Despliegue de una Aplicación Basada en Microservicios con Análisis de Datos Distribuidos en Clúster de Contenedores

Aragón-Vivas M. A., Astudillo-Ortega L. D., Marín-Villacorte D., y Molina-Castro D. S., Pinillo-Olave M. P.

Abstrac- This project presents the design of a modular architecture based on microservices deployed with Docker and orchestrated using Docker Swarm. A custom dataset on project and task management was created, processed using PySpark, and visualized through interactive dashboards. All data and results are managed from a MySQL database, integrated with the system's APIs. Functional and load tests showed stable performance, and the solution proved to be scalable and adaptable.

Keywords— Docker, Pyspark, Microservicios, Clúster de Contenedores, Dashboards, Visualización de Datos (key words)

INTRODUCCIÓN

En los últimos años, las arquitecturas de microservicios han ganado popularidad dentro del desarrollo de software debido a su enfoque modular que permite diseñar sistemas más escalables resilientes y alineados con sus equipos de desarrollo distribuidos [1]. Cada servicio puede ser desplegado y escalado de forma independiente, lo que facilita tanto el mantenimiento como la evolución tecnológica del sistema. Este enfoque permite dividir una aplicación en pequeños servicios que trabajan de forma conjunta, facilitando tanto el desarrollo como el crecimiento del sistema a medida que se incrementan los requerimientos.

Sin embargo, trabajar con microservicios también implica nuevos retos, especialmente relacionados con su implementación y despliegue en entornos reales. Por esta razón, es fundamental apoyarse en herramientas que ayuden a automatizar tareas como la creación de contenedores, la orquestación de servicios, y el monitoreo de su comportamiento en producción [2].

En este contexto, el presente proyecto tiene como objetivo El El proyecto consiste en diseñar e implementar una solución de infraestructura integral para el despliegue de una aplicación web de gestión de tareas y proyectos dentro de una empresa, basada en microservicios. Para ello, se integran herramientas de contenedorización como Docker, junto con mecanismos de orquestación mediante Docker Compose y Docker Swarm, permitiendo una gestión eficiente de entornos distribuidos. Asimismo, se incorpora un módulo de análisis de datos sobre un clúster de procesamiento distribuido utilizando Pyspark, con el propósito de generar visualizaciones interactivas a partir de datos estructurados provenientes de un conjunto de datos de fuente abierta [3].

Además, asegurar la escalabilidad y disponibilidad, la infraestructura se desplegará sobre máquinas virtuales en Microsoft Azure, permitiendo la creación de un clúster multi/nodo que soporta el procesamiento distribuido. Esta estrategia facilita la gestión flexible y confiable de los componentes del sistema en un entorno moderno y robusto.

La aplicación base, desarrollada en el primer módulo del curso, consiste en una solución backend y frontend orientada a servicios, que incluye funcionalidades como la asignación de tareas, el seguimiento de proyectos, la gestión de usuarios y la verificación del progreso. A partir de esta base funcional, el proyecto aborda su transformación en una arquitectura contenerizada, definiendo una topología distribuida con mecanismos de balanceo de carga, escalado dinámico y pruebas de rendimiento.

Este documento presenta de forma sistemática las etapas de análisis, diseño, implementación y validación de la solución propuesta, discutiendo los retos técnicos enfrentados, las decisiones arquitectónicas adoptadas y los resultados obtenidos en términos de rendimiento, estabilidad, escalabilidad y facilidad de despliegue.

GENERACIÓN Y SELECCIÓN DEL DATASET

Para el desarrollo del módulo de análisis de datos y visualización, se optó por la creación de un **dataset propio** que refleja un entorno realista de gestión de proyectos y tareas empresariales. Esta decisión permitió un mayor control sobre la estructura, volumen y calidad de los datos, así como la posibilidad de adaptarlos directamente a las funcionalidades de la aplicación desarrollada durante el primer módulo.

El dataset fue generado bajo el nombre df_company y posteriormente cargado en una base de datos MySQL a través del script init.sql, el cual define la estructura de las tablas y realiza la inserción inicial de los datos. Esta base de datos es utilizada tanto por los microservicios como por el módulo de análisis desarrollado en PySpark.

El dataset integra información proveniente de dos fuentes principales: por un lado, el registro de los proyectos en curso de una empresa; por otro, el seguimiento de tareas asignadas a empleados, Esta combinación permite analizar tanto el avance general de los proyectos como el desempeño individual.

Los principales campos incluidos son:

- **project_id, project_name, boss_id, staus, team_members:** Datos relacionados con proyectos en curso y sus responsabilidades.
- **id, empleado_id, task_id**, **fecha_asignacion, fecha_entrega_maxima, fecha_entrega, estado:** Información correspondiente a las tareas individuales.
- **calidad, iniciativa, comunicación, satisfaccion_cliente, calificacion_promedio:** indicadores utilizados para evaluar el desempeño de los empleados.

Este dataset fue diseñado para ser almacenado y gestionado en una base de datos (MySQL), desde la cual posteriormente se extraen los datos para ser procesados mediante PySpark. Los resultados del análisis se almacenan nuevamente en la base de datos en nuevas tablas, lo que permite que los microservicios accedan a ellos y los expongan a través de APIs. Finalmente, estos datos procesados son consumidos por el frontend web, brindando al usuario una visualización clara y dinámica del estado de los proyectos y el rendimiento del equipo.

ANÁLISIS ALTERNATIVAS INFRAESTRUCTURA

El diseño e implementación de una solución moderna basada en microservicios exige el uso de herramientas que permitan empaquetar, desplegar, escalar y comunicar componentes de manera eficiente. En este proyecto, se integraron tecnologías de infraestructura orientadas a automatizar el despliegue de servicios, facilitar su interacción y habilitar el procesamiento distribuido de datos con fines analíticos y visuales. A continuación, se presenta el análisis de las alternativas evaluadas y las decisiones tomadas en cada caso.

A. Contenerización con Docker

Para la contenerización de los servicios que conforman la aplicación, se empleó Docker, una tecnología ampliamente adoptada en la industria que permite empaquetar una aplicación junto con todas sus dependencias dentro de una imagen portátil [4]. Esta estrategia asegura que cada microservicio se ejecute de manera consistente en cualquier entorno, lo que reduce los errores asociados a discrepancias en las configuraciones y acelera tanto el desarrollo como el mantenimiento del sistema.

Cada componente funcional de la plataforma fue encapsulado en un contenedor independiente, utilizando un archivo **Dockerfile** específico para cada servicio. Esta separación lógica no solo habilita la ejecución aislada y segura de cada microservicio, sino que también permite aplicar escalabilidad horizontal, simplificar la gestión de versiones y facilitar la detección de fallos de forma localizada.

B. Orquestación con Docker Swarm

En el proceso de selección de la herramienta de orquestación, se evaluaron dos alternativas principales: Docker Swarm y Kubernetes. Si bien Kubernetes representa el estándar de facto para despliegues complejos a gran escala, se consideró que su adopción excedía los requerimientos de este proyecto, tanto en complejidad como en carga operativa. En su lugar, se optó por Docker Swarm, una solución más liviana y con integración directa con Docker, que resultó adecuada para los objetivos del sistema [5].

Docker Swarm permitió desplegar los servicios de forma distribuida, simular un entorno multi-nodo y evaluar el comportamiento de la arquitectura bajo diferentes niveles de carga. Entre sus principales ventajas se destacan:

- Configuración rápida y directa a partir de la CLI de Docker.
- Escalado automático de servicios mediante comandos simples.
- Balanceo de carga interno entre réplicas de servicios.
- Integración nativa con archivos docker-compose.yml.

Gracias a estas características, fue posible mantener el control total del clúster desde la consola del orquestador, lo que facilitó la administración y el monitoreo del sistema durante las pruebas de carga y escalabilidad.

C. Procesamiento de Datos con PySpark

Con el propósito de analizar y transformar los datos generados por la aplicación, se integró Pyspark, un motor de procesamiento distribuido diseñado para trabajar con grandes volúmenes de información [6]. A diferencia de enfoques tradicionales que operan sobre disco, PySpark realiza los procesos directamente en memoria, lo que se traduce en mejoras sustanciales en el rendimiento, especialmente para operaciones intensivas como filtrado, agregación y cálculo de métricas [3].

PySpark fue utilizado para procesar tanto datos simulados como datos generados por la propia plataforma. Los resultados obtenidos fueron estructurados y visualizados mediante dashboards, brindando información útil para la evaluación de la actividad del sistema. Además, se integró PySpark al entorno contenerizado para mantener coherencia en el despliegue y asegurar una comunicación fluida con los demás servicios del ecosistema.

ARQUITECTURA DE LA SOLUCIÓN

La solución propuesta está basada en una arquitectura modular y distribuida, donde cada componente del sistema es desplegado como un contenedor independiente mediante Docker. El sistema completo es orquestado con Docker Swarm, lo que permite escalar servicios individualmente, mantener su independencia y asegurar una comunicación eficiente entre ellos.

A. *Componentes del Sistema*

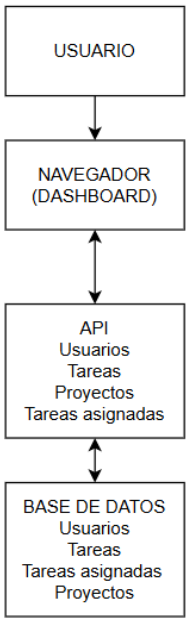
La solución está compuesta por los siguientes bloques funcionales:

- **Microservicios (Carpeta APIS/)**
  Cada subcarpeta dentro de **APIS** representa un microservicio específico:
  - Proyectos: gestiona información relacionada con los proyectos.
  - Tareas asignadas: administra las tareas asignadas a empleados.
  - Tareas: complementa funciones adicionales del sistema.
  - Users: maneja la gestión de usuarios.
  Todos estos servicios exponen endpoints a través de una API REST y se comunican con la base de datos de forma independiente o compartida según su lógica de negocio.
- **Procesamiento de datos (Carpeta DATA/)**
  Este módulo contiene el procesamiento analítico:
  - df_company, procesamiento_data y salida gestionan la entrada, análisis y salida de datos.
  - resultados: almacena los datos procesados (como métricas o reportes) que serán visualizados en el sistema.
  Aunque no se indica explícitamente Spark en la estructura, este módulo simula un flujo de procesamiento distribuido orientado al análisis de desempeño y gestión.
- **Base de Datos (Carpeta DATABASES/mysql/)**
  Se utiliza una base de datos MySQL inicializada mediante un script SQL (init.sql) que crea las tablas projects y tareasasignadas, con sus respectivas estructuras y registros simulados.
- **Interfaz Web (Carpeta WEB/)**
  La interfaz web está compuesta por varias páginas HTML (index.html, login.html, employee.html, boss.html) y scripts (scripts.js, styleBoss.css) que permiten la interacción del usuario con el sistema.
  Este frontend se comunica con los microservicios por medio de llamadas HTTP para visualizar y enviar información.
- **Archivo de orquestación (docker-compose.yml)**
  Este archivo define todos los servicios del sistema, redes, volúmenes y dependencias necesarias para levantar la solución completa de forma integrada. También es compatible con Docker Swarm para entornos distribuidos.
- **Orquestador (Docker Swarm)**
  Docker Swarm coordina la ejecución de los servicios como un clúster. Permite escalar los microservicios, distribuir la carga entre réplicas y reiniciar contenedores automáticamente en caso de fallo.
- **Microsoft Azure:**
  Su función es proveer infraestructura en la nube, incluyendo servicios de almacenamiento, bases de datos y recursos computacionales. Ofrece servicios para hospedar aplicaciones y gestionar recursos de forma escalable y segura[7].

B. *Flujo de Información*

1. El **usuario final** accede a la interfaz web, donde puede iniciar sesión y navegar entre las vistas disponibles según su rol (empleado o jefe de proyecto).
2. El **frontend (WEB)** envía solicitudes HTTP a los microservicios desplegados desde la carpeta APIS, los cuales procesan la información y acceden a la base de datos MySQL para consultar o actualizar registros.
3. Los datos almacenados en la base (projectscompany y TareasAsignadas) son accedidos también por el módulo de análisis ubicado en la carpeta DATA/, que se encarga de procesar esta información y generar métricas o reportes.
4. Los **resultados del procesamiento** son almacenados en archivos (.json, .csv, etc.) o rutas específicas dentro del contenedor, accesibles para el sistema web.
5. Finalmente, el **dashboard web** accede a estos resultados y los muestra de manera visual e interactiva para que el usuario pueda consultar el estado de los proyectos, desempeño de los empleados y otros indicadores clave.

C. *Di*agrama de Arquitectura



-
Gráfica 1. *Diagrama de arquitectura*.

DISEÑO DEL PIPELINE Y COMPONENTES

La solución propuesta se construyó siguiendo un pipeline de procesamiento estructurado en varias etapas, que permite transformar datos crudos almacenados en bases de datos relacionales en dashboards visuales interpretables por el usuario. Cada etapa del pipeline está a cargo de un componente independiente, desplegado como un microservicio en contenedores, lo que facilita su escalabilidad y mantenimiento.

D. Estructura General del Pipeline

El pipeline del sistema se compone de las siguientes fases:

1. **Extracción de datos**
   Los datos son consultados desde dos bases relacionales (**projectscompany** y **TareasAsignadas**) mediante la API REST. Esta API expone endpoints para acceder a información como proyectos activos, tareas asignadas, estados de avance y evaluaciones de desempeño.

2. **Procesamiento distribuido con PySpark**
   Un módulo específico ejecuta tareas de análisis utilizando PySpark. Este módulo lee los datos desde la base, realiza transformaciones, cálculos agregados y genera archivos intermedios con métricas como:

   a. Entregas a tiempo vs. entregas tardías.

   b. Promedio de desempeño por empleado

   c. Número de proyectos completados por jefe de proyecto

3. **Almacenamiento de resultados**
   Los resultados generados por Spark se almacenan en formato JSON o CSV, accesibles desde el servicio de visualización. Esta separación permite consultar resultados sin necesidad de procesar los datos en tiempo real, mejorando el rendimiento.

4. **Visualización con Dashboards**
   Una aplicación web, desplegada como microservicio, accede a los resultados y los representa mediante visualizaciones interactivas (gráficas de barras, tortas, tablas dinámicas, etc.). El objetivo es brindar al usuario una visión clara del estado de los proyectos y del rendimiento del personal.

E. Componentes del Sistema

Los componentes implementados en el sistema y sus responsabilidades son:

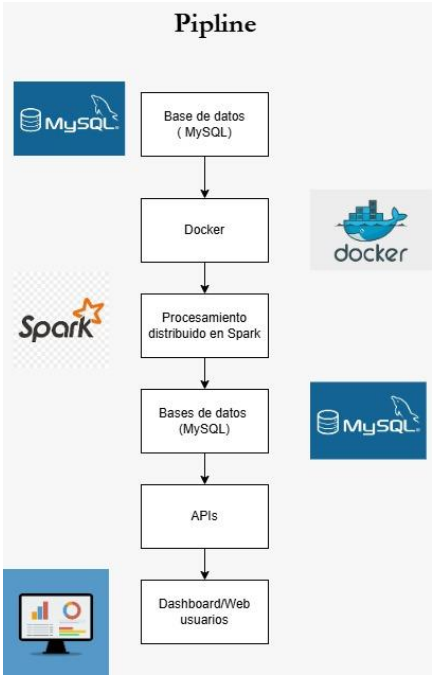
- **API REST (Backend)**
  Microservicio desarrollado en [Node.js / Python / otro], responsable de exponer los datos a los consumidores (dashboards o módulos externos). Gestiona la lógica de negocio relacionada con usuarios, proyectos y tareas.

- **Spark Processor**
  Script o servicio ejecutado con PySpark, encargado de procesar los datos de entrada. Está diseñado para ejecutarse periódicamente o bajo demanda y generar reportes listos para visualizar.

- **Dashboard Web**
  Interfaz visual que permite a los usuarios explorar los datos procesados. Puede estar desarrollada en Dash, React, u otra tecnología frontend, y se comunica con la API o accede directamente a los archivos de salida de Spark.

- **Base de Datos MySQL**
  Motor de almacenamiento relacional donde residen las tablas projects y tareasasignadas. Actúa como fuente de datos tanto para la API como para el módulo de análisis.

F. Diagrama del Pipeline



Grafica 2. Diagrama del pipline

IMPLEMENTACIÓN Y DESPLIEGUE

La implementación de la solución se basó en el principio de separación de responsabilidades, en donde cada componente del sistema fue desarrollado, empaquetado y desplegado como un microservicio utilizando contenedores Docker. Posteriormente, se utilizó Docker Swarm como plataforma de orquestación para gestionar el clúster de servicios de manera escalable y distribuida.

G. Contenerización con Docker

Cada componente del sistema fue encapsulado en su propio contenedor Docker mediante la creación de archivos **Dockerfile.** Estos archivos especificaron las dependencias necesarias, los comandos de instalación y la configuración para ejecutar los servicios en ambientes aislados.

Para facilitar el despliegue conjunto, se utilizó un archivo **docker-compose.yml** que define todos los servicios, redes, volúmenes y variables de entorno requeridas. Este archivo también sirve como base para el despliegue en Docker Swarm.

H. Despliegue con Docker Swarm

Para simular un entorno de infraestructura distribuida, se utilizó Docker Swarm, un orquestador que permite desplegar aplicaciones en clústeres. Algunos de los pasos ejecutados fueron:

1. **Inicializar Swarm**
2. **Desplegar el stack**
3. **Monitorear servicios**

Docker Swarm habilitó funcionalidades como:

- Balanceo de carga automático entre réplicas de servicios.
- Escalabilidad
- Tolerancia a fallos y reinicio automático de servicios caídos.

I. Visualización y Comunicación entre Servicios

El sistema fue diseñado para que los distintos servicios se comuniquen de forma eficiente mediante una red interna creada por Docker. Esta red permite que los contenedores interactúen entre sí usando los nombres de servicio defin

idos en el archivo docker-compose.yml, sin necesidad de exponer todos los puertos al exterior, lo que mejora la seguridad y el control del tráfico de datos.

Una vez que los datos son procesados por los módulos de análisis (ubicados en la carpeta DATA/), los resultados se almacenan en archivos intermedios —como .json o .csv— dentro de volúmenes compartidos. Esta información procesada es utilizada por el servicio web para construir la visualización final que verá el usuario.

La interfaz gráfica, desarrollada con HTML, CSS y JavaScript, permite a los usuarios consultar datos relevantes como el estado de los proyectos, desempeño de los empleados, cumplimiento de tareas y otros indicadores clave. Estas visualizaciones se generan a partir de los archivos de resultados o mediante solicitudes directas a los microservicios encargados de exponer la información.

Este diseño asegura que:

- **Los datos circulan de forma controlada** dentro del entorno de contenedores.
- **Cada componente tiene una responsabilidad clara**, desde la gestión de datos hasta su visualización.
- **El usuario final accede directamente a información útil y ya procesada**, sin necesidad de interactuar con la base de datos ni realizar cálculos adicionales.

PRUEBAS Y RESULTADOS

Las pruebas se diseñaron para evaluar el correcto funcionamiento del sistema y su comportamiento bajo distintas cargas de trabajo. Estas pruebas incluyeron validaciones funcionales, de carga y de escalabilidad, utilizando herramientas como Py JMeter y la línea de comandos de Docker Swarm para monitorear servicios.

J. *Pruebas funcionales*

Las pruebas funcionales tuvieron como objetivo verificar que cada uno de los microservicios implementara correctamente sus funcionalidades. Se realizaron solicitudes POST y GET a los distintos endpoints expuestos por los servicios app_proyectos, app_tareasasignadas, app_usersdillan, entre otros.

```
PS C:\Users\Steven\AppMicroserviciosDocker> docker service ls
ID             NAME                MODE         REPLICAS   IMAGE                          PORTS
wrbifwamfs04   app_db              replicated   1/1        steven462/db:v1                *:3333->3306/tcp
hwnxpenqucnv   app_proyectos       replicated   1/1        steven462/proyectos:v1         *:8001->8001/tcp
mmwtshfrpho6   app_tareas          replicated   1/1        steven462/tareas:v1            *:8002->8002/tcp
f2ofldwc1p8y   app_tareasasignadas  replicated   1/1        steven462/tareasasignadas:v1   *:8003->8003/tcp
r5m7x4v29ar8   app_usuarios        replicated   1/1        steven462/users:v1             *:8000->8000/tcp
jtcj77cn62kt   app_web             replicated   1/1        steven462/front:v1             *:8080->80/tcp
PS C:\Users\Steven\AppMicroserviciosDocker>
```

Figura 1. Servicios desplegados en Docker Swarm con una réplica cada uno.

Con una sola réplica del servicio app_tareasasignadas, se ejecutaron 20 solicitudes POST que fueron resueltas de manera correcta, lo que demostró el correcto funcionamiento básico del sistema.

K. *Pruebas de carga*

Se utilizaron pruebas de carga con Py JMeter para simular múltiples usuarios realizando solicitudes simultáneamente a los servicios. Se midieron métricas como:

- Tiempo promedio de respuesta.
- Solicitudes exitosas vs. fallidas.
- Rendimiento del sistema bajo condiciones de estrés.

Algunos resultados destacados:

- Con una sola réplica, se realizaron **consultas en bucle (loop infinito)** que arrojaron hasta **170,000 resultados efectivos** en servicios como `t`areasasignadas, projects y users, sin errores visibles en los logs.

Figura 2. Resultados de carga en el servicio tareasasignadas con loop infinito.

- En el endpoint http://appdocker.koreasouth.cloudapp.azure.com:8001/projects, se superaron los **60,000 resultados exitosos** con una sola réplica bajo una carga constante.

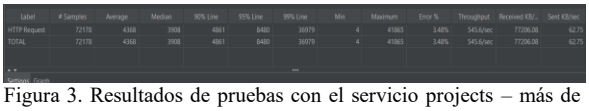


Figura 3. Resultados de pruebas con el servicio projects – más de 60,000 respuestas exitosas.

En el servicio http://appdocker.koreasouth.cloudapp.azure.com

- :8000/users, también con una sola réplica, se alcanzaron más de **100,000 solicitudes exitosas** sin errores.

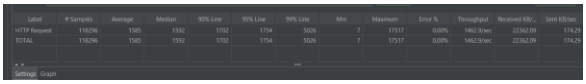


Figura 4. Servicio users – 100,000 respuestas exitosas sin errores.

Se realizó una prueba específica al endpoint http://appdocker.koreasouth.cloudapp.azure.com

- :8003/assignedT/2, enviando **5,000 solicitudes en 10 segundos**, todas con respuesta exitosa.

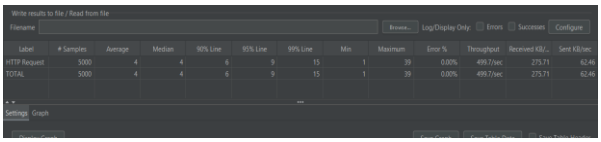


Figura 5. Resultado exitoso de 5,000 solicitudes en assignedT/2 en 10 segundos.

Estas pruebas permitieron identificar la capacidad máxima de respuesta de cada servicio individual y su comportamiento en condiciones intensas de uso.

L. *Pruebas de escalabilidad*

Se realizaron pruebas variando el número de réplicas en Docker Swarm para observar cómo cambiaba el rendimiento:

- Con **5 réplicas** y **50 solicitudes en 200 segundos**, el sistema presentó **más del 90 % de fallos**, lo que sugiere que no todos los nodos respondieron adecuadamente.

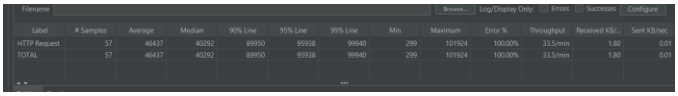


Figura 6. Fallos con 5 réplicas tras 50 solicitudes – más del 90 % de errores.

- Con **10 réplicas**, las **100 solicitudes volvieron a fallar** tras un tiempo de prueba extendido de 300 segundos. Se detectó que solo la primera solicitud fue atendida exitosamente, lo cual sugiere limitaciones en los recursos del equipo anfitrión.

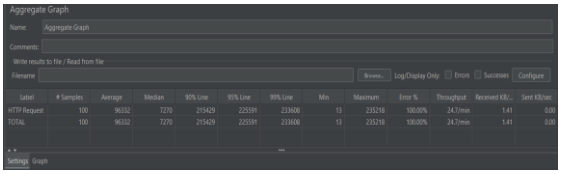


Figura 7. Prueba con 10 réplicas y 100 solicitudes – sólo la primera fue procesada.

- A pesar de escalar los servicios, no siempre se obtuvieron mejoras proporcionales, lo que indica que el **hardware disponible fue un factor limitante** durante las pruebas.

M. Conclusión de las pruebas

Los resultados muestran que, a nivel funcional, los servicios operan correctamente y son capaces de atender múltiples solicitudes en paralelo cuando la carga es moderada. Sin embargo, bajo pruebas más exigentes, el rendimiento se ve afectado por la capacidad de los equipos donde se ejecutó el clúster.

Aunque el sistema fue escalado correctamente con Docker Swarm, las mejoras esperadas no siempre se reflejaron debido a las **limitaciones del entorno local** (CPU, RAM y red). Esto demuestra que la arquitectura está preparada para escalar, pero que su eficiencia depende en gran medida de la infraestructura subyacente.

En general, las pruebas fueron útiles para verificar la estabilidad del sistema, identificar cuellos de botella y validar que los servicios pueden atender una carga considerable si se cuenta con los recursos adecuados.

VIII. CONCLUSIÓN

El presente proyecto logró implementar una arquitectura contenerizada basada en microservicios, con procesamiento distribuido y visualización dinámica de datos, utilizando herramientas como Docker, Docker Swarm, MySQL y PySpark. El uso de un dataset propio permitió adaptar los análisis a un contexto empresarial realista, optimizando la relación entre datos, procesamiento y visualización.

Las pruebas realizadas demostraron que el sistema es funcional, escalable y modular. Aunque las condiciones de hardware limitaron el rendimiento bajo alta carga, la arquitectura respondió de forma estable y coherente. El uso de Docker Swarm simplificó la gestión del clúster, y PySpark permitió automatizar el análisis de grandes volúmenes de datos, integrando los resultados directamente en la base de datos relacional.

En conjunto, la solución desarrollada evidencia una aplicación práctica de tecnologías modernas de infraestructura para sistemas distribuidos y orientados al análisis de datos.

REFERENCIAS

[1] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 1st ed., Sebastopol, CA: O'Reilly Media, 2015.

[2] [2] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux Journal*, vol. 2014, no. 239, pp. 2–11, Mar. 2014.

[3] M. Zaharia, B. Chambers, and M. Xin, *Spark: The Definitive Guide*, 1st ed. Sebastopol, CA: O'Reilly Media, 2018.

[4] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, pp. 2–2, 2014.

[5] Docker Inc., "Swarm mode overview," *Docker Documentation*. [Online]. Available: https://docs.docker.com/engine/swarm/ [Accessed: May 22, 2025].

[6] *Apache_Spark*. Accessed: Aug. 12, 2024. [Online]. Available: https://spark.apache.org/.

[7] Microsoft, "Introducción para desarrolladores de Azure," Microsoft Learn, 2023. [En línea]. Disponible: https://learn.microsoft.com/es-es/azure/developer/intro/azure-developer-overview. [Accedido: 22-may-2025].