Apache 星火工作效能調整 AWS Glue 的最佳作法

: Apache 星火工作效能調整 AWS Glue 的最佳作法

Table of Contents

簡介 1
關鍵主題 2
架構 2
彈性分散式資料 3
懶惰評價 4
星火應用術語 5
平行處理 6
催化劑優化 7
調查效能問題 10
使用星火 UI 識別瓶頸 10
調整效能的策略 12
效能調校的基線策略 12
Spark 工作績效的調整實務 12
擴充叢集容量 13
CloudWatch 度量 13
Spark UI 14
使用最新版本 15
減少資料掃描量 16
CloudWatch 度量 16
Spark UI 17
平行化工作 25
CloudWatch 度量 25
Spark UI 26
優化洗牌 30
CloudWatch 度量 31
Spark UI 32
最大限度地減少 39
CloudWatch 度量 39
Spark UI 39
優化用戶定義函 40
標準 Python 42
向量化的 UDF 42
Spark SQL 43
使用熊貓進行大數據 44

資源 45
文件歷史紀錄 46
詞彙表 47
..... 47
A 47
B 50
C 51
D 54
E 57
F 59
G 60
H 61
I 62
L 64
M 64
O 68
P 70
Q 72
R 72
S 75
T 78
U 79
V 79
W 80
Z 81
..... lxxxii

Apache 星火工作效能調整 AWS Glue 的最佳作法

羅馬·邁爾斯，御倉隆和關山則高，Amazon Web Services（）AWS

2023 年 12 月 (文件歷史記錄)

AWS Glue 提供不同的選項來調整效能。本指南定義了調整 Apache 星火 AWS Glue 的重要主題。然後，它會提供一個基準策略供您在調整 Apache Spark 工作時遵循這些 AWS Glue 策略。您可以使用本指南，瞭解如何藉由解譯中 AWS Glue的可用測量結果來識別效能問題。然後納入解決這些問題的策略，最大限度地提高性能並最小化成本。

本指南涵蓋以下調整做法：

- 擴充叢集容量
- 使用最新 AWS Glue 版本
- 減少資料掃描量
- 平行化工作
- 最大限度地減少
- 優化洗牌
- 優化用戶定義函

在阿帕奇星火關鍵主題

本節說明 Apache 星火的基本概念和關鍵主題，以及調整 AWS Glue Apache 星火效能的重要主題。在討論真實世界的調整策略之前，了解這些概念和主題非常重要。

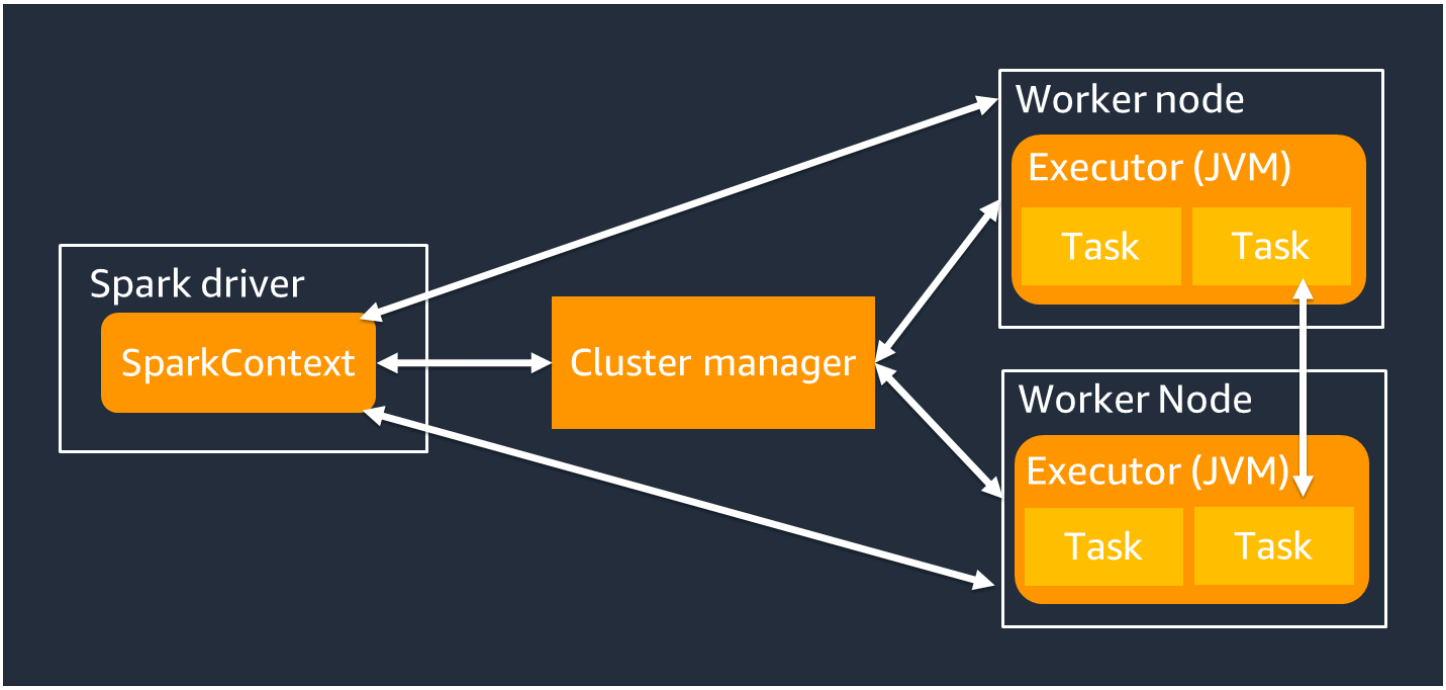
架構

Spark 驅動程序主要負責將您的 Spark 應用程序拆分為可以在個人工作人員上完成的任務。星火驅動程序具有以下職責：

- `main()`在您的代碼中運行
- 產生執行計畫
- 配合叢集管理員來佈建 Spark 執行程式，以管理叢集上的資源
- 為 Spark 執行程序調度任務和請求任務
- 管理任務進度和復原

您可以使用`SparkContext`物件與 Spark 驅動程式進行工作執行互動。

星火執行程序是用於保存數據和運行從 Spark 驅動程序傳遞的任務的工作。Spark 執行程序的數量將隨集群的大小而上下。

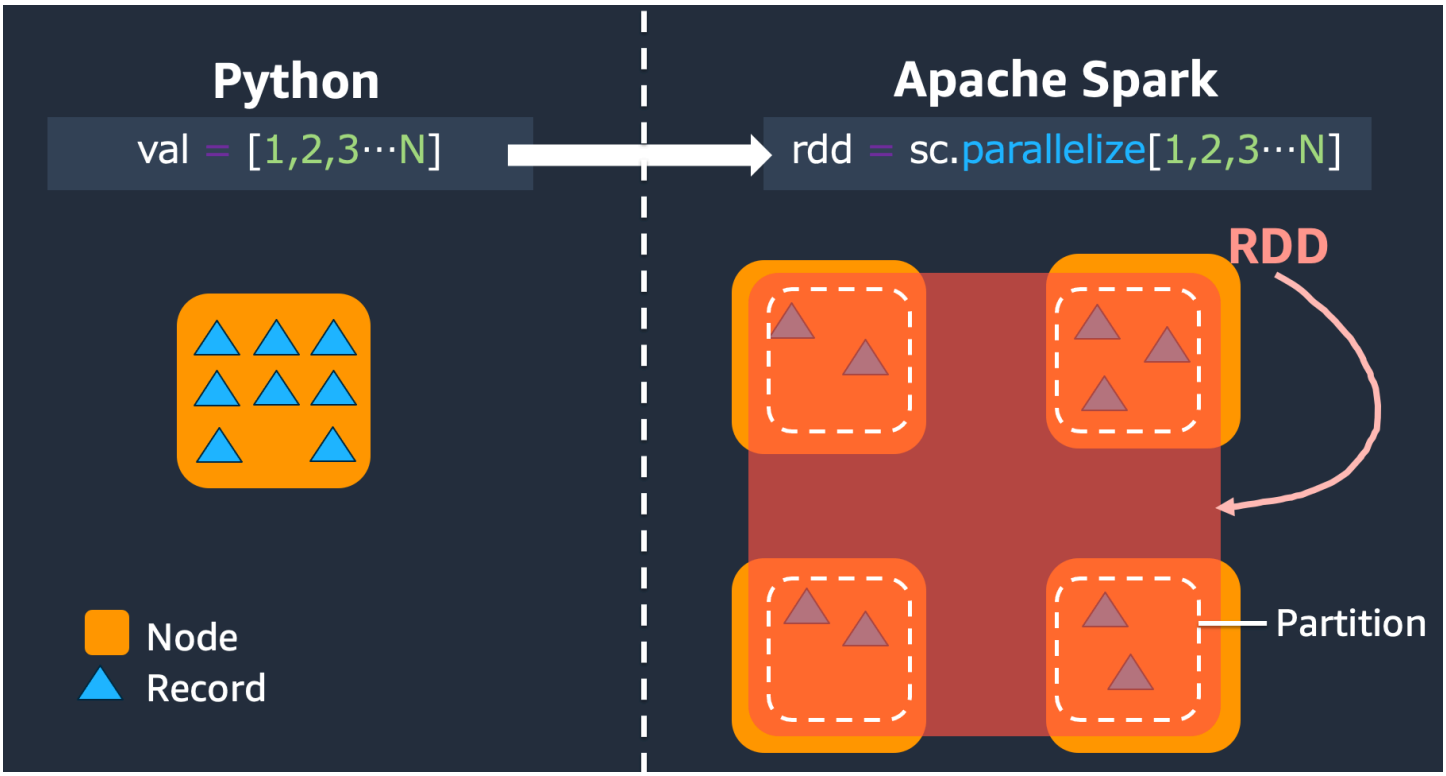


> **Note**
>
> 星火執行程序具有多個插槽，以便多個任務 parallel 處理。Spark 預設為每個虛擬 CPU (vCPU) 核心支援一項工作。例如，如果執行程序有四個 CPU 內核，它可以運行四個並發任務。

彈性分散式資料

Spark 不存儲和跟踪跨 Spark 執行程序大型數據集的複雜工作。當您撰寫 Spark 工作的程式碼時，不需要考慮儲存的詳細資料。星火提供了彈性的分佈式數據集（RDD）抽象，這是可以在並行操作，並且可以跨集群的 Spark 執行器進行分區元素的集合。

下圖顯示了當 Python 腳本在其典型環境中運行以及在 Spark 框架（PySpark）中運行時如何將數據存儲在內存中的差異。



- Python — 在 Python 腳本`val = [1,2,3...N]`中編寫將數據保留在運行代碼的單個機器上的內存中。

- PySpark— Spark 提供 RDD 資料結構，以載入和處理分佈在多個 Spark 執行程式記憶體中的資料。您可以使用代碼生成 RDD`rdd = sc.parallelize[1,2,3...N]`，Spark 可以在多個 Spark 執行程序中自動分發和保存內存中的數據。

  在許多 AWS Glue 工作中，您使用 RDD 通過 AWS Glue DynamicFrames和火花。DataFrames這些是抽象，允許您在 RDD 中定義數據的模式，並使用該附加信息執行更高級別的任務。因為它們在內部使用 RDD，因此資料會透明地分散並載入到下列程式碼中的多個節點：

  - DynamicFrame

    ```
    dyf= glueContext.create_dynamic_frame.from_options(
        's3', {"paths": [ "s3://<YourBucket>/<Prefix>/"]},
        format="parquet",
        transformation_ctx="dyf"
    )
    ```

  - DataFrame

    ```
    df = spark.read.format("parquet")
        .load("s3://<YourBucket>/<Prefix>")
    ```

一個 RDD 具有以下特點：

- RDD 由分成多個部分的數據組成，稱為分區。每個 Spark 執行程序存儲在內存中的一個或多個分區，並且數據分佈在多個執行程序。
- RDD 是不可變的，這意味著它們在創建之後不能更改。若要變更 DataFrame，您可以使用轉換，這些變形在下一節中定義。
- RDD 會跨可用節點複寫資料，以便自動從節點故障中復原。

懶惰評價

RDD 支援兩種類型的作業：轉換 (從現有資料集建立新資料集)，以及動作 (在資料集上執行計算後，將值傳回給驅動程式的驅動程式)。

- 轉換 — 由於 RDD 是不可變的，因此您只能使用轉換來變更它們。

  例如，`map`是一種轉換，會透過函數傳遞每個資料集元素，並傳回代表結果的新 RDD。請注意，該`map`方法不返回輸出。Spark 存儲 future 的抽象轉換，而不是讓你與結果進行交互。火花不會對轉換採取行動，直到你調用一個動作。

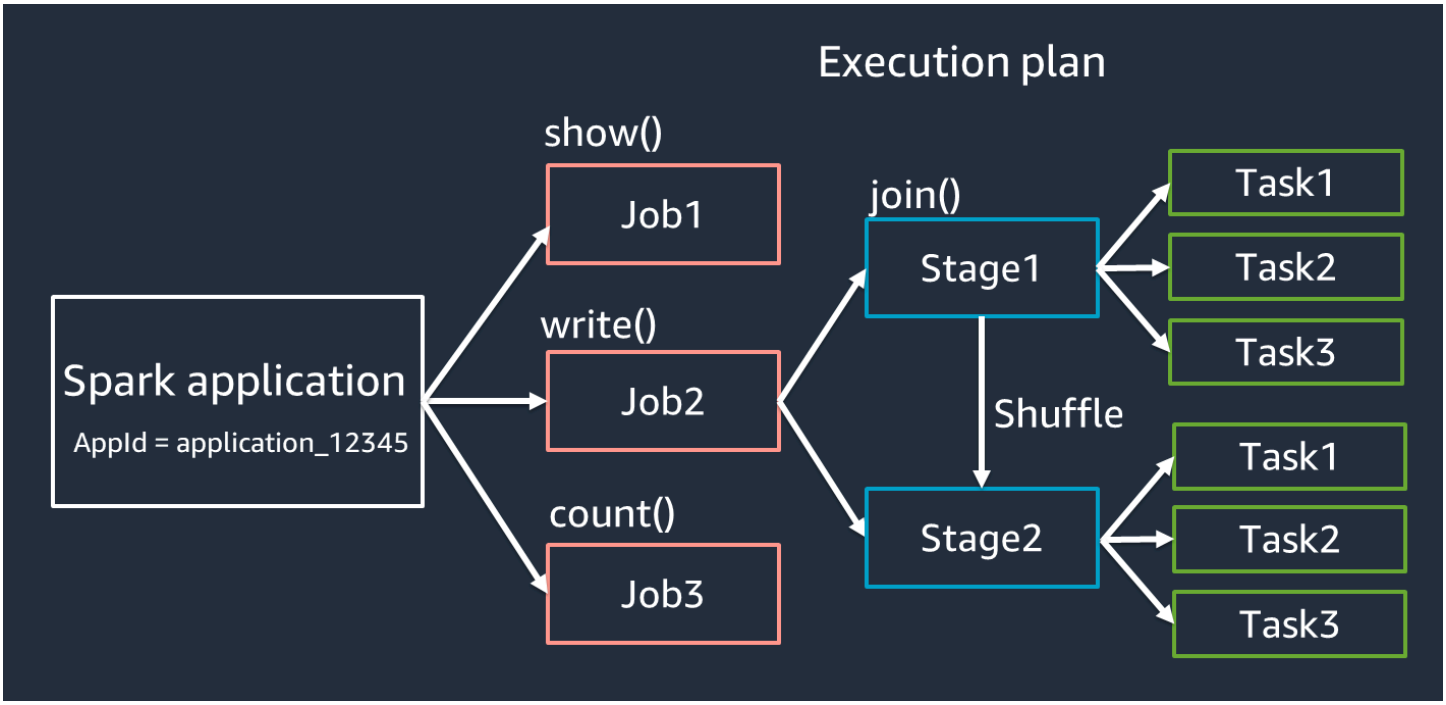
- 動作 — 使用轉換，您可以建立邏輯轉換計劃。若要啟動計算，請執行諸如`write`、`countshow`、或之類的動作`collect`。

  Spark 中的所有轉換都是懶惰的，因為它們不會立即計算結果。相反地，Spark 會記住套用至某些基本資料集的一系列轉換，例如亞馬遜簡單儲存服務 (Amazon S3) 物件。只有當動作需要將結果傳回給驅動程式時，才會計算轉換。這種設計使 Spark 能夠更有效地運行。例如，假設透過`map`轉換建立的資料集只會被大幅減少資料列數目的轉換所使用的情況，例如`reduce`。然後，您可以將經過兩種轉換的較小數據集傳遞給驅動程序，而不是傳遞較大的映射數據集。

星火應用術語

本節涵蓋 Spark 應用程式術語。Spark 驅動程序創建一個執行計劃，並在幾個抽象控制應用程序的行為。下列術語對於使用 Spark UI 進行開發、偵錯和效能調整非常重要。

- 應用程式 — 以 Spark 工作階段 (Spark 內容) 為基礎。由唯一的 ID 識別，例如`<application_XXX>`。
- 工作 — 根據針對 RDD 建立的動作。工作由一個或多個階段組成。
- 階段 — 根據為 RDD 建立的隨機播放。階段由一個或多個任務組成。隨機播放是 Spark 的機制，用於重新分配數據，以便它在 RDD 分區之間以不同的方式進行分組。某些轉換，例如`join()`，需要隨機播放。隨機播放會在「最佳化隨機播放」調整實務中更詳細地討論。
- 任務-任務是 Spark 安排的最小處理單位。任務是為每個 RDD 分區創建的，任務數是階段中同時執行的最大數量。

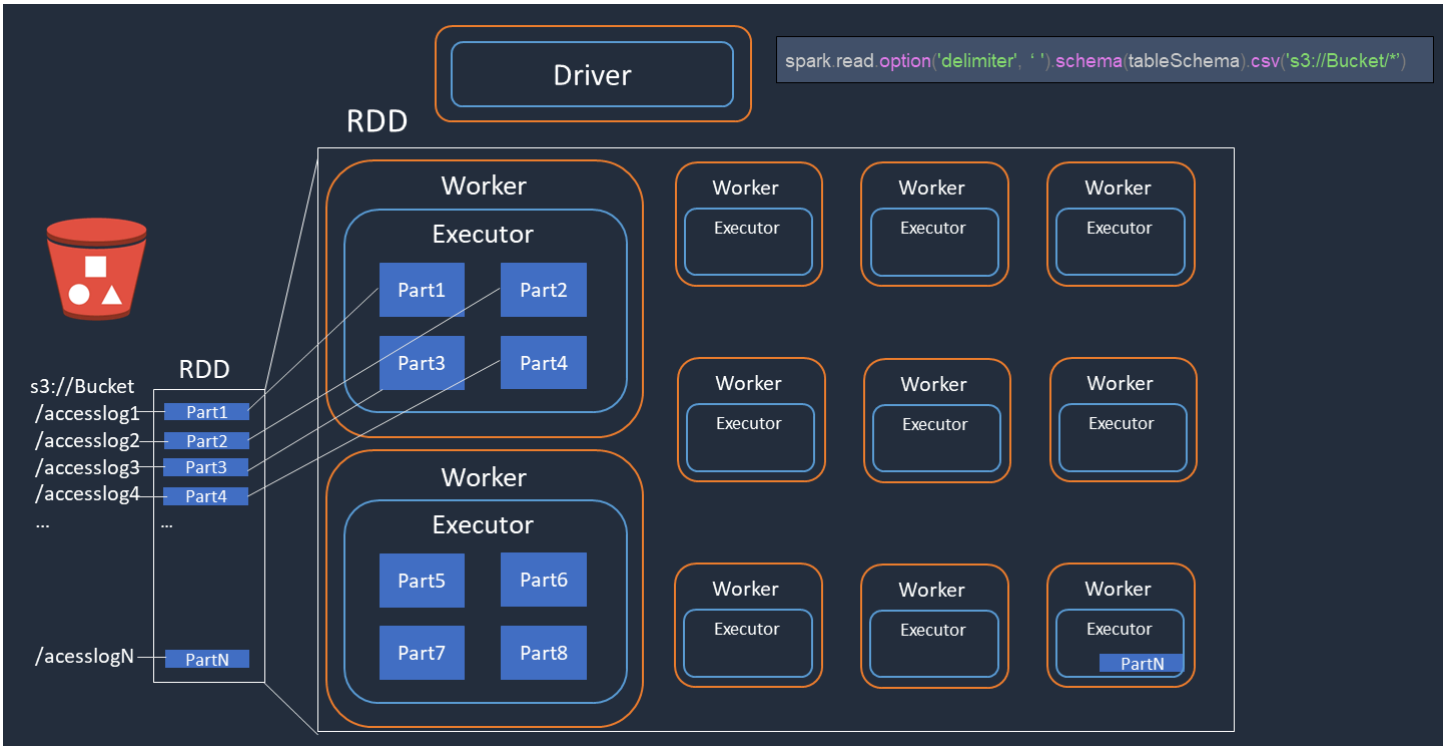


> **Note**
>
> 工作是最佳化平行性時最重要的事情。任務的數量與 RDD 的數量擴展

平行處理

星火並行處理任務，用於加載和轉換數據。

假設您在 Amazon S3 上執行存取日誌檔 (命名`accesslog1 ... accesslogN`) 的分散式處理的範例。下圖顯示了分佈式處理流程。

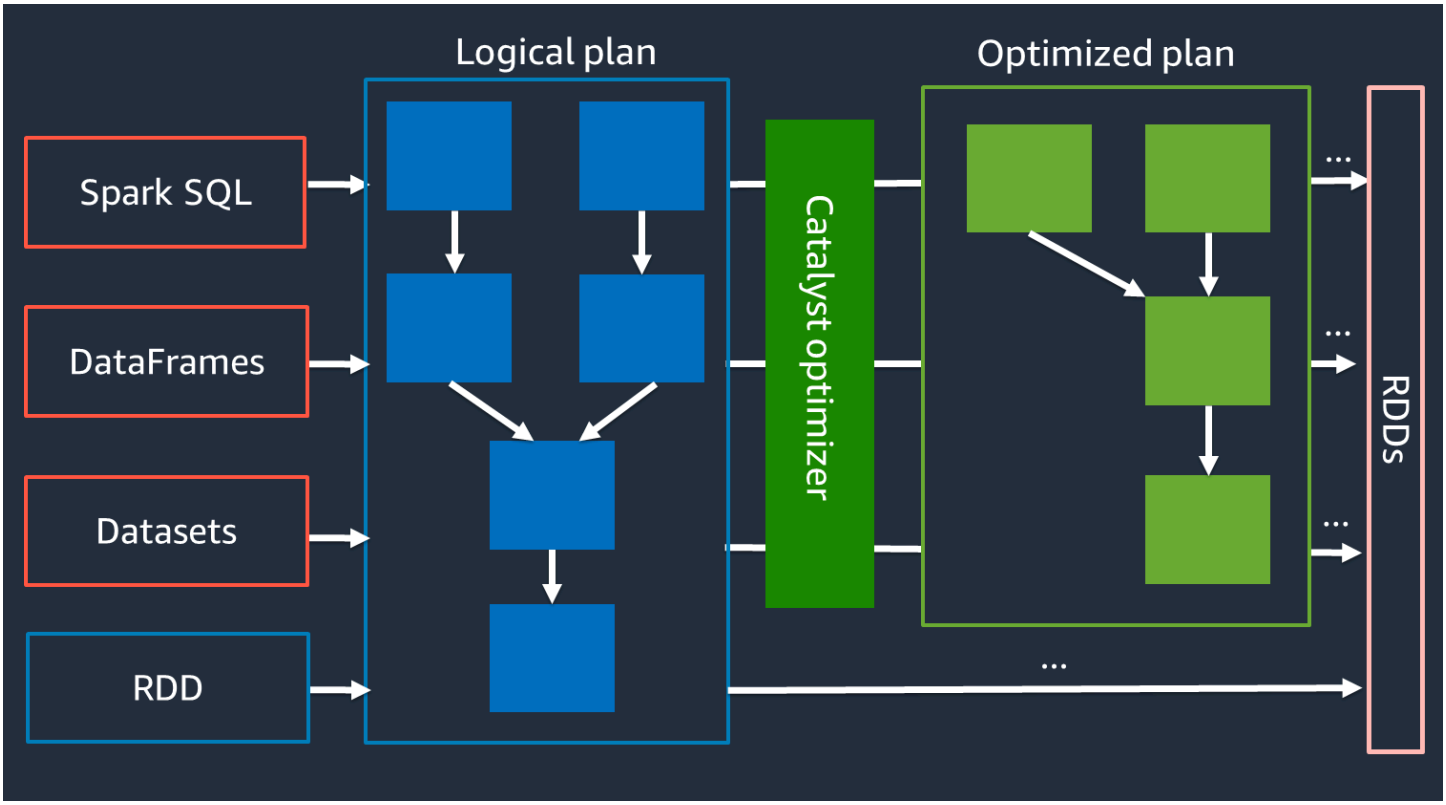


1. Spark 驅動程序創建跨許多 Spark 執行程序分佈式處理的執行計劃。
2. Spark 驅動程序根據執行計劃分配任務每個執行程序。根據預設，Spark 驅動程式會為每個 S3 物件 () 建立 RDD 分割區 (每個分割區都對應於 Spark 工作`Part1 ... N`)。然後 Spark 驅動程序將任務分配給每個執行人。
3. 每個 Spark 任務都會下載其指派的 S3 物件，並將其存放在 RDD 分割區的記憶體中。通過這種方式，多個 Spark 執行程序下載並 parallel 處理其分配的任務。

如需有關初始分割區數目和最佳化的詳細資訊，請參閱平行化工作一節。

催化劑優化

在內部，Spark 使用稱為催化劑優化器的引擎來優化執行計劃。催化劑有一個查詢最佳化工具，您可以在執行高階 Spark API 時使用DataFrame，例如 Spark SQL 和資料集，如下圖所述。



由於催化劑最佳化工具無法直接與 RDD API 搭配使用，因此高階 API 通常會比低階 RDD API 快。對於複雜的聯結，Catalyst 最佳化工具可以透過最佳化工作執行計劃來顯著改善效能 您可以在 Spark UI 的 SQL 選項卡上查看 Spark 任務的優化計劃。

自適應查詢執行

催化劑優化器通過稱為自適應查詢執行過程執行運行時優化。調適性查詢執行會使用執行階段統計資料，在工作執行時重新最佳化查詢的執行計畫。調適性查詢執行提供多種效能挑戰的解決方案，包括合併隨機後分割區、將排序合併聯結轉換為廣播聯結，以及傾斜聯結最佳化，如下列各節所述。

自適應查詢執行在 AWS Glue 3.0 及更高版本中可用，並且在 AWS Glue 4.0（Spark 3.3.0）及更高版本中默認啟用。自適應查詢執行可以通過在代碼`spark.conf.set("spark.sql.adaptive.enabled", "true")`中使用打開和關閉。

合併後隨機分割區

此功能會根據輸出統計資料，減少每次隨機播放後的 RDD 分割區 (合併)。`map`它簡化了運行查詢時洗牌分區號的調整。您不需要設置隨機分區編號來適合您的數據集。Spark 可以在您有足夠大的初始分區數量足夠大後，在運行時選擇適當的隨機分區號。

合併後隨機分割區會`spark.sql.adaptive.coalescePartitions.enabled`在兩者皆啟用`spark.sql.adaptive.enabled`且設定為 true 時。如需詳細資訊，請參閱 Apache 星火文件。

將排序合併聯結轉換為廣播加入

此功能可識別您何時加入兩個大小不同的資料集，並根據該資訊採用更有效率的聯結演算法。如需詳細資訊，請參閱 Apache 星火文件。聯結策略將在「最佳化洗牌」一節中討論。

傾斜加入最佳化

資料偏斜是 Spark 工作最常見的瓶頸之一。它描述了數據傾斜到特定的 RDD 分區（因此，特定的任務），這會延遲應用程序的整體處理時間的情況。這通常會降低聯結作業的效能。歪斜連接優化功能通過將傾斜任務拆分（並在需要時複製）為大致均勻大小的任務來動態處理排序合併聯接中的歪斜。

此功能在設定`spark.sql.adaptive.skewJoin.enabled`為 true 時啟用。如需詳細資訊，請參閱 Apache 星火文件。資料偏斜會在「最佳化洗牌」一節中進一步討論。

使用 Spark UI 調查效能問題

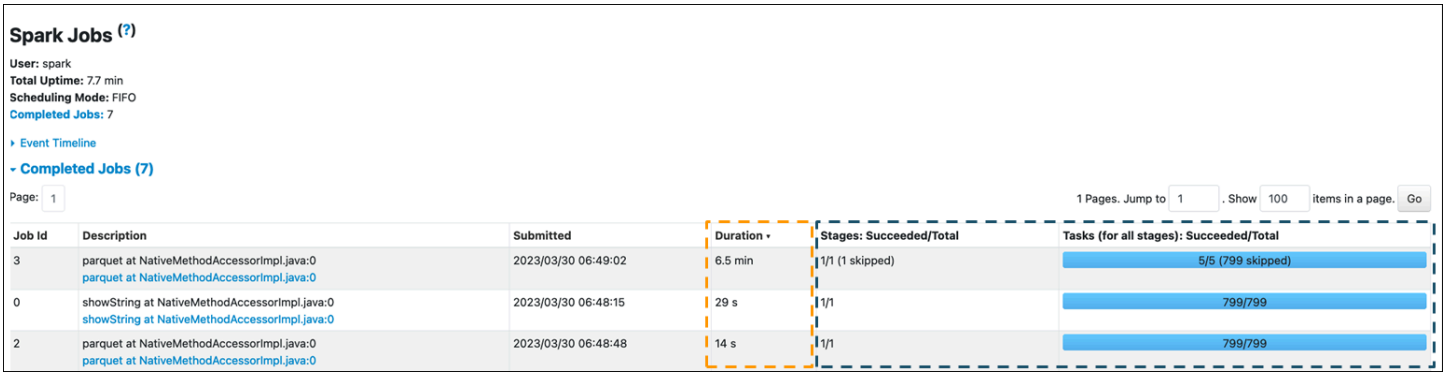
在您套用任何最佳作法來調整 AWS Glue 工作效能之前，我們強烈建議您先剖析效能並找出瓶頸。這將幫助您專注於正確的事情。

為了快速分析，Amazon CloudWatch 指標可提供任務指標的基本視圖。S park 使用者介面提供更深入的效能調整檢視。若要搭配使用 Spark 使用者介面 AWS Glue，您必須為您的 AWS Glue 工作啟用 Spark UI。熟悉 Spark UI 之後，請遵循調整 Spark 工作效能的策略，根據您的發現項目識別和減少瓶頸的影響。

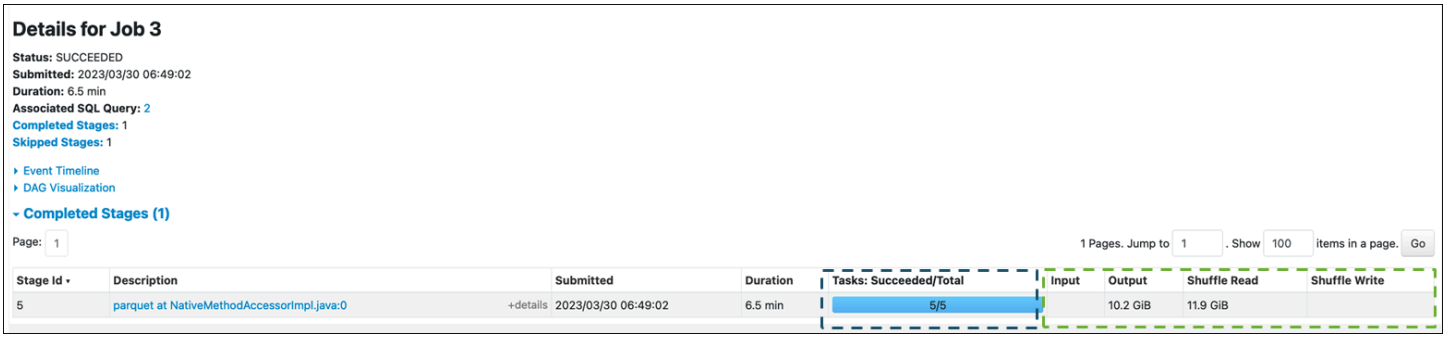
使用星火 UI 識別瓶頸

當您打開星火用戶界面，星火應用程序列在一個表中。根據預設， AWS Glue 工作的應用程式名稱為`nativespark-<Job Name>-<Job Run ID>`。根據作業執行 ID 選擇目標 Spark 應用程式，以開啟 [作業] 索引標籤。不完整的工作執行 (例如串流工作執行) 會列在顯示不完整的應用程式中。

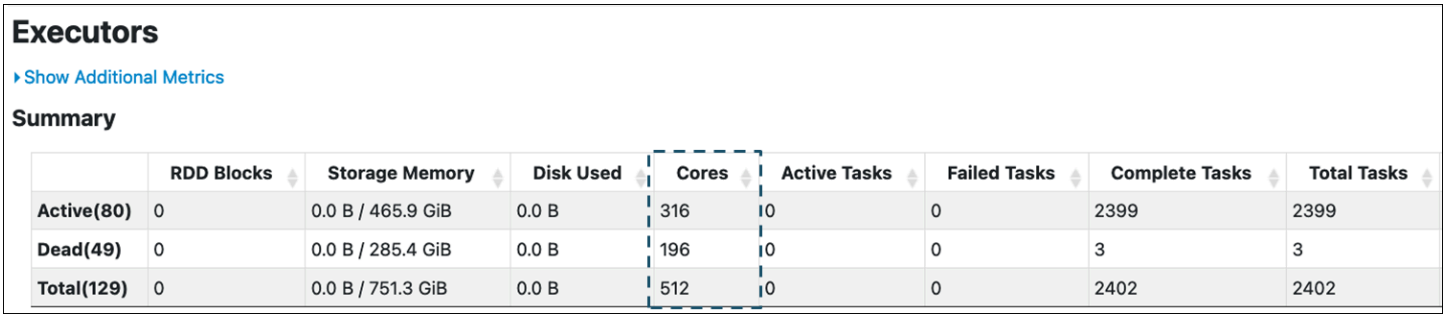
[工作] 索引標籤會顯示 Spark 應用程式中所有工作的摘要。若要判斷任何階段或工作失敗，請檢查工作總數。若要尋找瓶頸，請依選擇持續時間來排序。選擇描述資料欄中顯示的連結，向下展開至長時間執行工作的詳細資訊。



「Job 詳細資訊」頁面會列出階段。在此頁面上，您可以看到整體見解，例如持續時間、成功和工作總數、輸入和輸出的數量，以及隨機播放讀取和隨機寫入的數量。



[執行程式] 索引標籤會詳細顯示 Spark 叢集容量。您可以檢查核心的總數。以下屏幕截圖中顯示的集群總共包含 316 個活動內核和 512 個內核。默認情況下，每個核心可以同時處理一個 Spark 任務。



	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks
Active(80)	0	0.0 B / 465.9 GiB	0.0 B	316	0	0	2399	2399
Dead(49)	0	0.0 B / 285.4 GiB	0.0 B	196	0	0	3	3
Total(129)	0	0.0 B / 751.3 GiB	0.0 B	512	0	0	2402	2402

根據「Job 詳細資料」頁面上5/5顯示的值，階段 5 是最長的階段，但在 512 中只使用 5 個核心。由於此階段的平行程度非常低，但需要大量時間，因此您可以將其識別為瓶頸。為了提高性能，您想了解原因。若要深入瞭解如何辨識並降低常見效能瓶頸的影響，請參閱調整 Spark 工作效能的策略。

調整 Spark 工作績效的策略

準備調校參數時，請使用以下最佳實務：

- 在開始識別問題之前，先決定您的效能目標。
- 嘗試變更調校參數之前，先使用指標來識別問題。

為在調校任務時得到最一致的結果，應制定調校工作的基線策略。

效能調校的基線策略

一般而言，效能調校依照以下工作流程進行：

1. 決定效能目標。
2. 量測指標。
3. 識別瓶頸。
4. 降低瓶頸的影響。
5. 重複步驟 2-4，直到達到預期的目標。

首先，確定您的績效目標。例如，您的目標之一可能是在 3 小時內完成 AWS Glue 工作的執行。定義目標後，測量工作績效指標。確定指標和瓶頸的趨勢，以達到目標。特別是，識別瓶頸對於疑難排解、偵錯和效能調整最為重要。在 Spark 應用程式執行期間，Spark 會在 Spark 事件記錄檔中記錄每個工作的狀態和統計資料。

在中 AWS Glue，您可以透過 Spark 記錄伺服器所提供的 Spark Web UI 來檢視 Spark 度量。 AWS Glue 對於 Spark 任務，可以將 Spark 事件日誌傳送到您在 Amazon S3 中指定的位置。 AWS Glue 也提供範例AWS CloudFormation 範本和 Dockerfile，以便在 Amazon EC2 執行個體或本機電腦上啟動 Spark 歷史記錄伺服器，因此您可以將 Spark UI 與事件日誌搭配使用。

在確定績效目標並確定評估這些目標的指標之後，您可以使用以下各節中的策略開始識別和修復瓶頸。

Spark 工作績效的調整實務

您可以使用下列策略來調整 Spark AWS Glue 工作的效能：

- AWS Glue 資源：
  - 擴充叢集容量
  - 使用最新 AWS Glue 版本
- 星火應用：
  - 減少資料掃描量
  - 平行化工作
  - 優化洗牌
  - 最大限度地減少
  - 優化用戶定義函

使用這些策略之前，您必須能夠存取 Spark 工作的指標和設定。您可以在AWS Glue 文檔中找到此信息。

從 AWS Glue 資源的角度來看，您可以新增 AWS Glue Worker 並使用最新 AWS Glue 版本，以達到效能改善。

從 Apache Spark 應用程式的角度來看，您可以存取數種可改善效能的策略。如果不必要的資料載入 Spark 叢集，您可以將其移除以減少載入的資料量。如果 Spark 叢集資源未充分使用，而且資料 I/O 不足，您可以識別要平行化的工作。如果連接需要大量時間，您可能還需要優化繁重的數據傳輸操作（例如聯接）。您還可以優化工作查詢計劃或降低單個 Spark 任務的計算複雜性。

為了有效地應用這些策略，您必須通過諮詢您的指標來確定它們何時適用。如需詳細資訊，請參閱下列各節。這些技術不僅適用於效能調整，還可用於解決一般問題，例如 out-of-memory (OOM) 錯誤。

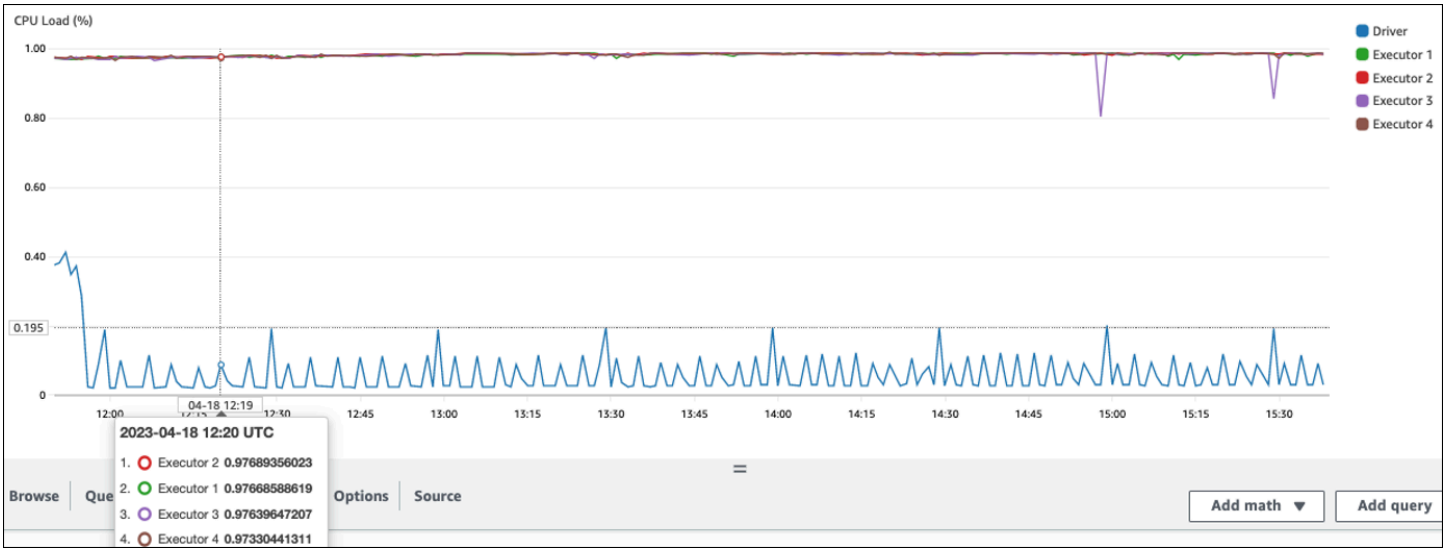
擴充叢集容量

如果您的工作花費太多時間，但執行者消耗了足夠的資源，並且 Spark 正在創建相對於可用內核的大量任務，請考慮擴展叢集容量。若要評估這是否適當，請使用下列測量結果。

CloudWatch 度量

- 檢查 CPU 負載和記憶體使用率，以判斷執行者是否耗用足夠的資源。
- 檢查工作執行的時間長度，以評估處理時間是否太長而無法達到您的績效目標。

在下列範例中，四個執行程式以超過 97% 的 CPU 負載執行，但大約三個小時後仍未完成處理。

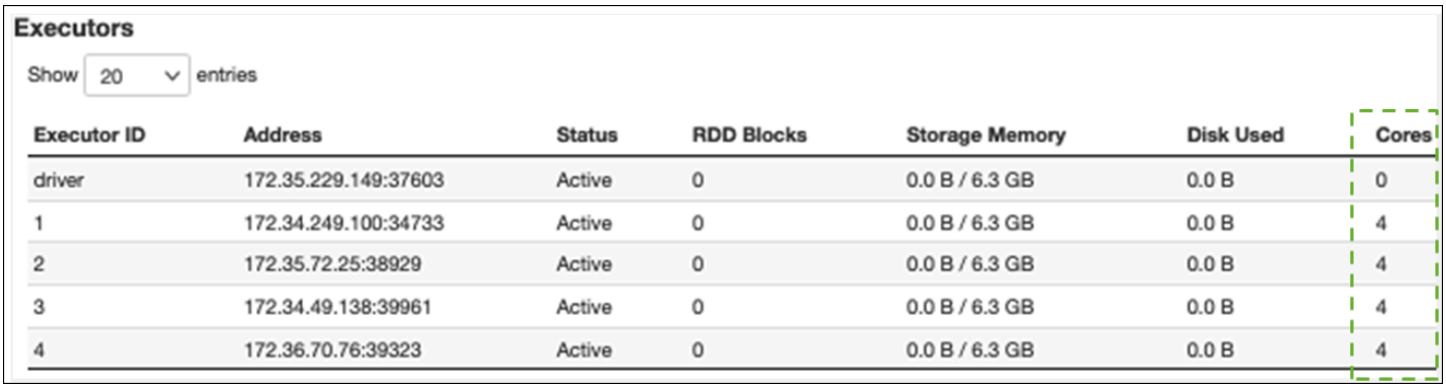


> **Note**
>
> 如果 CPU 負載很低，您可能無法從擴展叢集容量中獲益。

Spark UI

在 Job 標籤或階段索引標籤上，您可以查看每個工作或階段的工作數目。在下面的例子中，星火創建了58100任務。

在「執行程序」選項卡上，您可以看到執行者和任務的總數。在下面的屏幕截圖中，每個 Spark 執行程序具有四個內核，並且可以同時執行四個任務。



在此示例中，Spark 任務的數量（58100)遠大於執行者可以同時處理的 16 個任務（4 個執行程序 ×4 個內核）。

如果您發現這些徵狀，請考慮調整叢集的比例。您可以使用下列選項來調整叢集容量：

- 啟用 AWS Glue Auto Scaling — Auto Scaling 功能適用於 3.0 AWS Glue 版或更高版本中的 AWS Glue 擷取、轉換和載入 (ETL) 和串流任務。 AWS Glue 根據每個階段的分割區數目或工作執行時產生微批次的速率，自動從叢集中新增和移除 Worker。

  如果您發現即使啟用了「Auto Scaling」，Worker 數量仍未增加的情況，請考慮手動新增 Worker。但是，請注意，針對一個階段手動調整可能會導致許多 Worker 在後續階段閒置，因此花費更多成本以實現零效能增益。

  啟用 Auto Scaling 之後，您可以在執行程序度量中看到執行 CloudWatch 程序的數量。使用下列指標來監視 Spark 應用程式中執行程式的需求：

  - `glue.driver.ExecutorAllocationManager.executors.numberAllExecutors`
  - `glue.driver.ExecutorAllocationManager.executors.numberMaxNeededExecutors`

  如需有關指標的詳細資訊，請參閱AWS Glue 使用 Amazon CloudWatch 指標進行監控。
- 向外擴充：增加 AWS Glue 工作者數量 — 您可以手動增加 AWS Glue 工作者的數量。僅在您觀察到閒置的工作人員之前添加 Worker 此時，增加更多員工將在不改善結果的情況下增加成本。如需詳細資訊，請參閱平行化工作。
- 擴充：使用較大的 Worker 類型 — 您可以手動變更 AWS Glue Worker 的執行個體類型，以使用具有更多核心、記憶體和儲存空間的 Worker。較大的 Worker 類型可讓您垂直擴展和執行密集的資料整合工作，例如記憶體密集型資料轉換、傾斜彙總，以及涉及數 PB 資料的實體偵測檢查。

  如果 Spark 驅動程式需要更大容量 (例如，工作查詢計畫相當龐大)，所以向上擴充也會有所協助。如需有關背景工作者類型和效能的詳細資訊，請參閱 AWS 大數據部落格文章使用新的 AWS Glue 較大型背景工作類型 G.4X 和 G.8X 來調整 Apache Spark 工作的規模。

  使用較大的 Worker 也可以減少所需的 Worker 總數，藉由減少密集型作業 (例如 Join) 中的隨機播放，從而提高效能。

使用最新 AWS Glue 版本

我們建議使用最新 AWS Glue 版本。每個版本都內建了數個最佳化和升級，可能會自動改善工作效能。例如， AWS Glue 4.0 提供了以下新功能：

- 新的優化的阿帕奇星火 3.3.0 運行時- AWS Glue 4.0 建立在 Apache 的星火 3.3.0 運行時，帶來可比性能的改進，以開源星火。星火 3.3.0 運行時建立在許多從星火 2.x 的創新。
- 增強的 Amazon Redshift 連接器 — AWS Glue 4.0 及更高版本為阿帕奇星火提供 Amazon Redshift 集成。整合建立在現有的開放原始碼連接器之上，並增強其效能和安全性。此整合可協助應用程式執行速度提高達 10 倍。如需詳細資訊，請參閱有關 Amazon Redshift 與 Apache Spark 整合的部落格文章。
- 使用 CSV 和 JSON 資料進行向量化讀取的 SID 型執行 — 3.0 AWS Glue 版及更新版本新增了最佳化的讀取器，相較於以資料列為基礎的讀取器，可大幅加快整體工作效能。如需有關 CSV 資料的詳細資訊，請參閱使用向量化 SIMD CSV 讀取器最佳化讀取效能。如需有關 JSON 資料的詳細資訊，請參閱使用具有 Apache 箭頭資料欄格式的向量化 SIMD JSON 讀取器。

每個 AWS Glue 版本都將包括此類升級，其中包括連接器，驅動程序和庫更新。如需詳細資訊，請參閱AWS Glue 版本和將 AWS Glue 工作移轉至 4.0 AWS Glue 版。

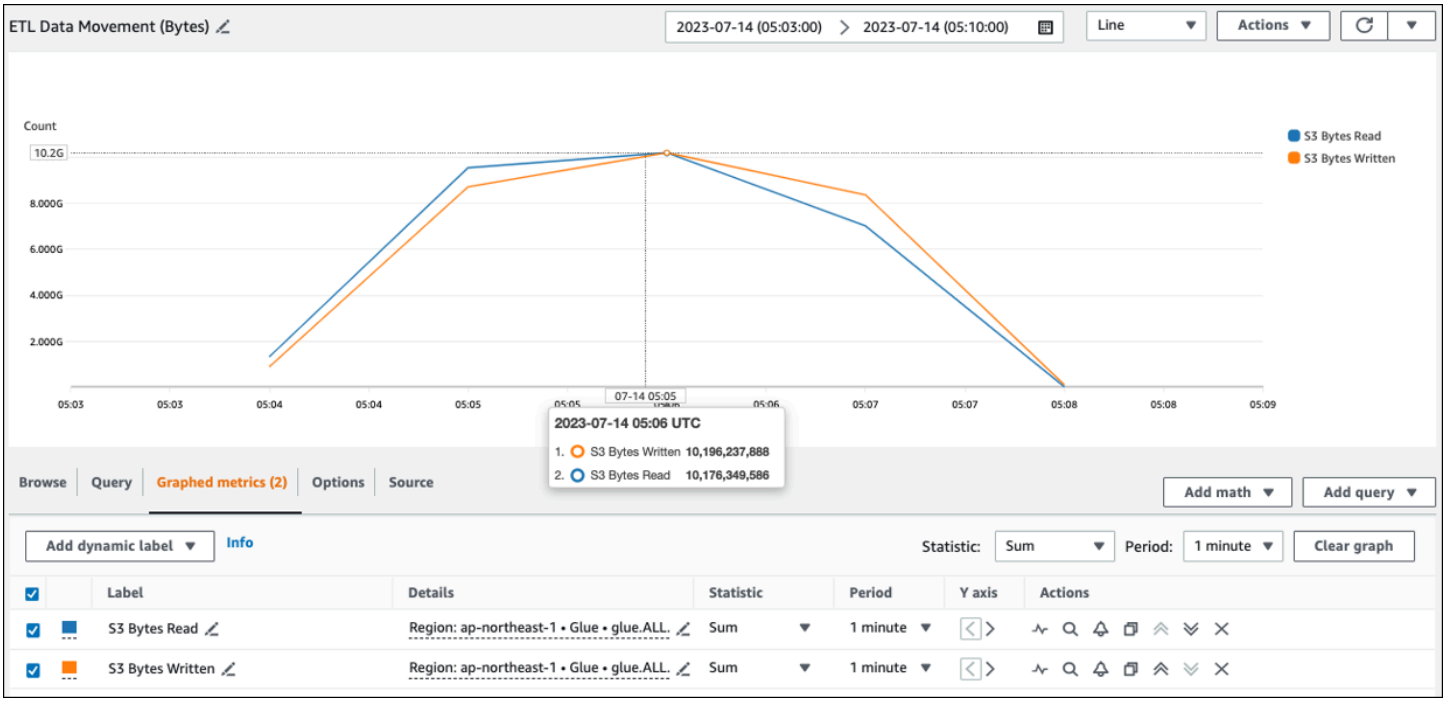
減少資料掃描量

首先，請考慮僅加載所需的數據。您只需減少每個資料來源載入 Spark 叢集的資料量，就可以改善效能。若要評估此方法是否適當，請使用下列量度。

您可以在 Spark UI 中檢查CloudWatch 指標中讀取 Amazon S3 的位元組，以及如 Spark UI 一節中所述的更多詳細資訊。

CloudWatch 度量

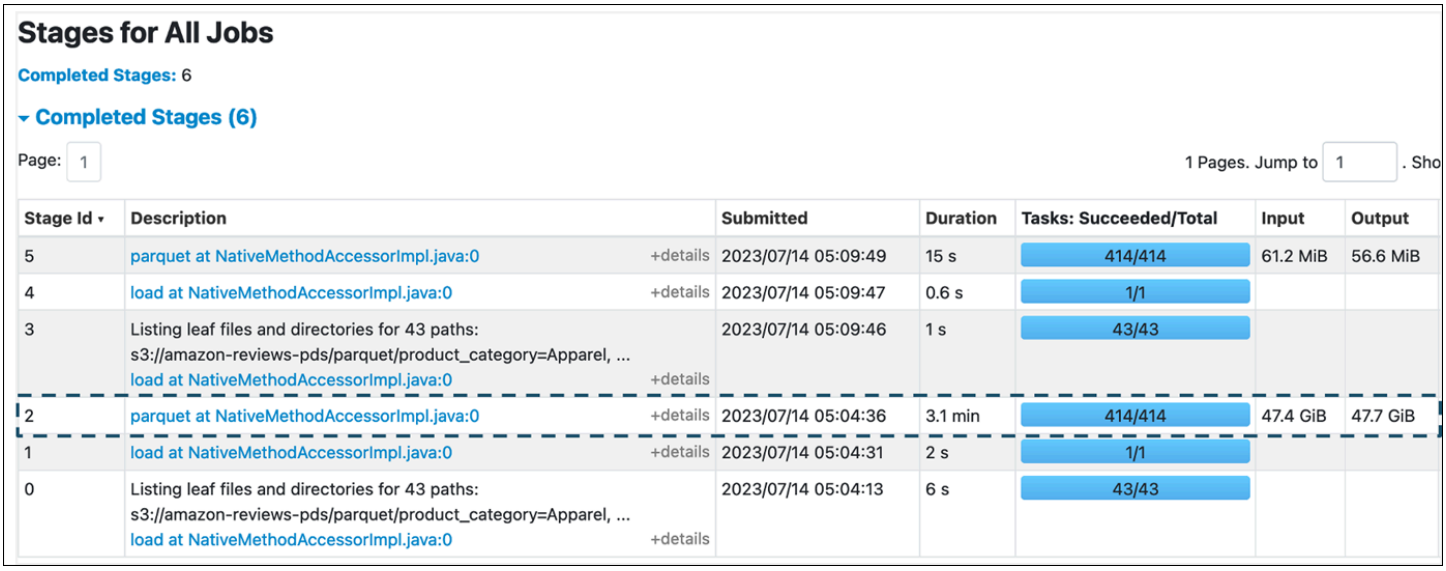
您可以在 ETL 資料移動 (位元組) 中查看 Amazon S3 的近似讀取大小。此指標顯示自上一份報告以來，所有執行者從 Amazon S3 讀取的位元組數。您可以使用它監控來自 Amazon S3 的 ETL 資料移動，並且可以將讀取與外部資料來源的擷取速率進行比較。



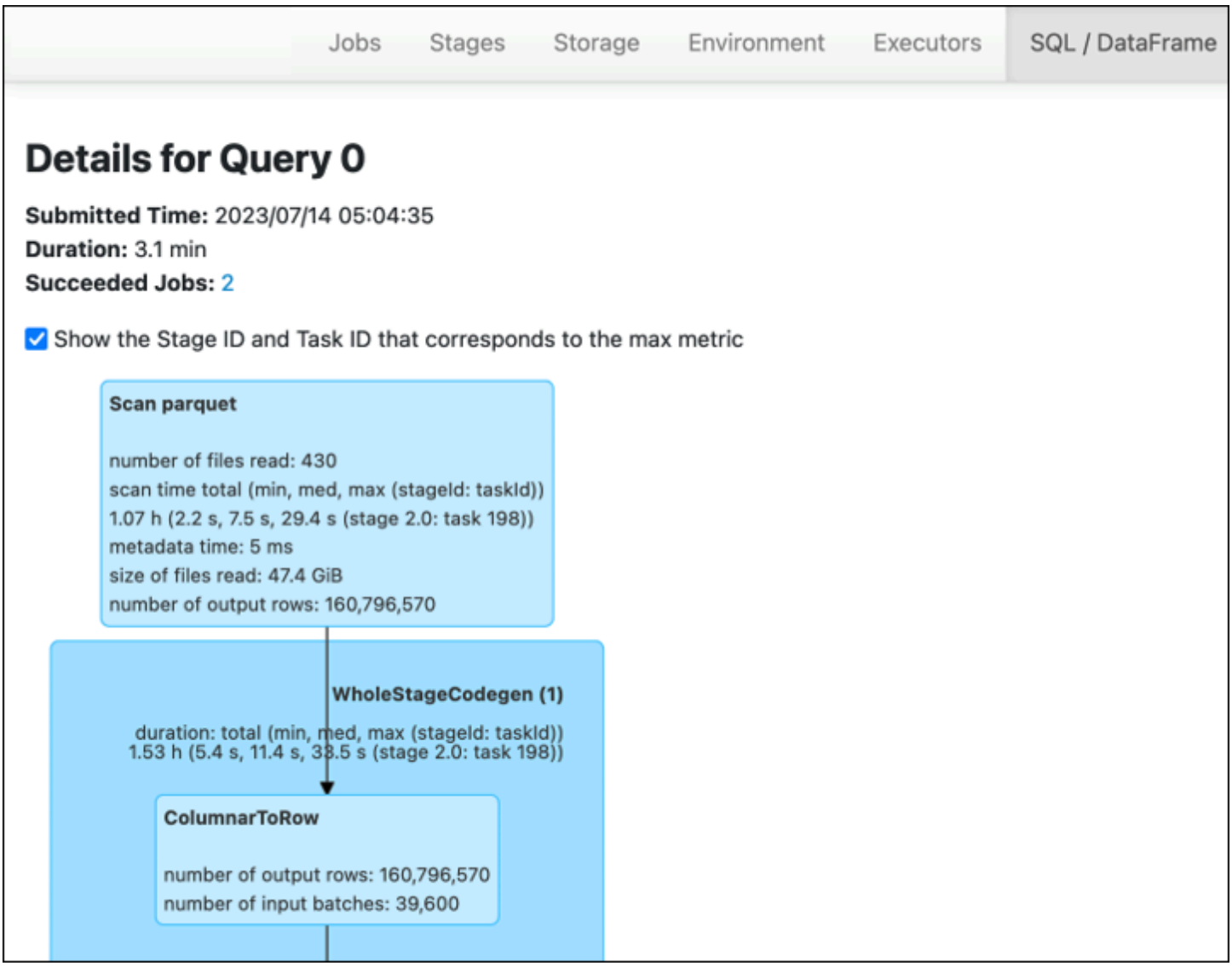
如果您發現的 S3 位元組讀取資料點超出預期，請考慮下列解決方案。

Spark UI

在 Spark UI 的「舞台」索引標籤上，您可以看到「輸入」和「輸出」大小。 AWS Glue 在下列範例中，階段 2 會讀取 47.4 GiB 輸入和 47.7 GiB 輸出，而階段 5 則讀取 61.2 MiB 輸入和 56.6 MiB 輸出。



當您在 AWS Glue 工作中使用 Spark SQL 或 DataFrame 方法時，SQL /D Ataframe 索引標籤會顯示有關這些階段的更多統計資料。在此情況下，階段 2 會顯示讀取的檔案數目：430、讀取的檔案大小：47.4 GiB，以及輸出列數目：160,796,570。



如果您發現正在讀取的資料與使用的資料之間存在大小差異，請嘗試下列解決方案。

Amazon S3

若要減少從 Amazon S3 讀取時載入任務的資料量，請考慮資料集的檔案大小、壓縮、檔案格式和檔案配置 (分割區)。 AWS Glue Spark 作業通常用於原始資料的 ETL，但為了有效率地分散式處理，您需要檢查資料來源格式的功能。

- 檔案大小 — 我們建議將輸入和輸出的檔案大小保持在中等範圍內 (例如 128 MB)。檔案太小和檔案太大可能會造成問題。

大量的小檔案會導致以下問題：

- Amazon S3 上的網路 I/O 負載過重，因為對許多物件發出請求 (例如`ListGet`、或`Head`) 所需的開銷 (與存放相同數量資料的一些物件相比)。
- Spark 驅動程序上的 I/O 和處理負載繁重，這將生成許多分區和任務，並導致過多的並行性。

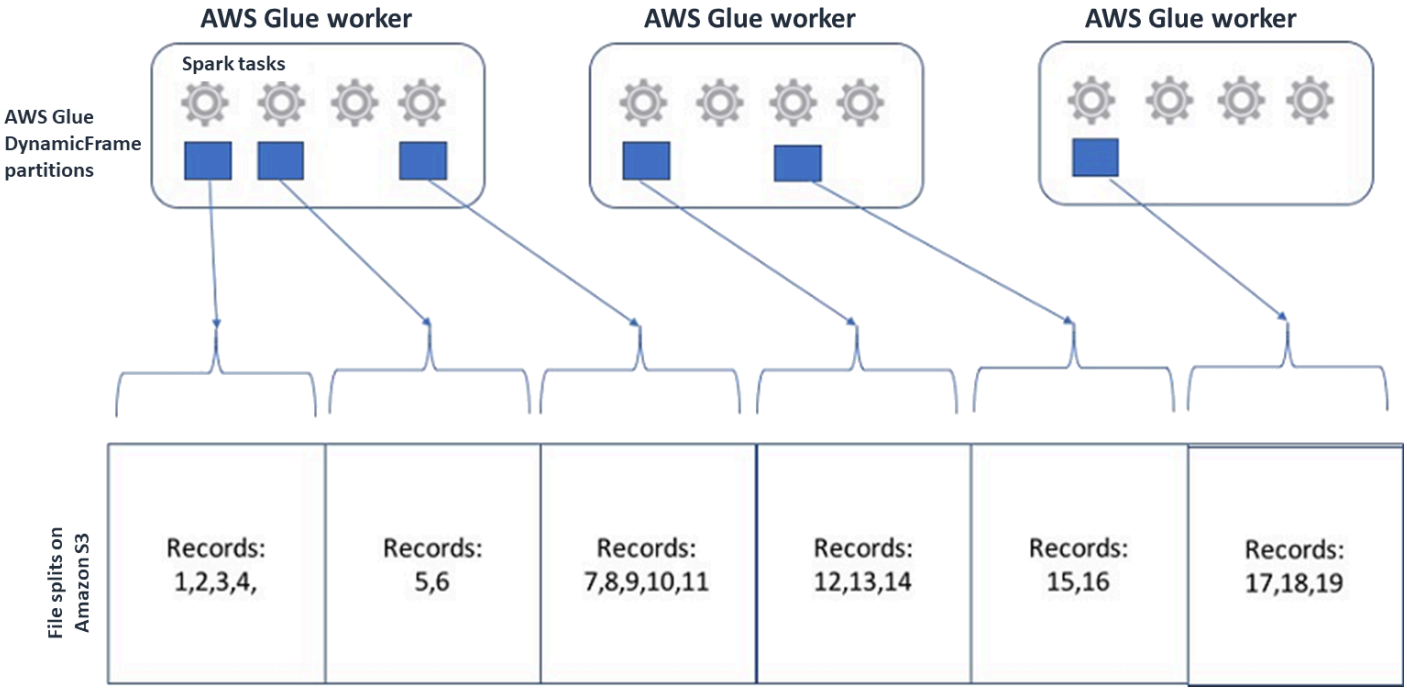
另一方面，如果您的文件類型不可分割（例如 gzip），並且文件太大，則 Spark 應用程序必須等到單個任務完成讀取整個文件。

若要減少為每個小型檔案建立 Apache Spark 工作時所產生的過多平行處理原則，請使用檔案群組。DynamicFrames這種方法可減少 Spark 驅動程式發生 OOM 例外狀況的機會。若要規劃檔案群組，請設定`groupFiles`和`groupSize`參數。下列程式碼範例會在具有這些參數的 ETL 指令碼中使用 AWS Glue DynamicFrame API。

```
dyf = glueContext.create_dynamic_frame_from_options("s3",
    {'paths': ["s3://input-s3-path/"],
    'recurse':True,
    'groupFiles': 'inPartition',
    'groupSize': '1048576'},
    format="json")
```

- 壓縮 — 如果您的 S3 物件大於數百 MB，請考慮壓縮它們。有多種壓縮格式，大致可分為兩種類型：
  - 不可分割的壓縮格式 (例如 gzip) 需要由一個 Worker 解壓縮整個檔案。
  - 可拆分的壓縮格式，例如 bzip2 或 LZO（索引），允許對文件進行部分解壓縮，這些格式可以並行化。

  對於 Spark (以及其他常見的分散式處理引擎)，您會將來源資料檔案分割成引擎可以 parallel 處理的區塊。這些單位通常被稱為拆分。當您的資料採用可分割格式後，最佳化的 AWS Glue 讀取器可以透過提供 `GetObject` API 僅擷取特定區塊的`Range`選項，從 S3 物件擷取分割。請看下面的圖表，看看這將如何在實踐中工作。



只要檔案具有最佳大小或檔案可分割，壓縮資料就可以大幅加快應用程式的速度。較小的資料大小可減少從 Amazon S3 掃描的資料，以及從 Amazon S3 到 Spark 叢集的網路流量。另一方面，壓縮和解壓縮數據需要更多的 CPU。所需的計算量會根據壓縮演算法的壓縮率進行縮放。選擇可拆分壓縮格式時，請考慮這種權衡。

> **Note**
>
> 雖然 gzip 文件通常不是可分割的，但是您可以使用 gzip 壓縮單個實木複合地板塊，並且這些塊可以並行化。

- 檔案格式 — 使用欄格式。阿帕奇鑲木地板和 Apache 的 ORC 是流行的柱狀數據格式。實木複合地板和 ORC 通過採用基於列的壓縮，編碼和基於其數據類型壓縮每列有效地存儲數據。有關鑲木地板編碼的更多信息，請參閱實木地板編碼定義。實木複合地板文件也是可分割的。

  柱格式按列對值進行分組，並將它們存儲在塊中。使用單欄格式時，您可以略過對應於您不打算使用的欄的資料區塊。Spark 應用程式只能擷取您需要的資料行。一般而言，較佳的壓縮比率或略過資料區塊表示從 Amazon S3 讀取較少的位元組，從而獲得更好的效能。這兩種格式也支援下列下推方法來減少 I/O：

  - 投影下推 — 投影下推是一種僅擷取應用程式中指定的欄的技術。您可以在 Spark 應用程式中指定欄，如下列範例所示：
    - DataFrame 例如：`df.select("star_rating")`

    - 火花 SQL 的例子：`spark.sql("select start_rating from <table>")`
  - 謂詞下推-謂詞下推是一種有效地處理和子句的技術。`WHERE` `GROUP BY`這兩種格式都有代表欄值的資料區塊。每個區塊都會保留區塊的統計資料，例如最大值和最小值。Spark 可以使用這些統計資料來判斷是否應該讀取或略過區塊，具體取決於應用程式中使用的篩選器值。若要使用此功能，請在條件中新增更多篩選器，如下列範例所示：
    - DataFrame 例如：`df.select("star_rating").filter("star_rating < 2")`
    - 火花 SQL 的例子：`spark.sql("select * from <table> where star_rating < 2")`
- 檔案配置 — 透過根據資料的使用方式將 S3 資料存放到不同路徑中的物件，您可以有效率地擷取相關資料。如需詳細資訊，請參閱 Amazon S3 文件中的使用前置詞組織物件。 AWS Glue 支援以格式將金鑰和值存放到 Amazon S3 前置詞`key=value`，並依 Amazon S3 路徑對資料進行分區。透過分割資料，您可以限制每個下游分析應用程式掃描的資料量，從而改善效能並降低成本。如需詳細資訊，請參閱〈管理〉中 AWS Glue ETL 輸出的分割區。

  分區將您的表格分成不同的部分，並根據列值（例如年，月和日）將相關數據保存在分組文件中，如以下示例所示。

```
# Partitioning by /YYYY/MM/DD
s3://<YourBucket>/year=2023/month=03/day=31/0000.gz
s3://<YourBucket>/year=2023/month=03/day=01/0000.gz
s3://<YourBucket>/year=2023/month=03/day=02/0000.gz
s3://<YourBucket>/year=2023/month=03/day=03/0000.gz
...
```

  您可以使用中的資料表建立模型，以定義資料集的分區 AWS Glue Data Catalog。然後，您可以使用分割區修剪來限制資料掃描量，如下所示：

  - 對於 AWS Glue DynamicFrame，設定 `push_down_predicate` (或`catalogPartitionPredicate`)。

```
dyf = Glue_context.create_dynamic_frame.from_catalog(
    database=src_database_name,
    table_name=src_table_name,
    push_down_predicate = "year='2023' and month ='03'",
)
```

  - 對於 Spark DataFrame，請設定修剪分割區的固定路徑。

```
df = spark.read.format("json").load("s3://<YourBucket>/year=2023/month=03/*/*.gz")
```

- 對於 Spark SQL，您可以將 where 子句設定為從資料目錄中修剪磁碟分割。

```
df = spark.sql("SELECT * FROM <Table> WHERE year= '2023' and month = '03'")
```

- 要在使用寫入數據時按日期進行分區 AWS Glue，請在列中設置 partitionKeys DynamicFrame 或分區By（ ）中，並在列中 DataFrame 使用日期信息，如下所示。
  - DynamicFrame

```
glue_context.write_dynamic_frame_from_options(
    frame= dyf, connection_type='s3',format='parquet'
    connection_options= {
        'partitionKeys': ["year", "month", "day"],
        'path': 's3://<YourBucket>/<Prefix>/'
    }
)
```

  - DataFrame

```
df.write.mode('append')\
    .partitionBy('year','month','day')\
    .parquet('s3://<YourBucket>/<Prefix>/')
```

這可以提高輸出數據的消費者的性能。

如果您無權更改建立輸入資料集的管道，則無法進行磁碟分割。相反，您可以使用 glob 模式排除不需要的 S3 路徑。在閱讀時設定排除項目 DynamicFrame。例如，下列程式碼會排除 2023 年 01 到 09 個月的天數。

```
dyf = glueContext.create_dynamic_frame.from_catalog(
    database=db,
    table_name=table,
    additional_options = { "exclusions":"[\"**year=2023/month=0[1-9]/**\"]" },
    transformation_ctx='dyf'
)
```

您也可以在「資料目錄」中的表格屬性中設定排除項：

- 索引鍵：`exclusions`

    - 值：["**year=2023/month=0[1-9]/**"]
- Amazon S3 分割區太多 — 避免在包含多種值的資料行 (例如具有數千個值的 ID 資料行) 上分割 Amazon S3 資料。這可能會大幅增加儲存貯體中的分割區數量，因為可能的分割區數量是您所分割之所有欄位的產品。分割區過多可能會導致以下情況：
    - 增加從資料目錄擷取分割區中繼資料的延遲時間
    - 小檔案數量增加，需要更多 Amazon S3 API 請求 (ListGet、和Head)

    例如，當您在partitionBy或中設置日期類型時partitionKeys，日期級分區（例如）對許多用例都yyyy/mm/dd很有用。但是，yyyy/mm/dd/<ID>可能會產生如此多的分割區，以至於整體效能會產生負面影響。

    另一方面，某些用例（例如實時處理應用程序）需要許多分區，例如yyyy/mm/dd/hh。如果您的使用案例需要大量的分割區，請考慮使用資料AWS Glue 分割索引來減少從資料目錄擷取分割區中繼資料的延遲。

數據庫和 JDBC

若要在從資料庫擷取資訊時減少資料掃描，您可以在 SQL 查詢中指定where述詞 (或子句)。不提供 SQL 介面的資料庫將會提供自己的查詢或篩選機制。

使用 Java 資料庫連線 (JDBC) 連線時，請提供具有下列參數where子句的選取查詢：

- 對於 DynamicFrame，請使用「範例查詢」選項。使用時create_dynamic_frame.from_catalog，請按如下方式配置additional_options引數。

```
query = "SELECT * FROM <TableName> where id = 'XX' AND"
datasource0 = glueContext.create_dynamic_frame.from_catalog(
  database = db,
  table_name = table,
  additional_options={
      "sampleQuery": query,
      "hashexpression": key,
      "hashpartitions": 10,
      "enablePartitioningForSampleQuery": True
    },
  transformation_ctx = "datasource0"
)
```

何時using create_dynamic_frame.from_options，如下配置connection_options引數。

```
query = "SELECT * FROM <TableName> where id = 'XX' AND"
datasource0 = glueContext.create_dynamic_frame.from_options(
    connection_type = connection,
    connection_options={
        "url": url,
        "user": user,
        "password": password,
        "dbtable": table,
        "sampleQuery": query,
        "hashexpression": key,
        "hashpartitions": 10,
        "enablePartitioningForSampleQuery": True
    }
)
```

- 對於 DataFrame，請使用查詢選項。

```
query = "SELECT * FROM <TableName> where id = 'XX'"
jdbcDF = spark.read \
.format('jdbc') \
.option('url', url) \
.option('user', user) \
.option('password', pwd) \
.option('query', query) \
.load()
```

- 對於 Amazon Redshift，請使用 AWS Glue 4.0 或更新版本來利用 Amazon Redshift 星火連接器中的下推支援。

```
dyf = glueContext.create_dynamic_frame.from_catalog(
    database = "redshift-dc-database-name",
    table_name = "redshift-table-name",
    redshift_tmp_dir = args["temp-s3-dir"],
    additional_options = {"aws_iam_role": "arn:aws:iam::role-account-id:role/rs-role-name"}
)
```

- 對於其他資料庫，請參閱該資料庫的文件。

AWS Glue 選項

- 若要避免對所有連續工作執行進行完整掃描，並且僅處理上次工作執行期間不存在的資料，請啟用工作書籤。
- 若要限制要處理的輸入資料數量，請使用工作書籤啟用有界執行。這有助於減少每個工作執行的掃描資料量。

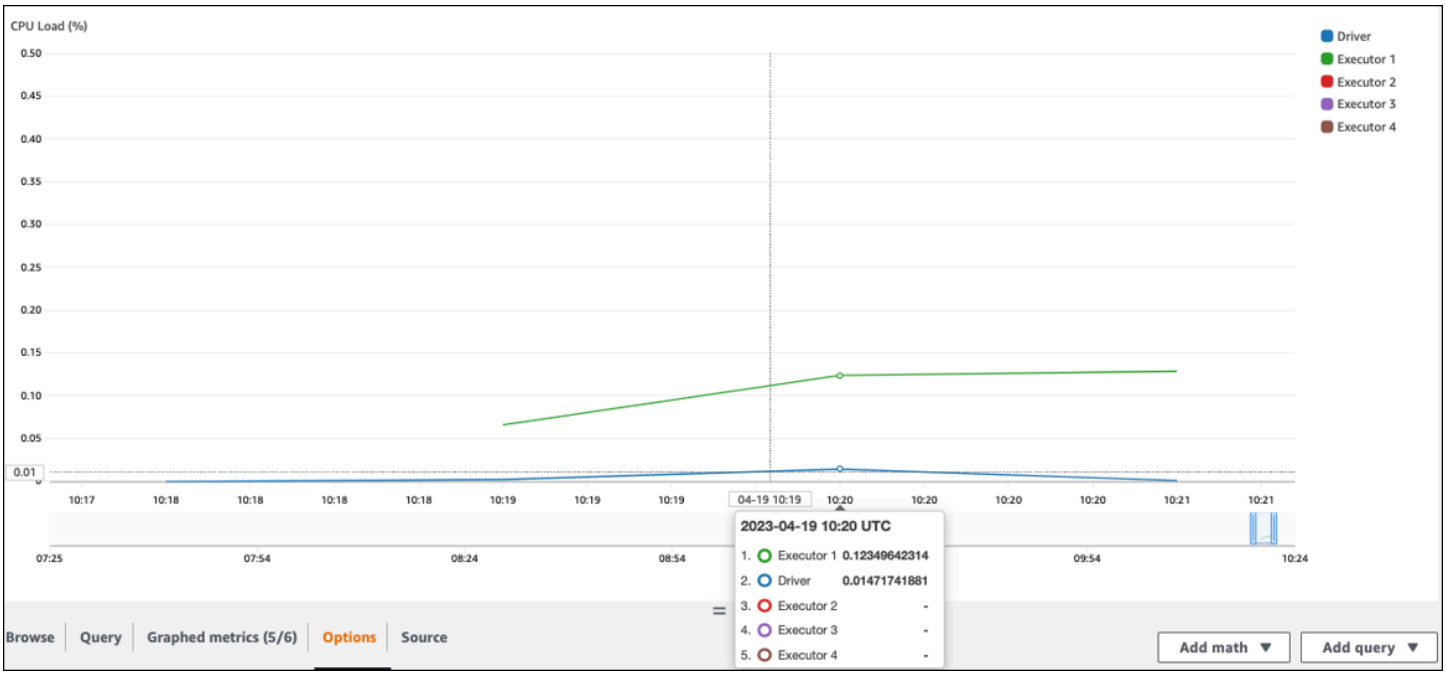
平行化工作

若要最佳化效能，請務必平行處理資料載入和轉換的工作。正如我們在 Apache Spark 的關鍵主題中所討論的那樣，復原分散式資料集 (RDD) 磁碟分割的數目很重要，因為它會決定平行處理原則的程度。Spark 創建的每個任務都對應於 1：1 的基礎上的 RDD 分區。為了達到最佳效能，您需要瞭解如何決定 RDD 分割區的數目，以及如何最佳化該數目。

如果您沒有足夠的平行性，則以下症狀將記錄在CloudWatch 度量和 Spark UI 中。

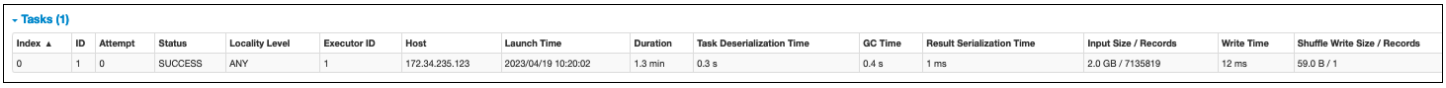
CloudWatch 度量

檢查 CPU 負載和記憶體使用率。如果某些執行者在工作的某個階段沒有處理，那麼改善平行性是適當的。在這種情況下，在可視化的時間範圍內，執行者 1 正在執行任務，但其餘的執行者（2，3 和 4）不是。你可以推斷這些執行者沒有被 Spark 驅動程序分配任務。



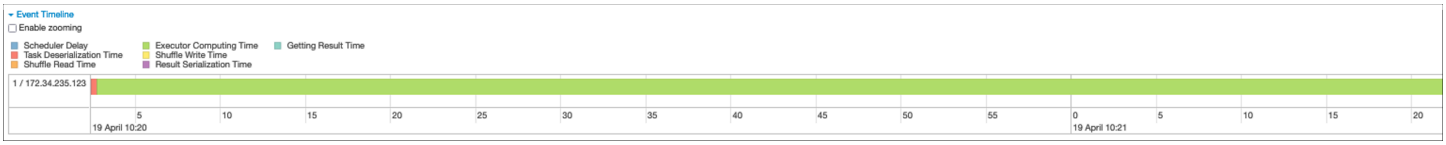
Spark UI

在 Spark UI 的「階段」索引標籤上，您可以看到階段中的工作數目。在這種情況下，Spark 只執行了一項任務。


- Tasks (1)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	Task Deserialization Time	GC Time	Result Serialization Time	Input Size / Records	Write Time	Shuffle Write Size / Records
0	1	0	SUCCESS	ANY	1	172.34.235.123	2023/04/19 10:20:02	1.3 min	0.3 s	0.4 s	1 ms	2.0 GB / 7135819	12 ms	59.0 B / 1

此外，事件時間表顯示執行者 1 正在處理一項任務。這意味著在這個階段的工作完全是在一個執行者上執行的，而其他執行者則處於閒置狀態。



如果您發現這些徵狀，請針對每個資料來源嘗試下列解決方案。

並行處理來自 Amazon S3 的資料載入

若要平行處理來自 Amazon S3 的資料載入，請先檢查預設的分割區數量。然後，您可以手動確定分割區的目標數量，但請務必避免分割區過多。

決定預設的分割區數

對於 Amazon S3，Spark RDD 分割區的初始數量 (每個分割區都對應於 Spark 任務) 是由 Amazon S3 資料集的功能 (例如格式、壓縮和大小) 決定。當您 DataFrame從存放在 Amazon S3 中的 CSV 物件建立 AWS Glue DynamicFrame 或 Spark 時，RDD 分割區 (`NumPartitions`) 的初始數目可大約計算如下：

- 物件大小等於 64 MB：`NumPartitions = Number of Objects`
- 物件大小 > 64 MB: `NumPartitions = Total Object Size / 64 MB`
- 不可分割 (壓縮)：`NumPartitions = Number of Objects`

如「減少資料掃描量」一節中所述，Spark 會將大型 S3 物件分割成可 parallel 處理的分割。當物件大於分割大小時，Spark 會分割物件，並為每個分割建立 RDD 分割區 (和工作)。Spark 的拆分大小基於您的數據格式和運行時環境，但這是一個合理的起始近似值。某些物件會使用不可分割的壓縮格式 (例如 gzip) 進行壓縮，因此 Spark 無法分割它們。

該`NumPartitions`值可能會因您的資料格式、壓縮、 AWS Glue 版本、 AWS Glue Worker 數量和 Spark 組態而有所不同。

例如，當您使用 Spark 載入單一 10 GB 的`csv.gz`物件時 DataFrame，星火驅動程式只會建立一個 RDD 磁碟分割 (`NumPartitions=1`)，因為 gzip 是不可分割的。這會導致一個特定的 Spark 執行程序負載沉重，並且沒有任務分配給剩餘的執行者，如下圖所述。

在 Spark Web UI 階段索引標籤上檢查階段的實際工作數 (**`NumPartitions`**)，或在程式碼**`df.rdd.getNumPartitions()`**中執行以檢查平行處理原則。

遇到 10 GB gzip 檔案時，請檢查產生該檔案的系統是否可以以可分割的格式產生檔案。如果這不是一個選項，您可能需要擴展叢集容量來處理檔案。若要在載入的資料上有效率地執行轉換，您需要使用重新分割叢集中的 Worker 重新平衡 RDD。

手動確定分區的目標數量

根據數據的屬性和 Spark 對某些功能的實現，即使基礎工作仍然可以並行化，您也可能會得到較低的`NumPartitions`價值。如果`NumPartitions`太小，請運行以增`df.repartition(N)`加分區的數量，以便處理可以分配到多個 Spark 執行程序。

在這種情況下，運行`df.repartition(100)`將`NumPartitions`從 1 增加到 100，創建 100 個數據分區，每個分區都有一個可以分配給其他執行者的任務。

此作業會平等`repartition(N)`地分割整個資料 (10 GB /100 個分割區 = 100 MB/ 分割區)，避免資料偏移到特定分割區。

> **Note**
>
> 執行隨機操作 (例如`join`) 時，會根據或的值動態增加或減少分割區的`spark.sql.shuffle.partitions`數目。`spark.default.parallelism`這有助於 Spark 執行程序之間更有效的數據交換。如需詳細資訊，請參閱 S park 文件。

決定分割區的目標數目時，您的目標是最大限度地使用佈建的 AWS Glue Worker。工作 AWS Glue 者數目和 Spark 工作的數目是透過 vCPUs 的數目相關聯。Spark 針對每個 vCPU 核心支援一項工作。在 3.0 AWS Glue 版或更新版本中，您可以使用以下公式計算分區的目標數量。

```
# Calculate NumPartitions by WorkerType
numExecutors = (NumberOfWorkers - 1)
numSlotsPerExecutor =
  4 if WorkerType is G.1X
  8 if WorkerType is G.2X
```

```
  16 if WorkerType is G.4X
  32 if WorkerType is G.8X
NumPartitions = numSlotsPerExecutor * numExecutors

# Example: Glue 4.0 / G.1X / 10 Workers
numExecutors = ( 10 - 1 ) = 9  # 1 Worker reserved on Spark Driver
numSlotsPerExecutor       = 4  # G.1X has 4 vCpu core ( Glue 3.0 or later )
NumPartitions = 9  * 4    = 36
```

在此範例中，每個 G.1X 工作者都會為 Spark 執行程式 () 提供四個 vCPU 核心。`spark.executor.cores = 4`Spark 為每個 vCPU 核心支援一項工作，因此 G.1X Spark 執行程式可以同時執行四項工作 ()。`numSlotPerExecutor`如果工作花費相等的時間，則此數目的分割區會充分利用叢集。但是，某些任務需要比其他任務更長的時間，從而創建閒置內核。如果發生這種情況，請考慮`numPartitions`乘以 2 或 3 來分解並有效地安排瓶頸任務。

分割區太多

過多的分區會創建過多的任務。這會導致 Spark 驅動程序的負載沉重，因為與分佈式處理相關的開銷，例如管理任務和 Spark 執行程序之間的數據交換。

如果工作中的分割區數量大於目標分割區數目，請考慮減少分割區數目。您可以使用下列選項來減少分割區：

- 如果您的檔案大小非常小，請使用 AWS Glue groupFiles 案。您可以減少因啟動 Apache Spark 工作而產生的過多平行處理原則，以處理每個檔案。
- 用`coalesce(N)`於將分割區合併在一起。這是一個低成本的過程。減少分區的數量時，優先`coalesce(N)`於`repartition(N)`，因為`repartition(N)`執行洗牌以平均分配每個分區中的記錄數量。這會增加成本和管理開銷。
- 使用星火 3.x 自適應查詢執行。如 Apache Spark 中關鍵主題中所討論的，調適性查詢執行提供了自動合併分割區數目的函數。當您在執行之前無法知道分區數量時，可以使用此方法。

並行處理來自 JDBC 的資料載入

星火 RDD 分區的數量由配置確定。請注意，預設情況下，只會執行單一工作，以透過`SELECT`查詢掃描整個來源資料集。

兩者 AWS Glue DynamicFrames 和星火 DataFrames 支持跨多個任務並行化 JDBC 數據加載。這是通過使用`where`謂詞將一個查詢拆分為多個`SELECT`查詢來完成的。若要平行化 JDBC 讀取，請設定下列選項：

- 對於 AWS Glue DynamicFrame，設定 `hashfield` (或`hashexpression`)和`hashpartition`。若要深入瞭解，請參閱從 JDBC 表格 parallel 讀取。

```
connection_mysql8_options = {
    "url": "jdbc:mysql://XXXXXXXXXX.XXXXXXX.us-east-1.rds.amazonaws.com:3306/test",
    "dbtable": "medicare_tb",
    "user": "test",
    "password": "XXXXXXXXX",
    "hashexpression":"id",
    "hashpartitions":"10"
}
datasource0 = glueContext.create_dynamic_frame.from_options(
    'mysql',
    connection_options=connection_mysql8_options,
    transformation_ctx= "datasource0"
)
```

- 對於星火 DataFrame，設定`numPartitions`、`partitionColumn`、`lowerBound`、、和`upperBound`。若要深入瞭解，請參閱 JDBC 至其他資料庫。

```
df = spark.read \
    .format("jdbc") \
    .option("url", "jdbc:mysql://XXXXXXXXXX.XXXXXXX.us-east-1.rds.amazonaws.com:3306/
test") \
    .option("dbtable", "medicare_tb") \
    .option("user", "test") \
    .option("password", "XXXXXXXXXX") \
    .option("partitionColumn", "id") \
    .option("numPartitions", "10") \
    .option("lowerBound", "0") \
    .option("upperBound", "1141455") \
    .load()

df.write.format("json").save("s3://bucket_name/Tests/sparkjdbc/with_parallel/")
```

使用 ETL 連接器時，將來自 DynamoDB 的資料載入平行化

星火 RDD 分區的數量由`dynamodb.splits`參數確定。若要平行化讀取來自 Amazon DynamoDB 作，請設定下列選項：

- 增加的值`dynamodb.splits`。

- 遵循 Spark 中 ETL 的連線類型和選項中所述的公式，將參數最佳化。 AWS Glue

從 Kinesis 資料串流平行化資料載入

Spark RDD 分割區的數量取決於來源 Amazon Kinesis Data Streams 中的碎片數量。如果您的資料串流中只有少數碎片，則只會有少數 Spark 工作。這可能會導致下游處理程序的平行處理程序較低。若要平行化從 Kinesis Data Streams 讀取，請設定下列選項：

- 從 Kinesis 資料串流載入資料時，增加碎片數量以獲得更多的平行處理。
- 如果您在微批次中的邏輯足夠複雜，請考慮在刪除不需要的資料欄之後，在批次開頭重新分割資料。

如需詳細資訊，請參閱最佳化 AWS Glue 串流 ETL 工作成本和效能的最佳做法。

資料載入後平行化工作

若要在資料載入後平行化工作，請使用下列選項增加 RDD 分割區的數目：

- 重新分區數據以生成更多數量的分區，尤其是在負載本身無法並行化的情況下，尤其是在初始加載之後立即。

  呼叫`repartition()` DynamicFrame 或 DataFrame，指定分割區數目。一個好的經驗法則是可用內核數的兩到三倍。

  但是，在寫入分區資料表時，這可能會導致檔案爆炸（每個分割區都可能會在每個資料表分割區中產生一個檔案）。為了避免這種情況，您可以 DataFrame 按列重新分區。這使用表分區列，以便在寫入之前組織數據。您可以指定較高數量的分區，而不會在表格分區上獲取較小的文件。但是，請小心避免資料偏差，其中某些分割區值最終會出現大部分資料，並延遲工作的完成。
- 當有洗牌時，增加`spark.sql.shuffle.partitions`值。洗牌時，這也可以幫助解決任何內存問題。

  當您有超過 2,001 個隨機播放分割區時，Spark 會使用壓縮的記憶體格式。如果您的數字接近該值，您可能想要將`spark.sql.shuffle.paritions`值設定為超過該限制，以取得更有效率的表示方式。

優化洗牌

某些作業 (例如`join()`和`groupByKey()`) 需要 Spark 才能執行隨機播放。隨機播放是 Spark 的重新分配數據的機制，以便它在 RDD 分區之間以不同的方式進行分組。洗牌可協助修正效能瓶頸。但是，

由於洗牌通常涉及在 Spark 執行程序之間複製數據，因此隨機播放是一個複雜且昂貴的操作。例如，隨機播放會產生下列成本：

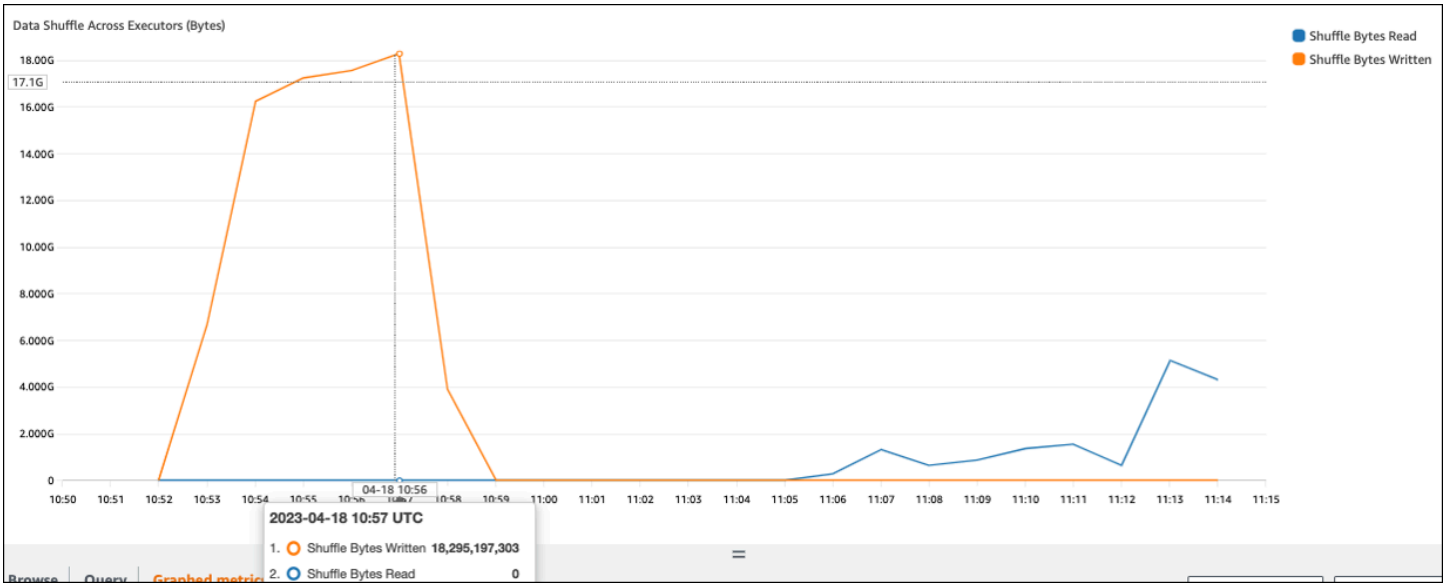
- 磁碟 I/O：
  - 在磁碟上產生大量的中繼檔案。
- 網路 I/O：
  - 需要許多網絡連接（連接數 =`Mapper × Reducer`）。
  - 由於記錄彙總到新的 RDD 分區，這些分區可能託管在不同的 Spark 執行程序上，因此數據集的很大一部分可能會在網絡上的 Spark 執行程序之間移動。
- CPU 和記憶體負載：
  - 排序值並合併資料集。這些操作計劃在執行人身上，對執行人造成沉重的負荷。

隨機播放是 Spark 應用程式效能下降的最重要因素之一。在存儲中間數據時，它可能會耗盡執行程序的本地磁盤上的空間，這會導致 Spark 作業失敗。

您可以在 CloudWatch 指標和 Spark UI 中評估隨機播放效能。

CloudWatch 度量

如果隨機播放字節寫入值與隨機字節讀取相比較高，您的 Spark 作業可能會使用隨機播放操作，例如或。`join()` `groupByKey()`



Spark UI

在 Spark UI 的「舞台」索引標籤上，您可以檢查「隨機播放讀取大小/記錄」值。您也可以在「執行者」選項卡上看到它。

在下面的屏幕截圖中，每個執行者與隨機播放過程交換大約 18.6GB/4020000 條記錄，總隨機播放讀取大小約為 75 GB）。

隨機溢出 (磁碟) 欄會顯示大量的資料溢滿記憶體到磁碟，這可能會造成磁碟已滿或效能問題。

Aggregated Metrics by Executor

Executor ID ▲	Address	Shuffle Read Size / Records	Shuffle Spill (Memory)	Shuffle Spill (Disk)
1	172.35.205.23:46731	18.6 GB / 40210300	98.1 GB	16.8 GB
2	172.35.195.173:46185	18.7 GB / 40246767	117.2 GB	17.3 GB
3	172.36.135.106:35913	18.6 GB / 40253921	101.6 GB	16.6 GB
4	172.34.131.223:46879	18.6 GB / 40190741	99.5 GB	16.4 GB

如果您觀察到這些症狀，而且階段與您的績效目標相比需要太長的時間，或者失敗`Out Of Memory`或`No space left on device`錯誤，請考慮下列解決方案。

優化加入

連接資料表的`join()`作業是最常用的隨機播放作業，但通常是效能瓶頸。由於 join 是一項昂貴的操作，因此我們建議您不要使用它，除非它對您的業務需求至關重要。請提出以下問題，仔細檢查您是否有效地使用資料管線：

- 您是否正在重新計算也在其他可重複使用的工作中執行的連接？
- 您是否加入以將外鍵解析為輸出消費者未使用的值？

確認加入作業對您的業務需求至關重要之後，請參閱下列選項，以符合您需求的方式最佳化加入。

加入前使用下推

在執行聯結 DataFrame 之前，請先篩選出中不必要的列和欄。這具有以下優點：

- 減少隨機播放期間的資料傳輸量
- 減少了 Spark 執行程序處理的量
- 減少資料掃描量

```
# Default
```

```
df_joined  = df1.join(df2, ["product_id"])

# Use Pushdown
df1_select =
 df1.select("product_id","product_title","star_rating").filter(col("star_rating")>=4.0)
df2_select = df2.select("product_id","category_id")
df_joined  = df1_select.join(df2_select, ["product_id"])
```

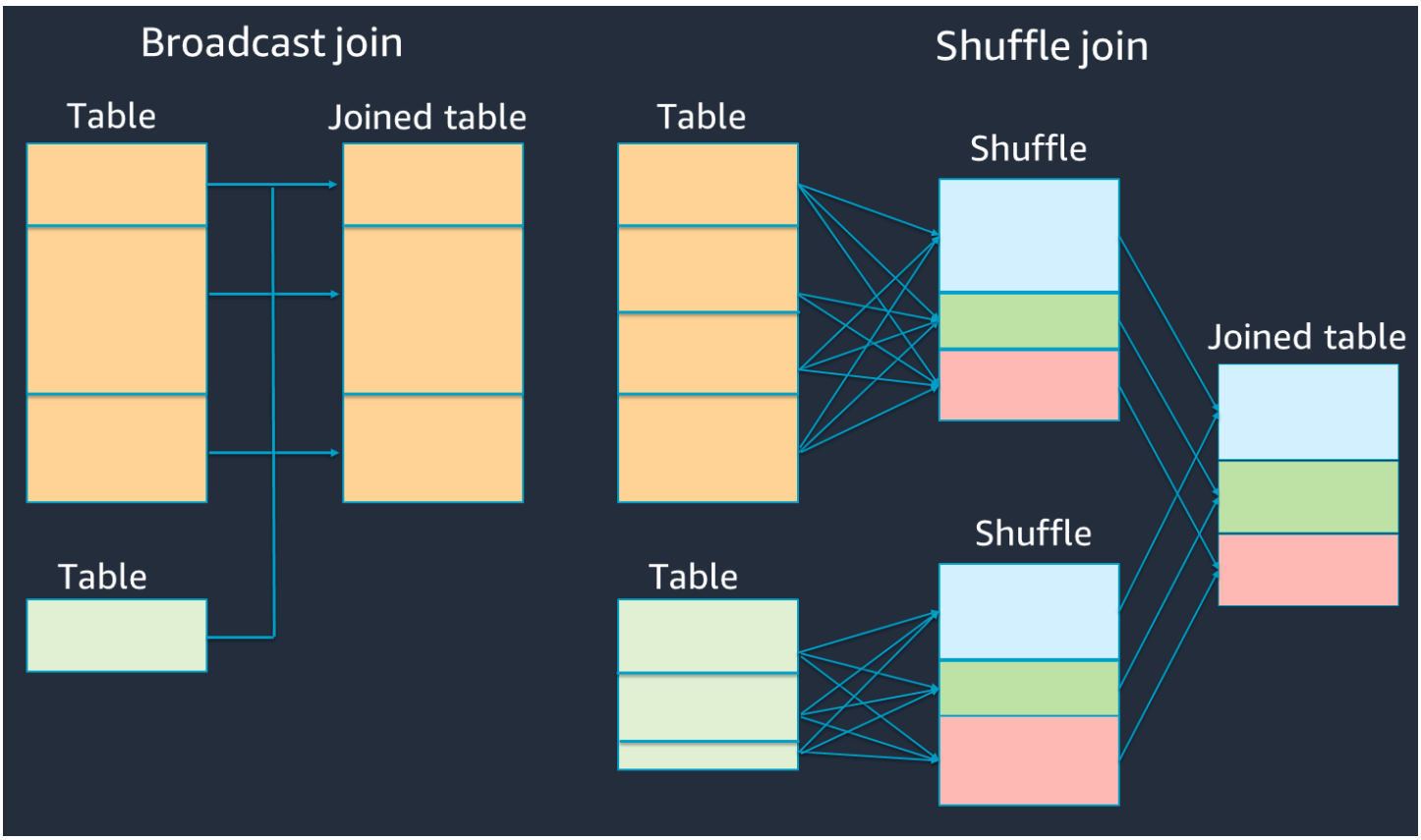
使用 DataFrame 加入

嘗試使用星火高層級 API，如 SparkSQL DataFrame，和數據集，而不是 RDD API 或 DynamicFrame 加入。您可以 DataFrame 使 DynamicFrame 用方法調用轉換為`dyf.toDF()`。如 Apache Spark 中「關鍵主題」一節所述，這些聯結作業會在內部利用 Catalyst 最佳化工具的查詢最佳化功能。

隨機播放和廣播雜湊連接和提示

星火支持兩種類型的連接：隨機連接和廣播哈希加入。廣播哈希連接不需要混洗，並且與隨機連接相比，它可能需要更少的處理。但是，僅在將小桌子連接到大桌子時才適用。當加入可容納單個 Spark 執行程序內存的表時，請考慮使用廣播哈希聯接。

下圖顯示了廣播哈希聯接和隨機連接的高級結構和步驟。



每個聯結的詳細資訊如下：

- 隨機加入：
  - 洗牌哈希連接連接兩個表，而不排序和分配兩個表之間的連接。它適用於可以存儲在 Spark 執行程序內存中的小表的連接。
  - 排序合併連接分配兩個表通過鍵連接和加入之前對它們進行排序。它適用於大型表格的連接。
- 廣播哈希加入：
  - 廣播雜湊聯結會將較小的 RDD 或資料表推送至每個工作節點。然後它將地圖側與較大的 RDD 或表的每個分區結合在一起。

    當您的 RDD 或表之一可以容納內存或可以製作以適合內存時，它適用於聯接。盡可能進行廣播哈希連接是有益的，因為它不需要隨機播放。您可以使用連接提示從 Spark 請求廣播加入，如下所示。

    ```
    # DataFrame
    from pySpark.sql.functions import broadcast
    df_joined= df_big.join(broadcast(df_small), right_df[key] == left_df[key],
     how='inner')

    -- SparkSQL
    SELECT /*+ BROADCAST(t1) /  FROM t1 INNER JOIN t2 ON t1.key = t2.key;
    ```

    如需有關聯結提示的詳細資訊，請參閱聯結提示。

在 AWS Glue 3.0 及更新版本中，您可以啟用「調適性查詢執行」和其他參數，以自動利用廣播雜湊聯結。當任一聯結端的執行階段統計資料小於調適性廣播雜湊聯結臨界值時，調適性查詢執行會將排序合併聯結轉換為廣播雜湊聯結。

在 AWS Glue 3.0 中，您可以通過設置啟用自適應查詢執行`spark.sql.adaptive.enabled=true`。AWS Glue 4.0 預設會啟用自適應查詢執行。

您可以設置與洗牌和廣播哈希連接相關的其他參數：

- `spark.sql.adaptive.localShuffleReader.enabled`
- `spark.sql.adaptive.autoBroadcastJoinThreshold`

如需相關參數的詳細資訊，請參閱將排序合併聯結轉換為廣播聯結。

在 AWS Glue 3.0 和更高版本中，您可以使用其他連接提示進行隨機播放來調整您的行為。

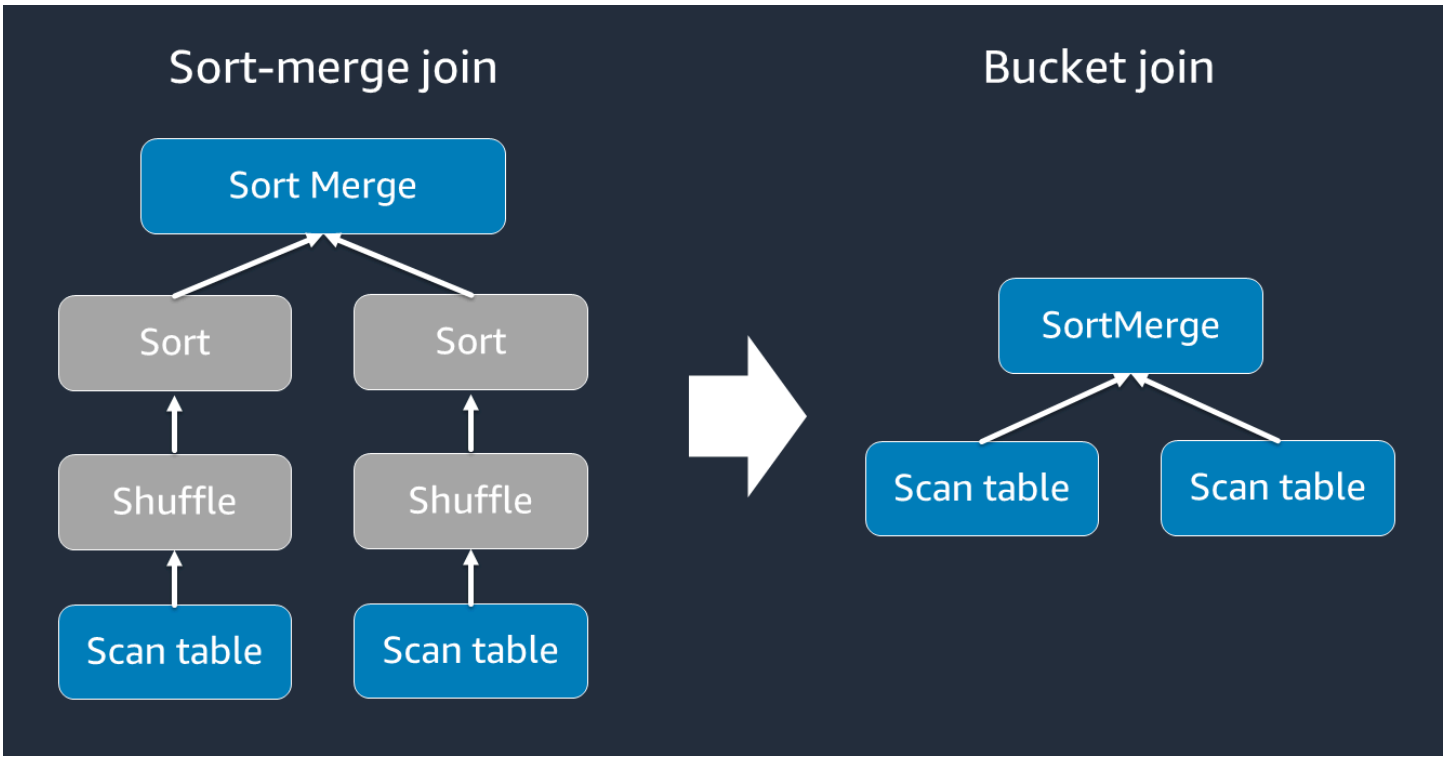
```
-- Join Hints for shuffle sort merge join
SELECT /*+ SHUFFLE_MERGE(t1) /  FROM t1 INNER JOIN t2 ON t1.key = t2.key;
SELECT /*+ MERGEJOIN(t2) /  FROM t1 INNER JOIN t2 ON t1.key = t2.key;
SELECT /*+ MERGE(t1) /  FROM t1 INNER JOIN t2 ON t1.key = t2.key;

-- Join Hints for shuffle hash join
SELECT /*+ SHUFFLE_HASH(t1) /  FROM t1 INNER JOIN t2 ON t1.key = t2.key;

-- Join Hints for shuffle-and-replicate nested loop join
SELECT /*+ SHUFFLE_REPLICATE_NL(t1) /  FROM t1 INNER JOIN t2 ON t1.key = t2.key;
```

使用分段設定

排序合併連接需要兩個階段，隨機排序和排序，然後合併。這兩個階段可能會超載 Spark 執行程序，並導致 OOM 和性能問題時，一些執行程序正在合併，而另一些正在同時排序。在這種情況下，可能可以通過使用分段來有效地加入。Bucketing 將在聯接鍵上預先洗牌和預先排序輸入，然後將排序的數據寫入中間表。透過預先定義已排序的中繼資料表，在連接大型資料表時，可以減少隨機排序和排序步驟的成本。



分組表格對下列項目非常有用：

- 經常透過同一個金鑰關連的資料，例如 `account_id`
- 載入每日累計資料表，例如可在公用資料欄上分組的基礎和差異資料表

您可以通過使用下面的代碼創建一個分組的表。

```
df.write.bucketBy(50, "account_id").sortBy("age").saveAsTable("bucketed_table")
```

在聯接之前 DataFrames 對聯接鍵進行重新分區

若要 DataFrames 在連接之前重新分割兩個連接鍵，請使用下列陳述式。

```
df1_repartitioned = df1.repartition(N,"join_key")
df2_repartitioned = df2.repartition(N,"join_key")
df_joined  = df1_repartitioned.join(df2_repartitioned,"product_id")
```

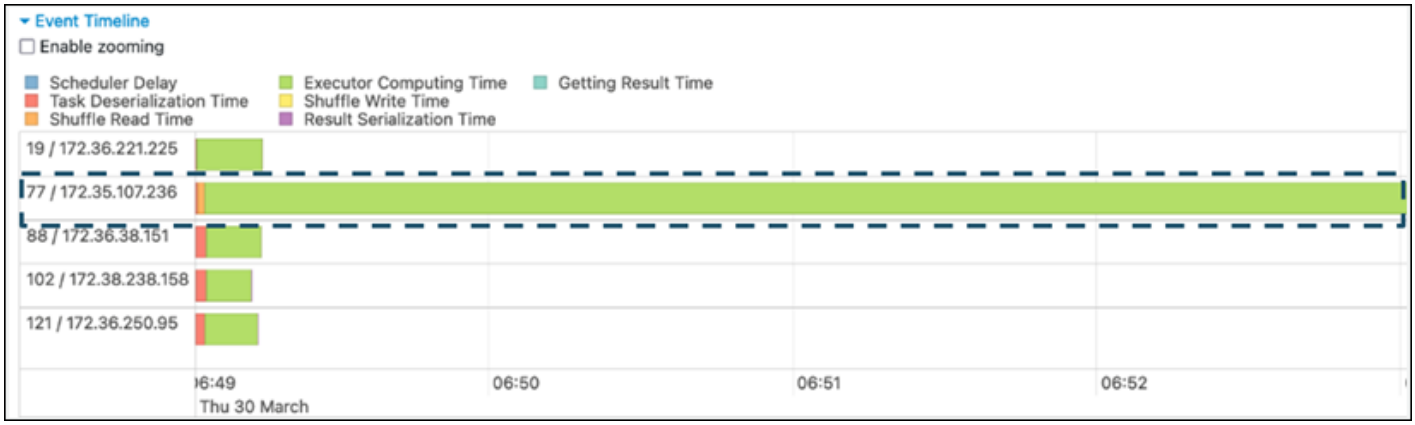
這將在啟動聯接之前對連接鍵上的兩個（仍然是單獨的）RDD 進行分區。如果兩個 RDD 在具有相同分區代碼的相同密鑰上進行分區，則您計劃聯接在一起的 RDD 記錄將很有可能在混洗聯接之前共置在同一個 Worker 上。這可能會透過減少聯結期間的網路活動和資料偏差來改善效能。

克服資料偏斜

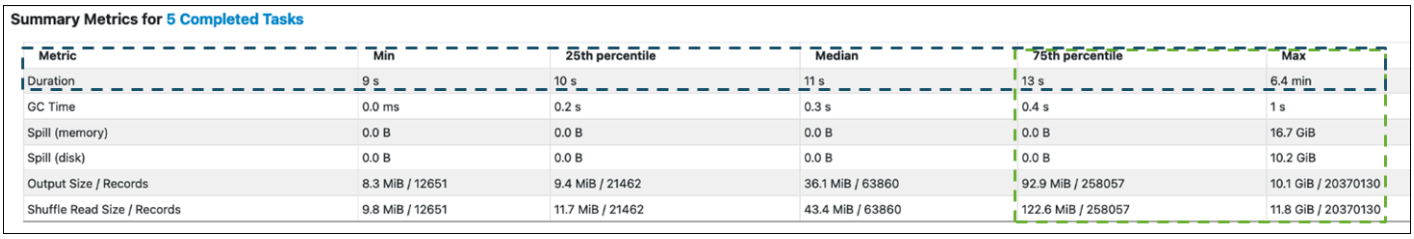
資料偏斜是 Spark 工作瓶頸的最常見原因之一。當數據不是在 RDD 分區中均勻分佈時，就會發生這種情況。這會導致該分區的任務比其他分區花費更長的時間，從而延遲應用程序的整體處理時間。

若要識別資料偏差，請在 Spark UI 中評估下列量度：

- 在 Spark UI 的「舞台」索引標籤上，檢查「事件時間軸」頁面。您可以在下面的屏幕截圖中看到任務分佈不均勻。分佈不均或花費太長時間執行的工作可能表示資料偏斜。



- 另一個重要頁面是「摘要量度」，它會顯示 Spark 工作的統計資料。下列螢幕擷取畫面顯示「持續時間」、「GC 時間」、「溢出 (記憶體)」、「溢出 (磁碟)」等百分位數的度量。



Summary Metrics for 5 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	9 s	10 s	11 s	13 s	6.4 min
GC Time	0.0 ms	0.2 s	0.3 s	0.4 s	1 s
Spill (memory)	0.0 B	0.0 B	0.0 B	0.0 B	16.7 GiB
Spill (disk)	0.0 B	0.0 B	0.0 B	0.0 B	10.2 GiB
Output Size / Records	8.3 MiB / 12651	9.4 MiB / 21462	36.1 MiB / 63860	92.9 MiB / 258057	10.1 GiB / 20370130
Shuffle Read Size / Records	9.8 MiB / 12651	11.7 MiB / 21462	43.4 MiB / 63860	122.6 MiB / 258057	11.8 GiB / 20370130

  當任務均勻分佈時，您將在所有百分位數中看到相似的數字。當數據偏斜時，您將在每個百分位數中看到非常偏差的值。在此範例中，工作持續時間小於 13 秒 (分鐘)、第 25 個百分位數、中位數和第 75 個百分位數。雖然「最大」工作處理的資料量是第 75 個百分位數的 100 倍，但其 6.4 分鐘的持續時間大約是 30 倍。這意味著至少有一個任務（或多達 25% 的任務）花費的時間遠遠超過其餘任務。

如果您看到資料偏斜，請嘗試下列動作：

- 如果您使用 AWS Glue 3.0，請透過設定啟用調適性查詢執行`spark.sql.adaptive.enabled=true`。自適應查詢執行默認情況下在 AWS Glue 4.0 中啟用。

  您也可以透過設定下列相關參數，對聯結引入的資料偏斜使用調適性查詢執行：

  - `spark.sql.adaptive.skewJoin.skewedPartitionFactor`
  - `spark.sql.adaptive.skewJoin.skewedPartitionThresholdInBytes`
  - `spark.sql.adaptive.advisoryPartitionSizeInBytes=128m (128 mebibytes or larger should be good)`
  - `spark.sql.adaptive.coalescePartitions.enabled=true (when you want to coalesce partitions)`

  如需詳細資訊，請參閱 Apache 星火文件。

- 使用具有大範圍值的鍵作為連接鍵。在隨機連接，分區是為一個鍵的每個哈希值確定。如果聯接鍵的基數太低，則散列函數更有可能在分區之間分配數據做了不好的工作。因此，如果您的應用程式和商務邏輯支援它，請考慮使用較高的基數索引鍵或複合索引鍵。

```
# Use Single Primary Key
df_joined = df1_select.join(df2_select, ["primary_key"])
```

```
# Use Composite Key
df_joined = df1_select.join(df2_select, ["primary_key","secondary_key"])
```

使用快取

使用重複性時 DataFrames，請使用或`df.persist()`將計算結果緩存在每個 Spark 執行程序的內存和磁盤上，避免額外的隨機播放`df.cache()`或計算。Spark 也支援在磁碟上持續存放 RDD 或跨多個節點 (儲存層級) 進行複製。

例如，您可以 DataFrames 通過添加來保留`df.persist()`. 當不再需要緩存時，您可以使`unpersist`用丟棄緩存的數據。

```
df = spark.read.parquet("s3://<Bucket>/parquet/product_category=Books/")
df_high_rate = df.filter(col("star_rating")>=4.0)
df_high_rate.persist()

df_joined1 = df_high_rate.join(<Table1>, ["key"])
df_joined2 = df_high_rate.join(<Table2>, ["key"])
df_joined3 = df_high_rate.join(<Table3>, ["key"])
...
df_high_rate.unpersist()
```

移除不需要的火花動作

避免執行不必要的動作`count`，例如`show`、或`collect`。正如在 Apache 星火一節中的關鍵主題討論的那樣，星火是懶惰的。每次對其執行動作時，可能會重新計算每個轉換的 RDD。當您使用許多 Spark 動作時，會呼叫每個動作的多個來源存取、工作計算和隨機播放執行。

如果您不需要`collect()`或在商業環境中執行其他動作，請考慮將其移除。

> **Note**
>
> 盡可能避免`collect()`在商業環境中使用 Spark。此`collect()`動作會將 Spark 執行程式中計算的所有結果傳回給 Spark 驅動程式，這可能會造成 Spark 驅動程式傳回 OOM 錯誤。為了避免 OOM 錯誤，Spark `spark.driver.maxResultSize = 1GB` 默認設置，這將返回給 Spark 驅動程序的最大數據大小限制為 1 GB。

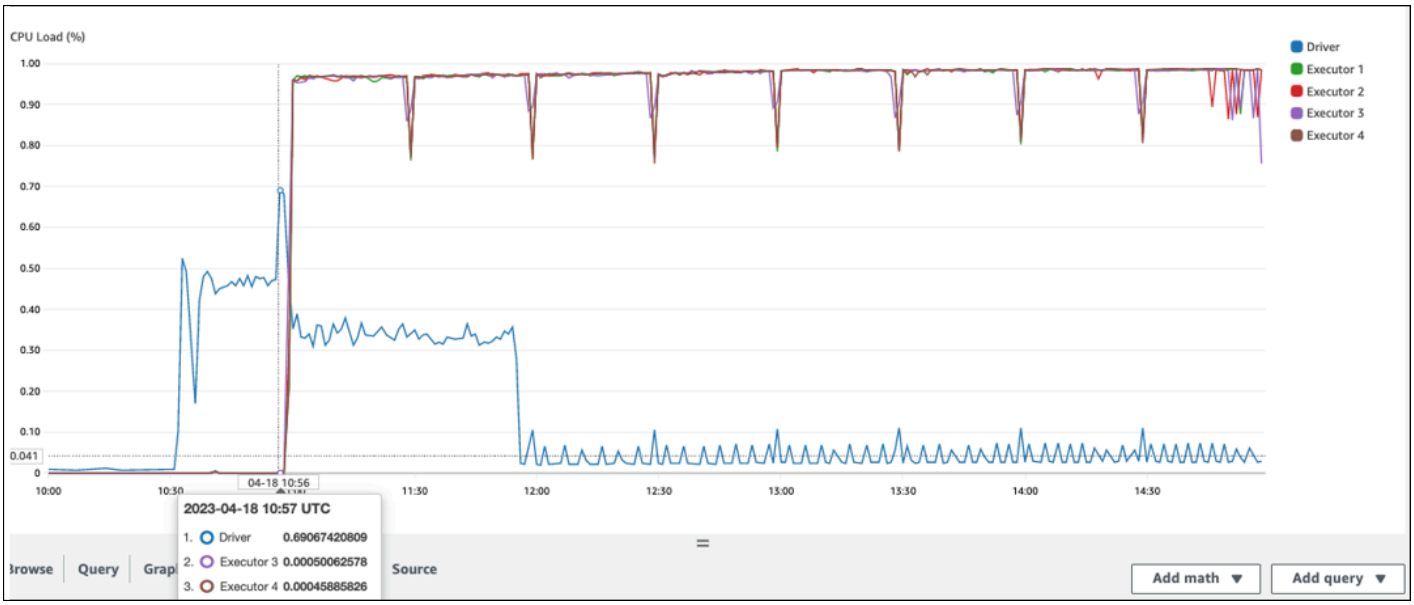
最大限度地減少

正如 Apache 星火中所討論的關鍵主題，星火驅動程序生成執行計劃。根據該計劃，任務被分配給 Spark 執行程序進行分佈式處理。不過，如果有大量的小檔案，或者如果 AWS Glue Data Catalog 包含大量的磁碟分割，Spark 驅動程式可能會成為瓶頸。若要識別高規劃額外負荷，請評估下列指標。

CloudWatch 度量

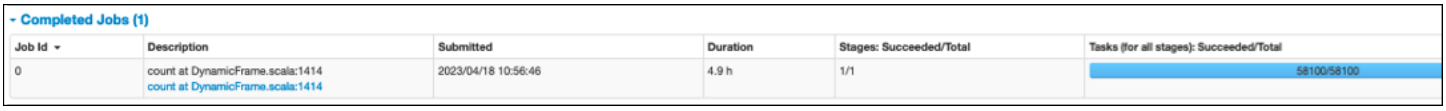
檢查 CPU 負載和記憶體使用率是否有下列情況：

- 火花驅動程序 CPU 負載和內存利用率記錄為高。通常情況下，Spark 驅動程序不會處理您的數據，因此 CPU 負載和內存使用率不會激增。但是，如果 Amazon S3 資料來源有太多小檔案，列出所有 S3 物件和管理大量任務可能會導致資源使用率很高。
- 在 Spark 執行程序處理開始之前有一個很長的差距。在下面的示例屏幕截圖中，Spark 執行程序的 CPU 負載過低，直到 10:57，即使 AWS Glue 工作在 10：00 開始。這表明 Spark 驅動程序可能需要很長時間才能生成執行計劃。在此範例中，擷取資料目錄中的大量分割區，並在 Spark 驅動程式中列出大量的小檔案需要很長時間。



Spark UI

在 Spark UI 的 Job 選項卡上，您可以看到提交的時間。在下列範例中，Spark 驅動程式會在 10:56：46 啟動工作，即使工作是在 10:00:00 開始。 AWS Glue



您也可以在「Job」標籤上看到「工作」(針對所有階段)：成功/總時間。在這種情況下，任務的數量被記錄為58100。如同平行化任務頁面的 Amazon S3 一節所述，任務數量大致對應於 S3 物件的數目。這意味著 Amazon S3 中有大約 58,100 個對象。

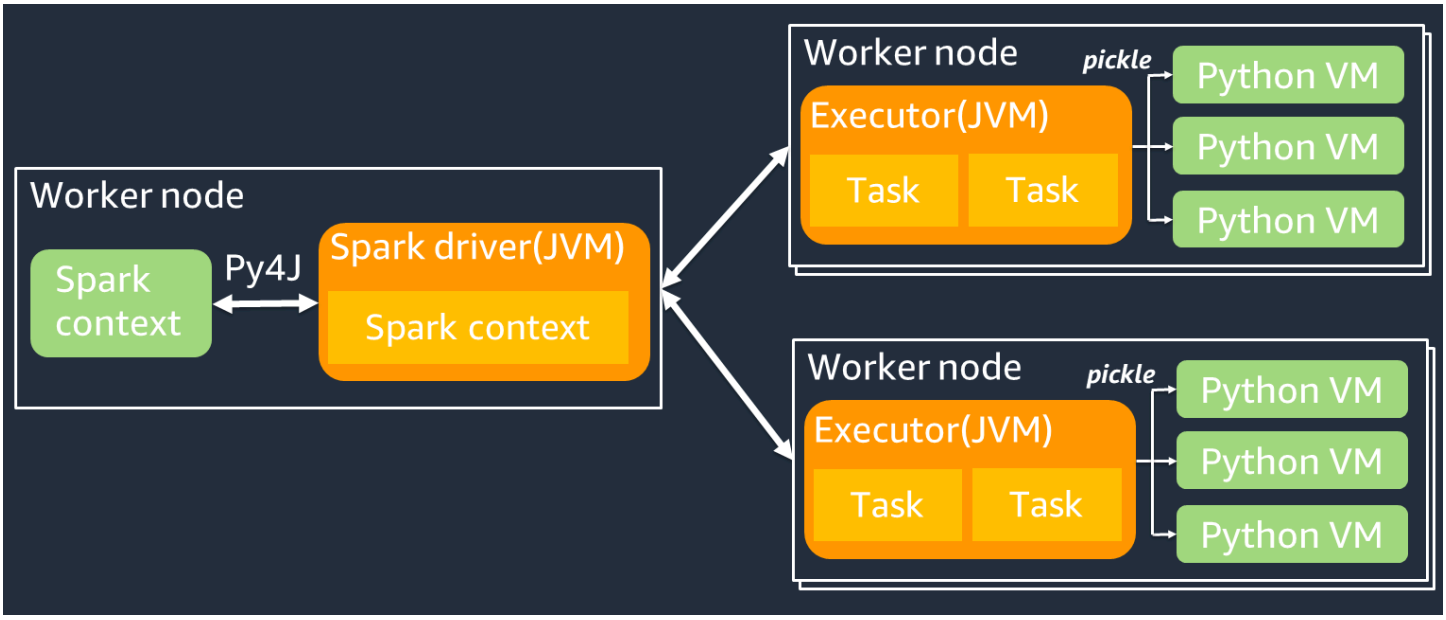
如需有關此工作和時間表的詳細資訊，請檢閱階段索引標籤。如果您發現 Spark 驅動程式出現瓶頸，請考慮下列解決方案：

- 當 Amazon S3 有太多檔案時，請考慮「並行化任務」頁面的「分割太多」區段中有關過度平行處理的指導。
- 當 Amazon S3 有太多的分割區時，請在「減少資料掃描量」頁面的 Amazon S3 分割區過多一節中考慮有關過度分割的指導。如果有許多分割區，請啟用AWS Glue 分割區索引，以減少從「資料目錄」擷取分割區中繼資料的延遲。如需詳細資訊，請參閱使用資料 AWS Glue 分割索引改善查詢效能。
- 當 JDBC 有太多的分割區時，請降低該`hashpartition`值。
- 當 DynamoDB 有太多分割區時，請降低該`dynamodb.splits`值。
- 當串流工作有太多的分割區時，請降低碎片數目。

優化用戶定義函

用戶定義函數（UDF）和`RDD.map` PySpark 通常會顯著降低性能。這是因為在 Spark 的底層 Scala 實現中準確表示您的 Python 代碼所需的開銷。

下圖顯示了 PySpark 作業的體系結構。當您使用時 PySpark，星火驅動程序使用 Py4j 的庫從 Python 調用 Java 方法。當調用 Spark SQL 或 DataFrame 內置函數時，Python 和 Scala 之間的性能差異很小，因為函數使用優化的執行計劃在每個執行程序的 JVM 上運行。



如果您使用自己的 Python 邏輯（例如使用）`map/ mapPartitions/ udf`，則該任務將在 Python 運行時環境中運行。管理兩個環境會產生額外負荷成本。此外，您必須轉換記憶體中的資料，以供 JVM 執行階段環境的內建函數使用。泡菜是默認情況下用於 JVM 和 Python 運行時之間交換的序列化格式。但是，這種序列化和反序列化的成本非常高，因此用 Java 或斯卡拉編寫的 UDF 比 Python UDF 更快。

為了避免中的序列化和反序列化開銷 PySpark，請考慮以下幾點：

- 使用內置的星火 SQL 函數-考慮用星火 SQL 或內 DataFrame 置函數替換自己的 UDF 或映射函數。當運行星火 SQL 或 DataFrame 內置函數時，Python 和斯卡拉之間的性能差異很小，因為任務是在每個執行程序的 JVM 上處理的。
- 在斯卡拉或 Java 中實現 UDF-考慮使用 Java 或斯卡拉編寫的 UDF，因為它們在 JVM 上運行。
- 對向量化工作負載使用 Apache 箭頭式 UDF — 考慮使用以箭頭為基礎的 UDF。此功能也稱為向量化 UDF（熊貓 UDF）。Apache 箭頭是一種與語言無關的內存中數據格式， AWS Glue 可用於在 JVM 和 Python 進程之間有效地傳輸數據。對於使用熊貓或 NumPy數據的 Python 用戶來說，這是目前最有益的。

  箭頭是一個柱狀（矢量化）格式。它的使用不是自動的，可能需要對配置或代碼進行一些小的更改，以充分利用並確保兼容性。如需詳細資訊和限制，請參閱中的 Apache 箭頭 PySpark。

  下面的例子比較標準 Python 中的基本增量 UDF，作為矢量化的 UDF，並在星火 SQL。

標準 Python

示例時間為 3.20（秒）。

範例程式碼

```
# DataSet
df = spark.range(10000000).selectExpr("id AS a","id AS b")

# UDF Example
def plus(a,b):
    return a+b
spark.udf.register("plus",plus)

df.selectExpr("count(plus(a,b))").collect()
```

執行計劃

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- HashAggregate(keys=[], functions=[count(pythonUDF0#124)])
+- Exchange SinglePartition, ENSURE_REQUIREMENTS, [id=#580]
+- HashAggregate(keys=[], functions=[partial_count(pythonUDF0#124)])
+- Project [pythonUDF0#124]
+- BatchEvalPython [plus(a#116L, b#117L)], [pythonUDF0#124]
+- Project [id#114L AS a#116L, id#114L AS b#117L]
+- Range (0, 10000000, step=1, splits=16)
```

向量化的 UDF

示例時間為 0.59（秒）。

向量化的 UDF 比上一個 UDF 範例快 5 倍。檢查`Physical Plan`，你可以看到`ArrowEvalPython`，這表明這個應用程序是由 Apache 箭矢量化。若要啟用向量化 UDF，您必須在程式碼`spark.sql.execution.arrow.pyspark.enabled = true`中指定。

範例程式碼

```
# Vectorized UDF
from pyspark.sql.types import LongType
```

```
from pyspark.sql.functions import count, pandas_udf

# Enable Apache Arrow Support
spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", "true")

# DataSet
df = spark.range(10000000).selectExpr("id AS a","id AS b")

# Annotate pandas_udf to use Vectorized UDF
@pandas_udf(LongType())
def pandas_plus(a,b):
    return a+b
spark.udf.register("pandas_plus",pandas_plus)

df.selectExpr("count(pandas_plus(a,b))").collect()
```

執行計劃

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- HashAggregate(keys=[], functions=[count(pythonUDF0#1082L)],
 output=[count(pandas_plus(a, b))#1080L])
+- Exchange SinglePartition, ENSURE_REQUIREMENTS, [id=#5985]
+- HashAggregate(keys=[], functions=[partial_count(pythonUDF0#1082L)],
 output=[count#1084L])
+- Project [pythonUDF0#1082L]
+- ArrowEvalPython [pandas_plus(a#1074L, b#1075L)], [pythonUDF0#1082L], 200
+- Project [id#1072L AS a#1074L, id#1072L AS b#1075L]
+- Range (0, 10000000, step=1, splits=16)
```

Spark SQL

示例時間是 0.087（秒）。

星火 SQL 比矢量化的 UDF 快得多，因為這些任務在沒有 Python 運行時的情況下在每個執行程序的 JVM 上運行。如果您可以使用內建函式取代 UDF，建議您這麼做。

範例程式碼

```
df.createOrReplaceTempView("test")
spark.sql("select count(a+b) from test").collect()
```

使用熊貓進行大數據

如果您已經熟悉熊貓並希望將 Spark 用於大數據，則可以在 Spark 上使用熊貓 API。 AWS Glue 4.0 及更高版本支持它。要開始使用，您可以使用官方筆記本快速入門：Spark 上的熊貓 API。如需詳細資訊，請參閱PySpark 文件。

資源

- AWS Glue
- 效能調整 (星火 SQL 指南)
- AWS Glue 優化工作坊

文件歷史紀錄

下表描述了本指南的重大變更。如果您想收到有關未來更新的通知，可以訂閱 RSS 摘要。

變更	描述	日期
初次出版	—	2024年1月2日

AWS 規定指引詞彙

以下是 AWS 規範性指引所提供的策略、指南和模式中常用的術語。若要建議項目，請使用詞彙表末尾的提供意見回饋連結。

數字

7 R

將應用程式移至雲端的七種常見遷移策略。這些策略以 Gartner 在 2011 年確定的 5 R 為基礎，包括以下內容：

- 重構/重新架構 – 充分利用雲端原生功能來移動應用程式並修改其架構，以提高敏捷性、效能和可擴展性。這通常涉及移植作業系統和資料庫。範例：將您的現場部署 Oracle 資料庫遷移到與 Amazon Aurora PostgreSQL 相容的版本。
- 平台轉換 (隨即重塑) – 將應用程式移至雲端，並引入一定程度的優化以利用雲端功能。範例：將您的現場部署 Oracle 資料庫遷移至 Amazon Relational Database Service 服務 (Amazon RDS)，適用於 AWS 雲端.
- 重新購買 (捨棄再購買) – 切換至不同的產品，通常從傳統授權移至 SaaS 模型。範例：將您的客戶關係管理 (CRM) 系統遷移至 SalesForce.com。
- 主機轉換 (隨即轉移) – 將應用程式移至雲端，而不進行任何變更以利用雲端功能。範例：將您的現場部署 Oracle 資料庫遷移至中 EC2 執行個體上的 Oracle 資料庫 AWS 雲端。
- 重新放置 (虛擬機器監視器等級隨即轉移) – 將基礎設施移至雲端，無需購買新硬體、重寫應用程式或修改現有操作。您可以將伺服器從內部部署平台遷移到相同平台的雲端服務。範例：將 Microsoft Hyper-V應用程式移轉至 AWS.
- 保留 (重新檢視) – 將應用程式保留在來源環境中。其中可能包括需要重要重構的應用程式，且您希望將該工作延遲到以後，以及您想要保留的舊版應用程式，因為沒有業務理由來進行遷移。
- 淘汰 – 解除委任或移除來源環境中不再需要的應用程式。

A

ABAC

請參閱以屬性為基礎的存取控制。

抽象的服務

請參閱受管理服務。

酸

請參閱原子性、一致性、隔離性、耐用性。

主動-主動式遷移

一種資料庫遷移方法，其中來源和目標資料庫保持同步 (透過使用雙向複寫工具或雙重寫入操作)，且兩個資料庫都在遷移期間處理來自連接應用程式的交易。此方法支援小型、受控制批次的遷移，而不需要一次性切換。它比主動-被動遷移更具彈性，但需要更多的工作。

主動-被動式遷移

一種資料庫遷移方法，其中來源和目標資料庫保持同步，但只有來源資料庫處理來自連接應用程式的交易，同時將資料複寫至目標資料庫。目標資料庫在遷移期間不接受任何交易。

聚合函數

在一組資料列上運作，並計算群組的單一傳回值的 SQL 函數。彙總函式的範例包括SUM和MAX。

AI

請參閱人工智慧。

艾奧運

請參閱人工智慧作業。

匿名化

永久刪除資料集中個人資訊的程序。匿名化可以幫助保護個人隱私。匿名資料不再被視為個人資料。

反模式

一種經常使用的解決方案，用於解決方案的生產力適得其反，效果不佳或效果低於替代方案。

應用控制

一種安全性方法，只允許使用核准的應用程式，以協助保護系統免受惡意軟體的攻擊。

應用程式組合

有關組織使用的每個應用程式的詳細資訊的集合，包括建置和維護應用程式的成本及其商業價值。此資訊是產品組合探索和分析程序的關鍵，有助於識別要遷移、現代化和優化的應用程式並排定其優先順序。

人工智慧 (AI)

電腦科學領域，致力於使用運算技術來執行通常與人類相關的認知功能，例如學習、解決問題和識別模式。如需詳細資訊，請參閱什麼是人工智慧？

人工智慧操作 (AIOps)

使用機器學習技術解決操作問題、減少操作事件和人工干預以及提高服務品質的程序。如需有關如何在 AWS 遷移策略中使用 AIOps 的詳細資訊，請參閱操作整合指南。

非對稱加密

一種加密演算法，它使用一對金鑰：一個用於加密的公有金鑰和一個用於解密的私有金鑰。您可以共用公有金鑰，因為它不用於解密，但對私有金鑰存取應受到高度限制。

原子性、一致性、隔離性、耐久性 (ACID)

一組軟體屬性，即使在出現錯誤、電源故障或其他問題的情況下，也能確保資料庫的資料有效性和操作可靠性。

屬性型存取控制 (ABAC)

根據使用者屬性 (例如部門、工作職責和團隊名稱) 建立精細許可的實務。如需詳細資訊，請參閱 AWS Identity and Access Management (IAM) 文件 AWS中的 ABAC。

授權資料來源

儲存資料主要版本的位置，被認為是最可靠的資訊來源。您可以將授權資料來源中的資料複製到其他位置，以便處理或修改資料，例如匿名化、編輯或將其虛擬化。

可用區域

一個獨立的位置，與其他 AWS 區域 可用區域中的故障隔離，並為相同區域中的其他可用區域提供廉價、低延遲的網路連線能力。

AWS 雲端採用架構 (AWS CAF)

指導方針和最佳做法的架構，可協 AWS 助組織制定有效率且有效的計畫，以順利移轉至雲端。AWS CAF 將指導組織到六個重點領域，稱為觀點：業務，人員，治理，平台，安全性和運營。業務、人員和控管層面著重於業務技能和程序；平台、安全和操作層面著重於技術技能和程序。例如，人員層面針對處理人力資源 (HR)、人員配備功能和人員管理的利害關係人。針對此觀點，AWS CAF 為人員開發、訓練和通訊提供指導，以協助組織為成功採用雲端做好準備。如需詳細資訊，請參閱 AWS CAF 網站和 AWS CAF 白皮書。

AWS 工作負載資格架構 (AWS WQF)

可評估資料庫移轉工作負載、建議移轉策略並提供工作預估的工具。 AWS WQF 包含在 AWS Schema Conversion Tool ()AWS SCT中。它會分析資料庫結構描述和程式碼物件、應用程式程式碼、相依性和效能特性，並提供評估報告。

B

壞機器人

旨在破壞或對個人或組織造成傷害的機器人。

BCP

請參閱業務連續性規劃。

行為圖

資源行為的統一互動式檢視，以及一段時間後的互動。您可以將行為圖與 Amazon Detective 搭配使用來檢查失敗的登入嘗試、可疑的 API 呼叫和類似動作。如需詳細資訊，請參閱偵測文件中的行為圖中的資料。

大端序系統

首先儲存最高有效位元組的系統。另請參閱 「位元順序」。

二進制分類

預測二進制結果的過程 (兩個可能的類別之一)。例如，ML 模型可能需要預測諸如「此電子郵件是否是垃圾郵件？」等問題 或「產品是書還是汽車？」

Bloom 篩選條件

一種機率性、記憶體高效的資料結構，用於測試元素是否為集的成員。

藍/綠部署

建立兩個獨立但相同環境的部署策略。您可以在一個環境中執行目前的應用程式版本 (藍色)，而在另一個環境 (綠色) 中執行新的應用程式版本。此策略可協助您以最小的影響快速回復。

機器人

透過網際網路執行自動化工作並模擬人類活動或互動的軟體應用程式。某些漫遊器是有用的或有益的，例如用於索引 Internet 上信息的網絡爬蟲。其他一些機器人（稱為不良機器人）旨在破壞或對個人或組織造成傷害。

殭屍網絡

受惡意軟件感染並受到單一方（稱為機器人牧民或機器人操作員）控制的機器人網絡。殭屍網絡是擴展機器人及其影響的最著名機制。

分支

程式碼儲存庫包含的區域。儲存庫中建立的第一個分支是主要分支。您可以從現有分支建立新分支，然後在新分支中開發功能或修正錯誤。您建立用來建立功能的分支通常稱為功能分支。當準備好發佈功能時，可以將功能分支合併回主要分支。如需詳細資訊，請參閱關於分支 (GitHub 文件)。

防碎玻璃訪問

在特殊情況下，並透過核准的程序，使用者可以快速取得他 AWS 帳戶 們通常沒有存取權限的存取權。如需詳細資訊，請參閱 AWS Well-Architected 指南中的實作防破玻璃程序指標。

棕地策略

環境中的現有基礎設施。對系統架構採用棕地策略時，可以根據目前系統和基礎設施的限制來設計架構。如果正在擴展現有基礎設施，則可能會混合棕地和綠地策略。

緩衝快取

儲存最常存取資料的記憶體區域。

業務能力

業務如何創造價值 (例如，銷售、客戶服務或營銷)。業務能力可驅動微服務架構和開發決策。如需詳細資訊，請參閱在 AWS上執行容器化微服務白皮書的圍繞業務能力進行組織部分。

業務連續性規劃 (BCP)

一種解決破壞性事件 (如大規模遷移) 對營運的潛在影響並使業務能夠快速恢復營運的計畫。

C

咖啡

請參閱AWS 雲端採用架構。

金絲雀部署

向最終用戶發行版本的緩慢和增量版本。當您有信心時，您可以部署新版本並完全取代目前的版本。

CCoE

請參閱雲端卓越中心。

CDC

請參閱變更資料擷取。

變更資料擷取 (CDC)

追蹤對資料來源 (例如資料庫表格) 的變更並記錄有關變更的中繼資料的程序。您可以將 CDC 用於各種用途，例如稽核或複寫目標系統中的變更以保持同步。

混沌工程

故意引入故障或破壞性事件來測試系統的彈性。您可以使用 AWS Fault Injection Service (AWS FIS) 執行實驗來 stress 您的 AWS 工作負載並評估其回應。

CI/CD

請參閱持續整合和持續交付。

分類

有助於產生預測的分類程序。用於分類問題的 ML 模型可預測離散值。離散值永遠彼此不同。例如，模型可能需要評估影像中是否有汽車。

用戶端加密

在目標 AWS 服務 接收資料之前，在本機加密資料。

雲端卓越中心 (CCoE)

一個多學科團隊，可推動整個組織的雲端採用工作，包括開發雲端最佳實務、調動資源、制定遷移時間表以及領導組織進行大規模轉型。如需詳細資訊，請參閱 AWS 雲端 企業策略部落格上的 CCoE 文章。

雲端運算

通常用於遠端資料儲存和 IoT 裝置管理的雲端技術。雲計算通常連接到邊緣計算技術。

雲端運作模式

在 IT 組織中，這是用來建置、成熟和最佳化一或多個雲端環境的作業模型。如需詳細資訊，請參閱建立您的雲端作業模型。

採用雲端階段

組織移轉至下列四個階段時通常會經歷 AWS 雲端：

- 專案 – 執行一些與雲端相關的專案以進行概念驗證和學習用途
- 基礎 – 進行基礎投資以擴展雲端採用 (例如，建立登陸區域、定義 CCoE、建立營運模型)
- 遷移 – 遷移個別應用程式
- 重塑 – 優化產品和服務，並在雲端中創新

這些階段是 Stephen Orban 在 AWS 雲端 企業策略部落格部落格文章「邁向雲端優先的旅程與採用階段」中所定義的。如需其與 AWS 移轉策略之間關聯的詳細資訊，請參閱移轉準備指南。

CMDB

請參閱組態管理資料庫。

程式碼儲存庫

透過版本控制程序來儲存及更新原始程式碼和其他資產 (例如文件、範例和指令碼) 的位置。常見的雲存儲庫包括 GitHub 或 AWS CodeCommit。程式碼的每個版本都稱為分支。在微服務結構中，每個儲存庫都專用於單個功能。單一 CI/CD 管道可以使用多個儲存庫。

冷快取

一種緩衝快取，它是空的、未填充的，或者包含過時或不相關的資料。這會影響效能，因為資料庫執行個體必須從主記憶體或磁碟讀取，這比從緩衝快取讀取更慢。

冷資料

很少存取且通常是歷史資料。查詢此類資料時，通常可以接受緩慢的查詢。將此資料移至效能較低且成本較低的儲存層或類別可降低成本。

計算機視覺（CV）

一個 AI 領域，它使用機器學習來分析和從數字圖像和視頻等視覺格式中提取信息。例如，提 AWS Panorama 供將 CV 添加到現場部署攝像機網絡的設備，Amazon 為 CV SageMaker 提供圖像處理算法。

配置漂移

對於工作負載，組態會從預期的狀態變更。這可能會導致工作負載變得不合規，而且通常是漸進且無意的。

組態管理資料庫 (CMDB)

儲存和管理有關資料庫及其 IT 環境的資訊的儲存庫，同時包括硬體和軟體元件及其組態。您通常在遷移的產品組合探索和分析階段使用 CMDB 中的資料。

一致性套件

AWS Config 規則和補救動作的集合，您可以組合這些動作來自訂合規性和安全性檢查。您可以使用 YAML 範本，將一致性套件部署為 AWS 帳戶 和區域中的單一實體，或跨組織部署。如需詳細資訊，請參閱文件中的AWS Config 一致性套件。

持續整合和持續交付 (CI/CD)

自動化軟體發行程序的來源、建置、測試、暫存和生產階段的程序。CI/CD 通常被描述為管道。CI/CD 可協助您將程序自動化、提升生產力、改善程式碼品質以及加快交付速度。如需詳細資訊，請參閱持續交付的優點。CD 也可表示持續部署。如需詳細資訊，請參閱持續交付與持續部署。

CV

請參閱電腦視覺。

D

靜態資料

網路中靜止的資料，例如儲存中的資料。

資料分類

根據重要性和敏感性來識別和分類網路資料的程序。它是所有網路安全風險管理策略的關鍵組成部分，因為它可以協助您確定適當的資料保護和保留控制。資料分類是 AWS Well-Architected 架構中安全性支柱的一個組成部分。如需詳細資訊，請參閱資料分類。

資料漂移

生產資料與用來訓練 ML 模型的資料之間有意義的變化，或輸入資料隨著時間的推移有意義的變化。資料漂移可降低機器學習模型預測中的整體品質、準確性和公平性。

傳輸中的資料

在您的網路中主動移動的資料，例如在網路資源之間移動。

資料網格

透過集中式管理和控管，提供分散式、分散式資料擁有權的架構架構。

資料最小化

僅收集和處理絕對必要的數據的原則。在中執行資料最小化 AWS 雲端 可降低隱私權風險、成本和分析碳足跡。

資料周長

您 AWS 環境中的一組預防性護欄，可協助確保只有受信任的身分正在存取來自預期網路的受信任資源。若要取得更多資訊，請參閱〈在上建立資料周長〉 AWS。

資料預先處理

將原始資料轉換成 ML 模型可輕鬆剖析的格式。預處理資料可能意味著移除某些欄或列，並解決遺失、不一致或重複的值。

數據來源

在整個生命週期中追蹤資料來源和歷史記錄的程序，例如資料的產生、傳輸和儲存方式。

資料主體

正在收集和處理資料的個人。

資料倉儲

支援商業智慧 (例如分析) 的資料管理系統。資料倉儲通常包含大量歷史資料，通常用於查詢和分析。

資料庫定義語言 (DDL)

用於建立或修改資料庫中資料表和物件之結構的陳述式或命令。

資料庫處理語言 (DML)

用於修改 (插入、更新和刪除) 資料庫中資訊的陳述式或命令。

DDL

請參閱資料庫定義語言。

深度整體

結合多個深度學習模型進行預測。可以使用深度整體來獲得更準確的預測或估計預測中的不確定性。

深度學習

一個機器學習子領域，它使用多層人工神經網路來識別感興趣的輸入資料與目標變數之間的對應關係。

defense-in-depth

這是一種資訊安全方法，其中一系列的安全機制和控制項會在整個電腦網路中精心分層，以保護網路和其中資料的機密性、完整性和可用性。在上採用此策略時 AWS，您可以在 AWS

Organizations 結構的不同層加入多個控制項，以協助保護資源。例如，一 defense-in-depth 種方法可能會結合多因素驗證、網路分段和加密。

委派的管理員

在中 AWS Organizations，相容的服務可以註冊成 AWS 員帳戶，以管理組織的帳戶並管理該服務的權限。此帳戶稱為該服務的委派管理員。如需詳細資訊和相容服務清單，請參閱 AWS Organizations 文件中的可搭配 AWS Organizations運作的服務。

部署

在目標環境中提供應用程式、新功能或程式碼修正的程序。部署涉及在程式碼庫中實作變更，然後在應用程式環境中建置和執行該程式碼庫。

開發環境

請參閱環境。

偵測性控制

一種安全控制，用於在事件發生後偵測、記錄和提醒。這些控制是第二道防線，提醒您注意繞過現有預防性控制的安全事件。如需詳細資訊，請參閱在 AWS上實作安全控制中的偵測性控制。

發展價值流映射

用於識別限制並排定優先順序，對軟體開發生命週期中的速度和品質產生不利影響的程序。DVSM 擴展了最初為精益生產實踐而設計的價值流映射流程。它著重於創造和通過軟件開發過程中移動價值所需的步驟和團隊。

數字雙胞胎

真實世界系統的虛擬表現法，例如建築物、工廠、工業設備或生產線。數位雙胞胎支援預測性維護、遠端監控和生產最佳化。

維度表

在 star 結構描述中，較小的資料表包含事實資料表中定量資料的相關資料屬性。維度表格屬性通常是文字欄位或離散數字，其行為類似於文字。這些屬性通常用於查詢限制、篩選和結果集標籤。

災難

防止工作負載或系統在其主要部署位置達成其業務目標的事件。這些事件可能是自然災害、技術故障或人為行為造成的結果，例如意外設定錯誤或惡意軟體攻擊。

災難復原 (DR)

您使用的策略和程序，將因災難造成的停機時間和資料遺失降到最低。如需詳細資訊，請參閱 AWS Well-Architected 的架構中的雲端中的工作負載的災難復原 AWS：雲端復原。

DML

請參閱資料庫操作語言。

領域驅動的設計

一種開發複雜軟體系統的方法，它會將其元件與每個元件所服務的不斷發展的領域或核心業務目標相關聯。Eric Evans 在其著作 Domain-Driven Design: Tackling Complexity in the Heart of Software (Boston: Addison-Wesley Professional, 2003) 中介紹了這一概念。如需有關如何將領域驅動的設計與 strangler fig 模式搭配使用的資訊，請參閱使用容器和 Amazon API Gateway 逐步現代化舊版 Microsoft ASP.NET (ASMX) Web 服務。

博士

請參閱災難復原。

漂移檢測

追蹤基線組態的偏差。例如，您可以用 AWS CloudFormation 來偵測系統資源中的漂移，也可以用 AWS Control Tower 來偵測 landing zone 中可能會影響法規遵循治理要求的變更。

DVSM

請參閱開發價值流映射。

E

EDA

請參閱探索性資料分析。

邊緣運算

提升 IoT 網路邊緣智慧型裝置運算能力的技術。與雲計算相比，邊緣計算可以減少通信延遲並縮短響應時間。

加密

一種計算過程，將純文本數據（這是人類可讀的）轉換為密文。

加密金鑰

由加密演算法產生的隨機位元的加密字串。金鑰長度可能有所不同，每個金鑰的設計都是不可預測且唯一的。

端序

位元組在電腦記憶體中的儲存順序。大端序系統首先儲存最高有效位元組。小端序系統首先儲存最低有效位元組。

端點

請參閱服務端點。

端點服務

您可以在虛擬私有雲端 (VPC) 中託管以與其他使用者共用的服務。您可以使用其他或 (IAM) 主體建立端點服務， AWS PrivateLink 並將權限授予其他 AWS 帳戶 或 AWS Identity and Access Management (IAM) 主體。這些帳戶或主體可以透過建立介面 VPC 端點私下連接至您的端點服務。如需詳細資訊，請參閱 Amazon Virtual Private Cloud (Amazon VPC) 文件中的建立端點服務。

企業資源規劃

自動化並管理企業的關鍵業務流程（例如會計、MES 和專案管理）的系統。

信封加密

使用另一個加密金鑰對某個加密金鑰進行加密的程序。如需詳細資訊，請參閱 AWS Key Management Service (AWS KMS) 文件中的信封加密。

環境

執行中應用程式的執行個體。以下是雲端運算中常見的環境類型：

- 開發環境 – 執行中應用程式的執行個體，只有負責維護應用程式的核心團隊才能使用。開發環境用來測試變更，然後再將開發環境提升到較高的環境。此類型的環境有時稱為測試環境。
- 較低的環境 – 應用程式的所有開發環境，例如用於初始建置和測試的開發環境。
- 生產環境 – 最終使用者可以存取的執行中應用程式的執行個體。在 CI/CD 管道中，生產環境是最後一個部署環境。
- 較高的環境 – 核心開發團隊以外的使用者可存取的所有環境。這可能包括生產環境、生產前環境以及用於使用者接受度測試的環境。

epic

在敏捷方法中，有助於組織工作並排定工作優先順序的功能類別。epic 提供要求和實作任務的高層級描述。例如， AWS CAF 安全史詩包括身份和訪問管理，偵探控制，基礎結構安全性，數據保護和事件響應。如需有關 AWS 遷移策略中的 Epic 的詳細資訊，請參閱計畫實作指南。

ERP

請參閱企業資源規劃。

探索性資料分析 (EDA)

分析資料集以了解其主要特性的過程。您收集或彙總資料，然後執行初步調查以尋找模式、偵測異常並檢查假設。透過計算摘要統計並建立資料可視化來執行 EDA。

F

事實表

星型架構中的中央表格。它存儲有關業務運營的定量數據。事實資料表通常包含兩種類型的資料欄：包含計量的資料欄，以及包含維度表格外部索引鍵的資料欄。

快速失敗

一種使用頻繁和增量測試來減少開發生命週期的理念。這是敏捷方法的關鍵部分。

故障隔離邊界

在中 AWS 雲端，可用區域、 AWS 區域控制平面或資料平面等界限，可限制故障的影響，並協助改善工作負載的彈性。如需詳細資訊，請參閱AWS 錯誤隔離邊界。

功能分支

請參閱分支。

特徵

用來進行預測的輸入資料。例如，在製造環境中，特徵可能是定期從製造生產線擷取的影像。

功能重要性

特徵對於模型的預測有多重要。這通常表示為可以透過各種技術來計算的數值得分，例如 Shapley Additive Explanations (SHAP) 和積分梯度。如需詳細資訊，請參閱機器學習模型可解釋性：AWS。

特徵轉換

優化 ML 程序的資料，包括使用其他來源豐富資料、調整值、或從單一資料欄位擷取多組資訊。這可讓 ML 模型從資料中受益。例如，如果將「2021-05-27 00:15:37」日期劃分為「2021」、「五月」、「週四」和「15」，則可以協助學習演算法學習與不同資料元件相關聯的細微模式。

FGAC

請參閱精細的存取控制。

精細的存取控制 (FGAC)

使用多個條件來允許或拒絕訪問請求。

閃切遷移

一種資料庫移轉方法，透過變更資料擷取使用連續資料複寫，在最短的時間內移轉資料，而不是使用階段化方法。目標是將停機時間降至最低。

G

地理阻塞

請參閱地理限制。

地理限制 (地理封鎖)

在 Amazon 中 CloudFront，防止特定國家/地區的使用者存取內容分發的選項。您可以使用允許清單或封鎖清單來指定核准和禁止的國家/地區。如需詳細資訊，請參閱 CloudFront 文件中的限制內容的地理分佈。

Gitflow 工作流程

這是一種方法，其中較低和較高環境在原始碼儲存庫中使用不同分支。Gitflow 工作流程被認為是遺留的，基於主幹的工作流程是現代的首選方法。

綠地策略

新環境中缺乏現有基礎設施。對系統架構採用綠地策略時，可以選擇所有新技術，而不會限制與現有基礎設施的相容性，也稱為棕地。如果正在擴展現有基礎設施，則可能會混合棕地和綠地策略。

防護機制

有助於跨組織單位 (OU) 來管控資源、政策和合規的高層級規則。預防性防護機制會強制執行政策，以確保符合合規標準。透過使用服務控制政策和 IAM 許可界限來將其實作。偵測性防護機制可偵測政策違規和合規問題，並產生提醒以便修正。它們是通過使用 AWS Config，Amazon AWS Security Hub GuardDuty， AWS Trusted Advisor亞馬遜檢查 Amazon Inspector 和自定義 AWS Lambda 檢查來實現的。

H

公頃

查看高可用性。

異質資料庫遷移

將來源資料庫遷移至使用不同資料庫引擎的目標資料庫 (例如，Oracle 至 Amazon Aurora)。異質遷移通常是重新架構工作的一部分，而轉換結構描述可能是一項複雜任務。AWS 提供有助於結構描述轉換的 AWS SCT。

高可用性 (HA)

工作負載在遇到挑戰或災難時持續運作的能力，無需干預。HA 系統的設計可自動容錯移轉、持續提供高品質的效能，以及處理不同的負載和故障，並將效能影響降到最低。

歷史學家現代化

一種用於現代化和升級操作技術 (OT) 系統的方法，以更好地滿足製造業的需求。歷史學家是一種類型的數據庫，用於收集和存儲工廠中的各種來源的數據。

異質資料庫遷移

將您的來源資料庫遷移至共用相同資料庫引擎的目標資料庫 (例如，Microsoft SQL Server 至 Amazon RDS for SQL Server)。同質遷移通常是主機轉換或平台轉換工作的一部分。您可以使用原生資料庫公用程式來遷移結構描述。

熱數據

經常存取的資料，例如即時資料或最近的轉譯資料。此資料通常需要高效能的儲存層或類別，才能提供快速的查詢回應。

修補程序

緊急修正生產環境中的關鍵問題。由於其緊迫性，修補程式通常是在典型的 DevOps 發行工作流程之外進行。

超級護理期間

在切換後，遷移團隊在雲端管理和監控遷移的應用程式以解決任何問題的時段。通常，此期間的長度為 1-4 天。在超級護理期間結束時，遷移團隊通常會將應用程式的責任轉移給雲端營運團隊。

I

IaC

查看基礎結構即程式碼。

身分型政策

附加至一或多個 IAM 主體的政策，用於定義其在 AWS 雲端 環境中的許可。

閒置應用程式

90 天期間 CPU 和記憶體平均使用率在 5% 至 20% 之間的應用程式。在遷移專案中，通常會淘汰這些應用程式或將其保留在內部部署。

IIoT

請參閱工業物聯網。

不可變基礎設施

為生產工作負載部署新基礎結構的模型，而不是更新、修補或修改現有基礎結構。不可變的基礎架構本質上比可變基礎架構更加一致、可靠且可預測。如需詳細資訊，請參閱 Well-Architected 的架構中的使用不可變基礎結 AWS 構進行部署最佳作法。

傳入 (輸入) VPC

在 AWS 多帳戶架構中，VPC 可接受、檢查和路由來自應用程式外部的網路連線。AWS 安全參考架構建議您使用傳入、傳出和檢查 VPC 來設定網路帳戶，以保護應用程式與更廣泛的網際網路之間的雙向介面。

增量遷移

一種切換策略，您可以在其中將應用程式分成小部分遷移，而不是執行單一、完整的切換。例如，您最初可能只將一些微服務或使用者移至新系統。確認所有項目都正常運作之後，您可以逐步移動其他微服務或使用者，直到可以解除委任舊式系統。此策略可降低與大型遷移關聯的風險。

工業 4.0

Klaus Schwab 於 2016 年推出的一個術語，指的是透過連線能力、即時資料、自動化、分析和 AI/ML 的進步實現製造流程的現代化。

基礎設施

應用程式環境中包含的所有資源和資產。

基礎設施即程式碼 (IaC)

透過一組組態檔案來佈建和管理應用程式基礎設施的程序。IaC 旨在協助您集中管理基礎設施，標準化資源並快速擴展，以便新環境可重複、可靠且一致。

工業物聯網 (IIoT)

在製造業、能源、汽車、醫療保健、生命科學和農業等產業領域使用網際網路連線的感測器和裝置。如需詳細資訊，請參閱建立工業物聯網 (IIoT) 數位轉型策略。

檢查 VPC

在 AWS 多帳戶架構中，集中式 VPC 可管理 VPC（相同或不同 AWS 區域）、網際網路和內部部署網路之間的網路流量檢查。AWS 安全參考架構建議您使用傳入、傳出和檢查 VPC 來設定網路帳戶，以保護應用程式與更廣泛的網際網路之間的雙向介面。

物聯網 (IoT)

具有內嵌式感測器或處理器的相連實體物體網路，其透過網際網路或本地通訊網路與其他裝置和系統進行通訊。如需詳細資訊，請參閱什麼是 IoT？

可解釋性

機器學習模型的一個特徵，描述了人類能夠理解模型的預測如何依賴於其輸入的程度。如需詳細資訊，請參閱 AWS 的機器學習模型可解釋性。

IoT

請參閱物聯網。

IT 資訊庫 (ITIL)

一組用於交付 IT 服務並使這些服務與業務需求保持一致的最佳實務。ITIL 為 ITSM 提供了基礎。

IT 服務管理 (ITSM)

與組織的設計、實作、管理和支援 IT 服務關聯的活動。如需有關將雲端操作與 ITSM 工具整合的資訊，請參閱操作整合指南。

ITIL

請參閱 IT 資訊庫。

ITSM

請參閱 IT 服務管理。

L

標籤式存取控制 (LBAC)

強制存取控制 (MAC) 的實作，其中每個使用者和資料本身都明確指派一個安全性標籤值。使用者安全性標籤與資料安全性標籤之間的交集決定了使用者可以看到哪些列與欄。

登陸區域

landing zone 是一個架構良好的多帳戶 AWS 環境，具有可擴展性和安全性。這是一個起點，您的組織可以從此起點快速啟動和部署工作負載與應用程式，並對其安全和基礎設施環境充滿信心。如需有關登陸區域的詳細資訊，請參閱設定安全且可擴展的多帳戶 AWS 環境。

大型遷移

遷移 300 部或更多伺服器。

LBAC

請參閱以標示為基礎的存取控制。

最低權限

授予執行任務所需之最低許可的安全最佳實務。如需詳細資訊，請參閱 IAM 文件中的套用最低權限許可。

隨即轉移

見 7 盧比

小端序系統

首先儲存最低有效位元組的系統。另請參閱 「位元順序」。

較低的環境

請參閱環境。

M

機器學習 (ML)

一種使用演算法和技術進行模式識別和學習的人工智慧。機器學習會進行分析並從記錄的資料 (例如物聯網 (IoT) 資料) 中學習，以根據模式產生統計模型。如需詳細資訊，請參閱機器學習。

主要分支

請參閱分支。

惡意軟體

旨在危及計算機安全性或隱私的軟件。惡意軟件可能會破壞計算機系統，洩漏敏感信息或獲得未經授權的訪問。惡意軟體的例子包括病毒、蠕蟲、勒索軟體、特洛伊木馬程式、間諜軟體和鍵盤記錄程式。

受管理服務

AWS 服務 用於 AWS 操作基礎架構層、作業系統和平台，並且您可以存取端點以儲存和擷取資料。Amazon Simple Storage Service (Amazon S3) 和 Amazon DynamoDB 是受管服務的範例。這些也稱為抽象服務。

製造執行系統

用於跟踪，監控，記錄和控制生產過程的軟件系統，可在現場將原材料轉換為成品。

MAP

請參閱 Migration Acceleration Program。

機制

一個完整的過程，您可以在其中創建工具，推動工具的採用，然後檢查結果以進行調整。機制是一個循環，它加強和改善自己，因為它運行。如需詳細資訊，請參閱 AWS Well-Architected 的架構中建置機制。

成員帳戶

屬於 AWS 帳戶 中組織的管理帳戶以外的所有帳戶 AWS Organizations。一個帳戶一次只能是一個組織的成員。

MES

請參閱製造執行系統。

郵件佇列遙測傳輸 (MQTT)

以發佈/訂閱模式為基礎的輕量型 machine-to-machine (M2M) 通訊協定，適用於資源受限 IoT 裝置。

微服務

一種小型的獨立服務，它可透過定義明確的 API 進行通訊，通常由小型獨立團隊擁有。例如，保險系統可能包含對應至業務能力 (例如銷售或行銷) 或子領域 (例如購買、索賠或分析) 的微服務。微服

務的優點包括靈活性、彈性擴展、輕鬆部署、可重複使用的程式碼和適應力。如需詳細資訊，請參閱使用 AWS 無伺服器服務整合微服務。

微服務架構

一種使用獨立元件來建置應用程式的方法，這些元件會以微服務形式執行每個應用程式程序。這些微服務會使用輕量型 API，透過明確定義的介面進行通訊。此架構中的每個微服務都可以進行更新、部署和擴展，以滿足應用程式特定功能的需求。如需詳細資訊，請參閱上 AWS的實作微服務。

Migration Acceleration Program (MAP)

提供諮詢支援、訓練和服務的 AWS 計畫，協助組織為移轉至雲端建立穩固的營運基礎，並協助抵消移轉的初始成本。MAP 包括用於有條不紊地執行舊式遷移的遷移方法以及一組用於自動化和加速常見遷移案例的工具。

大規模遷移

將大部分應用程式組合依波次移至雲端的程序，在每個波次中，都會以更快的速度移動更多應用程式。此階段使用從早期階段學到的最佳實務和經驗教訓來實作團隊、工具和流程的遷移工廠，以透過自動化和敏捷交付簡化工作負載的遷移。這是 AWS 遷移策略的第三階段。

遷移工廠

可透過自動化、敏捷的方法簡化工作負載遷移的跨職能團隊。移轉工廠團隊通常包括營運、業務分析師和擁有者、移轉工程師、開發人員和 DevOps 專業人員。20% 至 50% 之間的企業應用程式組合包含可透過工廠方法優化的重複模式。如需詳細資訊，請參閱此內容集中的遷移工廠的討論和雲端遷移工廠指南。

遷移中繼資料

有關完成遷移所需的應用程式和伺服器的資訊。每種遷移模式都需要一組不同的遷移中繼資料。移轉中繼資料的範例包括目標子網路、安全性群組和 AWS 帳戶。

遷移模式

可重複的遷移任務，詳細描述遷移策略、遷移目的地以及所使用的遷移應用程式或服務。範例：使 AWS 用應用程式遷移服務將遷移重新託管到 Amazon EC2。

遷移組合評定 (MPA)

這是一種線上工具，可提供驗證要移轉至的商業案例的 AWS 雲端資訊。MPA 提供詳細的組合評定 (伺服器適當規模、定價、總體擁有成本比較、遷移成本分析) 以及遷移規劃 (應用程式資料分析和資料收集、應用程式分組、遷移優先順序，以及波次規劃)。所有 AWS 顧問和 APN 合作夥伴顧問均可免費使用 MPA 工具 (需要登入)。

遷移準備程度評定 (MRA)

使用 AWS CAF 獲得有關組織雲端準備狀態、識別優勢和弱點，以及建立行動計劃以縮小已識別差距的過程。如需詳細資訊，請參閱遷移準備程度指南。MRA 是 AWS 遷移策略的第一階段。

遷移策略

將工作負載移轉至 AWS 雲端. 如需詳細資訊，請參閱本詞彙表中的 7 Rs 項目，並參閱動員您的組織以加速大規模移轉。

機器學習 (ML)

請參閱機器學習。

現代化

將過時的 (舊版或單一) 應用程式及其基礎架構轉換為雲端中靈活、富有彈性且高度可用的系統，以降低成本、提高效率並充分利用創新。如需詳細資訊，請參閱. AWS 雲端

現代化準備程度評定

這項評估可協助判斷組織應用程式的現代化準備程度；識別優點、風險和相依性；並確定組織能夠在多大程度上支援這些應用程式的未來狀態。評定的結果就是目標架構的藍圖、詳細說明現代化程序的開發階段和里程碑的路線圖、以及解決已發現的差距之行動計畫。如需詳細資訊，請參閱評估應用程式的現代化準備程度 AWS 雲端。

單一應用程式 (單一)

透過緊密結合的程序作為單一服務執行的應用程式。單一應用程式有幾個缺點。如果一個應用程式功能遇到需求激增，則必須擴展整個架構。當程式碼庫增長時，新增或改進單一應用程式的功能也會變得更加複雜。若要解決這些問題，可以使用微服務架構。如需詳細資訊，請參閱將單一體系分解為微服務。

MPA

請參閱移轉組合評估。

MQTT

請參閱佇列遙測傳輸的郵件。

多類別分類

一個有助於產生多類別預測的過程 (預測兩個以上的結果之一)。例如，機器學習模型可能會詢問「此產品是書籍、汽車還是電話？」 或者「這個客戶對哪種產品類別最感興趣？」

可變的基礎

一種模型，用於更新和修改生產工作負載的現有基礎結構。為了提高一致性，可靠性和可預測性，AWS Well-Architected 框架建議使用不可變的基礎結構作為最佳實踐。

O

OAC

請參閱原始存取控制。

OAI

請參閱原始存取身分。

OCM

請參閱組織變更管理。

離線遷移

一種遷移方法，可在遷移過程中刪除來源工作負載。此方法涉及延長停機時間，通常用於小型非關鍵工作負載。

OI

請參閱作業整合。

OLA

請參閱作業層級協定。

線上遷移

一種遷移方法，無需離線即可將來源工作負載複製到目標系統。連接至工作負載的應用程式可在遷移期間繼續運作。此方法涉及零至最短停機時間，通常用於關鍵的生產工作負載。

OPCA

請參閱開放程序通訊-統一架構。

開放程序通訊-統一架構 (OPC-UA)

用於工業自動化的 machine-to-machine (M2M) 通訊協定。OPC-UA 提供數據加密，身份驗證和授權方案的互操作性標準。

操作水準協議 (OLA)

一份協議，闡明 IT 職能群組承諾向彼此提供的內容，以支援服務水準協議 (SLA)。

操作準備程度檢討 (ORR)

問題和相關最佳做法的檢查清單，可協助您瞭解、評估、預防或減少事件和可能的故障範圍。如需詳細資訊，請參閱 AWS Well-Architected 的架構中的作業準備檢閱 (ORR)。

操作技術

可與實體環境搭配使用的硬體和軟體系統，以控制工業作業、設備和基礎設施。在製造業中，整合 OT 和資訊技術 (IT) 系統是工業 4.0 轉型的關鍵焦點。

操作整合 (OI)

在雲端中將操作現代化的程序，其中包括準備程度規劃、自動化和整合。如需詳細資訊，請參閱操作整合指南。

組織追蹤

由建立的追蹤 AWS CloudTrail 記錄中組織 AWS 帳戶 中所有人的所有事件 AWS Organizations。在屬於組織的每個 AWS 帳戶 中建立此追蹤，它會跟蹤每個帳戶中的活動。如需詳細資訊，請參閱CloudTrail文件中的為組織建立追蹤。

組織變更管理 (OCM)

用於從人員、文化和領導力層面管理重大、顛覆性業務轉型的架構。OCM 透過加速變更採用、解決過渡問題，以及推動文化和組織變更，協助組織為新系統和策略做好準備，並轉移至新系統和策略。在 AWS 移轉策略中，這個架構稱為人員加速，因為雲端採用專案所需的變更速度。如需詳細資訊，請參閱 OCM 指南。

原始存取控制 (OAC)

在中 CloudFront，限制存取權限以保護 Amazon Simple Storage Service (Amazon S3) 內容的增強選項。OAC 支援所有 S3 儲存貯體 AWS 區域、伺服器端加密 AWS KMS (SSE-KMS)，以及 S3 儲存貯體的動態PUT和DELETE請求。

原始存取身分 (OAI)

在中 CloudFront，用於限制存取以保護 Amazon S3 內容的選項。當您使用 OAI 時， CloudFront 會建立 Amazon S3 可用來進行驗證的主體。經驗證的主體只能透過特定散發存取 S3 儲存 CloudFront 貯體中的內容。另請參閱 OAC，它可提供更精細且增強的存取控制。

ORR

請參閱作業整備檢閱。

OT

請參閱操作技術。

傳出 (輸出) VPC

在 AWS 多帳戶架構中，處理從應用程式內啟動的網路連線的 VPC。AWS 安全參考架構建議您使用傳入、傳出和檢查 VPC 來設定網路帳戶，以保護應用程式與更廣泛的網際網路之間的雙向介面。

P

許可界限

附接至 IAM 主體的 IAM 管理政策，可設定使用者或角色擁有的最大許可。如需詳細資訊，請參閱 IAM 文件中的許可界限。

個人識別資訊 (PII)

直接查看或與其他相關數據配對時，可用於合理推斷個人身份的信息。PII 的範例包括姓名、地址和聯絡資訊。

PII

請參閱個人識別資訊。

手冊

一組預先定義的步驟，可擷取與遷移關聯的工作，例如在雲端中提供核心操作功能。手冊可以採用指令碼、自動化執行手冊或操作現代化環境所需的程序或步驟摘要的形式。

公司

請參閱可編程邏輯控制器

PLM

查看產品生命週期管理。

政策

可以定義權限 (請參閱以身分識別為基礎的策略)、指定存取條件 (請參閱以資源為基礎的策略) 或定義組織中所有帳戶的最大權限的物件 AWS Organizations (請參閱服務控制策略)。

混合持久性

根據資料存取模式和其他需求，獨立選擇微服務的資料儲存技術。如果您的微服務具有相同的資料儲存技術，則其可能會遇到實作挑戰或效能不佳。如果微服務使用最適合其需求的資料儲存，則可以更輕鬆地實作並達到更好的效能和可擴展性。如需詳細資訊，請參閱在微服務中啟用資料持久性。

組合評定

探索、分析應用程式組合並排定其優先順序以規劃遷移的程序。如需詳細資訊，請參閱評估遷移準備程度。

述詞

傳回`true`或的查詢條件`false`，通常位於子`WHERE`句中。

謂詞下推

一種資料庫查詢最佳化技術，可在傳輸前篩選查詢中的資料。這樣可減少必須從關聯式資料庫擷取和處理的資料量，並改善查詢效能。

預防性控制

旨在防止事件發生的安全控制。這些控制是第一道防線，可協助防止對網路的未經授權存取或不必要變更。如需詳細資訊，請參閱在 AWS上實作安全控制中的預防性控制。

委託人

中 AWS 可執行動作和存取資源的實體。此實體通常是 IAM 角色或使用者的根使用者。 AWS 帳戶如需詳細資訊，請參閱 IAM 文件中角色術語和概念中的主體。

隱私設計

一種系統工程方法，在整個工程過程中將隱私權納入考量。

私有託管區域

一種容器，它包含有關您希望 Amazon Route 53 如何回應一個或多個 VPC 內的域及其子域之 DNS 查詢的資訊。如需詳細資訊，請參閱 Route 53 文件中的使用私有託管區域。

主動控制

一種安全控制項，旨在防止部署不符合規範的資源。這些控制項會在資源佈建之前進行掃描。如果資源不符合控制項，則不會佈建該資源。如需詳細資訊，請參閱 AWS Control Tower 文件中的控制項參考指南，並參閱實作安全性控制中的主動控制 AWS。

產品生命週期管理 (PLM)

在產品的整個生命週期中管理資料和流程，從設計、開發、上市到成長與成熟度，再到下降和移除。

生產環境

請參閱環境。

可編程邏輯控制器 (PLC)

在製造業中，一台高度可靠且適應性強的計算機，可監控機器並自動化製造過程。

化名化

以預留位置值取代資料集中的個人識別碼的程序。化名化有助於保護個人隱私。假名化數據仍被認為是個人數據。

發布/訂閱 (發布/訂閱)

一種模式，可在微服務之間實現非同步通訊，以提高延展性和回應能力 例如，在微服務型 MES 中，微服務可以將事件訊息發佈到其他微服務可訂閱的通道。系統可以在不變更發佈服務的情況下新增微服務。

Q

查詢計劃

一系列步驟，如指示，用來存取 SQL 關聯式資料庫系統中的資料。

查詢計劃迴歸

在資料庫服務優化工具選擇的計畫比對資料庫環境進行指定的變更之前的計畫不太理想時。這可能因為對統計資料、限制條件、環境設定、查詢參數繫結的變更以及資料庫引擎的更新所導致。

R

拉齊矩陣

請參閱負責任，負責，諮詢，通知（RACI）。

勒索軟體

一種惡意軟體，旨在阻止對計算機系統或資料的存取，直到付款為止。

拉西矩陣

請參閱負責任，負責，諮詢，通知（RACI）。

RCAC

請參閱列與欄存取控制。

僅供讀取複本

用於唯讀用途的資料庫複本。您可以將查詢路由至僅供讀取複本以減少主資料庫的負載。

重新建築師

見 7 盧比

復原點目標 (RPO)

自上次資料復原點以來可接受的時間上限。這決定了最後一個恢復點和服務中斷之間可接受的數據丟失。

復原時間目標 (RTO)

服務中斷與恢復服務之間的最大可接受延遲。

重構

見 7 盧比

區域

地理區域中的 AWS 資源集合。每個都 AWS 區域 是隔離和獨立於其他的，以提供容錯能力，穩定性和彈性。如需詳細資訊，請參閱指定 AWS 區域 您的帳戶可以使用的項目。

迴歸

預測數值的 ML 技術。例如，為了解決「這房子會賣什麼價格？」的問題 ML 模型可以使用線性迴歸模型，根據已知的房屋事實 (例如，平方英尺) 來預測房屋的銷售價格。

重新主持

見 7 盧比

版本

在部署程序中，它是將變更提升至生產環境的動作。

重新定位

見 7 盧比

再平台

見 7 盧比

買回

見 7 盧比

彈性

應用程式抵抗或從中斷中復原的能力。在規劃備援時，高可用性和災難復原是常見的考量因素。AWS 雲端如需詳細資訊，請參閱AWS 雲端 復原力。

資源型政策

附接至資源的政策，例如 Amazon S3 儲存貯體、端點或加密金鑰。這種類型的政策會指定允許存取哪些主體、支援的動作以及必須滿足的任何其他條件。

負責者、當責者、事先諮詢者和事後告知者 (RACI) 矩陣

定義移轉活動和雲端作業所涉及之所有各方的角色與責任的矩陣。矩陣名稱衍生自矩陣中定義的責任型別：負責 (R)、負責 (A)、諮詢 (C) 及通知 (I)。支撐（S）類型是可選的。如果您包含支援，則該矩陣稱為 RASCI 矩陣，如果您將其排除，則稱為 R ACI 矩陣。

回應性控制

一種安全控制，旨在驅動不良事件或偏離安全基準的補救措施。如需詳細資訊，請參閱在 AWS上實作安全控制中的回應性控制。

保留

見 7 盧比

退休

見 7 盧比

旋轉

定期更新密碼以使攻擊者更難以存取認證的程序。

資料列與資料行存取控制 (RCAC)

使用已定義存取規則的基本、彈性 SQL 運算式。RCAC 由資料列權限和資料行遮罩所組成。

RPO

請參閱復原點目標。

RTO

請參閱復原時間目標。

執行手冊

執行特定任務所需的一組手動或自動程序。這些通常是為了簡化重複性操作或錯誤率較高的程序而建置。

S

SAML 2.0

許多身份提供者（IdPs）使用的開放標準。此功能可啟用聯合單一登入 (SSO)，因此使用者可以登入 AWS Management Console 或呼叫 AWS API 作業，而不必為組織中的每個人在 IAM 中建立使用者。如需有關以 SAML 2.0 為基礎的聯合詳細資訊，請參閱 IAM 文件中的關於以 SAML 2.0 為基礎的聯合。

斯卡達

請參閱監督控制和資料擷取。

SCP

請參閱服務控制策略。

秘密

您以加密形式儲存的機密或受限制資訊，例如密碼或使用者認證。 AWS Secrets Manager它由秘密值及其中繼資料組成。密碼值可以是二進位、單一字串或多個字串。如需詳細資訊，請參閱「Secrets Manager 碼中有什麼內容？ 在 Secrets Manager 文檔中。

安全控制

一種技術或管理防護機制，它可預防、偵測或降低威脅行為者利用安全漏洞的能力。安全性控制有四種主要類型：預防性、偵測、回應式和主動式。

安全強化

減少受攻擊面以使其更能抵抗攻擊的過程。這可能包括一些動作，例如移除不再需要的資源、實作授予最低權限的安全最佳實務、或停用組態檔案中不必要的功能。

安全資訊與事件管理 (SIEM) 系統

結合安全資訊管理 (SIM) 和安全事件管理 (SEM) 系統的工具與服務。SIEM 系統會收集、監控和分析來自伺服器、網路、裝置和其他來源的資料，以偵測威脅和安全漏洞，並產生提醒。

安全回應自動化

預先定義且程式化的動作，其設計用來自動回應或修復安全性事件。這些自動化作業可做為偵探或回應式安全控制項，協助您實作 AWS 安全性最佳實務。自動回應動作的範例包括修改 VPC 安全群組、修補 Amazon EC2 執行個體或輪換登入資料。

伺服器端加密

在其目的地的數據加密，通 AWS 服務 過接收它。

服務控制政策 (SCP)

為 AWS Organizations中的組織的所有帳戶提供集中控制許可的政策。SCP 會定義防護機制或設定管理員可委派給使用者或角色的動作限制。您可以使用 SCP 作為允許清單或拒絕清單，以指定允許或禁止哪些服務或動作。如需詳細資訊，請參閱 AWS Organizations 文件中的服務控制原則。

服務端點

的進入點的 URL AWS 服務。您可以使用端點，透過程式設計方式連接至目標服務。如需詳細資訊，請參閱 AWS 一般參考 中的 AWS 服務 端點。

服務水準協議 (SLA)

一份協議，闡明 IT 團隊承諾向客戶提供的服務，例如服務正常執行時間和效能。

服務等級指示器 (SLI)

對服務效能層面的測量，例如錯誤率、可用性或輸送量。

服務等級目標 (SLO)

代表服務狀況的目標測量結果，由服務層次指示器測量。

共同責任模式

描述您在雲端安全性和合規方面共享的責任的模型。 AWS AWS 負責雲端的安全性，而您則負責雲端的安全性。如需詳細資訊，請參閱共同責任模式。

暹

請參閱安全性資訊和事件管理系統。

單點故障 (SPF)

應用程式的單一重要元件發生故障，可能會中斷系統。

SLA

請參閱服務等級協議。

SLI

請參閱服務層級指示器。

SLO

請參閱服務等級目標。

split-and-seed 模型

擴展和加速現代化專案的模式。定義新功能和產品版本時，核心團隊會進行拆分以建立新的產品團隊。這有助於擴展組織的能力和服務，提高開發人員生產力，並支援快速創新。如需詳細資訊，請參閱中的應用程式現代化的階段化方法。 AWS 雲端

痙攣

請參閱單一故障點。

星型綱要

使用一個大型事實資料表來儲存交易或測量資料，並使用一或多個較小的維度表格來儲存資料屬性的資料庫組織結構。這種結構是專為在數據倉庫中使用或用於商業智能目的。

Strangler Fig 模式

一種現代化單一系統的方法，它會逐步重寫和取代系統功能，直到舊式系統停止使用為止。此模式源自無花果藤，它長成一棵馴化樹並最終戰勝且取代了其宿主。該模式由 Martin Fowler 引入，作為重寫單一系統時管理風險的方式。如需有關如何套用此模式的範例，請參閱使用容器和 Amazon API Gateway 逐步現代化舊版 Microsoft ASP.NET (ASMX) Web 服務。

子網

您 VPC 中的 IP 地址範圍。子網必須位於單一可用區域。

監督控制與資料擷取 (SCADA)

在製造業中，使用硬體與軟體來監控實體資產與生產作業的系統。

對稱加密

使用相同金鑰來加密及解密資料的加密演算法。

合成測試

以模擬使用者互動以偵測潛在問題或監控效能的方式測試系統。您可以使用 Amazon CloudWatch Synthetics 來創建這些測試。

T

標籤

作為組織 AWS 資源的中繼資料的索引鍵值配對。標籤可協助您管理、識別、組織、搜尋及篩選資源。如需詳細資訊，請參閱標記您的 AWS 資源。

目標變數

您嘗試在受監督的 ML 中預測的值。這也被稱為結果變數。例如，在製造設定中，目標變數可能是產品瑕疵。

任務清單

用於透過執行手冊追蹤進度的工具。任務清單包含執行手冊的概觀以及要完成的一般任務清單。對於每個一般任務，它包括所需的預估時間量、擁有者和進度。

測試環境

請參閱環境。

訓練

為 ML 模型提供資料以供學習。訓練資料必須包含正確答案。學習演算法會在訓練資料中尋找將輸入資料屬性映射至目標的模式 (您想要預測的答案)。它會輸出擷取這些模式的 ML 模型。可以使用 ML 模型，來預測您不知道的目標新資料。

傳輸閘道

可以用於互連 VPC 和內部部署網路的網路傳輸中樞。如需詳細資訊，請參閱 AWS Transit Gateway 文件中的傳輸閘道是什麼。

主幹型工作流程

這是一種方法，開發人員可在功能分支中本地建置和測試功能，然後將這些變更合併到主要分支中。然後，主要分支會依序建置到開發環境、生產前環境和生產環境中。

受信任的存取權

授與權限給您指定的服務，以代表您在組織內 AWS Organizations 及其帳戶中執行工作。受信任的服務會在需要該角色時，在每個帳戶中建立服務連結角色，以便為您執行管理工作。如需詳細資訊，請參閱 AWS Organizations 文件中的AWS Organizations 與其他 AWS 服務搭配使用。

調校

變更訓練程序的各個層面，以提高 ML 模型的準確性。例如，可以透過產生標籤集、新增標籤、然後在不同的設定下多次重複這些步驟來訓練 ML 模型，以優化模型。

雙比薩團隊

一個小 DevOps 團隊，你可以餵兩個比薩餅。雙披薩團隊規模可確保軟體開發中的最佳協作。

U

不確定性

這是一個概念，指的是不精確、不完整或未知的資訊，其可能會破壞預測性 ML 模型的可靠性。有兩種類型的不確定性：認知不確定性是由有限的、不完整的資料引起的，而隨機不確定性是由資料中固有的噪聲和隨機性引起的。如需詳細資訊，請參閱量化深度學習系統的不確定性指南。

無差別的任務

也稱為繁重工作，是創建和操作應用程序所必需的工作，但不能為最終用戶提供直接價值或提供競爭優勢。無差異化作業的範例包括採購、維護和容量規劃。

較高的環境

請參閱環境。

V

清空

一種資料庫維護操作，涉及增量更新後的清理工作，以回收儲存並提升效能。

版本控制

追蹤變更的程序和工具，例如儲存庫中原始程式碼的變更。

VPC 對等互連

兩個 VPC 之間的連線，可讓您使用私有 IP 地址路由流量。如需詳細資訊，請參閱 Amazon VPC 文件中的什麼是 VPC 對等互連。

漏洞

會危及系統安全性的軟體或硬體瑕疵。

W

暖快取

包含經常存取的目前相關資料的緩衝快取。資料庫執行個體可以從緩衝快取讀取，這比從主記憶體或磁碟讀取更快。

溫暖的數據

不常存取的資料。查詢此類資料時，通常可以接受中度緩慢的查詢。

視窗功能

一種 SQL 函數，可對以某種方式與當前記錄相關的一組行執行計算。視窗函數對於處理工作非常有用，例如計算移動平均值或根據目前列的相對位置存取列的值。

工作負載

提供商業價值的資源和程式碼集合，例如面向客戶的應用程式或後端流程。

工作串流

遷移專案中負責一組特定任務的功能群組。每個工作串流都是獨立的，但支援專案中的其他工作串流。例如，組合工作串流負責排定應用程式、波次規劃和收集遷移中繼資料的優先順序。組合工作串流將這些資產交付至遷移工作串流，然後再遷移伺服器和應用程式。

蠕蟲

看到寫一次，多讀。

WQF

請參閱 AWS 工作負載資格架構。

寫一次，多讀（WORM）

一種儲存模型，可單次寫入資料並防止資料遭到刪除或修改。授權用戶可以根據需要多次讀取數據，但無法更改數據。這種數據存儲基礎設施被認為是不可變的。

Z

零日漏洞

一種利用零時差漏洞的攻擊，通常是惡意軟件。

零時差漏洞

生產系統中未緩解的瑕疵或弱點。威脅參與者可以利用這種類型的漏洞攻擊系統。由於攻擊，開發人員經常意識到該漏洞。

殭屍應用程式

CPU 和記憶體平均使用率低於 5% 的應用程式。在遷移專案中，通常會淘汰這些應用程式。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。