Practica Kubernetes

# Objectivo

El objetivo de esta práctica es montar en un entorno local de Docker compose y en un entorno de Kubernetes, 2 versiones de una aplicación que lista los coches de una lista en mongo db a través. La solución tiene 2 imágenes, una de base de datos (mongodb) y otra donde hay que compilar una solución web que consulta la base de datos y expone esa información en una api consultable desde un navegador. En la api se expone la versión de la solución, un valor del entorno de la aplicación web y el listado de coches de la base de datos.

# Overview Solucion

La solución se implementó siguiendo los siguientes pasos, que se detallarán a continuación:

- Ajuste de archivos Dockerfile

- Ajuste de archivos \Controllers\PracticaController.cs

- Creación de archivos de configuración yaml

- Creación de archivos .dockerignore

- Build y run

En general, la solución diseñada permite la implementación de las 2 aplicaciones en el clúster de Kubernetes, que se ejecutan dentro de los pods correspondientes (que se encargan de la ejecución de las aplicaciones (presentes en ellos) que estan dentro de sus contenedores).

A través de los archivos de configuración .yaml se han definido en detalle las especificaciones para los objectos de deployments , el servicio para permitir la comunicación entre ellos y un configmap para inicializar el contenedor mongo

## Archivos entregado

*practica\_kubernetes*: Carpeta que contiene definicion del entorno de Kubernetes y local (los YAMLs estan dentro de la carpetas k8s)

# Solucion

## Step 1: Ajuste file Dockerfile

El comando ARG usa valores disponibles en la fase de compilación y no después de que se crea la imagen, asigno a través de ARG la variable "Entorno" resaltada con mi nombre:

1. ARG Entorno="Giovanni Quattrocchi"

Utilizo el comando ENV para exponer la variable anterior configurándola como variable de entorno del contenedor de tal manera que esté disponible una vez creada la imagen:

1. ENV Entorno=${Entorno}

Asigno a la variable global MONGOHOST el nombre del servicio que aloja el contenedor mongodb, esto para permitir que el contenedor webapp pueda conectarse directamente al servicio mongo

1. ENV MONGOHOST=database-mongo

## Step 2: Ajuste file \Controllers\PracticaController.cs

Cambio la línea asignando el valor de la variable de entorno Version con la variable de clase “result” para que pueda devolver la versión configurada:

1. Version = Environment.GetEnvironmentVariable("Version") ?? "1"

## Step 3: Creacion file .dockerignore

Creo que el archivo . dockerignore che serve per escludere file (Dockerfile) o directory (k8s) al momento della esecuzione della build del DockerFile.

## Step 4: build y publicación de la imagen web

Para poder utilizar, a través de kubernetes, publico el repositorio de la imagen.

Ejecuto el build del Dockerfile:

1. docker image build . -t appdotnet

Asigno correctamente una etiqueta de publicación:

1. docker tag appdotnet giovanniq/appdotnet

Realizo el pull para publicar la imagen:

1. docker push giovanniq/appdotnet

## Step 5: Creación de un StorageClass, PV y PVClaims

Para que nuestra app de mongo emplee volume persistente, creo un objeto de tipo StorageClass para definir una clase ad hoc para la práctica estableciendo el provisoner interno de minikube, septo VolumeBindingMode a “Immediate” para permitir que el binding de volumen ocurra una vez que se haya creado PVClaim.

Uso un volumen persistente “small-pv” para definir el directorio "/data/mongo/" en el clúster de minikube para permitir (proporcionando 1 giga de espacio) que pueda ser montado en el contenedor mongo en la carpeta donde se almacenan archivos de datos .

Entonces, una claim de volumen persistente “nodered-claim” se utiliza para definir una "solicitud" (de 1 giga de espacio) de espacio de volumen definido arriba que un contenedor específico puede hacer.

## Step 6: Creación de un Configmap

Creo un archivo de configuración de un objeto tipo ConfigMap que me permite inicializar automáticamente la base de datos una vez implementado e iniciar el pod asociado con mongo db:

1. apiVersion: v1
2. data:
3. mongo-init.sh: "mongosh testdb <<EOF\r\ndb.cars.insert({name: \"Audi\", price: 52642});\r\ndb.cars.insert({name:
4. \"Mercedes\", price: 57127});\r\ndb.cars.insert({name: \"Skoda\", price: 9000});\r\ndb.cars.insert({name:
5. \"Volvo\", price: 29000});\r\ndb.cars.insert({name: \"Bentley\", price: 350000});\r\ndb.cars.insert({name:
6. \"Citroen\", price: 21000});\r\ndb.cars.insert({name: \"Hummer\", price: 41400});\r\ndb.cars.insert({name:
7. \"Volkswagen\", price: 21600});\r\nEOF"
8. kind: ConfigMap
9. metadata:
10. name: mongo-initdb

**NOTA**: Soy consciente de que el uso de este objeto en particular está indicado para historizar información de tipo no confidencial (sería más apropiado explotarlo para operaciones de configuración en lugar de ejecutar inserts en la tabla) la mía fue una decisión dictada por querer poner en práctica el uso de este tipo de objeto con el fin de automatizar el proceso.

Una posible alternativa de inicialización sería entrar dentro del pod que contiene el contenedor mongo de db y ejecutar manualmente los comandos:

1. kubect execl -i -t mongo-56ddbb9dc8-7nstq -- /bin/bash
3. mongo testdb
4. db.cars.insert({name: "Audi", price: 52642})
5. db.cars.insert({name: "Mercedes", price: 57127})
6. db.cars.insert({name: "Skoda", price: 9000})
7. db.cars.insert({name: "Volvo", price: 29000})
8. db.cars.insert({name: "Bentley", price: 350000})
9. db.cars.insert({name: "Citroen", price: 21000})
10. db.cars.insert({name: "Hummer", price: 41400})
11. db.cars.insert({name: "Volkswagen", price: 21600})

Una segunda alternativa es crear el archivo con los comandos de inicialización dentro y a través del archivo de configuración de implementación del contenedor de mongo db montar un volumen especificando el archivo local creado, A continuación, se puede ejecutar el archivo montado mediante el acceso al contenedor.

## Step 7: Creación de un objeto de Deployment para mongo db

Creo un archivo de configuración de un objeto Deployment, me permite crear un pod (con una sola réplica) que gestiona la ejecución de una imagen pública descargada desde el docker hub del repositorio oficial de mongo (descarga la última versión).

Monto un volumen (name: mongo-initdb) que apunta a la carpeta docker-entrypoint-initdb.d, asociándolo con el nombre del ConfigMap que contiene el script de inicialización del db. De esta manera, durante el arranque, todo lo que está presente en docker-entrypoint-initdb.d.

Monto un volumen (name: data-storage) que apunta a la carpeta “/data/db” (aquì se quedan los archivos de datos de mongo, asociándolo con el nombre del PVC que lo utilizarà (“nodered-claim”)

Septo el puerto por defecto del contenedor donde aceptará desde el exterior las comunicaciones.

Configuro una limitación de recursos, para la memoria he configurado un limite hard desde el que no se puede ir más allá y un soft limit (reservation). De esta manera, el contenedor tratará de mantener la asignación de memoria por debajo del límite blando impuesto y, en caso de exceso, aumentará, pero solo hasta el límite rígido impuesto.

## Step 8: Creación de un objeto de Deployment para la web application

Creo un archivo de configuración de un objeto Deployment, me permite crear un pod (con 2 replicas, luego con dos instancias de la aplicación que se inician para tener más disponibilidad en caso de que uno de los dos se desconecte) que gestiona la ejecución de la imagen previamente publicada de la aplicación web.

Configuro una limitación de recursos, para la memoria he configurado un limite hard desde el que no se puede ir más allá y un soft limit (reservation). De esta manera, el contenedor tratará de mantener la asignación de memoria por debajo del límite blando impuesto y, en caso de exceso, aumentará, pero solo hasta el límite rígido impuesto.

Septo los puertos disponibles del contenedor (7007 para https y 7006 para http) donde aceptará desde el exterior las comunicaciones.

Septo la variable de entorno "Versión" asignando el valor 2, que será utilizado por la aplicación web en su interior.

## Step 9: Creación de servicio para mongo y web app

Creo dos archivos de configuración de tipo Service para habilitar la red de acceso para las dos aplicaciones de modo que el servicio pueda enrutar las solicitudes de red a los pods seleccionándolas en función de la etiqueta ("selector.app" name).

He creado los Servicios de tipo NodePort para que estén disponibles para solicitudes de red desde el exterior.

Específicamente para el pod de mongo, el contenedor escuchará en el puerto por defecto 27017 y expondrá internamente en el clúster en el mismo puerto, mientras que será accesible desde el exterior a través del puerto 30432.

Mientras que para el pod de la aplicación web, dado que los puertos expuestos son 2 (http,https):

* https: el contenedor escuchará en el puerto 7007 y expondrá internamente en el clúster en el mismo puerto, mientras que será accesible desde el exterior a través del puerto 30001.
* http: el contenedor escuchará en el puerto 7006 y expondrá internamente en el clúster en el mismo puerto, mientras que será accesible desde el exterior a través del puerto 30000.

## Step 10: Build y run

Inicio minikube:

1. minikube start

Con el archivo kustomization tengo la posibilidad de listar todos mis archivos de configuración yaml y tener la posibilidad de compilar objetos en la lista con un solo comando.

He creado el archivo kustomization:

1. kind: Kustomization
3. resources:
4. - db/mongo\_storageclass.yaml
5. - db/mongo\_pvc.yaml
6. - db/mongo\_pv.yaml
7. - db/mongo\_configmap.yaml
8. - db/mongo\_deploy.yaml
9. - db/mongo\_service.yaml
10. - web/appdotnet\_deploy.yaml
11. - web/appdotnet\_service.yaml

Lo he salvado como k8s/kustomization.yaml (dentro la carpeta k8s hay todos lo yaml repartidos por applicacion) y lo lanzo con el siguiente comando:

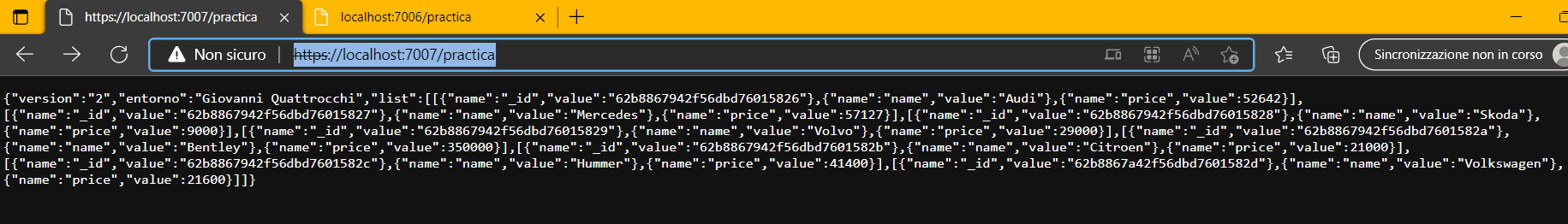
1. kubectl apply -k k8s

Verificación del resultado

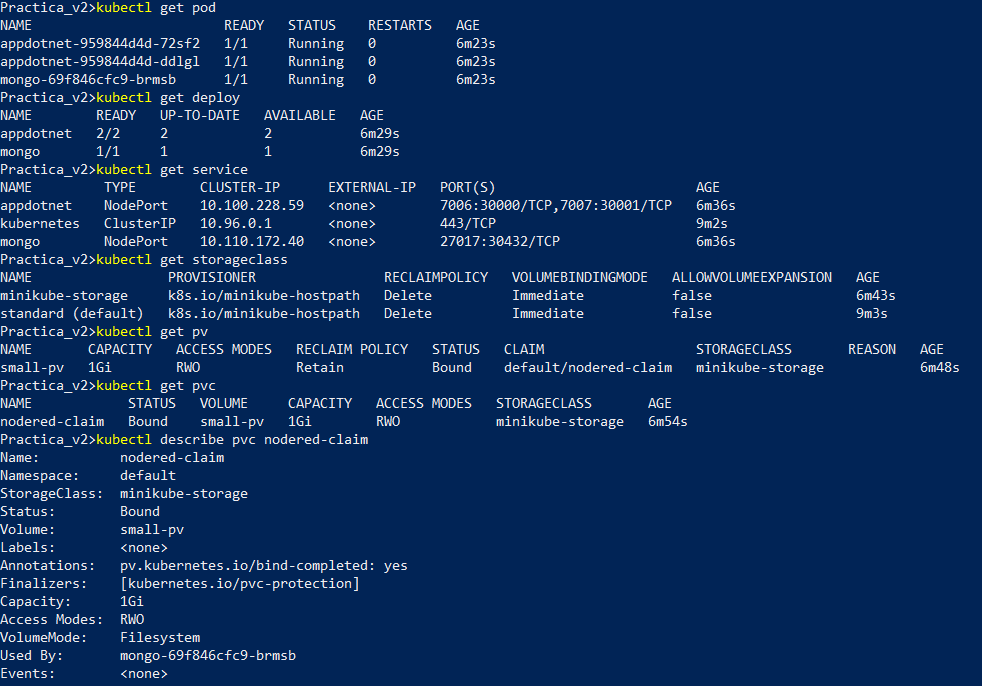
Para acceder necesitaríamos hacer un port forward a través del servicio de la applicacion web utilizando el nombre asignado “https” o “http”:

1. kubectl port-forward svc/appdotnet https
2. kubectl port-forward svc/appdotnet http

Desde el borwser por <https://localhost:7007/practica> (o <http://localhost:7006/practica> si se executa el port forward por “http”) serà possible ver el output:



Obtengo las listas de recursos para comprobar:



Compruebo si efectivamente hay el bind del volumen persistente “small-pv” desde de el PVC “nodered-claim” se haya ocurrido sin problema y que el volumen sea de modo corecto montado en la carpeta “/data/mongo” de minikube:

