# ESP32 + DHT11 + AWS IoT Core (MicroPython)

Project README, hardware setup, AWS setup, and step-by-step guide for configuring and running the project. The MicroPython code file is excluded from this document and should be uploaded in the repository `code/` directory.

## Skills Learned

* · **IoT fundamentals** – connecting sensors to cloud services.
* · **AWS and AWS IoT Core** – creating Things, certificates, and policies.
* · **ESP32 board programming with MicroPython.**
* · **DHT11 sensor integration** for temperature and humidity data.
* · **MQTT protocol** – publish/subscribe messaging model.
* · **Using MicroPython timers (RTOS-style feature)** for periodic tasks.

## Hardware Components

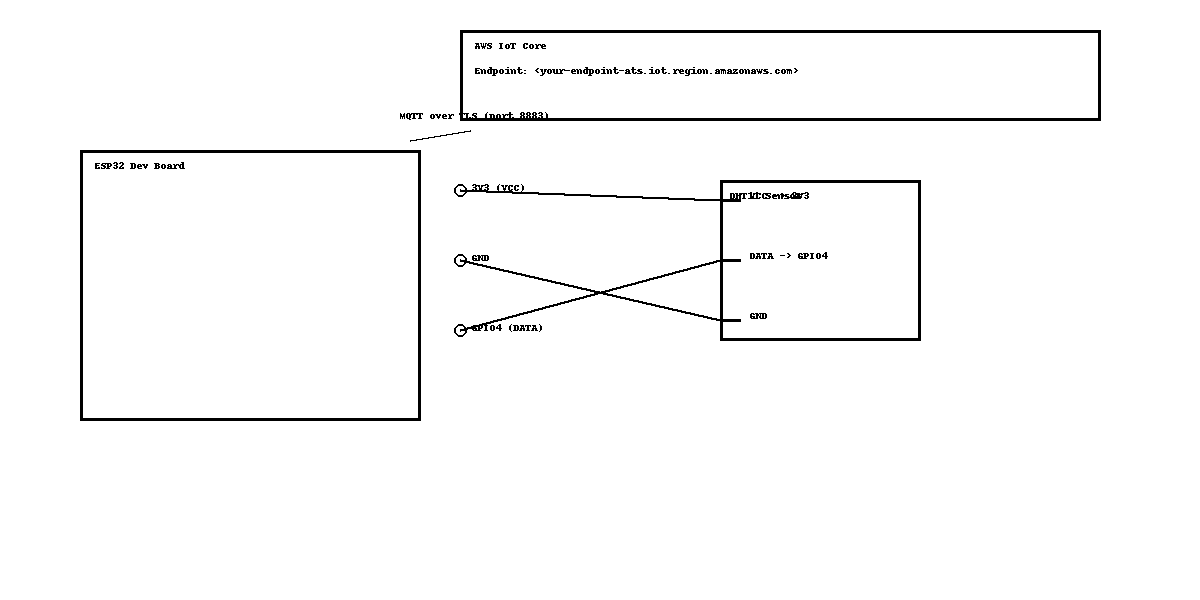
ESP32 dev board, DHT11 sensor, jumper wires, USB cable.

## Wiring

DHT11 VCC -> ESP32 3V3  
DHT11 GND -> ESP32 GND  
DHT11 DATA -> ESP32 GPIO4 (configurable in code)

## Circuit Diagram

See the circuit diagram below:



## AWS IoT Setup (step-by-step)

1. In AWS IoT Core, create a Thing (e.g., myESP32).  
2. Create certificates for the Thing and download: device cert, private key, and Amazon Root CA.  
3. Create an IoT policy with iot:Connect, iot:Publish, iot:Subscribe, iot:Receive and attach it to the certificate.  
4. Attach the certificate to the Thing.  
5. Copy the Device data endpoint from IoT Core Settings (use this as MQTT endpoint).

## Preparing & Uploading Certificates

Upload the following files to the ESP32 filesystem (root):  
- AmazonRootCA1.pem  
- certificate.pem.crt  
- private.pem.key  
Confirm files with os.listdir().

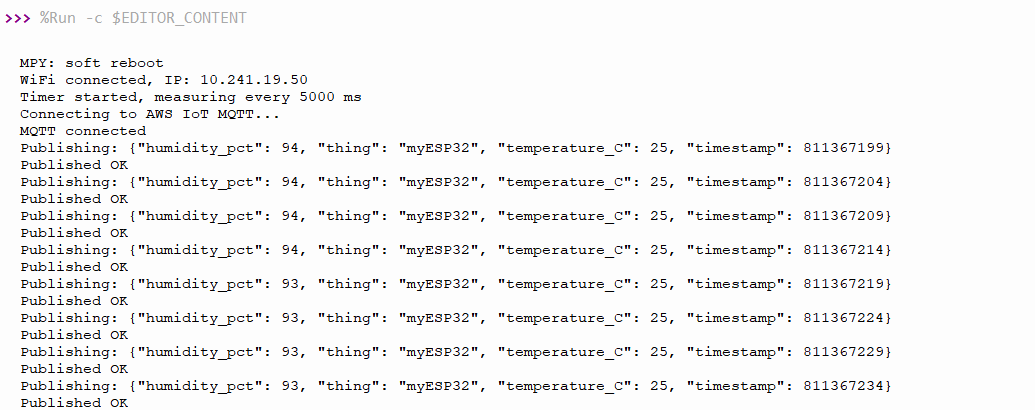
## Running & Testing

1. Flash MicroPython with TLS support to your ESP32.  
2. Upload code in a separate code/ folder in the repo.  
3. Update Wi-Fi credentials and AWS endpoint in the code before running.  
4. Run the script and monitor REPL output.  
5. Use AWS IoT MQTT test client to subscribe to the topic devices/<thing>/telemetry or use wildcards like devices/+/telemetry or devices/#.

## Troubleshooting (common issues)

- TLS handshake errors: ensure endpoint is the AWS IoT Device endpoint (not ARN); confirm cert files and firmware TLS support.  
- Publish issues: check reconnection logic, Wi-Fi stability, and IoT policy permissions.  
- File not found: check filenames and case sensitivity.

## Final Output / Evidence



# Screenshot 2025-09-16 195035

# Future Scope

* Extend the project to subscribe to AWS IoT topics so the ESP32 can receive commands (e.g., LED control, relay switching).
* Integrate with AWS Lambda / DynamoDB / Timestream for automated data storage and serverless processing.
* Add alerting (SNS, email, SMS) when temperature or humidity crosses thresholds.
* Support additional sensors (e.g., BMP280, DS18B20) to make a full environment station.
* Implement OTA (Over-The-Air) updates for MicroPython firmware and scripts.
* Create a web dashboard or mobile app to visualize data in real time.
* Transition from basic timers to FreeRTOS tasks for more advanced scheduling.

## License

MIT License (or choose your preferred license).

## Author

Shashanka Shekhar Chakraborty