This document has theoretical and practice aspect of HELM precisely

An Introduction to Helm, the Package Manager for Kubernetes

Introduction

Deploying applications to Kubernetes – the powerful and popular container-orchestration system – can be complex. Setting up a single application can involve creating multiple interdependent Kubernetes resources – such as pods, services, deployments, and replicasets – each requiring you to write a detailed YAML manifest file.

Helm is a package manager for Kubernetes that allows developers and operators to more easily package, configure, and deploy applications and services onto Kubernetes clusters.

Helm is now an official Kubernetes project and is part of the Cloud Native Computing Foundation, a non-profit that supports open source projects in and around the Kubernetes ecosystem.

In this article we will give an overview of Helm and the various abstractions it uses to simplify deploying applications to Kubernetes. If you are new to Kubernetes, it may be helpful to read *An Introduction to Kubernetes* first to familiarize yourself with the basics concepts.

An Overview of Helm

Most every programming language and operating system has its own package manager to help with the installation and maintenance of software. Helm provides the same basic feature set as many of the package managers you may already be familiar with, such as Debian's `apt`, or Python's `pip`.

Helm can:

- Install software.
- Automatically install software dependencies.
- Upgrade software.
- Configure software deployments.
- Fetch software packages from repositories.

Helm provides this functionality through the following components:

- A command line tool, `helm`, which provides the user interface to all Helm functionality.
- A companion server component, `tiller`, that runs on your Kubernetes cluster, listens for commands from `helm`, and handles the configuration and deployment of software releases on the cluster.
- The Helm packaging format, called *charts*.
- An official curated charts repository with prepackaged charts for popular open-source software projects.

We'll investigate the charts format in more detail next.

Charts

Helm packages are called *charts*, and they consist of a few YAML configuration files and some templates that are rendered into Kubernetes manifest files. Here is the basic directory structure of a chart:

Example chart directory

```
package-name/
  charts/
  templates/
  Chart.yaml
  LICENSE
  README.md
  requirements.yaml
  values.yaml
```

These directories and files have the following functions:

- charts/: Manually managed chart dependencies can be placed in this directory, though it is typically better to use `requirements.yaml` to dynamically link dependencies.
- templates/: This directory contains template files that are combined with configuration values (from `values.yaml` and the command line) and rendered into Kubernetes manifests. The templates use the Go programming language's template format.

- Chart.yaml: A YAML file with metadata about the chart, such as chart name and version, maintainer information, a relevant website, and search keywords.
- LICENSE: A plaintext license for the chart.
- README.md: A readme file with information for users of the chart.
- requirements.yaml: A YAML file that lists the chart's dependencies.
- values.yaml: A YAML file of default configuration values for the chart.

The `helm` command can install a chart from a local directory, or from a `.tar.gz` packaged version of this directory structure. These packaged charts can also be automatically downloaded and installed from chart repositories or *repos*.

We'll look at chart repositories next.

Chart Repositories

A Helm chart repo is a simple HTTP site that serves an `index.yaml` file and `.tar.gz` packaged charts. The `helm` command has subcommands available to help package charts and create the required `index.yaml` file. These files can be served by any web server, object storage service, or a static site host such as GitHub Pages.

Helm comes preconfigured with a default chart repository, referred to as stable. This repo points to a Google Storage bucket at `https://kubernetes-charts.storage.googleapis.com`. The source for the stable repo can be found in the helm/charts Git repository on GitHub.

Alternate repos can be added with the `helm repo add` command. Some popular alternate repositories are:

- The official incubator repo that contains charts that are not yet ready for stable. Instructions for using incubator can be found on the official Helm charts GitHub page.
- Bitnami Helm Charts which provide some charts that aren't covered in the official stablerepo.

Whether you're installing a chart you've developed locally, or one from a repo, you'll need to configure it for your particular setup. We'll look into configs next.

Chart Configuration

A chart usually comes with default configuration values in its `values.yaml` file. Some applications may be fully deployable with default values, but you'll typically need to override some of the configuration to meet your needs.

The values that are exposed for configuration are determined by the author of the chart. Some are used to configure Kubernetes primitives, and some may be passed through to the underlying container to configure the application itself.

Here is a snippet of some example values:

values.yaml

```
service:
  type: ClusterIP
  port: 3306
```

These are options to configure a Kubernetes *Service* resource. You can use `helm inspect valueschart-name` to dump all of the available configuration values for a chart.

These values can be overridden by writing your own YAML file and using it when running `helm install`, or by setting options individually on the command line with the `--set` flag. You only need to specify those values that you want to change from the defaults.

A Helm chart deployed with a particular configuration is called a *release*. We will talk about releases next.

Releases

During the installation of a chart, Helm combines the chart's templates with the configuration specified by the user and the defaults in `value.yaml`. These are rendered into Kubernetes manifests that are then deployed via the Kubernetes API. This creates a *release*, a specific configuration and deployment of a particular chart.

This concept of releases is important, because you may want to deploy the same application more than once on a cluster. For instance, you may need multiple MySQL servers with different configurations.

You also will probably want to upgrade different instances of a chart individually. Perhaps one application is ready for an updated MySQL server but another is not. With Helm, you upgrade each release individually.

You might upgrade a release because its chart has been updated, or because you want to update the release's configuration. Either way, each upgrade will create a new *revision* of a release, and Helm will allow you to easily roll back to previous revisions in case there's an issue.

Creating Charts

If you can't find an existing chart for the software you are deploying, you may want to create your own. Helm can output the scaffold of a chart directory with `helm create chart-name`. This will create a folder with the files and directories we discussed in the Charts section above.

From there, you'll want to fill out your chart's metadata in `Chart.yaml` and put your Kubernetes manifest files into the `templates` directory. You'll then need to extract relevant configuration variables out of your manifests and into `values.yaml`, then include them back into your manifest templates using the templating system.

The `helm` command has many subcommands available to help you test, package, and serve your charts. For more information, please read the official Helm documentation on developing charts.

Conclusion

In this article we reviewed Helm, the package manager for Kubernetes. We overviewed the Helm architecture and the individual `helm` and `tiller` components, detailed the Helm charts format, and looked at chart repositories. We also looked into how to configure a Helm chart and how configurations and charts are combined and deployed as releases on Kubernetes clusters. Finally, we touched on the basics of creating a chart when a suitable chart isn't already available.

For more information about Helm, take a look at the official Helm documentation. To find official charts for Helm, check out the official helm/charts Git repository on GitHub.

Practical implementation of helm, here I have used WordPress as an example. Below are the steps one can use for HELM implementation.

Helm Package Manager

Install Helm

To install helm you can follow following instructions.

```
1.  curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get > get_helm.sh
2.  chmod 700 get_helm.sh
3.  ./get_helm.sh
```

Verify the installtion is successful,

```
1.  helm --help
```

Set RBAC for Tiller

`file: tiller-rbac.yaml`

```
1.  apiVersion: v1
2.  kind: ServiceAccount
3.  metadata:
4.    name: tiller
5.    namespace: kube-system
```

```
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: tiller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: tiller
    namespace: kube-system
```

Apply the ClusterRole and ClusterRoleBinding.

```
kubectl apply -f tiller-rbac.yaml
```

Initialize

This is where we actually initialize Tiller in our Kubernetes cluster.

```
helm init --service-account tiller
```

[sample output]

```
Creating /root/.helm
Creating /root/.helm/repository
Creating /root/.helm/repository/cache
Creating /root/.helm/repository/local
Creating /root/.helm/plugins
Creating /root/.helm/starters
Creating /root/.helm/cache/archive
Creating /root/.helm/repository/repositories.yaml
Adding stable repo with URL: https://kubernetes-charts.storage.googleapis.com
Adding local repo with URL: http://127.0.0.1:8879/charts
$HELM_HOME has been configured at /root/.helm.

Tiller (the Helm server-side component) has been installed into your Kubernetes Cluster.

Please note: by default, Tiller is deployed with an insecure 'allow unauthenticated users' policy.
For more information on securing your installation see: https://docs.helm.sh/using_helm/#securing-your-helm-installation
Happy Helming
```

Install Wordpress with Helm

Search Helm repository for Wordpress chart

```
helm search wordpress
```

Fetch the chart to your local environment and change directory.

```
1. helm fetch --untar stable/wordpress
2. cd wordpress
```

Create a copy of default vaules file and edit it.

```
1. cp values.yaml my-values.yaml
2. vim my-values.yaml
```

Run it as a dry run to check for errors.

```
1. helm install --name blog --values my-values.yaml . --dry-run
```

Deploy the Wordpress stack.

```
1. helm install --name blog --values my-values.yaml .
```