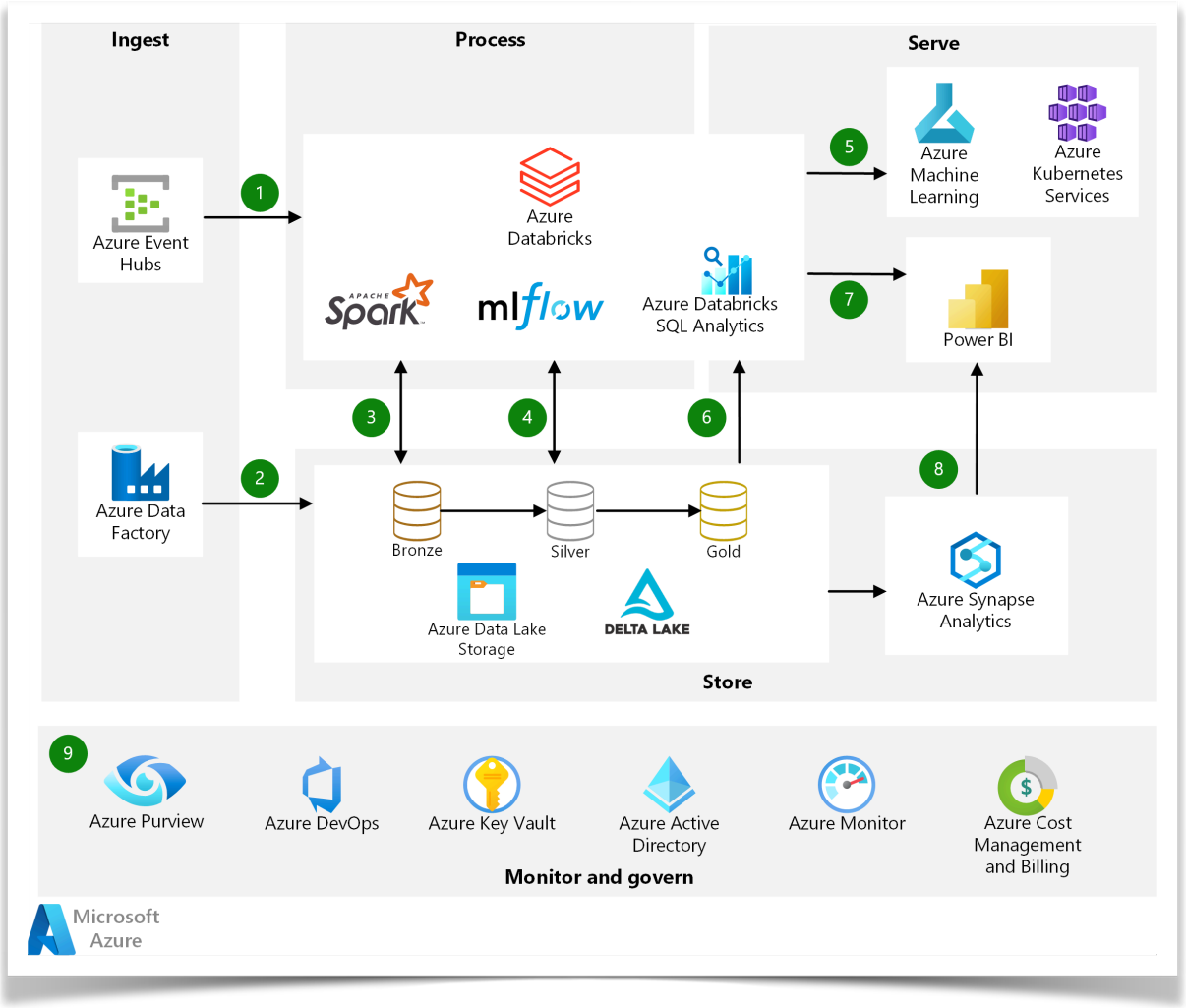
1. **Explain what is Data Engineering?**

   Data engineering is the process of designing, building, and maintaining the infrastructure and systems that enable the collection, storage, processing, and analysis of large amounts of data. It involves tasks such as data integration, data transformation, data storage and retrieval, data quality assurance, and data pipeline development. Data engineering is a crucial component of data-driven organizations and enables them to effectively leverage the power of data to gain insights, make informed decisions, and drive business growth.

2. **End to End Data Engineering Architecture?**

   Data engineering architecture refers to the design and structure of the systems and processes used to manage and process data in an organization. A well-designed data engineering architecture enables efficient, reliable, and scalable data processing, storage, and retrieval.



A typical data engineering architecture may involve several layers, including:

Data Sources: This layer includes all the systems and applications that generate data, such as databases, sensors, and web services.

Data Ingestion: This layer is responsible for collecting and aggregating data from various sources and bringing it into the data processing pipeline. Tools such as Apache Kafka, Apache Flume, and AWS Kinesis are often used for this purpose.

Data Storage: This layer involves storing data in a structured, semi-structured, or unstructured format, depending on the nature of the data. Common data storage systems include relational databases, NoSQL databases, and data lakes.

Data Processing: This layer is responsible for transforming, cleaning, and enriching the data to make it useful for downstream analytics and machine learning. Tools such as Apache Spark, Apache Beam, and AWS Glue are commonly used for this purpose.

Data Analytics: This layer involves using various tools and techniques to analyze the data and gain insights. This may involve data visualization, machine learning, and other data analysis techniques.

Data Delivery: This layer is responsible for delivering the data and insights to end-users or downstream applications, such as dashboards, reports, and APIs.

Overall, a good data engineering architecture should be scalable, reliable, secure, and flexible enough to accommodate changing data requirements and business needs.

3. How does the day to day look like for a Data Engineer?

The day-to-day responsibilities of a data engineer can vary depending on the organization, but here are some common tasks that a data engineer may perform:

Data Ingestion: Collecting data from various sources and systems and bringing it into the data processing pipeline. This may involve setting up ETL (Extract, Transform, Load) pipelines or working with tools such as ADF, AZCopy, FTP Tools, API Calls, Apache Kafka, Apache Flume, or AWS Kinesis.

Data Storage: Designing and implementing data storage solutions that can accommodate large volumes of structured, semi-structured, or unstructured data in data lake like Azure Data Lake Gen2, AWS S3, Google Cloud Storage, HDFS. This may involve working with Cloud Data Warehouses like Azure SQL DWH, AWS RedShift, Google Big Query and snowflake databases, NoSQL databases.

Data Transformation: Transforming and cleaning the data to make it useful for downstream analytics and machine learning. This may involve using tools such as Apache Spark,Databricks Spark SQL, Synapse Analytics with SQL/Spark, Apache Beam, or AWS Glue.

Data Quality: Ensuring the quality and consistency of the data by implementing data validation, verification, and monitoring processes using pyspark Transformations in Databricks, synapse analytics or any other cloud Data Warehouses using SQL.

Performance Optimization: Improving the performance of data processing pipelines by optimizing queries, improving data partitioning, or using caching strategies.

Data Security: Ensuring the security of the data by implementing encryption, access controls, and other security measures.

Documentation: Creating and maintaining documentation for the data processing pipelines and data storage systems to help ensure that they can be easily understood and maintained by other members of the team.

Collaboration: Working closely with data analysts, data scientists, and other stakeholders to understand their requirements and to ensure that the data engineering systems meet their needs.

Overall, the day-to-day responsibilities of a data engineer involve designing, building, and maintaining the infrastructure and systems that enable an organization to effectively manage and process large volumes of data.

4. Explain components of a typical data platform

- ✅ Data Ingestion
- ✅ Data Storage
- ✅ Data Transformation
- ✅ Data Quality
- ✅ Performance Optimization
- ✅ Data Security
- ✅ Documentation
- ✅ Collaboration
- ✅ BI Reporting
- ✅ Advanced Analytics (Data Science, Machine Learning and Deep Learning)

5. What are the various data engineering tools and technologies you used?

The following Azure services have been used in the architecture:

- ✅ Azure Synapse Analytics
- ✅ Azure DataBricks
- ✅ Databricks SQL Analytics
- ✅ Azure Data Lake Gen2
- ✅ Azure Cosmos DB
- ✅ Azure Cognitive Services
- ✅ Azure Machine Learning
- ✅ Azure Event Hubs

- ✅Azure IoT Hub
- ✅Azure Stream Analytics
- ✅Microsoft Purview
- ✅Azure Data Share
- ✅Microsoft Power BI
- ✅Azure Active Directory
- ✅Azure Cost Management
- ✅Azure Key Vault
- ✅Azure Monitor
- ✅Microsoft Defender for Cloud
- ✅Azure DevOps
- ✅Azure Policy
- ✅GitHub

6. What are various cloud data platforms you worked? explain data services available in one of the cloud provider.

Amazon Web Services (AWS): AWS offers a wide range of data-related services, including data storage (S3, EBS, EFS, etc.), data processing (EC2, EMR, Glue, etc.), and data analytics (Athena, Redshift, QuickSight, etc.).

Microsoft Azure: Azure offers a range of data-related services, including data storage (Blob storage, Azure Files, etc.), data processing (HDInsight, Azure Data Factory, etc.), and data analytics (Azure Synapse Analytics, Power BI, etc.).

Google Cloud Platform (GCP): GCP offers a range of data-related services, including data storage (Cloud Storage, Cloud SQL, etc.), data processing (Cloud Dataproc, Dataflow, etc.), and data analytics (BigQuery, Data Studio, etc.).

Snowflake: Snowflake is a cloud-based data warehousing platform that enables users to store, process, and analyze large volumes of data.

IBM Cloud: IBM Cloud offers a range of data-related services, including data storage (Cloud Object Storage, Cloud Databases, etc.), data processing (IBM Cloud Pak for Data, Watson Studio, etc.), and data analytics (IBM Cognos Analytics, Watson Discovery, etc.).

7. What are the the various programming languages used in data engineering

There are several programming languages that are commonly used in data engineering, including:

Python: Python is a popular language for data engineering due to its ease of use, versatility, and large selection of data-related libraries and frameworks, such as Pandas, NumPy, and Apache Spark. Python can be used for data processing, data analysis, and building data pipelines.

SQL: SQL (Structured Query Language) is a language used for querying and manipulating data in relational databases. It is a standard language that is widely used in data engineering, particularly for data warehousing and data analytics.

Java: Java is a commonly used language for building large-scale data processing systems, particularly those that use distributed computing frameworks such as Apache Hadoop and Apache Spark.

Scala: Scala is a high-performance language that is used in distributed computing frameworks such as Apache Spark. It is often used in conjunction with Java to build large-scale data processing systems.

R: R is a language that is commonly used for statistical computing and data analysis. It has a large selection of libraries and frameworks that make it well-suited for data engineering tasks such as data cleaning, data visualization, and data analysis.

JavaScript : Snowflake supports the use of JavaScript in Snowflake Stored Procedures and User-Defined Functions (UDFs).JavaScript can be used in Stored Procedures and UDFs to implement custom business logic and data transformations within Snowflake. This can include manipulating data, calling external APIs, and performing calculations.

Other programming languages that are commonly used in data engineering include **C++**, **Perl**, and **Go**. The choice of language will depend on factors such as the specific requirements of the project, the technical expertise of the team, and the available tools and libraries.

8. Why do data engineers need Python knowledge

Python is a popular choice for data engineering for several reasons:

Ease of use: Python is an easy-to-learn language with a simple syntax that makes it accessible for beginners. This means that data engineers can quickly get up to speed with Python and start building data pipelines.

Versatility: Python has a wide range of libraries and frameworks that are specifically designed for data processing, such as Pandas, NumPy, and Apache Spark. This makes it a powerful language for building data engineering pipelines, as well as for performing data analysis and machine learning tasks.

Interoperability: Python can easily interface with other languages and technologies commonly used in data engineering, such as SQL databases and big data platforms like Hadoop and Spark.

Community: Python has a large and active community of data engineers and data scientists, who contribute to open-source libraries and tools that make data engineering tasks faster and more efficient.

Scalability: Python's ability to scale horizontally through distributed computing frameworks like Apache Spark and Dask make it suitable for processing large data sets in parallel.

Flexibility: Python is a multi-purpose language that can be used for a variety of tasks beyond data engineering and data science. For example, it is used for web development, automation, and scripting.

Better support for deep learning: Python has gained popularity in recent years as a language for deep learning, and as such, it has several widely used deep learning libraries, such as TensorFlow, Keras, and PyTorch. These libraries are well-supported in Databricks, making Python a natural choice for data engineers and data scientists who are working on deep learning projects.

Integration with Jupyter Notebooks: Databricks supports the use of Jupyter Notebooks, which are a popular platform for interactive data analysis and visualization using Python. This integration enables data engineers and data scientists to work seamlessly in a single environment.

9. Why do data engineers need SQL knowledge

SQL (Structured Query Language) is a key tool for data engineering, as it allows data engineers to interact with and manipulate data stored in relational databases.

Some common use cases for SQL in data engineering include:

Data extraction: SQL can be used to extract data from databases and load it into data pipelines for further processing.

Data transformation: SQL can be used to transform data by manipulating or joining tables, filtering data based on certain conditions, or aggregating data into new forms.

Data quality: SQL can be used to ensure data quality by performing data validation checks, such as checking for missing or duplicate values.

Data warehousing: SQL can be used to create and manage data warehouses, which are centralized repositories of data used for reporting and analysis.

ETL (Extract, Transform, Load): SQL can be used in ETL pipelines to extract data from different sources, transform it into the desired format, and load it into the target data store.

SQL is a widely used and powerful tool for data engineering, and it is supported by many different relational database management systems (RDBMS). Additionally, the use of SQL can help ensure data integrity and consistency across different systems and applications.

10.What are the different data sources / targets you used in your project?

Data Engineering can have any number of sources like

Relational databases: These are databases that store data in tables with predefined relationships between them. Common examples include MySQL, PostgreSQL, Oracle, and Microsoft SQL Server.

NoSQL databases: These are databases that store data in non-tabular structures, such as key-value stores, document databases, and graph databases. Common examples include MongoDB, Cassandra, and Neo4j.

Cloud storage: Cloud storage services like Amazon S3, Google Cloud Storage, and Microsoft Azure Blob Storage are commonly used as sources and targets for data pipelines.

APIs: APIs (Application Programming Interfaces) allow data to be accessed and integrated from various online services and applications. Common examples include REST APIs and SOAP APIs.

File formats: Data can be stored and exchanged in a variety of file formats, such as CSV, JSON, XML, Parquet, and Avro. These formats are often used as sources and targets for data pipelines.

Message queues: Message queues, such as Apache Kafka and RabbitMQ, are used to send and receive data between different systems.

11. What is data ingestion means? what tools you used for data ingestion.

Ingestion in cloud computing refers to the process of bringing data from external sources into a cloud-based storage or processing system. This is a critical step in data engineering, as it enables organizations to centralize their data and make it accessible for analysis, reporting, and other purposes.

There are various types of ingestion tools that are commonly used in cloud computing, including:

Batch ingestion tools: These tools are used for processing large volumes of data in batches, typically on a daily or weekly basis. Examples of batch ingestion tools include Apache Hadoop, Apache Spark, and Apache Flume.

Real-time ingestion tools: These tools are used for processing data in real-time or near real-time, typically with low latency and high throughput. Examples of real-time ingestion tools include Azure Event Hub, Azure IOT, Apache Kafka, AWS Kinesis, and Google Cloud Pub/Sub.

Cloud-native ingestion tools: These tools are specifically designed for use in cloud environments and often provide seamless integration with cloud-based storage and processing systems. Examples of cloud-native ingestion tools include AWS Glue, Google Cloud Dataflow, and Microsoft **Azure Data Factory**.

Database replication tools: These tools are used to replicate data from one database to another, typically for disaster recovery or high availability purposes. Examples of database replication tools include AWS Database Migration Service, Google Cloud SQL, and Microsoft SQL Server Replication.

File transfer tools: These tools are used for transferring files from one location to another, typically over the internet. Examples of file transfer tools include AWS Transfer for SFTP, Google Cloud Storage Transfer Service, and Microsoft Azure File Sync.

12. How do you read / move data from on-premise to cloud? did you work with some tools.

Cloud Data Migration Tools: These tools are specifically designed for use in cloud environments and often provide seamless integration with cloud-based storage and processing systems. Examples of cloud-native ingestion tools include AWS Glue, Google Cloud Dataflow, and Microsoft **Azure Data Factory**.

Third-party data integration tools: There are many third-party data integration tools available that can help you ingest data from on-premises to the cloud. These tools offer features such as data transformation, data validation, and data cleansing, and support a wide range of data sources and targets.

Custom scripts and APIs: If you have specific data integration requirements, you can develop custom scripts or APIs to ingest data from on-premises to the cloud. This approach can be time-consuming and complex, but it provides more flexibility and control over the data integration process.

13. What are the different file types you used in your day to day work?

In data engineering, there are several types of data files that are commonly used to store and process data. Here are a few examples:

CSV files: CSV (Comma-Separated Values) files are a simple and widely used format for storing tabular data. Each row represents a record, and each column represents a field in the record. CSV files can be easily imported into a variety of data processing tools and databases.

JSON files: JSON (JavaScript Object Notation) files are a lightweight data format that is easy to read and write. They are commonly used for web-based applications and for exchanging data between different programming languages. JSON files are structured as key-value pairs and can contain nested objects and arrays.

Parquet files: Parquet is a columnar storage format that is optimized for big data processing. It is designed to be highly efficient for analytics workloads, allowing for faster query performance and reduced storage costs. Parquet files are often used in data warehousing and data lake environments.

Avro files: Avro is a binary data format that is designed to be compact and efficient. It supports schema evolution, meaning that the schema can be changed over time without breaking existing applications. Avro files are often used in Hadoop and other big data processing frameworks.

ORC files: ORC (Optimized Row Columnar) files are another columnar storage format that is designed for fast data processing. ORC files are highly compressed, making them efficient to store and transmit over networks. They are commonly used in Hadoop and other big data processing environments.

Delta files: Delta is a file format created by Databricks that is designed for building data lakes and data warehouses. Delta files are based on Parquet files but add transactional capabilities to support updates, deletes, and merges. Delta files are designed to be highly scalable and performant, making them well-suited for big data processing.

XML files: XML (Extensible Markup Language) is a markup language that is used to store and transport data. XML files are structured as nested elements, with each element representing a record or data item. XML is a flexible and self-describing format that can be used for a wide range of data processing needs, including web services, document exchange, and database integration.

The choice of data file format depends on the specific needs of the data processing and storage system. Data engineers need to consider factors such as performance, scalability, flexibility, and interoperability when selecting the appropriate format for a particular use case.

14. What are 3 prominent big data file types?

Orc, Avro, and Parquet are three popular columnar file formats used in data engineering

Parquet files: Parquet is a columnar storage format that is optimized for big data processing. It is designed to be highly efficient for analytics workloads, allowing for faster query performance and reduced storage costs. Parquet files are often used in data warehousing and data lake environments.

Avro files: Avro is a binary data format that is designed to be compact and efficient. It supports schema evolution, meaning that the schema can be changed over time without breaking existing applications. Avro files are often used in Hadoop and other big data processing frameworks.

ORC files: ORC (Optimized Row Columnar) files are another columnar storage format that is designed for fast data processing. ORC files are highly compressed, making them efficient to store and transmit over networks. They are commonly used in Hadoop and other big data processing environments.

15. What are the differences between ORC, AVRO and Parquet files?

If you have large-scale data processing needs: Orc and Parquet are both designed for efficient storage and processing of large datasets. They can provide high performance and scalability for data processing jobs that require processing of large volumes of data.

If you need advanced compression capabilities: Parquet and Orc support more advanced compression techniques like Zstandard and Snappy, which can lead to higher compression ratios and faster query performance. If you have specific compression requirements, these file formats may be a better choice than Avro.

If you need strong schema evolution support: Avro has strong support for schema evolution, meaning that the schema can be changed over time without breaking existing applications. If you expect your schema to evolve frequently, Avro may be a better choice than Parquet or Orc.

If you need a lightweight and flexible file format: Avro is a lightweight and flexible file format that can be used for a wide range of data processing needs. If you have a small dataset or require more flexible schema management, Avro may be a better choice than Parquet or Orc.

16. What are best use cases for ORC, AVRO, and Parquet files (Explain best suitable data scenarios for each file type)

Orc, Avro, and Parquet are three popular columnar file formats used in data engineering. Here are some key differences between them:

Compression: All three file formats support compression, but the specific compression algorithms and settings vary. Parquet and Orc support more advanced compression techniques like Zstandard and Snappy, which can lead to higher compression ratios and faster query performance. Avro supports simpler compression techniques like deflate and snappy, which are less efficient but more widely supported.

Schema evolution: Avro has strong support for schema evolution, meaning that the schema can be changed over time without breaking existing applications. Parquet and Orc also support schema evolution, but the process is more complex and requires more care to avoid breaking compatibility.

Performance: Parquet and Orc are optimized for big data processing and can handle large volumes of data quickly and efficiently. Avro is more lightweight and may be better suited for smaller datasets or applications that require more flexible schema management.

Integration with data processing tools: All three file formats are widely supported by data processing tools and platforms, including Hadoop, Spark, and others. However, some tools may have better performance or compatibility with certain file formats.

17. How do you convert data from one file format to another file format? for example, CSV to ORC? ORC to Parquet? JSON to CSV?

you can use the following steps to convert a file in Orc format to CSV:

Read the Orc file using the spark.read.orc() function and store it in a DataFrame.

```
orc_df = spark.read.orc("path/to/orc/file")
```

Write the DataFrame to a CSV file using the DataFrame.write() function, specifying the format as “csv”.

```
orc_df.write.format("csv").save("path/to/csv/file")
```

This will save the contents of the Orc file in CSV format to the specified file path.

Similarly, to convert a CSV file to Orc format, you can use the following steps:

Read the CSV file using the spark.read.csv() function and store it in a DataFrame.

```
csv_df = spark.read.csv("path/to/csv/file", header=True, inferSchema=True)
```

Write the DataFrame to an Orc file using the DataFrame.write() function, specifying the format as “orc".

```
csv_df.write.format("orc").save("path/to/orc/file")
```

18. What is metadata?

Metadata refers to data that describes other data. It provides information about the structure, content, quality, and other characteristics of a dataset. Metadata is used to help users understand, interpret, and use data, and it is a critical component of data management and data engineering.

Metadata can include various types of information, such as:

Data type and format
Data source and origin
Date and time of data creation, update, and access
Data schema and relationships
Data quality and completeness
Data privacy and security restrictions
Data access and usage permissions
Metadata can be stored in a separate database or system, or it can be embedded within the data itself, such as in file headers or database schemas. It can be managed manually or automatically using metadata management tools and processes.

Metadata is essential for effective data discovery, integration, transformation, and analysis. It helps to ensure that data is accurate, consistent, and usable across different applications and systems.

19. What are typical scenarios where you get data in the format JSON files?

JSON (JavaScript Object Notation) is a widely used data format that is simple, lightweight, and easy to parse. It is used for a variety of data scenarios, including:

Web APIs: JSON is commonly used as a data format for web APIs, which allow applications to exchange data over the internet. APIs can return JSON data in response to requests from client applications, which can then parse and use the data as needed.

Big data: JSON is often used for storing and processing big data, especially in NoSQL databases like MongoDB and Cassandra. JSON allows for flexible schema design, which can be beneficial for handling unstructured or semi-structured data.

IoT devices: JSON is used to transmit and store data from Internet of Things (IoT) devices, which can generate large amounts of data in real time. JSON can be used to encode data such as sensor readings, location data, and other metadata.

Configuration files: JSON can be used to store configuration data for applications or systems. This can include data such as server settings, user preferences, or other application-specific settings.

Log files: JSON can be used to store log data, which can be analyzed and monitored for application performance, security, or other purposes. JSON log data can be easier to parse and analyze than other formats like plain text or XML.

Overall, JSON is a versatile and widely used data format that can be used in a variety of data scenarios. Its simplicity, flexibility, and compatibility with many programming languages and tools make it a popular choice for developers and data engineers.

20. What are the different types of data you worked in you project?

Explain different types of source data files and target data files in your project. Like csv, tsv , parquet , orc and delta file formats.

21. What is the difference between Pandas and Spark?

Pandas and Spark are both popular data processing tools, but they have some key differences:

Architecture: Pandas is a Python library that runs on a single machine, whereas Spark is a distributed computing system that can run on a cluster of machines.

Data Size: Pandas is suitable for working with small to medium-sized datasets that can fit into memory, while Spark is designed to handle larger datasets that may be too big to fit into memory.

Performance: Due to its distributed architecture, Spark is generally faster than Pandas for large-scale data processing tasks. However, for small datasets that can fit into memory, Pandas can be faster due to its optimized data structures.

Ease of Use: Pandas has a simpler and more user-friendly API compared to Spark, which can require more complex code for certain operations.

In summary, if you are working with large datasets that require distributed processing, Spark is likely the better choice. However, if your data fits into memory and you are comfortable working with Python, Pandas can be a more convenient option.

22. What is best between pandas and Spark? explain why?

if you are working with large datasets that require distributed processing, Spark is likely the better choice. However, if your data fits into memory and you are comfortable working with Python, Pandas can be a more convenient option.

23. What is a cluster? did you work with a cluster?

In Databricks, a cluster is a set of computing resources (such as virtual machines or containers) that are used to process data and run computations. The cluster is managed by the Databricks platform and can be created, configured, and terminated as needed.

Each cluster in Databricks is assigned a specific set of resources, including CPU, memory, and storage, depending on the selected instance type and cluster configuration. The cluster can also be configured with additional libraries, runtime environments, and settings to optimize performance and ensure compatibility with the data being processed.

Clusters in Databricks are typically used to run distributed computing jobs, such as processing large datasets, training machine learning models, or performing real-time data analytics. Multiple users can share a cluster, and the platform automatically manages resource allocation and scheduling to ensure efficient use of the available resources.

24. What is Data Factory? Where do you use it?

Azure Data Factory is a cloud-based data integration service provided by Microsoft Azure. It allows you to create, schedule, and manage data workflows that move and transform data from various sources to destinations.

Azure Data Factory provides a graphical interface and a set of tools to enable you to build and manage data pipelines that can connect to various data sources, including on-premises and cloud-based sources. You can use the tool to perform a variety of data integration tasks, such as data ingestion, data transformation, and data loading.

Some key features of Azure Data Factory include:

Integration with other Azure services, such as Azure Blob Storage, Azure Data Lake Storage, Azure SQL Database, and more.
Support for a variety of data sources, including structured, unstructured, and semi-structured data.
Built-in data transformation and processing using Azure Databricks or HDInsight.
Advanced security and monitoring features, including role-based access control, auditing, and logging.
Overall, Azure Data Factory simplifies the process of building and managing data integration pipelines, allowing organizations to easily move, transform, and analyze data from various sources in a scalable, cost-effective way.

25. How do you connect data factory with on-premise data sources?

To connect Azure Data Factory with on-premises data sources, you can use the following two methods:

Self-hosted integration runtime: This method involves installing a self-hosted integration runtime (IR) on an on-premises machine, which acts as a gateway between your on-premises data sources and Azure Data Factory. Once the self-hosted IR is set up, you can use it to connect to your on-premises data sources and move data between them and Azure.

Azure Data Gateway: This method uses the Azure Data Gateway to connect to on-premises data sources. The Azure Data Gateway is a cloud-based service that provides secure access to on-premises data sources, allowing you to move data between them and Azure Data Factory.

To set up the connection, you would need to follow these general steps:

Create a linked service in Azure Data Factory for the on-premises data source you want to connect to.
For the self-hosted IR method, download and install the self-hosted IR on an on-premises machine, and register it with Azure Data Factory.
For the Azure Data Gateway method, download and install the gateway, and register it with Azure Data Factory.
Configure the linked service to use the self-hosted IR or Azure Data Gateway.
Use the linked service in your data pipeline to move data between the on-premises data source and Azure.

26. Did you work with custom activity / transformation in data factory?

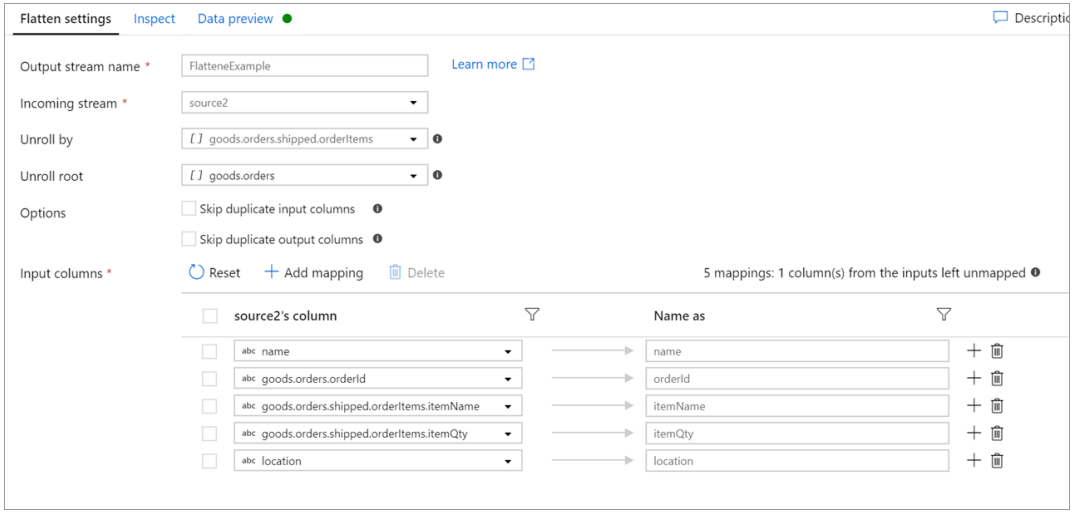
There are two types of activities that you can use in an Azure Data Factory or Synapse pipeline.
Data movement activities to move data between supported source and sink data stores.
Data transformation activities to transform data using compute services such as Azure HDInsight and Azure Batch.
To move data to/from a data store that the service does not support, or to transform/process data in a way that isn't supported by the service, you can create a Custom activity with your own data movement or transformation logic and use the activity in a pipeline. The custom activity runs your customized code logic on an Azure Batch pool of virtual machines.

27. Did you parse JSON in Azure Data factory?

When copying data from JSON files, copy activity can automatically detect and parse the following patterns of JSON files. When writing data to JSON files, you can configure the file pattern on copy activity sink.

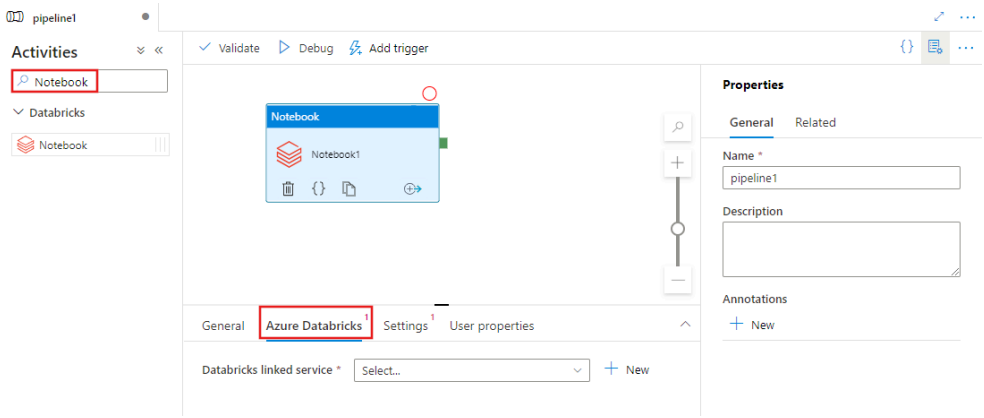
28. How do you parse Nested JSON in Azure Data factory?

Use the flatten transformation to take array values inside hierarchical structures such as JSON and unroll them into individual rows. This process is known as denormalization.
The flatten transformation contains the following configuration setting



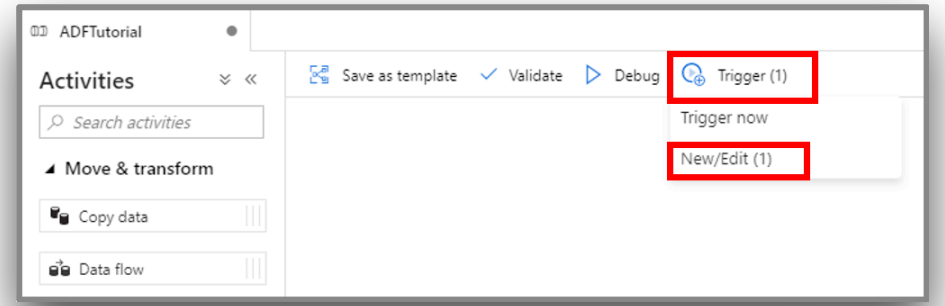
29. Can you write a custom activity / transformation in Data Factory using Python? Pandas? Spark?

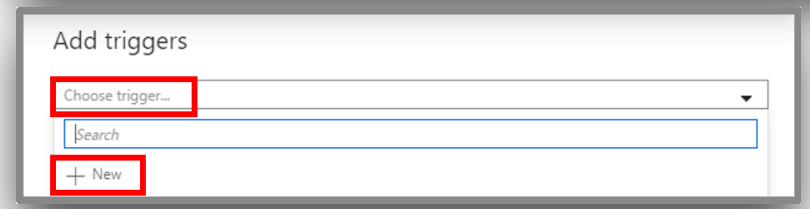
Use Databricks notebook for any kind of customer transformations using SQL, Python, Pyspark or Scala languages.

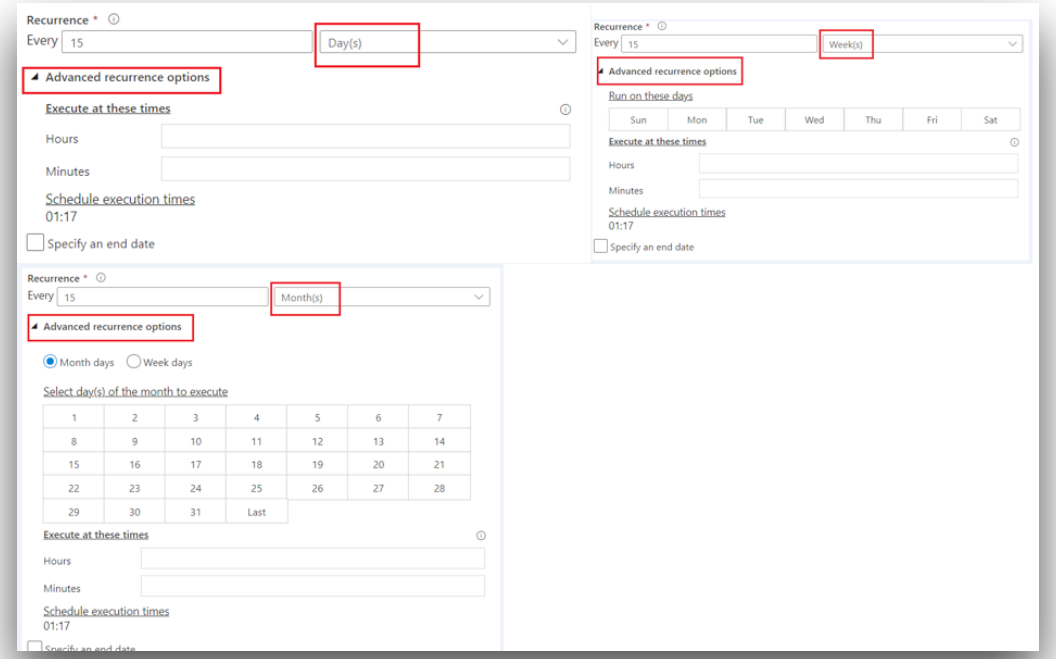


30. How do you schedule Data factory pipelines?

You can create a schedule trigger to schedule a pipeline to run periodically (hourly, daily, etc.).
When creating a schedule trigger, you specify a schedule (start date, recurrence, end date etc.) for the trigger, and associate with a pipeline. Pipelines and triggers have a many-to-many relationship. Multiple triggers can kick off a single pipeline. A single trigger can kick off multiple pipelines.

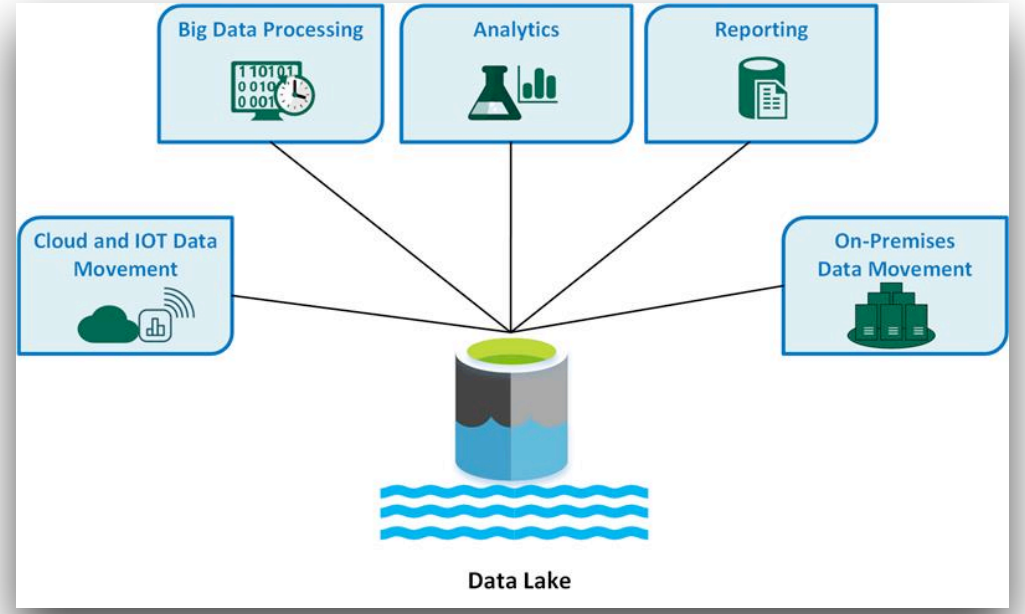






31. What is the big data SaaS Services in Azure?

Azure Data Lake is a big data solution based on multiple cloud services in the Microsoft Azure ecosystem. It allows organizations to ingest multiple data sets, including structured, unstructured, and semi-structured data, into an infinitely scalable data lake enabling storage, processing, and analytics. Learn about the 4 key components of an Azure Data Lake - core infrastructure, ADLS, ADLA, Databricks , Synapse Analytics and HDInsights - and best practices to using them effectively.



32. How do you store files in Azure?

The Azure Storage platform is Microsoft's cloud storage solution for modern data storage scenarios. Azure Storage offers highly available, massively scalable, durable, and secure storage for a variety of data objects in the cloud. Azure Storage data objects are accessible from anywhere in the world over HTTP or HTTPS via a REST API. Azure Storage also offers client libraries for developers building applications or services with .NET, Java, Python, JavaScript, C++, and Go. Developers and IT professionals can use Azure PowerShell and Azure CLI to write scripts for data management or configuration tasks. The Azure portal and Azure Storage Explorer provide user-interface tools for interacting with Azure Storage.

Azure Data Lake Storage Gen2 is a set of capabilities dedicated to big data analytics, built on Azure Blob Storage.

Data Lake Storage Gen2 converges the capabilities of Azure Data Lake Storage Gen1 with Azure Blob Storage. For example, Data Lake Storage Gen2 provides file system semantics, file-level security, and scale. Because these capabilities are built on Blob storage, you'll also get low-cost, tiered storage, with high availability/disaster recovery capabilities.

33. What are event driven data solutions?

An event-driven architecture consists of event producers that generate a stream of events, and event consumers that listen for the events.
An event driven architecture can use a publish/subscribe (also called pub/sub) model or an event stream model.

Pub/sub: The messaging infrastructure keeps track of subscriptions. When an event is published, it sends the event to each subscriber. After an event is received, it cannot be replayed, and new subscribers do not see the event.

Event streaming: Events are written to a log. Events are strictly ordered (within a partition) and durable. Clients don't subscribe to the stream, instead a client can read from any part of the stream. The client is responsible for advancing its position in the stream. That means a client can join at any time, and can replay events.

34. What is Kafka? what Kafka alternates you have in Azure?

Apache Kafka is an open-source distributed streaming platform that can be used to build real-time streaming data pipelines and applications. Kafka also provides message broker functionality similar to a message queue, where you can publish and subscribe to named data streams.

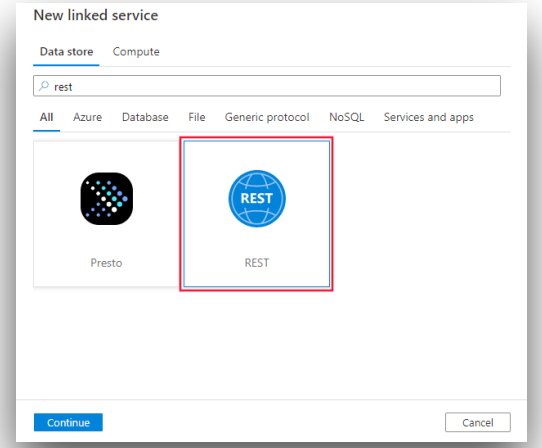
A Kafka cluster can have multiple topics. Similarly, an Event Hub Namespace can have multiple Event Hubs. Azure Event Hub Namespace is a logical container that can hold multiple Even Hub instances. So, both Kafka and Event Hub are similar on this aspect

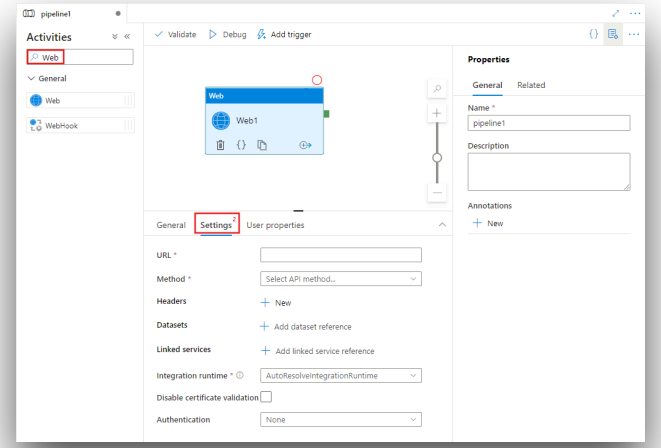
35. How do you call REST services from Data Factory?

Copy and transform data from and to a REST endpoint by using Azure Data Factory.
We can use below two activities.

1) copy activity with dataset type of REST API
2) Web activity with REST API URL - GET or PUT or POST Methods.

Web Activity can be used to call a custom REST endpoint from an Azure Data Factory or Synapse pipeline. You can pass datasets and linked services to be consumed and accessed by the activity.





36. What is batch data processing?

Batch processing is a technique for automating and processing multiple transactions as a single group. Batch processing helps in handling tasks like payroll, end-of-month reconciliation, or settling trades overnight.
Batch data processing is a method of processing data in which data is collected over a period of time and then processed as a group (or "batch") rather than in real-time. This involves storing the data in a file or database until there is enough data to process, and then running a job or script to process the data in bulk. Batch processing is commonly used for large-scale data processing and analysis, such as data warehousing, ETL (extract, transform, load), and report generation.

37. What is real-time data processing?

Real-time data processing is a method of processing data as soon as it is generated or received, without any delay. This involves continuously processing and analyzing incoming data, typically with low latency (i.e., minimal delay), in order to provide immediate insights and take timely actions. Real-time processing is commonly used in applications where timely responses are critical, such as fraud detection, financial trading, and real-time monitoring of systems and devices. Real-time data processing can be achieved through a variety of technologies, such as stream processing, complex event processing, and in-memory databases.

38. What is streaming?

Streaming refers to the continuous flow of data or media content (such as audio, video, or text) from a source to a recipient over a network in real-time, allowing the recipient to access and consume the content as it is being transmitted. Unlike traditional download-based methods, which require the entire content to be downloaded before it can be accessed, streaming allows users to access and use the content as it is being delivered, which provides faster access and reduces storage requirements. Streaming is commonly used for a wide range of applications, such as video and music streaming services, online gaming, live broadcasts, and real-time data processing.

39. What are the different streaming services available in Azure?

Azure provides several streaming services that enable real-time data processing and delivery. Some of the key streaming services available in Azure are:

Azure Stream Analytics: a fully managed real-time analytics service that can process streaming data from various sources and deliver insights in real-time.

Azure Event Hubs: a highly scalable data streaming platform that can collect and process millions of events per second from various sources.

Azure IoT Hub: a cloud-based platform that can connect, monitor, and manage IoT devices and process real-time device telemetry data.

Azure Notification Hubs: a service that enables push notifications to be sent to mobile and web applications at scale.

Azure Media Services: a platform that provides cloud-based media processing and delivery services for streaming video and audio content.

Azure Data Explorer: a fast and highly scalable data exploration and analytics service that can process and analyze large volumes of streaming data in real-time.

40. What is Spark SQL?

Spark SQL is a component of the Apache Spark open-source big data processing framework that enables developers to run SQL-like queries on large datasets stored in distributed storage systems like Hadoop Distributed File System (HDFS) and Apache Cassandra. It provides a programming interface to work with structured data using SQL queries, DataFrame APIs, and Datasets APIs.

Spark SQL allows users to combine the benefits of both relational and procedural programming paradigms to work with data in a distributed environment. It also provides support for reading and writing data in various file formats such as Parquet, ORC, Avro, JSON, and CSV.

Spark SQL includes an optimizer that can optimize SQL queries to improve query performance by pushing down filters, aggregations, and other operations to the data source. This optimization enables Spark SQL to process large datasets efficiently in a distributed environment.

41. How do you covert pandas dataframe to spark data frame vice versa?

Creating Pandas DataFrame and converting into Spark DataFrame.

```
# Create a sample Pandas DataFrame
pandas_df = pd.DataFrame({
    'id': [1, 2, 3],
    'name': ['John', 'Alice', 'Bob'],
    'age': [25, 30, 35]
})

# Convert Pandas DataFrame to Spark DataFrame
spark_df = spark.createDataFrame(pandas_df)

# Show Spark DataFrame
spark_df.show()
```

Creating spark DataFrame and converting into pandas DataFrame.

```
# Create a sample Spark DataFrame
spark_df = spark.createDataFrame([(1, "John", 25), (2, "Alice", 30), (3, "Bob", 35)], ["id", "name", "age"])

# Convert Spark DataFrame to Pandas DataFrame
pandas_df = spark_df.toPandas()

# Show Pandas DataFrame
print(pandas_df)
```

42. How do you slice a data file into 10 small files (a large CSV file with 10 million lines, slice them 1 million each file) using Pandas?

```
import pandas as pd

# Load the large CSV file into a Pandas DataFrame
df = pd.read_csv('large_file.csv')

# Calculate the number of rows in each file
rows_per_file = 1000000
num_files = int(len(df) / rows_per_file)

# Slice the data file into smaller files
for i in range(num_files):
    start_index = i * rows_per_file
    end_index = start_index + rows_per_file
    file_name = f'small_file_{i+1}.csv'
    df[start_index:end_index].to_csv(file_name, index=False)
```

43. How do you slice a data file into 10 small files (a large CSV file with 10 million lines, slice them 1 million each file) using Spark?

```
# Load the large CSV file into a Spark DataFrame
df = spark.read.csv('/tmp/amazon_reviews_us_Electronics_v1_00.tsv',header=True,sep="\t")

# We can control the number of records per file while writing a dataframe using property maxRecordsPerFile.
# use coalesce(1) for creating no of files based on maxRecrodsPerFile option.
df.coalesce(1).write.format('csv').option("maxRecordsPerFile","1000000").mode('overwrite').save("/output/",header=True)
```

```
1  from pyspark.sql.functions import *
2  df1 = spark.read.csv("dbfs:/output/",header=True).withColumn("file",input_file_name())
3  df1.groupBy("file").count().show(truncate=False)
```

▸ (2) Spark Jobs

```
+-----------------------------------------------------------------------------------------------------+-------+
|file                                                                                                 |count  |
+-----------------------------------------------------------------------------------------------------+-------+
|dbfs:/output/part-00000-tid-8702932682927122524-5804c810-7708-4991-af6b-49dd9fe03476-64-1-c000.csv|1000000|
|dbfs:/output/part-00000-tid-8702932682927122524-5804c810-7708-4991-af6b-49dd9fe03476-64-2-c001.csv|1000000|
|dbfs:/output/part-00000-tid-8702932682927122524-5804c810-7708-4991-af6b-49dd9fe03476-64-3-c002.csv|1000000|
|dbfs:/output/part-00000-tid-8702932682927122524-5804c810-7708-4991-af6b-49dd9fe03476-64-4-c003.csv|93869  |
+-----------------------------------------------------------------------------------------------------+-------+
```

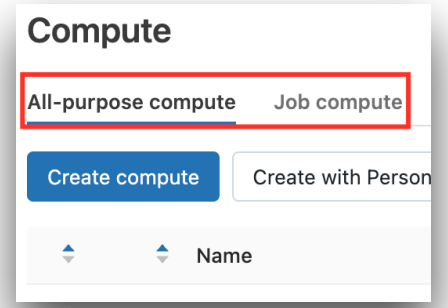
44. What is databricks? why is it required? what databricks runtimes you used?

Databricks Runtime is a managed computing environment for Apache Spark, which is an open-source distributed computing framework for big data processing. Databricks Runtime is optimized for running Spark-based workloads in a cloud-based environment and includes pre-configured clusters, drivers, and tools that make it easy to set up and manage Spark applications. It provides a unified platform for data engineers, data scientists, and business analysts to collaborate on big data processing and analytics tasks. Databricks Runtime is part of the Databricks Unified Analytics Platform, which also includes data integration, machine learning, and visualization tools.

Databricks provides a Docker image for the Databricks Runtime environment. The image can be used to run Databricks workloads locally or in a containerized environment. The image can be obtained from Docker Hub or built from source using the Databricks open-source repository on GitHub.

45. What is cluster in databricks? what are different cluster types available in databricks?

A Databricks cluster is a managed computing environment that allows users to run distributed data processing workloads on the Databricks platform. It is a group of virtual machines that are provisioned and configured to work together to execute distributed data processing tasks, such as data ingestion, transformation, machine learning, and deep learning. The cluster resources, such as the number and type of virtual machines, can be adjusted based on the workload requirements, and users can choose from various cluster configurations to optimize performance and cost. Databricks clusters are typically used to process large volumes of data and to train and deploy machine learning models at scale.



Databricks offers two types of Compute specifically designed for running batch workloads: All-purpose compute and Job compute.

All-purpose clusters: These clusters are designed to run long-running batch jobs and are optimized for high availability, fault tolerance, and resource isolation. They can be created and managed using the Databricks UI or API and can be used to run data processing jobs, ETL pipelines, and machine learning workflows.

Job clusters: Job clusters are a type of ephemeral cluster that are created on-demand to run a specific job and are terminated automatically once the job is complete. Job clusters are optimized for cost and performance, as they are created with the minimum required resources to run the job. They are typically used for running ad-hoc or one-time batch jobs, such as data transformations or model training.

Above Two clusters can support different powers.

Standard clusters: These are the most common type of cluster and are used for general-purpose data processing and analytics workloads. Standard clusters are highly customizable and can be configured with various virtual machine types, network settings, and storage options.

High Concurrency clusters: These clusters are optimized for running interactive workloads and serving multiple users concurrently. They are designed to handle small to medium-sized queries and are highly scalable, allowing users to increase or decrease the cluster size based on demand.

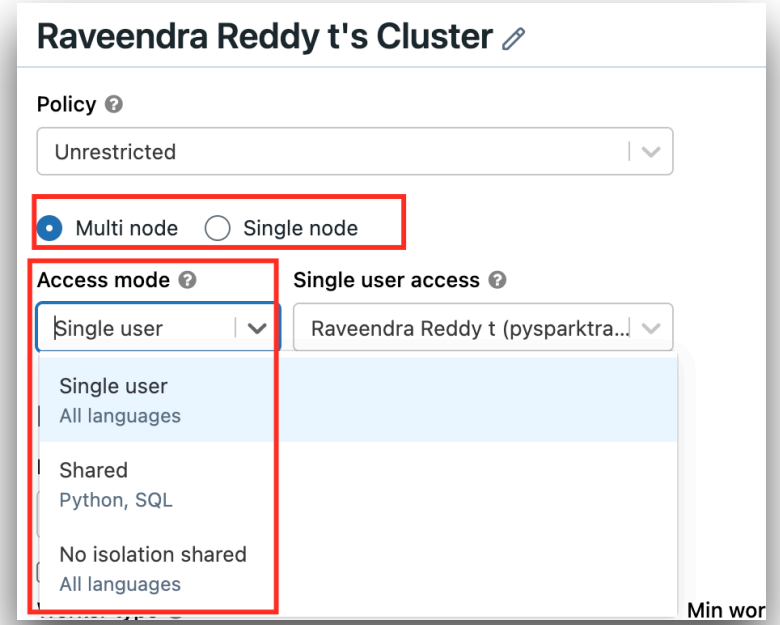
GPU clusters: These clusters are used for running deep learning workloads and training machine learning models that require high-performance GPUs. GPU clusters can be configured with different types of GPUs, such as NVIDIA V100, P100, and K80, and are optimized for running TensorFlow, PyTorch, and other deep learning frameworks.

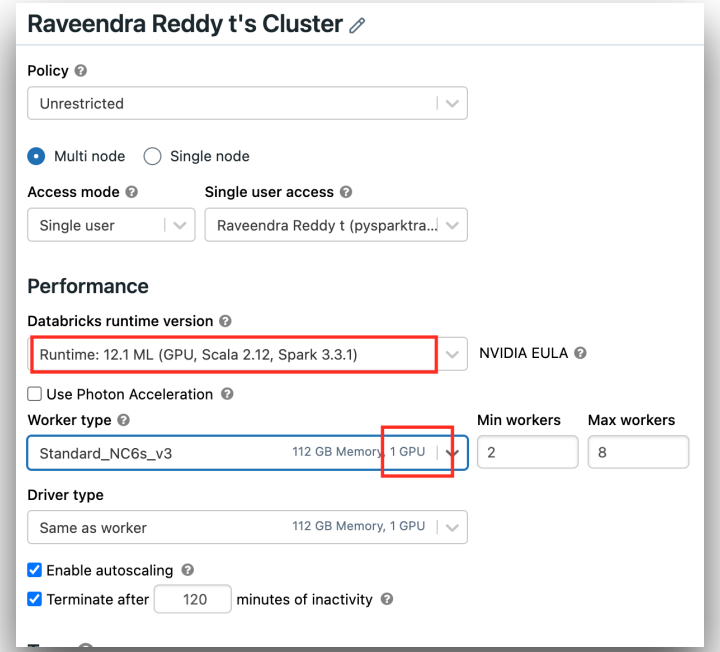
Serverless clusters: These clusters are designed to provide a highly scalable and cost-effective computing environment for ad-hoc workloads and bursty data processing. Serverless clusters automatically scale up or down based on the workload demand and are charged based on the actual usage.

Kubernetes clusters: These clusters allow users to run Databricks workloads on a Kubernetes cluster, giving them more control over the cluster environment and enabling them to leverage Kubernetes features such as auto-scaling, load balancing, and rolling updates.

Standard mode clusters are now called **No Isolation Shared access mode clusters**.

High Concurrency with Tables ACLs are **now called Shared access mode clusters**.





46. How do you connect (Mount) Data lake storage with Databricks?

Configure authentication for mounting

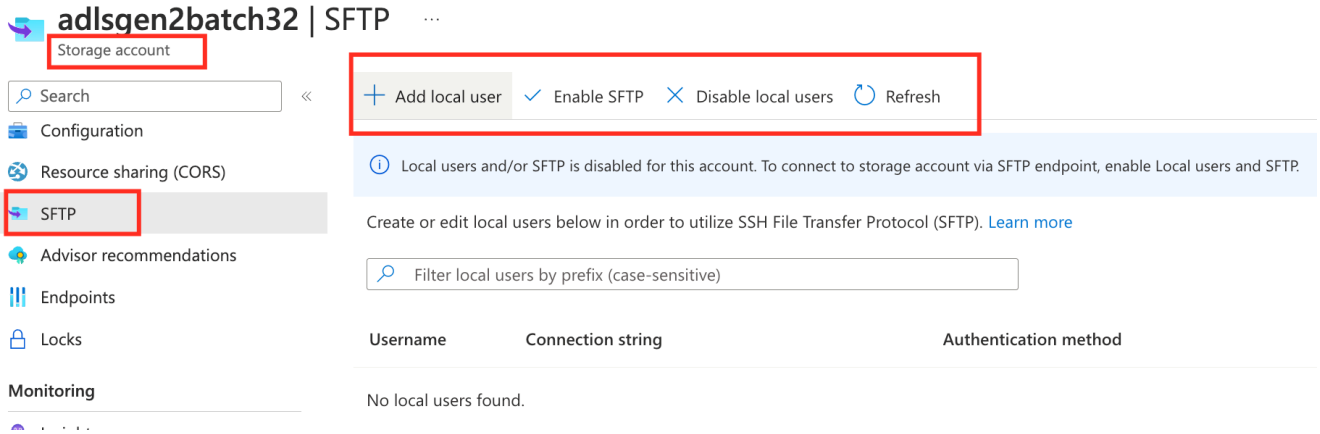
```
configs = {"fs.azure.account.auth.type": "OAuth",
"fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
"fs.azure.account.oauth2.client.id": <client-id>,
"fs.azure.account.oauth2.client.secret": <service-credential>,
"fs.azure.account.oauth2.client.endpoint": "https://login.microsoftonline.com/<directory-id>/oauth2/token"}
```

Cmd 8

Mount filesystem

```
dbutils.fs.mount(
  source = "abfss://<file_system>@<storage-account-name>.dfs.core.windows.net/",
  mount_point = "/mnt/mymount",
  extra_configs = configs)
```

47. How do you enable SFTP service into your Azure data lake storage?



48. How do you securely mount your data from ADLS to Databricks?

Configure authentication

```
service_credential = dbutils.secrets.get(scope="<scope>",key="<service-credential-key>")

spark.conf.set("fs.azure.account.auth.type.<storage-account>.dfs.core.windows.net", "OAuth")
spark.conf.set("fs.azure.account.oauth.provider.type.<storage-account>.dfs.core.windows.net", "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider")
spark.conf.set("fs.azure.account.oauth2.client.id.<storage-account>.dfs.core.windows.net", "<application-id>")
spark.conf.set("fs.azure.account.oauth2.client.secret.<storage-account>.dfs.core.windows.net", service_credential)
spark.conf.set("fs.azure.account.oauth2.client.endpoint.<storage-account>.dfs.core.windows.net", "https://login.microsoftonline.com/<directory-id>/oauth2/token")
```

Read Databricks Dataset IoT Devices JSON

```
df = spark.read.json("/databricks-datasets/iot/iot_devices.json")
```

Write Delta table to external path

```
df.write.save("abfss://<container-name>@<storage-account-name>.dfs.core.windows.net/<path-to-data>")
```

List filesystem

```
dbutils.fs.ls("abfss://<container-name>@<storage-account-name>.dfs.core.windows.net/<path-to-data>")
```

Read IoT Devices JSON from ADLS Gen2 filesystem

```
df2 = spark.read.load("abfss://<container-name>@<storage-account-name>.dfs.core.windows.net/<path-to-data>")
display(df2)
```

49. How do you read / write data from ADLS to databricks?

You can read and write data from Azure Data Lake Storage (ADLS) to Databricks using the following steps:

Mount ADLS Gen1 or Gen2 to Databricks: To mount ADLS to Databricks, you can use the Databricks UI or API. You will need to provide the ADLS account name and key or an Azure Active Directory token, along with the mount point and configuration options. This will create a virtual filesystem on Databricks that points to the ADLS storage account.

Read data from ADLS: Once the ADLS account is mounted, you can read data from it using the file APIs in Databricks, such as spark.read or dbutils.fs. For example, you can read a CSV file from ADLS using the following code:

```
df = spark.read.format("csv").option("header", "true").load("/mnt/adls/path/to/file.csv")
```

Write data to ADLS: To write data to ADLS, you can use the same file APIs, but specify the ADLS mount point as the output directory. For example, to write a DataFrame to a Parquet file on ADLS, you can use the following code:

```
df.write.format("parquet").save("/mnt/adls/path/to/output")
```

50. What are the different ways to schedule a data engineering job in Azure?

There are several ways to schedule a data engineering job in Azure, depending on the requirements and use case:

Azure Data Factory: Azure Data Factory (ADF) is a cloud-based data integration service that allows you to create, schedule, and orchestrate data pipelines. ADF provides a visual interface for building pipelines using drag-and-drop components, as well as code-based pipeline definition using Azure Resource Manager (ARM) templates. ADF also supports various data sources and destinations, including cloud and on-premises databases, files, and big data stores.

Azure Logic Apps: Azure Logic Apps is a cloud-based service that allows you to create and schedule workflows that integrate with various systems and services. Logic Apps provides a visual workflow designer that allows you to create workflows using pre-built connectors and custom code. Logic Apps can integrate with Azure services, as well as third-party services, such as Salesforce, Slack, and Twilio.

Azure Functions: Azure Functions is a serverless compute service that allows you to run event-driven code in response to various triggers. Functions can be used to run data processing or data integration code on a schedule or in response to an event, such as a file upload or a message in a queue. Functions can be written in several languages, including C#, Python, and JavaScript.

Azure Batch: Azure Batch is a cloud-based job scheduling and compute management service that allows you to run large-scale parallel and high-performance computing (HPC) workloads. Batch provides a managed environment for running jobs on clusters of virtual machines, with support for job scheduling, job dependencies, and scaling. Batch can be used to run data processing or machine learning workloads on a large scale.

Azure Kubernetes Service: Azure Kubernetes Service (AKS) is a managed Kubernetes service that allows you to deploy and manage containerized applications and services. AKS provides a scalable and highly available environment for running batch jobs or data processing workloads using containerized applications. AKS can be integrated with Azure services, such as Azure Container Registry, for a seamless end-to-end experience.

azure vm cron job: In Azure, you can schedule a cron job on a virtual machine (VM) using the built-in Linux cron service.

Here are the steps to create a cron job on an Azure VM:

SSH into your VM: Open a terminal and SSH into your VM using the ssh command and your VM's public IP address or DNS name.

Open the cron configuration file: Once you're connected to the VM, open the cron configuration file using the following command:

```
30 3 * * * /path/to/script.sh
```

```
systemctl status cron
```

```
sudo systemctl start cron
```

Databricks Workflow jobs: are used to automate and schedule data processing and analysis tasks. Here are some key features of Databricks jobs:

Scheduling: You can create jobs to run on a schedule, such as daily, weekly, or monthly. You can also specify the start time and end time for the job and the time zone in which it should run.

Recurrence: You can set the job to recur at a specific interval, such as every 15 minutes, every hour, or every day.

51. Explain some interesting Spark SQL functions you worked with?

coalesce
coalesce(expr1, expr2, ...) - Returns the first non-null argument if exists. Otherwise, null.
nvl
nvl(expr1, expr2) - Returns expr2 if expr1 is null, or expr1 otherwise
SELECT nvl(NULL, array('2'));
to_date
to_date(date_str[, fmt]) - Parses the date_str expression with the fmt expression to a date. Returns null with invalid input. By default, it follows casting rules to a date if the fmt is omitted.

SELECT to_date('2016-12-31', 'yyyy-MM-dd');

months_between
months_between(timestamp1, timestamp2[, roundOff]) - If timestamp1 is later than timestamp2, then the result is positive. If timestamp1 and timestamp2 are on the same day of month, or both are the last day of month, time of day will be ignored. Otherwise, the difference is calculated based on 31 days per month, and rounded to 8 digits unless roundOff=false.
SELECT months_between('1997-02-28 10:30:00', '1996-10-30');
trim
trim(str) - Removes the leading and trailing space characters from str.

52. How do read filename while reading data from ADLS into databricks?

To read the filename while reading data from ADLS into Databricks, you can use the **input_file_name()** function in PySpark. Here's an example code snippet:

```
from pyspark.sql.functions import input_file_name
df_airlines = spark.read.csv("/databricks-datasets/asa/airlines",header=True)
df_airlines = df_airlines.withColumn("FILE_NAME",input_file_name())
```

53. What is delta lake in databricks?

Delta Lake is an open-source storage layer that brings reliability to data lakes. Delta Lake provides ACID transactions, scalable metadata handling, and unifies streaming and batch data processing. It uses a transaction log to keep track of all the operations performed on the data, which provides features like ACID compliance, versioning, and data lineage.

A Delta table is a table that is stored in Delta Lake format. Delta tables can be queried using standard SQL commands, and can be accessed by a variety of different data processing frameworks and tools, including Apache Spark, Python, R, SQL, and machine learning frameworks like TensorFlow and Scikit-Learn. Delta tables can be used for both batch and streaming workloads, and support a variety of data sources and file formats, including Parquet, CSV, JSON, and Avro. With Delta Lake, users can easily build data pipelines, perform data engineering tasks, and build analytics applications with high reliability and performance.

54. How do you read schema from a particular file using Spark / Pandas?

To read the schema from a file using Spark, you can use the printSchema() method of a DataFrame object. This method prints the schema of the DataFrame in a tree format, showing the data types and structure of each column. Here's an example using PySpark:

```
# Read data from a file and create a DataFrame object
df = spark.read.format("csv").option("header", "true").load("path/to/file.csv")
# Print the schema of the DataFrame
df.printSchema()

import pandas as pd
# Read the file into a DataFrame with all columns as strings
df = pd.read_csv("path/to/file.csv", dtype=str)
# Print the schema of the DataFrame
print(df.dtypes)
```

55. What are the advantages of Delta lake?

Delta Lake provides several advantages over traditional data storage solutions for big data analytics. Here are some of the key advantages of using Delta Lake:

ACID transactions: Delta Lake provides ACID transactions, which ensure that data is processed reliably and without conflicts. This helps to eliminate common data integrity issues that can arise in distributed environments.

Data versioning: Delta Lake keeps track of every change that is made to the data, allowing users to revert to earlier versions of the data if necessary. This makes it easy to recover from data corruption or accidental data loss.

Schema enforcement: Delta Lake enforces schema on write, which ensures that all data written to the data lake conforms to a consistent schema. This helps to eliminate data quality issues and makes it easier to manage and analyze data.
Query optimization: Delta Lake provides features like data skipping, predicate pushdown, and Z-ordering that improve query performance by reducing the amount of data that needs to be scanned.
Stream processing: Delta Lake supports both batch and streaming workloads, allowing users to process data in real-time and perform near-real-time analysis.

Open source: Delta Lake is an open-source project that is supported by a large and active community. This makes it easy to get help and support, and to contribute to the development of the technology.

Overall, Delta Lake provides a reliable, scalable, and performant data storage solution for big data analytics, making it an attractive choice for many organizations.

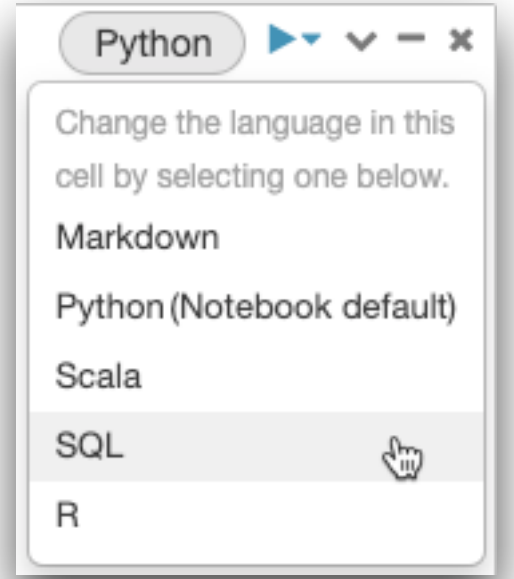
56. How do you list all databases and all tables from databricks?

```
databases = [db.databaseName for db in spark.sql('show databases').collect()]
tables = [f"{row['database']}.{row['tableName']}"
    for db_rows in [
        spark.sql(f'show tables in {db}').collect() for db in databases
    ]
    for row in db_rows]
print(tables)

['test1.test1', 'test2.test1']

Command took 0.77 seconds -- by pysparktraining32@gmail.com at 14/02/2023, 17:48:53 on My Cluster
```

57. What are magic commands in databricks?

you can use the language magic command %<language> at the beginning of a cell. The supported magic commands are: %python, %r, %scala, and %sql.

- ☑ %fs: Access the Databricks File System (DBFS) and interact with files.
- ☑ %sh: Run shell commands in a notebook cell.
- ☑ %md: Write markdown in a notebook cell.
- ☑ %sql: Execute SQL queries against tables in a database.
- ☑ %python: Switch to Python language mode in a notebook cell.
- ☑ %scala: Switch to Scala language mode in a notebook cell.
- ☑ %r: Switch to R language mode in a notebook cell.
- ☑ %run: Run a notebook, passing arguments if necessary.
- ☑ %pip: Install Python packages using pip.
- ☑ %conda: Install Python packages using conda.
- ☑ %load: Load external code into a notebook cell.
- ☑ %lsmagic: List all available magic commands.




```
%alias  %alias_magic  %autoawait  %autocall  %automagic  %autosave  %bookmark  %cat  %cd  %clear  %colors  %conda  %config  %connect_info  %cp  %debug  %dhist  %dirs  %docte
st_mode  %ed  %edit  %env  %gui  %hist  %history  %killbgscripts  %ldir  %less  %lf  %lk  %ll  %load  %load_ext  %loadpy  %logoff  %logon  %logstart  %logstate  %logstop  %l
s  %lsmagic  %lx  %macro  %magic  %man  %matplotlib  %mkdir  %more  %mv  %notebook  %page  %pastebin  %pdb  %pdef  %pdoc  %pfile  %pinfo  %pinfo2  %pip  %popd  %pprint  %pre
cision  %prun  %psearch  %psource  %pushd  %pwd  %pycat  %pylab  %qtconsole  %quickref  %recall  %rehashx  %reload_ext  %rep  %rerun  %reset  %reset_selective  %rm  %rmdir
%run  %save  %sc  %set_env  %store  %sx  %system  %tb  %time  %timeit  %unalias  %unload_ext  %who  %who_ls  %whos  %xdel  %xmode
```

58. How do you connect databricks with a SQL datastore?

Azure Databricks supports connecting to external databases using JDBC. This article provides the basic syntax for configuring and using these connections with examples in Python, SQL, and Scala.

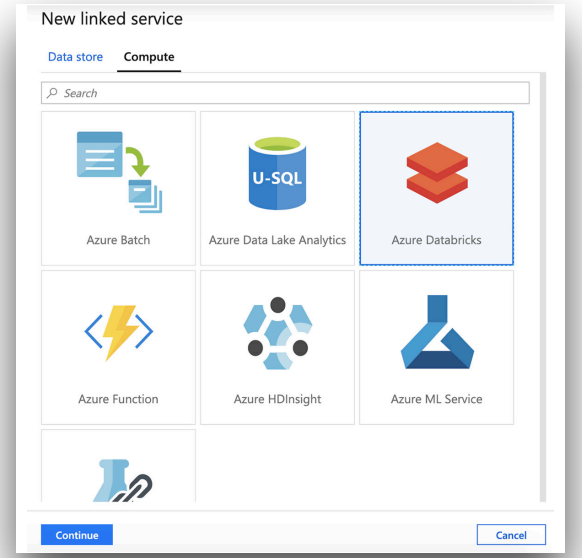
Read data with JDBC

You must configure a number of settings to read data using JDBC. Note that each database uses a different format for the <jdbc_url>.

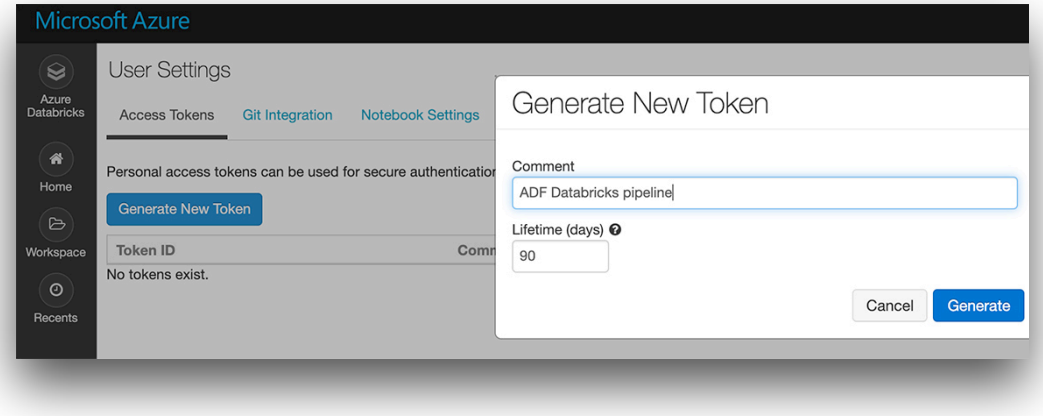
```
#Database Username
username="db_user"
password ="mypassword"
#Database Password - getting from Databricks Scope and Azure key vault
#password =dbutils.secrets.get(scope="sqldb-scope",key="db_password_key")
#Database Server URL
dbserver="db-server.database.windows.net"
#Database Name
dbname="mydatabase"
url="jdbc:sqlserver://"+dbserver+":1433;database="+dbname+";user="+username+";password="+password
df = spark.read.format("jdbc").option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver").option("url", url).jdbc(url,"mytable")
```

59. How do connect Azure data factory with azure databricks securely?

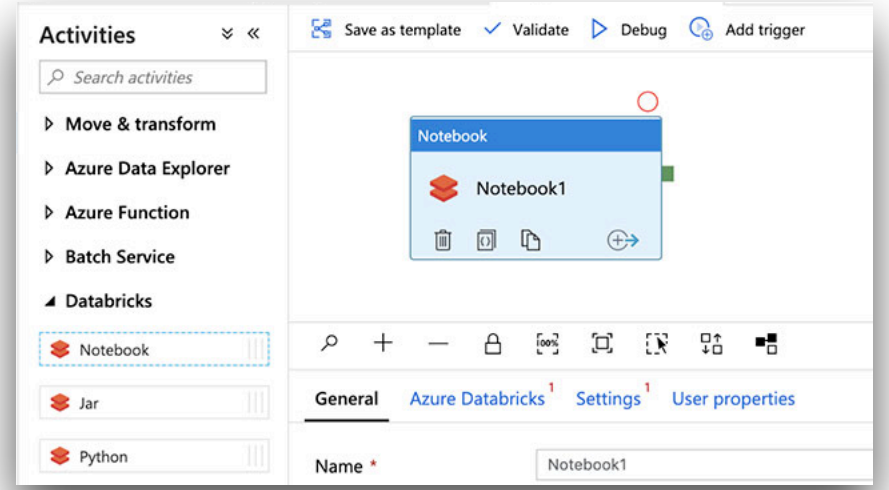
Create an Azure Databricks linked service in ADF to connect and integrate Databricks in data factory pipelines.



Generate ADB Token in User Settings. Use token based authentication ADF Linked Service.

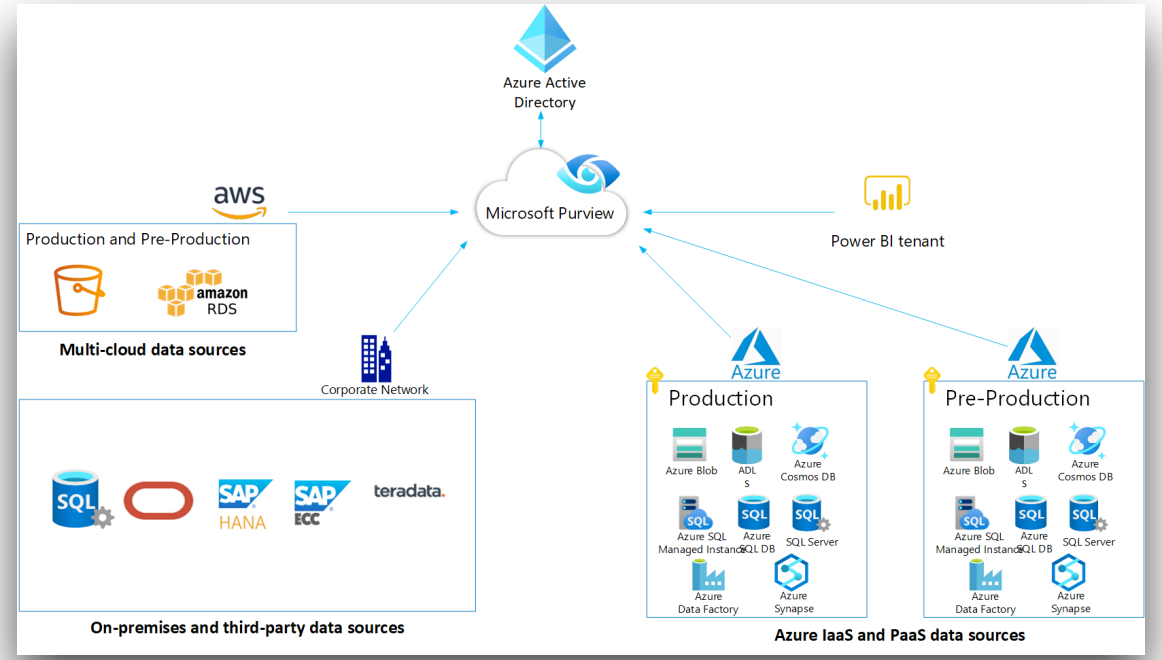


Create new pipeline and use databricks notebook activity in pipeline activities.
add a Databricks notebook to the pipeline by expanding the "Databricks" activity, then dragging and dropping a Databricks notebook onto the pipeline design canvas.



60. What is Azure purview?

Microsoft Purview provides a unified data governance solution to help manage and govern your on-premises, multicloud, and software as a service (SaaS) data. Easily create a holistic, up-to-date map of your data landscape with automated data discovery, sensitive data classification, and end-to-end data lineage. Enable data consumers to access valuable, trustworthy data management.



Azure Purview is a cloud-based data governance solution from Microsoft that helps organizations discover, manage, and secure their data assets across on-premises, multi-cloud, and SaaS environments. It provides a unified view of an organization's data landscape, enabling users to understand their data assets and their relationships, and to discover new data sources and insights.

Azure Purview is designed to help organizations address the challenges of data discovery, cataloging, and governance. It includes features such as automated data discovery and classification, data lineage and impact analysis, metadata management, data cataloging, and policy enforcement. These features help organizations ensure the accuracy, completeness, consistency, and security of their data assets throughout their lifecycle.

Azure Purview also provides integration with other Microsoft cloud services, such as Azure Data Factory, Azure Databricks, and Azure Synapse Analytics, as well as with third-party services, enabling organizations to manage and govern their data assets across a wide range of environments and tools.

Overall, Azure Purview is a comprehensive solution for data governance and management that helps organizations gain greater visibility and control over their data assets, while also improving compliance and reducing risks associated with data management.

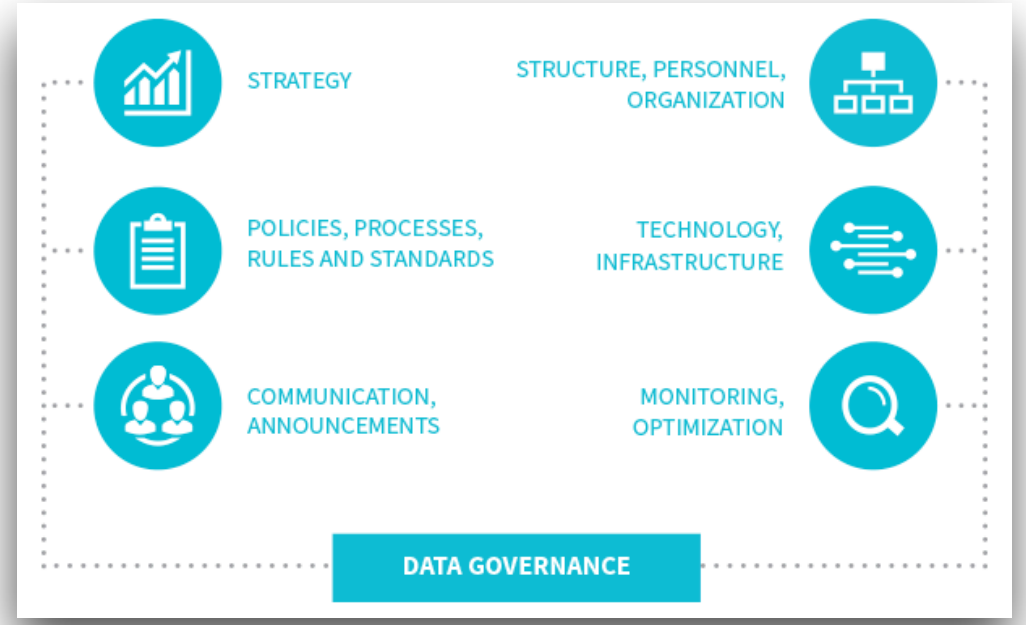
61. What is data governance?

Data governance is a set of policies, procedures, and practices that organizations use to manage their data assets. It encompasses the processes and controls that ensure the accuracy, completeness, consistency, and security of an organization's data throughout its lifecycle.

Data governance is concerned with ensuring that the right people have access to the right data at the right time, and that data is used in a responsible and ethical way. It also involves managing data quality, metadata, data standards, and data security.

The goals of data governance include improving the accuracy and consistency of data, ensuring compliance with regulations and standards, reducing risks associated with data management, and maximizing the value of an organization's data assets.

Data governance is typically managed by a dedicated team or department within an organization, which is responsible for defining policies and procedures, monitoring compliance, and enforcing standards. This team works closely with other departments, such as IT, legal, and business operations, to ensure that data governance is integrated throughout the organization.

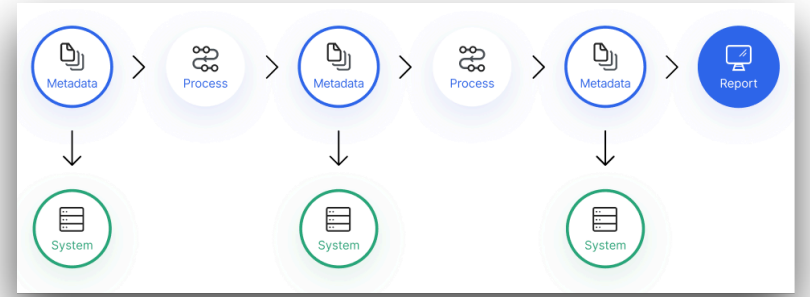


62. What is data lineage?

Data lineage refers to the journey or path that data takes from its source through various transformations and processes, to its final destination, such as a report or dashboard. It is a way of tracing the flow of data through a system, and it helps to understand how data is used and manipulated within an organization.

Data lineage typically includes information about the data's origins, its processing and transformations, and any other data that it may be related to or depend on. It can also include metadata about the data, such as its format, structure, and quality.

Data lineage is important for a number of reasons. It helps to ensure data accuracy and consistency, by identifying any potential sources of errors or discrepancies in the data. It also helps to meet regulatory and compliance requirements, by providing a clear audit trail of data usage and handling. Additionally, it helps to improve data governance and management, by providing a better understanding of how data is used within an organization, and who has access to it.



63. How do you connect Azure databricks with BI tools?

Databricks has validated integrations with your favorite BI tools, including Power BI, Tableau, and others, allowing you to work with data through Databricks clusters and SQL warehouses, in many cases with low-code and no-code experiences.

Refer below link for list of BI Tools integration with Databricks.
https://docs.databricks.com/partners/partners.html#bi

BI and visualization

Partner	Unity Catalog	Steps to connect
HEX		Connect to Hex manually
Looker		Connect to Looker
MicroStrategy		Connect to MicroStrategy
MODE		Connect to Mode
Power BI	Yes	Connect to Power BI Desktop manually
preset		Connect to Preset manually
Qlik		Connect to Qlik Sense manually
sigma	Yes	Connect to Sigma manually
+ableau	Yes	Connect to Tableau Desktop manually
ThoughtSpot		Connect to ThoughtSpot manually
TIBCO		Connect to TIBCO Spotfire Analyst

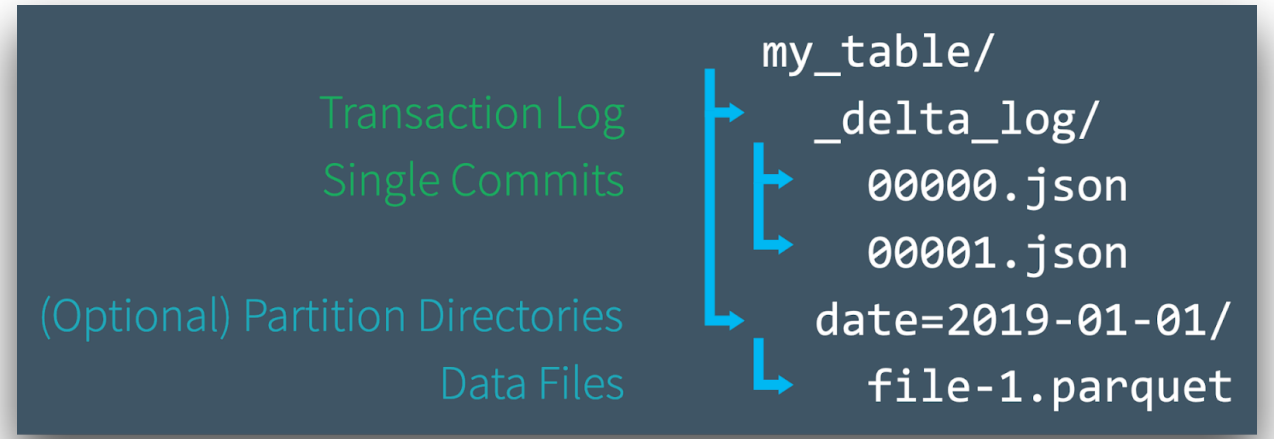
64. How do you store REST API data in databricks delta lake?

You can use Python to read REST API data and store in JSON format.
Then Read JSON format and create DataFrame and create target table as delta table using Dataframe Write API.

```python
import requests
# API RUL
api_url = "https://api.github.com"
# Call api using python request library
response = requests.get(api_url)
# Read Response data in JSON and Create RDD and Create DATAFRAME.
df = spark.read.json(sc.parallelize([response.json()]))
# Write Data into Delta format using df.write.format("delta")
#display(df)
df.write.format("delta").mode("append").saveAsTable("api_delta_table")
```

65. What is the default file type for Databricks delta lake?

Delta Lake uses versioned Parquet files to store your data in your cloud storage.
Apart from the versions, Delta Lake also stores a transaction log to keep track of all the commits made to the table or blob store directory to provide ACID transactions.



66. Can you use databricks delta as operation store? for example for an ordering system? or real time booking system?

Databricks Delta is primarily designed for OLAP (Online Analytical Processing) workloads, which involve analyzing large datasets to gain insights and make informed business decisions. OLAP workloads typically involve complex queries that aggregate and summarize data, and require fast access to large volumes of data.

While Delta can be used for OLTP (Online Transaction Processing) workloads as well, it is not optimized for this type of workload. OLTP workloads involve managing high volumes of small transactions, typically involving the insertion, deletion, and updating of individual records. These workloads require high throughput, low latency, and high concurrency, and typically involve relatively small datasets.

While Delta does provide transactional capabilities, including ACID transactions and data versioning, its design and performance characteristics are better suited for OLAP workloads, which involve larger datasets and more complex queries.

That being said, Delta can be used in combination with other tools and platforms to support OLTP workloads. For example, you could use Delta to store and manage historical data, and use a separate database or data store to handle real-time transactional processing.

Overall, Databricks Delta is a powerful and flexible platform for managing and processing large datasets, and can be a great choice for OLAP workloads that require fast access to large volumes of data.

67. How do you work various date formats in databricks?

There are several common scenarios for datetime usage in Spark:

CSV/JSON datasources use the pattern string for parsing and formatting datetime content.
Datetime functions related to convert StringType to/from DateType or TimestampType. For example, **unix_timestamp**, **date_format**, **to_unix_timestamp**, **from_unixtime**, **to_date**, **to_timestamp**, **from_utc_timestamp**, **to_utc_timestamp**,

68. How Does the Transaction log work in databricks delta?

Breaking Down Transactions Into Atomic Commits
Whenever a user performs an operation to modify a table (such as an INSERT, UPDATE or DELETE), Delta Lake breaks that operation down into a series of discrete steps composed of one or more of the actions below.

Add file - adds a data file.
Remove file - removes a data file.
Update metadata - Updates the table's metadata (e.g., changing the table's name, schema or partitioning).
Set transaction - Records that a structured streaming job has committed a micro-batch with the given ID.
Change protocol - enables new features by switching the Delta Lake transaction log to the newest software protocol.
Commit info - Contains information around the commit, which operation was made, from where and at what time.

69. What is best to use Python / Scala / SQL within databricks?

The choice of programming language to use in Databricks depends on a variety of factors, including the nature of the data and the processing that you need to perform, as well as your personal preferences and expertise.
In general, Databricks supports several programming languages, including Python, Scala, SQL, R, and Java. Each language has its own strengths and weaknesses, and may be better suited to certain types of tasks.

Here are a few factors to consider when deciding which language to use in Databricks:
Data types and processing: Python is a popular choice for data analysis and machine learning tasks, as it has a large number of libraries and tools for these tasks, including NumPy, Pandas, and Scikit-learn. Scala, on the other hand, is a good choice for tasks that require high-performance data processing, such as streaming or distributed computing. SQL is best for tasks that require querying and processing data stored in databases or data warehouses.

Integration with Spark: Scala is a native language for Apache Spark, and is often used for developing Spark applications. Python, on the other hand, has a Spark API that allows you to write Spark applications using Python. SQL is used to express relational queries in Spark SQL, which can be used to process large-scale data sets.

Team skills and preferences: The choice of language may also depend on the skills and preferences of your team. If your team has more experience with Python, it may be more efficient to use Python. If your team has more experience with SQL, it may be more efficient to use SQL.

Overall, the best language to use in Databricks depends on your specific use case and the nature of the data and processing that you need to perform. It's often a good idea to experiment with different languages to see which one works best for your needs. Databricks provides an environment that supports multiple languages and makes it easy to switch between them, so you can choose the language that is most appropriate for each task.

70. What are various data storage / processing services in Azure?

Azure offers a variety of data storage and processing services, including:

Azure Blob Storage: A massively scalable object storage service that can store large amounts of unstructured data such as text, images, and videos.

Azure Data Lake Storage: A highly scalable and secure data lake service that allows you to store and analyze large amounts of data.

Azure SQL Database: A fully managed relational database service that offers high availability, security, and scalability.

Azure Cosmos DB: A globally distributed, multi-model database service that supports NoSQL data models, including key-value, graph, and document.

Azure HDInsight: A fully managed cloud service that makes it easy to process big data using popular open-source frameworks such as Hadoop, Spark, Hive, and HBase.
Azure Stream Analytics: A real-time data processing service that allows you to analyze and gain insights from streaming data.
Azure Databricks: A collaborative, cloud-based platform for data engineering, machine learning, and analytics that is based on Apache Spark.
Azure Synapse Analytics: An analytics service that allows you to analyze large amounts of structured and unstructured data using both serverless and provisioned resources.
Azure Machine Learning: A cloud-based machine learning service that allows you to build, train, and deploy machine learning models.

Azure Cognitive Search: A fully managed search-as-a-service that allows you to add search capabilities to your applications using natural language processing and machine learning.

71. What is Azure Cosmos DB?

Cosmos Database (DB) is a globally distributed, low latency, multi-model database for managing data at large scales. It is a cloud-based NoSQL database offered as a PaaS (Platform as a Service) from Microsoft Azure. It is a highly available, high throughput, reliable database and is often called a serverless database. Cosmos database contains the Azure Document DB and is available everywhere.
The key features of Cosmos DB are:

Globally Distributed: With Azure regions spread out globally, the data can be replicated globally.
Scalability: Cosmos DB is horizontally scalable to support hundreds of millions of reads and writes per second.
Schema-Agnostic Indexing: This enables the automatic indexing of data without schema and index management.
Multi-Model: It can store data in Key-value Pairs, Document-based, Graph-based, Column Family-based databases. Global distribution, horizontal partitioning, and automatic indexing capabilities are the same irrespective of the data model.
High Availability: It has 99.99 % availability for reads and writes for both multi region and single region Azure Cosmos DB accounts.
Low Latency: The global availability of Azure regions allows for the global distribution of data, which further makes it available nearest to the customers. This reduces the latency in retrieving data.

72. How do you connect between Cosmos DB and Azure Databricks?

In order to query Cosmos DB, you need to first create a configuration object that contains the configuration information.
If you are curious, read the configuration reference for details on all of the options.
The core items you need to provide are:
Endpoint: Your Cosmos DB url (i.e. https://youraccount.documents.azure.com:443/)
Masterkey: The primary or secondary key string for you Cosmos DB account
Database: The name of the database
Collection: The name of the collection that you wish to query

```
# Read Configuration
readConfig = {
    "Endpoint": "https://doctorwho.documents.azure.com:443/",
    "Masterkey": "YOUR-KEY-HERE",
    "Database": "DepartureDelays",
    "Collection": "flights_pcoll",
    "query_custom": "SELECT c.date, c.delay, c.distance, c.origin, c.destination FROM c WHERE c.origin = 'SEA'" // Optional
}

# Connect via azure-cosmosdb-spark to create Spark DataFrame
flights = spark.read.format(
    "com.microsoft.azure.cosmosdb.spark").options(**readConfig).load()
flights.count()
```

73. What is delta time travel? did you work with delta time travel? explain where you used it

Delta Time Travel is a feature of Databricks Delta that allows users to access and query previous versions of a Delta table. With Delta Time Travel, users can query a table as it appeared at any point in time, without having to create and manage multiple versions of the table manually.

To use Delta Time Travel, you can specify a version number or a timestamp when querying a Delta table. For example, you can use the AS OF syntax in a SQL query to query the table as it appeared at a specific timestamp, like this:

```
SELECT * FROM my_table AS OF TIMESTAMP '2022-02-14T12:00:00Z'
```

```
SELECT * FROM my_table VERSION AS OF 5
```

I have used Delta Time Travel in a project where we needed to maintain a history of changes to a customer database, so we could track changes over time and understand how the data had evolved. We used Delta Time Travel to query the database as it appeared at different points in time, and to compare versions of the data to identify changes and trends. This allowed us to gain insights into the data and make better decisions based on historical trends.

74. What is the difference between OLAP / OLTP?

OLAP (Online Analytical Processing) and OLTP (Online Transaction Processing) are two different approaches to managing data in a database system. The main differences between them are:

Purpose: OLAP is designed for complex, ad-hoc queries that involve aggregation and summarization of large data sets, and is optimized for data analysis and reporting. OLTP, on the other hand, is designed for managing transactional data, such as sales, orders, and inventory, and is optimized for fast, efficient data processing.

Data model: OLAP databases typically use a multidimensional model, where data is organized into dimensions and measures. This allows users to slice and dice data from different angles and explore relationships between different data elements. OLTP databases, on the other hand, typically use a normalized data model, where data is organized into tables that represent entities and relationships between them.

Query complexity: OLAP queries tend to be more complex than OLTP queries, as they involve more aggregation, grouping, and filtering of data. OLTP queries, on the other hand, tend to be simpler, as they typically involve selecting, inserting, updating, or deleting individual records.

Performance requirements: OLAP systems are designed to handle large, complex queries that may involve scanning large amounts of data, so performance is optimized for read-heavy workloads. OLTP systems, on the other hand, are optimized for write-heavy workloads, with fast response times for individual transactions.

In summary, OLAP is designed for complex data analysis and reporting, while OLTP is designed for efficient transaction processing. While some systems may incorporate elements of both OLAP and OLTP, the two approaches are fundamentally different and are optimized for different types of workloads.

75. Do you have an option to work with Spark without databricks in Azure?

Yes, you can work with Apache Spark on Azure without using Databricks. Azure provides several services for running Spark workloads, including:

Azure HDInsight: This is a fully-managed cloud service that makes it easy to process big data using popular open-source frameworks such as Hadoop, Spark, Hive, and HBase. With HDInsight, you can deploy and manage Spark clusters in Azure, and run Spark jobs using familiar tools and languages.

Azure Synapse Analytics: This is an analytics service that allows you to analyze large amounts of structured and unstructured data using both serverless and provisioned resources. Synapse Analytics includes a Spark pool that allows you to run Spark jobs and Spark SQL queries on large data sets.

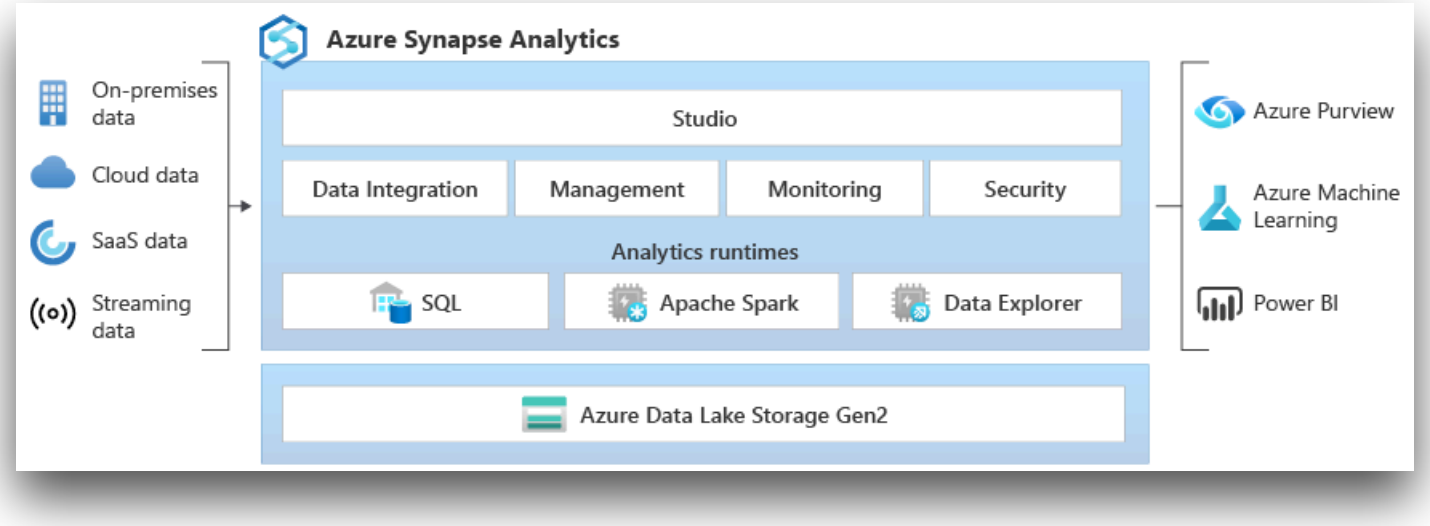
Azure Data Factory: This is a cloud-based data integration service that allows you to create and schedule data pipelines that can move and transform data between various sources and destinations, including Spark clusters.

Azure Kubernetes Service (AKS): This is a fully managed Kubernetes service that allows you to deploy and manage containerized applications and services, including Spark applications. With AKS, you can deploy Spark clusters as Kubernetes pods and manage them using Kubernetes tools and APIs.

These services provide different ways of running Spark workloads on Azure, depending on your needs and requirements. They offer varying levels of scalability, performance, and cost, and support different programming languages, data sources, and data processing frameworks.

76. What is Azure synapse analytics?

Azure Synapse Analytics is a cloud-based analytics service from Microsoft that enables organizations to analyze large amounts of data using both serverless and provisioned resources. Formerly known as Azure SQL Data Warehouse, Synapse Analytics is designed to help organizations break down data silos and gain insights from all their data, whether it's structured or unstructured, on-premises or in the cloud.



With Synapse Analytics, users can:

Ingest data from various sources, including streaming data, batch data, and big data.
Store data in a scalable and flexible data lake that uses Azure Blob Storage or Azure Data Lake Storage Gen2.
Analyze data using a variety of tools and services, including Apache Spark, Power BI, Azure Machine Learning, and Azure Databricks.
Build data pipelines and workflows to automate data integration and processing using Azure Data Factory.
Use a serverless SQL pool or dedicated SQL pool to run fast, scalable SQL queries on large data sets.
Secure data using advanced security and compliance features, including Azure Active Directory integration, network isolation, and row-level security.
Synapse Analytics also offers an integrated development environment (IDE) called Synapse Studio, which provides a unified workspace for data engineers, data scientists, and business analysts to collaborate on data-related projects. The IDE includes tools for data preparation, data transformation, data visualization, and machine learning, as well as a notebook environment for running code and exploring data.

77. How do you get all reporting employees of a manager as a list in Spark SQL?

To get all reporting employees of a manager as a list in Spark SQL, you can use a self-join and a GROUP BY clause. Here's an example SQL query:

This query joins the employees table with itself using the manager_id column to match each employee with their manager. The collect_list function is used to aggregate the employee_id values for each manager into a list. The GROUP BY clause groups the results by manager_id.

This query will return a result set with two columns: manager_id and reporting_employee_ids. The manager_id column contains the ID of each manager, and the reporting_employee_ids column contains a list of the IDs of all the employees reporting to that manager. You can further process this result set in Spark SQL or other Spark APIs to generate the desired output format.

```
SELECT mgr.manager_id,
       collect_list(emp.employee_id) AS reporting_employee_ids
FROM employees emp
JOIN employees mgr
ON emp.manager_id = mgr.employee_id
GROUP BY mgr.manager_id
```

78. How do you loop through all records of a delta table using SQL and Python?

You can loop through all records of a Delta table using SQL and Python by using the Delta Lake pySpark API in a Python script. Here's an example script:

```python
# Load the Delta table into a PySpark DataFrame
df = spark.read.format("delta").load("/mnt/delta/mytable")

# Loop through each row of the DataFrame and print the values
for row in df.collect():
    print(row)
```

it loops through each row of the DataFrame using the df.collect() method, and prints the values of each row.

Alternatively, you can use Spark SQL to loop through all records of a Delta table. Here's an example SQL query:

```python
# Load the Delta table into a PySpark DataFrame
df = spark.sql("select * from emp")

# Loop through each row of the DataFrame and print the values
for row in df.collect():
    print(row)
```

79. What are Graph Frames in databricks?

GraphFrames is a package for Apache Spark that provides DataFrame-based graphs. It provides high-level APIs in Java, Python, and Scala. It aims to provide both the functionality of GraphX and extended functionality taking advantage of Spark DataFrames.

80. How do you covert a dataframe to graph frame?

To convert a DataFrame to a GraphFrame in Databricks, you can use the GraphFrame.fromEdges method to create a GraphFrame from a DataFrame that represents the edges of the graph. Here's an example of how to do this:

```
from graphframes import *

# create a DataFrame of edges
edges = spark.createDataFrame([(1, 2), (2, 3), (3, 4), (4, 1)], ["src", "dst"])

# create a GraphFrame from the edges DataFrame
g = GraphFrame.fromEdges(edges)

# display the vertices and edges of the graph
display(g.vertices)
display(g.edges)

```

In this example, the createDataFrame method is used to create a DataFrame edges that contains two columns, src and dst, which represent the edges of the graph. The fromEdges method of the GraphFrame class is then used to create a GraphFrame g from the edges DataFrame. Finally, the display method is used to show the vertices and edges of the graph.

Note that before you can convert a DataFrame to a GraphFrame, you need to make sure that the DataFrame has the correct schema and contains the appropriate columns to represent the vertices and edges of the graph. In addition, you may need to perform additional transformations on the DataFrame to prepare it for use as a graph, such as adding vertex properties or filtering out irrelevant data.

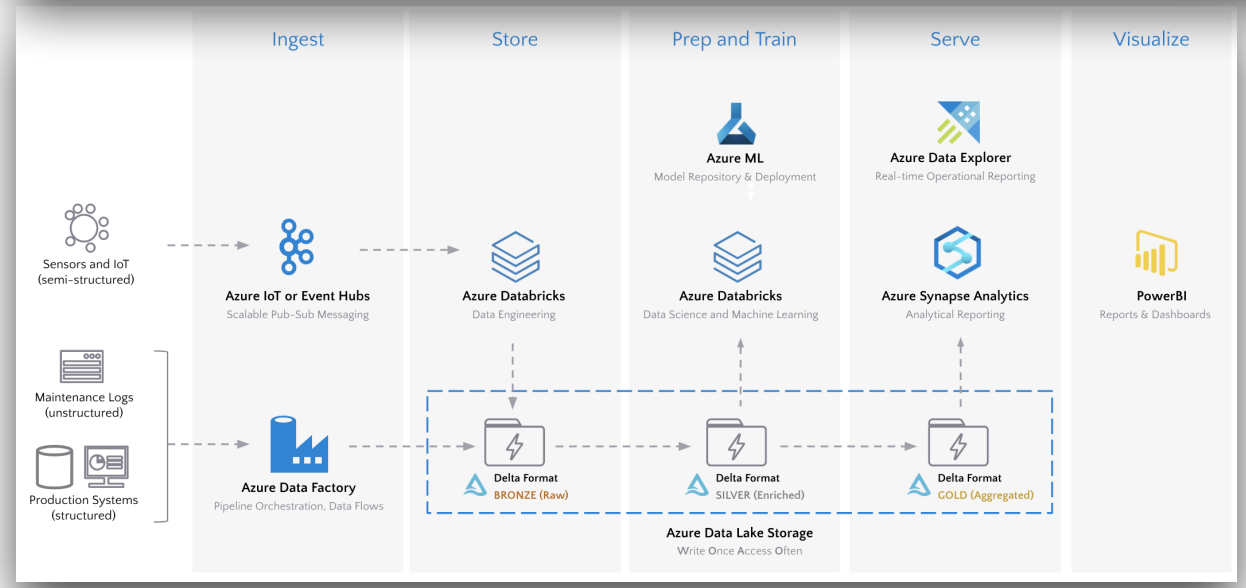
81. How do you stream data from IoT devices in Azure databricks?

Data Ingest - stream real-time raw sensor data from Azure IoT Hubs into the Delta format in Azure Storage
Data Processing - stream process sensor data from raw (Bronze) to silver (aggregated) to gold (enriched) Delta tables on Azure Storage

Azure Databricks Configuration Required

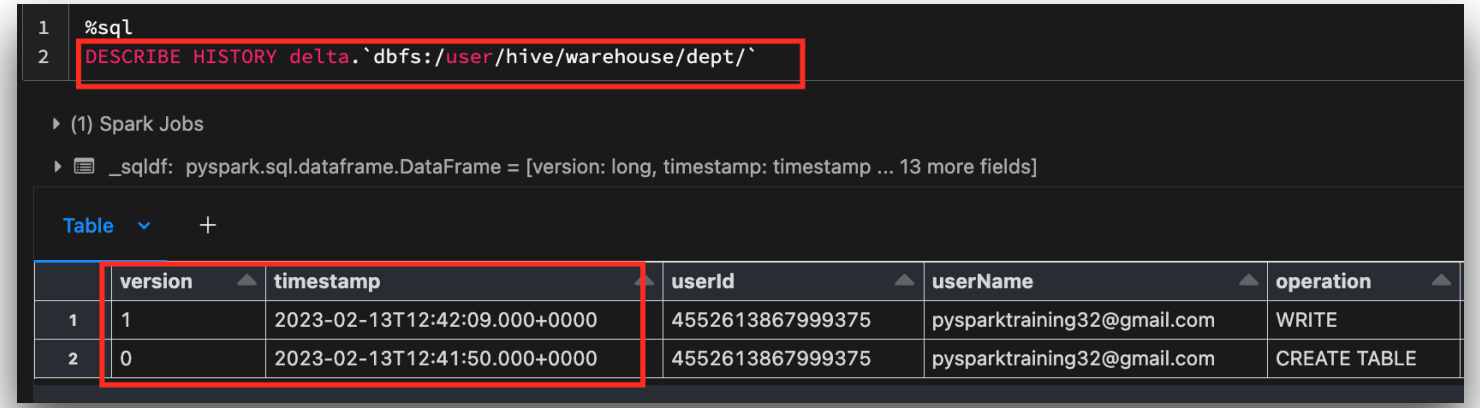
- 3-node (min) Databricks Cluster running **DBR 7.0+** and the following libraries:
- **Azure Event Hubs Connector for Databricks** - Maven coordinates `com.microsoft.azure:azure-eventhubs-spark_2.12:2.3.17`

```
IOT_CS = """Endpoint=sb://[redacted]/;
SharedAccessKeyName=iothubowner;
SharedAccessKey=[redacted]"""

# dbutils.secrets.get('iot','iothub-cs') # IoT Hub connection string (Event Hub Compatible)
ehConf = {
  'eventhubs.connectionString':sc._jvm.org.apache.spark.eventhubs.EventHubsUtils.encrypt(IOT_CS),
  'ehName':dbutils.widgets.get("Event Hub Name")
}

# Read directly from IoT Hub using the EventHubs library for Databricks
iot_stream = (
  spark.readStream.format("eventhubs")                                        # Read from IoT Hubs directly
    .options(**ehConf)                                                        # Use the Event-Hub-enabled connect string
    .load()                                                                   # Load the data
    .withColumn('reading', F.from_json(F.col('body').cast('string'), schema)) # Extract the "body" payload from the messages
    .select('reading.*', F.to_date('reading.timestamp').alias('date'))        # Create a "date" field for partitioning
)
```



82. How do you find available versions of a delta table?

Describe History Table Returns provenance information, including the operation, user, and so on, for each write to a table. Table history is retained for 30 days.



83. How do you find data history from Delta tables using time stamp or version?

Read older versions of data using time travel
You can query previous snapshots of your Delta table by using time travel. If you want to access the data that you overwrote, you can query a snapshot of the table before you overwrote the first set of data using the **VERSION AS OF / TIMESTAMP AS OF** option.

```
SELECT count(*) FROM my_table TIMESTAMP AS OF "2019-01-01"
SELECT count(*) FROM my_table TIMESTAMP AS OF date_sub(current_date(), 1)
SELECT count(*) FROM my_table TIMESTAMP AS OF "2019-01-01 01:30:00.000"
```

```
SELECT count(*) FROM my_table VERSION AS OF 5238
SELECT count(*) FROM my_table@v5238
SELECT count(*) FROM delta.`/path/to/my/table@v5238`
```

84. How do remove all versions of a delta table?

To remove all versions of a Delta table in Databricks, you can use the VACUUM command with the RETAIN 0 HOURS option. This command will remove all the older versions of the table that are older than the retention period, which in this case is set to zero hours. Here is an example:

```
%sql
VACUUM delta.`/path/to/table` RETAIN 0 HOURS;
```

85. Can we do subquery based update statements on databricks delta tables?

Yes, We can achieve Update statements on databricks delta tables except below two cases.

- Nested subqueries, that is, a subquery inside another subquery
- A NOT IN subquery inside an OR, for example, a = 3 OR b NOT IN (SELECT c from t)

```
UPDATE events SET eventType = 'click' WHERE eventType = 'clk'

UPDATE all_events
   SET session_time = 0, ignored = true
 WHERE session_time < (SELECT min(session_time) FROM good_events)

UPDATE orders AS t1
   SET order_status = 'returned'
 WHERE EXISTS (SELECT oid FROM returned_orders WHERE t1.oid = oid)

UPDATE events
   SET category = 'undefined'
 WHERE category NOT IN (SELECT category FROM events2 WHERE date > '2001-01-01')

UPDATE events
   SET ignored = DEFAULT
WHERE eventType = 'unknown'
```

86. Explain various SQL joins with example?

In Spark SQL, there are four types of joins that can be used to combine data from two or more tables:

Inner Join: Returns only the rows where there is a match between the keys in both tables. Syntax: SELECT ... FROM table1 JOIN table2 ON table1.key = table2.key

Left Outer Join: Returns all the rows from the left table and the matching rows from the right table. If there is no match in the right table, it returns null values for the right table's columns. Syntax: SELECT ... FROM table1 LEFT JOIN table2 ON table1.key = table2.key

Right Outer Join: Returns all the rows from the right table and the matching rows from the left table. If there is no match in the left table, it returns null values for the left table's columns. Syntax: SELECT ... FROM table1 RIGHT JOIN table2 ON table1.key = table2.key

Full Outer Join: Returns all the rows from both tables and fills null values for the columns that do not have a matching key in the other table. Syntax: SELECT ... FROM table1 FULL OUTER JOIN table2 ON table1.key = table2.key

Left Semi Join: This join returns all the rows from the left table for which there is a match in the right table, and it does not return any columns from the right table. In other words, it is similar to an inner join, but it only returns the columns from the left table. Syntax: SELECT ... FROM table1 LEFT SEMI JOIN table2 ON table1.key = table2.key

Left Anti Join: This join returns all the rows from the left table for which there is no match in the right table, and it does not return any columns from the right table. In other words, it returns all the rows from the left table that do not have a corresponding match in the right table. Syntax: SELECT ... FROM table1 LEFT ANTI JOIN table2 ON table1.key = table2.key

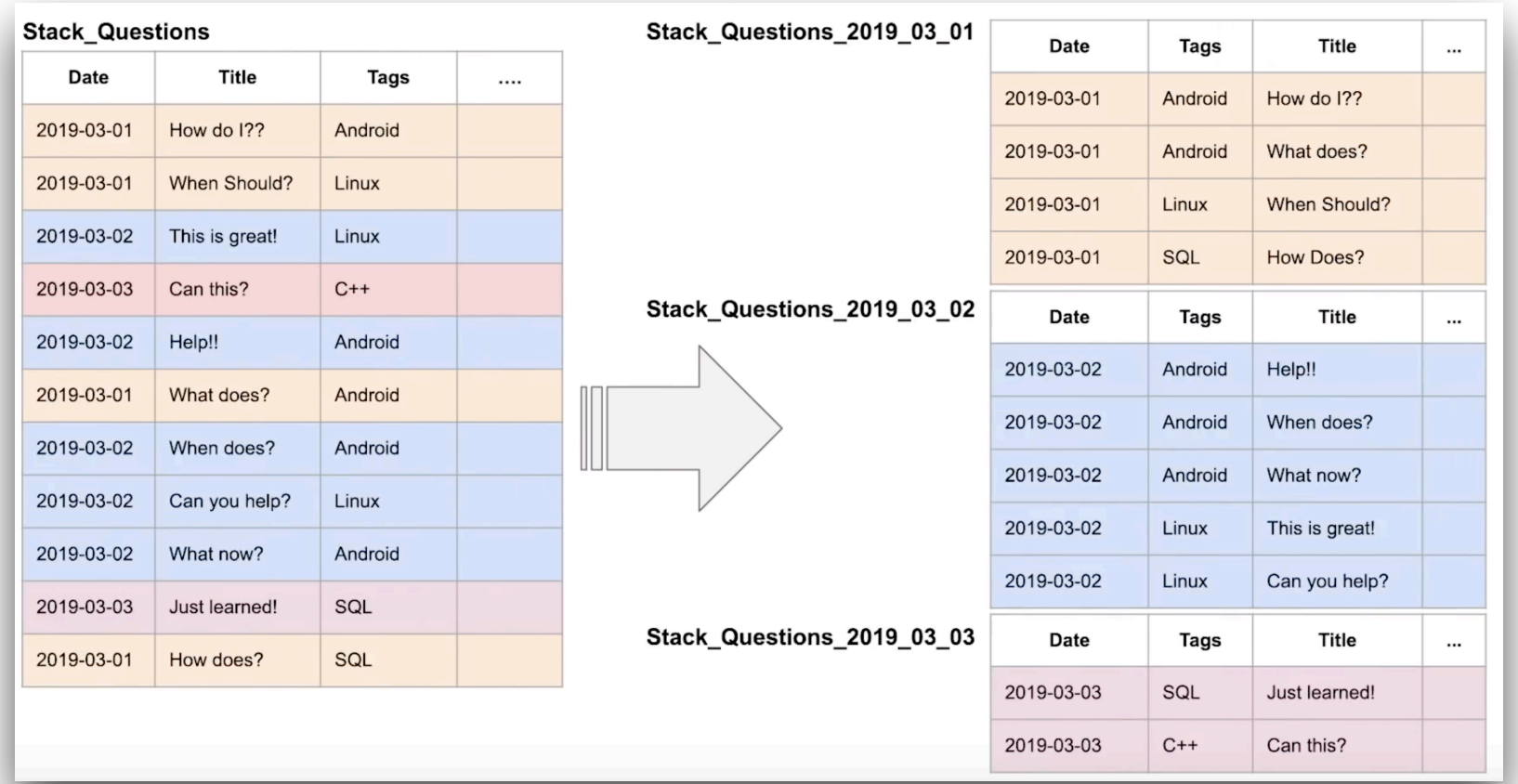
```
1  -- Inner Join - Matching data from both tables
2  SELECT * FROM emp as e INNER JOIN dept as d IN e.deptno = d.deptno;
3  -- Left Outer Join -- matching data from both tables + un-matched data from left table
4  SELECT * FROM emp as e LEFT OUTER JOIN dept as d IN e.deptno = d.deptno;
5  -- Right Outer Join -- matching data from both tables + un-matched data from right table
6  SELECT * FROM emp as e RIGHT OUTER JOIN dept as d IN e.deptno = d.deptno;
7  -- FULL Outer Join -- matching data from both tables + un-matched data from both tables
8  SELECT * FROM emp as e FULL OUTER JOIN dept as d IN e.deptno = d.deptno;
9  -- Left Semi Join -- Only Matched Data from Left Table
10 SELECT * FROM emp as e LEFT SEMI JOIN dept as d IN e.deptno = d.deptno;
11 -- Left Anti Join -- Only Un-Matched Data From Left Table
12 SELECT * FROM emp as e LEFT ANTI JOIN dept as d IN e.deptno = d.deptno;
13
```

87. What is partition in databricks? how does it work?

A partitioned table is divided into segments, called partitions, that make it easier to manage and query your data. By dividing a large table into smaller partitions, you can improve query performance and control costs by reducing the number of bytes read by a query. You partition tables by specifying a partition column which is used to segment the table.

If a query uses a qualifying filter on the value of the partitioning column, BigQuery can scan the partitions that match the filter and skip the remaining partitions. This process is called pruning.

In a partitioned table, data is stored in physical blocks, each of which holds one partition of data. Each partitioned table maintains various metadata about the sort properties across all operations that modify it.



88. Read a text contains millions of words, and find top 10 words from entire text excluding prepositions and articles?


```
# reading file and creating RDD
rdd_text = sc.textFile("dbfs:/FileStore/tables/wordcount.txt")
# removing white spaces using strip function
# converting into lower case using lower() function
# string to word convertion using split() function
rdd_flatmap = rdd_text.flatMap(lambda a :a.strip().lower().split(" "))
# filtering empty spaces or empty word
rdd_filter = rdd_flatmap.filter(lambda a:a!='')
# converting key value pair as adding 1 to each word
rdd_map = rdd_filter.map(lambda a:(a,1))
# taking word count at each key using reducebykey
rdd_reduce = rdd_map.reduceByKey(lambda a,b:a+b)
# changing posistions as key to value and value to key for sorting data based on max word count
rdd_map2 = rdd_reduce.map(lambda a:(a[1],a[0]))
# sorting max word count
rdd_final = rdd_map2.sortByKey(False)
# take top 10
rdd_final.take(10)
```

89. How do you find all possible substrings of a given name or string excluding space and special characters? for example, “data” → “d”, ”a”, “t”, “da”, “dt”, “td”, “ta”, “at”, “aa”…etc….”data”

```
# Define the input string
input_str = "data"

# Define a list to store the substrings
substrings = []

# Generate all possible substrings
for i in range(len(input_str)):
    for j in range(i+1, len(input_str)+1):
        substring = input_str[i:j]
        if " " not in substring and not any(char in substring for char in "!@#$%^&*()_-+={}[]|\:;\"'<>,.?/"):
            substrings.append(substring)

# Print the list of substrings
print(substrings)
```

```
['d', 'da', 'dat', 'data', 'a', 'at', 'ata', 't', 'ta', 'a']
```

90. Read data from multiple CSV files of same schema, and display number of records from each file with filename? get the file name into dataframe / table storage

Get no of rows on each file in a dataframe?

- **Answer:** Using `input_file_name()` function and groupBy transaformation we can achive no of row on each file

Cmd 5

```
from pyspark.sql.functions import input_file_name
df_airlines = spark.read.csv("/databricks-datasets/asa/airlines",header=True)
df_airlines = df_airlines.withColumn("FILE_NAME",input_file_name())
display(df_airlines.groupBy("FILE_NAME").count())
```

91. There are 10 million records in the table and the schema does not contain the ModifiedDate column. One cell was modified the next day in the table. How will you fetch that particular information that needs to be loaded into the warehouse?

In Spark, a left anti join is a type of join operation that returns all the rows from the left DataFrame that do not have a matching key in the right DataFrame. In other words, it returns the rows in the left DataFrame that are not present in the right DataFrame.

The left anti join operation is useful when you want to find the rows in one DataFrame that are not present in another DataFrame based on a common key column. This operation can be used to filter out the rows in the left DataFrame that do not have a corresponding key in the right DataFrame.

The left anti join operation can be performed using the join method of the DataFrame API, with the how parameter set to "left_anti".
Here's an example:

```
# Create two sample DataFrames
df1 = spark.sql("select * from source_table")
df2 = spark.sql("select * from target_table")

# Perform a left anti join on the "id" column
left_anti_join = df1.join(df2, ["id"], "left_anti")

# Show the result of the left anti join
left_anti_join.show()
```

92. Read covid data through from https://data.cdc.gov/ and provide number of diseased patients per day for the last 5 days.

```
import requests
# Set the URL for the CDC COVID data API
url = "https://data.cdc.gov/resource/vbim-akqf.json"

# Make a request to the API to retrieve the data
response = requests.get(url)
 # Group the data by submission date and sum the number of new deaths
df = spark.read.json(sc.parallelize([response.json()]))
# Print the result
display(df.filter("cdc_report_dt >current_date()-5"))

```

93. Make a secure connection from databricks to ADLS and read data from a CSV file and convert it into a delta table with appropriate schema

```python
# Import the necessary libraries
from pyspark.sql.types import *
from pyspark.sql.functions import *

# Define the storage account name, container name, and CSV file path
storage_account_name = "your_storage_account_name"
container_name = "your_container_name"
csv_file_path = "your_csv_file_path"
account_key = "account key"

# Define the schema for the Delta table
schema = StructType([
  StructField("id", IntegerType(), True),
  StructField("name", StringType(), True),
  StructField("price", FloatType(), True),
  StructField("timestamp", TimestampType(), True),
  StructField("ingestion_time", TimestampType(), True)
])

# Define the options for reading the CSV file
read_options = {"header": True,"inferSchema": False,"delimiter": ","}

spark.conf.set("fs.azure.account.key.{storage_account_name}.dfs.core.windows.net","{account_key}")
spark.conf.set("fs.azure.createRemoteFileSystemDuringInitialization", "true")
dbutils.fs.ls("abfss://{container_name}@{storage_account_name}.dfs.core.windows.net/")
spark.conf.set("fs.azure.createRemoteFileSystemDuringInitialization", "false")

# Read the CSV file from ADLS Gen2
df = spark.read.format("csv").options(read_options) \
  .load(f"abfss://{container_name}@{storage_account_name}.dfs.core.windows.net/{csv_file_path}")

# Add the ingestion time column to the DataFrame
df = df.withColumn("ingestion_time", current_timestamp())

# Write the DataFrame as a Delta table
df.write.format("delta").mode("overwrite").option("mergeSchema", "true").save("/mnt/data/my-delta-table")
```

94. If you are running more no of pipelines and its taking longer time to execute. How to resolve this type of issues?

If you are experiencing slow pipeline execution times in Azure Data Factory, there are several steps you can take to optimize performance:

Use parallelism: If your pipelines are processing large amounts of data, consider using parallelism to split the workload across multiple activities or pipelines. This can help speed up processing times and reduce the overall time required to complete the pipeline.

Optimize data movement: Data movement can be a bottleneck for pipeline performance, so it's important to optimize the data movement as much as possible. This could involve compressing data before transfer, using partitioning to move data in smaller chunks, or using a dedicated transfer service like Azure Data Box.

Optimize data transformation: If your pipelines involve complex data transformation, consider using more efficient data processing technologies such as Databricks / synapse analytics. This can help reduce the time required to transform data and speed up overall pipeline execution.

Optimize infrastructure: Consider upgrading the infrastructure used to run the pipelines. This might involve upgrading the VM size, increasing the number of nodes in a cluster, or scaling out by adding more worker nodes.

Monitor and optimize: Monitor the performance of your pipelines regularly and use performance metrics to identify areas for improvement. This could involve using tools like Azure Monitor or third-party monitoring solutions to track metrics such as execution time, data throughput, and resource utilization.

95. What is auto resolve integration runtime in azure data factory?

AutoResolveIntegrationRuntime. This is the default integration runtime, and the region is set to auto-resolve. That means that Azure Data Factory decides the physical location of where to execute activities based on the source, sink, or activity type.

In Azure Data Factory, an Integration Runtime (IR) is a compute infrastructure used to provide data integration capabilities across different network environments. The AutoResolve Integration Runtime is a type of Integration Runtime that provides the ability to automatically select the most appropriate Integration Runtime for a given data movement task.

When using the AutoResolve Integration Runtime, you don't need to manually specify which Integration Runtime to use for a specific task. Instead, the Data Factory service will automatically select the appropriate Integration Runtime based on the source and destination data store types, the connectivity requirements, and other factors.

This can be useful in scenarios where you have a variety of data stores and different data movement tasks with varying requirements. The AutoResolve Integration Runtime can help simplify the process of configuring and executing these tasks by automatically selecting the right Integration Runtime for each one.

Note that there are other types of Integration Runtimes available in Azure Data Factory, including the Azure-SSIS Integration Runtime for running SQL Server Integration Services (SSIS) packages in the cloud, and the Self-Hosted Integration Runtime for connecting to on-premises data stores.

96. Different Types of triggers in ADF?

In Azure Data Factory, there are three types of triggers available for executing pipelines:

Schedule Trigger: A Schedule Trigger allows you to run a pipeline on a recurring schedule, such as once a day or once an hour. You can define the frequency and start time for the trigger, as well as any additional parameters, and the trigger will automatically execute the pipeline at the specified times.

Event-Based Trigger: An Event-Based Trigger allows you to execute a pipeline in response to an event, such as a file being added to a data store, a message being posted to a queue, or an HTTP request being received. You can configure the trigger to monitor specific events and trigger the pipeline when those events occur.

Tumbling Window Trigger: A Tumbling Window Trigger allows you to execute a pipeline on a recurring schedule, but with a more complex definition of the trigger time. You can define a start time and end time for the trigger, as well as the duration of the window, and the trigger will execute the pipeline at the start of each window.

Each of these trigger types has its own specific use cases and benefits. For example, a Schedule Trigger might be appropriate for a pipeline that needs to run on a set schedule, while an Event-Based Trigger might be useful for a pipeline that needs to process data in real-time as it's generated. The Tumbling Window Trigger might be used when you need to process data in regular, overlapping time periods.

97. How many ways we can execute data factory pieplines?

There are several ways to execute pipelines in Azure Data Factory, including:

Trigger-based execution: You can create a trigger that automatically executes a pipeline on a specified schedule or when an event occurs (such as when new data is added to a data store).

Ad-hoc execution: You can manually execute a pipeline from the Azure Data Factory user interface or programmatically using the REST API or Azure PowerShell.

Event-based execution: You can use Azure Event Grid to trigger a pipeline when an event occurs in an Azure service such as Blob Storage, Event Hub, or IoT Hub.

External execution: You can use a third-party scheduling tool or an orchestration tool such as Azure Logic Apps or Azure Functions to execute pipelines.

Continuous integration and delivery (CI/CD): You can use Azure DevOps or another CI/CD tool to automatically build, test, and deploy Data Factory pipelines.

Overall, these execution options provide a lot of flexibility for running Data Factory pipelines in different scenarios, whether it's on a set schedule, in response to an event, or manually triggered as needed.

98. Best Way to copy large from on premises to data lake using ADF?

The performance of a self-hosted integration runtime in Azure Data Factory can be influenced by a variety of factors, including the configuration of the runtime and the performance of the hardware it is running on. Here are some tips to optimize the performance of a self-hosted integration runtime:

Use a dedicated machine: To optimize performance, use a dedicated machine for the self-hosted integration runtime. Avoid sharing the machine with other workloads that may affect its performance.
Optimize machine resources: Ensure that the machine used for the self-hosted integration runtime has sufficient CPU, memory, and disk space to handle the workloads.
Use compression: Compressing the data before transferring it can reduce the amount of data that needs to be transferred, which can result in faster transfer speeds.
Use multi-part uploads: If the file is very large, consider using multi-part uploads. This allows the file to be split into smaller chunks, which can be transferred in parallel. This can significantly improve the transfer speed.
Binary copy is a feature in Azure Data Factory that allows you to copy files between different file-based data stores, such as Azure Blob Storage, Azure Data Lake Storage, and on-premises file systems. Binary copy enables you to copy large files quickly and efficiently, without having to read the entire file into memory.
A self-hosted integration runtime can be installed on-premises and used to securely transfer data between on-premises data stores and cloud-based data stores. The self-hosted integration runtime can take advantage of the faster network speeds and reduced latency of an on-premises network.

Use Azure ExpressRoute: Azure ExpressRoute provides a dedicated, private connection between an on-premises network and Azure. This can improve the performance of data transfer by providing faster and more reliable connectivity.

Use Azure Data Box: If the data is very large, consider using Azure Data Box. Azure Data Box is a physical appliance that can be shipped to the on-premises location to transfer large amounts of data.

Optimize the on-premises network: Ensure that the on-premises network is optimized for data transfer, with sufficient bandwidth and low latency. Consider upgrading the network infrastructure if necessary.

99. How To Get the latest added file in a folder using Azure Data Factory?

To get the latest added file in a folder using Azure Data Factory, you can use the "Get Metadata" activity with a child item of "Child Items" and sort the results by creation or modification time. Here are the steps to do this:

Add a "Get Metadata" activity to your pipeline and configure it to connect to the folder you want to monitor.

In the "Get Metadata" activity, select the "Child Items" option as the child item.

Under the "Field List" tab, add the "creationTime" and/or "lastModified" fields.

Under the "Field List" tab, click on the "Add dynamic content" button to add an expression that sorts the files by the creation or modification time. The expression should look like this:

```
@orderBy(body('Get Metadata')?['childItems'], desc, item()?['lastModified'])
```

This will sort the files in descending order by their creation or modification time.

Finally, add an "If Condition" activity to check if any files were found in the folder. The expression should be:

```
@empty(activity('Get Metadata').output.childItems)
```

If the output is empty, you can use a "Set Variable" activity to set a default value. If there are files, you can use a "Set Variable" activity to set the latest file name and/or path using the expression:

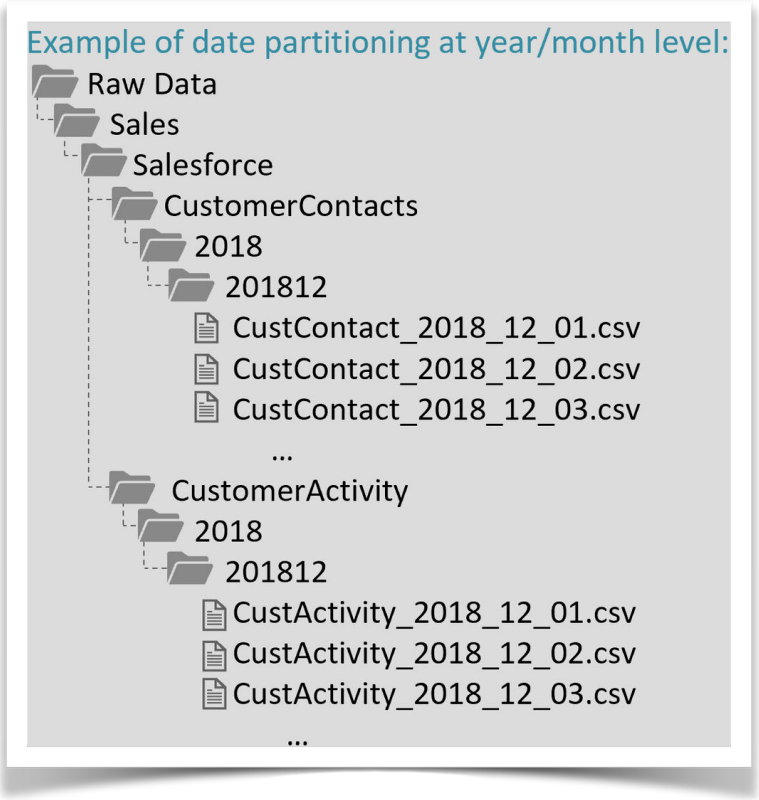
```
@first(activity('Get Metadata').output.childItems).name
```

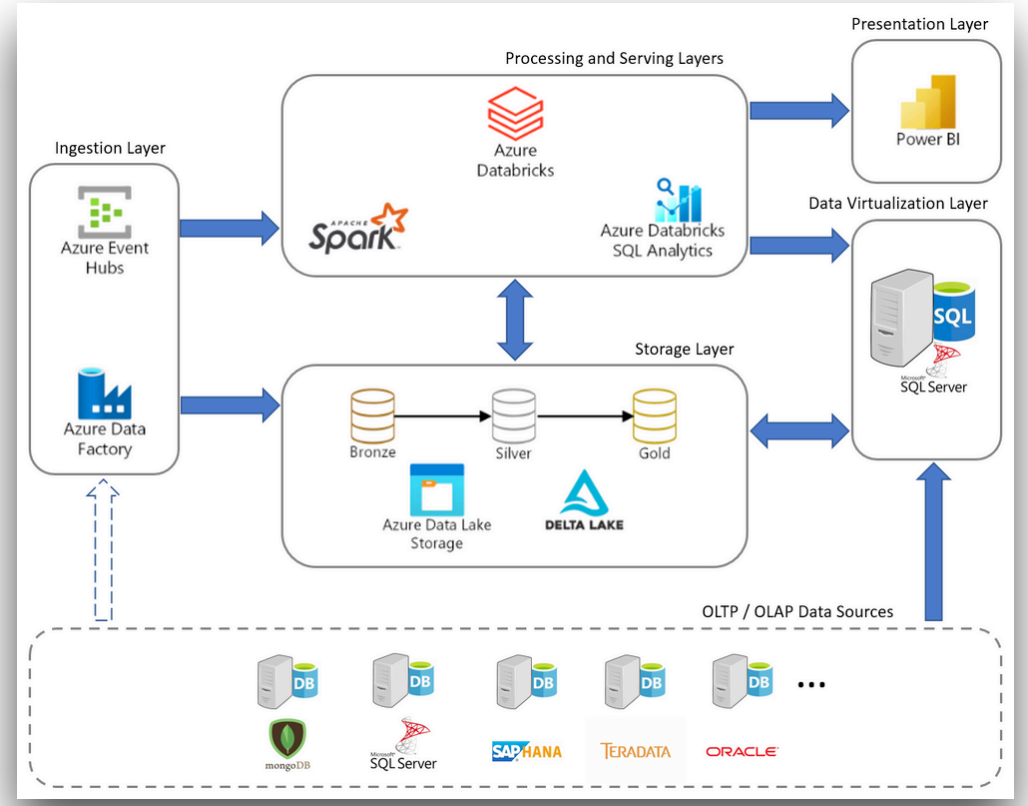
100.What is the meaning of a hierarchical folder structure in data engineering, and how is it used in organizing and managing data within a data lake??

In data engineering, hierarchical folder structure refers to the organization of data files and folders in a hierarchical or tree-like structure, where files and folders are arranged in a parent-child relationship. In this structure, each folder can contain one or more sub-folders, and each sub-folder can contain one or more files or additional sub-folders. This allows for a logical organization of data files and facilitates efficient storage and retrieval of data.

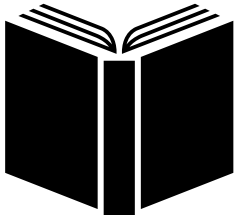
In the context of data lakes, a hierarchical folder structure can be used to organize data files in a way that reflects the different data domains, data sources, or business units. For example, a data lake may have a top-level folder for each data domain (such as sales, finance, or customer data), and within each domain folder, there may be sub-folders for each data source (such as ERP systems, CRM systems, or social media platforms).

A hierarchical folder structure can also be used to enforce data governance and access controls, by setting permissions at the folder or file level to control who can view or modify data.





Learn



&

Lead