📡 Arquitectura MQTT IoT + IA (Local ↔ Nube)

Este proyecto implementa una **arquitectura híbrida** que combina:

- 🚀 **Baja latencia y resiliencia local**: toda la telemetría y acciones automáticas funcionan dentro de la LAN sin Internet.
- ☁️ **Escalabilidad y control remoto**: un broker en EMQX Cloud replica tópicos clave y expone UI remota + app móvil.
- 🤖 **Inferencia dual (local + nube)**: para garantizar rapidez y respaldo en la detección de caras.

🗂️ Diccionario de Topics MQTT

Topic	Publica	Suscribe	Propósito
`state/telemetry/{device_id}`	Hardware	HA local, Broker Local, App Móvil	Envía temperatura, humedad y estado de actuadores
`command/{device_id}/light`	HA local, HA nube, App Móvil	Hardware	Comando ON/OFF para el LED
`command/{device_id}/fan`	HA local, HA nube, App Móvil	Hardware	Comando ON/OFF para el ventilador
`image/snapshot/{device_id}`	Hardware	Flask API local	Snapshot para inferencia
`result/{device_id}`	Flask API (local o nube)	HA local, Broker Local, Broker Nube	{has_face: true/false}
`alert/{device_id}`	Flask API local (alias)	App Móvil	Notificación push de detección de cara
`state/#`	Broker Local (bridge)	Broker Nube	Replica telemetría al broker en la nube
`result/#`	Broker Local (bridge)	Broker Nube	Replica resultados de inferencia a la nube
`command/#`	Broker Nube (bridge)	Broker Local	Replica comandos desde la nube
`alert/#`	Broker Nube (bridge)	Broker Local	Replica alertas desde la nube

---

🚀 Descripción Detallada por Actor

1. Hardware Embebido (ESP32 / Raspberry Pi)

- **Rol**: Sensado (temp, humidity, cámara) + control GPIO (LED, ventilador).
- **Publica**:

```
{
  "temperature": 28.5,
  "humidity": 65,
  "light": "OFF",
  "fan": "ON",
```

```
  "timestamp": "2025-05-26T14:32:10Z"
}
```

en `state/telemetry/{device_id}` cada 5–30 s.

- **Se suscribe**:
  - `command/{device_id}/light`
  - `command/{device_id}/fan`
- **Acción**: traduce MQTT→GPIO y refleja su nuevo estado republicando (opcional).

---

2. Broker MQTT Local (Mosquitto / EMQX)

- **Rol**:
  1. Enrutamiento en LAN (HA local, Flask, hardware, app WS).
  2. **Bridge bidireccional** con EMQX Cloud:

```
connection emqx-bridge
address   <tu-instancia>.ala.dedicated.aws.emqxcloud.com:```3

# Credenciales EMQX Cloud
remote_username  user123
remote_password  123456789
clientid          mosquitto-to-emqx

# TLS
bridge_cafile    /opt/homebrew/etc/mosquitto/certs/ca.crt
bridge_insecure  true

# OUT: local → nube
topic state/telemetry/# out 0 "" ""
topic result/#          out 0 "" ""
topic alert/#           out 0 "" ""

# IN: nube → local
topic command/# in 0 "" ""
topic alert/#   in 0 "" ""

try_private                false
bridge_attempt_unsubscribe false
cleansession              true
keepalive_interval        60
notifications             false
```

- **Notas**:
  - Ajusta `local_prefix` / `remote_prefix` si quieres renombrar árboles de tópicos.
  - Usa `no_local true` para evitar que un bridge reciba sus propios mensajes.

---

3. Flask API Local

- **Rol**: orquesta inferencia y persiste detecciones.
- **Flujo**:
  1. Suscribe `image/snapshot/{device_id}`.
  2. POST simultáneo a IA local y AWS (SageMaker/Lambda).

3. Publica primer resultado en `result/{device_id}`.
4. Inserta en PostgreSQL local (`pending_upload = TRUE`).

- **Código de ejemplo**:

```
def on_mqtt_image(client, userdata, msg):
    img_path = decode_snapshot(msg.payload)
    res_local = ia_local.infer(img_path)
    res_cloud = ia_cloud.infer(img_path)
    result = res_local or res_cloud
    client.publish(f"result/{DEVICE}", json.dumps(result))
    save_to_db(device=DEVICE, result=result, pending=True)
```

---

4. IA Local vs IA Nube

Característica	IA Local	IA Nube
Latencia	<50 ms	100–300 ms
Capacidad	CPU/GPU integrada RPi	Instancias dedicadas AWS
Falla fallback	AWS responde	Local responde si disponible

---

5. Home Assistant Local

- **Rol**: UI + automatización en LAN.
- **Subscripciones**:
  - `state/telemetry/raspi1`
  - `result/raspi1`
- **Publica**:
  - `command/raspi1/light`
  - `command/raspi1/fan`
- **Automatización ejemplo**:

```
- alias: "Ventilador > 30°C"
  trigger:
    platform: mqtt
    topic: "state/telemetry/raspi1"
  condition:
    value_template: "{{ value_json.temperature > 30 }}"
  action:
    service: mqtt.publish
    data:
      topic: "command/raspi1/fan"
      payload: "ON"
```

---

6. Broker MQTT Nube (EMQX Cloud)

- **Rol**: réplica global de mensajes con TLS/WSS.
- **Clientes típicos**:

- HA nube
- App React Native
- Servicios de análisis externos

---

7. Home Assistant en la Nube

- **Subscripciones**:
  - `state/telemetry/#`
  - `result/#`
- **Publica**:
  - `command/{device_id}/…`
- **UI**: dashboards remotos idénticos a los locales.

---

8. App React Native

- **Rol**: notificaciones push + control manual.
- **Lógica de conexión**:
  1. WebSocket → broker local (URL WS).
  2. Si falla, TLS ```3 → EMQX Cloud.
- **Subscripciones**:
  - `alert/#`
  - `state/telemetry/#`
- **Publica**:
  - `command/{device_id}/{…}`

> **Resumen**
>
> 1. **Telemetría**: Hardware → broker local → (bridge) → EMQX → UI nube + móvil.
> 2. **Inferencia**: Flask local → IA local/nube → `result/…` → lan + nube.
> 3. **Acciones**: HA local/nube o app → `command/…` → hardware.
>
> Esta configuración ofrece resiliencia offline, baja latencia en LAN y acceso global seguro.