AIDE-MÉMOIRE MICROSOFT FABRIC / SUPERVISION ET OBSERVABILITÉ / INTERMÉDIAIRE

Superviser les pipelines et notebooks Fabric

Supervisez les pipelines et notebooks Microsoft Fabric avec le Monitoring hub, Workspace Monitoring, la supervision Spark, KQL, les alertes et la télémétrie de capacité afin de rendre observables les échecs et les régressions de performances.

Mis à jour le 4 oct. 2026 Version en ligne →

PROBLÈME

Les pipelines et notebooks Fabric peuvent réussir, échouer, ralentir, être relancés, attendre dans une file ou consommer plus de capacité que prévu.

Ouvrir les pages d'exécution une par une suffit pour un incident isolé.

Cela ne suffit pas pour l'exploitation en production.

Vous devez pouvoir répondre à des questions telles que :

- Quels pipelines ont échoué aujourd'hui ?
- Quelle activité a échoué en premier ?
- Quels notebooks ralentissent au fil du temps ?
- Le ralentissement vient-il de l'exécution Spark ou de l'orchestration du pipeline ?
- Les échecs se concentrent-ils sur un notebook ou un type d'activité ?
- Quelles exécutions étaient planifiées et lesquelles étaient à la demande ?
- Les jobs attendent-ils en file à cause d'une pression sur la capacité ?
- Un déploiement a-t-il modifié le comportement à l'exécution ?
- L'équipe doit-elle recevoir une alerte automatiquement ?

L'objectif est de combiner **supervision opérationnelle**, **analyse historique des journaux**, **diagnostic Spark** et **télémétrie de capacité** sans construire une plateforme de supervision sur mesure à partir de zéro.

RÉPONSE RAPIDE

Utilisez la couche de supervision qui correspond à la question :

```
Besoin d'inspecter rapidement une exécution en cours ou récente ?
→ Monitoring hub

Besoin du détail des activités d'un pipeline ?
→ Détail de l'exécution du pipeline / historique des exécutions d'activités

Besoin des stages, tâches, executors et journaux Spark d'un notebook ?
→ Supervision des applications Spark / Spark UI

Besoin d'une analyse historique à l'échelle de l'espace de travail ?
→ Workspace Monitoring + Eventhouse + KQL

Besoin d'alertes ?
→ Alertes de supervision / Activator / workflow d'alerte pris en charge
```

```
Besoin du contexte de consommation de capacité ?
→ Application Fabric Capacity Metrics
```

Pour l'état des pipelines à l'échelle de l'espace de travail :

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

Pour les activités de pipeline en échec :

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

Pour l'historique des jobs de notebook :

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

Utilisez l'interface pour inspecter rapidement un incident.

Utilisez KQL lorsque vous avez besoin de tendances, de comparaisons, d'automatisation et d'une visibilité opérationnelle récurrente.

Code / commandes

1. Utiliser le Monitoring hub pour un tri opérationnel rapide

Ouvrez :

```
Fabric
→ Monitor
```

Utilisez le Monitoring hub lorsque vous devez :

- retrouver les exécutions récentes de pipelines
- retrouver les exécutions de notebooks/Spark
- filtrer par état
- filtrer par type d'élément
- examiner la durée d'exécution
- ouvrir le détail des exécutions
- identifier les relations d'exécution amont/aval
- relancer les exécutions de pipeline qui le permettent

C'est généralement le point de départ le plus rapide pendant un incident.

N'écrivez pas d'abord une requête KQL si une exécution en échec est déjà visible dans le portail.

---

2. Ouvrir le détail de l'exécution du pipeline pour trouver l'activité en échec

Un pipeline peut contenir de nombreuses activités, mais une seule est peut-être à l'origine de l'échec.

Utilisez :

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

Relevez :

```
nom du pipeline
ID d'exécution
nom de l'activité
type d'activité
statut
heure de début
heure de fin
durée
type d'échec
```

Remontez jusqu'au premier véritable échec plutôt que de vous arrêter sur les activités ignorées en aval.

---

3. Utiliser la supervision contextuelle du notebook pendant le développement

Dans un notebook Fabric, la supervision Spark contextuelle peut afficher la progression de l'exécution au niveau du notebook ou de la cellule.

Utilisez-la pour examiner :

- les jobs Spark déclenchés par une cellule
- les stages
- les tâches
- les executors
- les journaux
- les erreurs d'exécution
- les recommandations de Spark advisor

C'est utile lorsque :

```
le notebook est lent
```

mais que vous devez encore déterminer si la lenteur de la cellule est due à :

- un scan
- un shuffle
- un skew
- une défaillance d'executor
- une pression mémoire
- la configuration Spark

---

4. Utiliser la supervision des applications Spark pour les performances des notebooks

Ouvrez la page de détail de l'application Spark depuis :

```
Monitoring hub
```

ou :

```
Notebook
→ Recent runs
```

Les zones importantes sont notamment :

```
Jobs
Resources
Logs
Diagnostics
```

La vue Jobs peut afficher :

- la durée du job
- les stages
- les tâches
- les données lues
- les données écrites
- l'extrait de code

La vue Resources aide à corréler le job Spark avec l'utilisation des executors.

La vue Logs fournit :

- les journaux du driver
- les journaux Livy
- les journaux de prélancement (prelaunch)

Utilisez la Spark UI lorsque vous avez besoin d'un niveau de détail plus fin sur les stages et les tâches.

5. Utiliser la Spark UI pour un diagnostic au niveau des stages

La Spark UI est le bon outil lorsque la question devient :

```
Pourquoi cette application Spark a-t-elle pris 25 minutes ?
```

Examinez :

```
Jobs
Stages
Storage
Environment
Executors
```

Exemples :

```
Une tâche beaucoup plus lente que les autres
→ skew possible

Lecture/écriture de shuffle volumineuse
→ redistribution due à une jointure / agrégation

Échecs d'executors
→ problème de mémoire/ressources/runtime

Jeux de données en cache volumineux
→ pression sur le stockage/cache

Configuration Spark inattendue
→ dérive de l'environnement/de la configuration
```

Ne vous fiez pas uniquement à la durée totale (wall-clock) du notebook.

6. Activer Workspace Monitoring pour l'analyse historique

Workspace Monitoring crée un Eventhouse de supervision et une base de données KQL de supervision en lecture seule pour les éléments Fabric pris en charge.

Activez-le depuis :

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

Utilisez Workspace Monitoring lorsque vous avez besoin :

- des tendances historiques d'exécution
- des échecs à l'échelle de l'espace de travail
- d'une analyse récurrente des performances
- de requêtes KQL personnalisées
- de tableaux de bord
- d'une logique d'alerte
- de rapports opérationnels

C'est le changement clé, qui consiste à passer de :

```
inspecter une exécution
```

à :

```
exploiter la plateforme
```

7. Comprendre les deux principales tables de supervision des pipelines

Supervision au niveau du pipeline :

```
ItemJobEventLogs
```

Supervision des pipelines au niveau des activités :

```
FabricDataPipelinesActivityRunsLogs
```

Utilisez :

```
ItemJobEventLogs
```

pour des questions telles que :

- combien de pipelines ont échoué ?
- quel pipeline ralentit ?
- quels jobs sont planifiés et lesquels sont à la demande ?
- qui a déclenché l'exécution ?

Utilisez :

```
FabricDataPipelinesActivityRunsLogs
```

pour des questions telles que :

- quelle activité a échoué ?
- quel type d'activité est lent ?
- où le temps du pipeline s'accumule-t-il ?
- combien de CPU une activité a-t-elle consommé, lorsque l'information est disponible ?

8. Examiner le schéma de supervision avant d'écrire du KQL de production

Commencez par :

```kql
ItemJobEventLogs
| getschema
```

et :

```kql
FabricDataPipelinesActivityRunsLogs
| getschema
```

Faites-le, car les schémas de supervision peuvent évoluer.

Ne supposez pas que du code copié depuis un ancien article ou un autre tenant correspond exactement à la table actuelle.

---

9. Compter les exécutions de pipeline par état

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

Vous obtenez ainsi un résumé rapide de l'état de santé.

Soyez prudent lorsque le modèle de journalisation contient plusieurs événements de cycle de vie pour la même instance de job.

Pour les rapports sur les exécutions terminées, filtrez sur les états terminaux.

---

10. Compter uniquement les résultats terminaux des pipelines

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

Utilisez les états terminaux pour les rapports de réussite/échec.

Ne comptez pas :

```
En cours
Non démarré
```

comme des exécutions terminées.

---

11. Afficher les pipelines récemment en échec

```kql
ItemJobEventLogs
| where ItemKind == "Data Pipeline"
| where JobStatus == "Failed"
| project
    Timestamp,
    WorkspaceName,
    ItemName,
    JobInstanceId,
```

```
    JobInvokeType,
    JobStartTime,
    JobEndTime,
    DurationMs,
    ExecutingPrincipalType
| order by Timestamp desc
```

Utilisez le `JobInstanceId` pour corréler une exécution précise avec le portail ou avec une investigation plus poussée au niveau de l'exécution.

12. Afficher les activités de pipeline récemment en échec

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

C'est bien plus exploitable qu'un nombre d'échecs au niveau du pipeline lorsque ce même pipeline contient de nombreuses activités.

13. Trouver les activités qui échouent le plus souvent

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

Utilisez cette requête pour prioriser les problèmes structurels de fiabilité.

Une activité qui échoue de manière répétée mérite plus d'attention qu'un échec transitoire isolé.

14. Résumer la durée des activités 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
```

Utilisez une analyse fondée sur les percentiles pour suivre les tendances de performances.

La moyenne seule peut masquer des exécutions lentes intermittentes.

N'interprétez pas le P95 sans un historique suffisant et représentatif.

15. Trouver les régressions de durée des activités de pipeline

Une comparaison simple entre la période récente et la période précédente :

```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
```

Il s'agit d'une comparaison de tri, pas d'un verdict automatique sur les performances.

Vérifiez :

- le volume d'entrée
- la latence de la source
- les modifications du code
- la pression sur la capacité

avant de qualifier une exécution de régression.

16. Trouver les activités de pipeline récentes les plus lentes

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

Utilisez cette requête pour repérer rapidement les valeurs aberrantes.

Comparez ensuite chaque activité à sa propre référence habituelle.

17. Trouver les échecs d'activités de pipeline par type

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

Utilisez cette requête pour distinguer des tendances telles que :

```
Échecs de notebooks
Échecs de copie
Échecs liés à la passerelle
Autres catégories d'échecs récurrents
```

Ne considérez pas `FailureType` comme un message complet de cause racine.

Les journaux d'activité actuels de l'espace de travail ne contiennent pas forcément toutes les chaînes d'erreur détaillées disponibles dans l'interface d'exécution.

18. Afficher la consommation CPU des activités de pipeline lorsqu'elle est disponible

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

Utilisez-la comme un signal opérationnel parmi d'autres.

Ne comparez pas les métriques CPU entre des types d'activités fondamentalement différents sans contexte.

19. Trouver les exécutions de notebooks depuis la supervision de l'espace de travail

`ItemJobEventLogs` inclut les types de jobs Notebook pris en charge.

Exemple :

```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
```

Utilisez cette requête pour obtenir une visibilité historique au niveau des notebooks.

Pour le détail des stages et des tâches Spark, passez à la supervision Spark au lieu d'attendre des journaux de jobs de l'espace de travail qu'ils contiennent des métriques Spark de bas niveau.

20. Trouver les exécutions de notebooks en échec

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
```

Utilisez cette requête pour identifier les notebooks qui échouent de manière répétée.

Ouvrez ensuite l'application Spark correspondante pour un véritable diagnostic Spark.

21. Suivre les tendances de durée des 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
```

C'est utile pour les charges de travail de notebooks récurrentes.

Séparez les exécutions interactives de développement lorsqu'elles ne sont pas comparables aux exécutions planifiées de production.

22. Ventiler les exécutions de notebooks par type d'invocation

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

Cela permet d'éviter de mélanger :

```
développement interactif manuel
```

avec :

```
exécution planifiée en production
```

dans une même référence de performances.

23. Suivre le comportement des pipelines planifiés et à la demande

```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
```

Cela peut révéler qu'un pipeline se comporte différemment lorsqu'il est :

- planifié
- exécuté manuellement
- déclenché dans une autre fenêtre de charge

24. Superviser un pipeline précis

```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
```

Utilisez des requêtes propres à un pipeline lorsque vous diagnostiquez une charge de travail connue.

25. Superviser les activités d'un pipeline

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

Filtrez ensuite par `PipelineRunId` pour reconstituer une exécution.

26. Reconstituer une exécution de pipeline

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

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

Vous obtenez ainsi une vue chronologique de l'exécution des activités.

Utile pour identifier :

- où le temps s'est accumulé
- quelle activité a échoué en premier
- si des activités se sont chevauchées

27. Identifier les jobs de longue durée avec un seuil propre à la charge de travail

Exemple :

```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
```

Le seuil de 60 minutes n'est ici qu'un exemple.

Définissez les seuils à partir :

- du SLA
- de la planification attendue
- de la référence historique
- de la fenêtre métier

N'inventez pas un seuil universel de « pipeline lent ».

28. Détecter les échecs de pipeline dans une fenêtre d'alerte récente

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

Ce type de requête peut alimenter un workflow d'alerte.

Alignez l'intervalle d'interrogation de l'alerte et la fenêtre temporelle de la requête afin que les échecs ne soient ni manqués ni signalés plusieurs fois inutilement.

29. Créer des alertes uniquement pour des conditions actionnables

Exemples de bonnes alertes :

```
échec d'un pipeline de production
échec d'un notebook critique
pipeline ayant dépassé son SLA
échecs répétés sur une courte période
exécution planifiée non terminée dans la fenêtre prévue
```

Exemples de mauvaises alertes :

```
chaque démarrage de notebook
chaque nouvelle tentative
chaque avertissement
chaque événement transitoire qui se résout de lui-même
```

La lassitude face aux alertes réduit leur valeur opérationnelle.

30. Utiliser les identifiants d'instance de job pour la corrélation

`JobInstanceId` identifie une instance de job précise d'un élément Fabric.

Utilisez-le pour corréler :

```
journal de supervision
exécution dans le portail
alerte
incident
journal d'exécution
```

Pour les pipelines, le `PipelineRunId` au niveau des activités est le champ de corrélation d'exécution important dans :

```
FabricDataPipelinesActivityRunsLogs
```

Ne corrélez pas les incidents de production uniquement par nom d'élément et horodatage.

31. Ajouter votre propre identifiant d'exécution métier

Les identifiants de job Fabric sont des identifiants opérationnels.

Votre pipeline peut aussi avoir besoin d'un identifiant de niveau métier :

```
batch_id
business_date
source_file_id
logical_run_id
```

Exemple dans 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(),
    )
)
```

Vous pouvez ainsi corréler l'exécution sur la plateforme avec les données réellement écrites.

32. Conserver les métriques opérationnelles depuis les notebooks

Parmi les métriques utiles :

```
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
```

Exemple :

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

En production, écrivez des métriques structurées plutôt que de vous contenter de dictionnaires affichés avec print.

Soyez attentif aux actions coûteuses, comme les appels répétés à `count()`.

---

33. Journaliser l'état d'exécution des pipelines en mode ajout

Une table logique simple :

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

Privilégiez autant que possible des enregistrements d'événements en ajout seul (append-only).

Cela réduit la contention par rapport à de nombreux notebooks concurrents qui mettent à jour la même ligne de journal de façon répétée.

---

34. Éviter que la table de supervision devienne un goulet d'étranglement

Si de nombreux notebooks écrivent simultanément dans une même table de journal Delta, cette table peut devenir une cible d'écriture très sollicitée.

Envisagez :

```
journalisation d'événements en ajout seul
événements opérationnels partitionnés
une ligne par transition d'activité
rapprochement centralisé
```

Ne créez pas une solution de supervision qui provoque plus d'échecs de pipeline qu'elle n'en observe.

35. Utiliser la supervision des séries d'exécutions Fabric pour les charges Spark récurrentes

Fabric peut regrouper les applications Spark récurrentes en séries d'exécutions (run series).

Utilisez les vues de séries d'exécutions pour comparer :

- la tendance de durée
- la tendance des données en entrée/sortie
- les exécutions anormales

C'est utile pour les éléments récurrents suivants :

- notebooks
- définitions de jobs Spark
- activités notebook lancées par des pipelines

Utilisez cette fonctionnalité avant de construire une vue de tendance personnalisée lorsque la comparaison intégrée répond déjà à la question.

36. Comparer les exécutions d'applications Spark lorsqu'un notebook régresse

La comparaison d'applications Spark de Fabric permet de comparer plusieurs exécutions et de mettre en évidence les différences de performances.

Utilisez-la lorsque :

```
Le notebook prenait 8 minutes la semaine dernière
Le notebook prend désormais 25 minutes
```

Comparez :

- la durée
- les entrées/sorties
- le comportement d'exécution Spark
- les stages anormaux

Examinez ensuite les modifications du code, de la configuration ou des données.

37. Utiliser les API de supervision Spark pour l'automatisation lorsque l'interface ne suffit pas

Fabric expose des API de supervision Spark pour récupérer par programmation les informations sur les applications Spark.

Utilisez les API lorsque vous devez :

- collecter automatiquement les métadonnées des exécutions Spark des notebooks
- intégrer la supervision dans un autre système opérationnel
- construire un tableau de bord interne personnalisé
- corréler les identifiants d'applications Spark avec des systèmes externes de gestion des incidents

Privilégiez d'abord le Monitoring hub et Workspace Monitoring intégrés, sauf si l'automatisation par API apporte une réelle valeur.

---

38. Utiliser Capacity Metrics lorsque le problème peut venir d'une pression sur la capacité

La lenteur d'un pipeline ou d'un notebook peut provenir :

```
saturation de la capacité
charges de travail concurrentes
contention des ressources
throttling (limitation)
```

Utilisez l'application Microsoft Fabric Capacity Metrics pour examiner la consommation au niveau de la capacité.

C'est particulièrement important lorsque :

```
de nombreuses charges de travail sans lien ralentissent en même temps
```

Ce schéma est différent de celui d'un seul notebook inefficace.

---

39. Corréler la capacité de la plateforme avec le calendrier des charges de travail

Lorsqu'un pipeline ralentit tous les jours à 09:00, comparez :

```
durée du pipeline
durée Spark du notebook
utilisation de la capacité
autres jobs exécutés au même moment
```

N'optimisez pas le code du notebook de façon isolée si le véritable goulet d'étranglement est la pression sur une capacité partagée.

---

40. Suivre un petit ensemble de métriques de service opérationnelles

Une référence de production utile est la suivante :

```
Fiabilité
- nombre total d'exécutions
- taux de réussite
- nombre d'échecs
- nombre d'échecs répétés

Performances
- durée moyenne
- durée P95
- durée maximale
- temps d'attente/de démarrage si disponible

Qualité des données
- lignes lues/écrites
```

```
- lignes rejetées
- clés en double
- clés obligatoires nulles

Ressources
- métriques Spark de stage/runtime
- utilisation de la capacité
- CPU des activités si disponible
```

Ne collectez pas des métriques uniquement parce qu'elles sont disponibles.

Chaque métrique doit étayer une décision opérationnelle.

Interpréter le résultat

Surface de supervision	Meilleur usage
Monitoring hub	Vue opérationnelle rapide des exécutions en cours ou récentes
Détail de l'exécution du pipeline	Diagnostic des pipelines au niveau des activités
Supervision contextuelle du notebook	Exécution des cellules/de Spark pendant le développement
Détail de l'application Spark	Analyse des jobs, des ressources, des journaux et des diagnostics
Spark UI	Diagnostic approfondi des stages, tâches et executors
Workspace Monitoring	Télémétrie historique à l'échelle de l'espace de travail
`ItemJobEventLogs`	Historique d'exécution Fabric au niveau des éléments/jobs
`FabricDataPipelinesActivityRunsLogs`	Historique des pipelines au niveau des activités
Séries d'exécutions / comparaison d'exécutions	Tendances de performances Spark récurrentes
Capacity Metrics	Pression sur la capacité et consommation des charges de travail
Alertes / Activator	Notifier lorsqu'une condition opérationnelle survient

La distinction essentielle est la suivante :

```
Statut du pipeline
!=
cause racine de l'activité
!=
cause racine Spark
!=
cause racine liée à la capacité
```

Ne descendez dans la pile de supervision qu'aussi loin que l'incident l'exige.

Points à vérifier

Le taux d'échec augmente sur de nombreux pipelines sans lien entre eux

Examinez :

- une dépendance de source partagée
- les informations d'identification
- la passerelle (gateway)
- un problème d'espace de travail ou de capacité
- un incident de plateforme ou de service

Ne commencez pas par déboguer chaque pipeline séparément.

Une activité échoue de manière répétée

Il s'agit probablement d'un problème structurel.

Examinez :

- les paramètres de l'activité
- le contrat de données
- le comportement des nouvelles tentatives
- l'identité
- la dépendance de source ou de destination (sink)

La durée du pipeline augmente, mais pas la durée Spark

Examinez :

- l'attente en file
- les temps morts d'orchestration
- les activités externes
- le démarrage de session
- les attentes côté source ou destination

Le notebook n'est peut-être pas le goulet d'étranglement.

La durée du notebook augmente avec un volume d'entrée plus élevé

Il peut s'agir d'un comportement de montée en charge attendu.

Normalisez les comparaisons en fonction :

- du nombre de lignes en entrée
- des octets traités
- du nombre de fichiers
- du nombre de partitions

avant de parler de régression.

Une seule tâche Spark domine un stage

Examinez le skew.

La durée au niveau de l'espace de travail ne permet pas à elle seule de le diagnostiquer.

Plusieurs charges de travail ralentissent en même temps

Vérifiez la pression sur la capacité avant de réécrire le code de chacune.

Les échecs ne se produisent que lors des exécutions planifiées

Comparez :

- l'identité
- la fenêtre horaire d'exécution
- la disponibilité de la source
- le chevauchement des planifications
- le type d'invocation
- la charge de la capacité

Les exécutions planifiées et manuelles ne sont pas toujours équivalentes sur le plan opérationnel.

> **NOTES DE PRODUCTION**
>
> ### Workspace Monitoring est une fonctionnalité d'observabilité de production, mais la connaissance du schéma reste importante
>
> L'Eventhouse de supervision vous fournit des journaux durables, interrogeables en KQL.
>
> Examinez toujours le schéma actuel avant de standardiser des tableaux de bord ou une logique d'alerte.
>
> Les tables de supervision et leur couverture peuvent évoluer.
>
> ### Workspace Monitoring a un coût en capacité
>
> L'Eventhouse de supervision et les composants de supervision associés consomment de la capacité Fabric.
>
> L'observabilité n'est pas gratuite.
>
> Alignez les tableaux de bord, les requêtes et la rétention sur la valeur opérationnelle réelle.
>
> ### Workspace Monitoring est en lecture seule pour les journaux collectés
>
> Utilisez la base de données de supervision pour l'analyse.
>
> Ne la concevez pas comme la base de données de contrôle modifiable de votre application.
>
> ### Les journaux de jobs des éléments et les journaux d'activités n'ont pas la même granularité
>
> `ItemJobEventLogs` offre la vue générale au niveau des jobs.
>
> `FabricDataPipelinesActivityRunsLogs` offre la vue détaillée des activités de pipeline.
>
> Utilisez la granularité la plus fine nécessaire pour répondre à la question.
>
> ### Les messages d'erreur détaillés peuvent encore nécessiter l'interface d'exécution ou les journaux Spark
>
> Les journaux d'activités de la supervision de l'espace de travail fournissent des métadonnées utiles sur les états et les échecs, mais ils ne remplacent pas la sortie détaillée des activités ni les journaux du driver/des executors Spark pour chaque recherche de cause racine.
>
> ### La supervision des notebooks nécessite deux couches
>
> Utilisez :
>
> ```
> ItemJobEventLogs
> ```
>
> pour l'historique des exécutions de notebooks.

Utilisez :

```
Supervision Spark / Spark UI
```

pour les détails de l'exécution distribuée.

Chercher à extraire une analyse des stages et des tâches à partir des journaux de jobs de l'espace de travail revient à se tromper de couche.

Les requêtes de supervision doivent filtrer correctement les états terminaux

Les journaux du cycle de vie des jobs peuvent contenir des événements en cours et des événements terminaux.

Pour les rapports de SLA ou d'échecs, filtrez sur les états terminaux ou dédupliquez par instance de job si nécessaire.

Les seuils de performances doivent être propres à chaque charge de travail

Un pipeline de 30 minutes peut être :

```
sain pour un traitement mensuel
```

et :

```
inacceptable pour un SLA de 5 minutes
```

Appuyez-vous sur des références historiques et sur les exigences de service.

Les alertes nécessitent une logique de déduplication

Une conception de requête et de fenêtre peut générer des alertes répétées pour le même échec.

Intégrez :

- l'identifiant d'exécution
- une fenêtre temporelle récente
- l'état de l'alerte
- l'intervalle d'interrogation

dans la conception.

ERREURS FRÉQUENTES

1. Ne superviser que les échecs

Des exécutions réussies mais lentes peuvent être des signaux d'alerte précoces.

Suivez les tendances de durée en plus des états.

2. Se limiter à l'état au niveau du pipeline

Un pipeline en échec vous apprend très peu de chose sur l'origine de la cause racine :

- Copy
- Notebook
- SQL
- API externe
- écriture Delta

Descendez au niveau des activités.

3. Utiliser les journaux d'activités pour diagnostiquer le skew Spark

Ce n'est pas la bonne couche de télémétrie.

Ouvrez la supervision Spark / la Spark UI.

4. Considérer que la durée moyenne suffit

Les moyennes masquent les régressions intermittentes.

Suivez les percentiles et les valeurs maximales lorsque c'est utile.

5. Mélanger exécutions manuelles et planifiées dans une même référence

Des modes d'invocation différents peuvent traiter des volumes différents ou s'exécuter dans des conditions différentes.

Segmentez-les si nécessaire.

6. Coder en dur des seuils d'alerte sans historique de la charge de travail

Appuyez-vous sur le SLA ou sur le comportement historique.

Ne copiez pas un seuil arbitraire de « 60 minutes » dans chaque pipeline.

7. Journaliser toutes les métriques possibles

Trop de télémétrie rend les incidents plus difficiles à interpréter.

Choisissez des métriques pertinentes sur le plan opérationnel.

8. Construire une supervision sur mesure avant d'utiliser les vues natives de Fabric

Le Monitoring hub, la supervision Spark, les séries d'exécutions, Workspace Monitoring et Capacity Metrics couvrent déjà une grande partie de la pile opérationnelle.

N'étendez que là où c'est nécessaire.

9. Ignorer les métriques des données métier

Un pipeline au vert peut quand même écrire :

```
0 ligne
lignes dupliquées
clés invalides
plage de dates incorrecte
```

Supervisez l'exactitude des données autant que l'exécution technique.

10. Faire de la table de journal d'exécution un point chaud de concurrence

Les écritures d'observabilité nécessitent elles aussi une conception de niveau production.

Privilégiez une journalisation orientée ajout ou un modèle centralisé adapté.

ÉTAPE SUIVANTE

Utilisez le signal de supervision pour choisir la prochaine investigation.

- **Le pipeline a échoué** → ouvrez l'exécution des activités et identifiez la première activité en échec.
- **L'activité notebook a échoué** → ouvrez les diagnostics et les journaux de l'application Spark.
- **Le pipeline est plus lent que d'habitude** → comparez la durée des activités avant d'incriminer Spark.
- **Le notebook est plus lent que d'habitude** → examinez les séries d'exécutions, les stages Spark, le shuffle, le skew et le volume d'entrée.

- **De nombreux pipelines échouent en même temps** → examinez les dépendances partagées et l'état de la capacité ou de la plateforme.
- **La capacité est saturée** → passez à Capacity Metrics et à l'analyse de la planification et de la concurrence des charges de travail.
- **L'échec se répète** → mettez en place une analyse KQL au niveau de l'espace de travail et une alerte actionnable.
- **Le pipeline réussit mais la sortie est incorrecte** → ajoutez des métriques métier d'exécution et de qualité des données.
- **Les vérifications manuelles deviennent répétitives** → créez un KQL Queryset ou un tableau de bord enregistré et n'alertez que sur les conditions qui exigent une action.

Pour aller plus loin

- Aide-mémoire associé : Déboguer les pipelines Fabric
- Aide-mémoire associé : Performance des notebooks Fabric
- Aide-mémoire associé : Session tags dans Fabric
- Aide-mémoire associé : Modèles de lecture et d'écriture Delta Lake
- Aide-mémoire associé : Lakehouse ou Warehouse
- Aide-mémoire associé : Architecture Medallion dans Fabric
- Aide-mémoire associé : PySpark : référence rapide des performances
- Aide-mémoire associé : Lire et écrire des tables Delta
- Aide-mémoire associé : Contrôles de qualité des données
- Article associé : Superviser les pipelines et notebooks Microsoft Fabric