SpringbootHelm-1

Wednesday, January 5, 2022 11:36 PM

Building first Helm Chart with Spring Boot Microservices

Managing kubernets cluster means checking cluster, pods, nodes, application deployment, replicas, load-balancer and the list goes one. So the question which arises in my mind -

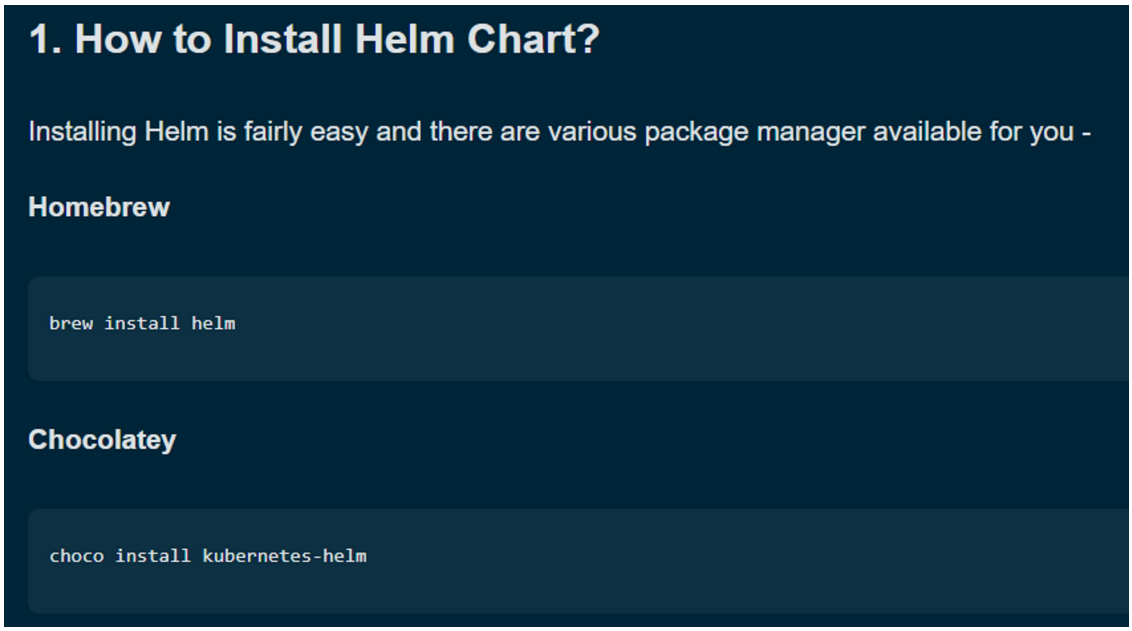
Is there an easy way to manage this or at least some part of it?
The answer is Yes and its Helm Chart

What is Helm Chart?
Helm Chart is an application package manager for the kubernetes cluster, just like we have apt packager manager in Linux.

So what can we do with Helm Chart -

1. Define k8s application
2. Install k8s application
3. Upgrade k8s application

Most important aspect of the Helm Chart is you do not have to use Kubernetes CLI(command line interface) and neither you need to remember complex kubernetes commands to manage the kubernetes manifest.



Scoop

```
scoop install helm
```

GoFish

```
gofish install helm
```

Snap

```
sudo snap install helm --classic
```

Binary

If you do not like any of the above packager manager then you could download the binaries as well

- Get the Binary - Download Binary
- Unpack it using - *tar -zxvf helm-vxxx-xxxx-xxxx.tar.gz*
- Move it - **mv linux-amd64/helm /usr/local/bin/helm**

Using Script

There is one more way to install latest Helm Version using script. Refer to the following terminal command for installing latest version of Helm -

BASH

```
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
```

https://github.com/helm/helm/releases

curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3

chmod 700 get_helm.sh

./get_helm.sh

Verify Helm Chart Installation

I am assuming you might have choose one of the installation option for Helm Chart.

To verify the installation use the following command

```bash
which helm
```

```bash
/usr/local/bin/helm
```

2. Let's create Our First Helm Chart

Before we create a our First Helm Chart, we need to have kubernetes cluster up and running. Use the following command to verify the status of kubernetes cluster -

```bash
kubectl get all
```

```bash
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.233.0.1   <none>        443/TCP   27d
```

If your kubernetes cluster is up and running then you should see default service .i.e. **service/kubernetes .**

Well enough with the per-requisites , lets run some Helm Commands for creating our first Helm Chart

```bash
helm create springboot
```

That is it and the basic Helm Chart skeleton with the name **springboot** is ready.

(**Spoiler Alert** - We are going to create our first Helm Chart for Springboot application but do not worry the same steps can be used for deploying any other application as well.)

3. Helm Chart Structure

Before we deep dive into the nitty gritty of Helm Chart, let's go through the Helm Chart Skeleton. Run the following command to see the tree structure of our Springboot Helm Chart -

```bash
tree springboot
```

```
springboot
├── Chart.yaml
├── charts
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── deployment.yaml
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── service.yaml
│   ├── serviceaccount.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml
```

(**Note - If you do not have tree command installed then use -* ***sudo apt-get install tree*** *or* ***sudo yum -y install tree***)

Inside Helm Chart ecosystem we define every configuration as YAML configuration. In the next section we will go through each YAML configuration

Chart.yaml

This file contains all the metadata about our Helm Chart for example -

```yaml
apiVersion: v2 #mandatory
name: springboot #mandatory
description: A Helm chart for Kubernetes
type: application
version: 0.1.0 #mandatory
appVersion: 1.16.0
```

1. apiVersion
2. name
3. version

Other configurations are optional

Versioning - Each chart should has its own version number and it should follow the Semantic Versioning 2.0 aka SemVer 2. But do not get confuse with apiVersion, there are no strict rules for apiVersion.

values.yaml

As the name suggests we have do something with the values and yes you are right about it. This configuration file holds values for the configuration.

Do not worry it is fairly simple to understand once you look at the following **values.yaml**

```
replicaCount: 1
image:
  repository: rahulwagh17/kubernetes:jhooq-k8s-springboot #updated url
  pullPolicy: IfNotPresent
  tag: ""
imagePullSecrets: []
nameOverride: ""
fullnameOverride: ""
serviceAccount:
  create: true
  annotations: {}
  name: ""
podAnnotations: {}
podSecurityContext: {}

securityContext: {}
service:
  type: ClusterIP
  port: 8080  #updated port
ingress:
  enabled: false
  annotations: {}
  hosts:
    - host: chart-example.local
      paths: []
  tls: []
resources: {}

autoscaling:
  enabled: false
  minReplicas: 1
  maxReplicas: 100
  targetCPUUtilizationPercentage: 80

nodeSelector: {}

tolerations: []

affinity: {}
```

It looks quite enormous but trust me you do not need to write or remember every configuration by heart. **Helm Create** command generates the bare minimum values.yaml for you.

Since in this example we are going to try our Helm Chart for spring boot application, lets go through the configuration which we need to modify for deploying our spring boot application.

1. **repository** : image:repository: rahulwagh17/kubernetes:jhooq-k8s-springboot
2. **port**: 8080

From <https://jhooq.com/building-first-helm-chart-with-spring-boot/>