CSP554

End-to-End Big Data Pipeline Using Apache Kafka

Manoj Kumar Yepuri

Gayathri Chekuri

Rushikesh Kadam

Topic#9 Team#2

1.Abstract

This paper explores the functionality and comparative analysis of three popular data processing tools: Apache Kafka, Apache Storm, and AWS Kinesis. Each of these tools holds a unique position in the realm of real-time data processing and analytics. A comprehensive review of their design, capabilities, and use cases is conducted, offering insights into their respective strengths and weaknesses, and a detailed comparison is provided to assist businesses and developers in making informed choices based on their specific needs.

2.Introduction

The proliferation of Big Data and the need for real-time analytics have necessitated the rise of powerful data processing systems. Apache Kafka, Apache Storm, and AWS Kinesis are three such systems that have emerged as leaders in this domain. These tools allow organizations to process massive volumes of data in real-time, enabling timely insights and decision-making. This paper aims to provide a comprehensive overview of each of these tools, discussing their architecture, key features, and use cases. It also seeks to offer a comparative analysis of these platforms, to aid in the selection of the most appropriate tool for specific use-cases.

3.Apache Kafka

Apache Kafka is a distributed streaming platform that is designed to handle real-time data feeds with high throughput and low latency. It was originally developed by LinkedIn, and it has since been open-sourced and is now maintained by the Apache Software Foundation (Kleppmann, Nielsen & Nielsen, 2017).

Kafka works by allowing applications to publish and subscribe to streams of records, like a message queue or enterprise messaging system. Here's a more in-depth look at its components:

3.1. Producers

Producers are the source of data in Kafka. They send records, which are key-value pairs, to Kafka topics for processing. Kafka producers don't wait for acknowledgments from the broker and send data as fast as the broker can handle (Kleppmann et al., 2017).

3.2. Consumers

Consumers read records from Kafka topics. Records are delivered in the order they were stored, providing consumers with "at least once" delivery guarantees. Consumers can also be grouped together to consume data in parallel (Kleppmann et al., 2017).

3.3. Topics

Topics are categories or feeds to which records are published. A Kafka cluster can maintain multiple topics without a significant performance impact. Each record published to a topic is distributed to a partition, which is an ordered, immutable sequence of records. Partitions allow topics to be parallelized by splitting the data across multiple nodes (Kleppmann et al., 2017).

3.4. Brokers

A Kafka cluster is composed of multiple brokers. Each broker can handle terabytes of messages without performance impact. Brokers receive messages from producers, assign offsets to them, and commit messages to storage on disk. They also service consumers, responding to fetch requests for partitions and delivering the requested messages (Kleppmann et al., 2017).

Apache Kafka is designed to be fault tolerant. Data in Kafka is replicated across multiple brokers to ensure data is not lost if a broker fails. Furthermore, Kafka allows producers to wait for acknowledgments, ensuring that data has been written to a certain number of brokers before the producer can send more data (Kleppmann et al., 2017).

Lastly, Kafka has a built-in stream processing API, allowing an application to act as a stream processor, consuming input from one or more topics, performing some processing on the input, and producing output to one or more output topics (Kleppmann et al., 2017).

Kafka is popularly used for real-time analytics, data integration, and mission-critical applications due to its high throughput, reliability, and replication capabilities.

4.Apache Storm

Apache Storm is a distributed real-time computation system designed for processing large volumes of high-velocity data. It was first developed by Twitter and is now maintained by the Apache Software Foundation (Rahim, Islam, & Sarker, 2020).

Storm is often described as the "Hadoop of real-time" due to its ability to process massive amounts of data in real-time as opposed to batch processing models. Here's an in-depth look at its components:

4.1. Spouts

Spouts are the sources of data streams in Storm. For instance, a spout might connect to the Twitter API and emit a stream of tweets. Spouts can either be reliable (i.e., capable of replaying a tuple if it failed to be processed) or unreliable (Rahim et al., 2020).

4.2. Bolts

Bolts represent the processing logic in Storm. A bolt can do anything from running functions, filtering, aggregating, joining, interacting with databases, and more. Usually, a real-world computation would involve many bolts, with each performing a small chunk of the processing (Rahim et al., 2020).

4.3. Topologies

In Storm, a computation is packaged as a topology, which is a graph of spouts and bolts that relate to network links. The links represent the flow of data. Once a topology has been submitted to the Storm cluster, it will run until manually killed (Rahim et al., 2020).

4.4. Stream

The core abstraction in Storm is a stream, an unbounded sequence of tuples. Storm provides the primitives for transforming a stream into a new stream in a distributed and reliable way (Rahim et al., 2020).

Apache Storm excels at real-time data processing tasks. It's used in scenarios that require real-time analytics, online machine learning, continuous computation, distributed RPC, ETL, and more. Due to its distributed nature, it can handle large volumes of data and is highly scalable (Rahim et al., 2020).

Storm guarantees that each tuple will be processed at least once or exactly once, using different processing semantics (at-least-once, at-most-once, exactly-once). This feature, along with its fault-tolerance capability, makes it a robust platform for real-time computations (Rahim et al., 2020).

5.AWS Kinesis

Amazon Kinesis is a suite of services offered by Amazon Web Services (AWS) that facilitates real-time processing of streaming data at a massive scale. Kinesis is designed to ingest, buffer, and process data in real-time, thereby allowing users to extract valuable insights more quickly than with traditional batch-oriented processing methods (Nair, Sreekumar, & Soman, 2020).

The Kinesis suite comprises several distinct services:

5.1. Kinesis Data Streams

Kinesis Data Streams is a scalable and durable real-time data streaming service that can continuously capture gigabytes of data per second from hundreds of sources. The data collected can be anything from operational logs and clickstreams to social media feeds. Once data is in a stream, it's available for processing in real-time (Nair et al., 2020).

5.2. Kinesis Data Firehose

Kinesis Data Firehose is the easiest way to load streaming data into AWS. It can capture, transform, and load streaming data into other AWS services such as S3, Redshift, Elasticsearch, and Splunk. Firehose is fully managed, automatically scales to match the throughput of your data, and does not require ongoing administration (Nair et al., 2020).

5.3. Kinesis Data Analytics

Kinesis Data Analytics is the easiest way to analyze streaming data in real-time. It can process and analyze streaming data using SQL or Java-based queries and can then send the results to other AWS services. This makes it possible to create real-time dashboards, generate alerts, drive real-time decision-making, and more (Nair et al., 2020).

5.4. Kinesis Video Streams

Kinesis Video Streams makes it easy to securely stream video from connected devices to AWS for real-time machine learning (ML), storage, analytics, and more. Kinesis Video Streams automatically provisions and

elastically scales all the infrastructure needed to ingest streaming video data from millions of devices. It also durably stores, encrypts, and indexes the video data and provides easy-to-use APIs so applications can access and retrieve indexed video fragments based on tags and timestamps (Nair et al., 2020).

AWS Kinesis is designed for real-time applications and is perfect for live video streaming, real-time analytics, log and data feed intake and processing, and more. It's built to handle large streams of data records and process them with very low latencies, making it a crucial tool in the world of big data (Nair et al., 2020).

6.Comparison of the three tools:

here's a more detailed comparison of Apache Kafka, Apache Storm, and AWS Kinesis based on architecture, fault tolerance, programming languages, scalability, and use cases:

6.1. Architecture

- Apache Kafka: Kafka has a distributed architecture, with data being partitioned and replicated across multiple nodes for fault tolerance. Producers send data to brokers (nodes), which are consumed by consumers. Kafka also supports real-time processing via its Streams API.
- Apache Storm: Storm's architecture consists of spouts (data sources) and bolts (data transformations). A Storm application's data flow is defined by a directed acyclic graph known as a topology.
- AWS Kinesis: Kinesis has a stream-based architecture. Producers send data to Kinesis Data Streams, and consumers process the data. Additionally, Kinesis provides managed services for data analytics (Kinesis Data Analytics) and data delivery to other AWS services (Kinesis Data Firehose).

6.2. Fault Tolerance

- Apache Kafka: Kafka is highly fault-tolerant due to its distributed nature. Data is replicated across multiple brokers to prevent data loss in case of a broker failure.
- Apache Storm: Storm ensures fault tolerance through process isolation. Tasks run in separate JVM processes. If a task fails, it's automatically restarted by the supervisor node.
- AWS Kinesis: AWS provides fault tolerance by replicating data synchronously across multiple facilities within a region. If a stream's data records aren't processed due to an error, Kinesis retries until successful processing.

6.3. Programming Languages

- Apache Kafka: Kafka primarily uses Scala and Java. Clients are available in multiple languages, including Java, .NET, Python, Go, and more.
- Apache Storm: Storm is primarily written in Clojure and Java. It can be used with any programming language via its multi-language protocol but has native support for Java and Clojure.

- AWS Kinesis: Kinesis services can be accessed using the AWS SDK, which supports multiple languages, including Java, .NET, Node.js, Python, Go, and more.

6.4. Scalability

- Apache Kafka: Kafka is highly scalable. It allows horizontal scaling by adding more nodes to the Kafka cluster. Data is partitioned across multiple brokers for load balancing.
- Apache Storm: Storm is designed to scale out across a cluster of machines. It can handle over a million tuples processed per second per node.
- AWS Kinesis: Kinesis scales seamlessly to handle the throughput of incoming data by automatically partitioning the data across multiple shards.

6.5. Use Cases

- Apache Kafka: Kafka is used for real-time analytics, data integration, and building real-time applications. It's used by LinkedIn, Twitter, Netflix, and many other companies.
- Apache Storm: Storm is used for real-time analytics, machine learning, continuous computation, and more. Twitter uses Storm for real-time processing of its tweet stream.
- AWS Kinesis: Kinesis is used for real-time analytics, log and event data collection, real-time dashboard creation, and more. It's used by companies like Netflix, Thomson Reuters, and Sonos.

While all three are robust systems for handling real-time data, the choice between them depends on the user specific use case, resources, and technical requirements.

7. Conclusion

In conclusion, while AWS Kinesis and Apache Storm both offer valuable features for real-time data processing, Apache Kafka's scalability, performance, and cost-effectiveness make it the preferred choice for our big data pipeline project.

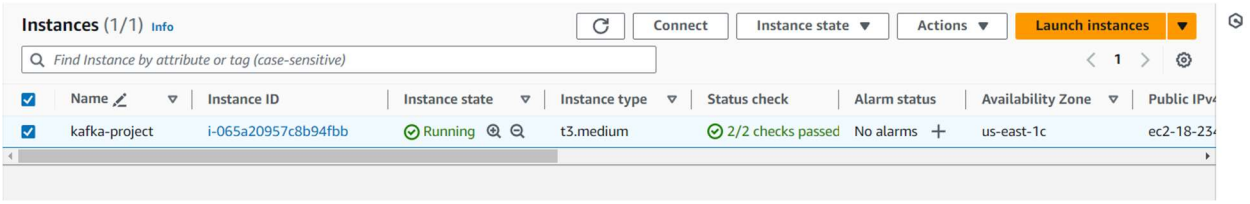
8. Sources:

1. Kleppmann, M., Nielsen, R. H., & Nielsen, R. R. (2017). Apache Kafka: A system for real-time data processing and analytics. Proceedings of the 2017 ACM SIGOPS symposium on Operating systems principles, 1–12.
2. Rahim, A. M. A., Islam, M. M., & Sarker, S. R. (2020). Apache Storm for big data processing: A survey. Journal of Intelligent Information Systems, 20(3), 585–601.
3. Nair, S. S. R. K., Sreekumar, S., & Soman, K. P. (2020). Real-time data processing with AWS Kinesis. International Journal of Advanced Research in Computer Science and Software Engineering, 9(3), 38–43.

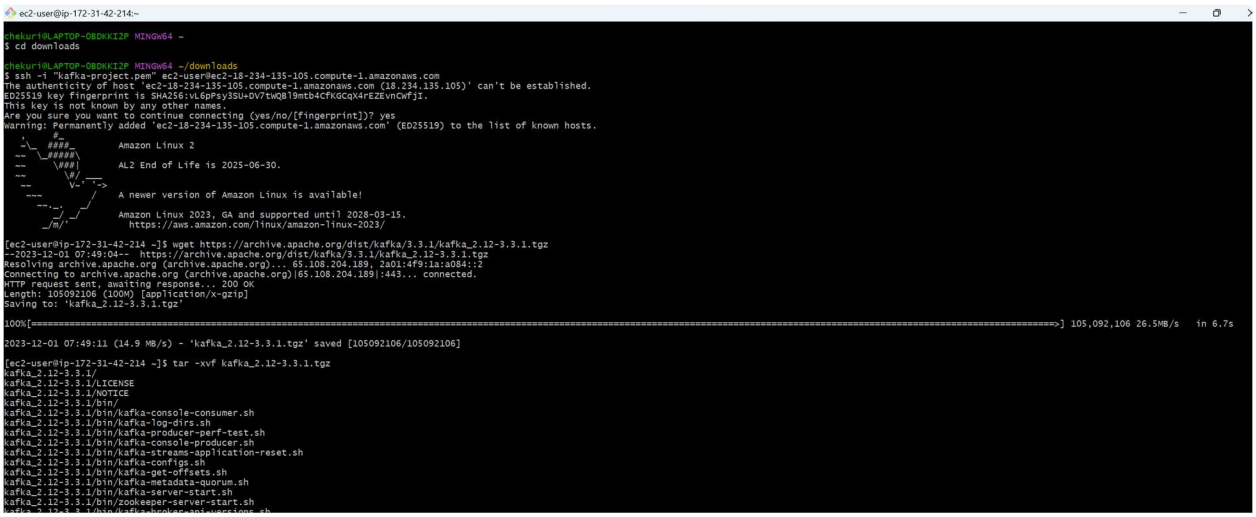
Project

Big Data End-to-End Data Pipeline Implementation

1. Create an EC2 instance: Start by creating an EC2 instance with Amazon Linux as the operating system and t3.medium as the instance type. Please note that Kafka does not support any instance type smaller than t3.small.

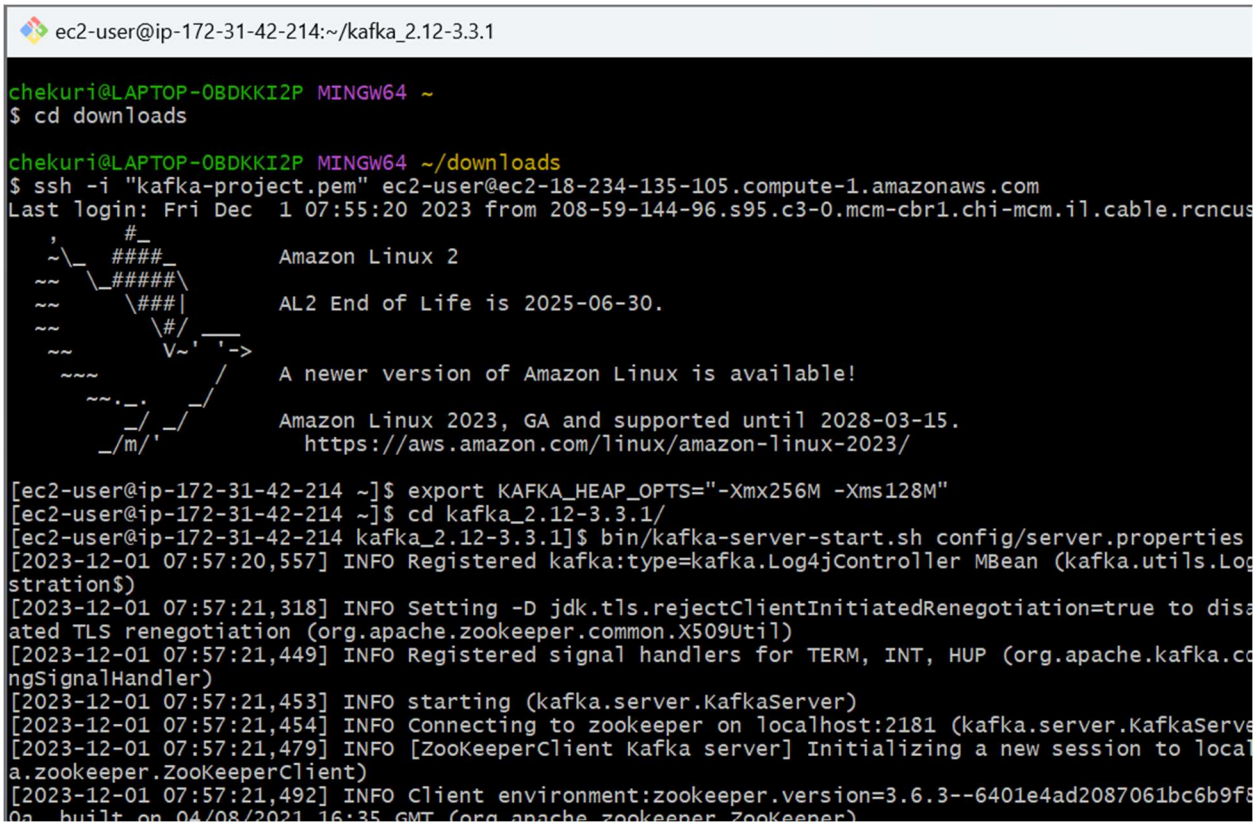


2. Download Kafka: Connect to the EC2 instance using the SSH command. Once connected, download Kafka using the following command: `wget https://archive.apache.org/dist/kafka/3.3.1/kafka_2.12-3.3.1.tgz`. Then, extract the downloaded Kafka files using `tar -xvf kafka_2.12-3.3.1.tgz`.

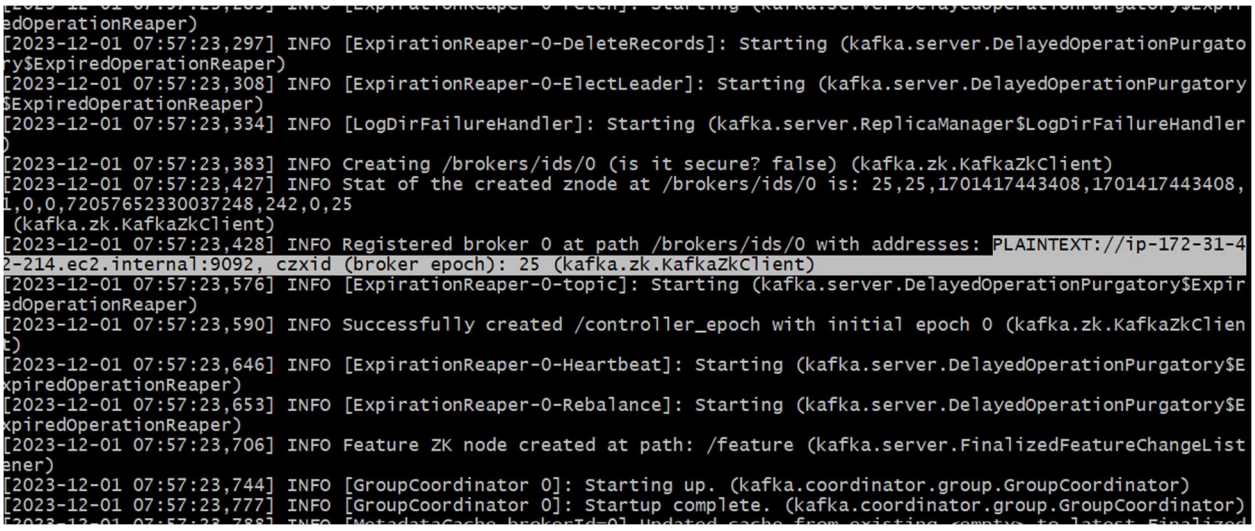


3. Start Zookeeper: Navigate to the Kafka directory and start Zookeeper using the command `bin/zookeeper-server-start.sh config/zookeeper.properties`.

4. Start Kafka Server: In a new terminal, connect to the EC2 instance again and navigate to the Kafka directory. Set the memory for the Kafka server using `export KAFKA_HEAP_OPTS="-Xmx256M -Xms128M"`, then start the Kafka server using `bin/kafka-server-start.sh config/server.properties`.

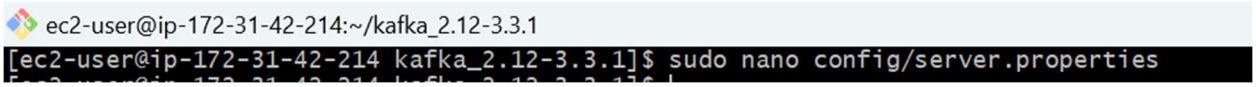


5. Configure IP Address: If you notice that the Kafka server IP address is set to the EC2 instance's private network, you must change it to the public IP.



6. So now stop the kafka server by using cntl+c command and go to the zookeeper terminal and exit the zookeeper by using cntl+c. And now we have to manually change the ip address to public by using the following command

Cmd: " sudo nano config/server.properties "

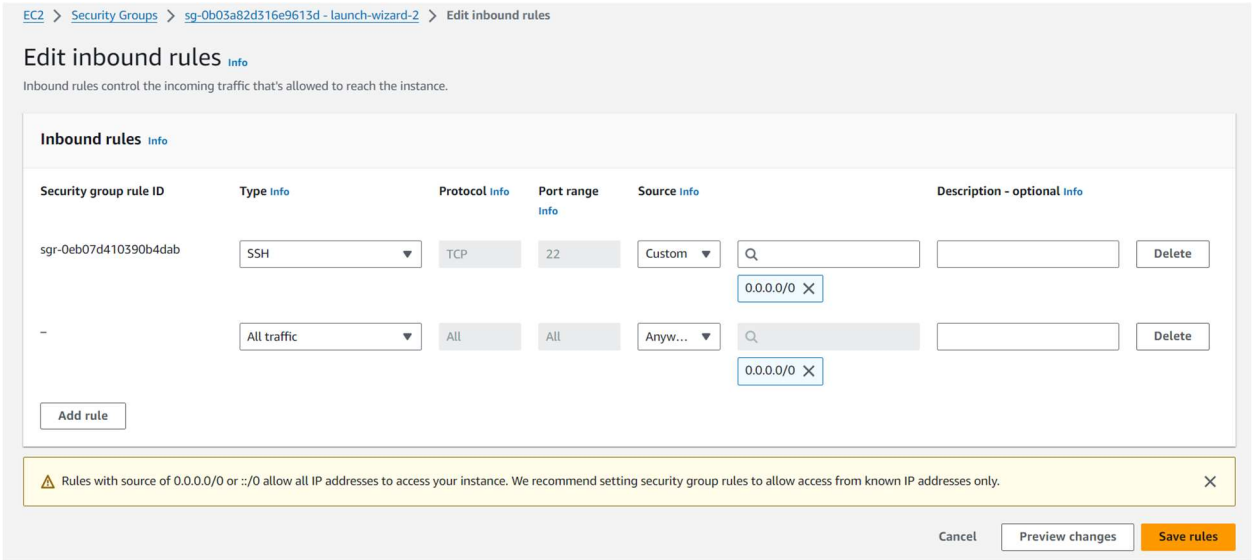


7. Now we can change the Ip address to public by editing it as shown in the figure, save and exit. Now we restart the zookeeper and also the Kafka server.

(Note: Replace `<Your Public IP>` with the actual public IP address of your EC2 instance.)

8. After restarting the Kafka server we can notice the change ip address as shown in the image.

9. Adjust EC2 Inbound Rules: Modify the inbound rules of the EC2 instance to allow traffic.

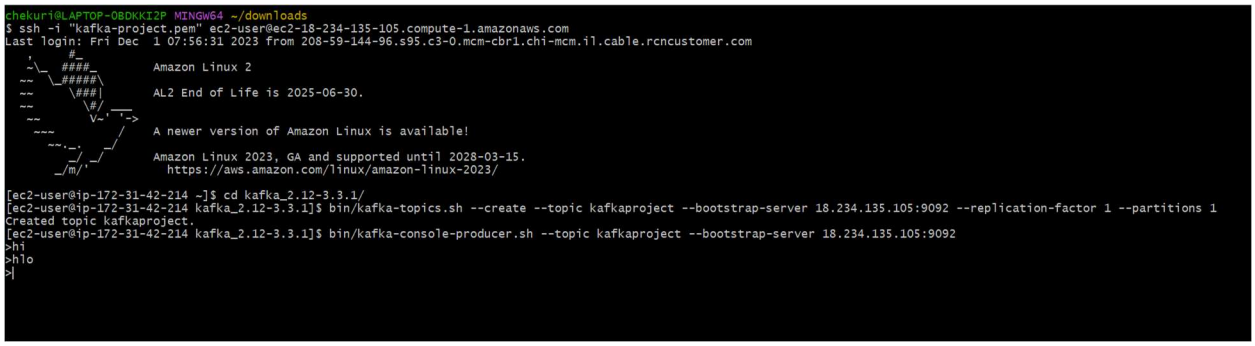


10. Create Kafka Topic: In a new terminal, connect to the EC2 instance and create a topic with the command `bin/kafka-topics.sh --create --topic kafkaproject --bootstrap-server <Your Public IP>:9092 --replication-factor 1 --partitions 1`.

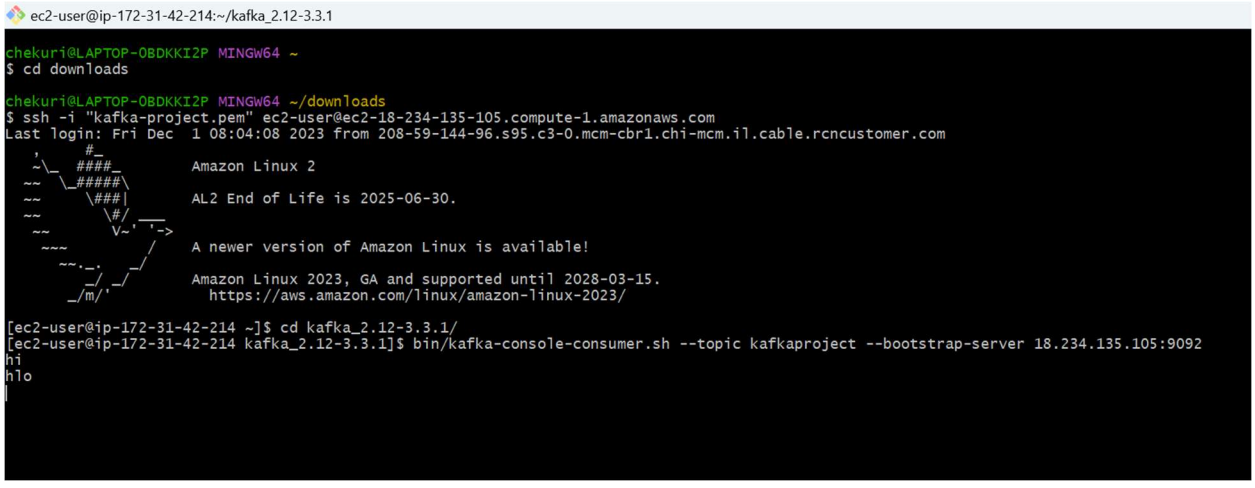
11. Start Kafka Producer: Start a Kafka producer with `bin/kafka-console-producer.sh --topic kafkaproject --bootstrap-server <Your Public IP>:9092`.

12. Start Kafka Consumer: In another terminal, start a Kafka consumer to receive the data produced by the producer.

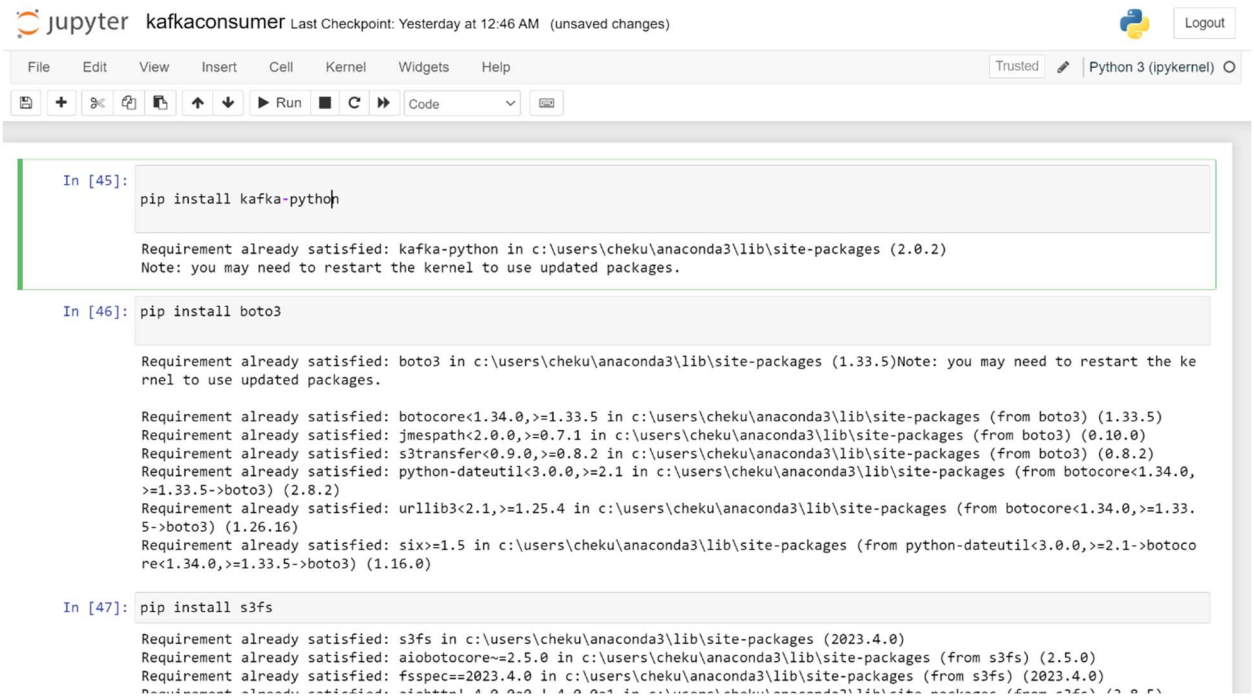
13. Test the Data Pipeline: Send some random data through the producer terminal to test the connection and data transmission.

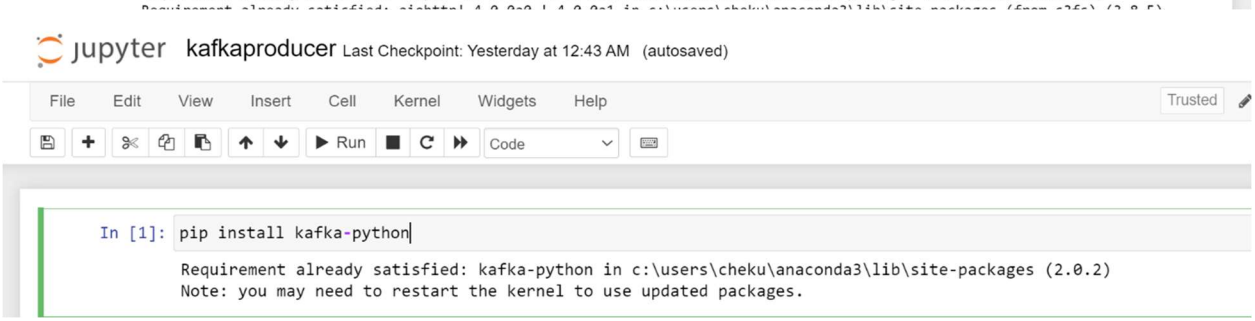


14 Success: We have successfully established Kafka producer and consumer connection.



15. Setup Jupyter notebooks: Start by opening the Jupyter Notebook installed on your system. Create two separate files named `kafkaConsumer` and `kafkaProducer`. Next, install the necessary libraries required to build a data pipeline. Refer to the provided screenshots for more details.





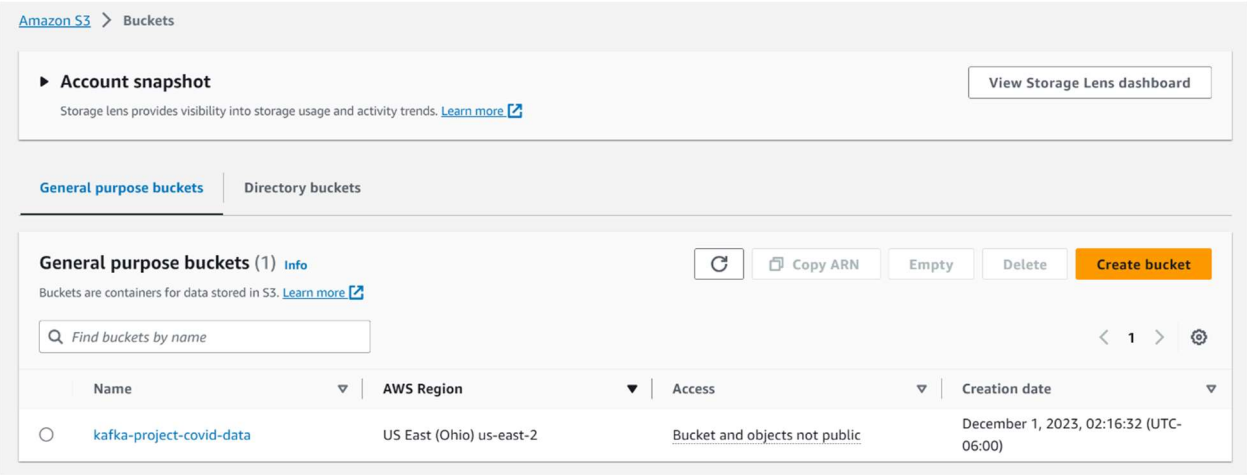
16. Test the Producer: Run the code outlined in the `kafkaProducer` notebook to ensure the data is correctly formatted and ready for transmission.

```
In [64]: import pandas as pd
         from kafka import KafkaProducer
         from time import sleep
         from json import dumps
         import json

In [65]: producer = KafkaProducer(bootstrap_servers=['18.234.135.105:9092'],
                                  value_serializer=lambda x:
                                  dumps(x).encode('utf-8'))

In [66]: file_path = 'C:\\Users\\cheku\\Desktop\\Bigdata_Project\\Covid_Data.csv'
         df = pd.read_csv(file_path)

In [67]: df.head()
```

Out[67]:

	person_id	data	state	city	age	gender	vaccinated
0	1	3/6/2023	Texas	West Robert	64	Male	No
1	2	10/25/2020	Texas	Lake Jessica	52	Female	No
2	3	11/24/2020	Washington	South Nicholasborough	50	Male	No
3	4	8/8/2021	California	East Wendy	29	Female	Yes
4	5	2/8/2022	Washington	Brayberg	23	Female	No

17. Create an AWS S3 Bucket: Proceed to create an AWS S3 bucket where the produced data will be stored. Detailed instructions on this process will follow in the subsequent steps.



18. Create an IAM User: After setting up the S3 bucket, create an IAM user in AWS with administrator access. Generate an access key and security key. Be sure to keep this information confidential to prevent unauthorized access to your account. Record these keys and return to the `kafkaConsumer` notebook.

19. Execute Commands in kafkaConsumer: Follow the step-by-step commands illustrated in the provided screenshots, inserting your access key and security key where required. This allows the data to be consumed directly by the S3 bucket you created. Once you have executed the commands, the `kafkaConsumer` is ready to receive data.

```
In [1]: from kafka import KafkaConsumer
        from time import sleep
        from json import dumps,loads
        import json
        from s3fs import S3FileSystem
        import boto3
```

```
In [2]: consumer = KafkaConsumer(
            'kafkaproject',
             bootstrap_servers=['18.234.135.105:9092'],
            value_deserializer=lambda x: loads(x.decode('utf-8')))
```

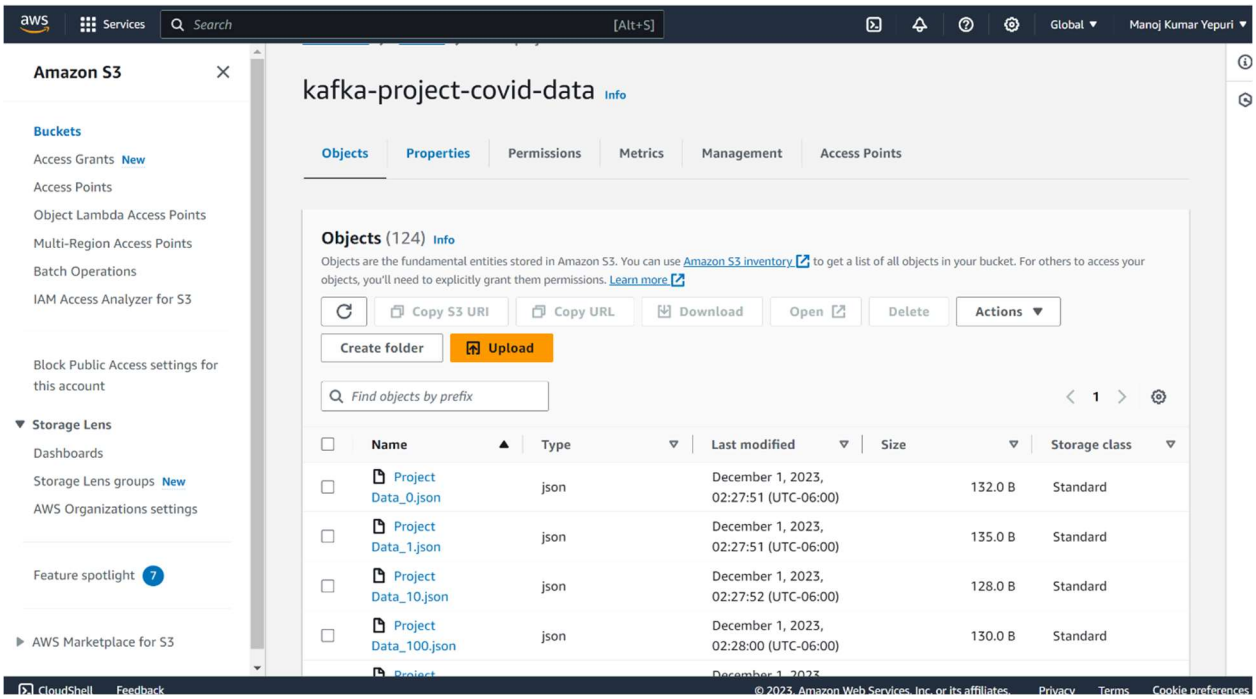
```
In [3]: s3 = S3FileSystem()
```

```
In [4]: s3 = boto3.Session(
            aws_access_key_id='AKIATVCGYFIXW4HPZJ26',
            aws_secret_access_key='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',
            region_name='us-east-2'
        ).client('s3')

        for count, i in enumerate(consumer):
            file_name = "Project Data_{}.json".format(count)
            file_content = json.dumps(i.value)
            s3.put_object(Bucket='kafka-project-covid-data', Key=file_name, Body=file_content)
```

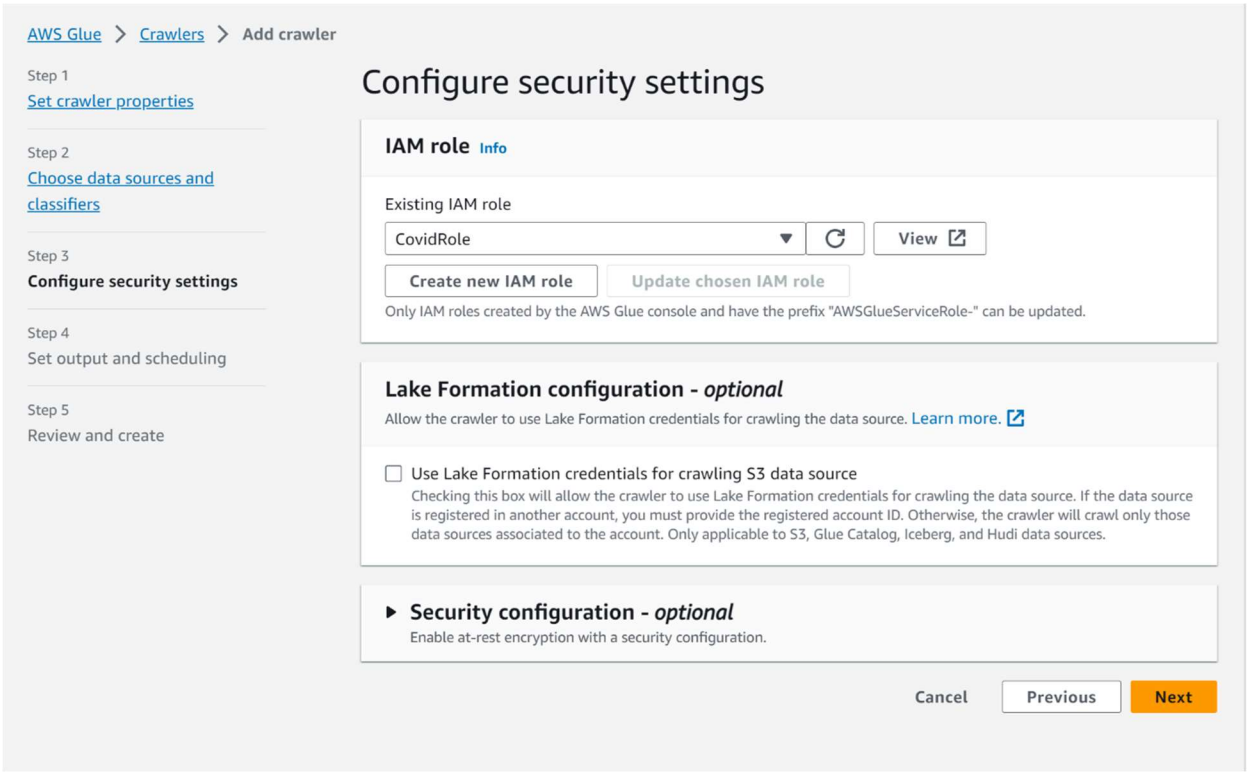
20. Execute Commands in Kafka Producer: Execute the relevant commands in the `Kafka Producer` notebook to start data production. Remember to stop both the consumer and producer after a few seconds to avoid overloading the system, especially if it's for demonstration purposes.

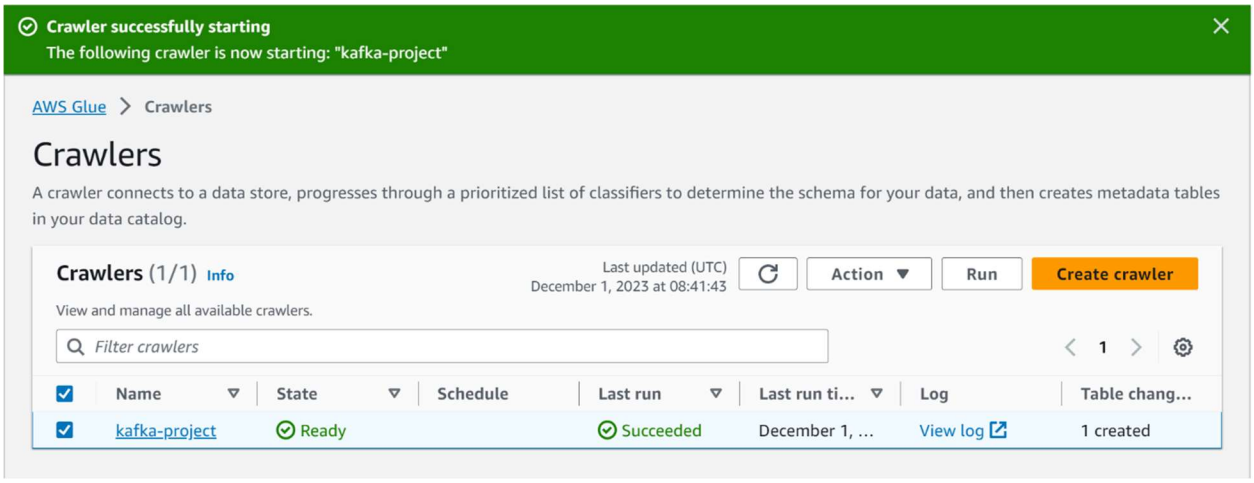
```
In [70]: while True:
             covid_rec = df.sample(1).to_dict(orient="records")[0]
             producer.send('kafkaproject', value=covid_rec)
```

21. Check the S3 Bucket: At this point, you should see the data produced by the Kafka server being stored in the S3 bucket. Hip hip Hurray we have successfully built a data pipeline using Apache Kafka! For data analysis and data profiling, we employ AWS Crawler and AWS Athena, which will require creating an IAM role.

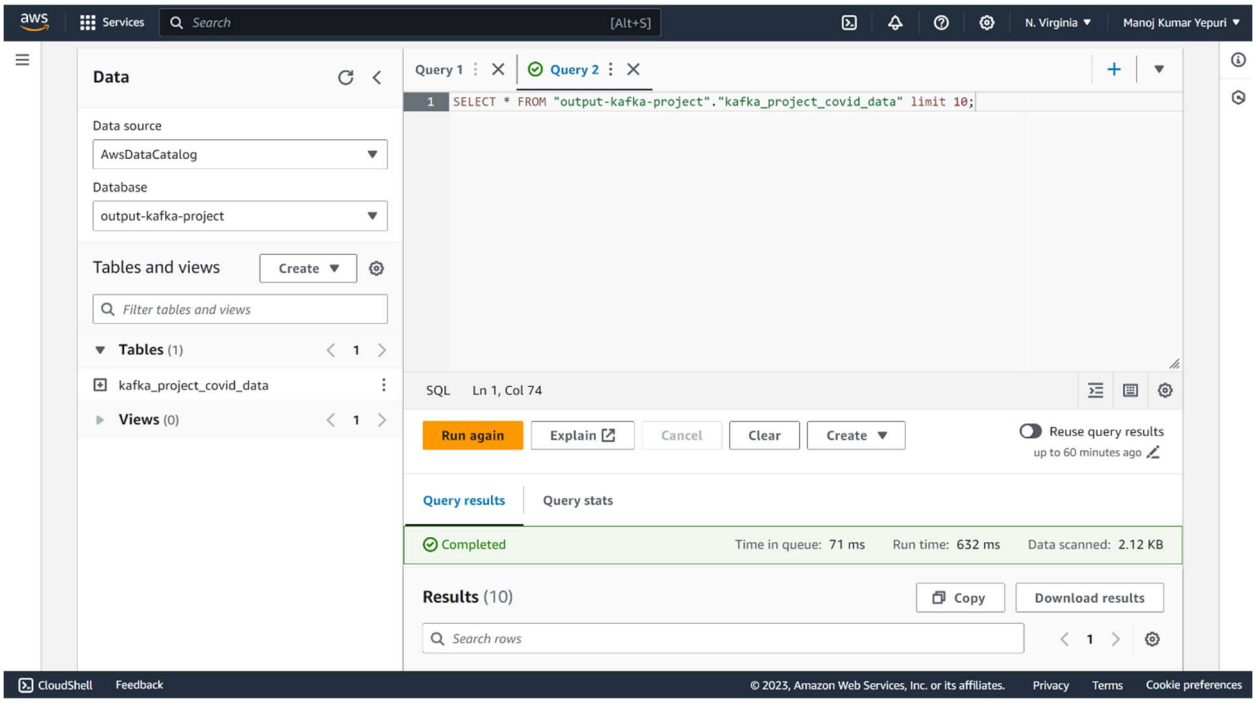


22. Create IAM Role and Crawler: After creating the IAM role, set up a Crawler to connect the S3 bucket (which contains the produced data) and the IAM role for security purposes and S3 bucket access. In this process, create a new database named `output-kafka-project` to store the analyzed data when using AWS Athena.





23. Use Athena Service: Navigate to the Athena service and connect to the database. Athena will provide a default query to verify the correct connection to the S3 bucket. The query should run smoothly, and you should be able to see the data from the S3 bucket.



Results (10)

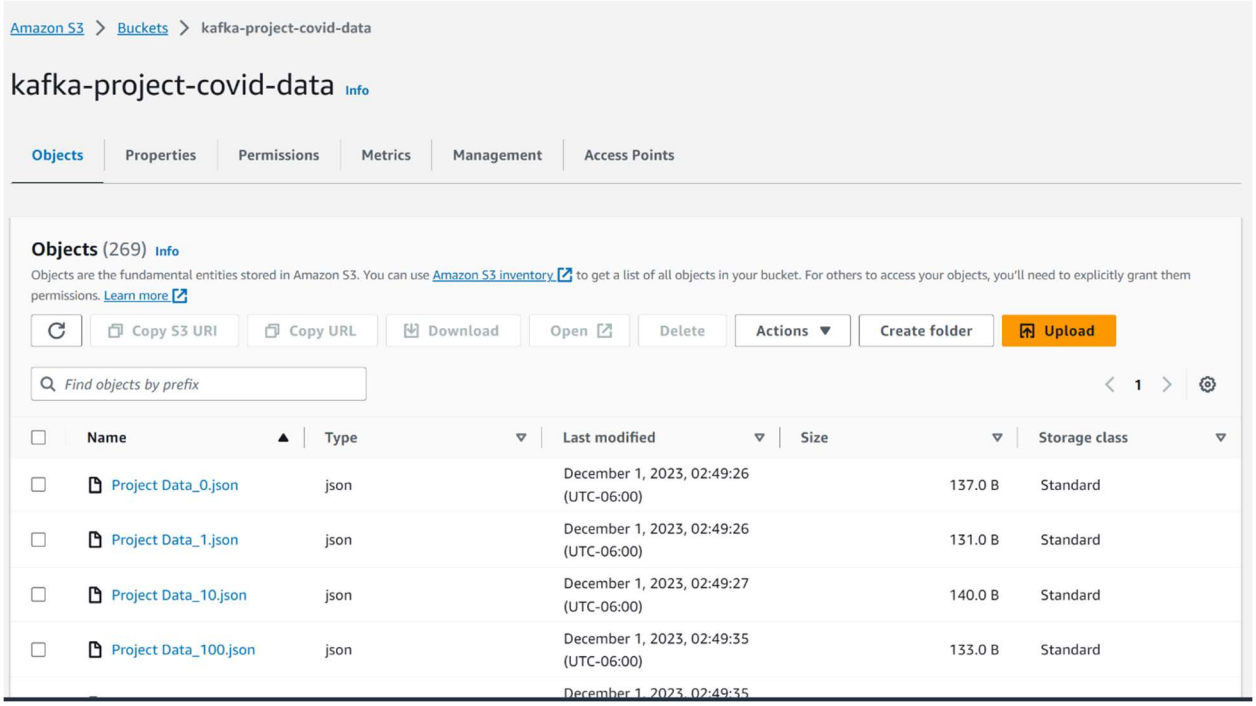
Copy | Download results

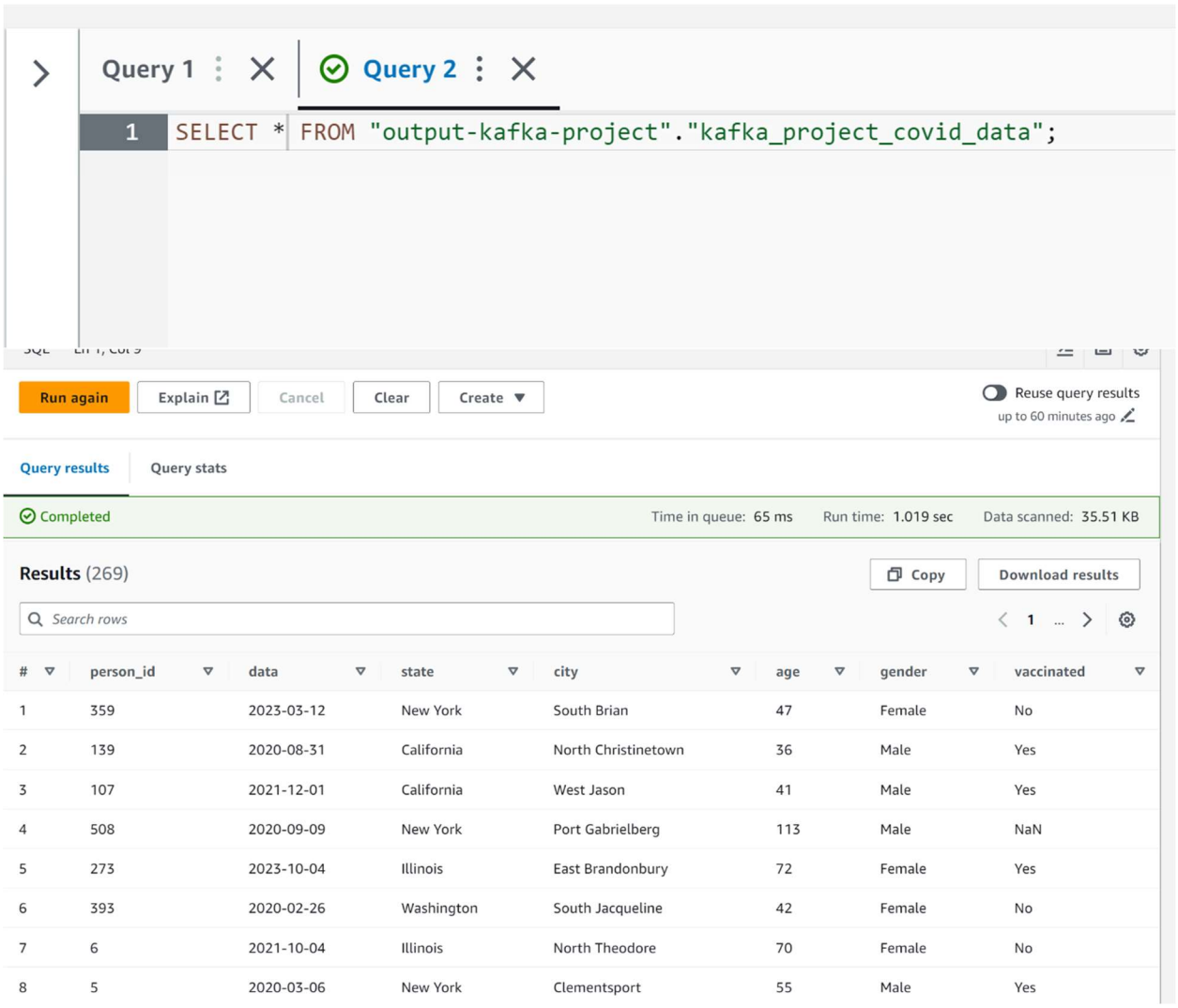
Search rows

1

#	person_id	data	state	city	age	gender	vaccinated
1	111	2020-10-17	Florida	New Vernon	76	Female	Yes
2	333	2021-06-23	Texas	Ramseyhaven	41	Female	Yes
3	138	2023-04-06	Florida	Escobarview	73	Female	No
4	159	2020-12-16	Florida	Port Donnachester	77	Male	No
5	232	2021-07-27	California	East Nicholeton	67	Female	No
6	505	2023-04-25	Washington	Knightville	110	Female	No
7	351	2021-09-25	Illinois	Timothychester	28	Male	Yes
8	121	2023-03-15	Illinois	Gilbertbury	49	Female	No
9	462	2021-04-07	New York	Smalltown	25	Male	Yes
10	483	2023-10-16	Washington	Stephaniehaven	80	Male	Yes

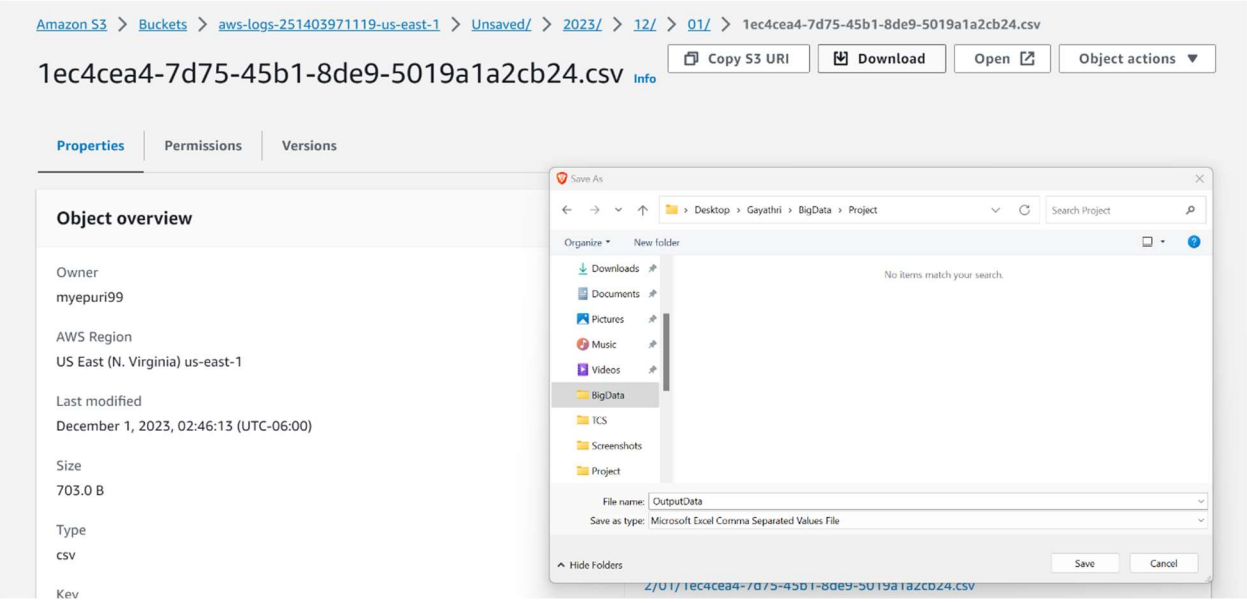
24. Update the Data: Run the producer and consumer code for a few more seconds. You should see the S3 bucket update with new data, which will also be reflected in Athena.





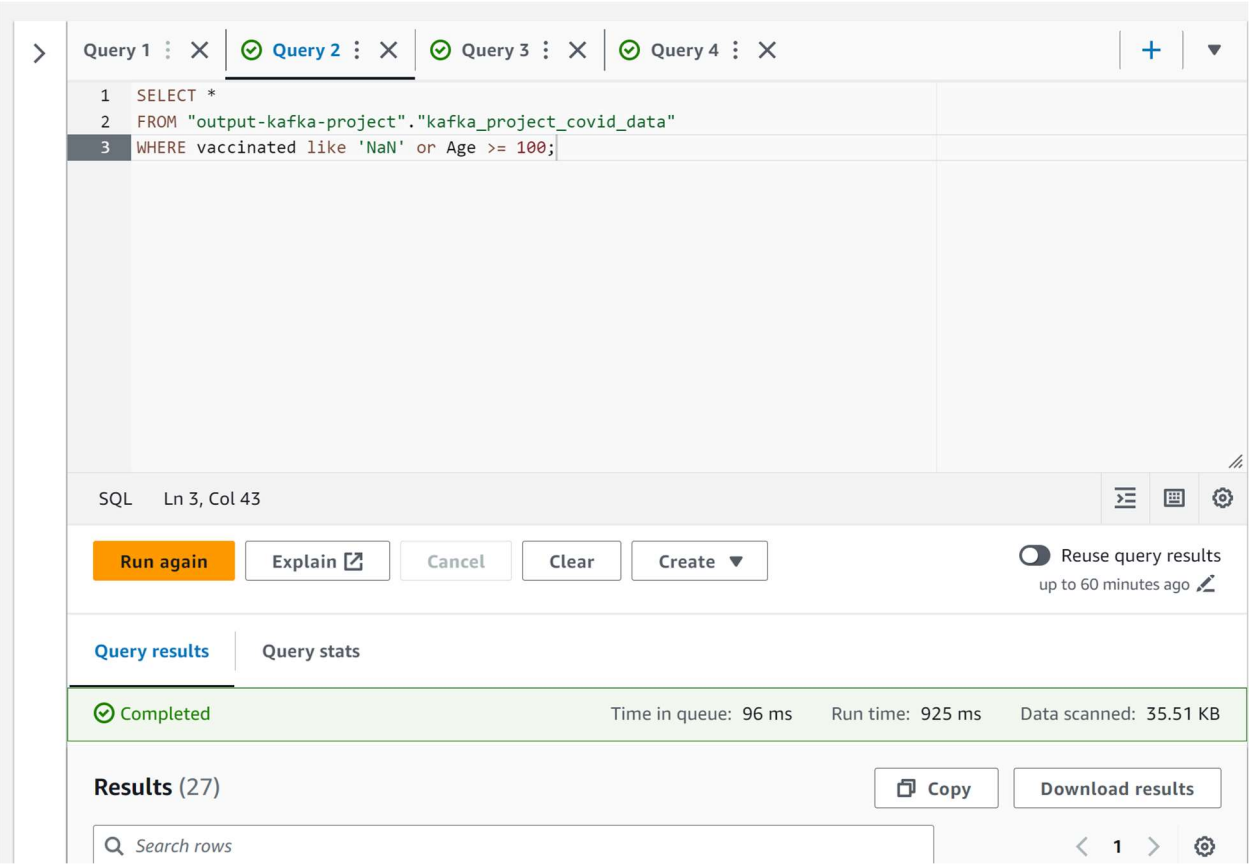
#	person_id	data	state	city	age	gender	vaccinated
1	359	2023-03-12	New York	South Brian	47	Female	No
2	139	2020-08-31	California	North Christinetown	36	Male	Yes
3	107	2021-12-01	California	West Jason	41	Male	Yes
4	508	2020-09-09	New York	Port Gabrielberg	113	Male	NaN
5	273	2023-10-04	Illinois	East Brandonbury	72	Female	Yes
6	393	2020-02-26	Washington	South Jacqueline	42	Female	No
7	6	2021-10-04	Illinois	North Theodore	70	Female	No
8	5	2020-03-06	New York	Clementsport	55	Male	Yes

24.1 Downloading the result for further analysis as **“OutputData.csv”**



25. Data Profiling: Perform data profiling by identifying outliers in the 'Age' column and null values in the 'Vaccinated' column using Athena's SQL query. Remove these outliers and null values for this project

25.1 Identifying the outliers and null values:



Completed | Time in queue: 96 ms | Run time: 925 ms | Data scanned: 35.51 KB

Results (27)

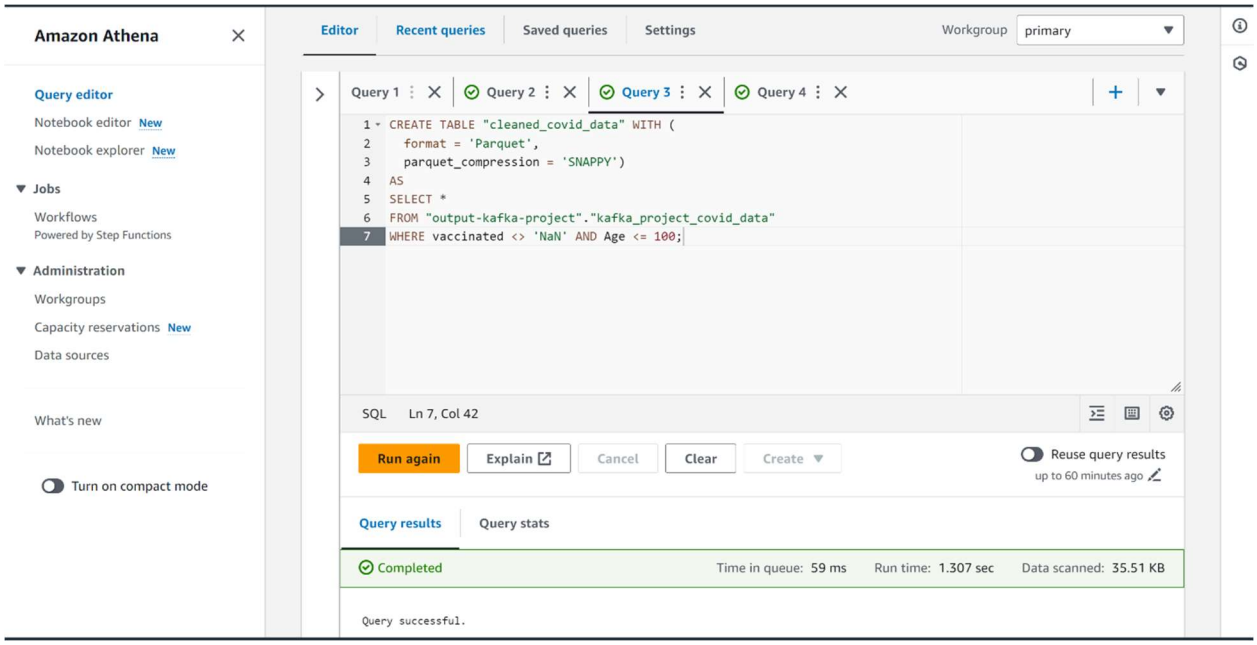
Copy | Download results

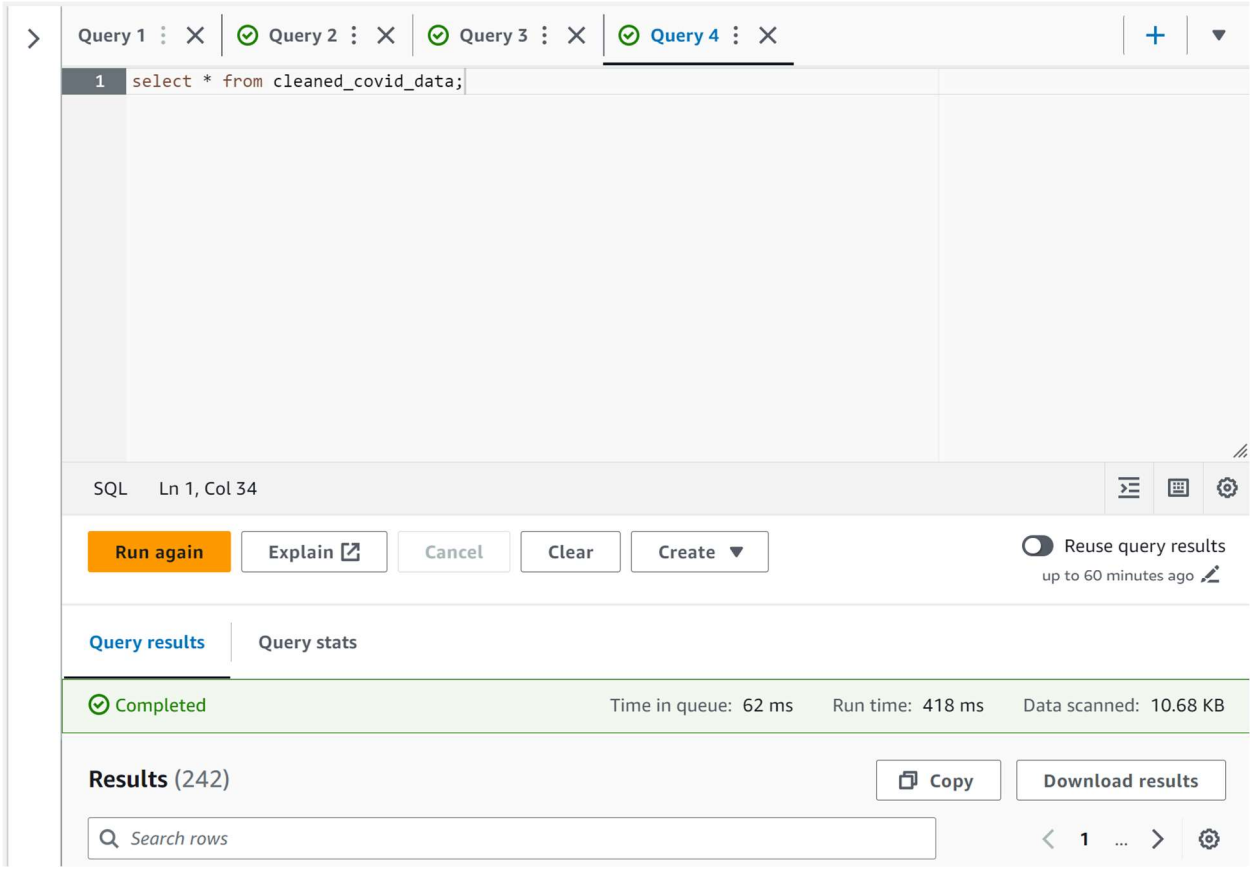
Search rows

1

#	person_id	data	state	city	age	gender	vaccinated
1	508	2020-09-09	New York	Port Gabrielberg	113	Male	NaN
2	512	2021-11-27	Florida	Port Geraldport	118	Female	Yes
3	511	2021-10-11	Texas	South Patriciaton	106	Female	Yes
4	429	2022-11-02	Illinois	Port Samuel	39	Female	NaN
5	2	2020-06-26	Illinois	South Marktown	37	Male	NaN
6	508	2020-09-09	New York	Port Gabrielberg	113	Male	NaN
7	294	2022-07-03	Florida	Port Bradley	43	Male	NaN
8	289	2020-10-29	Florida	East Allen	19	Female	NaN

25.2 Removing the Null/NaN and outliers in age and vaccinated columns another table, download the result as "**CleanedOutput.csv".**





Results (242)

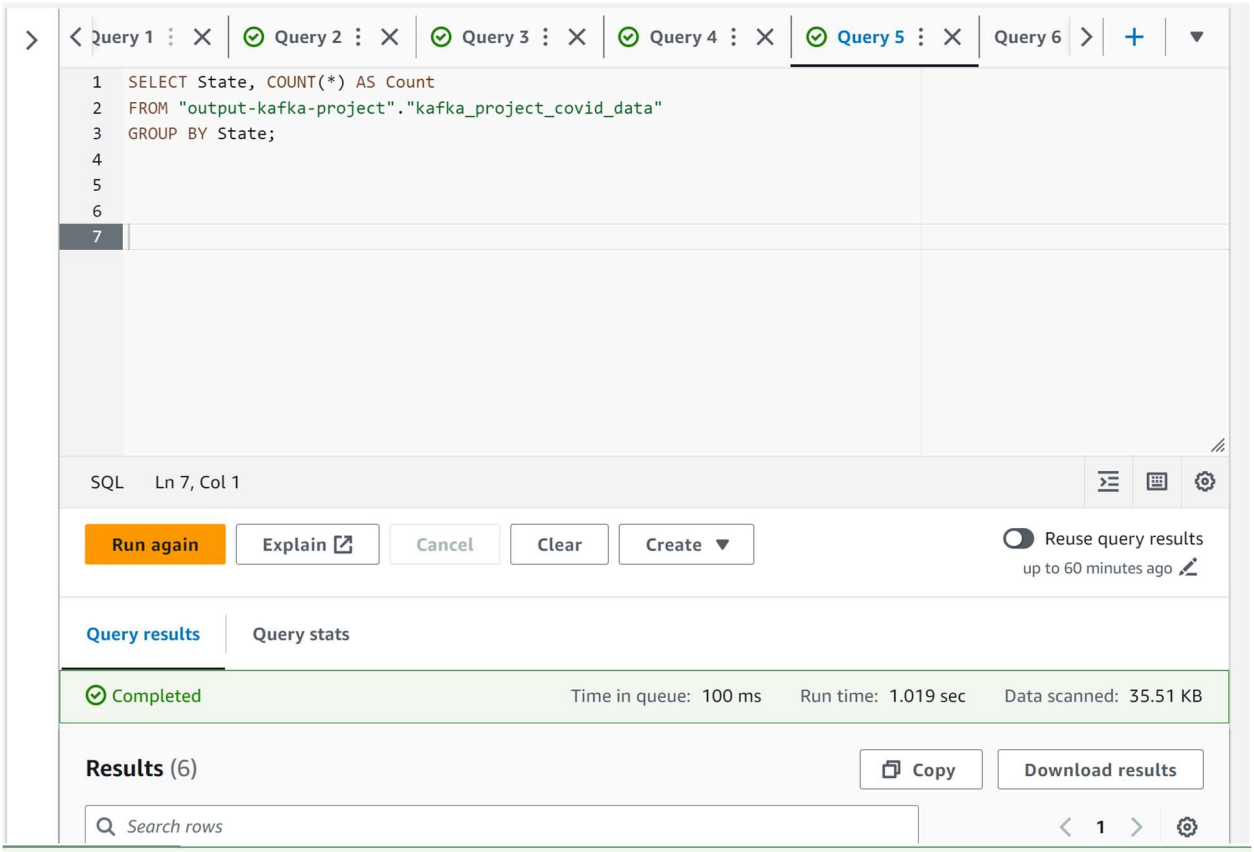
Copy | Download results

Search rows

1 ...

#	person_id	data	state	city	age	gender	vaccinated
1	189	2023-03-06	Illinois	Lauratown	18	Male	No
2	293	2022-04-29	New York	Sierraville	46	Male	Yes
3	31	2021-01-15	New York	Janicemouth	31	Male	No
4	204	2022-02-23	New York	Andreachester	78	Female	No
5	423	2022-01-19	Texas	Coopertown	32	Female	No
6	422	2022-01-08	Illinois	Craighaven	58	Male	No
7	432	2022-05-19	Illinois	Nelsonborough	29	Female	No
8	249	2020-01-14	Washington	East Jeffreymouth	61	Female	No
9	211	2022-06-23	Washington	Howardside	22	Male	Yes
10	432	2022-05-19	Illinois	Nelsonborough	29	Female	No
11	498	2020-10-15	Illinois	Lake William	59	Male	No
12	430	2021-12-12	California	Lake Matthew	45	Male	No
13	226	2022-02-10	Washington	New Elizabethhaven	58	Female	Yes

25.3 Grouping Categorical Attributes:

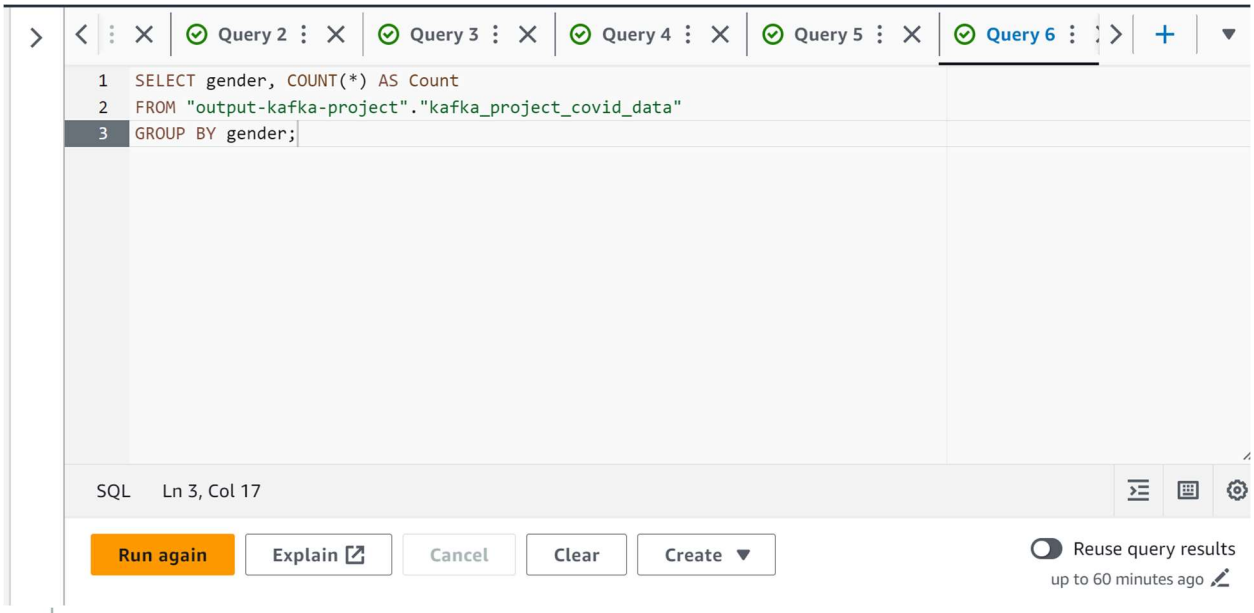


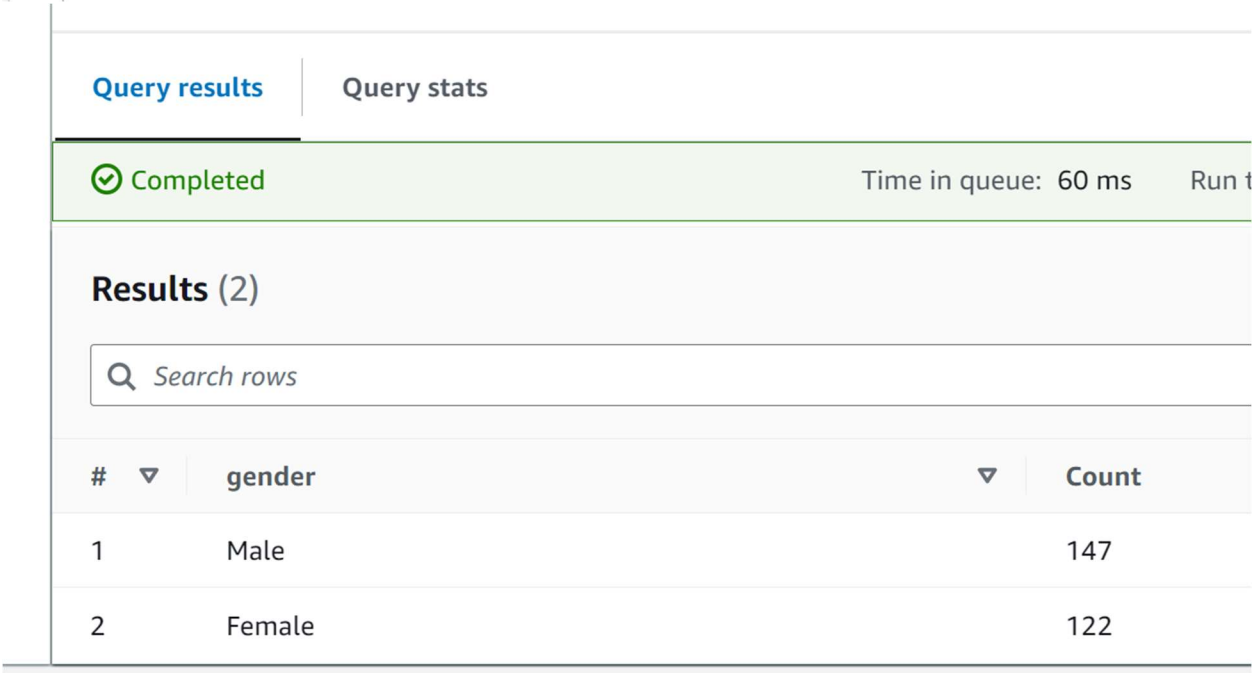
Completed Time in queue: 100 ms Run time: 1.019 sec Data scanned: 35.51 KB

Results (6) Copy Download results

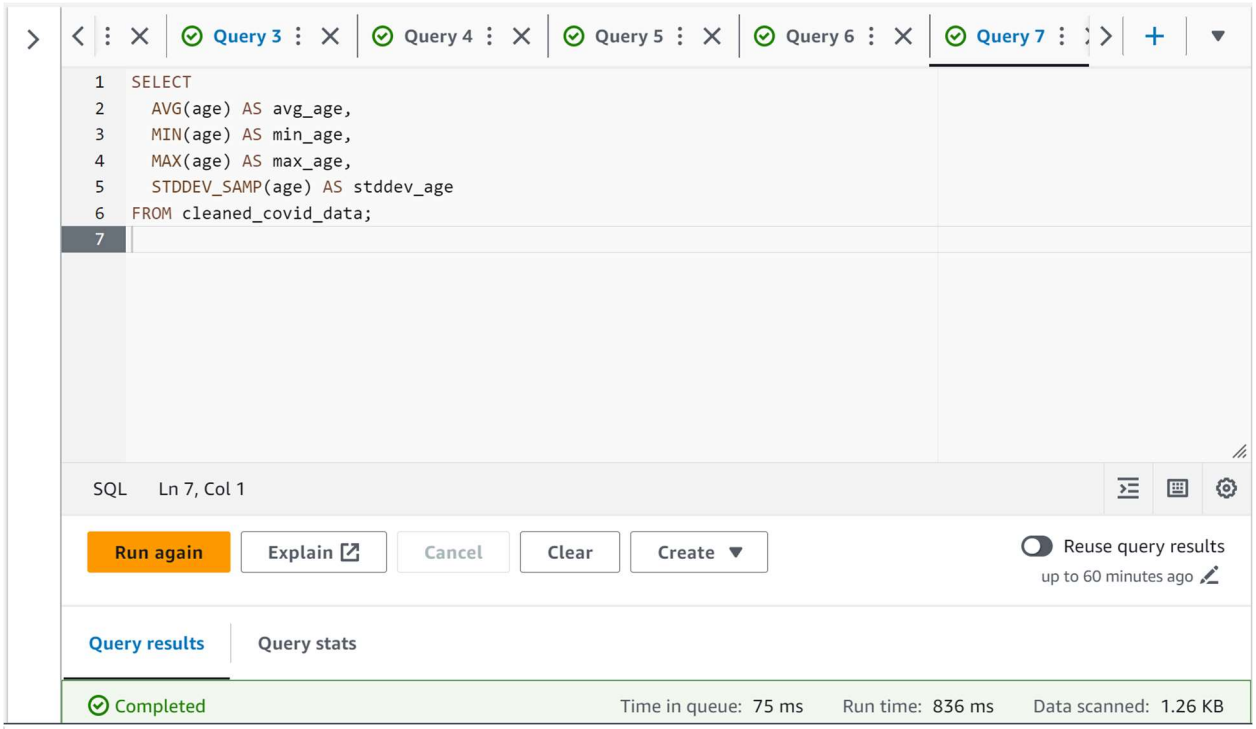
Search rows 1

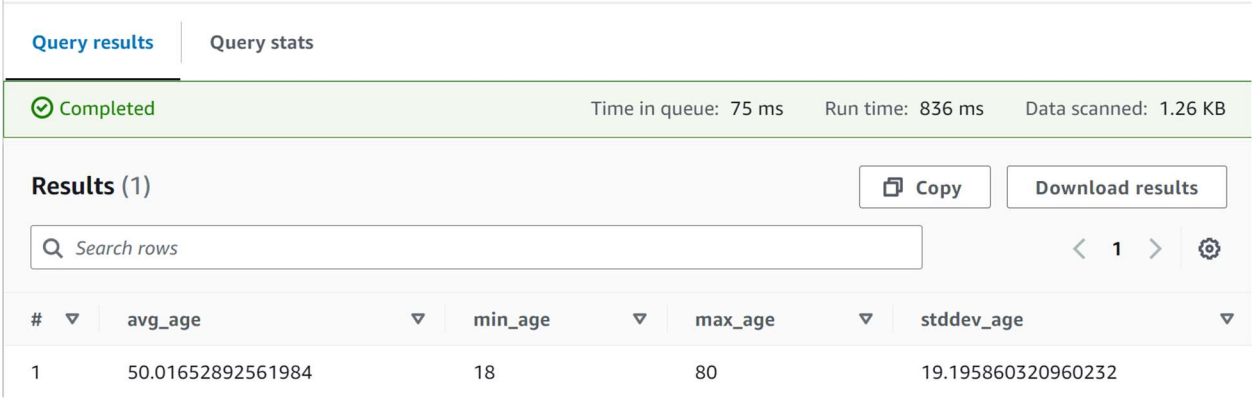
#	State	Count
1	Florida	60
2	New York	42
3	Illinois	66
4	Washington	39
5	California	36
6	Texas	26


Query 2
Query 3
Query 4
Query 5
Query 6
1 SELECT gender, COUNT(*) AS Count
2 FROM "output-kafka-project"."kafka_project_covid_data"
3 GROUP BY gender;
SQL Ln 3, Col 17
Run again
Explain
Cancel
Clear
Create
Reuse query results
up to 60 minutes ago

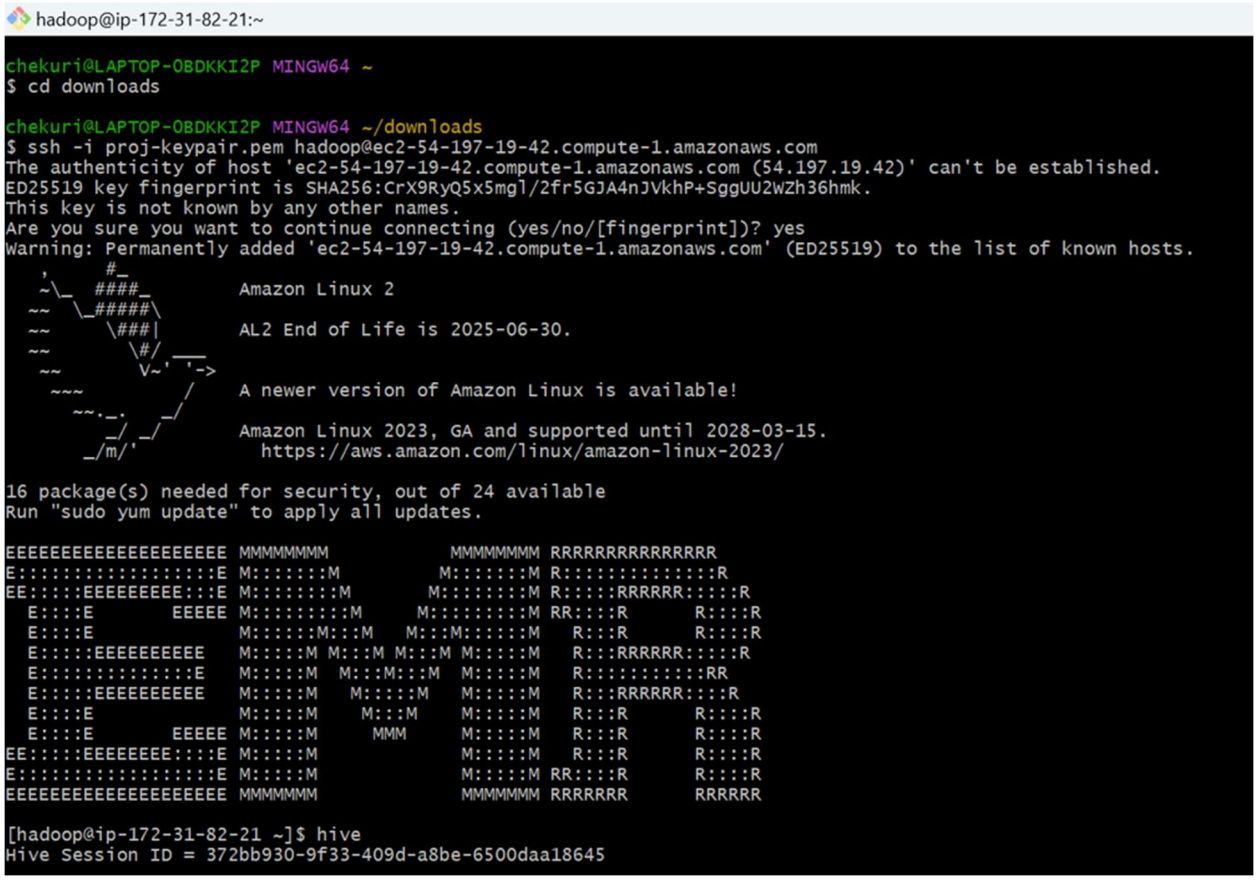

Query results
Query stats
Completed
Time in queue: 60 ms
Run t
Results (2)
Search rows
gender Count
1 Male 147
2 Female 122

26. Calculate Age Statistics: Compute the minimum, maximum, average, and standard deviation of the 'Age' attribute.





27. Data Profiling with Hive: Create an EMR cluster with core Hadoop that includes Hive. Upload the `OutputData.csv` file to the EMR cluster for further data profiling using Hive.



28 Create a table to store the data in original format, setting all data types as strings initially due to the JSON format of the CSV file. Later, convert these to their original data types into another table for manipulation purposes.

```
hive> CREATE TABLE MyDb.project_data(
    >     person_id string,
    >     data string,
    >     state string,
    >     city string,
    >     age string,
    >     gender string,
    >     vaccinated string)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > LOCATION '/home/hadoop/OutputData.csv'
    > TBLPROPERTIES ("skip.header.line.count"="1");
OK
Time taken: 0.279 seconds
hive> LOAD DATA LOCAL INPATH './OutputData.csv'
    > OVERWRITE INTO TABLE MyDb.project_data;
Loading data to table mydb.project_data
OK
Time taken: 0.617 seconds
hive> select * from MyDb.project_data limit 15;
OK
"295"   "2021-11-17"    "Florida"       "Jacksonmouth"  "68"    "Female"        "No"
"257"   "2021-04-29"    "Florida"       "Olsonland"     "76"    "Male"  "No"
"270"   "2021-05-05"    "Illinois"      "Edwinbury"     "64"    "Male"  "NaN"
"414"   "2022-07-28"    "Washington"    "Lake Jerry"    "74"    "Male"  "NaN"
"398"   "2023-06-13"    "Illinois"      "Lake Aaron"    "63"    "Male"  "Yes"
"125"   "2021-06-17"    "Washington"    "Jaredmouth"    "51"    "Male"  "Yes"
"487"   "2021-12-09"    "New York"      "Lake Rachel"   "23"    "Male"  "No"
"354"   "2022-04-24"    "California"    "Brandiburgh"   "34"    "Female"        "Yes"
"13"    "2022-08-20"    "California"    "Jonathanburgh" "73"    "Female"        "No"
"253"   "2022-07-03"    "Florida"       "Bakerland"     "67"    "Male"  "No"
"107"   "2021-12-01"    "California"    "West Jason"    "41"    "Male"  "Yes"
"295"   "2021-11-17"    "Florida"       "Jacksonmouth"  "68"    "Female"        "No"
"468"   "2022-02-25"    "New York"      "North Philip"  "67"    "Female"        "Yes"
"353"   "2020-03-18"    "Texas" "Robertland"    "72"    "Female"        "Yes"
"359"   "2023-03-12"    "New York"      "South Brian"   "47"    "Female"        "No"
Time taken: 1.898 seconds, Fetched: 15 row(s)
hive>
```

```
Time taken: 0.140 seconds
hive> CREATE TABLE MyDb.covid_data AS
    > SELECT
    >     CAST(REGEXP_REPLACE(person_id, '\\D', '') AS INT) pid,
    >     data AS stat_date,
    >     state,
    >     city,
    >     CAST(REGEXP_REPLACE(age, '\\D', '') AS INT) age,
    >     gender,
    >     vaccinated
    > FROM project_data;
Query ID = hadoop_20231201205144_a90e67d0-5dcb-4557-8baf-ad132eb11f52
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1701459769138_0006)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 01/01  [==========================>>] 100%  ELAPSED TIME: 5.77 s
--------------------------------------------------------------------------------
Moving data to directory hdfs://ip-172-31-82-21.ec2.internal:8020/user/hive/warehouse/mydb.db/covid_data
OK
Time taken: 6.181 seconds
hive> select * from covid_data limit 15;
OK
295     "2021-11-17"    "Florida"       "Jacksonmouth"  68      "Female"        "No"
257     "2021-04-29"    "Florida"       "Olsonland"     76      "Male"  "No"
270     "2021-05-05"    "Illinois"      "Edwinbury"     64      "Male"  "NaN"
414     "2022-07-28"    "Washington"    "Lake Jerry"    74      "Male"  "NaN"
398     "2023-06-13"    "Illinois"      "Lake Aaron"    63      "Male"  "Yes"
125     "2021-06-17"    "Washington"    "Jaredmouth"    51      "Male"  "Yes"
487     "2021-12-09"    "New York"      "Lake Rachel"   23      "Male"  "No"
354     "2022-04-24"    "California"    "Brandiburgh"   34      "Female"        "Yes"
13      "2022-08-20"    "California"    "Jonathanburgh" 73      "Female"        "No"
253     "2022-07-03"    "Florida"       "Bakerland"     67      "Male"  "No"
107     "2021-12-01"    "California"    "West Jason"    41      "Male"  "Yes"
295     "2021-11-17"    "Florida"       "Jacksonmouth"  68      "Female"        "No"
468     "2022-02-25"    "New York"      "North Philip"  67      "Female"        "Yes"
353     "2020-03-18"    "Texas" "Robertland"    72      "Female"        "Yes"
359     "2023-03-12"    "New York"      "South Brian"   47      "Female"        "No"
Time taken: 0.083 seconds, Fetched: 15 row(s)
hive>
```

29. Analyze the Data: Describe the created `covid_data` table. Identify null values in the 'Vaccinated' column and outliers in the 'Age' column. For the purpose of this project, remove these nulls and outliers using the commands shown in the provided images.

```
hive> DESCRIBE FORMATTED MyDb.covid_data;
OK
# col_name              data_type               comment
pid                     int
stat_date               string
state                   string
city                    string
age                     int
gender                  string
vaccinated              string

# Detailed Table Information
Database:               mydb
OwnerType:              USER
Owner:                  hadoop
CreateTime:             Fri Dec 01 20:51:50 UTC 2023
LastAccessTime:         UNKNOWN
Retention:              0
Location:               hdfs://ip-172-31-82-21.ec2.internal:8020/user/hive/warehouse/mydb.db/covid_data
Table Type:             MANAGED_TABLE
Table Parameters:
        COLUMN_STATS_ACCURATE   {\"BASIC_STATS\":\"true\"}
        bucketing_version       2
        numFiles                1
        numRows                 269
        rawDataSize             15679
        totalSize               15948
        transient_lastDdlTime   1701463910

# Storage Information
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        serialization.format    1
Time taken: 0.052 seconds, Fetched: 37 row(s)
hive> [hadoop@ip-172-31-82-21 ~]$ hive
Hive Session ID = 9cbcb5af-e9d8-4f47-a298-c2ffa2049167
```

29.1 Identifying Null/NaN values in the covid_data

```
hive> select * from covid_data where vaccinated not like '%No%' and vaccinated not like '%Yes%';
OK
270     "2021-05-05"    "Illinois"      "Edwinbury"     64      "Male"  "NaN"
414     "2022-07-28"    "Washington"    "Lake Jerry"    74      "Male"  "NaN"
508     "2020-09-09"    "New York"      "Port Gabrielberg"      113     "Male"  "NaN"
1       "2021-06-17"    "Illinois"      "Lopezton"      71      "Male"  "NaN"
368     "2020-03-09"    "Texas" "Michaelberg"   75      "Male"  "NaN"
429     "2022-11-02"    "Illinois"      "Port Samuel"   39      "Female"        "NaN"
30      "2021-06-01"    "Florida"       "Liutown"       36      "Male"  "NaN"
2       "2020-06-26"    "Illinois"      "South Marktown"        37      "Male"  "NaN"
294     "2022-07-03"    "Florida"       "Port Bradley"  43      "Male"  "NaN"
429     "2022-11-02"    "Illinois"      "Port Samuel"   39      "Female"        "NaN"
203     "2022-04-22"    "Washington"    "South Ericchester"     20      "Male"  "NaN"
508     "2020-09-09"    "New York"      "Port Gabrielberg"      113     "Male"  "NaN"
289     "2020-10-29"    "Florida"       "East Allen"    19      "Female"        "NaN"
280     "2022-11-23"    "Florida"       "East Brent"    39      "Female"        "NaN"
294     "2022-07-03"    "Florida"       "Port Bradley"  43      "Male"  "NaN"
29      "2020-07-09"    "Washington"    "South Michael" 60      "Male"  "NaN"
508     "2020-09-09"    "New York"      "Port Gabrielberg"      113     "Male"  "NaN"
Time taken: 0.107 seconds, Fetched: 17 row(s)
hive>
```

29.1 Identifying age outliers in the covid_data

```
hive> select * from covid_data where age>=100;
OK
508     "2020-09-09"    "New York"      "Port Gabrielberg"      113     "Male"  "NaN"
509     "2020-06-10"    "California"    "Martinchester" 115     "Male"  "Yes"
509     "2020-06-10"    "California"    "Martinchester" 115     "Male"  "Yes"
507     "2020-02-21"    "Florida"       "Jessicaview"   118     "Female"        "No"
510     "2021-06-07"    "Texas" "Stephanieland" 115     "Male"  "Yes"
504     "2020-11-20"    "Florida"       "Johnsonchester"        110     "Female"        "Yes"
517     "2020-02-27"    "California"    "North William" 109     "Female"        "No"
508     "2020-09-09"    "New York"      "Port Gabrielberg"      113     "Male"  "NaN"
511     "2021-10-11"    "Texas" "South Patriciaton"     106     "Female"        "Yes"
507     "2020-02-21"    "Florida"       "Jessicaview"   118     "Female"        "No"
524     "2021-05-01"    "Florida"       "North Ann"     114     "Female"        "Yes"
512     "2021-11-27"    "Florida"       "Port Geraldport"       118     "Female"        "Yes"
508     "2020-09-09"    "New York"      "Port Gabrielberg"      113     "Male"  "NaN"
Time taken: 0.099 seconds, Fetched: 13 row(s)
hive>
```

29.1 Handling the outliers and Null/NaN values by removing them from the table.

```
hive> INSERT OVERWRITE TABLE covid_data  select * from covid_data where vaccinated like '%No%' or vaccinated like '%Yes%';
Query ID = hadoop_20231201211515_1a08cdb8-728d-414c-9e44-827e83b97408
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1701459769138_0008)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 6.36 s
----------------------------------------------------------------------------------------------
Loading data to table mydb.covid_data
OK
Time taken: 11.368 seconds
```

```
hive> INSERT OVERWRITE TABLE covid_data SELECT * FROM covid_data WHERE age <100;
Query ID = hadoop_20231201212924_9225c690-9fe7-4799-ac6f-3ac9a700b91f
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1701459769138_0009)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 6.54 s
----------------------------------------------------------------------------------------------
Loading data to table mydb.covid_data
OK
Time taken: 13.222 seconds
hive> select *from covid_data limit 15;
OK
295     "2021-11-17"    "Florida"       "Jacksonmouth"  68      "Female"        "No"
257     "2021-04-29"    "Florida"       "Olsonland"     76      "Male"  "No"
398     "2023-06-13"    "Illinois"      "Lake Aaron"    63      "Male"  "Yes"
125     "2021-06-17"    "Washington"    "Jaredmouth"    51      "Male"  "Yes"
487     "2021-12-09"    "New York"      "Lake Rachel"   23      "Male"  "No"
354     "2022-04-24"    "California"    "Brandiburgh"   34      "Female"        "Yes"
13      "2022-08-20"    "California"    "Jonathanburgh" 73      "Female"        "No"
253     "2022-07-03"    "Florida"       "Bakerland"     67      "Male"  "No"
107     "2021-12-01"    "California"    "West Jason"    41      "Male"  "Yes"
295     "2021-11-17"    "Florida"       "Jacksonmouth"  68      "Female"        "No"
468     "2022-02-25"    "New York"      "North Philip"  67      "Female"        "Yes"
353     "2020-03-18"    "Texas" "Robertland"    72      "Female"        "Yes"
359     "2023-03-12"    "New York"      "South Brian"   47      "Female"        "No"
318     "2020-12-06"    "Florida"       "West Kathy"    38      "Male"  "Yes"
197     "2021-04-18"    "Florida"       "North Manuel"  20      "Male"  "Yes"
Time taken: 0.078 seconds, Fetched: 15 row(s)
hive> select * from covid_data where age>=100 or vaccinated not like '%No%' and vaccinated not like '%Yes%';
OK
Time taken: 0.099 seconds
hive>
```

30. Group Records: Group records according to their 'State' attribute.

```
hive> SELECT state, COUNT(*) FROM covid_data GROUP BY state;
Query ID = hadoop_20231201213502_f439f0e5-f825-4de5-98a8-d08da39ab465
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1701459769138_0010)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      2          2        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 6.46 s
--------------------------------------------------------------------------------
OK
"Illinois"      61
"California"    33
"Florida"       50
"New York"      39
"Texas" 23
"Washington"    36
Time taken: 12.657 seconds, Fetched: 6 row(s)
hive> SELECT gender, COUNT(*) FROM covid_data GROUP BY gender;
Query ID = hadoop_20231201213754_aba7c917-5975-4294-9130-24c9e4c9e121
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1701459769138_0010)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      2          2        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 5.77 s
--------------------------------------------------------------------------------
OK
"Male"  131
"Female"        111
Time taken: 6.053 seconds, Fetched: 2 row(s)
hive> SELECT vaccinated, COUNT(*) FROM covid_data GROUP BY vaccinated;
Query ID = hadoop_20231201213815_f6bffbe5-c08d-4fd6-9032-aa6db3df1c18
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1701459769138_0010)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      2          2        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 5.11 s
--------------------------------------------------------------------------------
OK
"No"    132
"Yes"   110
Time taken: 5.412 seconds, Fetched: 2 row(s)
hive>
```

30.1 Also, group records by 'Age', categorizing them into three age groups: 1-25, 26-50, and 51-100.

```
hive> SELECT COUNT(*) FROM covid_data WHERE age BETWEEN 1 AND 25;
Query ID = hadoop_20231201214031_a1fd4e65-ba81-4861-94cd-14daec715b39
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1701459769138_0010)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 5.67 s
--------------------------------------------------------------------------------
OK
37
Time taken: 6.012 seconds, Fetched: 1 row(s)
hive> SELECT COUNT(*) FROM covid_data WHERE age BETWEEN 26 AND 50;
Query ID = hadoop_20231201214050_15fadb10-be4b-4eba-b1b4-e910c0a30f78
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1701459769138_0010)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 5.61 s
--------------------------------------------------------------------------------
OK
83
Time taken: 5.893 seconds, Fetched: 1 row(s)
hive> SELECT COUNT(*) FROM covid_data WHERE age BETWEEN 51 AND 100;
Query ID = hadoop_20231201214102_93cd0148-5f6f-457e-bce0-b6bece0a77a9
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1701459769138_0010)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 0.37 s
--------------------------------------------------------------------------------
OK
122
Time taken: 0.638 seconds, Fetched: 1 row(s)
hive>
```

31. Further Analysis: Calculate the minimum, maximum, average, and standard deviation of 'Age' from the dataset.

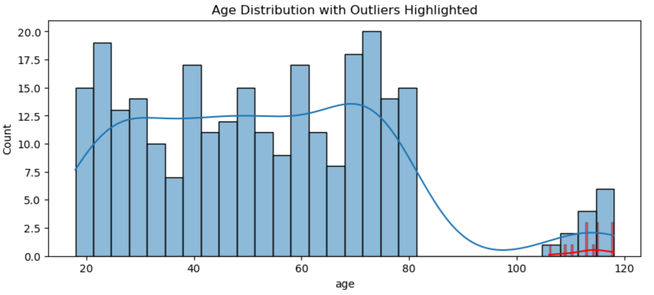
```
hive> SELECT MIN(age), MAX(age), AVG(age), STDDEV(age) FROM covid_data;
Query ID = hadoop_20231201214142_4c38f07a-936c-4dc1-8071-2fa3b81ceceb
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1701459769138_0010)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 5.27 s
--------------------------------------------------------------------------------
OK
18      80      50.01652892561984      19.156158395684876
Time taken: 5.636 seconds, Fetched: 1 row(s)
```

32. Data Analysis: Data analysis is done using python in the form of Graphs.

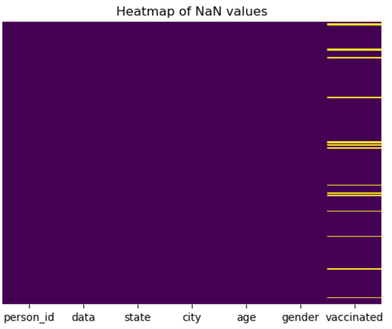
In [122…
```
# Graphs for Raw Data set representing Outliers and Null/NaN values
```

In [123…
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


# Load the data Output of raw data set
df = pd.read_csv('C://Users//myepu//Downloads//OutputData.csv')
```

In [124…
```
# Creating a mask for the outliers
outliers = df['age'] > 100

plt.figure(figsize=(10,4))

# Histogram for all data
sns.histplot(data=df, x='age', bins=30, kde=True)

# Histogram for the outliers
sns.histplot(data=df[outliers], x='age', bins=30, kde=True, color='red')

plt.title('Age Distribution with Outliers Highlighted')
plt.show()
```



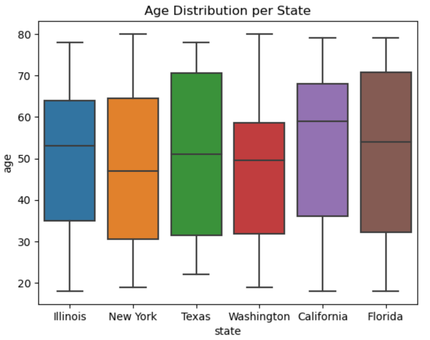
In [125…
```
# Creating a heatmap showing where the Null/NaN values are
sns.heatmap(df.isnull(), cbar=False, cmap='viridis', yticklabels=False)
plt.title('Heatmap of NaN values')
plt.show()
```



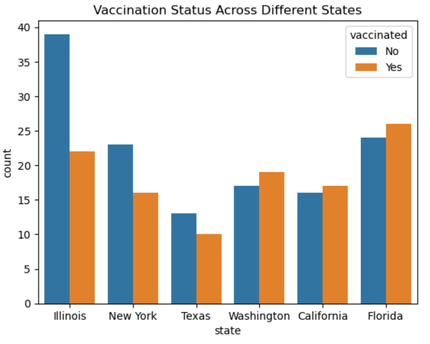
In [126…
```
#Graphs for the cleaned Data Set
```

In [127…
```
# Load the data of cleaned data set
df = pd.read_csv('C://Users//myepu//Downloads//CleanedOutput.csv')
```

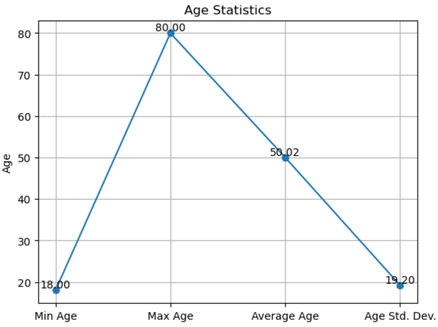
In [128…
```
# Box plot for 'age' per 'state' for the cleaned data set
sns.boxplot(x='state', y='age', data=df)
plt.title('Age Distribution per State')
plt.show()
```



In [129…
```
# Count plot for 'vaccinated' status across different 'states' for the cleaned data
sns.countplot(x='state', hue='vaccinated', data=df)
plt.title('Vaccination Status Across Different States')
plt.show()
```



In [131…
```
#Line plot for min,max,avg,std of 'AGE'
min_age = df['age'].min()
max_age = df['age'].max()
avg_age = df['age'].mean()
std_age = df['age'].std()

ages = [min_age, max_age, avg_age, std_age]
labels = ['Min Age', 'Max Age', 'Average Age', 'Age Std. Dev.']

plt.plot(labels, ages, marker='o')
plt.ylabel('Age')
plt.title('Age Statistics')

for i, age in enumerate(ages):
    plt.text(i, age, f'{age:.2f}', ha = 'center', va='bottom')

plt.grid(True)
plt.show()
```



In [ ]:
```
```

33. Project Roles and Responsibilities:

Manoj Kumar Yepuri - Data Production, Ingestion, Visualization, and Apache Kafka Research

Data production, ingestion, and visualization tasks were successfully completed. A raw dataset was generated using Python, then streamed into the Apache Kafka cluster, stored the produced data into Amazon S3 and visualized. Comprehensive research on Apache Kafka was also conducted, exploring its architecture, use cases, and how it facilitates real-time data processing. Insights on best practices for optimizing Kafka's performance and reliability were provided and implemented in the project.

Gayathri Chekuri - Data Profiling, Cleaning, Storage, and Apache Storm Research

The data profiling and cleaning of the project were effectively handled using Hive. The streaming data was profiled to understand its properties, outliers or null values were identified and removed. Detailed research on Apache Storm was undertaken, investigating its real-time computation capabilities, architecture and use cases.

Rushikesh Kadam - Data Cataloging, Analysis, and AWS Kinesis Research

Data cataloging and analysis were successfully completed using AWS Glue Crawler for cataloging the stored data and Amazon Athena for data analysis. An in-depth study on AWS Kinesis was conducted, focusing on its features, architecture, and benefits for real-time data streaming. Comparison of AWS Kinesis with similar technologies, such as Apache Kafka, provided valuable insights.

34. Conclusion

In conclusion, this project exemplifies the power of collaboration and the integration of various technologies to build a robust and efficient real-time data streaming pipeline. The journey of the data, from raw input to a cleaned, structured form ready for analysis, was a testament to the team's concerted efforts, where everyone participated actively in all phases of the project.

The project's success was rooted in the team's collective learning and application of Apache Kafka as the main data ingestion and streaming platform. Through this, the team witnessed Kafka's potential as a reliable, high-throughput solution for real-time data processing, and its capability to seamlessly transmit data between the producer and consumer.

Each team member contributed to the crucial task of data cleaning and profiling, enhancing the quality and reliability of the data for accurate analysis. The team learned the importance of these steps and successfully applied this knowledge to ensure our data was ready for the subsequent stages.

Our venture into cloud-based solutions, particularly the integration with AWS S3 for data storage, revealed the utility of such services in managing large volumes of data securely and efficiently. This experience broadened our understanding of cloud storage solutions and their role in modern data pipelines.

The team's exploration of AWS Glue Crawler and Amazon Athena for cataloging and analyzing streaming data provided invaluable insights into the effectiveness of these tools. Our ability to profile data, identify outliers using Hive, Athena's SQL queries, and extract detailed insights from our dataset was a significant achievement.

A crucial task was data cleaning and profiling, where Apache Hive played a significant role. Hive was instrumental in transforming raw data into a more structured and cleaner format, and its SQL-like interface made it approachable for everyone on the team. The use of Hive broadened our understanding of data cleaning processes and emphasized the importance of clean data for accurate analysis.

The versatility of Jupyter notebooks, used throughout the project for a variety of tasks, from setting up Kafka producers, consumers and executing Python code for data production to data analyzation, showcased the flexibility of this tool and its vital role in data science projects.

Above all, every team member was involved in every phase of the pipeline - from production, ingestion, cleaning, storage, cataloging, to analysis. This comprehensive involvement not only enhanced our individual skill sets but also fostered a collaborative learning environment. We gained hands-on experience in various tools and technologies and developed a deeper understanding of real-time data processing systems.

In essence, this project was more than just the successful creation of a data pipeline. It was a journey of collective learning, exploration, and application of various technologies. The lessons we take from this project will undoubtedly serve as a valuable foundation for our future endeavors in data processing and analysis.