2.1. Create Helm Chart – Añadir valores a los template

Supongamos que el template por defecto no es suficiente para satisfacer nuestras necesidades. Podemos modificar y/o añadir datos pertenecientes a los archivos de la carpeta **template** y hacerlos dinámicos para que obtengan su valor del archivo **values**.**yaml**.

Una cosa que hay que tener en cuenta es que los archivos de la carpeta **template** están escritos en YAML, pero se les añade el motor de plantilla de Go. Esto agrega la capacidad de llamar variables de otros archivos (Chart, Values, Release). La sintaxis que sigue Go para este cometido es:

* {{ **.Object.Parameter** }}

Los valores que pueden sustituir el valor de Object son:

* Chart: para acceder a los valores almacenados en el archivo Chart.yaml



* Values: nos permite acceder a los valores almacenados den el archivo value.yaml

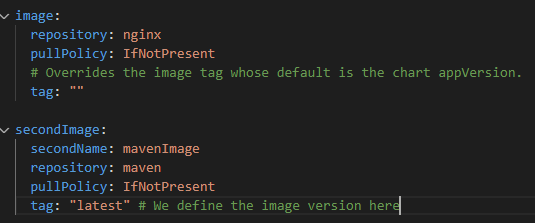


* Release: nos permite acceder a los datos del release, estos datos son creados cuando se está ejecutando la instalación del chart.



Vamos a ver un ejemplo sencillo de esto, supongamos que necesitamos crear una segunda imagen. Por defecto, en el **deployment** solo existen los datos correspondientes a un solo contenedor. Vamos a crear los valores para configuración dinámica y crear el contenedor.

* **values.yaml**



* **deployment.yaml**



De esta forma agregamos los datos básicos para una segunda imagen. Aquí vemos algo curioso y es el uso del operador “**|**” en la propiedad de **image**.



* Esto es lo que se llama Pipe y nos permite realizar una serie de acciones (lado derecho) a los datos del lado izquierdo. En este caso, se utiliza para crear un valor por defecto si el valor de **tag** es vacío.
  + Esta acción a la derecha define un valor por defecto y este será el valor correspondiente a la propiedad **appVersion** dentro de Chart.



**INCLUDE**

En algunos casos, dentro de nuestros archivos de **template**, podemos encontrar variables que tienen el mismo nombre con el que creamos el **chart**. En nuestro caso, el chart se llama springBoot y en algunas partes sale la siguiente expresión:



* Vamos a dividir esto.
  + **include**: nos permite añadir valores de archivos externos, en este caso estamos añadiendo valores del archivo **\_helpers**.**tpl**.
    - Este posee una serie de funciones que ayudan a la elaboración del template.
  + **springBoot**.**selectorLabels**: esto apunta a la función **selectorLabels** dentro del archivo **helpers**.
  + **nindent 6:** esta pequeña función nos permite crear la indentación de los valores que se van a añadir. En este caso, dejar 6 espacios de margen.
  + **.**: el punto hace referencia a traer todos los valores que devuelve dicha función.

**CONDICIONAL**

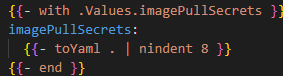
En otros casos, podemos ver la existencia de una condicional:



* Esto permite agregar o quitar propiedades si se logra cumplir algún parámetro en específico.
* En el ejemplo anterior, podemos ver
  + la sentencia **if**
  + seguida por una negación (**not**), si no queremos negar no debemos poner esto,
  + terminando con un valor tomado desde el archivo **Values**
* Para cerrar la sentencia **if** debemos utilizar un **end**. Todo esto dentro de dos llaves {{}} para permitir que Go interprete las sentencias escritas.

**With**

Esto nos permite utilizar datos cargados desde archivos externos y modificarlos a nuestro gusto dentro de los límites de la sentencia:



* En este caso traemos del objeto **Values**, los datos correspondientes a **imagePullSecrets**.
  + Estos valores son añadidos en forma de arreglo, por lo que no está acorde con la estructura de los archivos YAML. En estos casos, no es posible asignar su valor directamente.
* Una vez abierto el bloque correspondiente al with, dentro podremos darles estructura a los datos. En este caso, estamos convirtiendo el arreglo a una estructura basada en el YAML. Además, se está dando un margen de 8 espacios.