

学位論文

学内のElasticsearchシステムのデータ移行と 冗長化

提出年月日 令和 4 年 X 月 X 日

改定日 令和 年 月 日

指導教員 都築 伸二 教授

入学年度 令和 6 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、学内の Elasticsearch のデータ移行と冗長化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 太陽光発電の計測データの補正

ここでは太陽光発電の計測データの補正について述べる。

第 3 章 学内ゾーンにおける Elasticsearch クラスタへのデータ移行

ここでは学内ゾーンにおける Elasticsearch クラスタへのデータ移行について述べる。

第 4 章 サーバーゾーンでのクラスタ構築

ここでは、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簡単にまとめている。

目次

内容梗概	I
第 1 章 緒論	1
第 2 章 太陽光発電の計測データの補正	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 日射量の式の導出	3
2.4 節 日射量を計算するプログラムの開発	4
2.5 節 実測値と計算値のプロット	5
2.6 節 相互相関を用いたずれ時間の特定	5
2.7 節 実測値データの補間	5
2.8 節 相互相関による最小ラグの選定	5
2.9 節 日射量の計算値の精度改善	7
2.10 節 pvlib の概要	7
2.11 節 pvlib を使って求めた計算値と実測値のプロット	7
2.12 節 pvlib を用いて計算した日射量を使用した, 相互相関に よる最小ラグの選定	7
2.13 節 結言	9
第 3 章 学内ゾーンにおける Elasticsearch クラスタへのデータ移行	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 学内ゾーンで稼働している Elasticsearch システムの状況	12

3.5 節	データ移行対象の Elasticsearch インデックスについて . . .	12
3.6 節	CO ₂ データの移行手順について	12
3.6.1	データのエクスポート	12
3.6.2	データの重複削除	13
3.6.3	データのインポート	15
3.7 節	一度目のデータ移行で移行できなかった CO ₂ データの 移行について	15
3.8 節	kibana によるデータの可視化	16
3.9 節	LEAF の運行日誌に関するデータの移行について	18
3.10 節	LEAF の運行日誌に関するデータの移行手順について . .	20
3.10.1	データのエクスポート	20
3.10.2	データのインポート	20
3.11 節	結言	20

第 4 章 サーバ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		21
4.1 節	緒言	21
4.2 節	サーバゾーンで稼働している Elasticsearch システム の状況	21
4.3 節	Docker とは	22
4.3.1	コンテナとは	22
4.3.2	Docker イメージとは	23
4.4 節	Docker Compose とは	23
4.5 節	検証環境のセットアップ	23
4.5.1	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) . . .	23
4.5.2	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)	28
4.6 節	異なる Elasticsearch クラスタへのノード参加検証 . . .	28
4.7 節	手順	30

4.7.1	単一ノードで稼働するクラスター A の構築	30
4.7.2	クラスター B の構築	30
4.7.3	クラスター B への参加試行	33
4.8 節	Elasticsearch のバージョンアップ	36
4.9 節	バージョンアップ手順	36
4.9.1	インストール方法の特定	36
4.9.2	apt によるバージョンアップ	37
4.10 節	kibana のバージョンアップ	39
4.11 節	バージョンアップ後の動作確認	39
4.12 節	サーバーゾーンにおけるクラスターの構築	40
4.13 節	結言	40
第 5 章	結論と今後の課題	41
	謝 辞	42
	参考文献	43
付録 A	付録	44
A.1	水源監視システム (送信用 Python スクリプト)	44

第1章

緒論

急速に進化するデータ管理において、Elasticsearch は極めて重要な技術として登場し、データの保存、検索、管理方法に革命をもたらした。当初、大量のデータを効率的に処理するために設計された Elasticsearch は、シンプルなシングルノードシステムから複雑なクラスタ構成へと移行し、データハンドリング技術において大きな進歩を遂げた。この変遷は、様々な分野で堅牢でスケラブル、かつ冗長性のあるデータ管理システムに対する要求が高まっていることを裏付けている。

データシステムの冗長性、特に Elasticsearch クラスタにおける冗長性は、データの信頼性と可用性を確保する上で重要な要素となっている。冗長性とは、システムの重要なコンポーネントや機能を二重化することで、信頼性を高め、一点障害のリスクを低減することを指す。

Elasticsearch のノード管理における現在のトレンドは、クラスタ化されたシステムを好む傾向が強まっている。しかし、Elasticsearch の技術的な側面については多くの研究があるが、同じ環境内でシングルノードシステムからクラスタ化されたシステムへデータを移行し、さらに別のネットワークゾーンに新しくクラスタ化されたシステムを構築する際の実際的な課題や戦略について掘り下げた研究はほとんどない。

本研究の目的は、シングルノードの Elasticsearch システムから学内ゾーン内のクラスタ化システムへデータを移行するプロセスを分析することである。同時に本研究では、この2つのゾーン間のデータ移行を行わずに、サーバゾー

ンに新しい冗長化されたクラスタ化システムを構築する.

第 2 章では学内ゾーンにおける Elasticsearch ノードから新クラスタへのデータ移行について述べる. 第 3 章では仮想環境を使用してサーバーゾーンの Elasticsearch ノードをシミュレートし、マルチコンテナ Docker アプリケーションのツールである docker-compose を用いてクラスタ構築の実現可能性を検証したことについて述べる. 第 4 章ではサーバーゾーンで既存の Elasticsearch ノードを用いたクラスタ構築について述べる. 第 5 章では結論と課題を述べる.

第2章

太陽光発電の計測データの補正

2.1 節 緒言

本章では太陽光発電の計測データの補正について述べる。

2.2 節 太陽光発電の計測データの問題点について

CSV データなどで保存された太陽光発電の環境データはオフライン環境でファイル書き込みを行っているため、PC の内部時計がずれており日時情報が誤っている場合がある。そこで、任意の緯度経度と日射量から日時を求めることの第一段階として、任意の緯度経度と日時から日射量を計算するプログラムを開発した。

2.3 節 日射量の式の導出

任意の緯度経度、日時における日射量 Q は、任意の緯度 ϕ 、経度 λ の地点における任意の日時、太陽高度 α から求めることができる。

まず、次式により元旦からの通し日数 dn に基いて定めた θ を用いて、当該日の太陽赤緯 δ 、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。

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

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

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

$$E_q = 0.000075 + 0.001868 \cos \theta - 0.032077 \sin \theta - 0.014615 \cos 2\theta - 0.040849 \sin 2\theta \quad (2.4)$$

日本標準時間から、太陽の時角 h を求める。

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

δ, ϕ, h の値が既知となったので α は

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

として求めることができる。

最後に、 Q を

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

により求めることができる。また、 $1367\text{W}/\text{m}^2$ は太陽定数である。

2.4 節 日射量を計算するプログラムの開発

式 (1) から式 (7) をもとに、日射量を求めるプログラムを開発した。

日射量を求めるプログラムでは、2022 年 5 月 17 日 17 時 53 分 0 秒における、経度 132.75093、緯度 33.82794 の地点での日射量を計算している。日射量を求めるプログラムより、図 2.1 が算出される。図 2.1 から、今回与えた日時、緯度経度における日射量は、約 $299.15\text{W}/\text{m}^2$ となった。

```
sofue@sofue-MS-7B61:~/py/elasticsearch-tutorial$ python3 com_global.py  
日射量: 299.149672991502
```

図 2.1 日射量を求めるプログラムの結果

2.5 節 実測値と計算値のプロット

取得した Elasticsearch サーバーの日射量データと、計算式を用いて求めた日射量データをプロットしたものを図 2.2 に示す。図 2.2 は Elasticsearch サーバーから取得した 2022 年 6 月 2 日の日射量データと、リサイクル館の緯度経度と日付情報 (2022 年 6 月 2 日) を入力として求めた日射量の計算値をプロットしている。

2.6 節 相互相関を用いたずれ時間の特定

相互相関を用いて、実測データと計算データとの時間的遅延を検出することで、実測データの計測日時のずれ時間を特定する。

2.7 節 実測値データの補間

相互相関の計算を行う前に、実測データの計測日時の間隔を均一にするため、線形補間を行い、日時間隔を 1 s に統一する。

2.8 節 相互相関による最小ラグの選定

相互相関の計算に使用する実測データの範囲を、2022 年 6 月 2 日 0 時 0 分から 2022 年 6 月 2 日 23 時 59 分まで期間とする。

次に、実測データのタイムスタンプ列を入力として求めた計算データを 1 分ずつスライドさせ、その都度実測値データとの相互相関を計算する。

相互相関の計算結果より、計算値の日時を実測値より 124 秒進めた際に、相互相関の値が最大となることが分かった。

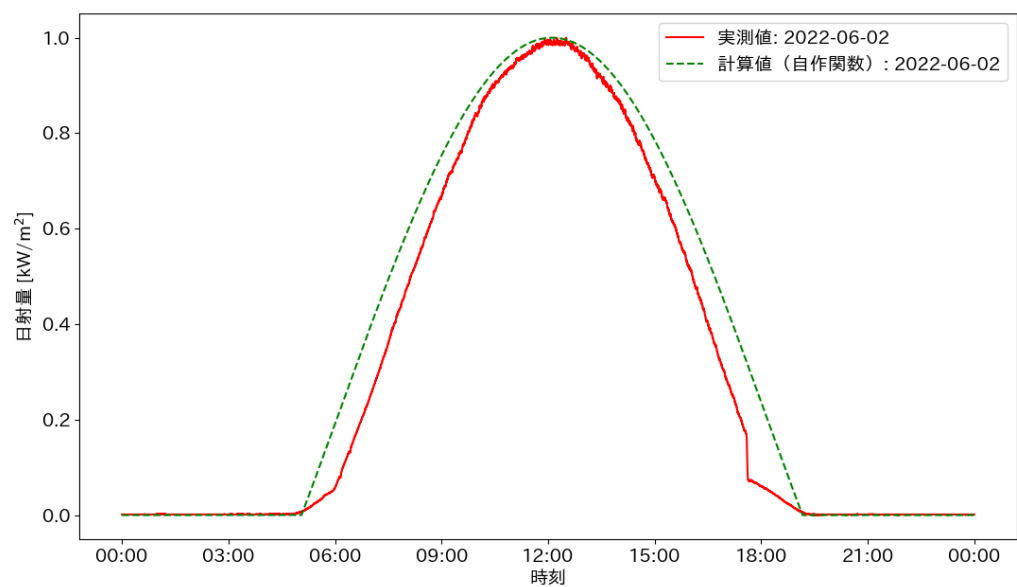


図 2.2 2022 年 6 月 2 日の日射量の実測値と計算値をプロットしたもの

しかし、実測データには計測日時のズレは殆どないため、0秒進めた際に相互相関の値が最大となるのが正しい。

これは、計算データの精度が実測値と乖離していることが原因であると考えられる。

2.9 節 日射量の計算値の精度改善

日射量の計算値の予測精度を改善するため、自作の日射量計算プログラムではなく、`pvlib` というライブラリを使用して、任意の緯度経度と日時における日射量を求め、相互相関を計算する。

2.10 節 `pvlib` の概要

`pvlib python` 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、コミュニティが開発したツールボックスである。

2.11 節 `pvlib` を使って求めた計算値と実測値のプロット

取得した Elasticsearch サーバーの日射量データと、`pbliv` を用いて求めた日射量データをプロットしたものを図 2.3 に示す。図 2.3 は Elasticsearch サーバーから取得した 2022 年 6 月 2 日の日射量データと、リサイクル館の緯度経度と日付情報 (2022 年 6 月 2 日) を入力として求めた日射量の計算値をプロットしている。

2.12 節 `pvlib` を用いて計算した日射量を使用した、相互相関による最小ラグの選定

相互相関の計算に使用する実測データの範囲を、2022 年 6 月 2 日 0 時 0 分から 2022 年 6 月 2 日 23 時 59 分まで期間とする。

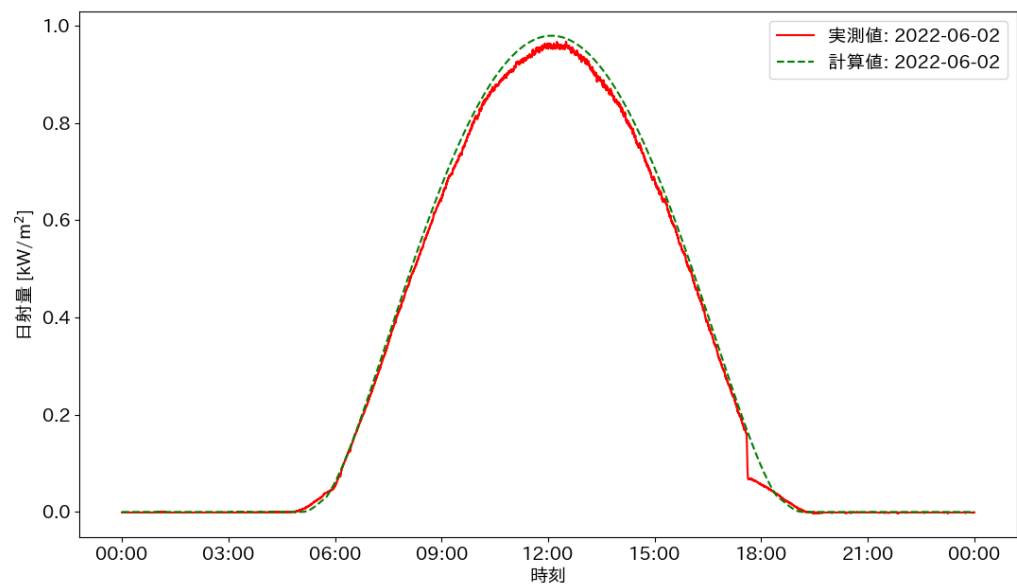


図 2.3 2022 年 6 月 2 日の日射量の実測値と計算値をプロットしたもの

次に、実測データのタイムスタンプ列を入力として求めた計算データを 1 分ずつスライドさせ、その都度実測値データとの相互相関を計算する。

相互相関を求めた結果、計算値の日時を実測値より 74 秒進めた際に、相互相関の値が最大となることが分かった。

計算式を用いて求めた日射量データ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 秒改善している。

2.13 節 結言

本章では太陽光発電の計測データの補正について述べた。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

学内ゾーンにおける Elasticsearch クラスタへのデータ移行

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 学内ゾーンで稼働している Elasticsearch システムの状況

学内ゾーンでは, 133.71.106.168 で単一ノードの Elasticsearch と, 133.71.106.170, 133.71.106.141, 133.71.106.136 の Elasticsearch ノードによって構成された Elasticsearch クラスタが稼働している.

133.71.106.168 の Elasticsearch には, CO₂ 濃度監視システムによって計測されたデータと LEAF の運行日誌に関するデータが保存されている.

3.5 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 CO₂ 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.

3.6 節 CO₂ データの移行手順について

CO₂ 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.

3.6.1 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.

3.6.2 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す。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ます：

軽量: SQLite は非常に小さく、リソースの少ない環境でも動作します。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です。自己完結型: SQLite はサーバーレスで、設定や管理が不要です。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を意味します。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ちます。言語のサポート: 多くのプログラミング言語で使用できます。幅広い用途: デスクトップ、モバイル、ウェブアプリケーションなど、さまざまな環境で利用されています。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、配布できます。SQL 標準準拠: 多くの SQL 標準機能をサポートしていますが、全てではありません。

SQLite は、その単純さと効率性から、広く使われているデータベースエンジンの一つです。特に、大規模なデータベースシステムのオーバーヘッドが不要な場合や、独立したアプリケーションでの使用に適しています。

SQLite データベースはリレーショナルデータベースの一種であり、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的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 データベースでは、部屋番号 (number) とタイムスタンプ (utctime) を一意的キーとして設定した。以下のリスト 4.1, 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、フィールドのメンバーが統一されておらず、一部センサー情報が存在しない場合がある。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": "2022-11-06T22:45:30.080925",
    "ip": "172.23.68.19/16",
    "utctime": "2022-11-06T13:45:30.080895",
  }
}
```

```
    "TEMP": 24.47,  
    "index_name": "co2_e411",  
    "ms": "",  
    "number": "E411"  
  }  
}
```

3.6.3 データのインポート

重複データ削除後のデータが保存された SQLite テーブル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。

3.7 節 一度目のデータ移行で移行できなかった CO₂ データの移行について

実装したデータ移行プログラムを使用して 133.71.201.197 から 133.71.106.141 の Elasticsearch サーバーへ CO₂ データを移行したのが 2023 年 5 月中旬頃であり、CO₂ 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 133.71.106.141 の Elasticsearch サーバーに対してラズベリーパイから CO₂ 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 CO₂ 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、移行先 ElasticSearch サーバーに対して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 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 CO₂ 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、移行元 ElasticSearch サーバー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 elasticdump [1] 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にエクスポートしたデータをフィルタリングする。
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。

3.8 節 kibana によるデータの可視化

計 2 回の CO₂ 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。

2 回目の CO₂ データの移行によって、2023 年 5 月中旬から 2023 年 7 月中旬までの期間とその前後の期間において、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。

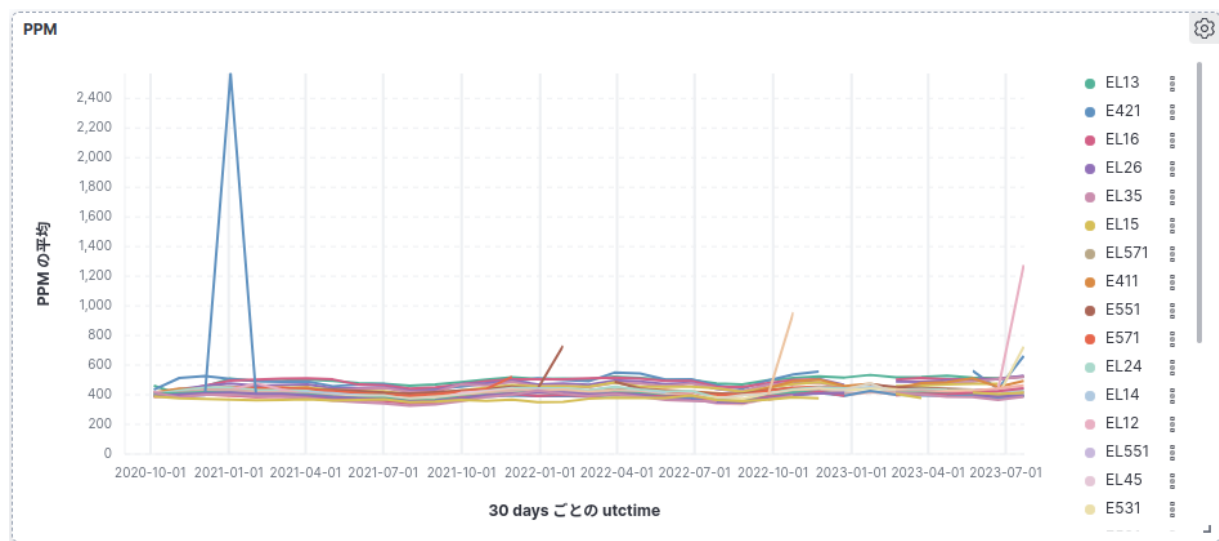


図 3.1 co2_modbus の PPM

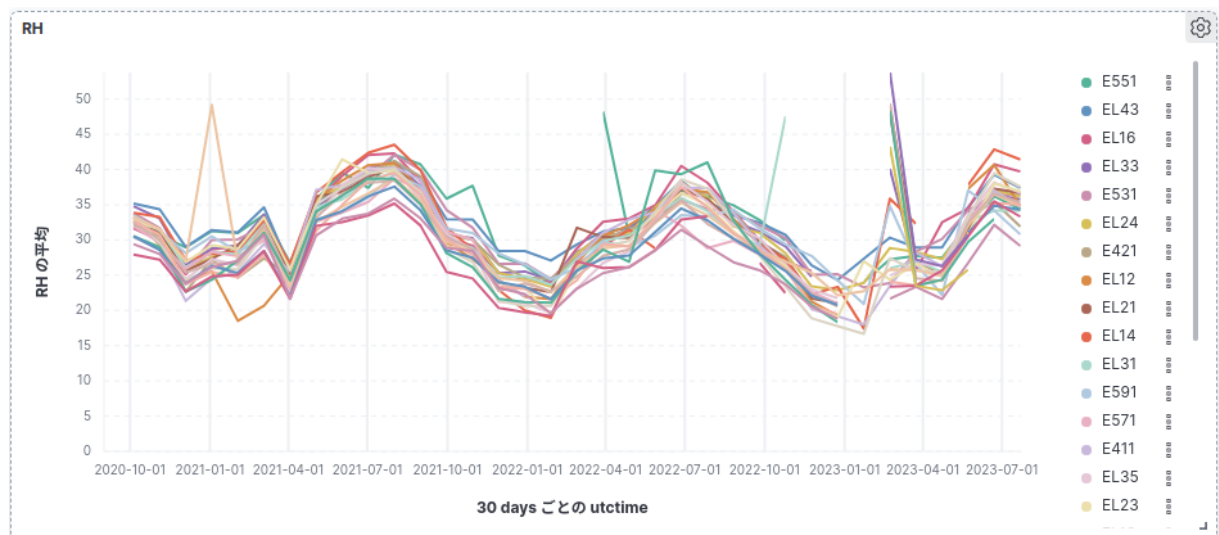


図 3.2 co2_modbus の RH

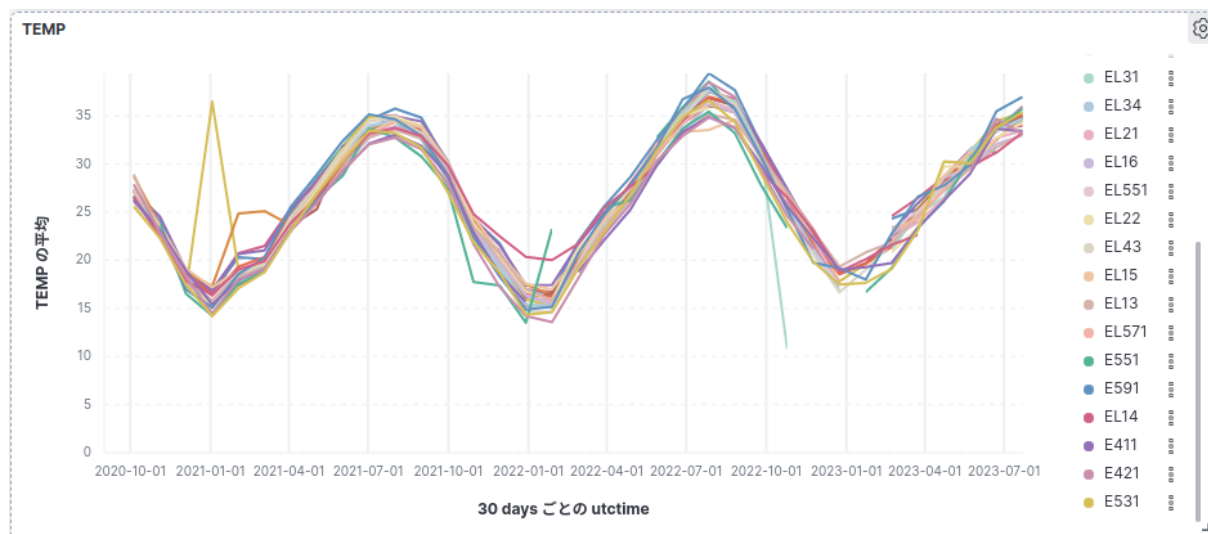


図 3.3 co2_modbus の TEMP

3.9 節 LEAF の運行日誌に関するデータの移行について

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

これらの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った。

次に、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.

movement_diary と movement_diary01 のドキュメントの違いより, movement_diary01 は movement_diary のもつ情報量を全て保持しており, その上で追加のフィールドを持っていることから, 移行するのは movement_diary01 インデックスのみで十分であることが分かった.

3.10 節 LEAF の運行日誌に関するデータの移行手順について

3.10.1 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, `elasticsearch-dump` 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.

3.10.2 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.

3.11 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.

第4章

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

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 サーバーゾーンで稼働している Elasticsearch システムの状況

サーバーゾーンでは、133.71.201.197 で単一ノードの Elasticsearch が稼働しており、リサイクル館の太陽光パネルの計測データが保存されている。

133.71.201.197 の Elasticsearch のバージョン 7.17.6 であり、学内ゾーンで稼働している Elasticsearch クラスタに参加している Elasticsearch ノードのバージョンは 7.17.9 である。

本研究室では現在、バージョン 7.17.9 の Elasticsearch を採用しているため、サーバーゾーンで構築しようとしているクラスタの Elasticsearch のバージョンも、学内ゾーンで稼働している Elasticsearch クラスタと同様、バージョン 7.17.9 を採用する。

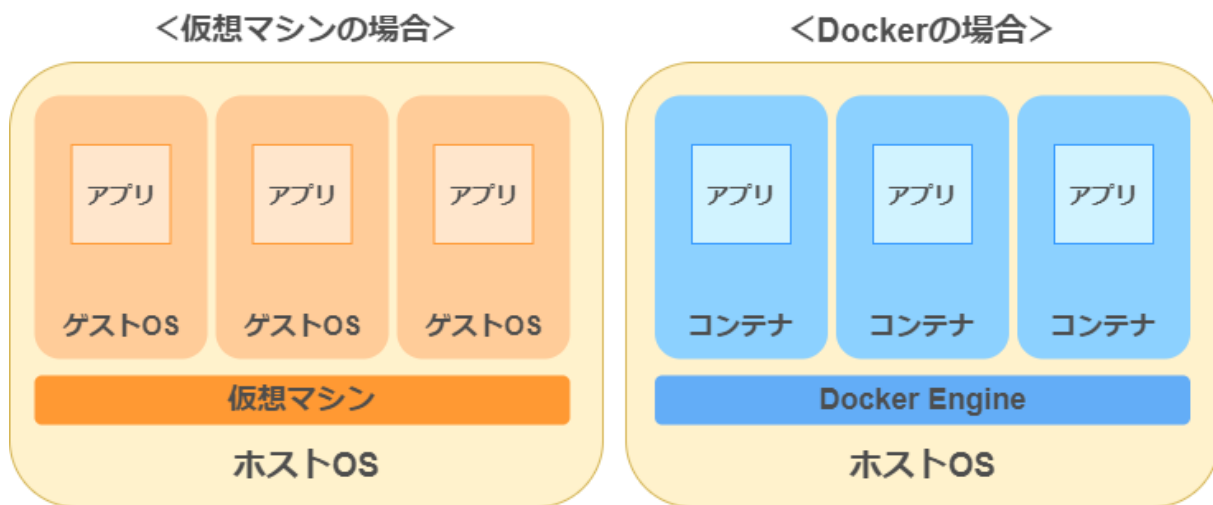


図 4.1 仮想マシンと Docker の違い [2]

そこで, Docker, Docker Compose を使用して, 異なるバージョンである 7.17.6 と 7.17.9 の Elasticsearch ノードをクラスタリングすることが可能かどうかを確認するために実施した.

4.3 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.

4.3.1 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, コンテナの作成には Docker イメージが必要となる.

4.3.2 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.

4.4 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.

4.5 節 検証環境のセットアップ

4.5.1 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)

Listing 4.1 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 を示す.

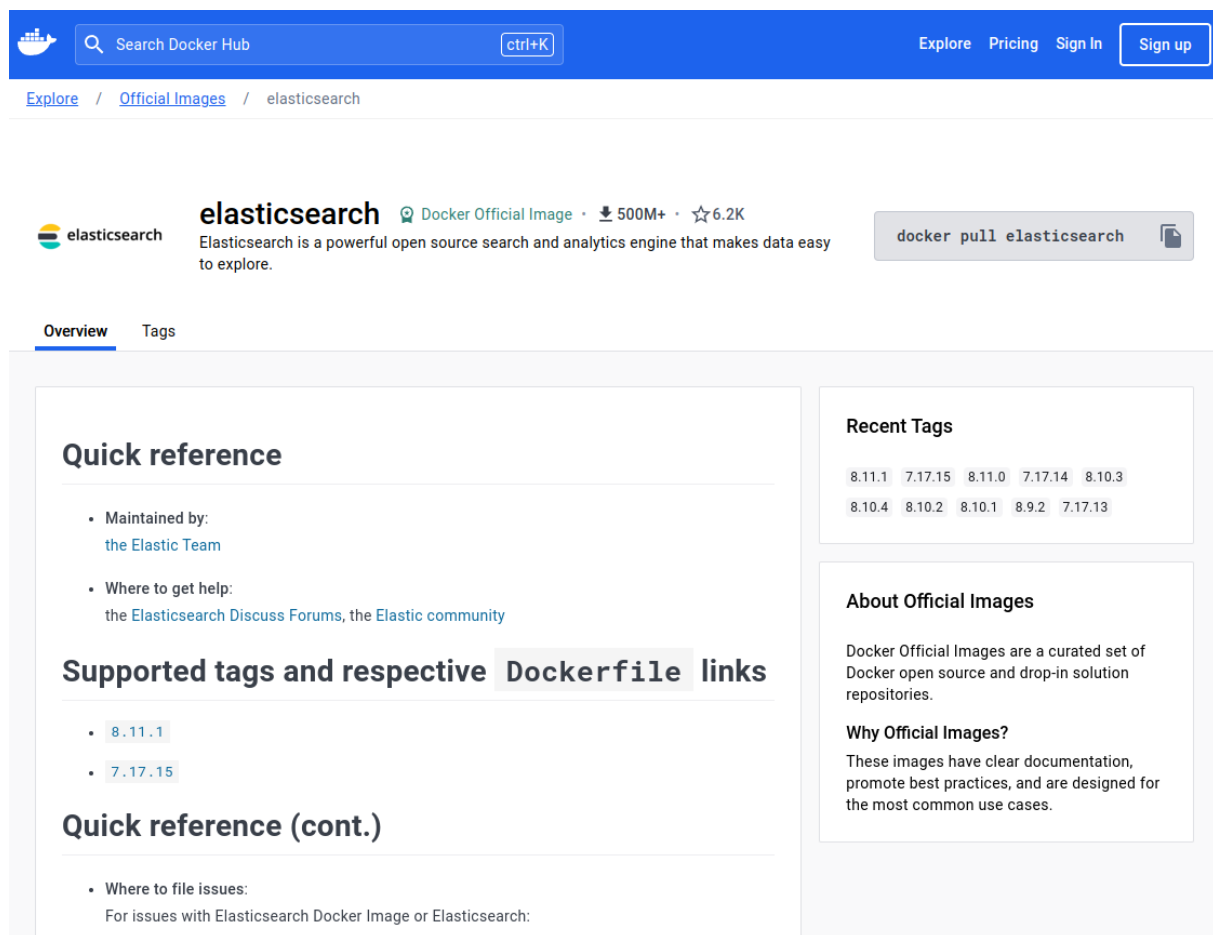


図 4.2 Elasticsearch の Docker イメージ

Listing 4.1 全て同じバージョンのElasticsearchを使用したクラスタを構成する docker-compose.yml

```
version: '2.2'

services:
  es01:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.17.9
    container_name: es01
    environment:
      - node.name=es01
      - cluster.name=es-docker-cluster
      - discovery.seed_hosts=es02,es03
      - cluster.initial_master_nodes=es01,es02,es03
    ports:
      - 9200:9200
    networks:
      - elastic
  es02:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.17.9
    container_name: es02
    environment:
      - node.name=es02
      - cluster.name=es-docker-cluster
      - discovery.seed_hosts=es01,es03
      - cluster.initial_master_nodes=es01,es02,es03
    networks:
      - elastic
  es03:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.17.9
    container_name: es03
    environment:
      - node.name=es03
```

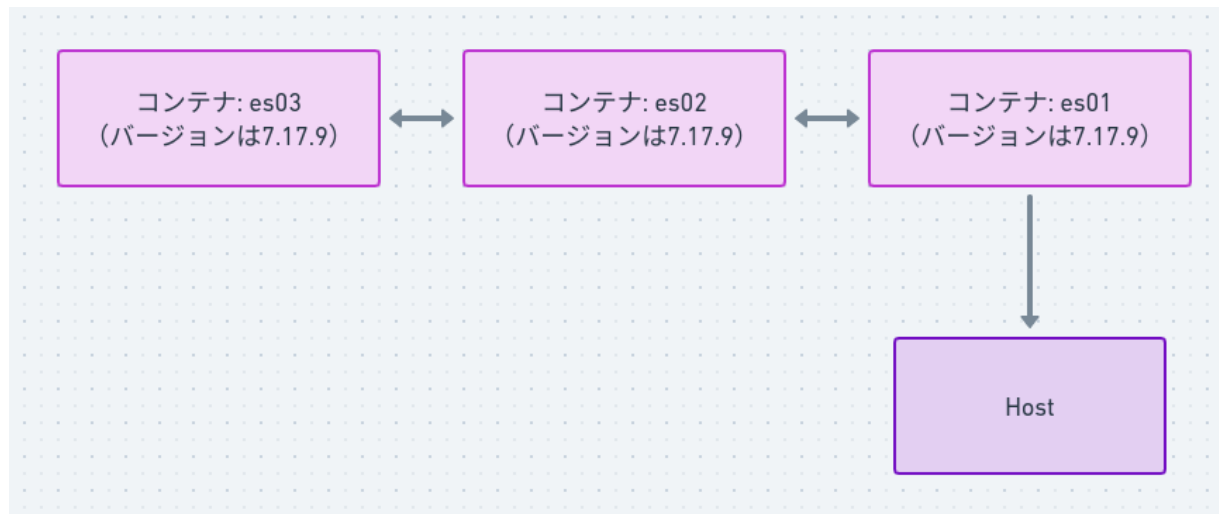


図 4.3 docker-compose.yml を図で表現したもの

```
- cluster.name=es-docker-cluster
- discovery.seed_hosts=es01,es02
- cluster.initial_master_nodes=es01,es02,es03
networks:
- elastic
```

```
networks:
  elastic:
    driver: bridge
```

また、図 4.3 に Listing 4.1 の docker-compose.yml を図で表現したものを示す。
Listing 4.1 の docker-compose.yml ファイルで記述している内容について説明する。

サービスの定義

- es01, es02, es03: これらは Elasticsearch のノード（サーバー）である。各ノードは異なるコンテナとして定義されている。es01, es02, es03 はそれぞれ異なるコンテナ名で、Elasticsearch の異なるインスタンスを

実行する.

各ノードの設定

- **image:** 使用する Docker イメージ. ここでは Elasticsearch の 7.17.9 バージョンを使用している.
- **container_name:** コンテナに割り当てられる名前.
- **environment:** 環境変数の設定. Elasticsearch のクラスタ設定を含む.
- **ports:** ホストマシンとコンテナ間のポートマッピング. 例えば, '9200:9200' はホストマシンの 9200 ポートをコンテナの 9200 ポートにマッピングする.
- **networks:** コンテナ間通信のためのネットワーク設定. ここでは elastic ネットワークが使用されている.

ボリュームとネットワークの設定

- **networks:** デフォルトのドライバである bridge ドライバを使用する elastic ネットワークを定義している. これにより, 異なるコンテナが相互に通信できるようになる.

この設定により, Elasticsearch の 3 ノードを含むクラスタが Docker 上で動作するようセットアップされる.

以降, 説明を簡単にするため, docker-compose.yml を図で表現したもののみを掲示する.

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 4.4 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.

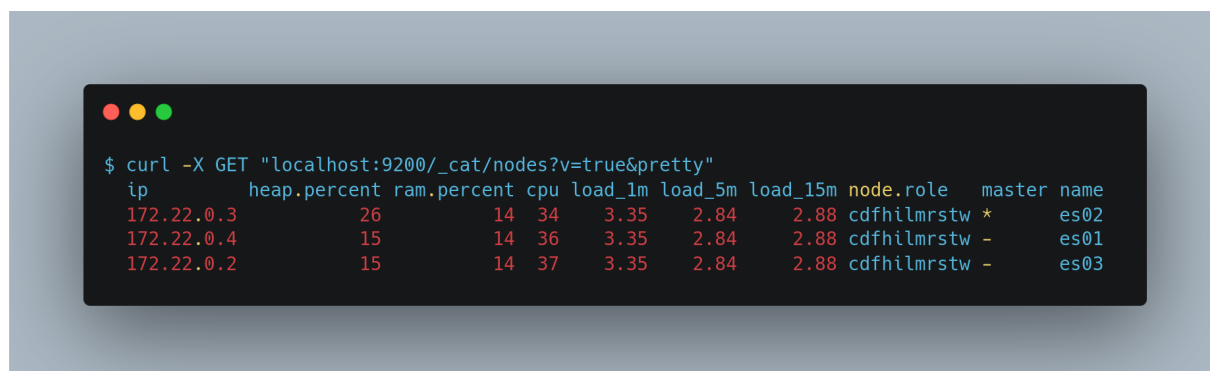


図 4.4 クラスタに参加しているノードを一覧表示した結果

4.5.2 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) のみが正常にクラスタに参加できていることが確認できる.

また, Elasticsearch 起動時に出力されたログを確認したところ, 図 4.7 に示すように, クラスタに参加できなかった es03 のコンテナで Elasticsearch がエラーログを出力して終了していることが分かった.

4.6 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.

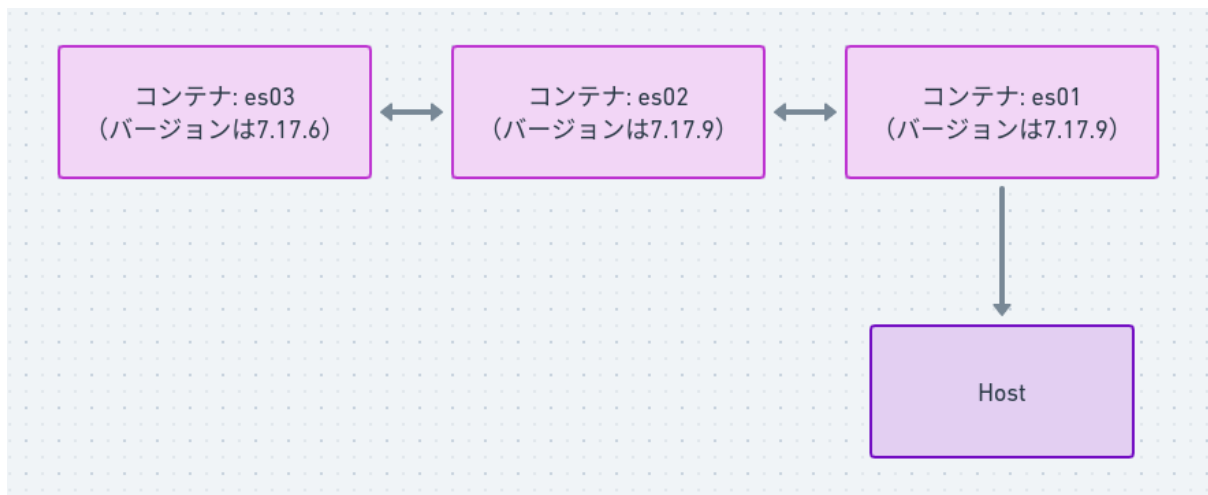


図 4.5 変更後の docker-compose.yml を図で表現したもの

```
$ 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 cdfhilmrstw -   es02
172.22.0.3      18           14      8    1.86    1.32    1.14 cdfhilmrstw *   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

```
es03 |   uncaught exception in thread [main]
es03 | java.lang.IllegalStateException: cannot downgrade a node from version [7.17.9] to version [7.17.6]
es03 |     at org.elasticsearch.env.NodeMetadata.upgradeToCurrentVersion(NodeMetadata.java:95)
es03 |     at org.elasticsearch.env.NodeEnvironment.loadNodeMetadadata(NodeEnvironment.java:484)
es03 |     at org.elasticsearch.env.NodeEnvironment.<init>(NodeEnvironment.java:356)
es03 |     at org.elasticsearch.node.Node.<init>(Node.java:429)
es03 |     at org.elasticsearch.node.Node.<init>(Node.java:309)
es03 |     at org.elasticsearch.bootstrap.Bootstrap$5.<init>(Bootstrap.java:234)
es03 |     at org.elasticsearch.bootstrap.Bootstrap.setup(Bootstrap.java:234)
es03 |     at org.elasticsearch.bootstrap.Bootstrap.init(Bootstrap.java:434)
es03 |     at org.elasticsearch.bootstrap.Elasticsearch.init(Elasticsearch.java:169)
es03 |     at org.elasticsearch.bootstrap.Elasticsearch.execute(Elasticsearch.java:160)
es03 |     at org.elasticsearch.cli.EnvironmentAwareCommand.execute(EnvironmentAwareCommand.java:77)
es03 |     at org.elasticsearch.cli.Command.mainWithoutErrorHandling(Command.java:112)
es03 |     at org.elasticsearch.cli.Command.main(Command.java:77)
es03 |     at org.elasticsearch.bootstrap.Elasticsearch.main(Elasticsearch.java:125)
es03 |     at org.elasticsearch.bootstrap.Elasticsearch.main(Elasticsearch.java:80)
es03 |   For complete error details, refer to the log at /usr/share/elasticsearch/logs/es-docker-cluster.log
es03 | exited with code 1
```

図 4.7 es03 のログ

4.7 節 手順

4.7.1 単一ノードで稼働するクラスタ A の構築

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。

4.7.2 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出す。

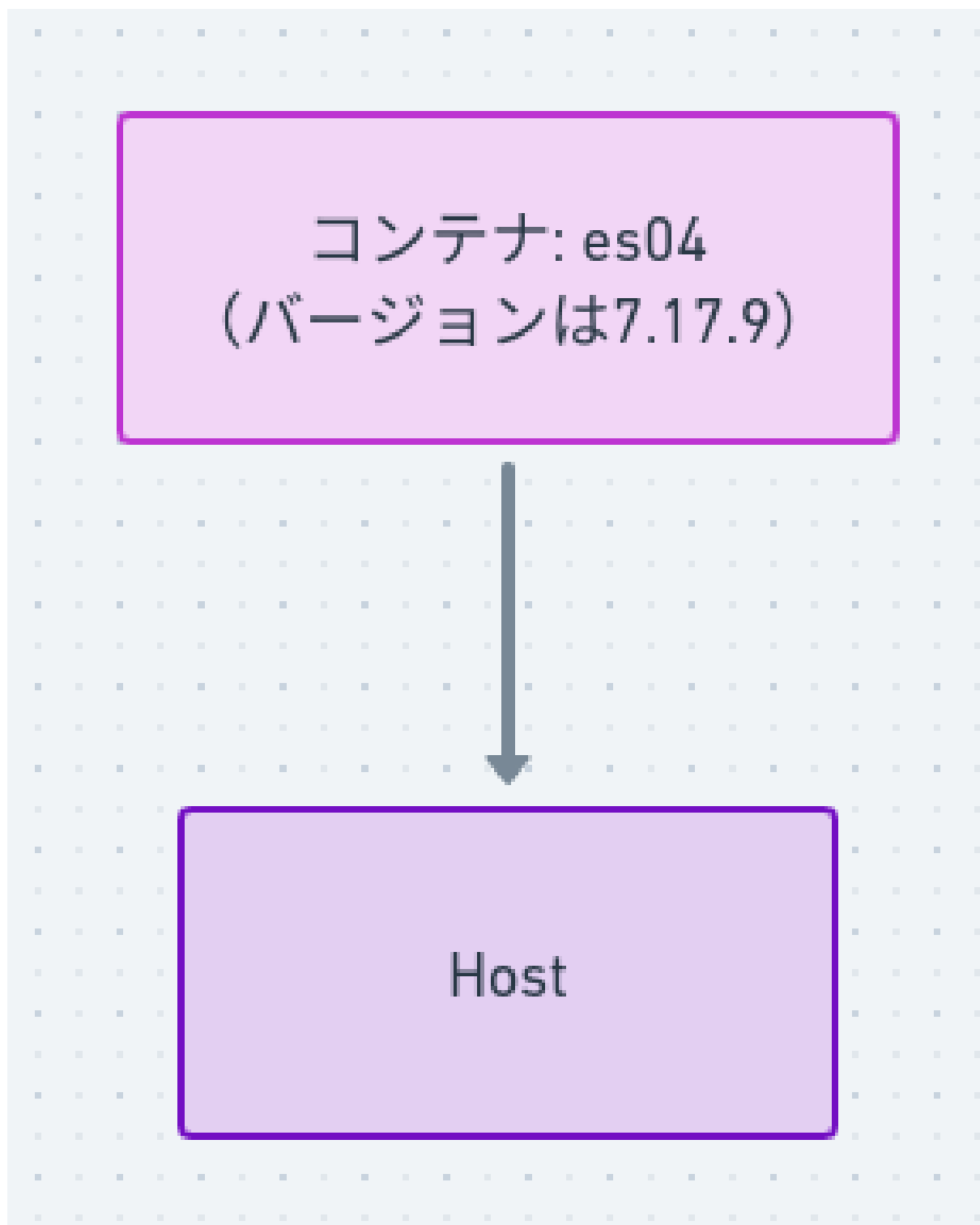


図 4.8 クラスタ A の構築の際に使用した docker-compose.yml を図で表現したもの

```
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

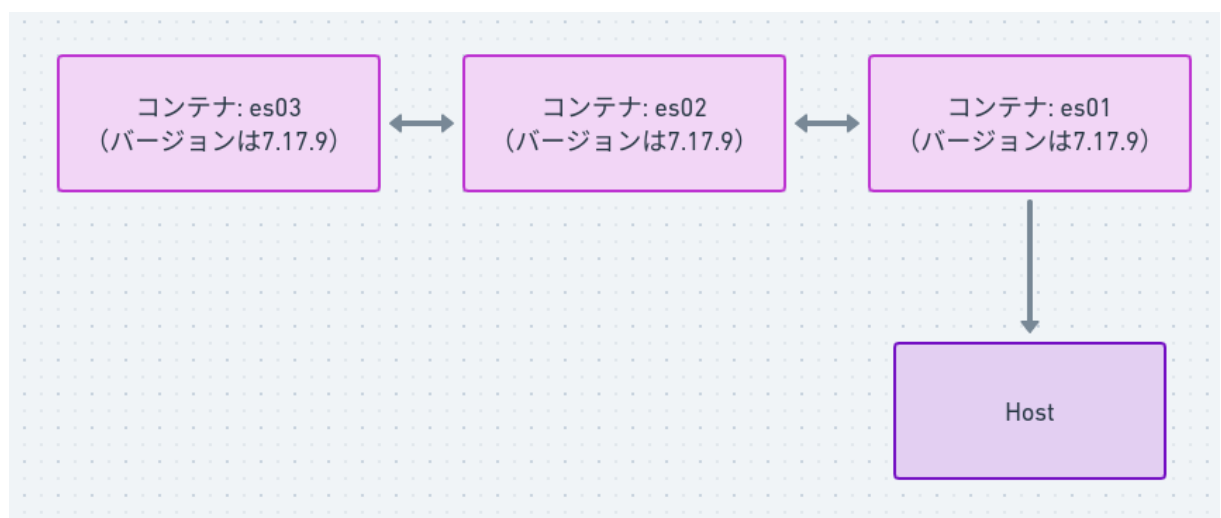


図 4.10 クラスタ B の構築の際に使用した docker-compose.yml を図で表現したもの

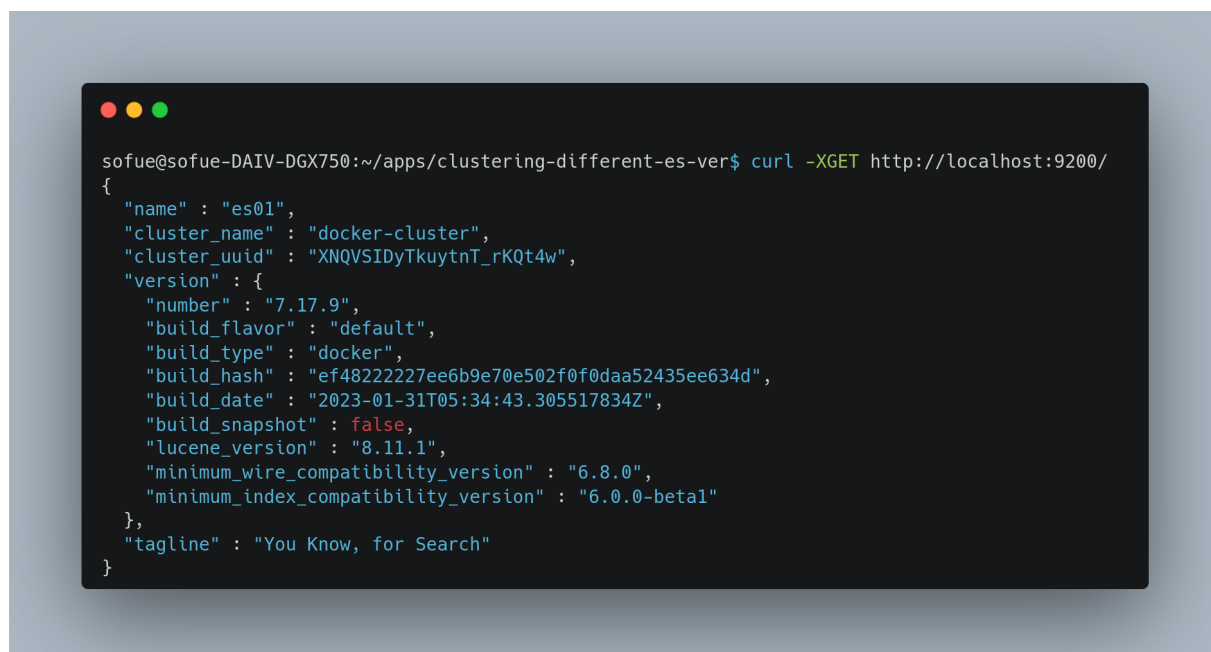


図 4.11 クラスタの情報について問い合わせた結果

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。

4.7.3 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。

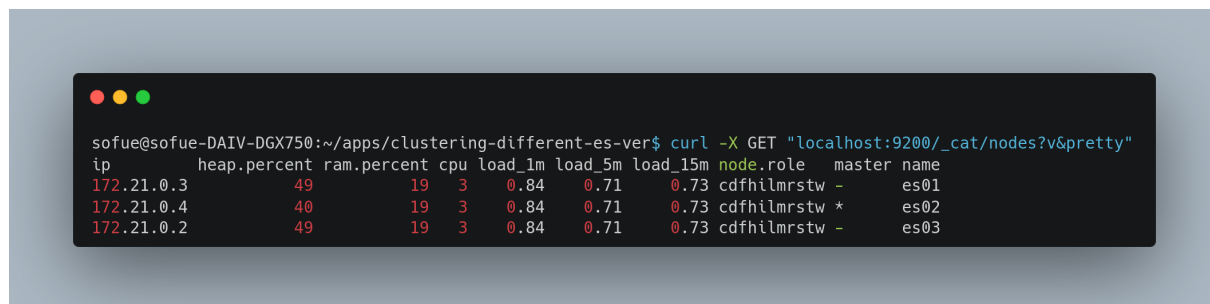


図 4.12 クラスタ B の起動後, クラスタに参加しているノードの一覧を取得した結果

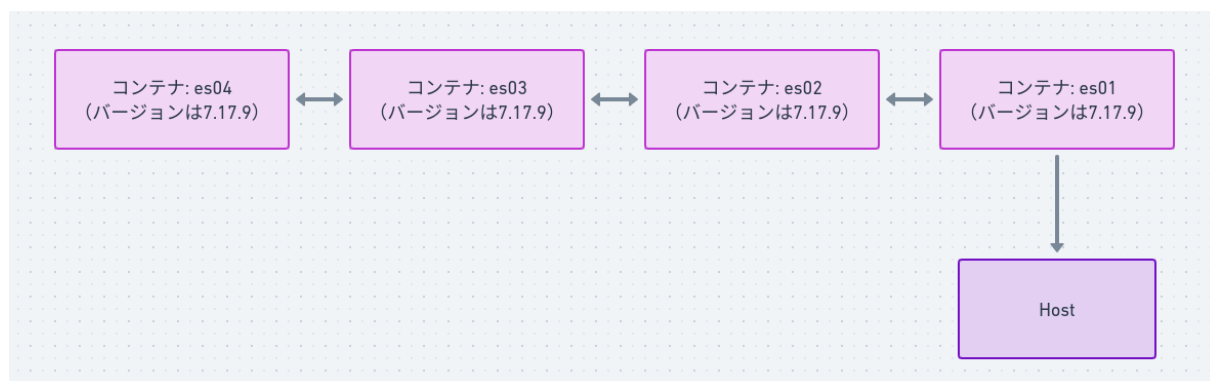


図 4.13 合計 4 ノードでクラスタ B の起動を試みた際に使用した docker-compose.yml を図で表現したもの

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

```
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 cdfhilmrstw -   es02
172.22.0.2      16           21      5    0.75   1.24   1.02 cdfhilmrstw -   es01
172.22.0.5      33           21      5    0.75   1.24   1.02 cdfhilmrstw *   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 [XNQVSiDyTKuytnT_rKQt4w]. 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

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、CO₂ 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
- CO₂ 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。

4.8 節 Elasticsearch のバージョンアップ

3章の検証結果より、異なるバージョンの Elasticsearch ノードでクラスタを構築することは出来ないため、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(133.71.201.197) をバージョンアップする必要がある。そこで、133.71.201.197 にインストールされた Elasticsearch のバージョンアップを行う。

4.9 節 バージョンアップ手順

4.9.1 インストール方法の特定

バージョンアップを行うためには、133.71.201.197 の UbuntuPC にどのように Elasticsearch をインストールしたか特定する必要がある。



```
matsubara@iris:~$ apt list --installed | grep elasticsearch  
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.  
elasticsearch/stable,now 7.17.6 amd64 [installed,upgradable to: 7.17.15]
```

図 4.16 apt によって elasticsearch がインストールされたか調べた結果

図 4.16 に、apt によってインストールされたパッケージの中に elasticsearch という文字列を含むパッケージが存在するか調べた結果を示す。図 4.16 より、apt によってインストールされたことが分かった。

次に、apt でインストール可能な elasticsearch のバージョンを一覧表示した結果を図 4.17 に示す。図 4.17 にターゲットである 7.17.9 が含まれているため、apt を使用してバージョンアップできることが確認できた。

4.9.2 apt によるバージョンアップ

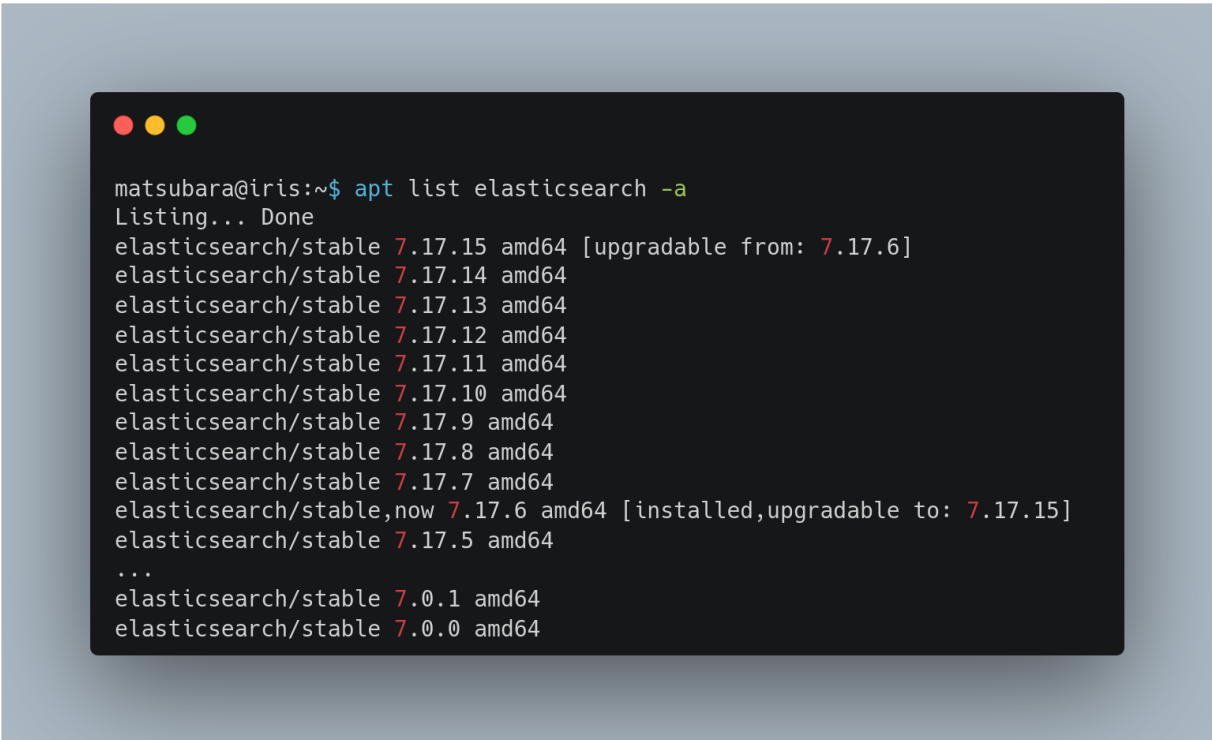
まず、`sudo systemctl stop elasticsearch.service` コマンドを実行して elasticsearch ノードをシャットダウンする。

次に、`sudo apt install elasticsearch=7.17.9` コマンドを実行して elasticsearch パッケージをバージョンアップする。

elasticsearch をバージョンアップ後、`sudo systemctl start elasticsearch` コマンドを実行して Elasticsearch ノードを起動する。

ノードの起動後、Elasticsearch のバージョンを確認した結果を図 4.18 に示す。

図 4.18 より、Elasticsearch のバージョンが 7.17.9 にバージョンアップ出来たことが確認できた。



```
matsubara@iris:~$ apt list elasticsearch -a
Listing... Done
elasticsearch/stable 7.17.15 amd64 [upgradable from: 7.17.6]
elasticsearch/stable 7.17.14 amd64
elasticsearch/stable 7.17.13 amd64
elasticsearch/stable 7.17.12 amd64
elasticsearch/stable 7.17.11 amd64
elasticsearch/stable 7.17.10 amd64
elasticsearch/stable 7.17.9 amd64
elasticsearch/stable 7.17.8 amd64
elasticsearch/stable 7.17.7 amd64
elasticsearch/stable,now 7.17.6 amd64 [installed,upgradable to: 7.17.15]
elasticsearch/stable 7.17.5 amd64
...
elasticsearch/stable 7.0.1 amd64
elasticsearch/stable 7.0.0 amd64
```

図 4.17 apt でインストール可能な elasticsearch のバージョンを一覧表示した結果



```
matsubara@iris:~$ curl -u takenaka:takenaka -s -XGET http://localhost:9200/ | grep number
"number" : "7.17.9",
```

図 4.18 ノードの起動後, Elasticsearch のバージョンを確認した結果



図 4.19 Elasticsearch のバージョンアップ後, 太陽光パネルの計測データが保存されているか kibana 上で確認した結果

4.10 節 kibana のバージョンアップ

kibana も elasticsearch と同様, apt を使用してインストールされていたため, `sudo systemctl stop kibana.service` コマンド, `sudo apt install kibana=7.17.9` コマンド, `sudo systemctl start kibana` コマンドをそれぞれ実行して, kibana のバージョンアップも行った.

4.11 節 バージョンアップ後の動作確認

Elasticsearch のバージョンアップ後, 太陽光パネルの計測データが Elasticsearch に保存されているか kibana 上で確認した結果を図 ??に示す.

図 4.19 より, バージョンアップ後の Elasticsearch ノードを起動した 14:22:00 以降にドキュメントがインサートされていることが確認できた.

4.12 節 サーバーゾーンにおけるクラスタの構築

4.13 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。

次章ではサーバーゾーンでのクラスタ構築について述べる。

第5章

結論と今後の課題

本研究では、単一ノードの Elasticsearch システムから学内ゾーン内のクラスタ化システムへデータを移行するプロセスを分析し、さらにサーバゾーンに新しい冗長化されたクラスタ化システムを構築する検証を行った。

データ移行プロセスの検証では、CO2 データと LEAF の運行日誌に関するデータの移行を行った。CO2 データの移行では、重複データの削除が必要であったが、SQLite データベースを用いた手法で対応した。LEAF の運行日誌に関するデータの移行では、同じ名前のインデックスを移行先の Elasticsearch サーバーに作成して、データをインサートすることで行った。

サーバゾーンでのクラスタ構築の検証では、Docker, Docker Compose を使用した事前検証を行った。異なるバージョンの Elasticsearch (7.17.6 と 7.17.9) を使用したクラスタの構築を試したが、正常に構築できないことを確認した。

学内ゾーンとサーバ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が出来る状態を目指す。

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] 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.
- [2] 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.

付録 A

付録

A.1 水源監視システム (送信用 Python スクリプト)

LoRa_obs_transmit.py のソースコードを A.1 に示す.

Listing A.1 LoRa_obs_transmit.py

```

1  ## *****coding:utf-8*****
2
3  import time
4  import os
5  from datetime import datetime
6  import serial
7
8  """ sleep()をいれて, 少し待たないとエラー落ちする """
9  time.sleep(60)
10
11 class Main() :
12
13     def __init__(self):
14         """ 初期値および対象ディレクトリの設定 """
15         self.s_num = 0
16
17         self.copy_dir = "C:/Users/taikimizukan/Dropbox/sumitomo
18         /obs_csv/"
19         self.target_dir = "C:/Users/taikimizukan/Desktop/
20         obs_data/"

```

```
19         self.temporary_log = "./temporary_log.txt"
20
21         """ 最終データを取得 """
22         with open(self.temporary_log,"r") as f :
23             self.old_line = f.readline()
24             print("前回のデータ:"+str(self.old_line))
25
26         """ 起動時 [$RFINF,ONコマンド送信***] """
27         INF = "$RFINF,ON***"
28         with serial.Serial("COM7",115200,timeout=2) as ser :
29             time.sleep(2)
30             while True :
31                 for i in INF :
32                     ser.write(i.encode("utf-8"))
33
34                     result = str(ser.readline())
35                     if result.find("RESULT,RFINF,ON,OK") > 0 :
36                         break
37                     else :
38                         time.sleep(2)
39
40         """ ループ関数実行 """
41         self.Loop()
42
43
44         """ 作成日時が最新ファイルのフルパスを取得し返す関数 """
45         def get_file_path(self, target_dir) :
46             """ 対象ディレクトリ下の . ファイルのパスを取得し
47                 dat, [target_filesに納める] """
48             target_files = []
49             for root, dir, files in os.walk(target_dir) :
50                 target_file = [os.path.join(root,f) for f in files
51                               if f.endswith(".dat")]# .txt -> .へ
52                               dat
53                 target_files.extend(target_file)
54             """ 取得した . ファイルのフルパスに作成時間を足してリストに納める
55                 dat """
56             file_ctime = []
57             for f in target_files :
58                 file_ctime.append((f,os.path.getctime(f)))
59             """ 取得時間でソートし最新の . ファイルのパスのみ返す dat """
60             sorted_file_ctime = sorted(file_ctime,key=lambda x :x
```

```
[1])

57
58         return sorted_file_ctime[len(sorted_file_ctime)-1][0]
59
60         """ 最終行を取得，シーケンス番号を加えてコピー """
61     def check_copy(self):
62         ##         name = self.target_file.replace(self.target_dir, "")
63         with open(self.target_file, "r") as f :
64             """ ファイルデータを全て読み込，最終行だけを取得 """
65             lines = f.readlines()
66             if len(lines) > 0 :
67                 line = lines[len(lines)-1]
68             else :
69                 line = self.old_line
70
71         """ 最終行が前回のものと異なるか? """
72         if line != self.old_line :
73
74             """ シーケンス番号を追加 """
75             self.s_num += 1
76
77             file_name = line.split(",")
78             file_name = "obs_" + "".join(file_name[0:3])
79             self.ymd = "".join(file_name[0:3])
80
81             self.old_line = line
82
83             """ にコピーDropbox """
84             with open(self.copy_dir+file_name+".csv", "a") as cf
85                 :
86                     data = line.strip() + "," + str(self.s_num)+"\n"
87                     "
88                     cf.write(str(data))
89
90             """ 最終行を保存 """
91             with open(self.temporary_log, "w") as f :
92                 f.write(line)
93
94             self.arduino_serial(data)
95             time.sleep(5)
96             self.TxMSG()
```

```

95         self.Ping()
96
97     else :
98         print("Not updated")
99         pass
100
101     """ を経由してにデータを送信する関数  arduinoLoRa """
102     def arduino_serial(self,d) :
103         print("---"*5 + "arduino_serial" + "---"*5)
104         buf = 0
105         with serial.Serial("COM7",115200,timeout=1) as ser :
106             """ ポートを開いて少し待機が必要 """
107             time.sleep(2)
108             """ごみの吸出し """
109             buf = ser.readlines()
110             d = d.strip()
111             """ 送信コマンドの形に """
112             d = "$RFSND,0004,"+d+"***"
113             print("To_arduino_Data-->" +d)
114
115             """ Python(PC) -> arduino -> LoRa だと文字ずつ送らないと
116                 いけない? 1 """
117             for i in d :
118                 ser.write(i.encode("utf-8"))
119
120             """ 0009 : 第二中継機にダミーをとばすMSG, 戻り値を保存 """
121     def TxMSG(self) :
122         target_add = "0009"
123         self.now = datetime.now().strftime("%Y,%m,%d,%H,%M,%S")
124         self.today = datetime.today().strftime("%Y%m%d")
125
126         msg = "$RFSND,{0},{1},{2},{2}***".format(target_add,
127             self.now,self.counter)
128
129         with serial.Serial("COM7",115200,timeout=15) as ser :
130             time.sleep(2)
131             for i in msg :
132                 ser.write(i.encode("utf-8"))
133                 time.sleep(0.05)
134             print(ser.readline().decode("utf-8"))
135             res = ser.readline().decode("utf-8")
```

```
134
135         if len(res) > 10 :
136             res = res.replace("□","").replace("*","").
                  replace(":","")
137             with open("C:/Users/taikimizukan/Dropbox/
                  sumitomo/RSSI_CHECK_TX/rssi_tx_obs_{}.csv".
                  format(str(self.today)),"a") as f :
138                 f.write(res+"\n")
139         else :
140             pass
141
142         """発電所のにを送って生存確認LoRaping"""
143     def Ping(self) :
144         PING = "$RPING,0004***"
145         with serial.Serial("COM7",115200,timeout=10) as ser :
146             time.sleep(2)
147             for i in PING :
148                 ser.write(i.encode("utf-8"))
149                 time.sleep(0.05)
150
151             print(ser.readline().decode("utf-8"))
152             res_ping = ser.readline().decode("utf-8")
153
154             if len(res_ping) > 10 :
155                 print(res_ping)
156                 now = datetime.now().strftime("%Y,%m,%d,%H,%M,%S")
157                 with open("C:/Users/taikimizukan/Dropbox/sumitomo/
                  PING/ping_{}.csv".format(str(self.today)),"a") as
                  f :
158                     f.write(str(now)+","+str(self.s_num)+"," +
                  str(res_ping))
159             else :
160                 pass
161
162         """ 繰り返し """
163     def Loop(self):
164         while True:
165             try :
166                 time.sleep(20)
167                 self.target_file = self.get_file_path(self.
                  target_dir)
```

```
168
169             self.check_copy()
170
171         except Exception as E :
172             with open("./Error_Log.txt","a") as ef :
173                 ef.write(str(E))
174
175
176 if __name__ == "__main__" :
177     Main()
```