DATAFLOWS GEN2 (CONCEPTOS Y USO)

1. ¿Qué es un Dataflow Gen2 en Fabric y en qué se diferencia de Gen1?
Un Dataflow Gen2 es un artefacto de Microsoft Fabric basado en Power Query Online que permite extraer, transformar y cargar (ETL) datos hacia destinos como Lakehouse, Warehouse o datasets de Power BI. A diferencia de Gen1 (más ligado a Power BI y normalmente a un único destino), Gen2:
- Soporta múltiples destinos (por ejemplo, escritura simultánea a un Lakehouse y a un Warehouse).
- Se integra de forma nativa con Data Pipelines (actividades de Dataflow Gen2 dentro de pipelines de Data Factory en Fabric).
- Está optimizado para escenarios de alto rendimiento sobre OneLake, reutilizando la infraestructura de Fabric (escalado, capacidad, ejecución distribuida).
- Ofrece mejoras de autosave, control de versión y alineación con el resto de artefactos de Fabric (lakehouse, warehouse, notebooks, etc.).

2. Rol de Power Query Online dentro de un Dataflow Gen2 y transformaciones típicas
Power Query Online es el motor de transformación dentro del Dataflow Gen2. Define la lógica de ETL usando pasos declarativos (lenguaje M) encadenados:
- Origen: conexiones a bases de datos relacionales, archivos en OneLake, servicios SaaS, APIs, etc.
- Transformación: filtrado de filas, selección de columnas, joins/merge entre tablas, agregaciones, ordenamientos, eliminación de duplicados, cambio de tipos de datos, pivot/unpivot, columnas calculadas, reemplazo de valores, limpieza de nulos, etc.
- Carga: escritura del resultado hacia destinos soportados (Lakehouse, Warehouse, Dataset).
La idea es que el analista o ingeniero pueda construir la lógica de negocio sin escribir código Spark o SQL complejo, basándose en la experiencia de Power Query que ya existe en Power BI y Excel

3. Importancia de aplicar transformaciones “upstream” en el Dataflow
Aplicar las transformaciones en el Dataflow, antes de que los datos lleguen al modelo de Power BI, es clave por:
- Rendimiento: Power BI recibe datos ya limpios, agregados y modelados, por lo que el procesamiento durante el refresh del dataset baja drásticamente y se acortan los tiempos de actualización.
- Reutilización: Ese mismo Dataflow Gen2 puede alimentar varios modelos de Power BI, Lakehouses o Warehouses. Evitas duplicar la misma lógica de negocio en cada informe.
- Gobernanza y calidad: Centralizas las reglas de negocio (definición de métricas, filtros, mapping de dominios) en un único artefacto, lo que reduce inconsistencias entre informes.
- Mantenimiento: cuando cambian requisitos, ajustas el Dataflow y todos los consumidores se benefician del cambio sin tocar decenas de Power BI reports.

4. Impacto de conectarse múltiples veces a la misma fuente
Cada consulta independiente contra una fuente (por ejemplo, varias queries que leen la misma tabla con ligeras variaciones) implica:
- Más viajes al origen (latencia de red y carga sobre el sistema transaccional).
- Más tiempo de ejecución del Dataflow.
- Mayor riesgo de inconsistencias si cada query filtra o transforma de forma distinta.
Buenas prácticas:
- Reutilizar consultas base mediante “Reference” en Power Query en lugar de duplicarlas.
- Parametrizar conexiones (por ejemplo, pasar fechas, clientes o regiones como parámetros) en lugar de clonar la misma consulta.
- Reducir al mínimo el número de toques al origen, trayendo el subconjunto necesario de datos y reutilizando internamente los resultados.

5. Destinos típicos de un Dataflow Gen2 y su impacto en reporting y analítica
Los principales destinos son:
- Lakehouse (tablas Delta en OneLake): ideal para escenarios de big data, análisis con Spark, y como “stage limpio” para otros procesos (machine learning, batch, etc.).
- Warehouse: un data warehouse relacional optimizado para consultas SQL y BI corporativo; encaja con

modelos estrella, reporting financiero, etc.
- Dataset de Power BI: para escenarios de self■service BI donde el modelo tabular es consumido directamente por reportes y dashboards.
El destino define el patrón de uso:
- Lakehouse → análisis avanzado y escenarios multi■motor.
- Warehouse → reporting corporativo, modelos altamente estructurados.
- Dataset → visualización rápida y autoservicio sobre datos ya curados.

6. Ventajas de exponer solo Dataflows a los analistas
Exponer Dataflows (entidades lógicas) en lugar de fuentes brutas aporta:
- Gobernanza: controlas qué tablas y transformaciones están “aprobadas” para uso analítico.
- Seguridad: puedes ocultar detalles de la infraestructura (nombres de servers, esquemas sensibles) y aplicar reglas de acceso a nivel de Dataflow.
- Simplicidad: los usuarios analíticos ven un catálogo de entidades de negocio (Clientes, Ventas, Productos) en vez de cientos de tablas técnicas.
- Consistencia: todos usan la misma semántica de campos (códigos, dim/measure, formatos) reduciendo discrepancias de resultados entre informes.

7. Limitaciones y prerrequisitos de Dataflows Gen2
Algunos puntos a considerar:
- Capacidad de Fabric: se ejecutan sobre workspaces con capacidad, por lo que hay un costo asociado; no es simplemente “gratis” como algunos escenarios pequeños en Power BI.
- Compatibilidad: no todas las transformaciones de Power Query Desktop están soportadas de forma idéntica en el entorno online; conviene revisar documentación y pruebas cuando migras queries complejas.
- Límites de tamaño y tiempo de ejecución: existen límites de tiempo, memoria y tamaño de datos por ejecución según el SKU de capacidad.
Esto implica que, en entornos enterprise, hay que planificar licenciamiento, dimensionamiento y gobierno de forma consciente.

8. Importancia del orden de los Applied Steps en Power Query
En Power Query, cada “Applied Step” toma la salida del paso anterior como entrada:
- Si cambias el orden, cambias la lógica. Por ejemplo, filtrar antes de un join vs. después puede alterar el resultado y el rendimiento.
- Algunos pasos pueden fallar si se reordenan (p.ej. un cambio de tipo que se aplica antes de que exista la columna).
Para depurar:
- Revisa la lista de pasos de arriba hacia abajo, verificando el resultado intermedio en la vista de datos.
- Desactiva temporalmente pasos (cuando sea posible) para identificar dónde aparece un error.
- Simplifica la lógica, renombrando pasos de forma descriptiva para entender mejor su función.

9. Integración de un Dataflow Gen2 dentro de un Data Pipeline
En un Data Pipeline de Fabric (Data Factory):
- Añades una actividad de tipo “Dataflow Gen2”.
- Seleccionas el Dataflow concreto del workspace.
- Puedes pasar parámetros al Dataflow (fechas, rutas, flags).
- Defines dependencias con otras actividades (p.ej. ejecutar el Dataflow después de copiar datos desde una API).
- Programas el pipeline por horario, evento o ejecución manual.
Esto permite que el Dataflow sea un eslabón dentro de un flujo E2E: ingestión → transformación (Dataflow) → carga/post■procesado.

10. Indicadores y vistas de monitoreo de Dataflows Gen2
Para comprobar refrescos:

- Historial de ejecuciones: fecha/hora, duración, estado (correcto, con error, cancelado).
- Detalles de error: mensajes específicos que indican problemas de credenciales, conexiones, esquemas incompatibles, etc.
- Métricas de capacidad: uso de CPU, memoria y throughput, que ayudan a identificar cuellos de botella.
- Alertas: notificaciones configuradas (por correo, Teams, etc.) cuando una actualización falla, permitiendo reaccionar rápido.

DATA PIPELINES EN MICROSOFT FABRIC

1. Definición de Data Pipeline y tipos de actividades
Un Data Pipeline es una secuencia orquestada de actividades de ingestión, transformación y carga de datos, basada en Data Factory dentro de Fabric. Puede contener:
- Copy activity (copiar datos entre orígenes y destinos).
- Actividades de Dataflow Gen2.
- Notebooks Spark.
- Scripts SQL/Warehouse, stored procedures.
- Actividades de control (If, Switch, ForEach, Wait, Web, HTTP, etc.).
Su objetivo es coordinar y automatizar el movimiento y procesamiento de datos de extremo a extremo.

2. Diferencia conceptual entre Dataflow Gen2 y Data Pipeline
- Dataflow Gen2: herramienta de ETL basada en Power Query forzada a la transformación y la modelización lógica de datos.
- Data Pipeline: herramienta de orquestación que decide qué se ejecuta, en qué orden, con qué parámetros y cuándo.
Normalmente, un pipeline llama a varios Dataflows, notebooks y otras actividades para construir un proceso completo.

3. Parámetros y variables en un pipeline
Permiten que el pipeline sea dinámico:
- Parámetros: se definen a nivel de pipeline y se proporcionan en cada ejecución (p.ej. fecha de proceso, carpeta destino, id de cliente).
- Variables: se usan dentro del pipeline para almacenar y manipular valores durante la ejecución (contadores, flags, resultados intermedios).
Con ellos, un solo pipeline puede manejar escenarios como “crear carpeta de salida con nombre basado en la fecha actual” sin multiplicar definiciones.

4. Consulta del historial de ejecuciones
En el panel de monitorización:
- Ves cada ejecución del pipeline, su estado y duración.
- Puedes inspeccionar cada actividad individual (tiempo, reintentos, errores).
- Puedes filtrar por fecha, estado o versión del pipeline.
Esto es esencial para debugging, optimización de rendimiento y evidencias de auditoría.

5. Buenas prácticas de diseño de pipelines reutilizables
- Separar ingestión (traer datos brutos) y transformación (limpieza/modelado).
- Usar parámetros para no duplicar pipelines por cada origen/destino.
- Modularizar: pipelines maestros que invocan sub■pipelines reutilizables.
- Integrar con CI/CD (por ejemplo, usar repos y despliegues automatizados).
- Manejar fallos explícitamente: reintentos, rutas alternativas (branching), alertas.

SPARK EN MICROSOFT FABRIC (POOLS, DATAFRAMES, DELTA)

1. ¿Qué es un Spark pool y parámetros clave?
Un Spark pool es un clúster de Apache Spark gestionado por Fabric:

- Tipo/tamaño de nodo: determina memoria y núcleos por nodo.
- Número mínimo y máximo de nodos: define el rango de escalado.
- Configuración de autoscale/dynamic allocation: cómo y cuándo se añaden o quitan nodos.
Estos parámetros impactan directamente en coste, rendimiento y capacidad de concurrencia.

2. Funcionamiento del autoscale en un Spark pool
Autoscale:
- Aumenta nodos cuando hay más trabajos o grandes volúmenes de datos, hasta el máximo configurado
- Reduce nodos cuando la carga baja, hasta el mínimo.
Es ideal cuando la carga es variable o por picos, ya que evita pagar por capacidad ociosa. Un número fijo de nodos es más apropiado cuando la carga es estable y predecible.

3. Entorno personalizado y resource files
Un entorno personalizado define el “runtime” del cluster:
- Resource files: librerías Python (wheel, egg), JARs, ficheros de configuración (por ejemplo, archivos de propiedades, certificados, etc.).
- Estos se cargan automáticamente cuando notebooks/jobs arrancan en ese entorno.
Ventaja: todos los desarrolladores y jobs usan la misma versión de dependencias, garantizando consistencia y evitando instalaciones manuales en cada sesión.

4. Conceptos básicos del DataFrame API
La API de DataFrame de Spark expone operaciones de alto nivel sobre datos tabulares distribuidos:
- select: escoger columnas o expresiones; útil para proyectar solo lo necesario.
- filter/where: filtrar filas; se usa para reducir volumen y aplicar condiciones de negocio.
- groupBy: agrupar datos para agregaciones (sum, avg, count, max, min).
- withColumn: crear o modificar columnas (derivadas, conversiones, cálculos).
- join: combinar DataFrames por claves; fundamental para modelar relaciones entre tablas.
Se eligen según el patrón: primero filtrar, luego seleccionar columnas relevantes, con groupBy para métricas y joins para enriquecimiento.

5. Eficiencia de seleccionar solo columnas necesarias
Seleccionar solo las columnas necesarias:
- Reduce I/O y memoria.
- Permite que el planificador de Spark optimice los scans y proyecciones.
- Evita arrastrar columnas pesadas (p.ej. blobs, textos largos) que no se usan.
Esto se traduce en jobs más ligeros, económicos y rápidos.

6. Delta Lake / Delta tables en Fabric
Delta Lake:
- Añade transacciones ACID, versionado (time travel), manejo de esquema y operaciones como MERGE
datos almacenados sobre formatos tipo Parquet.
En Fabric:
- df.write.format("delta").saveAsTable("nombre_tabla") crea una tabla Delta registrada en el catálogo.
- Esa tabla puede consultarse via Spark SQL y a través del SQL endpoint del Lakehouse/Warehouse.
Permite unificar workloads batch, streaming y BI sobre un mismo almacenamiento fiable.

7. Diferencia entre vista temporal y tabla Delta persistente
- createOrReplaceTempView("mi_vista"): vista lógica en memoria, sólo disponible en la sesión Spark actual. No persiste.
- saveAsTable("mi_tabla"): escribe datos en almacenamiento como tabla Delta persistente. Es visible para otros notebooks, sesiones y motores, y sobrevive reinicios.
La vista temporal se usa para análisis ad■hoc; la tabla persistente para datasets duraderos y compartidos.

8. Transformaciones con withColumn y funciones de Spark
withColumn:
- Para casting: df.withColumn("Fecha", to_date("FechaStr", "yyyy-MM-dd")).
- Para derivar campos: concatenar columnas, crear flags condicionales (when/otherwise), realizar cálculos numéricos.
Es la forma estándar de extender o modificar el esquema sin sobrescribir completamente el DataFrame.

9. Prácticas para manejar volúmenes variables
- Habilitar autoscale y dynamic allocation.
- Particionar datos por columnas de alta cardinalidad natural (fecha, región, etc.) para mejorar filtros.
- Evitar cachear DataFrames muy grandes sin necesidad; cache sólo lo que se reutiliza y cabe razonablemente en memoria.
- Optimizar joins (broadcast hash join para dimension tables pequeñas, evitar skew).
- Usar formatos columnar (Parquet/Delta) y evitar formatos fila■a■fila como CSV para procesamiento intensivo.

KQL, EVENTHOUSE Y MATERIALIZED VIEWS

1. ¿Qué es KQL y escenarios de uso en Fabric/Eventhouse?
Kusto Query Language (KQL) es un lenguaje declarativo optimizado para análisis interactivo de grandes volúmenes de datos de logs, telemetría y eventos. En Fabric/Eventhouse se usa para:
- Monitorizar aplicaciones, infraestructura y seguridad.
- Analítica near real■time sobre flujos de eventos.
- Construir dashboards operativos, alertas y detección de anomalías.

2. Funciones almacenadas (stored functions) en KQL
Las stored functions:
- Encapsulan lógica compleja (joins, filtros, proyecciones, agregaciones).
- Pueden aceptAR parámetros.
- Se reutilizan desde múltiples consultas, garantizando que todos apliquen la misma lógica de negocio.
- Reducen duplicación y facilitan el mantenimiento: cambias la función una vez y todas las consultas que la usan se actualizan.

3. Uso y limitaciones de take
take N:
- Devuelve N filas, típicamente no ordenadas.
- Excelente para muestrear datos, entender el esquema o revisar valores típicos sin escanear toda la tabla.
Limitaciones:
- No garantiza representatividad estadística.
- Sin un orden previo (por ejemplo, sort by Timestamp desc) el subset puede ser arbitrario.

4. Database shortcuts en KQL/Eventhouse
Database shortcuts:
- Permiten exponer bases externas (por ejemplo, Azure Data Explorer) dentro de tu base KQL como si fueran locales.
- Evitan duplicar datos: no se copian físicamente, sólo se referencia el origen.
- Ahorran almacenamiento y esfuerzo de ingestión, manteniendo una única fuente de verdad, pero habilitando análisis desde múltiples entornos.

5. Filtro por tiempo al inicio de la consulta
La mayoría de tablas de eventos están indexadas por tiempo:
- Aplicar where Timestamp > ago(1d) o un rango fijo al principio reduce drásticamente el volumen a

escanear.
- Mejora tiempos de respuesta y reduce consumo de recursos.
- Es una buena práctica de diseño de todas las consultas orientadas a eventos.

6. Ingesta continua desde fuentes no■Microsoft
Para fuentes como Amazon Kinesis:
- Se utilizan conectores de streaming que leen de la fuente, hacen mapping de campos y los escriben en tablas KQL.
- Permiten ingestión continua con baja latencia.
- Reducen la necesidad de jobs batch manuales para cargar archivos.

7. format_datetime y otros operadores de fecha/hora
format_datetime(datetime_col, "yyyy-MM-dd"):
- Devuelve un string con el datetime formateado.
Otros operadores:
- ago(N): rangos relativos (ago(1h), ago(7d)).
- bin(datetime_col, 1h): agrupar en intervalos de tiempo (1 hora, 1 minuto, etc.).
- datetime_add, datetime_diff: sumar/restar y calcular diferencias de tiempo.
- startofday, startofweek, startofmonth: normalizar a límites de períodos.

8. Materialized views en KQL
Materialized views:
- Ejecutan de forma continua una consulta definida (normalmente agregaciones sobre una tabla grande).
- Guardan resultados pre■calculados, sólo se actualizan con nuevos datos.
- Las consultas contra la vista leen menos datos y hacen menos trabajo, por lo que responden mucho más rápido.
Son ideales para KPIs, dashboards recurrentes y queries estándar sobre datasets masivos.

9. Impacto y trade■offs de materialized views
Impacto positivo:
- Tiempos de respuesta mucho menores para consultas repetitivas.
- Menor uso de CPU y memoria para usuarios finales.
Trade■offs:
- Coste de almacenamiento adicional para los resultados precomputados.
- Coste de mantenimiento: hay que diseñar y monitorizar la vista, controlar su frescura y su ventana de retención.
- Menos flexibilidad para consultas ad■hoc que cambian mucho; ahí sigue siendo mejor consultar directamente la tabla base.