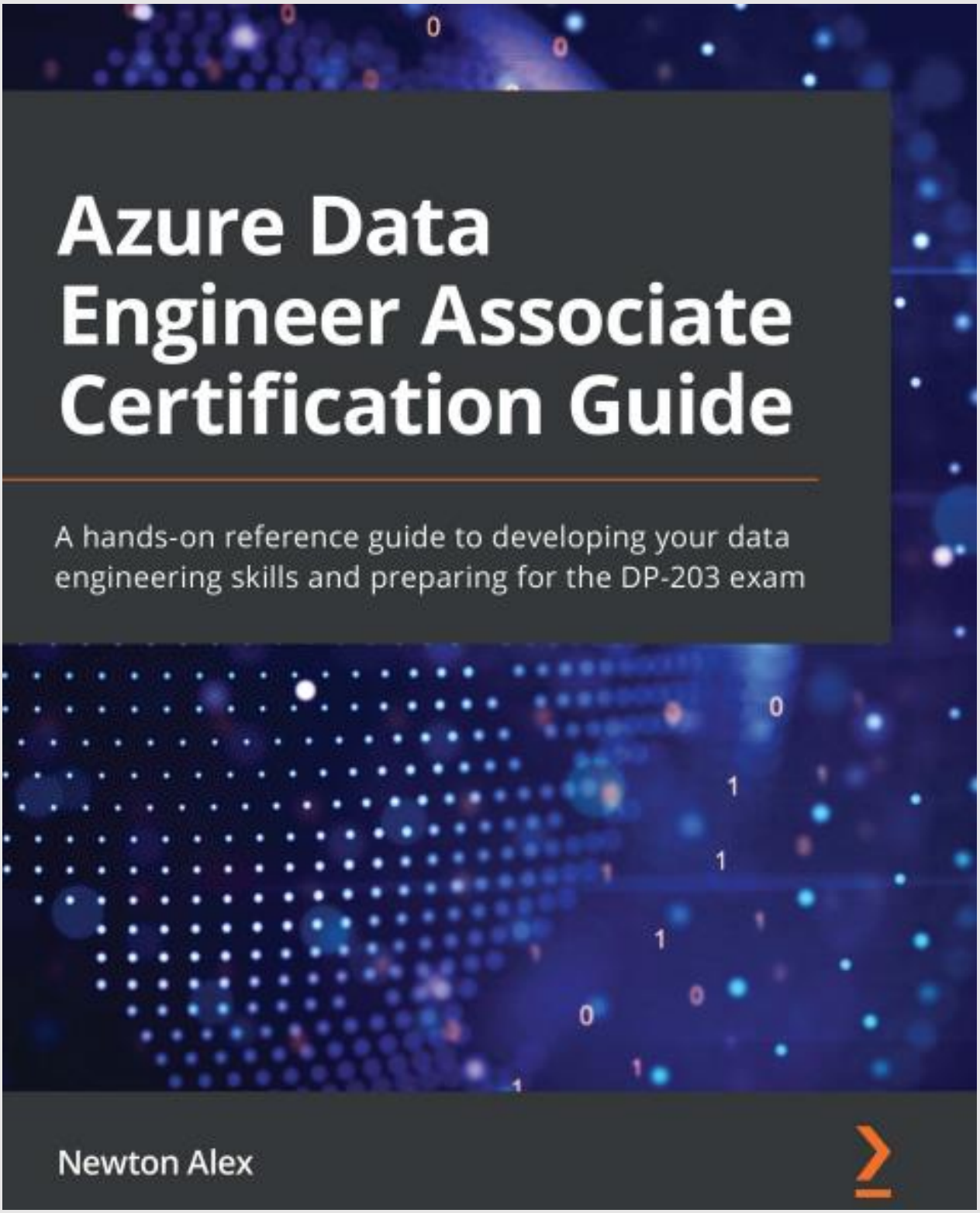

Azure Data Engineer Associate Certification Guide
A hands-on reference guide to developing your data engineering skills and preparing for the DP-203 exam
Newton Alex

Parte 2: Data Storage

Esta parte se sumerge en los detalles de los diferentes tipos de almacenamiento, las estrategias de partición de datos, los esquemas, los tipos de archivos, la alta disponibilidad, la redundancia, etc.

Esta sección comprende los siguientes capítulos:

- Capítulo 2, Diseño de una estructura de almacenamiento de datos
- Capítulo 3, Diseño de una estrategia de partición
- Capítulo 4, Diseño de la capa de servicio
- Capítulo 5, Implementación de estructuras físicas de almacenamiento de datos
- Capítulo 6, Implementación de estructuras lógicas de datos
- Capítulo 7, Implementación de la capa de servicio

Capítulo 2: Diseño de una estructura de almacenamiento de datos

Bienvenido al capítulo 2. En este capítulo nos centraremos en las tecnologías de almacenamiento de datos de Azure. Azure proporciona varias tecnologías de almacenamiento que pueden satisfacer una amplia gama de casos de uso en la nube e híbridos. Algunas de las tecnologías de almacenamiento de Azure más importantes son: Blobs, Files, Queues, Tables, SQL Database, Cosmos DB, Synapse SQL Warehouse y Azure Data Lake Storage (ADLS). Azure agrupa las cuatro tecnologías de almacenamiento fundamentales, a saber: Blobs, Files, Colas y Tablas, como Azure Storage. Otros servicios avanzados como Cosmos DB, SQL Database, ADLS, etc. se proporcionan como servicios Azure independientes.

A partir de este capítulo, seguiremos la secuencia exacta del programa de estudios del DP-203.

En este capítulo nos centraremos principalmente en los aspectos de diseño de las estructuras de almacenamiento. Los detalles de implementación correspondientes se cubrirán en los capítulos posteriores.

Este capítulo cubrirá los siguientes temas:

- Diseño de un data lake de Azure
- Selección de los tipos de archivo adecuados para el almacenamiento
- Elección de los tipos de archivo adecuados para las consultas analíticas
- Diseño del almacenamiento para una consulta eficiente
- Diseño del almacenamiento para la poda de datos
- Diseño de estructuras de carpetas para la transformación de datos
- Diseño de una estrategia de distribución
- Diseño de una solución de archivo de datos

Empecemos.

2.1. Requisitos técnicos

Para este capítulo, necesitará lo siguiente

- Una cuenta de Azure (gratuita o de pago)
- La interfaz de línea de comandos de Azure (Azure CLI) instalada en su área de trabajo. Por favor, consulte la sección, en Creación de una VM utilizando Azure CLI en el Capítulo 1, Introducción a los fundamentos de Azure para obtener instrucciones sobre la instalación de Azure CLI.

2.2. Diseño de un data lake de Azure

Si has estado siguiendo el dominio de las tecnologías de big data, seguro que te habrás encontrado con el término data lake. Los data lakes son almacenes de datos distribuidos que pueden contener volúmenes muy grandes de datos diversos. Pueden utilizarse para almacenar diferentes tipos de datos, como los estructurados, los semiestructurados, los no estructurados, los de flujo, etc.

Una solución de data lake suele constar de una **capa de almacenamiento**, una **capa de cómputo** y una **capa de servicio**. Las capas de cómputo pueden incluir el procesamiento de Extracción, Transformación y Carga (ETL), el procesamiento por lotes o el procesamiento de streaming. No hay plantillas fijas para crear data lakes. Cada data lake puede ser único y optimizado según los requisitos de la organización propietaria. Sin embargo, hay algunas directrices generales disponibles para construir data lakes eficaces, y las conoceremos en este capítulo.

2.2.1. ¿En qué se diferencia un data lake de un data warehouse?

La principal diferencia entre un data lake y un data warehouse es que un data warehouse almacena datos estructurados, mientras que un data lake puede utilizarse para almacenar diferentes formatos y tipos de datos. Los data lakes suelen ser las zonas de aterrizaje de diferentes fuentes y tipos de datos. Estos datos en bruto se procesan y los datos finalmente curados se cargan en almacenes de datos estructurados, como los data warehouses. El otro factor diferenciador principal es la escala. Los data lakes pueden almacenar fácilmente datos del orden de los Petabytes (PB), Exabytes (EB), o incluso superiores, mientras que los data warehouses pueden empezar a ahogarse en el rango de los PB.

2.2.2. ¿Cuándo se debe utilizar un data lake?

Podemos considerar el uso de data lakes para los siguientes escenarios:

- Si se tienen datos demasiado grandes para ser almacenados en sistemas de almacenamiento estructurados como data warehouses o bases de datos SQL.
- Cuando se tienen datos en bruto que necesitan ser almacenados para su posterior procesamiento, como un sistema ETL o un sistema de procesamiento por lotes
- Almacenamiento de datos continuos, como datos del Internet de las cosas (IoT), datos de sensores, tweets, etc., para escenarios de streaming de baja latencia y alto rendimiento
- Como zona de preparación (staging zone) antes de cargar los datos procesados en una base de datos SQL o en un data warehouse
- Almacenamiento de vídeos, audios, archivos binarios blob, archivos de registro y otros datos semiestructurados, como archivos de notación de objetos de JavaScript (JSON), lenguaje de marcado extensible (XML) o lenguaje de marcado YAML Ain't (YAML) para el almacenamiento a corto o largo plazo.

- Almacenamiento de datos procesados para tareas avanzadas como consultas ad hoc, aprendizaje automático (ML), exploración de datos, etc.

Veamos a continuación las diferentes zonas o regiones de un data lake.

2.2.3. Zonas del data lake

Un data lake puede dividirse a grandes rasgos en tres zonas en las que se llevan a cabo diferentes etapas de procesamiento, que se describen a continuación:

1. **Landing Zone o Raw Zone**: Aquí es donde se ingieren los datos en bruto desde diferentes fuentes de entrada.
1. **Transformation Zone**: Aquí es donde tiene lugar el procesamiento por lotes o por stream. Los datos sin procesar se convierten en un formato más estructurado y fácil de usar para la inteligencia empresarial (BI).
2. **Serving Zone**: Aquí es donde se almacenan los datos curados que pueden utilizarse para generar conocimientos e informes y se sirven a las herramientas de BI. Los datos de esta zona suelen ajustarse a esquemas bien definidos.

TIP

En las Landing zones, los datos se acumulan a partir de varias fuentes de entrada. Si la fuente es una entrada de streaming o una entrada de IoT, los archivos tienden a ser más pequeños. Se sabe que un gran número de archivos pequeños causa problemas de rendimiento en los data lakes. Por ejemplo, ADLS Gen2 recomienda tamaños de archivo de 256 Megabytes (MB) a 100 Gigabytes (GB) para un rendimiento óptimo. Si hay demasiados archivos pequeños, se recomienda fusionarlos (merge) en archivos más grandes.

Exploremos ahora algunos modelos estándar de data lake.

2.2.4. Arquitectura de data lake

La siguiente imagen muestra una arquitectura de data lake tanto para el procesamiento de lotes como de stream. El diagrama también incluye ejemplos de las tecnologías de Azure que pueden utilizarse para cada una de las zonas del data lake. Los nombres de los servicios enumerados por los iconos se presentan en la imagen siguiente:

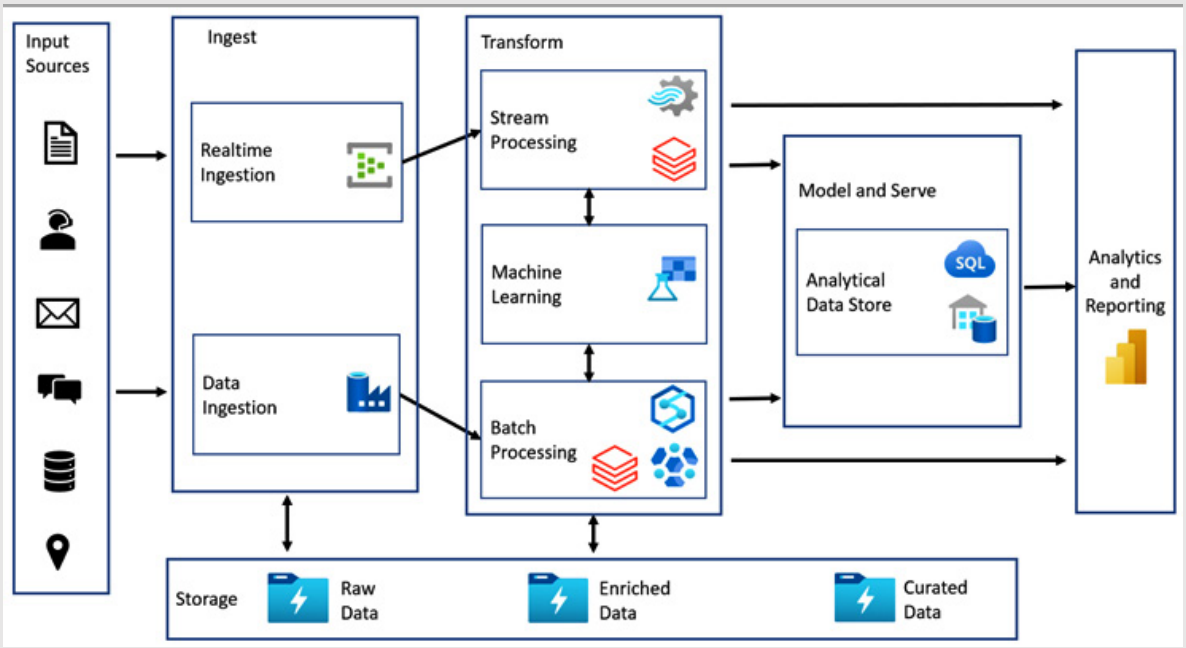


Figura 2.1 - Arquitectura de data lake con servicios de Azure

Estos son los nombres de los servicios representados por los iconos en el diagrama anterior:

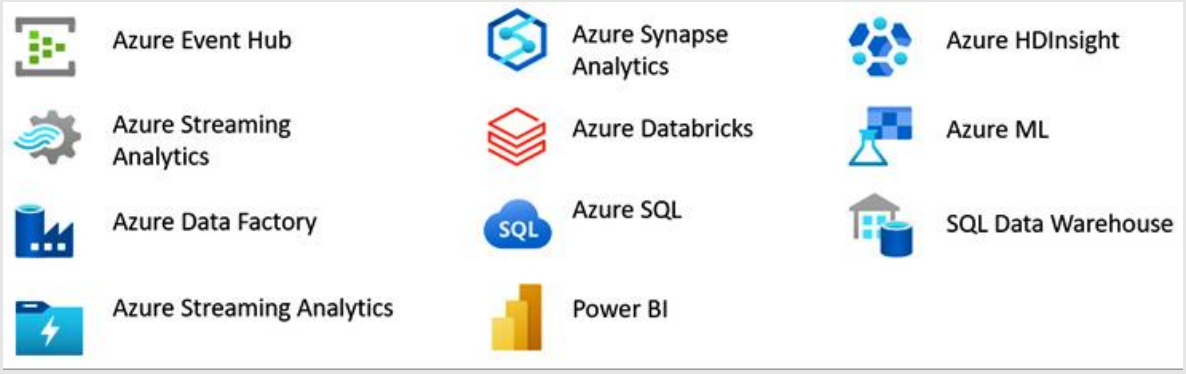


Figura 2.2 - Leyendas de los iconos

A lo largo de este libro aprenderemos más sobre estas tecnologías. A continuación, vamos a ver el procesamiento por lotes.

Procesamiento por lotes

En un framework de procesamiento por lotes, **los datos suelen llegar a la Landing Zone (o Raw Zone) desde varias fuentes de entrada**. Una vez que los datos llegan a la Landing Zone, el framework de orquestación activa los pipelines de procesamiento de datos. Los pipelines suelen tener múltiples etapas que llevarán los datos a través de varias etapas de procesamiento, como la limpieza de datos, el filtrado, la agregación de los datos de varias tablas y archivos, y finalmente la generación de datos curados para BI y exploración. Las etapas del pipeline pueden ejecutarse en

paralelo o de forma secuencial. Azure proporciona un servicio llamado Data Factory que puede ser utilizado para la construcción de tales pipelines. Azure Data Factory (ADF) es una herramienta muy versátil que también proporciona la capacidad de ingerir datos de una variedad de fuentes y realizar actividades simples de procesamiento de datos como uniones, filtrado, etc.

El procesamiento por lotes puede utilizarse para trabajos ETL, trabajos de preparación de datos, generación de informes periódicos, etc. Desde el punto de vista de la certificación, debe conocer las tecnologías de Azure que pueden utilizarse para crear los componentes por lotes de un data lake. En la siguiente tabla se enumeran algunas de las tecnologías de Azure más utilizadas para construir pipelines de procesamiento por lotes:

Storage Technologies	Azure Data Lake Gen2 Azure Blob Storage Azure CosmosDB Azure SQL Database
Data Transformation Technologies	Spark (via Azure Synapse, Azure HDInsight or Azure Databricks) Apache Hive (via Azure HDInsight) Apache Pig (via Azure HDInsight)
Analytical Datastore	Synapse SQL Warehouse (via Azure HDInsight) Apache HBase (via Azure HDInsight) Apache Hive (via Azure HDInsight)

Figura 2.3 - Tecnologías Azure disponibles para construir un sistema de procesamiento por lotes

A continuación vamos a aprender sobre el procesamiento de stream.

Procesamiento de stream o procesamiento en tiempo real

El procesamiento de stream se refiere al procesamiento de datos casi en tiempo real. A diferencia del procesamiento por lotes, que procesa los datos en reposo, el procesamiento por stream se ocupa de los datos a medida que llegan. Por ello, estos sistemas deben ser de baja latencia y alto rendimiento.

Por ejemplo, consideremos nuestro ejemplo de Imaginary Airport Cabs (IAC) que utilizamos en el Capítulo 1, Fundamentos de Azure. Es posible que queramos realizar un aumento de precios basado en la demanda de taxis, por lo que el sistema debe procesar las solicitudes de taxis procedentes de una zona geográfica concreta casi en tiempo real. Estos datos pueden ser agregados y utilizados para decidir el precio dinámico.

Por lo general, en los escenarios de procesamiento de stream, los datos llegan en pequeños archivos en rápida sucesión. **Estos pequeños archivos de datos se denominan mensajes**. El sistema de streaming suele realizar algunas comprobaciones rápidas para comprobar el formato correcto de los datos, los procesa y los escribe en un almacén. Por lo tanto, debemos asegurarnos de que los almacenes utilizados para el procesamiento en tiempo real admitan escrituras de gran volumen con baja latencia.

La siguiente tabla enumera algunas de las tecnologías de Azure que se pueden utilizar para construir un pipeline de procesamiento por streaming para un data lake:

Ingestion Tools	Azure Event Hub Azure IoT Hub Apache Kafka (via Azure HDInsight)
Stream Processing Tools	Azure Stream Analytics Spark Streaming (via Azure HDInsight or Azure Databricks) Apache Storm (via Azure HDInsight)
Analytical Datastore	Synapse SQL Warehouse Apache HBase (via Azure HDInsight) Apache Hive (via Azure HDInsight)

Figura 2.4 - Tecnologías de Azure disponibles para construir un sistema de procesamiento de streaming

A continuación, conozcamos **dos de las arquitecturas más comunes utilizadas en los data lakes**.

Arquitectura Lambda

Uno de los defectos de los sistemas de procesamiento por lotes es el tiempo que tardan en procesar los datos. Normalmente, un pipeline por lotes que procesa los datos de un día o de un mes puede tardar varias horas -o incluso días- en generar los resultados. Para superar este inconveniente, podemos utilizar una **arquitectura híbrida** llamada **arquitectura Lambda** que utiliza una combinación de **pipelines rápidos y lentos**. La **ruta lenta** procesa un mayor volumen de datos y produce resultados precisos, pero lleva más tiempo. La **ruta rápida**, en cambio, trabaja con un conjunto de datos mucho más pequeño (normalmente sampled data) y da un resultado aproximado mucho más rápido. La ruta rápida también puede utilizar diferentes tecnologías, como las de streaming, para acelerar el procesamiento. **Ambos pipelines alimentan una capa de servicio (Serving Layer)** que actualiza las actualizaciones incrementales de la ruta rápida en los datos de referencia de la ruta lenta. Cualquier cliente de análisis o herramienta de informes puede entonces acceder a los datos curados desde la capa de servicio.

Puede ver una visión general de la arquitectura Lambda en el siguiente diagrama:

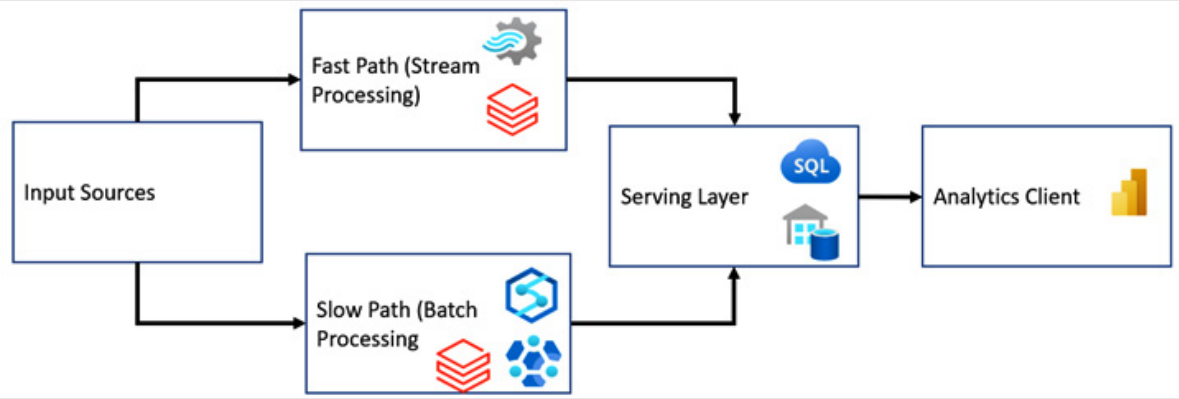


Figura 2.5 - Arquitectura Lambda

Este modelo Lambda nos ayudará a tomar decisiones informadas con mayor rapidez, en comparación con los modelos sólo por lotes.

Arquitectura Kappa

Kappa es una arquitectura alternativa a la arquitectura Lambda y **se centra únicamente en la ruta rápida o la ruta de streaming**. Se basa en el supuesto de que los datos considerados pueden representarse como data stream inmutable y que dichos streams puedan almacenarse durante largos periodos de tiempo en una solución de almacenamiento de datos con el mismo formato de streaming. Si necesitamos recalcular algún dato histórico, los datos de entrada correspondientes del data store pueden reproducirse a través de la misma capa de streaming para recalcular los resultados.

La principal ventaja de la arquitectura Kappa es la reducción de la complejidad, en comparación con la arquitectura Lambda, en la que implementamos dos pipelines. Así, se evita el doble procesamiento de los mismos datos, como ocurre en la arquitectura Lambda.

En la arquitectura Kappa, el componente de entrada es una cola de mensajes, como una cola de Apache Kafka o Azure Event Hubs, y todo el procesamiento se suele realizar a través de Azure Stream Analytics, Spark o Apache Storm.

Puede ver una visión general de la arquitectura de Kappa en el siguiente diagrama:

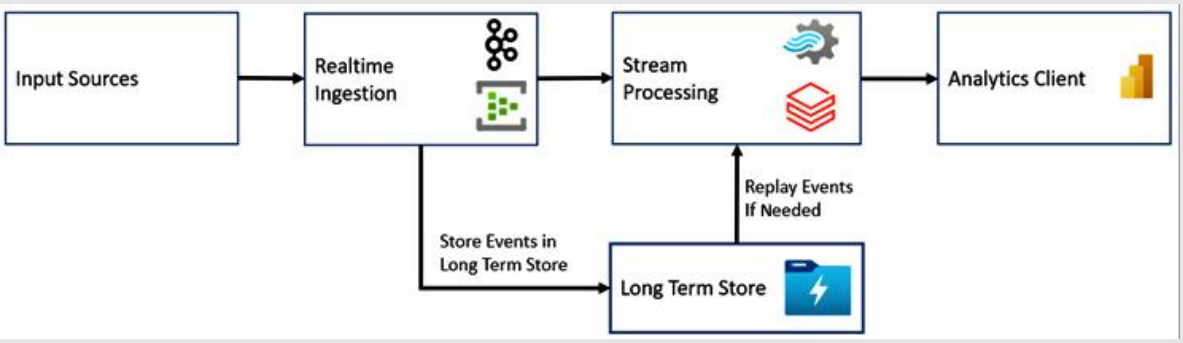


Figura 2.6 - Arquitectura Kappa

La arquitectura Kappa se puede utilizar para aplicaciones como el ML en tiempo real y aplicaciones en las que los datos de referencia no cambian muy a menudo.

2.2.5. Explorando las tecnologías de Azure que se pueden utilizar para construir un data lake

Ahora que entendemos algunas de las formas recomendadas para construir un data lake, vamos a explorar las tecnologías de Azure que se pueden utilizar para construir data lakes.

ADLS Gen2 o Blob storage

El componente más importante de un data lake es el de almacenamiento. La elección del almacenamiento adecuado podría ser la diferencia entre el éxito o el fracaso de un data lake. Azure proporciona una tecnología de almacenamiento especializada para construir data lakes, llamada ADLS Gen2. ADLS Gen2 suele ser una buena opción para la mayoría de los requisitos de almacenamiento de los data lake, pero también podemos optar por utilizar el almacenamiento Blob normal para el data lake si nuestros requisitos son principalmente para almacenar datos no estructurados y si no necesitamos listas de control de acceso (ACL) en los archivos y carpetas. Las ACLs permiten controlar qué usuarios tienen permisos de lectura, escritura o ejecución en los archivos y carpetas.

Azure Data Factory (ADF)

Una vez que tenemos el store decidido, necesitamos una forma de **mover los datos** automáticamente. Esta funcionalidad la cumple ADF. ADF es una herramienta muy potente que puede utilizarse para lo siguiente

- Ingesta de datos desde una amplia gama de entradas, como Azure Storage, Amazon Redshift, Google BigQuery, SAP HANA, Sybase, Teradata, protocolo genérico de transferencia de hipertexto (HTTP), protocolo de transferencia de archivos (FTP), etc.
- Transformaciones de datos sencillas, como uniones, agregaciones, filtrado y limpieza
- Programación de pipelines para mover los datos de una etapa a otra

Componentes de cómputo y procesamiento de datos

Ahora que hemos decidido dónde almacenar los datos, cómo obtenerlos y cómo moverlos, tenemos que decidir cómo procesar los datos para realizar análisis avanzados. Esto se puede lograr utilizando una variedad de tecnologías disponibles en Azure. Algunas de ellas se enumeran aquí:

- **Azure Synapse Analytics** para sus pools de SQL y Spark
- **Azure Databricks** para Spark

- **Azure HDInsight** para Hive, Spark y otras tecnologías de software de código abierto (OSS)
- **Azure Stream Analytics** para las necesidades de streaming

Puede que te preguntes: ¿Por qué Azure tiene tantas opciones para Spark? Bueno, Spark ha estado ganando mucha tracción últimamente como una de las herramientas analíticas más populares para los ingenieros de datos. Spark ofrece algo para cada desarrollador, ya sea procesamiento por lotes, streaming, ML, notebooks interactivos, etc., y Spark hace que sea muy fácil cambiar entre los diferentes casos de uso al proporcionar interfaces de codificación muy consistentes. Una vez que se sabe cómo escribir un programa por lotes, escribir un programa de streaming es muy similar, con una curva de aprendizaje muy poco profunda. Esta facilidad de uso y la compatibilidad con una amplia variedad de herramientas analíticas es lo que hace que Spark sea la opción preferida de la última generación de ingenieros de datos. Azure entiende esta tendencia y proporciona múltiples opciones de Spark. Proporciona la versión Azure Databricks para los clientes que adoran las características de Databricks Spark. Ofrece HDInsight Spark para los clientes que prefieren las tecnologías OSS, y también ofrece Synapse Spark, que es una versión de Spark OSS con un mayor rendimiento para aquellos clientes que prefieren una experiencia integrada de un solo panel dentro de Azure Synapse Analytics.

Exploremos un poco más estas tecnologías de cómputo.

Azure Synapse Analytics

Azure Synapse Analytics es el servicio analítico premium en el que Azure está invirtiendo bastante. Cuando oímos hablar de Azure Synapse Analytics, solemos pensar en él como un data warehouse de SQL pero, en realidad, Synapse Analytics es un conjunto completo de servicios analíticos integrados. Synapse Analytics tiene soporte integrado para varios almacenes de Azure, tecnologías de cómputo como SQL Data Warehouse y Synapse Spark, motores de orquestación como ADF, almacenes especializados como Cosmos DB, servicios de gestión de identidad y acceso (IAM) basados en la nube como Azure Active Directory (Azure AD), soporte de gobierno de datos a través de Azure Purview, y más.

Azure Synapse Analytics también hace las veces de almacén de análisis para la capa de servicio (Serving layer), ya que cuenta con un almacén SQL Warehouse altamente escalable en su núcleo. Esto puede ser utilizado para almacenar los datos procesados que se reducen en tamaño para ser utilizado para las consultas analíticas, conocimientos de datos, presentación de informes, y así sucesivamente.

Azure Databricks

Azure Databricks es la versión Databricks de Spark alojada en Azure. Azure Databricks proporciona una experiencia de Notebook muy rica en funcionalidades y está bien conectada con el resto de servicios de Azure, incluyendo Storage, ADF, Power BI, Authentication, y otras funcionalidades de Azure. Podemos utilizar Azure Databricks para escribir código o scripts de Spark que hagan el procesamiento de datos por nosotros. Por ejemplo, podemos utilizar Azure Databricks Spark para limpiar los datos de entrada, condensarlos a través de la filtración, la agregación o el muestreo, y añadir una estructura a los datos para que puedan ser utilizados para la analítica por sistemas similares a SQL.

Azure HDInsight

Azure también ofrece una opción para utilizar versiones de código abierto de Apache Spark, Apache Hive, Apache HBase, y más a través del producto HDInsight. Al igual que Azure Databricks, podemos utilizar la versión de código abierto de Spark o Hive para realizar el procesamiento de datos de nuestro data lake. La versión HDInsight de Spark utiliza Jupyter Notebooks. Este **es uno de los notebooks de código abierto más utilizados**. Hive, por otro lado, ha sido la tecnología de procesamiento de datos preferida hasta que Spark tomó el protagonismo. Hive sigue siendo uno de los servicios más desplegados en las plataformas de big data empresariales.

Azure Stream Analytics

El procesamiento de streaming se refiere al procesamiento rápido de los datos a medida que llegan. Por lo tanto, necesitamos tecnologías que proporcionen un alto rendimiento y capacidades de procesamiento casi en tiempo real. Azure ofrece opciones tanto propietarias como de código abierto para estos requisitos de procesamiento de streaming. **La tecnología de streaming nativa de Azure se llama Azure Stream Analytics (ASA)**. ASA funciona de forma nativa con todos los servicios de Azure Storage y puede conectarse directamente a herramientas de generación de informes como Power BI sin necesidad de un almacén de datos intermedio. Pero, **si se prefiere utilizar un servicio de código abierto para el streaming, podemos utilizar servicios como Apache Kafka y Apache Storm, disponibles a través de HDInsight**.

Informes y Power BI

Ahora que conocemos los componentes que se pueden utilizar para transformar los datos brutos del almacén del data lake en formas más curadas, tenemos que decidir cómo presentar estos datos. El último paso en el análisis de datos es la presentación de los datos de una manera que impulse las ideas de negocio o genere informes para indicar el estado de la empresa. En cualquier caso, necesitamos herramientas como Power BI para mostrar informes interactivos. Power BI es un conjunto de herramientas que pueden operar con big data y proporcionar una visión visual de los datos. Power BI tiene conectividad incorporada con Azure para integrarse perfectamente con servicios como Cosmos DB, Synapse Analytics, Azure Storage, etc.

Azure ML

Azure también proporciona algunos servicios avanzados como Azure ML para la analítica avanzada y la predicción. ML suele ser el siguiente paso en la progresión de un pipeline analítico. Una vez que se empieza a generar información a partir de los datos, el siguiente paso es predecir las tendencias futuras. Azure ML ofrece soporte para una amplia gama de algoritmos y modelos para el análisis de datos. Dado que ML no está en el programa de estudios de este libro, no vamos a profundizar en él, pero si quieres explorar el tema, puedes consultar los detalles en el siguiente enlace: https://azure.microsoft.com/en-in/services/machine-learning/.

Ahora deberías entender bastante bien cómo construir un data lake y qué tecnologías utilizar de Azure. Eso fue un montón de palabras clave y tecnologías para recordar. Si te sientes abrumado por todas las tecnologías y las palabras clave, no te preocupes: volveremos a hablar de estas tecnologías

a lo largo del libro. Para cuando lleguemos a la mitad del libro, tendrás un buen conocimiento de estas tecnologías.

2.3. Selección de los tipos de archivo adecuados para el almacenamiento

Ahora que entendemos los componentes necesarios para construir un data lake en Azure, tenemos que decidir los formatos de archivo que serán necesarios para el almacenamiento y la recuperación eficiente de los datos del data lake. Los datos suelen llegar en formatos como archivos de texto, archivos de registro, valores separados por comas **(CSV)**, **JSON**, **XML**, etc. Aunque estos formatos de archivo son más fáciles de leer y entender para los humanos, **puede que no sean los mejores formatos para el análisis de datos**. **Un formato de archivo que no se puede comprimir acabará pronto llenando las capacidades de almacenamiento; un formato de archivo no optimizado para las operaciones de lectura puede acabar ralentizando la analítica o los ETL; un archivo que no se puede dividir fácilmente no se puede procesar en paralelo. Para superar estas deficiencias, la comunidad de big data recomienda tres importantes formatos de datos: Avro**, **Parquet** y **Optimized Row Columnar (ORC)**. Estos formatos de archivo también son importantes desde el punto de vista de la certificación, por lo que en este capítulo exploraremos en profundidad estos tres formatos de archivo.

Elegir el tipo de archivo adecuado es fundamental, ya que **no será fácil cambiarlo más tarde**, cuando empiecen a llegar los datos. Cada uno de los formatos de archivo tiene sus propias ventajas y desventajas, pero rara vez hay un formato de archivo que se ajuste a todos los requisitos. Así que, en función de nuestros requisitos, podemos incluso elegir tener más de un formato de archivo dentro del data lake. Azure llama a esto persistencia políglota.

TIP

Intenta quedarte con uno o dos formatos de archivo, idealmente un formato de archivo por zona del data lake. Demasiados formatos de archivo pueden convertirse en una pesadilla de gestión, con varias copias duplicadas de los datos que se generan con el tiempo.

Hay que tener en cuenta múltiples factores a la hora de elegir un formato de archivo. A continuación se enumeran los más importantes:

- **Tipo de carga de trabajo**: Algunos formatos de archivo funcionan mejor con algunas herramientas que con otras; por ejemplo, **ORC funciona mejor con Hive, mientras que ORC y Parquet funcionan bien con Spark**. Por lo tanto, la tecnología de procesamiento de datos que planeamos adoptar también tendrá algo que decir a la hora de decidir el formato de archivo.

- **Coste**: Esta va a ser la principal preocupación para cualquier organización. ¿Cómo podemos mantener el coste bajo? Cuanto más almacenemos en el data lake, más acabaremos pagando por él. Por lo tanto, los formatos de archivo que soportan mejores ratios de compresión se vuelven importantes aquí.

- **Compresión**: Si sabemos que nuestro data lake no va a entrar nunca en rangos de PB o EB, entonces podemos decidirnos por formatos de datos que tengan un buen equilibrio entre compresión y rendimiento. Y, del mismo modo, si nuestro tamaño de almacenamiento va a ser enorme, definitivamente necesitamos un formato que soporte mayores niveles de compresión.

- **Rendimiento**: Aspectos como la velocidad de lectura, la velocidad de escritura, la posibilidad de dividir los archivos para procesarlos en paralelo y el acceso rápido a los datos relevantes repercutirán en el rendimiento de las herramientas analíticas. Por lo tanto, el requisito de rendimiento es otro punto clave a la hora de decidir el formato de archivo.

CONSEJO

Si todavía necesita almacenar los datos en alguno de los formatos semiestructurados como CSV, JSON, XML, etc., considere la posibilidad de comprimirlos utilizando la compresión Snappy.

Exploremos ahora los formatos de archivo en detalle.

2.3.1. Avro

Avro es un formato de almacenamiento basado en filas. Esto significa que almacena cada fila completa una tras otra en su almacenamiento de archivos. Por ejemplo, imaginemos que tenemos una tabla como ésta:

Driver ID	Name	License Number
111	Annie	A1234
222	Brian	B5678
333	Charlie	C3456

Figura 2.7 - Una tabla que muestra el formato Avro

Avro la almacenará lógicamente, como se muestra en la siguiente captura de pantalla. En realidad, estaría añadiendo los metadatos necesarios para la decodificación del archivo en el mismo archivo:

111	**Annie**	**A1234**	**222**	**Brian**	**B5678**	**333**	**Charlie**	**C3456**

Figura 2.8 - Una fila Avro de ejemplo, incluyendo los metadatos

Este formato es bueno para las **cargas** de trabajo transaccionales de **escritura intensiva**, como los trabajos ETL que necesitan leer archivos enteros para procesar los datos.

INFORMACIÓN SOBRE AVRO

Puede encontrar más información sobre Avro en el siguiente enlace: https://avro.apache.org.

2.3.2. Parquet

Parquet, por su parte, es un formato basado en columnas. Esto significa que almacena los datos de cada columna relacionada uno tras otro en su archivo. Por ejemplo, si consideramos la misma tabla del ejemplo de Avro, Parquet la almacenará lógicamente como se muestra en la siguiente captura de pantalla. Al igual que con Avro, Parquet también almacenará algunos metadatos junto con los datos reales en su archivo:

111	222	333	Annie	Brian	Charlie	A1234	B5678	C3456

Figura 2.9 - Una fila de Parquet de ejemplo

Este almacenamiento basado en columnas hace que Parquet sea excepcionalmente bueno para trabajos de **lectura intensiva** como las cargas de trabajo analíticas, ya que generalmente consultan sólo un subconjunto de columnas para su procesamiento. Esto garantiza que no tengamos que leer filas enteras sólo para procesar una pequeña subsección de esas filas, a diferencia de lo que ocurre con los formatos de datos orientados a filas. Proyectos como Apache Spark y Apache Drill son los más compatibles con Parquet y pueden aprovechar las características de optimización de consultas como Predicate Pushdowns mientras trabajan con Parquet.

INFORMACIÓN SOBRE PARQUET

Puede encontrar más información sobre Parquet en el siguiente enlace: https://parquet.apache.org.

2.3.3. ORC

ORC también es un formato basado en columnas similar a Parquet y funciona muy bien con cargas de trabajo analíticas por esa razón. **Proyectos como Apache Hive son los más compatibles con ORC**, ya que ORC soporta de forma nativa algunos de los tipos de datos de Hive y permite características como el soporte de atomicidad, consistencia, aislamiento y durabilidad (ACID) y los pushdowns de predicados en Hive.

INFORMACIÓN SOBRE ORC

Puede encontrar más información sobre ORC en el siguiente enlace: https://orc.apache.org.

2.3.4. Comparación de Avro, Parquet y ORC

Comparemos los tres formatos de archivo desde el punto de vista de los parámetros básicos de evaluación necesarios para el almacenamiento de data lake, como se indica a continuación:

Features	AVRO	Parquet	ORC
READ Performance	Low	High	High
WRITE Performance	High	Low	Low
Ability to Split files for Parallel Processing	Yes	Yes	Yes (Best)
Schema Evolution Support	Yes (Best)	Yes	Yes

Figura 2.10 - Comparación de formatos de archivo

Ahora que entendemos los fundamentos de los formatos de archivo, aquí hay algunos consejos para elegir el formato de archivo para cada una de las zonas descritas en la sección de zonas de data lake:

Landing Zone or Raw Zone

- Si el tamaño de los datos va a ser enorme (en terabytes (TB), PB, o superior), necesitarás una buena compresión, así que opta por Avro o por texto comprimido utilizando tecnologías como la compresión Snappy.

- Si piensa tener muchos trabajos ETL, opte por Avro.

- Si piensa tener un solo formato de archivo para facilitar el mantenimiento, Parquet sería un buen compromiso tanto desde el punto de vista de la compresión como del rendimiento.

Transformation Zone

- Si va a utilizar principalmente Hive, elija ORC.

- Si va a utilizar principalmente Spark, elija Parquet.

2.4. Elección de los tipos de archivo adecuados para las consultas analíticas

En la sección anterior, hemos analizado en detalle los tres formatos de archivo: Avro, Parquet y ORC. **Cualquier formato que admita lecturas rápidas es mejor para las cargas de trabajo analíticas**. Así que, naturalmente, los formatos basados en columnas, como **Parquet** y **ORC**, **son los más adecuados.**

En función de las cinco áreas principales que hemos comparado antes (lectura, escritura, compresión, evolución del esquema y capacidad de dividir archivos para el procesamiento en paralelo) y de su elección de tecnologías de procesamiento, como Hive o Spark, podría seleccionar ORC o Parquet.

Por ejemplo, considere lo siguiente:

- **Si tienes cargas de trabajo basadas en Hive o Presto, elige ORC.**
- **Si tiene cargas de trabajo basadas en Spark o Drill, elija Parquet.**

Ahora que conoce los diferentes tipos de formatos de archivo disponibles y los que debe utilizar para las cargas de trabajo analíticas, pasemos al siguiente tema: el diseño para una consulta eficiente.

2.5. Diseñando el almacenamiento para una consulta eficiente

Para diseñar para una consulta eficiente, tendremos que entender las diferentes tecnologías de consulta que están disponibles en Azure. Aunque este capítulo se centra principalmente en las tecnologías de almacenamiento, un pequeño desvío hacia el mundo de las consultas nos ayudará a entender mejor el proceso de diseño para una consulta eficiente. Hay dos tipos de servicios de consulta disponibles: Los basados en SQL, como Azure SQL, Synapse Serverless/Dedicated Pools (antes conocido como SQL Warehouse), y los motores analíticos de big data, como Spark y Hive. Vamos a explorar las técnicas de diseño importantes para estos dos grupos de motores de consulta.

A grandes rasgos, podemos agrupar las técnicas disponibles para realizar consultas eficientes en las tres capas siguientes:

- **Capa de almacenamiento**-Utilizando técnicas como la partición, la poda de datos (*data pruning*) y la consistencia eventual.
- **Capa de aplicación**: utilización de técnicas como el almacenamiento en caché de datos (*data caching*) y el ajuste de la aplicación (*application tuning*) (como la variación del tamaño de los contenedores o el aumento del paralelismo).
- **Capa de consulta**: utilización de técnicas especializadas, como la indexación y las vistas materializadas, disponibles en algunos de los servicios.

No te preocupes por todas las nuevas palabras clave en esta sección. Exploraremos todas estas tecnologías en detalle en las siguientes secciones. Comencemos con la capa de Almacenamiento.

2.5.1. Capa de almacenamiento

Diseñar una estrategia de partición correctamente desde el principio es importante porque una vez que implementamos las particiones, será difícil cambiarlas en un momento posterior. Cambiar las particiones en un momento posterior requerirá transferencias de datos, modificaciones en las consultas, cambios en las aplicaciones, etc. Una estrategia de partición para un tipo de consulta podría no funcionar para otra consulta, por lo que deberíamos centrarnos en diseñar una estrategia de partición para nuestras consultas más críticas. Veamos los puntos importantes que hay que tener en cuenta al diseñar las particiones.

Particiones

Una partición se refiere a la forma en que dividimos los datos y los almacenamos. Las mejores particiones son aquellas que pueden ejecutar consultas paralelas sin requerir demasiadas transferencias de datos entre particiones. Por lo tanto, dividimos los datos de tal manera que los datos se reparten uniformemente y los datos relacionados se agrupan dentro de las mismas particiones. Las particiones son un concepto muy importante desde la perspectiva de la certificación y, como tal, hay un capítulo entero dedicado a las particiones (Capítulo 3, Diseño de una estrategia de partición). Por lo tanto, vamos a centrarnos aquí en una visión general de alto nivel.

A continuación, se comentan algunos conceptos importantes de partición que pueden ayudar a acelerar el rendimiento de las consultas.

Replicación de datos

Los datos estáticos más pequeños y de uso más frecuente, como los datos de búsqueda o los datos del catálogo, pueden replicarse en las particiones. Esto ayudará a reducir el tiempo de acceso a los datos y, a su vez, acelerará las consultas considerablemente.

Reducing cross-partition operations and joins

Minimizar los cross-partition joins ejecutando trabajos en paralelo dentro de cada partición y agregando sólo los resultados finales ayudará en el acceso a los datos entre particiones. Cualquier acceso entre particiones es una operación costosa, por lo que intentar realizar el mayor procesamiento de datos dentro de la misma partición antes de exportar sólo los datos filtrados necesarios ayudará a mejorar el rendimiento general de las consultas.

Poda de datos (Data pruning)

La poda de datos o exclusión de datos se refiere al proceso de ignorar los datos innecesarios durante la consulta y, por tanto, reducir las operaciones de entrada/salida (I/O). Este es también un tema importante para la certificación, por lo que tenemos una sección completa dedicada a la poda de datos más adelante en este capítulo.

Consistencia eventual

Los sistemas de almacenamiento soportan diferentes niveles de consistencia. Los almacenes en la nube suelen mantener copias redundantes de sus datos en varios servidores. **La consistencia se refiere a la rapidez con la que todas las copias internas de sus datos reflejarán un cambio realizado en la copia primaria**. Los almacenes de datos suelen soportar una consistencia fuerte o una consistencia eventual, pero algunos almacenes como Cosmos DB incluso soportan varios niveles entre la consistencia fuerte y la eventual. Si el sistema de almacenamiento es fuertemente consistente, entonces asegura que todas las copias de datos se actualicen antes de que el usuario pueda realizar cualquier otra operación. Por otro lado, si el sistema de almacenamiento es eventualmente consistente, entonces el almacenamiento deja que los datos de cada una de las copias se actualicen gradualmente a lo largo del tiempo. **Las consultas que se ejecutan en almacenes de datos fuertemente consistentes tienden a ser más lentas**, ya que el sistema de almacenamiento se asegurará de que todas las escrituras (en todas las copias) se completen antes de que la consulta pueda regresar. Sin embargo, **en los sistemas eventualmente consistentes, la consulta regresará inmediatamente después de que la primera copia haya terminado**, y el resto de las actualizaciones de todas las demás copias se producen de forma

asíncrona. Por lo tanto, si tu sistema puede tolerar la consistencia eventual, tenderá a ser más rápido.

A continuación, veamos los componentes de la capa de aplicación.

2.5.2. Capa de aplicación

Las optimizaciones de la capa de aplicación se ocupan de la eficiencia con la que utilizamos los componentes principales, como la unidad central de procesamiento (CPU), la memoria, la red, etc. Veamos algunas formas de mejorar el rendimiento en la capa de aplicación.

Ajuste de las aplicaciones

Los servicios analíticos de big data como Spark y Hive pueden optimizarse configurando el número de ejecuciones paralelas y otros atributos como la memoria, la CPU, el ancho de banda de la red y el tamaño de los contenedores. A continuación se exponen algunos puntos que hay que tener en cuenta para afinar dichos servicios:

- Considere el tamaño adecuado de las máquinas virtuales (VM). Seleccione máquinas virtuales que tengan suficiente memoria, CPU, disco y ancho de banda de red para sus aplicaciones.

- Las máquinas virtuales más grandes pueden soportar más contenedores y, por tanto, aumentar la paralelización, pero esto está sujeto a la capacidad de la aplicación para escalar con el aumento de la paralelización. Encuentre el punto óptimo entre la capacidad de la aplicación para escalar y los tamaños de las máquinas virtuales.

- Por ejemplo, consideremos Spark. Cada uno de los worker containers de Spark se llama ejecutor. Puede considerar experimentar con las siguientes configuraciones para los ejecutores:

- **Num-executors**-Cuántos ejecutores desea ejecutar en cada máquina.

- **Executor-memory**-Cuánta memoria quieres asignar a cada ejecutor para que no se quede sin memoria. Esto suele depender del tamaño y la inclinación de los datos. En función de la cantidad de datos que vaya a procesar cada ejecutor, hay que configurar la memoria.

- **Executor-cores**-Cuántos núcleos de CPU quiere asignar a cada ejecutor. Si tu trabajo es más intensivo en cómputo, entonces añadir más núcleos a cada ejecutor podría acelerar el procesamiento.

Veamos ahora cómo el almacenamiento en caché de datos (caching data) puede ayudar a acelerar las consultas.

Almacenamiento en caché

El almacenamiento en caché consiste en guardar los datos intermedios en capas de almacenamiento más rápidas para acelerar las consultas. Podemos utilizar servicios externos, como la caché de Redis, para almacenar los datos a los que se accede con frecuencia en las consultas para ahorrar en latencias de lectura. También podemos activar las opciones de caché incorporadas, como la caché de Resultset disponible en tecnologías como Synapse SQL, para acelerar el rendimiento de las consultas.

A continuación, analicemos con más detalle la capa de consulta.

2.5.3. Capa de consulta

La capa de consulta también forma parte, técnicamente, de la capa de aplicación, pero la optimización de las consultas es un área de interés en sí misma. Hay miles de artículos académicos dedicados al área de optimización de consultas. Por lo tanto, vemos las opciones disponibles para la optimización de consultas como una sección separada en sí misma en el Capítulo 14, Optimización y resolución de problemas de almacenamiento y procesamiento de datos.

Indexación

Una de las técnicas más importantes para realizar consultas eficientes es la indexación. Si has utilizado alguna tecnología SQL antes, puede que hayas oído hablar de indexar tablas basándose en ciertas columnas clave. Los índices son como las claves de un **HashMap** que pueden utilizarse para acceder directamente a una fila concreta sin tener que escanear toda la tabla.

En los sistemas basados en SQL, puede ser necesario acceder a las filas utilizando valores distintos de la clave primaria. En estos casos, el motor de consulta necesita escanear todas las filas para encontrar el valor que buscamos. En cambio, si podemos definir un índice secundario basado en los valores de columna más buscados, podríamos evitar los escaneos completos de la tabla y acelerar la consulta. Los índices secundarios se calculan por separado de los índices primarios de la tabla, pero lo hace el mismo motor SQL.

En Azure, disponemos de tecnologías que pueden realizar la indexación de enormes volúmenes de datos. Estos índices pueden ser utilizados por motores analíticos como Spark para acelerar las consultas. Una de estas tecnologías que ofrece Azure se llama **Hyperspace**.

Hyperspace nos permite crear índices en conjuntos de datos de entrada como Parquet, CSV, etc., que pueden utilizarse para optimizar las consultas. La indexación de Hyperspace debe ejecutarse por separado para crear un índice inicial. Después, se puede actualizar de forma incremental para los nuevos datos. Una vez que tenemos el índice Hyperspace, cualquier consulta de Spark puede aprovechar el índice, de forma similar a como usamos los índices en SQL.

Vistas materializadas

Las vistas son proyecciones lógicas de datos de múltiples tablas. Las vistas materializadas no son más que versiones prepobladas de dichas vistas. Si una consulta necesita hacer merges y joins complejos de múltiples tablas o múltiples particiones, puede ser beneficioso realizar dichos merges y joins de antemano y mantener los datos listos para que la consulta los consuma. Las vistas materializadas pueden generarse a intervalos regulares y mantenerse listas antes de que se ejecute la consulta real.

A efectos prácticos, las vistas materializadas no son más que cachés especializados para las consultas. Los datos de las vistas materializadas son datos de sólo lectura y pueden regenerarse en cualquier momento. No se persiguen permanentemente en el almacenamiento.

Ahora que hemos considerado cómo las necesidades de consulta eficiente pueden influir en nuestros diseños, vamos a ampliar esa consideración a la poda de datos.

2.6. Diseñar el almacenamiento para la poda de datos (partición)

La poda de datos (data pruning), como su nombre indica, se refiere a la poda o recorte de los datos innecesarios para que las consultas no tengan que leer todo el conjunto de datos de entrada. La I/O es uno de los principales cuellos de botella de cualquier motor de análisis, por lo que la idea es que, reduciendo la cantidad de datos leídos, podemos mejorar el rendimiento de las consultas. La poda de datos suele requerir algún tipo de aportación del usuario al motor analítico para que éste pueda decidir qué datos pueden ignorarse con seguridad para una consulta concreta.

Tecnologías como Synapse Dedicated Pools, Azure SQL, Spark y Hive ofrecen la posibilidad de dividir los datos en función de criterios definidos por el usuario. Si podemos organizar los datos de entrada en carpetas físicas que se corresponden con las particiones, podemos omitir de forma efectiva la lectura de carpetas enteras de datos que no son necesarias para dichas consultas.

Veamos los ejemplos de Synapse Dedicated Pool y Spark, ya que son importantes desde el punto de vista de la certificación.

2.6.1. Ejemplo de Dedicated SQL pool con poda (with pruning)

Consideremos un ejemplo muy sencillo que cualquier persona con conocimientos sencillos de SQL entenderá. Una vez que entremos en los detalles de la implementación en los capítulos posteriores, exploraremos muchas de las características avanzadas proporcionadas por Dedicated SQL Pool.

Aquí, creamos una tabla trip para nuestro escenario IAC. Sólo para refrescar su memoria, IAC es nuestro escenario de cliente de ejemplo que estamos ejecutando a lo largo de este libro.

La tabla ha sido particionada usando la palabra clave PARTITION en tripDate, como se ilustra en el siguiente fragmento de código:

```
CREATE TABLE dbo.TripTable

(
   [tripId] INT NOT NULL,
   [driverId] INT NOT NULL,
   [customerID] INT NOT NULL,
   [tripDate] INT,
   [startLocation] VARCHAR(40),
   [endLocation] VARCHAR(40)
)
WITH
(
   PARTITION ([tripDate] RANGE RIGHT FOR VALUES
      ( 20220101, 20220201, 20220301 )
  )
)
```

La sintaxis **RANGE RIGHT** especificada sólo asegura que los valores especificados en la sintaxis **PARTITION** pertenecerán cada uno al lado derecho del rango. En este ejemplo, las particiones tendrán el siguiente aspecto

```
Partition 1: All dates < 20220101
Partition 2: 20220101 to 20220131
Partition 3: 20220201 to 20220228
Partition 4: 20220301 to 20220331
```

Si hubiéramos dado **RANGE LEFT**, se habrían creado particiones como esta

```
Partition 1:  All dates < 20220102
Partition 2: 20220102 to 20220201
Partition 3: 20220202 to 20220301
Partition 4: All dates > 20220301
```

Ahora, digamos que necesitamos encontrar todos los clientes que viajaron con IAC en el mes de enero. Lo único que hay que hacer es utilizar un filtro simple, como en el siguiente ejemplo

```
SELECT customerID FROM TripTable
WHERE tripDate BETWEEN '20220101' AND '20220131'
```

Si no fuera por la partición, la consulta tendría que escanear todos los registros de la tabla para encontrar las fechas dentro del rango de '20220101' Y '20220131'. Sin embargo, con la partición, la consulta sólo recorrerá los registros de la partición "20220101". Esto hace que la consulta sea más eficiente.

2.6.2. Ejemplo de Spark con pruning

Veamos cómo podemos implementar la poda de datos para Spark. En este ejemplo, vamos a crear un simple Data Frame de Spark y lo escribiremos en particiones de fechas como "año/mes/día". Luego, veremos cómo leer sólo de una partición de datos requerida. Procederemos de la siguiente manera:

1. Vamos a crear algunos datos de ejemplo utilizando un simple array, como sigue:

```
columnNames = ["tripID", "driverID", "customerID", "cabID", "date", "startLocation",
"endLocation"]

tripData = [

 ('100', '200', '300', '400', '20220101', 'Nueva York', 'Nueva Jersey'),
 ('101', '201', '301', '401', '20220102', 'Tempe', 'Phoenix') ]
```

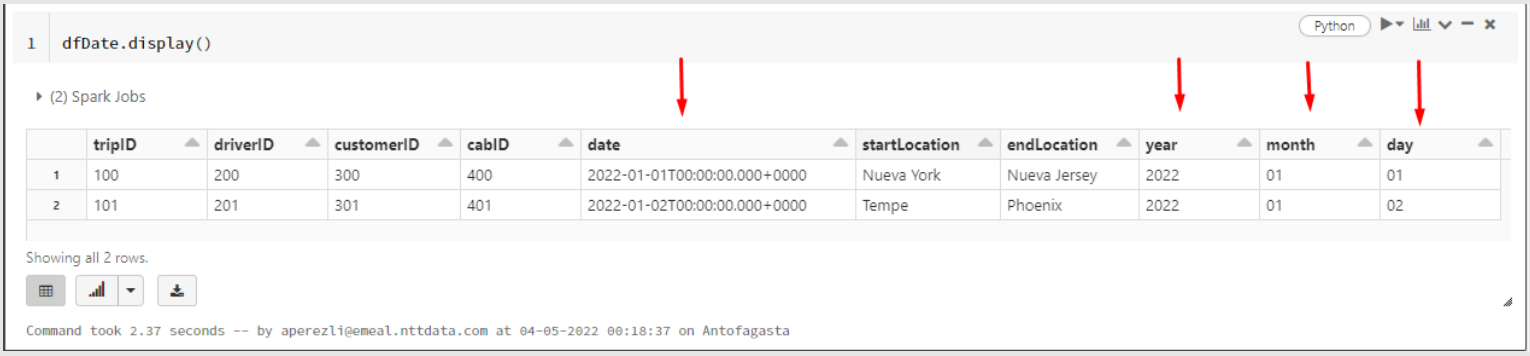
2. Crea un DataFrame con los datos anteriores, así

```
df = spark.createDataFrame(data= tripData, schema = columnNames)
```

3. Dado que el tripDate está en formato de fecha simple, vamos a dividirlo en año, mes y día, de la siguiente manera

```
from pyspark.sql.functions import to_timestamp, date_format, col

dfDate = df.withColumn("date", to_timestamp(col("date"), 'yyyyMMdd')) \
      .withColumn("year", date_format(col("date"), "yyyy")) \
      .withColumn("month", date_format(col("date"), "MM")) \
      .withColumn("day", date_format(col("date"), "dd"))
```



```
1  dfDate.display()
```

Python

▸ (2) Spark Jobs

	tripID	driverID	customerID	cabID	date	startLocation	endLocation	year	month	day
1	100	200	300	400	2022-01-01T00:00:00.000+0000	Nueva York	Nueva Jersey	2022	01	01
2	101	201	301	401	2022-01-02T00:00:00.000+0000	Tempe	Phoenix	2022	01	02

Showing all 2 rows.

Command took 2.37 seconds -- by aperezli@emeal.nttdata.com at 04-05-2022 00:18:37 on Antofagasta

4. Ahora, reparte los datos en memoria, así:

```
dfDate = dfDate.repartition("year", "month", "day")
```

5. Y, finalmente, escríbalo en diferentes archivos bajo el directorio de salida abfs://IAC/Trips/Out. Aquí, abfss se refiere al controlador de Azure Blob File System. El código se ilustra en el siguiente fragmento:

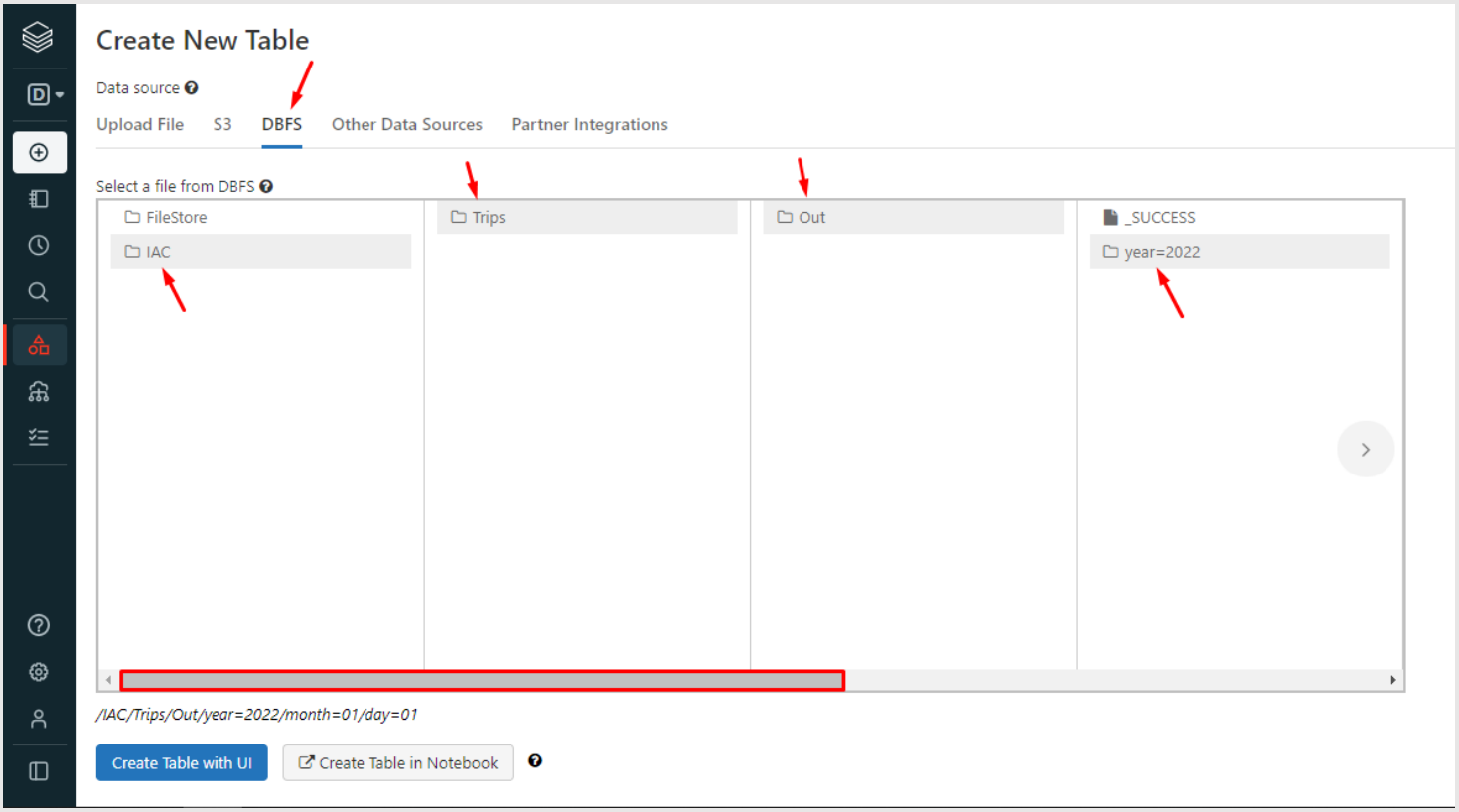
```
dfDate.write.partitionBy("year", "month", "day").parquet("abfss://IAC/Trips/Out")
```

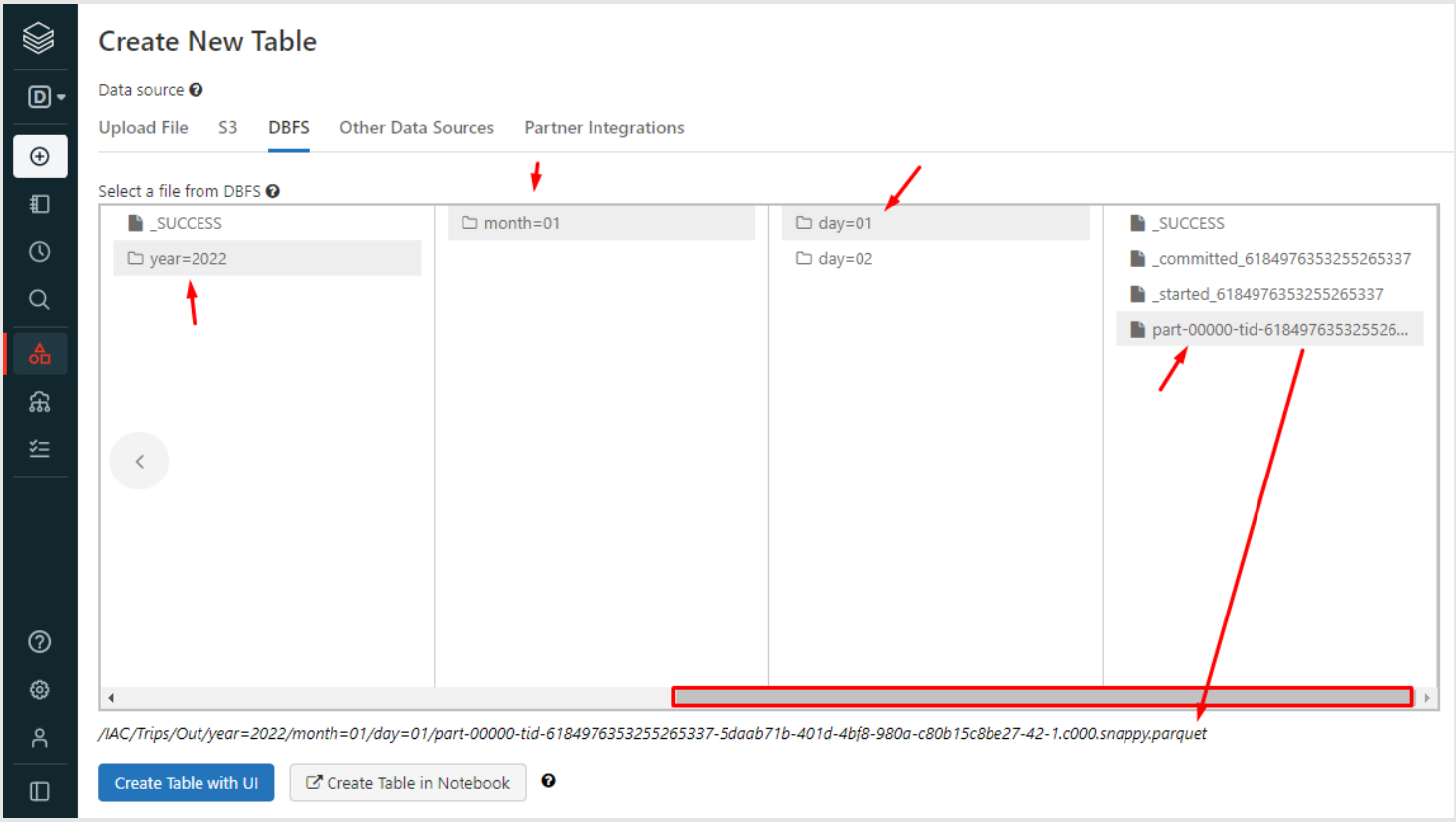
Así lo hice en Databricks Community:

```
1  dfDate.write.partitionBy("year", "month", "day").parquet("dbfs:/IAC/Trips/Out")
```

▸ (2) Spark Jobs

Command took 10.73 seconds -- by aperezli@emeal.nttdata.com at 04-05-2022 00:26:19 on Antofagasta


Create New Table
Data source
Upload File
S3
DBFS
Other Data Sources
Partner Integrations
Select a file from DBFS
FileStore
IAC
Trips
Out
_SUCCESS
year=2022
/IAC/Trips/Out/year=2022/month=01/day=01
Create Table with UI
Create Table in Notebook


Create New Table
Data source
Upload File
S3
DBFS
Other Data Sources
Partner Integrations
Select a file from DBFS
_SUCCESS
year=2022
month=01
day=01
day=02
_SUCCESS
_committed_6184976353255265337
_started_6184976353255265337
part-00000-tid-618497635325526...
/IAC/Trips/Out/year=2022/month=01/day=01/part-00000-tid-6184976353255265337-5daab71b-401d-4bf8-980a-c80b15c8be27-42-1.c000.snappy.parquet
Create Table with UI
Create Table in Notebook

6. En este punto, nuestro directorio de salida se creará con la siguiente estructura:

```
abfss://IAC/Trips/Out/
            year=2022/
                  day=01/
                        part*.parquet
```

7. Veamos ahora cómo funciona la poda. Por ejemplo, si ejecutamos la siguiente consulta, Spark leerá inteligentemente sólo la carpeta year="2022/month=01/day=03" sin que tengamos que escanear todos los datos de la carpeta IAC/Trips/Out:

```
readDF = spark.read.parquet("abfss://IAC/Trips/Out/year=2022").filter("month=01", "day=02")
```

Como habrás observado, la partición de los datos mediante una buena estructura de carpetas puede ayudar a mejorar la eficiencia de las consultas. Este es un buen punto de partida para nuestro siguiente tema, que explica cómo diseñar estructuras de carpetas para data lakes.

Así lo pude realizar en Databrick Community:

```
readDF = spark.read.parquet("dbfs:/IAC/Trips/Out/year=2022").filter("month=01").filter("day=02")
```

2.7. Diseño de estructuras de carpetas para la transformación de datos

Teniendo en cuenta la variedad y el volumen de datos que aterrizarán en un data lake, es muy importante diseñar una estructura de carpetas flexible, pero que se pueda mantener. Las estructuras de carpetas mal diseñadas o ad hoc se convertirán en una pesadilla de gestión y harán que el data lake sea inutilizable. A continuación, se indican algunos puntos que hay que tener en cuenta al diseñar la estructura de carpetas:

- **Legibilidad humana**-Las estructuras de carpetas legibles por el ser humano ayudarán a mejorar la exploración y la navegación de los datos.

- **Representación de la estructura organizativa**: Alinear la estructura de carpetas de acuerdo con la estructura organizativa ayuda a segregar los datos para su facturación y control de acceso. Esta estructura de carpetas ayudará a restringir el acceso a los datos entre equipos.

- **Distinguir los datos sensibles**-La estructura de carpetas debe ser tal que pueda separar los datos sensibles de los generales. Los datos sensibles requerirán niveles más altos de políticas de auditoría, privacidad y seguridad, por lo que mantenerlos separados facilita la aplicación de las políticas requeridas.

- **Capacidad de gestión de las ACLs**-Si recuerdas lo dicho anteriormente en el capítulo, las ACLs se utilizan para controlar qué usuarios tienen permisos de lectura, escritura o ejecución en archivos y carpetas. Deberíamos diseñar las carpetas de forma que necesitemos aplicar ACLs sólo en los niveles superiores de las carpetas y no en las carpetas hoja. No deberíamos llegar a una situación en la que tengamos que actualizar las ACL cada vez que se cree automáticamente una subcarpeta, como las nuevas carpetas de timestamp para las entradas de streaming.

- **Optimizar para que las lecturas sean más rápidas y estén distribuidas uniformemente**: si podemos distribuir los datos uniformemente, las cargas de trabajo podrán acceder a los datos en paralelo. Esto mejorará el rendimiento de las consultas. También hay que pensar en el soporte para la poda, que ya hemos comentado en una sección anterior.

- **Considere los límites de suscripción**: Azure tiene límites por suscripción sobre el tamaño de los datos, la entrada/salida de la red, el paralelismo, etc. Por lo tanto, si los datos van a ser enormes, debemos planificar su división a nivel de suscripción.

Teniendo en cuenta las directrices anteriores, vamos a explorar las estructuras de carpetas para tres casos de uso diferentes.

2.7.1. Escenarios de streaming e IoT

Los escenarios de streaming e IoT son complicados, ya que habrá miles de archivos que fluyen desde diferentes fuentes, como tweets, sensores, datos de telemetría, etc. Necesitamos una forma eficiente de rastrear y organizar los datos.

La plantilla de diseño recomendada por Microsoft es la siguiente

{Region}/{SubjectMatter(s)}/{yyyy}/{mm}/{dd}/{hh}/

Consideremos el ejemplo del IAC. Si queremos almacenar todos los viajes realizados por un taxi en particular, la estructura de la carpeta podría ser algo así

New York/cabs/cab1234/trips/2021/12/01

O, si queremos recoger los datos de los clientes de cada uno de los taxis cada día, podría ser algo así

New York/cabs/cab1234/customers/2021/12/01

Pero, ¿y si los datos de los clientes se consideran información sensible que debe cumplir con niveles más altos de seguridad y políticas de privacidad? En estos casos, la estructura de carpetas anterior se convierte en un inconveniente porque necesitamos iterar cada cabina y luego aplicar ACLs para cada una de las carpetas de clientes que se encuentran bajo ella. Por lo tanto, una estructura mejor sería la siguiente

New York/sensitive/customers/cab1234/2021/12/01

En la estructura de carpetas anterior, podemos aplicar fácilmente todas las políticas de seguridad y ACLs en la propia carpeta sensible. Las restricciones serán heredadas automáticamente por todas las subcarpetas que se encuentren bajo ella.

Por lo tanto, **si tiene datos categorizados como sensibles**, entonces podría considerar las siguientes plantillas:

{Region}/Sensitive/{SubjectMatter(s)}/{yyyy}/{mm}/{dd}/{hh}/

{Region}/General/{SubjectMatter(s)}/{yyyy}/{mm}/{dd}/{hh}/

Veamos a continuación los escenarios por lotes.

2.7.2. Escenarios por lotes

En los sistemas de procesamiento por lotes, leemos los datos de entrada y escribimos los datos procesados en los directorios de salida, por lo que es intuitivo mantener **directorios de entrada** y **salida** separados en las rutas de las carpetas. Además de los directorios de entrada y salida, también sería beneficioso tener un **directorio para todos los archivos defectuosos**, como los **archivos corruptos**, los **archivos incompletos**, **etc**., para que los administradores de datos puedan comprobarlos a intervalos regulares.

La plantilla de distribución recomendada por Microsoft se indica aquí:

{Region}/{SubjectMatter(s)}/In/{yyyy}/{mm}/{dd}/{hh}/

{Region}/{SubjectMatter(s)}/Out/{yyyy}/{mm}/{dd}/{hh}/

{Region}/{SubjectMatter(s)}/Bad/{yyyy}/{mm}/{dd}/{hh}/

Continuando con nuestro ejemplo del IAC, supongamos que queremos generar informes diarios para los viajes en taxi. Entonces, nuestro directorio de entrada podría tener el siguiente aspecto

New York/trips/In/cab1234/2021/12/01/*

Nuestro directorio de salida podría ser el siguiente

New York/trips/Out/reports/cab1234/2021/12/01

Y por último, el directorio de archivos malos podría tener el siguiente aspecto

New York/trips/Bad/cab1234/2021/12/01

También en este caso, **si tiene datos categorizados como sensibles**, podría considerar las siguientes plantillas:

{Region}/General/{SubjectMatter(s)}/In/{yyyy}/{mm}/{dd}/{hh}/

{Region}/Sensitive/{/{SubjectMatter(s)}/Out/{yyyy}/{mm}/{dd}/{hh}/

{Region}/General/{/{SubjectMatter(s)}/Bad/{yyyy}/{mm}/{dd}/{hh}/

CONSEJO

No ponga las carpetas de fechas al principio, ya que hace más tediosa la aplicación de ACL a cada subcarpeta. Además, hay límites en el número de ACLs que se pueden aplicar, por lo que hay posibilidades de que te quedes sin ACLs a medida que avanza el tiempo.

Ahora que hemos aprendido algunas de las mejores prácticas para la creación de estructuras de carpetas en un data lake, avancemos para explorar algunas estrategias de optimización que involucran a Azure Synapse Analytics, como lo requiere nuestro programa de certificación.

2.8. Diseñar una estrategia de distribución

Las estrategias de distribución son técnicas que se utilizan en Synapse Dedicated SQL Pools. Los Dedicated SQL Pools de Synapse son sistemas de procesamiento paralelo masivo (MPP) que dividen las consultas en 60 consultas paralelas y las ejecutan en paralelo. Cada una de estas consultas más pequeñas se ejecuta en algo llamado distribución. Una distribución es una unidad básica de procesamiento y almacenamiento para un dedicated SQL pool.

Dedicated SQL utiliza Azure Storage para almacenar los datos, y proporciona tres formas diferentes de distribuir (shard) los datos entre las distribuciones. Se enumeran a continuación:

- **Tablas Round-robin**
- **Tablas Hash**
- **Tablas replicadas**

En función de nuestros requisitos, tenemos que decidir cuál de estas técnicas de distribución debe utilizarse para crear nuestras tablas. Para elegir la estrategia de distribución correcta, debes entender tu aplicación, la disposición de los datos y los patrones de acceso a los datos mediante el uso de planes de consulta. Aprenderemos a generar y leer planes de consulta y patrones de datos en capítulos posteriores. En este momento, vamos a tratar de entender la diferencia de alto nivel en cada una de las técnicas de distribución y cómo elegir la opción correcta.

2.8.1. Tablas round-robin

En una tabla round-robin, los datos se distribuyen en serie entre todas las distribuciones. Es la más sencilla de las distribuciones y es la opción por defecto cuando no se especifica el tipo de distribución. Esta opción es la más rápida para cargar los datos, pero **no es la mejor para las consultas que incluyen joins**. Utilice las tablas round-robin para los datos de preparación (staging data) o los datos temporales, donde los datos van a ser leídos en su mayoría.

Este es un ejemplo sencillo de creación de una tabla distribuida round-robin en Dedicated SQL Pool:

```
CREATE TABLE dbo.CabTable
(
   [cabId] INT NOT NULL,
   [driverName] VARCHAR(20),
   [driverLicense] VARCHAR(20)
)
```

No es necesario especificar ningún atributo para las tablas round-robin; es la distribución por defecto.

2.8.2. Tablas hash

En una tabla hash, las filas se distribuyen a diferentes distribuciones basadas en una función hash. La clave hash suele ser una de las columnas de la tabla. Las tablas hash son las mejores para las consultas con joins y agregaciones. Son ideales para las tablas grandes.

A continuación, se muestra un ejemplo sencillo de creación de una tabla distribuida por hash en Dedicated SQL Pool:

```
CREATE TABLE dbo.CabTable
(
   [cabId] INT NOT NULL,
   [driverName] VARCHAR(20),
   [driverLicense] VARCHAR(20)
)
WITH
(
   DISTRIBUTION = HASH (cabId)
)
```

CONSEJO

Elija una clave de columna que sea distinta y estática, ya que esto puede equilibrar la distribución de datos entre las particiones.

2.8.3. Tablas replicadas

Con las tablas replicadas, los datos de la tabla se copian en todas las distribuciones. Son ideales para tablas pequeñas en las que el coste de copiar los datos para los joins de consulta supera los costes de almacenamiento de estas tablas pequeñas. Utilice las tablas replicadas para almacenar tablas de consulta rápida (LUT).

A continuación se muestra un ejemplo sencillo de creación de una tabla replicada en Dedicated SQL Pool:

```
CREATE TABLE dbo.CabTable
(
   [cabId] INT NOT NULL,
   [driverName] VARCHAR(20),
   [driverLicense] VARCHAR(20)
)
WITH
(
   DISTRIBUTION = REPLICATE
)
```

Volveremos a revisar estos conceptos en detalle cuando aprendamos a implementarlos para problemas prácticos en los siguientes capítulos. Pasemos ahora al último tema de nuestro capítulo, que trata apropiadamente del final de la vida de nuestros datos.

2.9. Diseñar una solución de archivo de datos

Ahora que hemos aprendido a diseñar un data lake y a optimizar el almacenamiento para nuestras consultas analíticas, queda un último componente por diseñar. ¿Cómo archivamos o limpiamos los datos antiguos? Sin una solución adecuada de archivado y/o eliminación, los datos crecerán y llenarán el almacenamiento muy pronto.

Azure proporciona tres niveles de almacenamiento: Hot Access Tier, Cold Access Tier y Archive Access Tier.

2.9.1. Hot Access Tier

La capa Hot Access es ideal para los datos a los que se accede con **frecuencia**. Proporciona el menor coste de acceso pero con un mayor coste de almacenamiento.

2.9.2. Cold Access Tier

La capa Cold Access es ideal para los datos a los que se accede **ocasionalmente**, como los datos un poco más antiguos que probablemente se utilizan para las copias de seguridad o los informes mensuales. Proporciona costes de almacenamiento más bajos pero con costes de acceso más altos. Azure espera que los datos de la capa de acceso en frío se almacenen durante al menos 30 días. El borrado anticipado o el cambio de nivel podría dar lugar a cargos adicionales.

2.9.3. Archive Access Tier

La capa Archive Acces es ideal para almacenar datos **durante largos periodos de tiempo**. Puede ser por razones de cumplimiento, copias de seguridad a largo plazo, archivo de datos, etc. El nivel de acceso de archivo es una solución de almacenamiento offline, lo que significa que no podrá acceder a los datos a menos que los rehaga desde el nivel de archivo a un nivel online. Esta es la opción de almacenamiento más barata entre todos los niveles. Azure espera que los datos de la capa de archivo se almacenen durante al menos 180 días. La eliminación anticipada o el cambio de nivel pueden dar lugar a cargos adicionales.

2.9.4. Gestión del ciclo de vida de los datos

Azure Blob storage proporciona herramientas para la gestión del ciclo de vida de los datos. Mediante estas herramientas, **podemos definir políticas como el tiempo que un dato concreto debe estar en la capa Hot Access, cuándo mover los datos entre las distintas capas de acceso, cuándo eliminar los blobs, etc**. Azure ejecuta estas políticas diariamente.

Veamos un ejemplo de cómo crear una política de ciclo de vida de datos.

Azure portal

Exploremos cómo crear una política de ciclo de vida de datos utilizando el portal de Azure, como sigue:

1. En el portal de Azure, seleccione su cuenta de almacenamiento.
2. Vaya a la pestaña **Gestión de datos** y seleccione **Gestión del ciclo de vida**.
3. Haz clic en el signo + para añadir una nueva regla.
4. En la página **Detalles**, añada un nombre para su regla y seleccione los blobs de almacenamiento que debe aplicar esta regla.
5. Haga clic en **Siguiente**, y en la página **Blobs base**, puede especificar la regla real. En la Figura 2.11 se muestra un ejemplo.
6. Una vez hecho esto, haga clic en el botón **Añadir**.

Puede ver un resumen de esto en la siguiente captura de pantalla:

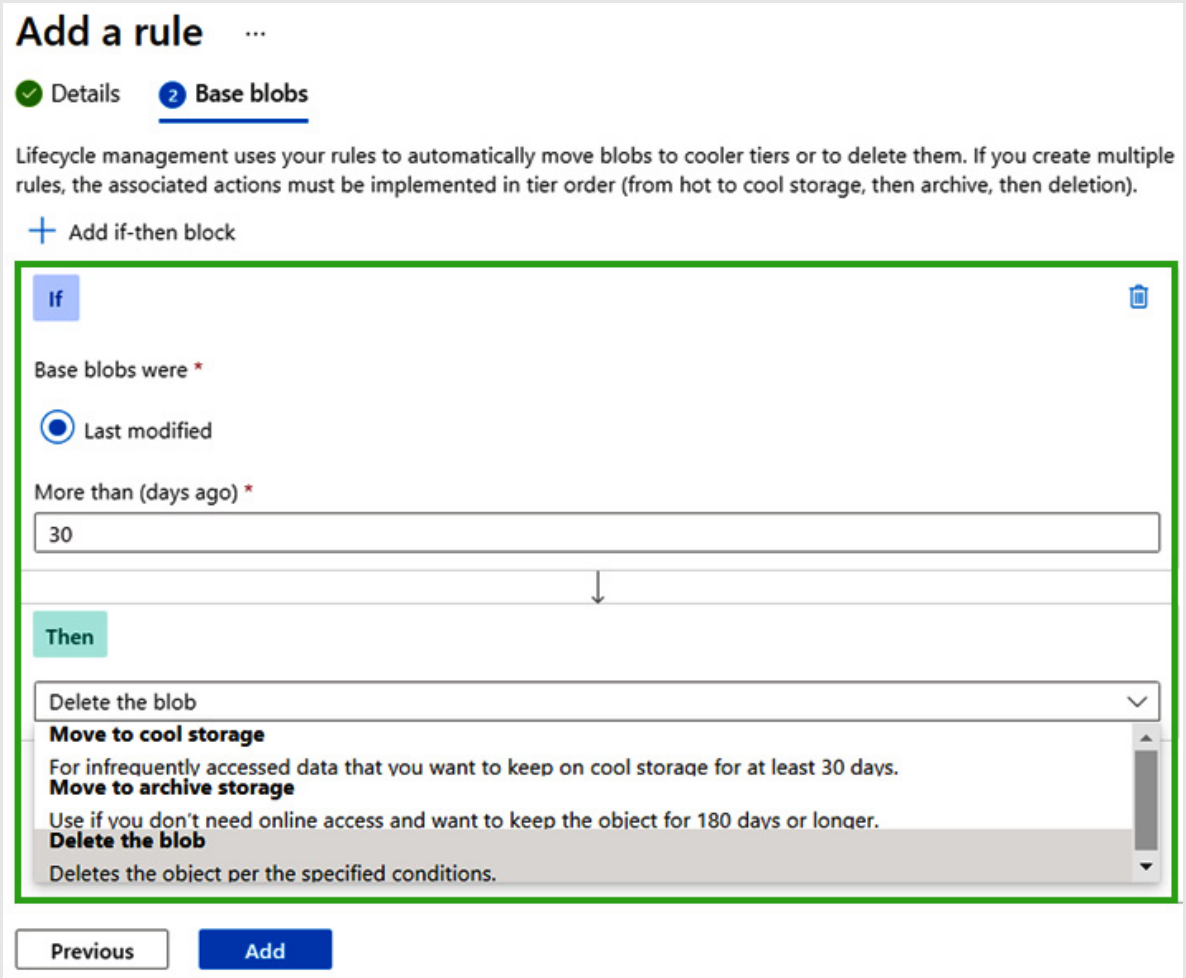


Figura 2.11 - Cómo especificar las reglas de gestión del ciclo de vida de los datos en Azure Storage

NOTA

Azure ejecuta las políticas de ciclo de vida de los datos sólo una vez al día, por lo que sus políticas podrían tardar hasta 24 horas en entrar en vigor.

Resumen

Con esto, hemos llegado al final de nuestro segundo capítulo. Hemos explorado los diferentes diseños de data lake en detalle y hemos aprendido buenas prácticas para diseñar uno. Ahora debería sentirse cómodo respondiendo a las preguntas relacionadas con las arquitecturas de los data lake y con las tecnologías de almacenamiento, cómputo y otras involucradas en la creación de un data lake. También debería estar familiarizado con los formatos de archivo más comunes, como Avro, Parquet y ORC, y saber cuándo elegir qué formatos de archivo. También exploramos diferentes técnicas de optimización como la poda de datos, el particionamiento, el almacenamiento en caché, la indexación, y más. También aprendimos sobre estructuras de carpetas, distribución de datos y, por último, el diseño de un ciclo de vida de los datos mediante el uso de políticas para archivar o eliminar datos. Esto cubre el temario del examen DP-203, capítulo Diseño de una estructura de almacenamiento de datos. Reforzaremos lo aprendido en este capítulo mediante detalles de implementación y consejos en los siguientes capítulos.

Exploremos el concepto de partición con más detalle en el siguiente capítulo.