Manual del Desarrollador: Dashboard para Huerto Urbano Inteligente

Autores: Azuero Maldonado Ronald Alejandro y Toro Ramon Diego Steeven

6 de junio de 2025

Resumen

Este manual proporciona una guía detallada para los desarrolladores que trabajan con el sistema de monitoreo y control del huerto urbano inteligente. Se describe la arquitectura basada en Node-RED, la estructura de los datos que fluyen a través del sistema, las interacciones con APIs externas (OpenWeatherMap) y los protocolos de comunicación (MQTT), así como la implementación del dashboard en Node-RED Dashboard (UI Template). El objetivo es facilitar el entendimiento, mantenimiento y futuras expansiones del proyecto.

Índice

1. Introducción

1.1. Propósito del Manual

El presente documento tiene como objetivo principal servir de referencia técnica para los desarrolladores involucrados en el proyecto del huerto urbano inteligente. Detalla la configuración y el flujo de datos del sistema, con énfasis en los componentes de Node-RED, la interacción con servicios externos y la visualización del dashboard. Se busca proporcionar una comprensión clara de la lógica implementada y las convenciones utilizadas.

1.2. Visión General del Proyecto

El proyecto "Huerto Urbano Inteligente"busca optimizar el cultivo de plantas en entornos urbanos mediante el monitoreo automatizado de condiciones ambientales y el control inteligente de actuadores. El sistema se basa en la plataforma Node-RED para la orquestación de datos y lógica, comunicándose con sensores y actuadores a través de MQTT, consumiendo datos climáticos externos de OpenWeatherMap y proporcionando una interfaz de usuario intuitiva a través de Node-RED Dashboard.

2. Arquitectura del Sistema

El sistema se compone de los siguientes elementos principales:

- **Node-RED**: Plataforma de programación basada en flujos para eventos, utilizada como el cerebro del huerto, integrando diversas fuentes de datos y lógicas de control.
- **MQTT Broker (HiveMQ)**: Protocolo de mensajería ligero para comunicación entre dispositivos IoT. Se utiliza para el control de actuadores (riego, sombra, ventilador) y la suscripción a sus estados.
- **API OpenWeatherMap**: Fuente externa para obtener datos climáticos en tiempo real (temperatura, humedad, descripción del clima) de la ubicación del huerto.
- **Node-RED Dashboard**: Interfaz de usuario (UI) para visualizar el estado del huerto, el historial de datos y controlar los actuadores. Implementado principalmente con nodos 'ui$_t emplate$'.**Sensores (Simulados/Reales)** : $Proporcionandatosdep Hdelsueloy, enunentornoreal$
- **Actuadores**: Dispositivos para realizar acciones físicas como riego, control de sombra y ventilación.
- **Telegram Bot**: Para enviar notificaciones automáticas basadas en la lógica de automatización.

3. Flujo de Datos y Lógica (Node-RED)

El JSON proporcionado corresponde a un flujo de Node-RED. A continuación, se detalla la función de los nodos clave y el flujo de información.

3.1. Obtención y Procesamiento de Datos Climáticos (OpenWeatherMap)

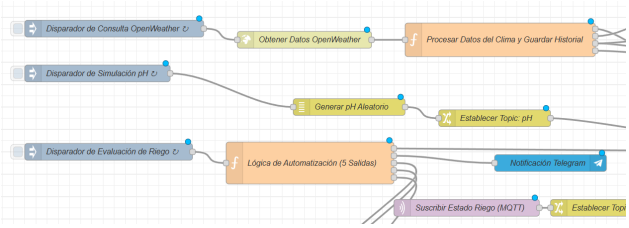

Figura 1: Enter Caption

- **'Disparador de Consulta OpenWeather' (Inject Node)**: Este nodo ('id: 3818de57d4a79f45') se configura para disparar una consulta HTTP cada 300 segundos (5 minutos) una vez al inicio. El 'topic' es 'fetch$_weather$'.**'Obtener Datos OpenWeather' (HTTP Request Node)** : ('$id : af193a01c6f6129d$')$RealizaunasolicitudGETalaAPIdeOpenWeatherMap.$
- - **URL**: 'https://api.openweathermap.org/data/2.5/weather?lat=-3.25757lon=-79.96195units=m
  - **Latitud/Longitud**: Coordenadas específicas de Machala, El Oro, Ecuador (-3.25757, -79.96195).
  - **Units**: 'metric' (grados Celsius).
  - **AppID**: Clave API para autenticación en OpenWeatherMap.
  - **Lang**: 'es' para descripción en español.
  - **Salida**: Objeto JSON ('ret: .ºbj«).
  - **'Procesar Datos del Clima y Guardar Historial' (Function Node)**: ('id: 501bbac507141bb2') Este nodo JavaScript es crucial.
    - Extrae 'temperatura' ('main.temp'), 'humedad' ('main.humidity') y 'descripcion' ('weather[0].description') del 'msg.payload' recibido de OpenWeatherMap.
    - Almacena un historial de estas lecturas en el contexto de flujo ('flow.climaHist'), limitando el historial a las últimas 50 entradas.
    - Genera cuatro salidas:
      1. Objeto consolidado para el ensamblador del dashboard ('topic: "weather$_data$"').$Tempe$ $datos.temperatura$').
2. Humedad para visualizaciones individuales ('msg.payload: datos.humedad').
3. Descripción del clima para visualizaciones individuales ('msg.payload: datos.descripcion').

3.2. Simulación y Procesamiento de Datos de pH

- **'Disparador de Simulación pH' (Inject Node)**: ('id: 8ac7e64d92231db2') Dispara una simulación de lectura de pH cada 15 segundos.
- **'Generar pH Aleatorio' (Random Node)**: ('id: 4feb035bcddafaa9') Genera un valor numérico aleatorio para el pH entre 4.5 y 8.5.

- **‘Establecer Topic: pH‘ (Change Node)**: (‘id: f5d5e7324c9c411d‘) Asigna el ‘topic‘ ‘ph‘ al mensaje, preparándolo para el ensamblador.

3.3. Control y Monitoreo de Actuadores (MQTT)

El sistema utiliza MQTT para comunicarse con los actuadores (riego, sombra, ventilador).

3.3.1. Control de Riego

- **‘Control Riego (Toggle Switch)‘ (UI Template Node)**: (‘id: 793ce3988d3526cb‘) Interfaz en el dashboard para que el usuario active/desactive el riego. Envía un ‘payload‘ (.ºN.º .ºFF”) con el ‘topic‘ ‘huerto/actuador/riego/set‘.
- **‘Publicar Comando Riego (MQTT)‘ (MQTT Out Node)**: (‘id: e6a9d2dcf74f682a‘) Publica el comando de riego al ‘topic‘ ‘utmach/fic/6to/Proyecto/Iot/HuertoUrbano/riego‘ en el broker MQTT.
- **‘Suscribir Estado Riego (MQTT)‘ (MQTT In Node)**: (‘id: 023195a9c3500969‘) Se suscribe al mismo ‘topic‘ (‘utmach/fic/6to/Proyecto/Iot/HuertoUrbano/riego‘) para recibir el estado actual del actuador (confirmación o cambios externos).

- **‘Establecer Topic: Estado Riego‘ (Change Node)**: (‘id: 8e3be2d6d2b7d084‘) Cambia el ‘topic‘ del mensaje entrante a ‘riego$_{s}tatus$‘$paraqueelensambladorloidentifique$.

3.3.2. Control de Sombra

- **‘Control Sombra (Toggle Switch)‘ (UI Template Node)**: (‘id: 5fa5fbf009c0b973‘) Interfaz de usuario para controlar la sombra. Envía ‘payload‘ (.ºN.º .ºFF”) con ‘topic‘ ‘huerto/actuador/sombra/set‘.
- **‘Publicar Comando Sombra (MQTT)‘ (MQTT Out Node)**: (‘id: 28dab68354b9298c‘) Publica el comando al ‘topic‘ ‘utmach/fic/6to/Proyecto/Iot/HuertoUrbano/sombra‘.
- **‘Suscribir Estado Sombra (MQTT)‘ (MQTT In Node)**: (‘id: 6d884fc4fc6a9a4c‘) Se suscribe al ‘topic‘ ‘utmach/fic/6to/Proyecto/Iot/HuertoUrbano/sombra‘ para el estado.

- **‘Establecer Topic: Estado Sombra‘ (Change Node)**: (‘id: 0c41cf93d232f4bf‘) Cambia el ‘topic‘ a ‘shade$_{s}tatus$‘.

3.3.3. Control de Ventilador

- **‘Control Ventilador (Toggle Switch)‘ (UI Template Node)**: (‘id: 1d6447a98391aa54‘) Interfaz de usuario para controlar el ventilador. Envía ‘payload‘ (.ºN.º .ºFF”) con ‘topic‘ ‘huerto/actuador/ventilador/set‘.
- **‘Publicar Comando Ventilador (MQTT)‘ (MQTT Out Node)**: (‘id: fcee5070de9f4c4c‘) Publica el comando al ‘topic‘ ‘utmach/fic/6to/Proyecto/Iot/HuertoUrbano/ventilador‘.
- **‘Suscribir Estado Ventilador (MQTT)‘ (MQTT In Node)**: (‘id: 715ccf24e468a4cc‘) Se suscribe al ‘topic‘ ‘utmach/fic/6to/Proyecto/Iot/HuertoUrbano/ventilador‘ para el estado.

- **‘Establecer Topic: Estado Ventilador‘ (Change Node)**: (‘id: e1f0691c83a873ce‘) Cambia el ‘topic‘ a ‘fan$_{s}tatus$‘.

3.4. Lógica de Automatización

- **‘Disparador de Evaluación de Riego‘ (Inject Node)**: (‘id: 363a10b98b73271e‘) Dispara la lógica de automatización cada 300 segundos (5 minutos).
- **‘Lógica de Automatización (5 Salidas)‘ (Function Node)**: (‘id: cb1e935c16fb2198‘) Contiene la lógica central para la automatización basada en el historial de clima (‘flow.climaHist‘).
  - **Lógica de Riego**: Si la temperatura es $> 20°C$ o la humedad es $< 70\,\%$, activa el riego.
  - **Lógica de Sombra**: Si la temperatura es $> 20°C$, activa la sombra.
  - **Lógica de Ventilador**: Si la temperatura es $> 20°C$, activa el ventilador.
  - **Salidas**: Genera mensajes para:
    1. Ensamblador del dashboard (‘topic: .ªutomated$_s tatus_u pdate$”‘).$Notificaciones de Telegram.$
2. Comandos MQTT para riego, sombra y ventilador.
   - **‘Notificación Telegram‘ (Telegrambot Notify Node)**: (‘id: 1195a3ef65e7f7c2‘) Recibe mensajes de la lógica de automatización y los envía al grupo de Telegram configurado (‘chatId: 1788532181‘).

3.5. Ensamblador de Datos para el Dashboard

- **‘Ensamblar Datos para Dashboard‘ (Function Node)**: (‘id: 8175f91cea9a6467‘) Este nodo centraliza los datos de diversas fuentes (clima, pH, estados de actuadores) y los combina en un único objeto ‘msg.payload‘ para el dashboard.
  - Utiliza el contexto del nodo (‘context.get‘/‘context.set‘) para almacenar los últimos valores de ‘temperatura‘, ‘humedad‘, ‘descripcion‘, ‘ph‘, ‘riego$_s tatus$‘, ‘$shade_s tatus$‘y‘$fan_s ta$

4. Interfaz de Usuario (Node-RED Dashboard)

El dashboard principal se implementa utilizando nodos ‘ui$_t emplate$‘$para una personalización avanzada a$

4.1. Dashboard Principal (UI Template)

- ◦ ‘Dashboard Principal (UI)‘ (UI Template Node): (‘id: 65775cbab450ab3e‘)
  Este es el nodo principal que renderiza la mayor parte del dashboard.
  Contiene HTML, CSS y JavaScript.
  - ◇ Secciones: Define varias secciones visuales:
  - ◇ Estado Actual del Huerto: Muestra la temperatura, humedad, descripción del clima, pH del suelo, estado de riego, estado de sombra y estado de ventilador en formato de "tarjetas"(‘sensor-item‘).
  - ◇ Calendario de Cultivos: Una tabla estática con un calendario de actividades de cultivo.
  - ◇ Visualización Cámara IP: Permite al usuario ingresar una URL/IP de una cámara para visualizar un stream de video en un ‘iframe‘.

- ◇ Historial de Datos (Últimos 30 registros): Una tabla dinámica que muestra un historial de las últimas 30 lecturas de sensores y estados de actuadores, con opción a exportar a CSV.
- ◇ Estilos (CSS): Incorpora CSS para una estética moderna y responsiva, utilizando fuentes de Google (Roboto, Poppins) e iconos de Font Awesome.
- ◇ Funcionalidad (JavaScript):
- ◇ ‘updateSensors(data)‘: Actualiza los valores de las tarjetas de sensores con los datos recibidos en ‘msg.payload‘ del ensamblador.
- ◇ Manejo de entrada para la URL de la cámara IP y actualización del ‘iframe‘.
- ◇ ‘exportBtn‘ Listener: Permite exportar los datos del historial a un archivo CSV.
- ◇ ‘renderSummaryTable()‘: Renderiza el historial de datos en la tabla, mostrando los últimos 30 registros.
- ◇ ‘$watch('msg', function(msg)$‘ : $Observalosmensajesentrantesparaactualizarla UI yañadirn$ $Aplicaestilosvisuales(coloresdefondoytexto)alvalordepH según rangos$:

◇ Menor a 5.5: ‘ph-alert-low‘ (rojo)
◇ Mayor a 7.5: ‘ph-alert-high‘ (naranja)
◇ Entre 5.5 y 7.5: ‘ph-ok‘ (verde)

4.2. Visualizaciones Individuales (UI Template)

Estos nodos (‘id: 18a8e71e073e4132‘, ‘id: 679994c8cfed7534‘, ‘id: 18e52f1a91b3300c‘) proporcionan visualizaciones dedicadas de temperatura, humedad y descripción del clima, respectivamente. Utilizan estilos con Tailwind CSS para un diseño de tarjeta simple.

4.3. Controles de Actuadores (UI Template)

Los nodos ‘Control Riego (Toggle Switch)‘, ‘Control Sombra (Toggle Switch)‘ y ‘Control Ventilador (Toggle Switch)‘ proporcionan interruptores de palanca (‘toggle switches‘) en la interfaz para controlar manualmente los respectivos actuadores. Envían comandos MQTT (‘ON‘/‘OFF‘) y actualizan su estado visual en función de los mensajes de ‘riego$_s tatus$‘, ‘$shade_s tatus$‘y‘$fan_s tatus$‘$recibidos$.

5. Configuraciones de Nodos

5.1. Configuración de MQTT Broker

- **Broker**: ‘broker.hivemq.com‘
- **Puerto**: ‘1883‘
- Se utilizan dos instancias del broker MQTT (‘id: 428204ff249e5d7c‘ y ‘id: 5143f1e78ea12321‘), aunque ambas apuntan al mismo servidor HiveMQ. Es una redundancia o posible error de configuración que se podría optimizar a una sola instancia si no hay requisitos específicos para dos conexiones separadas.

5.2. Configuración de Telegram Bot

- **Nombre del Bot**: 'huertitoUrbanoInteligenteBot'
- **Chat ID**: '1788532181'
- Este nodo requiere que un bot de Telegram haya sido creado previamente en Telegram ('BotFather') y que el 'chatId' sea válido para el destino de las notificaciones.

6. Consideraciones para Desarrolladores

6.1. Extensión de Sensores y Actuadores

- Para añadir nuevos sensores, cree un nuevo flujo que capture la lectura del sensor, establezca un 'topic' único (ej. 'humedad$_suelo$')$ydiríjalaalnodo$'$EnsamblarDatosparaDashboard$'$.Paranuevosactuaa

6.2. Modificación de la Lógica de Automatización

- El nodo 'Lógica de Automatización' es el lugar para ajustar o añadir reglas de control. Accede al historial de clima ('flow.climaHist') y puede acceder a los últimos valores de otros sensores almacenados en el contexto del nodo 'Ensamblar Datos para Dashboard' si se modifican los 'context.set' adecuadamente.
- Asegúrese de que cualquier nueva lógica de automatización genere mensajes de salida para los actuadores MQTT correspondientes y, si es necesario, para las notificaciones de Telegram.

6.3. Personalización del Dashboard

- El nodo 'Dashboard Principal (UI)' es altamente personalizable. El HTML, CSS y JavaScript pueden ser editados directamente para cambiar el diseño, añadir nuevos elementos o modificar el comportamiento de los existentes.
- Si se añaden nuevos tipos de datos, se deberán extender las funciones 'updateSensors' y 'renderSummaryTable' en el JavaScript del 'ui$_t$$emplate$'$paravisualizarlos$.

6.4. Optimización y Escalabilidad

- **Broker MQTT**: Asegúrese de que el broker MQTT (HiveMQ u otro) sea robusto y escalable para un despliegue en producción. Considere una instancia local si la latencia es crítica.
- **Historial de Datos**: El historial en 'flow.climaHist' y 'scope.historial' (en el 'ui$_t$$emplate$')$esvolátil(sepierde$
$RED)ylimitado.Paraunhistorialpersistenteymásgrande, sedebeintegrarunabasededatos(ej.InfluxDBp$
$Laclave$'$appid$'$estáexpuestaenelJSON.Paraunentornodeproducción, considereusarvariablesdeentornode$
$REDoCredencialesparaalmacenartokenssensiblesdeformasegura$.

7. Anexo A: Glosario de Términos

API Interfaz de Programación de Aplicaciones (Application Programming Interface). Conjunto de definiciones y protocolos que se utiliza para diseñar y construir software de aplicaciones.

Broker MQTT Servidor que recibe todos los mensajes de los clientes, los filtra y decide a qué clientes suscritos los debe enviar.

Dashboard Panel de control visual que presenta información clave de un vistazo, en este caso, el estado del huerto.

Flow Context (Node-RED) Almacenamiento de datos que es accesible por todos los nodos en un mismo flujo.

Function Node (Node-RED) Nodo que permite escribir código JavaScript personalizado para procesar mensajes.

Inject Node (Node-RED) Nodo para inyectar mensajes en un flujo, comúnmente usado como disparador manual o repetitivo.

IoT Internet de las Cosas (Internet of Things). Red de objetos físicos que incorporan sensores, software y otras tecnologías para conectarse e intercambiar datos con otros dispositivos y sistemas a través de internet.

JSON JavaScript Object Notation. Formato ligero de intercambio de datos.

MQTT Message Queuing Telemetry Transport. Protocolo de mensajería ligero para dispositivos pequeños y redes con poca ancho de banda.

Node-RED Herramienta de programación visual basada en flujos para cablear hardware, APIs y servicios en línea.

UI Template (Node-RED Dashboard) Nodo que permite insertar HTML, CSS y JavaScript personalizados para crear interfaces de usuario complejas en el dashboard.

8. Anexo B: Fragmentos de Código Relevantes

8.1. Procesar Datos del Clima (Node: 'Procesar Datos del Clima y Guardar Historial')

```
// Este nodo procesa la respuesta de OpenWeatherMap y la prepara para el
    dashboard.
// También guarda los datos en un historial de flujo (flow.climaHist).

let datos = {};
try {
    datos = {
        temperatura: msg.payload.main.temp,
        humedad: msg.payload.main.humidity,
        descripcion: msg.payload.weather[0].description
    };
} catch (e) {
```

```
    node.error("Error al procesar datos de OpenWeatherMap: " + e.message,
    msg);
    // Enviar valores por defecto para evitar errores en nodos siguientes si
     la API falla
    datos = {
        temperatura: '--',
        humedad: '--',
        descripcion: 'Error en API'
    };
}

// Guardar en historial de flujo (flow.climaHist)
let historial = flow.get('climaHist') || [];
historial.push({
    timestamp: Date.now(),
    ...datos // Almacenar el objeto de datos estructurado
});
if (historial.length > 50) historial.shift(); // Limitar a 50 entradas para
    evitar el crecimiento excesivo
flow.set('climaHist', historial);

// Salidas del nodo function:
// Output 1: Objeto completo para el ensamblador del dashboard (topic: "
    weather_data")
// Output 2: Temperatura para visualización directa (ej: en ui_text)
// Output 3: Humedad para visualización directa
// Output 4: Descripción del clima para visualización directa
return [
    { payload: datos, topic: "weather_data" },
    { payload: datos.temperatura },
    { payload: datos.humedad },
    { payload: datos.descripcion }
];
```

Listing 1: Función para Procesar Datos del Clima

8.2. Ensamblar Datos para Dashboard (Node: 'Ensamblar Datos para Dashboard')

```
// Este nodo combina los datos de clima, pH, estado de riego, sombra y
    ventilador
// en un solo objeto para el ui_template principal. Utiliza el contexto del
    nodo
// para almacenar los últimos valores recibidos de cada fuente.

let latestWeatherData = context.get('latestWeatherData') || { temperatura:
    '--', humedad: '--', descripcion: '--' };
```

```
let latestPhData = context.get('latestPhData') || { ph: '--' };
let latestRiegoStatus = context.get('latestRiegoStatus') || { status: '--'
    };
let latestShadeStatus = context.get('latestShadeStatus') || { status: '--'
    };
let latestFanStatus = context.get('latestFanStatus') || { status: '--' };

// Verificar el topic del mensaje entrante para identificar la fuente de
    datos y actualizar el contexto.
if (msg.topic === 'weather_data' && msg.payload) {
    latestWeatherData = {
        temperatura: msg.payload.temperatura,
        humedad: msg.payload.humedad,
        descripcion: msg.payload.descripcion
    };
    context.set('latestWeatherData', latestWeatherData);
} else if (msg.topic === 'ph' && typeof msg.payload === 'number') {
    latestPhData = { ph: msg.payload };
    context.set('latestPhData', latestPhData);
} else if (msg.topic === 'riego_status' && msg.payload !== undefined) {
    latestRiegoStatus = { status: msg.payload };
    context.set('latestRiegoStatus', latestRiegoStatus);
} else if (msg.topic === 'shade_status' && msg.payload !== undefined) {
    latestShadeStatus = { status: msg.payload };
    context.set('latestShadeStatus', latestShadeStatus);
} else if (msg.topic === 'fan_status' && msg.payload !== undefined) {
    latestFanStatus = { status: msg.payload };
    context.set('latestFanStatus', latestFanStatus);
}

// Construir el payload combinado con los datos más recientes para el UI.
msg.payload = {
    temperature: latestWeatherData.temperatura,
    humidity: latestWeatherData.humedad,
    description: latestWeatherData.descripcion,
    ph: latestPhData.ph,
    riego_status: latestRiegoStatus.status,
    shade_status: latestShadeStatus.status,
    fan_status: latestFanStatus.status
};

return msg;
```

Listing 2: Función para Ensamblar Datos para Dashboard

8.3. Lógica de Automatización (Node: 'Lógica de Automatización (5 Salidas)')

```
let climaHist = flow.get('climaHist');

// Inicializar todos los estados a OFF por defecto si no hay datos de clima válidos
let estadoRiego = "OFF";
let estadoSombra = "OFF";
let estadoVentilador = "OFF";
let mensajesTelegram = [];

if (!climaHist || climaHist.length === 0) {
    node.warn("Historial de clima (flow.climaHist) vacío. No se puede automatizar. Enviando OFF por defecto.");
} else {
    let actual = climaHist[climaHist.length - 1];
    if (typeof actual.temperatura === 'undefined' || actual.temperatura === '--' ||
        typeof actual.humedad === 'undefined' || actual.humedad === '--') {
        node.warn("Último dato del historial de clima no tiene el formato esperado o es inválido: " + JSON.stringify(actual));
    } else {
        // --- Lógica de Riego ---
        if (actual.temperatura > 20 || actual.humedad < 70) {
            estadoRiego = "ON";
            let mensajeTelRiego = ` Riego activado automáticamente.\nCondiciones: Temp=${actual.temperatura°}C, Hum=${actual.humedad}%`;
            if (actual.descripcion) mensajeTelRiego += `, Clima: ${actual.descripcion}`;
            mensajesTelegram.push({ payload: mensajeTelRiego });
        }

        // --- Lógica de Sombra ---
        if (actual.temperatura > 20) {
            estadoSombra = "ON";
            let mensajeTelSombra = ` Sombra activada automáticamente.\nCondiciones: Temp=${actual.temperatura°}C`;
            if (actual.descripcion) mensajeTelSombra += `, Clima: ${actual.descripcion}`;
            mensajesTelegram.push({ payload: mensajeTelSombra });
        }

        // --- Lógica de Ventilador ---
        if (actual.temperatura > 20) {
            estadoVentilador = "ON";
```

```
            let mensajeTelVentilador = ` Ventilador activado automáticamente.\nCondiciones: Temp=${actual.temperatura°}C`;
            if (actual.descripcion) mensajeTelVentilador += `, Clima: ${actual.descripcion}`;
            mensajesTelegram.push({ payload: mensajeTelVentilador });
        }
    }
}

// Mensajes de salida
let msgDashboard = {
    payload: {
        riego_status: estadoRiego,
        shade_status: estadoSombra,
        fan_status: estadoVentilador
    },
    topic: "automated_status_update" // Nuevo topic consolidado para el dashboard
};

let msgMqttRiegoCommand = { payload: estadoRiego, topic: "huerto/actuador/riego/set" };
let msgMqttSombraCommand = { payload: estadoSombra, topic: "huerto/actuador/sombra/set" };
let msgMqttVentiladorCommand = { payload: estadoVentilador, topic: "huerto/actuador/ventilador/set" };

// Definir las 5 salidas del nodo Function
// Salida 1 (0): Mensaje Consolidado para Dashboard (riego, sombra, ventilador)
// Salida 2 (1): Mensajes de Telegram (array de mensajes)
// Salida 3 (2): Comando MQTT Riego
// Salida 4 (3): Comando MQTT Sombra
// Salida 5 (4): Comando MQTT Ventilador

return [
    msgDashboard,
    mensajesTelegram.length > 0 ? mensajesTelegram : null, // Solo envía si hay mensajes
    msgMqttRiegoCommand,
    msgMqttSombraCommand,
    msgMqttVentiladorCommand
];
```

Listing 3: Función de Lógica de Automatización