An Industry-Oriented Mini Project Report

on

"**Real Time Stream Processing and with Azure Databricks and Apache Spark**"

Submitted to the

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

In partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY IN

COMPUTER SCIENCE & ENGINEERING - DATA SCIENCE

BY

GUJJA PRANAY KUMAR (21WJ5A6703)

Under the Esteemed Guidance of

Mrs. G.USHA RANI

Asst. Professor, CSE-Data Science

Department of Computer Science and Engineering-Data Science

GURU NANAK INSTITUTIONS TECHNICAL CAMPUS (AUTONOMOUS)

School of Engineering and Technology

Ibrahimpatnam R.R. District 501506

2023-2024

RAM
TECHNOLOGY
Reach Your Aim by Mastering

RAM Innovative Infotech
M : +91 9581 012 012
E : raminnovativeinfotech@gmail.com
Flat No.#309, Amrutha Ville,
Opp: Yashoda Hospital, Somajiguda,
Hyderabad-82, Telangana, India
Www.raminnovativeinfotech.webs.com

PROJECT COMPLETION CERTIFICATE

This is to certify that the following student of final year B.Tech, Department of **CSE-Data Science** - Guru Nanak Institutions Technical Campus (GNITC) has completed her training and project at GNITC successfully.

STUDENT NAME: GUJJA PRANAY KUMAR

Roll No: 21WJ5A6703

The training was conducted on BIG DATA Technology for the completion of the project titled REAL TIME STREAM PROCESSING AND WITH AZURE DATABRICKS AND APACHE SPARK in **APRIL - 2024**. The project has been completed in all aspects.

RAM INNOVATIVE INFOTECH

Signature

GURU NANAK INSTITUTIONS TECHNICAL CAMPUS
AUTONOMOUS
under Section 2 (f) & 12 (b) of University Grants Commission Act
www.gniindia.org

Date: 27-04-2024

Department of Computer Science and Engineering-Data Science

CERTIFICATE

This is to certify that the project work titled **"REAL TIME STREAM PROCESSING WITH AZURE DATABRICKS AND APACHE SPARK"** submitted to the **GURU NANAK INSTITUTIONS TECHNICAL CAMPUS**, affiliated to JNTUH, by **Gujja Pranay Kumar (21WJ5A6703)** is a bonafide record of the work done by the students towards partial fulfillment of requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering - Data Science** during the academic year 2023-2024.

Supervisor

Mrs.G.Usha Rani
Asst. Professor
Dept. of CSE(DS)

Head of the Department

Dr.Ch. Subbalakshmi
Professor, HOD - CSE(DS)

Project In-Charge

Mrs. Pooja Kulkarni
Asst. Professor
Dept. of CSE(DS)

External Examinar

DECLARATION

I hereby declare that this project report titled "**REAL TIME STREAM PROCESSING AND WITH AZURE DATABRICKS AND APACHE SPARK**" submitted to the Department of Computer Science & Engineering - Data Science, GURU NANAK INSTITUTIONS TECHNICAL CAMPUS is a record of original work done by me under the guidance of Mrs.G.Usha Rani. The information and data given in the report is authentic to the best of my knowledge. This project report is not submitted to any other university or institution for the award of any degree or diploma or published any time before.

GUJJA PRANAY KUMAR (21WJ5A6703)

Date : 18/04/2024

Place : Hyderabad

ACKNOWLEDGEMENT

We wish to express our sincere thanks to **Dr. Rishi Sayal,** Associate Director, GNITC for providing us the conductive environment for carrying through our academic schedules and Project with ease.

We have been truly blessed to have a wonderful adviser **Dr.Ch.Subbalakshmi,** Professor & HOD of CSE**-** DATA SCIENCE, GNITC for guiding us to explore the ramification of our work and we express our sincere gratitude towards her for leading me thought the completion of Project.

We would like to say our sincere thanks to **Mrs. Pooja Kulkarni**, Assistant Professor, Department of CSE- DATA SCIENCE, Project Coordinator, for providing seamless support and right suggestions are given in the development of the project.

"We extend our special thanks to our internal guide, **Mrs.G. Usha Rani**, Assistant Professor in CSE- DATA SCIENCE, for her valuable suggestions and constant guidance at every stage of the project.".

Finally, we would like to thank our family members for their moral support and encouragement to achieve goals.

Gujja Pranay Kumar (21WJ5A6703)

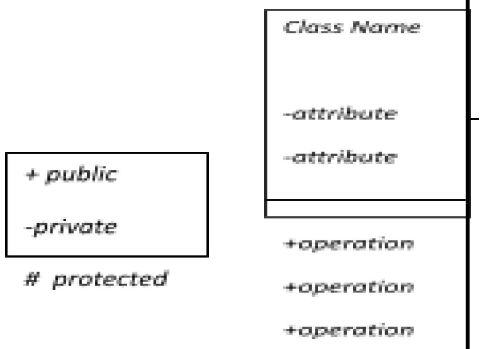
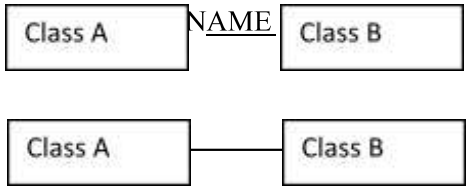
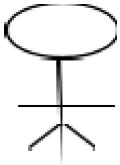
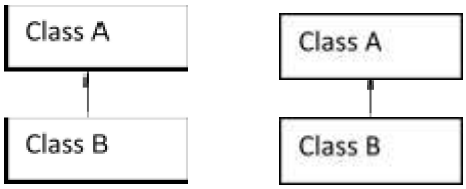
TABLE OF CONTENTS

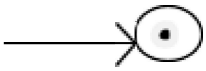
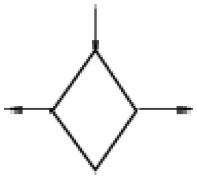
ABSTRACT

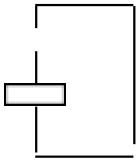
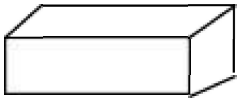
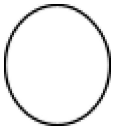
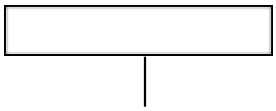
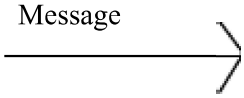
This project proposes the conception and execution of a real-time stream processing solution leveraging Azure Databricks and Apache Spark, with the primary objective of empowering organizations to exploit real-time data insights effectively. The initial paragraph delineates an elucidation of fundamental stream processing concepts, emphasizing its pivotal role in facilitating agile responses to evolving events and trends. It articulates the comprehensive exploration of stream processing architectures, event-driven processing, and stateful stream processing to furnish a holistic comprehension of the domain. Additionally, it underscores the paramount significance of real-time data processing in facilitating timely decision-making and proactive actions. Subsequently, the second paragraph of the abstract delves into the technical intricacies of the project, highlighting the exploration of Azure Databricks and Apache Spark features pertinent to real-time data processing. It elucidates the integration of Azure Databricks with other Azure services such as Event Hubs, Synapse Analytics, and Machine Learning, accentuating the platform's inherent scalability and adaptability. Furthermore, it delineates the implementation of data ingestion pipelines and stream processing logic utilizing Apache Spark Streaming, with a particular focus on incorporating state management to ensure data reliability and system resilience. The abstract concludes by emphasizing the pivotal role of monitoring, management, evaluation, efficacy, performance, and scalability of the proposed real-time stream processing solution.

LIST OF FIGURES

LIST OF SYMBOLS

S.NO	NOTATION NAME	NOTATION	DESCRIPTION
1.	Class		Represents a collection of similar entities grouped together.
2.	Association		Associations represents static relationships between classes. Roles represents the way the two classes see each other.
3.	Actor		It aggregates several classes into a single classes.
4.	Aggregation		Interaction between the system and external environment

5.	Relation (uses)	uses	Used for additional process communication.
6.	Relation (extends)	extends	Extends relationship is used when one use case is similar to another use case but does a bit more.
7.	Communication		Communication between various use cases.
8.	State	State	State of the processes.
9.	Initial State		Initial state of the object
10.	Final state		Final state of the object
11.	Control flow		Represents various control flow between the states.
12.	Decision box		Represents decision making process from a constraint
13.	Use case	Uses case	Interact ion between the system and external environment.

14.	Component		Represents physical modules which are a collection of components.
15.	Node		Represents physical modules which are a collection of components.
16.	Data Process/State		A circle in DFD represents a state or process which has been triggered due to some event or action.
17.	External entity		Represents external entities such as keyboard, sensors, etc.
18.	Transition		Represents communication that occurs between processes.
19.	Object Lifeline		Represents the vertical dimensions that the object communications.
20.	Message		Represents the message exchanged.

CHAPTER 1
INTRODUCTION

1.1 GENERAL

In an age dominated by the relentless flow of real-time data, organizations find themselves at the nexus of opportunity and challenge. The demand for instantaneous insights and proactive responses has propelled the reliance on stream processing technologies to unprecedented heights. Among these, Apache Spark stands as a stalwart, offering formidable stream processing capabilities. Combined with the scalability and flexibility of Azure Databricks, a compelling platform emerges, promising to revolutionize the landscape of real-time data processing.

At the heart of this paradigm shift lies the aspiration to harness the power of data as it arrives, to glean actionable insights and drive informed decisions in the blink of an eye. This aspiration fuels the core objectives of our project: to design and implement a real-time stream processing solution leveraging Azure Databricks and Apache Spark. Our aim is to empower organizations to seamlessly ingest, process, and analyze streaming data in real-time, ushering in a new era of agility and responsiveness.

Central to our endeavor is a comprehensive exploration of stream processing concepts, delving into architectures, event-driven processing, and stateful stream processing. We seek to unravel the significance of real-time data processing in enabling organizations to detect and respond to events and trends as they unfold, driving competitive advantage and operational efficiency.

Moreover, our project embarks on an in-depth exploration of Azure Databricks and Apache Spark features, unraveling their capabilities for real-time data processing. We aim to understand the seamless integration of Azure Databricks with other Azure services such as Event Hubs, Synapse Analytics, and Machine Learning, unlocking the full potential of these platforms for our real-time stream processing solution.

Our journey extends to the implementation of data ingestion pipelines, leveraging Azure Event

Hubs and other streaming data sources to seamlessly feed streaming data into Azure Databricks. Here, Apache Spark Streaming takes center stage, enabling real-time data transformation aggregation, and analysis with unparalleled efficiency and speed.

As we navigate the intricacies of real-time stream processing, we place a premium on state management and fault tolerance. Leveraging Apache Spark's state management capabilities, we endeavor to maintain and update state information across streaming data, ensuring data reliability and resilience in the face of failures.

Furthermore, our project is imbued with a commitment to monitoring and management, as we implement robust capabilities within Azure Databricks to track streaming data ingestion rates, processing latency, and system health. Configuring alerts, logging, and diagnostic tools, we endeavor to ensure the reliability and availability of our real-time stream processing solution in the face of evolving challenges.

Finally, our journey culminates in a rigorous evaluation and validation of our real-time stream processing solution. Through meticulous testing, we seek to assess its performance, scalability, and reliability under varying data volumes, processing loads, and streaming data rates, ensuring its readiness to meet the demands of today's dynamic data landscape.

1.2 SCOPE OF THE PROJECT

The scope of our project entails crafting a comprehensive real-time stream processing solution using Azure Databricks and Apache Spark. We delve into the complexities of stream processing, exploring architectures, event-driven processing, and stateful stream processing. Our focus extends to a thorough investigation of Azure Databricks and Apache Spark features, particularly their integration with Azure services like Event Hubs, Synapse Analytics, and Machine Learning.

Central to our endeavor is the development of robust data ingestion pipelines, seamlessly ingesting streaming data into Azure Databricks from sources like Azure Event Hubs. Leveraging Apache Spark Streaming, we will enact real-time data transformation, aggregation, and analysis

to extract actionable insights from the streaming data. Additionally, we will prioritize the implementation of state management and fault tolerance mechanisms to ensure data reliability and resilience in the face of failures.

Our project further encompasses deploying comprehensive monitoring and management capabilities within Azure Databricks. This involves configuring alerts, logging, and diagnostic tools to monitor streaming data ingestion rates, processing latency, and system health, ensuring the reliability and availability of the real-time stream processing solution.

Finally, rigorous testing and validation will assess the performance, scalability, and reliability of our solution under varying data volumes, processing loads, and streaming data rates.

1.3 OBJECTIVE

The objective of our project is to design and implement a real-time stream processing solution using Azure Databricks and Apache Spark. We aim to explore the fundamental concepts of stream processing, including architectures and stateful processing, to understand their significance in the realm of real-time data analysis. Additionally, our project involves a detailed investigation of the features and capabilities offered by Azure Databricks and Apache Spark for real-time data processing, with a particular focus on their seamless integration with other Azure services. Furthermore, we endeavor to develop robust data ingestion pipelines capable of seamlessly ingesting streaming data into Azure Databricks from a variety of sources. Finally, our objective includes the implementation of stream processing logic using Apache Spark Streaming, enabling real-time data transformation and analysis to derive actionable insights promptly.

1.4 EXISTING SYSTEM

Kafka Streams

1.4.1Integrated into the Apache Kafka ecosystem.

1.4.2Enables real-time processing of data streams.

1.4.3Provides a high-throughput, fault-tolerant messaging system.

Allows for parallel processing of data streams.

1.4.4Offers seamless integration with other Apache Kafka components.

Supports both stateless and stateful processing operations.

1.4.1 EXISTING SYSTEM DISADVANTAGES

Apache Storm

- **Complexity:** Setting up and configuring Apache Storm can be complex, requiring expertise in distributed systems and programming.
- **Learning Curve**: Users may face a steep learning curve, especially those unfamiliar with stream processing paradigms and concepts.
- **Operational Overhead:** Managing and monitoring Apache Storm clusters can require significant operational overhead, including resource allocation, fault tolerance configuration, and performance tuning.
- **Limited Ecosystem:** While Apache Storm integrates well with other Apache projects like Kafka and Hadoop, its ecosystem may not be as extensive as other streaming platforms like Apache Flink.
- **Performance Bottlenecks**: In certain use cases or configurations, Apache Storm may experience performance bottlenecks, leading to increased latency or decreased throughput.

1.5 LITERATURE SURVEY

Title: Real-Time Processing of Big Data Streams: Lifecycle, Tools, Tasks, and Challenges

Author: Fatih Gürcan, Muhammet Berigel

Year: 2018

Description:

In line with the developments formed under the leadership of information and communication technologies (ICTs), big data has become one of the most rapidly growing trends in recent years. From this perspective, big data is considered by many authorities as tomorrow's data architecture, and also it is regarded as one of the top 10 crucial technologies that will change the world. In a general sense, "big data" is defined by the three fundamental features including volume, variety, velocity. These dimensions describe the landscape of big data. Through the efficient use of many online resources such as internet of things (IoT), mobile devices, social networks, and sensors, the number of real-time applications of big data streams has increased considerably. The introduction of big data technologies has led to a great transformation in the methodologies of data progressing and analytics. In today's competitive business environments, the progressing and analytics of big data plays an important role in achieving successful business strategies. Big data processing is becoming a reality in many real-world applications and solutions. In general sense, the big data processing can be categorized into three distinct types, comprising batch processing, real-time processing, and hybrid (batch and real-time) processing. Batch data processing is a well-organized technique of processing high volumes of data.

1.5.1 REAL TIME PROCESSING OF BIG DATA STREAMS

A data stream is a continuous, real-time, and unbounded series of data items. The process of streaming divides non-stop flowing input data into distinct units for advanced processing. Stream processing is a low-latency processing approach and analyzing of streaming data. Stream processing is about real-time processing of nonstop streams of data in a workflow. Real-time processing, stream processing, and streaming processing are often used interchangeably. Real-time processing requires the continual and sequential transactions for limitless streams of input data. In real-time processing, it is essential that the big data streams be processed with very short latency, measured in seconds or milliseconds. So as to accomplish very short latency, big data streams are processed as small sets of input data stored in memory. In other words, the real-time processing can also be stated as a sequence of repeated operations, in which the data streams transferred to the memory instead of the disks are processed as small sets. Real time processing is crucial in order to continue the high-level functionality of nonstop operated or automated systems having intensive data streams. Numerous platforms, applications, and tools necessitate real-time processing of big data streams having different data structures. Radar systems, smart cities, disaster management systems, internet of things, bank ATMs, and social networks are remarkable examples for the applications of real-time processing.

1.5.2 LIFECYCLE OF REAL-TIME BIG DATA PROCESSING

In today's big data platforms, in a general sense, big data processing consists of the following stages: data acquisition, data storage, data analysis, and data reporting. The organization and application of these phases may vary depending on the background of the data processing model. Big data lifecycle can be applied in various ways for batch processing or real time processing. In terms of real-time big data processing, the lifecycle consists of phases with continuity and time-limited. Real time processing focuses on the big data streams that are ingested in real-time and processed with minimal latency. Since the data stream is continuous in this paradigm, the phase of data storage is usually implemented at the end of the life cycle. Contrary to the common big data lifecycle, the real-time processing lifecycle for big data streams has five conceptual phases, including real-time data ingestion, data storage, stream processing, analytical data store, and analysis and reporting.

• **Data ingestion:** This phase is the process in which big data is ingested from heterogeneous data sources. In data ingestion process, lots of real-time processing paradigms use a message ingestion store so as to act as a buffer for messages. In contrast batch processing systems, real-time ingestion can available a stream of data. The data stream is started and continued systematically.

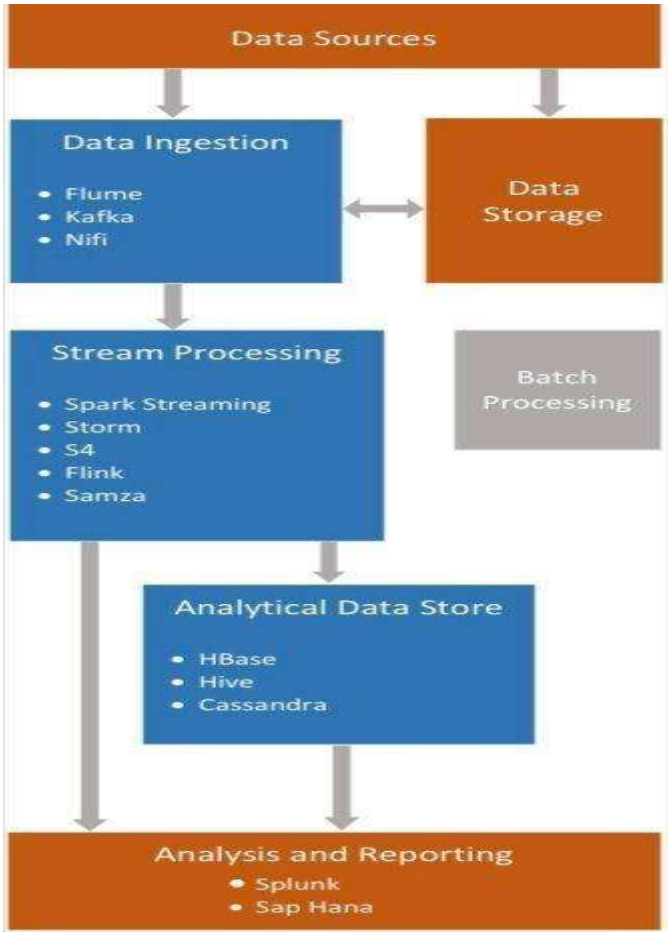


Fig. Lifecycle of real-time big data processing

• **Data storage:** This phase covers the operations for storage of real-time data streams having different data structures. In the majority of real-time big data applications, the data storage operations are also performed as the analytical data storage at the end of the stream processing.

• **Stream processing:** After the data ingestion, real-time big data streams are processed and structured for real-time analytics and decision-making. In this phase, various frameworks and paradigms are used according to the nature of the real-time applications.

• **Analytical data store:** In a big data lifecycle, analytical data store is often needed in order to store and serve processed data in a structured format. This process makes it also possible to query the data using analysis tools.

• **Analysis and reporting:** In this last phase of the big data lifecycle, it is aimed to provide implications and insights for effective decision-making through analysis and reporting.

1.5.3 TOOLS AND TASKS FOR REAL-TIME BIG DATA PROCESSING

This section analyzes the frameworks (platforms, tools, and technologies) are used in the realtime processing of big data streams. Taking into account the processes in the lifecycle, the most common 11 frameworks are discussed and their roles and tasks in the lifecycle are identified. And finally, these tools and tasks are compared.

A. Data Ingestion:

• **Flume:** Apache Flume is a reliable, distributed, and obtainable data ingestion system for gathering, combining and transferring large volumes of data streams such as events, log data (etc...) from many distinct data sources to a centralized data store. It has a simple and modular architecture that provides the online analytic application for data streams.

• **Kafka:** Apache Kafka is a distributed streaming framework having three main capabilities: First, publish, distribute, and subscribe the streams of records like a message queue or real-time messaging organization. Second, store data streams with faulttolerant is a robust approach. Third, process streams of events and logs. Kafka delivers reliable and low latency responses to support real-time applications and to connect streaming data systems. Kafka can be used for real-time event processing and integration of modules of large-scale software systems. Compared to Flume, Kafka provides better scalability and message consistency.

B. Stream Processing:

• **Storm:** Apache Storm is an advanced big data computation framework that allows thereal-time processing and distributed computation of data streams at a high speed measured in milliseconds. It is designed to process huge volume of data in a faulttolerant and horizontal scalable methodology. Storm can be easily integrated and implemented with any programming

language. Due to its flexible features, Storm continues to be a leading paradigm in real-time processing of big data streams.

• **Spark streaming:** It is a Spark technology for realtime processing, which is named Spark streaming, similar to Spark for batch processing. Spark streaming makes it easy to build scalable streaming applications of live data streams with fault-tolerant. It delivers the real-time processing similar to a sequence of very short batch tasks. In Spark streaming paradigm, data streams are received as live input and separated the data into small batches. After that, the batches are processed by the Spark engine in order to generate the final stream of data outcomes in batches.

1.5.4 CHALLENGES

In modern day environments, big data is mostly described by its main characteristics such as large volume, variety, and heterogeneity of data in which all processes require new technologies and methodologies in order to extract valuable information from it. In this phase, the essential challenges discussed in this study were grouped under three main categories: characteristic challenges, processing challenges, and management challenges, as shown in Fig. In the rest of the article, the challenges are discussed in more detail and specified under eight distinct headings.

A. Volume, Variety and Heterogeneity:

Unstructured data can contain all types of data in high volume and different formats. These data may include different information sharing platforms such as social media, forum, e-mail, chat, online communities, online shopping, and so on. Moreover, the data may include a combination of different file formats such as picture, video, audio, and text. The analytical processes on the data are more difficult and costly due to this complex structure of the data. This heterogeneous and complex structure of big data is a serious challenge that must be overcome in big data analytics process.

B. Data Capture and Storage:

In today's digital environments, it is generated daily 2.5 quintillion bytes of data, and this amount is ever-increasing day by day. The collection and storage of the data in this very large volumes can be performed by various technical processes that require large costs. Several log records are periodically deleted in different data sources, because of the cost of required storage systems. The advent of big data models changes the traditional data capture and storage methodologies,

data access systems, and data-driven implementations. Because existing storage technologies cannot provide to necessary performance in terms of real time data collection, processing, and storage. Big data processing and analytics requires high-speed data input/output (I/O) access patterns, because these processes cannot be achieved with current hard disk drives (HDDs) based on random I/O access.

C. Inconsistency and Incompleteness:

Big data is a combination of data obtained from different data sources in different formats. In this case, data inconsistency and incompleteness may occur in the processing of data obtained from different source, and so data processing may not be achieved in desired level. Considering the heterogeneous nature of big data, at the start, the reliability of the data collected from different sources should be increased by verifying as temporal and spatial. After that, a common homogeneous data structure should be determined and the data obtained from different sources should be combined on monotype basis. Also, data transfer process must be checked regularly to ensure whether the process is complete or not. Thus, the inconsistency and incompleteness errors may be controlled during big data processing.

D. Scalability:

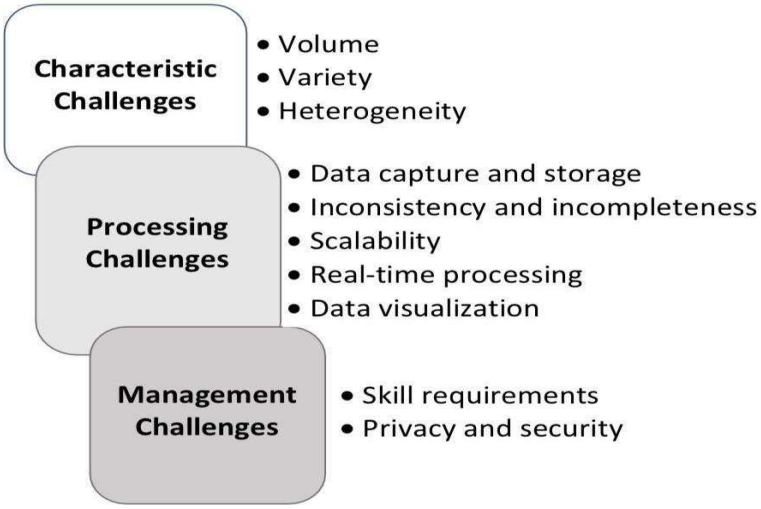
Recently, the optimization of rapidly growing data emerges a challenging process for providing scalability, a remarkable challenge should be solved in up-to-date database systems. It is thought to be a scalable system, whether the system's performance and capacity relatively growths after installing a hardware or application. Big data scalability is based on the essentials of scaling databases from a single node to large clusters [13]. One process employed by most of the major database management system (DBMS) applications is the partitioning of large tables, based on arrays of values in a key field.

E. Real-Time Processing:

As volume and variety of data increase, timeliness on big data processing becomes a more complicated challenge. The timeliness can be defined as the time delay between generating and ingestion of data. Data should be obtainable in this period in order to achieve for an effective analytics. For example, in a traffic monitoring system watching millions of vehicles instantly, finding the alternative ways and calculation of the arrival times require the real time data processing. In this respect, any delay or miscalculation may misdirection an ambulance, in this aspect, timeliness has a vital importance in terms of human life.

F. Data Visualization:

Data Visualization is used to report information clearly and effectively to users by employing different presentation techniques such as graphs, tables, charts, and animations. It can be helpful for users to understand and interpret the great amounts of processed data in a simpler way. Big Data visualization is not as easy as traditional small data sets. In complex and large-scale data sets, the visualization process requires many innovative approaches and components. Determination of the most suitable data presentation model is crucial in order to achieve an understandable visualization. The progression of traditional visualization approaches is relatively remarkable, but not at the desired level. In large scale data visualization, many generative techniques such as feature extraction and geometric modeling are suggested for data visualization and presentation.



G. Skill Requirements:

The big data industry is an emerging and innovative job market in which labor resources are employed effectively. In this competitive environment, in-demand skills, competencies, and requirements are ever-changing and progressing. Big data specialists are expected to have a wide spectrum of knowledge and skills together big data labor market. For this reason, big data specialists should always progress their knowledge and skills up to date. As big data industry advances and business demands increase, new career opportunities are opening up every day for big data specialists, and therefore skilled labor shortage is ever increasing in these days. The scarcity of big data talent will be a significant challenge in order to meet emerging market demands in this field.

H. Privacy and Security:

Some specific information such as individual health information, online shopping memberships, and bank account numbers, and social media profiles require a high level of protection in terms of privacy and security. In this regard, the most important threat for the security of personal information is that the personal data is irregularly accumulated by the numerous social media platforms. Undoubtedly, specific big data resources must be safeguarded by laws and regulations in terms of privacy and security. International Data Corporation (IDC) recommended five levels of increasing security: privacy, compliance driven, custodial, confidential, and lockdown. The private personal information of a person when combined with that person's shares provide the significant facts and details about the person.

1.5.5 CONCLUSION

This paper provides a brief overview on real-time processing of big data streams, with its lifecycle, tools and tasks and challenges. This paper initially revealed the lifecycle of real-time big data processing, consisting of four phases, that are data ingestion, data processing, analytical data store, and analysis and reporting. Secondly, it described tools and tasks of real-time big data processing. These tools are: Flume, Kafka,Storm, Spark Streaming. And finally, challenges of real-time big data processing were identified and categorized. The challenges are: “volume, variety and heterogeneity”, “data capture and storage”, “inconsistency and incompleteness”, “scalability”, “real-time processing”, “data visualization”, “skill requirements”, and “privacy and security”. This paper may provide valuable insights into: 1) companies, in employing a qualified big data workforce and in integrating new big data paradigm into evolving business strategies; 2) big data professionals, in assessing and improving their own qualifications; 3) academic communities, in designing big data programs and curricula in line with emerging trends and technologies. Entering the era of big data, in order to achieve more effective processing and analytics of big data streams and to take advantage of the opportunities of big data realm, the current approach and paradigms of big data processing should be analyzed in a comprehensive manner. To achieve this, it seems that more research and applications are needed about processing big data streams effectively.

1.6 PROPOSED SYSTEM

- The proposed system leverages Azure Databricks as a unified data platform to address the challenges of traditional systems and offer a streamlined approach to real-time data processing. Azure Databricks combines the capabilities of Apache Spark with the flexibility and scalability of the cloud, providing a more efficient and integrated solution for data processing, analysis, and storage.
- In the proposed system, data is ingested from various sources such as databases, cloud storage, and streaming platforms like Apache Kafka and Azure Event Hubs. Azure Databricks offers native support for these data sources, enabling seamless data ingestion and processing within the platform.
- Once data is ingested, Azure Databricks provides a collaborative environment for data scientists, engineers, and analysts to work with data using languages such as Python, Scala, R, and SQL. The platform's in-memory processing capabilities enable fast and efficient data transformations, aggregations, and complex operations like joins and window functions.
- Azure Databricks supports both real-time and batch processing modes. Spark Streaming in Azure Databricks allows for continuous data processing, providing real-time insights and timely responses to incoming data. Batch processing is used for larger, scheduled data processing tasks, enabling the handling of substantial data volumes at once.
- The platform offers built-in tools for data visualization, making it easy to explore and interpret data directly within the environment. Additionally, Azure Databricks supports advanced analytics, including machine learning model training and deployment, allowing users to derive actionable insights from processed data.
- The proposed system's integration with other Azure services, such as Azure Data Lake Storage and Azure Machine Learning, provides a seamless end-to-end data processing and analysis pipeline. The system also ensures scalability, fault tolerance, and resource efficiency, addressing the limitations of traditional systems.

1.6.1 PROPOSED SYSTEM ADVANTAGES

- The proposed system that utilizes Azure Databricks offers several advantages over traditional data processing frameworks and earlier versions of streaming solutions. These advantages make the platform an ideal choice for modern data processing needs and real-time analytics:
- Unified Platform: Azure Databricks combines data processing, analysis, and storage into a single, unified platform. This eliminates the need for disparate systems and complex integrations,streamlining data workflows and reducing operational overhead.
- Scalability and Performance: The platform is built on Apache Spark, providing efficient, in-memory data processing capabilities. It can scale elastically to handle large volumes of data, offering high performance even for real-time streaming data and batch processing tasks.
- Collaboration and Productivity: Azure Databricks offers collaborative workspaces where data scientists, engineers, and analysts can work together seamlessly. The platform supports multiple programming languages (Python, Scala, R, and SQL), enhancing productivity and enablingteams to use the languages they are most comfortable with.
- Real-Time Processing: Spark Streaming in Azure Databricks allows for continuous data processing, providing real-time insights and timely responses to incoming data streams. This is particularly beneficial for applications such as IoT, financial trading, and social media analytics.
- Advanced Analytics and Machine Learning: Azure Databricks integrates with Azure Machine Learning, enabling advanced analytics and model training directly within the platform. Users can leverage pre-built libraries and tools for machine learning and AI to derive actionable insights from processed data.
- Integration with Azure Services: Azure Databricks integrates seamlessly with other Azure services, such as Azure Data Lake Storage, Azure Blob Storage, and Azure SQL Database. This enables a smooth end-to-end data processing and analysis pipeline, as well as easy access to data for further processing and analysis.

CHAPTER 2

PROJECT DESCRIPTION

2.1 GENERAL:

The project focuses on developing a real-time stream processing solution using Azure Databricks and Apache Spark. This solution aims to empower organizations to handle streaming data in real-time, facilitating ingestion, processing, and analysis for timely insights and actionable intelligence. By leveraging the stream processing capabilities of Apache Spark and the scalability of Azure Databricks, the project addresses the growing need for real-time data processing in organizations. Key components include understanding stream processing concepts, exploring Azure Databricks and Apache Spark features, implementing data ingestion and processing pipelines, managing state and fault tolerance, and establishing monitoring and management capabilities. Ultimately, the project aims to deliver a reliable and scalable solution for organizations to make informed decisions and take prompt actions based on insights derived from streaming data.

2.2 METHODOLOGIES

2.2.1 MODULES NAME:

- **DATA INGESTION MODULE**
- **DATA TRANSFORMATION MODULE**
- **STREAMING ANALYTICS MODULE**
- **DATA DELIVERY MODULE**

2.2.2 MODULES EXPLANATION:

- **Data Ingestion Module:**
  - This module is responsible for retrieving streaming data from various sources inreal-time.
  - ADF supports a wide range of connectors for ingesting data from diverse sources

such as:

- Internet of Things (IoT) devices that generate sensor data streams.
- Social media feeds containing real-time updates and audience interactions.
- Sensor networks deployed in industrial settings to monitor equipment performance and environmental conditions.
- Application logs that capture user activity and system events within web applications or APIs.

- The specific configuration of this module will depend on the chosen data sources for your project. For instance, if the project focuses on analyzing social media sentiment, the data ingestion module would utilize connectors for specific social media platforms like Twitter or Facebook.

- **Data Transformation Module:**

- After data is ingested, this module prepares the data for analysis.
- Transformations may involve tasks like:
  - Data cleansing to remove inconsistencies or errors present in the raw data stream. This could involve identifying and handling missing values, outliers, or corrupt data points.
  - Data formatting to ensure compatibility with the chosen streaming analytics engine. Data formats may vary depending on the source and the processing requirements. Common data formats used in streaming analytics include JSON, Avro, and CSV.
  - Feature engineering to create new attributes that are more meaningful for analysis. This could involve deriving new features from existing data points or combining data from multiple sources to create a richer dataset.
- ADF provides a visual interface for building data transformation pipelines using

pre-built activities or custom code. This allows users to build complex transformations without requiring extensive programming expertise.

- **Streaming Analytics Module:**

  - This module performs real-time analysis on the transformed data stream.
  - Azure Stream Analytics, a fully managed service integrated with ADF, is used for this purpose.
  - Stream Analytics allows users to write queries in Stream Analytics Query Language (SAQL), a SQL-like language with extensions specifically designed for real-time data processing.
  - Queries within this module can perform operations that extract valuable insights from the data stream, including:
    - Filtering data based on specific criteria to focus on relevant portions of the stream. This could involve filtering data by location, user ID, or any other relevant attribute.
    - Aggregating data to identify trends and patterns over time. For example, calculating average sensor readings, user engagement metrics, or website traffic statistics.
    - Detecting anomalies or outliers within the data stream that may indicate potential issues or critical events. This could involve identifying sudden spikes in sensor readings, deviations from historical trends, or unusual user behavior patterns.
    - Joining data streams from multiple sources for holistic analysis. Combining data from various sources can provide a more comprehensive understanding of the situation. For instance, joining social media sentiment data with website traffic data can reveal correlations between user sentiment and website engagement.

- **Data Delivery Module:**

  - The final module delivers the results of the streaming analytics to appropriate destinations for further processing or visualization.

  - ADF offers connectors to various data stores and analytics platforms, providing flexibility in how the insights are utilized:

    - Azure Data Lake Storage: A scalable data lake for storing the raw or processed data streams for future analysis or historical reference.

    - Azure SQL Database: A managed relational database service for storing aggregated data or key metrics derived from the real-time analytics.

    - Power BI: A business intelligence platform for creating interactive dashboards and reports that visualize the real-time insights in an easily understandable format for stakeholders.

  - The chosen destination will depend on the specific needs of your project and how the actionable insights will be used to inform decision-making.

2.3 TECHNIQUES USED OR ALGORITHM USED

1. Apache Spark:

Azure Databricks is built on Apache Spark, an open-source, distributed computing framework that supports large-scale data processing. Spark provides powerful in-memory processing capabilities, enabling fast data transformations and analytics. It supports a wide range of operations, including filtering, mapping, reducing, joining, and aggregating data.

2. Azure Blob Storage:

Blob storage is a cloud-based storage solution provided by Azure that allows for the storage of large volumes of unstructured data, such as text, images, videos, and other multimedia files. Blob storage is often used for data lakes, where raw data is stored before processing and analysis.

3. Azure Databricks:

Databricks is a unified data analytics platform that offers tools and services for data engineering, data science, and machine learning. Built on Apache Spark, Databricks provides a collaborative workspace where data engineers, data scientists, and business analysts can work together using multiple programming languages such as Python, Scala, R, and SQL. Its scalability and performance make it ideal for handling large datasets efficiently.

4. Distributed Data Processing in Spark:

Distributed data processing in Spark is achieved through a resilient distributed dataset (RDD) abstraction that enables efficient data manipulation and transformation across a cluster of computers. Spark's execution model is based on the concept of dividing the data processing workload into smaller tasks that can be executed in parallel across multiple nodes in a cluster.

2.3.1 EXISTING TECHNIQUE:

Apache Kafka is frequently integrated with various technologies to support real-time data processing and streaming pipelines. One common approach is pairing Kafka with Apache Spark Streaming, enabling scalable, fault-tolerant stream processing. Spark Streaming can consume data directly from Kafka topics, leveraging Spark's distributed processing capabilities for tasks like filtering, aggregating, and joining streaming data. Another technique involves using Kafka Connect, which facilitates seamless data integration between Kafka and external data systems by providing connectors for different databases, storage platforms, and messaging systems. For direct stream processing applications, Kafka Streams offers a lightweight client library to build stateful stream processing tasks against Kafka topics. Moreover, Kafka serves as a foundational component in microservices architectures, facilitating event-driven communication among microservices. Additionally, Kafka is suitable for implementing event sourcing and Command Query Responsibility Segregation (CQRS) patterns, storing state change events in Kafka topics to maintain system consistency and enable efficient data querying. Overall, these techniques showcase Kafka's versatility in enabling scalable and robust real-time data processing architectures.

2.3.2 PROPOSED TECHNIQUE USED:

The proposed technique, referred to as the Fusion Method, aims to enhance data processing and analysis within the Azure Databricks environment by integrating multiple data sources and leveraging various data processing approaches. This method blends the strengths of different frameworks such as Spark's in-memory computing and Databricks' optimized execution engine. By uniting these powerful technologies, the Fusion Method facilitates efficient and comprehensive data analysis that combines both batch and streaming data processing.

The Fusion Method supports real-time analysis of incoming data while also managing large-scale data transformations and aggregations. This dual capability allows for immediate insights from streaming data as well as the handling of significant data sets in batch processing mode. As a result, users can achieve timely and efficient data processing across various sources, including structured, semi-structured, and unstructured data, which is crucial for drawing meaningful insights and making informed decisions.

In addition, the Fusion Method enhances the potential for advanced analytics, such as machine learning and predictive modeling, by ensuring data is well-prepared and harmonized for analysis. This approach enables businesses to leverage data-driven strategies for competitive advantage, providing greater accuracy and efficiency in decision-making processes. By integrating the various aspects of data processing, the Fusion Method serves as a comprehensive solution for modern data challenges in Azure Databricks.

CHAPTER 3
REQUIREMENTS ENGINEERING

3.1 GENERAL

The project aims to enhance real-time stream processing techniques by developing advanced methods for data classification. These methods will improve the accuracy and speed of classification, facilitating precise analysis and decision-making.

The goal is to enable efficient processing of streaming data for applications such as anomaly detection, trend analysis, and predictive modeling. By enhancing classification techniques, the project seeks to optimize information retrieval, enhance sound classification, and improve forecasting accuracy, particularly in domains such as finance and stock market analysis.

3.2 HARDWARE REQUIREMENTS

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It should what the system does and not how it should be implemented.

- PROCESSOR : intel core i5
- RAM : 4GB DDR RAM
- HARD DISK : 256 GB

3.3 SOFTWARE REQUIREMENTS

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification.

It is useful in estimating cost, planning team activities, performing tasks and tracking the teams and tracking the team's progress throughout the development activity.

- SOFTWARE TOOLS : AZURE DATA FACTORY ACCOUNT.
- OPERATING SYSTEM : WINDOWS 11

3.4 FUNCTIONAL REQUIREMENTS

A functional requirement defines a function of a software-system or its component. A function is described as a set of inputs, the behavior, Firstly, the system is the first that achieves the standard notion of semantic security for data confidentiality in attribute-based deduplication systems by resorting to the hybrid cloud architecture.

3.5 NON-FUNCTIONAL REQUIREMENTS

The major non-functional Requirements of the system are as follows

Usability

The system is designed with completely automated process hence there is no or less user intervention.

Reliability

The system is more reliable because of the qualities that are inherited from the chosen platform python. The code built by using python is more reliable.

Performance

This system is developing in the high level languages and using the advanced back-end technologies it will give response to the end user on client system with in very less time.

Supportability

The system is designed to be the cross platform supportable. The system is supported on a wide range of hardware and any software platform, which is built into the system.

Implementation

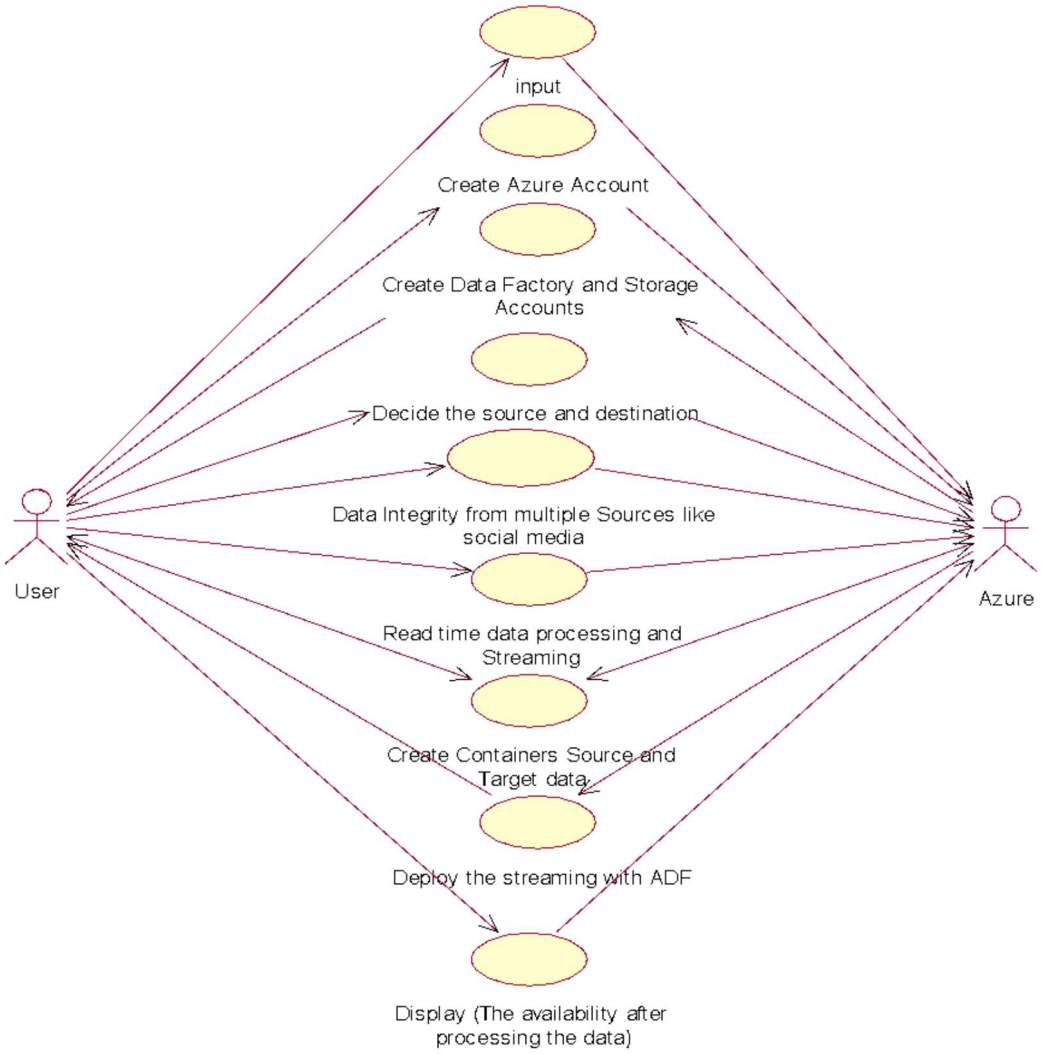
The system is implemented in web environment using Jupyter notebook software. The server is used as the intellignce server and windows 10 professional is used as the platform. Interface the user interface is based on Jupyter notebook provides server system.

CHAPTER 4
DESIGN ENGINEERING

4.1 GENERAL

Design Engineering deals with the various UML [Unified Modelling language] diagrams for the implementation of project. Design is a meaningful engineering representation of a thing that is to be built. Software design is a process through which the requirements are translated into representation of the software. Design is the place where quality is rendered in software engineering. Design is the means to accurately translate customer requirements into finished product.

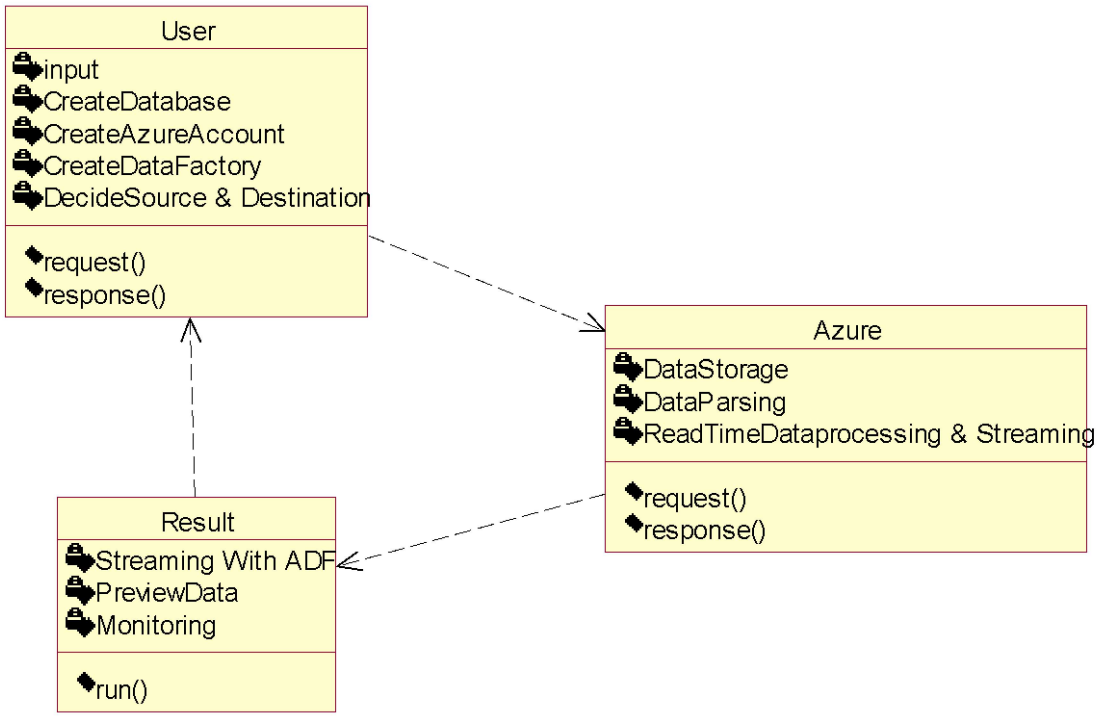
4.2 USE CASE DIAGRAM



EXPLANATION:

The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted. The above diagram consists of user as actor. Each will play a certain role to achieve the concept.

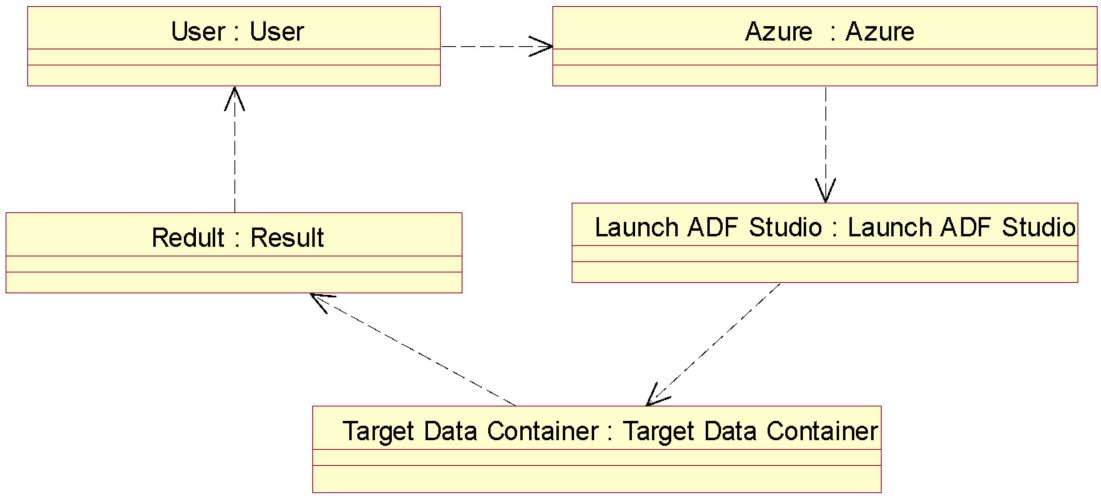
4.3 CLASS DIAGRAM



EXPLANATION

In this class diagram represents how the classes with attributes and methods are linked together to perform the verification with security. From the above diagram shown the various classes involved in our project

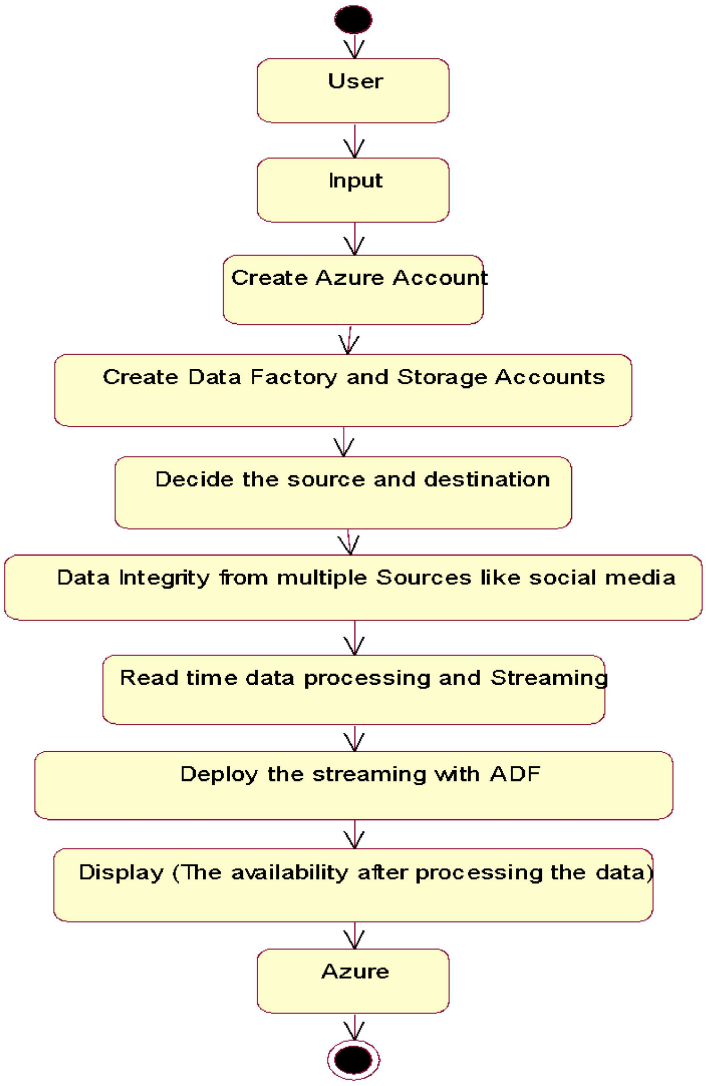
4.4 OBJECT DIAGRAM



EXPLANATION:

In the above digram tells about the flow of objects between the classes. It is a diagram that shows a complete or partial view of the structure of a modeled system. In this object diagram represents how the classes with attributes and methods are linked together to perform the verification with security.

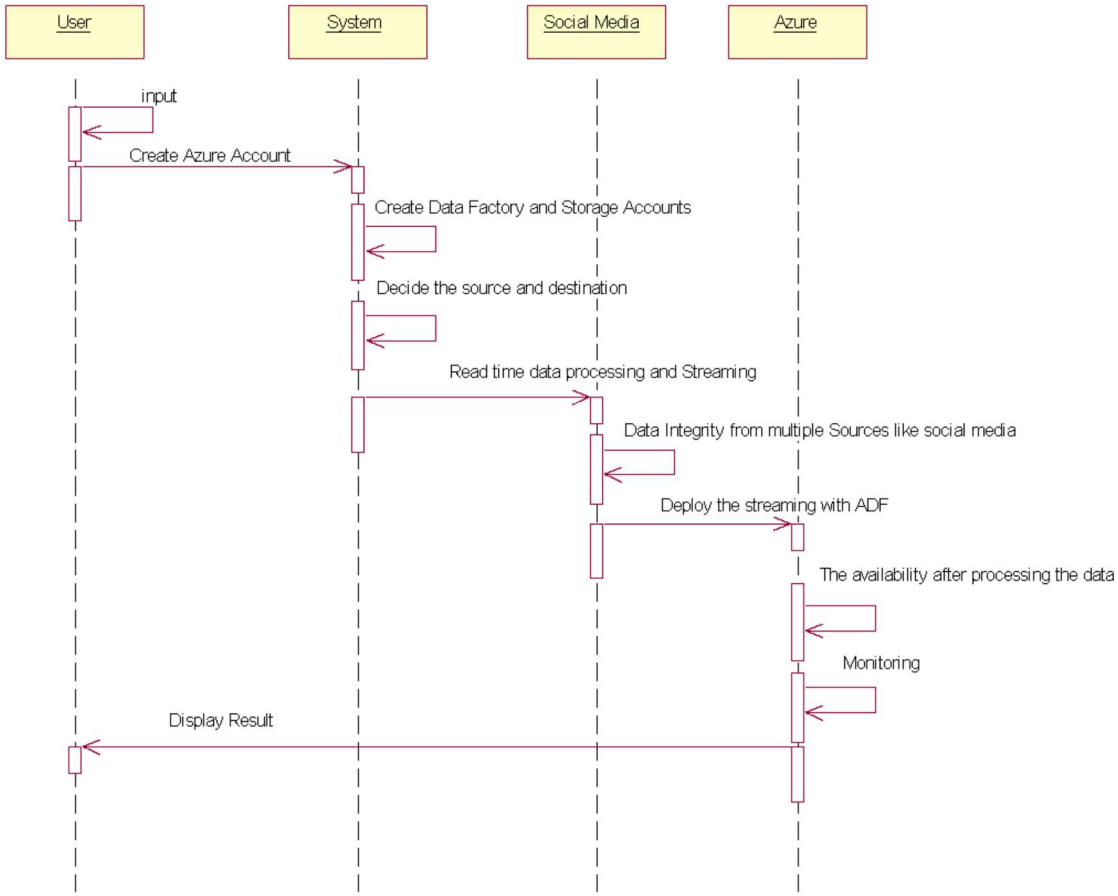
4.5 STATE CHART DIAGRAM



EXPLANATION:

State diagram are a loosely defined diagram to show workflows of stepwise activities and actions, with support for choice, iteration and concurrency. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction. Many forms of state diagrams exist, which differ slightly and have different semantics.

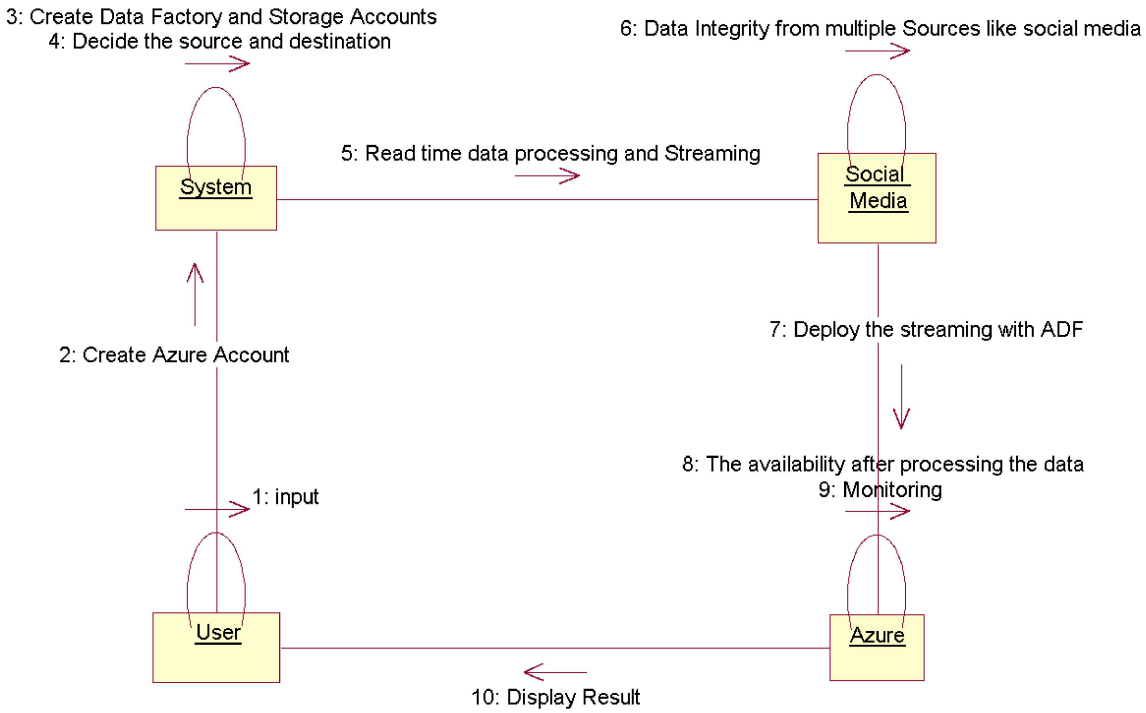
4.6 SEQUENCE DIAGRAM



EXPLANATION:

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

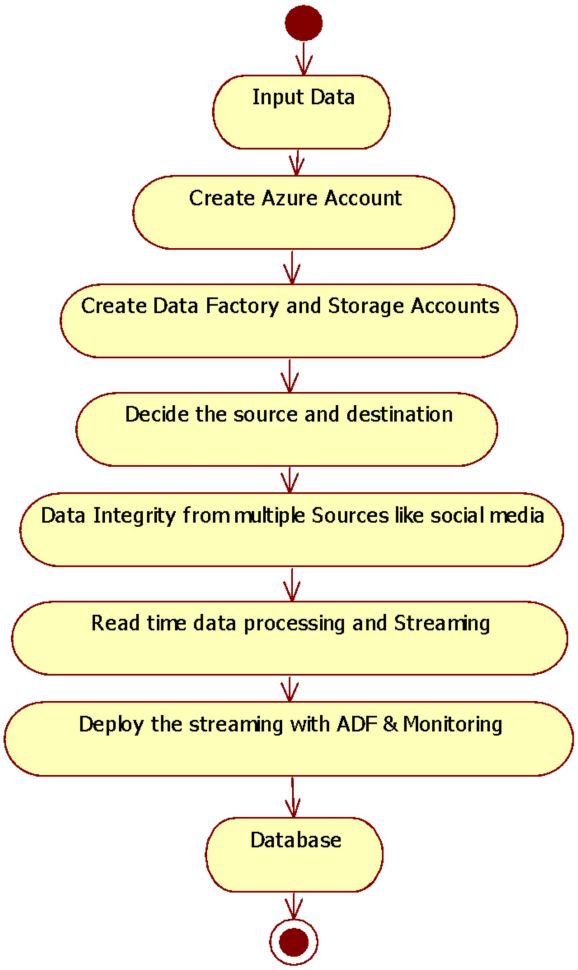
4.7 COLLABORATION DIAGRAM



EXPLANATION:

A collaboration diagram, also called a communication diagram or interaction diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). The concept is more than a decade old although it has been refined as modeling paradigms have evolved.

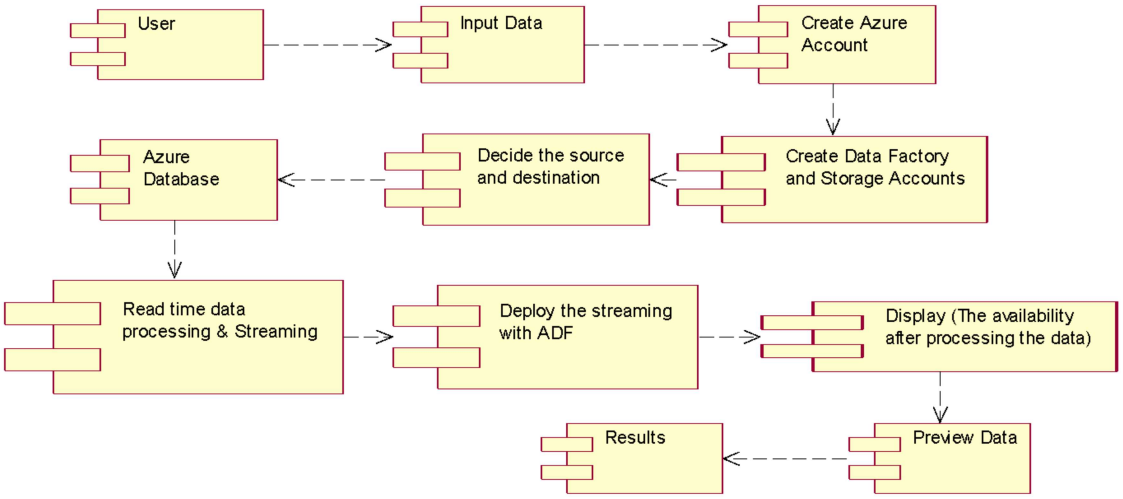
4.8 ACTIVITY DIAGRAM



EXPLANATION:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

4.9 COMPONENT DIAGRAM

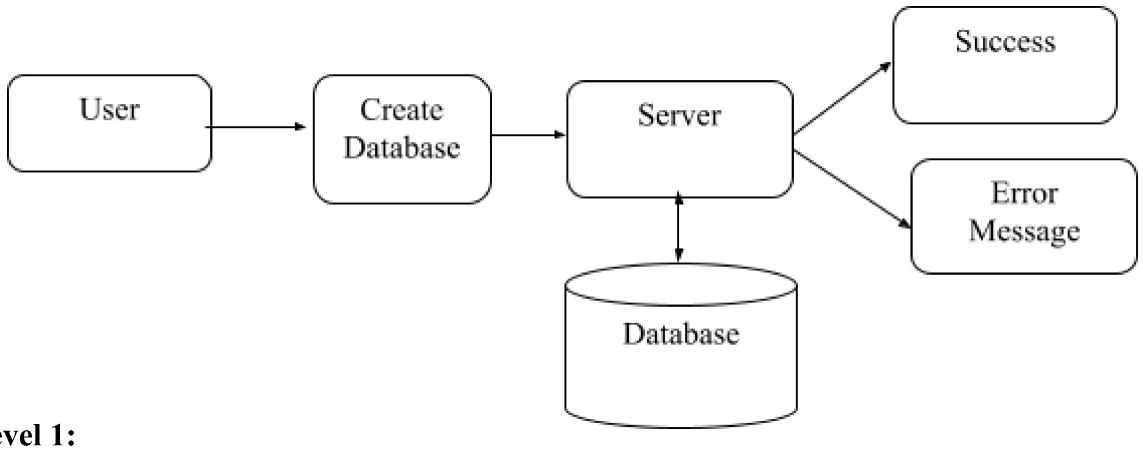


EXPLANATION:

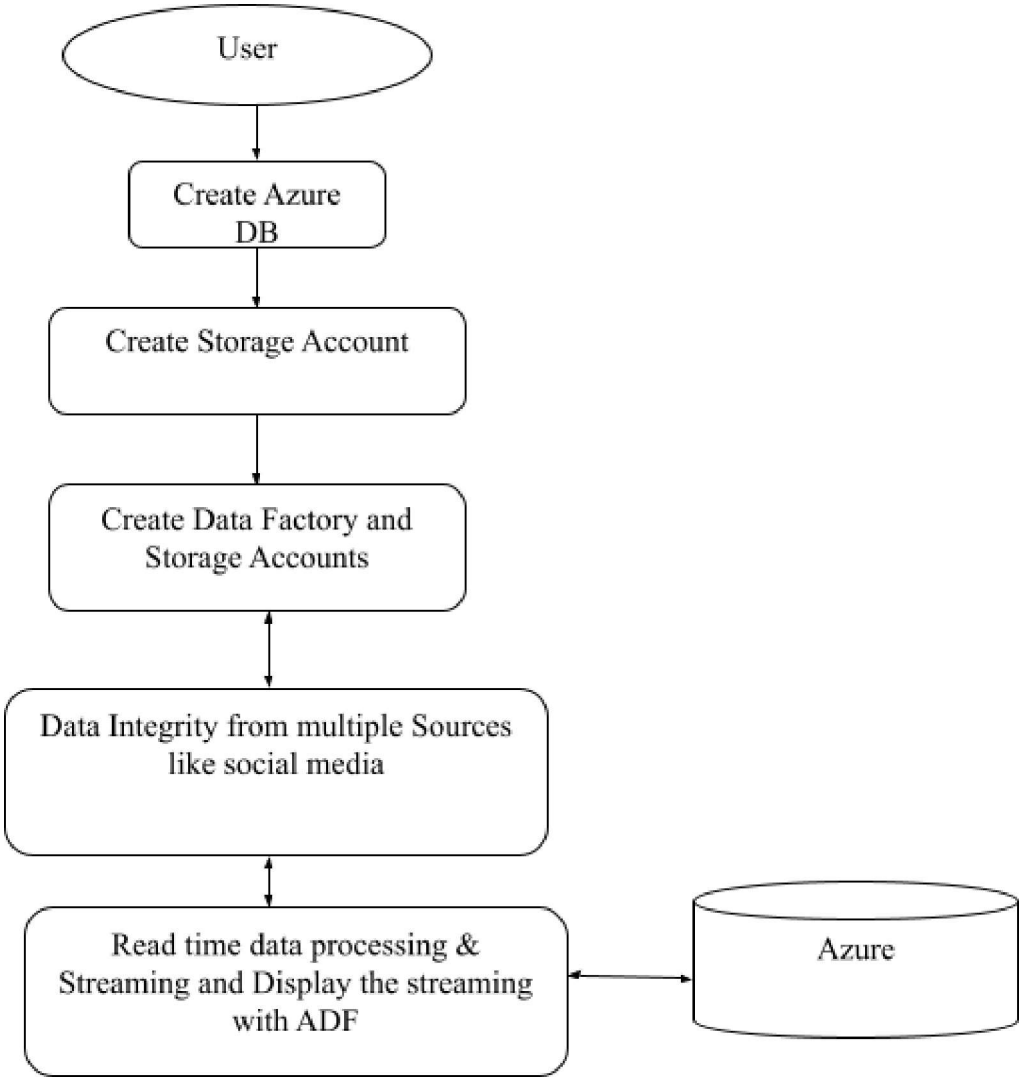
In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems. User gives main query and it converted into sub queries and sends through data dissemination to data aggregators. Results are to be showed to user by data aggregators. All boxes are components and arrow indicates dependencies.

4.10 DATA FLOW DIAGRAM:

Level 0:



Level 1:

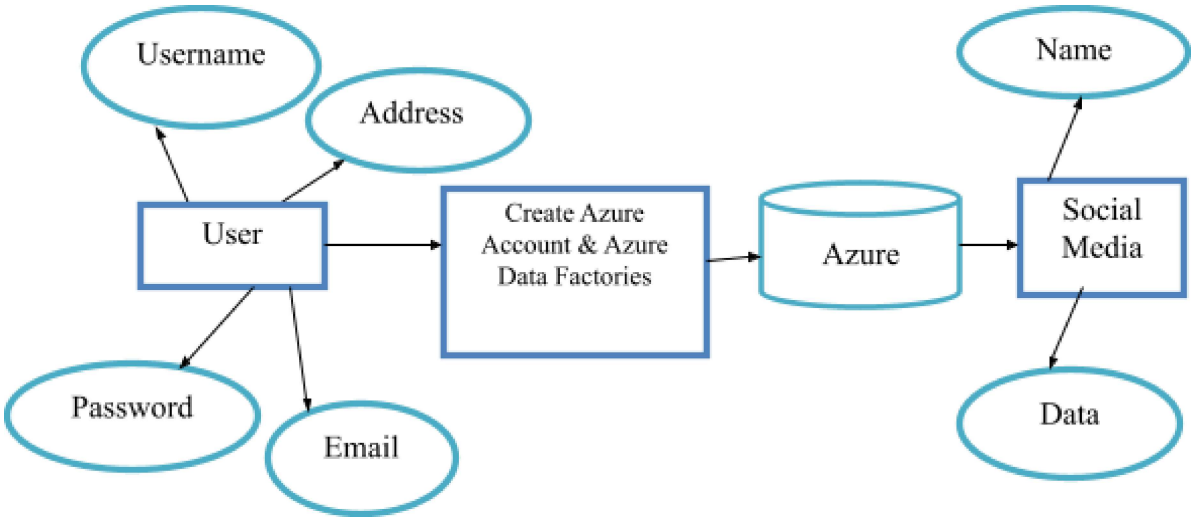


EXPLANATION:

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kinds of data will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel.

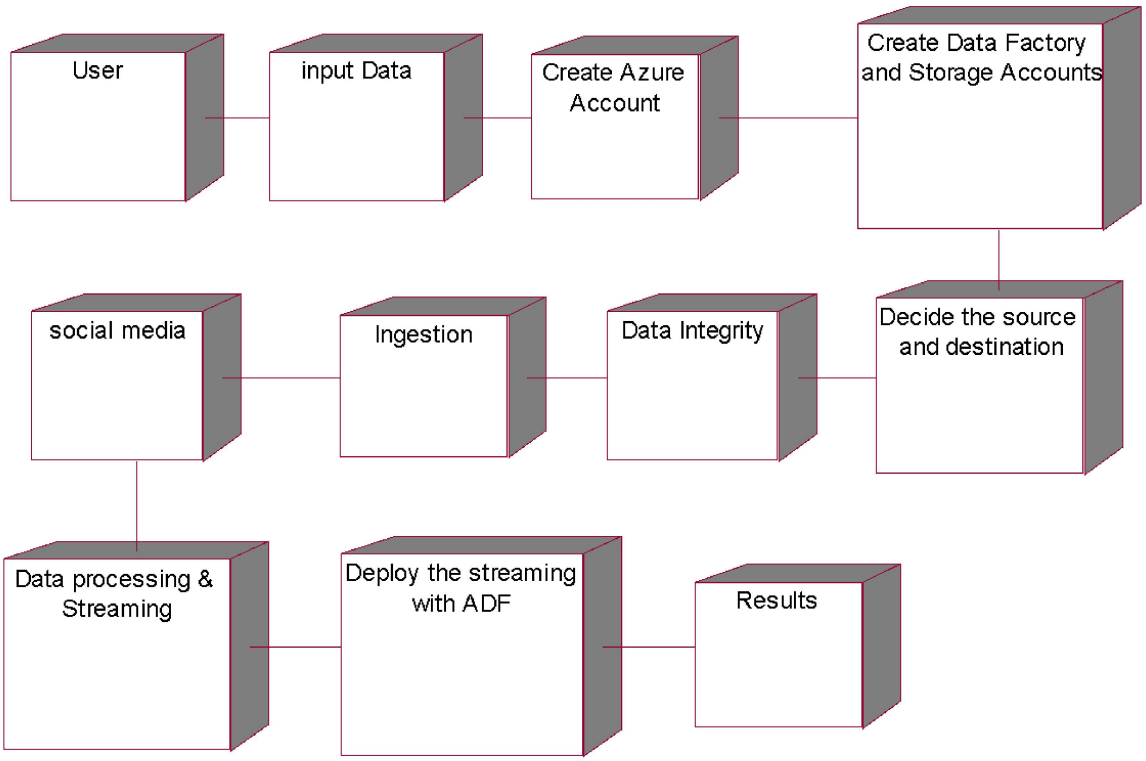
4.11 E-R DIAGRAM:



EXPLANATION:

Entity-Relationship Model (ERM) is an abstract and conceptual representation of data. Entity-relationship modeling is a database modeling method, used to produce a type of conceptual schema or semantic data model of a system, often a relational database.

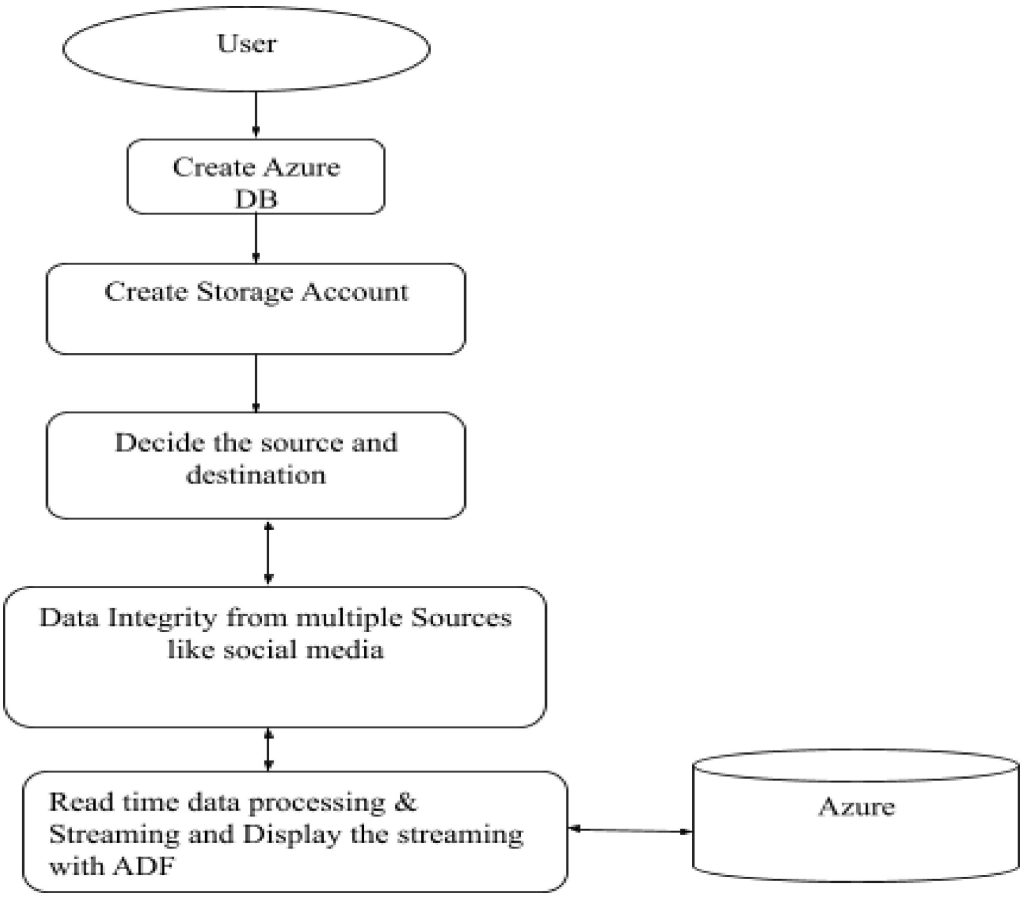
4.12 DEPLOYMENT DIAGRAM:



EXPLANATION:

In the UML, a component diagram depicts how components are wired together to form larger deployment and or software systems. They are used to illustrate the structure of arbitrarily complex systems. User gives main query and it converted into sub queries and sends through data dissemination. Results are to be showed to user by data aggregators. All boxes are arrow indicates dependencies.

4.13 SYSTEM ARCHITECTURE



System Architecture Model

To implement real-time data processing and streaming using Azure services, start by setting up an Azure Databricks workspace with optimized clusters for streaming workloads. Configure an Azure Storage account to store processed data. Define streaming data sources like social media APIs or IoT devices, and ingest this data into Databricks. Use Azure Data Factory (ADF) to orchestrate data movement and connect streaming data to Databricks for processing. Implement data integrity measures for reliable data quality. Utilize Azure Databricks' Structured Streaming for real-time processing, applying transformations and aggregations. Integrate with ADF to display processed streaming data on dashboards, providing actionable insights. This architecture offers scalability and flexibility for efficient real-time data processing and visualization.

CHAPTER 5
DEVELOPMENT TOOLS

5.1 GENERAL

Azure is Microsoft's cloud computing platform, providing a comprehensive suite of services to help organizations and developers build, deploy, and manage applications and services through Microsoft's global data centers. Azure offers various cloud services such as computing power, storage, databases, networking, artificial intelligence, machine learning, analytics, and Internet of Things. The platform supports different computing models, including Infrastructure as a Service (IaaS), where users can create and manage their own infrastructure in the cloud; Platform as a Service (PaaS), which offers a platform for developing, deploying, and managing applications without the need to manage underlying infrastructure; and Software as a Service (SaaS), allowing organizations to use third-party software as a service without worrying about maintenance.

5.2 HISTORY OF AZURE

Microsoft Azure, originally known as Windows Azure, was launched on February 1, 2010, as part of Microsoft's strategic move into the cloud computing market. Initially, Azure offered Platform as a Service (PaaS) options, allowing developers to build, deploy, and manage applications in the cloud without needing to handle the underlying infrastructure. As Azure evolved, it expanded to include Infrastructure as a Service (IaaS) and a diverse range of services such as data storage, databases, networking, artificial intelligence, machine learning, and Internet of Things (IoT).

Over the years, Microsoft continued to invest in Azure, expanding its global network of data centers to provide low-latency access and high availability for users worldwide. The platform's offerings have been continually updated to meet the changing needs of modern organizations, emphasizing security, compliance, and performance. Today, Azure is a major player in the cloud computing market, offering a comprehensive suite of services to support organizations of all sizes in various industries.

5.3 CLOUD SERVICES PROVIDED BY AZURE

Microsoft Azure offers a wide range of cloud services that cater to different needs and use cases across industries. These services can be grouped into various categories, such as computing, storage, databases, networking, artificial intelligence, and Internet of Things. Here's an overview of some of the key cloud services provided by Azure:

Compute: Azure provides virtual machines (VMs) and container services that allow you to run applications and services in the cloud. You can choose from a variety of VM configurations to match your performance and budget requirements. Azure also offers serverless computing through Azure Functions, which lets you run code on-demand without managing infrastructure.

Storage: Azure offers different types of storage services, including Blob Storage for storing unstructured data, File Storage for managed file shares, and Disk Storage for persistent data storage for virtual machines. There is also Azure Data Lake Storage, which is optimized for big data analytics.

Databases: Azure provides managed database services such as Azure SQL Database, Azure Cosmos DB (a NoSQL database service), and Azure Database for MySQL and PostgreSQL. These services offer scalable and reliable data management with built-in security and backup features.

Networking: Azure networking services include Virtual Networks for connecting resources within your Azure environment, Azure Load Balancer for distributing traffic across multiple resources, and Azure Application Gateway for managing web traffic. Azure also offers VPN and ExpressRoute services for secure connectivity between your on-premises network and Azure.

Artificial Intelligence (AI) and Machine Learning (ML): Azure offers AI and ML services such as Azure Machine Learning, which allows you to build, train, and deploy machine learning models. Azure Cognitive Services provides APIs for natural language processing, computer vision, and other AI tasks.

Analytics: Azure provides a suite of analytics services such as Azure Synapse Analytics (formerly known as Azure SQL Data Warehouse), which allows you to perform large-scale data analytics. Azure Data Lake Analytics offers on-demand data processing and analysis, while Azure Stream Analytics enables real-time data processing.

5.4 AZURE ROLE IN STREAMING

In the context of real-time stream processing, Microsoft Azure plays a significant role by providing a range of cloud services and tools that facilitate the ingestion, processing, and analysis of streaming data. Azure's ecosystem offers a scalable, flexible, and reliable platform for handling streaming data in real-time.

Azure Databricks is a collaborative platform built on Apache Spark that integrates seamlessly with Azure services. It provides an environment for developing, running, and managing data engineering, data science, and machine learning workloads. When used for stream processing, Databricks allows you to create streaming jobs using Apache Spark Streaming, enabling real-time data transformation, aggregation, and analysis.

Azure also offers Azure Stream Analytics, a fully managed, real-time event processing service that allows you to analyze and gain insights from streaming data as it arrives. It supports complex event processing and can output results to various destinations such as Azure Blob Storage, SQL Database, or Power BI for visualization.

5.5 APACHE SPARK

Apache Spark is a powerful, open-source data processing framework designed for speed, scalability, and versatility. It supports a wide range of data analytics workloads, including batch processing, stream processing, machine learning, and graph processing. Spark's in-memory computing capabilities enable fast data processing by caching data in memory, rather than on disk, allowing for efficient execution of tasks. The framework provides high-level APIs like Data Frame and Dataset for structured and semi-structured data manipulation, as well as SQL-like queries through its Spark SQL module.

Spark Streaming allows for real-time data processing, while it offers a comprehensive library for building and evaluating machine learning models. Spark's compatibility with multiple programming languages, including Scala, Java, Python, and R, makes it accessible to a diverse range of developers. The framework's resilience and fault tolerance are supported by Resilient Distributed Datasets (RDDs), and its integration with other big data technologies, such as Hadoop and NoSQL databases, enhances its capabilities. Spark's large, active community and rich ecosystem contribute to its popularity in data engineering, data science, and analytics projects across various industries.

5.6 ARCHITECTURE OF APACHE SPARK

1. Driver Program

In Apache Spark, the driver program is a central component that runs the main () function of the application and creates a Spark Context object. The Spark Context is responsible for coordinating Spark applications, which operate as independent sets of processes on a cluster. When running an application on a cluster, the Spark Context connects to different types of cluster managers and performs several critical tasks. First, it acquires executors on nodes within the cluster. It then sends the application code, defined by JAR or Python files, to the executors for execution. Finally, the Spark Context dispatches tasks to the executors for processing.

2. Cluster Manager

The cluster manager plays a crucial role in resource allocation across applications, as it ensures that Spark can run efficiently on various clusters. Spark supports multiple cluster managers, including Hadoop YARN, Apache Mesos, and its Standalone Scheduler. The Standalone Scheduler is a standalone Spark cluster manager that allows users to install Spark on an empty set of machines.

3. Worker Node

Worker nodes are slave nodes in the cluster responsible for running the application code.

4. Executor

Executors are processes launched for each application on worker nodes. They manage tasks, execute the application code, and handle data in memory or disk storage across tasks. Executors also read and write data to and from external sources as needed.

5. Task

A task is a unit of work sent to an executor to be executed. Tasks are the smallest units of work in Spark and are responsible for performing specific operations on data.

Spark's architecture is designed to be fault-tolerant, with built-in resilience mechanisms such as lineage information and task retries that enable the system to handle failures gracefully and maintain data integrity. These features make Spark a powerful tool for big data analytics, offering fast, reliable data processing and enhanced scalability.

5.7 STREAMING AND ANALYZING DATA IN SPARK

1. Streaming Data in Spark

Apache Spark offers a powerful framework for real-time data processing with Spark Streaming. This module enables the handling of continuous data streams in a fault-tolerant and efficient manner. Spark Streaming ingests real-time data from various sources such as Apache Kafka, Amazon Kinesis, Azure Event Hubs, and other streaming data platforms. The data is divided into small batches, also known as micro-batches, and is then processed using Spark's transformations. As the data flows in, Spark Streaming applies transformations such as clean, filter, and manipulate the data according to the application's needs.

2. Analyzing Data in Spark

Spark's robust ecosystem allows for comprehensive data analysis, whether on streaming data or data at rest. Spark Streaming integrates with other Spark modules such as Spark SQL for structured querying and machine learning analysis. By combining these capabilities, Spark can process and analyze real-time data streams for immediate insights.

Structured Streaming, a higher-level API for stream processing, enables developers to define data flows as structured tables, simplifying the creation of data pipelines. This API supports event-time processing, windowed operations, and watermarks to handle out-of-order data. Additionally, stateful stream processing allows the application to track and update state information across streaming data batches, enabling advanced analytics like aggregations, counting, and windowed operations over time.

5.8 INTRODUCTION TO AZURE DATABRICKS

Databricks is a unified analytics platform designed to simplify big data and AI workloads. It is built on top of Apache Spark and provides a comprehensive environment for data engineering, data science, machine learning, and analytics. The platform offers a collaborative workspace where teams can work together seamlessly, sharing code, insights, and visualizations in notebooks. Databricks is known for its high-performance distributed computing capabilities, allowing users to scale workloads efficiently as data volumes increase. The platform integrates with major cloud services such as Microsoft Azure, Amazon Web Services (AWS), and Google Cloud Platform (GCP), making it easy to work with other cloud-based services. Security and compliance are prioritized with features like encryption, identity management, and network

security. Databricks supports use cases such as data engineering, machine learning, and analytics, providing tools for building data pipelines, transforming and cleaning data, and handling large datasets while enabling the construction, training, and deployment of machine learning models at scale.

5.9 DATA PROCESSING USING DATABRICKS

Data processing in modern data platforms like Azure Databricks follows a structured workflow that begins with data ingestion from various sources. These sources include databases, file systems, cloud storage, and streaming platforms such as Apache Kafka, Azure Event Hubs, or Amazon Kinesis. The data may come in structured, semi-structured, or unstructured formats, providing a diverse range of information to work with.

Once data is ingested, it can be processed and transformed within Databricks notebooks using languages such as Python, Scala, R, and SQL. Spark's in-memory processing capabilities allow for efficient data transformations and aggregations, including filtering, mapping, and reducing data, as well as more complex operations such as joins and window functions. This flexibility enables users to manipulate and prepare data for analysis.

Moreover, the processed data can be leveraged for advanced analytics, including the training and deployment of machine learning models. This advanced use of data allows organizations to derive actionable insights that drive decision-making and facilitate predictive analytics. By utilizing Azure Databricks, users can gain a comprehensive understanding of their data and harness it to innovate and stay ahead in their respective industries.

Top of Form

CHAPTER 6
IMPLEMENTATION

6.1 GENERAL

1. Connection Establishment: The initial step involved establishing a seamless connection between the Apache Spark cluster and the storage system. This ensured efficient data exchange and accessibility of storage resources for analysis.
2. Data Generation: Following the successful connection setup, synthetic data was generated to simulate real-world streaming scenarios. This step allowed for the creation of diverse datasets representative of the expected input data streams.
3. Data Analysis: Leveraging the connected Spark cluster, comprehensive data analysis was performed on the generated datasets. This involved applying various analytical techniques, algorithms, and machine learning models to derive actionable insights and valuable information from the streaming data.

6.2 IMPLEMENTATION

Connecting spark to storage:

Step 1

```
dbutils.fs.mount(
  source = "wasbs://streaminput@cgitadlsb17.blob.core.windows.net/",
  mount_point = "/mnt/streaminput",
                                                    extra_configs                                                    =
{"fs.azure.account.key.cgitadlsb17.blob.core.windows.net":"1eOM0zhnPAa3LuGRVzoGDETVcbNJzP6LlL+k0zPZ
uBU2ZnR7cgI2duAZ1LN6zpGvSc2AcGNE8hdE+AStemKK/w=="}
)
```

Output:

True

Step 2

dbutils.fs.mounts()

```
dbutils.fs.mounts()
```

```
Out[8]: [MountInfo(mountPoint='/databricks-datasets', source='databricks-datasets', encryptionType=''),
 MountInfo(mountPoint='/Volumes', source='UnityCatalogVolumes', encryptionType=''),
 MountInfo(mountPoint='/databricks/mlflow-tracking', source='databricks/mlflow-tracking', encryptionType=''),
 MountInfo(mountPoint='/databricks-results', source='databricks-results', encryptionType=''),
 MountInfo(mountPoint='/databricks/mlflow-registry', source='databricks/mlflow-registry', encryptionType=''),
 MountInfo(mountPoint='/mnt/streaminput', source='wasbs://streaminput@cgitadlsb17.blob.core.windows.net/', encryptionType=''),
 MountInfo(mountPoint='/Volume', source='DbfsReserved', encryptionType=''),
 MountInfo(mountPoint='/volumes', source='DbfsReserved', encryptionType=''),
 MountInfo(mountPoint='/', source='DatabricksRoot', encryptionType=''),
 MountInfo(mountPoint='/volume', source='DbfsReserved', encryptionType='')]
```

Step 3

```
dbutils.fs.ls('/mnt/streaminput/inputdata/')
```

```
Out[10]: []
```

Generating Data

```
import time
import random
from datetime import datetime, timedelta

# Function to generate simulated streaming data
def generate_data(duration_seconds):
    start_time = datetime.now()
    end_time = start_time + timedelta(seconds=duration_seconds)
    while datetime.now() < end_time:
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        print(timestamp)
        temperature = random.uniform(0, 100)
        humidity = random.uniform(0, 100)
        yield (timestamp, temperature, humidity)
```

```
    time.sleep(1) # Simulate streaming data every second
# Create a Spark DataFrame with the generated data
schema = "timestamp STRING, temperature FLOAT, humidity FLOAT"
streaming_df = spark.createDataFrame(generate_data(1000), schema=schema)
```

Output:

```
2024-04-13 07:33:44
2024-04-13 07:33:45
2024-04-13 07:33:46
2024-04-13 07:33:47
2024-04-13 07:33:48
2024-04-13 07:33:49
2024-04-13 07:33:50
2024-04-13 07:33:51
2024-04-13 07:33:52
2024-04-13 07:33:53
2024-04-13 07:33:54
2024-04-13 07:33:55
2024-04-13 07:33:56
2024-04-13 07:33:57
2024-04-13 07:33:58
2024-04-13 07:33:59
2024-04-13 07:34:00
2024-04-13 07:34:01
2024-04-13 07:34:02
2024-04-13 07:34:03
2024-04-13 07:34:04
```

Displaying the Data

```
streaming_df.display()
```

Table

	timestamp	temperature	humidity
1	2024-04-13 07:17:25	71.1304	25.649565
2	2024-04-13 07:17:26	94.59711	55.473854
3	2024-04-13 07:17:27	40.390434	90.91488
4	2024-04-13 07:17:28	54.44489	55.95769
5	2024-04-13 07:17:29	50.6958	34.584946
6	2024-04-13 07:17:30	44.771812	4.469972
7	2024-04-13 07:17:31	5.517575	92.37506

999 rows

Transferring the generated data to the storage:

```
streaming_df.write.mode('overwrite').parquet('/mnt/streaminput/inputdata')
```

6.3 ANALYZING

connecting to a storage (a Parquet file) and performing windowed aggregation on streaming data

```
from pyspark.sql.functions import window
# Define the window duration and slide duration
windowDuration = "20 seconds"
slideDuration = "5 seconds"
streaming_df=spark.read.parquet('/mnt/streaminput/inputdata')
# Apply windowing to the streaming DataFrame
windowed_df = streaming_df \
    .withWatermark("timestamp", "1 minute") \
    .groupBy(window("timestamp", windowDuration, slideDuration)) \
    .agg({"temperature": "avg", "humidity": "avg"}) \
    .orderBy("window")
```

Displaying

windowed_df.display()

```
windowed_df.display()
```

Table

	window	avg(temperature)	avg(humidity)
1	{"start": "2024-04-13T07:17:10.000+0000", "end": "2024-04-13T07:17:30.000+0000"}	62.25172653198242	52.51618690490723
2	{"start": "2024-04-13T07:17:15.000+0000", "end": "2024-04-13T07:17:35.000+0000"}	45.7144540309906	45.350739812850954
3	{"start": "2024-04-13T07:17:20.000+0000", "end": "2024-04-13T07:17:40.000+0000"}	40.25641965866089	45.53593962987264
4	{"start": "2024-04-13T07:17:25.000+0000", "end": "2024-04-13T07:17:45.000+0000"}	42.13581826686859	43.82636210918427
5	{"start": "2024-04-13T07:17:30.000+0000", "end": "2024-04-13T07:17:50.000+0000"}	38.26128146648407	39.9446950674057
6	{"start": "2024-04-13T07:17:35.000+0000", "end": "2024-04-13T07:17:55.000+0000"}	39.2442773103714	48.45156364440918

203 rows

Creating a temporary view of original table:

```
streaming_df.createOrReplaceTempView('streamtable')
```

SQL

Query 1

A new table was created from the original table with selected parameters, and the resulting table was stored in the designated output folder within the storage system.

```
%sql

-- Execute SQL query and save result to a table with the name 'abc'
CREATE TABLE avg AS
SELECT AVG(temperature) AS avg_temperature, AVG(humidity) AS avg_humidity
FROM streamtable
```

Query returned no results

```
%sql
-- displaying the avg table we created above
select * from avg
```

Table

	avg_temperature	avg_humidity
1	51.09519531269436	50.6519705225845

1 row

```
# Saving the contents of the table named 'avg' to the output location
spark.table("avg").write.mode('overwrite').parquet('/mnt/streaminput/outputdata')
```

Query 2

Finding Average

```
%sql
-- finding out averages
select avg(temperature),avg(humidity) from streamtable
```

Table

	avg(temperature)	avg(humidity)
1	51.09519531269436	50.6519705225845

1 row

Query 3

Filtering Data

```
%sql
--filtering data
SELECT * FROM streamtable WHERE temperature > 30
```

Table

	timestamp	temperature	humidity
1	2024-04-13 07:29:53	62.954273	80.48248
2	2024-04-13 07:29:54	89.18685	68.93214
3	2024-04-13 07:29:55	90.516846	25.1791
4	2024-04-13 07:29:56	99.03825	97.23823
5	2024-04-13 07:29:57	52.91935	39.987476
6	2024-04-13 07:29:59	82.3789	45.9253
7	2024-04-13 07:30:00	98.871826	98.75583

717 rows

Query 4

Calculate Rolling Average Temperature and Humidity

```
%sql
--Calculate Rolling Average Temperature and Humidity
SELECT timestamp, temperature,
       AVG(temperature) OVER (ORDER BY timestamp ROWS BETWEEN 10 PRECEDING AND CURRENT ROW) AS rolling_avg_temperature,
       AVG(humidity) OVER (ORDER BY timestamp ROWS BETWEEN 10 PRECEDING AND CURRENT ROW) AS rolling_avg_humidity
FROM streamtable
```

Table

	timestamp	temperature	rolling_avg_temperature	rolling_avg_humidity
1	2024-04-13 07:17:25	71.1304	71.13040161132812	25.649564743041992
2	2024-04-13 07:17:26	94.59711	82.86375427246094	40.5617094039917
3	2024-04-13 07:17:27	40.390434	68.7059809366862	57.34609921773275
4	2024-04-13 07:17:28	54.44489	65.14070796966553	56.9989972114563
5	2024-04-13 07:17:29	50.6958	62.25172653198242	52.51618690490723
6	2024-04-13 07:17:30	44.771812	59.33840751647949	44.50848444302877
7	2024-04-13 07:17:31	5.517575	51.64971712657383	51.34656681333269

999 rows

Query 5

Identify Peaks in Temperature

```
%sql
--Identify Peaks in Temperature
SELECT timestamp, temperature
FROM streamtable
WHERE temperature > (SELECT AVG(temperature) FROM streamtable) +  (SELECT STDDEV(temperature) FROM streamtable)
```

Table

	timestamp	temperature
1	2024-04-13 07:29:54	89.18685
2	2024-04-13 07:29:55	90.516846
3	2024-04-13 07:29:56	99.03825
4	2024-04-13 07:29:59	82.3789
5	2024-04-13 07:30:00	98.871826
6	2024-04-13 07:30:06	81.285995
7	2024-04-13 07:30:08	83.92291

216 rows

Query 6

Calculating the average temperatures based on hourly intervals.

```
%sql
SELECT DATE_FORMAT(timestamp, 'yyyy-MM-dd HH') AS hour, AVG(temperature) AS avg_temperature
FROM streamtable
GROUP BY DATE_FORMAT(timestamp, 'yyyy-MM-dd HH')
```

Table

	hour	avg_temperature
1	2024-04-13 07	51.09519531269436

1 row

CHAPTER 7

SNAPSHOTS

7.1 GENERAL:

In this project, Azure Databricks, Azure Storage containers, and Azure Compute resources were utilized for real-time data processing. The Azure Databricks compute environment was leveraged to execute Spark notebooks, while data storage and management were handled using Azure Storage containers.

SNAPSHOTS

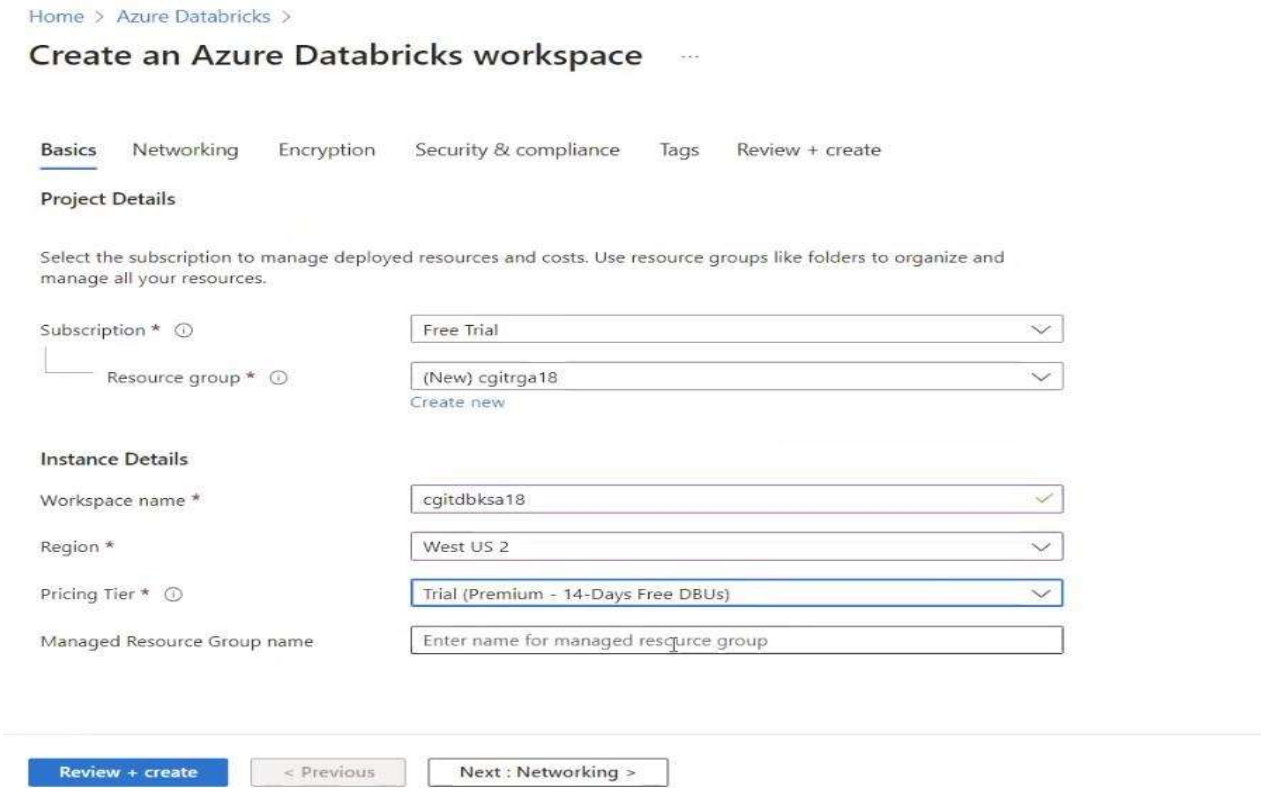


Fig 7.1 Azure Databricks Creation

Initial setup of Azure Databricks environment for data processing and collaboration.

AZURE DATABRICKS

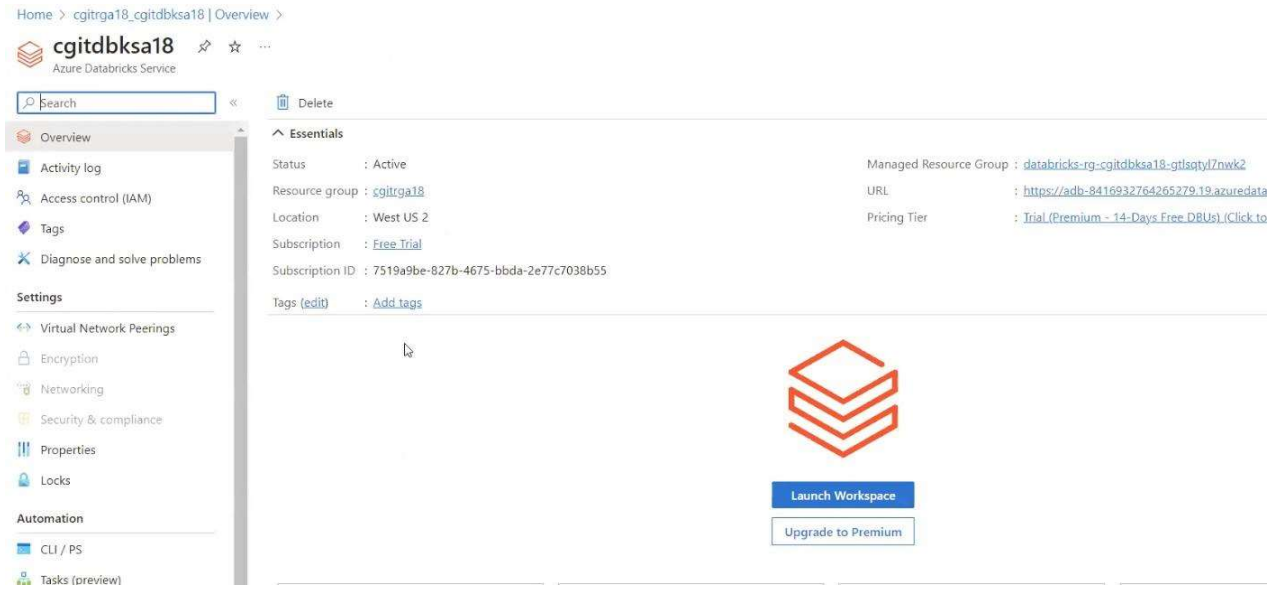


Fig 7.2 Azure Databricks launching

Accessing the Azure Databricks service after creation to begin work on data projects.

DATABRICKS WORKSPACE

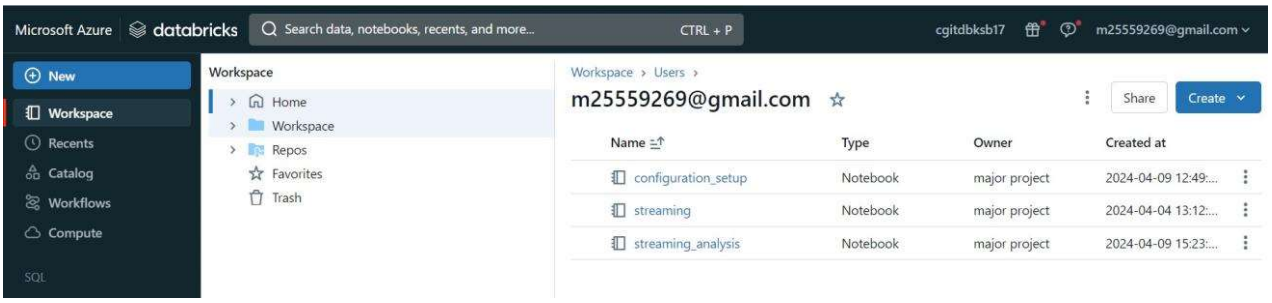


Fig 7.3 Azure Databricks Workspace

The interface where users can create, manage, and run code in Azure Databricks.

SPARK CLUSTER CREATION

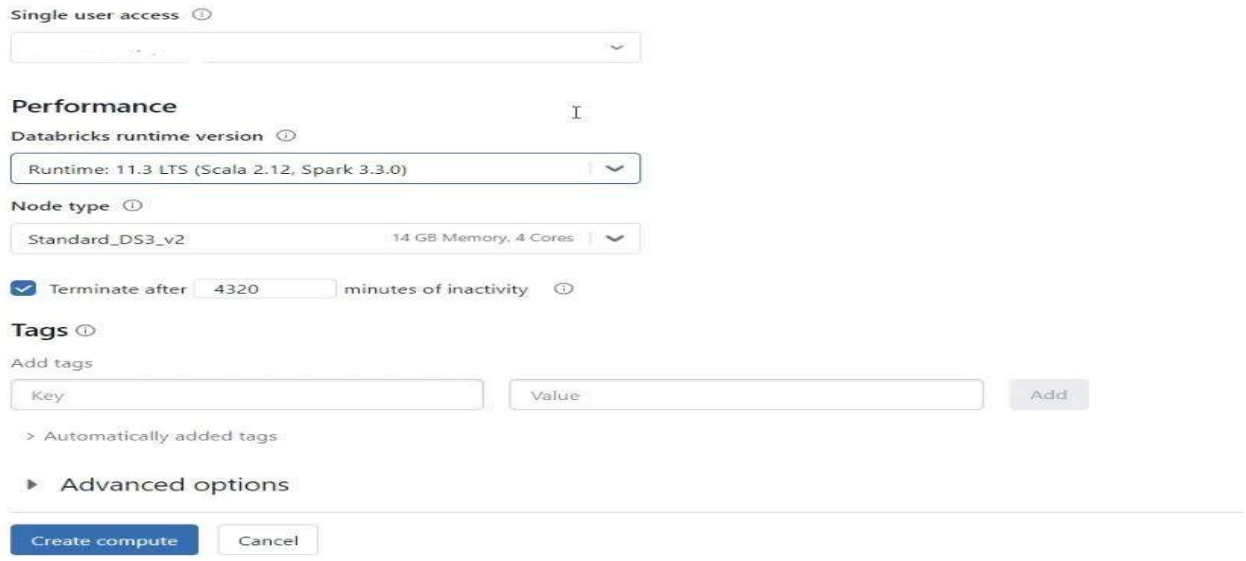


Fig 7.4 Computation(Spark cluster) creation

Creating a Spark cluster within Azure Databricks for distributed data processing.

AZURE STORAGE

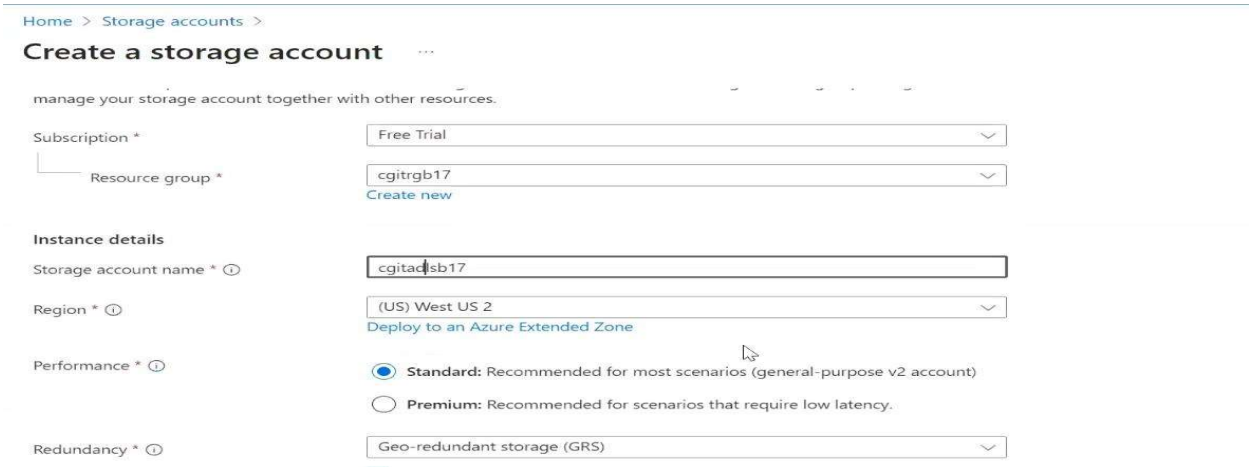


Fig 7.5Azure Storage creation

Setting up Azure Storage to store data used in Azure Databricks projects.

STORAGE CONTAINER

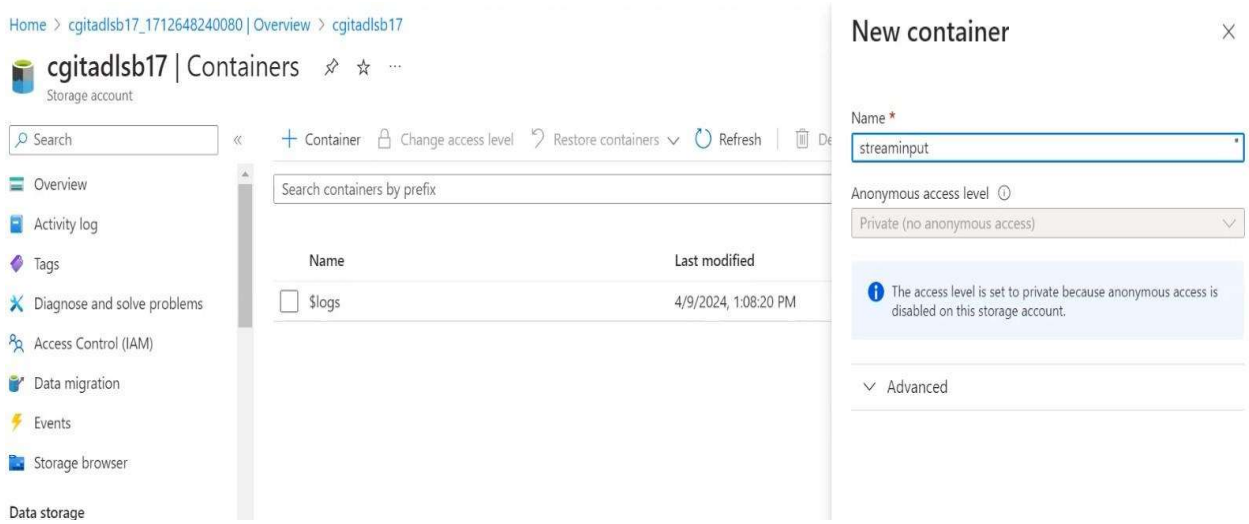


Fig 7.6 Storage container creation

Creating a container within Azure Storage to organize and manage data files.

DIRECTORIES CREATION

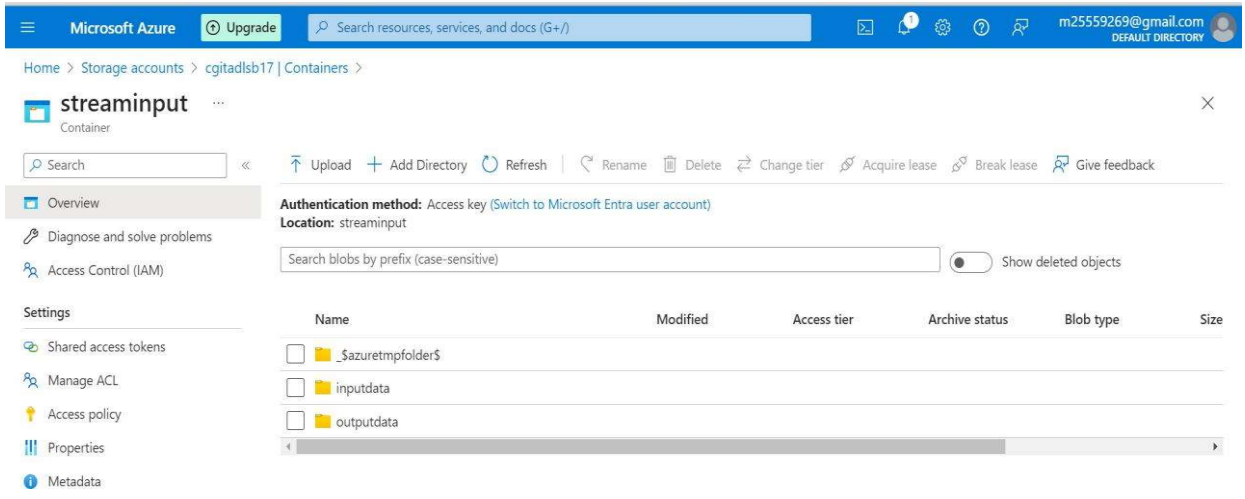


Fig 7.7 Directories creation under container

Establishing directories within the storage container to further organize data.

INPUT FILE

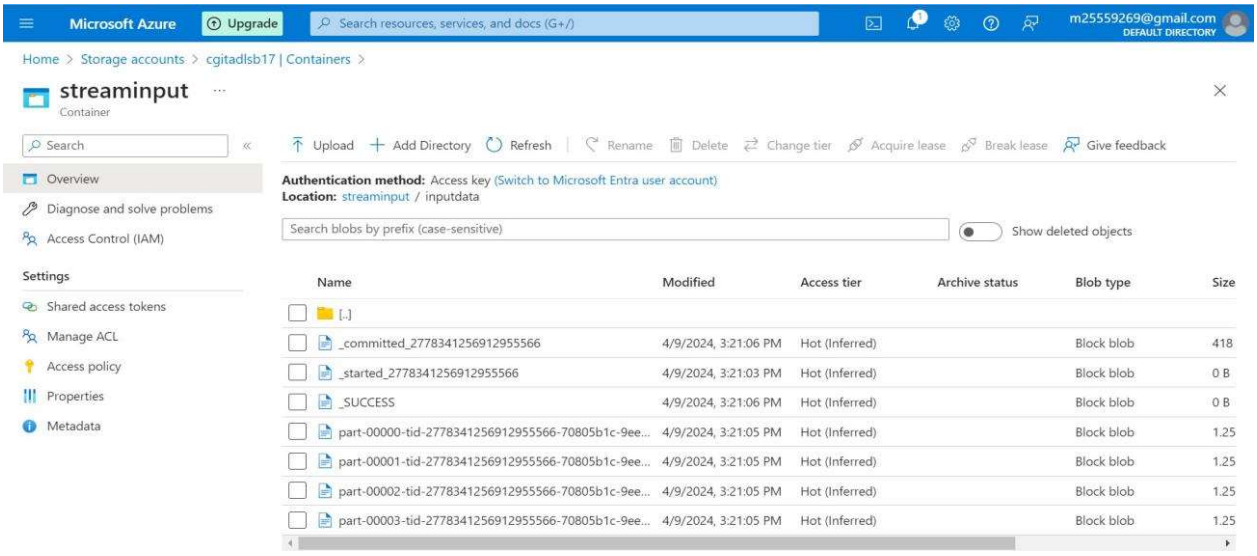


Fig 7.8 Input file inside container

Placing data files into the storage container to be used as input for Azure Databricks processes.

OUTPUT FILE

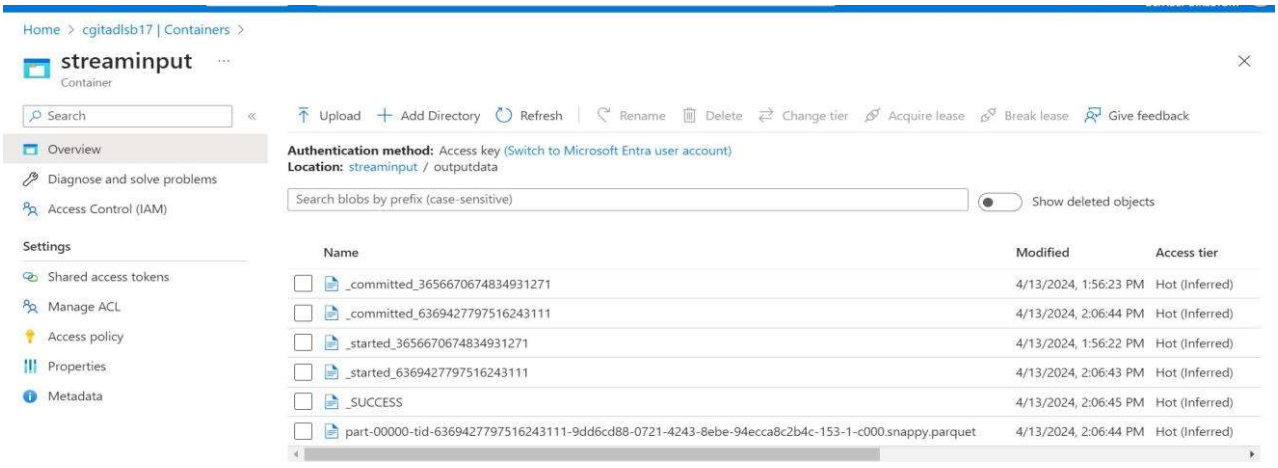


Fig 7.9 Output file inside container

Storing processed data or results within the storage container for further analysis or sharing.

CHAPTER 8
SOFTWARE TESTING

8.1 GENERAL

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

8.2 DEVELOPING METHODOLOGIES

The test process is initiated by developing a comprehensive plan to test the general functionality and special features on a variety of platform combinations. Strict quality control procedures are used. The process verifies that the application meets the requirements specified in the system requirements document and is bug free. The following are the considerations used to develop the framework from developing the testing methodologies.

8.3 TYPES OF TESTS

8.3.1 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

8.3.2 FUNCTIONAL TEST

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

8.3.3 SYSTEM TEST

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

8.3.4 PERFORMANCE TEST

The Performance test ensures that the output be produced within the time limits,and the time taken by the system for compiling, giving response to the users and request being send to the system for to retrieve the results.

8.3.5 INTEGRATION TESTING

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

8.3.6 ACCEPTANCE TESTING

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

ACCEPTANCE TESTING FOR DATA SYNCHRONIZATION:

- ➢ The Acknowledgements will be received by the Sender Node after the Packets are received by the Destination Node.
- ➢ The Route add operation is done only when there is a Route request in need
- ➢ The Status of Nodes information is done automatically in the Cache Updation process

8.3.7 BUILD THE TEST PLAN

Any project can be divided into units that can be further performed for detailed processing. Then a testing strategy for each of this unit is carried out. Unit testing helps to identity the possible bugs in the individual component, so the component that has bugs can be identified and can be rectified from errors.

CHAPTER 9

APPLICATIONS AND FUTURE ENHANCEMENT

9.1 FUTURE ENHANCEMENTS

Future enhancements in real-time stream processing with Azure Databricks and Apache Spark could involve several key areas, including enhanced integration with Azure services like Synapse Analytics and Data Lake Storage, optimized resource management for dynamic allocation, advanced stream processing algorithms for complex tasks, improved fault tolerance mechanisms, deeper integration with machine learning and AI frameworks, support for complex event processing patterns, native graph processing capabilities, comprehensive monitoring and management features, simplified development experiences, multi-language support, security enhancements, serverless deployment options, ongoing performance optimization efforts, and fostering community contributions for extending functionality. These enhancements aim to bolster performance, scalability, reliability, and ease of use, empowering organizations to derive actionable insights from streaming data with greater efficiency and agility.

Future enhancements in real-time stream processing with Azure Databricks and Apache Spark could focus on several key areas to improve performance, scalability, reliability, and ease of use. Here are some potential enhancements:

Enhanced Integration with Azure Services: Strengthen integration with other Azure services such as Azure Synapse Analytics, Azure Data Lake Storage, and Azure Event Hubs for seamless data ingestion, storage, and analytics.

Optimized Resource Management: Develop better resource management capabilities to dynamically allocate and deallocate resources based on workload demands, thereby optimizing cost and performance.

Advanced Stream Processing Algorithms: Research and implement advanced stream processing algorithms to handle complex event processing, anomaly detection, and pattern recognition tasks more efficiently.

Improved Fault Tolerance: Enhance fault tolerance mechanisms to ensure minimal data loss and downtime, even in the face of node failures or network issues, by leveraging features like checkpointing and data replication.

Integration with ML and AI: Deepen integration with machine learning and artificial intelligence frameworks to enable real-time model training, inference, and deployment directly within the stream processing pipeline.

Support for Complex Event Processing: Extend support for complex event processing (CEP) patterns such as sliding windows, session windows, and pattern matching to enable more sophisticated event-driven applications.

Native Support for Graph Processing: Introduce native support for graph processing algorithms to analyze interconnected data in real-time, enabling use cases like social network analysis, fraud detection, and recommendation systems.

Enhanced Monitoring and Management: Provide comprehensive monitoring and management capabilities, including real-time metrics, logging, and alerting, to ensure the health and performance of the stream processing pipeline.

Simplified Development Experience: Streamline the development experience by offering higher-level abstractions, built-in connectors, and intuitive APIs for common stream processing tasks, reducing the learning curve and development time.

Multi-language Support: Expand support for multiple programming languages beyond Scala, Python, and SQL to cater to a broader audience of developers and data engineers.

Security Enhancements: Strengthen security features such as data encryption, role-based access control (RBAC), and audit logging to meet compliance requirements and protect sensitive data in transit and at rest.

Serverless Deployment: Introduce serverless deployment options for Azure Databricks stream processing workloads, allowing users to focus on application logic without worrying about infrastructure management or provisioning.

Performance Optimization: Continuously optimize performance by leveraging advancements in distributed computing techniques, parallel processing, and hardware acceleration (e.g., GPU support) to achieve lower latency and higher throughput.

Community Contributions and Extensions: Encourage community contributions and provide support for third-party extensions and plugins to extend the functionality of Azure Databricks and Apache Spark for specific use cases and industries.

CHAPTER 10
CONCLUSION

CONCLUSION

In conclusion, real-time stream processing with Azure Databricks and Apache Spark presents a powerful solution for organizations seeking to harness the full potential of streaming data. By continually enhancing integration with Azure services, optimizing resource management, advancing stream processing algorithms, and bolstering fault tolerance mechanisms, these platforms offer a robust foundation for building scalable, reliable, and high-performance stream processing pipelines. Furthermore, deeper integration with machine learning and AI frameworks, support for complex event processing patterns, and native graph processing capabilities enrich the analytics capabilities, enabling organizations to derive valuable insights in real-time. With ongoing efforts to simplify development experiences, strengthen security, and foster community

In addition to the technical advancements, the future of real-time stream processing with Azure Databricks and Apache Spark also hinges on their ability to address evolving business needs. As organizations increasingly rely on streaming data to make critical decisions in real-time, there's a growing demand for features that enhance data governance, lineage tracking, and regulatory compliance. Furthermore, support for hybrid and multi-cloud deployments ensures flexibility and scalability, enabling seamless integration with existing infrastructure and future expansion. By continuously innovating and adapting to emerging trends, Azure Databricks and Apache Spark are poised to play a central role in shaping the future of real-time analytics, driving value and competitiveness for organizations across industries.

In conclusion, the future of real-time stream processing with Azure Databricks and Apache Spark holds tremendous promise for organizations seeking to derive actionable insights from streaming data. Through continuous innovation in areas such as integration, performance optimization, collaboration, and adaptation to emerging technologies, these platforms are poised to play a pivotal role in driving digital transformation and unlocking new opportunities across industries. By embracing these advancements and harnessing the power of real-time analytics, organizations can gain a competitive edge, drive innovation, and achieve their strategic objectives in the dynamic landscape of the data-driven world.

REFERENCES

[1] C. L. Philip Chen and C.-Y. Zhang, “Data-intensive applications, challenges, techniques and technologies: A survey on Big Data,” Inf. Sci. (Ny)., vol. 275, pp. 314–347, 2014.

[2] D. Laney, “3D data management: Controlling data volume, velocity and variety.,” META Gr. Res. Note, vol. 6, no. February 2001, p. 70, 2001.

[3] Z. Zheng, P. Wang, J. Liu, and S. Sun, “Real-Time Big Data Processing Framework: Challenges and Solutions,” Appl. Math. Inf. Sci, vol. 9, no. 6, pp. 3169–3190, 2015.

[4] S. Shahrivari, “Beyond batch processing: towards real-time and streaming big data,” Computers, vol. 3, no. 4, pp. 117–129, 2014.

[5] R. Casado and M. Younas, “Emerging trends and technologies in big data processing,” Concurr. Comput. Pract. Exp., vol. 27, no. 8, pp. 2078–2091, 2015.

[6] A. A. Safaei, “Real-time processing of streaming big data,” RealTime Syst., vol. 53, no. 1, 2017.

[7] “Real time processing|Microsoft Docs.” [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/data-guide/bigdata/real-time-processing#technology-choices. [Accessed: 08-Jun2018].

[8] B. Samal and M. Panda, “Real Time Product Feedback Review and Analysis Using Apache Technologies and NOSQL Database,” Int. J. Eng. Comput. Sci., vol. 6, no. 10, pp. 22551–22558, 2017.

[9] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, “Apache flink: Stream and batch processing in a single engine,” Bull. IEEE Comput. Soc. Tech. Comm. Data Eng., vol. 36, no. 4, 2015.

[10] G. Hesse and M. Lorenz, “Conceptual survey on data stream processing systems,” in Parallel and Distributed Systems (ICPADS), 2015 IEEE 21st International Conference on, 2015, pp. 797–802.

[11] Y. Shi, X. Meng, J. Zhao, X. Hu, B. Liu, and H. Wang, “Benchmarking cloud-based data management systems,” in Proceedings of the second international workshop on Cloud data management, 2010, pp. 47–54.

[12] A. Katal, M. Wazid, and R. H. Goudar, “Big data: Issues, challenges, tools and Good practices,” in 2013 6th International Conference on Contemporary Computing, IC3 2013, 2013, pp. 404–409.

[13] N. Khan, I. Yaqoob, I. A. T. Hashem, Z. Inayat, W. K. Mahmoud Ali, M. Alam, M. Shiraz, and A. Gani, “Big Data: Survey, Technologies, Opportunities, and Challenges,” Sci. World J., vol. 2014, pp. 1–18, 2014.

[14] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, “Big data: The next frontier for innovation, competition, and productivity.” May-2011.