Subject: Cloud Based IoT and Big Data Platforms

HOME PROJECT

Efficient LED Farming System

OE-NIK	Neptun code and Student’s name	**DQ4WX0 Fujiwara Takahiro** **BJSLS2 Li Yiran**
04.May.2025	Lecturer name:	Emődi Márk

Contents

1 Introduction

Modern Agriculture Challenges

Urban agriculture and indoor farming have become attractive new agricultural models that do not rely on traditional vast arable lands, driven by rapid urbanization and population growth. However, traditional agriculture depends heavily on the natural environment, especially sunlight, resulting in several issues:

- **Instability and Limitations of Natural Light:**
  Natural light varies considerably in intensity and spectrum with the seasons and weather. This variation makes it difficult to maintain a uniform, optimal lighting environment for crops. In urban areas, buildings and other obstructions can prevent sufficient light from reaching plants, leading to uneven lighting. Consequently, photosynthetic efficiency may decrease, and inconsistencies can occur in growth and harvest timing.
- **Inefficiencies of Fixed Lighting Systems**
  Fixed lighting systems supply only a constant spectrum, which makes it challenging to optimally adjust wavelengths according to a crop's growth stage. For example, red light is effective for promoting fruit formation and blue light is necessary for the growth of leaves and stems. Without the ability to flexibly adjust these wavelengths, opportunities to create an optimal growing environment, increase yields, and shorten growth cycles are lost.
- **Issues with Energy Efficiency and Environmental Impact**
  In urban agriculture and indoor farming, efficient cultivation in limited spaces is essential. However, conventional lighting devices often consume a high amount of power, leading to increased energy costs. Moreover, introducing technologies that reduce environmental impact is imperative for achieving sustainable agriculture.

Improving Agriculture Through Innovative Lighting Technologies

Against this backdrop, it is essential to adopt new lighting technologies, particularly LED lighting systems—that are unaffected by the weather, provide dynamic light control, and offer high energy efficiency. LED lighting allows for spectral adjustments appropriate to various growth stages, ensuring efficient energy use while reducing environmental impact. This technology holds great promise in contributing significantly to the advancement of urban agriculture and indoor farming.

2 Proposed Solution

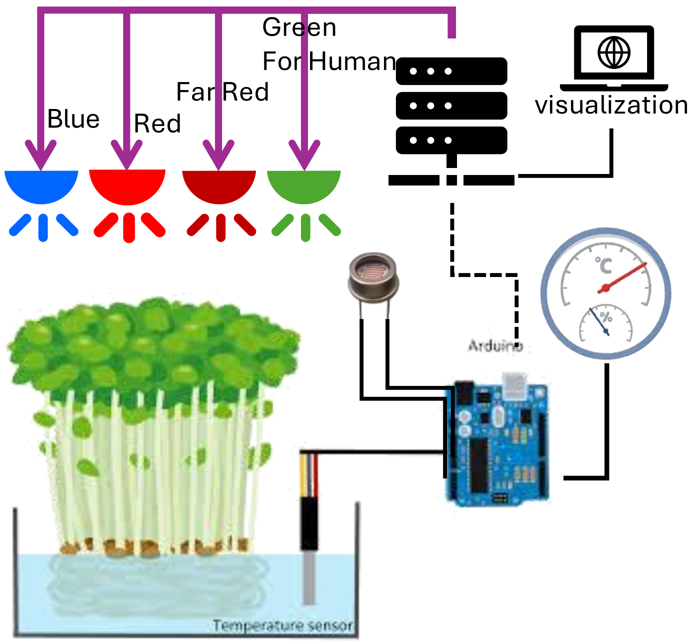
2.1 System Overview

Our solution replaces static lighting and reliance on natural sunlight with a fully controllable, dynamic light environment. Centered on LED technology, the system adjusts wavelengths based on growth stages, monitors conditions in real time via sensors, and utilizes cloud-based data analysis to enhance yield and energy efficiency.

Scope of This Project

Regarding our hardware, this time, we will not use any physical equipment; instead, we will rely on simulated data.

2.2 Hardware Configuration

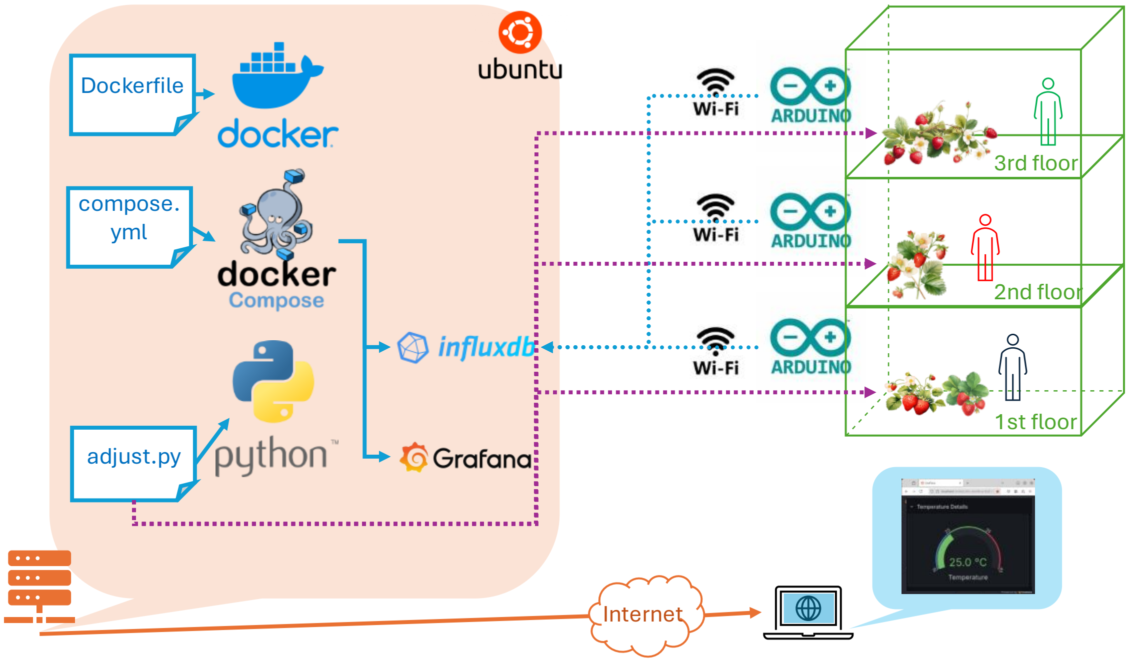


Far-Red light (Optional) (700–750nm) enhances strawberry growth and flowering. In combination with red light, it modulates phytochrome-mediated growth signals, potentially boosting yield. However, too much Far-Red can induce unwanted elongation, so a proper balance is essential.

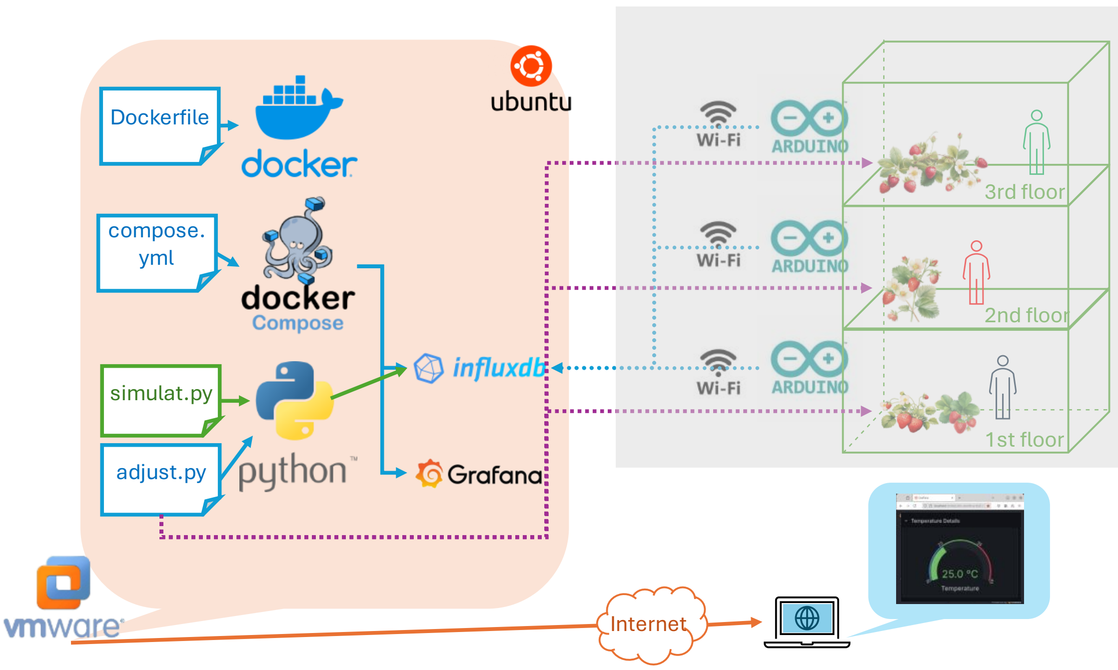
- **Adjustable LED Lighting:** Supplies optimal ratios of red (for fruit formation) and blue light (for leaf/stem growth) with dynamic wavelength and brightness adjustments tailored to each growth phase.
    - **Option:** Far Red
- **Environmental Sensors:** Deploy light, temperature, and humidity sensors to continuously monitor the crop environment, crucial for stabilizing conditions in urban and indoor settings.
- **Microcontroller:** Uses platforms like Raspberry Pi or Arduino to integrate sensor inputs and control LED outputs for centralized system management.

2.3 Software and Network Setup

Here it shows the real environment, below:



However, this time, we use simulation environment, below:



Difference between Real and Simulation Environment

Item	Real Environment	Simulation Environment
Server Hardware	Real Hardware	VMware
Sensor	Real Sensor	Simulation data by Python
LED	Real LED	Simulated LED

Common Environments:

- **Programming Environment:** Implements LED control and sensor data collection in Python, using automation algorithms to adjust lighting in real time.
- **Data Collection & Cloud Integration:** Periodically uploads sensor data to the cloud for analytical processing on an IoT platform, enabling trend analysis and optimization of growth patterns.
- **Data Visualization:** Utilizes Grafana dashboards for real-time display of plant growth and environmental variables, offering intuitive insights for operators.
- **Efficient Deployment:** Employs Docker containers to streamline development, testing, and deployment across the entire system.

2.4 Automation and Performance Validation

The system will be validated in a simulation environment where dynamic adjustments of LED lighting are linked to changes in light intensity, spectrum, and other environmental parameters. Automated on/off controls and output adjustments aim to demonstrate improvements in growth rate and harvest times.

2.5 Expected Benefits and Contributions

Compared with conventional fixed lighting, the proposed LED system offers: Together, these benefits are expected to transform urban and indoor farming by enabling a more efficient, environmentally friendly, and productive agricultural model.

- **Optimized Light Environments:** Improving growth and yield through precise spectral adjustments.
- **Enhanced Energy Efficiency:** Reducing operational costs by lowering power consumption.
- **Reduced Environmental Impact:** Supporting sustainable practices in agriculture.

2.6 List of Software Version

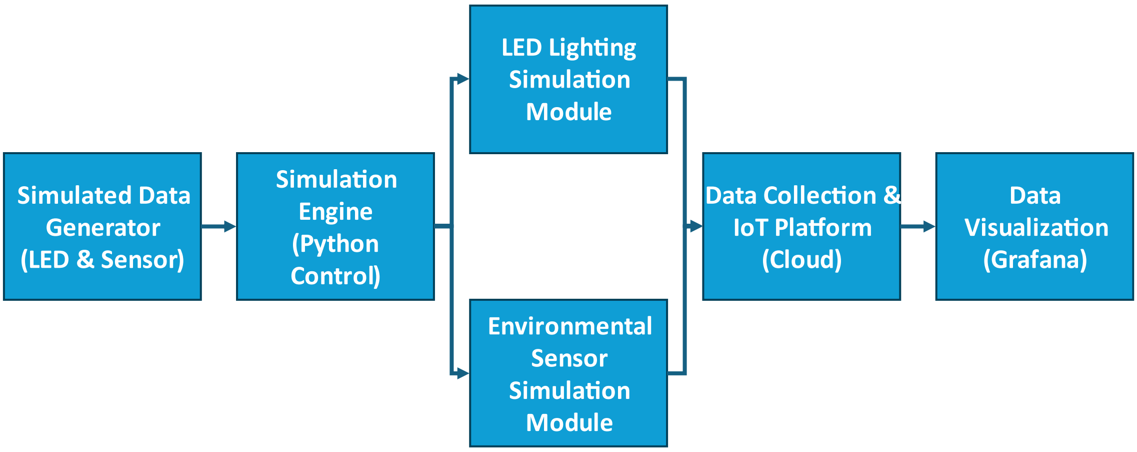
Purpose	Software Name	Version
Host OS	Windows 11 Pro	24H2
Virtual Machine	VMware(R) Workstation 17 Pro	17.6.3
Target OS	Ubuntu	2404LTS
Container Engine	Docker version	28.1.1
Container Management	Docker Compose	v2.35.1
Database	InfluxDB	2.0.9
Graphical Dashboard	Grafana	11.6.1
Programming	Python	3.11.12

3 Design Plan

3.1 Outlines of program

Here is an explanation of program using Components Diagram.

- **Simulated Data Generator:** Without using any physical hardware, it simulates LED lighting and sensor data.
- **Simulation Engine (Python Control):** Based on simulated data, it controls the operation of LED lighting and environmental sensors.
- **LED Lighting / Sensor Simulation Modules:** Each module generates simulation data and provides feedback to the control engine.
- **Data Collection & IoT Platform (Cloud):** It uploads the data obtained from the simulation to the cloud for analysis and optimization.
- **Data Visualization (Grafana):** It visualizes the analysis results and system status in real time using Grafana, providing operators with intuitive information.



3.2 Configuring Steps

The following setup is required to establish this simulation project. Detailed explanations can be found in the appendix of this document.

- Install VMware (Omitted from this document as it is out of scope.
- Setup Ubuntu 2404LTS (Omitted from this document as it is out of scope.)
- Install Git
- Install Docker/Docker Composer
- Setup Python in Docker
- Install InfluxDB
- Install Grafana

4 Implementation

4.1 Grafana Design

Grafana dashboards concisely display sensor data and LED activity, enabling rapid checks of temperature, humidity, and LED settings for optimal performance.

Quick View

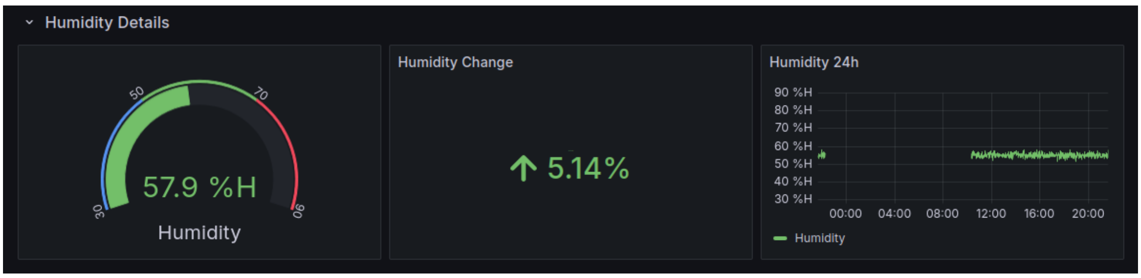
There is Quick View. You can understand the current situation just by looking at this Quick View.

Temperature Details

The temperature details show the current value, the rate of increase or decrease, and the 24-hour trend.

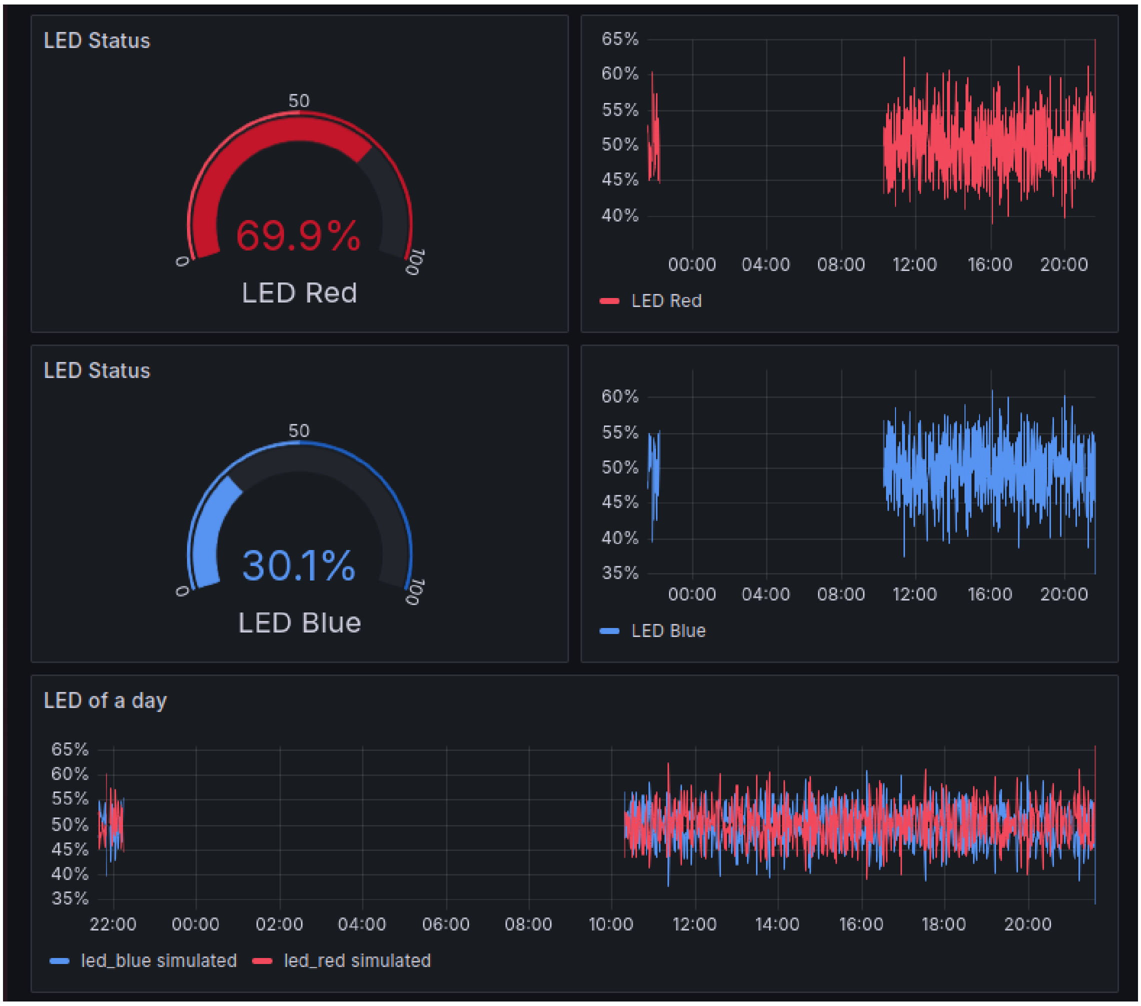
Humidity Details

The Humidity details show the current value, the rate of increase or decrease, and the 24-hour trend.



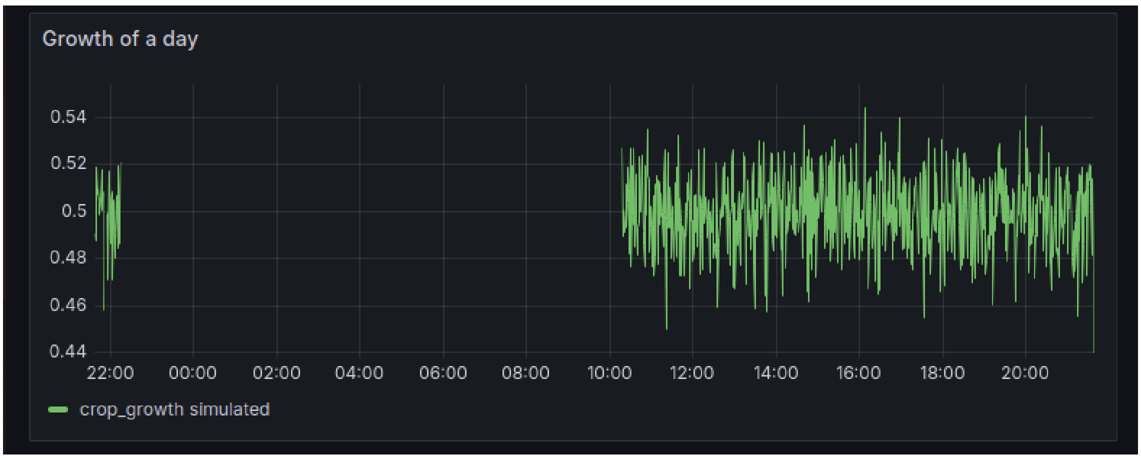
LED Details

The LED details are presented through graphs that separately display the current state and 24-hour transitions for the red and blue channels, along with a comparison between them.



Growth of the day

A graph displays the 24-hour growth, measured in grams.



4.2 Deployment

Deployment Strategy

Our system adopts GitHub as the primary repository and version control platform, ensuring that all relevant stakeholders always have access to the most up-to-date version.

Version Control and Code Sharing

All code modifications are managed through Git, with every commit and release tag carefully recorded. This practice enables us to quickly roll back to a stable previous version if issues arise, and it facilitates testing on earlier versions when necessary.

Automated CI/CD and Deployment Pipeline

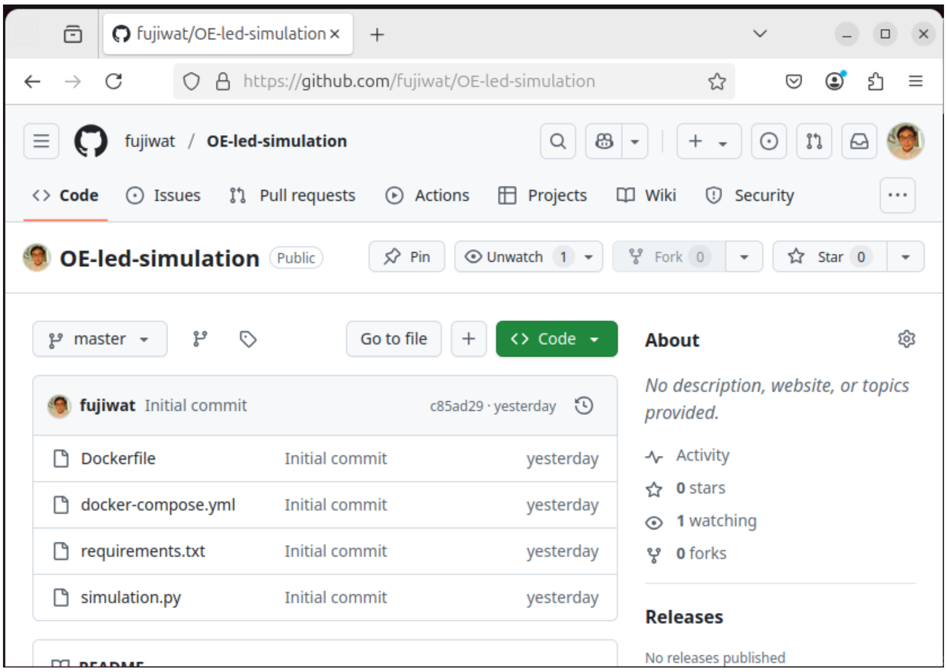
Our automated CI/CD (Continuous Integration/ Continuous Delivery) pipeline performs tests and builds with every GitHub push, ensuring that verified production deployments occur safely and efficiently while effectively reducing human error risks.

Branch Strategy and Version Tracking

A clear branch strategy is employed to segregate code for development, testing, and production. This organization not only simplifies version tracking for each release but also supports overall system quality assurance.

Access to the GitHub Page

https://github.com/fujiwat/OE-led-simulation



5 System Verification

In this chapter, we summarized the verification methods, test cases, and results that were conducted to confirm that the LED simulation system met its original design and specifications.

5.1 Scope

- **Functional Verification:** Confirm that the current state and 24-hour transitions of the LED channels (red and blue) were displayed correctly.
- **Integration Verification:** Verify that the interactions among the components (Python, InfluxDB, Grafana, etc.) were executed seamlessly.
- **Performance Verification:** Evaluate whether the system's responsiveness and real-time update speed met the required criteria when handling large volumes of data.

5.2 Test Cases and Scenarios

ID	Test Item	OK?	Remarks
Functional Tests			
11	Verify that the LED channels display separate graphs for red and blue	YES	
12	Verify showing both current states and 24-hour transitions	YES	Not able to change to 5 minutes transitions
13	Confirm that temperature and humidity displays correctly indicate current values	YES	
14	Confirm that temperature and humidity displays correctly indicate trends (upward or downward, and rates of change.	YES	
Integration Tests			
21	Validate that simulation data generated by the Python components is accurately recorded in InfluxDB.	YES	
22	Validate that the InfluxDB data is correctly visualized by Grafana dash boards.	YES	
23	Check the timestamp consistency, update frequency, and data accuracy across all components.	YES	
Performance Tests			
31	Conduct load tests to evaluate the system’s behavior under high volumes of incoming data.	YES	At first error -> Query is changed
32	Measure the responsiveness and update latency of the Grafana dashboards when handling extensive datasets.	YES	

6 Conclusion and Results

The system verification confirmed that the LED channels' current states and 24-hour transitions, as well as the integration among Python, InfluxDB, and Grafana, and the system's performance under high data loads, generally met the specified requirements.

Generated Web Page (Public URL)

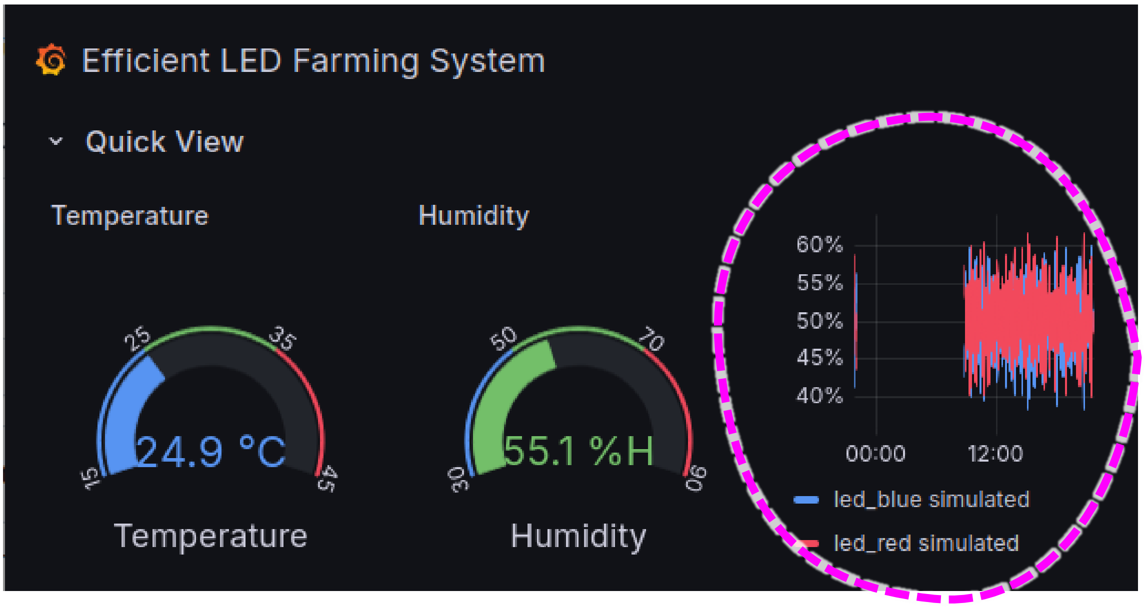
http://localhost:3000/public-dashboards/87bb76680bec4393a07ad21efa86a247

Access to the GitHub Page

https://github.com/fujiwat/OE-led-simulation

6.1 Key Findings and Improvements

It would be beneficial if the line graph display were adjustable—not fixed to a 24-hour view—but capable of showing the most recent 5 minutes or 1 hour. We explored configuration options in Grafana, but were unable to implement these adjustments within a short time frame, so we plan to incorporate this as a future improvement.



6.2 Overall Evaluation

Despite being a simulation, this project enabled us to experience a complete series of operations involved in implementing a Cloud Based IoT System.

Appendix

1 Installation and Setup

1.1 Install Git

```
$ sudo apt update
$ sudo apt install git -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
7 packages can be upgraded. Run 'apt list --upgradable' to see them.
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored i
:
Reading state information... Done
git is already the newest version (1:2.43.0-1ubuntu7.2).
git set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 7 not upgraded.
$ git -version
git version 2.43.0
git config --global user.name "Takahiro Fujiwara"
git config --global user.email "fujiwat0601@gmail.com"
```

1.2 Install Docker

```
$ sudo apt update
$ sudo apt install curl -y
$ curl -fsSL https://get.docker.com | sudo sh
$ docker -version
Docker version 28.1.1, build 4eba377
```

1.3 Setup Python in Docker

Preparing for Work Directory

```
$ mkdir led-simulation
$ cd led-simulation
```

Prepare Dockerfile

```
$ gedit Dockerfile
FROM python:3.11
WORKDIR /app
COPY requirements.txt ./
RUN pip install --upgrade pip && pip install -r requirements.txt
COPY . .
CMD ["python", "simulation.py"]
```

Prepare requremtns.txt

```
$ gedit requirements.txt
numpy
matplotlib
pandas
```

Appendix

Prepare simulation.py

```
$ gedit simulation.py
from influxdb import InfluxDBClient
import random, time

client = InfluxDBClient(host='influxdb', port=8086)
DATABASE = "simulation_db"
if not any(db['name'] == DATABASE for db in client.get_list_database()):
    client.create_database(DATABASE)
client.switch_database(DATABASE)

def generate_sim_data():
    temperature = random.uniform(20.0, 30.0)
    humidity = random.uniform(40.0, 70.0)
    led_red = random.randint(0, 100)
    led_blue = 100 - led_red
    crop_growth = (led_red * 0.3 + led_blue * 0.7) / 100
    return {
        "temperature": round(temperature, 2),
        "humidity": round(humidity, 2),
        "led_red": led_red,
        "led_blue": led_blue,
        "crop_growth": round(crop_growth, 2)
    }

def send_data_to_influx(data):
    json_body = [
        {
            "measurement": "simulation_data",
            "tags": { "sensor": "simulated" },
            "fields": data
        }
    ]
    client.write_points(json_body)

if __name__ == "__main__":
    while True:
        data = generate_sim_data()
        print("Simulated Data:", data)
        send_data_to_influx(data)
        time.sleep(1)  # 1 秒ごとにデータ送信
```

Prepare docker-compose.yml

This is a file to Integrate Multiple Services: InfluxDB, Grafana.

```
$ gedit docker-compose.yml
# version: '3'

services:
  simulation:
    build: .
    container_name: led-simulation
    depends_on:
      - influxdb
    networks:
      - sim-network
    restart: unless-stopped
    environment:
      # Example: Set the hostname for sending data to InfluxDB (resolved by service name)
      - INFLUXDB_HOST=influxdb
```

```
      - INFLUXDB_PORT=8086
    # You can add port mapping or other environment variables as needed
    # ports:
    #   - "8000:8000"

  influxdb:
    image: influxdb:2.0
    container_name: influxdb
    ports:
      - "8086:8086"
    environment:
      - DOCKER_INFLUXDB_INIT_MODE=setup
      - DOCKER_INFLUXDB_INIT_USERNAME=admin
      - DOCKER_INFLUXDB_INIT_PASSWORD=admin123
      - DOCKER_INFLUXDB_INIT_ORG=my-org
      - DOCKER_INFLUXDB_INIT_BUCKET=simulation_db
      - DOCKER_INFLUXDB_INIT_RETENTION=0
      - DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=my-super-secret-token
    volumes:
      - influxdb-data:/var/lib/influxdb2
    networks:
      - sim-network
    restart: unless-stopped

  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    ports:
      - "3000:3000"
    depends_on:
      - influxdb
    networks:
      - sim-network
    restart: unless-stopped
    volumes:
      - grafana-data:/var/lib/grafana

networks:
  sim-network:

volumes:
  influxdb-data:
  grafana-data:
```

RUN Docker Compose

No need to run `docker build -t led-simulation` because it is also in the `.yml` file.

```
$ docker compose up --build
[+] Running 22/22
 ✔ grafana Pulled                                                   171.0s
   ✔ f18232174bc9 Pull complete                                      13.0s
   ✔ dd95f48fea40 Pull complete                                      13.6s
                               :
Compose can now delegate builds to bake for better performance.
 To do so, set COMPOSE_BAKE=true.
[+] Building 6.5s (11/11) FINISHED                          docker:default
 => [simulation internal] load build definition from Dockerfile       0.1s
 => => transferring dockerfile: 201B                                  0.0s
                               :
```

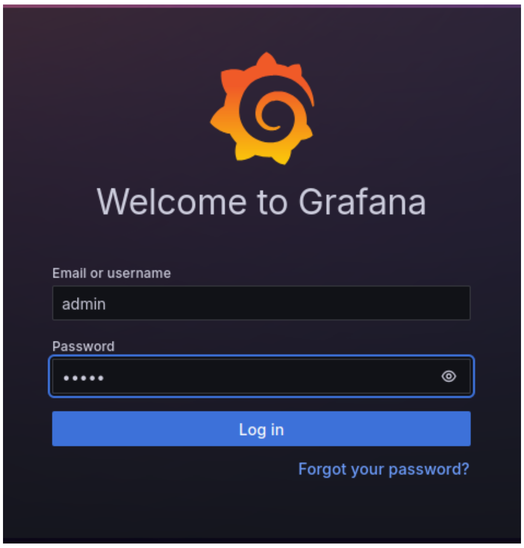
```
 => [simulation] resolving provenance for metadata file                   0.1s
[+] Running 7/7
 ✔ simulation                              Built                          0.0s
 ✔ Network led-simulation_sim-network      Created                        1.1s
 ✔ Volume "led-simulation_influxdb-data"   Created                        0.0s
                                     :
grafana           | logger=settings t=2025-05-02T10:02:15.882199675Z level=info msg=
"App mode production"
grafana           | logger=featuremgmt t=2025-05-02T10:02:15.88407754Z level=info ms
g=FeatureToggles groupToNestedTableTransformation=true recordedQueriesMulti=true a
nnotationPermissionUpdate=true dashboardScene=true userStorageAPI=true alertingRul
                                     :
led-simulation  |     from influxdb import InfluxDBClient
led-simulation  | ModuleNotFoundError: No module named 'influxdb'
led-simulation exited with code 1
led-simulation  | Traceback (most recent call last):
led-simulation  |   File "/app/simulation.py", line 1, in <module>
led-simulation  |     from influxdb import InfluxDBClient
led-simulation  | ModuleNotFoundError: No module named 'influxdb'
led-simulation exited with code 1
                                     :
```

The program led-simulation is repeating forever. Use another terminal window.

1.4 Data Visualization with Grafana

Access the Grafana Web UI by opening the following URL in your browser (using the host's IP or localhost).

```
http://localhost:3000
```

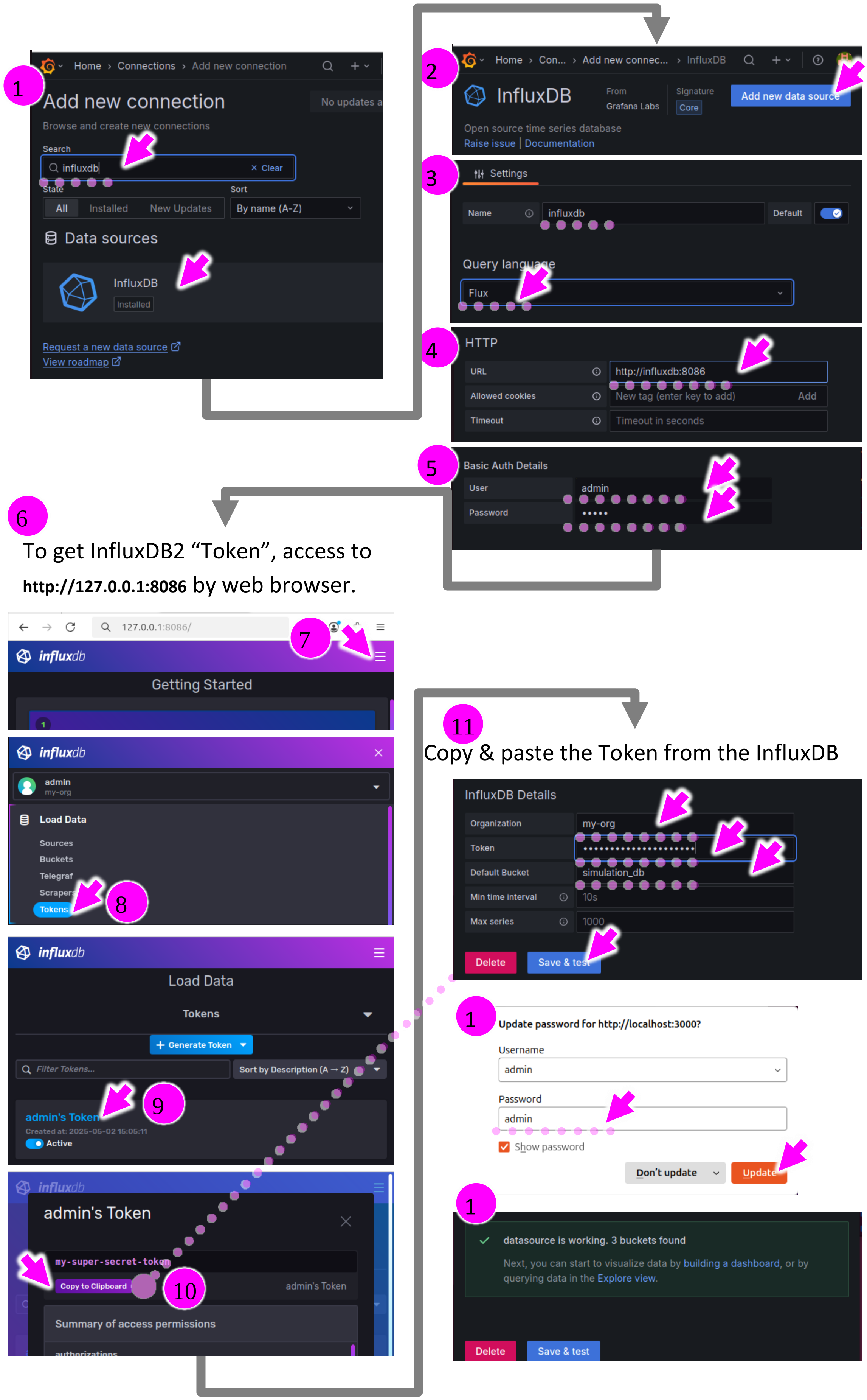


Use admin/admin to login. There is a window to change password, but no need to change (type the original password again).

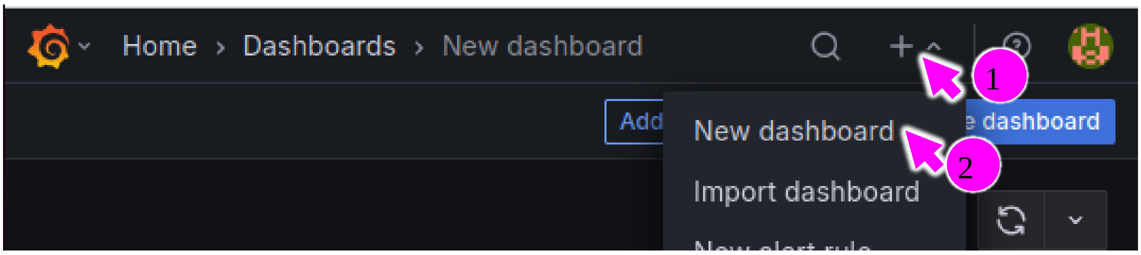
1.4.1 Create New Connection to the Database

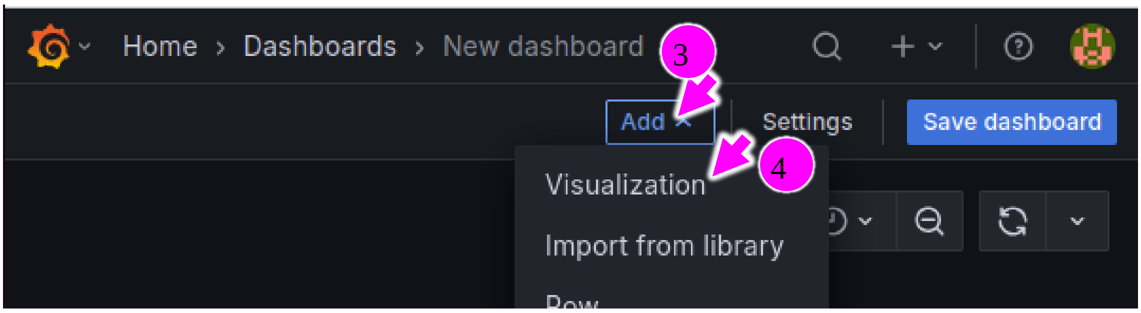
Home > Connections > Add new connection to get InfluxDB2 “Token”, access to `http://127.0.0.1:8086` by web browser.

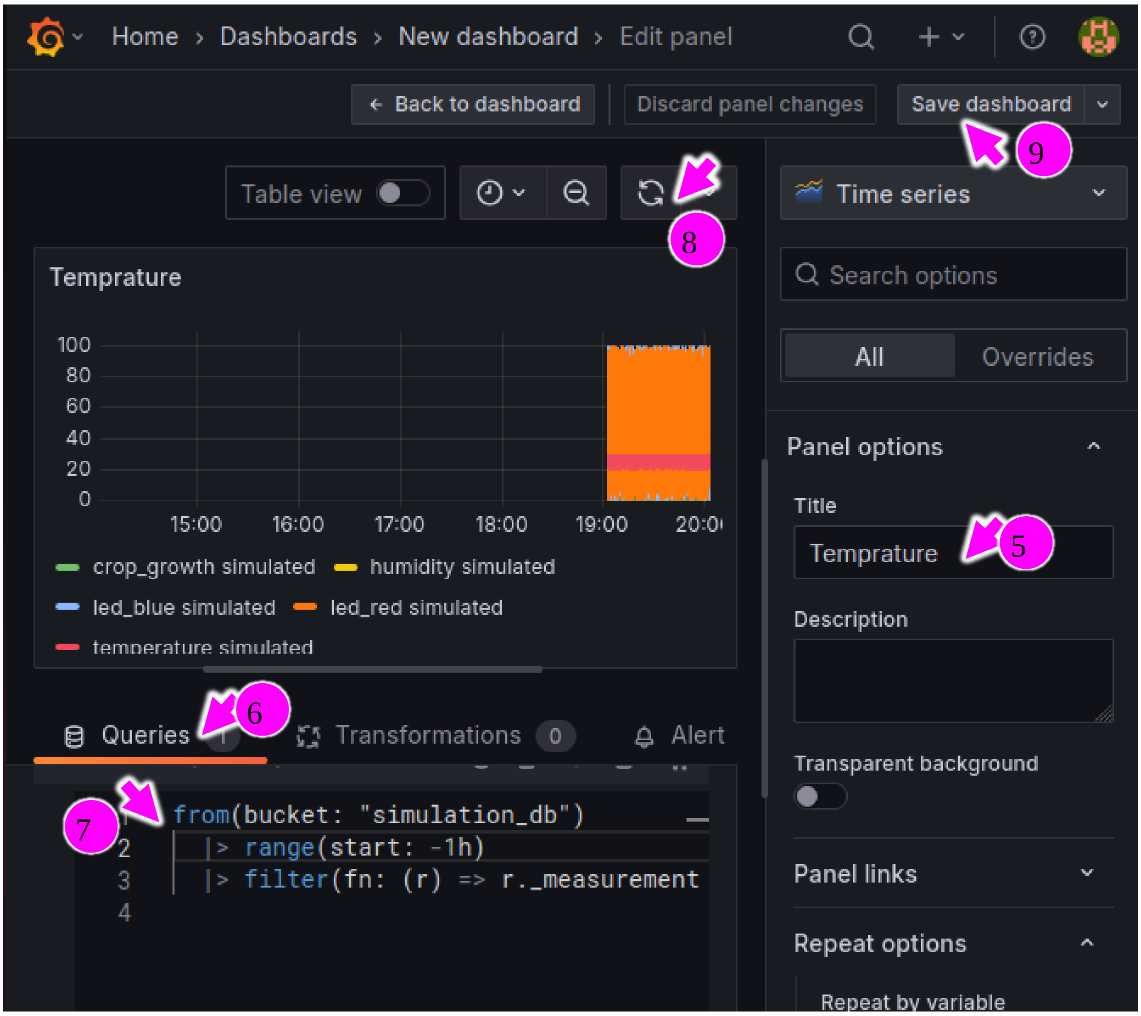
Appendix


1
Home › Connections › Add new connection
Add new connection
No updates a
Browse and create new connections
Search
influxdb
Clear
State
All
Installed
New Updates
Sort
By name (A-Z)
Data sources
InfluxDB
Installed
Request a new data source
View roadmap
2
Home › Con... › Add new connec... › InfluxDB
InfluxDB
From
Grafana Labs
Signature
Core
Add new data source
Open source time series database
Raise issue | Documentation
3
Settings
Name
influxdb
Default
Query language
Flux
4
HTTP
URL
http://influxdb:8086
Allowed cookies
New tag (enter key to add)
Add
Timeout
Timeout in seconds
5
Basic Auth Details
User
admin
Password
6
To get InfluxDB2 "Token", access to
http://127.0.0.1:8086 by web browser.
127.0.0.1:8086/
7
influxdb
Getting Started
influxdb
admin
my-org
Load Data
Sources
Buckets
Telegraf
Scrapers
Tokens
8
influxdb
Load Data
Tokens
Generate Token
Filter Tokens...
Sort by Description (A → Z)
admin's Token
9
Created at: 2025-05-02 15:05:11
Active
influxdb
admin's Token
my-super-secret-token
Copy to Clipboard
10
admin's Token
Summary of access permissions
authorizations
11
Copy & paste the Token from the InfluxDB
InfluxDB Details
Organization
my-org
Token
Default Bucket
simulation_db
Min time interval
10s
Max series
1000
Delete
Save & test
1
Update password for http://localhost:3000?
Username
admin
Password
admin
Show password
Don't update
Update
1
datasource is working. 3 buckets found
Next, you can start to visualize data by building a dashboard, or by querying data in the Explore view.
Delete
Save & test

1.4.2 Add Visualization








```
Query:
from(bucket: "simulation_db")
  |> range(start: -1h)
  |> filter(fn: (r) => r._measurement == "simulation_data")
```

1.4.3 Add Other Visualizations

Temperature Panel

Query
``from(bucket: "simulation_db")``` ``` |> range(start: -1h)``` ``` |> filter(fn: (r) => r._measurement == "simulation_data")``` ``` |> filter(fn: (r) => r._field == "temperature")``` ``` |> aggregateWindow(every: 1m, fn: mean)``` ``` |> yield(name: "mean")```

Humidity Panel

Query
``from(bucket: "simulation_db")``` ``` |> range(start: -1h)``` ``` |> filter(fn: (r) => r._measurement == "simulation_data")``` ``` |> filter(fn: (r) => r._field == "humidity")``` ``` |> aggregateWindow(every: 1m, fn: mean)``` ``` |> yield(name: "mean")```

LED Status (Red) Panel

Query
``from(bucket: "simulation_db")``` ``` |> range(start: -1h)``` ``` |> filter(fn: (r) => r._measurement == "simulation_data")``` ``` |> filter(fn: (r) => r._field == "led_red")``` ``` |> aggregateWindow(every: 1m, fn: mean)``` ``` |> yield(name: "mean")```

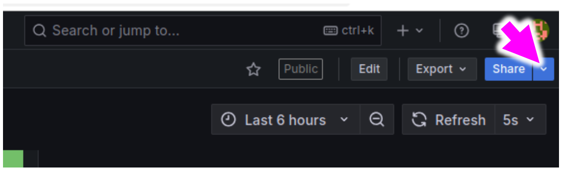
LED Status (Blue) Panel

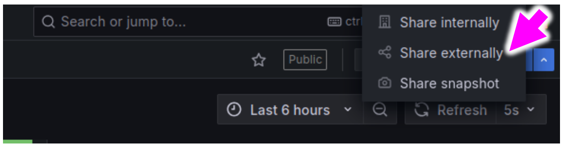
Query
``from(bucket: "simulation_db")``` ``` |> range(start: -1h)``` ``` |> filter(fn: (r) => r._measurement == "simulation_data")``` ``` |> filter(fn: (r) => r._field == "led_blue")``` ``` |> aggregateWindow(every: 1m, fn: mean)``` ``` |> yield(name: "mean")```

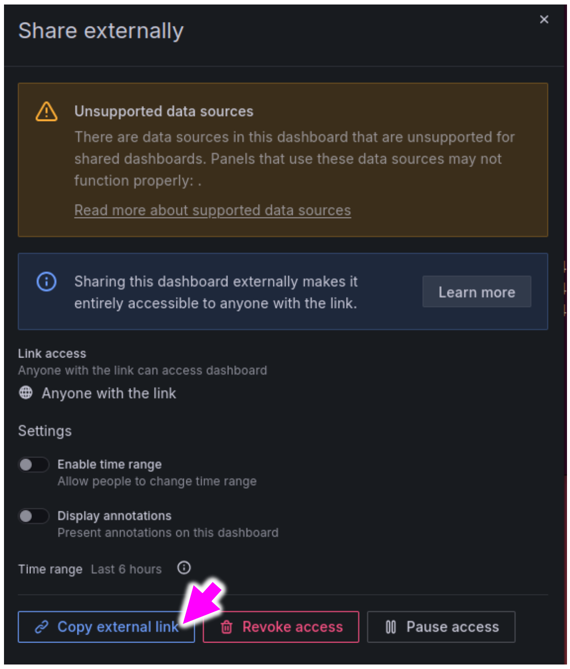
Growth Panel

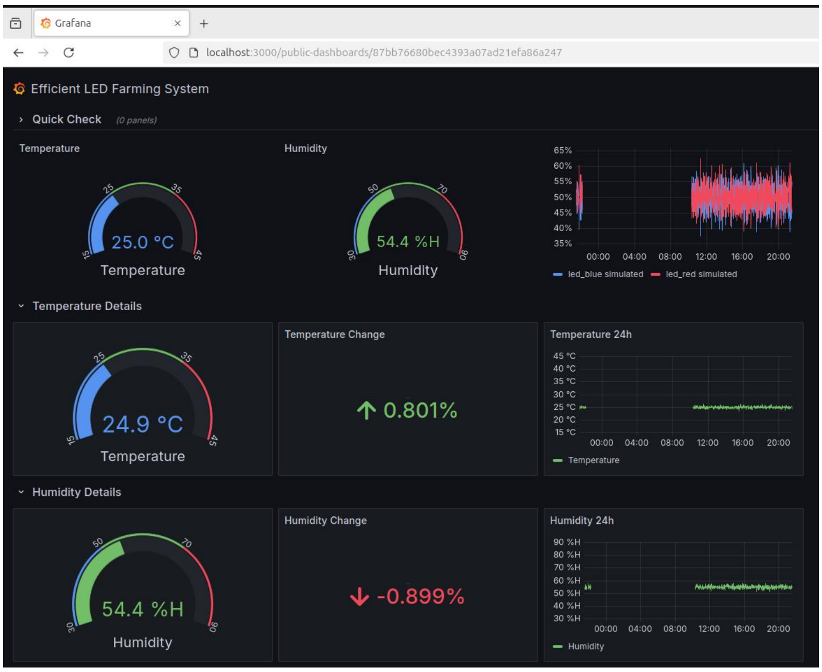
Query
``from(bucket: "simulation_db")``` ``` |> range(start: -1h)``` ``` |> filter(fn: (r) => r._measurement == "simulation_data")``` ``` |> filter(fn: (r) => r._field == "crop_growth")``` ``` |> aggregateWindow(every: 1m, fn: mean)``` ``` |> yield(name: "mean")```

1.5 Share Externally









1.6 Git Version Control

Git is used for version control of the code, and test results from each stage are accumulated.

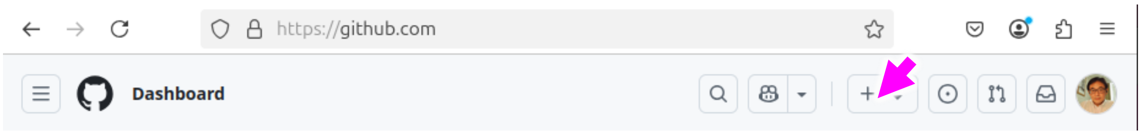
Preparation on Linux side

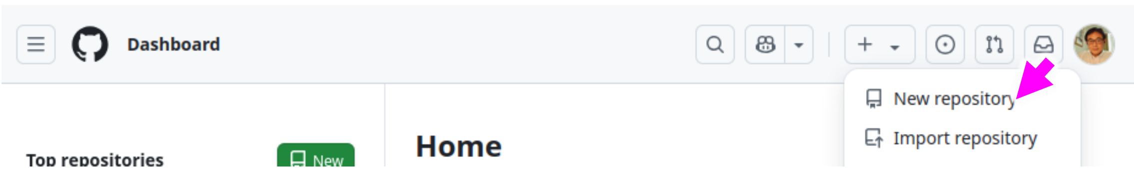
```
$ cd ~/led-simulation
$ git init
```

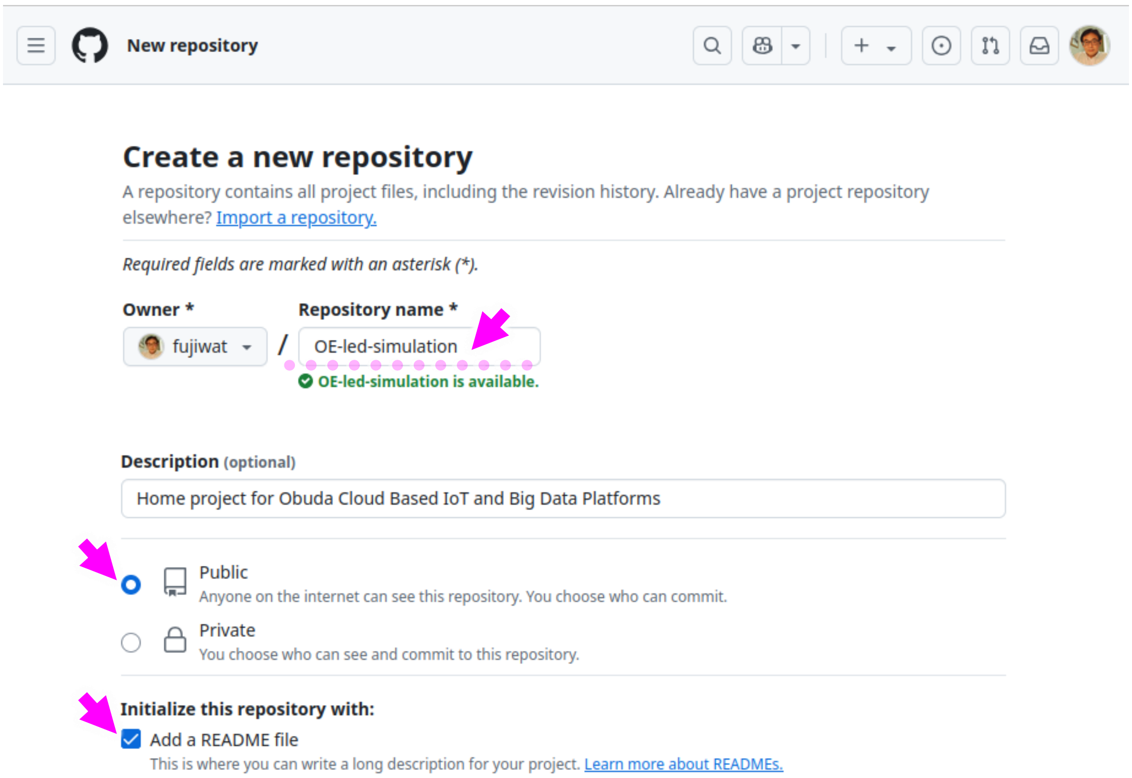
```
Initialized empty Git repository in /home/student/led-simulation/.git/
$ git add .git
$ git commit -m "Initial commit"
[master (root-commit) c85ad29] Initial commit
 4 files changed, 114 insertions(+)
 create mode 100644 Dockerfile
 create mode 100644 docker-compose.yml
 create mode 100644 requirements.txt
 create mode 100644 simulation.py
```

Preparation on GitHub web side

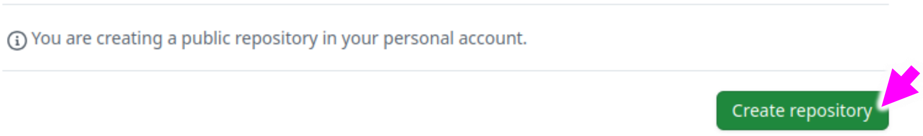
Access to https://github.com by the web browser and login.





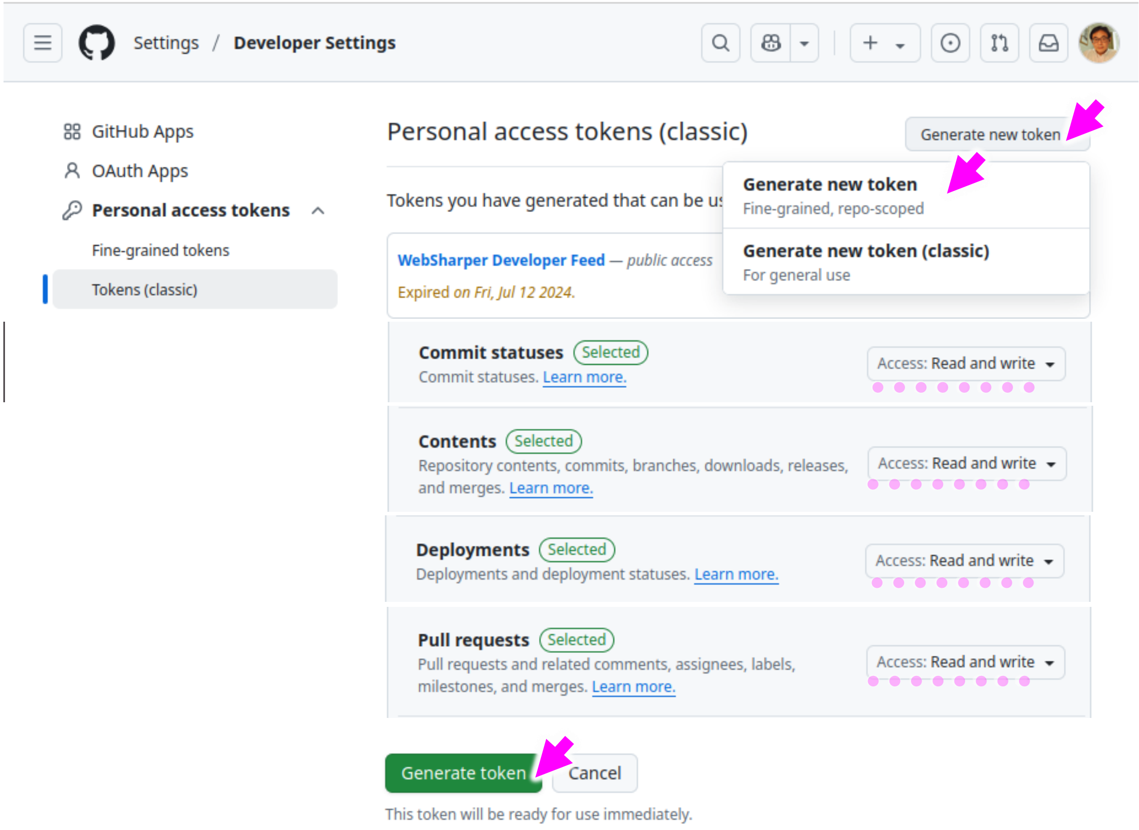


Appendix



Connect local and GitHub

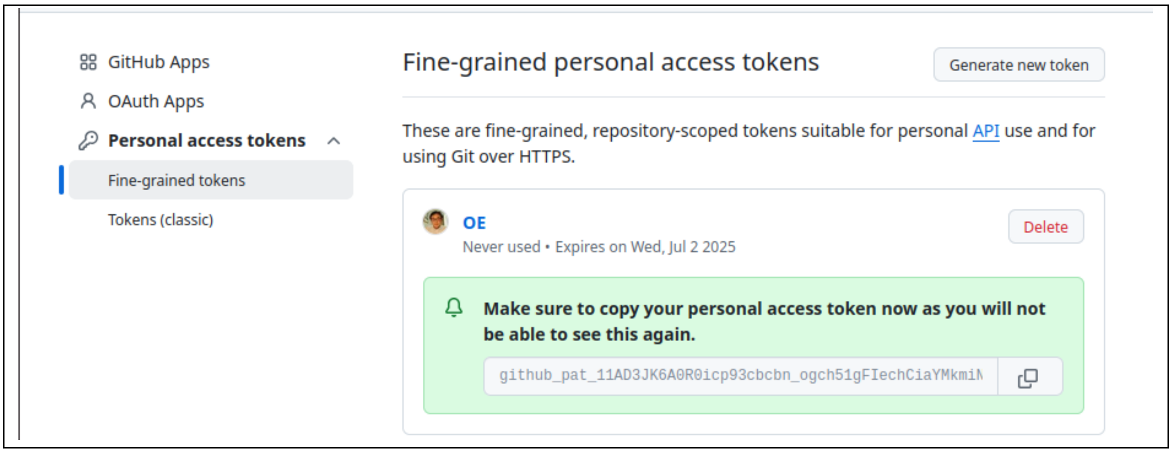
```
$ git remote add origin https://github.com/fujiwat/OE-led-simulation.git
$ git remote -v
origin  https://github.com/fujiwat/OE-led-simulation.git (fetch)
origin  https://github.com/fujiwat/OE-led-simulation.git (push)
```



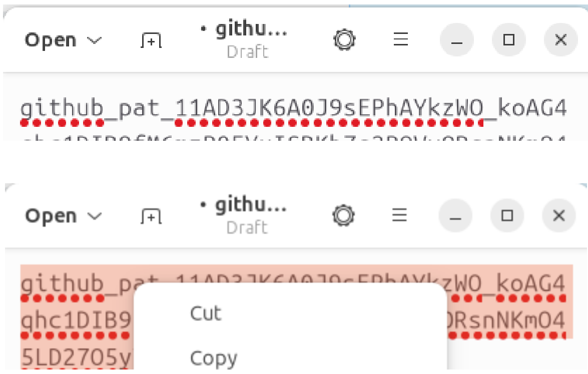
Get the Token

Access to https://github.com/settings/personal-access-tokens

Appendix



Paste it Text Editor to Intermediate

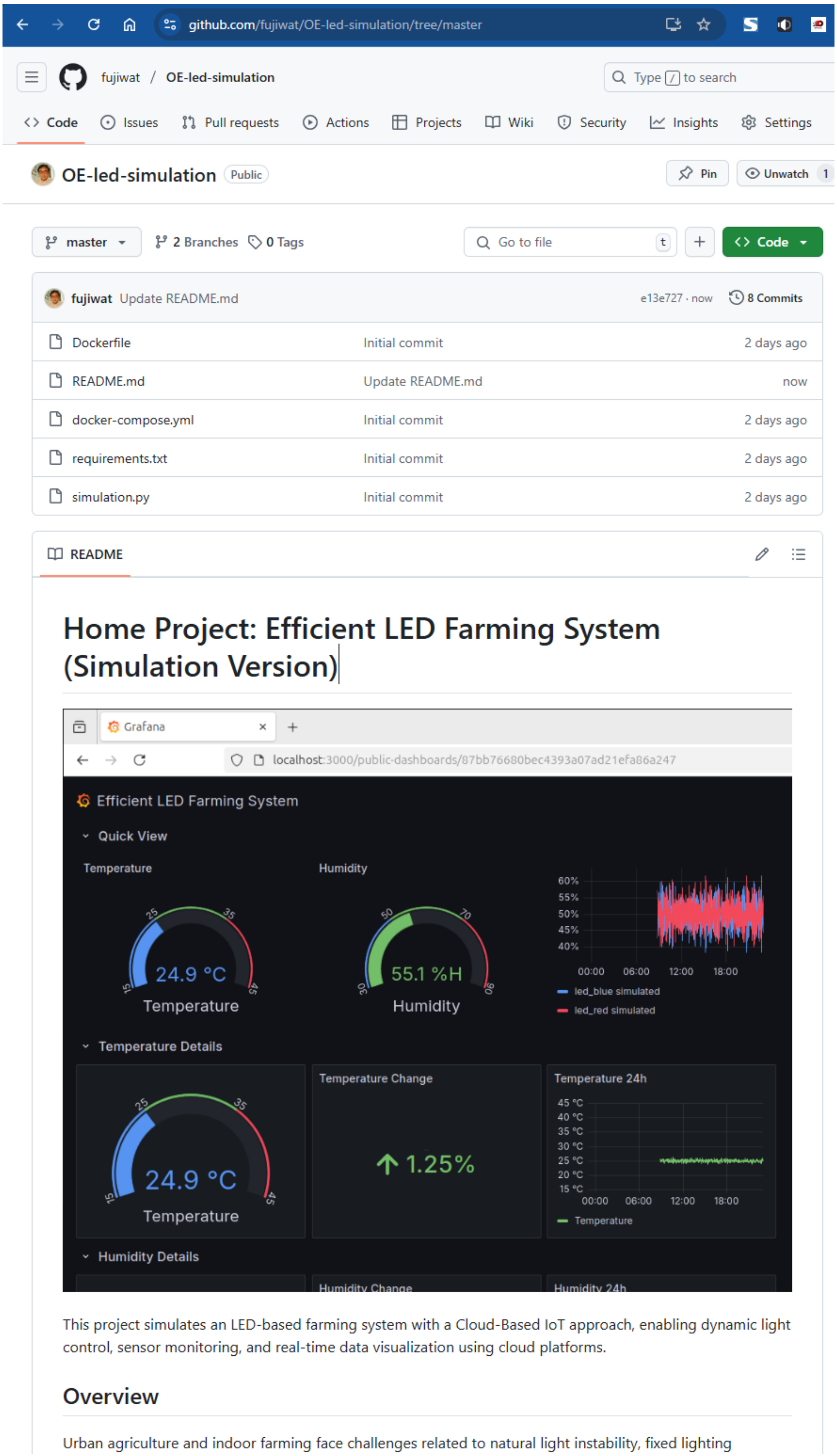


First Push

Paste the token from the clipboard to the place of Password (second line).

```
$ git push -u origin master
Username for 'https://github.com': fujiwat
Password for 'https://fujiwat@github.com':
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 4 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 1.66 KiB | 426.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:      https://github.com/fujiwat/OE-led-simulation/pull/new/master
remote:
To https://github.com/fujiwat/OE-led-simulation.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

Appendix



End of Document