Helm

¿Qué es helm?

Se trata de un manejador de paquetes para kubernetes. Permite buscar, modificar e instalar paquetes en el cluster kubernetes. es el estándar para la distribución de apps en kubernetes.
Posee 3 componentes:

- **Helm Chart**: receta que se utiliza para instalar paquetes en el cluster kubernetes.
- **Helm Repository**: dónde se guardan los helm charts.
- **Helm Releases**: versiones de los paquetes que instalamos. Podemos tener varias copias de un mismo paquete con diferentes releases. No implica que las versiones de los paquetes son diferentes, sino que hacen cosas distintas.

casos de uso

- Instalación de paquetes de 3ros con un solo comando
- modificación de paquetes personales
- guardado de paquetes personales en el repo de helm
- rollaback a una versión de configuración anterior
- acceso a las releases de nuestra propia app guardadas en el repo de helm
- despliegue de pila de aplicación a cluster de kubernetes

ventajas

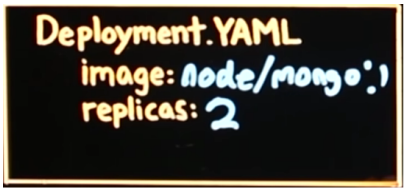
- cientos de paquetes disponibles para instalar
- personalizable a las especificaciones de nuestro cluster o caso de uso
- proveedores originales de software suman sus paquetes de buena gana en el repositorio de helm.
- como consecuencia de punto anterior, los paquetes del repositorio de helm estarán actualizados a la última versión.
- helm permite gestionar nuestra propia aplicación, por lo que no se limita a la instalación de paquetes de 3eros. Permite incluir nuestras propias dependencias (servidores web, bases de datos, etc) en un solo paquete, de manera que cuando se instalen en otro cluster o entorno se llevará todas las dependencias.
- Es personalizable en cada entorno, por lo que puede tener parámetros distintos cuando se levanta en el servidor local o cuando se levanta en un entorno de desarrollo.
- su gestión de actualizaciones incluye todos los despliegues automáticos y almacena todas las versiones de nuestra app en su base de datos.

ejemplo de uso

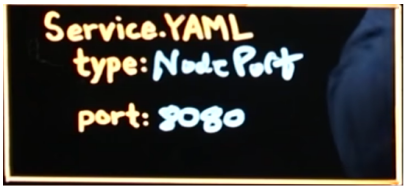
Una aplicación de comercio en línea desea registrar usuarios para una plataforma de compras para las fiestas de fin de año; hemos creado una aplicación en node js que vamos a desplegar en el cluster de kubernetes. Tenemos 2 réplicas de la aplicación para poder manejar todas las peticiones, una base de datos Mongo DB y hemos desarrollado servicios como un ***Node Port*** para que accesen nuestra aplicación.

Para poder implementar este tipo de pila de aplicación tenemos que definirla, una manera de hacerlo es escribiendo algunos ***archivos tipo YAML***.

En nuestro caso sabemos que se debe implementar una imagen de Node JS y MONGO DB; además necesitaremos 2 réplicas para poder alcanzar alta disponibilidad; esto se describe en los archivos **Deployment.YAML**.



Como decidimos previamente que utilizaríamos un tipo de ***Node Port***, entonces describimos esta característica en los archivos **Service.YAML**; dentro de estos archivos también se debe definir el puerto a utilizar.

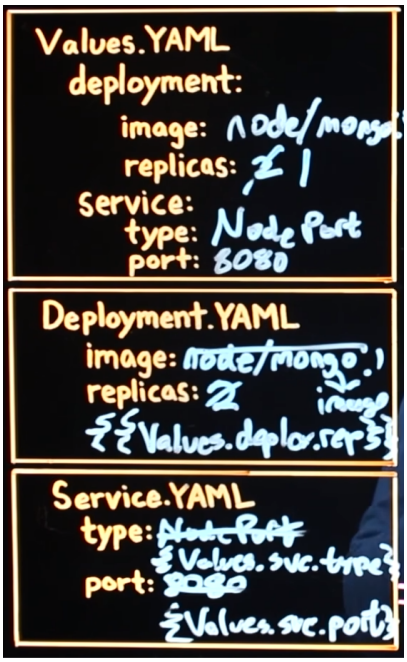


Hasta aquí no hay ningún problema para reconocer las características de nuestra app. Sin embargo si otra persona, ajena al desarrollo inicial de la app, decide cambiar el número de réplicas, puede que se le dificulte encontrar el archivo para realizar el cambio. Por ello existe una forma más sencilla de manejar la configuración de toda la pila de aplicaciones y separarla lógicamente de todo lo que tenemos con nuestra aplicación con plantilla particular.

Aquí se introduce el concepto de chart, ahora no solamente podemos usar helm para manejar la configuración a través de YAML files, sino que podemos crear una plantilla de pila de aplicación mediante ***helm charts***. Un chart consiste en todos los archivos que conforman nuestra plantilla.

¿Como injectamos variables dentro de estos archivos?

Tomamos nuestra configuración y usamos lenguaje de plantillas para decidir dónde esos valores estarán puestos en nuestra plantilla. **Es basicamente usar variables para definir parámetros de la configuración de nuestra app para que los cambios futuros se puedan hacer fácilmente**. Las varibles estarán definidas en código duro solamente en el archivo **Values.YAML** al que los demás harán referencia para accesar a sus parámetros.



¿Cómo se combina la pila de aplicación creada con .YAML a nuestro cluster de kubernetes?

El comando: "helm install myAPP" permite que helm recorra el ***Templating chart*** o tabla de plantillas y busque las partes donde hemos definido valores particulares de nuestra configuración. Irá a nuestro archivo **Values.YAML** e inyectará esos parámetros dentro de nuestro archivo de plantilla (**Deployment.YMAL** y **Service.YAML**). Cuando haya abaracado todo enviará el **helm chart** a un componente que debe ser instalado en nuestro cluster de kubernetes llamado ***Tiller***. **Este proceso ejecutado por Helm con ayuda de Tiller para mapear las estructuras del script en el cluster de kubernetes es similar a una labor de un compilador que traduce código de alto nivel a lenguaje máquina**

Todo esto es muy util cuando actualizamos a una nueva configuración y queremos retroceder ***rollback***. En lugar de botar la aplicación y relanzarla simplemente podemos reconfigurarla con el comando: "helm upgrade myApp"

si una configuración no salió de la manera esperada usamos el comando: "helm rollback myApp". Esto debido a que helm guarda un historial de versiones de las configuraciones que hayamos enviado.

por ultimo, usando el comando: "helm package myApp". Podemos guardar en el repositorio de helm nuestra configuración para que otros grupos de trabajo puedan accederla.

conceptos

- **Node Port**: Que hay un relación 1 a 1 de IPs dentro y fuera del clúster de Kubernetes.
- **archivos tipo YAML**: Describen cómo se van a ver las implementaciones (***deployments***), los servicios, etc.

- ***Tiller***: el componente tipo servidor de helm que tomará los comandos enviados con el cliente de helm y los convertirá a algo que el cluster de kubernetes pueda entender.

Bibliografía

- intro a helm
- por qué usar helm
- ejemplo de parametrización de un app