AWS白皮书

# SaaS 架構基礎知識

# SaaS 架構基礎知識: AWS白皮书

# Table of Contents

# SaaS 架構基礎

出版日期：二零二二年八月三日 (文件修訂)

在軟體即服務 (SaaS) 模型中執行業務的範圍、目標和性質可能很難定義。用來表徵 SaaS 的術語和模式會根據其來源而有所不同。本文檔的目標是更好地定義 SaaS 的基本要素，並創建在設計和交付 SaaS 系統時應用的模式，術語和價值系統的更清晰的圖片AWS。更廣泛的目標是提供一系列基礎見解，使客戶能夠更清楚地了解他們希望採用 SaaS 交付模型時應考慮的選項。

本白皮書針對 SaaS 旅程開始的 SaaS 建設者和建築師，以及希望改進對核心 SaaS 概念理解的經驗豐富的建設者。其中一些信息對於希望更熟悉 SaaS 領域的 SaaS 產品所有者和戰略家也很有用。

## 簡介

術語軟件即服務（SaaS）用於描述業務和交付模型。然而，挑戰在於成為 SaaS 的含義尚未普遍理解。

儘管對 SaaS 的一些核心支柱有一些協議，但對於 SaaS 意味著什麼，仍然存在一些混亂。團隊查看 SaaS 的方式有一些變化是很自然的。同時，對於那些探索 SaaS 交付模型的人來說，缺乏 SaaS 概念和術語的清晰度可能會造成一些混亂。

本文件著重於概述用來描述 SaaS 核心概念的術語。圍繞這些概念擁有共同的思維方式，可以清楚地了解 SaaS 架構的基礎元素，從而為您提供描述 SaaS 架構結構的共用詞彙。當您深入研究以這些主題為基礎的其他內容時，此功能特別有用。

本白皮書退出多租戶的架構細節，並探討了我們如何定義 SaaS 所代表的基本原理。理想情況下，這也將提供一組更清晰的術語，使組織能夠更快地對齊其 SaaS 解決方案的風格和性質。

## 你是否架構良好？

架AWS構良好的架構可協助您瞭解在雲端中建置系統時所做決策的優缺點。Framework 的六大支柱可讓您學習如何設計和操作可靠、安全、高效、符合成本效益且可持續發展的系統的架構最佳實務。使用中免費提供的 AWS Well-Architected ToolAWS 管理主控台，您可以針對每個支柱回答一組問題，根據這些最佳實務來檢閱工作負載。

在 SaaS 鏡頭中，我們專注於在其上架構軟體即服務 (SaaS) 工作負載的最佳實務。AWS

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

# SaaS 是一種商業模式

定義 SaaS 意味著什麼首先同意一個關鍵原則：SaaS 是一種商業模式。這意味著（最重要的是）採用 SaaS 交付模型是由一組業務目標直接驅動的。是的，技術將用於實現其中的一些目標，但 SaaS 是關於制定針對特定業務目標的思維方式和模型。

讓我們更仔細地看看與採用 SaaS 交付模型相關的一些關鍵業務目標。

- 敏捷性 — 本術語總結了 SaaS 的更廣泛目標。成功的 SaaS 公司構建的理念是，他們必須準備好不斷適應市場，客戶和競爭動態。偉大的 SaaS 公司的結構不斷接受新的定價模式，新的細分市場和新的客戶需求。
- 營運效率 — SaaS 公司仰賴營運效率來提升規模和敏捷性。這意味著實施一種文化和工具，該文化和工具專注於創建營運足跡，以促進頻繁和快速發布新功能。這也意味著擁有一個統一的體驗，可讓您統一管理、操作和部署所有客戶環境。一去不復返了支持一次性版本和自定義的想法。SaaS 企業將營運效率作為其成功發展和擴展業務能力的核心支柱。
- 無摩擦入職 — 作為更加敏捷和擁抱增長的一部分，您還必須在減少租戶客戶入職過程中的任何摩擦方面付出溢價。這通常適用於企業對企業（B2B）和business-to-customer（B2C）客戶。無論您支持哪個部分或類型的客戶，您仍然需要專注於為客戶創造價值的時間。轉向以服務為中心的模式需要 SaaS 業務專注於客戶體驗的各個方面，並特別強調整體入職生命週期的可重複性和效率。
- 創新 — 轉向 SaaS 不僅僅是滿足當前客戶的需求，而是為了實現可讓您進行創新的基礎元素。您想要在 SaaS 模型中回應和回應客戶需求。但是，您還希望利用這種敏捷性來推動 future 的創新，從而為客戶開拓新的市場，機會和效率。
- 市場反應 — SaaS 擺脫了季度發布和兩年計劃的傳統概念。它依賴於其敏捷性，使組織能夠以近乎實時的方式對市場動態做出反應和響應。對 SaaS 的組織、技術和文化元素的投資創造了基於新興客戶和市場動態的商業策略的機會。
- 成長 — SaaS 是以成長為中心的業務策略。將組織的所有移動部分圍繞敏捷性和效率進行調整，使 SaaS 組織能夠定位成長模型。這意味著制定適當的機制，以接受並歡迎您的 SaaS 產品的快速採用。

您會注意到，這些項目中的每一個都集中在業務成果上。有廣泛的技術策略和模式可用於構建 SaaS 系統。但是，這些技術策略沒有改變更廣泛的商業故事。

當我們與組織坐下來，詢問他們在採用 SaaS 的過程中想要達成什麼目標時，我們始終從這場以業務為中心的討論開始。技術選擇很重要，但必須在這些業務目標的背景下實現它們。舉例來說，作為多租戶而沒有達到敏捷性、營運效率或無摩擦上線，將會損害您的 SaaS 業務的成功。

以此為背景，讓我們嘗試將其形式化為 SaaS 的更簡潔的定義，該定義符合先前概述的原則：

SaaS 是一種商業和軟體交付模式，讓組織能夠以低摩擦、以服務為中心的模式提供解決方案，從而為客戶和供應商提供最大價值。它依賴靈活性和營運效率作為促進成長、觸及和創新的業務策略的支柱。

您應該看到業務目標之間的一致性，以及它們如何依賴為所有客戶提供共享體驗。轉向 SaaS 的很大一部分意味著擺脫可能是傳統軟件模型一部分的一次性自定義。為客戶提供專業化的任何努力通常都會使我們遠離我們試圖通過 SaaS 實現的核心價值觀。

# 您是服務，而不是產品

採用「服務」模式不僅僅是行銷或術語。在服務思維方式中，您會發現自己偏離了傳統以產品為基礎的開發方法的各個方面。雖然特色和功能對每個產品都很重要，但 SaaS 更加重視客戶對您服務的體驗。

這代表什麼意思？ 在以服務為中心的模型中，您可以更多地思考客戶如何加入您的服務、他們實現價值的速度，以及您可以如何快速推出滿足客戶需求的功能。與您的服務建立、運作和管理方式相關的詳細資訊，不在客戶的視野之外。

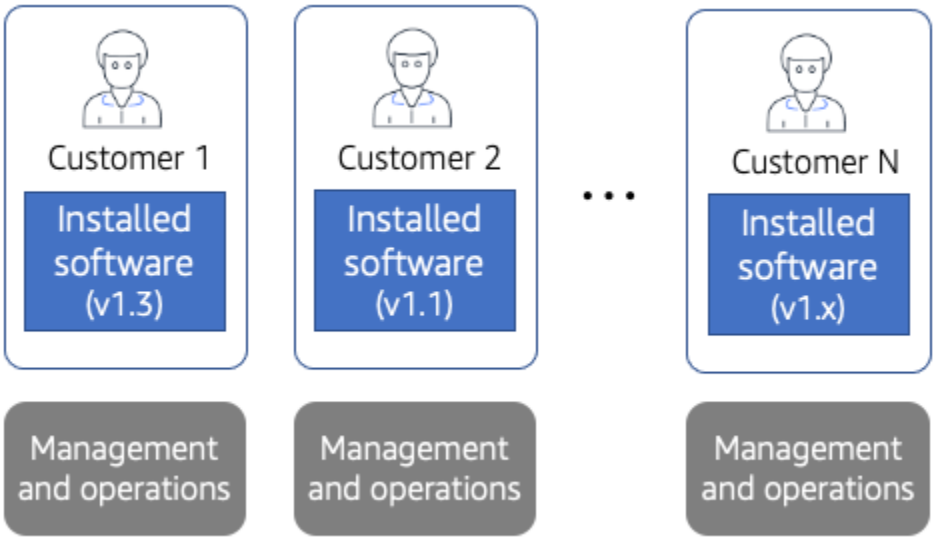
在這種模式下，我們將此 SaaS 服務與我們可能使用的任何其他服務一樣思考。如果我們在一家餐廳，我們當然關心食物，但我們也關心服務。您的服務器到達您的餐桌的速度有多快，它們多久補充水，食物來的速度有多快-這些都是服務體驗的所有措施。這是同樣的心態和價值體系，它應該塑造我們如何構建 SaaS 服務。

這種as-a-service模式應該對您如何建立團隊和服務有很大的影響。您積壓的工作現在會將這些體驗屬性放在與特性和功能相同或更高的基準上。業務還將其視為 SaaS 產品長期成長和成功的基礎。

# 初始動機

要了解 SaaS，讓我們從創建 SaaS 業務時嘗試實現的相當簡單的概念開始。最好的開始是查看傳統（非 SaaS）軟件的創建，管理和操作方式。

下圖提供數個廠商如何封裝及交付其解決方案的概念性檢視。




用於包裝和交付軟件解決方案的經典模型

在此圖中，我們描述了客戶環境的集合。這些客戶代表購買廠商軟體的不同公司或實體。這些客戶基本上都是在安裝軟體提供者產品的獨立環境中執行。

在此模式中，每位客戶的安裝都會被視為專屬於該客戶的獨立環境。這表示客戶將自己視為這些環境的擁有者，可能會要求一次性自訂或支援其需求的獨特組態。

這種心態的一個常見的副作用是，客戶將控制他們正在運行的產品的哪個版本。基於各種原因，可能會發生此問題。客戶可能會擔心新功能，或擔心採用新版本相關的中斷。

您可以想像這種動態如何影響軟體提供者的營運足跡。您允許客戶擁有一次性環境越多，管理、更新和支援每位客戶的各種組態就越具挑戰性。

對於一次性環境的需求，通常需要組織建立專門的團隊，為每位客戶提供獨立的管理和營運經驗。雖然其中一些資源可能會在客戶之間共用，但此模型通常會為每個已入職的新客戶引入增量費用。

讓每位客戶在自己的環境 (雲端或內部部署) 執行解決方案也會影響成本。雖然您可以嘗試擴展這些環境，但擴展將限制為單一客戶的活動。基本上，您的成本最佳化僅限於您在個別客戶環境範圍內可達成的目標。這也意味著您可能需要單獨的擴展策略來適應客戶之間的活動變化。

最初，一些軟件企業會將此模型視為一個強大的結構。他們利用提供一次性自訂作為銷售工具的功能，讓新客戶能夠強加其環境獨有的需求。儘管客戶數量和業務的增長仍然溫和，但這種模式似乎完全可持續。

然而，隨著公司開始取得更廣泛的成功，這種模式的限制開始創造真正的挑戰。例如，想像一下，您的業務達到顯著增長峰值的情況，您可以快速增加許多新客戶。這種增長將開始增加營運開銷，管理複雜性，成本以及許多其他問題。

最終，這種模式的集體開銷和影響可能會從根本上破壞軟件業務的成功。第一個痛點可能是運營效率。與引進客戶相關的增量人員配置和成本開始侵蝕業務的利潤。

但是，操作問題只是挑戰的一部分。真正的問題是，隨著規模的擴展，這種模式會直接開始影響企業發布新功能並與市場保持同步的能力。當每個客戶都有自己的環境時，提供者在嘗試將新功能引入其系統時，必須在許多更新、移轉和客戶需求之間取得平衡。

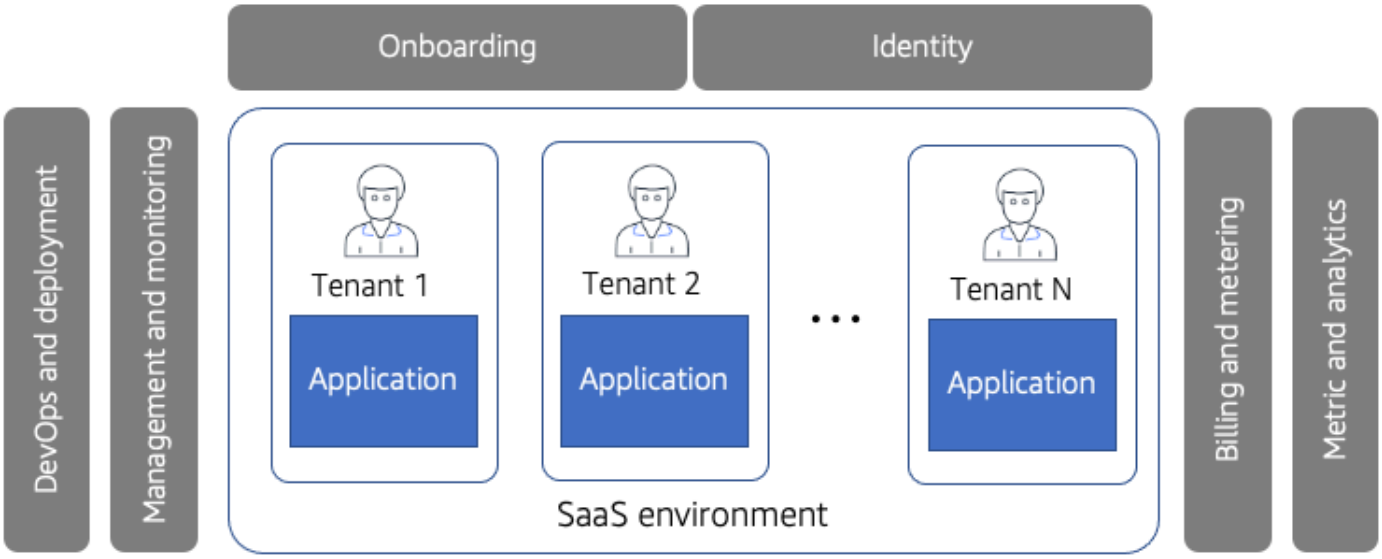
這通常會導致更長和更複雜的發行週期，這往往會減少每年發布的發布數量。更重要的是，這種複雜性使團隊在發布給客戶之前就花費更多時間來分析每個新功能。團隊開始更專注於驗證新功能，而減少交付速度。發布新功能的開銷變得如此重要，以至於團隊變得更加專注於測試機制，而不再關注推動其產品創新的新功能。

在這種較慢，更謹慎的模式下，團隊往往會有很長的循環時間，這些想法開始與落在客戶手中之間會有更大的差距。總體而言，這可能會阻礙您對市場動態和競爭壓力作出反應的能力。

# 轉移到統一的體驗

為了解決這種傳統軟體困境的需求，組織轉向一種模式，讓他們能夠建立單一、統一的體驗，讓客戶得以集體管理和運作。

下圖提供環境的概念性檢視，其中所有客戶都透過共用模型進行管理、登入、計費和操作。




透過共用模型管理、登入、計費和操作所有客戶的環境概念檢視

乍一看，這似乎並不是所有比以前的模型不同。但是，當我們進一步研究時，您會發現這兩種方法存在根本的重大差異。

首先，您會注意到客戶環境已重新命名為租用戶。租戶的這種概念是 SaaS 的基礎。基本概念是您擁有單一 SaaS 環境，而且每一位客戶都會被視為該環境的租用戶，消耗他們所需的資源。承租人可以是擁有許多使用者的公司，也可以直接與個別使用者建立關聯。

為了更好地理解租戶的想法，考慮公寓或商業建築的想法。這些建築物中的空間都出租給個人租戶。租戶依靠建築物的一些共享資源（水，電等），支付他們的消耗。

SaaS 租戶遵循類似的模式。您擁有 SaaS 環境的基礎結構，以及佔用該環境基礎結構的租用戶。每個租用戶使用的資源量可能會有所不同。這些租用戶也會共同管理、計費和運作。

如果你回到圖表，你會看到租賃的概念帶來了生活。在這裡，租戶不再擁有自己的環境。取而代之的是，所有租戶都在一個集體 SaaS 環境中安置和管理。

該圖還包括圍繞您的 SaaS 環境的一系列共享服務。這些服務適用於 SaaS 環境的所有租用戶。這表示上線和身分識別 (例如) 會由此環境的所有租用戶共用。管理、作業、部署、計費和指標也是如此。

這套統一服務的概念能夠普遍套用至所有租用戶，是 SaaS 的基礎。通過分享這些概念，您可以解決與上述傳統模型相關的許多挑戰。

此圖表中另一個重要的微妙元素是，此環境中的所有租用戶都執行相同版本的應用程式。一去不復返的想法是為每個客戶運行單獨的一次性版本。讓所有租用戶運行相同版本代表 SaaS 環境的基本區別屬性之一。

由於讓所有客戶執行相同版本的產品，您不再面臨傳統安裝軟體模型的許多挑戰。在整合模型中，可以透過單一共用程序將新功能部署至所有租用戶。

這種方法使您能夠使用單一操作窗格來管理和操作所有租用戶。這可讓您透過一般的操作體驗來管理和監控租用戶，讓您在不增加營運額外負荷的情況下新增承租人。這是 SaaS 價值主張的核心部分，它使團隊能夠降低營運支出並提高整體組織靈活性。

想像一下，在此模型中添加 100 或 1,000 個新客戶意味著什麼。您不必擔心這些新客戶會如何侵蝕利潤並增加複雜性，而是可以將此增長視為其所代表的商機。

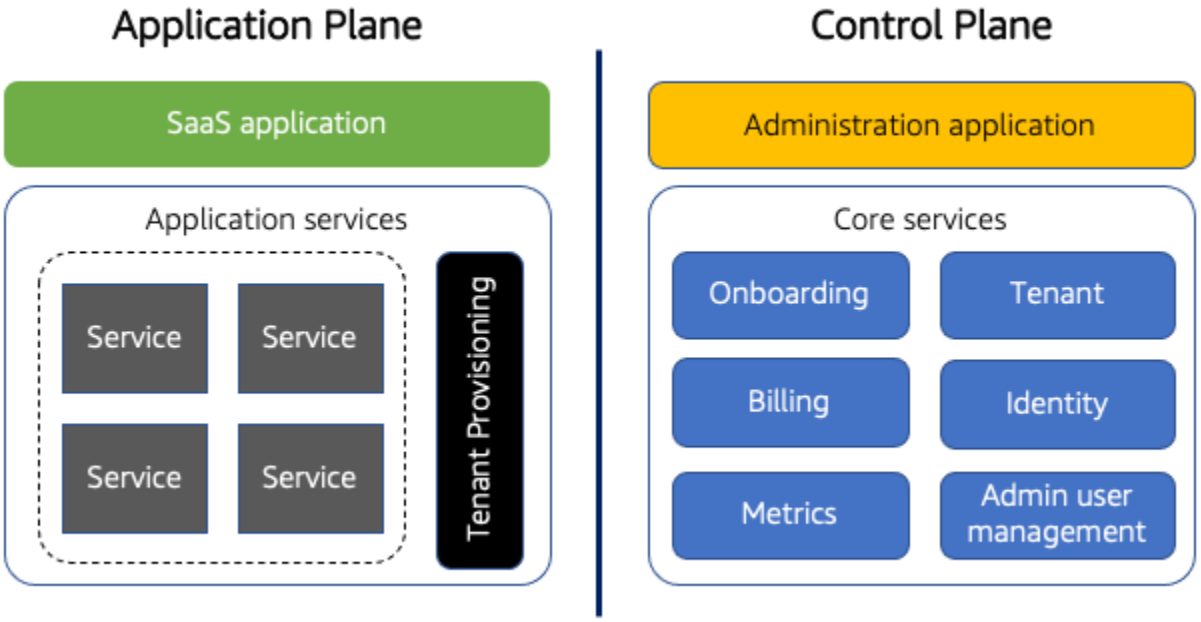
一般來說，SaaS 的重點放在這個模型中間的應用程序是如何實現的。企業希望專注於資料的儲存方式、資源的共用方式等等。但是，現實情況是，儘管這些細節絕對很重要，但可以構建應用程序的方法很多，並且仍然將其作為 SaaS 解決方案提供給客戶。

最重要的是擁有圍繞租用戶環境的單一統一體驗的更廣泛目標。擁有這種共同的經驗可以讓您推動與 SaaS 業務整體目標相關的成長、敏捷性和營運效率。

# 控制平面與應用程式平面

上圖提供核心 SaaS 架構概念的概念檢視。現在，讓我們深入研究這一點，並更好地定義您的 SaaS 環境如何分解為不同的層。對 SaaS 概念之間的界限進行更清晰的了解將使描述 SaaS 解決方案的移動部分變得更加容易。

下圖將您的 SaaS 環境劃分為兩個不同的平面。右側是控制平面。圖表的這一側包含用來上線、驗證、管理、操作和分析多租用戶環境的所有功能和服務。




控制平面與應用程式平面

此控制平面是任何多租戶 SaaS 模型的基礎。每個 SaaS 解決方案 (無論應用程式部署和隔離方案為何) 都必須包含那些服務，讓您能夠透過單一、統一的體驗管理和操作租用戶。

在控制平面內，我們進一步將其分解為兩個不同的元素。此處的核心服務代表用於協調多租用戶體驗的服務集合。我們提供了一些常見的服務範例，這些服務通常是核心的一部分，因此承認每個 SaaS 解決方案的核心服務可能會有所不同。

您也會注意到，我們會顯示一個單獨的管理應用程式。這代表 SaaS 提供者可能用來管理其多租戶環境的應用程式 (Web 應用程式、命令列介面或 API)。

一個重要的警告是，控制平面及其服務實際上並不是多租戶。該功能不提供 SaaS 應用程序的實際功能屬性（確實需要是多租戶）。例如，如果您查看其中一項核心服務，則找不到租用戶隔離以及屬於多租用戶應用程式功能一部分的其他結構。這些服務對所有租戶都是全球性的。

圖表的左側參照 SaaS 環境的應用程式平面。這是應用程式的多租戶功能所在的位置。圖表中出現的內容需要保持一些模糊，因為每個解決方案都可以根據您的域的需求，技術的佔用空間等進行不同的部署和分解。

應用程式網域會分成兩個元素。有 SaaS 應用程序代表您的解決方案的租戶體驗/應用程序。這是租戶觸摸以與您的 SaaS 應用程序進行交互的表面。然後有代表 SaaS 解決方案的業務邏輯和功能元素的後端服務。這些可能是微型服務，也可能是應用程式服務的其他封裝。

您也會注意到，我們已細分佈建。這樣做是為了強調一個事實，即任何在上線期間為租用戶佈建資源都是此應用程式網域的一部分。有些人可能會說，這是屬於控制平面。不過，我們已將它放在應用程式網域中，因為它必須佈建和設定的資源會更直接地連接到在應用程式平面中建立和設定的服務。

將其分解為不同的平面，可以更輕鬆地考慮 SaaS 架構的整體格局。更重要的是，它突出了一組完全超出應用程序功能範圍的服務的需求。

# 核心服務

先前參照的控制平面提到了一系列核心服務，這些服務代表了用來上線、管理和操作 SaaS 環境的典型服務。進一步強調其中一些服務的作用，以突出顯示其在 SaaS 環境中的範圍和目的可能會有所幫助。以下提供這些服務的簡短摘要：

- 上線 — 每個 SaaS 解決方案都必須提供順暢的機制，才能將新租用戶引入 SaaS 環境。這可以是自助註冊頁面，也可以是內部管理的體驗。無論哪種方式，SaaS 解決方案都應該盡力消除這種體驗中的內部和外部摩擦，並確保此過程的穩定性，效率和可重複性。它在支持 SaaS 業務的增長和規模方面起著至關重要的作用。一般而言，此服務會協調其他服務，以建立使用者、租用戶、隔離原則、佈建和每個租用戶資源。
- 承租人 — 租用戶服務提供集中管理租用戶原則、屬性和狀態的方法。關鍵在於租用戶不是個人用戶。事實上，租用戶可能與許多使用者相關聯。
- 身分識別 — SaaS 系統需要將使用者連線到租用戶的明確方法，以便將租用戶內容帶入其解決方案的驗證和授權體驗。這會影響入職體驗和使用者設定檔的整體管理。
- 計費 — 作為採用 SaaS 的一部分，組織通常採用新的計費模式。他們也可能會探索與第三方帳單供應商的整合。這項核心服務主要著重於支援新租用戶的上線，以及收集用於產生租用戶帳單的耗用和活動資料。
- 指標 — SaaS 團隊在很大程度上依賴他們擷取和分析豐富指標資料的能力，這些資料可以更清楚地瞭解租用戶如何使用其系統、使用資源的方式以及租用戶如何與其系統互動。此資料用於形成營運、產品和業務策略。
- 管理員使用者管理 — SaaS 系統必須支援租用戶使用者和管理員使用者。管理員使用者代表 SaaS 提供者的管理員。他們將登錄到您的操作體驗，以監控和管理您的 SaaS 環境。

# 重新定義多租戶

多租戶和 SaaS 的術語通常緊密連接。在某些情況下，組織將 SaaS 和多租戶描述為相同的事情。儘管這看起來很自然，但將 SaaS 和多租戶等同於往往會帶領團隊對 SaaS 進行純粹的技術觀點，而實際上，SaaS 更像是一種商業模式而不是架構策略。

為了更好地理解這個概念，讓我們從多租戶的經典視圖開始。在這個純粹以基礎架構為中心的檢視中，多租戶可用來描述租戶如何共用資源，以提升靈活性和成本效益。

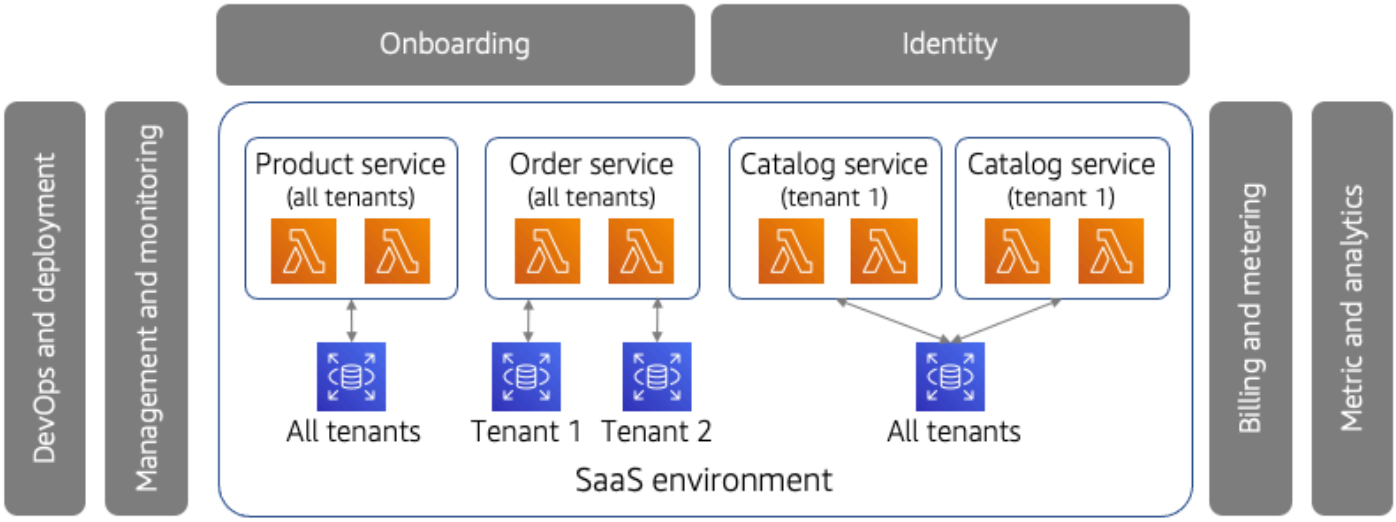
例如，假設您有一個微型服務或 Amazon 彈性運算雲端 (Amazon EC2) 執行個體，該執行個體已被 SaaS 系統的多個租用戶所使用。這項服務會被視為在多租用戶模型中執行，因為租用戶正在共用執行此服務之基礎結構的使用。

這個定義的挑戰在於，它太直接地將多租戶的技術概念附加到 SaaS。它假設 SaaS 的定義特徵是它必須具有共享的多租戶基礎結構。當我們研究 SaaS 在不同環境中實現的各種方式時，對 SaaS 的這種觀點開始崩潰。

下圖提供了 SaaS 系統的視圖，該系統揭露了我們在定義多租戶方面面臨的一些挑戰。

在這裡，您將看到先前描述的傳統 SaaS 模型，其中包含一系列應用程式服務，周圍環繞著共用服務，可讓您集體管理和操作租用戶。

新功能是我們已經包含的微服務。該圖包括三個微服務範例：產品、訂單和目錄。如果您仔細觀察這些服務的租賃模式，您會注意到它們都採用了略有不同的租賃模式。




軟體 SaaS 和多租戶

產品服務會與所有租用戶共用其所有資源 (運算和儲存體)。這與多租戶的典型定義保持一致。但是，如果您查看訂單服務，您會看到它具有共用計算，但是每個租用戶都有個別的儲存空間。

目錄服務新增了另一個變體，其中每個租用戶的計算是分開的 (每個租用戶的個別微服務部署)，但它會共用所有租用戶的儲存區。

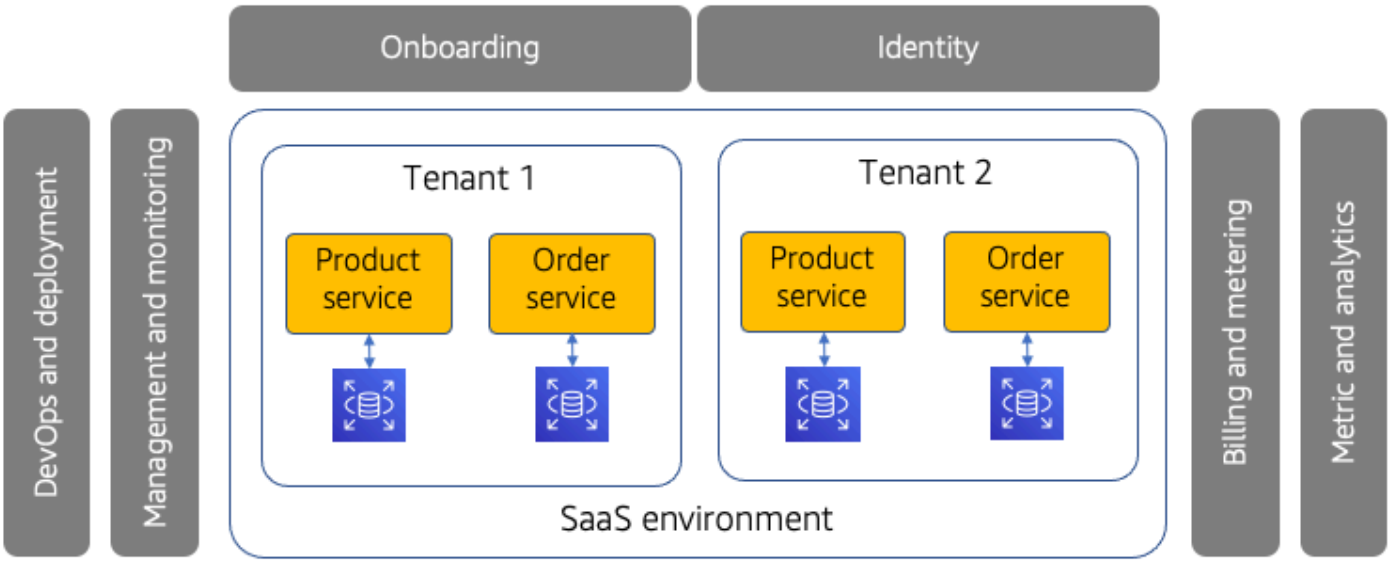
這種性質的變化在 SaaS 環境中很常見。嘈雜的鄰居、分層模型、隔離需求 — 這些都是您可能有選擇性地共用 SaaS 解決方案或隔離部分的原因之一。

考慮到這些變化以及許多其他可能性，要弄清楚如何使用多租戶一詞來分析此環境的特性變得更具挑戰性。總體而言，就客戶而言，這是一個多租戶環境。但是，如果我們使用最技術性的定義，則此環境的某些部分是多租戶，有些則不是。

這就是為什麼有必要擺脫使用術語多租戶來分析 SaaS 環境的原因。相反，我們可以討論如何在您的應用程序中實現多租戶，但避免使用它將解決方案表徵為 SaaS。如果要使用術語「多租戶」，則使用它將整個 SaaS 環境描述為多租戶是更有意義的，因為知道架構的某些部分可能是共享的，有些可能不會。整體而言，您仍在多租用戶模型中操作和管理此環境。

## 極端情況

為了更好地突出這種租賃概念，讓我們看一個 SaaS 模型，該模型具有共享零資源的租戶。下圖提供某些 SaaS 提供者使用的 SaaS 環境範例。




每個租戶的堆疊

在此圖中，您將看到我們仍然擁有圍繞這些租戶的共同環境。但是，每個承租人都會使用專用的資源集合進行部署。沒有任何東西是由此模型中的租戶共享。

此範例會挑戰多租戶的意義。即使沒有任何資源共享，這是一個多租戶環境嗎？ 使用此系統的租用戶對於具有共用資源的 SaaS 環境的期望相同。事實上，他們可能沒有意識到他們的資源是如何在 SaaS 環境的引擎蓋下部署的。

即使這些租用戶在孤立的基礎架構中執行，它們仍然是集體管理和運作的。他們共享一個統一的入職、身分識別、指標、計費和操作經驗。此外，當新版本發行時，它會部署到所有租用戶。若要使用此功能，您無法允許個別租用戶進行一次性自訂。

這個極端範例為測試多租戶 SaaS 概念提供了一個很好的模型。雖然它可能無法實現共享基礎架構的所有效率，但它是一個完全有效的多租戶 SaaS 環境。對於某些客戶，他們的網域可能會指定部分或全部客戶在此模型中執行。這並不意味著他們不是 SaaS。如果他們使用這些共用服務，而且所有租用戶都執行相同的版本，這仍然符合 SaaS 的基本原則。

鑑於這些參數和 SaaS 的這個更廣泛的定義，您可以看到需要發展術語多租戶的使用。將集體管理和運行的任何 SaaS 系統稱為多租戶，這更有意義。然後，您可以推遲到更精細的術語，以描述在 SaaS 解決方案實施中如何共享或專用資源。

# 移除單一租用戶期限

作為使用術語多租戶的一部分，人們想要使用單一租戶一詞來描述 SaaS 環境是很自然的。但是，鑑於先前概述的背景，「單租戶」一詞會造成混淆。

上圖是單一租用戶環境嗎？ 雖然每個租用戶技術上都有自己的堆疊，但這些租用戶仍在多租用戶模型中操作和管理。這就是為什麼一般避免單租戶一詞的原因。取而代之的是，所有環境都被特徵為多租戶，因為它們只是實施一些租賃的變化，其中部分或全部資源共享或專用。

## 介紹筒倉和游泳池

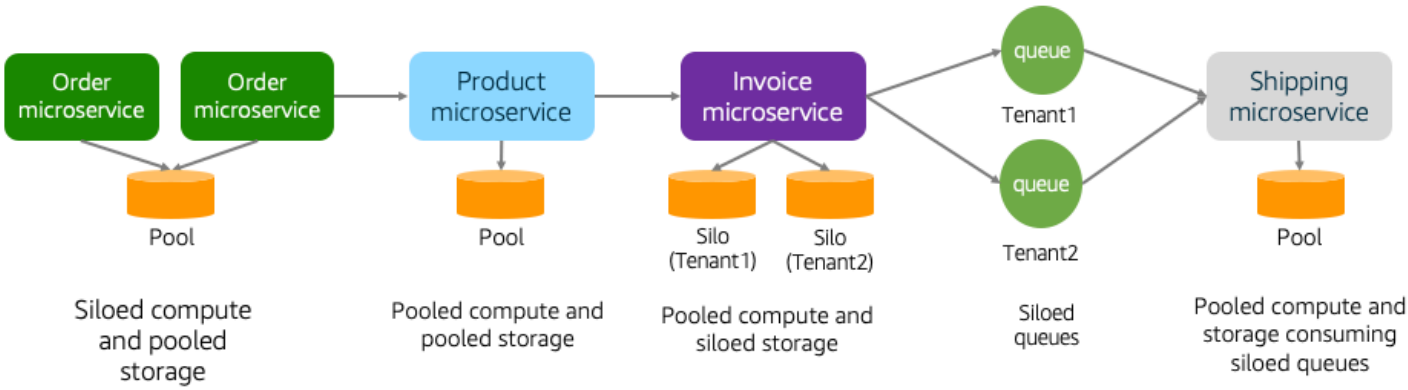
隨著所有這些模型的變化，並考慮到多租戶術語周圍的挑戰，我們引入了一些術語，可以讓我們更準確地捕獲和描述構建 SaaS 時使用的不同模型。

我們用來表徵 SaaS 環境中資源使用的兩個術語是筒倉和池。這些術語使我們能夠標記 SaaS 環境的性質，使用多租戶作為可應用於任何數量基礎模型的過度描述。

在最基本的層級中，「筒倉」一詞是用來描述資源專用於指定租用戶的案例。相反地，集區模型是用來描述租用戶共用資源的案例。

當我們查看筒倉和池術語的使用方式時，重要的是要清楚，筒倉和池不all-or-nothing是概念。筒倉和集區可套用至整個租用戶的資源堆疊，也可以選擇性地套用至整體 SaaS 環境的某些部分。因此，如果我們說某些資源正在使用筒倉模型，那並不意味著該環境中的所有資源都是孤立的。對於我們如何使用匯集術語也是如此。

下圖提供如何在 SaaS 環境中更精細地使用孤立和集區模型的範例：




筒倉和池模型

此圖表包括一系列範例，旨在說明筒倉和池模型的更有針對性的性質。如果您從左到右按照此操作，您將看到我們從訂單微服務開始。此微服務具有孤立的運算和集區儲存。它會與具有集區運算和集區儲存的產品服務互動。

然後，產品服務會與發票微服務互動，該微服務已集中運算和孤立儲存。此服務會透過佇列將訊息傳送至運送服務。佇列會部署在孤立的模型中。

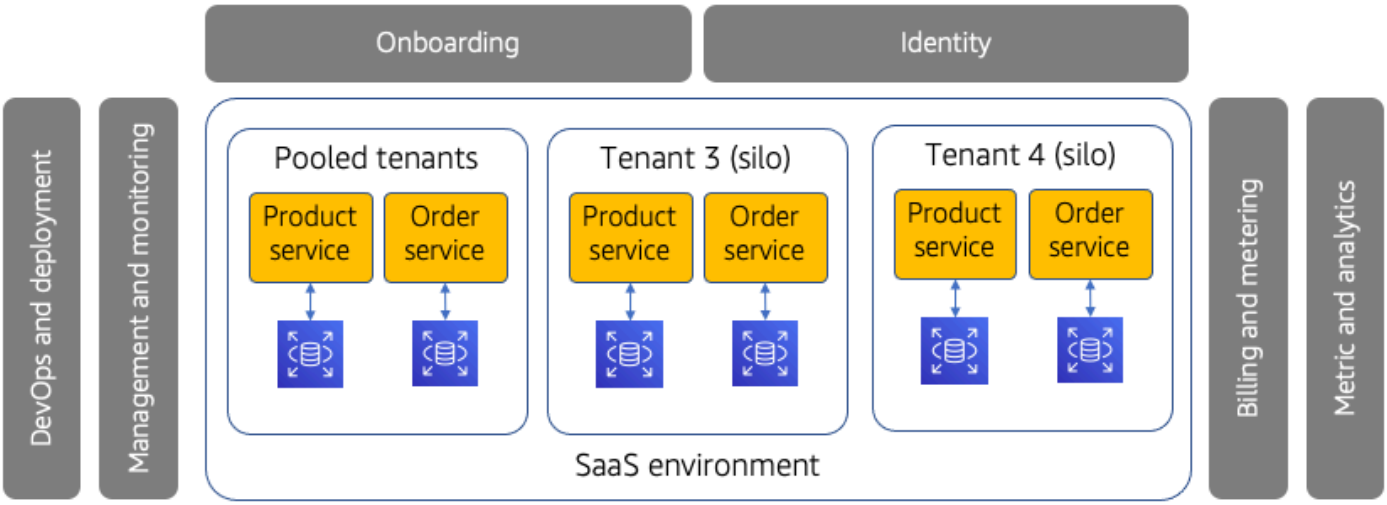
最後，傳送微服務會從孤立的佇列中取得訊息。它使用集區計算和存儲。

雖然這看起來有點複雜，但目標是強調筒倉和池概念的細微性質。當您設計和建置 SaaS 解決方案時，預期您會根據網域和客戶的需求做出這些筒倉和集區決策。

嘈雜的鄰居、隔離、分層和許多其他原因可能會影響您選擇套用筒倉或集區模型的方式和時機。

# 全棧筒倉和游泳池

筒倉和池也可用於描述整個 SaaS 堆棧。在這種方法中，租用戶的所有資源都是以專用或共用的方式部署。下圖提供了如何在 SaaS 環境中降落的示例。




完整堆疊筒倉和池模型

在此圖中，您會看到完整堆疊租用戶部署有三種不同的模型。首先，您會看到有完整堆疊集區環境。此集區中的租用戶會共用所有資源 (運算、儲存體等)。

顯示的其他兩個堆疊旨在代表完整堆疊孤立的租用戶環境。在此情況下，租用戶 3 和租用戶 4 會顯示為每個擁有自己的專用堆疊，其中沒有任何資源會與其他租用戶共用。

在同一 SaaS 環境中，這種組合筒倉和匯集模型並不是所有的非典型。例如，想像一下，您有一組基本層租用戶，這些租用戶需要支付適中的費用才能使用您的系統。這些租戶被放置在集區環境中。

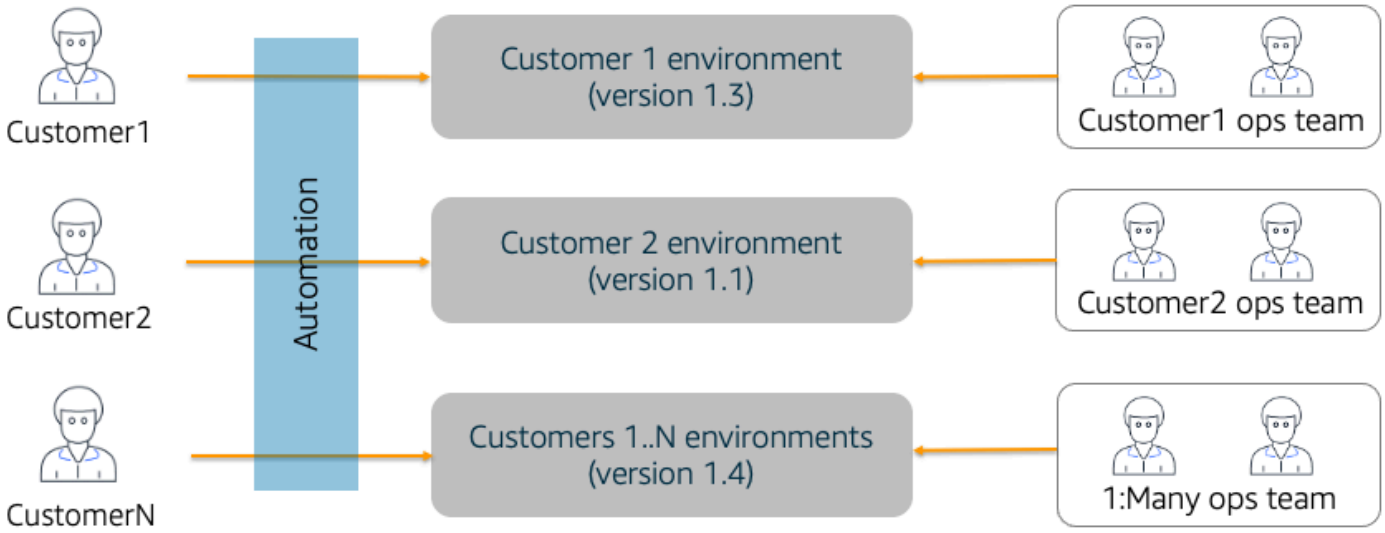
同時，您可能還有高級租戶願意為在筒倉中運行的特權付出更多費用。這些客戶會使用不同的堆疊來部署 (如圖所示)。

即使在這個模型中，您可能已允許租用戶在自己的完整堆疊筒倉中執行，這些孤島不允許對這些租戶進行任何一次性變化或自訂是至關重要的。在所有方面，這些堆疊中的每一個都應該使用相同版本的軟體執行堆疊的相同組態。當新版本發行時，它會部署到集區的租用戶環境，以及每個孤立的環境。

# SaaS 與託管服務提供商（MSP）的比較

SaaS 和託管服務提供者（MSP）模型之間的界限也存在一些混淆。如果您查看 MSP 模型，它似乎具有與 SaaS 模型相似的目標。

但是，如果您對 MSP 進行更多研究，您會發現 MSP 和 SaaS 實際上是不同的。下圖提供 MSP 環境的概念性檢視。




託管服務供應商 (MSP) 模型

此圖表示 MSP 模型的一種方法。在左側，您會看到在 MSP 模型中執行的客戶。一般來說，這裡的方法是使用任何可用的自動化來佈建每個客戶環境，並為該客戶安裝軟體。

右邊是 MSP 為了支援這些客戶環境而提供的營運足跡的一些近似值。

請務必注意，MSP 通常會安裝和管理特定客戶想要執行的產品版本。所有客戶都可以執行相同的版本，但在 MSP 模型中通常不需要這樣做。

一般策略是透過擁有這些環境的安裝和管理來簡化軟體提供者的生命週期。儘管這使提供者的生活更簡單，但它並不直接映射到 SaaS 產品至關重要的價值觀和思維方式。

重點是卸載管理責任。進行此舉並不等同於讓所有客戶在同一版本上執行，並具有單一、統一的管理和營運體驗。取而代之的是，MSP 通常允許不同的版本，並且經常將這些環境視為獨立的操作。

肯定有些領域 MSP 可能會開始與 SaaS 重疊。如果 MSP 基本上要求所有客戶執行相同的版本，且 MSP 能夠透過單一體驗集中上線、管理、操作和計費所有租用戶，那麼這可能開始 SaaS MSP 多。

更廣泛的主題是自動化環境安裝並不等於擁有 SaaS 環境。只有當您添加之前討論的所有其他警告時，這將代表更多的真正的 SaaS 模型。

如果我們從這個故事的技術和運營方面轉回來，MSP 和 SaaS 之間的界限就會變得更加明顯。一般而言，作為 SaaS 業務，您的產品項目的成功取決於您深入參與體驗所有移動部分的能力。

這通常意味著您可以掌握入職體驗的脈動，了解操作事件如何影響租戶，跟踪關鍵指標和分析，以及與客戶接近。在將其移交給其他人的 MSP 模型中，您最終可能會從經營 SaaS 業務的核心關鍵細節中刪除一個級別。

# SaaS 遷移

許多採用 SaaS 的提供商都從傳統安裝的軟件模型遷移到 SaaS（前面描述）。對於這些提供商而言，對 SaaS 的核心原則保持良好的一致性尤為重要。

在這裡，再次，可能會對遷移到 SaaS 模型的含義存在一些混亂。例如，有些人視圖遷移到雲中作為遷移到 SaaS。其他人認為將自動化添加到其安裝和佈建過程中，以實現遷移。

可以公平地說，每個組織都可能從不同的位置開始，有不同的遺留考慮因素，並且可能面臨不同的市場和競爭壓力。這意味著每次遷移都會有所不同。

儘管每個路徑都不同，但在某些區域中，圍繞塑造遷移策略的核心原則存在斷開連接。圍繞概念和原則進行良好的一致性可能會對 SaaS 遷移的整體成功產生重大影響。

根據先前概述的概念，應該清楚的是，轉向 SaaS 始於業務策略和目標。這一點可能會迷失在遷移設置中，其中存在盡快進入 SaaS 的壓力。

在此模式下，組織通常主要是將移轉視為技術練習。現實情況是，每個 SaaS 遷移都應該從清晰的目標客戶、服務體驗、營運目標等方面開始。更清楚地關注 SaaS 業務需要的外觀，將對將解決方案遷移到 SaaS 所採取的形狀、優先順序和路徑產生深遠影響。

從遷移一開始就擁有這個清晰的願景，為您遷移到 SaaS 的過程中如何遷移技術和業務奠定了基礎。當您在這條道路上展開時，請專注於那些可以告訴您最有關您要去哪裡的問題。

下表提供技術與業務移轉心態的對比性質檢視。

表 1 — 技術優先與企業優先移轉

| 科技第一的心態 | 企業第一的心態 |
| --- | --- |
| 我們如何隔離租戶數據？ | SaaS 如何幫助我們發展業務？ |
| 我們如何將用戶連接到租戶？ | 我們的目標是哪些區段？ |
| 我們如何避免嘈雜的鄰居條件？ | 這些區段的大小和輪廓是多少？ |
| 我們如何做 A/B 測試？ | 我們需要支援哪些層級？ |
| 我們如何根據租戶負載進行擴展？ | 我們的目標是什麼服務體驗？ |

| 科技第一的心態 | 企業第一的心態 |
| --- | --- |
| 我們應該使用哪個帳單提供商？ | 我們的定價和包裝策略是什麼？ |

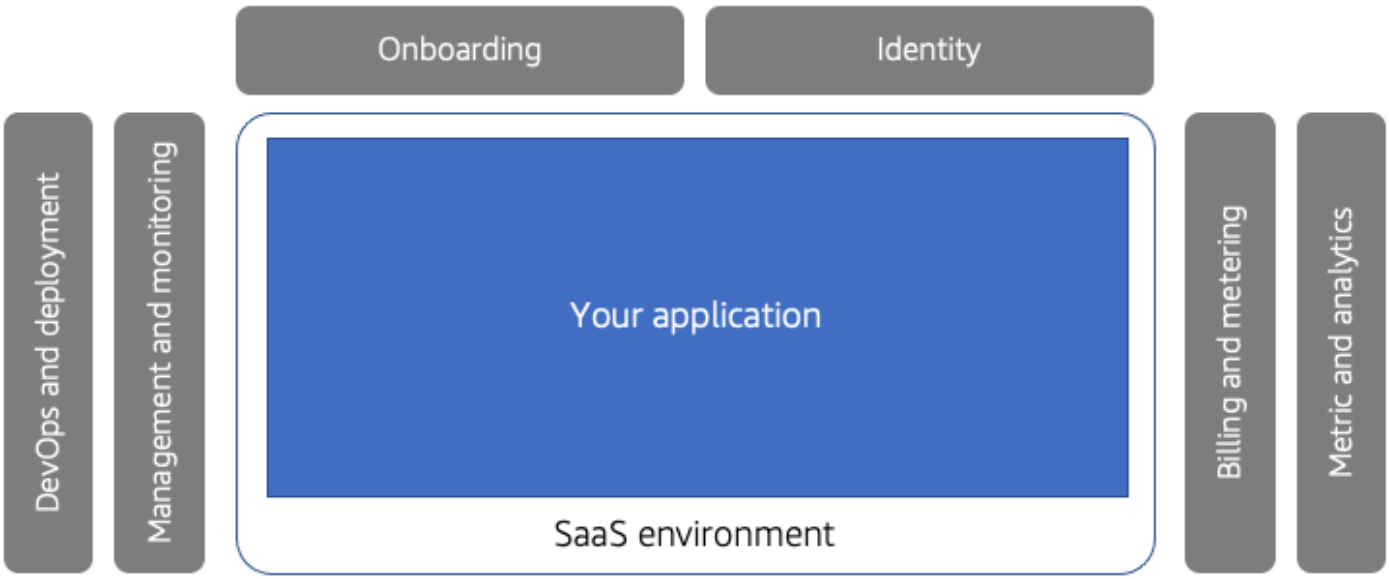
在左邊是可能的外觀範例。工程團隊非常專注於追求對任何 SaaS 架構都很重要的經典多租戶主題。

問題在於，左側許多問題的答案通常直接受右側問題的答案的影響。對於任何正在尋求遷移的人來說，這一點不太可能是新的。然而，現實情況是，許多組織開始追逐運營和成本效率作為他們的第一步，假設業務位將自己工作出來。

在此遷移策略中，您的舊環境如何演變為適合 SaaS 模型，也可能會產生混淆。這也是一個有許多遷移到 SaaS 選項的領域。但是，我們通常會主張任何遷移的共同價值體系。

在我們之前對 SaaS 原則的討論中，我們概述了用於描述 SaaS 環境的不同模式和術語。跨越所有這些解決方案的一個共同主題是擁有圍繞您的應用程序的共享服務的想法。身分識別、入職、指標、帳單 — 這些都稱為任何 SaaS 環境的核心元素。

現在，當我們看到遷移時，您會發現這些相同的共用服務在任何遷移故事中都扮演著關鍵角色。下圖提供移轉環境的概念檢視。




遷移至 SaaS

此圖表代表任何移轉路徑的目標體驗。它包括之前描述的所有相同共用服務。中間是應用程式的預留位置。

關鍵的想法是，您可以在此環境的中間著陸任意數量的應用程序模型。您在遷移的第一步可能會讓每個租用戶在自己的筒倉中執行。或者，您可能有一些混合架構，其中元素是孤立的，而其他位元的功能則透過現代化的微服務集合來解決。

您一開始將應用程式現代化的程度會根據舊有環境的性質、市場需求、成本考量等而有所不同。沒有改變的是這些共享服務的引入。

任何 SaaS 遷移都需要支援這些基礎共用服務，讓您的企業能夠以 SaaS 模式運作。例如，應用程序體系結構的所有變化都需要 SaaS 身份。您需要承租人感知作業，才能管理和監控 SaaS 解決方案。

將這些共用服務放在移轉的前端，可讓您向客戶呈現 SaaS 體驗，即使基礎應用程式仍在每個租用戶的完整堆疊中執行也是如此。

一般目標是讓您的應用程式在 SaaS 模型中執行。然後，您可以將注意力轉向應用程序的進一步現代化和改進。這種方法還允許您以更快的速度移動業務的其他部分（營銷，銷售，支持等）。更重要的是，這可讓您開始參與並收集客戶意見反應，這些意見反應可用於塑造持續的環境現代化。

重要的是要注意，您所設置的共享服務可能不包括您最終需要的每個功能或機制。主要目標是建立移轉一開始所需的共用機制。這可讓您專注於系統元素，這些元素對於應用程式架構的演進和營運演進至關重要。

# 取取 SaaS 身分

SaaS 將新增至應用程式的身分模型。當每個使用者都經過驗證時，他們必須連線至特定的承租人前後關聯。此承租人內容提供在整個 SaaS 環境中使用的租用戶的重要資訊。

租用戶與使用者的這種繫結通常稱為應用程式的 SaaS 身分。當每個使用者進行驗證時，您的身分識別提供者通常會產生包含使用者身分識別和租用戶身分的 Token。

將租用戶連接到使用者代表 SaaS 架構的基礎層面，具有許多下游影響。來自此身分識別程序的 Token 會流入應用程式的微服務，並用來建立租用戶感知記錄、記錄指標、計量計費、強制執行租用戶隔離等。

您必須避免使用依賴個別獨立機制來將使用者對應至租用戶的案例。這可能會破壞系統的安全性，並且通常會在您的架構中造成瓶頸。

# 隔離層級

您將客戶轉移到多租用戶模型中的次數越多，他們就越擔心一個租用戶存取另一個租用戶的資源的可能性。SaaS 系統包含明確的機制，可確保每個租用戶的資源（即使在共用基礎架構上執行）都是隔離的。

這就是我們所說的租戶隔離。租用戶隔離背後的想法是，您的 SaaS 架構引入了可嚴格控制資源存取的結構，並阻止任何嘗試存取其他租用戶資源的嘗試。

請注意，租用戶隔離與一般安全性機制是分開的。您的系統將支援驗證和授權；不過，租用戶使用者經過驗證並不表示您的系統已達到隔離狀態。隔離與可能是應用程式一部分的基本驗證和授權分開套用。

為了更好地理解這一點，假設您已經使用身份提供者來驗證對 SaaS 系統的訪問。此驗證體驗的 Token 也可能包含使用者角色的相關資訊，這些資訊可用來控制該使用者對特定應用程式功能的存取。這些結構提供安全性，但不是隔離。事實上，用戶可以通過身份驗證和授權，並且仍然可以訪問另一個租用戶的資源。關於身份驗證和授權的任何內容都不會阻止此訪問。

租用戶隔離專門著重於使用租用戶內容來限制對資源的存取。它會評估目前承租人的前後關聯，並使用該前後關聯來決定該承租人可存取的資源。它會將此隔離套用至該承租人內的所有使用者。

當我們研究如何在所有不同的 SaaS 架構模式中實現租戶隔離時，這會變得更具挑戰性。在某些情況下，可以透過將整個資源堆疊專用於租用戶來實現隔離，而網路 (或更粗糙的) 原則會阻止跨租用戶存取。在其他情況下，您可能有共用資源 (Amazon DynamoDB 表格中的項目)，這些資源需要更精細的政策來控制對資源的存取。

任何嘗試存取承租人資源的作用範圍應僅限於屬於該承租人的資源。SaaS 開發人員和架構師的工作是確定哪些工具和技術組合將支援特定應用程式的隔離需求。

# 資料分割

數據分區用於描述用於在多租戶環境中表示數據的不同策略。此術語廣泛用於涵蓋一系列不同的方法和模型，這些方法和模型可用於將不同的資料結構與個別租用戶產生關聯。

請注意，通常會有一種誘惑，將數據分區和租戶隔離視為可互換。這兩個概念並不意味著是等價的。當我們談論數據分區時，我們正在討論如何為單個租戶存儲租戶數據。分區數據不能確保數據被隔離。隔離仍必須個別套用，以確保一個租用戶無法存取另一個承租人的資源。

每種AWS儲存技術都會為資料分割策略帶來自己的一組考量。例如，在 Amazon DynamoDB 中隔離資料看起來會與使用 Amazon Relational Database Service 服務 (Amazon RDS) 隔離資料截然不同。

通常，當您考慮數據分區時，首先考慮數據是孤立還是匯集數據。在孤立的模型中，每個租用戶都有一個獨特的存儲構造，沒有共同混合的數據。對於集區分割，資料會根據決定與每個租用戶關聯的資料的承租人識別碼進行混合和分割。

舉例來說，使用 Amazon DynamoDB 時，孤立模型會為每個租用戶使用單獨的表格。在 Amazon DynamoDB 中共用資料的方式是將租用戶識別碼存放在管理所有租用戶資料的每個 Amazon DynamoDB 表的分區金鑰中，以達成資料共用。

您可以想像這在服務範圍內可能會有什麼不同，每個AWS服務都會引入自己的結構，這些建構可能需要不同的方法來實現每個服務的孤島和集區儲存模型。

雖然資料分割和租用戶隔離是單獨的主題，但您選擇的資料分割策略可能會受到資料隔離模型的影響。例如，您可能會倉儲一些存儲，因為這種方法最適合您的域或客戶的需求。或者，您可能會選擇 Silo，因為集區模型可能不允許您以解決方案所需的粒度層級強制執行隔離。

嘈雜的鄰居也會影響您的隔離方法。應用程式中的某些工作負載或使用案例可能需要分開保存，以限制來自其他租用戶的影響，或者符合服務等級協定 (SLA)。

# 計量、指標和計費

SaaS 的討論還傾向於包括計量，指標和計費的概念。這些概念經常被折疊成一個概念。但是，區分 SaaS 環境中計量、指標和計費的不同角色很重要。

這些概念的挑戰在於它們通常具有相同單詞的重疊用法。例如，我們可以談論用於生成帳單的計量。同時，我們還可以談論用於跟踪未連接到計費資源的內部消耗的計量。在許多情況下，我們還討論了指標和 SaaS，這些情況可能會混合在此討論中。

為了幫助解決這個問題，讓我們將一些特定的概念與每個術語關聯起來（知道這裡沒有絕對的）。

- 計量 — 此概念雖然具有許多定義，但最適合 SaaS 計費領域。這個想法是，您計量租戶活動或資源消耗，以收集生成帳單所需的數據。
- 指標 — 指標代表您擷取的所有資料，以分析業務、營運和技術領域的趨勢。此資料用於 SaaS 小組中的許多內容和角色。

這種區別並不重要，但有助於簡化我們如何思考計量和指標在 SaaS 環境中的作用。

現在，如果我們將這兩個概念連接到示例中，您可以考慮使用特定的計量事件來檢測應用程序，用於顯示生成帳單所需的數據。這可能是請求的數量，活躍用戶的數量，或者它可以映射到與某個單元相關的消耗（請求，CPU，內存）的一些聚合，這些消耗量與您的客戶有意義。

在 SaaS 環境中，您會從應用程式發佈這些帳單事件，並由 SaaS 系統採用的計費結構擷取和套用這些事件。這可能是第三方計費系統或自定義的東西。

相比之下，指標背後的思維方式是捕捉那些動作、活動、消費模式等，這些動作、活動、消費模式等對於評估不同租戶對系統所施加的健康狀態和營運足跡至關重要。您在此處發布和彙總的指標是由不同角色（操作團隊，產品所有者，架構師等）的需求決定了更多。在這裡，此指標資料會發佈並彙總到一些分析工具中，讓這些不同的使用者建立系統活動的檢視，以分析與其角色最符合其角色的系統層面。產品擁有者可能想要瞭解不同租戶如何使用功能。架構設計人員可能需要檢視來協助他們瞭解租用戶如何使用基礎結構資源，等等。

# B2B 和 B2C 軟體 SaaS

軟件 SaaS 產品是為 B2B 和 B2C 市場創建的。儘管市場和客戶肯定有不同的動態，但 SaaS 的整體原則並不會以某種方式改變每個市場。

例如，如果您查看入職，B2B 和 B2C 客戶可能會有不同的入職體驗。的確，B2C 系統可能會更專注於自助式入職流程（儘管 B2B 系統也可能支持）。

儘管向客戶提供入職的方式可能會有所不同，但入職的基本基礎值大多相同。即使您的 B2B 解決方案依賴於內部入職流程，我們仍然希望該流程盡可能無摩擦和自動化。作為 B2B 並不意味著我們正在逐步改變我們的期望，為我們的客戶提供價值。

# 結論

本文件的目標是概述 SaaS 架構的基本概念，並針對用來表徵 SaaS 模式和策略的模型和術語提供一些說明。希望這將使組織能夠更清楚地了解整體 SaaS 格局。

此處涵蓋的大部分內容都集中在 SaaS 的含義上，重點放在建立可讓您透過統一體驗管理和操作所有 SaaS 租用戶的環境。這與 SaaS 首先是商業模式的核心理念相連。您建立的 SaaS 架構旨在促進這些基礎業務目標。

# 深入閱讀

有許多資源對 SaaS 架構模式進行了更詳細的說明，這些模式符合此處描述的模式。

如需其他資訊，請參閱：

- SaaS 租戶隔離策略 (AWS白皮書)
- SaaS 儲存策略 (AWS白皮書)
- 架構良好的 SaaS 鏡頭 (白皮書) AWS

# 貢獻者

下列個人和組織為本文件作出了貢獻：

- 托德·戈爾丁，SaaS 工廠首席合作夥伴解決方案架構師 AWS

# 文件修訂

如要接收重大變更的通知，請訂閱 RSS 摘要。

| 變更 | 描述 | 日期 |
| --- | --- | --- |
| 初始出版 | 白皮書已發佈。 | 2022 年 8 月 3 日 |

# 注意

客戶有責任對本文件中的資訊進行獨立評估。本文件：(a) 僅供參考，(b) 代表目前的AWS產品供應項目和做法，如有變更，恕不另行通知，且 (c) 不會向其關聯公司、供應商或授權人建立任何承諾或保證。AWS AWS產品或服務係依「原狀」提供，不含任何明示或暗示之擔保、陳述或條件。客戶的責任和責任由AWS協議控制，本文件不屬於與客戶之間AWS的任何協議的一部分，也不會修改。AWS

# AWS 詞彙表

如需最新的 AWS 術語，請參閱《AWS 詞彙表 參考》中的 AWS 詞彙表。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。