Smart Stress Detection in and through Sleep

Software Engineering for the Internet of Things Project

UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA

European Commission
ERASMUS
MUNDUS

University of L'Aquila

Submitted: 25 - 01 - 2024
Submitted to: Prof. Davide Di Ruscio
Written by: Quazi Ghulam Rafi, Ruben Huygens, Bouhlal Hiba

Table of Contents

Overview

Our smart stress detection system provides real-time monitoring of various health metrics collected in and through sleep to indicate stress levels to the users. The system collects data from multiple sensors and visualizes it on a user-friendly Grafana dashboard. Specifically, the system tracks and visualizes snoring range, limb movement rate, respiration rate, hours of sleep, body temperature, eye movement, heart rate, blood oxygen level, and stress levels.

Users need to log in to the Grafana dashboard to view their health metrics related to their sleeping patterns. The dashboard refreshes autonomously to display real-time data, so users don't need to do anything else. Additionally, the displayed data includes time series, allowing users to check their metrics over various periods.

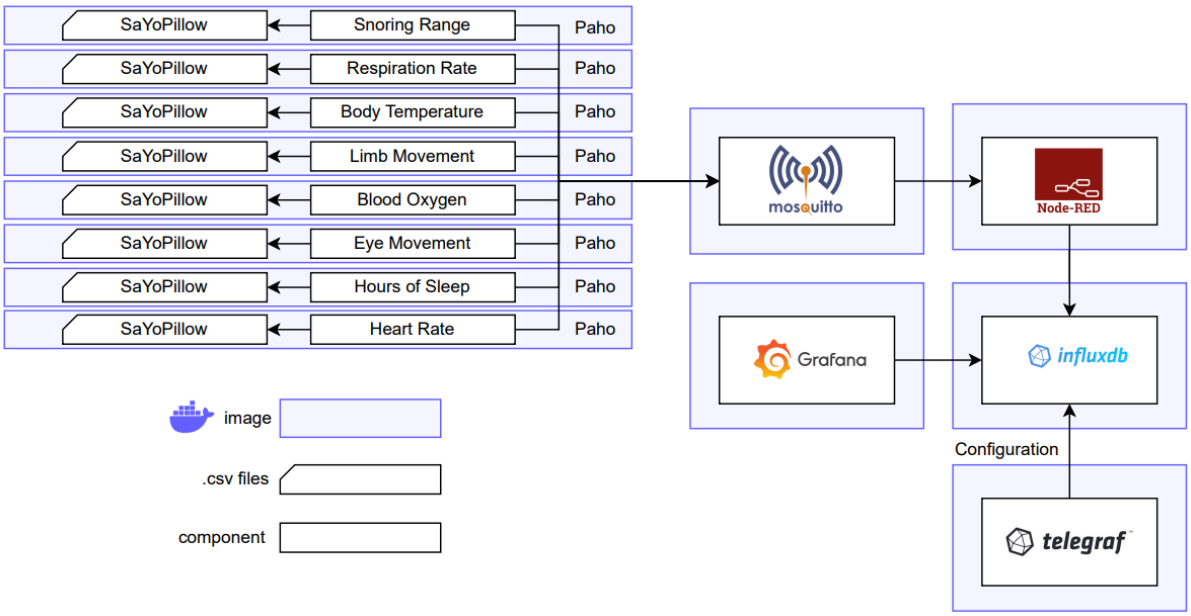
This system is dedicated to the movement towards a healthier lifestyle. Nowadays, fitness bands help people maintain a more active lifestyle. However, quality sleep is just as important as regular exercise for maintaining a healthy body. Thus, through this system, people can better understand their sleeping patterns and adjust their lifestyles to achieve healthier bodies. This system can also be highly appreciated by healthcare providers, as having a good understanding of patients' sleeping patterns can assist in diagnosing various health complexities.

Technologies

We used an MQTT broker, Telegraf, InfluxDB, Node-RED, and Grafana to publish the collected data and visualise it. All the services are dockerized using Docker Compose and communicate with each other inside the Docker network. The system can be run using a simple command: "docker-compose up". This command runs all the services, including the Python code, inside Docker. Docker is used to create volumes for InfluxDB and Grafana to store data for dashboards and panels.

System Architecture

The collected data from various sensors are sent to MQTT through Paho, where they are accessed by Node-RED and forwarded to InfluxDB using configurations provided by Telegraf. InfluxDB stores the data, while Grafana queries and retrieves data from the database to visualize it on several dashboard panels.



Data Preparation

The simulated data (SaYoPillow) was sourced from kaggle and contains 630 rows of snoring rate, respiration rate, body temperature, limb movement rate, blood oxygen levels, eye movement, hours of sleep, heart rate, and stress level.

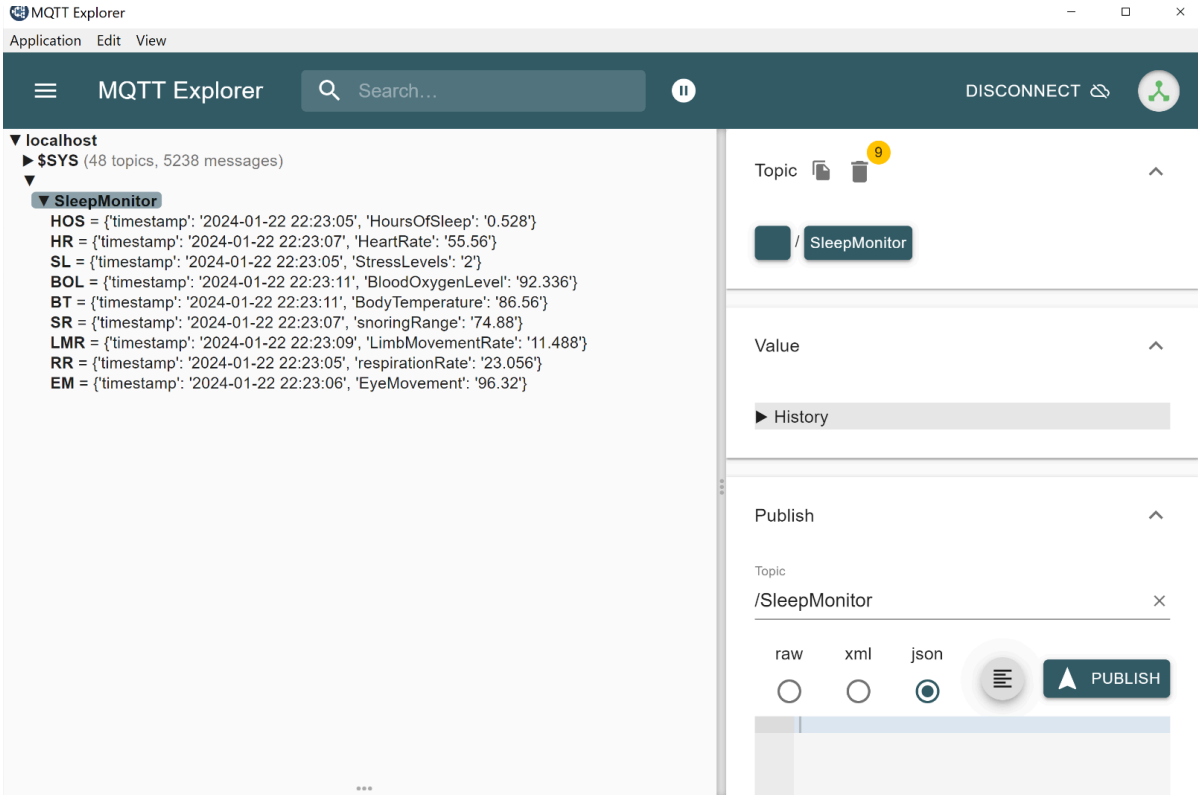
Python Paho Client

Our code contains 9 Python clients running with the Mosquitto Paho client library to simulate data coming from 9 different sensors. We divide the respective sensor's data from the initial SaYoPillow dataset to create a dataset related to each sensor and place the created dataset, along with a Python file that functions as the MQTT publisher, in a folder specific to the sensor. For example, we retrieve data related to blood oxygen levels from the SaYoPillow dataset to create a "BloodOxygenLevels.csv" file, and we create a Python file named "BloodOxygenLevels.py". These are placed in a folder named "BOL". Similarly, we created 8 other folders for snoring rate, respiration rate, body temperature, limb movement rate, eye movement, hours of sleep, heart rate, and stress level.

The Python MQTT publishers read their respective datasets and send data to their respective topics at one- to five-second intervals. The topics are defined

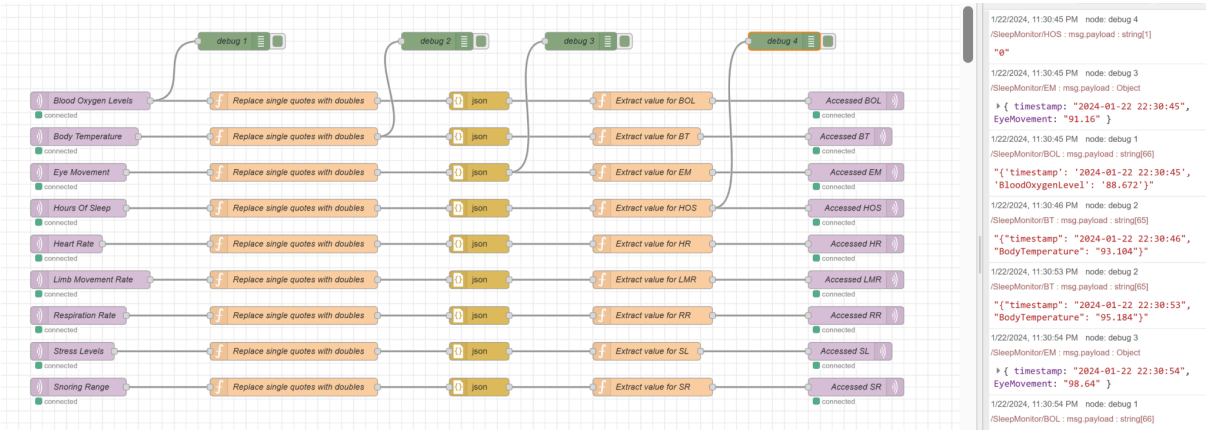
according to their sensor type. For example, blood oxygen levels are published to "/SleepMonitor/BOL". Similarly, snoring rate, respiration rate, body temperature, limb movement, eye movement, hours of sleep, heart rate, and stress level are published to "/SleepMonitor/SR", "/SleepMonitor/RR", "/SleepMonitor/BT", "/SleepMonitor/LMR", "/SleepMonitor/EM", "/SleepMonitor/HOS", "/SleepMonitor/HR", and "/SleepMonitor/SL", respectively. The code is dockerized, allowing it to be easily run on any machine. The Dockerfiles facilitate the running of the Python MQTT publisher files and are also included in the main docker-compose file. The connection between the sensors and the Mosquitto broker has been established in the docker-compose file and through the docker network.

MQTT explorer makes it possible to intercept data that passes from the sensors to Node-RED, as can be seen below.



Node-RED

Node-RED accesses the published data and, for each sensor's data, replaces single quotations with double quotes and converts the data into JSON format. Next, from these JSONs, the values are extracted and subsequently made available for storage in the database.



Data storage

The data passed by Node-RED is sent to InfluxDB. InfluxDB is, as they describe themselves, a "fully managed, performant, and elastic time series database service". This allows us to store data from individual sleep monitoring sensors in an accessible and performant way. Below, all the sensors' data is displayed in the InfluxDB web interface.

To set up InfluxDB, the docker container 'influxdb' runs the file 'entrypoint.sh'. This file sets up the InfluxDB's username, password, IP-address, and port using environment variables defined in the .env file.

Access

To access the InfluxDB database, the project should be started (with docker-compose up). Afterwards, 'localhost:8086' can be entered in the browser. This will prompt the user with an authentication screen. For the username, enter 'admin', and for the password, enter 'admin123'.

InfluxDB bucket

When running the project for the first time, the user will have to set up a new bucket with the name 'SleepMonitor'. Although this process can be automated, following InfluxDB's documentation did not lead to an automated bucket creation.

When the user is logged into the InfluxDB's dashboard with the correct bucket in place, they will be able to click on the graph symbol in the left panel, select the bucket and select the only available options until they reach the last column. From here, they can select a sensor's data they would like to view and press 'submit' to see the graph above.

Grafana

The data stored in the InfluxDB database can be viewed in grafana in the form of graphs. Grafana is an application that facilitates easy graph creation, and allows users to view real-time updates to these graphs based on incoming data. Below, all the sensors' data is displayed in the web interface.

The setup details of Grafana cannot readably be viewed in the project's source code, as this dashboard is entirely set up in the browser under port 3000.

Access

To access the Grafana dashboard, the project should be started (with docker-compose up). Afterwards, 'localhost:3000' can be entered in the browser. This will prompt the user with an authentication screen. For the username, enter 'admin', and for the password, enter 'admin'. The resulting form can be skipped below the input fields.

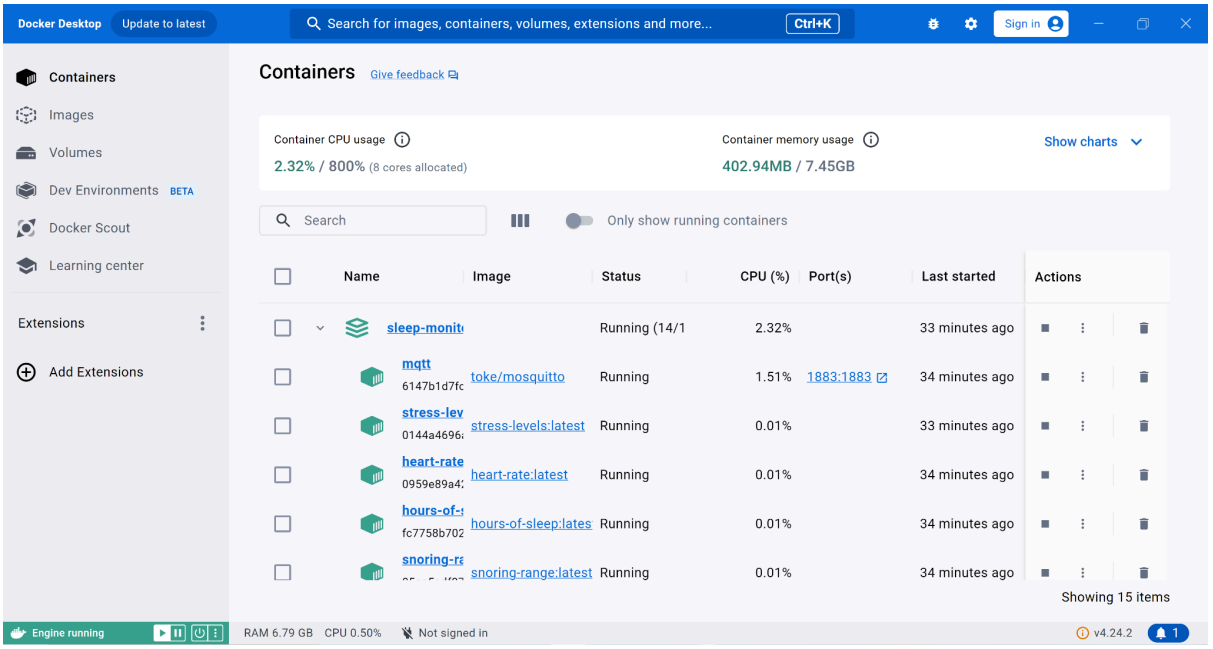
Docker-compose

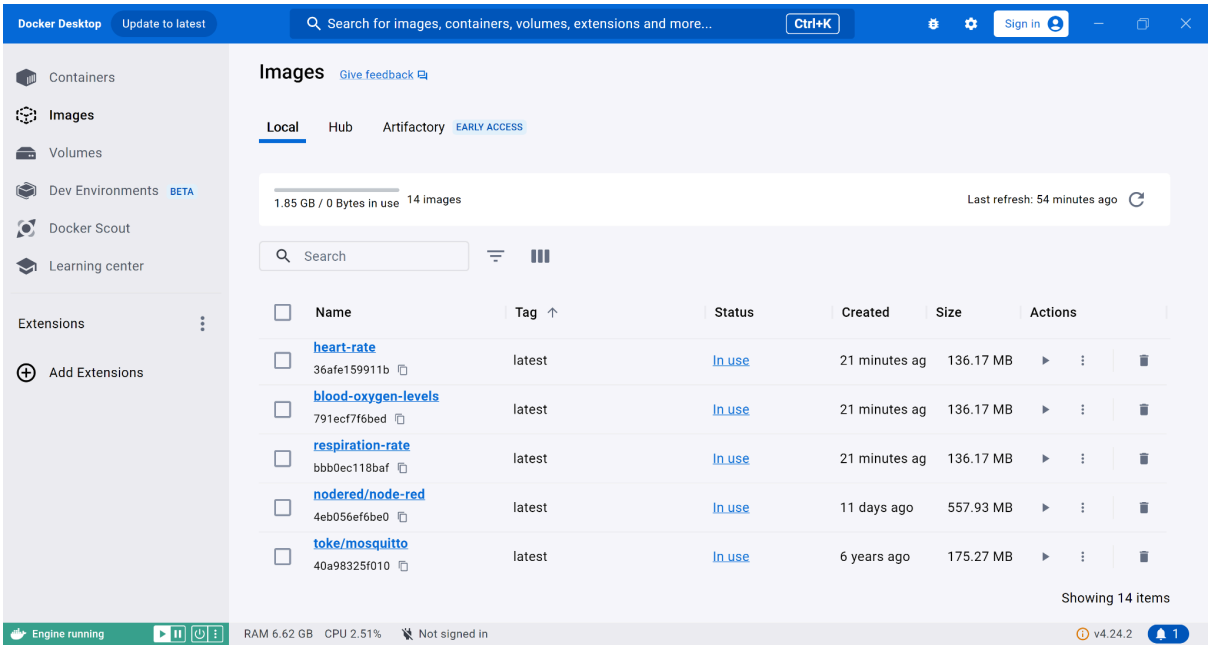
Docker-compose allows the execution of multiple containers with one command. Containers in our project's docker-compose.yml file are defined for the following services:

- Publishing sensors
    - Snoring range
    - Stress level
    - Respiration rate
    - Limb movement rate
    - Heart rate
    - Hours of sleep
    - Eye movement
    - Body temperature
    - Blood oxygen levels
- Grafana
- Telegraf
- InfluxDB
- MQTT

- Node-RED

Docker Desktop makes it possible to easily see which containers and images are running, as can be seen below.





Publishing sensors

Each publishing sensor has an associated couple of docker-compose.yml lines.

- 'container-name' specifies the name of the container as can be seen in Docker desktop. For example, the snoring range sensor runs in container 'snoring-range'
- 'build' allows us to specify how the docker container should be built

    - ‘context’ specifies the folder in which files relating to this container can be found. As the publishing sensors are stored in the ‘Docker’ folder, where each sensor has a folder named an abbreviation of the type of data the sensor senses, the context for, for example, snoring range is ‘./Docker/SR/’
    - ‘dockerfile’ specifies the path to the Dockerfile of this container. Since all sensor’s Dockerfiles are in their associated folder, all that needs to be written is ‘Dockerfile’
- ‘image’ specifies the name of the images, which were named the same name as the images’ container names by convention, postfixed with ‘:latest’.
- ‘restart’ specifies what happens when a container exits. This would be comparable to a crash within the sensor, which is why this is set to ‘always’ for all sensors; they should always try to restart upon a crash.
- ‘depends_on' specifies which services must be started before other services. In this project, the MQTT container must be started before the sensors.
- ‘networks’ specifies the network in which the docker containers find themselves. The network is called ‘sleep-monitor-network’
    - Below ‘sleep-monitor-network’ an ‘ipv4-address’ can be set, which can be used to access containers. All sensor’s IP addresses start with ‘172.30.0.1’, while the last two digits differ.

Other containers

- ‘image’ specifies the name of the image pulled from the internet. This project uses the official Grafana ‘grafana/grafana-oss:8.4.3’, Telegraf ‘telegraf:1.19‘, ‘influxdb:2.7.4‘, Node-RED ‘nodered/node-red’, and the unofficial MQTT ‘toke/mosquitto‘. All images are open-source.
- ‘volumes’ makes the persistency of changes during the running process possible. For example, setting ‘volumes’ for grafana to ‘/var/lib/grafana’ makes it possible to set up graphs in the web interface, and remember these graphs for the next time the project is running. Postfixing ‘:rw’ specifies the permissions the service has to this file.
- ‘depends_on’ specifies the service starting order. In this project, ‘depends_on’ is used to ensure that influxdb and mqtt start before Node-RED, Grafana, Telegraf, and the sensors.
- ‘env_file’ specifies the environment variable file used for each container. All ‘env_file’ tags are set to the project ‘.env’ file.
- ‘Entrypoint’ specifies the files that should be executed when the service starts up. For influxdb, the provided file initializes the service with, for example, a username and password.
- ‘ports’ maps a host-machine’s port to the port within the docker container. The host-machibe’s ports are specified as environment variables in this

project. For example '${DOCKER_INFLUXDB_INIT_PORT}:8086' maps the host-machine's port to port 8086 of the influxdb container. The user can access the influxdb web interface at localhost with the port specified in the '.env' file.

- 'networks' and 'container_name' work the same way as for the sensors.