l'école multimédia

Projet final - Big Data

Etude des données météorologiques

NOVOA Y RODRIGUEZ
Miguel-Angel

Sommaire

1. Présentation du projet

Contexte

Le projet consiste à concevoir et déployer une application web sécurisée pour analyser des données météorologiques structurées et non structurées. Les technologies employées incluent l'écosystème Hadoop pour le stockage et le traitement des données volumineuses, ainsi qu'ElasticSearch pour les recherches avancées.

Les données utilisées proviennent exclusivement du site de la *National Oceanic and Atmospheric Administration* (NOAA), une source de données fiables sur les conditions météorologiques et climatiques.
En raison de ressources limitées, l'environnement sera simulé dans des conteneurs Docker, reproduisant les conditions d'une infrastructure de production.

Présentation

L'objectif principal est de répondre aux besoins d'une startup spécialisée dans l'analyse des données météorologiques. Cette entreprise vise à :

- Prédire les tendances climatiques.
- Aider les agriculteurs à optimiser leurs récoltes en fonction des prévisions.

Votre mission couvre l'ensemble du cycle de vie du projet : de la collecte et du stockage des données à leur visualisation via une application web. Vous devrez également assurer la sécurisation des données et leur accessibilité en production.

2. Objectifs

Les objectifs du projet se déclinent en plusieurs étapes clés :

1. Installation et configuration de l'écosystème Hadoop dans Docker

- Mettre en place un cluster Hadoop pour le traitement parallèle des données volumineuses.
- Configurer les outils complémentaires de l'écosystème Hadoop tels que HDFS (pour le stockage) et YARN (pour la gestion des ressources).

2. Conception et déploiement d'un entrepôt de données

- Stocker les données météorologiques structurées (ex. tables de mesures) et non structurées (ex. fichiers CSV, JSON).
- Assurer une organisation optimale des données pour faciliter leur accès et leur analyse.

3. Définition de l'architecture des données

- Élaborer un diagramme des flux de données depuis leur collecte sur le site NOAA jusqu'à leur stockage et traitement.
- Identifier les outils et pipelines nécessaires pour chaque étape du traitement.

4. Mise en place d'ElasticSearch

- Configurer ElasticSearch pour indexer les données et permettre des recherches avancées sur des critères multiples.
- Tester des requêtes typiques pour s'assurer de la pertinence des résultats retournés.

5. Développement de requêtes SQL et NoSQL

- Écrire des requêtes SQL pour analyser les données structurées dans Hadoop.
- Utiliser des outils NoSQL comme HBase ou MongoDB pour interroger les données non structurées.

6. Sécurisation des bases de données et sauvegardes

- Mettre en place des mécanismes de contrôle d'accès pour protéger les données sensibles.
- Définir et tester des procédures de sauvegarde et de restauration en cas de panne.

7. Développement d’une application web sécurisée

- Concevoir une interface utilisateur intuitive pour la visualisation des données et l’interaction.
- Intégrer des graphiques et des tableaux pour représenter les tendances météorologiques.
- Garantir la sécurité de l’application via l’authentification et la gestion des droits des utilisateurs.

3. Présentation des technologies

3.1 Hadoop

Hadoop est un framework open-source conçu pour le stockage et le traitement de grandes quantités de données. Il repose sur deux composants principaux :

- **HDFS (Hadoop Distributed File System)** : permet de stocker les données sur plusieurs machines de manière distribuée.
- **YARN (Yet Another Resource Negotiator)** : gère les ressources et les tâches de calcul dans le cluster Hadoop.

Avantages :

- Capacité à traiter des données massives.
- Scalabilité horizontale (ajout de machines sans perturber le système).
- Robustesse face aux pannes grâce à la réplication des données.

Utilisation dans le projet :
Hadoop sera utilisé pour stocker les données météorologiques volumineuses provenant de la NOAA et permettre leur traitement à grande échelle.

3.2 Hive

Hive est un outil d'entrepôt de données construit sur Hadoop, permettant de transformer des requêtes SQL (appelées HiveQL) en tâches MapReduce.

- **Fonctionnalités clés** :
    - Simplifie l'accès aux données en utilisant un langage proche du SQL.
    - Convient pour des requêtes analytiques sur de grandes tables.

Avantages :

- Accès facilité aux données Hadoop grâce à une interface SQL-like.
- Intégration avec d'autres outils de l'écosystème Hadoop.

Utilisation dans le projet :
Hive sera utilisé pour exécuter des requêtes SQL sur les données structurées provenant de la NOAA.

3.3 HBase

HBase est une base de données NoSQL distribuée et orientée colonnes, fonctionnant sur Hadoop. Elle est idéale pour gérer des données non structurées ou semi-structurées.

- **Fonctionnalités clés** :
    - Performances élevées pour les opérations de lecture/écriture.
    - Gestion des données en temps réel.

Avantages :

- Conçu pour traiter des volumes massifs de données avec une faible latence.
- Compatible avec Hadoop pour un stockage efficace.

Utilisation dans le projet :
HBase sera utilisé pour stocker et accéder rapidement aux données non structurées de la NOAA, comme des fichiers JSON ou des flux de données temps réel.

3.4 ElasticSearch

ElasticSearch est un moteur de recherche et d'analyse distribué, capable d'indexer de grandes quantités de données en temps réel.

- **Fonctionnalités clés** :
    - Recherche full-text rapide et précise.
    - Support des agrégations pour des analyses approfondies.
    - Visualisation intégrée via Kibana.

Avantages :

- Recherches avancées grâce à un langage de requête riche (DSL).
- Haute disponibilité et scalabilité grâce à sa structure distribuée.

Utilisation dans le projet :
ElasticSearch sera utilisé pour indexer les données NOAA, permettant de réaliser des recherches rapides et complexes sur les informations météorologiques.

3.5 Spark

Apache Spark est un moteur de traitement de données rapide, conçu pour le calcul distribué sur des clusters. Contrairement à MapReduce, Spark exécute des tâches en mémoire, offrant des performances accrues.

- **Fonctionnalités clés** :
    - Support pour les tâches de traitement en batch et en streaming.
    - API riches pour plusieurs langages (Java, Scala, Python, R).

    - Bibliothèques intégrées pour le machine learning, le traitement de graphes, etc.

Avantages :

- Vitesse élevée grâce au traitement en mémoire.
- Compatible avec Hadoop, HDFS et d'autres outils du Big Data.

Utilisation dans le projet :
Spark sera utilisé pour des analyses avancées des données NOAA, notamment pour détecter des tendances climatiques et effectuer des calculs complexes.

3.6 FastAPI

FastAPI est un framework web moderne et rapide utilisé pour créer des APIs RESTful en Python. Il est conçu pour offrir des performances élevées et une intégration fluide avec les fonctionnalités modernes de Python comme les annotations de type.

Rôle de FastAPI dans le projet :

FastAPI agit comme le **backend** principal, jouant un rôle central dans la communication entre les utilisateurs et les différents composants du système.

1. **Interface API** :
    - FastAPI expose des points d'entrée RESTful pour que l'utilisateur puisse interagir avec les données météorologiques (recherches, filtrages, etc.).
2. **Communication avec ElasticSearch** :
    - Il transmet les requêtes de l'interface utilisateur (React.js) vers ElasticSearch, qui récupère les données de Hive et HBase.
3. **Sécurité** :
    - Authentifie les utilisateurs et sécurise l'accès aux API grâce à des mécanismes comme **JWT (JSON Web Tokens)**.

Avantages de FastAPI dans le projet :

- **Performance élevée** pour gérer un grand nombre de requêtes simultanées.
- **Documentation automatique** pour simplifier le test des APIs.

- **Intégration fluide** avec ElasticSearch et les autres composants.

FastAPI garantit une communication fluide, sécurisée et rapide entre les différents éléments de l'architecture du projet.

3.7 React.js

React.js est une bibliothèque JavaScript utilisée pour créer des interfaces utilisateur dynamiques et réactives.

Rôle de React.js dans le projet :

1. **Interface utilisateur** :
    - Fournit une plateforme intuitive pour que les utilisateurs puissent visualiser et interagir avec les données météorologiques.
2. **Communication avec FastAPI** :
    - Envoie des requêtes aux points d'entrée de l'API backend (FastAPI) pour récupérer les données nécessaires.
3. **Visualisation des données** :
    - Affiche les données sous forme de tableaux, graphiques et cartes interactives à l'aide de bibliothèques comme **Chart.js**.

Avantages de React.js dans le projet :

- **Réactivité** pour une expérience utilisateur fluide.
- **Composants réutilisables** pour un développement rapide.
- **Interopérabilité** avec FastAPI et les bibliothèques de visualisation de données.

React.js assure une interface moderne et performante pour interagir avec le système global.

3.8 Docker

Docker est une plateforme de conteneurisation permettant de packager, de distribuer et d'exécuter des applications dans des environnements isolés, appelés **conteneurs**. Ces conteneurs garantissent la portabilité des applications et leur fonctionnement cohérent, quelle que soit la machine hôte. Docker est particulièrement utile pour simuler des environnements de production complexes tout en minimisant les frais d'infrastructure.

Fonctionnement de Docker :

1. **Images Docker** :
   Une image Docker est un modèle immuable qui contient tout ce qui est nécessaire pour exécuter une application (code, bibliothèques, dépendances). Une image sert de base pour lancer un conteneur.
2. **Conteneurs Docker** :
   Les conteneurs sont des instances exécutables d'une image Docker. Ils sont isolés les uns des autres et partagent les ressources de la machine hôte, ce qui les rend légers et efficaces.
3. **Docker Compose** :
   Docker Compose est un outil qui permet de définir et de gérer des applications multi-conteneurs. Il utilise un fichier YAML (`docker-compose.yml`) pour spécifier la configuration des conteneurs et leur orchestration.

Rôle de Docker dans le projet :

Dans le cadre de ce projet, Docker est utilisé pour simuler un environnement de production complet et garantir une portabilité optimale entre les différents systèmes de développement et de déploiement.

- **Hadoop** : Le cluster Hadoop est déployé à l'aide de plusieurs conteneurs.
  - **NameNode** : Conteneur dédié à la gestion des métadonnées des fichiers.
  - **DataNode(s)** : Un ou plusieurs conteneurs pour stocker les blocs de données.
- **Spark** : Exécute les transformations des données dans un conteneur dédié.
- **Hive et HBase** : Chacun est déployé dans un conteneur pour gérer respectivement les données structurées et non structurées.
- **ElasticSearch** : Un conteneur ElasticSearch est configuré pour indexer et rechercher les données.
- **FastAPI** : L'API backend est contenue dans un conteneur distinct, isolé mais interconnecté avec les autres services.
- **React.js** : L'interface utilisateur frontend est également contenue dans un conteneur, facilitant la communication avec le backend.

Avantages de Docker dans ce projet :

1. **Isolation** : Chaque composant du projet (Hadoop, Spark, ElasticSearch, etc.) fonctionne indépendamment dans son propre conteneur, ce qui évite les conflits.
2. **Portabilité** : Les conteneurs garantissent que l'application fonctionnera de la même manière sur n'importe quelle machine ou infrastructure cloud.
3. **Facilité de mise à jour** : Les images peuvent être mises à jour sans impacter les autres conteneurs.

4. **Orchestration simplifiée** : Grâce à Docker Compose, tous les services sont démarrés simultanément avec des configurations prédéfinies.
5. **Optimisation des ressources** : Contrairement aux machines virtuelles, Docker utilise les ressources de la machine hôte sans surcouche importante.

Exemple de fichier docker-compose.yml :

```yaml
services:
  namenode:
    build:
      context: ./namenode
    container_name: namenode
    hostname: namenode
    ports:
      - "9870:9870"  # Hadoop NameNode Web UI
      - "8088:8088"  # Hadoop ResourceManager Web UI
    volumes:
      - namenode_data:/hadoopdata
    environment:
      - HDFS_NAMENODE_USER=root
      - HDFS_DATANODE_USER=root
      - HDFS_SECONDARYNAMENODE_USER=root
      - YARN_RESOURCEMANAGER_USER=root
      - YARN_NODEMANAGER_USER=root
    entrypoint: ["/entrypoint.sh"]
    networks:
      - hadoop-net

  datanode1:
    build:
      context: ./datanode
    container_name: datanode1
    hostname: datanode1
    depends_on:
      - namenode
    volumes:
      - datanode1_data:/hadoopdata
    environment:
      - HDFS_NAMENODE_USER=root
      - HDFS_DATANODE_USER=root
      - HDFS_SECONDARYNAMENODE_USER=root
      - YARN_RESOURCEMANAGER_USER=root
      - YARN_NODEMANAGER_USER=root
    entrypoint: ["/entrypoint.sh"]
    networks:
      - hadoop-net
```

4. Application des technologies sur notre projet

Notre architecture repose sur un écosystème Big Data entièrement conteneurisé avec Docker. Chaque composant joue un rôle spécifique dans le pipeline de traitement des données météorologiques issues de la NOAA. Voici une explication détaillée :

4.1 Récupération et traitement des données avec une ETL

Dans notre projet, une **ETL** (Extract, Transform, Load) est mise en place pour récupérer les données météorologiques du site NOAA. Les données sont principalement fournies sous forme de fichiers CSV ou de dossiers compressés pour les “storm events”.

Processus de l'ETL :

1. **Extraction** :
    - Téléchargement des fichiers depuis le site NOAA.
    - Décompression des fichiers, si nécessaire.
2. **Transformation** :
    - Nettoyage des données : suppression des valeurs manquantes ou incorrectes, correction des formats de dates, etc.
3. **Chargement** :
    - Chargement des données transformées dans **Hadoop** pour un stockage et un traitement ultérieur.

4.2 Stockage des données avec Hadoop

Dans notre projet, **Hadoop** joue un rôle essentiel dans **le stockage des données météorologiques** collectées depuis NOAA. Ces données sont principalement sous forme de fichiers CSV, qui peuvent être volumineux et nécessitent un système distribué pour leur gestion et traitement.

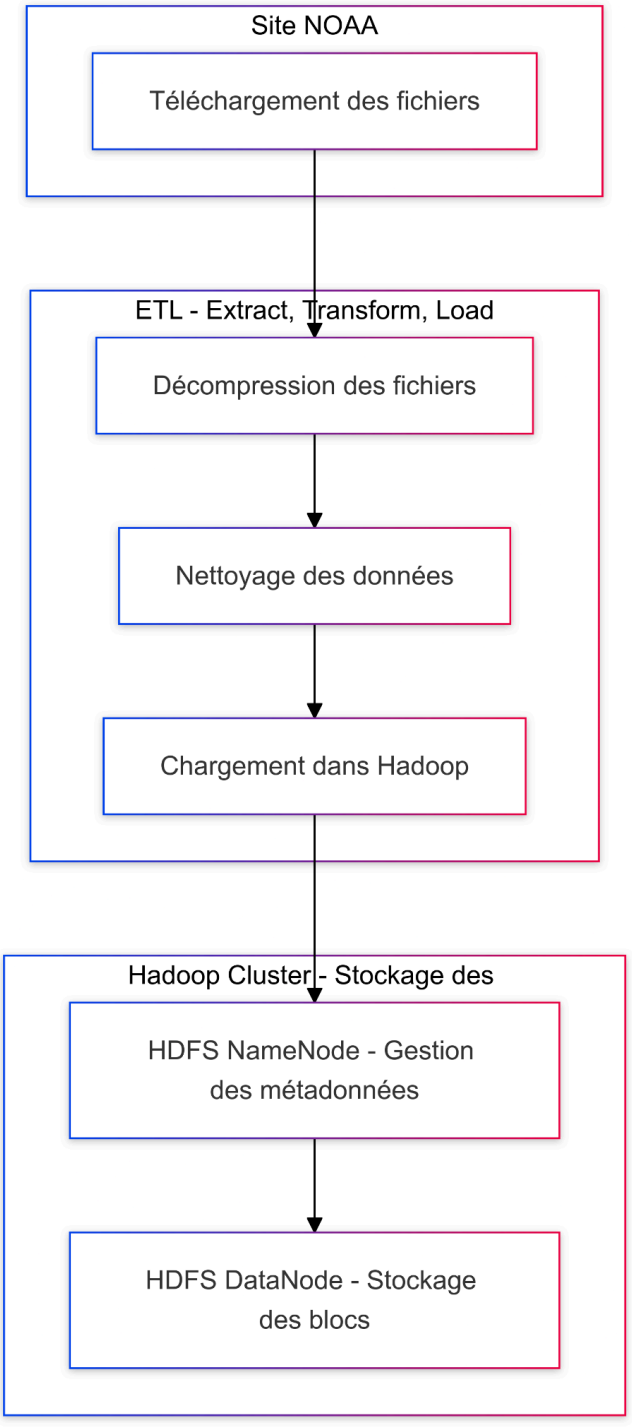
Rôle d'Hadoop dans notre projet :

1. **Stockage des fichiers CSV** :
    - Les fichiers CSV contenant les données météorologiques sont **chargés dans HDFS (Hadoop Distributed File System)**, qui permet de stocker les données de manière **répartie** sur plusieurs **DataNodes** (nœuds de stockage).

    - Chaque fichier est découpé en **blocs**, qui sont distribués sur les différents **DataNodes** pour assurer la scalabilité. Par exemple, un fichier CSV volumineux peut être coupé en plusieurs morceaux de 128 Mo, répartis sur plusieurs nœuds.
2. **Tolérance aux pannes** :
    - HDFS assure également une **réplication des blocs de données** pour garantir que les informations sont disponibles même en cas de panne d'un DataNode. Cela améliore la fiabilité du stockage.
3. **Accès rapide et optimisé** :
    - Grâce à l'architecture distribuée d'Hadoop, les fichiers sont accessibles de manière optimisée, ce qui permet aux outils comme **Spark** (qui viendra plus tard dans le processus) d'accéder rapidement aux données nécessaires pour les traiter.
4. **Gestion des métadonnées** :
    - Le **NameNode**, qui est le composant maître d'Hadoop, gère les **métadonnées** relatives à l'emplacement des blocs de données. Cela permet de suivre la structure du stockage et de coordonner l'accès aux données réparties sur les DataNodes.

Ainsi, Hadoop nous permet non seulement de **stocker** les données météorologiques à grande échelle, mais aussi d'assurer une **réplication** pour garantir la sécurité des données et un accès rapide aux fichiers pour les traitements ultérieurs.

Processus de récupération, transformation et stockage des données météorologiques:



Le schéma illustre le processus complet de récupération, transformation et stockage des données météorologiques dans le cadre du projet. Il commence par le téléchargement des fichiers depuis le site NOAA, qui représente la source des données brutes. Ensuite, ces fichiers sont décompressés et nettoyés dans l'ETL

(Extract, Transform, Load), où les données sont préparées pour être chargées dans Hadoop. Une fois les données prêtes, elles sont envoyées vers le cluster Hadoop pour un stockage optimisé dans HDFS. Le NameNode d'HDFS gère les métadonnées liées à la répartition des fichiers, tandis que les DataNodes assurent le stockage distribué des blocs de données. Ce processus permet d'assurer la scalabilité, la tolérance aux pannes et un accès rapide aux données pour les étapes de traitement suivantes.

4.3 Traitement et transformation des données avec Spark

Dans notre projet, **Spark** joue un rôle clé dans le **traitement et la transformation des données météorologiques** qui ont été stockées dans **Hadoop (HDFS)**. Après que les données aient été récupérées et stockées, Spark est utilisé pour les rendre **exploitables** et **préparer** ces données pour une utilisation future dans **Hive** et **HBase**.

Rôle de Spark dans notre projet :

1. **Lecture des fichiers bruts depuis HDFS** :
    - Spark accède directement aux fichiers stockés dans **HDFS**, en lisant les **blocs de données répartis sur les DataNodes**.
    - Cela permet à Spark de traiter des fichiers de manière distribuée et d'exécuter des calculs sur les **données brutes** stockées en parallèle sur plusieurs nœuds.
2. **Nettoyage et transformation des données** :
    - Une fois les données extraites, Spark effectue diverses **opérations de nettoyage et de transformation** pour rendre les données prêtes à être utilisées. Par exemple :
        - **Suppression des doublons** : Si certaines données sont dupliquées, Spark peut les identifier et les supprimer.
        - **Gestion des valeurs manquantes** : Spark peut remplacer les valeurs nulles par des valeurs par défaut ou les supprimer si nécessaire.
        - **Normalisation des formats** : Par exemple, Spark peut reformater les dates ou les valeurs numériques pour assurer l'uniformité des données.
    - Toutes ces transformations sont effectuées de manière **parallèle**, ce qui rend Spark très efficace pour traiter de grandes quantités de données.
3. **Stockage des résultats** :
    - Une fois les données transformées, elles sont **insérées dans les systèmes de stockage appropriés** :

    - **Hive** : Les données structurées, qui suivent un format tabulaire (comme des données météorologiques organisées par jour, station, température, etc.), sont chargées dans **Hive** sous forme de **tables SQL**. Cela permet des **requêtes SQL** sur les données.
    - **HBase** : Les données **non structurées** ou **semi-structurées** (comme des événements météorologiques qui ne suivent pas un format fixe) sont stockées dans **HBase**, permettant un accès rapide et flexible aux données pour des requêtes **NoSQL**.

Exemple concret dans notre projet :

- Les fichiers CSV contenant des **données météo** sont d'abord stockés dans **HDFS**.
- **Spark** les lit depuis **HDFS**, les nettoie (ex. formatage des dates, suppression des doublons) et effectue des transformations nécessaires.
- Les **données structurées** (comme les relevés de température par jour) sont ensuite insérées dans **Hive** pour des analyses futures via SQL.
- Les **données non structurées** (comme les descriptions détaillées des événements climatiques) sont envoyées vers **HBase** pour un accès rapide et flexible.

Ainsi, **Spark** dans notre projet sert à **nettoyer**, **transformer** et **organiser** les données météorologiques pour qu'elles soient facilement exploitables, en les stockant soit dans **Hive** pour des analyses SQL, soit dans **HBase** pour un traitement rapide et flexible des données non structurées.

Exemple de code spark :

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_date

# Initialisation de la session Spark
spark = SparkSession.builder \
    .appName("MeteoDataProcessing") \
    .getOrCreate()

# Chargement des fichiers CSV depuis HDFS
df = spark.read.csv("hdfs://data/*.csv", header=True, inferSchema=True)

# Suppression des lignes avec valeurs manquantes
df_cleaned = df.dropna()

# Formatage des dates
df_cleaned = df_cleaned.withColumn("DATE", to_date(col("DATE"), "yyyy-MM-dd"))

# Suppression des doublons
df_cleaned = df_cleaned.dropDuplicates()

# Conversion de la température de Fahrenheit en Celsius
df_cleaned = df_cleaned.withColumn("TEMP_C", (col("TEMP") - 32) * 5 / 9)

# Enregistrement des résultats dans Hive
df_aggregated.write.mode("overwrite").saveAsTable("meteo_aggregated_data")

# Arrêt la session Spark pour libérer les ressources
spark.stop()
```

Ce script fait plusieurs opérations sur les données météorologiques stockées dans des fichiers CSV. Voici ce qu'il fait étape par étape :

1. **Chargement des données** : Le script commence par charger les fichiers CSV depuis HDFS dans un format que Spark peut manipuler.
2. **Suppression des valeurs manquantes** : Les lignes qui contiennent des données manquantes (par exemple, des températures absentes) sont supprimées pour s'assurer qu'il n'y a pas de problèmes lors du traitement des données.
3. **Formatage des dates** : Le script transforme la colonne "DATE" pour s'assurer que toutes les dates sont au bon format (yyyy-MM-dd).
4. **Suppression des doublons** : Les lignes qui sont identiques sont supprimées pour ne garder qu'une seule entrée par donnée unique.
5. **Conversion des températures** : Une nouvelle colonne est ajoutée pour convertir les températures de Fahrenheit en Celsius, car les données sont initialement en Fahrenheit.

6. **Sauvegarde des résultats** : Enfin, les données nettoyées et transformées sont enregistrées dans une table Hive, prête à être utilisée pour des analyses futures.
7. **Arrêt de Spark** : Après avoir terminé toutes ces opérations, la session Spark est fermée pour libérer les ressources utilisées.

En résumé, ce script nettoie et prépare les données météorologiques pour qu'elles soient prêtes à être utilisées dans des analyses futures.

Processus de Traitement et Stockage des Données Météorologiques avec Spark :

Le schéma illustre le rôle central de Spark dans le pipeline de traitement des données météorologiques. Spark accède directement aux données brutes stockées dans HDFS, exécute des étapes de nettoyage et de transformation, telles que la gestion des doublons ou le formatage des données, puis distribue les données transformées vers deux systèmes de stockage :

- **Hive**, pour les données structurées nécessitant des analyses SQL.
- **HBase**, pour les données non structurées, accessibles via des requêtes NoSQL.

Ce processus garantit un traitement rapide et distribué des données volumineuses, en rendant les informations prêtes pour des analyses ou des requêtes à partir de systèmes dédiés.

4.4 Recherche et indexation avec ElasticSearch

Dans notre projet, **ElasticSearch** joue un rôle clé pour rendre les données accessibles et exploitables grâce à des capacités de recherche avancées et performantes. Voici comment il intervient :

Indexation des données

ElasticSearch crée des index à partir des données stockées dans **Hive** (données structurées) et **HBase** (données semi-structurées ou non structurées). Ces index permettent une recherche rapide et efficace sur de larges volumes de données.

- **Étape 1 : Extraction des données**
  Les données pertinentes sont récupérées de Hive et HBase grâce à des connecteurs ou via des scripts spécifiques.
- **Étape 2 : Création des index**
  Les données récupérées sont transformées en documents compatibles avec le format JSON attendu par ElasticSearch. Ces documents sont ensuite indexés pour structurer les informations de manière optimisée pour la recherche.

Recherche avancée

Une fois les données indexées, ElasticSearch permet d'exécuter des recherches complexes, comme :

- **Recherches géospatiales** : Trouver les températures moyennes ou les précipitations maximales pour une région donnée.
- **Recherches temporelles** : Filtrer les données par plages de dates pour analyser les tendances météorologiques.
- **Filtres combinés** : Permettre des recherches croisées, par exemple : *"Quelles sont les températures enregistrées en janvier dans une altitude supérieure à 1000 mètres ?"*

ElasticSearch répond rapidement en récupérant les données pré-indexées, évitant de parcourir directement Hive ou HBase, ce qui serait beaucoup plus lent.

Exemple d'utilisation dans notre projet

Un utilisateur via l'interface React peut demander :
"Quelle a été la température moyenne enregistrée en France le 15 janvier 2023 ?".

- La requête est transmise par **FastAPI** à ElasticSearch.
- ElasticSearch utilise son index pour trouver la réponse en interrogeant les données pertinentes, sans devoir effectuer des calculs complexes en temps réel.

Grâce à cette approche, ElasticSearch assure des performances élevées et une expérience utilisateur fluide, même avec de très grandes quantités de données météorologiques.

Exemple de code avec ElasticSearch :

```python
from pyhive import hive
import happybase
from elasticsearch import Elasticsearch
from datetime import datetime

# Connexions aux services
es = Elasticsearch(hosts=["http://localhost:9200"])
hive_conn = hive.Connection(host="hive-server", port=10000, database="meteo_data")
hbase_conn = happybase.Connection(host="hbase-master")

# Extraction des données depuis Hive
def get_data_from_hive():
    hive_cursor = hive_conn.cursor()
    hive_cursor.execute("SELECT station, region, date, temp_c FROM weather_data")
    return [
        {"station": row[0], "region": row[1], "date": row[2], "temp_celsius": row[3]}
        for row in hive_cursor.fetchall()
    ]

# Extraction des données depuis HBase
def get_data_from_hbase():
    hbase_table = hbase_conn.table("weather_data")
    return [
        {
            "station": row[b"station"].decode(),
            "region": row[b"region"].decode(),
            "date": row[b"date"].decode(),
            "temp_celsius": float(row[b"temp_celsius"])
        }
        for _, row in hbase_table.scan()
    ]

# Fusion des données extraites
def combine_data():
    hive_data = get_data_from_hive()
    hbase_data = get_data_from_hbase()
    return hive_data + hbase_data
```

```python
# Indexation des données dans ElasticSearch
def index_data(data):
    for record in data:
        es.index(index="weather_data", document=record)

# Calcul de la température moyenne pour une station dans un intervalle de dates
def calculate_avg_temp(station, start_date, end_date):
    # Formatage des dates
    start_date = datetime.strptime(start_date, "%Y-%m-%d").date()
    end_date = datetime.strptime(end_date, "%Y-%m-%d").date()

    query = {
        "query": {
            "bool": {
                "must": [
                    {"match": {"station": station}},
                    {"range": {"date": {"gte": start_date, "lte": end_date}}}
                ]
            }
        },
        "aggs": {
            "avg_temp": {
                "avg": {"field": "temp_celsius"}
            }
        }
    }

    response = es.search(index="weather_data", body=query)
    avg_temp = response['aggregations']['avg_temp']['value']
    return avg_temp

# Exemple d'utilisation :
data = combine_data()
index_data(data)
avg_temp = calculate_avg_temp("BARDUFOSS, NO", "1997-01-01", "1997-01-31")
print(f"La température moyenne pour la station BARDUFOSS, NO entre 1997-01-01 et 1997-01-31 est : {avg_temp} °C")
```

1. **Connexion aux services** :
    - **ElasticSearch** : On se connecte à ElasticSearch via l'URL. Ici "http://localhost:9200", est utilisé à titre d'exemple.
    - **Hive** : La connexion à Hive se fait via la bibliothèque `pyhive` en spécifiant l'adresse du serveur Hive (`hive-server`) et la base de données cible (`meteo_data`).
    - **HBase** : Pour HBase, on utilise la bibliothèque `happybase`, qui est une interface pour interagir avec HBase. On se connecte au serveur HBase via `hbase-master`.
2. **Extraction des données depuis Hive** (`get_data_from_hive`) :
    - Cette fonction exécute une requête SQL sur Hive pour récupérer des données dans la table `weather_data`, spécifiquement les colonnes `station`, `region`, `date`, et `temp_c` (température en degrés Celsius).
    - Une fois la requête exécutée, elle transforme les résultats (qui sont des lignes de données) en une liste de dictionnaires. Chaque dictionnaire représente un enregistrement avec la station, la région, la date et la température.
3. **Extraction des données depuis HBase** (`get_data_from_hbase`) :
    - Cette fonction scanne la table `weather_data` dans HBase, et pour chaque ligne, elle récupère les colonnes `station`, `region`, `date`, et `temp_celsius`.
    - Les données sont également transformées en dictionnaires, où chaque clé est le nom de la colonne et chaque valeur est le contenu de la ligne. On décode les valeurs binaires de HBase (par exemple, les champs `station`, `region`, `date`) et on les convertit dans des formats appropriés (comme des nombres pour la température).
4. **Fusion des données** (`combine_data`) :
    - Cette fonction combine les données extraites de Hive et de HBase en une seule liste. Les données de Hive et HBase sont concaténées pour que toutes les informations se retrouvent dans un seul ensemble de données.
5. **Indexation des données dans ElasticSearch** (`index_data`) :
    - Cette fonction prend chaque enregistrement (dictionnaire) dans la liste des données combinées et les envoie à ElasticSearch pour être indexés dans un index nommé `weather_data`.
    - Indexer les données dans ElasticSearch permet de les rendre facilement accessibles et optimisées pour des recherches et des analyses ultérieures.
6. **Calcul de la température moyenne pour une station dans un intervalle de dates** (`calculate_avg_temp`) :

    - Cette fonction permet de calculer la température moyenne pour une station donnée, dans une plage de dates spécifique.
    - Elle prend en entrée une station (nom de la station) et deux dates (start_date et end_date).
    - Les dates sont converties en format `date` pour être comparées avec les dates stockées dans ElasticSearch.
    - La fonction envoie ensuite une requête à ElasticSearch. Cette requête filtre les enregistrements pour correspondre à la station et à l'intervalle de dates donnés, puis calcule la température moyenne sur cette période avec une agrégation (`avg`).
    - Enfin, la température moyenne est extraite de la réponse et retournée.
7. **Exécution** :
    - On appelle d'abord la fonction `combine_data()` pour rassembler les données de Hive et HBase.
    - Ensuite, ces données sont indexées dans ElasticSearch via la fonction `index_data()`.
    - Enfin, on utilise la fonction `calculate_avg_temp()` pour calculer la température moyenne pour la station "BARDUFOSS, NO" entre le 1er janvier 1997 et le 31 janvier 1997, et on affiche le résultat.

En résumé, ce code récupère des données depuis Hive et HBase, les indexe dans ElasticSearch, puis permet de calculer des moyennes de température pour des stations sur des périodes spécifiques, tout en utilisant ElasticSearch pour effectuer les requêtes et les calculs de manière rapide et efficace.

4.5 Interface backend avec FastAPI

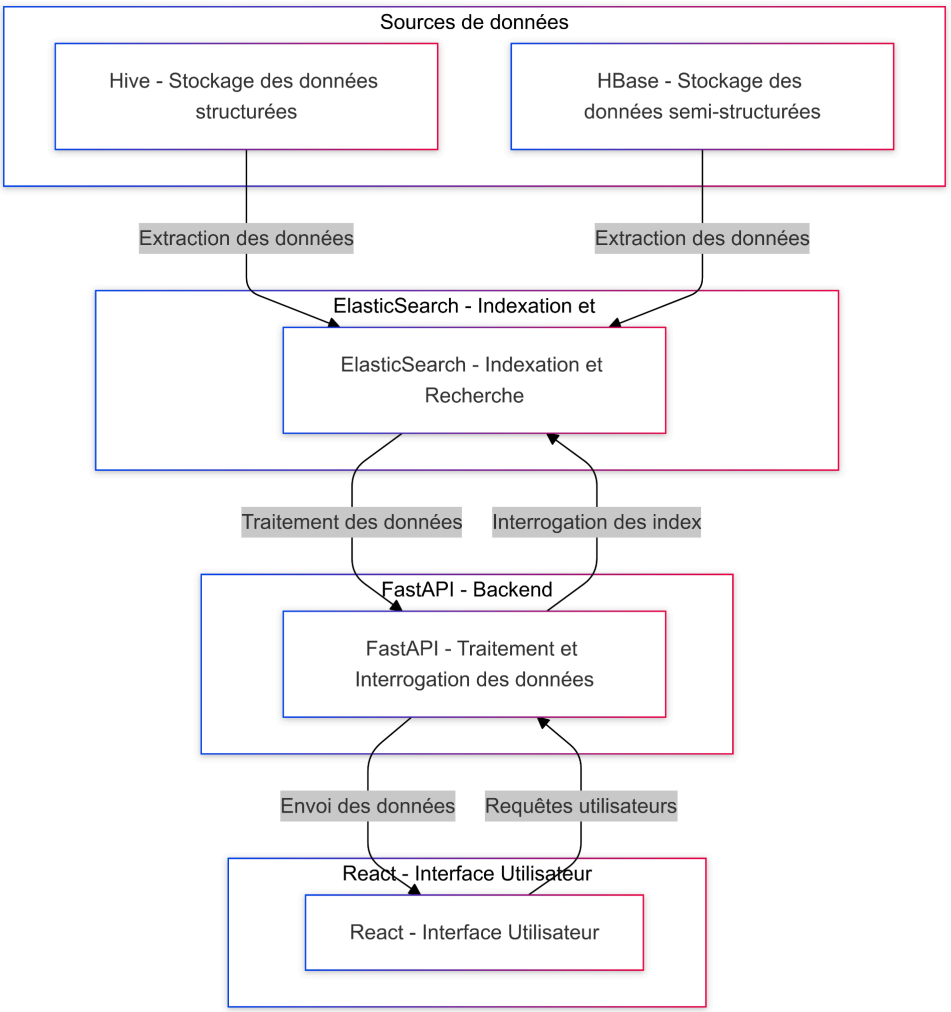
FastAPI agit comme une passerelle sécurisée entre le front-end et ElasticSearch :

- Les utilisateurs envoient des requêtes via l’application React.js.
- FastAPI traduit ces requêtes pour interagir avec ElasticSearch.
- Les résultats sont formatés pour être affichés de manière conviviale dans l’interface utilisateur.

4.6 Application front-end avec React.js

- **Fonctions principales** :
    - Visualisation des données sous forme de graphiques, tableaux et cartes interactifs.
    - Recherche et filtrage dynamique via des formulaires intuitifs.
    - Interaction fluide grâce à des composants réactifs.

Flux des données entre les systèmes de stockage, ElasticSearch, FastAPI et l'interface utilisateur



Ce schéma illustre le flux de données entre les différentes composantes de notre architecture, mettant en évidence les interactions clés dans le processus de recherche et de restitution des données.

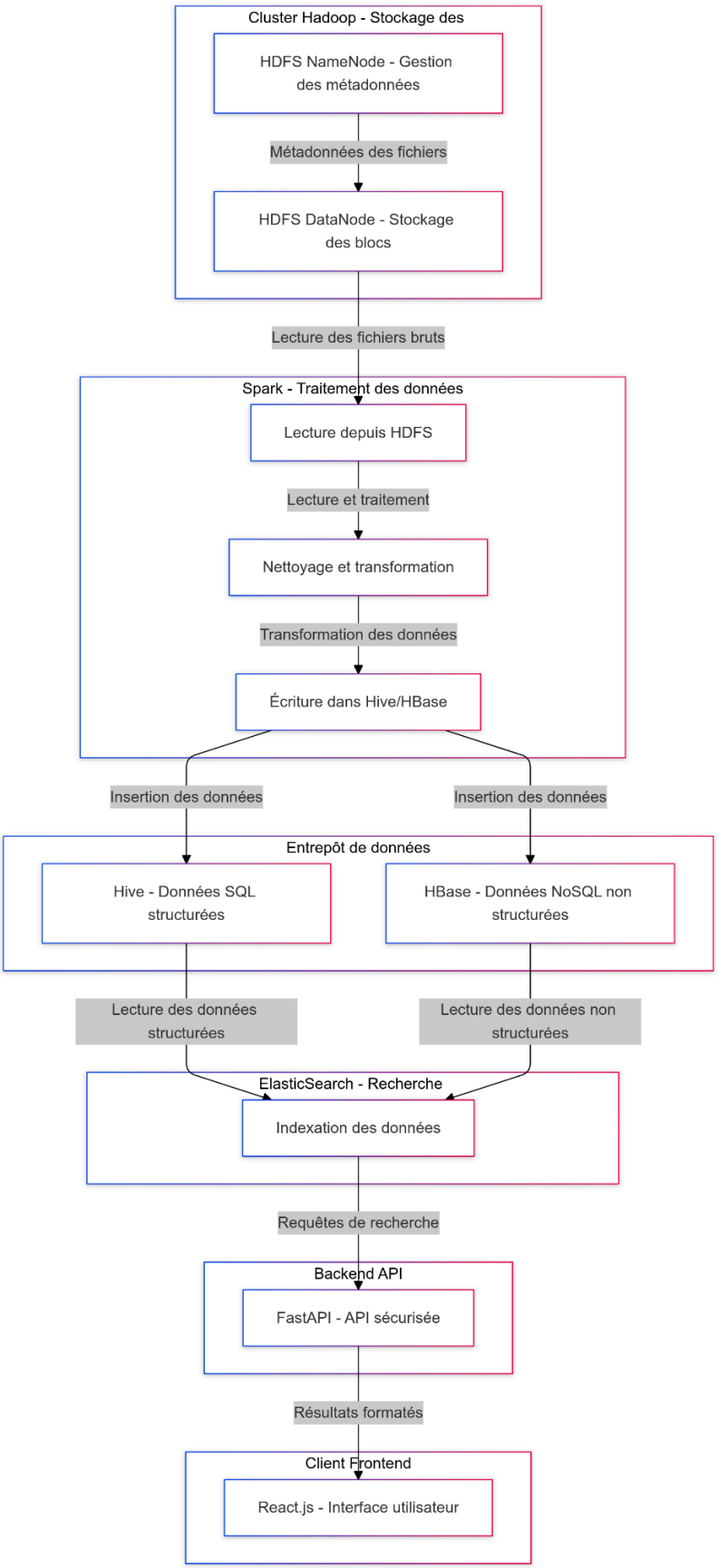
1. **Le client (React) initie la requête** : L'utilisateur, via l'interface React, effectue une recherche en formulant une requête, par exemple, en demandant des informations sur les températures dans une région à une date précise. Cette requête est envoyée vers le backend FastAPI.
2. **FastAPI traite la requête** : FastAPI reçoit la requête du client et joue le rôle de passerelle entre le front-end et les systèmes de stockage de données. Il traduit la demande en une requête compréhensible pour ElasticSearch. FastAPI interroge ensuite les index d'ElasticSearch pour récupérer les informations pertinentes.
3. **ElasticSearch récupère les données** : ElasticSearch, qui a indexé les données extraites de Hive et HBase, est chargé de répondre à la requête. En utilisant ses capacités de recherche avancées, il interroge ses index pour retrouver les données demandées de manière rapide et efficace.
4. **FastAPI renvoie les résultats au client** : Une fois qu'ElasticSearch a renvoyé les résultats, FastAPI les récupère, les formate si nécessaire (par exemple, en ajustant les formats de données) et les envoie au client React sous une forme appropriée pour l'affichage.
5. **Le client affiche les résultats** : Enfin, React prend les données renvoyées par FastAPI et les présente de manière conviviale dans l'interface utilisateur, permettant à l'utilisateur de visualiser les résultats de sa recherche de manière claire et interactive.

Ce flux décrit comment les données circulent dans le système, depuis la demande de l'utilisateur jusqu'à la présentation des résultats, en passant par l'indexation des données et la recherche dans ElasticSearch.

4.7 Coordination et containerisation avec Docker

- Tous les composants (Hadoop, Spark, Hive, HBase, ElasticSearch et FastAPI) sont déployés dans des conteneurs Docker.
- **Docker Compose** gère l’orchestration des conteneurs pour :
    - Assurer une communication fluide entre les nœuds du cluster Hadoop, Spark et les autres services.
    - Permettre un déploiement rapide et modulable de l’application complète.

Schéma présentant le flux de données du projet complet



5. Sécurité

5.1 Kerberos

Kerberos est un protocole d'authentification réseau utilisé pour sécuriser les communications dans un réseau informatique. Son objectif principal est de garantir que les parties communicantes, comme un utilisateur et un service, puissent se vérifier mutuellement de manière sécurisée et fiable. Il fonctionne selon un système de tickets pour éviter la transmission de mots de passe sur le réseau et pour assurer que seules les parties autorisées puissent accéder aux ressources protégées.

Principes de fonctionnement de Kerberos :

1. **Authentification centralisée** : Kerberos repose sur un **serveur centralisé**, appelé **KDC (Key Distribution Center)**, qui gère les tickets d'authentification. Lorsqu'un utilisateur ou un service veut se connecter à un autre service, il doit obtenir un ticket du KDC.
2. **Tickets** : Le KDC délivre un ticket pour chaque session d'authentification. Il existe deux types principaux :
    - **TGT (Ticket-Granting Ticket)** : Un ticket initial permettant d'obtenir d'autres tickets pour accéder à des services spécifiques.
    - **Service Ticket** : Un ticket pour accéder à un service spécifique (par exemple, une base de données ou un cluster Hadoop).
3. **Clé secrète partagée** : Le serveur et le client partagent une clé secrète, et tout l'échange de tickets est chiffré avec cette clé, garantissant la confidentialité des communications.
4. **Durée limitée** : Les tickets délivrés par Kerberos ont une durée de vie limitée, ce qui renforce la sécurité en limitant le temps pendant lequel un ticket peut être utilisé.

Application de Kerberos dans notre projet :

Dans notre projet, Kerberos est utilisé pour sécuriser l'authentification et l'autorisation dans l'écosystème Hadoop et ElasticSearch :

1. **Authentification sur Hadoop (HDFS, HBase, Hive)** :
    - Kerberos garantit que seules les entités autorisées (utilisateurs ou services) peuvent interagir avec les systèmes de stockage de données (HDFS, HBase, Hive). Cela peut servir à protéger les données sensibles.
2. **Sécurisation de l'accès aux services ElasticSearch et FastAPI** :
    - Pour interagir avec ElasticSearch (que ce soit pour l'indexation ou la recherche de données), Kerberos peut être utilisé pour authentifier les

requêtes de **FastAPI** vers **ElasticSearch**. Nous pouvons configurer une connexion sécurisée entre ces services pour éviter l'accès non autorisé et assurer que seules les requêtes légitimes puissent interroger ou indexer les données.

3. **Accès à ElasticSearch à partir du back-end (FastAPI)** :
    - L'authentification de **FastAPI** auprès de **ElasticSearch** peut être gérée par Kerberos. Cela permet d'assurer que les données météorologiques stockées et indexées dans ElasticSearch ne sont accessibles que par les utilisateurs autorisés, réduisant ainsi le risque de fuite ou de modification de données sensibles.

En résumé, **Kerberos** joue un rôle clé dans la sécurité de notre projet en protégeant les échanges de données entre les différents composants de notre infrastructure (Hadoop, HBase, Hive, ElasticSearch). Son application dans notre projet garantit une authentification forte, une gestion fine des accès, et une communication sécurisée, contribuant ainsi à la protection des données sensibles tout au long du processus de collecte, de traitement et de recherche des données météorologiques.

5.2 JWT Token

JWT est un standard ouvert qui définit un format compact et sécurisé pour la transmission de données entre deux parties sous forme d'un objet JSON. Ce format est largement utilisé pour gérer l'authentification et l'autorisation dans les applications web modernes. Un **JWT** contient trois parties principales :

1. **Header** (En-tête) : Contient des informations sur la manière dont le JWT est signé, typiquement le type de token (JWT) et l'algorithme de signature (par exemple, HMAC SHA256 ou RSA).
2. **Payload** (Charge utile) : Contient les informations ou les **claims** (revendications) sous forme de paires clé-valeur. Ces informations peuvent inclure des détails sur l'utilisateur (par exemple, ID utilisateur, rôle, permissions), la date d'expiration du token, etc.
3. **Signature** : Une signature cryptographique qui permet de vérifier l'intégrité du token et de s'assurer qu'il n'a pas été altéré. Elle est générée en signant le header et le payload avec une clé secrète.

Le token JWT est couramment utilisé dans les systèmes d'authentification basés sur des tokens, où l'utilisateur s'authentifie une fois pour obtenir un token, puis ce token est utilisé pour effectuer des requêtes authentifiées sans avoir à soumettre des identifiants à chaque requête.

Application de JWT dans notre projet :

Dans notre projet de gestion de données météorologiques, **JWT** est utilisé principalement pour la sécurisation des interactions entre **FastAPI** et les utilisateurs de votre application :

1. **Connexion et génération de JWT** :
    - L'utilisateur soumet son nom d'utilisateur et son mot de passe via un formulaire dans la page de connexion de l'interface client.
    - FastAPI authentifie l'utilisateur (en vérifiant le mot de passe) et génère un **JWT** qui contient les informations pertinentes de l'utilisateur (username, e-mail, ID etc).
    - Le JWT est renvoyé à l'utilisateur, qui le conserve dans le stockage local du navigateur.
2. **Utilisation du JWT pour sécuriser les requêtes** :
    - Lors de chaque requête suivante (par exemple, pour rechercher des données météorologiques), l'utilisateur envoie le **JWT** dans l'en-tête `Authorization` de la requête HTTP.
    - FastAPI extrait ce token, vérifie sa validité et, si valide, autorise l'accès aux ressources demandées (par exemple, accéder aux données stockées dans ElasticSearch).

Le **JWT** dans notre projet permet donc de gérer l'authentification et l'autorisation des utilisateurs de manière sécurisée et sans état. Grâce à ce mécanisme, on garantit que seules les personnes autorisées peuvent interagir avec notre application, et ainsi faciliter la gestion des sessions et des permissions, tout en assurant une communication fluide et sécurisée entre le **frontend React.js** et le **backend FastAPI**.

Conclusion

Dans ce projet, nous avons construit une architecture complète de traitement et d'analyse de données météorologiques, en utilisant une combinaison de technologies modernes adaptées à des besoins de scalabilité, de performance, et de sécurité.

En exploitant **Hadoop** et ses écosystèmes associés tels que **Hive** et **HBase**, nous avons pu gérer efficacement de grandes quantités de données structurées et non structurées. **Spark** a permis de traiter et transformer ces données à grande échelle, facilitant ainsi leur préparation pour l'analyse avancée.

L'intégration d'**ElasticSearch** a offert des capacités de recherche et d'indexation performantes, permettant aux utilisateurs de réaliser des requêtes complexes sur les données météorologiques de manière rapide et précise. **FastAPI**, de son côté, a agi comme un point d'accès sécurisé et rapide pour interagir avec ces données, offrant une interface backend performante et facile à maintenir.

Le frontend, développé en **React.js**, a permis de créer une interface interactive et dynamique, rendant la visualisation et l'interaction avec les données aussi intuitives que possible. L'utilisation de **Docker** pour la conteneurisation de l'ensemble de l'infrastructure a assuré la portabilité, la scalabilité et une gestion simplifiée des différents services dans des environnements de développement, de test et de production.

En matière de sécurité, l'implémentation de **Kerberos** a renforcé la protection des données sensibles contre les accès non autorisés, tandis que l'utilisation de **JWT** a permis de sécuriser les échanges entre le frontend et le backend, garantissant ainsi l'intégrité et la confidentialité des sessions utilisateur.

En conclusion, ce projet démontre comment une architecture basée sur des technologies big data et des outils modernes peut être mise en place pour traiter efficacement des volumes massifs de données et fournir des résultats rapides et accessibles, tout en assurant un haut niveau de sécurité et d'intégrité. Les différentes technologies utilisées, telles que **Hadoop**, **ElasticSearch**, **FastAPI** et **React.js**, ont été intégrées de manière cohérente pour répondre aux besoins de l'application et offrir une expérience utilisateur fluide et sécurisée. Ce projet constitue ainsi une solution robuste et scalable pour l'analyse et la visualisation des données météorologiques.