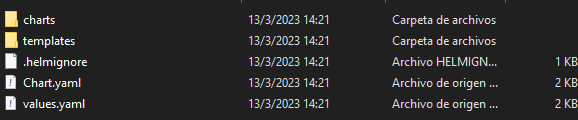
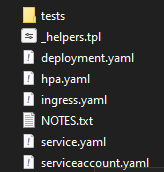
2. Create Helm Chart

En este tutorial vamos a ver como crear paso a paso un Chart de Helm para encapsular nuestra aplicación. Anteriormente, en la introducción, vimos que es HELM, las ventajas que este tiene y la forma de crear un chart genérico. Este tutorial, retoma desde ese momento, desde que se creó el chart.

1. Como en el tutorial 0.6.2, creamos un chart genérico de HELM.
   1. helm create NOMBRE\_DEL\_CHART



1. Todas las configuraciones que realicemos para definir las propiedades de los componentes de **kubernetes** se encuentran en la carpeta de **templates**.



1. Fuera, podemos encontrar dos archivos, **Chart** y **values**.
   1. **Chart** provee los datos como nombre y versión del **chart** que se va a utilizar.
      1. Aquí es posible agregar otros datos como “**maintainers**” para definir las personas que están trabajando sobre este archivo.



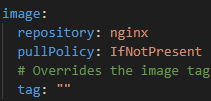
* 1. **Values** aquí dentro van todas las configuraciones que pueden ser definidas cuando se carga el chart; por ejemplo, cambiar el valor de una variable en específico.
     1. --set service.internalPort=8080

1. Vamos a trabajar en un principio, si no es necesario mucha manipulación de datos, en el archivo **values.yaml**. Esto es debido a que, los archivos dentro de la carpeta **template** ya están configurados para el despliegue de una aplicación. Esto es posible gracias a que están escritos con el motor de plantillas de GO; lo que ayuda a crear condicionales y ajustar dichos archivos a los valores declarados en **values.yaml**.
   1. Ojo, esto es solo para el caso de aplicaciones que no lleven mucha manipulación. En caso que la estructura base par aun archivo de **deployment**, por ejemplo, nos sea insuficiente, tenemos que modificar **deployment.yaml**.
2. Lo primero que vemos cuando entramos al archivo “**values.yaml**” es la propiedad de réplicas:



* 1. Aquí podemos definir cuantas réplicas iniciales queremos de nuestra aplicación; por defecto aparecerá en 1, pero podemos cambiar el valor a nuestro gusto.

1. Seguido, tenemos la información perteneciente a la imagen que vamos a utilizar:



* 1. **repository**: aquí definimos la imagen que vamos a descargar. Si queremos usar la imagen de nuestra aplicación, tenemos que tenerla subida a algún repositorio domo **DockerHub** por ejemplo. En caso de tener un repositorio propio, se puede cambiar esta propiedad en **Kubernetes**.
  2. **pullPolicy**: aquí definimos la forma en que se descargará la imagen; esta opción puede ser llenada con los siguientes valores:
     1. **IfNotPresent**: descarga una versión nueva de la imagen si no se encuentra en el **cluster**.
     2. **Always**: cada vez que se reinicie el **deployment**, intentará descargar una nueva versión de la imagen.
     3. **Latest**: descarga la versión más actualizada de la imagen.

1. Los chart ya vienen con un nombre predefinido, pero si queremos sobre escribir este nombre sin cambiar su valor dentro del archivo “**chart.yaml**” tenemos las dos siguiente propiedades:

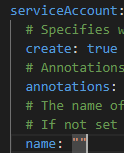


* 1. **nameOverride**: sobre escribe el nombre de la aplicación. Por ejemplo, dentro del archivo **chart**.**yaml** tenemos:



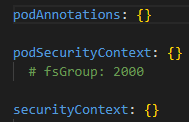
* + 1. Si llenamos el valor **nameOverride**, cuando la aplicación se despliegue su nombre no será **springBoot**, sino el que definamos en esta variable.
  1. **fullnameOverride**: es parecido a lo anterior, pero **Kubernetes** tiene nombres cortos y largos de sus componentes, por lo que, si no se llena nada en este campo, se tomará el valor de **name** o **nameOverride**.

1. Posteriormente tenemos el apartado de **service account**, aquí podemos definir los datos del creador del archivo de HELM o mejor dicho, asociar el **deployment** que estemos creando a un usuario:



* 1. Para que funcione, debemos tener el valor **create** en **true** (verdadero). En caso que sea **false** (falso), no se tomará en cuenta para el despliegue.
  2. **annotations**: nos permite agregar información extra de tipo clave valor.
  3. **name**: esta información no es necesaria, ya que, si está vacía, helm generará su valor en dependencia de la variable fullname; aunque es una buena práctica poner su valor manual.

1. Los siguientes tres elementos nos permiten:



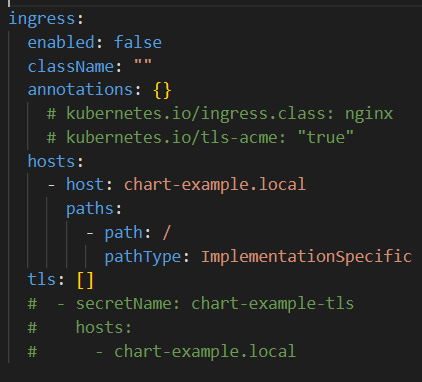
* 1. **podAnnotations**: añadir anotaciones del tipo clave valor a nuestro POD.
  2. **pdSecurityContext**: nos permite definir los niveles de seguridad que tendrá nuestro POD.
  3. **securityContext**: similar al anterior, pero este tiene un alcance más global.

1. Otro de los apartados más importantes del archivo **values**.**yaml** es el correspondiente al **service** que se va a crear junto al **deployment**:



* 1. Aquí podemos definir el tipo de servicio que queremos crear:
     1. Los tipos y sus diferentes los vimos en el tutorial extra, Diferentes entre ClusterIP, LoadBalance y NodePort.
  2. El puerto hace referencia al puerto de acceso del PC al **Kubernets**.

1. Posteriormente tenemos la configuración del ingress:



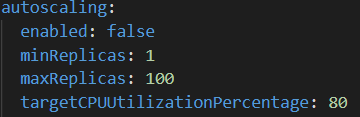
* 1. Esto nos permite configurar la dirección hacia un **ingress** si lo tenemos creado dentro del cluster.

1. Tal vez uno de los apartados más importantes si tenemos pocos recursos **Hardware**:



* 1. Dentro de este atributo podemos definir todos los recursos que va a poseer nuestro **POD**. Entre ellos está el limite de memoria o CPU. Incluso se puede añadir peticiones de recursos hardware; de esta forma, si se liberan recursos de otro **POD**, se le puede asignar a este.

1. Si queremos que nuestro sistema se escale de forma automática cuando se cumpla cierto criterio, podemos utilizar las siguientes propiedades:



* 1. Estas son útiles por si no queremos tener un número de replicas fijo. Es importante tener en cuenta que solo debemos usar esta opción si los recursos hardware del **cluster** son suficientes para aguantar un escalado automático.
  2. Por defecto está deshabilitado, por lo que tenemos que cambiar el valor del atributo **enabled**.
  3. Posteriormente definimos la menor cantidad de réplicas y lo máximo que pueden alcanzar.
  4. Seguido podemos comenzar a declarar las condiciones para el escalado:
     1. En este caso se escala cuando el uso de la **CPU** de las réplicas existentes llegue a un 80%.
     2. Ojo, es importante saber que no se va a escalar directamente de 1 a 100. Cada vez que un **POD** llegue a el uso del 80%, se creará uno nuevo. En caso que disminuya el uso de la **CPU**, se eliminará el **POD**.
     3. Otra condición que podemos poner es:
        1. **targetMemoryUtilizationPercentage**
     4. Esta apunta al uso de la memoria.