Helm

Be smart. Think open source.

What is Helm?

Helm is a tool for managing Kubernetes charts.

Charts are packages of pre-configured resources.

Think of it like apt/yum/homebrew for Kubernetes.

Why Helm?

- De-facto standard for templating k8s configs
- Official Kubernetes project maintained by CNCF
- Repeatable application installations
- Painless updates
- Ships with ready to use charts made by the community
- best practices are baked into offical charts
- Application Lifecycle / Upgrades

What's wrong with kubectl?

- Resources need to be mutated after a deploy
- You can `kubectl edit <resource>`
- But, need to implement your own updating and rollback or depend on vendor specific tooling for lifecycle management

Helm helps with lifecycle managment while allowing you to define your infrastructure as code

Helm Basics

Architecture

- helm client on your local machine (like kubectl)
- tiller server-side component (deployed on Kubernetes)

Quickstart

```
# deploy tiller to kube-system namespace
helm init
# Deploy a postgresql instance
helm install stable/postgresql
```

Inspecting helm managed resources

```
# Show deploys
helm list
# Get information on a release
helm status my-release
```

Helm releases

Differentiate multiple deploys of a chart with releases.

Possible values could be.

- `production`, `stage`, `integration`
- `<customer>-production`
- `<app>-postgresql`

This depends on your organizations structure and what you use the cluster for.

What did we just deploy?

- Deployment based on `postgres` image
- NetworkPolicy allowing access.
- PersistentVolumeClaim for storing data
- Secret containing autogenerated postgres-password

- Service to expose database in cluster

Rollback

```
# simulate rollback
helm rollback --dry-run my-release old-version
# actual rollback
helm rollback my-release old-version
```

Writing helm charts

```
helm create my-app
```

```
my-app/
├── charts
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── ingress.yaml
│   ├── NOTES.txt
│   └── service.yaml
└── values.yaml
```

Templating

The Helm template language is implemented in the strongly typed Go language.

Each Helm chart contains a templates directory that contains relevant templates.

Naming Templates

- Use `.yaml` or `.tpl` suffix for files
- Dasherize file names
- Reflect resource kind in names

Good	Bad
`foo-pod.yaml`	`foo-pod.yml`
`my-example-podtemplates.yaml`	`MyExamplePodTemplates.yaml`
`my-example-svc.yaml`	`my-example.yaml`

Built-in Objects

- Everything below `Release`
- `Chart` variable with infos from `Chart.yaml`
- `Values` with data from `values.yaml`, the cli and other sources

A complete list is in the Helm docs

```
# Current release
{{ .Release.Revision }}
# Chart infos
{{ .Chart.version }}
```

Values

The built-in object Values is empty by default. Charts and users can add to it through `values.yaml`, user-supplied files and on the CLI.

Let's see how values get populated with a simple template.

```
echo '{{ .Values.hello.world }}' > templates/hello.tpl
```

Default values are in a the charts `values.yaml`

```
# values.yaml
hello:
  world: Hello!
```

```
helm template . -x templates/hello.yaml
```

```
---
# Source: my-chart/templates/hello.yaml
Hello!
```

Overriding values on the CLI

```
helm template . --execute templates/hello.yaml \
  --set 'hello.world=Hallo!'
```

```
---
# Source: my-chart/templates/hello.yaml
Hallo!
```

Flow Control

- `if` / `else` for creating conditional blocks
- `with` to specify a scope
- `range`, which provides a "for each"-style loop


```
{{- if .Values.ingress.enabled -}}
apiVersion: extensions/v1beta1
kind: Ingress
# ...
{{- end -}}
```

range

```
# ...
spec:
  rules:
  {{- range .Values.server.ingress.hosts }}
    - host: {{ . }}
  {{- end -}}
# ...
```

Debugging Templates

```
# verify that chart follows best practices
helm lint
# let server render templates and return resulting manifest
helm install --dry-run --debug .
```

Helm SubCharts

Composing systems with subcharts

- Charts can depend on other charts.
- Dependencies are described in `requirements.yaml`.
- `helm dep build` creates `requirements.lock`.

```
# download dependencies
helm dep build stable/redmine

# install redmine with postgresql
helm install stable/redmine \
    --set databaseType.mariadb=false,databaseType.postgresql=true
```

Subcharts and Values

- subcharts cannot depend on their parent charts values.
- parent charts can override values for subcharts.
- global values can be accessed from any chart.

```
# override postgresql values
postgresql:
  postgresPassword: muchsecretverysecure

# define some global variables
global:
  myVariable: myValue
```

Please plan ahead when using globals or don't use them at all. Official charts rarely use them.

Feel Free to Contact Us

www.adfinis-sygroup.ch

Tech Blog

GitHub

info@adfinis-sygroup.ch

Twitter