📡 MQTT + IA USCO – Guía de Réplica Local Cross-Broker

Esta guía paso a paso te permitirá poner en marcha **Mosquitto** local con un puente bidireccional a **EMQX Cloud**, y probar los simuladores Python en cualquier OS (Linux, macOS o Windows).

🛠️ Prerrequisitos

- Python 3.7+
- Mosquitto broker y clientes (`mosquitto`, `mosquitto_pub`, `mosquitto_sub`)
- Cuenta y credenciales EMQX Cloud

1️⃣ Instalar Mosquitto

Linux (Debian/Ubuntu)

```
sudo apt update
sudo apt install -y mosquitto mosquitto-clients
sudo systemctl enable mosquitto
sudo systemctl start mosquitto
```

macOS (Homebrew)

```
brew install mosquitto
brew services start mosquitto
```

Windows (Chocolatey)

```
choco install mosquitto
nssm install mosquitto "C:\Program Files\Mosquitto\mosquitto.exe"
nssm start mosquitto
```

2️⃣ Configurar el Puente Local → EMQX Cloud

Edita (o crea) el fichero de puente:

```
# Linux/macOS
sudo nano /etc/mosquitto/conf.d/emqx_bridge.conf

# macOS Homebrew
sudo nano $(brew --prefix)/etc/mosquitto/conf.d/emqx_bridge.conf
```

Pega el siguiente contenido:

```
##################################################################################
# Bridge local → EMQX Cloud (TLS en ```3)
##################################################################################
```

```
connection emqx-bridge
clientid          mosquitto-local-to-emqx

address   cbbd0c65.ala.dedicated.aws.emqxcloud.com:8883

remote_username  user123
remote_password  123456789

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

# OUT: reenviamos estos topics DEL LOCAL → NUBE
topic state/telemetry/#   out 0    ""    ""
topic result/#            out 0    ""    ""
topic alert/#             out 0    ""    ""

# IN: traemos estos topics DE LA NUBE → LOCAL
topic command/#           in  0    ""    ""
topic alert/#             in  0    ""    ""

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

Guarda y reinicia Mosquitto:

```
# Linux
sudo systemctl restart mosquitto

# macOS (Homebrew)
brew services restart mosquitto
```

3️⃣ Verificar el Puente

Publica un mensaje local y comprueba que aparece en EMQX Cloud:

```
mosquitto_pub -h localhost \
  -t alert/raspi1 -m "TEST_LOCAL"

# En EMQX Cloud Web UI: busca alerta en alert/raspi1
```

Publica en la nube y comprueba retransmisión local:

```
mosquitto_pub -h cbbd0c65.ala.dedicated.aws.emqxcloud.com \
  -p ```3 --cafile /opt/homebrew/etc/mosquitto/certs/ca.crt \
  -u user123 -P 123456789 \
  -t command/raspi1/light -m "ON"

# En tu Mosquitto local log (mosquitto -v) verás llegada de ON en command/raspi1/light
```

4️⃣ Simuladores Python

Coloca los tres scripts en una carpeta y, desde allí, instala la dependencia:

```
pip install paho-mqtt
```

4.1 `arduino_sim.py` – Simula Hardware

```
# arduino_sim.py
import time, json
import paho.mqtt.client as mqtt

BROKER = "localhost"; PORT = 1883; DEVICE = "raspi1"
TOPIC_TELE = f"state/telemetry/{DEVICE}"
TOPIC_CMD  = f"command/{DEVICE}/#"

def on_message(c, u, msg):
    t, v = msg.topic, msg.payload.decode()
    if t.endswith("/fan"):
        print(f"[ARDUINO] {v=} → {'FAN ON' if v=='ON' else 'FAN OFF'}")
    if t.endswith("/light"):
        print(f"[ARDUINO] {v=} → {'LIGHT ON' if v=='ON' else 'LIGHT OFF'}")

client = mqtt.Client(client_id="arduino_sim")
client.on_message = on_message
client.connect(BROKER, PORT); client.subscribe(TOPIC_CMD)
client.loop_start()

try:
    while True:
        temp = 35 + 5 * (time.time() % 1)
        payload = json.dumps({"temperature": round(temp,1), "device": DEVICE})
        client.publish(TOPIC_TELE, payload)
        print(f"[ARDUINO] Published → {TOPIC_TELE}: {payload}")
        time.sleep(5)
except KeyboardInterrupt:
    pass
finally:
    client.loop_stop(); client.disconnect()
```

4.2 `has_sim.py` – Simula Home Assistant Local

```
# has_sim.py
import json
import paho.mqtt.client as mqtt

BROKER="localhost"; PORT=1883; DEVICE="raspi1"; TH=30.0
TEL = f"state/telemetry/{DEVICE}"
CMD_F = f"command/{DEVICE}/fan"
AL = f"alert/{DEVICE}"

def on_connect(c, u, flags, rc):
    print("[HA] Connected", rc)
    c.subscribe(TEL)

def on_message(c, u, msg):
    data = json.loads(msg.payload.decode())
```

```
    temp = data.get("temperature", 0)
    print(f"[HA] temp={temp}°C")
    if temp > TH:
        c.publish(CMD_F, "ON")
        c.publish(AL, json.dumps({
            "device": DEVICE,
            "alert": "high_temperature",
            "temperature": temp
        }))
        print(f"[HA] Fan ON + Alert sent")

client = mqtt.Client(client_id="ha_sim")
client.on_connect = on_connect
client.on_message = on_message
client.connect(BROKER, PORT)
client.loop_forever()
```

4.3 `react_sim.py` – Simula React Native

```
# react_sim.py
import ssl
import paho.mqtt.client as mqtt

BROKER="cbbd0c65.ala.dedicated.aws.emqxcloud.com"
PORT=```3; USER="user123"; PASS="123456789"; TOP="alert/raspi1"
CA="/opt/homebrew/etc/mosquitto/certs/ca.crt"

def on_connect(c, u, flags, rc):
    print("[REACT] Connected", rc)
    if rc==0:
        c.subscribe(TOP)
        print(f"[REACT] Subscribed to {TOP}")
    else:
        print("[REACT] Connection failed")

def on_message(c, u, msg):
    print(f"[REACT] 🔔 {msg.topic}: {msg.payload.decode()}")

client = mqtt.Client(client_id="react_sim")
client.username_pw_set(USER, PASS)
client.tls_set(
    ca_certs=CA,
    cert_reqs=ssl.CERT_REQUIRED,
    tls_version=ssl.PROTOCOL_TLS_CLIENT
)
client.on_connect = on_connect
client.on_message = on_message
print("[REACT] Connecting…")
client.connect(BROKER, PORT)
client.loop_forever()
```

5️⃣ Ejecución y Prueba Final

1. **Arranca Mosquitto** con el puente activo.
2. En tres terminales, lanza:

- `python3 arduino_sim.py`
- `python3 has_sim.py`
- `python3 react_sim.py`

3. **Observa** cómo:
   - Arduino publica telemetría local.
   - HA local detecta alta temperatura, enciende el ventilador y emite alerta.
   - El bridge reenvía `alert/raspi1` a EMQX Cloud.
   - React simulado recibe la alerta desde la nube.

✅ Conclusión

Con estos pasos reproducirás tu arquitectura **MQTT local ↔ EMQX Cloud** en cualquier sistema operativo, validando el ciclo completo de publicación, puente, inferencia y notificación móvil.