GUÍA PASO A PASO DE LABORATORIO — MÓDULO 07

Persistencia en Series Temporales, Orquestación con Docker Compose, Node-RED, InfluxDB y Grafana

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo: 07 — Capa de Almacenamiento Histórico, Contenedores ARM64 y Observabilidad Industrial

---

1. Objetivos de Ingeniería del Laboratorio

- Montar una unidad de estado sólido externa (`SSD USB 3.0`) en la Raspberry Pi y configurar el reloj de tiempo real por hardware (`DS3231` en el bus `I2C1` dirección `0x68`) junto con el detector optoacoplado de corte eléctrico (`PC817` en `GPIO 17`).
- Desplegar mediante `Docker Compose` una arquitectura multicontenedor aislada sobre Linux ARM64 compuesta por `Eclipse Mosquitto 2` , `InfluxDB 2.7` , `Node-RED` y `Grafana OSS` .
- Programar y auditar el demonio en Python de ingesta resiliente **Store-and-Forward** (`ingestor_store_and_forward_sqlite_influxdb.py`) que respalda muestras en una cola `SQLite3` en modo `WAL` cuando la base de datos principal se encuentra en mantenimiento o desconectada.
- Configurar políticas de retención de datos (`30 días` a `1 Hz`) y tareas automáticas de agregación o *Downsampling* en lenguaje `Flux` (`300:1` hacia ventanas de `5 minutos` con retención de `365 días`).
- Construir paneles de supervisión industrial en `Grafana` cumpliendo la norma de ergonomía visual **ANSI/ISA-101** y flujos de alerta con histéresis en **Node-RED**.

---

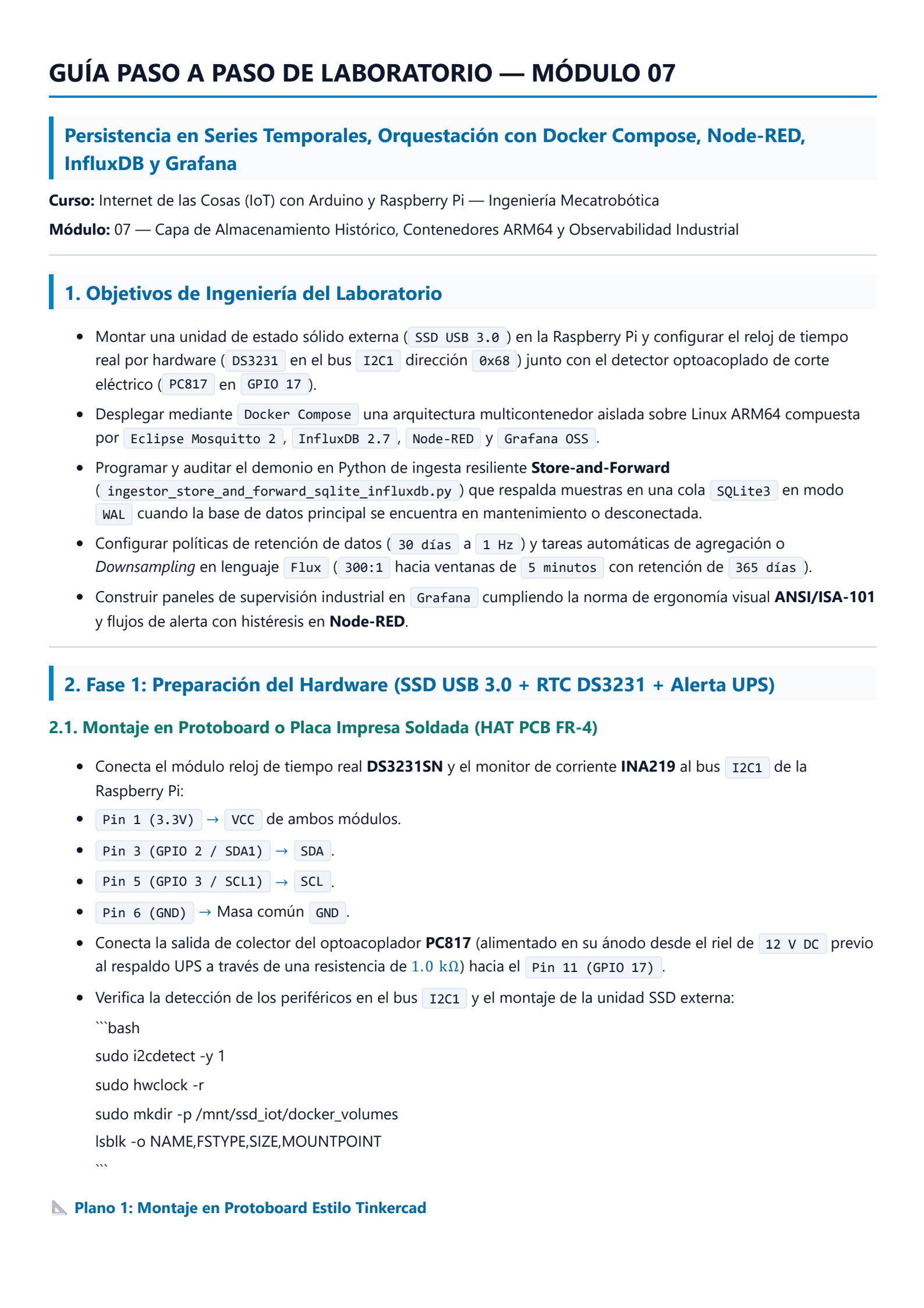
2. Fase 1: Preparación del Hardware (SSD USB 3.0 + RTC DS3231 + Alerta UPS)

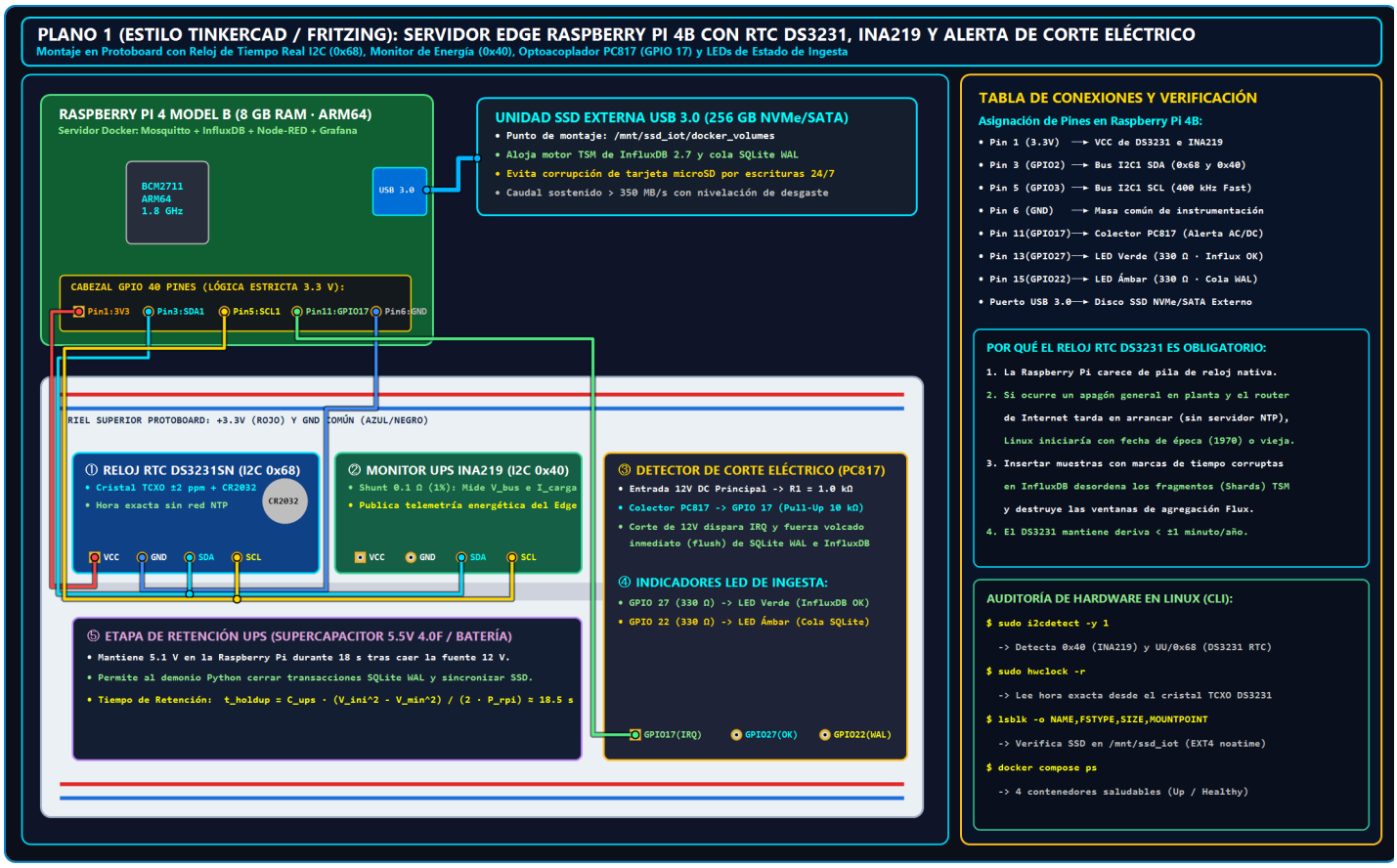
2.1. Montaje en Protoboard o Placa Impresa Soldada (HAT PCB FR-4)

- Conecta el módulo reloj de tiempo real **DS3231SN** y el monitor de corriente **INA219** al bus `I2C1` de la Raspberry Pi:
- `Pin 1 (3.3V)` → `VCC` de ambos módulos.
- `Pin 3 (GPIO 2 / SDA1)` → `SDA` .
- `Pin 5 (GPIO 3 / SCL1)` → `SCL` .
- `Pin 6 (GND)` → Masa común `GND` .
- Conecta la salida de colector del optoacoplador **PC817** (alimentado en su ánodo desde el riel de `12 V DC` previo al respaldo UPS a través de una resistencia de $1.0\ \text{k}\Omega$) hacia el `Pin 11 (GPIO 17)` .
- Verifica la detección de los periféricos en el bus `I2C1` y el montaje de la unidad SSD externa:

  ```bash
  sudo i2cdetect -y 1
  sudo hwclock -r
  sudo mkdir -p /mnt/ssd_iot/docker_volumes
  lsblk -o NAME,FSTYPE,SIZE,MOUNTPOINT
  ```

📐 Plano 1: Montaje en Protoboard Estilo Tinkercad





---

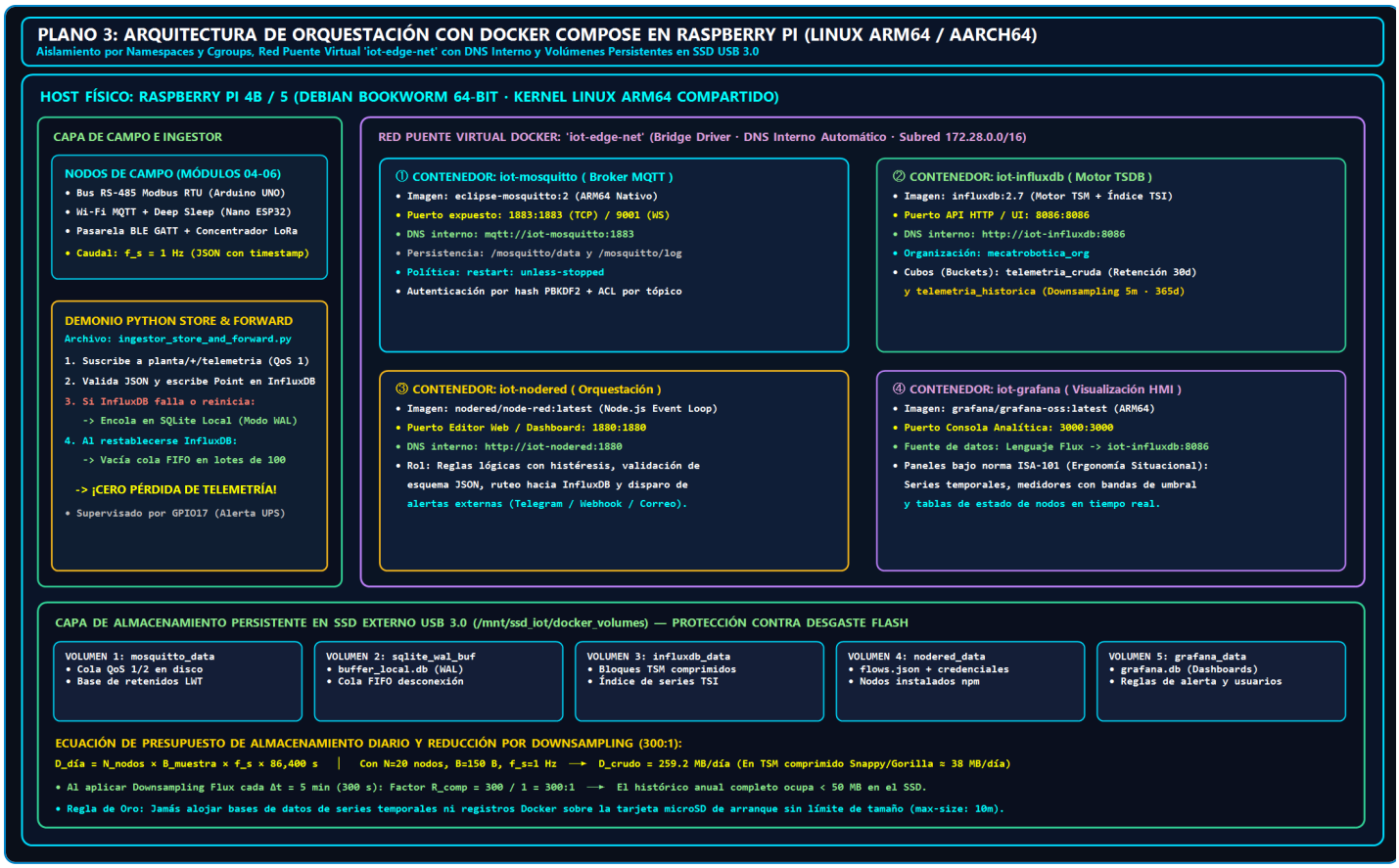
3. Fase 2: Despliegue de la Pila con Docker Compose en ARM64

- Copia el archivo `docker-compose.yml` incluido en la carpeta `codigo_fuente/` hacia `/opt/mecatrobotica/stack_v07/docker-compose.yml`.
- Levanta los cuatro servicios en segundo plano y verifica su estado de salud:

  ```bash
  cd /opt/mecatrobotica/stack_v07
  docker compose up -d
  docker compose ps
  ```

- Comprueba que los cuatro contenedores (`iot-mosquitto`, `iot-influxdb`, `iot-nodered` e `iot-grafana`) reporten estado `Up` y que `iot-influxdb` alcance el estado `healthy` en el puerto `8086`.

📐 Plano 3: Arquitectura Docker Compose en Raspberry Pi ARM64



---

4. Fase 3: Prueba de Fuego del Ingestor Store-and-Forward (SQLite WAL + InfluxDB)

- Instala las bibliotecas cliente en el entorno virtual de la Raspberry Pi:

  ```bash
  pip install paho-mqtt influxdb-client
  ```

- Ejecuta el demonio `ingestor_store_and_forward_sqlite_influxdb.py`.
- **Simulación de caída de base de datos**: Detén temporalmente el contenedor de InfluxDB mientras los nodos continúan publicando telemetría:

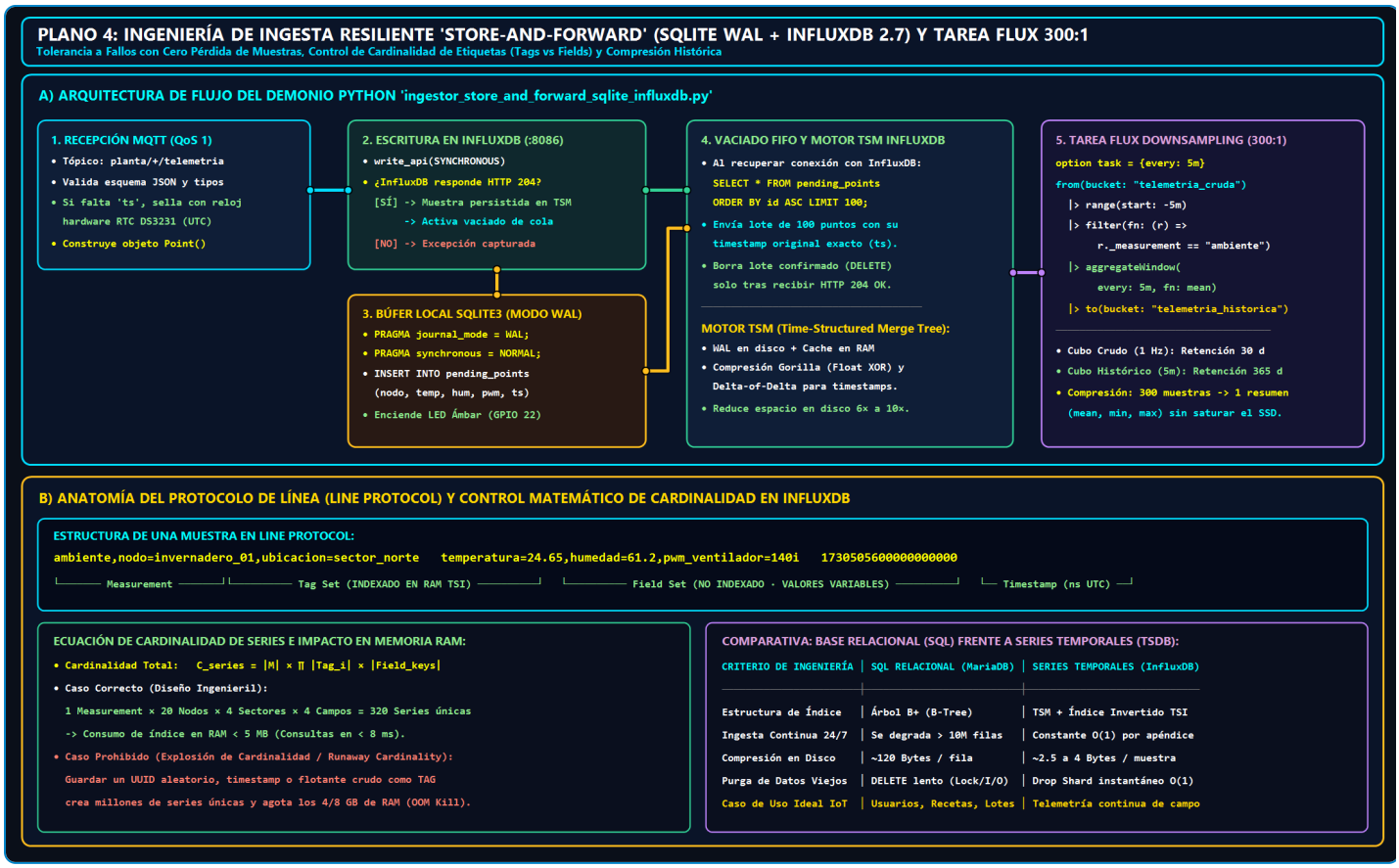
  ```bash
  docker stop iot-influxdb
  ```

- Observa en la consola cómo el demonio captura la excepción de conexión y desvía inmediatamente cada muestra entrante hacia la tabla `pending_points` de `buffer_local_wal.db` (`PRAGMA journal_mode=WAL`).
- **Restauración y vaciado por lotes**: Vuelve a iniciar el contenedor:

  ```bash
  docker start iot-influxdb
  ```

- Verifica cómo en el siguiente ciclo de recepción el demonio vacía automáticamente la cola FIFO en lotes de `100 muestras` (`ORDER BY id ASC LIMIT 100`), insertando toda la serie histórica con sus marcas de tiempo originales intactas.

📐 Plano 4: Arquitectura Store-and-Forward y Tarea Flux 300:1



5. Fase 4: Tarea de Downsampling en Flux y Paneles ISA-101 en Grafana

- Accede a la interfaz web de InfluxDB (`http://<IP_RASPBERRY>:8086`), crea el segundo cubo llamado `telemetria_historica` con retención de `365 días` y registra el script `tarea_downsampling_flux_300a1.flux` en la sección **Tasks**.
- Accede a **Node-RED** (`http://<IP_RASPBERRY>:1880`), importa el archivo `flujo_orquestacion_alertas_nodered.json` y despliega el flujo con validación de esquema JSON e histéresis térmica ($T_{on} = 30.0\ ^\circ C$, $T_{off} = 28.5\ ^\circ C$).
- Accede a **Grafana** (`http://<IP_RASPBERRY>:3000`), configura la fuente de datos **InfluxDB** seleccionando el lenguaje **Flux**, la URL interna `http://iot-influxdb:8086` y el token de administración.
- Diseña el panel de supervisión aplicando la jerarquía de tres niveles y la semántica de colores de la norma **ANSI/ISA-101**.

📐 Plano 5: Ergonomía Visual ISA-101 en Grafana y Flujo Node-RED

PLANO 5: DISEÑO DE INTERFACES HMI BAJO NORMA ISA-101 EN GRAFANA Y FLUJO DE REGLAS EN NODE-RED

Conciencia Situacional para Operadores de Planta, Jerarquía de 3 Niveles, Histéresis de Alerta y Disponibilidad Operativa (SLA)

A) PANEL DE SUPERVISIÓN INDUSTRIAL GRAFANA (:3000) DISEÑADO BAJO ESTÁNDAR ANSI/ISA-101.01

TEMPERATURA PROCESO (NODO 01)

24.6 °C

Banda Nominal: [18.0 .. 28.0 °C] (OK)

HUMEDAD RELATIVA Y PRESIÓN

61.2 %RH | 1013 hPa

Sensor Bosch BME280 (I2C 0x76)

PRE-ALARMA PREVENTIVA (ADVERTENCIA)

29.4 °C (Sector Sur)

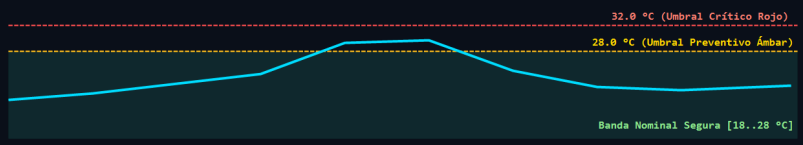
Supera umbral preventivo > 28.0 °C

ALARMA CRÍTICA ISA-18.2 (PRIORIDAD 1)

0 ALARMAS ACTIVAS

Disponibilidad SLA: A = 99.982% (Online)

CURVA DE SERIE TEMPORAL CON BANDA OPERATIVA SOMBREADA (CONSULTA FLUX aggregateWindow)



REGLAS DE ORO DE ERGONOMÍA VISUAL ISA-101:

1. Prohibido el efecto 'Árbol de Navidad': Si todo brilla en rojo, verde y amarillo saturado, el operador sufre fatiga visual y no detecta fallos.
2. Fondos oscuros mate (#0A101C / #121E36) y rejillas de bajo contraste para descanso ocular en turno 24/7.
3. Reserva cromática estricta:
   - Cian / Esmeralda -> Operación dentro de consigna
   - Ámbar (#FFBE1E) -> Desviación preventiva
   - Rojo (#FF5050) -> Alarma crítica que exige acción
4. Incluir siempre unidades físicas (°C, %RH, mA, dBm).

B) FLUJO DE ORQUESTACIÓN EN NODE-RED (:1880) CON HISTÉRESIS DE ALARMA Y CÁLCULO DE DISPONIBILIDAD (SLA)

① MQTT In
planta/+/telemetria
(QoS 1 · JSON)

② JSON Parser &
Schema Validator
(Verifica llaves)

③ Function Node
Filtro Histéresis
(T_on=30, T_off=28.5)

④ RBE / Filter
Report by Exception
(Bloquea spam)

⑤ Salida Dual:
• InfluxDB Batch Out
• Alerta Telegram/MQTT

LÓGICA JAVASCRIPT CON MEMORIA DE CONTEXTO E HISTÉRESIS EN NODE-RED:

```
const T_ALARMA_ON  = 30.0; // Umbral de disparo (°C)
const T_ALARMA_OFF = 28.5; // Umbral de reposición (Banda ΔT = 1.5 °C)
let enAlarma = context.get('enAlarma') || false;
if (!enAlarma && msg.payload.temp >= T_ALARMA_ON) {
    enAlarma = true;  context.set('enAlarma', true);
    return [msg, { payload: { alerta: 'ALTA_TEMP', valor: msg.payload.temp } }];
} else if (enAlarma && msg.payload.temp <= T_ALARMA_OFF) {
    enAlarma = false; context.set('enAlarma', false);
}
```

INGENIERÍA DE CONFIABILIDAD Y DISPONIBILIDAD DEL SISTEMA (MTBF / MTTR):

1. Ecuación de Disponibilidad Estacionaria (Availability):
   A = MTBF / (MTBF + MTTR) = T_operativo / T_total

2. Efecto de Docker 'restart: unless-stopped' + Healthchecks:
   Reduce el Tiempo Medio de Reparación (MTTR) de horas (intervención humana manual) a menos de 3.5 segundos (reinicio automático).

3. Combinado con el búfer SQLite WAL (Store-and-Forward):
   La disponibilidad efectiva de datos alcanza el 100.0% frente a microcortes de red o reinicios de mantenimiento de contenedores.