Whitestack challenge #1

Abril 2024

devsecdavid@gmail.com

ÍNDICE

Compilar y probar el código de manera local

Analizando el código se puede resumir que la aplicación realiza lo siguiente:
web app para jugar tetris con dos modalidades de juego solo o duo(competición entre dos jugadores).
Cuenta con un scoreboard donde se registran los resultados más altos en el juego en una base de datos sqlite.
Se guardan el nombre y el score de los respectivos jugadores (entre otros campos).

Análisis técnico de la web app

La aplicación es una webapp monolítica; es decir, tanto el frontend como el backend son servidos por el backend, que utiliza el framework Flask de Python, junto con una base de datos SQLite. Para habilitar la funcionalidad en tiempo real, se utiliza Socket.IO, lo que permite actualizar en tiempo real los puntajes y el modo de juego en duo.

Requerimientos

Se requiere actualizar la aplicación que previamente fue desplegada en el cluster, garantizando una mínima interrupción de su funcionamiento.
Este requisito se puede gestionar eficazmente mediante el uso de un recurso Deployment en Kubernetes.
Un Deployment es responsable de gestionar la creación y actualización de pods.
Al emplear un Deployment, se puede especificar la versión de la imagen del contenedor deseada. Kubernetes entonces se encargará de actualizar los pods de manera controlada, asegurando un flujo óptimo del tráfico y minimizando los tiempos de inactividad

Limitaciones de SQLite en Kubernetes

El uso de SQLite como base de datos en nuestra webapp conlleva limitaciones significativas en contextos que requieren escalabilidad. Aunque SQLite es capaz de manejar múltiples conexiones concurrentes, su capacidad para enfrentar grandes volúmenes de trabajo y escalar horizontalmente es restringida

Esto es particularmente limitante en un entorno como Kubernetes, diseñado para facilitar el escalado horizontal. Para aprovechar plenamente estas capacidades, será necesario migrar a un sistema de gestión de bases de datos más robusto y escalable, como NuoDB, Postgres, MySQL, o alternativas NoSQL como Cassandra o MongoDB.
Estos sistemas están mejor equipados para satisfacer las demandas de aplicaciones que requieren alta disponibilidad y escalabilidad.

Debido a esta limitación con SQLite, utilizaremos un deployment en Kubernetes con solo una réplica, pues el escalado horizontal no es factible con nuestra configuración actual de la base de datos.

Creación imagen docker

Para la creación de la imagen docker se pueden destacar los siguientes puntos:

- imagen alpine: basada en seguridad y buenas prácticas
- se crea un user independiente que corre la app y no root por defecto
- se crea un archivo requirements.txt donde se mantienen las dependencias del proyecto, con esto quedan de una forma centralizada.
- se agregan las versiones exactas de las dependencia del proyecto:En desarrollo de software es una buena práctica explicitar la versión de los paquetes que se están utilizando, ya que si no se especifica, al momento de hacer un despliegue, siempre se estará intentado buscar la versión más reciente, lo que puede abrir una brecha de error, ya que si la nueva versión de la librería sube a una versión mayor (x.0.3), pueden cambiar ciertas llamadas a la librería y por consecuencia generar errores en nuestra app.
- Caché de pip: Eliminar el caché de pip (/root/.cache/pip) ayuda a reducir el tamaño de la imagen final.
- para el tag de la imagen se ocupará semantic-release, para poder aprovechar las ventajas del rollback de versiones, ej: 1.0.0

Recursos kubernetes a utilizar

Recursos kubernetes

- configmap
- deployment
- service
- ingress

para actualizar una variable de entorno en la webapp

Algunas ventajas de configurar nuestras variables en configmap:
Los ConfigMaps en Kubernetes ofrecen una centralización efectiva al permitir que la configuración de las variables de entorno se almacene en un único lugar.
Esto simplifica la gestión y el mantenimiento, ya que las actualizaciones se pueden realizar sin necesidad de reconstruir las imágenes de los contenedores.
Además, un ConfigMap puede ser reutilizado en varios pods y entornos, facilitando la implementación de configuraciones consistentes a lo largo de diferentes fases como desarrollo, pruebas y producción.
Al separar la configuración del código, se mejora la portabilidad y escalabilidad de las aplicaciones, permitiendo que los desarrolladores se enfoquen en el código sin preocuparse por la configuración específica del entorno.
Finalmente, los ConfigMaps ofrecen la flexibilidad de actualizar configuraciones sin necesidad de reiniciar los contenedores, minimizando así la interrupción en el funcionamiento del sistema, en línea con requisitos de actualización garantizando una mínima interrupción de su funcionamiento..

instalación y actualización web app kubernetes vía helm chart

Instalación webapp

primero hacemos un build para crear nuestra imagen docker
`docker build -t tengen-tetris .`

agregamos un tag con el id de la imagen previamente creada
`docker tag <imagen_id>davidl21/tengen-tetris:0.0.1`

se realiza push al docker hub público de la imagen
`docker push davidl21/tengen-tetris:0.0.1`

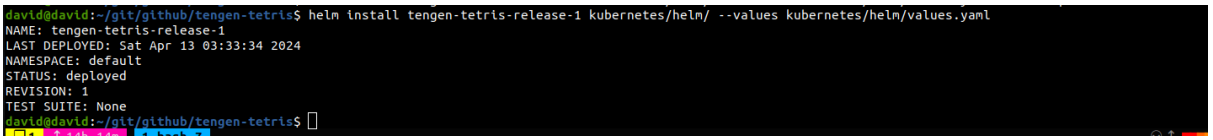
Ahora que contamos con nuestra imagen docker debemos editar nuestro archivo 'kubernetes/helm/values.yaml con los valores correspondientes y aplicar el siguiente comando:

`helm install release-1 kubernetes/helm/ --values kubernetes/helm/values.yaml --namespace mynamespace`

o mediante make

`make helm-install RELEASE=tengen-tetris-release-n NAMESPACE=default`

Con esto ya habremos desplegado nuestra webapp vía helm chart.

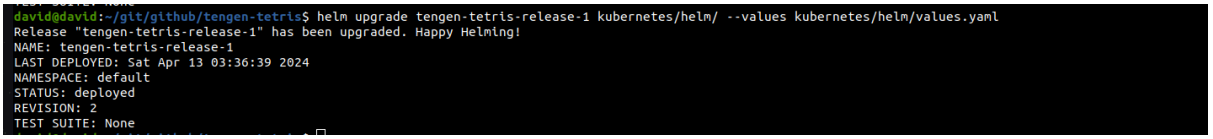


Actualización webapp

Para la actualización se deben ajustar los valores de helm correspondientes de nuestro archivo kubernetes/helm/values.yaml y adicionalmente updatear la versiones de version/appVersion en el archivo kubernetes/helm/Chart.yaml

luego aplicamos el siguiente comando:
`helm upgrade  release-1 kubernetes/helm/ --values kubernetes/helm/values.yaml --namespace mynamespace`

o mediante make
`make helm-upgrade RELEASE=tengen-tetris-release-n NAMESPACE=default`



Archivo values.schema.json

Cuando se trabaja con Helm para desplegar aplicaciones en Kubernetes, es importante gestionar y validar las configuraciones de manera efectiva y segura. Para lograr esto, una práctica recomendada es utilizar un archivo values.schema.json.
Este archivo contiene un esquema JSON que define la estructura esperada y las restricciones de los valores que se pueden proporcionar en el archivo values.yaml de un chart de Helm.

ejemplo al intentar pasar un valor de puerto como string en lugar de integer: