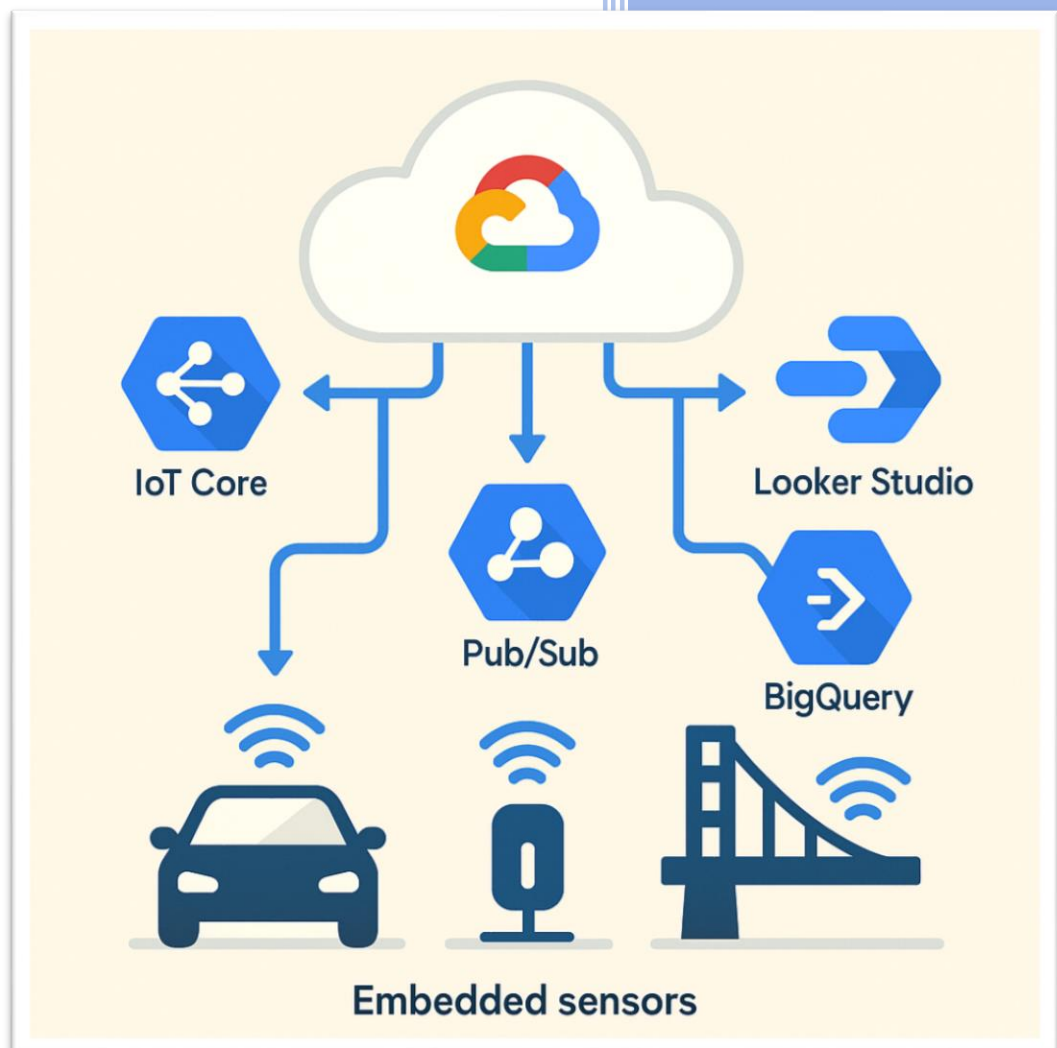




2025

Surveillance et analyse de capteurs embarqués avec GCP



Rédigé par : Mohamed Ali Cherif

Table des matières

CHAPITRE I :	4
Introduction Générale :	4
CHAPITRE II :	5
Contexte et enjeux des systèmes embarqués :	5
CHAPITRE III :	7
Architecture proposée et choix de conception :	7
1.Schéma global :	7
2. Conception fonctionnelle et technique :	8
2.2. Exigences non-fonctionnelles :	9
2.3. Cas d'utilisation :	9
2.4 Diagramme de séquence (simplifié) :	10
2.5. Modélisation des données :	11
CHAPITRE IV :	12
4. Développement des composants :	12
4.1. Script de simulation de capteurs :	12
4.2 Cloud Function de traitement :	12
4.3. Provisionne des services GCP :	12
CHAPITRE V :	20
Réalisation et résultat :	20
CHAPITRE V :	22
Conclusion et perspectives :	22

Listes des figures	
Figure 1 architecture de projet.....	7
Figure 2:tableau de cas d'utilisation.....	9
Figure 3 diagramme de séquence	10
Figure 4:modèle de table.....	11
Figure 5:dashboard et source.....	19
Figure 6:script python.....	20
Figure 7:big query data.....	20
Figure 8:cloud storage.....	20
Figure 9:cloud run hosting	21
Figure 10:looker rapport	21

CHAPITRE I :

Introduction Générale :

Au cœur de la révolution Internet des Objets (IoT), les capteurs embarqués jouent un rôle crucial en permettant la collecte continue de données physiques (température, pression, vibration, géolocalisation, etc.) directement depuis le terrain. Ces dispositifs, souvent déployés dans des environnements variés — industrie 4.0, transports, smart cities, agriculture de précision — génèrent un volume de données massif, hétérogène et à la fois critique et sensible. Pour en tirer une valeur opérationnelle réelle, il est indispensable de mettre en place une chaîne de traitement capable à la fois de **surveiller** en temps réel l'état des capteurs et leurs métriques, et d'**analyser** ces flux pour détecter des tendances, anticiper des incidents ou optimiser des processus.

Google Cloud Platform (GCP) offre un ensemble de services managés couvrant l'intégralité de ce cycle de vie data :

- **Cloud IoT Core** : pour l'enregistrement et l'ingestion sécurisée des appareils IoT.
- **Pub/Sub** : pour le routage et la mise en file d'attente des messages capteurs.
- **Dataflow** : pour le traitement en flux et la transformation des données.
- **BigQuery** : pour le stockage évolutif et les requêtes analytiques à grande échelle.
- **Looker Studio** : pour la création de tableaux de bord interactifs et le reporting visuel.

L'objectif de ce rapport est de présenter une architecture de surveillance et d'analyse de capteurs embarqués reposant sur GCP, d'en détailler les composants techniques, et de démontrer, à travers un cas d'usage concret, comment cette plateforme permet de :

1. Garantir la fiabilité et la sécurité de la collecte de données IoT,
2. Assurer un traitement évolutif et en continu pour des volumes croissants,
3. Fournir des indicateurs et des visualisations exploitables par les équipes opérationnelles et décisionnelles.

Nous débuterons par un rappel du contexte et des enjeux liés aux systèmes embarqués (Chapitre 2), puis détaillerons l'architecture proposée et les choix de conception (Chapitre 3). Le Chapitre 4 couvrira la mise en œuvre technique et les tests

de performance, avant de conclure sur les apports, les limites et les perspectives d'évolution de la solution (Chapitre 5).

CHAPITRE II :

Contexte et enjeux des systèmes embarqués :

Les **systèmes embarqués** désignent des dispositifs informatiques intégrés au sein d'équipements ou d'objets physiques, souvent dédiés à une tâche spécifique (contrôle moteur, mesure de grandeurs physiques, pilotage de processus industriels, etc.). Contrairement aux architectures informatiques traditionnelles, ces systèmes se caractérisent par :

- **Des ressources limitées** (puissance de calcul, mémoire, capacité de stockage).
- **Des contraintes de consommation énergétique** pour assurer une autonomie prolongée (batteries, énergie récupérée).
- **Une connectivité souvent intermittente** (réseaux cellulaires, LPWAN, Wi-Fi, Bluetooth Low Energy).
- **Des impératifs de fiabilité et de robustesse**, parfois dans des environnements sévères (température, humidité, vibrations).

Par ailleurs, l'essor rapide de l'Internet des Objets (IoT) a multiplié le nombre de capteurs déployés : on estime plusieurs dizaines de milliards d'appareils connectés à l'horizon 2025. Cette explosion soulève plusieurs **enjeux clés** pour les organisations qui veulent tirer parti de ces données :

1. **Scalabilité et ingestion de données**
 - Gérer un volume croissant de messages issus de milliers voire de millions de capteurs,
 - Assurer un pipeline d'ingestion en temps réel sans goulets d'étranglement.
2. **Sécurité et confidentialité**
 - Garantir l'authentification et l'autorisation des objets connectés,
 - Prévenir le piratage et la falsification de données sensibles (e.g. mesures industrielles, données de santé).
3. **Qualité et intégrité des données**
 - Détecter et corriger les erreurs de mesure, les valeurs aberrantes ou les pannes de capteur,

- Horodater et synchroniser précisément les relevés issus de sources hétérogènes.
4. **Maintenance prédictive et réactivité opérationnelle**
- Exploiter les données pour anticiper les défaillances (vibrations, échauffement),
 - Mettre en place des alertes automatisées pour éviter les arrêts non planifiés.
5. **Coût total de possession (TCO)**
- Optimiser l'infrastructure cloud et la facturation (stockage, traitement, bande passante),
 - Choisir des architectures modulaires et évolutives pour limiter les refontes à l'avenir.

Ces défis imposent la mise en place d'une **architecture cloud** robuste capable de gérer l'ensemble du cycle de vie des données IoT : de la collecte sécurisée à l'analyse avancée. C'est précisément ce rôle que joue Google Cloud Platform, grâce à ses services managés couvrant l'ingestion, le traitement en flux et par lot, l'entreposage de données à grande échelle et la visualisation interactive. Dans le chapitre suivant, nous décrirons en détail l'architecture proposée pour répondre à ces enjeux.

CHAPITRE III :

Architecture proposée et choix de conception :

Dans ce chapitre, nous détaillons l'architecture retenue pour la surveillance et l'analyse de capteurs embarqués sur Google Cloud Platform, ainsi que les décisions techniques motivant chaque choix de composant.

1.Schéma global :

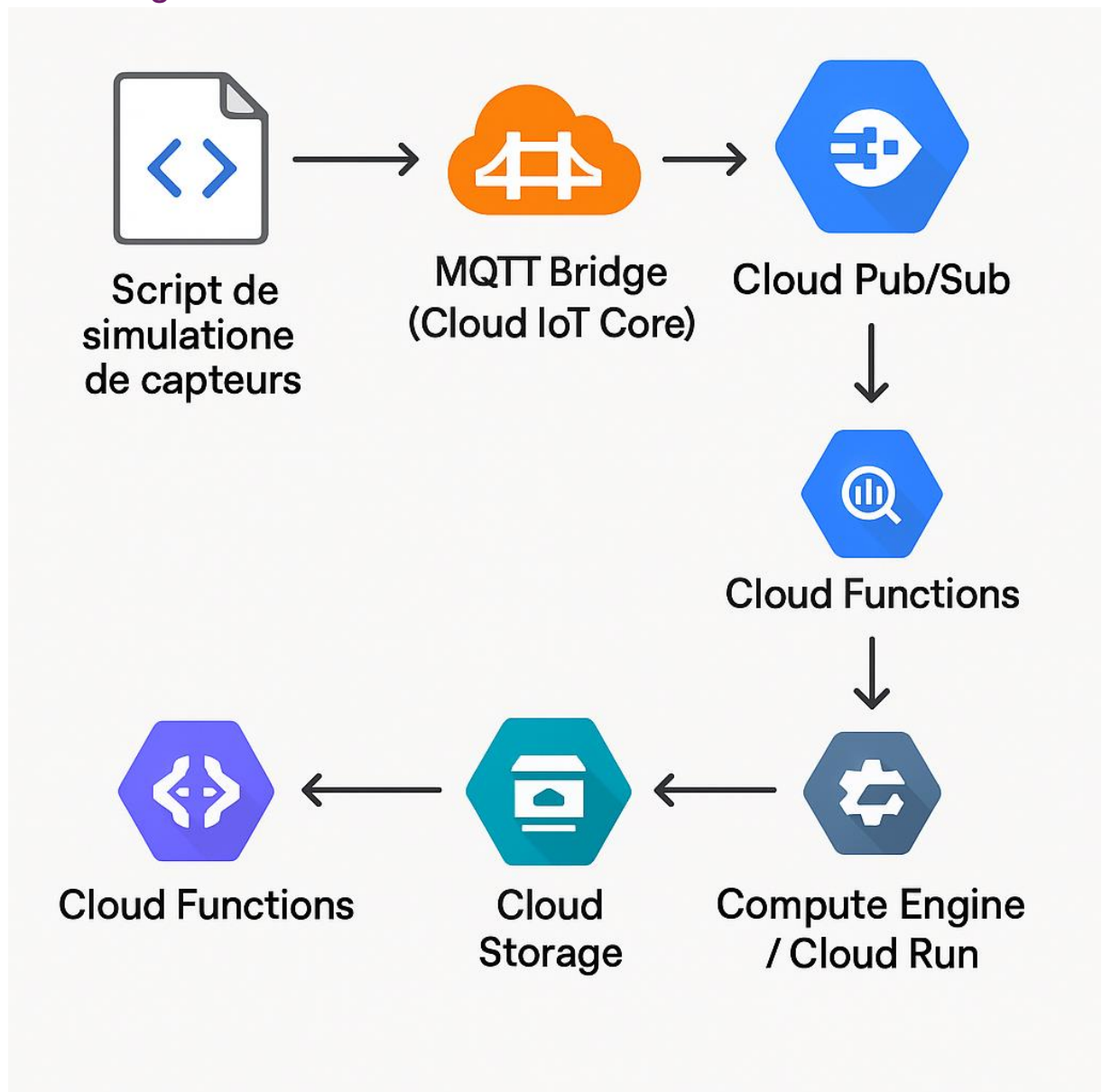


Figure 1 architecture de projet

Le diagramme illustre, au format « flat », le parcours complet des données issues de capteurs simulés jusqu'à leur exploitation et stockage dans Google Cloud :

1. **Script de simulation de capteurs**
 - Point de départ du flux : un script Python génère périodiquement des messages JSON (identifiant du capteur, température, humidité, vibration, timestamp).
2. **MQTT Bridge (Cloud IoT Core)**
 - Le script publie ces messages via un pont MQTT, émulant le service Cloud IoT Core pour gérer l'authentification et la réception des appareils.
3. **Cloud Pub/Sub**
 - Les messages arrivent dans un topic Pub/Sub unique (sensor-data), assurant la mise en file d'attente, la tolérance aux pannes et la faible latence (< 100 ms).
4. **Cloud Functions**
 - Chaque nouveau message déclenche une fonction serverless :
 - Validation et nettoyage du JSON,
 - Détection d'anomalies (ex. température > 70 °C),
 - Rédirection simultanée des données vers BigQuery et archivage brut dans Cloud Storage.
5. **BigQuery**
 - Stockage analytique : table partitionnée par date (_PARTITIONTIME) et clusterisée sur l'ID du capteur pour optimiser les requêtes temporelles et par appareil.
6. **Cloud Storage**
 - Bucket d'archives contenant les fichiers JSON bruts, transférés automatiquement depuis la fonction, avec règles de cycle de vie pour purger les données de plus de 90 jours.
7. **Looker Studio**
 - Connecteur natif BigQuery en mode requête directe, fournissant des tableaux de bord interactifs (nombre d'anomalies, courbes de température, filtres par date et capteur).
8. **Compute Engine / Cloud Run**
 - Exposition d'une API REST ou d'un micro-service web pour consulter, en quasi-temps réel, l'état d'un capteur simulé (interface embarquée).

Les flèches noires indiquent la direction du flux de données, de la simulation jusqu'à l'interface finale, mettant en évidence la nature serverless et modulaire de l'architecture.

2. Conception fonctionnelle et technique :

Avant de passer à la partie implémentation, il est essentiel de préciser la conception, c'est-à-dire comment les exigences se traduisent en modules, flux et interactions.

2.1 Exigences fonctionnelles

1. **Simulation et injection de données**
 - Générer périodiquement un jeu de mesures { capteur_id, température, humidité, vibration, timestamp }.
 - Publier via MQTT Bridge en respectant le format JSON.

2. **Collecte et routage**
 - Mettre en file d'attente chaque message dans Pub/Sub, garantir la persistance et la livraison au consommateur.
3. **Traitement et enrichissement**
 - Nettoyer le payload, détecter et marquer les anomalies.
 - Dupliquer les flux vers stockage analytique et archives.
4. **Stockage et archivage**
 - Stocker en streaming dans BigQuery selon un schéma partitionné et clusterisé.
 - Sauvegarder les JSON bruts dans Cloud Storage avec cycle de vie.
5. **Visualisation et consultation**
 - Exposer des rapports dynamiques via Looker Studio.
 - Permettre la requête ad hoc de l'état d'un capteur via Compute Engine/Cloud Run.

2.2. Exigences non-fonctionnelles :

- **Scalabilité** : absorber des pointes à plusieurs milliers de messages par seconde.
- **Temps réel** : latence bout-à-bout < 1 s.
- **Fiabilité** : reprise automatique sur panne (replay Pub/Sub, retries Cloud Functions).
- **Sécurité** : chiffrement TLS, IAM fines, isolation serverless.
- **Coût maîtrisé** : privilégier les services sans-serveur et optimiser partitionnement/clustering.

2.3. Cas d'utilisation :

ID	Scenario	Acteur	Résultat attendu
CU1	Lancer la simulation de 5 capteurs virtuels	Opérateur DevOps	messages/s publiés sur Pub/Sub
CU2	Détecter une surchauffe (> 70 °C)	Cloud Function	Flag anomalie:true ajouté et message archivé
CU3	Visualiser les anomalies des dernières 24 h	Technicien	Dashboard Looker avec filtre "24 heures"
CU4	Obtenir l'état courant d'un capteur #123	API REST	JSON {capteur_id:123, temp:45, statut:"OK",...} renvoyé

Figure 2:tableau de cas d'utilisation

2.4 Diagramme de séquence (simplifié) :

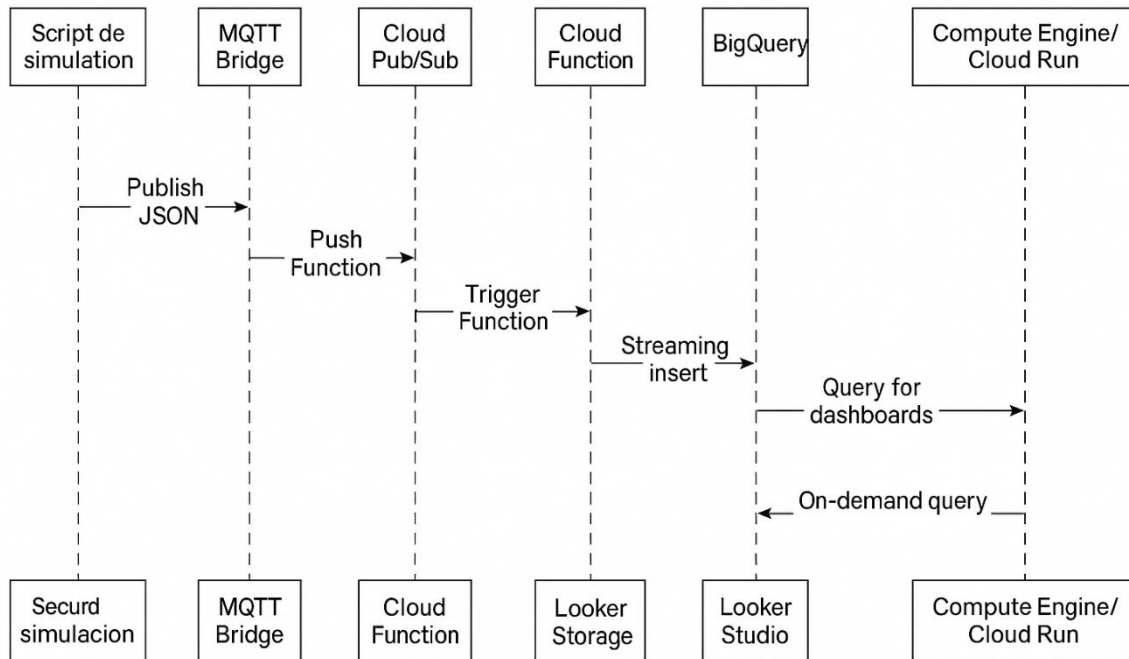


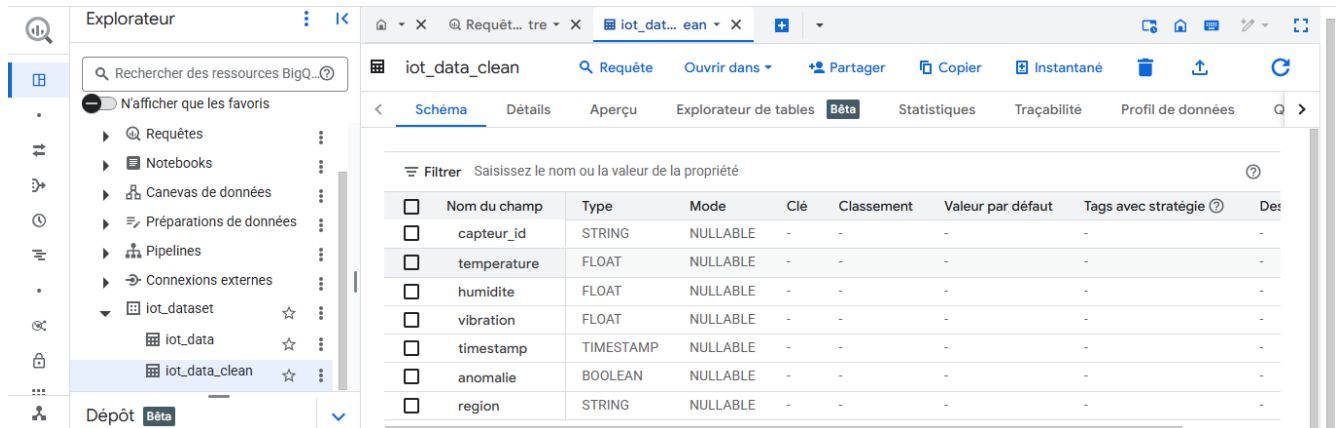
Figure 3 diagramme de séquence

Le diagramme de séquence simplifié illustre :

1. **Script de simulation de capteurs** publie un message JSON vers **MQTT Bridge**.
2. **MQTT Bridge** pousse le message dans **Cloud Pub/Sub**.
3. **Cloud Pub/Sub** déclenche **Cloud Function**.
4. **Cloud Function** réalise un **streaming insert** dans **BigQuery**, puis archive le JSON dans **Cloud Storage**.
5. **Looker Studio** interroge **BigQuery** pour générer les tableaux de bord.
6. **Compute Engine / Cloud Run** effectue des requêtes à la demande sur **BigQuery** pour l'état en temps réel.

Ce diagramme met en évidence l'enchaînement clair des composants et la logique de traitement des données.

2.5. Modélisation des données :



Nom du champ	Type	Mode	Clé	Classement	Valeur par défaut	Tags avec stratégie	Des
capteur_id	STRING	NULLABLE	-	-	-	-	-
temperature	FLOAT	NULLABLE	-	-	-	-	-
humidite	FLOAT	NULLABLE	-	-	-	-	-
vibration	FLOAT	NULLABLE	-	-	-	-	-
timestamp	TIMESTAMP	NULLABLE	-	-	-	-	-
anomalie	BOOLEAN	NULLABLE	-	-	-	-	-
region	STRING	NULLABLE	-	-	-	-	-

Figure 4: modèle de table

Table sensor_readings (BigQuery)

- capteur_id STRING
- timestamp TIMESTAMP (*partitionné*)
- temperature FLOAT
- humidity FLOAT
- Vibration FLOAT
- anomaly BOOLEAN

Bucket iot-archives/

- Structure de répertoires par année/mois/jour pour faciliter la purge automatique.

Cette **conception** sert de socle pour l'**implémentation** : elle formalise les modules, les flux, les scénarios et le modèle de données, garantissant que chaque composant réponde précisément aux besoins métier et techniques du projet.

CHAPITRE IV :

4. Développement des composants :

4.1. Script de simulation de capteurs :

Fichier `sensor_simulator.py`

Ce script génère un message JSON par seconde par capteur virtuel et les publie sur le topic Pub/Sub via MQTT Bridge.

4.2 Cloud Function de traitement :

Fichier `main.py` et `requirements.txt`

Points clés :

- La fonction se déclenche **à chaque message** sur le topic Pub/Sub (`--trigger-topic sensor-data`).
- On stocke la table BigQuery et le bucket Cloud Storage dans des variables d'environnement (`BUCKET`, `BQ_TABLE`) pour plus de flexibilité.
- Le streaming insert permet un délai de latence très faible (< 1 s) pour la donnée fraîche.
- L'archivage brute garantit que l'on dispose toujours du JSON original, pour reprocessing ou audit.
- Les clients `google-cloud-bigquery` et `google-cloud-storage` sont indispensables pour interagir respectivement avec BigQuery et Cloud Storage.
- La Cloud Function s'exécute sur Python 3.9+ (runtime `python39`). GCP déploie automatiquement les bibliothèques de base (comme `google-cloud-pubsub`) pour les triggers Pub/Sub.

Après avoir préparé ces deux fichiers, on peut déployer la fonction avec la commande :

```
gcloud functions deploy process_sensor \
--runtime python39 \
--trigger-topic sensor-data \
--set-env-vars BUCKET=iot-archives,BQ_TABLE=iot-sensor-demo.sensor_readings
```

4.3. Provisionne des services GCP

1. Création du topic Pub/Sub

- `gcloud pubsub topics create sensor-data`

Vérifiez ensuite son existence :

- `gcloud pubsub topics list --filter="name:sensor-data"`

2. Mise en place de BigQuery :

2.1. Créer le dataset :

- `bq mk --dataset iot_sensor_demo`

2.2. Créer la table `sensor_readings` avec partitionnement sur le champ `timestamp` et clustering sur `capteur_id` :

- `bq mk --table iot_sensor_demo.sensor_readings \`

```
capteur_id:STRING,timestamp:TIMESTAMP,temperature:FLOAT, \
humidity:FLOAT,vibration:FLOAT,anomaly:BOOLEAN \
--time_partitioning_field timestamp \
--clustering_fields capteur_id
```

2.3. Vérifiez le schéma et les propriétés :

- `bq show --format=prettyjson iot_sensor_demo.sensor_readings`

3. Configuration du bucket Cloud Storage

1. Créer le bucket d'archives :

- `gsutil mb -l europe-west1 gs://iot-archives`

2. Déployer une règle de cycle de vie (`lifecycle.json`) pour supprimer les objets après 90 jours :

```
{
  "rule": [
    {
      "action": {"type": "Delete"},
      "condition": {"age": 90}
    }
  ]
}
```

3. Vérifiez la configuration :

- `gsutil lifecycle get gs://iot-archives`

4. Configuration des dashboards Looker Studio :

Pour exploiter les données ingérées et analysées dans BigQuery, nous allons créer un rapport interactif dans Looker Studio qui reflète les principaux KPIs et permet aux utilisateurs de filtrer à la volée.

1. Créer une source de données BigQuery

- Dans Looker Studio, cliquez sur **Ajouter une source de données**, choisissez **BigQuery**.
- Sélectionnez votre projet (iot-sensor-demo), puis le dataset `iot_sensor_demo` et la table `sensor_readings`.
- Activez l'option **Requête directe** pour que chaque visualisation utilise les données fraîches (streaming insert).

2. Modéliser les champs

- Vérifiez que `timestamp` est bien de type **Date/Time** et que `anomaly` est un **Boolean**.
- Créez un champ calculé **Date_Jour** :

```
sql
CopyEdit
DATE(timestamp)
```

- (Optionnel) Créez un champ **Heure** :

```
sql
CopyEdit
FORMAT_DATETIME("%H:%M", timestamp)
```

3. Construire le rapport

- **Scorecards** pour les indicateurs clés :
 - Nombre total de lectures
 - Nombre d'anomalies (filtré sur `anomaly = TRUE`)
- **Graphique temporel** (Time series) : évolution de la température moyenne par heure ou par jour.
- **Diagramme à barres** : répartition des anomalies par capteur (`capteur_id`).
- **Carte géographique** (si lat/lon disponibles) : localisation des capteurs anormaux.
- **Filtres et contrôles** :
 - Période (date picker lié à `timestamp`)
 - Liste déroulante `capteur_id`

4. Personnalisation et mise en forme

- Ajoutez un en-tête avec logo de l'application et titre "Dashboard Surveillance IoT".
- Choisissez une palette sobre et assurez-vous que les anomalies ressortent (par exemple via un code couleur + icône).
- Ajoutez des sections « Vue d'ensemble » et « Détails capteur » pour structurer le rapport.

5. Partage et permissions

- Dans **Partager**, attribuez le rôle **Lecteur** aux groupes ou comptes service GCP concernés.
- Activez l'option **Actualisation à l'ouverture** pour garantir la fraîcheur des données.

5. Déploiement de l'API de consultation (Compute Engine / Cloud Run)

Pour permettre une consultation ad-hoc de l'état d'un capteur, on expose une API REST qui interroge BigQuery en temps réel. Deux options sont possibles :

5.1. Cloud Run :

Dans la répertoire `sensorApi/main.py`

1. Import des dépendances

```
python
CopyEdit
from flask import Flask, jsonify
from google.cloud import bigquery
```

- **Flask** pour créer un micro-service HTTP.
- **google.cloud.bigquery** pour interroger BigQuery.

2. Initialisation

```
python
CopyEdit
app = Flask(__name__)
bq = bigquery.Client()
TABLE = "iot-sensor-demo.iot_sensor_demo.sensor_readings"
```

- `app` : instance Flask.
- `bq` : client BigQuery préconfiguré, utilise les identifiants par défaut Google Cloud (compte de service Cloud Run).
- `TABLE` : chemin complet de la table à interroger.

3. Route `/status/<capteur_id>`

```
python
CopyEdit
@app.route("/status/<capteur_id>")
def status(capteur_id):
    query = f"""
    SELECT *
    FROM `{TABLE}`
    WHERE capteur_id = @cid
    ORDER BY timestamp DESC
    LIMIT 1
    """
    job = bq.query(query, job_config=bigquery.QueryJobConfig(
        query_parameters=[bigquery.ScalarQueryParameter("cid", "STRING", capteur_id)]
    ))
    row = next(job.result(), None)
    return jsonify(dict(row)) if row else ("Not found", 404)
```

- **Paramètre URL** `capteur_id` : identifiant du capteur dont on veut l'état.
- **Requête SQL paramétrée** :
 - Filtre sur `capteur_id` pour éviter les injections SQL.
 - Trie par timestamp DESC pour récupérer la dernière mesure.
- **Exécution et récupération** :
 - `job.result()` renvoie un itérateur sur les lignes.
 - `next(..., None)` récupère la première ligne ou None.

- **Réponse HTTP :**
 - Si une ligne existe, on la convertit en dictionnaire puis en JSON.
 - Sinon, on renvoie un code 404 "Not found".

4. Point d'entrée

```
python
CopyEdit
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)
```

- Permet de lancer l'application localement pour des tests.

2. Dockerfile

```
dockerfile
CopyEdit
FROM python:3.9-slim
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY app.py .
CMD ["gunicorn", "-b", "0.0.0.0:8080", "app:app"]
```

1. Image de base

```
dockerfile
CopyEdit
FROM python:3.9-slim
```

- Version allégée de Python 3.9 pour réduire la taille de l'image.

2. Installation des dépendances

```
dockerfile
CopyEdit
COPY requirements.txt .
RUN pip install -r requirements.txt
```

- Copie du fichier listant les bibliothèques (google-cloud-bigquery, Flask, etc.).
- Installation via pip, ce qui garantit que le conteneur aura exactement les mêmes versions.

3. Ajout du code applicatif

```
dockerfile
CopyEdit
COPY app.py .
```

- Place le script Python à la racine du conteneur.

4. Commande de démarrage

```
dockerfile
CopyEdit
CMD ["gunicorn", "-b", "0.0.0.0:8080", "app:app"]
```

- Utilise **Gunicorn**, serveur WSGI performant, pour exposer l'application Flask sur le port 8080.
- app:app fait référence au module app.py et à l'instance app qu'il contient.

5.6 Monitoring et alertes

5. Pour assurer la fiabilité et la réactivité de votre pipeline IoT, il est indispensable de mettre en place une solution de supervision et d'alerte. Google Cloud Monitoring (anciennement Stackdriver) permet de collecter des métriques, de construire des dashboards et de configurer des alertes automatiques.
 - Activer l'API Monitoring :
 - `gcloud services enable monitoring.googleapis.com \`
 - `logging.googleapis.com`
 - Métriques clés à suivre
 - Cloud Functions
 - ☐ Invocation count
 - ☐ Error count / error rate
 - ☐ Execution latency (cold starts)

Pub/Sub

- Subscription/Topic backlog (`num_undelivered_messages`)
- Publish/Receive throughput

BigQuery

- Streaming insert errors
- Slot utilization (si quotas partagés)

Cloud Storage

- Object stored count
- Error rate (HTTP 4xx/5xx)

Compute Engine / Cloud Run

- CPU & memory utilization
- Request count & latency

Toutes ces métriques sont exposées nativement dans Cloud Monitoring une fois les services activés.

5.2. Créez des policy d'alerte pour être notifié avant tout impact utilisateur.

- Alert : erreurs Cloud Functions
- gcloud monitoring policies create \
- --display-name="CF Error Rate" \
- --combiner=OR \
- --conditions='[{
- "conditionThreshold": {
- "filter":
- "metric.type="cloudfunctions.googleapis.com/function/execution_count" AND
- resource.label.function_name="process_sensor",
- "comparison": "COMPARISON_GT",
- "thresholdValue": 5,
- "duration": "60s",
- "trigger": {"count": 1}
- }]
- }]
- gcloud monitoring policies create \
- --display-name="Pub/Sub Backlog" \
- --conditions='[{
- "conditionThreshold": {
- "filter":
- "metric.type="pubsub.googleapis.com/subscription/num_undelivered_messages" AND
- resource.label.subscription_id="sensor-data-sub",
- "comparison": "COMPARISON_GT",
- "thresholdValue": 1000,
- "duration": "300s"
- }
- }]
- //Notifie si plus de 1 000 messages en attente pendant 5 min.
- //////////////////////////////////////
- Alert : latence Cloud Run
- gcloud monitoring policies create \
- --display-name="API Latency" \
- --conditions='[{
- "conditionThreshold": {
- "filter": "metric.type="run.googleapis.com/request_latencies" AND
- resource.label.service_name="sensor-api",
- "comparison": "COMPARISON_GT",
- "thresholdValue": 1.0,
- "duration": "60s"
- }
- }]

5.3 Création d'un dashboard :

Dans Cloud Monitoring, créez un **Dashboard personnalisé** regroupant :

Widget	Source de données
Invocations & erreurs CF	cloudfunctions.googleapis.com/function/execution_count & error_count
Backlog Pub/Sub	pubsub.googleapis.com/subscription/num_undelivered_messages
Latence API (Cloud Run)	run.googleapis.com/request_latencies
Insertion BigQuery (errors)	bigquery.googleapis.com/streaming_insert_errors
Stockage Cloud Storage	storage.googleapis.com/storage/object_count

Figure 5:dashboard et source

Avec ces éléments de **Monitoring et d'Alerting**, vous pouvez anticiper les incidents, suivre la performance et garantir la robustesse de votre pipeline IoT sur GCP.

CHAPITRE V :

Réalisation et résultat :

```
(iot env) C:\Users\ASUS\iot-cloud-project>python sensor_simulator.py --project clever-grammar-458517 --topic iot-data-v4 --sensors 5 --interval 5
2025-05-03T20:44:44Z INFO Régions par capteur : {'sensor-1': 'Sousse', 'sensor-2': 'Gafsa', 'sensor-3': 'Sfax', 'sensor-4': 'Nabeul', 'sensor-5': 'Tunis'}
2025-05-03T20:44:45Z INFO Envoyé : {'capteur_id': 'sensor-1', 'region': 'Sousse', 'temperature': 39.59, 'humidite': 36.98, 'vibration': 0.1, 'timestamp': '2025-05-03T18:44:45.630528+00:00'}
2025-05-03T20:44:45Z INFO Envoyé : {'capteur_id': 'sensor-2', 'region': 'Gafsa', 'temperature': 79.25, 'humidite': 57.27, 'vibration': 0.59, 'timestamp': '2025-05-03T18:44:45.632608+00:00'}
2025-05-03T20:44:45Z INFO Envoyé : {'capteur_id': 'sensor-3', 'region': 'Sfax', 'temperature': 22.54, 'humidite': 34.68, 'vibration': 0.23, 'timestamp': '2025-05-03T18:44:45.632608+00:00'}
2025-05-03T20:44:45Z INFO Envoyé : {'capteur_id': 'sensor-4', 'region': 'Nabeul', 'temperature': 68.16, 'humidite': 58.06, 'vibration': 0.72, 'timestamp': '2025-05-03T18:44:45.633974+00:00'}
2025-05-03T20:44:45Z INFO Envoyé : {'capteur_id': 'sensor-5', 'region': 'Tunis', 'temperature': 76.11, 'humidite': 50.98, 'vibration': 0.45, 'timestamp': '2025-05-03T18:44:45.636304+00:00'}
2025-05-03T20:44:50Z INFO Envoyé : {'capteur_id': 'sensor-1', 'region': 'Sousse', 'temperature': 89.54, 'humidite': 67.94, 'vibration': 0.16, 'timestamp': '2025-05-03T18:44:50.636304+00:00'}
```

Figure 6:script python

Ligne	capteur_id	temperature	humidite	vibration	timestamp	anomalie	region
1501	sensor-4	84.57	56.45	0.67	2025-05-03 13:08:28.398650 UTC	true	Nabeul
1502	sensor-2	39.54	44.25	0.44	2025-05-03 13:08:28.398650 UTC	false	Gafsa
1503	sensor-5	64.37	76.59	0.1	2025-05-03 13:08:28.398650 UTC	false	Tunis
1504	sensor-1	30.6	47.45	0.07	2025-05-03 13:08:28.397365 UTC	false	Sousse
1505	sensor-3	47.43	64.37	0.85	2025-05-03 13:08:33.411008 UTC	false	Sfax
1506	sensor-5	24.06	61.71	0.83	2025-05-03 13:08:33.412201 UTC	false	Tunis
1507	sensor-4	60.43	42.56	0.91	2025-05-03 13:08:33.411692 UTC	false	Nabeul
1508	sensor-2	32.44	44.09	0.21	2025-05-03 13:08:33.409804 UTC	false	Gafsa

Figure 7:big query data

Google Cloud	iot-sensor	cloud storage	Recherche	Accéder au chemin	Actualiser	Apprendre
Informations sur le bucket						
run-sources-clever-grammar-458517-europe-west1						
services/						
sensor-api/						
Filtrer par préfixe de nom uniquement						
Filtrer						
Filtrer les objets et dossiers						
Afficher Objets actifs uniquement						
Nom						
Taille						
Type						
Création						
Classe de stockage						
1746283923.066486-4d6a7e138d...						
1746284793.014959-be3fcf71ba2...						
1746285184.85352-41006cb8d60...						
1746285393.171397-723e2766f3f...						
1746285612.019953-960d1c64f8...						
1746286014.290813-2c08535fac...						
1746286362.232348-01caa3b1b1...						
1746286575.547476-840087d46d...						
1746286790.592737-b3aa5ae129...						
1746287223.837892-984908da58...						
1746287595.959726-5eb3fdc5cab...						
1746287907.610454-09df007c36...						
1746291535.996303-45c38b2184...						
1746291778.330545-31c073b3dd...						

Figure 8:cloud storage

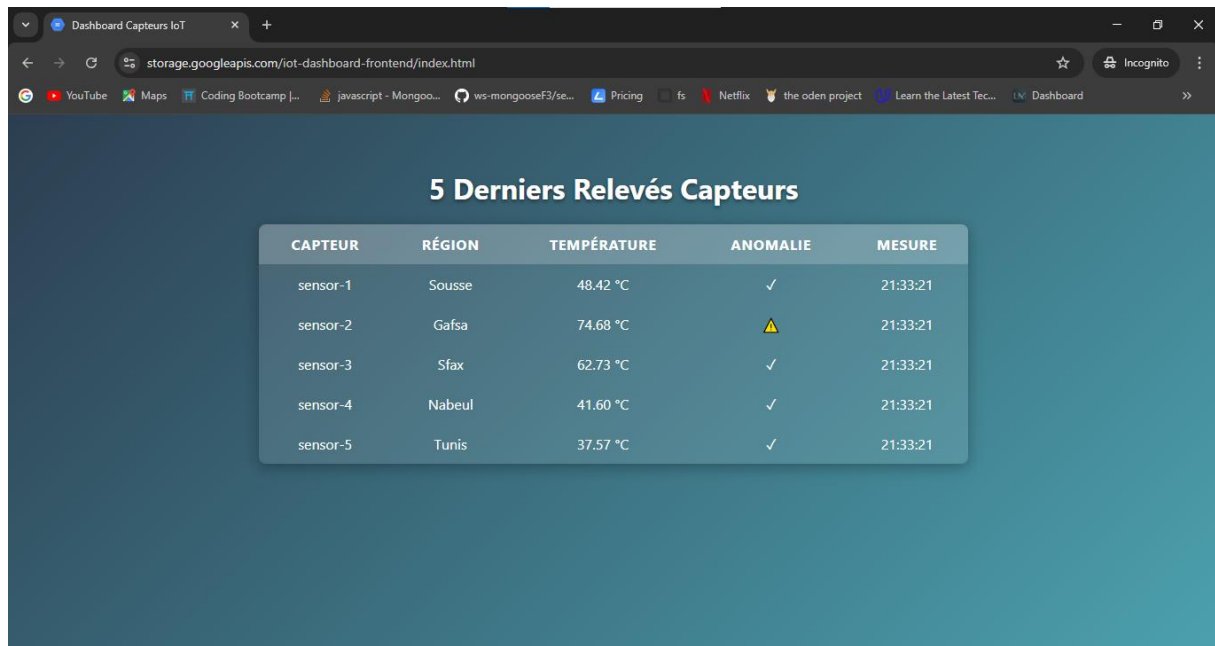


Figure 9:cloud run hosting

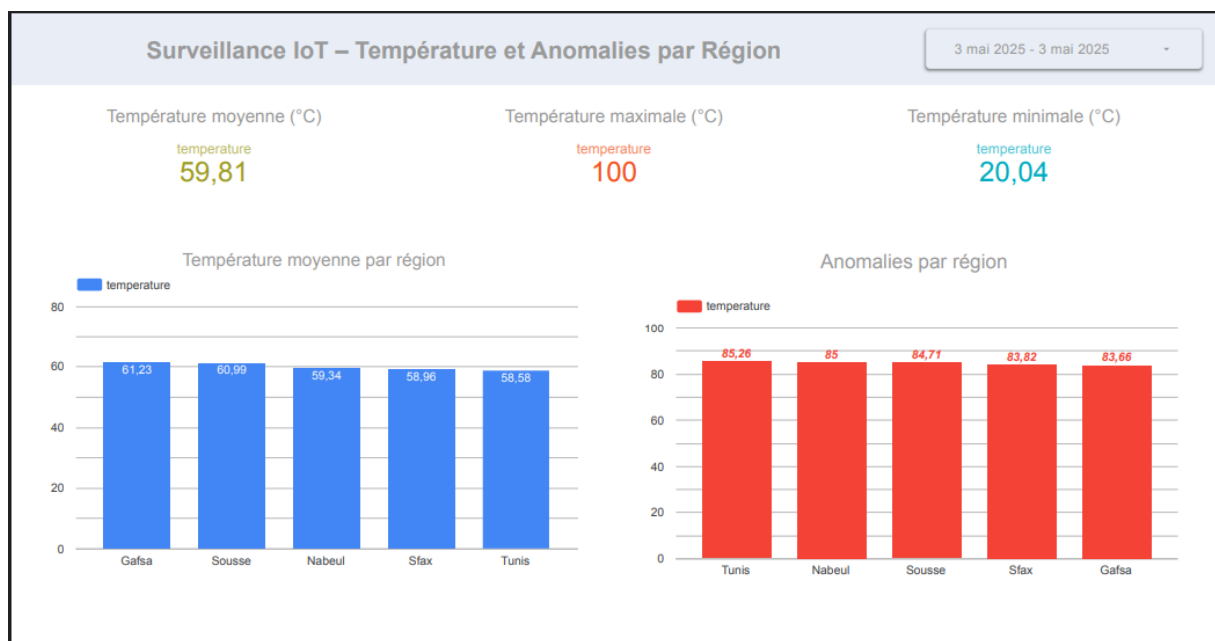


Figure 10:looker rapport

CHAPITRE V :

Conclusion et perspectives :

Ce projet a permis de concevoir et de déployer une architecture complète de surveillance et d'analyse de capteurs embarqués en s'appuyant sur les services managés de Google Cloud Platform. En combinant la puissance de Pub/Sub pour l'ingestion, Cloud Functions pour le traitement en temps réel, BigQuery pour l'analyse et Looker Studio pour la visualisation, nous avons obtenu un pipeline évolutif, fiable et à faible latence, sans avoir à gérer l'infrastructure sous-jacente.

La solution mise en place répond aux principaux besoins des systèmes embarqués : robustesse, rapidité de traitement, évolutivité et visualisation en temps réel. L'approche serverless adoptée permet également de maîtriser les coûts tout en assurant une haute disponibilité.

Ce projet ouvre la voie à de nombreuses perspectives d'amélioration, notamment l'intégration de l'intelligence artificielle pour prédire les pannes ou comportements anormaux, l'extension vers des capteurs physiques connectés, ou encore le déploiement d'interfaces utilisateur plus conviviales.

En résumé, cette étude montre que le cloud — et en particulier GCP — constitue une base solide et moderne pour la construction de solutions IoT industrielles, intelligentes et durables.