Overview

This system simulates humidity levels, stores the data in Azure Blob Storage, provides access to the data via a REST API, and sends SMS alerts using Twilio if humidity levels are out of bounds.

---

1. Prerequisites

Ensure the following are installed on your machine:

1. Python 3.6+
2. Required Python libraries:

bash

CopyEdit

pip install azure-storage-blob azure-iot-device flask twilio

3. Azure services:
   - **Azure Blob Storage** account with an active container.
   - **Azure IoT Hub** with a device registered (e.g., humidity-simulator).
4. A **Twilio account** (upgraded if necessary to remove free-tier limits).
5. SQLite (comes pre-installed with Python).

---

2. Code Components

2.1 Humidity Simulator

- Simulates random humidity values between 30% and 80%.
- Uploads humidity data to Azure Blob Storage in JSON format.
- Sends data to Azure IoT Hub.
- Triggers SMS alerts if humidity is:
  - **Below 35%**: "Humidity is too low!"
  - **Above 65%**: "Humidity is too high!"

2.2 REST API

- Allows storing and retrieving humidity data from the SQLite database.
- Endpoints:
    - **POST /humidity**: Save humidity data.
    - **GET /humidity**: Retrieve all humidity records.

2.3 Alert System

- Uses Twilio to send SMS alerts when humidity crosses safe thresholds (35% - 65%).

2.4 Integration

- Combines all components:
    - Humidity simulation, data storage, REST API, and alerts run seamlessly as one system.

---

3. How the Project Runs

Here's the flow:

Step 1: Start the System

Run the integrated script:

bash

CopyEdit

python humidity_monitoring_system.py

Step 2: Simulate Humidity

- The system generates a new humidity value every 2 seconds.
- Simulated data is:
    - Sent to Azure IoT Hub.
    - Uploaded as a JSON file to Azure Blob Storage.
    - Saved in the SQLite database.

Example of JSON file uploaded:

json

CopyEdit

```
{
  "humidity": 39.77,
  "timestamp": "2025-01-22 14:30:21"
}
```

Step 3: REST API Functionality

- Access the API while the system is running:
    - **Save data**:

bash

CopyEdit

```
curl -X POST http://127.0.0.1:5000/humidity -H "Content-Type: application/json" -d '{"humidity": 55.4}'
```

Response:

json

CopyEdit

```
{"message": "Humidity data saved successfully!"}
```

-
    - **Retrieve data**:

bash

CopyEdit

```
curl http://127.0.0.1:5000/humidity
```

Response:

json

CopyEdit

```
[
  {"id": 1, "humidity": 55.4, "timestamp": "2025-01-22 14:32:00"},
  ...
]
```

Step 4: Send Alerts

- Alerts are triggered and sent via SMS when humidity is:

        - **Below 35%** or **Above 65%**.

Example of an alert:

text

CopyEdit

ALERT: Humidity is too high! Current humidity: 69.48%

Step 5: End the Simulation

- Press CTRL+C to stop the script.
- The system will shut down gracefully:
    - Any unsent IoT Hub messages are cleared.
    - Processes stop running.

---

4. File Structure

Ensure the following files are in the project directory:

plaintext

CopyEdit

humidity_monitoring_system.py

humidity_data.db # SQLite database file

requirements.txt # Optional: List of dependencies

---

5. Deployment Guide

For deploying this project in the real world:

1. **Host the REST API**:
    - Use a platform like **AWS EC2**, **Azure App Service**, or **Heroku**.
    - Update the API's public IP or domain in relevant configurations.
2. **Run Humidity Simulator**:
    - Set it up as a **background service** or a scheduled task.
3. **Set Up Monitoring**:

  - Integrate with **Azure Monitor** or similar tools for additional logging and alerts.

4. **Twilio Alerts**:
  - Ensure your Twilio account has sufficient funds and production phone numbers.

---

6. Testing and Debugging

- **Local Testing**:
  - Test REST API using curl or Postman.
  - Verify simulated data in the database:

bash

CopyEdit

```
sqlite3 humidity_data.db "SELECT * FROM humidity_data;"
```

  - Check uploaded files in Azure Blob Storage via the Azure portal.
- **Debugging**:
  - Review logs printed to the terminal.
  - Check Twilio and Azure dashboards for detailed error information.

---

7. Troubleshooting

- **Error: Twilio Message Limit Exceeded**:
  - Upgrade your Twilio account.
  - Reduce the frequency of alerts.
- **Error: Blob Storage Upload Fails**:
  - Verify the Azure connection string and container name.
- **Error: API Fails to Start**:
  - Check Flask installation and ensure port 5000 is not in use.