Informe de actividad grupal: Construcción de imágenes Docker y despliegue en un clúster de Kubernetes

**Integrantes**

Martínez Flores, Ronny Alexander

Brito Casanova, Geovanny José

Guambo Heredia, José Luis

Sangoquiza Aviles, Alexis Dario

**Tabla de Contenido**

[**Introducción 3**](#_6palkh4a0pnk)

[**Aplicación para contenerización**](#_mtaotmm5d49u) **3**

[Estructura de aplicación](#_5xqsodjyhpty) 3

[Contenido de archivos de aplicación](#_yx39slrj2ba7) 3

[Ejecución de aplicación 5](#_ddy7jrgsjle7)

[**Creación y publicación de la imagen 5**](#_8oqfy9y6zhmx)

[Generación de la imagen con el fichero Dockerfile 5](#_dhylj0bgky59)

[Publicación de la imagen en Docker Hub 6](#_g56umthwiuk7)

[**Definir namespace, contexto, pod y servicio para MongoDB 7**](#_a06s3fia00x4)

[La estructura del proyecto se modificó para organizar los archivos de mejor manera. 7](#_wosj3ts7angq)

[Definir namespace para los recursos 8](#_44x4fsz29edb)

[Crear contexto asociado al namespace 8](#_dhrxj061lq02)

[Definir correctamente el pod 8](#_wcer8gq610o7)

[Definir el servicio para MongoDB 9](#_r7q1tm34l4fg)

[**Definir ReplicaSet, LoadBalancer en pod de NodeJs. 9**](#_qwntxvpog39z)

[Definir ReplicaSet 9](#_7qopk4aj0268)

[Definir servicio de LoadBalancer para aplicación de Nodejs 10](#_7uxoj3p3a6l)

[Definir PersistentVolume y PersistentVolumeClaim 10](#_le6vt8vggf9k)

[Actualizar definición de pod de MongoDB 13](#_tmsmfams8gq2)

**Conclusiones** 14

[**Tabla de valoración individual 15**](#_11rnxegb8n0e)

**Tabla de Ilustración**

[Ilustración 1 Estructura de la aplicación 3](#_Toc169741656)

[Ilustración 2 Contribuyentes en GitHub 3](#_Toc169741657)

[Ilustración 3 Ejecución de la aplicación 5](#_Toc169741658)

[Ilustración 4 Generación de la imagen con el fichero Dockerfile 5](#_Toc169741659)

[Ilustración 5 Inicio de sesión y etiqueta en Docker Hub 6](#_Toc169741660)

[Ilustración 6 Publicación de imagen 6](#_Toc169741661)

[Ilustración 7 Imagen publicada en DockerHub 6](#_Toc169741662)

[Ilustración 8 Definición de namespace, contexto, pod y servicio para MongoDB 7](#_Toc169741663)

[Ilustración 9 Definición de namespace para los recursos 7](#_Toc169741664)

[Ilustración 10 Definición correctamente el pod 8](#_Toc169741665)

[Ilustración 11 Definición el servicio para MongoDB 8](#_Toc169741666)

[Ilustración 12 Definición de ReplicaSet 9](#_Toc169741667)

[Ilustración 13 Definición servicio de LoadBalancer para aplicación de Nodejs 9](#_Toc169741668)

[Ilustración 14 Configuración del Persistentvolume (Pv) 10](#_Toc169741669)

[Ilustración 15 Aplicación la configuración del PersistentVolume 10](#_Toc169741670)

[Ilustración 16 Configuración del Persistentvolumeclaim (Pvc) 10](#_Toc169741671)

[Ilustración 17 Aplicación de la configuración del PersistentVolumeClaim 10](#_Toc169741672)

[Ilustración 18 Verificación El Estado De Los Recursos 11](#_Toc169741673)

[Ilustración 19 Creación un Pod para probar el PVC: 11](#_Toc169741674)

[Ilustración 20 Aplicación configuración del Pod. 11](#_Toc169741675)

[Ilustración 21 Verificación que el Pod esté en estado Running. 12](#_Toc169741676)

[Ilustración 22 Acceso al contenedor dentro del Pod para la creación de archivos en el volumen montado 12](#_Toc169741677)

[Ilustración 23 Despliegue a la nube de Google Cloud, usando el servicio de Kubernetes 12](#_Toc169741678)

[Ilustración 24 Validación del despliegue por medio del sdk de Google Cloud 13](#_Toc169741679)

[Ilustración 25 Prueba del funcionamiento del balanceador de carga 13](#_Toc169741680)

# **Introducción**

En esta actividad se realizó la contenerización y el despliegue de una aplicación Node.js con MongoDB en un clúster de Kubernetes. La actividad se centra en la creación de imágenes Docker, su publicación en Docker Hub y el despliegue de los recursos en Kubernetes, asegurando la escalabilidad y la eficiencia en los procesos de desarrollo y despliegue.

# **Aplicación para contenerización**

#### Estructura de aplicación

La aplicación Node.js-MongoDB utilizada para esta actividad tiene la siguiente estructura:



Ilustración 1 Estructura de la aplicación

La estructura incluye los directorios y archivos necesarios para el desarrollo y despliegue de la aplicación. El repositorio del proyecto está alojado en [alex95mf/unir-k8s (github.com)](https://github.com/alex95mf/unir-k8s).



Ilustración 2 Contribuyentes en GitHub

#### Contenido de archivos de aplicación

A continuación, se presenta el contenido y una breve descripción de cada archivo que compone la aplicación Node.js-MongoDB.

server.js: Este archivo es el punto de entrada de la aplicación. Configura y arranca el servidor Express.

1. const express = require('express');

2. const connectDB = require('./db');

3. const app = express();

4. const PORT = 3000;

5. connectDB();

6. app.get('/', (req, res) => {

7. res.send('Hello World!');

8. });

9. app.listen(PORT, () => {

10. console.log(`Server is running on http://localhost:${PORT}`);

11. });

12.

db.js: Este archivo se encarga de la conexión a la base de datos MongoDB.

1. const mongoose = require('mongoose');

2. const connectDB = async () => {

3. try {

4. await mongoose.connect('mongodb://mongo:27017/mydatabase', {

5. useNewUrlParser: true,

6. useUnifiedTopology: true

7. });

8. console.log('MongoDB connected...');

9. } catch (err) {

10. console.error(err.message);

11. process.exit(1);

12. }

13. };

14. module.exports = connectDB;

15.

Dockerfile: Este archivo define las etapas para construir la imagen Docker de la aplicación.

1. # Etapa de construcción

2. FROM node:14-alpine AS builder

3. # Crear y cambiar al directorio de la aplicación

4. WORKDIR /usr/src/app

5. # Instalar dependencias de producción

6. COPY package\*.json ./

7. RUN npm install --only=production

8. # Copiar los archivos de la aplicación

9. COPY . .

10. # Etapa final

11. FROM node:14-alpine

12. # Crear y cambiar al directorio de la aplicación

13. WORKDIR /usr/src/app

14. # Copiar los archivos necesarios desde la etapa de construcción

15. COPY --from=builder /usr/src/app /usr/src/app

16. # Crear un usuario no root y cambiar a él

17. RUN addgroup -S appgroup && adduser -S appuser -G appgroup

18. USER appuser

19. # Exponer el puerto en el que la aplicación está escuchando

20. EXPOSE 3000

21. # Ejecutar la aplicación

22. CMD ["node", "server.js"]

23.

package.json: Define las dependencias y scripts de la aplicación.

1. {

2. "name": "nodejs-mongodb-app",

3. "version": "1.0.0",

4. "description": "A simple Node.js app with MongoDB",

5. "main": "server.js",

6. "scripts": {

7. "start": "node server.js"

8. },

9. "dependencies": {

10. "express": "^4.17.1",

11. "mongoose": "^5.12.3"

12. }

13. }

14.

#### Ejecución de aplicación

Para ejecutar la aplicación localmente, se utiliza el siguiente comando: ***npm start***

Esto arranca el servidor en el puerto 3000, verificando que la configuración es correcta antes de proceder a la contenerización y despliegue en Kubernetes.

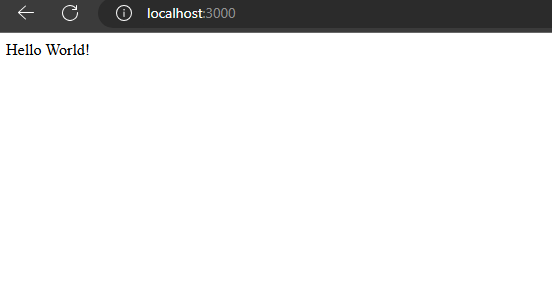


Ilustración 3 Ejecución de la aplicación

Se observa el correcto funcionamiento de la aplicación.

# **Creación y publicación de la imagen**

#### Generación de la imagen con el fichero Dockerfile

Se utilizó el siguiente comando para construir la imagen Docker a partir del Dockerfile: ***docker build -t nodejs-mongodb-app:latest .***

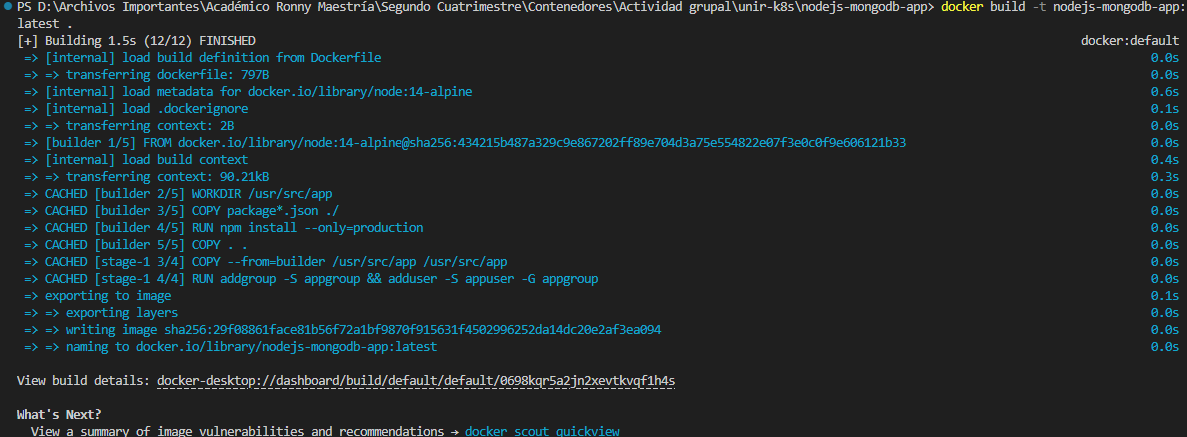


Ilustración 4 Generación de la imagen con el fichero Dockerfile

#### Publicación de la imagen en Docker Hub

Para publicar la imagen en Docker Hub, se siguieron los siguientes pasos:

1. Iniciar sesión en Docker Hub y Etiquetamos la imagen:

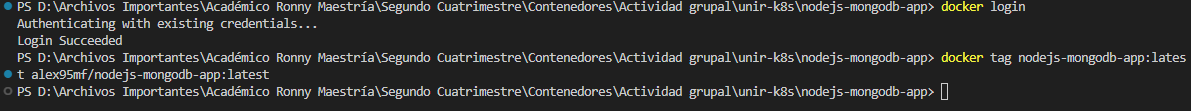


Ilustración 5 Inicio de sesión y etiqueta en Docker Hub

1. Publicar la imagen:

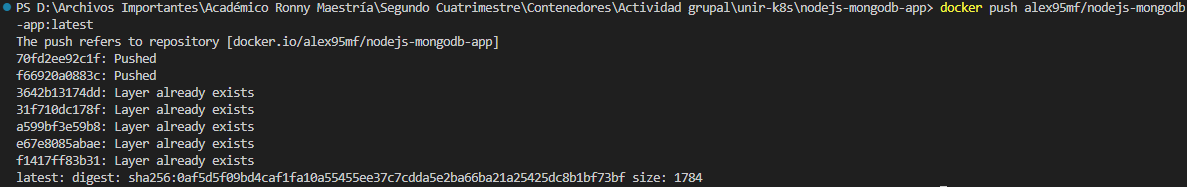


Ilustración 6 Publicación de imagen

Imagen publicada satisfactoriamente en DockerHub:

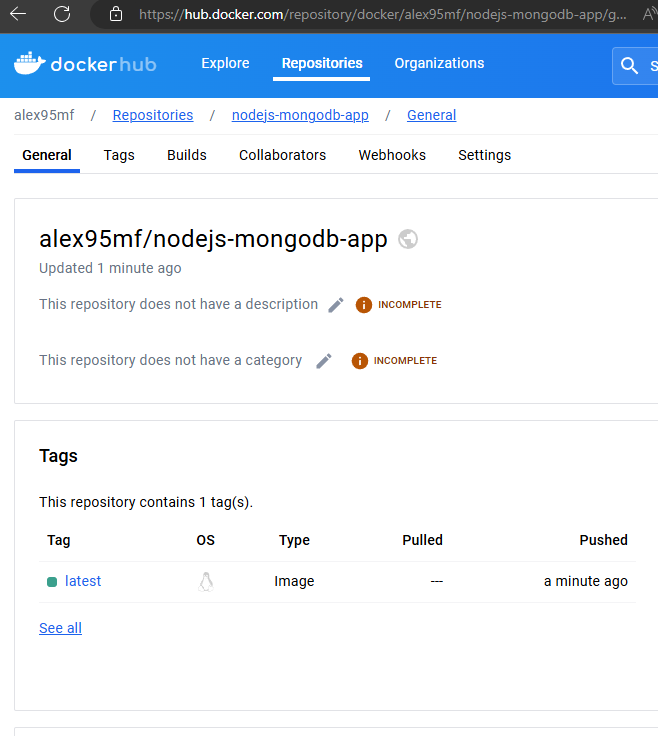


Ilustración 7 Imagen publicada en DockerHub

# **Definir namespace, contexto, pod y servicio para MongoDB**

#### La estructura del proyecto se modificó para organizar los archivos de mejor manera.

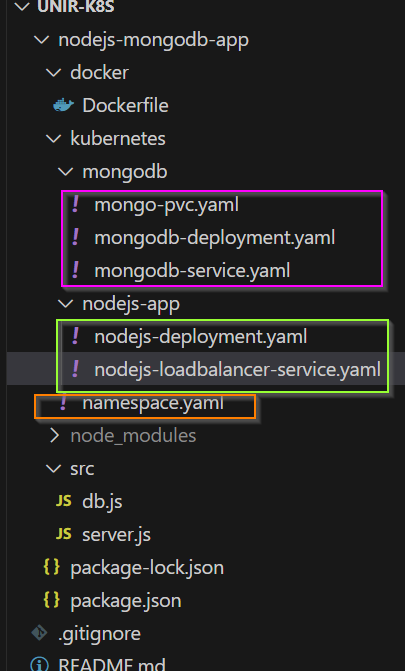


Ilustración 8 Definición de namespace, contexto, pod y servicio para MongoDB

#### Definir namespace para los recursos

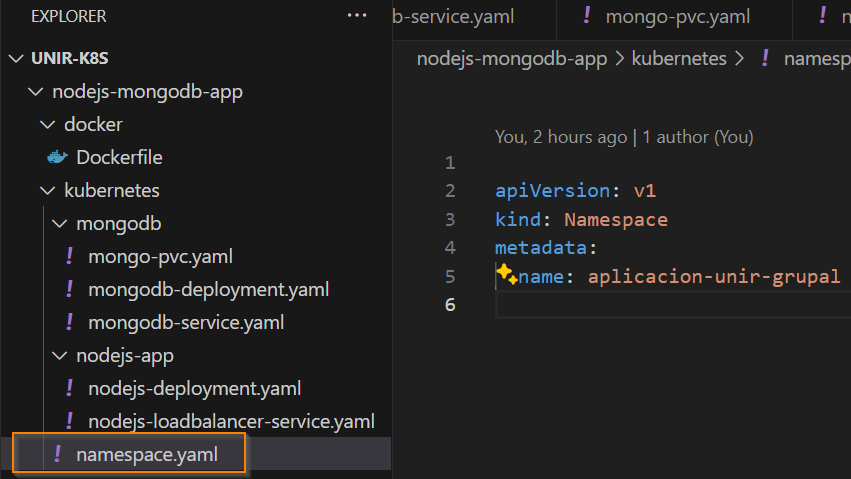


Ilustración 9 Definición de namespace para los recursos

#### Crear contexto asociado al namespace

#### Definir correctamente el pod

Definimos en la carpeta mongoDb, el pod y el servicio. El archivo correspondiente al pod, denominado mongodb-deployment.yaml

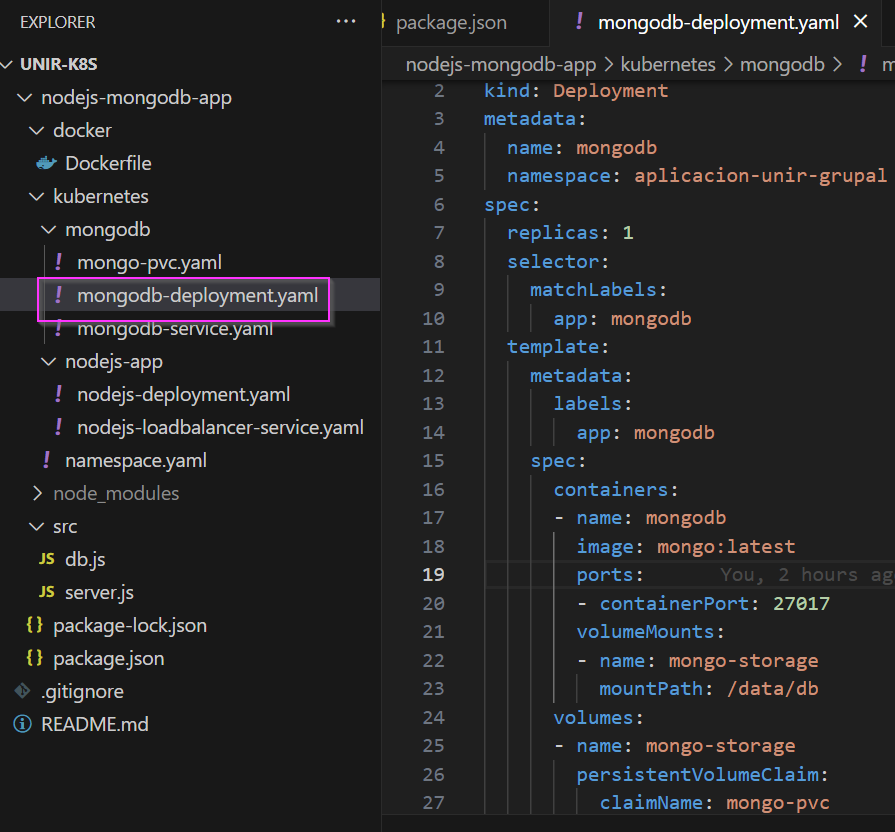


Ilustración 10 Definición correctamente el pod

#### Definir el servicio para MongoDB

Y el archivo correspondiente al servicio

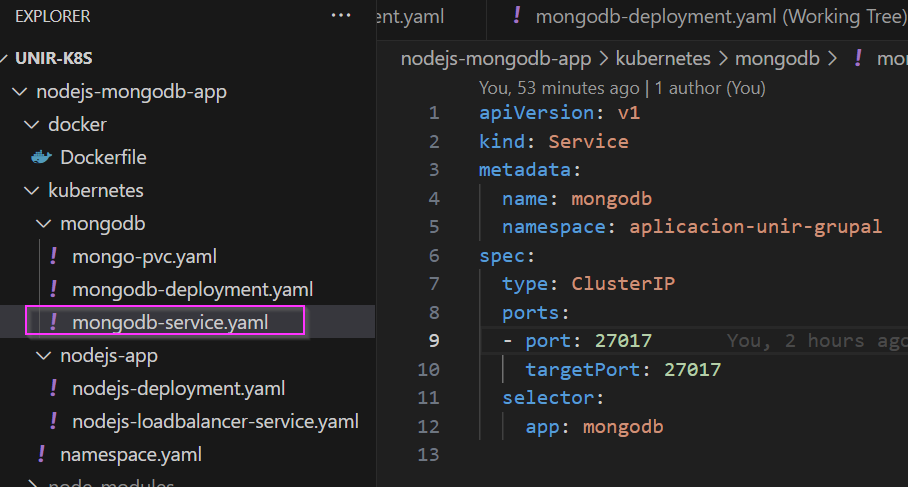


Ilustración 11 Definición el servicio para MongoDB

# **Definir ReplicaSet, LoadBalancer en pod de NodeJs.**

#### Definir ReplicaSet

Definimos el set de réplica en el archivo nodejs-deployment.yaml. El número de réplicas es de dos.

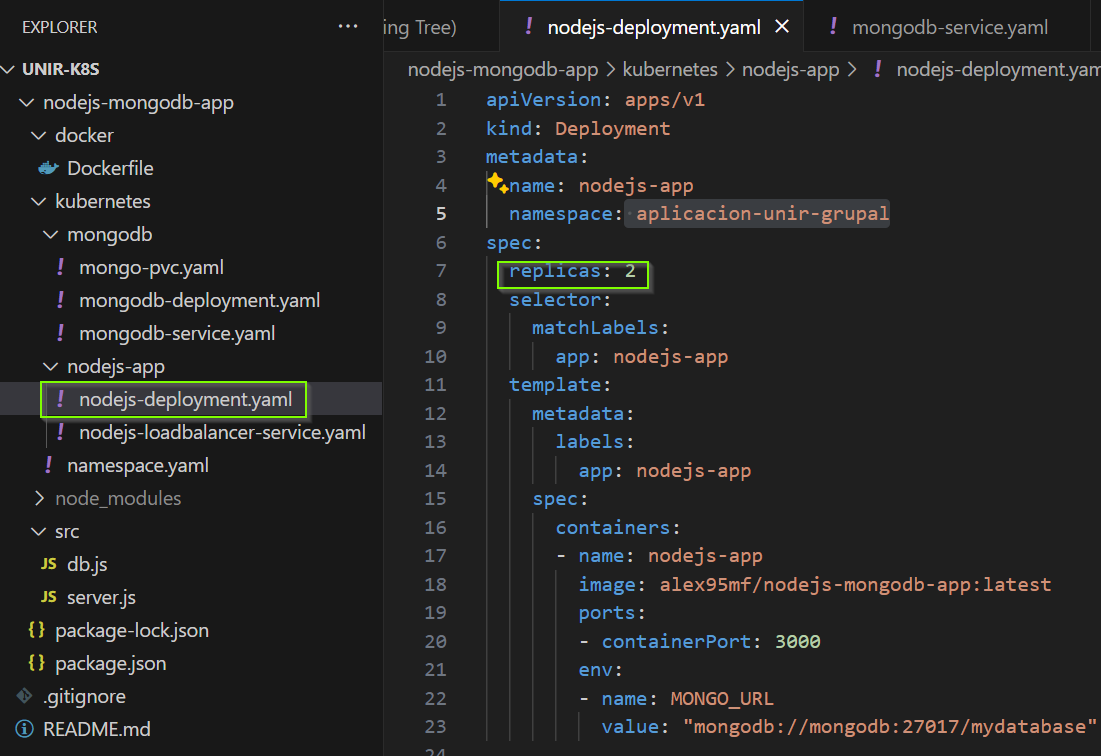


Ilustración 12 Definición de ReplicaSet

#### Definir servicio de LoadBalancer para aplicación de Nodejs

El balance de carga se realiza en el archivo nodejs-loadbalancer-service.yaml

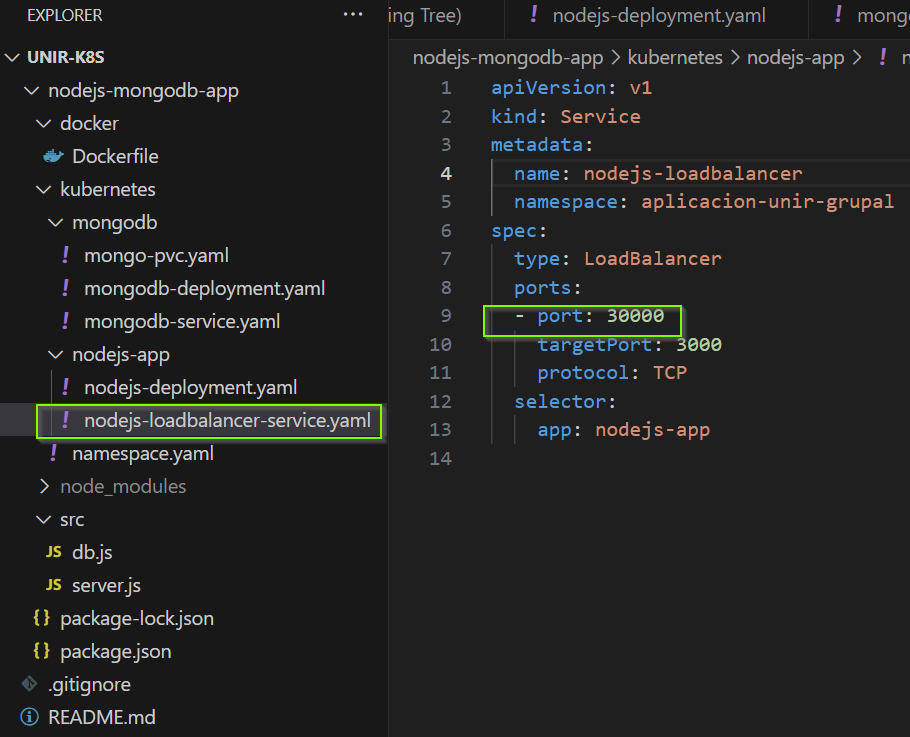


Ilustración 13 Definición servicio de LoadBalancer para aplicación de Nodejs

#### Definir PersistentVolume y PersistentVolumeClaim

**Configurar El Persistentvolume (Pv)**

Crear el archivo YAML para el PersistentVolume (mongo-pv.yaml): Se determina 1GB de espacio para el almacenamiento y directorio local.



Ilustración 14 Configuración del Persistentvolume (Pv)

Aplicar la configuración del PersistentVolume

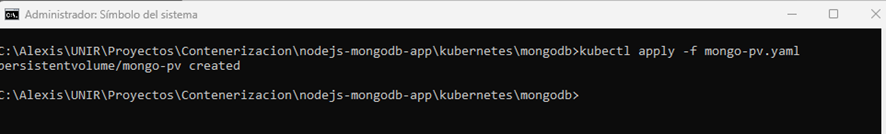


Ilustración 15 Aplicación la configuración del PersistentVolume

**Configurar El Persistentvolumeclaim (Pvc)**

Crear el archivo YAML para el PersistentVolumeClaim (mongo-pvc.yaml): Se determina 1GB de espacio para el almacenamiento y directorio local.

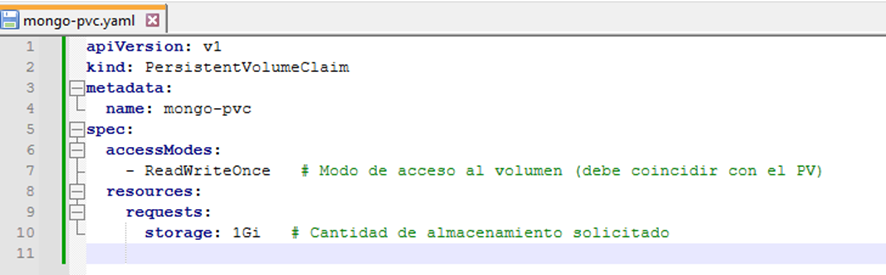


Ilustración 16 Configuración del Persistentvolumeclaim (Pvc)

Aplicar la configuración del PersistentVolumeClaim

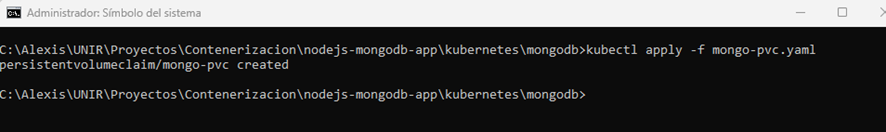
****

Ilustración 17 Aplicación de la configuración del PersistentVolumeClaim

**Verificar El Estado De Los Recursos**

Verificar que el PersistentVolume (PV) y el PersistentVolumeClaim (PVC) están disponibles. Se asegura que ambos recursos están en estado **Bound**, lo que indica que el PVC ha vinculado con éxito un PV disponible.

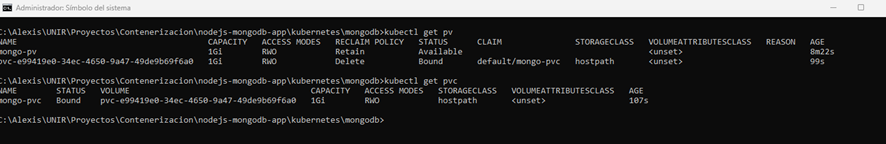


Ilustración 18 Verificación El Estado De Los Recursos

**Pruebas**

- **Crear un Pod para probar el PVC:**

Creación de archivo test-pod.yaml. Este Pod montará el volumen del PVC **mongo-pvc** en **/StoragePV** dentro del contenedor.

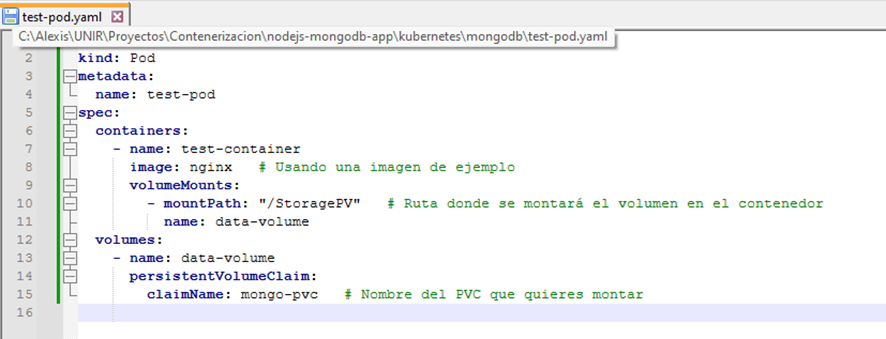


Ilustración 19 Creación un Pod para probar el PVC:

Aplicar configuración del Pod.

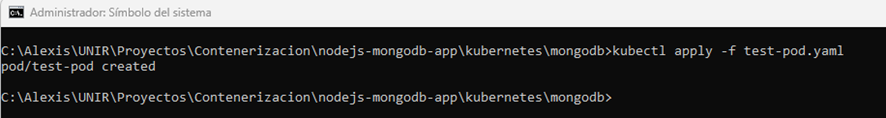


Ilustración 20 Aplicación configuración del Pod.

Verificar que el Pod esté en estado **Running.**

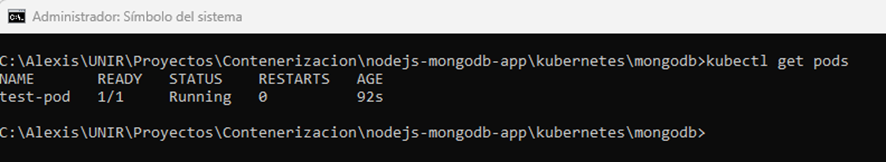


Ilustración 21 Verificación que el Pod esté en estado Running.

Acceder al contenedor dentro del Pod para la creación de archivos en el volumen montado.

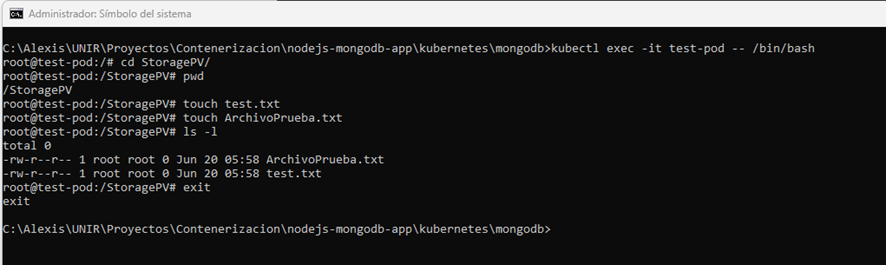


Ilustración 22 Acceso al contenedor dentro del Pod para la creación de archivos en el volumen montado

#### Actualizar definición de pod de MongoDB

Realizamos el despliegue a la nube de Google Cloud, usando el servicio de Kubernetes.

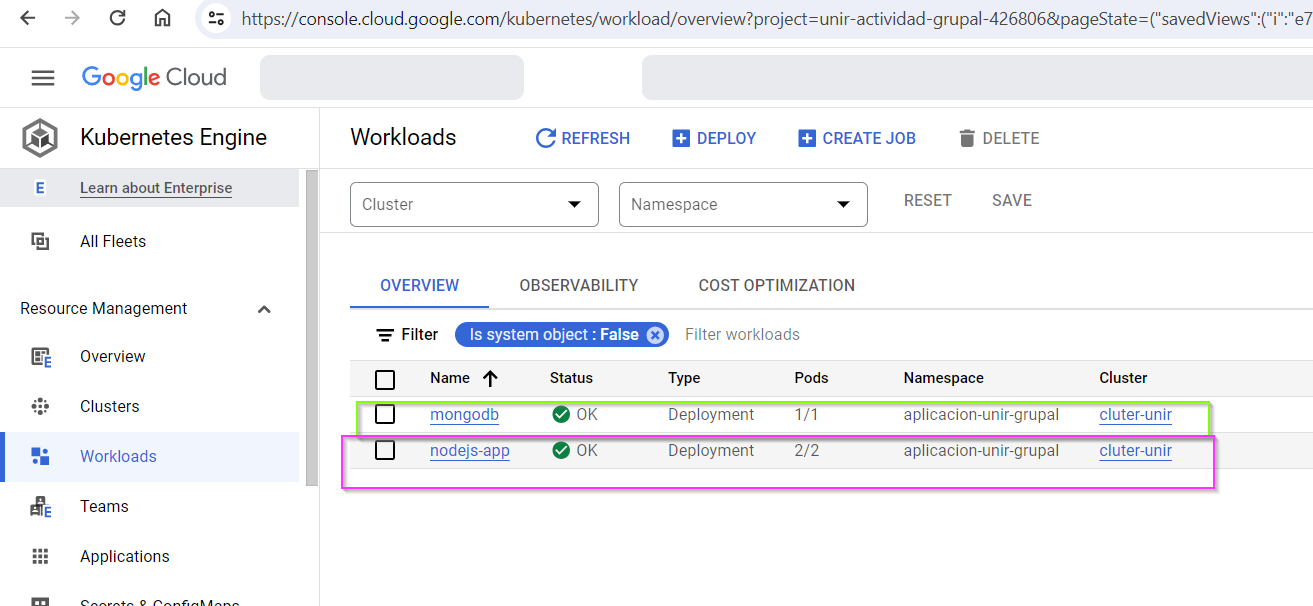


Ilustración 23 Despliegue a la nube de Google Cloud, usando el servicio de Kubernetes

Validamos el despliegue por medio del sdk de Google Cloud. Por medio del comando get pods, obtenemos los pods, y por medio de get svc obtenemos los servicios. Finalmente realizamos una petición por medio curl hacia la IP externa de nuestro balanceador de carga.

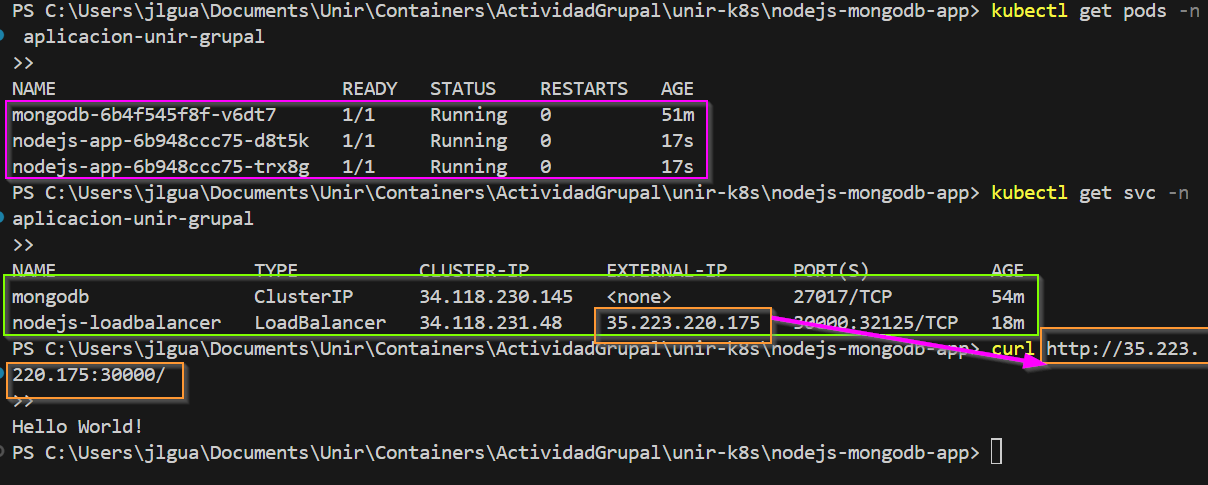


Ilustración 24 Validación del despliegue por medio del sdk de Google Cloud

Para probar el funcionamiento del balanceador de carga eliminamos uno de los pods por medio del comando kubectl delete, y verificamos cómo se crea de manera automática para tener un mínimo de dos réplicas.

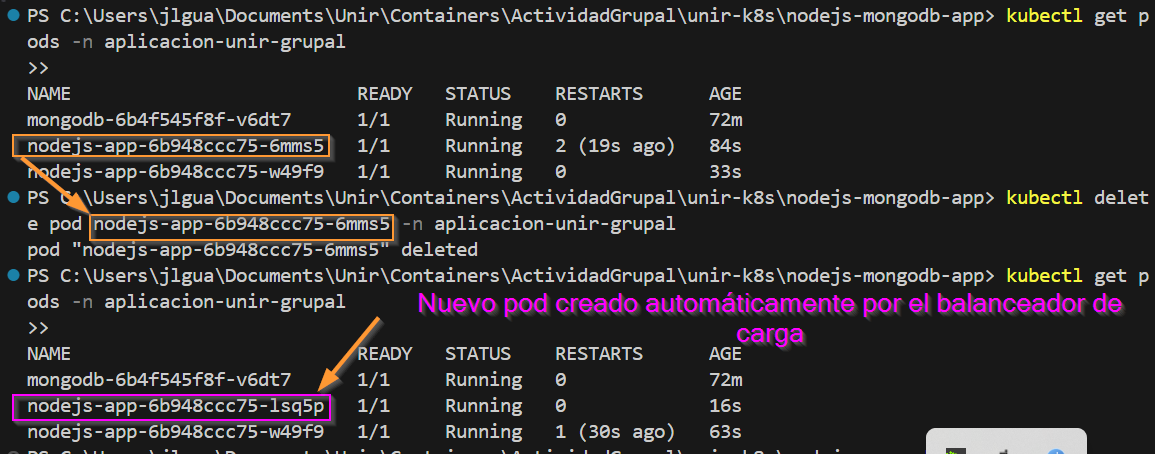


Ilustración 25 Prueba del funcionamiento del balanceador de carga

# **Conclusiones**

El desarrollo y despliegue de una aplicación Node.js junto con una base de datos MongoDB en un clúster de Kubernetes ha sido exitoso, cumpliendo con todos los objetivos y pautas establecidas en la actividad. A través de la creación de un fichero Dockerfile, se ha logrado contenerizar la aplicación, asegurando que se sigan las buenas prácticas de construcción de imágenes Docker. La imagen generada fue publicada en Docker Hub, demostrando la capacidad de gestionar y compartir contenedores de manera eficiente.

La configuración de los recursos de Kubernetes, incluyendo la definición de un namespace específico, la creación de un contexto asociado, y la correcta definición de Pods y Servicios, ha permitido un despliegue organizado y modular. El uso de un ReplicaSet para la aplicación Node.js y la implementación de un Servicio de tipo LoadBalancer han asegurado la alta disponibilidad y el balanceo de carga, garantizando que la aplicación sea accesible de manera continua y eficiente.

Adicionalmente, la implementación de volúmenes persistentes mediante PersistentVolume y PersistentVolumeClaim ha proporcionado una solución robusta para el almacenamiento de datos en MongoDB, permitiendo una gestión eficiente y segura de los datos persistentes. Este enfoque desacoplado para la gestión del almacenamiento refuerza la capacidad de escalar y administrar la base de datos en un entorno dinámico.

# **Tabla de valoración individual**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Sí | No | A veces |
| Todos los miembros se han integrado al trabajo del grupo | X |  |  |
| Todos los miembros participan activamente | X |  |  |
| Todos los miembros respetan otras ideas aportadas | X |  |  |
| Todos los miembros participan en la elaboración del informe | X |  |  |
| Me he preocupado por realizar un trabajo cooperativo con mis compañeros | X |  |  |
| Señala si consideras que algún aspecto del trabajo en grupo no ha sido adecuado | X |  |  |