CURSO 04 / 08 · PROGRAMA XDG

Casos prácticos de recolección y clientes MQTT.

DURACIÓN

10 horas · 2 días

PERFIL

Técnicos de campo · Backend · Arquitectos

MODALIDAD

Virtual · Pantalla compartida

DOCUMENTO

Apoyos visuales · Guía

DÓNDE ESTAMOS

Cuarta parada. MQTT, de cerca.

Este curso conecta la teoría del broker con una terminal en vivo. Lo que en el curso 3 era un diagrama, aquí se publica, se suscribe, se pierde, se reintenta.

C·01	C·02	C·03	C·04 · ahora	C·05	C·06	C·07	C·08
Fundamentos IoT industrial	Protocolos de campo	Arquitecturas de ingesta	Casos prácticos y clientes MQTT	Procesamiento y cola	Almacenamiento series	Visualización	Integración end-to-end

AL TERMINAR EL CURSO

Saldrás sabiendo hacer estas siete cosas.

No es un temario. Es una lista de capacidades concretas que podrás aplicar el lunes por la mañana, en producción.

01 **Diseñar una jerarquía de topics para una planta real**
Sin espacios, sin acentos, sin arrepentirte en tres meses.

02 **Elegir el QoS correcto para cada caso**
Y justificar por qué no es siempre el 2.

03 **Escribir un cliente productor Python production-ready**
Con TLS, reconexiones y presencia.

04 **Escribir un cliente consumidor idempotente**
Que no se rompa si el broker repite mensajes.

05 **Securizar un broker Mosquitto de extremo a extremo**
Transporte, autenticación, autorización.

06 **Simular carga y monitorizar el broker**
Con `$SYS`, Grafana y métricas que importan.

07 **Saber cuándo NO usar MQTT**
Lo más importante de la lista.

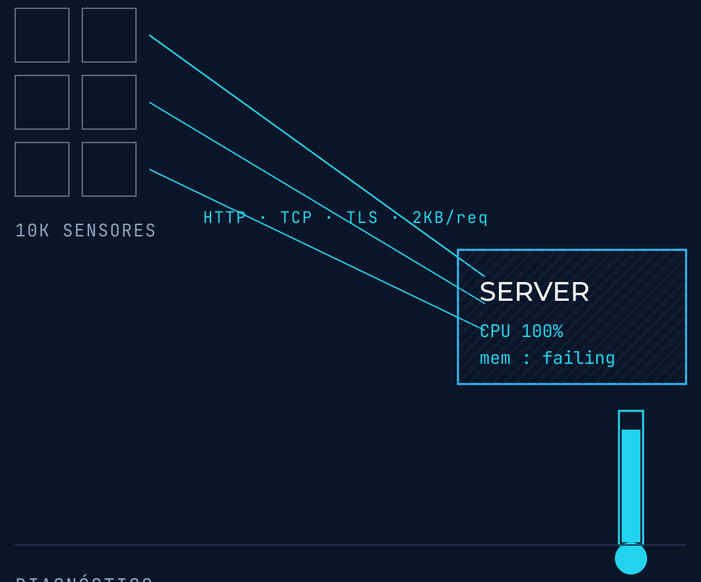
— BLOQUE I · FUNDAMENTOS

El protocolo, contado como lo entendería un técnico.

PUNTO DE PARTIDA

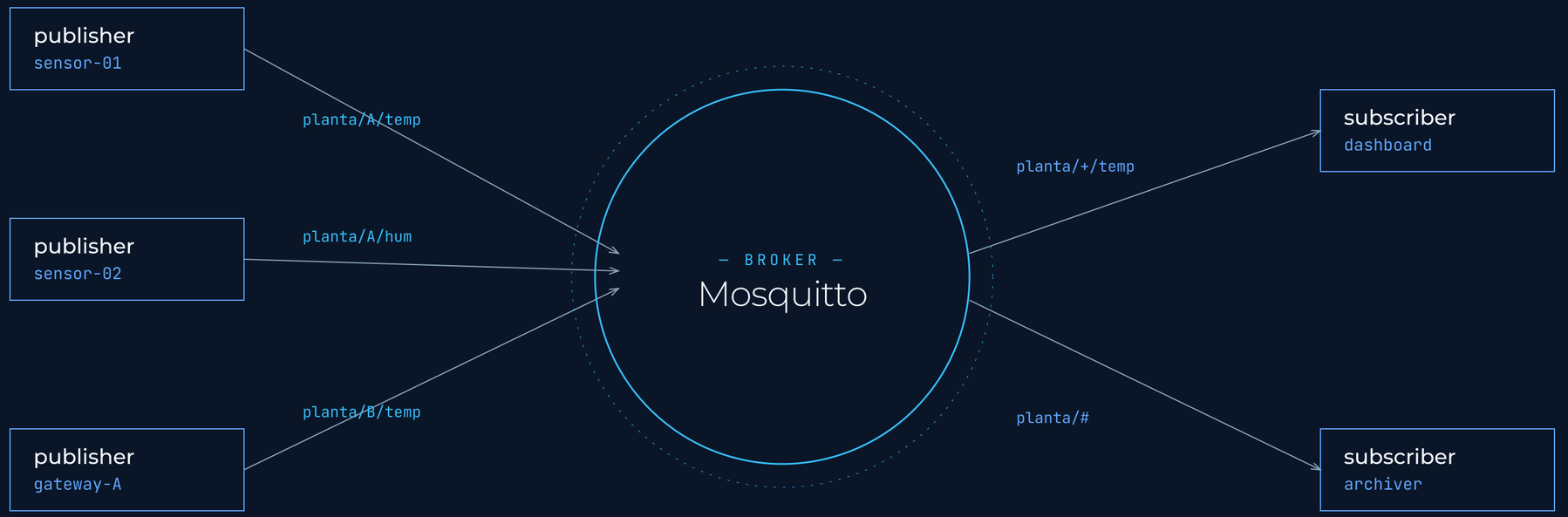
10.000 sensores × 1 msg / 5 s × HTTP = servidor muerto.

Handshake TCP + TLS + 2 KB por petición. Por sensor. Cada cinco segundos. Durante años.



DIAGNÓSTICO

HTTP no está diseñado para telemetría continua.


publisher
sensor-01
publisher
sensor-02
publisher
gateway-A
planta/A/temp
planta/A/hum
planta/B/temp
— BROKER —
Mosquitto
planta/+/temp
planta/#
subscriber
dashboard
subscriber
archiver

Dos formas de recoger datos.

SÍNCRONO · HTTP / REST

`cliente → espera → servidor`

`cliente ← espera ← servidor`

Cada mensaje es una conversación completa. El emisor reserva un hilo, abre un socket, negocia TLS y espera el `200 OK`. Multiplicado por diez mil, el servidor se asfixia.

ACOPLADO · BLOQUEANTE · PESADO

ASÍNCRONO · MQTT

`cliente → publica → broker`

`broker → reparte → suscriptores`

El emisor no espera a nadie. Suelta el paquete al broker y sigue. El broker decide quién lo recibe, cuándo, con qué garantía. Una sesión TCP larga, miles de mensajes cortos.

DESACOPLADO · LIGERO · PERSISTENTE

LA DIRECCIÓN DEL MENSAJE

Un topic es una ruta con sentido.

empresa/planta/zona/medida

WILDCARDS DE SUSCRIPCIÓN

+ un nivel cualquiera

`acme/+/norte/temp`

→ acme/planta-1/norte/temp

→ acme/planta-2/norte/temp

todo lo que cuelgue debajo

`acme/planta-1/#`

→ todos los topics de la planta

EJEMPLO REAL

`acme/planta-1/norte/temp`

`acme/planta-1/norte/humedad`

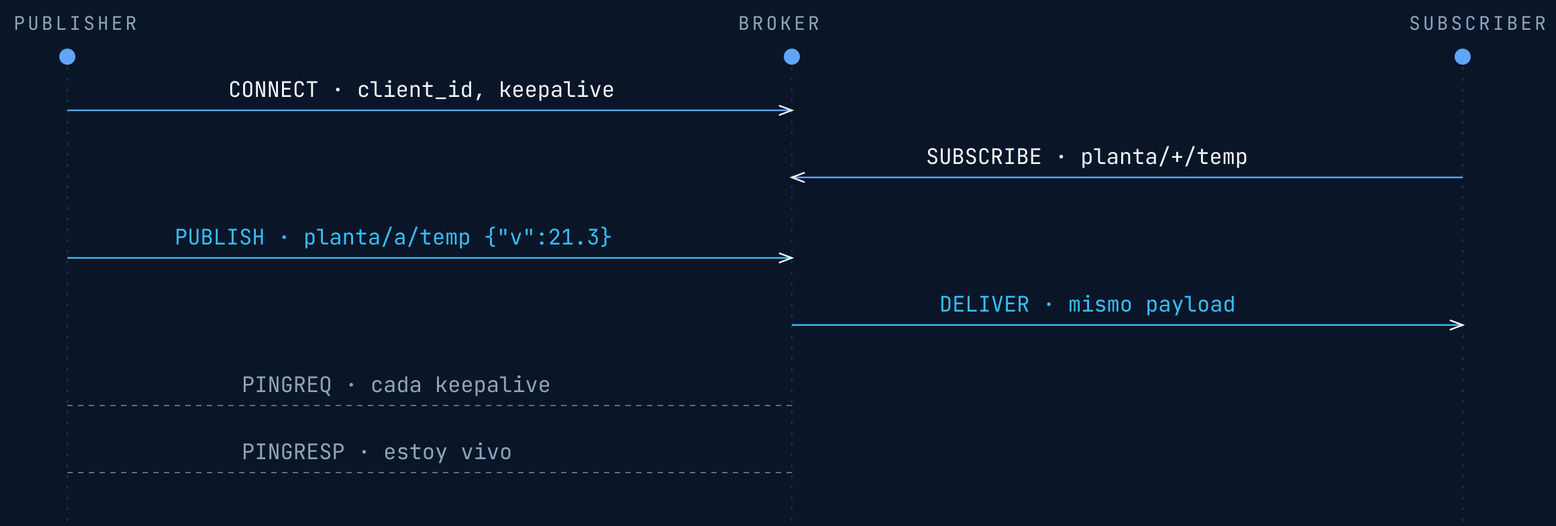
`acme/planta-1/sur/temp`

`acme/planta-1/sur/ruido-db`

`acme/planta-2/norte/temp`

La jerarquía es la API de tu flota. Una vez en producción, cambiarla cuesta semanas. Diséñala despacio.

Tres actores.
Una conversación.



— BLOQUE II · FIABILIDAD Y DISEÑO

Las garantías tienen precio.

QoS 0, 1, 2.

QoS·0

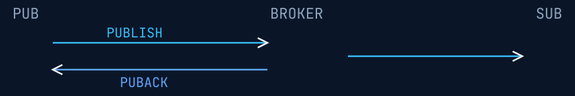
Fire and forget.

Puede perderse. Un mensaje, una red UDP-like.

QoS·1

Al menos una vez.

Puede duplicarse. El consumidor debe ser idempotente.



QoS·2

Exactamente una vez.

Cuatro mensajes por envío. Solo si lo necesitas de verdad.

REGLA POR DEFECTO → QoS 1. Siempre. Salvo que demuestres lo contrario.

RETAIN

El broker recuerda el último mensaje.

Cualquier cliente que se suscriba después lo recibe inmediatamente. Útil para estado: "último valor conocido", "configuración vigente".

PYTHON

```python
client.publish(
  "acme/v1/planta-1/estado",
  "online",
  retain=True
)
```

LAST WILL

Tu mensaje de despedida. Automático.

Lo declaras al conectar. Si te desconectas sucio (cable, corriente, kernel panic), el broker lo publica por ti. Patrón canónico de presencia.

PYTHON

```python
client.will_set(
  "acme/v1/planta-1/estado",
  "offline",
  retain=True
)
```

Seis reglas antes de escribir el primer topic.

01

Específico a general

✓ acme/planta-1/norte/temp

~~✗ temp/norte/planta-1/acme~~

02

Minúsculas, sin acentos

✓ edificio/a/sensor/temp

~~✗ Edificio/A/Sensór~~

03

Sin espacios, sin signos raros

✓ planta-3/sensor-01

~~✗ Planta 3/Sensor#01~~

04

Nunca uses + ni # en nombres

✓ planta/a/temp-max

~~✗ planta/a/temp+max~~

05

Separa datos de comandos

✓ acme/cmd/... acme/evt/...

~~✗ acme/todo/mezclado~~

06

Versiona la raíz

✓ acme/v1/planta-1/...

~~✗ acme/planta-1/...~~

ERRORES COMUNES

Lo que no debes hacer.

✗ **Meter el ID del dispositivo en el payload y no en el topic.**
No puedes filtrar. No puedes rutear. Todos escuchan todo.

✗ **Topics de 12 niveles con metadata de todo.**
Cuanto más largo el topic, más frágil la jerarquía. Cuatro o cinco niveles.

✗ **Usar QoS 2 "por si acaso".**
Cuatro mensajes por cada publish, overhead x4 en broker. Solo si es transaccional.

✗ **Un único cliente publicando por cien sensores.**
Pierdes trazabilidad. Si se cae, se caen cien. Un cliente por dispositivo físico.

✗ **No declarar Last Will.**
Tu dashboard nunca sabe si el sensor está muerto o simplemente callado.

✗ **Suscribirse con # desde la raíz.**
Recibes toda la flota. Un pequeño dashboard ahoga al broker en eventos.

Transporte, autenticación, autorización.

CAPA 01

Transporte

TLS 1.2 mínimo. Puerto 8883. Certificados del broker validados por el cliente. Si lo ves en el puerto 1883 en producción, tienes un problema.

```
mqtts://broker:8883
```

CAPA 02

Autenticación

Cada cliente tiene credenciales propias. Usuario/contraseña mínimo. Certificado mutuo si es crítico. Nunca credenciales compartidas entre dispositivos.

```
user: gw-planta-1-a72f
```

CAPA 03

Autorización

ACLs por cliente. Un sensor solo publica en su topic. Un dashboard solo lee. El principio: cada quien con el mínimo alcance.

```
topic write acme/planta-1/#
```

— BLOQUE III · CÓDIGO EN PRODUCCIÓN

De la teoría a la terminal. Escribimos clientes.

ANATOMÍA

Un productor production-ready en 14 líneas.

PUBLISHER.PY

```python
import paho.mqtt.client as mqtt

client = mqtt.Client(
    client_id="gw-planta-1-a72f",    # 1
    clean_session=False,             # 2
)

client.username_pw_set(user, pwd)
client.tls_set("ca.pem")             # 3
client.will_set(                      # 4
    "acme/v1/planta-1/estado",
    "offline", retain=True,
)

client.connect("broker", 8883)
client.loop_start()                   # 5

client.publish(
    "acme/v1/planta-1/norte/temp",
    payload='{"v":21.3}',
    qos=1,
)
```

① `client_id estable`
Para que el broker te reconozca tras una reconexión y mantenga tu sesión.

② `clean_session = False`
El broker guarda tu cola mientras no estás. Mensajes pendientes te esperan.

③ `tls_set()`
Siempre en producción. Sin excepciones.

④ `will_set()`
Tu mensaje de presencia automático si te caes sucio.

⑤ `loop_start()`
Hilo de fondo. Tu código principal no se bloquea.

ANATOMÍA

Un consumidor que aguanta duplicados por diseño.

CONSUMER.PY

```python
seen = LRUCache(maxsize=100_000)  # 1

def on_message(c, _, msg):
    data = json.loads(msg.payload)
    msg_id = data["id"]             # 2
    if msg_id in seen: return    # 3
    seen[msg_id] = True

    db.insert(                    # 4
        topic=msg.topic,
        ts=data["ts"],
        value=data["v"],
    )

client.on_message = on_message
client.subscribe(
    "acme/v1/+/+/temp",
    qos=1,
)
client.loop_forever()
```

① cache de vistos

En memoria o Redis. La clave del patrón idempotente.

② id único por mensaje

Lo añade el productor al payload. UUID, timestamp + secuencia, lo que sea estable.

③ dedupe

QoS 1 dice "al menos una vez". Tu código convierte eso en "exactamente una vez" aguas abajo.

④ escritura atómica

Si falla, el broker te reenvía en el siguiente intento. No pasa nada.

MQTT SOBRE WEBSOCKETS

El mismo broker, hablando en el navegador.

Puerto **9001** con MQTT-over-WS. La misma librería **mqtt.js** conecta desde un dashboard web. Un gráfico de Grafana o un panel React que se actualizan en tiempo real sin polling.

DASHBOARD.JS

```
const client = mqtt.connect(
  "wss://broker:9001"
);
client.subscribe("planta/+/+/temp");
client.on("message", render);
```

SIMULACIÓN DE CARGA

Miles de sensores en una sola máquina.

mqtt-bench, **emqtt-bench**, o un script propio con **asyncio**. Patrones: burst, steady, ramp. Mides latencia p99, tasa de PUBACK, memoria del broker.

BURST	10k msg en 1s ¿Aguanta picos? ¿Se desencola?
STEADY	50k msg/s sostenidos Memoria plana. Latencia estable.
RAMP	0 → 100k en 5 min ¿Dónde está el codo?

OBSERVABILIDAD DEL BROKER

Lo que tienes que mirar.

MÉTRICA	QUÉ MIDE	TOPIC $SYS
Clientes conectados	Sesiones activas. Si cae de golpe, mira la red.	`$SYS/broker/clients/connected`
Mensajes recibidos /s	Caudal de entrada. El ritmo de tu flota.	`$SYS/broker/load/messages/received/1min`
Mensajes pendientes	Cola acumulada. Si sube, hay consumidor lento.	`$SYS/broker/messages/inflight`
Bytes enviados /s	Ancho de banda. Útil para dimensionar tráfico TLS.	`$SYS/broker/load/bytes/sent/1min`
Uptime	Desde el último reinicio. Un clásico.	`$SYS/broker/uptime`

NO ES UNA BALA DE PLATA

MQTT no es la respuesta a todo.

NO LO USES SI...

01 **Necesitas request/response transaccional**
REST o gRPC, no MQTT.

02 **Los mensajes son grandes (>256 KB)**
Kafka, S3, HTTP multipart.

03 **El flujo es estrictamente punto a punto**
Un pub/sub con dos nodos es over-engineering.

04 **Necesitas ordering global estricto**
Kafka con una partición.

SÍ ES EL PROTOCOLO SI...

01 **Muchos emisores pequeños, pocos receptores**
La forma canónica del IoT industrial.

02 **Red poco fiable, ancho de banda limitado**
2G, LTE con cobertura irregular, mesh.

03 **Dispositivos con batería y poca CPU**
Sesiones persistentes, keepalive configurable.

04 **Necesitas desacoplar emisor y receptor**
Uno puede caer sin afectar al otro.

TRES IDEAS QUE TE LLEVAS

Si tuvieras que recordar solo tres cosas.

01 · DISEÑO

La jerarquía de topics es la API de tu flota.

Una vez en producción, cambiarla cuesta semanas. Piensa en frío, con lápiz, antes de escribir un `connect`.

02 · DEFAULTS

QoS 1, sesión persistente, Last Will, TLS.

Cuatro defaults que cubren el 95% de los casos. Cámbialos solo cuando sepas exactamente por qué.

03 · CONSUMIDOR

Idempotencia o nada.

Tu consumidor tiene que tolerar duplicados, reordenaciones y reenvíos. No es opcional, es la única forma sensata.

SIGUIENTE PARADA →

Curso 08 · FIWARE, Kafka y RabbitMQ.

GRACIAS

Preguntas al chat o al parking del Miro.