Actividad adicional – Kubernetes/Docker/Despliegue Continuo

Daniel Esteban Castellanos Echeverria
José Andrés Jaramillo Caica

Contexto y explicación de nuestro tema a investigar para la actividad adicional

Profesor
Andrés Sánchez Martin
Arquitectura de Software

Pontificia Universidad Javeriana
Ingeniería de Sistemas
Bogotá
2025

1. Introducción

El presente trabajo tiene como propósito exponer y analizar el funcionamiento de Docker y Kubernetes, dos herramientas fundamentales dentro del entorno DevOps y de las arquitecturas modernas orientadas a despliegue continuo. Estas tecnologías han transformado la manera en que se construyen, empaquetan, distribuyen y administran aplicaciones, permitiendo una mayor escalabilidad, portabilidad y eficiencia en comparación con los modelos tradicionales basados en máquinas virtuales.

Docker brinda la capacidad de ejecutar aplicaciones dentro de contenedores ligeros y aislados, mientras que Kubernetes actúa como un orquestador capaz de gestionar dichos contenedores a gran escala, automatizando tareas como balanceo de carga, escalado, recuperación ante fallos y despliegues progresivos. Gracias a su integración, hoy es posible implementar software de forma más rápida, segura y confiable, alineándose con los principios de entrega continua y arquitectura distribuida.

Esta actividad busca mostrar no solo sus conceptos generales, sino también una implementación práctica que evidencia cómo estas herramientas pueden utilizarse conjuntamente para facilitar el despliegue de aplicaciones en entornos reales. De esta manera, se pretende demostrar su relevancia dentro del desarrollo actual y su impacto directo en el ciclo de vida del software.

2. ¿Por qué elegimos el tema?

Seleccionamos este tema porque durante el semestre tuvimos la oportunidad de trabajar directamente con herramientas como Docker y Kubernetes, lo que nos permitió adquirir experiencia práctica en despliegue continuo y en la gestión de arquitecturas basadas en contenedores. Fue un proceso retador, ya que implicó aprender conceptos nuevos, configurar entornos y enfrentar errores durante la implementación; sin embargo, ese mismo desafío fue lo que hizo el aprendizaje valioso y significativo.

Gracias a este trabajo logramos comprender mejor cómo funciona el modelo DevOps en la práctica, cómo se automatizan despliegues y por qué estas tecnologías son tan relevantes en la industria actual. Implementarlo nos permitió pasar de la teoría a la aplicación real, consolidando los conocimientos adquiridos en clase y fortaleciendo nuestras habilidades técnicas.

3. Contexto e Historia

Tecnología / Concepto	**Año de surgimiento**	**Creador / Organización**	**Problema que venía a resolver**	**Aporte principal a la industria**
Cultura DevOps	~2009	Patrick Debois (popularización)	Separación entre áreas de desarrollo y operaciones, entregas lentas, baja coordinación	Integra desarrollo y operaciones, fomenta automatización, entregas más rápidas y ciclos continuos de mejora
Despliegue Continuo	~2010	Surge como extensión natural de CI/CD	Liberaciones manuales costosas y propensas a fallos	Automatiza el paso del código probado hacia producción, reduciendo riesgos y tiempos de entrega

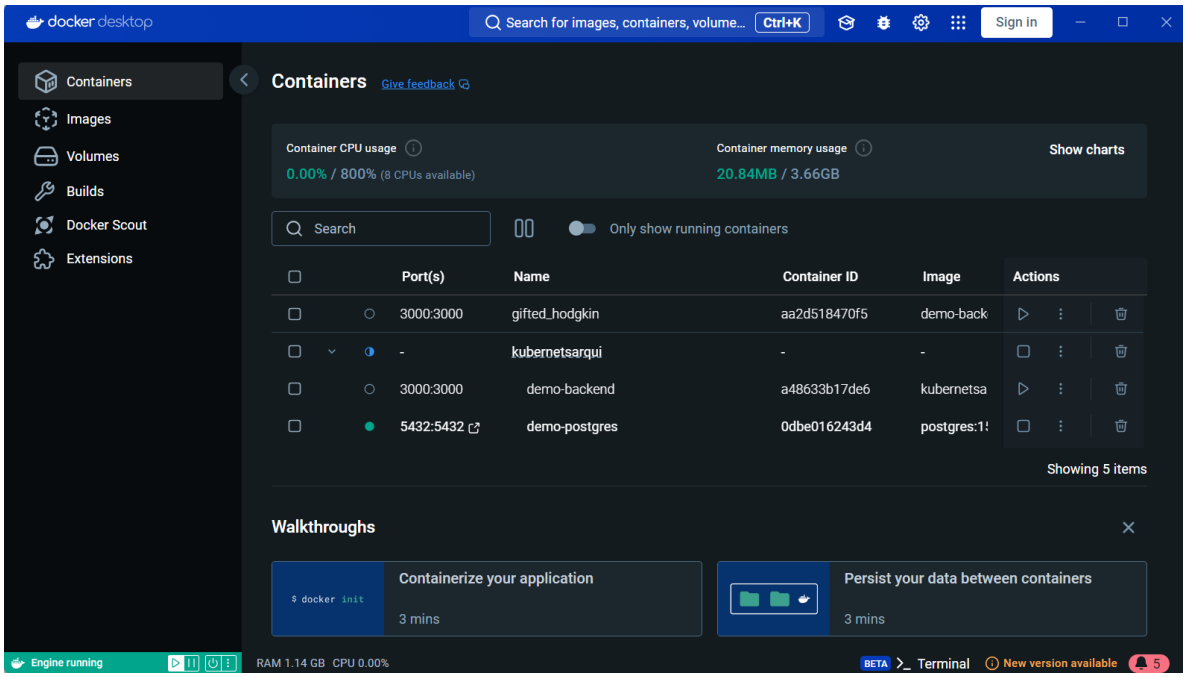
Docker	2013	Solomon Hykes – Docker Inc.	Complejidad en la instalación de entornos, incompatibilidades dependencias, VMs pesadas	Crea contenedores ligeros, portables y reproducibles con su propio entorno y dependencias
Kubernetes	2014 (open source en 2015)	Google → CNCF	Administrar miles de contenedores de forma automática, balanceo, escalabilidad y alta disponibilidad	Orquestación inteligente, escalado automático, self-healing, deployments progresivos y gestión de infraestructura a gran escala

4. ¿Cómo funciona Docker?

Docker funciona creando contenedores, que son como cajas donde se guarda una aplicación con todo lo que necesita para funcionar: su código, librerías, dependencias y configuración. Esto permite que la aplicación se ejecute igual en cualquier computador o servidor, sin importar el sistema operativo o versiones instaladas.

El proceso básico es este:

1. Se escribe un Dockerfile, que es un archivo donde se describen los pasos para construir la aplicación (qué instalar, qué ejecutar, etc.).
2. Con ese archivo se crea una imagen, que es como una plantilla o molde de la aplicación.
3. A partir de la imagen se inicia uno o varios contenedores, que son instancias reales en funcionamiento.
4. Los contenedores pueden ejecutarse en cualquier máquina que tenga Docker, lo que los hace portables, rápidos de iniciar y fáciles de distribuir.



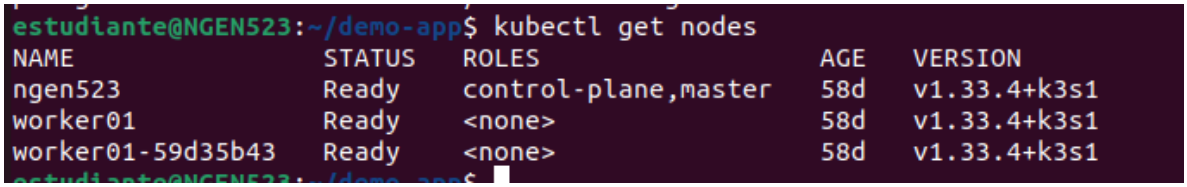
5. ¿Cómo funciona kubernetes?

Kubernetes funciona como el encargado de coordinar y controlar muchos contenedores al mismo tiempo.
Si Docker crea los contenedores, Kubernetes decide dónde se ejecutan, cuántos hay, los reinicia si fallan y los distribuye entre distintas máquinas.

En otras palabras, Kubernetes es como el jefe de los contenedores:

1. Recibe la aplicación en contenedores (generalmente empaquetados con Docker).
2. Decide en qué máquinas ejecutarla (según recursos disponibles).
3. Si un contenedor se cae, lo vuelve a levantar automáticamente.
4. Si la app necesita más capacidad, crea más contenedores; si hay menos uso, reduce la cantidad.

5. Permite distribuir carga, actualizar versiones sin detener el servicio y escalar aplicaciones fácilmente.

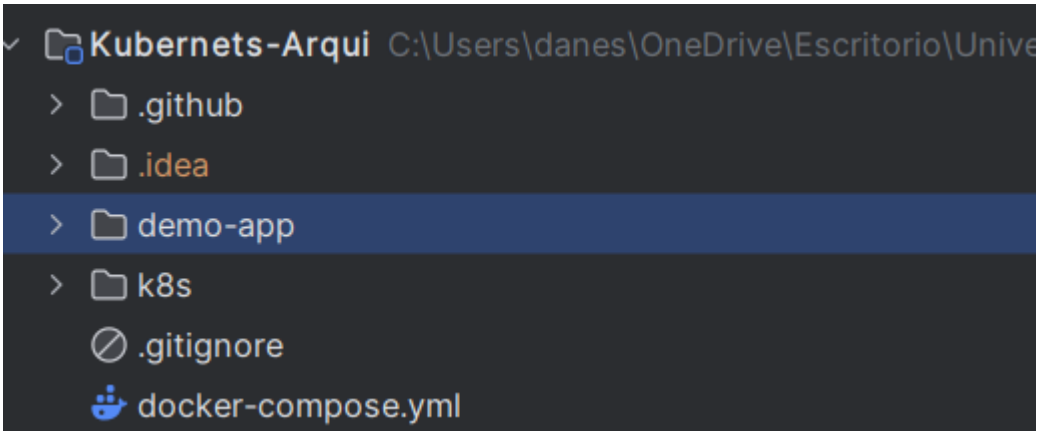


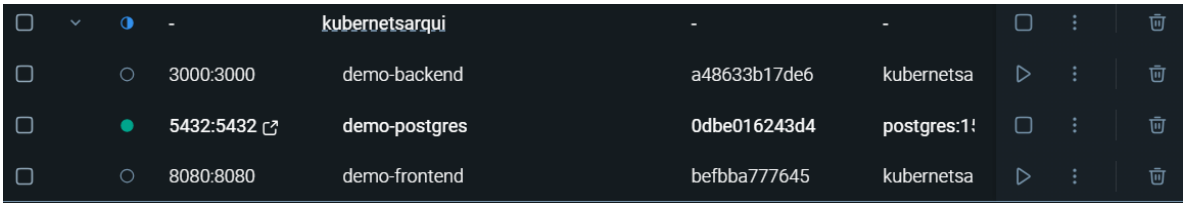
6. Ventajas y Desventajas

Tecnología	Ventajas	Desventajas
Docker	Ligero y rápido de desplegar. Portabilidad entre equipos/servidores.	Manejo complejo cuando hay muchos contenedores. Persistencia y redes requieren configuración extra.
Kubernetes	Escalabilidad automática. Alta disponibilidad y autorecuperación.	Curva de aprendizaje alta. Requiere más recursos e infraestructura.
Despliegue Continuo	Entregas rápidas y sin intervención manual. Reduce errores humanos en producción.	Necesita buena automatización y pipelines bien configurados. Fallos en el pipeline pueden detener el flujo.

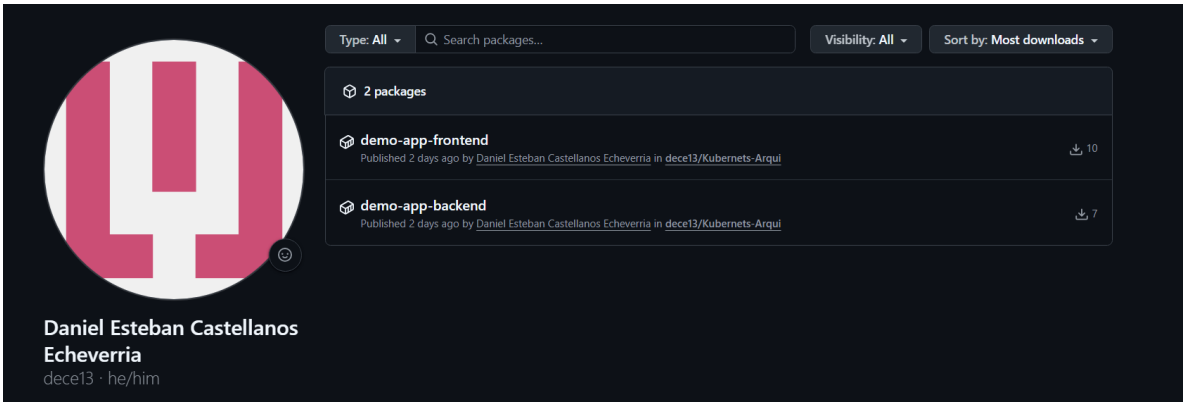
7. Ejemplo practico

Para la demostración construimos una aplicación pequeña compuesta por tres partes principales: front-end, back-end y base de datos. Una vez desarrollada la estructura del proyecto, procedimos a su contenedorización utilizando Docker. Para ello creamos un Dockerfile por cada servicio, definiendo dependencias, puertos y comandos de ejecución. Posteriormente utilizamos Docker Compose para unificar los tres contenedores y levantarlos como un solo entorno orquestado, permitiendo que la aplicación funcionara de forma conectada y estable entre sus componentes.





Con la arquitectura local ya funcionando con Docker, pasamos al proceso de integración continua. Subimos el proyecto a GitHub y configuramos el repositorio para que, cada vez que se realizara un push al branch main, se ejecutara automáticamente la construcción de nuevas imágenes de los servicios. Esto nos permitió mantener las versiones actualizadas sin intervención manual, como parte del pipeline CI/CD.



Finalmente conectamos el despliegue con nuestro entorno Kubernetes. Desde una máquina virtual configurada como control plane, creamos un namespace específico para el proyecto,

y utilizando los manifiestos (.yaml) necesarios para los deployments y services, vinculamos Kubernetes con las imágenes generadas en GitHub Packages.
Una vez aplicados los archivos con kubectl, el clúster levantó los pods correspondientes, gestionó su comunicación interna y completó la puesta en producción dentro del entorno.

```
estudiante@NGEN523:~/demo-app$ kubectl -n demo-app get pods -o wide
NAME                        READY   STATUS    RESTARTS        AGE   IP            NODE                NOMINATED NODE   READINESS GATES
backend-67676fd495-lbbbf    1/1     Running   3 (8m12s ago)   51m   10.42.2.102   worker01-59d35b43   <none>           <none>
backend-67676fd495-zh6rh    1/1     Running   3 (7m43s ago)   51m   10.42.1.30    worker01            <none>           <none>
frontend-6bcbf7d86d-54ct7   1/1     Running   0               51m   10.42.1.31    worker01            <none>           <none>
frontend-6bcbf7d86d-g86hp   1/1     Running   0               51m   10.42.2.103   worker01-59d35b43   <none>           <none>
postgres-88ff89899-85fxt    1/1     Running   0               51m   10.42.1.29    worker01            <none>           <none>
estudiante@NGEN523:~/demo-app$
```

8. Conclusión

El desarrollo y despliegue de nuestra aplicación utilizando Docker, Kubernetes y un flujo de integración/despliegue continuo nos permitió experimentar de manera real cómo funcionan las arquitecturas modernas dentro del modelo DevOps. Pasamos de tener un proyecto tradicional a convertirlo en un sistema escalable, portable y completamente automatizable, demostrando que estas tecnologías no solo son teoría, sino herramientas que transforman la forma de construir y entregar software.

Docker nos brindó un entorno reproducible para cada servicio, Kubernetes nos permitió orquestar los contenedores con eficiencia y GitHub automatizó la creación de imágenes y actualizaciones, eliminando procesos manuales y reduciendo riesgos. A pesar del reto técnico y la curva de aprendizaje, logramos implementar un pipeline funcional y desplegado en un cluster real, validando que es posible construir infraestructuras sólidas con herramientas open-source.

Este proyecto no solo representó una entrega final, sino un aprendizaje valioso que nos acerca a entornos profesionales y nos permite comprender cómo las grandes empresas gestionan sus aplicaciones a escala. Terminamos con más experiencia, mejor entendimiento y la satisfacción de haber superado el desafío.