Edge Computing in the IoT | Group Project Report **Fall 2023**

Authors: Alfio Vavassori, Cristiano Colangelo, Edoardo Riggio, Samuel Corecco

---

IoT Solar Station Due date: 21.12.2023

Contents

1. Abstract

The IoT Solar Station is an advanced weather station with an integrated self-positioning solar panel. The orthogonal 2-axis servo motors move the solar panel, while the attached voltmeter constantly measures the current power production. The solar panel moves at different angles and searches for the best one based on the voltmeter measurements. The onboard sensors send the collected data to a private MQTT broker, which forwards the data to a Python backend. The backend ingests the data into a DigitalOcean cloud instance hosting an ElasticSearch database via Docker containers. Finally, a Kibana dashboard deployed alongside the database showcases the collected data in nice, compact visualizations.

2. Hardware

Regarding the hardware used in this project, Figure 1 shows a complete overview of which elements have been used and how they were connected. The parts used where: two servos, a motor shield, a light intensity sensor, a voltmeter connected to a solar panel, an ambient sensor, and an Arduino Portenta H7. All the components were connected to the Portenta board using the I2C protocol. Please note that our delivered code, available in `src/arduino/arduino.ino`, has been well commented and cleaned up. Setup and loop code is well separated in the two homonoymous methods.

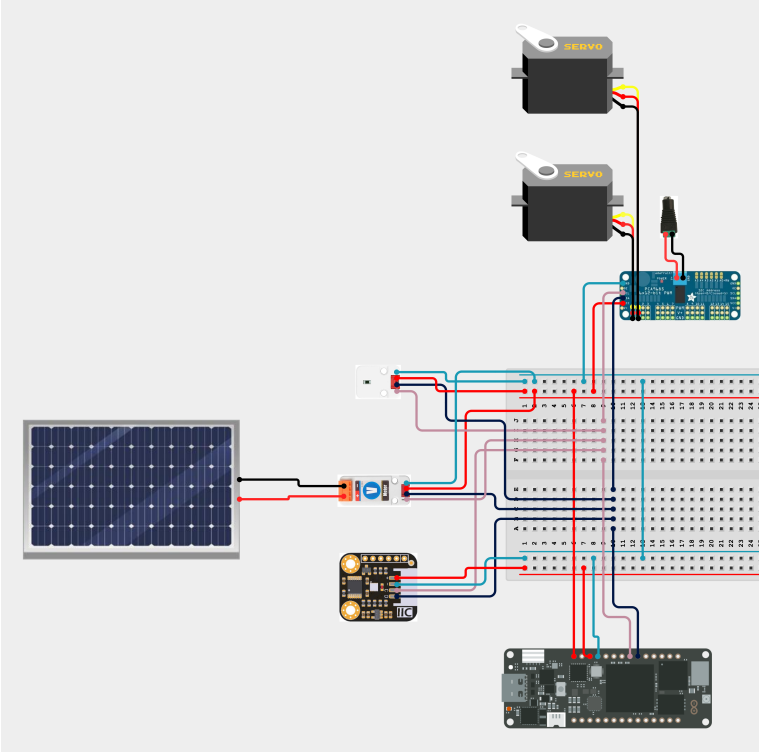

Figure 1: Overview of the hardware used in our project.

Servos

The two servos are the responsible for the movement of the solar panel. These are high-torque servos and are attached to a 3D-printed structure that holds the solar panel and moves it on two different planes. The bottom servo rotates the panel, while the top servo

adjusts the inclination of the panel.

To be able to interface with the Arduino board, we used a motor shield to which we connected both servos. Since they require some energy, the shield was connected to the wall outlet by means of a power supply cable.

This is our control algorithm for moving the servos in the best position:

```
// Initiate values and start servos movement loop
int best_servo_1 = 200;
int best_servo_2 = 300;
float curr_tension = read_tension();

// Perform scan only if current tension is < 95% of the last best
   tension, and it is daytime (tension > 0.15)
if ((curr_tension < 0.95 * max_value || max_value == 0) &&
   curr_tension > 0.15) {
    max_value = 0.0;

    for (int angle = angle_min_1; angle <= angle_max_1; angle +=
       ANGLE_INCREASE) {
        int pulse = map(angle, 0, 360, SERVOMIN, SERVOMAX);

        // Move bottom servo
        pwm.setPWM(SERVO_1, 0, pulse);

        for (int angle_2 = angle_min_2; angle_2 <= angle_max_2;
           angle_2 += ANGLE_INCREASE) {
            int pulse = map(angle_2, 0, 500, SERVOMIN, SERVOMAX);

            // Move top servo
            pwm.setPWM(SERVO_2, 0, pulse);
            delay(100);

            float current = read_tension();

            if (current > max_value) {
                best_servo_1 = angle;
                best_servo_2 = angle_2;
                max_value = current;
            }

            delay(500);
        }
    }

    angle_min_1 = best_servo_1 - ANGLE_INCREASE * 2;
    angle_max_1 = best_servo_1 + ANGLE_INCREASE * 2;

    int pulse1 = map(best_servo_1, 0, 360, SERVOMIN, SERVOMAX);
    int pulse2 = map(best_servo_2, 0, 360, SERVOMIN, SERVOMAX);

    // Move servos to best found position
```

```
    pwm.setPWM(SERVO_1, 0, pulse1);
    pwm.setPWM(SERVO_2, 0, pulse2);
}
```

Listing 1: The `get_direction` function keeps track of the 2 rotation axis and computes the direction.

As part of the telemetry data, we also added a small piece of code that keeps track of the orientation (NW = nord west, SO = south ovest, etc.). The IoT Solar Station should be positioned initially toward the north direction, since it is a closed-loop control algorithm for simplicity.

```
String get_direction(int servo_0, int servo_1) {
    String direction = "N";

    float tolerance = 115;
    float inclinations[3] = {300, 580, 640};
    float directions[4] = {150, 265, 380, 495};

    if (servo_0 >= directions[0] - tolerance && servo_0 <= directions
       [0] + tolerance) {
        direction = "N";
    } else if (servo_0 > directions[1] - tolerance && servo_0 <=
       directions[1] + tolerance) {
        direction = "NW";
    } else if (servo_0 > directions[2] - tolerance && servo_0 <=
       directions[2] + tolerance) {
        direction = "W";
    } else if (servo_0 > directions[3] - tolerance && servo_0 <=
       directions[3] + tolerance) {
        direction = "SW";
    }

    if (direction == "N" && servo_1 > inclinations[1]) {
        direction = "S";
    } else if (direction == "NW" && servo_1 > inclinations[1]) {
        direction = "SE";
    } else if (direction == "W" && servo_1 > inclinations[1]) {
        direction = "E";
    } else if (direction == "SW" && servo_1 > inclinations[1]) {
        direction = "NE";
    }

    return direction;
}
```

Listing 2: The `get_direction` function keeps track of the 2 rotation axis and computes the direction.

Ambient Sensor

Attached to the system, there is an ambient sensor which will capture several different measurements. These measurements are temperature, pressure, humidity, and light intensity.

These measurements are not used to drive the best-position-seeking algorithm for the solar panel, but are used to populate the weather station's dashboard.

Light Intensity Sensor

As with the ambient sensor, the light intensity sensor does not serve a real purpose towards the computation of the best position. The data taken from this sensor is used in the weather station dashboard.

Voltmeter

The system must actively adjust the solar panel in response to the position of the sun, adjusting its position to maximize energy absorption. This is done with two main actions, searching for the optimal position and scanning the various positions and angles.

Our model must also take into account times of low solar intensity such as cloudy conditions or night. In these scenarios, our system enters an energy-saving phase. The system will only capture sensor data, and as long as the produced voltage does not exceed 1V, the system will not search for new positions (Figure 2).

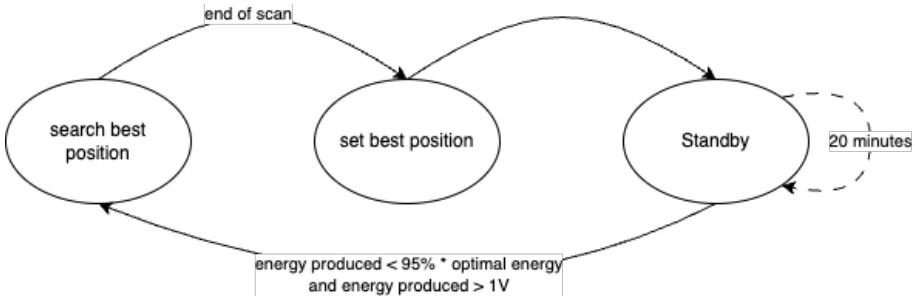

Figure 2: State machine that describes the lifecycle of our system.

3. Communication

4. Communication

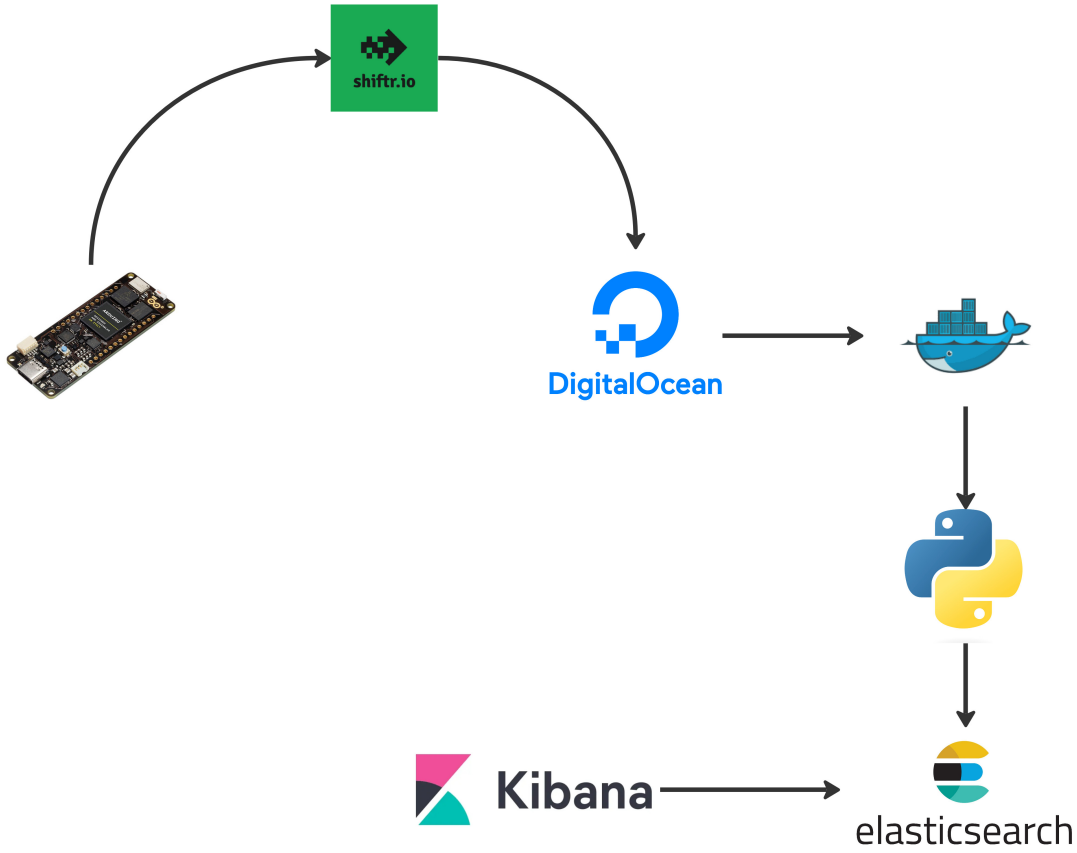

Figure 3: The IoT Solar Station deployment architecture.

For our project, we decided to implement communication with an MQTT (MQ Telemetry Transport) broker, which allows us to send lightweight, continuous messages to our cloud instance. We used the service shiftr.io, a high-performance broker that easily connects devices to the same network (Figure 4).

To implement communication, we used a MQTT client library in both the Arduino Portenta and our Python backend. The MQTT client on the Arduino board will send a message after every calibration phase. The client emits a JSON message on a defined topic, and the Python backend will subscribe to that topic. When a message is sent from the Arduino, the Python server will receive the message which will then be sent to our ElasticSearch.

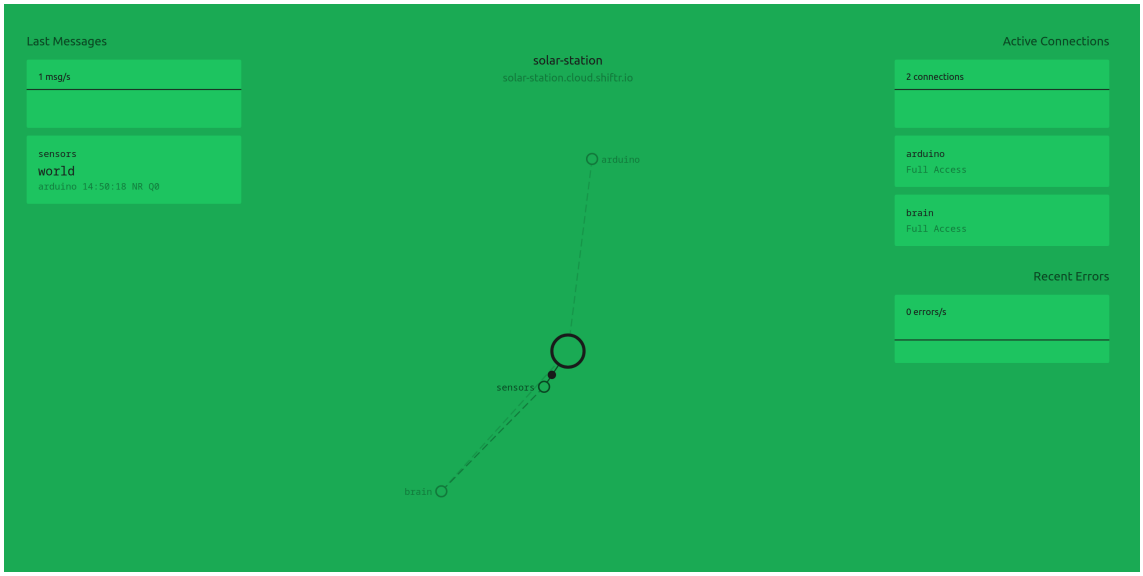

Figure 4: Visualization of our MQTT network through the shiftr.io interface. The "arduino" dot represents our Arduino Portenta device, the "brain" dot represents our Python backend, and the "sensors" dot represents the topic where messages are published.

In brief, we used the Arduino WiFi library to connect to the Internet. We had troubles with USI's WiFi network, as it was necessary to authenticate, so we simply used our mobile hotspots. We used the Python library Eclipse Paho [?] to make our Python client subscribe to the topic, which was trivial. The respective library on the Arduino side is Arduino MQTT by Joël Gähwiler and contributors [?]. We also separated the configuration for both the Arduino and the Python backend so that secrets and individual configuration of the access point are not shared in the git repo. The latter is done by importing the `config.h` and `config.ini` files for the Arduino and Python programs respectively.

```
void send_data(float temperature, float altitude, float humidity,
   float light, float energy, String direction) {
    StaticJsonDocument<200> doc;
    doc["t"] = round(temperature);
    doc["a"] = round(altitude);
    doc["h"] = round(humidity);
    doc["l"] = round(light * 100);
    doc["e"] = round(energy * 100);
    doc["d"] = direction;

    String body;
    serializeJson(doc, body);
    char char_array[body.length() + 1];
    body.toCharArray(char_array, body.length() + 1);

    Serial.println("Sending to server:");
    Serial.println(body);
    Serial.println(char_array);

    // Send data to the MQTT server
    client.publish("/sensors", body);
```

```
}
```

Listing 3: Snippet of our `send_data` function which serializes our sensor data into JSON and sends it over the MQTT broker.

```
@staticmethod
def on_message(client, userdata, msg):
    global es
    print(f'{msg.topic} {msg.payload}')

    data = json.loads(msg.payload)
    data['@timestamp'] = datetime.now(ZoneInfo('Europe/Rome')).
       isoformat()

    es.index(index=msg.topic, body=data)
```

Listing 4: Snippet of our `on_message` function which is triggered every time the MQTT server receives a message from the broker. It ingests the data into ElasticSearch.

5. 3D structure

Cloud setup

For our cloud setup, we used Docker [?] containers running in a DigitalOcean [?] instance (Figure 6). We decided to install ElasticSearch [?], a RESTful search and analytics database that works well with its dashboard builder, Kibana [?]. The setup was relatively straightforward, but we had some troubles at first because we discovered ElasticSearch is a quite memory-intensive program, therefore we had to resize our cloud instance to have more RAM available, as well as research how to allocate more memory for Docker containers.

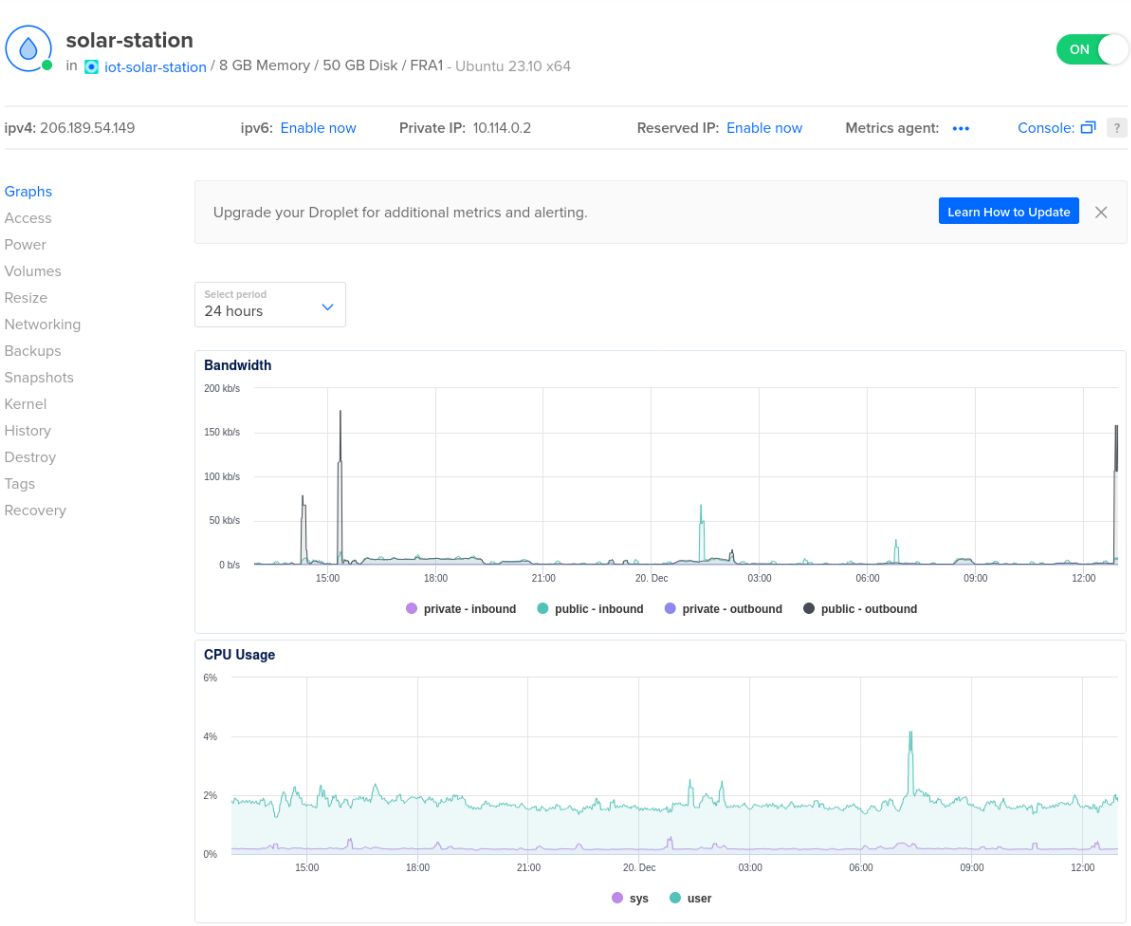

Figure 5: A sneak peek of the DigitalOcean interface showing some telemetry about our cloud instance.

This allowed us to develop faster in the end because we had a shared database for measurement. We could collaborate more easily asynchronously as we had the same development environment for everyone. It is also a more realistic setup, although no HTTPS certificate was installed. We also had to specify the necessary port-forwarding rules in the intuitive DigitalOcean web interface to allow outside connection.

A list of the steps undertaken for this part can be found in file `scripts/es_setup.sh` in the delivered zip.

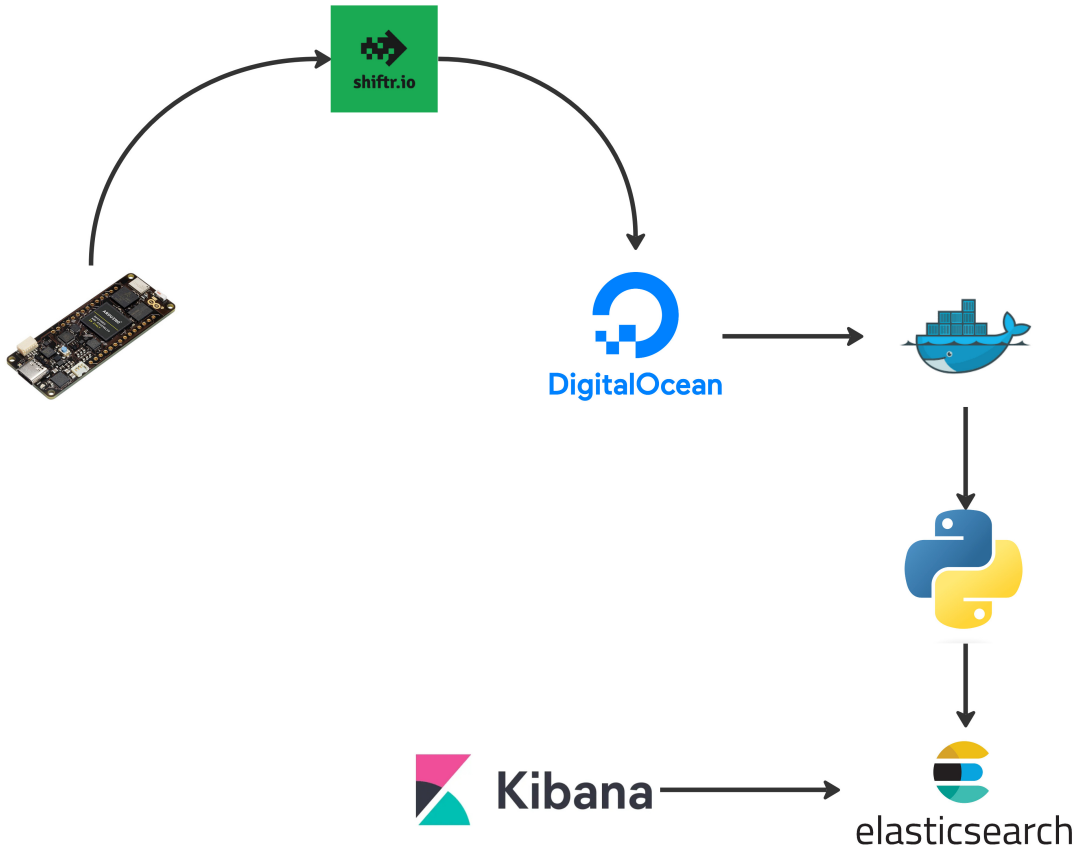

Figure 6: A simple diagram showing the deployment architecture of our application.

6. Dashboard

To nicely visualize all the data we collected through the MQTT broker, we decided to build a Kibana dashboard since it integrates seamlessly with our ElasticSearch installation and provides a wide range of visualizations. We visualize all the data we gather and plot them in linecharts and gauges with unit of measurement as well. Please see the below screenshot.

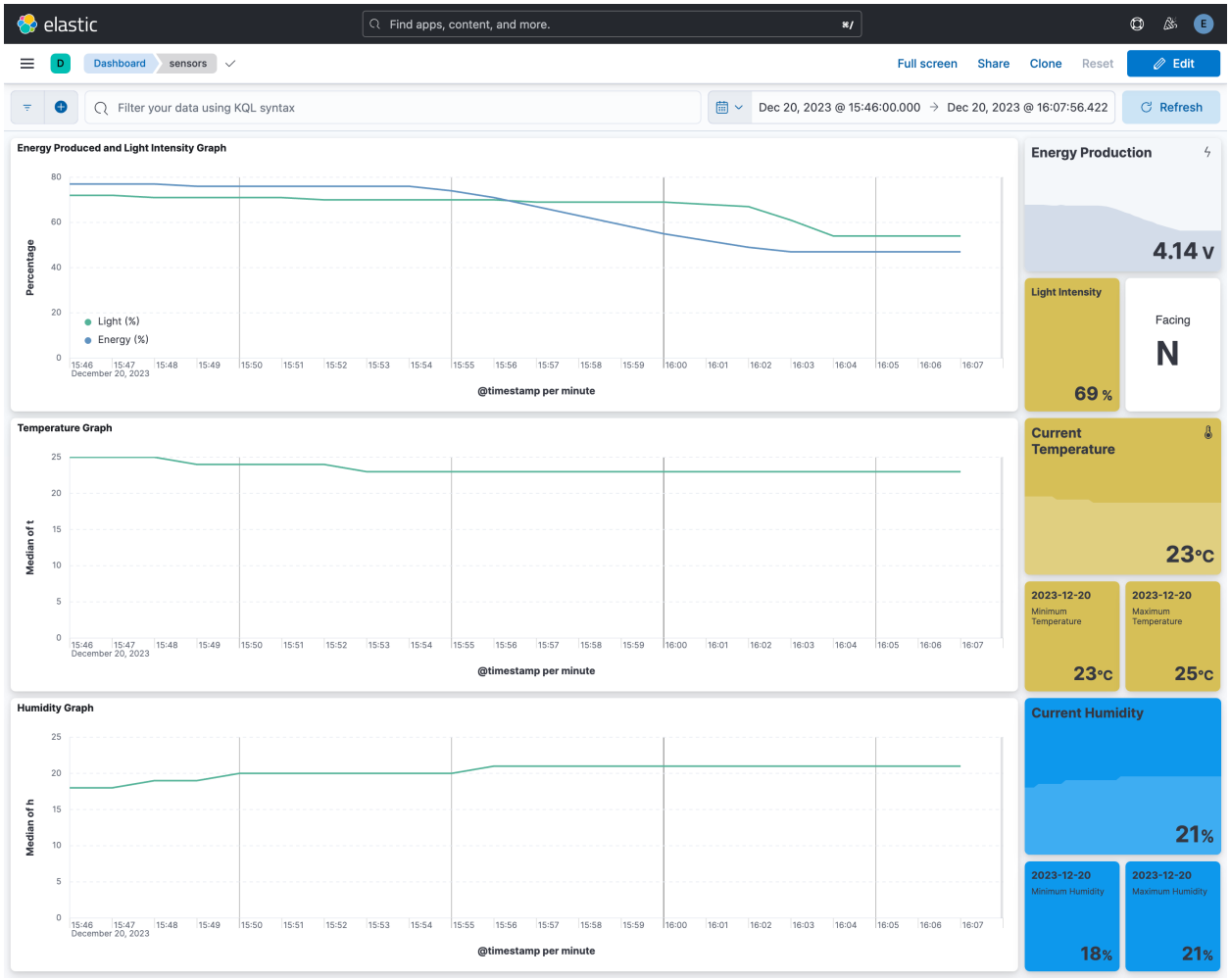

Figure 7: The IoT Solar Station Kibana Dashboard.