**Helm Charts**

Helm uses a packaging format called charts. A chart is a collection of files that describe a related set of Kubernetes resources. A single chart might be used to deploy something simple, like a memcached pod, or something complex, like a full web app stack with HTTP servers, databases, caches, and so on.

Charts are created as files laid out in a particular directory tree. They can be packaged into versioned archives to be deployed.

This document explains the chart format, and provides basic guidance for building charts with Helm.

**The Chart File Structure**

A chart is organized as a collection of files inside of a directory. The directory name is the name of the chart (without versioning information). Thus, a chart for **CDP-Service** would be stored in a **cdp-service/** directory.

Inside of this directory, Helm will expect a structure that matches this:

*cdp-service/*

*Chart.yaml # A YAML file containing information about the chart*

*values.yaml # The default configuration values for this chart*

*templates/ # A directory of templates that, when combined with values,*

*# will generate valid Kubernetes manifest files.*

**The Chart.yaml File**

The Chart.yaml file is required for a chart. It contains the following fields:

apiVersion: v1 *# The chart API version (required)*

description: A Helm chart for cdp-service *# A single-sentence description of this project (optional)*

name: cdp-service *# The name of the chart (required)*

version: 1.0.0 *# A SemVer 2 version (required)*

**Charts and Versioning**

Every chart must have a version number. For example, **cdp-service** chart whose version field is set to version: 1.2.3 will be named: *cdp-service-1.2.3.tgz*

**Templates and Values**

All template files are stored in a chart's templates/ folder. When Helm renders the charts, it will pass every file in that directory through the template engine. There is a file called *values.yaml* in the root folder. This file can contain default values.

When a user supplies custom values, these values will override the values in the chart's values.yaml file.

To override custom values, it must be specified with *helm install* and *helm upgrade* commands like

**helm install -f values.yaml cdp-service ./ --set image.dev.repository=*image\_name* --set image.dev.tag=*tag\_name***

**Template Files**

Template Files contain the Kubernetes resource files as yaml. Definition files of all Kubernetes objects to be created must be in **cdp-service/templates/** folder. We use 4 Kubernetes objects for **cdp-service** chart: Deployment, Service, Configmap, Secret.

Configmap and Secret objects are required to keep environment variables and using them inside of the container. Secret object keeps the sensitive environment variables as base64 encoded. Configmap object keeps the other environment variables as clear text. In addition to these objects, the most important Kubernetes object is the deployment. We use deployment object to create Kubernetes Pods. And the last definition yaml of object is service. We use service.yaml file to create Kubernetes Service object.

All of these templates contain the parametric values. And the default values are gathering from values.yaml file. For example, the container name and container image name of pod defines at deployment yaml at lines between 22 and 29.

- name: {{ .Values.application }}

{{ if eq .Values.environment "dev" }}

image: {{ .Values.image.dev.repository }}:{{ .Values.image.dev.tag }}

{{ else if eq .Values.environment "preprod" }}

image: {{ .Values.image.preprod.repository }}:{{ .Values.image.preprod.tag }}

{{ else if eq .Values.environment "prod" }}

image: {{ .Values.image.prod.repository }}:{{ .Values.image.prod.tag }}

{{ end }}

Name of the container parameter is read from values.yaml as seen above. And image name/tag of the container parameter contain conditional check. We use this condition to separate according to environments. Because, there may be differences between environments. As it is here, we use conditional check for Configmap and Secret objects, too. For example, if we check configmap.yaml file, we can see the condition lines. All parameters inside the configmap.yaml are gathering according to the *environment* variable inside the values.yaml. If it is *dev*, configmap uses *dev* variables from the values.yaml.

apiVersion: v1

kind: ConfigMap

metadata:

name: cm-{{ .Values.application }}-{{ .Values.environment }}

namespace: {{ .Release.Namespace }}

data:

{{ if eq .Values.environment "dev" }}

fletum-api-base-url: "{{ .Values.dev.fletum\_api\_base\_url }}"

{{ else if eq .Values.environment "preprod" }}

fletum-api-base-url: "{{ .Values.preprod.fletum\_api\_base\_url }}"

{{ else if eq .Values.environment "prod" }}

fletum-api-base-url: "{{ .Values.prod.fletum\_api\_base\_url }}"

{{ end }}

**Values files**

Considering the template in the previous section, a values.yaml file that supplies the necessary values would look like this:

environment: dev

application: cdp-service

containerPort: 5500

service:

name: cdp-service-svc

type: ClusterIP

port: 5500

Note: If the --set flag is used on helm install or helm upgrade, those values are simply converted to YAML on the client side.

Note: Once you have edited a chart, helm can package it into a chart archive for you:

helm package ./cdp-service

Archived cdp-service-1.0.0.tgz

Deploying helm charts to Kubernetes, you need to run this command:

helm install -f ./values.yaml cdp-service ./ --set environment=dev

Upgrading helm charts, you need to run this command

helm upgrade -f ./values.yaml cdp-service ./ --set environment=dev