Accessa – Atividade 3: Implementação de IoT (MQTT)
Data: 26/10/2025

> Documento único do grupo contendo broker escolhido, configuração básica, estrutura de tópicos, elementos de IoT e link público do canal MQTT (quando aplicável).

---

1. Broker escolhido e justificativa

Broker: **HiveMQ Cloud (Serverless Free)**
Justificativa resumida:
- Plano gratuito suficiente para PDI/Prototipagem (≈100 conexões, limites adequados).
- **TLS** nativo (segurança), suporte a **MQTT 3.1.1/5.0** e **WebSockets** (testes via navegador).
- Painel SaaS simples de operar (criação de usuários, rotacionar senhas, observar métricas básicas).
- Integração fácil com clientes **ESP8266** e **aplicativo Flutter**.

> Alternativa equivalente: **EMQX Cloud Serverless** (similar em recursos) ou **Mosquitto self-hosted** (em VM/RPi).
> Se a disciplina exigir canal/URL público específico, substituir por **MyQttHub.com** ou **ThingSpeak** (com restrições).

---

2. Configuração inicial do cluster (HiveMQ Cloud)

1. Criar conta e **Cluster Serverless**.
2. Anotar:
   - **Endpoint MQTT (TLS):** `xxxxxxxx.s1.eu.hivemq.cloud:8883`
   - **Endpoint WebSocket (TLS):** `xxxxxxxx.s1.eu.hivemq.cloud:8884`
   - **Usuário MQTT:** `accessa_app`
   - **Senha:** `********` (gire periodicamente)
3. Baixar **CA Root** (se necessário) para validação TLS no firmware.
4. Criar **usuários separados por função** (princípio de menor privilégio):
   - `device_<deviceId>` (conecta com LWT e publica em `evt/state/tele`, assina `cmd`)
   - `app_<user>` (publica `cmd`, assina `evt` do(s) dispositivo(s) autorizados)
5. (Opcional) **ACL por padrão de tópico** no broker (quando suportado).

Campos a preencher pelo grupo (caso já criado):
- Painel/Console: `<URL do Console>`
- Endpoint MQTT (TLS/8883): `<endpoint>`
- Endpoint WS (TLS/8884): `<endpoint>`
- Usuário(s) MQTT: `<lista>`
- **Link público** do canal/teste (quando aplicável): `<URL>`

---

3. Estrutura de tópicos MQTT (Accessa)

Convencionar **namespace base**: `accessa/`

3.1. Comando (App → Dispositivo)
- Tópico: `accessa/<deviceId>/cmd`
- **QoS:** 1
- **Retain:** `false`
- **Payload (JSON):**
```json
{
  "requestId": "uuid-v4",
  "deviceId": "<deviceId>",
  "userId": "<uid>",
  "timestamp": 1730000000,
  "nonce": "base64-rand-12b",
  "hmac": "hex(HMAC-SHA256(nonce+timestamp+deviceId+userId, key_dev))",
  "action": "unlock",
  "durationMs": 5000
}
```

3.2. Evento/Resultado (Dispositivo → App/API)
- Tópico: `accessa/<deviceId>/evt`
- **QoS:** 1
- **Retain:** `false`
- **Payload (JSON):**
```json
{
  "requestId": "uuid-v4",
  "deviceId": "<deviceId>",
  "userId": "<uid>",
  "ts": 1730000001,
  "result": "success|denied|invalid_token|timeout|tamper",
  "doorSensor": "open|closed",
  "hash": "sha256(event_body)"
}
```

3.3. Estado (online/offline) e LWT
- Tópico online: `accessa/<deviceId>/state` (publish `online` ao conectar, **retain=true**)
- **LWT:** `accessa/<deviceId>/state` = `offline` (**retain=true**, QoS 1)

3.4. Telemetria
- Tópico: `accessa/<deviceId>/tele` (**retain=false**, QoS 0/1)
- Campos: `uptime`, `rssi`, `fw`, `heap`, etc.

3.5. Administração (opcional)
- `accessa/<deviceId>/admin/rotate_key`
- `accessa/<deviceId>/admin/lockdown`

Curingas úteis (para assinaturas de admin):
- `accessa/+/evt`, `accessa/+/state`, `accessa/+/tele`

---

4. Elementos de IoT (sensores/atuadores)

Microcontrolador: ESP8266 (NodeMCU/Wemos D1 Mini)
Atuador: Relé (fechadura elétrica/solenóide)
Sensores: Reed switch (porta aberta/fechada)
Sinalização: LED RGB e Buzzer
Fallback offline (futuro): PIN temporário local

Mapeamento (exemplo):
- D1 → Relé (ativo BAIXO)
- D2 → Reed switch (pull-up interno)
- D5/D6/D7 → LED RGB (ou 1 LED + buzzer)

---

5. Políticas de QoS, Retain e Segurança

- **QoS**: `cmd` e `evt` com **1**; `tele` pode ser **0**.
- **Retain**: apenas para `state` (online/offline).
- **TLS** obrigatório; use CA válida para o cluster.
- **Segregação de credenciais** por dispositivo/usuário; **rotacione senhas**.
- **HMAC** por dispositivo (chave simétrica) para evitar replay e spoofing.
- **Janela temporal**: ±30s (validar `timestamp` no firmware).

---

6. Exemplos práticos de teste

6.1. mosquitto_sub / mosquitto_pub (substitua `<endpoint>`, `<user>`, `<pass>`)
```bash
# Assinar eventos e estado do device LAB01
mosquitto_sub -h <endpoint> -p 8883 --cafile ./ca.crt -t "accessa/LAB01/#" -u <user> -P <pass>
-q 1
```

```
# Enviar comando de destravar (5s) para LAB01
mosquitto_pub -h <endpoint> -p 8883 --cafile ./ca.crt -t "accessa/LAB01/cmd" -m '{"requestId":"
d1","deviceId":"LAB01","userId":"hagliberto","timestamp":1730000000,"nonce":"abc","hmac":"<calc
>","action":"unlock","durationMs":5000}' -u <user> -P <pass> -q 1
```

6.2. WebSocket (navegador) — quando habilitado
- Host: `<endpoint>` Porta: **8884** (TLS) Caminho: padrão do serviço
- Útil para **debug** de tópicos via ferramentas web.

---

7. Integração mínima (Flutter + pacote mqtt_client)

Pseudocódigo (conexão segura e publish `cmd`):
```dart
final client = MqttServerClient.withPort('<endpoint>', 'app_hagliberto_01', 8883)
  ..secure = true
  ..logging(on: false)
  ..keepAlivePeriod = 30
  ..onDisconnected = () { /* atualizar estado UI */ };

final connMess = MqttConnectMessage()
  .withClientIdentifier('app_hagliberto_01')
  .startClean()
  .withWillTopic('accessa/LAB01/state')
  .withWillMessage('offline')
  .withWillQos(MqttQos.atLeastOnce);

client.connectionMessage = connMess;

// TODO: configurar SecurityContext com CA se necessário (Web usa WSS)

await client.connect('<user>', '<pass>');

// Assinar eventos do device
client.subscribe('accessa/LAB01/evt', MqttQos.atLeastOnce);

// Publicar comando (JSON codificado em UTF-8)
final payload = jsonEncode({
  'requestId': 'uuid',
  'deviceId': 'LAB01',
  'userId': 'hagliberto',
  'timestamp': DateTime.now().millisecondsSinceEpoch ~/ 1000,
  'nonce': 'rand',
```

```
  'hmac': '<calc>',
  'action': 'unlock',
  'durationMs': 5000,
});
final builder = MqttClientPayloadBuilder()..addUTF8String(payload);
client.publishMessage('accessa/LAB01/cmd', MqttQos.atLeastOnce, builder.payload!);
```

---

8. Link público e participantes

- **Link público / Console / Canal:** `<cole aqui o link compartilhável>`
- **Membros que participaram desta etapa:**
  - Nome 1 — atividade/descrição
  - Nome 2 — atividade/descrição
  - Nome 3 — atividade/descrição

---

9. Checklist de aceitação

- [ ] Broker criado, com TLS e usuário(s) específicos por função
- [ ] Tópicos definidos: `cmd`, `evt`, `state`, `tele`, `admin`
- [ ] Dispositivo simulado/publicando `state` (`online`) com retain
- [ ] Teste de comando `unlock` e recebimento de `evt` (success/denied)
- [ ] Link público (quando aplicável) incluído no documento

---

10. Apêndice A — Firmware ESP8266 (esqueleto com LWT)

> **Observação:** usar **WiFiClientSecure** e validar tempo via NTP.

```cpp
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>

const char* ssid = "WIFI";
const char* pass = "SENHA";
const char* mqttHost = "<endpoint>";
const int   mqttPort = 8883;
const char* mqttUser = "device_LAB01";
const char* mqttPass = "********";
const char* deviceId = "LAB01";
```

```
WiFiClientSecure wifi;
PubSubClient mqtt(wifi);

void onMsg(char* topic, byte* payload, unsigned int len) {
  // TODO: validar HMAC/timestamp e acionar relé/LED/Buzzer
}

void setup() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) delay(500);

  // TLS (opcional: wifi.setFingerprint / setCACert)
  wifi.setInsecure(); // DEMO: não use em produção

  mqtt.setServer(mqttHost, mqttPort);
  mqtt.setCallback(onMsg);
}

void loop() {
  if (!mqtt.connected()) {
    String clientId = String("dev_") + deviceId;
    // LWT: offline
    mqtt.connect(clientId.c_str(), mqttUser, mqttPass,
("accessa/"+String(deviceId)+"/state").c_str(), 1, true, "offline");
    if (mqtt.connected()) {
      // online + retain
      mqtt.publish(("accessa/"+String(deviceId)+"/state").c_str(), "online", true);
      mqtt.subscribe(("accessa/"+String(deviceId)+"/cmd").c_str(), 1);
    }
  }
  mqtt.loop();
  delay(10);
}
```

Observações finais
- Para produção, **não** use `setInsecure()`. Valide CA e hostname.
- Calcule HMAC no app/firmware; rejeite `timestamp` fora da janela.
- Loguem os eventos e façam **rotação de chaves** periodicamente.