Iniciación a OpenShift Serverless y Knative Serving

Índice

1. Introducción

1.1. Introducción a Serverless

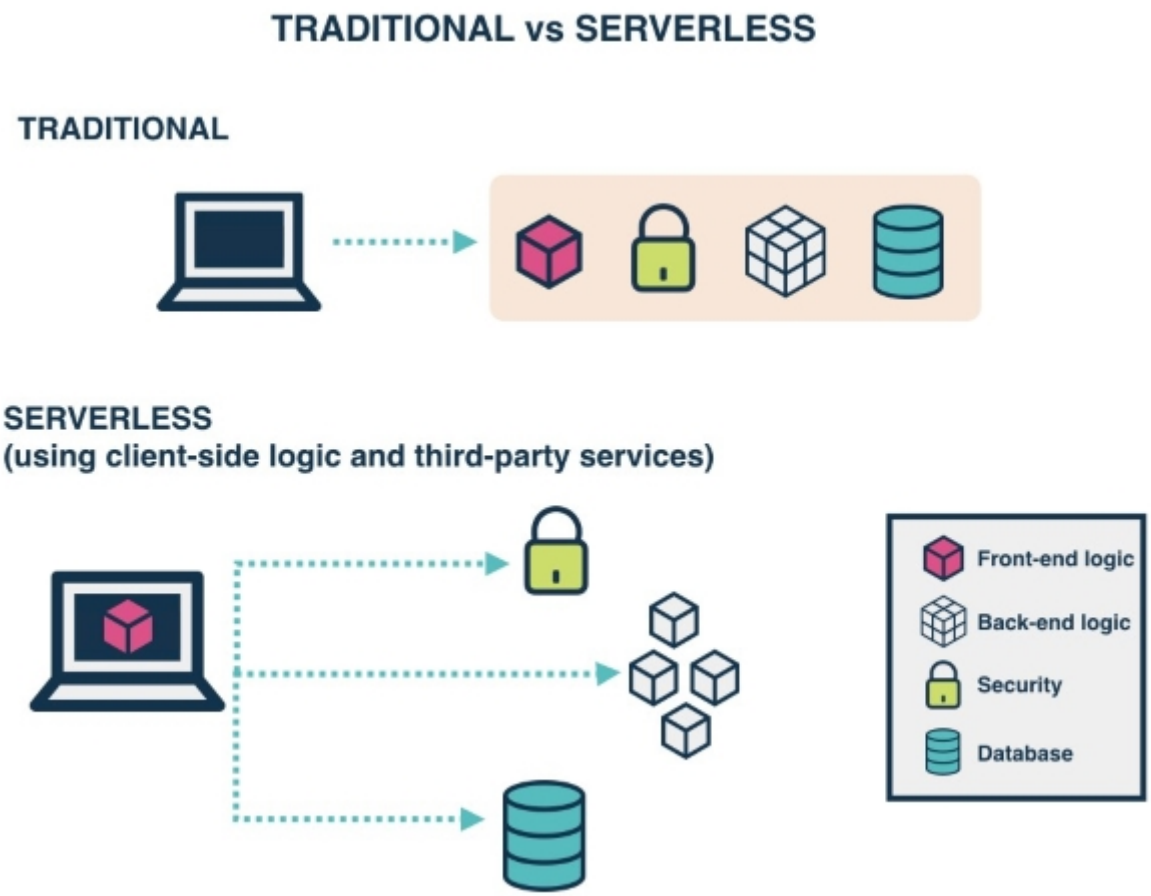
Serverless es un modelo de desarrollo para aplicaciones nativas de la nube, que evita cualquier tarea y requisito de gestión del servidor, permitiendo que los desarrolladores se centren en ofrecer aplicaciones sin preocuparse por la infraestructura en la que se ejecutan.

Estas aplicaciones suelen seguir una arquitectura impulsada por eventos para comunicarse entre aplicaciones, pero no es obligatorio. Serverless es más adecuado para aplicaciones asíncronas y sin estado, especialmente aquellas que tienen cargas de trabajo impredecibles.

¿Qué son las aplicaciones asíncronas y sin estado?

Las aplicaciones asíncronas son aquellas que no esperan a que se complete una tarea para continuar con la siguiente. Por ejemplo, cuando enviamos un correo electrónico, no esperamos a que el destinatario lo lea para enviar otro correo electrónico.

Las aplicaciones sin estado son aquellas que no almacenan información sobre el estado de la aplicación. Por ejemplo, cuando enviamos un correo electrónico, no guardamos información sobre el estado del correo electrónico, como si se ha leído o no.



1.2. Objetivos

Comprender la arquitectura Serverless de OpenShift y las principales características de Knative Serving para aplicaciones Serverless.

Aprender a implementar aplicaciones Serverless nativas de la nube mediante el uso de Knative Serving desde la línea de comandos y la consola web de OpenShift.

Se tratará de una aplicación de ejemplo que se desplegará en OpenShift y que se ejecutará en un entorno Serverless usando Knative Serving. Esta aplicación se desplegará en un entorno de laboratorio de Red Hat y poseerá Back-End para así ver la utilidad de Serverless, su funcionamiento y funcionalidades.

1.3. Público Objetivo

Desarrolladores de aplicaciones nativas de la nube interesados en desarrollar aplicaciones Serverless.

Administradores de OpenShift interesados en tecnologías Serverless para automatizar operaciones y desarrollar herramientas de utilidad para administrar y monitorear sus aplicaciones.

1.4. Requisitos previos

Poseer una cuenta Partner de Red Hat y acceso a Red Hat CloudForms Management Engine para realizar las pruebas de laboratorio.

Tener conocimientos previos sobre:

- Red Hat OpenShift v4 por Daniel Parrales García
- Red Hat OpenShift v4 por José Domingo Muñoz Rodríguez
- Red Hat OpenShift v4 Documentación Oficial
- Contenedores (Podman/Docker)
- Kubernetes

2. Red Hat OpenShift Serverless

OpenShift Serverless es una plataforma que funciona como una capa de abstracción sobre la plataforma de contenedores de Red Hat OpenShift (RHOCP) para facilitar la implementación de aplicaciones de forma estandarizada. Al estar basada en el proyecto de código abierto Knative, proporciona una interfaz uniforme y estrechamente integrada en todo el ecosistema de nube híbrida, incluyendo Edge Computing.

La implementación y actualización de aplicaciones se simplifican gracias a OpenShift Serverless, que también ofrece funcionalidades como interconexión de aplicaciones, enrutamiento de tráfico y escalado automático. Además, se integra con otros productos de Red Hat OpenShift, como el servicio de monitorización de clúster y Service Mesh.

También admite cientos de fuentes de eventos, incluidos mensajes de Kafka a través de Apache Camel K. El Operador oficial proporciona un proceso de instalación rápido y sencillo.

OpenShift Serverless tiene dos componentes principales, cada uno con sus propias responsabilidades distintas:

Serving: Componente encargado del despliegue y la escalabilidad automática de las aplicaciones.

Eventing: Infraestructura para consumir y producir eventos que impulsan las aplicaciones.

OpenShift Serverless se comunica principalmente a través de documentos YAML, los cuales permiten definir de manera declarativa el estado deseado de la aplicación. No obstante, también dispone de una alternativa de interacción imperativa, la cual puede realizarse a través de la herramienta de línea de comandos kn o mediante la consola web de Red Hat OpenShift.

2.1. Ventajas de Serverless

Provisión automatizada

La principal ventaja del serverless es que los servidores son completamente proporcionados por el proveedor serverless, quien se encarga de la provisión, mantenimiento y escalado de los mismos.

También puede llevar a minimizar el uso de recursos, y esto viene con una reducción de costos: ser cobrado solo por el uso de recursos significa que si no se usan recursos, entonces no se cobra.

Enfoque en el desarrollo

Eliminar la necesidad de provisión y mantenimiento del servidor permite a los desarrolladores concentrarse en la lógica de negocios y evitar cualquier fricción durante el despliegue porque el ambiente de desarrollo y producción son iguales.

Escalar a cero

Las infraestructuras serverless se escalan a cero instancias de aplicación cuando no hay tráfico y reaccionan a las solicitudes entrantes poniendo la solicitud en espera mientras se activan las aplicaciones necesarias.

Esto tiene el efecto de que algunas solicitudes pueden experimentar retrasos debido al inicio de los recursos: si la latencia es crucial para la carga de trabajo, entonces los desarrolladores pueden desactivar este comportamiento estableciendo un número mínimo de instancias activas en la aplicación requerida.

Eliminar la planificación de capacidad

Con la infraestructura tradicional, la capacidad de cómputo se reserva y paga por adelantado, mientras que las aplicaciones serverless solo se ejecutan cuando se necesitan. Esto reduce el trabajo de planificación de capacidad para las aplicaciones desplegadas.

2.2. Cuándo utilizar Serverless

Tareas periódicas: tareas pequeñas que necesitan ejecutarse en momentos predefinidos para tareas de mantenimiento.

Procesamiento de IoT: recibir y procesar datos y eventos de dispositivos inteligentes.

Reaccionar a eventos externos: actuar en los cambios de aplicaciones externas.

Back-end de aplicaciones web: lado del servidor de aplicaciones web que pueden dividirse en microservicios.

Procesamiento por lotes: trabajos de procesamiento en segundo plano.

2.3. Cuándo NO utilizar Serverless

No todas las aplicaciones son adecuadas para la arquitectura Serverless, algunos casos de uso lo hacen muy difícil de utilizar.

Algunos ejemplos de esto son:

Baja latencia: los requisitos de latencia extremadamente bajos son difíciles de lograr debido a su naturaleza dinámica.

Pruebas y depuración fáciles: las aplicaciones Serverless se dividen en muchos fragmentos pequeños, por lo que son más difíciles de probar y depurar como un todo.

Observabilidad: al igual que con las pruebas y depuración, una aplicación dividida en muchas partes es más difícil de rastrear y monitorear.

2.4. Instalación del Operador OpenShift Serverless

La instalación del Operador OpenShift Serverless nos permite instalar y usar Knative Serving, Knative Eventing y Knative Kafka en un clúster de plataforma de contenedores OpenShift. El Operador OpenShift Serverless administra las definiciones de recursos personalizados (CRDs) de Knative para nuestro clúster y nos permite configurarlos sin modificar directamente los mapas de configuración individuales para cada componente.

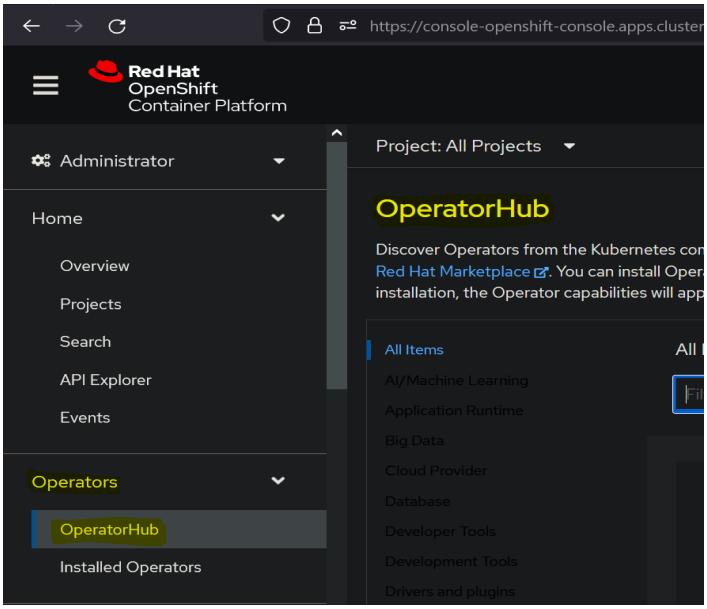
Desde la Consola Web:

Prerrequisitos

- Tener acceso a una cuenta de OpenShift Container Platform con acceso de administrador de clúster.
- Haber iniciado sesión en la consola web de la plataforma de contenedores OpenShift.

Instalación

1. En la consola web de la plataforma de contenedores OpenShift, vamos a la página Operadores → OperatorHub.



2. Nos desplazamos, o escribimos la palabra clave "Serverless" en el cuadro Filtrar por palabra clave para encontrar el Operador OpenShift Serverless.
3. Haremos clic en instalar, aunque antes podremos revisar la información se proporciona sobre el Operador.
4. A la hora de instalar el Operador, podremos seleccionar diversas opciones según nuestras necesidades.

Desde la CLI:

Prerrequisitos

- Tener acceso a una cuenta de OpenShift Container Platform con acceso de administrador de clúster.
- Que el clúster tenga habilitada la capacidad del Marketplace o la fuente de catálogo del operador de Red Hat configurada manualmente.
- Haber iniciado sesión en el clúster de OpenShift Container Platform.

Instalación

1. Creamos un archivo YAML que contenga objetos de Namespace, OperatorGroup y Subscription para suscribir un espacio de nombres al Operador OpenShift Serverless. Por ejemplo, creamos el archivo serverless-subscription.yaml con el siguiente contenido:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-serverless
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: serverless-operators
  namespace: openshift-serverless
spec: {}
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: serverless-operator
  namespace: openshift-serverless
spec:
  channel: stable
  name: serverless-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

2. Creamos el objeto de suscripción:

```
oc apply -f serverless-subscription.yaml
```

3. Knative Serving

Knative Serving es el componente responsable de:

• Desplegar aplicaciones

• Actualizar aplicaciones

• Enrutar el tráfico a aplicaciones

• Escalar automáticamente las aplicaciones

Knative Serving crea nuevos despliegues en nuevas versiones de cada aplicación. Implementa nuevas versiones para asegurarse de que el tráfico solo se redirige a versiones funcionales de la aplicación. Los pods creados por estos nuevos despliegues reciben tráfico a medida que están disponibles.

Knative Serving admite despliegues Blue/Green, despliegues Canary y despliegues progresivos.

Cuando no hay tráfico, Knative Serving escala los pods de la aplicación hasta cero y redirige el tráfico a un componente llamado Activator. El Activator envía una petición al componente Autoscaler para escalar los pods de la aplicación cuando hay nuevo tráfico. Mientras el pod de la aplicación se prepara, el Activator proxy almacena en búfer todas las solicitudes de la aplicación. Después de que finaliza el proceso de activación, el Autoscaler redirige el tráfico futuro al nuevo pod, y el Activator no se usa hasta que se escala de nuevo a cero.

Knative Serving utiliza varios servicios agrupados en las siguientes categorías:

Reconcilers: Los servicios en esta categoría determinan las diferencias entre el estado esperado de cualquier aplicación y el estado real en el clúster, y luego trabajan para hacer la transición de la aplicación al estado deseado.

Webhooks: Los servicios de webhooks capturan solicitudes a la API de Kubernetes, validan configuraciones de recursos e inyectan información de enrutamiento y red a los servicios.

Networking controllers: Estos controladores son responsables de crear y renovar certificados TLS, y de crear controladores de ingreso para redirigir el tráfico a los servicios apropiados.

Autoscaler: consiste en un conjunto de servicios que escalan hacia arriba y hacia abajo las aplicaciones de los usuarios en función del tráfico entrante. En resumen, establece el número de réplicas de pod, según el tráfico.

3.1. Instalación de Knative Serving

Antes de comenzar a usar Knative Serving, deberemos verificar que esté instalado en nuestro clúster RHOCP.

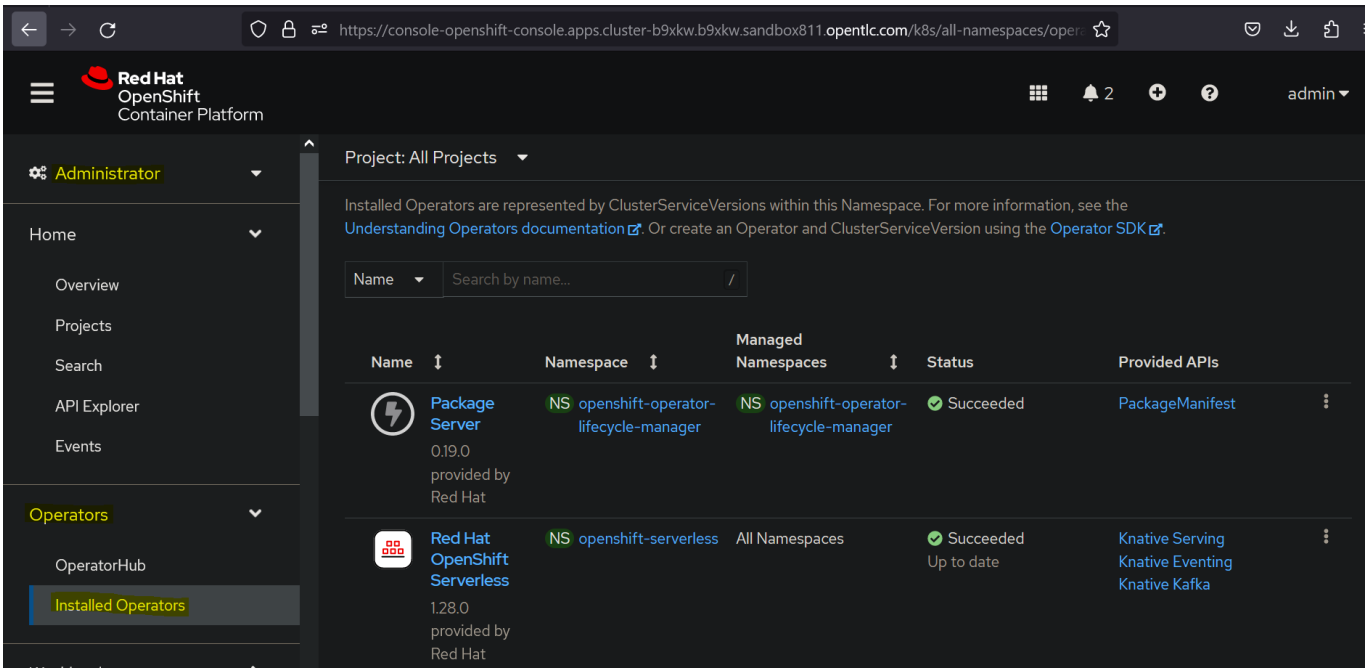
Mediante la consola web:

Prerrequisitos

- Deberemos tener acceso a una cuenta en OpenShift Container Platform con acceso de administrador de clúster.
- Deberemos haber iniciado sesión en la consola web de la plataforma de contenedores OpenShift.
- Deberemos haber instalado el operador Serverless de OpenShift.

Instalación

1. En la perspectiva del administrador de la consola web de la plataforma de contenedores OpenShift, nos vamos a Operadores → Operadores instalados.



2. Comprobamos que el menú desplegable de Proyecto en la parte superior de la página esté configurado en Proyecto: knative-serving.
3. Hacemos clic en Knative Serving en la lista de API proporcionadas para el operador Serverless de OpenShift para ir a la pestaña Knative Serving.
4. Hacemos clic en Crear Knative Serving.

5. En la página Crear Knative Serving, podemos instalar Knative Serving utilizando la configuración predeterminada haciendo clic en Crear.

6. Después de instalar Knative Serving, se crea el objeto KnativeServing y nos dirige automáticamente a la pestaña Knative Serving. Veremos el recurso personalizado knative-serving en la lista de recursos.

Desde la CLI:

Prerrequisitos

- Tener acceso a una cuenta de OpenShift Container Platform con acceso de administrador de clúster.
- Haber instalado el Operador Serverless de OpenShift.
- Tener instalado la CLI de OpenShift (oc).

Instalación

1. Crearemos un archivo llamado serving.yaml y copiaremos el siguiente YAML de ejemplo en él:

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
    name: knative-serving
    namespace: knative-serving
```

2. Aplicaremos el archivo serving.yaml:

```
oc apply -f serving.yaml
```

3.2. Instalación de Knative CLI

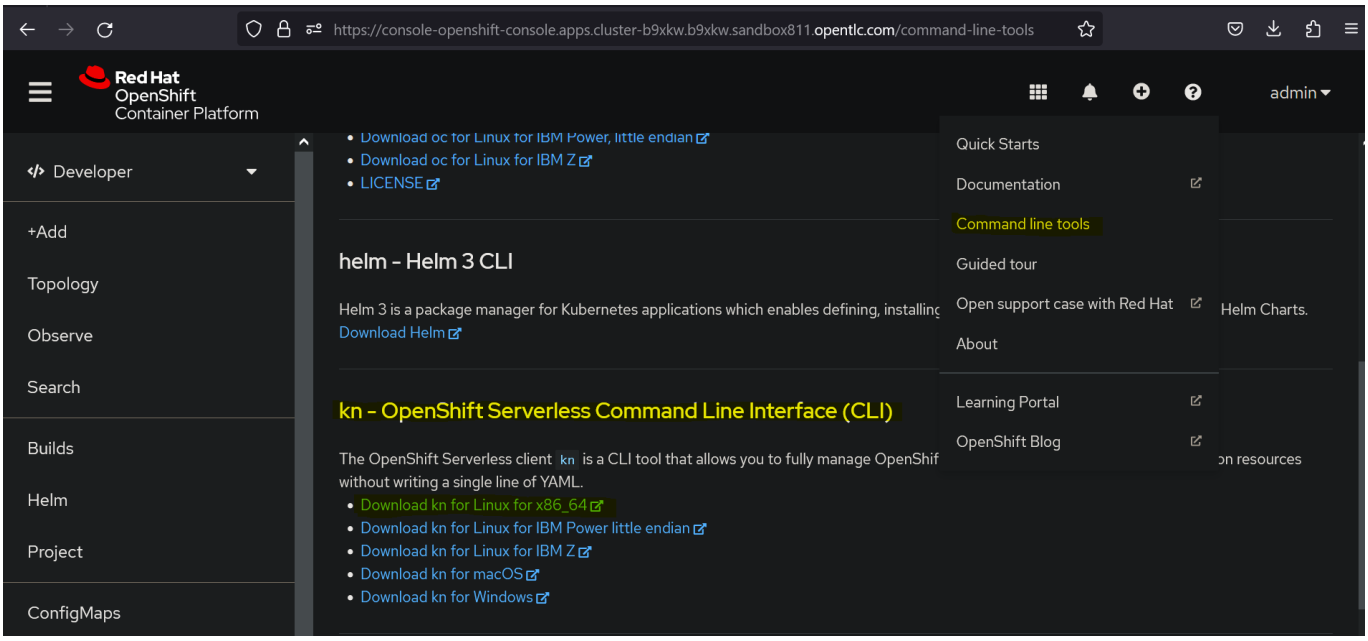
Knative CLI es una herramienta de línea de comandos que nos permite interactuar con los recursos de Knative. Podemos usar Knative CLI para crear y administrar recursos de Knative, como servicios, rutas, revisiones y configuraciones.

Prerrequisitos

- Tener acceso a una cuenta de OpenShift Container Platform con acceso de administrador de clúster.
- Tener instalado la CLI de OpenShift (oc).
- Haber instalado el Operador Serverless de OpenShift y Knative Serving.

Instalación

1. Nos dirigiremos a la consola web de Openshift y elegiremos la opción Command line tools en el icono de ayuda (?):



2. Nos descargaremos la herramienta que se adapte a nuestro sistema operativo.
3. Una vez descargada, la descomprimiremos y la añadiremos al PATH de nuestro sistema operativo. En mi caso:

```
tar -xf kn-linux-amd64.tar.gz
sudo cp kn /usr/local/bin
```

3.3. Desplegando aplicaciones Serverless

Podemos desplegar una aplicación Serverless creando un objeto de servicio Knative. El siguiente ejemplo muestra cómo podemos definir un servicio Knative simple usando un archivo YAML.

```
apiVersion: serving.knative.dev/v1 # La API de Knative Serving.
kind: Service  # El CRD de Knative Service.
metadata:
  name: echo  # El nombre de tu aplicación Serverless.
spec:
  template:
    spec:
      containers:
      - image: quay.io/redhattraining/kbe-knative-echo:v1 # La
imagen de contenedor para desplegar.
        ports:
        - containerPort: 8080  # El puerto de la aplicación.
        env:           # Variables de entorno inyectadas en la
aplicación.
        - name: MY_VAR
          value: "12345"
```

Para desplegar esta aplicación, podemos crear el servicio Knative usando la consola web de RHOCP o mediante línea de comandos oc. Además, también podemos crear un servicio Knative mediante línea de comandos Knative (kn).

Creando un servicio Knative desde un archivo YAML

De manera similar a la creación de otros objetos de Kubernetes, se puede utilizar el comando "oc apply" para crear un servicio Knative a partir de un archivo YAML. Primero, es necesario crear el archivo YAML que define el servicio Knative.

Ejecutaríamos el siguiente comando para implementar la aplicación:

```
oc apply -f service-deployment-example.yaml
```

Para verificar que el servicio Knative se ha creado correctamente, podemos ejecutar el siguiente comando:

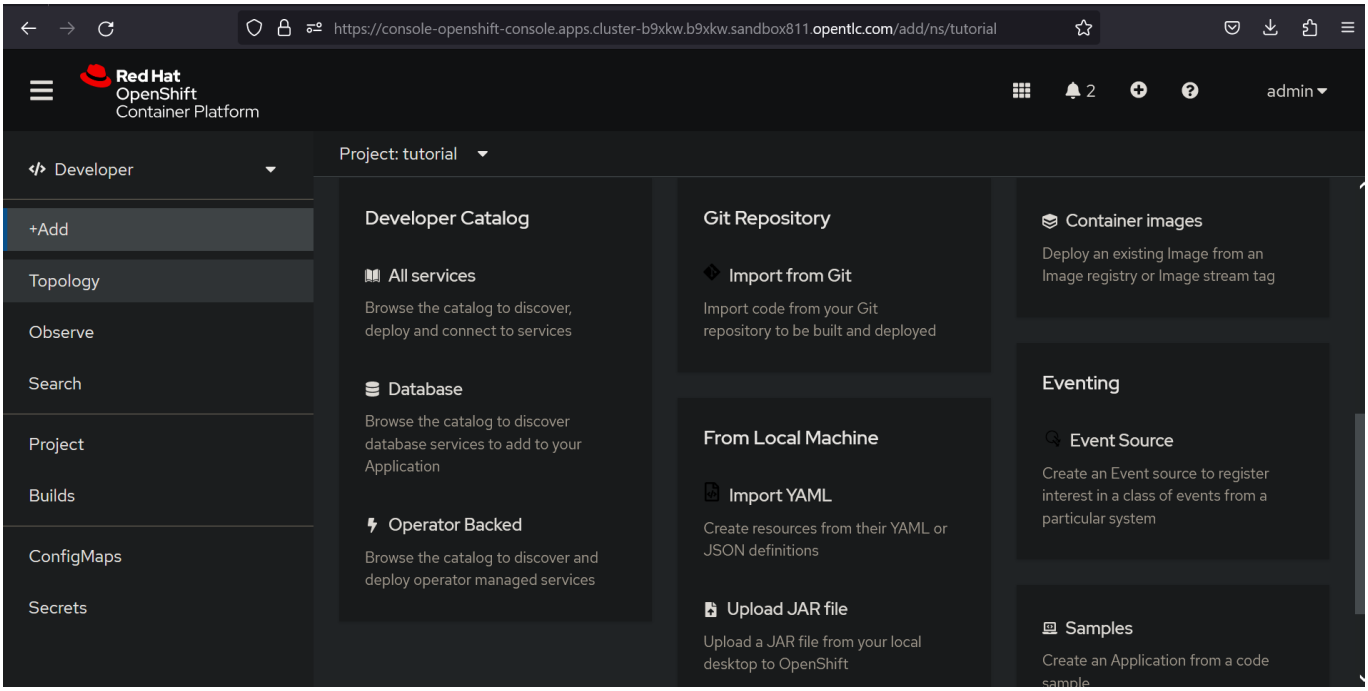
```
oc get ksvc
```

Para borrar el servicio Knative, podemos ejecutar el siguiente comando:

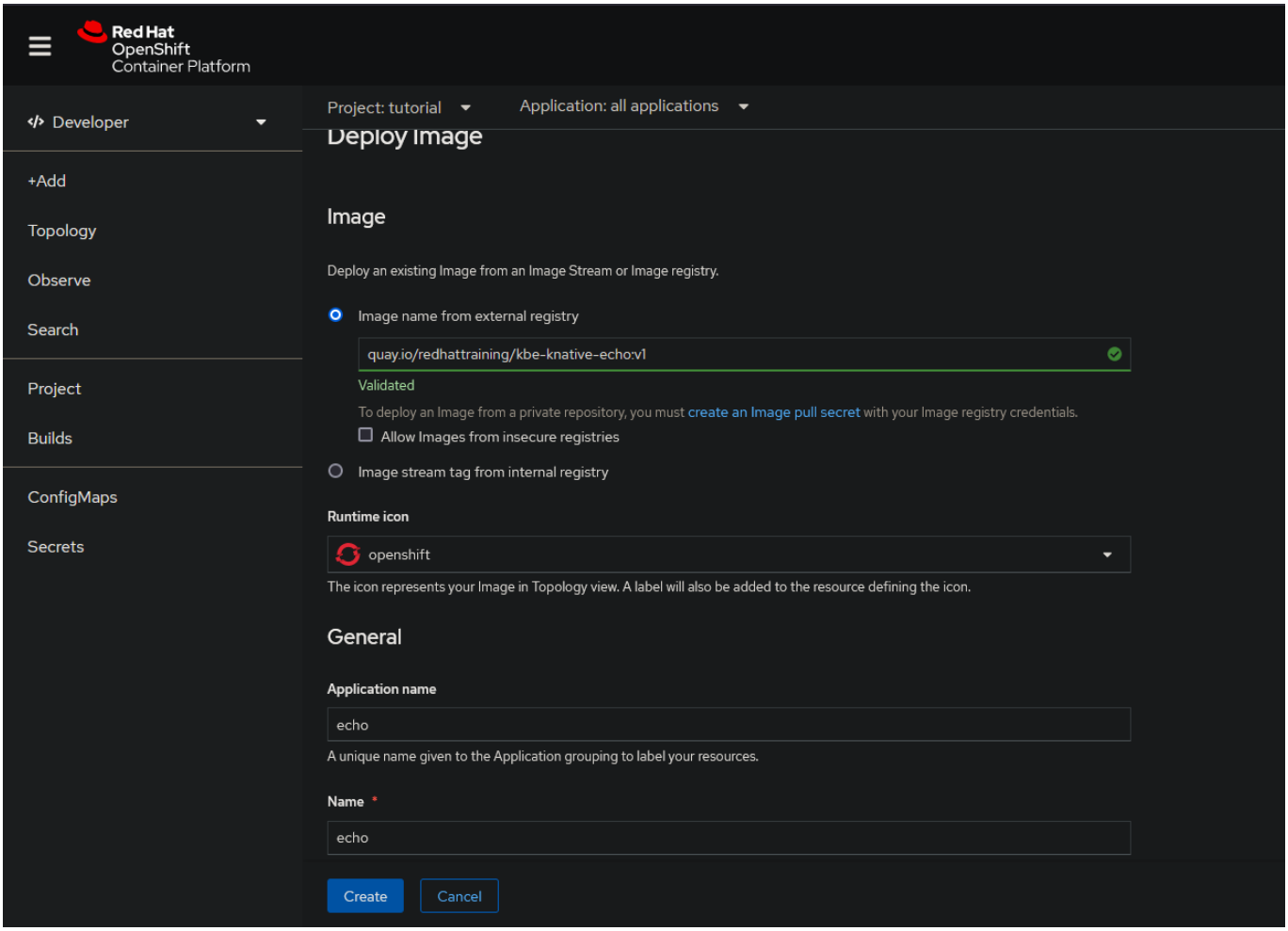
```
oc delete ksvc service-deployment-example.yaml
```

Creando un servicio Knative utilizando la consola web

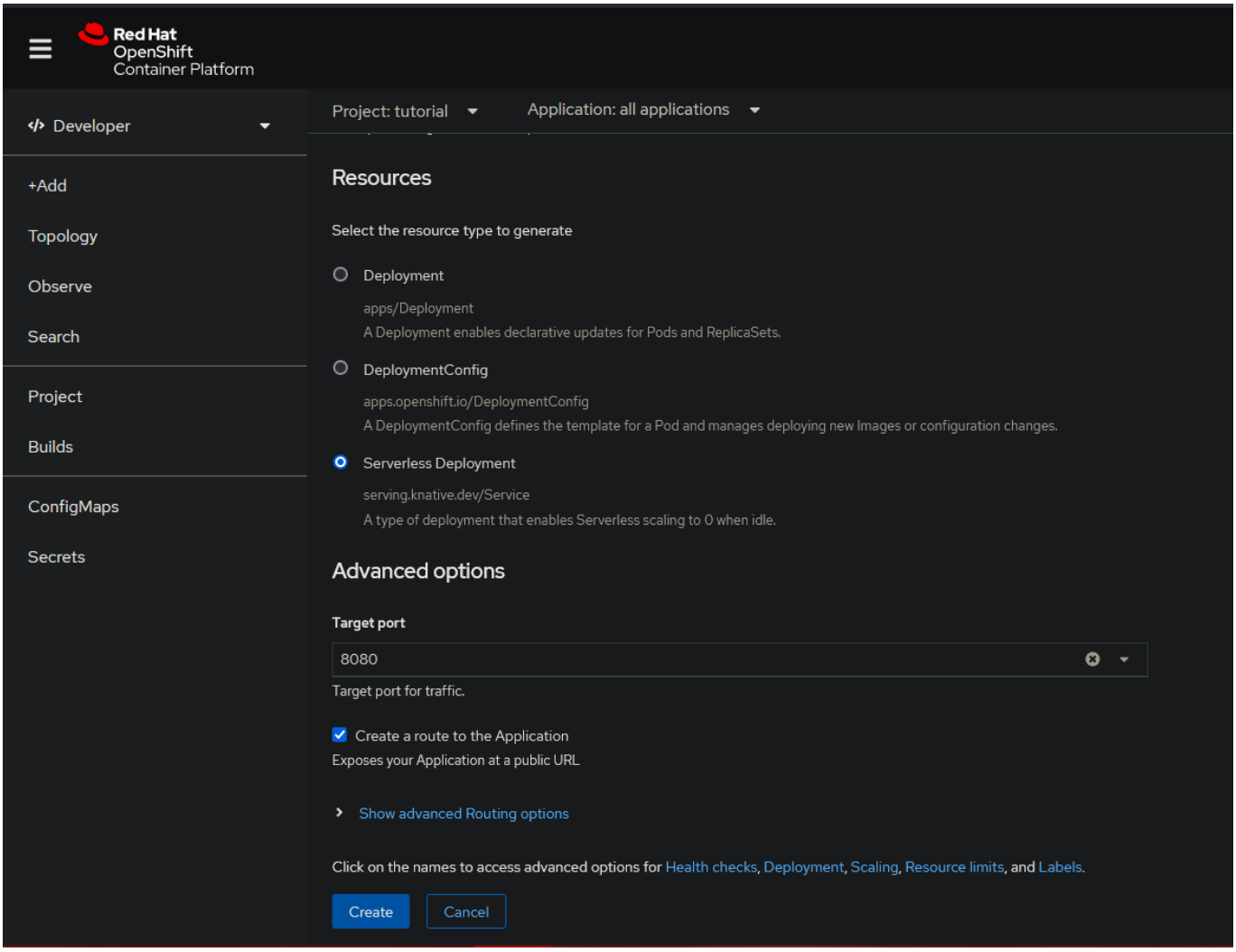
Desde la perspectiva del desarrollador, haremos clic en +Add. Si deseamos implementar una aplicación desde una imagen de contenedor, entonces hacemos clic en Imágenes de contenedor. Si deseamos implementar una aplicación desde un repositorio Git, entonces hacemos clic en Importar desde Git.



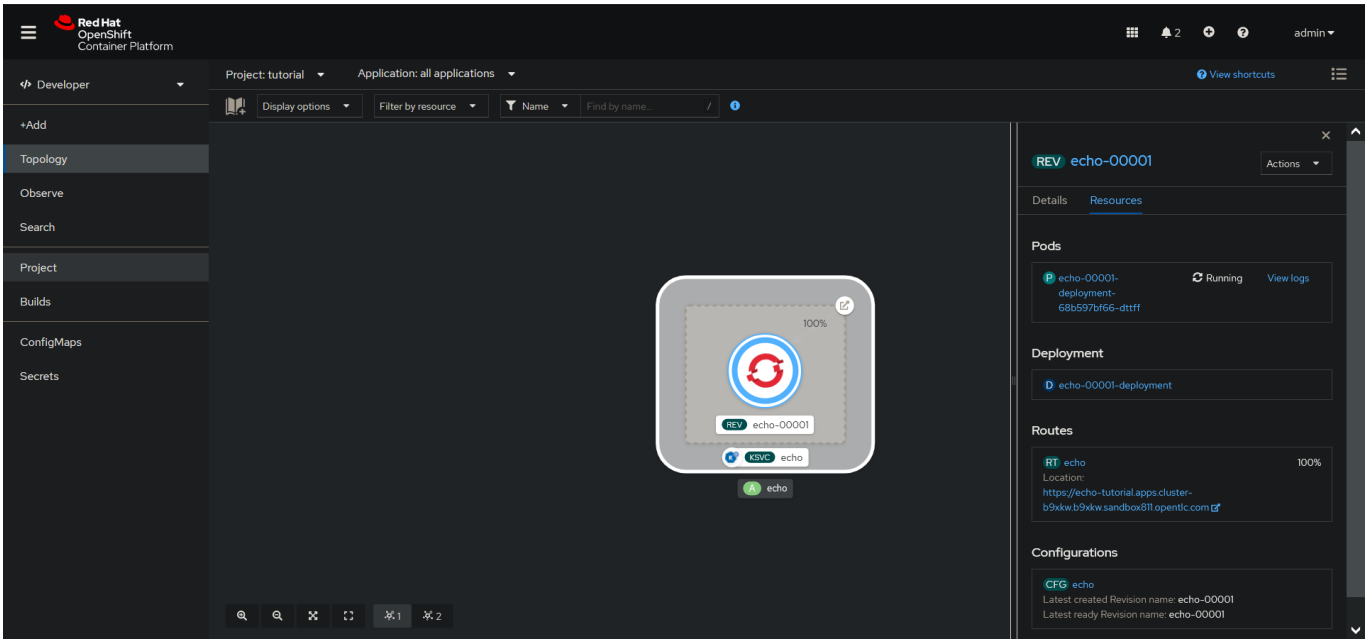
En este caso vamos a ver como se haría desde una imagen de contenedor. Debemos completar los campos requeridos en el formulario.



A continuación, en el panel de Recursos, seleccionamos la opción de Implementación Serverless. Al seleccionar esta opción, se implementa la aplicación como un servicio Knative.



Para terminar, hacemos clic en Create y ya tendremos nuestra aplicación desplegada.



Creando un Servicio Knative utilizando la CLI kn

También podemos implementar aplicaciones Serverless con la CLI kn. Para crear un nuevo servicio Knative, usaremos el comando kn service create especificando la imagen de contenedor que deseamos implementar. Opcionalmente, también podemos pasar otros parámetros de configuración, como se muestra el siguiente ejemplo:

```
kn service create echo \
--image=quay.io/redhattraining/kbe-knative-echo:v1 --port=8080
```

El comando anterior crea un nuevo servicio Knative llamado "echo", que implementa la imagen "quay.io/redhattraining/kbe-knative-echo:v1". El servicio Knative recién creado implementa la imagen especificada en un pod que escucha en el puerto 8080.

Para verificar que el servicio Knative se ha creado correctamente, podemos ejecutar el siguiente comando:

```
kn service list
```

Para borrar el servicio Knative, podemos ejecutar el siguiente comando:

```
kn service delete echo
```

Creando Servicios Privados

En algunos casos, es posible que deseemos crear servicios Knative privados que no estén expuestos al público. Para crear un servicio privado, utilizamos la opción --cluster-local, como se muestra en el siguiente ejemplo:

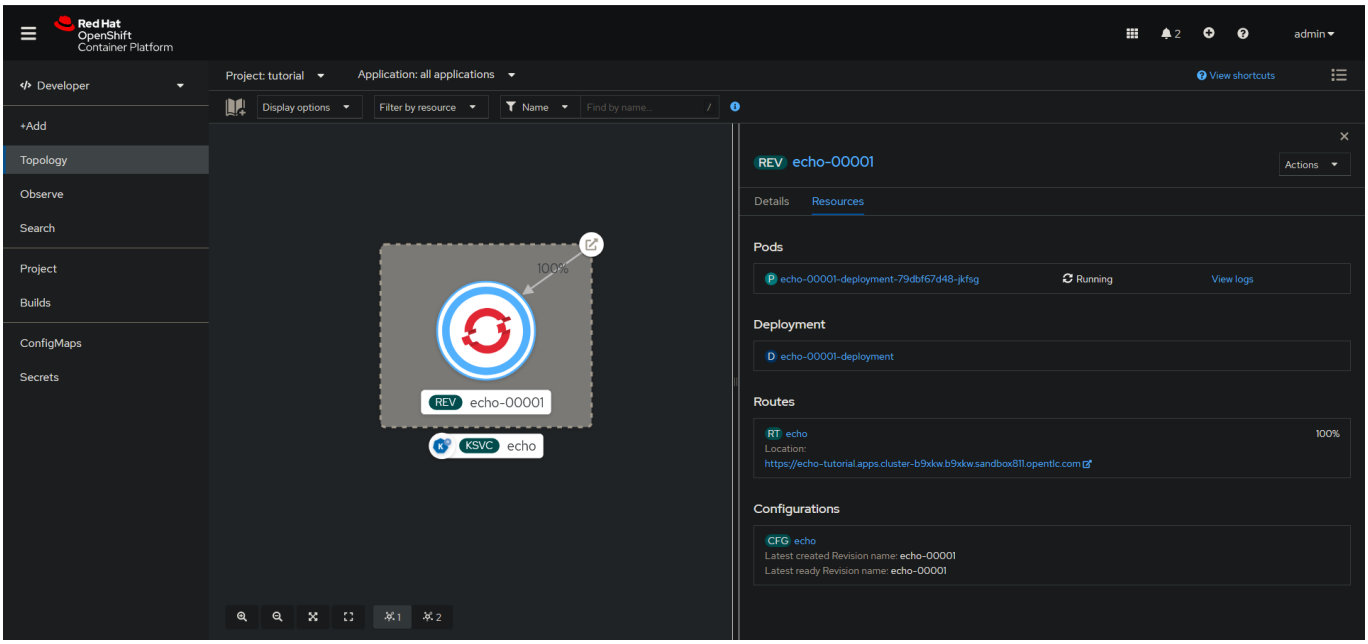
```
 kn service create echo \
--image=quay.io/redhattraining/kbe-knative-echo:v1 --port=8080 \
--cluster-local
```

Al usar la opción --cluster-local, se agrega la etiqueta networking.knative.dev/visibility=cluster-local al servicio recién creado. Cuando un servicio Knative incluye esta etiqueta, Knative crea una URL de ruta local para el servicio.

3.4. Inspeccionando una aplicación Serverless

Usando la consola web RHOCP, podemos verificar los objetos que Knative ha creado después de desplegar la aplicación. En la consola web RHOCP, cambiamos a la perspectiva de desarrollador y navegamos a la vista Topología. El gráfico de la topología muestra los servicios Knative como elementos KSVC y las revisiones como elementos REV. Si hacemos clic en alguno de estos elementos, se abrirá una barra lateral revelando información adicional.

Por ejemplo, si hacemos clic en el elemento KSVC, que representa un servicio Knative, la barra lateral mostrará una lista de los pods, revisiones y rutas Knative asociadas con el servicio.



Inspeccionando una aplicación Serverless con herramientas de línea de comandos

También podemos inspeccionar los recursos de Knative utilizando la interfaz de línea de comandos kn. Además del comando ya visto anteriormente **kn service list** podremos usar otros. Por ejemplo, para listar las revisiones de un servicio determinado, utilizamos el comando kn revision list de la siguiente manera:

```
kn revisions list --service echo
```

La opción --service filtrará las revisiones por el nombre del servicio Knative.

El CLI kn acepta varios comandos para obtener detalles adicionales sobre los recursos de Knative, como kn service y kn route, entre otros. Por ejemplo, usaremos el comando kn routes list para listar nuestras rutas de Knative y obtener las URLs públicas de nuestros servicios de Knative.

```
kn routes list
```

De igual manera, utilizaremos el comando kn routes describe para obtener detalles específicos acerca de una ruta en particular.

Además del panel web y la interfaz de línea de comandos kn, también podríamos utilizar comandos oc regulares, como list y describe, para inspeccionar los recursos personalizados de Knative. Por ejemplo, describiremos un servicio Knative en particular de la siguiente manera:

```
oc describe service.serving.knative.dev echo
```

Para ver los detalles de una revisión en particular, usaremos el comando oc describe de la siguiente manera:

```
oc describe revision.serving.knative.dev echo-00001
```

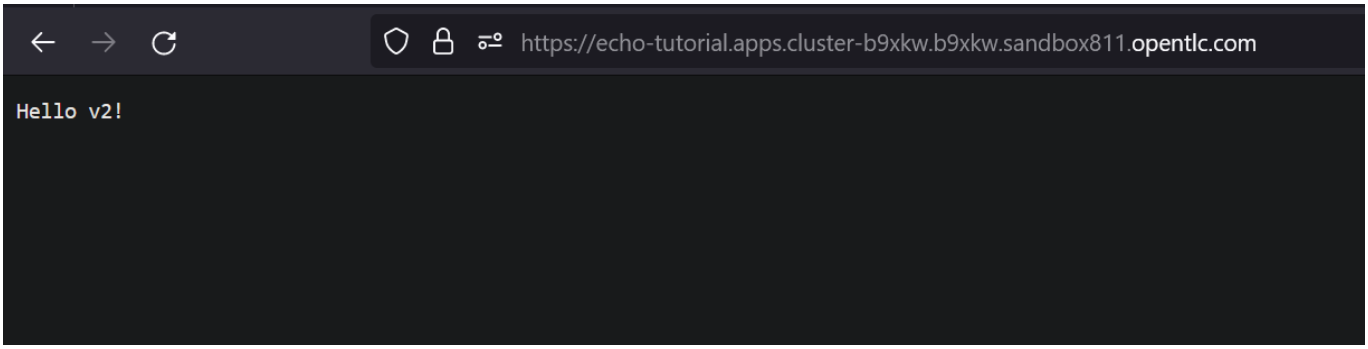
Del mismo modo, aplicaremos estos comandos para describir o listar otros recursos personalizados, como rutas de Knative.

3.5. Actualizando una aplicación Serverless

El método preferido, según la fuente oficial, para actualizar un servicio Knative es utilizar el comando "kn service update" de la siguiente manera:

```
kn service update echo \
--image=quay.io/redhattraining/kbe-knative-echo:v2
```

Si nos vamos a la web, podremos ver que ahora dice Hello v2 en lugar de Hello v1.



Tenemos que tener en cuenta que, al actualizar un servicio Knative con "kn", no es necesario especificar la lista completa de parámetros de configuración. Solo pasaremos los parámetros de configuración que necesita actualizar.

Por ejemplo, si deseamos actualizar un servicio Knative privado para que sea público, usaremos la opción "--no-cluster-local".

```
kn service update echo --no-cluster-local
```

En caso de querer actualizar un servicio Knative para que sea privado, usaremos la opción "--cluster-local".

```
kn service update echo --cluster-local
```

Alternativamente, podríamos actualizar un servicio Knative editando el YAML del servicio Knative en la consola web RHOCP o utilizando los comandos "oc patch" o "oc apply".

4. Gestionar Revisiones de Servicios y Control de Tráfico

Cada aplicación Serverless de Red Hat OpenShift tiene un recurso de Servicio Knative, un recurso de Configuración Knative y uno o más objetos de Revisión Knative. OpenShift Serverless crea una nueva revisión para cada nueva configuración y para las actualizaciones de la configuración.

Un cambio de configuración podría ser algo tan pequeño como cambiar el valor de una variable de entorno. El otro extremo es que el cambio de configuración podría usar una imagen completamente diferente. En ambos casos, el resultado es una nueva revisión.

Por ejemplo, el siguiente comando actualiza la aplicación de echo con un nuevo valor para la variable de entorno TARGET.

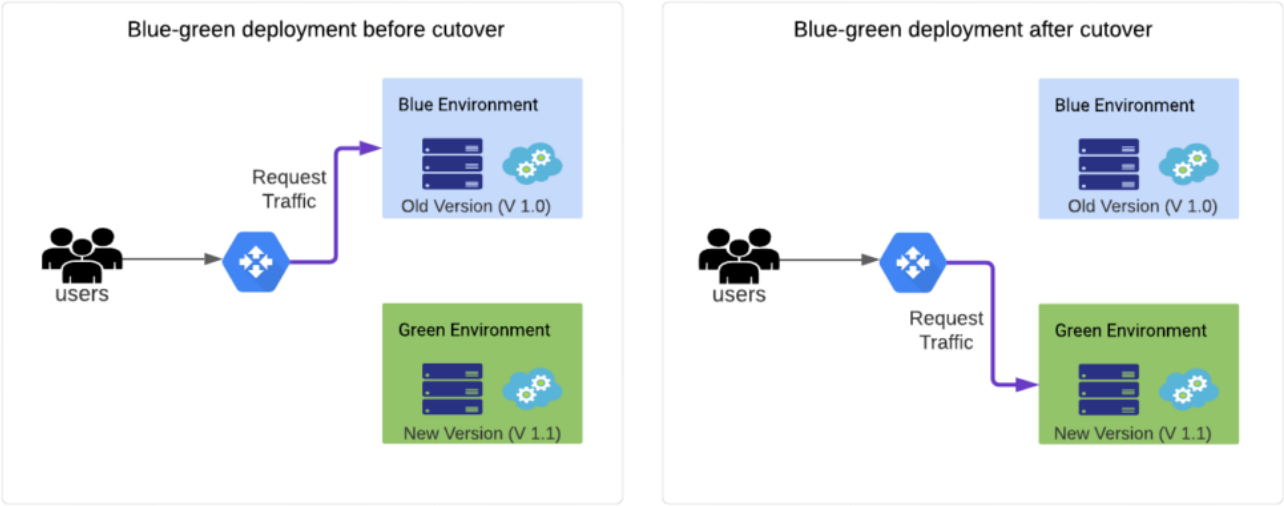
```
kn service update echo \
--env TARGET="Revision Green"
```

Por defecto, OpenShift Serverless asigna todas las solicitudes a la última revisión del servicio, sin embargo, todas las revisiones anteriores siguen estando disponibles. OpenShift Serverless proporciona controles que nos permiten especificar cuánto tráfico va a cada revisión. Este control de tráfico flexible permite muchos escenarios de implementación, como Blue/Green, Canary o incluso algo más personalizado que se adapte a nuestras necesidades.

4.1 Implementación Blue/Green

El escenario de Implementación Blue/Green describe la actualización de una aplicación desde una versión, referida como la versión Blue, a una segunda versión, referida como la versión Green. Un beneficio significativo de la implementación Blue/Green es que el escenario mantiene una disponibilidad del 100% para el usuario final. No hay tiempo de inactividad para una implementación Blue/Green.

El escenario comienza con todos los usuarios que acceden a la versión Blue. Luego, comienza la implementación de una nueva versión, la versión Green, mientras Blue sigue funcionando. Cuando la versión Green está lista para estar disponible públicamente, el enrutador dirige todo el tráfico nuevo para acceder a la versión Green. El escenario detiene la versión Blue, después de que finalice todas las solicitudes actuales. La transición de Blue a Green es perfecta para los usuarios finales.



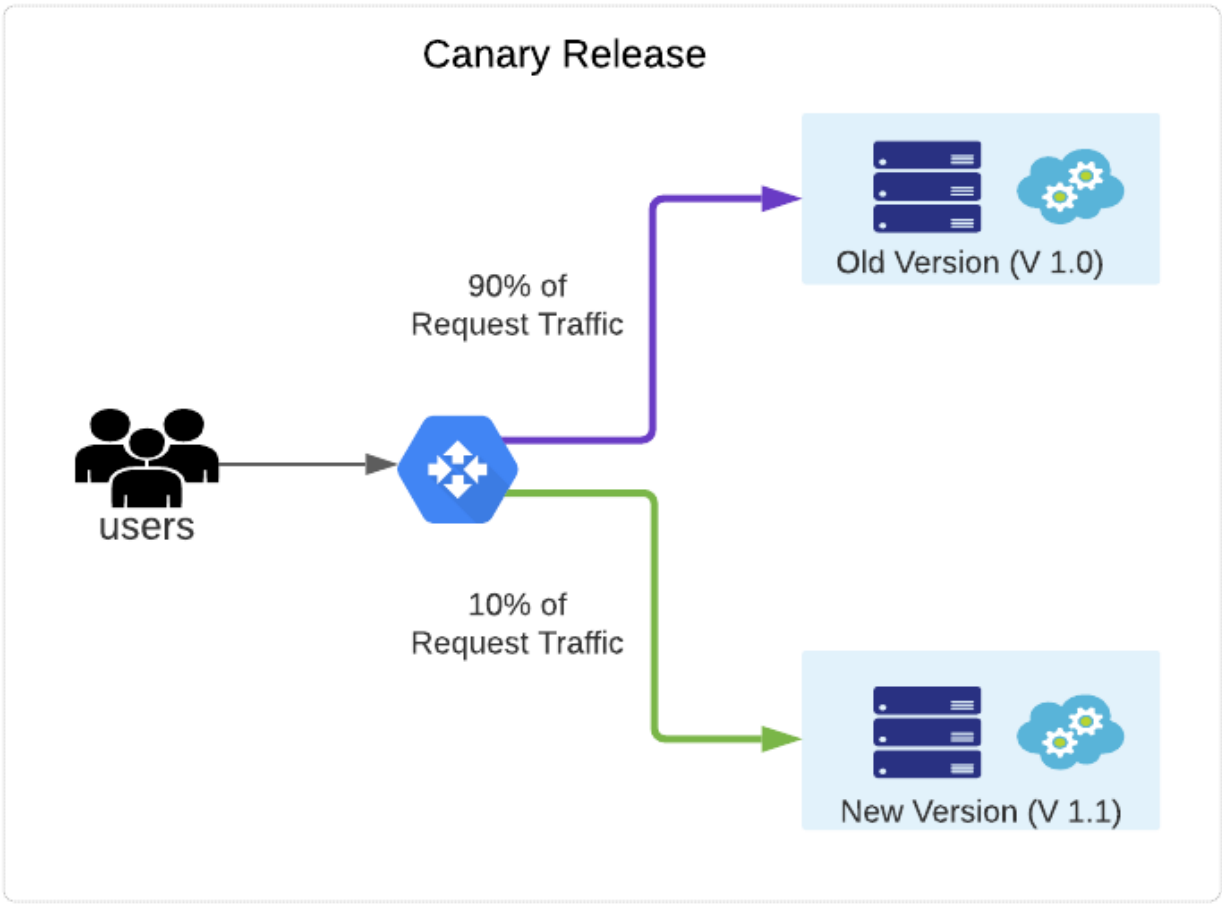
OpenShift Serverless realiza una implementación Blue/Green de forma predeterminada. OpenShift Serverless puede determinar cuándo una revisión está lista para recibir tráfico mediante las sondas de supervivencia. Al crear una nueva revisión, la revisión anterior recibe todo el tráfico hasta que la nueva revisión esté lista. Luego, el enrutador cambia automáticamente la ruta para dirigir todo el tráfico nuevo a utilizar la nueva revisión. La revisión anterior se cierra de forma ordenada.

El escenario de implementación Blue/Green funciona bien cuando hay una alta confianza en la nueva versión del software. Sin embargo, puede que no sea la mejor opción para un cambio de software importante.

4.2. Implementación Canary

El escenario de Implementación Canary es un enfoque más cauteloso que a menudo es mejor para cambios importantes de software. Para un cambio importante de software, es una buena idea implementar la nueva versión en una pequeña muestra de usuarios. La pequeña muestra puede ayudar a determinar si la nueva versión proporciona todos los beneficios esperados.

Si la nueva versión funciona según lo esperado, entonces se puede aumentar gradualmente el porcentaje de solicitudes que se dirigen a la nueva versión. De lo contrario, si la nueva versión resulta problemática, se puede deshacer redirigiendo a todos los usuarios a la versión anterior.



Los controles de enrutamiento de tráfico Serverless de OpenShift proporcionan la flexibilidad para dirigir las solicitudes a múltiples revisiones. Esta flexibilidad permite el uso de implementaciones de canarios o cualquier variación personalizada necesaria para apoyar el negocio.

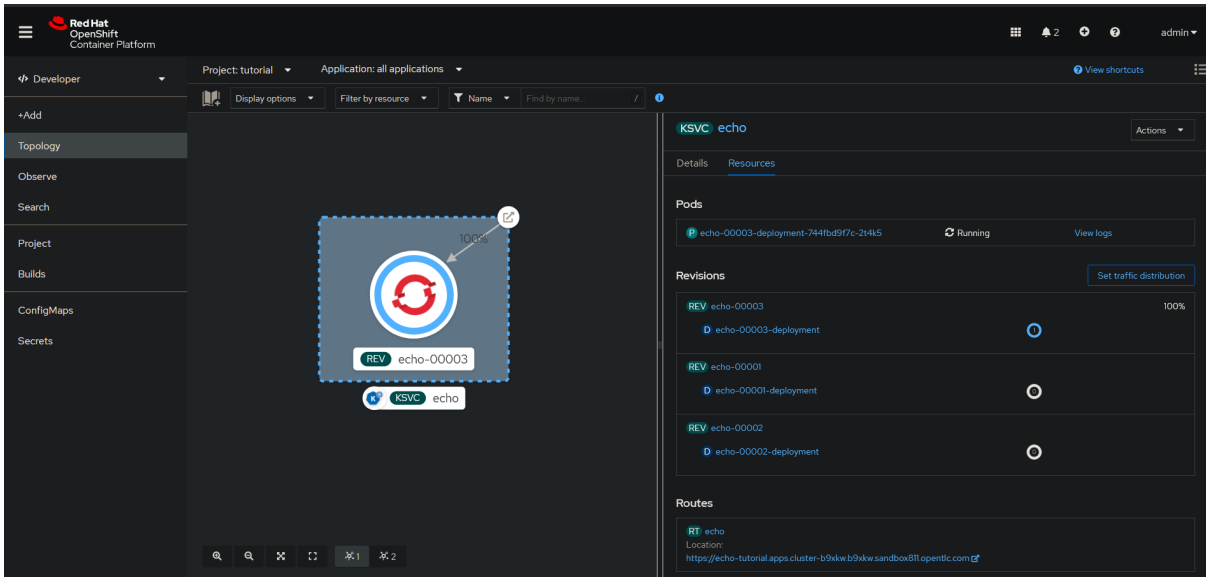
4.3. Enrutamiento de tráfico

Al crear o actualizar un servicio, OpenShift Serverless crea una nueva revisión para el servicio. Usaremos el comando kn revision list para ver una lista de revisiones disponibles. La salida de este comando tiene una columna llamada TRAFFIC que muestra el porcentaje de solicitudes enrutadas a esa revisión:

```
kn revision list
```

Asimismo, la consola web proporciona una vista gráfica de todas las revisiones. Para ver la lista de revisiones en la consola web, iniciaremos sesión en la plataforma de contenedores Red Hat OpenShift (RHOCP) y seleccionaremos la perspectiva Desarrollador.

A continuación, nos iremos a la vista Topología y haremos clic en el objeto KSVC que corresponde a su aplicación. En la barra lateral que se abre, haremos clic en Recursos.



Enrutamiento en OpenShift Serverless

Las rutas en OpenShift Serverless describen cómo asignar solicitudes HTTP entrantes a revisiones específicas en OpenShift Serverless. Usaremos kn route list para ver las rutas:

```
kn route list
```

Use kn route describe para ver los detalles de una ruta:

```
kn route describe echo
```

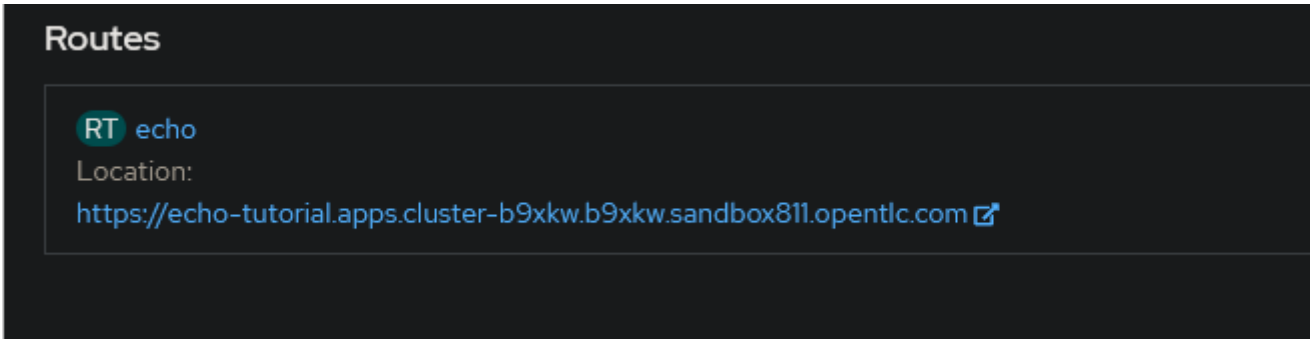
Configuración de implementaciones Canary

Supongamos que acaba de desarrollar una nueva versión de su software. La nueva versión tiene cambios significativos desde la versión anterior. Decidimos usar una implementación Canary para lanzar la nueva versión solo a una pequeña muestra de usuarios.

Recordemos que al crear una actualización, se dirige automáticamente el enrutador para enviar todo el tráfico a la última revisión. Para evitar esto, primero deberemos fijar la ruta para usar la revisión actualmente implementada.

Después de fijar la ruta, agregaremos una nueva versión no dirige ningún tráfico a la nueva revisión. El siguiente ejemplo muestra cómo fijar la ruta.

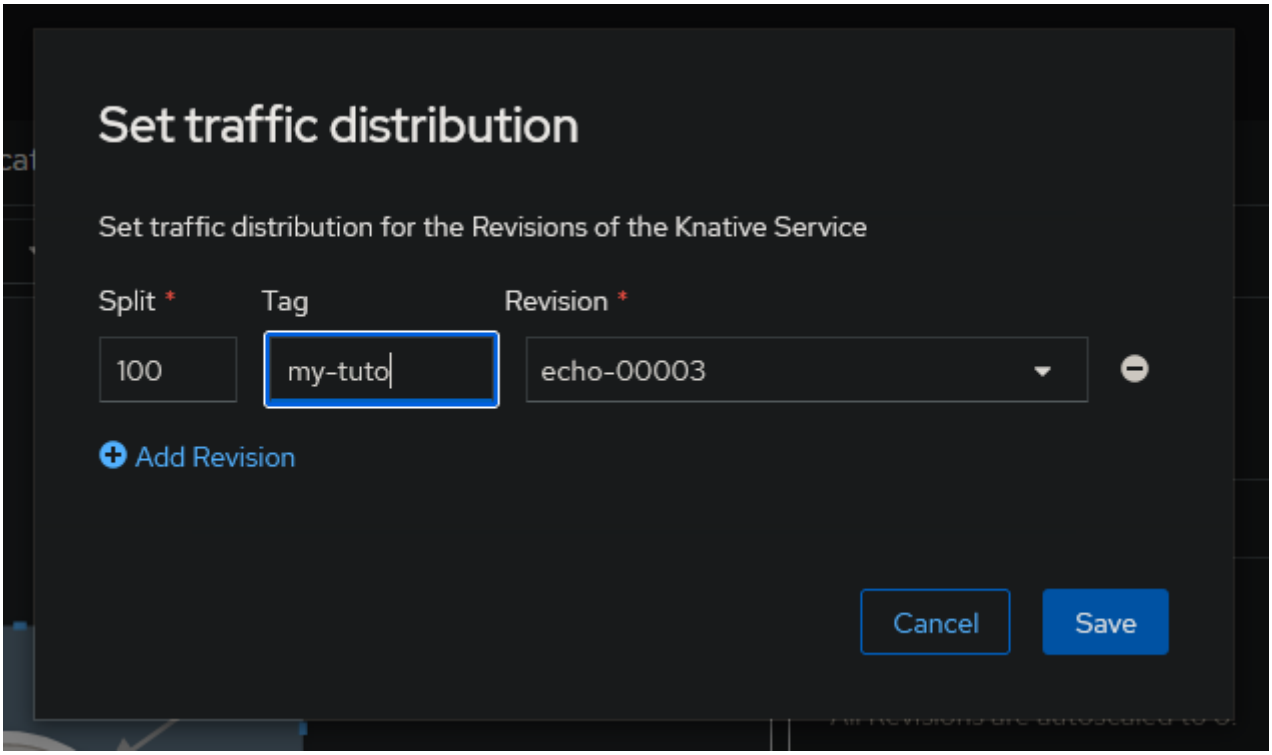
Iniciamos sesión en la consola web RHOCP y seleccionamos la perspectiva Desarrollador. A continuación, iremos a la vista Topología y haremos clic en el objeto KSVC que corresponde a su aplicación. En la barra lateral que se abre, hacemos clic en Recursos. Bajo la etiqueta Rutas, haremos clic en la ruta de su aplicación.



Desde la página de detalles de la ruta que se abre, haremos clic en YAML. La sección spec.traffic confirma que el 100 por ciento de todo el tráfico se envía a la última revisión. La sección de tráfico solo especifica la configuración y no una revisión específica.

```
apiVersion: serving.knative.dev/v1
kind: Route
...
spec:
  traffic:
    - configurationName: echo
      latestRevision: true
      percent: 100
status:
...
```

Regresaremos a la vista de Recursos y haremos clic en Establecer distribución de tráfico. En el campo de etiqueta, agregaremos un nombre descriptivo de la etiqueta y haremos clic en Guardar.

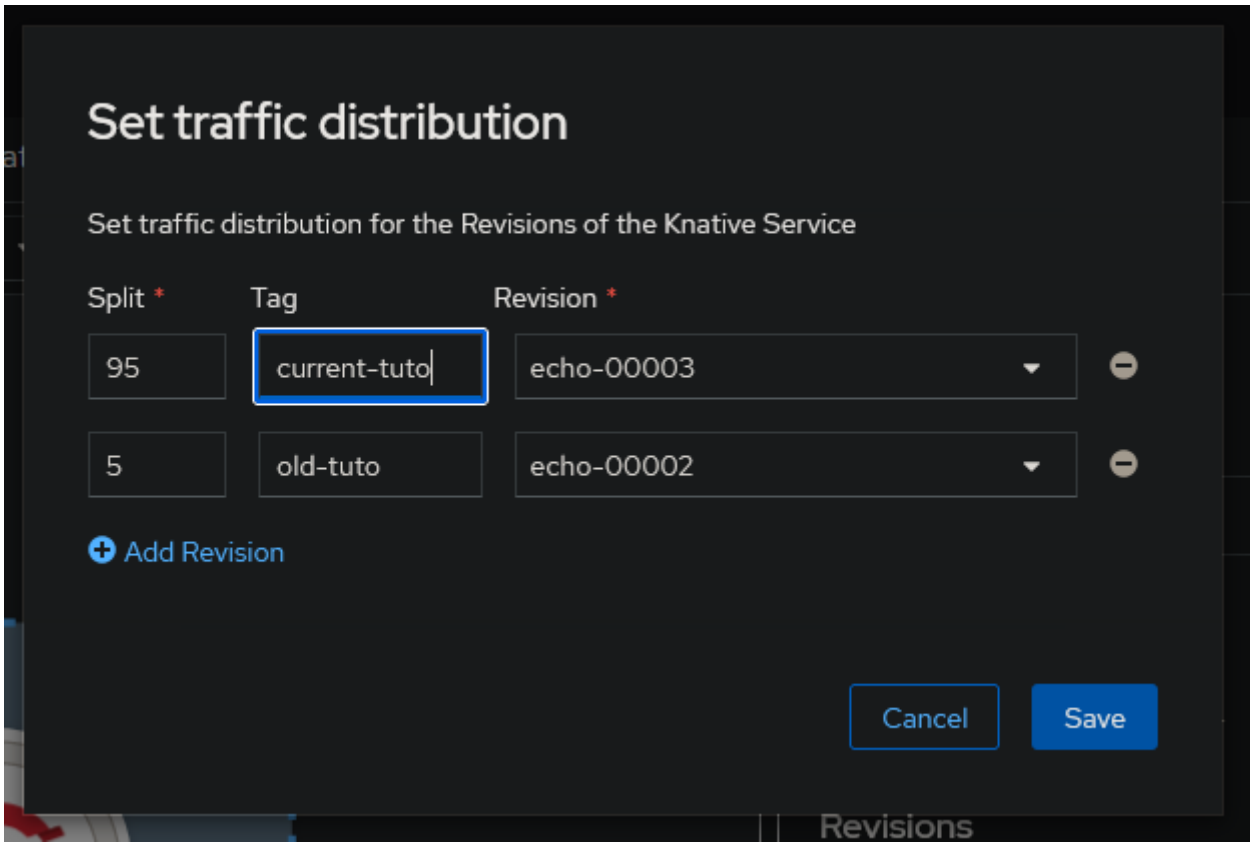


Volveremos a la vista YAML de los Detalles de la ruta. Observaremos que la sección spec.traffic ahora muestra que la ruta está fijada a una revisión y no solo a la configuración.

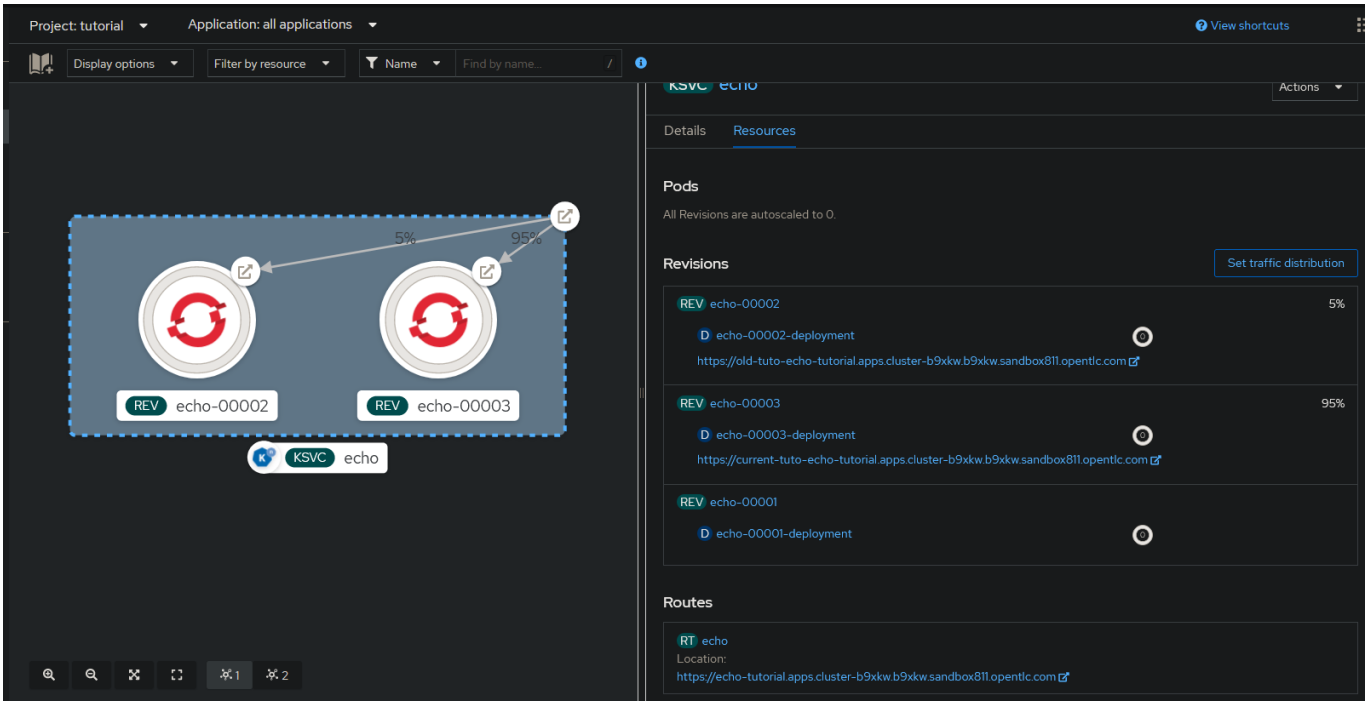
```
apiVersion: serving.knative.dev/v1
kind: Route
...
spec:
  traffic:
    - latestRevision: false
      percent: 100
      revisionName: echo-00003
      tag: my-tuto
....
```

Este cambio instruye a Knative a dirigir el 100 por ciento de todas las solicitudes a la revisión echo-00003. La ruta de servicio ignora cualquier otra revisión. Esto es exactamente lo que necesita para iniciar una Implementación Canary.

Regresemos a la vista de Recursos nuevamente y hagamos clic en Establecer distribución de tráfico. Haremos clic en Agregar revisión y seleccionaremos nuestra nueva revisión en el menú desplegable para la segunda fila de la lista. En la columna de División, configuraremos la división de tráfico entre las dos revisiones y guardaremos el cambio.



La gráfica de nuestra aplicación muestra la división entre las dos revisiones.



La división de tráfico también se puede hacer desde la línea de comandos. El siguiente comando ilustra cómo usar la opción --traffic para lograr este estado deseado:

```
 kn service update echo \
--traffic echo-00001=95 \
--traffic echo-00003=5
```

En cada opción --traffic de este comando, se especifica el nombre de la revisión, un signo igual, y luego el porcentaje de tráfico dirigido a la revisión. Después de ejecutar este comando, la revisión anterior recibe el 95 por ciento de todo el tráfico, mientras que la nueva revisión recibe el 5 por ciento. El comando puede usar tantas opciones --traffic como necesitemos, pero los valores deben sumar 100.

Usaremos el comando kn route describe para confirmar los resultados del cambio:

```
kn route describe echo
```

La sección Traffic Targets de la salida confirma que la nueva revisión recibe solo el 5 por ciento de todas las solicitudes mientras que la revisión anterior sigue manejando el resto. Cuando estemos satisfechos de que la última revisión es estable, usaremos la consola web RHOCP o la línea de comandos para dirigir todo el tráfico a la última revisión.

Usaremos el siguiente comando para enrutar todo el tráfico a la última revisión desde la línea de comandos:

```
kn service update echo --traffic @latest=100
```

La opción --traffic permite muchos patrones de implementación, incluyendo despliegues Blue/Green, Canary y progresivos.

Etiquetas de ruta

La etiqueta @latest utilizada en el ejemplo anterior es un caso especial de una etiqueta de enrutamiento. Una etiqueta es un nombre que apunta a una revisión. La etiqueta @latest está siempre disponible y siempre apunta a la última revisión. Esta etiqueta es un caso especial porque cambia cada vez que se crea una nueva revisión. Otras etiquetas son estáticas, apuntando solo a una revisión asignada.

El siguiente ejemplo ilustra cómo crear una nueva etiqueta.

```
kn service update echo \
--tag echo-00003=my-new-tag
```

Este comando crea una nueva etiqueta con el nombre my-new-tag, que apunta a la revisión echo-00003. La etiqueta es parte del recurso Route en lugar del recurso Configuration. Por lo tanto, crear una nueva etiqueta no genera una nueva revisión.

La adición de la etiqueta tiene el beneficio de crear también una URL especial que se dirige a la revisión etiquetada. Para el ejemplo anterior, el usuario puede enviar una solicitud a esa revisión específica.

La URL principal, donde fluye el tráfico según las reglas de tráfico, incluye solo el nombre del servicio y el nombre del host.

```
https://echo-tutorial.apps.cluster-
b9xkw.b9xkw.sandbox811.opentlc.com
```

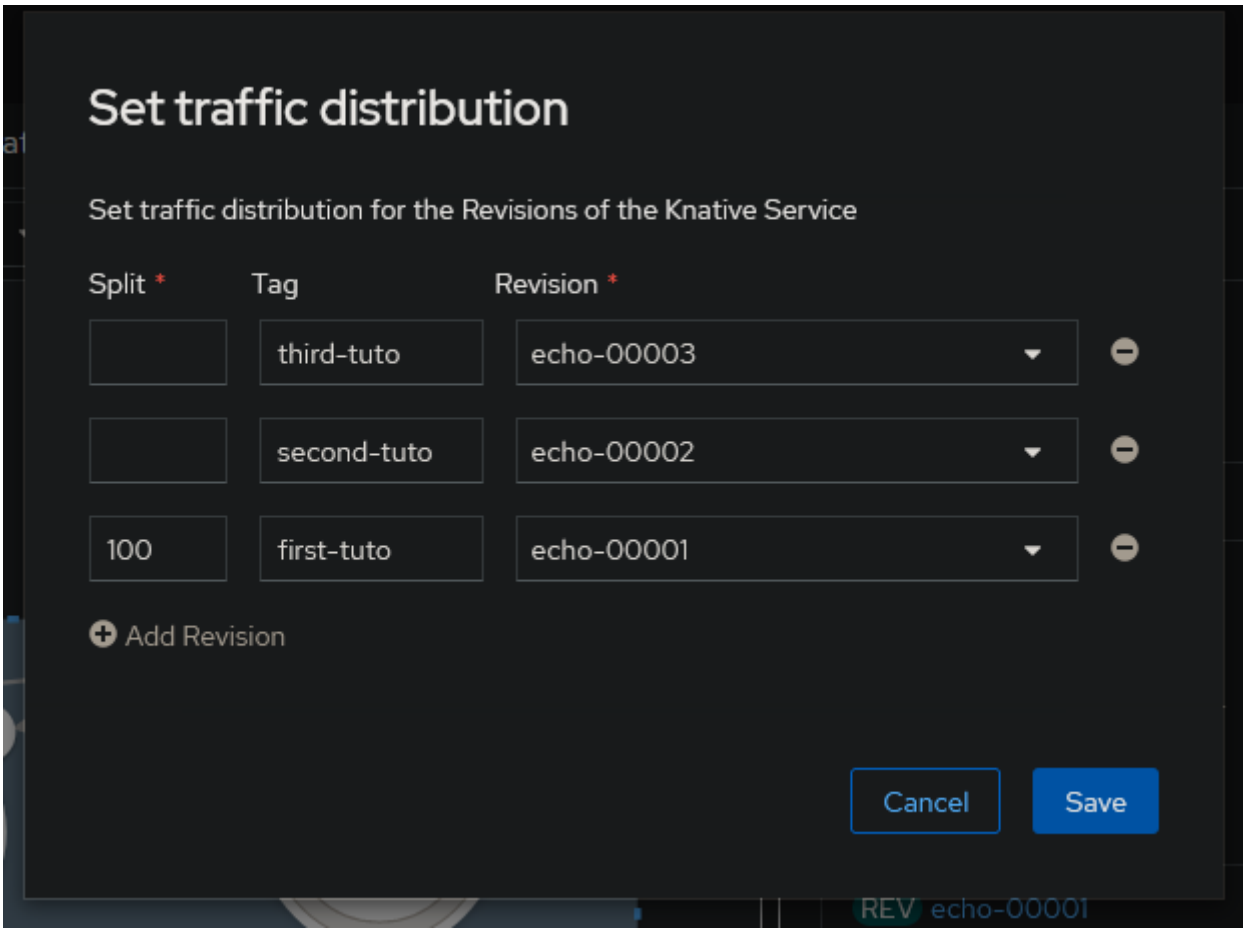
Agregar etiquetas a una revisión amplía la flexibilidad de implementación. Puede enviar una solicitud directamente a una revisión etiquetada, omitiendo las reglas de tráfico, con una URL que también incluye el nombre de la etiqueta.

```
https://my-new-tag-echo-tutorial.apps.cluster-
b9xkw.b9xkw.sandbox811.opentlc.com/
```

Implementación personalizada

Para implementar un escenario de implementación personalizado, primero fijaremos la ruta como se discutió anteriormente. Luego agregaremos tantas revisiones como sea necesario. Actualizaremos la ruta usando la opción --tag para asignar una etiqueta a cada revisión.

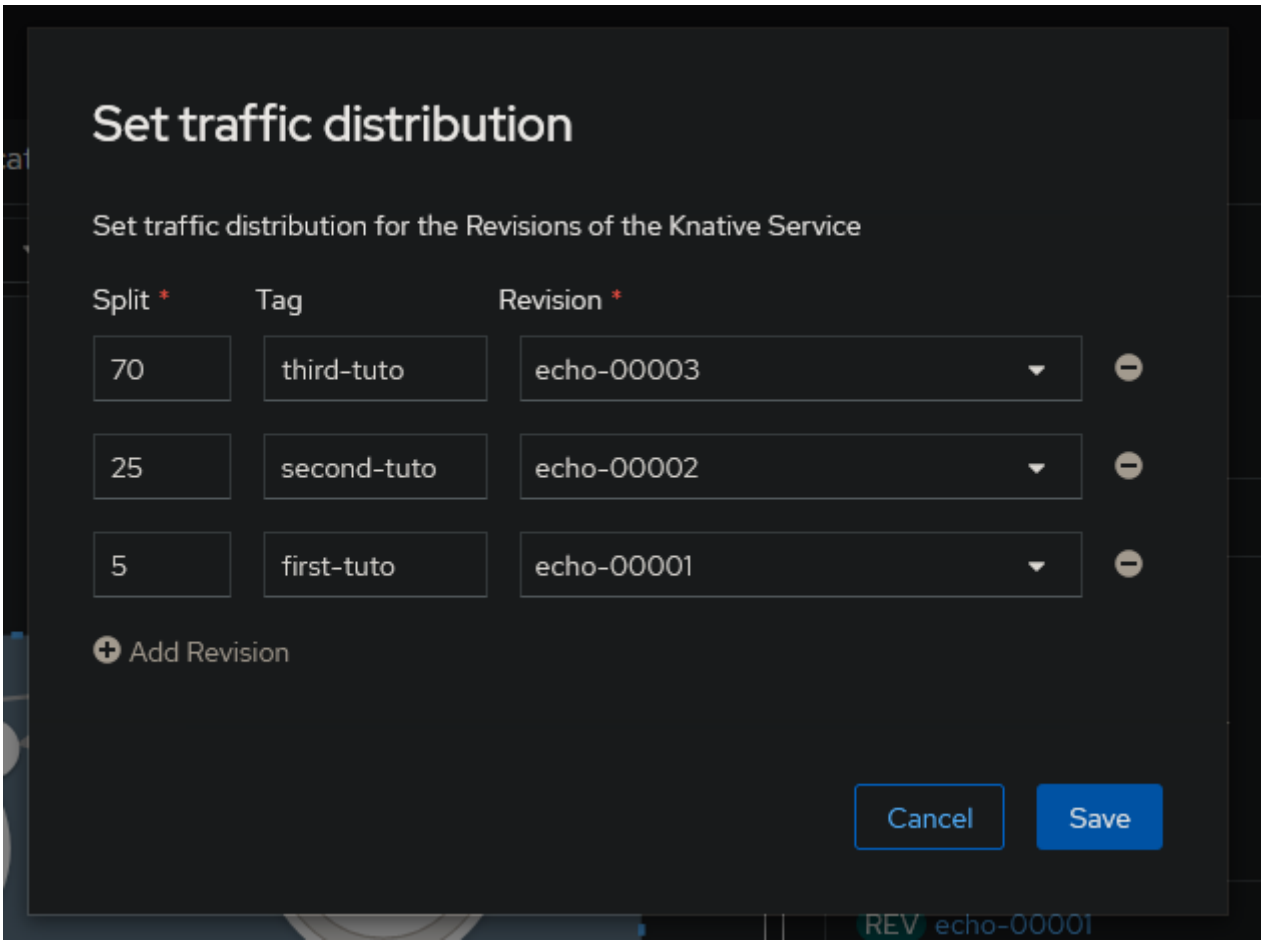
En este punto, podemos probar cualquier revisión usando las URL específicas de la etiqueta, mientras que todo el tráfico normal del usuario sigue llegando a la revisión fijada. Actualmente lo tengo configurado de la siguiente manera:



Actualizaremos la ruta para enviar un porcentaje de tráfico a las revisiones especificadas usando la opción --traffic. Podemos usar nombres de etiquetas en lugar de nombres de revisión para mayor comodidad.

```
kn service update echo \
--traffic first-tuto=5 \
--traffic second-tuto=25 \
--traffic third-tuto=70
```

Comprobamos que la ruta se ha actualizado correctamente:



Nuevamente, todas las asignaciones de tráfico deben sumar 100. El nombre de etiqueta @latest siempre está disponible y apunta a la última revisión. La asignación puede utilizar todas las revisiones disponibles o solo un subconjunto. La adición de etiquetas permite cualquier escenario de implementación personalizado.

5. Aplicaciones Serverless Autoscale

Definición de Escalado

El escalado es el proceso de ajustar los recursos del sistema disponibles para nuestra aplicación en respuesta a los cambios en la demanda de procesamiento.

Existen dos tipos de escalado:

Escalado vertical

En este tipo de escalado, aumentaremos o reduciremos los recursos de hardware asignados a nuestra aplicación o sistema. Por ejemplo, podemos aumentar la CPU y la RAM de nuestra instancia de aplicación para responder a un aumento en los visitantes de nuestro comercio electrónico durante la temporada de ventas.

Escalado horizontal

En este tipo de escalado, aumentaremos o reduciremos el número de unidades de procesamiento de nuestro aplicación o sistema. Por ejemplo, podremos aumentar el número de instancias de aplicación para responder a un aumento en los visitantes de nuestro comercio electrónico durante la temporada de ventas.

5.1. Definición de Escalado Automático

El escalado automático o autoscaling es una técnica para ajustar dinámicamente los recursos asignados a nuestra aplicación en respuesta a una fluctuación en la cantidad de trabajo a procesar.

En Knative, el componente autoscaler es responsable de escalar automáticamente nuestras aplicaciones y solo admite el escalado horizontal. El autoscaler supervisa el número de solicitudes enviadas a un servicio durante un período de tiempo, calcula el número de instancias necesarias en función del tráfico entrante y actualiza el servicio de Knative para reflejar el cálculo.

Por ejemplo, si nuestra aplicación recibe de repente un alto volumen de tráfico, entonces el autoscaler aumenta el número de instancias de nuestra aplicación para satisfacer las demandas. Cuando el pico de tráfico termina, el autoscaler reduce el número de instancias de nuestra aplicación para evitar el desperdicio de recursos.

5.2. Configuración de Autoscaling

Podemos configurar el autoscaling utilizando ajustes globales o ajustes por revisión. Cuando utilizamos ambos, los ajustes por revisión tienen prioridad sobre los ajustes globales.

El Operador Serverless de OpenShift gestiona la configuración global de una instalación de Knative. El operador propaga los valores de configuración almacenados en el Recurso Personalizado KnativeServing (CR) al ConfigMap llamado config-autoscaler. Los administradores del clúster pueden configurar los ajustes globales actualizando el CR de KnativeServing.

Como desarrollador, podemos usar el siguiente comando para inspeccionar las configuraciones globales de escalado automático:

```
oc get configmap config-autoscaler \
-n knative-serving \                # Espacio de nombres donde se
ejecuta el componente de servicio.
-o=yaml                             # Formato de salida.
```

5.2.1. Escalado a Cero

Una de las principales ventajas de los sistemas Serverless es la capacidad de escalar a cero cuando nuestra aplicación está esperando trabajo para procesar. Después de un tiempo de inactividad en nuestra aplicación, Knative marca la revisión del servicio como inactiva.

Luego, termina todos los pods que corresponden a la revisión inactiva y redirige las rutas al servicio activador. A partir de ese momento, el activador es el punto final para el tráfico de la aplicación.

Tan pronto como el activador recibe nuevas solicitudes, las almacena en búfer hasta que el escalador automático tenga un pod listo para procesar las solicitudes.

Para controlar cómo se escala nuestra aplicación a cero, podemos usar las siguientes configuraciones de escalado automático:

enable-scale-to-zero

Esta configuración global controla si las réplicas se reducen a cero o se detienen en una réplica. Podemos usar la clave enable-scale-to-zero para controlar esta configuración. El valor predeterminado para la configuración enable-scale-to-zero es verdadero, lo que significa que las réplicas se reducen a cero. Cuando el valor de la configuración es falso, entonces las réplicas se reducen a una.

stable-window

Define el período de tiempo en el que el escalador automático supervisa las revisiones para marcarlas como inactivas. Si no hay solicitudes durante el período definido, entonces el escalador automático establece la revisión como inactiva. El valor predeterminado de la ventana estable es de 60 segundos.

Podemos definir el período de tiempo globalmente con la clave stable-window o por revisión con la clave de anotación autoscaling.knative.dev/window.

Alternativamente, podemos usar la opción "--scale-window" disponible en el comando kn service para agregar o actualizar la ventana estable de un servicio Knative. El siguiente ejemplo actualiza el servicio Knative echo para tener una ventana estable de 90 segundos:

```
kn service update echo --scale-window 90s
```

scale-to-zero-grace-period

Esta configuración global define el período de tiempo en el que el escalador automático supervisa las pods inactivas y termina esas pods. El valor predeterminado es de 30 segundos, y podemos definir un período de tiempo diferente con la clave scale-to-zero-grace-period.

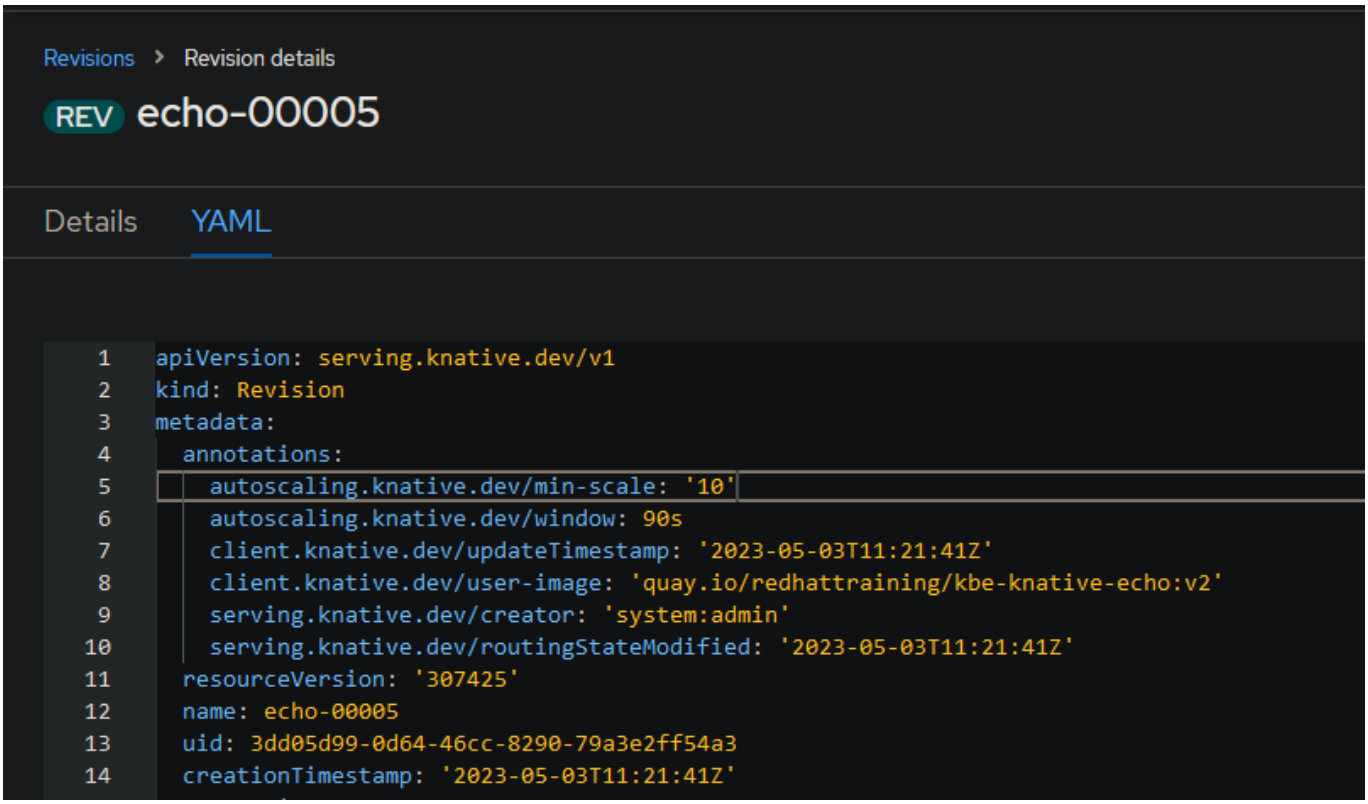
El período de terminación es el tiempo que el autoscaler tarda en terminar los pods inactivos. Puede calcularse sumando la ventana estable más el período de gracia para escalar a cero.

5.2.2. Configuración de límites

Los límites de escalado controlan el número de instancias creadas para una revisión. Podemos definir dos límites de escalado en nuestras aplicaciones Serverless:

Escala mínima

Este valor define el número mínimo de réplicas que necesita cada revisión. Podemos definir el límite globalmente con la clave min-scale, o por revisión con la clave de anotación autoscaling.knative.dev/min-scale.



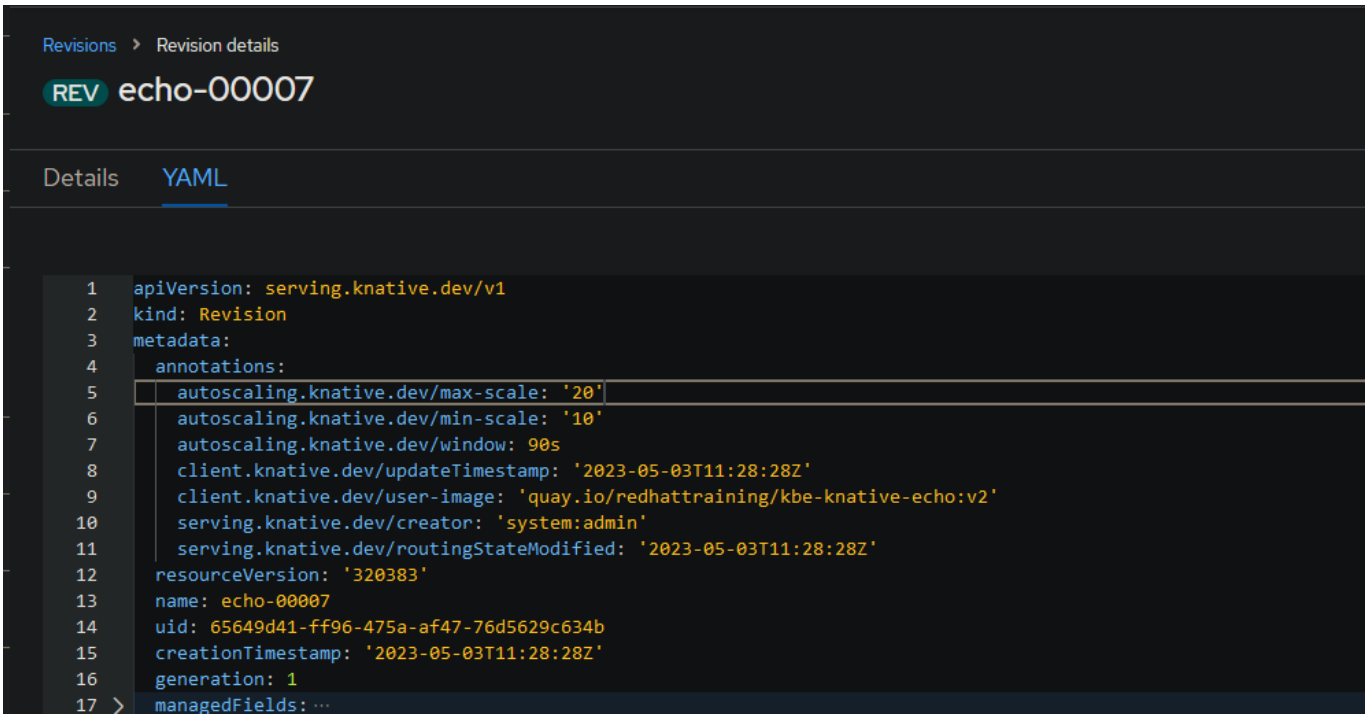
Alternativamente, podríamos utilizar la opción "--scale-min" disponible en el comando kn service para definir el límite. El siguiente ejemplo actualiza el servicio Knative echo para tener un mínimo de 10 réplicas para cada revisión:

```
kn service update echo --scale-min 10
```

Escala máxima

Esta configuración define el número máximo de réplicas que cada revisión puede tener. Knative por defecto no establece un límite superior al número de pods que podemos utilizar, lo que podría dar lugar a que una aplicación consuma un número excesivo de recursos.

Al establecer un límite máximo de escala, protegemos el sistema de la creación de demasiadas instancias de un servicio. Podemos definir el límite globalmente con la clave max-scale, o por revisión con la clave de anotación autoscaling.knative.dev/max-scale.



Alternativamente, podríamos utilizar la opción "--scale-max" disponible en el comando kn service para definir el límite. El siguiente ejemplo actualiza el servicio Knative echo para tener un máximo de 20 réplicas para cada revisión:

```
kn service update echo --scale-max 20
```

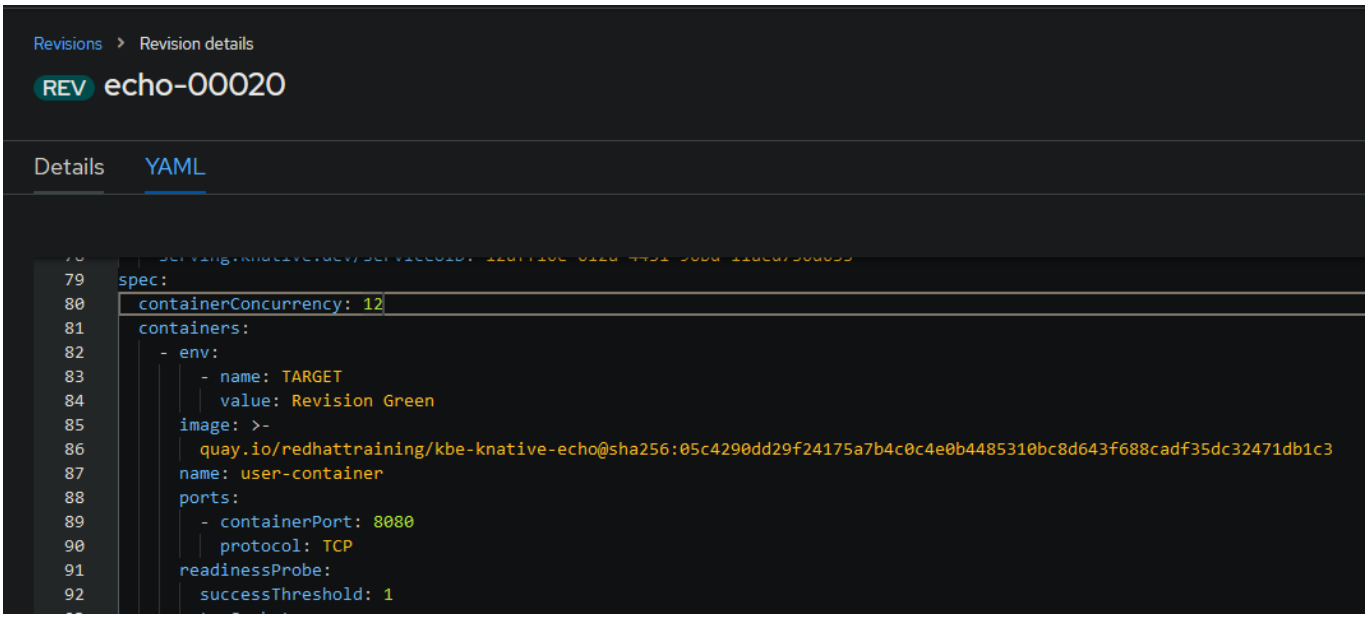
5.2.3. Configuración de concurrencia

La concurrencia determina el número de solicitudes simultáneas que una réplica puede procesar en un momento dado. Podemos configurar dos límites para la concurrencia de nuestras aplicaciones:

Límite duro

Este límite se aplica estrictamente. Es decir, si el número de solicitudes concurrentes alcanza el límite duro, entonces Knative almacena en búfer las solicitudes excedentes hasta que haya suficiente capacidad para procesarlas.

Podremos especificar un límite duro por revisión modificando la especificación containerConcurrency del servicio Knative, o utilizando la opción "--concurrency-limit" del comando kn service.

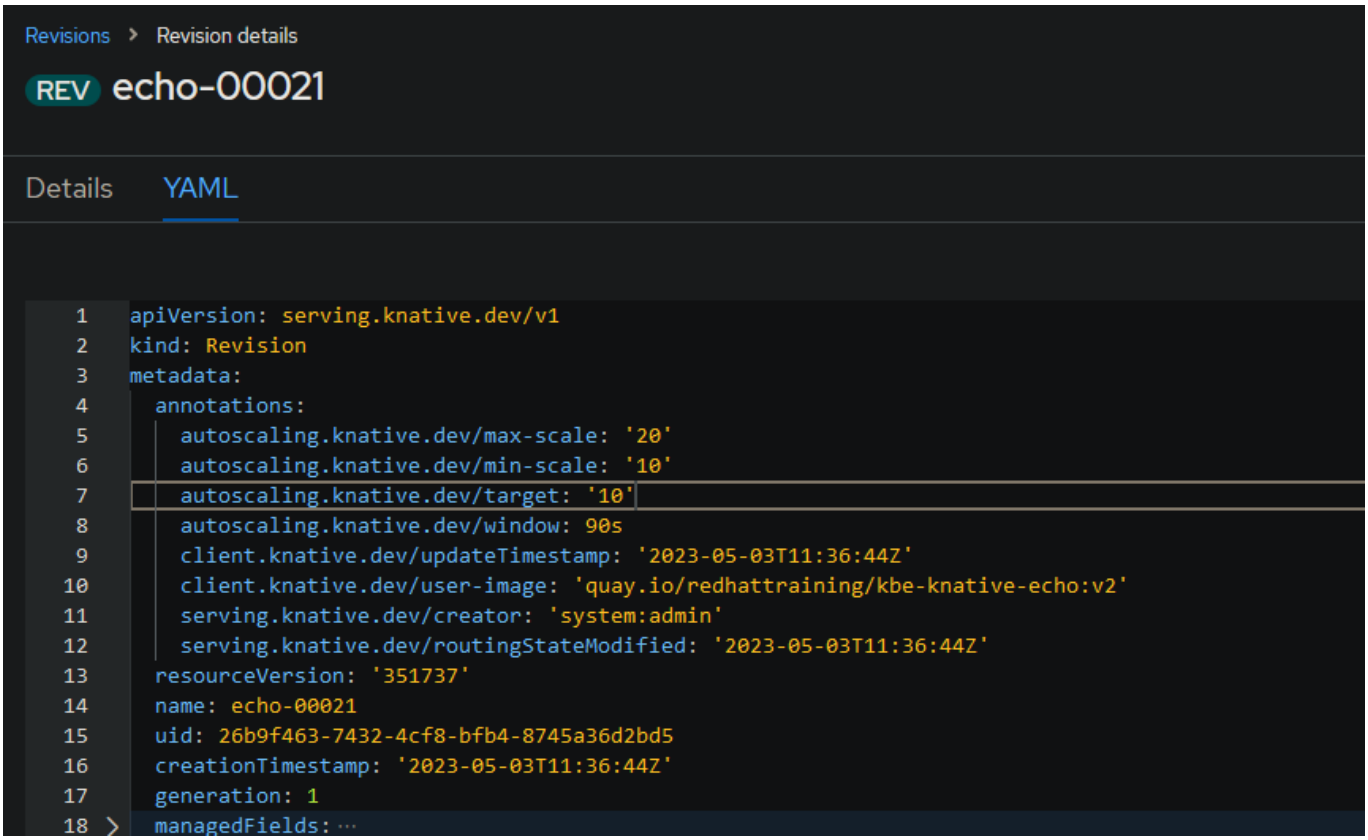


El siguiente ejemplo configura un límite duro de 12 solicitudes concurrentes en el servicio Knative llamado echo:

```
kn service update echo --concurrency-limit 12
```

Límite suave

Un límite suave es un límite de solicitudes objetivo. Por ejemplo, si hay una ráfaga repentina de tráfico, el objetivo de límite suave puede ser superado. Podemos definir el límite suave global con la clave container-concurrency-target-default, o por revisión con la clave de anotación autoscaling.knative.dev/target. El valor predeterminado es 100.



Alternativamente, podríamos utilizar la opción "--scale-target" disponible en el comando kn service para definir cuándo escalar en función del número concurrente de solicitudes entrantes.

El siguiente ejemplo actualiza el servicio Knative echo para tener un límite suave de 10 solicitudes concurrentes:

```
kn service update echo --scale-target 10
```

5.2.4. Arranques en frío

Un arranque en frío ocurre cuando nuestra aplicación no tiene pods listos y llega una solicitud. Debido a que no hay pods disponibles, la solicitud debe esperar a que Kubernetes cree un pod, y este proceso agrega latencia al procesamiento de la solicitud.

Podemos reducir este problema estableciendo un límite inferior de escala más bajo. Por ejemplo, si establecemos el número 1 como límite inferior de escala para nuestras revisiones, entonces garantizamos que tenemos un pod en ejecución y esperando para procesar solicitudes.

5.2.5. Configuración de Autoscaling mediante la Consola Web

Al utilizar la consola web de Red Hat OpenShift Container Platform (RHOCP), podremos definir o actualizar la configuración de autoscaling de los servicios de Knative desplegados. Antes vimos ejemplos de ficheros YAML, pero no se explicó como llegar a ellos.

Configuración de Autoscaling mediante la Consola Web

1. Iniciamos sesión en nuestra consola web de RHOCP y cambiamos a la perspectiva de administrador.
2. Hacemos clic en Serverless > Serving.
3. En la parte superior izquierda de la página de Serving, hacemos clic en el selector de proyecto y seleccionamos nuestro proyecto.
4. En la lista de servicios de Knative, hacemos clic en el nombre del servicio de Knative que deseamos actualizar.
5. En la parte superior de la página de detalles del servicio, hacemos clic en YAML.
6. Actualizamos la definición YAML para agregar o actualizar las anotaciones de autoscaling y hacemos clic en Guardar para persistir los cambios.

6. Casos Prácticos

Para la realización de estos dos casos prácticos, utilizaré una aplicación que posee back-end. La aplicación se llama **"temperaturas"**, es de José Domingo Muñoz Rodríguez y se encuentra en el siguiente repositorio de GitHub.

Partiremos con el código fuente de la aplicación y la desplegaremos en un cluster de OpenShift Container Platform (OCP) con Serverless, Knative Serving y Knative CLI ya instalados.

6.1. Despliegue de una aplicación Serverless con Knative Serving

Enunciado

Vamos a desplegar una aplicación Serverless con Knative Serving usando la consola web de Red Hat OpenShift Container Platform (RHOCP) y la CLI kn.

Resultados esperados de este caso práctico

Deberíamos ser capaces de realizar muchos escenarios de implementación de una aplicación Serverless con Knative Serving.

Pasos a seguir usando la CLI kn

Crearemos un nuevo proyecto llamado **"temperaturas-1"**.

```
oc new-project temperaturas-1
```

Ahora vamos a desplegar la aplicación usando la CLI kn, para ello vamos a crear un archivo YAML con la configuración de la aplicación front-end y back-end.

```
mkdir temperaturas && cd temperaturas && touch temperaturas-
frontend-service.yaml temperaturas-backend-deploymentconfig.yaml
temperaturas-backend-service.yaml
```

Contenido del archivo YAML front-end que define el servicio que proporciona nuestra aplicación, aquí es donde implementaremos la aplicación Serverless con Knative Serving:

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: temperaturas-frontend
  labels:
    app: temperaturas
    tier: frontend
spec:
  template:
    spec:
      containers:
        - name: contenedor-temperaturas
          image: iesgn/temperaturas_frontend
          ports:
            - containerPort: 3000
      replicas: 3
```

Contenido del archivo YAML back-end que en este caso es un DeploymentConfig:

```
apiVersion: apps.openshift.io/v1
kind: DeploymentConfig
metadata:
  name: temperaturas-backend
  labels:
    app: temperaturas
    tier: backend
spec:
  replicas: 1
  selector:
    app: temperaturas
    tier: backend
  template:
    metadata:
      labels:
        app: temperaturas
        tier: backend
    spec:
      containers:
        - name: contendor-servidor-temperaturas
          image: iesgn/temperaturas_backend
          ports:
            - name: api-server
              containerPort: 5000
```

Necesitará un servicio para exponer el puerto 5000 del contenedor:

```
apiVersion: v1
kind: Service
metadata:
  name: temperaturas-backend
  labels:
    app: temperaturas
    tier: backend
spec:
  type: ClusterIP
  ports:
  - name: api-server
    port: 5000
    targetPort: api-server
  selector:
    app: temperaturas
    tier: backend
```

El motivo de que el back-end sea un DeploymentConfig y el front-end use Knative Serving es porque no estamos usando Knative Eventing. Nuestro back-end siempre estaría consumiendo recursos, en cambio, el front-end solo consumiría recursos cuando haya peticiones de los usuarios. Si el back-end fuera serverless y solo estuvieramos usando Knative Serving jamás se ejecutaría, ya que no hay eventos que lo desencadenen cuando haya peteciiones de los usuarios al front-end.

Aplicamos los archivos YAML.

```
oc apply -f temperaturas-backend-deploymentconfig.yaml
oc apply -f temperaturas-backend-service.yaml
kn service apply -f temperaturas-frontend-service.yaml
```

Comprobamos que se ha creado el servicio serverless para el front-end con el comando:

```
kn service list
```

Comprobamos que se han creado los pods, las rutas, los deployments y los servicios con el comando:

```
oc get pods,route,deployment,service
```

Vamos a probar la aplicación.

Como vemos el front-end está funcionando correctamente, y conecta con el back-end. Al acceder a la web por primera vez, el front-end tarda unos segundos en cargar, esto es porque se han escalado a cero pods por inactividad, y al volver a recibir una petición se ha escalado a 3 pods (número de pods que hemos definido por defecto en el YAML).

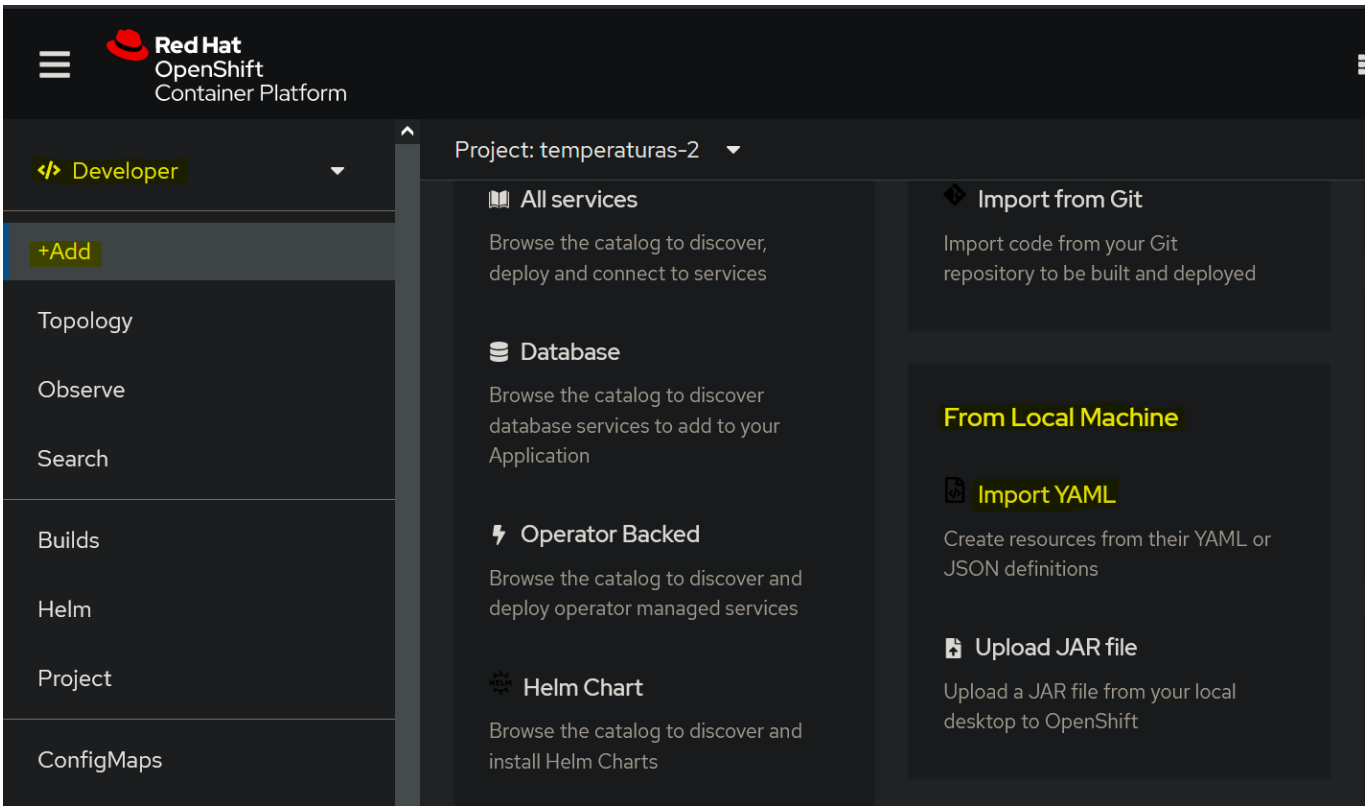
Pasos a seguir usando la consola web de RHOCP

Crearemos un nuevo proyecto llamado **"temperaturas-2"**.

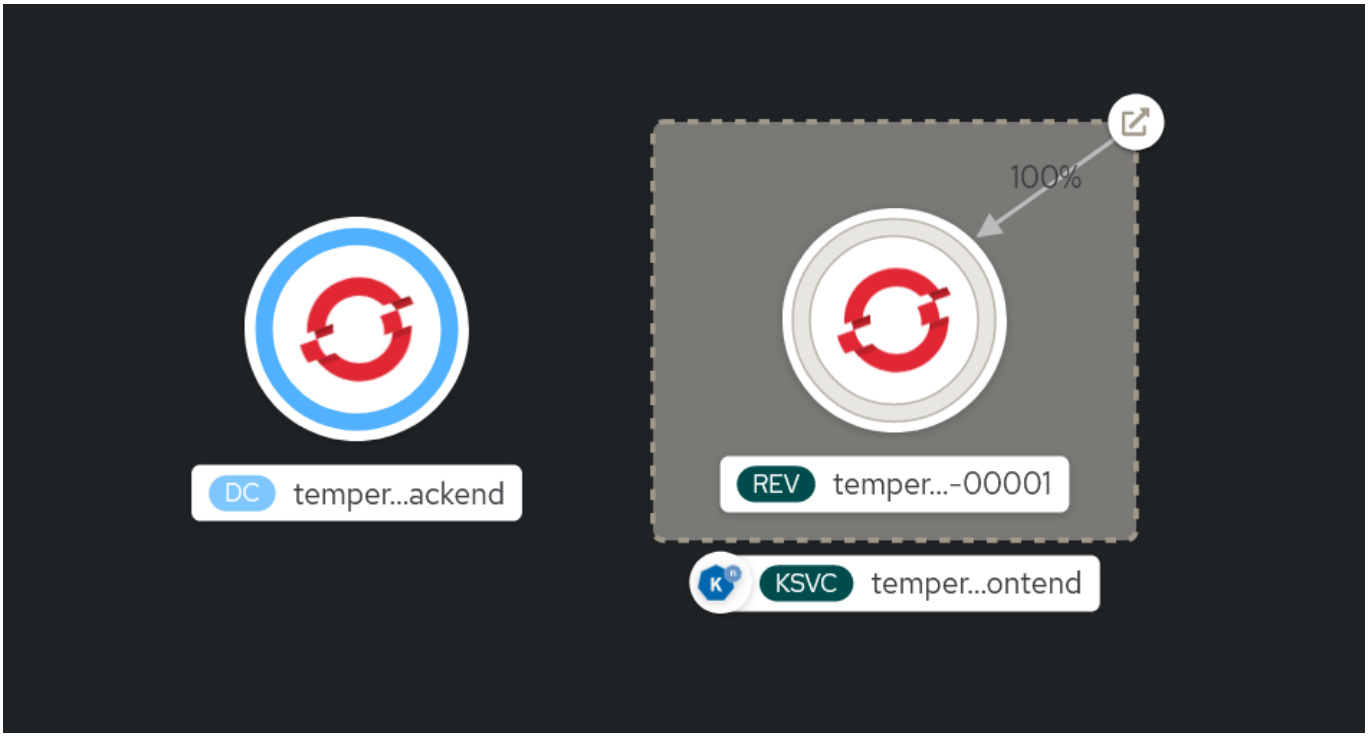
Reutilizaremos las imágenes de los contenedores que hemos creado en el apartado anterior.

Ahora vamos a desplegar la aplicación usando la consola web de RHOCP, para ello vamos a reutilizar los archivos YAML que hemos creado en el apartado anterior.

En mi caso para importar los archivos YAML he usado la opción de **"Import YAML"**.



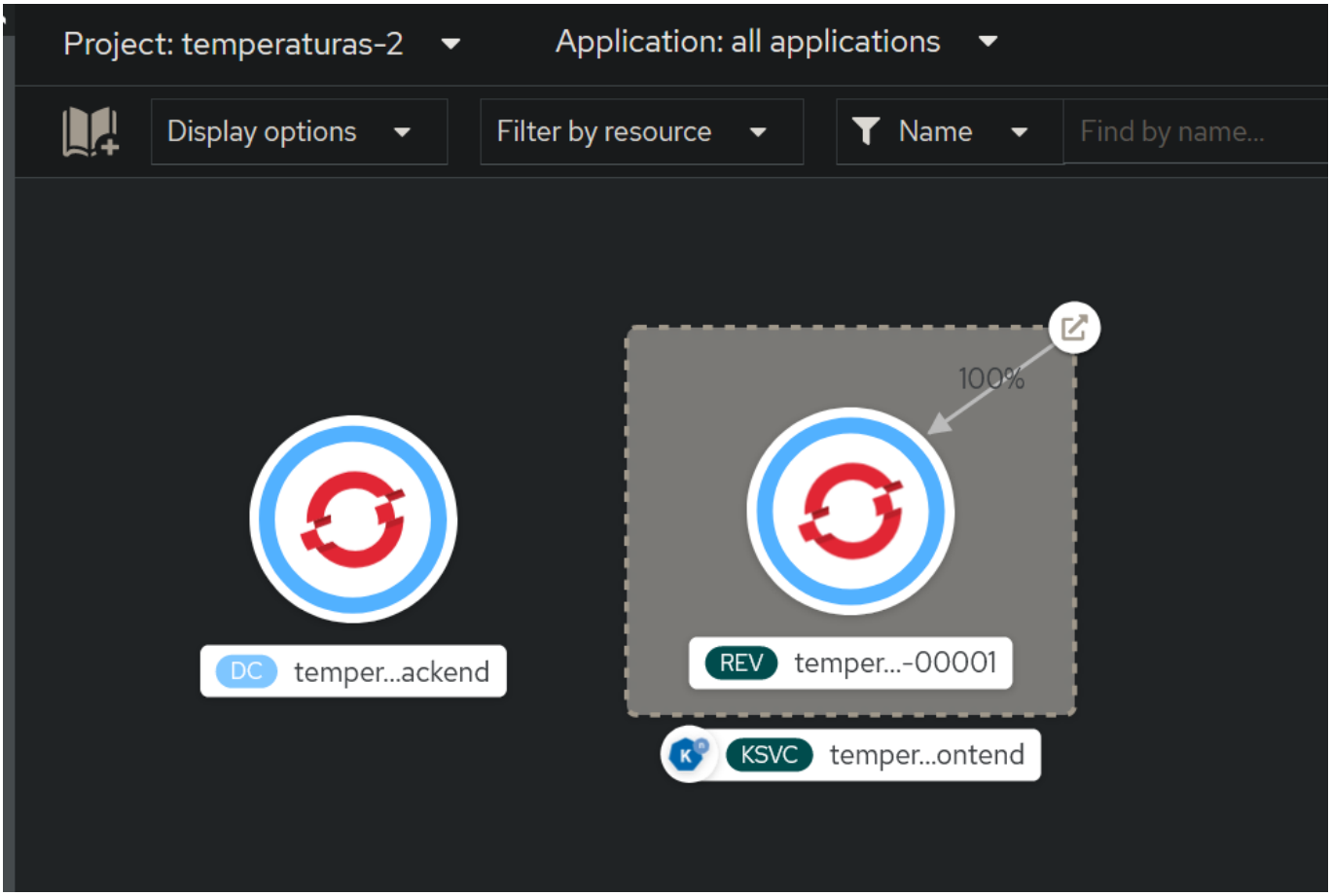
Comprobamos que al dirigirnos a Topology, se han creado los objetos que hemos definido en los archivos YAML.



Podemos apreciar que los pods del back-end están autoescalados a 1 y los del front-end a 0 debido a que aún no hemos accedido a la web.

Accederemos a la ruta del front-end y comprobaremos que la aplicación funciona correctamente.

Si volvemos a la consola web de RHOCP y nos dirigimos a Topology, podremos apreciar que los pods del front-end se han autoescalado.



Conclusión

Como conlusión del primer caso práctico, creo que trabajar con la interfaz web en este caso ha sido mucho más rápido y sencillo que trabajar con la CLI. Además, la interfaz web es muy intuitiva y visual, por lo que es bastante fácil de entender y de aprender a usar.

Como punto a favor de ambas formas de trabajar, la instalación del operador Serverless y Knative Serving ha sido muy sencilla en ambos casos. Lo único es que no debemos olvidar de instalar en la terminal el CLI kn, ya que si no no podremos trabajar con Knative Serving.

6.2. Gestionar Revisiones de Servicios, Controlar el Tráfico, Etiquetas y Autoscaling

Enunciado

Vamos a crear múltiples revisiones de la aplicación anterior y dividiremos el tráfico entre ellas usando la consola web de Red Hat OpenShift Container Platform (RHOCP) y la CLI kn. Además añadiremos etiquetas a las revisiones, cambiaremos el tiempo de vida de los pods, su escala máxima, su escala mínima y su concurrencia.

Resultados esperados de este caso práctico

Deberíamos ser capaces de realizar muchos escenarios de implementación dirigiendo el tráfico a las revisiones de la aplicación y manejando las etiquetas, el tiempo de vida de los pods, su escala máxima, su escala mínima y su concurrencia.

Pasos a seguir usando la CLI kn

Vamos a crear una nueva revisión de la aplicación, para ello cambiaremos el nombre de la aplicación con el siguiente comando:

```
kn service update temperaturas-frontend --revision-name
temperaturas-frontend-v2
```

Si ejecutamos el comando:

```
kn revision list
```

Podremos ver que se ha creado una nueva revisión de la aplicación.

```
NAME                          SERVICE               TRAFFIC   TAGS   GENERATION   AGE    CONDITIONS   READY   REASON
temperaturas-frontend-v2      temperaturas-frontend   100%            2          6m5s   3 OK / 4     True
temperaturas-frontend-00001   temperaturas-frontend                   1          108m   3 OK / 4     True
```

Y si ejecutamos el comando:

```
kn route describe temperaturas-frontend
```

Podremos ver que los detalles de la ruta de la nueva revisión.

```
Name:       temperaturas-frontend
Namespace:  temperaturas-1
Labels:     app=temperaturas, tier=frontend
Age:        1h
URL:        https://temperaturas-frontend-temperaturas-1.apps.cluster-drh75.drh75.sandbox2480.opentlc.com
Service:    temperaturas-frontend

Traffic Targets:
  100%  @latest (temperaturas-frontend-v2)

Conditions:
  OK TYPE                   AGE REASON
  ++ Ready                   8m
  ++ AllTrafficAssigned      1h
  ++ CertificateProvisioned  1h TLSNotEnabled
  ++ IngressReady            8m
```

No vamos a querer que el 100% del tráfico vaya a la nueva revisión (queremos una implementación Canary no Blue/Green), así que para ello usaremos el siguiente comando:

```
 kn service update temperaturas-frontend \
--traffic temperaturas-frontend-00001=75 \
--traffic temperaturas-frontend-v2=25
```

Ahora, el tráfico se dividirá entre la revisión original y la nueva revisión.

Si volvemos a ejecutar el comando:

```
kn route describe temperaturas-frontend
```

Podremos ver que el tráfico se ha dividido entre las dos revisiones.

```
Name:       temperaturas-frontend
Namespace:  temperaturas-1
Labels:     app=temperaturas, tier=frontend
Age:        1h
URL:        https://temperaturas-frontend-temperaturas-1.apps.cluster-drh75.drh75.sandbox2480.opentlc.com
Service:    temperaturas-frontend

Traffic Targets:
   75%  temperaturas-frontend-00001
   25%  temperaturas-frontend-v2

Conditions:
  OK TYPE                   AGE REASON
  ++ Ready                   5s
  ++ AllTrafficAssigned      1h
  ++ CertificateProvisioned  1h TLSNotEnabled
  ++ IngressReady            5s
```

Vamos a añadirle una etiqueta (TEST-VERSION) a la nueva revisión, para ello usaremos el siguiente comando:

```
kn service update temperaturas-frontend \
--tag temperaturas-frontend-v2='test-version'
```

Si ejecutamos el comando:

```
kn revision list
```

Podremos ver que la nueva revisión tiene una etiqueta.

```
NAME                          SERVICE                TRAFFIC   TAGS           GENERATION   AGE    CONDITIONS   READY   REASON
temperaturas-frontend-v2      temperaturas-frontend  25%       test-version   2            30m    3 OK / 4     True
temperaturas-frontend-00001   temperaturas-frontend  75%                      1            132m   3 OK / 4     True
```

Si accedemos a la aplicación web, podremos ver que el hostname cambia, esto es debido a la división del tráfico entre las dos revisiones.

Revisión 1:

Revisión 2:

Ya solo nos queda cambiar el tiempo de vida de los pods, su escala máxima, su escala mínima y su concurrencia, para ello usaremos el siguiente comando (he creado una nueva revisión de la aplicación también):

```
kn service update temperaturas-frontend \
--scale-window 10s \
--scale-min 1 \
--scale-max 5 \
--concurrency-limit 2 \
--scale-target 10 \
--revision-name temperaturas-frontend-v3 \
--traffic @latest=100 \
--tag temperaturas-frontend-v3='final-version'
```

Con esta configuración, los pods se escalarán entre 1 y 5, el tiempo de vida de los pods será de 10 segundos, la concurrencia máxima será de 2 y el número de pods objetivo será de 10. También hemos cambiado el tráfico para que vaya todo a la nueva revisión y le hemos añadido una etiqueta.

Si ejecutamos el comando:

```
kn service describe temperaturas-frontend
```

Podremos ver que se han cambiado los parámetros de la aplicación.

```
Name:       temperaturas-frontend
Namespace:  temperaturas-1
Labels:     app=temperaturas, tier=frontend
Age:        2h
URL:        https://temperaturas-frontend-temperaturas-1.apps.cluster-drh75.drh75.sandbox2480.opentlc.com

Revisions:
  100%  temperaturas-frontend-v3 (current @latest) #final-version [3] (15s)
        Image:     iesgn/temperaturas_frontend (at 7a2b7b)
        Replicas:  1/1
    +   temperaturas-frontend-v2 #test-version [2] (47m)
        Image:     iesgn/temperaturas_frontend (at 7a2b7b)
        Replicas:  0/0

Conditions:
  OK TYPE                  AGE REASON
  ++ Ready                 10s
  ++ ConfigurationsReady   10s
  ++ RoutesReady           10s
```

Y si accedemos a la aplicación web, podremos ver que el hostname ha cambiado:

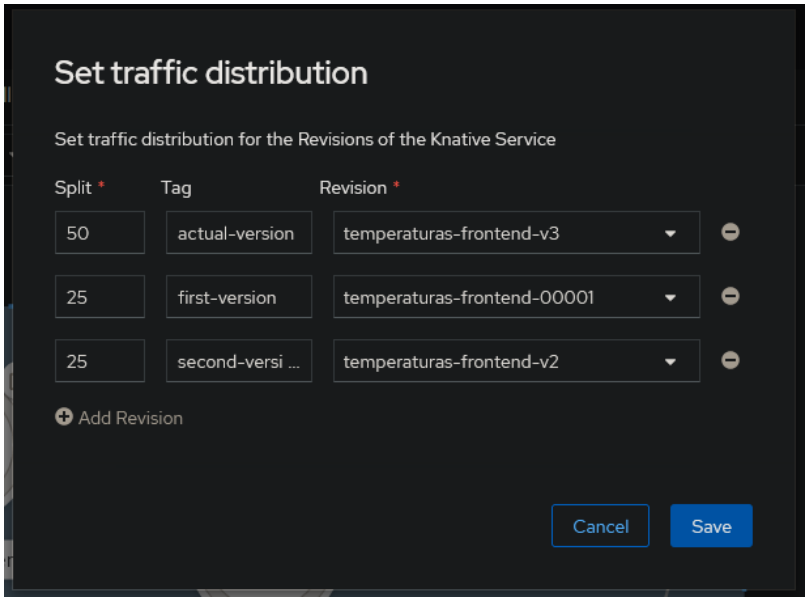
Pasos a seguir usando la consola web de RHOCP

Vamos a seguir trabajando con el mismo proyecto, temperaturas-1.

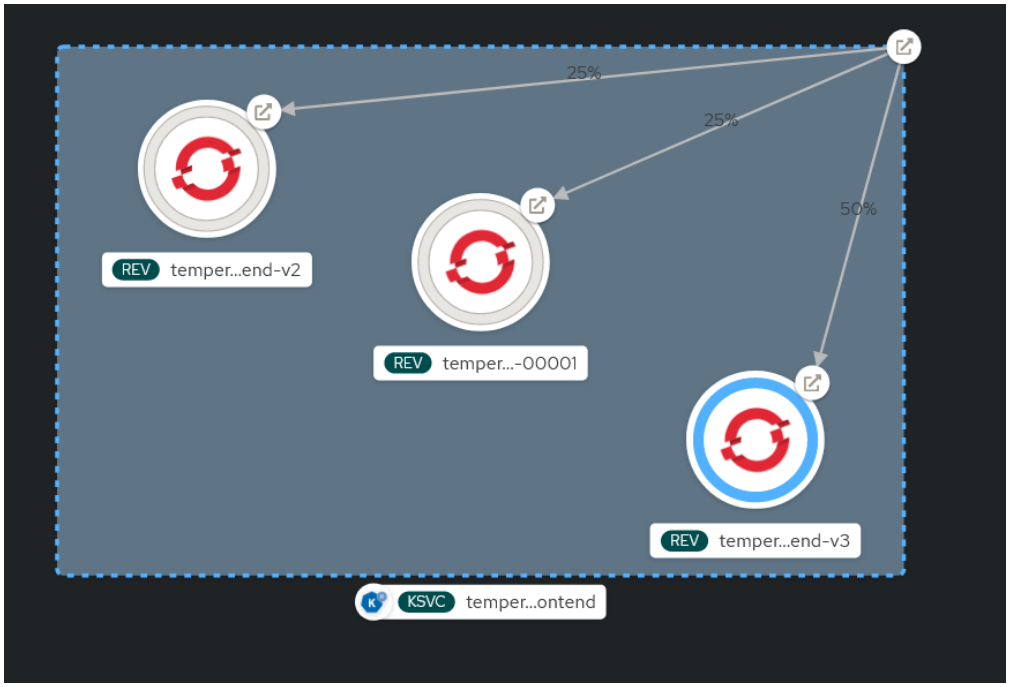
En este caso vamos a realizar algo parecido a lo que hemos hecho con la CLI kn, pero esta vez usando la consola web de RHOCP que es mucho más visual pero algo más limitada a mi parecer.

Nos dirigiremos a Topology y haremos click en el servicio temperaturas-frontend. Una vez dentro, haremos click en el botón Set Traffic.

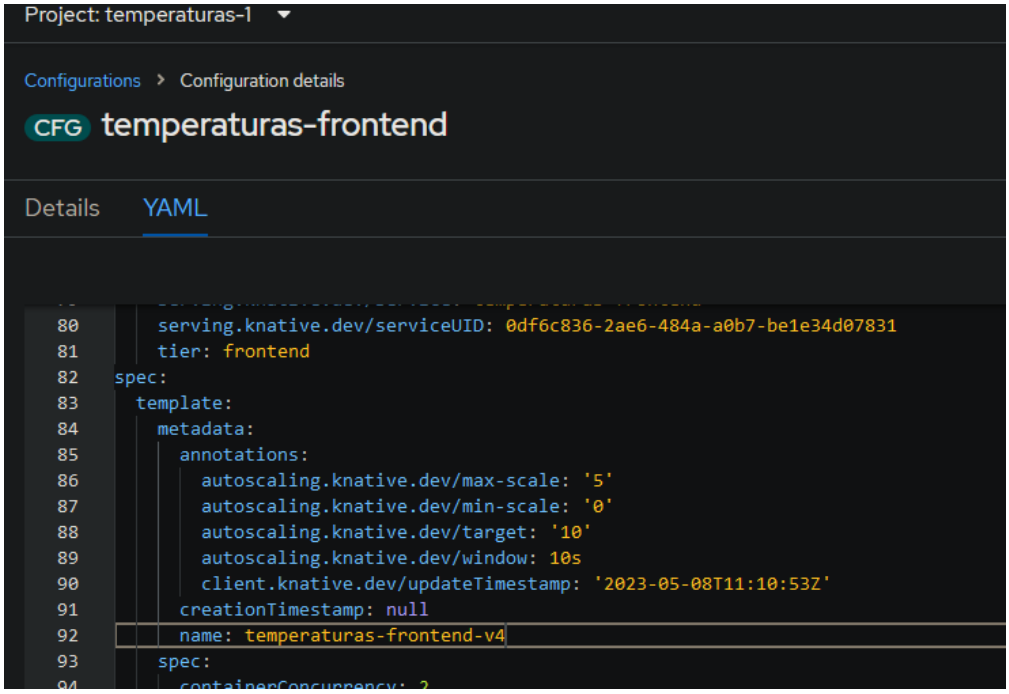
Como ya hemos visto anteriormente, podemos modificar la división del tráfico entre las revisiones de la aplicación. En mi caso he cambiado las etiquetas de las revisiones para que sea más fácil identificarlas y también el tráfico que va a cada revisión.



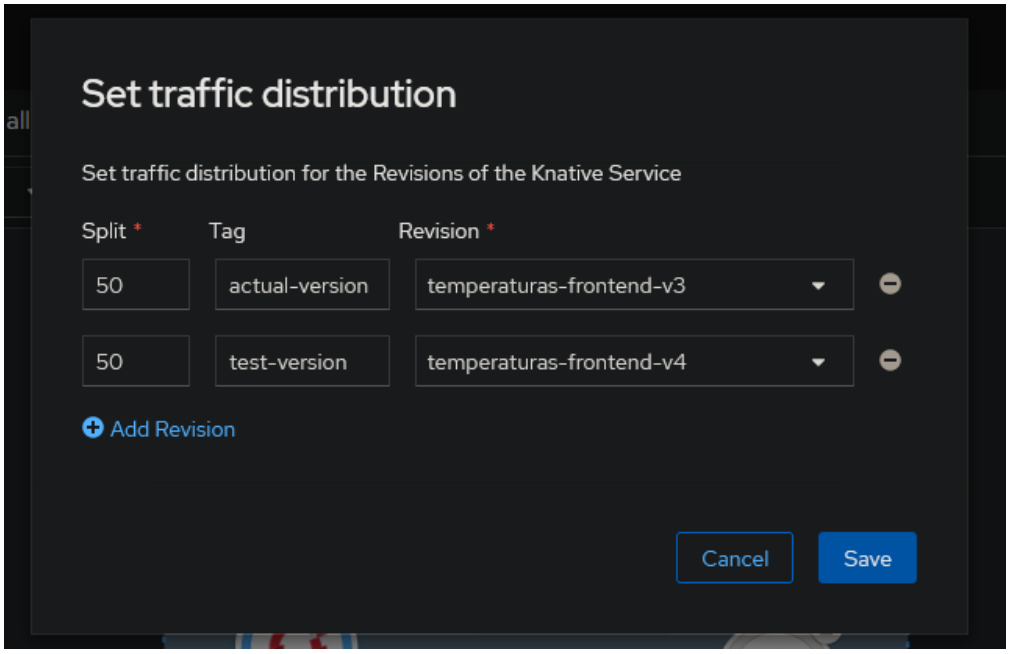
Veremos que nuestra topología cambiará, ya que ahora el tráfico se dividirá entre las tres revisiones y además podemos ver que la revisión 3 tiene un pod siempre escalado como configuramos anteriormente, mientras que las revisiones 1 y 2 no tienen ningún pod escalado.



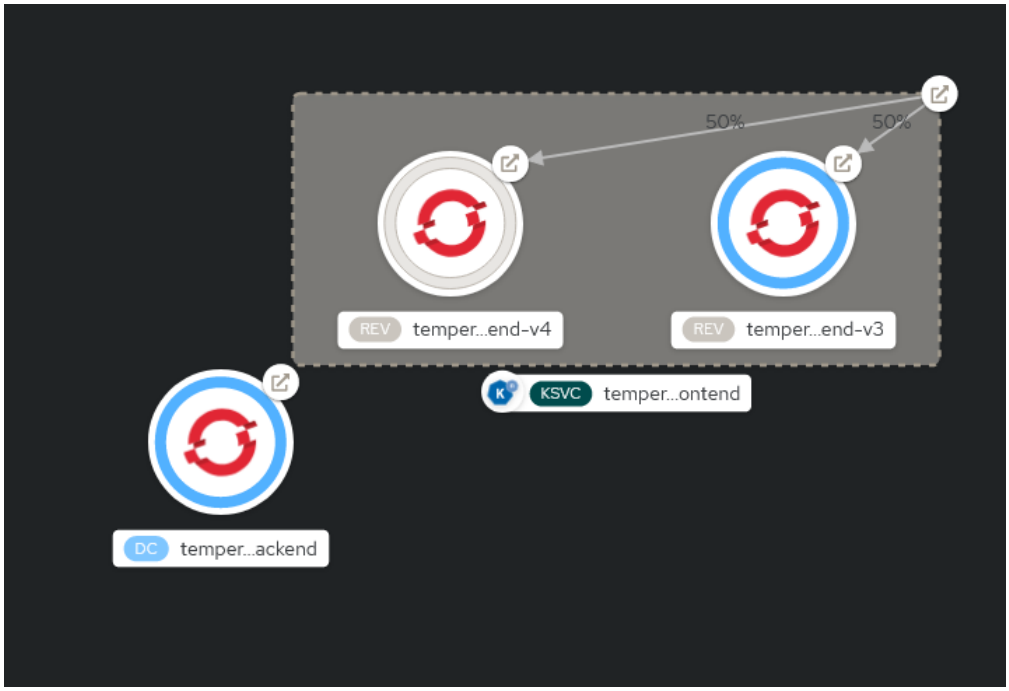
Podremos modifcar la configuración YAML de la cualquier revisión desde la interfaz web. Para ello nos dirigimos a la revisión que queramos en mi caso será la última. No dirigimos al apartado Configurations - YAML. En mi caso voy a modificar la escala mínima de 1 a 0. Una vez hecho esto, deberemos cambiar el nombre de la revisión, ya que no podemos modificar una revisión que ya existe. En mi caso la he llamado temperaturas-frontend-v4.



Luego nos dirigiremos de nuevo a Set Traffic y desde ahí modificaremos el tráfico para que se divida entra la nueva revisión y la última revisión.



Ahora tendremos una revisión con un pod escalado (aunque no reciba tráfico) y otra sin ningún pod escalado (al menos hasta que se reciba tráfico).



Conclusión

Como conclusión final de este caso práctico, he de decir que la CLI kn me ha parecido mucho más potente que la interfaz web de RHOCP, aunque la interfaz web es muy visual y fácil de usar. Al ya estar trabajando con muchas más opciones y parámetros, la CLI kn es mucho más cómoda y rápida de usar.

Un claro ejemplo es que con un simple comando en la CLI kn he creado una revisión nueva, le he cambiado el tráfico y le he añadido una etiqueta, además de muchas otras cosas. Sin embargo, para hacer lo mismo en la interfaz web he tenido que hacer varios pasos y he tenido que ir a varias secciones diferentes.

Lo único que rescataría de la interfaz web es la sección de Topology y Set Traffic, que son muy intuitivas y fáciles de usar.

7. Conclusiones

Como hemos visto, Openshift Serverless es una plataforma que nos permite desplegar aplicaciones sin preocuparnos por la infraestructura. Además, nos permite escalar automáticamente nuestras aplicaciones en función de la demanda de los usuarios, y nos permite desplegar nuestras aplicaciones en cualquier nube pública o privada. A mi parecer es una plataforma muy potente y muy fácil de usar, y que nos permite desplegar aplicaciones de una manera muy sencilla y rápida.

Tal vez para un administrador de sistemas no sea tan útil, ya que no tiene que preocuparse por la infraestructura; pero para un desarrollador sí que lo es, ya que le permite desplegar sus aplicaciones de una manera muy sencilla y rápida, no necesitando tener conocimientos de infraestructura o de administración de sistemas. Por no hablar de lo que a mi parecer es la principal ventaja, no pagar por recursos que no se están usando.

En cuanto a Knative Serving, es la herramienta que nos ha permitido desplegar nuestra aplicación, sin embargo, no es la única herramienta que nos ofrece Knative. Knative también nos ofrece Knative Eventing, que nos permite crear eventos y reaccionar a ellos.

Aunque este último se ha quedado fuera del proyecto por motivos obvios, ya que el proyecto entonces hubiera sido demasiado largo y complejo, me parece importante recalcar que con el uso de esto hubieramos podido incluso convertir el backend en serverless o por ejemplo crear un sistema de notificaciones que nos avisara cuando la temperatura de una ciudad superarase un cierto umbral.

También se ha dejado fuera del proyecto el uso de las Serverless Functions, que nos facilita plantillas para crear funciones serverless en distintos lenguajes de programación. Estas plantillas proporcionan al desarrollador los archivos y dependencias necesarios, y un código inicial vacío para una función. Finalmente, Serverless Functions acabaría construyendo una imagen de ejecución y tras esto instantáneamente OpenShift Serverless se encargaría de subir, desplegar y gestionar las funciones como cualquier otra aplicación.

Esto último a mi parecer sería útil principalmente para un desarrollador, no le veo el sentido a un administrador usando Serverless Functions.

Con todo esto que he explicado, lo que quiero dar a entender la gran cantidad de funcionalidades y opciones que nos proporciona Openshift Serverless y Knative, y que algunas de ellas muy importantes (como Knative Eventing) no se han podido ver en este proyecto.

8. Bibliografía

Introducción a Openshift Serverless por José Domingo Muñoz Rodríguez

RedHat Curso DO244: Developing Applications with Red Hat OpenShift Serverless and Knative

Documentación Oficial de RedHat sobre Openshift Serverless y Knative