📦GPU-Aware KEDA-Based Posture Analyzer - Documentation

This project enables intelligent posture analysis of images sent over MQTT, using GPU-optimized Kubernetes Jobs dynamically scheduled via KEDA. The system is designed to:

- Automatically select the worker node with the lowest GPU load.
- Trigger containerized posture-analysis jobs on that node.
- Log results to a PostgreSQL database.

⚙️Requirements

🔗Software Dependencies

Install the following using `pip install -r requirements.txt`

```
opencv-python
mediapipe
paho-mqtt
numpy
matplotlib
psycopg2-binary
psutil
prometheus-api-client
keyboard
```

🔗Environment

- Kubernetes Cluster with KEDA installed
- Prometheus running and scraping all GPU nodes
- Docker with `buildx` and access to Docker Hub (for pushing image)
- Supabase PostgreSQL credentials as environment variables:
- `SUPABASE_HOST`
- `SUPABASE_DB`
- `SUPABASE_USER`
- `SUPABASE_PASSWORD`
- Optional: `SUPABASE_PORT`, `SUPABASE_SSL`

📁Folder and File Overview

File/Folder	Description
`build_and_push.py`	Main launcher: builds Docker image, applies Kubernetes setup, starts watchers
`mqtt_posture_analyzer_with_db.py`	Core container logic: receives images, analyzes posture, logs results
`gpu_affinity_watcher.py`	Selects the worker node with least GPU load and patches KEDA ScaledJob
`patch_scaledjob.py`	Adds node affinity to `posture-job.yaml` to force it to run on selected worker
`cpu_monitor_and_offload.py`	Implements gRPC scaler for KEDA using Prometheus GPU metrics
`stop.py`	Graceful shutdown script to terminate all posture jobs and clean resources
`posture-job.yaml`	Base KEDA ScaledJob template
`Dockerfile`	Builds the posture analysis image
`docker-compose.yml`	Optional local testing setup
`externalscaler.proto`	gRPC proto definition for KEDA scaler
`externalscaler_pb2.py` / `externalscaler_pb2_grpc.py`	gRPC-generated interfaces for KEDA
`requirements.txt`	All Python dependencies

🗜️Setup Instructions

1. 🔧Configure Prometheus

- Ensure your Prometheus instance is running and scraping GPU metrics for all nodes:
- Node 1: `jetson_gpu_usage_percent`
- Node 2: `jetson_orin_gpu_load_percent`

2. 📠Build and Push Docker Image

Make sure you're logged in to Docker Hub and then:

```
python3 build_and_push.py
```

This will:

- Stop/remove any existing containers or jobs
- Build a multi-architecture image `shahroz90/posture-analyzer`
- Patch the ScaledJob YAML with node affinity
- Deploy it to the cluster
- Start GPU/CPU watchers and the ESC listener

3. 📡MQTT Broker

Set up an MQTT broker at `192.168.1.79` (or modify `mqtt_posture_analyzer_with_db.py` if needed).

- Devices should publish Base64-encoded JPEGs to topics like:
- `images/pi1`
- `images/pi2`

4. Supabase Database Setup

Create a PostgreSQL table:

```
CREATE TABLE posture_log (
  id SERIAL PRIMARY KEY,
  pi_id TEXT,
  filename TEXT,
  received_time TIMESTAMP,
  analyzed_time TIMESTAMP,
  neck_angle INTEGER,
  body_angle INTEGER,
  posture_status TEXT,
  landmarks_detected BOOLEAN,
  processed_by TEXT
);
```

🔦Runtime Process

Once `build_and_push.py` is executed:

1. **GPU Watcher** (`gpu_affinity_watcher.py`) selects the best GPU node and patches `posture-job.yaml`.
2. **CPU Monitor + ESC Listener** (`cpu_monitor_and_offload.py`) runs a gRPC server for KEDA.
3. KEDA uses the gRPC scaler to decide if GPU usage is low enough to scale.
4. If yes, KEDA schedules a new job based on `posture-job.yaml`.

5. The container runs `mqtt_posture_analyzer_with_db.py`, listens for MQTT images, and logs analysis results.

⚙️Stopping the System

Just run:

```
python3 stop.py
```

It will:

- Kill GPU/CPU watcher scripts
- Delete all posture-analyzer jobs and pods
- Untaint the master node (optional)
- Remove `patched-job.yaml`

🧪Testing the System

1. Run `python3 build_and_push.py`
2. Publish an image to `images/pi1` topic over MQTT
3. Within seconds, the container will:
4. Receive the image
5. Analyze posture
6. Save annotated image to `./analyzed_images/...`
7. Insert analysis metadata into the PostgreSQL database
