

Azure Data Engineer Associate Certification Guide

A hands-on reference guide to developing your data engineering skills and preparing for the DP-203 exam

Newton Alex



Capítulo 14: Optimización y resolución de problemas de almacenamiento y procesamiento de datos	4
14.1. Requisitos técnicos.....	4
14.2. Compactación de archivos pequeños	5
14.3. Reescribiendo funciones definidas por el usuario (UDFs)	9
14.3.1. Escribir UDFs en Synapse SQL Pool	9
14.3.2. Escribir UDFs en Spark.....	11
14.3.3. Escribir UDFs en Stream Analytics.....	12
14.4. Manejo de sesgos (skews) en los datos	14
14.4.1. Corregir los sesgos a nivel de storage	15
14.4.2. Corregir los sesgos en el nivel de computo.....	15
14.5. Manejar los derrames de datos (data spills)	17
14.5.1. Identificación de derrames de datos en Synapse SQL	17
14.5.2. Identificación de derrames de datos en Spark.....	18
14.6. Ajuste de las particiones shuffle	19
14.7. Encontrando el shuffling en un pipeline	20
14.7.1. Identificar shuffles en un plan de consulta SQL	20
14.7.2. Identificar shuffles en un plan de consulta de Spark	21
14.8. Optimización de la gestión de recursos	23
14.8.1. Optimización de los Synapse SQL pools	23
14.8.2. Optimización de Spark	23
14.9. Ajuste de las consultas mediante el uso de indexadores.....	24
14.9.1. Indexación en Synapse SQL.....	24
14.9.2. Indexación en el Synapse Spark pool usando Hyperspace	25
14.10. Ajuste de las consultas mediante el uso de la caché	27
14.11. Optimización de pipelines para fines analíticos o transaccionales.....	28
14.11.1. Sistemas OLTP	28
14.11.2. Sistemas OLAP	28
14.11.3. Implementación de HTAP utilizando Synapse Link y CosmosDB	29
Introduciendo CosmosDB.....	29
Introduciendo Azure Synapse Link	29
14.12. Optimización de los pipelines para cargas de trabajo descriptivas frente a las analíticas	37
14.12.1. Optimizaciones comunes para pipelines descriptivos y analíticos	38

14.12.2. Optimizaciones específicas para pipelines descriptivos y analíticos	38
14.13. Solución de problemas de un Spark job fallido	40
14.13.1. Depuración de los problemas del entorno.....	40
14.13.2. Depuración de los problemas de los jobs	41
14.14. Solución de problemas de la ejecución de un pipeline fallido.....	44
Resumen.....	47

Capítulo 14: Optimización y resolución de problemas de almacenamiento y procesamiento de datos

Bienvenido al último capítulo de la sección de Supervisión y optimización del almacenamiento y el procesamiento de datos del programa de estudios. El único capítulo que queda después de éste es la revisión y las preguntas de muestra para la certificación. Enhorabuena por haber llegado hasta aquí; ahora está a un paso de obtener la certificación.

En este capítulo, nos centraremos en las técnicas de optimización y resolución de problemas de las tecnologías de almacenamiento y procesamiento de datos. Comenzaremos con los temas para optimizar las consultas Spark y Synapse SQL utilizando técnicas como la compactación de archivos pequeños, el manejo de UDFs, la inclinación de los datos (data skews), los shuffles, la indexación, la gestión de la caché, y más. También veremos las técnicas para la resolución de problemas de los pipelines de Spark y Synapse y las directrices generales para la optimización de cualquier pipeline analítico. Una vez que haya completado este capítulo, tendrá los conocimientos necesarios para depurar los problemas de rendimiento o solucionar los fallos en los pipelines y los Spark jobs.

En este capítulo cubriremos los siguientes temas:

- ❖ Compactación de archivos pequeños
- ❖ Reescritura de funciones definidas por el usuario (UDFs)
- ❖ Manejar los sesgos de los datos (skews in data)
- ❖ Manejo de derrames de datos (data spills)
- ❖ Ajuste de las particiones aleatorias (shuffle partitions)
- ❖ Búsqueda de barajados en un pipeline (shuffling in a pipeline)
- ❖ Optimización de la gestión de recursos
- ❖ Ajuste de las consultas mediante el uso de indexadores
- ❖ Ajuste de las consultas mediante el uso de la caché
- ❖ Optimización de pipelines para fines analíticos o transaccionales
- ❖ Optimización de pipelines para cargas de trabajo descriptivas versus analíticas
- ❖ Resolución de problemas de un Spark job fallido
- ❖ Resolución de problemas de una ejecución de pipeline fallida

14.1. Requisitos técnicos

Para este capítulo, necesitará lo siguiente:

- ❖ Una cuenta de Azure (gratuita o de pago)
- ❖ Un workspace Azure Data Factory o Synapse activo
- ❖ Un workspace Azure Databricks activo

¡Empecemos!

14.2. Compactación de archivos pequeños

Los archivos pequeños son la pesadilla de los sistemas de procesamiento de big data. Los motores analíticos como Spark, Synapse SQL y Hive, y los sistemas de almacenamiento en la nube como Blob y ADLS Gen2, están todos intrínsecamente optimizados para archivos grandes. Por lo tanto, para hacer eficientes nuestros pipelines de datos, es mejor fusionar (merge) o compactar los archivos pequeños en otros más grandes. Esto se puede lograr en Azure utilizando Azure Data Factory y Synapse Pipelines. Veamos un ejemplo utilizando Azure Data Factory para concatenar un montón de pequeños archivos CSV en un directorio en un archivo grande. Los pasos para los pipelines de Synapse serán muy similares:

1. En el portal de Azure Data Factory, seleccione la actividad Copiar datos como se muestra en la siguiente captura de pantalla. En la pestaña Origen, elija un conjunto de datos de origen existente o cree uno nuevo, apuntando al almacenamiento de datos donde están los archivos pequeños. A continuación, elija la opción de ruta de archivo comodín para el tipo de ruta de archivo. En el campo Rutas con comodines, proporcione una ruta de carpeta que termine con un *. Esto asegurará que la actividad de Copiar datos considere todos los archivos de esa carpeta. En la siguiente captura de pantalla, todos los archivos pequeños están presentes en la carpeta denominada **sandbox/driver/in**.

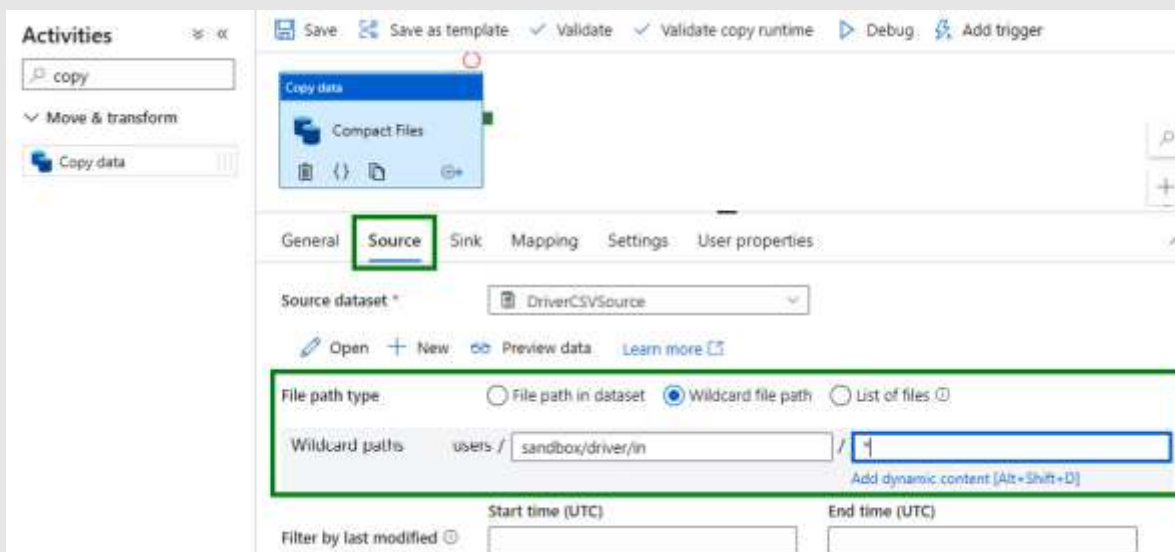


Figura 14.1 - Especificación de la carpeta de origen mediante comodines

A continuación, en la pestaña Sink, seleccione su conjunto de datos de destino y, para el comportamiento de copia, seleccione la opción Merge files, como se muestra en la siguiente captura de pantalla:

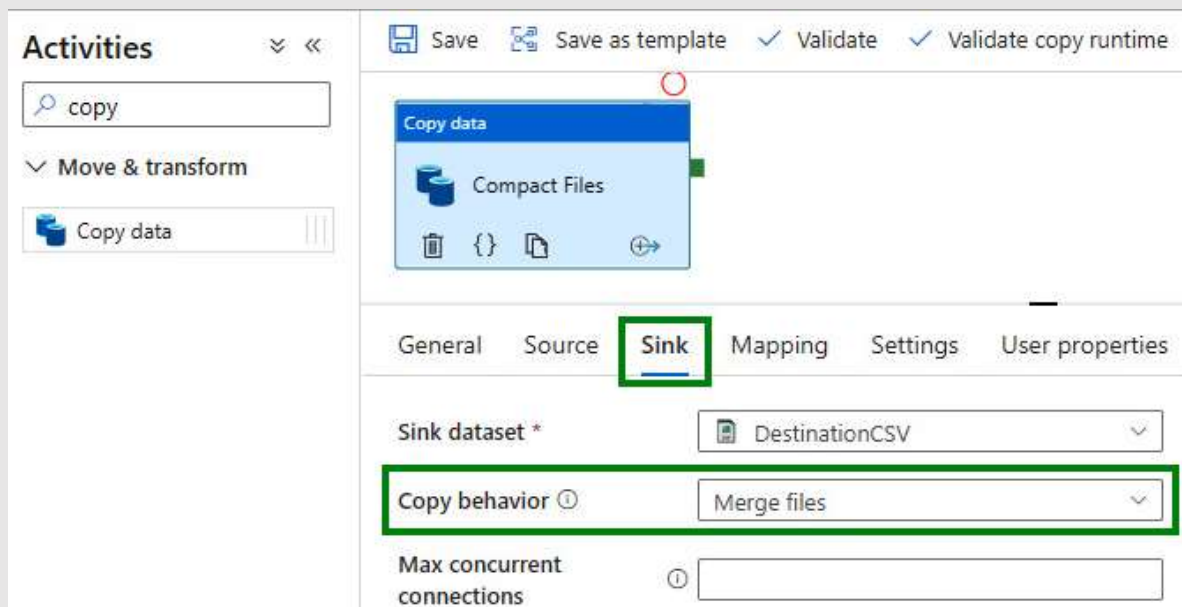


Figura 14.2 - Seleccionando el comportamiento de copia Merge files en Copy Activity del ADF

Ahora, si ejecuta este pipeline, los archivos de la carpeta de origen (archivos CSV en nuestro caso) se fusionarán en un archivo llamado Output.csv en la carpeta de destino que se define en el conjunto de datos DestinationCSV.

Puede obtener más información sobre la opción de fusión de la actividad de copia en el ADF aquí: <https://docs.microsoft.com/en-us/azure/data-factory/connector-file-system?tabs=data-factory#copy-activity-properties>.

Otra forma de compactar archivos pequeños es el método de copia incremental que ya hemos explorado en el Capítulo 4, Diseño de la Serving Layer, en la sección Diseño para la carga incremental. En esta opción, actualizamos incrementalmente las tablas con pequeños cambios entrantes. Esto también se calificaría como una opción de compactación, ya que estamos fusionando incrementalmente archivos pequeños en uno grande a través de tablas SQL.

Hay una tercera forma de compactar archivos pequeños utilizando Azure Databricks Spark y Synapse Spark. Spark proporciona una característica llamada bin packing, a través de su capa de almacenamiento (storage layer) especializada llamada Delta Lake (nótese la ortografía, no Data Lake).

Un Delta Lake es una capa de almacenamiento de código abierto que se ejecuta sobre los Data Lakes. Mejora el Data Lake para soportar características como las siguientes:

- ❖ **Transacciones ACID:** Al igual que en las bases de datos y los data warehouses, Delta Lake permite realizar transacciones ACID sobre los Data Lakes.
- ❖ **Sistema unificado de procesamiento por lotes, interactivo y de streaming:** Las tablas definidas en Delta Lakes pueden servir como fuente para sistemas de procesamiento por lotes, sistemas interactivos como notebooks, y sistemas de streaming.

- ❖ **Updates y Deletes:** Esta función admite actualizaciones, eliminaciones y fusiones (merges) de tablas. Esto permite el soporte automático de Dimensiones de Cambio Lento (SCDs), upserts en streaming, y cualquier otra operación que requiera la capacidad de actualizar o fusionar (merge) datos.

El motor Delta Lake está habilitado por defecto en Azure Databricks y Synapse Spark. Por lo tanto, todo lo que tienes que hacer es conectar tu notebook Spark a cualquier clúster Databricks o Synapse Spark y ejecutar los comandos Delta. Veamos algunos comandos de ejemplo de cómo leer, escribir y crear tablas en Delta Lake:

- ❖ Podemos escribir en el almacenamiento Azure Blob en formato delta como se muestra. abfss en el siguiente código se refiere al protocolo de almacenamiento de Azure: Azure Blob File System [Secure]:

```
df.write.mode("overwrite").format("delta").save("abfss://path/to/delta/files")
```

- ❖ Cargar los datos del archivo delta, como se muestra aquí:

```
Val df: DataFrame = spark.read.format("delta").load(abfss://path/to/delta/files)
```

- ❖ Crear una tabla con delta, como se muestra aquí:

```
Spark.sql("CREATE TABLE CUSTOMER USING DELTA LOCATION 'abfss://path/to/delta/files'")
```

Como habrás notado, trabajar con Delta Lake es muy similar a trabajar con formatos de archivo como Parquet, JSON o CSV.

Puede obtener más información sobre Delta Lake aquí: <https://docs.microsoft.com/en-us/azure/databricks/delta/>.

Ahora vamos a ver cómo habilitar el bin packing. Bin packing es la función que ofrece Delta Lake para compactar archivos pequeños en otros más grandes para mejorar el rendimiento de las consultas de lectura. Podemos ejecutar el bin packing en una carpeta utilizando el comando OPTIMIZE, como se muestra en el siguiente bloque de código.

```
Spark.sql("OPTIMIZE delta.' abfss://path/to/delta/files'")
```

Esto fusionará (merge) los archivos pequeños de la carpeta en archivos grandes óptimos.

La tercera opción es activar la optimización por defecto al crear las tablas utilizando las siguientes propiedades

- ❖ `delta.autoOptimize.optimizeWrite = true`
- ❖ `delta.autoOptimize.autoCompact = true`

A continuación se muestra un ejemplo de cómo utilizar las propiedades de optimización al crear una tabla:

```
CREATE TABLE Customer (  
  id INT,  
  name STRING,  
  location STRING  
) TBLPROPERTIES (  
  delta.autoOptimize.optimizeWrite = true,  
  delta.autoOptimize.autoCompact = true  
)
```

Puede obtener más información sobre el empaquetado de contenedores (bin packing) de Azure Databricks aquí: <https://docs.microsoft.com/en-us/azure/databricks/delta/optimizations/file-mgmt>.

Ahora ya conoces tres formas diferentes de compactar archivos pequeños. A continuación, vamos a ver el uso de funciones definidas por el usuario.

14.3. Reescribiendo funciones definidas por el usuario (UDFs)

Las funciones definidas por el usuario son funciones personalizadas que pueden definirse en bases de datos y en ciertos motores analíticos y de streaming como Spark y Azure Stream Analytics. Un ejemplo de una UDF podría ser una función personalizada para comprobar si un valor dado es una dirección de correo electrónico válida.

El programa de estudios del DP-203 sólo menciona el tema como Reescritura de funciones definidas por el usuario. En un sentido literal, se trata de eliminar una función definida por el usuario y volver a crear una nueva, como hacemos con las tablas de SQL o Spark. Sin embargo, creo que el comité del programa de estudios podría haberse referido a la reescritura de secuencias de comandos repetitivas normales como UDF para hacerlas eficientes desde una perspectiva de desarrollo. Hay que tener en cuenta que los UDFs también pueden disminuir el rendimiento en tiempo de ejecución (runtime) si no se diseñan correctamente. Veamos las formas de crear UDFs en SQL, Spark y Streaming.

14.3.1. Escribir UDFs en Synapse SQL Pool

Usted puede crear una función definida por el usuario en Synapse SQL usando el comando CREATE FUNCTION en Synapse SQL. A continuación se muestra un ejemplo de uso del comando CREATE FUNCTION para crear una simple función definida por el usuario de validación de correo electrónico. El script comprueba el patrón de correo electrónico y, si no es válido, devuelve la cadena "No disponible":

```
CREATE FUNCTION dbo.isValidEmail(@EMAIL VARCHAR(100))
RETURNS VARCHAR(100) AS
BEGIN
    DECLARE @returnValue AS VARCHAR(100)
    DECLARE @EmailText VARCHAR(100)
    SET @EmailText= isnull(@EMAIL, '')
    SET @returnValue = CASE WHEN @EmailText NOT LIKE '_%@_%._%' THEN 'Not Available'
                           ELSE @EmailText
    end
    RETURN @returnValue
END
```

Ahora que tenemos el UDF, veamos cómo utilizarlo en una consulta. Supongamos que tiene una tabla de ejemplo con algunos ID de correo electrónico válidos e inválidos.

```

30
31 -- View the rows in the table
32 SELECT * FROM dbo.CustomerContact;
33

```

Results Messages

View Table Chart Export results

Search

CustomerID	Name	Email
2	Bran	bryan@domain.com
4	Demin	demin@wrongdomain
1	Arielle	arielle
3	Cathy	cathy@domain.com
5	Ethan	ethan@domain.com

Figura 14.3 - Tabla de ejemplo con IDs de correo electrónico válidos e inválidos

Queremos marcar las filas con identificadores de correo electrónico no válidos como No disponibles. Podemos conseguirlo con la siguiente consulta.

```

35 -- Try to use the UDF
36 SELECT CustomerID, Name, dbo.isValidEmail>Email) AS Email FROM dbo.CustomerContact;
37

```

Results Messages

View Table Chart Export results

Search

CustomerID	Name	Email
2	Bran	bryan@domain.com
4	Demin	Not Available
1	Arielle	Not Available
3	Cathy	cathy@domain.com
5	Ethan	ethan@domain.com

Figura 14.4 - Uso de UDFs en una consulta

Como se esperaba, las filas con correos electrónicos no válidos fueron sustituidas por la cadena No Disponible.

Puedes aprender más sobre los UDFs en Synapse SQL aquí: <https://docs.microsoft.com/en-us/sql/t-sql/statements/create-function-sql-data-warehouse>.

A continuación, vamos a ver cómo escribir UDFs en Spark.

14.3.2. Escribir UDFs en Spark

Spark también soporta la función UDF. En Spark, las UDFs son como cualquier función regular. He aquí un ejemplo sencillo de registro de una UDF en Spark:

```
1 import org.apache.spark.sql.functions.{col, udf}
2 val double = udf((s: Long) => 2 * s)
3 display(spark.range(1, 20).select(double(col("id")) as "doubled"))
4
```

✓ 1 sec - Command executed in 1 sec 802 ms by newton.packt on 12:21:13 PM, 11/14/21

> **Job execution** Succeeded **Spark** 1 executors 4 cores [View in monitoring](#) [Open Spark UI](#)

```
import org.apache.spark.sql.functions.{col, udf}
double: org.apache.spark.sql.expressions.UserDefinedFunction =
UserDefinedFunction(<function1>,LongType,Some(List(LongType)))
```

View **Table** Chart [Export results](#)

doubled
2
4
6

Figura 14.5 - Ejemplo simple de UDF en Spark

El script de ejemplo simplemente duplica cualquier valor que se le dé. Puede definir UDFs para reducir la repetición de código.

Puedes aprender más sobre los UDFs de Spark aquí: <https://spark.apache.org/docs/latest/sql-ref-functions-udf-scalar.html>.

A continuación, veamos cómo escribir UDFs en Azure Stream Analytics.

14.3.3. Escribir UDFs en Stream Analytics

Azure Stream Analytics soporta UDFs simples a través de JavaScript. Los UDFs no tienen estado y sólo pueden devolver valores simples (escalares). A diferencia de los ejemplos de SQL y Spark, en Stream Analytics tenemos que registrar el UDF desde el portal antes de utilizarlo. Veamos un ejemplo:

1. Aquí tenemos una definición de UDF para extraer customer.id de un blob JSON de entrada:

```
function main(arg) {  
    var customer = JSON.parse(arg);  
    return customer.id;  
}
```

2. A continuación, tenemos que registrar el UDF. En la página de Stream Analytics job, seleccione Functions debajo de Job topology. Haga clic en el enlace + Add y seleccione Javascript UDF, como se muestra en la siguiente captura de pantalla:

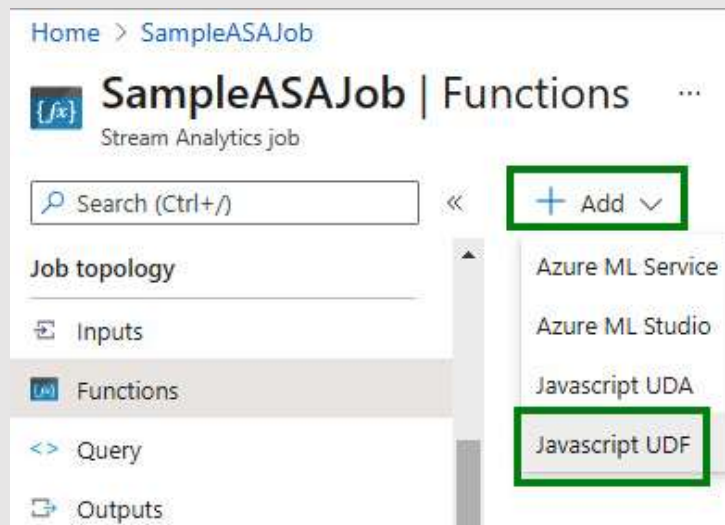


Figura 14.6 - Añadir UDFs en Azure Stream Analytics

3. Una vez que haga clic en la opción Javascript UDF, aparecerá una pantalla como la que se muestra en la siguiente captura de pantalla, donde puede rellenar el código UDF y proporcionar un nombre para la UDF en el campo Alias de función. Haga clic en Guardar para registrar la nueva UDF.

Home > SampleASAJob >

Javascript function

New function

Function alias *

getID ✓

Output type ⓘ

any ▼

```
1 function main(arg) {  
2   var customer = JSON.parse(arg);  
3   return customer.id;  
4 }  
5
```

Save

Figura 14.7 - Definición de la UDF en el portal de ASA

Una vez registrado el UDF, puede llamarlo en sus consultas de streaming, como se muestra aquí:

```
SELECT UDF.getID(input) AS Id  
INTO Stream_output  
FROM Stream_input
```

Puede obtener más información sobre el UDF de análisis de streams aquí:
<https://docs.microsoft.com/en-us/azure/stream-analytics/functions-overview>.

Ahora ya sabes cómo definir UDFs en SQL, Spark y Streaming. Veamos ahora cómo manejar las inclinaciones de los datos.

14.4. Manejo de sesgos (skews) en los datos

Data skew se refiere a una distribución extrema y desigual de los datos en un conjunto de datos. Tomemos como ejemplo el número de viajes por mes de nuestro ejemplo del taxi del aeropuerto imaginario (IAC). Supongamos que la distribución de los datos es la que se muestra en el siguiente gráfico:

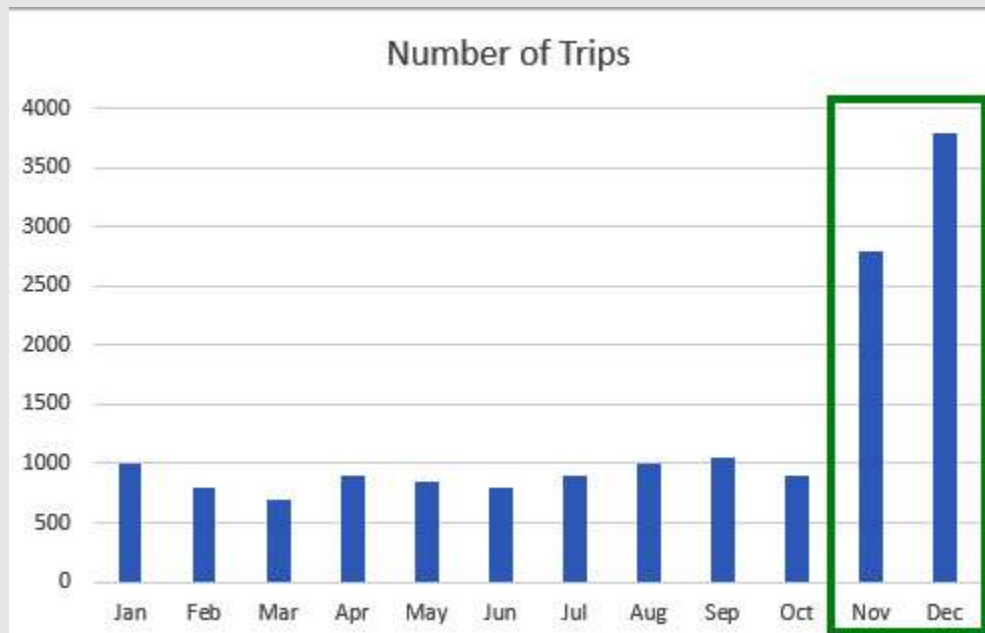


Figura 14.8 - Un ejemplo de datos sesgados

Como se puede ver en el gráfico, los números de viajes de noviembre y diciembre son bastante altos en comparación con los demás meses. Esta distribución desigual de los datos se denomina sesgo de los datos (data skew). Ahora bien, si distribuyéramos los datos mensuales a nodos de cálculo individuales, los nodos que procesan los datos de noviembre y diciembre van a tardar mucho más que los que procesan los demás meses. Y si estuviéramos generando un informe anual, todas las demás etapas tendrían que esperar a que las de noviembre y diciembre se completaran. Estos tiempos de espera son ineficientes para el rendimiento del trabajo. Para que el procesamiento sea más eficiente, tendremos que encontrar una forma de asignar cantidades similares de datos de procesamiento a cada uno de los nodos de cómputo. Exploraremos algunas opciones recomendadas por Azure para manejar tales desviaciones de datos.

Los sesgos pueden arreglarse tanto a nivel de almacenamiento como a nivel de cómputo. En función de dónde solucionemos el problema, hay diferentes opciones disponibles.

14.4.1. Corregir los sesgos a nivel de storage

A continuación se describen algunas técnicas para corregir los sesgos a nivel de almacenamiento:

- ❖ Encontrar una mejor estrategia de distribución/partición que equilibre el tiempo de cálculo de manera uniforme. En nuestro ejemplo del recuento mensual de viajes, podríamos explorar la partición de los datos en trozos más pequeños a nivel semanal o intentar la partición a lo largo de una dimensión diferente, como los códigos postales en conjunto, y ver si eso ayuda.
- ❖ Añadir una segunda clave de distribución. Si la clave principal no está dividiendo los datos de manera uniforme, y si la opción de pasar a una nueva clave de distribución no es posible, podría añadir una clave de partición secundaria. Por ejemplo, después de dividir los datos en meses, puede dividirlos aún más en, por ejemplo, estados del país dentro de cada mes. De este modo, si usted se encuentra en EE.UU., obtendrá 50 divisiones más, que podrían estar distribuidas de forma más uniforme.
- ❖ Aleatorice los datos y utilice la técnica de round-robin para distribuir los datos uniformemente en particiones. Si no puede encontrar una clave de distribución óptima, puede recurrir a la distribución round-robin. De este modo, se asegurará de que los datos se distribuyan uniformemente.

Tenga en cuenta que no siempre es posible volver a crear tablas o distribuciones. Por lo tanto, es muy importante hacer una planificación previa cuando decidimos la estrategia de partición. Sin embargo, si terminamos en una situación en la que las estrategias de partición no ayudan, todavía nos puede quedar una opción más para mejorar nuestro manejo del sesgo. Esta opción es confiar en que nuestro motor de cálculo produzca un plan de consulta inteligente que sea consciente del sesgo de los datos. Veamos cómo conseguirlo a continuación.

14.4.2. Corregir los sesgos en el nivel de computo

A continuación se describen algunas técnicas para corregir los sesgos a nivel de computación:

- ❖ **Mejorar el plan de consulta habilitando las estadísticas.** Ya hemos visto cómo habilitar las estadísticas en el Capítulo 13, Supervisión del almacenamiento y el procesamiento de datos, en la sección Supervisión y actualización de las estadísticas sobre los datos en un sistema. Una vez que habilitamos las estadísticas, los motores de consulta como el motor Synapse SQL, que utiliza un optimizador basado en el costo, utilizará las estadísticas para generar el plan más óptimo basado en el costo asociado a cada plan. El optimizador puede identificar los datos sesgados y aplicar automáticamente las optimizaciones apropiadas para manejar los sesgos.
- ❖ **Ignorar los datos atípicos si no son significativos.** Esta es probablemente la más sencilla de las opciones, pero puede no ser aplicable a todas las situaciones. Si los datos sesgados no son muy importantes, puede ignorarlos con seguridad.

Mientras estamos en el tema de la gestión de los sesgos a nivel de cómputo, Synapse Spark tiene una característica muy útil que ayuda a identificar los sesgos de los datos en cada etapa del trabajo de Spark. Si vas a la pestaña de aplicaciones de Apache Spark en la sección de monitorización de un workspace de Synapse, puedes ver los detalles de los sesgos. Véase la siguiente captura de pantalla:

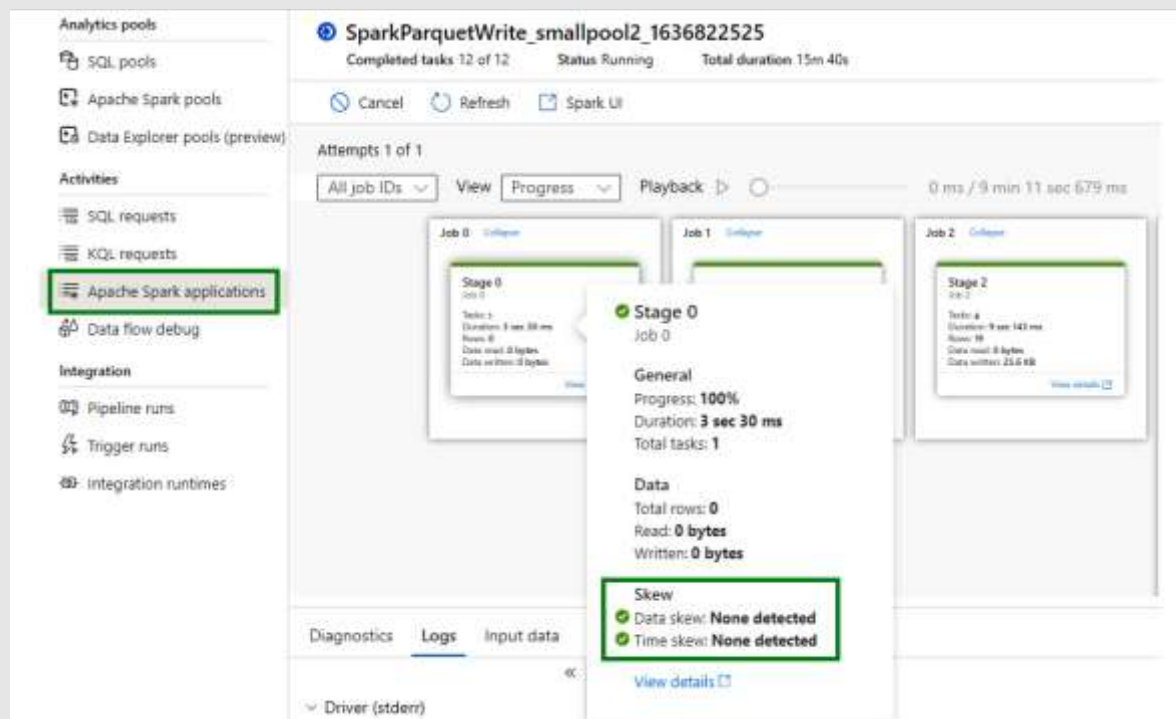


Figura 14.9 - Synapse Spark notificando los sesgos de los datos en las etapas de Spark

Esta característica de Synapse Spark hace que sea muy fácil para cualquier persona identificar los sesgos en sus conjuntos de datos.

Ahora ya conoce las técnicas básicas para manejar los sesgos de los datos. Veamos ahora cómo manejar los derrames de datos (data spills).

14.5. Manejar los derrames de datos (data spills)

Data spill se refiere al proceso en el que un motor de cálculo como SQL o Spark, mientras ejecuta una consulta, es incapaz de mantener los datos requeridos en la memoria y escribe (desborda) en el disco. Esto da lugar a un aumento del tiempo de ejecución de la consulta debido a las costosas lecturas y escrituras en disco. Los derrames pueden ocurrir por cualquiera de las siguientes razones:

- ❖ El tamaño de la partición de datos es demasiado grande.
- ❖ El tamaño de los recursos de cómputo es pequeño, especialmente la memoria.
- ❖ El tamaño de los datos explotados durante las fusiones (merges), uniones y demás supera los límites de memoria del nodo de cálculo.

Las soluciones para manejar los desbordamientos de datos serían las siguientes

- ❖ Aumentar la capacidad de cómputo, especialmente la memoria si es posible. Esto supondrá un mayor coste, pero es la más sencilla de las opciones.
- ❖ Reducir el tamaño de las particiones de datos y repartirlas si es necesario. Esta opción requiere más esfuerzo, ya que la repartición requiere tiempo y esfuerzo. Si no puede permitirse los mayores recursos informáticos, la mejor opción es reducir el tamaño de las particiones de datos.
- ❖ Eliminar los sesgos en los datos. A veces, es el perfil de los datos el que causa los desbordamientos. Si los datos están sesgados, pueden causar derrames en las particiones con mayor tamaño de datos. Ya vimos las opciones para manejar las inclinaciones de los datos en la sección anterior. Puede intentar utilizar esas opciones.

Estas son las técnicas generales para manejar los derrames. Sin embargo, para arreglar los derrames de datos, necesitamos primero identificar los derrames. Veamos cómo identificar los derrames en Synapse SQL y Spark.

14.5.1. Identificación de derrames de datos en Synapse SQL

El principal indicador en Synapse SQL que indica un exceso de derrames de datos es que TempDB se está quedando sin espacio. Si nota que sus consultas de Synapse SQL fallan debido a problemas con TempDB, podría ser un indicador de un derrame de datos.

Azure proporciona la siguiente consulta para supervisar el uso de la memoria y el uso de TempDB para las consultas de Synapse SQL. Como la consulta es grande y no será fácil para cualquier persona leer y reproducir de un libro de texto, sólo estoy proporcionando los enlaces aquí:

- ❖ Consulta para monitorizar el uso de memoria de las consultas SQL:

<https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/sql-data-warehouse-manage-monitor#monitor-memory>

- ❖ Consulta para monitorizar el uso de TempDB de las consultas SQL:

<https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/sql-data-warehouse-manage-monitor#monitor-tempdb>

A continuación, vamos a ver cómo identificar los derrames en Spark.

14.5.2. Identificación de derrames de datos en Spark

Identificar los derrames en Spark es relativamente sencillo ya que Spark publica métricas para los derrames. Puede comprobar los derrames en la pantalla de resumen de tareas de la interfaz de usuario de Spark.

Duration	GC Time	Spill (Disk)
76.0 ms		124 B / 4
76.0 ms		122 B / 4

Figura 14.10 - Identificar los derrames en Spark desde las métricas del resumen de tareas

La última columna Spill (Disk) indica los bytes de datos escritos en el disco. Así es como podemos identificar los derrames de datos en Spark. A continuación, vamos a ver una técnica de Spark para afinar las particiones shuffle.

14.6. Ajuste de las particiones shuffle

Spark utiliza una técnica llamada shuffle para mover datos entre sus ejecutores o nodos mientras realiza operaciones como join, union, groupby y reduceby. La operación shuffle es muy costosa ya que implica el movimiento de datos entre nodos. Por lo tanto, suele ser preferible reducir la cantidad de shuffle involucrada en una consulta de Spark. El número de particiones que Spark realiza al barajar (shuffle) los datos viene determinado por la siguiente configuración

```
spark.conf.set("spark.sql.shuffle.partitions",200)
```

200 es el valor por defecto y puedes ajustarlo a un número que se adapte mejor a tu consulta. Si tienes demasiados datos y muy pocas particiones, esto podría resultar en tareas más largas. Pero, por otro lado, si tiene muy pocos datos y demasiadas particiones barajadas, la sobrecarga de las tareas de barajado degradará el rendimiento. Por lo tanto, tendrás que ejecutar tu consulta varias veces con diferentes números de particiones aleatorias para llegar a un número óptimo.

Puedes aprender más sobre el ajuste del rendimiento de Spark y las particiones aleatorias aquí: <https://spark.apache.org/docs/latest/sql-performance-tuning.html>.

A continuación, vamos a ver cómo identificar los shuffles en los pipelines con el fin de ajustar las consultas.

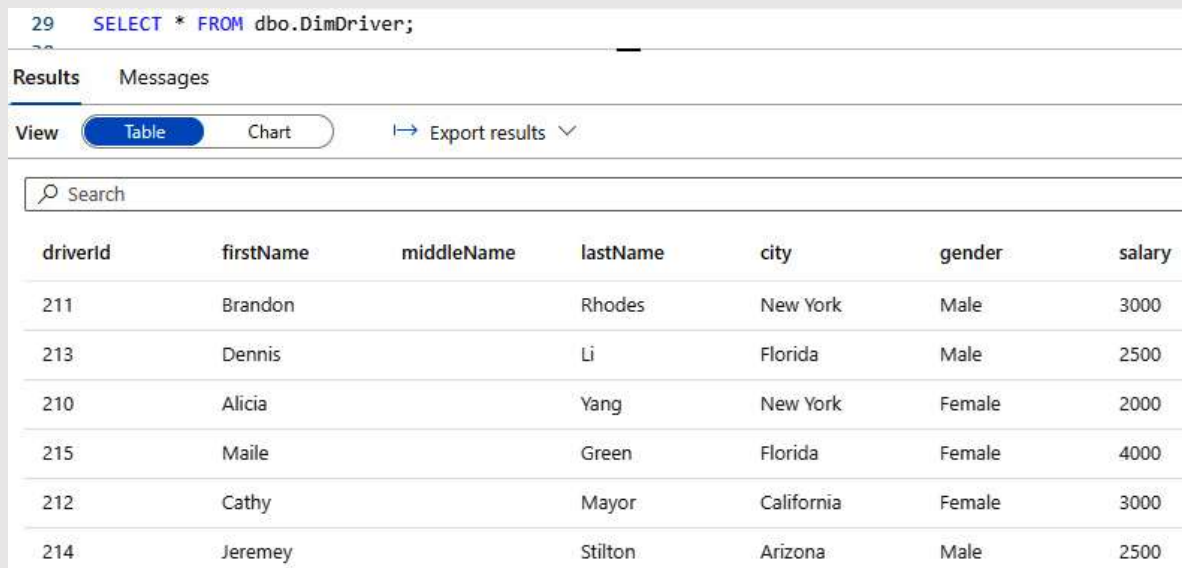
14.7. Encontrando el shuffling en un pipeline

Como aprendimos en la sección anterior, shuffling data es una operación muy costosa y debemos intentar reducirla al máximo. En esta sección, aprenderemos a identificar los shuffles en la ruta de ejecución de la consulta tanto para Synapse SQL como para Spark.

14.7.1. Identificar shuffles en un plan de consulta SQL

Para identificar los shuffles, imprime el plan de consulta usando la sentencia EXPLAIN. Aquí hay un ejemplo.

Considere una tabla de Synapse SQL, DimDriver, como se muestra en la siguiente captura de pantalla:



29 `SELECT * FROM dbo.DimDriver;`

Results Messages

View Table Chart Export results

Search

driverid	firstName	middleName	lastName	city	gender	salary
211	Brandon		Rhodes	New York	Male	3000
213	Dennis		Li	Florida	Male	2500
210	Alicia		Yang	New York	Female	2000
215	Maile		Green	Florida	Female	4000
212	Cathy		Mayor	California	Female	3000
214	Jeremey		Stilton	Arizona	Male	2500

Figura 14.11 - Ejemplo de tabla DimDriver

A continuación, se muestra una sentencia EXPLAIN de ejemplo:

```
EXPLAIN WITH_RECOMMENDATIONS
SELECT
    [gender], SUM([salary]) as Totalsalary
FROM
    dbo.DimDriver
GROUP BY
    [gender]
```

Esto generará un plan similar al que se muestra en la siguiente captura de pantalla. La consulta imprime un plan XML. He copiado y pegado el plan en un editor de texto para facilitar la lectura del XML.

```

<?xml version="1.0" encoding="utf-8"?>
<dsql_query number_nodes="1" number_distributions="60" number_distributions_per_node="60">
  <sql>SELECT [gender], SUM([salary]) as Totalsalary FROM dbo.DimDriver GROUP BY [gender]
</sql>
<materialized view candidates>
</materialized view candidates>
<dsql_operations total_cost="0.01056" total_number_operations="5">
  <dsql_operation operation_type="RND_ID">
  </dsql_operation>
  <dsql_operation operation_type="ON">
  </dsql_operation>
  <dsql_operation operation_type="SHUFFLE_MOVE">
    <operation_cost cost="0.01056" accumulative_cost="0.01056" average_rowsize="44"
    output_rows="31.6228" GroupNumber="8"/>
    <source_statement>SELECT [T1_1].[gender] AS [gender], [T1_1].[col] AS [col] FROM
    (SELECT SUM([T2_1].[salary]) AS [col], [T2_1].[gender] AS [gender] FROM
    [dp203dedicatedsql].[dbo].[DimDriver] AS T2_1 GROUP BY [T2_1].[gender]) AS T1_1
    OPTION (MAXDOP 1, MIN_GRANT_PERCENT = [MIN_GRANT], DISTRIBUTED_MOVE(N''))
    </source_statement>
    <destination_table>[TEMP_ID_5]</destination_table>
    <shuffle_columns>gender;</shuffle_columns>
  </dsql_operation>
  <dsql_operation operation_type="RETURN">
  </dsql_operation>
  <dsql_operation operation_type="ON">
  </dsql_operation>
</dsql_operations>
</dsql_query>

```

Figura 14.12 - Ejemplo de plan de consulta de Synapse SQL

En el plan de consulta, busque la palabra clave SHUFFLE_MOVE para identificar las etapas del shuffle. La sección de shuffle move tendrá los detalles del costo, el número de filas involucradas, la consulta exacta que causa el movimiento aleatorio (shuffle), y más. Puede utilizar esta información para reescribir sus consultas para evitar el shuffling.

CONSEJO

Lea un plan de consulta de abajo hacia arriba para entender el plan fácilmente.

A continuación, aprendamos a identificar las etapas del shuffle en Spark.

14.7.2. Identificar shuffles en un plan de consulta de Spark

Al igual que en SQL, podemos utilizar el comando EXPLAIN para imprimir los planes en Spark. Aquí tenemos un ejemplo sencillo para generar dos conjuntos de números, particionarlos y luego unirlos (join them). Esto provocará mucho movimiento de datos:

```

val jump2Numbers = spark.range(0, 100000, 2)
val jump5Numbers = spark.range(0, 200000, 5)
val ds1 = jump2Numbers.repartition(3)
val ds2 = jump5Numbers.repartition(5)
val joined = ds1.join(ds2)
joined.explain

```

La petición **joined.explain** imprimirá un plan similar al del ejemplo que se muestra a continuación:

== Physical Plan ==

BroadcastNestedLoopJoin BuildRight, Inner

:- **Exchange** RoundRobinPartitioning(3), [id=#216]

: +- *(1) Range (0, 100000, step=2, splits=4)

+ BroadcastExchange IdentityBroadcastMode, [id=#219]

+ **Exchange** RoundRobinPartitioning(5), [id=#218]

+ *(2) Range (0, 200000, step=5, splits=4)

Basta con buscar la palabra clave Exchange para identificar las etapas del shuffle.

Alternativamente, puede identificar la etapa de shuffle desde el DAG de Spark. En el capítulo anterior, vimos cómo ver el DAG de Spark desde la pantalla de Spark UI. En el DAG, busque las secciones llamadas Exchange. Estas son las secciones de shuffle. Aquí hay un ejemplo de DAG de Spark que contiene dos etapas de Exchange:

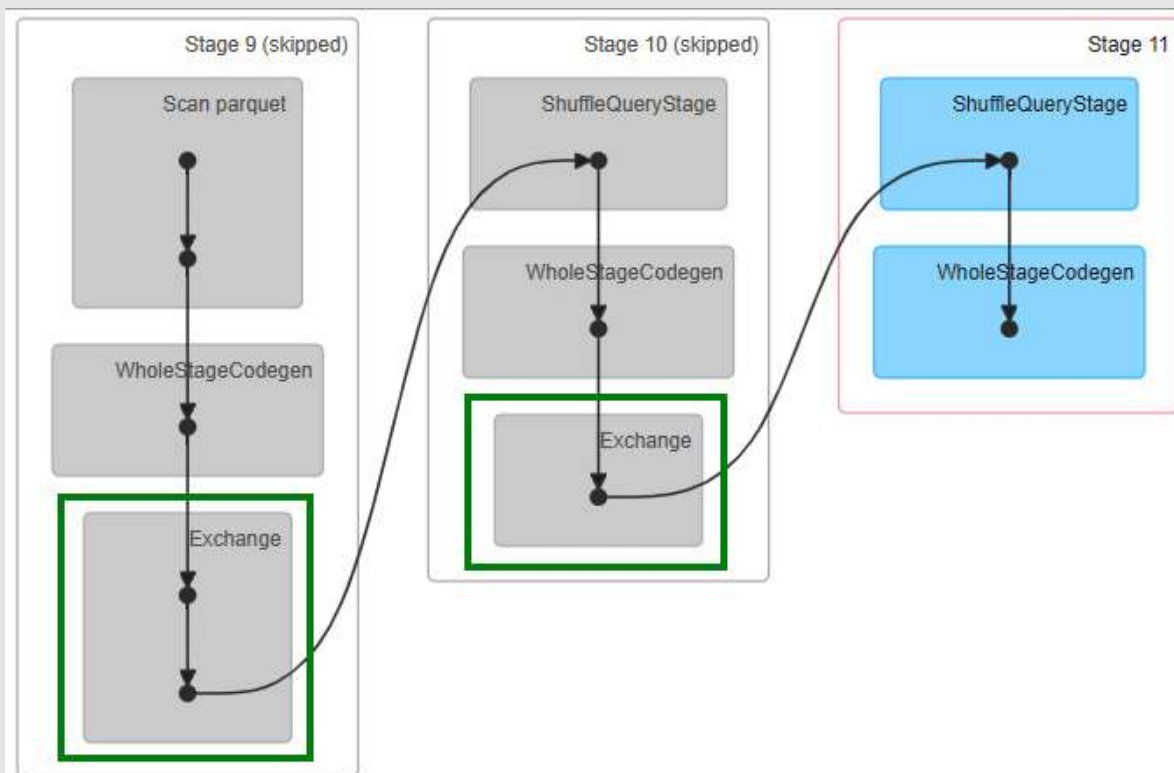


Figura 14.13 - Etapas de Exchange (etapas de shuffle) en un Spark job

Si hay secciones de shuffle muy caras, considere habilitar las estadísticas y comprobar si el motor genera un plan mejor. Si no es así, tendrás que reescribir la consulta para reducir los shuffles tanto como sea posible.

A continuación, vamos a ver la optimización de la gestión de recursos para los pipelines analíticos basados en la nube.

14.8. Optimización de la gestión de recursos

La optimización de la gestión de recursos en este contexto se refiere a cómo reducir los gastos de facturación mientras se utilizan los servicios analíticos de Azure. Estas son algunas de las técnicas generales que pueden ayudar.

14.8.1. Optimización de los Synapse SQL pools

Aquí hay algunas sugerencias para los Synapse dedicated SQL pools:

- ❖ Ya que el almacenamiento y el cómputo están desacoplados, usted puede pausar el cómputo de su SQL pool cuando no esté en uso. Esto no afectará a sus datos pero le ahorrará algunos costes.
- ❖ Utilice el tamaño correcto de las unidades de cómputo. En el SQL pool, las unidades de cálculo se definen en términos de Data Warehouse Units (DWU). Puede comenzar con la DWU más pequeña y luego aumentar gradualmente a DWUs más altas para lograr el equilibrio adecuado entre coste y rendimiento.
- ❖ Reduzca o aumente manualmente los recursos informáticos en función de la carga. También puede automatizar el escalado de salida y entrada utilizando Azure Functions.

Puede obtener más información sobre las optimizaciones de gestión de recursos para el SQL pool aquí: <https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/sql-data-warehouse-manage-compute-overview>.

14.8.2. Optimización de Spark

Aquí hay algunas sugerencias para Spark tanto en Synapse Spark como en Azure Databricks Spark:

- ❖ Elija las opciones de autoescala en Azure Databricks o Synapse Spark mientras configura el clúster. Esto eliminará la necesidad de gestionar los recursos manualmente.
- ❖ Seleccione la opción de auto-terminate en Azure Databricks y Synapse Spark para que el clúster se apague automáticamente si no se utiliza durante un periodo de tiempo configurado.
- ❖ Puede elegir instancias puntuales cuando estén disponibles para reducir el coste total del clúster. Se trata de nodos que son baratos pero que pueden ser retirados si hay jobs de mayor prioridad que necesitan los nodos.
- ❖ Elija el tipo correcto de nodos de clúster en función de los jobs que requieran mucha memoria, mucha CPU o mucha red. Seleccione siempre nodos que tengan más memoria que la memoria máxima requerida por sus jobs.

Estas son algunas de las formas de optimizar el uso de recursos. A continuación, vamos a ver cómo afinar las consultas utilizando indexadores.

14.9. Ajuste de las consultas mediante el uso de indexadores

La indexación es otra tecnología de optimización común utilizada en los sistemas de bases de datos, data warehouses y motores analíticos como Spark. Veamos las opciones de indexación y las directrices de ajuste tanto para Synapse SQL como para Spark.

14.9.1. Indexación en Synapse SQL

Si recuerdas, aprendimos sobre los diferentes tipos de indexación en el Capítulo 5, Implementación de estructuras de almacenamiento de datos físicos, en la sección Implementación de diferentes geometrías de tablas con los pools de Azure Synapse Analytics. Aquí volveré a resumir los diferentes tipos de indexadores que tenemos junto con consejos para afinar Synapse SQL.

Hay tres tipos principales de indexación disponibles en Synapse SQL:

- ❖ **Clustered Columnstore Index**: Esta es la opción de índice por defecto para Synapse SQL. Si usted no especifica ninguna opción de indexación, la tabla será indexada automáticamente usando este método. Utilice este índice para tablas grandes > 100 millones de filas. Proporciona niveles muy altos de compresión de datos y un buen rendimiento general.
- ❖ **Clustered Index**: Este tipo de indexación es mejor si tiene condiciones de filtro muy específicas que devuelven sólo unas pocas filas, por ejemplo, si tiene una cláusula WHERE que devuelve sólo 100 filas de un millón de filas. Normalmente, este tipo de indexación es bueno para < 100 millones de filas.
- ❖ **Heap**: Este índice es adecuado para las tablas temporales de preparación para cargar rápidamente los datos. También son buenos para pequeñas tablas de búsqueda y tablas con datos transitorios.

TIP

Si el rendimiento del índice se degrada con el tiempo debido a las cargas incrementales o a la deriva del esquema, intente reconstruir sus índices.

También puede crear un non-clustered index secundario sobre sus tablas clustered index para acelerar el filtrado.

Puede aprender más sobre la indexación en Synapse SQL aquí: <https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/sql-data-warehouse-tables-index#index-types>.

A continuación, veamos las opciones de indexación disponibles para Spark.

14.9.2. Indexación en el Synapse Spark pool usando Hyperspace

Spark no tiene ninguna opción de indexación incorporada en el momento de escribir este libro, aunque existe un sistema de indexación externo desarrollado por Microsoft. Microsoft ha introducido un proyecto llamado Hyperspace que ayuda a crear índices que pueden integrarse perfectamente en Spark para acelerar el rendimiento de las consultas. Hyperspace soporta formatos de datos comunes como CSV, JSON y Parquet.

Hyperspace proporciona un sencillo conjunto de APIs que pueden utilizarse para crear y gestionar los índices. Veamos ahora un ejemplo de cómo utilizar Hyperspace dentro de Spark. En este ejemplo, cargamos dos tablas, una que contiene datos de viajes y otra que contiene datos de conductores. A continuación, ejecutamos una consulta join para ver cómo entra en acción la indexación de Hyperspace. Veamos los pasos a seguir:

1. Cargar los datos en un DataFrame:

```
val tripsParquetPath = "abfss://path/to/trips/parquet/files"
val driverParquetPath = "abfss://path/to/driver/parquet/files"
val tripsDF: DataFrame = spark.read.parquet(tripsParquetPath)
val driverDF: DataFrame = spark.read.parquet(driverParquetPath)
```

2. Crear el índice Hyperspace:

```
import com.microsoft.hyperspace._
import com.microsoft.hyperspace.index._
val hs: Hyperspace = Hyperspace()

hs.createIndex(tripsDF, IndexConfig("TripIndex", indexedColumns = Seq("driverId"),
includedColumns = Seq("tripId")))

hs.createIndex(driverDF, IndexConfig("DriverIndex", indexedColumns = Seq("driverId"),
includedColumns = Seq("name")))
```

3. Habilita el índice Hyperspace y vuelve a cargar los DataFrames desde la misma ubicación del archivo

```
spark.enableHyperspace
val tripIndexDF: DataFrame = spark.read.parquet(tripsParquetPath)
val driverIndexDF: DataFrame = spark.read.parquet(driverParquetPath)
```

4. Ejecutar una consulta con join:

```
val filterJoin: DataFrame = tripIndexDF.join( driverIndexDF, tripIndexDF("driverId") ===
driverIndexDF("driverId")).select( tripIndexDF("tripId"), driverIndexDF("name"))
```

5. Comprueba cómo se ha incluido el índice Hyperspace en tu plan de consulta utilizando el comando explain

```
spark.conf.set("spark.hyperspace.explain.displayMode", "html")
hs.explain(filterJoin)(displayHTML(_))
```

6. A continuación se muestra un ejemplo de cómo sería un plan de consulta con el índice Hyperspace:

```
=====
Plan with indexes:
=====
Project [tripId#859, name#874]
+- BroadcastHashJoin [driverId#860], [driverId#873], Inner, BuildRight, false
   :- Filter isNotNull(driverId#860)
      : +- ColumnarToRow
         : +- FileScan Hyperspace(Type: CI, Name: TripIndex, LogVersion: 25) [driverId#860,tripId#859] Batched: true,
            DataFilters: [isNotNull(driverId#860)], Format: Parquet, Location:
            InMemoryFileIndex[abfss://sandbox@dp203teststorage.dfs.core.windows.net/synapse/workspaces/dp203t..., PartitionFilters: [],
            PushedFilters: [IsNotNull(driverId)], ReadSchema: struct
         +- BroadcastExchange HashedRelationBroadcastMode(List(input[0, string, false]),false), [id=#857]
            +- *(1) Filter isNotNull(driverId#873)
               +- *(1) ColumnarToRow
                  +- FileScan Hyperspace(Type: CI, Name: DriverIndex, LogVersion: 25) [driverId#873,name#874] Batched: true,
                     DataFilters: [isNotNull(driverId#873)], Format: Parquet, Location:
                     InMemoryFileIndex[abfss://sandbox@dp203teststorage.dfs.core.windows.net/synapse/workspaces/dp203t..., PartitionFilters: [],
                     PushedFilters: [IsNotNull(driverId)], ReadSchema: struct

=====
Plan without indexes:
=====
Project [tripId#859, name#874]
+- BroadcastHashJoin [driverId#860], [driverId#873], Inner, BuildRight, false
   :- Filter isNotNull(driverId#860)
      : +- ColumnarToRow
         : +- FileScan parquet [tripId#859,driverId#860] Batched: true, DataFilters: [isNotNull(driverId#860)], Format:
            Parquet, Location: InMemoryFileIndex[abfss://sandbox@dp203teststorage.dfs.core.windows.net/hyperspace/trips],
            PartitionFilters: [], PushedFilters: [IsNotNull(driverId)], ReadSchema: struct
         +- BroadcastExchange HashedRelationBroadcastMode(List(input[0, string, false]),false), [id=#810]
            +- *(1) Filter isNotNull(driverId#873)
               +- *(1) ColumnarToRow
                  +- FileScan parquet [driverId#873,name#874] Batched: true, DataFilters: [isNotNull(driverId#873)], Format:
            Parquet, Location: InMemoryFileIndex[abfss://sandbox@dp203teststorage.dfs.core.windows.net/hyperspace/driver],
            PartitionFilters: [], PushedFilters: [IsNotNull(driverId)], ReadSchema: struct
```

Figura 14.14 - Plan de consulta con índices Hyperspace

Observe que FileScan está leyendo el archivo de índices Hyperspace en lugar del archivo Parquet original.

Así de fácil es utilizar Hyperspace dentro de Spark. Puedes aprender más sobre la indexación Hyperspace para Spark aquí: <https://docs.microsoft.com/en-us/azure/synapse-analytics/spark/apache-spark-performance-hyperspace>.

A continuación, vamos a ver cómo afinar las consultas utilizando la caché.

14.10. Ajuste de las consultas mediante el uso de la caché

El almacenamiento en caché es un método bien conocido para mejorar el rendimiento de la lectura en las bases de datos. Synapse SQL admite una función llamada "Result set caching". Como su nombre indica, esto permite que los resultados se almacenen en caché y se reutilicen si la consulta no cambia. Una vez que se habilita el almacenamiento en caché del conjunto de resultados, las siguientes ejecuciones de la consulta obtienen directamente los resultados de la caché en lugar de volver a calcularlos. Result set cache sólo se utiliza en las siguientes condiciones

- ❖ La consulta que se está considerando es una coincidencia exacta.
- ❖ No hay cambios en los datos o el esquema subyacentes.
- ❖ El usuario tiene el conjunto correcto de permisos para las tablas referenciadas en la consulta.

Puede habilitar result set caching a nivel de la base de datos en Synapse SQL utilizando la siguiente sentencia SQL:

```
ALTER DATABASE [database_name]
SET RESULT_SET_CACHING ON;
```

También puede activar result set caching desde una sesión mediante el siguiente comando:

```
SET RESULT_SET_CACHING { ON | OFF };
```

NOTA

El tamaño máximo de result set cache es de 1 TB por base de datos. Synapse SQL desaloja automáticamente los datos antiguos cuando se alcanza el tamaño máximo o si los resultados se invalidan debido a cambios en los datos o en el esquema.

Puede obtener más información sobre result set caching aquí: <https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/performance-tuning-result-set-caching>.

Al igual que Synapse SQL, Synapse Spark y Azure Databricks Spark también soportan el almacenamiento en caché, pero en ámbitos mucho más pequeños como el almacenamiento en caché de un RDD o un Dataframe. Spark proporciona métodos como `cache()` y `persist()`, que pueden utilizarse para almacenar en caché resultados intermedios de RDDs, DataFrames o datasets. He aquí un ejemplo sencillo para almacenar en caché las entradas de un DataFrame creado a partir de un archivo CSV:

```
df = spark.read.csv("path/to/csv/file")
cached_df = df.cache()
```

Una vez que se almacena en caché el DataFrame, los datos se mantienen en la memoria y le ofrece un rendimiento de consulta más rápido. Puedes aprender más sobre las opciones de caché de Spark aquí: <https://docs.microsoft.com/en-us/azure/synapse-analytics/spark/apache-spark-performance#use-the-cache>.

Veamos ahora cómo optimizar los pipelines para casos de uso analítico y transaccional.

14.11. Optimización de pipelines para fines analíticos o transaccionales

Seguro que has oído los términos OLAP y OLTP si has trabajado en el ámbito de los datos. Los sistemas de datos en la nube pueden clasificarse a grandes rasgos como sistemas de procesamiento de transacciones en línea (OLTP) o de procesamiento analítico en línea (OLAP). Entendamos cada uno de ellos a alto nivel.

14.11.1. Sistemas OLTP

Los sistemas OLTP, como su nombre indica, están contruidos para procesar, almacenar y consultar transacciones de forma eficiente. Por lo general, los datos de las transacciones fluyen hacia una base de datos central que cumple con ACID. Las bases de datos contienen datos normalizados que se adhieren a esquemas estrictos. El tamaño de los datos suele ser menor, del orden de gigabytes o terabytes. Para la base de datos principal se utilizan predominantemente sistemas basados en RDBMS, como Azure SQL y MySQL.

14.11.2. Sistemas OLAP

Por otro lado, los sistemas OLAP suelen ser sistemas de big data que suelen tener un warehouse o almacén basado en valores clave como tecnología central para realizar el procesamiento analítico. Las tareas pueden ser la exploración de datos, la generación de conocimientos a partir de datos históricos, la predicción de resultados, etc. Los datos de un sistema OLAP proceden de diversas fuentes que no suelen adherirse a ningún esquema o formato. Suelen contener grandes cantidades de datos en el rango de terabytes, petabytes y más. La tecnología de almacenamiento utilizada suele ser el almacenamiento basado en columnas, como Azure Synapse SQL Pool, HBase y CosmosDB Analytical storage, que tienen un mejor rendimiento de lectura.

Podemos optimizar los pipelines para OLAP u OLTP, pero ¿cómo optimizamos para ambos? Aparecen los sistemas de Procesamiento Analítico Transaccional Híbrido (HTAP). Se trata de una nueva clase de sistemas de datos que pueden manejar tanto el procesamiento transaccional como el analítico. Combinan el almacenamiento basado en filas y columnas para proporcionar una funcionalidad híbrida. Estos sistemas eliminan la necesidad de mantener dos conjuntos de conductos, uno para el procesamiento transaccional y otro para el procesamiento analítico. Tener la flexibilidad de realizar simultáneamente sistemas transaccionales y analíticos abre un nuevo campo de oportunidades, como las recomendaciones en tiempo real durante transacciones, tablas de clasificación en tiempo real y la capacidad de realizar consultas ad hoc sin afectar a los sistemas de transacciones.

Veamos ahora cómo construir un sistema HTAP utilizando tecnologías Azure.

14.11.3. Implementación de HTAP utilizando Synapse Link y CosmosDB

HTAP se realiza en Azure utilizando Azure Synapse y Azure CosmosDB a través de Azure Synapse Link. Ya conocemos Azure Synapse, así que veamos aquí CosmosDB y Synapse Link.

Introduciendo CosmosDB

CosmosDB es una base de datos NoSQL totalmente gestionada y distribuida globalmente que soporta varios formatos de API, incluyendo SQL, MongoDB, Cassandra, Gremlin y Key-Values. Es extremadamente rápida y se escala sin problemas a través de las geografías. Puede habilitar escrituras multirregionales en todo el mundo con unos simples clics. CosmosDB es adecuado para casos de uso como las plataformas de venta al por menor para el procesamiento de pedidos, la catalogación, las aplicaciones de juegos que necesitan una baja latencia con la capacidad de manejar rachas de tráfico, la telemetría y las aplicaciones de registro para generar información rápida.

CosmosDB almacena internamente los datos operativos en un almacén transaccional basado en filas, que es bueno para las cargas de trabajo OLTP. Sin embargo, también ofrece soporte para habilitar un almacén analítico secundario basado en columnas que se persiste por separado del almacén transaccional, lo que es bueno para las cargas de trabajo analíticas. Por lo tanto, proporciona lo mejor de ambos entornos OLTP y OLAP. Por lo tanto, CosmosDB es perfectamente adecuado para las cargas de trabajo HTAP. Dado que el almacén de filas y el de columnas están separados el uno del otro, no hay impacto en el rendimiento al ejecutar cargas de trabajo transaccionales y analíticas simultáneamente. Los datos del almacén transaccional se sincronizan automáticamente con el almacén de columnas casi en tiempo real.

Introduciendo Azure Synapse Link

Synapse Link, como su nombre indica, enlaza Synapse y CosmosDB para proporcionar una capacidad HTAP nativa en la nube. Podemos utilizar cualquiera de los motores de cómputo de Synapse, ya sea Synapse Serverless SQL Pool o Spark pool, para acceder a los datos operacionales de CosmosDB y ejecutar analíticas sin impactar el procesamiento transaccional en Cosmos DB. Esto es significativo porque elimina por completo la necesidad de trabajos ETL, que eran necesarios anteriormente. Antes de la disponibilidad de Synapse Link, teníamos que ejecutar pipelines ETL para obtener los datos transaccionales en un almacén analítico como un Data Warehouse antes de poder ejecutar cualquier análisis o BI en los datos. Ahora, podemos consultar directamente los datos desde el almacén analítico CosmosDB para BI.

Esta es la arquitectura de enlace de Synapse reproducida de la documentación de Azure:

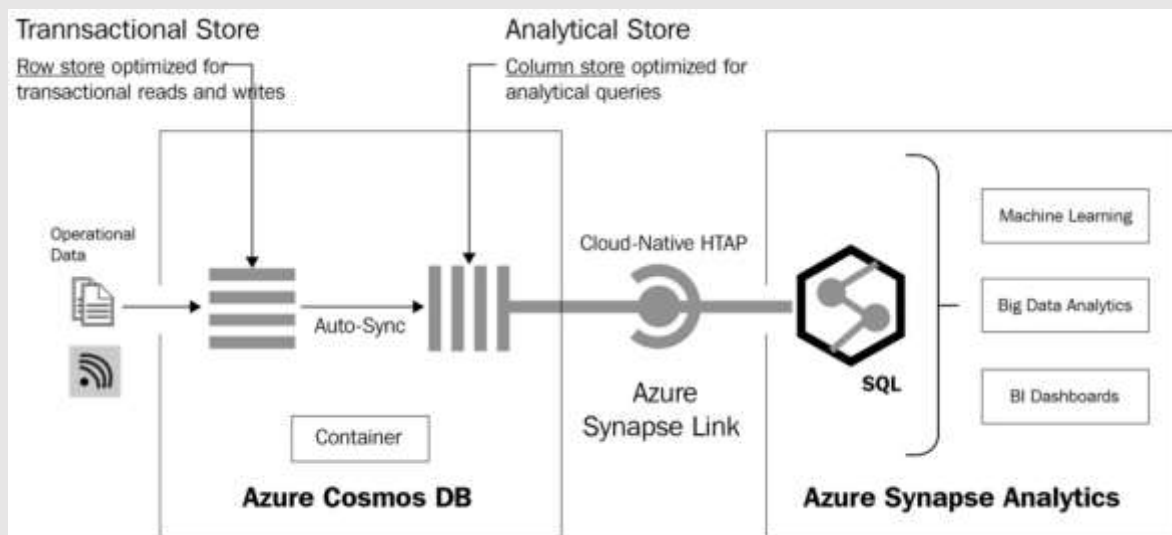


Figura 14.15 - Arquitectura de Azure Synapse Link for CosmosDB

NOTA

El acceso al almacén analítico de Azure Cosmos DB con Azure Synapse Dedicated SQL pool no estaba soportado en el momento de escribir este libro. Sólo se soportaba Serverless SQL pool.

Veamos ahora los pasos para crear un enlace Synapse y configurar un sistema HTAP:

1. Primero vamos a crear una instancia de CosmosDB. Busca CosmosDB en el portal de Azure y selecciónalo. En el portal de CosmosDB, seleccione el botón + Crear para crear una nueva instancia de CosmosDB, como se muestra:

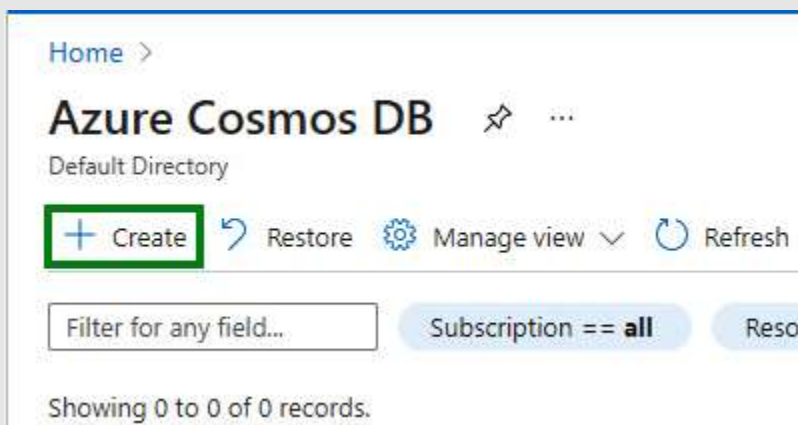


Figura 14.16 - Creación de una nueva instancia de CosmosDB

2. Al hacer clic en + Create, se mostrarán las diferentes opciones de la API disponibles en CosmosDB. Selecciona Core SQL. Una vez seleccionada, verás una pantalla como la que se muestra en la siguiente captura. Sólo tienes que completar los detalles, incluyendo el grupo de

recursos, el nombre de la cuenta, la ubicación y otros campos requeridos y hacer clic en Revisar + crear para crear la nueva instancia de CosmosDB.

... > Select API option >

Create Azure Cosmos DB Account - Core ...

Basics Global Distribution Networking Backup Policy

Azure Cosmos DB is a fully managed NoSQL database service for building scalable, high performance applications. [Try it for free](#), for 30 days with unlimited renewals. Go to production starting at \$24/month per database, multiple containers included. [Learn more](#)

Project Details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * Azure subscription 1

Resource Group * [Create new](#)

Instance Details

Account Name * Enter account name

Location *

Capacity mode [?](#) ☒ Provisioned throughput ☐ Serverless [Learn more about capacity mode](#)

With Azure Cosmos DB free tier, you will get the first 1000 RU/s and 25 GB of storage for free in an account. You can enable free tier on up to one account per subscription. Estimated \$64/month discount per account.

Apply Free Tier Discount ☒ Apply ☐ Do Not Apply

[Review + create](#) [Previous](#) [Next: Global Distribution](#)

Figura 14.17 - Pantalla de creación de CosmosDB

- Una vez creada la instancia de CosmosDB, te pedirá que crees un contenedor y añadas un conjunto de datos de ejemplo. Sigue adelante y añádelos.
- Ahora, ve a la pestaña Azure Synapse Link en Integraciones. Haz clic en el botón Enable para habilitar Synapse Link, como se muestra en la siguiente captura de pantalla:

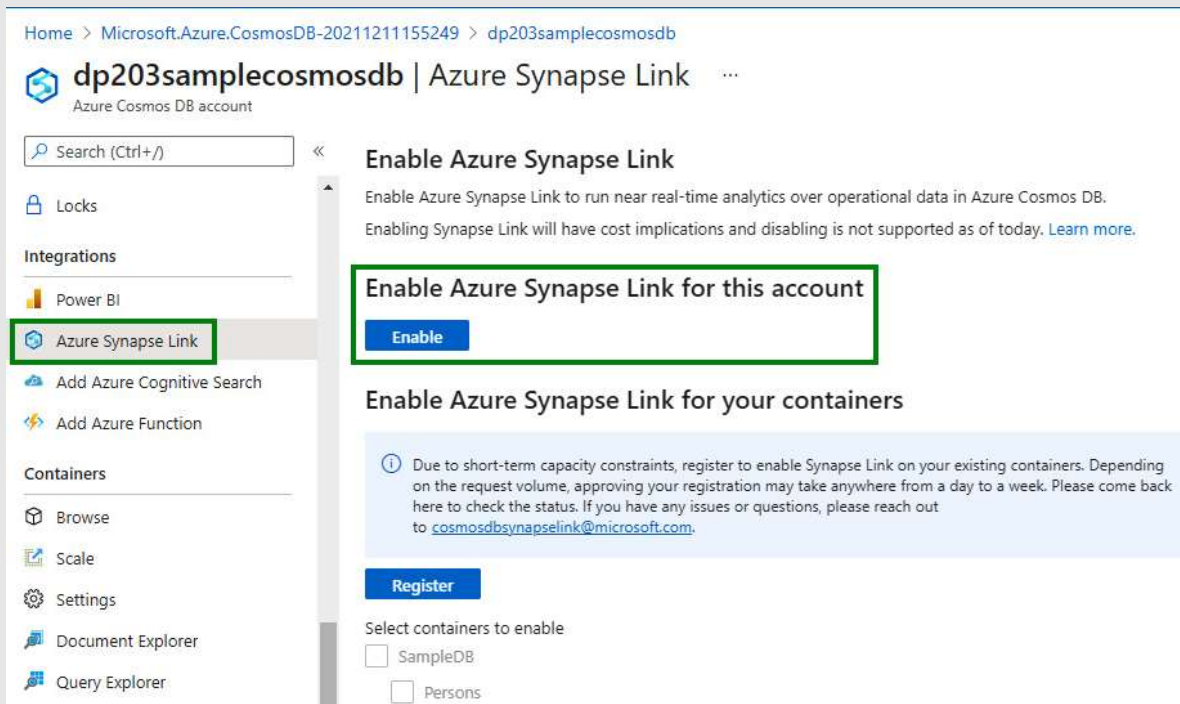


Figura 14.18 - Habilitación de Synapse Link en CosmosDB

5. A continuación, dirígete a la pestaña Data Explorer, haz clic en New Container, y completa los detalles en el formulario que aparece como se muestra en la siguiente captura de pantalla. Seleccione On para la opción de almacén analítico en la parte inferior de la pantalla. Esto debería configurar el enlace de CosmosDB para Synapse.

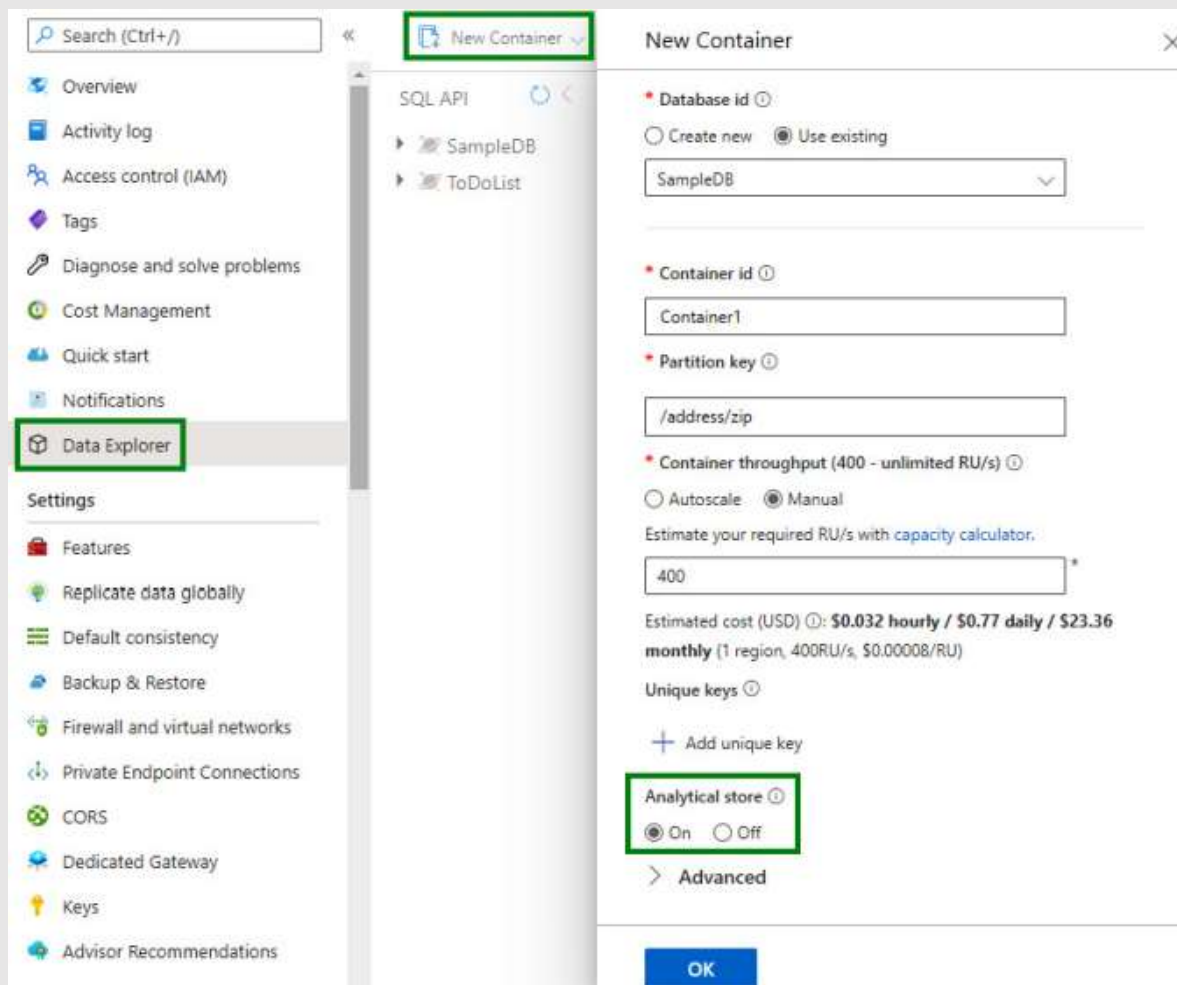


Figura 14.19 - Configuración de un nuevo contenedor con el almacén analítico activado

6. Ahora, tenemos que configurar Synapse para hablar con CosmosDB. Primero tenemos que configurar un servicio vinculado a CosmosDB desde el workspace de Synapse. Desde el portal de Synapse, vaya a la pestaña Manage y luego seleccione Linked Services. Haga clic en el botón + Nuevo y seleccione Azure CosmosDB (SQL API), como se muestra en la siguiente captura de pantalla:

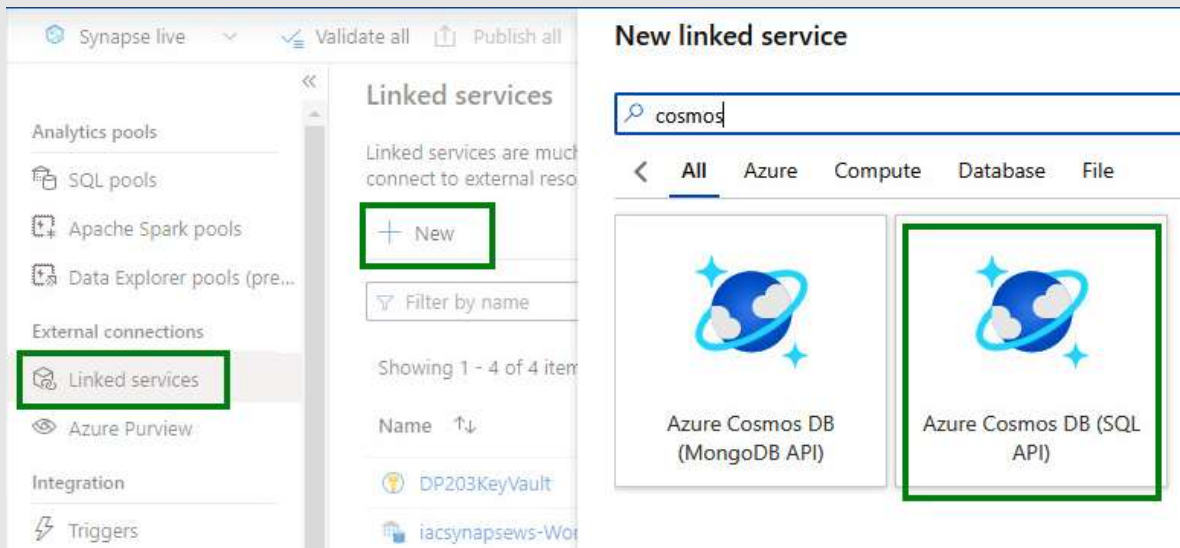


Figura 14.20 - Creación de un servicio vinculado a Cosmos DB

7. En la pantalla que aparecerá (no se muestra aquí), completa los detalles del CosmosDB que hemos creado anteriormente y crea el servicio vinculado.
8. Al hacer clic en el servicio vinculado recién creado, debería mostrar los detalles del servicio vinculado a CosmosDB junto con los detalles de la cadena de conexión, como se muestra en la siguiente captura de pantalla:

Edit linked service (Azure Cosmos DB (SQL API))

Name *
SampleCosmosDb

Description

Connect via integration runtime * ⓘ
✓ AutoResolveIntegrationRuntime

Authentication method
Account key

Connection string Azure Key Vault

Account selection method ⓘ
☐ From Azure subscription ☒ Enter manually

Azure Cosmos DB account URI *
https://[redacted]osdb.documents.azure.com:443/

Azure Cosmos DB access key Azure Key Vault

Azure Cosmos DB access key *

Database name *
SampleDB

Additional connection properties

Apply Cancel Test connection

Figura 14.21 - Detalles del servicio vinculado a CosmosDB

9. Ahora la configuración del enlace de Synapse está completa. Ve al workspace de Synapse, selecciona la pestaña Data, y ahora deberías poder ver una entrada de Azure Cosmos DB allí. Podemos explorar los datos en CosmosDB haciendo clic en los nombres de los contenedores bajo la entrada de Azure Cosmos DB y seleccionando Load to DataFrame, como se muestra en la siguiente captura de pantalla:

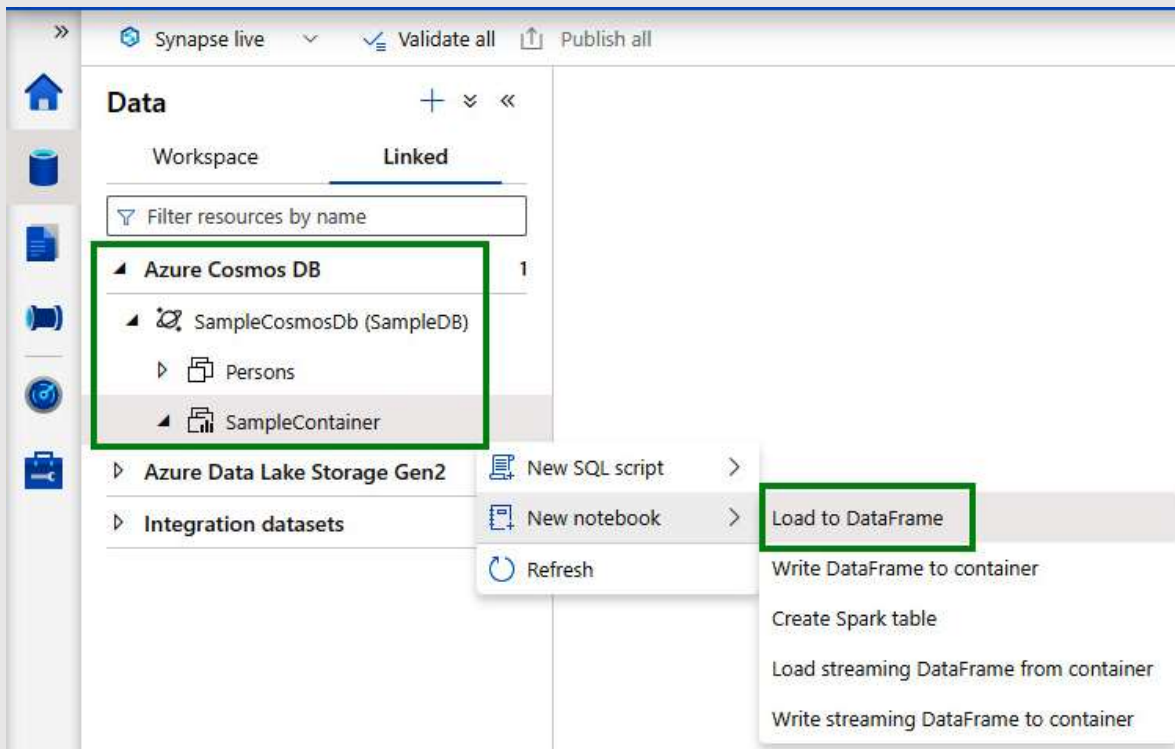


Figura 14.22 - Carga de datos de CosmosDB mediante el enlace Synapse

10. Puedes consultar los datos de CosmosDB desde el notebook de Spark usando OLAP, como se muestra:

```
df = spark.read.format("cosmos.olap")\  
  .option("spark.synapse.linkedService", "<Linked Service Name>")\  
  .option("spark.cosmos.container", "<CosmosDB Container Name>")\  
  .load()
```

Para las consultas OLTP, cambie el formato a cosmos.oltp en lugar de cosmos.olap.

11. Ahora ya sabes cómo implementar un sistema HTAP y realizar consultas desde él.

Puedes aprender más sobre HTAP y el enlace de Azure Synapse aquí: <https://docs.microsoft.com/en-us/azure/cosmos-db/synapse-link>.

Veamos ahora cómo optimizar los pipelines para cargas de trabajo descriptivas y analíticas.

14.12. Optimización de los pipelines para cargas de trabajo descriptivas frente a las analíticas

La analítica de datos se clasifica en cuatro tipos diferentes:

- ❖ **Analítica descriptiva:** El tipo de analítica que se ocupa del análisis de lo que ha ocurrido y cuándo ha ocurrido. La mayoría de los informes de BI, como los informes de ventas y los informes de viajes, que muestran puntos de datos actuales e históricos entran en esta categoría. Las tareas de análisis suelen ser recuentos, agregados, filtros, etc.
- ❖ **Análisis de diagnóstico:** Este tipo de análisis también hace la parte del por qué, junto con el qué y el cuándo. Algunos ejemplos son el Análisis de Causa Raíz (RCA). Además de identificar lo que ha ocurrido, también se profundiza en los registros o métricas para identificar por qué ha ocurrido algo. Por ejemplo, se podría analizar por qué una determinada ruta de taxi está teniendo un descenso en los ingresos, o por qué un tipo concreto de VM está fallando antes que otros, examinando la carga de esas máquinas.
- ❖ **Análisis predictivo:** Como su nombre indica, este tipo de análisis se refiere a la predicción de lo que va a suceder. Se suele asociar con el machine learning (aprendizaje automático). Se utilizan conjuntos de datos históricos y tendencias para predecir lo que podría ocurrir a continuación. Esto podría ser para predecir las ventas durante las temporadas de vacaciones, predecir la hora punta de los taxis, etc.
- ❖ **Análisis prescriptivo:** Este último tipo de análisis es la versión más avanzada, en la que, basándose en las predicciones, el sistema también recomienda acciones correctoras. Por ejemplo, basándose en la predicción de picos de ventas, el sistema puede recomendar que se abastezca más de ciertos artículos de venta al por menor o, basándose en la hora punta, recomendar que se desplacen más taxis a una región concreta.

El tema de esta sección dice que hay que optimizar los pipelines descriptivos frente a los analíticos, pero en esencia, todas las categorías enumeradas anteriormente son sólo diferentes tipos de cargas de trabajo analíticas. Supongo que el comité del programa de estudios se refería a la optimización de un sistema centrado en el data warehouse frente a un sistema basado en un data pipeline que pueda alimentar otros servicios, como el machine learning.

El análisis descriptivo casi siempre incluye un sistema de data warehouse como un Synapse SQL pool para alojar los datos finales que se pueden servir a las herramientas de BI. Y la analítica en un sentido genérico suele incluir motores de computación analítica de big data como Spark, Hive y Flink para procesar grandes cantidades de datos procedentes de diversas fuentes. Los datos generados serían entonces utilizados por varios sistemas, incluyendo BI, machine learning y consultas ad hoc. Así pues, veamos las técnicas de optimización para los pipelines que implican data warehouses como Synapse SQL y tecnologías como Spark.

14.12.1. Optimizaciones comunes para pipelines descriptivos y analíticos

Las técnicas de optimización que hemos aprendido a lo largo de este libro serán todas aplicables tanto a los pipelines descriptivos como a los analíticos en general. Por ejemplo, las siguientes técnicas son comunes independientemente de los enfoques centrados en SQL Pool (descriptivos) o centrados en Spark (analíticos).

Optimizaciones a nivel de almacenamiento:

- ❖ Dividir los datos claramente en zonas: Zonas Raw, Transformation y Serving.
- ❖ Definir una buena estructura de directorios, organizada en torno a fechas.
- ❖ Particionar los datos en función del acceso a diferentes directorios y diferentes niveles de almacenamiento.
- ❖ Elegir el formato de datos adecuado: Parquet con una comparación Snappy funciona bien para Spark.
- ❖ Configurar el ciclo de vida de los datos, purgando los datos antiguos o moviéndolos a niveles de archivo.

Optimizaciones a nivel de cómputo:

- ❖ Utilizar la caché.
- ❖ Utilizar la indexación cuando esté disponible.
- ❖ Manejar los desbordamientos de datos.
- ❖ Manejar los sesgos de los datos.
- ❖ Ajuste sus consultas leyendo los planes de consulta.

A continuación, veamos las optimizaciones específicas para los pipelines descriptivos y analíticos.

14.12.2. Optimizaciones específicas para pipelines descriptivos y analíticos

Para Synapse SQL, considere las siguientes optimizaciones:

- ❖ Mantener las estadísticas para mejorar el rendimiento mientras se utiliza el optimizador basado en costes de Synapse SQL.
- ❖ Utilice PolyBase para cargar los datos más rápidamente.
- ❖ Utilice la distribución de hash para las tablas grandes.
- ❖ Utilizar tablas heap temporales para los datos transitorios.
- ❖ No sobreparticionar ya que Synapse SQL ya particiona los datos en 60 subparticiones.
- ❖ Minimizar el tamaño de las transacciones.
- ❖ Reduzca el tamaño de los resultados de las consultas.
- ❖ Utilice Result set cache si es necesario.
- ❖ Utilice el menor tamaño de columna posible.
- ❖ Utilizar una clase de recurso más grande (mayor tamaño de memoria) para mejorar el rendimiento de la consulta.
- ❖ Utilizar una clase de recurso más pequeña (menor tamaño de memoria) para aumentar la concurrencia.

Usted puede aprender mas acerca de la optimización de un pipeline de Synapse SQL aquí:

<https://docs.microsoft.com/en-us/azure/synapse-analytics/sql/best-practices-dedicated-sql-pool>.

<https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/cheat-sheet#index-your-table>.

Y para Spark, considere las siguientes optimizaciones:

- ❖ Elegir la abstracción de datos adecuada - Los DataFrames y los datasets suelen funcionar mejor que los RDDs.
- ❖ Elegir el formato de datos adecuado - Parquet con una compresión Snappy suele funcionar bien para la mayoría de los casos de uso de Spark.
- ❖ Usar caché - ya sea las incorporadas en Spark, como `.cache()` y `.persist()`, o bibliotecas de caché externas.
- ❖ Utilizar indexadores - utilizar Hyperspace para acelerar las consultas.
- ❖ Ajustar las consultas - reducir shuffles en el plan de consulta, elegir el tipo correcto de merges, etc.
- ❖ Optimizar la ejecución de los jobs - elegir el tamaño correcto de los contenedores para que los jobs no se queden sin memoria. Esto suele hacerse observando los registros para conocer los detalles de las ejecuciones anteriores.

Puedes aprender más sobre la optimización de los pipelines de Spark aquí:
<https://docs.microsoft.com/en-us/azure/synapse-analytics/spark/apache-spark-performance>.

Veamos ahora cómo solucionar un trabajo de Spark que ha fallado.

14.13. Solución de problemas de un Spark job fallido

Hay dos aspectos para solucionar un Spark job fallido en un entorno de nube: problemas ambientales y problemas del job. Veamos ambos factores en detalle.

14.13.1. Depuración de los problemas del entorno

Estos son algunos de los pasos que hay que seguir para comprobar los problemas del entorno:

1. Comprueba el estado de los servicios de Azure en la región donde se ejecutan tus clústeres de Spark utilizando este enlace: <https://status.azure.com/en-us/status>.
2. A continuación, comprueba si tu clúster de Spark está bien. Puedes hacer esto para tus clusters de HDInsight comprobando la página de inicio de Ambari. Vimos cómo comprobar el estado de Ambari en el Capítulo 13, Supervisión del almacenamiento y procesamiento de datos, en la sección Supervisión del rendimiento general del clúster. Aquí está la página de inicio de la pantalla de Ambari de nuevo para su referencia:

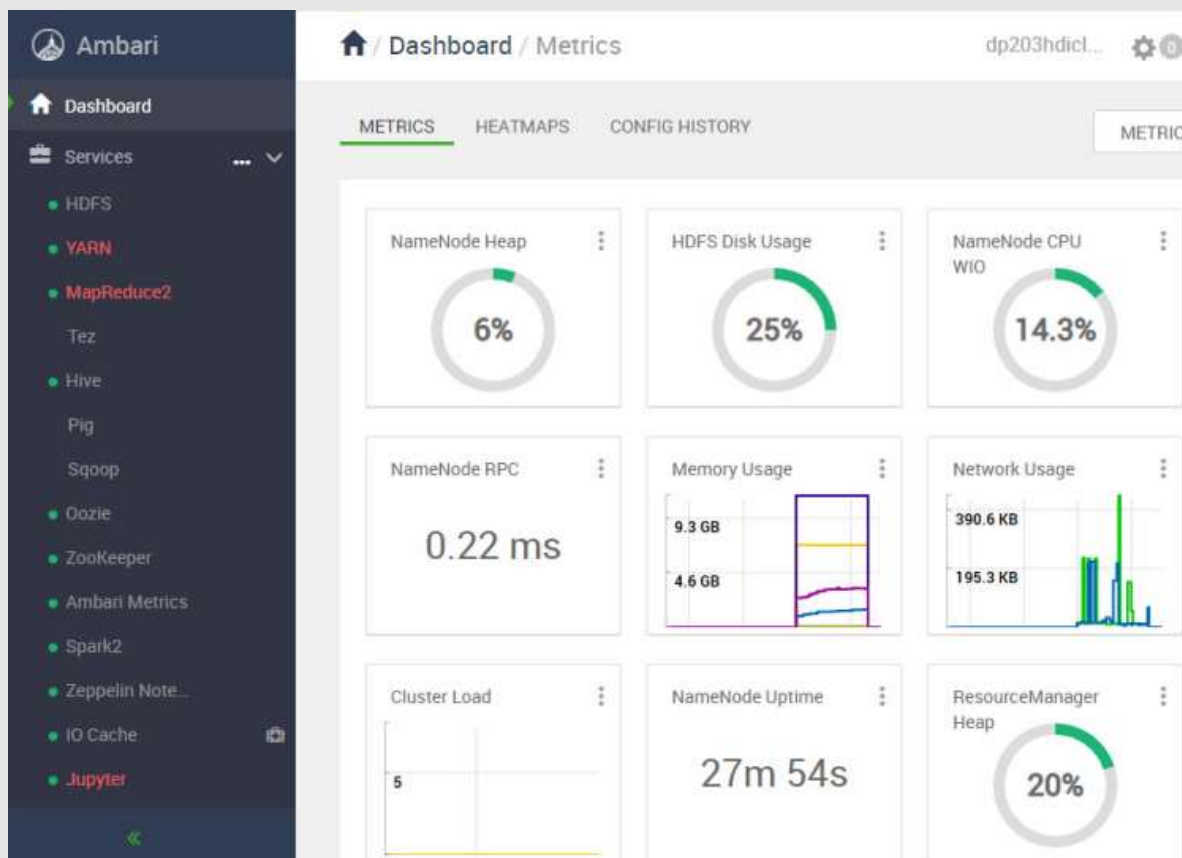


Figura 14.23 - Página de inicio de Ambari mostrando el estado del cluster

3. Comprueba si algún servicio está caído o si alguno de los recursos está funcionando en caliente con métricas, como un uso muy alto de la CPU o de la memoria.

A continuación, vamos a ver cómo depurar los problemas de los jobs.

14.13.2. Depuración de los problemas de los jobs

Si el entorno de la nube y los clústeres de Spark están en buen estado, tenemos que comprobar los problemas específicos del job. Hay tres archivos de registro principales que necesitas comprobar para cualquier problema relacionado con el job:

- ❖ **Driver Logs**: Puedes acceder al log del driver desde la pestaña Compute, como se muestra, en Azure Databricks:

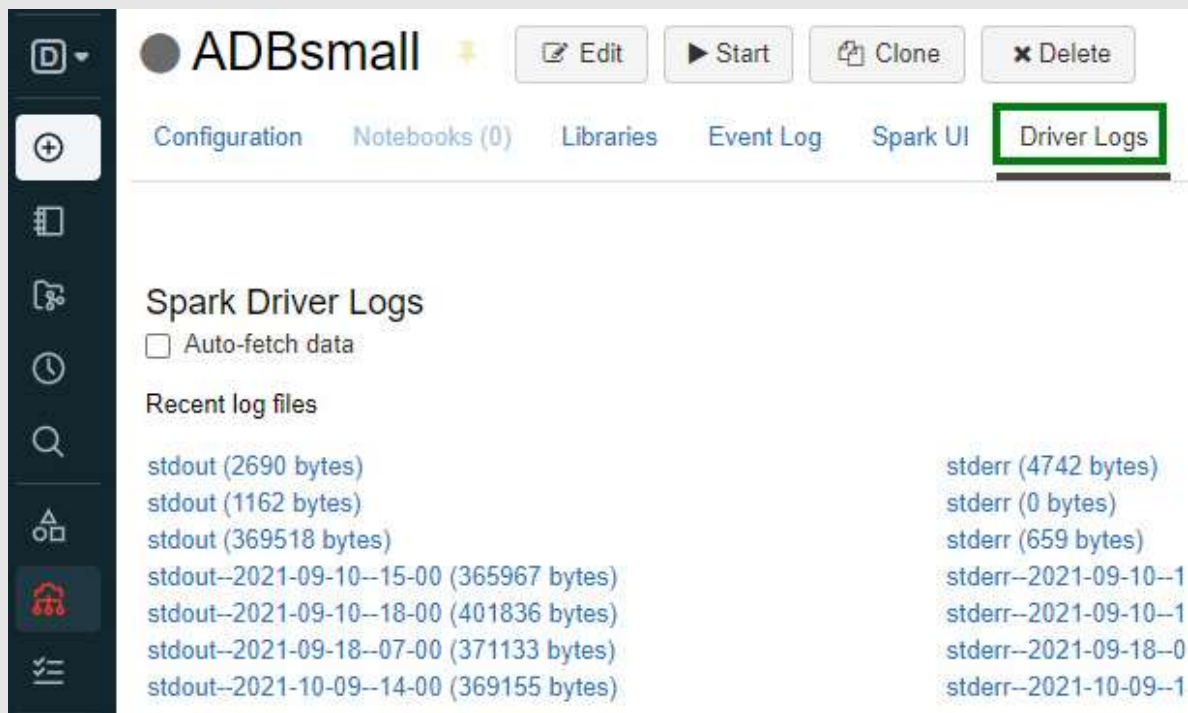


Figura 14.24 - Ubicación de los logs de los controladores

- ❖ **Tasks logs**: Puedes acceder a los logs de tareas desde la pestaña Stages de Spark UI.

The screenshot shows the Spark UI interface with the 'Stages' tab selected. The 'Summary Metrics for 2 Completed Tasks' table is displayed, followed by 'Aggregated Metrics by Executor' and a list of 'Tasks (2)'. The 'Host' column in the tasks table is highlighted with a green box.

Metric	Min	25th percentile	Median	75th percentile
Duration	19 ms	19 ms	19 ms	19 ms
GC Time	0 ms	0 ms	0 ms	0 ms
Shuffle Read Size / Records	122.0 B / 4	122.0 B / 4	124.0 B / 4	124.0 B / 4

Executor ID	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Shuffle Re
0	10.139.64.4:37045	81 ms	2	0	0	2	246.0 B / 8

Index	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time
0	211	0	SUCCESS	PROCESS_LOCAL	0	10.139.64.4:37045	2021/10/24 13:32:44	19 ms	
1	212	0	SUCCESS	PROCESS_LOCAL	0	10.139.64.4:37045	2021/10/24 13:32:44	19 ms	

Figura 14.25 - Ubicación del Task log en la interfaz de usuario de Spark

- ❖ **Executors log**: Los logs de los ejecutores también están disponibles tanto en la pestaña de Stages, como se muestra en la captura de pantalla anterior, como en la pestaña de Executors, como se muestra en la siguiente captura de pantalla:

The screenshot shows the Spark UI interface with the 'Executors' tab selected. It displays a 'Summary' table and a list of 'Executors'. The 'Logs' column in the executors table is highlighted with a green box.

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Shuffle Input	Shuffle Read	Shuffle Write	Blacklisted
Active(1)	0	0.0 B / 1.6 GB	0.0 B	4	0	0	213	213	27 s (1 s)	3.8 KB	1.9 KB	1.9 KB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0
Total(1)	0	0.0 B / 1.6 GB	0.0 B	4	0	0	213	213	27 s (1 s)	3.8 KB	1.9 KB	1.9 KB	0

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Shuffle Input	Shuffle Read	Shuffle Write	Logs
0	10.139.64.4:37045	Active	0	0.0 B / 1.6 GB	0.0 B	4	0	0	213	213	27 s (1 s)	3.8 KB	1.9 KB	1.9 KB	stdout stderr

Figura 14.26 - Ubicación del registro del ejecutor en la interfaz de usuario de Spark

Comience con el driver log, y luego continúe con los task logs, seguido por los executor logs, para identificar cualquier error o advertencia que pueda estar causando el fracaso del job.

Puedes aprender más sobre la depuración de Spark jobs aquí: <https://docs.microsoft.com/en-us/azure/hdinsight/spark/apache-troubleshoot-spark>.

A continuación, vamos a ver cómo solucionar un fallo en la ejecución de un pipeline.

14.14. Solución de problemas de la ejecución de un pipeline fallido

Los pipelines de Azure Data Factory y Synapse proporcionan mensajes de error detallados cuando los pipelines fallan. A continuación se indican tres pasos sencillos para depurar un pipeline que ha fallado:

- ❖ **Compruebe los datasets:** Haga clic en Servicios vinculados y, a continuación, haga clic en el enlace Probar conexión para asegurarse de que los servicios vinculados funcionan bien y de que nada ha cambiado en el origen. A continuación se muestra un ejemplo de cómo utilizar la conexión de prueba en la página Editar servicio vinculado.

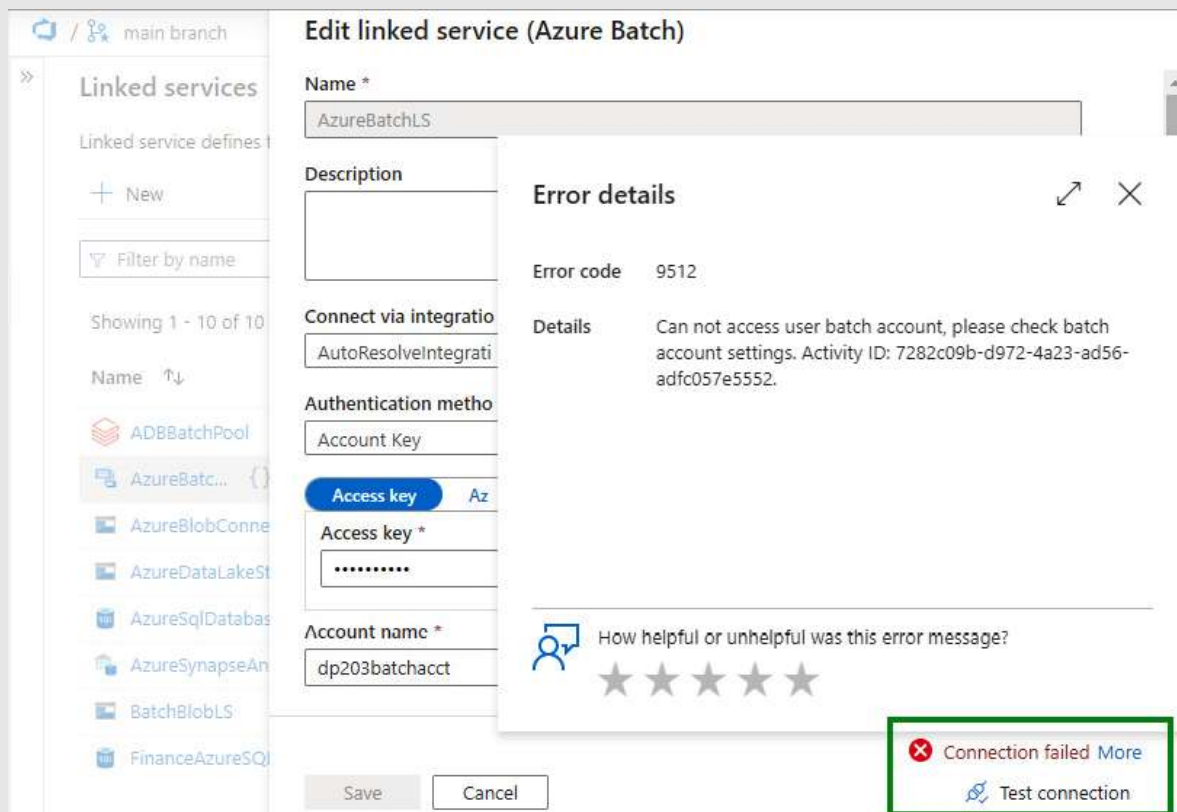


Figura 14.27 - Uso de la conexión de prueba para los servicios vinculados

- ❖ **Utilice las vistas previas de datos para comprobar sus transformaciones:** Active el modo de depuración del Data flow y compruebe las vistas previas de los datos para cada una de sus actividades del pipeline, empezando por el origen de los datos. Esto ayudará a reducir el problema. Este es un ejemplo de cómo utilizar la vista previa de datos:

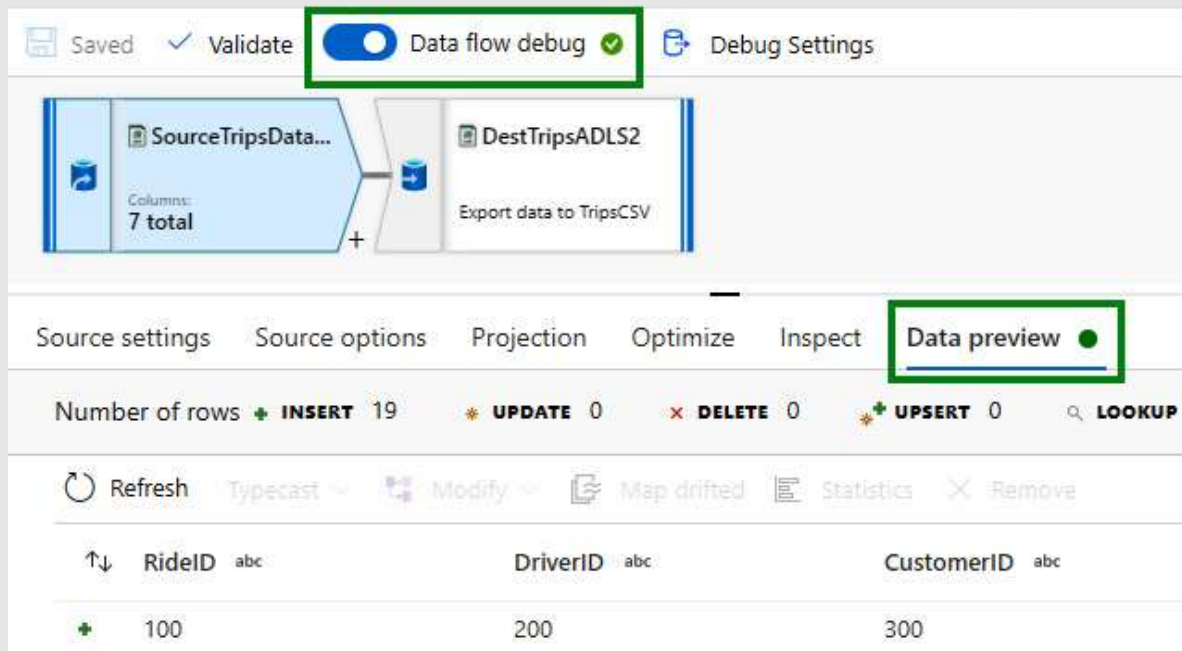


Figura 14.28 - Uso de la opción de vista previa de datos para ver si los datos se rellenan correctamente

- ❖ Si los problemas persisten, ejecute el pipeline y haga clic en la etiqueta de mensaje de error para ver cuál es el código de error. Este es un ejemplo:

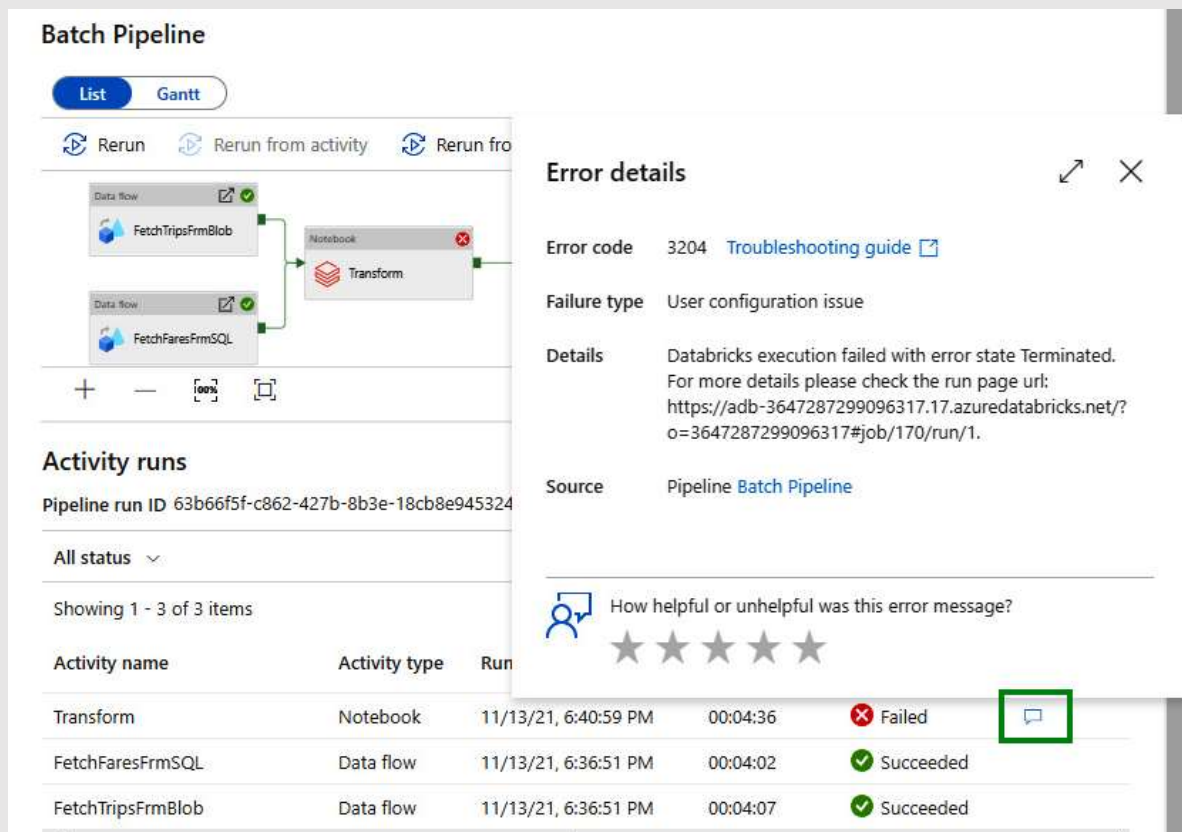


Figura 14.29 - Resolución de problemas desde la página de monitorización de tuberías

La siguiente guía de solución de problemas tiene los detalles de todos los códigos de error y recomendaciones sobre cómo solucionar los problemas: <https://docs.microsoft.com/en-us/azure/data-factory/data-factory-troubleshoot-guide>.

A continuación se muestra un ejemplo de código de error y recomendación de la guía de solución de problemas de Azure Data Factory:

Error code: 2105

Message: An invalid json is provided for property '%propertyName;'. Encountered an error while trying to parse: '%message;'.

Cause: The value for the property is invalid or isn't in the expected format.

Recommendation: Refer to the documentation for the property and verify that the value provided includes the correct format and type.

La depuración es una habilidad que se adquiere con la práctica. Utilice las directrices de este capítulo como punto de partida y practique con tantos ejemplos como sea posible para convertirse en un experto.

Con esto, hemos llegado al final de este capítulo.

Resumen

Al igual que el capítulo anterior, este capítulo también ha introducido muchos conceptos nuevos. Algunos de estos conceptos tomarán mucho tiempo para ser dominados, tales como la depuración de Spark, la optimización de las particiones shuffle, y la identificación y reducción de los desbordamientos de datos. Estos temas podrían ser libros separados por sí mismos. He hecho todo lo posible para darle una visión general de estos temas con enlaces de seguimiento. Por favor, revisa los enlaces para aprender más sobre ellos.

Recapitulemos lo que hemos aprendido en este capítulo. Comenzamos con la compactación de datos, ya que los archivos pequeños son muy ineficientes en el análisis de big data. Luego aprendimos sobre los UDFs, y cómo manejar los datos sesgados y los desbordamientos de datos tanto en SQL como en Spark. A continuación, exploramos las particiones aleatorias en Spark. Aprendimos a utilizar los indexadores y la caché para acelerar el rendimiento de nuestras consultas. También aprendimos sobre HTAP, que es un nuevo concepto que fusiona el procesamiento OLAP y OLTP. A continuación, exploramos los consejos generales de gestión de recursos para plataformas descriptivas y analíticas. Y, por último, terminamos viendo las pautas para depurar los trabajos de Spark y los fallos de los pipelines.

Ahora debería conocer las diferentes optimizaciones y técnicas de ajuste de consultas. Esto debería ayudarte tanto en la certificación como a la hora de convertirte en un buen ingeniero de datos de Azure.

Ahora ha completado todos los temas enumerados en el programa de estudios para la certificación DP-203. ¡Enhorabuena por tu persistencia en llegar al final!

El próximo capítulo cubrirá ejemplos de preguntas para ayudarle a preparar el examen.