Real-time Inventory Management: Leveraging Azure cloud services.

Apache Spark Streaming with Cosmos DB & Delta Lake

By: Anmol Preet Singh
Programmer Analyst, Cognizant

Summary:

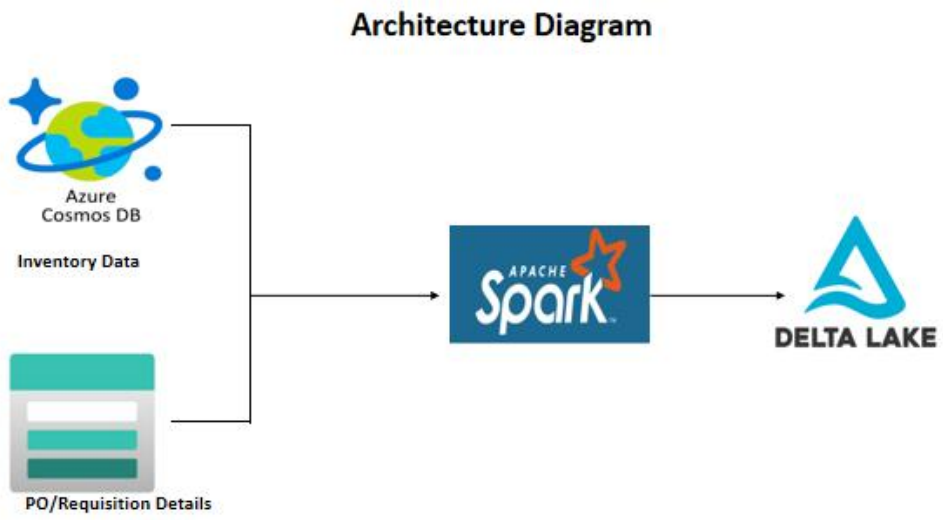
The project aims to harness the capabilities of Azure Synapse Analytics and Azure Cosmos DB to construct a robust Spark Streaming Pipeline. By leveraging Python and SQL with the PySpark package, it seeks to demonstrate the seamless integration of these services for real-time data processing. The core focus lies in implementing window functions—both tumbling and sliding—enhancing data analytics precision. Additionally, the project explores the establishment of joins across disparate data sources, fostering enriched insights. By integrating Logic Apps, the initiative aims to elevate data-driven decision-making through timely email alerts. Ultimately, this endeavor underscores the synergy between Azure's cloud services, emphasizing scalable, low-latency data solutions for modern applications..

Problem statement :

- **Data Deluge:** In today's digital age, organizations are inundated with vast amounts of data from various sources, requiring efficient processing mechanisms.
- **Low Latency Needs:** Traditional data processing methods often lack the speed required for real-time decision-making, leading to delays in insights extraction.
- **Diverse Data Models:** Managing and analyzing data from different models (e.g., document, graph) presents challenges in ensuring consistent and accurate results.
- **Scalability Concerns:** As businesses grow, the ability to scale data processing capabilities becomes critical, necessitating elastic and seamless scaling solutions.

- **Integration Hurdles:** Integrating disparate data sources and services into a unified analytics platform often poses technical challenges, hindering holistic insights generation.
- **Inconsistent Data Access:** Ensuring consistent and fast data access across global regions remains a challenge, impacting user experience and operational efficiency.
- **Complex Queries:** Advanced analytics require intricate query processing, which can be resource-intensive and time-consuming without optimized solutions.
- **Operational Overheads:** Managing and maintaining multiple data services and pipelines increases operational complexities and costs for organizations.
- **Delayed Alerts:** Inadequate mechanisms for timely event detection and alerts result in missed opportunities and potential business risks.
- **Lack of Flexibility:** Conventional analytics platforms may lack the flexibility to adapt to evolving business needs and data processing requirements swiftly.

Architecture:



Technical Details and Implementation of solution

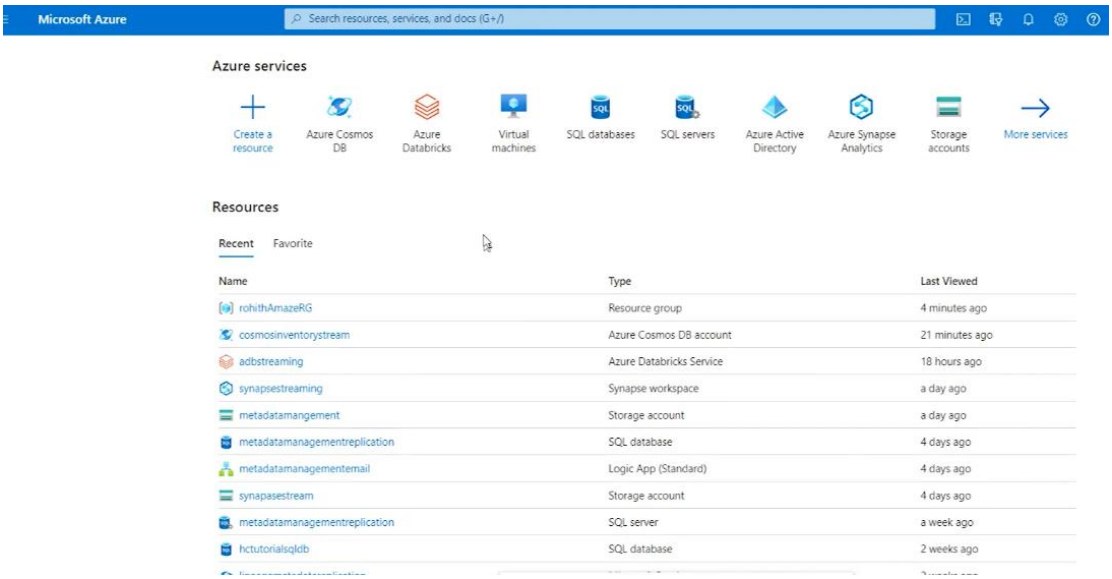
Tech Stack

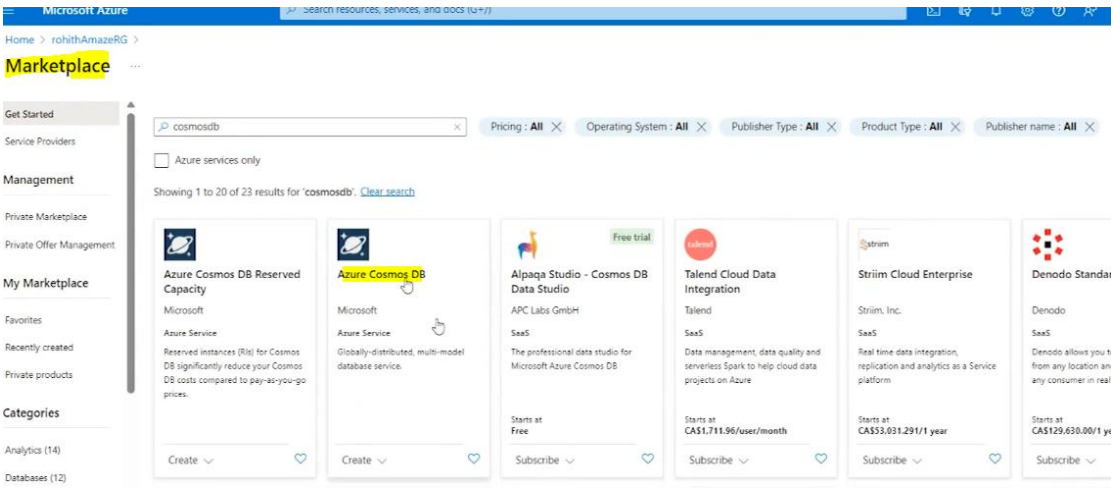
➔ Language: Python, SQL

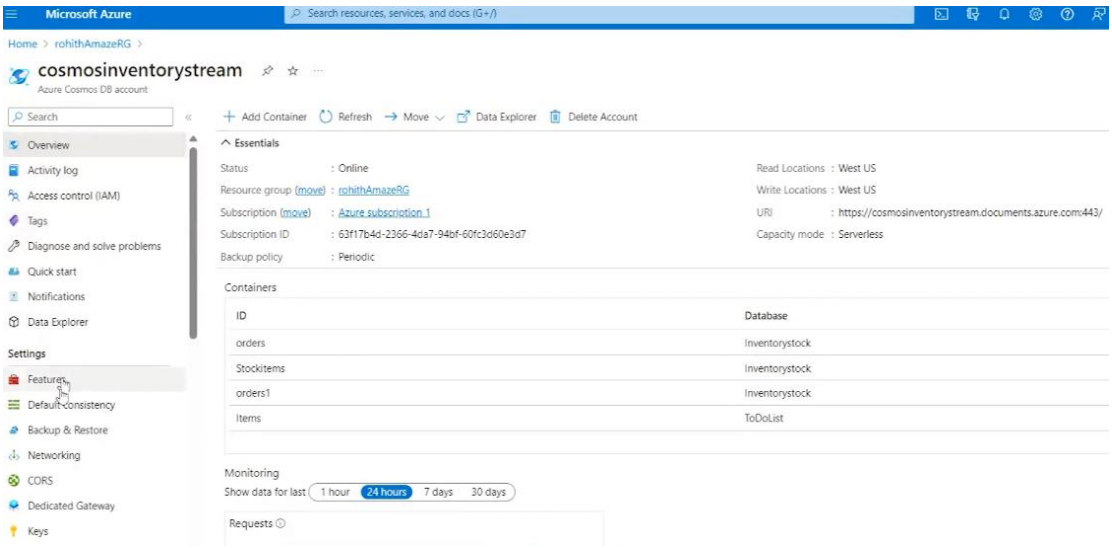
➔ Package: PySpark

➔ Services: Azure Blob Storage (ADLS Gen2), Azure Synapse Analytics, Logic Apps, Azure Cosmos DB

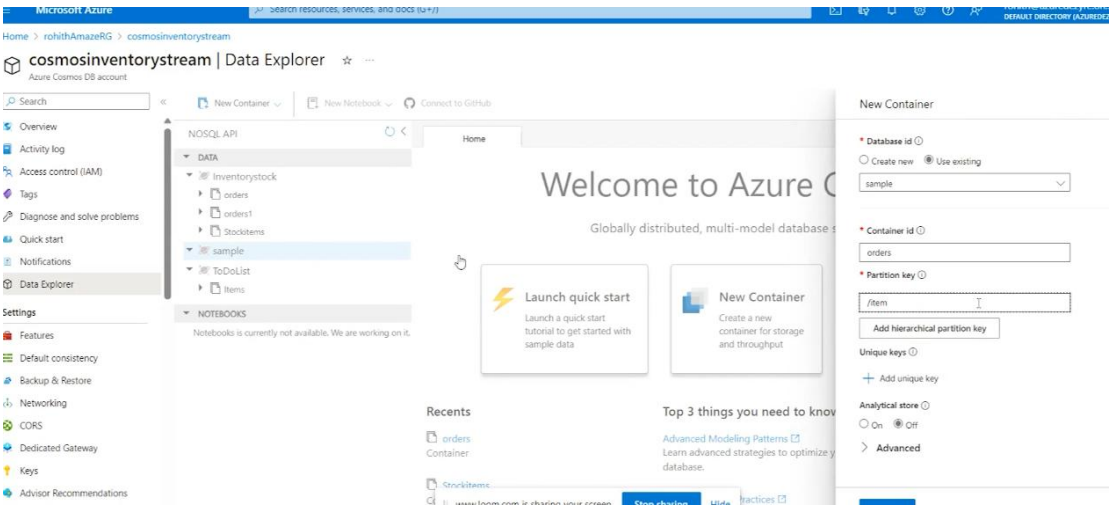
Setting up of Azure account and environment Creating an Azure



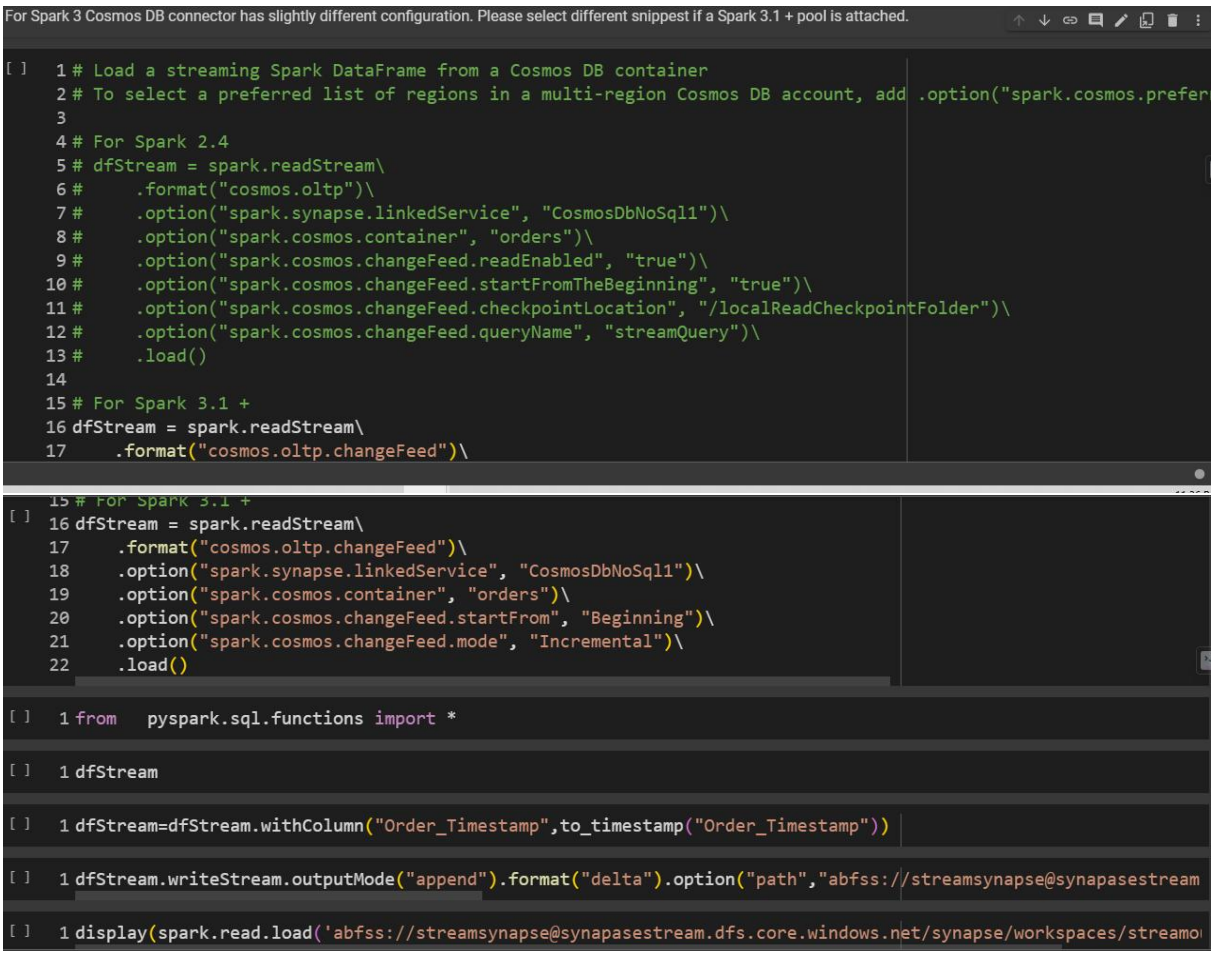




In cosmos every dataset placed in container, So creating container > order >



Whenever new json data inserted to our cosmos db we will streaming and de-streaming to spark job.



Sliding window function :

```
dfStream = spark.readStream\
    .format("cosmos.oltp.changeFeed")\
    .option("spark.synapse.linkedService",
"CosmosDbNoSql1")\
    .option("spark.cosmos.container", "orders")\
    .option("spark.cosmos.changeFeed.startFrom",
"Beginning")\
    .option("spark.cosmos.changeFeed.mode",
"Incremental")\
    .load()
```

```
from   pyspark.sql.functions import *
from pyspark.sql import *
from delta.tables import *
```

```
dfStream=dfStream.withColumn("Order_Date",to_date("Orde
r_Timestamp"))
dfStream=dfStream.withColumn("Order_Timestamp",to_times
tamp("Order_Timestamp"))
```

```
dfStream1=dfStream.withWatermark('Order_Timestamp', '10
minutes').groupBy("Item_Id","Order_Date",window("Order_
Timestamp","8 minutes","4
minutes")).agg(sum("qty")).alias("sum_qty")
```

```
dfStream1=dfStream1.withColumnRenamed("sum(qty)","sum_q
ty")
```

```
def microbatch(batch_df,batch_id):
    delta_table=DeltaTable.forPath(spark,
'abfss://streamsynapse@synapasestream.dfs.core.windows.
net/synapse/workspaces/streamoutput/tumbling_window')
    #print(delta_table)
    delta_table.alias('target').merge(batch_df.alias('u
pdates'),'target.item_id = updates.item_id and
target.window_Start_Time=updates.window_Start_Time and
target.window_end_Time=updates.window_end_Time and
target.order_date=updates.order_date').whenMatchedUpdat
eAll().whenNotMatchedInsertAll().execute()
```

```
dfStream1.writeStream.outputMode("complete").format("de
lta").option("path","abfss://streamsynapse@synapasestre
am.dfs.core.windows.net/synapse/workspaces/streamoutput
/sliding_window").option("checkpointLocation","abfss://
streamsynapse@synapasestream.dfs.core.windows.net/synap
se/workspaces/sliding_window_checkpoint/").start()
```

```
#dfStream1.writeStream.outputMode("complete").foreachBa
tch(microbatch).option("checkpointLocation","abfss://st
reamsynapse@synapasestream.dfs.core.windows.net/synapse
/workspaces/tumb_checkpoint/").start()
```

```
%%sql

select window.start,window.end,a.* from
delta.`abfss://streamsynapse@synapasestream.dfs.core.wi
ndows.net/synapse/workspaces/streamoutput/sliding_windo
w` a where item_id=10 order by 5 desc
```

Spark_Cosmos_Batch_Details_Code.ipyn

```
# Read from Cosmos DB analytical store into a Spark
DataFrame and display 10 rows from the DataFrame
# To select a preferred list of regions in a multi-
region Cosmos DB account,
add .option("spark.cosmos.preferredRegions",
"<Region1>,<Region2>")

df = spark.read\
    .format("cosmos.olap")\
    .option("spark.synapse.linkedService",
"CosmosDbNoSql2")\
    .option("spark.cosmos.container", "orders")\
    .load()
```

```
display(df.limit(10))
```

Stream Joins.ipynb

```
# NOTE

For Spark 3 Cosmos DB connector has slightly different
configuration. Please select different snippest if a
Spark 3.1 + pool is attached.
```

```
# Load a streaming Spark DataFrame from a Cosmos DB
container
# To select a preferred list of regions in a multi-
region Cosmos DB account,
```

```
add .option("spark.cosmos.preferredRegions",
"<Region1>,<Region2>")

# For Spark 2.4
# dfStream = spark.readStream\
#     .format("cosmos.oltp")\
#     .option("spark.synapse.linkedService",
"CosmosDbNoSql1")\
#     .option("spark.cosmos.container", "orders")\
#     .option("spark.cosmos.changeFeed.readEnabled",
"true")\
#     .option("spark.cosmos.changeFeed.startFromTheBegi
nning", "true")\
#     .option("spark.cosmos.changeFeed.checkpointLocati
on", "/localReadCheckpointFolder")\
#     .option("spark.cosmos.changeFeed.queryName",
"streamQuery")\
#     .load()
```

```
# For Spark 3.1 +
dfStream = spark.readStream\
    .format("cosmos.oltp.changeFeed")\
    .option("spark.synapse.linkedService",
"CosmosDbNoSql1")\
    .option("spark.cosmos.container", "orders")\
    .option("spark.cosmos.changeFeed.startFrom",
"Beginning")\
    .option("spark.cosmos.changeFeed.mode",
"Incremental")\
    .load()
```

```
from   pyspark.sql.functions import *
from pyspark.sql import *
from delta.tables import *
dfStream
```

```
dfStream=dfStream.withColumn("Order_Date",to_date("Orde
r_Timestamp"))
dfStream=dfStream.withColumn("Order_Timestamp",to_times
tamp("Order_Timestamp"))
```

```
dfStream
```

```
dfStream1=dfStream.withWatermark('Order_Timestamp', '10
minutes').groupBy("Item_Id","Order_Date",window("Order_
Timestamp","60
minutes")).agg(sum("qty")).alias("sum_qty")
```

```
dfStream1=dfStream1.withColumnRenamed("sum(qty)","sum_q
ty")
```

```
dfStream1=dfStream1.withColumn("window_Start_Time",expr
("window.start"))
dfStream1=dfStream1.withColumn("window_end_Time",expr("
window.end"))
```

```
dfStream1.writeStream.outputMode("complete").format("de
lta").option("path","abfss://streamsynapse@synapasestre
am.dfs.core.windows.net/synapse/workspaces/streamoutput
/Order_stream_output").option("checkpointLocation","abf
ss://streamsynapse@synapasestream.dfs.core.windows.net/
synapse/workspaces/tumb_checkpoint/").start()
```

```
# Load a streaming Spark DataFrame from a Cosmos DB
container
# To select a preferred list of regions in a multi-
region Cosmos DB account,
add .option("spark.cosmos.preferredRegions",
"<Region1>,<Region2>")

# For Spark 2.4
# dfStream = spark.readStream\
```

```
#     .format("cosmos.oltp")\
#     .option("spark.synapse.linkedService",
"CosmosDbNoSql1")\
#     .option("spark.cosmos.container", "orders")\
#     .option("spark.cosmos.changeFeed.readEnabled",
"true")\
#     .option("spark.cosmos.changeFeed.startFromTheBegi
nning", "true")\
#     .option("spark.cosmos.changeFeed.checkpointLocati
on", "/localReadCheckpointFolder")\
#     .option("spark.cosmos.changeFeed.queryName",
"streamQuery")\
#     .load()
```

```
# For Spark 3.1 +
dfStream2 = spark.readStream\
    .format("cosmos.oltp.changeFeed")\
    .option("spark.synapse.linkedService",
"CosmosDbNoSql1")\
    .option("spark.cosmos.container", "Stockitems")\
    .option("spark.cosmos.changeFeed.startFrom",
"Beginning")\
    .option("spark.cosmos.changeFeed.mode",
"Incremental")\
    .load()
```

```
1 dfStream1
```

```
1 dfStream2
```

```
1 dfStream2=dfStream2.withColumnRenamed("item_id","item_id_refill")
```

```
1 dfStream2=dfStream2.withColumn("refill_timestamp",to_timestamp("refill_timestamp"))
```

```
1 dfStream2=dfStream2.withColumn("refill_date",to_date("refill_timestamp"))
```

```
1 dfStream2
```

```
1 dfStream2=dfStream2.withWatermark('refill_timestamp', '10 minutes').groupBy("Item_Id_refill","refill_date",wi
```

```
1 dfStream2=dfStream2.withColumn("stock_start_window",expr("window.start"))
2 dfStream2=dfStream2.withColumn("stock_end_window",expr("window.end"))
```

```
dfStream2.writeStream.outputMode("complete").format("delta").option("path","abfss://streamsynapse@synapasestr
```

```
dfStream4=spark.readStream\
    .format("delta")\
    .option("path", "abfss://streamsynapse@synapasestream.dfs.core.windows.net/synapse/workspaces/streamoutpu
    .load()
```

```
dfStream5=spark.readStream\
    .format("delta")\
    .option("path", "abfss://streamsynapse@synapasestream.dfs.core.windows.net/synapse/workspaces/streamoutpu
    .load()
```

```
dfStream6=dfStream5.join(dfStream4,expr("Item_Id_refill=item_id and refill_date= Order_Date and stock_start_w
```

```
def microbatch(batch_df,batch_id):
    url=""
    myobj=""
    df1=batch_df.withColumn("difference_quantity",expr(
"sum_stock_Refill_qty-sum_qty"))
    if
df1.filter(df1.difference_quantity<100).count()>1:
        url =
'https://metadatamanagementemail.azurewebsites.net:443/
api/sample_email/triggers/When_a_HTTP_request_is_receiv
ed/invoke?api-version=2022-05-
01&sp=%2Ftriggers%2FWhen_a_HTTP_request_is_received%2Fr
un&sv=1.0&sig=Lz1K7ZusrhyeY1lxp9zV12sPUJUEHxlHVH1HTwL0M
MA'
        myobj = {
            "emailaddress":"rohith.amaze@gmail.com",
            "body":"triggered from python Databricks
stream",
            "subject":"Low Stock Alert"
        }
        trigger_email(url,myobj)
```

```
def trigger_email(url,myobj):
    x = requests.post(url=url, json = myobj)
```

```
dfStream6.writeStream.outputMode("append").foreachBatch
(microbatch).option("checkpointLocation","/mnt/streamsy
```

```
napse/synapse/workspaces/join/join_output_chkpoint/").start()
```

Window_Functions.ipynb

```
# NOTE

For Spark 3 Cosmos DB connector has slightly different configuration. Please select different snippest if a Spark 3.1 + pool is attached.
```

```
# Load a streaming Spark DataFrame from a Cosmos DB container
# To select a preferred list of regions in a multi-region Cosmos DB account,
add .option("spark.cosmos.preferredRegions", "<Region1>,<Region2>")

# For Spark 2.4
# dfStream = spark.readStream\
#     .format("cosmos.oltp")\
#     .option("spark.synapse.linkedService", "CosmosDbNoSql1")\
#     .option("spark.cosmos.container", "orders")\
#     .option("spark.cosmos.changeFeed.readEnabled", "true")\
#     .option("spark.cosmos.changeFeed.startFromTheBeginning", "true")\
#     .option("spark.cosmos.changeFeed.checkpointLocation", "/localReadCheckpointFolder")\
#     .option("spark.cosmos.changeFeed.queryName", "streamQuery")\
#     .load()
```

```
# For Spark 3.1 +
dfStream = spark.readStream\
    .format("cosmos.oltp.changeFeed")\
```

```
    .option("spark.synapse.linkedService",
"CosmosDbNoSql1")\
    .option("spark.cosmos.container", "orders")\
    .option("spark.cosmos.changeFeed.startFrom",
"Beginning")\
    .option("spark.cosmos.changeFeed.mode",
"Incremental")\
    .load()
from   pyspark.sql.functions import *
from pyspark.sql import *
from delta.tables import *
```

```
dfStream
```

```
dfStream=dfStream.withColumn("Order_Date",to_date("Orde
r_Timestamp"))
```

```
dfStream=dfStream.withColumn("Order_Timestamp",to_times
tamp("Order_Timestamp"))
```

```
dfStream
```

```
dfStream1=dfStream.withWatermark('Order_Timestamp', '10
minutes').groupBy("Item_Id","Order_Date",window("Order_
Timestamp","6
minutes")).agg(sum("qty")).alias("sum_qty")
```

```
dfStream1=dfStream1.withColumnRenamed("sum(qty)","sum_q
ty")
```

```
dfStream1=dfStream1.withColumn("window_Start_Time",expr
("window.start"))
dfStream1=dfStream1.withColumn("window_end_Time",expr("
window.end"))
```

```
def microbatch(batch_df,batch_id):
```

```
    delta_table=DeltaTable.forPath(spark,
'abfss://streamsynapse@synapasestream.dfs.core.windows.
net/synapse/workspaces/streamoutput/tumbling_window')
    #print(delta_table)
    delta_table.alias('target').merge(batch_df.alias('u
pdates'),'target.item_id = updates.item_id and
target.window_Start_Time=updates.window_Start_Time and
target.window_end_Time=updates.window_end_Time and
target.order_date=updates.order_date').whenMatchedUpdat
eAll().whenNotMatchedInsertAll().execute()
```

```
#dfStream1.writeStream.outputMode("complete").format("d
elta").option("path","abfss://streamsynapse@synapasestr
eam.dfs.core.windows.net/synapse/workspaces/streamoutpu
t/tumbling_window").option("checkpointLocation","abfss:
//streamsynapse@synapasestream.dfs.core.windows.net/syn
apse/workspaces/tumb_checkpoint/").start()
```

```
dfStream1.writeStream.outputMode("complete").foreachBat
ch(microbatch).option("checkpointLocation","abfss://str
eamsynapse@synapasestream.dfs.core.windows.net/synapse/
workspaces/tumb_checkpoint/").start()
```

```
%%sql

select
item_id,order_date,WINDOW_start_time,window_end_time,su
m_qty from
delta.`abfss://streamsynapse@synapasestream.dfs.core.wi
ndows.net/synapse/workspaces/streamoutput/tumbling_wind
ow` order by WINDOW_start_time desc;
```

Challenges in implementing the solution :

- **Complex Integration:** Combining Azure Synapse Analytics and Cosmos DB requires intricate configuration and synchronization, posing initial setup challenges.
- **Data Consistency:** Ensuring consistent data across distributed Cosmos DB instances while leveraging global distribution features can be technically demanding.
- **Window Function Complexity:** Implementing and optimizing both tumbling and sliding window functions necessitates a deep understanding of data partitioning and processing nuances.
- **Scalability Considerations:** Adapting the pipeline to handle increasing data volumes without compromising performance demands meticulous planning and resource allocation.
- **API Compatibility:** Utilizing Cosmos DB's multi-API support effectively requires aligning API-specific functionalities with Spark Streaming requirements, introducing potential compatibility issues.
- **Performance Tuning:** Achieving optimal performance in real-time analytics scenarios requires fine-tuning Spark Streaming configurations, which can be iterative and time-consuming.
- **Data Security:** Ensuring robust data encryption, access control, and compliance with regulatory standards across Azure services presents ongoing security challenges.
- **Error Handling:** Implementing comprehensive error handling mechanisms to address potential failures in data ingestion, processing, or transmission is essential but can introduce complexity.
- **Cost Management :** Balancing the cost-effectiveness of utilizing multiple Azure services, including storage, analytics, and database solutions, requires continuous monitoring and optimization.
- **Skill Set Requirements:** Navigating the diverse tech stack, including Python, SQL, PySpark, and Azure services, necessitates a multidisciplinary skill set and ongoing knowledge acquisition.

Business Benefits:

- **Real-time Insights:** Uncover actionable insights instantly, empowering proactive decision-making.
- **Scalable Architecture:** Seamlessly adapt to business growth with elastic data processing capabilities.
- **Global Reach:** Ensure low-latency data access across regions, enhancing user experience worldwide.
- **Cost Efficiency:** Optimize operational costs through streamlined data pipelines and resource utilization.
- **Enhanced Agility:** Rapidly adapt to market changes with flexible analytics and data processing workflows.
- **Unified Analytics:** Consolidate diverse data sources into a single, cohesive analytics platform for holistic insights.
- **Compliance Assurance:** Safeguard data integrity and meet regulatory requirements with robust security features.
- **Operational Excellence:** Boost efficiency by automating complex tasks and reducing manual interventions.
- **Customer Engagement:** Personalize user experiences and offerings based on real-time analytics and trends.
- **Innovative Growth:** Foster innovation by leveraging cutting-edge technologies for advanced data analytics and processing.