来自 VMware CTO 办公室的声音：

应用使能型基础设施

—— 应用平台的兴起助力云原生应用的落地

作者：Emad Benjamin
翻译：安浦，宋潇男

目录

1. 主旨

云原生运动已渐入佳境，为了适应当今IT市场的现实情况，其最初忽视现存企业应用的立场也已得到修正。其最初只关注构建无状态系统的立场不再适用 —— 现存企业应用很难做到完全的状态剥离。云原生运动是应用使能型基础设施背后的主导力量，而随之而来的应用平台思想则对高效的企业IT云化战略至关重要。

- 在我们接触过的客户中，半数面临着与应用平台相关的诸多挑战。此外，无论是在私有云和还是公有云中，客户置备的资源都超过了需求 —— 在某些情况下甚至是需求的两倍。

- VMware可以帮助业界构建一个真正的混合云运行时(Hybrid Cloud Runtime ，简称HCR)。HCR是横跨私有云和公有云的结缔组织，能够提供服务网格和运行时层面的应用优化能力，并感知应用工作负载的深层次特性。HCR提供的这种应用感知能力将有助于确保更低的应用响应时间和更少的资源消耗。

- 许多应用现代化项目中的微服务模式都饱受管中窥豹之苦。有些平台只优化了云原生方面而忽略了其他。甚至云原生平台也需要服务网格层的加持才能解决延迟和可扩展性方面的大量问题。

- 受到误导的CIO们认为问题来自于私有云。而事实上，许多已经转向公有云的公司也正在重新评估他们的应用平台战略。在很多案例中，过度的资源置备正导致公有云向私有云的迁移，而缺乏对应用平台本质的理解和能力沉淀才是这个问题的核心。

- 应用平台的能力沉淀正在成为行业中的一个关键模式。一个简单的Kubernetes集群无法构成一个应用平台。

- 为应用平台架构师(Application Platform Architect，简称APA)和站点可靠

性工程师(Site Reliability Engineer，简称SRE)塑造新角色，并在我们的产品中长期提供SRE的服务化，这将帮助客户在私有云和公有云中完成必要的转变。

- 计算机语言运行时旨在单一运行时内优化任意深度的方法和函数调用栈，通常是为代码中的热点区域预热或是使其被更加高效的访问，所有这些都发生在一个内存空间内。然而，微服务和函数即服务（FaaS）模式使应用高度分布于依赖网络连接的多个内存空间，现有的优化机制显得杯水车薪。具体地说，这些优化因无法感知应用的分布式特征而变得不再有效。根本性的解决这个问题，需要在现有的网络栈上添加一层能力，其行为类似于一个分布式运行时编译器，持续的在相互通信的服务网络上进行优化。我们将该层能力的第一个基本版本称为服务网格。

- 我们应该构建应用平台，而不仅仅是一般性的基础设施层。当我们创造 SDDC 这个词时，我们真正的意思是一个软件定义的应用平台(Software Defined Application Platform，简称 SDAP)，因为几乎所有客户都是采用通用的硬件设计并为一组应用平台进行定制。我们应该减轻客户的这种负担，并提供将我们的多种产品适配于特定应用平台用户场景的蓝图。

2. 技术概述

2.1. 概述

我们经常遇到一些用户在决定向公有云迁移之后才考虑其传统架构的关键应用是否已转换为公有云所需的实现模式，这让他们付出了高额成本。无论云的定位如何，真正重要的是如何沉淀企业工作负载的应用平台特性。如果您不了解应用工作负载的可扩展性、性能、可靠性、安全性和如何全面管理，那么您只是将问题从一个云迁移到了另一个云。

在我们与客户沟通时，我们不断的看到客户在构建良好的应用平台上所遇到的困难。在稳定性、部署和可扩展性方面常常存在着各种各样的挑战 —— 我们接触过的客户中过半数都面临着某种来自应用平台方面的挑战。更重要的是，我们不断的看到，无论在私有云还是公有云中，客户置备的资源都超过所需 —— 在某些情况下甚至是需求的两倍。为什么会这样呢？在过去的20年里，一个又一个的运动试图解决这种资源过度供应的现象，但我们仍然发现自己不断的重蹈覆辙，即每当应用表现不佳时就为其提供更多的资源。

IT从业者正用他们的老方法解决新问题。而这个问题的关键在于开发团队和运营团队之间的知识鸿沟。在本文中，我们将讨论***应用平台***的概念及其意义，它旨在缩小开发人员和基础设施架构师之间的差距。在最基本的层次上，您可以将应用平台看作三个主要部分的抽象：***1)应用代码逻辑；2)代码所在的应用运行时；3)基础设施抽象，如CaaS、K8s和基础IaaS。***

当前的云原生运动可以看作是应用平台思想的一个实例。例如，云原生实践是指开发人员进入基础设施实践，使用基础设施即代码技术来完全自动化当前手工步骤，以提供基础设施即服务(IaaS)和容器即服务(CaaS)。毫无疑问，作为一个关键抽象，CaaS是其中的一个重要部分。基础设施团队可以将其视为一种威胁，也可以将其视为迈向下一个成熟度级别的驱动力。从VMware的角度来看，我们必须站在这一转变的最前沿，为这个行业正在形成的新角色构建产品。

应用平台不仅仅意味着云原生运动。它是一个关于在云时代如何运行企业应用平台的流派，它将囊括一个典型企业中存在的所有应用类型。虽然云原生倡导者经常以“您必须重构您的代码”作为前提，但是也有很多不必重构代码的应用现代化路线，例如重构封装应用代码的应用运行时。您有多种应用现代化策略可以采用，在某些情况下，重写应用是可行的，而在更多情况下，重写应用的成本会非常高。当您试图迁移一个曾被认为具备良好内聚性的应用、却很快在错综复杂的关联中越陷越深时，您会对此更加深有感触。

事实上，许多应用都依赖于其他组件，这导致典型的企业应用出现了一种被我们称之为 “提取根因素” 的现象，随着时间的推移，盘根错节的依赖关系在组织范围内广泛的蔓延 —— 这使公有云迁移策略变得极端复杂。这些依赖关系通常是服务到服务的集成，但也有数据集成，其中表面上分离的数据库却彼此相互连接。这将牵制任何迁移策略并常常使其复杂化。

在某些情况下，您可以高效的重构、或者重新开发，几乎所有其他云原生厂商都这样倡导。在另外一些情况下，您还可以通过优化应用运行时，更加智能的扩展应用平台，从而更好地利用计算资源。我们知道，在一个真实的企业中，大多数应用是不能被重新编写的，而只能优化应用运行时。VMware可为该场景提供独一无二的解决方案，因为我们有效地拥有来自计算、网络和存储的所有信息，可以实现无需更改代码即可优化应用平台的承诺。

这就是我们长久以来的核心优势，我们的客户已经对这样的事实习以为常，即他们可以在不修改任何代码的情况下将应用部署到vSphere上。能够在不需要更改代码的情况下转换应用运行时，以及构建能够交付此类特性的智能控制平面，也有助于弥补当前微服务实现的缺陷。许多微服务实现在跨网络大规模扩展时都会遇到延迟问题，因为单体应用中的内存调用被转化为跨网络的分布式调用。我们可以把这个特殊的控制平面看作是一个可感知应用的多云运行时；我们称它为HCR。稍后将对此进行详细介绍，但首先让我们进一步了解云原生运动及其与微服务的密切关系。

从云原生运动的早期开始就一直存在着一股动力，推动微服务架构的发展，提倡快速部署代码的能力（您将听到对“代码速度”的引证，使开发团队彼

此独立，降低组装一个大型单体代码部署单元的固有复杂性)。在此阶段也经常会涉及到容器运动，但是容器仅提供了抽象和打包应用运行时的能力，并没有解决可扩展性和性能问题。

接下来出现了容器调度的概念，例如 Kubernetes 和其他一些实现。但是容器调度无法观察和理解应用运行时，因此这些系统继续遭受可扩展性和性能方面的挑战。根本问题在于，在微服务中，过去的内存调用现在需要若干网络跳转，这些来自应用平台上的新型流量犹如喋喋不休的对话，令现有的负载均衡器不堪重负。

这个行业窘境引发了一个“新层次”的出现，该层被认为能够降低微服务调用中网络跳转的成本。像 LinkerD （https://linkerd.io/）和 Istio （https://istio.io/）这样的项目被称为服务网格（Service Mesh）层，VMware 可以很容易地实现这些层，因为我们在应用平台中拥有横跨计算、网络和存储的充分可视性。新范式导致复杂性的新变化。自此，微服务在网络上引发的过多会话带来了服务网格（Service Mesh）的概念，一个处理微服务间通信的负载均衡控制平面层，为微服务之间的通信提供可预测的服务质量（QoS）。

2.2. 微服务入门

在我们继续本文之前，理解微服务体系结构背后的一些基本概念至关重要。这里有很多规则，但是微服务的核心是一个具有发布良好、用途自明的公共接口的业务逻辑。微服务定义并不在于代码行数的多少，而在于业务的领域知识、服务的生命周期、变更次数以及如何最小化变更对应用平台中的其他服务的影响。在设计云原生应用平台时，业务领域与微服务之间的映射是一项必不可少的工作。这项工作将在不对其他正在运行中的微服务造成较大扰动的前提下加快业务逻辑进入应用平台的速度。有些人可能将这种现象称为“开发者速度”或“代码速度”，其中主要是关于新业务逻辑的快速部署。另一些人将其称为“延迟集成”，因为当调用开发团队准备好之后即可进行服务发现并相互调用（“集成”），开发人员之间无需为了集成代码的发布而争吵。图 1 显示了相互正交的两个设计要素之间的关系，即生命周期灵活性 vs 运行时性能。要进一步了解微服务，请参阅附录 1 —— 微服务架构概述。

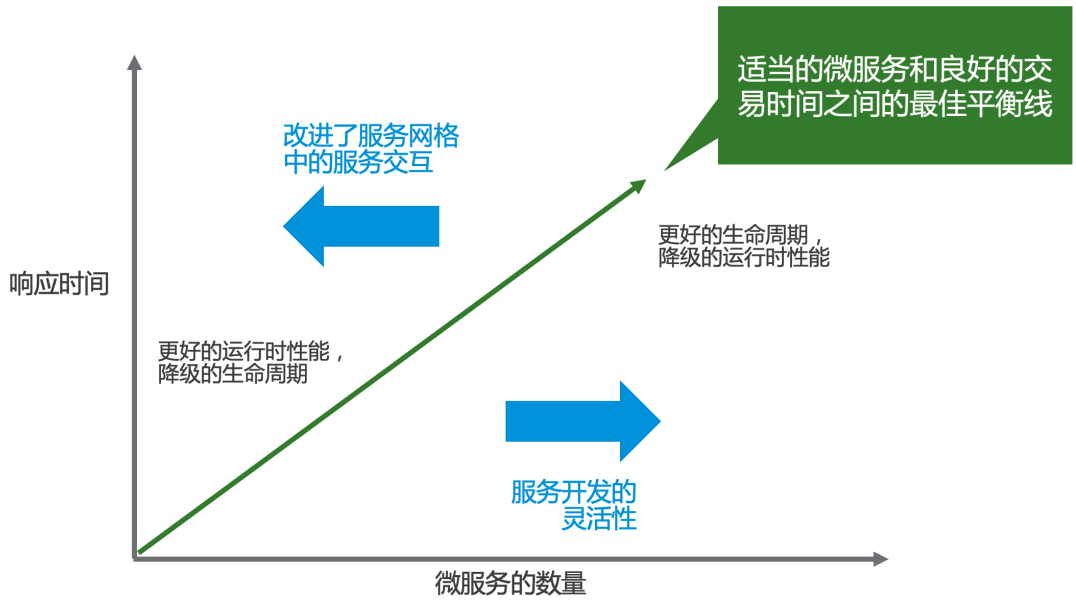

图 1 -微服务架构中生命周期和使用情况的阻抗关系

3. VMware 的观点

3.1. 企业中的云原生应用平台洞察 —— 应用使能型基础设施运动的开始

我们看到的第一个趋势是云原生应用（CNA）走过了它的成熟度曲线，从“一切都必须是无状态的”这一理想化和不切实际的立场开始，在曲线中间的某个位置转向“以无状态满足有状态”，以满足企业应用平台的切实需求。关于这一点有不同的观点，从早期某些布道者鼓吹的只支持无状态CNA的纯云原生，到标准的CNA平台也要允许有状态服务与无状态服务的交互。

实际上，我们发现CNA已经进入了企业实际应用阶段。这种转变花费了三年时间。显然，早期的布道者尝试了各种各样的模型，但随后受到了企业现实的限制。今天，在CNA平台中讨论无状态和有状态服务是很常见的，这无疑是最符合实际的方法。

第二个趋势是早期的CNA布道者提倡对企业中的一切进行重新编码，以遵循严格的CNA模式。一些布道者甚至提倡在任何地方都必须提供微服务。然而，这在企业中并不现实，因为重新编写系统的任何部分都会产生很高的成本。因此，“将某些部分重写为CNA，并尽可能让其他现有部分利用现代化平台”的趋势成熟起来，而不是像CNA早期所说的那样“重写一切”。这是一个“将某些部分重写为CNA，并尽可能让其他现有部分利用现代化平台”的双重故事。

企业中的许多人开始接受这种立场。这种方法有助于推动有机增长，最终将所有应用迁移到CNA平台成为可能。分为多个阶段的需求驱动方式要比一次性全部迁移更为可取。

我们还注意到业界实践CNA时遇到的另一个问题：客户在构建应用平台的过程中困难不断，在稳定性、部署和可扩展性等方面常常存在各种各样的挑战。我们接触过的客户中至少超过50%都面临着某种应用平台的挑战。更进一

步，我们不断的在私有云和公有云中看到，客户置备了过多的资源 —— 在某些情况下是所需的两倍甚至三倍。置备的资源远超所需，为什么会这样？

在过去的20年里，许多运动试图解决这种资源过度供应的现象，但我们仍然发现自己不断的重蹈覆辙，即每当应用表现不佳时就为其提供更多的资源。IT从业者正用他们的老方法解决新问题。**而这个问题的关键在于开发团队和运营团队之间的知识鸿沟。**

在图2中，我们展示了一个典型的组织，其中业务发起方与技术平台的领导者进行交互，进而影响整个组织以交付特定的应用服务。例如，业务发起方提出需求，技术领导层将对可行性进行初步评估，然后将其移交给开发团队。最终，这些需求的业务流程映射到一组IT服务，IT服务则映射到所需的应用组件、数据、操作流程和基础设施上。***这是将应用组件及其运行时映射到运维/基础设施的最后一步，大部分沟通错误都发生在此。这正是导致"在置备了两倍于所需的资源时，50%的应用却不能满足SLA"现象发生的根本原因。***

在这个领域中，开发人员使用一种语言，而操作团队使用另一种语言，反之亦然。这种根植于旧思想的实践需要被改变。我们发现，这种改变可以被云原生运动所推动，我们看到一些新类型工程师能够在开发人员和运维人员之间使用一种共同的语言。我们将这些新类型工程师称为平台架构师或应用平台架构师。

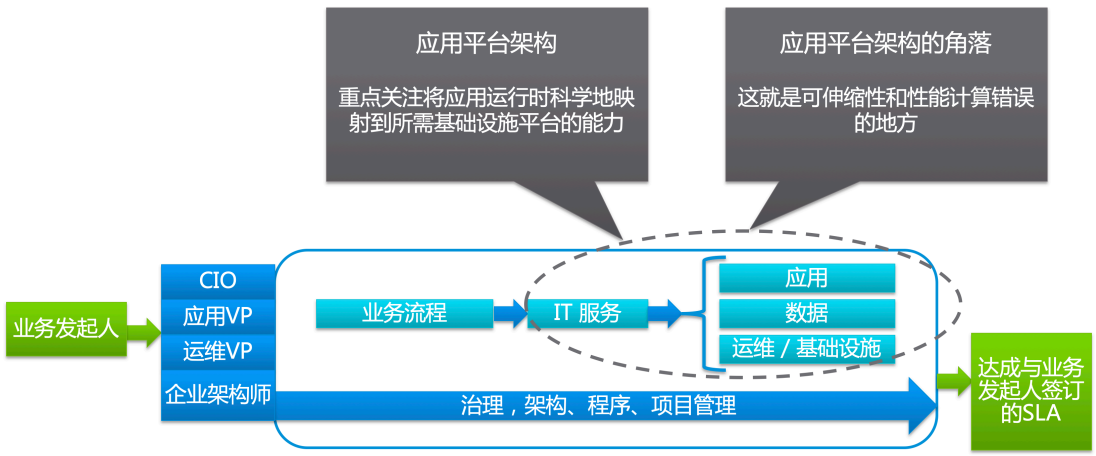

图 2 -典型的企业应用平台组织模式

要理解这个映射分解的背景，明确所涉及的各种制品（Artifacts）至关重要。开发人员首先编写代码，然后打包代码，如图3所示。本例中的包示例是“.war”文件。因此，这里展示的应用运行时很可能是一个应用服务器，比如Tomcat。然后应用运行时依次部署在虚拟机上。***这些应用运行时在应用的执行和可扩展性中起着至关重要的作用。***大多数配置文件或属性集允许您扩展线程数量、分配内存容量和设置垃圾回收算法。大多数开发人员都会忽略应用运行时及其可扩展性属性的重要性。事实上，在许多情况下，他们认为可以持续的扩展实例而不带来弊端。然而，弊端是显而易见的。

另一方面，运维工程师不知道这些应用运行时的存在，或者对它们知之甚少。因此，我们的情况是开发人员拥有应用运行时的一半所有权，而运维工程师只负责他们的SLA。事实上，这不是一件好事。这种错误的沟通机制导致了世界上一半的应用在可扩展性和性能方面表现不佳，甚至在置备了超过两倍所需的资源后依然如此。

正如您在图3中所看到的，您可以对这些流程进行纵向扩展和横向扩展(这里的“流程”等同于应用运行时)。显然，横向扩展需要额外的许可和管理成本，而纵向扩展可能是一种成本效益更高的解决方案，能够以最小的开销提供更多的处理容量。我们之所以强调这一点，是因为我们讨论的正是应用运行时的配置及其重要性。

很多时候，我们发现运维人员会为虚拟机置备更多的资源，但应用运行时并不会自动改变配置以利用新增资源。在本例中，您可以看到在虚拟机级别拥有足够的计算资源，但是应用运行时没有充分利用它。这就是资源膨胀发生的地方(置备两倍以上所需资源的本质原因)。性能差并不是由于资源膨胀，而是由于应用运行时没有通过自动改变配置而表明它可以支配更多的线程和更大的内存，从而实现纵向扩展。许多系统将通过过度扩展实例来应付这种糟糕的性能，从而导致严重的基础设施不充分利用和糟糕的应用性能表现。

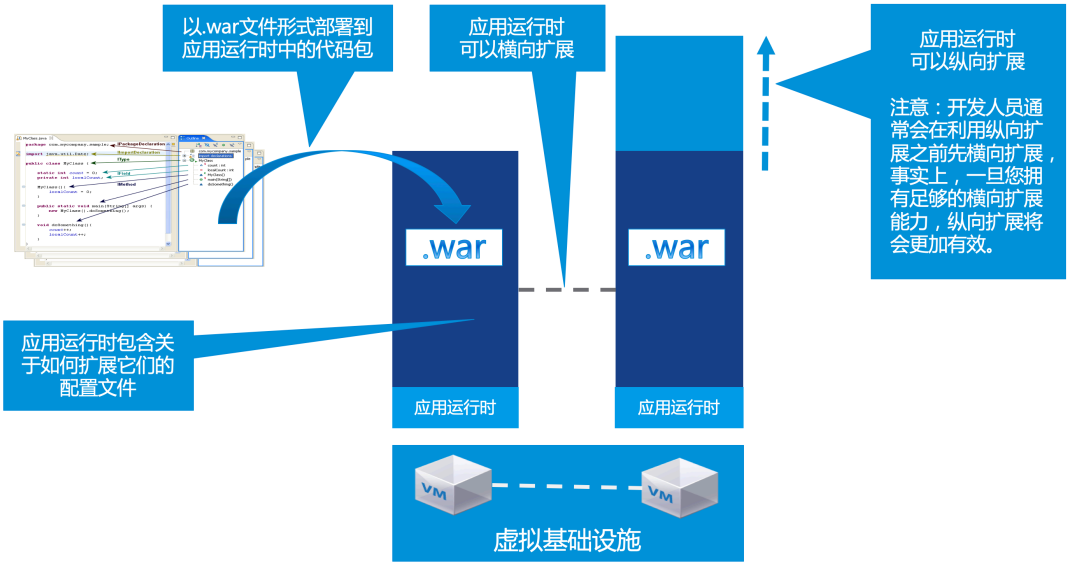

图 3 -开发制品和应用运行时

显然，从上面关于资源膨胀的讨论中可以看出，我们正处于一个难以为继的过程中。CNA正试图通过塑造跨领域的新型工程师来应对这一挑战：缩小应用平台的开发和运维知识之间的差距的工程师。首先，让我们更仔细地检视一下这种新型工程师。

3.2. 应用平台时代的到来

见图4，IT企业与***应用平台架构师(Application Platform Architect，简称APA)***的概念叠加。APA从事过广泛的软件开发，研究过如何纵向和横向扩展应用运行时，并且对基础设施平台有很好的理解。APA是对应用和相关应用平台的可扩展性和性能问题进行过深入研究的人员。

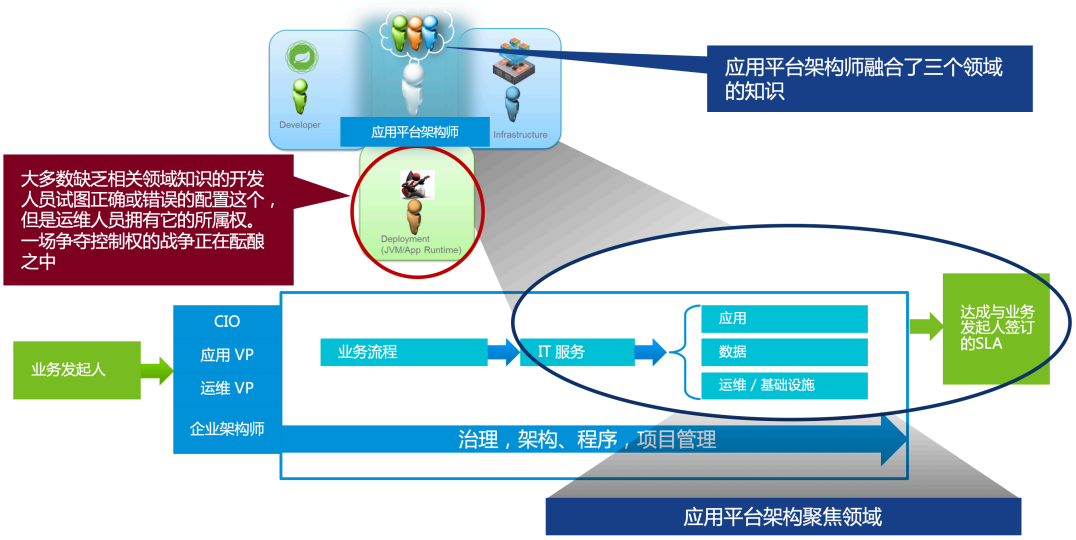

图 4 -应用平台架构师 —— CNA 时代的新角色

举例以帮助更好的理解什么是 APA。X 公司的业务是将交易平台作为服务出售，在最新的迭代中，它们有多个组件：消息中间件、基于 Java 的分布式服务和 GemFire 数据平台（内存数据库）。该平台的核心是内存数据库。GemFire 是一个基于 Java 的分布式数据库，在本例中运行在 VMware vSphere 上。因此，你需要一个对这三种技能有相当了解的人，或者至少是这三种技能中的两种。当这个平台首次发布时，开发人员总是参与这个平台的运维。这给开发工作带来了巨大的压力；然而，后来他们成功地培训和雇佣了 APA，基本上重新组建了现有的 DevOps 团队。他们现有的 DevOps 团队在应用平台的自动化部署方面有很好的技能，但是缺乏可扩展性和可靠性技术方面的技能。在一些情况下，我们训练他们现有的工程师成为优秀的 APA。在另一些情况下，他们可以雇佣最接近 APA 概念的角色，如 SRE。在这种团队中流行的说法是“……我们将 SRE 放回到 DevOps 中。”这表明 DevOps 的基本原则是 SRE，但经过多年的发展，DevOps 已经偏离了对每个人的所有意义，更多的是关于 CI/CD，而不是其关于可靠性的核心基本原则。他们面试的很多 DevOps 工程师并不具备 SRE 那样的可扩展性和性能背景 —— SRE 是业界影响 DevOps 的因素之一，它影响了某些厂商使用 CI/CD 工具的方向。

以下是X公司的APA/SRE日常工作的清单：

1. 他们负责交易平台的性能和可靠性。
2. 他们与来自开发和基础设施的跨领域高级工程师们合作，与之形成密

切关系，学习他们需要掌握的日常事务。这一点很重要，因为他们经常发现自己需要与多个组织和流程打交道，以解决平台方面的特定挑战。

3. 交易平台由40个虚拟机和其上的各种软件组件组成。为此，他们完全自动化了这些环境的创建和回收。他们有很多这样的环境，所以SRE完全自动化了交易平台的创建和回收工作。这是支撑他们的敏捷高频迭代开发团队的最低要求，因为每天都有多个创建这样一个交易平台的请求。
4. 他们着眼于最可行的方式来扩展平台，无论是一定数量的、健康的横向扩展，还是纵向扩展和横向扩展结合的策略。满足性能要求的可扩展性是他们的日常工作目标，因为作为一家公司，他们通过对交易量驱动的交易平台上的交易收取费用来盈利。
5. 他们的平台在执行交易的运行时的可靠性和可预测性上有很高的标准，无论交易量有多大。如果他们超过了约定的交易响应时间，就要向客户支付罚金（使用该交易平台的全球股票交易所都是他们的客户）。
6. 如果存在性能挑战，他们负责从代码和基础设施平台的角度进行调查。通常，使用通过获取线程转储、SAR报告、堆转储和许多其他度量进行故障排除，他们可以缩小问题的范围，必要时会通过组织团队会议或者建立作战室来解决问题。SRE团队将负责解决大多数生产问题，但如果他们需要帮助，周边团队也会参与其中。
7. 他们与软件工程师一起为新的应用设计运行时架构，帮助他们理解平台的功能，以及什么是可行的，什么不该是从纵向扩展和横向扩展的角度来考虑的。例如，哪些组件需要亲和性规则或反亲和性规则（亲和性意味着某些组件需要就近运行，而冗余配置的组件则不能在相同的虚拟机或物理主机上运行，即反亲和性)。
8. 它们创建各种平台和基础设施抽象层以及基础设施/平台API，以帮助开发人员编写智能的云原生应用。例如，如果某个平台发生故障，应用可以感知故障并做出修正。
9. 他们创建了非常高级的度量标准，可以显示交易量和每个交易的成本，并据此持续改进应用平台。他们还对花在管理、跟踪和修复可靠性问题上的时间进行了进一步的度量（交易平台业务分析师使用这些丰富的平台数据来帮助进一步分析业务的成本与利润）。

让我们进一步明确每个角色的关键技能是什么：在开发背景方面，在APA的职业生涯中要有重要的软件工程经验，曾经为重要的应用/服务写过代码，至少精通一门现代编程语言，如Java、JavaScript、Node.js、Ruby、Scala、Python或Go。这至关重要。同样重要的是，他们不仅作为开发狭义产品特性的纯软件工程师，而且还应涉及多个组件/平台层领域 —— 通常是在每个企业IT资产中都可以找到的组件/平台层。他们熟悉的系统应该包括负载均衡器、Web服务器、应用服务器、中间件、消息传递系统、内存数据库、传统关系数据库、操作系统、容器和虚拟化。最重要的是，作为软件工程师，他们能够处理常见的生产可靠性和可扩展性问题。最好的软件工程师是那些编写和学习如何在生产环境中最最好的运行和扩展代码的人。

3.2.1. 应用平台架构师概述

在图5中，我们展示了应用平台架构师新角色所需的三个主要技能类别：开发背景、在生产环境中部署和扩展应用的背景，以及对承载应用平台的基础设施堆栈的深刻认知。

图 5 -应用平台架构师的三合一技能

部署/应用运行时背景是一个关键部分，也是最难找到的技能组合。事实上，大多数组织甚至不了解这种技能的存在。在过去的十年里，我们在VMware建立了一个专门的技术研讨会，指导团队往应用平台上思考。我们通常同时与SRE、VMware架构师、平台/基础设施架构师、开发人员和应用所有者等混合受众一起开这个研讨会。这是有意而为之的，因为我们想要识别每一层解释中

的差距，并实时整体地解决它们。研讨会的议程包括我们在本文中介绍的材料，但也有一个深入的交互式会议，在那里我们审查现有的应用平台并提出设计改进建议。***当看到我们如何进行跨应用开发、应用运行时和基础设施团队的讨论以实现更好的设计时，大多数客户表示，他们希望在他们的组织中构建一个类似的应用平台团队。***

3.3. CIO 们说 “我们想退出基础设施业务” 是什么意思?

CIO并不想退出基础设施业务，他们主要是想避免前面提到的半数应用不能满足SLA和超过需求两倍的资源置备问题。他们想要处理的是这个不可持续的模式。***如果您不完全理解应用的可扩展性范式，那么从私有云迁移到公有云也不能解决由糟糕的扩展模式带来的底层基础设施膨胀。***注意：可扩展性模式指的是理解何时进行纵向扩展和何时进行横向扩展，以及需要扩展多少实例才能使应用平台满足SLA。

充分理解上述可扩展性模式后，CI/CD是下一个要解决的问题，目的是快速的测试和集成、并持续构建和部署应用平台。您会发现开发人员将此称为“开发者速度”。“我们指的是快速、准确地搭建平台的能力。当然，根本目的是促进开发团队服务交付的自主性和敏捷性，不要被较慢的团队拖累，因此催生了微服务架构。

已经获得应用平台开发者速度的 CIO 和 CTO 们将平台的重要性从第 1 层转移到了第 0 层，第 0 层意味着它对业务至关重要，如果没有它，业务就会失败。这种由云原生运动驱动的开发者速度，正在将向 CFO 汇报的战术型 CIO 转变为向 CEO 汇报的战略型 CIO。

4. 建议

当我们在几年前讨论SDDC时，我们实际上是在说软件定义应用平台（Software Defined Application Platform，简称SDAP)），一个可感知应用负载的的多云运行时公共层。该层可了解应用工作负载行为，例如在各种云上的运

行成本、在私有或公有云上的规模效益，以及与其他服务的数据亲和性要求。这是一个智能控制平面，具有应用负载感知能力，我们相信这是真HCR的本质。

请记住，如果没有交付所有这些特性，那只是在构建不知道应用工作负载需要什么的通用基础设施层。每一组通用硬件都需要特定的配置以适应客户的应用负载，虽然工作负载可以有许多种类型，但是大多数情况下可以将它们归类为十余种。有巨大的机会去开发和利用公共多云运行时的潜能，其上的应用开始变得更像是某种网络，要靠构建专门的应用负载感知功能来发挥其价值。

我们看到了构建多云应用平台的趋势，但是这种平台不能简单地以某种特定方式拆分到两个独立的云。正如一位客户所说，“在私有云和公有云前面放置负载均衡器并不能构成混合云或多云系统。这种事我一天就可以完成。我需要的是两者之间的公共运行时。因此，我们必须考虑建设延伸于多云之上的公共层。这个公共层可以称为HCR，作为应用平台运行时、服务网格、应用工作负载行为仿真、VMware HCX[1]-like工作负载迁移服务、软件定义计算、网络和存储，分布式跟踪（使用我们的管理产品，如Wavefront[2]）等特性的核心引擎。您可以将这些层看作是SRE -as-a- service （SREaaS），在这里，SRE可以完全自动化保证应用平台优化、持续提供实时分析数据、做出智能安置和决策优化所需的全部工作。现在的SRE也许是手工作业，但是他们当前的工作将完全由这些专门的层（类似于HCR的层）自动化，这是一个以服务网格为基础的应用平台运行时，如图6所示。在图中，我们还展示了一些专门的控制器，如可预测的响应时间控制器：用于在特定范围内优化服务间调用的响应时间；拓扑映射器：用于生成服务之间的调用图和确定服务的运行位置，以及用于清理计算空间碎片的去重控制器。

[1]http://hcx.vmware.com

[2]http://www.wavefront.com

图6 - HCR是应用平台的核心运行时

在图7中，我们展示了一个应用平台的功能图，该应用平台包含云原生基础设施层、CI/CD、服务网格、安全和策略管理，以及分布式跟踪功能，这些功能将有助于驱动SREaaS计划。这个应用运行时也适用于企业的非云原生产品，我们认为真正的联邦服务网格层可以在云原生服务和非云原生服务之间做编排。

图 7 -应用平台功能图

4.1. HCR 与服务网格

图7显示了带有关键功能的服务网格层(注意：在图6中，我们展示了服务网格是HCR的核心功能)。其中一些功能在业界得到了很好的推广，而另一些功能则是基于我们的经验发现的，即我们认为在典型的微服务云原生应用平台中，服务网格应该补充哪些功能。以下是服务网格的功能列表：

服务QoS负载均衡器： 这是任何服务网格实现的核心。随着服务网格规模的增长，许多微服务实现的响应时间会延长，因此需要一个智能的工作负载/服务感知负载均衡器来帮助优化路由和减少延迟。这对于使用“跨服务路由的可预测响应时间”和“服务到服务的协同定位和服务vMotion”模式来实现预期行为的功能非常关键。负载均衡器至少可以为HTTP、gRPC、WebSocket和TCP提供流量优化。

服务边车代理：对“Sidecar”的引用来自这样一个事实，即它是一个附加到主服务上下文的支持服务。就像摩托车的边车一样，它可以扩展摩托车的功能。要创建高度智能化的工作负载/服务感知控制层，必须能够以非硬编码的方式注入大量的服务网格自检和优化信息。开发人员经常会使用一些可以提高服务性能的库，但是需要在代码中硬编码特定的调用，这通常会将主要业务逻辑代码与管道代码混杂在一起。而服务边车代理则通过拦截调用并向调用添加所需的优化信息来减轻这种负担。所有这些都是在运行时进行的，没有任何特定的硬编码代码调用。

由于许多微服务实现未能满足其性能、规模和可靠性目标，服务网格在市场上获得了快速的发展。聚焦于可以让微服务开发者获得可控SLA的明确功能领域至关重要，但是下面我们将跳过负载均衡、安全、扩展控制器、冗余和边车代理等常见的功能领域，转而关注新领域。

服务复制器：服务网格可以轻松地创建各种可用区，而且可以基于用户注入的配置，通过将服务复制到其他可用区来维护服务的反亲和性。在大多数情况下，复制可以很简单。在有些情况下，用户可能需要提供服务依赖关系，以复制服务网格的子分支。主要目标是将复制行为从主代码中抽象出来，并保存

在关于应用复制逻辑的服务网格元数据配置中。

服务断路器：就像电路一样，当达到一定的电流水平时，断路器会跳闸，不再让电流通过。类似的，微服务也可以用断路器模式保护，该模式用服务调用断路器来封装微服务。断路器模式检测SLA的退化趋势，可以直接中断所有对被保护微服务的调用并报错，也可以在错误处理程序中添加额外的逻辑来应对过载。在大多数情况下，基本实现只是简单地切断对服务的后续调用。但在更高级的功能中还可以实现级联关闭行为。这意味着如果一个服务被关闭，它的相邻服务或相关服务也会被关闭。Finagle （https：//twitter.github.io/finagle/）和Hystrix（(https：//github.com/Netflix/Hystrix)是这种模式的早期示例。

服务发现和注册：在任何大规模且不断增长的服务网格部署中，知道网格中有哪些服务是一切的开始。更重要的是，要拥有一个全局性的、可信的，可配置策略的服务注册机制。

服务扩展器：这是一个服务网格边车流程，用来侦听特定的服务执行指标，然后确定哪些服务需要扩展。这似乎是一个简单的问题，但实际上，它要求能够对执行数据进行长期采样，实现启发式的更改并不断学习和优化决策。服务网格层中的许多领域都严重依赖于服务扩展器功能，尤其是实现跨服务路由的可预测响应时间、服务到服务的协同定位和服务vMotion。

服务故障补偿器：典型的SRE/APA处理大量的日常服务故障时，需要执行许多手动步骤。SRE可以编写许多这样的处理程序并将其加入服务网格，以便在发生特定事件时由补偿器处理程序回调。服务网格的精髓在于其部分功能由供应商进行特定的基线开发，然后借助用户加入的许多插件/补偿处理程序实现快速进化。大量的手动步骤被自动化所取代正是是服务网格的巨大价值之所在。

服务滚动更新：该功能提供了不中断业务执行的服务实例更新能力。

5. 结论

越来越多的开发人员开始着手构建满足特定工作负载需求、并内置IaaS能力的应用平台。当开发人员构建新的云原生平台时，他们很快就意识到他们的系统并不完整、缺乏真正的规模效益和效能，因为大部分企业应用并没有在他们的云原生平台上运行。这正是VMware可以提供帮助的地方。VMware不仅可以开发云原生平台，还可以开发一个承载所有企业应用并能够对应用工作负载的性质进行智能感知（云原生应用和其他应用[3]）的整体平台。这可以帮助开发人员解决性能和可扩展性难题，为他们提供高度可配置的应用运行时，比如HCR。

[3] https://blogs.vmware.com/cloudnative/products/

6. 附录 1 -简要介绍微服务架构

例如，在XYZcars.com，公司维护一个帐户信息服务和相关的业务逻辑，如图8所示，它是一个业务领域的功能，映射到组织内的多个帐户实体。此外，帐户信息及其业务功能与其他微服务之间的映射在应用平台中被当作元数据进行维护，以帮助确定相关变更造成的影响。每个微服务都有一个服务名称、服务端口、公共接口和配置/元数据。

注意：组织中的微服务本质上是将单体的业务功能分割成更小的自包含逻辑块，如果这些逻辑块封装得很好，则其变更就不会引起连锁反应，从而加快了开发速度，这就是微服务的根本意义之所在。***但是这还意味着，以前从一个业务功能到另一个业务功能的内存调用，现在不可避免的变成了网络跳转，这意味着性能将受到影响，而且我们经常发现，在微服务实现中，这些负载均衡器不善于处理的流量会激增，这就需要转向一种新的思维方式。***我们常说的复杂性系统已经被微服务架构改变，从N个数量的内存调用到N个数量的网络跳转，这种复杂性必然会带来专门的控制层，如平台即服务、应用平台和服务网格，已进行协同工作、优化路由（最小化过多的网络跳转的影）、服务注册、服务发现、自动扩展，滚动更新、和各种QoS功能。当然，这还意味着如何理解和设计企业的组织结构，让复杂通信、审批路径、跨团队测试和复杂的部署并行不悖。

微服务的主要特性如下所述，如图8所示：

- **服务名称**是AccountService，它将其业务逻辑封装在AccountService.jar文件中，该文件又封装在AccountService.war文件中。war文件将被部署到servlet容器（例如Tomcat）应用服务、或Spring Boot框架中，然后通过REST API调用它。

- **服务端口**，这是客户端用来调用帐户服务的端口

- **公共接口**，接口是一种契约，它定义了微服务上可用的API，任何调用

它的客户端都需要遵守契约。这种契约有助于促进客户端与服务或服务之间的快速集成。现在我们使用的术语是“客户端”，但是客户端可以是浏览器、前端、中间件层或任何调用帐户微服务的其他微服务，通常是促进与其他调用系统通信的中间件层。但是，如果该服务具有定义良好的契约，其他自动发现帐户服务的服务可以立即理解如何使用该微服务，而无需过多考虑其内部细节。这还将服务绑定到契约上，并在理想情况下防止将来的契约破坏，因此使用该接口的调用服务可以大体上确信契约不会被破坏。有时服务接口会由于初始设计不佳而更改，因此可能会影响调用AccountService的客户端或其他服务，但至少对代码进行快速重构可以修复这些问题。元数据和配置信息可以在影响最小化方面发挥重要作用。

- **注意**：如果你需要改造SOA应用中依赖SOAP访问协议的旧代码，不要为了避免对客户端代码造成影响而用REST代理SOAP调用，最好是直接使用原生REST调用来避免服务代码层改造的痛苦，并从你的代码库中删除所有SOAP相关的代码。

- **服务配置**，这是放置某些动态运行时属性的地方，例如运行服务的生产环境或测试环境的配置信息，或者继承自配置服务的配置信息，在重新部署和生命周期中对其他服务的依赖关系，如持久化数据库服务。还有一些与QoS有关的特性，如服务配置描述了它允许的服务执行时间，也会被注册到应用平台。如果超过此时间，则触发事件处理程序以进行适当的补救。还可以考虑一些其他高级选项，如服务需要多少CPU、内存和其他计算资源，但大多数人倾向于暂时忽略这些属性，并将它们配置到容器或容器调度程序中。

- *以上微服务的属性如图8所示，该图展示AccountMicroservice在AccountServcie.jar文件中驻留业务逻辑，这个jar文件进一步被封装在一个war文件里，以便部署在一个允许REST通信的Servlet容器中，比如SpringBoot。*

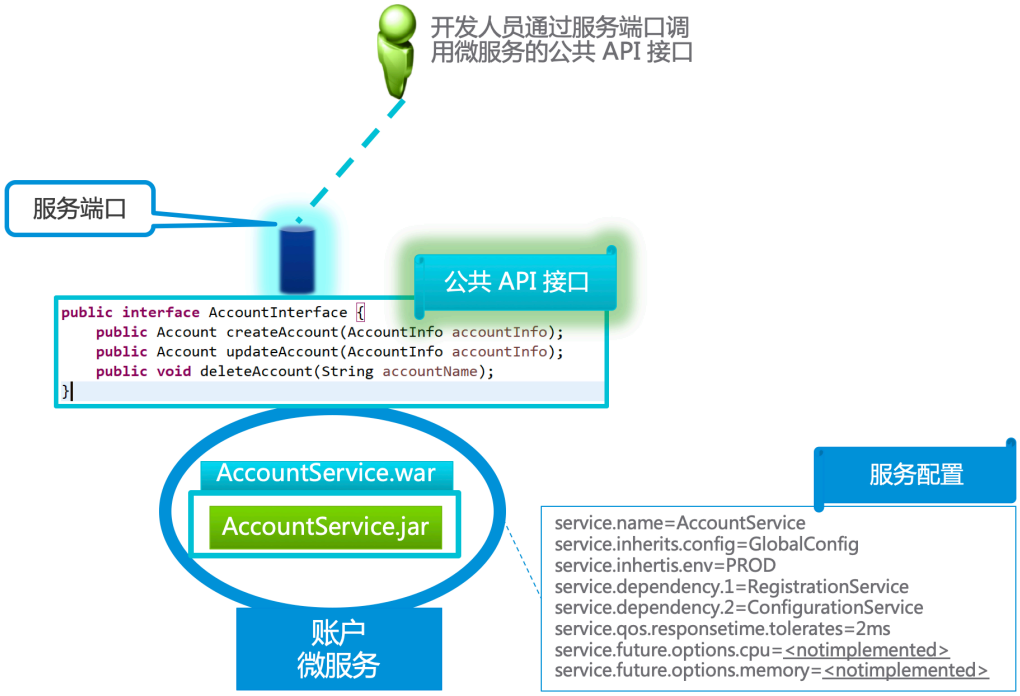

图 8 -微服务的主要属性

7. 致谢

作者感谢以下评审人员的宝贵反馈：

Greg Lavender,	Senior VP and CTO Cloud Architecture OCTO
Pere Monclus,	CTO NSBU
Tom Hite,	Sr. Director of Professional Services Emerging Engineering
John Blake,	Director Cloud Architecture OCTO
Roman Tarnavski,	Principal Architect，APJ OCTO
Michael Francis,	Principal Architect，PSO
Andrew Babakian,	Principal SE
Mitesh Pancholy,	Principal Architect，PSO
Niran Even Chen,	Staff SE
Michael Gasch,	Application Platform Architect OCTO (K8s Specialist)

Author:

Emad Benjamin,	Sr. Director and Chief Technologist of Application Platforms