SISTEMAS DE BIG DATA - Examen 1ª 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 descargalo en formato `.md o html` y conviértelo en tu máquina física).

NOMBRE: Iker Nieto Garrido

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 decision 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 ingesta).

In [27]:
```
# Función que carga los datos en InfluxDB

from influxdb_client.client.influxdb_client import InfluxDBClient
from urllib3.exceptions import NewConnectionError
from influxdb_client.client.write_api import ASYNCHRONOUS, WriteApi, Writ
from influxdb_client.client.write.point import Point
from influxdb_client.client.exceptions import InfluxDBError

ORG = "docs"
INFLUX_URL = "http://localhost:8086"
INFLUX_TOKEN = "MyInitialAdminToken0=="
BUCKET = "factory_logs"

def data_to_influx(data):
    try:
        client = InfluxDBClient(
            url=INFLUX_URL,
```

```
            token=INFLUX_TOKEN,
            org=ORG
        )
        options = WriteOptions(write_type=ASYNCHRONOUS)
        write_api = client.write_api(write_options=options)
        introduced_points = 0
        not_introduced_points = 0
        for point in data:
            if insert_point(point, write_api):
                introduced_points += 1
            else:
                not_introduced_points += 1
        print("Insertados: ", introduced_points)
        print("No insertados: ", not_introduced_points)
    except NewConnectionError:
        print("Error de conexion")
    finally:
        if client:
            client.close()
def insert_point(point, write_api: WriteApi) -> bool:
    try:
        point = Point("machine_measurement")\
            .tag("machine_id", point[1])\
            .tag("zone", point[2])\
            .tag("lat", point[5])\
            .tag("lon", point[6])\
            .tag("status", point[7])\
            .field("temperature", point[3])\
            .field("vibration", point[4])\
            .time(point[0])
        write_api.write(record=point, bucket=BUCKET, org=ORG)
        return True
    except ValueError as e:
        print(e)
        return False
    except InfluxDBError as e:
        print(e)
        return False
    return False

def read_csv():
    contents = []
    with open("./telemetria_agv.csv") as file:
        text = file.read()
    first = True
    for row in text.split("\n"):
        if first:
            first = False
            continue
        if len(row) < 8:
            continue
        date = True
        row_content = []
        content = row.split(",")
        date = "T".join(content[0].split())
        date += "Z"
        machine_id = content[1]
        zone = content[2]
        temperature = float(content[3])
        vibration = float(content[4])
```

```
        lat = float(content[5])
        lon = float(content[6])
        status = content[7]
        contents.append([date, machine_id, zone, temperature, vibration,
    return contents


data_to_influx(read_csv())
```

```
Insertados:  10000
No insertados:  0
```

Resultado

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:Recepcion...).
- **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 [50]:
```
# Función que genera las estadísticas agregadas
import redis
BASE_KEY = "stats:zone:"
READ_KEY = "total_reads"
ERROR_KEY = "total_errors"
TEMP_KEY = "max_temp"
def insert_zone_stats(zone: str, total_reads: int, total_errors: int, max
    client.hset(f"{BASE_KEY}{zone}", key=READ_KEY, value=total_reads)
    client.hset(f"{BASE_KEY}{zone}", key=ERROR_KEY, value=total_errors)
    client.hset(f"{BASE_KEY}{zone}", key=TEMP_KEY, value=max_temp)
```

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 [44]:
```
# Función que carga el sorted set
import redis
SORTED_KEY = "dashboard:hottest_machines"

def dashboard_hottest(machine_id: int, temperature: float, client: redis.
    client.zadd(SORTED_KEY, {machine_id: temperature})
```

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 [52]:
```
# Función que carga los datos geoespaciales
import redis
GEO_KEY = "factory:map"

def add_coordinates(machine_id: int, lat: float, lon: float, client: redi
    client.geoadd(GEO_KEY, (lon, lat, machine_id))
```

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 [37]:
```python
# Función que gestiona los contadores
import redis
PROCESSED_KEY = "stats:total_processed:"
ERRORS_KEY = "stats:total_errors"
WARNINGS_KEY = "stats:total_warnings"

def manage_counter(status: str, client: redis.Redis):
    if status == "ERROR":
        client.incrby(ERRORS_KEY, 1)
    elif status == "WARNING":
        client.incrby(WARNINGS_KEY, 1)
    client.incrby(PROCESSED_KEY, 1)
```

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 [35]:
```python
# Función que carga los datos en la cola
import redis
import json
ERROR_LIST_QUEUE = "alerts:queue"

def insert_alert(machine_id: int, timestamp: str, lat: float, lon: float,
    json_data = json.dumps({
        "machine_id": machine_id,
        "timestamp": timestamp,
        "message": f"Critical failure at: [{lat},{lon}]"
    })
    client.lpush(ERROR_LIST_QUEUE, json_data)
```

Programa principal

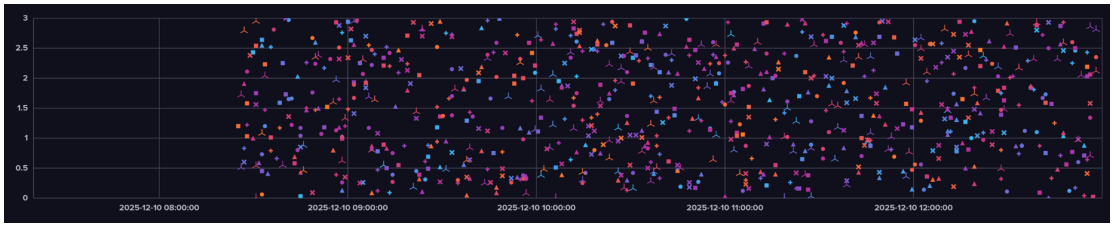
In [53]:
```python
# Aquí debes insertar el programa principal que llama al resto de funcion
import redis
csv_data = read_csv()
zones = dict()
r = redis.Redis(
        host="localhost",
        port=6379,
        db=0,
        decode_responses=True
        )
READS_KEY_DICT = "total_reads"
ERRORS_KEY_DICT = "total_reads"
MAX_TEMP_KEY_DICT = "max_temp"
for row in csv_data:
```

```
    manage_counter(row[-1], r)
    if row[-1] == "ERROR":
        insert_alert(row[1], row[0], row[5], row[6], r)
    dashboard_hottest(row[1], row[3], r)
    add_coordinates(row[1], row[5], row[6], r)
    if zones.get(row[2]):
        zones[row[2]][READS_KEY_DICT] = zones[row[2]].get(READS_KEY_DICT,
        if row[-1] == "ERROR":
            zones[row[2]][ERRORS_KEY_DICT] = zones[row[2]].get(ERRORS_KEY
        if row[3] > zones[row[2]][MAX_TEMP_KEY_DICT]:
            zones[row[2]][MAX_TEMP_KEY_DICT] = row[3]
    else:
        zones[row[2]] = {
            READS_KEY_DICT: 1,
            ERRORS_KEY_DICT: 1 if row[-1] == "ERROR" else 0,
            MAX_TEMP_KEY_DICT: row[3]
        }
for key, values in zones.items():
    insert_zone_stats(key, values[READS_KEY_DICT], values[ERRORS_KEY_DICT
```

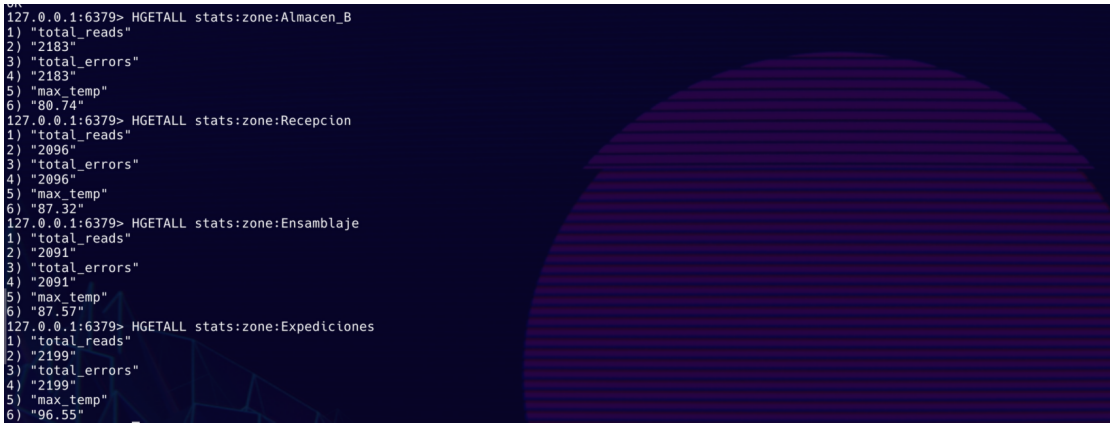
Capturas de pantalla

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



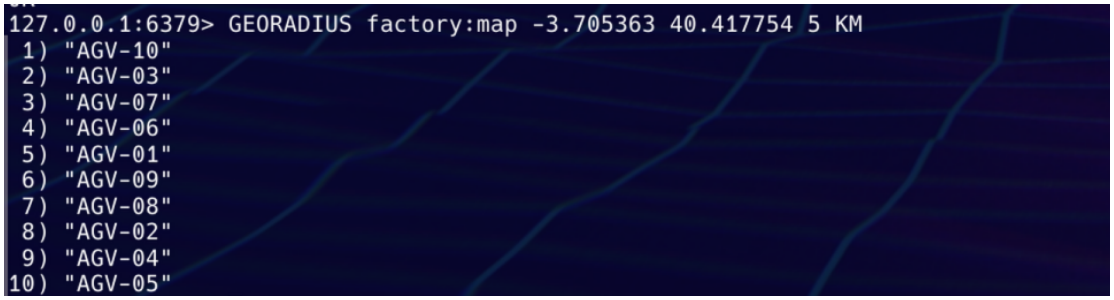
Captura de estadísticas agregadas



Captura de ranking de puntos calientes

Captura de seguimiento de flota

En Redis las coordenadas se guarda como longitud y luego latitud



Captura de contadores globales atómicos

Captura de cola de anomalías críticas