Helm Charts

Helm chart is a package manager for Kubernetes which helps in combining the YAML definition to single package. Once packaged, installing a Helm Chart into your cluster is as easy as running a single helm install.

Advantages

- Boosts productivity

An example of this might be that, in order to test a new feature, an engineer needs a SQL database. Instead of going through the process of installing the software locally, creating the databases and tables required, the engineer can simply run a single Helm Install command to create and prepare the database ready for testing.

- Reduces duplication & complexity

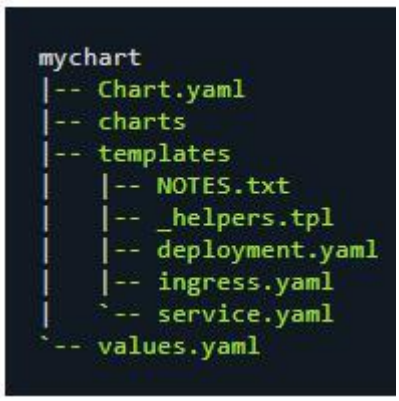
Once the chart is built once, it can be used over and over again and by anyone. The fact that you can use the same chart for any environment reduces complexity of creating something for dev, test and prod.

- Enables the easy integration to ci/cd

  Helm easily integrates into CI/CD pipelines and allows software engineers to focus on writing code—not deploying applications.

- Simplifies deployments

Deploying applications to Kubernetes is not a straightforward process, with different objects being tightly coupled. This required specific knowledge of these objects and what their functions are in order to be able to successfully deploy. Helm takes the complexity out of that doing much of the hard work for you.



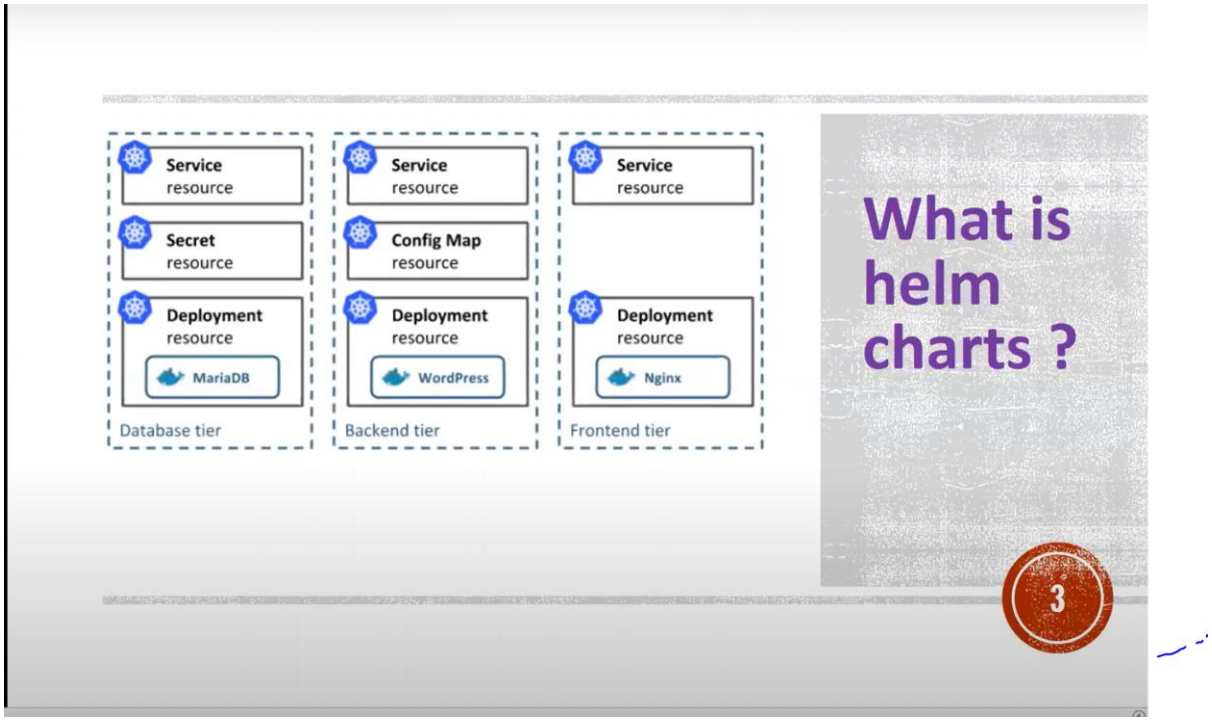
.helmignore: This holds all the files to ignore when packaging the chart. Similar to .gitignore, if you are familiar with git.

Chart.yaml: This is where you put all the information about the chart you are packaging. So, for example, your version number, etc. This is where you will put all those details.

Values.yaml: This is where you define all the values you want to inject into your templates. If you are familiar with terraform, think of this as helms variable.tf file.

Charts: This is where you store other charts that your chart depends on. You might be calling another chart that your chart need to function properly.

Templates: This folder is where you put the actual manifest you are deploying with the chart. For example you might be deploying an nginx deployment that needs a service, configmap and secrets. You will have your deployment.yaml, service.yaml, config.yaml and secrets.yaml all in the template dir. They will all get their values from values.yaml from above.



_hepers.tpl: This is file is used to create a templates which we can refer in the manifest files.

Example: Create a template in _helpers.yaml

```
{{- define "mychart.labels" }}
  labels:
    generator: helm
    date: {{ now | htmlDate }}
    chart: {{ .Chart.Name }}
    version: {{ .Chart.Version }}
{{- end }}
```

And refer in the deployment files

```
{{- include "mychart.labels" . }}
```

NOTES.txt: This file helps in displaying any any information at the time of helm installation.

Named_templated

Quote

```
{{ quote .Values.favorite.food }}
```

Repeat

```
{{  .Values.favorite.drink | repeat 3 | quote }}
```

Upper

```
{{  .Values.favorite.drink | upper | quote }}
```

Lower

```
{{  .Values.favorite.drink | lower | quote }}
```

Include

```
{{- include "mychart.selectorLabels" . }}
```

Required

```
{{ required “Enter the city” .Values.favorite.city }}
```

Default

```
{{.Values.favorite.city | default "Bangalore"}}
```

Nindent

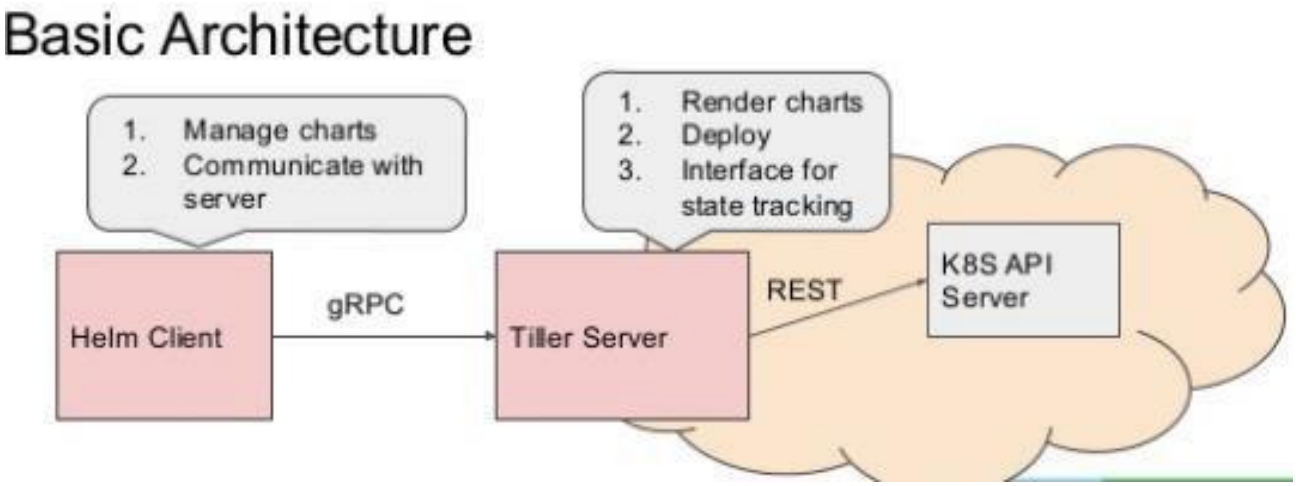
```
    matchLabels:
      {{- include "mychart.selectorLabels" . | nindent 6 }}
```

coalesce

```
  city: {{ coalesce .Values.town .Values.favorite.city }}
```

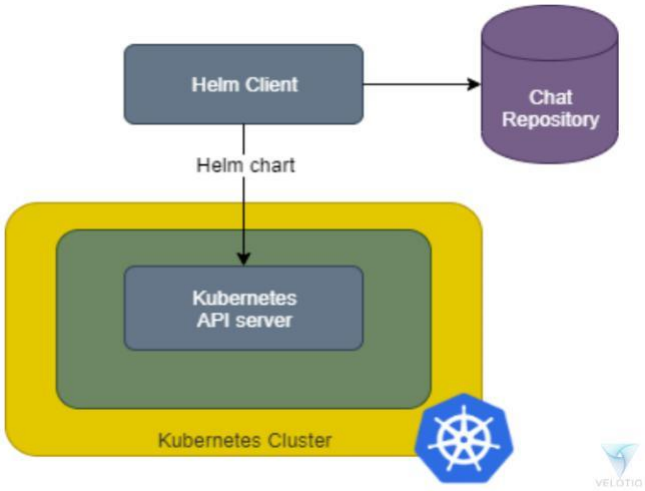
Architecture

Helm2



Helm 2 is a client-server architecture with the client called helm and the server called Tiller. The client is a CLI that users interact with to perform different operations like install, upgrade, and delete. As seen in the following diagram, the client interacts with Tiller and the chart repository. Tiller interacts with the Kubernetes API server. It renders Helm template files into Kubernetes manifest files, which it uses for operations on the Kubernetes cluster through the Kubernetes API.

Helm3(client only archi)



Difference bw helm2/helm3

Helm 3 has a client-only architecture with the client still called helm. As seen in the following diagram, it operates similar to the Helm 2 client, but the client interacts directly with the Kubernetes API server. The in-cluster server Tiller is now removed.

Difference bw helm2/helm3

Helm2	**Helm3**
Tiller server presen	Tiller server is removed
Chart dependencies should be mentioned in requirement.yaml	Chart dependencies can be mention in chart.yaml / requirement.yaml
Auto-genarate name is present	Auto-genarate name is absent, unless the auto-generate
Release details saved in configMaps	Release details saved in secrets.
Namespace will be created automatically	Namespace will not get created.
(CLI changes) helm delete helm inspect helm fetch ..	(CLI changes) Helm remove Helm show Helm pull ..

Helm default objects

Object which we can refer in the chart template

Release.Name : To get the release name

Chart.Name : To get the chart name

Chart.Version : To get the chart version

..

Helm control structure

If/else: for createing the conditional block

```
  {{ if eq .Values.favorite.drink "coffee"
  }} mug: true
  {{ end }}
```

range: for each loop

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

with: to specify a scope

```
  {{- with .Values.favorite }}
  drink: {{ .drink | default "tea" | quote }}
  food: {{ .food | upper | quote }}
  {{- end }}
```

Helm dependency

Given the following tags in tht charts.yaml we can include our sub charts to our deployment

```
dependencies:
- condition: mariadb.enabled
  name: mariadb
  repository: https://charts.bitnami.com/bitnami
  version: 9.x.x
```

Helm Hooks

Helm provides a hook mechanism to allow chart developers to intervene at certain points in a release's life cycle example at `pre-install, post-install, pre-upgrade, post-delete`

Declaring the annotation in the yaml manifest file we can run the particular object as a hook. Following manifest files has to be placed under template folder.

```
  annotations:
    # This is what defines this resource as a hook. Without this line, the
    # job is considered part of the release.
    "helm.sh/hook": post-install
    "helm.sh/hook-weight": "-5"
    "helm.sh/hook-delete-policy": hook-succeeded
```

`hook-weight` To maintain the sequence for hooks in case if there is a multiple post-install hooks

`hook-delete-policy` Delete policy for the hook

Helm command

----------Commands to create, validate, list install, uninstall ----------
helm create <chart directory>
helm lint <chart directory>
helm install <release name> --dry-run --debug <chart directory>
helm install <release name> --dry-run --debug <chart directory>
helm install <release name> <chart directory> --set service.type=NodePort
helm inspect
helm list
helm uninstall <release name>
---------- Upgrades ----------
helm upgrade --install <release name> --values <values file> <chart directory>
helm upgrade --install <release name> . <chart directory> --set image.tag=1.16.1
----------Rollback----------
helm ls
helm rollback <release name> 1
helm ls
----- web search ----------
helm repo add stable https://charts.helm.sh/stable
helm search repo nginx
----------Package----------
helm package mychart/
---------- plugins Install, Push chart ----------
helm plugin
helm plugin install https://github.com/chartmuseum/helm-push
helm push <repo url>

References
https://github.com/muddukrishnahm/Helm_charts
https://helm.sh/docs/chart_template_guide/getting_started/