YML de Kubernetes – Deployment

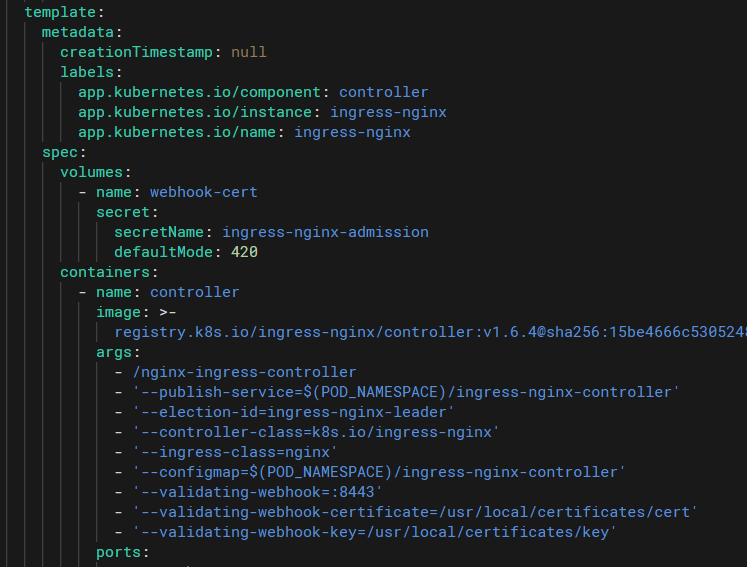
Los componentes de tipo **deployment** le especifican a **kubernetes** como crear o actualizar los PODs. Los despliegues permiten lograr un escalado eficiente de los PODs, ya que evitan que existan replicas con datos o propiedades que no cumplan con las características descritas en el archivo YAML.

Kubernetes posee una jerarquía de trabajo para los **deployment**. Se puede decir que los **deployments** son los componentes generales y son los encargados de administrar directamente los **ReplicaSet**. Siguiendo la jerarquía, los **ReplicaSet** se encargan de administrar los **PODs**; esto es debido a que los **ReplicaSet** se encargan de mantener y administrar la replicación de los **PODs** creados. Para ver esto de forma visual, podemos auxiliarnos del software **Octant**:



Estructura de un YAML de kubernetes:

* Todos los archivos de kuberentes están compuestos de 3 partes:
  + Metadata: información sobre el componente.
  + Specification: configuración del componente.
    - Los atributos que utilicemos en esta zona son específicos del “**kind**” que seleccionemos.
  + Información del YAML:
    - apiVersion: versión del api que vamos a utilizar.
      * apps/v1
    - kind: tipo de archivo que estamos creando.
      * Service, Deployment, etc.
  + Como extra tenemos los “**State**” o estados.
    - Los estados de kubernentes se basan en la configuración que creemos en el archivo YAML. En caso que el componente actual, no tenga una configuración idéntica al archivo YAML que lo originó; kubernetes sabe que existe un problema.
    - Los valores de este apartado son creados directamente por kubernetes.
* Dentro del apartado de especificaciones (spec) podemos encontrar una sub propiedad llamada template:

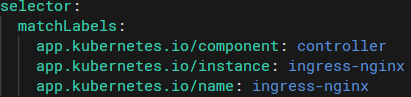


* + El “**template**” es una configuración que se le aplica directamente a los PODS. Debido a esto, también tiene un apartado de “**metadata**” (información general) y “**spec**” (especificaciones).
    - Específicamente, la sección de las especificaciones, definirá la estructura de trabajo del POD.
  + La propiedad “**selector**” o selector, nos permite definir un nombre para identificar un “**deployment**”. De esta forma, podemos decirle posteriormente a un servicio, a que selector se le debe aplicar la configuración.
    - No debemos confundir los **labels**, presentes en los metadatos, con el “**selector**”.
      * Los **labels** son una pareja de datos clave-valor, que nos permiten definir datos generales de nuestra aplicación (entre ellos los que utilizaremos como selectores)
        + Son datos propios del componente que estemos creando.
      * El selector, le permiten a kubernetes seleccionar un POD creado, lo que quiere decir, que los selectores que utilicemos deben ser iguales a un “**label**” de los metadatos del apartado de “**template**”.
        + O sea, si el label tenemos un par clave – valor (app:nginx), en matchLabels, dentro de selector tenemos que tener el mismo valor si usamos la misma clave.

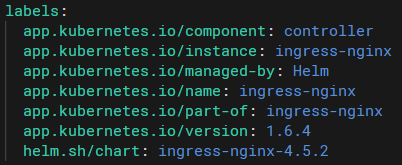
**Template** (Aquí se encuentra la configuración del POD)



**Selector (**los mismos **label** del **template)**:



En contraste con el **label** dentro del **metadata** (los que se encuentran al principio del archivo YAML):



* En este ejemplo podemos ver que que los **labels** correspondientes al **template** son idénticos a los que se encuentran en **matchLabels**. En cambio, los metadatos globales poseen mayor cantidad de atributos; aunque los más importantes se repiten en las tres secciones.

Ya vimos un ejemplo de mayor cantidad de atributos en los metadatos generales que en las otras dos secciones. Para evitar encasillarnos, pondremos un contra ejemplo, donde los **labels** del **template** poseen más atributos que las otras dos secciones.

**Global Metadata**



**Spec matchlabels**

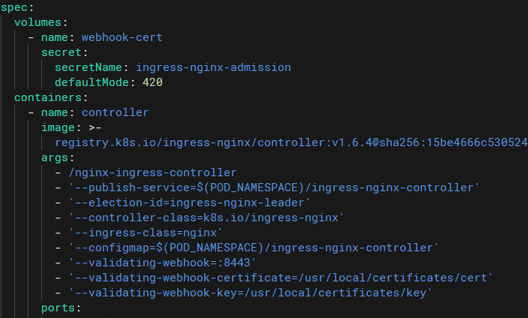


**Template Labels**



En general, podemos ver dos cosas. La primera es que para la sección de **matchLabels**, las etiquetas que se utilizan poseen como palabra clave **app** o una unión de esta con otro termino **app.kubernetes**. No importa la cantidad de **labels** que se definan en los metadatos globales o del **template**; es una buena práctica repetir los **lables** que se van a utilizar en **matchLabels** en ambas secciones.

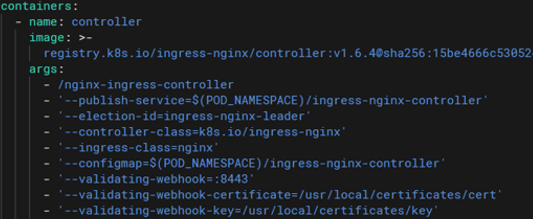
Después de saber cómo funcionan los metadatos, los **labels** y los selectores; es posible pasar a ver cómo funcionan las especificaciones dentro de los **pods**. Las especificaciones hacen referencia a los contenedores que serán desplegados dentro de los **PODS**. Las especificaciones son definidas debajo de la etiqueta **spec** que está dentro de **template**:



* Como primer atributo dentro de las especificaciones se encuentra **volumes**. Esta hace referencia a los volúmenes que serán utilizados dentro de los contenedores. En este caso en específico se está creando un volumen para permitir el uso de un **secret** dentro del contenedor; más adelante se abordará el tema de los **secrets** y los **configMap**:



* Seguido y probablemente la especificación con mayor importancia dentro del archivo YAML de tipo **Deployment** es el contenedor. Este permite la especificación del contenedor de Docker que será utilizado dentro de la aplicación. En este caso se define el contenedor del **ingress** de **nginx**:



* + Internamente las propiedades de estos contenedores hacen referencia a las propiedades de la imagen de **Docker** a utilizar. Mírenlo como una especie de **Docker Compose**; el cual define imagen, argumentos, puertos y volúmenes para nuestra aplicación.
  + El primer atributo es **name**, como su nombre lo indica, hace referencia al nombre que le pondremos al contenedor.
  + **Image** hace referencia a la imagen que se descargará de **Docker Hub** u otro repositorio de imágenes de **Docker**.
  + **Args** permite definir los diferentes argumentos con los que se puede inicializar una aplicación de **Nginx**.
  + Posteriormente se podrán encontrar los **ports**, los cuales definen los puertos por los que se puede acceder al contenedor. Estos deben ser aquellos que son expuestos el archivo Docker.



Buenas prácticas con los archivos YAML

* Mantener bien identados las propiedades, ya que es estricto con la identación.
  + En caso de tener un archivo YAML demasiado largo, es posible utilizar sitios en internet para comprobar la valides de los mismos.
* Almacenar los archivos YAML junto al código que estemos creando o a un repositorio.