# CONCEPTOS FUNDAMENTALES DE MICROSOFT FABRIC

## Cuestionario Detallado para DP-700

### FUNDAMENTOS DE FABRIC, ONELAKE Y DELTA

**1. ¿Qué significa que Microsoft Fabric sea un entorno unificado de datos?**

Microsoft Fabric es un SaaS integrado que reúne experiencias de datos: Data Engineering, Data Factory, Data Warehouse, Real-time Analytics, Data Science y Business Intelligence bajo un único entorno. Comparten almacenamiento (OneLake), capacidad de cómputo (Spark, SQL) y seguridad unificada. Permite que ingenieros, analistas y científicos de datos colaboren sin duplicación de datos, con versionado centralizado.

**2. ¿Qué es OneLake y por qué se le llama 'OneDrive para datos'?**

OneLake es el almacenamiento unificado de Fabric: espacio lógico único donde viven lakehouses, warehouses y dataflows sin duplicación física. Se le llama 'OneDrive para datos' porque proporciona un espacio coherente, centralizado y accesible desde múltiples herramientas (Spark, SQL, Power BI, Python).

**3. ¿Qué es el formato Delta-Parquet?**

**Parquet:** Formato columnar comprimido de alto rendimiento. Reduce I/O y memoria, mejora queries sobre subconjuntos de columnas.
**Delta:** Capa transaccional sobre Parquet que añade: (a) ACID (consistencia garantizada), (b) Schema enforcement/evolution (control de estructura), (c) Time travel (histórico de cambios), (d) Delta log (auditoría). Delta-Parquet = lo mejor de ambos.

**4. ¿Por qué Fabric usa Delta tables como almacenamiento base?**

Delta tables ofrecen fiabilidad (ACID), permitiendo múltiples escritores simultáneamente sin corrupción. El versionado permite auditoría y rollback. Parquet columnar es óptimo para queries SQL. Schema enforcement mantiene calidad de datos. Esto permite que el mismo artefacto sea consumido por SQL, Spark, Power BI sin inconsistencias.

**5. ¿Qué es un lakehouse en Fabric y cómo difiere de un data lake 'crudo'?**

Un data lake crudo es solo almacenamiento sin estructura. Un **lakehouse** añade: (1) metáfora relacional (tablas, esquema, catálogo), (2) motores de consulta (SQL endpoint, Spark SQL), (3) integración con Power BI para BI directo, (4) gobernanza (permisos, auditoría). El lakehouse combina flexibilidad del lake con potencia del warehouse.

### LAKEHOUSE, DELTA Y ARQUITECTURA DE DATOS

**6. ¿Qué es schema-on-read?**

Schema-on-read interpreta la estructura de datos **en la lectura**, no en la escritura. Permite ingerir datos variados (JSON, XML, anidados) sin transformar primero. Facilita data discovery: traes datos brutos y luego decides cómo interpretarlos. Contrasta con data warehouse clásico (schema-on-write): impone esquema fijo. Schema-on-read = más flexibilidad; schema-on-write = más control.

**7. ¿Qué garantizan las tablas Delta Lake?**

**ACID:** Atomicity, Consistency, Isolation, Durability. Evita reportes con cifras inconsistentes.
**Manejo de esquema:** Schema evolution permite añadir columnas. Schema enforcement rechaza datos inválidos.
**Time travel:** Consultar datos en estado anterior para reproducibilidad, auditoría y rollback. Versionado automático en Delta log.

**8. ¿Diferencia entre tablas gestionadas y externas al hacer DROP?**

**Tabla gestionada:** Al DROP, se elimina metadatos **y archivos de datos**.
**Tabla externa:** Al DROP, se elimina solo metadatos; archivos permanecen intactos en ubicación externa. Útil para compartir datos entre equipos.

**9. ¿Pasos ETL/ELT hacia un lakehouse?**

(1) **Ingestión:** Traer datos brutos de orígenes. (2) **Bronze:** Almacenar raw en Delta tables, cambios mínimos. (3) **Silver:** Aplicar reglas de negocio, joins, limpiar. (4) **Gold:** Modelos finales para reportes. Esta progresión es Medallion Architecture.

**10. ¿Qué es Medallion Architecture y separar en capas/workspaces?**

**Bronze:** Datos brutos. **Silver:** Datos limpios. **Gold:** Modelos finales.
Separar en workspaces aporta: (a) **Seguridad:** control de acceso por nivel, (b) **Capacidad:** dimensionar recursos, (c) **Costes:** facturación separada, (d) **Responsabilidades:** equipos especializados.

**11. ¿Cómo combinar datos estructurados + redes sociales en un lakehouse?**

El lakehouse ingiere ambos: datos transaccionales (SQL, Parquet) en tablas schema fijo, y feeds de redes sociales (JSON, variado) como Delta tables usando schema-on-read. Luego, Spark SQL permite JOINear ambas: 'SELECT t.customer_id, t.sales_amount, s.sentiment FROM transactions t JOIN social_feed s ON t.customer_id = s.user_id'. Flexibilidad del lake + potencia del warehouse.

## DATA FACTORY, DATAFLOWS GEN2 Y PIPELINES

### 12. ¿Qué es un Dataflow Gen2 y el rol de Power Query Online?

Un Dataflow Gen2 encapsula lógica ETL usando Power Query Online. Ofrece >300 transformaciones visuales: filtrar, joins, quitar duplicados, cambiar tipos, pivotar, etc. Sin código. Soporta múltiples destinos (Lakehouse, Warehouse, Dataset Power BI). La reutilización es clave: un Dataflow bien diseñado alimenta múltiples reportes, evitando duplicación.

### 13. Diferencia Dataflow Gen2 vs Data Pipeline

**Dataflow Gen2:** Herramienta de transformación (qué transformar). Bajo-código, reutilizable.
**Data Pipeline:** Herramienta de orquestación (qué, cuándo, en qué orden). Contiene múltiples actividades (Dataflows, notebooks, SQL scripts) encadenadas por dependencias y triggers. Caso típico: Pipeline que ejecuta Copy → Dataflow → Spark notebook → SQL.

### 14. Transformaciones típicas en Power Query

Filtrado, Joins/Merge, Quitar duplicados, Cambio de tipos, Limpieza de nulos, Agregaciones, Pivotar/Unpivot, Columnas calculadas. Todas garantizan datos limpios y consistentes antes de cargar al modelo.

### 15. ¿Por qué el orden de Applied Steps importa?

Cada paso ejecuta sobre el resultado del anterior. El orden afecta lógica y rendimiento: filtrar antes de join reduce volumen, mejora velocidad. Un join antes de filtrar vs después puede cambiar el resultado. Buena práctica: filtrar → select → transformar → agregar.

### 16. Integración Dataflows Gen2 en Pipelines

En Data Factory, añades actividad 'Dataflow Gen2', seleccionas el Dataflow, pasas parámetros, configuras dependencias y trigger. Result: Pipeline automatizada que copia → transforma vía Dataflow → carga resultado.

### 17. Ventajas de exponer solo Dataflows a analistas

Abstracción (ocultas complejidad), Gobernanza (controlas qué está aprobado), Consistencia (una semántica compartida), Seguridad (ocultas detalles sensibles), Catálogo simple (vs decenas de tablas técnicas).

### 18. ¿Por qué Dataflows Gen2 no sustituyen un data warehouse?

Un data warehouse es base **relacional optimizada** con modelo dimensional, histórico, índices avanzados, transacciones complejas. Dataflows son transformación puntual (lake to warehouse). No resuelven almacenamiento optimizado, modelado relacional complejo. Mejor arquitectura: Dataflows → cargan a Warehouse.

### 19. Parámetros y variables en pipelines

**Parámetro:** Valor definido al nivel del pipeline, pasado en ejecución. Se referencia como '@pipeline().parameters.FechaInicio'.
**Variable:** Valor temporal dentro ejecución, actualizable. Se referencia como '@variables('RowsProcessed')'. Caso: parámetro 'FolderName' permite copiar a 'Clientes/2024-01-15' o 'Clientes/2024-01-16' sin duplicar pipeline.

### 20. Vistas de monitorización Data Factory/Fabric

Run history (todas ejecuciones, estado, duración), Estado por actividad (qué falló, cuánto tardó), Errores detallados (mensajes específicos), Reintentos (cuántos intentos), Alertas/Notificaciones (email/Teams). Crítico para SLA y debugging rápido.

## SPARK, POOLS, DATAFRAMES Y ENTORNOS

### 21. ¿Qué es un Spark pool en Fabric?

Clúster Apache Spark gestionado. Parámetros: Tipo/tamaño de nodo (CPU, memoria por nodo), Min/max nodes (rango escalado), Autoscale (¿escalar automáticamente?), Dynamic allocation (Spark ajusta workers en tiempo real). Facturación: por cores·minutos consumidos.

**22. Autoscale vs número fijo de nodos**

**Autoscale:** Nodos se añaden/quitan según demanda. Ideal cargas variables, costo dinámico.
**Número fijo:** Cluster siempre tiene N nodos. Ideal cargas predecibles, pero pagas incluso ociosa. Recomendación: Autoscale para dev/test o picos; fijo para 24/7 predecible.

**23. Custom environment (librerías y JARs)**

Define runtime Spark: Python packages (numpy, pandas, scikit-learn, etc.), JARs (drivers JDBC, librerías custom), Archivos configuración. Se pre-cargan al arrancar, disponible para todos. Evita instalar manualmente; asegura consistencia.

**24. Native execution engine en Spark**

Optimización que usa **vectorización** (procesa columnas completas) vs row-by-row. Acelera queries, reduce CPU. Se activa configurando propiedades Spark en environment settings. Promete mejoras 2-10x en queries selectas.

**25. Operaciones básicas DataFrame API**

**select:** Proyectar columnas (reduce I/O, memoria).
**filter/where:** Filtrar filas (aplicar temprano).
**groupBy:** Agrupar para agregaciones.
**agg:** Agregación (sum, avg, count, max).
**withColumn:** Crear/modificar columna.
**join:** Combinar DataFrames. Patrón: filter → select → groupBy/agg → join.

**26. Por qué select solo columnas necesarias es eficiente**

I/O: menos datos leídos. Memoria: DataFrame cabe en caché. Shuffle: operaciones distribuidas mueven menos datos. Planificador: Spark Catalyst optimiza mejor. Una línea de código, mejora significativa.

**27. Vistas temporales vs tablas Delta persistentes**

**createOrReplaceTempView:** Vista en memoria, válida solo en sesión actual. Se pierde al reiniciar. Útil análisis ad-hoc.
**saveAsTable:** Escribe a disco como Delta persistente. Accesible desde otros notebooks, sesiones. Útil datasets reutilizados.

**28. Castings y columnas derivadas con withColumn**

**Casting:** df.withColumn('FechaStr', to_date('DateString', 'yyyy-MM-dd'))
**Columna derivada:** df.withColumn('GananciaMargen', df.PrecioVenta - df.PrecioCosto)
**Condicional:** df.withColumn('Categoria', when(df.Ventas > 1000, 'Alto').otherwise('Bajo')) withColumn es forma estándar extender DataFrame sin sobrescribir completamente.

**29. Buenas prácticas para grandes volúmenes**

Particionamiento (dividir por fecha, región). Caché selectivo (solo DataFrames reutilizados). Broadcast joins (dimension pequeña ■ fact grande). Evitar CSV (usar Parquet/Delta). Columnar projection (select columnas). Persist (control fino).

## KQL, EVENTHOUSE Y MATERIALIZED VIEWS

### 30. ¿Qué es Kusto Query Language (KQL)?

Lenguaje de consulta optimizado para analítica de **altos volúmenes de datos de eventos** (logs, telemetría, IoT, seguridad). Sintaxis parecida SQL pero operadores optimizados para series de tiempo, patrones. Escala a terabytes en segundos. Casos: monitoreo aplicaciones (APM), seguridad, dashboards tiempo real.

### 31. Stored functions en KQL

Encapsulan lógica reutilizable (joins, filtros, proyecciones).
Beneficios: reutilización, consistencia (un cambio afecta todos), mantenimiento centralizado. Ejemplo: MyCustomerSales() → múltiples consultas invocan y ven mismo comportamiento.

### 32. Operador 'take': uso y limitaciones

take N devuelve N filas (sin orden específico). Uso: muestreo rápido, esquemas, valores típicos. Limitaciones: no es aleatorio estadísticamente (puede no ser representativo), sin orden previo es arbitrario. Para muestreo: Logs | sort by Timestamp desc | take 1000. Útil exploración, no análisis estadístico serio.

### 33. Database shortcuts en KQL

Permiten referenciar base externa (Azure Data Explorer, otra instancia KQL) como si fuera local, sin copiar. Ventaja: evita duplicación storage, datos sincronizados en única copia, habilita análisis federado. Caso: datos raw en workspace centralizado, analistas en otros acceden vía shortcuts.

### 34. Filtros de tiempo al inicio en KQL

KQL tablas eventos casi siempre indexadas por timestamp. Aplicar 'where Timestamp > ago(1d)' al inicio: aprovecha índices (salta a datos 1 día atrás), reduce volumen escaneado (terabytes → gigabytes), mejora latencia (milisegundos vs minutos). Buena práctica: filtro tiempo primero, luego otros filtros.

### 35. Ingesta continua desde fuentes no-Microsoft

Para Kinesis, Kafka, etc., usan **conectores de streaming**: configurar conexión, definir mapeo campos, definir tabla destino. El conector ingiere continuamente en tiempo casi-real (~1-5s latencia). Ventaja: vs cargas batch manuales, datos siempre frescos.

### 36. format_datetime y funciones de fecha/hora

**format_datetime:** Convierte datetime a string. format_datetime(now(), 'yyyy-MM-dd HH:mm:ss').
**Otras frecuentes:** ago(N) (intervalo relativo), bin(Timestamp, 1h) (agrupar por buckets), datetime_diff (diferencia tiempos), startofday/week/month (normalizar), now() (hora actual). Permiten análisis series de tiempo fluido.

### 37. Materialized views en KQL

Ejecutan continuamente consulta definida (normalmente agregaciones), almacenando resultados pre-calculados. Tipos de consultas beneficiadas: Agregaciones recurrentes (KPIs, dashboards, alertas). Mantenimiento: actualización incremental con nuevos datos, no re-procesa todo.

### 38. Trade-offs de materialized views

**Ventajas:** Velocidad (milisegundos), CPU/Costo bajo, ideales dashboards recurrentes.
**Trade-offs:** Almacenamiento adicional, diseño cuidadoso (cambiarla luego caro), latencia actualización pequeña, menos flexibilidad ad-hoc.
**Conclusión:** Ideales métricas fijas recurrentes; no para exploraciones ad-hoc variadas.

## RESUMEN FINAL

El ecosistema Fabric integra: almacenamiento unificado (OneLake + Delta-Parquet), transformación (Dataflows, Spark, SQL), orquestación (Pipelines). Entiende cuándo usar cada pieza: Dataflows para ETL bajo-código, Spark para análisis complejos, KQL para eventos alta frecuencia, Delta Lake como fundamento fiable.