TP ARNG

Objectifs
- Mettre en service une chaîne IoT complète basée sur Mosquitto, un capteur simulé en Python, une API REST Python et InfluxDB.
- Implémenter un schéma cohérent topics MQTT ↔ mesures InfluxDB et l'exposer via l'API.
- Instrumenter la solution (QoS, rétention, supervision) par l'observation terrain.

Organisation Travail en binômes ; restitution technique courte (5 min) en fin de séance.

Fil rouge de la séance

Scénario : entrepôt multi-zones
- Des capteurs température/humidité répartis sur trois zones publient leurs mesures via MQTT.
- Un middleware applique des règles (seuils, alertes), alimente InfluxDB et expose une API REST.
- Les équipes métiers consultent des dashboards afin d'orienter leurs décisions.

Consigne générale Toutes les questions s'appuient sur ce fil rouge. Explicitez vos hypothèses lorsque nécessaire.

Stack conteneurisée fournie

Services Docker
- `broker` : image Mosquitto custom (`docker/mosquitto/mosquitto.conf` à adapter).
- `influxdb` : service 2.7 initialisé avec org `ares`, bucket `entrepot`, token `dev-token`.
- `api` : FastAPI + Uvicorn basé sur `services/api/main.py` et `tp_mqtt.py`.
- `sensor` : boucle Python (`services/sensor/main.py`) réutilisant `tp_mqtt.SensorSimulator`.

Commande "docker compose up –build"

Principes
- Il vous est demandé de modifier uniquement `docker/mosquitto/mosquitto.conf`, `tp_mqtt.py`, `services/api/main.py` et `services/sensor/main.py`.
- Les dépendances Python sont gérées via `requirements.txt`. Le volume `.:/app` permet le rechargement à chaud (relancer le conteneur si besoin).

Activité 1 — Mise en service Mosquitto

Question Déployez un broker Mosquitto (service local ou conteneur) et documentez : configuration minimale, ports exposés, sécurité retenue. (Rappel QoS — doc Mosquitto 2.0 : https://mosquitto.org/man/mqtt-7.html)

À préciser

- Vérifier la disponibilité via `mosquitto_sub`/`mosquitto_pub` sur les topics `entrepot/<zone>/<capteur>/v1`.
- Consolider les choix QoS/rétention pour les trois zones de tests.
- Commenter les paramètres modifiés dans `docker/mosquitto/mosquitto.conf`.
- Illustrer la différence entre les jokers + et # en montrant un exemple de souscription utile.

Procédure guidée “(terminal 1) docker compose up broker

(terminal 2) docker compose exec broker mosquitto_sub -t “entrepot/ -v

(terminal 3)
docker compose exec broker mosquitto_pub -t entrepot/nord/temp/v1” -m “{"temp":23.1,"hum":44.0}” -q 1”

Attention permissions

- Après chaque modification de `docker/mosquitto/mosquitto.conf` ou des variantes `qos*.conf`, réappliquez les droits attendus par le conteneur : “chown 1883:1883 docker/mosquitto/.conf”
- Sans cela, Mosquitto (UID 1883) ne pourra pas sauvegarder/recharger la configuration.

Restitution

- Capture d’écran / log montrant un aller-retour publish/subscribe opérationnel accompagné d’un court commentaire.
- Trois fichiers complets `qos0.conf`, `qos1.conf`, `qos2.conf` (copie de `mosquitto.conf` + vos ajustements pour chaque QoS).

Activité 2 — Capteur simulé en Python

Travail demandé

1. Écrire un script Python (paho-mqtt ou équivalent) simulant un capteur température/humidité par zone. Structurer le payload JSON (`timestamp`, `temp`, `hum`, `alert_level`, `version`) et publier périodiquement sur les topics définis. Préparer le measurement InfluxDB associé (measurement, tags zone/capteur, fields numériques).

Questions guidées

- Comment distinguer rapidement une alerte critique d'une mesure périodique côté topic/payload ?
- Quels tags permettent de filtrer par zone et type de capteur sans alourdir la base ?
- Où placer la logique (dans `tp_mqtt.py` vs `services/sensor/main.py`) pour faciliter les tests ?

Procédure guidée

- Compléter `SensorSimulator.compute_payload` et ajuster `SensorConfig` (`tp_mqtt.py`, `services/sensor/main.py`).
- Démarrer `sensor` (broker déjà lancé) et suivre les journaux.
- Contrôler les topics et payloads avec `mosquitto_sub`.

Commandes utiles

- `docker compose up sensor`
- `docker compose logs -f sensor`
- `docker compose exec broker mosquitto_sub -t "entrepot/#" -v`

Extrait heartbeat (tp_mqtt.py)

if int(time.time()) % 30 == 0: hb_topic = f“entrepot/heartbeat/{zone}” hb_payload = json.dumps({“zone”: zone, “timestamp”: time.time()}) self.client.publish(hb_topic, hb_payload, qos=0, retain=True)

Vérifications attendues

- Les logs `sensor` montrent un cycle complet sur les trois zones avec un payload JSON structuré.
- `mosquitto_sub` reflète le topic exact `entrepot/<zone>/<capteur>/v1` et met en évidence `alert_level`.

Livrable

- Tableau synthétique (topics tags InfluxDB) + schéma measurement/tags/fields.
- Extrait de code Python (ou lien vers commit) montrant `compute_payload` et la configuration `SensorConfig`.
- Capture `mosquitto_sub` confirmant les deux cas : mesure nominale + alerte.

Activité 3 — API REST Python & InfluxDB

Cas pratique

- Implémenter une API REST (FastAPI ou Flask) consommant InfluxDB et exposant les mesures.
- Routes minimales : `GET /zones/{zone}/latest`, `POST /alerts`, `GET /thresholds`.
- L'API doit corréler ce qui arrive via MQTT avec ce qui est écrit dans InfluxDB.
- Doc utiles : https://fastapi.tiangolo.com/ et https://docs.influxdata.com/influxdb/cloud/

Fichiers à compléter

- `tp_mqtt.py` : fonction `create_app`, requêtes Flux, modèles Pydantic.
- `services/api/main.py` : chargement des variables d'environnement et instanciation FastAPI.
- `docker-compose.yml` (service `api`) si vous avez besoin d'exposer d'autres variables.

Questions

- Comment garantir idempotence entre MQTT QoS et écritures InfluxDB ?
- Quelle politique de rétention mettre en place selon le besoin métier (temps réel vs historique) ?
- Comment exposer proprement les variables d'environnement dans `services/api/main.py` ?

Production attendue Documentation API (tableau routes / description / exemple curl) + justification des paramètres QoS/rétention.

Activité 4 — Dashboard InfluxDB & supervision

Questions guidées

- Construire un dashboard InfluxDB (ou Grafana branché sur InfluxDB) affichant : températures, humidités, alertes.
- Quels tests manuels (curl, Postman) documenter pour valider la cohérence MQTT ↔ REST ↔ Dashboard ?
- Quelles métriques de supervision collecter (latence, backlog messages, état broker) ?

Guidelines métriques

- `temp/hum` par zone (moyenne glissante + seuils) pour détecter une dérive capteur.
- `alert_level` (compteur par severité) synchronisé avec le flux REST `POST /alerts`.
- Latence approximative : comparer `_time` Influx et `time()` API (diff affichée).

- Supervision broker : publier un heartbeat sur `entrepot/heartbeat/<zone>` toutes les 30 s (`SensorSimulator`) et afficher `now() - last(_time)` dans InfluxDB pour vérifier que le broker répond.

Livrable Liste des widgets/graphes, checklist de tests bout-en-bout et trois indicateurs clés de supervision.