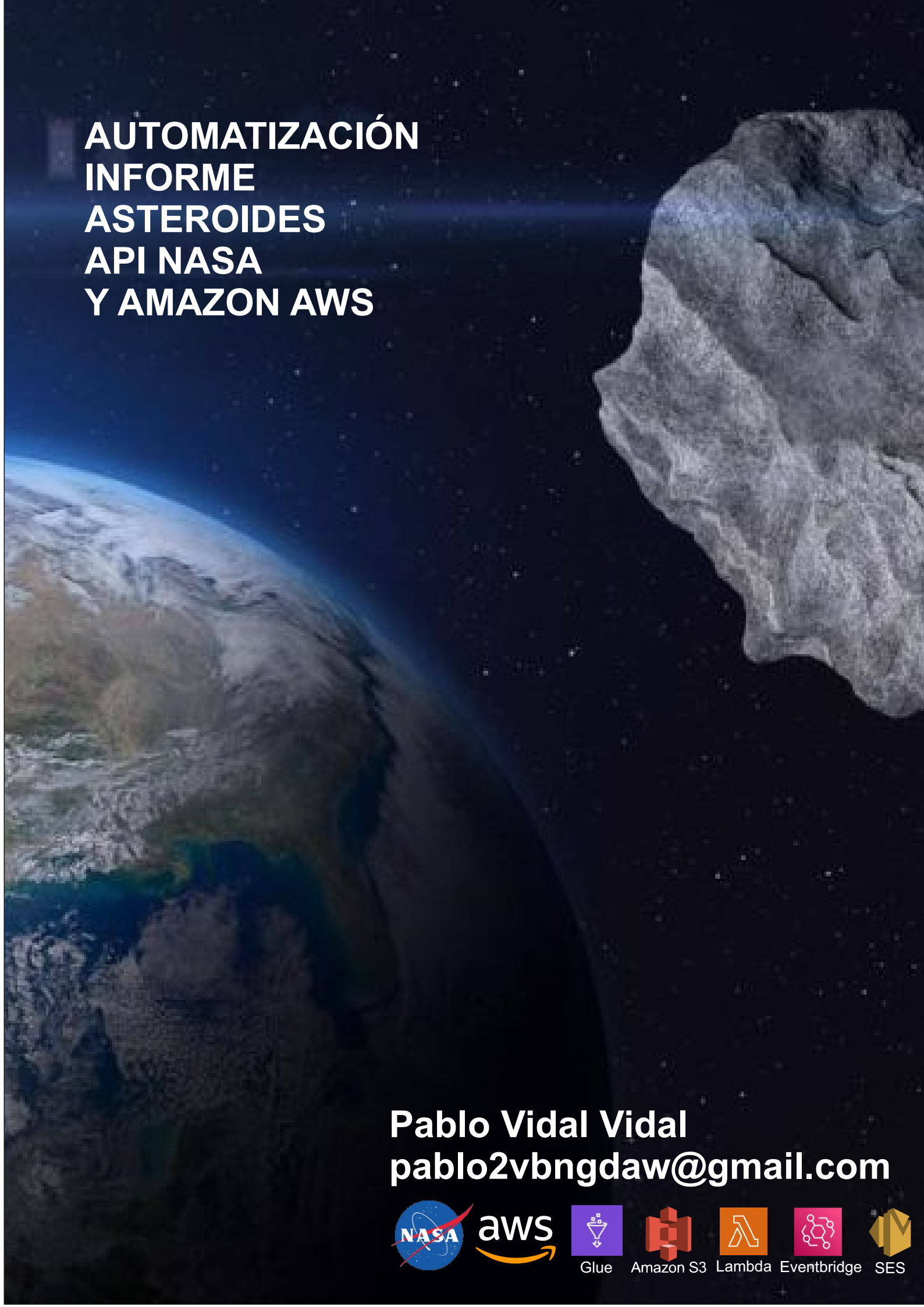
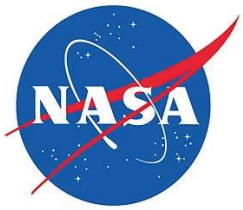

AUTOMATIZACIÓN
INFORME
ASTEROIDES
API NASA
Y AMAZON AWS
Pablo Vidal Vidal
pablo2vbngdaw@gmail.com
NASA
aws
Glue
Amazon S3
Lambda
Eventbridge
SES

📑 Índice General

🛰️ Reto 2 – Automatización del Análisis de Asteroides con AWS y la API de la



Autor: Pablo Vidal Vidal

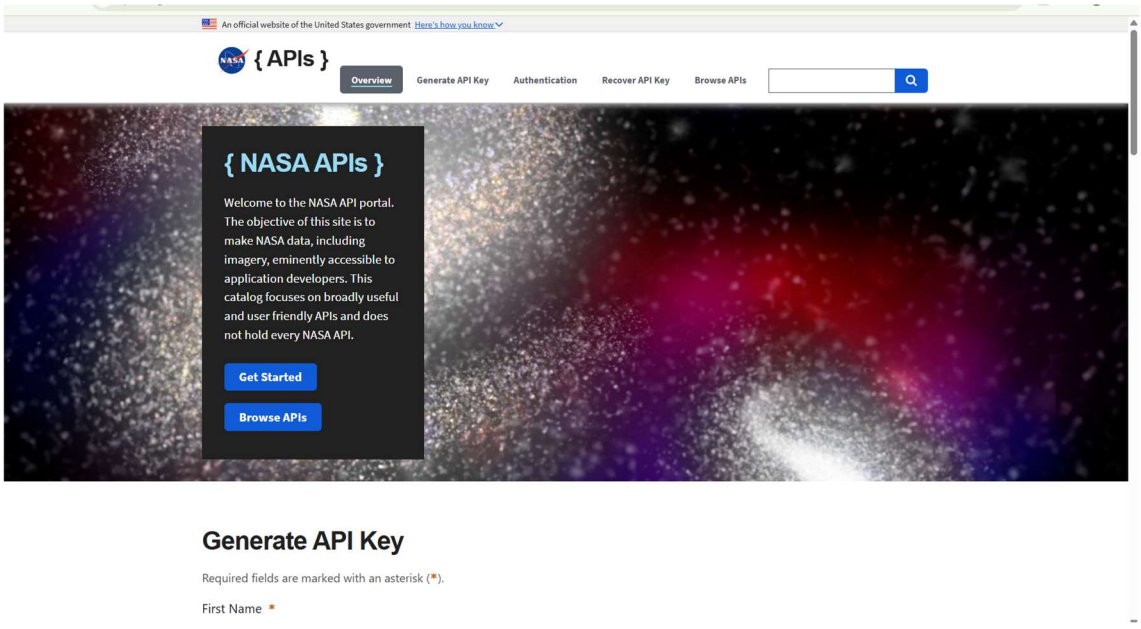
🎯 Objetivo del proyecto

El objetivo principal de este reto es automatizar el proceso de obtención, almacenamiento, transformación y análisis de datos provenientes de la API de la NASA sobre asteroides cercanos a la Tierra. A través del uso de servicios de AWS como Lambda, S3, Glue, y EventBridge, se construirá un flujo automatizado que identifique, clasifique y reporte los asteroides potencialmente más peligrosos para nuestro planeta.

🧩 Desarrollo del Proyecto: Paso a Paso

📌 Paso 1: Registro en la API de la NASA

Se inicia el proyecto registrándose en la plataforma de la NASA para obtener una clave de API gratuita:



API Key: 8AiSKeDnkA3vXFt8Jy6V12U32ZyfSTvE5bTF87lp

📌 Paso 2: Creación de la función Lambda GetDatosAsteroidesNasa

Esta función consulta diariamente la API de la NASA y guarda los datos obtenidos en un bucket de Amazon S3.

📌 Paso 3: Creación del Bucket de Entrada

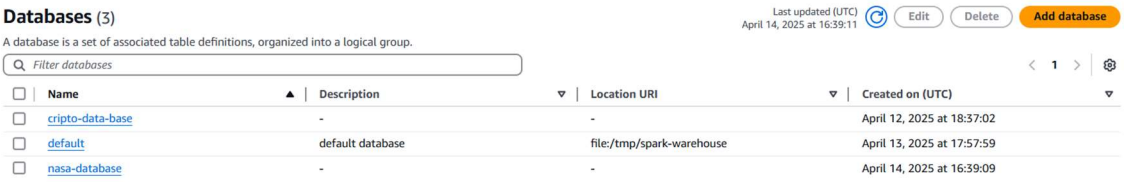
Se crea un bucket S3 para almacenar los datos crudos obtenidos:

Nombre del bucket: bucket-asteroides-pablobng

📌 Paso 4: Creación de la Base de Datos Glue

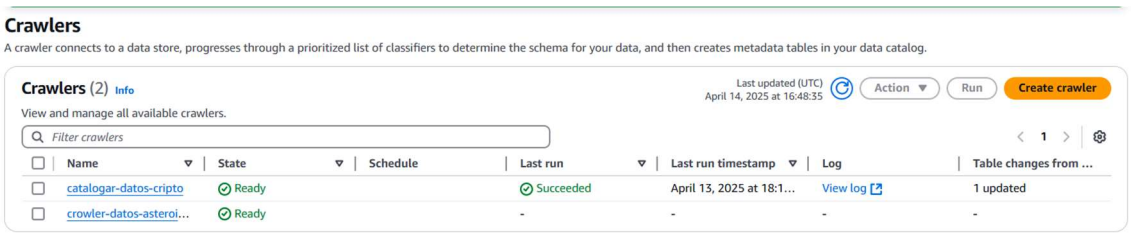
Se configura una base de datos en AWS Glue para alojar los datos catalogados:

Nombre: nasa-database



📌 Paso 5: Creación del Crawler

Se configura un crawler que analiza los datos almacenados en S3 y los cataloga en la base de datos Glue:



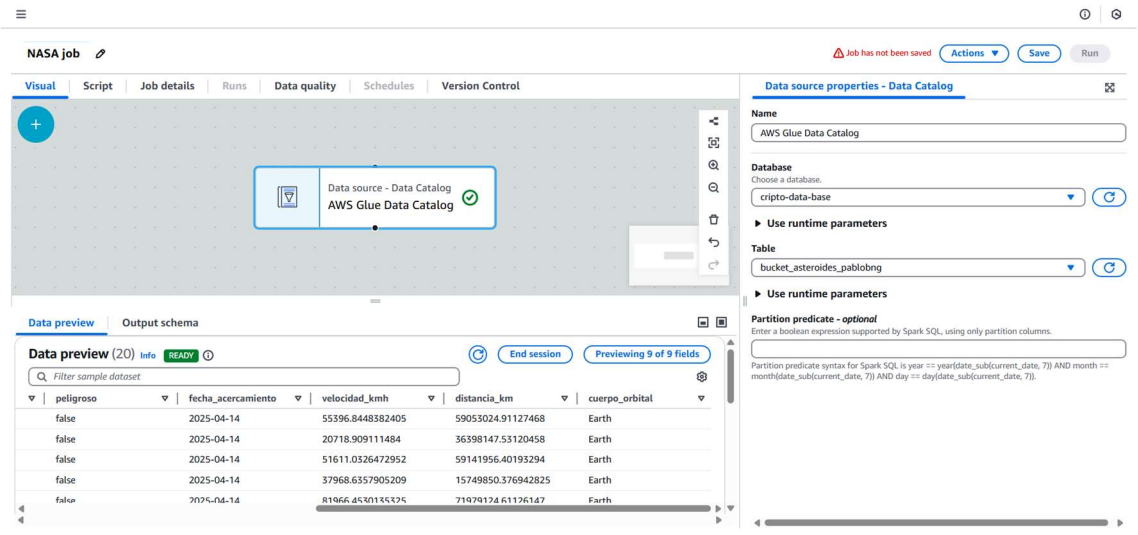
Nombre del crawler: crawler-datos-asteroides

📌 Paso 6: Creación del Job de Glue

Se configura un Glue Job llamado:

Nombre del job: Nasa Job

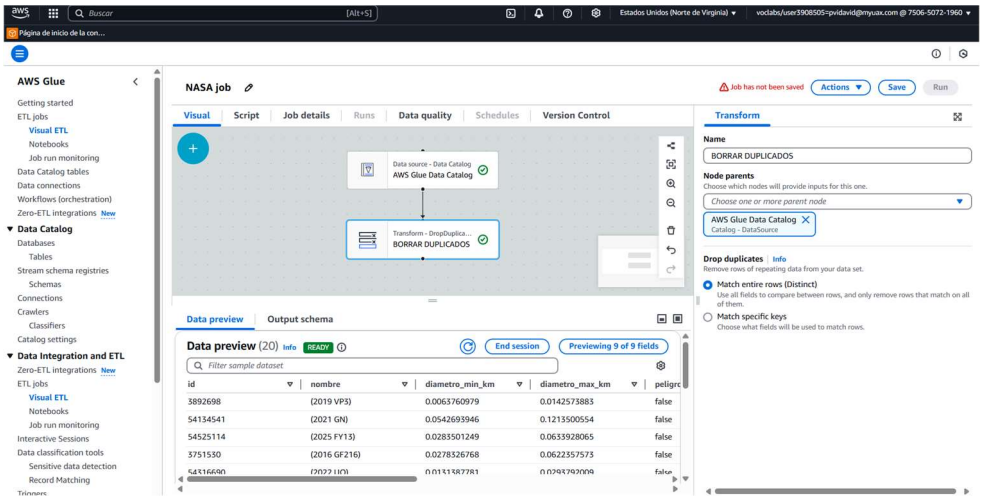
Este job accede a los datos catalogados y realiza transformaciones para preparar la información para análisis.



🔄 Transformaciones de Datos

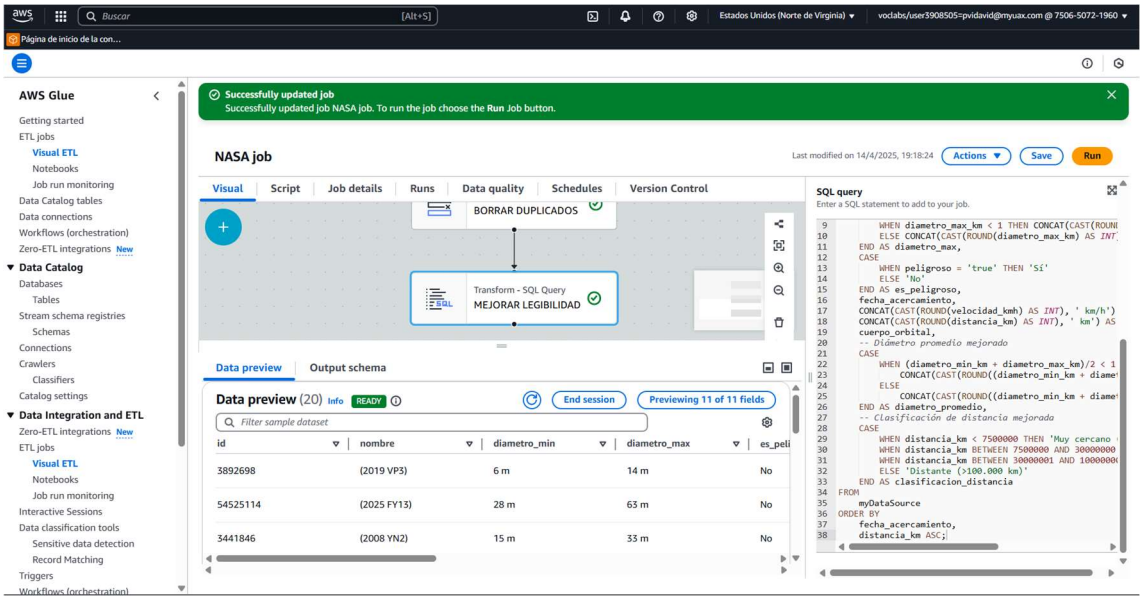
📌 Paso 7: Eliminación de Duplicados

Se eliminan filas duplicadas completamente idénticas, ya que los datos se actualizan diariamente a las 08:00.



📌 Paso 8: Simplificación para Lectura

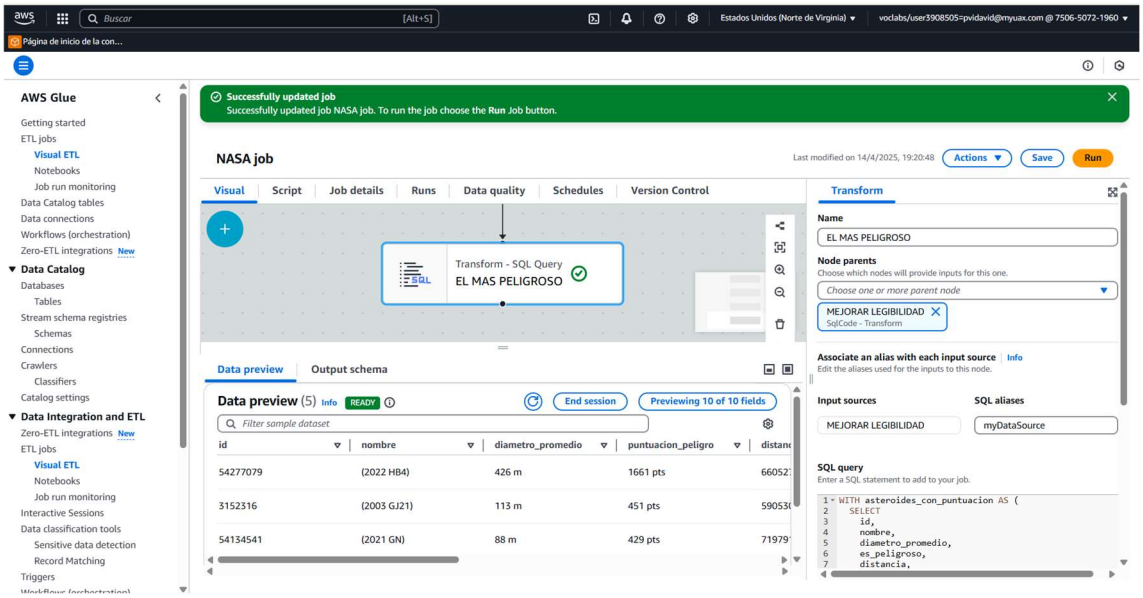
Se reformatean los datos para que sean más fáciles de leer y comprender por humanos.



📌 Paso 9: Cálculo del Asteroide Más Peligroso

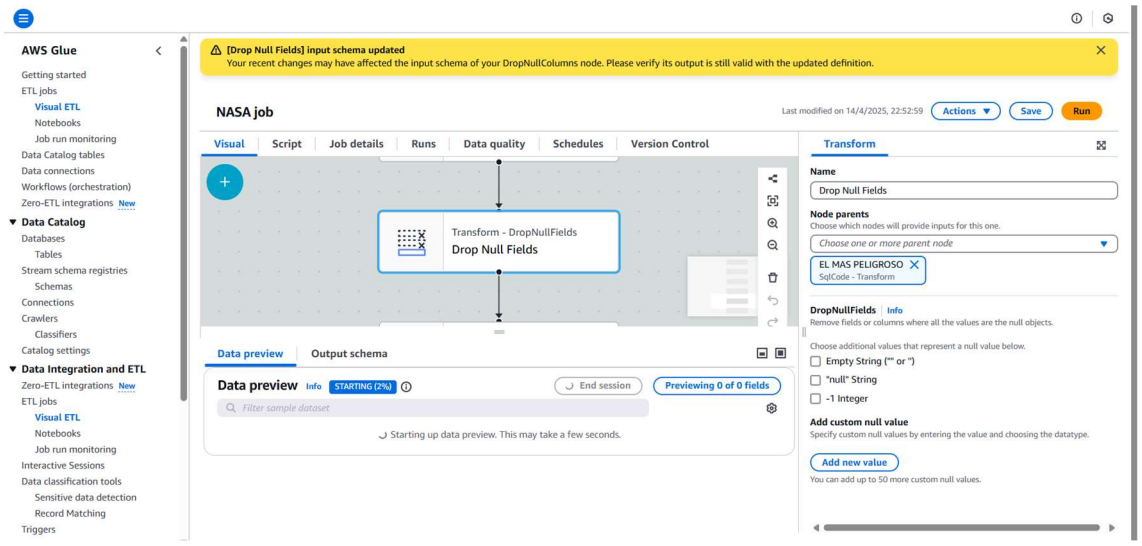
Se aplica una fórmula de puntuación para identificar el asteroide más peligroso considerando:

- **Peligrosidad** (100 puntos base si es marcado como tal)
- **Tamaño** (1 punto por cada 10 m de diámetro promedio)
- **Distancia de acercamiento**:
  - Muy cercano: 50 puntos
  - Cercano: 30 puntos
  - Medio: 10 puntos
- **Velocidad relativa** (1 punto por cada 1000 km/h)



📌 Paso 10: DROP NULL FIELDS

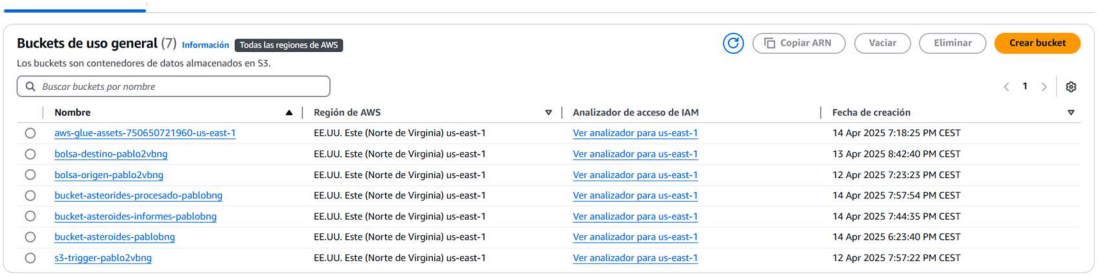
Hemos añadido este campo intermedio porque el job no se ejecutaba por tener algún camp nulo



📌 Paso 11: Bucket de Salida

Se crea un nuevo bucket S3 para almacenar los datos procesadoa:

Nombre del bucket: bucket-asteroides-procesado-pablobng



Buckets de uso general (7) Información — Todas las regiones de AWS

Los buckets son contenedores de datos almacenados en S3.

Copiar ARN | Vaciar | Eliminar | Crear bucket

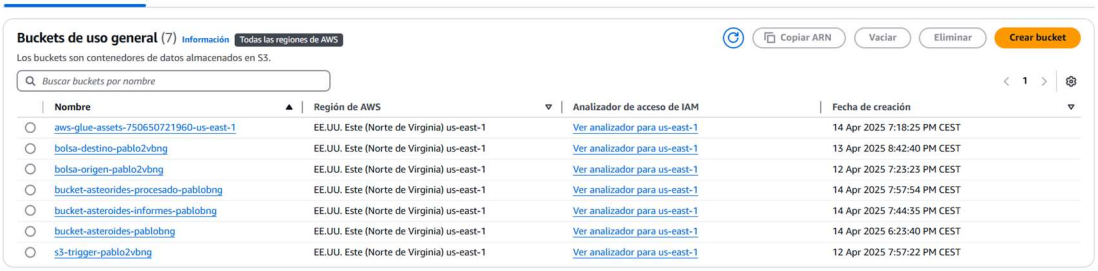
Buscar buckets por nombre

Nombre	Región de AWS	Analizador de acceso de IAM	Fecha de creación
aws-glue-assets-750650721960-us-east-1	EE.UU. Este (Norte de Virginia) us-east-1	Ver analizador para us-east-1	14 Apr 2025 7:18:25 PM CEST
bolsa-destino-pablo2vbng	EE.UU. Este (Norte de Virginia) us-east-1	Ver analizador para us-east-1	13 Apr 2025 8:42:40 PM CEST
bolsa-origen-pablo2vbng	EE.UU. Este (Norte de Virginia) us-east-1	Ver analizador para us-east-1	12 Apr 2025 7:23:23 PM CEST
bucket-asteorides-procesado-pablobng	EE.UU. Este (Norte de Virginia) us-east-1	Ver analizador para us-east-1	14 Apr 2025 7:57:54 PM CEST
bucket-asteroides-informes-pablobng	EE.UU. Este (Norte de Virginia) us-east-1	Ver analizador para us-east-1	14 Apr 2025 7:44:35 PM CEST
bucket-asteroides-pablobng	EE.UU. Este (Norte de Virginia) us-east-1	Ver analizador para us-east-1	14 Apr 2025 6:23:40 PM CEST
s3-trigger-pablo2vbng	EE.UU. Este (Norte de Virginia) us-east-1	Ver analizador para us-east-1	12 Apr 2025 7:57:22 PM CEST

📌 Paso 12: Bucket de Salida Informe

Se crea un nuevo bucket S3 para almacenar los datos procesadoa:

Nombre del bucket: bucket-asteroides-informes-pablobng



Buckets de uso general (7) Información — Todas las regiones de AWS

Los buckets son contenedores de datos almacenados en S3.

Copiar ARN | Vaciar | Eliminar | Crear bucket

Buscar buckets por nombre

Nombre	Región de AWS	Analizador de acceso de IAM	Fecha de creación
aws-glue-assets-750650721960-us-east-1	EE.UU. Este (Norte de Virginia) us-east-1	Ver analizador para us-east-1	14 Apr 2025 7:18:25 PM CEST
bolsa-destino-pablo2vbng	EE.UU. Este (Norte de Virginia) us-east-1	Ver analizador para us-east-1	13 Apr 2025 8:42:40 PM CEST
bolsa-origen-pablo2vbng	EE.UU. Este (Norte de Virginia) us-east-1	Ver analizador para us-east-1	12 Apr 2025 7:23:23 PM CEST
bucket-asteorides-procesado-pablobng	EE.UU. Este (Norte de Virginia) us-east-1	Ver analizador para us-east-1	14 Apr 2025 7:57:54 PM CEST
bucket-asteroides-informes-pablobng	EE.UU. Este (Norte de Virginia) us-east-1	Ver analizador para us-east-1	14 Apr 2025 7:44:35 PM CEST
bucket-asteroides-pablobng	EE.UU. Este (Norte de Virginia) us-east-1	Ver analizador para us-east-1	14 Apr 2025 6:23:40 PM CEST
s3-trigger-pablo2vbng	EE.UU. Este (Norte de Virginia) us-east-1	Ver analizador para us-east-1	12 Apr 2025 7:57:22 PM CEST

📌 Paso 12: Ejemplo de informe salida

Top 5 Asteroides Más Letales del Sistema Solar

⚠️ **Advertencia:** Algunos de estos cuerpos celestes están clasificados como *amenazas de extinción masiva*. Pero tranquilos, la última vez solo fue un susto... ¿verdad? 😅

#	Nombre	Diámetro	Velocidad	Distancia	Peligroso	Nivel de Amenaza
1	(2022 HB4)	426 m	190513 km/h	66052706 km	No	Clase DIOS 🕊️🙏
2	(2003 GJ21)	113 m	55397 km/h	59053025 km	No	Clase DEMONIO 😈💣
3	(2021 WT1)	99 m	48588 km/h	15645293 km	No	Clase DEMONIO 😈💣
4	(2021 GN)	88 m	81966 km/h	71979125 km	No	Clase DRAGÓN 🐉🔥
5	(2025 DC36)	74 m	17733 km/h	5430738 km	No	Clase TIGRE 🐯⚠️

🔥 Si alguno de estos impacta la Tierra... ¡bueno, fue un gusto conocerlos! 🌍

Informe generado por **Pablo Vidal Vidal** | 2025-04-15 07:00:29

💥 Fin del mundo patrocinado por **N.A.S.A. (No Aseguramos Supervivencia Astral)**

⚙️ Automatización del Proceso

📌 Paso 13: Funciones Lambda + EventBridge

Función Lambda	Descripción	Trigger (EventBridge)	Hora
GetDatosAsteroidesNasa	Consulta la API de la NASA y guarda datos en S3	TriggerAsteroide	08:00
funcionDisparadorCrawler	Ejecuta el crawler	TriggerCrawler	08:30
funcionDisparadorJOB	Ejecuta el Glue Job	TriggerJOB	08:45
Informe-asteroide-nasa	Genera y guarda el informe final	TriggerInforme	09:00

✅ Conclusión

Este proyecto demuestra cómo se puede construir un flujo de trabajo completamente automatizado para obtener y analizar datos científicos en la nube. Gracias a servicios como AWS Lambda, Glue y S3, y a la utilización de la API de la NASA, se ha desarrollado una solución escalable que permite identificar asteroides potencialmente peligrosos de manera diaria, eficiente y estructurada.

✅ Posibles mejoras

Automatización de correos electrónicos con el informe de salida .

✅ CÓDIGOS FUNCIONES LAMBDA

📌 GetDatosAsteroidesNasa

```
import json
import urllib.request
import csv
import boto3
from datetime import datetime
import os

def lambda_handler(event, context):
    # Configuración
    api_key = '8AiSKeDnkA3vXFt8Jy6V12U32ZyfSTvE5bTF87lp'
    s3_bucket = 'bucket-asteroides-pablobng'

    # Obtener fecha actual para el nombre del archivo
    current_date = datetime.now().strftime('%Y-%m-%d')
    filename = f'asteroides_{current_date}.csv'

    try:
        # 1. Obtener datos de la API de la NASA
        url = f'https://api.nasa.gov/neo/rest/v1/feed?start_date={current_date}&end_date={current_date}&api_key={api_key}'
        response = urllib.request.urlopen(url)
        data = json.loads(response.read())
```

```
    # 2. Procesar los datos y convertirlos a CSV
    asteroids = []

    # La API devuelve los asteroides por fecha, necesitamos extraerlos todos
    for date in data['near_earth_objects']:
      for asteroid in data['near_earth_objects'][date]:
        # Extraer información relevante
        asteroid_data = {
          'id': asteroid['id'],
          'name': asteroid['name'],
          'estimated_diameter_min_km':
asteroid['estimated_diameter']['kilometers']['estimated_diameter_min'],
          'estimated_diameter_max_km':
asteroid['estimated_diameter']['kilometers']['estimated_diameter_max'],
          'is_potentially_hazardous': asteroid['is_potentially_hazardous_asteroid'],
          'close_approach_date':
asteroid['close_approach_data'][0]['close_approach_date'],
          'relative_velocity_kmh':
asteroid['close_approach_data'][0]['relative_velocity']['kilometers_per_hour'],
          'miss_distance_km':
asteroid['close_approach_data'][0]['miss_distance']['kilometers'],
          'orbiting_body': asteroid['close_approach_data'][0]['orbiting_body']
        }
        asteroids.append(asteroid_data)

    # 3. Convertir a CSV
    csv_data = []
    headers = asteroids[0].keys() if asteroids else []
    csv_data.append(headers)
```

```
    for asteroid in asteroids:
        csv_data.append([str(asteroid[key]) for key in headers])

    # 4. Guardar en S3
    s3 = boto3.client('s3')
    csv_file = '\n'.join([','.join(row) for row in csv_data])

    s3.put_object(
        Bucket=s3_bucket,
        Key=filename,
        Body=csv_file,
        ContentType='text/csv'
    )

    return {
        'statusCode': 200,
        'body': f'Datos de asteroides guardados en s3://{s3_bucket}/{filename}'
    }

except Exception as e:
    return {
        'statusCode': 500,
        'body': f'Error: {str(e)}'
    }
```

📌 **FunciónDisparadorCrawler**

```
import boto3
import logging

# Configurar logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    # Nombre de tu crawler (asegúrate que coincida exactamente)
    CRAWLER_NAME = "crowler-datos-asteroides"

    # Cliente de AWS Glue
    glue_client = boto3.client('glue')

    try:
        # 1. Verificar estado actual del crawler
        response = glue_client.get_crawler(Name=CRAWLER_NAME)
        crawler_state = response['Crawler']['State']

        # 2. Solo iniciar si no está ya en ejecución
        if crawler_state != 'RUNNING':
            logger.info(f"Iniciando crawler: {CRAWLER_NAME}")
            start_response = glue_client.start_crawler(Name=CRAWLER_NAME)
            logger.info(f"Crawler iniciado. Respuesta: {start_response}")

            return {
                'statusCode': 200,
```

```
            'body': f'Crawler {CRAWLER_NAME} iniciado correctamente'
        }
    else:
        logger.warning(f"El crawler {CRAWLER_NAME} ya está en ejecución")
        return {
            'statusCode': 200,
            'body': f'Crawler {CRAWLER_NAME} ya estaba en ejecución'
        }

except Exception as e:
    logger.error(f"Error al iniciar crawler: {str(e)}")
    return {
        'statusCode': 500,
        'body': f'Error al iniciar crawler: {str(e)}'
    }
```

📌 **FunciónDisparadorCrawler**

```
import boto3
import logging

# Configurar logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    # Nombre de tu crawler (asegúrate que coincida exactamente)
    CRAWLER_NAME = "crowler-datos-asteroides"

    # Cliente de AWS Glue
    glue_client = boto3.client('glue')

    try:
        # 1. Verificar estado actual del crawler
        response = glue_client.get_crawler(Name=CRAWLER_NAME)
        crawler_state = response['Crawler']['State']

        # 2. Solo iniciar si no está ya en ejecución
        if crawler_state != 'RUNNING':
            logger.info(f"Iniciando crawler: {CRAWLER_NAME}")
            start_response = glue_client.start_crawler(Name=CRAWLER_NAME)
            logger.info(f"Crawler iniciado. Respuesta: {start_response}")

            return {
                'statusCode': 200,
```

```
            'body': f'Crawler {CRAWLER_NAME} iniciado correctamente'
        }
    else:
        logger.warning(f"El crawler {CRAWLER_NAME} ya está en ejecución")
        return {
            'statusCode': 200,
            'body': f'Crawler {CRAWLER_NAME} ya estaba en ejecución'
        }

except Exception as e:
    logger.error(f"Error al iniciar crawler: {str(e)}")
    return {
        'statusCode': 500,
        'body': f'Error al iniciar crawler: {str(e)}'
    }
```

📌 **FunciónDisparadorJob**

```
import boto3
import logging
import time
from datetime import datetime

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    JOB_NAME = "NASA job"
    BUCKET_PROCESADO = "bucket-asteroides-procesado-pablobng"
    PREFIX_S3_OUTPUT = "output/"  # Cambia esto si tu Glue job escribe en otro lugar
    MAX_WAIT_MINUTES = 14

    try:
        glue = boto3.client('glue')
        s3 = boto3.client('s3')

        logger.info(f"Iniciando Glue Job: {JOB_NAME}")
        job_run_id = glue.start_job_run(JobName=JOB_NAME)['JobRunId']
        logger.info(f"Job iniciado con ID: {job_run_id}")

        final_status = monitor_glue_job(glue, JOB_NAME, job_run_id, MAX_WAIT_MINUTES)
        if final_status != 'SUCCEEDED':
            raise Exception(f"Glue Job terminó con estado: {final_status}")
```

```
        logger.info("Verificando resultados en S3...")
        if not verify_s3_output(s3, BUCKET_PROCESADO, PREFIX_S3_OUTPUT):
            raise Exception(f"No se encontraron archivos en
s3://{BUCKET_PROCESADO}/{PREFIX_S3_OUTPUT}")

        success_msg = f"Proceso completado correctamente. Archivos en
s3://{BUCKET_PROCESADO}/{PREFIX_S3_OUTPUT}"
        logger.info(success_msg)
        return {
            'statusCode': 200,
            'body': success_msg,
            'jobRunId': job_run_id
        }

    except Exception as e:
        error_msg = f"Error en la ejecución: {str(e)}"
        logger.error(error_msg)
        return {
            'statusCode': 500,
            'body': error_msg
        }

def monitor_glue_job(glue_client, job_name, run_id, max_wait_minutes):
    start_time = time.time()
    max_seconds = max_wait_minutes * 60
    last_status = None
```

```
    while (time.time() - start_time) < max_seconds:
        response = glue_client.get_job_run(JobName=job_name, RunId=run_id)
        current_status = response['JobRun']['JobRunState']

        if current_status != last_status:
            logger.info(f"Estado actual del job: {current_status}")
            last_status = current_status

        if current_status in ['SUCCEEDED', 'FAILED', 'STOPPED', 'TIMEOUT']:
            return current_status

        time.sleep(20)

    return 'TIMEOUT'

def verify_s3_output(s3_client, bucket_name, prefix):
    max_attempts = 5
    for attempt in range(max_attempts):
        try:
            response = s3_client.list_objects_v2(Bucket=bucket_name, Prefix=prefix)
            contents = response.get('Contents', [])
            csv_files = [obj for obj in contents if obj['Key'].endswith('.csv')]

            if csv_files:
                logger.info(f"Se encontraron {len(csv_files)} archivo(s) CSV. Ejemplo:
{csv_files[0]['Key']}")
                return True
            else:
```

```
            logger.info(f"Intento {attempt+1}: No se encontraron archivos CSV aún.
Reintentando...")
            time.sleep(10)

        except Exception as e:
            logger.warning(f"Error al verificar archivos en S3 (intento {attempt+1}):
{str(e)}")
            time.sleep(5)

    return False
```

📌 **InformeAsteroideNasa**

```
import boto3
import csv
from datetime import datetime
from io import StringIO

def lambda_handler(event, context):
    BUCKET_DATOS = "bucket-asteorides-procesado-pablobng"
    BUCKET_INFORMES = "bucket-asteroides-informes-pablobng"

    s3 = boto3.client('s3')

    try:
        objetos = s3.list_objects_v2(Bucket=BUCKET_DATOS).get("Contents", [])
        archivos_csv = []

        for obj in objetos:
            key = obj['Key']
            if key.endswith('.csv'):
                archivos_csv.append(obj)
            elif '.' not in key.split('/')[-1]:
                obj['Key'] += '.csv'
                archivos_csv.append(obj)

        if not archivos_csv:
            raise Exception("No se encontraron archivos CSV en el bucket.")
```

```
    archivo_mas_reciente = sorted(archivos_csv, key=lambda x: x['LastModified'], reverse=True)[0]
    nombre_archivo = archivo_mas_reciente['Key'].replace('.csv', '')

    response = s3.get_object(Bucket=BUCKET_DATOS, Key=nombre_archivo)
    contenido = response['Body'].read().decode('utf-8')
    reader = csv.DictReader(StringIO(contenido))

    asteroides = []
    for row in reader:
      diam = row.get('diametro_promedio', '0 km')
      vel = row.get('velocidad', '0 km/h')
      asteroides.append({
        'nombre': row.get('nombre', 'Desconocido'),
        'diametro': diam,
        'velocidad': vel,
        'distancia': row.get('distancia', '0 km'),
        'peligroso': row.get('es_peligroso', 'No'),
        'nivel_amenaza': clasificar_amenaza(diam, vel)
      })

    if not asteroides:
      raise Exception("No se pudieron procesar datos del CSV.")

    def extraer_num(txt): return float(txt.strip().split()[0]) if txt else 0

    asteroides_ordenados = sorted(
      asteroides,
```

```
        key=lambda x: (-extraer_num(x['diametro']), -extraer_num(x['velocidad']))
    )

    top5 = asteroides_ordenados[:5]
    html = generar_html(top5)

    fecha = datetime.now().strftime("%Y%m%d-%H%M%S")
    nombre_html = f"informe-asteroides-{fecha}.html"
    s3.put_object(
        Bucket=BUCKET_INFORMES,
        Key=nombre_html,
        Body=html,
        ContentType="text/html"
    )

    return {
        'statusCode': 200,
        'body': f"Informe generado correctamente: s3://{BUCKET_INFORMES}/{nombre_html}"
    }

  except Exception as e:
    return {
        'statusCode': 500,
        'body': f"Error generando informe: {str(e)}"
    }

def clasificar_amenaza(diametro, velocidad):
```

```
    try:
        d = float(diametro.strip().split()[0])
        v = float(velocidad.strip().split()[0])
        if d > 1000 or v > 150000:
            return "Clase DIOS 🖊️😱"
        elif d > 500 or v > 80000:
            return "Clase DRAGÓN 🐲🔥"
        elif d > 100 or v > 40000:
            return "Clase DEMONIO 👹💣"
        elif d > 10 or v > 20000:
            return "Clase TIGRE 🐯⚠️"
        else:
            return "Clase LOBO 🐺💤"
    except:
        return "Clase DESCONOCIDA ❓"

def generar_html(asteroides):
    filas = ""
    for idx, ast in enumerate(asteroides, 1):
        clase = " style='background-color:#ffe5e5;'" if ast['peligroso'].lower() in ['sí', 'si', 'true'] else ""
        filas += f"""
        <tr{clase}>
            <td>{idx}</td>
            <td>{ast['nombre']}</td>
            <td>{ast['diametro']}</td>
            <td>{ast['velocidad']}</td>
            <td>{ast['distancia']}</td>
```

```
        <td>{ast['peligroso']}</td>
        <td>{ast['nivel_amenaza']}</td>
    </tr>
    """

html = f"""
<html>
<head>
    <title>🌍 Informe Apocalíptico de Asteroides | NASA-ish 🚀</title>
    <style>
        body {{ font-family: Arial, sans-serif; background-color: #f4f4f4; }}
        table {{ width: 100%; border-collapse: collapse; margin-top: 20px; }}
        th, td {{ border: 1px solid #ccc; padding: 10px; text-align: center; }}
        th {{ background-color: #222; color: white; }}
        .footer {{ margin-top: 30px; font-size: 0.9em; color: #555; }}
    </style>
</head>
<body>
    <h1>🛰️ Top 5 Asteroides Más Letales del Sistema Solar 🌀</h1>
    <p><strong>⚠️ Advertencia:</strong> Algunos de estos cuerpos celestes están clasificados como <em>amenazas de extinción masiva</em>. Pero tranquilos, la última vez solo fue un susto... ¿verdad? 😅</p>
    <table>
        <tr>
            <th>#</th>
            <th>Nombre</th>
            <th>Diámetro</th>
            <th>Velocidad</th>
```

```
            <th>Distancia</th>
            <th>Peligroso</th>
            <th>Nivel de Amenaza</th>
        </tr>
        {filas}
    </table>
    <div class="footer">
        <p>💥 Si alguno de estos impacta la Tierra... ¡bueno, fue un gusto conoceros! 🌍</p>
        <p>Informe generado por <strong>Pablo Vidal Vidal</strong> | {datetime.now().strftime("%Y-%m-%d %H:%M:%S")}</p>
        <p>🚀 Fin del mundo patrocinado por <strong>N.A.S.A. (No Aseguramos Supervivencia Astral)</strong></p>
    </div>
</body>
</html>
"""
return html
```