Centro Universitario de Ciencias Exactas e Ingenierías

CUCEI

Computación Tolerante a fallas

2023B

Kubernetes

Michel Emanuel López Franco

Gutiérrez Galan Ruben Alejandro

Codigo: 214798315

Contenido

Kubernetes:

Kubernetes es una plataforma de código abierto diseñada para automatizar la implementación, escala y operación de aplicaciones en contenedores. Fue desarrollado por Google y posteriormente se convirtió en un proyecto de la Cloud Native Computing Foundation (CNCF), que es una fundación independiente que se centra en impulsar la adopción de tecnologías nativas de la nube.

En términos más simples, Kubernetes proporciona un entorno para la orquestación de contenedores. Los contenedores son unidades ligeras y portátiles que pueden ejecutar aplicaciones y sus dependencias de manera consistente en cualquier entorno. Kubernetes facilita la gestión de estos contenedores, asegurando que se ejecuten de manera confiable y escalen según sea necesario.

Características:

1. Orquestación de Contenedores: Kubernetes automatiza el despliegue, la escalabilidad y la gestión de aplicaciones en contenedores.
2. Escalabilidad: Permite escalar las aplicaciones hacia arriba o hacia abajo según la demanda de tráfico o recursos.
3. Auto curación: Se encarga de la salud de las aplicaciones, reemplazando contenedores que fallan o no responden.
4. Despliegue Declarativo: Se define el estado deseado de las aplicaciones y Kubernetes se encarga de llevarlas a ese estado.
5. Descubrimiento de Servicios: Facilita la conexión y la descubribilidad entre servicios dentro del clúster.
6. Almacenamiento Orquestado: Gestiona el almacenamiento y permite a las aplicaciones acceder a él de manera uniforme.
7. Portabilidad: Funciona en una variedad de entornos, incluyendo entornos locales, en la nube y en configuraciones híbridas.

Ingress:

Ingress es un recurso que gestiona el acceso externo a los servicios dentro de un clúster. Permite exponer servicios HTTP y HTTPS de manera externa y proporciona funciones avanzadas de enrutamiento y reglas de tráfico. En otras palabras, el Ingress actúa como una capa de entrada para manejar las solicitudes entrantes desde fuera del clúster y dirigirlas a los servicios correspondientes.

Características:

1. Enrutamiento basado en hosts y rutas: Puedes configurar reglas basadas en el nombre del host o en rutas de URL para dirigir el tráfico a servicios específicos.
2. Soporte para TLS/SSL: Permite la terminación SSL en el clúster, lo que significa que puedes gestionar certificados SSL/TLS para tus servicios.
3. Balanceo de carga: Puedes utilizar Ingress para distribuir el tráfico entre varios servicios, proporcionando un equilibrio de carga.
4. Redirecciones: Permite configurar redirecciones de URL.
5. Autenticación y autorización: Algunos controladores de Ingress permiten configurar políticas de autenticación y autorización para proteger los servicios.

Loadbalancer:

Un Load Balancer (balanceador de carga) es un componente que distribuye el tráfico de red o las solicitudes de manera equitativa entre múltiples servidores o nodos para mejorar la disponibilidad, la confiabilidad y la escalabilidad de una aplicación o servicio. La función principal de un balanceador de carga es asegurarse de que cada servidor o nodo en el conjunto tenga una carga de trabajo equitativa, evitando así la congestión en algún servidor específico y mejorando la eficiencia general del sistema.

Existen varios tipos de balanceadores de carga, y uno de ellos es el balanceador de carga de red, que a menudo se menciona en el contexto de entornos en la nube y Kubernetes.

- Balanceador de Carga de Red (Network Load Balancer): En entornos en la nube como AWS (Amazon Web Services), un balanceador de carga de red se encarga de distribuir el tráfico entre instancias (máquinas virtuales) en una red. Asocia una dirección IP única con el balanceador de carga y dirige las solicitudes a instancias específicas basándose en reglas de enrutamiento.
- En el contexto de Kubernetes, el balanceador de carga de red puede ser utilizado para distribuir el tráfico entre los nodos del clúster. Cada nodo del clúster puede ejecutar contenedores que contienen aplicaciones, y el balanceador de carga de red se encarga de enrutar las solicitudes a los nodos adecuados según la carga de trabajo y las reglas de enrutamiento configuradas.
- Balanceador de Carga de Aplicación (Application Load Balancer): Este tipo de balanceador de carga opera a nivel de la capa de aplicación (Capa 7 del modelo OSI) y puede tomar decisiones basadas en información más detallada, como el contenido de la solicitud HTTP. Puede dirigir el tráfico a diferentes destinos según la URL solicitada, por ejemplo.
- Balanceador de Carga de Capa de Transporte (Transport Layer Load Balancer): Opera a nivel de la capa de transporte (Capa 4 del modelo OSI) y distribuye el tráfico basándose en direcciones IP y puertos de destino. Este tipo de balanceador de carga a menudo se asocia con protocolos como TCP (Transmission Control Protocol) o UDP (User Datagram Protocol).

Programa desarrollado:

Codigo:

```
''''
Gutierrez Galan Ruben Alejandro codigo: 214798315
Computación Tolerante a fallas
2023B
Arquitecturas monolíticas vs Microservicios
Michel Emanuel López Franco

'''

# app.py
from flask import Flask, request, jsonify
```

```
app = Flask(__name__)

def guardar_resultado(resultado_decimal, resultado_hexa):
    with open('resultados.txt', 'a') as file:
        file.write(f"Decimal: {resultado_decimal}, Hexadecimal:
{resultado_hexa}\n")
    file.close()

@app.route('/calcular', methods=['POST'])
def calcular():
    data = request.get_json()

    operacion = data['operacion']
    numero1 = data['numero1']
    numero2 = data['numero2']

    if operacion == 'sumar':
        resultado = numero1 + numero2
    elif operacion == 'restar':
        resultado = numero1 - numero2
    elif operacion == 'multiplicar':
        resultado = numero1 * numero2
    elif operacion == 'dividir':
        if not isinstance(numero1, (int, float)) or not isinstance(numero2,
(int, float)):
            return jsonify({"error": "Los números deben ser enteros o de
punto flotante"}), 400
        if numero2 != 0:
            resultado = numero1 / numero2
        else:
            return jsonify({"error": "No se puede dividir por cero"}), 400
    else:
        return jsonify({"error": "Operación no válida"}), 400
    resultado=int(resultado)
    resultado_hexa = hex(resultado)  # Convierte el resultado a hexadecimal
    guardar_resultado(resultado, resultado_hexa)  # Guarda el resultado en
el archivo

    return jsonify({"resultado": resultado, "resultado_hexa":
resultado_hexa})


if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

Archivos Importantes:

Dockerfile

```
# Dockerfile

FROM python:3.8

WORKDIR /app

COPY requirements.txt .

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

COPY . .

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

requeriments.txt

```
Flask==2.0.1
Werkzeug==2.0.1
```

deployment.yaml

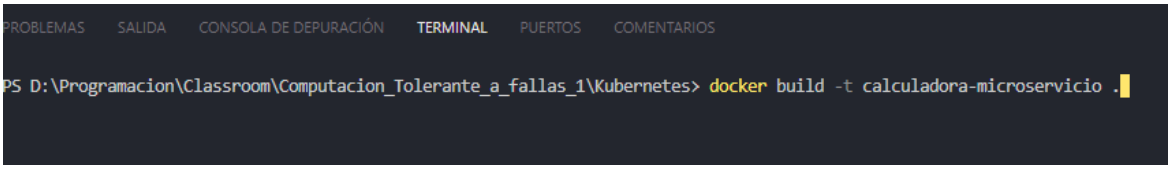
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mi-aplicacion
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mi-aplicacion
  template:
    metadata:
      labels:
        app: mi-aplicacion
    spec:
      containers:
      - name: mi-aplicacion
        image: calculador-microservicio
        ports:
        - containerPort: 5000
```

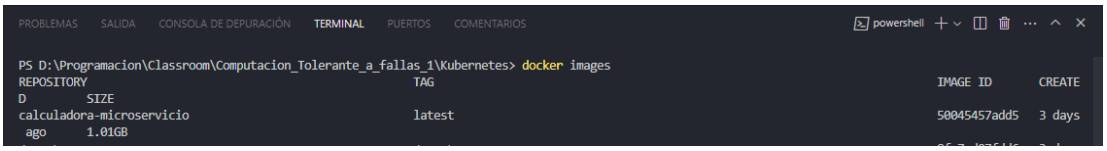
service.yaml

```yaml
# service.yaml
apiVersion: v1
kind: Service
metadata:
  name: mi-aplicacion-service
spec:
  selector:
    app: mi-aplicacion
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
  type: LoadBalancer
```

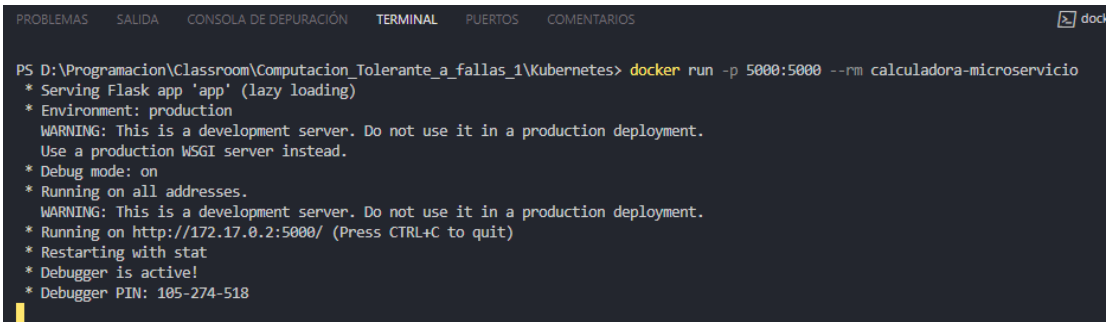
Comandos Usados:

Construir imagen:

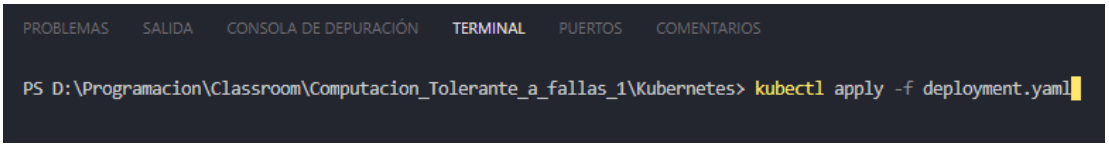




Lanzar el microservicio



Usar kubernetes



```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL   PUERTOS   COMENTARIOS

PS D:\Programacion\Classroom\Computacion_Tolerante_a_fallas_1\Kubernetes> kubectl apply -f service.yaml
```

```
PS D:\Programacion\Classroom\Computacion_Tolerante_a_fallas_1\Kubernetes> kubectl get services
NAME                            TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes                      ClusterIP      10.96.0.1        <none>        443/TCP        3d
mi-aplicacion-service           LoadBalancer   10.107.36.195    localhost     80:31675/TCP   3d
servicio-calculadora-kubernetes LoadBalancer   10.100.146.215   <pending>     80:30760/TCP   58m
tu-app-service                  LoadBalancer   10.103.57.207    <pending>     80:30055/TCP   63m
PS D:\Programacion\Classroom\Computacion_Tolerante_a_fallas_1\Kubernetes>
```

Ejecución:

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL   PUERTOS   COMENTARIOS

PS D:\Programacion\Classroom\Computacion_Tolerante_a_fallas_1\Kubernetes> kubectl get services
NAME                            TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes                      ClusterIP      10.96.0.1        <none>        443/TCP        3d
mi-aplicacion-service           LoadBalancer   10.107.36.195    localhost     80:31675/TCP   3d
servicio-calculadora-kubernetes LoadBalancer   10.100.146.215   <pending>     80:30760/TCP   58m
tu-app-service                  LoadBalancer   10.103.57.207    <pending>     80:30055/TCP   63m
PS D:\Programacion\Classroom\Computacion_Tolerante_a_fallas_1\Kubernetes> Invoke-RestMethod -Uri "http://localhost:5000/calcular" -Metho
d Post -Headers @{"Content-Type"="application/json"} -Body '{"operacion": "multiplicar", "numero1": 1000, "numero2": 20, "numero3": 15}'
>>

resultado resultado_hexa
--------- --------------
    20000 0x4e20


PS D:\Programacion\Classroom\Computacion_Tolerante_a_fallas_1\Kubernetes>
```

Conclusion:

Para esta actividad se hizo uso de kubernetes para generar una automatización en el manejo de aplicaciones desarrolladas en contenedores, use la propia aplicación desarrollada anteriormente cuando se creo el microservicio, pero ahora usamos los archivos yaml para tener nuestros servicios, tuve algunas complicaciones al momento de usar kubernetes porque tenía problemas con las direcciones IP en especial con la externa que no se me generaba y por ende no podía correr de manera correcta la aplicación al final si obtuve una que coincidió con el localhost lo que me permitió ejecutar de manera correcta la aplicación y mostrar una salida sartisfactoria.

Bibliografía:

- *Orquestación de contenedores para producción*. (n.d.). Kubernetes.

  https://kubernetes.io/es/

- *Docker: Accelerated Container Application Development*. (2023c, October 18).

  Docker. https://www.docker.com/