Universidad del País Vasco
Euskal Herriko Unibertsitatea

Administración de Sistemas

Ingeniería Informática de Gestión y Sistemas de Información

Coches.xyz

Autor:
Xabier Gabiña Barañano

3 de diciembre de 2023

Índice general

1. Introducción **2**

2. Descripción de la aplicación **3**

3. Listado de las tareas realizadas **4**

4. Explicaciones de las tareas realizadas **5**
4.1. Desarrollo de una aplicación web. 5
4.2. Creación de una imagen Docker propia. 6
4.3. Creación de un entorno Docker Compose. 7
4.4. Creación de un despliegue Kubernetes equivalente. 8
4.5. Inclusión de imágenes adicionales. 9
4.6. Utilización de funcionalidades Docker/Kubernetes no vistas en clase. . 10

5. Declaración sobre uso de asistentes virtuales **11**

6. Bibliografia **12**

Introducción

Esta tarea tiene como objetivo explorar la implementación de una aplicación web con una base de datos incorporada, así como su despliegue utilizando Docker y Kubernetes. Estos dos entornos tecnológicos ofrecen soluciones fundamentales para abordar los desafíos actuales en el desarrollo de aplicaciones web al proporcionar un conjunto de herramientas y prácticas que simplifican la orquestación, escalabilidad y seguridad de dichas aplicaciones.

Docker, como plataforma de contenedores, permite a los desarrolladores encapsular aplicaciones y sus dependencias en entornos aislados. Esta metodología de encapsulamiento elimina problemas de compatibilidad, asegura la consistencia y facilita la migración de aplicaciones entre diferentes entornos. Además, Docker posibilita una gestión eficiente de recursos, contribuyendo así al despliegue efectivo de aplicaciones web.

Por otro lado, Kubernetes emerge como la solución definitiva para la orquestación de contenedores en clústeres. Desarrollada por Google, esta plataforma simplifica la administración y escalabilidad de aplicaciones web a gran escala. Kubernetes automatiza la distribución de contenedores, garantizando su disponibilidad, escalabilidad y equilibrio de carga, aspectos esenciales para aplicaciones que deben manejar un alto volumen de tráfico o que requieren una alta disponibilidad.

La combinación de Docker y Kubernetes se ha vuelto esencial para implementar aplicaciones web modernas, proporcionando una base sólida para el desarrollo, despliegue y gestión de sistemas en un entorno altamente dinámico y desafiante. Este enfoque promete optimizar los recursos, reducir los tiempos de despliegue y asegurar la confiabilidad y seguridad de las aplicaciones en un mundo cada vez más centrado en la nube.

Este informe se sumergirá en la creación de una aplicación web con una base de datos y su posterior despliegue mediante Docker y Kubernetes.

GitHub del proyecto

Descripción de la aplicación

El proyecto consiste en un entorno web enfocado a la compra y venta de coches de segunda mano. Una vez accedas a la página web podrás ver un listado de vehículos en venta, con los datos del vehículo y del vendedor para poder comunicarte con él. Además, podrás crearte una cuenta e iniciar sesión para poder publicar tú mismo un vehículo y ponerlo a la venta.

Para almacenar tantos los usuarios como los anuncios que estos generan se ha implementado una base de datos PostgreSQL.

Dado que esto es un proyecto y por lo tanto no existen otros usuarios, se ha creado un programa en Python que cada minuto genera un anuncio en la web. Para ello el programa se comunica con una API de modelos de coches y obtiene un modelo aleatorio con el que creara un anuncio.

Como añadido, para facilitar el desarrollo y mantenimiento de la aplicación se ha agregado un panel de administración para la base de datos. En este caso se ha elegido Adminer, una herramienta web que permite gestionar diferentes sistemas de gestión de bases de datos entre los que se incluye PostgreSQL.

Para acabar, se ha implementado un gestor de claves llamado Vault. Este gestor nos permite almacenar de forma segura las claves de nuestro proyecto sin la necesidad de tenerlas escritas explícitamente en el código fuente de nuestra aplicación.

Listado de las tareas realizadas

- Tareas obligatorias:
  - Desarrollo de una aplicación web. ✓
  - Creación de una imagen Docker propia. ✓
  - Creación de un entorno Docker Compose. ✓
- Tareas opcionales:
  - Creación de un despliegue Kubernetes equivalente. ✓
  - Inclusión de imágenes adicionales. ✓
  - Utilización de funcionalidades Docker/Kubernetes no vistas en clase. ✓

Explicaciones de las tareas realizadas

4.1. Desarrollo de una aplicación web.

El desarrollo de la aplicación web se puede dividir en tres partes, el backend, el frontend y la base de datos. El frontend se ha desarrollado utilizando HTML y CSS. Para facilitar el desarrollo se ha utilizado Bootstrap, que nos permite crear páginas web responsive de forma sencilla. El backend se ha desarrollado utilizando PHP. PHP se utiliza para gestionar las conexiones de la base de datos como puede ser el registrar o inicio de sesión de un usuario, la creación de anuncios y la obtención y muestra de los datos de los anuncios. Para almacenar la información de la web he usado PostgreSQL. PostgreSQL es un sistema de gestión de bases de datos relacional orientado a objetos y de código abierto.

Todo esto he creado dos imágenes Docker mediante archivos Dockerfile, uno para el frontend y el backend y otro para la base de datos.

La primera imagen se ha creado a partir de la imagen oficial de PHP y Apache. Para ello he instalado las dependencias necesarias para el funcionamiento de la aplicación y la comunicación con la base de datos y también he copiado todos los archivos necesarios para ello con los permisos correctos. Esta aplicación está disponible en el puerto 80.

La segunda imagen se ha creado a partir de la imagen oficial de PostgreSQL. En este caso lo único que se ha hecho es copiar un archivo .sql con la estructura de la base de datos y con los datos de prueba. Esta base de datos está disponible en el puerto 5432. También se han definido las variables de entorno necesarias para la creación de la base de datos y el usuario. Para acabar he desactivado la cuenta root como medida de seguridad.

4.2. Creación de una imagen Docker propia.

Dado que la página web es un proyecto y por lo tanto no existen otros usuarios, he decidido crear un programa que cada minuto genera un anuncio en la web. Para la creación de la imagen he utilizado la imagen oficial de Python Slim. He elegido esta imagen dado el reducido tamaño de esta. Para la creación de la imagen he copiado el archivo .py con el programa y he instalado las dependencias necesarias para su funcionamiento. El funcionamiento del programa es el siguiente:

1. El programa obtiene las claves necesarias para la comunicación de la API de Vault.
2. Con las claves obtenidas se identifica con la API de carapi.
3. Se obtiene un modelo aleatorio de coche.
4. Se buscas la marca a la que pertenece el modelo.
5. Se escribe el anuncio directamente en la base de datos.
6. Se espera un minuto y se repite el proceso.

De esta forma se generan anuncios de forma automática y se simula el funcionamiento de la web manteniendo además la seguridad gracias a no dejar implícitas las claves.

4.3. Creación de un entorno Docker Compose.

Para orquestar la creación de las imágenes y el levantamiento de los contenedores he utilizado Docker Compose. Docker Compose es una herramienta que nos permite definir y ejecutar aplicaciones Docker de múltiples contenedores y que se comuniquen entre ellos. Para ello he creado un archivo docker-compose.yaml en el que he definido los servicios necesarios con sus respectivas imágenes, puertos y volúmenes. El comando para levantar los contenedores y crear las imágenes todo a la vez es

```
docker compose up -d --build
```

El listado de puertos que utiliza la aplicación es el siguiente:

- 80: Puerto de la web.
- 5432: Puerto de la base de datos.
- 8080: Puerto del panel de administración de la base de datos.
- 8200: Puerto de la API de Vault.

4.4. Creación de un despliegue Kubernetes equivalente.

Aviso: Para la realización de esta tarea he utilizado Minikube. Minikube es una herramienta que nos permite crear un clúster de Kubernetes en local. El formato del .yaml del Ingress no es exactamente igual en Kubernetes de GCP que el de Minikube, ten esto en cuenta si vas a desplegarlo en GCP. También tener en cuenta que Minikube no trae los Ingress activados por defectos y debe ser activados mediante:

```
minikube addons enable ingress
```

Para el despliegue he usado cinco objetos de Kubernetes, Deployment, Service, PersistentVolume, PersistentVolumeClaim e Ingress.

El Deployment es el objeto que se encarga de crear los pods y de asegurarse de que estos estén en el estado y numero deseado. He creado uno por cada imagen que he creado asignándole un nombre, una imagen de mi DockerHub y el puerto en caso de que haga uso de uno.

El Service es el objeto que se encarga de asignar una IP y un puerto a los pods para que puedan ser identificados. He creado uno para cada imagen a la que necesite acceder desde otro pod asignándole un nombre idéntico al usado en el Docker. Estos servicios son de tipo ClusterIP, lo que significa que solo pueden ser accedidos desde dentro del clúster.

El PersistentVolume es el objeto que se encarga de crear un volumen persistente en el nodo para que los datos no se pierdan al reiniciar el pod. He creado uno para la base de datos y otro para la web asignándole un nombre, un tamaño y un tipo de almacenamiento. El tipo de almacenamiento es hostPath, lo que significa que el volumen se crea en el nodo y no en un servicio externo.

El PersistentVolumeClaim es el objeto que se encarga de reclamar el volumen persistente para que pueda ser utilizado por el pod.

El Ingress es el objeto que se encarga de enrutar el tráfico a los servicios correspondientes. He creado uno con redirecciones a la web y otro al panel de administración de la base de datos. Para ello he definido las reglas de enrutamiento y el servicio al que deben ser redirigidas. En caso de no especificar ruta se redirige a la web mientras que usando /db se redirige al panel de administración de la base de datos.

4.5. Inclusión de imágenes adicionales.

Como imagin adicionales he utilizado Adminer y Vault.

Adminer es una herramienta web que nos permite gestionar diferentes sistemas de gestion de bases de datos entre los que se incluye PostgreSQL. Para la creacion de la imagen he utilizado la imagen oficial de Adminer.

Vault es un gestor de claves que nos permite almacenar de forma segura las claves de nuestro proyecto sin la necesidad de tenerlas escritas explicitamente en el codigo fuente de nuestra aplicacion. Para la creacion de la imagen he utilizado la imagen oficial de Vault. Para la configuracion de Vault he creado un archivo .hcl en el que se definen la configuracion de inicio de Vault. Esto es necesario dado que Vault, por defecto, almacena las claves en memoria y por lo tanto se pierden al reiniciar el contenedor. Entre las configuraciones que he definido se encuentra el almacenamiento de claves en ficheros y la configuracion de red de la API.

4.6. Utilización de funcionalidades Docker/Kubernetes no vistas en clase.

Como funcionalidad no vista en clase he utilizado el networking de Docker. Definir el networking de los contenedores es una parte muy importante y útil de Docker. La principal funcionalidad que nos ofrece es la de aislar los contenedores de forma que no puedan comunicarse más que con aquellos contenedores que nosotros queramos.

Esto puede ser muy útil si tenemos una API para comunicar la base de datos con la web ya que de esta forma la base de datos no podría ser accesible desde la web lo que impediría que un atacante pudiera acceder a ella. Por desgracia, mi aplicación web tiene el backend y el frontend en el mismo contenedor por lo que no he podido aprovechar esta funcionalidad al 100 %. Aun así, he añadido la funcionalidad de networking como prueba de concepto. La configuración de la red de Docker puede ser vista en el archivo docker-compose.yaml.

Además, he utilizado también un HEALTHCHECK en el Dockerfile del SGBD para comprobar que la base de datos está disponible antes de que el robot intente escribir en ella. Esta configuración es útil para evitar errores debido a que un servicio al que necesitas acceder aún no está disponible. La configuración del HEALTHCHECK puede ser vista en el Dockerfile del SGBD y la comprobacion en el docker-compose.yaml

Declaración sobre uso de asistentes virtuales

Para la realización de este proyecto se ha utilizado dos asistentes virtuales, GPT-3.5 y GitHub Copilot.

El uso que le he dado a GitHub Copilot ha sido bastante limitado. La única tarea que ha desempeñado ha sido la de autocompletado de código que aporta mediante su integración con Visual Studio Code.

Por otro lado, el uso que le he dado a GPT-3.5 ha sido bastante más amplio. Su uso principal ha sido el de ayudarme en las tareas de debugging mediante la interpretación de log y mensajes de error. También lo he utilizado para preguntas más generales y para aportarme ideas sobre que podía implementar en el proyecto. Por último, la corrección de errores ortográficos y gramaticales en este informe.

Bibliografia

- GPT-3.5. (2023). Respuesta a una pregunta sobre PHP. OpenAI. https://www.openai.com/
- GitHub Copilot. (2022). Autocompletado. GitHub. https://github.com/features/copilot
- Docker. (2021). Documentación. Docker. https://docs.docker.com/
- Kubernetes. (2021). Documentación. Kubernetes. https://kubernetes.io/docs/home/
- Minikube (2023). Documentación Minikube. https://minikube.sigs.k8s.io/docs/handbook/
- Apache. (2021). Documentación. Apache. https://httpd.apache.org/docs/
- PHP. (2021). Documentación. PHP. https://www.php.net/docs.php
- PostgreSQL. (2021). Documentación. PostgreSQL. https://www.postgresql.org/docs/
- Vault (2023). Documentación. Vault. https://developer.hashicorp.com/vault/docs