

# Scientific Report: Cloud-based Monitoring System for Temperature and Humidity in a Warehouse

- Socoteală Andrei-Alin
- Apricopoi Andrei-Constantin

## System Architecture Overview

The monitoring system comprises three major interconnected components, each fulfilling distinct yet complementary roles:

### 1. Data Ingestion and Communication (Component #1)

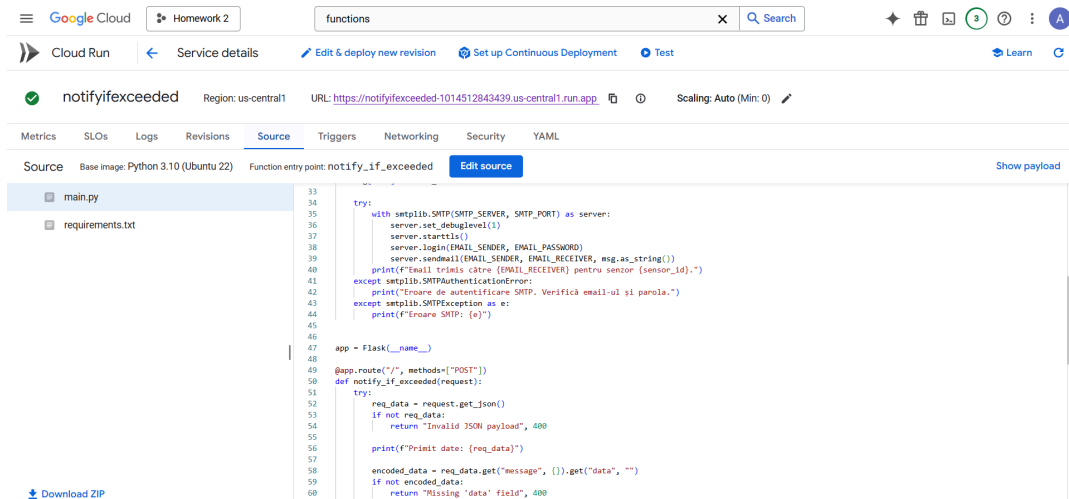
- **Clients (Sensors):** Deployed throughout the warehouse to measure temperature and humidity. Each sensor (would be ideal, but we used randomly generated data) is configured to periodically send data (sensor ID, temperature, humidity) to a central server using the WebSocket protocol. WebSocket, implemented in Python, ensures low-latency, bidirectional communication, making it suitable for real-time monitoring.
- **Server:** The server receives incoming data streams from sensors, validates the data, and transmits it to a Google Cloud Pub/Sub topic. By using Pub/Sub, the system decouples data producers (sensors) from consumers, enhancing scalability and flexibility.

### 2. Data Processing and Storage (Component #2)

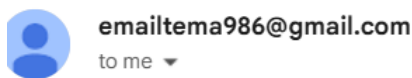
- **Google Kubernetes Engine (GKE):** A GKE cluster subscribes to the Pub/Sub topic, processes incoming data, and stores it in both Cloud Firestore and BigQuery. The GKE deployment uses containerized microservices, ensuring modularity, scalability, and fault tolerance.
  - **Cloud Firestore:** Stores processed data for fast, real-time querying and application-level usage. Firestore's NoSQL structure suits the rapid retrieval and updating of sensor data.
  - **BigQuery:** Stores data for large-scale analytics, enabling advanced querying capabilities. BigQuery datasets can be leveraged for predictive modeling, such as forecasting sudden spikes in temperature or humidity using BigQuery ML. Predictive models could be employed for:
    - **Anomaly Detection:** Identifying irregular fluctuations in environmental conditions.
    - **Trend Analysis:** Identifying seasonal variations or long-term environmental changes.
    - **Predictive Maintenance:** Estimating the likelihood of sensor malfunctions based on historical data.

### 3. Alerting and Notification System (Component #3)

- **Google Cloud Functions:** Acts as an event-driven, serverless function triggered by data received from the Pub/Sub topic. When temperature or humidity values exceed predefined thresholds, Cloud Functions send alert emails to designated stakeholders. Multiple recipients can be notified, and alerting thresholds are configurable to accommodate various warehouse zones or storage conditions.

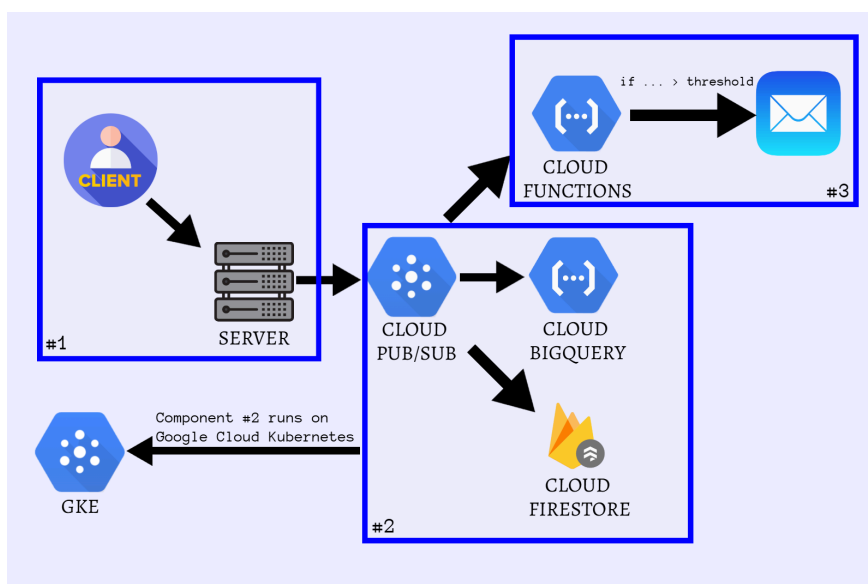


The screenshot shows the Google Cloud console interface for a function named 'notifyifexceeded' in the 'us-central1' region. The function's source code is displayed, showing a Python script that uses the smtplib library to send emails. The code includes a Flask app with a POST endpoint that triggers the notification function. The function checks if the request data contains 'message' and 'data' fields and returns a 400 status code if they are missing.



Senzorul sensor\_1 a înregistrat valori ridicate:

- Temperatură: 24.91°C (prag: 30.0°C)
- Umiditate: 82.2% (prag: 80.0%)



## Analysis of System Characteristics

- **Performance:** The WebSocket protocol facilitates real-time data transmission, minimizing delays. GKE's horizontal scaling ensures that data processing remains efficient under high loads. Based on the provided metrics:
  - **Firestore Request Latencies (P50 and P99):** The P50 latency for batch get documents, commits, list operations, and lookups is under 0.1s, indicating efficient handling of common requests. However, occasional spikes in P99 latency up to 0.5s suggest potential performance bottlenecks during peak loads.

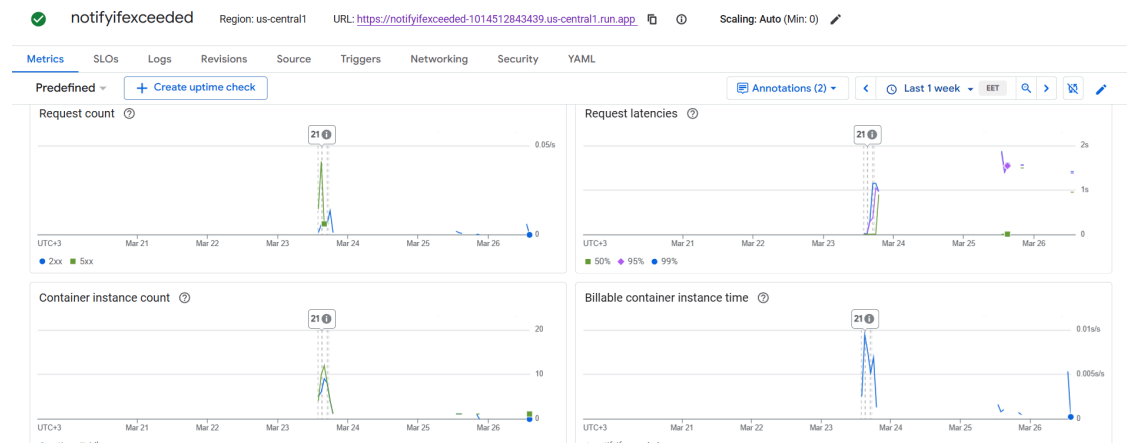


- **Bytes Count in Kubernetes:** A significant initial spike followed by stable values indicates efficient scaling and stabilization.



- **Latency:** The combined WebSocket and Pub/Sub approach generally ensures prompt data propagation.
  - **Pub/Sub Metrics:** The delivery latency health score of 80.22% indicates room for improvement in delivery efficiency. Publish request latencies (P50: 454.135 ms, P95: 1.514 s, P99: 1.714 s) suggest relatively higher delays under heavy traffic, necessitating optimizations. Acknowledgment latencies (P50: 12.34 ms, P95: 48.046 ms, P99: 54.39 ms) are within acceptable bounds, showcasing efficient message acknowledgment.

- **Cloud Function Metrics:**



- 
- **WebSocket Latency Testing:** The latency measurements are excellent, consistently around 1.6ms, indicating minimal delay in message transmission and processing. The temperature and humidity readings are stable, showing that the system handles data efficiently without significant fluctuations.

The image shows a PyCharm IDE with a project named 'homework2'. The code editor displays the following Python code:

```

server.py
44 async def websocket_handler(websocket):
45     await websocket.send(json.dumps(resp))
46 except Exception as e:
47     print(f"Error: {e}")
48
49
50 client.py
51 async def main():
52     server = await websockets.serve(websocket_handler, 'localhost', 8765)
53     print("WebSocket Server running on ws://localhost:8765")
54     await server.wait_closed()
55
56 if __name__ == "__main__":
57     asyncio.run(main())

```

The Run console shows the following output:

```

WebSocket Server running on ws://localhost:8765
Received data from sensor_1: Temp=23.37°C, Hum=78.3%, Latency=0.0011s
Received data from sensor_1: Temp=24.91°C, Hum=82.2%, Latency=0.0006s
Received data from sensor_1: Temp=28.41°C, Hum=72.17%, Latency=0.0006s
Received data from sensor_1: Temp=29.17°C, Hum=68.43%, Latency=0.0006s

```

- **Reliability:** Data redundancy in Firestore and BigQuery minimizes risks of data loss.
  - There are no data loss or constant errors in Firestore and Bigquery. The only errors are related to mistakes in the implementation of the serverless function.

Error Groups

Open, Acknowledged

All Resources

global

Configure notification channels to have more control over who is notified when errors occur.

Configure Notifications

Dismiss

Filter errors

1 hour

6 hours

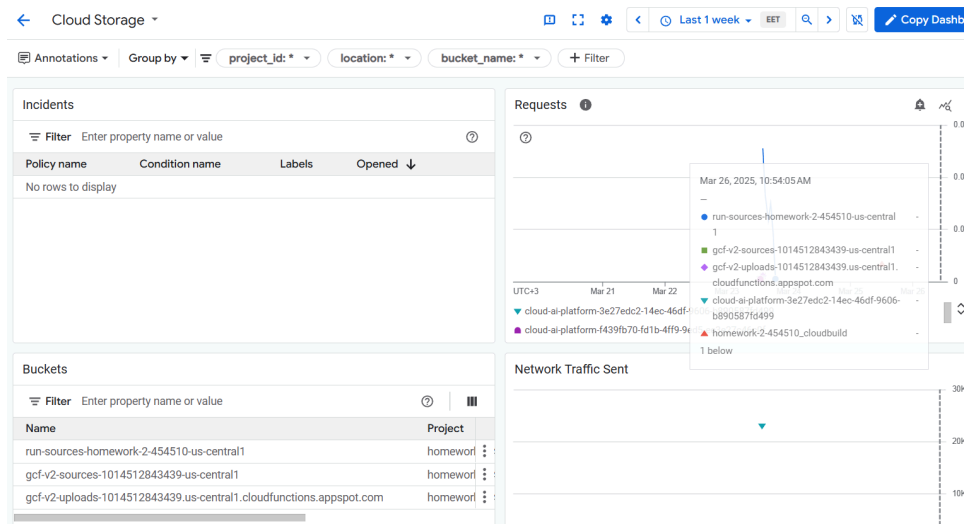
1 day

7 days

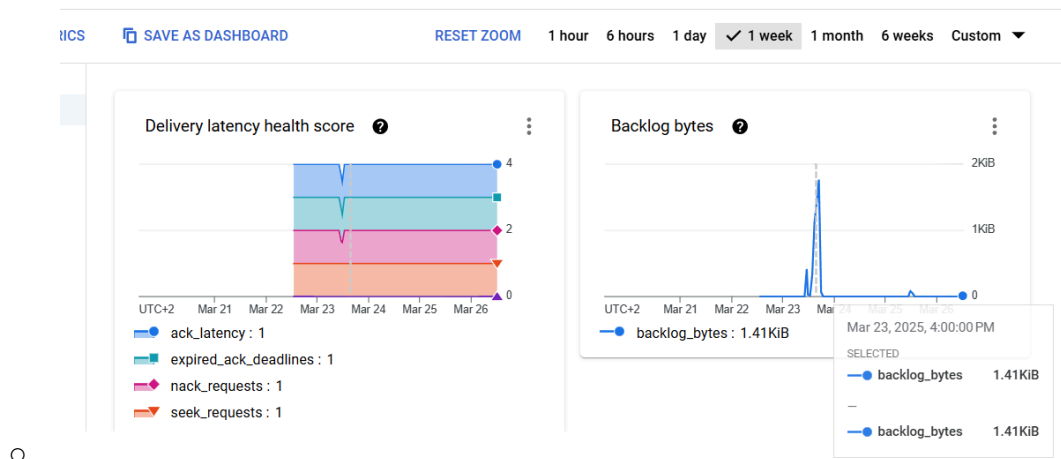
30 days

| Resolution Status | Occurrences                          | Error                                                                                                                                                                                 | Seen In                                                             | Type                         | First Seen                                  | Last Seen                                   |
|-------------------|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------|------------------------------|---------------------------------------------|---------------------------------------------|
| <div>Open</div>   | <div><div></div><div>199</div></div> | <div>TypeError: notify_if_exceeded() takes 0 positional arguments but 1 was given</div> <div>142 /layers/google.python.pip/pip/lib/python3.10/site-packages/functions_framework</div> | <div>notifyifexceeded:</div> <div>notify_...d-00002-hr54 more</div> | <div>Application error</div> | <div>3 days ago</div> <div>3 days ago</div> | <div>3 days ago</div> <div>3 days ago</div> |
| <div>Open</div>   | <div><div></div><div>1</div></div>   | <div>ImportError: cannot import name 'MIMEText' from 'email.mime.text' (/layers/google.py</div> <div>5 /workspace/main.py&lt;module&gt;)</div>                                        | <div>notifyifexceeded:</div> <div>notify_...d-00020-c7l</div>       | <div>Application error</div> | <div>3 days ago</div> <div>3 days ago</div> | <div>3 days ago</div> <div>3 days ago</div> |
| <div>Open</div>   | <div><div></div><div>1</div></div>   | <div>KeyError: 'EMAIL_SENDER'</div> <div>680 /layers/google.python.runtime/python/lib/python3.10/os.py(...getitem_)</div>                                                             | <div>notifyifexceeded:</div> <div>notify_...d-00002-kks</div>       | <div>Application error</div> | <div>3 days ago</div> <div>3 days ago</div> | <div>3 days ago</div> <div>3 days ago</div> |

- 
- At the time of writing this report, the BigQuery dataset contained 104 elements, while Firestore recorded the same number of writes, indicating data consistency.
- After checking Google Cloud Monitoring, we observed that no incidents were recorded for Firestore or BigQuery. Additionally, the system maintained stable performance, with no unexpected errors or delays. Resource utilization remained within normal limits, further confirming the reliability of the infrastructure.



- **Scalability:**
  - The data suggests that the system is handling message delivery effectively, but there are some temporary spikes in backlog bytes. The `ack_latency`, `expired_ack_deadlines`, and `nack_requests` indicate occasional delays in message acknowledgment, which might impact scalability under higher loads. However, since the backlog quickly returns to zero, it suggests that the system can recover efficiently. To ensure long-term scalability, it would be beneficial to monitor these metrics over time and optimize acknowledgment handling if needed.



## Advanced Use Cases and Future Enhancements

- **Predictive Analytics:** Utilizing BigQuery ML for deeper insights into environmental data, like identifying the optimal storage conditions for specific products.
- **Integration with IoT Dashboards:** Developing real-time dashboards for visual monitoring and alerting.
- **Edge Computing Integration:** Implementing edge devices to preprocess data, reducing cloud dependency and latency further.
- **Multi-region Deployment:** Ensuring disaster recovery and minimizing downtime by deploying in multiple regions.