

SISTEMAS DE BIG DATA - Examen 1^a

Evaluación

Instrucciones generales

1. Todas las sentencias deben ejecutarse desde la línea de comandos en las celdas que hay después del enunciado. No debes realizar ninguna tarea desde fuera de Jupyter.
2. Puedes **añadir** todas las celdas que necesites siempre y cuando estén antes del siguiente enunciado.
3. Todas las celdas **deben estar ejecutadas** y debe visualizarse el resultado de salida.
4. **No es necesario documentar** las respuestas, simplemente debes hacer lo que se pide en el enunciado.
5. Después de cada parte debes insertar una **captura de pantalla** del cliente gráfico de la base de datos correspondientes donde se vea que los datos se han cargado correctamente.
6. Debes entregar tanto el **notebook** (fichero `.ipynb`) como el mismo fichero convertido a **PDF** (es muy probable que si intentas convertirlo en el propio contenedor te falle por no tener instalado `pandoc`, si es así descárgalo en formato `.md` o `html` y conviértelo en tu máquina física).

NOMBRE:

Contexto del escenario

Has sido contratado por una fábrica inteligente que dispone de sensores de temperatura y vibración en sus máquinas críticas. La empresa necesita un sistema backend capaz de procesar los datos que llegan de los sensores en tiempo real.

El sistema debe cumplir dos objetivos simultáneos:

1. **Monitorización en vivo (Dashboard):** los operarios necesitan saber el estado *actual* de cada máquina y si hay alguna alarma activa en este preciso instante. Para esto usarás **Redis**.
2. **Histórico para mantenimiento predictivo:** el equipo de Data Science necesita almacenar todos los datos brutos a lo largo del tiempo para entrenar modelos de IA futuros. Para esto usarás **InfluxDB**.

Los Datos de Entrada

Los datos con los que vas a trabajar los tienes en el *dataset* sintético adjunto llamado `sensores.csv`. Este *dataset* contiene lecturas simuladas con las siguientes columnas:

- `timestamp` : fecha y hora del evento.
- `machine_id` : identificador único de la máquina.
- `zone` : zona de la fábrica.
- `temperature` : temperatura en grados Celsius.
- `vibration` : nivel de vibración (0-100).
- `lat` , `lon` : coordenadas del robot.
- `status` : estado reportado por la máquina ("OK", "WARNING", "ERROR").

IMPORTANTE

El desarrollo del examen debe de ser modular, con un programa principal que inicialice las conexiones a la base de datos y lea los datos del fichero y luego invocará **una función diferente para cargar cada tipo de dato** en la base de datos

Es decisión tuya elegir los parámetros que recibirá cada función, aunque es altamente aconsejable **no utilizar variables globales**.

Parte A: Persistencia histórica (InfluxDB)

2 puntos

En esta parte tienes que crear un script que lea el fichero CSV facilitado y almacene los datos en una base de datos InfluxDB.

Los aspectos que tienes que tener en cuenta son:

- **Bucket:** `factory_logs`
- **Measurement:** `maquinaria`
- **Requisito clave:** debes modelar correctamente los datos usando adecuadamente `tags` o `fields` según el tipo de datos. Se debe respetar el `timestamp` del datos (no usar el tiempo de ingestión).

```
In [9]: # Función que carga los datos en InfluxDB
import csv
from datetime import datetime
import influxdb_client
from influxdb_client.client.write_api import SYNCHRONOUS, WriteOptions
from influxdb_client.client.exceptions import InfluxDBError
from urllib3.exceptions import NewConnectionError
from influxdb_client import Point

INFLUX_URL = "http://influxdb-influxdb2-1:8086"
INFLUX_TOKEN = "MyInitialAdminToken0="

print("--- Iniciando conexión a InfluxDB ---")

client = None

try:
    client = influxdb_client.InfluxDBClient(
        url=INFLUX_URL,
        token=INFLUX_TOKEN,
```

```

        org="docs"
    )
print(f"Verificando estado de salud de InfluxDB en {INFLUX_URL}")
health = client.health()

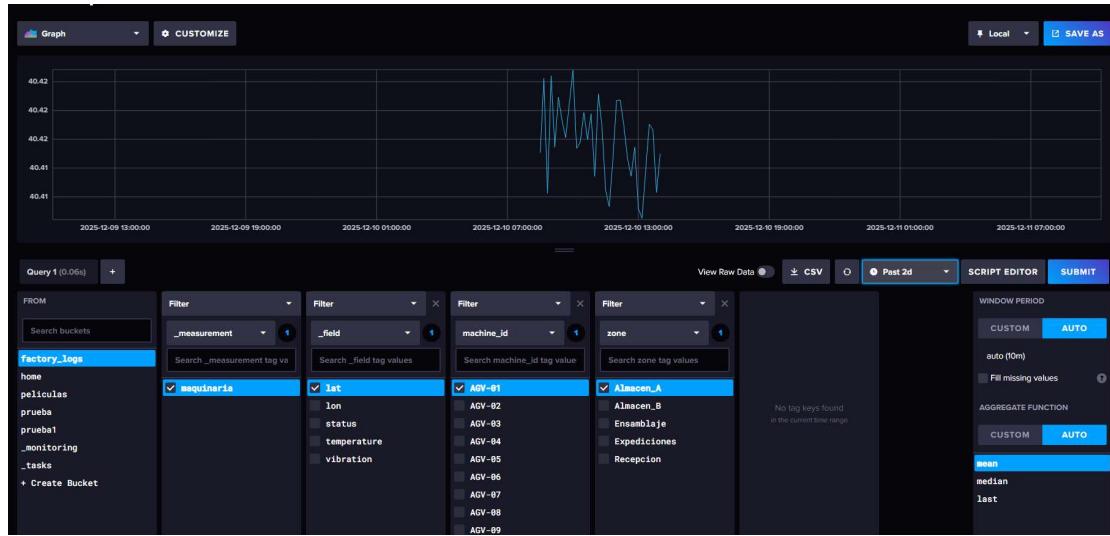
if health.status == "pass":
    print("[INFO] ¡Conexión Exitosa!")
    print(f"[INFO] Versión del servidor: {health.version}")
    ##Aqui va La Logica
    write_api = client.write_api(write_options=SYNCHRONOUS)

    with open("telemetria_agv_1.csv", "r") as f:
        lector = csv.DictReader(f)
        next(lector) #Saltar cabecera
        for fila in lector:
            #Recorremos las filas en forma de Diccionario
            insertarPuntos()
else:
    print(f"[Error] Conexion fallida. Estado: {health.status}")
    print(f"[INFO] Mensaje: {health.message}")
except (InfluxDBError, NewConnectionError) as e:
    print("[ERROR] Error al conectar con InfluxDB:")
    print(f"Detalle: {e}")
finally:
    if client:
        client.close()
        print("--- Conexión Cerrada ---")

```

--- Iniciando conexión a InfluxDB ---
Verificando estado de salud de InfluxDB en http://influxdb-influxdb2-1:8086
[INFO] ¡Conexión Exitosa!
[INFO] Versión del servidor: v2.7.12
--- Conexión Cerrada ---

```
In [7]: def insertarPuntos():
    fecha_original = fila["timestamp"].strip()
    fecha_obj = datetime.strptime(fecha_original, "%Y-%m-%d %H:%M:%S")
    fecha_iso = fecha_obj.isoformat()
    p = Point("maquinaria") \
        .tag("machine_id",fila["machine_id"]) \
        .tag("zone",fila["zone"]) \
        .field("temperature",float(fila["temperature"])) \
        .field("vibration",float(fila["vibration"])) \
        .field("lat",float(fila["lat"])) \
        .field("lon",float(fila["lon"])) \
        .field("status",fila["status"]) \
        .time(fecha_iso)
    write_api.write(bucket="factory_logs",org="docs", record=p)
```



Parte B - Analítica en tiempo real con Redis

Debes crear un script que alimente las siguientes estructuras en Redis por cada dato procesado:

1.- Estadísticas agregadas

1 punto

Al procesar masivamente datos de telemetría, es costoso consultar la base de datos histórica (InfluxDB) para preguntas simples como "¿Cuál ha sido la temperatura máxima hoy en el Almacén A?". Vamos a usar Redis Hashes para mantener un marcador actualizado de estadísticas por zona.

Para cada fila procesada del CSV, debes actualizar un Hash correspondiente a la Zona (zone) donde se encuentra el robot.

- **Clave:** stats:zone:{nombre_zona} (Ej: stats:zone:Almacen_A, stats:zone:Repcion...).
- **Campos::**
 - total_lecturas : contador total de datos recibidos de esa zona.
 - total_errores : contador de cuántas veces el status ha sido "ERROR".
 - max_temp : La temperatura más alta registrada hasta el momento en esa zona.

```
In [10]: # Función que genera las estadísticas agregadas
!pip install redis
```

Requirement already satisfied: redis in /opt/conda/lib/python3.11/site-packages (7.0.1)

```
In [12]: import redis
r = redis.Redis(
    host='redis',
    port=6379,
    db=0,
    decode_responses=True
```

```

)
import csv

In [116... r.set("stats:total_processed",0)
r.set("stats:total_errors",0)
r.set("stats:total_warnings",0)
def ingestar():
    with open("telemetria_agv 1.csv", "r") as f:
        lector = csv.DictReader(f)
        next(lector) #Saltar cabecera
        for fila in lector:
            r.incr("stats:total_processed")
            #Recorremos la filas en forma de diccionario
            if r.hget(f"stats_zone:{fila['zone']}", "total_lecturas") is None:

                if fila["status"] == "ERROR": #si no existe y tiene error que po
                    r.incr("stats:total_errors")
                    r.hset(f"stats_zone:{fila['zone']}", mapping={
                        "total_lecturas": 1,
                        "total_errores": 1,
                        "max_temp": float(fila["temperature"])
                    })
                else: #Si no existe y no tiene error
                    r.hset(f"stats_zone:{fila['zone']}", mapping={
                        "total_lecturas": 1,
                        "total_errores": 0,
                        "max_temp": float(fila["temperature"])
                    })
            else: #Si ya existe que incremente el numero de Lecturas de su zona
                lectura_aux = r.hget(f"stats_zone:{fila['zone']}", "total_lectura")
                lectura_aux = int(lectura_aux)
                lectura_aux += 1
                r.hset(f"stats_zone:{fila['zone']}", "total_lecturas", lectura_aux)
                if fila["status"] == "ERROR":
                    r.incr("stats:total_errors")
                    errores_aux = r.hget(f"stats_zone:{fila['zone']}", "total_lecturas")
                    errores_aux = int(lectura_aux)
                    errores_aux += 1
                    r.hset(f"stats_zone:{fila['zone']}", "total_errores", errores_aux)
                if float(r.hget(f"stats_zone:{fila['zone']}", "max_temp")) < float(fila["temperature"]):
                    r.hset(f"stats_zone:{fila['zone']}", "max_temp", fila["temperature"])

```

In [117... ingestar()

2.- Ranking de "puntos calientes" (Sorted Set)

1 punto

El jefe de planta quiere ver en una pantalla un "Top de Máquinas con mayor temperatura" ordenado de mayor a menor en tiempo real.

- **Estructura:** Sorted Set (ZSET)
- **Clave:** dashboard:hottest_machines
- **Score:** La temperatura actual (temperature).
- **Member:** El ID de la máquina (machine_id).

```
In [119... # Función que carga el sorted set
def leadeboard():
    with open("telemetria_agv 1.csv", "r") as f:
        lector = csv.DictReader(f)
        next(lector) #Saltar cabecera
        for fila in lector:
            #Recorremos la filas en forma de diccionario
            r.zadd(f"dashboard:hottest_machines", {fila["machine_id"]}:fila["tempe
print(r.zrevrange("dashboard:hottest_machines",0,10))
```

```
In [120... leadeboard()
['AGV-06', 'AGV-01', 'AGV-03', 'AGV-08', 'AGV-04', 'AGV-02', 'AGV-05', 'AGV-07',
'AGV-09', 'AGV-10']
```

3.- Seguimiento de flota (Geospatial)

1 punto

Las máquinas de este escenario son AGVs (robots móviles) que se mueven por la planta. Necesitamos saber su ubicación exacta.

- **Estructura:** Geo
- **Clave:** factory:map
- **Datos:** Usa la latitud y longitud que vienen en el CSV para posicionar el machine_id .

```
In [121... # Función que carga Los datos geoespaciales
def geoLoc():
    with open("telemetria_agv 1.csv", "r") as f:
        lector = csv.DictReader(f)
        next(lector) #Saltar cabecera
        for fila in lector:
            r.geoadd("factory_map", (fila["lon"], fila["lat"], fila["machine_id"]))
```

```
In [122... geoLoc()
```

4.- Contadores globales atómicos (String)

1 punto

Necesitamos estadísticas rápidas que no requieran contar filas en una base de datos histórica.

- **Estructura:** String (Contador)
- **Clave:** stats:total_processed -> Incrementar en 1 por cada fila procesada.
- **Clave:** stats:total_errors -> Incrementar en 1 solo si el status es "ERROR".
- **Clave:** stats:total_warnings -> Incrementar en 1 solo si el status es "WARNING".

```
In [108... def contador():
    print(r.get("stats:total_processed"),
```

```
r.get("stats:total_errors"),  
r.get("stats:total_warnings"))
```

In [118]: contador()

9999 658 0

5.- Cola de anomalías críticas (List)

1 punto

Queremos tener también una cola de anomalías críticas. Por cada registro cuyo `status` sea `ERROR` deberás crear un JSON y almacenarlo en una estructura tipo FIFO:

- **Estructura:** `List`
- **Clave:** `alerts:queue`
- **Datos:** el JSON debe incluir: `machine_id`, `timestamp` y un mensaje: "*Critical failure at [Lat, Lon]*".

In [115]: # Función que carga los datos en la cola
r.flushdb()

Out[115]: True

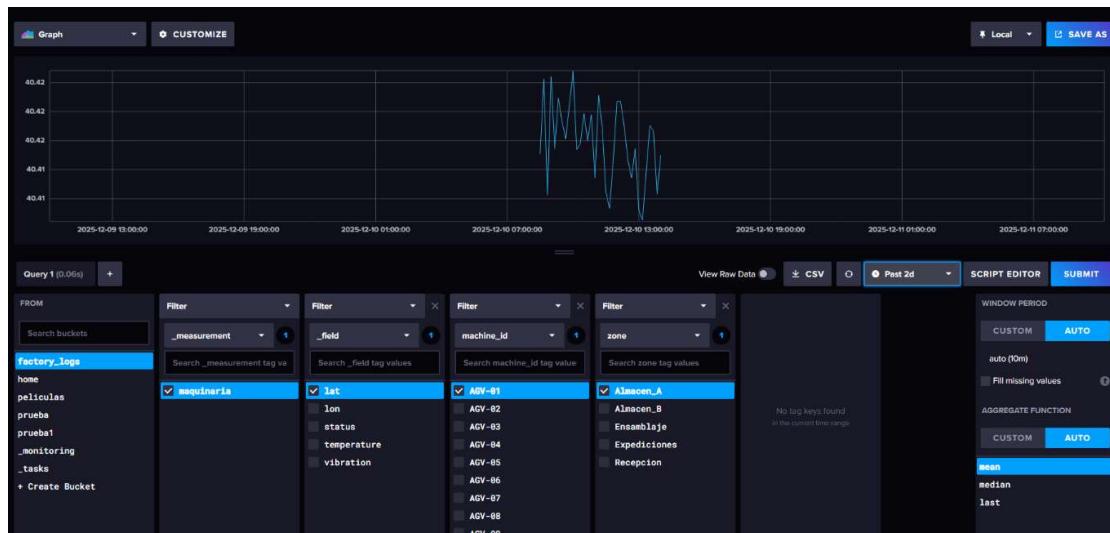
Programa principal

In []: # Aquí debes insertar el programa principal que llama al resto de funciones

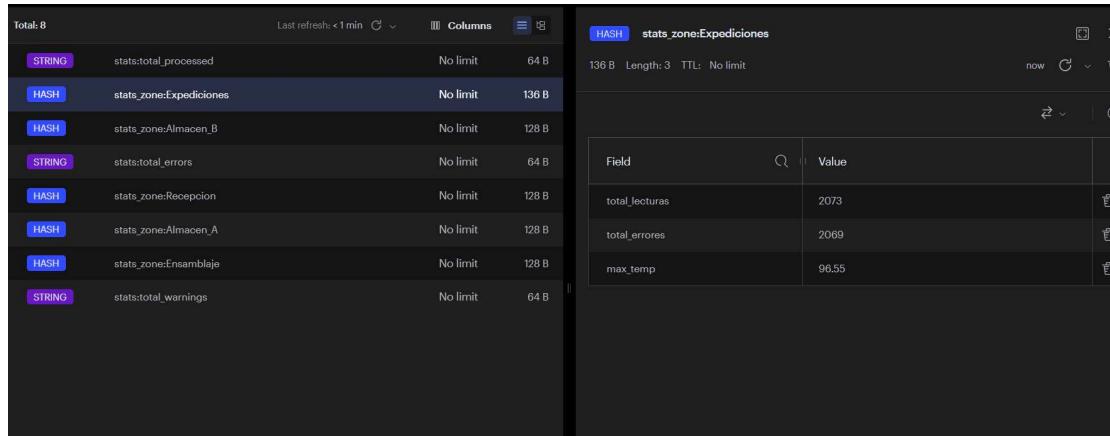
Capturas de pantalla

A partir de aquí tienes que insertar las capturas de pantalla correspondientes a cada punto. Las capturas de pantalla corresponderán a la interfaz gráfica de la base de datos correspondiente y se debe mostrar que los datos se han cargado correctamente. Los apartados que no tengan la captura de pantalla correspondiente **se considerarán no realizados**.

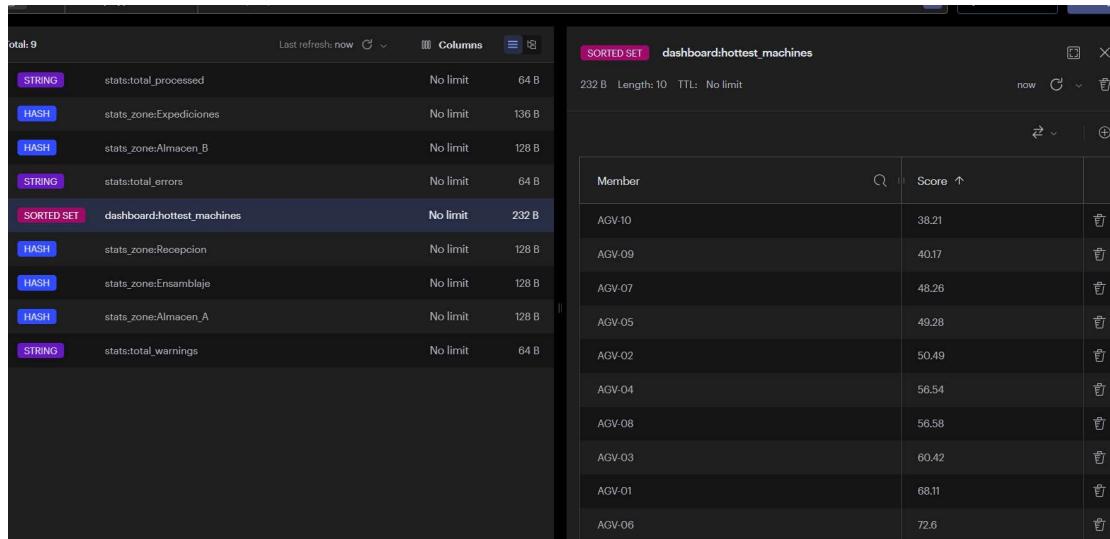
Captura de InfluxDB



Captura de estadísticas agregadas



Captura de ranking de puntos calientes



Captura de seguimiento de flota

The Redis CLI interface shows a list of keys with their types, last refresh time, column settings, and a search bar. A specific key, 'factory_map', is selected and expanded to show its members and scores.

Type	Name	Last Refresh	Score
STRING	stats:total_processed	now	No limit
HASH	stats_zone:Expediciones	now	No limit
SORTED SET	factory_map	now	No limit
HASH	stats_zone:Almacen_B	now	No limit
STRING	stats:total_errors	now	No limit
SORTED SET	dashboard:hottest_machines	now	No limit
HASH	stats_zone:Repcion	now	No limit
HASH	stats_zone:Ensamblaje	now	No limit
HASH	stats_zone:Almacen_A	now	No limit
STRING	stats:total_warnings	now	No limit

Member	Score
AGV-10	1969393805423532
AGV-03	1969393809498688
AGV-07	1969393809819978
AGV-06	1969393810262936
AGV-01	1969393812462438
AGV-09	1969393814818364
AGV-08	1969393821121335
AGV-02	1969393821322063
AGV-04	1969393821633534
AGV-05	1969393827395123

Captura de contadores globales atómicos

The Redis CLI interface shows a list of keys with their types, last refresh time, column settings, and a search bar. A specific key, 'stats:total_processed', is selected and expanded to show its current value.

Type	Name	Last Refresh	Score
STRING	stats:total_processed	< 1 min	No limit
HASH	stats_zone:Expediciones	now	No limit
HASH	stats_zone:Almacen_B	now	No limit
STRING	stats:total_errors	now	No limit
SORTED SET	dashboard:hottest_machines	now	No limit
HASH	stats_zone:Repcion	now	No limit
HASH	stats_zone:Ensamblaje	now	No limit
HASH	stats_zone:Almacen_A	now	No limit
STRING	stats:total_warnings	now	No limit

Value
9999

Captura de cola de anomalías críticas

In []: