ISSN: 2754-6659

Journal of Artificial Intelligence & Cloud Computing

Review Article

Open Access

Designing Distributed Artifact Ingestion Platform for Analytics & Machine Learning

Mahidhar Mullapudi[1]*, Narayana Challa[2] and Abhishek Shende[3]

[1]Senior Software Engineer, Microsoft, WA, USA
[2]Lead Integrations Developer , TX, USA
[3]Principal Software Engineer, CA, USA

ABSTRACT
Demand for efficient large-scale heterogeneous distributed data ingestion pipelines for transforming and publishing data that is essential for advanced analytics and machine learning models, have gained substantial importance. Services increasingly rely on near-real-time signals to accurately identify or predict customer behavior, sentiments, anomalies facilitating data-driven decision-making. This paper delves into the forefront of distributed and parallel computing, examining the latest advancements in storage, query, and ingestion methodologies. Furthermore, it systematically assesses cutting-edge tools designed for both periodic and real-time analysis of heterogeneous data.

"The data quality is more important than the Machine Learning model itself". Achieving precision in decision-making or generating precise output from the Machine Learning models necessitates a keen focus on input data quality and consistency.

Building a robust ingestion platform for handling hundreds of Gigabytes/Petabytes per day involves a comprehensive understanding of the overarching architecture, the intricacies of involved components, and the unique challenges within these ecosystems. Building a service platform demands thoughtful consideration and resolution of key aspects, including a scalable ingestion handler, a flexible and fault-tolerant data processing library, a highly scalable and resilient event system, an analytics/reporting platform, machine learning platform, and robust application health and security measures.

This paper delves into the overall architecture, explicates design choices, and imparts insights into best practices for implementing and maintaining such a platform, leveraging contemporary tools. The discussion encompasses critical aspects of the platform's functionality, emphasizing the need for scalability, fbility, resilience, and security to meet the demands of modern data-driven decision-making scenarios.

***Corresponding author**
Mahidhar Mullapudi, Senior Software Engineer, Microsoft, WA, USA.

Received: March 07, 2022; **Accepted:** March 15, 2022; **Published:** March 25, 2022

Keywords: Modern Ingestion Platform, Artifact Analytics, Azure Event Hub, Azure Synapse, Azure Streaming Service, Azure Machine Learning

Introduction
Real-time data ingestion and processing systems play a pivotal role in providing data to analytics and machine learning platforms, enabling the extraction of invaluable insights. Numerous companies have developed bespoke systems to address the unique challenges posed by this dynamic data ingestion ecosystem. In this paper, we delve into the essential attributes pivotal in the conception and construction of a near real-time data ingestion platform [1-3].

Ease-of-Use
This criterion assesses the complexity of data requirements, the need for maintaining data versions for consistency and testing, and the integration with other services.

Flexibility
Examining the frequency of changing requirements, the existence of diverse data schemas within the system, dependencies of other services on the data generated by this platform, and the language-agnostic nature of the design.

Performance
Evaluating acceptable latency levels, whether in seconds or minutes and determining the required throughput both per machine and in aggregate for each service [3].

Scalability
Addressing the scalability requirements of the system, considering potential traffic doubling at regular intervals, exploring the possibility of sharding and re-sharding data for parallel processing, and assessing the adaptability to volume changes.

Reliability
Ensuring the system consistently performs its intended functions, verifying its capability to handle varying loads and data volumes without compromising functionality.

Fault Tolerance
Identifying the types of failures the system can tolerate, establishing guaranteed semantics for the processing or output frequency of data, and detailing the storage and recovery mechanisms for in-memory states [3].

Versioning
Versioning is one of the mission critical capabilities that should be fully supported for iterative artifacts ingestion. It not only allows us to look at the history of each unique artifact but also leverages the extensibility that the platform offers to allow customization of how the version of each ingestion should be described. The versioning can also tell us if the ingestion is a publish to Production or an inflight change and yet made it to Production.

Within this paper, we meticulously detail the overarching architecture, present a comprehensive overview of the involved components, elucidate the system's design intricacies, and conscientiously adhere to industry’s best practices while integrating the considerations into our design decisions.

The structure of this paper is as follows. In Section 2, we provide a thorough examination of the system's architecture and delineate the intricate relationships among its integral components and takes deep dive into some of the key components, delves into the specifics within the system, shedding light on their roles, functionalities, and contributions to the overall framework. Section 3 serves as a reflection on the insights gained during the design and implementation phases of our large-scale distributed artifact ingestion platform. By sharing lessons learned, we aim to contribute valuable knowledge to the broader community engaged in similar endeavors. Finally, in Section 4, we conclude by summarizing the impact of these well-designed and developed data ingestion pipelines on machine learning and advanced analytics.

Systems Overview

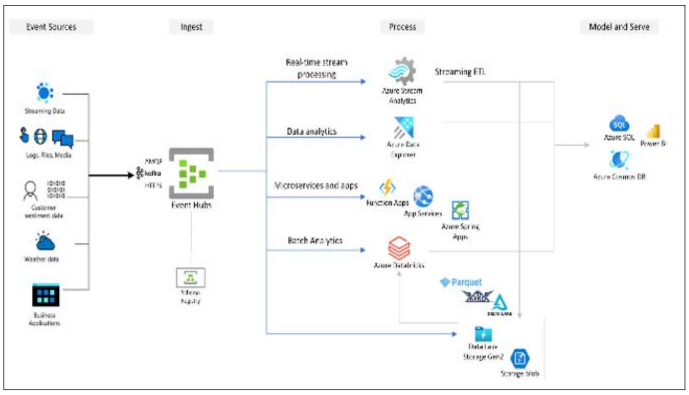


Figure 1: An Overview of the Systems Involved in Real-Time Data Processing: From Data Sources on the Left, Ingestion and Data Processors in the Middle, to Data Stores for Analysis on the Right

Designing and building large-scale distributed artifact ingestion platforms for Machine Learning and advanced analytics involves a multifaceted system. In this section, we offer a comprehensive overview of the architecture and diverse components integral to this ingestion pipeline, leveraging Azure services and other tools.

Illustrated in Figure 1 is the holistic architecture, depicting individual components and services. This depiction showcases the process of capturing data from diverse sources, ingesting and storing streaming data for real-time analytics, responding to events in real-time, and subsequently transporting the data across various processors based on specific use cases. Finally, the processed data is seamlessly fed into dependent systems for advanced data analytics and machine learning applications.

Data Sources
The exponential growth of data in contemporary ecosystems from various sources contributes to this expanding data landscape, encompassing structured and unstructured formats.

Structured data, often originating from relational databases like Oracle database, SQL server, Azure SQL etc., transactional systems, and organized datasets, poses unique challenges in terms of schema variability and evolving data structures.

In contrast, unstructured data, derived from sources like social media, sensor logs, and textual documents, presents challenges related to the lack of predefined data models. Additionally, the influx of streaming data from real-time sources like mobile devices and IOT devices further complicates the data ingestion process.

These diverse data sources demand a sophisticated ingestion platform capable of seamlessly assimilating data in its varied forms, ensuring consistency, and enabling effective utilization for machine learning and advanced analytics applications. The focus, therefore, lies in devising a versatile artifact ingestion platform capable of handling the intricacies posed by the heterogeneity and dynamic nature of data from diverse sources.

Ingestion
In the realm of building a robust artifact ingestion platform, the utilization of streaming libraries such as Apache Kafka, Apache Flink, and analogous frameworks plays a pivotal role in seamlessly capturing data from diverse sources and channeling it towards Azure Event Hubs. These libraries provide a powerful foundation for real-time data processing, ensuring reliability, scalability, and fault tolerance [4].

Apache Kafka
Apache Kafka is a scalable, fault tolerant and highly available distributed platform for processing and storing data streams [5]. It consists of three main components: the Kafka cluster, the Connect framework with the Connect API and the Streams stream processing library with the Streams API. The data ingestion capabilities are achieved by utilizing both Streams and Connect APIs, creating pipelines between heterogeneous data sources and the data storage using publish/subscribe model based on Kafka topics. Renowned for its distributed and fault-tolerant nature, it proves instrumental in handling high-throughput data streams. It’s publish-subscribe model facilitates the integration of data from multiple sources into Azure Event Hubs.

Example Code
```
# Kafka Producer Example
from kafka import KafkaProducer
producer = KafkaProducer
(bootstrap_servers='your_kafka_brokers')
producer.send('topic_name', value='your_data')
```

Apache Flink
Apache Flink excels in stream processing, offering stateful computations and event time processing. It seamlessly integrates with Azure Event Hubs, enabling efficient and scalable data streaming workflows [6].

Example Code

```
// Flink DataStream API Example
DataStream<String> stream =
env.addSource(new FlinkKafkaConsumer<>
("topic", new SimpleStringSchema(), properties));
stream.addSink(new FlinkEventHubsSink<>());
```

Apache Spark
Spark is a cluster computing solution and an intra-memory software framework for data processing based on the MapReduce model for the purpose of supporting operations such as interactive queries and data flow processing [1]. It is based on the Resilient Distributed Datasets (RDD) memory abstraction that enables a wide range of intra-memory computations with fault tolerance and provides a programming interface for performing operations. Apache Spark is based on Spark Core and four processing components. Spark Core contains RDD and is responsible for application scheduling, distribution and execution. Spark SQL component allows structured data processing based on RDD and contains Dataset API that extends data types with datasets and data frames. Spark Streaming contains framework for stream processing based on micro batches and utilizes discretized stream abstraction DStream responsible for splitting data streams into smaller batches inside small fault tolerant windows. MLLib programming library provides machine learning capabilities over data streams. GraphX library enables graph datasets creation using RDD API, which allows the data storage into graph nodes and edges.

Apache Spark Streaming, an extension of the Spark core, provides a micro-batch processing model, allowing for near-real-time processing of streaming data. It seamlessly integrates with Azure Event Hubs, offering fault tolerance and ease of use. Azure also supports Databricks integration where all these data processing and machine learning jobs using Spark can run on Spark cluster.

Example Code

```
// Spark Streaming Example
val stream =
SparkUtils.createDirectStreamFromEventHubs
(parameters, eventHubParams)
stream.foreachRDD { rdd =>
  // Process the RDD and send results to Azure Event Hubs
}
```

Comparative Analysis
Scalability
Kafka's partitioning mechanism allows for horizontal scalability, while Flink's parallelism can be adjusted dynamically, catering to varying workloads.

Processing Semantics
Flink's event time processing ensures accurate results even in the presence of delayed or out-of-order events, offering a nuanced advantage over Kafka. Spark Streaming's micro-batch model contrasts with the continuous processing paradigm of Flink, offering a different approach based on processing small, periodic batches of data. Spark's unified platform simplifies development with a consistent API for batch and stream processing, promoting ease of use.

Fault Tolerance
Both Kafka and Flink provide fault-tolerant mechanisms, Kafka relying on replication and Flink utilizing distributed snapshots. There are other valuable mentions and provide greater benefits in certain use cases like Apache Storm, Apache Samza etc.,

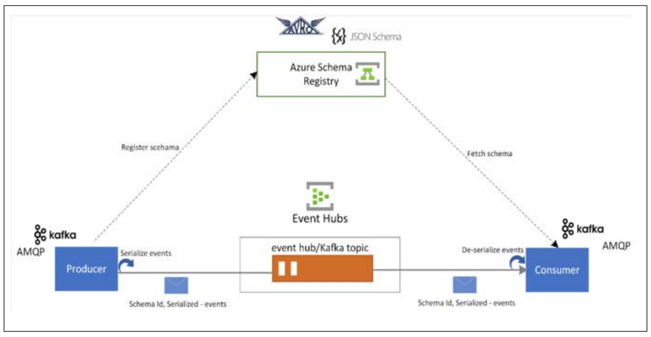
Azure Event Hubs
Azure Event Hubs is a cloud native data streaming service that can stream millions of events per second, with low latency, from any source to any destination. Azure Event Hubs is the preferred event ingestion layer of any event streaming solution that you build on top of Azure. It seamlessly integrates with data and analytics services inside and outside Azure to build your complete data streaming pipeline [7,8].

Ensure proper configuration of Azure Event Hubs credentials, connection strings, and policies for secure and reliable data transfer.

Example Code

```
# Azure Event Hubs Producer Example
from azure.eventhub import
EventHubProducerClient, EventData
producer = EventHubProducerClient
.from_connection_string('your_connection_string')
event_data = EventData('your_data')
producer.send(event_data)
```

Schema Registry in Azure Event Hubs



Azure Schema Registry in Event Hubs provides a centralized repository for managing schemas of events streaming applications. It ensures data compatibility and consistency across event producers and consumers. Schema Registry enables seamless schema evolution, validation, and governance, and promoting efficient data exchange and interoperability [9,10].

Data Processing
Once, the data is ingested and is emitted as an event from the event hubs, that can be processed by several types of tools or services depending on the specific use case. We discuss some common types:

Real-Time Processing of Streaming Events with Azure Stream Analytics
Event Hubs integrates seamlessly with Azure Stream Analytics to enable real-time stream processing. With the built-in no-code editor, you can effortlessly develop a Stream Analytics job using drag-and-drop functionality, without writing any code.

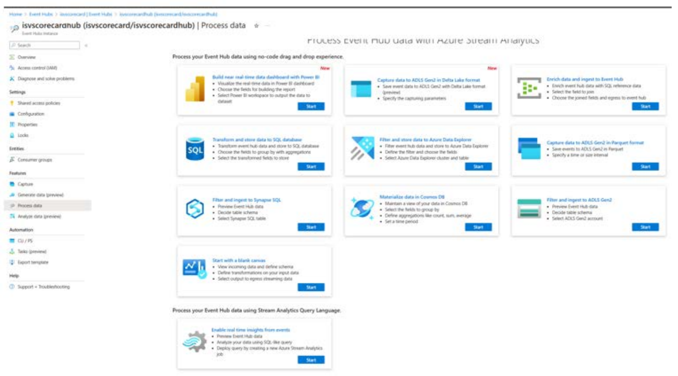
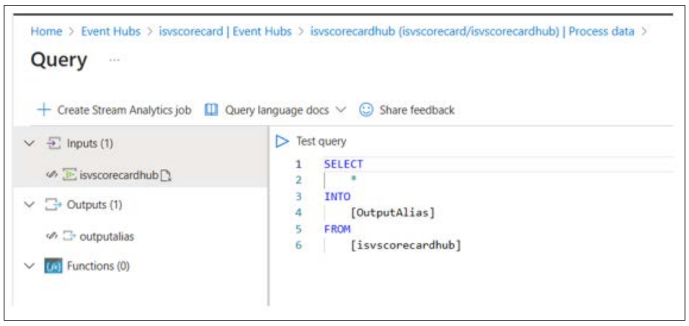

isvscorecardhub (isvscorecard/isvscorecardhub) | Process data
Process Event Hub data with Azure Stream Analytics
Process your Event Hub data using no-code drag and drop experience.
Build near real-time data dashboard with Power BI
Capture data to ADLS Gen2 in Delta Lake format
Enrich data and ingest to Event Hub
Transform and store data to SQL database
Filter and store data to Azure Data Explorer
Capture data to ADLS Gen2 in Parquet format
Filter and ingest to Synapse SQL
Materialize data in Cosmos DB
Filter and ingest to ADLS Gen2
Start with a blank canvas
Process your Event Hub data using Stream Analytics Query Language.
Enable real time insights from events
Start

Figure 2: Process Event Hub Data with Stream Analytics


Home > Event Hubs > isvscorecard | Event Hubs > isvscorecardhub (isvscorecard/isvscorecardhub) | Process data >
Query
Create Stream Analytics job
Query language docs
Share feedback
Inputs (1)
isvscorecardhub
Outputs (1)
outputalias
Functions (0)
Test query

```
SELECT
    *
INTO
    [OutputAlias]
FROM
    [isvscorecardhub]
```

Alternatively, developers can use the SQL-based Stream Analytics query language to perform real-time stream processing and take advantage of a wide range of functions for analyzing streaming data.

Azure Stream Analytics and Integration with Azure Machine Learning Services

Azure Stream Analytics is a fully managed, real-time analytics service designed to help you analyze and process fast moving streams of data that can be used to get insights, build reports or trigger alerts and actions. Machine Learning models can be implemented as a user-defined function (UDF) in Azure Stream Analytics jobs to do real-time scoring and predictions on the streaming input data. Azure Machine Learning supports use of any popular open-source tools, such as TensorFlow, scikit-learn, or PyTorch, to prep, train, and deploy models. Adding machine learning model as a function to the Stream Analytics pipeline involves the following:

Deploy ML Model as a Web Service

Utilize Azure Machine Learning's capabilities to deploy the trained model as a web service, enabling seamless integration with other Azure services. Emphasize the crucial role of the associated Swagger documentation in defining the input and output schemas for Stream Analytics comprehension [11].

Data Format Adherence

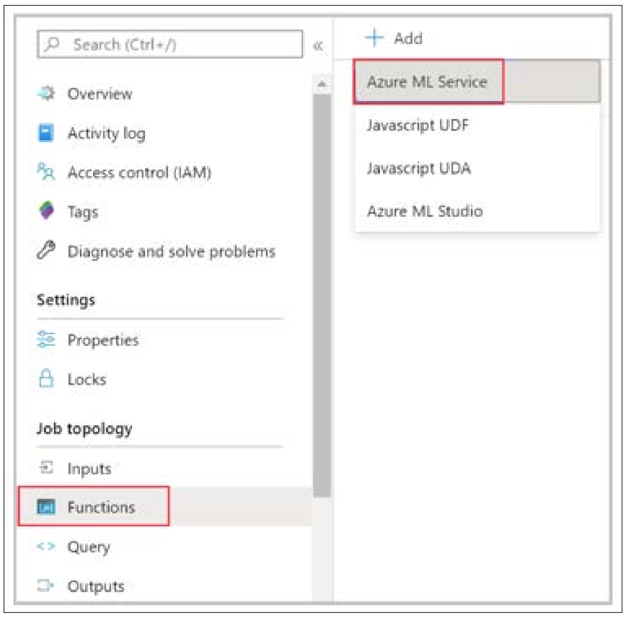
Ensure the web service's adherence to JSON serialization for both input and output data, aligning with Stream Analytics requirements.

Scalable Production Deployment

Underscore the significance of deploying the model on Azure Kubernetes Service (AKS) to cater to high-scale production environments. Highlighting the potential performance degradation and latency concerns if the web service's capacity is insufficient to handle the incoming request volume.

Integrate the Machine Learning Service with Streaming Analytics Service

Navigate your Stream Analytics job in the Azure portal, select Functions under Job topology and then select Azure Machine Learning Service from the dropdown [9].



Invoking Azure Machine Learning UDF from Stream Analytics Query

In the context of Stream Analytics queries that interact with Azure Machine Learning (AML) UDFs, the process involves the creation of a JSON serialized request to be sent to the associated machine learning web service. This request is meticulously crafted based on a model-specific schema, which Stream Analytics intelligently deduces from the endpoint's Swagger definition. This integration ensures seamless communication between the Stream Analytics job and the Azure Machine Learning model, enabling real-time inference within the data processing pipeline. Below is an illustrative example of a Stream Analytics query demonstrating how to invoke an Azure Machine Learning UDF:

```
SELECT udf.score
(<model-specific-data-structure>)
INTO output
FROM input
WHERE <model-specific-data-structure>
is not null
```

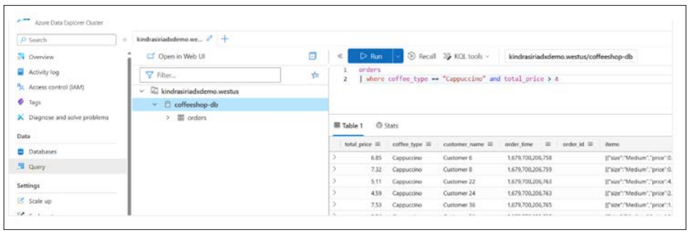
Create Pandas or PySpark Data Frame

You can use the WITH clause to create a serialized JSON Data Frame that can be passed as input to your Azure Machine Learning UDF as shown below. The following query creates a Data Frame by selecting the necessary fields and uses the Data Frame as input to the Azure Machine Learning UDF [9].

```
WITH Dataframe AS (
SELECT vendorid, weekday,
pickuphour, passenger, distance
FROM input)
SELECT udf.score(Dataframe)
INTO output
FROM Dataframe
WHERE Dataframe is not null
```

Exploring Streaming Data with Azure Data Explorer

Azure Data Explorer (also called kusto) is a fully managed platform for big data analytics that delivers high performance and allows for the analysis of large volumes of data in near real time. By integrating Event Hubs with Azure Data Explorer, you can easily perform near real-time analytics and exploration of streaming data [9].



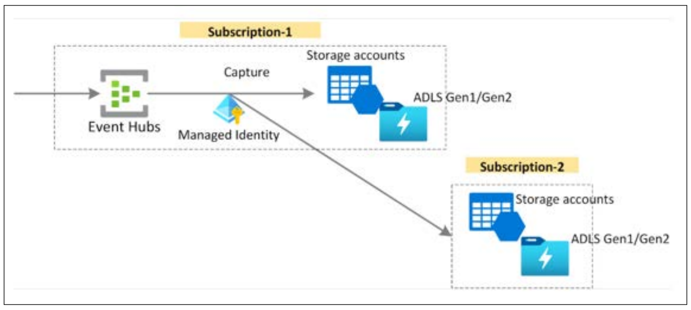
```
orders
| where coffee_type == "Cappuccino" and total_price > 4
```

total_price	coffee_type	customer_name	order_time	order_id	items
6.85	Cappuccino	Customer 6	1,679,700,206,758		[{"size":"Medium","price":0.
7.32	Cappuccino	Customer 8	1,679,700,206,759		[{"size":"Medium","price":0.
5.11	Cappuccino	Customer 22	1,679,700,206,763		[{"size":"Medium","price":4.
4.59	Cappuccino	Customer 24	1,679,700,206,763		[{"size":"Medium","price":2.
7.53	Cappuccino	Customer 36	1,679,700,206,765		[{"size":"Medium","price":1.

Figure 4: Azure Data Explorer Cluster

Capture Streaming Data for Long Term Retention and Batch Analytics

Capture your data in near-real time in an Azure Blob storage or Azure Data Lake Storage for long-term retention or micro-batch processing. You can achieve this behavior on the same stream you use for deriving real-time analytics. Setting up capture of event data is fast.



Model and Serve

The processed data can be fed to reporting systems like PowerBI or stored and can be consumed by other systems to represent that analyzed data for other use cases.

Lessons Learned

While designing and building these large-scale distributed data ingestion pipelines there are different things that come into play which become more complex. Providing the same level of experience for the services at an exponential level of growth involves solving problems related to the underlying infrastructure and horizontal scaling of resources to serve the needs.

Latency

The system should be able to scale horizontally to maintain similar latency with exponential growth in data. So, designing the system to be able to parallelize and run across multiple nodes, stream the data and get consistent results will be important.

Ease of Debugging

In traditional applications, iterative development is facilitated by storing data and rerunning queries as needed. However, in this stream processing system, where data is maintained in different formats, iterating becomes challenging, and updating operators may yield different results on new data streams. So, maintaining different versions of data and being able to use that to test the outputs that the transformation pipeline generates helps a lot with debugging for any scenarios.

Ease of Monitoring and Operations

Monitoring and operation of deployed apps are vital. We use alerts to detect processing lag, ensuring timely adjustments. Creating dashboards and monitoring key metrics to check the overall health and functionality of the system plays a crucial role.

Conclusion

In this paper, we proposed and implemented a distributed data ingestion platform using Azure stack for extracting valuable insights and actionable knowledge from different data streams. Our proposed architecture supports both real-time and historical data analytics using flexible data processing model. This architecture used Azure cloud services and open-source tools optimized for large scale distributed ingestion pipelines.

We implemented a large-scale distributed data ingestion and processing platform that collaborates with multiple systems for analytics and machine learning jobs. This demonstrates the amenability of our architecture that can scale depending on the load and can be applied in different scenarios for batch processing or stream processing to feed reporting or further analysis using Machine Learning algorithms [12-18].

References

1. Shayna Joubert (2019) Beyond The Buzzword: What Does Data-Driven Decision-Making Really Mean? Northeastern University Graduate Program https://graduate.northeastern.edu/resources/data-driven-decision-making./.
2. Mahak Agarwal (2020) Why the Data You Use Is More Important Than the Model Itself. Medium https://medium.com/swlh/why-the-data-you-use-is-more-important-than-the-model-itself-4a49736ea70c.
3. (2017) Is Data More Important Than Algorithms In AI? Forbes https://www.forbes.com/sites/quora/2017/01/26/is-data-more-important-than-algorithms-in-ai/?sh=111664a542c1.
4. Guoqiang JC, Janet LW, Shridhar Ir, Anshul J, Ran L (2016) Realtime Data Processing at Facebook. ACM 1087-1098.
5. Kleppmann M (2017) Designing Data-Intensive Applications. O'Reilly Media https://www.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/.
6. Tomislav H; Josip P (2021) An Overview of Current Trends in Data Ingestion and Integration. 44th International Convention on Information, Communication and Electronic Technology (MIPRO), Opatija, Croatia 1265-1270.
7. Azure Event Hubs - how it works. Microsoft https://learn.microsoft.com/en-us/azure/event-hubs/event-hubs-about#how-it-works.
8. A Flink. Apache Flink https://flink.apache.org/.
9. Azure Event Hubs. Microsoft https://learn.microsoft.com/en-us/azure/event-hubs/event-hubs-about.
10. Process data from your event hub using Azure Stream Analytics. Microsoft https://learn.microsoft.com/en-us/azure/

event-hubs/process-data-azure-stream-analytics.
11. Deploy and score a machine learning model by using an online endpoint. Microsoft https://learn.microsoft.com/en-us/azure/machine-learning/how-to-deploy-online-endpoints.
12. Tim Stobierski (2019) ADVANTAGES OF DATA-DRIVEN DECISION-MAKING. HBS https://online.hbs.edu/blog/post/data-driven-decision-making.
13. Apache Kafka. Kafka https://kafka.apache.org/.
14. What is Apache Spark? Apache Spark https://spark.apache.org/.
15. Use Azure Schema Registry in Event Hubs from Apache Kafka and other apps. Microsoft https://learn.microsoft.com/en-us/azure/event-hubs/schema-registry-overview.
16. Integrate Azure Stream Analytics with Azure Machine Learning. Microsoft https://learn.microsoft.com/en-us/azure/stream-analytics/machine-learning-udf?source=recommendations.
17. Chandan Prakash (2018) Spark Streaming vs Flink vs Storm vs Kafka Streams vs Samza : Choose Your Stream Processing Framework. Medium https://medium.com/@chandanbaranwal/spark-streaming-vs-flink-vs-storm-vs-kafka-streams-vs-samza-choose-your-stream-processing-91ea3f04675b.
18. Paula TS, Adnan A, Guy GG, Guy H, Francois C (2018) An Ingestion and Analytics Architecture for IoT Applied to Smart City Use Cases. IEEE Internet of Things Journal 5: 765-774.