYCLDDL31451:~/charts$ sudo microk8s enable hostpath-storage
Infer repository core for addon hostpath-storage
Enabling default storage class.
WARNING: Hostpath storage is not suitable for production environments.

deployment.apps/hostpath-provisioner created
storageclass.storage.k8s.io/microk8s-hostpath created
serviceaccount/microk8s-hostpath created
clusterrole.rbac.authorization.k8s.io/microk8s-hostpath created
clusterrolebinding.rbac.authorization.k8s.io/microk8s-hostpath created
Storage will be available soon.
```

```
hcluser@BLMYCLDDL31451:~/charts$ sudo microk8s kubectl get sc
NAME                          PROVISIONER            RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
microk8s-hostpath (default)   microk8s.io/hostpath   Delete          WaitForFirstConsumer   false                  36s
```

Pull a postgresql chart to your local system so you can modify the source code:

```
microk8s helm3 repo add my-bitnami-repo https://charts.bitnami.com/bitnami
microk8s helm3 pull my-bitnami-repo/postgresql
```

```
hcluser@BLMYCLDDL31451:~/charts$ microk8s helm3 pull my-bitnami-repo/postgresql
hcluser@BLMYCLDDL31451:~/charts$ ls
postgresql-12.1.9.tgz
```

Extract the contents of the chart (output truncated):

```
tar -xvf postgresql-12.1.9.tgz
```

```
hcluser@BLMYCLDDL31451:~/charts$ tar -xvf postgresql-12.1.9.tgz
postgresql/Chart.yaml
postgresql/Chart.lock
postgresql/values.yaml
postgresql/values.schema.json
postgresql/templates/NOTES.txt
postgresql/templates/_helpers.tpl
postgresql/templates/extra-list.yaml
postgresql/templates/networkpolicy-egress.yaml
postgresql/templates/primary/configmap.yaml
```

Example: database migration with Postgresql and post-install hook 2/8

Edit values.yaml and specify your desired values for:

```
auth:
      postgresPassword: "postgres_passwd"
      username: "user1"
      password: "user1_passwd"
      database: "postgresdb"
```

Else, the chart will generate random values which you can then find inspecting the generated ConfigMaps/Secrets.

```
## @section Global parameters
## Please, note that this will override the parameters, including dependencies, configured to use the global value
##
global:
  ## @param global.imageRegistry Global Docker image registry
  ##
  imageRegistry: ""
  ## @param global.imagePullSecrets Global Docker registry secret names as an array
  ## e.g.
  ## imagePullSecrets:
  ##   - myRegistryKeySecretName
  ##
  imagePullSecrets: []
  ## @param global.storageClass Global StorageClass for Persistent Volume(s)
  ##
  storageClass: ""
  postgresql:
    ## @param global.postgresql.auth.postgresPassword Password for the "postgres" admin user (overrides `auth.postgresPassword`)
    ## @param global.postgresql.auth.username Name for a custom user to create (overrides `auth.username`)
    ## @param global.postgresql.auth.password Password for the custom user to create (overrides `auth.password`)
    ## @param global.postgresql.auth.database Name for a custom database to create (overrides `auth.database`)
    ## @param global.postgresql.auth.existingSecret Name of existing secret to use for PostgreSQL credentials (overrides `auth.exis
    ## @param global.postgresql.auth.secretKeys.adminPasswordKey Name of key in existing secret to use for PostgreSQL credentials (
    ## @param global.postgresql.auth.secretKeys.userPasswordKey Name of key in existing secret to use for PostgreSQL credentials (o
    ## @param global.postgresql.auth.secretKeys.replicationPasswordKey Name of key in existing secret to use for PostgreSQL credent
    ##
    auth:
      postgresPassword: "postgres_passwd"
      username: "user1"
      password: "user1_passwd"
      database: "postgresdb"
```

Example: database migration with Postgresql and post-install hook 3/8

Install the chart and take note of the helpful messages in the output

```
hcluser@BLMYCLDDL31451:~/charts$ sudo microk8s helm3 install my-postgresql ./postgresql
NAME: my-postgresql
LAST DEPLOYED: Fri Jan 20 10:08:32 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: postgresql
CHART VERSION: 12.1.9
APP VERSION: 15.1.0

** Please be patient while the chart is being deployed **

PostgreSQL can be accessed via port 5432 on the following DNS names from within your cluster:

    my-postgresql.default.svc.cluster.local - Read/Write connection

To get the password for "postgres" run:

    export POSTGRES_ADMIN_PASSWORD=$(kubectl get secret --namespace default my-postgresql -o jsonpath="{.data.postgres-password}" | base64 -d)

To get the password for "user1" run:

    export POSTGRES_PASSWORD=$(kubectl get secret --namespace default my-postgresql -o jsonpath="{.data.password}" | base64 -d)

To connect to your database run the following command:

    kubectl run my-postgresql-client --rm --tty -i --restart='Never' --namespace default --image docker.io/bitnami/postgresql:15.1.0-debian-11-r20 --env="PGPASSWORD=$POSTGRES_PASSWORD" \
      --command -- psql --host my-postgresql -U user1 -d postgresdb -p 5432

    > NOTE: If you access the container using bash, make sure that you execute "/opt/bitnami/scripts/postgresql/entrypoint.sh /bin/bash" in order to avoid the error "psql: local user with ID 1001} does not exist"

To connect to your database from outside the cluster execute the following commands:

    kubectl port-forward --namespace default svc/my-postgresql 5432:5432 &
    PGPASSWORD="$POSTGRES_PASSWORD" psql --host 127.0.0.1 -U user1 -d postgresdb -p 5432
```

Example: database migration with Postgresql and post-install hook 4/8

Check that the pod and the StatefulSet are in correct state:

```
hcluser@BLMYCLDDL31451:~/charts$ sudo microk8s kubectl get all
NAME                    READY   STATUS    RESTARTS   AGE
pod/my-postgresql-0     1/1     Running   0          29m

NAME                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/kubernetes            ClusterIP   10.152.183.1    <none>        443/TCP    46h
service/my-postgresql-hl      ClusterIP   None            <none>        5432/TCP   29m
service/my-postgresql         ClusterIP   10.152.183.84   <none>        5432/TCP   29m

NAME                                READY   AGE
statefulset.apps/my-postgresql      1/1     29m
```

Note that because this is a StatefulSet, there is a corresponding Headless Service with ClusterIP set to None. The corresponding endpoints refer to the pod IP address:

```
hcluser@BLMYCLDDL31451:~/charts$ sudo microk8s kubectl get endpoints -o wide
NAME               ENDPOINTS              AGE
kubernetes         10.115.154.183:16443   46h
my-postgresql-hl   10.1.171.30:5432       3m19s
my-postgresql      10.1.171.30:5432       3m19s
hcluser@BLMYCLDDL31451:~/charts$ sudo microk8s kubectl get pods -o wide
NAME              READY   STATUS    RESTARTS   AGE     IP            NODE             NOMINATED NODE   READINESS GATES
my-postgresql-0   1/1     Running   0          4m25s   10.1.171.30   blmycldd131451   <none>           <none>
```

Example: database migration with Postgresql and post-install hook 5/8

Configure the environment variables (so you never type the passwords):

```
export POSTGRES_ADMIN_PASSWORD=$(kubectl get secret --namespace default my-postgresql -o jsonpath="{.data.postgres-password}" | base64 -d)
export POSTGRES_PASSWORD=$(kubectl get secret --namespace default my-postgresql -o jsonpath="{.data.password}" | base64 -d)
```

```
hcluser@BLMYCLDDL31451:~/charts$ export POSTGRES_ADMIN_PASSWORD=$(sudo microk8s kubectl get secret --namespace default my-postgresql -o jsonpath="{.data.postgres-password}" | base64 -d)
[sudo] password for hcluser:
hcluser@BLMYCLDDL31451:~/charts$ export POSTGRES_PASSWORD=$(sudo microk8s kubectl get secret --namespace default my-postgresql -o jsonpath="{.data.password}" | base64 -d)
```

Run a pod that uses the postgresql image but executes psql (client interface). The hostname `my-postgresql` is actually the service name (this only works if the CoreDNS add-on is enabled).

```
sudo microk8s kubectl run my-postgresql-client --rm --tty -i --restart='Never' --namespace default --image docker.io/bitnami/postgresql:15.1.0-debian-11-r20 --env="PGPASSWORD=$POSTGRES_PASSWORD" --command -- psql --host my-postgresql -U user1 -d postgresdb -p 5432
```

```
hcluser@BLMYCLDDL31451:~/charts$ sudo microk8s kubectl run my-postgresql-client --rm --tty -i --restart='Never' --namespace default --image docker.io/bitnami/postgresql:15.1.0-debian-11-r20 --env="PGPASSWORD=$POSTGRES_PASSWORD"       --c
ommand -- psql --host my-postgresql -U user1 -d postgresdb -p 5432
If you don't see a command prompt, try pressing enter.

postgresdb=> \l
                                                List of databases
    Name    |  Owner   | Encoding |   Collate   |    Ctype    | ICU Locale | Locale Provider |   Access privileges
------------+----------+----------+-------------+-------------+------------+-----------------+-----------------------
 postgres   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |            | libc            |
 postgresdb | user1    | UTF8     | en_US.UTF-8 | en_US.UTF-8 |            | libc            | =Tc/user1            +
            |          |          |             |             |            |                 | user1=CTc/user1
 template0  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |            | libc            | =c/postgres          +
            |          |          |             |             |            |                 | postgres=CTc/postgres
 template1  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |            | libc            | =c/postgres          +
            |          |          |             |             |            |                 | postgres=CTc/postgres
(4 rows)

postgresdb=>
```

Example: database migration with Postgresql and post-install hook 6/8

Verify that you can connect to the database and list the tables:

```
\c postgresdb
\dt
```

(In postgresql, ‘relation’ means Table)

```
hcluser@BLMYCLDDL31451:~/charts$ sudo microk8s kubectl run my-postgresql-client --rm --tty -i --restart='Never' --namespace default --image docker.io/bitnami/postgresql:15.1.0-debian-11-r20 --env="PGPASSWORD=$POSTGRES_PASSWORD"       --c
ommand -- psql --host my-postgresql -U user1 -d postgresdb -p 5432
If you don't see a command prompt, try pressing enter.

postgresdb=>  \c postgresdb
You are now connected to database "postgresdb" as user "user1".
postgresdb=>  \dt
Did not find any relations.
postgresdb=> exit
```

Now we want to create a table from a post-install hook.

Create a file users_sql.yaml which defnes a config map with a sql file (users.sql) that create a users table.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: users-sql
  namespace: default
  annotations:
    "helm.sh/hook": post-install
    "helm.sh/hook-weight": "1"
data:
  users.sql: |
    CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    age INT,
    first_name TEXT,
    last_name TEXT,
    email TEXT UNIQUE NOT NULL
    );
```

Example: database migration with Postgresql and post-install hook 7/8

Create a job with higher weight than the config map and that executes the psql command with a file in input.
The file is obtained by mounting the config map as a volume.
Improvement: templatize the command using the variables already defined in the Helm chart.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: init-db
  namespace: default
  annotations:
    "helm.sh/hook": post-install
    "helm.sh/hook-weight": "2"
>spec:
  template:
    metadata:
      name:  init-db
      labels:
        app: init-postgresdb
    spec:
      containers:
      - name: init-db
        image: "docker.io/bitnami/postgresql:15.1.0-debian-11-r20"
        command: [ "psql", "-h", "my-postgresql", "-U", "postgres", "-d", "postgresdb", "-f", "/sql-command/users.sql" ]
        volumeMounts:
        - name: sql-command
          mountPath: /sql-command
        env:
          - name: PGPASSWORD
            value: "postgres_passwd"
      volumes:
        - name: sql-command
          configMap:
            # Provide the name of the ConfigMap containing the files you want
            # to add to the container
            name: users-sql
      restartPolicy: OnFailure
```

Example: database migration with Postgresql and post-install hook 8/8

Now uninstall the Helm Chart, if present, and install it again.

Then connect with the psql client in a container (or from the host) and verify that the database table was created.

```
hcluser@BLMYCLDDL31451:~$ sudo microk8s kubectl run my-postgresql-client --rm --tty -i --restart='Never' -
-namespace default --image docker.io/bitnami/postgresql:15.1.0-debian-11-r20 --env="PGPASSWORD=$POSTGRES_P
ASSWORD" --command -- psql --host my-postgresql -U user1 -d postgresdb -p 5432
If you don't see a command prompt, try pressing enter.

postgresdb=> \c postgresdb
You are now connected to database "postgresdb" as user "user1".
postgresdb=> \dt
         List of relations
 Schema | Name  | Type  |  Owner
--------+-------+-------+----------
 public | users | table | postgres
(1 row)

postgresdb=>
```

Note that the ConfigMap, the Job and the Pod created by executing the two hooks are still present after the installation completed.

```
hcluser@BLMYCLDDL31451:~$ sudo microk8s kubectl get cm
NAME               DATA   AGE
kube-root-ca.crt   1      2d3h
users-sql          1      4m17s
hcluser@BLMYCLDDL31451:~$ sudo microk8s kubectl get job
NAME      COMPLETIONS   DURATION   AGE
init-db   1/1           24s        4m22s
hcluser@BLMYCLDDL31451:~$ sudo microk8s kubectl get pod
NAME              READY   STATUS      RESTARTS   AGE
my-postgresql-0   1/1     Running     0          4m28s
init-db-rsp2v     0/1     Completed   2          4m28s
hcluser@BLMYCLDDL31451:~$
```

HCLSoftware

hcltechsw.com