TP5 : Architecture hybride de bout en bout : Edge Computing, Sécurité, Routage Dynamique, Stockage Polyglot (SQL/NoSQL) et Cloud Analytics.

NiFi PostgreSQL InfluxDB Azure/AWS

1. Contexte & Liens Théoriques (Chapitre 5)

Ce **TP** met en œuvre les concepts fondamentaux du cycle de vie des données IoT dans un scénario industriel réel. Nous allons passer de la théorie à la pratique en construisant un pipeline résilient.

ETL & Streaming

Passage du **Batch** au **Streaming**. Utilisation d'Apache NiFi pour l'ingestion continue (Extract), le filtrage local (Transform) et le routage (Load).

Stockage Polyglot

Application de la stratégie **SQL vs NoSQL** : PostgreSQL pour la cohérence (ACID), InfluxDB pour la vélocité (Time-Series), MongoDB pour la variété (Logs).

Cloud Hybride

Architecture **Edge-Cloud**. Traitement critique en bordure pour la latence, analyse massive dans le Cloud (AWS/Azure) pour l'historique.

> ***Objectifs pédagogiques***
>
> - Construire un **pipeline IoT de bout en bout (Edge → Cloud).**
> - Justifier les choix de stockage **SQL/NoSQL** selon le besoin métier.
> - Mettre en place un **routage dynamique** basé sur des seuils techniques.
> - Valider la qualité des données avant exploitation analytique.

2. Architecture Cible

Node-RED (Simulateur & UI Locale)

MQTT

Apache NiFi (Edge Gateway - Sécurité & Filtrage)

Routage & Enrichissement

Stockage Polyglot (PostgreSQL, InfluxDB, MongoDB, S3)

Cloud Analytics & Dashboards (Grafana / Power BI)

> **Méthode de Travail:**
> 1. **Faire :** Exécuter l'étape avec une seule modification à la fois.
> 2. **Vérifier :** Capturer une preuve (log, capture écran).
> 3. **Expliquer :** Justifier en 1 phrase le choix technique réalisé.

Phase 1 : Guide d'Installation Multi-Plateforme

Avant de commencer le TP, assurez-vous d'avoir installé les outils suivants. Choisissez entre l'installation rapide via Docker ou l'installation native selon votre système d'exploitation.

> **Mission :** Installer et démarrer au minimum **NiFi**, **Node-RED** et **PostgreSQL**.
> **Critère de réussite :** accès web à **NiFi/Node-RED** + connexion **PostgreSQL** opérationnelle.

Installation Rapide via Docker

- Créez un fichier `docker-compose.yml` avec le contenu ci-dessous

```
version: '3.8'

services:
```

```
postgres:
  image: postgres:14
  environment:
    POSTGRES_DB: smart_factory
    POSTGRES_USER: admin
    POSTGRES_PASSWORD: admin
  ports:
    - "5432:5432"

influxdb:
  image: influxdb:2.0
  ports:
    - "8086:8086"

mongo:
  image: mongo:latest
  ports:
    - "27017:27017"

grafana:
  image: grafana/grafana
  ports:
    - "3001:3001"

node-red:
  image: nodered/node-red:latest
  ports:
    - "1880:1880"
```

- lancez `docker-compose up -d`

Note : Apache NiFi est souvent plus complexe à configurer via Docker pour un TP débutant. L'installation native est recommandée pour NiFi.

Installation Apatche NIFI

1. **Java JDK 21 (Requis pour NiFi)**.

Windows: `winget install EclipseAdoptium.Temurin.21.JDK`
macOS: `brew install openjdk@21`
Linux: `sudo apt install openjdk-21-jdk`

2. **Apache NiFi :**

Windows :

1. Télécharger l'archive, extraire(ex. C:\nifi), et lancer : `cd C:\nifi\bin`.
2. Démarrer : `nifi start`

macOS:

1. Installer Homebrew (si nécessaire) : `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`

2. Installer NiFi : `brew install nifi`
3. Gestion du service : `nifi start | status | stop`

Linux:

3. Télécharger & Extraire :

```
wget https://dlcdn.apache.org/nifi/2.9.0/nifi-2.9.0-bin.zip
unzip nifi-*.zip
cd nifi-*
```

4. Démarrer : `bin/nifi.sh start`

Remarques & Astuces

- **Accès Web :** Par défaut http://localhost:8080/nifi (ou https://localhost:8443/nifi si HTTPS activé).
- **Logs :** Les logs se trouvent dans logs/nifi-app

`bin/nifi.sh start` (Linux/Mac). Accès via http://localhost:8080/nifi ou https://localhost:8443/nifi

Phase 2 : Initialisation PostgreSQL

1. **Se connecter à PostgreSQL:** `psql -U admin -d smart_factory`

Si PostgreSQL installer sur Docker : `docker exec -it <nom_du_container_postgres> psql -U admin -d smart_factory`

2. **Créer table**

```
CREATE TABLE machines (
    id VARCHAR PRIMARY KEY,
    type VARCHAR,
    localisation VARCHAR,
    date_install DATE
);
```

3. **Insérer une machine:** `INSERT INTO machines (id, type, localisation, date_install) VALUES ('M-204','Presse','Zone A','2020-01-01');`
4. **Vérifier:** `SELECT * FROM machines;`

Résultat attendu : PostgreSQL contient les métadonnées de la machine M-204.

```
smart_factory=# SELECT * FROM machines;
  id   |  type  | localisation | date_install
-------+--------+--------------+--------------
 M-204 | Presse | Zone A       | 2020-01-01
(1 row)
```

Quitter PostgreSQL (quand fini): `\q`

Phase 3 : Pipeline NiFi - Ingestion, Sécurité & Filtrage

Configuration du **flux de données** dans **Apache NiFi** pour ***simuler la passerelle***.

3.1. Accéder à NiFi

- Ouvrir navigateur: http://localhost:8080/nifi (ou https://localhost:8443/nifi si HTTPS activé).
- Se connecter: Un utilisateur/mot de passe est généré automatiquement. Récupérez-le via : grep "Generated Username" logs/nifi-app.log

Mac:

- `$ cd ~/nifi/logs` ***ou*** `$ cd /chemin/vers/nifi/logs`

Ex. `cd /usr/local/Cellar/nifi/2.9.0/libexec/logs`

- `grep -i "Generated Username" nifi-app*`
- `grep -i "password" nifi-app*`

Windows:

- `cd %USERPROFILE%\nifi\logs` ***ou*** `$ cd C:\chemin\vers\nifi\logs`

Ex. `cd C:\nifi-2.9.0\logs`

- `findstr /i "Generated Username" nifi-app*`
- `findstr /i "password" nifi-app*`

Pour changer/créer username et password: `nifi set-single-user-credentials <username> <password>`

3.2. Créer un Process Group

- Glisser **Process Group**
- Nom : `IoT_Pipeline`
- Double-cliquer pour entrer dedans

3.3. **Source de Données : GenerateFlowFile.**

Simule un capteur envoyant un JSON toutes les 5 secondes.

- ***Ajouter Processor et Chercher :*** `GenerateFlowFile`

Configurer:

Onglet Scheduling

`Run Schedule = 5 sec`

Onglet Properties

Dans **Custom Text**, coller :

```
{
  "machine_id": "M-204",
  "timestamp": "2025-10-27T10:00:00Z",
  "vibration_x": 12.5,
  "vibration_y": 11.2,
  "temperature": 65.4,
  "status": "running"
}
```

-

3.4. Sécurité X.509 (Simulée)

Objectif

- Simuler une vérification de certificat avant traitement.

Utilisation de `UpdateAttribute` pour ajouter `cert_status = valid`, puis `RouteOnAttribute` avec la règle : `${cert_status:equals('valid')}`.

Étape A : UpdateAttribute

1. Ajouter Processor

- `UpdateAttribute`

2. Connecter depuis GenerateFlowFile

3. Configurer (Properties)

```
cert_status = valid
```

4. Apply

Étape B : RouteOnAttribute(security)

1. Ajouter Processor

- `RouteOnAttribute`

2. Connecter depuis UpdateAttribute

3. Configurer (Properties)

Ajouter :

```
valid = ${cert_status:equals('valid')}
```

IMPORTANT : corriger Relationships

Aller dans Relationships

- ☑ cocher **terminate** pour `unmatched`

- ☑ cocher **terminate** pour `valid` (temporairement)

Sinon → erreur “Invalid”

Résultat attendu

- Valid(flux valide) → passe vers étape suivante
- Unmatched(flux invalide) → rejeté

Les flux invalides sont envoyés vers un dossier "rejected" (PutFile)ou un log LogAttribute pour debug.

Extract JSON
Ajouter :

- Processor : `EvaluateJsonPath`

Configurer :

```
vibration_x = $.vibration_x
vibration_y = $.vibration_y
Destination = flowfile-attribute
Return Type = scalar
```

3.5. Filtrage du Bruit (Edge, data quality)

Rejet des données incohérentes (ex: vibration négative) avant l'envoi au Cloud:
`${vibration_x:gt(0):and(${vibration_y:gt(0)})}`

Étapes

1. Ajouter `RouteOnAttribute` (2ème)

2. Connecter depuis le premier RouteOnAttribute

3. Ajouter règle :

```
valid_data = ${vibration_x:gt(0):and(${vibration_y:gt(0)})}
```

4. Relationships

- ☑ terminate pour `unmatched`
- ☑ terminate pour `valid_data`

Résultat

- vibrations négatives → rejetées
- données correctes → passent

3.6. Gestion de la Surcharge (Back-pressure)

Configuration de la connexion : **Object Threshold = 10000**. Si le flux est trop important, NiFi ralentit automatiquement.

1. Cliquer sur la connexion (flèche entre processors)
2. Settings :

```
Object Threshold = 10000
```

Si trop de données → NiFi ralentit automatiquement
Adapter selon environnement :

- Dev : 100
- Prod : 100k+

Étape 7 : Ajouter un output (test)

Pour voir les résultats :

LogAttribute

- Ajouter processor `LogAttribute`

- Connecter à la fin du pipeline
- Permet de voir les données dans les logs

```
 cd /usr/local/Cellar/nifi/2.9.0/libexec/logs

 tail -f nifi-app*     (mac/Linux)

 type nifi-app.log (windows)

resultat:
….
```

FlowFile Properties
Key: 'entryDate'
Value: 'Tue Apr 21 23:55:42 WEST 2026'
Key: 'lineageStartDate'
Value: 'Tue Apr 21 23:55:42 WEST 2026'
Key: 'fileSize'
Value: '158'
FlowFile Attribute Map Content
Key: 'RouteOnAttribute.Route'
Value: 'unmatched'
Key: 'cert_status'
Value: 'valid'
Key: 'filename'
Value: 'a1fe8a9b-c5b0-4f14-909a-dbde276822e0'

Key: 'path'
Value: './'
Key: 'uuid'
Value: 'fe81faa6-2e8a-4e07-bca4-783c50a45f82'

{
"machine_id": "M-204",
"timestamp": "2025-10-27T10:00:00Z",
"vibration_x": 12.5,
"vibration_y": 11.2,
"temperature": 65.4,
"status": "running"
}

Ajouter le processor PutFile

Étape 1 :

1. Clique sur le canvas NiFi
2. Clique sur + **(Add Processor)**
3. Cherche : `PutFile`
4. Sélectionne **PutFile 2.9.0**
5. Clique sur **Add**

Étape 2 : Connecter PutFile

1. Clique sur ton processor précédent (`RouteOnAttribute`)
2. Tire une flèche vers `PutFile`
3. Choisis la relation :

`valid_data` (seulement le flux valide doit aller vers PutFile).

Étape 3 : Configurer PutFile

Double-clique sur `PutFile`

Onglet Properties

1. Directory (IMPORTANT)

Mets un chemin absolu

Windows :

```
C:\nifi_output
```

Linux ou mac :

```
/home/$USER/nifi_output
```

2. Conflict Resolution Strategy

Choisir : replace
évite les erreurs si fichier existe

3. Create Missing Directories
`true`

Étape 4 : Configurer Relationships

Va dans **Relationships**
Coche :
✔ `success → terminate`
✔ `failure → terminate`
obligatoire pour éviter “Invalid”

Étape 5 : Démarrer le processor

1. Clique droit sur `PutFile`
2. Clique sur **Start**

Étape 6 : Créer le dossier (IMPORTANT)

Windows : mkdir C:\nifi_output

Linux : mkdir -p /home/$USER/nifi_output

Architecture réaliste IoT

Pipeline final :

```
GenerateFlowFile || ConsumeMQTT || ListenHTTP || …
   ↓
UpdateAttribute (cert)
   ↓
RouteOnAttribute (security)
   ↓
EvaluateJsonPath
   ↓
RouteOnAttribute (data quality)
   ↓
[valid] → Cloud / Kafka / DB (Procecors dédiés)
[invalid] → rejected
```

Bonus :

- Intégrer avec Postgresql
- Consommer des données venant de capteurs via MQTT ou HTTP