

# 学位論文

太陽光発電データの時刻補正手法と  
Elasticsearch ノードのクラスタ化手法の提案

提出年月日 令和 6 年 1 月 31 日

指導教員 都築 伸二 教授

入学年度 令和 4 年

学科名 電子情報工学専攻

論文提出者 祖父江 匠真

# 内容梗概

本論文は、筆者が愛媛大学大学院理工学研究科電子情報工学専攻電気電子工学コースに在学中に行った、太陽光発電データの時刻補正手法と Elasticsearch ノードのクラスタ化手法の提案についてまとめたものであり、以下の5章から構成されている。

## 第1章 緒論

本研究を行うに至った経緯及び、本研究の目的について述べている。

## 第2章 太陽光発電データの時刻補正手法

ここでは、太陽光発電データの時間的ずれを特定するための相互相関を用いた時刻補正手法について述べており、大気外日射量や地表日射量と実測データのと比較や、前処理を施した実測データとの比較を通じて、時刻補正の効果とその必要性を示す。

## 第3章 学内ゾーンの Elasticsearch クラスタへのデータ移行

ここでは、Elasticsearch を使用した学内ゾーンのデータ移行手順と重複データの削除方法について述べており、CO<sub>2</sub> データの移行と LEAF の運行日誌のデータ移行プロセスを詳述し、移行後のデータ可視化による成功の確認を行っている。

## 第4章 サーバーゾーンでのクラスタ構築

ここでは、異なるバージョンの Elasticsearch ノードを用いたクラスタリングの検証と、既存クラスタへのノードの参加検証について述べている。

## 第5章 結論

本研究によって明らかになった事項や今後の研究課題についてまとめている。

# 目次

内容梗概	I
第 1 章 緒論	1
第 2 章 太陽光発電データの時刻補正手法	3
2.1 節 緒言	3
2.2 節 太陽光発電の計測データの問題点について	3
2.3 節 大気外日射量の計算式	3
2.3.1 実測データと大気外日射量との比較	5
2.3.2 相互相関による時刻補正法	5
2.4 節 地表日射量の予測	6
2.4.1 pvlib の概要	6
2.4.2 実測データと pvlib により求まる地表日射量の比較	6
2.4.3 地表日射量との相互相関による時刻補正法	7
2.5 節 前処理の追加による時刻補正法とその精度	8
2.6 節 結言	9
第 3 章 単一 Elasticsearch ノードをクラスタ化する前に行うデータ移行	10
3.1 節 緒言	10
3.2 節 Elasticsearch の概要	10
3.3 節 Kibana の概要	11
3.4 節 データ移行対象の Elasticsearch インデックスについて	12
3.4.1 CO <sub>2</sub> データ	12
3.4.2 LEAF の運行日誌に関するデータ	12

3.5 節	CO <sub>2</sub> データの移行手順について . . . . .	13
3.5.1	データのエクスポート . . . . .	14
3.5.2	データの重複削除 . . . . .	14
3.5.3	データのインポート . . . . .	16
3.6 節	一度目のデータ移行で移行できなかった CO <sub>2</sub> データの 移行について . . . . .	16
3.7 節	Kibana による CO <sub>2</sub> データの可視化 . . . . .	17
3.8 節	LEAF の運行日誌に関するデータの移行手順について . . . . .	18
3.8.1	データのエクスポート . . . . .	18
3.8.2	データのインポート . . . . .	19
3.9 節	結言 . . . . .	19
第 4 章	仮想環境を使用したクラスタリング動作の検証 . . . . .	21
4.1 節	緒言 . . . . .	21
4.2 節	Docker とは . . . . .	21
4.2.1	コンテナとは . . . . .	21
4.2.2	Docker イメージとは . . . . .	22
4.3 節	Docker Compose とは . . . . .	24
4.4 節	異なるバージョンの Elasticsearch ノードを用いたクラ スタ構築検証 . . . . .	24
4.4.1	全て同じバージョンの Elasticsearch を使用した クラスタ構成 (全ノード バージョン 7.17.9) . . . . .	24
4.4.2	異なるバージョンの Elasticsearch を使用したク ラスタ構成 (2 ノード バージョン 7.17.9, 1 ノー ド バージョン 7.17.6) . . . . .	24
4.5 節	異なる Elasticsearch クラスタへのノード参加検証 . . . . .	27
4.5.1	単一ノードで稼働するクラスタ A の構築 . . . . .	27
4.5.2	3 ノードで稼働するクラスタ B の構築 . . . . .	29
4.5.3	クラスタ A に参加しているノードのクラスタ B への参加試行 . . . . .	29
4.6 節	結言 . . . . .	34

第 5 章	結論と今後の課題	35
5.1 節	結論 . . . . .	35
5.2 節	今後の課題 . . . . .	35
謝 辞		37
参考文献		38

# 第1章

## 緒論

太陽光発電は、再生可能エネルギー源として世界中で注目されており、効率的な運用と管理には正確な計測データが不可欠である。しかしながら、本研究で管理している太陽光発電データは、データを計測しているPCの内部時計が標準時刻とずれているため、他地点で計測しているデータと時刻同期ができない問題があった。

本論文では、この問題に対処するため、太陽光発電データの計測日時の補正手法を提案する。また、pvlibと呼ばれる太陽光発電シュミレーターライブラリを導入して、計測データを前処理することによる補正誤差の改善手法についても提案する。次に、発電データ等を蓄積しているElasticsearchサーバをクラスタ化して故障耐性を向上する際に必要な作業の実施方法を提案する。単一で動いているサーバのデータをクラスタ化サーバにするためのプロセスについて提案する。バージョンの異なるElasticsearchノードを用いたクラスタ構築の可否や、既存Elasticsearchノードの異なるクラスタへの参加の可否の検証結果も述べる。

本論文は以下の構成となっている。第2章では、太陽光発電データの計測日時を標準時に補正するために、理論値との相互相関を用いる手法を提案する。第3章では、単一で動いているサーバのデータをクラスタ化するためのプロセスについて提案する。第4章では、バージョンの異なるElasticsearchノードを用いたクラスタ構築の可否や、既存Elasticsearchノードの異なるクラスタへの参加の可否の検証結果も述べる。第5章では、本研究の成果と課題を明らかに

する.



## 第2章

# 太陽光発電データの時刻補正手法

### 2.1 節 緒言

本章では、太陽光発電データの計測日時の補正手法を提案する。

### 2.2 節 太陽光発電の計測データの問題点について

本研究で管理している太陽光発電データは、データを計測している PC の内部時計が標準時刻とずれているため、他地点で計測しているデータと時刻同期ができない問題があった。

そこで、時刻がずれていない実測データと、計算式により求まる大気外日射量との間の時間的遅延の秒数を相互相関を用いて求めることで、実測データを標準時に補正する。

### 2.3 節 大気外日射量の計算式

任意の緯度経度、日時における大気外日射量  $Q$  は、任意の緯度  $\phi$ 、経度  $\lambda$  の地点における任意の日時、太陽高度  $\alpha$  から求めることができる。

まず、次式より元旦からの通し日数  $dn$  に基いて定めた  $\theta$  を用いて、当該日の

太陽赤緯  $\delta$ , 地心太陽距離  $\frac{r}{r^*}$ , 均時差  $E_q$  をそれぞれ以下の式により求める.

$$\theta = \frac{2\pi(dn - 1)}{365} \quad (2.1)$$

$$\begin{aligned} \delta = & 0.006918 - 0.399912 \cos \theta + 0.070257 \sin \theta - 0.006758 \cos 2\theta \\ & + 0.000907 \sin 2\theta - 0.002697 \cos 3\theta + 0.001480 \sin 3\theta \end{aligned} \quad (2.2)$$

$$\begin{aligned} \frac{r}{r^*} & \quad (2.3) \\ = & \frac{1}{\sqrt{1.000110 + 0.034221 \cos \theta + 0.001280 \sin \theta + 0.000719 \cos 2\theta + 0.000077 \sin 2\theta}} \end{aligned}$$

$$\begin{aligned} E_q = & 0.000075 + 0.001868 \cos \theta - 0.032077 \sin \theta \\ & - 0.014615 \cos 2\theta - 0.040849 \sin 2\theta \end{aligned} \quad (2.4)$$

日本標準時間から, 太陽の時角  $h$  を求める.

$$h = \frac{(\text{日本標準時間} - 12)\pi}{12} + \text{標準子午線からの経度差} + E_q \quad (2.5)$$

$\delta, \phi, h$  の値が既知となったので  $\alpha$  は

$$\alpha = \arcsin(\sin \phi \sin \delta + \cos \phi \cos \delta \cos h) \quad (2.6)$$

により求まる.

最後に,  $Q$  が

$$Q = 1367 \left( \frac{r^*}{r} \right)^2 \sin \alpha \quad (2.7)$$

により求まる.  $1367\text{W/m}^2$  は太陽定数である.

式 (2.1) ~ 式 (2.7) を用いることで, 任意の緯度経度, 日時における大気外日射量が求まる.

### 2.3.1 実測データと大気外日射量との比較

Elasticsearch サーバーから取得した 2022 年 6 月 2 日の実測データと、計算式から求めた大気外日射量をプロットしたものを図 2.1 に示す。実測データは地表に設置された太陽光パネルが計測したものであるのに対して、大気外日射量は地球大気の上端（約 8km 上空）で受け取る日射量を計算したものであるため、一日を通して最大となる日射量の大きさに差がある。

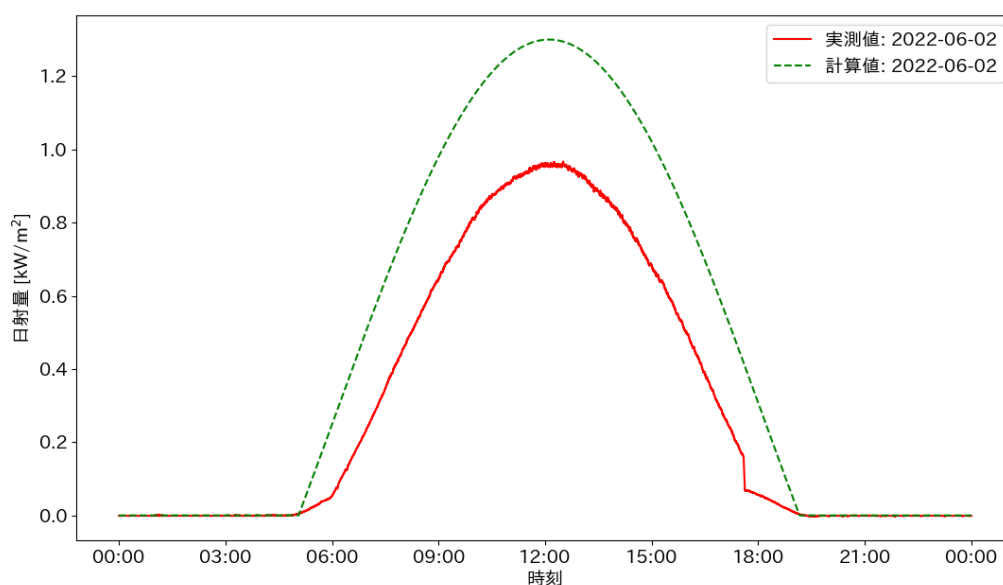


図 2.1 2022 年 6 月 2 日の日射量の実測データと大気外日射量をプロットしたもの

### 2.3.2 相互相関による時刻補正法

今回、相互相関の計算に使用する期間を、2022 年 6 月 2 日 0 時 0 分から 2022 年 6 月 2 日 23 時 59 分までとする。

相互相関の計算に使用する実測データの計測日時から大気外日射量を求め、実測データとの相互相関を計算する。

相互相関を計算した結果、実測データを124秒遅らせた際に、相関が最大となった。

今回使用した実測データには計測日時のずれは殆どないため、実測データを遅らせていない際に相関が最大となるのが正しい。

これは、相互相関を計算する際に地表日射量ではなく大気外日射量を使用していることが原因であると考えられる。

## 2.4 節 地表日射量の予測

地表日射量の予測を行うため、式(2.1)～式(2.7)を使った方法ではなく、pvlibライブラリを使用して地表日射量を求め、相互相関を計算する。

### 2.4.1 pvlib の概要

pvlib は、太陽光発電システムの性能シミュレーションや関連するタスクを実行するための関数とクラスのセットを提供するライブラリである。

以下は、pvlib の主な特徴である。

- 太陽位置計算: pvlib は、地球上の任意の場所における太陽の位置を計算する機能を提供する。これは、太陽の方位角や高度角を求めるのに使用される。
- 大気透過モデル: 大気を通過する太陽放射の量や質を推定するモデルが含まれている。
- 太陽光発電システムの性能モデリング: 太陽光発電モジュールやインバーターの性能モデルが含まれており、異なる条件下での太陽光発電システムの出力をシミュレートできる。

### 2.4.2 実測データと pvlib により求まる地表日射量の比較

Elasticsearch サーバーから取得した実測データと、pvlib を用いて求めた地表日射量をプロットしたものを図 2.2 に示す。

図 2.2 と図 2.1 と比較すると、大気外日射量より地表日射量の方が実測データにより近い概形となっていることが分かる。

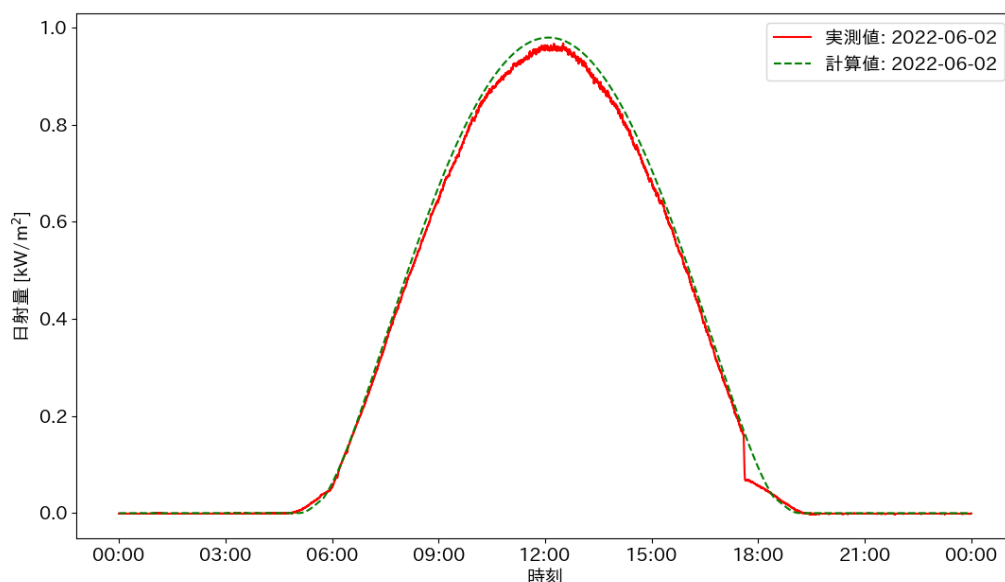


図 2.2 2022 年 6 月 2 日の実測データと地表日射量をプロットしたもの

### 2.4.3 地表日射量との相互相関による時刻補正法

今回、相互相関の計算に使用する期間を、2022 年 6 月 2 日 0 時 0 分から 2022 年 6 月 2 日 23 時 59 分までとする。

実測データの計測日時から地表日射量を求め、これらの値から相互相関を計算する。

相互相関を求めた結果、実測データ 74 秒遅らせた際に、相関が最大となることが分かった。

式 (2.1) ~ 式 (2.7) より求めた大気外日射量を用いて相互相関を計算した時と比較して、124 秒から 74 秒へと 50 秒改善した。

## 2.5 節 前処理の追加による時刻補正法とその精度

図 2.2 では、日没の辺りにおいて、実測データと地表日射量の概形が大きく異なっている。

太陽光パネルの周囲にある建造物や、天候といった外部要因による実測データのひずみを事前に取り除いた上で相互相関を計算することで、相互相関の計算結果が改善するか検証する。そこで、実測データに対して前処理を追加する。前処理を含めた相互相関の計算方法は以下の手順で行う。

1. 実測データの日射量を  $0 \text{ kW/m}^2$  と見なすしきい値の指定: まず、実測データをフィルタリングするために日射量のしきい値を設定する。今回は、 $0.2 \text{ kW/m}^2$  をしきい値として設定する。
2. しきい値に該当する計測日時の特定: 続いて、実測データの各データ点から  $0.2 \text{ kW/m}^2$  を減算して絶対値を取った際に最も 0 に近い値を取る計測日時を午前と午後でそれぞれ一点ずつ特定する。
3. 特定した計測日時を使った実測データのフィルタリング: 前のステップで得た計測日時を使用して、2 点の計測日時の外側にある実測データの日射量を  $0 \text{ kW/m}^2$  とする。
4. 地表日射量との相互相関の計算: 実測データをフィルタリングした後、地表日射量との相互相関を計算する。

図 2.3 に、上述の前処理を行った実測データと、地表日射量をプロットしたものを示す。

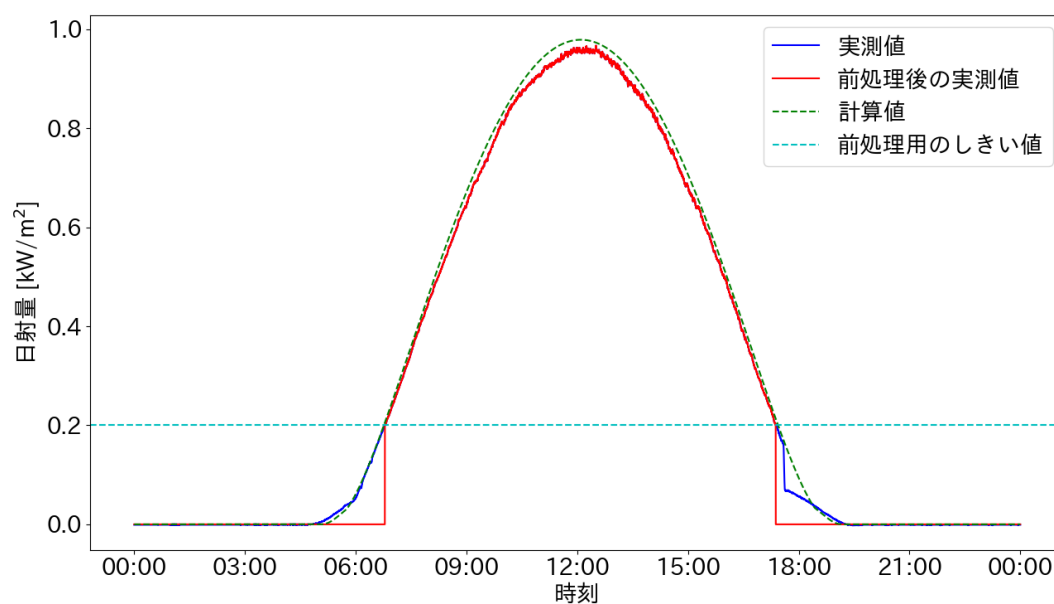


図 2.3 前処理を行った実測データと、地表日射量をプロットしたもの

前処理を行った実測データと、地表日射量から相互相関を計算した結果、実測データを 27 秒遅らせた際に、相関が最大となった。

前処理を追加したことで相互相関の結果が、74 秒から 27 秒へと 47 秒改善した。

## 2.6 節 結言

本章では太陽光発電データの計測時刻の補正手法を提案した。次章では学内ゾーンで稼働している Elasticsearch クラスタへのデータ移行について述べる。

## 第3章

# 単一Elasticsearch ノードをクラスタ化する前に行うデータ移行

### 3.1 節 緒言

本章では学内ゾーンで稼働している Elasticsearch クラスタへのデータ移行について述べる。

### 3.2 節 Elasticsearch の概要

Elasticsearch は、分散処理に対応した全文検索エンジンである。主な特徴は、以下の通りである。

- 高速な検索性能: ビッグデータなどの巨大で複雑なデータの集合にも対応可能
- 部分一致検索が可能: 検索キーワードの一部に一致するドキュメントも検索可能
- ほぼリアルタイムの検索: ドキュメントにインデックスを付けてから検索可能になるまで約 1 秒程度
- スケーラビリティ: サーバー数を増やすことで、検索性能と処理能力を拡張可能



これらの特徴から, Elasticsearch は, 以下のような用途に適している。

- ログ分析: Web サイトやアプリケーションのログから, アクセス状況やエラー情報を分析する
- セキュリティインテリジェンス: ネットワークやシステムから, セキュリティ脅威を検知する
- ビジネス分析: 顧客データや販売データから, トレンドや傾向を分析する

### 3.3 節 Kibana の概要

Kibana は, Elasticsearch に保存されたデータを可視化するためのツールである。主な特徴は, 以下の通りである。

- 直感的な操作性: ドラッグ&ドロップで簡単に可視化を作成できる
- 豊富な可視化機能: グラフ, 表, 地図など, さまざまな可視化機能を提供
- 高度なフィルタリング機能: 条件を指定して, データを詳細に絞り込むことができる

これらの特徴から, Kibana は, 以下のような用途に適している。

- ログ分析: Web サイトやアプリケーションのログから, アクセス状況やエラー情報を可視化する
- セキュリティインテリジェンス: ネットワークやシステムから, セキュリティ脅威を可視化する
- ビジネス分析: 顧客データや販売データから, トレンドや傾向を可視化する

## 3.4 節 データ移行対象のElasticsearch インデックスについて

133.71.106.168 で稼働している単一ノードの Elasticsearch に保存された CO<sub>2</sub> データと LEAF の運行日誌に関するデータを、学内ゾーンで稼働している Elasticsearch クラスタへ移行する。

### 3.4.1 CO<sub>2</sub> データ

CO<sub>2</sub> データが保存されたインデックスは、インデックス名に co2 という文字列が含まれているため、co2 という文字列を含む全てのインデックスを移行対象とする。

### 3.4.2 LEAF の運行日誌に関するデータ

LEAF の運行日誌に関するデータが保存されたインデックスは以下の2つである。

- movement\_diary
- movement\_diary01

上記のインデックスに保存されているデータについて説明する。

以下に movement\_diary と movement\_diary01 のドキュメントの違いを列挙する。

#### 1. driver フィールド:

- movement\_diary のドキュメントでは、driver フィールドは文字列である。
- movement\_diary01 のドキュメントでは、driver フィールドは配列で、その中に文字列と2つの null 値が含まれている。

#### 2. “destination” フィールド:

- movement\_diary のドキュメントでは、“destination” フィールドは単一の文字列である。

- movement\_diary01 のドキュメントでは, “destination” フィールドは配列で, その中に 2 つの文字列が含まれている.

3. “charge\_place” フィールド:

- movement\_diary のドキュメントには, “charge\_place” フィールドは存在しない.
- movement\_diary01 のドキュメントでは, “charge\_place” フィールドが追加されているが, その値は空文字列である.

4. “battery\_rate” フィールド:

- movement\_diary のドキュメントには, “battery\_rate” フィールドは存在しない.
- movement\_diary01 のドキュメントでは, “battery\_rate” フィールドが追加されており, その値は数値である.

5. “battery\_rate\_distance” フィールド:

- movement\_diary のドキュメントには, “battery\_rate\_distance” フィールドは存在しない.
- movement\_diary01 のドキュメントでは, “battery\_rate\_distance” フィールドが追加されており, その値は数値である.

上述した movement\_diary と movement\_diary01 のドキュメントの違いより, movement\_diary01 は movement\_diary のもつ全ての情報を保持しており, 更に movement\_diary にはないフィールドを持っている. 更に, movement\_diary と movement\_diary01 のドキュメント数は等しく, すべてのドキュメントのタイムスタンプが一致しているため, movement\_diary01 インデックスのみ移行する.

### 3.5 節 CO<sub>2</sub> データの移行手順について

まず, Elasticsearch に保存された CO<sub>2</sub> データの持つフィールドについて説明する. Elasticsearch に保存された CO<sub>2</sub> データには, 計測した日時, 部屋番号, 部屋の気温, CO<sub>2</sub> 濃度などの情報が含まれている.

CO<sub>2</sub> のデータ移行を行う上で、タイムスタンプと部屋番号の組み合わせが重複しているデータが一部存在しており、この重複データを取り除いた上でデータ移行を行う必要がある。そこで一度、移行元の ElasticSearch サーバーのデータをローカルマシンにエクスポートして、重複データを取り除いた上で、移行先の ElasticSearch サーバーにデータをインサートする。

### 3.5.1 データのエクスポート

移行元の ElasticSearch サーバーのデータのローカルマシンへのエクスポートには、`elasticdump` ライブラリを使用して、JSON 形式でエクスポートした。その際、`co2` という文字列を含むインデックスのデータのみをエクスポートした。

### 3.5.2 データの重複削除

重複データの削除は SQLite データベースを用いて行った。

SQLite は、軽量で自己完結型のデータベースエンジンである。これは、多くのアプリケーションに組み込まれており、以下の特徴を持っている。

- 軽量: SQLite は非常に小さく、リソースの少ない環境でも動作する。例えば、組み込みシステムやモバイルアプリケーションなどが挙げられる。
- 自己完結型: SQLite はサーバーレスで、設定や管理が不要である。これは、データベースが単一のファイルとして存在し、外部の依存関係がないことに起因する。
- トランザクション: SQLite は ACID トランザクションをサポートしており、データの整合性を保つ。
- フリーかつオープンソース: SQLite はパブリックドメインに属し、誰でも自由に使用、変更、配布可能である。

SQLite では、複合主キーを使って複数のテーブルカラムの組み合わせを一意の識別子として扱うことができる。これにより、同じ組み合わせのデータを重複して挿入しようとした場合、データベースエンジンがコンフリクトエラー

を発生させ、重複データの挿入を阻止する。そのため、今回の重複データ削除には適していると判断した。

今回使用した SQLite では、部屋番号 (number) とタイムスタンプ (utctime) を一意のキーとして設定した。以下のリスト 3.1, リスト 3.2 に示すように、移行元の ElasticSearch サーバーに保存されている co2 インデックスのドキュメントは、ドキュメントの持つフィールドが統一されておらず、一部のセンサー情報が存在しないドキュメントが存在する。そのため、データの挿入時にコンフリクトエラーが発生した場合は、既存のレコードと挿入しようとしたレコードを比較し、既存レコードの値が NULL であるカラムにおいて、挿入しようとしているレコードの値が非 NULL である場合には、既存レコードのカラムの値を更新するようにした。これにより、重複データ削除時に一部のセンサー情報などが欠けてしまう問題を解決した。

Listing 3.1 `_source` フィールドのメンバー数が少ないドキュメント

```
{
  "_index": "co2_e411",
  "_type": "_doc",
  "_id": "nEi2nnoB2 iFXnrMOobM",
  "_score": 1,
  "_source": {
    "utctime": "2020 10 09T05:09:06+00:00",
    "number": "E411",
    "PPM": "481",
    "data": "Thingspeak"
  }
}
```

Listing 3.2 `_source` フィールドのメンバー数が多いドキュメント

```
{
  "_index": "co2_e411",
```

```
"_type": "_doc",
"_id": "YKBqU4QBugDzeydA2gyi",
"_score": 1,
"_source": {
  "RH": 26.98,
  "PPM": 423,
  "JPtime": "2022 11 06 T22:45:30.080925",
  "ip": "172.23.68.19/16",
  "utctime": "2022 11 06 T13:45:30.080895",
  "TEMP": 24.47,
  "index_name": "co2_e411",
  "ms": "",
  "number": "E411"
}
```

### 3.5.3 データのインポート

重複データの削除を行った後のデータが保存されている SQLite からすべてのレコードを読み出して、ターゲットの Elasticsearch サーバーに移行した。

その際、python の elasticsearch ライブラリを使用し、co2\_modbus という名前のインデックスに保存した。

## 3.6 節 一度目のデータ移行で移行できなかった CO<sub>2</sub> データの移行について

2023 年 5 月中旬頃に、実装したデータ移行プログラムを使用して 133.71.106.168 の Elasticsearch から学内ゾーンの Elasticsearch クラスタへ CO<sub>2</sub> データ移行を行った。しかし、CO<sub>2</sub> 濃度監視システムを開発、運用している高木君が、移行先である学内ゾーンの Elasticsearch クラスタに対してラズベリーパイから CO<sub>2</sub> データのインサートを行うよう対応したのが 2023 年 7 月中旬頃であったため、

2023 年 5 月中旬から 2023 年 7 月中旬までの間の約 2 ヶ月間の CO<sub>2</sub> データが移行先の Elasticsearch サーバーに移行出来ていなかった。そこで、追加の移行作業を行った。

移行方法は以下のとおりである。

1. まず、2023 年 5 月中旬に移行した際の全ての移行データの中で最も最新の `utctime` フィールドの値を検索する。
  - 検索した結果、2023 年 5 月中旬に移行した際の全移行データの中で最も最新の `utctime` は「2023-05-16T05:48:30.081305」であった。
2. 次に、学内ゾーンの Elasticsearch クラスタに対して、ラズベリーパイからインサートした全データの中で最も古い `utctime` フィールドの値を検索する。
  - 検索した結果、ラズベリーパイからインサートされた全データの中で最も古い `utctime` は「2023-07-20T07:15:39.314008」であった。
3. 前回の CO<sub>2</sub> データの移行は 2023 年 5 月中旬頃に行ったため、2023 年 5 月 1 日 0 時 0 分 0 秒以降の `utctime` を持つドキュメントを、133.71.106.168 の Elasticsearch のインデックス名に `co2` という文字列を含むインデックスから `elasticsearch-dump` ライブラリを使用してローカルマシンにエクスポートする。
4. 部屋番号 (`number`) とタイムスタンプ (`utctime`) の組み合わせがユニークになるよう SQLite を用いてエクスポートデータから重複削除を行う。
5. 更に、1 と 2 で得られた `utctime` の範囲に含まれる `utctime` を持つドキュメントのみになるようフィルタリングする。
6. フィルタリング後のデータを移行先 Elasticsearch サーバーにバルクインサートする。

### 3.7 節 Kibana による CO<sub>2</sub> データの可視化

計 2 回の CO<sub>2</sub> データを移行した後の `co2_modbus` インデックスについて、横軸をタイムスタンプ (`utctime`) とし、縦軸を PPM, RH, TEMP としてそれぞれ

プロットしたものを図 3.1 ~ 図 3.3 に示す。

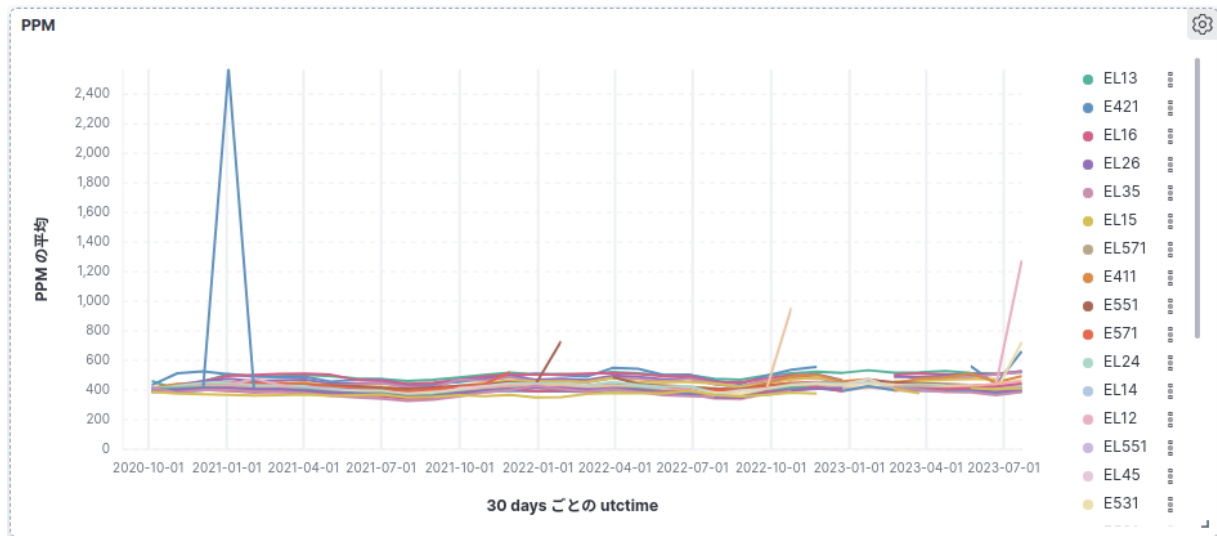


図 3.1 co2\_modbus の PPM

図 3.1 ~ 図 3.3 より、連続的にデータが変化していることが目視で確認できるので、データ移行は正常に出来たと判断できる。

## 3.8 節 LEAF の運行日誌に関するデータの移行手順について

movement\_diary01 インデックスのデータ移行は、同名のインデックスを移行先の ElasticSearch サーバーに作成して、作成したインデックスにデータを挿入することで行う。

### 3.8.1 データのエクスポート

移行元の ElasticSearch サーバーのデータのローカルマシンへのエクスポートには、elasticdump ライブラリを使用して、movement\_diary01 インデックスの全ドキュメントを JSON 形式でエクスポートした。



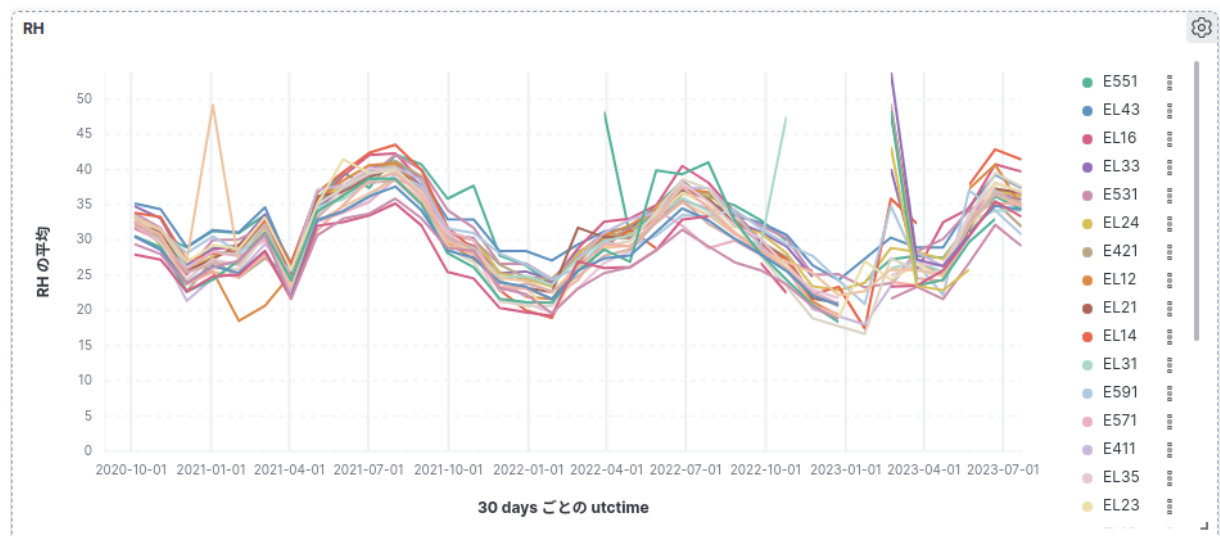


図 3.2 co2\_modbus の RH

### 3.8.2 データのインポート

python の elasticsearch ライブラリを使用し, 移行先の Elasticsearch に movement\_diary01 という名前のインデックスを作成して, エクスポートしたデータを全てインサートした.

## 3.9 節 結言

本章では学内ゾーンで稼働している Elasticsearch クラスタへのデータ移行について述べた。

次章ではサーバーゾーンでのクラスタ構築における仮想環境を使用した事前検証について述べる。

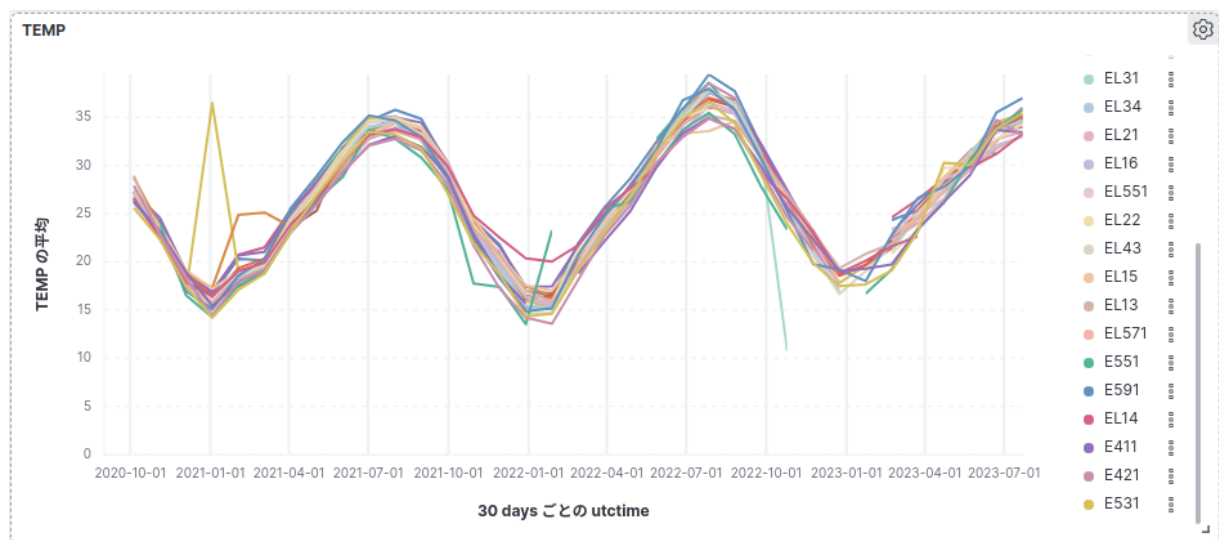


図 3.3 co2\_modbus の TEMP

## 第4章

# 仮想環境を使用したクラスタリング 動作の検証

### 4.1 節 緒言

本章では、サーバーゾーンでのクラスタ構築における仮想環境を使用した事前検証について述べる。

### 4.2 節 Docker とは

Docker は、軽量で独立したコンテナ型仮想環境用のプラットフォームである。従来の仮想化では、VMWare などの仮想化ソフトウェアを用いて、ホスト OS 上にゲスト OS を構築する形式だった。しかし、Docker はホスト OS 上にゲスト OS なしで独立したコンテナ型の仮想環境として構築される。Docker コンテナを利用する場合は、Docker Engine をインストールすることでコンテナの立ち上げ、停止、削除といった操作を行うことができる。

#### 4.2.1 コンテナとは

コンテナは、アプリケーションとそのすべての依存関係（ライブラリ、実行環境など）をカプセル化した軽量な実行単位である。Docker の場合、コンテナ

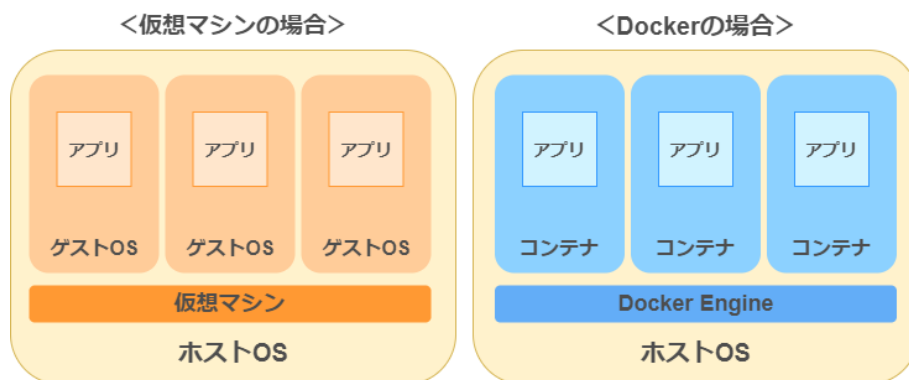


図 4.1 仮想マシンと Docker の違い [3]

の作成には Docker イメージが必要となる。

#### 4.2.2 Docker イメージとは

Docker イメージとは、Docker コンテナを作成するためのテンプレートであり、Docker イメージの中には、Docker コンテナの実行に必要な Linux ファイルシステムとメタ情報を含む。

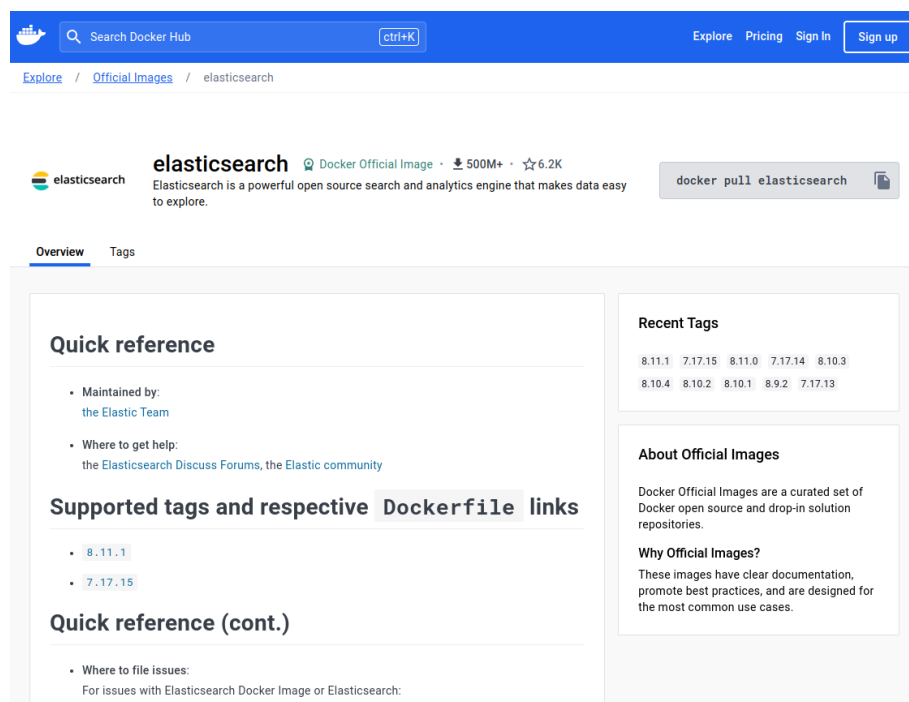
Linux ファイルシステムというのは、/ ディレクトリ以下の /etc /bin /sbin /usr などのディレクトリ階層およびファイルである。

Docker では、コンテナとして動かしたいアプリケーションが必要とする、最小限のファイルを Docker イメージの中に入れる。

さらに、そのアプリケーションを動かすために必要なデフォルトのコマンドや引数の指定、外に公開するポート番号の情報などの情報がある。これらをメタ情報として、同じく Docker イメージの中に入れられる。

Docker イメージは Docker Hub やその他のレジストリで共有されており、これらのサービスから取得することが可能である。

今回は Elasticsearch の開発元である Elastic 社が提供している Elasticsearch の Docker イメージを使って検証を行う。



## 図 4.2 Elasticsearch の Docker イメージ

## 4.3 節 Docker Compose とは

Docker Compose は、複数のコンテナを定義し、実行するためのツールである。これはYAML ファイルを使用して設定され、複数のコンテナで協調して動作するアプリケーションの開発を単純化する。

## 4.4 節 異なるバージョンの Elasticsearch ノードを用いたクラスタ構築検証

### 4.4.1 全て同じバージョンの Elasticsearch を使用したクラスタ構成 (全ノード バージョン 7.17.9)

図 4.3 に、7.17.9 バージョンの Elasticsearch のみを使用してクラスタを構築した時の docker-compose.yml を図で表現したものを示す。

クラスタの起動には、docker compose up -d コマンドを使用する。

docker compose up -d コマンドを実行した後、curl コマンドを使用してクラスタに参加しているノードを一覧表示した結果を図 ??に示す。

図 4.4 より、3 つのノード ( es01, es02, es03 ) すべてが正常にクラスタに参加できていることが確認できる。

### 4.4.2 異なるバージョンの Elasticsearch を使用したクラスタ構成 (2 ノード バージョン 7.17.9, 1 ノード バージョン 7.17.6)

図 4.3 の docker-compose.yml の es03 のコンテナが使用する Docker イメージを変更して、es03 のノードで使用する Elasticsearch のバージョンを 7.17.9 から 7.17.6 に変更する。

図 4.5 に変更後の docker-compose.yml を図で表現したものを示す。

変更後、docker compose up -d コマンドを実行してクラスタを起動する。

クラスタの起動後、curl コマンドを使用してクラスタに参加しているノードを一覧表示した結果を図 4.6 に示す。

図 4.6 より、バージョンが 7.17.9 である 2 つのノード ( es01, es02 ) のみが正



図 4.3 docker-compose.yml を図で表現したもの

```
$ curl -X GET "localhost:9200/_cat/nodes?v=true&pretty"
ip          heap.percent ram.percent cpu load_1m load_5m load_15m node.role master name
172.22.0.3  26         14 34    3.35   2.84    2.88 cdfhilmrstw *   es02
172.22.0.4  15         14 36    3.35   2.84    2.88 cdfhilmrstw -   es01
172.22.0.2  15         14 37    3.35   2.84    2.88 cdfhilmrstw -   es03
```

図 4.4 クラスタに参加しているノードを一覧表示した結果

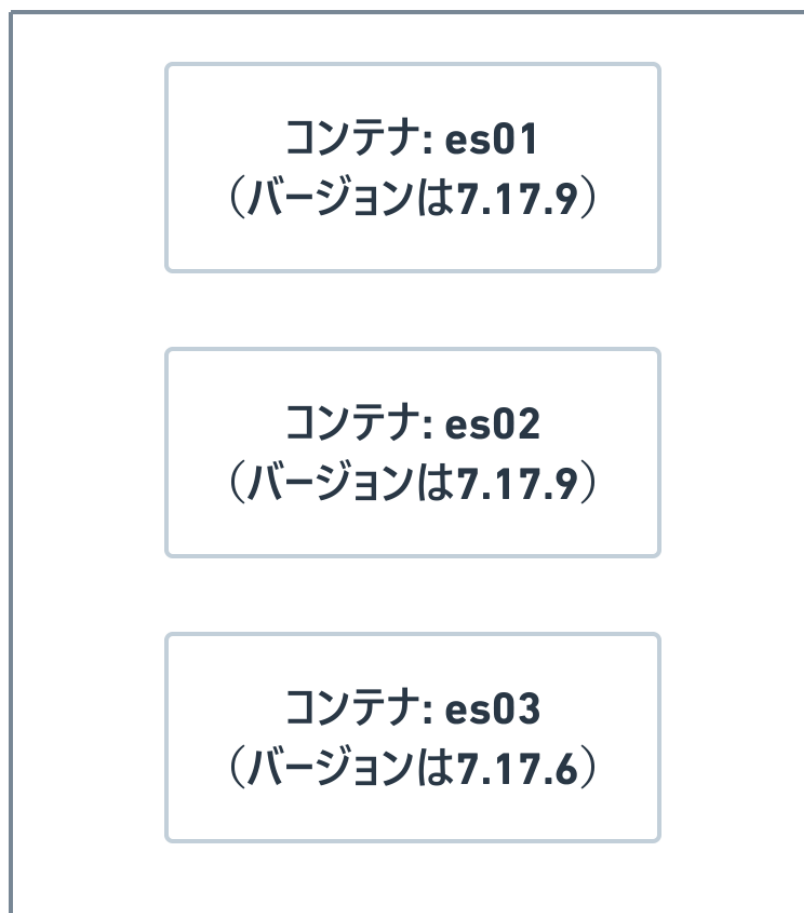


図 4.5 変更後の docker-compose.yml を図で表現したもの

```
$ curl -X GET "localhost:9200/_cat/nodes?v=true&pretty"
ip          heap.percent ram.percent cpu load_1m load_5m load_15m node.role master name
172.22.0.4   20           14      7    1.86   1.32   1.14 cdfhlmrstw -   es02
172.22.0.3   18           14      8    1.86   1.32   1.14 cdfhlmrstw *   es01
```

図 4.6 クラスタに参加しているノードを一覧表示した結果



[illegible]

常にクラスタに参加できていることが確認できる。

#### 4.5 節 異なる Elasticsearch クラスタへのノード参加検証

サーバーゾーンでのクラスタ構築において、リサイクル館の太陽光パネルの計測データを保存している Elasticsearch ノードを新たなノードとして構築したクラスタに参加できるか、Docker を用いて検証した。

まず, docker-compose を用いて単一ノード (コンテナ名は es04) でクラスタ  
以後このクラスタをクラスタ A と呼ぶを構築する. 以後このクラスタをクラ  
スタ A と呼ぶ.

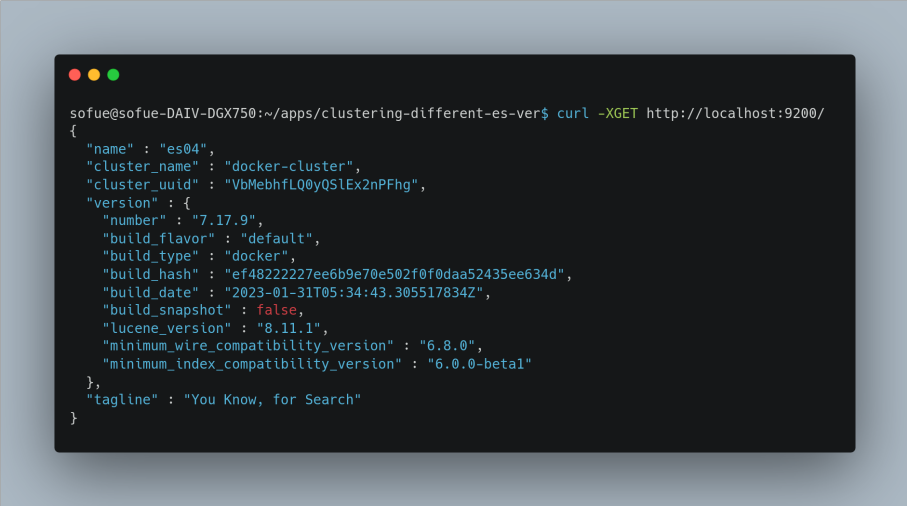
図 4.8 にクラスタ A の構築の際に使用した docker-compose.yml を図で表現したものを出す。

docker-compose を用いてノードを起動した後、クラスタの情報について問い合わせた結果を図 4.9 に示す。

クラスタの情報について問い合わせた後、Docker コンテナを停止してノードをシャットダウンした。

コンテナ: es04  
(バージョンは7.17.9)

図 4.8 クラスタ A の構築の際に使用した docker-compose.yml を図で表現したもの



```
sofue@sofue-DAIV-DGX750:~/apps/clustering-different-es-ver$ curl -XGET http://localhost:9200/
{
  "name" : "es04",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "VbMebhfLQ0yQSLEx2nPFhg",
  "version" : {
    "number" : "7.17.9",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "ef4822227ee6b9e70e502f0f0daa52435ee634d",
    "build_date" : "2023-01-31T05:34:43.305517834Z",
    "build_snapshot" : false,
    "lucene_version" : "8.11.1",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

図 4.9 クラスタの情報について問い合わせた結果

### 4.5.2 3 ノードで稼働するクラスタ B の構築

次に、クラスタ A に使用したノードとは別の 3 ノード (コンテナ名はそれぞれ es01, es02, es03) でクラスタを構築する。以後このクラスタをクラスタ B と呼ぶ。

図 4.10 にクラスタ B の構築の際に使用した docker-compose.yml を図で表現したものを示す。

docker-compose を用いて 3 つのノードを起動した後、クラスタの情報について問い合わせた結果を図 4.11 に示す。

図 4.9, 4.11 より、クラスタ A とクラスタ B はそれぞれ異なるクラスタ ID を付与されたことが分かる。

クラスタの起動後、クラスタに参加しているノードの一覧を取得した結果を図 4.12 に示す。

図 4.12 より、es01, es02, es03 ノードが全てクラスタ B に参加できていることが分かる。

クラスタに参加しているノードの一覧を取得した後、全ての Docker コンテナを停止してノードを全てシャットダウンした。

### 4.5.3 クラスタ A に参加しているノードのクラスタ B への参加試行

次に、図 4.10 の docker-compose.yml に対して、クラスタ A のノード (es04 コンテナ) を追加し、合計 4 ノードでのクラスタ B の起動を試みる。

図 4.13 に、合計 4 ノードでクラスタ B の起動を試みた際に使用した docker-compose.yml を図で表現したものを示す。

クラスタの起動後、クラスタに参加しているノードの一覧を取得した結果を図 4.14 に示す。

図 4.14 より、クラスタ A のノードがクラスタ B に参加できていないことが分かる。

es04 コンテナ (クラスタ A のノード) で出力されたログの一部を図 4.15 に示す。

図 4.15 には、異なるクラスタ ID を持つクラスタにノードが参加することは禁止されており、これを行うためにはインデックスやドキュメント情報などが

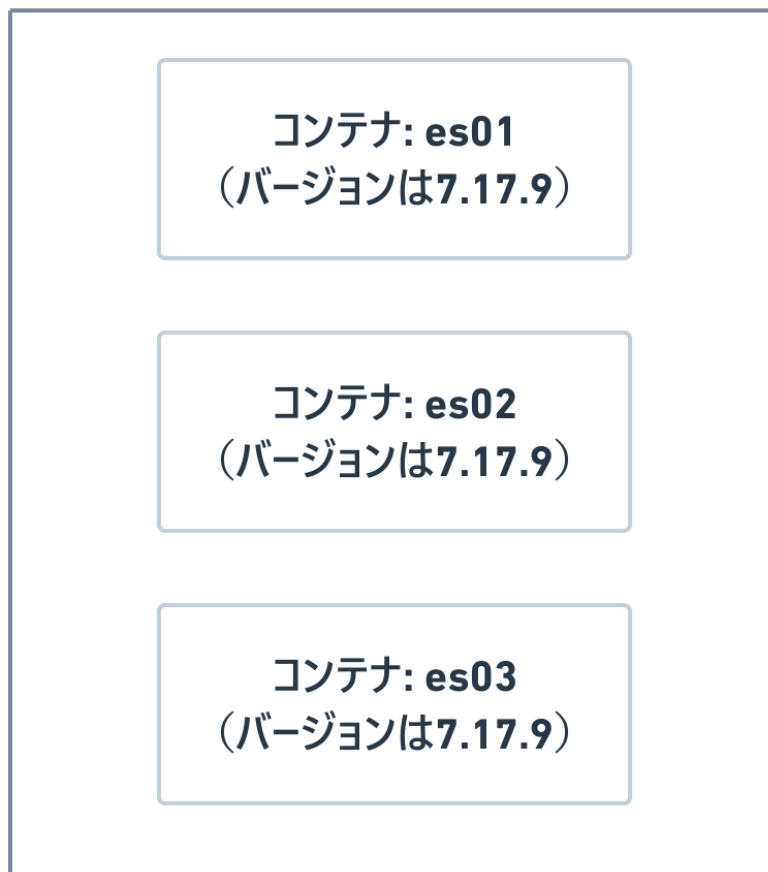
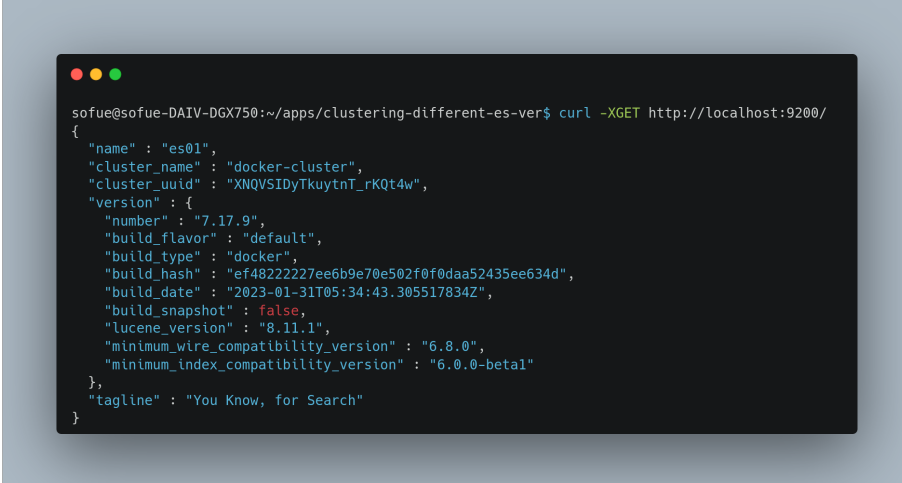
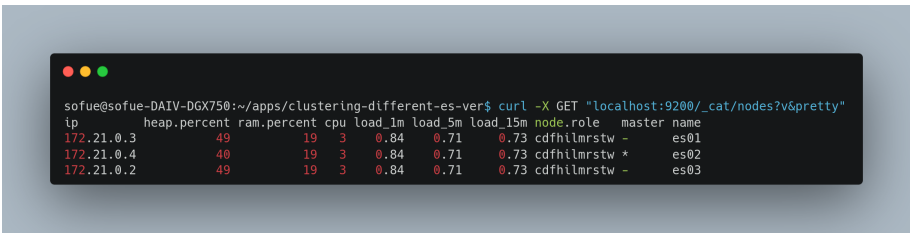


図 4.10 クラスタ B の構築の際に使用した docker-compose.yml を図で表現したもの



```
sofue@sofue-DAIV-DGX750:~/apps/clustering-different-es-ver$ curl -XGET http://localhost:9200/
{
  "name" : "es01",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "XNQVSIIDyTkuytnT_rKQt4w",
  "version" : {
    "number" : "7.17.9",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "ef4822227ee6b9e70e502f0f0daa52435ee634d",
    "build_date" : "2023-01-31T05:34:43.305517834Z",
    "build_snapshot" : false,
    "lucene_version" : "8.11.1",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

図 4.11 クラスタの情報について問い合わせた結果



```
sofue@sofue-DAIV-DGX750:~/apps/clustering-different-es-ver$ curl -X GET "localhost:9200/_cat/nodes?v&pretty"
ip          heap.percent ram.percent cpu load_1m load_5m load_15m node.role master name
172.21.0.3  49          19          3    0.84  0.71   0.73 cdhflmrstw -   es01
172.21.0.4  40          19          3    0.84  0.71   0.73 cdhflmrstw *   es02
172.21.0.2  49          19          3    0.84  0.71   0.73 cdhflmrstw -   es03
```

図 4.12 クラスタ B の起動後, クラスタに参加しているノードの一覧を取得した結果

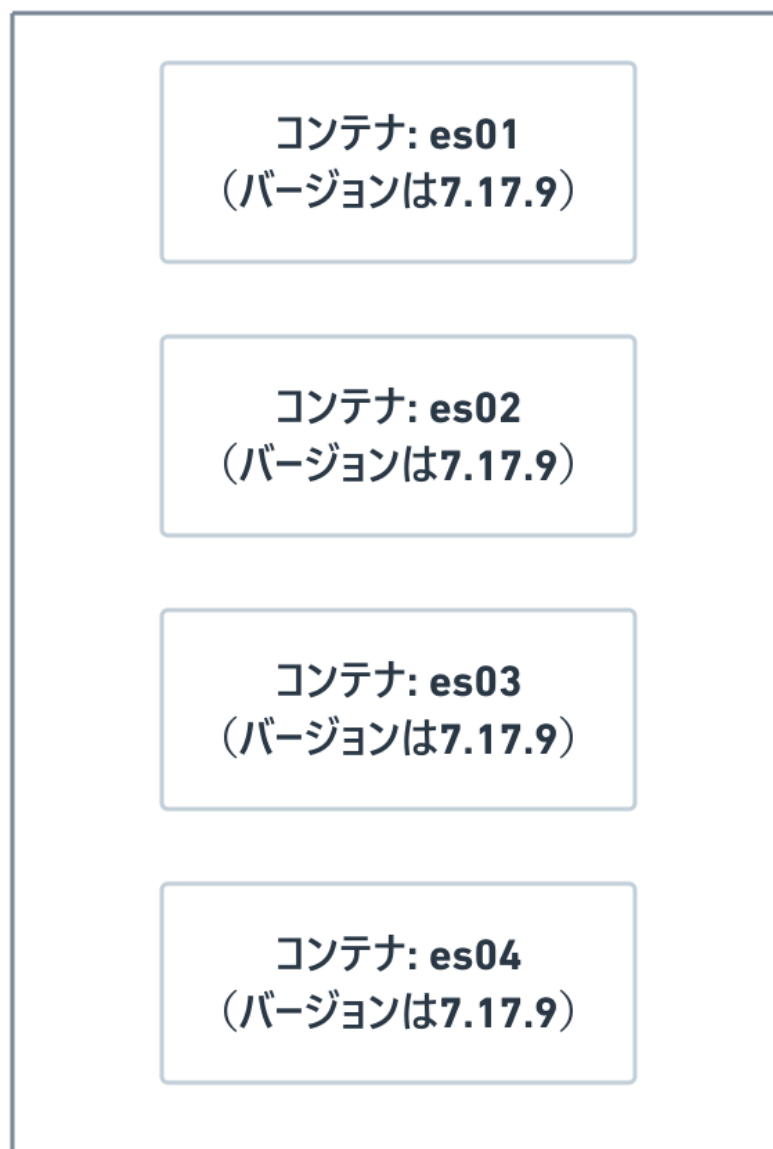


図 4.13 合計4ノードでクラスターBの起動を試みた際に使用した docker-compose.yml を図で表現したもの

```
sofue@sofue-DAIV-DGX750:~/apps/clustering-different-es-ver$ curl -X GET "localhost:9200/_cat/nodes?v&pretty"
ip      heap.percent ram.percent cpu load_1m load_5m load_15m node.role master name
172.22.0.4      18         21    5    0.75   1.24   1.02 cdfhlmrstw -   es02
172.22.0.2      16         21    5    0.75   1.24   1.02 cdfhlmrstw -   es01
172.22.0.5      33         21    5    0.75   1.24   1.02 cdfhlmrstw *   es03
```

図 4.14 合計4ノードでクラスタの起動を試みた後、クラスタに参加しているノードの一覧を取得した結果

```
es04 | "Caused by: org.elasticsearch.cluster.coordination.CoordinationStateRejectedException: This node previously joined a cluster with UUID [VbMebhFL00yQSlEx2nPfhg] and is now trying to join a different cluster with UUID [XNQVSiDyTkuytNrKQt4w]. This is forbidden and usually indicates an incorrect discovery or cluster bootstrapping configuration. Note that the cluster UUID persists across restarts and can only be changed by deleting the contents of the node's data paths [] which will also remove any data held by this node."
es04 | "at org.elasticsearch.cluster.coordination.JoinHelper.Lambda$new$8(JoinHelper.java:213) ~[elasticsearch-7.17.9.jar:7.17.9]",
es04 | "at org.elasticsearch.xpack.security.transport.SecurityServerTransportInterceptor$ProfileSecuredRequestHandler$1.doRun(SecurityServerTransportInterceptor.java:341) ~[?:?]",
es04 | "at org.elasticsearch.common.util.concurrent.AbstractRunnable.run(AbstractRunnable.java:26) ~[elasticsearch-7.17.9.jar:7.17.9]",
es04 | "at org.elasticsearch.xpack.security.transport.SecurityServerTransportInterceptor$ProfileSecuredRequestHandler.messageReceived(SecurityServerTransportInterceptor.java:417) ~[?:?]",
es04 | "at org.elasticsearch.transport.RequestHandlerRegistry.processMessageReceived(RequestHandlerRegistry.java:67) ~[elasticsearch-7.17.9.jar:7.17.9]",
es04 | "at org.elasticsearch.transport.InboundHandler$1.doRun(InboundHandler.java:272) ~[elasticsearch-7.17.9.jar:7.17.9]",
es04 | "at org.elasticsearch.common.util.concurrent.ThreadContext$ContextPreservingAbstractRunnable.doRun(ThreadContext.java:777) ~[elasticsearch-7.17.9.jar:7.17.9]",
es04 | "at org.elasticsearch.common.util.concurrent.AbstractRunnable.run(AbstractRunnable.java:26) ~[elasticsearch-7.17.9.jar:7.17.9]",
es04 | "at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1144) ~[?:?]",
es04 | "at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:642) ~[?:?]",
es04 | "at java.lang.Thread.run(Thread.java:1589) ~[?:?]" }
```

図 4.15 es04 コンテナのログ

格納されているデータパス配下のフォルダ、ファイルを削除する必要があると書かれている。

以上の検証結果から、既に稼働しているノードを別のクラスタに新しいノードとして参加させることは出来ないことが分かった。

したがって、リサイクル館の太陽光パネルの計測データが保存された Elasticsearch ノードをクラスタに参加させるには以下の2通りの方法が考えられる。

- リサイクル館の太陽光パネルの計測データが保存された Elasticsearch ノードのバックアップを取り、ノードに保存されたインデックスやドキュメントのデータを削除した上で、CO<sub>2</sub> データなどが保存されたクラスタに新しいノードとして参加させる
- CO<sub>2</sub> データなどが保存されたクラスタとは別で、サーバーゾーンに新たにクラスタを構築する。クラスタの構築にはリサイクル館の太陽光パネルの計測データが保存された Elasticsearch ノードが所属するクラスタを使用する。

## 4.6 節 結言

本章では、サーバーゾーンでのクラスタ構築における仮想環境を使用した事前検証について述べた。

次章では結論と今後の課題について述べる。



## 第5章

### 結論と今後の課題

#### 5.1 節 結論

本研究では、太陽光発電の計測データ補正と Elasticsearch のデータ移行およびクラスタ化について詳細に検討し、以下の主要な成果を達成した。

- 相互相関を用いた太陽光発電計測データの時間的ずれの特定手法の提案。
- Elasticsearch クラスタへのデータ移行に関する具体的な手順の確立と成功による、データ管理とアクセスの効率化。
- サーバーゾーンでの Elasticsearch クラスタ構築に向けた仮想環境を使用した事前検証を通じて、バージョンアップの重要性と手順の確立。

#### 5.2 節 今後の課題

本研究の成果を踏まえ、今後の研究の方向性として以下の課題が考えられる。

- 太陽光発電計測データの補正手法のさらなる改善。
- 学内ゾーンとサーバーゾーンでそれぞれ稼働しているクラスタごとに Kibana が存在しており、本研究室で管理する Elasticsearch に保存されたデータを一元的に管理、閲覧することが出来ないため、Kibana の統合による一元管理の実現。

- システムの継続的なモニタリングと定期的なメンテナンスの実施.

## 謝 辞

本研究を行うにあたり、終始、懇切丁寧な御指導と適切な御助言を賜りました本学工学部電気電子工学科通信システム工学研究室の都築伸二教授に深甚なる感謝の意を表します。

## 参考文献

- [1] 中川清隆, ”太陽方位、高度、大気外日射量の計算”,  
[http://es.ris.ac.jp/nakagawa/met\\_cal/solar.html](http://es.ris.ac.jp/nakagawa/met_cal/solar.html), 参照 May 23, 2022.
- [2] Elasticsearch B.V., ”Install Elasticsearch with Docker — Elasticsearch Guide [7.17] — Elastic”,  
<https://www.elastic.co/guide/en/elasticsearch/reference/7.17/docker.html>,  
参照 Nov 20,2023.
- [3] RAKUS Developers Blog, ”Docker とは一体何なんだ？【初心者向け】 - RAKUS Developers Blog — ラクス エンジニアブログ”, <https://tech-blog.rakus.co.jp/entry/20221007/docker>, 参照 Nov 20,2023.