

## Partie 2 : Conception d'une chaîne de traitement des données IoT

---

Dr Brice Ekane - (brice.ekane@univ-rennes.fr)

ISTIC Rennes - France

2024-2025

1 Introduction

2 Rappels Réseau

3 Plateformes IoT

4 Topologies de déploiement

5 API REST et HTTP

6 Communication MQTT

7 Stockage et Analyse

8 Mise en oeuvre

9 Conclusion

# Objectifs pédagogiques

---

À la fin de cette partie, vous serez capable de :

- ▶ Situer une chaîne IoT complète dans une architecture réseau nouvelle génération.
- ▶ Identifier les rôles clés du middleware et des plateformes de données dans cette chaîne.
- ▶ Mettre en œuvre un pipeline MQTT → InfluxDB minimal illustrant le fil rouge.

1 Introduction

2 Rappels Réseau

3 Plateformes IoT

4 Topologies de déploiement

5 API REST et HTTP

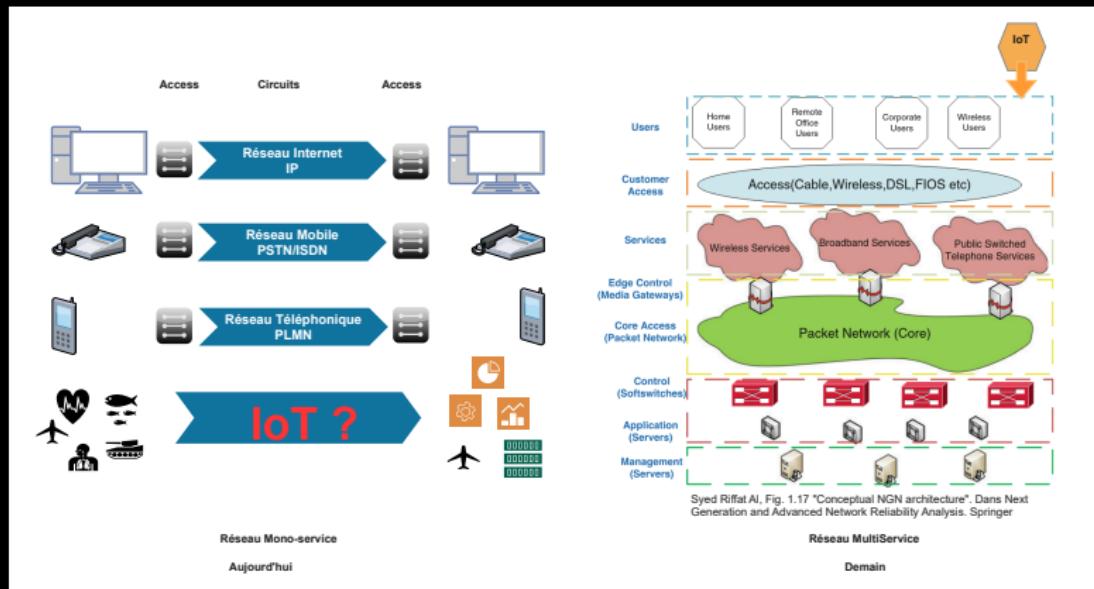
6 Communication MQTT

7 Stockage et Analyse

8 Mise en oeuvre

9 Conclusion

# Le contexte



# Problème

---

Comment s'intègre l'IoT dans une architecture réseau nouvelle génération ?

# Rappel!

## Architecture Réseau

### ① Services :

- ▶ *Ce qui est partagé* : Documents, bases de données, contenu multimédia, messagerie et outils collaboratifs (par exemple, Zoom).

### ② Infrastructure :

- ▶ *Voies physiques* : Médias comme les câbles à paires torsadées, câbles coaxiaux, fibres optiques ou options sans fil (WiFi, cellulaire, satellite).
- ▶ *Dispositifs permettant ces voies* : Routeurs, commutateurs (switches) et points d'accès qui facilitent la communication.

# Rappel!

## Architecture Réseau

### ③ Protocoles :

- ▶ *Règles pour gérer le flux de données* : Fournissent un « langage » commun pour permettre aux appareils de communiquer.
- ▶ *Politiques de sécurité* : Assurent la protection et la transmission sécurisée des données sur le réseau.

### ④ Hôtes :

- ▶ *Dispositifs fournissant ou utilisant les services* :
  - ▶ **Modèle Serveur/Client** : Les serveurs fournissent des services réseau ; les clients les consomment.
  - ▶ **Modèle Pair-à-Pair** : Les hôtes agissent à la fois comme serveurs et comme clients simultanément.

# Concepts IoT Fondamentaux

---

## L'IoT : Définition et Rôles

- ▶ **Capteurs** : Collectent les données (ex : température).
- ▶ **Passerelles** : Relient les capteurs au réseau global.
- ▶ **Réseaux** : Transportent les données (Wi-Fi, LoRaWAN, etc.).
- ▶ **Plateformes** : Traitent, stockent et visualisent les données.

# Lien avec la chaîne IoT

---

## Pourquoi ces rappels comptent ?

- ▶ Les services et applications orientent les cas d'usage IoT qui seront illustrés dans le fil rouge.
- ▶ L'infrastructure et les protocoles guident les choix de connectivité (MQTT, LoRaWAN, REST) à chaque bloc fonctionnel.
- ▶ La connaissance des hôtes et de leur rôle facilite la distinction entre dispositifs, passerelles, plateformes et applications finales.

Zoom sur les plateformes middleware IoT qui orchestrent ces éléments.

- 1 Introduction
- 2 Rappels Réseau
- 3 Plateformes IoT
- 4 Topologies de déploiement
- 5 API REST et HTTP
- 6 Communication MQTT
- 7 Stockage et Analyse
- 8 Mise en oeuvre
- 9 Conclusion

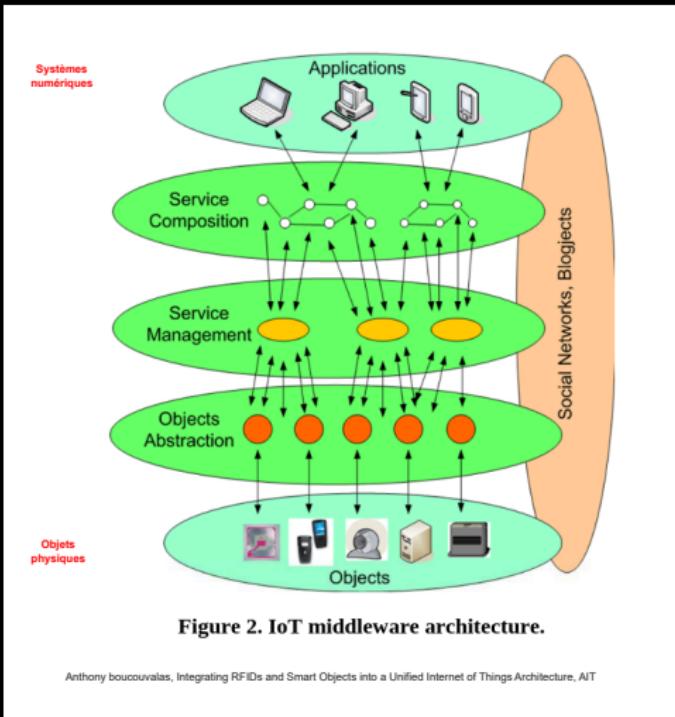
# Notre Focus

---

## Focus sur les Plateformes Middleware IoT

- ▶ **Middleware** : Pont entre objets physiques et systèmes numériques.
  - ▶ Mélange de logique haut et bas niveau, traitant protocoles et communications.
- ▶ Objectif : Rapidité de mise sur le marché, scalabilité, et coût réduit.

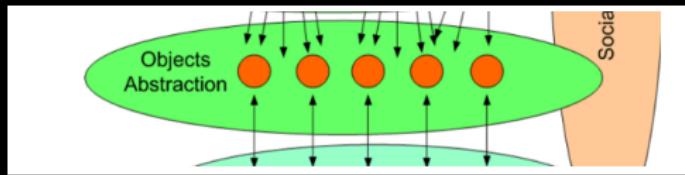
# Notre Focus



# Types de Plateformes IoT

## 1. Plateformes de Service Réseau

- ▶ Gèrent la communication au niveau MAC (ex : LoRaWAN).
- ▶ Convertissent les communications radio en données exploitables.
- ▶ Incluent les services de gestion d'identité et des clés.

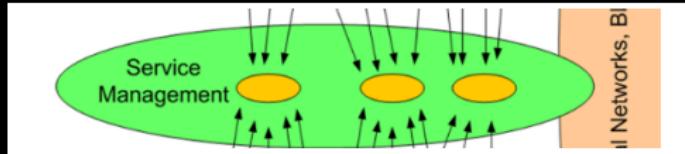


**Figure:** Adapted from : Anthony boucouvalas, *Integrating RFIDs and Smart Objects into a Unified Internet of Things Architecture*, AIT

# Types de Plateformes IoT (suite)

## 2. Plateformes Middleware (Pont Réseau-Application)

- ▶ Traitent les données après le réseau et avant les applications (ex : Chirpstack)..
- ▶ Gèrent le décodage des protocoles, l'orchestration des communications.
- ▶ Constituent la "plomberie" essentielle du système IoT.

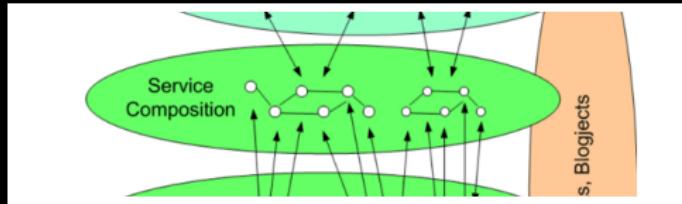


**Figure:** Adapted from : Anthony boucouvalas, *Integrating RFIDs and Smart Objects into a Unified Internet of Things Architecture*, AIT

# Types de Plateformes IoT (suite)

## 3. Plateformes pour le Développement Applicatif

- ▶ Permettent le développement de haut niveau sur le cloud ((ex : Node-Red).).
- ▶ Connectent les middleware aux applications comme ERP et CRM.
- ▶ Moins critiques pour le fonctionnement global en cas de panne.



**Figure:** Adapted from : Anthony boucouvalas, Integrating RFIDs and Smart Objects into a Unified Internet of Things Architecture, AIT

# Types de Plateformes IoT (suite)

## Caractéristiques d'un Bon Middleware IoT

- ▶ Supporte l'intégration des objets hétérogènes.
- ▶ Offre scalabilité, fiabilité, et efficacité économique.
- ▶ Facilite l'interaction entre logiciels haut niveau et protocoles bas niveau.

# Synthèse des blocs fonctionnels

## Ce qu'il faut retenir avant la mise en œuvre

- ▶ Les dispositifs collectent les données, les passerelles les remontent vers le broker MQTT et le middleware.
- ▶ Le middleware orchestre routage, règles et exposition d'API pour alimenter stockage et applications métiers.
- ▶ Les choix de protocoles et de bases (MQTT, InfluxDB) répondent aux contraintes de latence, volume et intégration.

Les sections suivantes exploitent ces blocs pour construire concrètement une chaîne MQTT → InfluxDB.

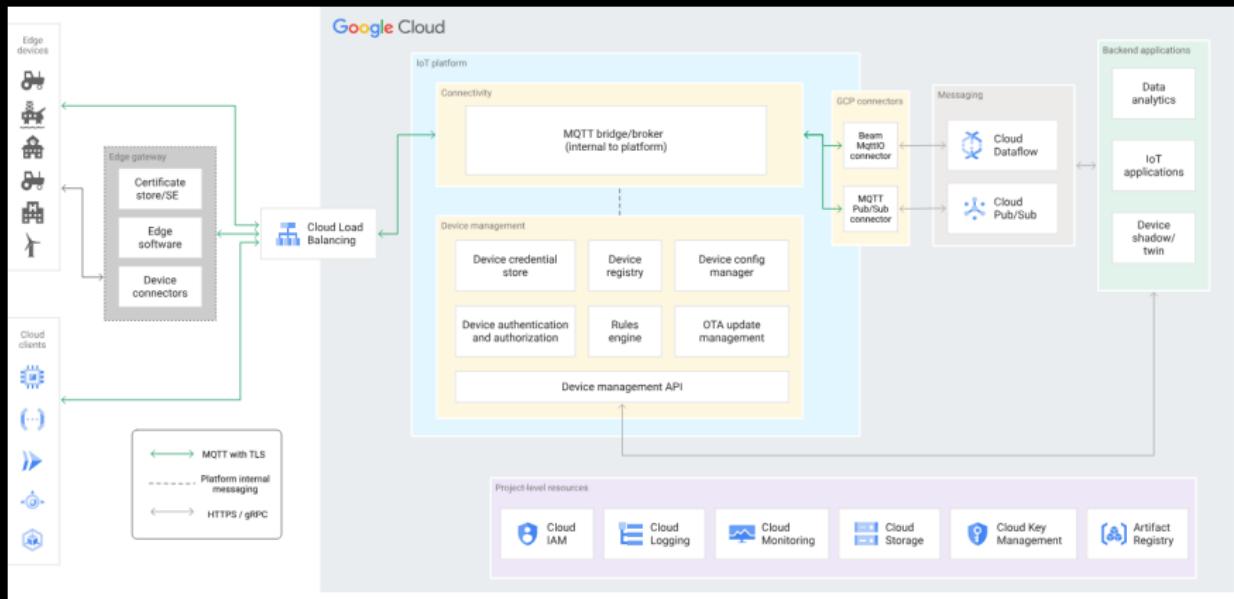
# Cas d'étude fil rouge

## Scénario : suivi de température d'entrepôt

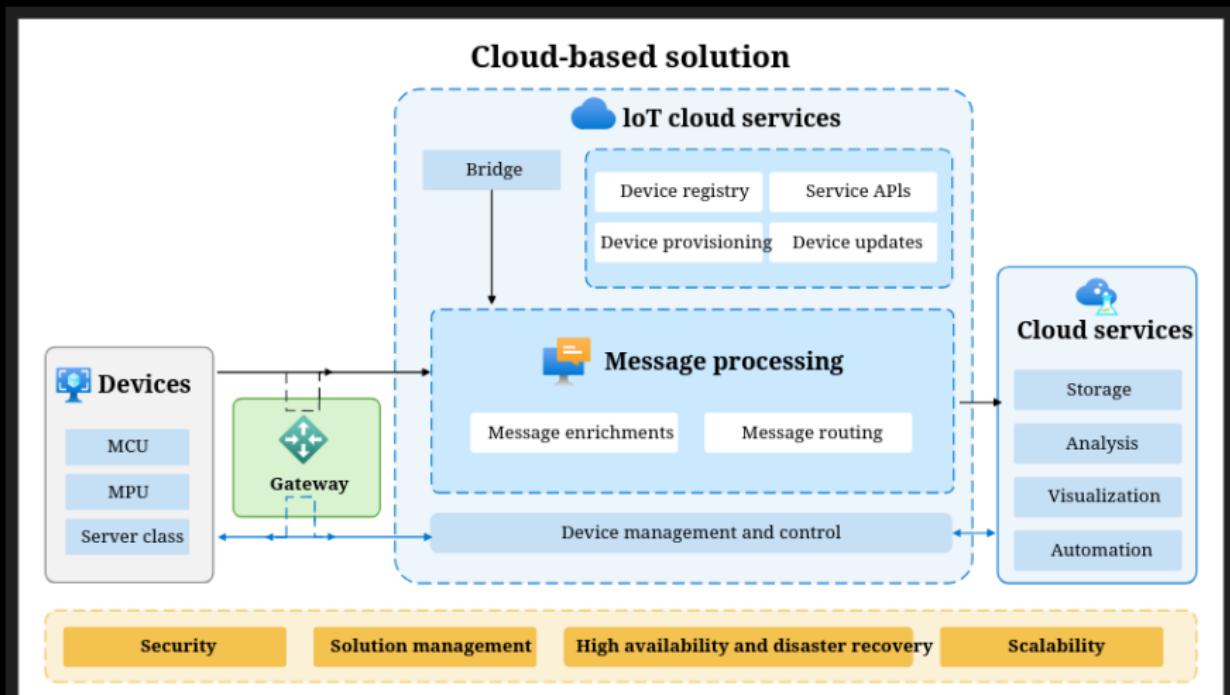
- ▶ Plusieurs capteurs publient température et humidité via MQTT à travers une passerelle locale.
- ▶ La plateforme middleware ingère, nettoie et aiguillage les données vers InfluxDB pour stockage temporel.
- ▶ Les applications de supervision exploitent les API pour déclencher des alertes ou afficher des tableaux de bord Grafana.

Chaque composant étudié ultérieurement sera relié explicitement à ce scénario pour garder le fil conducteur.

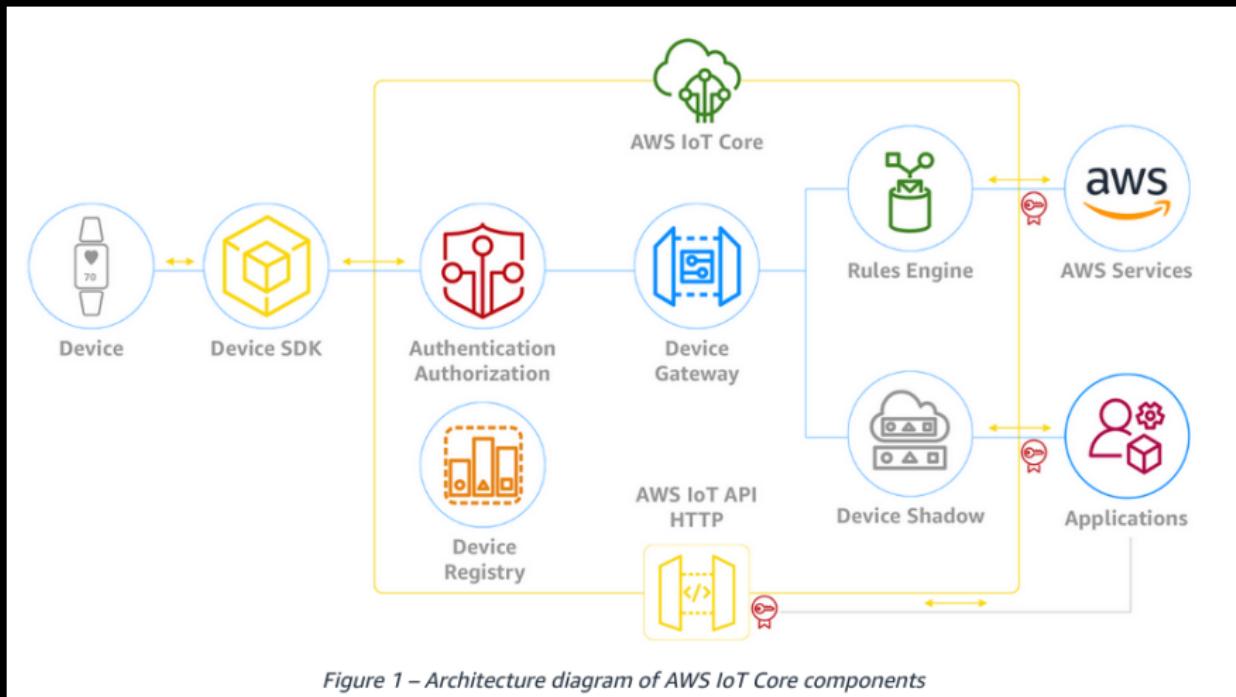
# Google IoT Platform



# Azure IoT Platform



# Aws IoT Platform



# Pourquoi construire votre propre plate-forme IoT ?

---

## Défis des plateformes existantes

- ▶ Dépendance aux plateformes externes limite l'évolution.
- ▶ Les solutions freemium ou d'essai mènent souvent à un verrouillage fournisseur.
- ▶ Les coûts augmentent de façon exponentielle à mesure que l'échelle croît.

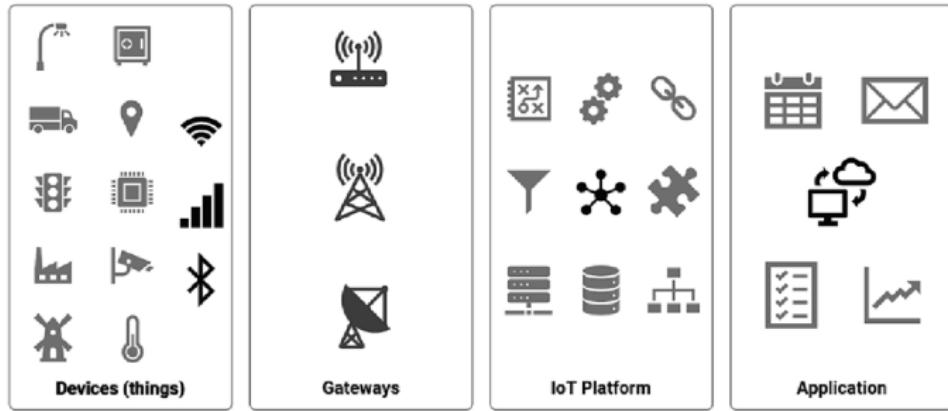
# Pourquoi construire votre propre plate-forme IoT ?(suite)

---

## Avantages de construire sa propre plateforme

- ▶ **Efficacité économique** : Commencez modestement et investissez selon les besoins.
- ▶ **Lancement rapide** : Déploiement possible en moins de 24 heures.
- ▶ **Flexibilité** : Contrôle total sur les fonctionnalités et l'évolution.
- ▶ **Apprentissage** : Une expérience enrichissante et formatrice.

# Block fonctionnels



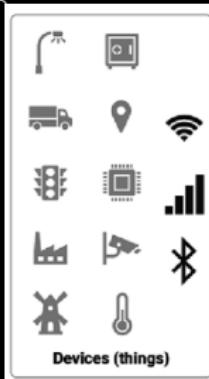
*Figure 2-1. Functional blocks of an IoT solution*

Anand Tamboli, *Build Your Own IoT Platform – Develop a Flexible and Scalable Internet of Things Platform*, Apress

# Les Blocs Fonctionnels d'une Solution IoT

## 1. Dispositifs (Devices)

- ▶ *Rôle* : Capteurs et actionneurs mesurant divers paramètres et les traduisant en données numériques.
- ▶ *Types* :
  - ▶ **Legacy** : Connectés aux dispositifs hôtes (mise à jour des systèmes existants).
  - ▶ **Modernes** : Intégrés dans les dispositifs hôtes.
- ▶ *Exemples* : Thermostats, réfrigérateurs connectés, pièges à souris intelligents.



# Les Blocs Fonctionnels d'une Solution IoT (suite)

## 2. Passerelles (Gateways)

- ▶ *Rôle* : Connectent les dispositifs en aval aux systèmes en amont.
- ▶ *Modes de communication* :
  - ▶ Directement via des protocoles IP (**REST, MQTT, CoAP**).
  - ▶ Indirectement via une passerelle physique avec des technologies duales (ex : LoRaWAN + Ethernet).
- ▶ *Fonctions supplémentaires* : Agrégation, nettoyage des données, déduplication, edge computing.
- ▶ *Exemples* : GSM + RF, Wi-Fi + Bluetooth, LoRaWAN + Ethernet.



Gateways

# Les Blocs Fonctionnels d'une Solution IoT (suite)

## 3. Plateformes IoT

- ▶ *Rôle* : Orchestration globale de la solution IoT (souvent dans le cloud).
- ▶ *Fonctions principales* :
  - ▶ Communication avec les dispositifs en aval.
  - ▶ Stockage des données (séries temporelles, format structuré).
  - ▶ Analyse avancée et traitement des données (selon la sophistication).



# Les Blocs Fonctionnels d'une Solution IoT (suite)

## 4. Applications

- ▶ *Rôle* : Interface utilisateur finale, enrichissant et présentant les données de manière utilisable.
- ▶ *Types* :
  - ▶ Basées sur le bureau, mobiles, ou les deux.
- ▶ *Exemples* : Suivi des stocks via une application mobile intégrée à un système ERP.



# Introduction : Architecture d'une Plate-forme IoT

---

## Composants Essentiels d'une Plateforme IoT

- ▶ Interface avec les dispositifs et Broker de messages
- ▶ Routeur de messages
- ▶ Stockage en série temporelle
- ▶ Moteur de règles
- ▶ Interface API REST

# Interface avec les dispositifs et Broker de messages

## Fonctionnalités

- ▶ Gère les communications avec des dispositifs hétérogènes (par ex. : Wi-Fi, LoRaWAN, Bluetooth).
- ▶ Utilise des modules complémentaires pour chaque protocole de communication.
- ▶ Consolide toutes les données dans un bus de messages unifié.

## Exemples

- ▶ API REST pour les dispositifs compatibles Wi-Fi.
- ▶ Broker MQTT pour les communications pub/sub.
- ▶ Décodeur LoRaWAN pour les serveurs réseau.

# Routeur de messages et Gestion des communications

---

## Fonctions principales

- ▶ Enrichit les données avec du contexte supplémentaire ou réalise des conversions de format.
- ▶ Effectue la déduplication pour supprimer les paquets de données redondants.
- ▶ Achemine efficacement les messages pour un traitement ultérieur.

## Tâches supplémentaires

- ▶ Conversion de formats (par ex. : CSV vers JSON).
- ▶ Interaction avec le moteur de règles pour des déclenchements automatisés.

# Stockage en série temporelle et Moteur de règles

---

## Stockage en série temporelle

- ▶ Stocke les données reçues dans un format séquentiel pour un usage temporaire.
- ▶ Permet un accès à court terme pour les brokers et routeurs.

## Moteur de règles

- ▶ Surveille le bus de messages pour des événements spécifiques.
- ▶ Automatise les actions basées sur des règles (par ex. : déclencher des alertes, rediffuser des messages).
- ▶ Interagit avec d'autres modules pour une exécution fluide.

# API REST et Microservices

## API REST

- ▶ Fournit un accès pour les applications montantes et dispositifs descendants.
- ▶ Soutient les opérations sur des ensembles de données volumineux et les requêtes système.
- ▶ Garantit une authentification basée sur les rôles pour un accès sécurisé.

## Microservices

- ▶ Regroupe des fonctionnalités auxiliaires comme les notifications ou l'intégration de paiements.
- ▶ Découpe les services pour les réutiliser au sein ou en dehors de la plateforme.

# Gestion des dispositifs et Gestion des utilisateurs/applications

---

## Gestion des dispositifs

- ▶ Gère l'état des dispositifs, les mises à jour et les contrôles d'accès.
- ▶ Fournit une surveillance centralisée pour les déploiements à grande échelle.

## Gestion des utilisateurs/applications

- ▶ Gère l'authentification des utilisateurs, les clés d'accès et les permissions.
- ▶ Assure une intégration fluide avec les applications montantes et les systèmes.

# Approche de développement proposée

## Étapes pour construire la plateforme IoT

- ① Configurer l'environnement cloud/onpremise et le système d'exploitation.
- ② Développer le module d'interface et de broker de messages.
- ③ Mettre en place le stockage en série temporelle et les fonctionnalités API REST.
- ④ Ajouter le moteur de règles et le routeur de messages pour des automatisations avancées.
- ⑤ Implémenter la gestion des dispositifs et des utilisateurs/applications .

- 1 Introduction
- 2 Rappels Réseau
- 3 Plateformes IoT
- 4 Topologies de déploiement
- 5 API REST et HTTP
- 6 Communication MQTT
- 7 Stockage et Analyse
- 8 Mise en oeuvre
- 9 Conclusion

# Topologie d'interconection

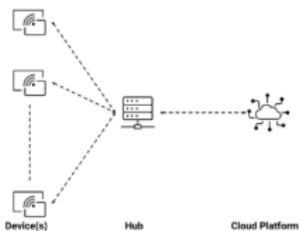


Figure 12-2. Device to the hub to the cloud platform arrangement

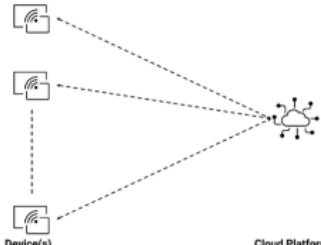


Figure 12-3. Device to the cloud platform direct connectivity

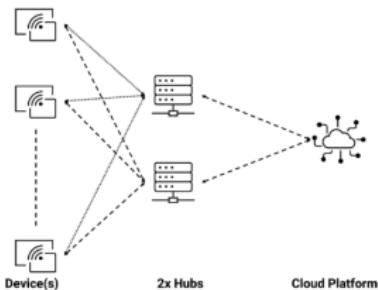


Figure 12-4. Device to multiple hubs to the cloud platform

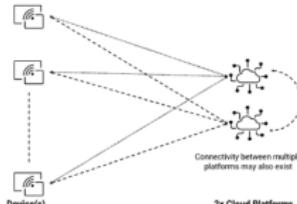
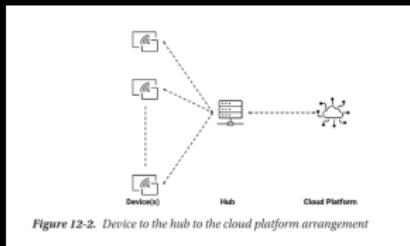


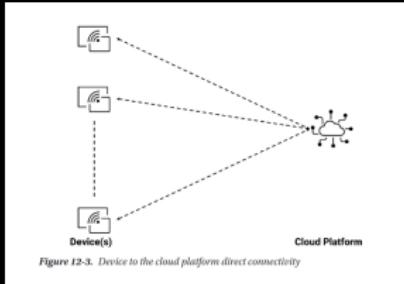
Figure 12-5. Device to multiple cloud platforms direct connectivity

# Topologie d'interconnection



## Device-Hub-Cloud

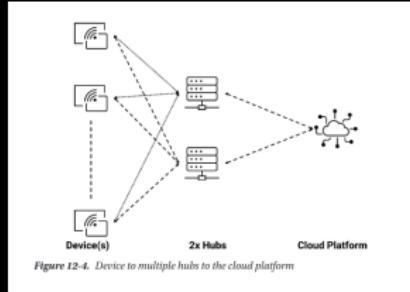
- ▶ *Description* : Les appareils se connectent à un hub local, qui agrège les données et les transmet au cloud.
- ▶ *Avantages* : Arrangement familier et courant.
- ▶ *Inconvénients* : Complexité ajoutée par la gestion du hub.



## Device-Cloud

- ▶ *Description* : Les appareils se connectent directement au cloud via Wi-Fi ou une technologie similaire.
- ▶ *Avantages* : Chaîne de communication simplifiée avec moins de points de défaillance.
- ▶ *Recommandation* : Option privilégiée pour sa simplicité et sa fiabilité.

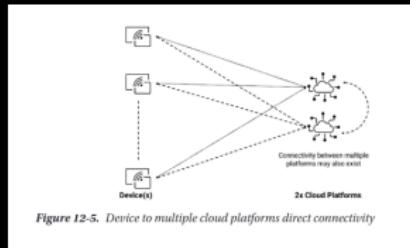
# Topologie d'interconnection



## Device-Multiple Hubs-Cloud

- ▶ *Description* : Les appareils utilisent plusieurs hubs pour la redondance et transmettent les données au cloud.
- ▶ *Avantages* : Tolérance aux pannes grâce à la redondance des hubs.
- ▶ *Cas d'usage* : Systèmes nécessitant une redondance côté appareil.

# Topologie d'interconnection



## Device-Multiple Clouds

- *Description* : Les appareils utilisent plusieurs plateformes cloud, souvent pour séparer la configuration et les opérations courantes.
- *Avantages* : Redondance et séparation des responsabilités côté cloud.
- *Cas d'usage* : Nécessaire pour des scénarios de haute disponibilité.

# Activité : topologie vs scénario

---

## Consigne (5 minutes)

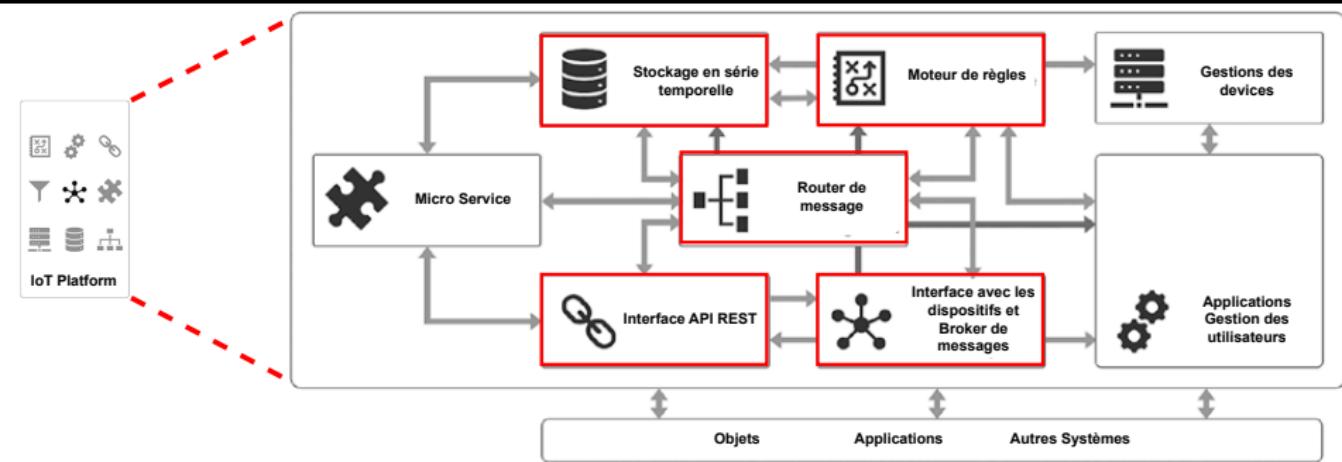
- ▶ En groupe, associez chaque topologie vue (Device-Hub-Cloud, Device-Cloud, Device-Multiple Clouds) à un scénario concret du fil rouge ou de votre expérience.
- ▶ Justifiez le choix en mentionnant les contraintes clés : latency, coût, résilience, gestion locale.

## Mise en commun

---

Chaque groupe partage un cas et une limite identifiée.

# Zoom dans la platform IoT



Adapted From: Build Your Own IoT Platform, Figure 2-1

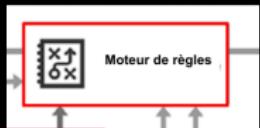
# Stockage en série temporelle



## Caractéristiques

- ▶ Indexation par timestamp, rétention configurable.
- ▶ Agrégations (moyenne, min/max, rolling windows).
- ▶ Compatible avec des TSDB : InfluxDB, TimescaleDB,...etc

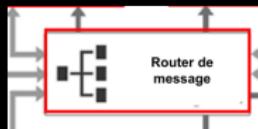
# Moteur de règles



## Caractéristiques

- ▶ Filtrage, transformation, détection d'anomalies.
- ▶ Actions : alertes, commandes vers devices, appels API.
- ▶ Support d'expression : SQL temps-réel, DSL, scripts.

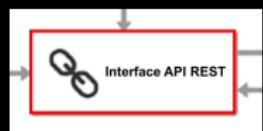
# Router de message



## Caractéristiques

- ▶ Découplage producteur/consommateur.
- ▶ Multiples protocoles : MQTT, AMQP, HTTP, WebSocket.
- ▶ Routage basé sur topics, règles, filtrage ou métadonnées.

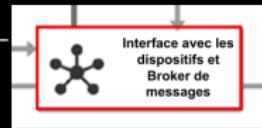
# Interface API REST



## Caractéristiques

- ▶ Exposition de données IoT (mesures, états, historiques).
- ▶ Administration : devices, utilisateurs, règles, dashboards.
- ▶ Sécurisation : authentification, rôles, rate limiting.

# Interface avec les dispositifs et Broker de messages



## Caractéristiques

- ▶ Protocoles supportés : MQTT, CoAP, LoRaWAN, HTTP.
- ▶ Gestion des connexions, authentification device, certificats.
- ▶ Garanties de livraison (QoS), rétention, sessions persistantes.

# Implémentation la platform IoT

## Plusieurs choix:

- ▶ Approche low-code(TP2):
  - ▶ Avec **Node-Red** , Mosquitto, InfluxDB et Grafana
- ▶ Approche avec des frameworks(Démo flask, mosquitto):
  - ▶ Avec Python (Flask), Mosquitto, Sqlite et une interface web simple (Reactjs, Vue.js)
- ▶ à partir de zéro:
  - ▶ bon courage !

# Node-Red: en bref

---

## Qu'est-ce que Node-Red ?

- ▶ *Description* : Node-Red est un outil de développement open-source basé sur une interface graphique, conçu pour connecter des dispositifs, des API et des services en utilisant des flux.
- ▶ *Caractéristiques principales* :
  - ▶ Interface glisser-déposer pour concevoir des flux.
  - ▶ Intégration facile avec IoT, API REST, et bases de données.
  - ▶ Basé sur Node.js, offrant une flexibilité et extensibilité importantes.
  - ▶ Une communauté riche avec de nombreux modules disponibles.

# Node-Red: en bref

---

## Qu'est-ce que Node-Red ?

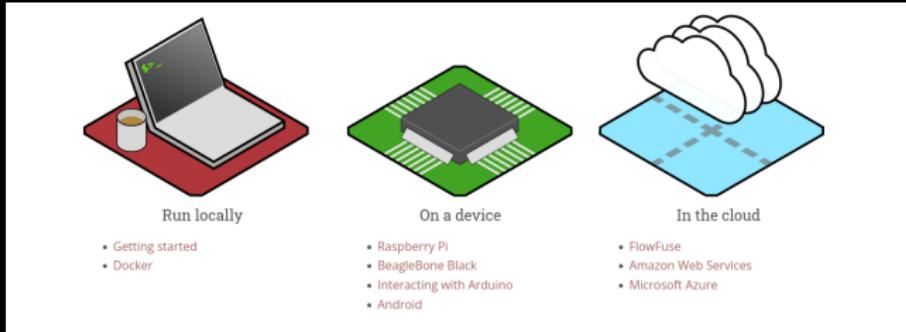
---

### ► *Domaines d'application :*

- ▶ Automatisation industrielle.
- ▶ Applications IoT et domotique.
- ▶ Orchestration d'API et transformation de données.

# Node-Red: en bref

## Large spectre de déploiement



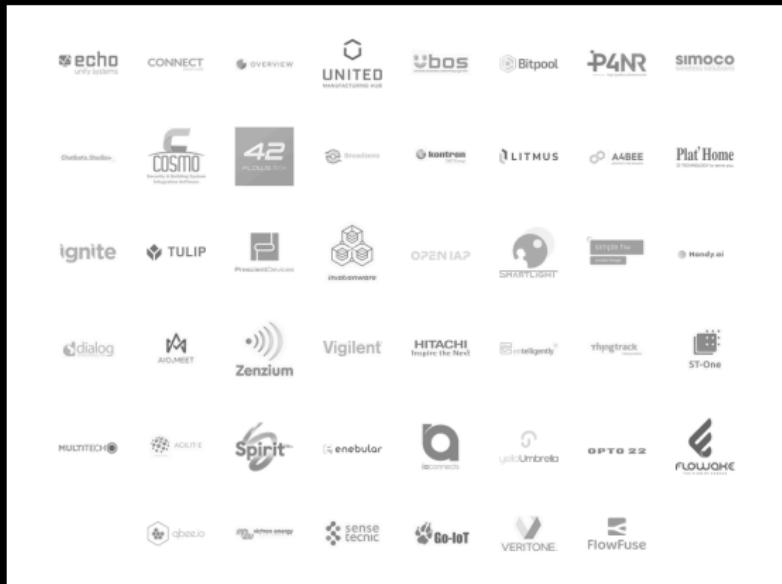
source: <https://nodered.org/#get-started>

 <p><b>Running locally</b> Installing Node-RED on your local computer</p>	 <p><b>Raspberry Pi</b> Get started using our all-in-one install script for the mighty Raspberry Pi</p>	 <p><b>Docker</b> Running Node-RED using Docker</p>
 <p><b>Install from git</b> Building Node-RED from source. Get the very latest development code and start contributing.</p>	 <p><b>BeagleBone Boards</b> Running Node-RED on BeagleBone boards</p>	 <p><b>Android</b> A bit experimental, but you can run on Android devices using Termux</p>
 <p><b>FlowFuse</b> Running a multi-tenant Node-RED solution</p>	 <p><b>AWS</b> Get started running on Elastic Beanstalk or EC2</p>	 <p><b>Microsoft Azure</b> Running on an Azure Virtual Machine Instance</p>

source: <https://nodered.org/#get-started>

# Node-Red: en bref

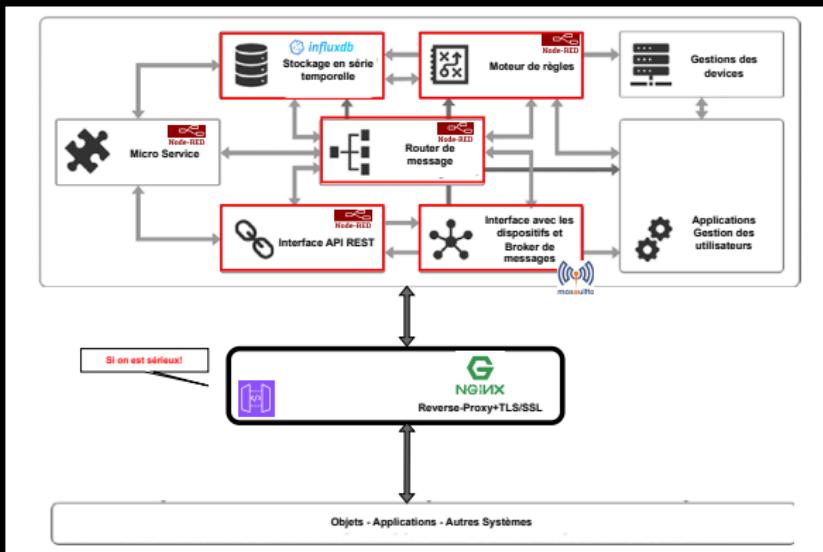
Utilisé par beaucoup d'entreprises



source <https://nodered.org/#get-started>

Brice Ekané - (brice.ekane@univ-rennes.fr)

# Résumons: liste des courses



# Qu'est-ce qu'un Reverse Proxy ?

## Définition

- ▶ *Description* : Un reverse proxy est un serveur intermédiaire placé devant un ou plusieurs serveurs backend pour gérer les requêtes des clients.
- ▶ *Objectifs* :
  - ▶ Distribuer les requêtes aux serveurs backend.
  - ▶ Améliorer les performances grâce à la mise en cache.
  - ▶ Renforcer la sécurité en cachant les serveurs backend.
  - ▶ Permettre le Load Balancing (répartition de charge).

# Exemple de configuration NGINX (proxy http)

```
# Installation : sudo apt install nginx
server {
    listen 80;
    server_name example.com;

    location / {
        proxy_pass http://backend_server;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

## Explications

- ▶ proxy\_pass : Redirige les requêtes vers le serveur backend.
- ▶ proxy\_set\_header : Transmet les en-têtes du client au backend.
- ▶ server\_name : Définit le domaine géré par le reverse proxy.

# Exemple de configuration NGINX (proxy tcp)

```
stream {
    upstream mqtt_brokers {
        # Define your MQTT brokers
        server 192.168.1.101:1883;    # First broker
        server 192.168.1.102:1883;    # Second broker
    }

    server {
        listen 1883;    # Port for MQTT clients to connect to
        proxy_pass mqtt_brokers;    # Forward traffic to the upstream block
        proxy_timeout 60s;    # Timeout for established connections
        proxy_connect_timeout 5s;    # Timeout for establishing connections
    }
}
```

## Explications

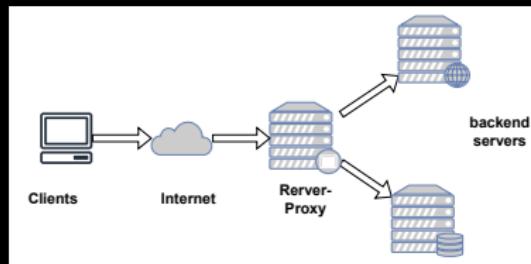
- ▶ `upstream mqtt_brokers` : liste des serveurs MQTT

# Mise en oeuvre du Reverse Proxy avec Nginx

---

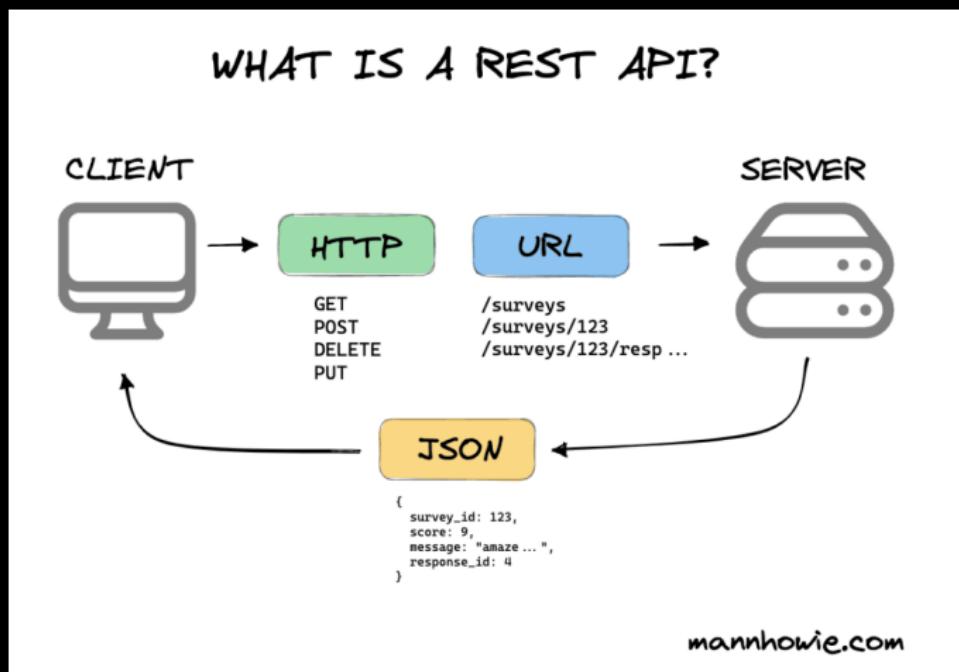
## Étapes de déploiement

- ▶ Installer NGINX sur un serveur dédié.
- ▶ Configurer le fichier nginx.conf.
- ▶ Associer les noms de domaine ou IP à l'adresse du reverse proxy.
- ▶ Tester la configuration avec nginx -t.
- ▶ Redémarrer le service NGINX : sudo systemctl restart nginx.



- 1 Introduction
- 2 Rappels Réseau
- 3 Plateformes IoT
- 4 Topologies de déploiement
- 5 API REST et HTTP
- 6 Communication MQTT
- 7 Stockage et Analyse
- 8 Mise en oeuvre
- 9 Conclusion

# Une API REST: en bref



source: <https://topexamples.fr/api-rest-exemple/>

# Une API REST: en bref

---

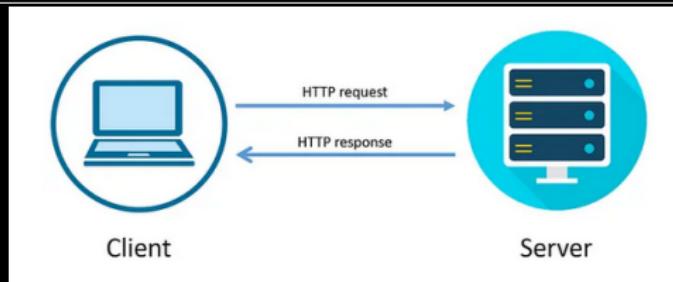
## API REST : Concepts Clés

- ▶ *Description* : Un style architectural basé sur HTTP permettant l'interaction entre systèmes.
- ▶ *Opérations* : Méthodes principales :
  - ▶ GET : Récupérer des données.
  - ▶ POST : Ajouter des données.
  - ▶ PUT : Modifier des données.
  - ▶ DELETE : Supprimer des données.

# Requête HTTP (Request)

## Définition

Message envoyé par le client au serveur pour demander une action.



# Requête HTTP (Request)

## Composants principaux

- ▶ **Méthode** : GET, POST, PUT, DELETE...
- ▶ **URL** : Ressource ciblée (ex.: /api/users/12)
- ▶ **Headers** : Métadonnées (ex.: Content-Type: application/json)
- ▶ **Body (optionnel)** : Données envoyées (souvent en JSON)

# Requête HTTP (Request)

---

## Exemple

---

```
GET /api/users/12 HTTP/1.1
Host: localhost
Accept: application/json
```

# Réponse HTTP Formats

---

## Réponse HTTP

► **Code de statut :**

- ▶ 200 OK — Succès
- ▶ 201 Created — Ressource créée
- ▶ 400 Bad Request
- ▶ 404 Not Found
- ▶ 500 Internal Server Error

► **Headers** : ex. Content-Type: application/json

► **Body** : Données retournées (souvent en JSON)

# Réponse HTTP Formats

---

## Exemple

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": 12,
  "name": "Alice"
}
```

## Formats d'échange

JSON, XML, Form-Data/Multipart, Texte/HTML

# Activité éclair : diagnostic HTTP

## Consigne (3 minutes)

- ▶ En binôme, analysez le scénario suivant : le capteur de température retourne 405 Method Not Allowed lors d'un envoi vers '/temp'.
- ▶ Identifiez deux causes possibles et proposez la correction côté client ou côté API.

## Partage rapide

Mise en commun des hypothèses puis lien avec les bonnes pratiques REST (méthodes, authentification, validation).

# Erreur #1 — Mauvaise méthode HTTP

## Symptôme côté client

Le capteur publie une requête avec la mauvaise méthode (POST au lieu de GET, ou inversement), ce qui est rejeté par l'API.

## Causes probables

- ▶ Client configuré pour POST alors que l'API n'accepte que GET.
- ▶ API n'expose que GET/POST et ignore l'autre méthode.

## Correctifs

- ▶ Aligner le client sur la méthode attendue (ex. POST pour publier une température).
- ▶ Étendre l'API si pertinent (handler POST /temp pour accepter la publication).

# Erreur #2 — Mauvais chemin /temp

## Symptôme côté API

L'API ne propose pas exactement la route /temp (ex. /temperature, /temps, /temp/read), donc la requête échoue.

## Correctifs

- ▶ Ajuster l'URL côté client pour correspondre à la route réellement exposée par l'API.
- ▶ Ajouter la route manquante dans l'API si la documentation prescrit /temp.

# Bonnes pratiques REST associées

---

- ▶ Documenter précisément les méthodes autorisées (GET, POST, etc.) et les routes dans une spécification OpenAPI tenue à jour.
- ▶ Retourner des codes HTTP cohérents (ex. 405 Method Not Allowed quand la méthode n'est pas supportée) pour guider les clients.
- ▶ Valider systématiquement, côté API, la combinaison chemin/méthode/authentification/payload avant traitement.

# Une API REST: Démo avec Nod-Red

---

dans le navigateur `http://localhost:1880/`

# Une API REST: Exemple de code en python

---

```
from flask import Flask, request, jsonify
app = Flask(__name__)

# Endpoint pour ajouter une donnée
@app.route('/temp', methods=['POST'])
def ingest():
    data = request.json
    return jsonify(data), 201

if __name__ == '__main__':
    app.run(debug=True, host="0.0.0.0")
```

pour aller plus loin :

<https://flask.palletsprojects.com/en/stable/quickstart/>

# Introduction à l'API REST avec Python

## Étapes pour Tester l'API

- ▶ Lancer le serveur : `python app.py`.
- ▶ Accéder via un navigateur ou un outil comme Postman.
- ▶ Exemple de test avec curl :
  - ▶ `curl -X POST -d "19" http://127.0.0.1:5000/ingest`

# Transition : API REST → MQTT

## Complémentarité dans le fil rouge

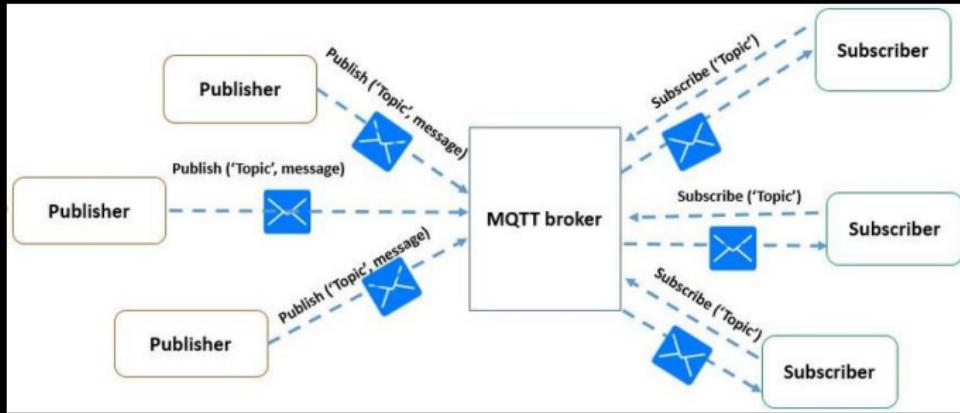
- ▶ Les API REST exposent les données long terme et les fonctions de supervision vers les applications métier.
- ▶ MQTT assure la collecte temps réel entre capteurs/passerelle et middleware avant stockage dans InfluxDB.
- ▶ Ensemble, ils couvrent les flux nord-sud et est-ouest de la plateforme.

## Question pour la suite

Quelles garanties (QoS, rétention) devons-nous assurer côté MQTT pour que les API REST restent alimentées ? Gardez cette question en tête !

- 1 Introduction
- 2 Rappels Réseau
- 3 Plateformes IoT
- 4 Topologies de déploiement
- 5 API REST et HTTP
- 6 Communication MQTT
- 7 Stockage et Analyse
- 8 Mise en oeuvre
- 9 Conclusion

# Modèle Pub/Sub et Broker de Messages



# Modèle Pub/Sub et Broker de Messages

## Rappel Théorique

- ▶ *Description* : Le modèle Pub/Sub utilise un broker pour transmettre les messages.
- ▶ *Rôles principaux* :
  - ▶ **Publisher** : Envoie des messages sur des sujets(topic) spécifiques.
  - ▶ **Subscriber** : S'abonne à des sujets pour recevoir des messages.
  - ▶ **Broker** : Intermédiaire qui gère la diffusion des messages.
- ▶ *Avantages* : Découplage des émetteurs et récepteurs, scalabilité.
- ▶ **MQTT**: utilisé dans l'IoT

# Présentation de MQTT

---

## Caractéristiques de MQTT

- ▶ *Protocole léger* : Idéal pour les communications M2M (Machine-to-Machine).
- ▶ *Basé sur Pub/Sub* : Conçu pour des réseaux peu fiables ou à faible bande passante.
- ▶ *Cas d'usage* :
  - ▶ IoT (Internet des Objets)
  - ▶ Monitoring temps réel
  - ▶ Communication microservices

# Qualité de Service (QoS) dans MQTT

## Qu'est-ce que le QoS ?

- ▶ Le QoS (Qualité de Service) définit le niveau de garantie pour la livraison des messages MQTT entre le client et le broker.
- ▶ Permet un compromis entre performance et fiabilité.

# Qualité de Service (QoS) dans MQTT (suite)

## Les niveaux de QoS

### ► **QoS 0 : Au maximum une fois**

- ▶ Transmission rapide, sans accusé de réception (**pertes**).
- ▶ Exemple : Données de capteurs.

### ► **QoS 1 : Au moins une fois**

- ▶ Le message est retransmis jusqu'à réception d'un accusé (PUBACK).
- ▶ Peut entraîner des doublons.
- ▶ Exemple : Alertes, notifications.

### ► **QoS 2 : Une seule fois**

- ▶ Le plus fiable, garantit l'absence de perte ou de duplication.
- ▶ Exemple : Transactions financières.

# Mini-quiz : choisir le bon QoS

---

## Consignes

- ▶ Associez un niveau de QoS à chacun de ces cas du fil rouge :
  - ▶ Alerte incendie publiée toutes les secondes.
  - ▶ Capteur de température envoyant une mesure toutes les 10 minutes.
- ▶ Justifiez votre choix en 2 phrases.

# Correction – QoS MQTT

---

Alerte incendie publiée toutes les secondes

**QoS recommandé : 0**

Un message perdu n'a pas d'impact car le suivant arrive immédiatement.  
Le QoS 0 minimise la latence et évite la surcharge réseau liée aux accusés de réception.

# Correction – QoS MQTT

---

Capteur de température, mesure toutes les 10 minutes

**QoS recommandé : 1**

Perdre une mesure crée un trou dans l'historique, ce qui est problématique pour l'analyse. Le QoS 1 garantit la livraison au moins une fois avec un surcoût faible.

# Mosquitto: une implémentation de MQTT

---

## Dépendances, packages:

---

- ▶ mosquitto
- ▶ mosquitto-clients
- ▶ paho-mqtt, pour le développement en python

Nécessité d'installer : pip install paho-mqtt

# Exemple : Publisher en Python

---

```
import paho.mqtt.client as mqtt

broker = "localhost"
port = 1883
topic = "mastercloud/test"

client = mqtt.Client()
client.connect(broker, port)
client.publish(topic, "Hello, MQTT!")
print("Message envoyé")
client.disconnect()
```

# Exemple : Subscriber en Python

---

```
import paho.mqtt.client as mqtt

broker = "localhost" # ip du broker
port = 1883 # le port du broker
topic = "mastercloud/test" #le topic
#function de rappel
def on_message(client, userdata, msg):
    print(f"Message reçu : {msg.payload.decode()}")

client = mqtt.Client()
client.connect(broker, port)
client.subscribe(topic)
client.on_message = on_message
client.loop_forever()
```

# Test et Résultat

---

## Étapes de Test

- ▶ Démarrez Mosquitto : `mosquitto`
- ▶ Lancez le Subscriber : `python subscriber.py`
- ▶ Lancez le Publisher : `python publisher.py`

## Résultat attendu

Le Subscriber affiche : Message reçu : Hello, MQTT!

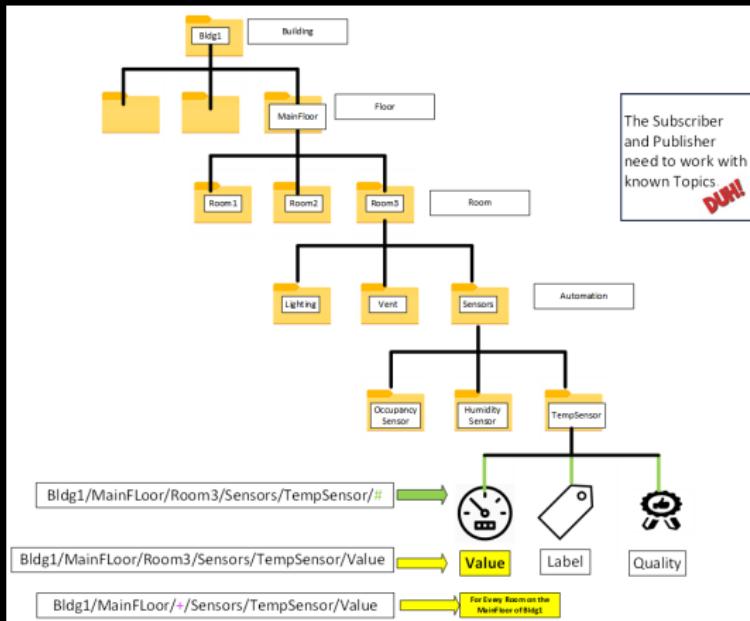
# Cas d'Usage de MQTT

---

## Exemples

- ▶ **IoT (Internet des Objets)** : Gestion de capteurs et actionneurs.
- ▶ **Monitoring** : Surveillance en temps réel des systèmes.
- ▶ **Cloud Computing** : Communication efficace entre microservices.

# Hierarchie des Topics



<https://store.chipkin.com/articles/mqtt-everything-you-need-to-know-in-5-simple-sentences>

# Règles de Nommage des Topics

---

- ▶ Les chaînes UTF-8 sont autorisées.
- ▶ Sensibles à la casse : Home/Room1  $\neq$  home/room1.
- ▶ Aucun / initial ou final n'est requis, mais ils sont autorisés.
- ▶ Les niveaux vides sont permis mais déconseillés :  
home//temperature (valide mais confus).

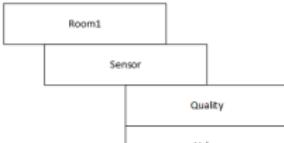
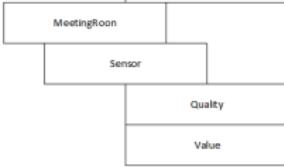
L'utilisation de "" est réservée aux topics d'information d'état MQTT.

# Jokers pour les Souscriptions

---

## Principes

- ▶ MQTT utilise deux jokers : + (un niveau) et # (multi-niveaux).
- ▶ Un subscriber peut recevoir les données de plusieurs capteurs grâce à ces jokers.
- ▶ Le publisher et le subscriber doivent connaître la hiérarchie des topics.

TOPIC Names	Multi Level Wild Card	SingleLevel Wild Card	SingleLevel Wild Card
Building1	Building1/#	Building1/Floor1+/Sensor/Value	Building1/Floor1+/Sensor
 Room1 Sensor Quality Value	Building1/Floor1		
Building1/Floor1			
Building1/Floor1/Room1		Building1/Floor1/Room1	
Building1/Floor1/Room1/Sensor		Building1/Floor1/Room1/Sensor	
Building1/Floor1/Room1/Sensor/Quality	Building1/Floor1/Room1/Sensor/Quality		Building1/Floor1/Room1/Sensor/Quality
Building1/Floor1/Room1/Sensor/Value	Building1/Floor1/Room1/Sensor/Value	Building1/Floor1/Room1/Sensor/Value	Building1/Floor1/Room1/Sensor/Value
 MeetingRoom Sensor Quality Value	Building1/Floor1/Room2		
Building1/Floor1/Room2		Building1/Floor1/Room2	
Building1/Floor1/Room2/Sensor		Building1/Floor1/Room2/Sensor	
Building1/Floor1/Room2/Sensor/Quality	Building1/Floor1/Room2/Sensor/Quality		Building1/Floor1/Room2/Sensor/Quality
Building1/Floor1/Room2/Sensor/Value	Building1/Floor1/Room2/Sensor/Value	Building1/Floor1/Room2/Sensor/Value	Building1/Floor1/Room2/Sensor/Value
	# always at end of topic filter	+ is only one level	

<https://store.chipkin.com/articles/mqtt-everything-you-need-to-know-in-5-simple-sentences>

# MQTT → Stockage

## Chaîne logique

- ▶ Les messages QoS choisis alimentent le middleware via Mosquitto.
- ▶ Le middleware transforme et achemine ces flux vers la base de séries temporelles.
- ▶ Les API REST consomment ensuite les données persistées (alertes, dashboards).

## Question de réflexion

Comment modéliser les mesures et tags InfluxDB pour retrouver facilement les capteurs de l'entrepôt ? Gardez cela en tête avant d'attaquer la prochaine section.

- 1 Introduction
- 2 Rappels Réseau
- 3 Plateformes IoT
- 4 Topologies de déploiement
- 5 API REST et HTTP
- 6 Communication MQTT
- 7 Stockage et Analyse
- 8 Mise en oeuvre
- 9 Conclusion

# Introduction à InfluxDB

## Qu'est-ce qu'InfluxDB ?

- ▶ *Description* : InfluxDB est une base de données orientée séries temporelles (Time Series Database) optimisée pour des cas d'usage liés à l'IoT, la surveillance, et l'analyse de données en temps réel.
- ▶ *Caractéristiques* :
  - ▶ Stocke des données structurées autour d'une notion de temps.
  - ▶ Permet l'ingestion, le stockage et la requête de données massives.
  - ▶ Écrit en Go, léger et rapide.

# Fonctionnalités principales

---

## Pourquoi utiliser InfluxDB ?

---

- ▶ *Ingestion rapide* : Gère des flux massifs de données avec une faible latence.
- ▶ *Langage de requête* : Offre InfluxQL (similaire à SQL) et Flux pour des analyses avancées.
- ▶ *Flexibilité* : Pas de schéma rigide, idéal pour des données dynamiques.
- ▶ *Visualisation* : Intégré avec des outils comme Grafana pour des tableaux de bord.

# Schéma InfluxDB — bonnes pratiques

---

- ▶ **Measurement** : un type métier (ex : temperature, machineStatus).
- ▶ **Tags** : métadonnées indexées, faible cardinalité (zone, capteur, QoS).
- ▶ **Fields** : valeurs mesurées (float, int, string) non indexées mais requêtables.
- ▶ **Timestamp** : généré par le client ou le serveur, précision ns conseillée.
- ▶ **Bucket & rétention** : regroupent les séries et définissent leur durée de vie.

# Exemple de schéma InfluxDB

Point type « warehouseSensors »

**Measurement** : warehouseSensors

**Tags** : zone=Nord, device=capteur-12, type=temperature

**Fields** : value=22.7, battery=89

**Timestamp** : 2024-12-01T10:00:00Z

Intérêt

- ▶ Filtrer rapidement par zone ou type via les tags indexés.
- ▶ Agréger les valeurs (value) tout en gardant d'autres métriques (batterie, RSSI).
- ▶ Partager un même bucket tout en contrôlant la rétention (ex : 30 jours).

# Activité guidée : stockage du fil rouge

## Question

En vous appuyant sur les caractéristiques d'InfluxDB, justifiez en binôme pourquoi ce choix est adapté au suivi de température de l'entrepôt et quels compromis il impose.

## Pistes de réflexion (5 minutes)

- ▶ Fréquence d'échantillonnage, volumétrie prévue et politique de rétention.
- ▶ Besoins de requêtes agrégées pour déclencher les alertes de la plateforme.
- ▶ Intégration prévue avec Grafana ou d'autres API descendantes.

# Activité guidée : stockage du fil rouge

## Éléments de réponse

- ▶ InfluxDB absorbe des mesures fréquentes (toutes les 10 min voire 1 s) grâce à son moteur time-series optimisé et permet de fixer une rétention (ex : 30 jours chaud + downsample vers 6 mois).
- ▶ Les requêtes Flux/InfluxQL calculent moyenne, min ou seuils glissants pour déclencher les alertes middleware sans exporter tout l'historique.
- ▶ Intégration native avec Grafana : dashboards temps réel, API Flux pour alimenter REST ; compromis : nécessite gérer cardinalité des tags et capacité disque selon la rétention.

# Ingestion des données de température

---

```
from influxdb_client import InfluxDBClient, Point
from influxdb_client.client.write_api import SYNCHRONOUS

client = InfluxDBClient(url="http://localhost:8086", token="my_token", org="my_org")
bucket = "my_bucket"

# Ingestion des données
def ingest_data():
    write_api = client.write_api(write_options=SYNCHRONOUS)
    data = [
        {"time": "2024-12-01T10:00:00Z", "temperature": 22.5},
        {"time": "2024-12-01T11:00:00Z", "temperature": 23.1}
    ]
    for record in data:
        point = Point("temperature").field("value", record["temperature"]).time(record["time"])
        write_api.write(bucket=bucket, org="my_org", record=point)
    print("Données ingérées.")
```

# Analyse des données de température

---

```
from influxdb_client import InfluxDBClient

def analyze_data():
    client = InfluxDBClient()
    query_api = client.query_api()
    query = """
from(bucket: "my_bucket")
|> range(start: -1d)
|> filter(fn: (r) => r["_measurement"] == "temperature")
|> filter(fn: (r) => r["_field"] == "value")
|> mean()
...
    tables = query_api.query(query, org="my_org")
    print("Température moyenne sur 24h:")
    for table in tables:
        for record in table.records:
            print(f"{record.get_value()}°C")
```

# Analyse des données de température

## Ce que calcule cette requête Flux

- ▶ Filtre toutes les mesures “temperature” sur les dernières 24h.
- ▶ Ne conserve que le champ “value” avant de calculer une moyenne temporelle.
- ▶ Prépare l’exploitation des résultats par l’API ou une règle d’alerte.

## Pré-requis pour l’exécution

- ▶ Paquet influxdb-client installé et accès réseau vers l’instance InfluxDB.
- ▶ Token et organisation configurés comme dans l’exemple d’ingestion.

# Flux — syntaxe de base (1/2)

---

- ▶ Les requêtes sont des pipelines : `from() → range() → filter()` → transformations.
- ▶ Chaque étape est séparée par `|> (pipe)` et reçoit les tables sorties par l'étape précédente.
- ▶ Paramètres essentiels :
  - ▶ `from(bucket: "myBucket")` :  
`sourcedesséries.range(start: -1h, stop: now())` : fenêtre temporelle.
  - ▶ `filter(fn: (r) =>`  
`r["measurement"] == "temperature")` : sélection des mesures.
  - ▶ Les colonnes standard : `_time, _value, _measurement, _field`, tags personnalisés.

# Flux — syntaxe de base (2/2)

---

- ▶ Assignations avec `myQuery =` permettent de réutiliser un pipeline.
- ▶ Les fonctions comme `aggregateWindow(every: 5m, fn: mean)` ou `derivative(unit: 1m)` s'appliquent après les filtres.
- ▶ `yield(name: "result")` donne un alias pour l'affichage dans UI/API.
- ▶ Exemple :

# Exemple de requête avec InfluxQL

---

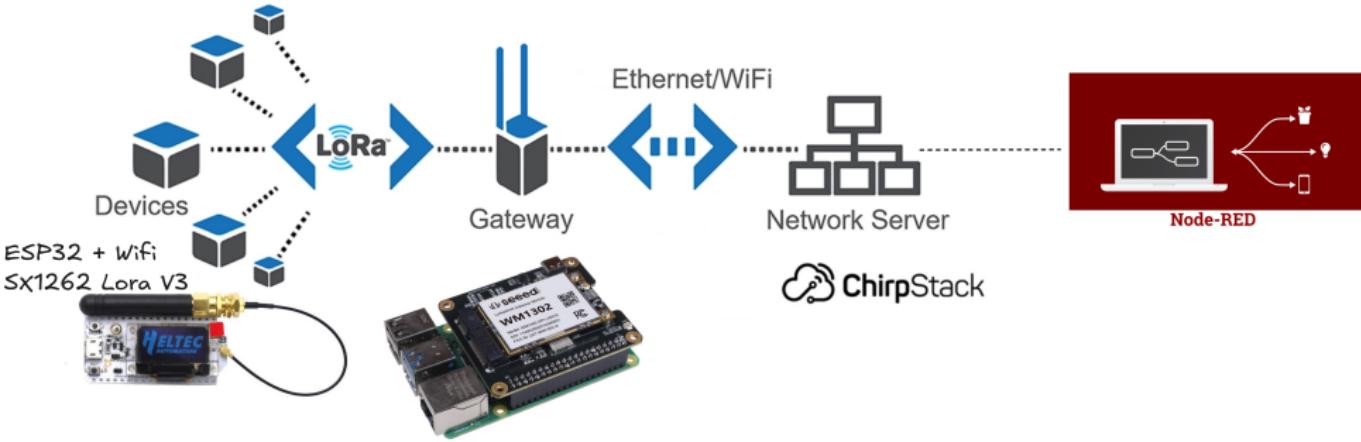
```
from(bucket: "my_bucket")
|> range(start: -6h)
|> filter(fn: (r) => r.device == "capteur-12")
|> aggregateWindow(every: 10m, fn: mean)
|> yield(name: "moyennes")
```

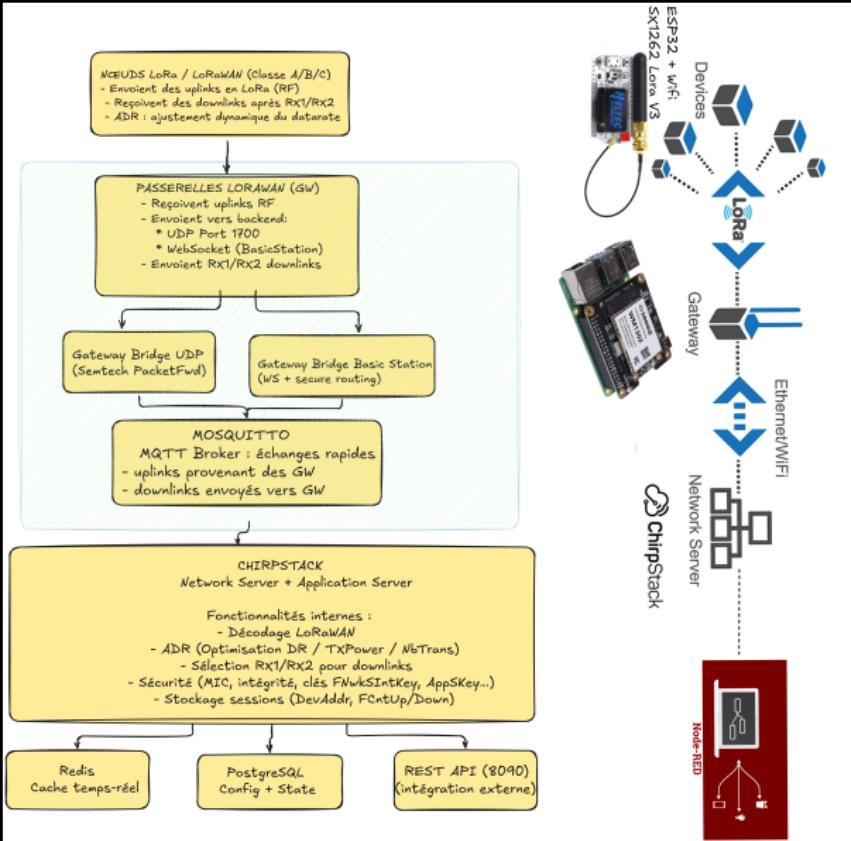
# Exemple de requête avec InfluxQL

---

```
SELECT temperature, humidity
FROM my_bucket
WHERE time > now() - 1h
AND location = 'Paris'
GROUP BY time(10m)
```

- 1 Introduction
- 2 Rappels Réseau
- 3 Plateformes IoT
- 4 Topologies de déploiement
- 5 API REST et HTTP
- 6 Communication MQTT
- 7 Stockage et Analyse
- 8 Mise en oeuvre
- 9 Conclusion





- 1 Introduction
- 2 Rappels Réseau
- 3 Plateformes IoT
- 4 Topologies de déploiement
- 5 API REST et HTTP
- 6 Communication MQTT
- 7 Stockage et Analyse
- 8 Mise en oeuvre
- 9 Conclusion

# Résumé des étapes

---

## Réponse à la problématique initiale

---

- ▶ L'architecture NGN fournit les services, protocoles et ressources nécessaires aux dispositifs IoT.
- ▶ Le middleware (broker, règles, API) assure l'intégration des flux MQTT issus des capteurs de l'entrepôt.
- ▶ InfluxDB et les applications amont ferment la boucle en fournissant stockage temporel, analyses et alertes.

# Refs

---

- ① Syed Riffat Ali, Next Generation and Advanced Network Reliability Analysis, Springer
- ② Anand Tamboli, Build Your Own IoT Platform, Apress