

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 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 3 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

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 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 学内ゾーンにおける Elasticsearch クラスタへのデータ移行	3
2.1 節 緒言	3
2.2 節 学内ゾーンで稼働している Elasticsearch システムの状況	3
2.3 節 データ移行対象となる Elasticsearch インデックスについて	4
2.4 節 CO ₂ データの移行手順について	4
2.4.1 データのエクスポート	4
2.4.2 データの重複削除	4
2.4.3 データのインポート	6
2.5 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	6
2.6 節 kibana によるデータの可視化	7
2.7 節 LEAF の運行日誌に関するデータの移行について	9
2.8 節 LEAF の運行日誌に関するデータの移行手順について	11
2.8.1 データのエクスポート	11
2.8.2 データのインポート	11
2.9 節 結言	11
第 3 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	12
3.1 節 緒言	12

3.2 節	サーバーゾーンで稼働している Elasticsearch システム の状況	12
3.3 節	Docker とは	13
3.3.1	コンテナとは	13
3.3.2	Docker イメージとは	14
3.4 節	Docker Compose とは	14
3.5 節	検証環境のセットアップ	14
3.5.1	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) . . .	14
3.5.2	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)	18
3.6 節	異なる Elasticsearch クラスタへのノード参加検証 . . .	19
3.7 節	手順	19
3.7.1	単一ノードで稼働するクラスタ A の構築	19
3.7.2	クラスタ B の構築	21
3.7.3	クラスタ B への参加試行	25
3.8 節	まとめ	29
3.9 節	結言	29
第 4 章	4 章のタイトル	30
4.1 節	緒言	30
4.2 節	結言	30
第 5 章	結論と今後の課題	31
	謝 辞	32
	参考文献	33
付録 A	付録	34
A.1	水源監視システム (送信用 Python スクリプト)	34
A.2	水源監視システム (受信用 Python スクリプト)	40

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

学内ゾーンにおける Elasticsearch クラスタへのデータ移行

2.1 節 緒言

本章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。

2.2 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 が稼働しており, CO₂ 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。

他には, 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。

2.3 節 データ移行対象となる 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。

2.4 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ったので、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アップロードした。

2.4.1 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。

2.4.2 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 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 (JPtime) を一意のキーとして設定した。以下のリスト 3.3, リスト 3.4 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 2.1 _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 2.2 _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"  
}  
}
```

2.4.3 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。

2.5 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 の計測システムを開発、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 (`Jptime`) 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.

2.6 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 2.1 ~ 図 2.3 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 2.1 ~ 図 2.3 より, 連続的にデータ

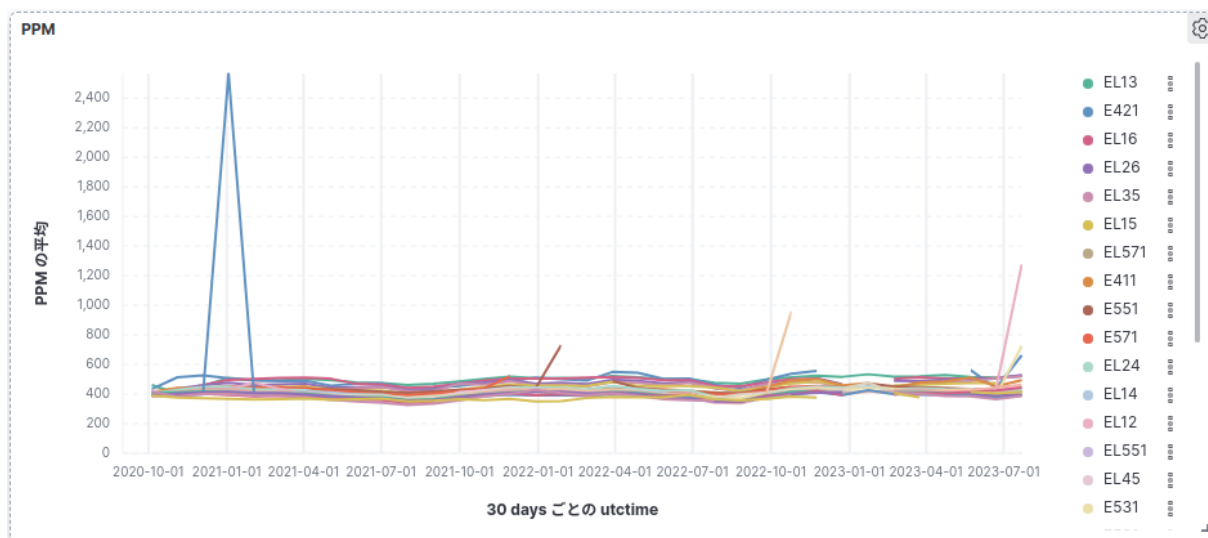


図 2.1 co2_modbus の PPM

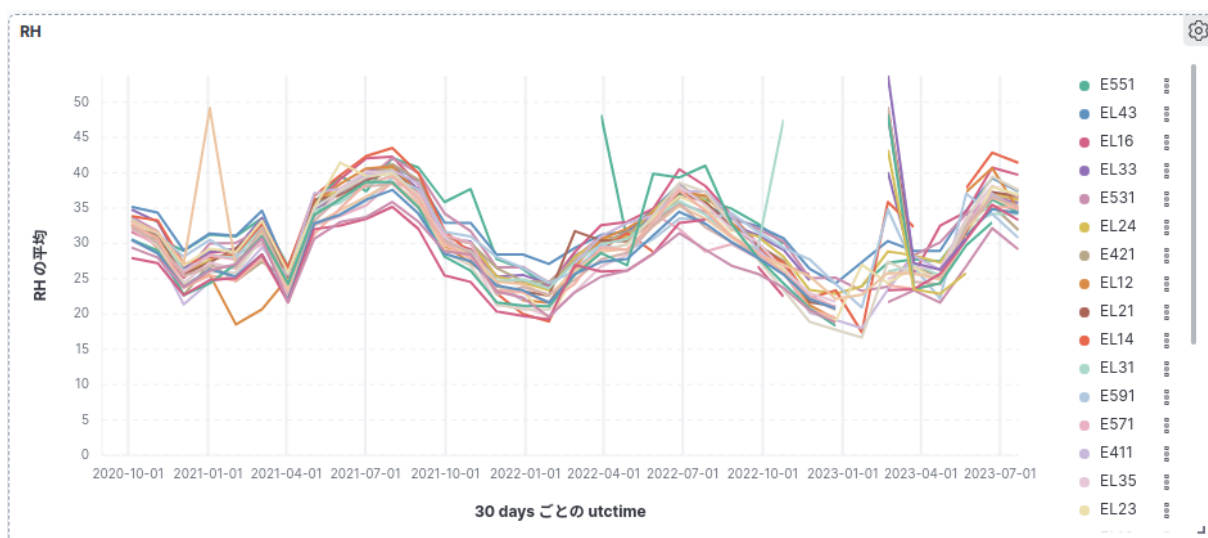


図 2.2 co2_modbus の RH

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。

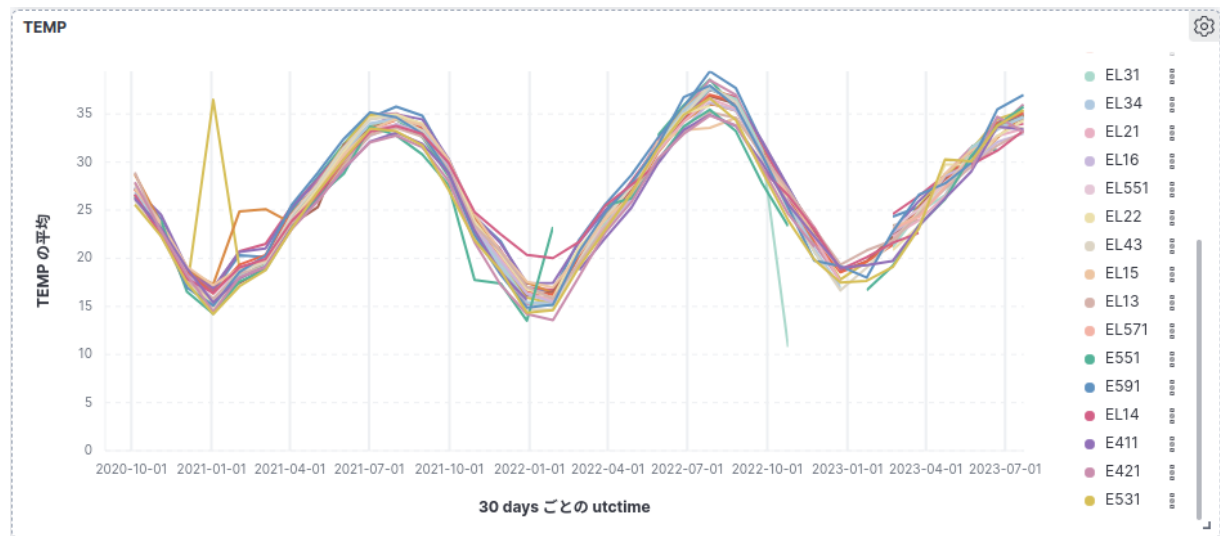


図 2.3 co2_modbus の TEMP

2.7 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.

2.8 節 LEAF の運行日誌に関するデータの移行手順について

2.8.1 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.

2.8.2 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.

2.9 節 結言

本章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3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

3.1 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。

3.2 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。

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.

3.3 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.

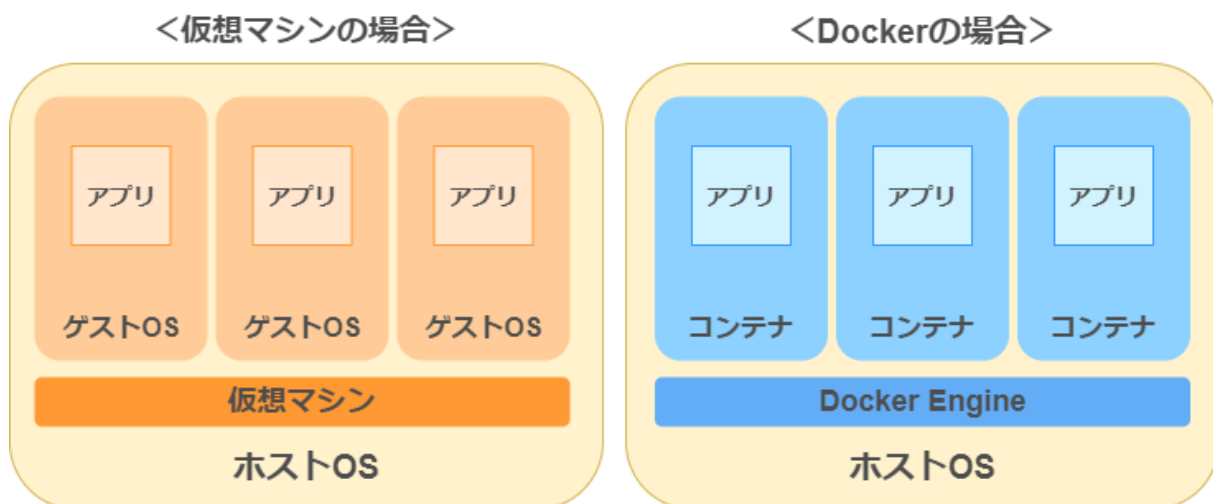


図 3.1 仮想マシンと Docker の違い [2]

3.3.1 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.

3.3.2 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.

3.4 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.

3.5 節 検証環境のセットアップ

3.5.1 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 3.3 にクラスタを構成するのに使用した docker-compose.yml ファイルの内容を記載する.

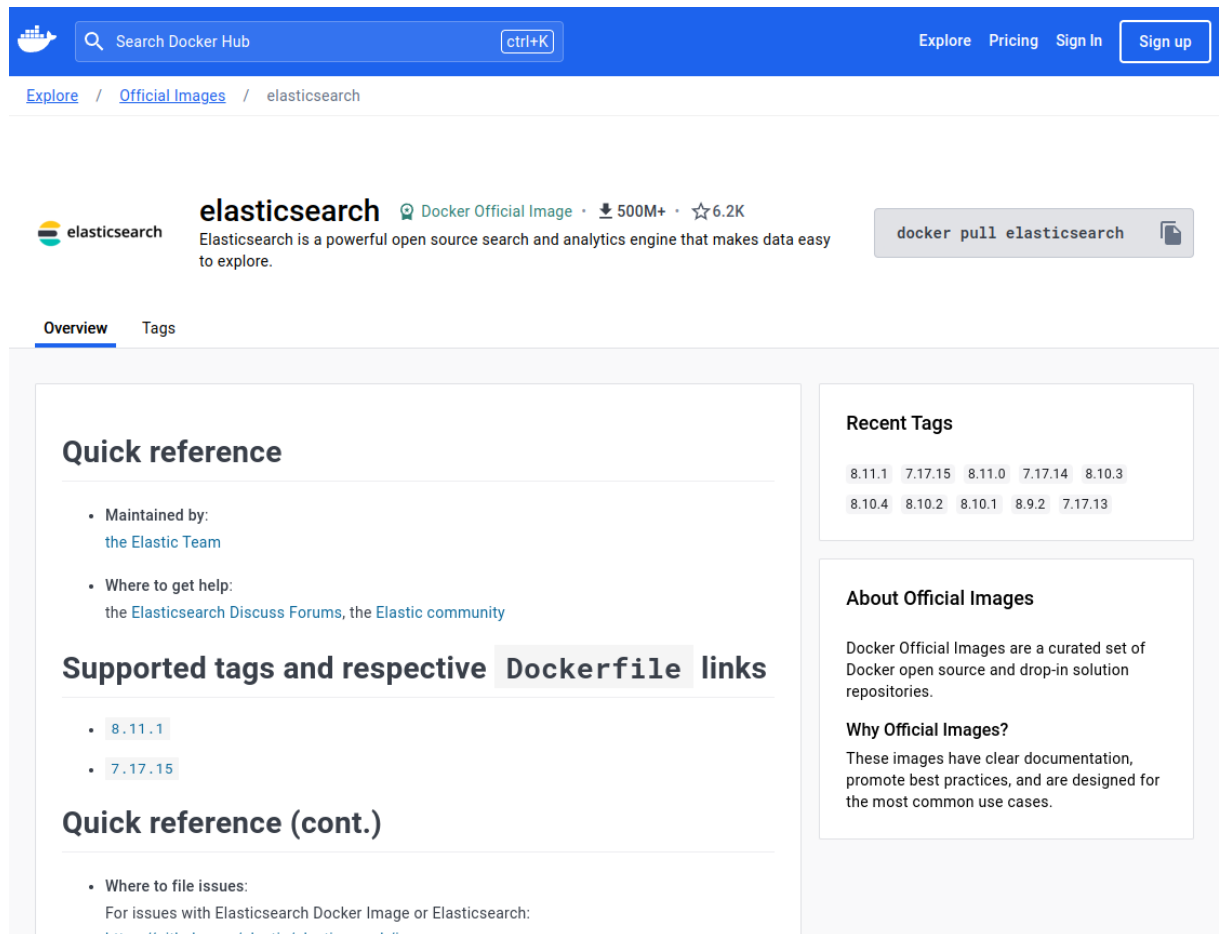


図 3.2 Elasticsearch の Docker イメージ

Listing 3.1 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
    - cluster.name=es-docker-cluster
    - discovery.seed_hosts=es01,es02
    - cluster.initial_master_nodes=es01,es02,es03
  networks:
    - elastic

networks:
  elastic:
    driver: bridge
```

Listing 3.3 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.

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 3.3 に示す.

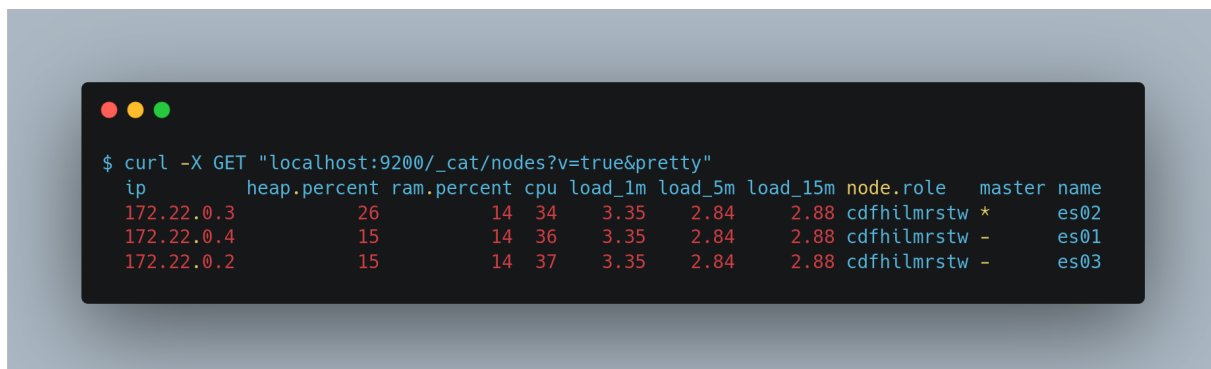


図 3.3 クラスタに参加しているノードを一覧表示した結果

図 3.3 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。

3.5.2 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)

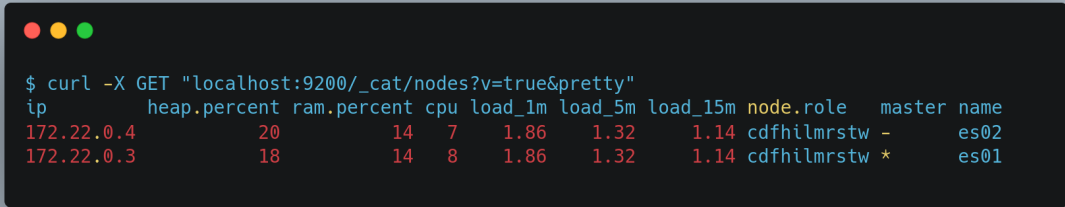
Listing 3.3 の docker-compose.yml の es03 のコンテナが使用する Docker イメージを Listing 3.4 に変更することで、es03 のノードで使用する Elasticsearch のバージョンを 7.17.9 から 7.17.6 に変更する。

Listing 3.2 Listing 3.3 の docker-compose.yml から変更を加えた箇所

```
version: '2.2'
services:
  ...
  es03:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.17.6
  ...
...
```

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 3.4 に示す.



```
$ 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 3.4 クラスタに参加しているノードを一覧表示した結果

図 3.4 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 3.5 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.

3.6 節 異なる Elasticsearch クラスタへのノード参加検証

今回は, CO₂ データなどが保存されている 3 ノードで構築されたクラスタに対して, 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 ノードが新たなノードとしてクラスタに参加できるか, Docker を用いて検証した.

3.7 節 手順

3.7.1 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.

```
es03 | {"type": "server", "timestamp": "2023-11-20T05:23:44,794Z", "level": "INFO", "component": "o.e.p.PluginsService",  
"cluster.name": "es-docker-cluster", "node.name": "es03", "message": "no plugins loaded" }  
es03 | {"type": "server", "timestamp": "2023-11-20T05:23:44,843Z", "level": "INFO", "component": "o.e.e.NodeEnvironment",  
"cluster.name": "es-docker-cluster", "node.name": "es03", "message": "using [1] data paths, mounts  
[[/usr/share/elasticsearch/data (/dev/sda6)]], net usable_space [153.4gb], net total_space [206.6gb], types [ext4]" }  
es03 | {"type": "server", "timestamp": "2023-11-20T05:23:44,843Z", "level": "INFO", "component": "o.e.e.NodeEnvironment",  
"cluster.name": "es-docker-cluster", "node.name": "es03", "message": "heap size [2gb], compressed ordinary object pointers  
[true]" }  
es01 | {"type": "server", "timestamp": "2023-11-20T05:23:44,872Z", "level": "INFO", "component": "o.e.n.Node",  
"cluster.name": "es-docker-cluster", "node.name": "es01", "message": "node name [es01], node ID [xeg-kFWGRMmh8M4u8A-ZEA],  
cluster name [es-docker-cluster], roles [transform, data_frozen, master, remote_cluster_client, data, ml, data_content,  
data_hot, data_warm, data_cold, ingest]" }  
es03 | {"type": "server", "timestamp": "2023-11-20T05:23:44,902Z", "level": "ERROR", "component":  
"o.e.b.ElasticsearchUncaughtExceptionHandler", "cluster.name": "es-docker-cluster", "node.name": "es03", "message":  
"uncaught exception in thread [main]",  
es03 | "stacktrace": ["org.elasticsearch.bootstrap.StartupException: java.lang.IllegalStateException: cannot downgrade a  
node from version [7.17.9] to version [7.17.6]",  
es03 | "at org.elasticsearch.bootstrap.Elasticsearch.init(Elasticsearch.java:173) ~[elasticsearch-7.17.6.jar:7.17.6]",  
es03 | "at org.elasticsearch.bootstrap.Elasticsearch.execute(Elasticsearch.java:160) ~[elasticsearch-7.17.6.jar:7.17.6]",  
es03 | "at org.elasticsearch.cli.EnvironmentAwareCommand.execute(EnvironmentAwareCommand.java:77) ~[elasticsearch-  
7.17.6.jar:7.17.6]",  
es03 | "at org.elasticsearch.cli.Command.mainWithoutErrorHandling(Command.java:112) ~[elasticsearch-cli-  
7.17.6.jar:7.17.6]",  
es03 | "at org.elasticsearch.cli.Command.main(Command.java:77) ~[elasticsearch-cli-7.17.6.jar:7.17.6]",  
es03 | "at org.elasticsearch.bootstrap.Elasticsearch.main(Elasticsearch.java:125) ~[elasticsearch-7.17.6.jar:7.17.6]",  
es03 | "at org.elasticsearch.bootstrap.Elasticsearch.main(Elasticsearch.java:80) ~[elasticsearch-7.17.6.jar:7.17.6]",  
es03 | "Caused by: 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) ~[elasticsearch-  
7.17.6.jar:7.17.6]",  
es03 | "at org.elasticsearch.env.NodeEnvironment.loadNodeMetadata(NodeEnvironment.java:484) ~[elasticsearch-  
7.17.6.jar:7.17.6]",  
es03 | "at org.elasticsearch.env.NodeEnvironment.<init>(NodeEnvironment.java:356) ~[elasticsearch-7.17.6.jar:7.17.6]",  
es03 | "at org.elasticsearch.node.Node.<init>(Node.java:429) ~[elasticsearch-7.17.6.jar:7.17.6]",  
es03 | "at org.elasticsearch.node.Node.<init>(Node.java:309) ~[elasticsearch-7.17.6.jar:7.17.6]",  
es03 | "at org.elasticsearch.bootstrap.Bootstrap$.<init>(Bootstrap.java:234) ~[elasticsearch-7.17.6.jar:7.17.6]",  
es03 | "at org.elasticsearch.bootstrap.Bootstrap.setup(Bootstrap.java:234) ~[elasticsearch-7.17.6.jar:7.17.6]",  
es03 | "at org.elasticsearch.bootstrap.Bootstrap.init(Bootstrap.java:434) ~[elasticsearch-7.17.6.jar:7.17.6]",  
es03 | "at org.elasticsearch.bootstrap.Elasticsearch.init(Elasticsearch.java:169) ~[elasticsearch-7.17.6.jar:7.17.6]",  
es03 | "... 6 more" ] }  
es03 | uncaught exception in thread [main]  
es03 | java.lang.IllegalStateException: cannot downgrade a node from version [7.17.9] to version [7.17.6]  
es03 |     at org.elasticsearch.env.NodeMetadata.upgradeToCurrentVersion(NodeMetadata.java:95)  
es03 |     at org.elasticsearch.env.NodeEnvironment.loadNodeMetadata(NodeEnvironment.java:484)  
es03 |     at org.elasticsearch.env.NodeEnvironment.<init>(NodeEnvironment.java:356)  
es03 |     at org.elasticsearch.node.Node.<init>(Node.java:429)  
es03 |     at org.elasticsearch.node.Node.<init>(Node.java:309)  
es03 |     at org.elasticsearch.bootstrap.Bootstrap$.<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 3.5 es03 のログ

Listing 3.3 にクラスタ A の構築の際に使用した docker-compose.yml を示す。

Listing 3.3 クラスタ A の構築の際に使用した docker-com-


```
pose.yml
version: '2.2'
services:
  es04:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.17.9
    container_name: es04
    environment:
      - node.name=es04
      - discovery.type=single-node
    volumes:
      - data04:/usr/share/elasticsearch/data
    ports:
      - 9200:9200
    networks:
      - esnet
```

```
volumes:
  data04:
    driver: local
```

```
networks:
  esnet:
```

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 3.6 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。

3.7.2 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 (以後このクラスタをクラスタ B と呼ぶ) を構

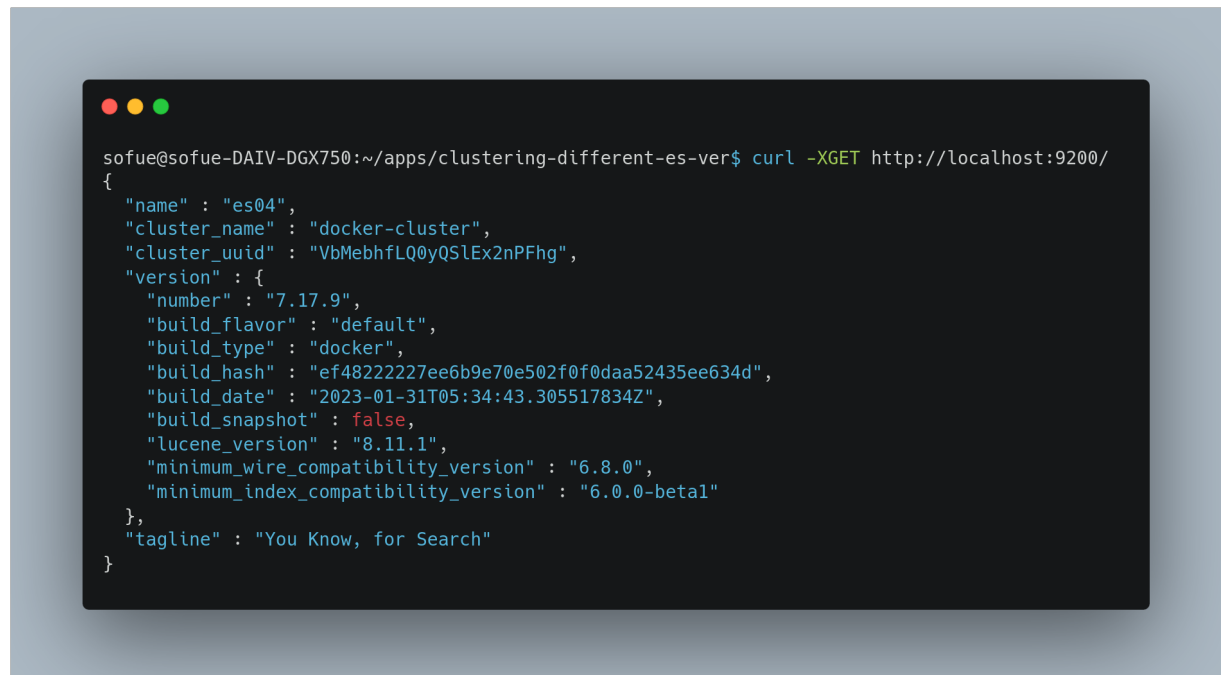


図 3.6 クラスタの情報について問い合わせた結果

築する。

Listing 3.4 にクラスタ B の構築の際に使用した docker-compose.yml を示す。

Listing 3.4 クラスタ B の構築の際に使用した docker-compose.yml

```
version: '2.2'
services:
  es01:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.17.9
    container_name: es01
    environment:
      - node.name=es01
      - discovery.seed_hosts=es01,es02,es03
      - cluster.initial_master_nodes=es01,es02,es03
volumes:
```

```
    - data01:/usr/share/elasticsearch/data
  ports:
    - 9200:9200
  networks:
    - elastic
es02:
  image: docker.elastic.co/elasticsearch/elasticsearch:7.17.9
  container_name: es02
  environment:
    - node.name=es02
    - discovery.seed_hosts=es01,es02,es03
    - cluster.initial_master_nodes=es01,es02,es03
  volumes:
    - data02:/usr/share/elasticsearch/data
  networks:
    - elastic
es03:
  image: docker.elastic.co/elasticsearch/elasticsearch:7.17.9
  container_name: es03
  environment:
    - node.name=es03
    - discovery.seed_hosts=es01,es02,es03
    - cluster.initial_master_nodes=es01,es02,es03
  volumes:
    - data03:/usr/share/elasticsearch/data
  networks:
    - elastic

volumes:
  data01:
    driver: local
```

```
data02:
  driver: local
data03:
  driver: local

networks:
  elastic:
    driver: bridge
```

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 3.7 に示す。

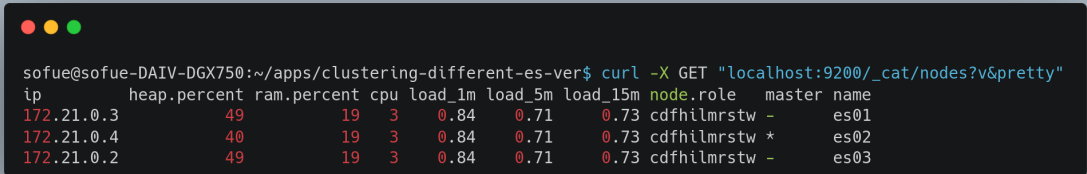


```
sofue@sofue-DAIV-DGX750:~/apps/clustering-different-es-ver$ curl -XGET http://localhost:9200/
{
  "name" : "es01",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "XNQVSiDyTkuytN_rKQt4w",
  "version" : {
    "number" : "7.17.9",
    "build_flavor" : "default",
    "build_type" : "docker",
    "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 3.7 クラスタの情報について問い合わせた結果

図 3.6, 3.7 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 3.8 に示す。



```
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 cdfhilmrstw -   es01
172.21.0.4      40           19      3    0.84    0.71    0.73 cdfhilmrstw *   es02
172.21.0.2      49           19      3    0.84    0.71    0.73 cdfhilmrstw -   es03
```

図 3.8 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 3.8 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。

3.7.3 クラスタ B への参加試行

次に, Listing 3.4 の docker-compose.yml に, 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。

Listing 3.5 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 を示す。

Listing 3.5 合計 4 ノードでクラスタ B を起動する際に使用した docker-compose.yml

```
version: '2.2'
services:
  es01:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.17.9
    container_name: es01
    environment:
      - node.name=es01
```

```
    - discovery.seed_hosts=es01 , es02 , es03 , es04
    - cluster.initial_master_nodes=es01 , es02 , es03 , es04
volumes:
    - data01:/usr/share/elasticsearch/data
ports:
    - 9200:9200
networks:
    - elastic
es02:
image: docker.elastic.co/elasticsearch/elasticsearch:7.17.9
container_name: es02
environment:
    - node.name=es02
    - discovery.seed_hosts=es01 , es02 , es03 , es04
    - cluster.initial_master_nodes=es01 , es02 , es03 , es04
volumes:
    - data02:/usr/share/elasticsearch/data
networks:
    - elastic
es03:
image: docker.elastic.co/elasticsearch/elasticsearch:7.17.9
container_name: es03
environment:
    - node.name=es03
    - discovery.seed_hosts=es01 , es02 , es03 , es04
    - cluster.initial_master_nodes=es01 , es02 , es03 , es04
volumes:
    - data03:/usr/share/elasticsearch/data
networks:
    - elastic
es04:
```

```
image: docker.elastic.co/elasticsearch/elasticsearch:7.17.9
container_name: es04
environment:
  - node.name=es04
  - discovery.seed_hosts=es01,es02,es03,es04
  - cluster.initial_master_nodes=es01,es02,es03,es04
volumes:
  - data04:/usr/share/elasticsearch/data
networks:
  - elastic
```

```
volumes:
  data01:
    driver: local
  data02:
    driver: local
  data03:
    driver: local
  data04:
    driver: local
```

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

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 3.9 に示す。

図 3.9 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 3.10 に示す。

図 3.10 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 3.9 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 [VbMebhflQ0yQSlEx2nPfHg] and is now trying to join a different cluster with UUID [XNOVSI0yTkuytnT_rK0t4w]. 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 3.10 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。

3.8 節 まとめ

今回は、CO₂ データなどが保存されている 3 ノードで構築されたクラスタに対して、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 ノードが新たなノードとしてクラスタに参加できるか、Docker を用いて検証した。

検証の結果、Elasticsearch のノードは異なるクラスタ ID を持つクラスタに参加することはできないことが分かった。ノードが別のクラスタに参加するためには、インデックスやドキュメント情報などを格納しているデータパスのフォルダやファイルを削除する必要があるが、これはそのノードのデータを失うことを意味する。

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。

3.9 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4章

4章のタイトル

4.1 節 緒言

本章では...について述べる .

4.2 節 結言

本章では...について述べた .

第5章

結論と今後の課題

.....

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。

最後に、有益な御助言を賜りました本大学大学院の〇〇に心より御礼申し上げます。

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                 target_files.extend(target_file)
53             """ 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
54                 dat """
55             file_ctime = []
56             for f in target_files :
57                 file_ctime.append((f,os.path.getctime(f)))
58             """ 取得時間でソートし最新の . ファイルのパスのみ返す dat """
59             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()
```

A.2 水源監視システム (受信用 Python スクリプト)

LoRa_obs_receive.py のソースコードを A.2 に示す。

Listing A.2 LoRa_obs_raceive.py

```
1  ## coding:utf-8
2
3  import paho.mqtt.client as mqtt
4  import serial
5  from datetime import datetime
6  import time
7
8  class Main() :
9      def __init__(self) :
10         self.INF_Input()
11         self.Loop()
12
13     def INF_Input(self) :
14         """ 起動時 [$RFINF,ONコマンド送信***] """
15
16         INF = "$RFINF,ON***"
17         with serial.Serial("COM3",115200,timeout=2) as ser :
18             time.sleep(2)
19             while True :
20                 for i in INF :
21                     ser.write(i.encode("utf-8"))
22
23                 result = str(ser.readline())
24                 if result.find("RESULT,RFINF,ON,OK") > 0 :
25                     print("RFINF,OK")
26                     break
27                 else :
28                     time.sleep(2)
29
30     def LoRa_Receive(self) :
31         try :
32             """ からデータを読み込むLoRa """
33             while True :
34                 with serial.Serial("COM3",115200,timeout=120)
35                     as ser :
```

```
35         res = ser.readline().decode("utf-8")
36         if len(res) > 15 and res.find("RFRX") > 0
           and res.find("RECEIVED") < 0 :
37             print("==="*25)
38             print("RXData_<_<=>_<_"+res)
39             break
40
41         else :
42             print("==="*25)
43             print("else_data=>"+res)
44
45         """ 必要なデータを取り出す """
46         res_list = res.split("*")[0].split(",")[1:]
47         data = ",".join(res_list)
48         add = res_list[0]
49
50         """ データの日付を確認(ymd) """
51         ymd = ",".join(res_list[1:4])
52
53
54         return = res, data, add, ymd
55
56     except Exception as E :
57         now = datetime.now().strftime("%y/%m/%d_%H:%M:%S")
58         with open("LoRa_Receive_Error_LOG.txt","a") as ef :
59             ef.write(now + "_:_"+ str(E)+"\n")
60         self.Loop()
61
62
63
64     def MQTT_Publish(self, res):
65         """ 情報MQTT(publish) """
66         host = "133.71.***.***"
67         port = 1883
68         topic = "*****"
69         """ MQTT-Publish """
70         try :
71             print("publish_<_<=>_<_"+str(res))
72             client = mqtt.Client(protocol=mqtt.MQTTv311)
73             client.connect(host,port=port,keepalive=10)
74             client.publish(topic,res)
```

```
75
76         except Exception as E :
77             now = datetime.now().strftime("%y/%m/%d_%H:%M:%S")
78             with open("Publish_Error_LOG.txt","a") as ef :
79                 ef.write(now + "_:" + str(E)+"\n")
80
81     def Storage(self,res,data,ymd):
82         try :
83             """ メタデータを保存 """
84             with open("./meta_data/meta_data_{}.txt".format(ymd)
85                 , "a") as f :
86                 f.write(res+"\n")
87
88             """ データを保存 """
89             with open("./data/data_{}.txt".format(ymd),"a") as
90                 f :
91                 f.write(data+"\n")
92
93             """ に保存Dropbox """
94             with open("C:/Users/sumitomo02/Dropbox/test_folder/
95                 RX/DATA/RX_{}.txt".format(ymd),"a") as f :
96                 f.write(data+"\n")
97
98             with open("C:/Users/sumitomo02/Dropbox/test_folder/
99                 RX/META_DATA/RX_{}.txt".format(ymd),"a") as f :
100                 f.write(res+"\n")
101
102         except Exception as E :
103             now = datetime.now().strftime("%y/%m/%d_%H:%M:%S")
104             with open("Storage_Error_LOG.txt","a") as ef :
105                 ef.write(now + "_:" + str(E)+"\n")
106         print(E)
107
108     def Loop(self) :
109         while True :
110             res, data, add, ymd = self.LoRa_Receive()
111             self.Storage(res, data, ymd)
112             self.MQTT_Publish(res)
113             self.Ping(add)
```

```
112 main = Main()
```