Open in app ↗ Get unlimited access

Oct 25, 2022 · 4 min read · Listen

Save

Helm—Named Templates

A deep dive into partial or subtemplates

Photo by Borderpolar Photographer on Unsplash

Overview

We can create partial or sub-templates in helm, mainly known as "named templates". A **named template** is simply a template defined inside a file and given a name.

We can consider the named templates just like functions. Named templates will allow us to reuse syntax or logic throughout the helm chart.

Typically the files under the t [illegible] a helm chart contain templates that create Kubernetes manifests. But files whose name begins with an **underscore (_)** are assumed to not have a manifest inside. These files are not rendered as Kubernetes object definitions but are available everywhere within other chart templates for use.

So, the file naming convention will be like this:

```
_filename.tpl
e.g.
_helpers.tpl
```

.tpl extension is widely used as the file intends to contain only templates.

Declaring and embedding "named templates" into other templates :

We can define a named template inside the template file called **_helpers.tpl** created earlier. To define a template we have to use `define` keyword. Since template names are global, there is a chance of conflicts if two templates are declared with the same name. Therefore, maintaining a naming convention will be a sensible idea to avoid duplications. A popular naming convention is to prefix each defined template with the name of the chart, For instance — `<CHART_NAME>.<things_the_template_will_do>`

```
{{- define "webserver.selectorLabels" -}}
# body of template here
{{- end }}
```

`define` functions should have a simple documentation block (`{{/* ... */}}`) describing what they do :

```
{{/*
Selector labels
*/}}
{{- define "webserver.selectorLabels" -}}
```

```
# body of the template here
{{- end }}
```

To embed a named template inside a normal template we can use either **`template`** action or **`include`** function.

Template Action

Let's define a named template and see how we can use it inside a normal template file.

```
# filename:  _helpers.tpl

{{/*
Common labels
*/}}
{{- define "webserver.labels" -}}
    app: nginx
    generator: helm
{{- end }}
```

Embed the **`webserver.labels`** named template inside a normal template file. (Only the relevant YAML is shown)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webserver-deployment
  labels:
  {{- template "webserver.labels" }}
```

Now, if we modify the **`webserver.labels`** named template a little bit, we will find out a different aspect :

```
# filename:  _helpers.tpl

{{/*
Common labels
*/}}
{{- define "webserver.labels" -}}
    app: nginx
    generator: {{ .Release.Service }}
{{- end }}
```

After modification, if we embed the `webserver.labels` like we did before :

```
{{- template "webserver.labels" }}
```

Unfortunately, it will not work. We have to pass a scope while calling the named template. Because we have used `{{ .Release.Service }}` object inside the `webserver.labels` named template. As we didn't pass any scope, within the named template we cannot access anything in `.` . In other words, we will not be able to access anything outside of the current scope. It is not really hard to solve this issue. We just need to pass a scope to the template:

```
{{- template "webserver.labels" . }}
```

Drawbacks :
Along with the `template` action, it is not allowed to use pipelines.

```
# It will not work
{{- template "webserver.labels" . | nindent 4 }}
```

To use pipelines while embedding named templates into a normal template we have to use `include` function.

```
{{- include "webserver.labels" . | nindent 4 }}
```

Include Function

To embed named templates into normal templates using `include` function we must pass two parameters.

> 1. The name of the named template.
> 2. The object scope.

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

And as we discussed earlier, we can use pipelines along with `include` function.

```
{{- include "webserver.labels" . | nindent 4 | qoute }}
```

It is also possible to embed a named template into another named template:

```
{{/*
Common labels
*/}}
{{- define "webserver.labels" }}
{{- include "webserver.selectorLabels" . }}
# body of template here
{{- end }}

{{/*
Selector labels
*/}}
{{- define "webserver.selectorLabels" }}
# body of template here
{{- end }}
```

Walkthrough :

Let's write two named templates and use them inside a deployment template that will generate Kubernetes manifests.

Following is the snippet of the **_helpers.tpl** file:

```
{{/*
Common labels
*/}}
{{- define "webserver.labels" -}}
{{- include "webserver.selectorLabels" . }}
app.kubernetes.io/managed-by: {{ .Release.Service }}
{{- end }}

{{/*
Selector labels
*/}}
{{- define "webserver.selectorLabels" -}}
```

```
app: {{ .Chart.Name }}
{{- end }}
```

We will embed the above-shown two named templates into a deployment template file, Following is the snippet of the **deployment.yaml** file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Chart.Name }}
  labels:
  {{- include "webserver.labels" . | nindent 4 }}
spec:
  replicas: 3
  selector:
    matchLabels:
    {{- include "webserver.selectorLabels" . | nindent 6 }}
  template:
    metadata:
      labels:
      {{- include "webserver.selectorLabels" . | nindent 8 }}
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: nginx:latest
        ports:
        - containerPort: 80
```

Now, Let's generate the template using the **helm template** command :

```
>> helm template ~/webserver

---------------------------------------------

# Source: webserver/templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webserver
  labels:
    app: webserver
    app.kubernetes.io/managed-by: Helm
spec:
  replicas: 3
  selector:
    matchLabels:
      app: webserver
  template:
    metadata:
      labels:
        app: webserver
    spec:
      containers:
      - name: webserver
        image: nginx:latest
        ports:
        - containerPort: 80
```

With that, we have successfully generated Kubernetes manifests.

Closure

Named templates can make it easier to maintain configurations that we want to share across two or more resources, and they help us to create a centralized place to edit common configurations.

> If you found this article helpful, please **don't forget** to hit the **Clap** and **Follow** buttons to help me write more articles like this.
> Thank You 🖤

All Articles on Helm Chart -

Helm — In Action
Helm — Advanced Commands
Helm — Create Your First Chart
Helm — Template Actions, Functions, and Pipelines
Helm — Flow Control
Helm — Variables
Helm — Named Template
Helm — Dependencies

References

Named Templates

It is time to move beyond one template, and begin to create others. In this section, we will see how to define a named...

helm. sh

Helm Helm Chart Kubernetes Kubernetes Cluster Package Management