了解並實作微前端 AWS

AWS 規範指南

AWS 規範指南: 了解並實作微前端 AWS

Table of Contents

了解和實施微前端 AWS

Amazon Web Services (貢獻者)

2024 年 7 月 (文件歷史記錄)

隨著組織追求靈活性和可擴展性，傳統的整體式架構通常會成為瓶頸，阻礙快速開發和部署。微型前端透過將複雜的使用者介面分解為可自主開發、測試和部署的小型獨立元件來減輕這種情況。這種方法可提高開發團隊的效率，並促進後端和前端之間的協作，從而促進分散式系統的一 end-to-end 致性。

此規範指引旨在協助不同專業領域的 IT 領導者、產品擁有者和架構師了解微型前端架構，並在 Amazon Web Services 上建置微型前端應用程式 ()。AWS

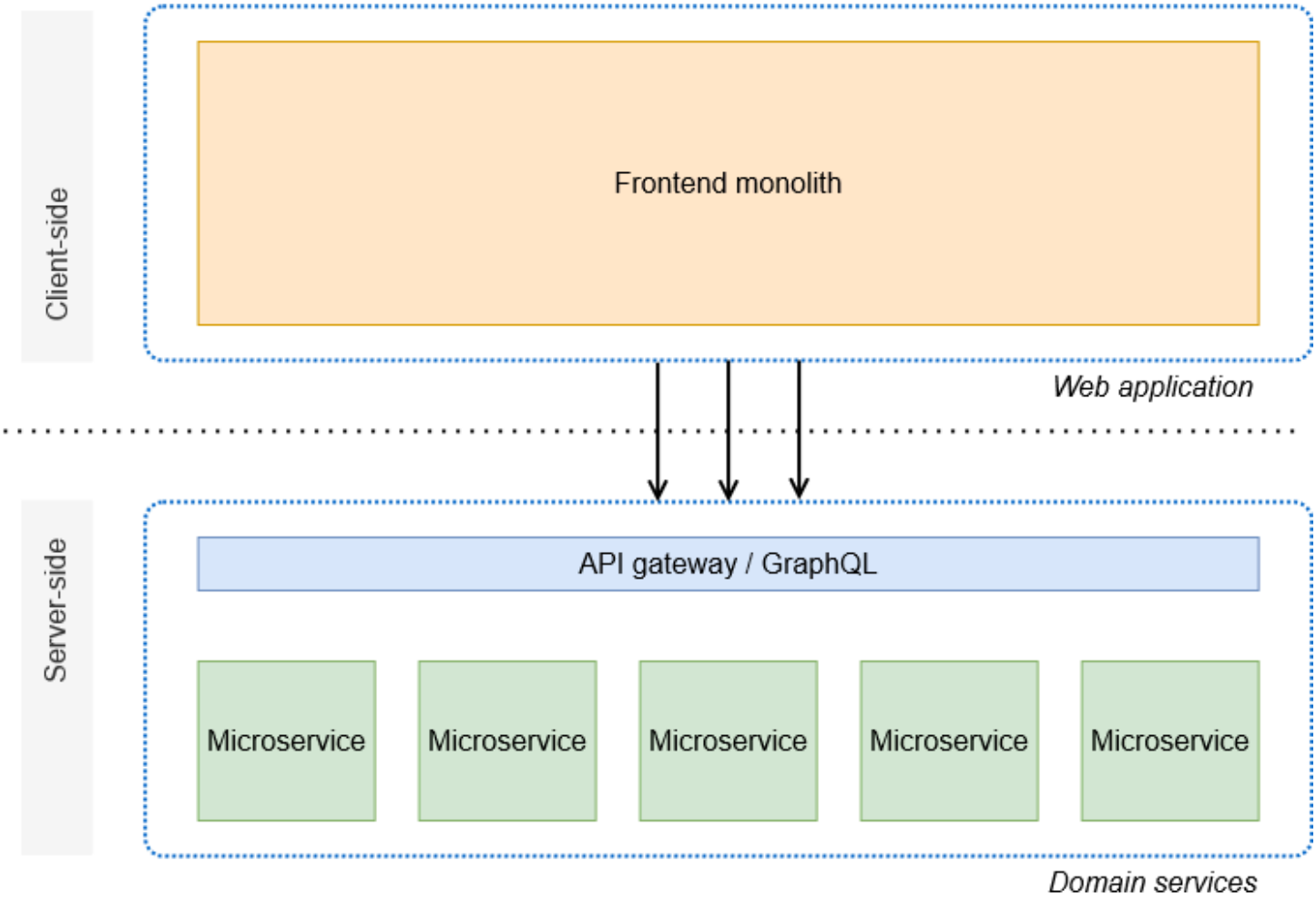
概觀

微前端是建立在將應用程式前端分解為獨立開發和部署成品之上的架構。當您將大型前端分割為自主軟體成品時，您可以封裝商務邏輯並減少相依性。這支持更快，更頻繁地交付產品增量。

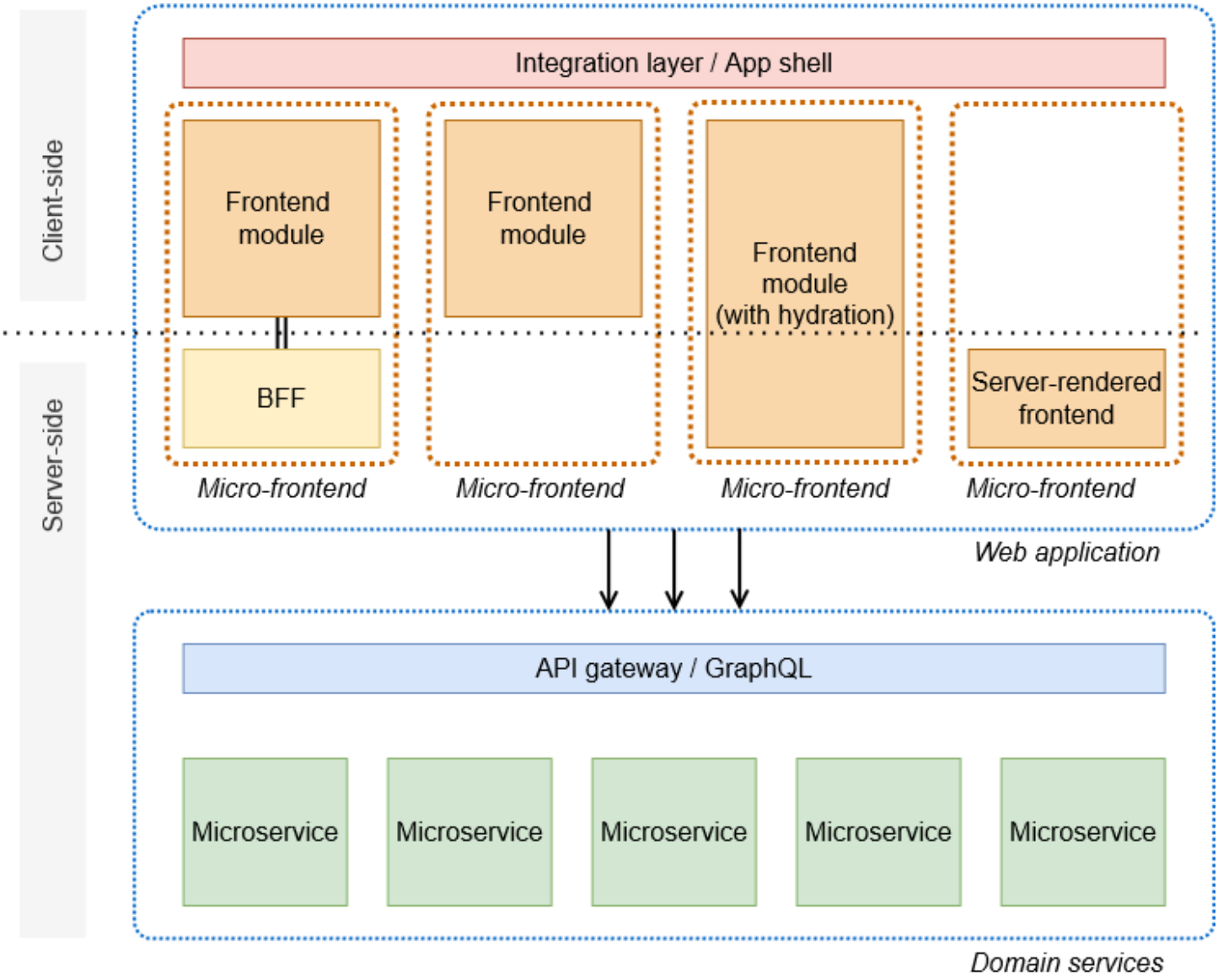
微前端類似於微服務。事實上，「微前端」一詞來源於「微服務」一詞，其目的是傳達微服務作為前端的概念。微服務架構通常會將後端中的分散式系統與整合式前端，但微前端則是獨立的分散式前端服務。您可以透過兩種方式設定這些服務：

- 僅前端，與共用 API 層整合，在後面執行微服務架構
- 全棧，這意味著每個微前端都有自己的後端實現。

下圖顯示了傳統的微服務架構，其前端整體式使用 API 閘道連接到後端微服務。



下圖顯示了具有不同微服務實現的微型前端架構。



如前圖所示，您可以將微前端與用戶端轉譯或伺服器端轉譯架構搭配使用：

- 用戶端呈現的微前端可以直接使用集中式 API Gateway 公開的 API。
- 小組可以在有界上下文中創建一個 backend-for-frontend （ BFF ），以減少前端對 API 的聊天性。
- 在服務器端，微前端可以通過使用稱為水化的技術在客戶端增強服務器端方法來表達。當瀏覽器呈現頁面時，關聯的頁面會 JavaScript 被水合，以允許與 UI 元素進行互動，例如按一下按鈕。
- 微前端可以在後端呈現，並使用超鏈接路由到網站的新部分。

微型前端非常適合想要執行以下操作的組織：

- 與多個團隊在同一個項目上工作進行擴展。

- 擁抱決策的分散化，讓開發人員能夠在已識別的系統邊界內進行創新。

這種方法顯著減少了團隊的認知負荷，因為他們成為負責系統的特定部分。它可以提高業務敏捷性，因為可以對系統的一部分進行修改，而不會中斷其餘部分。

微型前端是一種獨特的架構方法。雖然有不同的方法來構建微前端，它們都有共同的特徵：

- 微型前端架構由多個獨立元素組成。結構類似於後端微服務所發生的模組化。
- 微前端完全負責其有界上下文中的前端實現，其中包括以下內容：
  - 使用者界面
  - 資料
  - 狀態或工作階段
  - 业务逻辑
  - 流程

有界的上下文是一個內部一致的系統，具有精心設計的界限，可以調解可以進入和退出的內容。微型前端應盡可能少地與其他微型前端共享業務邏輯和數據。無論共享需要發生，它都會通過明確定義的接口（例如自定義事件或反應流）進行。但是，當涉及到一些交叉問題（例如設計系統或日誌庫）時，歡迎有意共享。

建議的模式是使用跨職能團隊來構建微前端。這意味著每個微型前端都是由從後端到前端工作的同一個團隊開發的。從編碼到生產中系統的操作化，團隊擁有權至關重要。

本指引並不打算推薦一種特定的方法。相反，它討論了不同的模式，最佳實踐，權衡以及架構和組織考慮因素。

基礎概念

微型前端架構深受三個先前建築概念的啟發：

- 領域驅動設計是將複雜應用程序構建為一致領域的心理模型。
- 分散式系統是將應用程式建置為鬆散耦合子系統的一種方法，這些子系統是獨立開發並在自己的專用基礎架構上執行的。
- 雲端運算是以模型執行 IT 基礎架構即服務的一 pay-as-you-go 種方法。

領域驅動設計

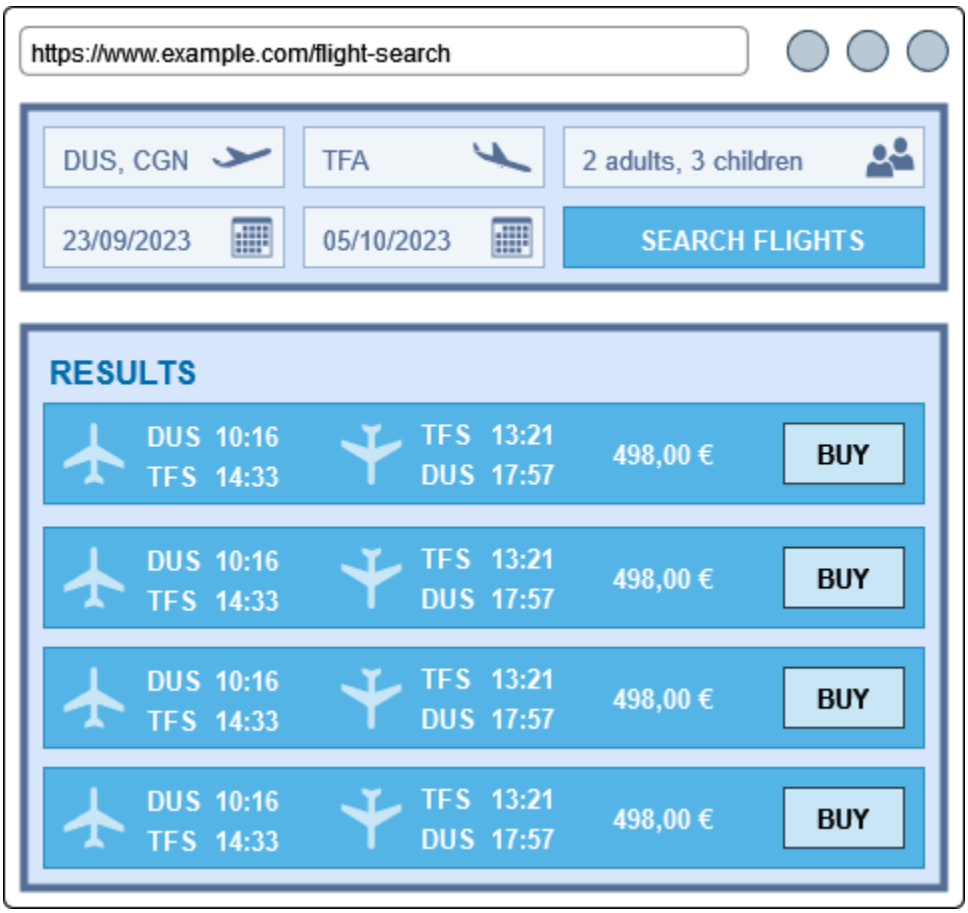
領域驅動設計（DDD）是埃里克·埃文斯開發的範例。Evans 在 2003 年著作《領域驅動設計：解決軟體核心的複雜性》中，假設軟體開發應該是由業務問題而不是技術問題所驅動。Evans 建議 IT 專案首先開發一種無處不在的語言，以幫助技術和領域專家找到共同的理解。基於這種語言，他們可以制定一個相互理解的商業現實模型。

很明顯，因為這種方法可能是，許多軟件項目遭受業務和 IT 之間的斷開。這些斷線通常會造成重大誤解，導致預算超支、品質下降或專案失敗。

Evans 引入了多個其他重要術語，其中一個是有界的上下文。有界的上下文是大型 IT 應用程序的自包含部分，其中包含僅針對一個業務問題的解決方案或實現。一個大型的應用程序將由通過集成模式鬆散耦合的多個有界上下文組成。這些有界的上下文甚至可以有自己的無處不在的語言的方言。例如，應用程式付款內容中的使用者可能與傳送內容中的使用者有不同的層面，因為付款期間的託運概念無關緊要。

埃文斯沒有定義有界上下文應該有多小或大。大小由軟件項目確定，並且可能會隨著時間的推移而發展。上下文邊界的良好指標是實體（域對象）和業務邏輯之間的凝聚程度。

在微前端的背景下，領域驅動的設計可以通過複雜的網頁例如航班預訂頁面來說明。



在此頁面上，主要構建塊是搜索表單，過濾器面板和結果列表。若要識別邊界，您必須識別獨立的功能前後關聯。此外，請考慮非功能性方面，例如可重用性，性能和安全性。最重要的指標是「屬於在一起的東西」是他們的溝通模式。如果架構中的某些元素必須經常通信並交換複雜的信息，則它們可能共享相同的有界上下文。

單獨的 UI 元素，如按鈕不是有界的上下文，因為它們在功能上不是獨立的。此外，整個頁面不適合有界的前後關聯，因為它可以分解為較小的獨立前後關聯。合理的方法是將搜尋表單視為一個有界前後關聯，並將結果清單視為第二個有界前後關聯。這兩個有界的上下文中的每一個現在都可以實現為一個單獨的微前端。

分散式系統

為了簡化維護並支持發展能力，大多數不平凡的 IT 解決方案都是模塊化的。在這種情況下，模塊化意味著 IT 系統由可識別的構建模塊組成，這些構建模塊通過接口進行分離以實現關注點分離。

除了是模塊化的，分佈式系統應該是獨立的系統在他們自己的權利。在一個僅僅模塊化的系統中，每個模塊都被理想地封裝並通過接口公開其功能，但它不能獨立部署，甚至不能單獨部署，甚至可以單獨運

行。此外，模塊通常遵循相同的生命週期作為同一系統的其他模塊的一部分。另一方面，分散式系統的建構區塊都有自己的生命週期。應用領域驅動的設計範例，每個構建塊可以處理一個業務領域或子域，並存在於其自己的界限環境中。

當分佈式系統在構建期間進行交互時，常見的方法是開發快速識別問題的機制。例如，您可能會採用類型語言並在單元測試上進行大量投資。多個團隊可以在模塊的開發和維護上進行協作，這些模塊通常作為庫分發，供系統使用 npm，Apache Maven 和 pip 等工具一起使用。 NuGet

在執行階段期間，互動的分散式系統通常由個別團隊擁有。消耗相依性會導致作業複雜性，因為錯誤處理、效能平衡和安全性。集成測試和可觀察性的投資是降低風險的基礎。

當今分佈式系統最常見的例子是微服務。在微服務架構中，後端服務是由領域驅動的（而不是由 UI 或身份驗證等技術問題驅動），並由自治團隊擁有。微型前端共用相同的原則，將解決方案範圍擴展到前端。

雲端運算

雲端運算是透過 pay-as-you-go 模型購買 IT 基礎架構即服務的一種方式，而不是建立自己的資料中心，並購買硬體以便在內部部署進行操作。雲計算提供了以下幾個優點：

- 您的組織能夠嘗試新技術，而無需預先做出龐大的長期財務承諾，從而獲得顯著的業務敏捷性。
- 透過使用雲端供應商 AWS，您的組織就可以存取廣泛的低維護和高度整合的服務產品組合 (例如 API 閘道、資料庫、容器協調和雲端功能)。使用這些服務可讓您的員工更加專注於使您的組織與競爭對手區分開來的工作。
- 當您的組織準備好在全球推出解決方案時，您可以將解決方案部署到世界各地的雲端基礎架構。

雲端運算透過提供高度管理的基礎架構來支援微前端。這使得跨職能團隊的 end-to-end 擁有權變得更加容易。雖然團隊應該具備強大的營運知識，但是基礎結構佈建、作業系統更新和網路等手動工作會令人分心。

由於微前端生活在有界的環境中，所以團隊可以選擇最合適的服務來運行它們。例如，團隊可以在雲端函式和運算容器之間進行選擇，並且可以在不同風格的 SQL 和 NoSQL 資料庫或記憶體內快取之間進行選擇。團隊甚至可以在高度整合的工具組上建置他們的微前端 AWS Amplify，例如為無伺服器基礎架構提供預先設定的建置區塊。

比較微前端與替代架構

與所有架構策略一樣，採用微型前端的決定必須根據組織原則所指引的評估標準為基礎。微前端有優點和缺點。如果您的組織決定使用微前端，您必須制定適當的策略來解決分散式系統的挑戰

在選擇應用程式架構時，微前端最受歡迎的替代方案是單一頁面應用程式 (SPA) 前端的巨石、n 層應用程式和微服務。這些都是有效的方法，每種方法都有優點和缺點。

石柱

一個不需要頻繁更改的小型應用程序可以非常快速地作為一個整體交付。即使在預計顯著增長的情況下，巨石也是自然的第一步。之後，整體可以被淘汰或重構為更靈活的結構。從整體開始，您的組織可以進入市場，獲得客戶反饋並更快地改進產品。

但是，如果不仔細維護或代碼庫的大小隨著時間的推移而增加，單片應用程序往往會降級。當多個團隊為相同的代碼庫做出重大貢獻時，他們很少都有助於其維護和操作。這會導致責任失衡，這會影響速度並導致效率低下。同時，隨著代碼庫的發展，整體模塊之間的意外耦合會導致意外的副作用。這些副作用可能導致故障和中斷.

N 層應用

具有相對靜態進化步伐的更複雜的應用程式可以建置為三層架構 (簡報、應用程式、資料)，並在前端和後端之間使用 REST 或 GraphQL 層。這更加靈活，不同層次的團隊可以在一定程度上獨立開發。n 層應用程式的缺點是部署功能要困難得多。前端和後端是通過 API 合同分離的，因此必須將重大更改一起部署，否則必須對 API 進行版本控制。

請考慮下列常見案例：如果發行新功能需要變更資料結構描述，產品擁有者可能需要數天的時間才能與前端團隊共同意一組功能。然後，前端團隊將要求後端團隊開發和釋放他們身邊的功能。後端團隊將與資料擁有者合作發行資料庫結構描述更新。接下來，後端團隊將發布新版本的 API，以便前端團隊可以開發和發布他們的更改。在這個案例中，將所有變更傳播至生產環境可能需要數週甚至數月的時間，因為每個小組都有自己的積壓、優先順序，以及圍繞開發、測試和發行變更的機制。

微服務

在微服務架構中，後端被分解為小型服務，每個服務都在有界上下文中解決特定的業務問題。通過公開明確定義的接口合同，每個微服務也與其他服務強烈分離。

值得一提的是，有界的上下文和接口合同也應該存在於精心製作的巨石和 n 層體系結構中。然而，在微服務架構中，通訊會透過網路 (通常是 HTTP 通訊協定) 進行，而且服務具有專用的執行階段基礎結構。這支持每個後端服務的獨立開發，交付和操作。

選擇符合您需求的方法

Monoliths 和 n 層架構將多個域問題捆綁在一個技術成品中。這使得諸如依賴關係和內部數據流之類的方面易於管理，但這使得新功能的交付變得更加困難。為了維持一致的程式碼基底，團隊通常會投入時間來重構和解耦，因為他們必須處理龐大的程式碼基底。

由少數團隊開發的應用程式可能不需要移至微前端所帶來的額外複雜性。如果團隊沒有支付高耦合和長交貨時間來釋放更改的罰則尤其如此。

總而言之，對於複雜且快速移動的應用程式而言，更複雜且分散式的架構通常是正確的選擇。對於中小型應用程序，分佈式體系結構不一定優於單體架構，特別是如果應用程序不會在短時間內顯著發展。

微前端的架構決策

針對應用程式套用微型前端架構模式的團隊，必須儘早就架構做出幾項決策：

- 微前端的識別和界限的定義
- 使用微型前端撰寫頁面和檢視
- 跨微前端的路由、狀態管理和通訊
- 管理跨領域問題的相依性

以下幾節將更深入地介紹這些主題。

在做出架構決策時，必須具備正確的指標並瞭解使用模式應用程式特徵和權衡。例如，與視訊編輯工具或可觀察性儀表板相比，電子商務網站具有不同的特性和使用模式。

具有高流量和短工作階段深度的公開應用程式可針對初始頁面載入量度進行最佳化，例如互動時間 (TTI) 和首次內容繪圖 (FCP)。相反地，使用者在一天開始時登入並持續與之互動的應用程式，可能會針對應用程式內體驗進行最佳化。應用程式小組可能會在每次瀏覽之後針對「首次輸入延遲」(FID) 量度進行最佳化，而非初始頁面載入。

公共網站必須滿足各種瀏覽器環境。對用戶端環境具有已知限制的企業應用程式，可以根據其限制來最佳化其微型前端組合。

架構決策沒有一個正確的選擇。瞭解權衡、業務營運的內容、使用模式和指標，以指導適合每個應用程式的決策。

識別微前端邊界

為了提高團隊自主性，應用程序提供的業務能力可以分解為幾個微前端，而彼此之間的依賴性最小。

遵循先前討論的 DDD 方法，專案團隊可以將應用程式網域劃分為企業子網域和有界的前後關聯。然後，自治團隊可以擁有其邊界上下文的功能，並以微觀前端的形式提供這些情境。如需關注點分離的詳細資訊，請參閱無伺服器土地圖表。

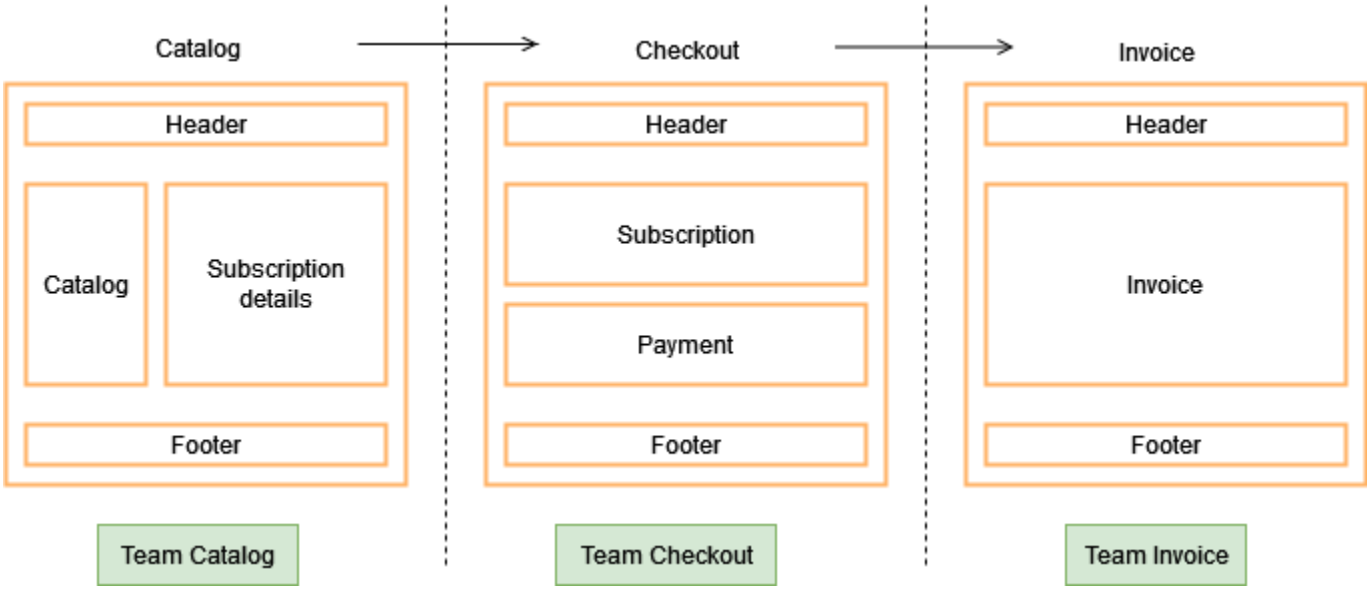
定義良好的有界上下文應該最大限度地減少功能重疊和跨上下文運行時通信的需求。所需的通信可以使用事件驅動的方法來實現。這與微服務開發的事件驅動架構沒有什麼不同。

架構良好的應用程式也應該支援新團隊提供 future 的擴充功能，以便為客戶提供一致的體驗。

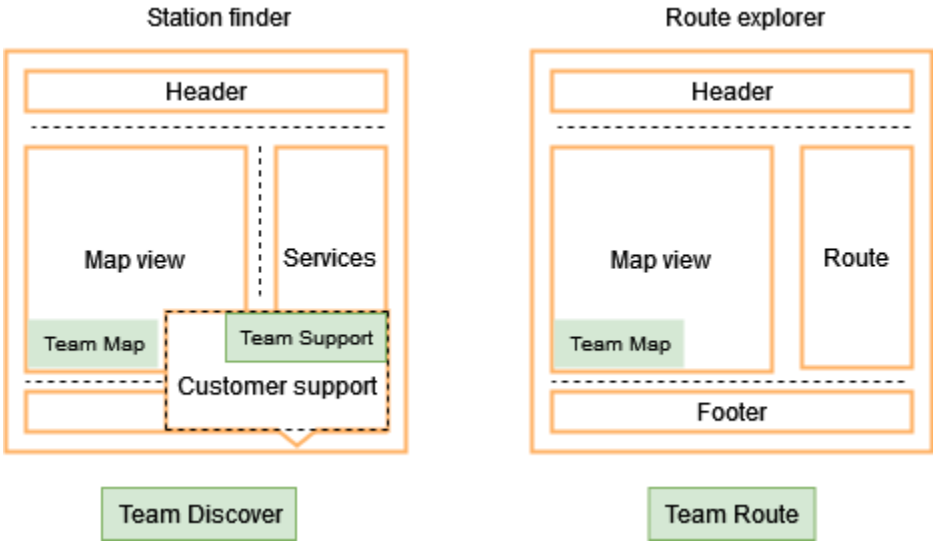
如何將整體應用程式切割成微前端

「概觀」一節包含識別網頁上獨立功能前後關聯的範例。出現幾種在用戶界面上拆分功能的模式。

例如，當業務網域形成使用者旅程的階段時，可以在前端套用垂直分割，其中使用者旅程中的檢視集合會以微前端形式提供。下圖顯示垂直分割，其中「目錄」、「結帳」和「發票」步驟是由不同團隊作為單獨的微型前端進行交付。



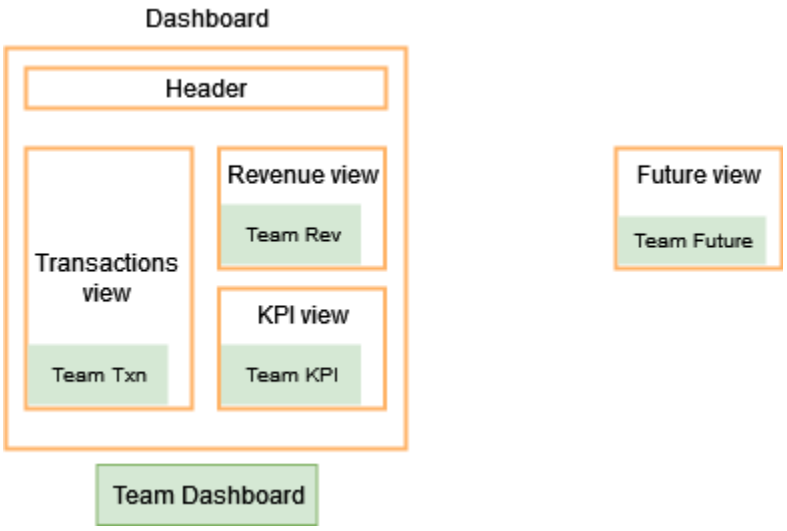
對於某些應用程序，單獨垂直拆分可能是不夠的。例如，某些功能可能需要在許多視圖中提供。對於這些應用程式，您可以套用混合分割。下圖顯示混合分割解決方案，其中 Station 搜尋器和 Route 總管的微型前端都使用地圖檢視功能。



入口類型或儀表板類型的應用程序通常將前端功能集成在一個單一視圖中。在這些類型的應用程式中，每個 Widget 都可以作為微型前端交付，而託管應用程式則定義微前端應實作的限制和介面。

此方法為微前端提供了一種機制，以處理諸如視埠大小調整、驗證提供者、組態設定和中繼資料等問題。這些類型的應用程式最佳化可擴充性。新團隊可以開發新功能以擴展儀表板功能。

下圖顯示由屬於「小組控制面板」一部分的三個個別團隊開發的儀表板應用程式。



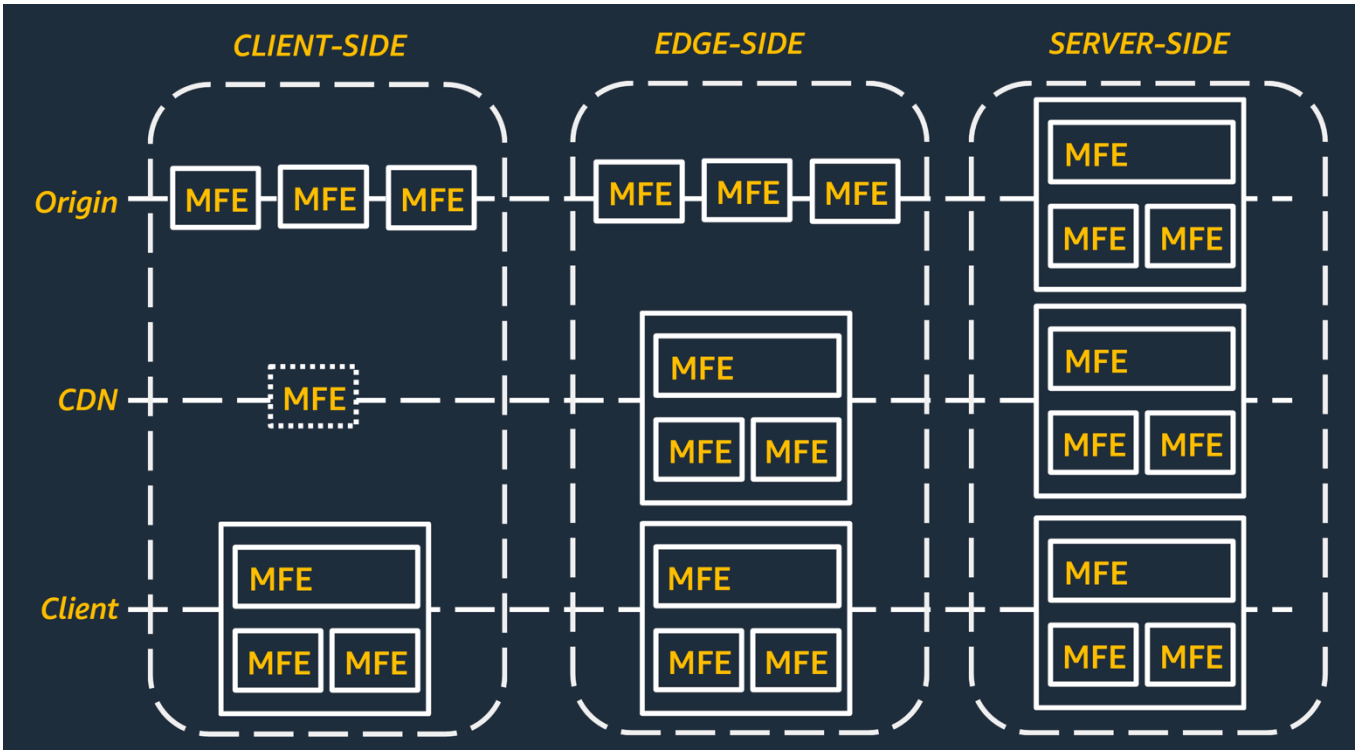
在圖中，future 的檢視表示新團隊為擴展「團隊控制面板」和儀表板功能而開發的新功能。

入口網站和儀表板應用程式通常會在 UI 中使用混合分割來構成功能。微型前端可透過明確定義的設定進行配置，包括位置和大小限制。

使用微型前端撰寫頁面和檢視

您可以使用用戶端構成、邊緣端構成和伺服器端構成來撰寫應用程式的檢視。組成模式在必要的團隊技能，容錯能力，性能和緩存行為方面具有不同的特徵。

下圖顯示構成如何發生在微型前端架構的用戶端、邊緣端和伺服器端層。



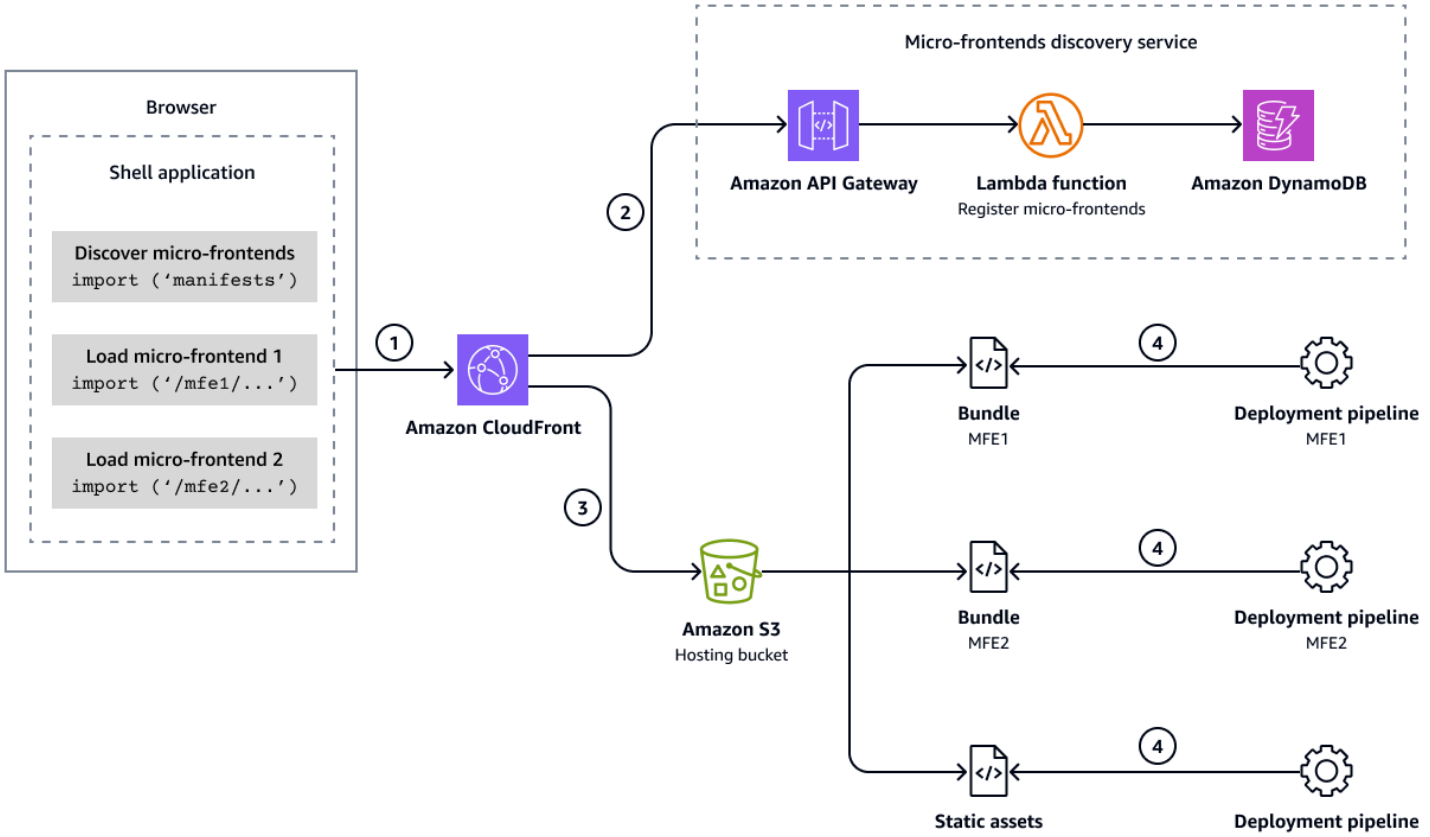
客戶端，邊緣端和服務器端層將在以下各節中討論。

用戶端合成

在用戶端 (瀏覽器或行動網頁檢視) 上以文件物件模型 (DOM) 片段的形式動態載入和附加微前端。微型前端成品 (例如 JavaScript 或 CSS 檔案) 可從內容傳遞網路 (CDN) 載入，以減少延遲。用戶端構成需要下列項目：

- 擁有和維護 shell 應用程序或微型前端框架的團隊，以便在瀏覽器中的運行時發現，加載和渲染微型前端組件
- HTML、CSS 和前端技術的高技能水平，以及 JavaScript對瀏覽器環境的深入了解
- 優化頁面中 JavaScript 加載的量，以及避免全局命名空間衝突的紀律

下圖顯示無伺服器用戶端組成的範例 AWS 架構。



客戶端構成通過 shell 應用程序發生在瀏覽器環境中。此圖表顯示下列詳細資訊：

1. 殼層應用程式載入後，它會向 Amazon 發出初始請求，以探索 CloudFront要透過資訊清單端點載入的微前端。
2. 資訊清單包含每個微型前端的相關資訊 (例如名稱、URL、版本和後援行為)。這些清單由微前端探索服務提供服務。在圖中，此探索服務是由 Amazon API Gateway、 AWS Lambda 函數和 Amazon DynamoDB 表示。shell 應用程序使用清單信息來請求單個微前端在給定的佈局中構成頁面。
3. 每個微型前端包都是由靜態文件（例如 JavaScript CSS 和 HTML）組成。這些文件託管在亞馬遜簡單存儲服務（Amazon S3）存儲桶中，並通過提供 CloudFront。
4. 團隊可以部署新版本的微前端，並使用他們擁有的部署管道來更新資訊清單資訊。

邊緣側組成

使用邊緣端包含 (ESI) 或伺服器端包含 (SSI) 的外包技術，例如原始伺服器前方的某些 CDN 和 Proxy 所支援的伺服器端包含 (SSI) 來撰寫網頁，然後再透過網路傳送至用戶端。ESI 需要下列條件：

- 具有 ESI 功能的 CDN，或伺服器端微前端前端的代理部署。代理實現，例如哈代理，清漆和 NGINX 支持 SSI。

- 瞭解 ESI 和 SSI 實作的使用與限制。

開始新應用程式的團隊通常不會為其構成模式選擇邊緣構圖。但是，此模式可能會為依賴於包含的舊版應用程式提供途徑。

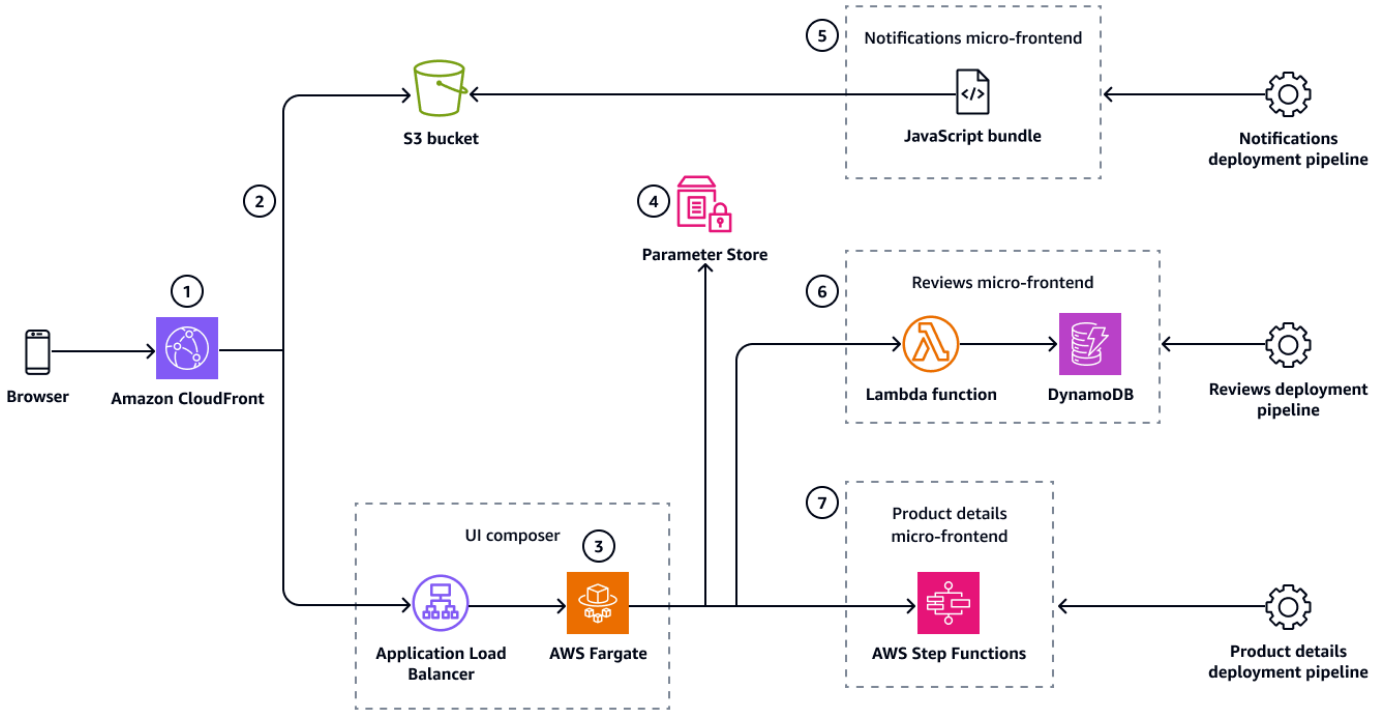
伺服器端合成

使用原始伺服器在邊緣快取頁面之前撰寫頁面。這可以通過傳統技術來完成，例如 PHP，雅加達服務器頁面（JSP）或模板庫，通過包括微前端的片段來構成頁面。您也可以使用在伺服器上執行的 JavaScript 架構，例如 Next.js，透過伺服器端轉譯 (SSR) 在伺服器上撰寫網頁。

在服務器上呈現頁面後，可以在 CDN 上緩存它們以減少延遲。部署新版微前端時，必須重新轉譯頁面，且必須更新快取，才能將最新版本提供給客戶。

伺服器端構成需要深入瞭解伺服器環境，以建立用於部署、探索伺服器端微前端和快取管理的模式。

下圖顯示了服務器端組成。



此圖表包含下列元件與程序：

1. Amazon CloudFront 提供了一個獨特的入口點的應用程序。該發行版有兩個來源：第一個用於靜態文件，第二個用於 UI 作曲家。

2. 靜態檔案託管在 Amazon S3 儲存貯體中。它們由瀏覽器和 UI 撰寫器用於 HTML 模板。
3. UI 撰寫器會在中的容器叢集上執行AWS Fargate。透過容器化解決方案，您可以視需要使用串流功能和多執行緒轉譯。
4. 參數存放區 (的 AWS Systems Manager功能) 可做為基本的微前端探索系統使用。此功能提供 UI 撰寫器用來擷取要使用的微型前端端點的索引鍵值存放區。
5. 通知微型前端會將最佳化的 JavaScript 服務包存放在 S3 儲存貯體中。這會呈現在用戶端上，因為它必須對使用者互動做出反應。
6. 評論微型前端由 Lambda 函數組成，而使用者評論則儲存在 DynamoDB 中。評論微前端完全呈現在服務器端，並輸出一個 HTML 片段。
7. 產品詳細信息微前端是使用的低代碼微前端。AWS Step FunctionsExpress 工作流程可以同步呼叫，它包含用於呈現 HTML 片段和快取層的邏輯。

如需伺服器端構成的詳細資訊，請參閱部落格文章伺服器端轉譯微前端 — 架構。

跨微前端的路由和通訊

路由選項取決於構成方法。通信可以通過減少前端組件之間的耦合來優化通信。

路由

使用垂直分割的用戶端構成的應用程式可以使用伺服器端路由 (多頁應用程式) 或用戶端路由 (單一頁面應用程式)。如果他們對 UI 構成使用混合分割，則需要用戶端路由才能支援頁面上更深層次的微前端路由階層。

使用邊緣端構成和伺服器端構成的應用程式更能配合伺服器端路由，或使用邊緣運算 (例如 Lambda @Edge 與 Amazon) 進行路由。 CloudFront

微前端之間的溝通

使用微型前端架構，我們建議您減少前端元件之間的耦合。減少耦合的一種方法是從同步函數調用轉為異步消息傳遞。

瀏覽器運行時和用戶交互本質上是異步的。事件可以通過消息生產者和消費者之間進行交換。這些事件為跨微前端的通信提供了明確定義的接口。

如果您遵循 DDD 做法來識別微前端的有界內容，下一個步驟是識別必須跨越界限通訊的事件。

事件的消息傳遞機制可以是本機 DOM 事件（`CustomEvents`），JavaScript 事件發射器，或平台團隊提供的反應流庫。微前端發佈事件，並訂閱與其界限內容相關的事件。使用此方法，發布者和訂閱者無需彼此了解。合約是事件定義。有關這方面的可視化表示，請參閱與事件體系結構圖的有界上下文的事件通信一節。

管理跨領域問題的相依性

有意識的相依性管理對於分散式架構 (例如微前端) 的成功至關重要。依賴管理是微型前端開發中最具挑戰性的部分之一。

在微型前端架構中，相依性管理的兩個重要層面是將大型程式碼構件傳輸至用戶端所造成的效能損失，以及運算資源的額外負荷。理想情況下，您的組織需要授權如何維護分散式前端架構中的依賴關係。

使用諸如導入映射和模塊聯盟之類的 Web 標準，強制執行依賴維護的三種可行策略是什麼都不共享。其他方法是反模式，因為它們違反了分佈式架構的基本原則。

在可能的情況下，不分享

無共享方法假設不應共享獨立軟件工件之間的依賴關係，或者至少不應在集成或運行時共享。這意味著，如果兩個微型前端依賴於同一個庫，則每個微型前端都必須在構建時在庫中烘烤並單獨發貨。此外，每個微型前端都必須驗證程式庫不會污染全域命名空間和共用資源。

這導致冗餘，但它是一個有意識的權衡，具有最大的敏捷性。由於沒有共享運行時依賴關係，團隊擁有最大的靈活性，以他們認為有用的任何方式開發軟件，只要他們在解決方案的範圍內這樣做，並且不會破壞任何接口合同。

在微型前端遵循「無共享」原則的平台上，儘可能保持微型前端的輕量級非常重要。這需要熟練且勤奮的開發人員將其微前端最佳化以提高效能，並且不會犧牲開發人員體驗的使用者體驗。

當您共用程式碼時

當您決定共享一些代碼時，可以將其作為庫或運行時模塊共享。例如，前端核心團隊透過 CDN 提供微型前端消費的程式庫。業務價值團隊可以在運行時加載庫，或者他們可以使用軟件包存儲庫來發布其庫。微型前端團隊可以在建置階段針對特定版本的封裝程式庫進行開發，類似於使用混合架構的行動應用程式。

第三個選項是使用私有軟件包註冊表來支持通用庫的構建時集成。這樣可以降低程式庫合約中的中斷變更在執行階段啟動錯誤的風險。但是，這種更保守的方法需要更多的治理來將所有微前端與更新的庫版本同步。

為了改善頁面載入時間，微前端可以將要從 CDN (例如 Amazon) 的快取區塊載入的程式庫相依性外部化。 CloudFront

要管理運行時依賴關係，微前端可以使用導入映射（或諸如庫`System.js`）來指定在運行時加載每個模塊的位置。webpack Module Federation 是另一種指向遠程模塊的託管版本並解決跨獨立微前端的共同依賴關係的方法。

另一種方法是利用初始請求到探索端點來促進動態載入匯入映射。

共用狀態

為了減少微前端的耦合，請務必避免在相同檢視中從所有微前端存取的全域狀態管理，類似於整體架構。例如，從所有微前端訪問全局 Redux 存儲會增加耦合。

消除共享狀態的模式是將其封裝在微前端中，並按照先前所述與異步消息進行通信。

在絕對必要時，為全局狀態引入定義良好的接口，並選擇只讀共享以避免意外行為：

- 存在垂直分割時，您可以使用 URL 元件和瀏覽器儲存體來存取主機環境中的資訊。
- 當您進行混合分割時，您也可以使用 DOM 標準自訂事件或 JavaScript 程式庫 (例如事件發射器或雙向串流)，將資訊傳遞至微前端。

如果您需要跨微型前端共用數個資訊，我們建議您重新檢視微型前端邊界。共享的需求可能是由於業務演進或初始設計低於標準的原因。

也可以使用服務器端會話，其中每個微前端通過使用會話標識符獲取所需的數據。為了減少耦合，重要的是要消除共享狀態，並保持微型前端特定的會話數據分開。

框架和工具

不乏前端框架，例如 Angular 和 Next.js，但其中大多數都不是考慮到微前端的創建。因此，它們有時會缺少解決微型前端架構挑戰的機制。

一般框架考量

本指南並不旨在推薦或比較個別框架。由於多個微前端通常在同一個 Web 應用程序頁面上運行，因此加載和運行時性能是主要考慮的問題。選擇一個引入盡可能少開銷的框架非常重要。

框架根據渲染層進行劃分：

- 用戶端轉譯 (CSR)
- 伺服器端渲染 (SSR)

前端架構包括其他功能，例如靜態網站產生 (SSG)。不過，SSG 只會執行一次。微前端主要是在運行時組成，因此 CSR 和 SSR 是主要選項。

用戶端渲染

對於 CSR，有兩個熱門選項：

- 單水療中心框架
- 模塊聯合

單 SPA 是構成微型前端的輕量級選擇。它解決了微型前端架構中最常見的挑戰，例如在同一頁面中撰寫多個微前端，以及避免相依性衝突。

模塊聯合會開始作為一個插件，由 webpack 5 提供，它解決了微前端體系結構中的絕大多數挑戰，包括跨不同工件的依賴關係管理。模塊聯合 2.0 本地與 Rspack，webpack，電子版本地工作，現在使用. JavaScript

考慮根本不使用框架。根據 caniuse.com 的說法，現代瀏覽器的整體市場份額為 98%，本地提供了諸如自定義元素之類的功能，並且它們足以用於微型前端應用程序。必要時，請將自訂元素與輕量型程式庫結合在一起，以進行事件傳播、國際化或其他特定問題。

伺服器端渲

在 SSR 方面，兩個主要選項比較複雜：

- 擁抱現有的框架，例如 Next.js，並應用使用模塊聯合的微前端原則。
- 使用 HTML-交over-the-wire 換代表微前端的 HTML 片段，並在執行階段在範本中撰寫這些片段。這種方法的一個例子是講台。

API 集成-前端的後端

前端 (BFF) 模式的後端通常用於微服務環境。在微前端的背景下，BFF 是屬於微型前端的伺服器端服務。並非所有微型前端都需要擁有 BFF。但是，如果您使用的是 BFF，它必須在相同的有界上下文中運行，並且不能跨其他有界上下文共享。

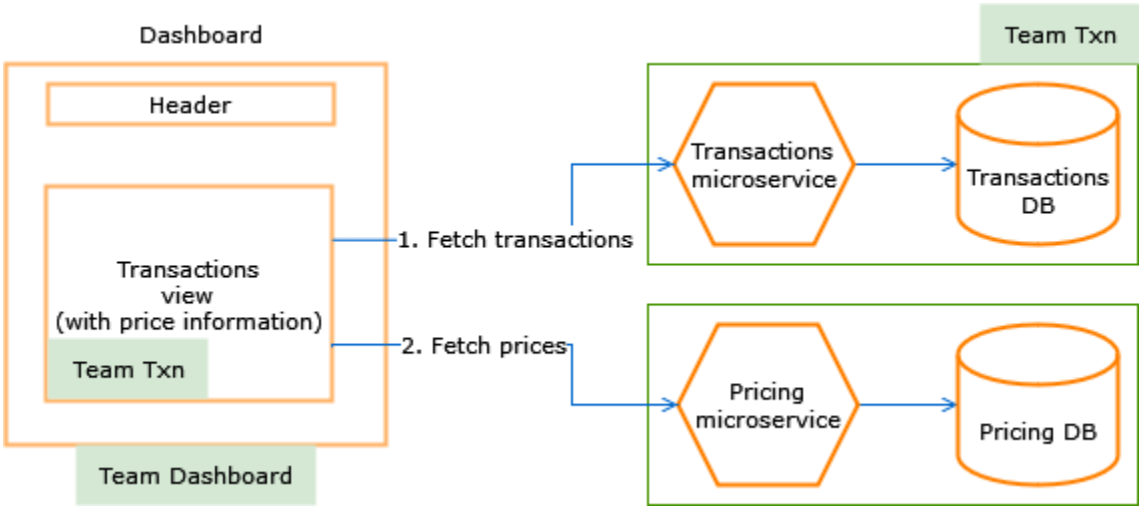
與傳統服務不同，BFF 不遵循域模型。相反，它是一個 API 層，供微前端在數據到達客戶端之前預先處理數據。這很有用的區域包括以下內容：

- 對私有 API 的授權
- 匯總來自不同來源的數據
- 數據轉換以減少網絡負載並減輕客戶端對數據的消耗

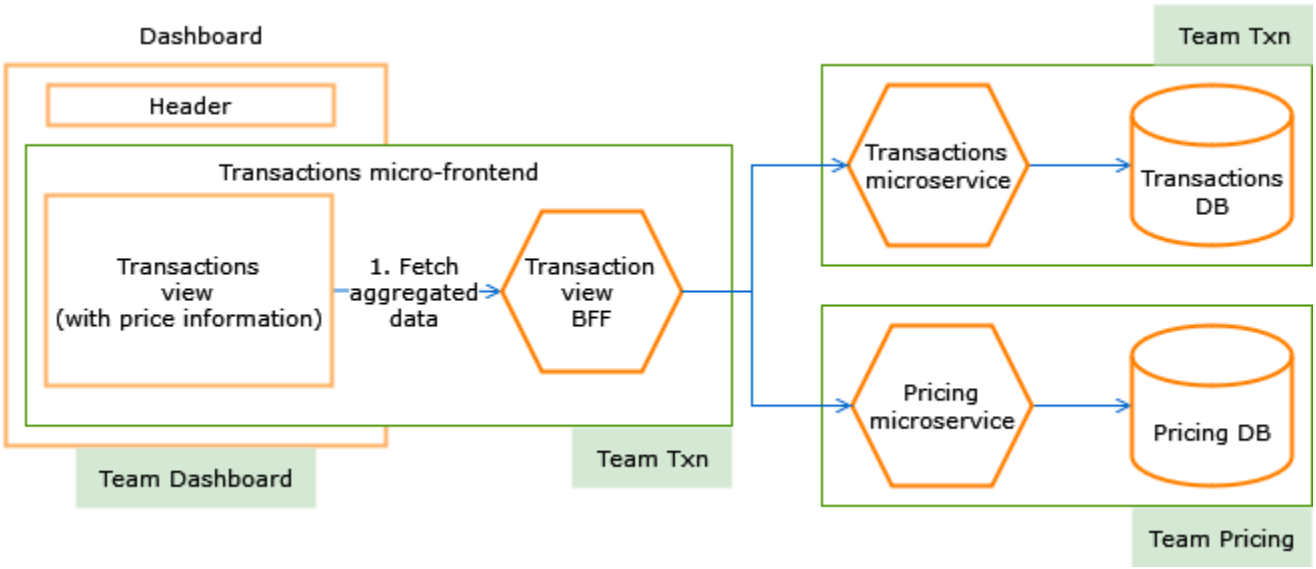
因此，BFF 是由微型前端所擁有，而不是由網域服務層所擁有。BFF 可以通過使用以下方法進行部署：

- AWS AppSync GraphQL API
- 一組 AWS Lambda 函數
- 作為在 Amazon ECS，Amazon EKS 或 AWS 上運行的容器 AppRunner

下圖顯示如果沒有 BFF 模式，微前端必須連線到個別微服務 API 端點，才能擷取和彙總資料。



相反，使用下圖中的 BFF 模式，微前端可以與自己的後端進行通信並獲取匯總數據。



團隊可以為不同渠道（例如移動設備，Web 或特定視圖）開發 BFF，並且需要通過減少聊天性來優化後端互動。

樣式和 CSS

階層式樣式表 (CSS) 是一種用來集中判斷文件呈現方式的語言，而不是文字和物件的硬式編碼格式設定。語言的階層式功能旨在透過使用繼承來控制樣式之間的優先順序。當您使用微型前端並建立管理相依性的策略時，語言的階層式功能可能是一項挑戰。

例如，兩個微前端共存在於同一個頁面上，每個微前端都會定義自己的 HTML 元素樣式。body如果每個文件獲取自己的 CSS 文件，並通過使用style標籤將其附加到 DOM，則 CSS 文件覆蓋到第一個，如果它們都有常見的 HTML 元素，類名或元素 ID 的定義。有不同的策略來處理這些問題，這取決於您為管理樣式所選取的相依性策略而定。

目前，在效能、一致性和開發人員經驗之間取得平衡的最常用方法包括開發和維護設計系統。

設計系統-一種共享的方法

此方法使用系統在適當的時候共用樣式，同時支援偶爾的分歧，以平衡一致性、效能和開發人員體驗。設計系統是由清晰標準指導的可重複使用組件的集合。設計系統開發通常由一個團隊驅動，他們有許多團隊的意見和貢獻。實際上，設計系統是一種共享可以導出為 JavaScript 庫的低級元素的方法。微型前端開發人員可以使用該庫作為依賴項，通過組成預製的可用資源來構建簡單的接口，並作為創建新接口的起點。

考慮需要一個表單的微前端的例子。典型的開發人員經驗包括使用設計系統中可用的預製組件來組成文本框，按鈕，下拉列表和其他 UI 元素。開發人員不需要為實際組件編寫任何樣式，只是為了它們如何一起看。要構建和發布的系統可以使用 webpack 模塊聯合或類似的方法將設計系統聲明為外部依賴項，以便在不包含設計系統的情況下打包表單的邏輯。

然後，多個微型前端可以執行相同的操作來處理共同的擔憂。當團隊開發可在多個微型前端之間共用的新元件時，這些元件會在成熟度之後新增至設計系統。

設計系統方法的一個主要優點是高水平的一致性。雖然微型前端可以編寫樣式並偶爾覆蓋設計系統中的樣式，但是很少需要這樣做。主要的低級元素不會經常改變，並且它們提供了默認情況下可擴展的基本功能。另一個優點是性能。透過建置和發行的良好策略，您可以產生由應用程式殼層檢測的最小共用套裝軟體。當多個微型前端特定的服務包按需非同步加載時，在網絡帶寬方面佔用空間最小，您可以進一步改進。最後但並非最不重要的一點是，開發人員體驗是理想的，因為人們可以專注於構建豐富的界面，而無需重新發明輪子（例如每次需要將按鈕添加到頁面時寫入 JavaScript 和 CSS）。

缺點是任何類型的設計系統都是依賴關係，因此必須維護它並有時更新。如果多個微前端需要新版本的共用相依性，您可以使用下列其中一項：

- 一種協調流程機制，偶爾可以擷取該共用相依性的多個版本而不會發生衝突
- 一種共享策略，將所有從屬項轉移到使用新版本

例如，如果所有微前端都依賴於設計系統的 3.0 版，並且有一個名為 3.1 的新版本以共用方式使用，您可以為所有微型前端實作功能旗標，以最小的風險進行移轉。如需詳細資訊，請參閱 < 功能旗標 > 一節。另一個潛在的缺點是，設計系統通常不僅僅是造型。它們還包括 JavaScript 做法和工具。這些方面需要通過辯論和協作達成共識。

實施設計系統是一項很好的長期投資。這是一種流行的方法，任何在複雜的前端架構上工作的人都應該考慮它。它通常需要前端工程師、產品和設計團隊協同合作並定義彼此互動的機制。安排時間以達到所需狀態非常重要。同樣重要的是要獲得領導力的贊助，以便人們可以長期建立可靠，維護良好和高性能的東西。

完全封裝的 CSS-一個沒有共享的方法

每個微前端都使用慣例和工具來克服 CSS 的級聯功能。一個例子是確保每個元素的樣式始終與類名稱相關聯，而不是元素的 ID，並且類名始終是唯一的。通過這種方式，一切都將範圍限制在單個微前端，並且將不必要衝突的風險降到最低。應用程序 shell 通常負責在將微前端的樣式加載到 DOM 後加載，儘管某些工具通過使用將樣式捆綁在一起。 JavaScript

不分享的主要優點是減少微前端之間引入衝突的風險。另一個優點是開發人員的經驗。每個微型前端與其他微型前端沒有任何內容。單獨發布和測試更簡單快捷。

無共享方法的一個主要缺點是潛在的缺乏一致性。沒有制度來評估一致性。即使複製共享的內容是目標，在平衡發布和協作速度時，它也變得具有挑戰性。常見的緩解措施是創建工具來衡量一致性。例如，您可以建立一個系統，以自動擷取使用無頭瀏覽器在頁面中呈現的多個微型前端的螢幕擷取畫面。然後，您可以在發布之前手動查看屏幕截圖。但是，這需要紀律和治理。如需詳細資訊，請參閱「平衡對齊自主權」一節。

根據用例，另一個潛在的缺點是性能。如果所有微前端都使用了大量樣式，則客戶必須下載大量重複的代碼。這將對用戶體驗產生負面影響。

只有少數團隊的微型前端架構，或者可以容忍低一致性的微型前端架構，才應考慮這種無共享方法。它也可以是一個自然的初始步驟，而一個組織正在設計系統上工作。

共享全球 CSS— 一種共享的方法

使用這種方法，所有與樣式相關的代碼都存儲在一個中央存儲庫中，貢獻者通過處理 CSS 文件或使用預處理器（例如 Sass）為所有微前端編寫 CSS。進行變更時，建置系統會建立單一 CSS 套件，該套

件可以託管在 CDN 中，並由應用程式殼層包含在每個微型前端中。微型前端開發人員可以透過本機代管的應用程式殼層執行程式碼，以設計和建置應用程式。

除了降低微前端之間衝突風險的明顯優勢之外，這種方法的優點是一致性和性能。但是，從標記和邏輯中解耦樣式會使開發人員更難理解樣式的使用方式，它們如何演變以及如何棄用它們。例如，引入新的類別名稱可能會比瞭解現有類別及編輯其屬性的後果更快。建立新類別名稱的缺點是套件大小的成長，這會影響效能，以及可能引入使用者體驗中不一致的情況。

雖然共用的全域 CSS 可以是 monolith-to-micro-frontends移轉的起點，但對於涉及一或兩個以上團隊共同合作的微型前端架構來說，這並不是有益的。我們建議您盡快投資於設計系統，並在設計系統正在開發時實施一個無共用的方法。

組織和工作方式

與所有架構策略一樣，微前端的影響遠遠超出了組織選擇實施的技術。構建微型前端應用程序的決策必須與業務，產品，組織，營運甚至文化保持一致（例如，賦予團隊能力和去中心化決策）。作為回報，這種類型的微前端架構支持真正敏捷的產品驅動開發，因為它顯著減少了其他獨立團隊之間的通信開銷。

敏捷開發

近年來，敏捷軟件開發的想法變得如此普遍，幾乎每個組織都聲稱工作敏捷。雖然敏捷的決定性定義超出了此策略的範圍，但值得回顧一下與微型前端開發相關的關鍵要素。

敏捷範式的基礎是敏捷宣言（2001），它假設了四個主要原則（例如，「過程和工具上的個人和互動」）和十二個原則。流程框架，如 Scrum 和縮放敏捷框架（SAFE）已經出現在敏捷宣言周圍，並找到了他們的方式進入日常實踐。但是，它們背後的哲學在很大程度上被誤解或忽略。

在微型前端架構的背景下，以下敏捷原則對於擁抱很重要：

- 「經常交付工作軟件，從幾週到幾個月，偏好更短的時間尺度。」

  這一原則強調了以增量方式工作，並儘可能定期和盡可能經常地將軟件交付到生產環境中是多麼重要。從技術角度來看，這是指持續集成和持續交付（CI/CD）。在 CI/CD 中，用於構建，測試和部署的工具和過程是每個軟件項目不可或缺的組成部分。priniciple 還意味著運行時基礎結構和操作責任必須由團隊擁有。在獨立子系統對基礎設施和操作的需求可能有明顯不同的分佈式系統中，該所有權尤為重要。
- 「圍繞積極進取的個人構建項目。為他們提供所需的環境和支持，並相信他們能夠完成工作。」

  「最好的架構、需求和設計來自自組織團隊。」

  這兩個原則都強調了所有權，獨立性和 end-to-end 責任的好處。當（且僅當）團隊真正擁有自己的微前端時，微型前端架構才會成功。E 從概念到設計和實施到交付和運營的nd-to-end責任確保團隊實際上可以行使所有權。在技術上和組織上都需要這種獨立性，才能讓團隊對策略方向擁有自主權。我們不建議在使用瀑布式開發模型的集中式組織中使用微型前端平台。

團隊組成和規模

為了讓軟體團隊行使所有權，它必須在組織規定的界限內管理自己，包括團隊如何提供什麼以及如何提供什麼。

為了有效率，團隊必須能夠獨立交付軟體，並有權決定交付軟體的最佳方式。從外部產品經理接收來自外部設計師的功能需求或 UI 設計的團隊，而不參與這些項目的規劃，則無法視為自主。這些功能可能違反現有的合約或功能。這種違規行為將需要進一步的討論和談判，有可能延遲交付並引入團隊之間不必要的衝突。

同時，球隊不應該變得太大。雖然較大的團隊擁有更多資源並且可以因應個人缺勤情況，但是每個新成員的溝通複雜性都會呈指數級增長。無法陳述通用有效的最大群組大小。項目所需的人數取決於諸如團隊成熟度，技術複雜性，創新步伐和基礎設施等因素。例如，Amazon 遵循雙披薩規則：一個太大而無法用兩個比薩餅餵食的團隊應該分成較小的團隊。這可能是一個挑戰。分裂應該沿著自然界限發生，並且應該賦予每個團隊對其工作的自主權和所有權。

DevOps 文化

DevOps 是指從組織和技術角度緊密整合開發生命週期的步驟的軟體工程實務。與普遍的看法相反，DevOps 非常關於文化和思維方式，並且很少關於角色和工具。

傳統上，軟件組織會有專家團隊，例如設計，實施，測試，部署和操作。每當一個團隊完成他們的工作，他們會把項目交給下一個團隊。但是，通過專業團隊的孤島交付軟件會導致切換過程中的摩擦。同時，當專家被迫以狹窄的焦點工作時，他們缺乏鄰近領域的知識，並且他們對產品沒有系統性的看法。這些缺陷可能導致軟件產品的低一致性。

例如，當軟體架構設計人員設計的解決方案要由不同團隊中的某個人實作時，他們可能會忽略實作的固有方面 (例如相依性不相符)。然後，開發人員採取捷徑（例如猴子補丁），或者在架構師和開發團隊之間啟動正式化 back-and-forth 。由於管理這些流程的開銷，開發不再是靈活的（在靈活，適應性，增量和非正式的意義上）。

雖然這個術語 DevOps 主要涉及文化，但它意味著在實踐中成為 DevOps 可能的技術和過程。DevOps 與 CI/CD 密切相關。當開發人員完成軟件的增量實施時，他們將其提交到版本控制系統（例如 Git）。傳統上，建置系統會建置並整合軟體，並以或多或少的統一且集中化的程序來測試和發行。使用 CI/CD，軟體的建置、整合、測試和發行都是固有且自動化的。理想情況下，這個過程是軟件項目本身通過是量身定制給定的項目具體配置文件的一部分。

盡可能多的步驟是自動化的。例如，手動測試實踐應該減少，因為幾乎所有類型的測試都可以自動化。以這種方式設定專案時，每天可以放心地傳送數次軟體產品的更新。支援的另一項技術 DevOps 是基礎架構即程式碼 (IaC)。

傳統上，設置和維護 IT 基礎架構需要手動安裝和維護硬件（在數據中心設置電纜和服務器）和操作軟件。這是必要的，但它有許多缺點。安裝非常耗時且容易出錯。硬體通常過度佈建或佈建不足，導致超

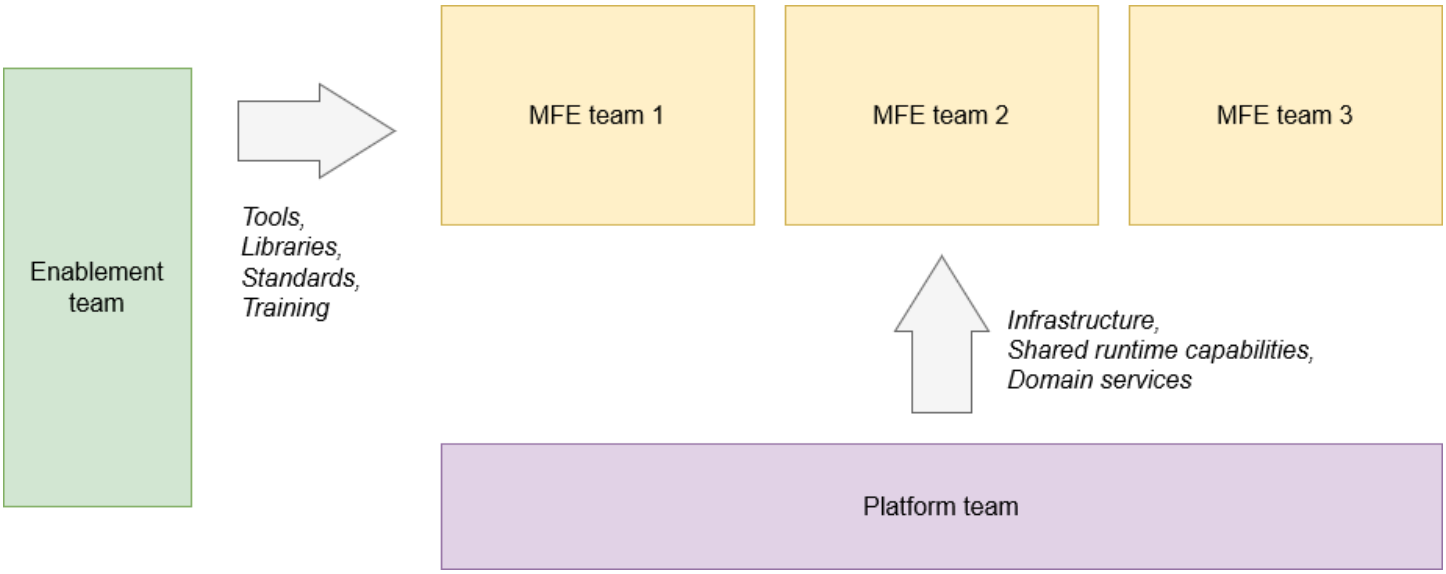
額支出或效能降低。通過使用 IaC，您可以通過配置文件來描述 IT 系統的基礎結構需求，從雲服務可以部署和自動更新。

這一切與微前端有什麼關係？ DevOps、CI/CD 和 IAC 是微型前端架構的理想補充。微前端的優勢依賴於快速且無摩擦的交付流程。只有在團隊 end-to-end 負責擁有軟體專案的環境中， DevOps 文化才能蓬勃發展。

協調跨多個團隊的微型前端開發

在跨多個跨職能團隊擴展微型前端開發時，會出現兩個問題：首先，團隊開始開發自己對範式的解釋，進行框架和庫的選擇，並創建自己的工具和幫助程序庫。其次，完全自主的團隊必須負責執行一般功能，例如低階基礎架構管理。因此，在多團隊微型前端組織中引入另外兩個團隊是有意義的：支持團隊和平台團隊。這些概念在具有分佈式系統的現代 IT 組織中廣泛採用，並在團隊拓撲中有詳細記錄。

下圖顯示支援團隊為三個微型前端團隊提供工具、程式庫、標準和測試。平台團隊為這三個微型前端團隊提供基礎結構、共用執行階段功能和網域服務。



該平台團隊通過使微型前端團隊擺脫無差別的繁重工作來支持他們。這項支援可能包括基礎架構服務，例如容器執行階段、CI/CD 管線、協同作業工具和監控。但是，建立平台團隊不應導致開發與作業分離的組織。相反的情況是：平台團隊提供工程產品，微型前端團隊對其在平台上的服務擁有權和運行時負責。

支持團隊通過專注於治理並確保微型前端團隊的一致性來提供支持。（平台團隊不應該參與此。）啟用團隊會維護共用資源，例如 UI 程式庫，並建立如架構選擇、效能預算和互通性慣例等標準。同時，它為新的團隊或團隊成員提供了如何根據治理定義的應用標準和工具的培訓。

部署

微型前端團隊的自主權北極星是擁有一條獨立於其他微型前端團隊的生產路徑的自動化管道。遵循無共享原則的團隊可以實施獨立的管道。共用程式庫或依賴平台小組的團隊必須決定如何在部署管道中管理相依性。

通常，每條配管都會執行以下操作：

- 構建前端資產
- 將資產部署到託管以供消費
- 確保註冊表和緩存已更新，以便將新版本交付給客戶

實際的管線步驟會根據技術堆疊和頁面構成方法而有所不同。

對於客戶端構成，這意味著將應用程序包上傳到託管存儲桶，並通過 CDN 上的緩存釋放以消耗。搭配服務工作線程使用瀏覽器快取的應用程式也應實作更新服務工作線程快取的方法。

對於伺服器端構成，這通常意味著部署新版本的伺服器元件，並更新微型前端登錄，使新版本可被探索。您可以使用藍/綠或初期測試部署模式來逐步推出新版本。

控管

多個角色通常在微前端上工作，每個角色都在不同的約束下工作，朝著共同的業務目標。雖然人與人之間的溝通和協作是成功的關鍵，但過度溝通和實施過於複雜的流程會減慢開發週期。這會導致士氣下降，並降低品質標準。

通過使用多個團隊實施微前端的最成功的公司創建了機制來平衡自主性和一致性。他們使決策者能夠在本地採取行動，並僅在需要時以階層方式升級。機制包括以下內容：

- API 合約
- 使用事件進行交叉互動
- 平衡自主權與對齊
- 功能旗標
- 服務探索

API 合約

每個微前端都是一個能夠封裝意見，邏輯和複雜性的系統。橫切問題通常包括以下內容：

- 設計系統-工具開發作為庫分佈的用戶界面
- 組合-微前端需要與應用程序 shell 進行交互以渲染和繼承其上下文的方式
- 邏輯處理-與 API 進行交互以處理持久狀態
- 與其他微型前端的交互性-例如發布和消費事件或從一個微型前端導航到另一個情況

為了加速消耗和故障排除，通常投資於標準化這些接口聲明和記錄的方式，包括微型前端依賴關係。由人類策劃的維基是一個很好的開始。更具擴展性的方法是將此信息存儲為代碼中的結構化元數據。然後，您可以使用自動化追蹤歷史變更並提供全文檢索搜尋，將其集中以供使用。

當微型前端涉及大量團隊時，您需要在團隊之間進行協調的策略。以統一的方式共用 API 合約成為必須的，因為它可以減少通訊負荷並改善開發人員體驗。

OpenAPI 是一種用於 HTTP API 的規範語言，它支援以統一的方式定義 API 介面和合約。您可以在 Amazon API Gateway 中使用 OpenAPI 來實作其餘 API。您也可以使用各種可以在容器或虛擬機器中託管的開放原始碼架構。一個顯著的優勢是 OpenAPI 可以自動生成一致格式的文檔，因此多個團隊可以用最少的初始投資共享知識。

當多個團隊在微型前端工作時，他們通常會形成群組。在這些小組中，人們可以在思考並為更大的圖片做出貢獻的同時相遇和互相學習。這些方案通常定義並記錄所有權界限，討論跨領域的關注，並儘早識別解決常見問題的任何重複工作。

使用事件進行交叉互動

在某些情況下，多個微前端可能需要相互交互以對狀態更改或用戶操作做出反應。例如，頁面上的多個微型前端可以包含可收合的選單。當使用者選擇按鈕時，會出現一個功能表。當用戶單擊其他任何地方時，菜單將被隱藏，包括在不同的微前端中呈現的另一個菜單。

從技術上講，諸如 Redux 之類的共享狀態庫可以由多個微前端使用並由 shell 協調。不過，這會在應用程式之間建立明顯的耦合，導致程式碼難以測試，而且在轉譯期間可能會降低效能。

一種常見的有效方法是開發事件匯流排，該匯流排是以程式庫的形式散佈、由應用程式殼層協調，並由多個微型前端使用。通過這種方式，每個微前端異步發布和偵聽特定事件，僅基於其自己的內部狀態其行為。然後，多個團隊可以維護一個共用的 Wiki 頁面，該頁面描述了使用者體驗設計師同意的事件和文件行為。

在事件匯流排範例的實作中，下拉式清單元件會使用共用匯流排來發佈名為的有效負載`drop-down-open-menu`的事件。`{"id": "homepage-aboutus-button"}`該組件添加一個偵聽器的`drop-down-open-menu`事件，以確保如果一個事件觸發一個新的 ID，下拉組件被渲染為隱藏其可折疊部分。通過這種方式，微型前端可以異步響應更改，並提高了性能和更好的封裝，從而使多個團隊更容易設計和測試行為。

我們建議使用由現代瀏覽器原生實施的標準 API，以提高簡單性和可維護性。MDN 事件參考提供與用戶端轉譯應用程式搭配使用事件的相關資訊。

平衡自主權與對齊

微型前端架構強烈偏向於團隊自主權。但是，區分可以支持靈活性的區域和解決問題的多種方法以及需要標準化以實現對齊的區域非常重要。資深領導者和架構師必須儘早識別這些領域，並優先考慮投資，以平衡微前端的安全性、效能、營運卓越性和可靠性。找到這個平衡包括以下內容：微型前端建立、測試、發行和記錄、監視和警示。

創建微前端

理想情況下，所有團隊都保持一致，以便在最終用戶性能方面獲得最大收益 在實踐中，這可能很困難，並且可能需要更多的努力。我們建議從一些書面準則開始，多個團隊可以通過公開和透明的辯論做

出貢獻。然後，團隊可以逐步採用 Cookiecutter 軟件模式，該模式支持創建提供統一方式來支架項目的工具。

使用這種方法，您可以在意見和約束中烘烤。缺點是這些工具在創建和維護方面需要大量投資，並確保在不影響開發人員生產力的情況下快速解決阻止程序。

微前nd-to-end 端的 E 測試

單元測試可以留給業主。我們建議您儘早實施策略，以便在獨特的 shell 上執行交叉測試微前端。該策略包括在生產發行之前和之後測試應用程序的能力。我們建議為技術人員和非技術人員開發流程和文檔，以手動測試關鍵功能。

確保變更不會降低功能性或非功能性客戶體驗非常重要。理想的策略是逐步投資於自動化測試，包括關鍵功能和安全性和效能等架構特性。

釋放微前端

每個團隊可能有自己的方式來部署他們的代碼，烘烤意見和自己的基礎設施。維護此類系統的複雜性成本通常是一種威懾力。相反，我們建議您儘早投資，以實施可以通過共享工具執行的共享策略。

使用選擇的 CI/CD 平台開發範本。然後，團隊可以使用預先核准的範本和共用基礎結構來發行生產環境的變更。您可以儘早開始投資這項開發工作，因為這些系統在最初的測試和整合期之後，很少需要進行重大更新。

日誌記錄和監控

每個團隊可以有不同的業務和系統指標，他們想要跟踪的操作或分析目的。您也可以在這裡套用「曲奇」軟體模式。事件的傳遞可以抽象化並提供為多個微前端可以使用的程式庫。為了平衡彈性並提供自主權，請開發用於記錄自訂指標和建立自訂儀表板或報告的工具。該報告促進了與產品所有者的緊密合作，並減少了最終客戶的反饋循環。

透過標準化傳遞，多個團隊可以共同作業以追蹤指標。例如，電子商務網站可以跟踪從「產品詳細信息」微前端到「購物車」微前端的用戶旅程，再到「購買」微前端來衡量參與度，流失率和問題。如果每個微型前端都使用單一資料庫來記錄事件，您可以整體使用這些資料、全面探索資料，並找出具洞察力的趨勢。

提醒

與記錄和監控類似，標準化的警示功能可提供一定程度的彈性。不同的團隊可能對功能性和非功能性警報的反應不同。但是，如果所有團隊都有整合的方式來根據在共用平台上收集和分析的指標來啟動警

示，則企業可以識別跨團隊問題。此功能在事件管理事件期間非常有用。例如，警示可由下列方式啟動：

- 特定瀏覽器版本的 JavaScript 用戶端例外狀況數量提升
- 在指定臨界值內大幅降低彩現的時間
- 使用特定 API 時，5xx 狀態碼數量增加

根據系統的成熟度，您可以平衡基礎結構不同部分的工作量，如下表所示。

采用	研究與開發	上升	到期日
創建微前端。	實驗, 記錄, 和分享學習.	將工具投資到腳手架新的微型前端。傳福音採用。	鞏固腳手架的工具。推動採用。
測試微前端端到端。	實作手動測試所有相關微前端的機制。	投資自動化安全性和效能測試的工具。調查功能旗標和服務探索。	整合用於服務探索、生產中測試和自動化 end-to-end 測試的工具。
釋放微前端。	投資於共用的 CI/CD 基礎架構和自動化多環境版本。傳福音採用。	整合 CI/CD 基礎架構的工具實作手動復原機制。推動採用。	建立機制以根據系統和業務指標和警示啟動自動回復。
觀察微型前端效能。	投資共用監控基礎架構和程式庫，以持續記錄系統和業務事件。	整合用於監控和警示的工具。實作跨團隊儀表板，以監控一般健康狀況並改善事件管理。	標準化記錄結構描述。優化成本。根據複雜的業務指標實作警示。

功能旗標

功能標誌可以在微前端中實現，以便在多個環境中協調測試和釋放功能。功能旗標技術包括將決策集中在以布爾為基礎的存儲中，以及基於此的驅動行為。它通常用於靜默地傳播可以保持隱藏，直到特定時間點的更改，同時解鎖新版本以用於否則將被阻止的新功能，從而降低團隊速度。

考慮團隊在微型前端功能上工作的示例，該功能將在特定日期啟動。該功能已準備就緒，但需要與獨立發布的另一個微型前端上的更改一起發布。阻止兩個微前端的釋放將被視為一種反模式，並且在部署時會增加風險。

相反地，團隊可以在資料庫中建立 Boolean 功能旗標，這些旗標都會在呈現時間期間使用 (可能是透過 HTTP 呼叫共用功能旗標 API)。小組甚至可以在測試環境中發行變更，其中 Boolean 值設定為，以在啟動`True`到生產環境之前驗證跨專案的功能性和非功能性需求。

功能標誌使用的另一個示例是實現一種機制，通過`QueryString`參數設置特定值或將特定的測試字符串存儲在 cookie 中來覆蓋標誌的值。產品所有者可以迭代功能，而不會阻止其他功能的發布或錯誤修復，直到發布日期。在指定日期，變更資料庫上的旗標值會立即在生產環境中顯示變更，而不需要跨團隊協調發行版本。功能發布後，開發團隊會清理代碼以刪除舊行為。

其他使用案例包括發行以內容為基礎的功能旗標系統。例如，如果單一網站以多種語言為客戶提供服務，則某項功能可能僅適用於特定國家/地區的訪客。功能標誌系統可以依賴於發送國家/地區上下文的消費者（例如，通過使用 `Accept-Language` HTTP 標頭），並且根據該上下文可能會有不同的行為。

雖然功能標誌是促進開發人員和產品擁有者之間協作的強大工具，但他們依靠人們的努力來避免代碼庫顯著降低。在多個功能上保持旗標作用中，可能會增加疑難排解問題、增加 JavaScript 套裝軟體大小並最終累積技術負債時的複雜 常見的緩解措施包括下列項目：

- 單元測試標誌後面的每個功能以減少錯誤的可能性，這可能會在運行測試的自動 CI/CD 管道中引入更長的反饋循環
- 建立工具以在程式碼變更期間測量套裝軟體大小增加，這可在程式碼檢閱期間減少

AWS 提供一系列解決方案，可透過使用 Amazon CloudFront 函數或 Lambda @Edge 來最佳化邊緣上的 A/B 測試。這些方法有助於降低整合解決方案或您用來說明假設的現有 SaaS 產品的複雜性。如需詳細資訊，請參閱 A/B 測試。

服務探索

前端探索模式可改善開發、測試和交付微前端時的開發體驗。該模式使用可共享的配置來描述微前端的進入點。可共用的組態還包括使用初期測試版本，用於每個環境中安全部署的其他中繼資料。

現代的前端開發需要使用各種各樣的工具和庫來支持開發過程中的模塊化。傳統上，此過程包括將代碼捆綁到可以託管在 CDN 中的單個文件中，目的是在運行時保持最低網絡調用，包括初始負載（當應用程序在瀏覽器中打開時）和使用（當客戶執行諸如選擇按鈕或插入信息之類的操作時）。

分割束

微型前端架構解決了由單獨捆綁大量功能組合而生成的非常大的捆綁引起的性能問題。例如，一個非常大的電子商務網站可以捆綁成一個 6 MB 的 JavaScript 文件。儘管壓縮，該文件的大小可能會對用戶在加載應用程序並從邊緣優化的 CDN 下載文件時產生負面影響。

如果您將應用程式分割為首頁、產品詳細資料和購物車微型前端，則可以使用捆綁機制產生三個個別的 2 MB 套裝軟體。當使用者使用首頁時，這項變更可能會將第一次載入的效能提升 300%。只有當使用者造訪某項目的產品頁面並決定購買時，才會以非同步方式載入產品或購物車微前端組合包。

基於這種方法可以使用許多框架和庫，並且對客戶和開發人員都有優勢。若要找出可能導致程式碼中解耦相依性的業務界限，您可以將不同的商務功能對應至多個團隊。分佈式所有權引入了獨立性和敏捷性。

當您分割組建套件時，您可以使用設定來對應微前端，並驅動初始負載和載入後導覽的協調流程。然後，配置可以在運行時使用，而不是在構建期間使用。例如，用戶端前端程式碼或伺服器端後端程式碼可以對 API 進行初始網路呼叫，以動態擷取微前端清單。它也會擷取構成和整合所需的中繼資料。您可以設定容錯移轉策略和快取，以提高可靠性和效能。對應微前端有助於透過先前部署的微前端 (Shell 應用程式協調) 來探索微前端的個別部署。

金絲雀版本

金絲雀版本是部署微服務的完善且流行的模式。Canary 會將發行版目標使用者的儲存貯體釋放到多個群組中，並逐步發行變更，而不是立即取代 (也稱為藍/綠部署)。Canary 發布策略的一個例子是向 10% 的目標用戶推出新的更改，並每分鐘增加 10%，總持續時間為 10 分鐘，以達到 100%。

Canary 版本的目標是及早獲得有關更改的反饋，監控系統以減少任何問題的影響。進行自動化時，可以由可停止部署或啟動復原的內部系統監督商業或系統測量結果。

例如，變更可能會導致一個錯誤，在發行版本的前幾分鐘內會導致收入損失或效能下降。自動監控可以啟動警報。透過服務探索模式，該警示可以停止部署並立即復原，僅影響 20% 的使用者，而不是 100% 的使用者。業務受益於該問題的範圍縮小。

如需使用 DynamoDB 做為儲存以實作 REST 管理 API 的範例架構，請參閱上的 AWS 前端服務探索解決方案。 GitHub使用 AWS CloudFormation 範本將架構整合到您自己的 CI/CD 管線中。該解決方案包括一個 REST 消費者 API，用於將解決方案與您的前端應用程序集成。

您需要平台團隊嗎？

有些公司擁有一個負責擁有和維護程式碼、基礎結構和程序的團隊，這些程序被其他團隊採用來處理微型前端。共同的職責包括：

- 建立並維護可與包含微前端的儲存庫搭配使用的 CI/CD 管線。建置和測試程式碼變更，並在多個環境中發行這些變更。
- 創建和維護觀察性相關的工具，例如共享儀表板，警報機制和系統來對問題做出反應。
- 建立並維護用於事件處理、共用服務消耗和協力廠商相依性的共用程式庫。
- 建立並維護可持續監視非功能性品質的工具，例如系統的效能、安全性和可靠性。
- 創建和維護設計系統。
- 建立、維護及支援微型前端系統的應用程式殼層。

根據專案的規模，您可以使用下列其中一種方法來管理這些職責：

- 創建一個專門的平台團隊，其唯一的責任就是使用共享工具。
- 建立由多個專案團隊的成員組成的群組。小組成員在處理微型前端和使用共用工具之間分割時間。這也被稱為老虎隊。

雖然老虎團隊的方法是保持客戶專注的有效方法，但如果項目獲得牽引力和責任，老虎團隊通常會演變成一個平台團隊。對於平台團隊和老虎團隊來說，最成功的微型前端工作的公司組成了這些團隊，以便具有多種背景和技能的多個人可以做出貢獻。團隊成員可能包括後端工程師，前端工程師，用戶體驗（UX）設計師和技術產品經理。這種多樣性促使人們在考慮到簡單性的情況下不斷參與健康的辯論和設計。

後續步驟

本指南涵蓋架構和組織模式、關鍵決策的權衡，以及與微觀前端相關的治理問題。這些表格總結了本文件中所討論的實務權衡，其範圍如下：

- 自主性-每個微前端團隊獨立發展其實施和發布給最終用戶的能力。
- 一致性-應用程序的整體體驗，其中每個微前端的行為如預期。高一致性意味著微前端與應用程序的其餘部分一致，並且不會損害整體應用程序的用戶體驗。
- 複雜性：實作和測試微前端、整體應用程式和治理控制所需的基礎結構、程式碼和工作量。

實踐	自治	一致性	复杂性
使用微型前端而非整合式應用程式進行建置	高	中	高

代碼共享做法	自治	一致性	复杂性
不分享	高	低	低
分享跨領域的關注	中	高	中
共用商務邏輯	低	高	中
在建置階段透	中	高	低

代碼共享做法	自治	一致性	複杂性
過程式庫共用			
在執行階段共用	高	高	高

微型前端探索實務	自治	一致性	複杂性
在應用程式建置期間	低	高	低
伺服器端探	高	高	中
用戶端 (執行階段) 探索	高	高	中

檢視構成實務	自治	一致性	複杂性
伺服器端合成	高	中	高
邊緣側組成	中	中	高

檢視構成實務	自治	一致性	复杂性
用戶端合成	高	中	中

若要進一步了解本指南中介紹的概念，請參閱「資源」一節。

資源

- 上下文中的微前端
- 領域驅動設計
- EDA 視覺
- 前端探索
- 前端服務探索 AWS
- 敏捷宣言
- MDN 事件參考資料
- OpenAPI

貢獻者

以下人員對本指南做出了貢獻。

- Figus，首席解決方案架構師， AWS
- 亞歷山大·甘斯謝，高級解決方案建築師， AWS
- 哈倫·哈斯達爾，高級解決方案架構師， AWS
- 盧卡 Mezzalira, 主要去市場專家解決方案建築師無服務器英國, AWS

文件歷史記錄

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7月12日

AWS 規範指南詞彙表

以下是 AWS Prescriptive Guidance 所提供策略、指南和模式的常用術語。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：將您的內部部署 Oracle 資料庫遷移至 Amazon Aurora Postgre SQL-Compatible Edition。
- 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：將您的內部部署 Oracle 資料庫遷移至 中的 Oracle 的 Amazon Relational Database Service （Amazon RDS） 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：將內部部署 Oracle 資料庫遷移至 中EC2執行個體上的 Oracle 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至相同平台的雲端服務。範例：遷移 Microsoft Hyper-V 應用程式至 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

請參閱屬性型存取控制 。

抽象服務

請參閱 受管服務 。

ACID

請參閱原子、一致性、隔離、耐久性 。

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。它更靈活，但比主動被動遷移需要更多工作。

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。

彙總函數

在一組資料列上操作並計算該群組單一傳回值的SQL函數。彙總函數的範例包括 SUM和 MAX。

AI

請參閱人工智慧。

AIOps

請參閱人工智慧操作 。

匿名化

在資料集中永久刪除個人資訊的程序。匿名化有助於保護個人隱私權。匿名資料不再被視為個人資料。

反模式

經常性問題的常用解決方案，其解決方案具有反效益、無效或效果不如替代方案。

應用程式控制

一種安全方法，僅允許使用核准的應用程式，以協助保護系統免受惡意軟體侵害。

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。如需如何在遷移策略AIOps中使用 AWS 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、一致性、隔離、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ABAC AWS中的 。

權威性資料來源

存放主要版本資料的位置，被視為最可靠的資訊來源。您可以將資料從授權資料來源複製到其他位置，以處理或修改資料，例如匿名、編輯或假名化資料。

可用區域

中與其他可用區域中的故障 AWS 區域 隔離的不同位置，並對相同區域中的其他可用區域提供低成本、低延遲的網路連線。

AWS 雲端採用架構（AWS CAF）

的指導方針和最佳實務架構 AWS ，可協助組織開發高效且有效的計劃，以成功地遷移至雲端。AWS CAF 將指導方針整理成六個重點領域：業務、人員、治理、平台、安全和操作。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。為此， AWS CAF 提供人員開發、訓練和通訊的指引，協助組織成功採用雲端。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）

評估資料庫遷移工作負載、建議遷移策略並提供工作估算的工具。 AWS WQF 隨附於 AWS Schema Conversion Tool（AWS SCT）。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

錯誤的機器人

旨在中斷或傷害個人或組織的機器人。

BCP

請參閱業務持續性規劃 。

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。您可以搭配 Amazon Detective 使用行為圖表來檢查失敗的登入嘗試、可疑API的呼叫和類似的動作。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。另請參閱永久性 。

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

一種部署策略，您可以在其中建立兩個不同但相同的環境。您可以在一個環境（藍色）中執行目前的應用程式版本，並在另一個環境（綠色）中執行新的應用程式版本。此策略可協助您在影響最小的情況下快速復原。

機器人

透過網際網路執行自動化任務並模擬人類活動或互動的軟體應用程式。某些機器人很有用或有益，例如在網際網路上為資訊編製索引的 Web 爬蟲程式。某些其他稱為壞機器人的機器人，旨在中斷或傷害個人或組織。

殭屍網路

受到惡意軟體感染且由單一方控制的機器人網路，稱為機器人繼承器或機器人運算子。殭屍網路是擴展機器人及其影響的最佳已知機制。

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）。

碎片存取

在特殊情況下，以及透過核准的程序，使用者取得其通常無權存取 AWS 帳戶 之 存取權的快速方法。如需詳細資訊，請參閱 Well-Architected 指南中的 AWS 實作碎片程序指標。

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

CAF

請參閱AWS 雲端採用架構。

Canary 部署

版本向最終使用者緩慢且增量的版本。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 Cloud Center of Excellence 。

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。您可以使用 CDC進行各種用途，例如稽核或複寫目標系統中的變更，以維持同步。

混亂工程

故意引入故障或破壞性事件，以測試系統的復原能力。您可以使用 AWS Fault Injection Service（AWS FIS）來執行實驗，以強調 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。

Cloud Center of Excellence（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。雲端運算通常連接到邊緣運算技術。

雲端操作模型

在 IT 組織中，用於建置、成熟和最佳化一或多個雲端環境的操作模型。如需詳細資訊，請參閱建置您的雲端操作模型。

採用雲端階段

組織在遷移至 時通常會經歷的四個階段 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您的雲端採用 （例如，建立登陸區域、定義 CCoE、建立操作模型）
- 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

這些階段是由 Stephen Orban 在企業 AWS 雲端 策略部落格上的 The Journey Toward Cloud-First 和採用階段部落格中定義的。如需有關它們如何與 AWS 遷移策略關聯的資訊，請參閱遷移準備指南 。

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。常見的雲端儲存庫包括 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

很少存取且通常為歷史資料的資料。查詢這類資料時，通常可接受慢查詢。將此資料移至效能較低且成本較低的儲存層或類別，可以降低成本。

電腦視覺 （CV）

使用機器學習從數位映像和影片等視覺化格式分析和擷取資訊的 AI 欄位。例如， AWS Panorama 提供將 CV 新增至內部部署攝影機網路的裝置，而 Amazon 則 SageMaker 提供 CV 的影像處理演算法。

組態偏離

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

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。您通常會在遷移CMDB的產品組合探索和分析階段使用來自 的資料。

一致性套件

您可以組合的 AWS Config 規則和修復動作集合，以自訂合規和安全檢查。您可以使用 YAML 範本，將一致性套件部署為 AWS 帳戶 和 區域中或整個組織中的單一實體。如需詳細資訊，請參閱 AWS Config 文件中的一致性套件。

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

請參閱電腦視覺效果 。

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 Framework 中安全支柱的元件。如需詳細資訊，請參閱資料分類。

資料漂移

生產資料與用於訓練 ML 模型的資料之間有意義的變化，或輸入資料隨時間有意義的變化。資料偏離可以降低 ML 模型預測的整體品質、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

架構架構架構，提供分散式、分散式資料擁有權，並具有集中式管理與治理。

資料最小化

僅收集和處理嚴格必要資料的原則。在 中實作資料最小化 AWS 雲端 可以降低隱私權風險、成本和分析碳足跡。

資料周邊

AWS 環境中的一組預防性防護機制，可協助確保只有信任的身分才能從預期的網路存取信任的資源。如需詳細資訊，請參閱在 上建立資料周邊 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。

資料來源

追蹤資料整個生命週期的原始伺服器和歷史記錄的程序，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

支援商業智慧的資料管理系統，例如分析。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。

資料庫操作語言 （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。當您在 上採用此策略時 AWS，您可以在 AWS

Organizations 結構的不同層新增多個控制項，以協助保護資源。例如， defense-in-depth 方法可能會結合多重要素驗證、網路分割和加密。

委派的管理員

在 中 AWS Organizations，相容的服務可以註冊 AWS 成員帳戶，以管理組織的帳戶並管理該服務的許可。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。

開發值串流映射 （DVSM）

用於識別和排序限制的程序，這些限制會對軟體開發生命週期中的速度和品質產生不利影響。DVSM 延伸了最初為精實生產實務設計的價值串流映射程序。它專注於透過軟體開發程序建立和移動價值所需的步驟和團隊。

數位分身

真實世界系統的虛擬表示法，例如建築物、工廠、工業設備或生產線。數位分身支援預測性維護、遠端監控和生產最佳化。

維度資料表

在星狀結構描述 中，較小的資料表包含有關事實資料表中量化資料的資料屬性。維度資料表屬性通常是文字欄位或離散數字，其行為與文字類似。這些屬性通常用於查詢限制、篩選和結果集標籤。

災難

阻止工作負載或系統在其主要部署位置實現其業務目標的事件。這些事件可能是自然災難、技術故障或人為動作的結果，例如意外錯誤組態或惡意軟體攻擊。

災難復原 （DR）

您用來將災難造成的停機時間和資料遺失降至最低的策略和程序。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的雲端中的工作負載災難復原 AWS：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。如需有關如何搭配 strangler fig 模式使用網域驅動設計的資訊，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。

DR

請參閱災難復原。

漂移偵測

追蹤與基準組態的偏差。例如，您可以使用 AWS CloudFormation 來偵測系統資源 中的漂移，或者您可以使用 AWS Control Tower 來偵測您的登陸區域中可能會影響對治理要求合規性的變更。https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-stack-drift.html

DVSM

請參閱開發值串流映射 。

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。與雲端運算 相比，邊緣運算可以減少通訊延遲並縮短回應時間。

加密

將純文字資料轉換為可人工讀取的運算程序。

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 建立端點服務， AWS PrivateLink 並將許可授予其他 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端點，私下連線至您的VPC端點服務。如需詳細資訊，請參閱 Amazon Virtual Private Cloud （AmazonVPC） 文件中的建立端點服務。

企業資源規劃 （ERP）

可自動化和管理企業關鍵業務流程 （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安全特徵包括身分和存取管理、偵測控制、基礎設施安全、資料保護和事件回應。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。EDA 透過計算摘要統計資料和建立資料視覺化來執行。

F

事實資料表

星狀結構描述 中的中央資料表。它存放有關業務操作的量化資料。一般而言，事實資料表包含兩種類型的資料欄：包含量值的資料，以及包含維度資料表外部索引鍵的資料。

快速失敗

使用頻繁且增量測試來縮短開發生命週期的哲學。這是敏捷方法的關鍵部分。

故障隔離界限

在 中 AWS 雲端，邊界，例如可用區域 AWS 區域、控制平面或資料平面，這些邊界會限制故障的影響，並有助於改善工作負載的復原能力。如需詳細資訊，請參閱AWS 故障隔離界限 。

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。這通常表示為數值分數，可透過各種技術計算，例如 Shapley 累加解釋（SHAP）和整合式漸層。如需詳細資訊，請參閱使用 的機器學習模型可解譯性：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

請參閱精細存取控制 。

精細存取控制（FGAC）

使用多個條件來允許或拒絕存取請求。

閃切遷移

一種資料庫遷移方法，透過變更資料擷取使用連續資料複寫，以盡可能在最短時間內遷移資料，而不是使用分階段方法。目標是將停機時間降至最低。

G

地理封鎖

請參閱地理限制 。

地理限制 (地理封鎖)

在 Amazon 中 CloudFront，此選項可防止特定國家/地區的使用者存取內容分發。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 工作流程被視為舊版，而以中繼線為基礎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

高階規則，可協助管理組織單位 （ ） 之間的資源、政策和合規性OUs。預防性防護機制會強制執行政策，以確保符合合規標準。它們是透過使用服務控制政策和IAM許可界限來實作。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。其實作方式是使用 AWS Config、 AWS Security Hub、Amazon GuardDuty、 AWS Trusted Advisor、Amazon Inspector 和自訂 AWS Lambda 檢查。

H

HA

請參閱高可用性 。

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，無需介入。HA 系統設計為自動容錯移轉、持續提供高品質效能，以及處理不同的負載和故障，並將效能影響降至最低。

歷史現代化

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。歷史記錄器是一種資料庫，用於從工廠的不同來源收集和存放資料。

異質資料庫遷移

將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。

常用資料

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，修正程式通常在典型 DevOps 的發行工作流程之外建立。

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

將基礎設施視為程式碼 。

身分型政策

連接至一或多個IAM主體的政策，其定義其在 AWS 雲端 環境中的許可。

閒置應用程式

在 90 天內，平均CPU和記憶體用量介於 5% 到 20% 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。

不可變的基礎設施

為生產工作負載部署新基礎設施的模型，而不是更新、修補或修改現有基礎設施。與可變基礎設施 相比，不可避免的基礎設施本質上更一致、可靠且可預測。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的使用不可變基礎設施部署最佳實務。

傳入（傳入）VPC

在 AWS 多帳戶架構中， VPC接受、檢查和路由來自應用程式外部的網路連線。AWS Security Reference Architecture 建議設定具有傳入、傳出和檢查的網路帳戶VPCs，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

由 Klaus Schwab 於 2016 年推出的術語，指透過連線能力、即時資料、自動化、分析和 AI/ML 的進步來現代化製造程序。

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，請參閱建置工業物聯網（IIoT）數位轉型策略。

檢查 VPC

在 AWS 多帳戶架構中，集中VPC管理 VPCs（在相同或不同的 中 AWS 區域）、網際網路和內部部署網路之間的網路流量檢查。AWS Security Reference Architecture 建議設定具有傳入、傳出和檢查的網路帳戶VPCs，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，請參閱使用 的機器學習模型可解譯性AWS。

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 提供 的基礎ITSM。

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。如需整合雲端操作與ITSM工具的相關資訊，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

標籤型存取控制（LBAC）

強制存取控制（MAC）的實作，其中使用者和資料本身都會被明確指派安全標籤值。使用者安全標籤與資料安全標籤之間的交集決定使用者可以看到哪些資料列和資料欄。

登陸區域

登陸區域是架構良好的多帳戶 AWS 環境，可擴展且安全。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

請參閱標籤型存取控制 。

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

請參閱 7 Rs 。

小端序系統

首先儲存最低有效位元組的系統。另請參閱端點 。

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

旨在危及電腦安全或隱私權的軟體。惡意軟體可能會中斷電腦系統、洩露敏感資訊或取得未經授權的存取。惡意軟體的範例包括病毒、蠕蟲、勒索軟體、特洛伊木馬程式、間諜軟體和鍵盤記錄器。

受管服務

AWS 服務 可 AWS 操作基礎設施層、作業系統和平台，而且您可以存取端點來存放和擷取資料。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 （ MES ）

用於追蹤、監控、記錄和控制生產程序的軟體系統，可將原物料轉換為工廠的成品。

MAP

請參閱遷移加速計劃 。

機制

建立工具、推動工具採用，然後檢查結果以進行調整的完整程序。機制是在運作時強化和改善自身的循環。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的建置機制。

成員帳戶

除了屬於 中組織一部分的管理帳戶 AWS 帳戶 之外的所有 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 。

訊息佇列遙測傳輸 （ MQTT ）

根據發佈/訂閱模式的輕量型 machine-to-machine （ M2M ） 通訊協定，適用於資源受限的 IoT 裝置。

微服務

小型、獨立的服務，透過定義明確的方式進行通訊，APIs通常由小型、獨立的團隊擁有。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，請參閱使用無 AWS 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 ，透過定義明確的介面進行通訊APIs。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，請參閱在 上實作微服務 AWS。

Migration Acceleration Program （MAP）

提供諮詢支援、訓練和服務，以協助組織建立強大的操作基礎以遷移至雲端，並協助抵銷遷移初始成本的 AWS 計畫。MAP 包含以有系統方式執行舊版遷移的遷移方法，以及一組可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。遷移工廠團隊通常包括操作、業務分析師和擁有者、遷移工程師、開發人員和在衝刺中工作的 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。遷移中繼資料的範例包括目標子網路、安全群組和 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：EC2使用 AWS Application Migration Service 重新託管遷移至 Amazon。

遷移產品組合評估 （MPA）

線上工具，提供驗證商業案例以遷移至 的資訊 AWS 雲端。MPA 提供詳細的產品組合評估 （伺服器大小調整、定價、TCO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、遷移優先順序和波規劃）。MPA 工具 （需要登入） 可供所有 AWS 顧問和APN合作夥伴顧問免費使用。

遷移就緒狀態評估 （MRA）

使用 取得組織雲端整備狀態的洞見、識別優缺點，以及建立行動計劃以消除已識別差距的程序 AWS CAF。如需詳細資訊，請參閱遷移準備程度指南。MRA 是AWS 遷移策略 的第一個階段。

遷移策略

用於將工作負載遷移至 的方法 AWS 雲端。如需詳細資訊，請參閱本詞彙表中的 7 個 Rs 項目，並請參閱將組織動員以加速大規模遷移。

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 中的應用程式現代化策略 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 中的評估應用程式的現代化準備 AWS 雲端程度。

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

請參閱遷移產品組合評估 。

MQTT

請參閱訊息佇列遙測傳輸 。

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？」

可變基礎設施

更新和修改生產工作負載現有基礎設施的模型。為了提高一致性、可靠性和可預測性， AWS Well-Architected Framework 建議使用不可變的基礎設施作為最佳實務。

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 操作整合。

OLA

請參閱操作層級協議。

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。

OPC-UA

請參閱開放程序通訊 - Unified Architecture。

開放程序通訊 - Unified Architecture（OPC-UA）

工業自動化的 machine-to-machine（M2M）通訊協定。OPC-UA 提供資料加密、身分驗證和授權方案的互通性標準。

操作層級協議（OLA）

闡明哪些功能性 IT 群組承諾相互交付的協議，以支援服務層級協議（SLA）。

操作預備檢閱（ORR）

問題及相關最佳實務的檢查清單，可協助您了解、評估、預防或減少事件和可能失敗的範圍。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的操作就緒審核（ORR）。

操作技術（OT）

使用實體環境控制工業操作、設備和基礎設施的硬體和軟體系統。在製造中，整合 OT 和資訊技術（IT）系統是 Industry 4.0 轉型的關鍵重點。

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 會記錄 AWS 帳戶 中組織中所有 的事件 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 文件中的 CloudTrail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 透過加速變革採用、解決轉型問題，以及推動文化和組織變革，協助組織準備和轉換新系統和策略。在 AWS 遷移策略中，由於雲端採用專案所需的變更速度，此架構稱為人員加速 。如需詳細資訊，請參閱 OCM指南。

原始存取控制（OAC）

在 中 CloudFront，用於限制存取以保護您的 Amazon Simple Storage Service（Amazon S3）內容的增強型選項。OAC 支援所有 中的所有 S3 儲存貯體 AWS 區域，伺服器端加密搭配 AWS KMS（SSE-KMS），以及對 S3 儲存貯體的動態PUT和DELETE請求。

原始存取身分（OAI）

在 中 CloudFront，此選項用於限制存取，以保護您的 Amazon S3 內容。當您使用 時OAI， 會 CloudFront 建立 Amazon S3 可以驗證的主體。已驗證的主體只能透過特定 CloudFront 分發存取 S3 儲存貯體中的內容。另請參閱 OAC，它提供更精細和增強的存取控制。

ORR

請參閱操作預備檢閱 。

OT

請參閱操作技術 。

傳出（輸出）VPC

在 AWS 多帳戶架構中，VPC處理從應用程式內啟動之網路連線的 。AWS Security Reference Architecture 建議設定具有傳入、傳出和檢查的網路帳戶VPCs，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

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

個人身分資訊（PII）

直接檢視或與其他相關資料配對時，可用來合理推斷個人身分的資訊。的範例PII包括名稱、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。

PLC

請參閱可程式邏輯控制器 。

PLM

請參閱產品生命週期管理。

政策

可以定義許可（請參閱身分型政策 ）、指定存取條件（請參閱資源型政策 ）或定義組織中所有帳戶的最大許可 AWS Organizations（請參閱服務控制政策 ）的物件。

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`WHERE`子句中。

述詞下推

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

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

設計隱私

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

容器，其中包含您希望 Amazon Route 53 如何回應一個或多個 內網域及其子網域的DNS查詢的資訊VPCs。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

旨在防止部署不合規資源的安全控制。這些控制項會在佈建資源之前對其進行掃描。如果資源不符合控制項，則不會佈建。如需詳細資訊，請參閱 AWS Control Tower 文件中的控制項參考指南，並請參閱在 上實作安全控制項中的主動控制項。 AWS

產品生命週期管理 （PLM）

從設計、開發和啟動，到成長和成熟，再到拒絕和移除，產品整個生命週期的資料和程序管理。

生產環境

請參閱環境 。

可程式設計邏輯控制器（PLC）

在製造中，高度可靠、可調整的電腦，可監控機器並自動化製造程序。

擬匿名化

將資料集中的個人識別碼取代為預留位置值的程序。假名化有助於保護個人隱私權。假名化資料仍被視為個人資料。

發佈/訂閱（pub/sub）

一種模式，可在微服務之間啟用非同步通訊，以提高可擴展性和回應能力。例如，在微服務型中MES，微服務可以將事件訊息發佈到其他微服務可以訂閱的頻道。系統可以新增新的微服務，而無需變更發佈服務。

Q

查詢計劃

一系列步驟，如指示，用於存取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

RACI 矩陣

請參閱負責、負責、已諮詢、知情（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。

RASCI 矩陣

請參閱負責、負責、已諮詢、知情（RACI）。

RCAC

請參閱資料列和資料欄存取控制。

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。

重新架構師

請參閱 7 Rs 。

復原點目標 （RPO）

自上次資料復原點以來可接受的時間上限。這會決定最後一個復原點與服務中斷之間可接受的資料遺失。

復原時間目標 （RTO）

服務中斷與服務還原之間的可接受延遲上限。

重構

請參閱 7 Rs 。

區域

地理區域 AWS 的資源集合。每個 AWS 區域 都獨立於其他 ，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 哪些帳戶可以使用 。

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。

重新託管

請參閱 7 Rs 。

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

請參閱 7 Rs 。

轉譯形式

請參閱 7 Rs 。

回購

請參閱 7 Rs 。

彈性

應用程式抵禦中斷或從中斷中復原的能力。在 中規劃恢復能力時，高可用性和災難復原是常見的考量事項 AWS 雲端。如需詳細資訊，請參閱AWS 雲端 復原能力 。

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。

負責任、負責、已諮詢、知情（RACI）矩陣

定義所有涉及遷移活動和雲端操作之各方的角色和責任的矩陣。矩陣名稱衍生自矩陣中定義的責任類型：責任（R）、責任（A）、已諮詢（C）和知情（I）。支援（S）類型為選用。如果您包含支援，則矩陣稱為RASCI矩陣 ，如果您排除它，則稱為RACI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

請參閱 7 Rs 。

淘汰

請參閱 7 Rs 。

輪換

定期更新秘密的程序，讓攻擊者更難存取憑證。

資料列和資料欄存取控制（RCAC）

使用已定義存取規則的基本靈活SQL表達式。RCAC 包含資料列許可和資料欄遮罩。

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

許多身分提供者（IdPs）使用的開放標準。此功能會啟用聯合單一登入（SSO），因此使用者可以登入 AWS Management Console 或呼叫操作， AWS API而不必IAM為您組織中的每個人建立使用者。如需 SAML 2.0 型聯合的詳細資訊，請參閱IAM文件中關於 SAML 2.0 型聯合。

SCADA

請參閱 監控控制和資料擷取 。

SCP

請參閱服務控制政策 。

秘密

在 中 AWS Secrets Manager，以加密形式存放的機密或限制資訊，例如密碼或使用者憑證。它由秘密值及其中繼資料組成。秘密值可以是二進位、單一字串或多個字串。如需詳細資訊，請參閱 Secrets Manager 文件中的 Secrets Manager 秘密中的內容？。

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。安全控制有四種主要類型：預防性 、偵測性 、回應性 和主動 。

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。

安全資訊和事件管理（SIEM）系統

結合安全資訊管理（SIM）和安全事件管理（SEM）系統的工具和服務。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，並產生警示。

安全回應自動化

預先定義和程式設計的動作，旨在自動回應或修復安全事件。這些自動化可做為偵測或回應式安全控制，協助您實作 AWS 安全最佳實務。自動化回應動作的範例包括修改VPC安全群組、修補Amazon EC2執行個體或輪換憑證。

伺服器端加密

由接收資料的 AWS 服務 加密其目的地的資料。

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。SCPs 定義管理員可委派給使用者或角色之動作的防護機制或設定限制。您可以使用 SCPs作為允許清單或拒絕清單，以指定允許或禁止的服務或動作。如需詳細資訊，請參閱 AWS Organizations 文件中的服務控制政策。

服務端點

URL 的 進入點的 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。

服務層級協議（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。

服務層級指標（SLI）

服務效能方面的測量，例如其錯誤率、可用性或輸送量。

服務層級目標（SLO）

代表服務運作狀態的目標指標，由服務層級指標 測量。

共同責任模式

描述您共享 AWS 的雲端安全與合規責任的模型。 AWS 負責雲端的安全，而您要負責雲端的安全。如需詳細資訊，請參閱共同責任模式。

SIEM

請參閱安全資訊和事件管理系統 。

單一失敗點（SPOF）

應用程式的單一關鍵元件故障，可能會中斷系統。

SLA

請參閱服務層級協議 。

SLI

請參閱服務層級指示器 。

SLO

請參閱服務層級目標 。

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 中的階段式應用程式現代化方法 AWS 雲端。

SPOF

請參閱單一失敗點 。

星狀結構描述

使用一個大型事實資料表來存放交易或測量資料的資料庫組織結構，並使用一或多個較小的維度資料表來存放資料屬性。此結構專為資料倉儲或商業智慧用途而設計。

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

中 IP 地址的範圍VPC。子網必須位於單一可用區域。

監控控制和資料擷取 （SCADA）

在製造中，使用硬體和軟體來監控實體資產和生產操作的系統。

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

以模擬使用者互動的方式測試系統，以偵測潛在問題或監控效能。您可以使用 Amazon CloudWatch Synthetics 來建立這些測試。

T

標籤

作為中繼資料的鍵值對，用於組織您的 AWS 資源。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

可用來互連 VPCs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 文件中的什麼是傳輸閘道。

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

將許可授予您指定的服務，以代表您在組織中執行任務 AWS Organizations ，並在其帳戶中執行任務。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 文件中的 AWS Organizations AWS Organizations 搭配使用其他 AWS 服務。

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

一個小型 DevOps 團隊，您可以使用兩個披薩來餵送。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。

未區分的任務

也稱為繁重型 ，是建立和操作應用程式的必要工作，但不為最終使用者提供直接價值或提供競爭優勢。未區分任務的範例包括採購、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

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

漏洞

損害系統安全性的軟體或硬體缺陷。

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。

暖資料

不常存取的資料。查詢這類資料時，通常可接受中等慢的查詢。

視窗函數

對以某種方式與目前記錄相關聯之資料列群組執行計算的SQL函數。視窗函數適用於處理任務，例如根據目前資料列的相對位置計算移動平均值或存取資料列的值。

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。

WORM

請參閱寫入一次，讀取許多。

WQF

請參閱AWS工作負載資格架構。

寫入一次，讀取許多（WORM）

一次性寫入資料的儲存模型，可防止刪除或修改資料。授權使用者可以視需要多次讀取資料，但無法變更資料。此資料儲存基礎設施被視為不可變的。

Z

零時差漏洞

利用零時差漏洞的攻擊，通常是惡意軟體。

零時差漏洞

生產系統中未緩解的缺陷或漏洞。威脅發動者可以使用這種類型的漏洞來攻擊系統。開發人員經常因為攻擊而意識到漏洞。

殭屍應用程式

平均CPU和記憶體用量低於 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。