Intro to Helm

Joshua Haupt

joshua@hauptj.com

What is Helm?

- “Helm helps you manage Kubernetes applications — Helm Charts help you define, install, and upgrade even the most complex Kubernetes application.”
- “Charts are easy to create, version, share, and publish — so start using Helm and stop the copy-and-paste.”
- Source: helm.sh

Client Server model

- Two parts
  - Helm: client that runs on your local system
  - Tiller: server that runs inside your Kubernetes cluster
    - Manages installations of charts

What it provides

- A package manager for Kubernetes with templating
  - Flow Control
  - Version control
  - Dependency management
- Written in Golang
  - Utilizes the Go template rendering engine

Flow Control

- Enables specifying variables for specific environments
  - Local, development, production
- Control Structures
  - If / else –conditional blocks
  - With – specify a scope
    - Like an “if” statement
  - Range – for / “for each” loop
- Chart template guide on Flow Control

Pipelining Example – if / else

```
{{-  if eq .Values.stage “prod” -}}
                debug: false
                host: example.com
                port: 443
{{- else if eq .Values.stage “dev” -}}
                debug: true
                host: dev.example.com
                port: 443
{{- else -}}
                debug: true
                host: local.example.com
                port: 8443
{{- end -}}
```

Pipelining Example – if / else

```
{{-  if eq .Values.stage “prod” -}}
            debug: false
            host: example.com
            port: 443
{{- else if eq .Values.stage “dev” -}}
            debug: true
            host: dev.example.com
            port: 443
{{- else -}}
            debug: true
            host: local.example.com
            port: 8443
{{- end -}}
```

- **Note:** a dash **and a space** removes whitespace
  - {{- :removes whitespace from the left
  - -}} : removes whitespace from the right

Pipelining Example - with

```
{{- with .Values.production -}}
        debug: false
        host: example.com
        port: 443
        log: {{ .path | quote }}
{{- end -}}
```

- Note: (.) specifies the current scope
- With changes the scope
- Scope is reset with {{end}}
- Values.yaml:

```
production:
    path: /dev/null
```

Flow Control - range

```
users: |-
        {{- range  .Values.users }}
        - {{ . | title | quote }}
        {{- end }}
```

- Values.yaml

```
users:
    - Alice
    - Bob
    - Charlie
```

Version Control with ChartMuseam

- Open source repository for Helm charts
- Supports many 3rd party object storage platforms for storage
- Supports local storage on the filesystem

- github.com/helm/chartmuseum

Dependency Management

- Umbrella / parent charts that reference dependency charts
  - Example: ELK / Elastic stack chart

Live Walkthrough

- What you will need:
  - A Kubernetes environment
    - Minikube will work
  - Helm and Tiller
- Based on Bitnami's How To Create Your First Helm Chart

Initialize Helm

- $ Helm init
- Tiller (the Helm server-side component) has been installed into your Kubernetes Cluster. Happy Helming!

Creating a simple chart

- $ helm create mychart
- $ tree mychart

```
mychart
|-- Chart.yaml
|-- charts
|-- templates
|   |-- NOTES.txt
|   |-- _helpers.tpl
|   |-- deployment.yaml
|   |-- ingress.yaml
|   `-- service.yaml
`-- values.yaml
```

- The **templates directory** is where Helm reads YAML definitions for Kubernetes objects such as Services and Deployments
- **Chart.yaml** contains the metadata that describes and manages the version of the chart
- **Values.yaml** contains the default values that are set at the time of deployment

ChartMuseam via Docker

```
docker run --rm -it \
  -p 8080:8080 \
  -e DEBUG=1 \
  -e STORAGE=local \
  -e STORAGE_LOCAL_ROOTDIR=/charts \
  -v $(pwd)/charts:/charts \
  chartmuseum/chartmuseum:latest
```