SmartRainHarvest

AWS Deployment Guide

Version: 1.0
Date: February 22, 2026
Platform: AWS EC2 (Ubuntu 24.04 LTS)
Stack: Flask + DynamoDB + Apache + Qt WebAssembly

IoT Sensor Data Collection & Visualization Platform
Raspberry Pi → AWS Cloud → Web Dashboard

Contents

1 Architecture Overview

SmartRainHarvest is an IoT system that collects weather and sensor data from a Raspberry Pi, stores it in AWS DynamoDB via a Flask API, and visualizes it through a Qt WebAssembly dashboard served by Apache.

1.1 Data Flow

Raspberry Pi *HTTPPOST* **Flask API** *put_item* **DynamoDB** *HTTPGET* **Wasm Dashboa**
Qt/C++ app :5000 sensor_readings Apache :80

1.2 Components

Component	Technology	Port	Purpose
Sensor App	Qt 6 / C++	—	Reads NOAA data, HC-SR04 sensor, controls valve
Flask API	Python / Flask	5000	REST endpoint for sensor data ingestion/retrieval
Database	AWS DynamoDB	—	Stores sensor readings (serverless, on-demand)
Web Dashboard	Qt WebAssembly	80	Browser-based visualization of all sensor data
Web Server	Apache 2.4	80	Serves the Wasm dashboard with required headers

1.3 AWS Resources

Service	Resource	Details
EC2	t2.micro (or similar)	Ubuntu 24.04, hosts Flask API + Apache
DynamoDB	sensor_readings table	Partition key: `sensor_id`, Sort key: `timestamp`
IAM	sensor-api-user	Credentials for DynamoDB access

2 EC2 Instance Setup

2.1 Launch an EC2 Instance

1. Go to **AWS Console** → **EC2** → **Launch Instance**.
2. Select **Ubuntu 24.04 LTS** AMI.
3. Choose instance type (t2.micro is sufficient for this workload).
4. Create or select a **key pair** (.pem file) for SSH access.
5. Under **Network settings**, create a security group with:

2.2 Connect via SSH

```
ssh -i your-key.pem ubuntu@<EC2-PUBLIC-IP>
```

Type	Port	Source	Purpose
SSH	22	Your IP	SSH access
HTTP	80	0.0.0.0/0	Web dashboard
Custom	5000	0.0.0.0/0	Flask API

2.3 Install System Dependencies

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y python3 python3-pip python3-venv apache2
```

3 IAM User Configuration

3.1 Create an IAM User

1. Go to **AWS Console** → **IAM** → **Users** → **Create User**.
2. Name: `sensor-api-user`.
3. Attach the **AmazonDynamoDBFullAccess** managed policy.
4. Create **Access Keys** (CLI use case) and note the Access Key ID and Secret.

Warning

The IAM user needs `DeleteItem`, `BatchWriteItem`, `DeleteTable`, and `CreateTable` permissions in addition to basic read/write if you intend to manage or wipe the table from the EC2 instance. `AmazonDynamoDBFullAccess` covers all of these.

3.2 Configure AWS CLI on EC2

```
aws configure
# AWS Access Key ID: <your-key-id>
# AWS Secret Access Key: <your-secret>
# Default region name: us-west-2
# Default output format: json
```

4 DynamoDB Table Setup

4.1 Create the Table

```
aws dynamodb create-table \
  --table-name sensor_readings \
  --attribute-definitions \
    AttributeName=sensor_id,AttributeType=S \
    AttributeName=timestamp,AttributeType=S \
  --key-schema \
    AttributeName=sensor_id,KeyType=HASH \
    AttributeName=timestamp,KeyType=RANGE \
  --billing-mode PAY_PER_REQUEST \
  --region us-west-2
```

4.2 Schema

Attribute	Type	Key	Description
sensor_id	String (S)	Partition Key	e.g. `temperature`, `precip_amount`
timestamp	String (S)	Sort Key	ISO 8601 format
value	String (S)	—	Sensor reading (stored as string)
unit	String (S)	—	Unit of measurement

4.3 Wipe All Data (if needed)

```
import boto3

dynamodb = boto3.resource('dynamodb', region_name='us-west-2')
table = dynamodb.Table('sensor_readings')

response = table.scan(
    ProjectionExpression='sensor_id, #ts',
    ExpressionAttributeNames={'#ts': 'timestamp'}
)
items = response['Items']

while 'LastEvaluatedKey' in response:
    response = table.scan(
        ProjectionExpression='sensor_id, #ts',
        ExpressionAttributeNames={'#ts': 'timestamp'},
        ExclusiveStartKey=response['LastEvaluatedKey']
    )
    items.extend(response['Items'])

for i, item in enumerate(items):
    table.delete_item(Key={
        'sensor_id': item['sensor_id'],
        'timestamp': item['timestamp']
    })

print(f"Deleted {len(items)} items")
```

5 Flask API Server

5.1 Create the Project

```
mkdir -p /home/ubuntu/sensor-api
cd /home/ubuntu/sensor-api
python3 -m venv venv
source venv/bin/activate
pip install flask flask-cors boto3
```

5.2 Flask Application (app.py)

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import boto3
from boto3.dynamodb.conditions import Key
from datetime import datetime

app = Flask(__name__)
CORS(app)

dynamodb = boto3.resource('dynamodb', region_name='us-west-2')
table = dynamodb.Table('sensor_readings')


@app.route('/sensor', methods=['POST'])
def receive_sensor_data():
    data = request.json
    timestamp = data.get('timestamp',
                         datetime.utcnow().isoformat() + 'Z')
    item = {
        'sensor_id': data['sensor_id'],
        'timestamp': timestamp,
        'value': str(data['value']),
        'unit': data.get('unit', '')
    }
    table.put_item(Item=item)
    return jsonify({'status': 'success'}), 200


@app.route('/sensor/<sensor_id>', methods=['GET'])
def get_sensor_data(sensor_id):
    start = request.args.get('start')
    end   = request.args.get('end')
    key_condition = Key('sensor_id').eq(sensor_id)

    if start and end:
        key_condition = key_condition & \
            Key('timestamp').between(start, end)
    elif start:
        key_condition = key_condition & \
            Key('timestamp').gte(start)
    elif end:
        key_condition = key_condition & \
            Key('timestamp').lte(end)

    response = table.query(
        KeyConditionExpression=key_condition)
    return jsonify(response['Items']), 200


@app.route('/sensors', methods=['GET'])
def list_sensors():
    response = table.scan(
        ProjectionExpression='sensor_id')
    ids = sorted(set(
        item['sensor_id'] for item in response['Items']))
    return jsonify(ids), 200
```

```
@app.route('/health', methods=['GET'])
def health():
    return jsonify({'status': 'ok'}), 200


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

5.3 API Endpoints

Method	Endpoint	Description
POST	`/sensor`	Store a sensor reading
GET	`/sensor/<id>?start=&end=`	Query readings with optional date range
GET	`/sensors`	List all distinct sensor IDs
GET	`/health`	Health check

5.4 Systemd Service

Create `/etc/systemd/system/sensor-api.service`:

```
[Unit]
Description=Sensor API Flask Server
After=network.target

[Service]
User=ubuntu
WorkingDirectory=/home/ubuntu/sensor-api
ExecStart=/home/ubuntu/sensor-api/venv/bin/python3 app.py
Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target
```

Enable and start:

```
sudo systemctl daemon-reload
sudo systemctl enable sensor-api
sudo systemctl start sensor-api
sudo systemctl status sensor-api
```

Note

The systemd service ensures the Flask API survives SSH disconnections and automatically restarts on crash or reboot.

5.5 Verify the API

```
# Health check
curl http://localhost:5000/health
```

```
# Test POST
curl -X POST http://localhost:5000/sensor \
  -H "Content-Type: application/json" \
  -d '{"sensor_id":"test","value":42.0,"unit":"C"}'

# List sensors
curl http://localhost:5000/sensors

# Query with date range
curl "http://localhost:5000/sensor/temperature?\
start=2026-02-22T00:00:00&end=2026-02-28T00:00:00"
```

6 Qt WebAssembly Dashboard

6.1 Prerequisites

1. **Emscripten SDK** (3.1.25 or later):

```
git clone https://github.com/emscripten-core/emsdk.git
cd emsdk
./emsdk install latest
./emsdk activate latest
source ./emsdk_env.sh
```

2. **Qt 6.8 for WebAssembly**: Install via the Qt Online Installer. Select your Qt version and check **WebAssembly (single-threaded)**.

Warning

The **multi-threaded** Wasm build requires `SharedArrayBuffer`, which browsers only enable over **HTTPS**. For deployment over plain HTTP, use the **single-threaded** build.

6.2 Project Structure

```
SensorDashboard/
  SensorDashboard.pro
  main.cpp
  SensorDashboard.h
  SensorDashboard.cpp
```

6.3 Build for Desktop (Testing)

```
cd SensorDashboard
mkdir build && cd build
qmake ..
make -j$(nproc)
./SensorDashboard
```

6.4 Build for WebAssembly

```
cd SensorDashboard
mkdir build-wasm && cd build-wasm

# Use the single-threaded Wasm kit
~/Qt/6.8.2/wasm_singlethread/bin/qmake ..
make -j$(nproc)
```

This produces the following files:

- `SensorDashboard.html` — entry page
- `SensorDashboard.js` — JavaScript glue code
- `SensorDashboard.wasm` — compiled binary
- `qtloader.js` — Qt's Wasm bootstrap loader

6.5 Dashboard Features

- Displays **all sensors simultaneously** in stacked charts (no combo box).
- **Date range pickers** default to ±7 days from current time.
- **Auto-refresh** mode polls the API every 60 seconds with a countdown display.
- Color-coded series: temperature (red), precipitation amount (blue), precipitation probability (purple), water depth (teal), valve state (green).
- Dynamically discovers sensors via `GET /sensors` endpoint.

7 Apache Web Server Configuration

7.1 Install Apache

```
sudo apt install -y apache2
sudo a2enmod headers
```

7.2 Create Virtual Host

Create `/etc/apache2/sites-available/dashboard.conf`:

```
<VirtualHost *:80>
    ServerName 54.213.147.59
    DocumentRoot /home/ubuntu/dashboard

    <Directory /home/ubuntu/dashboard>
        Options Indexes FollowSymLinks
        AllowOverride None
        Require all granted
    </Directory>

    DirectoryIndex SensorDashboard.html

    # Required for multi-threaded Wasm (SharedArrayBuffer)
    Header set Cross-Origin-Opener-Policy "same-origin"
    Header set Cross-Origin-Embedder-Policy "require-corp"
```

```
    # Correct MIME type for .wasm files
    AddType application/wasm .wasm
</VirtualHost>
```

7.3 Enable and Start

```
sudo a2dissite 000-default
sudo a2ensite dashboard
sudo apache2ctl configtest
sudo systemctl restart apache2
```

7.4 Upload Dashboard Files

Use FileZilla (or `scp`) to upload the Wasm build output to `/home/ubuntu/dashboard/`:

```
# Required files:
/home/ubuntu/dashboard/
  SensorDashboard.html
  SensorDashboard.js
  SensorDashboard.wasm
  qtloader.js
```

Set permissions:

```
chmod 755 /home/ubuntu /home/ubuntu/dashboard
chmod 644 /home/ubuntu/dashboard/*
```

7.5 Verify Deployment

```
# Check Apache is serving files
curl -I http://54.213.147.59/

# Verify CORS headers are present
# Look for:
#   Cross-Origin-Opener-Policy: same-origin
#   Cross-Origin-Embedder-Policy: require-corp
```

Open `http://54.213.147.59` in a web browser. The dashboard should load and begin displaying sensor data.

8 Updating the Dashboard

When you modify and rebuild the Qt Wasm project:

1. Rebuild in Qt Creator using the single-threaded Wasm kit.
2. Upload the new `.html`, `.js`, `.wasm`, and `qtloader.js` files to `/home/ubuntu/dashboard/` via FileZilla, overwriting existing files.
3. No Apache restart is needed — just hard-refresh the browser (`Ctrl+Shift+R`).

To update the Flask API:

```
# Edit the file
sudo nano /home/ubuntu/sensor-api/app.py

# Restart the service
sudo systemctl restart sensor-api
sudo systemctl status sensor-api
```

9 Troubleshooting

9.1 Common Issues

Symptom	Solution
Connection refused on :5000	Check `sudo systemctl status sensor-api`. Restart if needed.
403 Forbidden on :80	Run `chmod 755 /home/ubuntu /home/ubuntu/dashboard`
SharedArrayBuffer not defined	Use the single-threaded Wasm build, or add HTTPS with Let's Encrypt.
`qtLoad is not defined`	Ensure `qtloader.js` is uploaded to the dashboard directory.
CORS errors in browser	Verify `flask-cors` is installed and `CORS(app)` is in `app.py`.
Flask ModuleNotFoundError	Ensure `ExecStart` in the systemd unit points to the venv Python.
Port 80 already in use	Another web server is running. Stop it: `sudo systemctl stop apache2` (or nginx) and disable it before starting the intended server.

9.2 Useful Commands

```
# Check Flask API logs
sudo journalctl -u sensor-api -f

# Check Apache logs
sudo tail -f /var/log/apache2/error.log

# Check what's using a port
sudo lsof -i :80
sudo lsof -i :5000

# Restart services
sudo systemctl restart sensor-api
sudo systemctl restart apache2

# Verify DynamoDB connectivity
aws dynamodb describe-table \
  --table-name sensor_readings --region us-west-2
```

10 Adding HTTPS (Optional)

HTTPS is required if you want to use the multi-threaded Wasm build in production. It also secures data in transit.

10.1 Prerequisites

- A domain name pointing to your EC2 public IP (A record).
- Port 443 open in your EC2 security group.

10.2 Install Certbot

```
sudo apt install -y certbot python3-certbot-apache
sudo certbot --apache -d yourdomain.com
```

Certbot automatically configures Apache with SSL and sets up auto-renewal. After this, the multi-threaded Wasm build will work because the browser will enable `SharedArrayBuffer` over HTTPS.

11 Security Recommendations

1. **Restrict port 5000** in the security group to only the Raspberry Pi's IP and the EC2 instance's own IP (127.0.0.1), rather than 0.0.0.0/0.
2. **Use HTTPS** in production to encrypt data in transit.
3. **Add API authentication** (e.g. an API key header) to the Flask POST endpoint to prevent unauthorized writes.
4. **Enable DynamoDB encryption at rest** (enabled by default with AWS-owned keys).
5. **Rotate IAM credentials** periodically and use the principle of least privilege.
6. **Enable CloudWatch alarms** for DynamoDB throttling and EC2 instance health.