Introduction to Helm

What is Helm?

Helm is a package manager for Kubernetes.

Helm helps you define, install and upgrade Kubernetes applications. Helm uses Helm Charts, which allow you to inject variables, functions and Go scripting to create environment specific configurations for your Kubernetes object specifications.

Helm Charts use Go templates for templating your resource files, adds all functions in the Sprig library and a few special template functions like `include` and `required`.

History of Helm

2015 Helm Classic introduced as a Deis project at Kubecon

Jan 2016, Helm code merge with Google's Kubernetes Deployment Manager (DM) as Kubernetes subproject

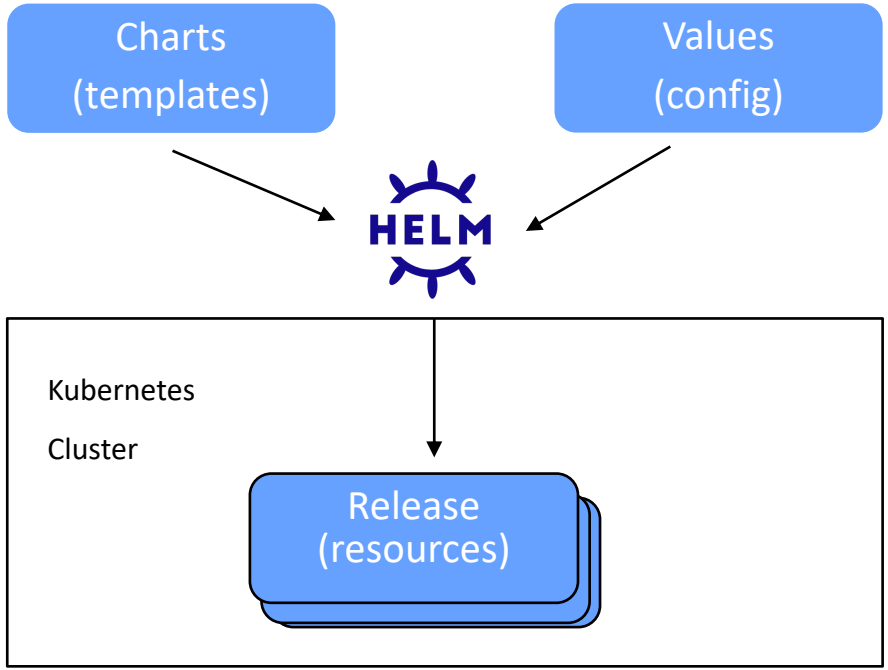
2016, Helm 2.0, server-side component of DM was renamed to Tiller

June 2018, Helm promoted to CNCF Project and includes Monocular, Helm Chart Repo, Chart Museum and later Helm Hub

Nov 2019, Helm 3.0 released, Tiller removed

Helm components

Helm allows developers to package simple to complex Kubernetes applications as a single unit, called a Helm Chart, that can be installed, upgraded and uninstalled for revision management.



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*.

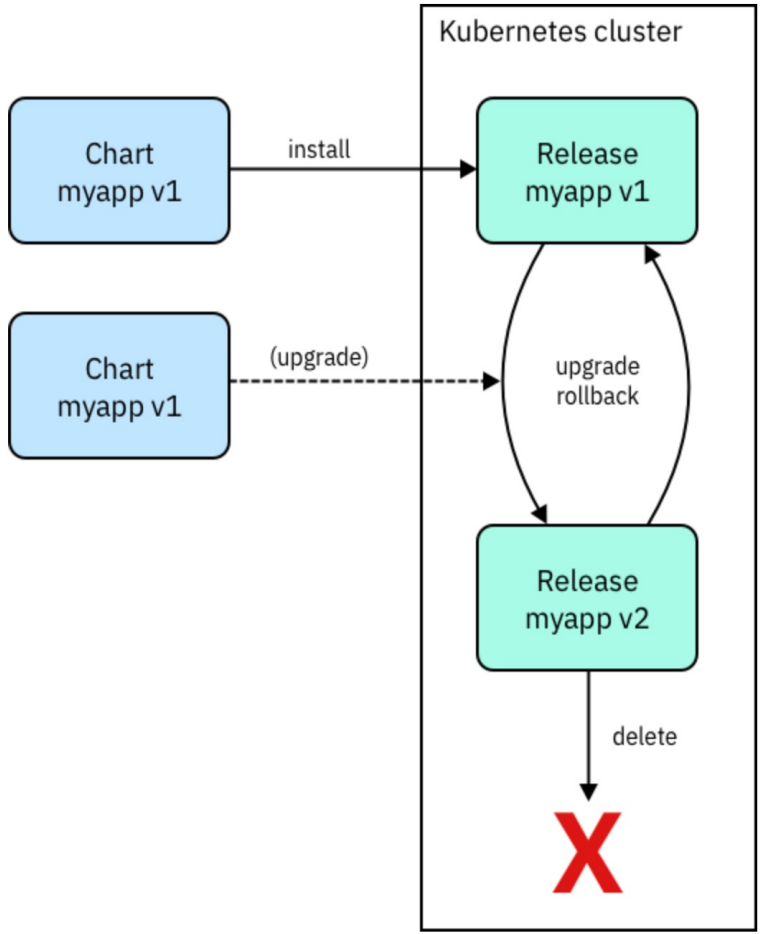
- **Release:** An instance of a chart that is running in a Kubernetes cluster. You can install the same chart multiple times to create many releases.

Why use Helm?

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

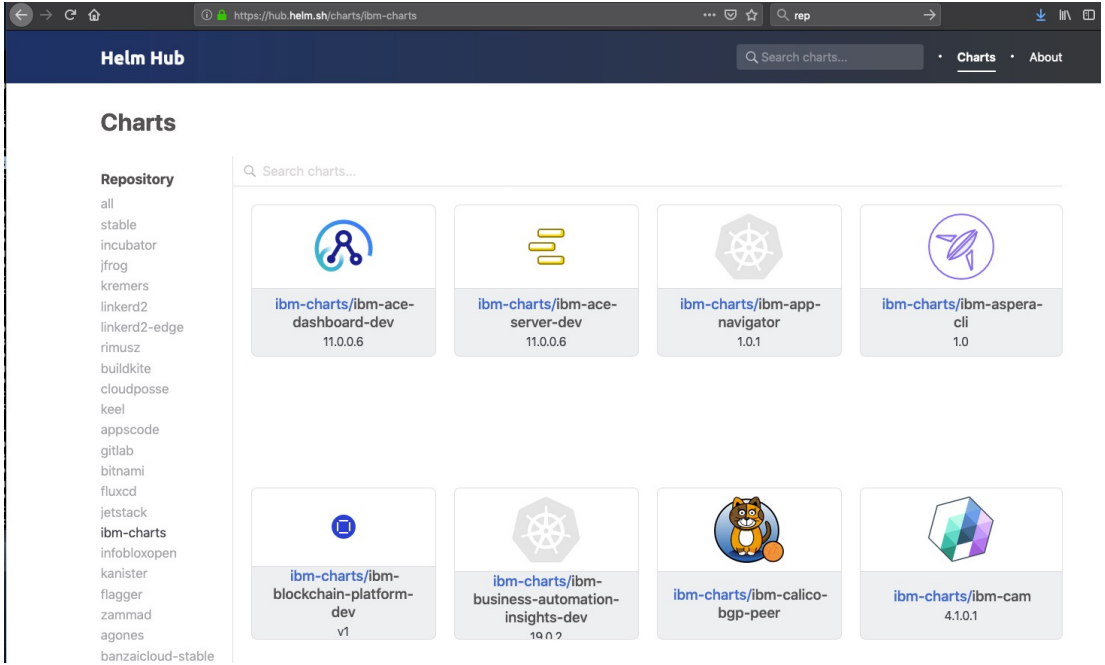
```
$ helm install myapp <chart>
```

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



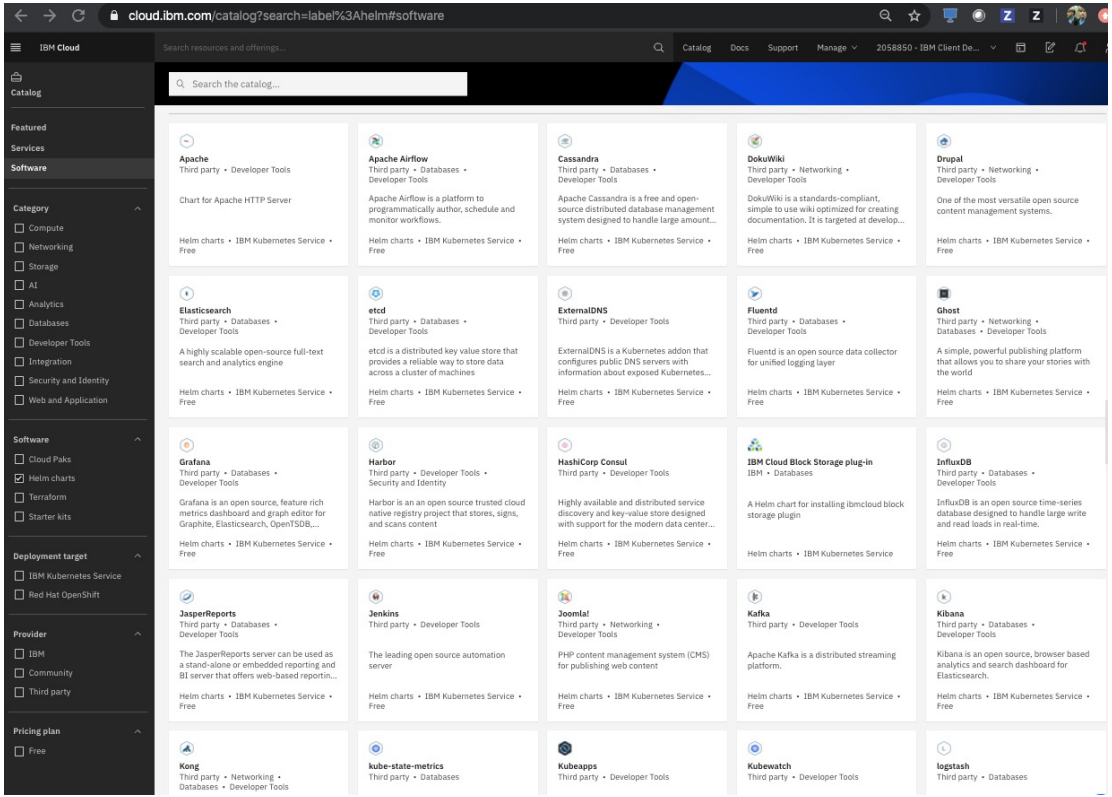
What is a chart repository?

- 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.
- **Helm Hub** https://hub.helm.sh/ is a searchable repository that can be used to search across several public Helm repositories
- IBM Cloud Catalog supports Helm



Helm Hub at https://hub.helm.sh/

IBM Cloud Catalog - Charts



IBM Cloud Catalog at https://cloud.ibm.com/catalog?search=label%3Ahelm#software

Deploying an Application

```
$ 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
```

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

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
```

Default and Custom Deployment Values

The default values for a chart are stored in the **values.yaml** file. You can customize the deployment of the chart by overriding some or all of these default values

- You can override default values using the **--set** flag

```
$ helm install mydb2 ibm-charts/ibm-db2oltp-dev --set persistence.enabled=false
```

- You can also override values using your own yaml file with non default values

```
$ helm install mydb2 ibm-charts/ibm-db2oltp-dev –f myvalues.yaml
```

- If you do both **--set** takes precedence

Revision Management

Kubernetes does not support revision management. Kubernetes logs every spec change, but you manage the rollout and rollback yourself. With Helm, every time an install, upgrade, or rollback happens, the revision number is incremented by 1

```
$ helm history guestbook-demo -n helm-demo
REVISION UPDATED STATUS CHART APP VERSION DESCRIPTION
1 Mon Feb 24 18:08:02 2020 superseded guestbook-0.2.0 Install complete
2 Tue Feb 25 14:23:27 2020 deployed guestbook-0.2.0 Upgrade complete
```

```
$ helm rollback guestbook-demo 1 -n helm-demo
Rollback was a success! Happy Helming!
```

```
$ helm history guestbook-demo -n helm-demo
REVISION UPDATED STATUS CHART APP VERSION DESCRIPTION
1 Mon Feb 24 18:08:02 2020 superseded guestbook-0.2.0 Install complete
2 Tue Feb 25 14:23:27 2020 superseded guestbook-0.2.0 Upgrade complete
3 Tue Feb 25 14:53:45 2020 deployed guestbook-0.2.0 Rollback to 1
```

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

Go Templates

```
**values.yaml**

nodeSelector:
   key1: value1
   key2: value2
```

```
**deployment.yaml

   {{- with .Values.nodeSelector }}
   nodeSelector:
     {{- toYaml . | nindent 8 }}
   {{- end }}
```

Gets rendered as,

```
   nodeSelector:
      key1: value1
      key2: value2
```

Sprig Functions

The Sprig library provides over 70 template functions for Go's template language.

- String Functions: `trim`, `wrap`, `randAlpha`, `plural`, etc.
  - String List Functions: `splitList`, `sortAlpha`, etc.
- Math Functions: `add`, `max`, `mul`, etc.
  - Integer Slice Functions: `until`, `untilStep`
- Date Functions: `now`, `date`, etc.
- Defaults Functions: `default`, `empty`, `coalesce`, `toJson`, `toPrettyJson`, `toRawJson`, `ternary`
- Encoding Functions: `b64enc`, `b64dec`, etc.
- Lists and List Functions: `list`, `first`, `uniq`, etc.
- Dictionaries and Dict Functions: `get`, `set`, `dict`, `hasKey`, `pluck`, `deepCopy`, etc.
- Type Conversion Functions: `atoi`, `int64`, `toString`, etc.
- File Path Functions: `base`, `dir`, `ext`, `clean`, `isAbs`
- Flow Control Functions: `fail`
- Advanced Functions
  - UUID Functions: `uuidv4`
  - OS Functions: `env`, `expandenv`
  - Version Comparison Functions: `semver`, `semverCompare`
  - Reflection: `typeOf`, `kindIs`, `typeIsLike`, etc.
  - Cryptographic and Security Functions: `derivePassword`, `sha256sum`, `genPrivateKey`, etc.

Encoding functions:
http://masterminds.github.io/sprig/encoding.html
b64enc/b64dec

```
{{- define "imagePullSecret" }}
{{- printf "{\"auths\": {\"%s\": {\"auth\": \"%s\"}}}"
.Values.imageCredentials.registry (printf "%s:%s"
.Values.imageCredentials.username .Values.imageCredentials.password |
b64enc) | b64enc }}
{{- end }}
```

Charts

Helm charts are structured like this:

```
Mychart/
  charts/
  templates/
    NOTES.txt
    _helpers.tpl
  Chart.yaml
  values.yaml
```

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.

To create a new chart,

```
$ helm create mychart
```

Chart.yaml

The Chart.yaml file contains metadata with a description of the chart. To see all valid fields, go to https://helm.sh/docs/topics/charts/#the-chartyaml-file.

Example of a Chart.yaml

```
apiVersion: v2
name: mychart
description: A Helm chart for Kubernetes
type: application
version: 0.1.0
appVersion: 1.16.0
```

values.yaml

The **values.yaml** file contains the default configuration values for the chart.

Example of a values.yaml file

```
replicaCount: 1

image:
  repository: nginx
  pullPolicy: IfNotPresent
  tag: “”

author:
    firstname: John
    lastname: Doe
```

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

Built-in Objects

Objects are passed into a template from the template engine, your code can pass objects around, or you can create new objects in your template. Built-in values always begin with a capital letter, in keeping with Go's naming convention. It is recommended to use only initial lower case letters in order to distinguish local names from those built-in.

You can access several top-level objects in your templates:

- Release
- Values
- Chart
- Files
- Capabilities
- Template

For more details about built-in objects, see
https://helm.sh/docs/chart_template_guide/builtin_objects/

Kubernetes Service Object

A Helm Chart also includes templated versions of Kubernetes object specifications called templates in the /templates directory.

Example of a Service template using built-in objects

```
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
```

_helpers.tpl

The _helpers.tpl is the default location for template partials. Files whose name begins with an underscore (_) are assumed to *not* have a Kubernetes object spec inside. These files are used to store partials and helpers.

```
{{/* Generate basic labels */}}
{{- define "mychart.labels" }}
  labels:
    generator: helm
    date: {{ now | htmlDate }}
{{- end }}
```

By convention, define functions should have a simple documentation block ({{/* … */}}) describing what they do. The defined Chart template can now be re-used in other templates.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
  {{- template "mychart.labels" }}
data:
  myvalue: "Hello World"
  {{- range $key, $val := .Values.favorite }}
  {{ $key }}: {{ $val | quote }}
{{- end }}
```

NOTES.txt

The NOTES.txt file is a way to provide instructions to your chart users.

```
Thank you for installing {{ .Chart.Name }}.

Your release is named {{ .Release.Name }}.

To learn more about the release, try:

$ helm status {{ .Release.Name }}
$ helm get all {{ .Release.Name }}
```

If you run the `helm install` command the output will append a `NOTES:` section at the end, which renders the NOTES.txt file.

Resources

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

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

Lab Time