Published in CodeX

Md Shamim Follow
Oct 14, 2022 · 3 min read · Listen

Helm — Variables

Overview of Helm Templates Variables

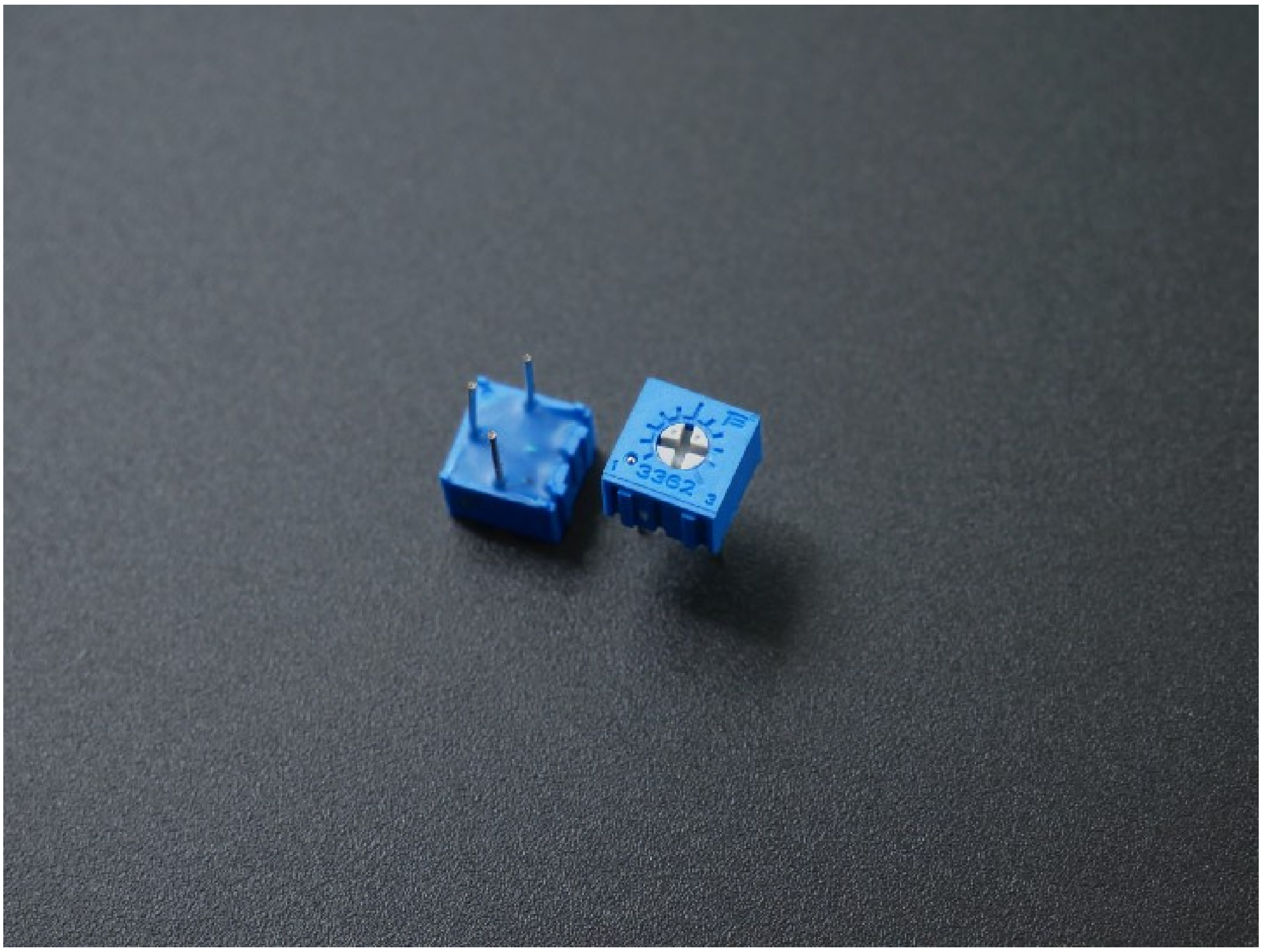
Photo by Vishnu Mohanan on Unsplash

In **helm** templates, variables are less frequently used. But we will see how to use them to simplify code and make better use of `with` and `range` .

To learn in-depth about `with` and `range` read this article: **Helm — Flow Control**

Variable initialization:

```
{{- $namespace := .Release.Namespace -}}
```

Usage of variables within "with":

In the Helm — Flow Control article, we have already seen that the following code will throw an error :

```
{{- with .Values.configMap.data.conf }}
  operating-system: {{ .os }}
  database-name: {{ .database }}
  k8s-namespace: {{ .Release.Namespace }}
{{- end }}
```

`Release.Namespace` is not inside of the scope that's restricted in the `with` block. One way to solve this issue is to use the $ sign in front of the Release object. Because root scope is also represented by the $ sign. If we change our code to `$.Release.Namespace` then the issue will be resolved.

Alternatively, it is also possible to solve the above-mentioned issue using a **variable.**

```
{{- $namespace := .Release.Namespace -}}
{{- with .Values.configMap.data.conf }}
  operating-system: {{ .os }}
  database-name: {{ .database }}
  namespace: {{ $namespace }}
{{- end }}
```

In the above demonstration, before defining the `with` block, we assigned `$namespace := .Release.Namespace`. And then, inside the `with` block we used the `$namespace` variable because it still points to the release namespace.

Usage of variables within "range":

Variables are particularly useful in `range` loops. They can be used on list-like objects to capture both the **index** and the **value**:

Suppose, we have **values.yaml** file containing the following entries :

```
# values.yaml

configMap:
  data:
    platfrom:
     - java
     - python
     - golang
```

And we want to generate a **configmap** manifest file like this:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: release-name-configmap
data:
  0 : "java"
  1 : "python"
  2 : "golang"
```

To generate the manifest file shown above, we can use `range` and **variables** together to write a **configmap.yaml** template file :

```
# configmap.yaml

apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  {{- range $index, $value := .Values.configMap.data.platfrom }}
  {{ $index }} : {{ $value | quote }}
  {{- end }}
```

For data structures that have both a **key** and a **value**, we can use `range` to get both.

Suppose, we have to write a template file for the Kubernetes **Secrets** object. And currently, we have the following entries in the **values.yaml** file :

```
# values.yaml

secrets:
  db:
    MYSQL_USER: admin
    MYSQL_PASSWORD: Admin@123
```

In Kubernetes **Secrets**, values for all keys in the `data` field must be **base64-encoded** strings. To learn in-depth about Kubernetes Secrets read this article: **Secrets**

We want to generate a Kubernetes Secrets manifest file like this :

```
apiVersion: v1
kind: Secret
metadata:
  creationTimestamp: "2022-10-12T04:11:18Z"
  name: mysql-secret
  namespace: default
  resourceVersion: "11349"
  uid: 85dec499-2744-4062-9deb-1e9f1dd71fb6
type: Opaque
data:
  MYSQL_PASSWORD : QWRtaW5AMTIz
  MYSQL_USER : YWRtaW4=
```

To generate the manifest file shown above, we can create a template file in the following way :

```
# secretes.yaml

apiVersion: v1
kind: Secret
metadata:
  creationTimestamp: "2022-10-12T04:11:18Z"
  name: mysql-secret
  namespace: default
  resourceVersion: "11349"
  uid: 85dec499-2744-4062-9deb-1e9f1dd71fb6
type: Opaque
data:
  {{- range $key, $value := .Values.secrets.db }}
  {{ $key }} : {{ $value | b64enc }}
  {{- end }}
```

Notice that, we have used two variables `$key` and `$value` to get both key and value from `.Values.secretes.db` object residing in the **values.yaml** file. In addition to that, we have used `b64enc` function to convert the plaintext to a **base64 encoded** format.

> 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 🖤

References

Variables

With functions, pipelines, objects, and control structures under our belts, we can turn to one of the more basic ideas…

helm.sh

Kubernetes Helm Helm Chart K 8 S Variables