AWS 白皮書

在 上實作微服務 AWS

在 上實作微服務 AWS: AWS 白皮書

Table of Contents

在 上實作微服務 AWS

發佈日期：2023 年 7 月 31 日 (文件歷史記錄)

Microservices 提供簡化的軟體開發方法，可加速部署、鼓勵創新、增強可維護性，並提高可擴展性。此方法依賴小型、鬆散的耦合服務，這些服務會透過由自主團隊管理的明確定義 APIs 進行通訊。採用微服務可提供優勢，例如改善可擴展性、彈性、彈性和更快的開發週期。

此白皮書探索三種熱門的微服務模式：API 驅動、事件驅動和資料串流。我們提供每種方法的概觀、概述微服務的主要功能、解決其開發中的挑戰，並說明 Amazon Web Services (AWS) 如何協助應用程式團隊解決這些障礙。

考量資料存放區、非同步通訊和服務探索等主題的複雜性質，建議您在進行架構決策時，將應用程式的特定需求和使用案例與提供的指南一起權衡。

簡介

Microservices 架構結合來自各種欄位的成功和經過驗證的概念，例如：

- 敏捷的軟體開發
- 服務導向架構
- API 優先設計
- 持續整合/持續交付 (CI/CD)

通常，微服務會整合十二要素應用程式的設計模式。

雖然微服務提供許多好處，但評估使用案例的獨特需求和相關成本至關重要。在某些情況下，單體架構或替代方法可能更合適。微服務或整體之間的決策應依case-by-case進行，並考量規模、複雜性和特定使用案例等因素。

我們首先探索高度可擴展、容錯的微服務架構（使用者介面、微服務實作、資料存放區），並示範如何使用 AWS 容器技術在 上建置它。然後，我們建議使用 AWS 服務來實作典型的無伺服器微服務架構，從而降低操作複雜性。

Serverless 的特性如下原則：

- 沒有要佈建或管理的基礎設施

- 依消耗單位自動擴展
- 「支付價值」帳單模型
- 內建可用性和容錯能力
- 事件驅動架構 (EDA)

最後，我們會檢查整體系統並討論微服務架構的跨服務層面，例如分散式監控、記錄、追蹤、稽核、資料一致性和非同步通訊。

本文件著重於在 中執行的工作負載 AWS 雲端，不包括混合式案例和遷移策略。如需遷移策略的相關資訊，請參閱容器遷移方法白皮書。

您是 Well-Architected 嗎？

AWS Well-Architected Framework 可協助您了解在雲端建置系統時所做決策的優缺點。架構的六個支柱可讓您了解架構最佳實務，以設計和操作可靠、安全、高效、經濟實惠且永續的系統。使用 AWS Well-Architected Tool免費提供的 AWS 管理主控台，您可以透過回答每個支柱的一組問題，根據這些最佳實務來檢閱工作負載。

在 Serverless Application Lens 中，我們專注於建構無伺服器應用程式的最佳實務 AWS。

如需雲端架構的更多專家指導和最佳實務，請參閱AWS 架構中心，參考架構部署、圖表和白皮書。

將 現代化為微服務

Microservices 基本上是組成應用程式的小型獨立單位。從傳統單體結構轉換到微服務可以遵循各種策略。

此轉換也會影響您的組織運作方式：

- 它鼓勵敏捷的開發，其中團隊以快速週期運作。
- 團隊通常很小，有時被描述為兩個比薩團隊，足夠小到兩個比薩可以為整個團隊提供食物。
- 從建立到部署和維護，團隊對其服務負完全責任。

上的簡易微服務架構 AWS

典型的單體應用程式 由不同的圖層組成：呈現圖層、應用程式圖層和資料圖層。另一方面， 微服務架構會根據特定網域，而不是技術圖層，將功能分隔成具有凝聚力的垂直。圖 1 說明典型微服務應用程式的參考架構 AWS。

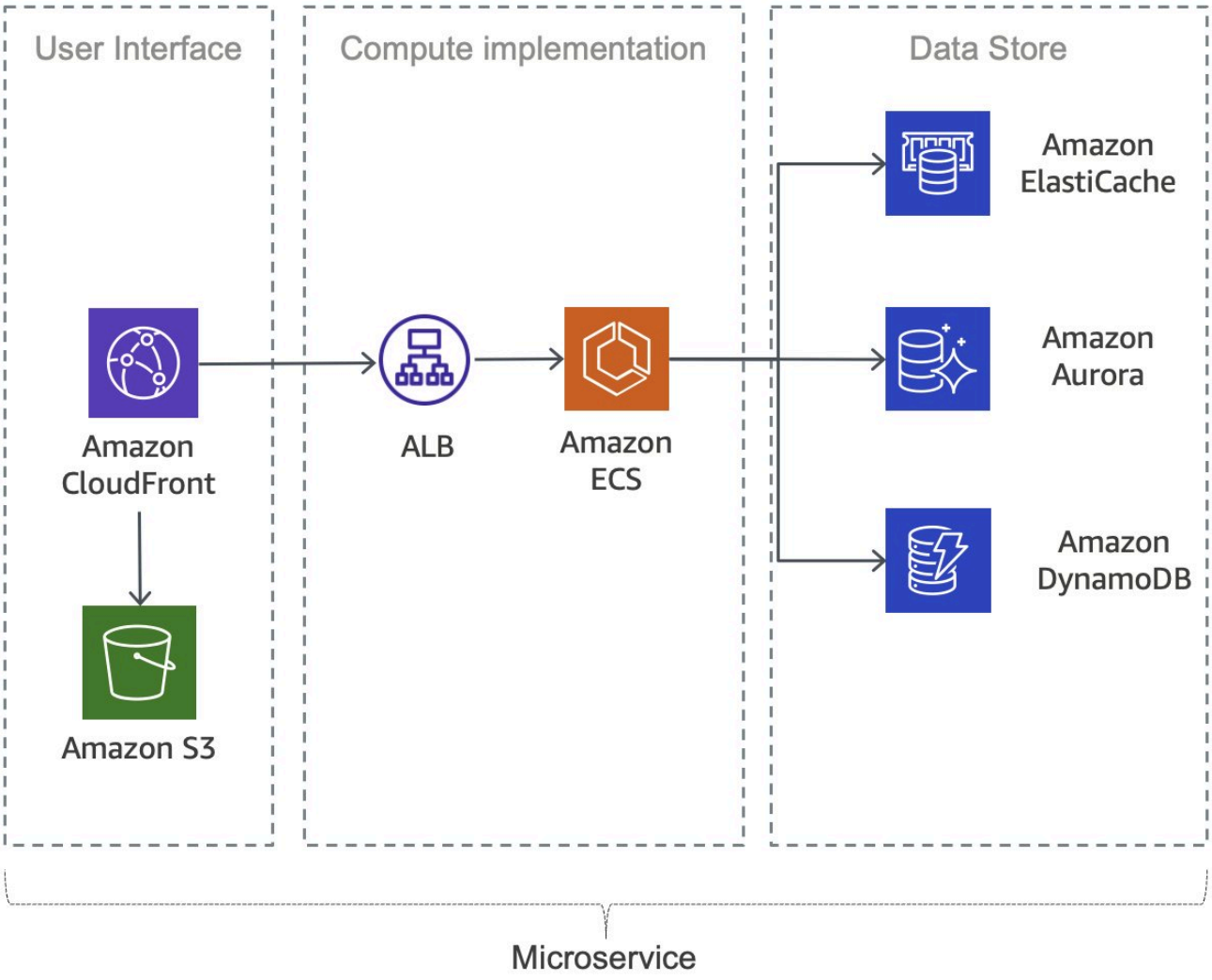

圖 1： 上的典型微服務應用程式 AWS

使用者界面

現代 Web 應用程式通常使用 JavaScript 架構來開發與後端 APIs單一頁面應用程式。這些 APIs通常使用代表狀態傳輸 (REST) 或 RESTful APIs 或 GraphQL APIs 建置。您可以使用 Amazon Simple Storage Service (Amazon S3) 和 Amazon CloudFront 提供靜態 Web 內容。

微服務

APIs被視為微服務的前門，因為它們是應用程式邏輯的進入點。一般而言，會使用 RESTful Web 服務 API 或 GraphQL APIs。這些 APIs會管理和處理用戶端呼叫、處理流量管理、請求篩選、路由、快取、身分驗證和授權等函數。

微服務實作

AWS 提供建置區塊來開發微服務，包括 Amazon ECS 和 Amazon EKS 作為容器協同運作引擎的選擇 AWS Fargate ，以及 EC2 作為託管選項。 AWS Lambda 是建置微服務的另一種無伺服器方式 AWS。這些託管選項之間的選擇取決於客戶管理基礎基礎設施的需求。

AWS Lambda 可讓您上傳程式碼，以高可用性自動擴展和管理其執行。這消除了基礎設施管理的需求，因此您可以快速移動並專注於您的商業邏輯。Lambda 支援多種程式設計語言，並且可由其他服務觸發 AWS 或直接從 Web 或行動應用程式呼叫。

由於可攜性、生產力和效率，容器型應用程式越來越受歡迎。 AWS 提供數種服務來建置、部署和管理容器。

- App2Container 是命令列工具，可將 Java 和 .NET Web 應用程式遷移和現代化為容器格式。 AWS A2C 會分析並建置在裸機、虛擬機器、Amazon Elastic Compute Cloud (EC2) 執行個體或雲端中執行的應用程式庫存。

- Amazon Elastic Container Service (Amazon ECS) 和 Amazon Elastic Kubernetes Service (Amazon EKS) 可 管理您的容器基礎設施，讓您更輕鬆地啟動和維護容器化應用程式。
  - Amazon EKS 是一項受管 Kubernetes 服務，可在 AWS 雲端和內部部署資料中心 (Amazon EKS Anywhere) 中執行 Kubernetes。這會將雲端服務延伸至內部部署環境，以滿足低延遲、本機資料處理、高資料傳輸成本或資料駐留需求（請參閱「使用 Amazon EKS Anywhere 執行混合容器工作負載」的白皮書）。您可以透過 EKS 從 Kubernetes 社群使用所有現有的外掛程式和工具。
  - Amazon Elastic Container Service (Amazon ECS) 是一種全受管容器協同運作服務，可簡化容器化應用程式的部署、管理和擴展。客戶選擇 ECS 以簡化和深度整合 AWS 服務。

如需進一步閱讀，請參閱部落格 Amazon ECS 與 Amazon EKS：了解 AWS 容器服務。

- AWS App Runner 是一種全受管容器應用程式服務，可讓您建置、部署和執行容器化 Web 應用程式和 API 服務，而不需要先前的基礎設施或容器體驗。

- AWS Fargate是無伺服器運算引擎，可與 Amazon ECS 和 Amazon EKS 搭配使用，以自動管理容器應用程式的運算資源。
- Amazon ECR 是全受管容器登錄檔，提供高效能託管，因此您可以可靠地將應用程式映像和成品部署到任何地方。

持續整合和持續部署 (CI/CD)

持續整合和持續交付 (CI/CD) 是 DevOps 快速軟體變更計劃 的重要部分。 AWS 提供實作微服務 CI/CD 的服務，但詳細討論超出本文件的範圍。如需詳細資訊，請參閱在白皮書上練習持續整合和持續交付 AWS。

私有網路

AWS PrivateLink 是一項技術，透過允許虛擬私有雲端 (VPC) 與支援的 服務之間的私有連線，增強微 AWS 服務的安全性。它有助於隔離和保護微服務流量，確保其永遠不會跨越公有網際網路。這對於遵守 PCI 或 HIPAA 等法規特別有用。

資料存放區

資料存放區用於保留微服務所需的資料。工作階段資料的熱門存放區是記憶體內快取，例如 Memcached 或 Redis。 AWS 提供兩種技術做為受管 Amazon ElastiCache 服務的一部分。

在應用程式伺服器和資料庫之間放置快取是減少資料庫讀取負載的常見機制，進而允許資源用於支援更多寫入。快取也可以改善延遲。

關聯式資料庫在儲存結構化資料和商業物件方面仍然非常熱門。 透過 Amazon Relational Database Service (Amazon RDS) AWS 提供六個資料庫引擎 (Microsoft SQL Server、Oracle、MySQL、MariaDB、PostgreSQL 和 Amazon Aurora) 做為受管服務。

不過，關聯式資料庫並非針對無限擴展而設計，這可能會讓套用技術以支援大量查詢變得困難且耗時。

NoSQL 資料庫旨在有利於關聯式資料庫一致性的可擴展性、效能和可用性。NoSQL 資料庫的一個重要元素是它們通常不會強制執行嚴格的結構描述。資料分散到可以水平擴展的分割區，並使用分割區索引鍵擷取。

由於個別微服務的設計是很好地執行一件事，因此它們通常具有簡化的資料模型，可能非常適合 NoSQL 持久性。請務必了解 NoSQL 資料庫的存取模式與關聯式資料庫不同。例如，無法聯結資料

表。如果需要，則必須在應用程式中實作邏輯。您可以使用 Amazon DynamoDB 建立資料庫資料表，以存放和擷取任意數量的資料，並為任何層級的請求流量提供服務。DynamoDB 提供單一位數毫秒的效能，不過，某些使用案例需要以微秒為單位的回應時間。 DynamoDB Accelerator (DAX) 提供快取功能來存取資料。

DynamoDB 還提供自動擴展功能，可動態調整輸送量容量以回應實際流量。不過，在某些情況下，由於應用程式的短期活動高峰，容量規劃很難或無法進行。在這種情況下，DynamoDB 提供隨需選項，提供簡單的pay-per-request定價。DynamoDB 隨需能夠立即處理數千個請求，無需規劃容量。

如需詳細資訊，請參閱 分散式資料管理和如何選擇資料庫。

簡化操作

為了進一步簡化執行、維護和監控微服務所需的操作工作，我們可以使用 完全無伺服器架構。

部署 Lambda 型應用程式

您可以透過上傳zip檔案封存，或使用有效的 Amazon ECR 映像 URI 透過主控台 UI 建立和上傳容器映像，來部署 Lambda 程式碼。不過，當 Lambda 函數變得複雜時，表示它具有層、相依性和許可，透過 UI 上傳可能會變得難以進行程式碼變更。

使用 AWS CloudFormation 和 AWS Serverless Application Model (AWS SAM) AWS Cloud Development Kit (AWS CDK)或 Terraform 可簡化定義無伺服器應用程式的程序。CloudFormation 原生支援的 AWS SAM 提供簡化的語法，用於指定無伺服器資源。 AWS Lambda Layers 可協助管理跨多個 Lambda 函數的共用程式庫、盡可能減少函數使用量、集中租用戶感知程式庫，並改善開發人員體驗。Lambda SnapStart for Java 可增強對延遲敏感應用程式的啟動效能。

若要部署，請在 CloudFormation 範本中指定資源和許可政策、套件部署成品，以及部署範本。SAM Local 是一種 AWS CLI 工具，允許在上傳至 Lambda 之前對無伺服器應用程式進行本機開發、測試和分析。

與 AWS Cloud9 IDE 等工具整合 AWS CodeBuild AWS CodeDeploy，並 AWS CodePipeline 簡化撰寫、測試、偵錯和部署 SAM 型應用程式。

下圖顯示使用 CloudFormation 和 AWS CI/CD 工具部署 AWS Serverless Application Model 資源。

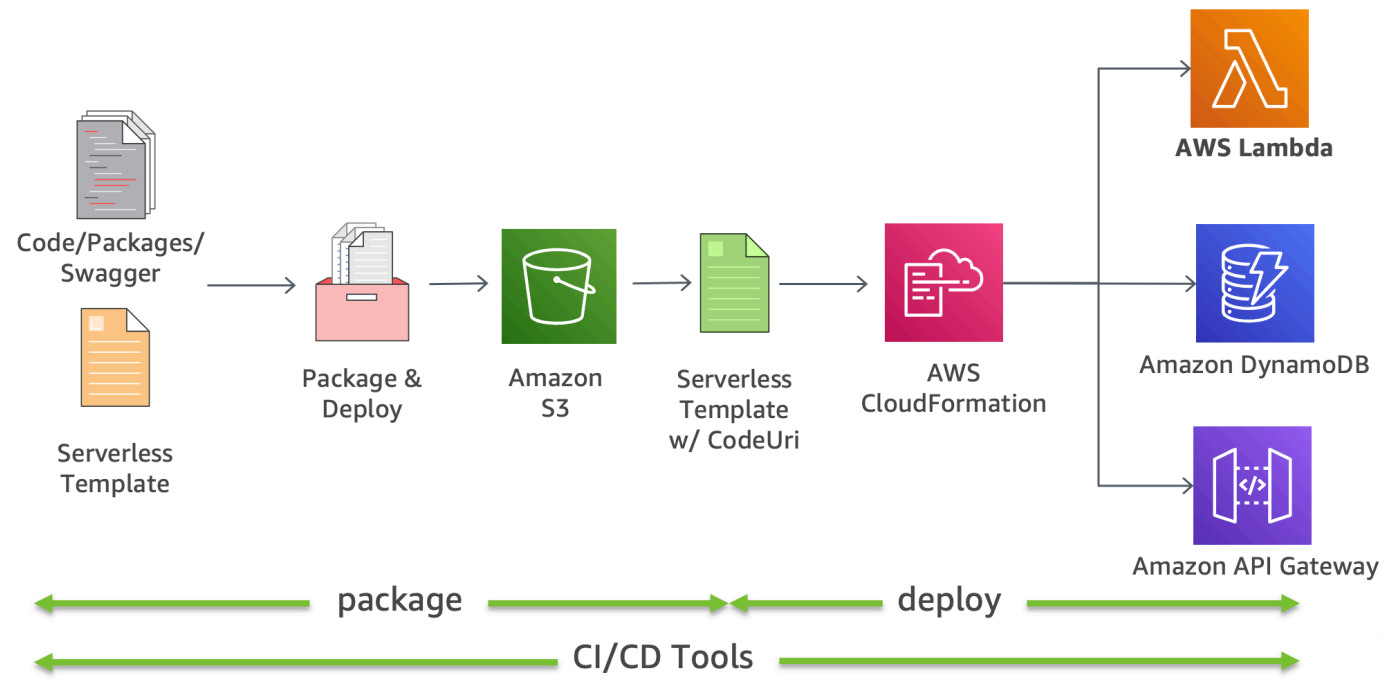

圖 2： AWS Serverless Application Model (AWS SAM)

抽象化多租用戶複雜性

在 SaaS 平台等多租戶環境中，簡化與多租戶相關的複雜性至關重要，讓開發人員能夠專注於特徵和功能開發。這可以使用 AWS Lambda Layers 等工具來實現，該工具提供共同程式庫來解決交叉切割問題。這種方法背後 的原理是，共用程式庫和工具在正確使用時，可以有效地管理租戶內容。

但是，由於業務邏輯可能帶來的複雜性和風險，他們不應該擴展到封裝業務邏輯。共用程式庫的基本問題是與更新相關的複雜性增加，相較於標準程式碼複製，管理更具挑戰性。因此，在尋找最有效的抽象時，在共用程式庫的使用與重複之間取得平衡至關重要。

API 管理

管理 APIs 可能很耗時，特別是在考慮多個版本、開發週期階段、授權和其他功能時，例如限流和快取。除了 API Gateway 之外，有些客戶也會使用 ALB (Application Load Balancer) 或 NLB (Network Load Balancer) 進行 API 管理。Amazon API Gateway 有助於降低建立和維護 RESTful APIs的操作複雜性。它可讓您以程式設計方式建立 APIs，做為「前門」，從後端服務存取資料、商業邏輯或功能、授權和存取控制、速率限制、快取、監控和流量管理，並在不管理伺服器的情況下執行 APIs。

圖 3 說明 API Gateway 如何處理 API 呼叫並與其他元件互動。來自行動裝置、網站或其他後端服務的請求會路由至最近的 CloudFront 存在點 (PoP)，以減少 延遲 並提供最佳的 使用者體驗。

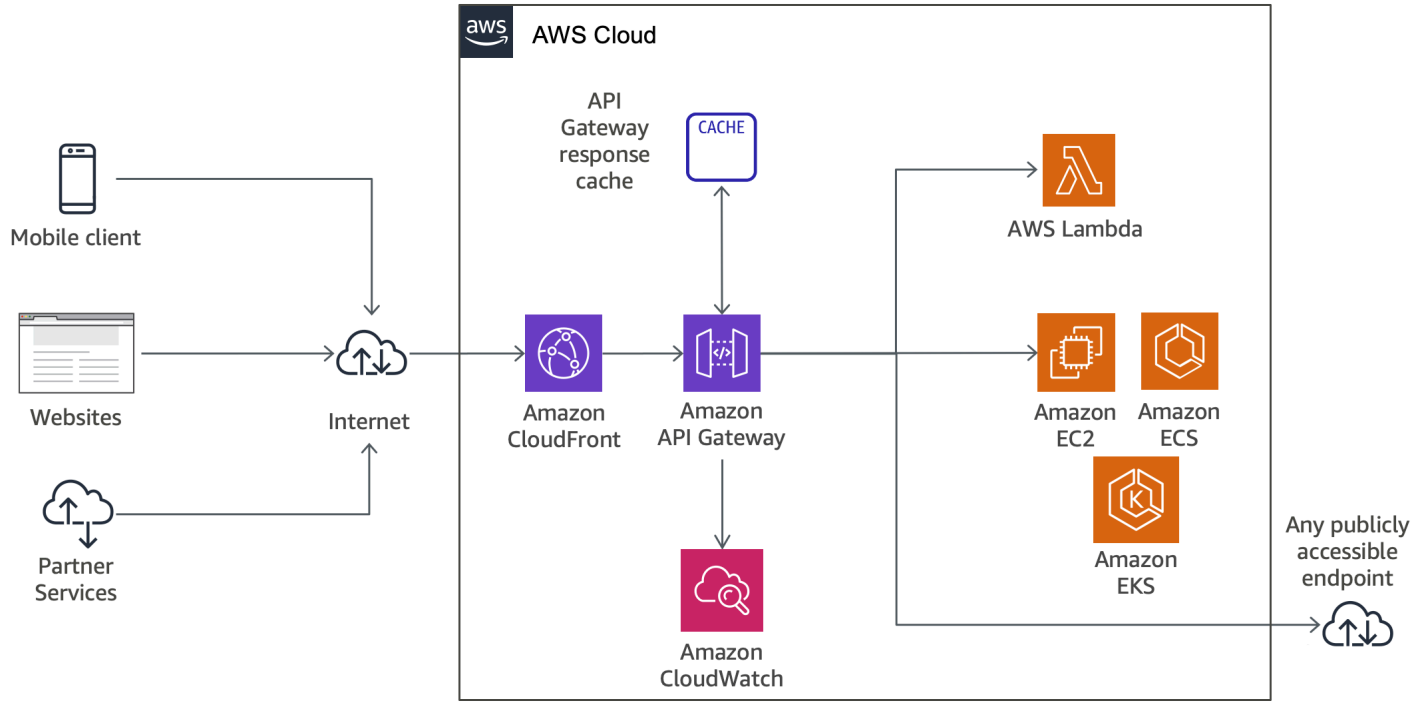

圖 3：API Gateway 呼叫流程

無伺服器技術的微服務

將微服務與無伺服器技術搭配使用可大幅降低操作複雜性。 AWS Lambda 與 API Gateway AWS Fargate整合後，可建立完全無伺服器的應用程式。自 2023 年 4 月 7 日起，Lambda 函數可以逐步將回應承載串流回用戶端，從而增強 Web 和行動應用程式的效能。在此之前，使用傳統請求回應調用模型的 Lambda 型應用程式必須先產生和緩衝回應，再將其傳回給用戶端，這可能會延遲到第一個位元組的時間。透過回應串流，函數可以在用戶端就緒時將部分回應傳回用戶端，大幅縮短第一個位元組的時間，而 Web 和行動應用程式特別敏感。

圖 4 示範使用 AWS Lambda 和 受管服務的無伺服器微服務架構。此無伺服器架構可減少設計擴展和高可用性的需求，並減少執行和監控基礎基礎設施所需的工作量。

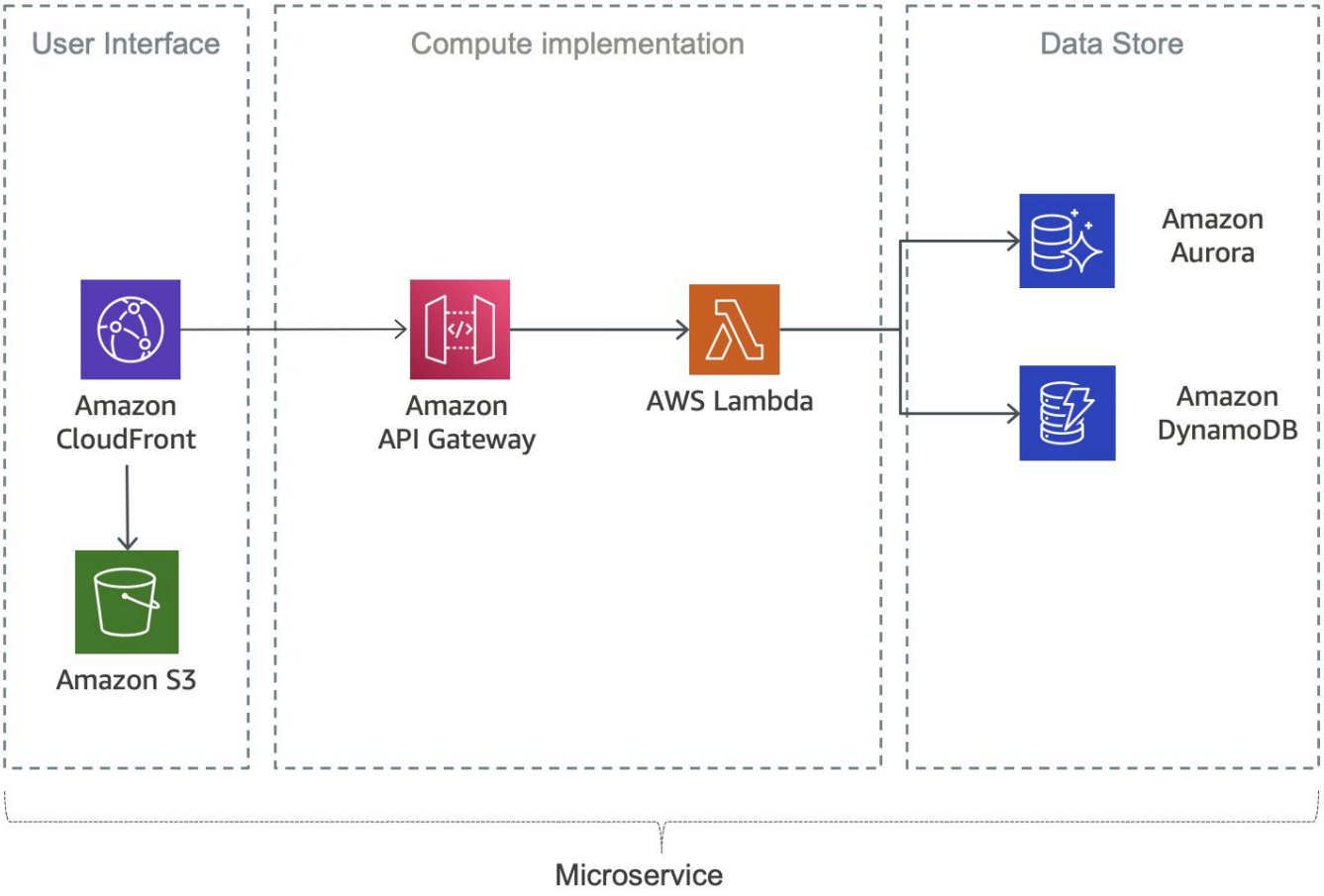

圖 4：使用 的無伺服器微服務 AWS Lambda

圖 5 顯示使用容器搭配 的類似無伺服器實作 AWS Fargate，消除對基礎基礎設施的疑慮。它還具有 Amazon Aurora Serverless，這是一種隨需的自動擴展資料庫，可根據應用程式的需求自動調整容量。

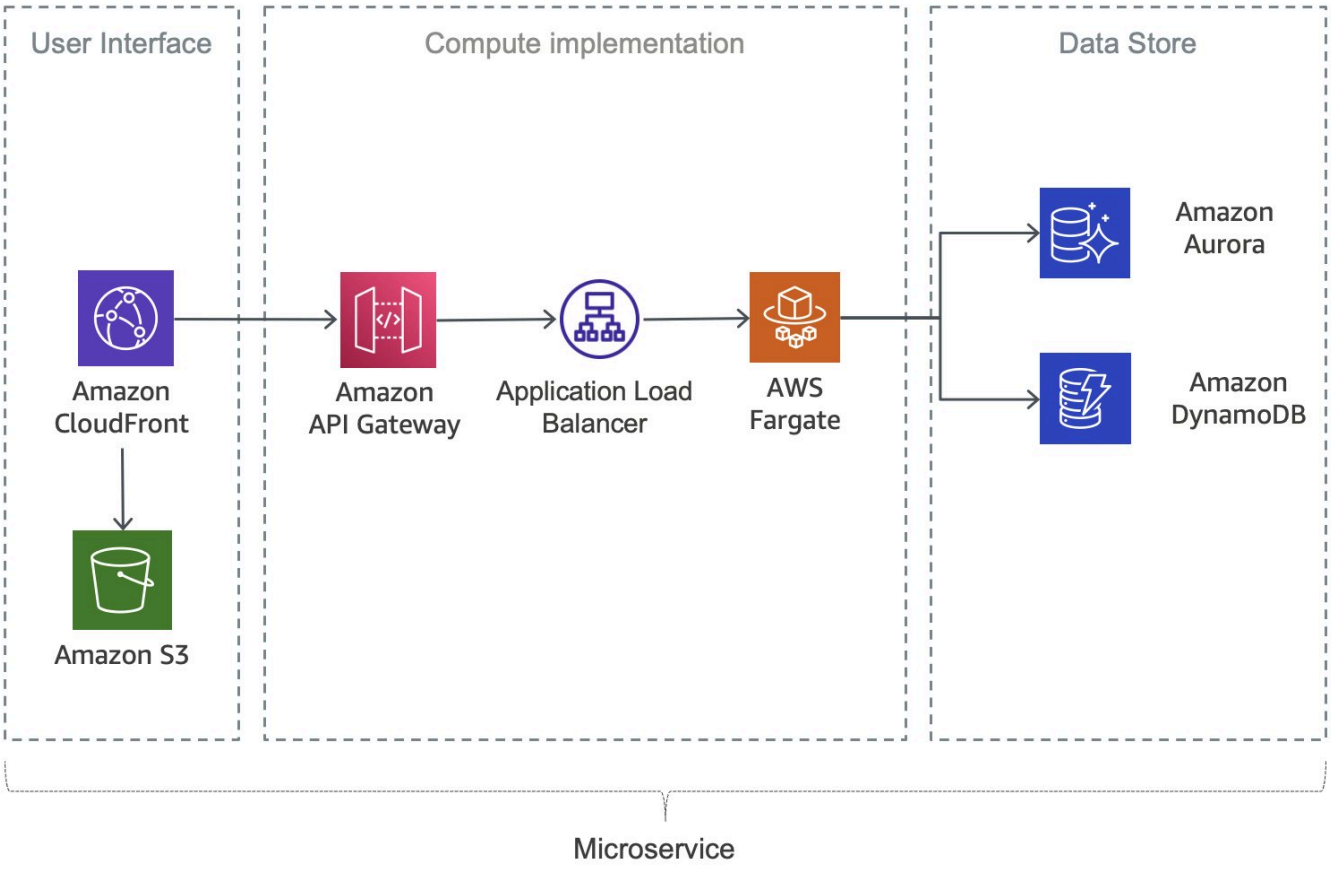

圖 5：使用 的無伺服器微服務 AWS Fargate

彈性且有效率的系統

災難復原 (DR)

Microservices 應用程式通常遵循十二因素應用程式模式，其中程序是無狀態的，持久性資料存放在資料庫等具狀態的備份服務中。這可簡化災難復原 (DR)，因為如果服務失敗，可以輕鬆啟動新的執行個體以還原功能。

微服務災難復原策略應著重於維護應用程式狀態的下游服務，例如檔案系統、資料庫或佇列。組織應規劃復原時間目標 (RTO) 和復原點目標 (RPO)。RTO 是服務中斷和還原之間的可接受延遲上限，而 RPO 是自上次資料復原點以來的最長時間。

如需災難復原策略的詳細資訊，請參閱 雲端白皮書中的工作負載的災難復原 AWS：復原。

高可用性 (HA)

我們將檢查微服務架構的各種元件的高可用性 (HA)。

Amazon EKS 透過跨多個可用區域執行 Kubernetes 控制和資料平面執行個體來提供高可用性。它會自動偵測並取代運作狀態不佳的控制平面執行個體，並提供自動化版本升級和修補。

Amazon ECR 使用 Amazon Simple Storage Service (Amazon S3) 進行儲存，讓您的容器映像具有高度可用性和可存取性。它適用於 Amazon EKS、Amazon ECS AWS Lambda，並簡化生產工作流程的開發。

Amazon ECS 是一項區域服務，可在區域內多個可用區域中以高可用性的方式簡化執行中的容器，提供多個排程策略，以放置容器以滿足資源需求和可用性需求。

AWS Lambda 會在多個可用區域中運作，確保單一區域中服務中斷期間的可用性。如果將函數連接到 VPC，請在多個可用區域中指定子網路，以獲得高可用性。

分散式系統元件

在微服務架構中，服務探索是指在分散式系統中動態定位和識別個別微服務的網路位置 (IP 地址和連接埠）的程序。

選擇 方法時 AWS，請考慮下列因素：

- 程式碼修改：您可以獲得優勢，而無需修改程式碼嗎？
- 跨 VPC 或跨帳戶流量：如果需要，您的系統是否需要跨不同 VPCs或 進行有效的通訊管理 AWS 帳戶？
- 部署策略：您的系統是否使用或計劃使用進階部署策略，例如藍綠或金絲雀部署？
- 效能考量：如果您的架構經常與外部服務通訊，對整體效能會有什麼影響？

AWS 在微服務架構中提供數種實作服務探索的方法：

- Amazon ECS Service Discovery：Amazon ECS 支援使用以 DNS 為基礎的方法或與 整合的服務探索 AWS Cloud Map（請參閱 ECS Service Discovery)。ECS Service Connect 進一步改善連線管理，這對於具有多個互動服務的大型應用程式特別有益。
- Amazon Route 53：Route 53 與 ECS 和其他 AWS 服務整合，例如 EKS，以促進服務探索。在 ECS 內容中，Route 53 可以使用 ECS Service Discovery 功能，該功能利用 Auto Naming API 自動註冊和取消註冊服務。
- AWS Cloud Map：此選項提供動態 API 型服務探索，可在您的服務間傳播變更。

對於更進階的通訊需求，Amazon VPC Lattice 是一種應用程式聯網服務，可一致地連線、監控和保護服務之間的通訊，有助於提高生產力，讓您的開發人員可以專注於建置對業務重要的功能。您可以定義網路流量管理、存取和監控的政策，以簡化且一致的方式跨執行個體、容器和無伺服器應用程式連接運算服務。

如果您已經使用第三方軟體，例如 HashiCorp Consul 或 Netflix Eureka 進行服務探索，您可能偏好在遷移至 時繼續使用這些軟體 AWS，以便更順暢地進行轉換。

這些選項之間的選擇應符合您的特定需求。對於更簡單的需求，如 Amazon ECS 或 等 DNS 型解決方案 AWS Cloud Map 可能就足夠了。對於更複雜或更大的系統，像 Amazon VPC Lattice 之類的服務網格可能更合適。

總而言之，在 上設計微服務架構 AWS ，就是選擇適當的工具來滿足您的特定需求。請記住討論的考量事項，您可以確保做出明智的決策，以最佳化系統的服務探索和服務間通訊。

分散式資料管理

在傳統應用程式中，所有元件通常共用單一資料庫。相反地，微服務型應用程式的每個元件都會維護自己的資料，進而促進獨立和分散。這種方法稱為分散式資料管理，帶來了新的挑戰。

其中一種挑戰來自分散式系統中一致性和效能之間的權衡。接受資料更新（最終一致性）的輕微延遲通常比堅持即時更新（立即一致性）更為實際。

有時候，業務營運需要多個微服務才能一起運作。如果某個部分失敗，您可能必須復原一些已完成的任務。Saga 模式可透過協調一系列補償動作來協助管理此項目。

為了協助微服務保持同步，可以使用集中式資料存放區。此存放區使用 AWS Lambda AWS Step Functions和 Amazon EventBridge 等工具進行管理，可協助清除和刪除重複資料。

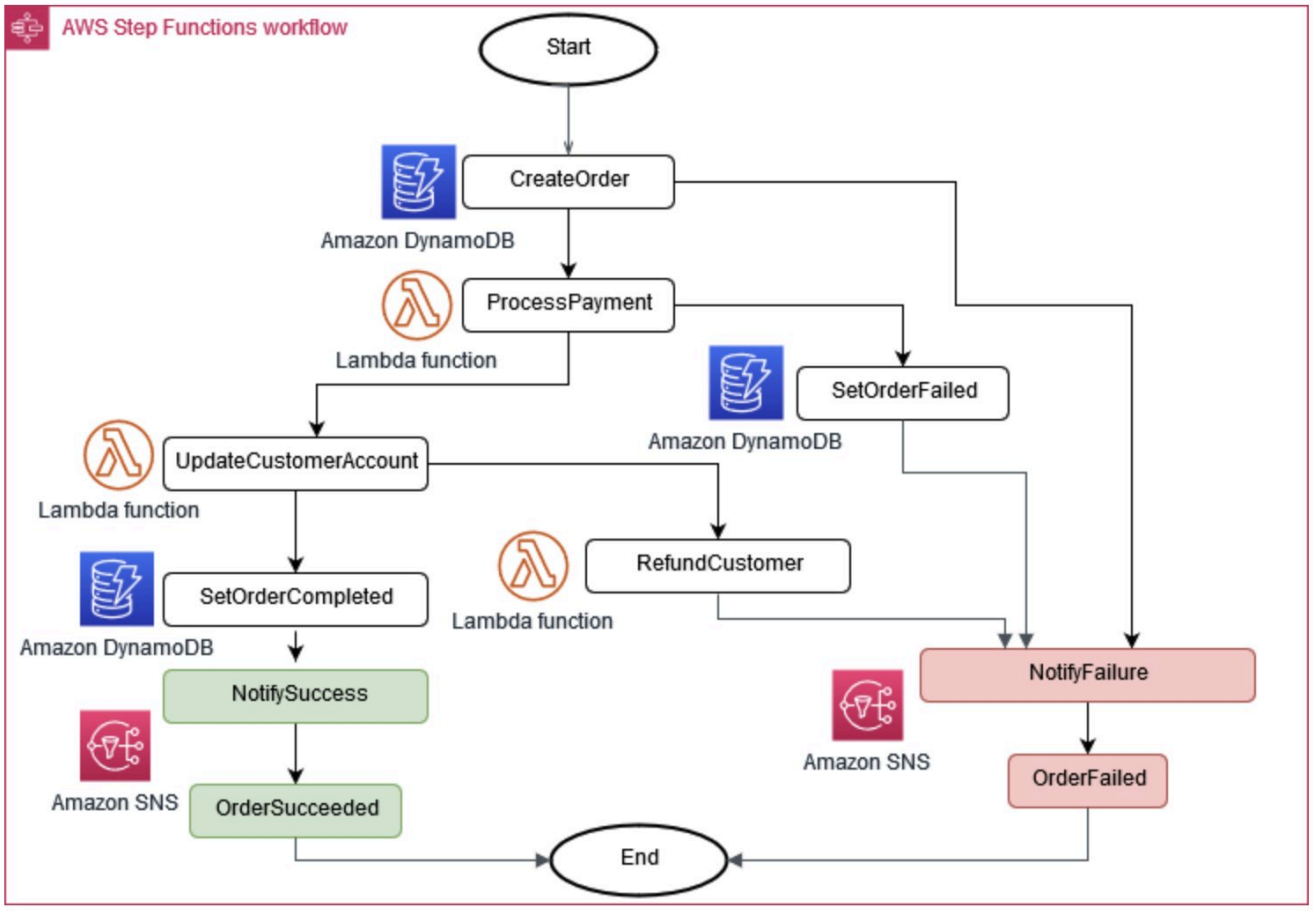

圖 6：Saga 執行協調器

管理跨微服務變更的常見方法是事件來源。應用程式中的每個變更都會記錄為事件，建立系統狀態的時間軸。此方法不僅有助於偵錯和稽核，還允許應用程式的不同部分對相同的事件做出反應。

事件來源通常會與 Command Query Responsibility Segregation (CQRS) 模式hand-in-hand，這會將資料修改和資料查詢分開到不同的模組中，以提高效能和安全性。

在 上 AWS，您可以使用 服務組合來實作這些模式。如圖 7 所示，Amazon Kinesis Data Streams 可以做為您的中央事件存放區，而 Amazon S3 則為所有事件記錄提供耐用的儲存體。 AWS Lambda、Amazon DynamoDB 和 Amazon API Gateway 會一起合作來處理這些事件。

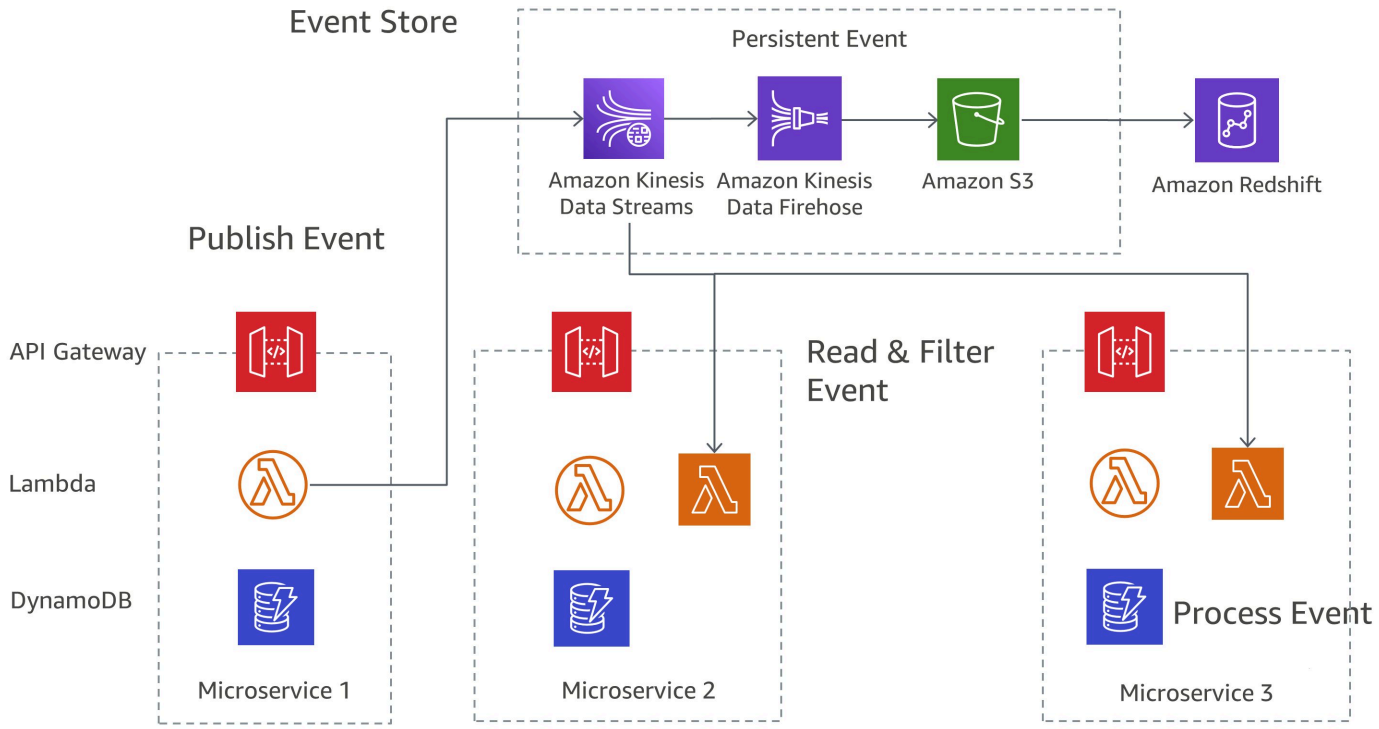

圖 7： 上的事件來源模式 AWS

請記住，在分散式系統中，事件可能會因為重試而多次交付，因此設計您的應用程式來處理這一點非常重要。

組態管理

在微服務架構中，每個服務都會與資料庫、佇列和其他服務等各種資源互動。設定每個服務的連線和操作環境的一致方式至關重要。理想情況下，應用程式應適應新的組態，而不需要重新啟動。此方法是十二要素應用程式原則的一部分，建議將組態儲存在環境變數中。

不同的方法是使用 AWS App Config。這是 AWS Systems Manager 的一項功能，可讓客戶輕鬆快速且安全地設定、驗證和部署功能旗標和應用程式組態。您的特徵標記和組態資料可以在部署前階段以語法或同義方式進行驗證，如果觸發您已設定的警示，則可以監控和自動復原。AppConfig 可以使用 theAppConfig 代理程式與 Amazon ECS 和 Amazon EKS AWS AppConfig 整合。代理程式可做為與 Amazon ECS 和 Amazon EKS 容器應用程式一起執行的附屬容器。如果您在 AWS Lambda 函數中使用 AppConfig 功能旗標或其他動態組態資料，建議您將 AWS AppConfig Lambda 延伸模組做為圖層新增至 Lambda 函數。

GitOps 是一種創新的組態管理方法，使用 Git 作為所有組態變更的事實來源。這表示對組態檔案所做的任何變更都會透過 Git 自動追蹤、版本化和稽核。

秘密管理

安全性至關重要，因此不應以純文字傳遞登入資料。 為此 AWS 提供安全服務，例如AWS Systems Manager 參數存放區 和 AWS Secrets Manager。這些工具可以將秘密以磁碟區形式傳送至 Amazon EKS 中的容器，或以環境變數形式傳送至 Amazon ECS。在 中 AWS Lambda，環境變數會自動提供給程式碼。對於 Kubernetes 工作流程，外部秘密運算子會直接從 服務擷取秘密 AWS Secrets Manager，例如建立對應的 Kubernetes 秘密。這可讓 與 Kubernetes 原生組態無縫整合。

成本最佳化和永續性

Microservices 架構可以增強成本最佳化和永續性。透過將應用程式分成較小的部分，您只能擴展需要更多資源的服務，從而降低成本和浪費。這在處理可變流量時特別有用。微服務是獨立開發的。因此，開發人員可以執行較小的更新，並減少用於端對端測試的資源。更新時，他們只需要測試一部分功能，而不是單體。

架構中的無狀態元件（存放在外部資料存放區而非本機資料存放區的服務）可以使用 Amazon EC2 Spot 執行個體，其可在 AWS 雲端中提供未使用的 EC2 容量。這些執行個體比隨需執行個體更具成本效益，非常適合可處理中斷的工作負載。這可以進一步降低成本，同時維持高可用性。

透過隔離服務，您可以為每個自動擴展群組使用成本最佳化運算選項。例如， AWS Graviton 為適合 ARM 型執行個體的工作負載提供經濟實惠的高效能運算選項。

最佳化成本和資源用量也有助於將環境影響降至最低，並與 Well-Architected Framework 的永續性支柱保持一致。您可以使用 AWS Customer Carbon Footprint Tool 監控減少碳排放的進度。此工具可讓您深入了解 AWS 用量對環境的影響。

通訊機制

在微服務範例中，應用程式的各種元件必須透過網路進行通訊。常見的方法包括 REST 型、GraphQL 型、gRPC 型和非同步傳訊。

以 REST 為基礎的通訊

HTTP/S 通訊協定廣泛用於微服務之間的同步通訊，通常透過 RESTful APIs操作。API Gateway 提供簡化的方式來建置 API，做為後端服務的集中存取點，處理流量管理、授權、監控和版本控制等任務。

GraphQL 型通訊

同樣地，GraphQL 是同步通訊的廣泛方法，使用與 REST 相同的通訊協定，但限制對單一端點的暴露。使用 AWS AppSync，您可以建立和發佈直接與服務 AWS 和資料存放區互動的 GraphQL 應用程式，或將 Lambda 函數納入商業邏輯。

gRPC 型通訊

gRPC 是一種同步、輕量、高效能的開放原始碼 RPC 通訊協定。gRPC 使用 HTTP/2 改善其基礎通訊協定，並啟用壓縮和串流優先順序等更多功能。它使用以二進位編碼的 Protobuf 界面定義語言 (IDL)，因此會利用 HTTP/2 二進位框架。

非同步傳訊和事件傳遞

非同步傳訊可讓服務透過佇列傳送和接收訊息來進行通訊。這可讓服務保持鬆散耦合，並提升服務探索。

訊息可以定義以下三種類型：

- 訊息佇列：訊息佇列可做為緩衝區，將訊息的寄件者（生產者）和接收者（消費者）分離。生產者將訊息排入佇列中，消費者取消排入佇列並進行處理。此模式適用於非同步通訊、負載平衡和處理流量爆增。
- Publish-Subscribe：在發佈訂閱模式中，訊息會發佈至主題，且多位有興趣的訂閱者會收到訊息。此模式可讓多個消費者以非同步方式廣播事件或訊息。
- 事件驅動訊息：事件驅動訊息涉及擷取和回應系統中發生的事件。事件會發佈到訊息中介裝置，感興趣的服務會訂閱特定的事件類型。此模式可讓鬆散耦合，並允許 服務在沒有直接相依性的情況下對事件做出反應。

為了實作這些訊息類型， AWS 提供各種受管服務，例如 Amazon SQS、Amazon SNS、Amazon EventBridge、Amazon MQ 和 Amazon MSK。這些服務具有專為特定需求量身打造的獨特功能：

- Amazon Simple Queue Service (Amazon SQS) 和 Amazon Simple Notification Service (Amazon SNS)：如圖 8 所示，這兩個服務互相補充，Amazon SQS 提供存放訊息的空間，而 Amazon SNS 可將訊息交付給多個訂閱者。當相同的訊息需要交付到多個目的地時，它們是有效的。

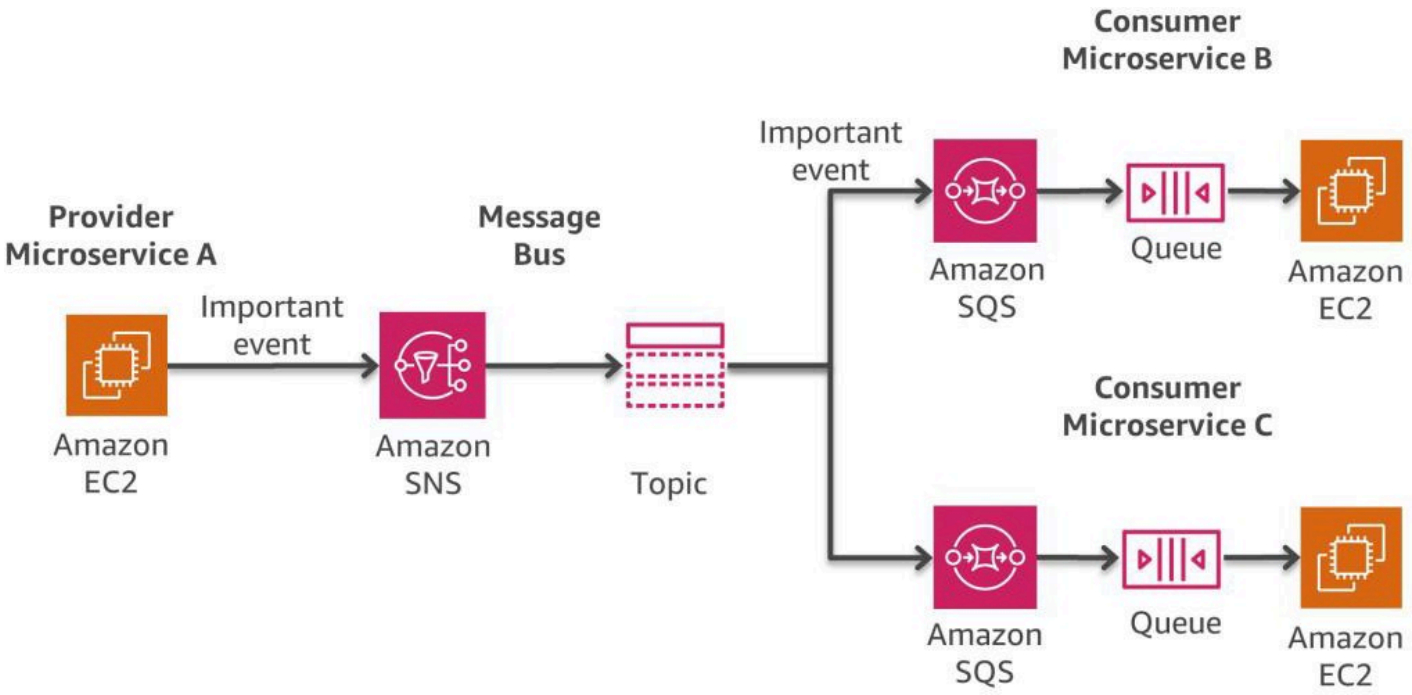

圖 8： 上的訊息匯流排模式 AWS

- Amazon EventBridge：一種無伺服器服務，使用事件將應用程式元件連接在一起，讓您更輕鬆地建置可擴展的事件驅動型應用程式。使用它將事件從自製應用程式、 AWS 服務和第三方軟體等來源路由到整個組織的消費者應用程式。EventBridge 提供簡單且一致的方式來擷取、篩選、轉換和交付事件，讓您可以快速建立新的應用程式。EventBridge 事件匯流排非常適合事件驅動服務之間的many-to-many事件路由。
- Amazon MQ：如果您有一個預先存在的訊息系統使用標準通訊協定，例如 JMS、AMQP 或類似通訊協定，這是不錯的選擇。此受管服務可取代您的系統，而不會中斷操作。
- Amazon MSK （受管 Kafka)： 一種用於儲存和讀取訊息的訊息系統，適用於必須多次處理訊息的情況。它也支援即時訊息串流。
- Amazon Kinesis： 即時處理和分析串流資料。這允許開發即時應用程式，並提供與 AWS 生態系統的無縫整合。

請記住，最適合您的服務取決於您的特定需求，因此請務必了解每個服務提供的內容，以及它們如何符合您的需求。

協同運作和狀態管理

微服務協調是指集中式方法，其中稱為協調器的中央元件負責管理和協調微服務之間的互動。跨多個微服務協調工作流程可能具有挑戰性。不鼓勵將協同運作程式碼直接嵌入服務，因為它引入更緊密的耦合，並阻礙取代個別服務。

Step Functions 提供工作流程引擎來管理服務協調複雜性，例如錯誤處理和序列化。這可讓您快速擴展和變更應用程式，而無需新增協調程式碼。Step Functions 是無 AWS 伺服器平台的一部分，並支援 Lambda 函數、Amazon EC2、Amazon EKS、Amazon ECS、SageMaker AI AWS Glue等。

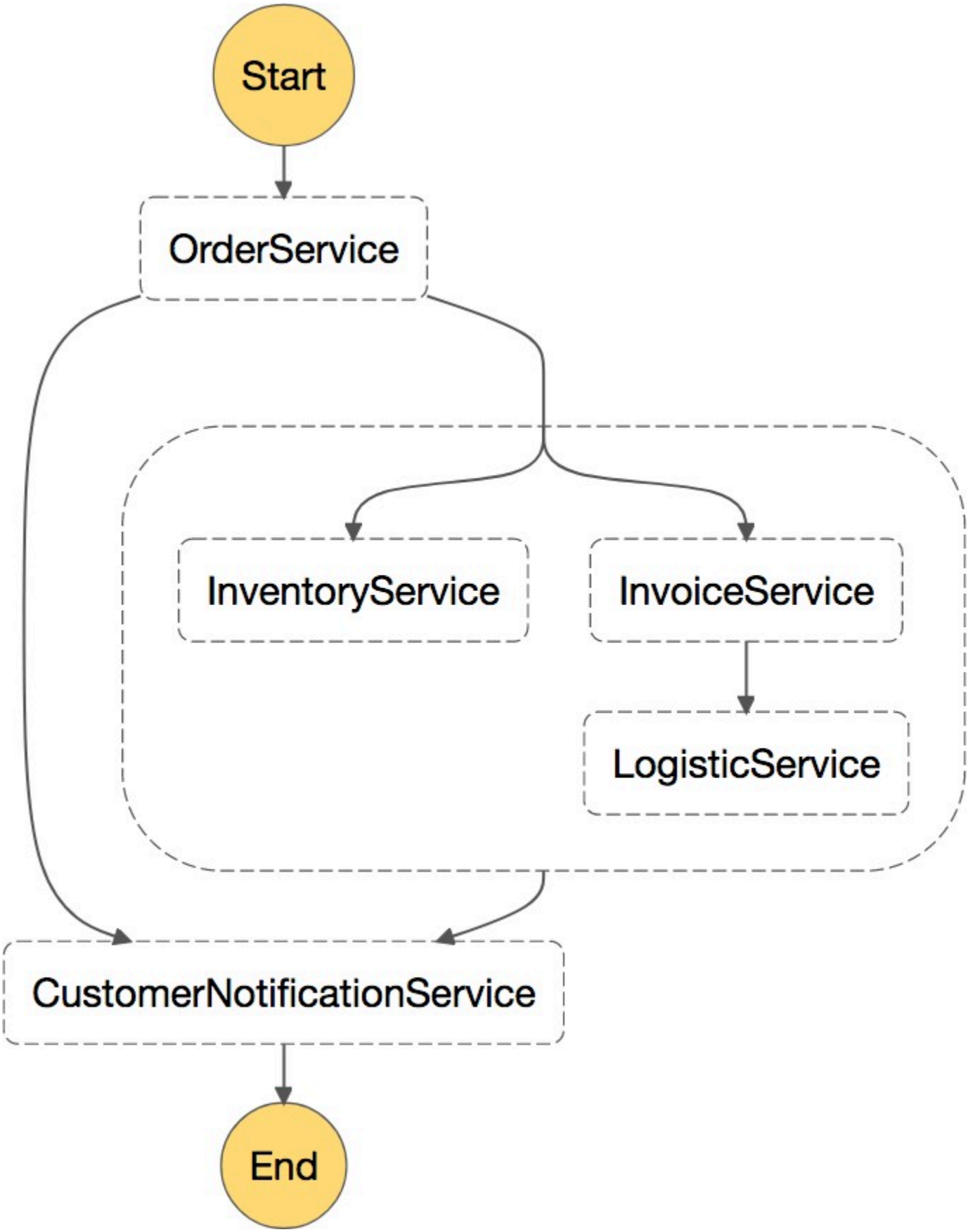
Start
OrderService
InventoryService
InvoiceService
LogisticService
CustomerNotificationService
End

圖 9：由 調用平行和循序步驟的微服務工作流程範例 AWS Step Functions

Amazon Managed Workflows for Apache Airflow (Amazon MWAA) 是 Step Functions 的替代方案。如果您優先考慮開放原始碼和可攜性，則應使用 Amazon MWAA。Airflow 具有大型且活躍的開放原始碼社群，可定期提供新功能和整合。

可觀測性

由於微服務架構本質上由許多分散式元件組成，因此所有這些元件的可觀測性變得至關重要。Amazon CloudWatch 啟用此功能、收集和追蹤指標、監控日誌檔案，以及對 AWS 環境中的變更做出反應。它可以監控應用程式和服務產生的 AWS 資源和自訂指標。

主題

- 監控
- 集中日誌
- 分散式追蹤
- 上的日誌分析 AWS
- 其他分析選項

監控

CloudWatch 提供整個系統的資源使用率、應用程式效能和運作狀態的可見性。在微服務架構中，透過 CloudWatch 進行自訂指標監控很有幫助，因為開發人員可以選擇收集哪些指標。動態擴展也可以以這些自訂指標為基礎。

CloudWatch Container Insights 擴展此功能，自動收集 CPU、記憶體、磁碟和網路等許多資源的指標。它有助於診斷容器相關問題，簡化解決方案。

對於 Amazon EKS，常用的選擇是 Prometheus，這是一種提供全方位監控和提醒功能的開放原始碼平台。它通常與 Grafana 結合使用，以實現直覺式指標視覺化。Amazon Managed Service for Prometheus (AMP) 提供與 Prometheus 完全相容的監控服務，讓您輕鬆監督容器化應用程式。此外，Amazon Managed Grafana (AMG) 可簡化指標的分析和視覺化，無需管理基礎基礎設施。

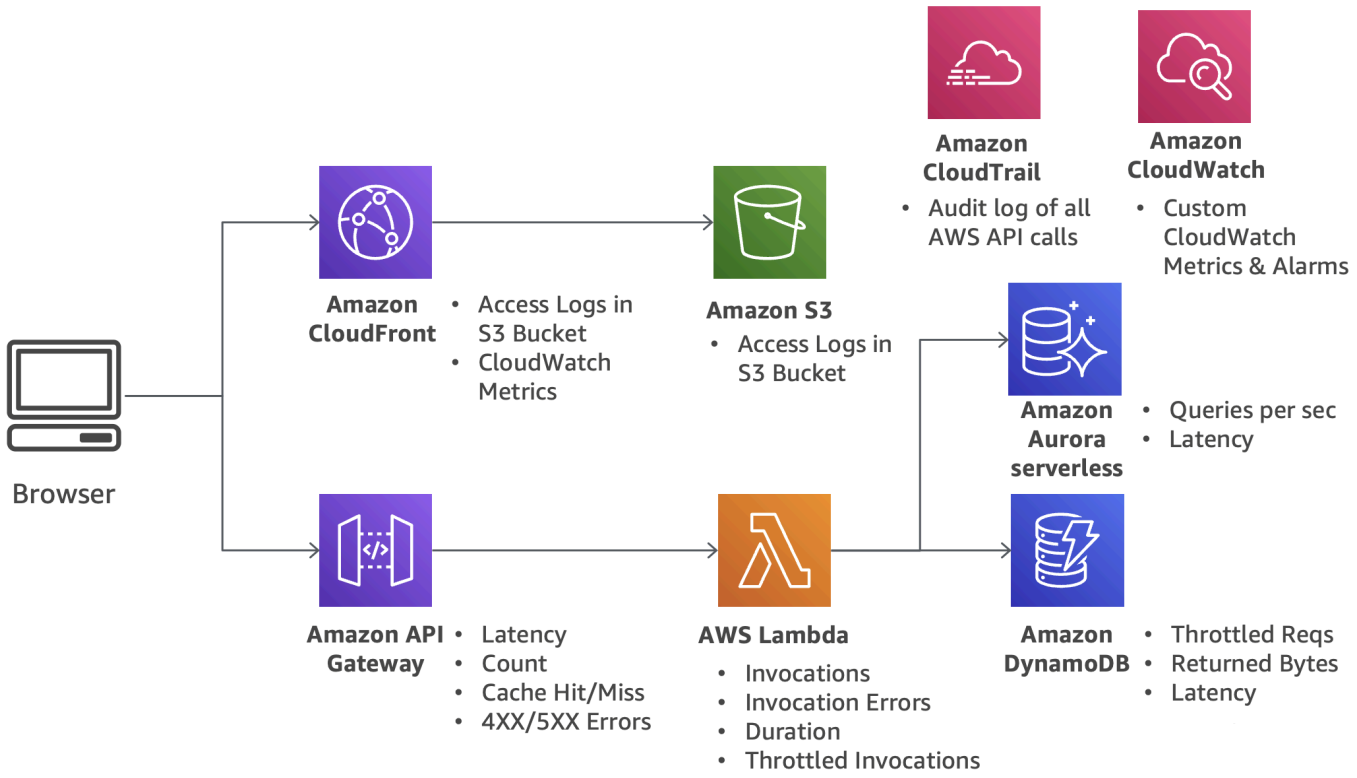

圖 10：具有監控元件的無伺服器架構

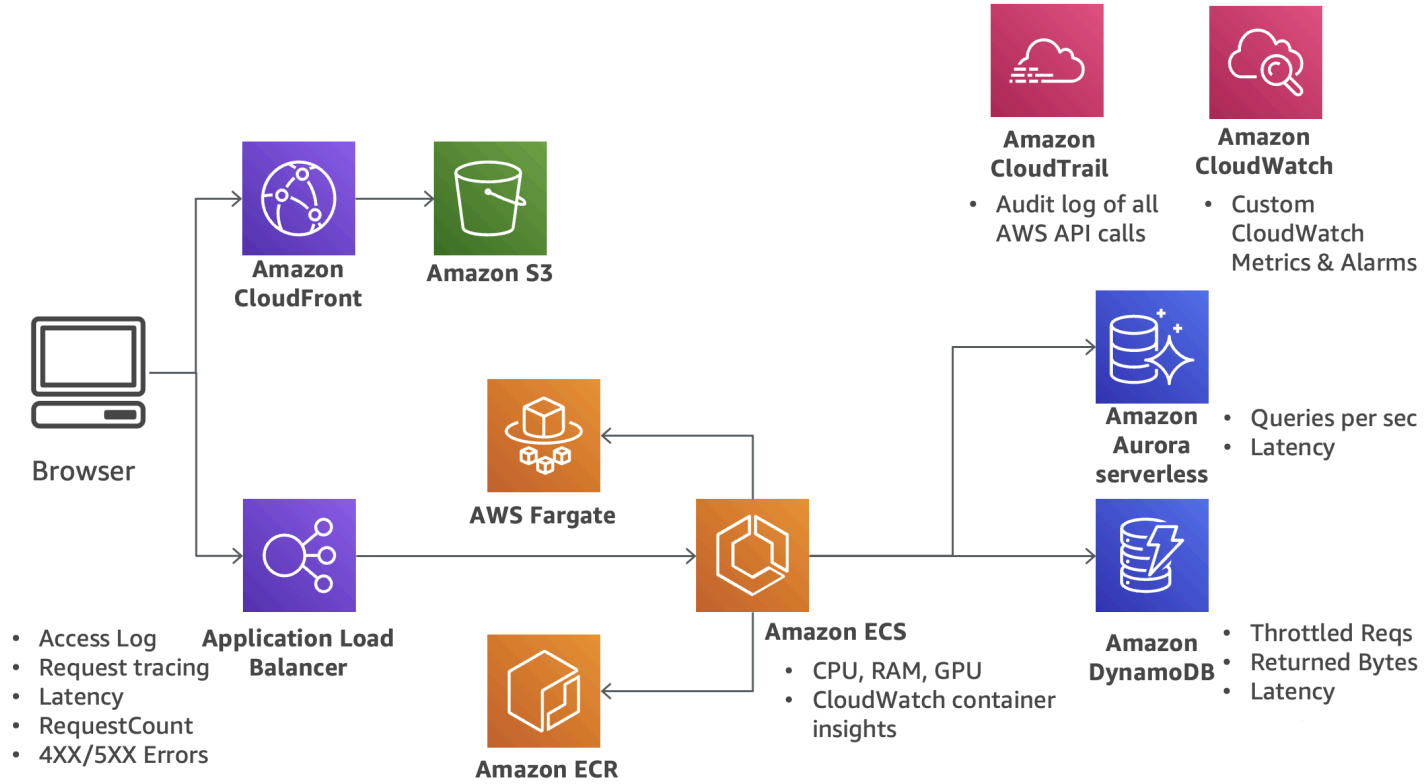

圖 11：具有監控元件的容器型架構

集中日誌

記錄是找出並解決問題的關鍵。透過微服務，您可以更頻繁地發佈並試驗新功能。 AWS 提供 Amazon S3、 CloudWatch Logs 和 Amazon OpenSearch Service 等服務來集中化日誌檔案。Amazon EC2 使用協助程式將日誌傳送至 CloudWatch，而 Lambda 和 Amazon ECS 則會在那裡原生傳送其日誌輸出。對於 Amazon EKS，Fluent Bit 或 Fluentd 可用於將日誌轉送至 CloudWatch，以使用 OpenSearch 和 Kibana 進行報告。不過，由於佔用空間 和效能優勢較小，建議使用 Fluent Bit 而非 Fluentd。

圖 12 說明如何將各種 AWS 服務的日誌導向 Amazon S3 和 CloudWatch。這些集中式日誌可以使用 Amazon OpenSearch Service 進一步分析，包括用於資料視覺化的 Kibana。此外，Amazon Athena 可用於針對存放在 Amazon S3 中的日誌進行臨機操作查詢。

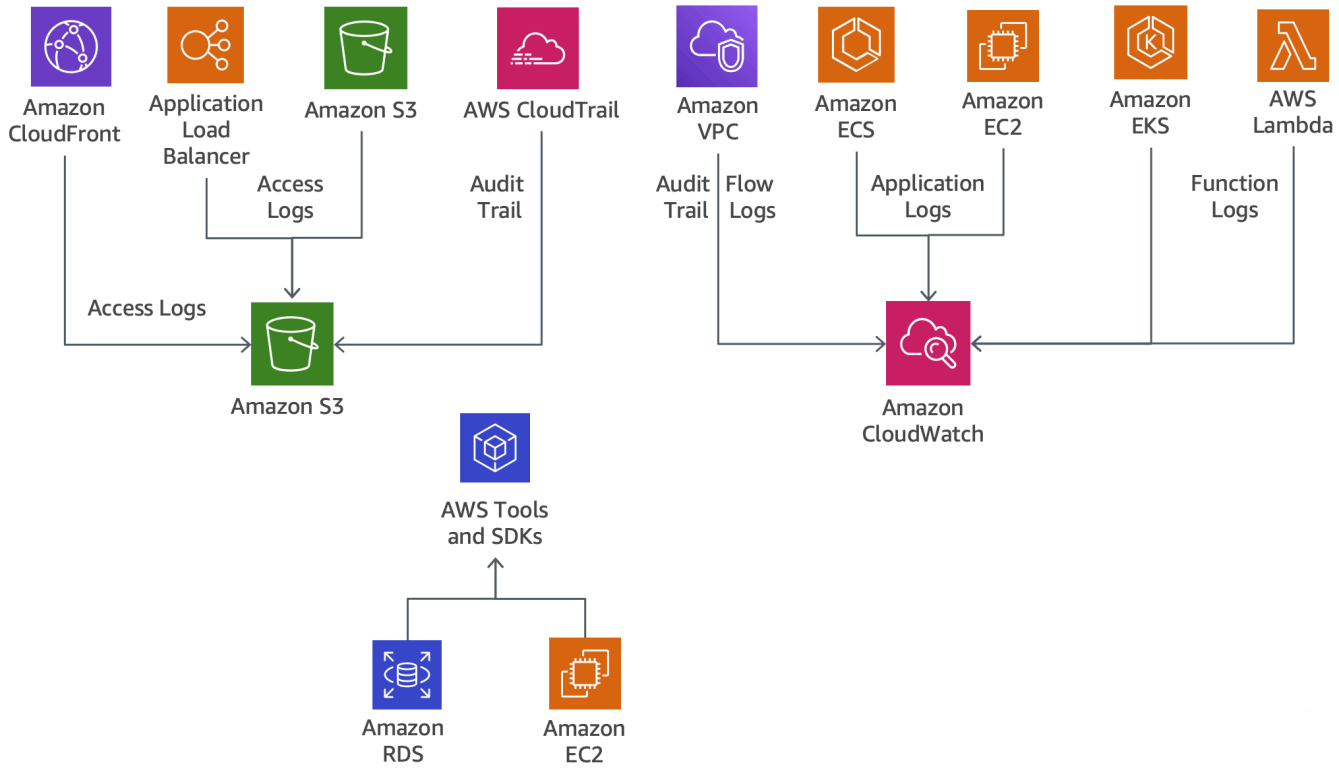

圖 12：記錄 AWS 服務的功能

分散式追蹤

微服務通常會共同處理請求。 AWS X-Ray 會使用相互關聯 IDs來追蹤這些服務的請求。X-Ray 可與 Amazon EC2、Amazon ECS、Lambda 和 Elastic Beanstalk 搭配使用。

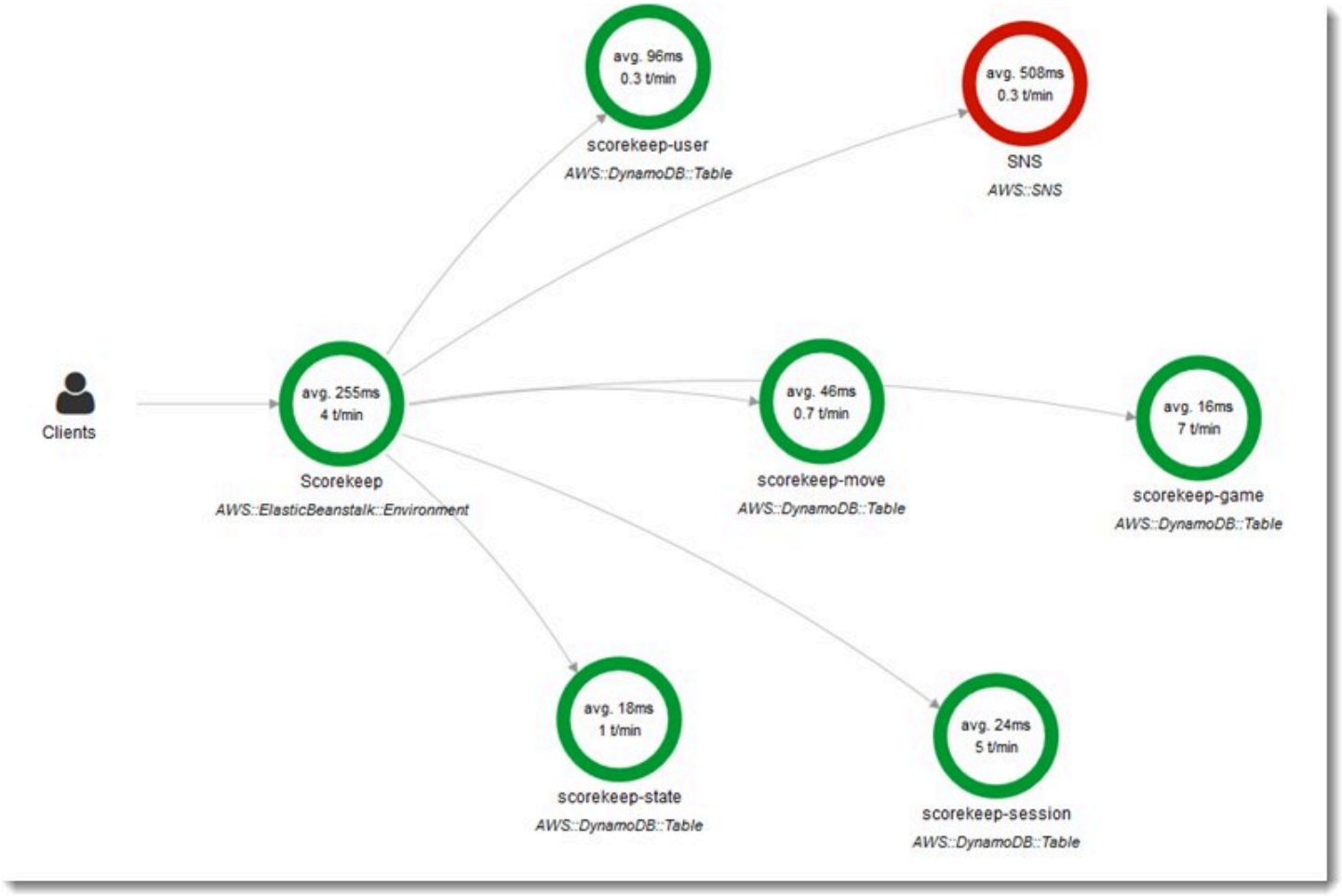

圖 13： AWS X-Ray 服務映射

AWS Distro for OpenTelemetry 是 OpenTelemetry 專案 的一部分，提供開放原始碼 APIs 和代理程式來收集分散式追蹤和指標，並改善您的應用程式監控。 它會將指標和追蹤傳送至多個 AWS 和合作夥伴監控解決方案。透過從您的 AWS 資源收集中繼資料，它將應用程式效能與基礎基礎設施資料保持一致，從而加速解決問題。此外，它與各種 AWS 服務相容，可用於內部部署。

上的日誌分析 AWS

Amazon CloudWatch Logs Insights 允許即時日誌探索、分析和視覺化。如需進一步的日誌檔案分析，包括 Kibana 的 Amazon OpenSearch Service 是功能強大的工具。CloudWatch Logs 可以即時將日誌

項目串流至 OpenSearch Service。Kibana 與 OpenSearch 無縫整合，可視覺化此資料並提供直覺式搜尋界面。

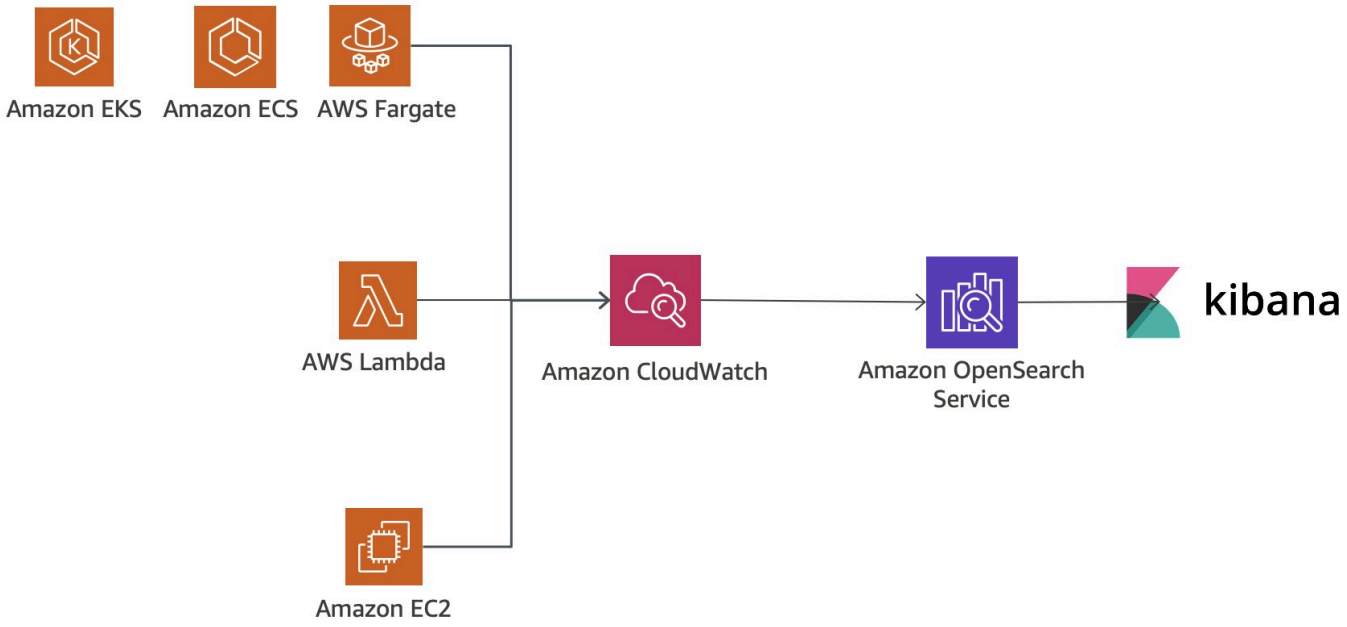

圖 14：使用 Amazon OpenSearch Service 進行日誌分析

其他分析選項

如需進一步的日誌分析，全受管資料倉儲服務 Amazon Redshift 和可擴展的商業智慧服務 Quick 提供有效的解決方案。QuickSight 可輕鬆連線至各種 AWS 資料服務，例如 Redshift、RDS、Aurora、EMR、DynamoDB、Amazon S3 和 Kinesis，簡化資料存取。

CloudWatch Logs 可以將日誌項目串流至 Amazon Data Firehose，這是一種提供即時串流資料的服務。然後，QuickSight 會使用存放在 Redshift 中的資料進行全面分析、報告和視覺化。

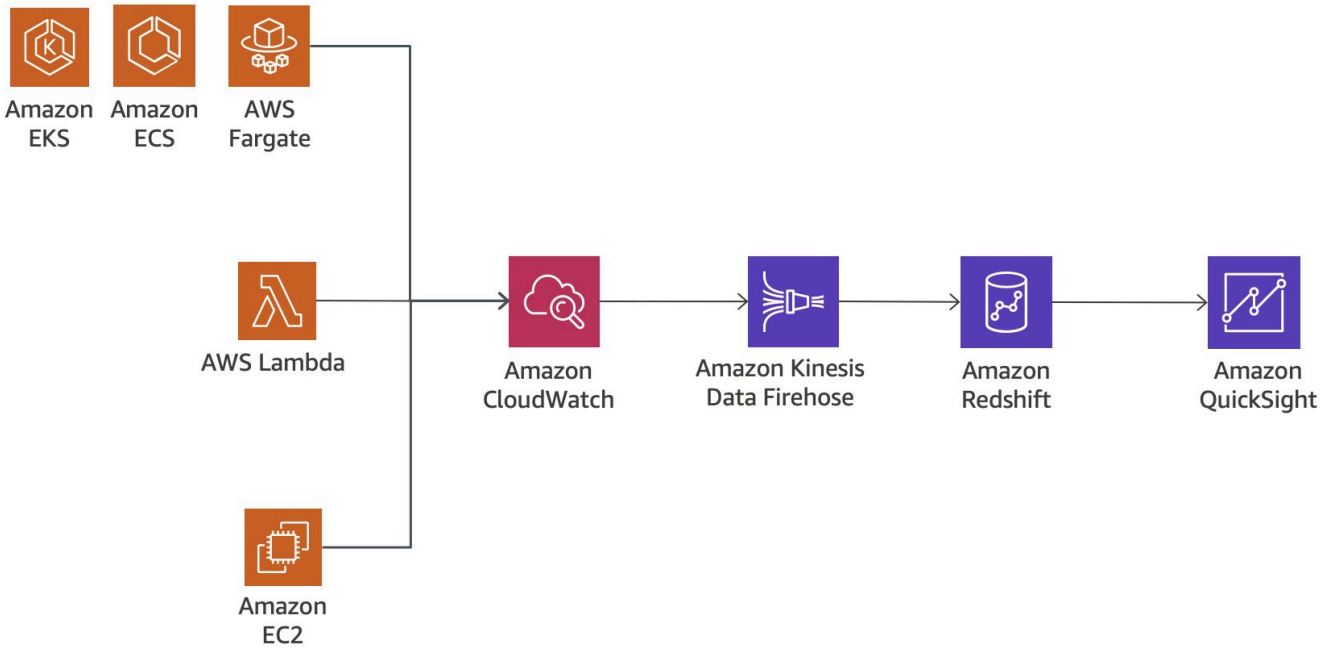

圖 15：使用 Amazon Redshift 和 Quick 進行日誌分析

此外，當日誌存放在物件儲存服務的 S3 儲存貯體時，資料可以載入到 Redshift 或 EMR 等服務，這是雲端型大數據平台，允許對儲存的日誌資料進行徹底分析。

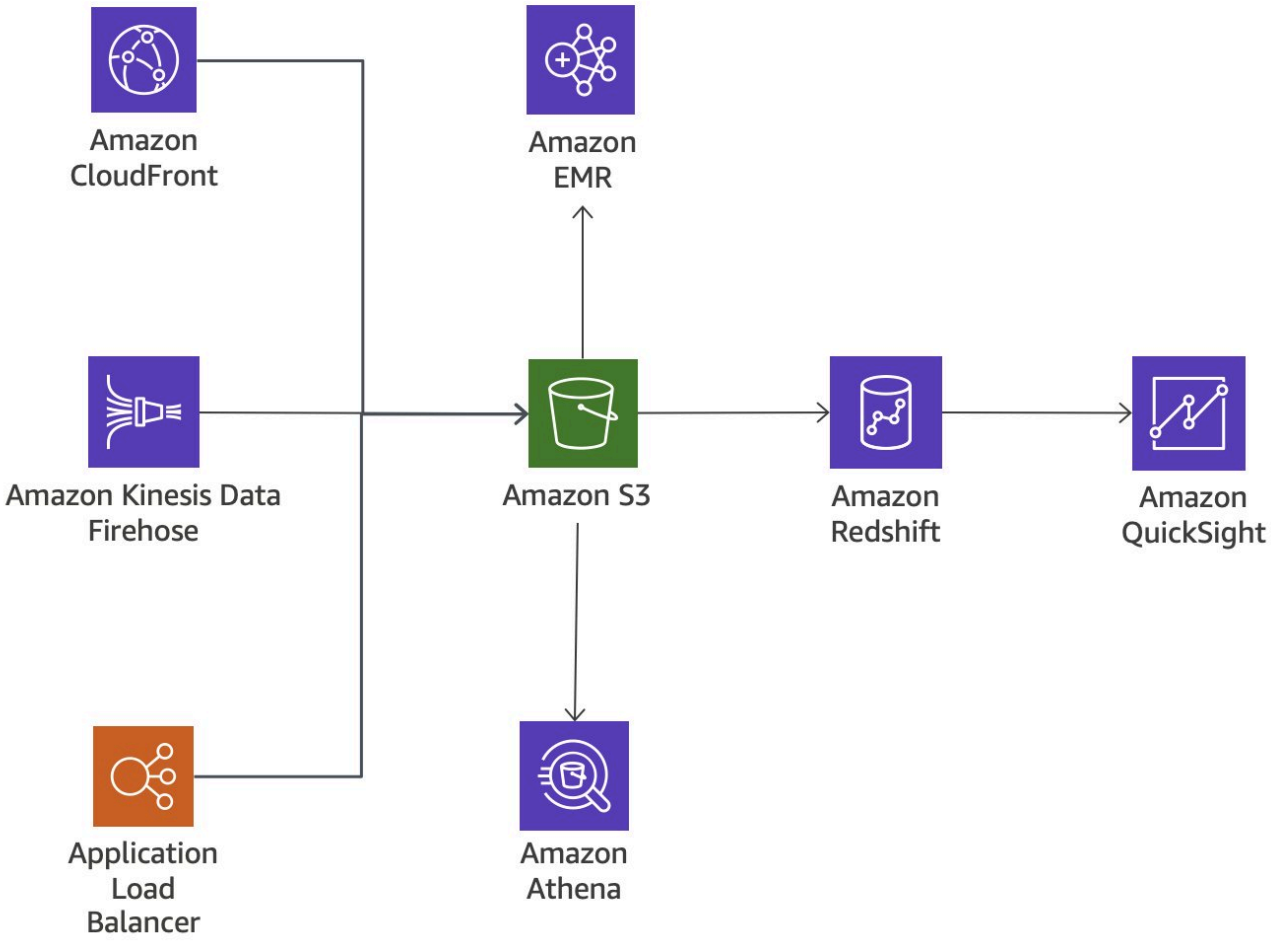

圖 16：簡化日誌分析：從 AWS 服務到 QuickSight

管理微服務通訊中的雜音

匱乏是指微服務之間的過度通訊，這可能會導致因網路延遲增加而導致效率低下。對於功能良好的系統，有效管理聊天至關重要。

管理雜亂的一些關鍵工具是 REST APIs、HTTP APIs 和 gRPC APIs。REST APIs提供各種進階功能，例如 API 金鑰、每個用戶端限流、請求驗證、 AWS WAF 整合或私有 API 端點。HTTP APIs的設計具有最少的功能，因此價格較低。如需本主題的詳細資訊，以及 REST APIs 和 HTTP APIs 中可用的核心功能清單，請參閱在 REST APIs和 HTTP APIs 之間進行選擇。

通常，微服務透過 HTTP 使用 REST 進行通訊，因為其廣泛使用。但在大量情況下，REST 的額外負荷可能會導致效能問題。這是因為通訊使用 TCP 交握，這是每個新請求的必要條件。在這種情況下，gRPC API 是更好的選擇。gRPC 可降低延遲，因為它允許透過單一 TCP 連線進行多個請求。gRPC 也支援雙向串流，允許用戶端和伺服器同時傳送和接收訊息。這會導致更有效率的通訊，特別是針對大型或即時資料傳輸。

如果儘管選擇正確的 API 類型仍持續忙碌，則可能需要重新評估您的微服務架構。整合服務或修訂您的網域模型，可以減少聊天並提高效率。

使用通訊協定和快取

微服務通常使用 gRPC 和 REST 等通訊協定進行通訊（請參閱上一節。）通訊機制gRPC 使用 HTTP/2 進行傳輸，而 REST 通常使用 HTTP/1.1。gRPC 使用通訊協定緩衝區進行序列化，而 REST 通常使用 JSON 或 XML。為了減少延遲和通訊額外負荷，可以套用快取。Amazon ElastiCache 或 API Gateway 中的快取層等服務有助於減少微服務之間的呼叫次數。

稽核

在微服務架構中，了解所有 服務的使用者動作至關重要。 AWS 提供 等工具 AWS CloudTrail，可記錄所有在 中進行的 API 呼叫 AWS，以及用於擷取應用程式日誌的 AWS CloudWatch。這可讓您追蹤變更並分析微服務的行為。Amazon EventBridge 可以快速回應系統變更，通知適當的人員，甚至自動啟動工作流程以解決問題。

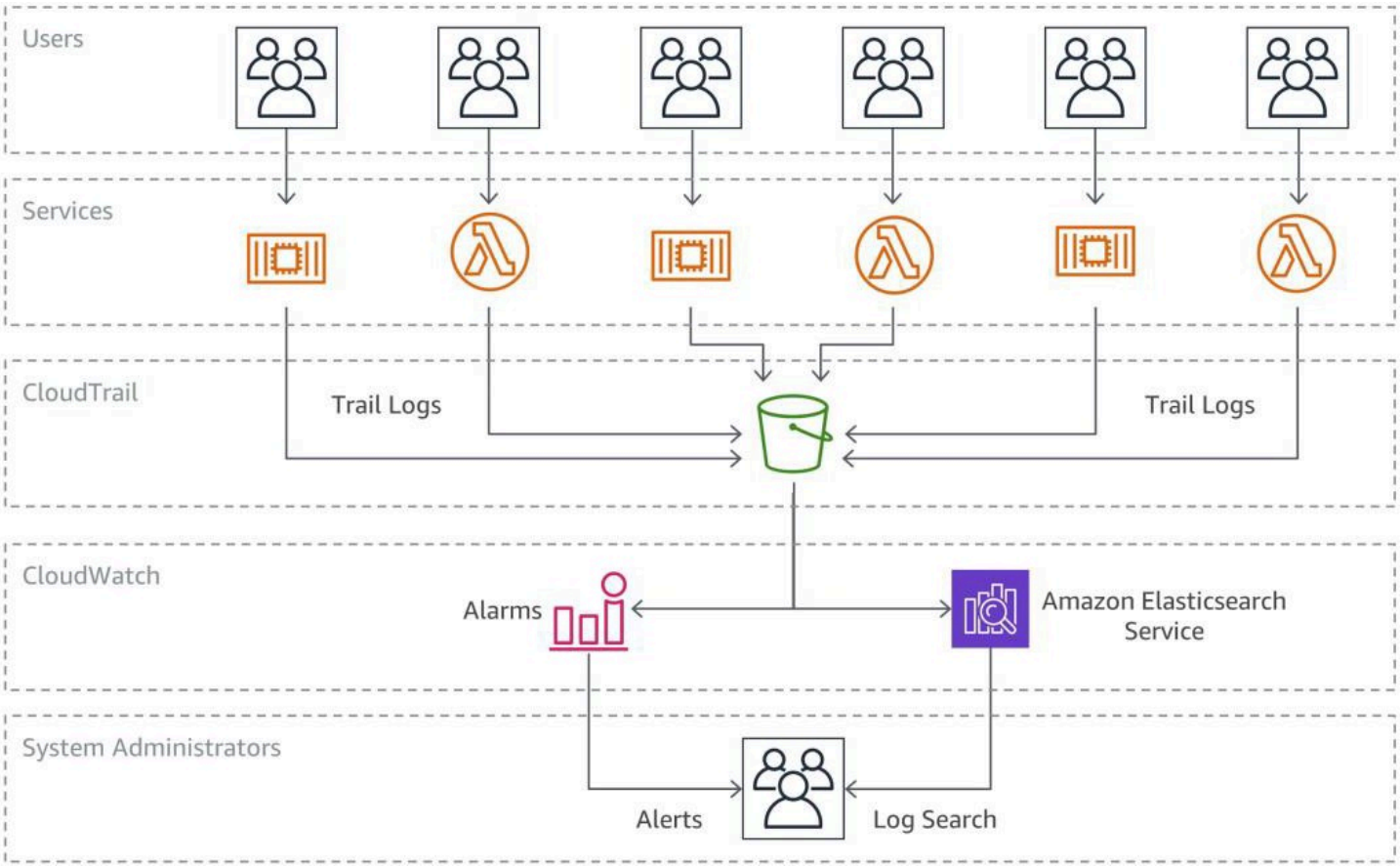

圖 17：跨微服務進行稽核和修復

資源庫存和變更管理

在具有快速發展基礎設施組態的敏捷開發環境中，自動化稽核和控制至關重要。 AWS Config 規則 提供受管方法來監控微型服務之間的這些變更。它們會啟用特定安全政策的定義，以自動偵測、追蹤和傳送政策違規的提醒。

例如，如果微服務中的 API Gateway 組態被更改為接受傳入 HTTP 流量，而不是僅接受 HTTPS 請求，則預先定義的 AWS Config 規則可以偵測此安全違規。它會記錄稽核的變更，並觸發 SNS 通知，還原合規狀態。

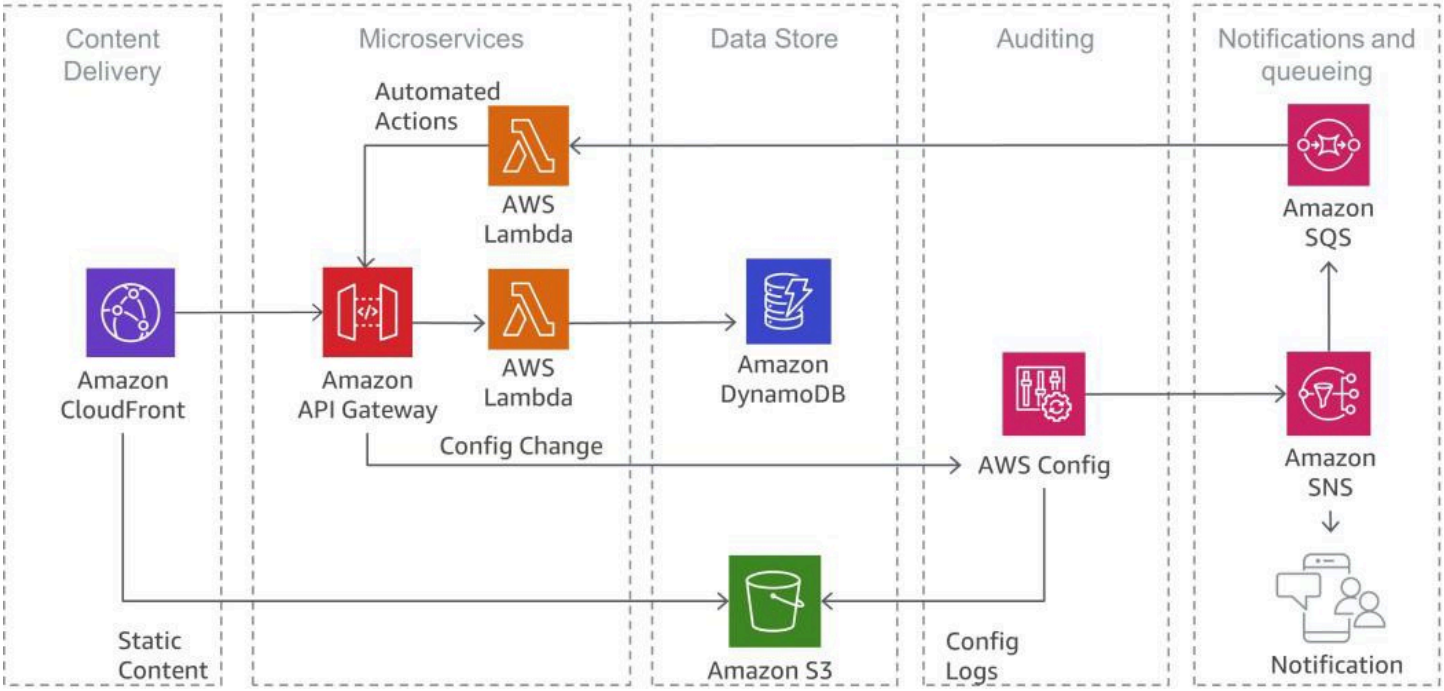

圖 18：使用 偵測安全違規 AWS Config

結論

Microservices 架構是一種多功能設計方法，可提供傳統單體系統的替代方案，協助擴展應用程式、提高開發速度和促進組織成長。藉助其適應性，可以使用容器、無伺服器方法或兩者的混合來實作，以根據特定需求量身訂做。

不過，它不是one-size-fits-all。考慮到架構複雜性和操作需求的潛在增加，每個使用案例都需要精細的評估。但是，當策略上接近時，微型服務的好處可能會明顯超過這些挑戰。關鍵在於主動規劃，特別是在可觀測性、安全性和變更管理方面。

也請務必注意，除了微服務之外，還有完全不同的架構架構，例如 Retrieval Augmented Generation (RAG) 等生成式 AI 架構，提供各種選項以滿足您的需求。

AWS及其強大的受管服務套件，可讓團隊建置高效的微服務架構，並有效將複雜性降至最低。本白皮書旨在引導您完成相關 AWS 服務和實作金鑰模式。目標是讓您具備知識，以利用微服務的力量 AWS，讓您能夠利用其優勢並改變您的應用程式開發旅程。

貢獻者

下列個人和組織為本文件作出了貢獻：

- Sascha Möllering，解決方案架構，Amazon Web Services
- Amazon Web Services 解決方案架構，Christian Müller
- Amazon Web Services 解決方案架構 Matthias Jung
- Peter Dalbhanjan，Amazon Web Services 解決方案架構
- Peter Chapman，Amazon Web Services 解決方案架構
- Christoph Kassen，Amazon Web Services 解決方案架構
- Umair Ishaq，Amazon Web Services 解決方案架構
- Rajiv Kumar，Amazon Web Services 解決方案架構
- Ramesh Dwarakanath，Amazon Web Services 解決方案架構
- Andrew Watkins，Amazon Web Services 解決方案架構
- Yannstoneman，Amazon Web Services 解決方案架構
- Mainak Chaudhuri，Amazon Web Services 解決方案架構
- Gaurav Acharya，Amazon Web Services 解決方案架構

文件歷史記錄

若要收到此白皮書更新的通知，請訂閱 RSS 摘要。

變更	描述	日期
主要更新	新增有關 AWS Customer Carbon Footprint Tool、Amazon EventBridge、 AWS AppSync (GraphQL)、 AWS Lambda Layers、Lambda SnapStart、大型語言模型 (LLMs)、Amazon Managed Streaming for Apache Kafka (MSK)、Amazon Managed Workflows for Apache Airflow (MWAA)、Amazon VPC Lattice、 AWS AppConfig 的資訊。新增了有關成本最佳化和永續性的個別章節。	2023 年 7 月 31 日
次要更新	已將 Well-Architected 新增至摘要。	2022 年 4 月 13 日
白皮書已更新	整合 Amazon EventBridge、AWS OpenTelemetry、AMP、AMG、Container Insights、次要文字變更。	2021 年 11 月 9 日
次要更新	調整後的頁面配置	2021 年 4 月 30 日
次要更新	次要文字變更。	2019 年 8 月 1 日
白皮書已更新	整合 Amazon EKS、AWS Fargate、Amazon MQ、AWS PrivateLink、AWS App Mesh、AWS Cloud Map	2019 年 6 月 1 日

白皮書已更新	整合 AWS Step Functions 、AWS X-Ray 和 ECS 事件串流。	2017 年 9 月 1 日
初次出版	在 AWS 上實作微服務已發佈 。	2016 年 12 月 1 日

> **Note**
>
> 若要訂閱 RSS 更新，您必須為正在使用的瀏覽器啟用 RSS 外掛程式。

注意

客戶有責任對本文件中的資訊進行自己的獨立評定。本文件：(a) 僅供參考，(b) 代表目前的 AWS 產品產品和實務，這些產品和實務可能隨時變更，恕不另行通知，且 (c) 不會從 AWS 及其附屬公司、供應商或授權方建立任何承諾或保證。 AWS 產品或服務「原樣」提供，而無任何明示或暗示的保證、陳述或條件。 AWS 對其客戶的責任和責任由 AWS 協議控制，本文件不屬於 AWS 與其客戶之間的任何協議，也未對其進行修改。

AWS 詞彙表

如需最新的 AWS 術語，請參閱 AWS 詞彙表 參考中的AWS 詞彙表。

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。