Proyecto de IoT Agrícola con MQTT y Raspberry Pi

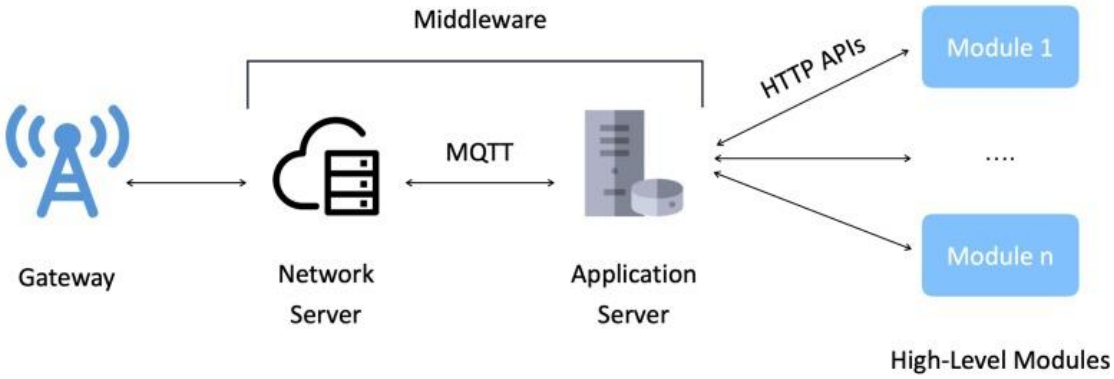

Figura 2: Arquitectura modular de la plataforma LoRaFarM. Los nodos de sensores (EN) se comunican vía LoRaWAN con el gateway Raspberry Pi, que reenvía datos al **Network Server** en la nube. El NS publica los datos en un broker MQTT, de donde un servidor de aplicación los consume para almacenarlos en una base de datos y servirlos a dashboards u otras aplicaciones de usuario. *(Fuente: Codeluppi* et al*., 2020)*

Análisis de Requisitos Comunes (Funcionales y Técnicos)

A partir de los casos estudiados, es posible delinear un conjunto de **requisitos funcionales** y **requisitos técnicos** que son comunes en proyectos de IoT agrícola con MQTT y gateway Raspberry Pi. Estos requisitos son importantes para el diseño de una solución vendible y operativa en este dominio:

Requisitos Funcionales Comunes

- **Monitoreo en Tiempo Real:** El sistema debe permitir la **recolección continua de datos** desde los sensores agrícolas (temperatura, humedad de suelo/aire, pH, posición de ganado, etc.) y proporcionar su visualización en tiempo casi real en un dashboard. Los usuarios (agricultores, técnicos) necesitan **paneles de control** con indicadores clave actualizados al momento.
- **Alertas y Notificaciones:** Debe incorporar lógica para **detectar condiciones fuera de rango** y generar alertas automáticas. Por ejemplo, notificar al usuario si la temperatura en invernadero excede N °C, si la humedad de suelo cae por debajo de un umbral crítico, o si un animal presenta signos anómalos (inactividad, fiebre). Estas alertas deben enviarse vía notificación móvil, SMS o email en forma inmediata.
- **Control de Actuadores:** Además de sensar, muchos casos requieren **actuar automáticamente**. El sistema debe permitir controlar dispositivos como bombas de riego, ventiladores, calentadores, trampas o alimentadores. Las acciones pueden ser automáticas (regla IFTTT local: si humedad < Y -> activar riego) o manuales por el usuario desde la aplicación (botón de "Encender riego ahora"). Debe existir retroalimentación para confirmar que la acción se ejecutó.
- **Configuración de Parámetros por el Usuario:** Un requisito funcional es la capacidad de **ajustar parámetros operativos** sin reprogramar. Por ejemplo, el agricultor debería poder cambiar los umbrales de temperatura que disparan ventilación, fijar horarios de riego, calibrar sensores, etc., todo a través de la interfaz (dashboard) de forma intuitiva. Esto da flexibilidad para adaptar el sistema a diferentes cultivos o condiciones.
- **Registro Histórico y Análisis:** El sistema debe almacenar históricos de datos para permitir **análisis temporal** (gráficas diarias, semanales) e incluso aplicaciones de **analytics**. Los usuarios querrán consultar cómo evolucionó la humedad del suelo en la última semana, o exportar datos

para informes. Esto implica un requisito de persistencia de datos con timestamp y herramientas para consulta (filtros, exportación CSV).

- **Multi-ubicación y Escalabilidad de Dispositivos:** Funcionalmente, la solución debe soportar múltiples nodos y posiblemente **múltiples ubicaciones** (ej. varios invernaderos, diferentes parcelas, o varios ranchos en un mismo sistema). El usuario debería gestionar todos sus sensores desde una sola plataforma, agrupándolos lógicamente (por lote, por zona). Esto requiere que la app maneje **jerarquías** o etiquetas de ubicación, y un número potencialmente grande de dispositivos sin degradar la usabilidad.
- **Funcionamiento Offline (Tolerancia a Fallos de Red):** Dado que en entornos rurales la conexión a Internet puede fallar, el sistema debe funcionar localmente de forma **degradada pero funcional**. Es decir, aunque no haya conexión externa, el gateway de borde (Pi) debe seguir recogiendo datos y controlando actuadores según lo último configurado. Cuando vuelva la conexión, sincronizará datos pendientes. Desde la perspectiva de usuario, quizás no pueda ver datos remotos en vivo durante la desconexión, pero el cultivo no se debe ver afectado por esa caída.
- **Seguridad y Control de Acceso:** A nivel funcional, solo personal autorizado debe acceder a los datos y controles. Deben implementarse roles de usuario (por ejemplo, operador puede ver datos pero no cambiar configuración; administrador sí) y **autenticación segura** en la aplicación. Asimismo, el sistema debería llevar un registro (log) de acciones críticas (p.ej. quién manualmente activó un actuador y cuándo) para trazabilidad.

Requisitos Técnicos Comunes

- **Conectividad e Integración de Protocolos:** Uso de **MQTT** como protocolo central de mensajería, por su eficiencia en IoT. El sistema debe proveer un **broker MQTT confiable** (a menudo Mosquitto) ya sea local (en Raspberry Pi) o en la nube, capaz de manejar las conexiones de todos los dispositivos[9][22]. Además, puede requerir soportar protocolos complementarios: LoRaWAN para larga distancia, HTTP/REST para APIs, WebSockets o CoAP para apps móviles, etc. La arquitectura suele ser multiprotocolo pero con MQTT como columna vertebral de intercambio de datos.
- **Infraestructura de Red Local:** Es necesario contar con una red local robusta en la granja. Según el caso, puede ser Wi-Fi de largo alcance cubriendo el campo, redes malladas, o gateways LoRa. Técnicamente, esto implica planificación de cobertura inalámbrica, configuración de routers/APs, antenas adecuadas, y asegurar **baja latencia** en la red local para que MQTT opere con eficiencia.
- **Capacidad de Cómputo en el Borde:** El Raspberry Pi u otro gateway deben tener recursos suficientes (CPU, RAM, almacenamiento) para ejecutar servicios como el broker MQTT, procesamiento (Node-RED, scripts Python, contenedores Docker, etc.) y bases de datos locales si las hay. A menudo se opta por Raspberry Pi 4 (más potente) si habrá múltiples servicios concurrentes. Si se requieren algoritmos de ML en el borde (p.ej. visión con cámara), puede evaluarse aceleradores (Google Coral, etc.). El sistema debe dimensionarse para el peor caso de carga (ej. decenas de mensajes por segundo).
- **Fiabilidad y Resiliencia:** En entornos de producción, se requieren medidas técnicas de resiliencia: por ejemplo, auto-reconexión de dispositivos MQTT con **QoS** apropiado (uso de QoS 1 o 2 para garantizar entrega de mensajes importantes), implementaciónde retención (*retain flag*) para estados clave (p. ej., último comando conocido para un actuador), y mecanismos de watchdog/restart en el Pi para recuperarse de cuelgues. Idealmente el gateway Pi debería arrancar al encender y restaurar automáticamente los servicios (broker, aplicaciones) sin

intervención. Asimismo, se puede requerir redundancia: por ejemplo, baterías de respaldo (UPS) para el Pi, o incluso nodos gateway duplicados en caso de fallo.

- **Almacenamiento de Datos y Gestión de Historicos:** En el diseño técnico es clave seleccionar la base de datos para históricos. En proyectos pequeños, una **base SQL (MariaDB/PostgreSQL)** en el Pi puede bastar[2], pero para mayor escala se recomienda bases en servidor/cloud. En entornos cloud puede usarse una base de series de tiempo (InfluxDB, Timescale) para eficiencia en datos sensor. Cualquiera sea el caso, debe soportar la frecuencia de datos planificada y el periodo de retención deseado (p.ej. N años de datos). También se necesita considerar políticas de limpieza o archivo para no agotar el almacenamiento local.
- **Seguridad de Comunicación:** Todos los datos sensibles que viajan por la red (local o Internet) deben estar cifrados. Técnicamente, esto implica usar **MQTT sobre TLS** para cualquier comunicación Pi↔Cloud, utilización de VPN o TLS incluso en la LAN si hay riesgo de intrusos, y cifrar las APIs web (HTTPS). Adicionalmente, se deben gestionar credenciales seguras para el broker MQTT (usuarios/contraseñas o certificados cliente) en lugar de permitir un broker abierto. Los dispositivos IoT deben tener credenciales únicas para evitar que un compromiso de uno afecte a todos.
- **Escalabilidad y Modularidad:** El sistema debería diseñarse de forma distribuida para poder crecer. Por ejemplo, soportar **múltiples Raspberry Pi gateways** comunicándose con un servidor central, si la extensión de la granja lo requiere. O poder migrar componentes a la nube si la carga aumenta (por ejemplo, empezar con Node-RED local pero luego mover la lógica a un servicio cloud manteniendo el Pi solo como gateway). Modularidad también en el software: usar contenedores Docker en el Pi puede ayudar a aislar el broker, la app Node-RED, la base de datos, facilitando actualizaciones sin afectar todo el sistema.
- **Interoperabilidad y APIs:** Un requisito técnico importante es exponer **APIs abiertas** para integrar con otros sistemas o expandir funcionalidades. Por ejemplo, una API REST para consultar datos históricos, o posibilidad de suscribirse a ciertos tópicos MQTT de manera externa (con las debidas autenticaciones). Esto permite en un futuro conectar la plataforma a sistemas de información agrícola más amplios o plataformas IoT comerciales, aumentando el valor del demo al ser integrable.
- **Gestión y Actualización Remota:** Dado que los dispositivos (RPi, sensores) pueden estar desplegados en campo, es deseable poder administrarlos de forma remota. Técnicamente, incluir capacidades de **OTA (Over-the-Air)** para actualizar firmware de sensores y actualización remota del software del Pi (ya sea vía SSH, o usando alguna plataforma de device management). Esto garantiza que se puedan aplicar mejoras o parches de seguridad sin desplazarse físicamente.

En resumen, estos requisitos garantizan que la solución resultante sea **funcionalmente útil para la agricultura** (monitoreo, control, alertas, histórico) y **técnicamente sólida** (conectividad robusta, procesamiento adecuado en borde/nube, seguridad y escalabilidad). Al diseñar una demo operativa vendible, cumplir con estos puntos asegura que la propuesta sea confiable y atractiva para implementaciones reales.

Casos de Uso Típicos para una Demo Operativa

A partir de las capacidades anteriores, se pueden definir varios **casos de uso demostrativos** que resaltan el valor de un sistema IoT agrícola con MQTT y Raspberry Pi. Estos casos de uso son ejemplos concretos, fáciles de entender para clientes potenciales, que muestran cómo la tecnología resuelve problemas reales en el campo:

- **Monitorización de Microclima en Invernaderos:** Demostrar cómo sensores de temperatura, humedad y CO_2 dentro de un invernadero envían datos al instante al dashboard. Si la temperatura sube demasiado, el sistema automáticamente enciende ventiladores y envía una alerta al móvil del encargado. El usuario puede ver en la interfaz el historial diario de temperatura y cómo las acciones del sistema la mantienen en rango óptimo. *(Valor:* optimiza el crecimiento de cultivos protegidos y evita golpes de calor).*

- **Riego de Precisión en Campo Abierto:** Mostrar un sector de cultivo (por ejemplo, hortalizas o un viñedo) equipado con sensores de humedad de suelo y un Raspberry Pi conectado a válvulas de riego. La demo enseñaría cómo, al caer la humedad bajo cierto nivel en una parcela, el sistema activa esa zona de riego automáticamente vía MQTT. En el dashboard se ve qué sectores están siendo regados y el agricultor puede manualmente ajustar la duración o posponer el riego desde su teléfono. *(Valor:* ahorro de agua y mejora de rendimientos al regar solo cuando es necesario).*

- **Seguimiento de Ganado en Tiempo Real:** Presentar collares inteligentes con GPS y sensor de actividad en vacas que envían datos periódicamente a través de un gateway Pi con conexión celular. En la demo, el mapa del dashboard muestra la ubicación de cada animal en la finca. Si una vaca se sale de un perímetro geográfico (geo-cerca) o deja de moverse por horas (posible problema de salud), el sistema alerta al ganadero. Incluso se podría integrar un sensor de parto que notifique automáticamente cuando una vaca está de parto. *(Valor:* reduce pérdidas de animales extraviados y permite atención oportuna a problemas de salud).*

- **Control Remoto de Sistemas de Irrigación y Equipos:** Un caso de uso orientado a **telecontrol**: por ejemplo, encender/apagar un sistema de aspersores o una bomba de agua de un pozo de forma remota. La demo podría incluir un botón en el dashboard que envía via MQTT un comando al Raspberry Pi en campo conectado a un relé de la bomba. También feedback de sensores de caudal que confirman si el agua fluye. Otro ejemplo es controlar cortinas de sombra en un invernadero o luces UV para cultivo indoor mediante la plataforma. *(Valor:* brinda comodidad y capacidad de reacción rápida sin estar físicamente presente).*

- **Analítica de Nutrientes y Fertilización Automatizada:** Para cultivos intensivos, un caso de uso avanzado es mostrar sensores de CE (conductividad eléctrica) o NPK en solución de riego, donde el sistema detecta niveles bajos de fertilizante y activa una bomba dosificadora para corregir. En la demo, el usuario vería gráficas de nutrientes y pH en el suelo, y cómo el sistema añade fertilizante o acidulante automáticamente manteniendo valores óptimos. El usuario podría configurar las concentraciones deseadas y recibir un reporte del consumo de fertilizante. *(Valor:* optimiza fertilización, evitando tanto carencias como desperdicios/costos por fertilizar de más).*

- **Detección Temprana de Incendios o Heladas:** Un caso orientado a gestión de riesgos climáticos: desplegar sensores de temperatura del aire, humo y velocidad de viento en zonas forestales o agrícolas. La demo podría simular una caída brusca de temperatura nocturna, donde el sistema avisa riesgo de helada y automáticamente enciende ventiladores o quemadores anti-heladas. O bien detectar humo/alta temperatura en un campo, indicando un posible incendio forestal, generando una alarma inmediata con ubicación. *(Valor:* protege activos agrícolas y permite intervenir antes de que el daño sea catastrófico).*

Cada uno de estos casos de uso muestra aspectos vendibles de la solución: **ahorro de costes** (agua, insumos), **mejora de productividad** (ambiente óptimo para plantas/animales), **seguridad y protección** (alertas tempranas), y **comodidad** (automatización y control remoto). Una demo integral podría

incorporar varios de estos escenarios en un entorno simulado o real a pequeña escala, para impresionar a los stakeholders con un sistema funcional y versátil.

Diagramas de Arquitectura y Flujo de Operación de un Sistema Típico

Para visualizar la solución integrada que combina las lecciones de los casos anteriores, a continuación se presentan diagramas de la **arquitectura general** y del **flujo de operación** de un sistema IoT agrícola típico con Raspberry Pi, MQTT y procesamiento local/nube.

Arquitectura General (Capas y Componentes): En un esquema general, se distinguen tres capas principales: **dispositivos de campo**, **gateway de borde** y **backend/cloud**. En la figura a continuación se ilustra esta arquitectura: los sensores y actuadores en campo se conectan al gateway (Raspberry Pi) mediante protocolos locales (Wi-Fi, LoRa, Zigbee, etc.), el Pi ejecuta servicios locales (broker MQTT, Node-RED) y conecta con la nube para almacenamiento y visualización.

- En la capa de **dispositivos**, se incluyen sensores ambientales (ej. temperatura, humedad, luminosidad), sensores de suelo (humedad, nutrientes), sensores de ganado (GPS, biométricos), así como actuadores (válvulas de riego, motores, ventiladores, alimentadores automáticos). Estos dispositivos pueden estar conectados directamente por Wi-Fi, o via dispositivos intermedios (por ejemplo, sensores conectados a un microcontrolador Arduino/ESP que a su vez comunica con el Pi).
- El **Raspberry Pi gateway** constituye la capa de **borde (fog)**. Suele ejecutar un broker MQTT (punto de reunión de mensajes de sensores y actuadores)[1], y la lógica local (por ejemplo, flujos Node-RED, scripts Python o programas en C/C++ o Java dependiendo de la complejidad). Puede incorporar también una base de datos local para buffering o almacenamiento temporal. El Pi se conecta a Internet mediante Ethernet, Wi-Fi o modem 4G, actuando como puente seguro entre la red local y la nube.
- En la **nube/servidor**, corre la aplicación de backend: esta recibe datos del Pi (vía MQTT bridge, HTTP API u otros) y los almacena en bases de datos de larga duración[24]. También hospeda el dashboard web y APIs para que los usuarios accedan. Aquí se pueden integrar servicios adicionales como análisis de datos, entrenamiento de modelos con históricos, etc. Los usuarios finales (agricultores, técnicos) interactúan mediante navegadores web o apps móviles con esta capa, que a su vez se comunica con el gateway según sea necesario (solicitando datos o enviando comandos MQTT de control).

Figura 3: Diagrama de arquitectura típico para IoT agrícola con MQTT. Los sensores/actuadores en campo (izquierda) se comunican con el **gateway Raspberry Pi** (centro) mediante protocolos locales (ej. MQTT sobre Wi-Fi o LoRa). El Raspberry Pi procesa datos localmente y sirve de puente al **backend en la nube** (derecha), donde se almacenan los datos en BD y se proporciona el dashboard para los usuarios. *(Fuente: Adaptado de D'Ortona* et al.*, 2022)*

Flujo de Operación (Secuencia de Eventos): A continuación se describe el flujo de operación paso a paso de este sistema cuando, por ejemplo, un sensor de humedad de suelo detecta que la tierra está seca y el sistema realiza un riego automatizado:

1. **Medición y Publicación:** El sensor de humedad (ubicado en el campo) mide el valor actual (por ejemplo 20% de humedad volumétrica) y lo envía inmediatamente. Si es un sensor LoRa, transmite al gateway; si es un sensor Wi-Fi/ESP, publica ese dato en un tópico MQTT (p.ej. `finca1/suelo/sectorA/humedad`) apuntando al broker del Raspberry Pi.

2. **Recepción en el Gateway:** El **broker MQTT en el Raspberry Pi** recibe el mensaje del sensor. Un flujo de procesamiento en Node-RED está suscrito a `finca1/suelo/sectorA/humedad` y desencadena una función al llegar el nuevo dato (20%).
3. **Lógica Local de Decisión:** Node-RED (u otro componente local) compara el 20% recibido con el **umbral de humedad mínimo** configurado (supongamos 30%). Dado que está por debajo, determina que se necesita riego. Entonces, publica un **mensaje MQTT** en el tópico correspondiente al actuador de la electroválvula de riego (por ejemplo `finca1/actuadores/sectorA/riego` con payload "ON"). También marca una hora de inicio de riego.
4. **Activación del Actuador:** Un controlador (puede ser el mismo Pi o un microcontrolador conectado a él) suscrito al tópico de la válvula recibe el comando "ON" y enciende la bomba o abre la electroválvula. El suelo empieza a regarse en el Sector A. El actuador puede opcionalmente enviar un mensaje de **feedback** confirmando que se ejecutó la acción (ej. `finca1/actuadores/sectorA/riego/estado` = "abierto").
5. **Notificación y Registro:** El sistema local envía opcionalmente una **notificación** al usuario (por ejemplo, Node-RED dispara un correo o push notificando "Riego automatizado iniciado en Sector A"). Asimismo, registra el evento en la base de datos (tanto la lectura baja de humedad como la acción realizada con timestamp). Si hay conectividad, también publica este evento en la **nube** – por ejemplo, a través de un bridge MQTT que replica ciertos tópicos en un broker en la nube o vía una API REST. De ese modo, el dashboard remoto podrá reflejar que el riego está activo.
6. **Visualización en Dashboard:** En el dashboard web, el usuario puede ver en tiempo real que la humedad del Sector A estaba baja (20%) y que el sistema puso el riego en marcha a tal hora. Una animación o indicador "Riego ON" aparece en ese sector del mapa. El usuario, si desea, puede intervenir: por ejemplo, podría decidir apagar el riego de forma remota (lo que generaría un comando manual MQTT "OFF" al actuador) o ajustar el umbral para la próxima vez.
7. **Finalización y Ciclo:** Tras un tiempo predeterminado (o cuando un sensor de flujo indique suficiente riego), Node-RED envía el comando de **apagar** la válvula. El actuador cierra el paso de agua, y el sistema registra la duración del riego y el nuevo valor de humedad del suelo tras el riego (que idealmente subió por encima del umbral). El ciclo continúa, volviendo a monitorear lecturas periódicamente para futuras decisiones.

Este flujo ejemplifica un **ciclo de control cerrado IoT**: sensor -> gateway (decisión) -> actuador -> retroalimentación. Aprovecha MQTT para comunicaciones rápidas y desacopladas entre componentes. Cabe destacar que, en paralelo, otros sensores y procesos pueden estar ocurriendo (por ejemplo, monitoreo de clima, detección de batería baja en un sensor, etc.), todos coexistiendo en distintos tópicos MQTT sin interferencia gracias al modelo pub/sub.

En resumen, la arquitectura y flujos descritos integran las fortalezas observadas en proyectos reales: un **gateway de borde inteligente** con Raspberry Pi que asegura autonomía local y respuesta inmediata, combinado con la **conectividad a la nube** que brinda supervisión holística y almacenamiento central. Esta combinación proporciona una solución IoT agrícola **fiable, escalable y práctica**, capaz de aumentar la eficiencia en el campo y reducir las incertidumbres propias de la agricultura tradicional. Las demos y casos de uso construidos sobre este patrón muestran un camino claro hacia la agricultura inteligente, donde la tecnología trabaja de la mano con el productor para optimizar los resultados.

Fuentes: Los casos y análisis presentados se basan en documentación de proyectos open-source, estudios académicos y experiencias reportadas, incluyendo: un proyecto Node-RED de control de invernadero con Raspberry Pi, una arquitectura de computación en la niebla para granjas inteligentes, y

la plataforma modular LoRaFarM evaluada en campos reales[14][23], entre otros. Estas referencias respaldan las mejores prácticas tecnológicas y lecciones aprendidas en entornos de IoT agrícola con MQTT y edge computing.