Final Project: Big Data Application Pipeline

Course Code: CPE 032	**Program: BSCpE**
Course Title: Big Data Engineering	**Date Performed:** 11/18/25
Section: CPE31S1	**Date Submitted:** 11/21/25
Name: Edson Ray San Juan	**Instructor: Engr. Lloyd Pornobi**

Instructions:

- Fork and Clone this neeeal/big-data-streamlit-template to an external site. template.
- Modify the Code to complete the Requirements.
- Discuss the chosen pipeline, application, and significance of the Big Data Application.

Requirements:

- Must use either **MongoDB-Kafka-Spark** or **Hadoop-Kafka-Spark** Pipelines
- Data Source for Producer must be from an Application Programming Interface (API) (e.g. Free Weather API - WeatherAPI.com to an external site.)
- Monitor the streamed data in Live Dashboard (e.g. updates every 15 seconds)
- Present all collected data in History
- Discuss the API used, Pipeline implemented, and significance of the data dashboard.

For Additional Points:

- Include Data Aggregation
- Improve UI/UX
- Include formatted data export (e.g. pdf, jpg, xslx, etc.)

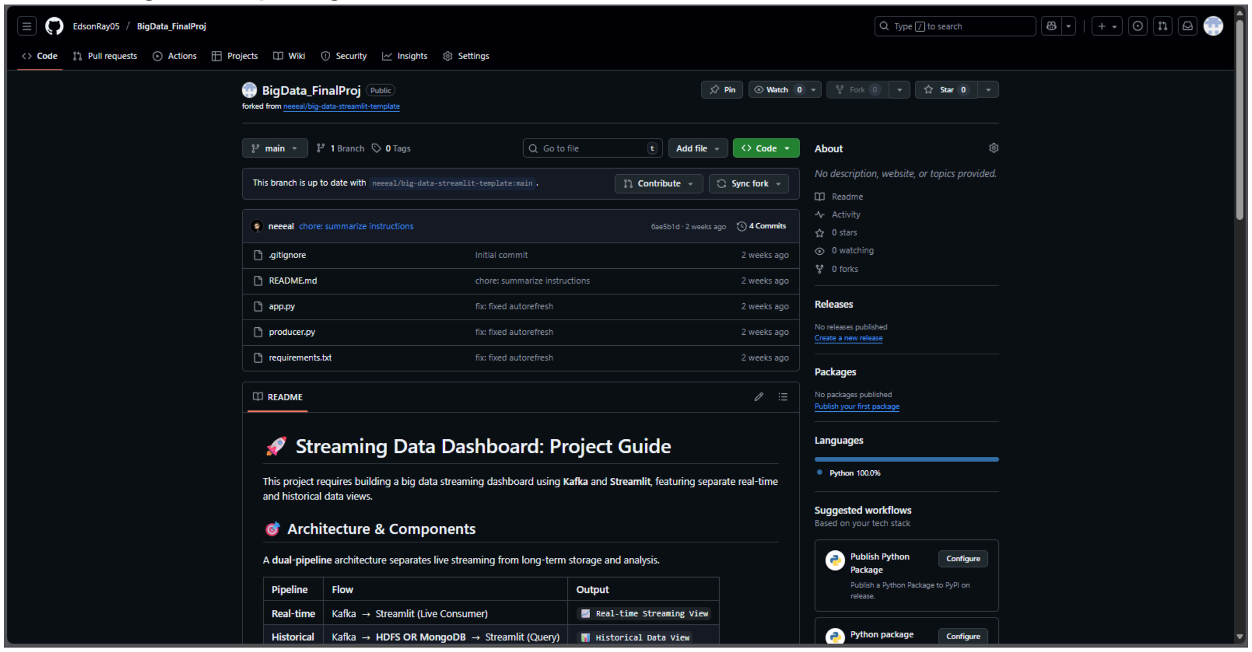
Deliverables:

- Documentation PDF (Activity Report Template)
- Project Demo Video (Maximum 5 minutes)

Github Repository: https://github.com/EdsonRay05/BigData_FinalProj

API: Current weather data - OpenWeatherMap

After forking the template github



BigData_FinalProj Public
forked from neeeal/big-data-streamlit-template

This branch is up to date with `neeeal/big-data-streamlit-template:main`.

neeeal chore: summarize instructions 6ae5b1d · 2 weeks ago 4 Commits

.gitignore	Initial commit	2 weeks ago
README.md	chore: summarize instructions	2 weeks ago
app.py	fix: fixed autorefresh	2 weeks ago
producer.py	fix: fixed autorefresh	2 weeks ago
requirements.txt	fix: fixed autorefresh	2 weeks ago

🚀 Streaming Data Dashboard: Project Guide

This project requires building a big data streaming dashboard using **Kafka** and **Streamlit**, featuring separate real-time and historical data views.

🎯 Architecture & Components

A **dual-pipeline** architecture separates live streaming from long-term storage and analysis.

Pipeline	Flow	Output
Real-time	Kafka → Streamlit (Live Consumer)	📈 `Real-time Streaming View`
Historical	Kafka → **HDFS OR MongoDB** → Streamlit (Query)	📊 `Historical Data View`

About

No description, website, or topics provided.

Readme
Activity
0 stars
0 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

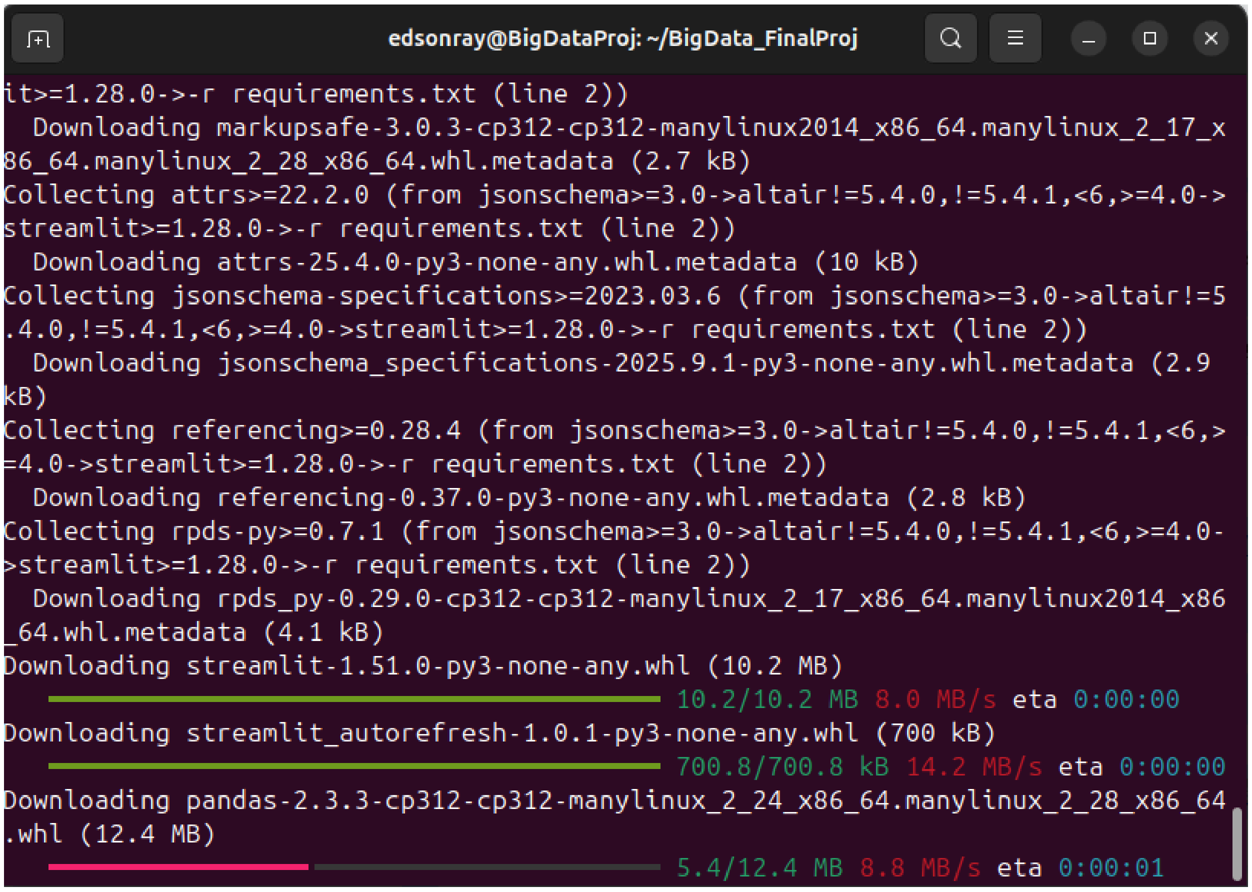
Languages

Python 100.0%

Suggested workflows
Based on your tech stack

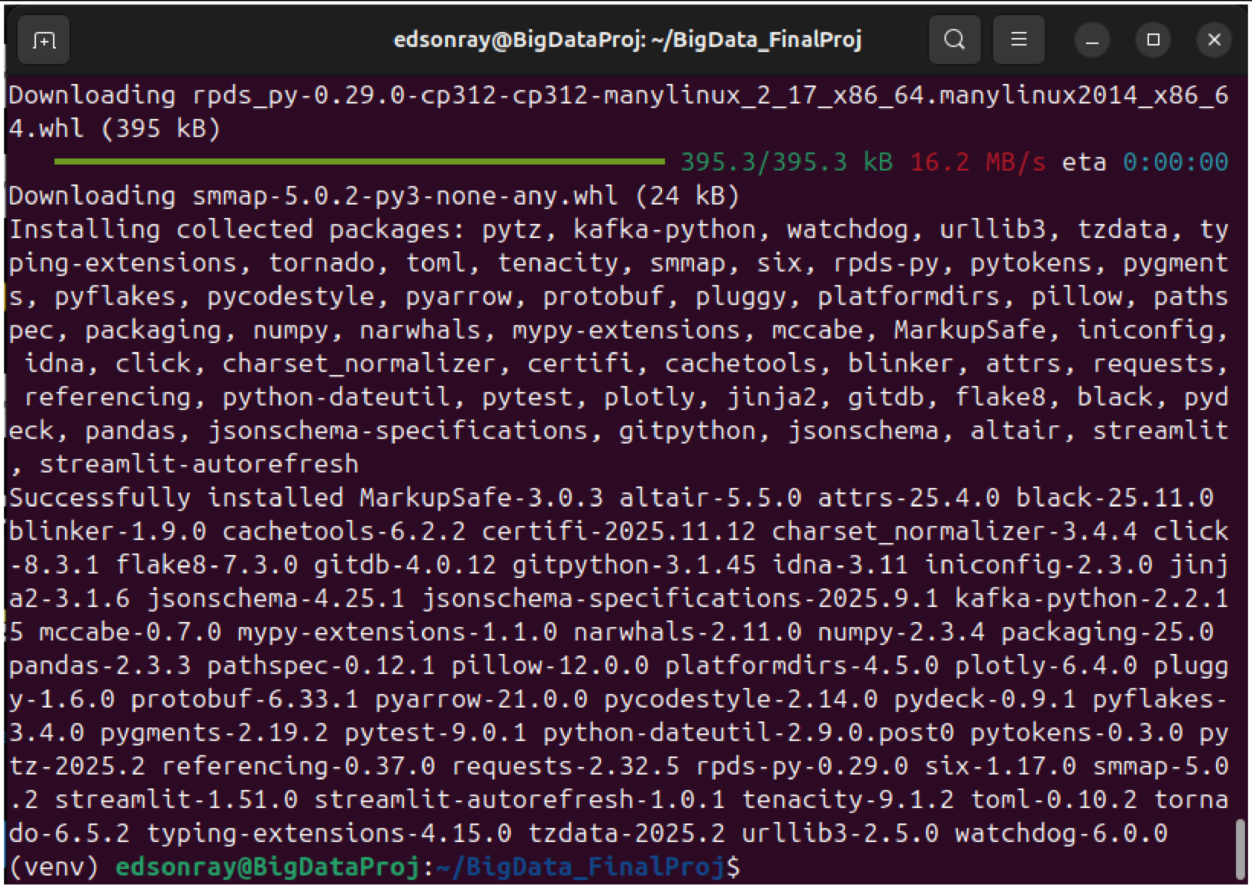
Publish Python Package
Publish a Python Package to PyPI on release.

Python package


edsonray@BigDataProj: ~/BigData_FinalProj

```
it>=1.28.0->-r requirements.txt (line 2))
  Downloading markupsafe-3.0.3-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x
86_64.manylinux_2_28_x86_64.whl.metadata (2.7 kB)
Collecting attrs>=22.2.0 (from jsonschema>=3.0->altair!=5.4.0,!=5.4.1,<6,>=4.0->
streamlit>=1.28.0->-r requirements.txt (line 2))
  Downloading attrs-25.4.0-py3-none-any.whl.metadata (10 kB)
Collecting jsonschema-specifications>=2023.03.6 (from jsonschema>=3.0->altair!=5
.4.0,!=5.4.1,<6,>=4.0->streamlit>=1.28.0->-r requirements.txt (line 2))
  Downloading jsonschema_specifications-2025.9.1-py3-none-any.whl.metadata (2.9
kB)
Collecting referencing>=0.28.4 (from jsonschema>=3.0->altair!=5.4.0,!=5.4.1,<6,>
=4.0->streamlit>=1.28.0->-r requirements.txt (line 2))
  Downloading referencing-0.37.0-py3-none-any.whl.metadata (2.8 kB)
Collecting rpds-py>=0.7.1 (from jsonschema>=3.0->altair!=5.4.0,!=5.4.1,<6,>=4.0-
>streamlit>=1.28.0->-r requirements.txt (line 2))
  Downloading rpds_py-0.29.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl.metadata (4.1 kB)
Downloading streamlit-1.51.0-py3-none-any.whl (10.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.2/10.2 MB 8.0 MB/s eta 0:00:00
Downloading streamlit_autorefresh-1.0.1-py3-none-any.whl (700 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 700.8/700.8 kB 14.2 MB/s eta 0:00:00
Downloading pandas-2.3.3-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64
.whl (12.4 MB)
   ━━━━━━━━━━━━━━━━━╺━━━━━━━━━━━━━━━━━━━━━━ 5.4/12.4 MB 8.8 MB/s eta 0:00:01
```

Downloading the requirements.txt

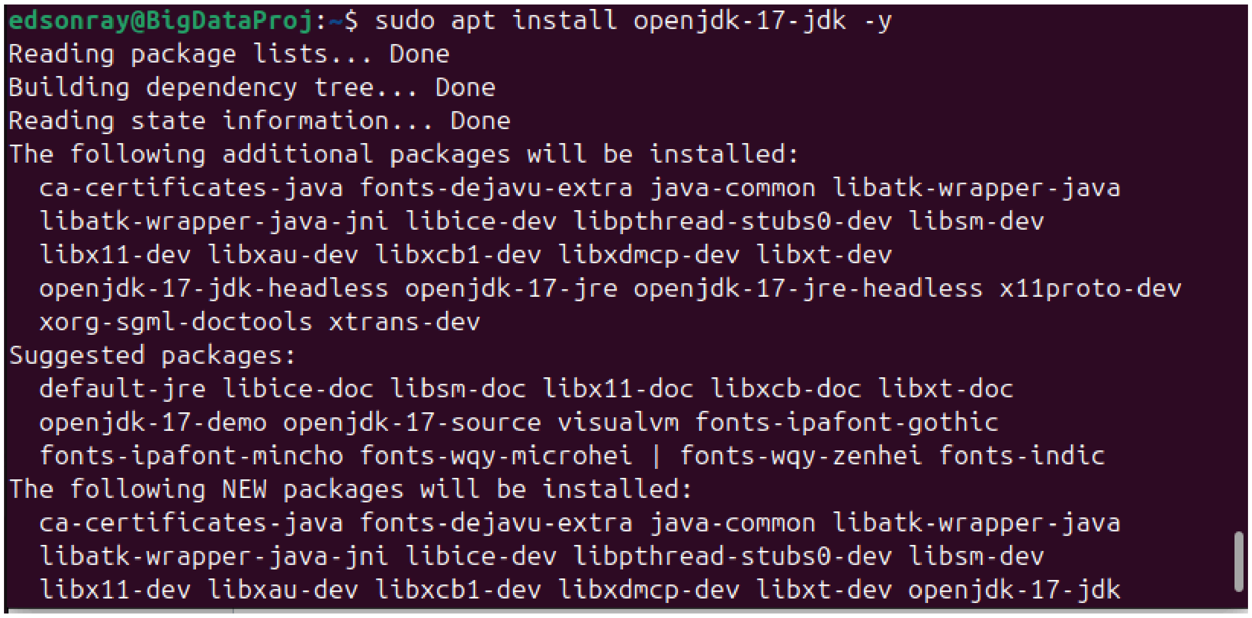


```
edsonray@BigDataProj: ~/BigData_FinalProj
Downloading rpds_py-0.29.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl (395 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 395.3/395.3 kB 16.2 MB/s eta 0:00:00
Downloading smmap-5.0.2-py3-none-any.whl (24 kB)
Installing collected packages: pytz, kafka-python, watchdog, urllib3, tzdata, ty
ping-extensions, tornado, toml, tenacity, smmap, six, rpds-py, pytokens, pygment
s, pyflakes, pycodestyle, pyarrow, protobuf, pluggy, platformdirs, pillow, paths
pec, packaging, numpy, narwhals, mypy-extensions, mccabe, MarkupSafe, iniconfig,
 idna, click, charset_normalizer, certifi, cachetools, blinker, attrs, requests,
 referencing, python-dateutil, pytest, plotly, jinja2, gitdb, flake8, black, pyd
eck, pandas, jsonschema-specifications, gitpython, jsonschema, altair, streamlit
, streamlit-autorefresh
Successfully installed MarkupSafe-3.0.3 altair-5.5.0 attrs-25.4.0 black-25.11.0
blinker-1.9.0 cachetools-6.2.2 certifi-2025.11.12 charset_normalizer-3.4.4 click
-8.3.1 flake8-7.3.0 gitdb-4.0.12 gitpython-3.1.45 idna-3.11 iniconfig-2.3.0 jinj
a2-3.1.6 jsonschema-4.25.1 jsonschema-specifications-2025.9.1 kafka-python-2.2.1
5 mccabe-0.7.0 mypy-extensions-1.1.0 narwhals-2.11.0 numpy-2.3.4 packaging-25.0
pandas-2.3.3 pathspec-0.12.1 pillow-12.0.0 platformdirs-4.5.0 plotly-6.4.0 plugg
y-1.6.0 protobuf-6.33.1 pyarrow-21.0.0 pycodestyle-2.14.0 pydeck-0.9.1 pyflakes-
3.4.0 pygments-2.19.2 pytest-9.0.1 python-dateutil-2.9.0.post0 pytokens-0.3.0 py
tz-2025.2 referencing-0.37.0 requests-2.32.5 rpds-py-0.29.0 six-1.17.0 smmap-5.0
.2 streamlit-1.51.0 streamlit-autorefresh-1.0.1 tenacity-9.1.2 toml-0.10.2 torna
do-6.5.2 typing-extensions-4.15.0 tzdata-2025.2 urllib3-2.5.0 watchdog-6.0.0
(venv) edsonray@BigDataProj:~/BigData_FinalProj$
```

After installing the requirements.txt

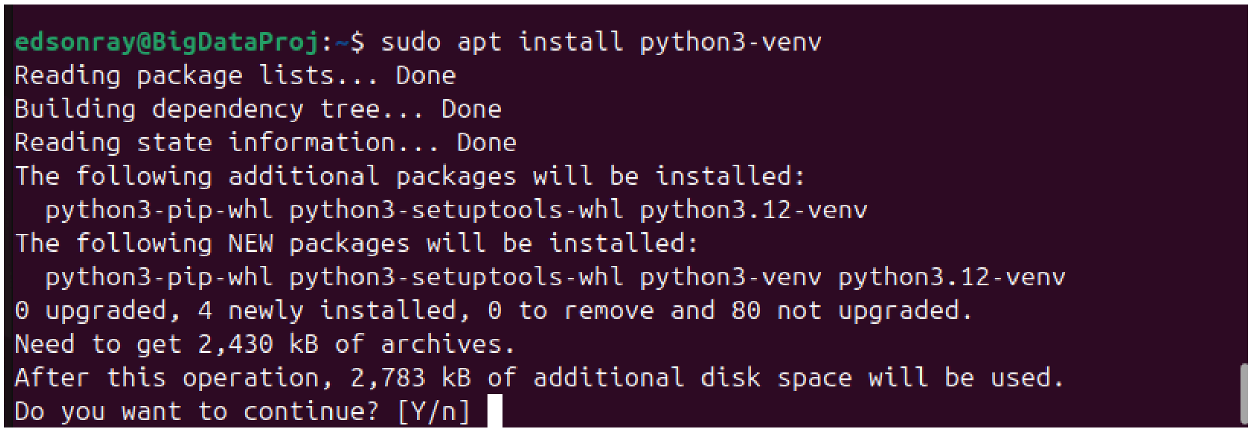
Output (Screenshot)

1. Setting up pyspark



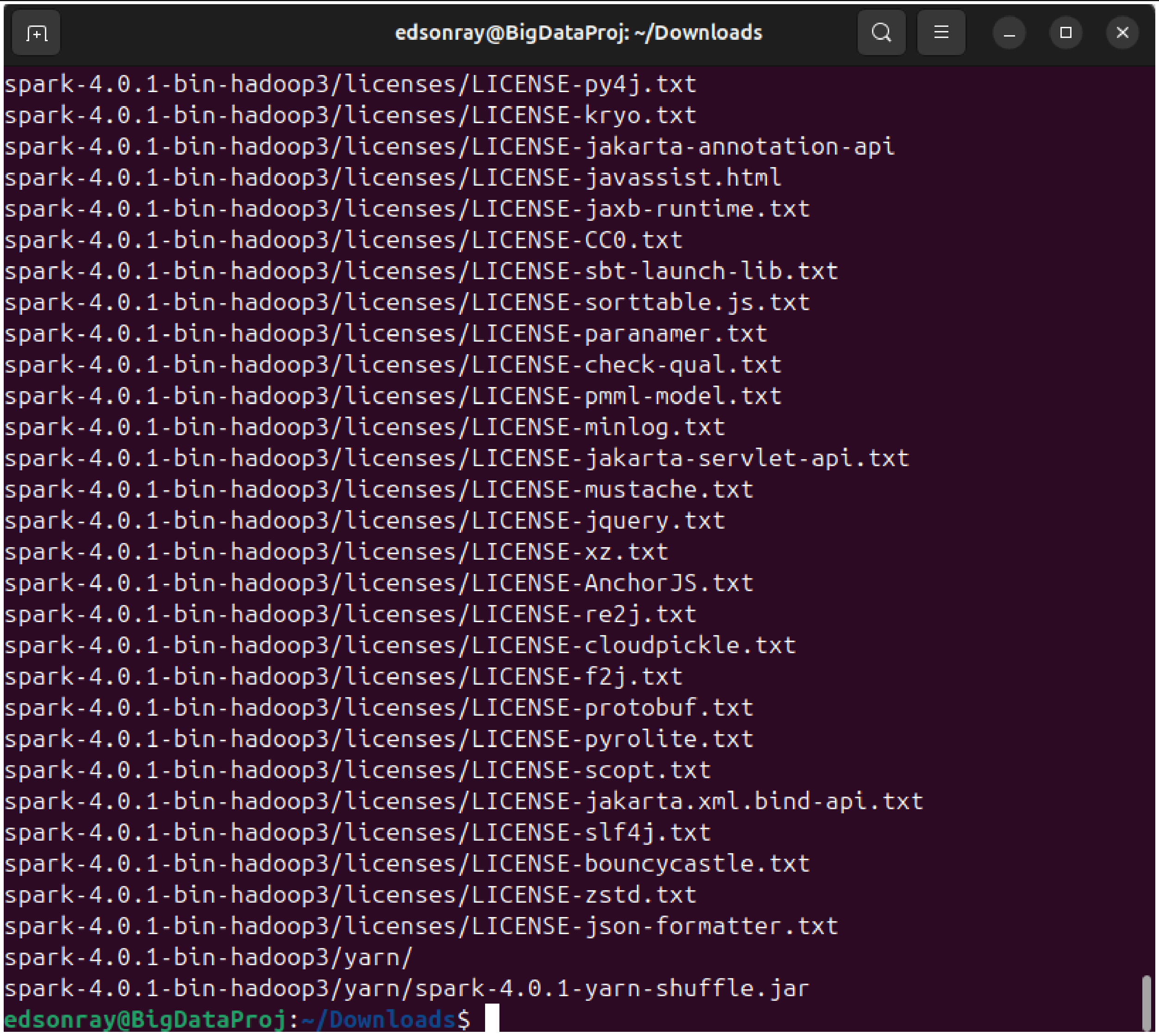
```
edsonray@BigDataProj:~$ sudo apt install openjdk-17-jdk -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  ca-certificates-java fonts-dejavu-extra java-common libatk-wrapper-java
  libatk-wrapper-java-jni libice-dev libpthread-stubs0-dev libsm-dev
  libx11-dev libxau-dev libxcb1-dev libxdmcp-dev libxt-dev
  openjdk-17-jdk-headless openjdk-17-jre openjdk-17-jre-headless x11proto-dev
  xorg-sgml-doctools xtrans-dev
Suggested packages:
  default-jre libice-doc libsm-doc libx11-doc libxcb-doc libxt-doc
  openjdk-17-demo openjdk-17-source visualvm fonts-ipafont-gothic
  fonts-ipafont-mincho fonts-wqy-microhei | fonts-wqy-zenhei fonts-indic
The following NEW packages will be installed:
  ca-certificates-java fonts-dejavu-extra java-common libatk-wrapper-java
  libatk-wrapper-java-jni libice-dev libpthread-stubs0-dev libsm-dev
  libx11-dev libxau-dev libxcb1-dev libxdmcp-dev libxt-dev openjdk-17-jdk
```

sudo apt install openjdk-17-jdk -y installs the OpenJDK 17 Java Development Kit package on the system and uses -y to confirm the installation without prompting for user approval.



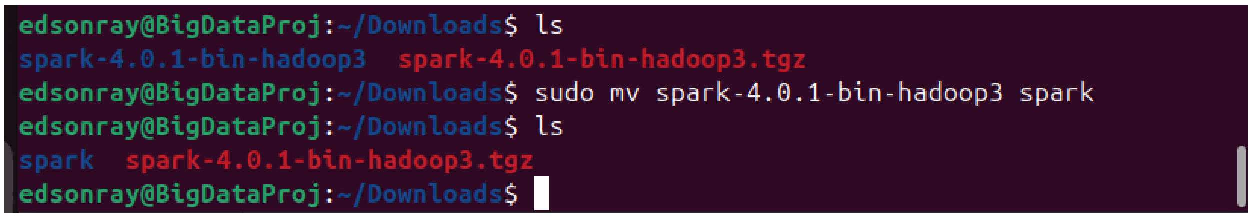
```
edsonray@BigDataProj:~$ sudo apt install python3-venv
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  python3-pip-whl python3-setuptools-whl python3.12-venv
The following NEW packages will be installed:
  python3-pip-whl python3-setuptools-whl python3-venv python3.12-venv
0 upgraded, 4 newly installed, 0 to remove and 80 not upgraded.
Need to get 2,430 kB of archives.
After this operation, 2,783 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

installing the python3-venv package using the apt command, which will set up the necessary files to create Python virtual environments

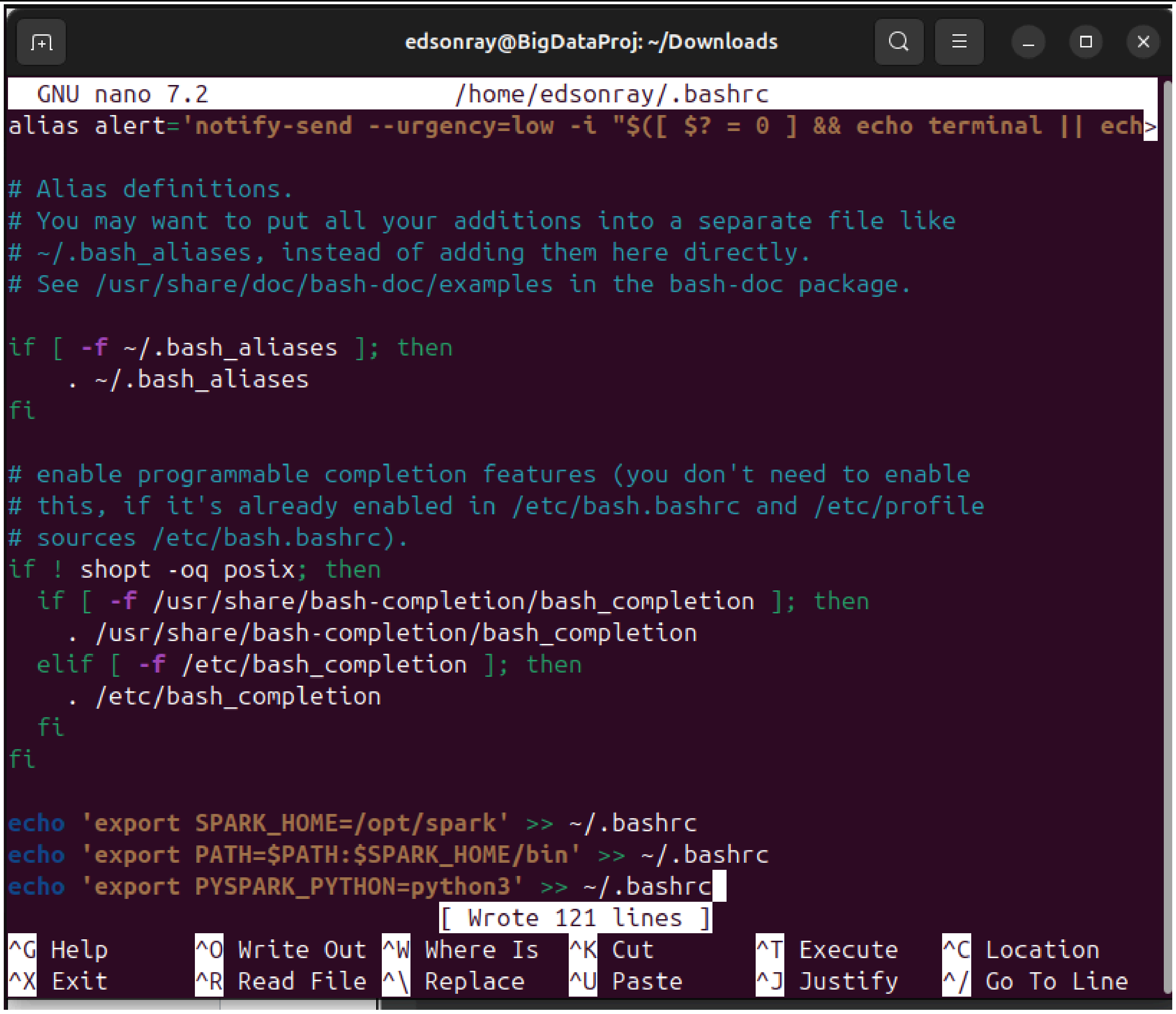


```
spark-4.0.1-bin-hadoop3/licenses/LICENSE-py4j.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-kryo.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-jakarta-annotation-api
spark-4.0.1-bin-hadoop3/licenses/LICENSE-javassist.html
spark-4.0.1-bin-hadoop3/licenses/LICENSE-jaxb-runtime.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-CC0.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-sbt-launch-lib.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-sorttable.js.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-paranamer.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-check-qual.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-pmml-model.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-minlog.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-jakarta-servlet-api.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-mustache.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-jquery.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-xz.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-AnchorJS.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-re2j.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-cloudpickle.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-f2j.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-protobuf.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-pyrolite.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-scopt.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-jakarta.xml.bind-api.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-slf4j.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-bouncycastle.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-zstd.txt
spark-4.0.1-bin-hadoop3/licenses/LICENSE-json-formatter.txt
spark-4.0.1-bin-hadoop3/yarn/
spark-4.0.1-bin-hadoop3/yarn/spark-4.0.1-yarn-shuffle.jar
edsonray@BigDataProj:~/Downloads$ 
```

sudo tar -xvzf spark-4.0.1-bin-hadoop3.tgz, extract the spark-4.0.1-bin-hadoop3.tgz file using tar with sudo privileges.

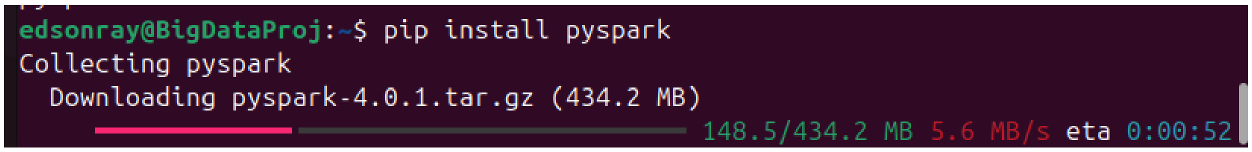


```
edsonray@BigDataProj:~/Downloads$ ls
spark-4.0.1-bin-hadoop3  spark-4.0.1-bin-hadoop3.tgz
edsonray@BigDataProj:~/Downloads$ sudo mv spark-4.0.1-bin-hadoop3 spark
edsonray@BigDataProj:~/Downloads$ ls
spark  spark-4.0.1-bin-hadoop3.tgz
edsonray@BigDataProj:~/Downloads$ 
```



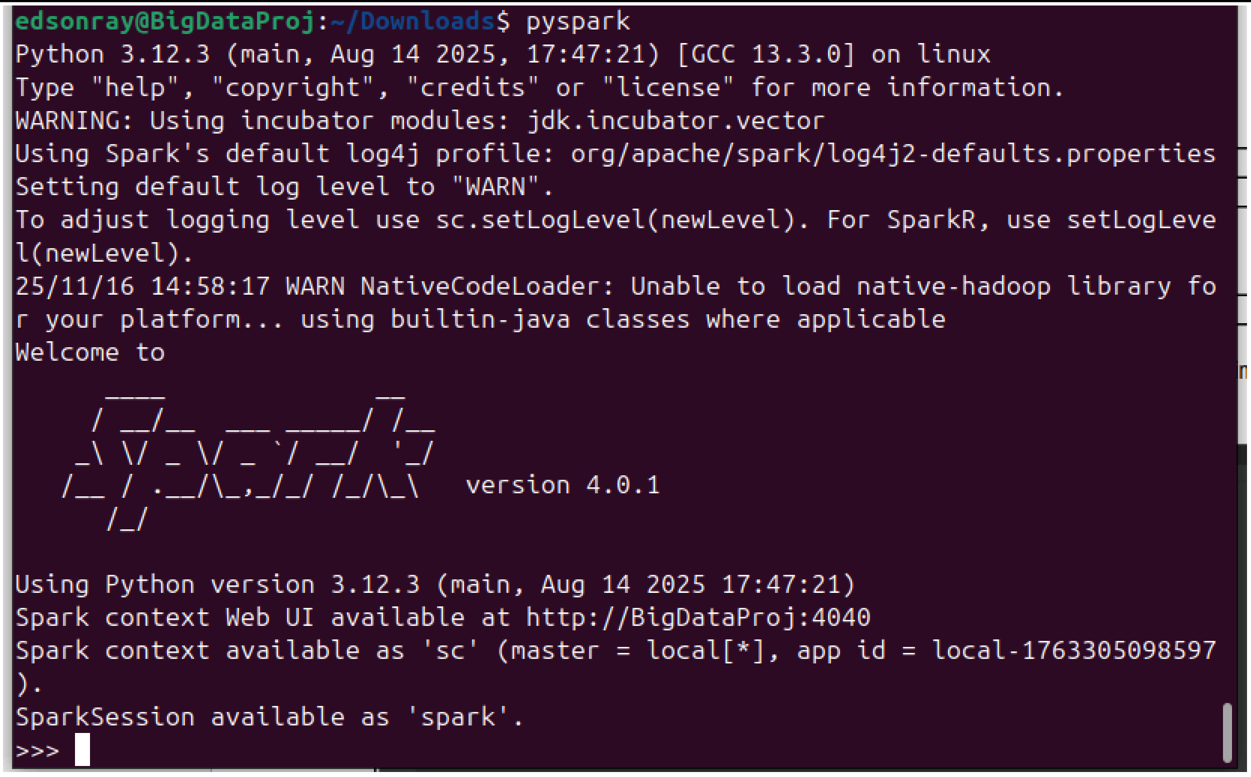
```
  GNU nano 7.2                    /home/edsonray/.bashrc
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || ech>

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

echo 'export SPARK_HOME=/opt/spark' >> ~/.bashrc
echo 'export PATH=$PATH:$SPARK_HOME/bin' >> ~/.bashrc
echo 'export PYSPARK_PYTHON=python3' >> ~/.bashrc
                              [ Wrote 121 lines ]
^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line
```

~/.bashrc file is open in the Nano text editor, showing settings for command aliases and environment variables related to Spark and Python.



```
edsonray@BigDataProj:~$ pip install pyspark
Collecting pyspark
  Downloading pyspark-4.0.1.tar.gz (434.2 MB)
                                               148.5/434.2 MB 5.6 MB/s eta 0:00:52
```

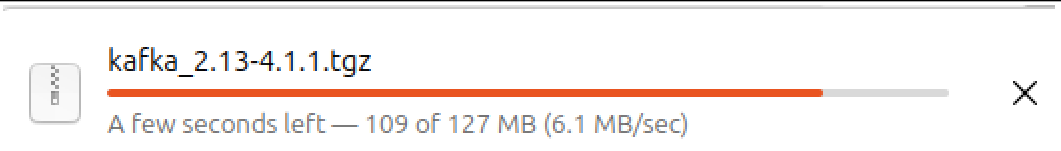
Installing pyspark



```
edsonray@BigDataProj:~/Downloads$ pyspark
Python 3.12.3 (main, Aug 14 2025, 17:47:21) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
25/11/16 14:58:17 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 4.0.1
      /_/

Using Python version 3.12.3 (main, Aug 14 2025 17:47:21)
Spark context Web UI available at http://BigDataProj:4040
Spark context available as 'sc' (master = local[*], app id = local-1763305098597
).
SparkSession available as 'spark'.
>>> 
```

Activating pyspark

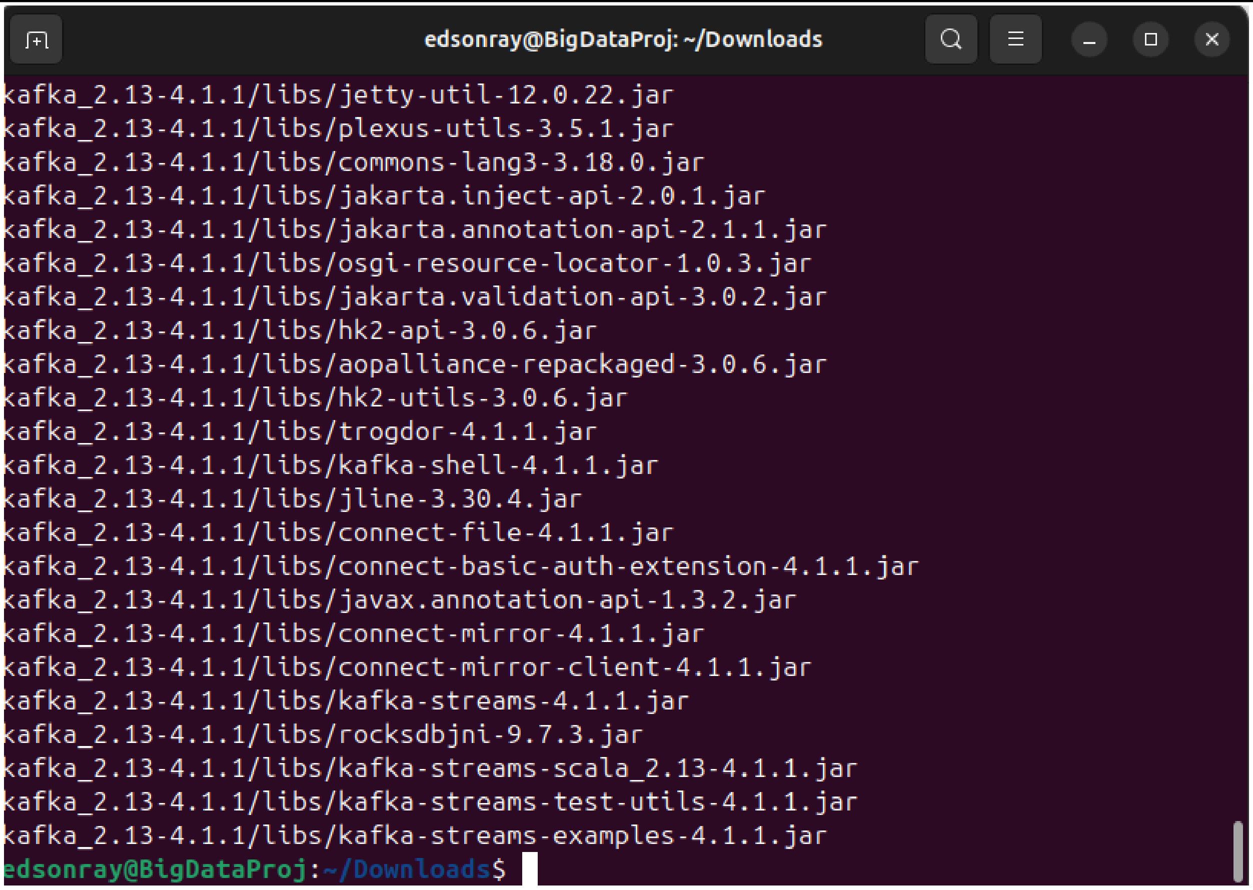
2. Setting up Kafka



kafka_2.13-4.1.1.tgz

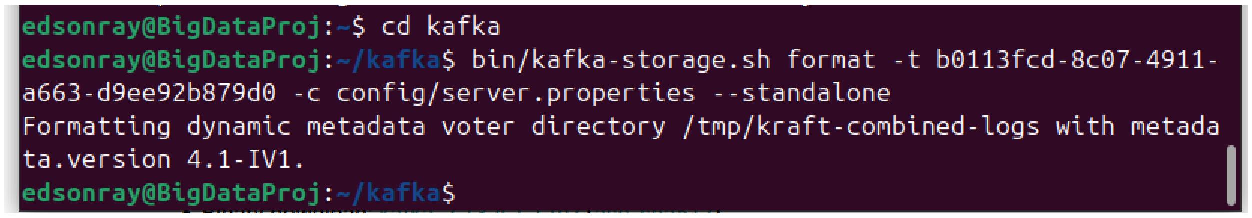
A few seconds left — 109 of 127 MB (6.1 MB/sec)

Downloads kafka tgz file



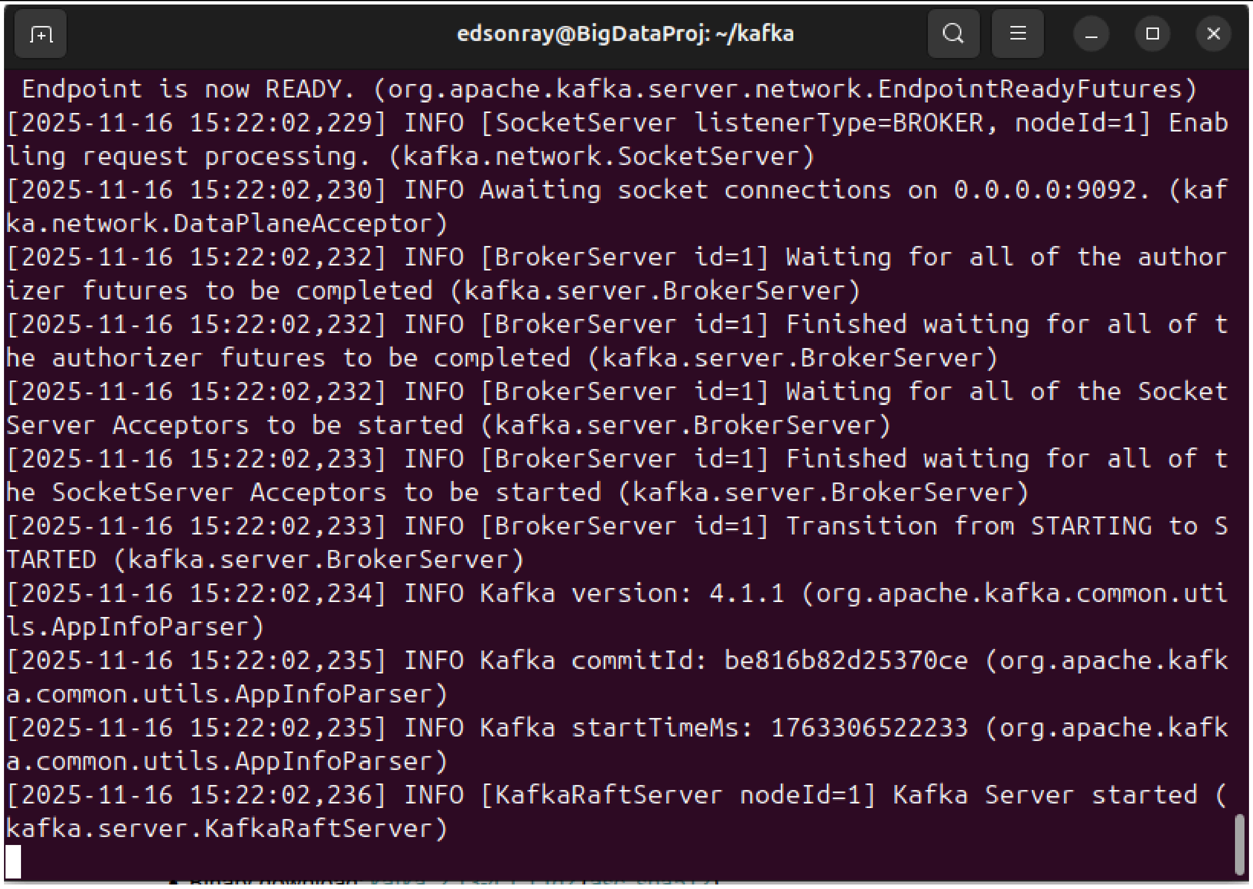
```
edsonray@BigDataProj: ~/Downloads
kafka_2.13-4.1.1/libs/jetty-util-12.0.22.jar
kafka_2.13-4.1.1/libs/plexus-utils-3.5.1.jar
kafka_2.13-4.1.1/libs/commons-lang3-3.18.0.jar
kafka_2.13-4.1.1/libs/jakarta.inject-api-2.0.1.jar
kafka_2.13-4.1.1/libs/jakarta.annotation-api-2.1.1.jar
kafka_2.13-4.1.1/libs/osgi-resource-locator-1.0.3.jar
kafka_2.13-4.1.1/libs/jakarta.validation-api-3.0.2.jar
kafka_2.13-4.1.1/libs/hk2-api-3.0.6.jar
kafka_2.13-4.1.1/libs/aopalliance-repackaged-3.0.6.jar
kafka_2.13-4.1.1/libs/hk2-utils-3.0.6.jar
kafka_2.13-4.1.1/libs/trogdor-4.1.1.jar
kafka_2.13-4.1.1/libs/kafka-shell-4.1.1.jar
kafka_2.13-4.1.1/libs/jline-3.30.4.jar
kafka_2.13-4.1.1/libs/connect-file-4.1.1.jar
kafka_2.13-4.1.1/libs/connect-basic-auth-extension-4.1.1.jar
kafka_2.13-4.1.1/libs/javax.annotation-api-1.3.2.jar
kafka_2.13-4.1.1/libs/connect-mirror-4.1.1.jar
kafka_2.13-4.1.1/libs/connect-mirror-client-4.1.1.jar
kafka_2.13-4.1.1/libs/kafka-streams-4.1.1.jar
kafka_2.13-4.1.1/libs/rocksdbjni-9.7.3.jar
kafka_2.13-4.1.1/libs/kafka-streams-scala_2.13-4.1.1.jar
kafka_2.13-4.1.1/libs/kafka-streams-test-utils-4.1.1.jar
kafka_2.13-4.1.1/libs/kafka-streams-examples-4.1.1.jar
edsonray@BigDataProj:~/Downloads$
```

I extracted the Kafka archive (kafka_2.13-4.1.1.tgz) in my Downloads folder, and now the terminal is displaying the list of JAR files inside the libs directory.



```
edsonray@BigDataProj:~$ cd kafka
edsonray@BigDataProj:~/kafka$ bin/kafka-storage.sh format -t b0113fcd-8c07-4911-
a663-d9ee92b879d0 -c config/server.properties --standalone
Formatting dynamic metadata voter directory /tmp/kraft-combined-logs with metada
ta.version 4.1-IV1.
edsonray@BigDataProj:~/kafka$
```

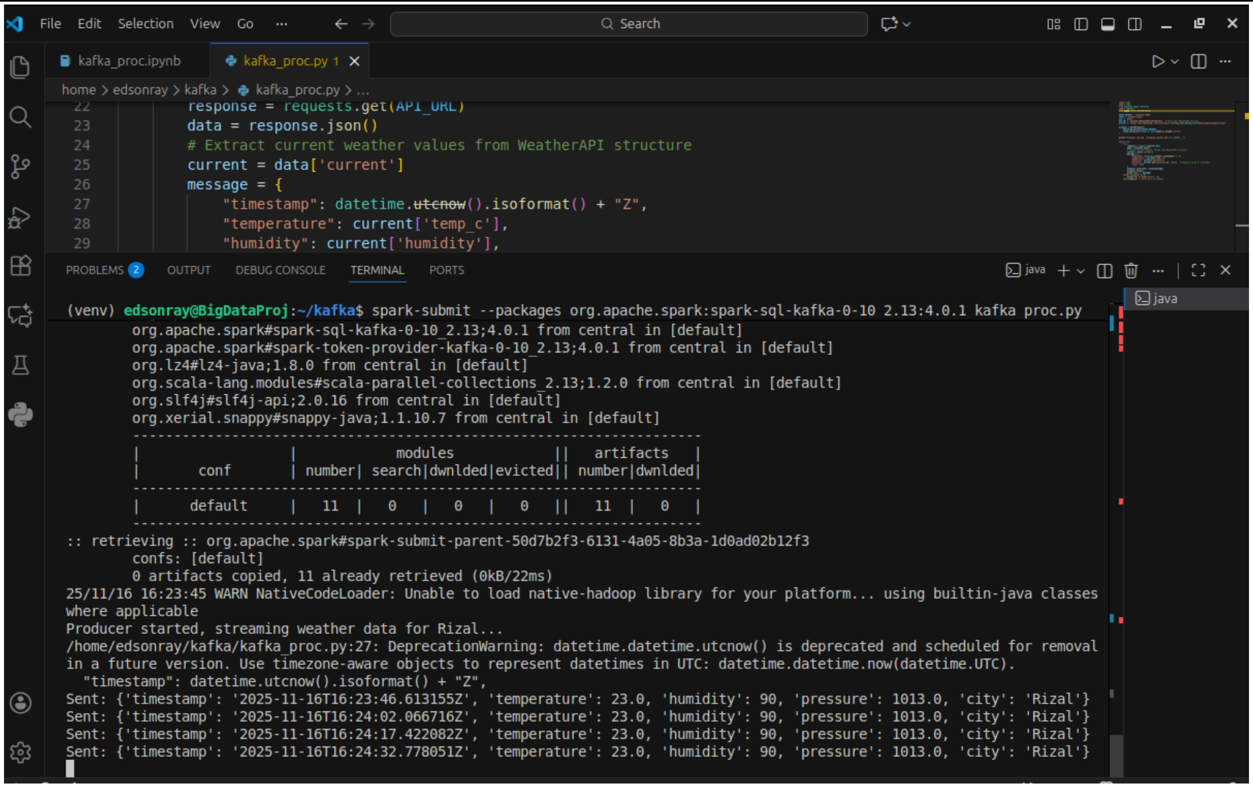
First step to activate kafka



```
 Endpoint is now READY. (org.apache.kafka.server.network.EndpointReadyFutures)
[2025-11-16 15:22:02,229] INFO [SocketServer listenerType=BROKER, nodeId=1] Enab
ling request processing. (kafka.network.SocketServer)
[2025-11-16 15:22:02,230] INFO Awaiting socket connections on 0.0.0.0:9092. (kaf
ka.network.DataPlaneAcceptor)
[2025-11-16 15:22:02,232] INFO [BrokerServer id=1] Waiting for all of the author
izer futures to be completed (kafka.server.BrokerServer)
[2025-11-16 15:22:02,232] INFO [BrokerServer id=1] Finished waiting for all of t
he authorizer futures to be completed (kafka.server.BrokerServer)
[2025-11-16 15:22:02,232] INFO [BrokerServer id=1] Waiting for all of the Socket
Server Acceptors to be started (kafka.server.BrokerServer)
[2025-11-16 15:22:02,233] INFO [BrokerServer id=1] Finished waiting for all of t
he SocketServer Acceptors to be started (kafka.server.BrokerServer)
[2025-11-16 15:22:02,233] INFO [BrokerServer id=1] Transition from STARTING to S
TARTED (kafka.server.BrokerServer)
[2025-11-16 15:22:02,234] INFO Kafka version: 4.1.1 (org.apache.kafka.common.uti
ls.AppInfoParser)
[2025-11-16 15:22:02,235] INFO Kafka commitId: be816b82d25370ce (org.apache.kafk
a.common.utils.AppInfoParser)
[2025-11-16 15:22:02,235] INFO Kafka startTimeMs: 1763306522233 (org.apache.kafk
a.common.utils.AppInfoParser)
[2025-11-16 15:22:02,236] INFO [KafkaRaftServer nodeId=1] Kafka Server started (
kafka.server.KafkaRaftServer)
```

Activate kafka **bin/kafka-server-start.sh config/server.properties**

Python
```python
# kafka_proc.py

import json
import time
import random
from datetime import datetime
import requests
from kafka import KafkaProducer

KAFKA_BROKER = 'localhost:9092'
TOPIC = 'weather_topic'
CITY = 'Antipolo City'
API_KEY = '100a505f5ee7872f379bd184039ebd46'
LAT = 14.6255
LON = 121.1245
API_URL =
f"https://api.openweathermap.org/data/2.5/weather?lat={LAT}&lon={LON}&appid={AP
I_KEY}"

producer = KafkaProducer(
    bootstrap_servers=[KAFKA_BROKER],
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

print(f"Producer started, streaming weather data for {CITY}...")

while True:
    try:
        response = requests.get(API_URL)
        data = response.json()
        main = data.get('main', {})

        # Use some random jitter to simulate live sensor changes
        base_temp = main.get("temp", 0) - 273.15 if "temp" in main else None
        base_humidity = main.get("humidity", None)
        base_pressure = main.get("pressure", None)

        message = {
            "timestamp": datetime.utcnow().isoformat() + "Z",
            "temperature": (base_temp + random.uniform(-0.5, 0.5)) if base_temp
is not None else None,
            "humidity": (base_humidity + random.randint(-2, 2)) if
base_humidity is not None else None,
            "pressure": (base_pressure + random.randint(-1, 1)) if
base_pressure is not None else None,
            "city": CITY
        }
        producer.send(TOPIC, value=message)
        producer.flush()
        print("Sent:", message)
    except Exception as e:
        print("API or Kafka Error:", e)
    time.sleep(15)
```

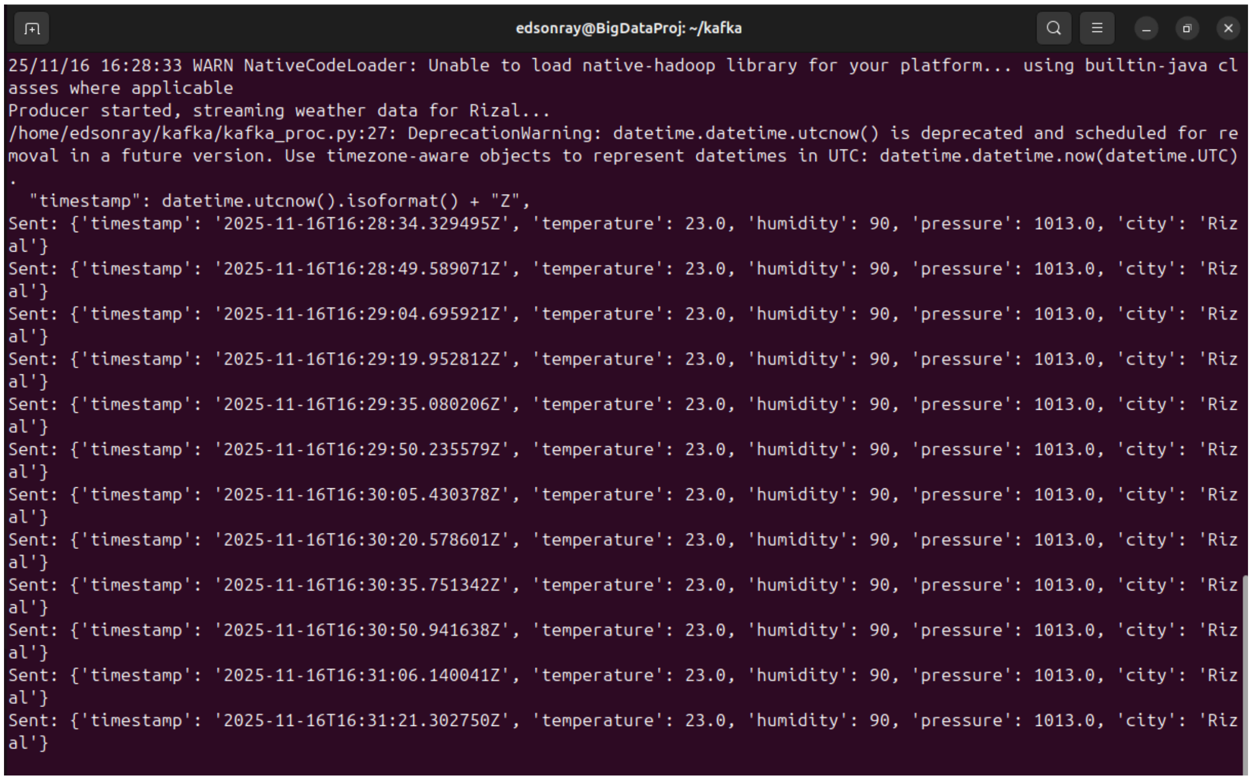


```python
        response = requests.get(API_URL)
        data = response.json()
        # Extract current weather values from WeatherAPI structure
        current = data['current']
        message = {
            "timestamp": datetime.~~utcnow~~().isoformat() + "Z",
            "temperature": current['temp_c'],
            "humidity": current['humidity'],
```

```
(venv) edsonray@BigDataProj:~/kafka$ spark-submit --packages org.apache.spark:spark-sql-kafka-0-10 2.13:4.0.1 kafka proc.py
        org.apache.spark#spark-sql-kafka-0-10_2.13;4.0.1 from central in [default]
        org.apache.spark#spark-token-provider-kafka-0-10_2.13;4.0.1 from central in [default]
        org.lz4#lz4-java;1.8.0 from central in [default]
        org.scala-lang.modules#scala-parallel-collections_2.13;1.2.0 from central in [default]
        org.slf4j#slf4j-api;2.0.16 from central in [default]
        org.xerial.snappy#snappy-java;1.1.10.7 from central in [default]
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   11  |   0   |   0   |   0   ||   11  |   0   |
        ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent-50d7b2f3-6131-4a05-8b3a-1d0ad02b12f3
        confs: [default]
        0 artifacts copied, 11 already retrieved (0kB/22ms)
25/11/16 16:23:45 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes
where applicable
Producer started, streaming weather data for Rizal...
/home/edsonray/kafka/kafka_proc.py:27: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal
in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
  "timestamp": datetime.utcnow().isoformat() + "Z",
Sent: {'timestamp': '2025-11-16T16:23:46.613155Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city': 'Rizal'}
Sent: {'timestamp': '2025-11-16T16:24:02.066716Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city': 'Rizal'}
Sent: {'timestamp': '2025-11-16T16:24:17.422082Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city': 'Rizal'}
Sent: {'timestamp': '2025-11-16T16:24:32.778051Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city': 'Rizal'}
```

Code to run the kafka producer:

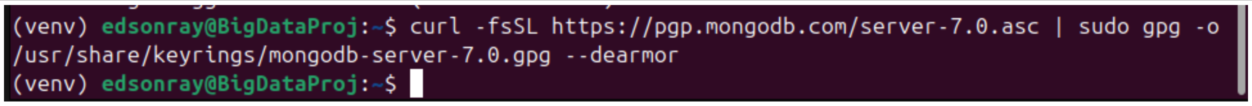
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.13:4.0.1 kafka proc.py



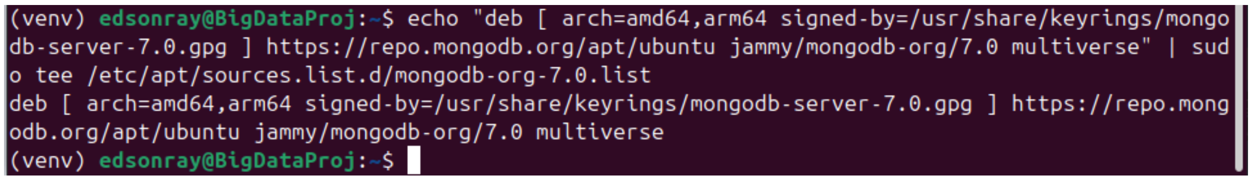
```
25/11/16 16:28:33 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
Producer started, streaming weather data for Rizal...
/home/edsonray/kafka/kafka_proc.py:27: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for re
moval in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC)
.
  "timestamp": datetime.utcnow().isoformat() + "Z",
Sent: {'timestamp': '2025-11-16T16:28:34.329495Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city': 'Riz
al'}
Sent: {'timestamp': '2025-11-16T16:28:49.589071Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city': 'Riz
al'}
Sent: {'timestamp': '2025-11-16T16:29:04.695921Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city': 'Riz
al'}
Sent: {'timestamp': '2025-11-16T16:29:19.952812Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city': 'Riz
al'}
Sent: {'timestamp': '2025-11-16T16:29:35.080206Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city': 'Riz
al'}
Sent: {'timestamp': '2025-11-16T16:29:50.235579Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city': 'Riz
al'}
Sent: {'timestamp': '2025-11-16T16:30:05.430378Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city': 'Riz
al'}
Sent: {'timestamp': '2025-11-16T16:30:20.578601Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city': 'Riz
al'}
Sent: {'timestamp': '2025-11-16T16:30:35.751342Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city': 'Riz
al'}
Sent: {'timestamp': '2025-11-16T16:30:50.941638Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city': 'Riz
al'}
Sent: {'timestamp': '2025-11-16T16:31:06.140041Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city': 'Riz
al'}
Sent: {'timestamp': '2025-11-16T16:31:21.302750Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city': 'Riz
al'}
```

Producer output in VM terminal

3. **Setting up** Mongodb

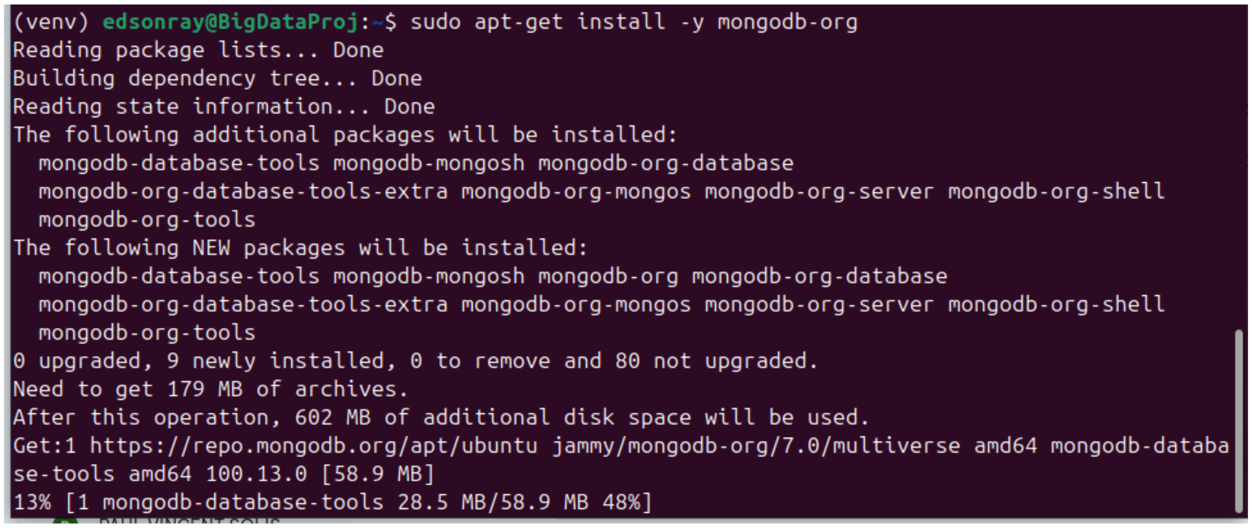


```
(venv) edsonray@BigDataProj:~$ curl -fsSL https://pgp.mongodb.com/server-7.0.asc | sudo gpg -o
/usr/share/keyrings/mongodb-server-7.0.gpg --dearmor
(venv) edsonray@BigDataProj:~$
```



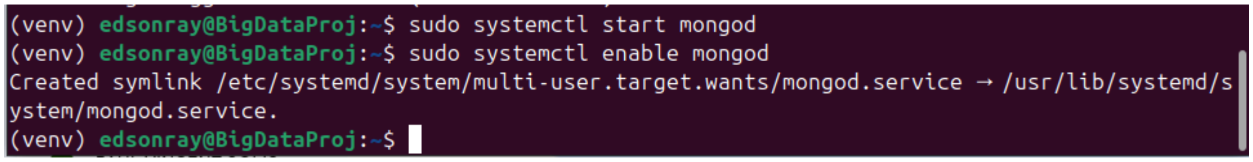
```
(venv) edsonray@BigDataProj:~$ echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongo
db-server-7.0.gpg ] https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0 multiverse" | sud
o tee /etc/apt/sources.list.d/mongodb-org-7.0.list
deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server-7.0.gpg ] https://repo.mong
odb.org/apt/ubuntu jammy/mongodb-org/7.0 multiverse
(venv) edsonray@BigDataProj:~$
```

Downloaded the MongoDB public GPG key and saved it to my keyrings, then added the MongoDB 7.0 repository to my system sources list for Ubuntu, allowing me to install MongoDB using apt commands.



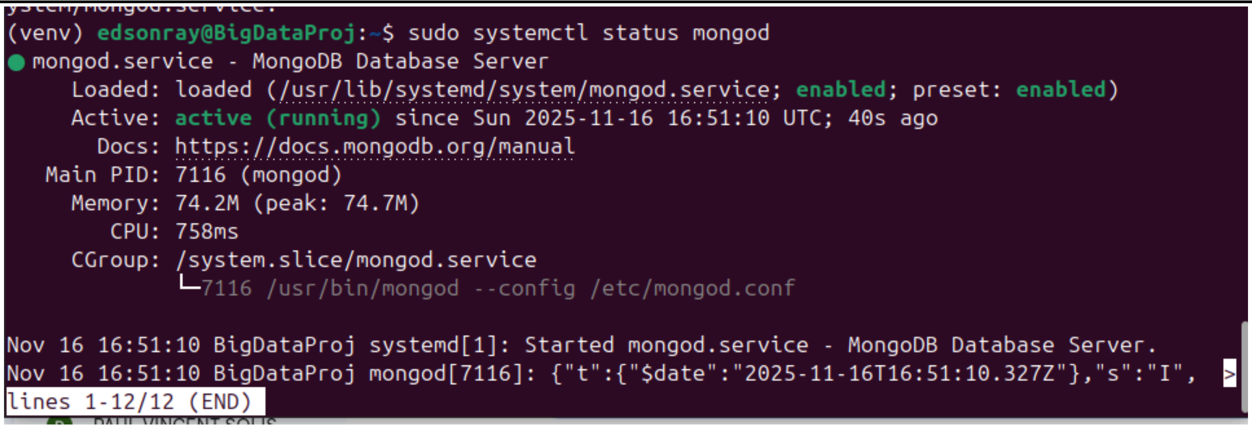
```
(venv) edsonray@BigDataProj:~$ sudo apt-get install -y mongodb-org
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  mongodb-database-tools mongodb-mongosh mongodb-org-database
  mongodb-org-database-tools-extra mongodb-org-mongos mongodb-org-server mongodb-org-shell
  mongodb-org-tools
The following NEW packages will be installed:
  mongodb-database-tools mongodb-mongosh mongodb-org mongodb-org-database
  mongodb-org-database-tools-extra mongodb-org-mongos mongodb-org-server mongodb-org-shell
  mongodb-org-tools
0 upgraded, 9 newly installed, 0 to remove and 80 not upgraded.
Need to get 179 MB of archives.
After this operation, 602 MB of additional disk space will be used.
Get:1 https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0/multiverse amd64 mongodb-databa
se-tools amd64 100.13.0 [58.9 MB]
13% [1 mongodb-database-tools 28.5 MB/58.9 MB 48%]
```

Installing mongodb in linux terminal



```
(venv) edsonray@BigDataProj:~$ sudo systemctl start mongod
(venv) edsonray@BigDataProj:~$ sudo systemctl enable mongod
Created symlink /etc/systemd/system/multi-user.target.wants/mongod.service → /usr/lib/systemd/s
ystem/mongod.service.
(venv) edsonray@BigDataProj:~$
```

Start and enable mongodb

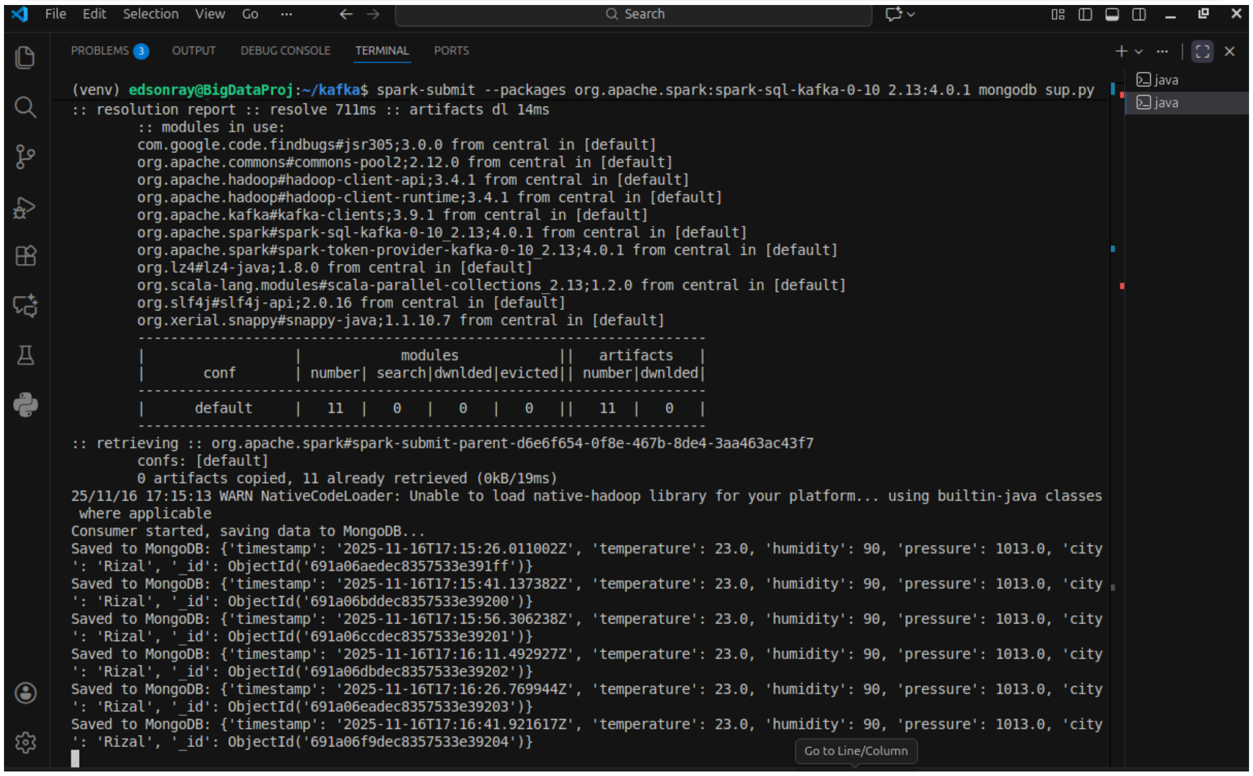


```
(venv) edsonray@BigDataProj:~$ sudo systemctl status mongod
● mongod.service - MongoDB Database Server
     Loaded: loaded (/usr/lib/systemd/system/mongod.service; enabled; preset: enabled)
     Active: active (running) since Sun 2025-11-16 16:51:10 UTC; 40s ago
       Docs: https://docs.mongodb.org/manual
   Main PID: 7116 (mongod)
     Memory: 74.2M (peak: 74.7M)
        CPU: 758ms
     CGroup: /system.slice/mongod.service
             └─7116 /usr/bin/mongod --config /etc/mongod.conf

Nov 16 16:51:10 BigDataProj systemd[1]: Started mongod.service - MongoDB Database Server.
Nov 16 16:51:10 BigDataProj mongod[7116]: {"t":{"$date":"2025-11-16T16:51:10.327Z"},"s":"I",  >
lines 1-12/12 (END)
```

Verify if mongodb successfully running

Python

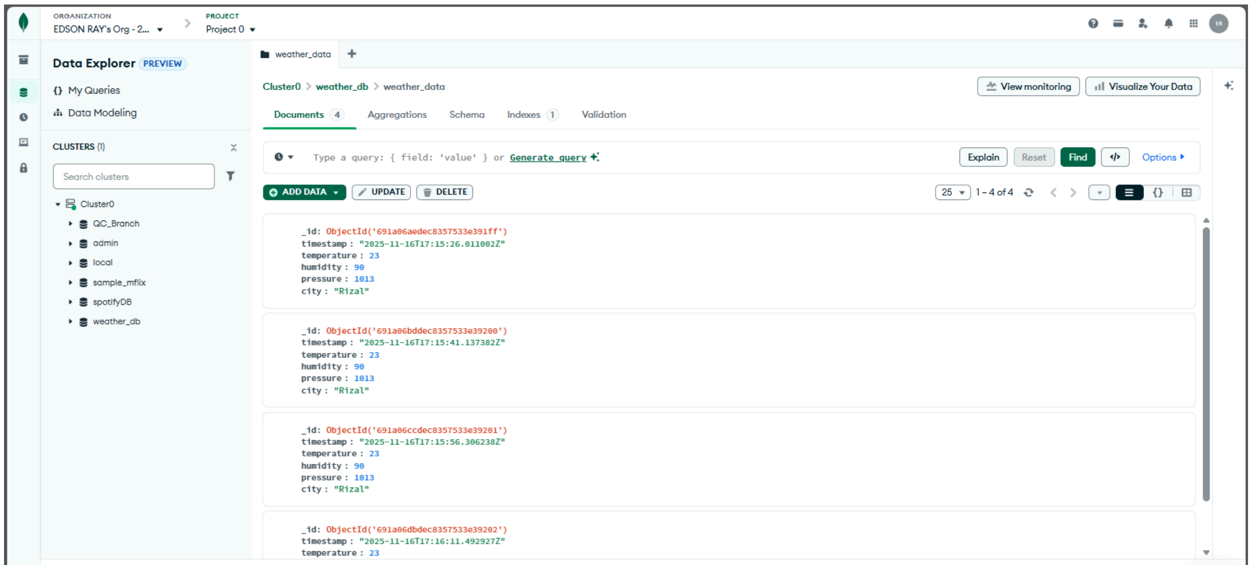
```python
# mongodb_sup.py
from kafka import KafkaConsumer
from pymongo import MongoClient
import json

# -------------------------
# CONFIGURATION
# -------------------------
KAFKA_BROKER = 'localhost:9092'
TOPIC = 'weather_topic'
MONGO_URI = # Replace with your mongodb uri
DB_NAME = 'weather_db'                          # Database name in MongoDB
COLLECTION_NAME = 'weather_data'                # Collection inside DB

# MongoDB connection
client = MongoClient(MONGO_URI)
db = client[DB_NAME]
collection = db[COLLECTION_NAME]

# Kafka consumer
consumer = KafkaConsumer(
    TOPIC,
    bootstrap_servers=[KAFKA_BROKER],
    value_deserializer=lambda m: json.loads(m.decode('utf-8')),
    auto_offset_reset='latest',
    enable_auto_commit=True
)

print("Consumer started, saving data to MongoDB...")

for message in consumer:
    data = message.value
    collection.insert_one(data)
    print("Saved to MongoDB:", data)
```



```
(venv) edsonray@BigDataProj:~/kafka$ spark-submit --packages org.apache.spark:spark-sql-kafka-0-10 2.13:4.0.1 mongodb sup.py
:: resolution report :: resolve 711ms :: artifacts dl 14ms
        :: modules in use:
        com.google.code.findbugs#jsr305;3.0.0 from central in [default]
        org.apache.commons#commons-pool2;2.12.0 from central in [default]
        org.apache.hadoop#hadoop-client-api;3.4.1 from central in [default]
        org.apache.hadoop#hadoop-client-runtime;3.4.1 from central in [default]
        org.apache.kafka#kafka-clients;3.9.1 from central in [default]
        org.apache.spark#spark-sql-kafka-0-10_2.13;4.0.1 from central in [default]
        org.apache.spark#spark-token-provider-kafka-0-10_2.13;4.0.1 from central in [default]
        org.lz4#lz4-java;1.8.0 from central in [default]
        org.scala-lang.modules#scala-parallel-collections_2.13;1.2.0 from central in [default]
        org.slf4j#slf4j-api;2.0.16 from central in [default]
        org.xerial.snappy#snappy-java;1.1.10.7 from central in [default]
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   11  |   0   |   0   |   0   ||   11  |   0   |
        ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent-d6e6f654-0f8e-467b-8de4-3aa463ac43f7
        confs: [default]
        0 artifacts copied, 11 already retrieved (0kB/19ms)
25/11/16 17:15:13 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes
 where applicable
Consumer started, saving data to MongoDB...
Saved to MongoDB: {'timestamp': '2025-11-16T17:15:26.011002Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city
': 'Rizal', '_id': ObjectId('691a06aedec8357533e391ff')}
Saved to MongoDB: {'timestamp': '2025-11-16T17:15:41.137382Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city
': 'Rizal', '_id': ObjectId('691a06bddec8357533e39200')}
Saved to MongoDB: {'timestamp': '2025-11-16T17:15:56.306238Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city
': 'Rizal', '_id': ObjectId('691a06ccdec8357533e39201')}
Saved to MongoDB: {'timestamp': '2025-11-16T17:16:11.492927Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city
': 'Rizal', '_id': ObjectId('691a06dbdec8357533e39202')}
Saved to MongoDB: {'timestamp': '2025-11-16T17:16:26.769944Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city
': 'Rizal', '_id': ObjectId('691a06eadec8357533e39203')}
Saved to MongoDB: {'timestamp': '2025-11-16T17:16:41.921617Z', 'temperature': 23.0, 'humidity': 90, 'pressure': 1013.0, 'city
': 'Rizal', '_id': ObjectId('691a06f9dec8357533e39204')}
```

Code to run the mongodb consumer:

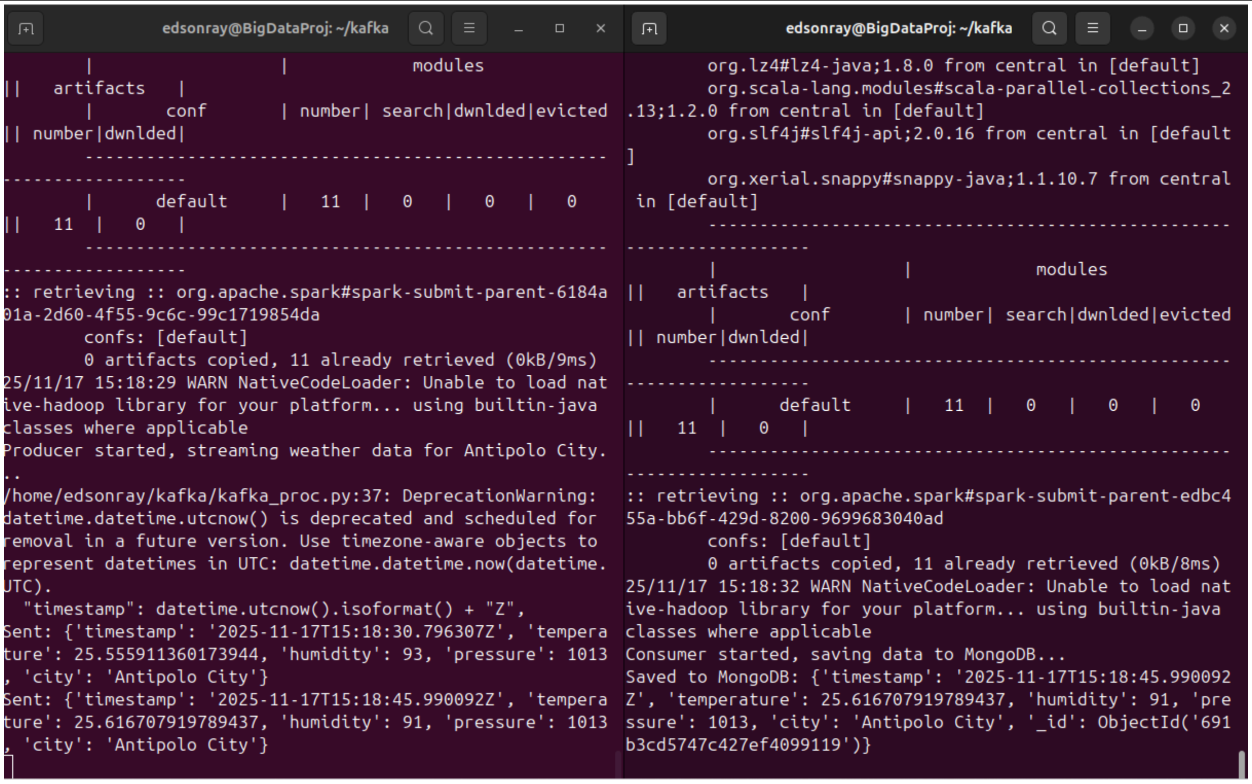
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.13:4.0.1 mongodb_sup.py



```
_id: ObjectId('691a06aedec8357533e391ff')
timestamp : "2025-11-16T17:15:26.011002Z"
temperature : 23
humidity : 90
pressure : 1013
city : "Rizal"

_id: ObjectId('691a06bddec8357533e39200')
timestamp : "2025-11-16T17:15:41.137382Z"
temperature : 23
humidity : 90
pressure : 1013
city : "Rizal"

_id: ObjectId('691a06ccdec8357533e39201')
timestamp : "2025-11-16T17:15:56.306238Z"
temperature : 23
humidity : 90
pressure : 1013
city : "Rizal"

_id: ObjectId('691a06dbdec8357533e39202')
timestamp : "2025-11-16T17:16:11.492927Z"
temperature : 23
```

Output in mongodb

Streamlit



edsonray@BigDataProj: ~/kafka

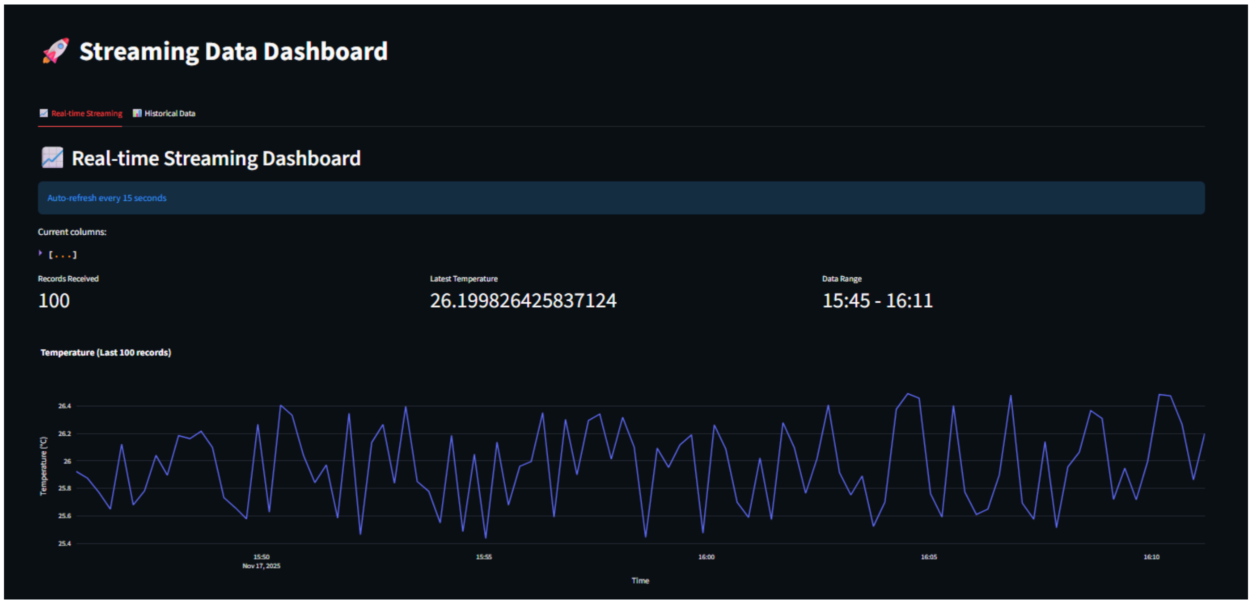
```
        |                  |            modules
||   artifacts   |
        |       conf       | number|  search|dwnlded|evicted
|| number|dwnlded|
        ---------------------------------------------------
-----------------
        |      default     |   11  |   0   |   0   |   0
||   11  |   0   |
        ---------------------------------------------------
-----------------
:: retrieving :: org.apache.spark#spark-submit-parent-6184a
01a-2d60-4f55-9c6c-99c1719854da
        confs: [default]
        0 artifacts copied, 11 already retrieved (0kB/9ms)
25/11/17 15:18:29 WARN NativeCodeLoader: Unable to load nat
ive-hadoop library for your platform... using builtin-java
classes where applicable
Producer started, streaming weather data for Antipolo City.
..
/home/edsonray/kafka/kafka_proc.py:37: DeprecationWarning:
datetime.datetime.utcnow() is deprecated and scheduled for
removal in a future version. Use timezone-aware objects to
represent datetimes in UTC: datetime.datetime.now(datetime.
UTC).
  "timestamp": datetime.utcnow().isoformat() + "Z",
Sent: {'timestamp': '2025-11-17T15:18:30.796307Z', 'tempera
ture': 25.555911360173944, 'humidity': 93, 'pressure': 1013
, 'city': 'Antipolo City'}
Sent: {'timestamp': '2025-11-17T15:18:45.990092Z', 'tempera
ture': 25.616707919789437, 'humidity': 91, 'pressure': 1013
, 'city': 'Antipolo City'}
```

edsonray@BigDataProj: ~/kafka

```
        org.lz4#lz4-java;1.8.0 from central in [default]
        org.scala-lang.modules#scala-parallel-collections_2
.13;1.2.0 from central in [default]
        org.slf4j#slf4j-api;2.0.16 from central in [default
]
        org.xerial.snappy#snappy-java;1.1.10.7 from central
 in [default]
        ---------------------------------------------------
-----------------
        |                  |            modules
||   artifacts   |
        |       conf       | number|  search|dwnlded|evicted
|| number|dwnlded|
        ---------------------------------------------------
-----------------
        |      default     |   11  |   0   |   0   |   0
||   11  |   0   |
        ---------------------------------------------------
-----------------
:: retrieving :: org.apache.spark#spark-submit-parent-edbc4
55a-bb6f-429d-8200-9699683040ad
        confs: [default]
        0 artifacts copied, 11 already retrieved (0kB/8ms)
25/11/17 15:18:32 WARN NativeCodeLoader: Unable to load nat
ive-hadoop library for your platform... using builtin-java
classes where applicable
Consumer started, saving data to MongoDB...
Saved to MongoDB: {'timestamp': '2025-11-17T15:18:45.990092
Z', 'temperature': 25.616707919789437, 'humidity': 91, 'pre
ssure': 1013, 'city': 'Antipolo City', '_id': ObjectId('691
b3cd5747c427ef4099119')}
```

Running kafka_proc (procedure) and mongodb_sup (consumer) at the same time

Dashboard.py



🚀 Streaming Data Dashboard

📈 Real-time Streaming | 📊 Historical Data

📈 Real-time Streaming Dashboard

Auto-refresh every 15 seconds

Current columns:

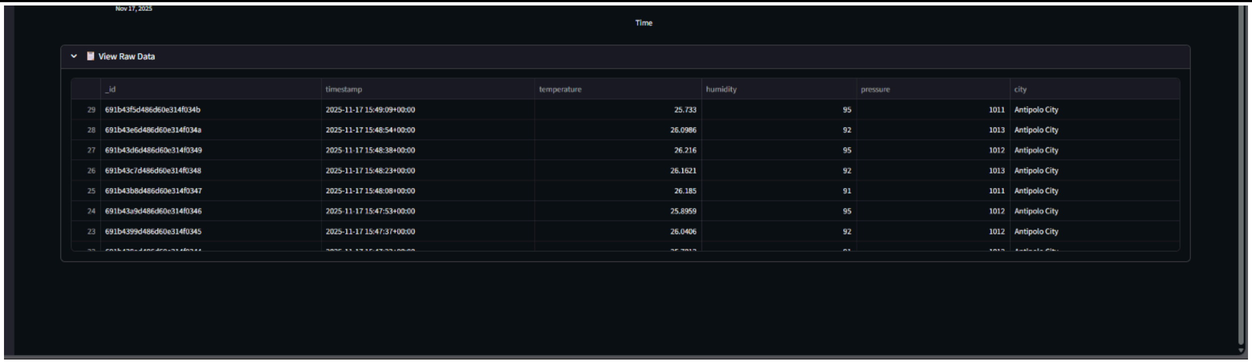
▸ [...]

Records Received
100

Latest Temperature
26.199826425837124

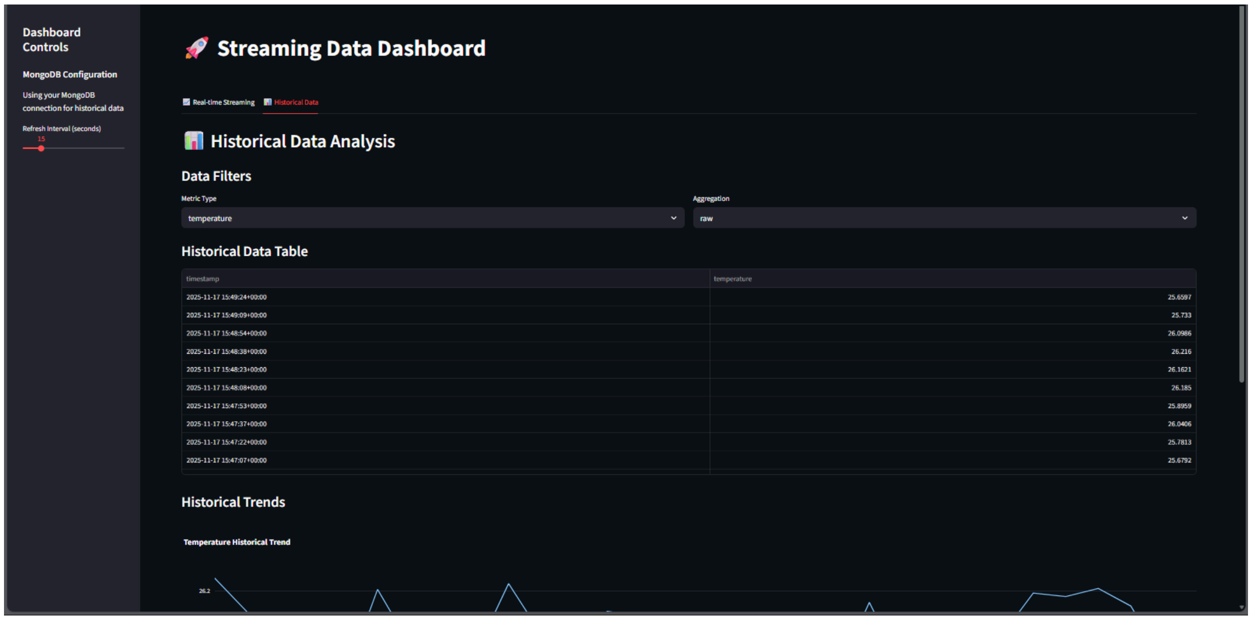
Data Range
15:45 - 16:11

Temperature (Last 100 records)



	_id	timestamp	temperature	humidity	pressure	city
29	691b43f5d486d60e314f034b	2025-11-17 15:49:09+00:00	25.733	95	1011	Antipolo City
28	691b43e6d486d60e314f034a	2025-11-17 15:48:54+00:00	26.0986	92	1013	Antipolo City
27	691b43d6d486d60e314f0349	2025-11-17 15:48:38+00:00	26.216	95	1012	Antipolo City
26	691b43c7d486d60e314f0348	2025-11-17 15:48:23+00:00	26.1621	92	1013	Antipolo City
25	691b43b8d486d60e314f0347	2025-11-17 15:48:08+00:00	26.185	91	1011	Antipolo City
24	691b43a9d486d60e314f0346	2025-11-17 15:47:53+00:00	25.8959	95	1012	Antipolo City
23	691b4399d486d60e314f0345	2025-11-17 15:47:37+00:00	26.0406	92	1012	Antipolo City

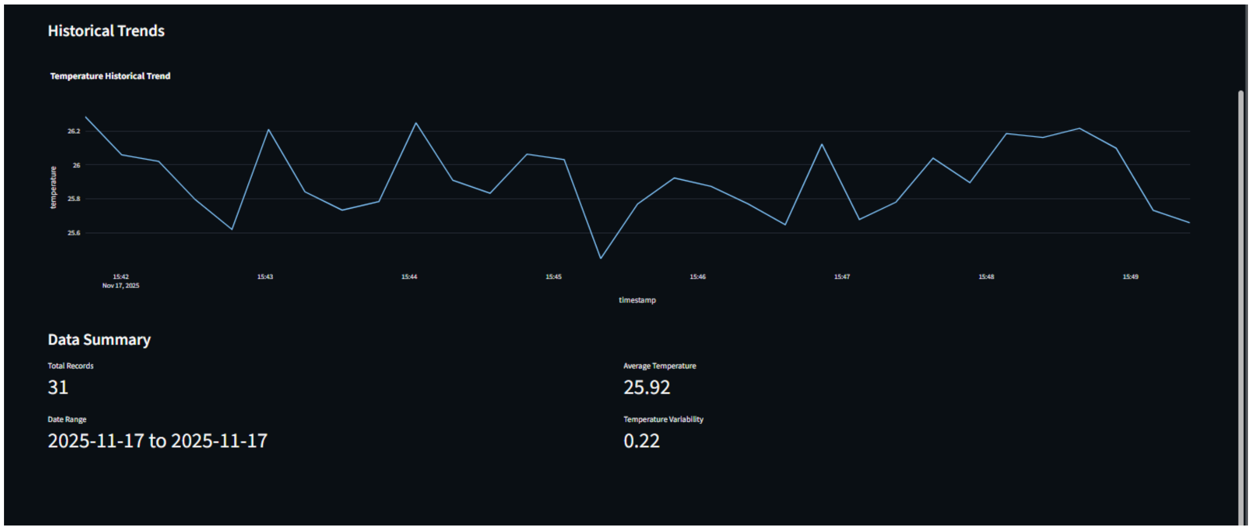
Historical Data



Metric Type: temperature

Aggregation: raw

timestamp	temperature
2025-11-17 15:49:24+00:00	25.6597
2025-11-17 15:49:09+00:00	25.733
2025-11-17 15:48:54+00:00	26.0986
2025-11-17 15:48:38+00:00	26.216
2025-11-17 15:48:23+00:00	26.1621
2025-11-17 15:48:08+00:00	26.185
2025-11-17 15:47:53+00:00	25.8959
2025-11-17 15:47:37+00:00	26.0406
2025-11-17 15:47:22+00:00	25.7813
2025-11-17 15:47:07+00:00	25.6792



Total Records
31

Average Temperature
25.92

Date Range
2025-11-17 to 2025-11-17

Temperature Variability
0.22

Python

```python
# dashboard.py
import streamlit as st
from pymongo import MongoClient
import pandas as pd
import time

# -------------------------
# CONFIGURATION
# -------------------------
MONGO_URI = # Replace with your mongodb uri
DB_NAME = 'weather_db'
COLLECTION_NAME = 'weather_data'

# Connect to MongoDB
client = MongoClient(MONGO_URI)
db = client[DB_NAME]
collection = db[COLLECTION_NAME]

st.title("Live Weather Dashboard - Rizal")
REFRESH_INTERVAL = 15  # seconds

data_placeholder = st.empty()

def parse_timestamp(ts):
    """Convert timestamps to pandas datetime, handling both Unix and ISO
formats."""
    try:
        return pd.to_datetime(float(ts), unit='s')
    except:
        return pd.to_datetime(ts)

while True:
    # Fetch latest data from MongoDB, newest records first
    data_cursor = collection.find().sort("timestamp", -1)
    data_list = list(data_cursor)

    if data_list:
        df = pd.DataFrame(data_list)
        df['timestamp'] = df['timestamp'].apply(parse_timestamp)

        with data_placeholder.container():
            st.subheader("Latest Weather Data")
            st.write(df.head(1))  # Display the newest record
            st.subheader("History")
            st.dataframe(df)      # Show all data
    else:
        with data_placeholder.container():
            st.write("No data available yet.")

    time.sleep(REFRESH_INTERVAL)
```

Python

```python
# app.py
import streamlit as st
import pandas as pd
import plotly.express as px
from datetime import datetime, timedelta
from pymongo import MongoClient
from streamlit_autorefresh import st_autorefresh

# ------ Page configuration ------
st.set_page_config(
    page_title="Streaming Data Dashboard",
    page_icon="\U0001f4ca",
    layout="wide",
    initial_sidebar_state="expanded"
)

# ------ Sidebar Configuration ------
def setup_sidebar():
    st.sidebar.title("Dashboard Controls")
    st.sidebar.subheader("MongoDB Configuration")
    st.sidebar.text("Using your MongoDB connection for historical data")

    return {
        "mongodb_uri": # Replace with your mongodb uri,
        "database": "weather_db",
        "collection": "weather_data"
    }

# ------ Historical Data from MongoDB ------
def query_historical_data(time_range="1h", metrics=None, config=None):
    try:
        client = MongoClient(config["mongodb_uri"])
        db = client.get_database(config["database"])
        collection = db.get_collection(config["collection"])

        query = {}
        if metrics:
            query["metric_type"] = {"$in": metrics}

        data = list(collection.find(query))
        if not data:
            return pd.DataFrame()

        df = pd.DataFrame(data)
        df['timestamp'] = pd.to_datetime(df['timestamp'])
        return df

    except Exception as e:
        st.error(f"Error fetching historical data: {e}")
        return pd.DataFrame()

# ------ Real-time Streaming View ------
def display_real_time_view(refresh_interval, config):
    st.header("\U0001f4c8 Real-time Streaming Dashboard")
    st.info(f"Auto-refresh every {refresh_interval} seconds")
    with st.spinner("Fetching real-time data..."):
        real_time_data = query_historical_data(config=config)
    if not real_time_data.empty:
        st.write("Current columns:", list(real_time_data.columns))  # Debug
line

        col1, col2, col3 = st.columns(3)
        with col1:
            st.metric("Records Received", len(real_time_data))
        with col2:
            if 'value' in real_time_data.columns and not
real_time_data['value'].empty:
                st.metric("Latest Value",
f"{real_time_data['value'].iloc[-1]:.2f}")
            else:
                st.warning("No 'value' column found in real-time data!")
                st.write(real_time_data.head())
        with col3:
            if 'timestamp' in real_time_data.columns and not
real_time_data['timestamp'].empty:
                st.metric(
                    "Data Range",
                    f"{real_time_data['timestamp'].min().strftime('%H:%M')} -
{real_time_data['timestamp'].max().strftime('%H:%M')}"
                )
            else:
                st.warning("No 'timestamp' column in your data!")
        if 'timestamp' in real_time_data.columns and 'value' in
real_time_data.columns:
            fig = px.line(
                real_time_data,
                x='timestamp',
                y='value',
                title=f"Real-time Data Stream (Last {len(real_time_data)}
records)",
                labels={'value': 'Sensor Value', 'timestamp': 'Time'},
                template='plotly_white'
            )
            st.plotly_chart(fig, use_container_width=True)
        with st.expander("\U0001f4cb View Raw Data"):
            # Prepare latest 100 records, sorted, and with 1-based top-down
"No." column
            df_raw = df.sort_values('timestamp',
ascending=False).head(100).sort_values('timestamp').reset_index(drop=True)
```

```python
            # Insert the index/row "No." column as last-in = 100, first-in = 1
            df_raw.insert(0, "No.", range(1, len(df_raw) + 1))

            # Optionally remove Mongo _id if you don't want to show it
            if "_id" in df_raw.columns:
                df_raw = df_raw.drop("_id", axis=1)

            st.dataframe(df_raw, height=300)

# ------ Historical Data View ------
def display_historical_view(config):
    st.header("\U0001f4ca Historical Data Analysis")
    st.subheader("Data Filters")
    col1, col2 = st.columns(2)
    with col1:
        metric_type = st.selectbox(
            "Metric Type",
            ["temperature", "humidity", "pressure", "all"]
        )
    with col2:
        aggregation = st.selectbox(
            "Aggregation",
            ["raw", "hourly", "daily"]
        )

    # Query WITHOUT using 'metric_type' as a filter!
    historical_data = query_historical_data(config=config)

    if not historical_data.empty:
        # Select which metric to plot/show, default to all if 'all' is selected
        show_all = metric_type == "all"
        cols_to_show = ['timestamp']
        if show_all:
            cols_to_show += [col for col in ['temperature', 'humidity',
'pressure'] if col in historical_data.columns]
        else:
            if metric_type in historical_data.columns:
                cols_to_show.append(metric_type)
            else:
                st.warning(f"No data for '{metric_type}' in your records!")
        st.subheader("Historical Data Table")
        st.dataframe(
            historical_data[cols_to_show].sort_values('timestamp',
ascending=False).reset_index(drop=True),
            hide_index=True
        )

        # Plot trend(s)
        if not show_all and metric_type in historical_data.columns:
            st.subheader("Historical Trends")
            fig = px.line(
                historical_data,
                x='timestamp',
                y=metric_type,
                title=f"{metric_type.capitalize()} Historical Trend"
            )
            st.plotly_chart(fig, use_container_width=True)
        elif show_all:
            st.subheader("Historical Trends (Temperature; use other metric
types for specific plots)")
            if 'temperature' in historical_data.columns:
                fig = px.line(
                    historical_data,
                    x='timestamp',
                    y='temperature',
                    title="Temperature Historical Trend"
                )
                st.plotly_chart(fig, use_container_width=True)
        st.subheader("Data Summary")
        col1, col2 = st.columns(2)
        with col1:
            st.metric("Total Records", len(historical_data))
            st.metric(
                "Date Range",
                f"{historical_data['timestamp'].min().strftime('%Y-%m-%d')} to
{historical_data['timestamp'].max().strftime('%Y-%m-%d')}"
            )
        with col2:
            if not show_all and metric_type in historical_data.columns:
                st.metric(
                    f"Average {metric_type.capitalize()}",
                    f"{historical_data[metric_type].mean():.2f}"
                )
                st.metric(
                    f"{metric_type.capitalize()} Variability",
                    f"{historical_data[metric_type].std():.2f}"
                )
    else:
        st.warning("No historical data available")


# Graph
def display_real_time_view(refresh_interval, config):
    st.header("\U0001f4c8 Real-time Streaming Dashboard")
    st.info(f"Auto-refresh every {refresh_interval} seconds")
    with st.spinner("Fetching real-time data..."):
        df = query_historical_data(config=config)

    if not df.empty:
        # Only last 100 records, sorted by timestamp
        df = df.sort_values('timestamp',
ascending=False).head(100).sort_values('timestamp')

        st.write("Current columns:", list(df.columns))  # Debug line

        col1, col2, col3 = st.columns(3)
        with col1:
            st.metric("Records Received", len(df))
        with col2:
            if "temperature" in df.columns and not df["temperature"].empty:
                st.metric("Latest Temperature",
f"{df['temperature'].iloc[-1]}")
            else:
                st.warning("No 'temperature' column found in real-time data!")
                st.write(df.head())
        with col3:
            if "timestamp" in df.columns and not df["timestamp"].empty:
                st.metric(
                    "Data Range",
                    f"{df['timestamp'].min().strftime('%H:%M')} -
{df['timestamp'].max().strftime('%H:%M')}"
                )
            else:
                st.warning("No 'timestamp' column in your data!")

        # Plot temperature graph for last 100 records
        if "timestamp" in df.columns and "temperature" in df.columns:
            fig = px.line(
                df,
                x='timestamp',
                y='temperature',
                title=f"Temperature (Last {len(df)} records)",
                labels={'temperature': 'Temperature (°C)', 'timestamp':
'Time'},
                template='plotly_white'
            )
            st.plotly_chart(fig, use_container_width=True)

        with st.expander("\U0001f4cb View Raw Data"):
            st.dataframe(df.sort_values('timestamp', ascending=False),
height=300)
    else:
        st.warning("No data available for real-time view!")


# ------ Main App ------
def main():
    st.title("\U0001f680 Streaming Data Dashboard")
    config = setup_sidebar()
    refresh_interval = st.sidebar.slider(
        "Refresh Interval (seconds)",
        min_value=5,
        max_value=60,
        value=15
    )
    st_autorefresh(interval=refresh_interval * 1000, key="auto_refresh")
    tab1, tab2 = st.tabs(["\U0001f4c8 Real-time Streaming", "\U0001f4ca
Historical Data"])
    with tab1:
        display_real_time_view(refresh_interval, config)
    with tab2:
        display_historical_view(config)

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

Pipeline Overview

1. Kafka
    - **Role**: Central message broker for real-time streaming data.
    - **Where**: Kafka serves as the bridge between your producer (data ingester/API client) and consumers (Spark, database loader).
    - **How Used:** PySpark data producer fetches current weather data (from OpenWeatherMap API) and publishes each reading to a Kafka topic. This allows decoupling of data ingestion from downstream consumers and supports real-time, fault-tolerant streaming.

2. **PySpark (Spark Structured Streaming)**
    - **Role**: Distributed, scalable stream processing and transformation engine.
    - **Where**: Spark consumes data directly from the Kafka topic using its native Kafka connectors (via Spark Structured Streaming).
    - **How Used**: A Spark job often written in PySpark subscribes to the Kafka topic. It can aggregate (e.g., hourly/daily means), transform, or process the records as they arrive in real time. You may use code like:

```python
python
spark.readStream.format("kafka")
     .option("kafka.bootstrap.servers", "localhost:9092")
     .option("subscribe", "weather_topic")
     .load()
```

    - Spark can then write the transformed/aggregated results to MongoDB or another sink.

3. **MongoDB**
   - **Role**: Scalable NoSQL storage for streaming and historical data.
   - **Where**: MongoDB receives the cleaned or raw streaming data from either a lightweight Kafka consumer or a Spark streaming job.
   - **How Used**: Data from Kafka, is written into a MongoDB collection for persistence, querying, and dashboard visualization. Your Streamlit dashboard fetches both real-time and historical data from MongoDB for user display and analytics.

Component	Role	Technology (Software used)	Description/Usage
PySpark	Stream processing	PySpark	Aggregates and transforms Kafka data streams
Kafka	Streaming broker	Apache Kafka	Buffers/streams JSON data between producer and consumers
MongoDB	Storage/analytics	MongoDB Atlas	Stores results for analytics/historical dashboards
Producer	Data ingestion/API	Python, kafka-python	Fetch API data; pushes to Kafka topic
Consumer	Stream sink/storage loader	Python (kafka-python)	Subscribes to Kafka topic, parses messages, writes to MongoDB

Conclusion

I therefore conclude that this final project demonstrates an effective implementation of a real-time big data streaming application pipeline combining PySpark, Kafka, and MongoDB with a user-friendly Streamlit dashboard for visualization. Leveraging the OpenWeatherMap API as the data source, the pipeline ingests live weather data, streams it through Apache Kafka for scalable and fault-tolerant message handling, and employs a Kafka consumer to persist the data efficiently into MongoDB. PySpark provides a powerful framework for potential stream processing and analytics, while MongoDB serves as a flexible, high-performance storage solution for both real-time and historical data. The data dashboard effectively bridges the gap between complex streaming data systems and end-user insights, offering live monitoring and historical trend analysis with rich visualizations. It highlights the significance of integrating modern big data tools to handle continuous data inflow, enable low-latency analysis, and support scalable, maintainable data-driven applications. Overall, this project not only fulfills the academic requirements for demonstrating mastery over big data streaming technologies but also establishes a versatile template that can be extended for real-world environmental monitoring and similar IoT-driven analytics systems.