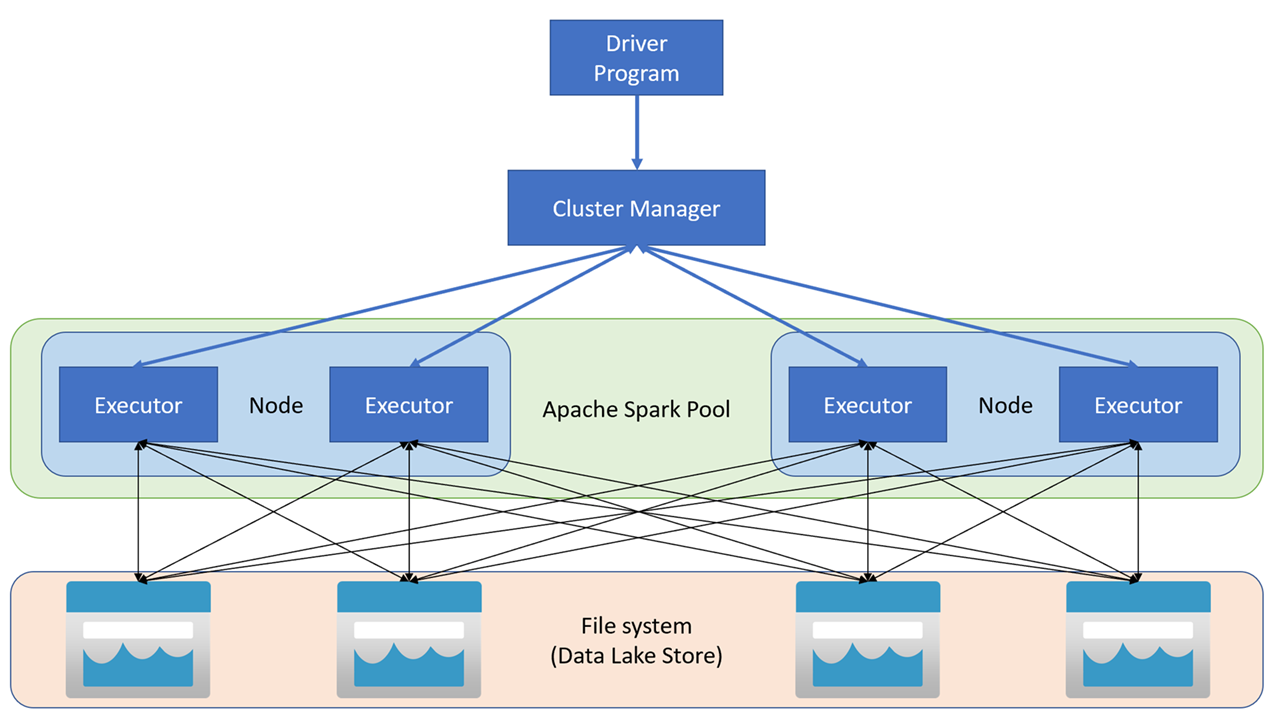
Spark Introduction

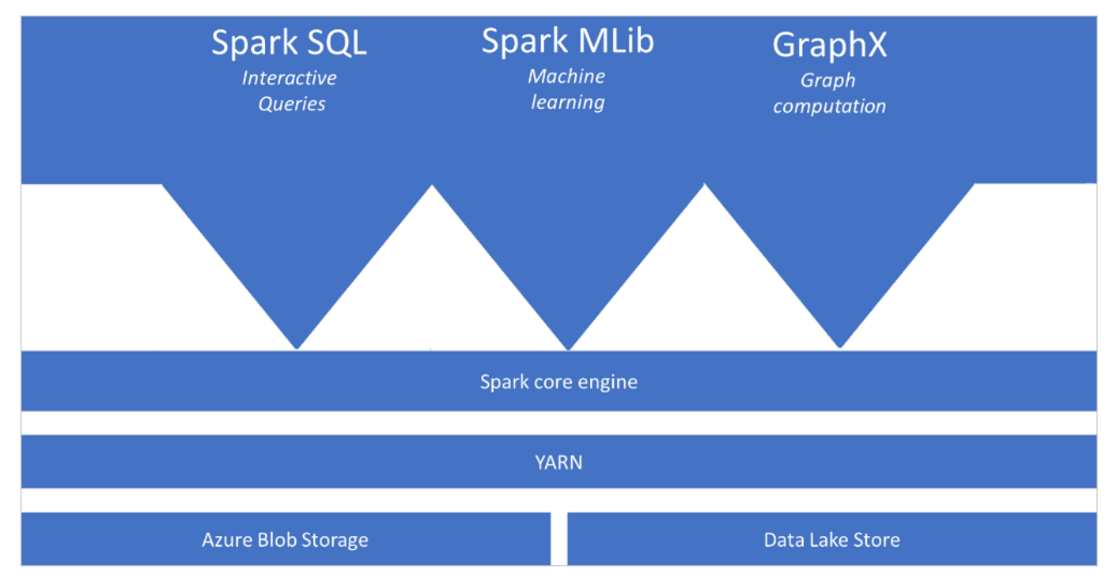
* Apache Spark is distributed data processing framework that enables large-scale data analytics by coordinating work across multiple processing nodes in a cluster.
* Apache Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in your main program (called the driver program).
* The SparkContext connects to the cluster manager, which allocates resources across applications using an implementation of Apache Hadoop YARN.
* Once connected, Spark acquires executors on nodes in the cluster to run your application code.



* The SparkContext runs the main function and parallel operations on the cluster nodes, and then collects the results of the operations.
* The nodes read and write data from and to the file system and cache transformed data in-memory as Resilient Distributed Datasets (RDDs).
* The SparkContext is responsible for converting an application to a directed acyclic graph (DAG).
* The graph consists of individual tasks that get executed within an executor process on the nodes.
* Each application gets its own executor processes, which stay up for the duration of the whole application and run tasks in multiple threads.

Synapse Spark Pool Introduction

* Spark pools in Azure Synapse offer a fully managed Spark service.
* Apache Spark is a parallel processing framework that supports in-memory processing to boost the performance of big-data analytic applications.
* **Apache Spark for Azure Synapse is** used for data preparation, data engineering, ETL, and machine learning
* It supports multiple languages (Scala, Python, SparkSQL, and C#)
* It supports Batch processing, Streaming, Machine Learning and Advanced Analytics.



**Synapse Spark Pool Features**

Azure Synapse Analytics makes it easy to create and configure a serverless Apache Spark pool in Azure

* Multiple spark pool with different settings
* Scaling
* Auto Termination
* Pre-installed Libraries

**Spark Pool Components:**

Driver Node: Runs Spark **driver service** and additional management services

Worker Node: Run **spark Executer** service. There can be more than one executer on node.

**Lab 1: Create Spark Pool**

Manage Tab🡪Apache Spark Pool🡪New

Graphical user interface, text, application, email

Description automatically generated

Configure Auto Pause setting in Additional settings tab.

Notebook

* A Synapse notebook is a web interface for you to create files that contain live code , visualizations, and narrative text.
* Notebook supports PySpark (Python),Spark (Scala),Spark SQL and .NET Spark (C#) languages.
* Multiple users can work in collaboration
* Each notebook uses set of resources assigned to it from spark pool. Each notebook has its on spark session.
* It is possible to use multiple languages in one notebook by specifying the language using a magic command at the beginning of a cell

|  |  |
| --- | --- |
| **Language** | **Magic Command** |
| Python | %%pyspark |
| Scala | %%spark |
| SparkSQL | %%sql |
| .NET for Spark C# | %%csharp |

**Lab 2: First Notebook**

Develop Tab🡪+🡪Notebook

|  |
| --- |
| "Hello Spark" |
| no=1  no=no+10  print(no) |
| ## This is my First Notebook |
| new\_rows = [('CA',22, 45000),("WA",35,65000) ,("WA",50,85000)]  demo\_df = spark.createDataFrame(new\_rows, ['state', 'age', 'salary'])  demo\_df.show() |

**For Shortcut Keys**

Refer: https://docs.microsoft.com/en-us/azure/synapse-analytics/spark/apache-spark-development-using-notebooks#shortcut-keys-under-command-mode

Understanding DataFrames

* DataFrames are a collection of data organized into named columns.
* DataFrames are to Apache Spark, what tables are to relational databases.

**Example:**

new\_rows = [('CA',22, 45000),("WA",35,65000) ,("WA",50,85000)]

demo\_df = spark.createDataFrame(new\_rows, ['state', 'age', 'salary'])

demo\_df.show()

## Accessing data from a Synapse Spark pool

You can use Spark in Azure Synapse Analytics to work with data from various sources, including:

* A data lake based on the primary storage account for the Azure Synapse Analytics workspace.
* A data lake based on storage defined as a linked service in the workspace.
* A dedicated or serverless SQL pool in the workspace.
* An Azure SQL or SQL Server database (using the Spark connector for SQL Server)
* An Azure Cosmos DB analytical database defined as a linked service and configured using Azure Synapse Link for Cosmos DB.
* An Azure Data Explorer Kusto database defined as a linked service in the workspace.
* An external Hive metastore defined as a linked service in the workspace.

One of the most common uses of Spark is to work with data in a data lake, where you can read and write files in multiple commonly used formats, including delimited text, Parquet, Avro, and others.

Load Data in Notebook

**Lab 3: Read Data From Storage**

|  |
| --- |
| #Read data from data lake (Ensure that you have Employee.txt in DataLake)  from pyspark.sql import SparkSession  from pyspark.sql.types import \*  account\_name = "dssdemodatalake1"  container\_name = "taxidata"  relative\_path = "Your path"  adls\_path = 'abfss://%s@%s.dfs.core.windows.net/' % (container\_name, account\_name)  spark.conf.set("fs.azure.account.auth.type.%s.dfs.core.windows.net" %account\_name, "SharedKey")  spark.conf.set("fs.azure.account.key.%s.dfs.core.windows.net" %account\_name ,"JbH7YWLRSfGNP79krBtcTcGVCQppCnAhe+OC6MCFmBjnWI1jxStwZnEo+PX/6SpXipZ0z20Kfsd9+ASt6yHBKQ==")  df1 = spark.read.option('header', 'true') \                  .option('delimiter', ',') \                  .csv(adls\_path + '/Employee.txt') |
| df1.show() |
| #Read data from Azure Storage (Ensure that you have orders-2013.txt in blob)  from pyspark.sql import SparkSession  from pyspark.sql.types import \*  blob\_account\_name = "dssdemosa1"  blob\_container\_name = "mycontainer"  blob\_relative\_path = "orders-2013.txt"  blob\_sas\_token = "sp=r&st=2022-03-23T12:36:29Z&se=2022-03-23T20:36:29Z&spr=https&sv=2020-08-04&sr=b&sig=U2Gn3X3761KwCcw%2BG9gvt0mMzNLv9uTE0zvZ4Zk1%2BzI%3D"  wasbs\_path = 'wasbs://%s@%s.blob.core.windows.net/%s' % (blob\_container\_name, blob\_account\_name, blob\_relative\_path)  spark.conf.set('fs.azure.sas.%s.%s.blob.core.windows.net' % (blob\_container\_name, blob\_account\_name), blob\_sas\_token)  df = spark.read.option("header", "true") \              .option("delimiter","\t") \              .csv(wasbs\_path) |
| %%pyspark  **#Read data from primary adls storage**  df = spark.read.load('abfss://taxidata@dssdemodatalake1.dfs.core.windows.net/Employee.txt', format='text')  display(df.limit(10)) |
| df1 = spark.read.option('header', 'true') \                  .option('delimiter', ',') \                  .csv('abfss://taxidata@dssdemodatalake1.dfs.core.windows.net/Employee.txt') |

Lab:Analyse Product Data.

Refer:

Lab: Analyse Orders Data

Refer:

MSSparkUtil library

* Microsoft Spark Utilities (MSSparkUtils) is a builtin package to help you easily perform common tasks.
* You can use MSSparkUtils to work with file systems, to get environment variables, to chain notebooks together, and to work with secrets.
* To Access Data Lake Storage Gen2 ensure the account and the (MSI) Managed service Identity has **Storage Blob Data Contributor Role**

**Lab: Perform various File system related operations**

Create sampledata folder in primary container attached to synapse and Load some sample files

Refer: **mssparkutilityDemo**

Mount Storage in Azure Synapse

Mssparkutils Provides APIs for mount/unmount remote Blob Storage or data lake storage.

Azure Synapse Analytics supports three authentication methods for the trigger mount operation:

1. LinkedService
2. AccountKey
3. SAStoken

**Mount by using a linked service**

1. Assign blob storage contributor role to user and MSI in data lake account
2. Create linked service using account key/Managed Identity
3. Use the mount command

**Mount Via SAS or Account key stored in Key Vault**

1. Create Key Vault
2. Create secret for access key for data lake and store the access key
3. Create access policy in key vault to access secret using synapase workspace
4. Use the Mount command

Transform Data

**Lab 4: Transform GreenTaxiData(**GreenTaxiTripData\_201812.csv)

Data Hub🡪Add new Resource🡪Connect to external data🡪Configure blob storage

GreenTaxi File🡪New Notebook🡪Load to Data Frame

|  |
| --- |
| %%pyspark  blob\_account\_name = "dssdemosa1"  blob\_container\_name = "mycontainer"  from pyspark.sql import SparkSession  sc = SparkSession.builder.getOrCreate()  token\_library = sc.\_jvm.com.microsoft.azure.synapse.tokenlibrary.TokenLibrary  blob\_sas\_token = token\_library.getConnectionString("connblob")  spark.conf.set(      'fs.azure.sas.%s.%s.blob.core.windows.net' % (blob\_container\_name, blob\_account\_name),      blob\_sas\_token)  df = spark.read.load('wasbs://mycontainer@dssdemosa1.blob.core.windows.net/GreenTaxiTripData\_201812.csv', format='csv'  ## If header exists uncomment line below  , header=True,delimiter="\t",inferschema=True  )  display(df.limit(10)) |

Refer:**GreenTaxiTransformation**.ipynb

Working With Spark Table

**Lab 5: Write Transformed data in**

1. Data Lake
2. Managed Table
3. UnManaged Table

|  |
| --- |
| #Data Lake  (df      .write      .partitionBy("TripYear","TripMonth","TripDay")      .mode("overwrite")      .parquet("abfss://taxidata@dssdatalake.dfs.core.windows.net/Facts/GreenTaxidata.parquet")  ) |
| #Managed Table  (df      .write      .partitionBy("TripYear","TripMonth","TripDay")      .mode("overwrite")      .saveAsTable("TaxiDatabase.GreenTaxiManaged")  ) |
| #Unmanaged Table  (df      .write      .partitionBy("TripYear","TripMonth","TripDay")      .mode("overwrite")      .option("path","abfss://taxidata@dssdatalake.dfs.core.windows.net/Facts/GreenTaxidata1.parquet")      .saveAsTable("TaxiDatabase.GreenTaxiUnManaged")  ) |

Access Data From SQLPool using Notebook

* Synapse has built in polybase connector to SQLPool.
* You can read data from Dedicated SQL Pool and save the data in spark table.

**Lab6: Read data from sqlpool and create table spark table.**

|  |
| --- |
| %%spark  val df = spark.read.synapsesql("SQLPool1.main.YellowTaxis\_Hash")  **df.createOrReplaceTempView( "myYellowTaxiTable" )** |
| display(df.limit(100)) |
| val yellowtaxi =df.filter("trip\_distance >10")  display(yellowtaxi) |
| %%sql  **SELECT \* FROM myYellowTaxiTable** |
| **%%pyspark**  **myNewPythonDataFrame = spark.sql("SELECT \* FROM myYellowTaxiTable Where** trip\_distance >10**")** |
| %%spark  spark.sql("CREATE DATABASE IF NOT EXISTS nyctaxi")  df.write.mode("overwrite").saveAsTable("nyctaxi.trip") |

*Note: You can write Data to Dedicated SQL Pool as regular table or external Table*

***Refer: https://docs.microsoft.com/en-us/azure/synapse-analytics/spark/synapse-spark-sql-pool-import-export***