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