Smart Agriculture - Field Monitoring System

Software Engineering for the Internet of Things

2024/2025

Students

- **Stefano Decina**,
  - matr: **295433**,
  - e-mail: stefano.decina@student.univaq.it
- **Alexander Mattei**,
  - matr: **295441**,
  - e-mail: alessandro.mattei1@student.univaq.it

Table of Contents:

Introduction

The **Smart Agriculture - Field Monitoring System** project is a system designed to improve agricultural management through the use of IoT technologies. The goal is to develop a solution that enables real-time monitoring of key soil parameters and environmental conditions, providing farmers with tools to optimize resource use, prevent problems related to suboptimal conditions, and improve agricultural productivity in a sustainable manner.

The system relies on advanced IoT sensor simulation to collect data on soil moisture, temperature, soil pH, water salinity, relative humidity and precipitation. Through data analysis and automatic alert generation, the system supports data-driven decisions for modern agricultural management.

An additional strength is the ability to easily configure and change monitoring thresholds, and the immediate receipt of alerts if they occur allows for constant control over essential parameters and timely action for efficient and sustainable management of the agricultural environment.

Benefits of using the system

1. Optimization of the use of water resources.

With soil moisture sensors built into the system, farmers can tailor irrigation to actual crop needs. This significantly reduces water waste and ensures more sustainable use of water resources, even in scarcity environments.

2. Improved productivity

Continuous monitoring of key parameters such as soil pH and water salinity helps maintain optimal conditions for crop growth. This results in increased crop yields, supporting farmers in obtaining higher quality productions.

3. Reduction of manual interventions

By centralizing data, the system reduces the need for frequent manual checks. Farmers can intervene only when sensors detect abnormal conditions, saving time and operational resources.

4. Prevention of agricultural hazards

The system's ability to detect water stress conditions, pH imbalances or excessive rainfall in real time allows farmers to take timely action to prevent crop damage. Automatic alerts improve the ability to prevent losses and ensure more proactive management.

5. Environmental sustainability

The system promotes environmentally sustainable agricultural management by reducing overuse of resources and minimizing environmental impact. For example, optimal irrigation management reduces the risk of soil salinization and limits soil erosion.

6. Intuitive interface for advanced management

The ability to configure alert thresholds and monitor data through a user-friendly interface, supported by automatic notifications on platforms such as Telegram, makes the system

accessible even to users with limited technical skills. This facilitates adoption of the technology by a wide range of farms.

These benefits, which are closely linked to the functionality offered by the **Smart Agriculture - Field Monitoring System**, demonstrate how the project represents a concrete solution for improving the efficiency and sustainability of agricultural activities.

Sensors simulated in the system

1. Soil moisture sensor

Description: Measures the water content in the soil.

Simulated output: Moisture percentage (%), with typical values between 0% (dry) and 100% (saturated).

Application: Determine when it is necessary to irrigate the field.

Example of values:

Humid: 50%-70%

Dry: <30%.

Saturated: >70%

2. Temperature sensor

Description: Measures ambient temperature to assess weather conditions.

Simulated output: Temperature in °C.

Application: Control temperature to prevent heat stress on plants.

Example of values:

Temperate climate: 15°C - 30°C

Thermal stress: <10°C or >35°C.

3. Soil pH sensor

Description: Measures the acidity or alkalinity of the soil.

Simulated output: pH values (0 to 14).

Application: Determine whether the soil is suitable for specific crops or requires corrections.

Example of values:

Neutral: pH 6.5-7.5 (optimal for most crops).

Acid: pH <6.5.

Alkaline: pH >7.5.

4. Water salinity sensor

Description: Measures the concentration of salts in water used for irrigation.

Simulated output: Electrical conductivity (EC) in µS/cm or dS/m.

Application: Avoid the use of excessively saline water that could damage crops.

Example of values:

Fresh water: <0.7 dS/m.

Moderately saline: 0.7-2 dS/m.

High salinity: >2 dS/m.

5. Relative humidity sensor

Description: Measures the amount of water vapor in the air relative to the maximum capacity.

Simulated output: percentage of relative humidity (%).

Application: Monitor weather conditions to optimize irrigation.

Example of values:

Dry: <30%.

Humid: 50%-80%.

Saturated: >90%.

6. Rain detection sensor

Description: Detects the presence of rain and its intensity.

Simulated output: Binary values (rain: yes/no) or mm of rain.

Application: Turn off irrigation in case of rainfall.

Example of values:

No rain: 0 mm.

Light rain: 1-5 mm.

Heavy rain: >10 mm.

Functional Requirements

- **Simulation setup** (number of agricultural fields, type of sensors present in goni field and measurement interval) **via Telegram bot** resulting in automatic setup of the whole system including Grafana dashboards.
- **Monitor IoT devices** serving the land, in particular:
    - Soil monitoring through dedicated sensors:
        - Soil moisture
        - Soil Ph
    - Monitoring of weather conditions using sensors from:
        - Ambient temperature
        - Ambient humidity
        - Rainfall level

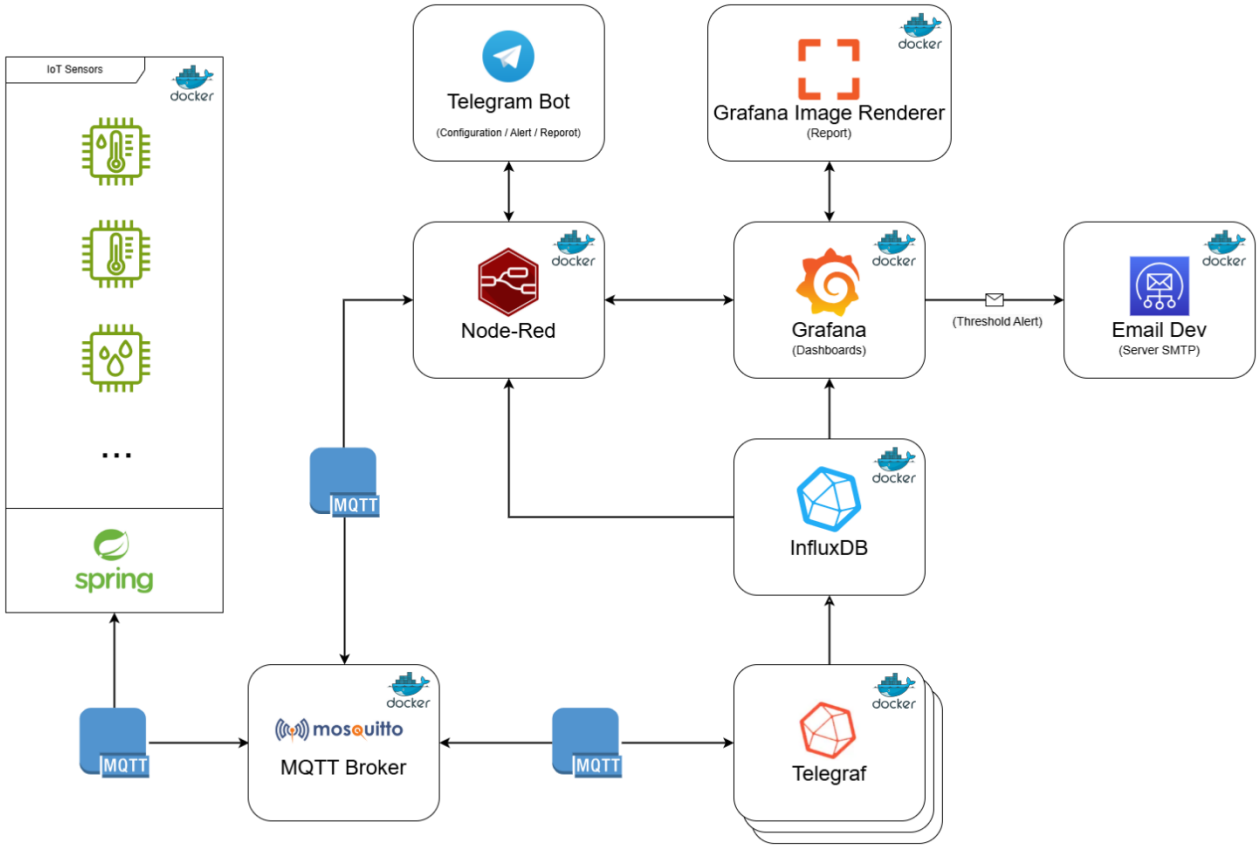
    - Water salinity monitoring.
- **Record historical values** of components for the purpose of:
    - Optimize resource management and improve the efficiency of agricultural activities.
    - Identify suboptimal conditions for crops, enabling timely and targeted interventions.
    - Support data-driven decisions for:
        - Optimize the use of water resources,
        - Preventing crop damage,
        - Increasing agricultural productivity in a sustainable way.
    - Report anomalies **by sending summary reports and** dashboard **images** via Telegram Bot and a backup email alert system run directly from Grafana.
- **Configuration and modification of sensor alert thresholds via** simple and intuitive **Telegram Bot** editing.

Non-Functional Requirements

- Portability:
    - The architecture supports modularity with Dockerized containers for each component, enabling scalability and independent deployment of services such as MQTT, Telegraf, and Node-red, facilitating horizontal and vertical scalability.
- Scalability:
    - The system allows customization of agricultural field simulation parameters (e.g., number of fields, types of sensors installed on individual fields, starting environmental conditions) either through a Telegram bot or through commands on a Node-Red flow.
    - Changing the simulation parameters will also be possible at Runtime, and the system will also be able to simulate different simulation conditions. The parameters that can be set at Runtime will be:
        - Agricultural fields:
            - Number of fields to be simulated
            - Sensors installed for each field
            - Sensor measurement range
        - Global Weather:
            - Ambient starting temperature
            - Starting ambient humidity
            - Weather Condition (*SUNNY*, *CLOUDY*, *LIGHT_RAIN, MODERATE_RAIN*, *HEAVY_RAIN*, *HURRICANE*)
        - Alert thresholds for each sensor type:
            - Minimum
            - Maximum
    - A dashboard is created for each field that includes all the information from the sensors in it
- Resilience:
    - Telegraf will be configured with two agents to ensure redundancy and continuous data acquisition, even in case of partial system failures.

- User friendly design:
  - A Telegram bot is implemented to manage all simulation operations (e.g., start simulation, change parameters, ...), edit and view information about the generated alerts
  - The project integrates Grafana by creating visual dashboards automatically for monitoring sensor data, system status and real-time alerts.
  - The system monitors the threshold values set and will dynamically change the user interface with colors and scales to provide a better user experience.
- Security:
  - The system includes authentication mechanisms for MQTT and Node-Red using username-password pairs to limit unauthorized access.
  - The credentials of the various systems are configured through environment variables to avoid exposing sensitive data in the code.
  - The use of Telegraf agents ensures the isolation of data flows, reducing the risk of data manipulation or unauthorized access.

System Architecture



The architecture of the **Smart Agriculture - Field Monitoring** System is structured to ensure modularity, scalability and flexibility. It is based on a containerized implementation of the main components and a data flow managed via MQTT protocols. The architecture is described in detail below, highlighting its main components.

1. IoT Sensors (Spring Framework)

Component Structure:

The **IoT Sensors** component is implemented in **Java** using the **Spring Boot** and **Spring Integration** frameworks to handle asynchronous communication and the **MQTT** protocol, via the **Eclipse Paho MQTTv5** library. The solution is designed to simulate the behavior of IoT sensors in an agricultural context, enabling real-time configuration and monitoring.

Main Features:

Simulation of Sensors

Each sensor is dynamically simulated, generating realistic data based on configurable weather conditions and parameters.

Supported sensors include:

- Temperature (e.g. outdoor temperature detection).
- **Relative humidity** (values in percent).
- **Soil moisture**.
- **Soil pH**.
- **Salinity of water**.
- Rainfall (rainfall intensity).

Simulation Configuration

The system supports dynamic configuration via MQTT messages received on specific topics:

- *sensors/simulation/config*: Allows setting the number of agricultural fields, the active sensors for each field and the simulation interval.
- *sensors/simulation/update*: Allows you to update the weather conditions and/or simulation interval.
- sensors/simulation/start: Starts the simulation.
- sensors/simulation/stop: Stops the simulation.
- *sensors/simulation/config/get*: Requests the current configuration.
- *sensors/simulation/config/current*: Publishes the current simulation configuration, including:
  - Climate status (weather conditions, outdoor temperature, relative humidity).
  - Number of agricultural fields.
  - Simulation interval.

Publication of Data

Data generated by sensors are posted on MQTT topics, organized by field and sensor type, for example:

- *agriculture/field1/temperature*
- *agriculture/field2/soilMoisture*

Technical Components

MqttGateway

It manages the reception of MQTT messages and routes requests to the respective logical channels. Key features include:

- **Simulation configuration management**: Decodes and applies the settings received on *sensors/simulation/config*.
- **Climate condition update**: Processes data received on *sensors/simulation/update*.
- **Management of start/stop commands**: Performs operations to start or stop the simulation.
- **Publication of current status**:
    - Send current configuration to *sensors/simulation/config/current* including details on:
        - Weather conditions.
        - Sensors configured and active.
        - Simulation interval.

Simulation

It represents the heart of the simulation system:

- Maintains the current state of the simulation, including configured fields and climate context.
- Schedule periodic tasks to generate sensor data according to the configured interval.
- Supports dynamic task recalculation when conditions or intervals are changed.
- It handles the propagation of simulated data via the MQTT publisher.

MqttPublisher

It is responsible for sending MQTT messages. It uses a dedicated channel to publish sensor-generated data or simulation status.

MqttConfiguration

Configures the MQTT infrastructure, including clients, input/output channels, and topic routers.

Simulation Process

1. **Initial configuration**:
    - At startup, a default configuration is created with two agricultural fields and all sensors enabled.
    - The simulation interval is set to **5000 ms** by default.

2. **Starting the simulation**:
    - The simulation is started by the MQTT command on *sensors/simulation/start*.
    - Periodic tasks are scheduled for each field, generating data from the configured sensors.
3. **Data generation**:
    - For each field, active sensors generate data based on current weather conditions.
    - The data are sent to the corresponding topic via MQTT.
4. **Dynamic update**:
    - Weather conditions or simulation interval can be updated dynamically via MQTT messages on *sensors/simulation/update*.
5. **Publication of the current configuration**:
    - Any changes to the configuration (e.g., climate updates, new configurations, etc.) are notified on *sensors/simulation/config/current*.
    - Other affected systems can use this information to synchronize with the current state of the simulation.
6. **Stopping the simulation**:
    - The simulation can be stopped by the MQTT command on *sensors/simulation/stop*.
    - All scheduled tasks are cancelled

Error Management

- Errors in message decoding or simulation processing are handled and recorded in the logs.
- A dedicated error channel is configured to handle any processing problems.

Benefits of Design

- **Scalability**: The design uses a pool of threads that can adapt to the number of simulated fields and sensors.
- **Flexibility**: Dynamic configurations allow complex scenarios to be simulated in real time.
- **Efficiency**: The MQTT protocol ensures light and fast data transmission.
- **Real-time monitoring**: By posting to *sensors/simulation/config/current,* other systems can monitor the status of the simulation constantly.

2. MQTT Broker (Mosquitto)

The MQTT broker, implemented with Mosquitto is the central point for communication between simulated sensors and other system components. It manages:

- **Simulation** configuration: Configuration parameters are sent to the simulated sensors to generate online values.
- **Sensor subscriptions**: Each sensor publishes data on specific topics (e.g., agriculture/field1/temperature).
- **Data distribution**: Published data are sent to subscribers for further processing.

3. Telegram Bot

The Telegram Bot is the main user interface component. It provides simple and direct access for simulation management. Its features include:

- **Simulation configuration**: Allows users to set main parameters (number of fields, sensors for each field, weather conditions, monitoring interval, thresholds, e-mail) via commands.
- **Simulation management**: Allows starting and stopping sensor simulation in addition to displaying current settings
- **Receipt of notifications**: Ability to enable or disable the sending of alerts when set thresholds are exceeded and the changing of these thresholds.
- **Runtime control**: Allows users to change simulation parameters in real time.
- **Report generation**: allows you to request screenshots of dashboards to get a practical and intuitive view of the report in graphical format.

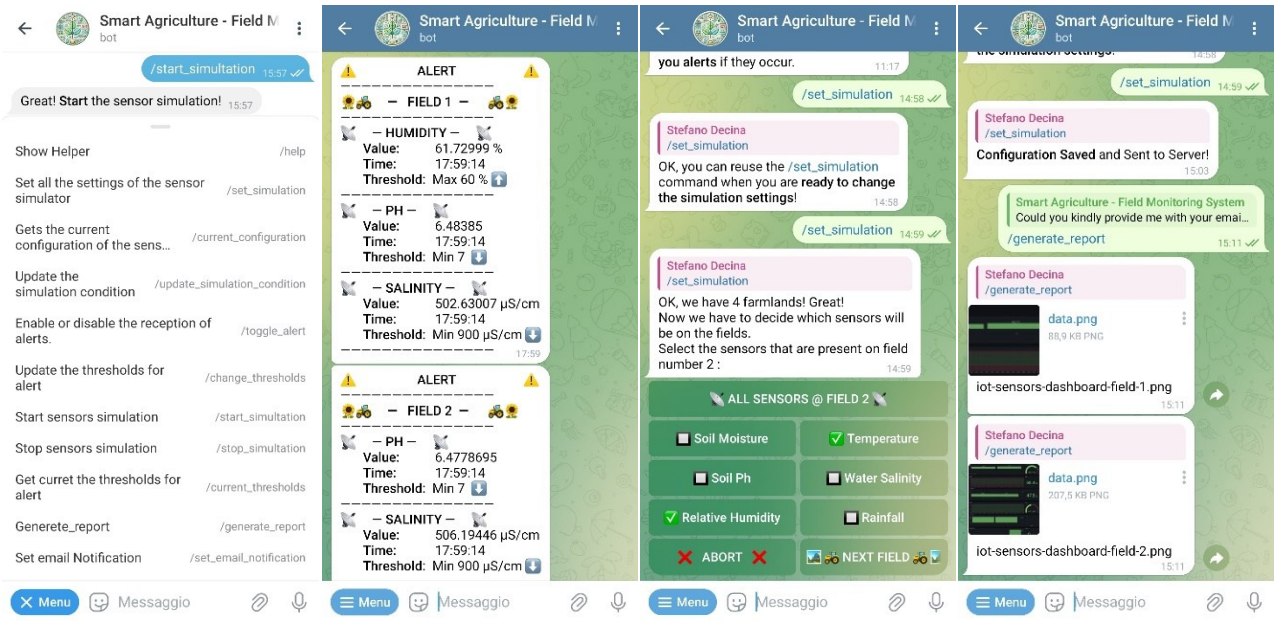


Figure 1List of available commands / Example of Alerts / Configuration Setting / Report

4. Node-RED

Node-RED is the main component for data flow orchestration and application logic processing and is containerized. The main streams defined are:

- **Telegram**: This stream handles all commands that can be sent to the system by the user via the Telegram Bot.
    - ***/help*** Briefly introduces the Telegram Bot and provides brief documentation to the main commands that can be used.
    - Set all the settings of the sensor
    - ***/set_simulation*** Starts the procedure of setting all simulation parameters, from choosing the number of fields to the type of sensors present, as well as the measurement interval and atmospheric conditions. These parameters are sent to the sensor simulation system by Node-Red on the relevant MQTT topic.
    - ***/current_configuration*** Returns a JSON containing all the parameters of the current configuration.
    - ***/update_simulation_condition*** Allows you to change values related to atmospheric conditions such as weather, starting temperature, humidity, etc. These values then are manipulated by the sensor simulation system to simulate real conditions. These values are sent to the sensor simulation system by Node-Red on the relevant MQTT topic.
    - ***/toggle_alert*** Activating and deactivating sensor alert threshold exceedance notifications via Telegram chat. Email notifications still remain active to maintain a communication of any problems at all times.
    - ***/change_thresholds*** Initiates the process of changing the minimum and/or maximum alert thresholds of the sensors in the fields. These will then go to change alert thresholds of notifications, Thresholds of Grafana dashboards and Alert Rules including the resulting emails.
    - ***/start_simultation*** Starts the simulation. Via the related Node-Red stream, the start message is sent on the MQTT topic that starts the sensor simulation system.
    - /stop_simultation Stops the simulation.
    - ***/current_thresholds*** Returns the JSON containing all thresholds currently set in the system.
    - ***/generate_report*** Via Node-Red sends a request to the Grafana service to render images of the dashboards and returns PNGs containing a screenshot of the dashboards of the agricultural fields set up.
    - ***/set_email_notification*** allows you to enter the e-mails used in Grafana's alert notification service via its Contact Point and Alert Rules. Nodered sends a request to the Grafana API which is responsible for modifying the Grafana Contact Point containing the e-mails to be contacted in case of an alert.

The image shows a part of the flow related to the Telegram Bot containing some of the nodes involved in command management. For the full view, please refer to the Node-Red interface.

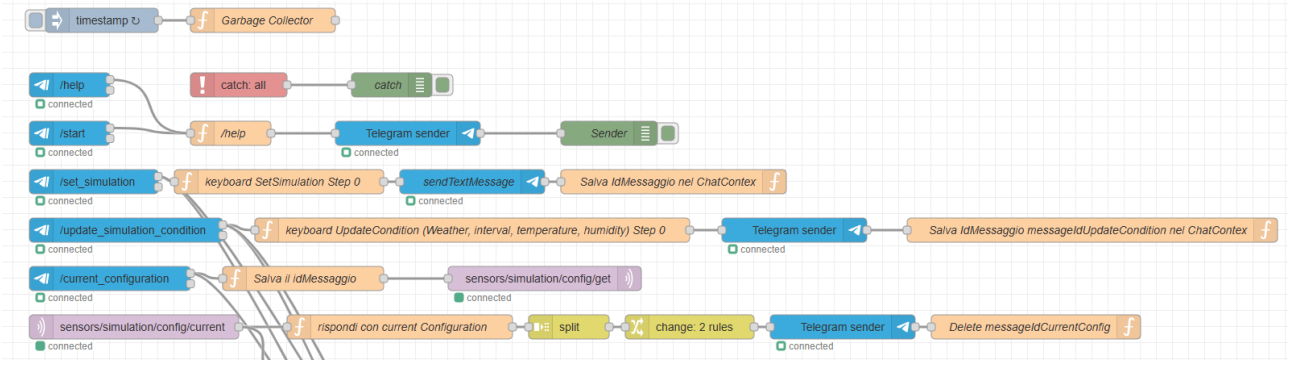
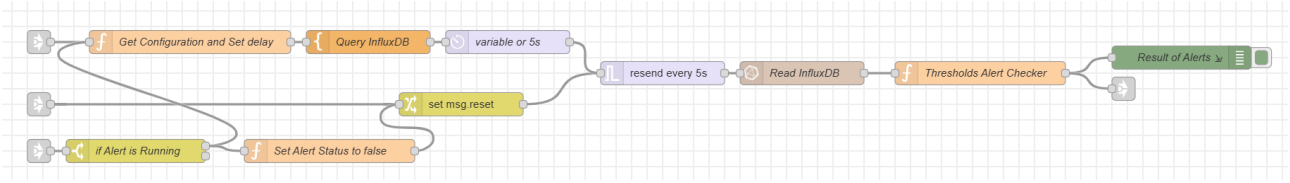
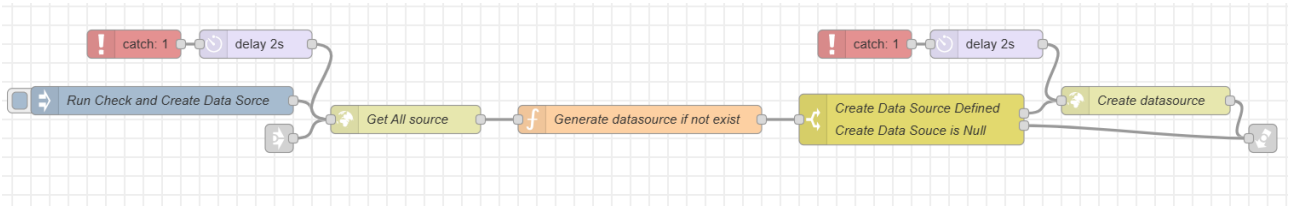


Figure 2Portion of Node-Red streams related to Telegram Bot commands.

- **Alert**: Flow that takes care of creating threshold alert notifications on Telegram. If the alert option is enabled, it runs Queries to InfluxDB every 5 seconds on all fields and their sensors. Then it performs a threshold check and in case the thresholds are exceeded, it sends via the Telegram flow a message containing the alerts to the connected chats.

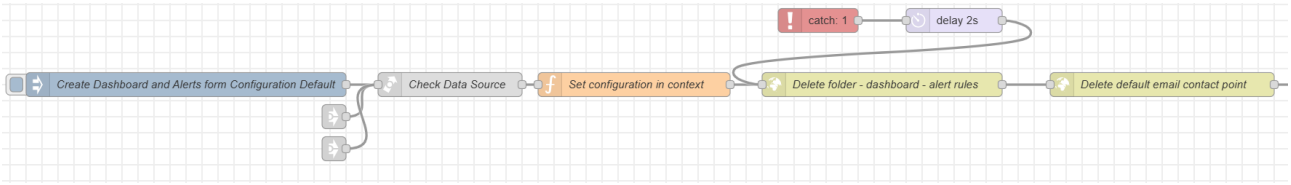


- **Grafana Dashboards, Contact points and Alert Rules**: This flow handles all requests that are executed to the Grafana API[1] . They are used for the dynamic creation of Dashboards and Alert Rules and for PNG reports via Grafana's render service. Each Request is accompanied by two nodes "*catch*" and "*delay*" which catches any errors of the "*HTTP Request*" node to which it is connected and after 2 seconds re-executes it.
    - The first flow checks for the presence of the InfluxDB DataSource on graphana Graphana, if it is not present it creates it. This procedure is performed via "*Link Call*" node because it is necessary for more than one flow.

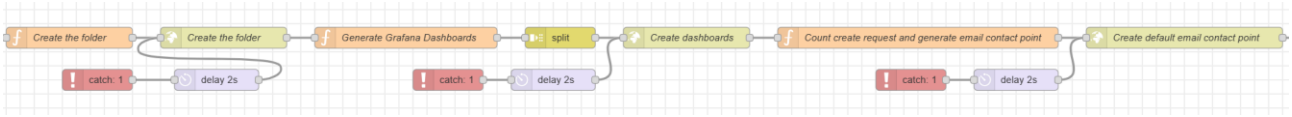


    - This flow represented of the next three images concerns the dynamic creation and editing of Grafana dashboards.
      As a first step it inserts/edits in the global context of Node-Red the configurations set via Telegram and related flow. Since it is also used for parameter editing, it makes sure to delete the dashboards, rules and contact point already on Grafana.

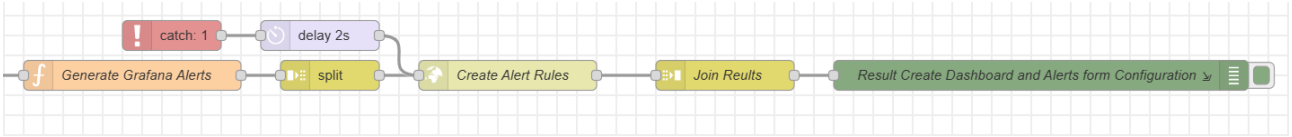
[1] https://editor.swagger.io/?url=https://raw.githubusercontent.com/grafana/grafana/main/public/openapi3.json



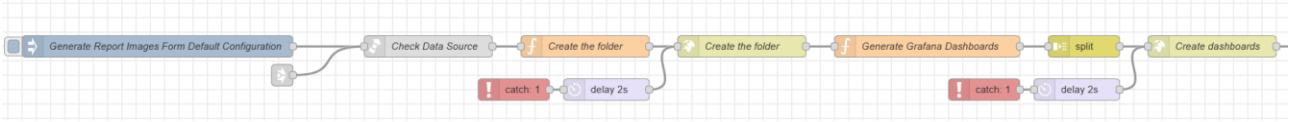
Create the folder that will contain the dashboards, then create the dashboards based on the configurations in the global context, and finally create the contact point with the specified e-mails via the Telegram Bot.



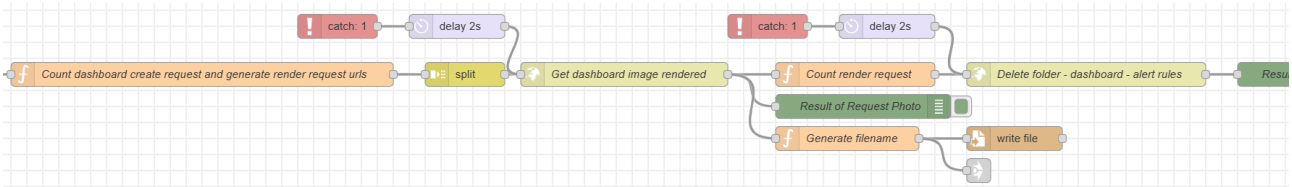
Finally, Create Alert rules according to the configuration parameters and thresholds.



- **Image Report**: Flow that deals with the generation of report images.
  As a first procedure it performs the Data Source check via "*Link call*" to the related flow. Then it creates ad hoc Dashboards containing only the sensor panels without the initial description and alerts, so the resulting images are more user friendly.



It runs requests for all newly created dashboards to Grafana's render service "*grafana-image-renderer*" which returns the images in *binary buffer* and then sends them to the Telegram Bot. A copy of the report images is also saved in the path "*/data/out/${msg.filename}.png*"



5. Telegraf

Telegraf is configured parallel agents to ensure redundancy and resilience in data collection. The main tasks include:

- **Data collection**: Subscribes MQTT topics to acquire data from sensors and send them to InfluxDB.
- **Flow Isolation**: Ensures reliable and secure data flow, minimizing the risk of loss or manipulation.

- ***Field_id* and *sensor_type*** extraction: using *processors.regex*, the agricultural field identifier and the sensor type referenced by the value in the message are extracted from the topic. These two pieces of information are used as keys for the values entered into InfluxDB.
- **Redundancy**: The compose docker is configured to have multiple instances of Telegraf, which having the same *mqtt_client_id* subscribe exclusively, that is, each message is read, processed and sent to InfluxDB by only one instance. Should one of the instances have problems, the message will be read by another available one.

```
# Configuration for telegraf agent
[agent]
  interval = "1s"
  round_interval = true

  metric_batch_size = 1000
  metric_buffer_limit = 10000

  collection_jitter = "0s"
  flush_interval = "1s"
  flush_jitter = "0s"

  precision = ""
  debug = true
  quiet = false
  logfile = ""
```

```
[[inputs.mqtt_consumer]]
  servers = ["tcp://${MQTT_BROKER}:1883"]
  topics = ["agriculture/+/+"]
  qos = 0
  client_id = "telegraf_mqtt_consumer_${CLIENT_ID}"
  connection_timeout = "30s"
  data_format = "value"
  data_type = "float"
  name_override = "smart_agriculture_measurements"t
  tagexclude = ["host"]

[[outputs.influxdb_v2]]
  urls = ["http://${INFLUX_HOST}:8086"]
  token = "${DOCKER_INFLUXDB_INIT_ADMIN_TOKEN}"
  organization = "${DOCKER_INFLUXDB_INIT_ORG}"
  bucket = "${DOCKER_INFLUXDB_INIT_BUCKET}"
```

Figure Configuring the Telegraf agent and its inpunt and output sources.

```
[[processors.regex]]
  namepass = ["smart_agriculture_measurements"]
  # 1) Extracts 'field_id' (e.g., "field1") from the topic tag
  [[processors.regex.tags]]
    key = "topic"
    pattern = "^agriculture/([^/]+)/([^/]+)$"
    replacement = "${1}"
    result_key = "field_id"
  # 2) Extracts 'sensor_type' (e.g., "temperature") from the topic tag
  [[processors.regex.tags]]
    key = "topic"
    pattern = "^agriculture/([^/]+)/([^/]+)$"
    replacement = "${2}"
    result_key = "sensor_type"
```

Figure Extraction of agricultural field id and sensor type from MQTT topic.

6. InfluxDB

InfluxDB, a *time-series* database, is the component responsible for storing historical sensor data. It is optimized to handle time data efficiently, allowing:

- **High-performance queries**: Grafana dashboards use queries on InfluxDB to provide real-time and historical visualizations.
- **Storage of large volumes of data**: Sensor data are archived with accurate timestamps for long-term analysis

The next images show how the docker-compose, Docker file and environment variables used by the container were configured.

```yaml
influxdb:
  build:
    context: InfluxDB-SA-FMS
    dockerfile: Dockerfile
  image: influxdb_sa_fms:latest
  container_name: influxdb-SA-FMS
  env_file:
    - se4iot-SA-FMS.env
  ports:
    - "8086:8086"
  volumes:
    - influxdb_sa_fms_data:/var/lib/influxdb2
  networks:
    - se4iot-SA-FMS-network
```

```
#influx.env
#Influxdb configurations
DOCKER_INFLUXDB_INIT_MODE=setup
DOCKER_INFLUXDB_INIT_USERNAME=admin
DOCKER_INFLUXDB_INIT_PASSWORD=admin123456!
DOCKER_INFLUXDB_INIT_ORG=se4iot
DOCKER_INFLUXDB_INIT_BUCKET=SA-FMS
DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=G-UFVBDaSWH
DOCKER_INFLUXDB_INIT_RETENTION=12h
```

```
FROM influxdb:2.7.4

# Create the configuration folder (if it does not exist)
# and copies the configuration file inside the container.
RUN mkdir -p /etc/influxdb2/configs
COPY config/influx-configs /etc/influxdb2/configs/influx-configs

# Expose port 8086 (not mandatory if done in docker-compose)
EXPOSE 8086

# Default command (optional,
# InfluxDB 2.x already starts with 'influxd' by default)
CMD ["influxd"]
```

Figure docker-compose, env file and and Dokerfile for InfluxDB configuration

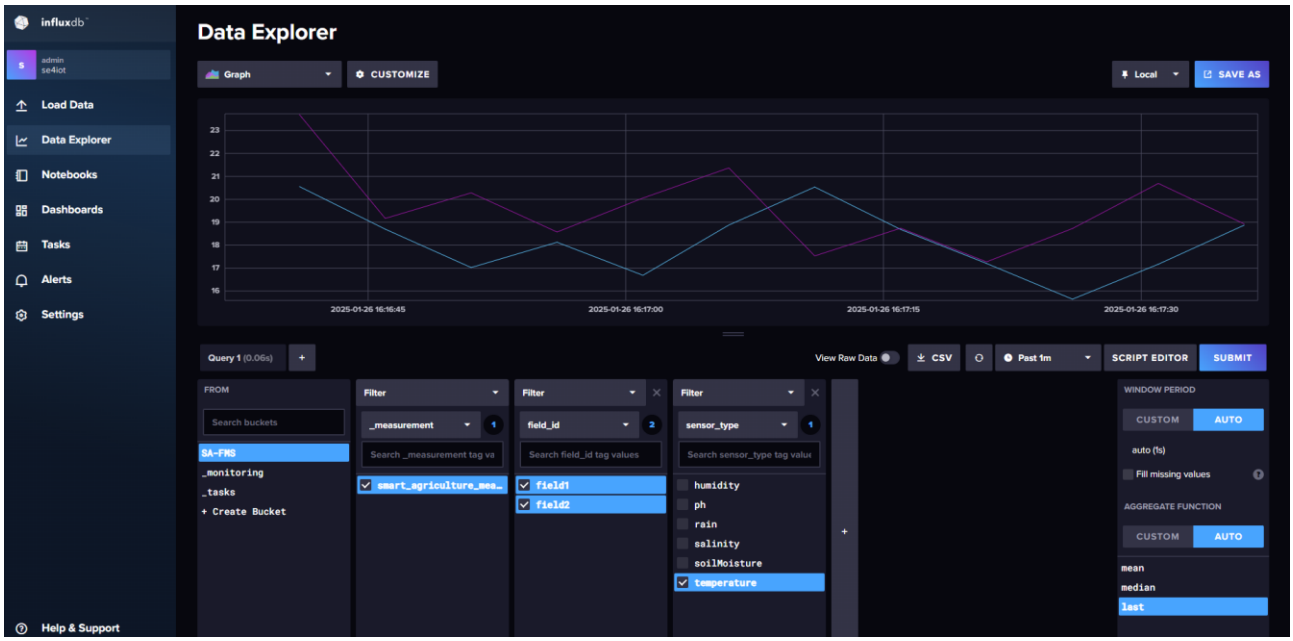


Figure InfluxDB Dashboard. Shows which keys the data are saved with in the database.

7. Graphana

Grafana is used for data visualization through dynamically defined dashboards based on terrain and sensor configuration. Its main features include:

- **Automatic dashboard generation**: Based on the simulation parameters and exploiting if API provided, a dashboard is generated in Grafana for each agricultural field, showing only the panels containing the associated sensor data. The dashboards are placed in the "*Smart Agriculture - Field Monitoring System*" folder created specifically to contain them.
- **Dynamic thresholds**: Leveraging the API provided to manipulate panel thresholds, visualizations adapt to the threshold values set, dynamically changing colors and visual indicators.
- **Threshold exceedance alerts**: There is a section at the top of the dashboards intended to accommodate threshold exceedance alerts. These alerts are always generated by running requests to the API that Grafana exposes.
- **Email alerts**: In the configuration flow of the Grafana environment, the *contact point* containing user-specified e-mails via Telegram Bot is also created, which is used to send messages whenever a threshold is exceeded.
- **Report export**: Using Grafana Image Renderer, dashboards can be converted to images and sent to the Telegram Bot chat that requests them.

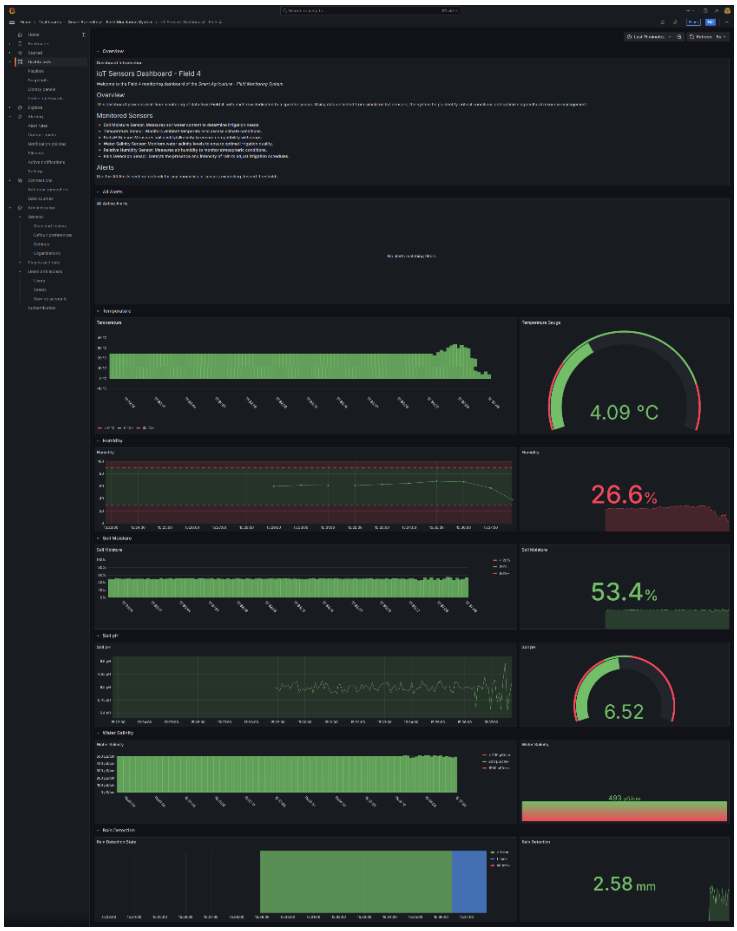


Figure Dashboard agricultural field with all sensors installed

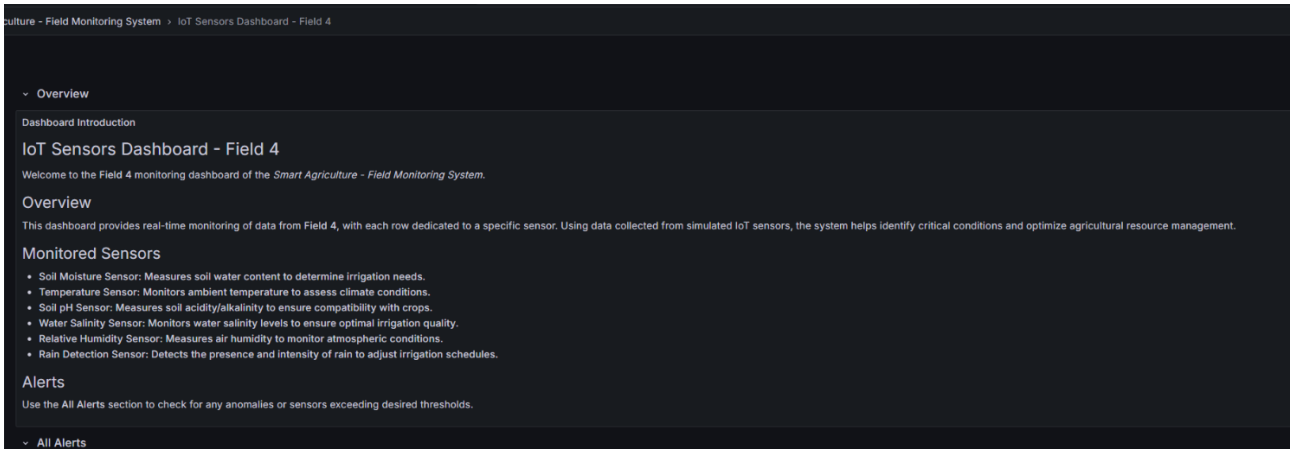


Figure Descriptive panel collecting all the information from the sensors in the field

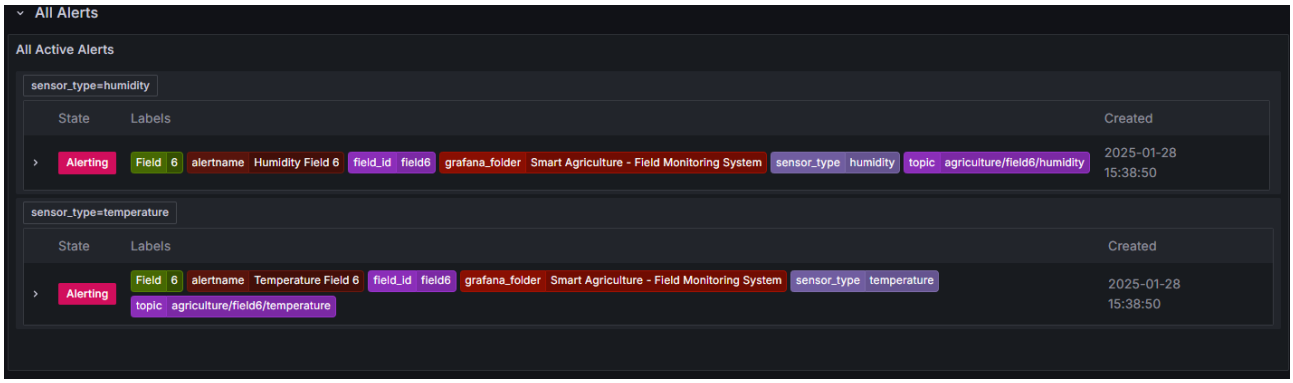


Figure Section containing any alarm warnings regarding the field sensors

Figure Sensor panels - Which contains only the panels corresponding to the sensors in the field

8. Grafana Image Renderer

The **Grafana Image Renderer** component is integrated into the architecture to enable snapshot generation of dashboards configured in Grafana. This component is used for:

- **Automatic generation of visual reports**: Converts dashboards into static images for later sending via Telegram bot.
- **Notification support**: Provides a visual alternative to textual data sent via alerts, improving immediate understanding of critical sensor states.
- **Flow automation**Image creation is fully automated and managed in a Node-Red flow in which temporary ad hoc dashboards are created so that they are visually usable as images on Telegram chat.

The component is containerized and operates in close integration with Grafana to provide a clear and shareable visual representation of the data.
In the Grafana configuration, the dependency to the render container must be inserted and the two environment variables shown in the following images must be entered.

```yaml
grafana-renderer:
  image: grafana/grafana-image-renderer:latest
  container_name: grafana-renderer-SA-FMS
  ports:
    - "8081:8081"
  volumes:
    - grafana_renderer_sa_fms_data:/var/lib/renderer
  networks:
    - se4iot-SA-FMS-network
```

Figure Docker-compose for Grafana Image Renderer

```yaml
depends_on:
  - influxdb
  - maildev
  - grafana-renderer
```

```
GF_RENDERING_SERVER_URL=http://grafana-renderer-SA-FMS:8081/render
GF_RENDERING_CALLBACK_URL=http://grafana-SA-FMS:3000/
```

Figure Section of the docker-compose and environment variables of Grafana for Image Render.

9. E-mail Dev (SMTP Server)

The system includes an **SMTP server** for sending backup notifications via e-mail. This ensures that users always receive critical alerts even in case of problems with Telegram.

The section of the docker-compose concerning E-mail Dev and its GUI is shown below. An example of an e-mail containing a threshold exceeded warning is also shown.

```yaml
maildev:
  image: maildev/maildev:latest
  container_name: maildev-SA-FMS
  env_file:
    - se4iot-SA-FMS.env
  volumes:
    - maildev_sa_fms_data:/data
  ports:
    - "1080:1080"
    - "1025:1025"
  networks:
    - se4iot-SA-FMS-network
```

Figure Docker-compose for Mail Dev

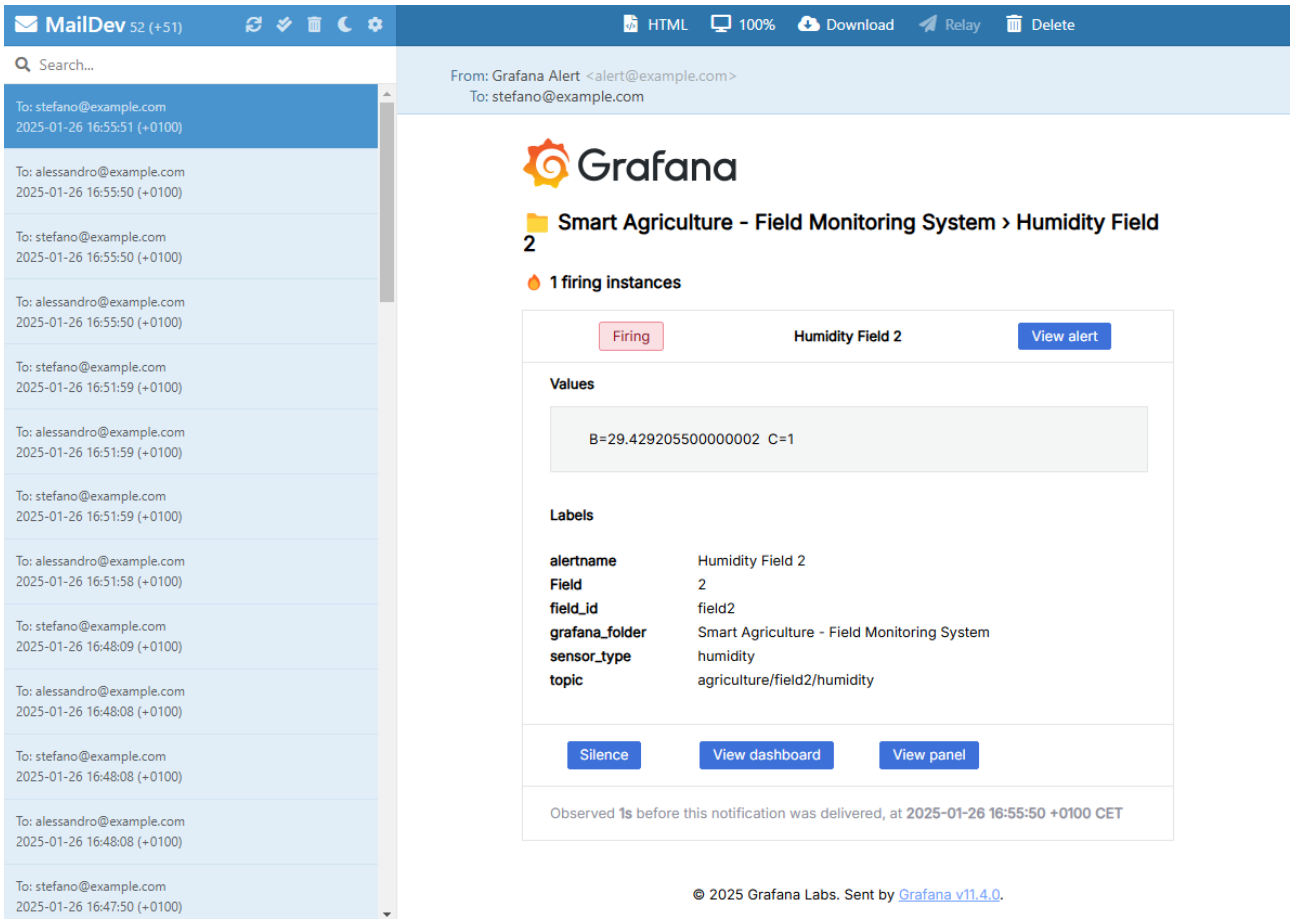


Figure Mail Dev interface and threshold exceedance warning email

10. Deploying the solution

The solution uses **Docker** containers to simulate and monitor an intelligent agricultural environment. A *docker-compose.yml* file is provided for *deployment* via **Docker** and a set of configuration files for **Kubernetes**, located in the *k8s* folder.

Deploy with Docker

General Description

The *docker-compose.yml* file defines the services required for the **Smart Agriculture Field Monitoring** System **(SA-FMS)**. This configuration uses Docker Compose to orchestrate several containers, each with a specific role in the system.

Services

1. **hemostat-SA-FMS**:
    - **Role**: MQTT broker for communication between system components.
    - **Door exposed**: 1883.
    - **Volumes**:
        - */mosquitto/data*: Data persistence.
        - */mosquitto/log*: Persistence of logs.
    - **Network***: se4iot-SA-FMS-network*.

2. **sensor-simulator-SA-FMS**:
    - **Role**: It simulates IoT sensors, generating and sending realistic data to the MQTT broker.
    - **Dependencies**: *hemostat-SA-FMS*.
    - **Volumes**: Persistence of simulation data.
    - **Configuration**: Environment variables defined in *se4iot-SA-FMS.env*.
3. **node-red-SA-FMS**:
    - **Role**: Provides a graphical platform to orchestrate IoT flows and process data in real time.
    - **Port exposed**: 1880 (Node-RED web interface).
    - **Environment variables**:
        - Includes Telegram token (BOT_TOKEN) for notifications.
    - **Addictions**:
        - *hemostat-SA-FMS*
        - *influxdb-SA-FMS*
        - *graphana-SA-FMS*
        - *sensor-simulator-SA-FMS*.
4. **influxdb-SA-FMS**:
    - **Role**: Time-series database to store sensor data.
    - **Exposed port**: 8086.
    - **Volumes**:
        - */var/lib/influxdb2*: Persistence of data.
        - */etc/influxdb2*: Database configuration.
5. **telegraf-SA-FMS**:
    - **Role**: Agent for collecting and monitoring data from sensors and *MQTT* broker, storing them in *InfluxDB*.
    - **Addictions**:
        - *hemostat-SA-FMS*
        - *influxdb-SA-FMS*.
6. **maildev-SA-FMS**:
    - **Role**: Email server simulator for testing notifications.
    - **Exposed doors**:

        - 1080: Web interface.
        - 1025: SMTP.
7. **graphana-renderer-SA-FMS**:
    - **Role**: Service to generate images of Grafana dashboards.
    - **Exposed port**: 8081.
8. **graphan-SA-FMS**:
    - **Role**: Dashboard for visualizing data collected from sensors and metrics.
    - **Exposed port**: 3000.
    - **Addictions**:
        - *influxdb-SA-FMS*
        - *maildev-SA-FMS*
        - *graphana-renderer-SA-FMS*.

Volumes

Description and Benefits

Docker volumes are used to ensure **persistence of data** and configurations between container restarts. This approach allows services to maintain state even after containers are discontinued or recreated. The main benefits of volumes include:

1. **Data Persistence**:
    - Sensitive or critical data, such as database configurations, logs or sensor-generated data, are stored in volumes that persist regardless of container life cycles.
2. **Easy Backup and Restore**:
    - Volumes can be easily copied or saved for backup, making it easier to restore in case of errors or accidents.
3. **Insulation and Security**:
    - Data are separated from containers, reducing the risk of accidental loss during operations such as upgrades or changes to containers.
4. **Sharing between Containers**:
    - Multiple containers can access the same volumes to share data and configurations.
5. **Performance Improvement**:
    - Volumes are managed directly by the Docker engine, offering better performance than using bind mounts or local directories.

List of Volumes

- **mosquitto_sa_fms_data**:
    - Stores persistent data from the Mosquitto MQTT broker.
- **mosquitto_sa_fms_logs**:
    - Contains Mosquitto log files.
- **sensor_simulator_sa_fms_data**:
    - Stores data generated by the IoT sensor simulator.
- **nodered_sa_fms_data**:
    - Stores the flows and configurations of the Node-RED environment.
- **influxdb_sa_fms_data**:
    - Contains persistent data from the InfluxDB database.
- **influxdb_sa_fms_config**:
    - Save InfluxDB configuration files.
- **telegraf_sa_fms_data**:
    - Stores configuration data and metrics collected by Telegraf.
- **maildev_sa_fms_data**:
    - Save simulated emails and other configuration data for MailDev.
- **grafana_renderer_sa_fms_data**:
    - Contains image renderer configuration data for Grafana.
- **grafana_sa_fms_data**:
    - Stores data about Grafana dashboards, users, and configurations.

Network

- **se4iot-SA-FMS-network**: A bridge-type Docker network shared by all services to facilitate internal communication.

Installation Scripts

Purpose

The *install.cmd* (Windows) and *install.sh* (Linux/Mac) scripts automate the steps to configure and boot the system via Docker.

Features

1. **Unification of configurations**:
    - It combines the individual .env files of each component into a single file (se4iot-SA-FMS.env).
    - Verify the existence of the required .env files for each service.

- In case of missing .env files, it ends with an error message.

2. **Creation of the combined .env file**:
   - The .env files of the various components are concatenated in a specified order, creating a single file needed for deployment.
3. **Container construction and startup**:
   - Constructs containers using the command:

     docker compose build --no-cache
   - Start containers in detached mode with

     docker compose up -d
   - Includes option:

     --scale telegraf-SA-FMS=2

     To start two instances of the telegraf-SA-FMS service.
4. **Conflict check**:
   - If the combined file already exists, the script asks for confirmation before overwriting it.