ESP32 enviando dados para um SCADA

O SCADA não quer o seu JSON: ele quer tags



Placa	ESP32 ou ESP32-S3 com Wi-Fi
Framework	Arduino Core, via PlatformIO
Transporte	MQTT 3.1.1 (PubSubClient)
Do outro lado	Mosquitto, e um SCADA ou cliente MQTT
Nível	Intermediário para avançado
Tutorial online	iiot.esp32.tec.br/tutoriais/esp32-enviando-dados-para-um-scada

O firmware deste pacote foi escrito a partir do funcionamento documentado do ESP32, do PubSubClient e do MQTT 3.1.1, e revisado linha a linha. **Ele não foi gravado em uma placa com um SCADA real conectado**, e a temperatura publicada é simulada por software. O objeto do material é o contrato de dados, não a leitura do sensor. Teste antes de usar em instalação que importe.

1. Publicar não é integrar

Fazer o ESP32 publicar num tópico MQTT leva dez minutos. Você chama publish, abre o mosquitto_sub do outro lado, vê o JSON chegando, e a sensação é de trabalho terminado. Aí o pessoal da automação pede para ligar aquilo no SCADA da planta, e a conversa muda de assunto por completo.

O SCADA não pensa em mensagens. Ele pensa em **tags**: um ponto de medição com nome estável, unidade, faixa, qualidade e histórico. Ele quer saber quando o valor foi lido, se dá para confiar nele, e quer distinguir um processo parado de um gateway morto. Nada disso vem de graça no publish.

Este manual trata das cinco decisões que decidem se a integração funciona: a modelagem dos tópicos, o carimbo de tempo, a qualidade do dado, o heartbeat e a retenção.

2. Modelagem de tópicos

Esta é a primeira decisão do projeto e a mais cara de mudar depois. Alterar a hierarquia com os equipamentos já em campo significa reconfigurar o SCADA e reprogramar todos os gateways, um por um. Pense nela antes de gravar o primeiro.

Do geral ao específico, um nível por dimensão:

```
planta1/utilidades/linha1/motor03/temperatura
planta1/utilidades/linha1/motor03/corrente
planta1/utilidades/linha1/gw-linha1-01/status
```

Três regras que resolvem quase tudo:

Uma grandeza por tópico. Tentador juntar tudo num JSON só, mas o SCADA assina tag a tag, e quem quiser só a corrente não deveria receber a temperatura junto.

O valor nunca entra no nome. Tópico é endereço, não dado. `.../motor03/temperatura/78` cria um tópico novo a cada leitura e entope o broker.

Identificadores estáveis. `motor03` continua sendo 03 quando trocarem o modelo do motor. Nunca use o número de série do equipamento no caminho: ele muda na primeira manutenção.

Sem acento, sem espaço, tudo minúsculo. O MQTT aceita, mas metade das ferramentas do outro lado não.

3. O horário é o da leitura

O erro clássico: o SCADA carimba o dado na hora em que a mensagem chega. Enquanto a rede está boa, ninguém percebe a diferença. No dia em que o link engasga por três minutos, chegam de uma vez cento e oitenta leituras carimbadas com o mesmo segundo, e o histórico daquele período vira uma linha vertical que não aconteceu.

Pior ainda quando há buffer local: o gateway guardou meia hora de dados e despeja tudo na reconexão. Sem carimbo de origem, meia hora de processo colapsa em dois segundos de gráfico.

Quem sabe a hora da leitura é quem leu. O carimbo tem que sair do gateway, dentro da carga:

```
{"t":1755990000,"v":78.42,"q":192,"u":"C"}
```

Campos curtos de propósito: a carga se repete milhares de vezes por dia e cada byte multiplica. `t` é o instante da leitura em epoch de segundos, sempre UTC - fuso é problema de quem exibe, não de quem mede. `v` é o valor, `q` a qualidade e `u` a unidade.

Para isso o ESP32 precisa de relógio, que ele não tem ao ligar. Sincronize NTP antes de publicar. E se o NTP falhar, **não invente horário**: publique qualidade incerta e deixe o outro lado decidir. O firmware do pacote faz exatamente isso.

4. Qualidade: zero é um valor válido

Se o sensor não respondeu e você publica zero, o SCADA mostra zero grau e o operador acredita. Não há nada na mensagem que distinga *a temperatura é zero* de *eu não consegui ler*. O campo de qualidade é o que separa as duas coisas.

A convenção vem do OPC e o SCADA já entende:

Valor	Significado	Quando usar
192	Boa	leitura normal, dentro do esperado
64	Incerta	valor antigo, relógio sem NTP, sensor fora de faixa
0	Ruim	falha de leitura, sensor sem resposta, barramento mudo

Publicar sempre 192 é o mesmo que não ter o campo. O valor está em ele **mudar**. A regra prática: toda vez que o seu código tiver um caminho de erro que ainda assim publica alguma coisa, esse caminho tem que baixar a qualidade.

Para testar, o firmware aceita o comando `ruim` pelo monitor serial: ele passa a publicar valor zero com qualidade zero. Confira no seu SCADA que aparece falha de leitura, e não zero grau. Um SCADA que ignora o campo engana o operador - é melhor descobrir isso na bancada.

5. Heartbeat: parado não é morto

Um valor que não muda há dez minutos pode significar duas coisas opostas: processo estável, ou gateway travado. Do lado do SCADA, as duas se parecem exatamente iguais.

A solução tem duas metades. A primeira é o **Last Will**, que o broker publica por você quando a conexão cai sem despedida:

```
mqtt.connect(GATEWAY_ID, MQTT_USUARIO, MQTT_SENHA,
topicoStatus.c_str(), 1, true,
"{\"online\":false}");
```

A segunda é o heartbeat periódico, que o firmware publica a cada trinta segundos com o instante da publicação (`t`), uptime, RSSI e versão de firmware. O Last Will cobre a morte subita; o heartbeat cobre o gateway que continua conectado mas parou de ler. Os dois vão **retidos e no mesmo tópico de status**: assim o Last Will sobrescreve o último "online" que ficou guardado no broker.

Do lado do SCADA, configure um alarme por **falta de atualização**: se o `t` do status ficar mais velho que dois minutos, o gateway sumiu. Alarme por "tópico ausente" não serve, porque com retenção o broker devolve o último status mesmo com o gateway morto. É a configuração que mais se esquece de fazer.

6. Retenção: estado sim, comando nunca

Mensagem retida é a que o broker guarda e entrega de imediato a quem se inscrever depois. Sem ela, o SCADA que reinicia às três da manhã fica com a tela vazia até a próxima publicação de cada tag.

Retenha valor de processo, porque o último valor conhecido tem significado, e status do gateway, que é estado por definição.

Não retenha alarme momentaneo, porque reentregar alarme velho confunde quem opera, nem comando.

O caso do comando merece parágrafo próprio, porque é o modo padrão de errar. Um comando publicado com retenção é reentregue a cada reconexão do cliente - e reexecuta a ação. Você liga o motor pelo SCADA às dez da manhã; às três da tarde o gateway reconecta depois de uma queda e recebe o mesmo comando de novo. O motor liga sozinho. Isso não é hipótese de manual: é o que acontece.

Se você só vai lembrar de uma linha deste manual, que seja esta: **comando nunca é retido**.

7. Passo a passo

1. Abra a pasta do pacote no VS Code com o PlatformIO instalado.

2. Edite `include/config.h`: rede, broker e - com calma - a hierarquia de tópicos.

3. Compile e grave com PlatformIO: Upload.

4. Abra o monitor serial a 115200. O firmware imprime os três tópicos que vai usar antes de conectar - confira se são os que você quer, antes de o SCADA ser configurado em cima deles.

5. Do outro lado, acompanhe:

```
mosquitto_sub -h 192.168.1.100 -u gateway -P senha -v -t 'planta1/#'
```

6. Digite `ruim` no monitor e confira a reação do SCADA. Depois `bom` para voltar. Esse teste vale mais que uma semana de dado bonito.

8. Segurança, é o que ela custa

MQTT sem TLS trafega usuário, senha e todo o dado em texto claro. Dentro de uma rede industrial isolada, muita gente aceita o risco. Saindo dela, não há discussão.

TLS no ESP32 tem preço em RAM: o handshake sozinho consome dezenas de kilobytes, e o certificado da autoridade fica na flash. Em um ESP32 clássico com firmware já carregado, é comum o handshake falhar por falta de heap - o sintoma é uma conexão que cai sempre no mesmo ponto. Placa com PSRAM resolve; reduzir o firmware também.

E TLS exige relógio correto: o ESP32 rejeita certificado válido se acha que é 1970. Mais um motivo para o NTP vir antes de tudo.

9. Quando o próximo passo é Sparkplug B

Tudo aqui é MQTT cru com um contrato de dados combinado entre você e o SCADA. Funciona bem e cabe na cabeça. A partir de algumas dezenas de gateways, o contrato informal começa a divergir: cada firmware carimba de um jeito, cada integração reinventa a qualidade.

O Sparkplug B padroniza justamente isso - modelagem, nascimento e morte de dispositivo, tipos e qualidade - ao custo de payload binário e de ferramental próprio. Não comece por ele; saiba que existe para quando o número de gateways justificar.

10. Checklist antes de entregar

[ ]	Hierarquia de tópicos definida e revisada com quem opera
[ ]	Uma grandeza por tópico; nenhum valor dentro do nome
[ ]	Identificadores estáveis, sem número de série no caminho
[ ]	NTP sincronizado antes da primeira publicação
[ ]	Carimbo de tempo da leitura, em UTC, dentro da carga
[ ]	Sem NTP: qualidade incerta, e nenhum horário inventado
[ ]	Campo de qualidade presente e que realmente muda
[ ]	Todo caminho de erro que publica baixa a qualidade
[ ]	Teste do comando 'ruim' conferido no SCADA de destino
[ ]	Last Will configurado no connect
[ ]	Heartbeat periódico publicado
[ ]	Alarme por ausência do heartbeat criado no SCADA
[ ]	Valor de processo e status retidos
[ ]	Nenhum tópico de comando com flag de retenção
[ ]	Client id único por gateway no broker
[ ]	TLS avaliado, e heap medido se estiver ligado