GUÍA RÁPIDA MICROSOFT FABRIC / MONITORIZACIÓN Y OBSERVABILIDAD / INTERMEDIO

Monitorear pipelines y notebooks de Fabric

Monitorea pipelines y notebooks de Microsoft Fabric con Monitoring hub, Workspace Monitoring, el monitoreo de Spark, KQL, alertas y telemetría de capacidad para que los errores y las regresiones de rendimiento sean observables.



Actualizado 4 oct 2026 Versión en línea →

PROBLEMA

Los pipelines y notebooks de Fabric pueden completarse, fallar, ralentizarse, reintentarse, quedar en cola o consumir más capacidad de lo esperado.

Abrir las páginas de cada ejecución por separado basta para un incidente puntual.

No basta para las operaciones en producción.

Necesitas responder preguntas como:

- ¿Qué pipelines fallaron hoy?
- ¿Qué actividad falló primero?
- ¿Qué notebooks se vuelven más lentos con el tiempo?
- ¿La lentitud se debe a la ejecución de Spark o a la orquestación del pipeline?
- ¿Los errores se concentran en un notebook o en un tipo de actividad?
- ¿Qué ejecuciones fueron programadas y cuáles bajo demanda?
- ¿Los jobs quedan en cola por presión sobre la capacidad?
- ¿Un despliegue cambió el comportamiento en tiempo de ejecución?
- ¿Debe el equipo recibir una alerta automáticamente?

El objetivo es combinar **monitoreo operativo**, **análisis histórico de logs**, **diagnóstico de Spark** y **telemetría de capacidad** sin construir desde cero una plataforma de monitoreo a medida.

RESPUESTA RÁPIDA

Usa la capa de monitoreo que corresponda a la pregunta:

```
¿Necesitas inspeccionar rápidamente una ejecución actual o reciente?
→ Monitoring hub

¿Necesitas el detalle de las actividades del pipeline?
→ Detalle de la ejecución del pipeline / historial de ejecuciones de actividades

¿Necesitas los stages, tareas, executors y logs de Spark del notebook?
→ Monitoreo de aplicaciones Spark / Spark UI

¿Necesitas un análisis histórico de todo el workspace?
→ Workspace Monitoring + Eventhouse + KQL

¿Necesitas alertas?
→ Alertas de monitoreo / Activator / flujo de alertas compatible
```

```
¿Necesitas contexto del consumo de capacidad?
→ Aplicación Fabric Capacity Metrics
```

Para el estado de los pipelines en todo el workspace:

```kql
ItemJobEventLogs
| where ItemKind == "Data Pipeline"
| summarize Runs = count() by JobStatus
| order by Runs desc
```

Para las actividades de pipeline con errores:

```kql
FabricDataPipelinesActivityRunsLogs
| where Status == "Failed"
| project
    Timestamp,
    ItemName,
    PipelineRunId,
    ActivityName,
    ActivityType,
    FailureType,
    DurationMs
| order by Timestamp desc
```

Para el historial de jobs de notebooks:

```kql
ItemJobEventLogs
| where ItemKind == "Notebook"
| where JobType in (
    "RunNotebook",
    "RunNotebookInteractive",
    "PipelineRunNotebook"
)
| project
    Timestamp,
    ItemName,
    JobType,
    JobStatus,
    DurationMs,
    JobStartTime,
    JobEndTime,
    JobInstanceId
| order by Timestamp desc
```

Usa la interfaz para inspeccionar rápidamente un incidente.

Usa KQL cuando necesites tendencias, comparaciones, automatización y visibilidad operativa recurrente.

Código / comandos

1. Usa Monitoring hub para un triaje operativo rápido

Abre:

```
Fabric
→ Monitor
```

Usa Monitoring hub cuando necesites:

- encontrar ejecuciones recientes de pipelines
- encontrar ejecuciones de notebooks/Spark
- filtrar por estado
- filtrar por tipo de elemento
- revisar la duración de la ejecución
- abrir el detalle de las ejecuciones
- identificar relaciones de ejecución upstream/downstream
- reintentar las ejecuciones de pipeline que lo admitan

Suele ser el punto de partida más rápido durante un incidente.

No escribas primero una consulta KQL si ya ves en el portal la ejecución que falló.

---

2. Abre el detalle de la ejecución del pipeline para encontrar la actividad que falló

Un pipeline puede contener muchas actividades, pero quizá solo una sea el error original.

Usa:

```
Monitoring hub
→ Pipeline run
→ Run details
→ Activity runs
```

Anota:

```
nombre del pipeline
ID de ejecución
nombre de la actividad
tipo de actividad
estado
hora de inicio
hora de fin
duración
tipo de error
```

Sigue el primer error real en lugar de las actividades omitidas aguas abajo.

---

3. Usa el monitoreo contextual del notebook durante el desarrollo

Dentro de un notebook de Fabric, el monitoreo contextual de Spark puede mostrar el progreso de la ejecución a nivel de notebook o de celda.

Úsalo para inspeccionar:

- los jobs de Spark que lanza una celda
- los stages
- las tareas
- los executors
- los logs
- los errores de ejecución
- las recomendaciones de Spark advisor

Es útil cuando:

```
el notebook es lento
```

pero todavía necesitas determinar si la lentitud de la celda se debe a:

- scan
- shuffle
- skew
- un fallo de executor
- presión de memoria
- la configuración de Spark

---

4. Usa el monitoreo de aplicaciones Spark para el rendimiento de los notebooks

Abre la página de detalle de la aplicación Spark desde:

```
Monitoring hub
```

o desde:

```
Notebook
→ Recent runs
```

Las áreas importantes incluyen:

```
Jobs
Resources
Logs
Diagnostics
```

La vista Jobs puede mostrar:

- la duración del job
- los stages
- las tareas
- los datos leídos
- los datos escritos
- el fragmento de código

La vista Resources ayuda a correlacionar el job de Spark con el uso de los executors.

La vista Logs ofrece:

- logs del driver
- logs de Livy
- logs de prelanzamiento (prelaunch)

Usa Spark UI cuando necesites un detalle de stages y tareas de más bajo nivel.

---

5. Usa Spark UI para el diagnóstico a nivel de stage

Spark UI es la herramienta adecuada cuando la pregunta pasa a ser:

```
¿Por qué esta aplicación Spark tardó 25 minutos?
```

Inspecciona:

```
Jobs
Stages
Storage
Environment
Executors
```

Ejemplos:

```
Una tarea mucho más lenta que las demás
→ posible skew

Lectura/escritura de shuffle grande
→ redistribución por join / agregación

Errores de executors
→ problema de memoria/recursos/runtime

Conjuntos de datos en caché grandes
→ presión sobre el almacenamiento/caché

Configuración de Spark inesperada
→ deriva del entorno/la configuración
```

No te fíes solo de la duración total (wall-clock) del notebook.

6. Habilita Workspace Monitoring para el análisis histórico

Workspace Monitoring crea un Eventhouse de monitoreo y una base de datos KQL de monitoreo de solo lectura para los elementos de Fabric compatibles.

Habilítalo desde:

```
Workspace settings
→ Monitoring
→ Add / enable monitoring Eventhouse
```

Usa Workspace Monitoring cuando necesites:

- tendencias históricas de ejecución
- errores en todo el workspace
- análisis recurrente del rendimiento
- KQL personalizado
- dashboards
- lógica de alertas
- reportes operativos

Este es el cambio clave, que consiste en pasar de:

```
inspeccionar una ejecución
```

a:

```
operar la plataforma
```

7. Entiende las dos tablas principales de monitoreo de pipelines

Monitoreo a nivel de pipeline:

```
ItemJobEventLogs
```

Monitoreo de pipelines a nivel de actividad:

```
FabricDataPipelinesActivityRunsLogs
```

Usa:

```
ItemJobEventLogs
```

para preguntas como:

- ¿cuántos pipelines fallaron?
- ¿qué pipeline se está volviendo más lento?
- ¿qué jobs son programados y cuáles bajo demanda?
- ¿quién lanzó la ejecución?

Usa:

```
FabricDataPipelinesActivityRunsLogs
```

para preguntas como:

- ¿qué actividad falló?
- ¿qué tipo de actividad es lento?
- ¿dónde se acumula el tiempo del pipeline?
- ¿cuánta CPU consumió una actividad, cuando el dato está disponible?

---

8. Inspecciona el esquema de monitoreo antes de escribir KQL de producción

Empieza con:

```kql
ItemJobEventLogs
| getschema
```

y:

```kql
FabricDataPipelinesActivityRunsLogs
| getschema
```

Hazlo porque los esquemas de monitoreo pueden evolucionar.

No des por hecho que el código copiado de un artículo antiguo o de otro tenant coincide exactamente con la tabla actual.

---

9. Cuenta las ejecuciones de pipelines por estado

```kql
ItemJobEventLogs
| where ItemKind == "Data Pipeline"
| summarize Runs = count() by JobStatus
| order by Runs desc
```

Así obtienes un resumen rápido del estado de salud.

Ten cuidado cuando el modelo de logs incluya varios eventos de ciclo de vida para la misma instancia de job.

Para reportar ejecuciones completadas, filtra por estados terminales.

10. Cuenta solo los resultados terminales de los pipelines

```kql
ItemJobEventLogs
| where ItemKind == "Data Pipeline"
| where JobStatus in ("Completed", "Failed")
| summarize Runs = count() by JobStatus
```

Usa los estados terminales para reportar éxitos y errores.

No cuentes:

```
En curso
No iniciado
```

como ejecuciones completadas.

11. Muestra los pipelines que fallaron recientemente

```kql
ItemJobEventLogs
| where ItemKind == "Data Pipeline"
| where JobStatus == "Failed"
| project
    Timestamp,
    WorkspaceName,
    ItemName,
    JobInstanceId,
    JobInvokeType,
    JobStartTime,
    JobEndTime,
    DurationMs,
    ExecutingPrincipalType
| order by Timestamp desc
```

Usa el `JobInstanceId` para correlacionar una ejecución concreta con el portal o con una investigación posterior a nivel de ejecución.

12. Muestra las actividades de pipeline que fallaron recientemente

```kql
FabricDataPipelinesActivityRunsLogs
| where Status == "Failed"
| project
    Timestamp,
    ItemName,
    PipelineRunId,
    ActivityName,
    ActivityType,
    OperationStartTime,
    OperationEndTime,
    DurationMs,
    FailureType
| order by Timestamp desc
```

Esto resulta mucho más accionable que un recuento de errores a nivel de pipeline cuando el mismo pipeline contiene muchas actividades.

13. Encuentra las actividades que fallan con más frecuencia

```kql
FabricDataPipelinesActivityRunsLogs
| where Status == "Failed"
| summarize
    Failures = count()
    by ItemName, ActivityName, ActivityType
| order by Failures desc
```

Úsalo para priorizar los problemas estructurales de confiabilidad.

Una actividad que falla repetidamente merece más atención que un único error transitorio aislado.

14. Resume la duración de las actividades de pipeline

```kql
FabricDataPipelinesActivityRunsLogs
| where Status == "Succeeded"
| summarize
    Runs = count(),
    AvgDurationMs = avg(DurationMs),
    P95DurationMs = percentile(DurationMs, 95),
    MaxDurationMs = max(DurationMs)
    by ItemName, ActivityName, ActivityType
| order by P95DurationMs desc
```

Usa un análisis basado en percentiles para las tendencias de rendimiento.

El promedio por sí solo puede ocultar ejecuciones lentas intermitentes.

No interpretes el P95 sin un historial suficiente y representativo.

15. Encuentra regresiones en la duración de las actividades de pipeline

Una comparación sencilla entre el periodo reciente y el anterior:

```kql
let RecentStart = ago(1d);
let BaselineStart = ago(8d);
```

```
let Recent =
    FabricDataPipelinesActivityRunsLogs
    | where Status == "Succeeded"
    | where Timestamp >= RecentStart
    | summarize
        RecentAvgMs = avg(DurationMs)
        by ItemName, ActivityName;

let Baseline =
    FabricDataPipelinesActivityRunsLogs
    | where Status == "Succeeded"
    | where Timestamp between (BaselineStart .. RecentStart)
    | summarize
        BaselineAvgMs = avg(DurationMs)
        by ItemName, ActivityName;

Recent
| join kind=inner Baseline
    on ItemName, ActivityName
| extend
    DurationRatio = RecentAvgMs / BaselineAvgMs
| order by DurationRatio desc
```

Es una comparación de triaje, no un veredicto automático sobre el rendimiento.

Revisa:

- el volumen de entrada
- la latencia del origen
- los cambios de código
- la presión sobre la capacidad

antes de calificar una ejecución como regresión.

---

16. Encuentra las actividades de pipeline recientes más lentas

KQL

```
FabricDataPipelinesActivityRunsLogs
| where Status == "Succeeded"
| project
    Timestamp,
    ItemName,
    PipelineRunId,
    ActivityName,
    ActivityType,
    DurationMs
| order by DurationMs desc
| take 50
```

Úsalo para detectar rápidamente valores atípicos.

Después compara cada actividad con su propia línea base habitual.

---

17. Encuentra los errores de actividades de pipeline por tipo

```kql
FabricDataPipelinesActivityRunsLogs
| where Status == "Failed"
| summarize
    Failures = count()
    by FailureType, ActivityType
| order by Failures desc
```

Úsalo para distinguir patrones como:

```
Errores de notebooks
Errores de copia
Errores relacionados con la puerta de enlace
Otras categorías de errores recurrentes
```

No trates `FailureType` como un mensaje completo de causa raíz.

Es posible que los logs de actividad actuales del workspace no contengan todas las cadenas de error detalladas disponibles en la interfaz de la ejecución.

---

18. Muestra el uso de CPU de las actividades de pipeline cuando esté disponible

```kql
FabricDataPipelinesActivityRunsLogs
| where Status == "Succeeded"
| summarize
    TotalCpuCoreMs = sum(CpuCoreMs),
    AvgCpuCoreMs = avg(CpuCoreMs)
    by ItemName, ActivityName, ActivityType
| order by TotalCpuCoreMs desc
```

Úsalo como una señal operativa más.

No compares métricas de CPU entre tipos de actividad fundamentalmente distintos sin contexto.

---

19. Encuentra ejecuciones de notebooks desde el monitoreo del workspace

`ItemJobEventLogs` incluye los tipos de job de Notebook compatibles.

Ejemplo:

```kql
ItemJobEventLogs
| where ItemKind == "Notebook"
| where JobType in (
    "RunNotebook",
    "RunNotebookInteractive",
    "PipelineRunNotebook"
)
| project
    Timestamp,
    WorkspaceName,
    ItemName,
    JobType,
    JobStatus,
    JobInvokeType,
    JobStartTime,
    JobEndTime,
    DurationMs,
```

```
    JobInstanceId,
    ExecutingPrincipalType
| order by Timestamp desc
```

Úsalo para obtener visibilidad histórica a nivel de notebook.

Para el detalle de stages y tareas de Spark, pasa al monitoreo de Spark en lugar de esperar que los logs de jobs del workspace contengan métricas de Spark de bajo nivel.

20. Encuentra las ejecuciones de notebooks que fallaron

KQL

```
ItemJobEventLogs
| where ItemKind == "Notebook"
| where JobType in (
    "RunNotebook",
    "RunNotebookInteractive",
    "PipelineRunNotebook"
)
| where JobStatus == "Failed"
| project
    Timestamp,
    ItemName,
    JobType,
    JobInstanceId,
    JobStartTime,
    JobEndTime,
    DurationMs
| order by Timestamp desc
```

Úsalo para identificar qué notebooks fallan repetidamente.

Después abre la aplicación Spark correspondiente para hacer el diagnóstico real de Spark.

21. Sigue las tendencias de duración de los notebooks

KQL

```
ItemJobEventLogs
| where ItemKind == "Notebook"
| where JobStatus == "Completed"
| where JobType in (
    "RunNotebook",
    "PipelineRunNotebook"
)
| summarize
    Runs = count(),
    AvgDurationMs = avg(DurationMs),
    P95DurationMs = percentile(DurationMs, 95),
    MaxDurationMs = max(DurationMs)
    by ItemName
| order by P95DurationMs desc
```

Es útil para cargas de trabajo de notebooks recurrentes.

Separa las ejecuciones interactivas de desarrollo cuando no sean comparables con las ejecuciones programadas de producción.

22. Separa las ejecuciones de notebooks por tipo de invocación

```kql
ItemJobEventLogs
| where ItemKind == "Notebook"
| where JobStatus == "Completed"
| summarize
    Runs = count(),
    AvgDurationMs = avg(DurationMs)
    by ItemName, JobType, JobInvokeType
| order by ItemName asc
```

Esto ayuda a no mezclar:

```
desarrollo interactivo manual
```

con:

```
ejecución programada en producción
```

en una misma línea base de rendimiento.

---

23. Sigue el comportamiento de los pipelines programados frente a los ejecutados bajo demanda

```kql
ItemJobEventLogs
| where ItemKind == "Data Pipeline"
| where JobStatus in ("Completed", "Failed")
| summarize
    Runs = count(),
    Failures = countif(JobStatus == "Failed"),
    AvgDurationMs = avg(DurationMs)
    by ItemName, JobInvokeType
| order by ItemName asc
```

Esto puede revelar que un pipeline se comporta de forma distinta cuando:

- está programado
- se ejecuta manualmente
- se lanza en otra ventana de carga

---

24. Monitorea un pipeline concreto

```kql
ItemJobEventLogs
| where ItemKind == "Data Pipeline"
| where ItemName == "Daily Sales"
| where JobStatus in ("Completed", "Failed")
| project
    Timestamp,
    JobStatus,
    JobInvokeType,
    JobStartTime,
    JobEndTime,
    DurationMs,
    JobInstanceId
| order by Timestamp desc
```

Usa consultas específicas de un pipeline cuando diagnostiques una carga de trabajo conocida.

25. Monitorea las actividades de un pipeline

```kql
FabricDataPipelinesActivityRunsLogs
| where ItemName == "Daily Sales"
| project
    Timestamp,
    PipelineRunId,
    ActivityName,
    ActivityType,
    Status,
    DurationMs,
    FailureType
| order by Timestamp desc
```

Filtra además por `PipelineRunId` para reconstruir una ejecución.

26. Reconstruye una ejecución de pipeline

```kql
let RunId = "<pipeline-run-id>";

FabricDataPipelinesActivityRunsLogs
| where PipelineRunId == RunId
| project
    ActivityName,
    ActivityType,
    Status,
    OperationStartTime,
    OperationEndTime,
    DurationMs,
    FailureType
| order by OperationStartTime asc
```

Así obtienes una vista cronológica de la ejecución de las actividades.

Es útil para identificar:

- dónde se acumuló el tiempo
- qué actividad falló primero
- si las actividades se solaparon

27. Identifica los jobs de larga duración con un umbral específico de la carga de trabajo

Ejemplo:

```kql
let LongRunMs = 60 * 60 * 1000;

ItemJobEventLogs
| where ItemKind == "Data Pipeline"
| where JobStatus == "Completed"
| where DurationMs > LongRunMs
| project
    Timestamp,
    ItemName,
    DurationMinutes = round(DurationMs / 60000.0, 1),
    JobStartTime,
    JobEndTime,
    JobInstanceId
```

```
| order by DurationMs desc
```

El umbral de 60 minutos es solo un ejemplo.

Define los umbrales a partir de:

- el SLA
- la programación prevista
- la línea base histórica
- la ventana de negocio

No inventes un umbral universal de «pipeline lento».

28. Detecta errores de pipelines en una ventana de alerta reciente

```kql
ItemJobEventLogs
| where ItemKind == "Data Pipeline"
| where JobStatus == "Failed"
| where Timestamp >= ago(10m)
| project
    Timestamp,
    WorkspaceName,
    ItemName,
    JobInstanceId,
    JobStartTime,
    JobEndTime
| order by Timestamp desc
```

Este tipo de consulta puede alimentar un flujo de alertas.

Mantén alineados el intervalo de sondeo de la alerta y la ventana de tiempo de la consulta para que los errores no se pierdan ni generen alertas repetidas innecesariamente.

29. Crea alertas solo para condiciones accionables

Ejemplos de buenas alertas:

```
falló un pipeline de producción
falló un notebook crítico
un pipeline superó su SLA
errores repetidos en un periodo corto
una ejecución programada no terminó en la ventana esperada
```

Ejemplos de malas alertas:

```
cada notebook que se inicia
cada reintento
cada advertencia
cada evento transitorio que se recupera solo
```

La fatiga de alertas reduce el valor operativo.

30. Usa los identificadores de instancia de job para correlacionar

`JobInstanceId` identifica una instancia de job concreta de un elemento de Fabric.

Úsalo para correlacionar:

```
log de monitoreo
ejecución en el portal
alerta
incidente
log de ejecución
```

En los pipelines, el `PipelineRunId` a nivel de actividad es el campo de correlación de ejecución importante dentro de:

```
FabricDataPipelinesActivityRunsLogs
```

No correlaciones los incidentes de producción solo por nombre del elemento y marca de tiempo.

31. Añade tu propio identificador de ejecución de negocio

Los identificadores de job de Fabric son identificadores operativos.

Tu pipeline también puede necesitar un identificador a nivel de negocio:

```
batch_id
business_date
source_file_id
logical_run_id
```

Ejemplo en un notebook:

```python
from pyspark.sql import functions as F

output = (
    output
    .withColumn(
        "pipeline_run_id",
        F.lit(pipeline_run_id),
    )
    .withColumn(
        "batch_id",
        F.lit(batch_id),
    )
    .withColumn(
        "ingested_at_utc",
        F.current_timestamp(),
    )
)
```

Así puedes correlacionar la ejecución en la plataforma con los datos realmente escritos.

32. Persiste métricas operativas desde los notebooks

Entre las métricas útiles están:

```
rows_read
rows_written
rows_rejected
source_date_min
source_date_max
duplicate_key_count
null_key_count
target_table
batch_id
pipeline_run_id
notebook_name
status
```

Ejemplo:

```python
metrics = {
    "pipeline_run_id": pipeline_run_id,
    "batch_id": batch_id,
    "rows_written": output.count(),
    "target_table": target_table,
}
```

En producción, escribe métricas estructuradas en lugar de depender solo de diccionarios impresos.

Sé consciente de las acciones costosas, como las llamadas repetidas a `count()`.

---

33. Registra el estado de ejecución del pipeline de forma compatible con inserciones (append)

Una tabla lógica sencilla:

```
pipeline_execution_log
----------------------
pipeline_name
pipeline_run_id
activity_name
batch_id
status
started_at
completed_at
rows_read
rows_written
error_type
```

Prefiere registros de eventos de solo inserción (append-only) siempre que sea práctico.

Esto reduce la contención frente a muchos notebooks concurrentes que actualizan repetidamente la misma fila del log.

---

34. Evita que la tabla de monitoreo se convierta en un cuello de botella

Si muchos notebooks escriben simultáneamente en una misma tabla Delta de log, esa tabla puede convertirse en un destino de escritura muy disputado.

Considera:

```
registro de eventos solo de anexado
eventos operativos particionados
una fila por transición de actividad
conciliación centralizada
```

No crees una solución de monitoreo que provoque más errores de pipelines de los que observa.

35. Usa el monitoreo de series de ejecuciones de Fabric para cargas Spark recurrentes

Fabric puede agrupar las aplicaciones Spark recurrentes en series de ejecuciones (run series).

Usa las vistas de series de ejecuciones para comparar:

- la tendencia de duración
- la tendencia de datos de entrada/salida
- las ejecuciones anómalas

Es útil para los siguientes elementos recurrentes:

- notebooks
- definiciones de jobs de Spark
- actividades de notebook lanzadas por pipelines

Úsalo antes de construir una vista de tendencias personalizada cuando la comparación integrada ya responda la pregunta.

36. Compara ejecuciones de aplicaciones Spark cuando un notebook sufre una regresión

La comparación de aplicaciones Spark de Fabric permite comparar varias ejecuciones y resaltar diferencias de rendimiento.

Úsala cuando:

```
El notebook tardaba 8 minutos la semana pasada
El notebook ahora tarda 25 minutos
```

Compara:

- la duración
- la entrada/salida
- el comportamiento de ejecución de Spark
- los stages anómalos

Después revisa los cambios de código, configuración o datos.

37. Usa las API de monitoreo de Spark para automatizar cuando la interfaz no basta

Fabric expone API de monitoreo de Spark para obtener por programación la información de las aplicaciones Spark.

Usa las API cuando necesites:

- recopilar automáticamente los metadatos de las ejecuciones Spark de los notebooks
- integrar el monitoreo en otro sistema operativo
- construir un dashboard interno personalizado
- correlacionar los identificadores de aplicaciones Spark con sistemas externos de gestión de incidentes

Prioriza primero Monitoring hub y Workspace Monitoring integrados, salvo que la automatización con API aporte un valor real.

38. Usa Capacity Metrics cuando el problema pueda ser presión sobre la capacidad

La lentitud de un pipeline o de un notebook puede deberse a:

```
saturación de la capacidad
cargas de trabajo concurrentes
contención de recursos
throttling (limitación)
```

Usa la aplicación Microsoft Fabric Capacity Metrics para inspeccionar el consumo a nivel de capacidad.

Esto es especialmente importante cuando:

```
muchas cargas de trabajo no relacionadas se vuelven lentas al mismo tiempo
```

Ese patrón es distinto del de un único notebook ineficiente.

39. Correlaciona la capacidad de la plataforma con el horario de las cargas de trabajo

Cuando un pipeline se vuelve lento todos los días a las 09:00, compara:

```
duración del pipeline
duración de Spark del notebook
utilización de la capacidad
otros jobs ejecutándose al mismo tiempo
```

No optimices el código del notebook de forma aislada si el verdadero cuello de botella es la presión sobre una capacidad compartida.

40. Sigue un conjunto reducido de métricas de servicio operativas

Una línea base de producción útil es:

```
Fiabilidad
- total de ejecuciones
- tasa de éxito
- número de errores
- número de errores repetidos

Rendimiento
- duración media
- duración P95
- duración máxima
- tiempo en cola/de arranque si está disponible

Calidad de datos
- filas leídas/escritas
- filas rechazadas
- claves duplicadas
- claves obligatorias nulas

Recursos
- métricas de Spark de stage/runtime
- utilización de la capacidad
```

```
- CPU de actividades si está disponible
```

No recopiles métricas solo porque estén disponibles.

Cada métrica debe respaldar una decisión operativa.

Qué significa el resultado

Superficie de monitoreo	Mejor uso
Monitoring hub	Vista operativa rápida de ejecuciones actuales o recientes
Detalle de la ejecución del pipeline	Diagnóstico de pipelines a nivel de actividad
Monitoreo contextual del notebook	Ejecución de celdas/Spark durante el desarrollo
Detalle de la aplicación Spark	Análisis de jobs, recursos, logs y diagnósticos
Spark UI	Diagnóstico profundo de stages, tareas y executors
Workspace Monitoring	Telemetría histórica de todo el workspace
`ItemJobEventLogs`	Historial de ejecución de Fabric a nivel de elemento/job
`FabricDataPipelinesActivityRunsLogs`	Historial de pipelines a nivel de actividad
Series de ejecuciones / comparación de ejecuciones	Tendencias recurrentes de rendimiento de Spark
Capacity Metrics	Presión sobre la capacidad y consumo de las cargas de trabajo
Alertas / Activator	Notificar cuando se produce una condición operativa

La distinción clave es:

```
Estado del pipeline
!=
causa raíz de la actividad
!=
causa raíz de Spark
!=
causa raíz de capacidad
```

Baja por la pila de monitoreo solo hasta donde lo requiera el incidente.

Qué revisar

La tasa de errores aumenta en muchos pipelines no relacionados

Investiga:

- una dependencia de origen compartida
- las credenciales
- el gateway
- un problema del workspace o de la capacidad
- un incidente de la plataforma o del servicio

No empieces depurando cada pipeline por separado.

Una actividad falla repetidamente

Probablemente se trate de un problema estructural.

Inspecciona:

- los parámetros de la actividad
- el contrato de datos
- el comportamiento de los reintentos
- la identidad
- la dependencia de origen o destino (sink)

La duración del pipeline aumenta, pero la de Spark no

Revisa:

- el tiempo en cola
- los huecos de orquestación
- las actividades externas
- el inicio de sesión
- las esperas en origen o destino

Puede que el notebook no sea el cuello de botella.

La duración del notebook aumenta con un mayor volumen de entrada

Puede tratarse de un comportamiento de escalado esperado.

Normaliza las comparaciones según:

- las filas de entrada
- los bytes procesados
- el número de archivos
- el número de particiones

antes de calificarlo como regresión.

Una sola tarea de Spark domina un stage

Investiga el skew.

La duración a nivel de workspace no basta por sí sola para diagnosticarlo.

Varias cargas de trabajo se ralentizan a la vez

Revisa la presión sobre la capacidad antes de reescribir el código de cada una.

Los errores solo ocurren en las ejecuciones programadas

Compara:

- la identidad
- la ventana horaria de ejecución
- la disponibilidad del origen
- el solapamiento de programaciones
- el tipo de invocación
- la carga de la capacidad

Las ejecuciones programadas y las manuales no siempre son equivalentes desde el punto de vista operativo.

NOTAS DE PRODUCCIÓN

Workspace Monitoring es una funcionalidad de observabilidad para producción, pero conocer el esquema importa

El Eventhouse de monitoreo te da logs duraderos que se pueden consultar con KQL.

Inspecciona siempre el esquema actual antes de estandarizar dashboards o lógica de alertas.

Las tablas de monitoreo y su cobertura pueden evolucionar.

Workspace Monitoring tiene un costo de capacidad

El Eventhouse de monitoreo y los componentes de monitoreo relacionados consumen capacidad de Fabric.

La observabilidad no es gratuita.

Mantén los dashboards, las consultas y la retención alineados con el valor operativo real.

Workspace Monitoring es de solo lectura para los logs recopilados

Usa la base de datos de monitoreo para el análisis.

No la diseñes como la base de datos de control modificable de tu aplicación.

Los logs de jobs de elementos y los logs de actividades tienen distinta granularidad

`ItemJobEventLogs` es la vista general a nivel de job.

`FabricDataPipelinesActivityRunsLogs` es la vista más detallada de las actividades del pipeline.

Usa la granularidad más fina que necesite la pregunta.

Los mensajes de error detallados pueden seguir requiriendo la interfaz de la ejecución o los logs de Spark

Los logs de actividad del monitoreo del workspace aportan metadatos útiles sobre estados y errores, pero no sustituyen la salida detallada de las actividades ni los logs del driver/executors de Spark en todas las investigaciones de causa raíz.

El monitoreo de notebooks necesita dos capas

Usa:

```
ItemJobEventLogs
```

para el historial de ejecuciones de notebooks.

Usa:

```
Monitoreo de Spark / Spark UI
```

para los detalles de la ejecución distribuida.

Intentar sacar un análisis de stages y tareas de los logs de jobs del workspace es usar la capa equivocada.

Las consultas de monitoreo deben filtrar correctamente los estados terminales

Los logs del ciclo de vida de los jobs pueden incluir eventos en curso y eventos terminales.

Para reportar SLA o errores, filtra por estados terminales o deduplica por instancia de job cuando sea necesario.

Los umbrales de rendimiento deben ser específicos de cada carga de trabajo

Un pipeline de 30 minutos puede ser:

```
saludable para un procesamiento mensual
```

y:

```
inaceptable para un SLA de 5 minutos
```

Usa líneas base históricas y requisitos de servicio.

Las alertas necesitan lógica de deduplicación

Un diseño de consulta y ventana puede generar alertas repetidas para el mismo error.

Incluye:

- el identificador de ejecución
- una ventana de tiempo reciente
- el estado de la alerta
- el intervalo de sondeo

en el diseño.

ERRORES COMUNES

1. Monitorear solo los errores

Las ejecuciones correctas pero lentas pueden ser señales de alerta temprana.

Sigue las tendencias de duración además del estado.

2. Usar solo el estado a nivel de pipeline

Un pipeline fallido dice muy poco sobre si la causa raíz está en:

- Copy
- Notebook
- SQL
- una API externa
- una escritura Delta

Profundiza hasta el nivel de actividad.

3. Usar los logs de actividad para diagnosticar el skew de Spark

Esa es la capa de telemetría equivocada.

Abre el monitoreo de Spark / Spark UI.

4. Considerar que la duración promedio es suficiente

Los promedios ocultan regresiones intermitentes.

Sigue los percentiles y los valores máximos cuando sea útil.

5. Mezclar ejecuciones manuales y programadas en una misma línea base

Los distintos patrones de invocación pueden procesar volúmenes diferentes o ejecutarse en condiciones diferentes.

Segméntalos cuando sea necesario.

6. Fijar en el código umbrales de alerta sin historial de la carga de trabajo

Usa el SLA o el comportamiento histórico.

No copies un umbral arbitrario de «60 minutos» en todos los pipelines.

7. Registrar todas las métricas posibles

Demasiada telemetría hace que los incidentes sean más difíciles de interpretar.

Elige métricas con significado operativo.

8. Construir un monitoreo a medida antes de usar las vistas nativas de Fabric

Monitoring hub, el monitoreo de Spark, las series de ejecuciones, Workspace Monitoring y Capacity Metrics ya cubren gran parte de la pila operativa.

Amplíalos solo donde sea necesario.

9. Ignorar las métricas de datos de negocio

Un pipeline en verde aun así puede escribir:

```
0 filas
filas duplicadas
claves no válidas
rango de fechas incorrecto
```

Monitorea la exactitud de los datos además de la ejecución técnica.

10. Convertir la tabla de log de ejecución en un punto caliente de concurrencia

Las escrituras de observabilidad también necesitan un diseño de nivel de producción.

Prefiere un registro orientado a inserciones (append) o un patrón centralizado adecuado.

SIGUIENTE PASO

Usa la señal de monitoreo para elegir la siguiente investigación.

- **El pipeline falló** → abre la ejecución de las actividades e identifica la primera actividad que falló.
- **La actividad de notebook falló** → abre los diagnósticos y los logs de la aplicación Spark.
- **El pipeline es más lento de lo normal** → compara la duración de las actividades antes de culpar a Spark.
- **El notebook es más lento de lo normal** → inspecciona las series de ejecuciones, los stages de Spark, el shuffle, el skew y el volumen de entrada.
- **Muchos pipelines fallan a la vez** → investiga las dependencias compartidas y el estado de la capacidad o de la plataforma.
- **La capacidad está saturada** → pasa a Capacity Metrics y al análisis de la programación y la concurrencia de las cargas de trabajo.
- **El error se repite** → crea un análisis KQL a nivel de workspace y una alerta accionable.
- **El pipeline se completa pero la salida es incorrecta** → añade métricas de negocio de ejecución y de calidad de datos.
- **Las comprobaciones manuales se vuelven repetitivas** → crea un KQL Queryset o un dashboard guardado y alerta solo sobre las condiciones que requieran acción.

Para profundizar

- Guía rápida relacionada: Depurar pipelines de Fabric
- Guía rápida relacionada: Rendimiento de los notebooks de Fabric
- Guía rápida relacionada: Session tags en Fabric
- Guía rápida relacionada: Patrones de lectura y escritura en Delta Lake
- Guía rápida relacionada: Lakehouse o Warehouse
- Guía rápida relacionada: Arquitectura Medallion en Fabric
- Guía rápida relacionada: Referencia rápida de rendimiento de PySpark
- Guía rápida relacionada: Leer y escribir tablas Delta
- Guía rápida relacionada: Controles de calidad de datos
- Artículo relacionado: Monitoreo de pipelines y notebooks de Microsoft Fabric