Smart Parking System with Raspberry Pi & AWS

Dalair Franzen – Capstone Project

Overview
This guide walks through the step-by-step process of deploying the smart parking system. It covers the Raspberry Pi and Arduino setup, AWS cloud configuration, and frontend visualization. The system uses AWS IoT Core, Lambda, DynamoDB, and API Gateway to show live parking spot status based on sensor input.

System Requirements

Hardware
- Raspberry Pi 4 (Wi-Fi enabled)
- Arduino Uno R3
- Hall Effect Sensor
- Micro USB cables for Arduino & Pi
- Breadboard + jumper wires
- Magnet (for testing)

Software / Cloud
- Python 3.11+
- AWS CLI + IAM user
- AWS services: IoT Core, Lambda, DynamoDB, API Gateway
- Dependencies listed in requirements.txt

Step-by-Step Setup

1. Configure Arduino + Sensor
- Connect Hall Effect sensor to Arduino:
  - VCC → 5V
  - GND → GND
  - OUT → A0 or digital pin
- Upload arduino_sensor.ino sketch to Arduino
- Sensor will output 1 (available) or 0 (occupied)

2. Raspberry Pi Setup
- Install Raspbian (or Raspberry Pi OS Lite)
- Connect to Wi-Fi
- Enable serial communication:
  sudo raspi-config → Interface Options → Serial

- Clone source code repo:
  git clone https://github.com/YOUR_USERNAME/smart-parking-system.git
  cd smart-parking-system
- Install dependencies:
  pip install -r requirements.txt

3. AWS IoT Core Configuration

1. Create a 'Thing' (e.g., ParkingPi01)
2. Download and save:
   - certificate.pem.crt
   - private.pem.key
   - AmazonRootCA1.pem
3. Attach a policy that allows publishing to a topic:

```
{
  "Effect": "Allow",
  "Action": ["iot:Connect", "iot:Publish"],
  "Resource": "*"
}
```

4. Note the endpoint from Settings (you'll need this for .env)

4. Setup .env File on Raspberry Pi

Create a file named .env:

```
AWS_IOT_ENDPOINT=a1b2c3d4e5f6-ats.iot.us-east-1.amazonaws.com
AWS_REGION=us-east-1
IOT_TOPIC=parking/spots
```

Run your main script:

```
python3 parking_system.py
```

5. DynamoDB Table Setup

- Create a table named ParkingSpots
  - Primary Key: spot_id (String)
- Sample items:

```
{
  "spot_id": "2",
  "status": "1",
  "timestamp": "2025-05-19T18:45:00Z"
}
```

6. Lambda + API Gateway Setup

- Create a new Lambda function (parkingSpotStatus)
- Use Python or Node.js to query DynamoDB
- Return JSON of all spot statuses

- Grant DynamoDB read access via IAM role
- Connect to API Gateway (REST API or HTTP API)
- Enable CORS if using frontend

7. Web Frontend
- Open web_frontend/index.html in your browser
- It uses fetch() to call the Lambda API via API Gateway
- Displays a grid like:
  X X O
  O X O
- Refreshes periodically (optional)

Reset / Rollback
- Use AWS Console to delete:
  - IoT Thing, Certs, Policies
  - Lambda function
  - API Gateway route
  - DynamoDB table
- Delete .env from Pi if changing credentials

Troubleshooting
Issue: Pi not sending data → Check serial port + sensor output
Issue: No update in DynamoDB → Confirm IoT policy and MQTT topic match
Issue: Frontend shows blank grid → Confirm Lambda response format
Issue: SSL or connection error → Ensure cert files are present and .env is correct