

Project Overview

DIXELL -> JETSON NANO -> AWS CLOUD

13-12-2023

Contents

Overview:	2
Jetson Nano Analytics	2
RS485 to Jetson	2
Jetson to Local CSV Endpoint	3
Jetson to AWS IoT Core Endpoint	4
Commands	5
AWS Endpoint	5

Project Walkthrough

Overview:

The aim of this project was to interface Dixell RS485 connection to Jetson Nano and using edge computing on jetson, we log and push the data to AWS cloud server. Jetson Nano was programmed to acquire data from XR70CX using the RS485 connection. Python script along with minimalmodbus library worked to get data from the Dixell Temperature Sensor and the AWS-SDK was implemented for pushing data to AWS IoT Core.

Following is the view of the final result developed at end of this project:

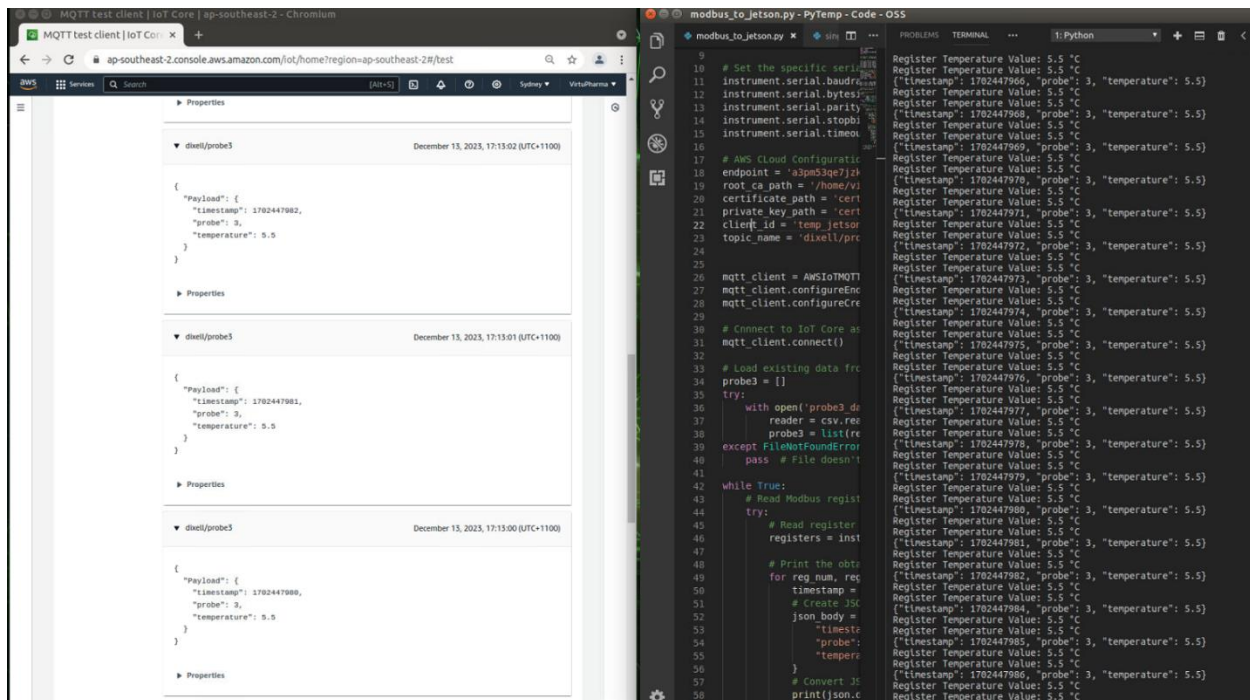


Figure 1 (AWS) -> (Python) -> (Debug Console)

Jetson Nano Analytics

The Jetson Nano itself is used to acquire data from the MODBUS and process it. We apply limiting range of 2C and 8C and set respective high and low flags to indicate the data is in range. Moreover an hourly temperature average is calculated suggesting the temperature trend for the last hour.

Additionally a percentage deviation is sent along showing how much the current value is deviated from the ideal setpoint of 5C.

RS485 to Jetson

The data from sensor was connected to the Jetson Nano using RS485 to USB serial converter and the device shows up as USB serial device occupying a `/dev/ttyUSB0` port on the Jetson Nano. The fride itself

utilizes MODBUS protocol to this provided a method to communicate MODBUS commands to sensor and read the current data.

The protocol has following parameters:

Baud Rate = 9600

Data Bits = 8

Parity Bit = None

Stop Bit = 1

USB Port = As assigned by the Nano -> (/dev/ttyUSB0)

These parameters were added to the MODBUS node settings in python script to ensure proper communication between both.

Jetson to Local CSV Endpoint

The incoming data is logged into a csv file for further computational requirements that may arise further down the timeline. Incoming data is read from the registers, scaled accordingly and then stored in the csv file along with the timestamp in unix format. For each session, if the file already has data, the code reads this file and then appends the new data at the end of last session acquired data.

The csv file can be read by opening it in LibreOffice or using the python script I've made for this purpose.

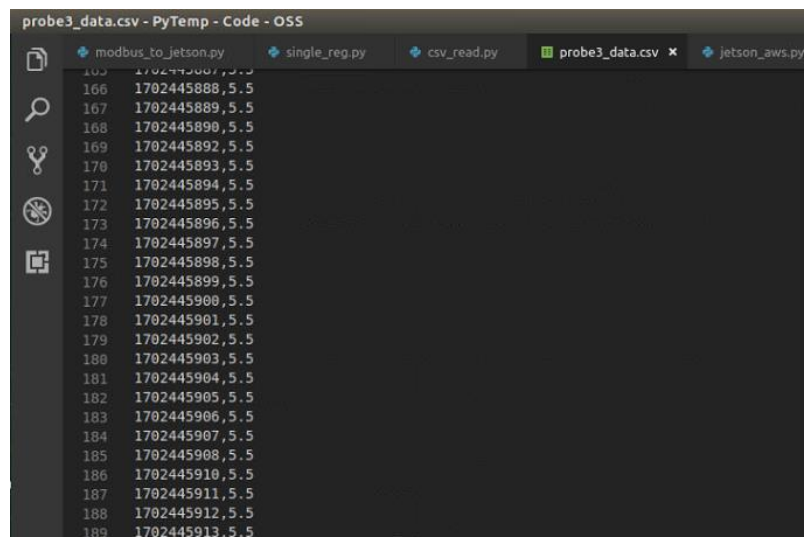
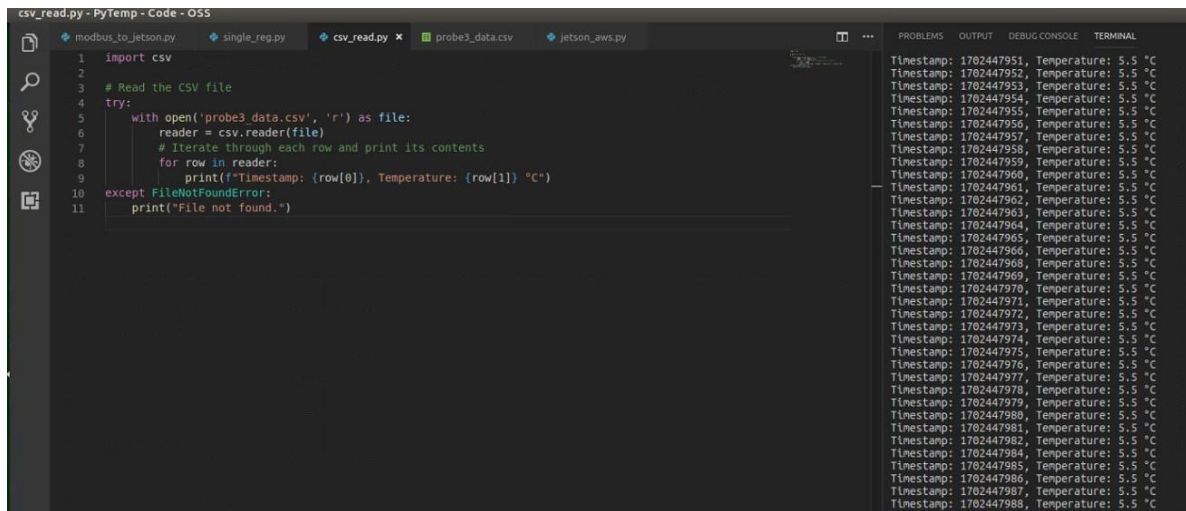


Figure 2 CSV file direct Reading



The screenshot shows a PyTemp IDE window titled 'csv_read.py - PyTemp - Code - OSS'. The editor displays a Python script that reads a CSV file named 'probe3_data.csv'. The script imports the 'csv' module, opens the file, and iterates through each row, printing the timestamp and temperature. The terminal on the right shows the output of the script, displaying a series of timestamps and temperature values (5.5 °C).

```
1 import csv
2
3 # Read the CSV file
4 try:
5     with open('probe3_data.csv', 'r') as file:
6         reader = csv.reader(file)
7         # Iterate through each row and print its contents
8         for row in reader:
9             print(f"Timestamp: {row[0]}, Temperature: {row[1]} °C")
10 except FileNotFoundError:
11     print("File not found.")
```

Timestamp: 1702447951, Temperature: 5.5 °C
Timestamp: 1702447952, Temperature: 5.5 °C
Timestamp: 1702447953, Temperature: 5.5 °C
Timestamp: 1702447954, Temperature: 5.5 °C
Timestamp: 1702447955, Temperature: 5.5 °C
Timestamp: 1702447956, Temperature: 5.5 °C
Timestamp: 1702447957, Temperature: 5.5 °C
Timestamp: 1702447958, Temperature: 5.5 °C
Timestamp: 1702447959, Temperature: 5.5 °C
Timestamp: 1702447960, Temperature: 5.5 °C
Timestamp: 1702447961, Temperature: 5.5 °C
Timestamp: 1702447962, Temperature: 5.5 °C
Timestamp: 1702447963, Temperature: 5.5 °C
Timestamp: 1702447964, Temperature: 5.5 °C
Timestamp: 1702447965, Temperature: 5.5 °C
Timestamp: 1702447966, Temperature: 5.5 °C
Timestamp: 1702447967, Temperature: 5.5 °C
Timestamp: 1702447968, Temperature: 5.5 °C
Timestamp: 1702447969, Temperature: 5.5 °C
Timestamp: 1702447970, Temperature: 5.5 °C
Timestamp: 1702447971, Temperature: 5.5 °C
Timestamp: 1702447972, Temperature: 5.5 °C
Timestamp: 1702447973, Temperature: 5.5 °C
Timestamp: 1702447974, Temperature: 5.5 °C
Timestamp: 1702447975, Temperature: 5.5 °C
Timestamp: 1702447976, Temperature: 5.5 °C
Timestamp: 1702447977, Temperature: 5.5 °C
Timestamp: 1702447978, Temperature: 5.5 °C
Timestamp: 1702447979, Temperature: 5.5 °C
Timestamp: 1702447980, Temperature: 5.5 °C
Timestamp: 1702447981, Temperature: 5.5 °C
Timestamp: 1702447982, Temperature: 5.5 °C
Timestamp: 1702447984, Temperature: 5.5 °C
Timestamp: 1702447985, Temperature: 5.5 °C
Timestamp: 1702447986, Temperature: 5.5 °C
Timestamp: 1702447987, Temperature: 5.5 °C
Timestamp: 1702447988, Temperature: 5.5 °C
Timestamp: 1702447989, Temperature: 5.5 °C

Figure 3 CSV reading using python

Jetson to AWS IoT Core Endpoint

Sending data from Jetson Nano to AWS cloud involved using the MQTT protocol by first filtering the incoming data upon MODBUS read, extracting the temperature values and sending it over to AWS using MQTT.

The MQTT protocol was setup in AWS by creating an IoT core “Thing” and using it to manage MQTT data.

Inside the python script the MQTT was implemented using the AWS SDK, and necessary certificates were attached to configure secure TLS communication between AWS and Jetson Nano. After the setup, data was pushed to the relevant AWS IoT Core topic and can now be further utilized as setup.

And the resultant data received at the AWS showed Probe 3 Temperature values (as only those were connected).

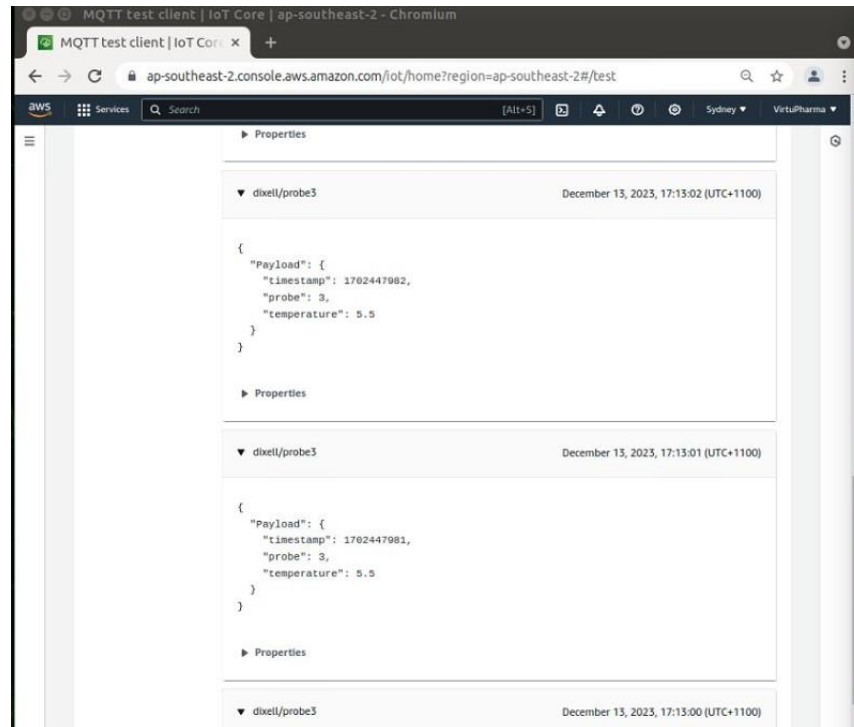


Figure 4 AWS IoT Core Endpoint

Commands

Once the Jetson Nano is powered up the PyTemp folder on the desktop contains all the necessary files and certificates. It also contains the csv file that logs the data.

For easier access I've constructed a bash script that starts the whole process with ease. It's on desktop and can be executed by opening terminal on desktop and executing following commands.

cd Desktop

./begin.sh

AWS Endpoint

At the AWS endpoint we can see the incoming data by subscribing to the topic **"dixell/probe0"** and the data is being stored into the timestream database table. To look up the data, navigate to timestream and then to the tables and run the following query to see data stored in table.

The table has standard memory retention of 1 day and magnetic memory retention for 1 Year, this can be modified according to your needs.

Query:

```
SELECT * FROM jetson_database."jetson_data" order by time desc
```

this gives the complete table data in decreasing order of timestamps.

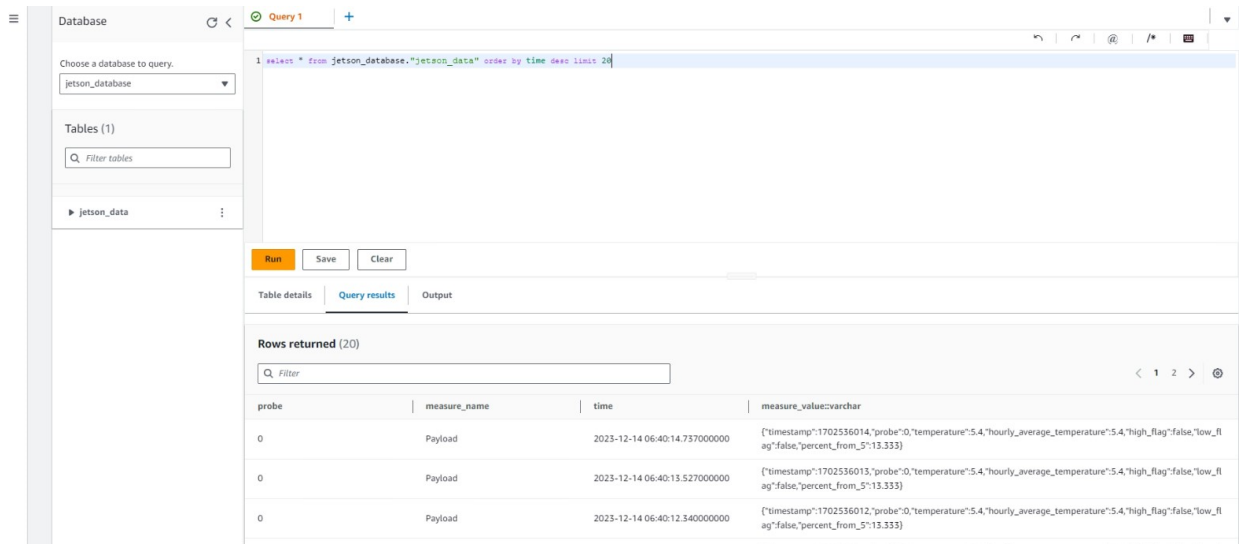


Figure 5 Timestream database view for realtime data

Data can be further managed by modifying the query to your requirements.

This completes the data acquisition, local csv storage and cloud database management part along with edge analytics and logging.