Sidewalk-Rust-Flutter
IoT プロトタイプ構築ガイド

第1.0版

2026.01.01

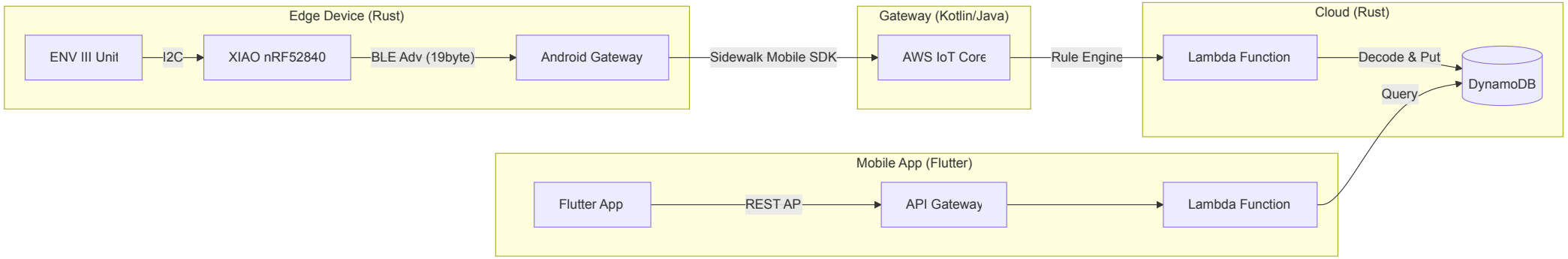
合同会社ヴォール

目次

1 プロジェクト概要

本プロジェクトは、Amazon Sidewalk の広域ネットワーク特性を模倣した BLE 通信環境を構築し、温度データをクラウドで収集・可視化するシステムです。

1.1 システム構成図



1.2 ハードウェア要件 (BOM)

1. **MCU:** Seeed Studio XIAO nRF52840 (Sense または Standard)
2. **Base:** Seeed Studio XIAO Expansion Base
3. **Sensor:** M5Stack ENV III Unit (SHT30 + QMP6988)
4. **Gateway:** Android スマートフォン (Amazon Sidewalk Mobile SDK 対応)
5. **PC:** Rust 開発環境 (VS Code 推奨)

2 データ構造設計 (Payload Protocol)

LoRa (Sub-GHz) のペイロード制限（最大19byte）に適合するよう、`postcard` クレートを用いてバイナリ化します。

データレイアウト (Total: 11 bytes)

Byte Offset	Field	Type (Rust)	Description
0-3	`device_id`	`u32`	デバイス固有ID
4-5	`temperature`	`i16`	温度 (℃) × 100
6	`humidity`	`u8`	湿度 (%)
7-10	`pressure`	`u32`	気圧 (Pa)

*残り*8byte*は、将来的にタイムスタンプやステータスフラグに使用可能。*

3 実装フェーズ

3.1 Phase 1: AWS クラウド基盤構築 (Rust Backend)

データを受け皿を先に作ります。

手順:

1. **DynamoDB テーブル作成:**
   - Table Name: `SidewalkTemperatures`
   - Partition Key: `device_id` (String)
   - Sort Key: `timestamp` (Number)
2. **Rust Lambda (Receiver) 実装:**
   - Sidewalk から届く Base64 エンコードされたバイナリをデコードし、DBへ保存します。

```
// Cargo.toml
// dependencies: lambda_runtime, serde, postcard, aws-sdk-dynamodb, base64

#[derive(Deserialize, Serialize)]
struct Payload {
    device_id: u32,
    temperature: i16, // real_temp = temperature / 100.0
    humidity: u8,
    pressure: u32,
}

async fn function_handler(event: LambdaEvent<IoTEvent>) -> Result<(), Error> {
    let payload_bytes = base64::decode(&event.payload.data)?;
    // 19byte以下のバイナリを構造体に復元
    let data: Payload = postcard::from_bytes(&payload_bytes)?;

    // DynamoDBへ保存 (aws-sdk-dynamodb使用)
    client.put_item()
        .table_name("SidewalkTemperatures")
        .item("device_id", AttributeValue::S(data.device_id.to_string()))
        .item("timestamp", AttributeValue::N(Utc::now().timestamp().to_string()))
        .item("temperature", AttributeValue::N((data.temperature as f64 / 100.0).to_string
()))
        .send().await?;
```

```
    Ok(())
}
```

3.2 Phase 2: エッジデバイス開発 (XIAO nRF52840)

センサー値を読み取り、BLEアドバタイズパケットに乗せます。

I2C 接続情報:

- **SDA:** Pin D4
- **SCL:** Pin D5
- **ENV III Address:** `0x44` (SHT30), `0x70` (QMP6988)

Rust 実装のポイント:

`embassy-nrf` (非同期ランタイム) を使用し、省電力動作を実装します。

```
// main.rs (抜粋)

#[embassy_executor::main]
async fn main(spawner: Spawner) {
    let p = embassy_nrf::init(Default::default());

    // I2C 初期化 (Expansion Baseの仕様に合わせる)
    let config = twim::Config::default();
    let i2c = twim::Twim::new(p.TWIM0, p.P1_08, p.P0_07, config); // ピン番号は要確認(XIAO D4/
D5)

    // センサー初期化
    let mut sht30 = sht3x::Sht3x::new(i2c, sht3x::Address::Low);

    loop {
        // 1. 計測
        let measurement = sht30.measure().await.unwrap();
        let temp_int = (measurement.temperature.as_degrees_celsius() * 100.0) as i16;

        // 2. パッキング (postcard)
        let payload = Payload {
            device_id: 1001,
            temperature: temp_int,
            humidity: measurement.humidity.as_percent() as u8,
```

```
            pressure: 0, // 今回は省略
        };
        let mut buf = [0u8; 19];
        let bytes = postcard::to_slice(&payload, &mut buf).unwrap();

        // 3. 送信 (BLE Advertise)
        // Sidewalk Mobile SDKが検知できる特定のUUIDまたはManufacturer Dataに設定
        let mut adv_data = LegacyAdvertisement::new();
        adv_data.manufacturer_specific_data(0xFFFF, bytes); // テスト用ID

        // アドバタイズ実行（例えば5秒間）
        advertiser.advertise(&adv_data).await;

        // 4. Deep Sleep（例えば10分）
        Timer::after(Duration::from_secs(600)).await;
    }
}
```

3.3 Phase 3: ゲートウェイ構築 (Android)

スマートフォンをブリッジとして機能させます。

手順:

1. **Sidewalk Mobile SDK 導入:**
   - Amazon Developer Portal から SDK を入手し、Android Studio プロジェクトにインポート。
2. **ブリッジ機能の実装:**
   - `SidewalkManager` を初期化。
   - Bluetooth スキャンを実行し、XIAO からのパケットをキャッチ。
   - SDK のメソッド `sendData()` を使用して、受信したペイロードをそのまま AWS へ転送。

> **Note:** 開発段階では、単なるBLEスキャナーアプリを作り、受信データを「AWS IoT CoreのHTTPエンドポイント」にPOSTする簡易実装でも代用可能です（Sidewalk SDKの認証周りが複雑なため）。

3.4 Phase 4: 可視化アプリ開発 (Flutter)

ユーザーがデータを見るためのダッシュボードです。

技術スタック:

- **State Management:** Riverpod
- **Chart:** `fl_chart`
- **Networking:** `dio` or `http`

実装フロー:

1. **API クライアント:**

   Phase 1 で作成したデータを読み出す Lambda (API Gateway経由) を叩くリポジトリクラスを作成。
2. **UI 構築:**

```
// Chart Widget 簡易例
LineChart(
  LineChartData(
    lineBarsData: [
      LineChartBarData(
        spots: points.map((e) => FlSpot(e.time, e.temp)).toList(),
        isCurved: true,
        color: Colors.blue,
      ),
    ],
    // ...軸の設定など
  ),
);
```

4 開発ロードマップとチェックリスト

- [ ] **Rust環境構築:** `rustup target add thumbv7em-none-eabihf`, `cargo install probe-rs`.
- [ ] **ハードウェアテスト:** XIAO + Expansion Base で OLED に "Hello" を表示する。
- [ ] **センサー疎通:** Rust で I2C 経由の温度取得を成功させる。
- [ ] **クラウド疎通:** テスト用バイナリデータを AWS IoT Core のテスト画面から投げ、DynamoDB に入ることを確認する。
- [ ] **エンドツーエンド (BLE):** デバイス → Android → AWS のパスを通す。
- [ ] **アプリ表示:** Flutter アプリでグラフが描画されることを確認する。

5 Tips & トラブルシューティング

- **XIAO nRF52840 ピン配置:** Rust の `bsp` (Board Support Package) クレートを使う場合、`D4`, `D5` という名前でアクセスできるか、`P0_07` などの生ピン番号が必要かを確認してください。XIAO の回路図と照らし合わせるのが確実です。
- **エンディアン:** `postcard` はリトルエンディアンを使用します。デバッグ時にバイナリを手動で読む際は注意してください。
- **権限:** Android アプリには `BLUETOOTH_SCAN`, `BLUETOOTH_CONNECT`, `ACCESS_FINE_LOCATION` の権限付与が必要です。