AWS 白皮书

SaaS 架构基础知识

SaaS 架构基础知识: AWS 白皮书

Table of Contents

SaaS 架构基础知识

发布日期：2022 年 8 月 3 日（文档修订）

在软件即服务（SaaS）模式中运行业务的范围、目标和性质很难有一个明晰的定义。描述 SaaS 的术语和模式也因来源而有所不同。本文旨在更好地介绍 SaaS 的基本要素，并更清楚地描述在 AWS 上设计和交付 SaaS 系统时所应用的模式、术语和价值体系。更广泛的目标是提供一系列最基本的见解，让客户更清楚地了解他们在准备采用 SaaS 交付模式时应该考虑的选项。

本文的目标读者是刚开始 SaaS 之旅的 SaaS 构建者和架构师，以及希望加深对核心 SaaS 概念理解的经验更丰富的构建者。其中一些信息对于希望熟悉 SaaS 场景的 SaaS 产品拥有者和策略师可能也会有帮助。

简介

软件即服务（SaaS）一词用于描述业务和交付模式。但问题是，并非所有人都理解 SaaS 的含义。

尽管人们对于有关 SaaS 的某些核心问题达成了共识，但对于 SaaS 的含义仍然存在一些困惑。团队对 SaaS 的看法有所不同是很自然的事情。同时，SaaS 的概念和术语不明确可能会给那些探索 SaaS 交付模式的人带来一些困惑。

本文重点介绍用于描述核心 SaaS 概念的术语。针对这些概念达成共识有助于您清楚地了解 SaaS 架构的基本元素，还能让您掌握用于描述 SaaS 架构结构的通用词汇。当您深入研究基于这些主题的其他内容时，这一点尤其有用。

本白皮书回顾了多租户架构的详细信息，探讨了有关 SaaS 含义的基础知识。理想情况下，这还将提供一系列更清晰的术语，使组织能够更快地针对其 SaaS 解决方案的风格和性质进行调整。

您使用 Well-Architected 了吗？

当您在云端构建系统时，AWS Well-Architected Framework 可帮助您了解所做决策的利弊。利用此框架的六个支柱，您可以了解到设计和运行可靠、安全、高效、经济有效且可持续的系统的架构最佳实践。您可以使用 AWS Management Console免费提供的 AWS Well-Architected Tool，回答与每个支柱相关的一组问题，即可根据这些最佳实践检查自己的工作负载。

在 SaaS 剖析中，我们重点介绍在 AWS 上构建软件即服务（SaaS）工作负载的最佳实践。

有关云架构的更多专家指导和最佳实践（参考架构部署、图表和白皮书），请参阅 AWS 架构中心。

SaaS 是一种业务模式

要解释 SaaS 的含义，首先要就一个关键原则达成共识：SaaS 是一种业务模式。这意味着（最重要的是），一系列业务目标会直接推动 SaaS 交付模式的采用。是的，我们会使用技术来实现其中的一些目标，但是 SaaS 就是要建立一种针对一系列特定业务目标的思维方式和模式。

让我们更深入地看一下与采用 SaaS 交付模式相关的一些关键业务目标。

- 敏捷性 — 该术语概括了 SaaS 的更广泛目标。成功的 SaaS 公司建立在这样一种理念之上：他们必须做好准备，不断适应市场、客户和竞争的动态变化。优秀的 SaaS 公司需要能够不断接受新的定价模式、新的细分市场和新的客户需求。
- 运营效率 — SaaS 公司依靠运营效率来提升规模和提高敏捷性。这意味着要建立专注于创建运营足迹的文化和工具，从而推动频繁而快速地发布新特征。这还意味着您能够拥有单一、统一的体验，从而可以集中管理、运营和部署所有客户环境。支持一次性版本和自定义的想法已经不复存在了。SaaS 企业非常重视运营效率，将其作为成功发展和扩展业务能力的核心支柱。
- 顺畅加入 — 作为提高敏捷性和促进发展的一部分，您还必须重视减少租户客户在加入过程中的任何摩擦。这普遍适用于企业对企业（B2B）和企业对客户（B2C）客户。无论您支持哪个细分市场或哪种类型的客户，您仍然需要重点关注为客户实现价值的时间。向以服务为中心的模式转变要求 SaaS 企业关注客户体验的各个方面，还需要特别关注整个加入生命周期的可重复性和效率。
- 创新 — 转向 SaaS 不仅仅是为了满足当前客户的需求；还要建立允许您创新的基本要素。您希望在 SaaS 模式中对客户需求做出反应和响应。但是，您还希望利用这种敏捷性来推动未来的创新，从而为客户开启新的市场、机会和效率。
- 市场反应 — SaaS 摆脱了传统的季度发布和两年计划的概念。借助它的敏捷性，组织能够近乎实时地对市场动态做出反应和响应。组织可以对 SaaS 的结构、技术和文化要素进行投资，从而有机会根据新兴客户和市场动态调整业务战略。
- 增长 — SaaS 是一种以增长为中心的业务策略。围绕敏捷性和效率调整组织的所有动态部分，使 SaaS 组织能够锁定增长模式。这意味着要建立支持并接受您的 SaaS 产品快速采用的机制。

您会注意到，这些项目中的每一项都侧重于业务成果。可以使用各种各样的技术策略和模式来构建 SaaS 系统。然而，这些技术战略并没有给更广泛的商业案例带来改变。

当我们向组织询问他们采用 SaaS 想要实现的目标时，我们总是从这种以业务为中心的讨论开始。技术选择很重要，但必须在这些业务目标的背景下进行选择。例如，如果多租户无法实现敏捷性、运营效率或顺畅加入，就会阻碍您的 SaaS 业务获得成功。

以此为背景，让我们尝试遵照前面概述的原则，更简洁地描述 SaaS 的含义：

SaaS 是一种业务和软件交付模式，使组织能够通过以服务为中心的模式顺畅提供解决方案，从而最大限度地提高为客户和提供商带来的价值。它依靠敏捷性和运营效率作为促进增长、扩大覆盖范围和推动创新的业务战略的支柱。

您应该了解业务目标之间的一致性，以及组织如何依赖这些业务目标为所有客户提供共享体验。迁移到 SaaS 在很大程度上意味着摆脱传统软件模式中可能包含的一次性定制。通常来说，任何为客户提供专业化服务的努力都会与我们试图通过 SaaS 实现的核心价值观背道而驰。

您提供的是服务，不是产品

采用“服务”模式不仅仅是营销或术语。在服务思维中，您会发现自己远离了传统的基于产品的开发方法。虽然特征和功能对每个产品都很重要，但 SaaS 更加重视客户对您的服务的体验。

这意味着什么？ 在以服务为中心的模式中，您可以更多地考虑如何把客户加入您的服务、如何让他们更快地实现价值以及如何更快地推出能够满足客户需求的特征。客户看不到与您服务的构建、运营和管理方式相关的细节。

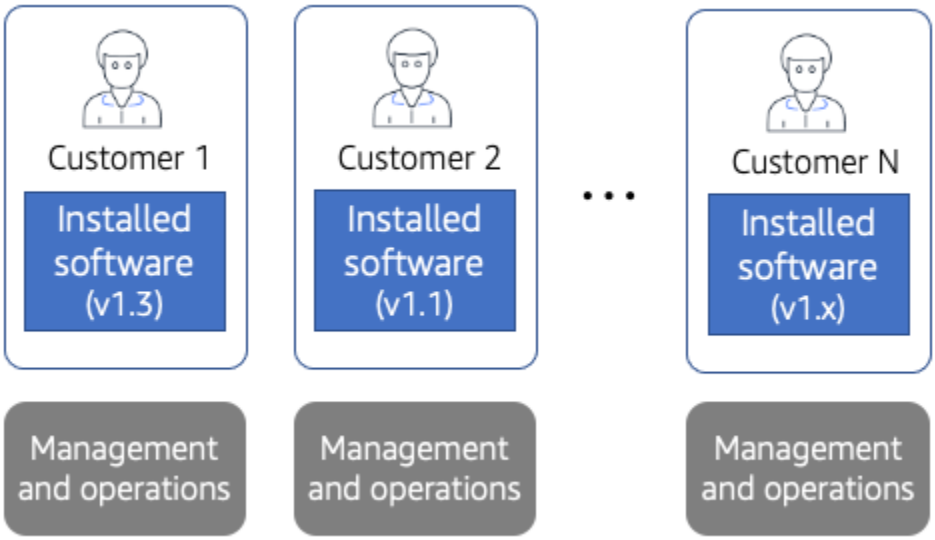
在这种模式下，我们像考虑可能使用的任何其他服务一样考虑此 SaaS 服务。如果我们去餐厅用餐，我们当然会关心食物，但我们也会关心服务。服务员到达您餐桌的速度有多快、他们多久给您添一次水、食物上桌的速度有多快等，都是衡量服务体验的标准。我们正应该采用这种思维方式和价值体系塑造我们对构建 SaaS 服务的看法。

这种 XX 即服务模式应该对您建立团队和服务的方式产生重大影响。现在，您的待办工作将使这些体验属性处于与特征和功能相同或更高的地位。企业还会将这些视为您的 SaaS 产品长期增长和成功的基础。

最初的动机

要理解 SaaS，让我们从创建 SaaS 业务时想要实现的目标这个简单的概念开始。最好首先了解下传统（非 SaaS）软件是如何创建、管理和运营的。

下图提供了有关几家供应商如何打包和交付其解决方案的概念视图。



打包和交付软件解决方案的传统模式

在此图中，我们介绍了一系列客户环境。这些客户代表已购买供应商软件的不同公司或实体。每一位客户实际上都是在独立的环境中运行的，并且他们已经在环境中安装了软件提供商的产品。

在此模式下，每个客户的安装都被视为专用于该客户的独立环境。这意味着客户将自己视为这些环境的拥有者，他们可能会要求一次性定制或独特的配置来支持他们的需求。

同时，这种想法通常会带来这样的结果：客户将控制他们正在运行的产品版本。这是由多种原因造成的。客户可能对新特征心存顾虑，或者担心采用新版本会带来中断。

您可以想象这种动态会如何影响软件提供商的运营足迹。您允许客户拥有的一次性环境越多，管理、更新和支持每位客户的不同配置就越困难。

这种对一次性环境的需求通常要求组织创建专门的团队，为每位客户提供单独的管理和运营体验。虽然可能会在客户之间共享其中一些资源，但这种模式通常会让每位新加入客户的支出越来越多。

让每位客户在自己的环境（云端或本地）中运行解决方案也会影响成本。虽然您可以尝试扩展这些环境，但这种扩展只能局限于单个客户的活动。从本质上讲，您的成本优化仅可以在单个客户环境范围内实现。这也意味着您可能需要单独的扩展策略来适应客户之间的活动差异。

最初，一些软件企业会将这种模式视为一种强大的结构。他们把提供一次性的定制功能作为销售工具，允许新客户提出特定于其环境的要求。尽管客户数量和业务增长不多，但这种模式似乎完全可持续。

但是，随着公司开始取得更大的成功，这种模式的限制逐渐带来了真正的挑战。例如，想象一下，您的业务显著增长、达到峰值，这会让您迅速增加大量新客户。这种增长将会带来运营开销、管理复杂性、成本和许多其他问题。

最终，这种模式的整体开销和影响可能会开始从根本上破坏软件业务的成功。第一个痛点可能是运营效率。与维护客户相关的人员配备和成本逐渐增加，进而开始侵蚀业务的利润率。

但是，运营问题还只是挑战的一部分。真正的问题是，随着这种模式的扩展，它直接开始影响企业发布新特征和跟上市场步伐的能力。如果每个客户都有自己的环境，提供商在尝试将新功能引入客户的系统时，就必须平衡大量的更新、迁移和客户需求。

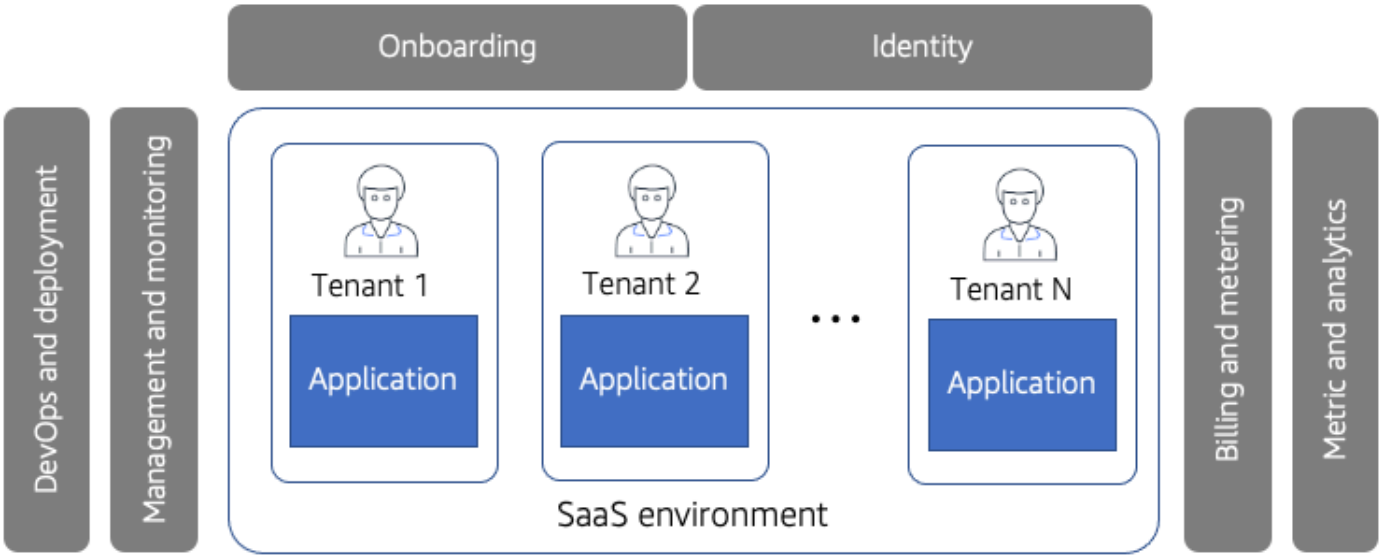
这通常会导致更长、更复杂的发布周期，因此往往会减少每年发布的数量。更重要的是，由于这种复杂性，团队在向客户发布每个新特征之前，往往会投入更多时间对其进行分析。团队开始更多地关注验证新特征，而不是关注交付速度。发布新特征的开销变得如此之大，以至于团队更加关注测试机制，而不是关注推动其产品创新的新特征。

在这种更慢、更谨慎的模式下，团队的周期往往会很长。因此，一个想法从诞生到在客户身上投入使用之间的时间会越来越大。总的来说，这可能会阻碍您对市场动态和竞争压力做出及时的反应。

转向统一的体验

为了解决这种传统软件困境的需求，组织转向了一种模式，该模式允许他们创建单一、统一的体验，以便对客户进行集中管理和运营。

下面提供了一个环境的概念视图。在该环境中，所有客户都通过共享模式进行管理、加入、计费和运营。



环境的概念视图，在该环境中，所有客户都通过共享模式进行管理、加入、计费和运营

乍一看，这似乎与之前的模式没有太大区别。但是，随着我们进一步深入研究，您会发现这两种方法存在根本上的显著差异。

首先，您会注意到客户环境已重命名为租户。这种租户的概念是 SaaS 的基础。其基本思想是，您只有一个 SaaS 环境，每个客户都被视为该环境的一个租户，使用他们所需的资源。租户可以是拥有许多用户的公司，也可以直接关联到个人用户。

为了更好地理解租户的概念，可以想一下公寓或商业建筑的概念。这些建筑物中的空间会出租给各个租户。租户需要使用建筑物中的某些共享资源（水、电等），并根据他们的使用量支付费用。

SaaS 租户也遵循类似的模式。您拥有 SaaS 环境的基础设施，以及使用该环境基础设施的租户。每个租户消耗的资源量可能有所不同。您对这些租户进行集中管理、计费和运营。

如果您再来看一下这个图，就会更生动、直观地了解租赁的概念。在这里，租户不再拥有自己的环境。相反，所有租户都在一个共同的 SaaS 环境中进行安置和管理。

该图还包括基于 SaaS 环境的一系列共享服务。这些服务面向您的 SaaS 环境中的所有租户。这意味着，例如，加入和身份验证由此环境中的所有租户共享。管理、运营、部署、计费和指标也是如此。

将一系列统一的服务普遍应用于所有租户，这种想法是 SaaS 的基础。通过分享这些概念，您可以解决与上述传统模式相关的许多难题。

此图中的另一个关键而微妙的元素是，该环境中的所有租户都运行相同版本的应用程序。分别为每个客户运行不同版本的想法已经不复存在了。让所有租户运行相同的版本是 SaaS 环境区别于其他环境的基本属性之一。

通过让所有客户运行相同版本的产品，您将不用再去面对传统的软件安装模式带来的众多挑战。在统一模式下，可以通过单一的共享流程将新特征部署到所有租户。

借助这种方法，您就可以使用单一的操作窗格来管理和运营所有租户。这使您能够通过集中的运营体验来管理和监控租户，从而在不增加增量运营开销的情况下添加新租户。这是 SaaS 价值主张的核心部分，它使团队能够减少运营开销，还能从整体上提高组织的灵活性。

想象一下，在这种模式下添加 100 或 1000 个新客户意味着什么。您可以将这种增长视为其所代表的机会，而不必担心这些新客户会如何侵蚀利润并增加复杂性。

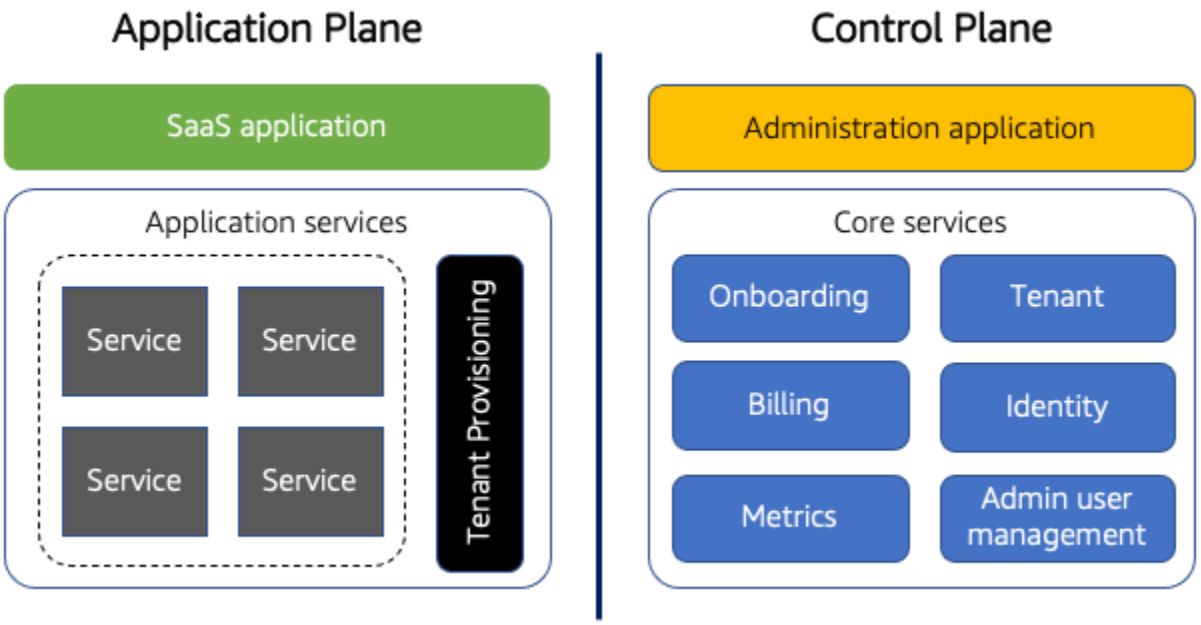
通常，SaaS 重点关注如何实施此模式中间的应用程序。企业希望将重点放在如何存储数据、如何共享资源等方面上。然而实际上，虽然这些细节确实很重要，但您的应用程序可以通过很多方法构建出来，并且仍然可以作为 SaaS 解决方案呈现给客户。

更重要的是要实现更广泛的目标，即在租户环境中提供单一、统一的体验。拥有这种共享经验可以让您推动与 SaaS 业务总体目标相关的增长、敏捷性和运营效率。

控制平面与应用平面

上图提供了核心 SaaS 架构概念的概念视图。现在，让我们深入探讨这个问题，以便更好地解释您的 SaaS 环境如何分解成不同的层。更清楚地了解 SaaS 概念之间的界限将更容易描述 SaaS 解决方案的动态部分。

下图将您的 SaaS 环境划分为两个不同的平面。右侧是控制平面。控制平面包括用于加入、身份验证、管理、操作和分析多租户环境的所有功能和服务。



控制平面与应用平面

该控制平面是任何多租户 SaaS 模式的基础。每个 SaaS 解决方案（无论采用何种应用程序部署和隔离方案）都必须包括那些让您能够通过单一、统一的体验管理和运营租户的服务。

在控制平面内，我们进一步将其分解为两个不同的元素。此处的核心服务表示的是一系列用于协调您的多租户体验的服务。我们列举了一些通常属于核心服务的常见示例，不过每个 SaaS 解决方案的核心服务可能会有所不同。

您还会注意到，我们展示了一个单独的管理应用程序。这表示的是 SaaS 提供商可能用来管理其多租户环境的应用程序（Web 应用程序、命令行界面或 API）。

必须注意的是，控制平面及其服务实际上并不是多租户的。该功能不可以提供您的 SaaS 应用程序的实际功能属性（必须是多租户的才可以）。例如，如果您深入研究任何一个核心服务，您都找不到租户隔离和属于多租户应用程序功能的其他结构。这些服务面向所有租户。

示意图的左侧展示了 SaaS 环境的应用平面。这是应用程序的多租户功能所在的位置。图中显示的内容要尽量模糊一些，因为每个解决方案都可以根据您的域需求、技术足迹等以不同方式进行部署和分解。

应用程序域分为两个元素。SaaS 应用程序代表您的解决方案的租户体验/应用程序。这是租户用来与 SaaS 应用程序交互的层面。然后这里还有后端服务，其代表的是 SaaS 解决方案的业务逻辑和功能元素。这些可能是微服务，也可能打包了其他应用程序服务。

您还会注意到，我们已经停止了预置。这样做是为了强调这样一个事实：在租户加入期间为租户预置的任何资源都将成为该应用程序域的一部分。有些人可能会争辩说这属于控制层面。但是，我们已将其置于应用程序域中，因为它必须预置和配置的资源会更直接地连接到在应用程序平面中创建和配置的服务。

将其分解为不同的平面有助于我们更轻松地思考 SaaS 架构的整体情况。更重要的是，它突显了对一系列完全不在应用程序功能范围内的服务的需求。

核心服务

前面提到的控制平面提到了一系列核心服务，这些服务代表了用于加入、管理和运营 SaaS 环境的典型服务。进一步强调其中一些服务的作用，以突出它们在 SaaS 环境中的范围和目的，可能会有所帮助。以下是这些服务的简要介绍：

- 加入 — 每个 SaaS 解决方案都必须提供一种顺畅的机制，用于将新租户加入到您的 SaaS 环境。这可以是一个自助注册页面，也可以是一种内部管理的体验。无论采用哪种方式，SaaS 解决方案都应尽其所能消除这种体验中的内部和外部摩擦，并确保此过程的稳定性、效率和可重复性。它在支持 SaaS 业务的增长和规模方面发挥着至关重要的作用。通常，此服务会协调其他服务，以创建用户、租户、隔离策略、预置和按租户分配的资源。
- 租户 — 租户服务提供了一种集中管理租户策略、属性和状态的方法。关键之处在于租户不是个人用户。实际上，一个租户可能与多个用户相关联。
- 身份 — SaaS 系统需要一种明确的方法来将用户与租户关联起来，从而为其解决方案的身份验证和授权体验带来租户上下文。这会影响加入体验和用户配置文件的整体管理。
- 计费 — 作为采用 SaaS 的一部分，组织通常会使用新的计费模式。他们可能还会探索与第三方计费提供商的集成。这项核心服务主要侧重于支持新租户的加入，以及收集用于为租户生成账单的消费和活动数据。
- 指标 — SaaS 团队在很大程度上依赖于他们捕获和分析丰富的指标数据的能力，从而能够更清楚地了解租户如何使用系统、如何消耗资源以及如何与系统互动。这些数据用于制定运营、产品和业务战略。
- 管理员用户管理 — SaaS 系统必须同时支持租户用户和管理员用户。管理员用户代表 SaaS 提供商的管理员。他们将登录您的运营体验，以监控和管理您的 SaaS 环境。

重新定义多租户

多租户和 SaaS 这两个术语通常紧密相连。在某些情况下，组织将 SaaS 和多租户混为一谈。虽然这看起来可能很自然，但将 SaaS 等同于多租户往往会导致团队从纯粹的技术角度看待 SaaS。而实际上，与其说 SaaS 是一种架构策略，倒不如说它是一种业务模式。

为了更好地理解这个概念，让我们从传统的多租户视图开始。在这个纯粹以基础设施为中心的视图中，多租户用于描述租户如何共享资源以提高敏捷性和成本效益。

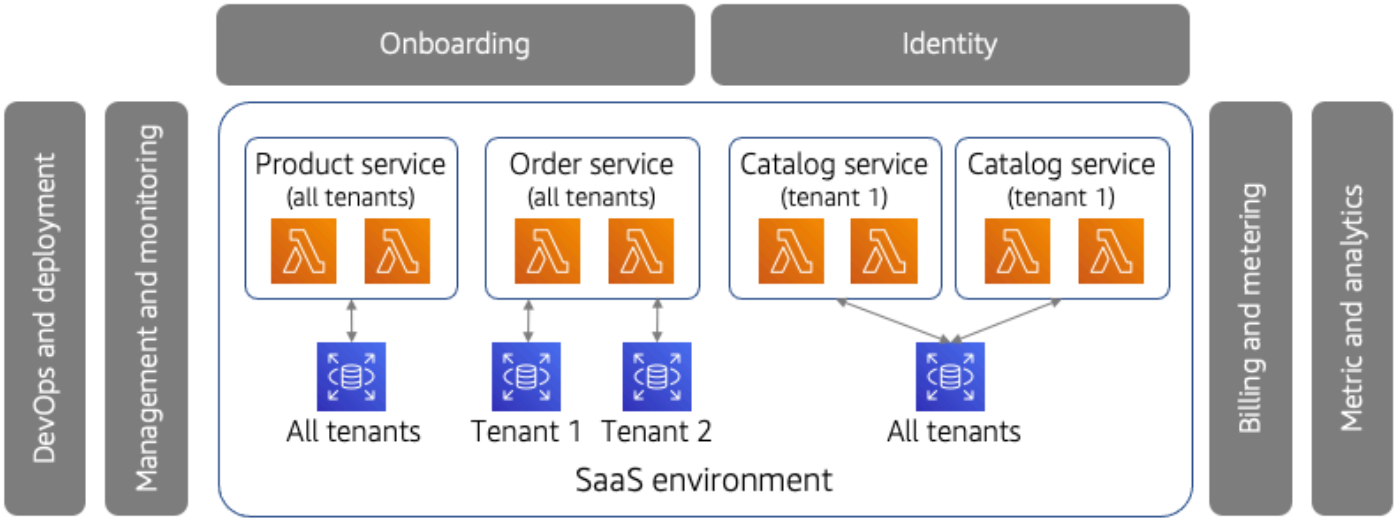
例如，假设您有一个微服务或 Amazon Elastic Compute Cloud (Amazon EC2) 实例，该实例由您的 SaaS 系统中的多个租户使用。该服务将被视为在多租户模式下运行，因为租户共享使用运行该服务的基础架构。

这个定义的问题在于，它过于直接地将多租户的技术概念与 SaaS 联系起来。它假设 SaaS 的决定性特征是它必须具有共享的多租户基础设施。当我们研究在不同环境中实现 SaaS 的各种方式时，关于 SaaS 的这种观点就不再适用。

下面提供了 SaaS 系统的视图，揭示了我们在解释“多租户”一词时遇到的一些难题。

在这里，您将看到前面描述的传统 SaaS 模式，它围绕共享服务提供了一系列应用程序服务，以便您能够集中管理和运营租户。

新增的是我们所包含的微服务。该图包括三个微服务示例：产品、订单和目录。如果您仔细观察每种服务的租赁模式，就会发现它们采用的租赁模式略有不同。



SaaS 和多租户

产品服务与所有租户共享其所有资源（计算和存储）。这与多租户的传统定义一致。但是，如果您查看订单服务，就会发现它使用的是共享计算资源，但为每个租户提供独立的存储空间。

目录服务增加了另一种变化，该服务为每个租户提供独立的计算（为每个租户单独部署微服务），但面向所有租户共享存储。

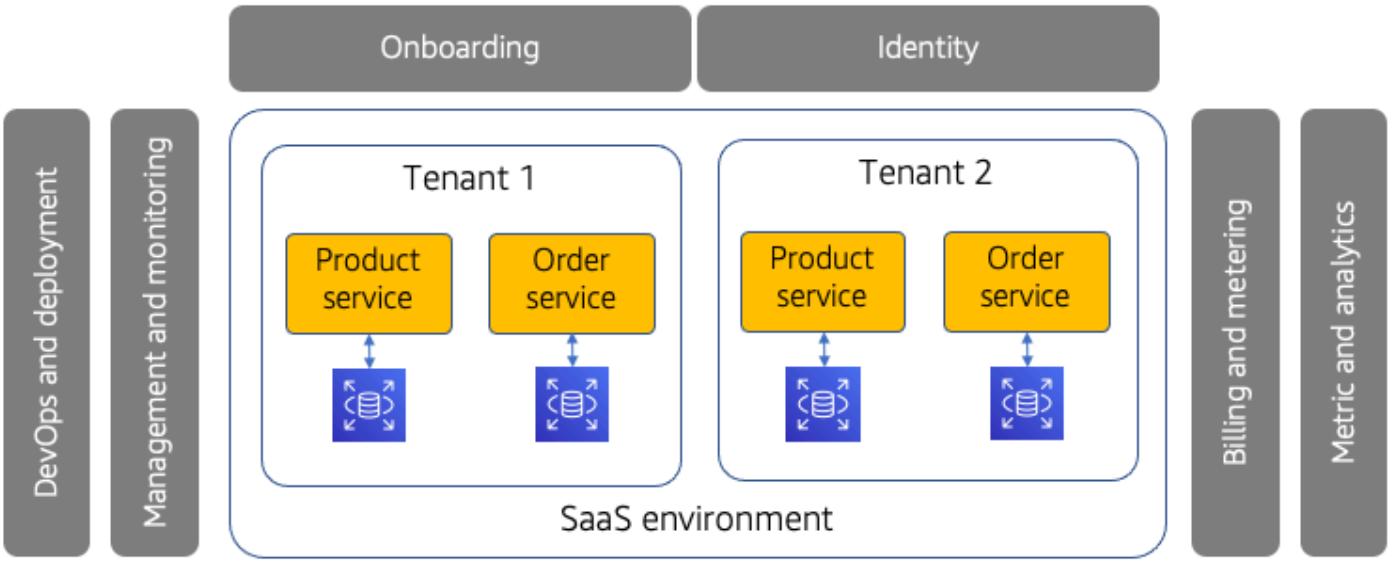
这种性质的变化在 SaaS 环境中十分常见。近邻干扰、分层模式、隔离需求等因素都可能会导致您有选择性地共享或孤立 SaaS 解决方案的某些部分。

鉴于这些差异以及许多其他可能性，弄清楚如何使用多租户一词来描述这种环境就变得更加困难。总体来说，就客户而言，这是一个多租户环境。但是，如果我们使用最具技术性的定义，则该环境的有些部分是多租户的，而有些则不是。

这就是为什么有必要停止使用多租户一词来描述 SaaS 环境的原因。相反，我们可以讨论如何在您的应用程序中实现多租户，但要避免使用这个词来将解决方案描述为 SaaS。如果要使用多租户一词，则将整个 SaaS 环境描述为多租户更有意义，因为架构的某些部分可能是共享的，而有些则可能不是。总体而言，您仍是在多租户模式下操作和管理此环境。

极端案例

为了更好地突出这种租赁的概念，让我们来看一个租户没有共享任何资源的 SaaS 模式。下图提供了某些 SaaS 提供商采用的 SaaS 环境的示例。



每个租户的堆栈

在此图中，您将看到我们仍然有针对这些租户的共同环境。但是，每个租户都部署了一系列专用的资源。在此模式下，租户不共享任何资源。

这个例子挑战了多租户的含义。即使没有共享任何资源，这也能算是多租户环境吗？ 使用此系统的租户对具有共享资源的 SaaS 环境抱有同样的期望。事实上，他们可能不知道自己的资源在 SaaS 环境下是如何部署的。

尽管这些租户在孤立的基础设施中运行，但系统仍会对他们进行集中管理和运营。他们共享统一的加入、身份、指标、计费和运营体验。此外，当发布新版本时，也会部署到所有租户。为了实现这一点，您不能允许为单个租户进行一次性定制。

这个极端的例子为测试多租户 SaaS 的概念提供了一个很好的模式。尽管可能无法实现共享基础设施的所有效率，但它是一个完全有效的多租户 SaaS 环境。对于某些客户来说，他们的域可能决定他们的部分或全部客户在这种模式下运行。这并不意味着它们不是 SaaS。如果他们使用这些共享服务，并且所有租户都运行相同的版本，则仍然符合 SaaS 的基本原则。

鉴于这些参数和 SaaS 的更广泛定义，您可以看到需要改进多租户这一术语的使用。将任何集中管理和运营的 SaaS 系统都称为多租户更有意义。然后，您可以参考更精细的术语来描述如何在 SaaS 解决方案的实施过程中共享或专用资源。

删除“单租户”这一术语

由于使用了多租户这一术语，人们自然会想使用单租户一词来描述 SaaS 环境。但是，鉴于前面概述的背景，单租户一词会造成混乱。

上图是单租户环境吗？ 虽然从技术上讲，每个租户都有自己的堆栈，但系统仍以多租户模式运营和管理这些租户。这就是为什么通常避免使用单租户这一术语的原因。相反，所有环境都被描述为“多租户”，因为它们只是在实现租赁的过程中发生了某些变化，其中部分或全部资源是共享或专用的。

“孤岛”和“池”简介

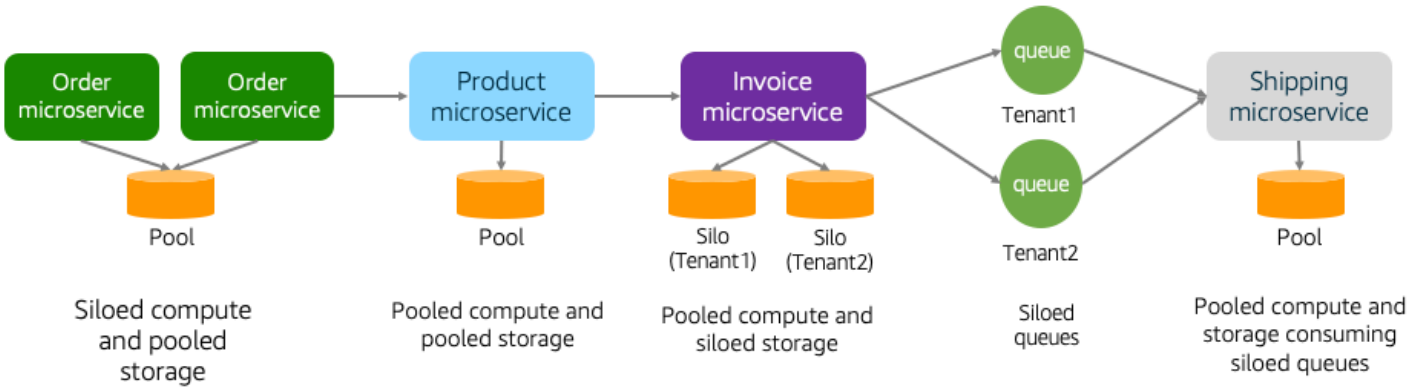
考虑到所有这些模式的变化，并考虑到有关多租户一词的挑战，我们引入了一些术语，帮助我们更准确地捕获和描述构建 SaaS 时使用的不同模式。

我们用来描述 SaaS 环境中资源使用的两个术语是孤岛和池。这些术语允许我们标记 SaaS 环境的本质，使用多租户作为总体描述，可以应用于任意数量的底层模型。

在最基本的层面上，孤岛这一术语旨在描述资源专用于指定租户的场景。相反，池模式用于描述租户共享资源的场景。

当我们研究如何使用“孤岛”和“池”这两个术语时，重要的是要清楚“孤岛”和“池”不是非此即彼的概念。“孤岛”和“池”可以应用于整个租户的资源堆栈，也可以有选择地应用于整个 SaaS 环境的某些部分。因此，如果我们说某些资源使用了孤岛模式，也并不意味着该环境中的所有资源都是孤立的。我们使用池化一词时也是如此。

下图举例说明了如何在 SaaS 环境中更精细地使用孤岛模式和池模式：



孤岛模式和池模式

此图包括一系列示例，旨在说明孤岛模式和池模式更具针对性的性质。如果您按照从左到右的顺序观察此图，就会发现我们从订单微服务开始。该微服务具有孤立计算和池化存储。它与具有池化计算和池化存储的产品服务进行交互。

然后，产品服务与具有池化计算和孤立存储的发票微服务进行交互。该服务通过队列将消息发送到配送服务。队列部署在孤岛模式中。

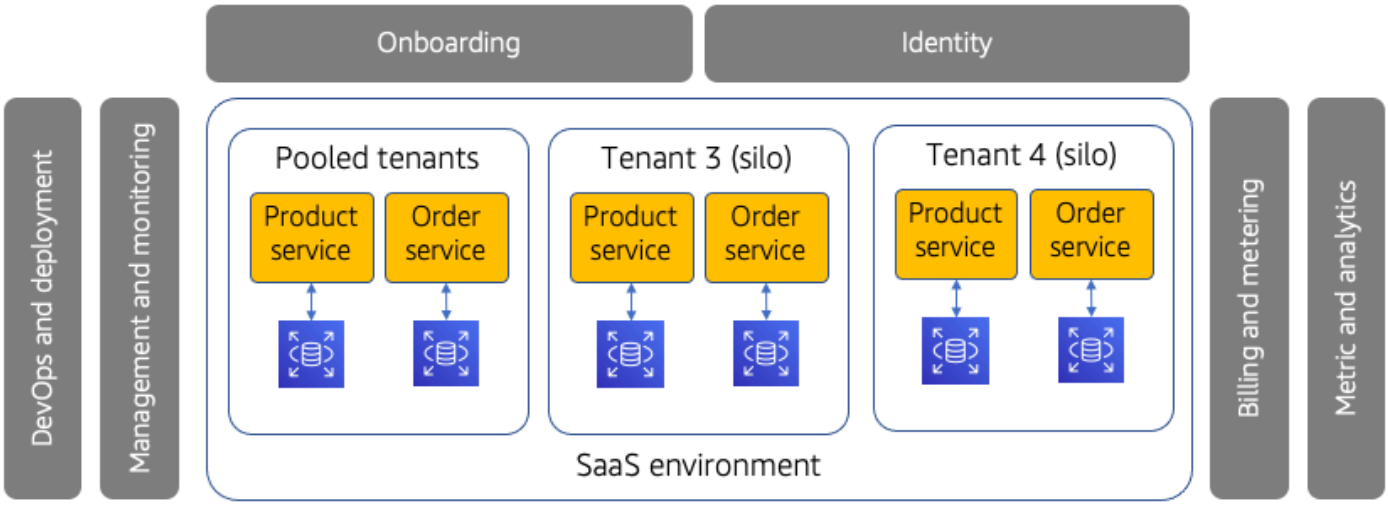
最后，配送微服务从孤立队列中获取消息。它使用池化计算和存储。

虽然这可能看起来有点复杂，但目的是突出孤岛和池这两个概念的精细本质。当您想要设计和构建 SaaS 解决方案时，预计您会根据自己的域和客户的需求做出使用孤岛或池模式的决策。

近邻干扰、隔离、分层以及许多其他原因可能会影响您选择应用孤岛模式或池模式的方式和时间。

全堆栈孤岛和池

孤岛和池也可以用来描述整个 SaaS 堆栈。在这种方法中，租户的所有资源都以专用或共享的方式部署。下图提供了一个示例，说明可能如何在 SaaS 环境中实现此操作。



全堆栈孤岛和池模式

在此图中，您会看到全堆栈租户部署有三种不同的模式。首先，您会看到有一个全堆栈池环境。此池中的租户共享所有资源（计算、存储等）。

另外两个堆栈代表全堆栈孤岛租户环境。在本例中，租户 3 和租户 4 分别拥有自己的专用堆栈，不与其他租户共享任何资源。

在同一 SaaS 环境中混合使用孤岛模式和池模式的情况也比较常见。例如，想象一下，您有一些基本级别的租户，他们为使用您的系统支付了适中的价格。这些租户被安置在池化环境中。

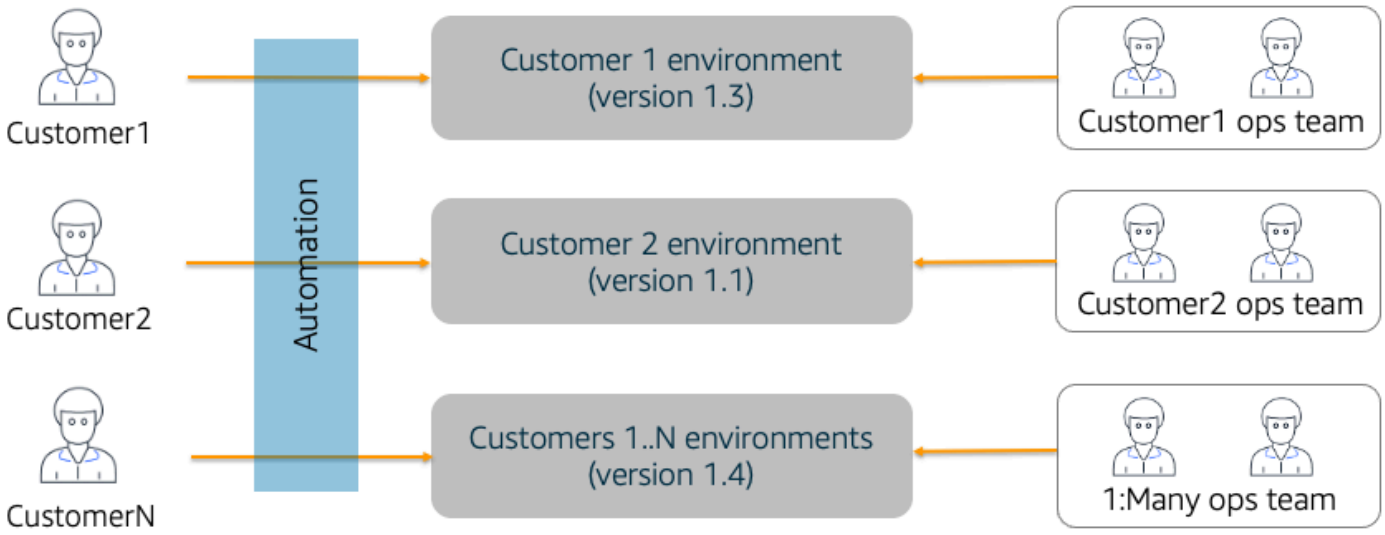
同时，您可能还有一些高级租户，他们愿意支付更多费用以便能够在孤岛中运行。这些客户使用单独的堆栈进行部署（如图所示）。

在这个模式中，即使您可能已经允许租户在他们自己的全堆栈孤岛中运行，这些孤岛也不允许对这些租户进行任何一次性的更改或自定义，这一点至关重要。在所有方面，每一个堆栈都应该使用相同版本的软件运行相同的堆栈配置。发布新版本时，会将其部署到池化租户环境和每个孤立环境中。

SaaS 与托管服务提供商（MSP）

有关 SaaS 和托管服务提供商（MSP）模式之间的界限也存在一些混乱。MSP 模式可能有一些与 SaaS 模式相似的目标。

但是，如果您深入研究 MSP，就会发现 MSP 和 SaaS 实际上是不同的。下图提供了 MSP 环境的概念视图。



托管服务提供商（MSP）模式

此图代表了 MSP 模式的一种方法。左边是在 MSP 模式下运行的客户。通常，这里提到的方法是使用任何可用的自动化来预置每个客户环境，然后为该客户安装软件。

右边是 MSP 为支持这些客户环境而提供的运营足迹的近似值。

需要注意的是，MSP 通常会安装和管理指定客户想要运行的产品版本。所有客户可以运行相同的版本，但 MSP 模式通常不要求这样做。

一般策略是通过拥有这些环境的安装和管理来简化软件提供商的生命周期。尽管这简化了提供商的生命周期，但它与 SaaS 产品所必需的价值观和思维方式没有直接的关联。

它侧重的是减轻管理责任。采取这一举措并不意味着让所有客户都在同一版本上运行，同时还能获得单一、统一的管理和运营体验。相反，MSP 通常允许使用不同的版本，并且将每个环境都视为独立操作的环境。

当然，在某些领域，MSP 开始的时候可能会与 SaaS 重叠。如果 MSP 本质上要求所有客户运行相同的版本，而且 MSP 能够通过一种体验对所有租户集中加入、管理、运营和计费，那可能开始的时候更像是 SaaS 而不是 MSP。

更广泛的主题是，自动化环境安装并不等同于拥有 SaaS 环境。只有当您加入了前面讨论过的所有其他注意事项时，这才更能代表真正的 SaaS 模式。

如果我们回过头来看一下这个案例的技术和运营方面，MSP 和 SaaS 之间的界限就会变得更加明显。通常来说，作为一家 SaaS 企业，您的产品成功与否取决于您是否有能力深入参与体验的所有动态部分。

这通常意味着要了解加入体验、了解运营事件如何影响租户、跟踪关键指标和分析，并与客户保持密切联系。在将其移交给其他人的 MSP 模式中，您最终可能会偏离 SaaS 业务运营的核心关键细节。

SaaS 迁移

许多采用 SaaS 的提供商都通过传统的软件安装模式（如前所述）迁移到 SaaS。对于这些提供商来说，在 SaaS 的核心原则上保持一致性尤为重要。

在这里，人们可能会对迁移到 SaaS 模式的含义感到困惑。例如，有些人将迁移到云视为迁移到 SaaS。有些人则认为在安装和预置过程中增加自动化就是实现了迁移。

公平地说，每个组织可能从不同的位置开始，需要考虑不同的遗留问题，并且可能面临不同的市场和竞争压力。这意味着每次迁移看起来都会有所不同。

尽管迁移途径可以不同，但在某些领域却并没有遵循迁移策略的核心原则。在概念和原则上保持良好的一致性可以对 SaaS 迁移的整体成功产生重大影响。

基于前面所述的概念，应该清楚的是，向 SaaS 的迁移应该从了解业务战略和目标开始。但在迫切需要尽快迁移到 SaaS 的迁移环境中，这一点可能会被忽视。

在这种模式下，组织通常主要将迁移视为一项技术活动。实际上，每次 SaaS 迁移都应该从清晰地了解目标客户、服务体验、运营目标等开始。更明确地关注您需要什么样的 SaaS 业务，这将对您将解决方案迁移到 SaaS 的形式、优先级和路径产生深远影响。

从迁移一开始就要有这个清晰的愿景，这将为在迁移到 SaaS 的过程中如何迁移技术和业务奠定基础。当您开始迁移时，就要把注意力放在那些最能为您指明前进方向的问题上。

下表显示了侧重技术和侧重业务的迁移思维方式的对比。

表 1 — 侧重技术的迁移与侧重业务的迁移

侧重技术的思维方式	侧重业务的思维方式
我们如何隔离租户数据？	SaaS 如何帮助我们发展业务？
我们如何将用户与租户联系起来？	我们的目标是哪些细分市场？
我们如何避免近邻干扰？	这些细分市场的规模和概况如何？
我们如何进行 A/B 测试？	我们需要支持哪些层级？
我们如何根据租户负载进行扩展？	我们的目标是怎样的服务体验？
我们应该使用哪个计费提供商？	我们的定价和包装策略是什么？

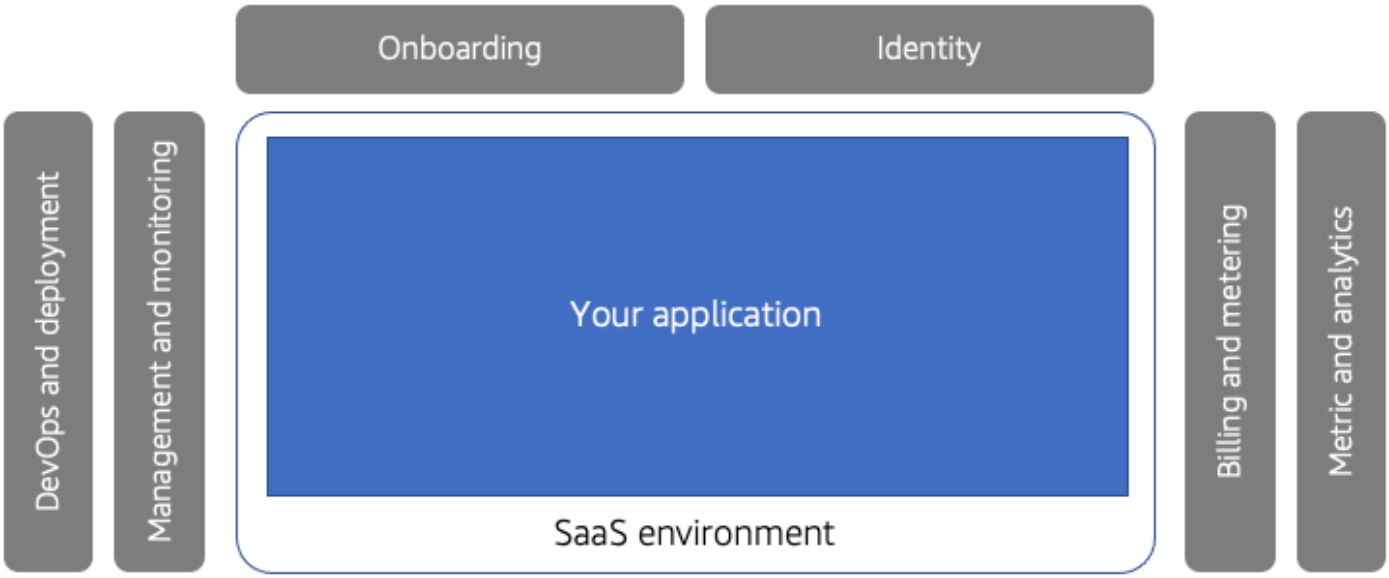
左边是侧重技术的迁移思维方式的示例。工程团队专注于努力获得对任何 SaaS 架构都很重要的经典多租户话题。

但问题是，左边许多问题的答案通常会直接受到右边问题答案的影响。任何关注迁移的人都应该已经很清楚这一点。然而，现实是，许多组织从一开始就将追求运营和成本效率作为第一步，认为业务部分能自行解决。

在这种迁移策略下，人们可能会对如何改进原有环境以适应 SaaS 模式感到困惑。在这个领域，迁移到 SaaS 的选择也很多。但是，对于任何迁移，我们通常都提倡要有一个共同的价值体系。

在之前对 SaaS 原则的讨论中，我们概述了用于描述 SaaS 环境的不同模式和术语。涵盖所有这些解决方案的共同主题是：围绕应用程序提供共享服务。身份识别、加入、指标、账单 — 这些都被称为任何 SaaS 环境的核心要素。

现在，当我们研究迁移时，您会发现这些相同的共享服务在任何迁移案例中都起着关键作用。下图提供了迁移场景的概念视图。



迁移到 SaaS

此图展示了任何迁移路径的目标体验。它包括前面描述的所有共享服务。中间是您的应用程序的占位符。

关键思想是，您可以在此环境的中间放置任意数量的应用程序模型。迁移的第一步可能是让每个租户在自己的孤岛中运行。或者，您可能有一些混合架构，在这些架构中，元素是孤立的，而其他功能则是通过一系列现代化的微服务来实现的。

根据原有环境的性质、市场需求、成本考虑等，您起初对应用程序进行现代化改造的程度会有所不同。相同之处是引入了这些共享服务。

任何 SaaS 迁移都需要支持这些基础共享服务，以使您的业务能够在 SaaS 模式下运营。例如，应用程序架构的所有变化都需要 SaaS 身份。您需要具备租户感知能力的运营来管理和监控您的 SaaS 解决方案。

在迁移时优先部署这些共享服务，这样即使底层应用程序仍在各个租户的全堆栈孤岛中运行，您也可以向客户呈现 SaaS 体验。

总体目标是让您的应用程序在 SaaS 模式下运行。然后，您可以将更多精力放在应用程序的进一步现代化和完善上。借助这种方法，您能以更快的速度迁移业务的其他部分（营销、销售、支持等）。更重要的是，您能够开始参与和收集客户反馈，并使用这些反馈持续构建现代化的环境。

值得注意的是，您提供的共享服务可能不包括您最终需要的所有特征或机制。主要目标是创建迁移开始时所需的共享机制。这使您可以专注于系统中对应用程序架构的发展和运营演变至关重要的元素。

SaaS 身份

SaaS 为应用程序的身份验证模式添加了新的注意事项。由于每个用户都经过了身份验证，因此必须将他们与特定的租户上下文关联起来。此租户上下文提供了有关您的租户的基本信息，可以在整个 SaaS 环境中使用。

这种租户与用户之间的绑定通常被称为应用程序的 SaaS 身份。由于每个用户都会经过身份验证，因此您的身份提供商通常会生成一个包含用户身份和租户身份的令牌。

将租户与用户相连接是 SaaS 架构的一个基本方面，具有许多后续影响。来自此身份验证流程的令牌进入应用程序的微服务，并用于创建租户感知日志、记录指标、计量计费、强制实施租户隔离等。

必须避免依赖于将用户单独映射到租户的独立机制的场景。这可能会破坏系统的安全性，并且通常会在架构中造成瓶颈。

租户隔离

您迁移到多租户模式的客户越多，他们就越担心一个租户是否有可能访问另一个租户的资源。SaaS 系统包括明确的机制，可确保每个租户的资源(即使它们运行在共享基础设施上)都是隔离的。

这就是我们所说的租户隔离。租户隔离背后的想法是，您的 SaaS 架构引入了严格控制资源访问权限的结构，并阻止任何访问其他租户资源的尝试。

请注意，租户隔离与一般安全机制是分开的。您的系统将支持身份验证和授权；但是，租户用户经过身份验证并不意味着您的系统已经实现了隔离。隔离与应用程序中可能包含的基本身份验证和授权分开应用。

为了更好地理解这一点，假设您使用了身份提供商来验证对您的 SaaS 系统的访问权限。这种身份验证体验提供的令牌还可能包含有关用户角色的信息，这些信息可用于控制该用户对特定应用程序功能的访问权限。这些结构提供了安全性，但不能提供隔离。事实上，系统可能会对用户进行了身份验证和授权，但其仍然可以访问其他租户的资源。任何有关身份验证和授权的内容都不一定会阻止这种访问。

租户隔离重点关注使用租户上下文来限制对资源的访问。它评估当前租户的上下文，并使用该上下文来确定该租户可以访问哪些资源。它对与该租户关联的所有用户应用此隔离。

当我们研究如何在所有不同的 SaaS 架构模式中实现租户隔离时，这变得更具挑战性。在某些情况下，可以通过将整个资源堆栈专用于租户来实现隔离，其中网络（或更粗粒度的）策略会阻止跨租户访问。在其他场景下，您可能拥有池化资源（Amazon DynamoDB 表中的项目），这些资源需要更精细的策略来控制对资源的访问权限。

访问租户资源的任何尝试都应该限定在属于该租户的资源范围内。SaaS 开发人员和架构师的职责是，确定结合使用哪些工具和技术可以支持您的特定应用程序的隔离要求。

数据分区

数据分区用于描述在多租户环境中表示数据的不同策略。该术语用途广泛，涵盖一系列不同的方法和模式，这些方法和模式可用于将不同的数据结构与单个租户关联起来。

请注意，人们往往倾向于认为数据分区和租户隔离是可以互换的。其实这两个概念并不完全相同。当我们提及数据分区时，我们谈论的是如何存储单个租户的租户数据。对数据进行分区并不能确保数据是隔离的。仍必须单独应用隔离，以确保一个租户无法访问另一个租户的资源。

每种 AWS 存储技术都为数据分区策略带来了一系列特有的注意事项。例如，在 Amazon DynamoDB 中隔离数据与使用 Amazon Relational Database Service (Amazon RDS) 隔离数据截然不同。

通常，在考虑数据分区时，首先要考虑数据是孤立的还是池化的。在孤岛模式中，每个租户都有不同的存储结构，没有混合数据。对于池化分区，系统根据租户标识符对数据进行混合和分区，租户标识符决定了哪些数据与每个租户相关联。

例如，在 Amazon DynamoDB 中，孤岛模式为每个租户使用单独的表。Amazon DynamoDB 中的池化数据是通过将租户标识符存储在管理所有租户数据的每个 Amazon DynamoDB 表的分区键中来实现的。

您可以想象的到，这可能会因 AWS 服务范围而异，因为每种服务都引入了自己的结构，可能需要不同的方法来实现每项服务的孤岛和池化存储模式。

尽管数据分区和租户隔离是不同的主题，但您选择的数据分区策略可能会受到数据隔离模式的影响。例如，您可能会孤立一些存储，因为这种方法最符合您的域或客户的需求。或者，您可能会选择孤岛模式，因为池化模式可能不允许您以解决方案所需的粒度级别强制实施隔离。

近邻干扰也会影响您的隔离方法。应用程序中的某些工作负载或使用案例可能需要分开，以限制来自其他租户的影响或满足服务水平协议（SLA）的要求。

计量、指标和计费

关于 SaaS 的讨论往往还会包括计量、指标和计费的概念。这些概念通常会合并为一个概念。但是，区分计量、指标和计费在 SaaS 环境中的不同作用非常重要。

这些概念的难点在于，它们经常对同一个词有重叠的用法。例如，我们所说的计量可以指用于生成账单的计量数据，也可以指代用于跟踪与计费无关的资源内部消耗。我们在本次讨论中谈到的指标和 SaaS 也可能会涉及不同的上下文。

为了帮助解决这个问题，让我们把一些特定的概念与各个术语关联起来（这种关联并不是绝对的）。

- 计量 — 这个概念有许多定义，但最适合的是 SaaS 计费领域。它指的是，您可以计量租户活动或资源消耗，从而收集生成账单所需的数据。
- 指标 — 指标是指您为分析业务、运营和技术领域的趋势而捕获的所有数据。这些数据可以由 SaaS 团队中的多种角色在多种上下文中使用。

这种区分并不重要，但有助于简化我们对“计量和指标在 SaaS 环境中的作用”这一问题的思考。

现在，如果我们将这两个概念与示例联系起来，您可以考虑使用特定的计量事件来检测您的应用程序，这些事件用于显示生成账单所需的数据。这可能是请求数、活跃用户数，也可能会映射到与某个对您的客户有意义的单元相关的消耗（请求、CPU、内存）的数据汇总。

在您的 SaaS 环境中，您将从您的应用程序发布这些计费事件，这些事件将由您的 SaaS 系统所采用的计费结构提取和应用。这可能是第三方计费系统或自定义系统。

相比之下，指标背后的思维方式是捕获相关操作、活动、消费模式等，这些对于评估不同租户对系统的运行状况和运营的影响至关重要。您在此处发布和汇总的指标更多地取决于不同角色（运营团队、产品负责人、架构师等）的需求。系统在此处发布这些指标数据并将其汇总到一些分析工具中，以便这些不同的用户构建系统活动视图，进而根据他们各自的角色分析系统中的各个方面。产品所有者可能想了解不同的租户如何使用相关特征。架构师可能需要视图来帮助他们了解租户如何消耗基础设施资源，诸如此类。

B2B 和 B2C SaaS

SaaS 服务是为 B2B 和 B2C 市场创建的。尽管不同市场和客户的动态不同，但各个市场中 SaaS 的总体原则大致相同。

例如，如果您看一下加入情况，B2B 和 B2C 客户的加入体验可能会有所不同。的确，B2C 系统可能更关注自助加入流程（尽管 B2B 系统可能也支持自助流程）。

虽然客户加入的方式可能存在差异，但加入的基本潜在价值大致相同。即使您的 B2B 解决方案依赖于内部加入流程，我们仍希望该流程尽可能顺畅和自动化。使用 B2B 并不意味着我们会随着时间的推移改变对客户价值的期望。

结论

本文旨在概述 SaaS 架构的基本概念，对用于描述 SaaS 模式和策略的一些模式和术语进行澄清。希望这能帮助组织更清晰地了解完整的 SaaS 场景。

此处涵盖的大部分内容都集中在 SaaS 的含义上，重点是创建一个环境，使您能够通过统一的体验管理和运营所有 SaaS 租户。这符合 SaaS 首先是一种商业模式的核心理念。您构建的 SaaS 架构旨在促进这些基本业务目标。

延伸阅读

针对与此处描述相符的 SaaS 架构模式，有许多资源进行了更详细的介绍。

有关更多信息，请参阅：

- SaaS 租户隔离策略（AWS 白皮书）
- SaaS 存储策略（AWS 白皮书）
- Well-Architected SaaS 剖析（AWS 白皮书）

贡献者

以下个人和组织参与了本文档的编撰：

- Tod Golding，AWS SaaS Factory 首席合作伙伴解决方案架构师

文档修订

如需获取有关本白皮书更新的通知，请订阅 RSS 源。

变更	说明	日期
初次发布	已发布白皮书。	2022 年 8 月 3 日

注意事项

客户有责任对本文档中的信息进行单独评测。本文档：（a）仅供参考，（b）代表当前的 AWS 产品和实践，如有更改，恕不另行通知，以及（c）不构成 AWS 及其附属公司、供应商或许可方的任何承诺或保证。AWS 产品或服务“按原样”提供，不附带任何明示或暗示的保证、陈述或条件。AWS 对其客户承担的责任和义务受 AWS 协议制约，本文档不是 AWS 与客户直接协议的一部分，也不构成对该协议的修改。

AWS 术语表

有关最新的 AWS 术语，请参阅《AWS 词汇表参考》中的 AWS 词汇表。