IoT Innovation Design Department

**IoT イノベーションデザイン学科新設のためのカリキュラム開発事業**

# 教材成果報告書

## － IoT 基礎教材(クラウド編) －

令和７年度

本報告書は、文部科学省の教育政策推進事業委託費による委託事業として、九州技術教育専門学校が実施した令和７年度「地方やデジタル分野における専修学校理系転換等推進事業」の成果をとりまとめたものです。

文科省委託「地方やデジタル分野における専修学校理系転換等推進事業」

KTEC 学校法人赤山学園 九州技術教育専門学校

# 目次

教材の概要.....1

教材の目的.....1

教材の内容.....2

　教材の対象.....2

　教材の構成.....2

教材の成果.....2

　成果物一覧.....2

　講座で使用したテキスト.....3

今後の課題.....10

# IoT 基礎教材(クラウド側)成果報告

## 教材の概要

本教材「IoT 基礎教材(クラウド編)」は、SORACOM と AWS IoT Core を活用したクラウド IoT システムの構築・運用方法を学ぶ実践的な教科書である。スマート民泊管理システム「IoT Stay」の開発を題材とし、実際のビジネス課題（物件巡回の負担、環境把握の遅れ、エネルギーの無駄、水道使用量の管理）を IoT 技術で解決する体験を提供する。学生は各自が 1 つの客室（デバイス）を担当し、クラス全体で 20 客室を管理する統合システムを構築する。これにより、チーム開発と IoT システム運用の実践的な経験を得られる。

## 教材の目的

本教材の目的は以下のとおりである。

1. IoT の基本概念の習得: デバイスからクラウドへのデータ送信（アップリンク）、クラウドからデバイスへの制御（ダウンリンク）の仕組みを理解する

2. クラウドサービスの実践的活用: SORACOM（Arc、Harvest）、AWS IoT Core など実際の商用サービスを使用した開発を体験する

3. 通信プロトコルの理解: HTTP 通信（データ送信向け）と MQTT 通信（リアルタイム制御向け）の特性と使い分けを学ぶ

4.データ可視化技術の習得: Streamlit、Pandas、Plotly を使用したダッシュボード開発を通じて、データ可視化の基礎を身につける

5.実務スキルの獲得: 実際の IoT 現場でも通用する技術スタックと設計パターンを習得する

# 教材の内容

## 教材の対象

前提知識:

- Python の基本文法（変数、関数、リスト、辞書）
- コマンドライン操作の基礎
- ネットワークの基本概念（IP アドレス、ポートなど）

前提知識に欠ける場合でもテキスト内で適宜説明し、理解できるように構成する。

## 教材の構成

- 第 1 章: これから作るもの - プロジェクト全体像、民泊の課題と IoT 解決策
- 第 2 章: クラウドと IoT の基礎 - クラウドの本質、IoT×クラウドの価値
- 第 3 章: システム全体像とセットアップ - アーキテクチャ理解、SORACOM Arc 設定
- 第 4 章: 開発環境のセットアップ - Git、Python 環境、プロジェクト構成
- 第 5 章: アップリンク機能の実装 - HTTP 通信、センサーデータ送信、Harvest 可視化
- 第 6 章: ダウンリンク機能の実装 - MQTT 通信(Pub/Sub)、AWS IoT Core、LED
- 第 7 章: Streamlit ダッシュボード基礎 - 基本 UI、フィルタリング、KPI 表示
- 第 8 章: データ分析とグラフ化 - Pandas、Plotly、高度な可視化
- 第 9 章: 水量監視とデータ分析 -水量センサー連携、異常検知、バルブ制御
- 付録 A: AI 支援開発ガイド - GitHub Copilot、Claude Code 活用法

# 教材の成果

## 成果物一覧

- 教材（PDF/HTML 形式）
- ソースコード成果物

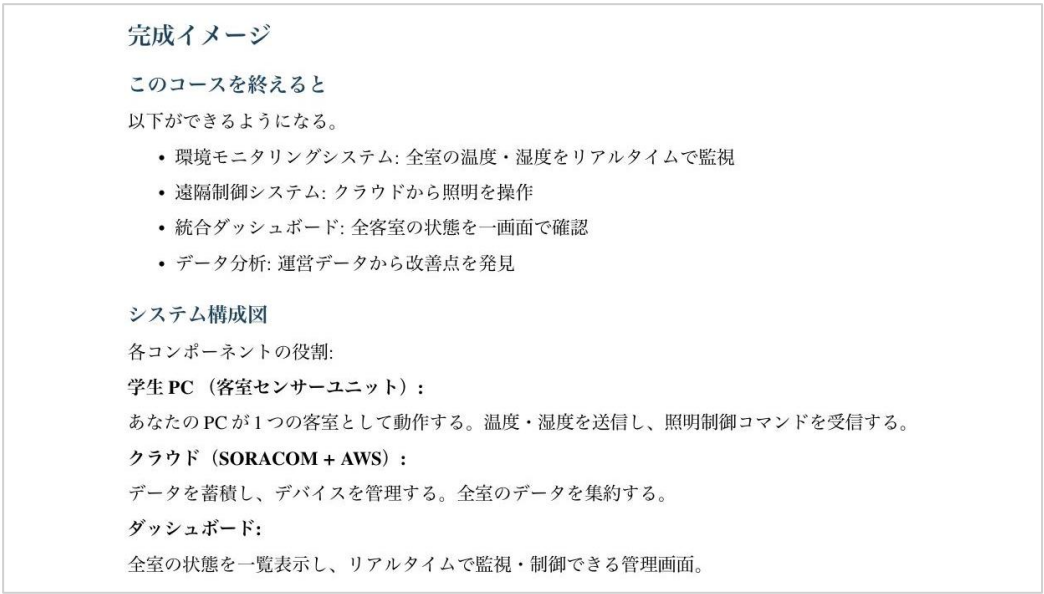
- 学生用テンプレート
  - ✧ デバイス関連：app.py, app_with_mqtt.py, app_water.py, config.py, device_controller.py, payload_builder.py
  - ✧ ダッシュボード関連：app_basic.py, app_advanced.py, app_water.py, utils/data_processor.py, utils/soracom_api.py
- 模範解答
  - ✧ 学生用テンプレートと同一構成の完成版コード。演習の解答確認および教師用リファレンスとして使用。

- 教師用管理ツール
  - 学生アカウント一括作成
  - 認証情報配布

## 講座で使用したテキスト

教材（一部抜粋）

- 第 1 章: これから作るもの – 完成イメージと学習の流れ



### 完成イメージ

#### このコースを終えると

以下ができるようになる。

- 環境モニタリングシステム: 全室の温度・湿度をリアルタイムで監視
- 遠隔制御システム: クラウドから照明を操作
- 統合ダッシュボード: 全客室の状態を一画面で確認
- データ分析: 運営データから改善点を発見

#### システム構成図

各コンポーネントの役割:

**学生 PC（客室センサーユニット）:**

あなたの PC が 1 つの客室として動作する。温度・湿度を送信し、照明制御コマンドを受信する。

**クラウド（SORACOM + AWS）:**

データを蓄積し、デバイスを管理する。全室のデータを集約する。

**ダッシュボード:**

全室の状態を一覧表示し、リアルタイムで監視・制御できる管理画面。

- 第 2 章: クラウドと IoT の基礎 – クラウドの本質を理解する

# 第2章: クラウドとIoTの基礎

## クラウドとは何か？

### 身近なクラウドサービスから理解する

あなたは既にクラウドを使っている：

**例 1: Gmail**:

```
- メールが「Google のコンピュータ」に保存されている
- スマホでも PC でも同じメールが見られる
- 容量不足？ Google が勝手に増やしてくれる
→ 自分でサーバーを管理していない
```

**例 2: Netflix**:

```
- 世界中で 2 億人が同時視聴
- でも Netflix は自前でサーバーを持たない（AWS を利用）
- 急にアクセスが増えても自動でスケール
→ インフラ管理から解放される
```

**例 3: Notion / Slack**:

```
- チームのドキュメントやメッセージが「クラウド」に
- 誰かの PC が壊れてもデータは消えない
- バックアップ？セキュリティ？運営会社が全部やってくれる
→ 使うことだけに集中できる
```

## クラウドの本質: 「管理しなくて良い」

**従来の方法（自前サーバー）との比較:**

| やるべきこと | 自前サーバー | クラウド |
|---|---|---|
| **サーバー購入** | 数十万円の初期投資 | 不要（すぐ使える） |
| **設置場所** | オフィスに専用スペース | 不要 |
| **電気代** | 24 時間稼働で月¥5,000〜 | 使用料に含まれる |
| **故障対応** | 深夜でも駆けつける | クラウド業者が対応 |
| **セキュリティ対策** | 自分でパッチ適用 | 自動で最新化 |
| **バックアップ** | 毎日手動で実行 | 自動・多重化 |
| **容量不足** | 新サーバー購入 | クリック 1 つで拡張 |
| **災害対策** | 火事・地震でデータ消失 | 世界中に分散保存 |

18

- 第 3 章: システム全体像とセットアップ – アーキテクチャの理解



(前のページからの続き)

```
- 実運用へスムーズに移行できる
- どこでも設置できる IoT を実現できる
```

## システム概要

### IoT Stay のアーキテクチャ

本システムは、実際のビジネスで使われる技術を学ぶためのプラットフォームである。

**主な特徴:**

- **実践的**: 実際の民泊運営を想定した設計
- **スケーラブル**: 小規模から大規模まで拡張可能
- **エンタープライズグレード**: AWS IoT Core、 SORACOM など本格的なクラウド
- **学習者中心**: 複雑な部分は隠蔽し、本質に集中

**あなたの役割:**

- **客室担当エンジニア**: 1 客室のセンサーユニット開発
- **チームメンバー**: 全客室を協力して管理
- **データオーナー**: 自分の客室データを可視化・分析

## アーキテクチャ全体図

## 主要なデータフロー（民泊シナリオ）

**1. 環境モニタリング（HTTP）** 📊:

```
【民泊での使い方】
客室の温度・湿度を 10 秒ごとに送信
→ オフィスのダッシュボードでリアルタイム確認

【技術】
デバイスアプリ → SORACOM Arc → SORACOM Harvest
- プロトコル: HTTP
- 用途: 大量データの蓄積
- 実装: 第 4 章・第 5 章
```

**なぜ HTTP を使うのか？**

- シンプル: Web ブラウザと同じ技術
- 安定: 枯れた技術で信頼性が高い
- 蓄積に最適: データベースへの保存が容易

**2. リアルタイム制御（MQTT）** 🎛:

```
【民泊での使い方】
チェックイン前に照明を点灯
→ ワンクリックで即座に反応
```

(次のページに続く)

- 第 5 章: アップリンク機能の実装 – SORACOM Harvest へのデータ送信



データベース構築やインフラ管理は本質ではない。それらは後から学べばよい。

**まず動かす → 理解する → 深める**

## SORACOM Harvest 設定

クラウド側の準備をする。ここでは教師が事前に設定した内容を確認する。

### なぜ SORACOM Beam は不要なのか？

第 5 章の MQTT 制御では SORACOM Beam を使うが、この章の HTTP アップリンクでは不要だ。

**理由:**:

```
【HTTP アップリンク】
客室センサー → SORACOM Arc → SORACOM Harvest （直接）

→ Harvest 自体が HTTP エンドポイントを提供
→ 中継（Beam）が不要

【MQTT 制御（次章）】
制御ツール → SORACOM Arc → SORACOM Beam → AWS IoT Core

→ AWS IoT Core は証明書認証が必要
→ Beam が証明書を管理
```

**アーキテクチャ設計の原則**

**シンプルなものはシンプルに保つ**

データ送信だけなら直接 Harvest へ。複雑な認証が必要なら中継（Beam）を使う。

**必要なものだけ使う = コスト削減 + 保守性向上**

### データフロー

## PC アプリケーションの実装

次に、 PC から SORACOM Harvest に HTTP でデータを送信する Python スクリプトを実装する。

### 前提条件

- プロジェクトの依存関係が `pyproject.toml` に定義されており、 `requests` ライブラリがインストールされている。

  以下のコマンドで環境を同期する。

  ```
  uv sync
  ```

- 第 6 章: ダウンリンク機能の実装 – HTTP と MQTT の比較



表 6: MQTT が制御に最適な理由

| 特徴 | HTTP | MQTT |
|---|---|---|
| **通信方向** | クライアント→サーバー（一方向） | **双方向（Pub/Sub）** |
| **レイテンシ** | ポーリング間隔に依存（数秒〜数十秒） | **即座（1 秒以内）** |
| **接続維持** | リクエストごとに接続 | **常時接続（Keep Alive）** |
| **通信量** | ヘッダー大（数百バイト） | **ヘッダー小（数バイト）** |
| **用途** | データ送信・ API 呼び出し | **リアルタイム制御・通知** |

**実務での選択基準**

**即座の反応が必要 → MQTT**

- 照明制御、ドア開閉
- チャット、通知
- ゲームのリアルタイム通信

**データ蓄積・分析 → HTTP**

- センサーデータ送信
- ログ収集
- ファイルアップロード

## MQTT の仕組み

### MQTT ブローカーとは

MQTT を使うには **ブローカー（仲介サーバー）** が必要だ。

**ブローカーの役割:**

メッセージの配達員のようなもの。

**今回使用するブローカー:**

- **AWS IoT Core** - AWS が提供する高性能 MQTT ブローカー

**SORACOM Beam の役割**

**SORACOM Beam とは？**

学生が AWS IoT Core に接続するための「証明書管理を代行するサービス」である。

**通常の MQTT 接続（複雑）:**

**SORACOM Beam 経由（シンプル）:**

**学生にとってのメリット:**

- 証明書ファイルの配布・管理が不要
- SORACOM Arc に接続するだけで動作

- 第 7 章: Streamlit ダッシュボード基礎 – 異常検知アラートの実装

### 演習 3: 異常検知アラート

**目標**: 温度閾値を超えた客室を検知し、警告を表示する

#### ステップ 1: 異常検知ロジック

温度閾値を超えた客室をリストアップする：

```
# テーブル表示の後に追加
st.markdown("---")
st.subheader("⚠️ 異常検知")

# 高温の客室を検出
high_temp_rooms = [r for r in filtered_data if r["温度"] > temp_high]

# 低温の客室を検出
low_temp_rooms = [r for r in filtered_data if r["温度"] < temp_low]
```

#### ステップ 2: アラート表示

検出された異常を表示する：

```
# 高温警告
if high_temp_rooms:
    st.error("🔥 高温警告")
    for room in high_temp_rooms:
        st.error(f"  客室 {room['客室']}: {room['温度']:.1f}°C （閾値: {temp_
↪high}°C 超過）")

# 低温注意
if low_temp_rooms:
    st.warning("❄️ 低温注意")
    for room in low_temp_rooms:
        st.warning(f"  客室 {room['客室']}: {room['温度']:.1f}°C （閾値:
↪{temp_low}°C 未満）")

# すべて正常
if not (high_temp_rooms or low_temp_rooms):
    st.success("✓ 全客室が正常範囲内です")
```

**実行して確認:**

- 103 号室が 28.5°C なので、高温警告が表示される
- サイドバーで閾値を 30°C に変更すると、警告が消える

**ℹ 💡 ヒント: Streamlit のメッセージ**

Streamlit は、メッセージの種類に応じて色分けされる：

- 第 8 章: データ分析とグラフ化 – Pandas DataFrame の活用



(前のページからの続き)

```
# Pandas: 条件式で一行
filtered = df[df["温度"] > 28]
```

**3. 集計:**

```
# 辞書のリスト: for ループ
avg_temp = sum(r["温度"] for r in rooms_data) / len(rooms_data)

# Pandas: メソッド一つ
avg_temp = df["温度"].mean()
```

### よく使う DataFrame 操作

表 12: DataFrame の主要メソッド

| メソッド | 説明 | 例 |
|---|---|---|
| df.head(n) | 最初の n 行を表示 | df.head(5) |
| df["列名"] | 列を選択 | df["温度"] |
| df[["列 1", "列 2"]] | 複数列を選択 | df[["客室", "温度"]] |
| df[条件] | 条件でフィルタリング | df[df["温度"] > 28] |
| df.sort_values("列名") | 列でソート | df.sort_values("温度") |
| df["列名"].mean() | 平均値 | df["温度"].mean() |
| df["列名"].sum() | 合計 | df["湿度"].sum() |
| df["列名"].unique() | ユニークな値 | df["客室"].unique() |
| df.iterrows() | 行をループ | for idx, row in df.iterrows(): |

### 実用例

```
import pandas as pd

# DataFrame の作成
df = pd.DataFrame([
    {"客室": "101", "温度": 25.3, "湿度": 60, "ステータス": "空室"},
    {"客室": "102", "温度": 26.1, "湿度": 58, "ステータス": "入居中"},
    {"客室": "103", "温度": 28.5, "湿度": 65, "ステータス": "入居中"},
    {"客室": "104", "温度": 24.8, "湿度": 55, "ステータス": "清掃中"},
    {"客室": "105", "温度": 22.9, "湿度": 62, "ステータス": "空室"}
])

# 基本統計
print(f"平均温度: {df['温度'].mean():.1f}°C")
print(f"最高温度: {df['温度'].max():.1f}°C")
print(f"最低温度: {df['温度'].min():.1f}°C")

# 高温の客室を抽出
high_temp = df[df["温度"] > 28]
print(f"\n高温警告: {len(high_temp)}室")
print(high_temp[["客室", "温度"]])
```

(次のページに続く)

# 今後の課題

本教材の運用および今後の発展に向けて、以下の課題が挙げられる。

- **チーム開発演習の導入**

  本教材は 3 年次 PBL（Project-Based Learning）の基盤として設計されている。20 客室統合システムの協調開発体験を通じて、実践的なチーム開発スキルの習得を目指す発展演習の検討が必要である。各学生が担当する客室ユニットを実際に統合し、全体システムとして運用する演習により、Git/GitHub によるバージョン管理、コードレビュー、CI/CD パイプラインなど、現場で求められる開発プロセスを体験できる。

- **セキュリティ教育の拡充**

  現在の教材では SORACOM Beam による証明書管理の簡略化を行っているが、IoT セキュリティの重要性を考慮すると、認証・認可・暗号化の基礎的な理解を深める補足教材の開発が望まれる。特に、TLS/SSL 証明書の仕組み、AWS IAM ポリシーの設計、デバイス認証のベストプラクティスなど、セキュリティ関連のトピックを段階的に学べる構成を検討する。

- **クラウドコスト管理の理解促進**

  AWS/SORACOM の従量課金モデルへの理解を深めるため、コスト最適化に関する実践的な内容の追加を検討する。データ送信頻度とコストの関係、リザーブドインスタンスの活用、不要リソースの削除など、実運用を見据えたコスト意識を醸成する教材コンテンツの開発が課題である。