Integration of Tektork IoT kit with TIG Stack

1. Introduction

1.1. Overview of ESP32 in IoT Systems

The ESP32 is a versatile and powerful microcontroller featuring integrated Wi-Fi and Bluetooth capabilities, making it a popular choice for Internet of Things (IoT) applications. It supports a broad range of sensors and peripherals and provides robust wireless communication, which is essential for connecting embedded devices to cloud platforms and local servers. The ESP32's low power consumption, rich peripheral sets, and extensive software ecosystem enable developers to implement diverse IoT solutions, including data acquisition, device control, and remote monitoring. This microcontroller bridges the gap between physical sensors and IoT software frameworks such as MQTT, Node-RED, and cloud services like ThingSpeak and Firebase, facilitating seamless device-to-cloud integration and real-time data visualization.

1.2. Objective

This manual aims to guide users through the comprehensive setup, programming, and deployment of various IoT projects by interfacing Tektork IoT Kit with multiple popular platforms and technologies including ThingSpeak, MQTT brokers, Firebase cloud databases, Node-RED for flow-based programming, and the TIG Stack (Telegraf, InfluxDB, Grafana) for advanced data storage, monitoring and visualization. The objective is to provide a step-by-step reference combining hardware setup, firmware development, platform configuration, and dashboard creation, enabling readers to design scalable, networked IoT systems from scratch with detailed technical insights and best practices.

1.3. Required Hardware and Software

Hardware Requirements

For this manual and related projects, the hardware setup is kept minimal and focused on essential components:

- Tektork IoT Kit: The core microcontroller module with built-in Wi-Fi and Bluetooth connectivity, serving as the primary IoT device.
- USB Type-C Cable: Used to both power the ESP32 board and program it from a PC or laptop.
- PC or Laptop: Required for software development, programming, and interfacing with cloud platforms and local systems.

This simplified hardware setup optimizes project accessibility, making it feasible for users to start IoT development with the Tektork IoT Kit, connecting cable, and a host computer without additional peripheral sensors or modules.

Software Requirements:

The software environment consists of tools and platforms necessary for programming the ESP32, managing MQTT communications, cloud data aggregation, and IoT visualization:

- Arduino IDE or ESP-IDF: Primary development environments for writing and uploading firmware to the ESP32 microcontroller.
- MQTT Broker Service: Public brokers like HiveMQ or Mosquitto provide the messaging backbone for IoT device communication.
- ThingSpeak Account: A cloud-based IoT data platform for acquiring, storing, and visualizing data from connected devices.
- Firebase Console: Google's cloud service offering a real-time database, authentication, and hosting for IoT backend needs.
- Node-RED: A visual flow-based programming tool for creating IoT workflows and dashboards without coding complexity.
- TIG Stack Components:
  Telegraf: An agent for collecting and processing time-series data.
  InfluxDB: A time-series optimized database for storing sensor data.
  Grafana: A dashboard platform to visualize and analyze IoT data effectively.
- Supporting Libraries and Tools: MQTT client libraries for ESP32, HTTP client libraries, and SDKs compatible with ESP32 and relevant platforms.

This software setup targets versatility and scalability, enabling development from basic sensor data visualization up to advanced real-time monitoring and control systems.

2. Setting Up the Tektork IoT Kit Development Environment

Refer the Document tilted

a) Tektork-Kit-ArduinoIDE-Driver-Installation-Notes.pdf

b) Tektork-IoT-Board-Hardware-Manual-V1.0.pdf

3. TIG Stack(Telegraf,InfluxDB,Grafana)

3.1. Introduction to TIG Stack for IoT Monitoring

The TIG stack consists of Telegraf for data collection, InfluxDB for time-series storage, and Grafana for dashboard visualization[Installation.docx].

- Ideal for IoT: It supports real-time, scalable data flows from devices such as ESP32 sensors via MQTT.
- MQTT allows ESP32s to push sensor readings (e.g. potentiometer, LDR) to a broker; Telegraf reads these and writes structured time-series data into InfluxDB.
- Grafana queries InfluxDB for immediate historical and live sensor visualization.

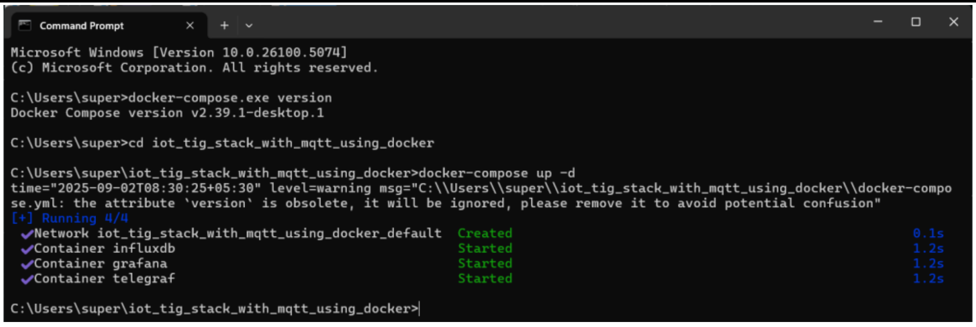
3.2. Installing TIG Stack using Docker Compose

- Clone the repository for a ready-made TIG stack setup:

    - git clone https://github.com/rishikreddycheruku/iot_tig_stack_with_mqtt_using_docker.git
    - cd iot_tig_stack_with_mqtt_using_docker
- The provided docker-compose.yml starts three services: InfluxDB, Telegraf, Grafana.Example (excerpt, modify to match tokens and orgs):

```yaml
version: "3.8"

services:
  influxdb:
    image: influxdb:2.7
    container_name: influxdb
    ports:
      - "8086:8086"
    volumes:
      - influxdb2_data:/var/lib/influxdb2
    environment:
      - DOCKER_INFLUXDB_INIT_MODE=setup
      - DOCKER_INFLUXDB_INIT_USERNAME=admin
      - DOCKER_INFLUXDB_INIT_PASSWORD=admin1234
      - DOCKER_INFLUXDB_INIT_ORG=my-org
      - DOCKER_INFLUXDB_INIT_BUCKET=iotdb
      - DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=mysecrettoken123

  telegraf:
    image: telegraf:1.26
    container_name: telegraf
    depends_on:
      - influxdb
    volumes:
      - ./telegraf.conf:/etc/telegraf/telegraf.conf:ro

  grafana:
    image: grafana/grafana
    container_name: grafana
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=admin
    volumes:
      - grafana_data:/var/lib/grafana

volumes:
  influxdb2_data:
  grafana_data:
```

- Start the stack:
    - docker-compose up -d



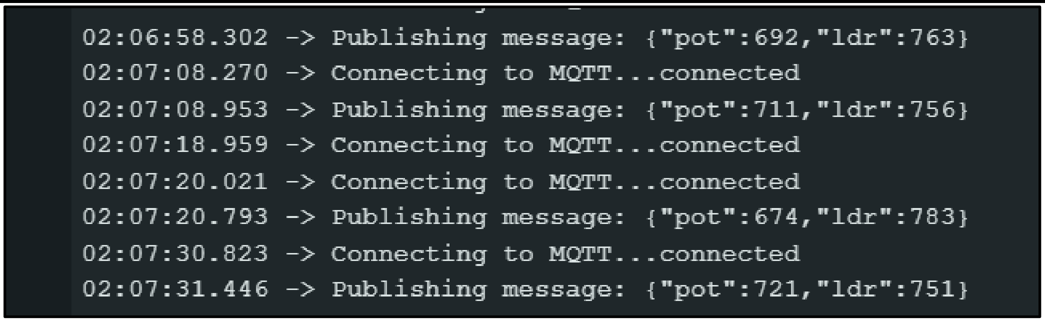
```
Microsoft Windows [Version 10.0.26100.5074]
(c) Microsoft Corporation. All rights reserved.

C:\Users\super>docker-compose.exe version
Docker Compose version v2.39.1-desktop.1

C:\Users\super>cd iot_tig_stack_with_mqtt_using_docker

C:\Users\super\iot_tig_stack_with_mqtt_using_docker>docker-compose up -d
time="2025-09-02T08:30:25+05:30" level=warning msg="C:\\Users\\super\\iot_tig_stack_with_mqtt_using_docker\\docker-compo
se.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 4/4
 ✔Network iot_tig_stack_with_mqtt_using_docker_default  Created                                                     0.1s
 ✔Container influxdb                                    Started                                                     1.2s
 ✔Container grafana                                     Started                                                     1.2s
 ✔Container telegraf                                    Started                                                     1.2s

C:\Users\super\iot_tig_stack_with_mqtt_using_docker>
```

- This boots all services in detached mode. The default ports are 8086 (InfluxDB), 3000 (Grafana).

3.3. Sending ESP32 Data to InfluxDB via MQTT and Telegraf

- ESP32 publishes sensor data as a JSON string via MQTT (HiveMQ broker, topic: iotsensor):

```
#include <WiFi.h>
#include <PubSubClient.h>
const char* ssid = "S23FE";
const char* password = "Sanjey123@";
const char* mqttserver = "broker.hivemq.com";
const int mqttport = 1883;
const char* mqtttopic = "iot/sensor";
const int potPin = 36;
const int ldrPin = 39;
WiFiClient espClient;
PubSubClient client(espClient);

void setupwifi() {
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) delay(500);
}
void reconnect() {
  while (!client.connected()) {
    client.connect("ESP32Client");
  }
}
void setup() {
  Serial.begin(115200);
  setupwifi();
  client.setServer(mqttserver, mqttport);
}
void loop() {
  if (!client.connected()) reconnect();
  client.loop();
  int potValue = analogRead(potPin);
  int ldrValue = analogRead(ldrPin);
  String payload = "{\"pot\":" + String(potValue) + ",\"ldr\":" + String(ldrValue) + "}";
  client.publish(mqtttopic, payload.c_str());
  delay(10000); // matches Telegraf interval
}
```

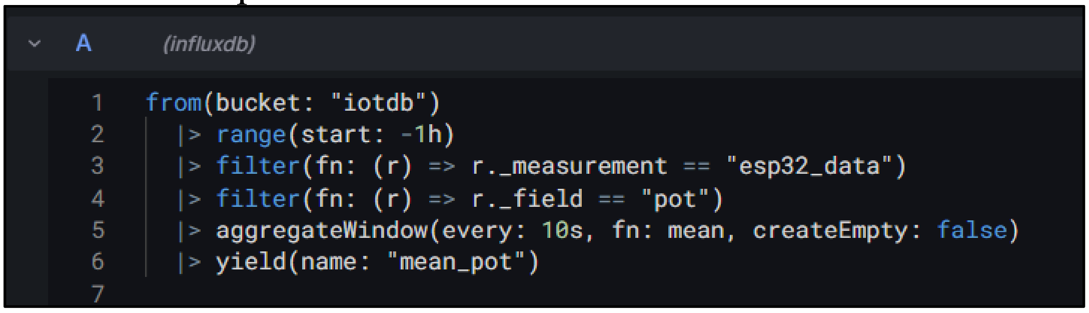

```
02:06:58.302 -> Publishing message: {"pot":692,"ldr":763}
02:07:08.270 -> Connecting to MQTT...connected
02:07:08.953 -> Publishing message: {"pot":711,"ldr":756}
02:07:18.959 -> Connecting to MQTT...connected
02:07:20.021 -> Connecting to MQTT...connected
02:07:20.793 -> Publishing message: {"pot":674,"ldr":783}
02:07:30.823 -> Connecting to MQTT...connected
02:07:31.446 -> Publishing message: {"pot":721,"ldr":751}
```
Telegraf connects to the same broker and topic with inputs.mqtt_consumer:

```
[[inputs.mqtt_consumer]]
  servers = ["tcp://broker.hivemq.com:1883"]
  topics = ["iotsensor"]
  data_format = "json"
  name_override = "esp32data"
```

- It parses the payload and stores each value as a time-stamped measurement in InfluxDB.

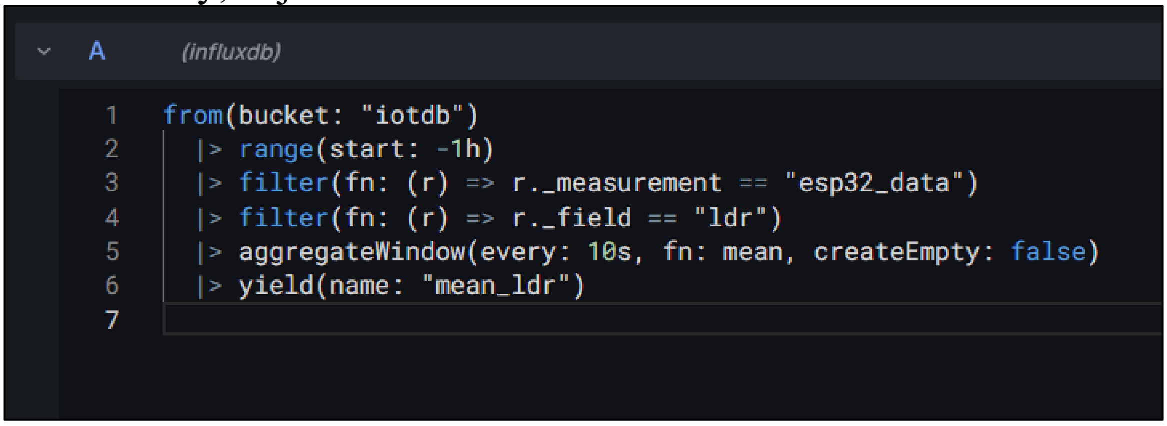
3.4. Querying and Storing Sensor Data in InfluxDB

- Grafana is configured to use InfluxDB as a data source.
- Use Flux queries to retrieve data:


A (influxdb)

```
from(bucket: "iotdb")
  |> range(start: -1h)
  |> filter(fn: (r) => r._measurement == "esp32_data")
  |> filter(fn: (r) => r._field == "pot")
  |> aggregateWindow(every: 10s, fn: mean, createEmpty: false)
  |> yield(name: "mean_pot")
```

- Similarly, adjust for ldr field:


A (influxdb)

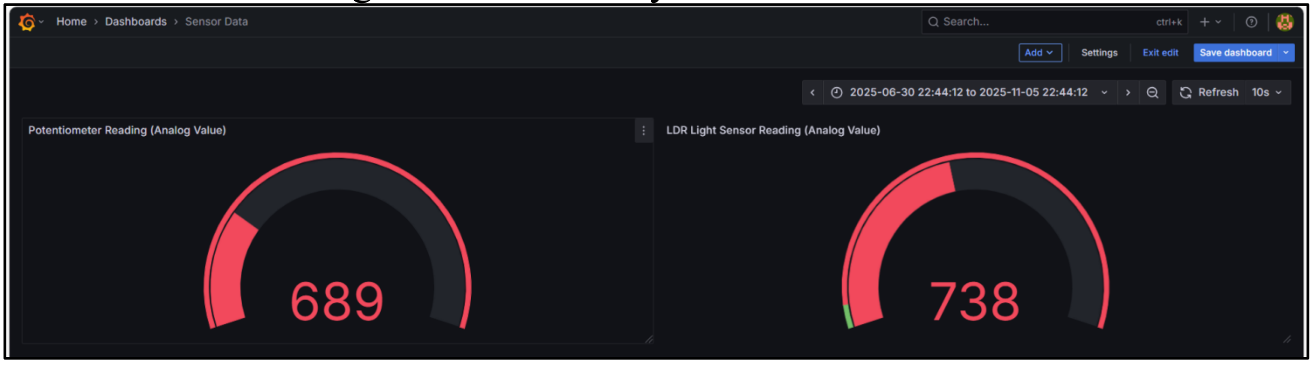
```
from(bucket: "iotdb")
  |> range(start: -1h)
  |> filter(fn: (r) => r._measurement == "esp32_data")
  |> filter(fn: (r) => r._field == "ldr")
  |> aggregateWindow(every: 10s, fn: mean, createEmpty: false)
  |> yield(name: "mean_ldr")
```

- These queries provide instant access to historic and live sensor logs, visualized as line graphs or gauge widgets within Grafana.

3.5. Building Visual Dashboards with Grafana

- Access Grafana at http://localhost:3000 (default) and log in with configured credentials.
- Add InfluxDB as a data source (token, org, bucket as set in Docker Compose).
- Create panels and use Flux queries above to visualize each sensor:
  - Panel: "Potentiometer Reading Analog Value"

    - Panel: "LDR Light Sensor Reading Analog Value"
- Set auto-refresh interval for live updates. Style panels with custom colors, add value axes and legends to suit analysis needs.


Home › Dashboards › Sensor Data
Search... ctrl+k
Add
Settings
Exit edit
Save dashboard
2025-06-30 22:44:12 to 2025-11-05 22:44:12
Refresh 10s
Potentiometer Reading (Analog Value)
689
LDR Light Sensor Reading (Analog Value)
738

3.6. Working Model Demonstration(ESP32 to TIG Stack Visualization)

- Boot Docker Compose (step 9.2), flash the ESP32 code (step 9.3), and confirm MQTT publish in serial and broker (topic: iotsensor).
- Live data flows:
    - ESP32 → MQTT (HiveMQ) → Telegraf → InfluxDB.
    - Grafana queries InfluxDB and presents live graphs and logs (both historical and real-time) for sensor data on the dashboards.
- Validate each pipeline step by reviewing logs in the containers and UI widgets in Grafana.

This model delivers a complete, robust pipeline for IoT sensor collection, storage, and visualization, directly matching your ESP32 hardware pin assignments and software settings.

4. Comparative Analysis of Tools

4.1. ThingSpeak vs Google Sheets vs Firebase vs TIG Stack

Feature	**ThingSpeak**	**Google Sheets**	**Firebase Realtime DB**	**TIG Stack(Telegraf,InfluxDB, Grafana)**
Integration	Direct HTTP request, MQTT	HTTP GET/POST (Apps Script)	Client libraries, REST, token auth	MQTT → Telegraf input, Docker Compose setup
Data Model	Channel-fields (numeric)	Spreadsheet (tabular)	Hierarchical JSON, nested	Time-series, measurements, tags

Visualization	Basic built-in charts	Built-in, flexible, live	None native, 3rd-party dashboards	Grafana: Rich, real-time, customizable
Storage	Cloud, limited history	Cloud, unlimited (if <10M)	Cloud, scalable, offline sync	On-premise/cloud, retention policies
Security	Channel keys/API keys	Script URL (can be restricted)	Auth tokens, rules, granular	Credentials, tokens, Docker isolation
Complexity	Easiest setup	Beginner, scripting required	Moderate, needs setup & key mgmt	Advanced, best with Docker basics
Best Use	Academic/Pilot, quick view	Student logging, basic dashboards	Mobile IoT, remote actuation	Industrial, advanced IoT analytics

4.2. MQTT with Node-RED vs MQTT Explorer

Feature	**MQTT with Node-RED**	**MQTT Explorer**
Purpose	Automation, visualization, dashboard control	Debugging, topic and payload inspection
Integration	Drag-and-drop flows, connect to hardware/services	Direct connect to broker
Visualization	Rich dashboards (switches, gauges, charts, UI)	Topic tree, live payloads, raw view
Control	User-initiated, event-based flows (alerts, logic)	Can manually publish/test, no logic

Scripting	JavaScript, Node-RED function nodes	None, GUI only
Best Use	End-user dashboards, automation, monitoring	Protocol debug, rapid troubleshooting

4.3. Recommended Applications for Academic and Industrial Projects

- ThingSpeak
  - Academic: Quick prototyping, classroom demonstrations, teaching MQTT and HTTP IoT basics.
  - Industrial: Limited—used for proofs-of-concept or simple remote logging.
- Google Sheets
  - Academic: Student projects, easy data logging and live graphing, IoT experiment tracking.
  - Industrial: Not recommended for critical or high-speed logging (API limits and scale).
- Firebase Realtime Database
  - Academic: Mobile/web IoT integrations, app-based monitoring/control, real-time updates projects.
  - Industrial: Mobile control panels, scalable remote actuation, multi-device system sync.
- TIG Stack (Telegraf, InfluxDB, Grafana)
  - Academic: High-powered demonstration of scalable IoT pipelines, cloud-native concepts.
  - Industrial: Robust, professional-grade, resilient: high-frequency logging, analytics, alerting, and visualization.
- MQTT with Node-RED
  - Academic: Visual programming, automation, experimentation, hands-on IoT labs.
  - Industrial: Edge automation, HMI design, integration with enterprise software and industrial protocols.
- MQTT Explorer
  - Academic: Teaching/debugging MQTT concepts and troubleshooting flows.
  - Industrial: Pre-deployment checks, maintaining message integrity, protocol analysis.

5. Conclusion and Future Scope

5.1. Summary of Implementations

This manual documented the integration of Tektork IoT Kit based systems with multiple platforms for data acquisition, visualization, and remote control. Implementations covered MQTT messaging (HiveMQ, Mosquitto), direct dashboarding with Node-RED, cloud logging using Google Sheets and Firebase, as well as advanced analytics with the TIG stack (Telegraf, InfluxDB, Grafana).

Real-time monitoring and control were enabled through both public and self-hosted infrastructures, demonstrating use cases ranging from simple classroom prototypes to scalable industrial data flows.
A modular approach was used, making each subsystem (sensors, dashboards, databases) inter-operable and adaptable for future expansions.

5.2. Challenges Faced and Solutions

- Network Reliability:Unstable Wi-Fi caused intermittent sensor reporting. Solution: Implemented robust reconnection and status-check logic in ESP32 code for both MQTT and HTTP endpoints.
- Data Synchronization:Timestamp mismatches and duplicate entries occurred on some cloud platforms. Solution: Ensured all log messages included device-side timestamps, leveraged batch logging when supported.
- Security and Access Control:Initial open access for quick prototyping led to data exposure. Solution: Gradually introduced API keys, restricted access, and rule-based permissions in Firebase, ThingSpeak, and InfluxDB configurations.
- Integration Complexity:Docker Compose setups and Telegraf configuration for the TIG Stack were initially complex. Solution: Detailed stepwise documentation and use of version-controlled config files minimized errors and improved reproducibility.
- Scalability Limits:Cloud spreadsheets and basic dashboards struggled at higher sample rates. Solution: Transitioned high-frequency data to InfluxDB and Grafana, keeping Sheets and Firebase for summary or mobile-friendly views.

5.3. Future Extensions(AI and IoT,Edge Computing)

- AI & IoT Integration:Future systems can utilize on-device or cloud-based machine learning for anomaly detection, predictive maintenance, and adaptive control, leveraging real-time data flows established here.Models may be deployed using TensorFlow Lite on ESP32 or by streaming sensor data to cloud AI services for pattern recognition and forecasting.
- Edge Computing:With increasing hardware capabilities, preprocessing (e.g., local data filtering, initial analysis, secure encryption) can occur directly on the ESP32 or edge gateways before sending to the cloud, reducing latency and bandwidth usage.
- Expanded Interoperability:Integration with home automation tools (e.g., Home Assistant), voice assistants, or industrial protocols (Modbus, OPC-UA) offers further real-world applications.
- Automated Scaling and Auto-Healing:Container orchestration (e.g. Kubernetes) and CI/CD pipelines can automate deployment, scaling, and recovery processes for large deployments, building on the Docker-based architectures used here.

Adopting these directions will ensure that IoT setups remain future-ready, secure, and adaptable to new academic and industrial challenges.