在微服務中啟用資料持久性

AWS 方案指引

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS 方案指引: 在微服務中啟用資料持久性

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

Table of Contents

簡介 1
目標業務成果 2
啟用資料持久性的模式 3
Database-per-service模式 3
API 合成模式 5
CQRS 模式 7
事件來源模式 10
Amazon Kinesis Data Streams 實作 10
Amazon EventBridge 實作 11
Saga 模式 13
Shared-database-per-service 模式 15
常見問答集 17
作為現代化旅程的一部分，我何時可以現代化單體資料庫？ 17
我可以保留多個微服務的傳統單體資料庫嗎？ 17
為微服務架構設計資料庫時應考慮什麼？ 17
維持不同微服務間資料一致性的常見模式是什麼？ 17
如何維護交易自動化？ 17
我是否必須為每個微服務使用單獨的資料庫？ 17
如果微型服務的持久性資料都共用單一資料庫，如何保持其私密性？ 18
Resources 19
相關指南和模式 19
其他資源 19
文件歷史紀錄 20
..... xxi

在微服務中啟用資料持久性

Tabby Ward 和 Balaji Mohan，Amazon Web Services (AWS)

2025 年 4 月 (文件歷史記錄)

組織會持續尋求新的程序，以創造成長機會並縮短上市時間。您可以透過現代化應用程式、軟體和 IT 系統來提高組織的靈活性和效率。現代化也可協助您為客戶提供更快、更好的服務。

應用程式現代化是組織持續改進的閘道，從將單一應用程式重構為一組獨立開發、部署和管理的微服務開始。此程序包含下列步驟：

- 將整體分解為微服務 – 使用模式將整體應用程式分解為微服務。
- 整合微服務 – 使用 Amazon Web Services (AWS) 無伺服器服務，將新建立的微服務整合到微服務架構中。
- 啟用微服務架構的資料持久性 – 透過分散資料存放區來提升微服務之間的多槽持久性。

雖然您可以在某些使用案例中使用單體應用程式架構，但現代應用程式功能通常無法在單體架構中運作。例如，當您升級個別元件時，整個應用程式無法保持可用，而且您無法擴展個別元件來解決瓶頸或熱點（應用程式資料中相對密集的區域）。單體可能會變成大型、無法管理的應用程式，而多個團隊之間需要大量努力和協調，才能引進微小的變更。

傳統應用程式通常使用集中式單體資料庫，這使得結構描述變更變得困難，建立具有垂直擴展的技術鎖定作為回應增長的唯一方法，並強加單一失敗點。單體資料庫也可讓您建置實作微服務架構所需的分散式獨立元件。

先前，典型的架構方法是在單一關聯式資料庫中建立所有使用者需求的模型，供單體應用程式使用。此方法受到熱門關聯式資料庫架構的支援，應用程式架構師通常會在開發程序的最早階段設計關聯式結構描述、建置高度標準化的結構描述，然後將其傳送給開發人員團隊。不過，這表示資料庫驅動應用程式使用案例的資料模型，而不是反之亦然。

透過選擇分散資料存放區，您可以提升微服務之間的多槽持久性，並根據資料存取模式和微服務的其他需求來識別資料儲存技術。每個微服務都有自己的資料存放區，可以透過低影響的結構描述變更獨立擴展，並透過微服務的 API 封鎖資料。分解單體資料庫並不容易，而最大的挑戰之一是建構資料以獲得最佳效能。分散式多槽持久性通常也會導致最終的資料一致性，而需要徹底評估的其他潛在挑戰包括交易期間的資料同步、交易完整性、資料重複，以及聯結和延遲。

本指南適用於應用程式擁有者、企業擁有者、架構師、技術主管和專案經理。本指南提供以下六個模式，以啟用微服務之間的資料持久性：

- Database-per-service模式
- API 合成模式
- CQRS 模式
- 事件來源模式
- Saga 模式
  - 如需使用 實作saga模式的步驟 AWS Step Functions，請參閱 AWS 規範指引網站上的使用 實作無伺服器saga模式 AWS Step Functions。
- Shared-database-per-service 模式

本指南是內容系列的一部分，涵蓋 建議的應用程式現代化方法 AWS。系列也包含：

- AWS 雲端中應用程式現代化策略
- 將 AWS 雲端中的應用程式現代化的分階段方法
- 評估 AWS 雲端中應用程式的現代化準備程度
- 將整體分解為微服務
- 使用無 AWS 伺服器服務整合微服務

目標業務成果

許多組織發現，創新和改善使用者體驗會受到整體應用程式、資料庫和技術的負面影響。舊版應用程式和資料庫可減少您採用現代技術架構的選項，並限制您的競爭和創新。不過，當您將應用程式及其資料存放區現代化時，它們會更容易擴展並加快開發速度。解耦資料策略可改善容錯能力和彈性，有助於加速新應用程式功能的上市時間。

您應該預期在微服務之間提升資料持久性的六個結果：

- 從應用程式產品組合中移除舊版單體資料庫。
- 改善應用程式的容錯能力、彈性和可用性。
- 縮短新應用程式功能的上市時間。
- 降低整體授權費用和營運成本。
- 利用開放原始碼解決方案（例如 MySQL 或 PostgreSQL)。
- 在 AWS 雲端上從超過 15 個專用資料庫引擎中選擇，以建置高度可擴展且分散式的應用程式。

啟用資料持久性的模式

下列模式用於在微服務中啟用資料持久性。

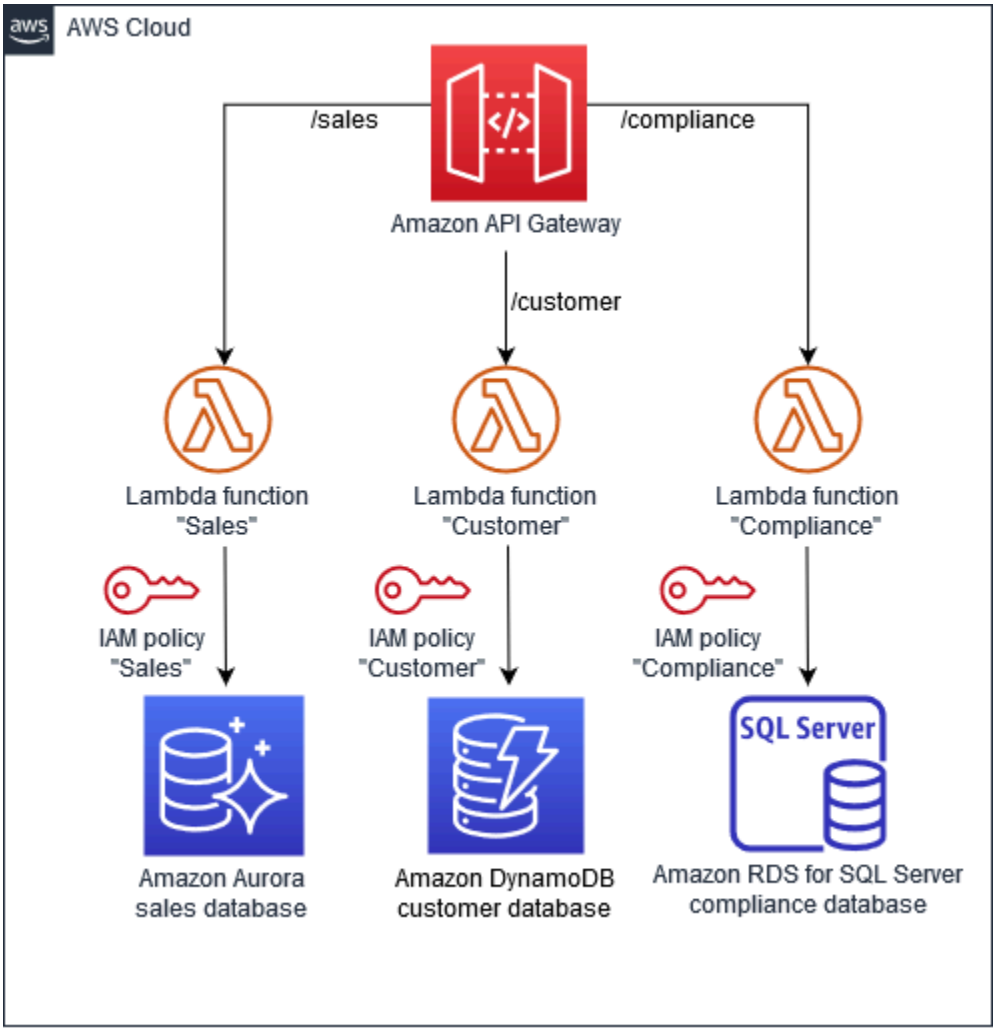
主題

- Database-per-service模式
- API 合成模式
- CQRS 模式
- 事件來源模式
- Saga 模式
- Shared-database-per-service 模式

Database-per-service模式

鬆耦合是微服務架構的核心特性，因為每個微服務都可以獨立儲存並從自己的資料存放區擷取資訊。透過部署database-per-service模式，您可以為您的應用程式和業務需求選擇最適當的資料存放區（例如關聯式或非關聯式資料庫）。這表示微服務不會共用資料層、微服務個別資料庫的變更不會影響其他微服務、其他微服務無法直接存取個別資料存放區，而且持久性資料只能由 APIs存取。解耦資料存放區也會改善整體應用程式的彈性，並確保單一資料庫無法成為單一故障點。

在下圖中，「銷售」、「客戶」和「合規」微服務會使用不同的 AWS 資料庫。這些微服務會部署為 AWS Lambda 函數，並透過 Amazon API Gateway API 存取。 AWS Identity and Access Management (IAM) 政策可確保資料保持私有，而不會在微服務之間共用。每個微服務都使用符合其個別需求的資料庫類型；例如，「銷售」使用 Amazon Aurora，「客戶」使用 Amazon DynamoDB，而「合規」使用 Amazon Relational Database Service (Amazon RDS) for SQL Server。



在以下情況下，您應該考慮使用此模式：

- 微服務之間需要鬆散耦合。
- 微服務對其資料庫有不同的合規或安全要求。
- 需要更精細地控制擴展。

使用database-per-service資料庫模式有以下缺點：

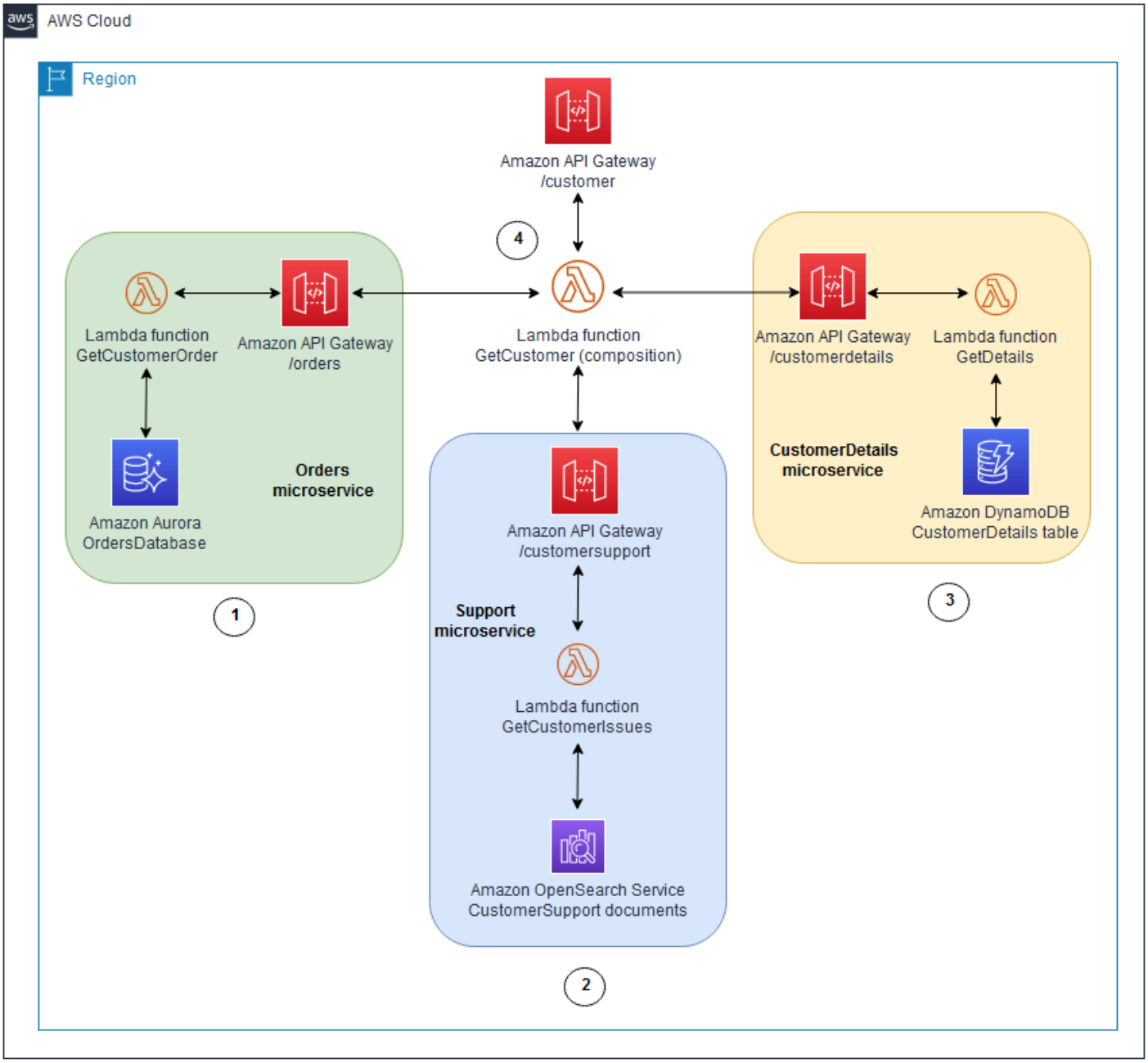
- 實作橫跨多個微服務或資料存放區的複雜交易和查詢可能具有挑戰性。
- 您必須管理多個關聯式和非關聯式資料庫。
- 您的資料存放區必須符合兩個 CAP 理論要求：一致性、可用性或分割區公差。

> **Note**
>
> 如果您使用 database-per-service模式，則必須部署另一個模式，以實作跨越多個微服務的查詢。您可以使用 API 合成模式（您可以使用 加快速度CQRS 模式) 或事件來源模式來建立彙總結果。

API 合成模式

此模式使用 API 編譯器或彙總器，透過叫用擁有資料的個別微服務來實作查詢。然後，它會透過執行記憶體內聯結來合併結果。

下圖說明此模式的實作方式。



該圖顯示以下工作流程：

1. API 閘道提供「/客戶」 API，其具有可追蹤 Aurora 資料庫中客戶訂單的「訂單」微服務。
2. 「支援」微服務會追蹤客戶支援問題，並將其存放在 Amazon OpenSearch Service 資料庫中。
3. 「CustomerDetails」微服務會在 DynamoDB 資料表中維護客戶屬性（例如地址、電話號碼或付款詳細資訊）。

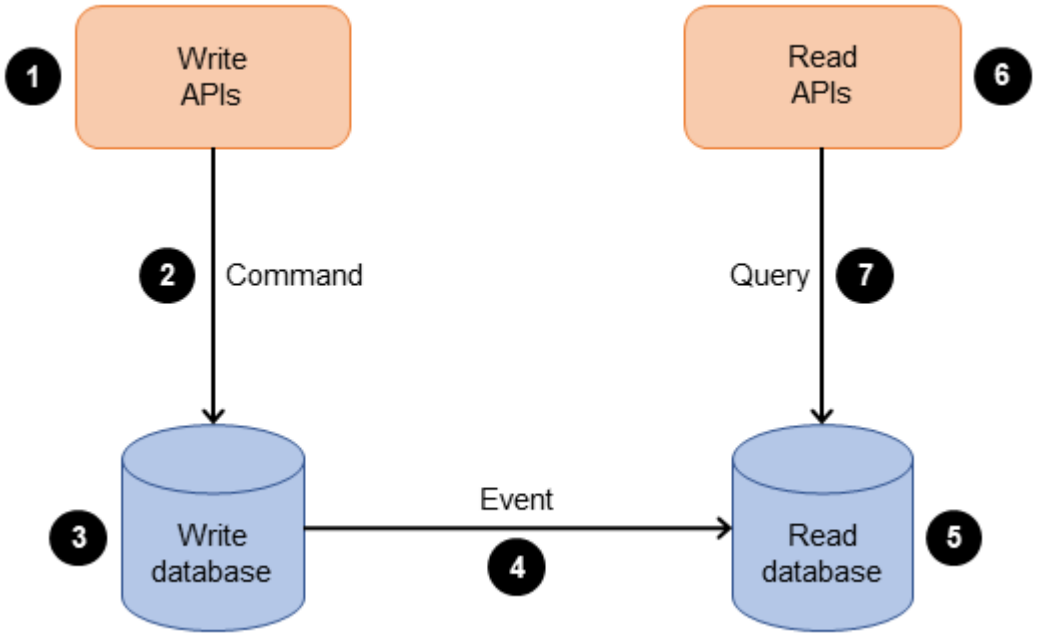
4. 「GetCustomer」Lambda 函數會執行這些微服務的 APIs，並在將資料傳回請求者之前，對資料執行記憶體內聯結。這有助於在使用者面向的 API 的一次網路呼叫中輕鬆擷取客戶資訊，並保持非常簡單的界面。

API 合成模式提供從多個微服務收集資料的最簡單方法。不過，使用 API 合成模式有以下缺點：

- 它可能不適合需要記憶體內聯結的複雜查詢和大型資料集。
- 如果您增加連線至 API 編寫器的微服務數量，您的整體系統會變得較不可用。
- 增加的資料庫請求會建立更多的網路流量，進而增加您的營運成本。

CQRS 模式

命令查詢責任隔離 (CQRS) 模式會將資料變動或系統的命令部分與查詢部分分開。如果更新和查詢對輸送量、延遲或一致性的需求不同，您可以使用 CQRS 模式來分隔更新和查詢。CQRS 模式會將應用程式分成兩個部分：命令端和查詢端，如下圖所示。命令端會處理 `create`、 `update`和 `delete`請求。查詢端會使用僅供讀取複本執行該`query`部分。



圖表顯示下列程序：

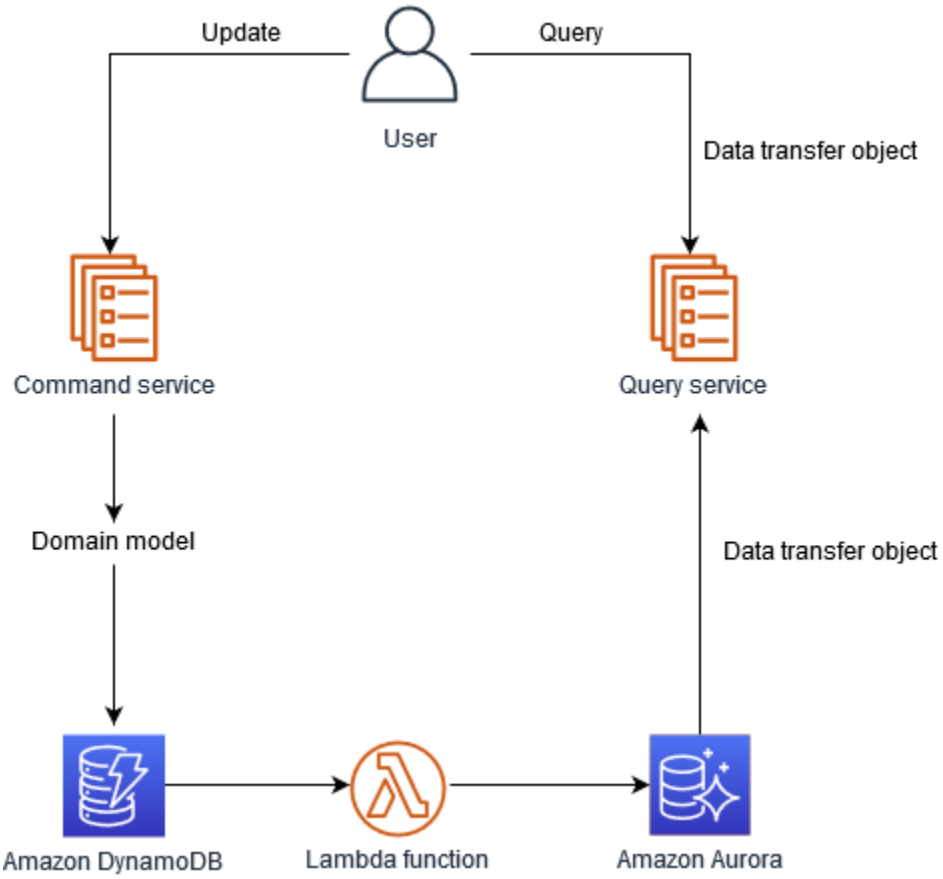
1. 企業會透過 API 傳送命令來與應用程式互動。命令是建立、更新或刪除資料等動作。
2. 應用程式會在命令端處理傳入的命令。這涉及驗證、授權和執行 操作。

3. 應用程式會將命令的資料保留在寫入（命令）資料庫中。
4. 在寫入資料庫中存放命令之後，會觸發事件來更新讀取（查詢）資料庫中的資料。
5. 讀取（查詢）資料庫會處理並保留資料。讀取資料庫旨在針對特定查詢需求進行最佳化。
6. 企業會與讀取 APIs 互動，將查詢傳送至應用程式的查詢端。
7. 應用程式會在查詢端處理傳入的查詢，並從讀取資料庫擷取資料。

您可以使用各種資料庫組合來實作 CQRS 模式，包括：

- 針對 命令和查詢端使用關聯式資料庫管理系統 (RDBMS) 資料庫。寫入操作會移至主要資料庫，而讀取操作可以路由至僅供讀取複本。範例：Amazon RDS 僅供讀取複本
- 對命令端使用 RDBMS 資料庫，對查詢端使用 NoSQL 資料庫。範例：使用事件來源和搭配 的 CQRS 來現代化舊版資料庫 AWS DMS
- 對命令和查詢端使用 NoSQL 資料庫。範例：使用 Amazon DynamoDB 建置 CQRS 事件存放區
- 將 NoSQL 資料庫用於命令端，將 RDBMS 資料庫用於查詢端，如下列範例所述。

在下圖中，DynamoDB 等 NoSQL 資料存放區用於最佳化寫入輸送量並提供靈活的查詢功能。當您新增資料時，這可在具有明確定義存取模式的工作負載上實現高寫入可擴展性。Amazon Aurora 等關聯式資料庫提供複雜的查詢功能。DynamoDB 串流會將資料傳送至更新 Aurora 資料表的 Lambda 函數。



使用 DynamoDB 和 Aurora 實作 CQRS 模式可提供下列主要優點：

- DynamoDB 是全受管 NoSQL 資料庫，可處理大量寫入操作，而 Aurora 可為查詢端的複雜查詢提供高讀取可擴展性。
- DynamoDB 提供低延遲、高輸送量的資料存取，因此非常適合處理命令和更新操作，而且 Aurora 效能可以針對複雜的查詢進行微調和最佳化。
- DynamoDB 和 Aurora 都提供無伺服器選項，讓您的企業僅根據用量支付資源費用。
- DynamoDB 和 Aurora 是全受管服務，可減少管理資料庫、備份和可擴展性的操作負擔。

在以下情況下，您應該考慮使用 CQRS 模式：

- 您已實作database-per-service模式，並想要聯結來自多個微服務的資料。
- 您的讀取和寫入工作負載有不同的擴展、延遲和一致性需求。
- 讀取查詢可接受最終一致性。

> **Important**
>
> CQRS 模式通常會導致資料存放區之間的最終一致性。

事件來源模式

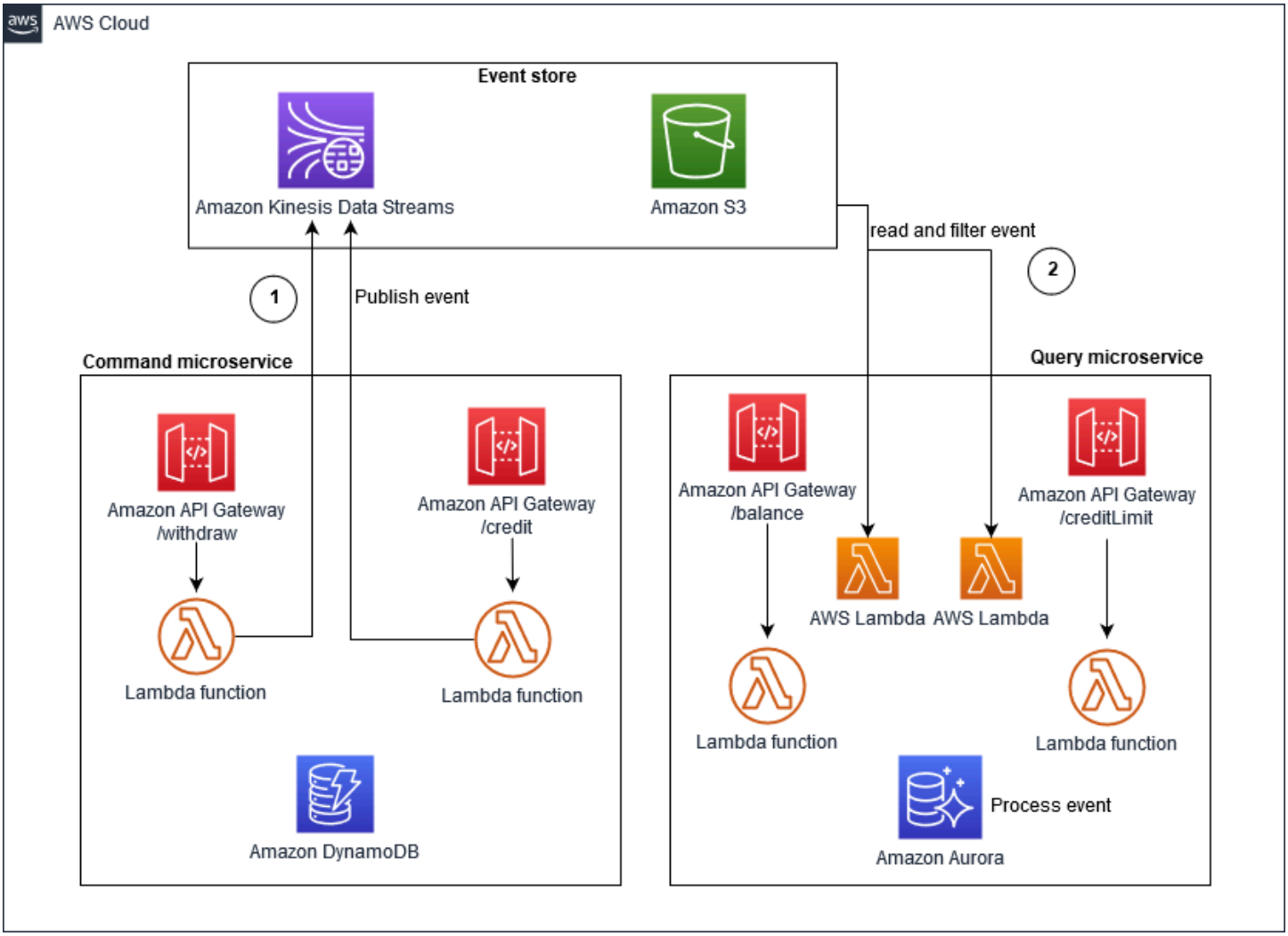
事件來源模式通常與 搭配使用，CQRS 模式以分離寫入工作負載的讀取，並最佳化效能、可擴展性和安全性。資料會儲存為一系列事件，而不是直接更新資料存放區。Microservices 從事件存放區重播事件，以計算其資料存放區的適當狀態。模式提供應用程式目前狀態的可見性，以及應用程式如何到達該狀態的其他內容。事件來源模式可有效地與 CQRS 模式搭配使用，因為即使命令和查詢資料存放區具有不同的結構描述，也可以針對特定事件重現資料。

透過選擇此模式，您可以識別和重建任何時間點的應用程式狀態。這會產生持久性稽核線索，並使偵錯更容易。不過，資料最終會保持一致，這可能不適用於某些使用案例。

您可以使用 Amazon Kinesis Data Streams 或 Amazon EventBridge 來實作此模式。

Amazon Kinesis Data Streams 實作

在下圖中，Kinesis Data Streams 是集中式事件存放區的主要元件。事件存放區會將應用程式變更擷取為事件，並將其保留在 Amazon Simple Storage Service (Amazon S3) 上。

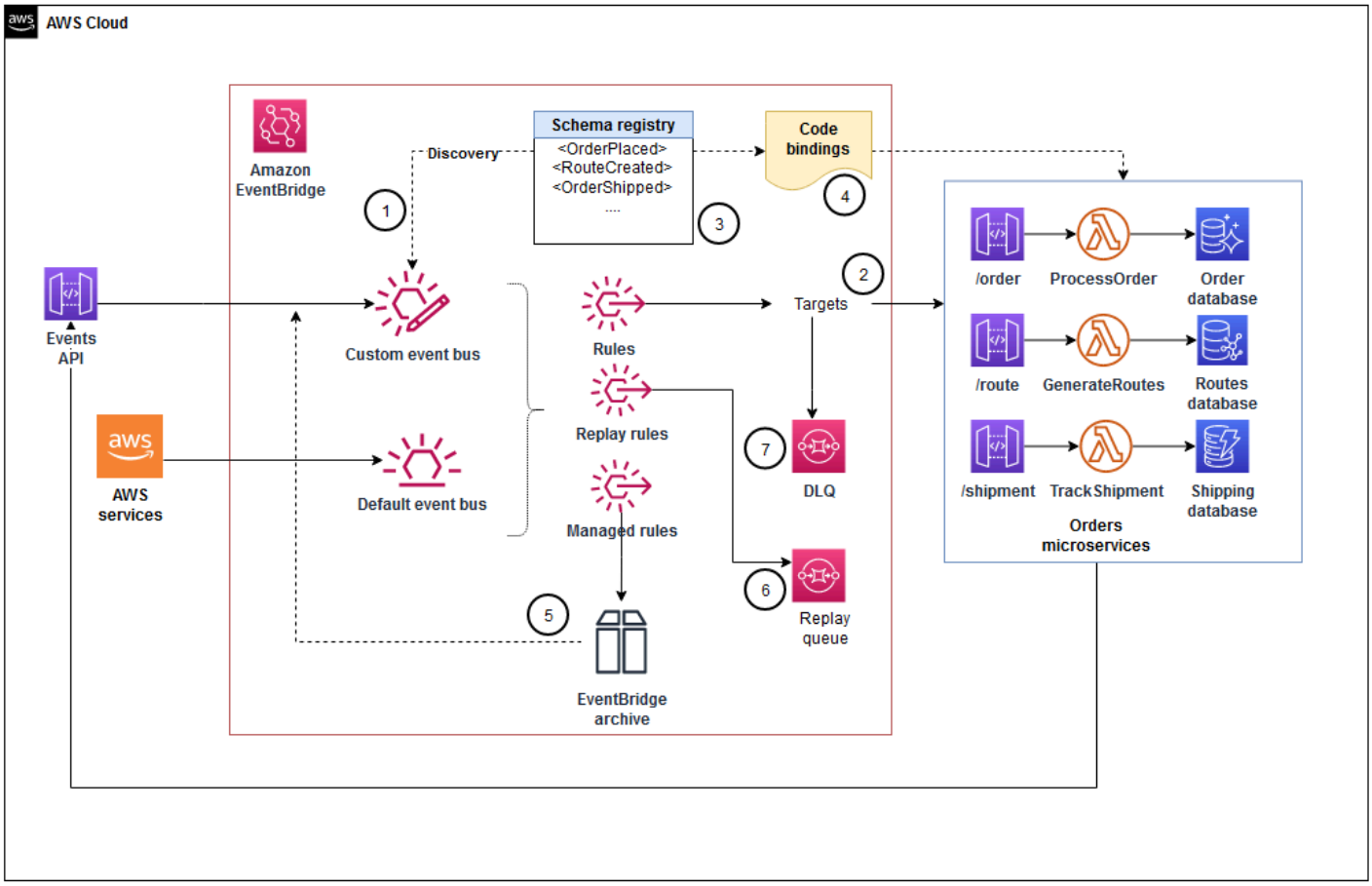


工作流程由以下步驟組成：

1. 當「/撤回」或「/額度」微服務發生事件狀態變更時，他們會透過將訊息寫入 Kinesis Data Streams 來發佈事件。
2. 其他微服務，例如「/balance」或「/creditLimit」，讀取訊息的副本，篩選其相關性，然後將其轉送以進行進一步處理。

Amazon EventBridge 實作

下圖中的架構使用 EventBridge。EventBridge 是一種無伺服器服務，使用事件來連接應用程式元件，讓您更輕鬆地建置可擴展、事件驅動的應用程式。事件驅動型架構是一種透過發出和回應事件來建置鬆耦合軟體系統的方式。EventBridge 為 AWS 服務發佈的事件提供預設事件匯流排，您也可以為網域特定的匯流排建立自訂事件匯流排。



工作流程由以下步驟組成：

1. 「OrderPlaced」事件由「Orders」微服務發佈至自訂事件匯流排。
2. 在下訂單後需要採取動作的微服務，例如「/路由」微服務，是由規則和目標啟動。
3. 這些微服務會產生路由，將訂單運送給客戶並發出「RouteCreated」事件。
4. 需要採取進一步動作的微服務也會由「RouteCreated」事件啟動。
5. 事件會傳送至事件封存（例如 EventBridge 封存），以便在需要時重新播放以供重新處理。
6. 如有需要，歷史訂單事件會傳送至新的 Amazon SQS 佇列（重播佇列）進行重新處理。
7. 如果未啟動目標，受影響的事件會放置在無效字母佇列 (DLQ) 中，以供進一步分析和重新處理。

在以下情況下，您應該考慮使用此模式：

- 事件用於完全重建應用程式的狀態。
- 您需要在系統中重播事件，並且可以隨時判斷應用程式的狀態。

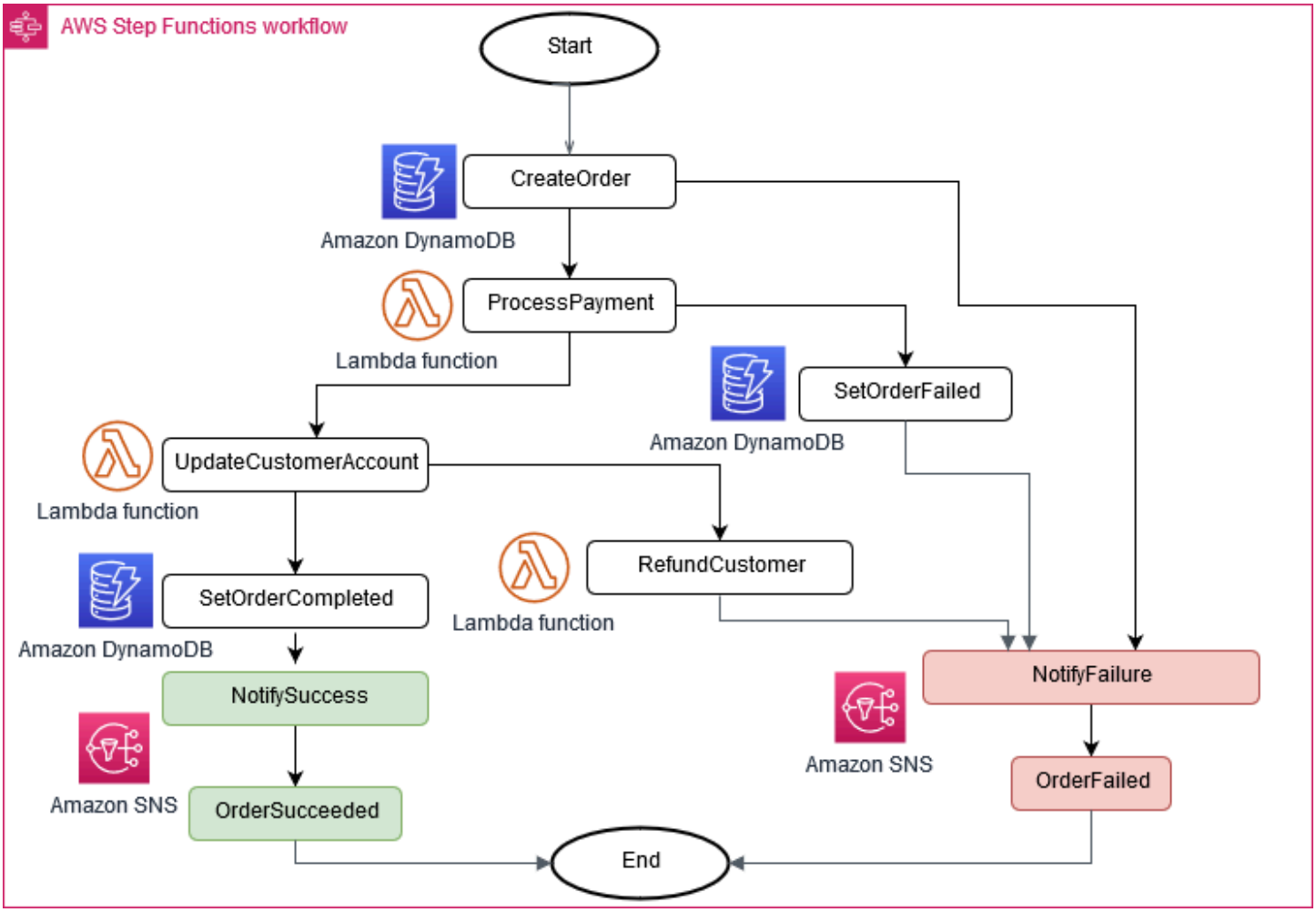
- 您想要能夠反轉特定事件，而不必從空白應用程式狀態開始。
- 您的系統需要可輕易序列化的事件串流，才能建立自動化日誌。
- 您的系統需要繁重的讀取操作，但寫入操作很輕；繁重的讀取操作可以導向至記憶體內資料庫，該資料庫會隨著事件串流保持更新。

> **Important**
>
> 如果您使用事件來源模式，則必須部署 Saga 模式以維持跨微服務的資料一致性。

Saga 模式

saga 模式是一種故障管理模式，可協助在分散式應用程式中建立一致性，並協調多個微服務之間的交易，以維持資料一致性。微服務會為每個交易發佈事件，並根據事件的結果啟動下一個交易。它可能需要兩個不同的路徑，取決於交易的成功或失敗。

下圖顯示 saga 模式如何使用 實作訂單處理系統 AWS Step Functions。每個步驟（例如，「ProcessPayment」）也有單獨的步驟來處理程序的成功（例如，「UpdateCustomerAccount」）或失敗（例如，「SetOrderFailure」）。



在以下情況下，您應該考慮使用此模式：

- 應用程式需要維持多個微服務之間的資料一致性，而無需緊密耦合。
- 有長期交易，而且如果某個微服務長時間執行，您不希望其他微服務遭到封鎖。
- 如果 操作在序列中失敗，您需要能夠復原。

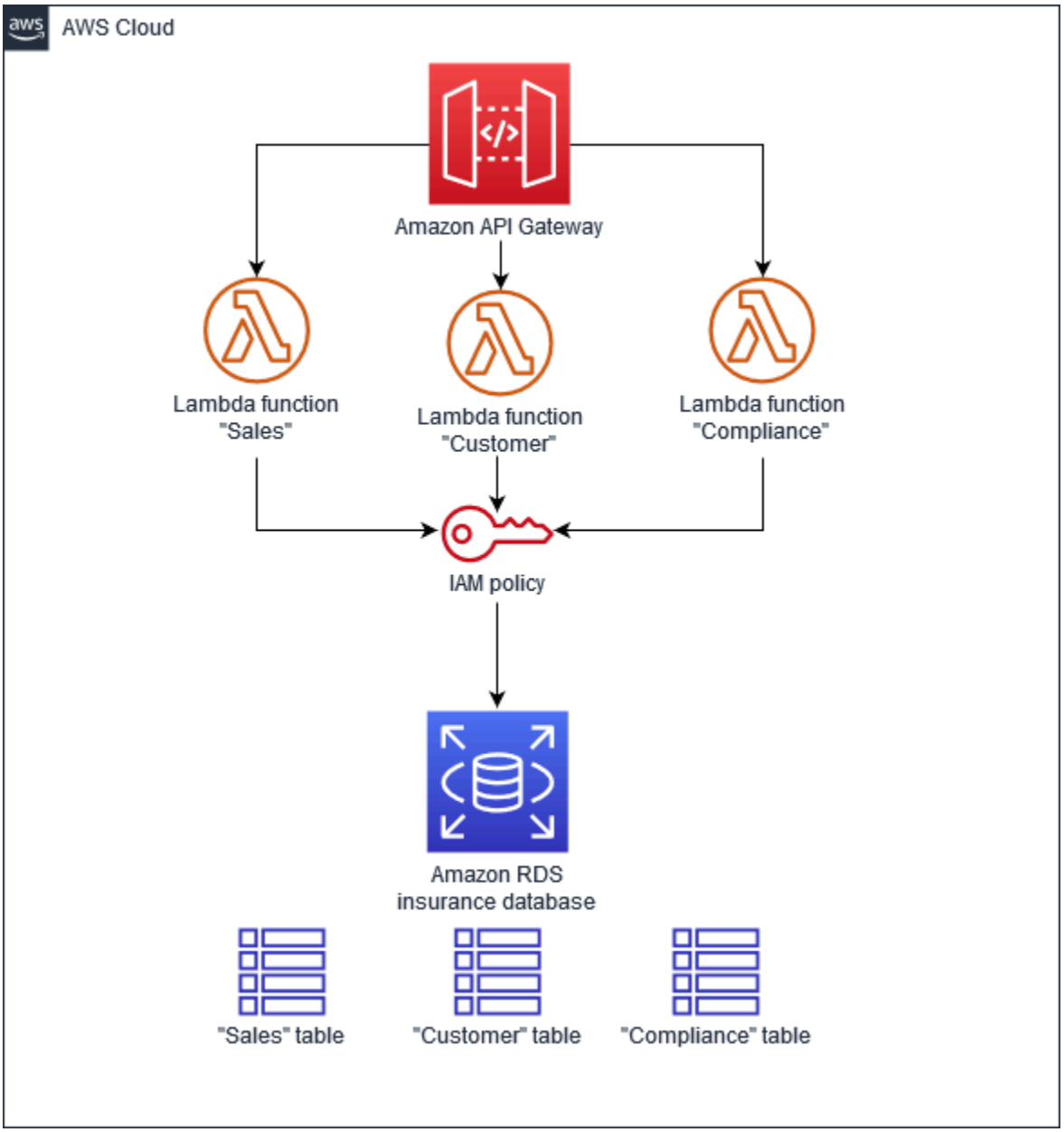
> **Important**
>
> 模式saga很難除錯，而且其複雜性會隨著微服務的數量而增加。模式需要複雜的程式設計模型，開發和設計補償交易以復原和復原變更。

如需在微服務架構中實作saga模式的詳細資訊，請參閱 AWS 規範指引網站上的 模式使用 實作無伺服器saga模式 AWS Step Functions。

Shared-database-per-service 模式

在 shared-database-per-service 模式中，數個微服務共用相同的資料庫。您需要在採用此模式之前仔細評估應用程式架構，並確定避免熱資料表（多個微服務寫入的單一資料表）。所有資料庫變更也必須回溯相容；例如，開發人員只有在所有微服務目前和先前版本未參考物件時，才能捨棄資料欄或資料表。

在下圖中，所有微服務都會共用保險資料庫，而 IAM 政策會提供資料庫的存取權。這會建立開發時間耦合；例如，「銷售」微服務的變更需要與「客戶」微服務協調結構描述變更。此模式不會減少開發團隊之間的相依性，並引入執行期耦合，因為所有微服務共用相同的資料庫。例如，長時間執行的「銷售」交易可以鎖定「客戶」資料表，這會封鎖「客戶」交易。



在以下情況下，您應該考慮使用此模式：

- 您不希望對現有程式碼庫進行過多重構。
- 您可以使用提供原子性、一致性、隔離和耐久性 (ACID) 的交易來強制執行資料一致性。
- 您想要僅維護和操作一個資料庫。
- 由於現有微database-per-service模式相當困難。
- 您不想完全重新設計現有的資料層。

常見問答集

本節提供有關在微服務中啟用資料持久性的常見問題解答。

作為現代化旅程的一部分，我何時可以現代化單體資料庫？

當您開始將單體應用程式分解為微服務時，您應該專注於現代化單體資料庫。請務必建立策略，將資料庫分割成多個與您的應用程式一致的小型資料庫。

我可以保留多個微服務的傳統單體資料庫嗎？

保留多個微服務的共用整體資料庫會產生緊密耦合，這表示您無法獨立部署微服務的變更，而且必須在微服務之間協調所有結構描述變更。雖然您可以使用關聯式資料存放區做為單體資料庫，但 NoSQL 資料庫可能是部分微服務的最佳選擇。

為微服務架構設計資料庫時應考慮什麼？

您應該根據符合應用程式功能的網域來設計應用程式。請務必評估應用程式的功能，並決定是否需要關聯式資料庫結構描述。如果 NoSQL 資料庫符合您的需求，您也應考慮使用它。

維持不同微服務間資料一致性的常見模式是什麼？

最常見的模式是使用事件驅動型架構。

如何維護交易自動化？

在微服務架構中，交易包含由不同微服務處理的多個本機交易。如果本機交易失敗，您需要轉返先前已完成的成功交易。您可以使用 Saga 模式來避免這種情況。

我是否必須為每個微服務使用單獨的資料庫？

微服務架構的主要優勢是鬆散耦合。每個微服務的持久性資料都必須保持私有，並且只能透過微服務的 API 存取。如果您的微服務共用相同的資料庫，則必須仔細評估資料結構描述的變更。

如果微型服務的持久性資料都共用單一資料庫，如何保持其私密性？

如果您的微服務共用關聯式資料庫，請確定每個微服務都有私有資料表。您也可以建立個別微服務私有的個別結構描述。

Resources

相關指南和模式

- AWS 雲端中應用程式現代化策略
- 將 AWS 雲端中的應用程式現代化的分階段方法
- 評估 AWS 雲端中應用程式的現代化準備程度
- 將整體分解為微服務
- 使用無 AWS 伺服器服務整合微服務
- 使用 實作無伺服器saga模式 AWS Step Functions

其他資源

- 使用 進行應用程式現代化 AWS
- 建置高可用性的微服務，為任何大小和規模的應用程式提供支援
- 使用 進行雲端原生應用程式現代化 AWS
- 成本最佳化和創新：應用程式現代化簡介
- 開發人員指南：使用微服務擴展
- 分散式資料管理 – Saga 模式
- 使用 實作微服務架構 AWS 服務：命令查詢責任隔離模式
- 使用 實作微服務架構 AWS 服務：事件來源模式
- 現代應用程式：透過應用程式設計建立價值
- 現代化您的應用程式、推動成長並降低 TCO

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
釐清	我們釐清了database-per-service和shared-database-per-service模式的一些資訊。	2025 年 4 月 7 日
更新模式	我們已更新事件來源模式的 Amazon EventBridge 實作區段。	2023 年 12 月 4 日
展開的區段	我們更新了 CQRS 模式，並提供更多資訊。	2023 年 11 月 17 日
新增使用 Step Functions 實作saga模式的連結	我們更新了主目錄和Saga模式區段，其中包含模式的連結使用 實作無伺服器saga模式 AWS Step Functions，方法為從 AWS 規範指引網站。	2021 年 2 月 23 日
初次出版	—	2021 年 1 月 27 日

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。