Kubernetes Advanced

kubernetes

Session #06
Helm

kubernetes

Session Contents

- What is Helm
- How to use Helm
- Helm Charts
- Author your chart

kubernetes

What is Helm

Motivation

- To deploy your applications, you need to create several manifest files
- When you want to deploy on different target clusters/environments you may need to make minor changes to reflect those differences
- Additionally, you may want to publish your manifest files on a centralized registry to make it available for other people/teams to reuse them
- To achieve this, a package manager-like tool is what you need
- Helm is the Kubernetes Package Manager

kubernetes

Helm Architecture

- Helm is a tool for managing Kubernetes packages called charts
- Helm can do the following:
  - Create new charts from scratch
  - Package charts into chart archive (tgz) files
  - Interact with chart repositories where charts are stored
  - Install and uninstall charts into an existing Kubernetes cluster
  - Manage the release cycle of charts that have been installed with Helm

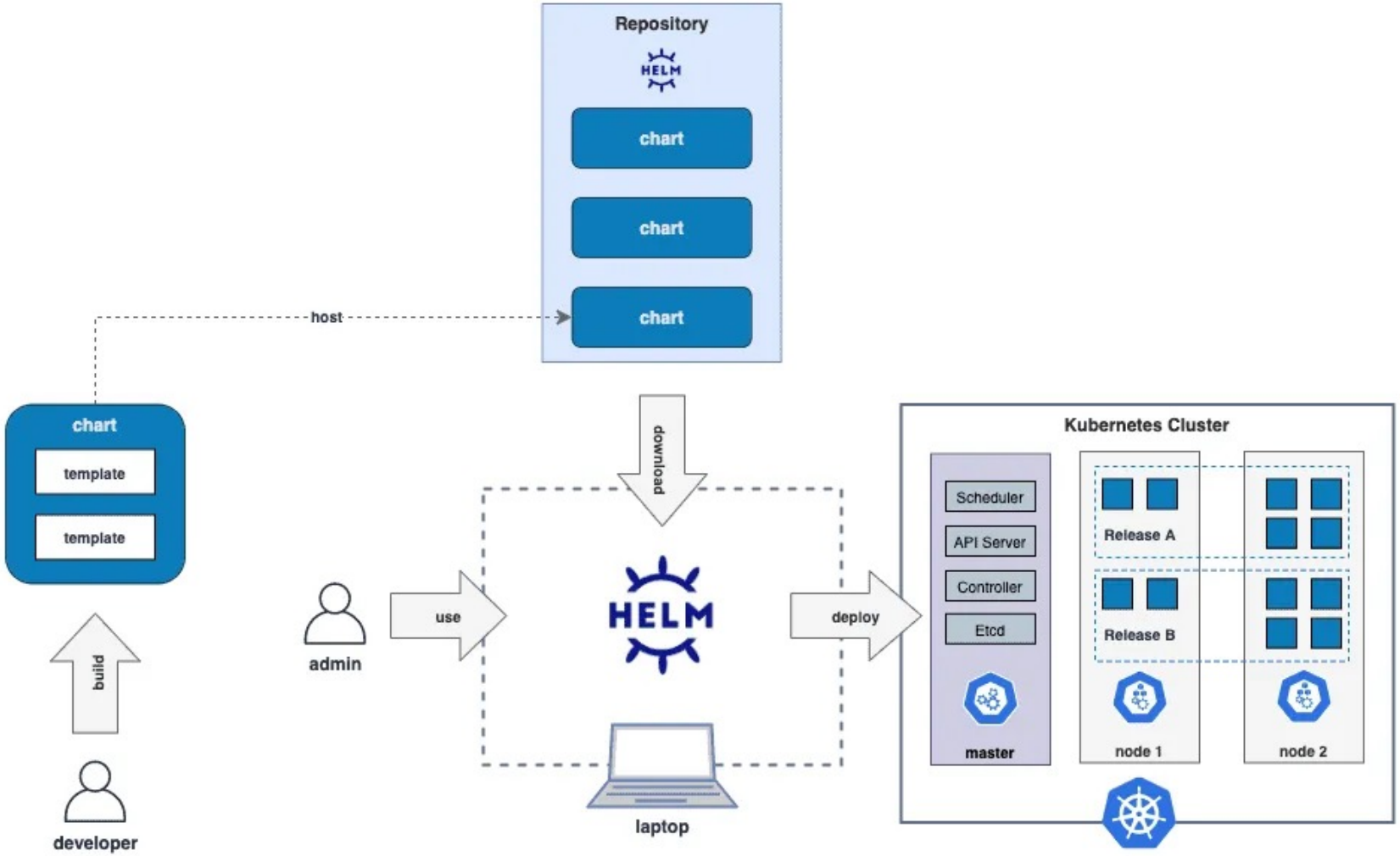
kubernetes

Helm Concepts

- The chart is a bundle of information necessary to create an instance of a Kubernetes application
- The config contains configuration information that can be merged into a packaged chart to create a releasable object.
- A release is a running instance of a chart, combined with a specific config
- Optionally, you may use a registry where you place/publish your charts to be used by other

kubernetes

Helm Concepts



kubernetes

Helm Benefits

- Deployment speed: you can deploy any application available at the Helm chart repository within a single command.
- Prebuilt application configurations: Helm allows you to install community-supported applications with ease.
- Easy rollbacks: Helm allows you to easily roll back your application deployment to the previous version if something goes wrong.

kubernetes

How to use Helm

Steps to use Helm

1. Install Helm
2. Initialize a Helm Chart Repository
3. Use Helm Chart

kubernetes

Install Helm

- From Binary Releases
- Using a Helm Installation Script -> Preferable way
- Using Package Manager
  - Brew
  - Chocolatey
  - Apt, Yum, ...

kubernetes

Initialize a Helm Chart Repository

- Before you run a Helm Chart you need to have access to them
- You can author your charts (next topic) or you can download them
- To download a pre-existing chart, you may connect to a repository
- The way you interact with this repositories are similar the way you use any package manager

kubernetes

Helm OCI Registries

- OCI (Open Container Initiative) have a registry specification for container images registries
- Helm (on version 3) can use container registries with OCI support to store helm charts
- Several container registries have OCI Support
  - Docker Hub
  - GitHub Packages
  - All cloud providers Container Registries
- Central place to find Helm Charts is ArtifactHUB

kubernetes

Use Helm Chart

- After getting access to the repository, you may search the helm chart that you want to use
- After knowing is name and version, you install the helm chart on your cluster
- Helm uses your active `kubectl` context configuration (cluster and permissions) to install the Helm Chart

kubernetes

Helm Chart Values

- To allow a dynamic behavior of a Helm Chart you may set different inputs when installing your Helm Chart
- You can pass those values directly on command line commands
- But to have a consistent way of configure different environments/context you should use Helm Chart Values file

kubernetes

Demo | Use Helm

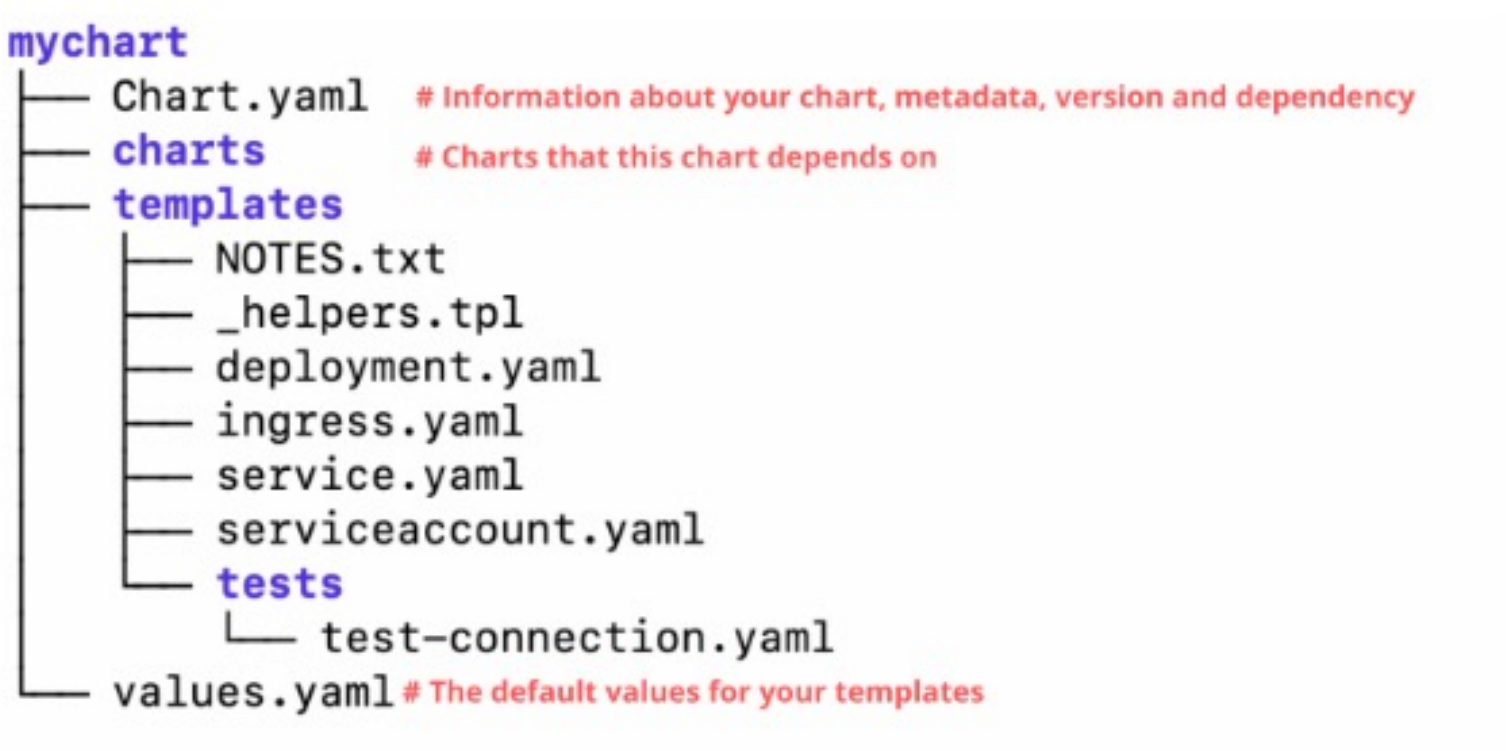
Helm Charts

Motivation

- Until now we've used Helm to deploy chart that someone produced, like using Docker Hub to pull images
- Then I want to create my own charts, from my own applications, to make deploys
- After having those charts working, I may (or not) publish them to a registry

kubernetes

Helm Chart Structure



kubernetes

Helm: Chart File

- Main file that makes a folder be a Helm Chart folder
- You can see it like chart metadata
- Includes
  - Name
  - Description
  - Chart version
  - App version (different from chart version)

kubernetes

Helm: values File

- This file allows you to define values (variables) that can be set when someone uses your chart
- Can be more restrictive or more open
- All variables defined where can be used on your templates

kubernetes

Helm: templates Folder

- Where you define all Kubernetes objects that your application will need
- Are created using a Helm templating language that can use variables to implement dynamic behavior
- Can use functions to add extra features like conditionals and cycles
- Can have as much templates as needed to create all Kubernetes resources
- This files uses Go templating to make it fully dynamic

kubernetes

Author your Charts

Helm Templating

- Helm uses Go Templating language
- All template code needs to be placed inside {{ }}
- Everything inside this pattern will be translated automatically and producing a text file
- If you miss the format or the code inside it you'll an error an the template will not be produced
- Again, biggest issue is about YAML indentation

kubernetes

Templates: Built-in Objects

- Some static built-in objects can be used to retrieve information from Helm details
- `Release`: This object describes the release itself
- `Values`: Values passed into the template from the `values.yaml` file and from user-supplied files
- `Chart`: The contents of the `Chart.yaml` file. Any data in `Chart.yaml` will be accessible here.

kubernetes

Templates: Built-in Objects

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favoriteDrink }}
```

kubernetes

Templates: Built-in Objects

```yaml
# values.yml

favorite:
  drink: coffee
  food: pizza
```

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favorite.drink }}
  food: {{ .Values.favorite.food }}
```

kubernetes

Templates: Functions

- You can use built-in function on templating using 2 approaches

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ quote .Values.favorite.drink }}
  food: {{ quote .Values.favorite.food }}
```

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favorite.drink | quote }}
  food: {{ .Values.favorite.food | quote }}
```

kubernetes

Templates: Functions

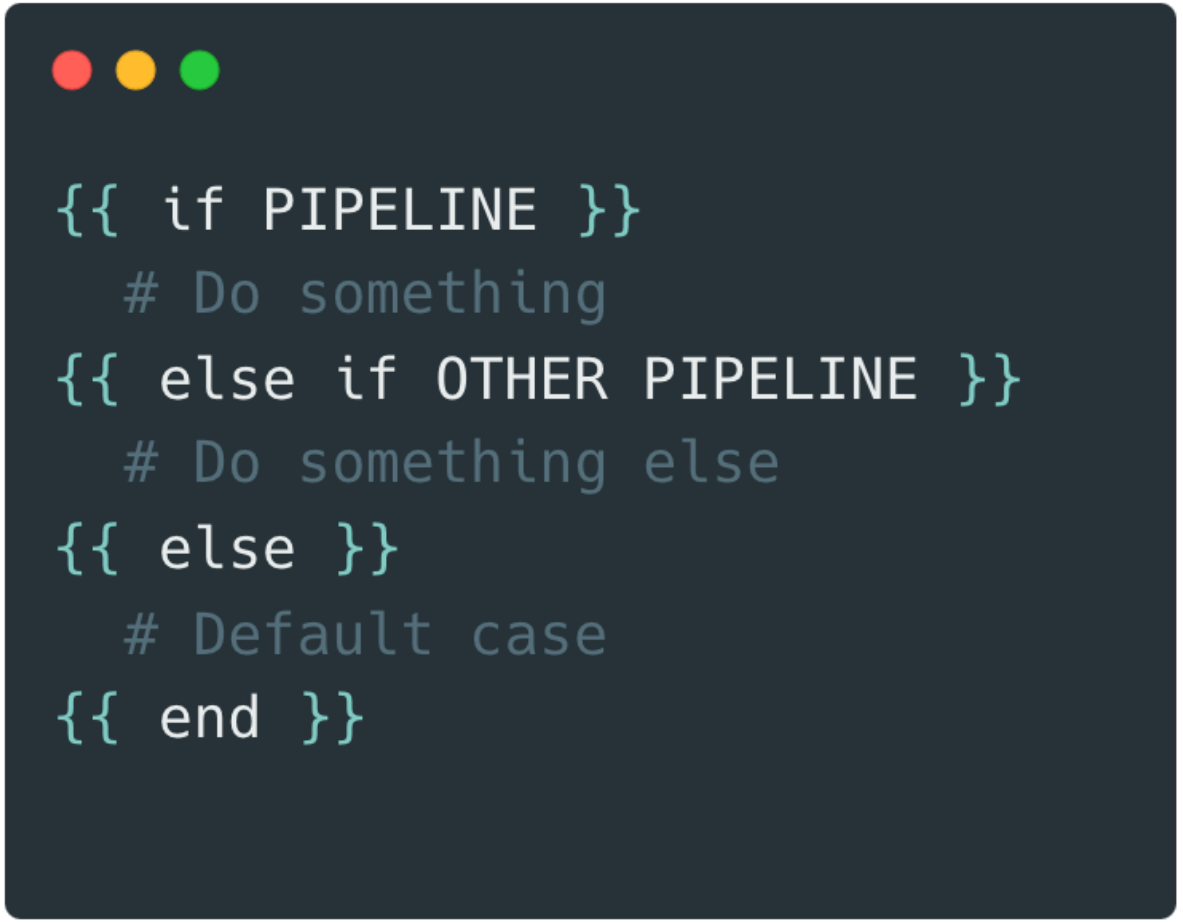
- Pipelines is a great solution because allow you to create a chain of function call
- The input of a function is the output of the previous on the chain

```
drink: {{ .Values.favorite.drink | default "tea" | quote }}
```

- [Helm | Template Function List]

kubernetes

Templates: If/Else



```
{{ if PIPELINE }}
  # Do something
{{ else if OTHER PIPELINE }}
  # Do something else
{{ else }}
  # Default case
{{ end }}
```

kubernetes

Templates: If/Else

kubernetes

Templates: Handle whitespaces

- The templates need to be produced using the YAML indentation in mind
- Not the right indentation level or empty lines should be give errors
- You may use `{{-` and `-}}` to clean whitespaces from left or right (newline is a whitespace)

kubernetes

Templates: Handle Whitespaces

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favorite.drink | default "tea" | quote }}
  food: {{ .Values.favorite.food | upper | quote }}
  {{- if eq .Values.favorite.drink "coffee" }}
  mug: "true"
  {{- end }}
```

kubernetes

Templates: With (scope)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  {{- with .Values.favorite }}
  drink: {{ .drink | default "tea" | quote }}
  food: {{ .food | upper | quote }}
  {{- end }}
```

kubernetes

Templates: With (scope)

```
{{- with .Values.resources }}
resources:
  {{- toYaml . | nindent 12 }}
{{- end }}
```

kubernetes

Templates: Named Templates

- Helm chart create a file called `_helpers.tpl` that allow you to create named templates
- Those are helpful for you to reuse in several templates
- Follow all rules and format defined before

kubernetes

Templates: Named Template

```
{{- define "mychart.labels" }}
  labels:
    generator: helm
    date: {{ now | htmlDate }}
{{- end }}
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
  {{- template "mychart.labels" }}
data:
  myvalue: "Hello World"
```

kubernetes

Templates: Debugging

- `helm lint`: For verifying that your chart follows best practices
- `helm template --debug`: will test rendering chart templates locally.
- `helm install --dry-run --debug`: will also render your chart locally without installing it, but will also check if conflicting resources are already running on the cluster.
- `helm get manifest`: This is a good way to see what templates are installed on the server.

Demo | Helm Charts

Questions?

kubernetes