MENU

FOSSTechNix

MENU

Kubernetes Configmap with Helm Chart

January 28, 2024 by Prasad Hole

Home » Kubernetes Configmap with Helm Chart

In this article, we explore the integration of ConfigMap, a key Kubernetes resource for managing configuration data, within a Helm chart. The Helm chart includes deployment, service, and ingress manifests, and our goal is to enhance the flexibility and customization of our deployment using ConfigMap | Kubernetes Configmap with Helm Chart.

Table of Contents

Prerequisites

- AWS Account with Ubuntu 22.04 LTS EC2 Instance
- Minikube and kubectl Installed

Install Minikube and kubectl by following the official documentation for your operating system:

Minikube Installation Guide

Install Minikube on Ubuntu 22.04 LTS

- Helm Installed:

Install Helm by following the official documentation:

Helm Installation Guide

How to use configmap in Kubernetes with Helm Chart

We begin by creating a dedicated `configmap.yaml` file within the Helm chart's "templates" folder. This ConfigMap contains HTML content that serves as a custom index page for an Nginx deployment. Subsequently, we modify the deployment manifest to incorporate this ConfigMap as a volume, ensuring the dynamic injection of configuration data into the Nginx container.

As part of the Helm chart upgrade process, we witness the seamless transition from the default Nginx welcome page to our custom content sourced from the ConfigMap. Additionally, we delve into the use of Go template syntax within the `configmap.yaml` file, enabling parameterization and enhanced template rendering.

Create a helm chart

Firstly, let's create a Helm chart

```
helm create demo
```

```
ubuntu@ip-172-31-5-188:~$ helm create demo
Creating demo
```

open the chart

```
cd demo
```

```
ubuntu@ip-172-31-5-188:~$ cd demo
```

run the following command to see the files in it.

```
ls
```

```
ubuntu@ip-172-31-5-188:~/demo$ ls
Chart.yaml  charts  templates  values.yaml
```

exit the directory

```
cd
```

now let's deploy the chart

```
helm install mychart demo
```

```
ubuntu@ip-172-31-5-188:~$ helm install mychart demo
NAME: mychart
LAST DEPLOYED: Fri Jan 26 15:37:58 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=demo,app.kubernetes.io/instance=mychart" -o jsonpath="{.items[0].metadata.
name}")
  export CONTAINER_PORT=$(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
```

now lets see the pod deployment

```
kubectl get pods
```

```
ubuntu@ip-172-31-5-188:~$ kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
mychart-demo-b6774cf5c-7q2fl   1/1     Running   0          10s
```

get the services

```
kubectl get svc
```

```
ubuntu@ip-172-31-5-188:~$ kubectl get svc
NAME           TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes     ClusterIP   10.96.0.1      <none>        443/TCP   2m45s
mychart-demo   ClusterIP   10.108.47.1    <none>        80/TCP    16s
```

Use port-forward to forward the local port to service.

```
kubectl port-forward --address 0.0.0.0 svc/mychart-demo 8888:80
```

```
ubuntu@ip-172-31-5-188:~$ kubectl port-forward --address 0.0.0.0 svc/mychart-demo 8888:80
Forwarding from 0.0.0.0:8888 -> 80
```

Check browser. Then you'll see Nginx welcome page.

To access the application on Browser write the ip address:port number in url.

ip address is the public ip address of your **Minikube EC2 instance** created on **AWS** and port number which is `8888` which we have used in forwarding mychart-demo pod.

Create Configmap in Helm Chart

Firstly, we will create a configMap.yaml file in templates folder with the configuration given below.

Enter the demo and templates directory

```
cd demo/templates
```

```
ubuntu@ip-172-31-5-188:~$ cd demo/templates
```

create a configMap.yaml here in templates directory.

```
nano configMap.yaml
```

```
ubuntu@ip-172-31-5-188:~/demo/templates$ nano configMap.yaml
```

now add the following content into it.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-configmap
data:
```

```
  index.html:
    <h1>Welcome to the Helm !!!</h1>
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-configmap
data:
  index.html: |
    <h1>Welcome to the Helm!!!</h1>
```

And mount this configMap volume to deployment

```
nano deployment.yaml
```

```
ubuntu@ip-172-31-5-188:~/demo/templates$ nano deployment.yaml
```

```
    volumeMounts:
      - name: nginx
        mountPath: /usr/share/nginx/html
volumes:
  - name: nginx
    configMap:
      name: nginx-configmap
```

```
    containers:
      - name: {{ .Chart.Name }}
        securityContext:
          {{- toYaml .Values.securityContext | nindent 12 }}
        image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default .Chart.AppVersion }}"
        imagePullPolicy: {{ .Values.image.pullPolicy }}
        ports:
          - name: http
            containerPort: {{ .Values.service.port }}
            protocol: TCP
        livenessProbe:
          {{- toYaml .Values.livenessProbe | nindent 12 }}
        readinessProbe:
          {{- toYaml .Values.readinessProbe | nindent 12 }}
        resources:
          {{- toYaml .Values.resources | nindent 12 }}
        volumeMounts:
          - name: nginx
            mountPath: /usr/share/nginx/html
    volumes:
      - name: nginx
        configMap:
          name: nginx-configmap
    {{- with .Values.nodeSelector }}
    nodeSelector:
      {{- toYaml . | nindent 8 }}
    {{- end }}
    {{- with .Values.affinity }}
```

exit the directories

```
cd
```

Then upgrade the chart

```
helm upgrade mychart demo
```

```
ubuntu@ip-172-31-5-188:~$ helm upgrade mychart demo
Release "mychart" has been upgraded. Happy Helming!
NAME: mychart
LAST DEPLOYED: Fri Jan 26 15:58:34 2024
NAMESPACE: default
STATUS: deployed
REVISION: 2
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=demo,app.kubernetes.io/instance=mychart" -o jsonpath="{.items[0].metadata.
name}")
  export CONTAINER_PORT=$(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
```

Use port-forward to forward the local port to service.

```
kubectl port-forward --address 0.0.0.0 svc/mychart-demo 8888:80
```

```
ubuntu@ip-172-31-5-188:~$ kubectl port-forward --address 0.0.0.0 svc/mychart-demo 8888:80
Forwarding from 0.0.0.0:8888 -> 80
```

And check browser again, you can see the welcome page was changed to our custom page.

Welcome to the Helm!!!

Using Go Template syntax

Now again open the demo and templates directory

```
cd demo/templates
```

```
ubuntu@ip-172-31-5-188:~$ cd demo/templates
```

open the configMap.yaml

```
nano  configMap.yaml
```

```
ubuntu@ip-172-31-5-188:~/demo/templates$ nano configMap.yaml
```

change the index.html

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-configmap
data:
  index.html: {{ .Values.pageContent }}
```

Now this time, Helm will render the message from the values.yaml, so let's add the parameter in values.yaml file.

first exit the templates directory.

```
cd ..
```

open the values.yaml file

```
nano values.yaml
```

add the following content into it.

```
pageContent: |
  <h1> Helm Template using Go syntax </h1>
```

```
# Additional volumes on the output Deployment definition.
volumes: []
# - name: foo
#   secret:
#     secretName: mysecret
#     optional: false

# Additional volumeMounts on the output Deployment definition.
volumeMounts: []
# - name: foo
#   mountPath: "/etc/foo"
#   readOnly: true

nodeSelector: {}

tolerations: []

affinity: {}

pageContent: |
  <h1> Helm Template using Go syntax </h1>
```

exit the directory

```
cd
```

Upgrade it and check it out with browser

```
helm upgrade mychart demo
```

```
ubuntu@ip-172-31-5-188:~$ helm upgrade mychart demo
Release "mychart" has been upgraded. Happy Helming!
NAME: mychart
LAST DEPLOYED: Fri Jan 26 16:21:30 2024
NAMESPACE: default
STATUS: deployed
REVISION: 3
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=demo,app.kubernetes.io/instance=mychart" -o jsonpath="{.items[0].metadata.
name}")
  export CONTAINER_PORT=$(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
```

Use port-forward to forward the local port to service.

```
kubectl port-forward --address 0.0.0.0 svc/mychart-demo 8888:80
```

```
ubuntu@ip-172-31-5-188:~$ kubectl port-forward --address 0.0.0.0 svc/mychart-demo 8888:80
Forwarding from 0.0.0.0:8888 -> 80
```

And check browser again,

Accessing file inside templates

Now again open the demo and templates directory

```
cd demo
```

Create a static directory and open it.

```
mkdir static
```

```
cd static
```

```
ubuntu@ip-172-31-5-188:~/demo/templates$ mkdir static
ubuntu@ip-172-31-5-188:~/demo/templates$ cd static
```

inside it create a index.html file and the following content into it

```
nano index.html
```

```
ubuntu@ip-172-31-5-188:~/demo/templates/static$ nano index.html
```

```
<h1> Helm is accessing files ! </h1>
```

```
<h1> Helm is accessing files ! </h1>
```

exit the static directory

```
cd ..
```

open the templates directory

```
cd templates
```

Modify configMap.yaml.

```
nano configMap.yaml
```

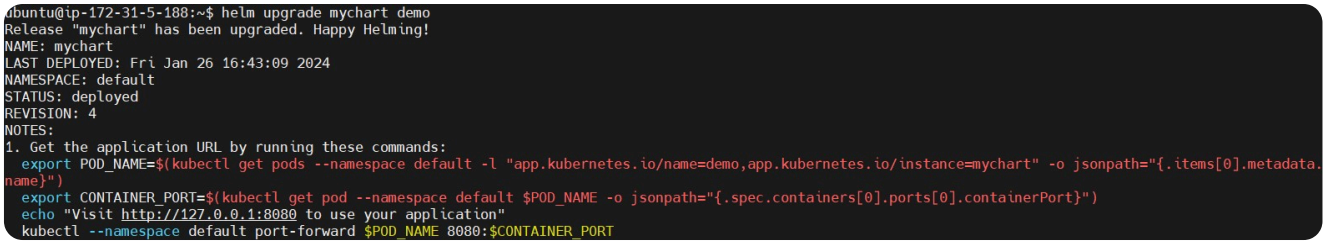
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-configmap
data:
  {{- (.Files.Glob "static/*").AsConfig | nindent 2 }}
```

exit the directory

```
cd
```

Upgrade it

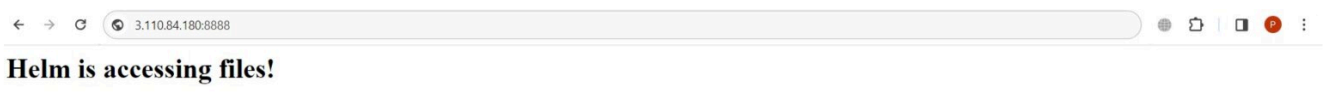
```
helm upgrade mychart demo
```



use kubectl port forward

```
kubectl port-forward --address 0.0.0.0 svc/mychart-demo 8888:80
```

check it out with browser again.



You can view the revisions created of your helm chart

```
helm history mychart
```

```
ubuntu@ip-172-31-8-22:~$ helm history mychart
REVISION        UPDATED                         STATUS          CHART           APP VERSION     DESCRIPTION
1               Sat Jan 27 12:43:11 2024        superseded      demo-0.1.0      1.16.0          Install complete
2               Sat Jan 27 12:46:58 2024        superseded      demo-0.1.0      1.16.0          Upgrade complete
3               Sat Jan 27 12:54:29 2024        superseded      demo-0.1.0      1.16.0          Upgrade complete
4               Sat Jan 27 12:54:34 2024        deployed        demo-0.1.0      1.16.0          Upgrade complete
```

Let's try to rollback the chart to the first revision.

```
helm rollback mychart 1
```

```
ubuntu@ip-172-31-5-188:~$ helm rollback mychart 1
Rollback was a success! Happy Helming!
```

Then you'll see Nginx welcome page come back and add another revision into the history as well.


```
helm history mychart
```

```
ubuntu@ip-172-31-8-22:~$ helm history mychart
REVISION        UPDATED                         STATUS          CHART           APP VERSION     DESCRIPTION
1               Sat Jan 27 12:56:12 2024        superseded      demo-0.1.0      1.16.0          Install complete
2               Sat Jan 27 12:56:17 2024        superseded      demo-0.1.0      1.16.0          Upgrade complete
3               Sat Jan 27 12:56:20 2024        superseded      demo-0.1.0      1.16.0          Upgrade complete
4               Sat Jan 27 12:56:22 2024        superseded      demo-0.1.0      1.16.0          Upgrade complete
5               Sat Jan 27 12:56:33 2024        deployed        demo-0.1.0      1.16.0          Rollback to 1
```

Conclusion:

In summary, this comprehensive guide takes you through the steps of creating a Helm chart, integrating configMap for enhanced configuration management, utilizing Go template syntax, accessing files within templates, and managing chart revisions efficiently.

Reference:

Kubernetes configmap with helm chart official page

‹ What is a Data Sources in Grafana?

› How to Deploy Application on Minikube using ArgoCD UI

Prasad Hole

Leave a Comment

Name *

Email *

Website

Post Comment

This site uses Akismet to reduce spam. Learn how your comment data is processed.

Search ...

RECENT POSTS

Deploy Django Application to AWS EC2 using Jenkins Pipeline

How to Configure Email Alerts in Grafana

Kubernetes Kustomize Tutorial with Examples

Create an Amazon ECS Cluster Using Terraform

How to Monitor Redis with Prometheus and Grafana

ABOUT FOSS TECHNIX

FOSS TechNix (Free ,Open Source Softwares and Technology Nix*) is a community site where you can find How-To Guides, Articles, Tips and Tricks for DevOps Tools, Linux, Databases, Clouds and Automation.

DEVOPS TOOLS & TIPS

What does DevOps Engineer do ?

Top 11 Open Source Monitoring Tools for Linux

TUTORIAL FOR BEGINNERS

Docker Tutorial for Beginners

Kubernetes Tutorial for Beginners

GitLab CI/CD Tutorial

Prometheus Tutorial for Beginners