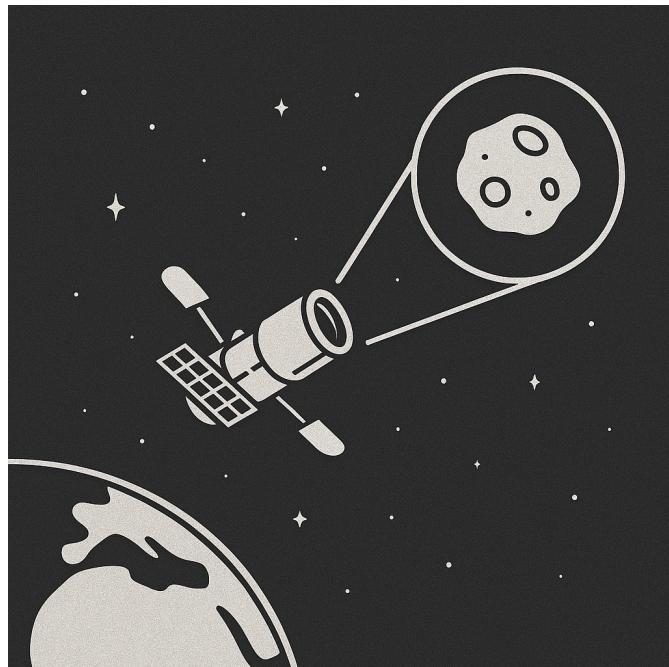


AsteroidTracking

Departamento de informática
I.E.S. Gran Capitán - Córdoba



Inteligencia artificial y Big data
3 de junio de 2025

Desarrollado por:
Víctor Páez Anguita

Índice

1. Requisitos	2
1.1. Proyectos. Requisitos mínimos	3
1.2. Consideraciones	3
2. Introducción	4
3. Arquitectura del sistema	5
4. Generación de datos sintéticos	6
5. Procesamiento de datos (ETL)	6
5.1. Consumidor	7
5.2. Transformación de datos	7
5.3. Almacenamiento en HDFS	11
6. Visualización	11
6.1. Prometheus + JMX Exporter	11
6.2. Exportador personalizado	12
6.3. Dashboards de Grafana	12
7. Business Intelligence	14
7.1. Detección y evaluación de riesgo en tiempo real	14
7.2. Simulación de escenarios y entrenamiento de modelos	14
7.3. Investigación científica	14
8. Conclusión	15
9. Webgrafía	16

1. Requisitos

- El proyecto deberá tener todo el stack de todos los sistemas vistos en clase perfectamente instalado, configurado y funcionando como un Sistema completo de Big Data, desde la ingestión de datos, ETL, BI y su visualización.
- El alumnado elegirá el origen, los tipos y la temática de los datos que se van a procesar en el Sistema Big Data.
- Deben establecer, desarrollar y justificar el tipo de conocimiento que van a obtener de los datos origen después de su ingestión y procesamiento (ETL) en el sistema.
- El procesamiento de los datos lo realizarán a través de SPARK, utilizando alguna de sus 3 APIs disponibles. Esto no quita que puedan realizar algún tipo de procesamiento de datos anteriormente, como por ejemplo en Kafka.
- El sistema debe poder soportar la ingestión de datos tanto en batch como en streaming.
- Los datos de origen podrán ser sintéticos, reales o una combinación de ambos.
- Puedes usar las API/s que creas necesaria/s. Incluso la creación de tus propios datos sintéticos en batch y streaming (Estos deben cumplir con los requisitos del puntos 1 al 3)
- Todo el ETL realizado deberá estar correctamente desarrollado y justificado.
- Se deberá añadir al stack algún sistema, servicio, ... de investigación propia (al menos 1, aunque puede añadir todos los que quieras). Se propone una lista de ellos, que podrán ser ampliados a propuesta del alumnado:
 - AWS GLUE
 - AWS S3
 - Nifi
 - Flink
 - Tableau
 - PowerBI
 - Elasticsearch
 - Kibana
 - RabbitMQ
 - Otros (deben ser consensuados y aprobados)
- La calificación del proyecto se hará de forma global, dependiendo de los niveles de aprendizaje que se demuestren superados acorde al nivel de conocimiento que exige el módulo del CE.

1.1. Proyectos. Requisitos mínimos

- El sistema completo será, como mínimo (más la investigación propia):
 - Apache Hadoop Common
 - HDFS
 - MapReduce
 - Yarn
 - SPARK
 - Grafana
- Debe haber como mínimo 3 nodos en los clusters (en cada uno):
 - Hadoop (HDFS/Yarn)
 - Spark
 - Kafka
- Añade todos los nodos que necesites para desplegar todo el stack Big Data del proyecto.
- Deben soportar acceso concurrente desde varios nodos Edge.

1.2. Consideraciones

- A mayor y mejor ETL y mayor y mejor Business Intelligence, mejor calificación.
- Si un proyecto no tiene suficiente procesamiento de datos y obtención de conocimiento, mayor será la exigencia de la investigación propia o viceversa.

2. Introducción

AsteroidTracking consiste en el desarrollo de un sistema completo de Big Data cuyo objetivo es la detección y análisis en tiempo real de objetos cercanos a la Tierra (NEOs, por sus siglas en inglés), como asteroides, cometas, satélites y otros tipos de objetos potencialmente peligrosos. La idea principal es obtener estos datos a partir de un conjunto de datos base real proporcionado por la API de la NASA [5, (Sentry System)]. El inconveniente es que este conjunto de datos no es lo suficientemente grande ni contiene datos en tiempo real, por lo que he optado por generar datos sintéticos.

El objetivo final es crear un business Intelligence que permita la visualización y análisis de estos datos, descubriendo patrones y tendencias que puedan ayudar a identificar que objetos y que características son potencialmente peligrosos, así como la posibilidad de generar alertas en función de parámetros críticos como la proximidad, la masa o la velocidad del asteroide.

El sistema implementa todo el stack tecnológico necesario. Teniendo una arquitectura basada en un clúster de Apache hadoop desplegado en un entorno distribuido con HDFS, que permite realizar la ingestión de datos con un cluster de kafka que se encargará de recibir los datos en tiempo real y almacenarlos en un topic. para su posterior procesamiento utilizando las APIs de Spark Streaming y Spark SQL.

Los datos origen han sido diseñados para incluir valores realistas como si recopilarlos de la imagen de un telescopio se tratase (como diámetro, brillo del objeto y ruido de la imagen(simulando la interferencia)), junto con datos erróneos o incompletos, con el fin de justificar y aplicar transformaciones durante la fase de ETL. La información que se pretende extraer de estos datos son métricas que nos ayudaran a extraer conocimiento como (superficie, velocidad, distancia, masa, etc.).

Mediante la integración de Prometheus y Grafana, se realizarán las correspondientes visualizaciones a través de los dashboards para facilitar la toma de decisiones y el análisis exploratorio de los resultados.

3. Arquitectura del sistema

Para el desarrollo de este proyecto se ha utilizado una arquitectura distribuida basada en un clúster de Apache Hadoop, que permite el procesamiento y almacenamiento de grandes volúmenes de datos. El clúster está compuesto por 4 máquinas virtuales, las cuales 1 de ellas actúa como nodo maestro y las otras 3 como nodos esclavos. El nodo maestro se encarga de gestionar el clúster, coordinando las tareas de procesamiento y almacenamiento, mientras que los nodos esclavos se encargan de ejecutar las tareas de procesamiento y almacenamiento de datos.

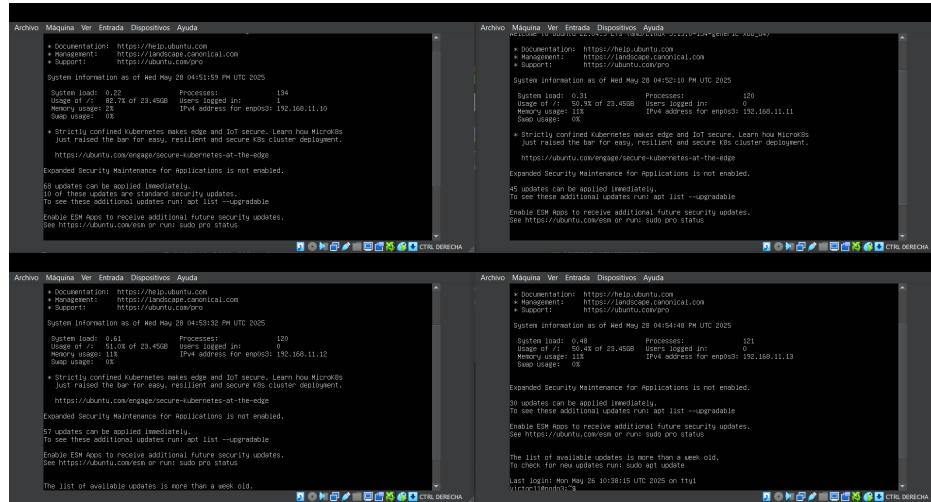


Figura 1: Maquinas virtuales del clúster

Figura 2: Controller y brokers

Siguiendo con la estructura, tenemos kafka que se encargará de la ingestión de datos en tiempo real, este constituido por un controlador y 2 brokers, actuando como un sistema de mensajería distribuido que permite la publicación y suscripción de mensajes en tiempo real. Los datos se envían a un topic específico que se encargará de recibir los datos y enviarlos al consumidor de Spark Streaming, que se encargará de procesar los datos en tiempo real. Spark lee los datos, los procesa recibiendo un job que lo divide en tareas y los distribuye a los workers para su procesamiento. Cada worker se encarga de procesar los datos y enviarlos a HDFS, donde se almacenarán los datos procesados en formato JSON.

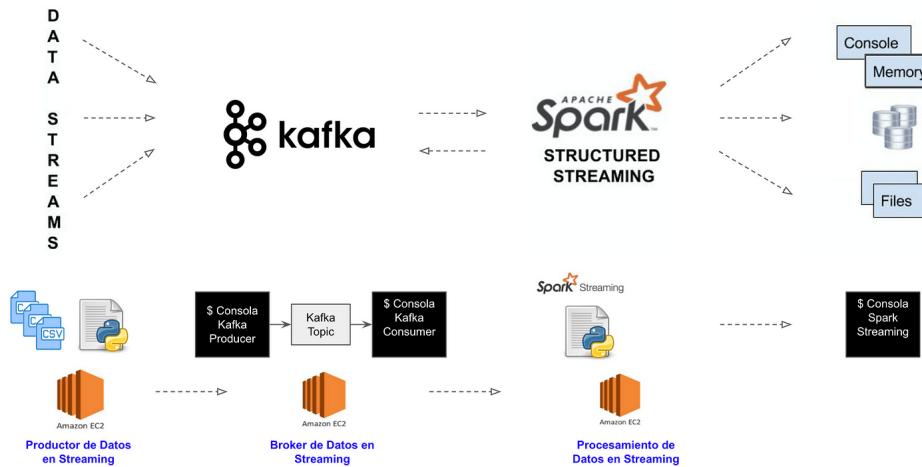


Figura 3: Estructura del sistema

4. Generación de datos sintéticos

Como he mencionado anteriormente, el conjunto de datos se genera de forma sintética para simular la ingestión de datos en tiempo real. Para ello, se ha utilizado la librería Faker [2, Faker] y [7, numpy] que permite generar datos sintéticos de forma aleatoria y realista. El script que se encargará de esto es `neo_generate_synthetic_data.py`, (el cual se encuentra dentro de la carpeta `src/scripts`) del repositorio. Dentro del script tenemos las respectivas funciones que se encargarán de generar los datos. Simulando la recolección de información de un telescopio (diametro, brillo del objeto y ruido de la imagen), además de la generación de coordenadas celestiales sobre estos objetos (horas y grados).

Finalmente tenemos la generación de un archivo pipe o FIFO que se encargará de recoger estos datos en formato JSON y enviarlos al productor de Kafka para simular la ingestión de datos en tiempo real de una API. El script `neo_data_producer.py` actuará como productor, recibiendo los datos del archivo pipe y enviándolos al topic de Kafka correspondiente.

5. Procesamiento de datos (ETL)

Para el procesamiento de datos se ha utilizado Apache Spark, que permite el procesamiento de grandes volúmenes de datos de forma distribuida. Teniendo 3 archivos que actuarán como workers, los cuales se encargarán de procesar los datos recibidos del topic de Kafka `neo_raw_data`. Para ello tenemos el script `spark_asteroidTrackingProcessing.py`, que se encargará de:

5.1. Consumidor

Utiliza Spark Structured Streaming para:

- Conectarse al broker Kafka (192.168.11.10:9094).
- Leer mensajes en tiempo real del tópico `neo_raw_data`.
- Interpretar cada mensaje como un JSON que contiene información observacional (timestamp, coordenadas, diámetro, brillo, ruido, etc.).

5.2. Transformación de datos

Una vez leídos los datos en bruto, se aplican transformaciones para calcular métricas físicas y orbitales de cada asteroide:

- Se le asigna un identificador único a cada asteroide (simulando como se identifican en la realidad, pero puesto que son sintéticos y necesitamos variedad se hará de forma aleatoria).
 - YYYY es el año de la observación
 - L1, L2 son letras aleatorias (excepto I y Z en algunas posiciones) (simulando el sistema de clasificación de asteroides por quincenas).
 - NN es un número aleatorio entre 01 y 99 (simulando el número de descubrimiento del objeto).

Este identificador sigue una simplificación de las reglas reales del Minor Planet Center.

Cada objeto generado tiene una aleatoriedad de replica que introduce variaciones en los datos para simular el seguimiento del objeto. Siguiendo a las metricas tenemos:

- Conversión de RA/DEC a coordenadas cartesianas tridimensionales (x, y, z) para averiguar la distancia del asteroide a la Tierra.

Asumiendo una distancia fija $d=1,000,000$ km al objeto, se transforma la posición desde coordenadas esféricas RA/DEC a cartesianas:

$$\begin{aligned} \text{RA}_{\text{rad}} &= \text{RA}_{\text{hours}} \times 15 \times \frac{\pi}{180} \\ \text{DEC}_{\text{rad}} &= \text{DEC}_{\text{deg}} \times \frac{\pi}{180} \\ x &= d \cdot \cos(\text{DEC}_{\text{rad}}) \cdot \cos(\text{RA}_{\text{rad}}) \\ y &= d \cdot \cos(\text{DEC}_{\text{rad}}) \cdot \sin(\text{RA}_{\text{rad}}) \\ z &= d \cdot \sin(\text{DEC}_{\text{rad}}) \end{aligned}$$

- Distancia mínima al origen (en km y UA)

$$\begin{aligned} \text{distancia}_{\text{km}} &= \sqrt{x^2 + y^2 + z^2} \\ \text{distancia}_{\text{AU}} &= \frac{\text{distancia}_{\text{km}}}{149,597,870,7} \end{aligned}$$

- Cálculo de propiedades físicas (A partir del diámetro se derivan múltiples propiedades físicas)
 - Radio:

$$r = \frac{d}{2}$$

- Volumen del asteroide (asumido esférico):

$$V = \frac{4}{3}\pi r^3$$

- Área superficial:

$$A = 4\pi r^2$$

- Densidad: valor fijo típico para roca: $p=2600 \text{ kg/m}^3$
- Masa:

$$m = V \cdot p$$

- Velocidad de escape:

$$v_{\text{escape}} = \sqrt{\frac{2Gm}{r}}$$

Donde:

- d: diámetro en metros
- r: radio
- $p=2600 \text{ kg/m}^3$
- $G = 6,67430 \times 10^{-11} \text{ m}^3\text{kg}^{-1}\text{s}^{-2}$
- Parámetros orbitales básicos
 - Eje semi-mayor (suponiendo órbita elíptica):

$$a = \text{distancia}_{\text{AU}} \cdot 149,597,870,700 \text{ m}$$

- Excentricidad: valor fijo: $e=0.3$
- Período orbital (en segundos y días):

$$T = 2\pi\sqrt{\frac{a^3}{Gm}}$$

$$T_{\text{días}} = \frac{T}{86400}$$

- Inclinación orbital: valor fijo: $i=10$ grados

- Cálculo de elementos orbitales avanzados

- Periapsis y apoapsis:

$$q = a \cdot (1 - e)$$

$$Q = a \cdot (1 + e)$$

- ¿Cruza la órbita terrestre? (Se compara si el perihelio es menor a 1.017 UA y el afelio mayor a 0.983 UA)

$$\text{Cruza órbita terrestre} \iff (q < 1,017 \text{ AU}) \wedge (Q > 0,983 \text{ AU})$$

- Factor de sincronía (para calcular la probabilidad de impacto):

$$\text{Factor de sincronía} = \frac{1}{\max(1, |T_{\text{días}} - 365,25|)}$$

- Probabilidad de impacto

Si el objeto cruza la órbita terrestre, la probabilidad de impacto se estima mediante una fórmula determinista que considera tamaño, distancia y sincronía temporal:

$$P_{\text{impacto}} = \left(\frac{\left(\frac{d}{1000}\right)^2}{\text{distancia}_{\text{AU}}^2 + 0,1} \right) \cdot \text{Factor de sincronía} \cdot 0,01$$

- Clasificación de nivel de amenaza (Según el valor de P impacto, se clasifica el riesgo)

- ALTO si $P > 0,05$
- MEDIO si $0,01 < P \leq 0,05$
- BAJO si $P \leq 0,01$

Como podemos ver son calculos para metricas físicas y orbitales que nos permitiran obtener conocimiento sobre cada objeto y el comportamiento de estos. Dando la posibilidad de realizar analisis y visualizaciones sobre estos datos.

Este ultimo tipo de calculo es algo un tanto artificial ya que en la practica real se utilizan otro tipo de calculos y algoritmos más complejos. Por lo que he utilizado calculos más sencillos y deterministas para llevar a cabo el proyecto. También si se revisa el script se puede ver que se han aumentado las probabilidades de impacto en ALTO Y MEDIO para tener variedad de datos y poder realizar pruebas de visualización y análisis.

Esta seria la estructura del JSON que se generará y se almacenará en HDFS:

```
[  
  {  
    "timestamp": "2025-05-30T19:06:57.422979",  
    "object_id": "(2025 MS99)",  
    "celestial_coords": {  
      "ra_hours": 2.341542682368541,  
      "dec_degrees": 64.98281334075511  
    },  
    "diameter_m": 9506.85,  
    "brightness_mag": 15.31,  
    "image_noise_level": 0.54,  
    "cartesian_coords_km": {  
      "x": 345889.182,  
      "y": 243303.745,  
      "z": 906180.976  
    },  
    "min_distance_km": 999999.9999099834,  
    "min_distance_au": 0.006684587121666722,  
    "density_kg_m3": 2600,  
    "volume_m3": 4.49892286525126E11,  
    "surface_area_m2": 2.8393776268172485E8,  
    "mass_kg": 1.1697199449653275E15,  
    "escape_velocity_m_s": 5.7313309126899155,  
    "semi_major_axis_m": 9.99999999099835E8,  
    "eccentricity": 0.3,  
    "orbital_period_days": 8230426.036019449,  
    "inclination_degrees": 10.0,  
    "periapsis_m": 6.999999994E8,  
    "apoapsis_m": 1.2999999988E9,  
    "orbital_period_years": 22533.68,  
    "impact_probability": 0.0,  
    "threat_level": "BAJO"  
  }]
```

Figura 4: Estructura JSON de los datos enriquecidos

5.3. Almacenamiento en HDFS

Los datos enriquecidos se escriben directamente en HDFS, usando el formato JSON (en formato JSON para dar más versatilidad a los datos). La escritura está configurada para:

- Guardar en la ruta: `hdfs://cluster-bda:9000/bda/kafka/AsteroidTracking`
- Organizar los datos en carpetas por:
 - Tópico (topic=asteroid-events)
 - Partición lógica Spark (partition=X)
 - Fecha (date=YYYY-MM-DD)

El uso de checkpointLocation garantiza tolerancia a fallos y recuperación del estado del stream.

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	victor11	supergroup	0 B	May 30 21:07	0	0 B	_spark_metadata
drwxr-xr-x	victor11	supergroup	0 B	May 30 21:06	0	0 B	checkpoints
drwxr-xr-x	victor11	supergroup	0 B	May 30 21:07	0	0 B	topic=asteroid-events

Figura 5: Controller y brokers

6. Visualización

Para la visualización de los datos como he mencionado anteriormente, se implementó una arquitectura de monitorización y visualización utilizando Prometheus y Grafana, con el objetivo de supervisar tanto las métricas técnicas del sistema como los indicadores científicos derivados del procesamiento de los asteroides.

6.1. Prometheus + JMX Exporter

Prometheus fue configurado como sistema de monitorización principal. A través del JMX Exporter, se habilitó la recolección de métricas JVM en los servicios Spark y Kafka, permitiendo supervisar su rendimiento en tiempo real: uso de CPU, memoria, latencia de procesamiento y throughput.

6.2. Exportador personalizado

Además de las métricas del sistema, hay un exportador personalizado `asteroid_exporter.py` con el objetivo de exponer métricas científicas enriquecidas sobre los asteroides procesados por Spark. Este exportador se encarga de conectarse a HDFS a través de WebHDFS para acceder directamente a los archivos JSON enriquecidos generados por Spark Streaming. Luego recorre todas las particiones y fechas disponibles para procesar los datos sin necesidad de descargarlos previamente. Finalmente expone las métricas en formato Prometheus en el puerto :9091, tanto agregadas como individuales que son recogidas por Prometheus por un job definido en `prometheus_asteroidTracking_mon_kafka.yml`

6.3. Dashboards de Grafana

Para visualizar las métricas generadas por el exportador, se diseñó un dashboard en Grafana `asteroid_dashboard.json` que ofrece una visión clara, interactiva y dinámica del estado actual de los asteroides detectados. El dashboard contiene los siguientes paneles:

- Número de asteroides procesados: gráfico de líneas que muestra la evolución del número de asteroides procesados en tiempo real.
- Distribución de amenaza: gráfico tipo donut que muestra el porcentaje de objetos clasificados como ALTO, MEDIO o BAJO. Incluye enlaces interactivos que permiten filtrar el resto de paneles por nivel de amenaza.
- Top 50 asteroides por probabilidad de impacto: tabla ordenada por la métrica `asteroid_impact_probability`, útil para identificar rápidamente los objetos más peligrosos.
- Tabla unilineal del asteroide seleccionado: muestra información detallada.
- Comparativa de masa y densidad: gráfico de barras que muestra la masa o densidad a lo largo del tiempo.
- Evolución de la distancia mínima (AU): gráfico de líneas que muestra la distancia mínima estimada para cada asteroide con posibilidad de impacto, permitiendo observar su evolución en tiempo real.

Además, el dashboard incluye variables globales para filtrar la visualización por:

- Nivel de amenaza (threatVar): todos, ALTO, MEDIO, BAJO.
- Período orbital máximo (periodMax): filtra los asteroides cuyo período orbital está por debajo de un umbral definido en días.
- Selección de asteroide (asteroidVar): permite seleccionar un asteroide específico para ver su información detallada en la tabla unilineal.

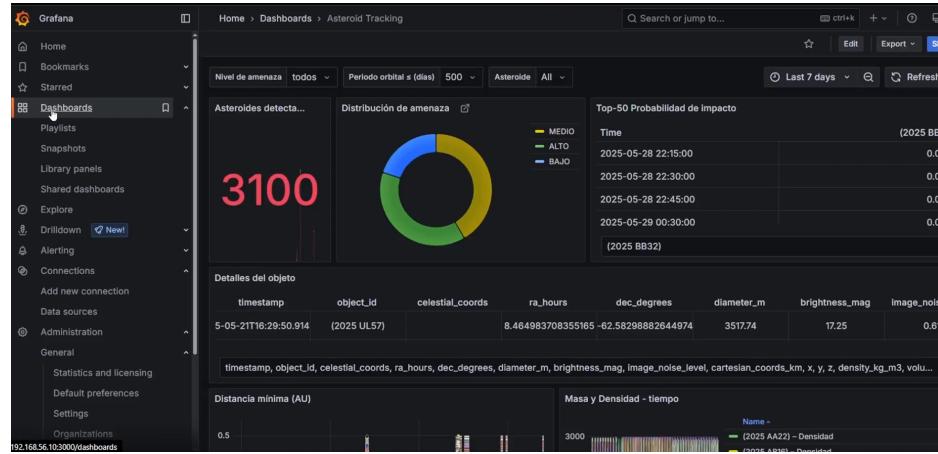


Figura 6: Dashboard

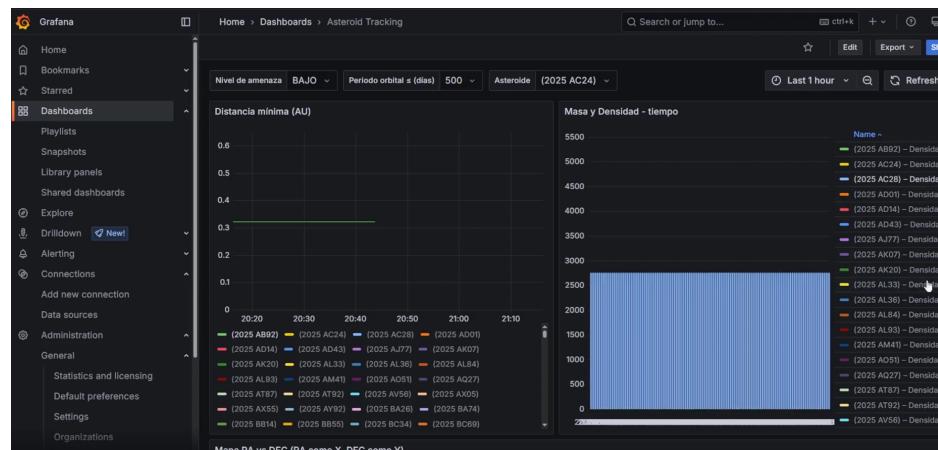


Figura 7: Dashboard

7. Business Intelligence

Gracias al conjunto de datos enriquecidos y las métricas generadas, se pueden realizar análisis exploratorios para descubrir patrones y tendencias en los objetos cercanos a la Tierra, tanto para aplicaciones científicas como operativas.

7.1. Detección y evaluación de riesgo en tiempo real

- Se identifican asteroides cercanos.
- Se calcula su probabilidad de impacto usando una fórmula determinista basada en distancia mínima, diámetro, sincronía orbital y cruce con la órbita terrestre.
- Se clasifican en niveles de amenaza (ALTO, MEDIO, BAJO) para priorizar la atención.
- Distribución histórica de amenazas por nivel.
- Visualización de la evolución del objeto a lo largo del tiempo.
- Visualización de metricas físicas y orbitales de los asteroides para determinar su comportamiento y características.
- Cuáles podrían requerir atención técnica o monitoreo adicional.
- Qué asteroides tienen mayor probabilidad de retorno cercano en el futuro.

7.2. Simulación de escenarios y entrenamiento de modelos

- Entrenar modelos de predicción de impacto.
- Validar algoritmos de clasificación automática.
- Generar datasets para entrenar modelos de machine learning.
- Simular escenarios de impacto y evaluar consecuencias.

7.3. Investigación científica

- Estudiar la composición y estructura de los asteroides.
- Analizar la evolución de sus órbitas a lo largo del tiempo.
- Investigar la formación y dinámica del sistema solar.
- Comparar propiedades físicas con otros cuerpos celestes.
- Identificar patrones en la distribución espacial de los NEOs.

8. Conclusión

AsteroidTracking es un sistema completo de Big Data que permite la detección y análisis en tiempo real de objetos cercanos a la Tierra, utilizando un conjunto de sintéticos basados en observaciones parecidas a las reales simulando una ingesta de datos en tiempo real. El sistema implementa todo el stack tecnológico necesario, desde la ingestión de datos con Kafka, el procesamiento de datos con Spark, hasta la visualización de los resultados con Grafana.

En el repositorio de Github se puede encontrar el código fuente utilizado para cada script del proyecto y su configuración del cluster. Así como un video demostrativo del funcionamiento del sistema.

9. Webgrafía

[4] [5] [2] [7] [9] [3] [8] [1] [11] [6] [10]

Referencias

- [1] Grafana. *Grafana*. Fecha de consulta: 18 de mayo de 2025. 2025. URL: <https://grafana.com/>.
- [2] joke2k. *Faker*. Fecha de consulta: 29 de abril de 2025. 2025. URL: <https://github.com/joke2k/faker>.
- [3] Apache Kafka. *Apache Kafka*. Fecha de consulta: 30 de abril de 2025. 2025. URL: <https://kafka.apache.org/>.
- [4] NASA. *asteroids-neows-api*. Fecha de consulta: 28 de abril de 2025. 2025. URL: <https://data.nasa.gov/dataset/asteroids-neows-api>.
- [5] NASA. *Sentry System*. Fecha de consulta: 28 de abril de 2025. 2025. URL: <https://cneos.jpl.nasa.gov/sentry/>.
- [6] nasa. *Como Se Calcula El Movimiento Orbital*. Fecha de consulta: 22 de mayo de 2025. 2025. URL: <https://pwg.gsfc.nasa.gov/stargaze/Mmotion.htm>.
- [7] Numpy. *Numpy*. Fecha de consulta: 29 de abril de 2025. 2025. URL: <https://numpy.org/>.
- [8] Prometheus. *Prometheus*. Fecha de consulta: 18 de mayo de 2025. 2025. URL: <https://prometheus.io/>.
- [9] Apache Spark. *Apache Spark*. Fecha de consulta: 30 de abril de 2025. 2025. URL: <https://spark.apache.org/>.
- [10] wikipedia. *Impact event*. Fecha de consulta: 22 de mayo de 2025. 2025. URL: https://en.wikipedia.org/wiki/Impact_event.
- [11] wikipedia. *Standard asteroid physical characteristics*. Fecha de consulta: 22 de mayo de 2025. 2025. URL: https://en.wikipedia.org/wiki/Standard_asteroid_physical_characteristics#:~:text=Krasinsky%20et%20al.,and%205.32%20g%2Fcm3..