



Kubernetes

Nombre: Diana Fernanda Castillo Rebolledo

Materia: Computación Tolerante a Fallas

Horario: L-I 11:00 am-1:00 pm

Profesor: Dr. Michel Emanuel López Franco

Sección: D06

¿Qué es Kubernetes?

Kubernetes es una plataforma de código abierto que sirve para la organización en contenedores, la cual automatiza muchos de los procesos manuales involucrados en la implementación, gestión y ajuste de las aplicaciones que se alojan en ellos. Un clúster de Kubernetes es un conjunto de máquinas de nodos que ejecutan aplicaciones en contenedores. Si se ejecuta Kubernetes, se está ejecutando un clúster.

En el caso de los contenedores de Docker, se trata de un proceso por contenedor. Kubernetes coloca estos contenedores en un grupo que gestiona en la misma máquina para reducir la sobrecarga de red y aumentar la eficiencia del uso de recursos.

¿Qué es Ingress?

Un servicio en Kubernetes es un recurso que se crea para mantener un único punto constante de acceso a un grupo de pods. Cada servicio tiene una dirección IP y un puerto que nunca cambian. Todos los servicios de Kubernetes tienen la capacidad de proporcionar acceso a un cluster desde el exterior, pero cada uno lo hace de una manera diferente. En Kubernetes existen 4 tipos de servicios que son ClusterIP, NodePort, LoadBalancer e Ingress.

Ingress se usa para exponer rutas HTTP y HTTPS desde el exterior a servicios dentro del cluster. El tráfico entrante es controlado por las reglas que se definen en un archivo de configuración. Una razón para utilizar Ingress es que por cada servicio del tipo LoadBalancer se necesita aprovisionar un balanceador de carga, y eso puede suponer un sobre coste innecesario. Además, cada balanceador de carga requiere su propia dirección IP, en cambio, con Ingress una IP es suficiente.

Un controlador de Ingress es un Pod o conjunto de Pods que se ejecutan en el cluster y su función es asegurarse de que el tráfico entrante se administre del modo que se le haya especificado.

¿Qué es un LoadBalancer?

Un load balancer permite repartir la carga de trabajo entre los diferentes servidores o aplicaciones y puede instalarse en una infraestructura tanto física como virtual. Así pues, los programas de load balancing adoptan la forma de un controlador de entrega de aplicaciones o ADC (del inglés «aplicación delivery controller»), permitiendo al usuario escalar la carga automáticamente en función de las previsiones de tráfico.

El controlador de entrega de aplicaciones 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 cluster. En caso de avería, es posible redirigir el tráfico hacia otro recurso con capacidad para absorberlo. Así pues, existen diferentes configuraciones posibles.

El load balancer interviene entre el visitante y el servidor analizando las peticiones, determinando qué máquina está disponible para responder y, por último, transmitiendo estas peticiones. También es posible añadir servidores en caso de necesidad.

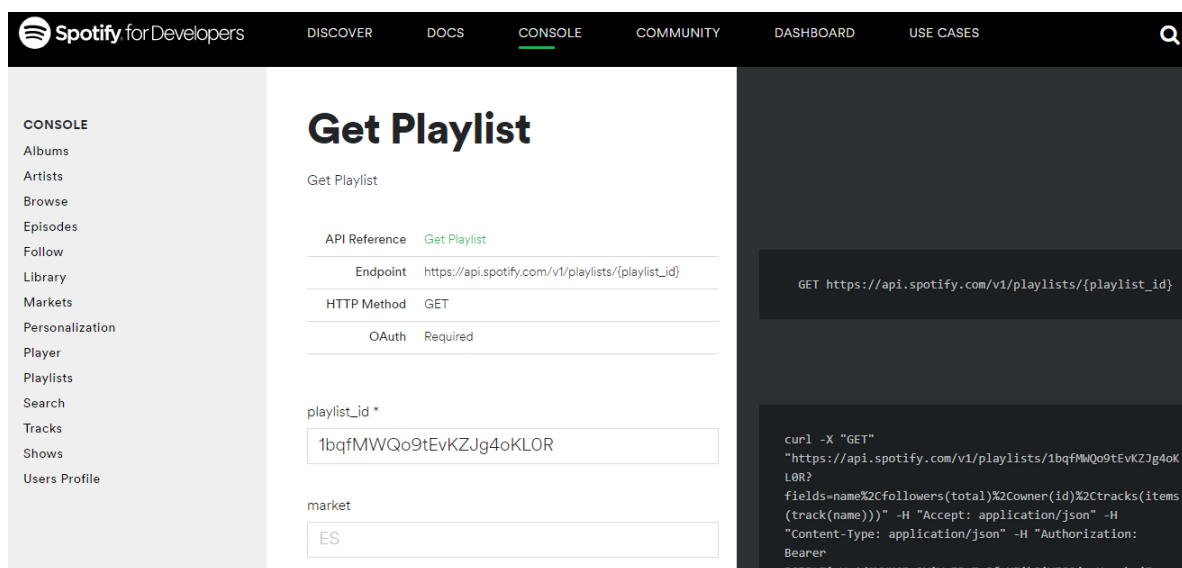
¿Qué es Rancher?

Rancher es una pila de software que se utiliza para gestionar clústeres Kubernetes. Se trata básicamente de un software que DevOps puede utilizar al adoptar el usuario de contenedores. Rancher incluye una distribución completa de Kubernetes, Docker Swarm y Apache Mesos, lo que facilita la gestión de clústeres de contenedores en cualquier plataforma de nube.

Una de las ventajas significativas de Rancher es la capacidad de gestionar múltiples clústeres Kubernetes de forma simplificada. Ofrece una gestión simplificada de múltiples clústeres Kubernetes que pueden ser creados manualmente utilizando la distribución de Kubernetes de Rancher llamada RKE (Rancher Kubernetes Engine) o importados en el panel de gestión del gestor de clústeres.

Una vez definido lo anterior, se explicará un ejemplo utilizando la herramienta Kubernetes para la organización de un contenedor en Docker de una aplicación en Python.

Primero cree el programa de Python que consiste en la verificación y obtención de datos de una playlist a través de la API de Spotify, al colocar el id de la playlist y solicitar un token de autorización en la página de Spotify, aparece al lado derecho la petición que debe realizarse para acceder a los datos de esa playlist:



The screenshot shows the Spotify for Developers console. On the left is a sidebar with a 'CONSOLE' menu and a list of API endpoints including Albums, Artists, Browse, Episodes, Follow, Library, Markets, Personalization, Player, Playlists, Search, Tracks, Shows, and Users Profile. The main area is titled 'Get Playlist' and contains a table with API details:

API Reference	Get Playlist
Endpoint	https://api.spotify.com/v1/playlists/{playlist_id}
HTTP Method	GET
OAuth	Required

Below the table, there are input fields for 'playlist_id' (containing '1bqfMWQo9tEvKZJg4oKLOR') and 'market' (containing 'ES'). On the right side of the console, a dark-themed box displays the HTTP request:

```
GET https://api.spotify.com/v1/playlists/{playlist_id}

curl -X "GET"
"https://api.spotify.com/v1/playlists/1bqfMWQo9tEvKZJg4oKLOR?
fields=name%2Cfollowers(total)%2Cowner(id)%2Ctracks(items
(track(name)))" -H "Accept: application/json" -H
"Content-Type: application/json" -H "Authorization:
Bearer
BQB3t7iokW4d1V4W6G3n2Y1Ve7BcTn56WtIk81V78Qjwdlnu4v3T-
```

Esto lo hice con Flask como se muestra a continuación:

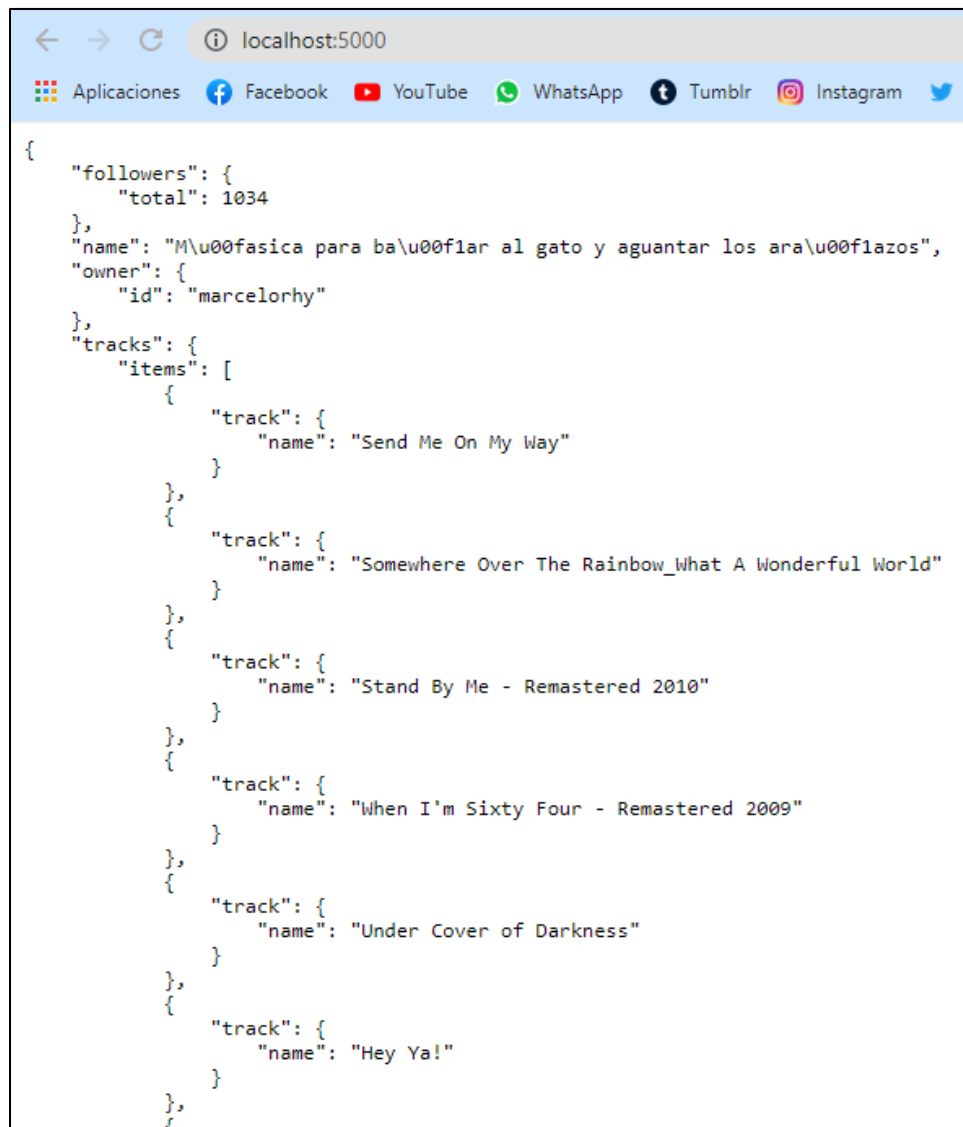
```

main.py > ...
1  from flask import Flask
2  import requests
3  import json
4
5  app = Flask(__name__)
6
7  @app.route('/')
8  def spotify():
9
10     playlistId = "1bqfMWQo9tEvKZJg4oKL0R"
11     endpoint = "https://api.spotify.com/v1/playlists/{playlistId}?fields=name%2Cfollowers(total)%2Cowner("
12     token = "BQCGoFMgvM3gr5CnqdfwXByZstCxGZuhcG612rW_SKuFByeuJ74u9r6H23UxT4i0rg25TYwZLl3l23mK4wuwWFD01mn6j"
13
14     headers = {
15         "Accept": "application/json",
16         "Content-Type": "application/json",
17         "Authorization": "Bearer {token}".format(token=token)
18     }
19
20     resultado = requests.get(endpoint, headers=headers)
21     datos = resultado.json()
22
23     return '<pre>{</pre>'.format(json.dumps(datos, indent=4))
24
25 app.run()

```

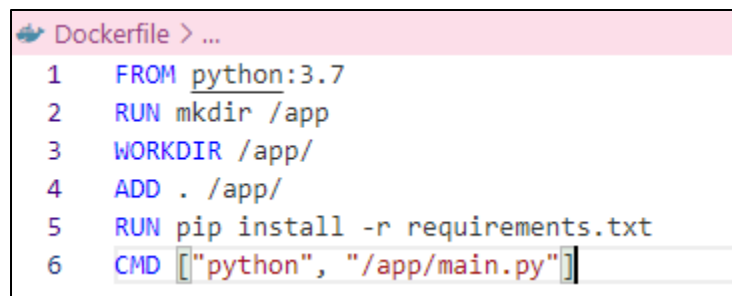
Para ello primero importé la librería de Flask e instancié un objeto Flask con el nombre de app, después con `@app.route('/')` definí el path para visualizar la aplicación que será la raíz, y dentro de una función definí un diccionario con los encabezados necesarios para la petición de la ruta que incluye el token de autorización, y después con requests realizar la petición get del endpoint junto con sus encabezados y retornar los datos de la playlist en un formato json para una sencilla visualización. Fuera de la función con `app.run()` se define que se ejecutará el objeto app de Flask.

Al ejecutar la aplicación en la dirección 127.0.0.1:5000 dada por flask en la ruta raíz como se definió, así es como se muestran los datos de la playlist:



```
{
  "followers": {
    "total": 1034
  },
  "name": "M\u00fasica para ba\u00flar al gato y aguantar los ara\u00f1azos",
  "owner": {
    "id": "marcelorhy"
  },
  "tracks": {
    "items": [
      {
        "track": {
          "name": "Send Me On My Way"
        }
      },
      {
        "track": {
          "name": "Somewhere Over The Rainbow_What A Wonderful World"
        }
      },
      {
        "track": {
          "name": "Stand By Me - Remastered 2010"
        }
      },
      {
        "track": {
          "name": "When I'm Sixty Four - Remastered 2009"
        }
      },
      {
        "track": {
          "name": "Under Cover of Darkness"
        }
      },
      {
        "track": {
          "name": "Hey Ya!"
        }
      }
    ]
  }
}
```

Una vez implementada la aplicación, para meterla en un contenedor de Docker primero creé el archivo Dockerfile:



```
Dockerfile > ...
1 FROM python:3.7
2 RUN mkdir /app
3 WORKDIR /app/
4 ADD . /app/
5 RUN pip install -r requirements.txt
6 CMD ["python", "/app/main.py"]
```

En requirements.txt coloqué flask y requests porque son los módulos que se necesitan instalar para que la aplicación de Python funcione:

```

requirements.txt
1 flask
2 requests

```

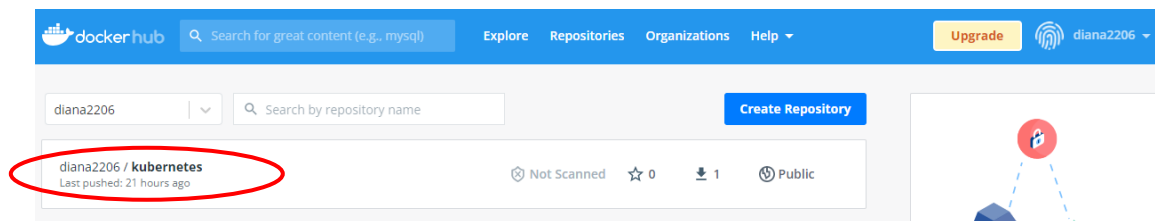
Se construye la imagen (a la cual le llamé “kubernetes”) con el comando:

docker build -t kubernetes .

Una vez que se construye la imagen, ya se puede subir a docker hub para utilizarla posteriormente, para subirla utilicé los siguientes comandos:

docker tag kubernetes diana2206/kubernetes:v1

docker push diana2206/kubernetes:v1



Esta imagen se utiliza en el archivo.yaml, este archivo se crea para poder implementar la aplicación en kubernetes:

```

! deployment.yaml
3 metadata:
4   name: flask-service
5 spec:
6   selector:
7     app: flask-app
8   ports:
9     - protocol: "TCP"
10     port: 6000
11     targetPort: 5000
12   type: LoadBalancer
13
14 ---
15 apiVersion: apps/v1
16 kind: Deployment
17 metadata:
18   name: flask-app
19 spec:
20   selector:
21     matchLabels:
22       app: flask-app
23   replicas: 5
24   template:
25     metadata:
26       labels:
27         app: flask-app
28     spec:
29       containers:
30       - name: flask-app
31         image: diana2206/kubernetes:v1
32         command: [ "/bin/bash", "-ce", "tail -f /dev/null" ]
33         imagePullPolicy: IfNotPresent
34         ports:
35         - containerPort: 5000

```

Este archivo se compone de dos partes, una es para el servicio y la otra es de la implementación, es decir, se está creando un servicio que funcionará como un LoadBalancer y la implementación es la parte de la aplicación. El usuario accederá a través del LoadBalancer y este distribuirá la solicitud en la aplicación. Se define que tendrá 5 replicas o instancias, esto con el fin de que, si alguna llega a fallar, la aplicación seguirá funcionando en otra de ellas. En la parte inferior se observa que este archivo está conectado a la imagen de docker que subí a docker hub que es de la aplicación, por lo que para implementar la aplicación en el cluster de kubernetes se estará usando esta imagen.

Para implementar kubernetes utilicé minikube, entonces una vez iniciado, y al ejecutar este archivo con el servicio y la implementación en kubernetes se crean automáticamente los pods en el cluster. El archivo se ejecuta de la siguiente manera:

kubectl apply -f deployment.yaml

Ahora al entrar al dashboard de minikube se podrán ver los 5 pods que se definieron en el archivo yaml:

Workload Status

Running 1

Deployments

Running 5

Pods

Running 1

Replica Sets

Deployments

Name	Namespace	Images	Labels	Pods	Created
flask-app	default	diana2206/kubernetes:v1	-	5 / 5	a minute ago

Pods

Name	Namespace	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created
flask-app-5bd877c5df-24gln	default	diana2206/kubernetes:v1	app: flask-app pod-template-hash: 5bd877c5df	minikube	Running	0	-	-	a minute ago
flask-app-5bd877c5df-gqkqv	default	diana2206/kubernetes:v1	app: flask-app pod-template-hash: 5bd877c5df	minikube	Running	0	-	-	a minute ago
flask-app-5bd877c5df-qpk5	default	diana2206/kubernetes:v1	app: flask-app pod-template-hash: 5bd877c5df	minikube	Running	0	-	-	a minute ago
flask-app-5bd877c5df-r1ip	default	diana2206/kubernetes:v1	app: flask-app pod-template-hash: 5bd877c5df	minikube	Running	0	-	-	a minute ago
flask-app-5bd877c5df-v57cb	default	diana2206/kubernetes:v1	app: flask-app pod-template-hash: 5bd877c5df	minikube	Running	0	-	-	a minute ago

Al hacer clic a cada pod se puede ver toda su información, como su dirección IP:

Metadata

Name

flask-app-5bd877c5df-24g8n

Namespace

default

Created

Apr 4, 2022

Age

47 seconds ago

UID

fb630c8f-4dff-4c41-b2ff-4c0e99eb47d4

Labels

app: flask-app

pod-template-hash: 5bd877c5df

Resource information

Node

minikube

Status

Running

IP

172.17.0.8

GoS Class

BestEffort

Restarts

0

Service Account

default

Conditions

Type	Status	Last probe time	Last transition time	Reason	Message
Initialized	True	:-	47 seconds ago	-	-
Ready	True	:-	39 seconds ago	-	-
ContainersReady	True	:-	39 seconds ago	-	-
PodScheduled	True	:-	47 seconds ago	-	-

Controlled by

Name

flask-app-5bd877c5df

Kind

replicaset

Pods

5 / 5

Age

47 seconds ago

Labels

app: flask-app

pod-template-hash: 5bd877c5df

Aunque no se puede acceder a un pod por medio de su dirección IP, para eso está el LoadBalancer, pues es el que se comunica internamente con los pods. Cada uno de los pods son instancias de la aplicación asociadas con el LoadBalancer, lo que este hace es distribuir la cantidad de visitas entre los pods de la aplicación por igual, por lo que, si se elimina uno de los pods, Kubernetes automáticamente crea uno nuevo.

Aquí se observa el LoadBalancer creado:

Services							
	Namespace	Labels	Type	Cluster IP	Internal Endpoints	External Endpoints	Created
flask-service	default	-	LoadBalancer	10.110.41.121	flask-service/6000 TCP flask-service/31930 TCP	-	5 minutes ago
flaskapp	default	run: flaskapp	LoadBalancer	10.97.230.192	flaskapp/8080 TCP flaskapp/31546 TCP	-	13 hours ago
kubernetes	default	component: apiserver provider: kubernetes	ClusterIP	10.96.0.1	kubernetes/443 TCP kubernetes/0 TCP	-	14 hours ago

Al hacer clic en él, se despliega su información, en donde se encuentra la dirección IP para ya poder acceder a la aplicación:

Metadata

flask-service

default

Apr. 4, 2022

6 minutes ago

11e6b2cd-3708-4405-acdf-c79b3b14640b

Annotations

kubernetes.io/last-applied-configuration

Resource information

LoadBalancer

Cluster IP

10.110.41.121

Session affinity

None

Selector

app: flask-app

Endpoints

Host	Ports (Name, Port, Protocol)	Node	Ready
172.17.0.3	~unset~:5000,TCP	minikube	true
172.17.0.6	~unset~:5000,TCP	minikube	true
172.17.0.7	~unset~:5000,TCP	minikube	true
172.17.0.8	~unset~:5000,TCP	minikube	true
172.17.0.9	~unset~:5000,TCP	minikube	true

Pods

Name	Namespace	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created
flask-app-5bd877c5df-24gln	default	diana2206/kubernetes:v1	app: flask-app pod-template-hash: 5bd877c5df	minikube	Running	0	-	-	a minute ago
flask-app-5bd877c5df-q6q9w	default	diana2206/kubernetes:v1	app: flask-app pod-template-hash: 5bd877c5df	minikube	Running	0	-	-	a minute ago

Código en GitHub: <https://github.com/diana-castillo/kubernetes.git>

Enlace a la imagen en Docker Hub:

<https://hub.docker.com/repository/docker/diana2206/kubernetes>

Conclusiones:

Esta actividad me pareció un poco más complicada que las anteriores, esto debido a que no tenía muchas opciones para llevarla a cabo en mi computadora y al momento de habilitar la opción de kubernetes en docker desktop y comenzar el cluster en minikube mi computadora se ponía demasiado lenta así que tuve que pedir una prestada. Pero fuera de eso me pareció interesante conocer cómo funciona kubernetes, y considero que puede ser de mucha utilidad a la hora de gestionar aplicaciones más robustas y que necesitan estar ejecutando, pues kubernetes tiene la ventaja de que, si algún elemento falla, la aplicación seguirá corriendo en otro, y creo que esa tolerancia a fallos es bastante eficaz para ejecutar aplicaciones cuyo funcionamiento es crítico.

Referencias:

Rehkopf, M. (s. f.). *¿Qué es Kubernetes?* Atlassian. Recuperado 1 de abril de 2022, de <https://www.atlassian.com/es/continuous-delivery/microservices/kubernetes>

Red Hat. (2020, 27 marzo). *¿Qué es Kubernetes?* Recuperado 1 de abril de 2022, de <https://www.redhat.com/es/topics/containers/what-is-kubernetes>

Pujol, J. (2020, 30 mayo). *Ingress en Kubernetes Desmitificado: ¿Qué lo diferencia de un NodePort o un LoadBalancer?* Medium. Recuperado 1 de abril de 2022, de <https://desarrollofront.medium.com/ingress-en-kubernetes-desmitificado-qu%C3%A9-lo-diferencia-de-un-nodeport-o-un-loadbalancer-b0cf060a6f8a>

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

Todo lo que necesita saber sobre Rancher: gestión de Kubernetes para empresas. (2021, 4 mayo). SiXe Ingeniería. Recuperado 1 de abril de 2022, de <https://sixe.es/noticias/suse-rancher-kubernetes-toda-la-informacion>

Digitalthinking with sotobotero. (2021, 9 marzo). *Crear un pod y un servicio en kubernetes* [Vídeo]. YouTube. Recuperado 3 de abril de 2022, de <https://www.youtube.com/watch?v=P98ySHpb3r4>

Dev Sense. (2020, 22 agosto). *Deploying Any Dockerized Application To Kubernetes | Ashutosh Hathidara | #kubernetes.* YouTube. Recuperado 3 de abril de 2022, de <https://www.youtube.com/watch?v=XQNNAeyMAkk&t=422s>