Computación Tolerante a Fallas

Sección D06

Proyecto Final

Fernández Venegas David Guadalupe

Código: 216437352

Profesor: López Franco Michel Emanuel

Introducción

Se busca realizar un proyecto para la materia de Computación Tolerante a Fallas. Este se basa en una serie de principios y prácticas que buscan garantizar la disponibilidad y confiabilidad de una aplicación:

1. División en Servicios Independientes: Este enfoque arquitectónico promueve la modularidad y la independencia de los componentes de la aplicación. Cada microservicio es responsable de una tarea específica y se comunica con otros a través de interfaces bien definidas. Esto facilita el desarrollo, el mantenimiento y la escalabilidad de la aplicación.
2. Contenerización con Docker: La contenerización con Docker permite encapsular cada microservicio junto con sus dependencias en un contenedor ligero y portable. Esto garantiza que cada servicio pueda ejecutarse de manera consistente y aislada en cualquier entorno, lo que facilita el despliegue y la gestión de la aplicación.
3. Orquestación con Kubernetes: Kubernetes es una herramienta poderosa para la gestión de contenedores a escala. Permite automatizar tareas como el despliegue, el escalado y la recuperación ante fallos de los microservicios. Con Kubernetes, se puede garantizar que la aplicación se ejecute de manera eficiente y resiliente en cualquier entorno.
4. Comunicación entre Microservicios: La comunicación entre microservicios es fundamental en una arquitectura basada en servicios. Se pueden utilizar diferentes mecanismos, como APIs REST, mensajería asíncrona o llamadas remotas, dependiendo de los requisitos de la aplicación y las preferencias del equipo de desarrollo.
5. Monitorización y Observabilidad: La monitorización y observabilidad son aspectos críticos para entender el comportamiento de la aplicación y detectar problemas de manera proactiva. Herramientas como Istio o Azure Monitor permiten recopilar métricas, registros y trazas que pueden utilizarse para optimizar el rendimiento y la confiabilidad de la aplicación.
6. Automatización CI/CD: La automatización del proceso de construcción, pruebas e implementación de los microservicios mediante herramientas de CI/CD garantiza una entrega rápida y confiable de la aplicación. Esto reduce el tiempo de comercialización y minimiza el riesgo de errores humanos en el proceso de implementación.
7. Diseño Escalable y Resiliente: El diseño de la arquitectura para que los microservicios sean escalables y resilientes es fundamental para garantizar la disponibilidad de la aplicación. Esto implica diseñar estrategias de escalado horizontal y vertical, así como implementar mecanismos de recuperación ante fallos, como la redundancia y la tolerancia a fallos.

Proyecto realizado

El proyecto realizado consiste en un sistema web básico usado para la gestión de pedidos de una tienda. En dicho sistema se puede gestionar una lista de clientes y productos para poder generar ordenes conforme a estos, pudiendo realizar acciones de alta, consulta, modificación y eliminación por medio de un listado.

Este proyecto emplea Kubernetes para gestionar el entorno Docker. Kubernetes facilita tanto el despliegue de servicios como el balanceo de carga. Las solicitudes a los servicios se redirigen mediante un proxy inverso configurado con Apache httpd y los servicios están implementados en Java utilizando Spring y Spring Cloud.

Balanceador de Carga Apache HTTP

Apache HTTP se utiliza para proporcionar la página web en el puerto 8080. También reenvía las solicitudes HTTP a los microservicios. Aunque esto no es estrictamente necesario, ya que cada servicio tiene su propio puerto en el host Minikube, ofrece un punto de entrada único para todo el sistema. Apache HTTP está configurado como un proxy inverso para este propósito. El equilibrio de carga se deja a Kubernetes.

Para configurar esto, Apache HTTP necesita obtener todos los servicios registrados en Kubernetes, utilizando DNS.

Microservicios.

- microservice-kubernetes-demo-catalog: Se encarga de gestionar los datos de los artículos.
- microservice-kubernetes-demo-customer: Se encarga de gestionar los datos de los clientes.
- microservice-kubernetes-demo-order: Procesa los pedidos y utiliza los servicios de utilizando los dos servicios anteriores.

Los microservicios se comunican entre sí a través de REST (Representational State Transfer). Este es un estilo arquitectónico para diseñar sistemas distribuidos, especialmente aplicaciones web, que se basa en estándares y protocolos HTTP.

Los microservicios tienen una aplicación principal en Java ubicada en src/test/java para ejecutarlos de manera independiente. En este modo, microservice-demo-order usa stubs para los otros servicios.

Despliegue de la aplicación

Prerrequisitos:

- Tener instalado Docker y configurado correctamente.
- Instalar minikube: Minikube es una herramienta que permite ejecutar un clúster de Kubernetes de un solo nodo en una máquina local. Está diseñada para facilitar el aprendizaje, el desarrollo y las pruebas de aplicaciones y servicios en un entorno Kubernetes sin necesidad de desplegar un clúster completo en la nube o en un entorno de producción.
- Kubectl: kubectl es una herramienta de línea de comandos para interactuar con clústeres de Kubernetes. Permite a los usuarios ejecutar comandos contra sus clústeres de Kubernetes, facilitando la gestión y operación de aplicaciones contenedorizadas.

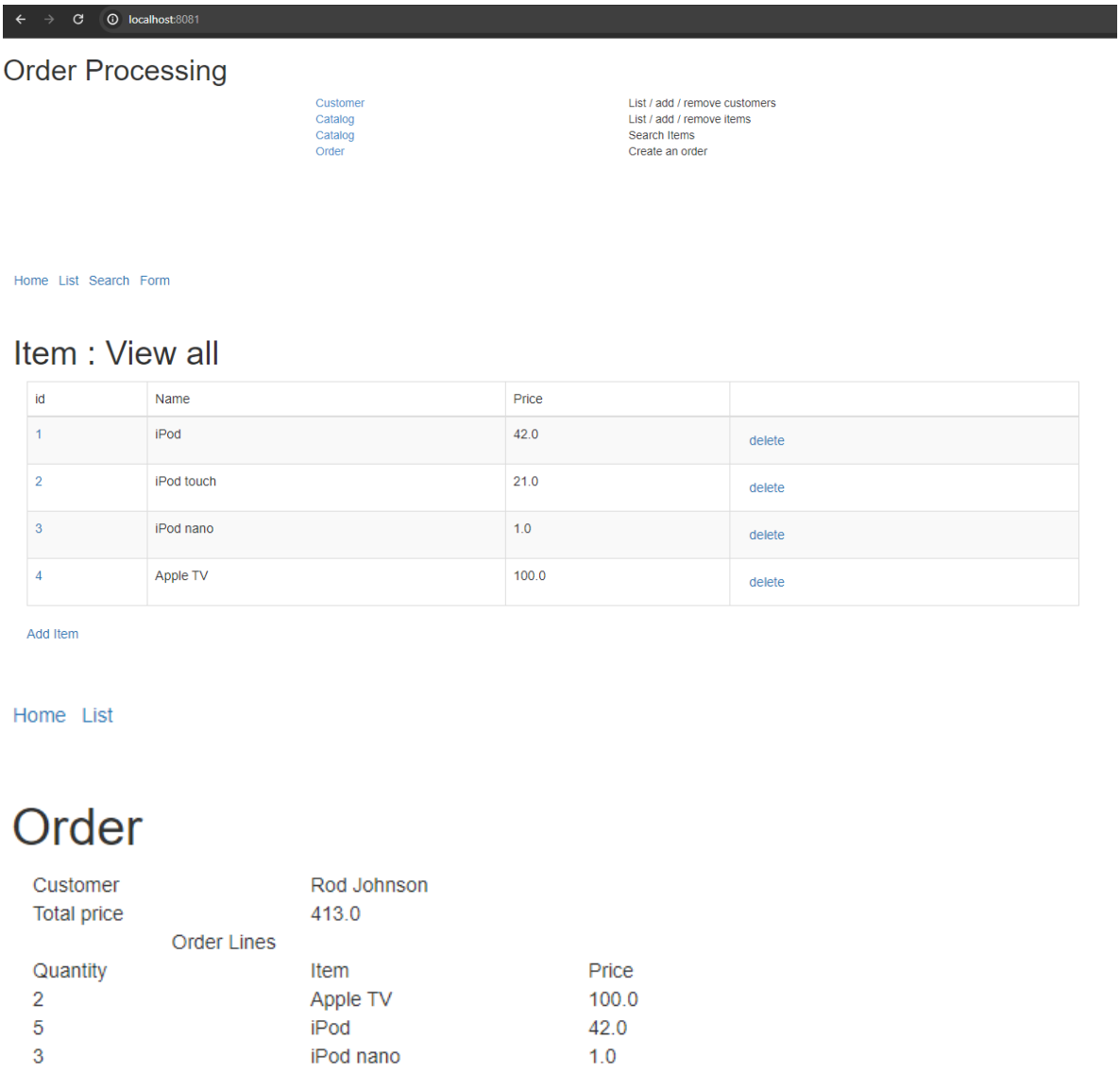
Pasos para seguir:

1. Copia el repositorio de GitHub: 'git clone https://github.com/ewolff/microservice-kubernetes.git'
2. Crea una instancia de minikube: 'minikube start --memory=4000'.
3. Entra al directorio ´microservice-kubernetes-demo´ y usa el comando ´ kubectl apply -f microservices.yaml´ para tomar la descripción de los servicios y desplegarlos desde el archivo 'microservices.yaml'. Esto crea los servicios si aún no existen y además despliega las imágenes y crea los Pods. Cada Pod tiene solo un contenedor Docker.
4. Ejecuta el siguiente comando para ver todos los servicios: 'kubectl get services'.

```
NAME         CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
apache       10.0.0.90     <pending>     80:31214/TCP     46s
catalog      10.0.0.219    <pending>     8080:30161/TCP   46s
customer     10.0.0.163    <pending>     8080:30620/TCP   45s
kubernetes   10.0.0.1      <none>        443/TCP          3m
order        10.0.0.21     <pending>     8080:30616/TCP   45s
```

5. Por último, ejecuta 'kubectl port-forward deployment/apache 8081:80' para crear un proxy al servidor Apache httpd en tu máquina local. Luego, abre http://localhost:8081 para ver la página web del servidor Apache httpd en el navegador.

Proyecto desplegado

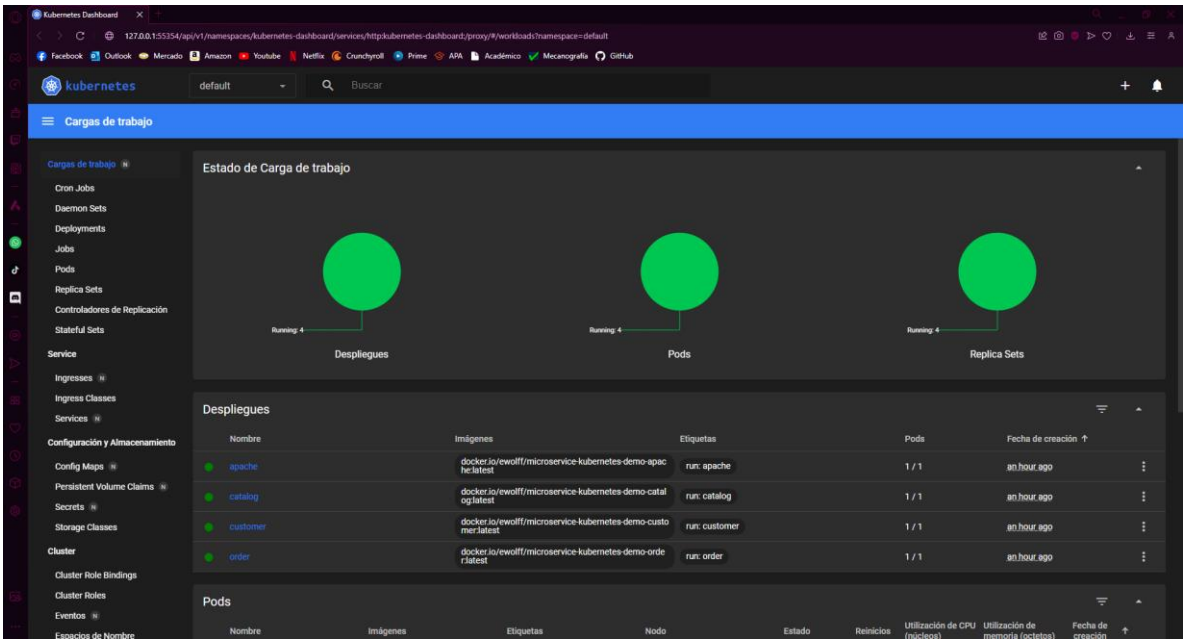


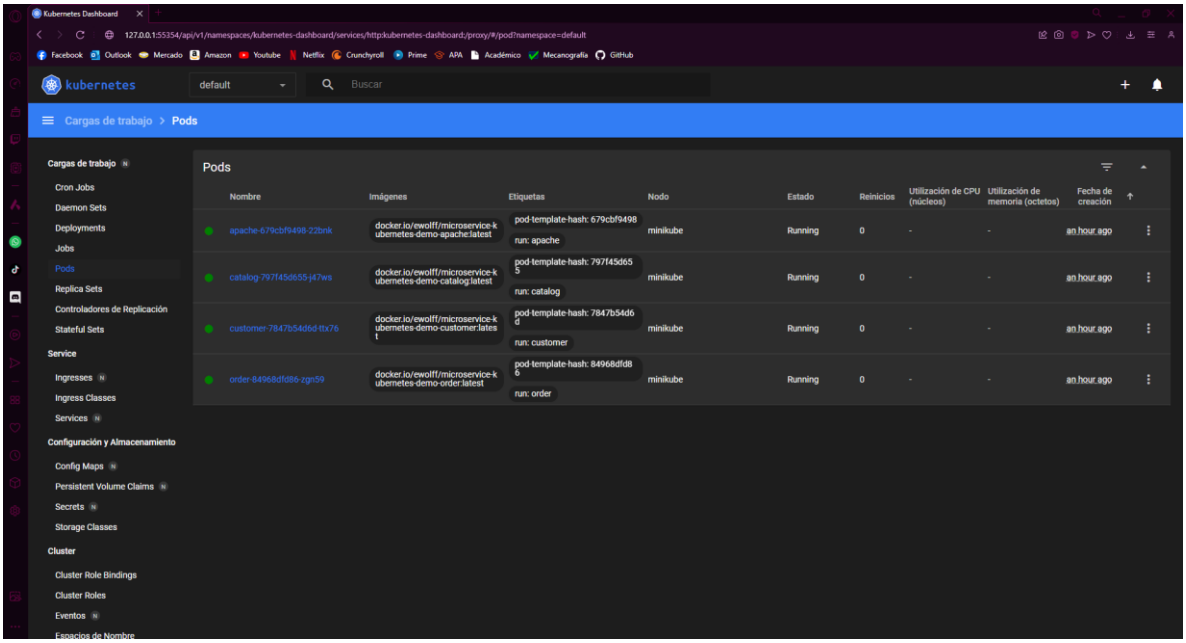
Dashboard de Kubernetes

El dashboard web de Kubernetes proporciona una vista en tiempo real de todos los recursos del clúster de Kubernetes, lo que facilita la monitorización y la solución de problemas. Este es especialmente útil para los usuarios nuevos en Kubernetes, que pueden encontrar más fácil trabajar con una interfaz gráfica de usuario que con la línea de comandos. Además, es compatible con muchas herramientas de terceros que pueden integrarse para proporcionar una mejor visibilidad, monitoreo y administración del clúster de Kubernetes.

Este dashboar es una herramienta esencial para administrar y monitorear los recursos de Kubernetes de manera visual. Proporciona una vista general del estado del clúster, facilita la solución de problemas y mejora la productividad de los equipos de operaciones y desarrollo.

Para acceder al dashboard de kubernetes solamente tenemos que utilizar el comando ‘minikube dashboard’ para poder visualizarlo en nuestro navegador.





Escalabilidad del proyecto

- Escalado horizontal de los microservicios: Kubernetes facilita el escalado horizontal de los microservicios, lo que significa que podemos aumentar o disminuir la cantidad de réplicas de cada microservicio según la demanda. Por ejemplo, si notamos un aumento en el tráfico de pedidos, podemos escalar horizontalmente el servicio de pedidos para manejar la carga adicional de manera eficiente.
- Optimización de recursos: Mediante el monitoreo continuo del rendimiento de los microservicios a través del dashboard de Kubernetes, podemos identificar cuellos de botella y optimizar los recursos asignados a cada microservicio. Esto puede implicar ajustar la asignación de CPU y memoria, así como optimizar el código y las consultas de base de datos para mejorar la eficiencia.
- Uso de servicios de almacenamiento escalables: Si el volumen de datos aumenta significativamente, podríamos considerar utilizar servicios de almacenamiento escalables, como Amazon S3 o Google Cloud Storage, para almacenar archivos estáticos, imágenes de productos, etc. Esto nos permitiría escalar verticalmente el almacenamiento según sea necesario y garantizar un rendimiento consistente.
- Implementación de una arquitectura de eventos: Para mejorar la modularidad y la escalabilidad del sistema, podríamos implementar una arquitectura de eventos utilizando herramientas como Kafka o RabbitMQ. Esto nos permitiría desacoplar los microservicios y facilitaría la integración de nuevos servicios en el futuro, lo que haría que el sistema fuera más flexible y escalable a largo plazo.