MICROSOFT FABRIC – REAL-TIME INTELLIGENCE

Preguntas y Respuestas Detalladas (Eventstream, KQL, Dashboards, Activator)

1. REAL-TIME INTELLIGENCE: ACTIVATOR, EVENTHOUSE Y KQL

1. En un proyecto de Fabric, quieres automatizar notificaciones cuando la temperatura de un sensor supera un umbral. ¿Qué componente define la condición y dispara la alerta?

El componente clave es **Activator Rules**. En Microsoft Fabric Activator defines reglas que describen condiciones sobre tus eventos, como “Temperature > 40”. La regla se evalúa continuamente sobre los eventos que llegan desde fuentes (Eventstreams, Eventhouse, etc.). Cuando la condición se cumple, Activator dispara acciones configuradas (Teams, email, Power Automate, ejecución de pipelines). Así separas claramente la lógica de detección de anomalías (reglas) de la lógica de ingestión (Eventstreams) y almacenamiento (Eventhouse).

2. Optimizar rutas de entrega en tiempo real ante congestión: ¿qué componente dispara el re-enrutamiento?

Se usa **Activator**. Los patrones de tráfico se detectan en datos de streaming (por ejemplo vía Eventstream → Eventhouse → KQL). Activator consume esos eventos o resultados agregados, aplica reglas como “delay > X minutos en tramo Y” y, cuando se cumple, dispara acciones: llamar un flujo de Power Automate o un pipeline que recalcula la ruta y actualiza el sistema de logística. Activator es el “motor de orquestación de acciones basadas en eventos”.

3. Spikes de datos que indican fraude: ¿qué componente almacena y permite consultar eficientemente?

El componente adecuado es **Eventhouse**, una base de datos KQL optimizada para eventos de alta frecuencia. Eventstreams ingiere los eventos de transacciones hacia Eventhouse; allí se almacenan en tablas KQL con índices de tiempo. Luego se consultan con KQL para detectar patrones sospechosos (picos, outliers, comportamientos anómalos) y alimentar tanto dashboards como Activator para disparar acciones de fraude.

4. Paso necesario antes de visualizar datos en un dashboard usando KQL

Primero se debe **definir un KQL Queryset** que extraiga y procese los datos desde las tablas de Eventhouse. En el Queryset escribes la consulta (filtros, agregaciones, joins) que genera el dataset que el tile del Real-Time Dashboard mostrará. Luego ese queryset se conecta al tile como fuente. Sin esta capa, el dashboard no tiene lógica de consulta definida para representar la información en tiempo real.

5. Ventaja principal de KQL para análisis en tiempo real

KQL está optimizado para datos de series temporales y consultas de alta performance. El motor indexa por tiempo y usa estructuras internas que permiten escanear rápidamente grandes volúmenes de eventos, soportando filtros por time windows y agregaciones complejas (summarize) con baja latencia. Es ideal para logs, telemetría de IoT, trazas de aplicaciones y otros escenarios de Real-Time Analytics.

6. Rol de 'summarize' en KQL

El operador **summarize** agrega datos sobre una o más columnas usando funciones como *sum()*, *avg()*, *min()*, *max()*, *count()*. Es la forma principal de producir métricas agregadas en grandes volúmenes de datos. Por ejemplo:
Events | where Timestamp > ago(1d) | summarize AvgTemp=avg(Temperature) by bin(Timestamp, 1h) Agrupar y resumir reduce drásticamente el dataset a procesar en dashboards y vistas posteriores.

7. Configurar Activator para optimizar respuesta a anomalías de rendimiento

Se deben **definir reglas basadas en umbrales de métricas de rendimiento** (CPU, memoria, latencia, errores) que disparen acciones correctivas. Por ejemplo: “si errorRate > 5% y requestLatency > 2s durante 5 minutos, entonces ejecutar pipeline de escala de recursos o reinicio de servicios”. Activator deja de ser solo un sistema de alertas para convertirse en un mecanismo de

remediación automática.

8. ¿Qué feature usar para alertas automáticas ante patrones de datos?

La funcionalidad apropiada es **Activator**. Real-Time Dashboards visualizan; Eventhouse almacena y permite analizar; pero Activator es el que define reglas sobre stream de datos y dispara alertas/acciones automáticamente cuando detecta un patrón (p. ej., valores fuera de rango, caídas de servicio, anomalías de tráfico).

9. Componente esencial para que Activator ajuste recursos cuando el uso supera un umbral

Son necesarias las **reglas de Activator basadas en métricas de uso**. Las métricas de uso (CPU, memoria, costo) pueden llegar por Eventstream o estar en Eventhouse; las reglas se configuran para condiciones como “CPU > 80% por más de 10 minutos”. Al cumplirse, Activator dispara acciones que ajustan recursos (por ejemplo, pipelines que cambian el tamaño de un pool o desencadenan scripts de escalado).

2. EVENTSTREAMS: TRANSFORMACIONES Y DESTINOS

10. Routing basado en contenido hacia distintos destinos en Eventstreams

Se realiza mediante un **Derived stream**. Un derived stream se construye a partir de un stream base aplicando filtros y transformaciones; se le pueden asignar destinos específicos. Así, un stream principal puede generar sub-flujos: uno con solo alertas críticas hacia Activator y otro con todos los datos agregados hacia Lakehouse. Esto implementa content-based routing dentro del canvas de Eventstream.

11. Datos de Service Bus se transforman pero no llegan al destino

Si ves que el origen y las transformaciones funcionan, pero el destino no recibe nada, la causa más probable es una **configuración incorrecta del destino en el canvas** (URL, credenciales, formato, tabla, etc.) o que el endpoint esté caído. Debes revisar la configuración del destino, logs de entrega y estado de conectividad.

12. Transformación para calcular ventas por hora en un stream

Se utiliza una transformación de **Group by** (agregación) en Eventstream. Configuras el group by para agrupar por una clave temporal (por ejemplo, ventana de una hora) y quizá por tienda o producto, y aplicas funciones de agregación como SUM(SalesAmount). El resultado es un flujo de eventos agregados por hora listo para almacenar o notificar.

13. Integrar datos de eventos en una aplicación propia en tiempo real

Para alimentar una aplicación personalizada en tiempo real desde Eventstreams, se configura un destino de tipo **Custom endpoint**. Este suele ser una API REST o endpoint HTTP/HTTPS que la app expone. Eventstream empuja los eventos transformados a ese endpoint, permitiendo que la app reaccione inmediatamente a nuevos datos.

14. Transformación para filtrar datos inválidos

Se usa la transformación **Filter**. Configuras expresiones que descartan eventos con valores fuera de rango, nulos o con tipos incorrectos. Ejemplo: solo pasar eventos con Temperature entre -50 y 80 y DeviceId no nulo. Esto limpia el stream antes de enviar a almacenamiento o Activator.

15. Transformación ok, pero datos no llegan a ningún destino

Una causa probable es que **el endpoint de destino esté mal configurado o sea inaccesible** (firewall, DNS, credenciales, permisos). Aunque la lógica de transformación sea correcta, si Eventstream no consigue conectarse al destino o escribir en él, los eventos no se entregarán.

16. Quitar eventos donde el customer type no es 'premium'

También aquí se utiliza una transformación de **Filter**. Definirías una condición como CustomerType == 'premium' para que el stream solo retenga eventos de clientes premium. El resto se descarta, reduciendo ruido y volumen en el pipeline.

17. Solo high-temperature alerts a notificación y promedios horarios a histórico

Se usa la funcionalidad de **content-based routing** mediante derived streams: un derived stream filtra eventos de altas temperaturas para enviarlos a un destino de notificaciones (como Activator), mientras otro stream aplica agregaciones horarias y envía el resultado a Lakehouse o Eventhouse para análisis histórico.

18. Diseño completo: Event Hubs → agregación → Lakehouse

La combinación correcta es: **Azure Event Hubs como origen**, una transformación de **Aggregate** (Group by) en el canvas de Eventstream para calcular métricas (por ejemplo por minuto/hora), y un **Lakehouse como destino**. Así obtienes un pipeline de streaming que ingiere, agrega y persiste en formato Delta en el lakehouse.

3. KQL, MATERIALIZED VIEWS Y DATABASE SHORTCUTS

19. Integrar streaming desde un broker MQTT externo en una base KQL

Se debe usar un **conector de fuente de streaming no-Microsoft** (non-Microsoft streaming source connector). Este conector lee del broker MQTT y empuja los eventos a Eventhouse o Eventstream, donde luego se almacenan en tablas KQL para consultas en tiempo real.

20. ¿Cómo mejoran rendimiento las materialized views en KQL?

Las **materialized views** guardan resultados de consultas de agregación precomputadas, de manera incremental. En lugar de recalcular desde datos crudos cada vez, las consultas posteriores leen agregados ya almacenados, lo que reduce enormemente el tiempo de respuesta y el uso de CPU.

21. Consultas lentas de métricas diarias IoT: ¿qué hacer?

La mejor opción es crear **materialized views** que almacenen métricas diarias precomputadas (por sensor, por región, etc.). Así, los dashboards que piden promedios diarios ya leen la vista materializada, sin escanear toda la tabla de eventos crudos.

22. Escenario típico para materialized views

La ventaja se ve cuando se **almacenan resultados de agregación precomputados para fuentes de streaming**. Por ejemplo, KPIs de IoT agregadas por minuto/hora que se usan en dashboards frecuentes. En lugar de agregar millones de eventos cada vez, se consulta directamente la vista materializada, muchas veces más rápida.

23. ¿Por qué poner la tabla más pequeña primero en un join de KQL?

Colocar la tabla más pequeña primero puede ayudar a **minimizar filas procesadas y memoria usada** durante el join, especialmente en algunos patrones de implementación. La idea general: siempre que se pueda, reducir cuánto dato entra en la parte más costosa de la operación (join + shuffle). Menos filas = mejor eficiencia.

24. Ejemplo de uso de materialized view

Un escenario típico es **agregar ventas mensuales para un dashboard que se actualiza cada hora**. La vista materializada mantiene el total por mes y categoría; cada actualización sólo añade los nuevos eventos, no recalcula todo el histórico desde cero. El dashboard lee la vista y responde rápido.

25. Función para formatear datetime a 'yyyy-MM-dd'

Se usa **format_datetime**. Por ejemplo: format_datetime(Timestamp, 'yyyy-MM-dd') devuelve una cadena de texto con la fecha formateada. Es útil para agrupar por día, construir labels o exportar datos legibles.

26. Significado de 'summarize' en optimización de consultas

El operador **summarize** permite agrupar y agregar datos, lo que reduce el volumen para análisis posterior. Aplicarlo pronto en la consulta reduce filas, mejora uso de memoria y velocidad. Es la base para optimizar consultas sobre grandes datasets en KQL.

27. Feature para consultar Azure Data Explorer desde tu Eventhouse

Se utilizan **database shortcuts**. Un atajo de base permite exponer una base externa (por ejemplo, en Azure Data Explorer) como si fuera parte de tu Eventhouse, sin copiar físicamente los datos. Ahorras almacenamiento y mantienes una sola fuente de verdad.

4. REAL-TIME DASHBOARDS: BASE QUERIES, TILES Y UX

28. El dashboard en tiempo real no se actualiza: ¿qué revisar primero?

Lo primero es verificar que la opción de **auto-refresh** esté configurada correctamente. Si el tile no se actualiza, puede que la frecuencia de actualización sea muy baja o que el auto-refresh esté deshabilitado. Sin auto-refresh, tendrás que refrescar manualmente para ver nuevos datos.

29. Dashboard de disponibilidad de bicis está saturado

La estrategia más efectiva es **organizar el dashboard en múltiples páginas** segmentadas por tipo de datos (por ejemplo: overview, por barrio, por tiempo, alertas). Así no sobrecargas una sola página con demasiados tiles y mejoras la navegación y la experiencia de usuario.

30. Varias tiles con queries similares: cómo mejorar mantenibilidad

La mejor práctica es implementar una **base query** que contenga la lógica compartida de obtención de datos y que cada tile la referencie, aplicando sólo filtros o visualizaciones específicas. De este modo, si cambian las reglas de negocio o la query base, solo se actualiza un sitio y todos los tiles se benefician.

31. Optimizar dashboard con múltiples tiles que comparten datos

Usar **base queries** evita duplicar lógica KQL en cada tile. Esto reduce errores, facilita cambios y mejora rendimiento, ya que la base query puede estar optimizada con filtros y summarizations adecuados que sirven para todos los tiles derivados.

32. Componente crítico para visualización en tiempo real en un nuevo tile

Lo esencial es una **consulta KQL correctamente configurada** apuntando a la fuente correcta (Eventhouse, base query, etc.) y con el filtro temporal adecuado. Sin una query correcta, el tile no mostrará datos correctos independientemente del diseño visual.

33. Dashboard de bicis de nuevo saturado: mejor método de reorganización

De nuevo, la recomendación es **usar múltiples páginas para diferentes tipos de datos** en vez de intentar poner todo en una sola. También conviene agrupar tiles relacionados y usar títulos/secciones claros.

34. Base queries para evitar escribir lógica repetida en tiles

Las base queries permiten **consolidar lógica compartida** en una única consulta que luego se reutiliza en varios tiles. Esto evita copias casi idénticas de la misma query, reduce errores de mantenimiento y asegura coherencia entre visualizaciones.

35. Mostrar disponibilidad por barrio

La mejor opción es habilitar **parámetros interactivos** para filtrar por barrio. El usuario puede seleccionar un barrio desde un control y el tile ejecuta la misma base query pero aplicada al filtro del parámetro. Esto evita crear tiles estáticos por barrio y mejora la experiencia.

36. ¿Por qué se recomiendan base queries cuando hay múltiples tiles?

Porque permiten **evitar duplicación de lógica** y asegurar que todas las tiles que muestran el mismo tipo de datos parten de la misma definición. Esto simplifica mantenimiento, reduce bugs y contribuye a que todos vean la misma “verdad” en el dashboard.

5. ACTIVATOR: REGLAS, ACCIONES Y CONFIGURACIÓN

37. Regla de Activator no se dispara aunque la temperatura supera el umbral

Lo primero que hay que revisar es si **la condición de la regla está bien configurada** (por ejemplo, Temperature > 40 en vez de Temperature >= 40 o < 40 por error). Un fallo en el operador o en la propiedad usada hace que la regla nunca se cumpla, aunque los datos sí superen el umbral.

38. Método de resumen para suavizar ruido en lecturas de temperatura

Se debe usar **Average** sobre una ventana de tiempo. Por ejemplo, promedio de temperatura en los últimos 5 minutos en lugar de un valor puntual. Esto reduce disparos por picos breves o ruido, activando la regla solo cuando existe una tendencia sostenida fuera de rango.

39. Automatizar restarts/analítica mediante pipelines desde Activator

Para ejecutar data pipelines u otros artefactos de Fabric cuando una condición se cumple, se usan **Fabric item actions**. Estas acciones permiten iniciar un pipeline, notebook u otro item directamente, integrando Activator con la capa de procesamiento de Fabric.

40. Acción adecuada para flujos multi-step en distintos sistemas

Cuando necesitas orquestar un proceso que involucra múltiples sistemas (CRM, ERP, ITSM, etc.), se configuran **Power Automate actions**. Activator dispara el flujo de Power Automate, que puede llamar APIs, actualizar registros, enviar correos, crear tickets, etc., todo en una cadena de pasos automatizados.

41. Sección donde se configuran las acciones en una regla de Activator

Las acciones se definen en la sección **Actions** de la configuración de la regla. Allí eliges el tipo de acción (Teams, email, Power Automate, Fabric item) y rellenas los detalles (destinatarios, item a ejecutar, etc.).

42. Pasos clave para monitorizar flota de camiones refrigerados

La secuencia correcta es:
1) **Conectar a las fuentes de datos** (Eventstream/Eventhouse con temperaturas de camiones).
2) **Agrupar eventos en objetos** (por TruckId) para tener una entidad “camión” con su historial de lecturas.
3) **Definir reglas** sobre esos objetos (temperatura media > umbral durante N minutos) y configurar acciones.

43. Acción adecuada para notificar al equipo de logística cuando T > 40°F

Una opción muy efectiva es usar **Teams actions**, que envían mensajes inmediatos a un canal o chat de Teams del equipo de logística. Pueden incluir detalles del camión, valor de temperatura, ubicación, y enlaces a dashboards.

44. Envío sobre umbral sin respuesta automática: posible causa

Una causa común es que **la condición de la regla no está configurada para disparar cuando la temperatura excede el umbral** (por ejemplo, se puso > 50 en vez de > 40, o se filtró el camión equivocado en la sección de Property filter). Revisar la lógica de la condición es el primer paso del troubleshooting.

45. Sección para limitar qué eventos evalúa la regla

La sección **Property filter** permite acotar el conjunto de eventos que se consideran para la regla. Por ejemplo, solo camiones refrigerados, o solo una ruta específica. De esta forma, las reglas no se evaluan sobre datos irrelevantes y se evita ruido.

RESUMEN

Este documento compila las preguntas del bloque de Real-Time Intelligence de Microsoft Fabric y las responde en detalle, cubriendo Eventstreams, Eventhouse, KQL, Real-Time Dashboards y Activator. Puedes usarlo como referencia de estudio para DP-700 y como guía conceptual para diseñar soluciones de streaming y acciones basadas en eventos en Fabric.