Proyecto despliegue de aplicación con contenedores: Local y Nube.
Infraestructuras paralelas y distribuidas

Carlos Andres Delgado S, Msc
carlos.andres.delgado@correounivalle.edu.co

Mayo 2025

1. Descripción del Problema

En el desarrollo moderno de aplicaciones, la separación de componentes es fundamental para lograr escalabilidad, mantenimiento y mejor organización del código. Una aplicación comúnmente se divide en tres partes principales: Backend, Base de Datos y Frontend. La separación de estos componentes permite que cada uno realice tareas específicas y comunique de manera eficiente, manteniéndose lo más autónomos posible entre sí.

En este proyecto, la solución propuesta se centrará en la creación de una aplicación que se compone de estos tres componentes fundamentales, integrándolos bajo arquitecturas locales y en la nube. Para un enfoque eficiente y escalable, se emplearán principios de microservicios y tecnologías como Docker Compose y Kubernetes para la solución local, así como Azure para replicar la estructura en la nube con un balanceador de carga.

1.1. Enfoque de Separación entre Componentes y Microservicios

La separación entre componentes, o en algunos casos la adopción de un enfoque de microservicios, implica descomponer una aplicación monolítica en módulos más pequeños y autónomos. Cada módulo, o microservicio, es responsable de una funcionalidad particular de la aplicación. Esto permite:

- **Escalabilidad**: Componentes individuales pueden ser escalados de manera independiente según la demanda.
- **Flexibilidad**: Permite elegir el mejor lenguaje y marco de trabajo para cada componente.
- **Mantenibilidad**: El cambio o la corrección de errores en un componente no afecta directamente a otros.

En nuestro proyecto, la comunicación entre el frontend y el backend se llevará a cabo a través de una API Rest, realizando operaciones CRUD sobre la base de datos.

1.2. API REST y sus Métodos

REST (Representational State Transfer) es un estilo arquitectónico ampliamente utilizado para construir APIs que interactúan con servicios web. Una API REST es independiente de la plataforma o el lenguaje, y utiliza los métodos HTTP para realizar operaciones. Los métodos más comunes son:

- **GET**: Recupera información del servidor. Por ejemplo, para obtener una lista de todos los libros en una biblioteca.
- **POST**: Envía datos al servidor para crear un nuevo recurso. Por ejemplo, para añadir un nuevo libro a la biblioteca.
- **PUT**: Actualiza un recurso existente en el servidor. Por ejemplo, para modificar la información de un libro.
- **DELETE**: Elimina un recurso del servidor. Por ejemplo, para borrar un libro de la base de datos.

La API Rest se diseñará de tal modo que el frontend pueda interactuar con el backend, que a su vez, comunicará con la base de datos para realizar las operaciones CRUD necesarias.

2. Solución Local

La solución local para el proyecto utilizando Docker Compose o Kubernetes debe cumplir con una serie de criterios que garanticen la modularidad, escalabilidad y facilidad de mantenimiento de la aplicación. A continuación, se describen los aspectos principales que se deben contemplar al implementar esta solución.

2.1. Contenedorización con Docker Compose

Docker Compose es una herramienta que permite definir y ejecutar aplicaciones multicontenedor. Para esta aplicación, se deben crear los siguientes contenedores:

- **Backend:** Este contenedor alojará la lógica de negocio y las API REST. Será responsable de todas las operaciones de manipulación de datos y comunicación con la base de datos.
- **Base de Datos:** Este contenedor manejará el almacenamiento persistente de la aplicación. Se recomienda utilizar una base de datos ligera y conocida como PostgreSQL o MySQL.
- **Frontend:** Este contenedor o servicio servirá archivos estáticos que componen la interfaz de usuario de la aplicación. Podría utilizar tecnologías como React, Angular o Vue.js.

En el archivo `docker-compose.yml`, se debe definir la configuración de cada uno de estos servicios:

```yaml
version: '3'

services:
  backend:
    build: ./backend
    ports:
      - "5000:5000"
    depends_on:
      - db

  db:
    image: postgres
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: password
    volumes:
      - db-data:/var/lib/postgresql/data

  frontend:
    build: ./frontend
    ports:
      - "8080:80"
    depends_on:
      - backend

volumes:
  db-data:
```

2.2. Configuración con Kubernetes

Si se opta por usar Kubernetes, se debe crear un conjunto de archivos YAML para definir los despliegues (Deployments), Servicios (Services) y Configuraciones de persistencia (Persistent Volumes) necesarias. Esto garantiza el despliegue escalable y la disponibilidad continua de los servicios.

- **Deployments:** Configura y gestiona la implementación de los pods para Backend, Base de Datos y Frontend.
- **Services:** Expone cada uno de los deploys, asegurando que los componentes puedan comunicarse entre sí.
- **Persistent Volumes Claims (PVC):** Para mantener la data de la base de datos persistente entre reinicios.

El siguiente es un ejemplo de configuración en Kubernetes para el Backend:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: backend
        image: backend-image
        ports:
        - containerPort: 5000
---
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  type: ClusterIP
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
```

Implementar la aplicación de forma modular con Docker Compose o Kubernetes facilita la transición hacia un entorno de producción en la nube, asegurando que la aplicación cumple con los estándares modernos de despliegue de software.

3. Solución en la Nube

La solución en la nube consiste en replicar nuestra solución local para asegurar que nuestra aplicación pueda soportar una mayor cantidad de tráfico y que esté disponible en todo momento. La migración a la nube implica desplegar nuestras aplicaciones, que originalmente funcionan en contenedores, en una plataforma de nube como Azure. Esto requiere algunas configuraciones adicionales, como el uso de un balanceador de carga para distribuir el tráfico entrante de forma equilibrada entre las instancias de backend.

3.1. Componentes Desplegados

Para implementar nuestra solución en la nube debemos considerar los siguientes pasos y componentes:

- **Balanceador de Carga**: Se utiliza para dirigir automáticamente las solicitudes de los usuarios a una de las instancias disponibles de la aplicación backend, asegurando así una distribución equitativa de la carga de trabajo. Esto también ayuda a asegurar un alto tiempo de actividad, ya que si una instancia de backend falla, el balanceador de carga redirigirá el tráfico a las instancias restantes.
- **Instancias Backend Replicadas**: Deben existir al menos tres réplicas de la aplicación del backend ejecutándose de manera simultánea. Esto no solo distribuye la carga de trabajo, sino que también proporciona redundancia en caso de fallas.
- **Base de Datos en la Nube**: La base de datos debe estar alojada en un servicio de base de datos gestionado en la nube. Esto asegura que los datos se almacenen en un entorno seguro y escalable, y que se realicen copias de seguridad automáticas. Se puede tener un contenedor, usar un servicio externo o mediante una instancia de máquina virtual en la nube.
- **Frontend**: Aunque el frontend no se replica, debe estar configurado para comunicarse con el balanceador de carga y las instancias de backend en la nube.

4. Informe del Proyecto Final

4.1. Estructura del Informe

El informe del proyecto final para el curso de Infraestructuras con Docker, Kubernetes y Cloud debe contener las siguientes secciones:

1. **Introducción**:
   - Breve descripción del proyecto.
   - Objetivos del proyecto.
2. **Solución Local**:
   - Descripción detallada de la arquitectura local usando Docker Compose o Kubernetes.
   - Diseño y separación de componentes: backend, base de datos y frontend.
   - Configuración de cada contenedor y cómo estos interactúan.
   - Ejemplo de configuración con Docker Compose:

```
version: '3.7'
services:
  backend:
    build: ./backend
```

```
    ports:
      - "5000:5000"
    depends_on:
      - db
  db:
    image: postgres
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: pass
    ports:
      - "5432:5432"
  frontend:
    build: ./frontend
    ports:
      - "3000:3000"
```

3. **Solución en la Nube**:
   - Replicación de la arquitectura local en la nube eg. Azure.
   - Configuración del balanceador de carga para el backend replicado tres veces.
   - Uso de servicios en la nube, configuración de redes y seguridad.
   - Descripción del flujo para subir la aplicación a la nube.
4. **Anaĺisis de calidad de código**: Explicar cómo se realizó la integración continua y la evaluación de calidad del código usando herrramientas como SonarQube, Codacy, etc. Debe integrar su aplicación sin usar aplicaciones de Github si no a través de los actions de Github.
5. **Análisis y Conclusiones**:
   - Analizar el rendimiento de la aplicación en ambiente local vs en la nube.
   - Analizar la latencia, escalabilidad y disponibilidad del sistema.
   - Interpretación de posibles retos y cómo fueron abordados.
   - Reflexiones sobre el uso de tecnologías como Docker y Kubernetes.
   - Elementos clave que deben considerarse al analizar un proyecto de esta índole:
     - *Escalabilidad*: Capacidad del sistema para manejar incrementos en la carga.
     - *Fiabilidad*: Asegurarse que el sistema se mantenga funcional bajo distintas condiciones.
     - *Costo*: Evaluar el costo operativo y cualquier costo oculto de ejecutar el sistema en la nube.
     - *Optimización de recursos*: Uso eficiente de los recursos virtuales, tanto localmente como en la nube.
     - *Seguridad*: Políticas y prácticas para mantener la integridad y confidencialidad de los datos.

El diagrama anterior ilustra la arquitectura propuesta para el proyecto, destacando la separación y comunicación entre los componentes Frontend, Backend y Base de Datos.

4.2. Conclusiones

Recuerda que el informe final debe ser claro, conciso y bien presentado. Cada sección debe estar bien estructurada para facilitar la lectura y comprensión del proyecto desarrollado.

5. Condiciones de Entrega

El proyecto deberá ser entregado en un repositorio de Github Classroom con la siguiente estructura:

- La fecha de entrega es **Lunes, 01 de diciembre de 2025 a las 17:58:59** por el enlace de Classroom disponible en el enlace de entrega en el campus virtual. **No se aceptan entregas por ningún otro medio**.
- Debe inscribir su grupo en el campus virtual hasta el día **Lunes, 24 de noviembre de 2025 a las 17:58:59**. Los grupos deben ser de 4 o 5 estudiantes, grupos más pequeños serán disueltos y reubicados en otros grupos, un grupo de 4 estudiantes podría recibir un estudiante adicional.p
- Se espera que el **código fuente** de la solución esté organizado de manera clara y accesible. Se tienen 3 carpetas: `backend`, `frontend` y `database` para el entregue del código.
- El **enlace al repositorio** de Github debe ser enviado a través del campus virtual antes de la fecha límite.

Recuerde que el uso de buenas prácticas en el código y una adecuada documentación enriquecerán la percepción y comprensión de su trabajo. El informe debe ser entregado en formato Markdown dentro de la carpeta docs del proyecto y contener todas las explicaciones necesarias sobre las configuraciones y archivos creados para resolver el problema, así como detalles del flujo de implementación en la nube.

6. Rúbricas y Evaluación

Para la evaluación del proyecto final del curso de Infraestructuras con Docker, Kubernetes y Cloud, se utilizará una serie de criterios distribuidos en diferentes categorías. A continuación, se describen los aspectos que serán evaluados, así como las respectivas rúbricas y escalas de puntuación.

6.1. Rúbrica de evaluación

Criterio	Nivel 0 (0 pts)	Nivel 1 (5 pts)	Nivel 2 (10 pts)	Nivel 3 (15 pts)
Separación de Componentes: Código fuente	Sin separación clara de componentes	Solo se cumple una entre A, B o C	Se cumplen dos entre A, B o C	A) Carpetas separadas para Backend, Base de Datos y Frontend; B) Interfaces definidas para comunicación entre componentes; C) Documentación de la arquitectura modular incluida
Despliegue local	No se realiza el proceso	Solo se cumple una entre A, B o C	Se cumplen dos entre A, B o C	A) Dockerfile correctos para Backend y Frontend; B) `docker-compose.yml` o configuración de Kubernetes presente para despliegue local; C) Integración local funcional: los componentes se comunican entre sí
Uso de servicios en la nube	No utiliza servicios en la nube	Solo se cumple una entre A, B o C	Se cumplen dos entre A, B o C	A) Uso de servicios para la gestión de contenedores; B) Configuración de servicios evidenciada en el informe usando kubernetes; C) Servicios funcionan correctamente ocultando el Backend y la Base de Datos a acceso externo
Balanceador de Carga	Sin evidencia del balanceador de carga	Sólo se cumple una entre A, B o C	Se cumplen dos entre A, B o C	A) Se proporcionan evidencias de la configuración del balanceador en la nube; B) Se proporciona evidencia de métricas del balanceador de carga en la nube; C) La aplicación se encuentra replicada y conectada al balanceador de carga

Criterio	Nivel 0 (0 pts)	Nivel 1 (5 pts)	Nivel 2 (10 pts)	Nivel 3 (15 pts)
Integración para evaluación de código: Informe	No se incluye integración	Sólo se cumple una entre A, B o C	Se cumplen dos entre A, B o C	A) Pruebas automatizadas (unitarias o de integración) integradas; B) Documentación de los procesos de integración continua incluida; C) Flujo de CI/CD configurado y funcional
Claridad de Explicaciones: Informe	Explicaciones ausentes o confusas	Sólo se cumple una entre A, B o C	Se cumplen dos entre A, B o C	A) Explicaciones claras de la arquitectura y componentes; B) Ejemplos e ilustraciones presentes para respaldar conceptos; C) Terminología técnica correctamente definida y contextualizada
Descripción de Configuraciones: Informe	Sin descripción presente	Sólo se cumple una entre A, B o C	Se cumplen dos entre A, B o C	A) Descripción detallada de la configuración de Docker Compose o Kubernetes; B) Explicación clara de la configuración de la base de datos en la nube; C) Documentación de redes y políticas de seguridad en la nube presente
Análisis de Resultados: Informe	No se incluyen análisis	Sólo se cumple una entre A, B o C	Se cumplen dos entre A, B o C	A) Análisis de rendimiento en ambiente local con métricas; B) Análisis de latencia y escalabilidad en la nube; C) Comparativa de resultados local vs nube con datos cuantitativos

Criterio	Nivel 0 (0 pts)	Nivel 1 (5 pts)	Nivel 2 (10 pts)	Nivel 3 (15 pts)
Conclusiones	Conclusiones ausentes	Sólo se cumple una entre A, B o C	Se cumplen dos entre A, B o C	A) Conclusiones basadas en métricas obtenidas y análisis realizados; B) Reflexiones sobre retos enfrentados y soluciones implementadas; C) Recomendaciones para mejoras futuras y lecciones aprendidas

Tener en cuenta que las rúbricas del informe dependen de la correcta implementación de la solución local y en la nube. Por lo tanto, si no se cumple con la implementación de la solución local o de la nube no se tendrá en cuenta el informe, es decir que se tomará nivel 0 como puntuación.

6.2. Sustentación

El proyecto debe ser sustentado por todos los miembros del grupo, la nota de la sustentación es entre 0 y 1, la cual se multiplica por la nota grupal. Por ejemplo, si su grupo obtuvo 4.5 y la nota de sustentación fue 0.8, la nota final del grupo sería 4.5 * 0.8 = 3.6. En caso de no presentarse a la subsection su nota será 0.0.

6.3. Aporte al repositorio

La nota del proyecto se multiplicará por el siguiente factor, dependiendo de la cantidad de commits realizados por el estudiante:

1. Se considera como aporte al código, la generación de los docs (documentación), los archivos relacionados con el despligue (Yaml, Dockerfile, etc.) y el código fuente de la aplicación.

2. Sólo serán considerados el código en la rama principal (main o master), no se considerarán los cambios en ramas secundarias.

3. Los cambios al código deben ser significativos y no solo cambios de formato o comentarios, estos últimos son ignorados por la herramienta de Github

4. No se consideran importantes cambios relacionados con la generación de archivos temporales o de compilación, los cuales deben ser ignorados por el gitignore.

6.4. Calculo de la nota

Las rubricas suman un total de 135 puntos, para calcular la nota final se debe aplicar la siguiente formula:

$$\text{Nota final} = \left(\frac{\text{Puntos obtenidos}}{135}\right) \times 5$$

Recuerde que esta es la grupal. La individual se obtiene multiplicando el factor de desempeño por el factor de aporte al repositorio por la nota de proyecto.