**UNIVERSIDAD INTERNACIONAL DE LA RIOJA (UNIR)**  
*La Universidad en Internet*

*Contenedores*

*(MEXDEVOPS)*

**Maestría en Desarrollo y Operaciones de Software**  
**2025**

Actividad 3 grupal: Construcción de aplicativo en contenedores y despliegue en un clúster de Kubernetes

Equipo 2\_H

**Índice**

[**Generar las imágenes de los aplicativos con Dockerfile** 3](#_Toc201254458)

[**Estructura basada en contenedores: separación de responsabilidades** 3](#_Toc201254459)

[**Dockerfile App** 3](#_Toc201254460)

[**Dockerfile DB** 3](#_Toc201254461)

[**Configuración con docker-compose.yml para hacer pruebas** 3](#_Toc201254462)

[**Pruebas locales** 4](#_Toc201254463)

[**Publicación en Docker Hub** 5](#_Toc201254464)

[**Kubernetes** 5](#_Toc201254465)

[**Resultado:** 8](#_Toc201254466)

[**Limpiar espacio local** 8](#_Toc201254467)

[**EKS (Amazon Elastic Kubernetes Service)** 8](#_Toc201254468)

[**Permisos AWS CLI** 9](#_Toc201254469)

[**Creación del clúster** 9](#_Toc201254470)

[**Comprobación de creación y contexto del clúster** 10](#_Toc201254471)

[**Aplicación de los manifiestos dentro de nuestro clúster** 11](#_Toc201254472)

[**Resultados finales** 12](#_Toc201254473)

[**Conclusión** 13](#_Toc201254474)

[**Referencias** 13](#_Toc201254475)

# **Generar las imágenes de los aplicativos con Dockerfile**

## **Estructura basada en contenedores: separación de responsabilidades**

Como ya se ha mencionado, para esta actividad se ha diseñado una aplicación compuesta por dos contenedores:

* Contendor App: expone una API Rest construida con Node.js y Express. Consulta una base de datos para recuperar diferente información entre ella productos.
* Contenedor DB: instancia PostgreSQL con datos precargados mediante un script SQL (init.sql)

Esta arquitectura refleja el principio de separación de responsabilidades, facilita el despliegue, la escalabilidad independiente de los servicios y una migración sencilla a un clúster de Kubernetes.

### **Dockerfile App**

Para este Dockerfile se usa una imagen base minimalista la cual reduce el tamaño final, un usuario sin privilegios para mejorar la seguridad para evitar la ejecución como modo root, separación de dependencias, exposición explícita del puerto 3000, uso de npm ci haciendo una instalación determinista basada en package-lock.json ideal para producción.

### **Dockerfile DB**

Este Dockerfile extiende la imagen oficial de PostgreSQL y añade un script SQL para inicializar una tabla de productos con datos de prueba. Esto permite a que la API tenga datos desde el inicio y sea más realista su integración en Kubernetes.

### **Configuración con docker-compose.yml para hacer pruebas**

Se crea un archivo compose para hacer la prueba de que todo este funcionando correctamente, las principales decisiones para esta configuración son:

Separación en servicios: se definen servicios independientes (app y db).

Red compartida personalizada (actividad3-net): permite que los contenedores se comuniquen usando el nombre del servicio como host (DB\_HOST = db)

Variables de retorno explícitas: se utilizan para definir credenciales, puertos y nombres de base de datos, facilitando su futura gestión con Secrets en Kubernetes.

Montaje de un volumen persistente (db-data): asegura que los datos de PostgreSQL no se pierdan al reiniciar el contenedor, lo que es un paso intermedio antes de definir PersistentVolumeClaims en Kubernetes.

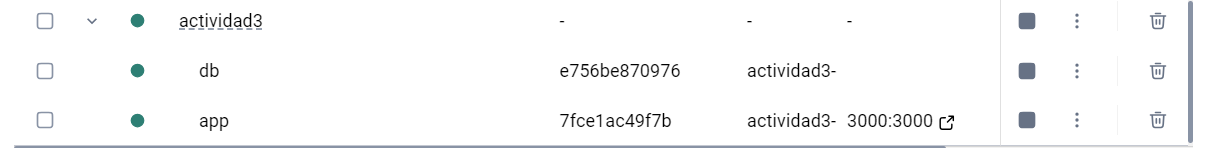
Mapeo de puertos (3000:3000): permite exponer el contenedor de la aplicación localmente a través del puerto que nosotros elijamos, facilitando pruebas manuales.

### **Pruebas locales**

Hacemos un docker compose up –build para poder iniciar nuestro proyecto

Texto

El contenido generado por IA puede ser incorrecto.

Podemos comprobar en la consola que todo esté en orden y también en nuestra interfaz de Docker Desktop:  


Por último de manera local podemos comprobar que la aplicación web funciona correctamente y también puede conectarse al contenedor de la base de datos:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Para detener nuestros contenedores podemos hacer uso del comando docker compose down.

# **Publicación en Docker Hub**

Una vez validada las imágenes localmente se procede a la publicación en Docker Hub. Esto es necesario para garantizar que los manifiestos de Kubernetes puedan acceder a imágenes públicas sin necesidad de reconstruirlas en el entorno distinto.

Los paso realizados fueron:

1. Se etiquetan las imágenes locales con el nombre del repositorio en Docker Hub utilizando: docker tag actividad3-app:latest lemendezc/actividad3-app:1.0.0

docker tag actividad3-db:latest lemendezc/actividad3-db:1.0.0

1. Nos autenticamos con docker login.
2. Se realiza el envío (push) de ambas imágenes (actividad3-app y actividad3-db) con una versión explícita 1.0.0.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

# **Kubernetes**

Para trabajar con Kubernetes antes de hacerlo directamente en una nube como AWS haremos y configuraremos todo para que funcione localmente.

* El primer archivo es 01-namespace.yaml, el cual define un recurso de tipo Namespace dentro del clúster de Kubernetes.
* El archivo 02-secret.yaml define un recurso de tipo Secret, cuya finalidad es almacenar de forma segura las credenciales de acceso a la base de datos PostgreSQL, gestiona la contraseña de la base de datos de manera separada y segura.
* El archivo 03-pvc-db.yaml define un recurso de tipo PersistentVolumeClaim (PVC), que solicita a Kubernetes un volumen persistente de 1 GiB para almacenar los datos de la base de datos PostgreSQL
* 04-deployment-db.yaml este archivo define un Deployment de Kubernetes para desplegar la base de datos PostgreSQL como un pod controlado y administrado automáticamente.

La configuración del contenedor incluye:

* + envFrom.secretRef: importa las variables de entorno necesarias (POSTGRES\_DB, POSTGRES\_USER, POSTGRES\_PASSWORD) desde el Secret previamente creado (db-secret), evitando así escribir valores sensibles directamente en el manifiesto.
  + ports.containerPort: 5432: expone el puerto estándar de PostgreSQL dentro del contenedor.
  + volumeMounts + volumes: monta el PVC (db-pvc) en la ruta /var/lib/postgresql/data, que es el directorio donde PostgreSQL guarda sus datos. Esto garantiza persistencia aunque el contenedor sea destruido y recreado.
* 05-deployment-app.yaml este manifiesto define el Deployment de la aplicación web, es decir, el componente que expone la API REST desarrollada en Node.js.
* 06-service-db.yaml este archivo define un recurso de tipo Service que permite exponer el pod de PostgreSQL a otros recursos dentro del clúster de Kubernetes. El servicio se llama postgres y está asociado al namespace actividad3.
* 7-service-app.ymla este manifiesto define un recurso de tipo Service para exponer la aplicación web al exterior del clúster. A diferencia del servicio de base de datos, este Service está pensado para ser accedido por usuarios desde fuera de Kubernetes. permite acceder desde fuera del clúster
* 08-pv-db.yaml necesario para configurar el PersistentVolume especificando el tamaño, id del volumen y la región.

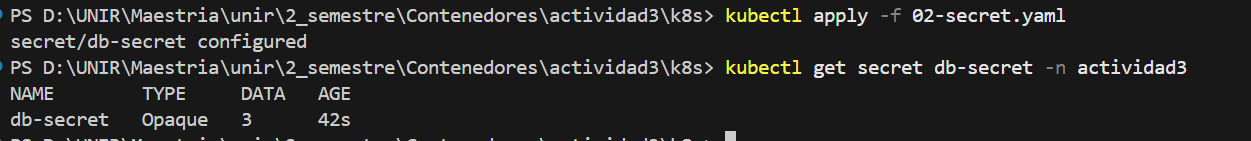
Localmente: Aplicar los manifiestos en orden

Namespace:

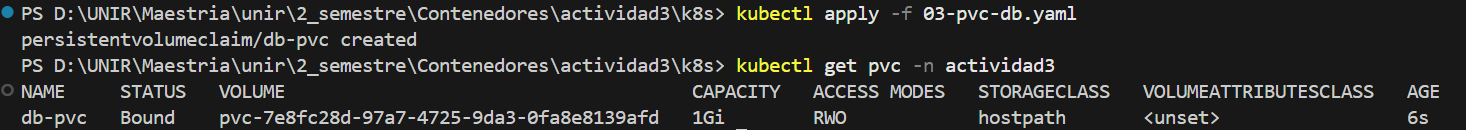
Texto

El contenido generado por IA puede ser incorrecto.

Secret:



PersistentVolumeClaim:

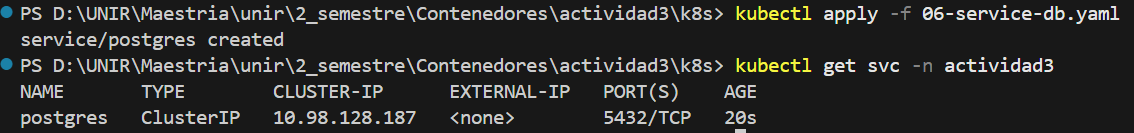


Deployment DB:

Texto

El contenido generado por IA puede ser incorrecto.

Service DB:



Deployment App:

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Service App:

Texto

El contenido generado por IA puede ser incorrecto.

### **Resultado:**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

### **Limpiar espacio local**

Ya que todo fue creado dentro del namespace actividad3 es sencillo eliminar todo lo creado con el siguiente comando:

kubectl delete namespace actividad3



# **EKS (Amazon Elastic Kubernetes Service)**

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Instalación de eksctl para poder crear el cluster de kubernetes desde la consola de nuestra máquina.

choco install eksctl -y

Texto

El contenido generado por IA puede ser incorrecto.

## **Permisos AWS CLI**

Hay que tener en cuenta que es posible que el usuario que tenemos logeado en nuestra consola aws cli en nuestra computadora puede no tener los permisos necesarios para ello hay que asignar los siguientes permisos:  
Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

En nuestro caso fue necesario agregar un política un poco más laxa para el laboratorio y fue creado con json.

## **Creación del clúster**

Ahora podemos crear el clúster con un comando en PowerShell:  
eksctl create cluster --name actividad3-cluster --region us-west-1 --version 1.29 --nodegroup-name actividad3-nodes --node-type t3.medium --nodes 2 --nodes-min 1 --nodes-max 3 --managed

Esto creará:

* una VPC con subredes y seguridad,
* un clúster EKS con control plane gestionado,
* un grupo de nodos EC2 administrado (2 instancias t3.medium),
* y actualizará el archivo ~/.kube/config para que kubectl ya pueda usarlo directamente.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

## **Comprobación de creación y contexto del clúster**

Primero comprobamos los nodos:

Texto

El contenido generado por IA puede ser incorrecto.

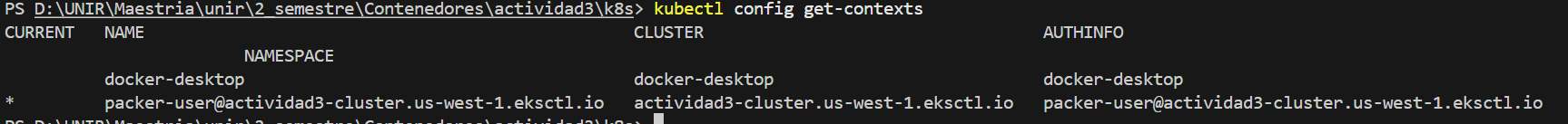
Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Nos tenemos que dar cuenta que estemos en el contexto adecuado:

kubectl config get-contexts  


## **Aplicación de los manifiestos dentro de nuestro clúster**

Dentro de nuestra carpeta k8s corremos el siguiente comando:  
kubectl apply -f .\

Nota: Es importante hacer cambio al archivo 07-service-app.yaml ya que haremos uso de un LoadBalancer.

Resultado:

Texto

El contenido generado por IA puede ser incorrecto.

Crear un volumen para después ser montado, al igual que agregar un addon dentro de nuestro clúster the ebs csi.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

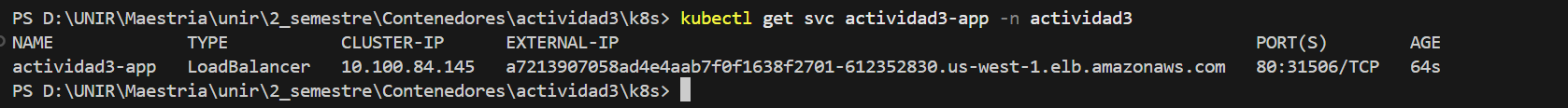
Texto

El contenido generado por IA puede ser incorrecto.

# **Resultados finales**

Obtenemos la dirección con el siguiente comando:

kubectl get svc actividad3-app -n actividad3



<http://abff2f8c78fac4fe4874d16f34649353-2026088694.us-west-1.elb.amazonaws.com/health>

Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.Eliminar el contenido y después el clúster completamente:

Texto

El contenido generado por IA puede ser incorrecto.

eksctl delete cluster --name actividad3-cluster --region us-west-1

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

# **Conclusión**

La implementación de una aplicación web con base de datos en un clúster de Kubernetes permitió comprender a fondo la gestión de volúmenes persistentes, servicios, despliegues y namespaces. Además, se fortalecieron habilidades clave para el manejo de errores, la depuración de contenedores y la eliminación segura de recursos en entornos productivos y de desarrollo.

# **Referencias**

* Hightower, K., Burns, B., & Beda, J. (2017). Kubernetes: Up and Running: Dive into the Future of Infrastructure. O'Reilly Media.
* The Kubernetes Authors. (2024). Kubernetes Documentation. Retrieved from https://kubernetes.io/docs/
* Amazon Web Services. (2024). Amazon EKS User Guide. Retrieved from https://docs.aws.amazon.com/eks/