Brief Report For Question-1 Solutions

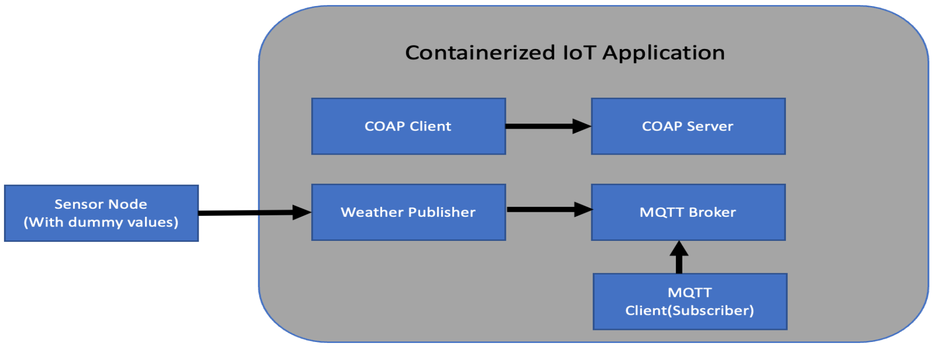
1- Source code for the sensor node (Python/JavaScript) and Dockerfile

The code for sensor node communication with both the CoAP server (coap_server.py), CoAP client (coap_client.py), and MQTT client (publish_weather.py) is available in the Iot_container/ directory along with corresponding Docker files(Dockerfile.coap-server, Dockerfile.coap-client, Dockerfile.publish and docker-compose file). These scripts are documented with code comments and include necessary URL modifications for containerized environments.

For scripts intended to run without containerization, the corresponding versions are located in the Iot_code/ directory.

I used an open-source MQTT broker (Mosquitto) to publish weather data from a sensor node. Since the actual sensor setup was unavailable, I used dummy values for temperature and humidity in the provided scripts.

These scripts located at Iot_code/ directory can be easily modified to incorporate actual sensor data capture and publish it to the MQTT broker. The Mosquitto broker was installed on a Mac using the command *brew install mosquitto*.



2- Instructions to set up and run the MQTT broker and CoAP server

COAP server was implemented using Python libraries. Code is available at Iot_code/coap_server.py and is documented using code comments.

Instructions to setup MQTT broker and COAP Server:

- Install mosquitto on Mac:

```
brew update
brew install mosquitto
```

- Start Mosquitto on separate terminal by running mosquitto command.
- Create ~/mosquitto.conf and include below lines for testing purpose in case there is connectivity issue from mqtt client.

  -
    ```
    listener 1883 0.0.0.0
    # Allow anonymous connections (for testing purposes)
    allow_anonymous true
    ```
- COAP server is setup using Iot_code/coap_server.py script. It needs required module to install to run it successfully. Script is documented using code comments.

3- Code for integrating with the chosen IoT platform (AWS IoT, Azure IoT Hub, or Google Cloud IoT)

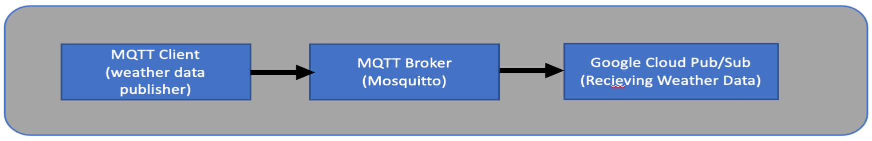
To connect the IoT system with Google Cloud, I utilized the Google Cloud Pub/Sub service to publish weather data from the MQTT broker. Since Google Cloud IoT has been retired, this alternative approach was necessary. To ensure the weather data was published using the MQTT protocol, I used the Mosquitto MQTT broker as a bridge between the MQTT client and the Google Cloud Pub/Sub service.

The integration code is available in `Iot_code/publish_weather_to_google.py`, and it is thoroughly documented with code comments.

Using Google Cloud Pub/Sub requires creating a project and setting up credentials, which are then used in the publisher to send sample weather data to Google Cloud Pub/Sub via the MQTT broker.

After running the `publish_weather_to_google.py` script in a separate terminal, you can publish data to the MQTT broker, which will then forward it to Google Cloud Pub/Sub, using the following command:

```
mosquitto_pub -t weather/data -m '{"temperature": 24, "humidity": 90}'
```


MQTT Client (weather data publisher)
MQTT Broker (Mosquitto)
Google Cloud Pub/Sub (Recieving Weather Data)

4- Smart contract code on Solana (Rust) and the script to interact with it

The smart contract for recording weather data on the Solana blockchain is implemented in lib.rs. The code is located in the `smart_contract/solana_smart_contract/src/` directory and is documented with code comments. It includes instructions for processing and logging received data as UTF-8 strings.

4.1- Recording Data Transactions:

- The Rust script `main.rs` demonstrates how to interact with the Solana smart contract. It includes functionality to send weather data to the smart contract and manage the required cryptographic and blockchain interactions.

4.2- Fetching Data:

- The Rust script `fetch_data_from_smart_contract.rs` shows how to connect to the Solana blockchain, fetch data from an account, deserialize it, and handle any potential errors.

All necessary dependencies for deploying the smart contract are specified in the `smart_contract/solana_smart_contract/cargo.toml` file.