Smart Water Management System

Ayesha Jan
Maha Ibrahim
Anunitha K U

Github:
https://github.com/Ayesha-Jan/water-management

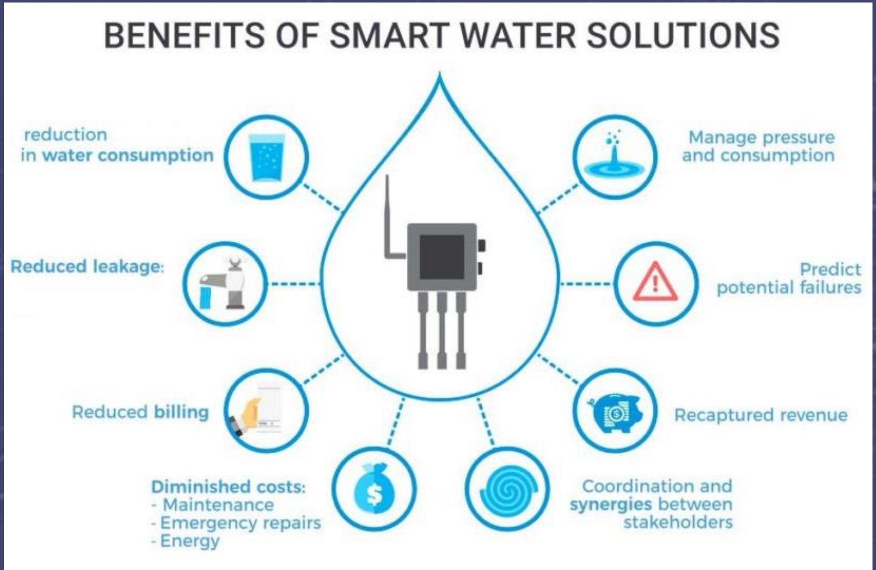
01 Problem Definition

The Problem

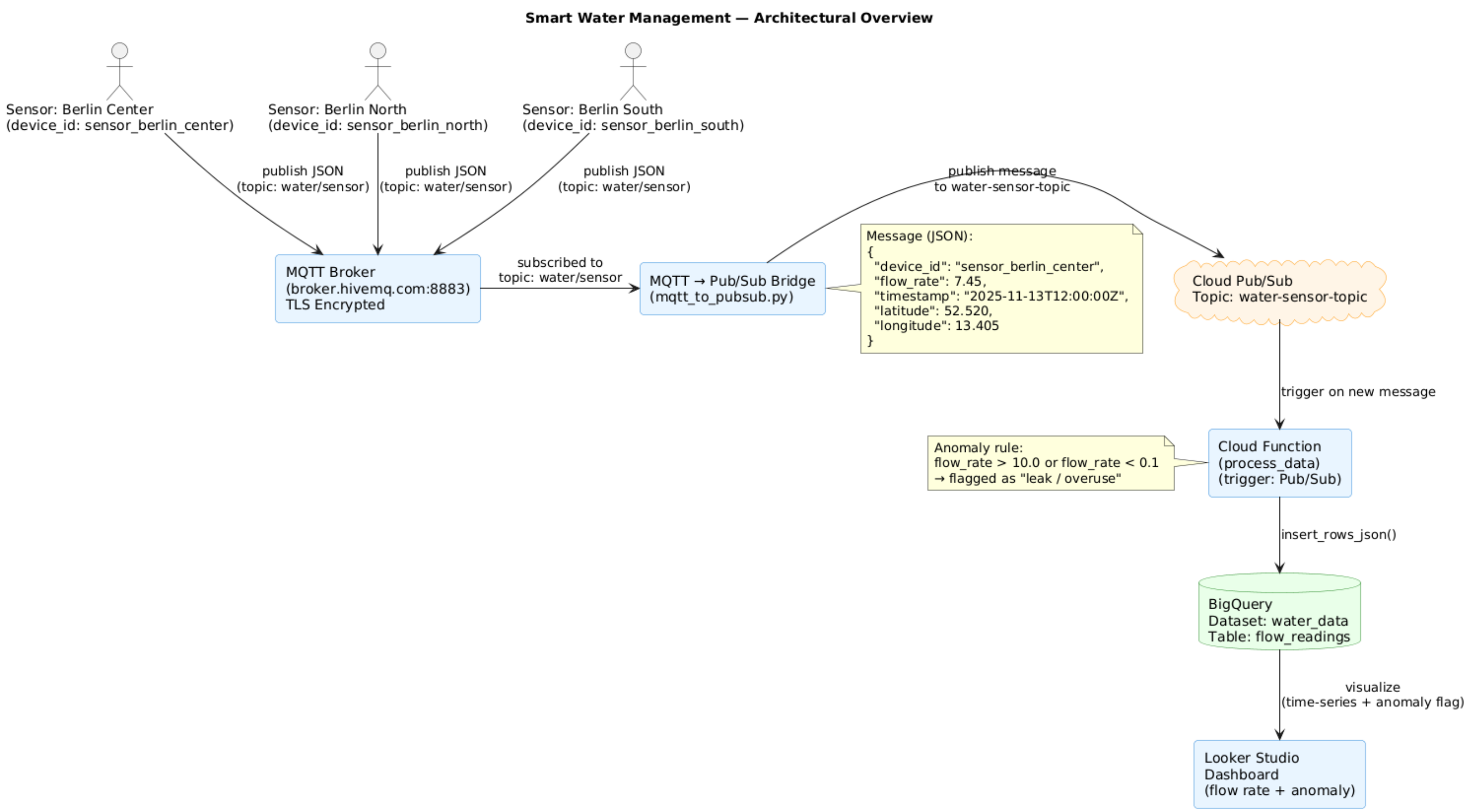
- Water wastage and undetected leaks cause massive resource loss in cities, households, and agriculture.
- Traditional systems lack real-time monitoring and anomaly detection.

The Need

- Continuous data collection requires distributed IoT sensors at the edge and cloud scalability for storage and analysis.
- A purely local setup cannot handle large-scale analytics; a cloud-only approach lacks low-latency detection at the source.
- A hybrid IoT → Cloud approach enables both real-time alerting and historical trend insights.



Architectural Overview



02 The Edge/IoT Implementation

Water Sensor Simulator

Acts as a **virtual IoT water-flow device** that continuously generates sensor readings from three locations across Berlin.

Generates the following data fields:

- **Flow Rate (L/min):** Random values between 0.0 and 12.0 L/min.
- **Timestamp:** UTC time in ISO 8601 format for chronological tracking in BigQuery.
- **GPS Coordinates:** Fixed latitude and longitude per sensor (used for spatial analysis).
- **Device ID:** Unique per sensor (e.g., `"sensor_berlin_center"`).

Implements:

- `paho-mqtt` library for connecting and publishing to MQTT broker.
- Iterates through all three sensors, publishing a reading every 2 seconds, then pauses 5 seconds before the next cycle – ensuring a steady, near-real-time data stream.

Local Data Processing

Key Functions at the Edge:

- **Looping & Timing:** ensures continuous, near-real-time data flow from all sensors
- Formats sensor readings as JSON for cloud ingestion
- **MQTT Publishing:** Sends readings to HiveMQ broker over TLS (port 8883).

Benefits:

- Provides continuous data from multiple virtual sensors.
- Data is sent at regular intervals to the MQTT broker.

JSON Payload Examples

Simulator.py:

- Sending: {'device_id': 'sensor_berlin_center', 'flow_rate': 10.49, 'timestamp': '2025-11-12T15:00:52Z', 'latitude': 52.52, 'longitude': 13.405}

Mqtt_to_pubsub.py:

- Received MQTT message: {"device_id": "sensor_berlin_center", "flow_rate": 10.49, "timestamp": "2025-11-12T15:00:52Z", "latitude": 52.52, "longitude": 13.405}

Technologies used

- **Python 3** - for scripting the simulator logic.
- **paho-mqtt library**- connects and publishes to MQTT broker.
- **HiveMQ MQTT Broker (broker.hivemq.com)** - open-source lightweight message transport.
- **TLS (Port 8883)** - ensures encrypted and authenticated data transfer.

MQTT Communication Setup

Public HiveMQ Broker(`broker.hivemq.com`)

Flow of Communication:

Simulator (`simulator/simulator.py`) → HiveMQ Broker → Python bridge (`mqtt_to_pubsub.py`) → GCP Pub/Sub → Cloud Function

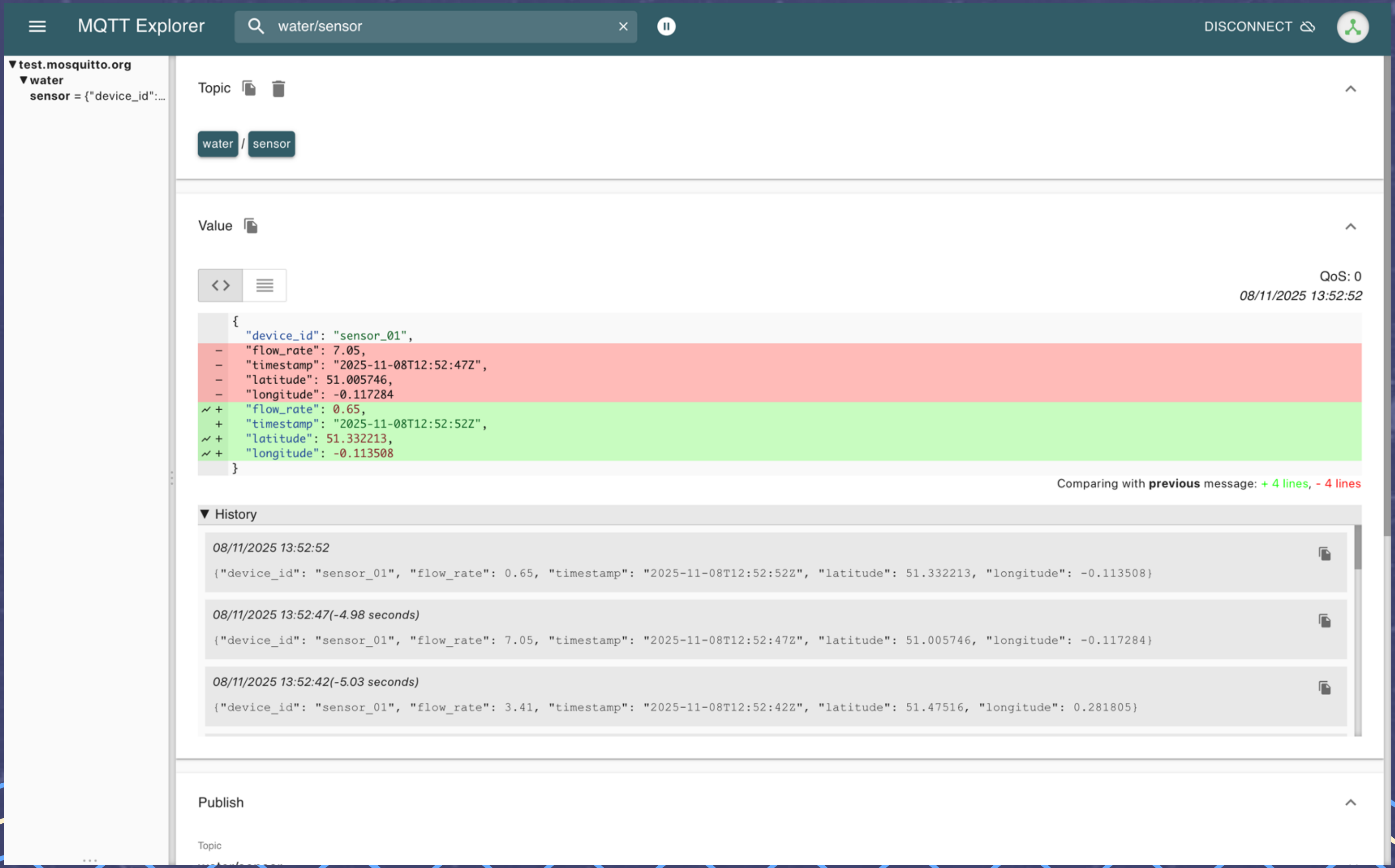
Security Mechanism:

- TLS encryption on port 8883
- Ensures encrypted communication between simulator and broker

MQTT Advantages:

- Lightweight protocol, ideal for sending JSON messages from multiple virtual sensors.
- Supports continuous publishing at fixed intervals (every 2-5 seconds per sensor).
- Simple and reliable for IoT edge devices.

MQTT Explorer



03 The Cloud Implementation

Overview of Cloud Architecture

Core Components:

- Pub/Sub
- Cloud Function
- BigQuery
- Looker Studio

Principles:

- **Event-Driven Architecture** - Cloud Function triggers only when a new message arrives.
- **Serverless Computing** - GCP manages scaling; no server setup required.
- **Pay-per-Use Cost Model** - you are billed only for messages processed, storage used, and queries run.

Service Model

Chosen Model:

- **Platform-as-a-Service (PaaS) -** BigQuery, Pub/Sub, Looker Studio
- **Function-as-a-Service (FaaS) -** Cloud Functions

Explanation:

- Cloud Functions run code only when triggered by events, with no server management.
- BigQuery, Pub/Sub, and Looker Studio are fully managed PaaS services—Google handles scaling, maintenance, and infrastructure.
- The system uses a pay-per-use model, so you are billed only for messages processed, queries run, and storage used.

Justification:

- Minimizes operational overhead and cost, perfect for IoT streams that arrive intermittently.
- GCP automatically provisions resources on demand, eliminating the need for manual server management.

Cloud Infrastructure

Component	Function	Key Benefit
MQTT Broker	Simulator publishes sensor data to broker.hivemq.com	Lightweight, reliable messaging.
Cloud Pub/Sub	Ingests and queues MQTT messages	Scalable, asynchronous messaging
Cloud Function	Transforms and validates each message	Serverless, event-driven logic
BigQuery	Stores and queries processed sensor data	Fast SQL analytics on massive datasets
Looker Studio	Builds dashboards from BigQuery data	Easy visualization and sharing

Programming Model

Event-Driven Architecture:

Trigger: Automatically invoked by new message in Pub/Sub topic.

Core Logic: anomaly = flow_rate > 10.0 or flow_rate < 0.1

- Marks the reading as anomalous if leak or unusual consumption detected.
- Converts each message to structured JSON for BigQuery.
- Logs any insertion errors and confirms successful writes.

OUTPUT EXAMPLE:

Device	Flow Rate	Anomaly	Timestamp
sensor_berlin_center	11.3	TRUE	2025-11-10T12:05Z
sensor_berlin_north	4.8	FALSE	2025-11-10T12:06Z
sensor_berlin_south	0.0	TRUE	2025-11-10T12:07Z

IoT-to-Cloud Data Pipeline

- **Simulator → MQTT Broker:** `simulator.py` publishes flow readings from three virtual sensors to `broker.hivemq.com` on topic `water/sensor`.
- **MQTT Bridge → Pub/Sub Topic:** `mqtt_to_pubsub.py` subscribes to `water/sensor` and forwards every message to GCP Pub/Sub topic `water-sensor-topic`.
- **Pub/Sub → Cloud Function:** Cloud Function `process_data` triggers automatically for each Pub/Sub message.
- **Cloud Function:** Decodes JSON, reads device info, flow rate, timestamp, latitude, longitude, calculates anomaly flag, inserts record into BigQuery.
- **BigQuery:** Stores structured sensor data for analytics.
- **Looker Studio:** Connects to BigQuery to visualize flow trends, detect anomalies, and monitor multiple sensors in near-real-time.

Cloud Benefits

- **Scalability:**
  Pub/Sub and Functions scale automatically to millions of messages.
- **Cost Efficiency:**
  Fully pay-per-use model – no idle resources, no VM management.
- **Security:**
  Built-in encryption for Pub/Sub messages.
  IAM roles and service accounts for controlled access.
- **Availability & Reliability:**
  GCP provides **high availability and reliability**
  Fault-tolerant by design; automatic failover and retry.
- **Integration:**
  BigQuery connects directly to Looker Studio and AI/ML APIs for future predictive analysis.

04 Economic, Risks, and Scientific Method

Economic Consideration

- The project follows **Google Cloud's pay-per-use model**, which charges only for actual usage.
- **Cloud Functions** pay only for runtime and invocations.
- **Pub/Sub** near-zero cost for small-scale IoT since free tier covers millions of messages.
- **BigQuery** billed per stored data and query usage; efficient for time-series analysis.
- **Advantage:** No fixed server costs → scalable for startups or city pilots.
- **Total Estimated Monthly Cost (Prototype):** < €5 using GCP free credits.

Service	Monthly Cost (Est:)
Cloud Functions	~€0.00 (free tier)
Pub/Sub	~€0.00
BigQuery	~€1–2
Total	**< €5 / month**

Opportunity / Risk

Primary Risk: Data privacy and transmission security due to use of a public MQTT broker(Port 1883).

Mitigation Strategies:

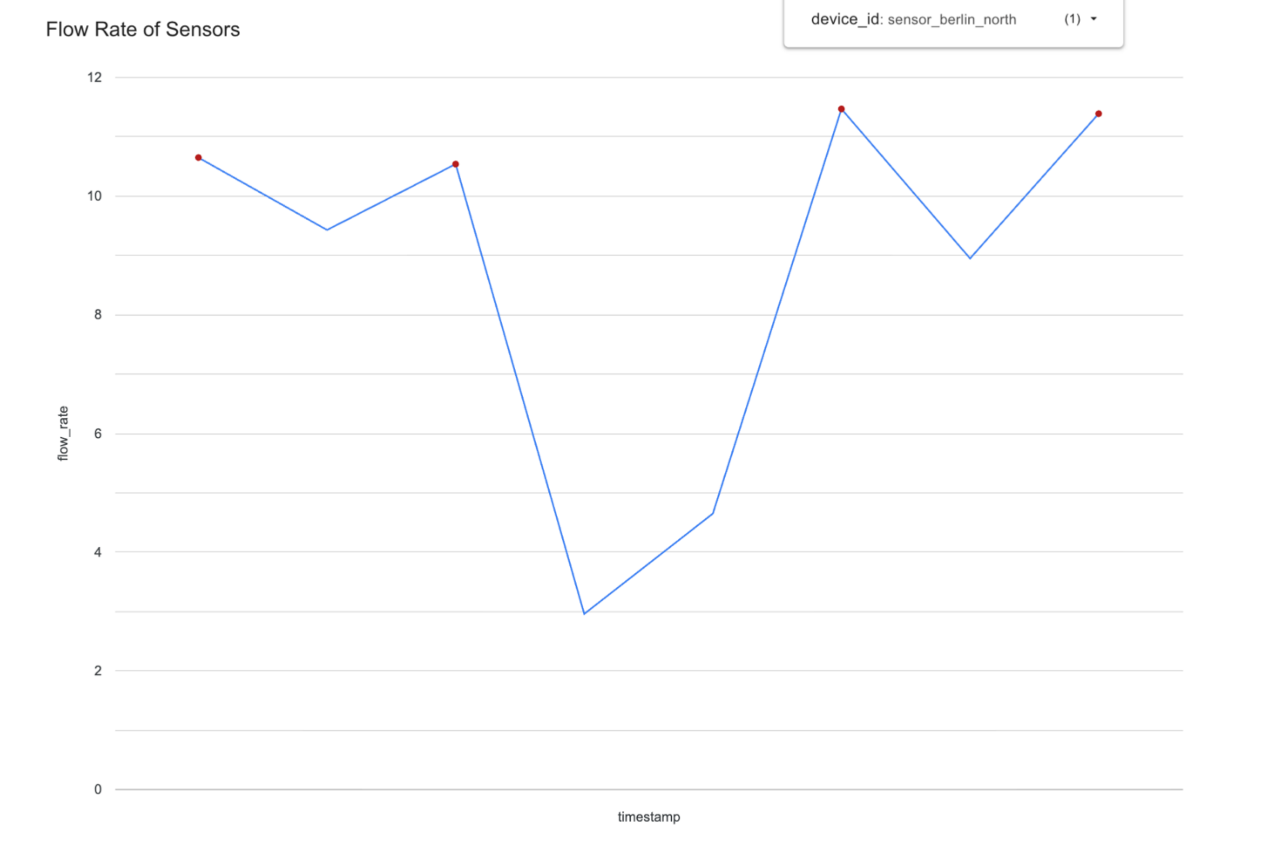
- Enforced **TLS (port 8883)** for encrypted communication.
- Implemented **IAM permissions** to restrict write access in BigQuery.
- Future step: deploy a **private MQTT broker** (e.g., EMQX on a VM) or use **IoT Core replacement** for authenticated messaging.
- Opportunity: Can easily scale to smart city deployments and integrate with other IoT ecosystems.

Scientific Working Method

- **Observation:** Water systems often face undetected leaks or irregular usage patterns.
- **Hypothesis:** An IoT + Cloud data pipeline can detect abnormal water flow in near real time.
- **Experiment Steps:**

1. Simulated flow sensor data using Python and MQTT.
2. Forwarded data via MQTT → Pub/Sub → Cloud Function → BigQuery.
3. Applied simple anomaly rule: flow_rate $>$ 10 or $<$ 0.1.
    - **Result:** System reliably captured and flagged anomalies.
    - **Conclusion:** The hypothesis is validated – the architecture works for real-time water flow monitoring.

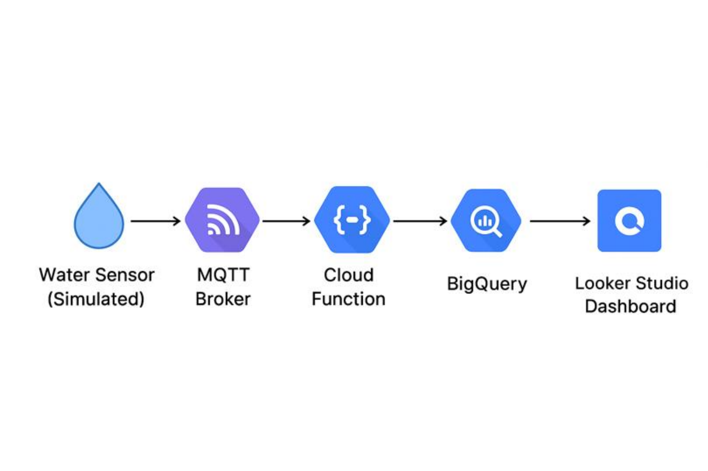
Looker Studio Visualization



05 Conclusion and Future Work

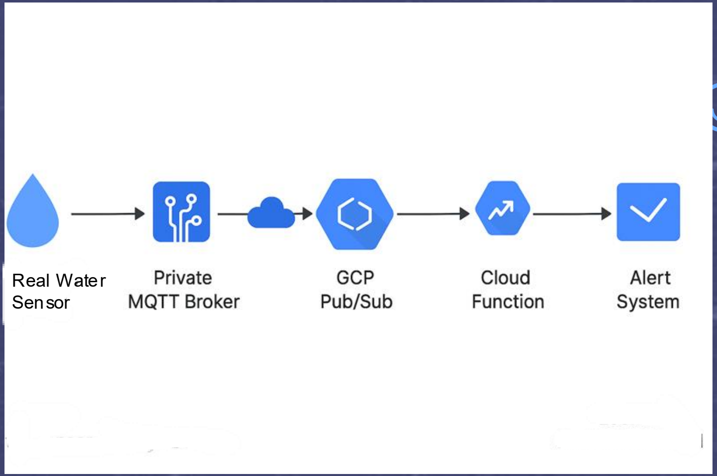
Summary of Achievement

- Developed a **fully functional IoT-to-Cloud data pipeline** using open-source and GCP services.
- Achieved **real-time data flow** from sensor → MQTT → Pub/Sub → Cloud Function → BigQuery.
- Built a **Looker Studio dashboard** for live visualization and anomaly highlighting.
- Demonstrated that **serverless architecture** reduces costs, simplifies scaling, and maintains reliability.
- System successfully identifies abnormal flow rates – proving the model's efficiency.



Future Work

- **Integrate real sensors** (YF-S201, ultrasonic water meters) for real-world testing.
- **Add alerting mechanisms** push notifications or email via Cloud Functions.
- **Machine Learning extension** use BigQuery ML to predict leaks or consumption spikes.
- **Edge Computing optimization** filter noise locally before cloud upload to save cost.
- **Deploy as a multi-zone infrastructure** supporting multiple smart city regions.



Thanks!

Do you have any questions?