



Kubernetes

Computación Tolerante a Fallas

Alumno:

Código:

Profesor(a): Dr. Michel Emanuel López Franco

Sección: D06

Fecha de Entrega: 07 de noviembre de 2022

Preguntas

¿Qué es Kubernetes?

Es una plataforma de código abierto diseñada con el objetivo de poder administrar cargas de servicio y trabajos. Automatiza muchos de los procesos manuales involucrados en la implementación, la gestión y el ajuste de las aplicaciones que se alojan en contenedores.

Entre sus principales características, se encuentra ser una plataforma de contenedores, de microservicios, y portable en la nube. Orquesta la infraestructura de cómputo, redes y almacenamiento para que las cargas de trabajo de los usuarios no tengan que hacerlo.

Dentro de Kubernetes, existen los servicios. Estos son un recurso utilizado para crear un único punto de acceso a los pods, identificándolo con una dirección IP y puerto que nunca cambian. Existen cuatro servicios principales: Ingress, LoadBalancer, ClusterIP, y NodePort.

¿Qué es Ingress?

Es un objeto de una API que permite a los usuarios externos y a las aplicaciones clientes acceder a los servicios de un clúster, comúnmente HTTP. Se divide en dos componentes:

- **Ingress Resource:** colección de reglas que determina el tráfico que llega a los servicios.
- **Ingress Controller:** actúa sobre las reglas establecidas en el componente anterior, comúnmente en forma de balanceador de cargas.

¿Qué es un LoadBalancer?

Un balanceador de carga permite repartir la carga de trabajo entre los diferentes servidores o aplicaciones; puede ser instalado en un entorno físico o virtual. Así pues, los programas de balanceador adoptan la forma de un controlador de entrega de aplicaciones (ADC), permitiendo al usuario escalar la carga automáticamente en función de las previsiones de tráfico.

El ADC identifica en tiempo real qué servidores o aplicaciones son las más adecuadas para responder a una petición, garantizando un nivel de rendimiento estable en el clúster. En caso de avería, es posible redirigir el tráfico hacia otro recurso con capacidad para absorberlo.

¿Qué es Rancher?

Es una aplicación para gestionar de una forma centralizada los clústeres de Kubernetes, con una propuesta principal basada en crear una capa de abstracción desde la que poder operar clústeres a diferentes niveles.

Incluye una distribución completa de Kubernetes, Docker Swarm, y Apache Mesos, que facilitan la gestión de los clústeres.

Ejemplo

Para poner en práctica lo aprendido acerca de Kubernetes, decidí crear una pequeña aplicación con Python y el framework Flask. Esta se encarga de realizar peticiones a una API que recupera pasos aleatorios de todos los tutoriales disponibles en la pagina web WikiHow. No tiene mucha utilidad, pero resulta entretenido ver las combinaciones que se pueden llegar a generar.

Lo primero que se hace es importar los módulos requeridos. En este caso, se necesitara Flask, requests para realizar las peticiones, dotenv y os para manejar las variables de entorno y no exponer la clave de la API, y json para formatear las peticiones.

```
from flask import Flask
import requests
import dotenv
import os
import json
```

Creemos una aplicación de Flask que lleve el mismo nombre que el script actual.

```
app = Flask(__name__)
```

Usamos dotevn y os para obtener la clave de la API almacenada como una variable de entorno dentro de un archivo .env, de forma que los usuarios externos no puedan hacer mal uso de esta.

```
# Obtener la clave de la API.
dotenv.load_dotenv ()
apiKey = os.environ["RAPIDAPI_API_KEY"]
```

Luego, creamos la ruta principal de la aplicación utilizando un decorador y la función *route*. Justo debajo, se inicializa la función que realiza la petición y el formateo.

```
@app.route ("/")
def llamadaAPI ():
```

Dentro de esta, establecemos la URL a la que se van a hacer las peticiones, indicamos la cantidad de pasos que se van a recuperar, y configuramos un diccionario con los *headers* necesarios para realizar la petición.

```
# URL de la API.
url = "https://hargrimm-wikihow-v1.p.rapidapi.com/steps"

# Numero de pasos a obtener.
querystring = {"count": "5"}

# Headers de la petición.
headers = {
    "X-RapidAPI-Key": apiKey,
    "X-RapidAPI-Host": "hargrimm-wikihow-v1.p.rapidapi.com"
}
```

Utilizando *requests*, realizamos la petición. La respuesta se convierte a texto, y se usa la función *loads* de *json* para convertirla en un diccionario de Python.

```
# Petición a la API.
response = requests.request("GET", url, headers=headers, params=querystring)
respuestaJSON = response.text

# Formateamos el JSON a un diccionario de Python.
datos = json.loads (respuestaJSON)
```

Al retornar los valores, escribimos un poco de código de HTML para que salgan con un formato agradable a la vista.

```

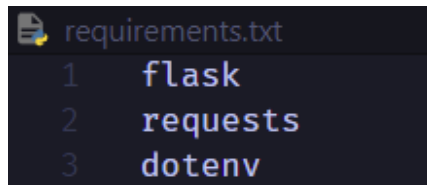
return f"
<div align='left'><h1>Cinco pasos aleatorios de tutoriales de WikiHow</h1>
<br>
<ol>
<li>{datos['1']}</li>
<li>{datos['2']}</li>
<li>{datos['3']}</li>
<li>{datos['4']}</li>
<li>{datos['5']}</li>
</ol>
</div>"

```

Finalmente, usamos la función `run` para correr la aplicación en el puerto 8080 con el debugging activado.

```
app.run (debug=True, port=8080)
```

Ahora, podemos iniciar con la subida de la aplicación a Kubernetes. Creamos un archivo denominado `requirements.txt` para indicarle a Docker que módulos necesita instalar.

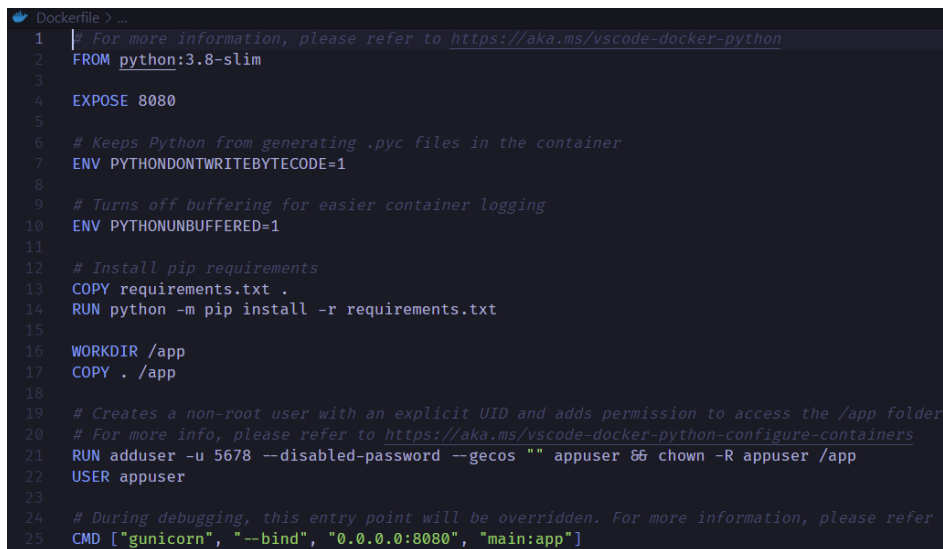


```

requirements.txt
1 flask
2 requests
3 dotenv

```

Creamos un Dockerfile con la información necesaria.



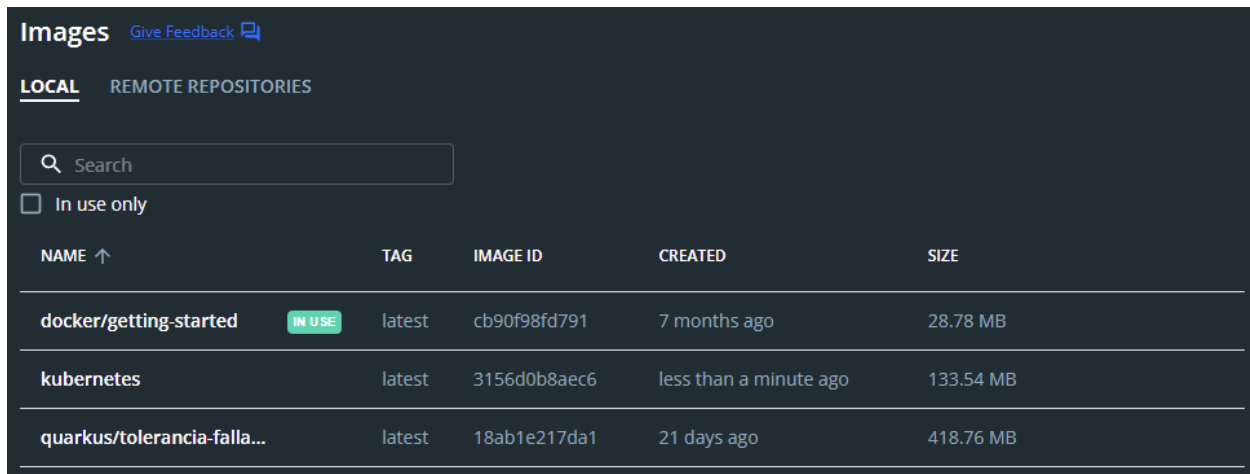
```

Dockerfile > ...
1 # For more information, please refer to https://aka.ms/vscode-docker-python
2 FROM python:3.8-slim
3
4 EXPOSE 8080
5
6 # Keeps Python from generating .pyc files in the container
7 ENV PYTHONDONTWRITEBYTECODE=1
8
9 # Turns off buffering for easier container logging
10 ENV PYTHONUNBUFFERED=1
11
12 # Install pip requirements
13 COPY requirements.txt .
14 RUN python -m pip install -r requirements.txt
15
16 WORKDIR /app
17 COPY . /app
18
19 # Creates a non-root user with an explicit UID and adds permission to access the /app folder
20 # For more info, please refer to https://aka.ms/vscode-docker-python-configure-containers
21 RUN adduser -u 5678 --disabled-password --gecos "" appuser 86 chown -R appuser /app
22 USER appuser
23
24 # During debugging, this entry point will be overridden. For more information, please refer
25 CMD ["gunicorn", "--bind", "0.0.0.0:8080", "main:app"]

```

Utilizando la extensión de VSCode, creamos la imagen.

```
> docker build --rm --pull -f "C:\Users\raul\Desktop\UDG 2022B\Computación Tolerante a Fallas\Programas\Kubernetes\Dockerfile" --label "com.microsoft.created-by=visual-studio-code" -t "kubernetes:latest" "C:\Users\raul\Desktop\UDG 2022B\Computación Tolerante a Fallas\Programas\Kubernetes" <
```



Images [Give Feedback](#)

LOCAL REMOTE REPOSITORIES

Search

☐ In use only

| NAME ↑ | TAG | IMAGE ID | CREATED | SIZE |
|-----------------------------|--------|--------------|------------------------|-----------|
| docker/getting-started | latest | cb90f98fd791 | 7 months ago | 28.78 MB |
| kubernetes | latest | 3156d0b8aec6 | less than a minute ago | 133.54 MB |
| quarkus/tolerancia-falla... | latest | 18ab1e217da1 | 21 days ago | 418.76 MB |

Para crear los Kubernetes, estaré utilizando *minikube*. Esta herramienta permite crear Kubernetes localmente y de forma sencilla para practicar.

```
$ minikube start
🐳 minikube v1.27.1 on Microsoft Windows 10 Home 10.0.19044 Build 19044
🌟 Automatically selected the docker driver. Other choices: virtualbox, ssh
🚀 Using Docker Desktop driver with root privileges
👍 Starting control plane node minikube in cluster minikube
📦 Pulling base image ...
📦 Downloading Kubernetes v1.25.2 preload ...
```

Cargamos la imagen con el comando *minikube image load*.

```
$ minikube image load kubernetes
```

Es importante que antes de continuar activemos Ingress.

```
$ minikube addons enable ingress
💡 ingress is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
```

Lo siguiente es crear un archivo yaml para indicar los parámetros y configuraciones del clúster. Aquí se indica en base a que imagen se va a crear, los puertos sobre los que se va a alojar, y demás configuraciones.

```

flask_deployment.yaml > {} spec > {} template > {} s
io.k8s.api.apps.v1.Deployment (v1@deployment.json)
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: flask-app
5  spec:
6    replicas: 5
7    selector:
8      matchLabels:
9        app: flask-app
10   template:
11     metadata:
12       labels:
13         app: flask-app
14     spec:
15       containers:
16       - name: flask-app-container
17         image: kubernetes
18         imagePullPolicy: Never
19         resources:
20           limits:
21             memory: "128Mi"
22             cpu: "500m"
23         ports:
24         - containerPort: 5000
25           protocol: TCP

```

Usamos *kubectl* para aplicar las configuraciones del archivo *yaml*.

```

$ kubectl apply -f ./flask_deployment.yaml
deployment.apps/flask-app created

```

Como vemos, se han creado 5 *pods*.

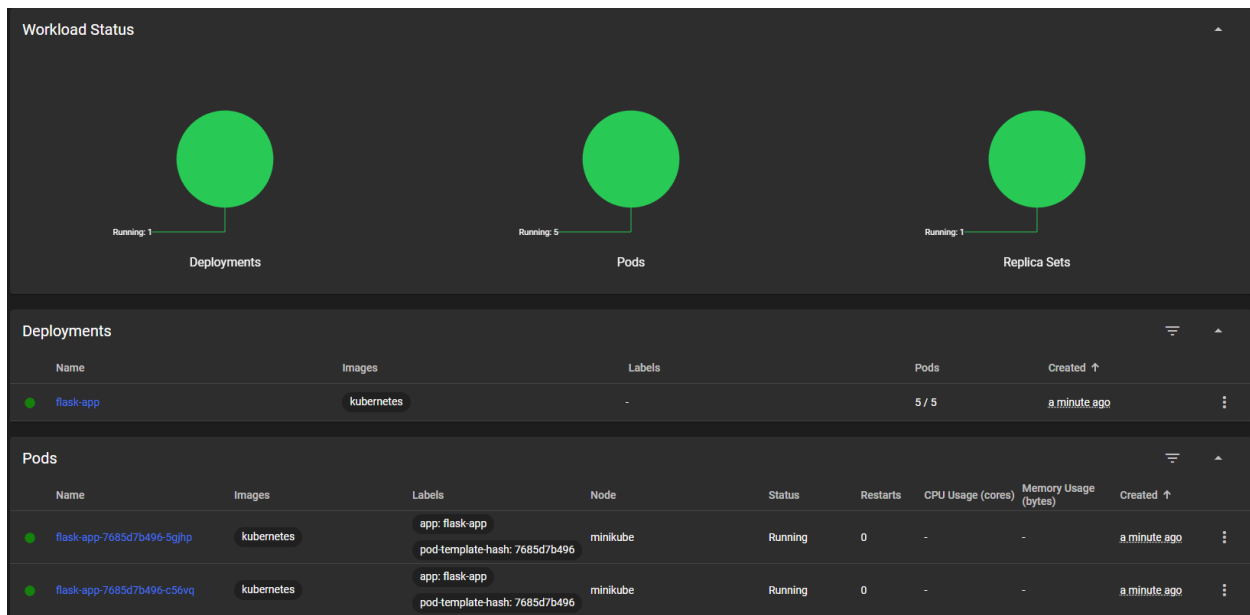
```

$ kubectl get pod

```

| NAME | READY | STATUS | RESTARTS | AGE |
|----------------------------|-------|---------|----------|-----|
| flask-app-7685d7b496-5gjhp | 1/1 | Running | 0 | 82s |
| flask-app-7685d7b496-c56vq | 1/1 | Running | 0 | 82s |
| flask-app-7685d7b496-dw87t | 1/1 | Running | 0 | 82s |
| flask-app-7685d7b496-hkrzs | 1/1 | Running | 0 | 82s |
| flask-app-7685d7b496-js677 | 1/1 | Running | 0 | 82s |

Dentro de la interfaz gráfica, se nos muestra nuestra aplicación creada con Kubernetes.



Ahora, es necesario crear otro archivo yaml para convertir nuestro clúster en un servicio. Dentro de este, se especifica el tipo de servicio que se va a crear, que será ClusterIP, y el puerto donde se alojara.

```
apiVersion: v1
kind: Service
metadata:
  name: flask-app
spec:
  type: ClusterIP
  selector:
    app: flask-app
  ports:
    - port: 5000
```

```
$ kubectl apply -f ./flask_service.yaml
service/flask-app created
```

```
$ kubectl get svc
NAME         TYPE        CLUSTER-IP    EXTERNAL-IP  PORT(S)    AGE
flask-app    ClusterIP   10.111.251.127 <none>       5000/TCP   23s
kubernetes   ClusterIP   10.96.0.1     <none>       443/TCP    14h
```

Finalmente, es necesario crear un ultimo archivo yaml para implementar Ingress en el clúster.


```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: flask-app-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
  labels:
    name: flask-app
spec:
  rules:
  - host: <Host>
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: flask-app-service
            port:
              number: 5000

```

```

$ kubectl apply -f ./flask_ingress.yaml
ingress.networking.k8s.io/flask-app-ingress created

```

```

$ kubectl get ing

```

| NAME | CLASS | HOSTS | ADDRESS | PORTS | AGE |
|-------------------|--------|-------|--------------|-------|-----|
| flask-app-ingress | <none> | * | 192.168.49.2 | 80 | 46s |

La aplicación es visible localmente en la dirección y puerto indicados por Ingress.



Cinco pasos aleatorios de tutoriales de WikiHow

1. Scroll down on your profile page to locate the "Your password" section on the left.
2. Press the OK button to sign in and go to the search page.
3. Thread your needle and tie a knot at the end of the long thread.
4. Choose the destination path using the file browser at the top of the General tab.
5. Ask others.

Conclusiones

Esta actividad fue bastante desafiante, ya que nunca había trabajado con Kubernetes, y desconocía completamente lo que se requería para trabajarlos. No obstante, después de revisar videos en YouTube y leer algo de la documentación, pude implementar satisfactoriamente una aplicación dentro de un clúster local.

Considero que lo aprendido me servirá más adelante para implementar arquitecturas de microservicios.

Bibliografía

¿Qué es Kubernetes? (2022, 17 julio). Kubernetes.
<https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/>

Load balancer con Kubernetes. (s. f.). OVHcloud. Recuperado 1 de noviembre de 2022, de <https://www.ovhcloud.com/es/public-cloud/kubernetes/kubernetes-load-balancer/>

Fernández, A. (2021, 13 julio). Rancher, un solo cluster para gobernarlos a todos. Paradigma Digital. <https://www.paradigmadigital.com/dev/rancher-cluster-para-gobernarlos-a-todos/>