



模組 13：構建微服務和無伺服器架構

AWS Academy Cloud Architecting

© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

歡迎學習模組 13：構建微服務和無伺服器架構。

模組概覽

章節

1. 架構需求
2. 介紹微服務
3. 使用 AWS 容器服務構建微服務應用程式
4. 介紹無伺服器架構
5. 使用 AWS Lambda 構建無伺服器架構
6. 使用 Amazon API Gateway 擴展無伺服器架構
7. 使用 AWS Step Functions 編排微服務

演示

- 創建 AWS Lambda 函數
- 結合使用 AWS Lambda 和 Amazon S3

實驗

- (可選) 指導實驗 1：將整體式 Node.js 應用程式拆分為微服務
- 指導實驗 2：在 AWS 上實施無伺服器架構
- 挑戰實驗：為咖啡館實施無伺服器架構



知識考核



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

2

本模組包括以下章節：

架構需求

1. 介紹微服務
2. 使用 AWS 容器服務構建微服務應用程式
3. 介紹無伺服器架構
4. 使用 AWS Lambda 構建無伺服器架構
5. 使用 Amazon API Gateway 擴展無伺服器架構
6. 使用 AWS Step Functions 編排微服務

本模組還包括：

- 兩個 AWS Lambda 演示
- 一個可選的指導實驗，您可以將整體式應用程式重構為微服務
- 一個指導實驗，您可以使用 Amazon S3、AWS Lambda、Amazon DynamoDB 和 Amazon SNS 在 AWS 上實施無伺服器架構
- 一個挑戰實驗，您可以使用 AWS Lambda 和 Amazon Simple Notification Service (Amazon SNS) 生成並發送咖啡館的每日銷售報告。

最後，您需要完成一個知識考核，以測試您對本模組中涵蓋的關鍵概念的理解程度。

模組目標

學完本模組後，您應該能夠：

- 指出微服務的特性
- 將整體式應用程式重構為微服務，然後使用 Amazon ECS 部署容器化微服務
- 解釋無伺服器架構的含義
- 使用 AWS Lambda 實施無伺服器架構
- 描述 Amazon API Gateway 的通用架構
- 描述 AWS Step Functions 支持的工作流類型



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

3

學完本模組後，您應該能夠：

- 指出微服務的特性
- 將整體式應用程式重構為微服務，然後使用 Amazon ECS 部署容器化微服務
- 解釋無伺服器架構的含義
- 使用 AWS Lambda 實施無伺服器架構
- 描述 Amazon API Gateway 的通用架構
- 描述 AWS Step Functions 支持的工作流類型

第 1 節：架構需求

模組 13：構建微服務和無伺服器架構

 AWS

© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

咖啡館業務需求

咖啡館希望通過電子郵件獲得有關網站上所有訂單的每日報告。他們想要獲取這些資訊，以便能夠預測需求，然後烘焙正確數量的甜點（減少浪費）。他們還希望識別業務中的任何模式（分析）。



© 2023, Amazon Web Services, Inc.或其聯屬公司。保留所有權利。

5

Frank 和 Martha 希望通過電子郵件獲得有關網站上所有訂單的每日報告。Frank 希望能夠預測需求，以便烘焙正確數量的甜點（減少浪費）。Martha 希望識別咖啡館業務中的任何模式（分析）。目前，Sofía 已經在 Web 伺服器實例上設置了 cron 作業，將這些每日訂單報告電子郵件消息發送給 Frank 和 Martha。但是，cron 作業為資源密集型作業，會降低 Web 伺服器的性能。

Olivia 建議 Sofía 和 Nikhil 將非業務關鍵型報告任務分離開。Sofía 和 Nikhil 希望進一步解耦架構，並將 cron 作業移動到能夠很好地擴展且能夠降低成本的託管式無伺服器環境中。

第 2 節：介紹微服務

模組 13：構建微服務和無伺服器架構

 AWS

© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

什麼是微服務？

應用程式由獨立的服務組成，這些服務通過明確定義的 API 通信



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

7

微服務是一種開發軟體的架構和組織方法，採用這種方法開發出的應用程式由獨立的服務組成，這些服務通過明確定義的 Application Programming Interface (API) 進行通信。這種方法旨在加快部署週期。

微服務方法促進了創新並明確了所有權，還提高了軟體應用程式的可維護性和可擴展性。

整體式應用程式與微服務應用程式的對比

整體式應用程式



AWS

© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

微服務應用程式



8

要瞭解微服務的益處，首先要瞭解整體式應用程式。

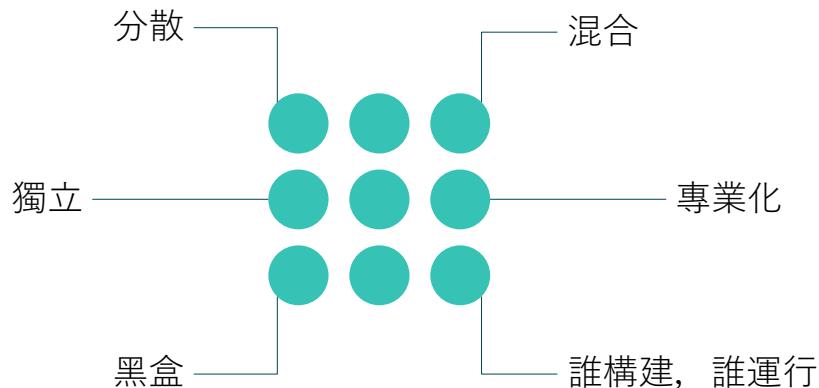
在左側的示例中，整體式論壇應用程式的三個進程（使用者、主題和消息）緊耦合。它們一起作為單一服務運行。如果應用程式的一個進程遇到需求峰值，則必須擴展整個架構。隨著代碼庫的增長，添加或改進功能變得更加複雜，這不僅限制了試驗，還讓實現新創意變得困難。整體式應用程式的可用性也會有風險，因為許多相互依賴、緊耦合的進程會放大單個進程故障帶來的影響。

現在，假設同一個應用程式在微服務架構中運行。應用程式的每個進程都構建為獨立的元件，作為服務運行。服務通過使用羽量級 API 操作進行通信。每個服務執行一項可支援多個應用程式的功能。由於服務獨立運行，因此可以更新、部署和擴展這些服務，以滿足應用程式特定功能的需求。

微服務架構提供更快的反覆運算、自動化和整體敏捷性。支持快速啟動、快速失敗和快速恢復。

有關 AWS 上微服務的概覽，請參閱[什麼是微服務？](#)

微服務的特性



微服務具有一些共同的特性：

- 分散 – 微服務架構是採用分散式資料管理的分散式系統。它們不依賴中央資料庫中的統一架構。每個微服務都有自己對資料模型的看法。微服務的開發、部署、管理和運行方式也是分散式的。
- 獨立 – 微服務架構中的每個元件服務都可以獨立進行更改、升級或替換，而不會影響其他服務的功能。這些服務不需要與其他服務共用任何代碼或實施。同樣，負責不同微服務的團隊可以彼此獨立行事。
- 專業化 – 每項元件服務都專為實現一組功能而設計，並專注於特定領域。如果特定元件服務的代碼達到一定的複雜程度，則該服務可以拆分為兩個或更多服務。
- 混合 – 微服務並不遵循單一的方法。團隊可以自由選擇最佳工具來解決他們的具體問題。因此，微服務架構在作業系統、程式設計語言、資料存儲和工具方面採用了異構方法。這種方法被稱為混合持久性和程式設計。
- 黑盒 – 單個元件服務被設計為黑盒，這意味著其複雜性的細節對其他組件隱藏不見。服務之間的任何通信都通過明確定義的 API 進行，以防止隱式和隱藏的依賴項。
- 誰構建，誰運行 – DevOps 是微服務的關鍵組織原則，負責構建服務的團隊還負責在生產中運行和維護服務。

第 2 節要點



- 微服務應用程式由獨立的服務組成，這些服務通過明確定義的 API 通信
- 微服務具有以下特性 -
 - 分散
 - 獨立
 - 專業化
 - 混合
 - 黑盒
 - 誰構建，誰運行

本模組中這節內容的要點包括：

- 微服務應用程式由獨立的服務組成，這些服務通過明確定義的 API 通信
- 微服務具有以下特性 -
 - 分散：微服務的開發、部署、管理和運行方式是分散式的
 - 獨立：微服務架構中的每個元件服務都可以在不影響其他服務功能的情況下進行開發、部署、運行和擴展
 - 專業化：每項元件服務都專為實現一組功能而設計，側重於解決特定問題
 - 混合：微服務架構在作業系統、程式設計語言、資料存儲和工具方面採用異構方法
 - 黑盒：微服務元件複雜性的細節對其他元件隱藏不見
 - 誰構建，誰運行：DevOps 是微服務的關鍵組織原則

第 3 節：使用 AWS 容器服務構建微服務應用程式

模組 13：構建微服務和無伺服器架構

aws

© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

介紹第 3 節：使用 AWS 容器服務構建微服務應用程式。

什麼是容器？

您的容器



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

12

構建微服務架構時，可以使用容器來提供處理能力。

容器是實現作業系統虛擬化的一種途徑，讓您可以在資源受到隔離的進程中運行應用程式及其依賴項。容器是一個羽量級的獨立套裝軟體。它包含軟體應用程式運行所需的一切，例如應用程式碼、運行時引擎、系統工具、系統庫和配置。

容器解決的問題

讓軟體在不同工作環境中可靠運行



開發人員的工作站



生產環境



測試環境

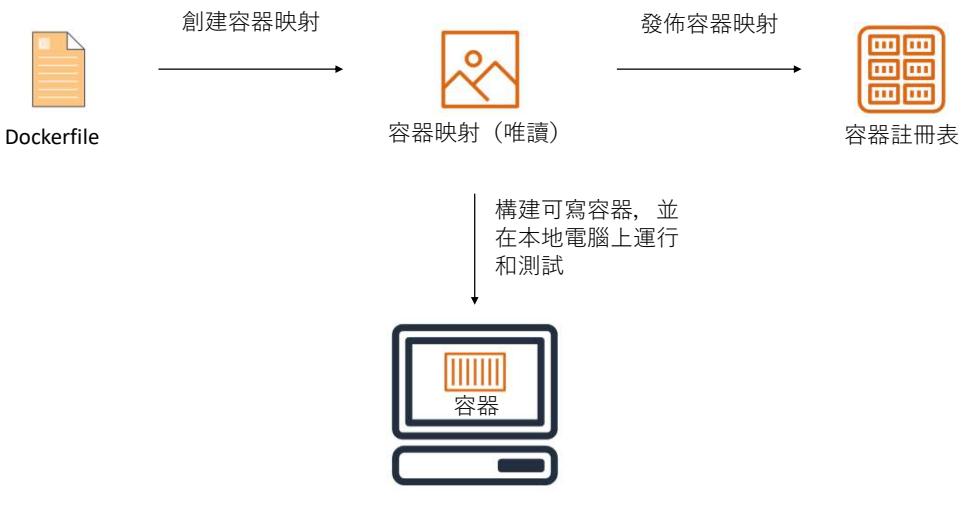


© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

13

容器有助於確保快速、可靠且一致地部署應用程式，不受部署環境的影響。容器還可以讓您更精細地控制資源，提高您的基礎設施效率。

容器術語



容器是通過一種稱為映射的唯讀範本創建的。映射通常是利用 Dockerfile 構建出來的，後者是一個純文字檔，指定容器中包含的所有元件。您既可以從頭開始創建映射，也可以使用其他人創建並發佈到公有或私有容器註冊表的映射。

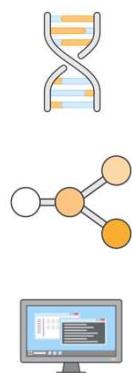
容器映射是容器可用的檔案系統的快照。例如，您可能使用 Debian 作業系統作為容器映射。當您運行這個容器時，它就可以使用 Debian 作業系統。您還可以將所有代碼依賴項打包到容器映射中，並將其用作代碼構件。

容器映射存儲在註冊表中。您可以從註冊表下載映射並在集群上運行它們。註冊表可以存在於您的 AWS 基礎設施內部或外部。

Amazon ECS



Amazon Elastic
Container Service
(Amazon ECS)



編排容器的執行時間

維護和擴展運行容器的實例併列

消除構建基礎設施的複雜性



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

15

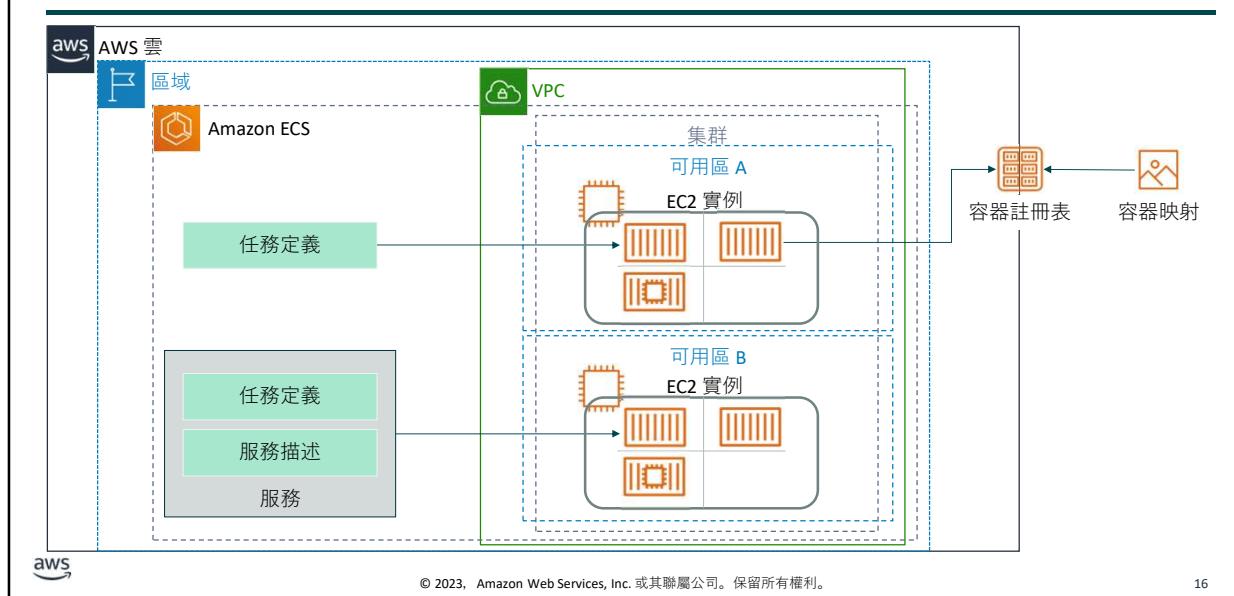
您可以在 Amazon Elastic Container Service (Amazon ECS) 上運行容器。Amazon ECS 是一項高度可擴展的高性能容器管理服務。它支持 Docker 容器，使您能夠在 Amazon Elastic Compute Cloud (Amazon EC2) 實例的託管集群上輕鬆運行應用程式。

Amazon ECS 是一種可擴展的集群服務，用於託管容器：

- 可以在數秒內擴展到數以千計的 Docker 容器
- 監控容器部署
- 管理運行容器的集群的狀態
- 使用內置的調度器或協力廠商調度器（Apache Mesos、Blox）對容器進行調度
- 可使用 API 進行擴展
- 可使用 AWS Fargate 或 Amazon EC2 [啟動類型](#) 啟動

通過將 Spot 實例與按需實例和預留實例混合使用，可以大規模運行 ECS 集群。

Amazon ECS 編排容器



Amazon ECS 是一項區域服務，可在區域內的多個可用區中以高度可用的方式簡化運行應用程式容器。您可以在新的或現有的 Virtual Private Cloud (VPC) 中創建 ECS 集群。集群是資源的邏輯分組。

集群啟動並運行後，您可以定義任務定義和服務，指定在集群中運行哪些 Docker 容器映射。

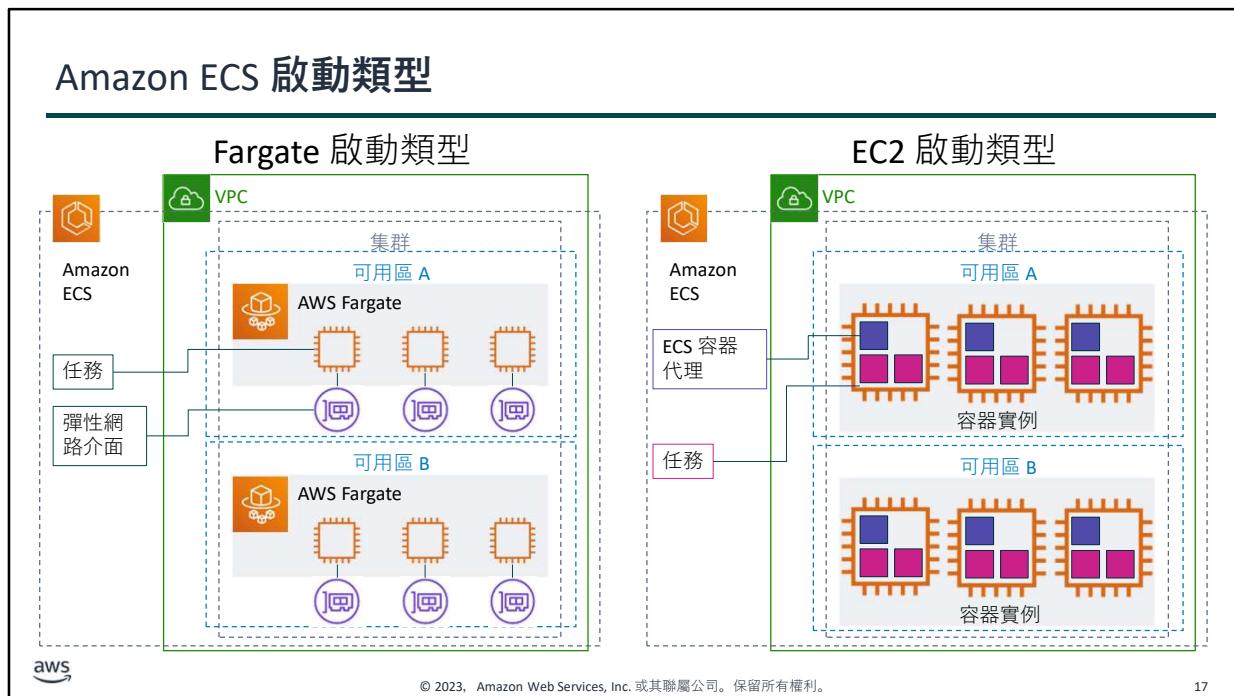
任務定義是 JavaScript 物件標記法 (JSON) 格式的文字檔。它描述構成應用程式的一個或多個容器，最多 10 個。您可以將它視為應用程式的藍圖。任務定義指定應用程式的參數 - 例如，要使用的容器和啟動類型。其他參數包括應為應用程式打開哪些埠，以及任務中的容器應使用哪些資料卷。

服務讓您能夠指定要在集群中運行並維護多少個任務定義副本。您可以選擇使用彈性負載均衡負載等化器將傳入流量分配給服務中的容器。Amazon ECS 保持該數量的任務，並使用負載等化器來協調任務調度。

在為應用程式創建任務定義後，您可以指定將在集群上運行的任務的數量。任務是集群內的任務定義的產生實體。當您使用 ECS 運行任務時，可以將它們放在集群中。

Amazon ECS 從您指定的註冊表中下載容器映射，並在集群內運行這些映射。

Amazon ECS 啟動類型



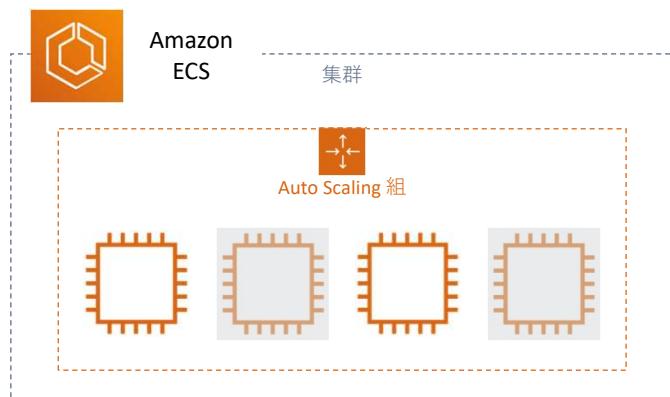
Amazon ECS 提供兩種啟動類型，用於託管容器化應用程式。

您可以使用 Fargate 啓動類型在 Amazon ECS 管理的無伺服器基礎設施上託管集群。您只需將應用程式打包到容器中，指定 CPU 和記憶體要求，定義聯網和 AWS Identity and Access Management (IAM) 策略，然後啟動應用程式即可。

或者，如果您想要更多的控制權，可以使用 EC2 啓動類型在您管理的 EC2 容器實例集群上託管任務。容器實例是運行 Amazon ECS 容器代理的 EC2 實例。您可以使用 Amazon ECS，根據資源需求、隔離策略和可用性要求安排集群中的容器放置。有關不同調度選項的資訊，請參閱[調度 Amazon ECS 任務](#)。Amazon ECS 會跟蹤集群中的所有 CPU、記憶體和其他資源。它還會根據您指定的資源需求為容器找到最佳的運行伺服器。

有關 Fargate 和 EC2 啓動類型的更多資訊，請參閱[Amazon ECS 啟動類型](#)。

Amazon ECS 集群彈性伸縮



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

18

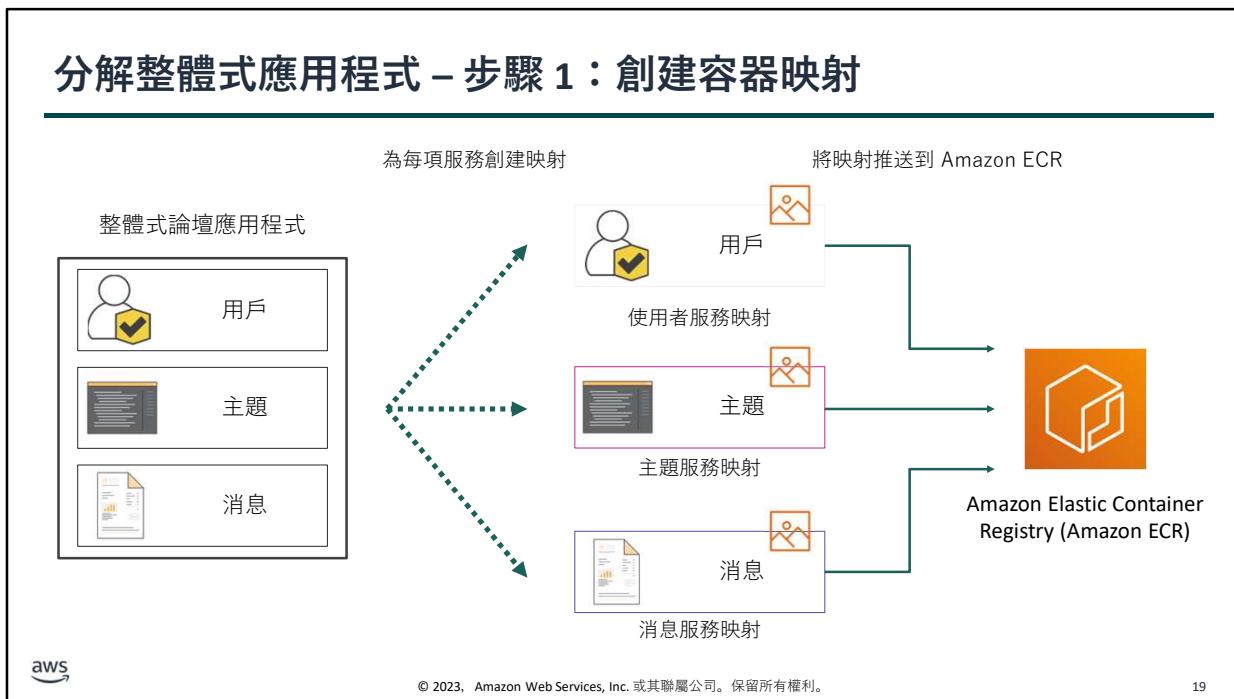
您可以為 [Amazon ECS 集群創建 Auto Scaling 組](#)。Auto Scaling 組包含您可以使用 Amazon CloudWatch 警報擴展（和縮減）的容器實例。如果您將 Auto Scaling 組配置為刪除容器實例，則任何正在已刪除容器實例上運行的任務都將停止。如果您的任務作為服務的一部分運行，則 Amazon ECS 會在所需資源可用的情況下在另一個實例上重新開機這些任務。此類所需資源的示例包括 CPU、記憶體和埠。不過，手動啟動的任務不會自動重新啟動。

您還可以利用 [Amazon ECS 集群彈性伸縮](#)，這使您可以更好地控制集群中的任務擴展方式。它提高了集群擴展的速度和可靠性。它使您能夠控制集群中維護的空閒容量，並在縮減時自動管理實例終止。

通過集群彈性伸縮，您可以配置 Amazon ECS 以自動縮減和擴展 Auto Scaling 組。集群彈性伸縮依賴於容量提供程式，這些提供程式將 ECS 集群連結到您想要使用的 Auto Scaling 組。每個 Auto Scaling 組都與一個容量提供程式關聯，並且每個容量提供程式只有一個 Auto Scaling 組。但是，許多容量提供程式可以與一個 ECS 集群相關聯。為了自動擴展整個集群，每個容量提供程式都會管理其關聯的 Auto Scaling 組的擴展。

有關集群彈性伸縮的更多資訊，請參閱 [Amazon ECS 集群彈性伸縮](#) AWS 新聞博客文章。

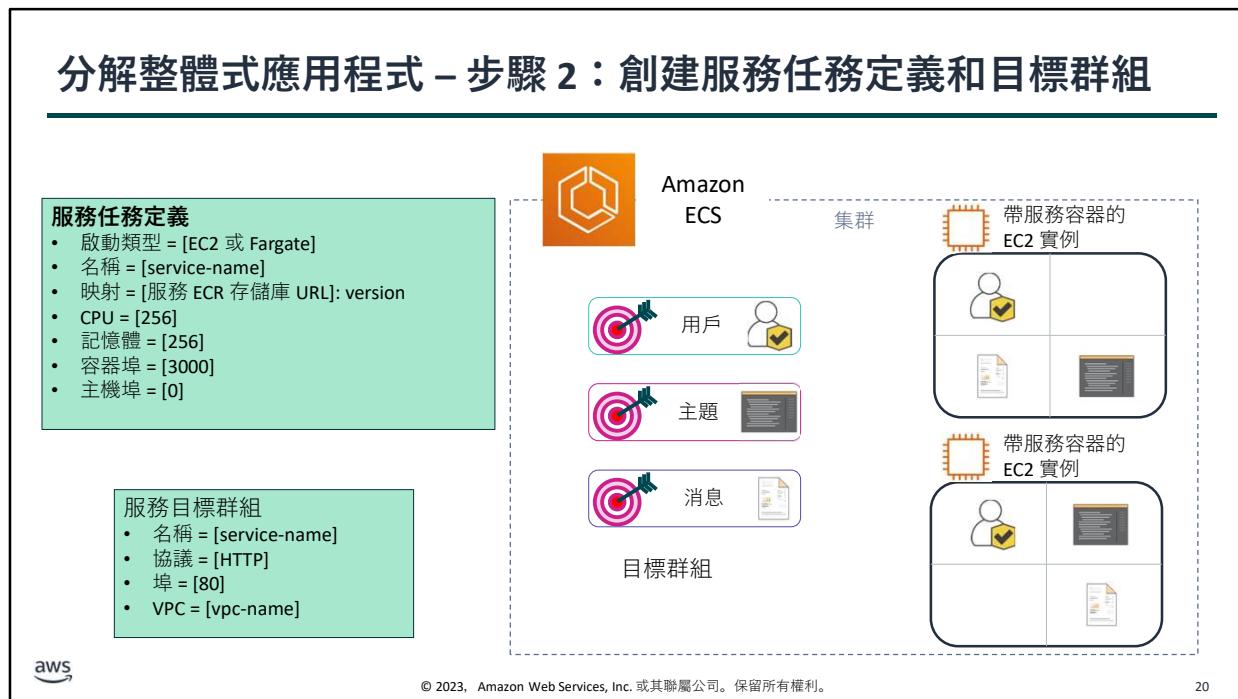
分解整體式應用程式 – 步驟 1：創建容器映射



再次考慮一下您之前看到的整體式論壇應用程式，其中整個應用程式作為單個服務運行。要使用微服務架構重新設計此應用程式的架構，您可以將每個應用程式進程作為單獨的服務在其自己的容器中運行。借助微服務架構，這些服務可以獨立於其他服務進行擴展和更新。

要將整體式應用程式部署為微服務應用程式，首先要為每項服務構建並標記映射。然後，在 Amazon Elastic Container Registry (Amazon ECR) 中註冊映射。

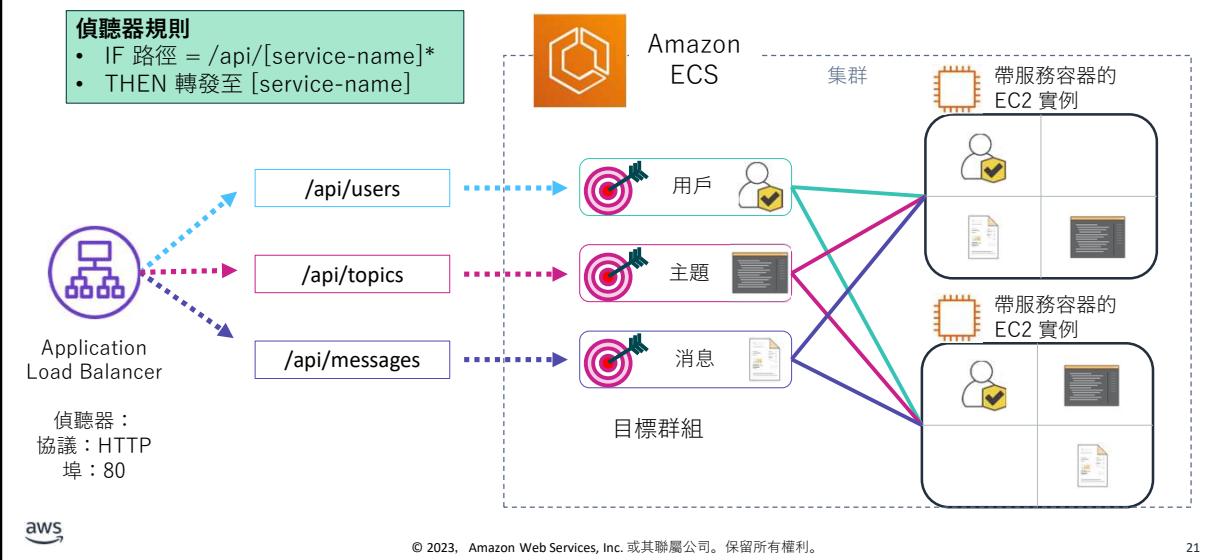
分解整體式應用程式 – 步驟 2：創建服務任務定義和目標群組



接下來，選擇啟動類型，然後為原始整體式應用程式的每個部分創建新服務。Amazon ECS 會在 ECS 集群上將每項服務部署到自己的容器中。

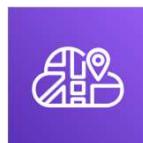
然後，為每項服務創建一個目標群組。目標群組可跟蹤為該服務運行的每個容器的實例和埠。

分解整體式應用程式 – 步驟 3：將負載等化器連接到服務



最後，創建 Application Load Balancer 並配置偵聽器規則以連接到服務。偵聽器會檢查傳入負載等化器的連接請求，並使用規則對流量進行適當路由。在本示例中，Application Load Balancer 的偵聽器會偵聽埠 80 上的 HTTP 服務請求並將其路由到相應的服務。

用於構建高度可用的微服務架構的工具



AWS Cloud
Map

- 是完全託管式的雲資源發現服務
- 可用於為應用程式資源定義自訂名稱
- 維護動態變化資源的更新位置，從而提高應用程式的可用性



AWS
App Mesh

- 捕獲來自所有微服務的指標、日誌和跟蹤
- 您可以將此類資料匯出到 Amazon CloudWatch、AWS X-Ray 以及相容的 AWS 合作夥伴網路 (APN) 合作夥伴和社群工具
- 您可以控制微服務之間的流量，以便確保服務的高可用性

AWS

© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

22

AWS Cloud Map 和 AWS App Mesh 這兩個工具，有助於您構建高度可用的微服務架構。

[AWS Cloud Map](#) 是一種完全託管式的雲資源發現服務。您可以使用此服務為應用程式資源（如資料庫、佇列、微服務和其他雲資源）定義自訂名稱。AWS Cloud Map 維護這些動態變化的資源的更新位置。這種位置維護可提高應用程式的可用性，因為 Web 服務總能發現其資源的最新位置。您只需對映射進行最少的人工干預，即可添加和註冊任何資源。AWS Cloud Map 可在微服務和應用程式的服務發現、持續集成和運行狀況監控方面提供幫助。

有關 AWS Cloud Map 的更多資訊，請閱讀此 [AWS 開源博客文章](#)。要進一步瞭解如何使用 AWS Cloud Map 啟用容器化服務以發現彼此並相互連接，請閱讀 [AWS Fargate](#)、[Amazon EKS](#) 和 [Amazon ECS 現已與 AWS Cloud Map 集成](#)。

創建任務定義時，可以啟用 App Mesh 集成。[AWS App Mesh](#) 可以捕獲所有微服務的指標、日誌和跟蹤。您可以將此類資料匯出到 Amazon CloudWatch、AWS X-Ray 以及相容的 AWS 合作夥伴網路 (APN) 合作夥伴和社群工具，以進行監控和跟蹤。AWS App Mesh 還能讓您控制微服務之間的流量流向，確保每項服務在部署期間、故障後以及應用程式的擴展時都高度可用。

App Mesh 可讓您配置微服務，通過代理直接相互連接，而無需在應用程式內編寫代碼或使用負載等化器。App Mesh 使用開源服務網格代理 Envoy，它與微服務容器一起部署。

有關 AWS Cloud Map 和 AWS App Mesh 的更多資訊，請參閱此 [AWS YouTube 視頻](#)。

AWS Fargate



- 是完全託管式容器服務
- 可與 [Amazon Elastic Container Service \(Amazon ECS\)](#) 和 [Amazon Elastic Kubernetes Service \(Amazon EKS\)](#) 配合使用
- 預置、管理和擴展容器集群
- 管理運行時環境
- 提供彈性伸縮



© 2023, Amazon Web Services, Inc.或其聯屬公司。保留所有權利。

23

在本節中，您瞭解到 Amazon ECS 提供兩種啟動類型：EC2 和 Fargate。

[AWS Fargate](#) 是一項完全託管式的容器服務，可與 Amazon ECS 和 Amazon Elastic Kubernetes Service (Amazon EKS) 配合使用。它使您無需管理伺服器或集群即可運行容器。借助 AWS Fargate，您不再需要預置、配置和擴展虛擬機器集群就可以運行容器。因此，您就不需要選擇伺服器類型，不需要決定在什麼時候擴展集群，也不需要優化集群包裝。AWS Fargate 讓您省去了考慮伺服器和集群以及與之交互的麻煩。Fargate 使您能夠專注於設計和構建應用程式，而不是管理運行應用程式的基礎設施。

第 3 節要點



aws

- [Amazon ECS](#) 是一項高度可擴展的高性能容器管理服務。它支援 Docker 容器，使您能夠在 Amazon EC2 實例的託管集群上輕鬆運行應用程式。
- [集群彈性伸縮](#)讓您能夠更好地控制集群內的任務擴展方式。
- [AWS Cloud Map](#) 使您能夠為應用程式資源定義自訂名稱。它維護這些動態變化的資源的更新位置。
- [AWS App Mesh](#) 是一種提供應用程式級聯網的服務網格。它使您的服務能夠在多種類型的計算基礎設施之間輕鬆地相互通信。
- [AWS Fargate](#) 是一項完全託管式容器服務，使您無需管理伺服器或集群即可運行容器。

© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

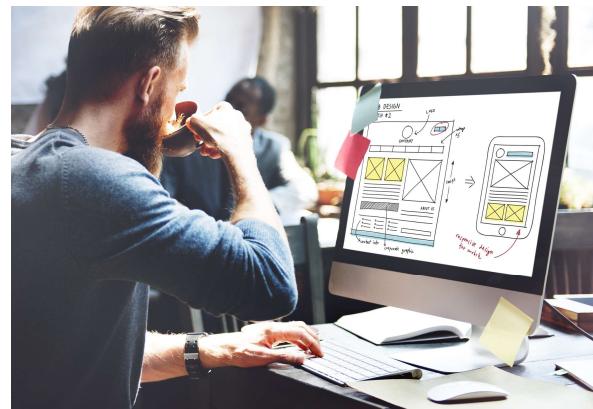
24

本模組中這節內容的要點包括：

- Amazon ECS 是一項高度可擴展的高性能容器管理服務。它支援 Docker 容器，使您能夠在 Amazon EC2 實例的託管集群上輕鬆運行應用程式。
- 集群彈性伸縮讓您能夠更好地控制集群內的任務擴展方式。
- AWS Cloud Map 使您能夠為應用程式資源定義自訂名稱。它維護這些動態變化的資源的更新位置。
- AWS App Mesh 是一種服務網格，可提供應用程式級聯網，讓您的服務輕鬆跨多種類型的計算基礎設施相互通信。
- AWS App Mesh 是一種提供應用程式級聯網的服務網格。它使您的服務能夠在多種類型的計算基礎設施之間輕鬆地相互通信。
- AWS Fargate 是一項完全託管式容器服務，使您無需管理伺服器或集群即可運行容器。

模組 13 – 指導實驗

1：將整體式 Node.js 應用程式 拆分為微服務 (可選實驗)



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

25

您可以選擇完成模組 13 – 指導實驗 1：將整體式 Node.js 應用程式拆分為微服務。本實驗為可選實驗。

指導實驗 1：任務

1. 準備 AWS Cloud9 開發環境
2. 在基本 Node.js 啟動器上運行整體式應用程式
3. 為 Amazon ECS 容器化整體式應用程式
4. 將整體式應用程式部署至 Amazon ECS
5. 將整體式應用程式重構為容器化微服務



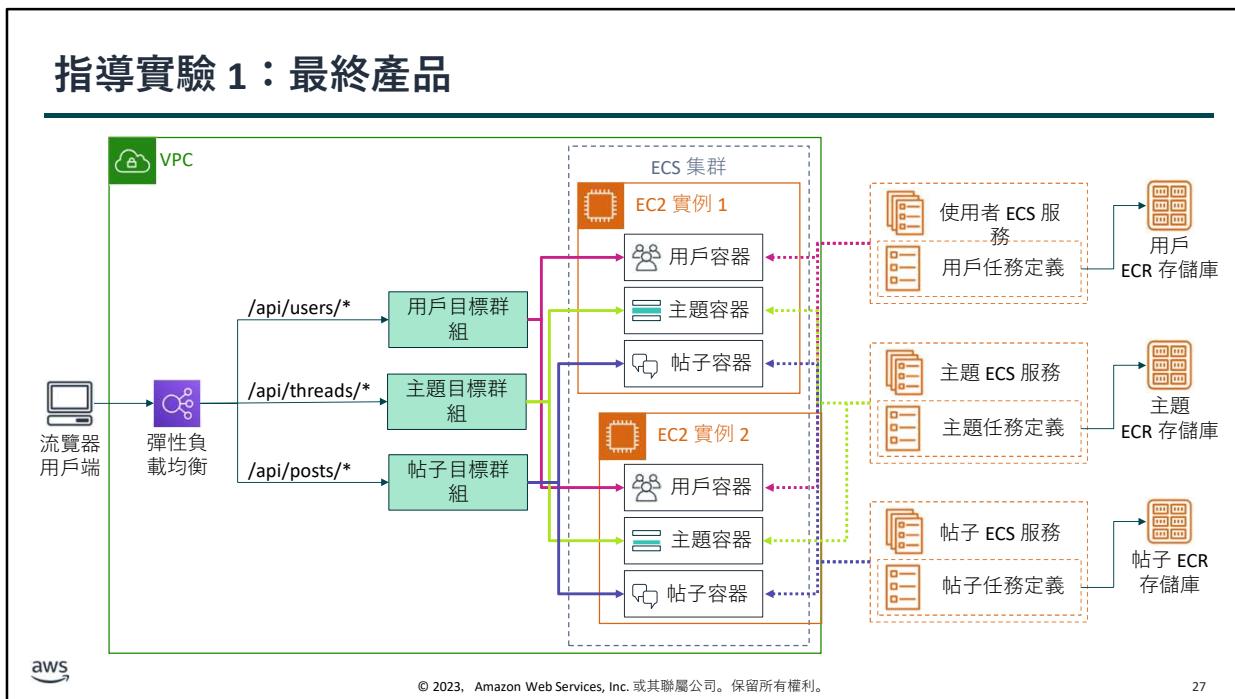
© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

26

在本指導實驗中，您將完成以下任務：

1. 準備 AWS Cloud9 開發環境
2. 在基本 Node.js 啟動器上運行整體式應用程式
3. 為 Amazon ECS 容器化整體式應用程式
4. 將整體式應用程式部署至 Amazon ECS
5. 將整體式應用程式重構為容器化微服務

指導實驗 1：最終產品



該圖總結了您完成實驗後將構建的內容。



大約 3 小時



開始模組 13 – 指導實驗 1： 將整體式 Node.js 應用程 式拆分為微服務



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

28

現在可以開始可選的指導實驗了。

指導實驗 1 總結： 要點



完成這個指導實驗之後，您的講師可能會帶您討論此指導實驗的要點。

第 4 節：介紹無伺服器架構

模組 13：構建微服務和無伺服器架構

 AWS

© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

介紹第 4 節：介紹無伺服器架構。

無伺服器意味著什麼？

使您可以在**不考慮伺服器的情況下構建並運行應用程式和服務的方法**



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

31

到目前為止，您已經瞭解到可以使用容器通過 Amazon ECS 構建微服務應用程式。Amazon ECS 是一項容器編排服務，您可以在其中管理應用程式碼、資料來源集成、安全配置、更新、網路配置、防火牆和管理任務。您還瞭解到可以使用 Fargate 啟動類型在 Amazon ECS 管理的**無服務器**基礎設施上託管集群。

但是，無伺服器意味著什麼？

無伺服器是雲原生架構，使您能夠將更多的運營職責轉移到 AWS，從而提高敏捷性和創新能力。無伺服器使您可以在**不考慮伺服器的情況下構建並運行應用程式和服務**。應用程式仍在伺服器上運行。但是，AWS 會執行所有伺服器管理任務，例如伺服器或集群預置、修補、作業系統維護和容量預置。

無伺服器架構的原則

無需基礎設施預置,
無需管理工作

彈性伸縮

按價值付費

高度可用且安全



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

32

將無伺服器定義為運營模型的原則包括：

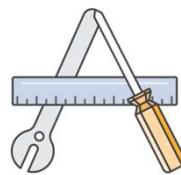
- 無需預置或管理基礎設施（無需預置、運行或修補伺服器）
- 按消費單位自動擴展（按工作單位或消費單位進行擴展，而不是按伺服器單位進行擴展）
- 按價值付費定價模型（您只需為資源運行的持續時間付費，而不是按伺服器單位付費）
- 內置的可用性和容錯能力（因為可用性內置在服務中，因此無需設計可用性架構）

有關無伺服器介紹的更多資訊，請參閱[此 AWS 網站](#)。

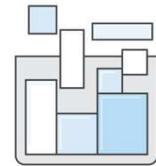
無伺服器的益處



更低的總體擁有成本



專注于應用程式
而不是配置



構建微服務應用程式



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

33

無伺服器計算讓您能夠以更高的敏捷性和更低的總體擁有成本 (TCO) 構建現代化應用程式。通過使用無伺服器架構，您可以專注於核心產品。不管是在雲中還是在本地部署，您都無需擔心伺服器或運行時的管理和運行。這樣可以減少開銷，讓您能夠將時間和精力投入到可擴展並且可靠的產品的開發上。最後，無伺服器架構使您能夠構建微服務應用程式。

AWS 無伺服器產品和服務



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

34

AWS 提供了許多產品，您可以使用它們在 AWS 上構建無伺服器架構。到目前為止，在本課程中，您已經學習了其中幾個產品。

本模組的其餘部分重點介紹如何使用 AWS Lambda、Amazon API Gateway 和 AWS Step Functions 構建無伺服器架構。

第 4 節要點



- 無伺服器計算使您可以構建並運行應用程式和服務，而無需預置或管理伺服器
- 無服務器架構提供以下益處 -
 - 更低的總體擁有成本 (TCO)
 - 您可以專注于應用程式
 - 您可以使用它們構建微服務應用程式

本模組中這節內容的要點包括：

- 無伺服器計算使您可以構建並運行應用程式和服務，而無需預置或管理伺服器
- 無服務器架構提供以下益處 -
 - TCO 更低
 - 您可以專注于應用程式
 - 您可以使用它們構建微服務應用程式

第 5 節：使用 AWS Lambda 構建無伺服器架構

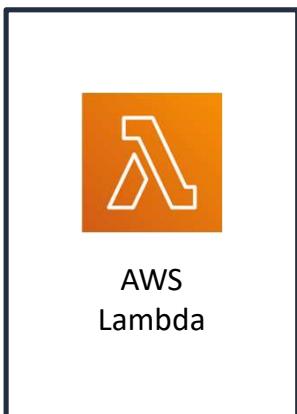
模組 13：構建微服務和無伺服器架構

aws

© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

介紹第 5 節：使用 AWS Lambda 構建無伺服器架構。

AWS Lambda



- 是完全託管式計算服務
- 按計畫或根據事件（例如 Amazon S3 存儲桶或 Amazon DynamoDB 表的更改）運行代碼
- 支持 Java, Go, PowerShell, Node.js, C#, Python, Ruby 和 Runtime API
- 可以在靠近用戶的邊緣網站運行



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

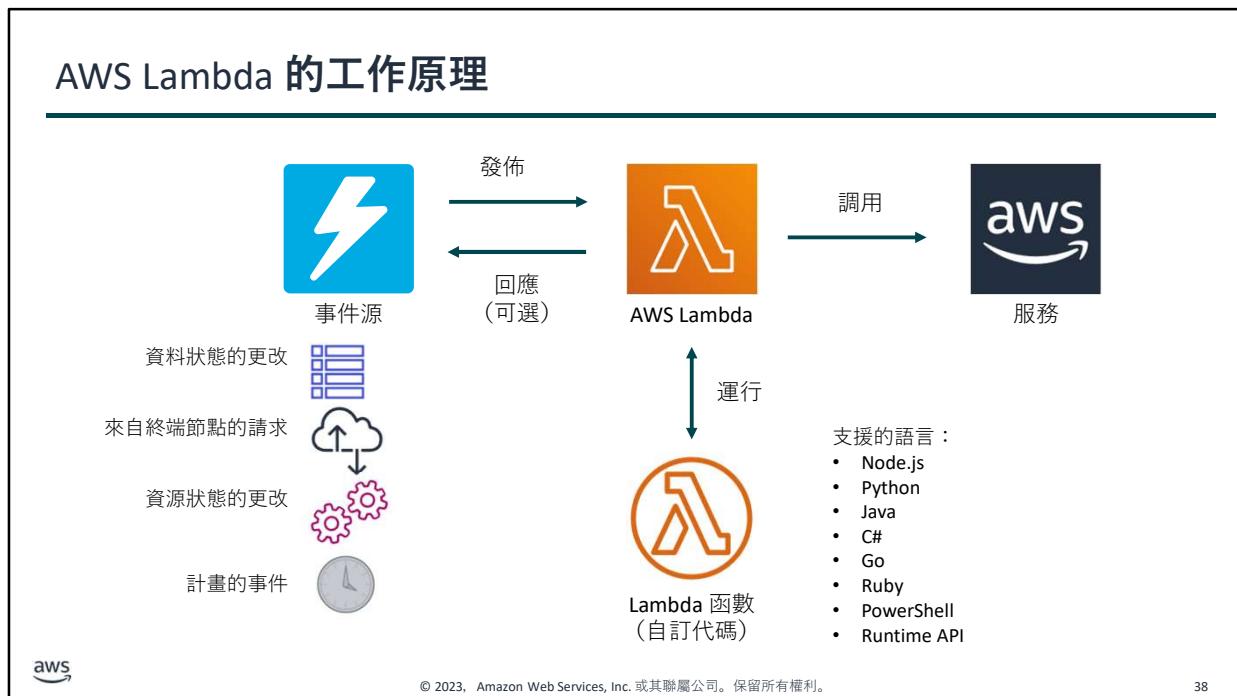
37

[AWS Lambda](#) 是一項完全託管式的計算服務，可運行代碼來回應事件並為您自動管理底層計算資源。Lambda 在高可用性計算基礎設施上運行您的代碼，執行計算資源的所有管理工作，包括伺服器和作業系統維護、容量預置、彈性伸縮、代碼監控和日誌記錄。

AWS Lambda 原生支持 Java、Go、PowerShell、Node.js、C#、Python 和 Ruby 代碼，還提供 Runtime API，讓您能夠使用任何其他程式設計語言來編寫函數。

[Lambda@Edge](#) 是 Amazon CloudFront 的一項功能，它可讓您在靠近應用程式使用者的地方運行代碼，從而提高性能、降低延遲。Lambda@Edge 根據 Amazon CloudFront 內容分發網路 (CDN) 生成的事件運行代碼。利用 Lambda@Edge，您可以運行 Node.js 和 Python Lambda 函數，以自訂 Amazon CloudFront 分發的內容。有關如何添加 HTTP 安全回應標頭的資訊，請閱讀這篇 [AWS 聯網和內容分發博客文章](#)。

AWS Lambda 的工作原理

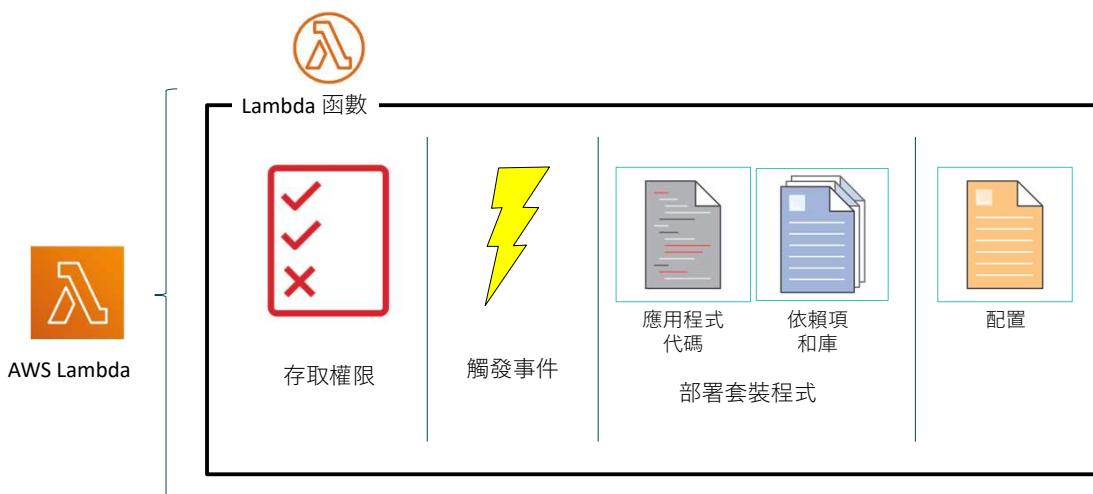


AWS Lambda 與其他 AWS 服務集成以調用 Lambda 函數。*Lambda 函數*是您使用 Lambda 支援的其中一種語言編寫的自訂代碼。您可以配置觸發器來調用函數以回應資源生命週期事件、回應傳入的 HTTP 請求、使用佇列中的事件或按計劃運行。

事件源是將事件發布到 Lambda 的實體。您的 Lambda 函數會處理事件，Lambda 代表您運行您的 Lambda 函數。

Lambda 函數是無狀態的，這意味著它們與底層基礎設施沒有密切關係。Lambda 可以根據需要快速啟動足夠多的函數副本，以根據傳入事件的速率進行擴展。

Lambda 函數



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

39

當您創建 Lambda 函數時，您將定義該函數的許可權並指定觸發該函數的事件。您還將創建一個部署套裝程式，其中包括您的應用程式碼以及運行您的代碼所需的所有依賴項和庫。最後，您將配置記憶體、超時和併發性等運行時參數。調用函數時，Lambda 將基於您選擇的運行時和配置選項運行環境。

有關 AWS Lambda 運行方式的更多資訊，請參閱 [AWS 文檔](#)。

Lambda 函數剖析

Handler()

調用時要運行的函數

事件物件

Lambda 函式呼叫期間發送的資料

上下文對象

可用于與運行時資訊交互的方法（請求 ID, 日誌組等）

```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello world')
    }
```



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

40

調用 Lambda 函數時，代碼開始在處理常式處運行。處理常式是您創建並包含在套裝程式中的特定代碼方法或函數。創建 Lambda 函數時指定處理常式。對於如何在套裝程式中定義和引用函數處理常式，所支援的每種語言都有各自不同的要求。在 Lambda 函數內成功調用處理常式後，運行時環境屬於您編寫的代碼。

處理常式始終採用兩個物件：事件物件和上下文物件。

事件物件提供有關觸發 Lambda 函數的事件的資訊。此類事件可以是 AWS 服務生成的預定義物件，也可以是可序列化字串形式的使用者自訂物件。此類字串的示例可能是普通的舊 Java 對象 (POJO) 或 JSON 流。

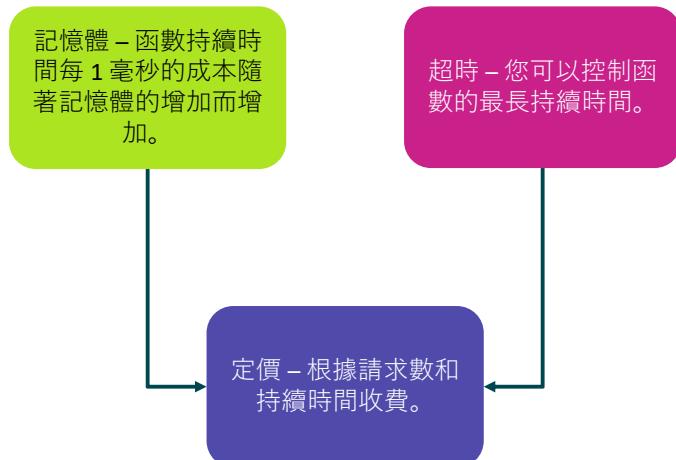
事件物件的內容包括 Lambda 函數驅動其邏輯所需的所有資料和中繼資料。事件物件的內容和結構各不相同，具體取決於由哪個事件源創建。例如，API Gateway 創建的事件包含與 API 用戶端發出的 HTTPS 請求相關的詳細資訊 – 例如路徑、查詢字串和請求體。而 Amazon 創建的事件包括有關存儲桶和新物件的詳細資訊。

上下文物件由 AWS 生成，提供有關運行時環境的中繼資料。上下文物件使您的函數代碼能與 Lambda 運行時環境進行交互。上下文物件的內容和結構因 Lambda 函數使用的語言運行時而異。

不過，上下文物件至少包含：

- awsRequestId – 此屬性用於跟蹤 Lambda 函數的特定調用（對於錯誤報告或聯繫 AWS Support 時很重要）
- logStreamName – 您的日誌語句將被發送到的 CloudWatch 日誌流
- getRemainingTimeInMillis() – 該方法返回函數運行超時前剩餘的毫秒數

Lambda 函數配置和計費



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

41

記憶體和超時是確定 Lambda 函數執行方式的配置。這些配置會影響您的帳單。如果您使用 AWS Lambda，我們將根據您函數的請求數量（所有函數的請求總數）和持續時間（代碼運行花費的時間）向您收費。價格取決於您為函數分配的記憶體量。

記憶體 – 您可以指定要分配給 Lambda 函數的記憶體量。然後，Lambda 分配與記憶體成比例的 CPU 處理能力。Lambda 的定價方式是，每 1 毫秒函數持續時間的成本隨著記憶體配置的增加而增加。例如，假設您有一個記憶體為 256 MB 的 Lambda 函數，執行時間為 110 毫秒。與執行時間相同、記憶體容量為 128 MB 的 Lambda 函數相比，該函數的成本將增加一倍。

超時 – 您可以使用超時配置來控制函數的最長持續時間。您可以將函數的超時值設置為不超過 15 分鐘的任意值。當達到指定超時後，AWS Lambda 會停止 Lambda 函數的運行。使用超時可以避免因長時間運行函數而產生更高的成本。您必須在不讓函數運行太久和正常情況下能夠完成函數之間找到適當的平衡。

請遵循以下最佳實踐：

- 測試 Lambda 函數的性能，確保選擇了最佳記憶體大小配置。您可以在 Amazon CloudWatch Logs 中查看函數的記憶體使用情況。
- 對 Lambda 函數進行負載測試，分析函數的執行時間，確定最佳超時值。當您的 Lambda 函數對可能無法處理 Lambda 函數擴展的資源進行網路調用時，這一點非常重要。

有關更多資訊，請參閱以下資源：

- [AWS Lambda 限制](#)
- [AWS Lambda 定價](#)

演示：創建 AWS Lambda 函數

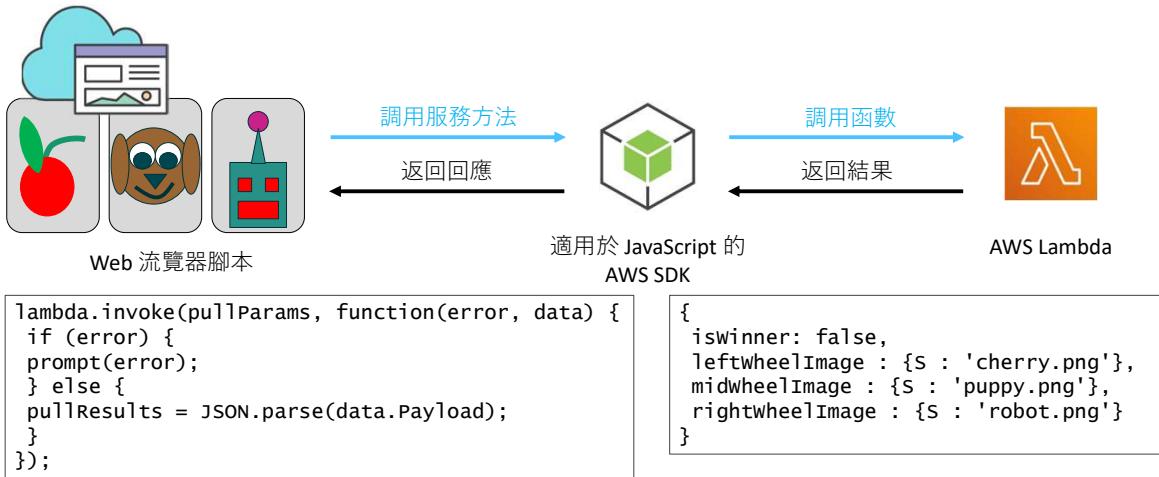


© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

42

現在，講師可能會選擇演示如何創建 AWS Lambda 函數。

AWS Lambda 實例： 模擬老虎機流覽器遊戲

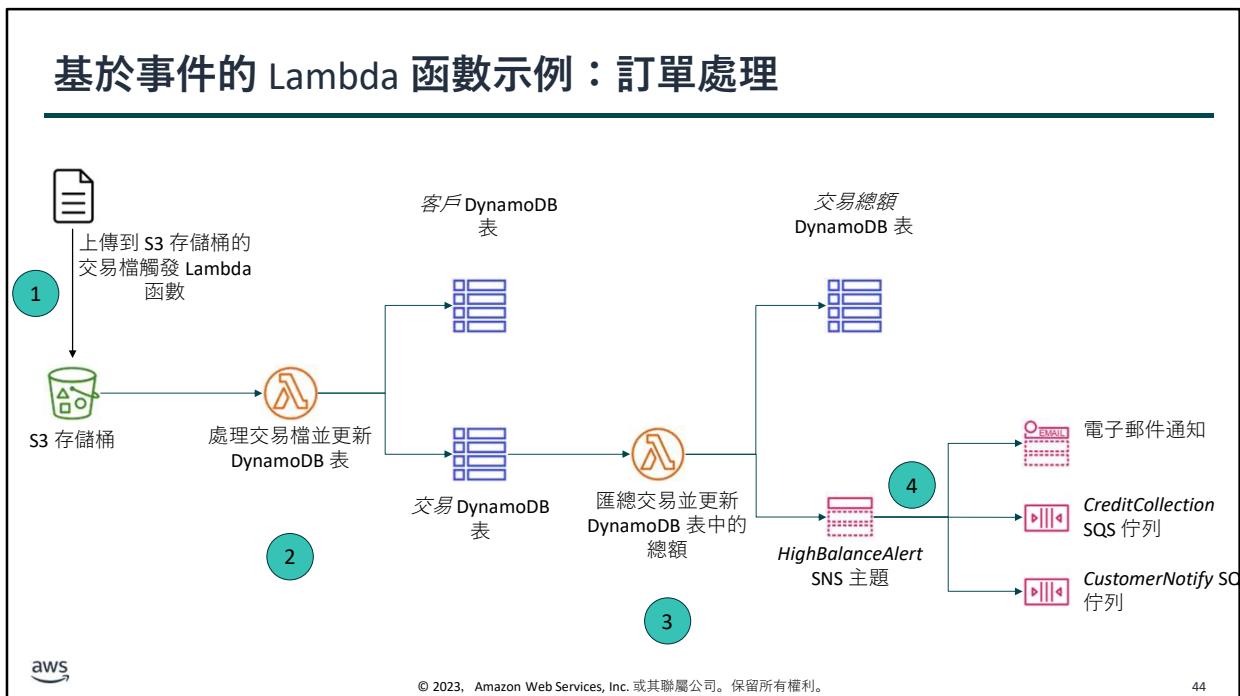


© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

43

您可以創建 Lambda 函數來執行各種任務。此示例使用的是一款模擬老虎機的基於流覽器的遊戲。該遊戲會調用一個 Lambda 函數，生成每次拉動老虎機的隨機結果。函數返回這些結果，並使用相應的影像檔名來顯示結果。圖像存儲在 Amazon S3 存儲桶中，而存儲桶配置為靜態 Web 主機，用於託管提供應用程式體驗所需的 HTML、CSS 和其他資產。

基於事件的 Lambda 函數示例：訂單處理



此示例展示了如何在訂單處理解決方案中使用 Lambda。

在本架構中：

1. 客戶將交易檔上傳到 S3 存儲桶，從而觸發 Lambda 函數的運行。
2. Lambda 函數會處理交易檔並更新客戶和交易 DynamoDB 表。
3. 對交易 DynamoDB 表的更改將觸發第二個 Lambda 函數來聚合事務，並更新交易總額 DynamoDB 表中的總數。它還向 *HighBalanceAlert SNS* 主題推送一條消息。
4. *HighBalanceAlert SNS* 主題向客戶發送電子郵件通知，並更新 *CreditCollection* 和 *CustomerNotify* SQS 佇列，以便進行付款處理。

Lambda 層



- 使函數能夠輕鬆共用代碼 – 您可以一次上傳一層並在任何函數中引用它
- 促進責任分離 – 開發人員可以更快地反覆運算編寫業務邏輯
- 使您能夠保持較小的部署套裝程式
- 限制 –
 - 最多五層
 - 250 MB

AWS

© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

45

在構建無伺服器應用程式時，通常會在 Lambda 函數之間共用代碼。可以是兩個或更多函數使用的自訂代碼，也可以是為簡化業務邏輯實施而添加的標準庫。

以前，您會將這些共用代碼與使用這些代碼的所有函數一起打包和部署。現在，您可以將 Lambda 函數配置為以層的形式包括其他代碼和內容。層是一個包含庫、自訂運行時或其他依賴項的 .zip 歸檔檔。

使用 Lambda 層，函數可以共用代碼。開發人員使用層一次性上傳代碼，然後多次重複使用。利用層，您可以在函數中使用庫，而不必將庫包含在部署套裝程式中。

這樣共用代碼有助於促進責任分工。一個人可以負責管理核心庫。另一個人可以負責使用庫代碼並在庫代碼的基礎上構建應用程式邏輯。

通過使用層，您可以將部署套裝程式保持較小，從而使開發變得更輕鬆。

一個函數一次最多可以使用五個層。函數和所有層的解壓後總大小不能超出 250 MB 的解壓後部署套裝程式大小限制。

有關層的更多資訊，請參閱 [AWS Lambda 層](#)。

演示：結合使用 AWS Lambda 和 Amazon S3



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

46

現在，講師可能會選擇演示如何配置 Amazon S3 事件以觸發 Lambda 函數。

Demonstration diagram

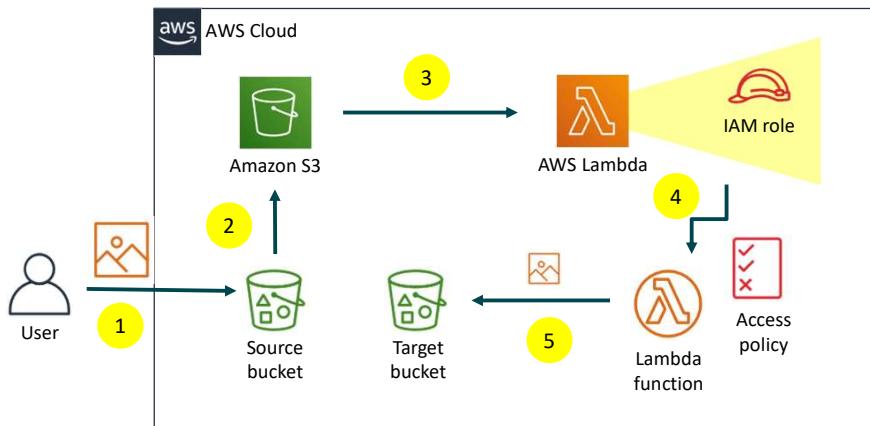
1. User uploads image to source S3 bucket.

2. Amazon S3 detects the object-created event.

3. Amazon S3 publishes the event to Lambda.

4. Lambda runs the Lambda function.

5. The Lambda function resizes the original image and saves the thumbnail to the target S3 bucket.



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

47

This demonstration covers the following steps:

1. A user uploads an image to a source S3 bucket.
2. Amazon S3 detects the object-created event.
3. Amazon S3 publishes the event to Lambda by invoking the Lambda function and passing event data as a function parameter.
4. Lambda assumes an IAM role that allows it to run the function, and then runs the Lambda function.
5. From the event data it receives, the Lambda function knows the source bucket and the object key name. The Lambda function reads the object, creates a thumbnail of the original image, and saves the thumbnail to the target S3 bucket.

```

import boto3
import os
import sys
import uuid
from PIL import Image
import PIL.Image

s3_client = boto3.client('s3')

def resize_image(image_path, resized_path):
    with Image.open(image_path) as image:
        image.thumbnail((128, 128))
        image.save(resized_path)

def handler(event, context):
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = record['s3']['object']['key']
        download_path = '/tmp/{}{}'.format(uuid.uuid4(), key)
        upload_path = '/tmp/resized-{}'.format(key)

        s3_client.download_file(bucket, key, download_path)
        resize_image(download_path, upload_path)
        s3_client.upload_file(upload_path, '{}-resized'.format(bucket), key)

```

The diagram illustrates the execution flow of the Lambda function. It starts with the function receiving an S3 event (labeled "Receives S3 event and downloads the image to local storage"). This triggers the download of the image from the source bucket to a temporary local path. The image is then resized using the `resize_image` function. Finally, the resized image is uploaded back to a new bucket named with a suffix "-resized".



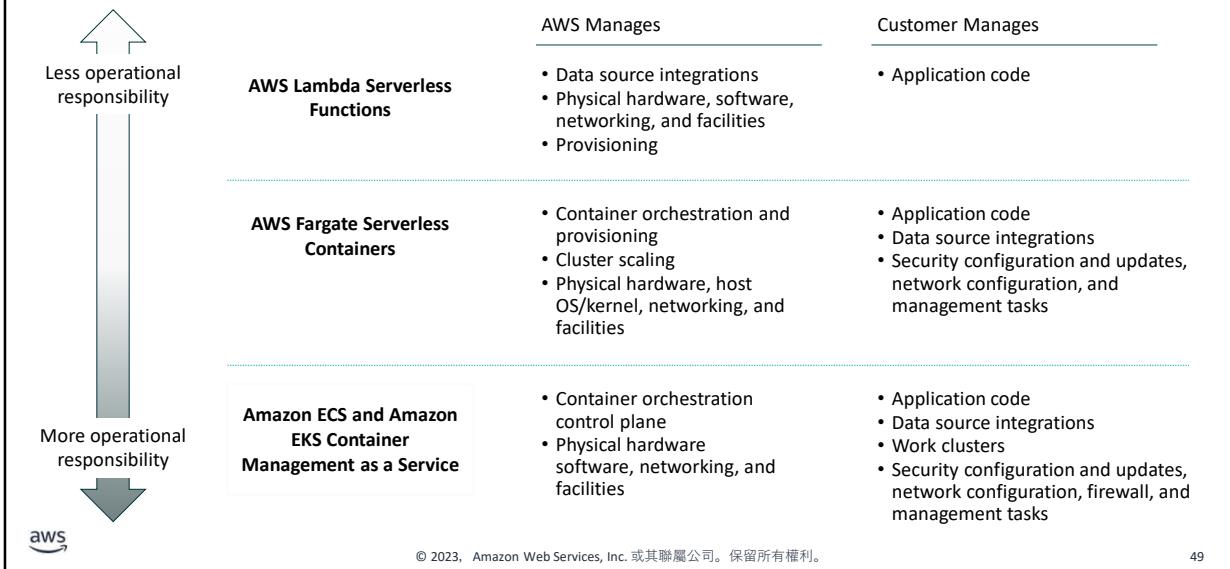
© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

48

The Lambda function code performs the following steps:

- Receives an S3 event, which contains the name of the incoming object (Bucket, Key)
- Downloads the image to local storage
- Resizes the image using the *Pillow* library
- Uploads the resized image to the *-resized* S3 bucket

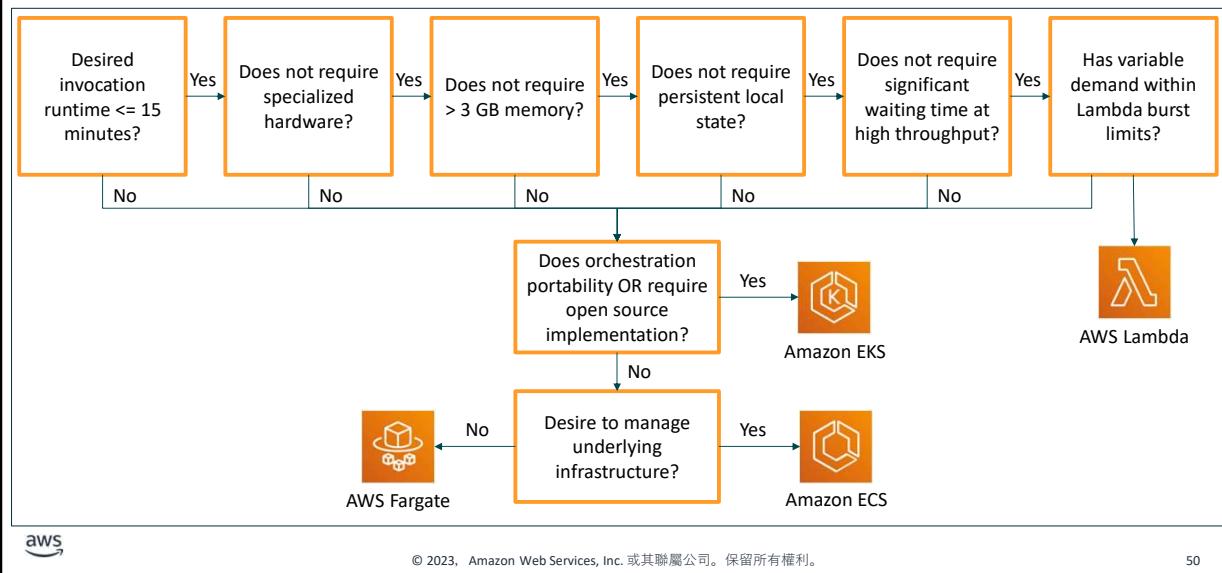
Comparison of operational responsibility for container and serverless architectures



49

Which compute service you use to build your application depends on the level of control that you want. You share a spectrum of operational responsibility with AWS over your compute options for container and serverless architectures.

Choosing a compute platform: Containers versus AWS Lambda



This slide displays a rough guideline for selecting your compute platform. The recommendation is based on [AWS Lambda limits](#) (memory limitation, time limitation).

第 5 節要點



aws

- Lambda 是一種無伺服器計算服務，提供內置的容錯能力和彈性伸縮功能
- Lambda 函數是您編寫的用於處理事件的自定義代碼
- Lambda 函數由處理程式調用，處理常式使用事件對象和上下文對象作為參數
- 事件源是觸發 Lambda 函數運行的 AWS 服務或開發人員創建的應用程式
- Lambda 層使函數能夠共用代碼並能夠保持部署套裝程式較小

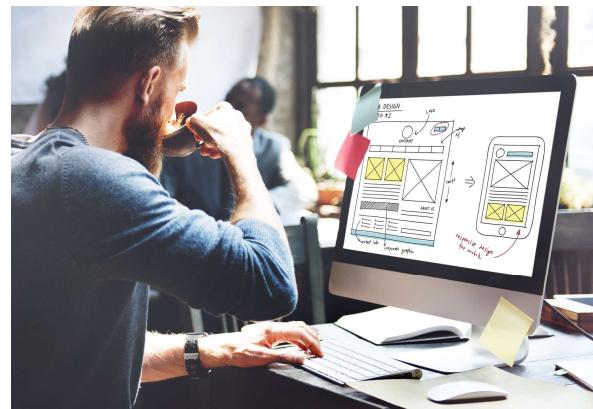
© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

51

本模組中這節內容的要點包括：

- Lambda 是一種無伺服器計算服務，提供內置的容錯能力和彈性伸縮功能。
- Lambda 函數是您編寫的用於處理事件的自訂代碼。
- Lambda 函數由處理常式調用，處理常式使用事件物件和上下文物件作為參數。
- 事件源是觸發 Lambda 函數運行的 AWS 服務或開發人員創建的應用程式。
- Lambda 層使函數能夠共用代碼並能夠保持部署套裝程式較小。

模組 13 – 指導實驗 2：在 AWS 上實施 無伺服器架構



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

52

現在您將完成模組 13 – 指導實驗 2：在 AWS 上實施無伺服器架構。

指導實驗 2：任務

1. 創建 Lambda 函數以載入資料
2. 配置 Amazon S3 事件
3. 測試載入過程
4. 配置通知
5. 創建 Lambda 函數以發送通知
6. 測試系統



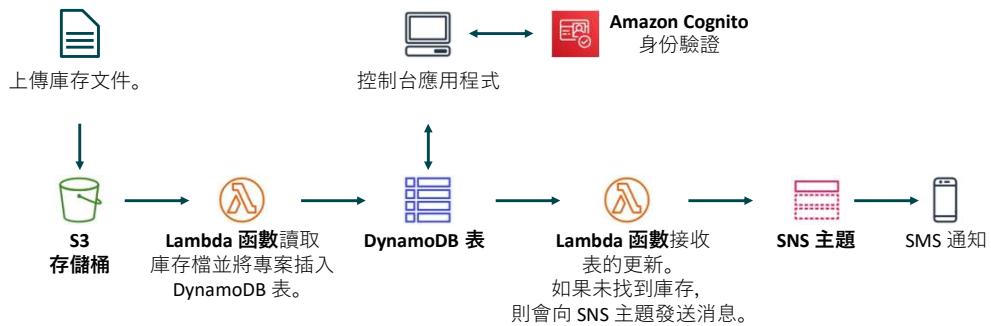
© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

53

在本指導實驗中，您將完成以下任務：

1. 創建 Lambda 函數以載入資料
2. 配置 Amazon S3 事件
3. 測試載入過程
4. 配置通知
5. 創建 Lambda 函數以發送通知
6. 測試系統

指導實驗 2：最終產品



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

54

該圖總結了您完成實驗後將構建的內容。



大約 40 分鐘



開始模組 13 – 指導實驗 2： 在 AWS 上實施無服務 器架構



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

55

現在可以開始指導實驗了。

指導實驗 2 總結： 要點



完成這個指導實驗之後，您的講師可能會帶您討論此指導實驗的要點。

第 6 節：使用 Amazon API Gateway 擴展無伺服器架構

模組 13：構建微服務和無伺服器架構

 AWS

© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

介紹第 6 節：使用 Amazon API Gateway 擴展無伺服器架構。

Amazon API Gateway



- 使您能夠創建、發佈、維護、監控和保護作為應用程式後端資源入口點的 API
- 處理高達數十萬個併發 API 調用
- 可以處理在以下工具上運行的工作負載 –
 - Amazon EC2
 - Lambda
 - 任何 Web 應用程式
 - 實時通信應用程式
- 可以託管和使用多個版本和多個階段的 API



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

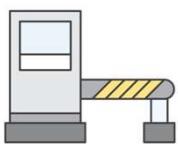
58

Amazon API Gateway 是一項完全託管式服務，使您可以創建、發佈、維護、監控和保護任意規模的 API。您可以使用此服務創建表述性狀態轉移 (RESTful) 和 WebSocket API，這些 API 充當應用程式的入口點，以便它們可以訪問後端資源。然後應用程式可從後端服務訪問資料、業務邏輯或功能。此類服務包括在 Amazon EC2 上運行的應用程式、在 Lambda 上運行的代碼、任何 Web 應用程式或即時通信應用程式。

API Gateway 可處理接受和處理多達數十萬次併發 API 調用所涉及的所有任務。這些調用可能包括流量管理、授權和存取控制、監控以及 API 版本管理。API Gateway 沒有最低費用，也沒有啟動成本。您只需為收到的 API 調用以及傳出的資料量付費。利用 API Gateway 分級定價模式，您可以隨著 API 使用量的增加而降低成本。

您可以使用 API Gateway 託管多個版本和階段的 API。

Amazon API Gateway 安全



需要授權



應用資源策略



限流設置



防止分散式拒絕服務
(DDoS) 和注入攻擊



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

59

當您將 API 設為公開可用時，您就暴露在攻擊者面前，他們會試圖利用您的服務。借助 Amazon API Gateway，您可以通過多種方式保護 API。

借助 Amazon API Gateway，您可以選擇性地將 API 方法設置為需要授權。在設置需要授權的方法時，可以使用 AWS 簽名版本 4 或 Lambda 授權方來支持自己的不記名權杖身份驗證策略。AWS 簽名版本 4 是為通過 HTTP 發送的 AWS 請求添加身份驗證資訊的過程。出於安全考慮，大多數發送到 AWS 的請求都必須使用訪問金鑰（包括訪問金鑰 ID 和秘密訪問金鑰）進行簽名。與其他 AWS 服務一樣，您可以使用這些 AWS 憑證來簽署對您服務的請求並授權訪問。您可以使用 Amazon Cognito 檢索與 AWS 帳戶中的角色相關聯的臨時憑證。Lambda 授權方是一種 Lambda 函數，它通過使用像 OAuth 這樣的持有者權杖身份驗證策略來授權對 API 的訪問。

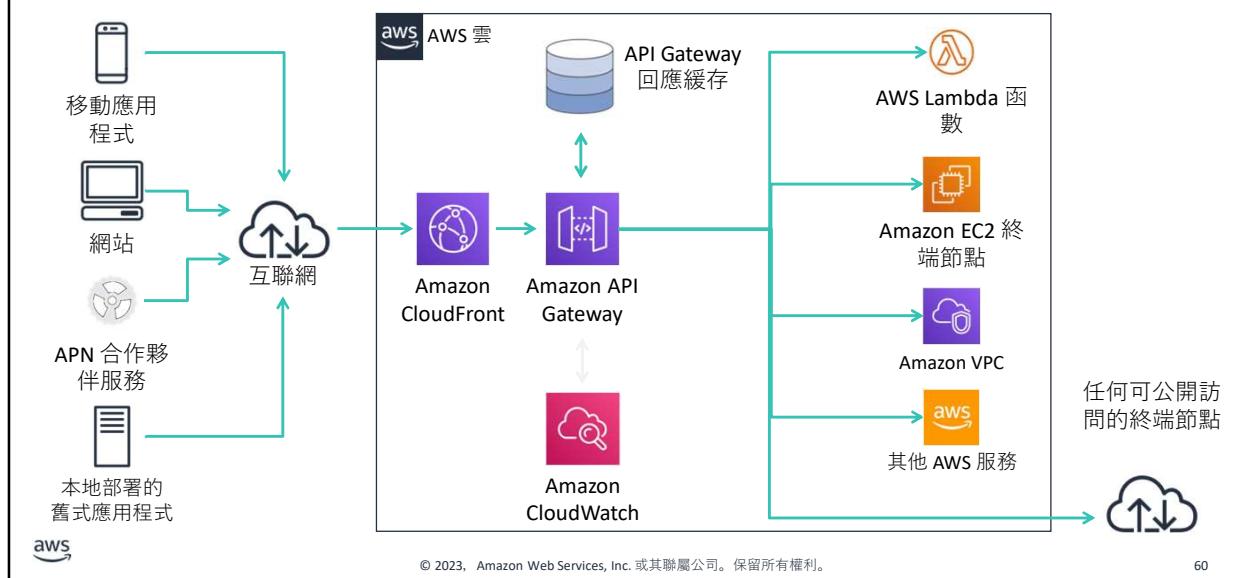
您還可以將資源策略應用到 API，以限制對特定 Amazon VPC 或 VPC 終端節點的訪問。您可以通過使用資源策略，讓不同帳戶的 Amazon VPC 或 VPC 終端節點訪問私有 API。

Amazon API Gateway 支持對 API 中的每個方法或路由進行限流設置。您可以為 REST API 中的每個方法和 WebSocket API 中的每個路由設置標準速率限制和每秒突發速率限制。

此外，您可以使用 AWS WAF 來保護 API Gateway API 的安全。[AWS WAF](#) 是一款 Web 應用程式防火牆，有助於保護您的 Web 應用程式免受常見 Web 攻擊，這些攻擊可能會影響可用性、危及安全性或消耗過多資源。

有關如何使用 Amazon API Gateway 保護 API 的更多資訊，請參閱 Amazon API Gateway 常見問題中的[安全和授權部分](#)。

Amazon API Gateway：通用架構示例



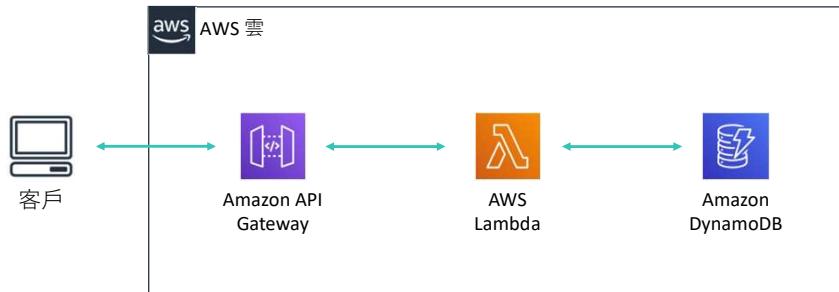
您可以使用 Amazon API Gateway 為應用程式提供 API 層。以下是使用 Amazon API Gateway 的通用架構示例。

在此示例中，前端用戶端應用程式和應用程式服務通過互聯網向 API Gateway 發送流量。通常，Amazon CloudFront 用於緩存靜態內容。API Gateway 抽象並公開可以調用各種後端應用程式的 API。這些應用程式包括 Lambda 函數、在 EC2 實例上運行的 Docker 容器、Virtual Private Cloud (VPC) 或任何可公開訪問的終端節點。如有必要，API Gateway 可以緩存回應。最後，所有 API 調用都可以使用 Amazon CloudWatch 進行監控。

您可以將 API Gateway 與其他 AWS 託管服務結合使用，為應用程式構建無伺服器後端。例如，API Gateway 可以將請求代理到運行代碼並生成回應的 Lambda 函數。您甚至可以創建將請求代理到其他 AWS 服務（例如 Amazon Simple Storage Service (Amazon S3)）的 API，而不必編寫任何代碼。您可以更快地運行生產規模的後端，因為您只需編寫更少的代碼，而且還可以將操作負擔轉移到 AWS 服務。

有關如何將 API Gateway 與 AWS Lambda 結合使用的示例，請參閱此 [AWS 博客文章](#)。

示例：RESTful 微服務



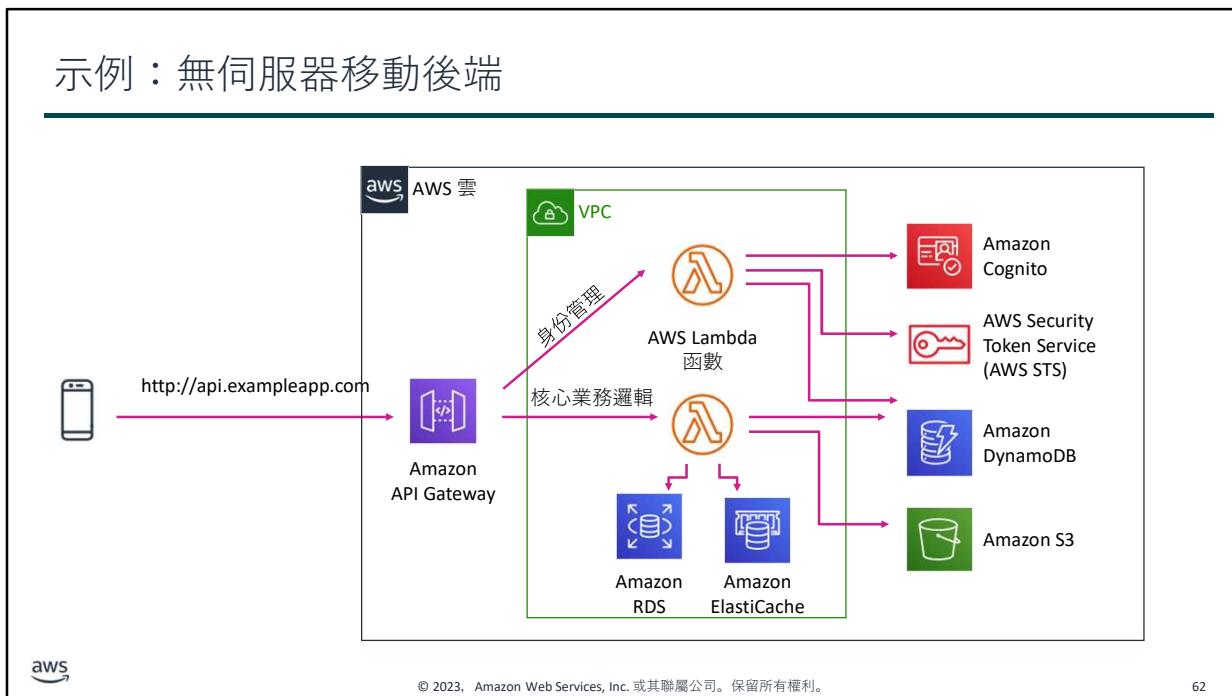
© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

61

Amazon API Gateway 與 AWS Lambda 深度集成。此示例展示了如何在 RESTful 微服務應用程式的架構中將這兩種服務結合使用。

在此示例中，客戶可以通過進行 HTTP API 調用來使用微服務。Amazon API Gateway 託管來自客戶的 RESTful HTTP 請求和對客戶的回應。在這種情況下，API Gateway 提供內置的授權、限流、安全性、容錯能力、請求/回應映射和性能優化。AWS Lambda 包含處理傳入 API 調用的業務邏輯，並使用 DynamoDB 作為持久性存儲。Amazon DynamoDB 可持久存儲微服務資料，並根據需求進行擴展。由於微服務的設計目的是做好一件事，因此會定期採用無架構 NoSQL 資料存儲。

示例：無伺服器移動後端



再看一個例子，說明如何在無伺服器架構中將 Amazon API Gateway 與 Lambda 結合使用，作為移動應用程式的後端閘道。人們期望移動應用程式能夠提供快速、一致、功能豐富的用戶體驗，而且這些應用程式通常會覆蓋全球。此外，移動使用者模式是動態的，其使用高峰無法預測。移動應用程式需要一套豐富的移動服務，這些服務可以無縫協作，同時又不影響後端基礎設施的控制和靈活度。

在此示例中，移動應用程式向 Amazon API Gateway 發送請求，後者將請求轉發給一個 Lambda 函數，該函數呼叫 Amazon Cognito 和 AWS Security Token Service (AWS STS) 來管理身份。Amazon Cognito 通過支援安全斷言標記語言 (SAML) 或 OpenID Connect 的外部身份提供商 (IdP)、社交 IdP（如 Facebook、Twitter、Amazon）和自訂 IdP 對移動應用程式的使用者進行身份驗證。確認移動使用者身份後，另一個 Lambda 函數將運行應用程式的核心業務邏輯。

第 6 節要點



- Amazon API Gateway 是一項完全託管式服務，使您可以創建、發佈、維護、監控和保護任意規模的 API。
- Amazon API Gateway 可充當應用程式後端資源的入口點。它抽象並公開可以調用各種後端應用程式的 API。這些應用程式包括 Lambda 函數、在 EC2 實例上運行的 Docker 容器、VPC 或任何可公開訪問的終端節點。
- Amazon API Gateway 與 Lambda 深度集成。

本模組中這節內容的要點包括：

- Amazon API Gateway 是一項完全託管式服務，使您可以創建、發佈、維護、監控和保護任意規模的 API。
- Amazon API Gateway 可充當應用程式後端資源的入口點。它抽象並公開可以調用各種後端應用程式的 API。這些應用程式包括 Lambda 函數、在 EC2 實例上運行的 Docker 容器、VPC 或任何可公開訪問的終端節點。
- Amazon API Gateway 與 Lambda 深度集成。

第 7 節：使用 AWS Step Functions 編排微服務

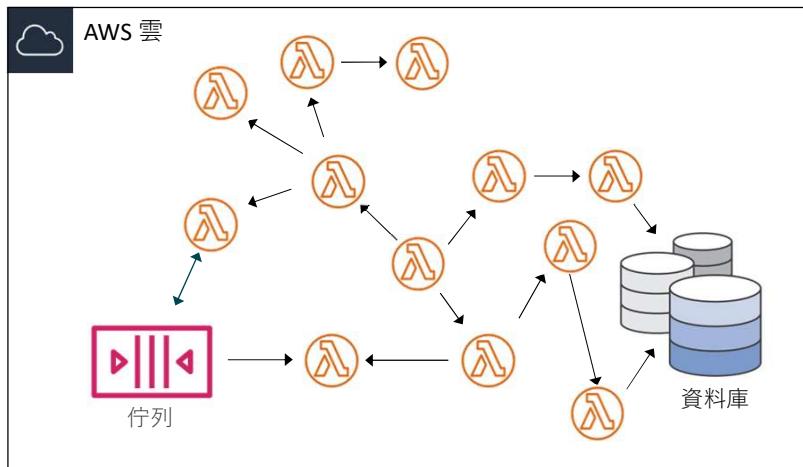
模組 13：構建微服務和無伺服器架構



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

介紹第 7 節：使用 AWS Step Functions 編排微服務。

微服務應用程式的挑戰



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

65

隨著應用程式規模的擴大，元件的數量也會增加，因此可能會出現多種運行任務的模式和順序。例如，假設有一個使用 Lambda 函數構建的微服務應用程式。您可能希望當且僅當另一個函數成功運行後，立即調用 Lambda 函數。您可能希望並行調用兩個函數，然後將合併後的結果提供給第三個函數。或者，您可能希望根據另一個函數的輸出來選擇調用兩個函數中的哪一個。

出於多種原因，函式呼叫可能會導致錯誤。您的代碼可能引發異常、超時或用盡記憶體。運行您的代碼的運行時可能會遇到錯誤並因此停止。當發生錯誤時，您的代碼可能已經完全運行、部分運行，或者完全未運行。在大多數情況下，調用您函數的用戶端或服務會在遇到錯誤時重試，因此您的代碼必須能夠重複處理同一事件，而不會產生不必要的影響。如果您的函數管理資源或向資料庫寫入內容，您必須處理多次提出相同請求的情況。

您需要一種方法來協調應用程式的各個元件。該協調層必須能夠根據不斷變化的工作負載自動擴展，並處理錯誤和超時。它還必須在應用程式運行時維護其狀態，例如跟蹤當前正在運行的步驟，並存儲在工作流各步驟之間移動的資料。這些功能有助於您構建和運行應用程式。您還希望應用程式具有可視性，以便可以排查錯誤和跟蹤性能。

AWS Step Functions

- 通過使用視覺化工作流協調微服務
- 讓您可以逐步執行應用程式的功能
- 自動觸發和跟蹤每個步驟
- 在步驟失敗時提供簡單的錯誤捕獲和日誌記錄

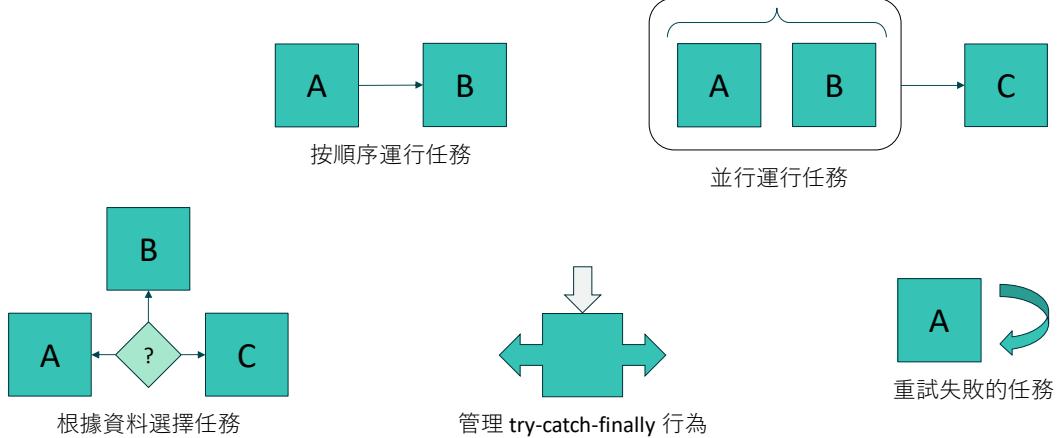


© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

66

[AWS Step Functions](#) 是一項 Web 服務，讓您能夠使用視覺化工作流來協調分散式應用程式和微服務的元件。Step Functions 是協調元件和逐步執行應用程式函數的可靠方法。Step Functions 提供圖形控制台，以按照一系列步驟實現應用程式元件的視覺化。它能自動觸發和跟蹤每個步驟，還能在出現錯誤時重試，從而使應用程式按預期順序運行。Step Functions 記錄每個步驟的狀態，因此您可以快速診斷並調試問題。

工作流協調



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

67

AWS Step Functions 為您管理應用程式的邏輯。它實施了基本的基元，如按順序或並行運行任務、分支和超時。這種技術可以刪除微服務和函數中可能重複的額外代碼。AWS Step Functions 利用內置的 try-catch 和重試功能自動處理錯誤和異常，無論任務需要幾秒鐘還是幾個月才能完成均如此。您可以自動重試失敗或超時的任務。您可以對不同類型的錯誤做出不同的回應，並通過回退到指定的清理和恢復代碼來從容恢復。

狀態機



狀態機是可以執行工作的狀態集合。

售賣機

等待交易

選擇飲料

售賣飲料



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

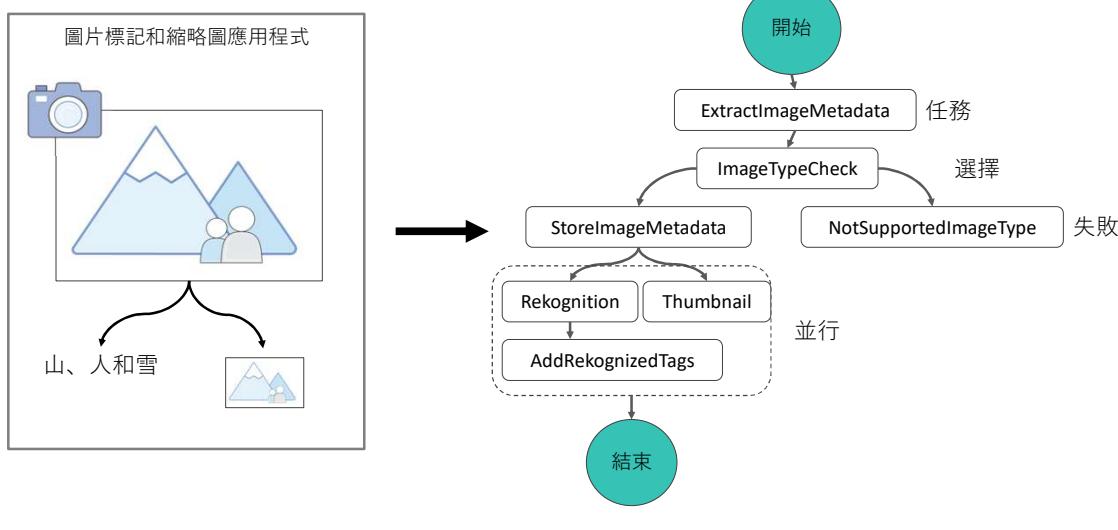
68

借助 AWS Step Functions，您可以使用基於 JSON 的 Amazon States Language 在 AWS 環境中創建自己的狀態機並讓其自動運行。該語言包含由各種狀態、任務、選項、錯誤處理等組成的結構。

狀態機是可以執行工作的狀態集合。

飲料售賣機就是一種常見的狀態機。機器在開始時處於運行狀態（等待交易），然後在放入錢時轉到飲料選擇狀態。然後進入售賣狀態，將飲料提供給客戶。完成後，售賣機返回運行狀態。

狀態



AWS

© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

69

有限狀態機可以將演算法表示為許多狀態、其關係，及其輸入和輸出。狀態是您狀態機中的元素。各個狀態可以根據其輸入作出決定，執行操作並將輸出傳遞給其他狀態。

狀態類型

任務	由狀態機執行的一個工作單元
選擇	向狀態機添加分支邏輯
失敗	停止運行中的狀態機並將其標記為失敗
成功	成功停止運行中的狀態機
傳遞	將其輸入傳遞到其輸出，而不執行任何工作
等待	延遲指定時間，然後再繼續
並行	創建在狀態機中運行的並行分支
映射	動態反覆運算步驟



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

70

狀態可以在狀態機中執行各種函數：

- 在您的狀態機中執行一些工作（任務狀態）
- 在狀態機分支之間進行選擇（選擇狀態）
- 停止正在運行的狀態機，返回失敗或成功（失敗狀態或成功狀態）
- 將其輸入傳遞到其輸出或者注入一些固定資料（傳遞狀態）
- 提供一定時間量的延遲或直至指定時間和日期（等待狀態）
- 開始創建在狀態機中運行的並行分支（並行狀態）
- 動態地反覆運算步驟（映射狀態）

有關狀態類型的更多資訊，請參閱 AWS 文檔中的[狀態](#)。

Amazon States Language

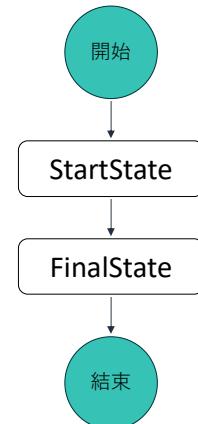
使用 Amazon States Language
以 JSON 格式定義工作流

```
{  
    "Comment": "An example of the ASL.",  
    "StartAt": "StartState",  
    "States": {  
        "StartState": {  
            "Type": "Task",  
            "Resource": "arn:aws:lambda:us-east...",  
            "Next": "FinalState"  
        }  
        "Finalstate": {  
            "Type": "Task",  
            "Resource": "arn:aws:lambda:us-east...",  
            "End": true  
        }  
    }  
}
```



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

在 Step Functions 控制台
中視覺化工作流



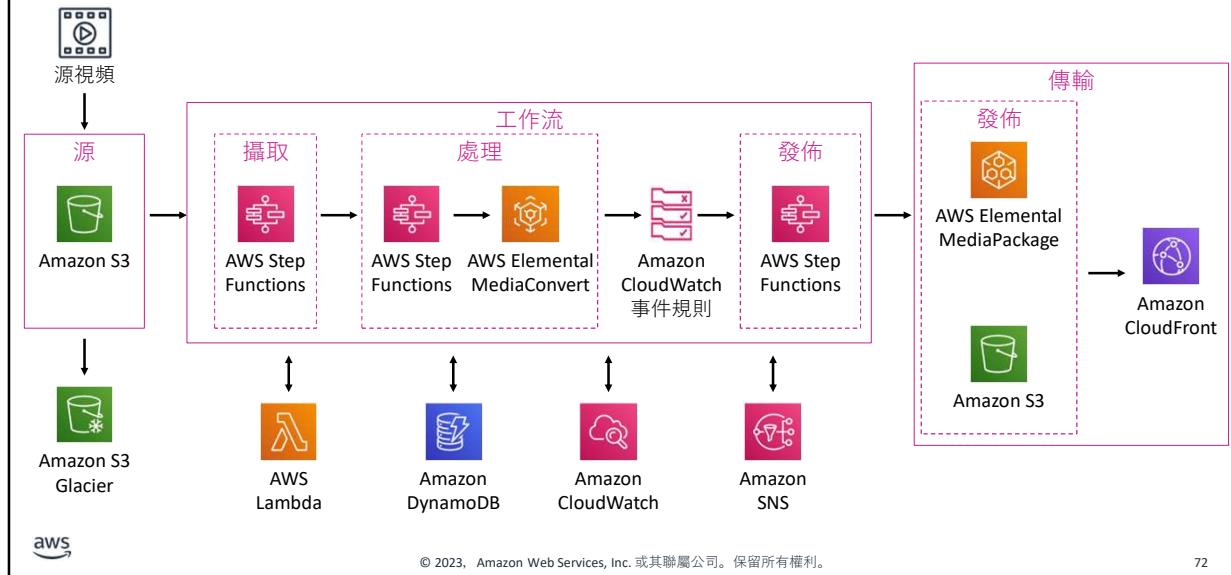
71

您可以使用 Amazon States Language 來定義狀態機。Amazon States Language 是一種基於 JSON 的結構化語言。然後，AWS Step Functions 在即時圖形視圖中表示 JSON 結構。通過這種方式，您可以直接在 Step Functions 控制台中視覺化狀態機。

在這裡，您可以看到具有兩種任務狀態類型的狀態機的示例。

有關 Amazon States Language 的更多資訊，請參閱 [AWS 文檔](#)。

AWS Step Functions 示例： 視頻點播 (VOD) 架構



72

此架構圖表顯示了視頻點播 (VOD) 解決方案中 AWS Step Functions 的示例使用案例。此架構中的另一個關鍵元件是 AWS Elemental MediaConvert，這是一款具有廣播級功能的基於檔的視頻轉碼服務。它使您能夠創建視頻點播 (VOD) 內容，實現大規模的廣播和多螢幕傳輸。

在該解決方案中，源視頻和中繼資料檔經過攝取和處理，可在各種設備上播放。

- AWS Step Functions 可創建攝取、處理和發佈階躍函數。
- AWS Lambda 函數執行每個步驟的工作並處理錯誤消息。
- Amazon S3 存儲桶可存儲源媒體檔和目標媒體檔。
- Amazon CloudWatch 用於日誌記錄。
- AWS Elemental MediaConvert 通知的 Amazon CloudWatch 事件規則。
- Amazon DynamoDB 表存儲通過工作流捕獲的資料。
- Amazon SNS 主題發送編碼、發佈和錯誤通知。
- 轉碼的媒體檔已存儲，以便通過 Amazon CloudFront 按需交付給用戶。

有關此架構的更多資訊，請參閱[架構概覽](#)。

第 7 節要點



AWS

- AWS Step Functions 是一項 Web 服務，讓您能夠使用視覺化工作流來協調分散式應用程式和微服務的元件
- 借助 AWS Step Functions，您可以在 AWS 環境中創建自己的狀態機並讓其自動運行
- AWS Step Functions 為您管理應用程式的邏輯，並實施基本基元，例如順序或並行分支和超時
- 您可以使用 Amazon States Language 來定義狀態機

© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

73

本模組中這節內容的要點包括：

- AWS Step Functions 是一項 Web 服務，讓您能夠使用視覺化工作流來協調分散式應用程式和微服務的元件
- 借助 AWS Step Functions，您可以在 AWS 環境中創建自己的狀態機並讓其自動運行
- AWS Step Functions 為您管理應用程式的邏輯，並實施基本基元，例如順序或並行分支和超時
- 您可以使用 Amazon States Language 來定義狀態機

模組 13 – 挑戰實驗： 為咖啡館實施無伺服器架構



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

74

現在您將完成模組 13 – 挑戰實驗：為咖啡館實施無伺服器架構。

業務需求：無伺服器環境



Sofía 和 Nikhil 希望進一步解耦架構，並將 cron 作業移動到能夠很好地擴展且能夠降低成本的託管式無伺服器環境中。



Frank 和 Martha 希望獲得有關網站上所有訂單的每日電子郵件報告。Olivia 建議 Sofía 和 Nikhil 將非業務關鍵型報告任務與生產 Web 伺服器實例分離開。



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

75

Frank 和 Martha 希望通過電子郵件獲得有關網站上所有訂單的每日報告。Frank 希望能夠預測需求，以便烘焙正確數量的甜點（減少浪費）。Martha 希望識別咖啡館業務中的任何模式（分析）。目前，Sofía 已經在 Web 伺服器實例上設置了 cron 作業，將這些每日訂單報告電子郵件消息發送給 Frank 和 Martha。但是，cron 作業為資源密集型作業，會降低 Web 伺服器的性能。

Olivia 建議 Sofía 和 Nikhil 將非業務關鍵型報告任務分離開。Sofía 和 Nikhil 希望進一步解耦架構，並將 cron 作業移動到能夠很好地擴展且能夠降低成本的託管式無伺服器環境中。

挑戰實驗：任務

1. 下載原始程式碼
2. 在 VPC 中創建 *DataExtractor* Lambda 函數
3. 創建 *salesAnalysisReport* Lambda 函數
4. 創建 SNS 主題
5. 創建 SNS 主題的電子郵件訂閱
6. 測試 *salesAnalysisReport* Lambda 函數
7. 將 Amazon EventBridge 事件設置為每天觸發 Lambda 函數



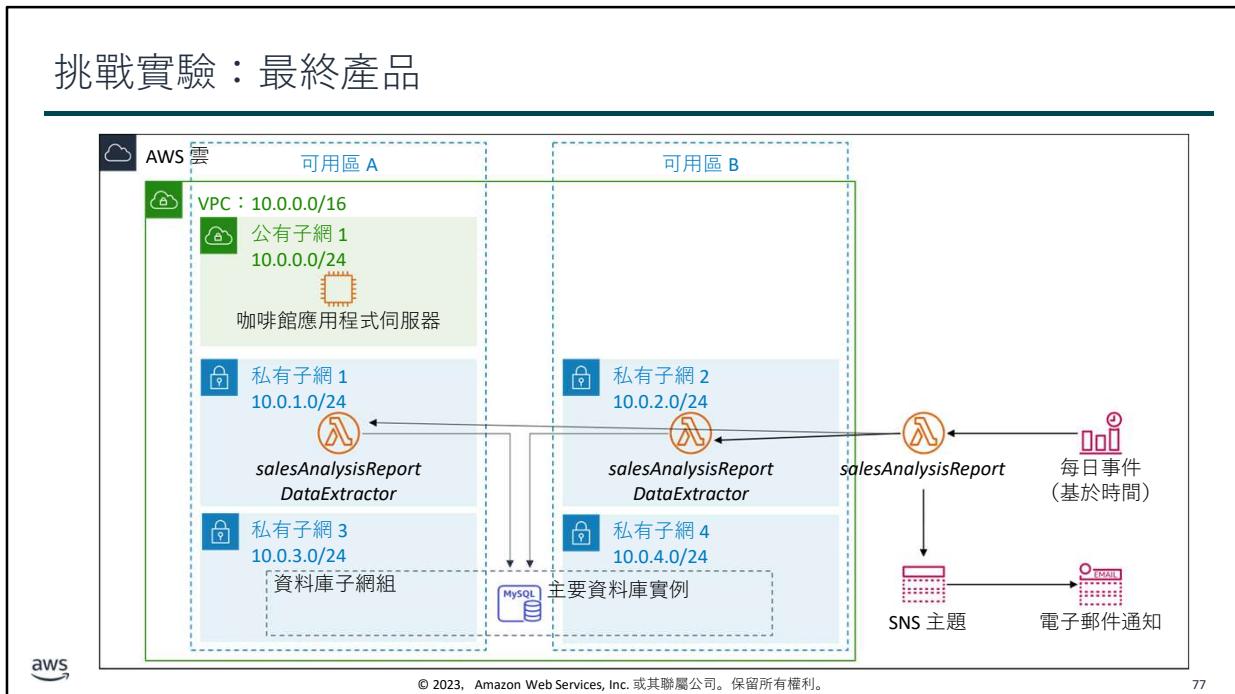
© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

76

在本挑戰實驗中，您將完成以下任務：

1. 下載原始程式碼
2. 在 VPC 中創建 *DataExtractor* Lambda 函數
3. 創建 *salesAnalysisReport* Lambda 函數
4. 創建 SNS 主題
5. 創建 SNS 主題的電子郵件訂閱
6. 測試 *salesAnalysisReport* Lambda 函數
7. 將 Amazon EventBridge 事件設置為每天觸發 Lambda 函數

挑戰實驗：最終產品



該圖總結了您完成實驗後將構建的內容。

內容說明：三層架構圖，其中咖啡館應用程式伺服器位於公有子網 1，銷售分析報告資料提取器 Lambda 函數位於私有子網 1 和 2，主要資料庫實例位於橫跨私有子網 3 和 4 的資料庫子網組。每日事件會觸發銷售分析報告 Lambda 函數，該函數會觸發資料提取器 Lambda 函數，還會從 SNS 主題發送電子郵件通知。內容說明結束。



大約 90 分鐘



開始模組 13 – 挑戰實驗： 為咖啡館實施無伺服器架構



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

78

現在可以開始挑戰實驗了。

挑戰實驗總結： 要點



完成這個挑戰實驗之後，您的講師可能會帶您討論此挑戰實驗的要點。

模組總結

模組 13：構建微服務和無伺服器架構

 AWS

© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

現在該複習本模組，並完成最後的知識考核和對實踐認證考試問題的討論了。

模組總結

總的來說，在本模組中，您學習了如何：

- 指出微服務的特性
- 將整體式應用程式重構為微服務，然後使用 Amazon ECS 部署容器化微服務
- 解釋無伺服器架構的含義
- 使用 AWS Lambda 實施無伺服器架構
- 描述 Amazon API Gateway 的通用架構
- 描述 AWS Step Functions 支持的工作流類型



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

81

總的來說，在本模組中，您學習了如何：

- 指出微服務的特性
- 將整體式應用程式重構為微服務，然後使用 Amazon ECS 部署容器化微服務
- 解釋無伺服器架構的含義
- 使用 AWS Lambda 實施無伺服器架構
- 描述 Amazon API Gateway 的通用架構
- 描述 AWS Step Functions 支持的工作流類型

完成知識考核



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

82

現在該完成本模組的知識考核了。



考試樣題

企業託管了 10 項微服務，每項微服務都位於單個 Classic Load Balancer 後面的 Auto Scaling 組中。每個 EC2 實例均以最佳負載運行。

以下哪些措施可以讓企業在不影響性能的情況下降低成本？

選項	答案
A	減少每個 Classic Load Balancer 後面的 EC2 實例數量。
B	在 Auto Scaling 組啟動配置中更改實例類型。
C	更改 Auto Scaling 組的最大大小，但保留所需容量。
D	將 Classic Load Balancer 與 Application Load Balancer 互換。

思考答案選項，並根據關鍵字排除錯誤選項。

考試樣題答案

企業託管了 10 項微服務，每項微服務都位於單個 Classic Load Balancer 後面的 Auto Scaling 組中。每個 EC2 實例均以最佳負載運行。

以下哪些措施可以讓企業在不影響性能的情況下降低成本？

正確答案是 D。

問題的要點是在不影響性能的情況下降低成本，並將 Classic Load Balancer 替換為單個 Application Load Balancer。

以下是要識別的關鍵字：在不影響性能的情況下降低成本，並將 Classic Load Balancer 替換為單個 Application Load Balancer。

正確答案是 D：“將 Classic Load Balancer 替換為單個 Application Load Balancer。”通過排除法，選項 D 是正確的。您可以使用單個 Application Load Balancer 將請求路由到應用程式的所有服務，而不是為每個微服務使用一個 Classic Load Balancer。

錯誤的答案：

- 選項 A 可以排除 – 因為 EC2 實例以最佳負載運行，因此如果您減少實例的數量，它們將變得超載。
- 選項 B 可以排除 – 選擇較大的實例大小並不會降低成本。而如果減小實例大小，EC2 實例就會超載，因為它們已經以最佳負載運行。
- 選項 C 也可以排除 – 如果增加最大大小，而保留所需容量，成本將保持不變或增加。

其他資源

- [將整體式應用程式拆分為微服務項目](#)
- [使用 AWS Lambda 的無伺服器架構白皮書](#)
- [使用 Amazon API Gateway 和 AWS Lambda 的 AWS 無伺服器多層架構白皮書](#)
- [AWS Well-Architected Framework：無伺服器應用程式詳解白皮書](#)
- [創建和使用 Lambda 函數教程](#)



© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

85

如果您想進一步瞭解本模組中涵蓋的主題，以下額外資源可能會對您有所幫助：

- [將整體式應用程式拆分為微服務項目](#)
- [使用 AWS Lambda 的無伺服器架構白皮書](#)
- [使用 Amazon API Gateway 和 AWS Lambda 的 AWS 無伺服器多層架構白皮書](#)
- [AWS Well-Architected Framework：無伺服器應用程式詳解白皮書](#)
- [創建和使用 Lambda 函數教程](#)

謝謝

想要提出更正、回饋或其他問題？
請聯繫我們：<https://support.aws.amazon.com/#/contacts/aws-academy>。
所有商標均為各自擁有者的財產。

© 2023, Amazon Web Services, Inc. 或其聯屬公司。保留所有權利。

86

感謝您完成本模組的學習。