Building first Helm Chart with Spring Boot Microservices

Jul 15, 2020 · 11 min read · Last Modified : Dec 8, 2020 Share on: Author : Rahul Wagh

When someone talks about managing the kubernetes cluster then I can very much assume that a particular someone is running a lot of **kubectl**, **kubeadm** command on terminal and which is quite obvious because managing kubernetes cluster is not as easy as it seems to be.

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**.

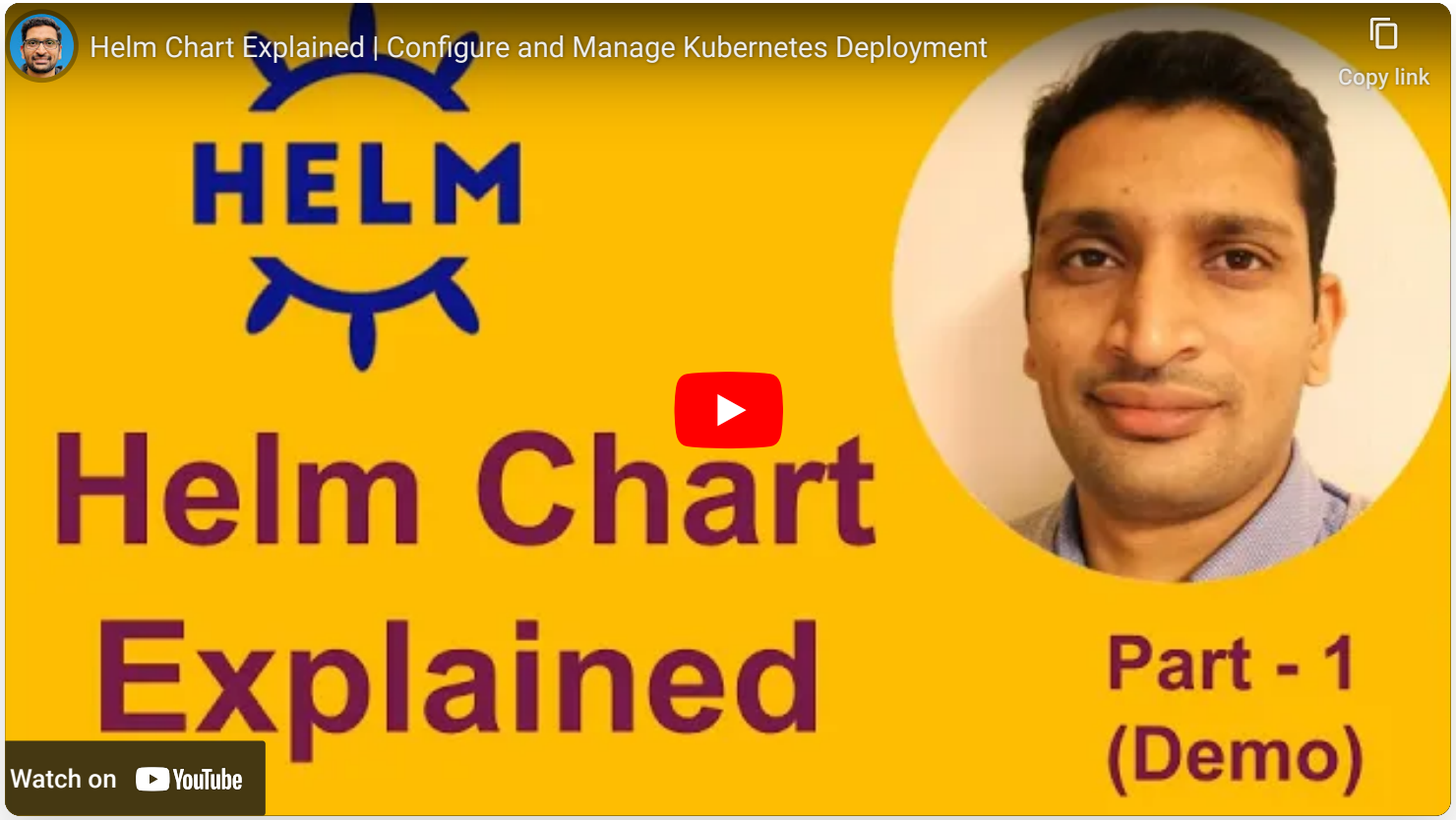
What is Helm Chart?

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 -

- Define k8s application
- Install k8s application
- 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.



Lets start playing with Helm Chart.

1. How to Install Helm Chart?

Installing Helm is fairly easy and there are various package manager available for you -

Homebrew

```bash
brew install helm
```


TERRAFORM 61 DOCKER 28
KUBERNETES 27 ANSIBLE 11
HELM-CHART 11 AWS 2
JENKINS 2 LINUX-COMMANDS 1

Tags

KUBERNETES 18
HELM-CHART 10 BLOGGING 4
QUARKUS 4 DOCKER 3
GITHUB 3 SSL 3
KUBESPRAY 2 SPRING-BOOT 2
ANSIBLE 1 HADOOP 1
INDEX 1 NGINX 1
TERRAFORM 1

Recent Posts

Chocolatey

```bash
choco install kubernetes-helm
```

Scoop

```bash
scoop install helm
```

GoFish

```bash
gofish install helm
```

Snap

```bash
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

- Can Terraform be used to provision on-premises infrastructure?
- Fixing the Terraform Error creating IAM Role. MalformedPolicyDocument Has prohibited field Resource
- In terraform how to handle null value with default value?
- Terraform use module output variables as inputs for another module?

Rahul Wagh

Its all about Open Source and DevOps, here I talk about Kubernetes, Docker, Java, Spring boot and practices.

READ MORE

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
```

```bash
chmod 700 get_helm.sh
```

```bash
./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** .

If you do not know how to prepare or push your docker image to docker hub then I would highly recommend you to follow my guide from Step 1 to step 7

And for setting up local kubernetes cluster you can refer to - 12 Steps for Installing a Production Ready Kubernetes Cluster.

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
```

```yaml
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 -

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

You do not need to define each configuration here but you must need to define -

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**

```YAML
replicaCount: 1
image:
  repository: rahulwagh17/kubernetes:jhooq-k8s-springboot #updated url
  pullPolicy: IfNotPresent
  tag: ""
```

```yaml
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

So in the whole **values.yaml** you need to update the configuration at two places **repository** and **port**

Note :You should have a docker image of your spring boot application uploaded into the docker hub

Alright now we are ready with our Chart.yaml and values.yaml, lets jump to next configuration yamls

deployment.yaml

The next configuration we need to update is deployment.yaml and as the name suggest it is used for deployment purpose. So lets see how does it looks like

YAML

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "springboot.fullname" . }}
  labels:
    {{- include "springboot.labels" . | nindent 4 }}
spec:
{{- if not .Values.autoscaling.enabled }}
  replicas: {{ .Values.replicaCount }}
{{- end }}
  selector:
    matchLabels:
      {{- include "springboot.selectorLabels" . | nindent 6 }}
  template:
    metadata:
    {{- with .Values.podAnnotations }}
      annotations:
        {{- toYaml . | nindent 8 }}
    {{- end }}
```

```
    labels:
      {{- include "springboot.selectorLabels" . | nindent 8 }}
  spec:
    {{- with .Values.imagePullSecrets }}
    imagePullSecrets:
      {{- toYaml . | nindent 8 }}
    {{- end }}
    serviceAccountName: {{ include "springboot.serviceAccountName" . }}
    securityContext:
      {{- toYaml .Values.podSecurityContext | nindent 8 }}
    containers:
      - name: {{ .Chart.Name }}
        securityContext:
          {{- toYaml .Values.securityContext | nindent 12 }}
        image: "{{ .Values.image.repository }}"   #update here
        imagePullPolicy: {{ .Values.image.pullPolicy }}
        ports:
          - name: http
            containerPort: 8080     #update here
            protocol: TCP           #comment it
            #livenessProbe:           #comment it
            #httpGet:           #comment it
            #path: /           #comment it
            #port: http           #comment it
            #readinessProbe:           #comment it
            #httpGet:           #comment it
            #path: /           #comment it
            #port: http           #comment it
        resources:
          {{- toYaml .Values.resources | nindent 12  }}
```

Do not get scared we do not need to update each and every configuration. But we need to update the **containerPort** to 8080

1. **containerPort : 8080**

Because we need to deploy our spring boot application at port 8080.

service.yaml

The service.yaml is basically used for exposing our kubernetes springboot deployment as service. The good thing over here we do not need to update any configuration here.

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

4. Run $ helm template springboot

Now we are into the step no 4 and we are pretty much ready with our first Helm Chart for our spring boot application.

But wait - *Wouldn't it be nice if you could see the service.yaml, deployment.yaml with its actual values before running the helm install command*?

Yes you can do that with the helm template command -

```bash
helm template springboot
```

(*You can not run the above command from inside the springboot directory, so you should get out from the springboot directory and then execute the command*)

After running the above command it should return you will service.yaml, deployment.yaml and test-connection.yaml with actual values

```yaml
---
# Source: springboot/templates/serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: RELEASE-NAME-springboot
  labels:
    helm.sh/chart: springboot-0.1.0
    app.kubernetes.io/name: springboot
    app.kubernetes.io/instance: RELEASE-NAME
    app.kubernetes.io/version: "1.16.0"
    app.kubernetes.io/managed-by: Helm
---
# Source: springboot/templates/service.yaml
apiVersion: v1
kind: Service
metadata:
  name: RELEASE-NAME-springboot
  labels:
    helm.sh/chart: springboot-0.1.0
    app.kubernetes.io/name: springboot
    app.kubernetes.io/instance: RELEASE-NAME
    app.kubernetes.io/version: "1.16.0"
    app.kubernetes.io/managed-by: Helm
spec:
  type: ClusterIP
  ports:
    - port: 8080
      targetPort: http
```

```
      protocol: TCP
      name: http
  selector:
    app.kubernetes.io/name: springboot
    app.kubernetes.io/instance: RELEASE-NAME
---
# Source: springboot/templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: RELEASE-NAME-springboot
  labels:
    helm.sh/chart: springboot-0.1.0
    app.kubernetes.io/name: springboot
    app.kubernetes.io/instance: RELEASE-NAME
    app.kubernetes.io/version: "1.16.0"
    app.kubernetes.io/managed-by: Helm
spec:
  replicas: 1
  selector:
    matchLabels:
...
```

I love this command because it locally renders the templates by replacing all the placeholders with its actual values.

There is one more sanitary command **lint** provided by helm which you could run to identify possible issues forehand.

BASH

```
helm lint springboot
```

BASH

```
==> Linting springboot
[INFO] Chart.yaml: icon is recommended

1 chart(s) linted, 0 chart(s) failed
```

As you can see the command output **1 chart(s) linted, 0 chart(s) failed**. So there is no failure and you are good to go.

<

/>

5. helm -debug -dry-run

The next check which we are going to do is **-dry-run**. Helm is full of such useful utility which allows developer to test its configuration before running the final install command

Use the following **-dry-run** command to verify your Spring Boot Helm Chart

```bash
helm install springboot --debug --dry-run springboot
```

If there is something wrong with your Helm chart configuration then it will going to prompt you immediately.

<

/>

6. helm install

Alright lets run the install command.

```bash
helm install myfirstspringboot springboot
```

helm install myfirstspringboot springboot

Lets break down the command because if you are doing it for the first time then it might be little confusing for you.

There are two name which we used during the installation command -

1. **myfirstspringboot** : It's a release name for helm chart otherwise helm will generate its own release name that is why we have assigned this name.
2. **springboot** : It is our actual chart name which we created in Step 2.

7. Verify the helm install

Next question which comes into my mind - "**How to see all the releases?** "

Use the following list command to list down all the releases -

```bash
helm list -a
```

helm list -a

It should return you the release .i.e. - **myfirstspringboot** which we did in the **Step 6**.

Lets do some cross verification using kubectl commands also, so that we can make sure helm has done its work.

```bash
kubectl get all
```

helm install kubectl get all

Now we have installed our first helm chart of springboot application. Lets test the spring boot application by accessing it

```bash
curl 10.233.28.240:8080/hello
```

```bash
Hello - Jhooq-k8s
```

As you can it return with a response **Hello - Jhooq-k8s**.

Kudos if you made this far then I would say you learned *"How to create and install your first Helm Chart"*

8. Upgrade helm release

There is one more feature of Helm Chart which is **helm upgrade**. It makes it easy for devops to release the new version of application.

Lets take the same **myfirstspringboot** release and update its **replicaCount** from **1** to **2**

But before we need to update the **replicaCount**. First we need to udpate the version in **Chart.yaml** from **0.1.0** to **0.1.1**

```yaml
apiVersion: v2
name: springboot
description: A Helm chart for Kubernetes
type: application
version: 0.1.1
appVersion: 1.16.0
```

Alright now lets update the **replicaCount** in **values.yaml**

```yaml
replicaCount: 2
image:
  repository: rahulwagh17/kubernetes:jhooq-k8s-springboot
  pullPolicy: IfNotPresent
  tag: ""
imagePullSecrets: []
nameOverride: ""
fullnameOverride: ""
serviceAccount:
  create: true
  annotations: {}
  name: ""
podAnnotations: {}
podSecurityContext: {}

securityContext: {}
service:
  type: ClusterIP
  port: 8080
ingress:
  enabled: false
```

```
  annotations: {}
  hosts:
    - host: chart-example.local
      paths: []
  tls: []
resources: {}

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

nodeSelector: {}

tolerations: []

affinity: {}
```

Okay now we are done with the update and ready to make our new release upgrade. Use the following command -

BASH

```
helm upgrade myfirstspringboot .
```

helm upgrade myfirstspringboot .

You should see a message with your release name .i.e. - ***Release "myfirstspringboot" has been upgraded. Happy Helming!***

Verify your helm upgrade by running following list command

$ helm list -a

As you can see now the revision count is 2.

Lets again run kubectl get deployment

```bash
kubectl get deployments
```

kubectl get deployments

And now you can see we have successfully upgraded the replica count from 1 to 2.

<

/>

9. Rollback Helm release

In the **Step 8** we **upgraded** the Helm chart release from **version 1** to **version 2**.

So let's see one more rollback feature of **Helm Chart**.

```bash
helm rollback myfirstspringboot 1
```

helm rollback myfirstspringboot 1

As you can see from the above screenshot we successfully rolled back the release to the previous version. But one interesting thing about Helm is, it still updates the **REVISION to 3**

```bash
helm list -a
```

helm list -a after the rollback

To confirm that we have actually **rolled back** our **Helm Chart release**, lets run some kubectl commands

```bash
kubectl get deployments
```

kubectl get deployments after helm rollback

Now as you can see from the screenshot we now have only one replica running for "**myfirstspringboot**" deployments.

<

/>

10. Delete Helm release

Wouldn't it be nice if you need to run only one command to delete your Helm release and you do not have to do anything else.

```bash
helm delete myfirstspringboot
```

As you can see now you have successfully delete your release with single **helm delete** command.

helm delete myfirstspringboot

Summary

As I can summarize what we have learned -

1. We started from scratch and as a first step we installed Helm Chart on the development box
2. Then we created our first Helm Chart using **$ helm create springboot**
3. After the helm chart creation, we went through the structure of Helm Chart

4. We explored some of the utility commands provided by the helm .i.e. **$ helm template springboot**
5. Then we have seen how to do -dry-run for the Helm Release
6. After verifying our Helm Chart then we executed helm install **$ myfirstspringboot springboot**
7. We used **$helm list -a** to verify the helm release
8. Then we have used **$ helm upgrade** command to upgrade the release
9. Finally, we **rolled back** and **delete** the release

You can follow all the above steps for almost any application but provided you have the docker image uploaded to either docker hub or some other container registry.

> **Read More** -
>
> 1. Helm chart - How to Add/Install plugins
> 2. Getting started with Helm Chart
> 3. Helm chart - WordPress Installation with MariaDB on Kubernetes
> 4. Helm chart - Build you first helm chart with Spring Boot
> 5. Helm Chart - Convert Kubernetes YAML into Helm Chart YAML
> 6. Helm Chart - Pass environment variables
> 7. Helm Chart - Plugin
> 8. Helm Chart - Dry Run Install
> 9. Helm Chart - How to create multiple values files inside helm chart?
> 10. Helmfile - How to use Helmfile for managing helm chart?

Posts in this series

- How to use Helmfile for managing helm chart?
- How to create multiple values files inside helm chart?

- Pass environment variables into Helm Chart?
- How to fix - Helm install unknown flag --name/Error must either provide a name or specify --generate-name?
- Understanding Helm dry run for template debugging
- How to fix - Error create failed to create Secret invalid metadata.name Invalid value DNS-1123 subdomain must consist of lower case alphanumeric characters - or ., and must start and end with an alphanumeric character (e.g. example.com, regex used for validation is)
- Convert Kubernetes deployment YAML into Helm Chart YAML
- Helm chart - Wordpress Installation with MariaDB on Kubernetes
- Helm chart - How to Add/Install plugins
- Getting Started with Helm Chart
- Building first Helm Chart with Spring Boot Microservices