Helm Chart

What is Helm Chart?

Helm Chart: An application package for Kubernetes, like traditional packages for package managers such as apt, yum, or npm. It is a set of files describing Kubernetes resources, such as deployments, services, and configuration files necessary for running an application on Kubernetes.

Helm Charts can be stored in remote repositories for easy sharing and distribution across Kubernetes clusters.

Several YAML files define Kubernetes resources for your application. You can create or modify these files based on your application's requirements. There are two main types of configuration files in Helm:

- Template files
- Values files

There are two types of values files in a Helm Chart:

- **values.yaml:** Contains default values for the templates in the Chart. It is typically included in every Chart.
- **User-supplied values files:** Can be used to override the default values in the values.yaml file when the Chart is installed.

Prerequisites

Before we begin creating our Helm Chart, make sure you have the following tools installed:

Helm: You will need Helm installed on your local machine.

Kubernetes Cluster: Ensure you have a running Kubernetes cluster to deploy your application. You can set up a local cluster using tools like Minikube or a cloud-based Kubernetes service.

Helm Chart example

We'll create a Helm Chart of a simple web application using NGINX.

1. Step 1: Initialize the Helm Chart

The first step is to initialize a new Helm Chart. You can do this using the helm create command.

```
helm create mywebapp



cd mywebapp
```

2. Step 2: Define your application in values.yaml

The *values.yaml* file contains default values for the templates in a Helm Chart. It is a YAML file that is typically included in every Chart. You can use it to override the defaults when the Chart is installed.

The following are some examples of the information that can be found in a *values.yaml* file:

- Image name and tag for a container image
- Number of replicas for a Deployment
- Port number for a Service
- Name of a PersistentVolumeClaim
- Environment variables for a container

```
osboxes@osboxes:~$ helm create mywebapp
Creating mywebapp
osboxes@osboxes:~$ cd my
my-git-reset-example/ mywebapp/
osboxes@osboxes:~$ cd mywebapp/
osboxes@osboxes:~/mywebapp$ ls
charts  Chart.yaml  templates  values.yaml
osboxes@osboxes:~/mywebapp$ cat values.yaml
# Default values for mywebapp.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

replicaCount: 1

image:
  repository: nginx
  pullPolicy: IfNotPresent
  # Overrides the image tag whose default is the chart appVersion.
  tag: ""

imagePullSecrets: []
nameOverride: ""
fullnameOverride: ""

serviceAccount:
  # Specifies whether a service account should be created
  create: true
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname template
  name: ""

podAnnotations: {}

podSecurityContext: {}
  # fsGroup: 2000
```

Below is an example of NGINX configuration values that you can configure in *values.yaml*:

```
replicaCount: 1  # Number of NGINX replicas to deploy


image:


  repository: nginx


  tag: latest  # NGINX Docker image tag


  pullPolicy: IfNotPresent  # Image pull policy
```

```
service:


  port: 80  # Port to expose the NGINX service on
```

A **ReplicaSet ensures that a specified number of pod replicas are running at any given time**.

`IfNotPresent`

the image is pulled only if it is not already present locally.

`Always`

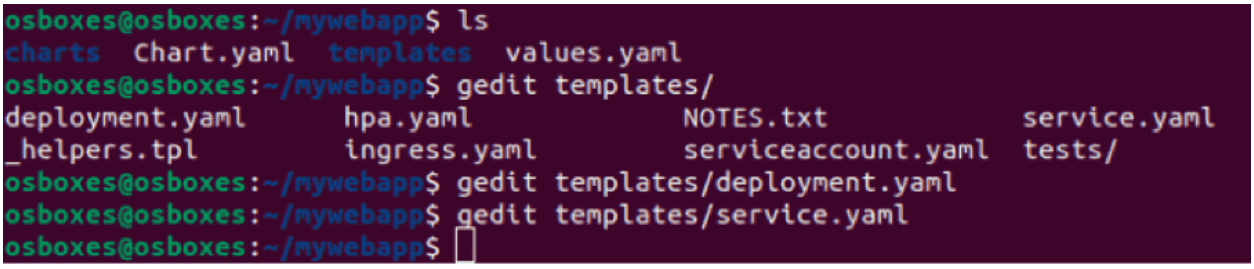
every time the kubelet launches a container, the kubelet queries the container image registry to resolve the name to an image digest. If the kubelet has a container image with that exact digest cached locally, the kubelet uses its cached image; otherwise, the kubelet pulls the image with the resolved digest, and uses that image to launch the container.

`Never`

the kubelet does not try fetching the image.

3. Step 3: Create deployment resources in deployment.yaml

In this example, we're creating a Kubernetes Deployment for the NGINX container.



Below is an example of a *deployment.yaml* file.

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: my-deployment

spec:

  replicas: 2

  selector:

    matchLabels:

      app: my-app

  template:

    metadata:

      labels:

        app: my-app

    spec:
```

```
      containers:

      - name: nginx

        image: nginx:latest

        ports:

        - containerPort: 80
```

4. Step 4: Create Service resources in services.yaml

```
apiVersion: v1

kind: Service

metadata:

  name: my-service

spec:

  selector:

    app: my-app

  ports:
```

```
- port: 80


  targetPort: 80
```

5. Step 5: Customize your Helm Chart in Chart.yaml

The following fields are available in the Chart.yaml file:

- apiVersion: The API version of the Chart
- name: The name of the Chart
- version: The version of the Chart
- kubeVersion: A SemVer range of compatible Kubernetes versions
- description: A single-sentence description of the Chart
- type: The type of the Chart
- keywords: A list of keywords about the Chart
- home: The URL of the Chart's home page
- sources: A list of URLs to source code for the Chart
- dependencies: A list of the Chart's dependencies

It must be located in the root directory of the Chart.

Here is an example of a Chart.yaml file:

```
- apiVersion: v2


- name: mywebapp


- description: Nginx Helm Chart for Kubernetes


- version: 1.0.0
```

- kubeVersion: ">=1.19.0"
- description: A Helm Chart for my application.
- type: application
- keywords: [my-application, helm-chart]
- home: https://example.com/my-application
- sources:
- - https://github.com/example/my-application
- dependencies:
- - name: nginx
- version: "1.23.1"
- apiVersion: v2
- name: mywebapp
- description: Nginx Helm Chart for Kubernetes
- version: 1.0.0

```
- kubeVersion: ">=1.19.0"
- description: A Helm Chart for my application.
- type: application
- keywords: [my-application, helm-chart]
- home: https://example.com/my-application
- sources:
-   - https://github.com/example/my-application
- dependencies:
-   - name: nginx
-     version: "1.23.1"
```

6. Step 6: Package your Chart

Once you've defined your Kubernetes resources and customized your Helm Chart, you can package it into a compressed Chart archive (.tgz) using the helm package command:

```
helm install mywebapp-chart ./mywebapp-0.1.0.tgz
```

This command will create a .tgz file in the current directory, which contains your Helm Chart.

7. Step 7: Install your Helm Chart

```
helm install mywebapp-chart ./mywebapp-0.1.0.tgz
```

```
osboxes@osboxes:~/mywebapp$ cd ..
osboxes@osboxes:~$ helm package mywebapp
Successfully packaged chart and saved it to: /home/osboxes/mywebapp-0.1.0.tgz
osboxes@osboxes:~$ helm install mywebapp-chart /home/osboxes/mywebapp-0.1.0.tgz
NAME: mywebapp-chart
LAST DEPLOYED: Sat Sep  9 11:42:04 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=mywebapp,app.kubernetes.io/instance=mywebapp-chart" -o jsonpath="{.items[0].metadata.name}")
  export CONTAINER_PORT=$(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
```

Step 8: Verify your Chart Deployment

```
kubectl get pods
```

```
osboxes@osboxes:~$ kubectl get pods
NAME                              READY   STATUS             RESTARTS          AGE
hello-node-7475554dc6-kspn6       1/1     Running            1 (20d ago)       119d
liveness-probe                    1/1     Running            1 (20d ago)       118d
liveness-probe-new                0/1     CrashLoopBackOff   254 (4m45s ago)   118d
my-app                            1/1     Running            0                 14d
my-deployment-7f4c5fc875-44qp9    0/1     ContainerCreating  0                 87s
my-deployment-7f4c5fc875-78vvc    0/1     ContainerCreating  0                 86s
readiness-probe                   1/1     Running            1 (20d ago)       118d
startup-probe                     1/1     Running            1 (20d ago)       118d
osboxes@osboxes:~$
```

Kubectl get svc

Now, you can access your NGINX web application in your browser at http://127.0.0.1:8080.