KU LEUVEN

Faculty of Economics & Business

Edge Computing

Version 1.0

by

Pauline De Ryck, Vince Coppens

Professor F. Put

This paper investigates the concept of *Edge Computing*, exploring its architecture, relevance, existing standards and open-source implementations, and evaluating its role within modern distributed systems through literature research and practical experimentation.

Academic year 2025–2026
First Master of Business and Information Systems Engineering
Catholic University Leuven

Contents

List of Figures

Chapter 1

Implementation of an *edge–cloud* architecture: experimental setup and technologies

1.1 Experiment overview and architecture

The goal of this experiment is to design and implement a functional prototype that demonstrates how an edge–cloud architecture can combine local intelligence with centralized cloud coordination. The prototype illustrates how modern container and orchestration technologies can be used to process data close to its source while maintaining integration with cloud services. At least, that was the theoretical goal; as will be shown later, the actual implementation differs slightly from this plan.

The setup follows a three-layer model that mirrors a typical edge-to-cloud continuum. At the device layer, a Raspberry Pi acts as an IoT endpoint that generates and publishes sensor data. It represents the multitude of field devices producing information near the physical environment.

The edge cloud layer, hosted on a MacBook, provides the intermediate computing environment where data is processed before being sent to the cloud. Inside this layer, lightweight containers run within a K3D Kubernetes cluster, forming a small-scale edge cloud capable of orchestrating local services.

The cloud layer, implemented through AWS IoT Core, serves as the data endpoint. Communication between these layers is handled via the MQTT protocol (see Section 2.2.1).

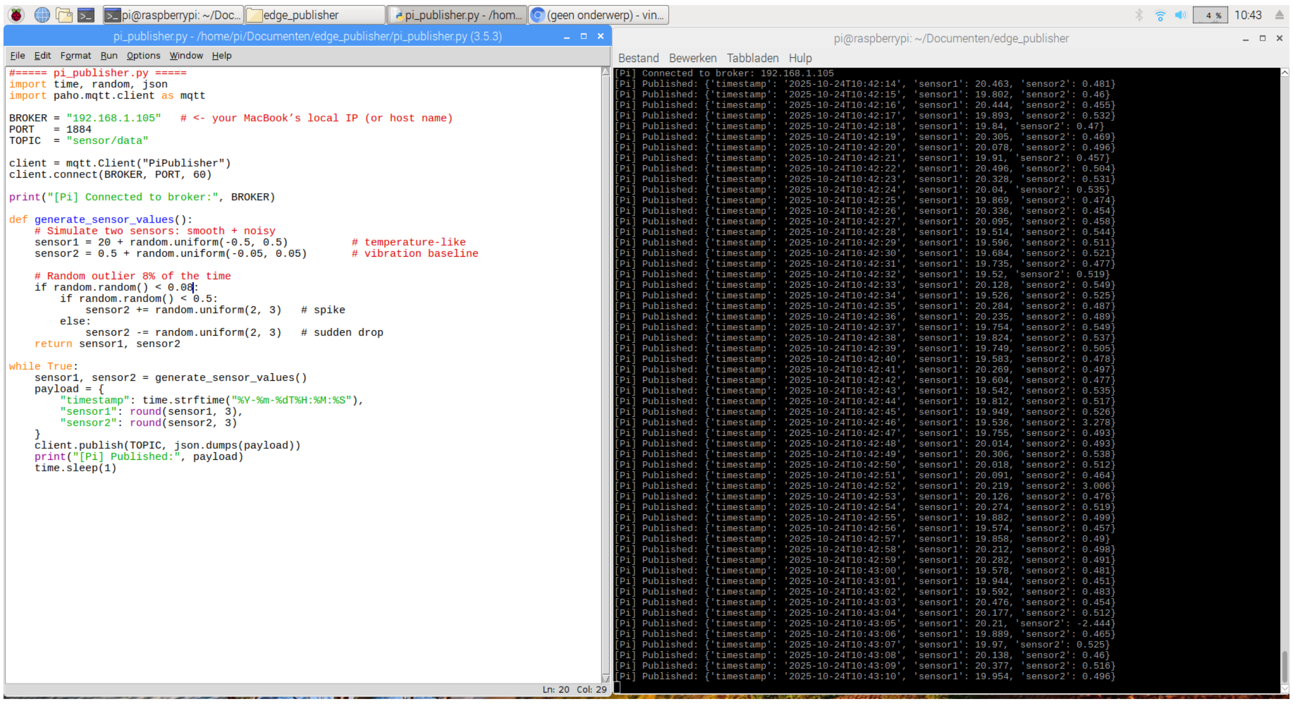
A complete architectural visualisation of the experiment is provided in Appendix D. In the following sections, each layer and its components are discussed in detail.

1.2 Device layer — Raspberry Pi

The device layer represents the physical interface between the digital architecture and its surrounding environment. In this experiment, it is implemented by a single Raspberry Pi running Raspbian Stretch (2017), which acts as a simulated IoT endpoint. The device continuously generates and publishes sensor data to the edge cloud layer using the MQTT protocol[1] following the principles introduced in Section 2.2.1.

The Python publisher emulates two physical sensors:

[1]OASIS MQTT Technical Committee, "MQTT — the standard for IoT messaging." `https://mqtt.org`. Accessed: 9 Nov. 2025; see also Amazon Web Services, "What is MQTT?." `https://aws.amazon.com/what-is/mqtt/`. Accessed: 9 Nov. 2025.


pi@raspberrypi: ~/Doc...
edge_publisher
pi_publisher.py - /hom...
(geen onderwerp) - vin...
10:43
pi_publisher.py - /home/pi/Documenten/edge_publisher/pi_publisher.py (3.5.3)
File Edit Format Run Options Window Help
Ln: 20 Col: 29
pi@raspberrypi: ~/Documenten/edge_publisher
Bestand Bewerken Tabbladen Hulp

```python
#===== pi_publisher.py =====
import time, random, json
import paho.mqtt.client as mqtt

BROKER = "192.168.1.105"   # <- your MacBook’s local IP (or host name)
PORT   = 1884
TOPIC  = "sensor/data"

client = mqtt.Client("PiPublisher")
client.connect(BROKER, PORT, 60)

print("[Pi] Connected to broker:", BROKER)

def generate_sensor_values():
    # Simulate two sensors: smooth + noisy
    sensor1 = 20 + random.uniform(-0.5, 0.5)          # temperature-like
    sensor2 = 0.5 + random.uniform(-0.05, 0.05)       # vibration baseline

    # Random outlier 8% of the time
    if random.random() < 0.08:
        if random.random() < 0.5:
            sensor2 += random.uniform(2, 3)   # spike
        else:
            sensor2 -= random.uniform(2, 3)   # sudden drop
    return sensor1, sensor2

while True:
    sensor1, sensor2 = generate_sensor_values()
    payload = {
        "timestamp": time.strftime("%Y-%m-%dT%H:%M:%S"),
        "sensor1": round(sensor1, 3),
        "sensor2": round(sensor2, 3)
    }
    client.publish(TOPIC, json.dumps(payload))
    print("[Pi] Published:", payload)
    time.sleep(1)
```

```
[Pi] Connected to broker: 192.168.1.105
[Pi] Published: {'timestamp': '2025-10-24T10:42:14', 'sensor1': 20.463, 'sensor2': 0.481}
[Pi] Published: {'timestamp': '2025-10-24T10:42:15', 'sensor1': 19.802, 'sensor2': 0.46}
[Pi] Published: {'timestamp': '2025-10-24T10:42:16', 'sensor1': 20.444, 'sensor2': 0.455}
[Pi] Published: {'timestamp': '2025-10-24T10:42:17', 'sensor1': 19.893, 'sensor2': 0.532}
[Pi] Published: {'timestamp': '2025-10-24T10:42:18', 'sensor1': 19.84, 'sensor2': 0.47}
[Pi] Published: {'timestamp': '2025-10-24T10:42:19', 'sensor1': 20.305, 'sensor2': 0.469}
[Pi] Published: {'timestamp': '2025-10-24T10:42:20', 'sensor1': 20.078, 'sensor2': 0.496}
[Pi] Published: {'timestamp': '2025-10-24T10:42:21', 'sensor1': 19.91, 'sensor2': 0.457}
[Pi] Published: {'timestamp': '2025-10-24T10:42:22', 'sensor1': 20.496, 'sensor2': 0.504}
[Pi] Published: {'timestamp': '2025-10-24T10:42:23', 'sensor1': 20.328, 'sensor2': 0.531}
[Pi] Published: {'timestamp': '2025-10-24T10:42:24', 'sensor1': 20.04, 'sensor2': 0.535}
[Pi] Published: {'timestamp': '2025-10-24T10:42:25', 'sensor1': 19.869, 'sensor2': 0.474}
[Pi] Published: {'timestamp': '2025-10-24T10:42:26', 'sensor1': 20.336, 'sensor2': 0.454}
[Pi] Published: {'timestamp': '2025-10-24T10:42:27', 'sensor1': 20.095, 'sensor2': 0.458}
[Pi] Published: {'timestamp': '2025-10-24T10:42:28', 'sensor1': 19.514, 'sensor2': 0.544}
[Pi] Published: {'timestamp': '2025-10-24T10:42:29', 'sensor1': 19.596, 'sensor2': 0.511}
[Pi] Published: {'timestamp': '2025-10-24T10:42:30', 'sensor1': 19.684, 'sensor2': 0.521}
[Pi] Published: {'timestamp': '2025-10-24T10:42:31', 'sensor1': 19.735, 'sensor2': 0.477}
[Pi] Published: {'timestamp': '2025-10-24T10:42:32', 'sensor1': 19.52, 'sensor2': 0.519}
[Pi] Published: {'timestamp': '2025-10-24T10:42:33', 'sensor1': 20.128, 'sensor2': 0.549}
[Pi] Published: {'timestamp': '2025-10-24T10:42:34', 'sensor1': 19.526, 'sensor2': 0.525}
[Pi] Published: {'timestamp': '2025-10-24T10:42:35', 'sensor1': 20.284, 'sensor2': 0.487}
[Pi] Published: {'timestamp': '2025-10-24T10:42:36', 'sensor1': 20.235, 'sensor2': 0.489}
[Pi] Published: {'timestamp': '2025-10-24T10:42:37', 'sensor1': 19.754, 'sensor2': 0.549}
[Pi] Published: {'timestamp': '2025-10-24T10:42:38', 'sensor1': 19.824, 'sensor2': 0.537}
[Pi] Published: {'timestamp': '2025-10-24T10:42:39', 'sensor1': 19.749, 'sensor2': 0.505}
[Pi] Published: {'timestamp': '2025-10-24T10:42:40', 'sensor1': 19.583, 'sensor2': 0.478}
[Pi] Published: {'timestamp': '2025-10-24T10:42:41', 'sensor1': 20.269, 'sensor2': 0.497}
[Pi] Published: {'timestamp': '2025-10-24T10:42:42', 'sensor1': 19.604, 'sensor2': 0.477}
[Pi] Published: {'timestamp': '2025-10-24T10:42:43', 'sensor1': 19.542, 'sensor2': 0.535}
[Pi] Published: {'timestamp': '2025-10-24T10:42:44', 'sensor1': 19.812, 'sensor2': 0.517}
[Pi] Published: {'timestamp': '2025-10-24T10:42:45', 'sensor1': 19.949, 'sensor2': 0.526}
[Pi] Published: {'timestamp': '2025-10-24T10:42:46', 'sensor1': 19.536, 'sensor2': 3.278}
[Pi] Published: {'timestamp': '2025-10-24T10:42:47', 'sensor1': 19.755, 'sensor2': 0.493}
[Pi] Published: {'timestamp': '2025-10-24T10:42:48', 'sensor1': 20.014, 'sensor2': 0.493}
[Pi] Published: {'timestamp': '2025-10-24T10:42:49', 'sensor1': 20.306, 'sensor2': 0.538}
[Pi] Published: {'timestamp': '2025-10-24T10:42:50', 'sensor1': 20.018, 'sensor2': 0.512}
[Pi] Published: {'timestamp': '2025-10-24T10:42:51', 'sensor1': 20.091, 'sensor2': 0.464}
[Pi] Published: {'timestamp': '2025-10-24T10:42:52', 'sensor1': 20.219, 'sensor2': 3.006}
[Pi] Published: {'timestamp': '2025-10-24T10:42:53', 'sensor1': 20.126, 'sensor2': 0.476}
[Pi] Published: {'timestamp': '2025-10-24T10:42:54', 'sensor1': 20.274, 'sensor2': 0.519}
[Pi] Published: {'timestamp': '2025-10-24T10:42:55', 'sensor1': 19.882, 'sensor2': 0.499}
[Pi] Published: {'timestamp': '2025-10-24T10:42:56', 'sensor1': 19.574, 'sensor2': 0.457}
[Pi] Published: {'timestamp': '2025-10-24T10:42:57', 'sensor1': 19.858, 'sensor2': 0.49}
[Pi] Published: {'timestamp': '2025-10-24T10:42:58', 'sensor1': 20.212, 'sensor2': 0.498}
[Pi] Published: {'timestamp': '2025-10-24T10:42:59', 'sensor1': 20.282, 'sensor2': 0.491}
[Pi] Published: {'timestamp': '2025-10-24T10:43:00', 'sensor1': 19.578, 'sensor2': 0.481}
[Pi] Published: {'timestamp': '2025-10-24T10:43:01', 'sensor1': 19.944, 'sensor2': 0.451}
[Pi] Published: {'timestamp': '2025-10-24T10:43:02', 'sensor1': 19.592, 'sensor2': 0.483}
[Pi] Published: {'timestamp': '2025-10-24T10:43:03', 'sensor1': 20.476, 'sensor2': 0.454}
[Pi] Published: {'timestamp': '2025-10-24T10:43:04', 'sensor1': 20.177, 'sensor2': 0.512}
[Pi] Published: {'timestamp': '2025-10-24T10:43:05', 'sensor1': 20.21, 'sensor2': -2.444}
[Pi] Published: {'timestamp': '2025-10-24T10:43:06', 'sensor1': 19.889, 'sensor2': 0.465}
[Pi] Published: {'timestamp': '2025-10-24T10:43:07', 'sensor1': 19.97, 'sensor2': 0.525}
[Pi] Published: {'timestamp': '2025-10-24T10:43:08', 'sensor1': 20.138, 'sensor2': 0.46}
[Pi] Published: {'timestamp': '2025-10-24T10:43:09', 'sensor1': 20.377, 'sensor2': 0.516}
[Pi] Published: {'timestamp': '2025-10-24T10:43:10', 'sensor1': 19.954, 'sensor2': 0.496}
```

Figure 1.1: Python script `pi_publisher.py` running on the Raspberry Pi and broadcasting live JSON-formatted sensor data on the MQTT network.

- **Sensor 1** — temperature-like readings fluctuating around 20 °C.
- **Sensor 2** — vibration-like readings with a baseline of approximately 0.5 g.

To simulate irregularities typically found in industrial or environmental signals, around 8 % of the values are replaced by random spikes or drops. Each message, containing both sensor values and a timestamp, is encoded as JSON and published once per second to the MQTT topic `sensor/data` on port 1884 of the broker running in the edge-cloud environment.

Although the initial plan was to analyse both simulated sensors, the final experiment used only the vibration signal (sensor 2) as input for the anomaly detection model on the edge cloud. This adjustment does not diminish the experiment's purpose, as the single-sensor setup still demonstrates the complete functionality of the edge–cloud pipeline — from data generation and local preprocessing to cloud transmission and monitoring.

The Raspberry Pi's function is therefore limited to data publication. It does not host a KubeEdge EdgeCore component, because its operating-system version lacked compatibility with recent releases.[2] Nevertheless, the architecture remains KubeEdge-ready for future extensions in which the Pi — or other IoT devices — could be managed directly by the local Kubernetes cluster through KubeEdge.

This setup, its configuration, and the overall experimental workflow were iteratively refined with the support of generative AI tools,[3] including ChatGPT and Gemini, which provided interactive feedback and troubleshooting assistance during implementation.

[2] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, "Extend cloud to edge with kubeedge," in *ACM/IEEE Symposium on Edge Computing*, 2018. Accessed: 9 Nov. 2025.

[3] OpenAI and Google, "Generative AI (chatgpt and gemini) — interactive support in experiment design." `https://openai.com`, 2025. Accessed: 9 Nov. 2025.

1.3 Edge cloud layer — MacBook (Docker, Kubernetes, K3D, KubeEdge, and anomaly detection)

The edge cloud layer, hosted on a MacBook running macOS, forms the computational centre of the prototype. It combines local orchestration and lightweight artificial intelligence within a small-scale container infrastructure. This layer demonstrates how several cloud-native technologies — Docker Desktop,[4] K3D,[5] K3s,[6] Kubernetes,[7] and KubeEdge[8] — can work together to manage and execute applications directly at the network edge.

Docker Desktop acts as the foundation of the edge cloud. Because macOS cannot run Linux containers natively, Docker first creates a small Linux virtual machine that provides the environment required for containers to operate.[9] Inside this environment, K3D runs a compact version of Kubernetes (called K3s) entirely within Docker containers. In other words, K3D uses Docker to "host" a Kubernetes cluster locally on the laptop. Within this cluster, one container acts as the control plane — the central brain of Kubernetes, which decides what should run where — and another acts as the worker node, the environment in which the actual applications are executed. On top of this layer, Kubernetes performs the orchestration tasks: it deploys containers, monitors their health, restarts them if they fail, and manages internal networking and configuration. Finally, KubeEdge extends the Kubernetes model to the physical edge by allowing real devices, such as the Raspberry Pi, to connect as managed nodes.

In this experiment, only the CloudCore part of KubeEdge was installed on the MacBook, ensuring compatibility with future extensions, while the Pi did not run the EdgeCore agent due to OS limitations. Together, these components form a layered hierarchy: macOS provides the host system, Docker runs Linux containers, K3D builds the local Kubernetes cluster inside Docker, Kubernetes orchestrates everything, and KubeEdge acts as a bridge to potential edge devices.

The first part of the output illustrates the process of building the Docker image from the local Dockerfile, followed by its import into the K3D cluster (`k3d image import edgecloud-subscriber:latest -c edgecloud`). The subsequent lines confirm that the image is distributed across both the Kubernetes server and agent nodes. After that, the command `kubectl apply -f subscriber-deployment.yaml` instructs Kubernetes to start the subscriber pod defined in the YAML configuration. Once the pod is running, the log output shows continuous MQTT messages being received and processed. Each line represents one JSON message received from the Raspberry Pi through the local MQTT broker and analysed for anomalies.

All source files of this layer — the `mac_subscriber.py` script, `Dockerfile`, and `subscriber-deployment.yaml` — are provided in Appendices A and B. The Dockerfile defines how the container image is built (base image, dependencies, copy of the script), while the YAML file specifies how Kubernetes should deploy the image, set environment variables (e.g. broker IP, port, MQTT topic, and AWS endpoint), and automatically restart the pod if needed.

The `mac_subscriber.py` script executes two main tasks within the container: (1) MQTT data ingestion[10] and (2) anomaly detection using an Isolation Forest model[11] from scikit-learn. The script subscribes to the topic `sensor/data`, extracts the vibration signal (`sensor2`) from each message, and buffers the latest 100 samples in memory. After every 20 new samples, the Isolation Forest is retrained on the current buffer to adapt to

[4]Docker, Inc., "Docker desktop documentation." `https://docs.docker.com/desktop/`, 2025. Accessed: 9 Nov. 2025.

[5]K3D Project, "k3d — lightweight wrapper to run k3s in docker." `https://k3d.io`. Accessed: 9 Nov. 2025.

[6]D. Yakubov and D. Hästbacka, "Comparative analysis of lightweight kubernetes distributions for edge computing: Performance and resource efficiency," *arXiv preprint*, 2025. Accessed: 9 Nov. 2025.

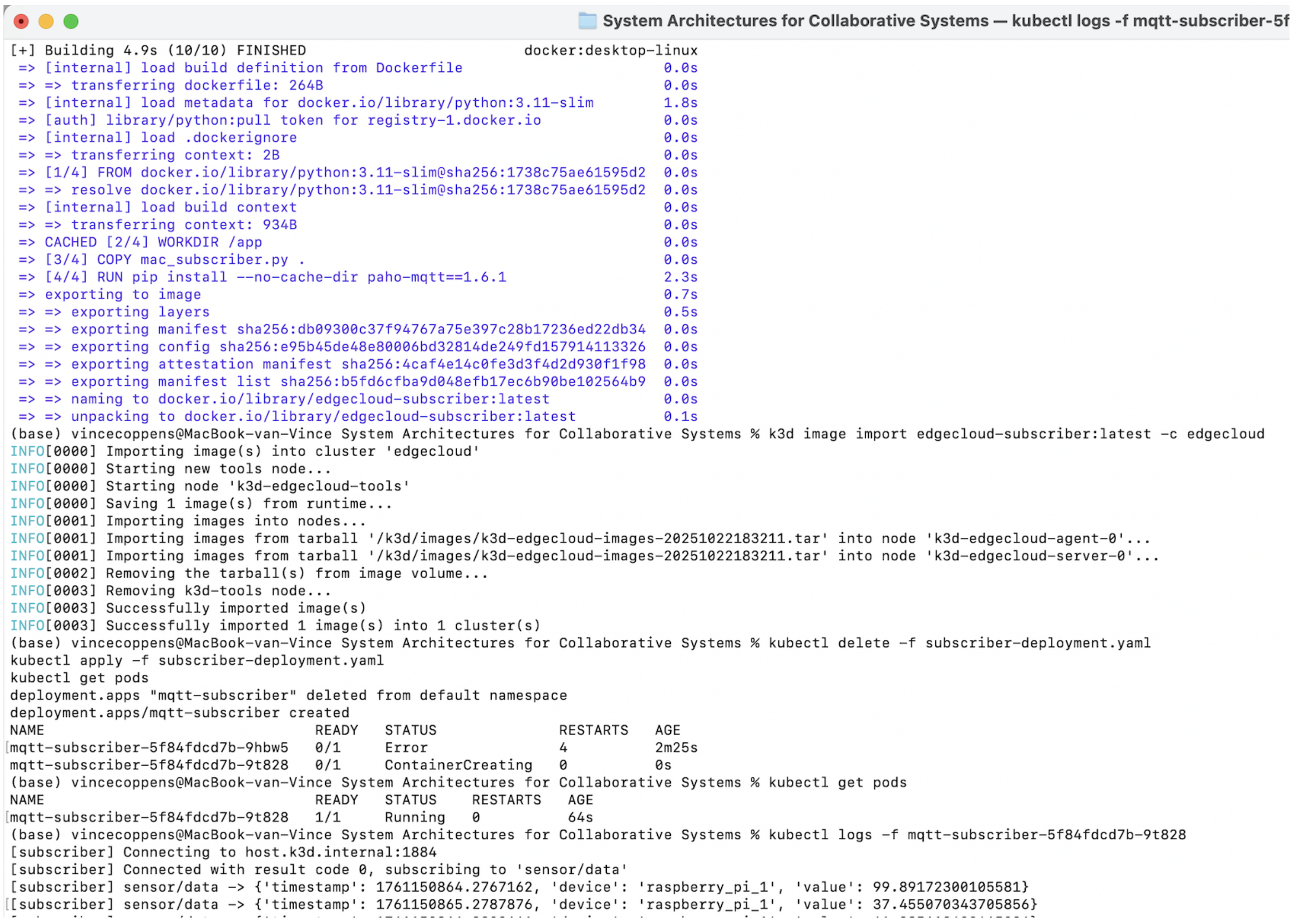
[7]DataCamp, "What is kubernetes? an introduction with examples." `https://www.datacamp.com/blog/what-is-kubernetes`, 2024. Accessed: 9 Nov. 2025; OpsRamp, "An overview of kubernetes architecture." `https://www.opsramp.com/guides/why-kubernetes/kubernetes-architecture/`. Accessed: 9 Nov. 2025; Theodo Cloud, "Kubernetes overview: All you need to know." `https://cloud.theodo.com/en/blog/kubernetes-overview`. Accessed: 9 Nov. 2025.

[8]Y. Xiong, Y. Sun, L. Xing, and Y. Huang, "Extend cloud to edge with kubeedge," in *ACM/IEEE Symposium on Edge Computing*, 2018. Accessed: 9 Nov. 2025.

[9]Collabnix, "How docker desktop works under the hood." `https://collabnix.com/how-docker-desktop-for-windows-works-under-the-hood/`, 2023. Accessed: 9 Nov. 2025.

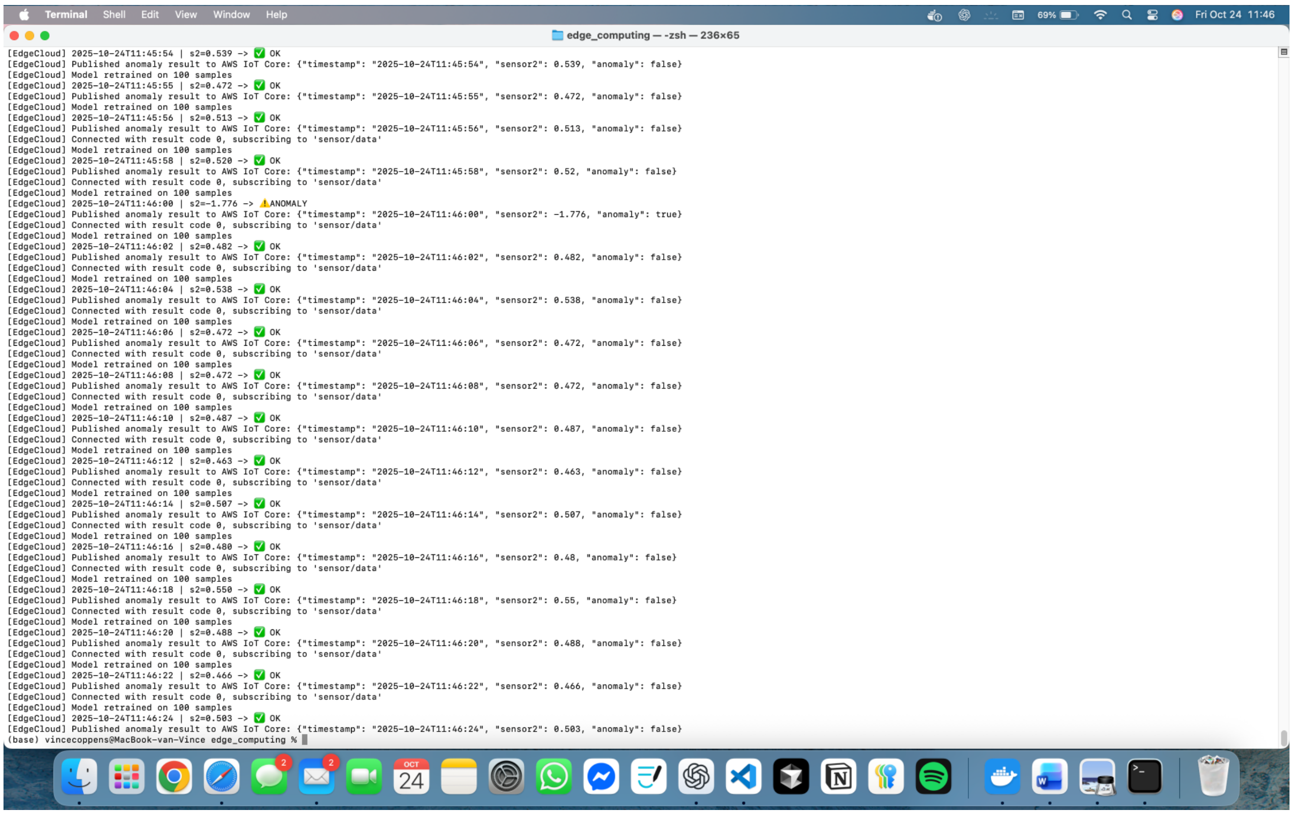
[10]OASIS MQTT Technical Committee, "MQTT — the standard for IoT messaging." `https://mqtt.org`. Accessed: 9 Nov. 2025.

[11]scikit-learn Developers, "Isolationforest — scikit-learn documentation." `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html`. Accessed: 9 Nov. 2025.


System Architectures for Collaborative Systems — kubectl logs -f mqtt-subscriber-5f

```
[+] Building 4.9s (10/10) FINISHED                         docker:desktop-linux
 => [internal] load build definition from Dockerfile                       0.0s
 => => transferring dockerfile: 264B                                       0.0s
 => [internal] load metadata for docker.io/library/python:3.11-slim        1.8s
 => [auth] library/python:pull token for registry-1.docker.io              0.0s
 => [internal] load .dockerignore                                          0.0s
 => => transferring context: 2B                                            0.0s
 => [1/4] FROM docker.io/library/python:3.11-slim@sha256:1738c75ae61595d2  0.0s
 => => resolve docker.io/library/python:3.11-slim@sha256:1738c75ae61595d2  0.0s
 => [internal] load build context                                          0.0s
 => => transferring context: 934B                                          0.0s
 => CACHED [2/4] WORKDIR /app                                              0.0s
 => [3/4] COPY mac_subscriber.py .                                         0.0s
 => [4/4] RUN pip install --no-cache-dir paho-mqtt==1.6.1                  2.3s
 => exporting to image                                                     0.7s
 => => exporting layers                                                    0.5s
 => => exporting manifest sha256:db09300c37f94767a75e397c28b17236ed22db34  0.0s
 => => exporting config sha256:e95b45de48e80006bd32814de249fd157914113326  0.0s
 => => exporting attestation manifest sha256:4caf4e14c0fe3d3f4d2d930f1f98  0.0s
 => => exporting manifest list sha256:b5fd6cfba9d048efb17ec6b90be102564b9  0.0s
 => => naming to docker.io/library/edgecloud-subscriber:latest             0.0s
 => => unpacking to docker.io/library/edgecloud-subscriber:latest          0.1s
(base) vincecoppens@MacBook-van-Vince System Architectures for Collaborative Systems % k3d image import edgecloud-subscriber:latest -c edgecloud
INFO[0000] Importing image(s) into cluster 'edgecloud'
INFO[0000] Starting new tools node...
INFO[0000] Starting node 'k3d-edgecloud-tools'
INFO[0000] Saving 1 image(s) from runtime...
INFO[0001] Importing images into nodes...
INFO[0001] Importing images from tarball '/k3d/images/k3d-edgecloud-images-20251022183211.tar' into node 'k3d-edgecloud-agent-0'...
INFO[0001] Importing images from tarball '/k3d/images/k3d-edgecloud-images-20251022183211.tar' into node 'k3d-edgecloud-server-0'...
INFO[0002] Removing the tarball(s) from image volume...
INFO[0003] Removing k3d-tools node...
INFO[0003] Successfully imported image(s)
INFO[0003] Successfully imported 1 image(s) into 1 cluster(s)
(base) vincecoppens@MacBook-van-Vince System Architectures for Collaborative Systems % kubectl delete -f subscriber-deployment.yaml
kubectl apply -f subscriber-deployment.yaml
kubectl get pods
deployment.apps "mqtt-subscriber" deleted from default namespace
deployment.apps/mqtt-subscriber created
NAME                                READY   STATUS              RESTARTS   AGE
mqtt-subscriber-5f84fdcd7b-9hbw5    0/1     Error               4          2m25s
mqtt-subscriber-5f84fdcd7b-9t828    0/1     ContainerCreating   0          0s
(base) vincecoppens@MacBook-van-Vince System Architectures for Collaborative Systems % kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
mqtt-subscriber-5f84fdcd7b-9t828    1/1     Running   0          64s
(base) vincecoppens@MacBook-van-Vince System Architectures for Collaborative Systems % kubectl logs -f mqtt-subscriber-5f84fdcd7b-9t828
[subscriber] Connecting to host.k3d.internal:1884
[subscriber] Connected with result code 0, subscribing to 'sensor/data'
[subscriber] sensor/data -> {'timestamp': 1761150864.2767162, 'device': 'raspberry_pi_1', 'value': 99.89172300105581}
[subscriber] sensor/data -> {'timestamp': 1761150865.2787876, 'device': 'raspberry_pi_1', 'value': 37.455070343705856}
```

Figure 1.2: MacBook terminal during cluster setup and deployment of the subscriber container. The procedure illustrated here corresponds to the steps described in Appendix C (Build and run instructions).



```
[EdgeCloud] 2025-10-24T11:45:54 | s2=0.539 -> ✅ OK
[EdgeCloud] Published anomaly result to AWS IoT Core: {"timestamp": "2025-10-24T11:45:54", "sensor2": 0.539, "anomaly": false}
[EdgeCloud] Model retrained on 100 samples
[EdgeCloud] 2025-10-24T11:45:55 | s2=0.472 -> ✅ OK
[EdgeCloud] Published anomaly result to AWS IoT Core: {"timestamp": "2025-10-24T11:45:55", "sensor2": 0.472, "anomaly": false}
[EdgeCloud] Model retrained on 100 samples
[EdgeCloud] 2025-10-24T11:45:56 | s2=0.513 -> ✅ OK
[EdgeCloud] Published anomaly result to AWS IoT Core: {"timestamp": "2025-10-24T11:45:56", "sensor2": 0.513, "anomaly": false}
[EdgeCloud] Connected with result code 0, subscribing to 'sensor/data'
[EdgeCloud] Model retrained on 100 samples
[EdgeCloud] 2025-10-24T11:45:58 | s2=0.520 -> ✅ OK
[EdgeCloud] Published anomaly result to AWS IoT Core: {"timestamp": "2025-10-24T11:45:58", "sensor2": 0.52, "anomaly": false}
[EdgeCloud] Connected with result code 0, subscribing to 'sensor/data'
[EdgeCloud] Model retrained on 100 samples
[EdgeCloud] 2025-10-24T11:46:00 | s2=-1.776 -> ⚠️ANOMALY
[EdgeCloud] Published anomaly result to AWS IoT Core: {"timestamp": "2025-10-24T11:46:00", "sensor2": -1.776, "anomaly": true}
[EdgeCloud] Connected with result code 0, subscribing to 'sensor/data'
[EdgeCloud] Model retrained on 100 samples
[EdgeCloud] 2025-10-24T11:46:02 | s2=0.482 -> ✅ OK
[EdgeCloud] Published anomaly result to AWS IoT Core: {"timestamp": "2025-10-24T11:46:02", "sensor2": 0.482, "anomaly": false}
[EdgeCloud] Connected with result code 0, subscribing to 'sensor/data'
[EdgeCloud] Model retrained on 100 samples
[EdgeCloud] 2025-10-24T11:46:04 | s2=0.538 -> ✅ OK
[EdgeCloud] Published anomaly result to AWS IoT Core: {"timestamp": "2025-10-24T11:46:04", "sensor2": 0.538, "anomaly": false}
[EdgeCloud] Connected with result code 0, subscribing to 'sensor/data'
[EdgeCloud] Model retrained on 100 samples
[EdgeCloud] 2025-10-24T11:46:06 | s2=0.472 -> ✅ OK
[EdgeCloud] Published anomaly result to AWS IoT Core: {"timestamp": "2025-10-24T11:46:06", "sensor2": 0.472, "anomaly": false}
[EdgeCloud] Connected with result code 0, subscribing to 'sensor/data'
[EdgeCloud] Model retrained on 100 samples
[EdgeCloud] 2025-10-24T11:46:08 | s2=0.472 -> ✅ OK
[EdgeCloud] Published anomaly result to AWS IoT Core: {"timestamp": "2025-10-24T11:46:08", "sensor2": 0.472, "anomaly": false}
[EdgeCloud] Connected with result code 0, subscribing to 'sensor/data'
[EdgeCloud] Model retrained on 100 samples
[EdgeCloud] 2025-10-24T11:46:10 | s2=0.487 -> ✅ OK
[EdgeCloud] Published anomaly result to AWS IoT Core: {"timestamp": "2025-10-24T11:46:10", "sensor2": 0.487, "anomaly": false}
[EdgeCloud] Connected with result code 0, subscribing to 'sensor/data'
[EdgeCloud] Model retrained on 100 samples
[EdgeCloud] 2025-10-24T11:46:12 | s2=0.463 -> ✅ OK
[EdgeCloud] Published anomaly result to AWS IoT Core: {"timestamp": "2025-10-24T11:46:12", "sensor2": 0.463, "anomaly": false}
[EdgeCloud] Connected with result code 0, subscribing to 'sensor/data'
[EdgeCloud] Model retrained on 100 samples
[EdgeCloud] 2025-10-24T11:46:14 | s2=0.507 -> ✅ OK
[EdgeCloud] Published anomaly result to AWS IoT Core: {"timestamp": "2025-10-24T11:46:14", "sensor2": 0.507, "anomaly": false}
[EdgeCloud] Connected with result code 0, subscribing to 'sensor/data'
[EdgeCloud] Model retrained on 100 samples
[EdgeCloud] 2025-10-24T11:46:16 | s2=0.480 -> ✅ OK
[EdgeCloud] Published anomaly result to AWS IoT Core: {"timestamp": "2025-10-24T11:46:16", "sensor2": 0.48, "anomaly": false}
[EdgeCloud] Connected with result code 0, subscribing to 'sensor/data'
[EdgeCloud] Model retrained on 100 samples
[EdgeCloud] 2025-10-24T11:46:18 | s2=0.550 -> ✅ OK
[EdgeCloud] Published anomaly result to AWS IoT Core: {"timestamp": "2025-10-24T11:46:18", "sensor2": 0.55, "anomaly": false}
[EdgeCloud] Connected with result code 0, subscribing to 'sensor/data'
[EdgeCloud] Model retrained on 100 samples
[EdgeCloud] 2025-10-24T11:46:20 | s2=0.488 -> ✅ OK
[EdgeCloud] Published anomaly result to AWS IoT Core: {"timestamp": "2025-10-24T11:46:20", "sensor2": 0.488, "anomaly": false}
[EdgeCloud] Connected with result code 0, subscribing to 'sensor/data'
[EdgeCloud] Model retrained on 100 samples
[EdgeCloud] 2025-10-24T11:46:22 | s2=0.466 -> ✅ OK
[EdgeCloud] Published anomaly result to AWS IoT Core: {"timestamp": "2025-10-24T11:46:22", "sensor2": 0.466, "anomaly": false}
[EdgeCloud] Connected with result code 0, subscribing to 'sensor/data'
[EdgeCloud] Model retrained on 100 samples
[EdgeCloud] 2025-10-24T11:46:24 | s2=0.503 -> ✅ OK
[EdgeCloud] Published anomaly result to AWS IoT Core: {"timestamp": "2025-10-24T11:46:24", "sensor2": 0.503, "anomaly": false}
(base) vincecoppens@MacBook-van-Vince edge_computing %
```

Figure 1.3: Real-time output of the running subscriber container on the MacBook. Each line confirms receipt and anomaly classification of one MQTT message before publication to AWS IoT Core.

potential slow drifts in the signal. For each incoming value, the model computes an anomaly score based on how isolated that data point is within a randomly constructed forest of decision trees. If the score exceeds a threshold, the reading is classified as an outlier (`"anomaly": true`); otherwise it is marked as normal. This allows the model to operate in a sliding-window fashion rather than static batches, meaning it continuously learns from the most recent data instead of relying on fixed training sets. The approach is lightweight and suitable for limited computing resources typical at the edge.

Once the classification result is determined, the container republishes the enriched JSON message — including timestamp, `sensor2` value, and anomaly flag — to AWS IoT Core[12] over an encrypted MQTT/TLS connection using the device certificates stored inside the container. This completes the local processing pipeline: raw sensor data are ingested, analysed, and securely transmitted to the cloud within the same edge node.

Originally, the plan was to publish aggregated results to the cloud only periodically, for example after every 100 analysed samples, to further reduce data-transmission frequency and make the pipeline more representative of edge–cloud behaviour. However, due to time constraints during testing and the need to repeatedly rebuild and redeploy the system, this optimisation was not implemented. It could easily be added later with a few extra lines of code in the subscriber loop, without requiring any architectural changes. Although the current implementation sends messages more frequently, it still demonstrates the full interaction between local inference and cloud communication within the designed edge–cloud infrastructure.

[12] Amazon Web Services, "AWS IoT core — what is AWS IoT?." `https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html`. Accessed: 9 Nov. 2025; Amazon Web Services, "How AWS IoT works." `https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html`. Accessed: 9 Nov. 2025; P. Borra, "Analyzing AWS edge computing solutions to enhance IoT deployments," *International Journal of Engineering and Advanced Technology (IJEAT)*, 2024. Accessed: 9 Nov. 2025.

1.4 Cloud layer — AWS IoT Core

The cloud layer completes the edge–cloud architecture by providing a secure and scalable environment for data collection, monitoring, and potential post-processing. In this experiment, the cloud layer is implemented through AWS IoT Core,[13] a managed service that enables IoT devices and edge nodes to communicate reliably with the cloud using the MQTT protocol[14] over TLS encryption.

The setup of this layer began by registering a new IoT device, or *Thing*, named `EdgeCloudAWS` in the AWS IoT Core management console. During registration, AWS automatically generated the required security credentials, including a device certificate, a private key, and the `AmazonRootCA1.pem` file for authentication. These three files were downloaded and mounted into the subscriber container on the MacBook, ensuring that all messages published to the AWS endpoint were transmitted securely using mutual TLS encryption. The subscriber connected to the endpoint `akzdc7a60ugm9-ats.iot.eu-north-1.amazonaws.com` on port 8883, which is the default MQTT port for encrypted communication.

Within AWS IoT Core, messages arriving at the topic `edgecloud/alerts` can be processed further through IoT Rules, which define automated actions such as forwarding messages to AWS Lambda, storing them in Amazon S3, or inserting them into a DynamoDB table.[15] These integrations were conceptually explored during the design phase but were not implemented in the prototype, as the focus of the experiment was to demonstrate the successful connection between the local intelligence at the edge and the cloud service, not long-term data storage or analytics. The system architecture, however, fully supports such extensions without modification—only a few configuration steps in the AWS console would be required to enable them.

Figure 1.4 therefore illustrates the final step of the pipeline: each message published by the `mac_subscriber.py` container is received almost instantly in the AWS IoT Core dashboard, proving the reliability of the MQTT/TLS connection and the validity of the device credentials. The message structure contains the timestamp, the measured vibration value, and the anomaly flag determined by the Isolation Forest model running at the edge. This confirms that inference and preprocessing occur locally, while the cloud remains responsible for central data management and potential integration with other AWS services.

1.5 Experiment conclusion

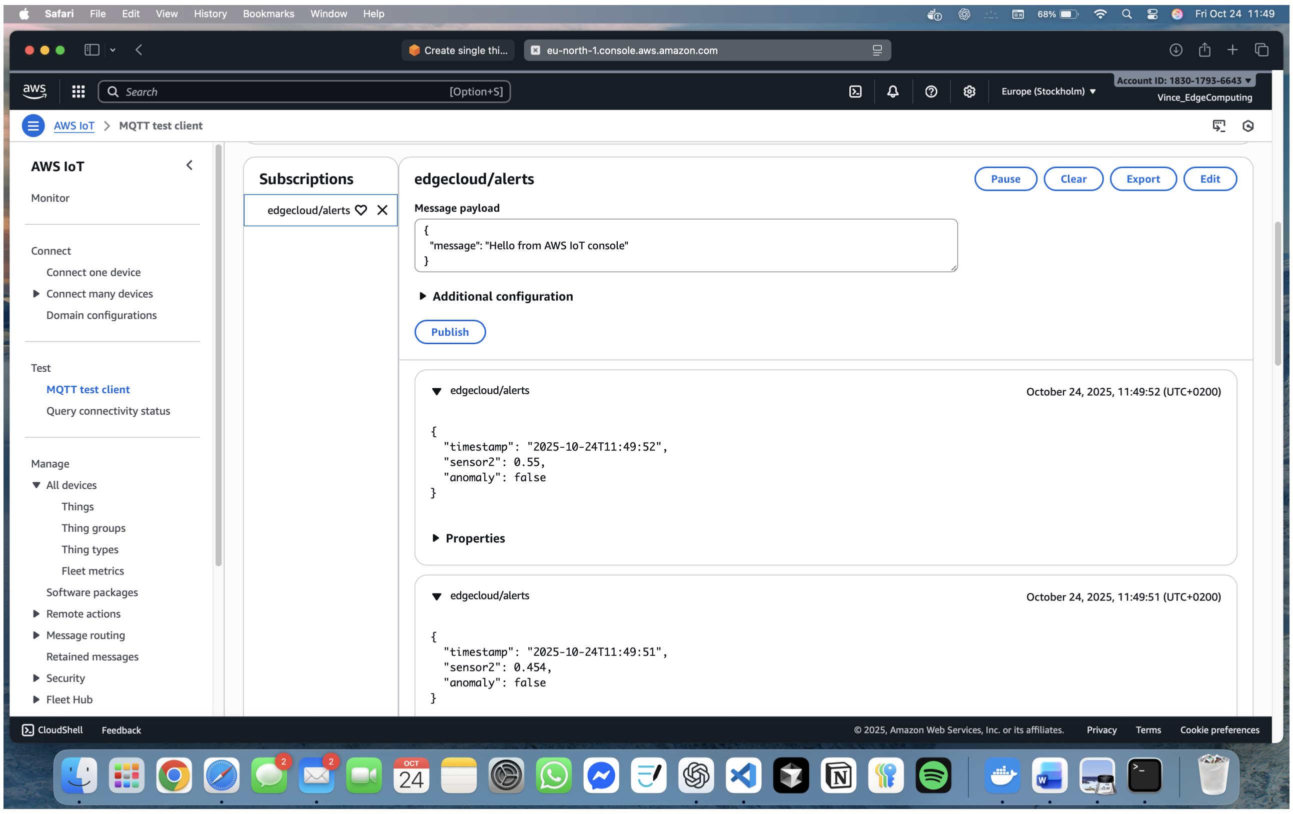
The experiment ultimately succeeded in producing a fully functional prototype of an edge–cloud system, despite the considerable challenges faced during its development. At the start, none of the tools or technologies — Docker, Kubernetes, or KubeEdge — were familiar. Building the system therefore became a process of exploration and persistence, relying heavily on generative AI as an interactive assistant. Many hours were spent resolving connection and configuration issues. For example, early in the setup, the Raspberry Pi could successfully ping the MacBook, confirming that both were on the same local network, yet MQTT messages were not arriving. After long troubleshooting sessions, it turned out that the broker was simply listening on the wrong port. Similarly, numerous Kubernetes error messages had to be understood and resolved one by one. Over time, more features were added, such as anomaly detection and the Isolation Forest model, followed by the creation of a Docker image and its integration with Kubernetes. In the end, everything was working together as intended. This process — entirely built through interaction between human reasoning and AI support — illustrates what true generative AI collaboration can look like in practice: problem-solving through conversation, iteration, and contextual reasoning, rather than copying existing solutions from the internet.

The experiment demonstrates that it is feasible for a small team — or even an individual — to set up a complete edge–cloud architecture with KubeEdge, K3D, and AWS IoT Core, even when the edge hardware is outdated.

[13] Amazon Web Services, "AWS IoT core — what is AWS IoT?." `https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html`. Accessed: 9 Nov. 2025; Amazon Web Services, "How AWS IoT works." `https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html`. Accessed: 9 Nov. 2025; academic overview in P. Borra, "Analyzing AWS edge computing solutions to enhance IoT deployments," *International Journal of Engineering and Advanced Technology (IJEAT)*, 2024. Accessed: 9 Nov. 2025.

[14] OASIS MQTT Technical Committee, "MQTT — the standard for IoT messaging." `https://mqtt.org`. Accessed: 9 Nov. 2025.

[15] Amazon Web Services, "AWS IoT core — what is AWS IoT?." `https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html`. Accessed: 9 Nov. 2025.



```
{
  "message": "Hello from AWS IoT console"
}
```

edgecloud/alerts — October 24, 2025, 11:49:52 (UTC+0200)

```
{
  "timestamp": "2025-10-24T11:49:52",
  "sensor2": 0.55,
  "anomaly": false
}
```

edgecloud/alerts — October 24, 2025, 11:49:51 (UTC+0200)

```
{
  "timestamp": "2025-10-24T11:49:51",
  "sensor2": 0.454,
  "anomaly": false
}
```

Figure 1.4: AWS IoT Core console showing a live subscription to the topic `edgecloud/alerts`. Each message originates from the edge-cloud subscriber container and represents a vibration reading enriched with its anomaly classification.

The system successfully showed how local computation, container orchestration, and cloud connectivity can be integrated into one coherent pipeline. Although the Raspberry Pi's older operating system prevented the deployment of KubeEdge's EdgeCore, the setup was kept KubeEdge-compatible, allowing future expansion with minimal adjustments. This achievement highlights how complex, industry-level architectures are now accessible to students and researchers without extensive infrastructure or prior experience, provided that sufficient time and curiosity are invested in understanding the underlying mechanisms.

Nevertheless, the resulting system also has limitations. Functionally, it does not yet represent the most efficient example of edge computing: the edge node still sends a continuous stream of data rather than aggregated or filtered results, and only a single simulated sensor is analysed. These simplifications were deliberate choices, prioritising architectural understanding over application complexity. For this experiment, the primary focus was the infrastructure — how the layers interact, deploy, and communicate securely — rather than the semantics of the data itself. The sensor readings were merely a tool to demonstrate functionality. Future work could expand this foundation by implementing true data reduction at the edge, managing multiple sensors, or deploying actual physical devices as managed KubeEdge nodes. Despite these open points, the experiment achieved its core goal: demonstrating, in a realistic and reproducible way, how modern edge–cloud architectures can be designed, deployed, and understood from scratch using open-source tools and the guidance of generative AI.

Bibliography

[1] OASIS MQTT Technical Committee, "MQTT — the standard for IoT messaging." `https://mqtt.org`. Accessed: 9 Nov. 2025.

[2] Amazon Web Services, "What is MQTT?." `https://aws.amazon.com/what-is/mqtt/`. Accessed: 9 Nov. 2025.

[3] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, "Extend cloud to edge with kubeedge," in *ACM/IEEE Symposium on Edge Computing*, 2018. Accessed: 9 Nov. 2025.

[4] OpenAI and Google, "Generative AI (chatgpt and gemini) — interactive support in experiment design." `https://openai.com`, 2025. Accessed: 9 Nov. 2025.

[5] Docker, Inc., "Docker desktop documentation." `https://docs.docker.com/desktop/`, 2025. Accessed: 9 Nov. 2025.

[6] K3D Project, "k3d — lightweight wrapper to run k3s in docker." `https://k3d.io`. Accessed: 9 Nov. 2025.

[7] D. Yakubov and D. Hästbacka, "Comparative analysis of lightweight kubernetes distributions for edge computing: Performance and resource efficiency," *arXiv preprint*, 2025. Accessed: 9 Nov. 2025.

[8] DataCamp, "What is kubernetes? an introduction with examples." `https://www.datacamp.com/blog/what-is-kubernetes`, 2024. Accessed: 9 Nov. 2025.

[9] OpsRamp, "An overview of kubernetes architecture." `https://www.opsramp.com/guides/why-kubernetes/kubernetes-architecture/`. Accessed: 9 Nov. 2025.

[10] Theodo Cloud, "Kubernetes overview: All you need to know." `https://cloud.theodo.com/en/blog/kubernetes-overview`. Accessed: 9 Nov. 2025.

[11] Collabnix, "How docker desktop works under the hood." `https://collabnix.com/how-docker-desktop-for-windows-works-under-the-hood/`, 2023. Accessed: 9 Nov. 2025.

[12] scikit-learn Developers, "Isolationforest — scikit-learn documentation." `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html`. Accessed: 9 Nov. 2025.

[13] Amazon Web Services, "AWS IoT core — what is AWS IoT?." `https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html`. Accessed: 9 Nov. 2025.

[14] Amazon Web Services, "How AWS IoT works." `https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html`. Accessed: 9 Nov. 2025.

[15] P. Borra, "Analyzing AWS edge computing solutions to enhance IoT deployments," *International Journal of Engineering and Advanced Technology (IJEAT)*, 2024. Accessed: 9 Nov. 2025.

Appendix A

Python subscriber and publisher

This appendix contains the edge subscriber and the Raspberry Pi publisher used in the experiment.

A.1 Mac subscriber (`mac_subscriber.py`)

Listing A.1: Mac subscriber with Isolation Forest anomaly detection and AWS IoT Core publishing

```python
# mac_subscriber.py -- Edge Cloud AI Subscriber
import os, json, time
from collections import deque
import numpy as np
from sklearn.ensemble import IsolationForest
import paho.mqtt.client as mqtt

# AWS IoT Core configuration
AWS_ENDPOINT = os.getenv("AWS_ENDPOINT", "akzdc7a60ugm9-ats.iot.eu-north-1.amazonaws.com")
AWS_PORT = 8883
AWS_TOPIC = "edgecloud/alerts"
AWS_CERT_DIR = os.path.join(os.path.dirname(__file__), "aws_certs")

AWS_CERT = os.path.join(AWS_CERT_DIR,
    ↪ "664ca5debe852d4e4b57ebe0688388694278de7c350202d2ed0d7a9c60d8133b-certificate.pem.crt")
AWS_KEY  = os.path.join(AWS_CERT_DIR,
    ↪ "664ca5debe852d4e4b57ebe0688388694278de7c350202d2ed0d7a9c60d8133b-private.pem.key")
AWS_ROOT_CA = os.path.join(AWS_CERT_DIR, "AmazonRootCA1.pem")

aws_client = mqtt.Client(client_id="EdgeCloudAWS", protocol=mqtt.MQTTv311)
aws_client.tls_set(ca_certs=AWS_ROOT_CA, certfile=AWS_CERT, keyfile=AWS_KEY)
aws_client.connect(AWS_ENDPOINT, AWS_PORT, keepalive=60)
aws_client.loop_start()

# MQTT configuration
BROKER = os.getenv("MQTT_BROKER", "host.docker.internal")
PORT = int(os.getenv("MQTT_PORT", "1884"))
TOPIC = os.getenv("MQTT_TOPIC", "sensor/data")

# Buffer for recent sensor2 values
window_size = 100
buffer_s2 = deque(maxlen=window_size)

# Isolation Forest model for anomaly detection
```

```python
model = IsolationForest(contamination=0.05, random_state=42)

def train_model():
    """Retrains the model with the current buffer data."""
    if len(buffer_s2) >= 20:
        data = np.array(buffer_s2).reshape(-1, 1)
        model.fit(data)
        print(f"[EdgeCloud] Model retrained on {len(buffer_s2)} samples")

def detect_anomaly(value):
    """Returns True if an anomaly is detected."""
    data = np.array([[value]])
    pred = model.predict(data)[0]  # 1 = normal, -1 = anomaly
    return pred == -1

def on_connect(client, userdata, flags, rc):
    print(f"[EdgeCloud] Connected with result code {rc}, subscribing to '{TOPIC}'")
    client.subscribe(TOPIC)

def on_message(client, userdata, msg):
    global model
    try:
        payload = json.loads(msg.payload.decode("utf-8"))
        sensor2 = payload.get("sensor2", None)
        timestamp = payload.get("timestamp", "?")
        if sensor2 is None:
            return

        # Add to buffer and retrain periodically
        buffer_s2.append(sensor2)
        if len(buffer_s2) % 20 == 0:
            train_model()

        # Detect anomalies when trained
        if len(buffer_s2) >= 20:
            is_anomaly = detect_anomaly(sensor2)
            status = "[WARN] ANOMALY" if is_anomaly else "[OK] OK"
            print(f"[EdgeCloud] {timestamp} | s2={sensor2:.3f} -> {status}")

            # Publish anomaly results to AWS IoT Core with safe serialization
            try:
                result_payload = {
                    "timestamp": str(timestamp),
                    "sensor2": float(sensor2),
                    "anomaly": bool(is_anomaly)
                }
                payload_json = json.dumps(result_payload, default=str)
                aws_client.publish(AWS_TOPIC, payload_json)
                print(f"[EdgeCloud] Published anomaly result to AWS IoT Core: {payload_json}")
            except Exception as e:
                print("[EdgeCloud] Error publishing to AWS:", e)

    except Exception as e:
        print("[EdgeCloud] Error parsing message:", e)

# MQTT setup
client = mqtt.Client(client_id="EdgeCloudSubscriber", protocol=mqtt.MQTTv311)
client.on_connect = on_connect
client.on_message = on_message

print(f"[EdgeCloud] Connecting to {BROKER}:{PORT}")
client.connect(BROKER, PORT, keepalive=60)
client.loop_forever()
```

A.2 Pi publisher (`pi_publisher.py`)

Listing A.2: Raspberry Pi publisher generating `sensor1` and `sensor2` with occasional outliers

```python
#===== pi_publisher.py =====
import time, random, json
import paho.mqtt.client as mqtt

BROKER = "192.168.1.105"   # <- your MacBook's local IP (or host name)
PORT   = 1884
TOPIC  = "sensor/data"

client = mqtt.Client("PiPublisher")
client.connect(BROKER, PORT, 60)

print("[Pi] Connected to broker:", BROKER)

def generate_sensor_values():
    # Simulate two sensors: smooth + noisy
    sensor1 = 20 + random.uniform(-0.5, 0.5)          # temperature-like
    sensor2 = 0.5 + random.uniform(-0.05, 0.05)       # vibration baseline

    # Random outlier 8% of the time
    if random.random() < 0.08:
        if random.random() < 0.5:
            sensor2 += random.uniform(2, 3)   # spike
        else:
            sensor2 -= random.uniform(2, 3)   # sudden drop
    return sensor1, sensor2

while True:
    sensor1, sensor2 = generate_sensor_values()
    payload = {
        "timestamp": time.strftime("%Y-%m-%dT%H:%M:%S"),
        "sensor1": round(sensor1, 3),
        "sensor2": round(sensor2, 3)
    }
    client.publish(TOPIC, json.dumps(payload))
    print("[Pi] Published:", payload)
    time.sleep(1)
```

Appendix B

Container image and Kubernetes manifest

This appendix lists the container image definition and the Kubernetes Deployment used to run the subscriber.

B.1 Dockerfile

Listing B.1: Dockerfile for the edgecloud-subscriber image

```
FROM python:3.11-slim
WORKDIR /app
# COPY mac_subscriber.py .
COPY mac_subscriber.py /app/
COPY aws_certs /app/aws_certs/
# Pin to paho-mqtt 1.x to avoid the v2 callback API change
RUN pip install --no-cache-dir paho-mqtt numpy scikit-learn
ENV PYTHONUNBUFFERED=1
CMD ["python", "mac_subscriber.py"]
```

B.2 Kubernetes Deployment (`subscriber-deployment.yaml`)

Listing B.2: Deployment manifest for the MQTT subscriber pod

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mqtt-subscriber
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mqtt-subscriber
  template:
    metadata:
      labels:
        app: mqtt-subscriber
    spec:
```

```yaml
      containers:
      - name: subscriber
        image: edgecloud-subscriber:latest
        imagePullPolicy: Never
        env:
        - name: MQTT_BROKER
          value: host.docker.internal
        - name: MQTT_PORT
          value: "1884"
```

Appendix C

Build and run instructions

This appendix summarises how to build the container image, load it into the local cluster, and deploy the subscriber.

C.1 Build the image locally

Listing C.1: Build the subscriber image

```
cd edge_computing
docker build -t edgecloud-subscriber:latest .
```

C.2 Run in a local Kubernetes cluster

If you use Docker Desktop, the local image is available to the cluster automatically. For a kind cluster:

Listing C.2: Load image into a kind cluster (if used)

```
kind load docker-image edgecloud-subscriber:latest
```

Deploy the subscriber:

Listing C.3: Deploy the MQTT subscriber

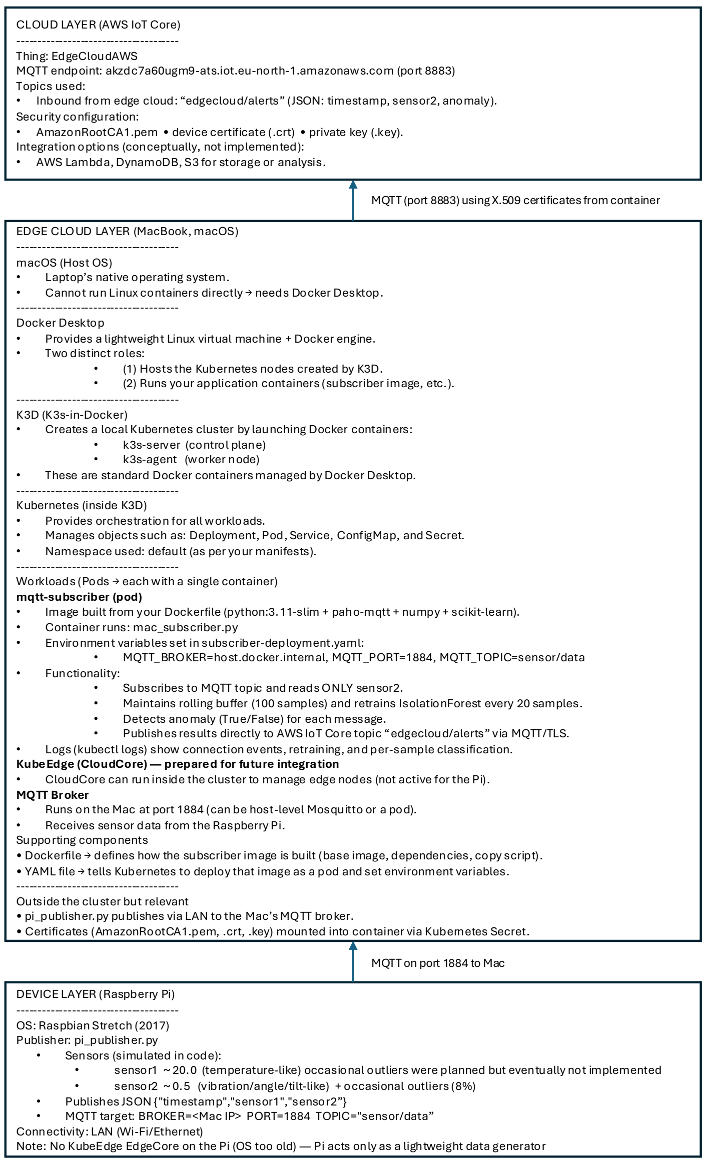
```
kubectl apply -f edge_computing/subscriber-deployment.yaml
kubectl get pods -l app=mqtt-subscriber
kubectl logs -f deploy/mqtt-subscriber
```

C.3 Certificates and environment

The subscriber publishes results to AWS IoT Core using certificates located under `edge_computing/aws_certs/`. Do not commit real keys or certificates to the repository.

Appendix D

Architectural overview diagram



CLOUD LAYER (AWS IoT Core)

Thing: EdgeCloudAWS
MQTT endpoint: akzdc7a60ugm9-ats.iot.eu-north-1.amazonaws.com (port 8883)
Topics used:

- Inbound from edge cloud: “edgecloud/alerts” (JSON: timestamp, sensor2, anomaly).

Security configuration:

- AmazonRootCA1.pem • device certificate (.crt) • private key (.key).

Integration options (conceptually, not implemented):

- AWS Lambda, DynamoDB, S3 for storage or analysis.

MQTT (port 8883) using X.509 certificates from container

EDGE CLOUD LAYER (MacBook, macOS)

----

macOS (Host OS)

- Laptop’s native operating system.
- Cannot run Linux containers directly → needs Docker Desktop.

Docker Desktop

- Provides a lightweight Linux virtual machine + Docker engine.
- Two distinct roles:
    - (1) Hosts the Kubernetes nodes created by K3D.
    - (2) Runs your application containers (subscriber image, etc.).

K3D (K3s-in-Docker)

- Creates a local Kubernetes cluster by launching Docker containers:
    - k3s-server (control plane)
    - k3s-agent (worker node)
- These are standard Docker containers managed by Docker Desktop.

Kubernetes (inside K3D)

- Provides orchestration for all workloads.
- Manages objects such as: Deployment, Pod, Service, ConfigMap, and Secret.
- Namespace used: default (as per your manifests).

Workloads (Pods → each with a single container)
mqtt-subscriber (pod)

- Image built from your Dockerfile (python:3.11-slim + paho-mqtt + numpy + scikit-learn).
- Container runs: mac_subscriber.py
- Environment variables set in subscriber-deployment.yaml:
    - MQTT_BROKER=host.docker.internal, MQTT_PORT=1884, MQTT_TOPIC=sensor/data
- Functionality:
    - Subscribes to MQTT topic and reads ONLY sensor2.
    - Maintains rolling buffer (100 samples) and retrains IsolationForest every 20 samples.
    - Detects anomaly (True/False) for each message.
    - Publishes results directly to AWS IoT Core topic “edgecloud/alerts” via MQTT/TLS.
- Logs (kubectl logs) show connection events, retraining, and per-sample classification.

KubeEdge (CloudCore) — prepared for future integration

- CloudCore can run inside the cluster to manage edge nodes (not active for the Pi).

MQTT Broker

- Runs on the Mac at port 1884 (can be host-level Mosquitto or a pod).
- Receives sensor data from the Raspberry Pi.

Supporting components
• Dockerfile → defines how the subscriber image is built (base image, dependencies, copy script).
• YAML file → tells Kubernetes to deploy that image as a pod and set environment variables.

Outside the cluster but relevant
• pi_publisher.py publishes via LAN to the Mac’s MQTT broker.
• Certificates (AmazonRootCA1.pem, .crt, .key) mounted into container via Kubernetes Secret.

MQTT on port 1884 to Mac

DEVICE LAYER (Raspberry Pi)

OS: Raspbian Stretch (2017)
Publisher: pi_publisher.py

- Sensors (simulated in code):
    - sensor1 ~20.0 (temperature-like) occasional outliers were planned but eventually not implemented
    - sensor2 ~0.5 (vibration/angle/tilt-like) + occasional outliers (8%)
- Publishes JSON {"timestamp","sensor1","sensor2”}
- MQTT target: BROKER=<Mac IP> PORT=1884 TOPIC="sensor/data”

Connectivity: LAN (Wi-Fi/Ethernet)
Note: No KubeEdge EdgeCore on the Pi (OS too old) — Pi acts only as a lightweight data generator