



Universidad de Guadalajara  
Centro Universitario de Ciencias Exactas e Ingenierías  
División de Tecnologías para la Integración Ciber-Humana  
Departamento de Ciencias Computacionales  
Ingeniería en Computación



# Kubernetes

*Ejemplo de uso de Kubernetes.*

## Alumno

Hernández Cortez Kevin Uriel.

217734547.

## Materia

Computación Tolerante a Fallas.

D06

## Profesor

Lopez Franco Michel Emanuel.

## Fecha de entrega

Lunes 29 de abril de 2024.

## Introducción

En esta tarea, aprenderé a crear y desplegar una aplicación Node.js en Kubernetes utilizando Minikube. Utilizaremos Docker para contenerizar nuestra aplicación Node.js y luego la desplegaremos en un clúster de Kubernetes utilizando Minikube. En los pasos se verán la creación de la aplicación Node.js, la creación de un contenedor Docker, el despliegue de la aplicación en Kubernetes y la exposición del servicio para que pueda ser accesible externamente.

## Desarrollo

### Códigos

```
- PACKAGE.JSON
{
  "name": "kubernetes-app",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "express": "^4.19.2"
  }
}

- DOCKERFILE
FROM node:13-alpine

WORKDIR /app

COPY package.json package-lock.json ./

RUN npm install --production

COPY . .

EXPOSE 3000

CMD node index.js

- INDEX.JS
const express = require('express')
const os = require('os')

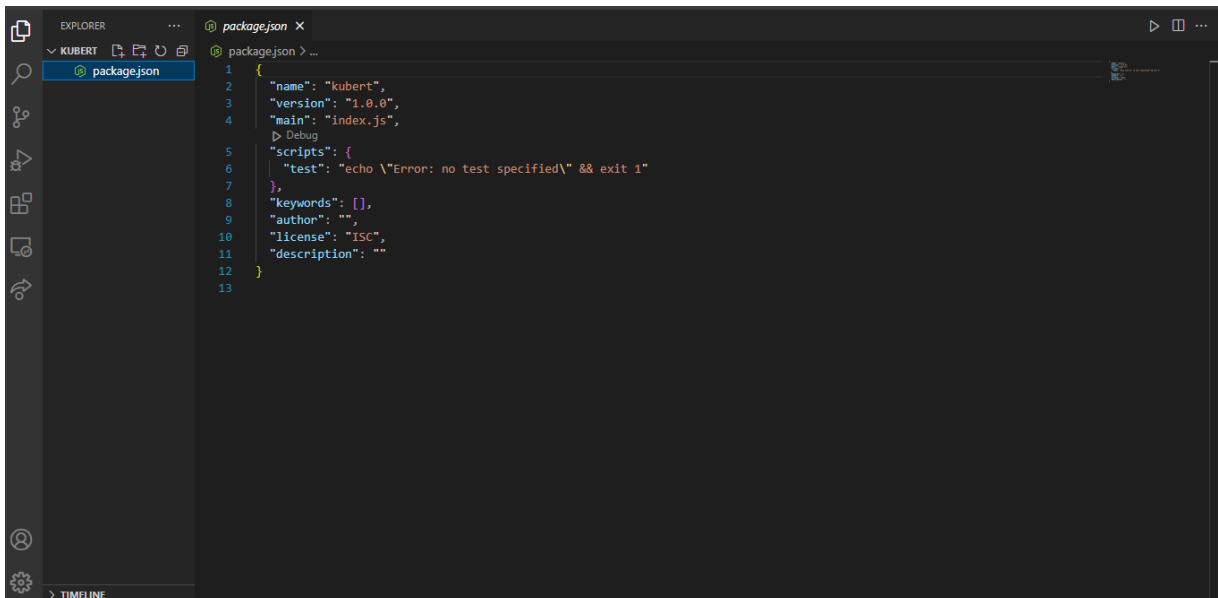
const app = express()
```

```
app.get('/', (req, res) => {  
    res.send(`Hello from ${os.hostname()}!`)  
})  
  
const port = 3000  
app.listen(port, () => console.log(`listening on port ${port}`))
```

### Explicación de programa

### Node App

```
C:\Users\52332\Desktop\Kubert>npm init -y  
Wrote to C:\Users\52332\Desktop\Kubert\package.json:  
  
{  
  "name": "kubert",  
  "version": "1.0.0",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "description": ""  
}
```



## Kubernetes

Se crea el archivo llamado *package.json* con el comando *npm init -y* en el directorio a elección (en este caso, *Kubert*).

```
C:\Users\52332\Desktop\Kubert>npm install express

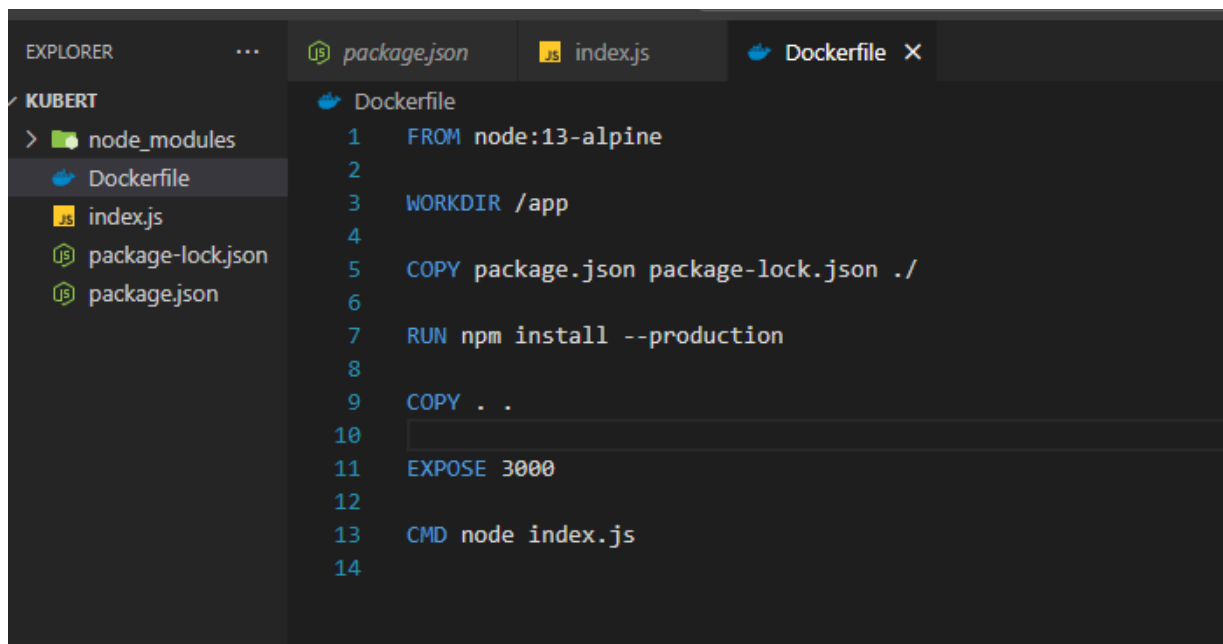
added 64 packages, and audited 65 packages in 11s

12 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Luego, se instala Express como una dependencia para el proyecto utilizando el comando *npm install express*. Creamos finalmente un archivo llamado *index.js*.

## Docker



```
EXPLORER    ...    package.json    index.js    Dockerfile X
KUBERT
  > node_modules
  Dockerfile
  index.js
  package-lock.json
  package.json

Dockerfile
1 FROM node:13-alpine
2
3 WORKDIR /app
4
5 COPY package.json package-lock.json ./
6
7 RUN npm install --production
8
9 COPY . .
10
11 EXPOSE 3000
12
13 CMD node index.js
14
```

Creamos un archivo llamado Dockerfile en el directorio de nuestra aplicación.

- Utilizamos la imagen *node:13-alpine* como base para nuestro contenedor.
- Configuramos el directorio de trabajo como */app*.
- Copiamos los archivos *package.json* y *package-lock.json* al directorio de trabajo.
- Ejecutamos *npm install --production* para instalar las dependencias de producción.
- Copiamos el resto de los archivos de la aplicación al contenedor.
- Exponemos el puerto 3000 en el contenedor.
- Definimos el comando *node index.js* como el comando por defecto para ejecutar la aplicación.

```

C:\Users\52332\Desktop\Kubert>docker build -t kevinhc47/node-hello-app .
[+] Building 30.2s (11/11) FINISHED
=> [internal] load build definition from Dockerfile                                docker:default
=> => transferring dockerfile: 198B                                              0.1s
=> [internal] load metadata for docker.io/library/node:13-alpine                 0.0s
=> [auth] library/node:pull token for registry-1.docker.io                      3.0s
=> [internal] load .dockerignore                                                 0.0s
=> => transferring context: 2B                                                  0.1s
=> [1/5] FROM docker.io/library/node:13-alpine@sha256:527c70f74817f6f6b5853588c28de33459178ab72421f1fb7b63a281a 17.3s
=> => resolve docker.io/library/node:13-alpine@sha256:527c70f74817f6f6b5853588c28de33459178ab72421f1fb7b63a281a 0.1s
=> => sha256:4d6b35f8d14b2f476005883f9dec0a93c83213838fd3def3db0d130b1b148653 1.16kB / 1.16kB 0.0s
=> => sha256:8216bf4583a5d42edd579a3b77f66a8fe127c4acb5eabecf085d911786f3e68 6.77kB / 6.77kB 0.0s
=> => sha256:527c70f74817f6f6b5853588c28de33459178ab72421f1fb7b63a281ab670258 1.65kB / 1.65kB 0.0s
=> => sha256:cbdbe7a5bc2a134ca8ec91be58565ec07d037386d1f1d8385412d224deafca08 2.81MB / 2.81MB 3.7s
=> => sha256:780514bed1adda642a7f7008ea5b1477bb78a23e3f76eabbdde21031b2185a2f 35.30MB / 35.30MB 12.1s
=> => sha256:5d74fb112a7d313344f9f679ccb975062377464235fac36e1b952aa2b125d5e2 2.24MB / 2.24MB 4.6s
=> => extracting sha256:cbdbe7a5bc2a134ca8ec91be58565ec07d037386d1f1d8385412d224deafca08 0.6s
=> => sha256:4b9536424fa1675675e7edd59ae9a92573f0c010c5458733a493e2c3e1daf1ae 283B / 283B 4.4s
=> => extracting sha256:780514bed1adda642a7f7008ea5b1477bb78a23e3f76eabbdde21031b2185a2f 4.0s
=> => extracting sha256:5d74fb112a7d313344f9f679ccb975062377464235fac36e1b952aa2b125d5e2 0.2s
=> => extracting sha256:4b9536424fa1675675e7edd59ae9a92573f0c010c5458733a493e2c3e1daf1ae 0.0s
=> [internal] load build context                                                8.1s
=> => transferring context: 2.21MB                                              8.0s
=> [2/5] WORKDIR /app                                                         8.0s
=> [3/5] COPY package.json package-lock.json ./                             0.1s
=> [4/5] RUN npm install --production                                         8.0s
=> [5/5] COPY . .                                                            0.2s

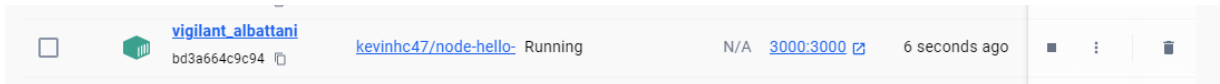
```

Ejecutamos el comando `docker build -t kevinhc47/node-hello-app .` para construir la imagen de Docker.

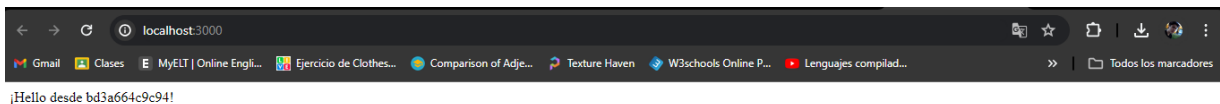
```

C:\Users\52332\Desktop\Kubert>docker run --rm -d -p 3000:3000 kevinhc47/node-hello-app
bd3a664c9c9456984ee840a8e08ba04f0780a82272b6d9534d9abc727673633d

```



Ejecutamos el contenedor utilizando el comando `docker run --rm -d -p 3000:3000 kevinhc47/node-hello-app`.



## Kubernetes

El contenedor se ejecuta en segundo plano y se mapea el puerto 3000 del contenedor al puerto 3000 del host.

```
C:\Users\52332\Desktop\Kubert>docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
bd3a664c9c94	kevinhc47/node-hello-app	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:3000->3000/tcp

Utilizamos *docker ps* para encontrar el ID del contenedor en ejecución.

```
C:\Users\52332\Desktop\Kubert>docker stop bd3a664c9c9456984ee840a8e08ba04f0780a82272b6d9534d9abc727673633d
bd3a664c9c9456984ee840a8e08ba04f0780a82272b6d9534d9abc727673633d
```

Luego, detenemos el contenedor utilizando *docker stop CONTAINER\_ID*.

```
C:\Users\52332\Desktop\Kubert>docker push kevinhc47/node-hello-app
Using default tag: latest
The push refers to repository [docker.io/kevinhc47/node-hello-app]
cd6d4e39d221: Pushed
150d1215460e: Pushed
3f2cf7112582: Pushed
7b5dbf92dd51: Pushed
629960860aca: Mounted from library/node
f019278bad8b: Mounted from library/node
8ca4f4055a70: Mounted from library/node
3e207b409db3: Mounted from library/node
latest: digest: sha256:b17b2383caab6a6ccb2b408b8973b63b50cccbdbfda33a372b4e80030b51bb88 size: 1994
```

Kevin Uriel Hernández Cortez [Edit profile](#)

Community User Joined April 12, 2024

Repositories Starred Contributed

Displaying 1 to 1 repositories

kevinhc47/node-hello-app

By kevinhc47 · Updated a few seconds ago

Image

0 0

docker

Explore Containers Account Billing Resources Blog Support Feedback Company About Us

Utilizamos *docker push kevinhc47/node-hello-app* para subir la imagen a DockerHub.

## Kubernetes

```

PS C:\WINDOWS\system32> minikube start
W0427 18:54:33.882335 14444 main.go:291] Unable to resolve the current Docker CLI context "default": context "default"
: context not found: open C:\Users\52332\.docker\contexts\meta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a
Bf0688f\meta.json: The system cannot find the path specified.
* minikube v1.33.0 en Microsoft Windows 11 Home Single Language 10.0.22621.3447 Build 22621.3447
* Controlador docker seleccionado automáticamente
* Using Docker Desktop driver with root privileges
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.43 ...
* Descargando Kubernetes v1.30.0 ...
  > preloaded-images-k8s-v18-v1...: 342.90 MiB / 342.90 MiB 100.00% 2.27 Mi
  > gcr.io/k8s-minikube/kicbase...: 480.29 MiB / 480.29 MiB 100.00% 2.53 Mi
* Creating docker container (CPUs=2, Memory=3000MB) ...
* Preparando Kubernetes v1.30.0 en Docker 26.0.1...
  - Generando certificados y llaves
  - Iniciando plano de control
  - Configurando reglas RBAC...
* Configurando CNI bridge CNI ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Complementos habilitados: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
PS C:\WINDOWS\system32>

```

Iniciamos un clúster de Kubernetes local utilizando Minikube con el comando *minikube start*

```

C:\Users\52332\Desktop\Kubert>kubectl create deployment --image kevinhc47/node-hello-app node-app
deployment.apps/node-app created

```

Creamos un deployment para nuestra aplicación utilizando el comando *kubectl create deployment --image kevinhc47/node-hello-app node-app*.

```

C:\Users\52332\Desktop\Kubert>kubectl scale deployment node-app --replicas 3
deployment.apps/node-app scaled

```

Escalamos el deployment a 3 réplicas con *kubectl scale deployment node-app --replicas 3*.

```

C:\Users\52332\Desktop\Kubert>kubectl expose deployment node-app --type=NodePort --port 3000
service/node-app exposed

```

Exponemos el deployment como un servicio NodePort utilizando *kubectl expose deployment node-app --type=NodePort --port 3000*.

```

C:\Users\52332\Desktop\Kubert>kubectl get services

```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	17m
node-app	NodePort	10.111.36.7	<none>	3000:32379/TCP	11s

Utilizamos *kubectl get services* para obtener información sobre el servicio recién creado.

```

C:\Users\52332\Desktop\Kubert>kubectl get nodes -o wide

```

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION
minikube	Ready	control-plane	18m	v1.30.0	192.168.49.2	<none>	Ubuntu 22.04.4 LTS	5.10.102.1-microso
ft-standard-WSL2		docker://26.0.1						

## Kubernetes

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: "2024-04-28T01:19:49Z"
  labels:
    app: node-app
    name: node-app
    namespace: default
    resourceVersion: "1283"
    uid: 77ea67ee-c31e-419d-93b7-0bd0f701e3ab
spec:
  clusterIP: 10.111.36.7
  clusterIPs:
  - 10.111.36.7
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - nodePort: 32379
    port: 3000
    protocol: TCP
    targetPort: 3000
  selector:
    app: node-app
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

```
- nodePort: 32379
  port: 80
  protocol: TCP
  targetPort: 3000
selector:
  app: node-app
sessionAffinity: None
type: LoadBalancer
status:
  loadBalancer: {}
```

Ejecutamos *kubectl edit service node-app* y cambiamos el tipo del servicio a LoadBalancer y el puerto a 80.

```
C:\Users\52332\Desktop\Kubert>kubectl get service
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kubernetes    ClusterIP     10.96.0.1     <none>         443/TCP          25m
node-app      LoadBalancer 10.111.36.7   <pending>      80:32379/TCP     7m21s
```

Utilizamos *kubectl get service* para obtener la dirección IP del Load Balancer.

## Conclusión

En esta tarea, he aprendido cómo crear, contenerizar y desplegar una aplicación Node.js en Kubernetes utilizando Minikube. Comencé creando una aplicación Node.js simple para finalmente exponer el servicio para que fuera accesible desde fuera del clúster. Este ejercicio me proporciona una comprensión básica sobre cómo trabajar con aplicaciones en contenedores y desplegarlas en un entorno de Kubernetes.



## **Bibliografía.**

- DigitalOcean. (2020, 27 mayo). *Production-ready Node.js on Kubernetes* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=T4lp6wtS--4>
- Nasser, K. (2020, 29 septiembre). *How to Deploy a Resilient Node.js Application on Kubernetes From Scratch*. DigitalOcean.  
<https://www.digitalocean.com/community/tech-talks/how-to-deploy-a-resilient-node-js-application-on-kubernetes-from-scratch>