將整體分解為微服務

AWS 方案指引

AWS 方案指引: 將整體分解為微服務

Table of Contents

將整體分解為微服務

Tabby Ward 和 Dmitry Gulin，Amazon Web Services (AWS)

2023 年 4 月 (文件歷史記錄)

遷移至 Amazon Web Services (AWS) 雲端有許多優點，包括技術和業務敏捷性、新的營收機會，以及降低成本。為了充分利用這些優勢，您應該透過將整體應用程式重構為微服務，持續現代化組織的軟體。此程序包含三個主要步驟：

- 將整體分解為微服務 – 使用本指南提供的分解模式，將整體應用程式分解為微服務。
- 整合微服務 – 使用AWS 無伺服器服務，將新建立的微服務整合到微服務架構中。
- 啟用微服務的資料持久性 – 透過分散資料存放區來提升微服務之間的多槽持久性。

現代化通常涉及兩種類型的專案：

- 布朗菲爾德專案涉及在現有或舊版系統的環境中開發和部署新的軟體系統。
- Greenfield 專案涉及從頭開始為全新的環境建立系統，而不涉及任何舊版程式碼。

對於布朗欄位專案，應用程式現代化旅程中的第一個步驟之一是將產品組合中的整體分解為微服務。

大多數應用程式一開始都是針對特定商業使用案例所設計的單體。如果單體的架構未強制執行模組化設計，則單體對於在成熟網域知識範圍內沒有明確定義責任的應用程式，仍然是有效的選擇。整體做為單一部署單位的中心特性也有助於減輕設計瑕疵，例如緊密耦合或缺乏內部結構。

雖然單體是某些使用案例的有效選項，但通常不適合現代應用程式。整體定義不佳的內部結構可能會使維護程式碼變得困難，這會為新開發人員建立陡峭的學習曲線，並導致額外的支援成本。高耦合和低黏著性可能會大幅增加新增新功能所需的時間，而且您可能無法根據流量模式擴展個別元件。Monoliths 也需要多個團隊協調一個大型版本，這會增加協同合作和知識轉移負擔。最後，您可以發現，當您的企業或使用者群成長時，新增功能或建置新的使用者體驗會變得困難。

若要避免這種情況，您可以使用分解模式來分解單體應用程式、將其轉換為數個微服務，並將它們遷移至微服務架構。微服務架構會將應用程式建構為一系列鬆耦合的服務。微服務旨在透過啟用持續交付和持續部署 (CI/CD) 程序來加速軟體開發。

開始分解程序之前，您應該評估要分解的單體。請務必包含具有可靠性或效能問題的單體，或在緊密耦合的架構中包含多個元件。我們也建議您完全了解整體、其技術及其與其他應用程式的相互依存性的商業使用案例。

本指南適用於應用程式擁有者、企業擁有者、架構師、技術主管和專案經理。它討論了以下六個用於分解單體的雲端原生模式，並描述了每個單體的優點和缺點：

- 依業務能力分解
- 依子網域分解
- 依交易分解
- 每個團隊模式的服務
- Strangler 無花果模式
- 依抽象模式分支

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

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

目標業務成果

將整體分解為微服務後，您應該會預期下列結果：

- 將單體應用程式有效率地轉換為微服務架構。
- 快速調整以因應業務需求波動，而不會中斷核心活動，例如高可擴展性、改善彈性、持續交付和故障隔離。
- 加快創新速度，因為每個微服務都可以個別測試和部署。

分解單體的模式

在您決定在應用程式產品組合中分解整體之後，您應該選擇適當的分解模式，並將其引入您的組織。

> **Note**
>
> 您可以使用多種模式來分解整體。例如，您可以依業務能力分解整體，然後使用子網域模式來進一步細分。

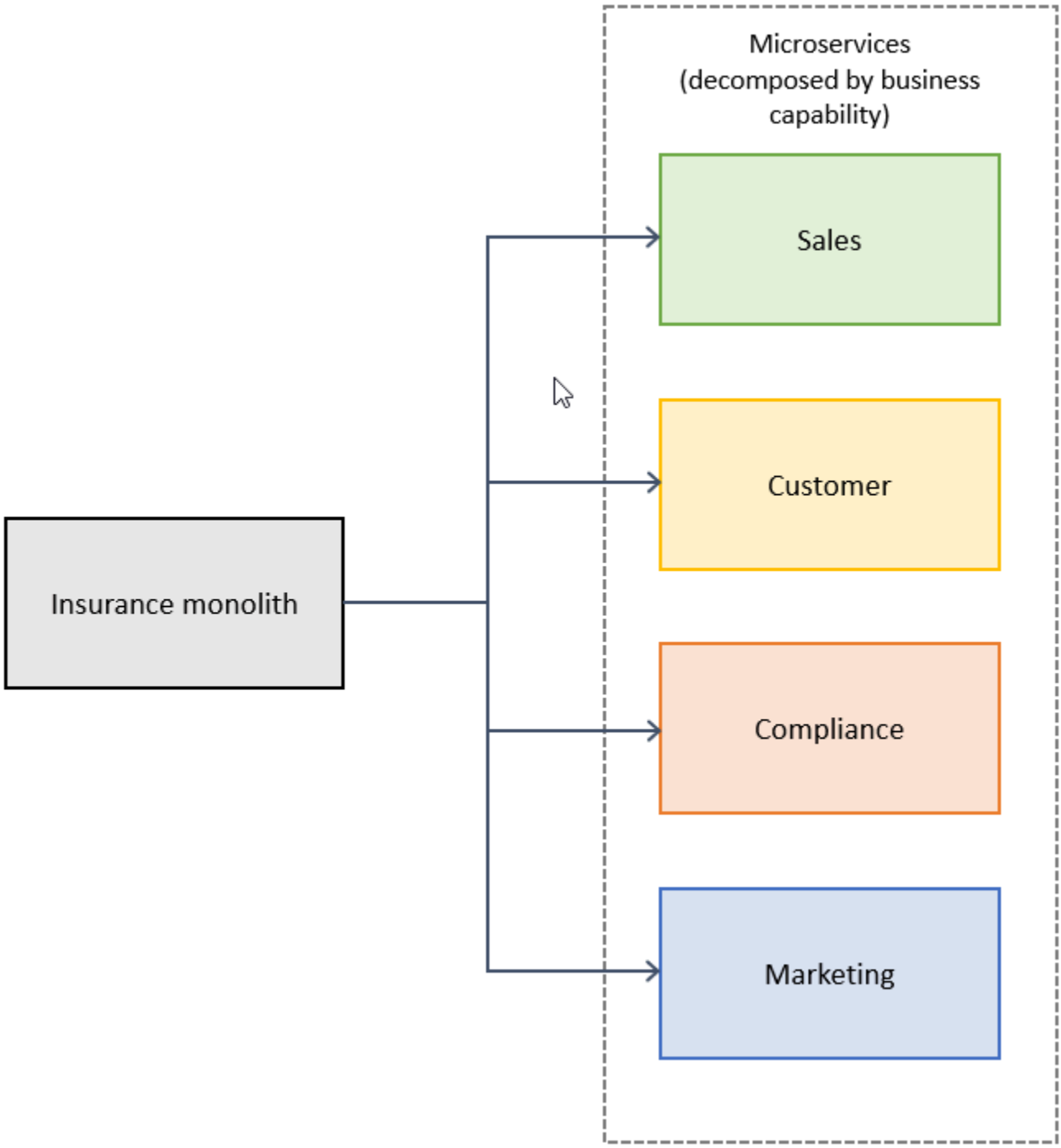
主題

依業務能力分解

您可以使用組織的業務流程或功能來分解整體。業務能力是企業用來產生價值的方式（例如，銷售、客戶服務或行銷）。一般而言，組織具有多種業務功能，這些功能因產業或產業而異。如果您的團隊充分了解組織的業務單位，而且您擁有每個業務單位的主題專家 (SMEs)，請使用此模式。下表說明使用此模式的優點和缺點。

優點	缺點
• 如果業務功能相對穩定，則產生穩定的微服務架構。 • 開發團隊是跨職能的，並圍繞提供商業價值而非技術功能進行組織。 • 服務鬆散耦合。	• 應用程式設計與商業模型緊密結合。 • 需要深入了解整體業務，因為識別業務功能和服務可能很困難。

在下圖中，保險體會根據業務功能分解為四個微服務。



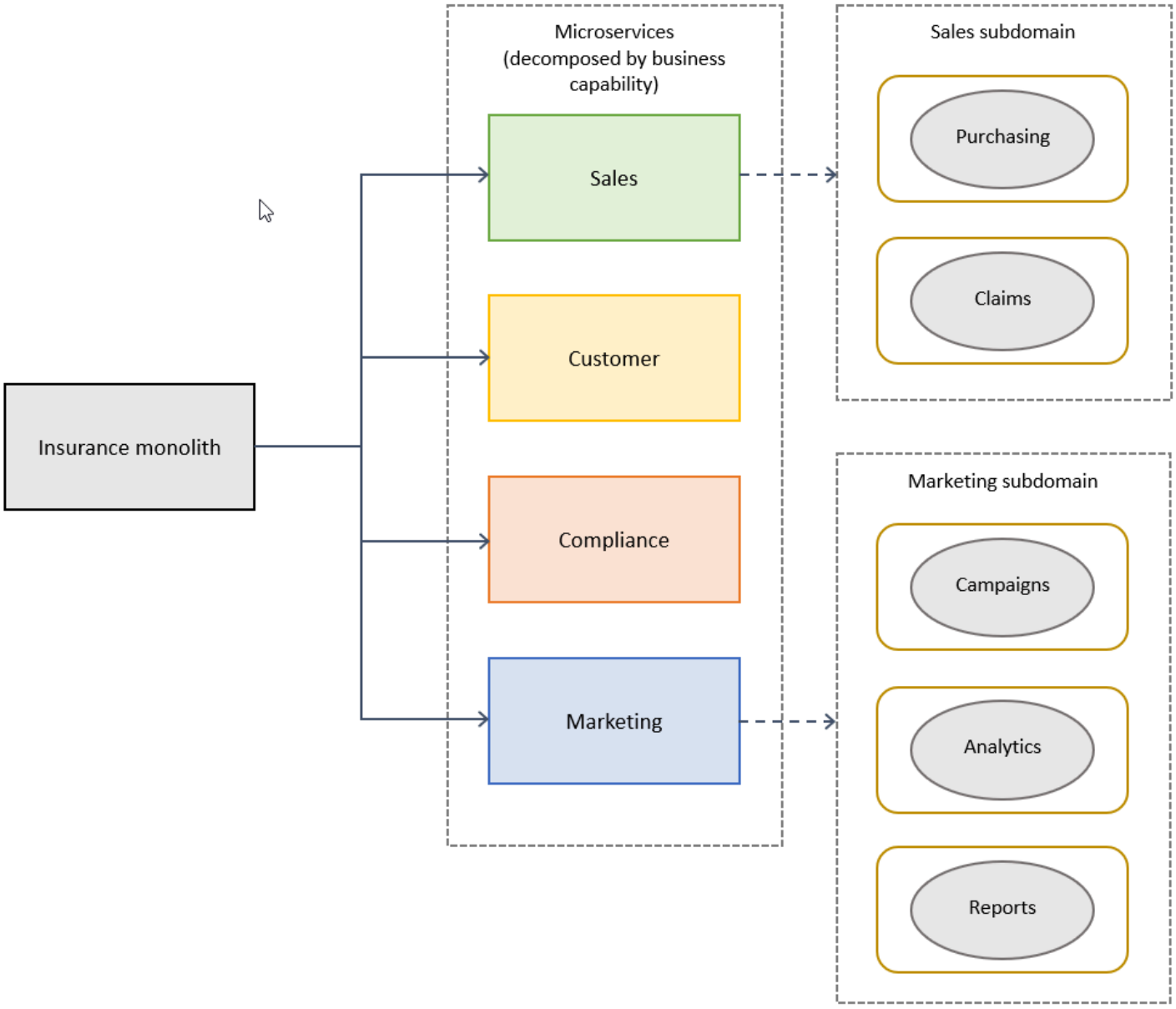
依子網域分解

此模式使用網域驅動設計 (DDD) 子網域來分解單體。此方法會將組織的網域模型細分為標示為核心（業務的關鍵差異因素）、支援（可能與業務相關，但與差異因素無關）或一般（非業務特定）的個別子網域。此模式適用於在子網域相關模組之間具有明確定義界限的現有單體系統。這表示您可以透

過將現有模組重新封裝為微服務來分解整體，但不會大幅重寫現有程式碼。每個子網域都有一個模型，該模型的範圍稱為邊界內容。微服務是圍繞此邊界內容開發的。下表說明使用此模式的優點和缺點。

優點	缺點
• 鬆耦合架構提供可擴展性、彈性、可維護性、可擴展性、位置透明度、通訊協定獨立性和時間獨立性。 • 系統變得更具可擴展性和可預測性。	• 可以建立太多微服務，這使得服務探索和整合變得困難。 • 業務子網域很難識別，因為它們需要深入了解整體業務。

下圖顯示保險整體如何在由業務功能分解之後分解為子網域。



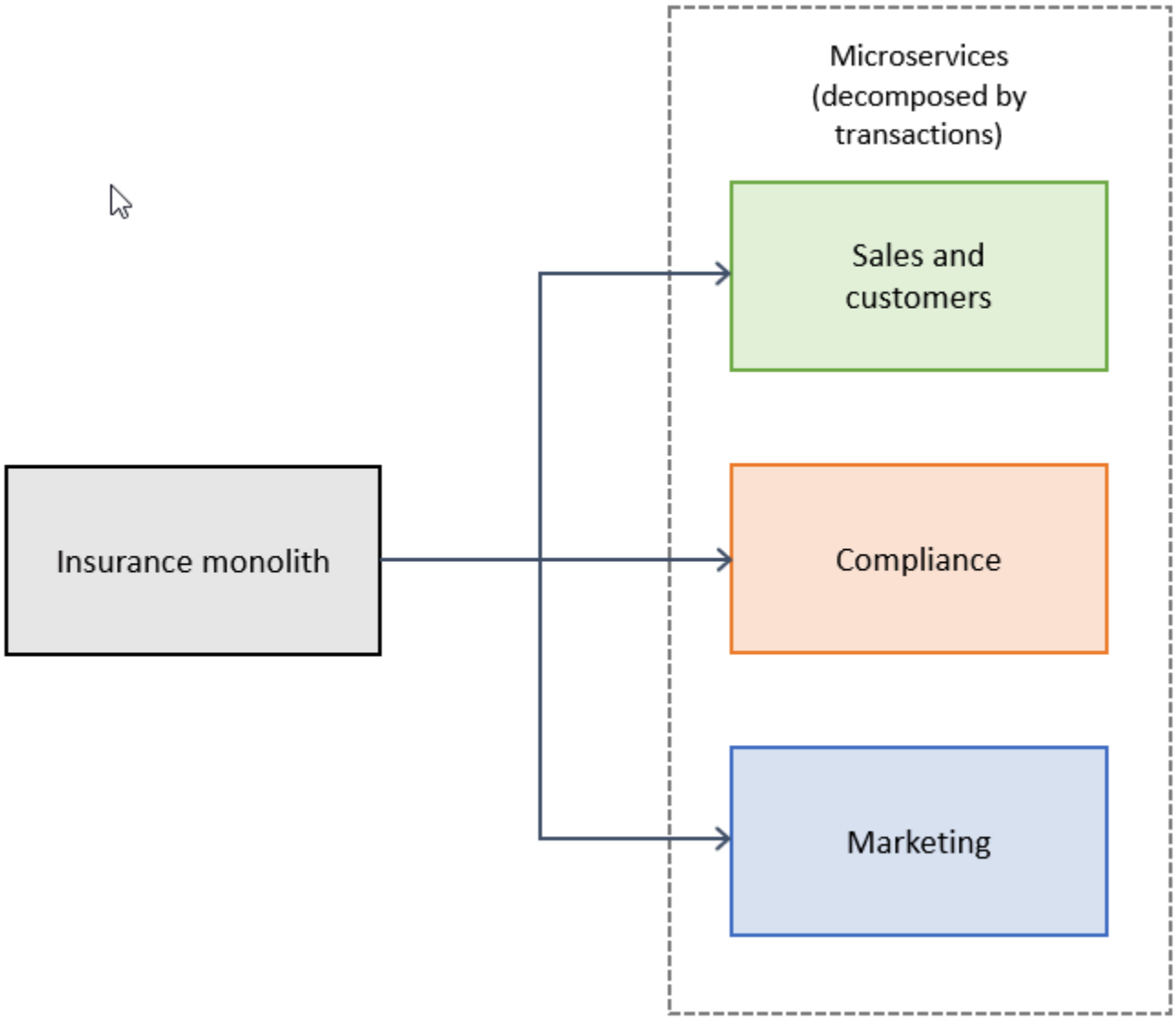
下圖顯示銷售和行銷服務細分為較小的微服務。購買和宣告模型是銷售的重要業務差異，並分為兩個單獨的微服務。透過使用行銷活動、分析和報告等支援的業務功能來分解行銷。

依交易分解

在分散式系統中，應用程式通常必須呼叫多個微服務才能完成一項商業交易。為了避免延遲問題或兩階段遞交問題，您可以根據交易將微服務分組。如果您認為回應時間很重要，且不同的模組在封裝之後不會建立整體，則此模式是適當的。下表說明使用此模式的優點和缺點。

優點	缺點
• 回應時間更快。 • 您不需要擔心資料一致性。 • 改善可用性。	• 多個模組可以封裝在一起，這可以建立整體。 • 在單一微服務中實作多個功能，而不是個別的微服務，這會增加成本和複雜性。 • 如果交易導向的微服務之間的業務網域數量和相依性很高，則交易導向的微服務可能會成長。 • 同一業務網域可能會同時部署不一致的版本。

在下圖中，根據交易，保險整體分為多個微服務。



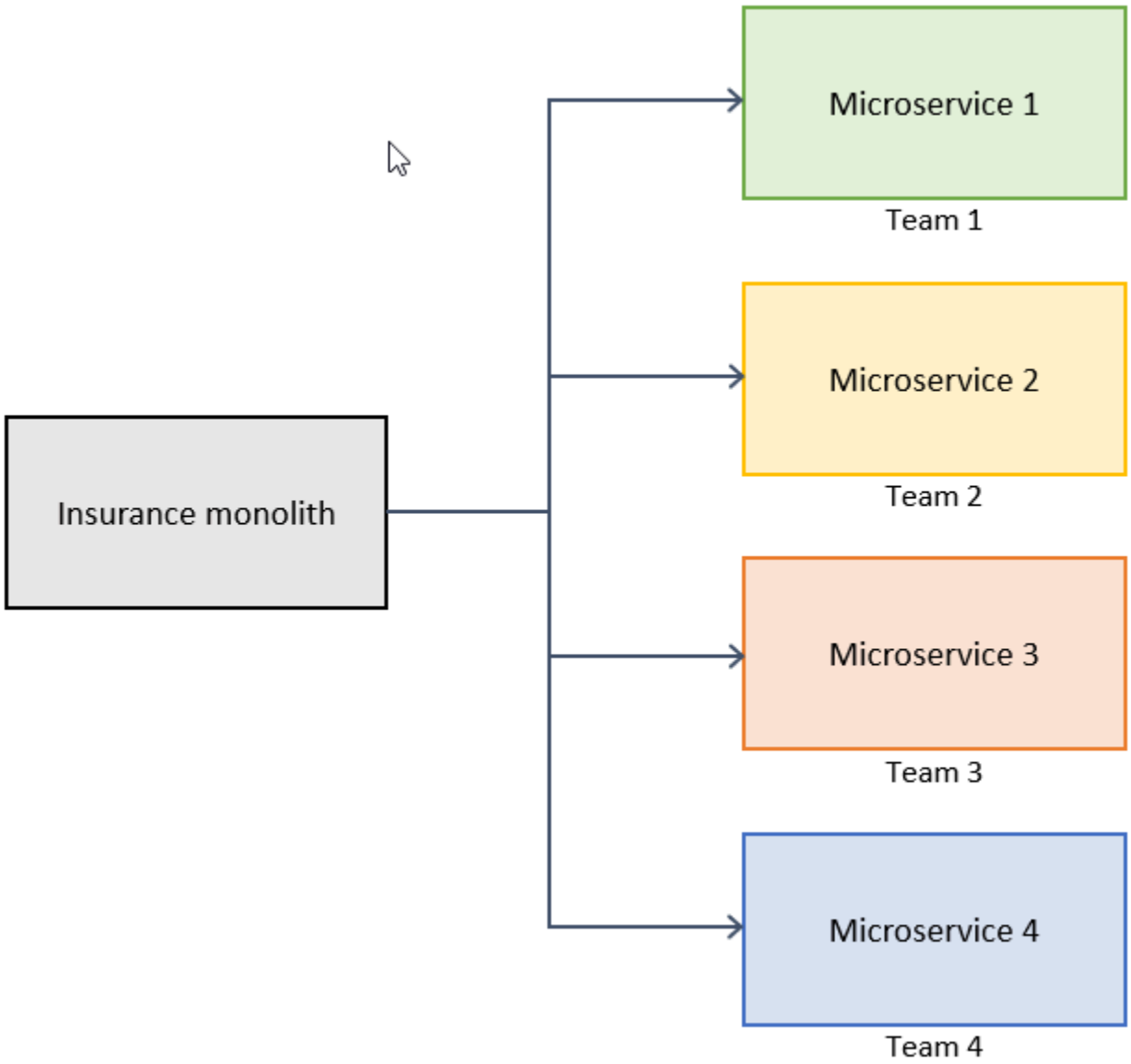
在保險系統中，索賠請求通常會在提交後標記給客戶。這表示如果沒有客戶微服務，就無法存在宣告服務。銷售和客戶封裝在一個微服務套件中，而商業交易需要與兩者協調。

每個團隊模式的服務

每個團隊模式的服務不會透過業務功能或服務分解整體，而是將其分解為由個別團隊管理的微服務。每個團隊都負責業務功能，並擁有該功能的程式碼庫。團隊獨立開發、測試、部署或擴展其服務，主要與其他團隊互動以交涉 APIs。建議您將每個微服務指派給單一團隊。不過，如果團隊夠大，多個子團隊可以在相同的團隊結構中擁有單獨的微服務。下表說明使用此模式的優點和缺點。

優點	缺點
• 團隊以最小的協調性獨立運作。 • 程式碼基礎和微服務不會由多個團隊共用。 • 團隊可以快速創新和迭代產品功能。 • 不同的團隊可以使用不同的技術、架構或程式設計語言。重要：這些應該隱藏在定義明確且穩定的公有 API 後方。	• 將團隊與最終使用者功能或業務功能保持一致可能很困難。 • 需要額外的努力來提供更大的協調應用程式增量，特別是在團隊之間有循環相依性時。

下圖顯示單體如何分割為由個別團隊管理、維護和交付的微服務。



Strangler 無花果模式

到目前為止，本指南中討論的設計模式適用於分解綠地專案的應用程式。涉及大型單體應用程式的布朗菲爾德專案呢？ 將先前的設計模式套用到它們會很困難，因為在主動使用它們時將其分成較小的部分是一項重大任務。

Strangler fig 模式是由 Martin Fowler 引入的熱門設計模式，其受到特定類型的 fig 的啟發，該類 fig 會在樹的上分支中植入。現有的樹最初會成為新無花果的支援結構。 然後，該無花果會將其根傳送到地面，逐漸包圍原始樹狀結構，並只將新的自助式無花果留在原處。

此模式通常用於將特定功能取代為新服務，以累加方式將單體應用程式轉換為微服務。目標是讓舊版和新的現代化版本共存。新系統最初由現有系統支援並包裝。此支援可讓新系統有時間成長，並可能完全取代舊系統。

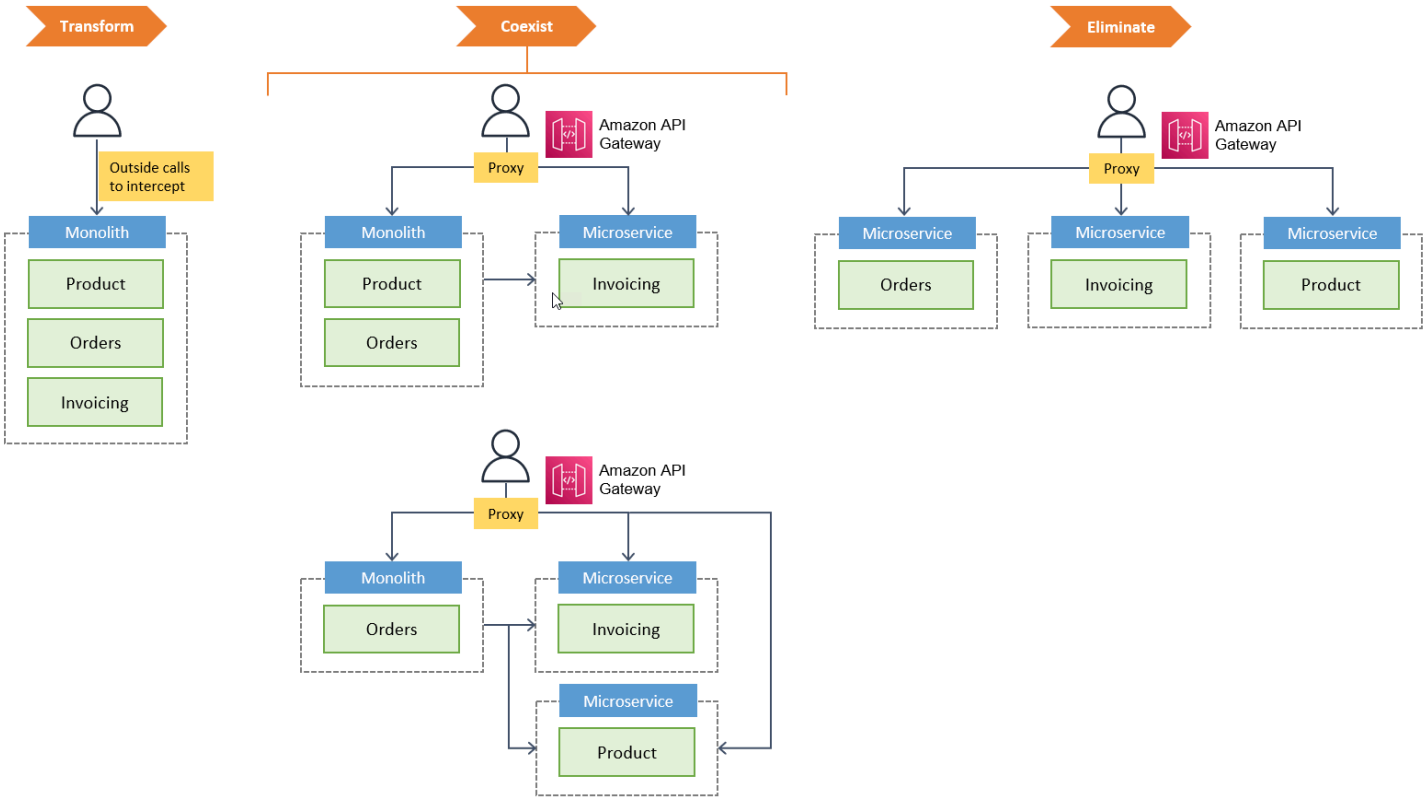
透過實作 strangler fig 模式，從單體應用程式轉換為微服務的程序包含三個步驟：轉換、共存和消除：

- 轉換 – 透過與舊版應用程式平行移植或重寫這些元件，來識別和建立現代化元件。
- 共存 – 保留整體應用程式以進行轉返。在整體周邊整合 HTTP 代理 （例如 Amazon API Gateway) 以攔截外部系統呼叫，並將流量重新導向至現代化版本。這可協助您逐步實作功能。
- 消除 – 從整體淘汰舊功能，因為流量會從舊版整體重新導向到現代化服務。

下表說明使用 strangler fig 模式的優點和缺點。

優點	缺點
• 允許從服務正常遷移到一或多個替代服務。 • 讓舊服務保持運作狀態，同時重構為更新的版本。 • 提供新增服務和功能的能力，同時重構較舊的服務。 • 此模式可用於 APIs的版本控制。 • 模式可用於未升級或不會升級之解決方案的舊版互動。	• 不適用於複雜性較低且大小很小的小型系統。 • 無法在無法攔截和路由請求後端系統的系統中使用。 • 如果未正確設計，代理或臉部圖層可能會成為單一故障點或效能瓶頸。 • 每個重構服務都需要復原計劃，以便在發生錯誤時，快速安全地恢復到舊的執行方式。

下圖顯示如何透過將 strangler fig 模式套用至應用程式架構，將單體分割為微服務。這兩個系統都平行運作，但您會開始將功能移到整體程式碼基礎之外，並使用新功能增強功能。這些新功能讓您有機會以最符合您需求的方式架構微服務。您將繼續從整體中剔除功能，直到全部被微服務取代為止。此時，您可以消除整體應用程式。這裡要注意的重點是整體和微服務都會一起存活一段時間。



依抽象模式分支

當您可以在整體周邊攔截呼叫時，strangler fig 模式運作良好。不過，如果您想要將存在於舊版應用程式堆疊中更深且具有上游相依性的元件現代化，建議您依抽象模式來分支。此模式可讓您變更現有的程式碼基礎，讓現代化版本與舊版安全地共存，而不會造成中斷。

若要透過抽象模式成功使用分支，請遵循此程序：

1. 識別具有上游相依性的整合元件。
2. 建立抽象層，代表要現代化之程式碼與其用戶端之間的互動。
3. 當抽象化就位時，變更現有的用戶端以使用新的抽象化。
4. 使用整合之外的重做功能建立新的抽象實作。
5. 準備好時，將抽象概念切換到新實作。
6. 當新實作提供所有必要功能給使用者，且整體不再使用時，請清除較舊的實作。

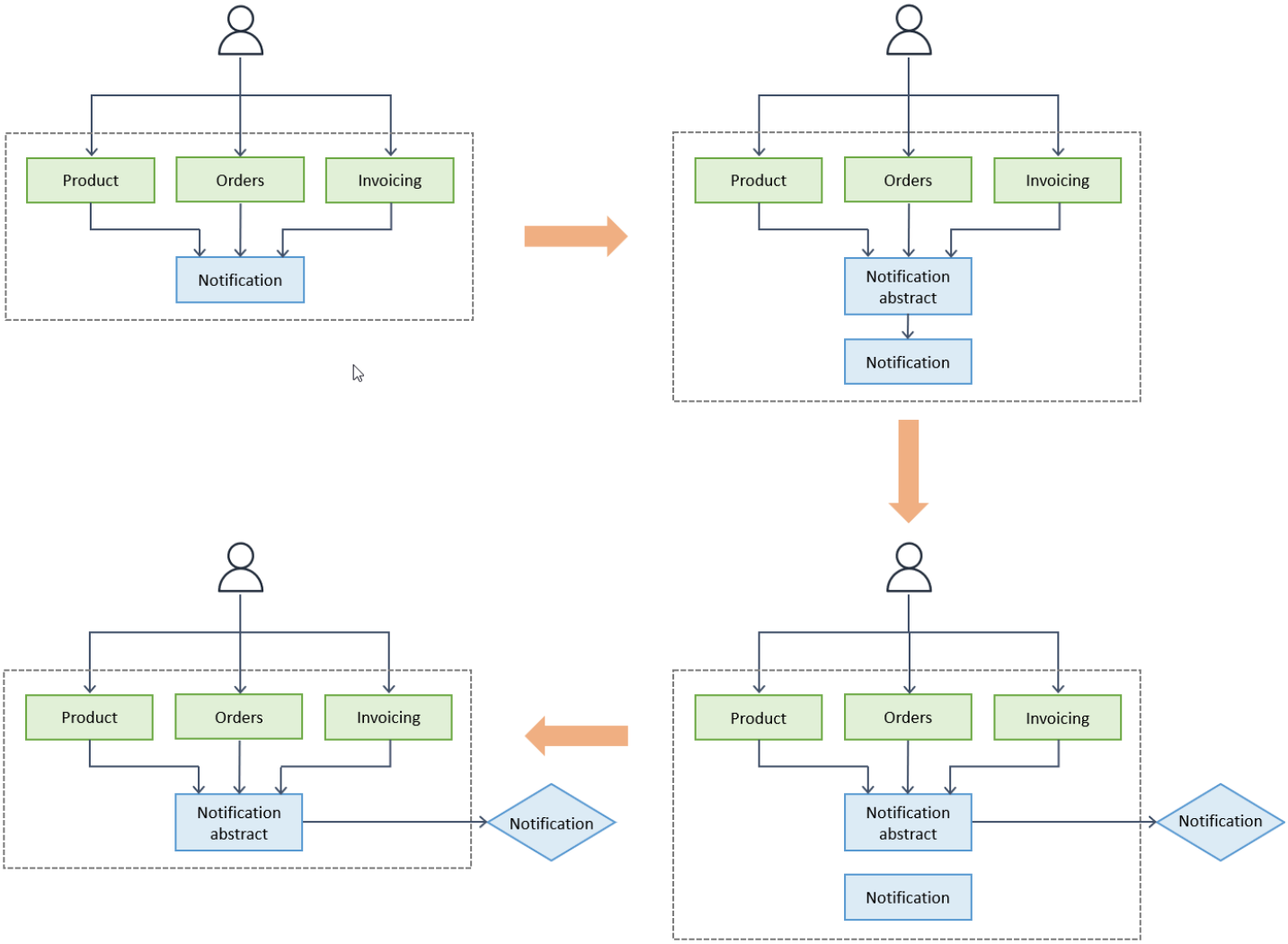
依抽象模式劃分的分支通常與功能切換混淆，這也可讓您逐步變更系統。差別在於功能切換旨在允許開發新功能，並在系統執行時讓使用者看不到這些功能。因此，功能切換會在部署時間或執行時間使用，

以選擇應用程式中是否顯示特定功能或一組功能。依抽象分割是一種開發技術，可與功能切換結合，以在舊實作和新實作之間切換。

下表說明透過抽象模式使用分支的優點和缺點。

優點	缺點
• 允許在發生錯誤（回溯相容）時可還原的增量變更。 • 當您無法在單體邊緣攔截對單體的呼叫時，可讓您擷取在單體內部深處的功能。 • 允許多個實作在軟體系統中共存。 • 透過使用中繼驗證步驟來呼叫新舊功能，提供實作備用機制的簡單方法。 • 支援持續交付，因為您的程式碼在整個重組階段一直運作。	• 如果涉及資料一致性，則不適用。 • 需要變更現有系統。 • 可能會為開發程序增加更多額外負荷，特別是在程式碼基底結構不良時。（在許多情況下，上行值得額外的努力，而重組越大，考慮透過抽象模式使用分支就越重要。）

下圖顯示保險體中通知元件的分支抽象模式。首先建立通知功能的摘要或界面。以小幅度遞增，現有用戶端會變更為使用新的抽象。這可能需要搜尋程式碼基底，以呼叫與通知元件相關的 APIs。您會建立通知功能的新實作，做為整體外部的微服務，並將其託管在現代化架構中。在您的整體中，您新建立的抽象界面充當中介裝置，並呼叫新的實作。以小幅度遞增，您可以將通知功能移植到新實作，直到完全測試並就緒為止，保持非作用中狀態。當新的實作準備就緒時，您可以切換抽象以使用它。建議您使用可輕鬆翻轉的切換機制（例如功能切換），以便在遇到任何問題時輕鬆切換回舊功能。當新的實作開始為您的使用者提供所有通知功能，且不再使用整體時，您可以清除較舊的實作，並移除您可能已實作的任何切換功能旗標


Product
Orders
Invoicing
Notification
Product
Orders
Invoicing
Notification abstract
Notification
Product
Orders
Invoicing
Notification abstract
Notification
Product
Orders
Invoicing
Notification abstract
Notification
Notification

分解整體常見問答集

本節提供有關分解整體的常見問題解答。

您可以使用多個模式來分解一個整體嗎？

是，您可以使用多種模式來分解整體。最常見的方法是透過業務能力模式分解整體，然後使用子網域模式分解來分解更多。

將整體分解為微服務如何影響 DevOps 程序？

由於您不需要在應用程式變更之後重新部署所有項目，因此您必須支援和擁有新增至部署程序的新建立的微服務。這可能會讓 DevOps 程序更複雜。

Resources

相關指南

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

其他資源

- 使用 AWS Copilot、Amazon ECS、Docker 和 將單體應用程式分解為微服務 AWS Fargate

文件歷史紀錄

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

變更	描述	日期
已新增模式	依抽象模式新增有關 strangler fig 和分支的資訊。 https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-decomposing-monoliths/branch-by-abstraction.html	2023 年 4 月 6 日
初次出版	—	2021 年 1 月 11 日

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。