smart-factory-prototype

from Serdar Akol and Jonathan Zöllinger

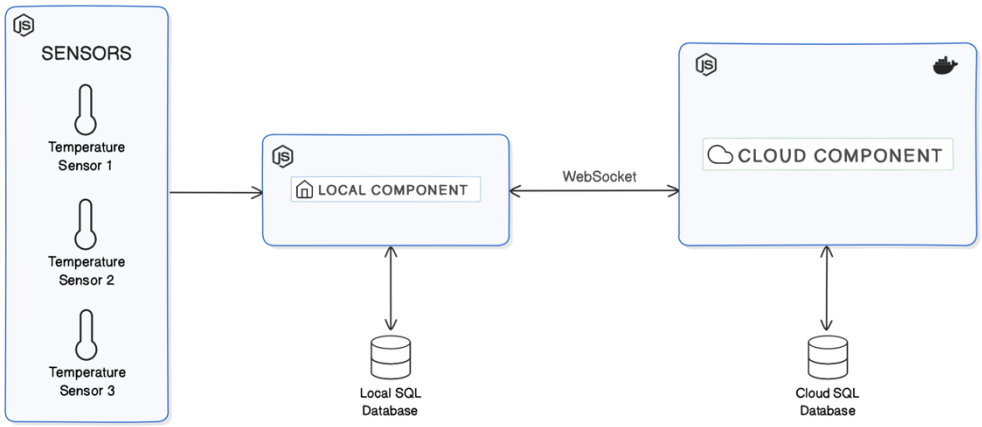
08.07.2024, Berlin

https://github.com/serdarakol/smart-factory-prototype

Abstract:

This projcect simulates a smart factory with 3 temperature sensors that monitor the temperature of a production step in a manufacturing plant where even slight temperature variations can impact the quality, safety, and effectiveness of the final product.

Architecture:



Cloud:

This Node.js script sets up a WebSocket server on port 3001 and connects to a SQLite database stored in `cloudData.sqlite`. It initializes two tables, `cloud_sensor_data` and `cloud_command_data`, to store sensor data and command data, respectively. The script defines several functions to handle incoming WebSocket connections from fog nodes, process received sensor data, generate commands, and send commands to connected fog nodes. It also periodically generates new commands and attempts to resend any unsent commands every 3 and 6 seconds, respectively. The script logs connections, disconnections, and errors, providing a basic framework for a cloud server that communicates with fog nodes in an IoT setup.

Local:

The local component sets up a fog node server using WebSocket and SQLite for handling sensor data and cloud communication. It creates a WebSocket server on port 3000 to receive sensor data, which is stored in an SQLite database. The server also connects to a cloud WebSocket server and attempts to reconnect every 5 seconds if

disconnected. Incoming sensor data is logged, stored in the database, and sent to the cloud server, updating the database to reflect whether the data was successfully sent. Additionally, the server processes any unsent data from the database at regular intervals. Commands received from the cloud server are also stored in the database.

Edge:

The Edge devices are in our cases 3 temperature sensors. The JavaScript code defines a `Sensor` class that uses WebSockets to simulate a sensor sending data to a specified local component URL. The `Sensor` class constructor initializes the sensor with a name, sensor ID, minimum and maximum value range, and a URL for the WebSocket connection. The `connect` method establishes a WebSocket connection and sets up event handlers for connection open, close, and error events. If the connection closes, it retries after 5 seconds. The `startGeneratingData` method initiates periodic data generation, sending random values within the specified range every 2 seconds. The `sendData` method sends the generated data via the WebSocket if the connection is open. The class ensures continuous data transmission and automatic reconnection in case of disconnection.

Starting the project:

Cloud

- make sure you have npm node installed in your vm instance

- cd smart-factory-prototype

- cd cloudServer

Option 1

- npm install

- node clousServer.js

Option 2

- docker build -t cloud-server

- docker tag cloud-server gcr.io/tu-berlin-fc-prototype/cloud-server

- docker run -d -p 3001:3001 gcr.io/tu-berlin-fc-prototype/cloud-server

Local

- cd localServer

- npm install

- node localServer.js

Sensor

- cd sensorSimulator
- npm install
- node index.js