Helm – A gentle introduction

David Carew
carew@us.ibm.com

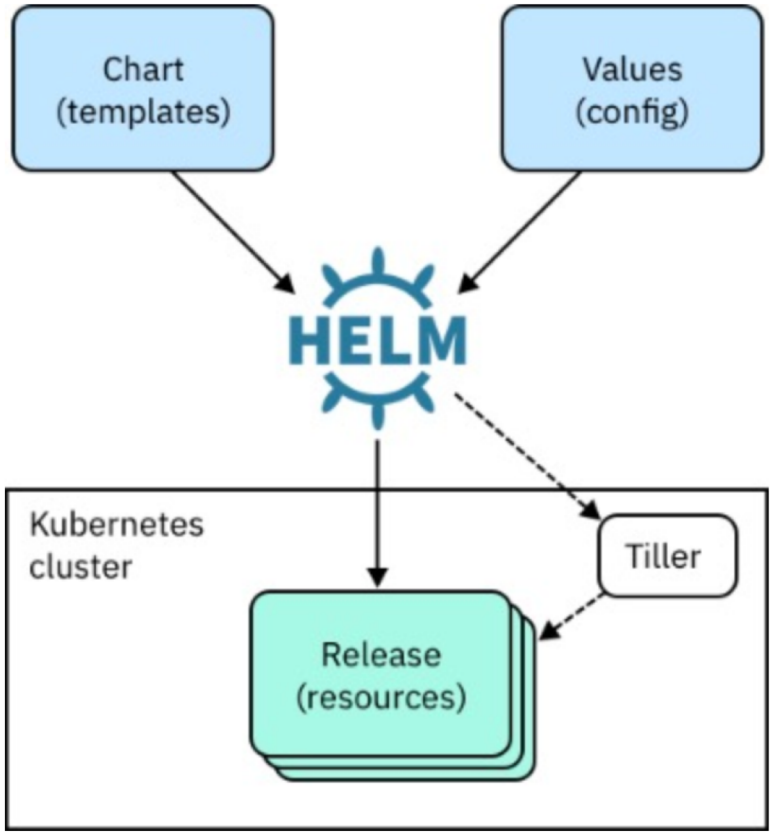
IBM **Developer**

What is Helm ?

A package manager for Kubernetes

Helm components

- Helm has two elements, a client (Helm) and a server (Tiller). The server element runs inside a Kubernetes cluster and manages the installation of charts.



Helm terminology

- **Helm:** A command-line interface (CLI) that installs charts into Kubernetes, creating a release for each installation. To find new charts, you search Helm chart repositories.

- **Chart:** An application package that contains *templates* for a set of resources that are necessary to run the application. A template uses variables that are substituted with values when the manifest is created. The chart includes a values file that describes how to configure the resources.

- **Repository:** Storage for Helm charts. The namespace of the hub for official charts is *stable*.

- **Release:** An instance of a chart that is running in a Kubernetes cluster. You can install the same chart multiple times to create many releases.

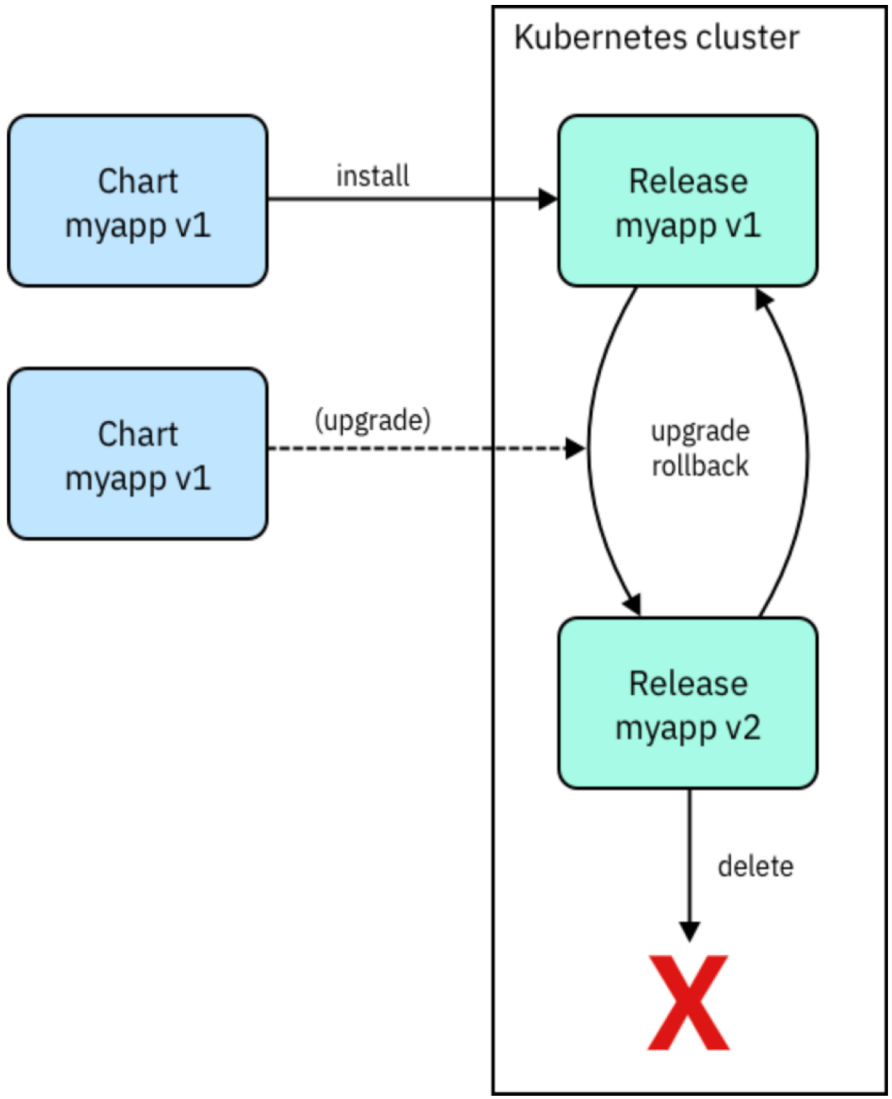
- **Tiller:** The Helm server-side templating engine, which runs in a pod in a Kubernetes cluster. Tiller processes a chart to generate Kubernetes resource manifests, which are YAML-formatted files that describe a resource and then installs the release into the cluster. Tiller stores each release as a Kubernetes ConfigMap.

Why use Helm?

Helm make deployments easier and repeatable because all resources for an application are deployed by running one command:

```
$ helm install <chart>
```

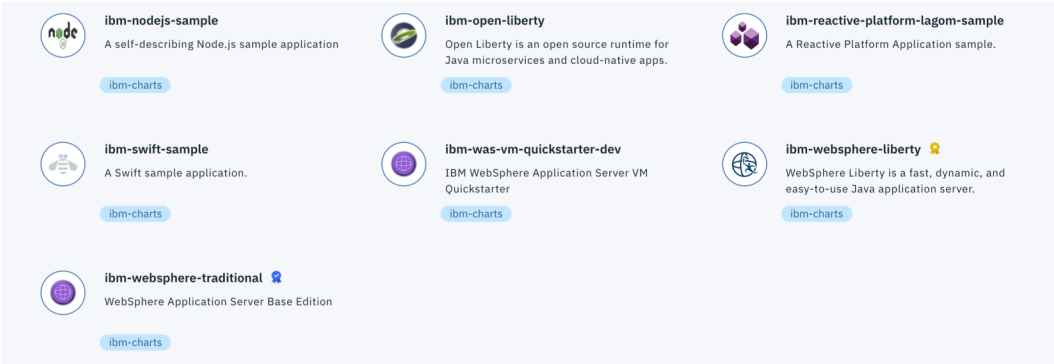
You can use single commands for installing, upgrading, and deleting releases.



What is a chart repository?

- An HTTP server that houses packaged charts and an index.yaml file.
- That file has an index of all the charts in the repository.
- A chart repository can be any HTTP server that can serve YAML and .tar files and can answer GET requests.

In IBM Cloud Private, catalog entries are Helm charts that can be deployed from the chart repositories.



Deploying an application

The Helm install command deploys an application. The command output includes details about the release and resources. For the chart in this example, stable/mysql, the generated release name is loping-toad. One resource of each type exists, all named loping-toad-mysql:

- Secret
- Service
- Deployment
- PersistentVolumeClaim

```
$ helm search mysql
```

```
NAME            VERSION                 DESCRIPTION
stable/mysql    0.1.1                   Chart for MySQL
```

```
$ helm install stable/mysql
```

```
Fetched stable/mysql to mysql-0.1.1.tgz
NAME: loping-toad
LAST DEPLOYED: Thu Oct 20 14:54:24 2016
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/Secret
NAME TYPE               DATA            AGE
loping-toad-mysql       Opaque    2         3s

==> v1/Service
NAME                    CLUSTER-IP              EXTERNAL-IP           PORT(S)    AGE
loping-toad-mysql    192.168.1.5              <none>                3306/TCP   3s

==> extensions/Deployment
NAME                    DESIRED      CURRENT    UP-TO-DATE     AVAILABLE     AGE
loping-toad-mysql    1              0               0          0                3s

==> v1/PersistentVolumeClaim
NAME           STATUS     VOLUME   CAPACITY     ACCESSMODES AGE
loping-toad-mysql Pending
```

Default and custom deployment values

The default values for a deployment are stored in the **values.yaml** file in the chart. You can customize aspects of the deployment by overriding those values.

- To specify a release's name, use the `--name` flag:

```
$ helm install --name CustomerDB stable/mysql
```

- To deploy the release into a Kubernetes namespace, use the `--namespace` flag:

```
$ helm install --namespace ordering-system stable/mysql
```

- To override a value, use the `--set` flag:

```
$ helm install --set user.name='student',user.password='passw0rd' stable/mysql
```

- To override values with a values file, use the `--values` or the `--f` flag:

```
$ helm install --values myvalues.yaml stable/mysql
```

Packaging charts

A chart is a directory. A Helm client can use chart directories on the same computer, but it's difficult to share with other users on other computers.

You package a chart by bundling the **chart.yaml** and related files into a .tar file and then installing the chart into a chart file:

```
$ helm package <chart-path>

$ helm install <chart-name>.tgz
```

To add a chart to a repository, copy it to the directory and regenerate the index:

```
$ helm repo index <charts-path>  # Generates index of the charts in the repo
```

index files and .tgz files can then be served up via an HTTP/S server so they can be installed remotely

Templates and settings files

Creating a chart consists of implementing a template and populating a settings file, which is the configuration file that the template uses. Settings files, specifically the **values.yaml** file, define the chart's API. The settings files list the variables that the templates can use.

A template can create the manifest for any type of Kubernetes resource.

Example of a service template

```
apiVersion: v1
kind: Service
metadata:
  name: {{ .Release.Name }}-{{ .Chart.Name }}
  labels:
     app: {{ .Release.Name }}-{{ .Chart.Name }}
     chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
     release: {{ .Release.Name }}
     heritage: {{ .Release.Service }}
     tier: database

spec:
  type: {{ .Values.service.type }}
  ports:
     - port: {{ .Values.service.internalPort }}
  selector:
    app: {{ .Release.Name }}-{{ .Chart.Name }}
    tier: database
```

template + *set var=val command line args* + *values.yaml* = kubernetes *config .yaml*

Resources

Helm best practices
https://docs.helm.sh/chart_best_practices/

Helm documentation
https://docs.helm.sh/