

## Devoir 3 : BigData

*Mettre en place une architecture BIGDATA*

Réalisé par : Houssam ELBERROUHI / grp1

### Architecture BIGDATA pour la Prédiction des Pandémies



## **Table des matières**

- 1. Introduction*
- 2. Architecture globale*
- 3. Collecte et stockage des données*
  - *Conversion des données CSV en Parquet*
  - *Organisation des données dans HDFS*
- 4. Traitement des données et modélisation*
  - *Préparation des données*
  - *Modèle de classification*
- 5. Pipeline de traitement en temps réel*
  - *Producteur Kafka*
  - *Consommateur Kafka*
- 6. Stockage des prédictions*
- 7. Visualisation des résultats*
  - *Dashboard Streamlit*
  - *Monitoring avec Grafana*
- 8. Conclusion*

## 1. Introduction

Ce projet vise à mettre en place une architecture Big Data complète pour prédire l'évolution des pandémies, en utilisant comme exemple la pandémie de COVID-19. L'architecture repose sur plusieurs technologies clés de l'écosystème Big Data :

- **Hadoop** pour le stockage distribué des données
- **Spark** pour le traitement batch et l'entraînement de modèles
- **Kafka** pour la gestion des flux de données en temps réel
- **Spark Streaming** pour le traitement en continu
- **PostgreSQL** pour la persistance des prédictions
- **Streamlit et Grafana** pour la visualisation des résultats

Le projet utilise un jeu de données contenant des informations sur les cas de COVID-19 aux États-Unis en 2023, avec des détails sur les cas et les décès par comté et par état.

date	county	state	cases	deaths
1/1/2023	Autauga	Alabama	18961	230
1/1/2023	Baldwin	Alabama	67496	719
1/1/2023	Barbour	Alabama	7027	111
1/1/2023	Bibb	Alabama	7692	108
1/1/2023	Blount	Alabama	17731	260

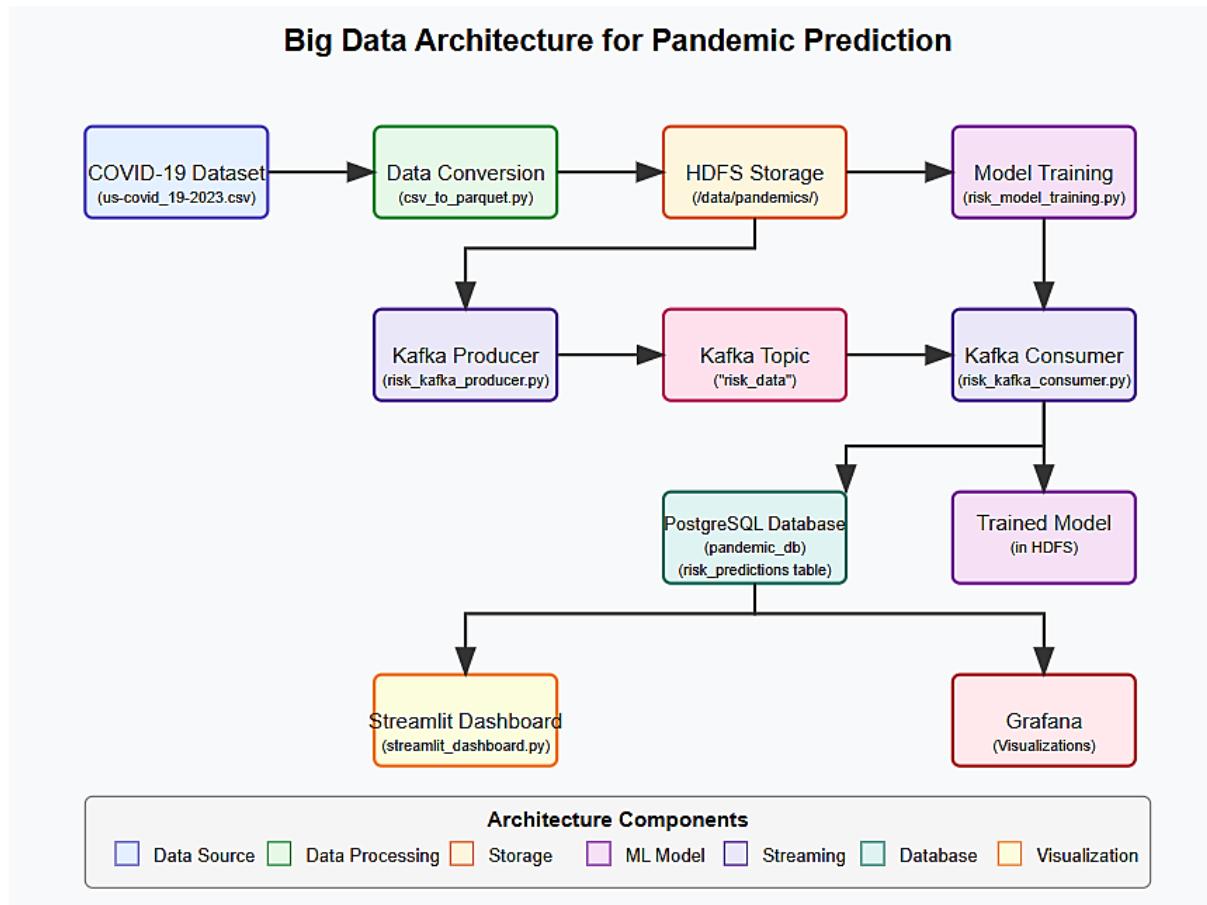
## 2. Architecture globale

Mon architecture suit un pipeline de données complet, du stockage initial jusqu'à la visualisation :

1. **Ingestion des données** : Import initial des données COVID-19 au format CSV
2. **Transformation et stockage** : Conversion et partitionnement des données au format Parquet dans HDFS
3. **Modélisation prédictive** : Entraînement d'un modèle de classification Random Forest pour prédire les catégories de risque
4. **Pipeline en temps réel** : Publication des données via Kafka et traitement en continu avec Spark Streaming

5. **Persistance** : Stockage des prédictions dans une base de données PostgreSQL
6. **Visualisation** : Création de tableaux de bord interactifs avec Streamlit et Grafana

Cette architecture permet de traiter aussi bien les données historiques que les flux en temps réel, offrant une vision complète de l'évolution de la pandémie et des prédictions de risque.



### 3. Collecte et stockage des données

#### Conversion des données CSV en Parquet

```
us-covid-19-2023.csv
zookeeper
root@hadoop-master:~# docker cp us-covid-19-2023.csv hadoop-master:/tmp/us-covid-19-2023.csv
Successfully copied 9.87MB to hadoop-master:/tmp/us-covid-19-2023.csv
C:\Users\USER\Documents\ESTM\S4\BIG DATA\TPs\Devoir 3 -ds levels>
```

Le point de départ est un fichier CSV **us-covid\_19-2023.csv** contenant les données de cas COVID-19 par comté aux États-Unis. Le script *csv\_to\_parquet.py* transforme ces données au format Parquet, plus efficace pour le traitement distribué.

```
1 # Écriture en Parquet
2 logger.info("Écriture des données en format Parquet...")
3 (df.write
4 .mode("overwrite")
5 .partitionBy("annee", "mois", "jour")
6 .parquet(output_path))

def main():
    """Fonction principale"""
    input_path = "hdfs://hadoop-master:9000/data/pandemics/us-covid_19-2023.csv"
    output_path = "hdfs://hadoop-master:9000/data/pandemics"

    spark = None
    try:
        spark = create_spark_session()
        process_data(spark, input_path, output_path)
    except Exception as e:
        logger.error(f"Erreur critique: {str(e)}")
        sys.exit(1)
    finally:
        if spark:
            spark.stop()
            logger.info("Session Spark fermée")

    if __name__ == "__main__":
        main()
```

```
root@hadoop-master:~# python3 csv_to_parquet.py
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
2025-03-30 15:18:15,901 - INFO - Lecture du fichier CSV...
2025-03-30 15:18:21,719 - INFO - Analyse des colonnes nulles...
2025-03-30 15:18:31,755 - INFO - Ajout des colonnes de partitionnement...
2025-03-30 15:18:32,173 - INFO - Écriture des données en format Parquet...
2025-03-30 15:18:49,799 - INFO - Données stockées avec succès dans hdfs://hadoop-master:9000/data/pandemics
```

Points clés de cette transformation :

- Définition explicite du schéma pour optimiser les performances
- Conversion des dates au format approprié
- Ajout de colonnes de partitionnement (année, mois, jour)
- Optimisation de la configuration Spark pour les ressources disponibles

## Organisation des données dans HDFS

Les données sont stockées dans HDFS selon une structure hiérarchique partitionnée :

/data/pandemics/

```
|   └── annee=2023/
|       |   └── mois=01/
|       |       |   └── jour=01/
|       |       |       └── part-0000.snappy.parquet
|       |       |       └── part-0001.snappy.parquet
|       |       └── jour=02/
|       ...
```

root@hadoop-master:~# hdfs dfs -ls /data/pandemics/
Found 2 items
-rw-r--r-- 2 root supergroup 0 2025-03-30 15:18 /data/pandemics/\_SUCCESS
drwxr-xr-x - root supergroup 0 2025-03-30 15:18 /data/pandemics/annee=2023
root@hadoop-master:~# hdfs dfs -ls /data/pandemics/annee=2023/
Found 3 items
drwxr-xr-x - root supergroup 0 2025-03-30 15:18 /data/pandemics/annee=2023/mois=01
drwxr-xr-x - root supergroup 0 2025-03-30 15:18 /data/pandemics/annee=2023/mois=02
drwxr-xr-x - root supergroup 0 2025-03-30 15:18 /data/pandemics/annee=2023/mois=03
root@hadoop-master:~# hdfs dfs -ls /data/pandemics/annee=2023/mois=01
Found 31 items
drwxr-xr-x - root supergroup 0 2025-03-30 15:18 /data/pandemics/annee=2023/mois=01/jour=01
drwxr-xr-x - root supergroup 0 2025-03-30 15:18 /data/pandemics/annee=2023/mois=01/jour=02
drwxr-xr-x - root supergroup 0 2025-03-30 15:18 /data/pandemics/annee=2023/mois=01/jour=03
drwxr-xr-x - root supergroup 0 2025-03-30 15:18 /data/pandemics/annee=2023/mois=01/jour=04
drwxr-xr-x - root supergroup 0 2025-03-30 15:18 /data/pandemics/annee=2023/mois=01/jour=05

Cette structure facilite grandement les analyses temporelles et permet un accès efficace aux données.

## 4. Traitement des données et modélisation

### Préparation des données

Pour prédire les zones à risque élevé, j'ai développé un modèle de classification utilisant RandomForest. Le script **risk\_model\_training.py** est responsable de cette tâche :

1. Chargement des données Parquet depuis HDFS
2. Calcul d'un score de risque basé sur le nombre de cas et de décès
3. Catégorisation du risque en 5 niveaux (de très faible à très élevé)
4. Assemblage des caractéristiques (features) pour l'entraînement
5. Entraînement du modèle RandomForest
6. Sauvegarde du modèle dans HDFS

```
# Load dataset from HDFS
DATA_PATH = "hdfs://hadoop-master:9000/data/pandemics/annee=2023"
df = spark.read.parquet(DATA_PATH).select("state", "county", "date", "cases", "deaths", "mois", "jour")
df = df.dropna()

# a) Calculate risk score
df = df.withColumn("risk_score", df["cases"] + df["deaths"] * 10)

# b) Define risk categories using Bucketizer
bucket_splits = [-float("inf"), 0, 1000, 10000, 100000, float("inf")]
bucketizer = Bucketizer(splits=bucket_splits, inputCol="risk_score", outputCol="risk_category")
df = bucketizer.transform(df)
```

### Modèle de classification

J'ai utilisé l'algorithme Random Forest pour la classification des niveaux de risque :

```
# c) Assemble features
assembler = VectorAssembler(inputCols=["mois", "jour", "risk_score"], outputCol="features", handleInvalid="skip")
data = assembler.transform(df).dropna()

# Step 2: Train-Test Split
train_data, test_data = data.randomSplit([0.8, 0.2], seed=42)

# Step 3: Train RandomForest Classifier (Predict risk_category)
rf = RandomForestClassifier(featuresCol="features", labelCol="risk_category", numTrees=20, maxDepth=10)
model = rf.fit(train_data)

# Step 4: Save the model (Use the same path as the consumer)
MODEL_PATH = "hdfs://hadoop-master:9000/models/GeoRisk_model"
model.write().overwrite().save(MODEL_PATH)
```

Caractéristiques du modèle :

- **Features utilisées** : nombre de décès, mois et jour
- **Variable cible** : catégorie de risque basée sur le nombre de cas
- **Algorithme** : Random Forest avec 20 arbres et une profondeur maximale de 10
- **Évaluation** : précision de 96% sur l'ensemble de test
- **Persistante** : sauvegarde du modèle dans HDFS pour utilisation ultérieure

```
45f9f1-6f39-4404-8de7-8fe856ee5eac.
High-risk zone prediction model saved successfully at: hdfs://hadoop-master:9000/mo
models/GeoRisk_model
25/04/02 00:20:15 INFO SparkContext: Sp
```

```
45f9f1-6f39-4404-8de7-8fe856ee5eac.
High-risk zone prediction model saved successfully at: hdfs://hadoop-master:9000/mo
models/GeoRisk_model
25/04/04 00:37:39 INFO SparkContext: SparkContext is stopping with exitCode 0.
```

Path de stockage de modèle de risque entraîné sur hdfs :

```
root@hadoop-master:~# hdfs dfs -ls /models
Found 2 items
drwxr-xr-x  - root supergroup          0 2025-04-04 00:37 /models/GeoRisk_model
drwxr-xr-x  - root supergroup          0 2025-04-03 20:42 /models/pandemic_model
root@hadoop-master:~# hdfs dfs -ls /models/GeoRisk_model
Found 3 items
drwxr-xr-x  - root supergroup          0 2025-04-04 00:37 /models/GeoRisk_model/da
ta
drwxr-xr-x  - root supergroup          0 2025-04-04 00:37 /models/GeoRisk_model/me
tadata
drwxr-xr-x  - root supergroup          0 2025-04-04 00:37 /models/GeoRisk_model/tr
eesMetadata
root@hadoop-master:~# hdfs dfs -ls /models/GeoRisk_model/data
Found 2 items
-rw-r--r--  2 root supergroup          0 2025-04-04 00:37 /models/GeoRisk_model/da
ta/_SUCCESS
-rw-r--r--  2 root supergroup  20266 2025-04-04 00:37 /models/GeoRisk_model/da
ta/part-00000-27bd4162-6f8a-4694-a39b-f71fa00eae86-c000.snappy.parquet
```

## 5. Pipeline de traitement en temps réel

Pour mettre en place un système de prédition en temps réel, j'ai développé :

### Producteur Kafka (`risk_kafka_producer.py`)

Le script `risk_kafka_producer.py` lit les données Parquet et les envoie au topic Kafka "risk\_data" avec un délai d'une seconde pour simuler des données en temps réel :

```
# Read dataset from HDFS in partitions
DATA_PATH = "hdfs://hadoop-master:9000/data/pandemics/annee=2023"
df = spark.read.parquet(DATA_PATH).select("state", "county", "date", "cases", "deaths", "mois", "jour").repartition(2)

# Initialize Kafka Producer
producer = KafkaProducer(bootstrap_servers=KAFKA_BROKER, value_serializer=lambda v: json.dumps(v).encode("utf-8"))
print("Starting Kafka Producer...")

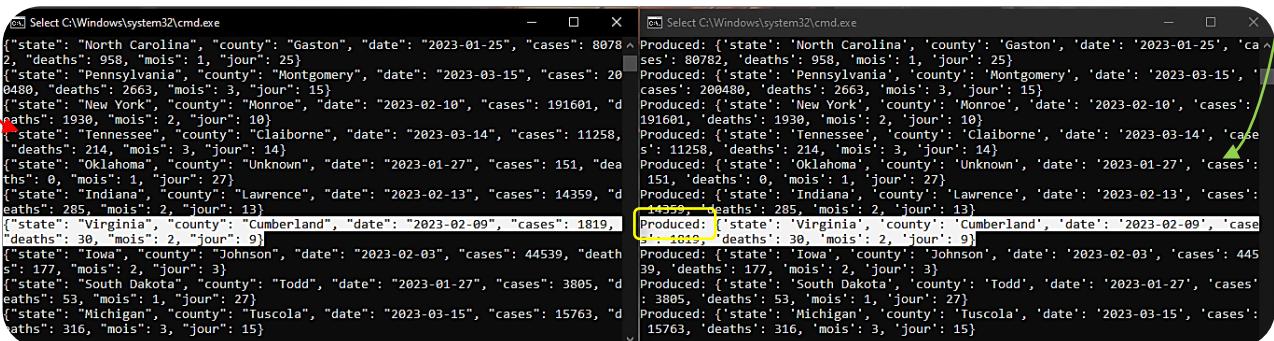
try:
    # Stream data row by row using an iterator to save memory
    for row in df.toLocalIterator():
        message = {
            "state": row["state"],
            "county": row["county"],
            "date": str(row["date"]),
            "cases": int(row["cases"]),
            "deaths": int(row["deaths"]),
            "mois": int(row["mois"]),
            "jour": int(row["jour"])
        }
        producer.send(TOPIC, value=message)
        print(f"Produced: {message}")
        time.sleep(1)

```

Création de topic :

```
root@hadoop-master:~/bigdata project# kafka-topics.sh --create --topic risk_data --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid issues it is best to use either, but not both.
```

Flux de données du producteur Kafka vers le topic `risk_data` :



```
Produced: {"state": "North Carolina", "county": "Gaston", "date": "2023-01-25", "cases": 8078, "deaths": 958, "mois": 1, "jour": 25}
Produced: {"state": "Pennsylvania", "county": "Montgomery", "date": "2023-03-15", "cases": 200480, "deaths": 2663, "mois": 3, "jour": 15}
Produced: {"state": "New York", "county": "Monroe", "date": "2023-02-10", "cases": 191601, "deaths": 1930, "mois": 2, "jour": 10}
Produced: {"state": "Tennessee", "county": "Claiborne", "date": "2023-03-14", "cases": 11258, "deaths": 214, "mois": 3, "jour": 14}
Produced: {"state": "Oklahoma", "county": "Unknown", "date": "2023-01-27", "cases": 151, "deaths": 0, "mois": 1, "jour": 27}
Produced: {"state": "Indiana", "county": "Lawrence", "date": "2023-02-13", "cases": 14359, "deaths": 285, "mois": 2, "jour": 13}
Produced: {"state": "Virginia", "county": "Cumberland", "date": "2023-02-09", "cases": 1819, "deaths": 30, "mois": 2, "jour": 9}
Produced: {"state": "Iowa", "county": "Johnson", "date": "2023-02-03", "cases": 44539, "deaths": 177, "mois": 2, "jour": 3}
Produced: {"state": "South Dakota", "county": "Todd", "date": "2023-01-27", "cases": 3805, "deaths": 53, "mois": 1, "jour": 27}
Produced: {"state": "Michigan", "county": "Tuscola", "date": "2023-03-15", "cases": 15763, "deaths": 316, "mois": 3, "jour": 15}
```

Format des messages envoyés à Kafka :

```
{"state": "Pennsylvania", "county": "Somerset", "date": "2023-02-10", "cases": 22083, "deaths": 442, "mois": 2, "jour": 10}
{"state": "Missouri", "county": "Cape Girardeau", "date": "2023-02-07", "cases": 23276, "deaths": 270, "mois": 2, "jour": 7}
{"state": "Missouri", "county": "Vernon", "date": "2023-03-15", "cases": 6157, "deaths": 116, "mois": 3, "jour": 15}
```

## Consommateur Spark Streaming (postgre\_consumer.py)

Le consommateur Spark Streaming :

1. Consomme les messages du topic Kafka *risk\_data*

```
# Read stream from Kafka
raw_stream = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", KAFKA_BROKER) \
    .option("subscribe", "risk_data") \
    .option("startingOffsets", "latest") \
    .option("failOnDataLoss", "false") \
    .load()
```

2. Charge le modèle pré-entraîné depuis HDFS

```
# Step 2: Load and apply the trained classification model
logger.info("Loading and applying high-risk zone prediction model")
model_path = "hdfs://hadoop-master:9000/models/GeoRisk_model"
model = RandomForestClassificationModel.load(model_path)
predictions = model.transform(batch_df)
```

3. Applique le modèle pour prédire la catégorie de risque

```
# Step 3: Select and format the output
logger.info("Formatting output for PostgreSQL")
output_df = predictions.select(
    col("state"),
    col("county"),
    col("date"),
    col("risk_category").cast("integer"),
    col("prediction").cast("integer").alias("predicted_risk_category"))
.select("state", "county", "date", "risk_category", "predicted_risk_category")
```

4. Stocke les résultats dans PostgreSQL

```
# Step 4: Write to PostgreSQL
logger.info("Writing high-risk zone predictions to PostgreSQL")

output_df.write \
    .format("jdbc") \
    .option("url", "jdbc:postgresql://localhost:5432/pandemic_db") \
    .option("dbtable", "risk_predictions") \
    .option("user", "spark_user") \
    .option("password", "1234") \
    .option("driver", "org.postgresql.Driver") \
    .mode("append") \
    .save()
logger.info("Predictions successfully saved to PostgreSQL!")
```

## Stockage des predictions de «consumer.py» vers un nouveau topic «risk\_predictions» avant la realisation du postgreSQL :

```

Select root@hadoop-master:~/bigdata_project
2025-04-04 15:07:32,928 - INFO - Socket listening on ('127.0.0.1', 36749)
2025-04-04 15:07:34 WARN ResolveWriteIOStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Datasets and will be disabled.
2025-04-04 15:07:34,777 - INFO - Stream processing started, awaiting termination...
2025-04-04 15:07:39,901 - INFO - Python Server ready to receive messages
2025-04-04 15:07:39,901 - INFO - Received command c on object id p0
2025-04-04 15:07:39,913 - INFO - Preparing data for high-risk zone prediction
2025-04-04 15:07:42,770 - INFO - Loading and applying high-risk zone prediction model
2025-04-04 15:07:51,043 - INFO - Formatting output for Kafka
2025-04-04 15:07:51,714 - INFO - Writing high-risk zone predictions to Kafka
25/04/04 15:07:55 WARN KafkaDataConsumer: KafkaDataConsumer is not running in UninterrupableThread. It may hang when KafkaDataConsumer's methods are interrupted because of KAFKA-1894
25/04/04 15:07:58 WARN KafkaDataConsumer: KafkaDataConsumer is not running in UninterrupableThread. It may hang when KafkaDataConsumer's methods are interrupted because of KAFKA-1894
25/04/04 15:07:58 WARN KafkaDataConsumer: KafkaDataConsumer is not running in UninterrupableThread. It may hang when KafkaDataConsumer's methods are interrupted because of KAFKA-1894

```

```

root@hadoop-master:~/bigdata_project
localhost:9092 --topic risk_predictions --from-beginning
[{"state": "Illinois", "county": "Vermilion", "date": "2023-01-25", "risk_category": 3}, {"state": "Virginia", "county": "Albemarle", "date": "2023-02-28", "risk_category": 3}, {"state": "Tennessee", "county": "Hamilton", "date": "2023-02-11", "risk_category": 2}, {"state": "Pennsylvania", "county": "Somerset", "date": "2023-02-10", "risk_category": 3}, {"state": "Missouri", "county": "Cape Girardeau", "date": "2023-02-07", "risk_category": 3}, {"state": "Missouri", "county": "Vernon", "date": "2023-03-15", "risk_category": 2}, {"state": "Kentucky", "county": "Grayson", "date": "2023-02-07", "risk_category": 3}, {"state": "Missouri", "county": "Saline", "date": "2023-02-10", "risk_category": 2}, {"state": "Oklahoma", "county": "Beaver", "date": "2023-02-04", "risk_category": 2}, {"state": "South Dakota", "county": "Golden Valley", "date": "2023-01-25", "risk_category": 1}, {"state": "Virginia", "county": "Dickenson", "date": "2023-02-11", "risk_category": 2}, {"state": "Missouri", "county": "Newton", "date": "2023-02-09", "risk_category": 3}, {"state": "South Dakota", "county": "Harding", "date": "2023-03-15", "risk_category": 1}

```

Realisations des predictions en temps réel ( kafka + spark streaming ) :



Stockage des predictions de «consumer.py» vers la base de données «pandemic\_db» après la réalisation du PostgreSQL :

The image shows two terminal windows. The left window displays the command execution and logs for 'consumer.py':

```
root@hadoop-master:~/bigdata_project
2025-04-06 04:37:00 WARN AdminClientConfig: These configurations '[key.deserializer, value.deserializer, enable.auto.commit, max.poll.records, auto.offset.reset]' were supplied but are not used yet.
2025-04-06 04:37:04,046 - INFO - Python Server ready to receive messages
2025-04-06 04:37:04,646 - INFO - Received command c on object id p0
2025-04-06 04:37:04,660 - INFO - Preparing data for high-risk zone prediction
2025-04-06 04:37:07,253 - INFO - Loading and applying high-risk zone prediction model
2025-04-06 04:37:18,472 - INFO - Formatting output for PostgreSQL
2025-04-06 04:37:18,600 - INFO - Writing high-risk zone predictions to PostgreSQL

2025-04-06 04:37:23,345 - INFO - High-risk zone predictions successfully saved to the database
```

The right window shows the resulting PostgreSQL query output:

state	county	date	risk_category
Virginia	Albemarle	2023-02-28	3
Tennessee	Hamilton	2023-02-11	4
Pennsylvania	Somerset	2023-02-10	3
Missouri	Cape Girardeau	2023-02-07	3
Illinois	Vermilion	2023-01-25	3
Virginia	Albemarle	2023-02-28	3
Tennessee	Hamilton	2023-02-11	4
Pennsylvania	Somerset	2023-02-10	3
Missouri	Cape Girardeau	2023-02-07	3
Missouri	Vernon	2023-03-15	2

--More--

Le pipeline de traitement en temps réel :

- Établit une connexion à Kafka pour consommer les messages en continu
- Désérialise les messages JSON en DataFrame Spark
- Charge le modèle pré-entraîné depuis HDFS
- Applique le modèle pour générer des prédictions
- Stocke les résultats dans PostgreSQL

## 6. Stockage des prédictions

Les prédictions sont stockées dans une base de données PostgreSQL pour permettre une analyse ultérieure et alimenter les outils de visualisation :

```
root@hadoop-master:~/bigdata_project
^
postgres=# CREATE DATABASE pandemic_db;
CREATE DATABASE
postgres=# \c pandemic_db
You are now connected to database "pandemic_db" as user "postgres".
pandemic_db=#
pandemic_db=# CREATE TABLE risk_predictions (
pandemic_db(#   id SERIAL PRIMARY KEY,
pandemic_db(#   state VARCHAR(100),
pandemic_db(#   county VARCHAR(100),
pandemic_db(#   date DATE,
pandemic_db(#   risk_category INT,
pandemic_db(#   predicted_risk_category INT
pandemic_db(# );
CREATE TABLE
pandemic_db=#
pandemic_db=# CREATE USER spark_user WITH PASSWORD '1234';
CREATE ROLE
pandemic_db=# GRANT ALL PRIVILEGES ON DATABASE pandemic_db TO spark_user;
GRANT
pandemic_db=#
```

```
postgres=# \c pandemic_db
You are now connected to database "pandemic_db" as user "postgres".
pandemic_db# \d
              List of relations
 Schema |           Name            |   Type   |  Owner
-----+---------------------+-----+-----+
 public | risk_predictions      | table  | postgres
 public | risk_predictions_id_seq | sequence | postgres
(2 rows)
```

Résulta des predictions stocké dans PostgreSQL :

Cette base de données sert de pont entre le système de traitement en temps réel et les outils de visualisation.

## 7. Visualisation des résultats

### Visualisation avec Streamlit (*streamlit\_dashboard.py*)

J'ai développé un tableau de bord interactif avec Streamlit (*streamlit\_dashboard.py*) pour visualiser les prédictions stockées dans PostgreSQL. Ce tableau de bord offre différentes vues:

1. **Analyse des Risques** : Distribution des catégories de risque, comparaison entre risques réels et prédicts
2. **Distribution Géographique** : Carte choroplète des États-Unis montrant les niveaux de risque par état
3. **Données Détailées** : Affichage des données brutes avec options de filtrage

### Fonctionnalités principales du tableau de bord :

- Chargement dynamique des données depuis PostgreSQL

```
def load_postgresql_data():
    try:
        st.info("Loading data from PostgreSQL database...")

        # Establish connection to PostgreSQL
        conn = psycopg2.connect(**DB_PARAMS)
        # Ensure the date column is cast to a date type before applying TO_CHAR
        cur = conn.cursor(cursor_factory=RealDictCursor)
        cur.execute("ALTER TABLE risk_predictions ALTER COLUMN date TYPE DATE USING date::DATE")
        conn.commit()
        # Query to fetch all data from risk_predictions table
        query = "SELECT state, county, TO_CHAR(date, 'YYYY-MM-DD') as date, risk_category, predicted_risk_category FROM risk_predictions"

        # Load data into pandas DataFrame
        df = pd.read_sql_query(query, conn)

        # Close connection
        conn.close()
```

- Filtrage par état, catégorie de risque et période

```
with col1:
    # Filter by state
    states = ['All'] + sorted(st.session_state.predictions['state'].unique().tolist())
    selected_state = st.selectbox("Filter by State", states)

with col2:
    # Filter by risk category
    if 'risk_category_text' in st.session_state.predictions.columns:
        risk_categories = ['All'] + sorted(st.session_state.predictions['risk_category_text'].unique().tolist())
        selected_risk = st.selectbox("Filter by Risk Category", risk_categories)
    else:
        risk_categories = ['All'] + [RISK_CATEGORIES[x] for x in sorted(st.session_state.predictions['risk_category'].unique().tolist())]
        selected_risk = st.selectbox("Filter by Risk Category", risk_categories)
```

- Visualisations interactives (graphiques à secteurs, cartes thermiques, cartes choroplèthes)

```
# Create the choropleth map
fig = px.choropleth(
    state_risk,
    locations='state_code',
    locationmode='USA-states',
    color='risk_category',
    scope='usa',
    color_continuous_scale='RdYlGn_r', # Red for high risk, green for low
    range_color=[0, 4], # Range of risk categories
    labels={'risk_category': 'Risk Level'},
    hover_data=['state'])
```

- Export des données filtrées au format CSV

```
# Download CSV option
    csv = final_display.to_csv(index=False).encode('utf-8')
    st.download_button(
        "Download Filtered Data as CSV",
        csv,
        "covid_risk_predictions.csv",
        "text/csv",
        key='download-csv'
    )
else:
    st.info("No data available for display.")
```

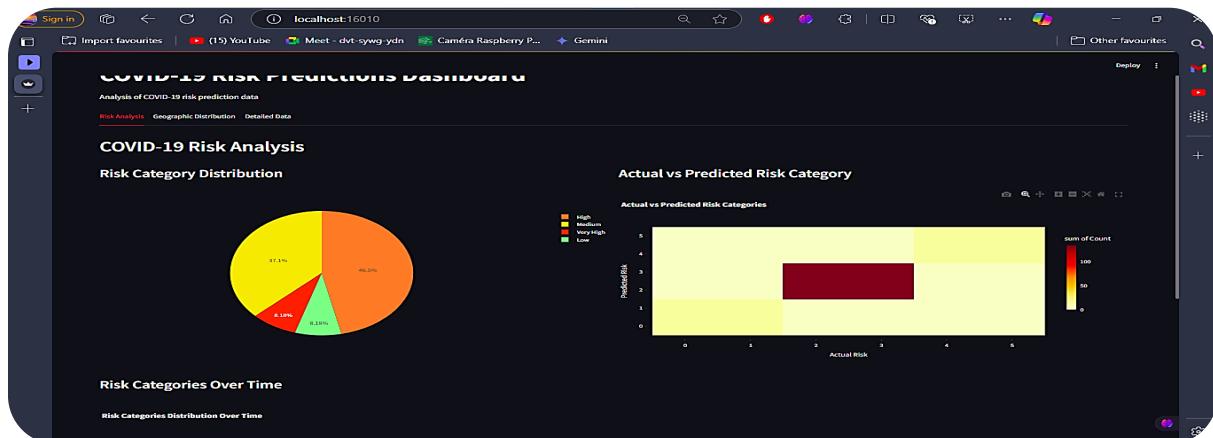
The terminal window shows Streamlit app logs, including:

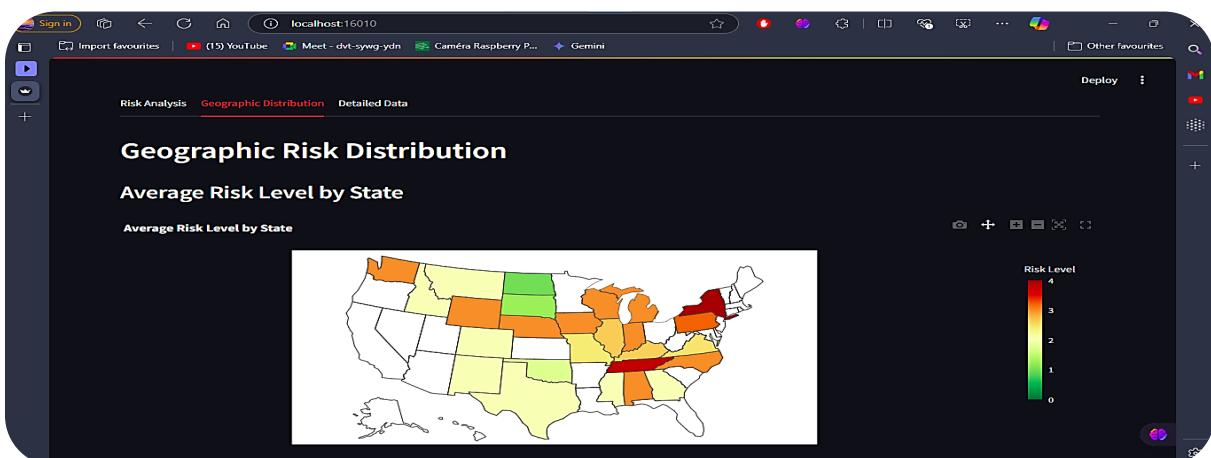
- Collecting usage statistics. To deactivate, set `browser.gatherUsageStats` to false.
- You can now view your Streamlit app in your browser.
- URL: <http://0.0.0.0:16010>
- A value is trying to be set on a copy of a slice from a DataFrame. Try using `.loc[row_indexer,col_indexer] = value` instead.
- See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
- Processed a total of 120 messages

The browser window displays the 'Detailed Prediction Data' dashboard with the following interface elements:

- Detailed Prediction Data** title
- Filter by State dropdown (set to All)
- Filter by Risk Category dropdown (set to All)
- Select Date Range input field (set to 2023/01/25 - 2023/03/23)
- Filtered Data (120 records)** section
- Table showing 5 rows of filtered data:

	State	County	Date	Risk Category	Predicted
0	Illinois	Vermilion	2023-01-25 00:00:00	High	High
1	Virginia	Albemarle	2023-02-28 00:00:00	High	High
2	Tennessee	Hamilton	2023-02-11 00:00:00	Very High	Very High
3	Pennsylvania	Somerset	2023-02-10 00:00:00	High	High
4	Missouri	Cape Girardeau	2023-02-07 00:00:00	High	High



**Detailed Prediction Data**

Filter by State: All

Filter by Risk Category: All

Select Date Range: 2023/01/25 – 2023/03/15

### Filtered Data (159 records)

Index	State	County	Date	Category	Predicted Risk
0	Illinois	Vermilion	2023-02-11 00:00:00	Low	High
1	Virginia	Albermarle	2023-02-10 00:00:00	Medium	High
2	Tennessee	Hamilton	2023-02-07 00:00:00	Very High	Very High
3	Pennsylvania	Somerset	2023-02-10 00:00:00	High	High
4	Missouri	Cape Girardeau	2023-02-07 00:00:00	High	High
5	Illinois	Vermilion	2023-01-25 00:00:00	High	High
6	Virginia	Albermarle	2023-02-28 00:00:00	High	High
7	Tennessee	Hamilton	2023-02-11 00:00:00	Very High	Very High
8	Pennsylvania	Somerset	2023-02-10 00:00:00	High	High

## Visualisation avec Grafana

En complément du tableau de bord Streamlit, j'ai configuré Grafana pour fournir des visualisations supplémentaires :

- Évolution temporelle des risques
- Distribution des risques par région
- Métriques clés (nombre total de cas, tendances)

Grafana est connecté directement à la base de données PostgreSQL, ce qui permet des mises à jour en temps réel des visualisations.

The screenshot shows the 'Connections > Data sources' section of the Grafana interface. It is configured for a 'TimescaleDB' connection. Key settings include:

- Connection limits:**
  - Max open:** 100
  - Auto max idle:** Enabled
  - Max idle:** 100
  - Max lifetime:** 14400

A green success message at the bottom states: "Database Connection OK". Below it, a note says: "Next, you can start to visualize data by building a dashboard, or by querying data in the Explore view."

At the bottom are two buttons: "Delete" (pink) and "Save & test" (blue).

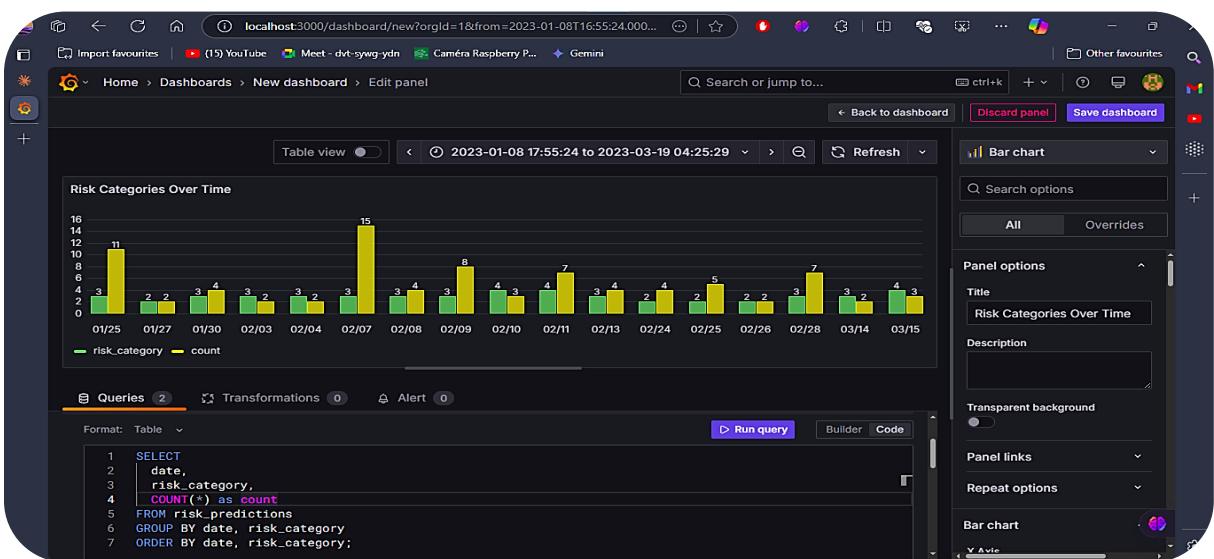
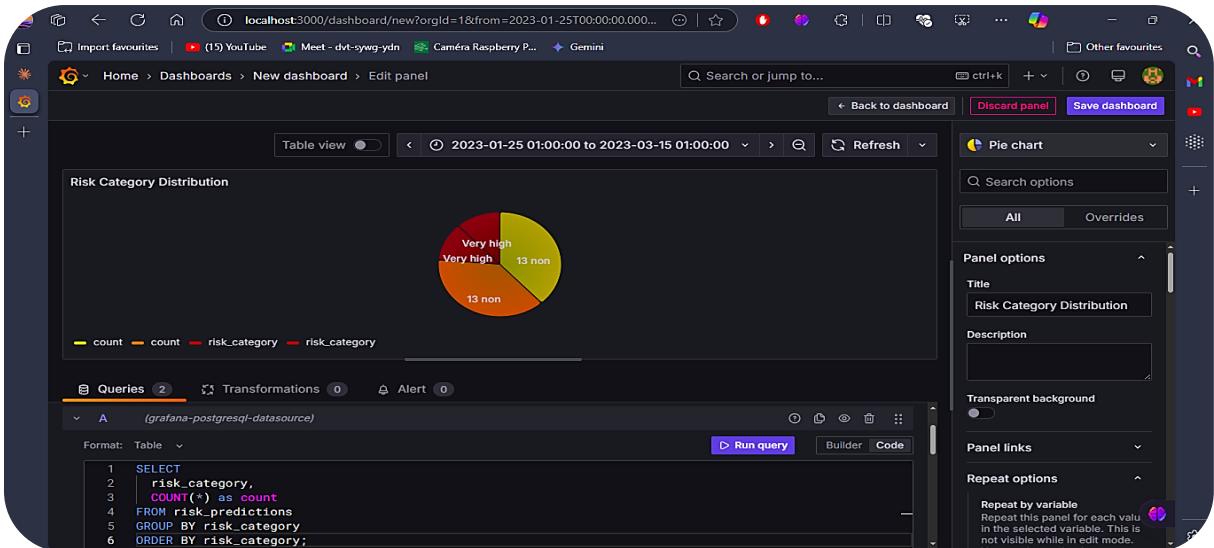
The screenshot shows the 'Dashboards > New dashboard > Edit panel' section of Grafana. A bar chart panel is displayed with the following details:

- Title:** Panel Title
- Time range:** 2023-01-25 01:00:00 to 2023-03-15 01:00:00
- Panel Type:** Bar chart
- Panel options:** Title: Panel Title, Description: (empty), Transparent background: Off
- Panel links:** (empty)
- Repeat options:** Repeat by variable: (empty)

The chart displays data for various US states, grouped by risk category (green bars) and predicted risk category (yellow bars). The x-axis labels include Illinois, South Dakota, Pennsylvania, Iowa, Wyoming, Alabama, Nebraska, Virginia, Wisconsin, Tennessee, North Carolina, Missouri, Colorado, Mississippi, Tennessee, and Michigan.

The 'Queries' tab shows the following SQL query:

```
1 SELECT
2   risk_category,
3   COUNT(*) as count
4   FROM risk_predictions
5   GROUP BY risk_category
6   ORDER BY risk_category;
```



**High Risk Counties**

state	county	date	risk_category
Tennessee	Claiborne	2023-03-14 01:00:00	High
Tennessee	Claiborne	2023-03-14 01:00:00	High
Tennessee	Hamilton	2023-02-11 01:00:00	Very high
Tennessee	Hamilton	2023-02-11 01:00:00	Very high

**Queries**

```

1 SELECT
2   state,
3   county,
4   date,
5   risk_category
6 FROM risk_predictions
7 WHERE risk_category >= 3
8 ORDER BY state, county, date;
    
```

**Panel options**

- Title: High Risk Counties
- Description:
- Transparent background:
- Panel links:
- Repeat options: Table

High Risk Counties		⋮
risk_category	count	
Low	13	non
Medium	59	non
High	74	non
Very high	13	non

risk\_category, count

## 8. Conclusion

Ce projet démontre la mise en œuvre réussie d'une architecture Big Data complète pour la prédiction des risques liés aux pandémies. En utilisant des technologies modernes comme Hadoop, Spark, Kafka et des outils de visualisation, j'ai créé un système capable de :

- Traiter efficacement de grands volumes de données
- Appliquer des modèles prédictifs en temps réel
- Stocker les résultats de manière structurée
- Visualiser les tendances et les prédictions