So before we can use helm with a kubernetes cluster, you need to install tiller on it. It’s as easy as running :

helm init

**Building a Helm chart**

Let’s see Helm in action, using a small little Go test api I created specifically for testing use cases like this, let’s build a helm chart of it.

git clone https://[github.com/daemonza/testapi.git](http://github.com/daemonza/testapi.git); cd testapi

First create a skeleton structure chart

helm create testapi-chart

This will create a testapi-chart directory. Inside this directory the three files we are the most interested in for is Chart.yaml, values.yaml and NOTES.txt.

* Chart.yaml describes the chart, as in it’s name, description and version.
* values.yaml is stores variables for the template files templates directory. If you have more complex deployment needs, that falls outside the default templates capability, edit the files in this directory. They are normal Go templates, Hugo ([https://gohugo.io](https://gohugo.io/)) which btw powers this blog, have a nice Go template primer (<https://gohugo.io/templates/go-templates/>), if you need more information on how to work with Go templates.
* NOTES.txt is used to give information after deployment to the user that deployed the chart. For example it might explain how to use the chart, or list default settings, etc. For this post I will keep the default message in it.

Open Chart.yaml and fill out the details of the application your deploying. Using the testapi as a example, this is how my Chart.yaml looks like :

apiVersion: v1

description: A simple api for testing and debugging

name: testapi-chart

version: 0.0.1

Now open values.yaml and edit it as needed. Again, using the testapi as a example this is how my values.yaml file looks like.

replicaCount: 2

image:

repository: daemonza/testapi

tag: latest

pullPolicy: IfNotPresent

service:

name: testapi

type: ClusterIP

externalPort: 80

internalPort: 80

resources:

limits:

cpu: 100m

memory: 128Mi

requests:

cpu: 100m

memory: 128Mi

Run

helm lint

in your testapi\_chart directory to make sure everything is ok. If everything is good, you can package the chart as a release by running :

helm package testapi-chart --debug

I like to add the --debug flag to see the output of the packaged chart. Output should look similar to the following

Saved /Users/daemonza/testapi/testapi-chart/testapi-chart-0.0.1.tgz to current directory

Saved /Users/daemonza/testapi/testapi-chart/testapi-chart-0.0.1.tgz to /Users/daemonza/.helm/repository/local

From that we can see that the chart is placed in our current directory as well as in our local helm repository. To deploy this release, we can point helm directly to the chart file as follows :

helm install testapi-chart-0.1.0.tgz

And your output should look similar to the following :

NAME: ordered-quoll

LAST DEPLOYED: Wed Mar 1 09:39:48 2017

NAMESPACE: default

STATUS: DEPLOYED

RESOURCES:

==> v1/Service

NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE

ordered-quoll-testapi-ch 10.0.0.133 <none> 80/TCP 0s

==> extensions/Deployment

NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE

ordered-quoll-testapi-ch 2 2 2 0 0s

NOTES:

1. Get the application URL by running these commands:

export POD\_NAME=$(kubectl get pods --namespace default -l "app=ordered-quoll-testapi-ch" -o jsonpath="{.items[0].metadata.name}")

echo "Visit [http://127.0.0.1:8080](http://127.0.0.1:8080/) to use your application"

kubectl port-forward $POD\_NAME 8080:80

From the above we can see that a deployment was created in kubernetes, the testapi got scaled to two pods and a service got created to expose the deployment on the cluster IP on port 80. And the NOTES.txt file tells us how to access the pod.

List the deployed packages with their release versions by running :

helm ls

Which should return output similar to the following

NAME REVISION UPDATED STATUS CHART

ordered-quoll 1 Wed Mar 1 11:48:52 2017 DEPLOYED testapi-chart-0.1.0

Modify the Chart.yaml file and change the version from 0.1.0 to 0.1.1 package and deploy the 0.1.1 chart. Running helm ls again now shows us that we have two packages of the testapi deployed

ordered-quoll 1 Wed Mar 1 11:48:52 2017 DEPLOYED testapi-chart-0.1.0

wishful-ibis 1 Wed Mar 1 12:03:31 2017 DEPLOYED testapi-chart-0.1.1

Let’s confirm that the testapi indeed deployed, and that there is two versions of it running :

kubectl get deployments

You should have output similar to :

NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE

ordered-quoll-testapi-cha 1 1 1 1 11m

wishful-ibis-testapi-cha 1 1 1 1 3m

Time to get rid of the older 0.1.0 deployment of the testapi chart. Using it’s package name from helm ls run :

helm delete ordered-quoll

Confirm again with kubectl get deployments that it really is removed. Expected output similar to :

NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE

wishful-ibis-testapi-cha 1 1 1 1 7m

As we can see the ordered-quoll 0.1.0 version is removed. But what happens if we want to go back. Imagine this scenario, we deployed a new version of a application, and for some reason its got a problem and we need to rollback to the previous version. No worries, helm got our back. Simply run

helm rollback ordered-quoll 1

Which means we want to rollback the testapi package ordered-quoll one revision back.

Expected output

Rollback was a success! Happy Helming!

But what, if you cannot remember what the name was of a deleted package? Or just want to see all the packages that’s been deleted? No problem, run :

helm ls --deleted

And if you want to see it ordered by date just add a -d

We have really gone a round about way of deploying packages so far with helm, normally you would only want to upgrade a package, instead of deploying a new version alongside it, unless off course your following a blue / green style deployment process.

Let’s test upgrading a release. Open the testapi project, edit the Chart.yaml as a example and change the description, and version number, package the release with helm package ., but now instead of using helm install run

helm upgrade ordered-quoll .

This will upgrade our ordered-quoll release to the changes we just made. Running helm ls now, you should see the new version next to ordered-quoll. And off course we can rollback this release as before.

For more information on using Helm look at <https://github.com/kubernetes/helm/blob/master/docs/using_helm.md>

While this works well, it would be nicer to put our chart in a Helm repository, as it’s then easy to share the chart or access it from other clusters, etc.

**Setting up Helm repository**

A Helm repository is nothing more than just a web server that’s able to serve a index.yaml file and chart files, which is really just tar.gz file containing the generated kubernetes resource manifest files from our helm chart templates. So almost any web server will do. For local testing you can also use the helm command itself. Here is a example of helm serving the charts from a charts directory.

helm serve --repo-path ./charts

Problem is you need to get your charts onto the web server somehow, and there is a myriad amount of solutions on how to do it, it can as a example be as simple as just using scp to get your chart to the web server, or setting up Caddy(<https://caddyserver.com/>) with the Upload plugin(<https://caddyserver.com/docs/upload>). You can also use AWS S3 or a Google GCS bucket as well to host a chart repository, which does make uploading easier, by using the google cloud command line utility or the UI or for S3 use one of the many S3 tools out there. I personally prefer s3cmd (<http://s3tools.org/s3cmd>) for uploading files to S3.

However I wanted something a little simpler, that generates the Helm index for me, without me having to do it by hand, and also be able to host it myself in a Kubernetes cluster, so I wrote a small Go server called Helmet to act as my helm repository. It’s basically just a web server, to which you can upload chart files using something like curl and it then handles the repository indexing for you using helm in the backend.

You can find out more about Helmet at <https://github.com/daemonza/helmet>

**Using a Helm repository**

Using Helmet as our helm repository, let’s deploy it to our Kubernetes cluster and then add a chart. And off course we can deploy Helmet with Helm :D

git clone <https://github.com/daemonza/helmet.git>; cd helmet/helmet-chart

helm package . --debug

A helmet chart should be created, in my case it is helmet-chart-0.0.1.tgz

Deploy the same way we deployed the testapi.

helm install helmet-chart-0.0.1.tgz --debug

For this blog post, I deployed everything to Kubernetes Minikube(<https://github.com/kubernetes/minikube>) So using minikube, let’s see how we can access helmet

HELMET=(`kubectl get services | awk '/helmet/ {print $1}'`)

minikube service $HELMET --url

Which gives me back [http://192.168.99.100:31162](http://192.168.99.100:31162/). Using this URL, let’s add it as a helm repository.

helm repo add helmet <http://192.168.99.100:31162/charts/>

And confirm that our helmet repo is there by running helm repo list. You should see the following :

NAME URL

stable <https://kubernetes-charts.storage.googleapis.com/>

helmet <http://192.168.99.100:31162/charts/>

We can now add our testapi chart to this helmet repository with :

curl -v -T testapi-chart-0.1.1.tgz -X PUT http://192.168.99.100:31162/upload/

We can confirm that the chart is uploaded and the helm repo index got created by running

curl <http://192.168.99.100:31162/charts/index.yaml>

Which will give us the following output :

apiVersion: v1

entries:

testapi-chart:

- apiVersion: v1

created: 2017-03-01T12:11:09.746867088Z

description: A Helm chart for Kubernetes

digest: fe0c17d87b523c91cc59bd1e4d2f997defb2a215c4cc0fc02a1725922471e88a

name: testapi-chart

urls:

- http://masked-macaw-helmet-char:1323/charts/testapi-chart-0.1.1.tgz

version: 0.1.1

generated: 2017-03-01T12:11:09.746407601Z

We can now search for the testapi, across all our repositories.

helm search testapi

Expected output :

NAME VERSION DESCRIPTION

helmet/testapi-chart 0.1.1 A Helm chart for Kubernetes

Let’s try searching for something else. Searching for Jenkins gives us this

NAME VERSION DESCRIPTION

stable/jenkins 0.1.14 Open source continuous integration server. It s...

Which we can see comes from the stable repository. Very nice! Have I mentioned I love helm? :D Now, to continue let’s install the testapi from our helmet repository :

helm install helmet/testapi-chart

And that’s how simple it is to use helm repositories. We can now use this helmet helm repository from anywhere upload charts to and deploy from.