Universidad Panamericana
Maestría en Ciencia de Datos
Datos Masivos

Proyecto Final: *Monitoreo Inteligente de Tráfico Marítimo con Big Data AIS*

Enrique Ulises Báez Gómez Tagle, Luis Alejandro Guillén Alvarez

24 de septiembre de 2025

Índice

1. Resumen ejecutivo

1.1. Introducción

El tráfico marítimo global genera millones de registros de posicionamiento a través del Sistema de Identificación Automática (AIS), que transmite automáticamente la identidad, posición, velocidad y otros datos de los buques. Originalmente implementado para mejorar la seguridad de la navegación, hoy es crucial para la gestión del tráfico marítimo, la conciencia del entorno y operaciones de búsqueda y rescate.

Este proyecto propone aprovechar Big Data y aprendizaje automático (ML) para analizar una base de datos masiva de mensajes AIS y ofrecer una solución innovadora a la pregunta: **¿Cómo detectar y entender comportamientos anómalos en el tráfico marítimo para mejorar la seguridad y la eficiencia?**

Responder esta pregunta implica analizar patrones normales de navegación y descubrir desviaciones significativas. En esencia:

- **¿Qué estamos haciendo?** Diseñamos un sistema de análisis que procesa grandes volúmenes de datos AIS para identificar anomalías en el comportamiento de los buques (posiciones fuera de lugar, velocidades inusuales, maniobras erráticas, etc.) y extraer patrones útiles sobre la operación de diferentes tipos de embarcaciones.
- **¿Para qué lo hacemos?** Para mejorar la seguridad marítima, la gestión del tráfico y la toma de decisiones, ofreciendo alertas tempranas de potenciales riesgos (colisiones, actividades ilícitas o fallos) y conocimiento profundo a autoridades y empresas navieras.
- **¿Cómo lo hacemos?** Empleando herramientas de Big Data de la Suite de Google Cloud (Storage + DataProc + BigQuery) para procesar datos geoespaciales masivos en en poco tiempo, complementado con modelos de ML que aprenden patrones habituales y detectan comportamientos que se apartan de lo normal.
- **¿A quién beneficia?** A organismos de seguridad marítima (marinas, guardacostas), a empresas navieras optimizando rutas y monitoreo de flotas, a aseguradoras evaluando riesgos operativos e incluso a investigadores del medio marino en estudios ecológicos.

Finalmente, los resultados se integraron en un *dashboard* interactivo con visualizaciones (mapas geográficos, gráficos de tendencias y rankings) que facilitan la interpretación y la toma de decisiones.

2. Visión general del desarrollo

2.1. Solución Actual / visión general

La solución actual se implementa de extremo a extremo sobre la nube de Google Cloud. El flujo completo es el siguiente:

- Una máquina virtual en Google Compute Engine se encarga de **scrapear los datos AIS** y cargarlos en Google Cloud Storage.
- En Cloud Storage se organizan dos capas diferenciadas: *raw* y *curated*, con prefijos de carpeta y particiones por mes (`YYYY-MM=`).
- Un clúster de Google Cloud DataProc (perfil *high-memory*; véase Sección 4.1) ejecuta dos jobs, habilitando procesamientos por series de meses con menos derrames a disco y menor tiempo total:
  - Job **raw**: descomprime los archivos obtenidos y organiza los datos en particiones por año y mes, y finalmente los guarda en formato Parquet en Google Cloud Storage.
  - Job **curated**: lee la capa cruda, aplica transformaciones iniciales y genera la capa refinada lista para análisis.

- Una Cloud Function crea el **dataset y la tabla en BigQuery**, cargando la información desde la capa curated.
- En BigQuery se centraliza la explotación de datos mediante consultas SQL optimizadas, generando los datasets finales que alimentan las visualizaciones.
- El **dashboard de Streamlit**, desplegado en la misma VM, consume los resultados de BigQuery y presenta mapas, gráficos de tendencias y rankings de manera interactiva.

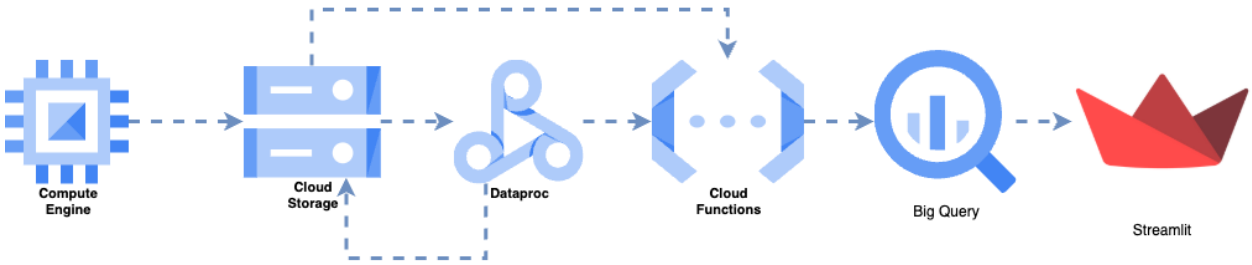

Figura 1: Arquitectura actual de la solución: flujo end-to-end en Google Cloud.

2.2. Limitaciones actuales de la solución

- **Sin orquestación event-driven.** Los disparadores son manuales o programados; la activación automática por eventos de GCS/Pub Sub o Workflows aún no está implementada. Migrar a un esquema event-driven reduciría intervención manual y tiempos de ciclo.
- **No tiempo real.** El pipeline es *batch*; no hay ingesta en streaming ni SLA de latencia. El dashboard refleja cortes al cierre de cada corrida.
- **Evolución de esquema limitada.** La Cloud Function crea tablas y puede autodetectar esquema, pero no hay un proceso formal de migraciones/versionado ni *backfills* automatizados.
- **Pruebas y monitoreo.** Podrían generarse alertas proactivas (Cloud Monitoring/Error Reporting) ante fallas o derivas.

2.3. Propósito, uso y alcance de la herramienta

3. Revisión y uso de datos

3.1. Orígenes y control de datos

Los datos utilizados provienen del portal oficial de la NOAA (`https://coast.noaa.gov/htdata/CMSP/AISDataHandler/2024/index.html`), el cual publica archivos comprimidos (.zip) con los mensajes AIS recolectados a lo largo del año 2024. Para garantizar la trazabilidad y el control de los datos, se desarrolló un script en Python (`scrapper.py`) que automatiza la descarga de todos los archivos disponibles, verificando duplicados mediante el tamaño de los ficheros y evitando descargas innecesarias. De esta manera se asegura que la totalidad de los datos originales publicados por la NOAA sean capturados sin alteración.

3.2. Preparación de datos

En esta sección se describe el **qué** produce la preparación de datos (entradas, salidas y *esquema*); la orquestación y operación del pipeline (el **cómo**) se documenta en la Sección 4.1.

La preparación de los datos comienza en una máquina virtual de Google Compute Engine, donde se ejecuta el script de *scraping*. Este proceso descarga en paralelo (con *threads*) los archivos `.zip` hacia un directorio local (`ais_2024`) y posteriormente los traslada a Google Cloud Storage en la capa **raw**.

Dado el objetivo de tener un conjunto uniforme, trazable y listo para análisis, con: (i) esquema estandarizado, (ii) claves y derivadas temporales consistentes, (iii) coordenadas y velocidades normalizadas, y (iv) datos particionados por `ym` (AAAA-MM).

Entradas → salidas por etapa y la salida de cada una.

1. **RAW INGESTION**: entrada = lotes `.zip` de NOAA; salida = archivos Parquet en **raw** *particionados por* ***ym***, con trazabilidad mínima (archivo fuente y timestamp de ingesta).
2. **CURATED TRANSFORMATIONS**: entrada = **raw**; salida = capa **curated** con tipos normalizados, derivados temporales y desduplicación por `MMSI+BaseDateTime`. Las reglas de limpieza y validaciones se detallan en la Sección 3.3.

Modelo de datos de la capa *curated*. El Cuadro 1 presenta los campos principales que expone la capa analítica.

Cuadro 1: Esquema de la capa *curated*.

Campo	Tipo	Descripción
NavStatusInt	INTEGER	Representación entera del estado de navegación del buque.
MMSI	STRING	Número de Identidad del Servicio Móvil Marítimo del buque.
BaseDateTime	TIMESTAMP	Marca temporal que indica la fecha y hora del mensaje AIS.
LAT	FLOAT	Latitud de la posición del buque.
LON	FLOAT	Longitud de la posición del buque.
SOG	FLOAT	Velocidad sobre el fondo del buque en nudos.
COG	FLOAT	Rumbo sobre el fondo del buque en grados.
Heading	FLOAT	Rumbo de la proa del buque en grados.
VesselName	STRING	Nombre del buque.
IMO	STRING	Número de la Organización Marítima Internacional del buque.
CallSign	STRING	Indicativo de llamada por radio del buque.
VesselType	STRING	Tipo de buque.
Status	STRING	Estado del buque.
Length	FLOAT	Eslora del buque.
Width	FLOAT	Manga del buque.
Draft	FLOAT	Calado del buque.
Cargo	STRING	Tipo de carga que transporta el buque.
TransceiverClass	STRING	Clase del transpondedor AIS.
_source_file	STRING	Nombre del archivo fuente desde el cual se ingirieron los datos.
_ingest_ts	TIMESTAMP	Marca temporal que indica cuándo se ingirieron los datos en la tabla.
ymd	STRING	Fecha en formato año-mes-día.
VesselTypeInt	INTEGER	Representación entera del tipo de buque.
VesselTypeCode	STRING	Código que representa el tipo de buque.
VesselTypeName	STRING	Nombre del tipo de buque.
VesselTypeClass	STRING	Clasificación del tipo de buque.
NavStatusName	STRING	Nombre del estado de navegación del buque.
date	DATE	Fecha en que se registró el mensaje AIS.
hour	INTEGER	Hora del día en que se registró el mensaje AIS.
dow	STRING	Día de la semana en que se registró el mensaje AIS.
week	INTEGER	Semana del año en que se registró el mensaje AIS.
month	INTEGER	Mes en que se registró el mensaje AIS.
quarter	INTEGER	Trimestre del año en que se registró el mensaje AIS.
SOG_ms	FLOAT	Velocidad sobre el fondo del buque en metros por segundo.
geohash9	STRING	Representación geohash de la posición del buque con precisión de 9 caracteres.

Una vez procesado, el resultado de este pipeline se publica en BigQuery como tabla particionada por `ym`, optimizada para consulta y consumo por el *dashboard*. Los detalles de automatización de creación/carga (Cloud Function, idempotencia y autodetección de esquema) se abordan en la Sección 4.1.

3.3. Limpieza y tratamiento de datos

En esta sección se describen de manera detallada las transformaciones y criterios aplicados para convertir los datos AIS en una capa **curated** funcional para el análisis. Se listan los pasos con su *porqué* operativo o estadístico.

Normalizaciones y tipos.

- **Casteo y estandarización**: se convierten columnas numéricas a tipos adecuados (`double/int`) y se normalizan textos (`trim`, mayúsculas controladas) para evitar variabilidad en joins/agrupaciones.
- **Timestamps en UTC**: `BaseDateTime` se parsea a tipo timestamp con zona UTC para análisis temporales coherentes; sobre esta base se derivan `date`, `hour`, `week`, `month`, `quarter` y una columna `ym` (AAAA-MM) para particionado.

Coordenadas y límites físicos.

- **LAT/LON válidos**: se recortan latitudes al rango [-90, 90] y longitudes se *envuelven* a [-180, 180] (corrigiendo valores fuera de dominio por errores/reporte). Se descartan registros con coordenadas nulas tras la corrección.
- **Redondeo razonable**: se redondean LAT/LON a 5 decimales para homogenizar precisión sin perder resolución náutica relevante.

Velocidad, rumbos y estados.

- **SOG (Speed Over Ground)**: se acota a [0, 70] nudos para eliminar outliers físicos/telemetría corrupta; adicionalmente se calcula `SOG_ms` en m/s para modelos.
- **COG/Heading**: se normaliza módulo 360°. El valor `Heading=511` (no disponible) se marca como nulo para evitar sesgos en métricas de rumbo.
- **NavStatus**: se genera `NavStatusInt` a partir de `Status` y se une contra un catálogo para `NavStatusName`; códigos fuera de 0–15 se etiquetan como *Unknown* y faltantes como *Not reported*.

Dimensiones y calado.

- **Length/Width**: se acotan a rangos plausibles (1–450 m y 1–70 m, respectivamente) para descartar errores de orden de magnitud.
- **Draft**: se limita a [0, 25] m; valores negativos o excesivos suelen provenir de carga mal reportada o errores de parsing.

Identificadores y texto.

- **MMSI**: se extrae la porción numérica válida (1–9 dígitos) y se rellena a 9 con ceros a la izquierda para formato consistente.
- **IMO/CallSign/VesselName**: se normalizan espacios y caracteres no alfanuméricos; se limpian IMOs genéricos (p.ej., `IMO0000000`, `0`, vacío) para evitar falsas uniones.

Clasificación de tipo de buque.

- **VesselType**: se derivan `VesselTypeInt`, `VesselTypeName` (mediante mapa IMO) y `VesselTypeClass` (WIG, Cargo, Tanker, Passenger, etc.). Esto permite comparar métricas por clase homogénea y soportar reglas dependientes del tipo.

Enriquecimiento geoespacial.

- **Geohash9**: se calcula un `geohash9` distribuido (UDF con `pygeohash`) para agregación espacial eficiente y enlaces a capas geográficas (vallas, puertos, TSS) en análisis posteriores.

Calidad e idempotencia.

- **Desduplicación**: se eliminan duplicados por clave compuesta `(MMSI, BaseDateTime)` para asegurar series temporales limpias por embarcación.
- **Validación de conteos**: en *raw*, se comparan conteos diarios (CSV vs Parquet) y se aborta si no coinciden; en *curated*, se escriben marcadores de éxito por partición (`_markers/ym=.../_SUCCESS`) para reanudación y ejecución incremental sin sobrescribir trabajo correcto.

3.4. Integridad de los datos

3.5. Limitaciones de los datos

4. Proceso de desarrollo

4.1. Metodología

Esta sección cubre el **cómo** se ejecuta y opera el pipeline (orquestación, confiabilidad, despliegue y costos); las reglas de transformación y el esquema se tratan en las Secciones 3.3 y 1.

Proceso de desarrollo.

- **Ejecución y calendarización**: actualmente los disparadores son manuales o programados; aún no se emplea un flujo completamente *event-driven* (ver Sección 2.2).
- **Orquestación y repetibilidad**: los jobs se ejecutan en DataProc con tolerancia a fallos (speculation, retries) e *idempotencia* (sobrescritura controlada por partición y marcadores `_SUCCESS`). Esto permite relanzar sin duplicados ni corrupción de particiones.
- **Publicación a BigQuery**: una Cloud Function idempotente verifica/crea **datasets** y **tablas**; si no hay esquema, lo autodetecta a partir de una muestra en GCS. Su objetivo es que la capa *curated* quede lista para consulta por el dashboard.
- **Particionado y costos**: el diseño de partición por `ym`, permite escaneos selectivos y costos predecibles.
- **Versionamiento de dependencias**: se distribuyen *wheels* (p.ej., `pygeohash`) y un entorno reproducible (tarball del venv) a los ejecutores vía `spark.yarn.dist.archives`, asegurando UDFs consistentes.
- **Observabilidad**: se registran conteos globales y por día, además de tiempos por etapa, para detectar desviaciones (picos inusuales, lag de E/S). Los logs documentan fallas y se retienen para auditoría.

Configuración del clúster (DataProc). La capacidad de cómputo y la configuración actual del clúster es:

Cuadro 2: Tamaño del clúster de DataProc y características de nodos.

Rol	Tipo	vCPU	Memoria
Maestro	e2-highmem-4	4	32 GB
Trabajador	e2-highmem-2	2	16 GB
Trabajador	e2-highmem-2	2	16 GB
Trabajador	e2-highmem-2	2	16 GB
Trabajador	e2-highmem-2	2	16 GB

Optimización y diferenciador. La configuración *high-memory* actual mitiga la restricción operativa de procesar estrictamente mes a mes: ahora es posible ejecutar ventanas de *series de meses* más amplias con menor probabilidad de *spills*/OOM y menos corridas totales. No modifica la arquitectura, pero sí incrementa la capacidad efectiva del pipeline y mejora el tiempo de pared global.

4.2. Pruebas

5. Resultados y conclusiones

6. Código utilizado

6.1. Link al repositorio con código fuente y salidas correspondientes

`https://github.com/enriquegomeztagle/MCD-BigData-SmartMaritimeTrafficMonitoring-FinalProject`