SaaS 架构基础

AWS 白皮书

目录

SaaS 架构基础

发布日期： 2022 年 8 月 3 日（文档修订（第 31 页））

抽象的

在软件即服务 (SaaS) 模型中运行业务的范围、目标和性质可能难以定义。用于表征 SaaS 的术语和模式因其来源而异。本文档的目标是更好地定义 SaaS 的基本元素，并更清晰地描述在 AWS 上设计和交付 SaaS 系统时应用的模式、术语和价值系统。更广泛的目标是提供一系列基础见解，让客户更清楚地了解他们在寻求采用 SaaS 交付模型时应考虑的选项。

本白皮书面向处于 SaaS 旅程初期的 SaaS 构建者和架构师，以及希望加深对核心 SaaS 概念的理解的经验丰富的构建者。其中一些信息对于想要更熟悉 SaaS 环境的 SaaS 产品所有者和战略家也很有用。

你是 Well-Architected 吗？

AWS架构完善的框架帮助您了解在云中构建系统时所做决策的利弊。该框架的六大支柱使您能够学习设计和运行可靠、安全、高效、经济高效且可持续的系统的架构最佳实践。

使用AWS 架构完善的工具，在 AWS 管理控制台中免费提供，您可以通过回答每个支柱的一组问题来对照这些最佳实践来审查您的工作负载。

有关您的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。

介绍

术语软件即服务 (SaaS) 用于描述业务和交付模型。然而，挑战在于，SaaS 的含义并未得到普遍理解。

尽管对 SaaS 的一些核心支柱达成了一些共识，但对于 SaaS 的含义仍然存在一些困惑。团队对 SaaS 的看法存在一些差异是很自然的。同时，SaaS 概念和术语的不明确会给那些探索 SaaS 交付模型的人带来一些困惑。

本文档重点概述用于描述核心 SaaS 概念的术语。
围绕这些概念拥有共同的思维方式可以清晰地了解 SaaS 架构的基本元素，为您提供描述 SaaS 架构构造的共享词汇表。

这在您深入研究基于这些主题的其他内容时特别有用。

本白皮书从多租户的架构细节出发，探索我们如何定义 SaaS 的基础知识。理想情况下，这还将提供一组更清晰的术语，使组织能够更快地调整其 SaaS 解决方案的风格和性质。

SaaS 是一种商业模式

定义 SaaS 的含义首先要就一个关键原则达成一致:SaaS 是一种商业模式。
这意味着 最重要的是 采用 SaaS 交付模型直接由一组业务目标驱动。是的,技术将用于实现其中一些目标,但
SaaS 是关于建立一种针对一组特定业务目标的思维方式和模型。

让我们更仔细地了解一些与采用 SaaS 交付模型相关的关键业务目标。

·敏捷性 该术语概括了SaaS 的更广泛目标。成功的 SaaS 公司建立在这样的理念之上,即他们必须准备好不断适应市场、客户和竞争动态。伟大的 SaaS 公司的结构是不断接受新的定价模型、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高规模和敏捷性。
这意味着要建立一种文化和工具,专注于创建可促进频繁快速发布新功能的运营足迹。它还意味着拥有单一、统一的体验,使您能够集中管理、运营和部署所有客户环境。支持一次性版本和定制的想法已经一去不复返了。 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。

·无摩擦入职 作为更加敏捷和拥抱增长的一部分,您还必须重视减少租户客户入职流程中的任何摩擦。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,您仍然需要关注为客户创造价值的时间。向以服务为中心的模式转变需要 SaaS 业务关注客户体验的各个方面,特别强调整个入职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;是关于
制定允许您进行创新的基本要素。您希望在您的 SaaS 模型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,从而为您的客户打开新的市场、机会和效率。

·市场反应 SaaS 摆脱了传统的季度发布和两次发布的概念
年计划。它依靠其敏捷性使组织能够近乎实时地对市场动态做出反应和响应。对 SaaS 的组织、技术和文化元素的投资创造了根据新兴客户和市场动态调整业务战略的机会。

·增长 SaaS 是一种以增长为中心的业务战略。围绕敏捷性和效率调整组织的所有活动部分,使 SaaS 组织能够以增长模式为目标。这意味着建立支持并欢迎快速采用您的 SaaS 产品的机制。

您会注意到这些项目中的每一项都侧重于业务成果。有多种技术策略和模式可用于构建 SaaS 系统。然而,这些技术策略并没有改变更广泛的商业故事。

当我们与组织坐下来询问他们在采用 SaaS 的过程中想要实现什么时,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实现。例如,在没有实现敏捷性、运营效率或无摩擦入职的情况下成为多租户,将破坏您的 SaaS 业务的成功。

以此为背景,让我们尝试将其形式化为符合前面概述的原则的更简洁的 SaaS 定义:

SaaS 是一种业务和软件交付模型，使组织能够以低摩擦、以服务为中心的模型提供解决方案，从而为客户和提供商实现价值最大化。它依靠敏捷性和运营效率作为促进增长、影响力和创新的业务战略的支柱。

您应该看到业务目标与它们如何依赖于为所有客户提供共享体验之间的一致性。转向 SaaS 的很大一部分意味着摆脱可能是传统软件模型一部分的一次性定制。任何为客户提供专业化的努力通常都会使我们偏离我们试图通过 SaaS 实现的核心价值。

你是服务,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,您会发现自己正在远离传统的基于产品的开发方法的各个方面。虽然特性和功能对每个产品都很重要,但 SaaS 更强调客户对您的服务的体验。

这是什么意思?在以服务为中心的模型中,您更多地考虑客户如何开始使用您的服务、他们实现价值的速度以及您可以多快地引入满足客户需求的功能。与您的服务构建、运营和管理方式相关的详细信息不在客户的视野范围内。

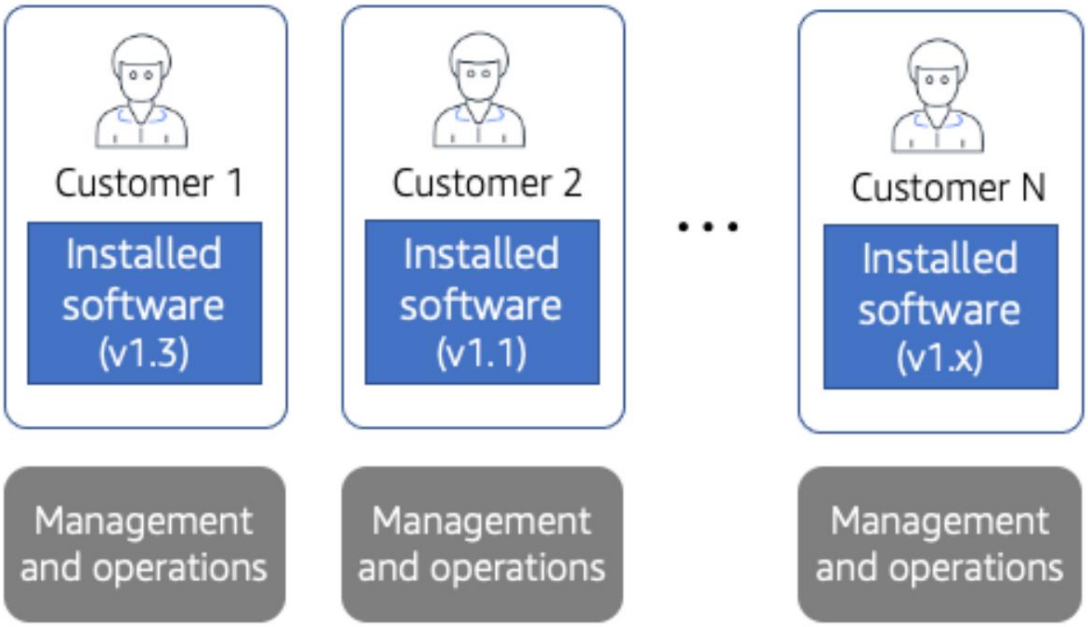
在这种模式下,我们将此 SaaS 服务视为我们可能使用的任何其他服务。如果我们在餐厅,我们当然关心食物,但我们也关心服务。服务员上桌的速度、给你加水的频率、上菜的速度 这些都是衡量服务体验的指标。这与塑造我们构建 SaaS 服务的方式的思维方式和价值体系是一样的。

这种即服务模型应该对您如何构建团队和服务产生重大影响。
您的积压工作现在将使这些体验属性与特性和功能处于同等或更高的地位。企业还将把这些视为 SaaS 产品长期增长和成功的基础。

最初的动机

要了解 SaaS,让我们从一个相当简单的概念开始,了解我们在创建 SaaS 业务时试图实现的目标。最好的起点是查看传统（非 SaaS)软件的创建、管理和操作方式。

下图提供了几个供应商如何打包和交付他们的解决方案的概念视图。



打包和交付软件解决方案的经典模型

在此图中,我们描述了一组客户环境。这些客户代表购买了供应商软件的不同公司或实体。这些客户中的每一个基本上都在安装了软件提供商产品的独立环境中运行。

在这种模式下,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所有者,可能会请求支持其需求的一次性定制或独特配置。

这种心态的一个常见副作用是客户将控制他们运行的产品版本。发生这种情况的原因有多种。客户可能对新功能感到担忧,或者担心与采用新版本相关的中断。

您可以想象这种动态如何影响软件提供商的运营足迹。您允许客户拥有一次性环境的次数越多,管理、更新和支持每个客户的不同配置就越具有挑战性。

这种对一次性环境的需求通常要求组织创建专门的团队,为每个客户提供单独的管理和运营经验。虽然其中一些资源

可能会在客户之间共享,但此模型通常会为每个新加入的客户引入增量费用。

让每个客户在自己的环境（云端或本地)中运行他们的解决方案也会影响成本。虽然您可以尝试扩展这些环境，但扩展将仅限于单个客户的活动。从本质上讲,您的成本优化仅限于您在单个客户环境范围内可以实现的目标。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变化。

最初,一些软件企业会将此模型视为一种强大的结构。他们使用提供一次性定制的能力作为一种销售工具,允许新客户提出对其环境独有的要求。虽然客户数量和业务增长仍然不大,但这种模式似乎完全可持续。

然而,随着公司开始取得更广泛的成功,这种模式的限制开始带来真正的挑战。想象一下,例如,如果您的业务达到显着增长峰值,您会快速增加大量新客户。这种增长将开始增加运营开销、管理复杂性、成本和许多其他问题。

最终,这种模式的集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。与吸引客户相关的增量人员配置和成本开始侵蚀业务利润。

然而,运营问题只是挑战的一部分。真正的问题是,这种模型随着规模的扩大,直接开始影响企业发布新功能和跟上市场步伐的能力。
当每个客户都有自己的环境时,提供商在尝试将新功能引入其系统时必须平衡大量更新、迁移和客户需求。

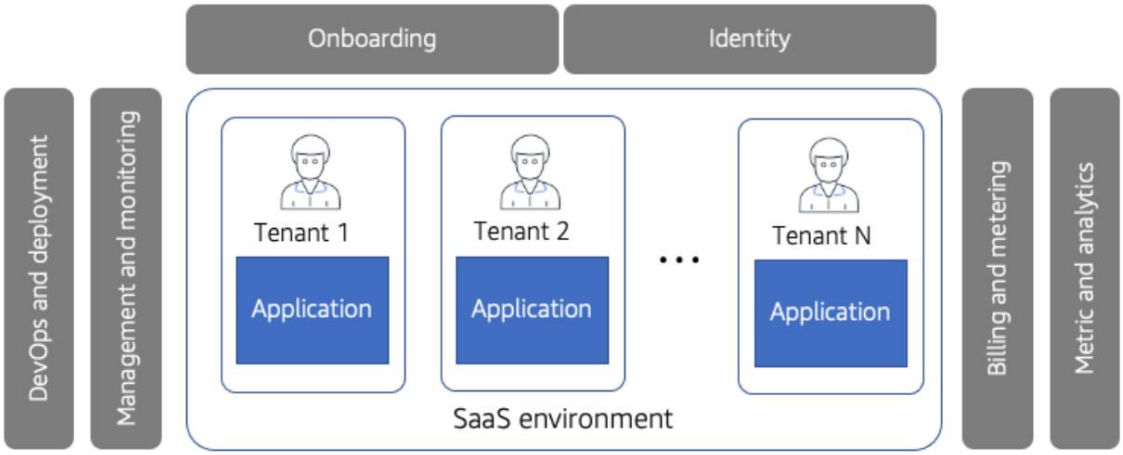
这通常会导致更长、更复杂的发布周期,这往往会减少每年发布的版本数量。更重要的是,这种复杂性让团队在每个新功能发布给客户之前就投入更多时间来分析它。团队开始更加关注验证新功能,而不是交付速度。发布新功能的开销变得如此巨大,以至于团队更加关注测试机制,而更少关注推动产品创新的新功能。

在这种更慢、更谨慎的模式下,团队往往会有很长的周期时间,这使得一个想法的开始与它落到客户手中之间的差距越来越大。总的来说,这会阻碍您对市场动态和竞争压力做出反应的能力。

转向统一体验

为了解决这一经典软件困境的需求,组织转向了一种模型,该模型允许他们创建单一、统一的体验,从而允许对客户进行集体管理和运营。

下图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型进行管理、入职、计费和运营。



一个环境的概念视图,其中所有客户都通过共享模型进行管理、入职、计费和运营

乍一看,这似乎与之前的模型没有什么不同。但是,随着我们深入研究,您会发现这两种方法存在根本的、显着的差异。

首先,您会注意到客户环境已重命名为租户。租户的概念是 SaaS 的基础。基本思想是您有一个单一的 SaaS 环境,您的每个客户都被视为该环境的租户,消耗他们需要的资源。租户可以是拥有许多用户的公司,也可以直接与个人用户相关联。

为了更好地理解租户的想法,请考虑公寓或商业建筑的想法。
这些建筑物中的每一个空间都出租给各个租户。租户依赖建筑物的一些共享资源(水、电等),为他们所消耗的资源付费。

SaaS 租户遵循类似的模式。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不同。这些租户也是集体管理、计费和运营的。

如果您返回该图,您会看到租户的概念栩栩如生。在这里,租户不再有自己的环境。相反,所有租户都在一个集体 SaaS 环境的围墙内安置和管理。

该图还包括围绕您的 SaaS 环境的一系列共享服务。这些服务对您的 SaaS 环境的所有租户都是全局的。这意味着,例如,入职和身份由该环境的所有租户共享。管理、运营、部署、计费和指标也是如此。

这种普遍应用于所有租户的统一服务集的想法是 SaaS 的基础。通过分享这些概念，您能够解决与上述经典模型相关的许多挑战。

此图的另一个关键且有点微妙的元素是此环境中的所有租户都运行相同版本的应用程序。为每个客户运行单独的一次性版本的想法已经一去不复返了。让所有租户运行相同的版本代表了 SaaS 环境的基本区别属性之一。

通过让所有客户运行相同版本的产品，您将不再面临经典安装软件模型的许多挑战。在统一模型中，可以通过单个共享流程将新功能部署到所有租户。

这种方法使您能够使用单一的操作面板来管理和操作所有租户。这使您可以通过共同的操作体验来管理和监控您的租户，从而允许添加新租户而无需增加运营开销。这是 SaaS 价值主张的核心部分，它使团队能够降低运营费用，并提高整体组织敏捷性。

想象一下在这个模型中添加 100 或 1,000 个新客户意味着什么。您不必担心这些新客户会如何侵蚀利润并增加复杂性，您可以将这种增长视为它所代表的机会。

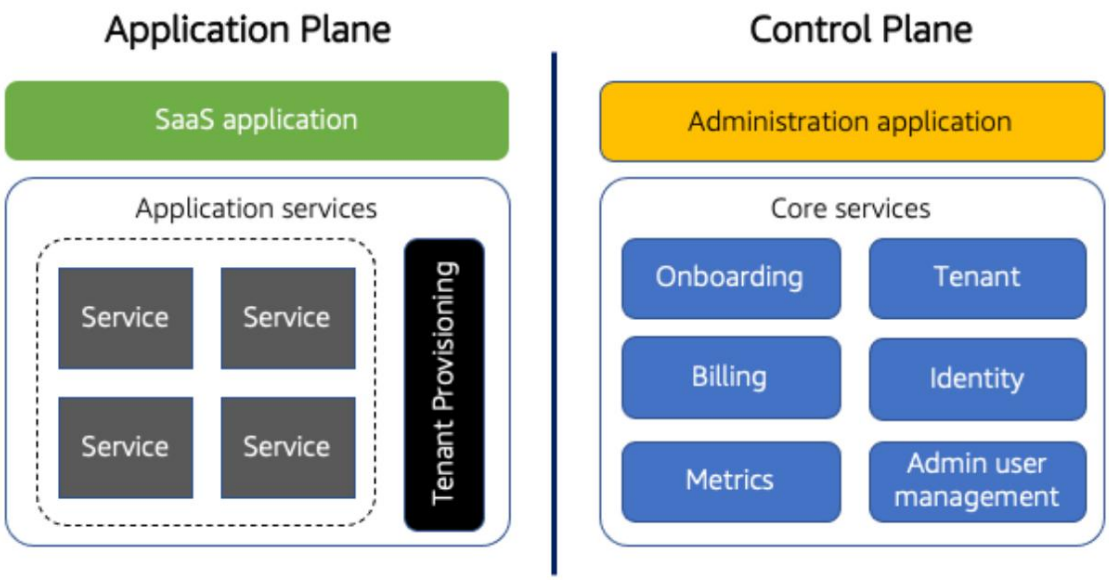
通常，SaaS 的重点放在这个模型中间的应用程序是如何实现的。
企业希望关注数据的存储方式、资源的共享方式等。然而，现实情况是，虽然这些细节绝对重要，但可以通过多种方式构建您的应用程序，并将其本身作为 SaaS 解决方案呈现给您的客户。

关键是拥有围绕您的租户环境的单一、统一体验的更广泛目标。拥有这种共享的经验可以让您推动与 SaaS 业务的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 体系结构概念的概念视图。现在，让我们深入了解一下，更好地定义您的 SaaS 环境如何分解为不同的层。
更清楚地了解 SaaS 概念之间的界限将使描述 SaaS 解决方案的移动部分变得更加容易。

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。图的这一侧包括用于载入、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

此控制平面是任何多租户 SaaS 模型的基础。每个 SaaS 解决方案（无论应用程序部署和隔离方案如何）都必须包含那些使您能够通过单一、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。此处的核心服务代表用于协调您的多租户体验的服务集合。我们包含了一些通常属于核心部分的常见服务示例，承认核心服务可能因每个 SaaS 解决方案而有所不同。

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 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）。

一个重要的警告是控制平面及其服务实际上并不是多租户的。该功能不提供 SaaS 应用程序的实际功能属性（需要多租户）。例如，如果您查看任何一项核心服务，您将找不到租户隔离和属于多租户应用程序功能一部分的其他结构。这些服务对所有租户都是全球性的。

图的左侧引用了 SaaS 环境的应用程序平面。这是应用程序的多租户功能所在的位置。图中出现的需要保留

有点模糊,因为每个解决方案都可以根据您的领域需求、技术足迹等进行不同的部署和分解。

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。这是租户接触以与您的 SaaS 应用程序交互的表面。然后是表示 SaaS 解决方案的业务逻辑和功能元素的后端服务。这些可能是微服务,或您的应用程序服务的其他一些包装。

您还会注意到我们已经打破了配置。这样做是为了强调一个事实,即在入职期间为租户提供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平面。但是,我们将它放在应用程序域中,因为它必须提供和配置的资源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可以更容易地考虑 SaaS 架构的整体格局。更重要的是,它强调了对一组完全超出应用程序功能范围的服务的需求。

核心服务

前面提到的控制平面提到了一系列核心服务，这些核心服务代表用于载入、管理和操作 SaaS 环境的典型服务。进一步突出其中一些服务的作用可能有助于突出它们在 SaaS 环境中的范围和目的。

以下提供了这些服务中的每一个的简要总结：

·入职 每个 SaaS 解决方案都必须提供一种无摩擦的机制来引入新的
租户进入您的 SaaS 环境。这可以是自助注册页面或内部管理的体验。无论哪种方式，SaaS 解决方案都应该尽其所能消除这种体验中的内部和外部摩擦，并确保该过程的稳定性、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重要作用。通常，此服务会编排其他服务以创建用户、租户、隔离策略、供应和每个租户

资源。

·租户 租户服务提供了一种集中租户策略、属性和状态的方法。
关键是租户不是个人用户。事实上，一个租户很可能与许多用户相关联。 ·身份 SaaS 系统需要一种清晰的方式将用户与租户联系起来，将租户上下文引入其解决方案的身份验证和授权体验中。这会影响入职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，组织通常采用新的计费模型。他们也可能
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入职，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，这些数据可以更清楚地了解租户如何使用他们的系统、他们如何消耗资源以及他们的租户如何使用他们的系统。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

术语多租户和SaaS通常紧密相关。在某些情况下,组织将 SaaS 和多租户描述为同一事物。虽然这看起来很自然,但将 SaaS 和多租户等同起来往往会导致团队对 SaaS 采取纯粹的技术观点,而实际上,SaaS 更像是一种业务模型,而不是一种架构策略。

为了更好地理解这个概念,让我们从多租户的经典观点开始。在这种纯粹以基础设施为中心的观点中,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率。

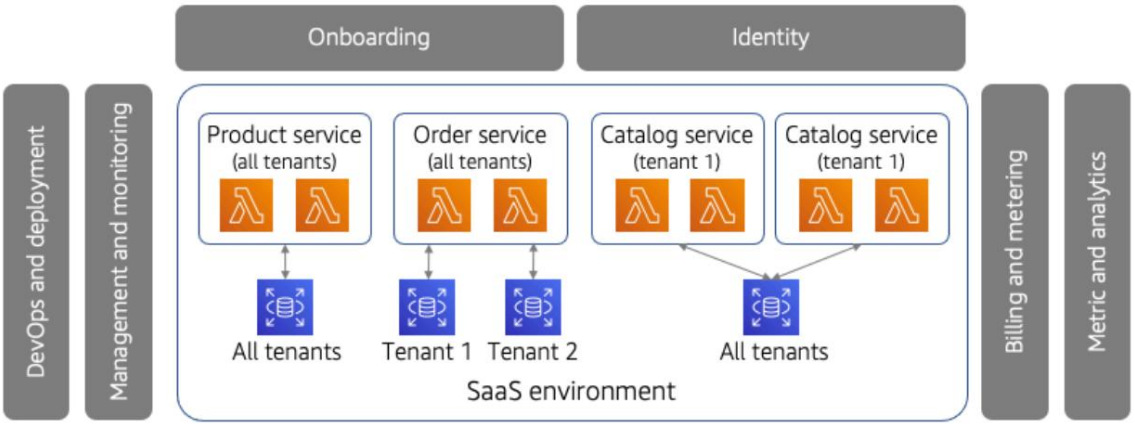
例如,假设您有微服务或Amazon Elastic Compute Cloud (Amazon EC2) 实例,由您的 SaaS 系统的多个租户使用。该服务将被视为在多租户模型中运行,因为租户共享使用运行该服务的基础设施。

这个定义的挑战在于它过于直接地将多租户的技术概念附加到 SaaS。它假定 SaaS 的定义特征是它必须具有共享的多租户基础设施。当我们查看在不同环境中实现 SaaS 的各种方式时,这种 SaaS 观点开始分崩离析。

下图提供了一个 SaaS 系统的视图,它揭示了我们在定义多租户时遇到的一些挑战。

在这里,您将看到前面描述的经典 SaaS 模型,以及一系列被共享服务包围的应用程序服务,这些服务允许您集中管理和运营您的租户。

新功能是我们包含的微服务。该图包括三个示例微服务:产品、订单和目录。如果您仔细观察这些服务中的每一个的租赁模式,您会发现它们都采用略有不同的租赁模式。



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)。这符合多租户的经典定义。但是,如果您查看订单服务,您会发现它具有共享计算,但每个租户都有单独的存储。

目录服务添加了另一种变体,其中每个租户的计算都是独立的（每个租户都有一个单独的微服务部署）,但它为所有租户共享存储。

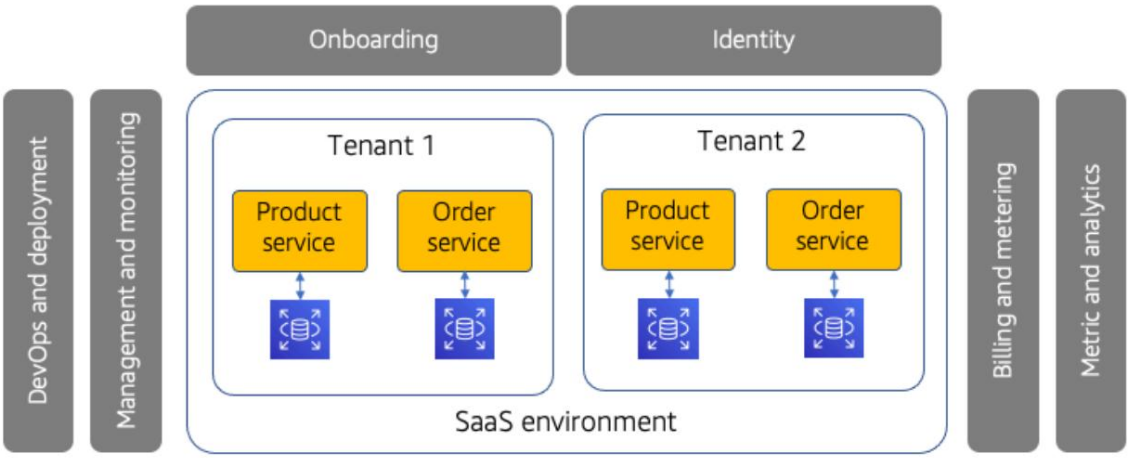
这种性质的变体在 SaaS 环境中很常见。吵闹的邻居,分层模型、隔离需求 这些都是您可能有选择地共享或孤立 SaaS 解决方案部分的原因之一。

考虑到这些变化 以及许多其他可能性 弄清楚应该如何使用术语多租户来描述这种环境变得更具挑战性。总的来说,就客户而言,这是一个多租户环境。但是,如果我们使用最技术性的定义,这个环境的某些部分是多租户的,而另一些则不是。

这就是为什么有必要放弃使用术语多租户来描述 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,但避免使用它来将解决方案描述为 SaaS。如果要使用多租户这个术语,那么用它来描述整个 SaaS 环境是多租户更有意义,因为知道架构的某些部分可能是共享的,而某些部分可能不是。总的来说,您仍然在多租户模型中操作和管理这个环境。

极端情况

为了更好地强调这种租赁概念,让我们看一下具有共享零资源的租户的 SaaS 模型。下图提供了一些 SaaS 提供商使用的示例 SaaS 环境。



每个租户堆栈

在此图中,您会看到我们仍然拥有围绕这些租户的公共环境。
但是,每个租户都部署了一组专用资源。在此模型中,租户不共享任何内容。

这个例子挑战了多租户的意义。这是一个多租户环境,即使没有共享任何资源吗?使用该系统的租户与您对具有共享资源的 SaaS 环境有着相同的期望。事实上,他们可能不知道他们的资源是如何在 SaaS 环境的幕后部署的。

尽管这些租户在孤立的基础设施中运行,但它们仍然是集体管理和运营的。他们共享统一的入职、身份、指标、计费和运营经验。

此外,当发布新版本时,它会部署到所有租户。为此,您不能允许对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型。虽然它可能无法实现共享基础架构的所有效率,但它是一个完全有效的多租户 SaaS 环境。对于某些客户,他们的域可能要求他们的部分或所有客户在此模型中运行。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,这仍然符合 SaaS 的基本原则。

鉴于这些参数和 SaaS 的这种更广泛的定义,您可以看到需要发展多租户一词的使用。将任何共同管理和运营的 SaaS 系统称为多租户系统更有意义。然后,您可以使用更细化的术语来描述如何在 SaaS 解决方案的实施中共享或专用资源。

删除单租户术语

作为使用术语多租户的一部分,人们很自然地希望使用术语单租户来描述 SaaS 环境。然而,鉴于前面概述的背景,单租户一词会造成混淆。

上图是单租户环境吗?虽然每个租户在技术上都有自己的堆栈,但这些租户仍在多租户模型中进行操作和管理。这就是为什么通常避免使用单租户一词的原因。相反,所有环境都具有多租户的特征,因为它们只是实施一些租户变体,其中部分或全部资源是共享或专用的。

筒仓和池简介

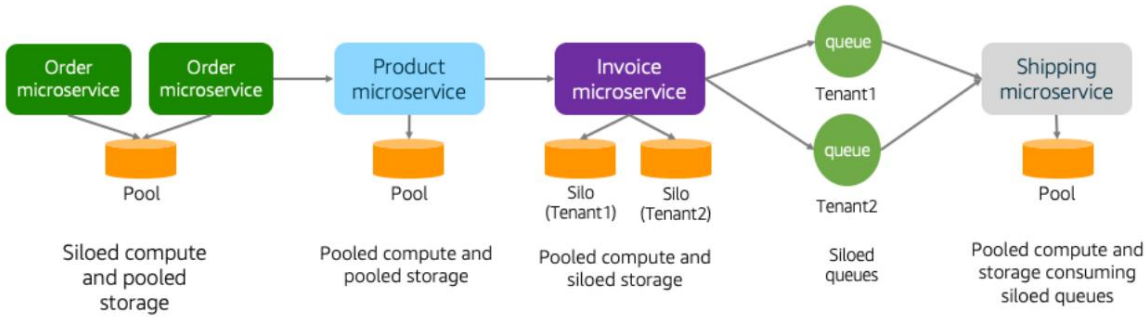
鉴于模型的所有这些变化以及多租户一词面临的挑战,我们引入了一些术语,使我们能够更准确地捕捉和描述构建 SaaS 时使用的不同模型。

我们用来描述 SaaS 环境中资源使用的两个术语是筒仓和池。
这些术语允许我们标记 SaaS 环境的性质,使用多租户作为可以应用于任意数量的底层模型的总体描述。

在最基本的层面上,术语筒仓是用来描述资源专用于给定租户的场景。相反,池模型用于描述租户共享资源的场景。

当我们查看筒仓和池术语的使用方式时,重要的是要清楚筒仓和池不是全有或全无的概念。筒仓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 环境的一部分。因此,如果我们说某些资源正在使用筒仓模型,那并不意味着该环境中的所有资源都是筒仓式的。这同样适用于我们如何使用术语汇集。

下图提供了一个示例,说明如何在 SaaS 环境中更精细地使用孤立模型和池模型:



筒仓和水池模型

该图包括一系列示例,旨在说明筒仓和池模型更具针对性的性质。如果您从左到右按照此操作,您会看到我们从订单微服务开始。该微服务具有孤立的计算和池化存储。它与具有池计算和池存储的产品服务交互。

然后,产品服务与具有池化计算和孤立存储的发票微服务交互。此服务通过队列将消息发送到运送服务。队列部署在一个孤立的模型中。

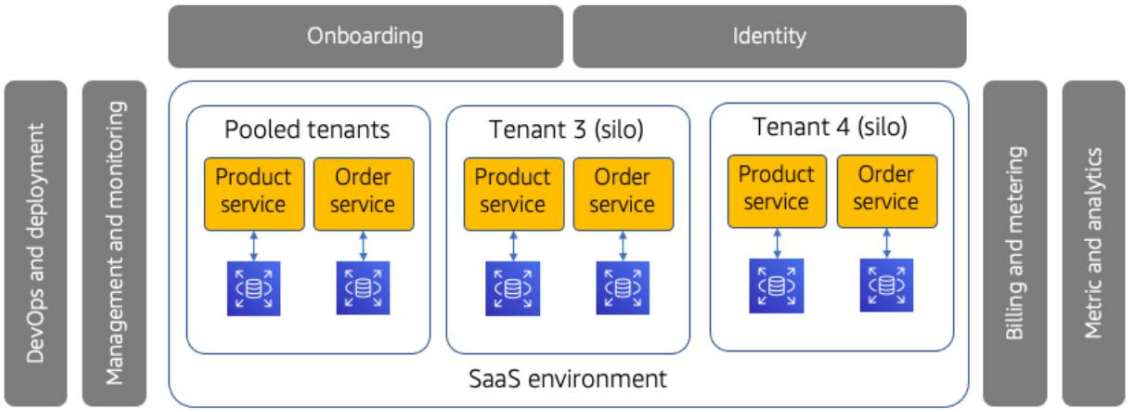
最后,运输微服务从孤立队列中获取消息。它使用池化计算和存储。

虽然这看起来有点令人费解,但目标是突出筒仓和池概念的粒度性质。当您希望设计和构建 SaaS 解决方案时,预计您会根据您的域和客户的需求做出这些孤岛和池决策。

嘈杂的邻居、隔离、分层和许多其他原因可能会影响您选择应用筒仓或池模型的方式和时间。

全栈筒仓和池

Silo 和 pool 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方式部署。下图提供了一个示例,说明这可能如何落入 SaaS 环境。



全栈筒仓和池模型

在此图中,您将看到针对全堆栈租户部署的三种不同模型。
首先,您会看到有一个完整的堆栈池环境。此池中的租户共享所有资源(计算、存储等)。

显示的其他两个堆栈旨在表示完整的堆栈孤立租户环境。在这种情况下,租户 3 和租户 4 显示为每个都有自己的专用堆栈,其中没有任何资源与其他租户共享。

在同一个 SaaS 环境中混合使用筒仓和池化模型并不是那么不寻常。例如,想象一下,您有一组基本层租户,他们支付适中的价格来使用您的系统。这些租户被放置在池化环境中。

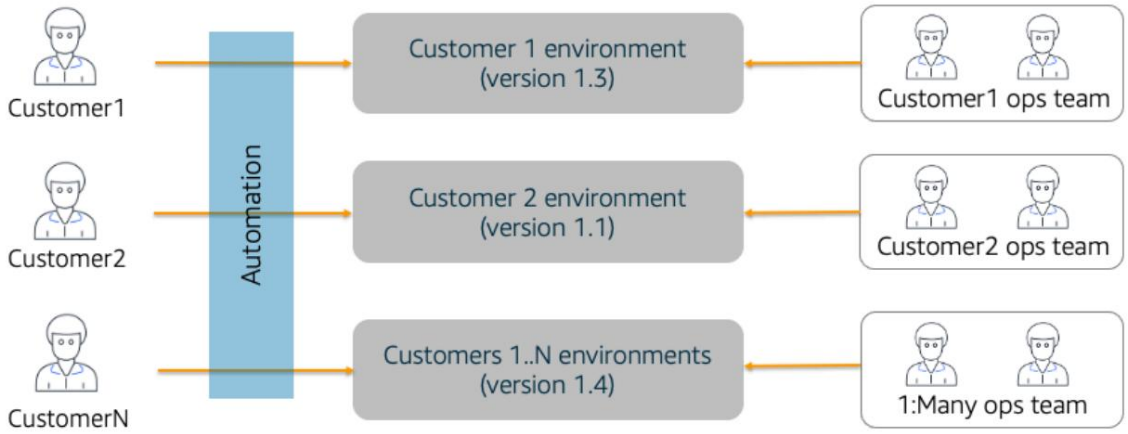
同时,您可能还有高级租户愿意为在筒仓中运行的特权支付更多费用。这些客户部署有单独的堆栈(如图所示)。

即使在这个模型中,您可能允许租户在他们自己的全栈孤岛中运行,这些孤岛也不允许对这些租户进行任何一次性变化或定制,这一点至关重要。在所有方面,这些堆栈中的每一个都应该运行相同的堆栈配置,以及相同版本的软件。发布新版本时,会将其部署到共用租户环境和每个孤立环境。

SaaS 与托管服务提供商 (MSP)

围绕 SaaS 和托管服务提供商 (MSP) 模型之间的界线也存在一些混淆。如果您查看 MSP 模型,它似乎与 SaaS 模型有一些相似的目标。

但是,如果您更深入地研究 MSP,您会发现 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) 模型

此图表示 MSP 模型的一种方法。在左侧,您会看到在 MSP 模型中运行的客户。一般来说,这里的方法是使用任何可用的自动化来配置每个客户环境,并为该客户安装软件。

右边是 MSP 为支持这些客户环境而提供的运营足迹的一些近似值。

请务必注意,MSP 通常会安装和管理给定客户想要运行的产品版本。所有客户都可以运行相同的版本,但这在 MSP 模型中通常不是必需的。

总体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活。虽然这让供应商的生活变得更简单,但它并没有直接映射到 SaaS 产品所必需的价值观和心态。

重点是卸载管理责任。采取这一举措并不等同于让所有客户都在具有单一、统一管理和运营体验的同一版本上运行。相反,MSP 通常允许使用单独的版本,并且通常将这些环境中的每一个视为操作上独立的。

在某些领域,MSP 可能会开始与 SaaS 重叠。如果 MSP 本质上要求所有客户运行相同的版本,并且 MSP 能够通过一种体验集中对所有租户进行入职、管理、运营和计费,那么它可能开始更像是 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。只有当您添加之前讨论的所有其他注意事项时，这才更像是一个真正的 SaaS 模型。

如果我们从这个故事的技术和运营方面回过头来看，MSP 和 SaaS 之间的界限会变得更加明显。通常，作为 SaaS 企业，您的产品的成功取决于您深入参与体验的所有活动部分的能力。

这通常意味着要把握入职体验的脉搏，了解运营事件如何影响租户，跟踪关键指标和分析，并贴近客户。

在将其移交给其他人的 MSP 模型中，您最终可能会从关键细节中移除一个级别，而关键细节是运营 SaaS 业务的核心。

SaaS 迁移

许多采用 SaaS 的提供商从传统的安装软件模型（如前所述）迁移到 SaaS。对于这些供应商，在 SaaS 的核心原则上保持良好的一致性尤为重要。

在这里，对于迁移到 SaaS 模型意味着什么，可能存在一些混淆。
例如，有些人将迁移到云视为迁移到 SaaS。其他人则将在他们的安装和配置过程中添加自动化视为实现迁移。

可以公平地说，每个组织可能起步不同，有不同的遗留考虑因素，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不一样。

尽管如此，尽管每条路径都不同，但在某些领域，围绕塑造迁移策略的核心原则存在脱节。围绕概念和原则保持良好的一致性会对 SaaS 迁移的整体成功产生重大影响。

根据前面概述的概念，应该清楚转向 SaaS 始于业务战略和目标。这一点可能会在迁移设置中丢失，因为迁移设置存在尽快进入 SaaS 的压力。

在这种模式下，组织通常主要将迁移视为一种技术练习。现实情况是，每次 SaaS 迁移都应该从目标客户、服务体验、运营目标等的清晰视图开始。更清楚地关注您的 SaaS 业务需要的样子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的影响。

从迁移一开始就拥有这种清晰的愿景，为您在迁移到 SaaS 的过程中如何迁移技术和业务奠定了基础。当你踏上这条道路时，关注那些最能告诉你前进方向的问题。

下表提供了技术迁移思维和业务迁移思维的对比性质的视图。

表 1　技术优先与业务优先迁移

技术第一的心态	商业第一的心态
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？	我们针对哪些细分市场？
我们如何避免嘈杂的邻居条件？	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？	我们的目标是什么服务体验？
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？

左边是技术优先迁移思维模式的示例。工程团队非常专注于追寻对任何 SaaS 架构都非常重要的经典多租户主题。

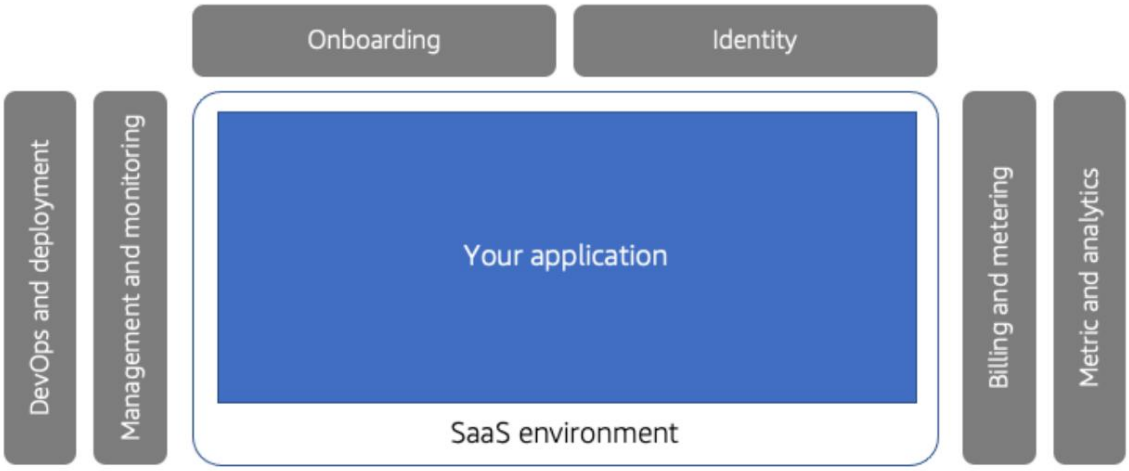
问题是很多左边问题的答案往往直接受到右边问题答案的影响。对于关注迁移的任何人来说，这一点不太可能是新的。

然而,现实情况是,许多组织一开始都将追求运营和成本效率作为第一步,并假设业务部分会自行解决。

在此迁移策略中,对于如何将您的遗留环境演进以适应 SaaS 模型,也可能存在混淆。这也是一个有多种迁移到 SaaS 选项的领域。但是,我们通常提倡任何迁移都有一个共同的价值体系。

在我们之前对 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一个共同主题是围绕您的应用程序提供共享服务的想法。身份、入职、指标、计费 这些都被称为是任何 SaaS 环境核心的共同元素。

现在,当我们查看迁移时,您会看到这些相同的共享服务在任何迁移故事中都发挥着关键作用。下图提供了迁移环境的概念视图。



迁移到 SaaS

此图表示任何迁移路径的目标体验。它包括前面描述的所有相同的共享服务。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,您可能拥有一些混合架构,其中的元素是孤立的,而其他功能则通过一系列现代化的微服务来解决。

根据遗留环境的性质、市场需求、成本考虑等因素,您最初对应用程序的现代化程度会有所不同。不变的是这些共享服务的引入。

任何 SaaS 迁移都需要支持这些基础共享服务,以使您的企业能够在 SaaS 模型中运营。例如,应用程序架构的所有变体都需要 SaaS 身份。
您需要租户感知操作来管理和监控您的 SaaS 解决方案。

将这些共享服务放在迁移的前端,可以让您向客户提供 SaaS 体验 即使底层应用程序仍在每个租户的完整堆栈孤岛中运行。

总体目标是让您的应用程序在 SaaS 模型中运行。然后,您可以将注意力转移到应用程序的进一步现代化和改进上。这种方法还允许您以更快的速度转移业务的其他部分(营销、销售、支持等)。更重要的是,

这使您可以开始参与并收集客户反馈,这些反馈可用于塑造您环境的持续现代化。

请务必注意,您部署的共享服务可能不包括您最终需要的所有功能或机制。主要目标是创建迁移开始时所需的共享机制。这使您可以专注于对应用程序架构的发展和操作发展至关重要的系统元素。

SaaS 身份

SaaS 为您的应用程序的身份模型添加了新的注意事项。当每个用户都经过身份验证时,他们必须连接到特定的租户上下文。此租户上下文提供有关在整个 SaaS 环境中使用的租户的基本信息。

租户与用户的这种绑定通常称为应用程序的SaaS 身份。当每个用户进行身份验证时,您的身份提供者通常会生成一个包含用户身份和租户身份的令牌。

将租户连接到用户代表了 SaaS 架构的一个基础方面,它具有许多下游影响。来自此身份过程的令牌流入您的应用程序的微服务,并用于创建租户感知日志、记录指标、计量计费、强制执行租户隔离等。

您必须避免依赖于将用户映射到租户的独立机制的场景。这会破坏系统的安全性,并经常在您的体系结构中造成瓶颈。

租户隔离

您将客户转移到多租户模型中的次数越多,他们就越担心一个租户访问另一个租户资源的可能性。 SaaS 系统包括确保每个租户的资源（即使它们运行在共享基础设施上)的明确机制。

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对资源的访问并阻止任何访问其他租户资源的尝试的结构。

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,租户用户通过身份验证这一事实并不意味着您的系统已实现隔离。隔离与可能是您的应用程序一部分的基本身份验证和授权分开应用。

为了更好地理解这一点,假设您已经使用身份提供者来验证对您的 SaaS 系统的访问。来自此身份验证体验的令牌还可能包含有关用户角色的信息,该信息可用于控制该用户对特定应用程序功能的访问。这些构造提供安全性,但不提供隔离。事实上,用户可以通过身份验证和授权,并且仍然可以访问另一个租户的资源。关于身份验证和授权的任何内容都不一定会阻止此访问。

租户隔离专门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。它对该租户内的所有用户应用这种隔离。

当我们研究如何跨所有不同的 SaaS 架构模式实现租户隔离时,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其中网络（或更粗粒度的)策略阻止跨租户访问的租户来实现隔离。在其他情况下,您可能拥有池化资源（ Amazon DynamoDB中的项目表)需要更细粒度的策略来控制对资源的访问。

任何访问租户资源的尝试都应仅限于属于该租户的那些资源。 SaaS 开发人员和架构师的工作是确定哪种工具和技术组合将支持您的特定应用程序的隔离要求。

数据分区

数据分区用于描述用于表示多租户环境中数据的不同策略。该术语广泛用于涵盖一系列不同的方法和模型,这些方法和模型可用于将不同的数据结构与各个租户相关联。

请注意,人们常常倾向于将数据分区和租户隔离视为可以互换。
这两个概念并不等同。当我们谈论数据分区时,我们是在谈论如何为各个租户存储租户数据。分区数据并不能确保数据是隔离的。仍然必须单独应用隔离,以确保一个租户无法访问另一个租户的资源。

每种 AWS 存储技术都对数据分区策略提出了自己的一套注意事项。例如,在 Amazon DynamoDB 中隔离数据与在Amazon Relational Database Service中隔离数据看起来非常不同(亚马逊 RDS)。

通常,当您考虑数据分区时,首先会考虑数据是孤立的还是集中的。在孤立模型中,每个租户都有不同的存储结构,没有混合数据。对于合并分区,数据根据租户标识符混合和分区,租户标识符确定哪些数据与每个租户相关联。

例如,对于 Amazon DynamoDB,孤立模型为每个租户使用一个单独的表。 Amazon DynamoDB 中的数据池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 表的分区键中实现的。

您可以想象这在 AWS 服务范围内可能会有何不同,每个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筒仓和池存储模型。

虽然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型的影响。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要求。或者,您可能会选择筒仓,因为池模型可能不允许您以解决方案所需的粒度级别强制执行隔离。

吵闹的邻居也会影响您的隔离方式。您的应用程序中的某些工作负载或用例可能需要保持独立,以限制来自其他租户的影响或满足服务水平协议 (SLA)。

计量、指标和计费

SaaS 的讨论还倾向于包括计量、指标和计费的概念。这些概念经常被折叠成一个概念。但是，重要的是要区分计量、指标和计费在 SaaS 环境中扮演的不同角色。

这些概念的挑战在于它们经常重叠使用同一个词。例如，我们可以讨论用于生成账单的计量。同时，我们还可以讨论用于跟踪与计费无关的资源内部消耗的计量。我们还在许多上下文中讨论指标和 SaaS，这些上下文可以混入此讨论中。

为了帮助解决这个问题，让我们将一些特定的概念与这些术语中的每一个相关联（知道这里没有绝对的概念）。

- 计量　这个概念虽然有很多定义，但最适合SaaS 计费领域。这个想法是您测量租户活动或资源消耗以收集生成账单所需的数据。
- 指标　指标代表您为分析整个企业的趋势而捕获的所有数据，操作和技术领域。此数据用于 SaaS 团队中的许多上下文和角色。

这种区别并不重要，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，或者它可能映射到与对您的客户有意义的某个单位相关的某种消耗总量（请求、CPU、内存）。

在您的 SaaS 环境中，您将从您的应用程序发布这些计费事件，并且它们将被您的 SaaS 系统采用的计费结构摄取和应用。这可能是第三方计费系统或自定义的东西。

相比之下，指标背后的思维方式是捕获那些对于评估不同租户强加给您的系统的健康和运营足迹至关重要的行动、活动、消费模式等。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、产品所有者、架构师等）的需求。在这里，这些指标数据被发布并聚合到一些分析工具中，这些工具允许这些不同的用户构建系统活动的视图，以分析最符合他们角色的系统方面。产品所有者可能想了解不同的租户如何使用功能。架构师可能需要一些视图来帮助他们了解租户如何使用基础设施资源等。

B2B 和 B2C 软件即服务

SaaS 产品是为 B2B 和 B2C 市场创建的。虽然市场和客户肯定有不同的动态，但 SaaS 的总体原则不会以某种方式改变这些市场中的每一个。

例如，如果您查看入职，B2B 和 B2C 客户可能会有不同的入职体验。的确，B2C 系统可能更关注自助式入职流程（尽管 B2B 系统也可能支持这一点）。

虽然为客户提供入职培训的方式可能有所不同，但入职培训的基本价值基本相同。即使您的 B2B 解决方案依赖于内部入职流程，我们仍然希望该流程尽可能顺畅和自动化。

成为 B2B 并不意味着我们正在改变我们对时间的期望，为我们的客户创造价值。

结论

本文档的目标是概述基本的 SaaS 架构概念,围绕用于描述 SaaS 模式和策略的模型和术语提供一些说明。希望这将使组织对整体 SaaS 格局有更清晰的认识。

此处涵盖的大部分内容都集中在 SaaS 的含义上,特别强调创建一个环境,让您可以通过统一的体验管理和操作所有 SaaS 租户。这与 SaaS 首先是一种商业模式这一核心理念相关。您构建的 SaaS 架构旨在促进这些基本业务目标。

延伸阅读

有许多资源更详细地介绍了符合此处描述的模式的 SaaS 架构模式。

有关其他信息,请参阅:

· SaaS 租户隔离策略（AWS 白皮书） · SaaS 存储策略（AWS 白皮书）·架构完善的 SaaS 镜头（AWS 白皮书）

贡献者

以下个人和组织为本文档做出了贡献：

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

文档修订

要获得有关本白皮书更新的通知,请订阅 RSS 提要。

改变	描述	日期
首次发布（第 31 页）	白皮书发布。	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自己的独立评估。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,并且 (c) 不构成 AWS 及其附属公司、供应商或许可人。 AWS 产品或服务“按原样”提供,没有任何明示或暗示的保证、陈述或条件。 AWS 对其客户的责任和义务由 AWS 协议控制,本文档不属于也不修改 AWS 与其客户之间的任何协议。

AWS 词汇表

有关最新的 AWS 术语,请参阅AWS 词汇表在AWS 一般参考中。