DESPLIEGUE DE UN MODELO DE DEEP LEARNING PARA LA PREDICCIÓN DE BAJO GASTO CARDIACO A PARTIR DE SEÑALES HEMODINÁMICAS EN UNA UNIDAD DE CUIDADOS INTENSIVOS (DMDL-PC-SH-UCI)

DAVID ARTURO PEREZ DIAZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍAS
ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2025

DESPLIEGUE DE UN MODELO DE DEEP LEARNING PARA LA PREDICCIÓN DE BAJO GASTO CARDIACO A PARTIR DE SEÑALES HEMODINÁMICAS EN UNA UNIDAD DE CUIDADOS INTENSIVOS (DMDL-PC-SH-UCI).

DAVID ARTURO PEREZ DIAZ

Trabajo para optar al título de Ingeniero Electrónico

Director
Carlos A. Fajardo Ph.D

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍAS
ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2025

CONTENIDO

LISTA DE ANEXOS

RESUMEN

TÍTULO: DESPLIEGUE DE UN MODELO DE DEEP LEARNING PARA LA PREDICCIÓN DE BAJO GASTO CARDIACO A PARTIR DE SEÑALES HEMODINÁMICAS EN UNA UNIDAD DE CUIDADOS INTENSIVOS (DMDL-PC-SH-UCI) [*] .

AUTOR: DAVID ARTURO PEREZ DIAZ [**]

PALABRAS CLAVE: API, CONTENEDOR, PYTHON, PETICION, REDES NEURONALES, ORQUESTADOR.

DESCRIPCIÓN:

Este proyecto aborda la detección temprana de eventos cardíacos críticos (paro cardíaco, bajo gasto cardíaco y shock cardiogénico) en UCIs mediante inteligencia artificial. Los sistemas actuales presentan limitaciones en rapidez y confiabilidad, por lo que se desarrolló un modelo de deep learning entrenado con datos históricos de pacientes para analizar datos hemodinámicos en tiempo real.

Como contribución central, se implementó una API escalable desplegada en contenedores, capaz de responder hasta 1,000 solicitudes en 120 segundos en infraestructura hospitalaria real. La solución incorpora autenticación JWT para seguridad y orquestación que garantiza escalabilidad, representando un avance pionero en integración operativa clínica diaria.

Su impacto potencial incluye reducción de mortalidad mediante diagnóstico temprano, alivio de carga para personal médico y optimización de recursos hospitalarios. Esta herramienta establece un modelo replicable para medicina crítica, combinando velocidad de respuesta con robustez operativa en entornos de alta exigencia.

---

[*] Trabajo de grado

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Director:Carlos Augusto Fajardo, MS en Computer Science.

ABSTRACT

TITLE: DEPLOYMENT OF A DEEP LEARNING MODEL FOR PREDICTING LOW CARDIAC OUTPUT FROM HEMODYNAMIC SIGNALS IN AN INTENSIVE CARE UNIT (DMDL-PC-SH-ICU). *

AUTHOR: DAVID ARTURO PEREZ DIAZ **

KEYWORDS: NEURAL NETWORKS, REQUEST, API, CONTAINER, ORCHESTRATOR.

DESCRIPTION:

This project addresses the early detection of critical cardiac events (cardiac arrest, low cardiac output, and cardiogenic shock) in ICUs using artificial intelligence. Current monitoring systems face limitations in speed and reliability, prompting the development of a deep learning model trained on historical patient data to analyze real-time hemodynamic metrics.

The core contribution is a scalable API deployed via container orchestration, capable of processing up to 1,000 requests within 120 seconds on real hospital infrastructure. The solution integrates JWT authentication for security and orchestration for scalability, representing a pioneering advancement in daily clinical integration.

Potential impacts include reduced mortality through early diagnosis, decreased workload for medical staff, and optimized hospital resource allocation. This tool establishes a replicable framework for critical care medicine, combining rapid response with operational robustness in high-demand environments.

---

* Bachelor Thesis

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Director: Carlos Augusto Fajardo, MS en Computer Science.

INTRODUCCIÓN

En el marco de la investigación, este trabajo de grado explora las consideraciones esenciales para garantizar una relación efectiva entre el desarrollo algorítmico y la operación a gran escala en la industria hospitalaria. Concretamente, se enfoca en el diseño e implementación de una solución de software capaz de operar con alta concurrencia y seguridad, adaptándose a las especificaciones de infraestructura de un entorno hospitalario. Este proyecto propone una solución a la medida para operaciones de inferencia, abarcando la operatividad robusta y eficiente en condiciones reales de despliegue.

En estudios previos, como el de Hyeonhoon Lee y colaboradores [1], se presentan consideraciones para el diseño y desarrollo de un modelo de aprendizaje automático de tipo *boosting*, aplicado a la predicción de eventos críticos cardíacos mediante datos de variabilidad de la frecuencia cardíaca en intervalos de 1 a 24 horas. Este estudio se enfocó principalmente en hacer aportes con relación a la precisión de modelos predictivos, es decir, en la construcción del algoritmo sin abordar los retos asociados al despliegue en un entorno hospitalario real.

La solución propuesta se despliega mediante una API que expone los servicios como rutas HTTP, habilitando la consulta y ejecución de los modelos de *deep learning* en un servidor REST. La API se ha diseñado con una lógica asíncrona, que optimiza el uso de recursos de hardware en función de la latencia. La contenedorización de la aplicación en Docker Swarm facilita una gestión eficiente y escalable de los recursos, permitiendo además una implementación segura y controlada mediante autenticación JWT. De este modo, esta solución responde a las necesidades específicas de un entorno de cuidados intensivos, en el cual la rapidez y confiabilidad de los modelos de predicción son esen-

[1] H. Lee y et. al. «Real-time machine learning model to predict in-hospital cardiac arrest using heart rate variability in ICU». En: *NPJ digital medicine* 6.1 (2023), pág. 215. DOI: https://doi.org/10.1038/s41746-023-00960-2.

ciales.

Entre los resultados obtenidos, destaca el desarrollo de una interfaz de programación de aplicaciones que facilita la comunicación cliente-servidor para servicios de inteligencia artificial en unidades de cuidados intensivos. La API demuestra un rendimiento elevado en situaciones de alta concurrencia y multitudinalidad, con funcionalidades que incluyen la creación de usuarios, sesiones y autenticaciones, así como la gestión centralizada de la trazabilidad y los recursos de hardware. Este sistema no solo optimiza la respuesta en entornos críticos, sino que además constituye una plataforma adaptable a futuras necesidades de escalabilidad y mejora.

1. OBJETIVOS

OBJETIVOS ESPECIFICOS

- Desarrollar un proceso de despliegue eficiente y replicable para modelos de inteligencia artificial en entornos basados en Ubuntu.
- Implementar FastAPI como interfaz de programación para la comunicación entre el modelo de IA y las aplicaciones cliente.
- Utilizar contenedores Docker para garantizar la portabilidad y el aislamiento del entorno de ejecución del modelo.

---

2. ESTADO DEL ARTE.

El despliegue de modelos de deep learning en entornos clínicos reales, específicamente en unidades de cuidados intensivos (UCI), constituye un gran desafío debido a la necesidad de equilibrios rigurosos entre la eficiencia computacional, la confiabilidad de los sistemas y la seguridad de los datos de pacientes. En este trabajo de grado, se abordó este reto mediante la implementación de un servicio de predicción de eventos críticos, particularmente el shock hemodinámico y otras complicaciones cardiovasculares, en la infraestructura local de un hospital. Para ello, se diseñó una arquitectura centrada en contenedores Docker y se empleó el orquestador Docker Swarm para asegurar la escalabilidad y la alta disponibilidad del sistema.

La literatura respalda la importancia de estos esfuerzos de implementación real para la prevención de eventos cardíacos graves. En trabajos como el de Tseng et al [2], se presenta la introducción de un modelo de deep learning para la detección temprana de paros cardíacos en un hospital terciario, subrayando la necesidad de sinergia entre la innovación tecnológica y los flujos de trabajo clínicos establecidos. De manera similar en el trabajo de Li Yijing[3], se describe un sistema basado en deep learning para la detección en tiempo real de eventos cardiopulmonares en pacientes críticamente enfermos, haciendo hincapié en la importancia de contar con un procesamiento sostenido de datos de monitorización. Un trabajo un poco más desarrollado es el realizado por Lorenzo et. al.[4], en el cual se describe una solución análoga en la que se utilizan infraestructuras conte-

---

[2] Komorowski M. y et. al. «The Artificial Intelligence Clinician learns optimal treatment strategies for sepsis in intensive care». En: *Nat Med* 24.11 (2018), págs. 1716-1720. DOI: https://doi.org/10.1038/s41591-018-0213-5.

[3] Li Yijing et al. «Prediction of cardiac arrest in critically ill patients based on bedside vital signs monitoring». En: *Computer Methods and Programs in Biomedicine* 214 (2022), págs. 106-568. DOI: https://doi.org/10.1016/j.cmpb.2021.106568.

[4] Lorenzo Socias y et. al. «Application of a machine learning model for early prediction of in-hospital cardiac arrests: Retrospective observational cohort study». En: (2024).

nedorizadas para detectar inestabilidades hemodinámicas, demostrando la viabilidad de estos sistemas en la práctica clínica diaria.

3. TERMINOLOGIA Y PROPUESTA METODOLOGICA.

3.1. TERMINOLOGÍA

3.1.1. Conceptos previos Este documento asume el dominio de conceptos tales como:

1. Entornos de desarrollo.
2. Protocolo de comunicaciones HTTP.
3. Lógica asíncrona.
4. REST Application Programming Interface (API).
5. Protocolo IP.
6. Redes y puertos.
7. Encapsulado de ambientes.
8. Orquestación de servicios.

3.1.2. Tecnologías Durante los procesos de elaboración, implementación y despliegue se involucraron tecnologías como:

1. Python 3.11.0 .
2. SQLite.
3. JSON.
4. JWT.
5. HTTP.
6. OAuth 2.0.

7. Docker Swarm.

8. CLI Commands.

3.2. PROPUESTA METODOLÓGICA

El presente trabajo de grado tiene como objetivo principal desarrollar e implementar un proceso eficiente y replicable para el despliegue de un modelo de inteligencia artificial en una máquina virtual con Ubuntu, utilizando contenedores Docker y la tecnología FastAPI. Este enfoque garantiza un entorno flexible, escalable y portable para la ejecución de modelos de Deep Learning, facilitando su integración en aplicaciones y sistemas existentes.

3.2.1. Etapas de desarrollo Para alcanzar los objetivos propuestos, se seguirá una metodología estructurada en cuatro etapas:

1. **Preparación del Entorno:** Instalación y configuración de una máquina virtual con Ubuntu, asegurando la disponibilidad de los componentes necesarios para la ejecución de contenedores Docker.

2. **Desarrollo de la API:** Implementación de una API con FastAPI para exponer el modelo de Deep Learning y permitir la interacción con aplicaciones externas.

3. **Creación de Contenedores Docker:** Configuración de un entorno contenedorizado que encapsule el modelo de IA junto con sus dependencias, garantizando portabilidad y aislamiento.

4. **Despliegue y Pruebas:** Implementación del modelo en la máquina virtual y ejecución de pruebas para validar su funcionamiento y rendimiento en el entorno configurado.

4. DESCRIPCIÓN GENERAL Y DISEÑO DEL SISTEMA.

El entorno dedicado a producción y operaciones deberá poder recibir consultas y estar en la capacidad de comunicar sus resultados. Para ello, se implementa el protocolo HTTP, de modo que existirá la relación cliente-servidor, lo cual hace necesario la declaración de puertos y reglas del host.

Usando contenedores, la virtualización del ambiente es una de las condiciones de operación, lo que conlleva la necesidad de asignar las relaciones pertinentes entre los puertos del ambiente virtualizado y la máquina local host, permitiendo así la vía libre de peticiones y respuestas entre los clientes de la aplicación (otras aplicaciones con acceso a la red privada) y la máquina local host (máquina que contiene el contenedor en ejecución).

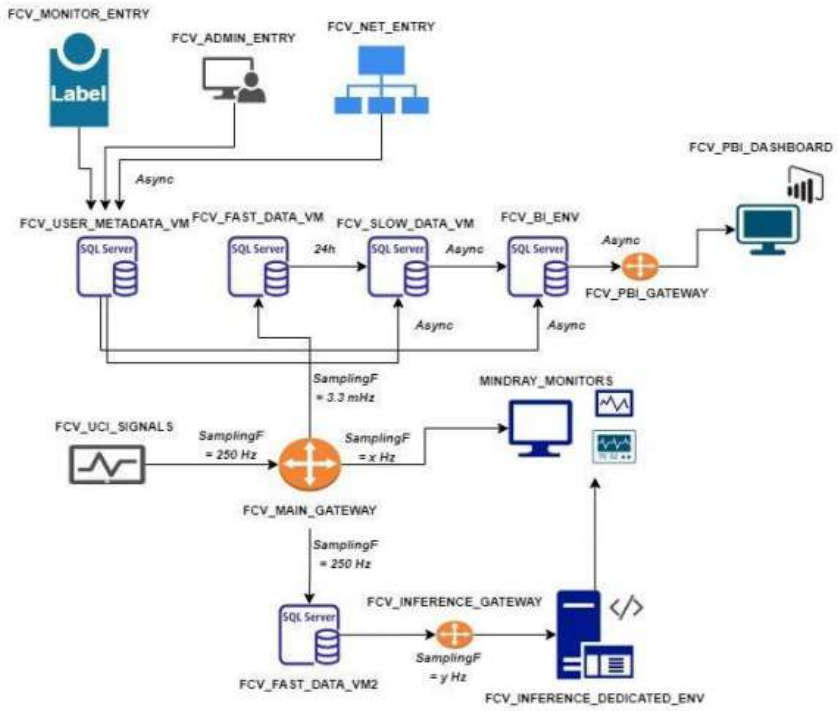

Figura 1. Infra-estructura de la solución

La figura 1 describe la composición actual de la infraestructura de software de la Fundación Cardiovascular de Colombia con sus diferentes entradas y salidas de datos.

4.1. CONSIDERACIONES PRINCIPALES

El modelo de *Deep Learning* expuesto está diseñado para realizar inferencias a partir de parámetros hemodinámicos históricos de cada paciente, considerando un rango temporal de 1 hora con 48 minutos (equivalente a 108 minutos). Esto implica que, para ejecutar una sesión de inferencia, se requiere la información hemodinámica del paciente correspondiente a los últimos 108 minutos previos a la consulta.
Con el fin de estandarizar el proceso de integración y despliegue, el Grupo de Investigación en Conectividad y Procesamiento de Señales (CPS) proporcionó un conjunto de artefactos que establecen el acuerdo de servicio. Dicho acuerdo define el formato de entrada del modelo, provee la versión oficial del modelo a ser desplegada, y especifica los requerimientos tecnológicos y de entorno necesarios para su correcta ejecución.
Los artefactos entregados son los siguientes:

- `LSTM.h5`: Modelo de *Deep Learning* en formato Keras.
- `predict.py`: Script encargado de cargar y preprocesar los datos, así como de cargar y ejecutar el modelo.
- `requirements.txt`: Archivo de texto plano que detalla las dependencias del entorno necesarias para la ejecución del modelo.

Dado que la frecuencia de muestreo de las señales hemodinámicas por paciente es de una muestra cada cinco minutos, se prevé que el intervalo entre cada sesión de inferencia sea de igual duración. Bajo la suposición de un alto flujo de pacientes, es razonable concluir que la solución desplegada deberá ser capaz de gestionar eficientemente múltiples solicitudes de inferencia realizadas de manera simultánea o con muy poca diferencia temporal. Por tanto, la aplicación deberá estar diseñada para recibir, procesar y responder múltiples consultas concurrentes provenientes de uno o varios usuarios.
Cada solicitud de inferencia enviada por un cliente deberá incluir los datos necesarios como parámetros de entrada. El formato esperado para esta información es el siguiente:

Listado 4.1. Objeto de entrada – Parámetros de inferencia

```
data_input = {
    'idAtencion': int,
    'idSigno': int,
    'nomSigno': str,
    'valor': float,
    'fecRegistro': str
}
```

El modelo de inferencia requiere como entrada un arreglo de datos con dimensiones que segmentan los registros por ID de usuario, variables hemodinámicas, y la cantidad de registros necesarios por lote de inferencia, que corresponde a 30 registros por cada variable. Este formato asegura que se cumpla con el rango temporal definido de 108 minutos (30 registros con una frecuencia de 5 minutos por registro).
La salida generada por el modelo corresponde a una ponderación de probabilidad entre 0 y 1, donde 0 representa el caso menos probable y 1 la mayor probabilidad de ocurrencia. En cada sesión de inferencia, el modelo evalúa y pondera cuatro posibles estados clínicos del paciente, los cuales están relacionados con su condición rítmica. Estos estados son:

- ***Arresto Cardíaco***
- ***Bajo Gasto Cardíaco***
- ***Sano***
- ***Shock Cardiogénico***

4.1.1. Formato de salida El servicio expuesto para realizar sesiones de inferencia está diseñado para generar una respuesta estructurada que incluya: el identificador del paciente asociado a la sesión, las ponderaciones obtenidas para cada posible estado clínico, y el estado con la mayor probabilidad. Esta respuesta será devuelta al cliente a través de una consulta HTTP en formato `JSON`.
A continuación, se presenta un ejemplo del objeto de salida:

Listado 4.2. Objeto de salida – Respuesta del sistema

```
response_object = {
    "idAtencion": 0,
    "inferences": ["string"],
    "State": "string"
}
```

El campo `idAtencion` identifica al paciente o episodio clínico evaluado. El campo `inferences` contiene un arreglo con las ponderaciones de probabilidad generadas por el modelo para cada uno de los posibles estados clínicos. Finalmente, el campo `State` representa el estado más probable, determinado a partir del valor máximo entre las ponderaciones.
El formato de salida ha sido diseñado para permitir la segmentación de resultados por usuario, lo que habilita dos modalidades de uso del servicio: inferencia individual, dirigida a un único paciente; o inferencia masiva, que evalúa múltiples pacientes de forma simultánea. En cualquiera de los casos, la respuesta del servicio será entregada al cliente en un documento `JSON`, el cual podrá ser consumido y procesado según los requerimientos de la aplicación cliente.

4.1.2. Frameworks La implementación del servidor se constituye mediante el framework FastAPI con el que se suplen requerimientos como tasas de consulta multitudinales, compatibilidad JSON, seguridad mediante el uso de tokens JWT para autenticar al cliente, así como permitir que el cliente pueda autenticar las respuestas del servidor. El token se empleará para validar al cliente durante el acceso al servicio, y cada cliente, a partir de sus credenciales, dispondrá de un token único para emitir consultas al servidor.

4.1.3. Dependencias Las dependencias y configuración del ambiente se prevén cubiertas al contemplar el despliegue de la aplicación como un contenedor Docker. Sin embargo, es importante aclarar que para el correcto funcionamiento de Docker se hace necesario contar con óptimas cantidades de memoria RAM, ya que este funciona basado en una máquina virtual.
También es necesario garantizar el acceso a este recurso mediante conexiones locales

o de red local. En otras palabras, se debe resolver la relación cliente-servidor en torno a identidades y métodos de autenticación.

Como se ha mencionado previamente, otro de los requerimientos para el correcto funcionamiento es proveer al host con el motor Docker, por lo que se hace necesaria la instalación del mismo antes de suponer cualquier despliegue.

4.2. INFRAESTRUCTURA INTERNA

La Figura 2 muestra la infraestructura interna de la aplicación y como esta interactúa con sus diferentes componentes.

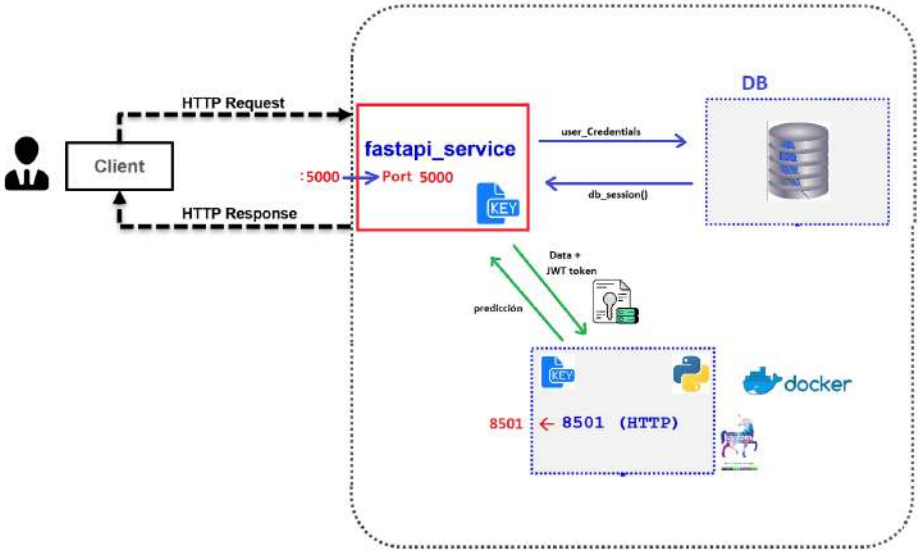

Figura 2. Interactividad interna

Bajo el protocolo de funcionamiento de una REST API, el cliente se comunica enviando peticiones HTTP a los dos diferentes componentes de la aplicación: gestión de sesión e inferencia. Cada aplicación trabaja con una imagen diferente, por lo que son dos contenedores distintos interactuando directamente con las solicitudes del cliente. Docker Swarm activa sus funcionalidades sobre el aplicativo de inferencia, creando réplicas del mismo y ejecutándolas en una red interna, permitiendo la paralelización de las solicitudes y aprovechando las capacidades de hardware del servidor.

4.3. COMPONENTES

4.3.1. Componente Principal La aplicación está construida utilizando `FastAPI` y proporciona servicios de autenticación, creación de usuario y comprobación de estado del modelo.

El archivo `main.py` se encarga de importar todos las dependencias necesarias e inicializar todas las clases, métodos y bases de datos. Este componente instancia todos los métodos de autenticación y expone todos los servicios de la aplicación, siendo el componente creador de la instancia de FastAPI, es también el encargado de llevar a cabo su ejecución en servidor y ser el gestor de casi todas las consultas y respuestas.

Cada servicio es expuesto como una URL bajo la red local, permitiendo acceder a estos servicios mediante un navegador común y corriente. A estos tipos de servicios se les conoce como endpoints y básicamente son los que componen todas las funcionalidades del aplicativo (Capitulo 6 - Manual de Usuario).

Ruta Principal Esta ruta verifica la autenticación del usuario y retorna la información del mismo si está autenticado.
URL: `http://172.30.19.97:5001/`

Ruta de Inferencia Esta ruta se encarga de ejecutar el modelo de deep learning alojado para la inferencia de eventos criticos.
URL: `http://172.30.19.97:5001/predict`

Registro y creación de usuario: Esta ruta le permite al administrador registrar nuevos usuarios en la base de datos para el acceso a los servicios del aplicativo.
URL: `http://172.30.19.97:5001/create_user`

Solicitud de la sesión Esta ruta asigna al usuario una sesión con caducidad definida establecida por el administrador. Ninguna funcionalidad estaría disponible sin que el usuario cuente con una sesión vigente.
URL: `http://172.30.19.97:5001/token`

Para una ampliación de la información relacionada con el componente principal, remitirse al Manual de Usuario Capitulo 4 sección 4.3.1.

4.3.2. Componente de registros Este artefacto contiene las pautas y configuraciones para el registro de la ejecucion y su trazabilidad.

Todas las funcionalidades o servicios del aplicativo estan bajo monitoreo y registro en el archivo `app.log`, permitiendo al administrador entender posibles fallas en la ejecución o limitaciones y detalles de rendimiento del sistema.

Toda la trazabilidad esta configurada para ser consignada de forma central, por lo cual todos los registros son generados bajo un formato definido, permitiendo la accionabilidad de los datos de ejecución.

Para una extensión de la información relacionada al registro de la ejecución y la generación de logs, remitirse al Manual de Usuario Capitulo 4 sección 4.3.2.

4.3.3. Componente de inferencia Este artefacto, de manera analoga al Componente Principal, constituye un segundo aplicativo, ejecutando en modo servidor, dedicado a los modelos de aprendizaje profundo alojados en el.
Inicialmente fue diseñado para alojar dos endpoints y una primera versión de un modelo de aprendizaje profundo para la predicción de eventos cardíacos.

Ruta Principal Esta ruta verifica la autenticación del usuario y retorna la información de la versión en uso del modelo de aprendizaje profundo.
URL: `http://172.30.19.97:5002/`.

Ruta de Inferencia Esta ruta se encarga de ejecutar el modelo de deep learning alojado para la inferencia de eventos críticos.
URL: `http://172.30.19.97:5002/models`.

De la misma forma que en el Componente Principal, este servidor también hace registro continuo de su ejecución mediante el mismo componente de registros.

Toda la información sobre su desarrollo puede ser encontrada en el Manual de Usuario Capitulo 4 sección 4.3.3.

4.3.4. Seguridad y cifrado de datos Este componente expone las funcionalidades implementadas con FastAPI, SQLAlchemy y otras librerías de autenticación y encriptación. Su objetivo principal es proporcionar un sistema de autenticación basado en tokens para una aplicación web.

Este artefacto define todo el protocolo de seguridad necesario para acceder a los servicios expuestos por la aplicación. Todos los datos sensibles como contraseñas y sesiones, son validadas a través de algoritmos de encriptación de forma que todo la transaccionalidad de los datos es cifrada.

Para ampliación de esta información, remitirse al Manual de Usuario Capitulo 4 sección 4.3.4.

4.3.5. Componente base de datos La selección del tipo de servicio para alojar datos ha sido basada principalmente en el nivel de concurrencia que este pueda llegar a experimentar y el tamaño de los datos almacenados.

Para este proyecto la base de datos es solo con propósitos de almacenar la información relacionada a la seguridad de la aplicación y mas precisamente a sus usuarios por lo que, se estima que su frecuencia de escritura sea baja y su frecuencia de lectura sea media - baja.
SQLAlchemy, un ORM (Object-Relational Mapping) y un toolkit de Microsoft SQL para Python que permite interactuar con bases de datos relacionales (como PostgreSQL, MySQL, SQLite, etc.) de manera más intuitiva, usando objetos y clases en lugar de escribir consultas SQL directamente fue la elección de preferencia para implementar la base de datos de la seguridad.

Consultar la el capitulo 4 sección 4.3.5 del Manual de Usuario para ampliación de la información relacionada al desarrollo de este componente.

5. ORQUESTACIÓN DE DESPLIEGUE.

Docker es una plataforma de contenedorización que permite empaquetar aplicaciones y sus dependencias en contenedores ligeros y portátiles, garantizando consistencia entre entornos. Facilita el desarrollo, pruebas y despliegue mediante integración continua (CI/CD). Entre sus ventajas destacan la portabilidad, aislamiento de procesos, escalabilidad horizontal (usando Docker Swarm o Kubernetes) y eficiencia en recursos. Sin embargo, presenta desafíos como una curva de aprendizaje inicial, complejidad en la gestión de almacenamiento persistente y posibles riesgos de seguridad si no se configura adecuadamente.

Un componente clave es el Dockerfile, un script que define cómo construir una imagen de Docker. Incluye instrucciones como FROM (imagen base), COPY (transferencia de archivos), RUN (instalación de dependencias) y CMD (comando de inicio). Adicionalmente, el archivo .dockerignore excluye archivos innecesarios (como __pycache__ o .env), optimizando el tamaño y seguridad de la imagen.

Para aplicaciones multi-contenedor, Docker Compose simplifica la orquestación mediante un archivo YAML (como compose-config-swarm.yaml). Este permite definir servicios, redes, volúmenes y políticas de despliegue (réplicas, límites de recursos y reinicios automáticos). Por ejemplo, en entornos Swarm, se pueden gestionar clusters con balanceo de carga y alta disponibilidad.

Remitirse al capitulo 5 del Manual de Usuario para extender esta información en mayor detalle.

6. CASOS DE USUARIO

6.1. DESPLIEGUE BÁSICO CON DOCKER

El flujo de implementación requiere Docker instalado en el sistemas host. Los pasos clave incluyen:

1. Ejecución de `docker-compose up -d` desde el directorio raíz que contiene los archivos de configuración
2. Verificación del estado con `docker ps -a` para confirmar la creación correcta de contenedores
3. Monitoreo de logs mediante `docker-compose logs` para diagnóstico operativo

Los mensajes de advertencia sobre drivers CUDA reflejan la configuración sin GPU del ambiente host, pero no comprometen la funcionalidad básica del sistema. Las pruebas de conectividad mediante peticiones HTTP al endpoint `/docs` confirman la correcta exposición de puertos y comunicación entre contenedor y host.

6.2. ARQUITECTURA DE LA SOLUCION

La aplicación implementa una arquitectura dual con:

6.2.1. Componente principal (App1)

- Puerto 5000
- Gestión de autenticación (JWT)
- Control de usuarios y sesiones
- Integración con base de datos

6.2.2. Servicio de inferencia (App2)

- Puerto 5001
- Procesamiento de modelos .keras
- API REST para predicciones
- Sistema de logging integrado

6.3. ORQUESTACIÓN AVANZADA CON SWARM

La configuración Swarm en `compose-config-swarm.yml` habilita capacidades empresariales:

Tabla 1. Parámetros clave de despliegue Swarm

Parámetro	Valor
Réplicas	3-10 instancias
Límite CPU	1 core por contenedor
Memoria	1024MB por contenedor
Política de reinicio	on-failure
Red	overlay network

El despliegue se gestiona mediante:

```
docker swarm init
docker stack deploy -c compose-config-swarm.yml <nombre>
```

6.4. PRUEBAS DE RENDIMIENTO

Las métricas de desempeño se recopilaron mediante `deployment_test.py`, destacando:

6.5. AUTOMATIZACIÓN DE DESPLIEGUE

El HIC implementó scripts de automatización para integración continua:

- `main_docker.sh`: Construcción de imagen para App1

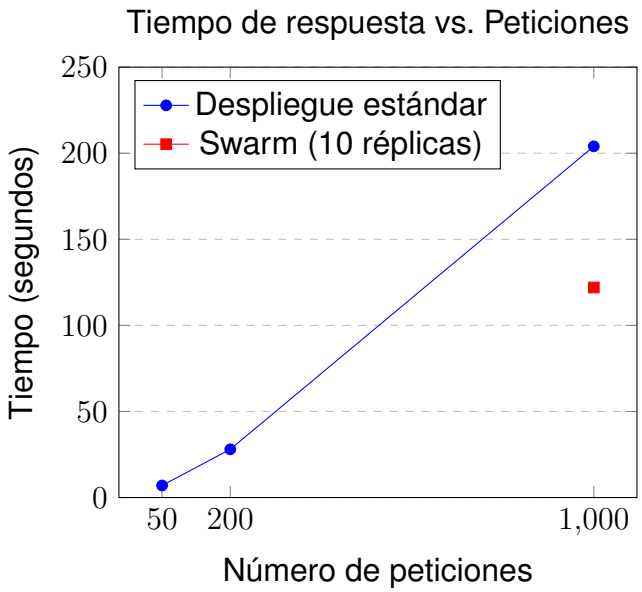

Figura 3. Escalabilidad comparativa

- `predict_docker.sh`: Creación de imagen para servicio de IA
- `main_cont.sh`: Configuración de red y volúmenes para App1
- `predict_cont.sh`: Despliegue del servicio de inferencia

6.6. RECOMENDACIONES OPERATIVAS

Basado en las pruebas realizadas:

- Usar Swarm para cargas superiores a 500 peticiones concurrentes
- Asignar mínimo 1GB RAM por instancia de inferencia
- Monitorear temperatura CPU en despliegues prolongados
- Considerar Kubernetes para escalamiento beyond 15 réplicas

Remitirse a el Manual de Usuario Capitulo

7. CONCLUSIONES Y REFERENCIAS

7.1. CONCLUSIONES

A partir de los objetivos planteados en el plan de trabajo de grado, se confirma que todos fueron alcanzados con éxito. Además, los resultados obtenidos superaron el alcance inicialmente propuesto, logrando avances adicionales en la optimización y evaluación del desempeño del sistema.
Con base en la implementación realizada y los resultados obtenidos, se concluye que:

- La aplicación desarrollada es compatible con frameworks de *Deep Learning* y cuenta con la lógica necesaria para garantizar la correcta ingesta de datos, asegurando que estos cumplan con los requisitos necesarios para ser utilizados como parámetros de inferencia.

- La seguridad del sistema está garantizada mediante el uso de transacciones *JWT* para la autenticación y la gestión de credenciales, respaldadas por una base de datos *SQL*.

- La solución es altamente compatible y adaptable, ya que no depende de un sistema operativo específico. Puede ser desplegada en cualquier entorno que cuente con Docker, lo que permite una gran flexibilidad en su implementación.

- El procesamiento de consultas no sigue un esquema secuencial ni sincronizado con una referencia específica. La solución es capaz de manejar peticiones de forma asíncrona, permitiendo la ejecución concurrente de múltiples procesos sin restricciones de orden.

- La arquitectura desarrollada ofrece capacidades de escalabilidad y optimización de latencia, lo que la hace adaptable a futuras versiones de modelos de *Deep Learning* con diferentes tasas de muestreo.

- Con la infraestructura actual del Hospital Internacional de Colombia, la solución propuesta es capaz de procesar solicitudes para hasta 2000 camillas de UCI en menos de 130 segundos, lo que demuestra su eficiencia y viabilidad operativa.

7.2. REFERENCIAS

- M. J. H. Faruk, J. Basney and J. Q. Cheng, "Blockchain-Based Decentralized Verifiable Credentials: Leveraging Smart Contracts for Privacy-Preserving Authentication Mechanisms to Enhance Data Security in Scientific Data Access,"2023 IEEE International Conference on Big Data (BigData), Sorrento, Italy, 2023, pp. 5493-5502, doi: 10.1109/BigData59044.2023.10386360.

- P. Solapurkar, "Building secure healthcare services using OAuth 2.0 and JSON web token in IOT cloud scenario,"2016 2nd International Conference on Contemporary Computing and Informatics (IC3I), Greater Noida, India, 2016, pp. 99-104, doi: 10.1109/IC3I.2016.7917942.

- Mohammad-Rahimi H, Sohrabniya F, Ourang SA, Dianat O, Aminoshariae A, Nagendrababu V, Dummer PMH, Duncan HF, Nosrat A. Artificial intelligence in endodontics: Data preparation, clinical applications, ethical considerations, limitations, and future directions. Int Endod J. 2024 Nov;57(11):1566-1595. doi: 10.1111/iej.14128. Epub 2024 Jul 29. PMID: 39075670.

- João Guedes, Júlio Duarte, Maria Manuel, César Quintas, João Cunha, Tiago Guimarães, Manuel Filipe Santos,Interoperability Architecture proposal for Adaptive Business Intelligence Systems in Healthcare Environments,Procedia Computer Science,https://doi.org/10.1016/j.procs.2024.06.113.

- Rubulotta, Francesca MD, PhD, MBA1; Bahrami, Sahar PhD1; Marshall, Dominic C. MBBS2; Komorowski, Matthieu MD, PhD2. Machine Learning Tools for Acute Respiratory Distress Syndrome Detection and Prediction. Critical Care Medicine 52(11):p 1768-1780, November 2024. | DOI: 10.1097/CCM.0000000000006390

Escrito por —

David Perez: www.linkedin.com

Referencia GitHub: GitHub.com

BIBLIOGRAFÍA

Lee, H. y et. al. «Real-time machine learning model to predict in-hospital cardiac arrest using heart rate variability in ICU». En: *NPJ digital medicine* 6.1 (2023), pág. 215. DOI: https://doi.org/10.1038/s41746-023-00960-2 (vid. pág. 8).

M., Komorowski y et. al. «The Artificial Intelligence Clinician learns optimal treatment strategies for sepsis in intensive care». En: *Nat Med* 24.11 (2018), págs. 1716-1720. DOI: https://doi.org/10.1038/s41591-018-0213-5 (vid. pág. 11).

Socias, Lorenzo y et. al. «Application of a machine learning model for early prediction of in-hospital cardiac arrests: Retrospective observational cohort study». En: (2024) (vid. pág. 11).

Yijing, Li et al. «Prediction of cardiac arrest in critically ill patients based on bedside vital signs monitoring». En: *Computer Methods and Programs in Biomedicine* 214 (2022), págs. 106-568. DOI: https://doi.org/10.1016/j.cmpb.2021.106568 (vid. pág. 11).

ANEXOS

Anexo A. Manual de Usuario.

El presente anexo describe los planos correspondientes al encapsulado utilizado en el desarrollo de esta tesis, proporcionando información detallada acerca de sus características y especificaciones técnicas. Los planos presentan las dimensiones, formas y detalles constructivos del encapsulado, lo que permite una mejor compresión de su diseño y funcionamiento.

La inclusión del Manual de Usuario en este anexo es fundamental para facilitar la compresión de los detalles técnicos del encapsulado, lo que resulta relevante para futuras investigaciones y desarrollos en el campo. De esta forma, se garantiza que la información contenida en los planos sea accesible y comprensible para cualquier interesado en el tema.

Remitirse a Manual de Usuario