NOKIA

Helm introduction

Comparing pure Kubernetes API usage with Helm on top of it.

Gergo Huszty

13-12-2017

Helm introduction

Why Kubernetes is not enough?

- Kubernetes APIserver is an object store
- So it can only consume complete manifest files
  - This means it can not substitute any user-defined or runtime parameters
  - Manifest files should usually contain environment specific data
- Kubernetes handles only individual API objects
  - No shared properties
  - No package concept

Helm introduction
Features

- Implements software package concept on top of the container orchestrator
- Manages SW package lifecycle (deploy, delete, upgrade, fallback)
- Parameter handling for the application (renders Kubernetes manifest files before use)
- Dependency handling in case when application is formed from multiple SW packages
- Provides framework for user-defined hooks (e.g. pre-upgrade, post-install)
- Does not limit the Kubernetes features in any way (as Kubernetes does with docker API)
- Terms
  - Chart: the packaging format (it contains: manifest templates, default variable values, dependencies towards other charts)
  - Release: a deployed instance of a Chart
  - Revision: a particular state of a Release

Helm introduction
What Helm isn't?

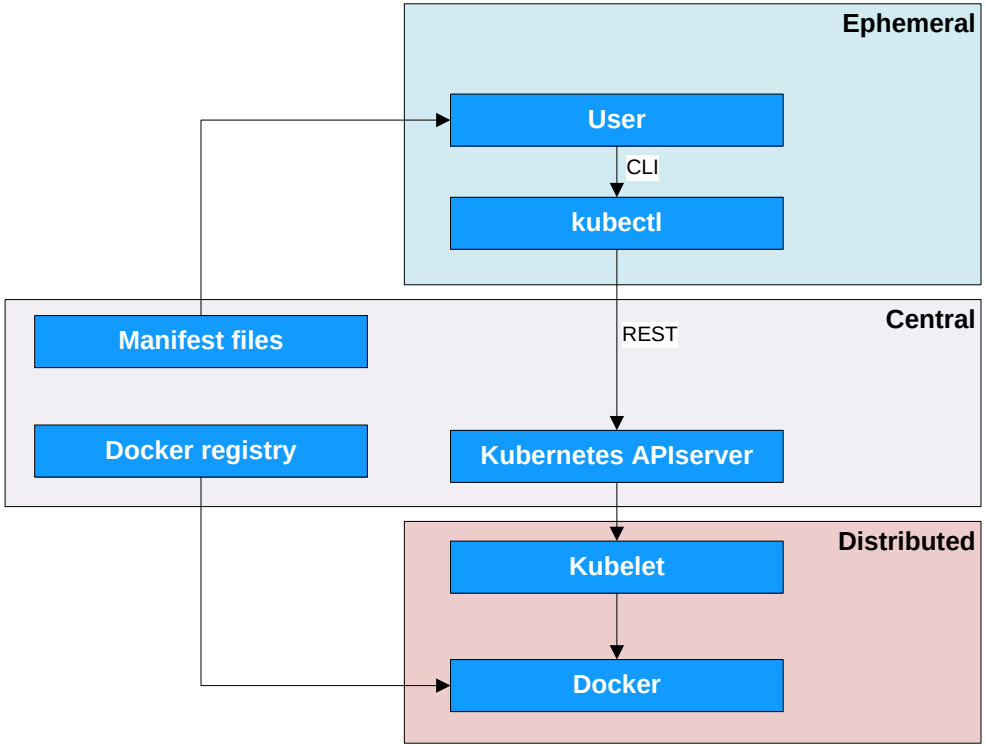
- It is not able to do application specific things
  - It is not like Kubernetes operators (concept from CoreOS)
    - But Helm can greatly cooperate with operators
  - However, application chart developer can create custom actions for install/upgrade (hooks)

Helm introduction
Helm architecture

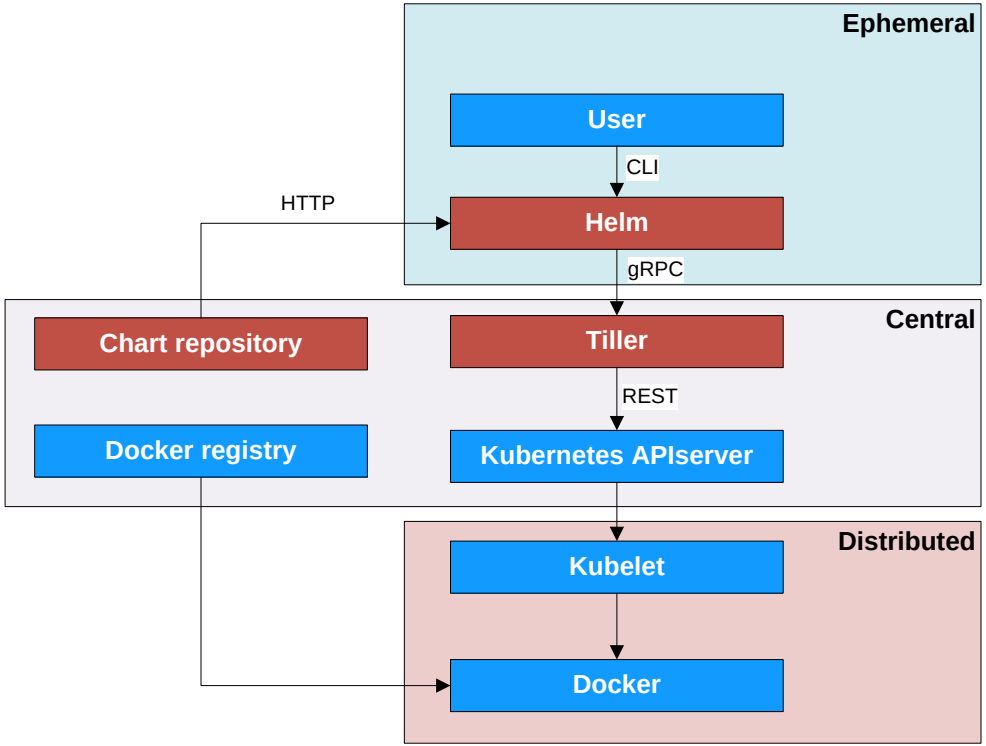
Decoupled to server-client

- Server: Tiller
  - Lives in the Kubernetes cluster (single-pod deployment)
  - Has gRPC NBI towards clients
  - Roles
    - Talking directly to Kubernetes apiserver
    - Manifest rendering
    - Release history maintenance
- Client: Helm
  - CLI (core codebase is prepared for alternative client implementations)
  - Roles
    - Chart handling
      - Downloading
      - Toolbox for chart development
      - Dependency handling, etc.
    - Instructing Tiller

Helm introduction
Kubernetes without Helm



Helm introduction
Kubernetes with Helm



Helm introduction
Life Cycle Management hooks

- Chart developer can define hook implementations as k8s manifests, just like the application
- Hooks has special annotations in the manifest files, which is identified by Helm
- Hooks are usually implemented in Kubernetes Jobs
- Hooks can do whatever they want (talk to the application, talk to Kubernetes, etc.)
- Typical use-cases for hooks
  - Backup data before upgrade, restore before or after fallback
  - Bootstrapping in-memory application content (e.g. DB schema)

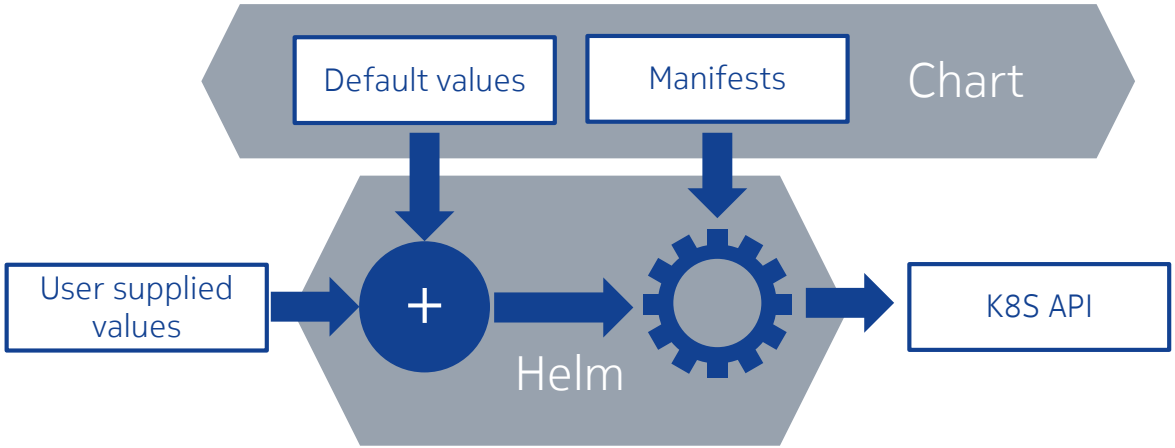
Helm introduction
Upgrade

- From Helm's perspective, everything between install and delete is an "upgrade" or "fallback"
  - Upgrade of a release can include new Chart version AND/OR new input Values (Chart+Values content is a "revision" for Helm)
  - Upgrade hooks will be always executed!
- The change effect depends on k8s API behavior
  - Some property can be changed only with object replace (e.g. Pod re-instantiate)
  - Some of them is seamless (e.g. replica count increase for Deployment)
- Use cases – everything which is expected to be a permanent change on the application
  - Minor upgrade (change some image versions)
  - Major upgrade (change structure/architecture of the application)
  - Manual scale
- Non use cases
  - One-time or app specific operations (backup or inquire something)
    - Such operation shall be supported by the "operator" of the application

```
helm upgrade <RELEASE_NAME> <NEW_CHART_URL> [-f <VALUE_FILE>] [--set NAME=VALUE]
```

Helm introduction
Chart rendering

- Chart vendor shall make everything a parameter which is allowed to change by the user
- Chart rendering can invoke not only parameter substitution but based on the values it can make any operation what go template language allows, like:
  - Disable complete (or fragments of) manifest files
  - Loops
- The value change effect depends on k8s behavior



Helm introduction
Chart template example

- Chart vendor shall make everything a parameter which is allowed to change by the user
- Chart rendering can invoke not only parameter substitution but based on the values it can make any operation what go template language allows, like:
  - Disable complete (or fragments of) manifest files
  - Loops
- The value change effect depends on k8s behavior

```
{{- if and .Values.persistence.enabled (not .Values.persistence.existingClaim) }}
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: {{ template "fullname" . }}
  labels:
    app: {{ template "fullname" . }}
    chart: "{{ .Chart.Name }}-{{ .Chart.Version }}"
    release: "{{ .Release.Name }}"
    heritage: "{{ .Release.Service }}"
  annotations:
  {{- if .Values.persistence.storageClass }}
    volume.beta.kubernetes.io/storage-class: {{ .Values.persistence.storageClass | quote }}
  {{- else }}
    volume.alpha.kubernetes.io/storage-class: default
  {{- end }}
spec:
  accessModes:
    - {{ .Values.persistence.accessMode | quote }}
  resources:
    requests:
      storage: {{ .Values.persistence.size | quote }}
{{- end }}
```

Helm introduction
References

- http://helm.sh/
- https://github.com/kubernetes/helm/
- https://github.com/kubernetes/charts
- https://www.gcppodcast.com/post/episode-50-helm-with-michelle-noorali-and-matthew-butcher/

Helm introduction
Demo

- Kubernetes cluster prepared, with:
  - Dashboard
  - Flannel
  - Helm
  - Etcd operator deployed
- Helm chart is deployed, content:
  - Etcd cluster 3rd party resource definition
  - Etcd hello-world application Pod
  - Etcd data bootstrap as post-install hook
- Expectation:
  - Helm value defines the Etcd cluster name
  - Cluster is created
  - Application get the service name for Etcd
  - Reports Etcd cluster health in log

https://github.com/libesz/etcd_healthchecker

NOKIA