IBM Training

Introduction to Helm

Agenda

- What is Helm?
- Using Helm
- Developing charts
- Developing templates
- Conclusion
- Resources

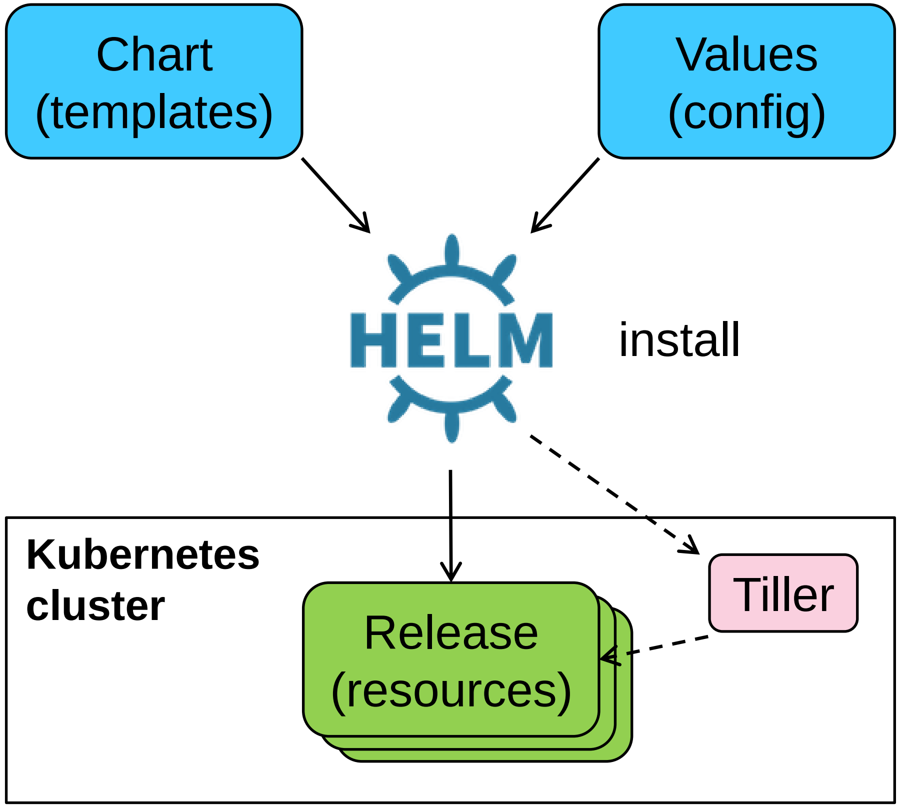
What is Helm?

Helm – A package manager for Kubernetes

- What is a package manager?
  - Automates the process of installing, configuring, upgrading, and removing computer programs
  - Examples: Red Hat Package Manager (RPM), Homebrew, Windows Pkgmgr/PackageManagement
- Helm enables multiple Kubernetes resources to be created with a single command
  - Deploying an application often involves creating and configuring multiple resources
  - A Helm chart defines multiple resources as a set
- An application in Kubernetes typically consists of (at least) two resource types
  - Deployment – Describes a set of pods to be deployed together
  - Services – Endpoints for accessing the APIs in those pods
  - Could also include ConfigMaps, Secrets, Ingress, etc.
- A default chart for an application consists of a deployment template and a service template
  - The chart creates all of these resources in a Kubernetes cluster as a set
  - Rather than manually having to create each one separately via `kubectl`

Helm Terminology

- Helm
  - Helm installs charts into Kubernetes, creating a new release for each installation
  - To find new charts, search Helm chart repositories
- Chart
  - Templates for a set of resources necessary to run an application
  - The chart includes a values file that configures the resources
- Repository
  - Storage for Helm charts
  - `stable` – The namespace of the hub for official charts
- Release
  - An instance of a chart running in a Kubernetes cluster
  - The same chart installed multiple times creates many releases
- Tiller
  - Helm templating engine, runs in a pod in a Kubernetes cluster
  - Tiller processes the chart to generate the resource manifests, then installs the release into the cluster
  - Tiller stores each release as a Kubernetes config map



Advantages of Using Helm

- Deploy all of the resources for an application with a single command
  - Makes deployment easy and repeatable

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

- Separates configuration settings from manifest formats
  - Edit the values without changing the rest of the manifest
  - `values.yaml` – Update to deploy the application differently
- Upgrade a running release to a new chart version

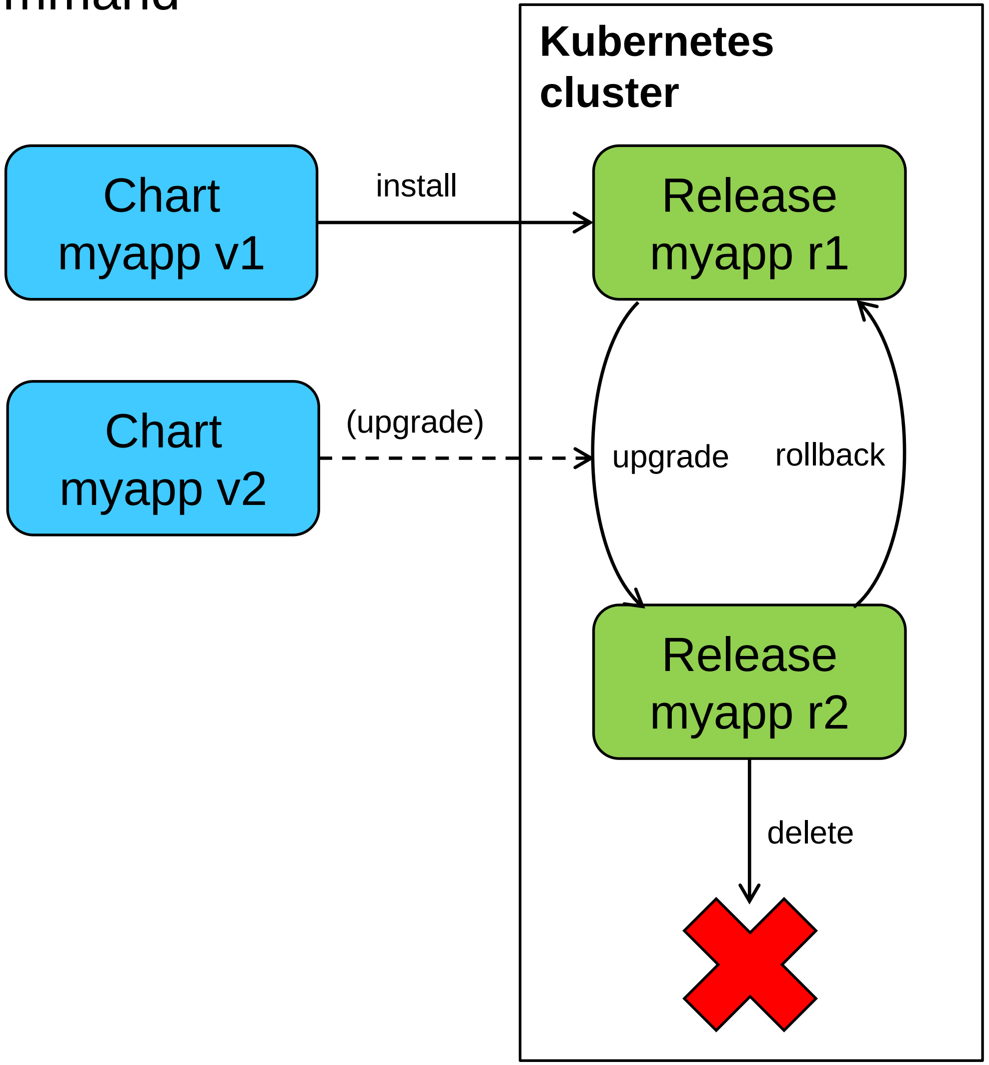
```
$ helm upgrade <release> <chart>
```

- Rollback a running release to a previous revision

```
$ helm rollback <release> <revision>
```

- Delete a running release

```
$ helm delete <release>
```



Installing Helm

- Helm runs as a CLI client, so is installed on your laptop

- See Installing Helm
  - https://docs.helm.sh/using_helm/#installing-helm

- Options for installing Helm
  1. Download the release, including the binary
     - https://github.com/kubernetes/helm/releases
  2. Homebrew on MacOS
     - `brew install kubernetes-helm`
  3. Installer script
     - `curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get > get_helm.sh`

Using Helm

Helm Commands

- Install Tiller

```
$ helm init
```

- Create a chart

```
$ helm create <chart>
```

- List the repositories

```
$ helm repo list
```

- Search for a chart

```
$ helm search <keyword>
```

- Info about a chart

```
$ helm inspect <chart>
```

- Deploy a chart (creates a release)

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

- List all releases

```
$ helm list --all
```

- Get the status of a release

```
$ helm status <release>
```

- Get the details about a release

```
$ helm get <release>
```

- Upgrade a release

```
$ helm upgrade <release> <chart>
```

- Rollback a release

```
$ helm rollback <release> <revision>
```

- Delete a release

```
$ helm delete <release>
```

Working with Repositories

```
$ helm repo list
NAME        URL
stable      https://kubernetes-charts.storage.googleapis.com/
```

```
$ helm search jenkins
NAME              VERSION     DESCRIPTION
stable/jenkins    0.1.14      A Jenkins Helm chart for Kubernetes.
```

```
$ helm repo add my-charts https://my-charts.storage.googleapis.com
$ helm repo list
NAME        URL
stable      https://kubernetes-charts.storage.googleapis.com/
my-charts   https://my-charts.storage.googleapis.com
```

Installing an Application

- To deploy an application into Kubernetes, install that application’s Helm chart

```
$ 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
```

- Install output
  - Details about the release
  - Details about its resources
- Chart
  - `stable/mysql`
- Release name
  - `loping-toad` (auto generated)
- Resources
  - Four total, one of each type
  - All named `loping-toad-mysql`
  - `Secret`
  - `Service`
  - `Deployment`
  - `PersistentVolumeClaim`

Overriding Values

- Default values are stored in the chart

```
<chart-path>/values.yaml
```

- Helm CLI uses Kubernetes CLI’s config to connect to your current cluster

```
~/.kube/config
$ kubectl config view
```

- To specify a release’s name, use the *name* flag

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

- To deploy the release into a particular Kubernetes namespace, use the *namespace* flag

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

- To override an individual 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 *f* flag

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

Developing Charts

Creating a Chart

- Creating a new chart generates a directory with sample files

```
$ helm create my-chart
$ tree my-chart
```

```
my-chart/
    |- Chart.yaml                       # Information about the chart
    |- values.yaml                      # The default configuration values for this chart
    |- charts/                          # Charts that this chart depends on
    |- templates/                       # The template files
        |- NOTES.txt                                      # OPTIONAL: A plain text file
containing short usage notes
        |- _helpers.tpl                 # OPTIONAL: The default location for template partials
        |- deployment.yaml
        |- service.yaml
```

- By default, a chart starts with sample templates for a Kubernetes deployment and service
  - In the simplest case, just edit the `values.yaml` file

How Install Uses Charts

- The main step of installing a chart is rendering its templates

- How Helm installs a chart
  1. User runs an install in the Helm CLI
     ```
     $ helm install myapp
     ```
  2. Helm CLI loads the chart into Tiller
  3. **Tiller renders the `myapp` templates**
  4. Tiller loads the resulting resources into Kubernetes
  5. Tiller returns the release data to the client
  6. The client exits

- Rendering the templates
  - Each template generates a Kubernetes resource manifest file (yaml)
  - Tiller runs each of the template files, generating the resource files
- Tiller then loads the resources—as described by the manifests—into the Kubernetes cluster

Chart Lifecycle Hooks

Hooks

- **pre-install**
  - Executes after templates are rendered
  - Before any resources are created in Kubernetes
- **post-install**
  - Executes after all resources are loaded into Kubernetes
- **pre-delete**
  - Executes before any resources are deleted from Kubernetes
- **post-delete**
  - Executes after all of the release's resources have been deleted
- **pre-upgrade**
  - Executes after templates are rendered
  - Before any resources are loaded into Kubernetes
- **post-upgrade**
  - Executes after all resources have been upgraded
- **pre-rollback**
  - Executes after templates are rendered
  - Before any resources have been rolled back
- **post-rollback**
  - Executes after all resources have been modified

Hooks in the Helm Install Lifecycle

1. User runs an install in the Helm CLI
2. Helm CLI loads the chart into Tiller
3. Tiller renders the `myapp` templates
4. **Tiller executes the pre-install hooks**
5. Tiller loads the resulting resources into Kubernetes
6. **Tiller executes the post-install hook**
7. Tiller returns the release data to the client
8. The client exits

- A hook can be any Kubernetes resource
  - A hook is often a Kubernetes job
  - Goes in the `templates` directory

Sharing Charts

- A chart is a directory
  - Easy for a Helm client to use the chart directories on the same computer
  - Difficult to share with other users on other computers

- Packaging a chart
  - Bundle `Chart.yaml` and related files into a tar file

```
$ helm package <chart-path>            # Bundles chart directory into a tar file
$ helm install <chart-name>.tgz        # Installs the chart in the chart file
```

- Chart repository
  - HTTP server that houses an `index.yaml` file and optionally some packaged charts
  - Server can be any HTTP server that can serve YAML and tar files and can answer GET requests
    - Ex: Google Cloud Storage (GCS) bucket, Amazon S3 bucket, Github Pages, or even create your own web server
  - To add a chart to the repository, copy it to the directory and regenerate the index

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

Developing Templates

Creating Templates

- The main aspect of implementing a chart is implementing its templates
- A related task: Create and populate the files that contain the settings used by the templates
  - These settings files, particularly `values.yaml`, define the chart's API
  - The settings files list the variables the templates can use, therefore the only values worth changing
- Examples of chart templates can be found in https://github.com/kubernetes/charts/
  - Each file is a Golang template
  - Includes functions from the Sprig template library
  - A template can create the manifest for any type of Kubernetes resource
- Each file in a chart's `templates` directory is expected to be a template
  - Expected to generate a Kubernetes resource manifest
  - Filename can be anything, should describe the resource it defines
  - Exception: The notes file (i.e. `NOTES.txt`) provides instructions to the chart's users
  - Exception: Files whose names begin with an underscore (e.g. `_helpers.tpl`) are expected to contain partials

Chart Template for Deployment Manifest

Kubernetes Deployment Manifest

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
           - containerPort: 80
```

Helm Deployment Template

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: {{ template "fullname" . }}
  labels:
    app: {{ template "name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version }}
    heritage: {{ .Release.Service }}
    release: {{ .Release.Name }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    metadata:
{{- if .Values.podAnnotations }}
      annotations:
{{ toYaml .Values.podAnnotations | indent 8 }}
{{- end }}
      labels:
        app: {{ template "name" . }}
        release: {{ .Release.Name }}
    spec:
      containers:
        - name: {{ template "name" . }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          ports:
          - name: http
            containerPort: 80
            protocol: TCP
. . .
```

Chart Template for Service Manifest

Kubernetes Service Manifest

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Helm Service Template

```
apiVersion: v1
kind: Service
metadata:
{{- if .Values.service.annotations }}
  annotations:
{{ toYaml .Values.service.annotations | indent 4 }}
{{- end }}
  name: {{ template "fullname" . }}
  labels:
    app: {{ template "name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version }}
    heritage: {{ .Release.Service }}
    release: {{ .Release.Name }}
spec:
  selector:
    app: {{ template "name" . }}
    release: {{ .Release.Name }}
  ports:
    - name: http
      protocol: TCP
      port: {{ .Values.service.port }}
      targetPort: http
      {{- if (and (eq .Values.service.type "NodePort") ...) }}
      nodePort: {{ .Values.service.nodePort }}
      {{- end }}
. . .
```

Values YAML – A Chart’s API

Values (`values.yaml`)

```
replicaCount: 1
restartPolicy: Never

# Evaluated by the post-install hook
sleepyTime: "10"

index: >-
  <h1>Hello</h1>
  <p>This is a test</p>

image:
  repository: nginx
  tag: 1.11.0
  pullPolicy: IfNotPresent

service:
  annotations: {}
  clusterIP: ""
  externalIPs: []
  loadBalancerIP: ""
  loadBalancerSourceRanges: []
  type: ClusterIP
  port: 8888
  nodePort: ""

podAnnotations: {}
resources: {}
nodeSelector: {}
```

Helm Deployment Template

```
. . .
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    metadata:
{{- if .Values.podAnnotations }}
      annotations:
{{ toYaml .Values.podAnnotations | indent 8 }}
{{- end }}
. . .
```

Helm Service Template

```
. . .
spec:
  ports:
    - name: http
      protocol: TCP
      port: {{ .Values.service.port }}
      targetPort: http
      {{- if (and (eq .Values.service.type "NodePort") ...) }}
      nodePort: {{ .Values.service.nodePort }}
      {{- end }}
. . .
```

Chart YAML – A Chart's Meta Information

Chart (`Chart.yaml`)

```
name: nginx
description: A basic NGINX HTTP server
version: 0.1.0
keywords:
  - http
  - nginx
  - www
  - web
home: https://github.com/kubernetes/helm
sources:
  - https://hub.docker.com/_/nginx/
maintainers:
  - name: technosophos
    email: mbutcher@deis.com
```

Helm Template

```
. . .
metadata:
{{- if .Values.service.annotations }}
  annotations:
{{ toYaml .Values.service.annotations | indent 4 }}
{{- end }}
  name: {{ template "fullname" . }}
  labels:
    app: {{ template "name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version }}
    heritage: {{ .Release.Service }}
    release: {{ .Release.Name }}
. . .
```

Chart Template Helpers – More Default Settings

Helpers (templates/_helpers.tpl)

```
{{/* vim: set filetype=mustache: */}}
{{/* Expand the name of the chart. */}}
{{- define "name" -}}
{{- default .Chart.Name .Values.nameOverride | trunc 63 | trimSuffix "-" -}}
{{- end -}}
{{/* Create a default fully qualified app name. We truncate at 63 chars because . . . */}}
{{- define "fullname" -}}
{{- $name := default .Chart.Name .Values.nameOverride -}}
{{- printf "%s-%s" .Release.Name $name | trunc 63 | trimSuffix "-" -}}
{{- end -}}
```

Helm Template

```
. . .
metadata:
  name: {{ template "fullname" . }}
  labels:
    app: {{ template "name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version }}
    heritage: {{ .Release.Service }}
    release: {{ .Release.Name }}
. . .
```

Chart Predefined Values – More Default Settings

Predefined Values

- Release – Information about the release being created
  - Release.Name – The name of the release (not the chart)
  - Release.Service – The service that conducted the release
    - Usually this is Tiller
  - Release.Revision – The revision number
    - It begins at 1, and increments with each helm upgrade
  - Lots of other Release values
- Chart – The contents of the Chart.yaml
  - Chart.Name – The chart name
  - Chart.Version – The chart version
  - Chart.Maintainers – The maintainers
  - Etc.
- Files – Map of all non-special files in the chart
- Capabilities – Map of info about Kubernetes and Helm
  - Capabilities.KubeVersion – Version of Kubernetes
  - Capabilities.TillerVersion – Version of Tiller
  - Capabilities.APIVersions – Kubernetes API versions
- Template – Information about the current template

Helm Chart Template

```
. . .
metadata:
{{- if .Values.service.annotations }}
  annotations:
{{ toYaml .Values.service.annotations | indent 4 }}
{{- end }}
  name: {{ template "fullname" . }}
  labels:
    app: {{ template "name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version }}
    heritage: {{ .Release.Service }}
    release: {{ .Release.Name }}
. . .
```

Conclusion

Conclusion

- What is Helm?
- Using Helm
- Developing charts
- Developing templates

Resources – Introduction

- Helm - The Kubernetes Package Manager
  - https://helm.sh
  - https://docs.helm.sh
  - https://github.com/kubernetes/helm
  - https://github.com/kubernetes/helm/blob/master/docs/index.md
- Taking the Helm: Delivering Kubernetes-Native Applications by Michelle Noorali (KubeCon 2016)
  - https://www.youtube.com/watch?v=zBc1goRfk3k
- Installing Helm
  - https://docs.helm.sh/using_helm/#installing-helm

Resources – Developing Charts

- Helm examples
  - https://github.com/kubernetes/helm/tree/master/docs/examples
- Stable Helm charts
  - https://github.com/kubernetes/charts/tree/master/stable
- Golang templates
  - https://golang.org/pkg/text/template
- Sprig template library
  - https://godoc.org/github.com/Masterminds/sprig
- Getting Started Authoring Helm Charts
  - https://deis.com/blog/2016/getting-started-authoring-helm-charts
- How to Create Your First Helm Chart
  - https://docs.bitnami.com/kubernetes/how-to/create-your-first-helm-chart
- Packaged Kubernetes Deployments – Writing a Helm Chart
  - https://www.influxdata.com/packaged-kubernetes-deployments-writing-helm-chart