

How To Create A Helm Chart

February 3, 2021

HELM

KUBERNETES

[Home](#) » [DevOps and Development](#) » How To Create A Helm Chart

Introduction

Helm charts are one of the [best practices for building efficient clusters in Kubernetes](#). It is a form of packaging that uses a collection of Kubernetes resources. Helm charts use those resources to define an application.


Helm charts use a template approach to deploy applications. Templates give structure to projects and are suitable for any type of application.

This article provides step-by-step instructions to create and deploy a Helm chart.



Prerequisites

- Access to a CLI
- Minikube cluster installed and configured. (For assistance, follow our guides [How to Install Minikube on Ubuntu](#) and [How to Install Minikube on CentOS](#).)
- [Helm installed](#) and configured.



Note: To confirm Helm installed properly, run `which helm` in the terminal. The output should return a path to Helm.

Create Helm Chart

Creating a Helm chart involves creating the chart itself, configuring the image pull policy, and specifying additional details in the *values.yaml* file.

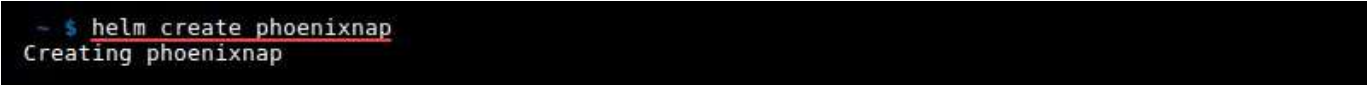
Step 1: Create a New Helm Chart

1. To create a new Helm chart, use:

```
helm create <chart name>
```

For example:

```
helm create phoenixnap
```



```
~ $ helm create phoenixnap
Creating phoenixnap
```

2. Using the [ls command](#), list the chart structure:

```
ls <chart name>
```



```
~ $ ls phoenixnap
charts  Chart.yaml  templates  values.yaml
```

The Helm chart directory contains:

- **Directory *charts*** – Used for adding dependent charts. Empty by default.
- **Directory *templates*** – Configuration files that deploy in the cluster.
- **YAML file** – Outline of the Helm chart structure.
- **YAML file** – Formatting information for configuring the chart.

Step 2: Configure Helm Chart Image Pull Policy

1. Open the *values.yaml* file in a [text editor](#). Locate the *image* values:

```
1 # Default values for test chart.
2 # This is a YAML-formatted file.
3 # Declare variables to be passed into your templates.
4
5 replicaCount: 1
6
7 image:
8   repository: nginx
9   pullPolicy: IfNotPresent
10   # Overrides the image tag whose default is the chart appVersion.
11   tag: ""
```

There are three possible values for the *pullPolicy*.

- **IfNotPresent** – Downloads a new version of the image if one does not exist in the cluster.
- **Always** – Pulls the image on every restart or deployment.
- **Latest** – Pulls the most up-to-date version available.

2. Change the image *pullPolicy* from **IfNotPresent** to **Always**:

```
1 # Default values for test chart.
2 # This is a YAML-formatted file.
3 # Declare variables to be passed into your templates.
4
5 replicaCount: 1
6
7 image:
8   repository: nginx
9   pullPolicy: Always
10   # Overrides the image tag whose default is the chart appVersion.
11   tag: ""
```

Step 3: Helm Chart Name Override

To override the chart name in the *values.yaml* file, add values to the *nameOverride* and *fullnameOverride*:

```
7 image:
8   repository: nginx
9   pullPolicy: Always
10   # Overrides the image tag whose default is the chart appVersion.
11   tag: ""
12
13 imagePullSecrets: []
14 nameOverride: ""
15 fullnameOverride: ""
16
```

For example:

```
7 image:
8   repository: nginx
9   pullPolicy: Always
10   # Overrides the image tag whose default is the chart appVersion.
11   tag: ""
12
13 imagePullSecrets: []
```

```
14 fullnameOverride: phoenix-app
15 fullnameOverride: "phoenix-chart"
16
```

Overriding the Helm chart name ensures configuration files also change.

Step 4: Specify Service Account Name

The service account name for the Helm chart generates when you run the cluster. However, it is good practice to set it manually.

The service account name makes sure the application is directly associated with a controlled user in the chart.

1. Locate the *serviceAccount* value in the *values.yaml* file:

```
16
17 serviceAccount:
18   # Specifies whether a service account should be created
19   create: true
20   # Annotations to add to the service account
21   annotations: {}
22   # The name of the service account to use.
23   # If not set and create is true, a name is generated using the
  fullname template
24   name: ""
```

2. Specify the *name* of the service account:

```
16
17 serviceAccount:
18   # Specifies whether a service account should be created
19   create: true
20   # Annotations to add to the service account
21   annotations: {}
22   # The name of the service account to use.
23   # If not set and create is true, a name is generated using the
  fullname template
24   name: "phoenixnap"
```

Step 5: Change Networking Service Type

The recommended networking service type for Minikube is **NodePort**.

1. To change the networking service type, locate the *service* value:

```
39 service:
40   type: ClusterIP
41   port: 80
```

2. Change the *type* from **ClusterIP** to **NodePort**:

```
39 service:
40   type: NodePort
41   port: 80
```

Deploy Helm Chart

After configuring the *values.yaml* file, check the status of your Minikube cluster and deploy the application using [Helm commands](#).

Step 1: Check minikube Status

If Minikube isn't running, the install Helm chart step returns an error.

1. Check Minikube status with:

```
minikube status
```

The status shows up as *Running*.

```
~ $ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
timeToStop: Nonexistent
```

2. If the status shows *Stopped*, run:

```
minikube start
```

```
~ $ minikube status
minikube
type: Control Plane
host: Stopped
kubelet: Stopped
apiserver: Stopped
kubeconfig: Stopped
timeToStop: Nonexistent

~ $ minikube start
🐳 minikube v1.17.1 on Linuxmint 18.3
🔗 Using the docker driver based on existing profile
🔗 Starting control plane node minikube in cluster minikube
🔗 Restarting existing docker container for "minikube" ...
🔗 Preparing Kubernetes v1.20.2 on Docker 20.10.2 ...
🔗 Verifying Kubernetes components...
🔗 Enabled addons: storage-provisioner, default-storageclass
🔗 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

The output shows *Done* and the status changes to *Running*.

Step 2: Install the Helm Chart

Install the Helm chart using the [helm install command](#):

```
helm install <full name override> <chart name>/ --values <chart name>/values.yaml
```

For example:


```
helm install phoenix-chart phoenixnap/ --values phoenixnap/values.yaml
```

```
~ $ helm install phoenix-chart phoenixnap/ --values phoenixnap/values.yaml
NAME: phoenix-chart
LAST DEPLOYED: Mon Feb  1 15:24:22 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" service phoenix-chart)
  export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")
  echo http://$NODE_IP:$NODE_PORT
```

The `helm install` command deploys the app. The next steps are printed in the *NOTES* section of the output.

Step 3: Export the Pod Node Port and IP Address

1. Copy the two `export` commands from the `helm install` output.
2. Run the commands to get the Pod node port and IP address:

```
~ $ export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services phoenix-chart)
~ $ export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")
```

Step 4: View the Deployed Application

1. Copy and paste the `echo` command and run it in the terminal to print the IP address and port:

```
~ $ echo http://$NODE_IP:$NODE_PORT
http://192.168.49.2:30230
```

2. Copy the link and paste it into your browser, or press **CTRL+click** to view the deployed application:

