

Table des matières

I. Projet PUE avec ELK.....	2
A. Configuration.....	2
1. Configuration Initiale VsCode.....	2
2. Configuration d'une VM pour ELK	3
3. Configuration Variables d'Environnements pour la connexion	3
4. Initialisation du système de Commit de Git.....	4
5. Si Projet Terminé ! Configuration des ressources pour l'exportation (Docker).....	5
B. Script Principal : index.js	6
C. Visualisation avec Kibana	9
II. Explication générale	10
A. Explication du script index.js :	10
B. Explication d'un index dans ELK	12
✦ 1. Création automatique de l'index.....	12
✦ 2. Détection automatique des types de propriétés	12
✦ 3. Forcer un mapping personnalisé (optionnel/déconseillé)	13
✦ 4. Vérifier le mapping d'un index dans ELK	13
✦ 5. Rappel des termes employés.....	14
Ressources :	15

I. Projet PUE avec ELK

Application Node.js (version v22.13.0) qui récupère et stocke la température, consommation d'énergie, taux d'humidité, etc des PDU's et des COOLINGS grâce aux OIDs dans Elasticsearch et Kibana (Version 8.17.0 ou Version 8.17.1) .

A. Configuration

1. Configuration Initiale VsCode

1. Créer un répertoire pour le projet :

```
PS C:\Users\joan.perrey-prato\Documents\Projet_E-SI_PUE\Version_fonctionnelle\puev4>
```

2. Initialiser un projet Node.js (créer un fichier package.json) :

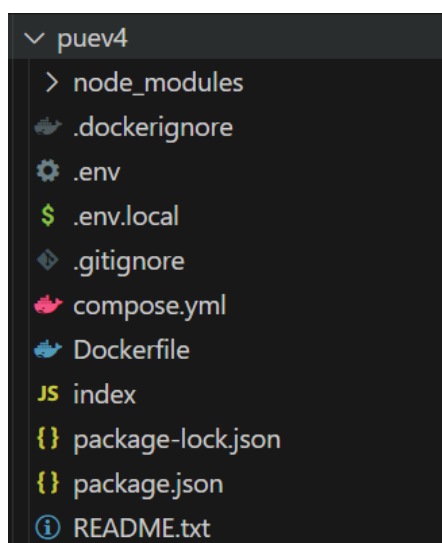
```
npm init
```

3. Installer les dépendances nécessaires :

```
npm install dotenv winston ...
```

```
"dependencies": {  
  "@elastic/elasticsearch": "^8.17.0",  
  "dotenv": "^16.4.7",  
  "moment-timezone": "^0.5.47",  
  "net-snmp": "^3.18.2",  
  "node-cron": "^3.0.3",  
  "winston": "^3.17.0"  
},
```

4. Créer tous les fichiers nécessaires :



```
▼ puev4  
  > node_modules  
  .dockerignore  
  .env  
  .env.local  
  .gitignore  
  compose.yml  
  Dockerfile  
  JS index  
  {} package-lock.json  
  {} package.json  
  ⓘ README.txt
```

2. Configuration d'une VM pour ELK

Dans une VM linux,

- 1) Mise en place de Docker et des conteneurs ELK
- 2) Configuration du réseau et de l'accès

3. Configuration Variables d'Environnements pour la connexion

- 1) Créer le fichier .env.local qui sera avec les valeurs adaptées à votre environnement :

```
projet_pue_git > bay-power-consumption-collector > pue > $ .env.local
1 # Elasticsearch
2 ELASTICSEARCH_URL=https://<votre_adresse_ip>:9200
3 ELASTICSEARCH_USER=changeMe!
4 ELASTICSEARCH_PASSWORD=changeMe!
5
6 # SNMP SHNEIDER
7 SNMP_USER=changeMe!
8 SNMP_AUTH_PASSWORD=changeMe!
9 SNMP_PRIV_PASSWORD=changeMe!
10
11 # =====
12 # LISTE DES PDU (Prises intelligentes)
13 # =====
14 # Format : IP1,NOM1,IP2,NOM2,IP3,NOM3,...
15 # Exemple : 192.168.1.1,PDU1,192.168.1.2,PDU2
16 PDU_IPS=192.168.1.1,PDU1,192.168.1.2,PDU2
17
18 # =====
19 # LISTE DES SYSTÈMES DE REFROIDISSEMENT (COOLING)
20 # =====
21 # Format : IP1,NOM1,IP2,NOM2,IP3,NOM3,...
22 # Exemple : 192.168.1.10,COOLING1,192.168.1.11,COOLING2
23 COOLING_IPS=192.168.1.10,COOLING1,192.168.1.11,COOLING2
```

- 2) Créer le fichier .env qui sera avec des valeurs par défaut :

```
projet_pue_git > bay-power-consumption-collector > pue > .env
1 # # Elasticsearch
2
3 # # ELASTICSEARCH_URL=https://<votre_adresse_ip>:9200
4 # # ELASTICSEARCH_USER=changeMe!
5 # # ELASTICSEARCH_PASSWORD=changeMe!
6
7 # # SNMP SHNEIDER
8
9 # # SNMP_USER=changeMe!
10 # # SNMP_AUTH_PASSWORD=changeMe!
11 # # SNMP_PRIV_PASSWORD=changeMe!
12
13 # # =====
14 # # LISTE DES PDU (Prises intelligentes)
15 # # =====
16 # # Format : IP1,NOM1,IP2,NOM2,IP3,NOM3,...
17 # # Exemple : 192.168.1.1,PDU1,192.168.1.2,PDU2
18
19 # # PDU_IPS=192.168.1.1,PDU1,192.168.1.2,PDU2
20
21 # # =====
22 # # LISTE DES SYSTÈMES DE REFROIDISSEMENT (COOLING)
23 # # =====
24 # # Format : IP1,NOM1,IP2,NOM2,IP3,NOM3,...
25 # # Exemple : 192.168.1.10,COOLING1,192.168.1.11,COOLING2
26
27 # # COOLING_IPS=192.168.1.10,COOLING1,192.168.1.11,COOLING2
```

3) Ajouter au fichier .gitignore (ne fera pas de commit avec les fichiers suivant) :

```
Version_fonctionelle > puev4 > .gitignore
1  # Fichiers d'environnement
2  .env.local
3
4  # Node.js
5  node_modules/
```

4. Initialisation du système de Commit de Git

1) Étape 1 : Vérifier la clé SSH

```
ls ~/.ssh/id_rsa.pub
cat ~/.ssh/id_rsa.pub
ssh -T git@gitlab.e-si.fr
```

2) Étape 2 : Cloner le dépôt

```
git clone git@gitlab.e-si.fr:e-si/bay-power-consumption-
collector.git
```

3) Étape 3 : Ajouter et commit les modifications

```
cd bay-power-consumption-collector
git add .
git commit -m "Description claire des modifications"
```

4) Étape 4 : Gérer les branches

```
git checkout -b feature/nom-de-la-branche
git push -u origin feature/nom-de-la-branche
```

5) Étape 5 : Fusionner les branches

```
git checkout master
git merge feature/nom-de-la-branche
```

6) Étape 6 : Pousser les modifications sur GitLab

```
git push origin master
git push --tags
```

5. Si Projet Terminé ! Configuration des ressources pour l'exportation (Docker)

1) Création du réseau Docker personnalisé avec la commande suivante :

```
joan@P-0000094:/mnt/c/Users/joan.perrey-prato$ docker network create --driver=bridge --subnet=192.168.101.0/24 --gateway=192.168.101.1 puev4_custom_network
```

2) Vérification de la création du réseau :

```
joan@P-0000094:/mnt/c/Users/joan.perrey-prato$ docker network inspect puev4_custom_network
```

3) Modification du compose.yml y ajouter le réseau externe (déjà créé) :

```
1  ∨ services:
2  ∨   pue:
3      image: joan/pue
4      container_name: pue
5  ∨   networks:
6      - puev4_custom_network
7  ∨   environment:
8      ELASTICSEARCH_URL: https://10.10.3.109:9200
9      ELASTICSEARCH_USER: elastic
10     ELASTICSEARCH_PASSWORD: nb3tzuxdVWovVpXRaDjB
11     SNMP_USER: apc
12     SNMP_AUTH_PASSWORD: jfjdhfjkbfigremqsf345dsjhdebu
13     SNMP_PRIV_PASSWORD: dfnkeemKgrtbrtgbdc4
14     restart: always
15
16  ∨ networks:
17  ∨   puev4_custom_network:
18      external: true
```

4) Création du Dockerfile, avec ajout de la variable d'environnement pour OpenSSL avec `ENV NODE_OPTIONS=--openssl-legacy-provider` :

```
Version_fonctionelle > puev4 > Dockerfile
1  FROM node:23-alpine
2  # Create app directory
3  WORKDIR /usr/src/app
4  # Install app dependencies
5  # A wildcard is used to ensure both package.json
6  # where available (npm@5+)
7  COPY package*.json ./
8  RUN npm install
9
10  ENV NODE_OPTIONS=--openssl-legacy-provider
11
12  # Bundle app source
13  COPY . .
14  CMD [ "node", "index" ]
```

B. Script Principal : index.js

```

projet_pue_git > bay-power-consumption-collector > pue > .js index.js > ...
1  process.env.NODE_TLS_REJECT_UNAUTHORIZED = '0'; // Ignorer les erreurs TLS
2  process.env.NODE_OPTIONS = "--openssl-legacy-provider";
3
4  const dotenv = require('dotenv');
5  const snmp = require("net-snmp");
6  const { Client } = require('@elastic/elasticsearch');
7  const cron = require('node-cron');
8  dotenv.config({ path: '.env' });
9  dotenv.config({ path: '.env.local' });
10
11 const client = new Client({
12   node: process.env.ELASTICSEARCH_URL,
13   auth: {
14     username: process.env.ELASTICSEARCH_USER,
15     password: process.env.ELASTICSEARCH_PASSWORD
16   }
17 });
18
19 /* // Verification des acces :
20 console.log("Elasticsearch URL:", process.env.ELASTICSEARCH_URL);
21 console.log("Elasticsearch User:", process.env.ELASTICSEARCH_USER);
22 console.log("Elasticsearch Password:", process.env.ELASTICSEARCH_PASSWORD);
23 */
24
25 // ♦ Fonction pour récupérer le nom de l'index Elasticsearch
26 function getCurrentIndexName(deviceType) {
27   const now = new Date();
28   const month = String(now.getMonth() + 1).padStart(2, '0'); // Mois actuel (01-12)
29   const year = now.getFullYear(); // Année actuelle
30   return `${deviceType}_data_${month}_${year}`;
31 }
32
33 // ♦ Vérifie si l'index existe, sinon le crée
34 async function ensureIndexExists(indexName) {
35   try {
36     const exists = await client.indices.exists({ index: indexName });
37     if (!exists) {
38       await client.indices.create({ index: indexName });
39       console.log(`Index ${indexName} créé.`);
40     }
41   } catch (error) {
42     console.error(`Erreur lors de la création de l'index ${indexName}:`, error);
43   }
44 }
45

```

```

46 // * Fonction pour insérer les données dans Elasticsearch
47 async function insertDataToElasticsearch(deviceType, data) {
48   const indexName = getCurrentIndexName(deviceType);
49   await ensureIndexExists(indexName);
50
51   try {
52     await client.index({
53       index: indexName,
54       body: data
55     });
56     console.log(`Données insérées dans Elasticsearch (${indexName})`);
57   } catch (error) {
58     console.error(`Erreur d'insertion dans Elasticsearch (${indexName})`, error);
59   }
60 }
61
62 function parseDevices(envVar) {
63   const devicesArray = process.env[envVar]?.split(',') || [];
64   const devices = {};
65   for (let i = 0; i < devicesArray.length; i += 2) {
66     devices[devicesArray[i]] = devicesArray[i + 1];
67   }
68   return devices;
69 }
70
71 const deviceMapping = {
72   PDU: {
73     devices: parseDevices("PDU_IPS"),
74     oids: {
75       humidity: "1.3.6.1.4.1.318.1.1.26.10.2.2.1.10.1",
76       tempFront: "1.3.6.1.4.1.318.1.1.26.10.2.2.1.8.1",
77       power: "1.3.6.1.4.1.318.1.1.12.1.16.0"
78     },
79     snmpConfig: {
80       authProtocol: snmp.AuthProtocols.sha,
81       privProtocol: snmp.PrivProtocols.aes
82     }
83   },
84   COOLING: {
85     devices: parseDevices("COOLING_IPS"),
86     oids: {
87       fanSpeed: "1.3.6.1.4.1.318.1.1.13.3.2.2.2.16.0",
88       enteringFluidTemp: "1.3.6.1.4.1.318.1.1.13.3.2.2.2.24.0",
89       leavingFluidTemp: "1.3.6.1.4.1.318.1.1.13.3.2.2.2.26.0"
90     },
91     snmpConfig: {
92       authProtocol: snmp.AuthProtocols.md5,
93       privProtocol: snmp.PrivProtocols.des
94     }
95   }
96 };

```

```

99 // * Fonction pour récupérer les données SNMP avec la bonne configuration
100 function fetchDeviceData(deviceType, ip) {
101   const deviceName = deviceMapping[deviceType].devices[ip];
102   const oids = Object.values(deviceMapping[deviceType].oids);
103   const snmpConfig = deviceMapping[deviceType].snmpConfig;
104
105   const options = {
106     port: 161,
107     retries: 2,
108     timeout: 5000,
109     transport: "udp4",
110     version: snmp.Version3,
111     backoff: 2.5
112   };
113
114   const user = {
115     name: process.env.SNMP_USER, // Nom d'utilisateur SNMP commun aux deux
116     level: snmp.SecurityLevel.authPriv,
117     authProtocol: snmpConfig.authProtocol,
118     authKey: process.env.SNMP_AUTH_PASSWORD,
119     privProtocol: snmpConfig.privProtocol,
120     privKey: process.env.SNMP_PRIV_PASSWORD
121   };
122
123   const session = snmp.createV3Session(ip, user, options);
124
125   session.get(oids, function (error, varbinds) {
126     if (error) {
127       console.error(`Erreur SNMP pour ${ip} (${deviceName}):`, error.toString());
128     } else {
129       const data = {
130         device_name: deviceName,
131         device_ip: ip,
132         timestamp: new Date().toISOString()
133       };
134
135       // Associer les valeurs récupérées aux clés de l'OID
136       Object.keys(deviceMapping[deviceType].oids).forEach((key, index) => {
137         let value = parseFloat(varbinds[index].value.toString());
138
139         // * Modifications spécifiques pour certains OIDs
140         if (key === "humidity") {
141           value = value / 100; // pour mettre la valeur sur 1 ( en % sur ELK )
142         } else if (key === "tempFront") {
143           value = value / 10; // pour remettre la valeur par défaut
144         } else if (key === "fanSpeed") {
145           value = value / 1000; // pour mettre la valeur sur 1 ( en % sur ELK )
146         } else if (key === "enteringFluidTemp") {
147           value = value / 10; // pour remettre la valeur par défaut
148         } else if (key === "leavingFluidTemp") {
149           value = value / 10; // pour remettre la valeur par défaut
150         }
151
152         data[key] = value;
153       });
154
155       // Insérer les données dans Elasticsearch
156       insertDataToElasticsearch(deviceType.toLowerCase(), data);
157     }
158   });
159
160   session.on('error', function (error) {
161     console.error(`Erreur SNMP lors de la connexion à ${ip} (${deviceName}):`, error.toString());
162   });
163 }
164
165 // * Exécuter le job pour tous les appareils
166 const runJob = () => {
167   console.log(`Début de la collecte SNMP : ${new Date().toISOString()}`);
168   Object.keys(deviceMapping).forEach(deviceType => {
169     Object.keys(deviceMapping[deviceType].devices).forEach(ip => {
170       fetchDeviceData(deviceType, ip);
171     });
172   });
173 };
174
175 // * Planification du job CRON toutes les 5m
176 cron.schedule('* * * * *', () => {
177   console.log('Exécution du job CRON toutes les 5 minutes');
178   runJob();
179 });
180
181 // * Exécution immédiate au lancement du script
182 runJob();

```

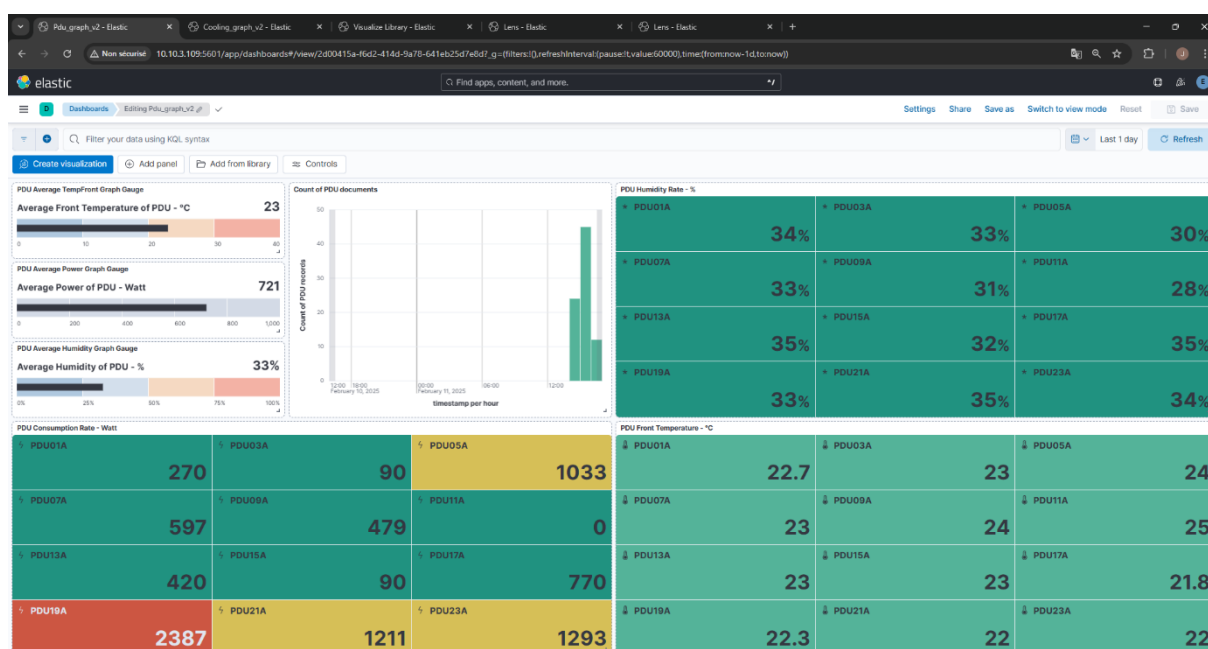
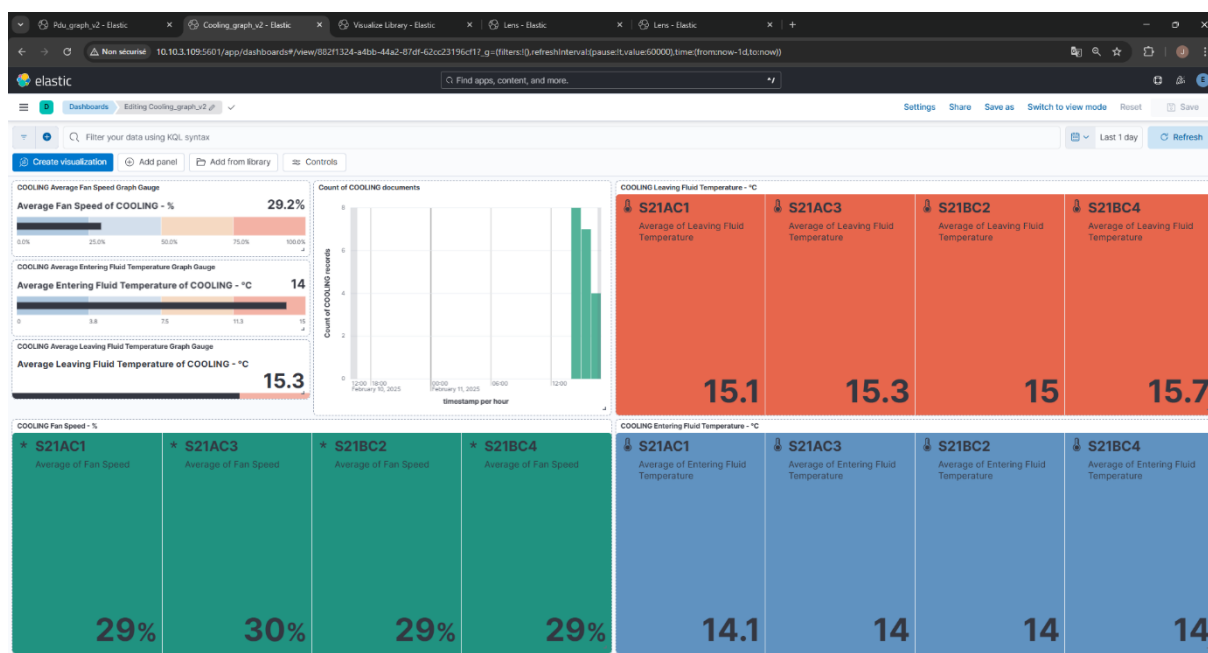
C. Visualisation avec Kibana

J'ai créé **deux tableaux de bord dans Kibana** pour **surveiller** les données collectées.

Le premier tableau de bord est dédié aux systèmes de refroidissement et affiche des métriques clés telles que la température des fluides et la vitesse des ventilateurs.

Le second tableau de bord est consacré aux unités de distribution d'alimentation (PDUs) et présente des informations sur la consommation d'énergie, la température, et l'humidité.

Chaque tableau de bord comprend **7 visualisations**, offrant une **vue d'ensemble complète et en temps réel des performances des appareils**.



II. Explication générale

A. Explication du script index.js :

1. Configuration Initiale

- **Variables d'environnement :**
 - Le script utilise dotenv pour charger des variables d'environnement à partir de fichiers .env et .env.local. Ces fichiers contiennent des informations sensibles comme les URL, les identifiants et les mots de passe.
- **Configuration TLS et OpenSSL :**
 - Le script ignore les erreurs TLS et utilise une version spécifique d'OpenSSL pour éviter les problèmes de compatibilité.

2. Connexion à Elasticsearch

- **Client Elasticsearch :**
 - Un client Elasticsearch est créé en utilisant les informations d'authentification et l'URL fournies dans les variables d'environnement.

3. Gestion des Index Elasticsearch

- **Nom de l'index :**
 - La fonction getCurrentIndexName génère un nom d'index basé sur le type d'appareil et la date actuelle (mois et année).
- **Création de l'index :**
 - La fonction ensureIndexExists vérifie si un index existe dans Elasticsearch et le crée s'il n'existe pas.

4. Insertion des Données dans Elasticsearch

- **Fonction insertDataToElasticsearch :**
 - Cette fonction insère les données collectées dans l'index approprié d'Elasticsearch.

5. Configuration des Appareils

- **Mapping des appareils :**
 - Le script utilise un objet deviceMapping pour mapper les types d'appareils (PDU, COOLING) à leurs adresses IP et OIDs (Object Identifiers) SNMP.

- **Parsing des adresses IP :**

- La fonction `parseDevices` extrait les adresses IP des appareils à partir des variables d'environnement.

6. Collecte des Données SNMP

- **Fonction** `fetchDeviceData` :

- Cette fonction récupère les données SNMP pour un appareil donné en utilisant les OIDs spécifiés.
- Elle utilise les configurations SNMP appropriées (protocole d'authentification et de chiffrement) pour chaque type d'appareil.

- **Traitement des données :**

- Les valeurs récupérées sont traitées (par exemple, conversion de valeurs brutes en pourcentages ou en unités appropriées) avant d'être insérées dans Elasticsearch.

7. Exécution du Job

- **Fonction** `runJob` :

- Cette fonction exécute la collecte de données pour tous les appareils configurés.

- **Planification CRON :**

- Le script utilise `node-cron` pour planifier l'exécution de `runJob` toutes les 5 minutes.

8. Exécution Immédiate

- **Exécution initiale :**

- Le job est exécuté immédiatement lors du lancement du script pour commencer la collecte de données sans attendre le prochain cycle CRON.

Points Clés :

- **Modularité** : Le script est modulaire, ce qui permet d'ajouter facilement de nouveaux types d'appareils ou de modifier les configurations existantes.
- **Robustesse** : Il gère les erreurs potentielles lors de la connexion SNMP et de l'insertion dans Elasticsearch.
- **Automatisation** : L'utilisation de CRON permet une collecte de données régulière et automatisée.

B. Explication d'un index dans ELK

📌 1. Création automatique de l'index

Insertion d'un document avec `client.indices.exists()`, Elasticsearch :

- ✓ Vérifie si l'index existe.
- ✓ **S'il n'existe pas**, il le crée automatiquement.
- ✓ Insère le document dans cet index.

Exemple de l'insertion dans le code (ligne 38) : voir la fonction `ensureIndexExists()`

```
38      await client.indices.create({ index: indexName });
```

Ici, si `{deviceType}_data_{month}_{year}` n'existe pas, Elasticsearch le crée tout.

📌 2. Détection automatique des types de propriétés

Quand Elasticsearch crée un index **automatiquement**, il **devine** le type des champs du premier document qu'il reçoit (**mapping dynamique**).

📌 Elasticsearch va interpréter les types ainsi :

Voir différence entre les type d'entier (long ,integer,...) : [Leçon types entiers](#)

Les propriétés communes aux 2 type d'appareil sont :

Propriétés (ou champs) et sous-champs	Type détecté (ou mapping)
device_ip	Text, sous-champ = keyword
device_name	Text, sous-champ = keyword
timestamp	date si bien formaté (sinon text)

Ensuite, les propriétés seront associées en fonction du type de l'appareil en + des propriétés communes :

📌 PDUs :

Propriétés (ou champs) et sous-champs	Type détecté (ou mapping)
humidity	float
tempFront	long
power	long

📌 COOLINGs :

Propriétés (ou champs) et sous-champs	Type détecté (ou mapping)
fanSpeed	float
enteringFluidTemp	float
leavingFluidTemp	float

⚠ Problème potentiel :

- **date** risque d'être détecté comme text au lieu de date, ce qui peut poser problème pour les recherches par date.
- Pour éviter ce souci, il faut **prédéfinir** l'index avec un **mapping**.

📌 3. Forcer un mapping personnalisé (optionnel/déconseillé)

Pour que **Elasticsearch ne choisisse pas les types automatiquement**, il faut créer l'index avec un **mapping précis** AVANT d'insérer des données (soit par nodejs soit directement dans Kibana), exemple dans la console DEV TOOLS de Kibana :

```

1  PUT /pdu_data_02_2025
2  {
3    "mappings": {
4      "properties": {
5        "device_name": {"type": "text"},
6        "device_ip": {"type": "keyword"},
7        "timestamp": {"type": "date"},
8        "humidity": {"type": "float"},
9        "tempFront": {"type": "float"},
10       "power": {"type": "float"}
11     }
12   }
13 }
```

📌 4. Vérifier le mapping d'un index dans ELK

Pour consulter les types d'un index :	1 GET pdu_data_02_2025
Extrait d'affichage :	<pre> 1 { 2 "pdu_data_02_2025": { 3 "aliases": {}, 4 "mappings": { 5 "properties": { 6 "device_ip": { 7 "type": "text", 8 "fields": { 9 "keyword": { 10 "type": "keyword", 11 "ignore_above": 256 12 } 13 } 14 }, 15 "device_name": { 16 "type": "text", 17 "fields": { 18 "keyword": { 19 "type": "keyword", 20 "ignore_above": 256 21 } 22 } 23 } 24 } 25 } 26 }</pre> Etc....

5. Rappel des termes employés

Voici comment on pourrait organiser les termes :

- **index** : pdu_data_02_2025
- **document** : chaque enregistrement JSON avec des propriétés comme device_ip, device_name, etc...
- **propriétés** (ou champs) : device_ip, device_name, etc...
- **mapping** : description de chaque propriété et de son type

Ressources :

1 - C'est quoi un UPS ?

UPS = **Uninterruptible Power Supply** = **Onduleur** en français.

- 👉 C'est un appareil qui sert à **fournir du courant en cas de coupure**.
- 👉 Il protège aussi les équipements contre les surtensions.

Exemple :

- Si ton data center **perd l'électricité**, l'UPS continue d'alimenter les serveurs pendant quelques minutes avec ses batteries.
 - Il peut aussi **stabiliser le courant** pour éviter les problèmes.
-

2 - C'est quoi un PDU ?

PDU = **Power Distribution Unit** = **Unité de Distribution Électrique**.

- 👉 C'est une **multiprise intelligente** pour data centers.
- 👉 Il distribue l'électricité des UPS vers les serveurs.
- 👉 Il peut mesurer la consommation électrique **de chaque prise** (suivi des kW consommés).

Exemple :

- Un **UPS** alimente plusieurs **PDU**.
 - Chaque **PDU** alimente **plusieurs serveurs** et équipements réseau.
 - Un **PDU intelligent** permet de **suivre la consommation** des serveurs à distance.
-

3 - C'est quoi un OID ?

OID = **Object Identifier** = **Identifiant d'Objet** SNMP.

- 👉 C'est une **adresse unique** qui permet de récupérer une info via SNMP.
- 👉 Chaque équipement réseau (UPS, PDU, switch, serveur...) a ses propres **OIDs** pour chaque donnée qu'il surveille.

Exemple :

- L'OID **.1.3.6.1.4.1.318.1.1.1.4.2.8** → Donne la **consommation totale** d'un UPS APC.
- L'OID **.1.3.6.1.2.1.1.1.0** → Donne le **modèle de l'équipement** interrogé.

En gros, un OID est comme une "adresse" qui te permet de demander une donnée spécifique à un équipement.

4 - C'est quoi SNMP ?

SNMP = **Simple Network Management Protocol** = **Protocole de gestion réseau**.

- 👉 C'est une **façon de communiquer** avec des équipements pour récupérer des infos à distance.
 - 👉 Il utilise **des OIDs** pour poser des questions et récupérer des valeurs.
-

5 – Résumé simple

Terme	Définition	Rôle
UPS (Onduleur)	Alimente les équipements en cas de coupure	Évite les pannes de courant
PDU (Multiprise intelligente)	Distribue le courant des UPS vers les serveurs	Suivi et gestion de la consommation
OID (Identifiant d'objet)	Adresse unique pour récupérer une info via SNMP	Permet d'obtenir des données précises
SNMP (Protocole réseau)	Sert à interroger des équipements (UPS, PDU, etc.)	Communication et récupération des infos