

## 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

<b>Introducción .....</b>	<b>3</b>
<b>Aplicación para contenerización .....</b>	<b>3</b>
Estructura de aplicación .....	3
Contenido de archivos de aplicación .....	3
Ejecución de aplicación .....	5
<b>Creación y publicación de la imagen .....</b>	<b>5</b>
Generación de la imagen con el fichero Dockerfile .....	5
Publicación de la imagen en Docker Hub .....	6
<b>Definir namespace, contexto, pod y servicio para MongoDB.....</b>	<b>7</b>
La estructura del proyecto se modificó para organizar los archivos de mejor manera. ....	7
Definir namespace para los recursos .....	8
Crear contexto asociado al namespace .....	8
Definir correctamente el pod.....	8
Definir el servicio para MongoDB .....	9
<b>Definir ReplicaSet, LoadBalancer en pod de NodeJs. ....</b>	<b>9</b>
Definir ReplicaSet .....	9
Definir servicio de LoadBalancer para aplicación de Nodejs .....	10
Definir PersistentVolume y PersistentVolumeClaim .....	10
Actualizar definición de pod de MongoDB .....	13
<b>Conclusiones.....</b>	<b>14</b>
<b>Tabla de valoración individual .....</b>	<b>15</b>

## Tabla de Ilustración

Ilustración 1 Estructura de la aplicación.....	3
Ilustración 2 Contribuyentes en GitHub.....	3
Ilustración 3 Ejecución de la aplicación.....	5
Ilustración 4 Generación de la imagen con el fichero Dockerfile.....	6
Ilustración 5 Inicio de sesión y etiqueta en Docker Hub.....	6
Ilustración 6 Publicación de imagen .....	6
Ilustración 7 Imagen publicada en DockerHub .....	7
Ilustración 8 Definición de namespace, contexto, pod y servicio para MongoDB.....	7
Ilustración 9 Definición de namespace para los recursos .....	8
Ilustración 10 Definición correctamente el pod .....	8
Ilustración 11 Definición el servicio para MongoDB.....	9
Ilustración 12 Definición de ReplicaSet.....	9
Ilustración 13 Definición servicio de LoadBalancer para aplicación de Nodejs .....	10
Ilustración 14 Configuración del Persistentvolume (Pv) .....	10
Ilustración 15 Aplicación la configuración del PersistentVolume .....	10
Ilustración 16 Configuración del Persistentvolumeclaim (Pvc).....	11
Ilustración 17 Aplicación de la configuración del PersistentVolumeClaim .....	11
Ilustración 18 Verificación El Estado De Los Recursos .....	11
Ilustración 19 Creación un Pod para probar el PVC: .....	12
Ilustración 20 Aplicación configuración del Pod.....	12
Ilustración 21 Verificación que el Pod esté en estado Running. ....	12
Ilustración 22 Acceso al contenedor dentro del Pod para la creación de archivos en el volumen montado .....	13
Ilustración 23 Despliegue a la nube de Google Cloud, usando el servicio de Kubernetes ...	13
Ilustración 24 Validación del despliegue por medio del sdk de Google Cloud.....	13
Ilustración 25 Prueba del funcionamiento del balanceador de carga.....	14

## 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

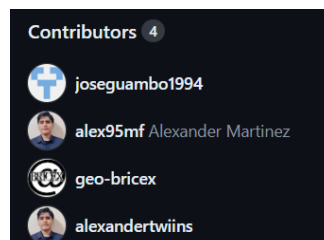
### Estructura de aplicación

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

```
unir-k8s/  
├── .gitignore  
├── README.md  
├── nodejs-mongodb-app/  
│   ├── kubernetes/  
│   │   ├── deployment.yaml  
│   │   ├── service.yaml  
│   │   └── mongodb-pv.yaml  
│   ├── node_modules/  
│   ├── db.js  
│   ├── Dockerfile  
│   ├── package-lock.json  
│   ├── package.json  
│   └── server.js
```

*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 .***

```
PS D:\Archivos Importantes\Académico Ronny Maestría\Segundo Cuatrimestre\Contenedores\Actividad grupal\unir-k8s\nodejs-mongodb-app> docker build -t nodejs-mongodb-app:latest .
[+] Building 1.5s (12/12) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 797B
=> [internal] load metadata for docker.io/library/node:14-alpine
=> [internal] load .dockerignore
=> => transferring context: 28
=> [builder 1/5] FROM docker.io/library/node:14-alpine@sha256:434215b487a329c9e867282ff89e704d3a75e554822e07f3e0c0f9e606121b33
=> [internal] load build context
=> => transferring context: 90.21kB
=> CACHED [builder 2/5] WORKDIR /usr/src/app
=> CACHED [builder 3/5] COPY package*.json ./
=> CACHED [builder 4/5] RUN npm install --only=production
=> CACHED [builder 5/5] COPY . .
=> CACHED [stage-1 3/4] COPY --from=builder /usr/src/app /usr/src/app
=> CACHED [stage-1 4/4] RUN addgroup -S appgroup && adduser -S appuser -G appgroup
=> exporting to image
=> => exporting layers
=> => writing image sha256:29f08861face81b56f72a1bf9870f915631f4502996252da14dc20e2af3ea894
=> => naming to docker.io/library/nodejs-mongodb-app:latest

View build details: docker-desktop://dashboard/build/default/default/0698kqr5a2jn2xevtkvgf1h4s

What's Next?
View a summary of image vulnerabilities and recommendations -> docker scout quickview
```

*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:

```
PS D:\Archivos Importantes\Académico Ronny Maestría\Segundo Cuatrimestre\Contenedores\Actividad grupal\unir-k8s\nodejs-mongodb-app> docker login
Authenticating with existing credentials...
Login Succeeded
PS D:\Archivos Importantes\Académico Ronny Maestría\Segundo Cuatrimestre\Contenedores\Actividad grupal\unir-k8s\nodejs-mongodb-app> docker tag nodejs-mongodb-app:latest alex95mf/nodejs-mongodb-app:latest
PS D:\Archivos Importantes\Académico Ronny Maestría\Segundo Cuatrimestre\Contenedores\Actividad grupal\unir-k8s\nodejs-mongodb-app>
```

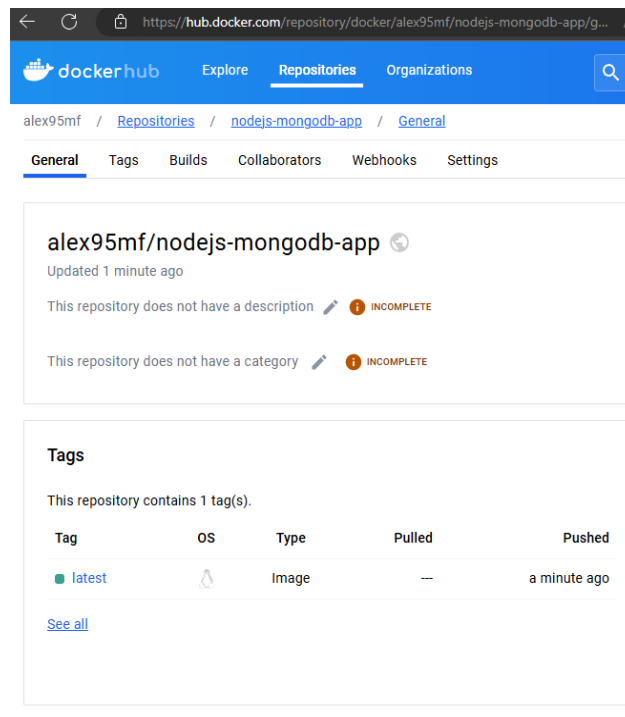
*Ilustración 5 Inicio de sesión y etiqueta en Docker Hub*

### 2. Publicar la imagen:

```
PS D:\Archivos Importantes\Académico Ronny Maestría\Segundo Cuatrimestre\Contenedores\Actividad grupal\unir-k8s\nodejs-mongodb-app> docker push alex95mf/nodejs-mongodb-app:latest
The push refers to repository [docker.io/alex95mf/nodejs-mongodb-app]
70fd2ee92c1f: Pushed
f66920a0883c: Pushed
3642b13174dd: Layer already exists
31f710dc178f: Layer already exists
a599bf3e59b8: Layer already exists
e67e8085abae: Layer already exists
f1417ff83b31: Layer already exists
latest: digest: sha256:0af5d5f09bd4caf1fa10a55455ee37c7cdda5e2ba66ba21a25425dc8b1bf73bf size: 1784
```

*Ilustración 6 Publicación de imagen*

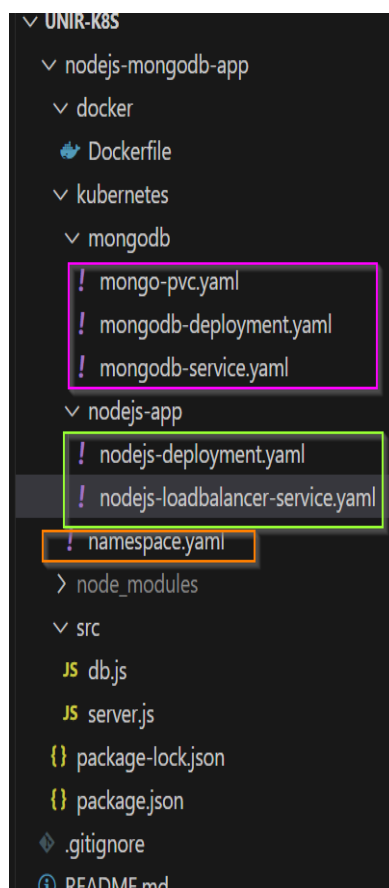
Imagen publicada satisfactoriamente en DockerHub:



*Ilustración 7 Imagen publicada en DockerHub*

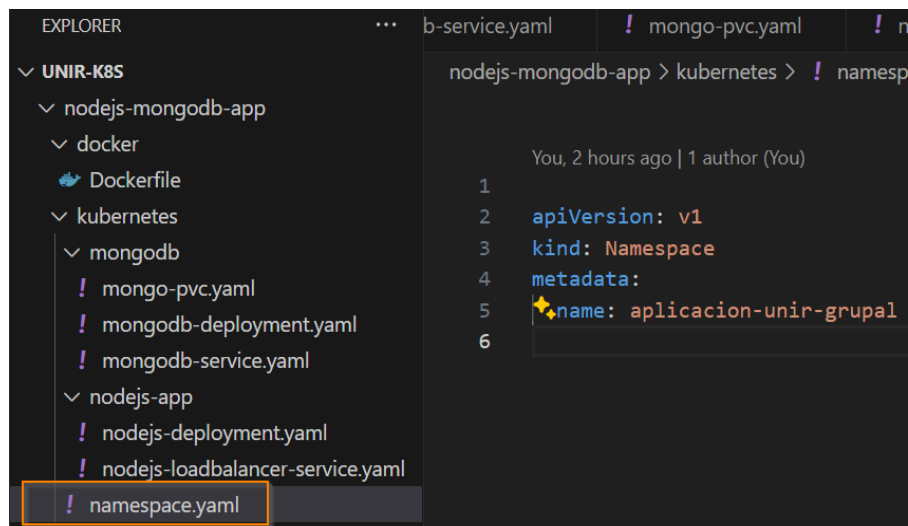
## Definir namespace, contexto, pod y servicio para MongoDB

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

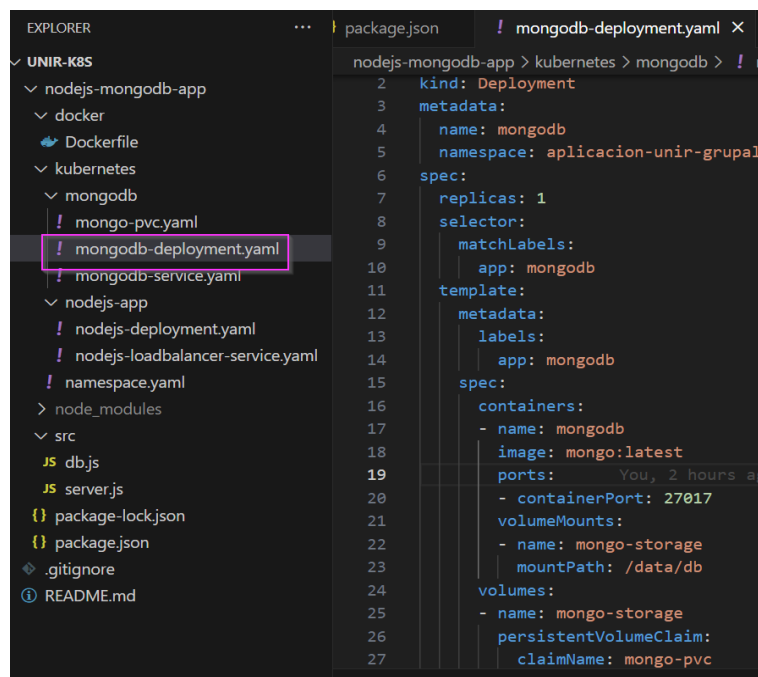


*Ilustración 9 Definición de namespace para los recursos*

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



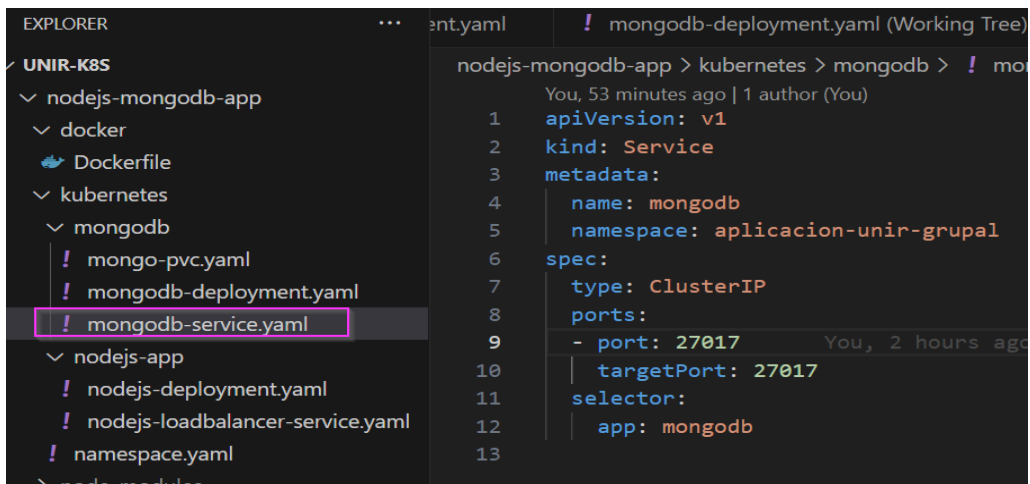


Ilustración 11 Definición el servicio para MongoDB

## 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.

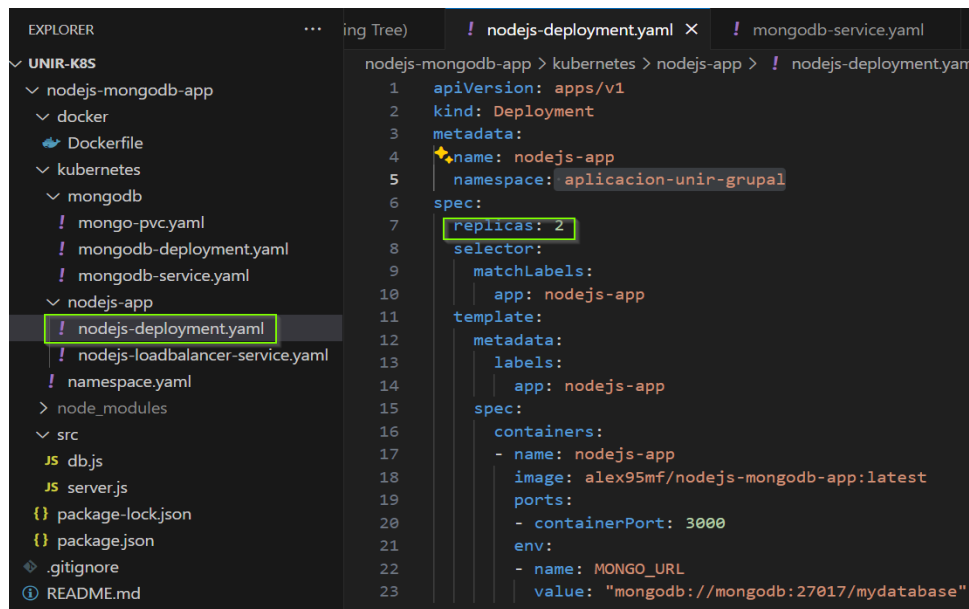


Ilustración 12 Definición de ReplicaSet

### Definir servicio de LoadBalancer para aplicación de Nodejs

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

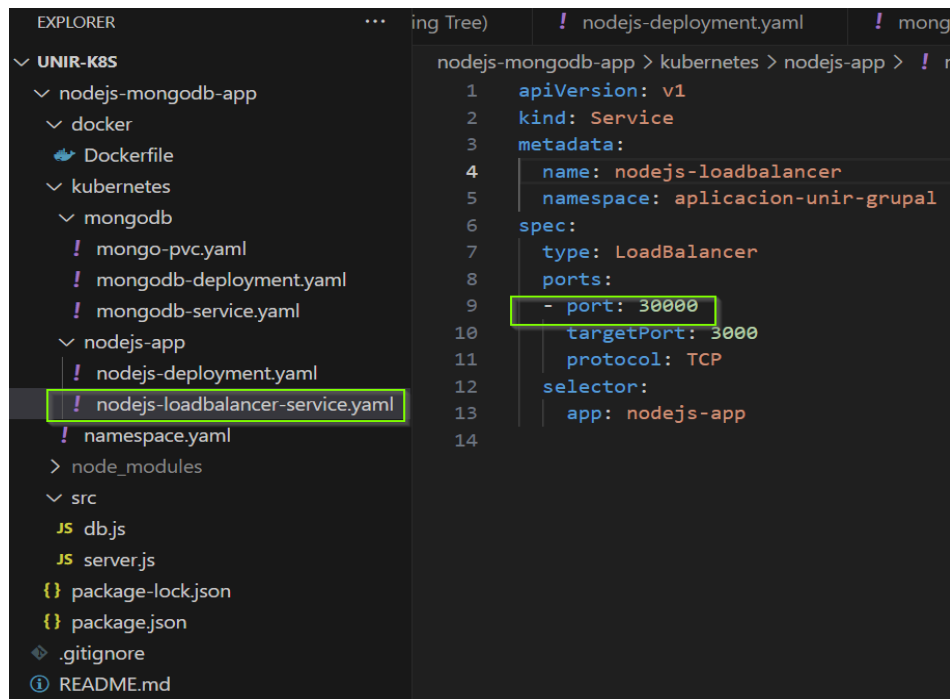


Ilustración 13 Definición servicio de LoadBalancer para aplicación de Nodejs

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.

```

1  apiVersion: v1
2  kind: PersistentVolume
3  metadata:
4    name: mongo-pv
5  spec:
6    capacity:
7      storage: 1Gi # Capacidad del volumen persistente
8    volumeMode: Filesystem
9    accessModes:
10     - ReadWriteOnce # Modo de acceso al volumen
11    hostPath:
12      path: C:\Alexis\UNIR\Proyectos\Contenerizacion\nodejs-mongodb-app\kubernetes\mongodb\StoragePV # Ruta en el host donde se almacenarán los datos

```

Ilustración 14 Configuración del Persistentvolume (Pv)

Aplicar la configuración del PersistentVolume

```

C:\Alexis\UNIR\Proyectos\Contenerizacion\nodejs-mongodb-app\kubernetes\mongodb>kubectl apply -f mongo-pv.yaml
persistentvolume/mongo-pv created

C:\Alexis\UNIR\Proyectos\Contenerizacion\nodejs-mongodb-app\kubernetes\mongodb>

```

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.

```

1  apiVersion: v1
2  kind: PersistentVolumeClaim
3  metadata:
4    name: mongo-pvc
5  spec:
6    accessModes:
7      - ReadWriteOnce # Modo de acceso al volumen (debe coincidir con el PV)
8    resources:
9      requests:
10     storage: 1Gi # Cantidad de almacenamiento solicitado

```

*Ilustración 16 Configuración del PersistentVolumeClaim (Pvc)*

Aplicar la configuración del PersistentVolumeClaim

```

C:\Alexis\UNIR\Proyectos\Contenerizacion\nodejs-mongodb-app\kubernetes\mongodb>kubectl apply -f mongo-pvc.yaml
persistentvolumeclaim/mongo-pvc created
C:\Alexis\UNIR\Proyectos\Contenerizacion\nodejs-mongodb-app\kubernetes\mongodb>

```

*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.

```

C:\Alexis\UNIR\Proyectos\Contenerizacion\nodejs-mongodb-app\kubernetes\mongodb>kubectl get pv
NAME             CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM             STORAGECLASS  VOLUMEATTRIBUTESCLASS  REASON  AGE
mongo-pv         1Gi       RWO           Retain          Available  default/mongo-pvc  hostpath           <unset>  8m22s
pvc-e99419e0-34ec-4650-9a47-49de9b69f6a0  1Gi       RWO           Delete          Bound     default/mongo-pvc  hostpath           <unset>  99s

C:\Alexis\UNIR\Proyectos\Contenerizacion\nodejs-mongodb-app\kubernetes\mongodb>kubectl get pvc
NAME             STATUS  VOLUME             CAPACITY  ACCESS MODES  STORAGECLASS  VOLUMEATTRIBUTESCLASS  AGE
mongo-pvc        Bound   pvc-e99419e0-34ec-4650-9a47-49de9b69f6a0  1Gi       RWO           hostpath       <unset>             107s
C:\Alexis\UNIR\Proyectos\Contenerizacion\nodejs-mongodb-app\kubernetes\mongodb>

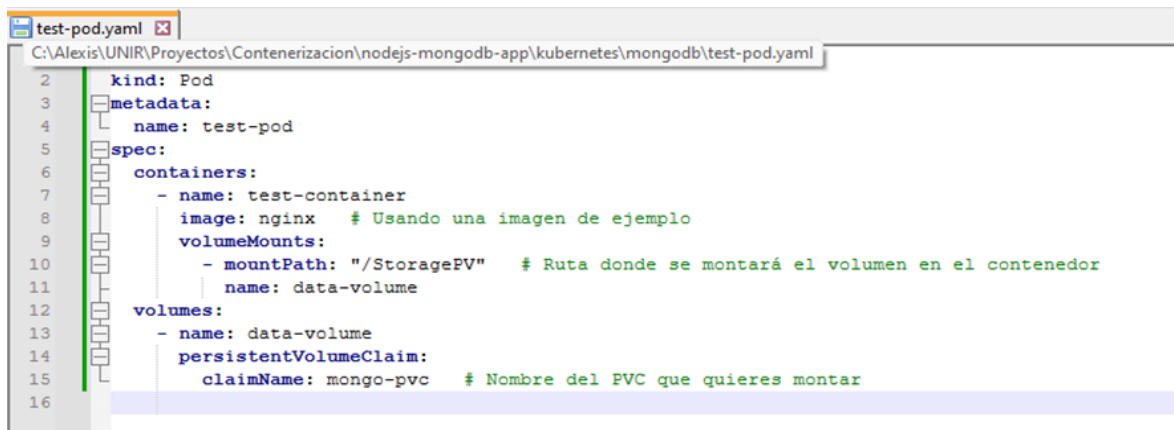
```

*Ilustración 18 Verificación El Estado De Los Recursos*

## Pruebas

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

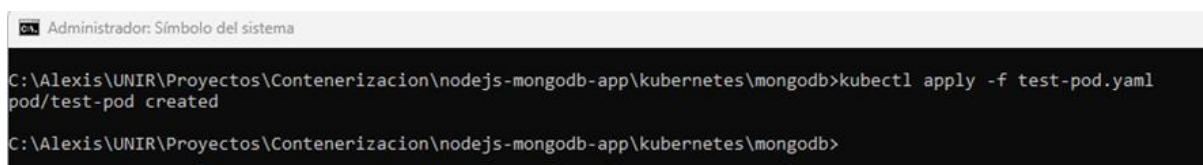
Creación de archivo test-pod.yaml. Este Pod montará el volumen del PVC **mongo-pvc** en **/StoragePV** dentro del contenedor.



```
2 kind: Pod
3 metadata:
4   name: test-pod
5 spec:
6   containers:
7     - name: test-container
8       image: nginx # Usando una imagen de ejemplo
9       volumeMounts:
10        - mountPath: "/StoragePV" # Ruta donde se montará el volumen en el contenedor
11          name: data-volume
12   volumes:
13     - name: data-volume
14       persistentVolumeClaim:
15         claimName: mongo-pvc # Nombre del PVC que quieres montar
16
```

*Ilustración 19 Creación un Pod para probar el PVC:*

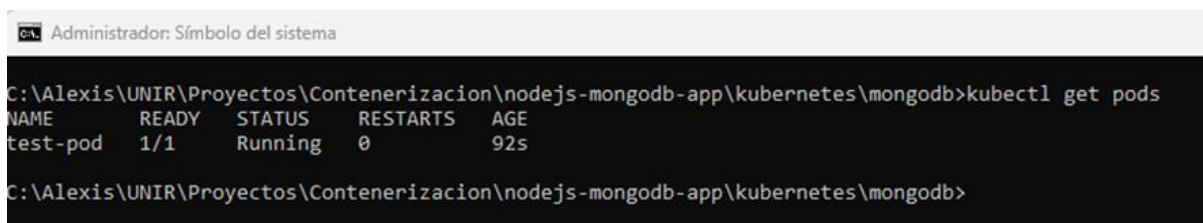
Aplicar configuración del Pod.



```
C:\Alexis\UNIR\Proyectos\Contenerizacion\nodejs-mongodb-app\kubernetes\mongodb>kubectl apply -f test-pod.yaml
pod/test-pod created
C:\Alexis\UNIR\Proyectos\Contenerizacion\nodejs-mongodb-app\kubernetes\mongodb>
```

*Ilustración 20 Aplicación configuración del Pod.*

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



```
C:\Alexis\UNIR\Proyectos\Contenerizacion\nodejs-mongodb-app\kubernetes\mongodb>kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
test-pod  1/1     Running   0           92s
C:\Alexis\UNIR\Proyectos\Contenerizacion\nodejs-mongodb-app\kubernetes\mongodb>
```

*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.

```

C:\Alexis\UNIR\Proyectos\Contenerizacion\nodejs-mongodb-app\kubernetes\mongodb>kubectl exec -it test-pod -- /bin/bash
root@test-pod:/# cd StoragePV/
root@test-pod:/StoragePV# pwd
/StoragePV
root@test-pod:/StoragePV# touch test.txt
root@test-pod:/StoragePV# touch ArchivoPrueba.txt
root@test-pod:/StoragePV# ls -l
total 0
-rw-r--r-- 1 root root 0 Jun 20 05:58 ArchivoPrueba.txt
-rw-r--r-- 1 root root 0 Jun 20 05:58 test.txt
root@test-pod:/StoragePV# exit
exit
C:\Alexis\UNIR\Proyectos\Contenerizacion\nodejs-mongodb-app\kubernetes\mongodb>

```

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.

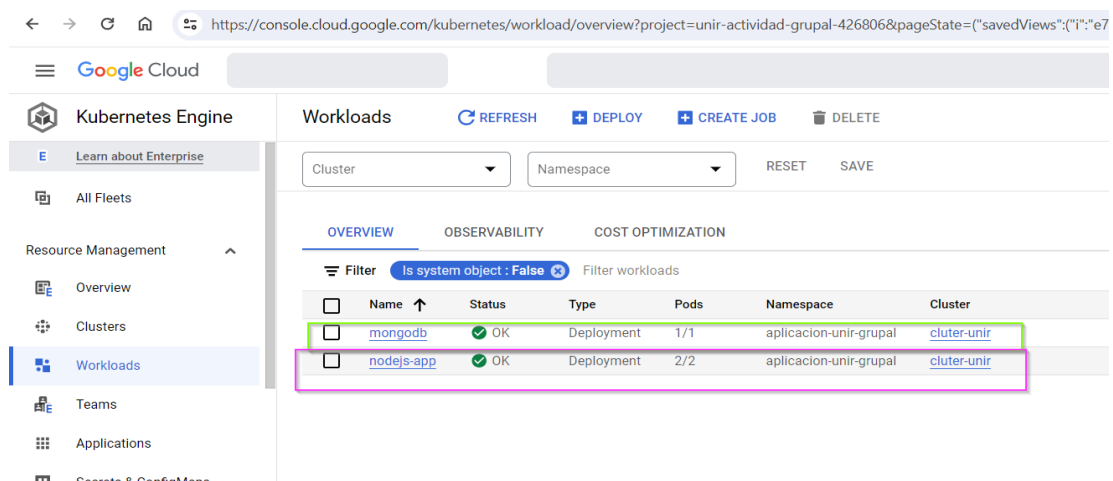


Ilustración 23 Despliegue a la nube de Google Cloud, usando el servicio de Kubernetes

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.

```

PS C:\Users\jlgua\Documents\Unir\Containers\ActividadGrupal\unir-k8s\nodejs-mongodb-app> kubectl get pods -n aplicacion-unir-grupal
>>
NAME                                READY   STATUS    RESTARTS   AGE
mongodb-6b4f545f8f-v6dt7            1/1     Running   0           51m
nodejs-app-6b948ccc75-d8t5k         1/1     Running   0           17s
nodejs-app-6b948ccc75-trx8g         1/1     Running   0           17s
PS C:\Users\jlgua\Documents\Unir\Containers\ActividadGrupal\unir-k8s\nodejs-mongodb-app> kubectl get svc -n aplicacion-unir-grupal
>>
NAME                                TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
mongodb                            ClusterIP      34.118.230.145  <none>           27017/TCP        54m
nodejs-loadbalancer                 LoadBalancer  34.118.231.48   35.223.220.175  30000:32125/TCP  18m
PS C:\Users\jlgua\Documents\Unir\Containers\ActividadGrupal\unir-k8s\nodejs-mongodb-app> curl http://35.223.220.175:30000/
>>
Hello World!
PS C:\Users\jlgua\Documents\Unir\Containers\ActividadGrupal\unir-k8s\nodejs-mongodb-app>

```

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.

```
PS C:\Users\jlgua\Documents\Unir\Containers\ActividadGrupal\unir-k8s\nodejs-mongodb-app> kubectl get pods -n aplicacion-unir-grupal
>>
NAME                                READY   STATUS    RESTARTS   AGE
mongodb-6b4f545f8f-v6dt7            1/1     Running   0           72m
nodejs-app-6b948ccc75-6mms5         1/1     Running   2 (19s ago) 84s
nodejs-app-6b948ccc75-w49f9         1/1     Running   0           33s
PS C:\Users\jlgua\Documents\Unir\Containers\ActividadGrupal\unir-k8s\nodejs-mongodb-app> kubectl delete pod nodejs-app-6b948ccc75-6mms5 -n aplicacion-unir-grupal
pod "nodejs-app-6b948ccc75-6mms5" deleted
PS C:\Users\jlgua\Documents\Unir\Containers\ActividadGrupal\unir-k8s\nodejs-mongodb-app> kubectl get pods -n aplicacion-unir-grupal
>>
NAME                                READY   STATUS    RESTARTS   AGE
mongodb-6b4f545f8f-v6dt7            1/1     Running   0           72m
nodejs-app-6b948ccc75-lsq5p         1/1     Running   0           16s
nodejs-app-6b948ccc75-w49f9         1/1     Running   1 (30s ago) 63s
```

Nuevo pod creado automáticamente por el balanceador de carga

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		