


Unit 7

# Helm Charts



Business  
Training

1

## Outline

- What is Helm ?
- The Helm 3 CLI
- Creating Helm Charts
- Helm Release Lifecycle
- Helm Repositories

M.Romdhani, 2022

2

2

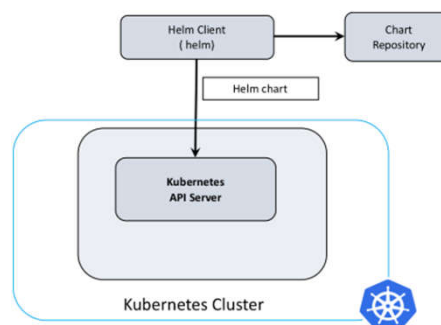
## What is Helm ?

3

## What is Helm?

Helm Charts

- **Deploying applications to Kubernetes can be complex.**
  - Setting up a single application can involve creating multiple interdependent Kubernetes resources – such as pods, services, deployments, and replicasets – each requiring you to write a detailed YAML manifest file.
- **Helm is a package manager for Kubernetes that allows developers and operators to more easily package, configure, and deploy applications and services onto Kubernetes clusters.**
- **Helm is now an official Kubernetes project and is part of the Cloud Native Computing Foundation (CNCF), a non-profit that supports open source projects in and around the Kubernetes ecosystem.**
  - Helm 3, an important architectural change. Helm 3 has a client-only architecture with the client still called helm



M.Romdhani, 2022

4

4

Helm Charts

## Installing Helm

---

- **Helm can be installed either from source or from pre-built binaries. It is also available for popular package managers such as Homebrew or Chocolatey.**
  - For example, you can install Helm on Windows using Chocolatey:
 

```
> choco install helm
```
  
- **Once finished, you can verify the installation by invoking**

```
> helm version --short
```

M.Romdhani, 2022
5

5

Helm Charts

## Helm 3 basic concepts

---

- **There are three basic concepts to understand.**
  - A **Chart** is a Helm package. It contains all of the resource definitions.
  - A **Repository** is the place where charts can be collected and shared. It's like Perl's CPAN archive or the Fedora Package Database, but for Kubernetes packages.
  - A **Release** is an instance of a chart running in a Kubernetes cluster. One chart can often be installed many times into the same cluster. And each time it is installed, a new release is created. Consider a MySQL chart. If you want two databases running in your cluster, you can install that chart twice. Each one will have its own release, which will in turn have its own release name.
  
- **With these concepts in mind, we can now explain Helm like this:**
  - Helm installs **charts** into Kubernetes, creating a new **release** for each installation. And to find new charts, you can search Helm chart **repositories**.

M.Romdhani, 2022
6

6

Helm Charts

## Chart Metadata

- Every chart comes with a set of metadata attached to it. Metadata is used to expose essential information about the chart to its consumers. Metadata is located in **Chart.yaml** and can only be specified by the author of the chart.
  - Good examples of chart-metadata-properties are:
    - **name**: The name of the chart
    - **version**: The version of the chart
    - **description**: A short description of the chart
    - **kubeVersion**: A SemVer range of compatible Kubernetes versions

M.Romdhani, 20227

7

Helm Charts

## Helm Templates

- Helm compiles templates into Kubernetes definitions (YAML files). They are written using the Go template language. As chart author, you can use several things to build powerful templates:
  - pre-defined variables
  - custom variables
  - template functions
- More than 60 template functions are available in Helm 3. Some of them are rooted from the Go Template Language, but most powerful template functions are part of Sprig.
- As a chart author, you provide default values for variables using the **values.yaml** file.

M.Romdhani, 20228

8

## Values in Helm

- **Before deploying charts to a cluster, consumers can specify custom values to override variable default-values.**
  - Single values can be applied using the `--set` argument when executing helm install.
  - Alternatively, consumers can put all their values in a `.yaml` file and call helm install `--values=myvalues.yaml`.
- **Helm specifies a bunch of pre-defined values like Release.Name (the unique name of the release) or Release.Namespace (name of the Kubernetes namespace the chart was deployed to).**

## Helm 3 CLI

## Helm 3 CLI

- **Helm 3 CLI offers a bunch of sub-commands that you will be using regularly. See the following list which contains eight sub-commands you will use quite often:**
  - **helm search** - Search for charts in Helm Hub and Helm Repositories
  - **helm install** - Install charts into Kubernetes
  - **helm list** - List all releases of a given Kubernetes Namespace
  - **helm show** - Show information about a chart
  - **helm upgrade** - Upgrade a release to a new version of the underlying chart
  - **helm pull** - Download and extract a chart from Helm hub or a Helm repository
  - **helm repo** - Add, update, index, list, or remove chart repositories
  - **helm package** - Create an tgz archive for the chart in the current folder
- **There are more sub-commands available in Helm 3 CLI.**

M.Romdhani, 2022

11

11

## Helm install: Installing a package

- **To install a new package, use the helm install command. At its simplest, it takes two arguments: A release name that you pick, and the name of the chart you want to install.**

```
helm install happy-panda stable/mariadb
```

```

Fetches stable/mariadb-0.3.0 to
/Users/mattbutcher/Code/Go/src/helm.sh/helm/mariadb-0.3.0.tgz
happy-panda
Last Deployed: Wed Apr 28 12:32:28 2019
Namespace: default
Status: DEPLOYED

```

  - Now the mariadb chart is installed. Note that installing a chart creates a new release object. The release above is named happy-panda. (If you want Helm to generate a name for you, leave off the release name and use --generate-name.)
- **Customizing the Chart Before Installing**
  - Installing the way we have here will only use the default configuration options for this chart. Many times, you will want to customize the chart to use your preferred configuration.
  - To see what options are configurable on a chart, use helm show values:

```
helm show values stable/mariadb
```
  - You can then override any of these settings in a YAML formatted file, and then pass that file during installation.

M.Romdhani, 2022

12

12

## Creating Helm Charts

13

### Create a Helm 3 Chart manually

Helm Charts

- Although Helm 3 provides a command to create new charts (`helm create <chart-name>`), you can create charts also manually.
  - Creating a chart manually is the best way to identify what it takes to author charts and an excellent way to get started.
- Create a Helm chart manually
  - Every Helm chart is isolated in the scope of a directory and consists of at least three files. The Chart.yaml file, a values.yaml file and a template file, which belongs to the templates sub-folder. The following script generates all files and folders underneath the first-chart folder.
 

```
mkdir -p first-chart/templates
cd first-chart
touch Chart.yaml values.yaml templates/serviceaccount.yaml
```
  - First, let's provide a set of minimal metadata in Chart.yaml.
 

```
apiVersion: v2
name: firstchart
description: A simple Helm chart for k8s
type: application
version: 0.0.1
```

M.Romdhani, 2022

14

14

## Create a Helm 3 Chart manually

- Having all metadata in place, we can move on and create a simple Kubernetes object. For demonstration purposes, we will create a ServiceAccount.

- Add the following to templates/serviceaccount.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: {{ .Values.serviceAccount.name }}
```

- To make the template robust, we should also deal with empty values and replace them with a fallback value of development. Last but not least, we have to ensure that our template generates valid YAML. Considering those new requirements, the templates/serviceaccount.yaml will look like this:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: {{ .Values.serviceAccount.name }}
  labels:
    environment: {{ .Values.environment | default "development" | quote }}
}}
```

M.Romdhani, 2022

15

15

## Create a Helm 3 Chart manually

- The last part of the chart is the [values.yaml](#). Chart authors can use this file to provide default values for chart consumers. You can also think about [values.yaml](#) as the API documentation of your chart.

```
# environment defaults internally to development
# however, consumers can override it using a custom values file.
environment:
serviceAccount:
  name: sample-sa
```

M.Romdhani, 2022

16

16



## Helm Release Lifecycle

17

### Packaging Helm charts

Helm Charts

- When you are done with authoring your chart, it is time to prepare for distribution. Helm charts are packaged using the helm package sub-command:

```
# ensure you are in the folder of your chart
cd firstchart

helm package .
```
- You will now find a `firstchart-0.0.1.tgz` in the project folder. If you prefer a different output folder, you can specify it using the `--destination` argument. Also, notice the `.` as the first argument of `helm package`, it specifies the context for the packaging operation (and should point to your charts root directory).

M.Romdhani, 2022

18

18

## Creating a Release (install a chart)

- To install our firstchart in version 0.0.1 into a cluster, make sure that your local kubectl is pointing to the correct one.
  - `helm install firstchart-0.0.1.tgz --namespace app-one --generate-name`
- To install a chart, you have to specify a name for the release.
  - Alternatively, you can use the `--generate-name` argument to get a random name assigned by Helm itself. The command above specified the targetting namespace explicitly. However, if you target the default namespace, you don't have to specify the `--namespace` argument.

## Verifying a Release

- There are several ways to verify a release.
  - First, you can use the `helm list` and `helm get` commands. Those are great to get an overview or detailed information about a given release in a cluster.
  - The alternative way is to use regular `kubectl` commands to inspect the contents of the desired namespace.

## Upgrading a Release

- Before we look at CLI commands to upgrade an existing release, let's create a new version of our chart. For demonstration purpose, we will modify the template of our ServiceAccount and add another custom tag.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: {{ .Values.serviceAccount.name }}
  labels:
    environment: {{ .Values.environment | default "development" | quote }}
app: foo
```

- We have added app: foo, nothing more. Finally let's increase the version in Chart.yaml from 0.0.1 to 0.0.2 and package the chart again by invoking helm package. Now you will find a firstchart-0.0.2.tgz in the root folder of your chart.
- To upgrade an existing release, two things are required. First, we need the name of the release you want to update. Second, we need a new chart archive, which contains the updated chart. We use both with helm upgrade
  - helm upgrade my-release firstchart-0.0.2.tgz

M.Romdhani, 2022

21

21

## Retracting a release (uninstall a chart)

- To remove or uninstall a release from a Kubernetes cluster, we use the `helm uninstall` command in combination with the unique release name.
  - Helm will quickly confirm the removal of all release artifacts.

M.Romdhani, 2022

22

22

## Helm repositories

23

### Helm Repositories

Helm Charts

- In Helm, a repository (or repo) is a simple HTTP server that is responsible for serving charts. Users can use the Helm CLI to consume charts from repositories, or to publish their own charts to repositories for distribution.
  - Helm CLI can interact with multiple repositories. We manage repositories using the helm repo command.
- Adding the official repository to Helm 3
  - By default, Helm CLI has no repositories configured. Execute helm repo list and verify that no repository is configured.
  - To add the official, stable repository, execute:  
`helm repo add stable https://kubernetes-charts.storage.googleapis.com/`

M.Romdhani, 2022

24

24