Kubernetes, Docker y Despliegue de un Juego Multijugador

David Díaz Jhonatan Díaz David López

1. ¿Qué es Kubernetes?

Kubernetes es una **plataforma de orquestación de contenedores** de código abierto diseñada para automatizar la *implementación*, el *escalado* y la *gestión* de aplicaciones distribuidas en contenedores [4]. Fue desarrollada originalmente por Google basándose en su experiencia con sistemas internos como *Borg*, y posteriormente donada a la *Cloud Native Computing Foundation (CNCF)* en 2015. El nombre *Kubernetes* proviene del griego y significa "timón" o "piloto", reflejando su función de dirigir la infraestructura de aplicaciones contenedorizadas.

1.1. Características principales

- **Automatización del despliegue y escalado:** Kubernetes permite lanzar y replicar contenedores automáticamente, ajustando el número de instancias según la carga del sistema (*Horizontal Pod Autoscaling*).
- **Gestión declarativa:** las configuraciones se definen en archivos YAML o JSON, permitiendo al sistema mantener el estado deseado de forma automática.
- **Alta disponibilidad y tolerancia a fallos:** detecta fallos en contenedores o nodos y los reemplaza sin afectar la disponibilidad del servicio.
- **Balanceo de carga y descubrimiento de servicios:** asigna automáticamente direcciones IP y nombres DNS internos, además de distribuir el tráfico de red de manera equitativa.
- **Portabilidad y compatibilidad:** puede ejecutarse en servidores locales, nubes públicas o entornos híbridos.

1.2. Aplicaciones de Kubernetes

Kubernetes se utiliza ampliamente en:

- **Despliegue de microservicios:** permite dividir aplicaciones grandes en componentes independientes y escalables.
- **Entornos CI/CD:** facilita la integración y entrega continua gracias a su automatización del despliegue.
- **Computación en la nube y edge computing:** gestiona cargas distribuidas en distintos entornos.
- **Sistemas de aprendizaje automático:** permite ejecutar clústeres de procesamiento de datos y modelos de IA en producción.

1.3. Relación con los contenedores

Kubernetes **no crea contenedores**, sino que los **orquesta**. Utiliza herramientas como Docker, containerd o CRI-O para ejecutar los contenedores y, sobre ellos, gestiona su ciclo de vida: creación, despliegue, actualización, monitoreo y eliminación.

Mientras que **Docker** empaqueta la aplicación y sus dependencias, **Kubernetes** coordina miles de esos contenedores, asegurando su comunicación, escalado y mantenimiento operativo. En resumen:

> *Docker crea los contenedores; Kubernetes los administra.*

2. Creación de Contenedores con Docker

Docker es una plataforma que permite empaquetar aplicaciones y sus dependencias en unidades ligeras llamadas **contenedores**. Estos contenedores se basan en **imágenes**, que son plantillas inmutables definidas mediante un archivo denominado *Dockerfile*. A continuación se describen los conceptos clave y el procedimiento paso a paso para crear contenedores desde cero.

2.1. Conceptos Clave

- **Imagen**: plantilla que contiene el sistema de archivos, dependencias y configuración necesaria para ejecutar una aplicación.
- **Contenedor**: instancia en ejecución de una imagen. Puede iniciarse, detenerse o eliminarse sin afectar la imagen original.

- **Dockerfile**: archivo de texto con instrucciones para construir una imagen personalizada.
- **Registro (Registry)**: repositorio donde se almacenan imágenes, por ejemplo Docker Hub.
- **Volumen**: área de almacenamiento persistente para datos generados por contenedores.
- **Red de Docker**: mecanismo que permite la comunicación entre contenedores o entre un contenedor y el exterior.

2.2. Procedimiento para Crear un Contenedor desde Docker

El proceso general para crear un contenedor consta de cuatro pasos principales:

1. **Crear un Dockerfile**: definir los pasos necesarios para construir la imagen.
2. **Construir la imagen**: utilizando el comando `docker build`.
3. **Ejecutar un contenedor**: mediante el comando `docker run`.
4. **Gestionar el contenedor**: visualizar, detener y eliminar contenedores según sea necesario.

2.3. Ejemplo Sencillo: Aplicación Web en Python

1. Archivo `app.py`

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def index():
    return "Hola desde Docker!"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

2. Archivo `requirements.txt`

```
Flask==2.2.5
```

3. Archivo `Dockerfile`

```
FROM python:3.11-slim
WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY app.py .
EXPOSE 5000

CMD ["python", "app.py"]
```

2.4. Construcción y Ejecución del Contenedor

```
docker build -t miapp:1.0 .

docker run -d --name miapp_con -p 5000:5000 miapp:1.0
```

3. despliegue juego

El proyecto se compone de los siguientes archivos principales:

- **server.js**:
  - Servidor Node.js que gestiona el juego multijugador usando `Socket.io`.
  - Mantiene un objeto `players` con la posición de cada jugador.
  - Implementa **mutex** para evitar condiciones de carrera al modificar el estado global.
  - Implementa un **semáforo** para limitar el número máximo de jugadores conectados simultáneamente.
  - Define la ruta `/` que entrega la página HTML inicial del juego.
- **Dockerfile**:
  - Contiene las instrucciones para construir la imagen Docker del servidor.

- Se usa la imagen base `node:18-alpine`.
- Copia `server.js` dentro del contenedor.
- Instala las dependencias `express` y `socket.io`.
- Expone el puerto 3000 y ejecuta el servidor con `CMD ["node", "server.js"]`.

- **juego-deployment.yaml**:
  - Archivo de Kubernetes que define el `Deployment`.
  - Configura el número de réplicas de pods (en este caso 1, pero puede aumentarse).
  - Selecciona la imagen Docker creada (`juego-multijugador:v1`) para ejecutar dentro de los pods.
  - Expone el puerto 3000 dentro del pod.
- **juego-service.yaml**:
  - Define el `Service` de tipo `NodePort`.
  - Permite que los jugadores accedan al juego desde el exterior del cluster.
  - Balancea la carga entre los pods automáticamente.
  - Mapea el puerto interno 3000 al puerto externo 30080.

3.0.1. Ejecución del Proyecto

1. **Construcción de la imagen Docker:**

   ```
   docker build -t juego-multijugador:v1 .
   ```

2. **Usar Docker de Minikube:**

   ```
   eval $(minikube docker-env)
   ```

3. **Aplicar Deployment:**

   ```
   kubectl apply -f juego-deployment.yaml
   ```

4. **Aplicar Service:**

   ```
   kubectl apply -f juego-service.yaml
   ```

5. **Verificar:**

   ```
   kubectl get pods
   kubectl rollout status deployment/juego-deployment
   ```

6. **Acceder al juego:**

   ```
   minikube service juego-service --url
   ```

3.1. Referencias

Referencias

[1] Docker Inc. *Docker Documentation*. Disponible en: `https://docs.docker.com`

[2] Boettiger, C. (2015). "An Introduction to Docker for Reproducible Research". *ACM SIGOPS Operating Systems Review*, 49(1), 71–79.

[3] Merkel, D. (2014). "Docker: Lightweight Linux Containers for Consistent Development and Deployment". *Linux Journal*, 2014(239).

[4] Google Cloud. (2024). *What is Kubernetes?* Recuperado de `https://cloud.google.com/learn/what-is-kubernetes`

[5] Red Hat. (2023). *¿Qué es Kubernetes y para qué sirve?* Recuperado de `https://www.redhat.com/es/topics/containers/what-is-kubernetes`