Helm

Kubernetes Application

- Kubernetes application are the resources that are deployed on a cluster
  - Eg. pods, volumes, volume claims, secrets, ingress, etc
  - Resources maps to servers/nodes, disk, network, etc
- Resources are describe by Kubernetes objects
  - Think the YAML file
  - Eg. Deployment
- Some application requires many YAML file
  - Eg. WordPress application requires about 13 Kubernetes objects
- Complex if everyone need to repeatedly create these YAML files repeatedly
- Package manager can automate deployment of complex applications

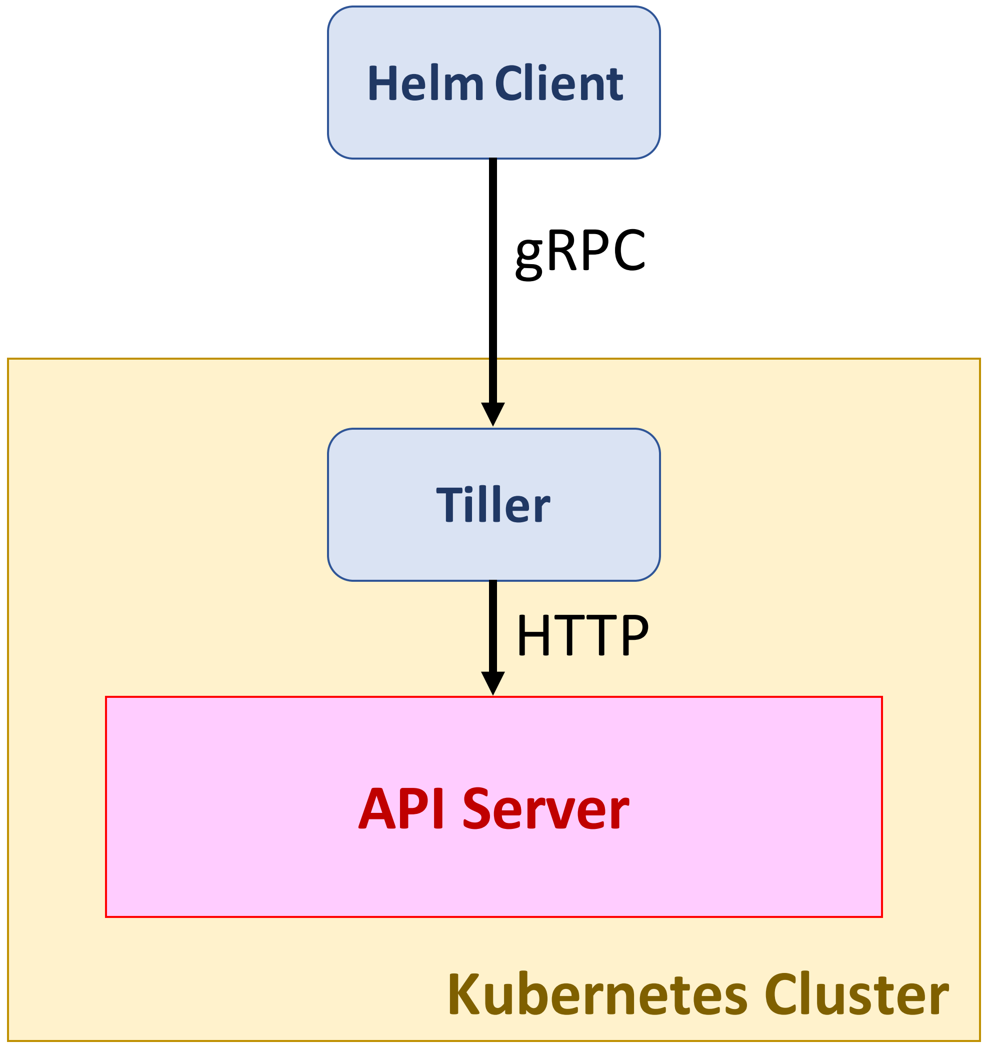
What is Helm?

- Package manager for Kubernetes applications
  - Like NPM, apt,
- Applications are packaged in Charts
- Benefits of using Helm
  - Single command to provision an application instead of multiple `kubectl create`/`delete` command
  - Easily upgrade or rollback releases
    - Releases are apps that Helm installed in a Kubernetes cluster
  - Charts can be versioned allow developers to keep track of their application
  - Use Charts published by others
    - Eg. to deploy MySQL, search for an appropriate MySQL Chart

Helm Architecture

- Helm Client
  - CLI to issue commands to tiller
- Tiller
  - Installed in a Kubernetes cluster
  - Accepts commands from Helm client
  - Interacts with the API Server to manage Kubernetes objects
  - Note: Tiller will be removed in Helm V3
- Install helm

```
helm init
```



Installing Helm

- Download an appropriate release
  - https://github.com/helm/helm/releases
- Create a service account called tiller and give it a cluster wide role binding of `cluster-admin`

```
kubectl -n kube-system create serviceaccount tiller
kubectl create clusterrolebinding tiller \
  --clusterrole cluster-admin \
  --serviceaccount=kube-system:tiller
```

- Install Tiller

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

Concepts

- Charts
  - A Helm package
  - Consists of parameterized Kubernetes resource definitions
  - Meta information like package name, version, substitutable values
- Release
  - A instances of an installed chart running in Kubernetes
  - You can set the release name when installing an application or helm will generate a name
- Repository
  - Public location of a chart
  - The helm comes preconfigured with the stable repository

Helm Command

- Searching the repository
  - Will list as `<repo-name>/<package-name>`

    ```
    helm search mysql
    ```
- Install a package
  - See package documentation at https://hub.helm.sh
  - Eg. For MySQL https://hub.helm.sh/charts/stable/mysql

    ```
    helm install stable/mysql --name warehouse
    ```
- List the releases (installed packages)

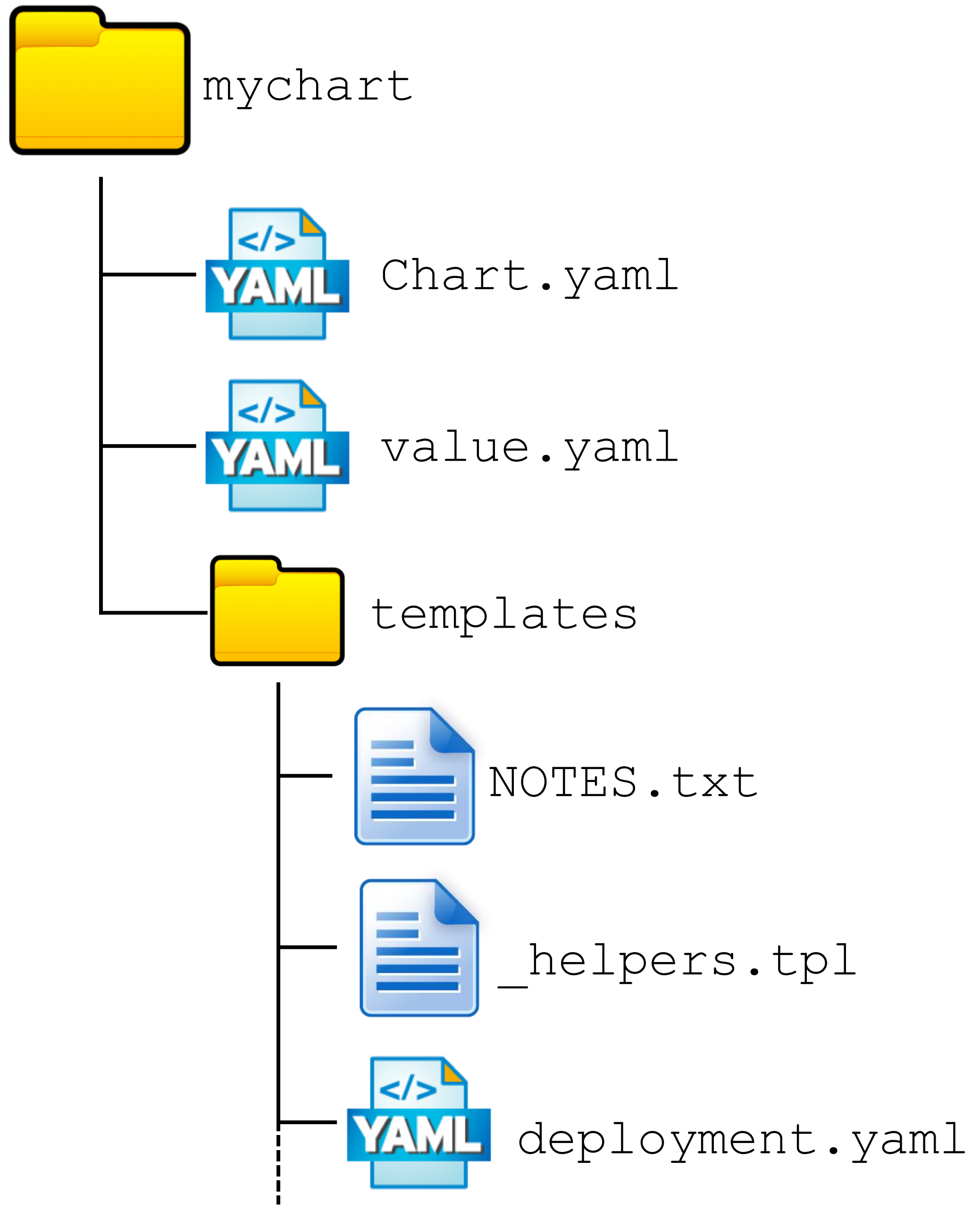
  ```
  helm list --all
  ```
- Delete a package

  ```
  helm delete --purge warehouse
  ```

Create a Chart

```
helm create mychart
```

- Creates a directory with the following structure/contents
  - `Charts.yaml` - contains information about the chart
  - `values.yaml` - holds the default values of the chart
    - Templates lookup values from here
  - `NOTES.txt` - help text. Will be displayed after the chart has been installed
  - `_helpers.tpl` - template helpers
  - One or YAML files that defines Kubernetes resources
    - Eg. `deployment.yaml`, `service.yaml`, `ingress.yaml`, etc.



Example of a Template

```
apiVersion: apps/v1
kind: Deployment

metadata:
  name: {{ include "mychart.fullname" . }}
  labels:
    app.kubernetes.io/name: {{ include "mychart.name" . }}
    app.kubernetes.io/instance: {{ .Release.Name }}
    app.kubernetes.io/managed-by: {{ .Release.Service }}
    helm.sh/chart: {{ include "mychart.chart" . }}

spec:
  replicas: {{ .Values.replicaCount }}
```

Expression to be evaluated

Helm function that inserts a template

Build in object with data on the current release

Values from `values.yaml` file

Templates

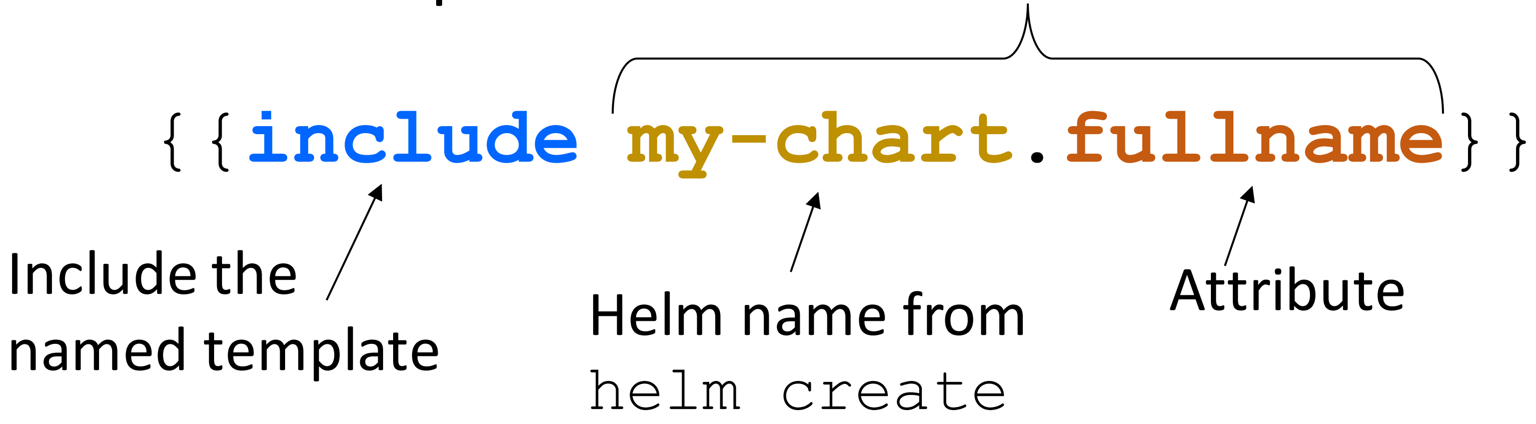
- Uses Golang template engine
  - Additional features from Sprig and Helm
  - See https://golang.org/pkg/text/template
- Build in objects
  - Values - from `values.yaml`
  - Release - information about the current release
    - `Release.Name`, `Release.Time`, etc
  - Chart - from `Chart.yaml`
  - Template - information about the current template
    - `Template.Name`, `Template.BasePath`
  - See https://github.com/helm/helm/blob/master/docs/chart_template_guide/builtin_objects.md

Example of Build-in Objects

- `Release` - details of this release
  - `.Name` - release name
  - `.Namespace` - namespace release into
  - `.Revision` - revision number
- `Values` - the namespace for `values.yaml` file
  - Eg. `Values.username`
- `Capabilities` - information about Kubernetes
  - `.KubeVersion` - Kubernetes version
  - `.APIVersion` - Set of versions
- `Template` - information about the current template being evaluated
  - `.Name` - path of the current template

Chart name

- Helm creates named templates to standardize naming
  - `name` - chart name eg. `my-chart` becomes `mychart`
  - `fullname` - fully qualified name eg. `thisrelease-mychart`
  - `chart` - chart name with version number eg. `mychart-1`
- Defined in `_helpers.tpl` in templates
- Uses the following convention `<cart-name>.<attribute>` as names for the template



```
{{include my-chart.fullname}}
```

Expression

deployment.yaml

```
imagePullPolicy: {{.Values.image.pullPolicy}}
```

Start at the 'root' of the Values object

values.yaml

```
image:
  repository: mysql
  tag: stable
  pullPolicy: IfNotPresent
```

Condition

If key exists then execute the body

```
{{if .Values.cipher}}
  encryptWith: {{.Values.cipher | lower | quote}}
{{end}}
```

Pipeline produces the quoted lower case string eg. `AeS` to `"aes"`

Loop

```
volumes:
- name: tmp
  mountPath: /opt/tmp
- name: mysql
  mountPath: /var/lib/mysql
```

```
containers:
- name: app
  volumeMounts:
{{range $vol := .Values.volumes}}
    - name: {{$vol.name}}
      mountPath: {{$vol.mountPath}}
{{end}}
```

Loops - Index

```
values.yaml

users:
- fred
- barney
- wilma
- betty
```

```
{{range $value := Values.users}}
- name: $value
{{end}}
```

Output

```
- name: fred
- name: barney
- name: Wilma
- name: betty
```

```
{{range $index, $value := Values.users}}
- uid: $index
  name: $value
{{end}}
```

Output

```
- uid: 0
  name: fred
- uid: 1
  name: barney
- uid: 2
  name: Wilma
- uid: 3
  name: betty
```

If range over an object, then `$index` will hold the key

With

```
volumes:
- name: tmp
  mountPath: /opt/tmp
- name: mysql
  mountPath: /var/lib/mysql
```

```
containers:
- name: app
  volumeMounts:
{{range $vol := .Values.volumes}}
    {{with $vol}}
    - name: {{.name}}
      mountPath: {{.mountPath}}
    {{end}}
{{end}}
```

Sets the current scope to start from `$vol`

Kubernetes Standard Labels

- Kubernetes define a set of standard labels to allow different tools to interoperate with each other based on these labels
  - Eg instead of labelling your deployment name with app, name
- Recommended standard labels
  - `app.kubernetes.io/name` - the application name
  - `app.kubernetes.io/instance` - unique instance
  - `app.kubernetes.io/version` - version number
  - `app.kubernetes.io/part-of` - name of the higher level application eg. `database` deployment is part of `mycrm`
- See https://kubernetes.io/docs/concepts/overview/working-with-objects/common-labels

Recommended Template Setup

```
apiVersion: apps/v1
kind: Deployment

metadata:
  name: {{ include "mychart.fullname" . }}
  labels:
    app.kubernetes.io/name: {{ include "mychart.name" . }}
    app.kubernetes.io/instance: {{ .Release.Name }}
    app.kubernetes.io/managed-by: {{ .Release.Service }}
    helm.sh/chart: {{ include "mychart.chart" . }}

spec:
  ...
```

Fixed value: Tiller

Deploying Your Chart

- Installing
  - Assume you are issuing the command from inside `my-chart` directory

    ```
    helm install ../my-chart
    ```
  - Helm will generate a release name
- with a specific release name

  ```
  helm install ../my-chart --name my-release
  ```
- Render the templates without installing it

  ```
  helm install ../my-chart --dry-run --debug
  ```

Customizing a Release

- Examine the `values.yaml` file

```
helm inspect values <repo-name>/<chart-name>
```

- Create a file `myvalues.yaml` with the values that you wish to update
- Install the chart with myvalues.yalm to override the default values

```
helm install <repo-name>/<chart-name> \
      -f myvalues.yaml
```

Publishing to Your Own Repository

- See https://medium.com/containerum/how-to-make-and-share-your-own-helm-package-50ae40f6c221