

持续集成/持续集成

和

库伯内特斯

新堆栈

使用 Kubernetes 的 CI/CD

Alex Williams, 创始人兼总编辑

核心团队：

Bailey Math, AV 工程师

Benjamin Ball, 营销总监 Gabriel H. Dinh,

执行制片人 Judy Williams, 文案编辑 Kiran

Oliver, 播客制作人 Lawrence Hecht, 研究总

监 Libby Clark, 编辑总监 Norris Deajon, AV

工程师

© 2018 新堆栈。版权所有。

20181112

# 目录

介绍 .....	4
赞助商.....	7
贡献者 .....	8
使用 KUBERNETES 的 CI/CD	
DevOps 模式.....	9
KubeCon + CloudNativeCon :目前还没有最好的 Kubernetes CI/CD 工具.....	39
云原生应用程序模式.....	40
Aqua Security :通过 CI/CD 自动扫描图像提高安全性.....	61
使用 Spinnaker 持续交付.....	62
谷歌云 :在 Kubernetes 上使用 Spinnaker 实现 DevOps 的新方法.....	88
云原生时代的监控.....	89
CloudBees :使用 Jenkins X 和 Kubernetes 的 CI/CD .....	115
关闭.....	116
披露 .....	118

# 介绍

Kubernetes 是首选的云编排器。它的核心就像一个蜂巢：编排容器，调度，作为自我修复集群的声明性基础设施。随着其功能以如此速度增长，Kubernetes 的扩展能力迫使人们质疑组织如何管理自己的团队和采用 DevOps 实践。

从历史上看，持续集成成为 DevOps 团队提供了一种将应用程序投入生产的方式，但现在持续交付变得越来越重要。如何实现持续交付在很大程度上取决于使用分布式架构来管理复杂和快速的基础架构上的服务，这些基础架构使用计算、网络和存储来提供持续的按需服务。开发人员将尽可能贪婪地使用服务，以充分利用它们。他们将尝试开发、部署以及越来越多地管理微服务及其整体健康和行为的新方法。

Kubernetes 类似于其他大型、云软件项目，它们非常复杂，只有在实施时才能确定其价值。容器编排技术越来越多地被用作由 DevOps、持续交付和可观察性的联合力量定义的应用程序部署平台。当一起使用时，这三种力量可以更快、更有效地交付应用程序，并且更接近客户的需求。团队首先在基于容器的云原生架构中将应用程序构建为一组微服务。但是 DevOps 实践才是真正改变组织应用程序架构的东西；它们是使应用程序在 Kubernetes 上运行的所有模式和实践的基础。而 DevOps 转型只会伴随着组织的价值观与其开发应用程序架构的方式保持一致。

在这种新优化的云原生转型方式中,Kubernetes 是推动者 它不是一个完整的解决方案。您的组织必须实施最适合您自己的业务需求和结构的工具和实践,以实现这个开源平台的全部承诺。 Kubernetes[项目文档](#)本身是这样说的:

---

Kubernetes “不部署源代码,也不构建您的应用程序。持续集成、交付和部署 (CI/CD) 工作流由组织文化和偏好以及技术要求决定。”

这本电子书是 The New Stack 的 Kubernetes 生态系统系列的第三本也是最后一本,它为理解和构建团队的实践和管道以在 Kubernetes 上交付和持续改进应用程序奠定了基础。这是怎么做的?这不是一套规则。它是一组流入组织并影响应用程序架构开发方式的实践。这就是 DevOps,它的潮流现在深入到具有现代应用程序架构的组织内部,通过持续交付得到体现。

## 部分摘要

·第 1 部分:由 ReactiveOps 的 Rob Scott 撰写的 DevOps 模式探讨了 DevOps 的历史、它如何影响云原生架构以及 Kubernetes 如何再次转变 DevOps。本节将追溯 Docker 和容器打包的历史,直至 Kubernetes 的出现,以及它如何影响应用程序开发和部署。

·第 2 节:云原生应用程序模式由 Janakiram & Associates 首席分析师 Janakiram MSV。它回顾了 Kubernetes 如何自动管理资源分配,根据

DevOps 团队制定的政策。它详细介绍了关键的云原生属性，并将工作负载类型映射到 Kubernetes 原语。

·第 3 部分:Kenzan 工程高级副总裁 Craig Martin 的 Spinnaker 持续交付分析了使用云原生技术进行持续交付如何需要更深入地了解 DevOps 实践，以及这如何影响组织部署和管理微服务的方式。 Spinnaker 作为一种新兴的 CD 工具受到特别关注，它本身就是一个云原生、基于微服务的应用程序。

·第 4 部分:来自 Container Solutions 的工程师团队在云原生时代的监控，解释了微服务日益复杂的情况如何更加强调结合传统监控实践以获得更好的可观察性的需求。他们在编排环境中的容器上运行的横向扩展应用程序定义了可观察性，特别关注作为新兴管理工具的 Prometheus。

虽然本书以可观察性为结尾，但越来越清楚的是，云原生监控并不是应用程序开发生命周期中的一个端点。相反，它是细粒度数据收集和分析的过程，它定义了模式，并在改进和交付的持续循环中从头到尾通知开发人员和运营团队。同样，本书旨在作为整个规划、开发、发布、管理和改进周期的参考。

# 赞助商

我们感谢电子书基金会赞助商的支持：



以及我们对这本电子书的赞助商：



## 贡献者



[罗布·斯科特](#)作为 ReactiveOps 的站点可靠性工程师,他在查塔努加的家中工作。他帮助为多个客户构建和维护高度可扩展的、基于 Kubernetes 的基础设施。自 2016 年以来,他一直在使用 Kubernetes,一路为官方文档做出贡献。在构建世界级基础设施之余,Rob 喜欢与家人共度时光、探索户外以及就 Kubernetes 的所有方面进行演讲。



[贾纳基拉姆 MSV](#)是 Janakiram & Associates 的首席分析师和国际信息技术学院的兼职教员。他还是 Google 合格云开发人员和 Amazon 认证

解决方案架构师、开发人员和 SysOps 管理员是 Microsoft 认证的 Azure 专家,也是首批获得认证的 Kubernetes 管理员和应用程序开发人员之一。他之前的工作经历包括 Microsoft、AWS、Gigaom Research 和 Alcatel-Lucent。



[克雷格·马丁](#)是 Kenzan 的高级工程副总裁,他帮助领导公司的技术方向,确保探索新技术和新兴技术并将其纳入战略愿景。最近,克雷格一直专注于通过构建大型微服务应用程序来帮助企业进行数字化转型。在加入 Kenzan 之前,Craig 是 Flatiron Solutions 的工

程总监。



Ian Crosby、Maarten Hoogendoorn、Thijs Schnitger 和 Etienne Tremel 是[容器解决方案在 Kubernetes 上应用程序部署的工程师和专家](#),为进行云迁移的客户提供支持的咨询组织。

# 开发模式

罗伯·斯科特

DevOps 实践帮助开发人员和工程师创造了一个空间,以构建优化资源  
**DevOps 实践深深植根于现代应用程序架构中。**

通过持续交付实践构建应用程序架构。云原生技术利用容器的效率来构建比组合或单体环境更有用和适应性更强的微服务架构。组织在构建基于微服务的云原生应用程序时正在转向 DevOps 原则。 DevOps 和云原生架构的结合正在帮助组织通过促进能够快速适应市场变化的简化、精益的产品开发流程来实现其业务目标。

云原生应用程序基于一组松散耦合的组件或微服务,这些组件大部分在容器上运行,并使用 Kubernetes 等编排引擎进行管理。

但是,它们也开始在无服务器架构中作为一组离散函数运行。服务或功能由开发人员和工程团队定义,然后由越来越多的跨职能团队不断构建、重建和改进。运营现在不太关注

基础架构以及运行轻型工作负载的应用程序的更多信息。  
综合效果是对产生更高效率的自动化流程的塑造。

事实上,有些人会争辩说,一个应用程序不是真正的云原生,除非它背后有 DevOps 实践,因为云原生架构是为网络规模计算构建的。 DevOps 专业人员需要构建、部署和管理安全、有弹性和高性能的声明式基础架构。使用传统的孤立方法无法满足这些要求。

作为云原生应用程序事实上的平台,Kubernetes 不仅处于这一转变的中心,而且还通过抽象出底层计算、存储和网络资源的细节来实现这一转变。该开源软件提供了一个一致的平台,容器化应用程序可以在该平台上运行,而不管它们各自的运行时要求如何。使用 Kubernetes,您的服务器可以变得很笨 它们不关心它们在运行什么。多个应用程序可以分布在同一组服务器上,而不是在特定服务器上运行特定应用程序。 Kubernetes 简化了应用程序更新,使团队能够快速将应用程序和功能交付到用户手中。

然而,为了在 DevOps 上取得成功,企业必须有意识地决定构建云原生应用程序。将 DevOps 付诸实践所需的组织转型只有在业务团队愿意投资于 DevOps 实践时才会发生 转型伴随着产品团队在应用程序开发中的一致性。这些团队共同创造了不断将技术开发改进为精简、简化的工作流程所需的环境,这些工作流程反映了基于 DevOps 原则构建的持续交付流程。

对于使用容器编排技术的组织,产品方向是通过开发微服务架构来定义的。只有当组织了解 DevOps 和持续开发流程如何支持创建最终用户真正觉得有用的应用程序时,这才有可能实现。

这就是挑战所在:您必须确保您的组织已准备好改变产品团队所有成员的工作方式。

最终,DevOps 是一个故事,讲述了为什么您首先要进行精简的精益产品开发这与您迁移到 Kubernetes 之上的微服务架构的原因相同。

本章的作者是[ReactiveOps 的站点可靠性工程师 Rob Scott](#)。 Scott 是 DevOps 实践方面的专家,他运用所学技术帮助客户运行可在 Kubernetes 架构上扩展的服务。他在构建横向扩展架构方面的专业知识源于多年的经验,这些经验让他见证了:

- 容器如何将开发人员和运维人员聚集到DevOps 领域。
- 像Kubernetes 这样的容器编排工具在容器生态系统中所扮演的角色。
- Kubernetes 如何导致整个DevOps 生态系统的革命性转变 这最终改变了企业。

容器之前的传统 DevOps 模式需要不同的流程和工作流。容器技术是从 DevOps 的角度构建的。抽象容器正在影响我们如何看待 DevOps,因为传统的架构开发随着微服务的出现而发生变化。这意味着遵循跑步的最佳实践

Kubernetes 上的容器,以及将 DevOps 扩展到 GitOps 和 SecOps 实践中。

## DevOps 和 CI/CD 的演变 图案

### DevOps 简史

DevOps 诞生于大约 10 年前,尽管近年来组织表现出更大的兴趣。根据 Forrester Research 的数据, 2017 年接受调查的组织中有一半实施了 DevOps 实践,它宣布 2018 年为“企业 DevOps 年”。尽管 DevOps 是一个广泛的概念,但其基本思想涉及开发和运营团队更紧密地合作。

传统上,软件开发和部署的速度不允许在不同团队中工作的工程师和操作人员之间有很多时间进行协作。许多组织已经采用了精益产品开发实践,并且一直承受着快速发布软件的压力。开发人员将构建他们的应用程序,而运营团队将部署它们。两个团队之间的任何冲突都源于核心脱节 运营团队不熟悉正在部署的应用程序,而开发团队不熟悉应用程序的部署方式。

因此,应用程序开发人员有时会发现他们的平台并未以最能满足他们需求的方式进行配置。而且由于运营团队并不总是了解软件和功能要求,有时他们会过度配置或配置不足的资源。接下来发生的事情并不神秘:运营团队负责对应用程序性能和可靠性产生负面影响的工程决策。更糟糕的是,糟糕的结果影响了组织的底线。

DevOps 的一个关键概念涉及将这些团队聚集在一起。随着开发和运营团队开始更频繁地协作,很明显自动化将加速部署并降低运营风险。这些团队紧密合作,构建了一些强大的 DevOps 工具。这些工具使用代码自动化了重复的、手动的和容易出错的过程。

最终,这些开发和运营团队开始组建他们自己的“DevOps”团队,将来自开发和运营背景的工程师结合在一起。在这些新团队中,运维工程师获得了开发经验,而开发人员则了解了应用程序运行的幕后方式。随着这一新专业的不断发展,正在设计和构建下一代 DevOps 工具,这将继续改变行业。加强协作对于提高效率和业务成果仍然是必要的,但采用 DevOps 的更多优势正在显现。

由云原生架构实现并通过持续交付管道管理的声明式环境减少了对协作的依赖,并将重点转移到应用程序编程接口 (API) 调用和自动化上。

## CI/CD 工作流的演变

有许多用于开发迭代软件的模型,以及无数的持续集成/持续交付 (CI/CD) 实践。虽然 CI/CD 流程在现场并不新鲜,但它们在开始时更加复杂。现在,随着越来越多的组织迁移到微服务和基于容器的架构,持续交付已成为提高效率的下一个前沿。一套全新的工具

最佳实践正在出现,允许使用红/黑部署和自动金丝雀分析 (ACA) 等策略实现越来越自动化和精确的部署。[第 3 章](#) 有更多细节。

在不可变基础设施的想法流行之前,服务器通常是高度专业化的并且难以替代。每个服务器都有一个特定的目的,并且已经手动调整以实现该目的。[厨师之](#)类的工具和[木偶](#)普及了编写可用于构建和调整这些服务器的可重现代码的概念。服务器仍在频繁更改,但现在代码已提交到[版本控制中](#)。对服务器的更改变得更容易跟踪和重新创建。这些工具也开始简化与 CI/CD 工作流的集成。他们启用了一种标准方式来引入新代码并在所有服务器上重新启动应用程序。当然,最新的应用程序总是有可能崩溃,从而导致难以快速恢复的情况。

考虑到这一点,该行业开始转向避免对现有服务器进行更改的模式:不可变的基础架构。

虚拟机与云基础架构相结合,大大简化了为每个应用程序更新创建新服务器的过程。在此工作流中,CI/CD 管道将创建包含应用程序、依赖项和基本操作系统 (OS) 的机器映像。然后可以使用这些机器映像来创建相同的、不可变的服务器来运行应用程序。在部署到生产环境之前,它们还可以在质量保证 (QA) 环境中进行测试。

在图像投入生产之前对其进行测试的能力极大地提高了 QA 团队的可靠性。

不幸的是,创建新机器映像然后使用它们运行一套全新的服务器的过程也相当缓慢。

正是在这个时候,Docker 开始流行起来。基于 Linux 内核特性、cgroup 和命名空间,Docker 是一个开源项目,可在隔离容器内自动开发、部署和运行应用程序。Docker 提供了很多相同的东西

具有机器图像的优势,但它使用更轻量级的图像格式来实现。Docker 镜像不包括整个基本操作系统,而是简单地包括应用程序及其依赖项。这个过程仍然提供了前面描述的可靠性优势,但在速度上有了一些实质性的改进。Docker 镜像的构建速度更快,引入速度更快,启动速度更快。不是为每个新部署创建新服务器,而是创建可以在相同服务器上运行的新 Docker 容器。

借助 Docker 提供的轻量级方法,CI/CD 工作流真正开始发挥作用。例如,对 Git 存储库的每个新提交都可以构建一个相应的 Docker 映像。每个 Git 提交都可以触发一个多步骤、可自定义的构建过程,其中包括对容器映像的漏洞扫描。缓存的图像随后可用于后续构建,从而加快未来迭代中的构建过程。这些工作流的最新改进之一是容器编排工具,如 Kubernetes。这些工具有

使用容器极大地简化了应用程序更新的部署。此外,它们对资源利用产生了变革性影响。之前您可能在服务器上运行单个应用程序,而通过容器编排,具有截然不同工作负载的多个容器可以在同一台服务器上运行。借助 Kubernetes,CI/CD 正在经历另一次演变,这对通过 DevOps 获得的业务效率具有巨大影响。

## 现代 DevOps 实践

Docker 是第一个获得广泛普及的容器技术,尽管存在替代品并且由 Open Container Initiative (OCI) 标准化。容器允许开发人员将应用程序与其运行所需的所有依赖项捆绑在一起,并将其打包并以

单包。以前,每台服务器都需要拥有所有操作系统级别的依赖项才能运行 Ruby 或 Java 应用程序。容器改变了这一点。它是一个薄包装器 单个包 包含运行应用程序所需的一切。让我们探讨一下现代 DevOps 实践如何体现容器的核心价值。

## 容器带来便携性

Docker 既是一个守护进程 一个在后台运行的进程 也是一个客户端命令。它就像一个虚拟机,但在重要方面有所不同。首先,重复更少。每运行一个额外的虚拟机 (VM),就会复制中央处理器 (CPU) 和内存的虚拟化,并很快耗尽本地资源。 Docker 非常擅长设置本地开发环境,因为它可以轻松添加正在运行的进程,而无需复制虚拟化资源。其次,它更加模块化。 Docker 可以轻松运行同一程序的多个版本或实例,而不会出现配置问题和端口冲突。

因此,您可以将每个单独的应用程序和支持服务链接到一个单元中,并在没有 VM 开销的情况下水平扩展单个服务,而不是单个 VM 或多个 VM。它通过一个单一的描述性 Dockerfile 语法来完成这一切,改善了开发体验,加快了软件交付并提高了性能。而且由于 Docker 基于开源技术,任何人都可以为其开发做出贡献,以构建尚不可用的功能。

使用 Docker,开发人员可以专注于编写代码,而不必担心他们的代码将在哪个系统上运行。应用程序变得真正可移植。您可以自信地在任何其他运行 Docker 的机器上重复运行您的应用程序。对于运维人员来说,Docker 是轻量级的,可以轻松运行和管理应用程序

在隔离的容器中并排具有不同的要求。这种灵活性可以提高每台服务器的资源利用率，并且由于较低的开销可以减少所需的系统数量，从而降低成本。

## 容器进一步模糊了运营和开发之间的界限

容器代表了开发和运营团队之间传统关系的重大转变。构建容器的规范编写起来非常简单，这越来越多地导致开发团队编写这些规范。因此，开发和运营团队更加紧密地合作以部署这些容器。

容器的流行导致 CI/CD 管道的显着改进。在许多情况下，可以使用一些简单的 YAML 文件配置这些管道。此管道配置通常也与应用程序代码和容器规范位于同一存储库中。与传统方法相比，这是一个巨大的变化，在传统方法中，用于构建和部署应用程序的代码存储在单独的存储库中，并完全由运营团队管理。

随着向与应用程序代码共存的简化构建和部署配置的转变，开发人员越来越多地参与以前完全由运营团队管理的流程。

## 容器的初始挑战

尽管容器现在已被大多数组织广泛采用，但从历史上看，三个基本挑战阻碍了组织进行转换。首先，需要转变思维，将当前的开发解决方案转化为容器化开发解决方案。为了

例如,如果您将容器视为虚拟机,您可能希望在其中塞入很多东西,例如服务、监控软件和您的应用程序。这样做可能会导致通常称为“地狱矩阵”的情况。不要把很多东西放到一个容器镜像中;相反,使用许多容器来实现完整的堆栈。换句话说,你可以让你的支持服务容器与你的应用程序容器分开,它们都可以在不同的操作系统和版本上运行,同时链接在一起。

其次,当 Docker 首次普及该技术时,容器的工作和行为方式在很大程度上是未定义的。许多组织想知道容器化是否真的能带来回报,有些组织仍然持怀疑态度。

虽然工程团队可能在实施基于 VM 的方法方面拥有丰富的经验,但它可能对容器本身的工作和行为方式没有概念性的理解。容器技术的一个关键原则是映像永远不会改变,每次运行映像时都为您提供一个不可变的起点,并且无论您在哪里运行,每次运行时它都会执行相同的操作。要进行更改,您需要创建一个新映像并将当前映像替换为较新的版本。这可能是一个难以接受的概念,直到你看到它在行动。

然而,随着采用的普及和组织开始意识到容器的好处 或者看到他们的竞争对手意识到这些好处,这些挑战在很大程度上已被克服。

## 使用容器的 DevOps

Docker 在隔离的容器中运行进程 在本地或远程主机上运行的进程。当你执行命令 docker run 时,运行的容器进程是隔离的 它有自己的文件系统,它自己的

网络和它自己的独立进程树与主机分开。

本质上它是这样工作的 容器镜像是文件系统层的集合,相当于一个固定的起点。当您运行图像时,它会创建一个容器。这种基于容器的部署功能从开发机器到暂存到 QA 再到生产始终保持一致。当您将应用程序放在容器中时,您可以确定您在本地测试的代码与投入生产的构建工件完全相同。应用程序运行时环境没有变化。

您曾经拥有专门的服务器,担心它们会崩溃并不得不更换它们。现在服务器很容易更换并且可以按比例放大或缩小 你的服务器需要做的就是运行容器。哪个服务器正在运行您的容器,或者该服务器是在本地、在公共云中还是在两者的混合体中,都不再重要。您不需要为每个正在运行的应用程序使用应用程序服务器、Web 服务器或不同的专用服务器。如果你失去了一台服务器,另一台服务器可以运行同一个容器。您可以使用相同的工具和相同的服务器部署任意数量的应用程序。

分区化、一致性和标准化工作流已经改变了部署。

容器化为每台服务器上的应用程序部署提供了重大改进。无需担心在服务器上安装应用程序依赖项,它们直接包含在容器映像中。这项技术为 Mesos 和 Kubernetes 等变革性编排工具奠定了基础,这些工具可以简化容器的大规模部署。

容器发展了 DevOps 和行业  
开发人员始终与运营相关,无论他们是否愿意

是或不是。如果他们的应用程序没有正常运行,他们就会被请来解决问题。谷歌是最早引入站点可靠性工程概念的组织之一,其中有才华的开发人员也具有运营领域的技能。这本书,网站可靠性工程:谷歌如何运行生产系统(2016),描述了构建、部署、监控和维护世界上一些最大的软件系统的最佳实践,使用 50% 的开发工作和 50% 的运营工作。随着越来越多的组织采用 DevOps 实践以迁移到微服务和基于容器的架构,这一概念在过去两到三年中得到了应用。

最初是两个不同的工作职能交叉,现在已经成为自己的工作职能。运营团队正在使用代码库;开发人员正在努力部署应用程序,并且正在深入到操作系统中。从操作的角度来看,开发人员可以回顾并阅读 CI 文件并了解部署过程。您甚至可以查看 Dockerfiles 并查看应用程序所需的所有依赖项。从操作的角度来看,理解代码库更简单。

那么这位 DevOps 工程师到底是谁呢?看看 DevOps 的专业化是如何演变的很有趣。一些 DevOps 团队成员具有操作背景,而其他人则具有强大的软件开发背景。连接这些不同背景的是对系统自动化的渴望和欣赏。

运营工程师获得开发经验,而开发人员则接触到应用程序运行的幕后方式。随着这种新的专业化不断发展,下一代 DevOps 工具不断被设计和构建,以适应容器化基础设施中不断变化的角色和架构。

# 使用 Kubernetes 运行容器

2015 年,能够以编程方式将工作负载“调度”到与应用程序无关的基础架构中是前进的方向。今天,最佳实践是迁移到某种形式的容器编排。

许多组织仍然使用 Docker 来打包他们的应用程序,理由是它的一致性。

Docker 是朝着正确方向迈出的重要一步,但它是达到目的的一种手段。事实上,在编排工具出现之前,容器的部署方式并没有发生变革。正如 Docker 之前存在许多容器技术一样,许多容器编排技术先于 Kubernetes。最著名的工具之一是 Apache Mesos,它是由 Twitter 构建的工具。Mesos 在容器编排方面做得很强大,但它过去 现在仍然可能 难以设置和使用。Mesos 仍然被具有规模和规模的企业所使用,它是适合正确用例和规模的优秀工具。

如今,组织越来越多地选择使用 Kubernetes 而不是其他编排工具。越来越多的组织认识到容器提供了比他们一直使用的更传统工具更好的解决方案,并且 Kubernetes 是可用的最佳容器部署和管理解决方案。让我们进一步研究这些想法。

## Kubernetes 简介

Kubernetes 是一个功能强大的下一代开源平台,用于跨主机集群自动部署、扩展和管理应用程序容器。它可以运行任何工作负载。Kubernetes 提供卓越的开发人员用户体验 (UX),并且创新速度惊人。从一开始,Kubernetes 的基础设施就承诺使组织能够大规模快速部署应用程序

并在仅使用所需资源的同时轻松推出新功能。

借助 Kubernetes,组织可以在自己的公共云或本地环境中运行自己的 Heroku。

Kubernetes 于 2014 年首次由 Google 发布,从一开始就看起来很有前景。每个人都想要零停机部署、全自动部署管道、自动扩展、监控、警报和日志记录。然而,在那个时候,建立一个 Kubernetes 集群是很困难的。

当时,Kubernetes 本质上是一个自己动手做的项目,有很多手动步骤。许多复杂的决定曾经 现在 都涉及到 你必须生成证书,启动具有正确角色和权限的虚拟机,将包放到这些虚拟机上,然后使用云提供商设置、IP 地址、DNS 条目等构建配置文件。事实上,起初并非一切都按预期工作,业内许多人对使用 Kubernetes 犹豫不决也就不足为奇了。

Kubernetes 1.2 于 2016 年 4 月发布,包含更多面向通用用途的功能。它被准确地吹捧为下一件大事。从一开始,这个开创性的开源项目就是一个优雅的、结构化的、真实的大规模容器化解决方案,它解决了其他技术无法解决的关键挑战。Kubernetes 包括智能架构决策,有助于在容器化中构建应用程序。很多事情都在你的掌控之中。

例如,您可以决定如何设置、维护和监控不同的 Kubernetes 集群,以及如何将这些集群集成到您基于云的基础架构的其余部分。

Kubernetes 得到了主要行业参与者的支持,包括亚马逊、谷歌、微软和红帽。拥有超过 14,000 名个人贡献者和不断增长的势头,这个项目将继续存在。

在过去的几年里,想一想开发团队希望了解运营部署的频率。开发人员和运营团队一直对部署感到紧张,因为维护窗口有扩大的趋势,从而导致停机。反过来,运营团队传统上会守卫自己的领地,这样就不会有人干扰他们完成工作的能力。

然后容器化和 Kubernetes 出现了,软件工程师想了解它并使用它。这是革命性的。这不是传统的操作范例。它是软件驱动的,非常适合工具和自动化。Kubernetes 使工程师能够专注于任务驱动的编码,而不是提供桌面支持。同时,它将工程师带入运维世界,为开发和运维团队提供了一个了解彼此世界的清晰窗口。

## Kubernetes 是游戏规则的改变者

Kubernetes 正在改变游戏规则,不仅在工作的完成方式上,而且在谁被吸引到这个领域。Kubernetes 已经发展成为容器编排的标准。对行业的影响是巨大的。

过去,服务器是为运行特定应用程序而定制的,如果服务器出现故障,您必须想办法重建它。Kubernetes 简化了部署过程并提高了资源利用率。正如我们之前所说,使用 Kubernetes,您的服务器可能是愚蠢的 它们不关心它们在运行什么。您可以堆叠资源,而不是在特定服务器上运行特定应用程序。例如,Web 服务器和后端处理服务器可能都在 Docker 容器中运行。假设您有三台服务器,每台服务器上可以运行五个应用程序。如果一台服务器出现故障,你就会有冗余,因为一切都在过滤。

## Kubernetes 的一些好处

·可独立部署的服务 :您可以开发应用程序作为一套可独立部署的模块化服务。可以使用 Kubernetes 为几乎任何软件堆栈构建基础架构代码 ,因此组织可以创建可重复的流程 ,这些流程可以跨许多不同的应用程序进行扩展。

·部署频率 :在 DevOps 世界中 ,整个团队拥有相同的业务目标 ,并负责构建和运行符合预期的应用程序。更频繁地部署更短的工作单元可以最大程度地减少诊断问题时必须筛选的代码量。Kubernetes 部署的速度和简单性使团队能够部署频繁的应用程序更新。

·弹性： DevOps 团队的核心目标是通过自动化实现更高的系统可用性。考虑到这一点 ,Kubernetes 旨在自动从故障中恢复。例如 ,如果一个应用挂掉了 ,Kubernetes 会自动重启它。

·可用性： Kubernetes 有一个文档齐全的 API ,简单、简单的配置 ,提供非凡的开发人员用户体验。DevOps 实践和 Kubernetes 一起还允许企业将应用程序和功能快速交付到用户手中 ,这转化为更具竞争力的产品和更多的收入机会。

## Kubernetes 简化了编排 你的申请

除了改进传统的 DevOps 流程外 ,速度、效率和弹性通常被认为是 DevOps 的优势

DevOps、Kubernetes 解决了容器带来的新问题和

基于微服务的应用程序架构。换句话说,Kubernetes 强化了 DevOps 目标,同时还支持微服务架构带来的新工作流。

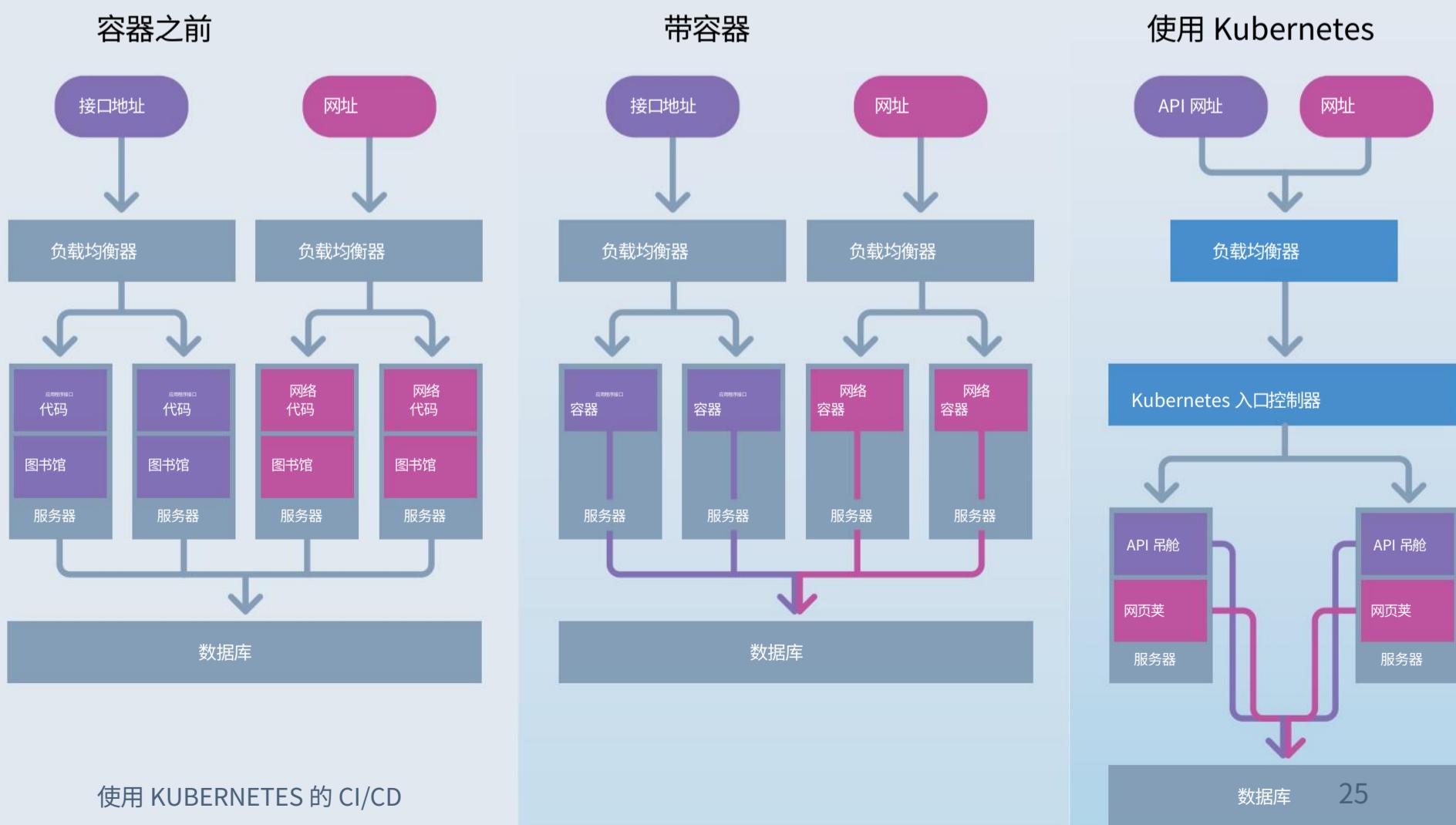
## 强大的积木

Kubernetes 使用 Pod 作为部署的基本单元。Pod 代表一组使用相同存储和网络的一个或多个容器。尽管 Pod 通常仅用于运行单个容器,但它们已被用于一些创造性的方式,包括作为构建服务网格的一种方式。

单个 pod 中的多个容器的常见用法遵循边车模式。使用这种模式,容器将在您的核心应用程序旁边运行以提供一些额外的价值。这通常用于

图 1.1:使用 Kubernetes,pod 分布在具有内置负载平衡和路由的服务器上。以这种方式分布应用程序工作负载可以显着提高资源利用率。

## 应用基础设施的演变



代理请求,甚至处理身份验证。

有了这些强大的构建块,将容器化之前可能已经在虚拟机中运行的服务映射到在同一个 pod 中运行的多个容器变得非常简单。

## 简化的服务发现

在一个整体式应用程序中,不同的服务各有各的用途,但自包含有助于通信。

在微服务架构中,微服务需要相互通信 您的用户服务需要与您的邮政服务和地址服务等通信。弄清楚这些服务如何简单而一致地进行通信并非易事。

使用 Kubernetes,DevOps 工程师定义服务 例如,用户服务。在同一个 Kubernetes 命名空间中运行的任何东西都可以向该服务发送请求,而 Kubernetes 会弄清楚如何为您路由请求,从而使微服务更易于管理。

## 集中、易读的配置

Kubernetes 在声明式模型上运行:您描述一个期望的状态,Kubernetes 将尝试实现该状态。Kubernetes 具有易于阅读的 YAML 文件,用于描述您想要达到的状态。使用 Kubernetes YAML 配置,您可以定义从应用程序负载均衡器到一组 Pod 的任何内容来运行您的应用程序。部署配置可能具有应用程序的 Docker 容器之一的三个副本和两个不同的环境变量。

这个易于阅读的配置可能存储在 Git 存储库中,因此您可以随时看到配置更改。在 Kubernetes 之前,很难知道跨服务器的互连系统实际发生了什么。

除了配置运行在您的应用程序容器中

集群或可用于访问它们的端点,Kubernetes 可以帮助进行配置管理。 Kubernetes 有一个名为 ConfigMap 的概念,您可以在其中为您的应用程序定义环境变量和配置文件。同样,称为机密的对象包含敏感信息,有助于定义应用程序的运行方式。 Secret 的工作方式与 ConfigMap 非常相似,但对最终用户来说更加晦涩难懂。

[第 2 章](#)详细探讨了所有这些。

## 实时真相源

过去,手动和脚本化发布压力极大。你只有一次机会做对了。借助 Kubernetes 的内置部署功能,任何人都可以使用 Kubernetes 的无限部署历史记录来部署和检查交付状态:kubectl rollout history。

Kubernetes API 提供了有关部署状态的实时事实来源。任何有权访问集群的开发人员都可以快速了解交付情况或查看所有发出的命令。

出于安全和历史目的,此永久系统审核日志保存在一个地方。您可以轻松了解以前的部署,查看部署之间的差异或回滚到任何列出的版本。

## 简单的健康检查能力

这在应用程序的生命周期中是一件大事,尤其是在部署阶段。过去,应用程序在崩溃时通常不会自动重启;相反,有人在半夜被传呼,不得不重新启动它们。另一方面,Kubernetes 具有自动健康检查功能,如果应用程序因任何原因(包括内存不足或只是锁定)无法响应,Kubernetes 会自动重新启动它。

澄清一下,Kubernetes 会检查您的应用程序是否正在运行,但它不知道如何检查它是否正确运行。然而,

Kubernetes 使为您的应用程序设置健康检查变得简单。  
您可以通过两种方式检查应用程序的运行状况：

1. 使用 liveness probe 检查应用程序是否来自  
健康状态到不健康状态。如果它进行了转换，它将尝试为您重新启动您的应用程序。

2. 使用就绪探针检查应用程序是否准备就绪  
接受流量。在新容器健康之前，它不会摆脱以前工作的容器。基本上，就绪探测器是防  
止破损容器出现的最后一道防线。

这两个探针都是有用的工具，Kubernetes 使它们易于配置。

此外，如果您有适当配置的就绪探测器，则回滚很少见。如果所有健康检查都失败了，一个  
单行命令将为您回滚该部署并让您回到稳定状态。它不常用，但如果需要它，它就在那里。

## 滚动更新和本机回滚

进一步构建实时来源和健康检查的想法

功能，Kubernetes 的另一个关键特性是使用上述本机回滚进行滚动更新。部署可以而  
且应该是频繁的，而不用担心到达不归路。在 Kubernetes 之前，如果你想部署一些东西，  
一个常见的部署模式涉及服务器拉入最新的应用程序代码并重新启动你的应用程序。该过  
程存在风险，因为某些功能不向后兼容。如果在部署过程中出现问题，软件将变得不可用。  
例如，如果服务器发现新代码，它会提取这些更新并尝试使用新代码重新启动应用程序。如  
果该管道中出现故障，则该应用程序可能已死。回滚过程一点也不简单。

在 Kubernetes 之前,这些工作流是有问题的。Kubernetes 通过部署回滚功能解决了这个问题,该功能消除了大维护窗口和对停机时间的担忧。从 Kubernetes 1.2 开始,部署对象是一个声明性清单,其中包含正在交付的所有内容,包括正在部署的副本数量和软件映像的版本。这些项目被抽象并包含在部署声明中。这种基于清单的部署刺激了新的 CD 工作流,并且是 Kubernetes 不断发展的最佳实践。

在 Kubernetes 关闭现有应用程序容器之前,它将开始启动新的应用程序容器。只有当新的启动并正确运行时,它才会摆脱贫旧的、稳定的版本。假设 Kubernetes 没有捕捉到失败的部署 应用程序正在运行,但它处于 Kubernetes 无法检测到的某种错误状态。在这种情况下,DevOps 工程师可以使用简单的 Kubernetes 命令来撤消该部署。此外,您可以将其配置为根据需要存储少至两次更改或任意多的修订,并且您可以返回到上次部署或之前的许多部署,所有这些都可以通过一个简单的自动化 Kubernetes 命令完成。整个概念改变了游戏规则。其他编排框架无法像 Kubernetes 那样以无缝和合乎逻辑的方式处理这个过程。

## 简化监控

虽然从表面上看,监控 Kubernetes 似乎相当复杂,但这个领域已经有了很多发展。

尽管 Kubernetes 和容器为您的基础架构增加了一定程度的复杂性,但它们也确保您的所有应用程序都在一致的 pod 和部署中运行。这种一致性使监控工具在许多方面变得更简单。

Prometheus 是一个开源监控工具的示例,它具有

在云原生生态系统中变得非常流行。该工具提供高级监控和警报功能,具有出色的 Kubernetes 集成。

监控 Kubernetes 时,需要关注几个关键组件: Kubernetes 节点 (服务器) ; Kubernetes 系统部署,例如 DNS 或网络;当然,还有您的应用程序本身。有许多监控工具可以简化对这些组件的监控。

Kubernetes 和 CI/CD 相互补充在 Kubernetes 之上设置 CI/CD 管道将加快您的发布生命周期 使您能够一天发布多次 并使灵活的团队能够快速迭代。使用 Kubernetes,构建变得更快。您的构建过程无需启动全新的服务器,而是快速、轻量级和直接的。

当您不必担心构建和部署单体来更新所有内容时,开发速度会加快。通过将整体拆分为微服务,您可以改为更新各个部分 这个服务或那个。一个好的 CI/CD 工作流的一部分还应该包括一个强大的测试套件。虽然并非 Kubernetes 独有,但容器化方法可以使测试运行起来更加直接。如果您的应用程序测试依赖于其他服务,您可以针对这些容器运行测试,从而简化测试过程。更新 Kubernetes 部署通常只需要一行命令。

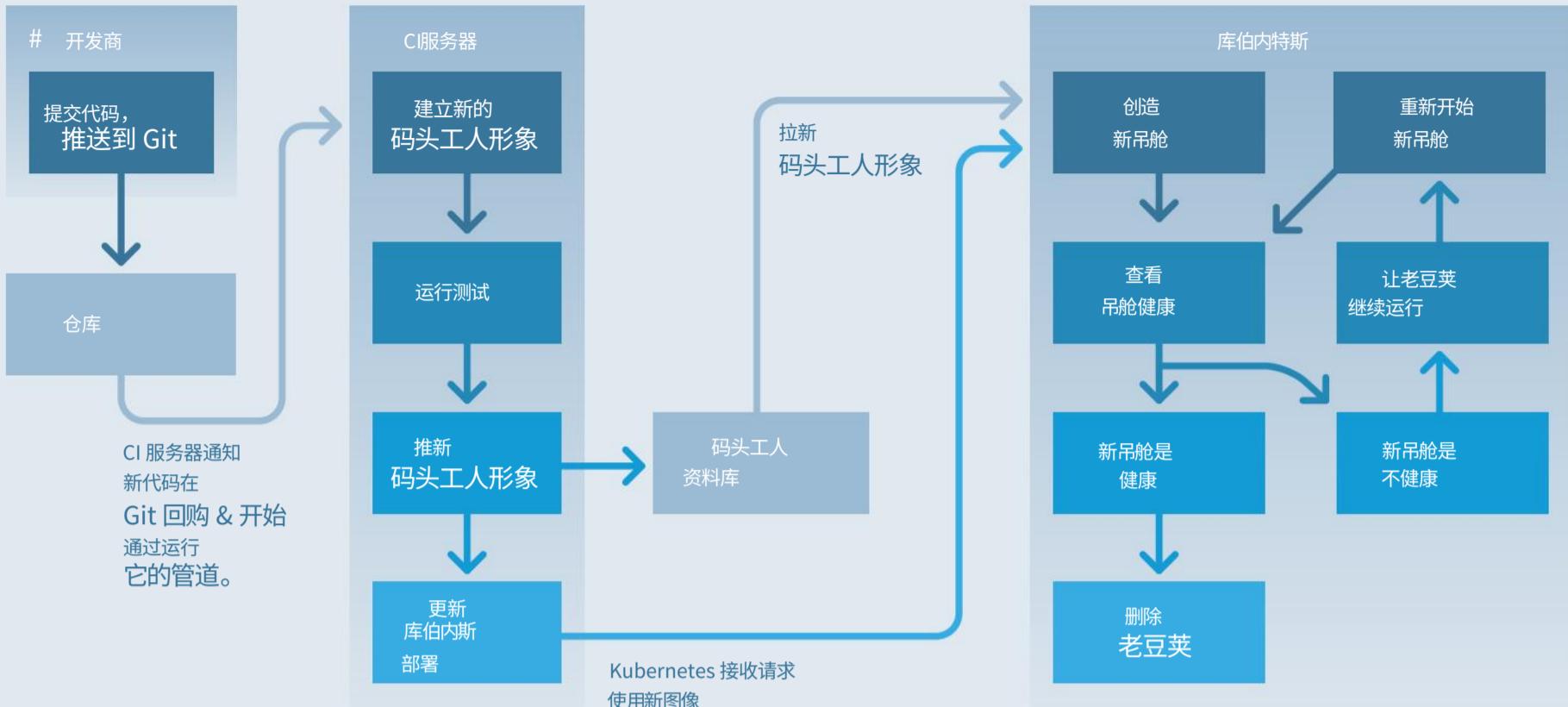
在 CI/CD 工作流中,理想情况下您运行许多测试。如果这些测试失败,您的镜像将永远无法构建,您也永远无法部署该容器。

但是,如果测试未能发现问题,Kubernetes 会提供更好的保护,因为 Kubernetes 简化了零停机时间部署。

长期以来,部署意味着停机。运营团队过去常常

# 使用 Kubernetes 的 CI/CD 管道工作流

开发模式



资料来源:ReactiveOps

© 2018 THE NEW STACK

图 1.2:在 Kubernetes 关闭现有 pod 之前,它会开始启动新的 pod。只有当新的启动并正确运行时,它才会摆脱旧的、稳定的版本。这种滚动更新和本机回滚功能是 DevOps 的游戏规则改变者。

手动或通过脚本处理部署工作,这是一个可能需要数小时甚至整夜的实时过程。因此,对 CI/CD 的担忧之一是部署会中断并且站点会崩溃。

Kubernetes 的零停机部署能力消除了对维护窗口的担忧,使计划延迟和停机成为过去 并在此过程中节省资金。它还可以让每个人都了解情况,同时满足开发、运营和业务团队的需求。革命性的 Kubernetes 部署对象具有自动执行此操作的内置功能。

特别是,上述测试和健康检查可以防止错误代码进入生产环境。作为滚动更新的一部分,Kubernetes 会启动单独的新 pod 来运行您的应用程序,同时旧的 pod 仍在运行。当新的 pod 健康时,Kubernetes 摆脱

旧的。这是一个聪明、简单的概念,对于每个应用程序和 CI/CD 工作流,您不必担心的事情就更少了。

## 补充工具

作为 Kubernetes 令人难以置信的势头的一部分,出现了许多 DevOps 工具,它们特别有助于开发 CI/

使用 Kubernetes 的 CD 工作流。请记住,随着 Kubernetes 上云原生部署的出现,CI/CD 工具和实践仍在不断发展。目前还没有任何一种工具能够提供完美的解决方案来管理从构建到部署和持续交付的云原生应用程序。尽管这里有太多无法提及的内容,但值得重点介绍一些专为云原生应用程序构建的 DevOps 工具:

- **Draf:** Microsoft 的这个工具以开发人员工作流为目标。通过一些简单的命令,Draf 可以将应用程序容器化并部署到 Kubernetes。这里应用程序的自动化容器化可能非常强大。 Draf 使用流行框架和语言的最佳实践来构建适用于大多数情况的镜像和 Kubernetes 配置。
- **Helm:**被称为 Kubernetes 包管理器,该框架简化了将应用程序部署到 Kubernetes 的过程。许多流行项目的部署配置都可以在维护良好的“图表”中找到。这意味着 helm install prometheus 是让 Prometheus 这样的项目在您的集群中运行所需要的一切。在部署您自己的自定义应用程序时,Helm 也可以提供同样的便利。
- **Skafold:**与 Draf 类似,这是来自 Google 的新工具,可实现激动人心的新开发工作流。除了支持更多标准的 CI/CD 工作流之外,它还提供了构建和部署代码的选项

每次代码在本地发生变化时,都迁移到 Kubernetes 开发环境。该工具高度可配置,甚至支持使用 Helm 进行部署。

- Spinnaker:这个开源持续交付平台是由 Netflix 开发,用于通过其云网络处理大规模 CD 操作。它是一个云原生管道管理工具,支持与所有主要云提供商的集成:亚马逊网络服务 (AWS)、Azure、谷歌云平台和 OpenStack。它本机支持 Kubernetes 部署,但其范围远远超出 Kubernetes。

## DevOps 的扩展

云原生应用程序的持续部署改变了团队协作的方式。开发和部署每个阶段的透明度和可观察性越来越成为常态。因此,GitOps 和 SecOps 都由云原生架构支持,它们通过为基础架构的更改以及安全策略和规则的更改提供单一的真实来源来构建当前的 DevOps 实践,这可能并不奇怪。以下部分重点介绍了这些进化发展。

## GitOps

Git 正在成为分布式版本控制的标准,其中 Git 存储库包含整个系统代码、配置、监控规则、仪表板和完整的审计跟踪。[GitOps](#)是 DevOps 的迭代,其中 Git 是整个系统的单一事实来源,支持在云原生系统上快速开发应用程序,尤其是使用 Kubernetes。“GitOps”是 Weaveworks 开发的一个术语,用于描述 Kubernetes 时代的 DevOps 最佳实践,它非常强调声明式基础架构。

GitOps 的基本定理是,如果你能描述它,你就能自动化它。如果你可以自动化它,你就可以控制和加速它。

目标是描述所有内容 策略、代码、配置和监控 然后对所有内容进行版本控制。

使用 GitOps,您的代码应该代表您的基础设施的状态。

GitOps 借用了 DevOps 逻辑:

- 所有代码都必须受版本控制。
- 配置是代码。
- 配置也必须是版本控制的。

GitOps 背后的理念是透明度。声明式环境捕获状态,使您能够轻松比较观察到的状态和所需状态。事实上,您可以随时观察系统。简而言之,每个服务都有两个真实来源:系统的期望状态和观察到的系统状态。

在 GitOps 的最真实意义上,Git 维护着一个存储库,该存储库描述了您的基础设施和所有 Kubernetes 配置,而您的本地代码副本为您提供了一个完整的版本控制存储库。当您将新代码或配置推送到 Git 时,另一端会监听此新推送并为您进行更改。所有的变化都以同样的方式发生。您所有的基础设施和配置都位于一个集中的存储库中,每一个更改都是由该存储库的提交和推送驱动的。

工作原理:代码被提交并推送到 GitHub,然后你有一个 CI/CD 工作流在另一端监听,该 CI/CD 工作流进行这些更改并在配置中提交这些更改。

关键区别 工程师不是直接与 Kubernetes 或系统配置交互 比如,使用 Kubernetes CLI 他们正在做

一切都通过 Git 编写配置,推送配置,然后通过 CI/CD 工作流应用这些更改。

GitOps 的三个关键目标包括:

1.管道:构建完全自动化的 CI/CD 管道,使 Git 成为所需系统状态的真实来源。

2.可观察性:实施 24/7 全天候监控、日志记录和安全性

观察和测量每个服务拉取请求,以获得当前系统状态的整体视图。

3.控制:版本控制一切,并创建一个包含单一事实来源的存储库,用于恢复目的。

借助 GitOps,Weaveworks 看到人们从每周使用 CI 系统进行几次部署到每天进行 30 到 50 次部署。此外,DevOps 团队修复错误的速度提高了一倍。管理 Git 与 Kubernetes 中的代码状态可以更好地跟踪和恢复。它还允许在 Kubernetes 架构上进行持续的实验,例如 A/B 测试和对客户想法的响应。

## 安全运营

安全团队传统上将安全测试结果和漏洞扫描交给运营团队进行审查和实施,通常是在部署应用程序时。只要应用程序按预期运行和执行,安全参与该过程就会获得绿灯。但是,信息交换和批准周期可能会导致延迟并减慢原本敏捷的 DevOps 工作流。出现这种情况并不奇怪,因为两支球队正在处理两组截然不同的目标。运营试图让系统以尽可能直接和有弹性的方式运行。另一方面,安全性寻求控制环境运行的东西越少越好。

实际上,出于多种原因,运维和安全之间的后期交接是有问题的,其中最重要的是两个团队的见解和专业知识是孤立的,而不是共享的。因此,潜在的威胁可能会发展成为障碍,与安全相关的问题往往会被恶化,未被发现的时间比保证的时间更长。

当组织没有持续通信和传输关键安全数据的机制时,这些组织不可避免地要努力降低安全风险、确定安全威胁和漏洞的优先级并进行补救,并最终保护其应用程序环境。但是,组织没有必要为了保持正常运行时间和性能而牺牲安全性。这就是 SecOps 发挥作用的地方。

SecOps 连接安全和运营团队的工作,就像 DevOps 连接软件开发人员和运营团队的工作一样。正如 DevOps 方法允许产品开发人员战略性地部署、管理、监控和保护他们自己的应用程序一样,SecOps 方法为工程师提供了一个了解操作和安全问题的窗口。这是从个人战术家(系统管理员和数据库管理员)到组织内更具战略意义的角色的转变。这些团队共享优先级、流程、工具,最重要的是,问责制,为组织提供漏洞和补救措施的集中视图,同时自动化和加速纠正措施。

在 GitOps 方法中,您所有的配置和基础设施都存储在中央 Git 存储库中。DevOps 工程师编写开发和部署策略,而安全工程师编写安全防火墙规则和网络策略。所有这些规则最终都在同一个存储库中。这些团队之间的协作 DevOps 和 SecOps,或者

SecDevOps 提高赌注,提高效率和透明度,了解已经发生的事情和未来应该发生的事情。

在用统一的、主动的、基于 CI/CD 的云和本地系统安全解决方案取代断开连接的、被动的安全工作时,SecOps 获得了一支来自不同背景的更具凝聚力的团队,朝着一个共同的目标努力:频繁、快速、零停机时间,安全部署。这一目标使运营和安全部门能够分析安全事件和数据,着眼于缩短响应时间、优化安全控制以及在开发和部署的每个阶段检查和纠正漏洞。模糊运营和安全团队之间的界限可以更好地了解任何必要的开发或部署变更,以及这些变更的潜在影响。

## 结论

DevOps 和 Kubernetes 的故事是一个持续的、快节奏的发展过程。例如,在 DevOps 的当前状态下,可能只有几年历史的技术可能会开始让人觉得古老。这个行业正在瞬息万变。

Kubernetes 仍然非常新颖和令人兴奋,市场对它的需求令人难以置信。组织正在利用 DevOps 迁移到云、自动化基础架构并将软件即服务 (SaaS) 和 Web 应用程序提升到一个新的水平。考虑一下通过更高的可用性、自动缩放和更丰富的功能集可以完成什么。Kubernetes 极大地改进了 CI/CD 工作流程,使开发人员能够完成他们以前无法做到的惊人事情。反过来,GitOps 提供了使 Kubernetes 变得简单的集中配置功能。通过透明和集中的配置,更改不会随意单独应用,而是通过相同的管道进行。今天,产品

团队正在集中开发、运营和安全,努力实现相同的业务目标:更轻松、更快的部署、更少的停机时间、更少的中断和更快的恢复时间。

越来越多的组织转向 Kubernetes,因为他们确定它是完成这项工作的正确工具。Kubernetes 开箱即用。Kubernetes 让做错事变得很难,而做对事却很容易。[据 RedMonk 称](#),超过一半的财富 100 强公司正在使用 Kubernetes。但故事并没有就此结束。

---

中型公司使用 Kubernetes。十人创业公司使用 Kubernetes。Kubernetes 不仅仅适用于企业。对于各种规模的具有前瞻性思维的公司来说,这都是真正的交易。它正在彻底改变软件创新、设计、构建和部署的方式。

DevOps 和 Kubernetes 是未来。它们一起具有良好的商业意义。

# 最好的CI/CD Kubernetes工具 不存在



Kubernetes 的持续集成/持续交付 (CI/CD) 没有单一、最佳的工具集。每个组织都将使用最适合其特定用例的工具。

“[关于在 Kubernetes 上运行] 最困难的事情是将所有部分粘合在一起。您需要更全面地考虑您的系统，并深入了解您正在使用的内容，”  
[Chris Short 说](#)， DevOps 顾问和云原生计算基金会 (CNCF) 大使。

我们与 Short 和 [Ihor Dvoretskyi 交谈](#)，CNCF 的开发者倡导者，关于他们在 DevOps 和 CI/CD 与 Kubernetes 中看到的趋势，Kubernetes 社区在改进 CI/CD 中的作用

CD 以及组织在考虑当今可用工具过多时面临的一些挑战。[在 SoundCloud 上收听»](#)



Chris Short 在各种 IT 学科工作了 20 多年，在私营和公共部门任职期间一直是开源解决方案的积极支持者。读

更多信息请访问[chrissshort.net](http://chrissshort.net)，或关注他的以 DevOps、云原生和开源为重点的时事通讯[DevOps-ish](#)。



Ihor Dvoretskyi 是 Cloud Native Computing Foundation 的开发倡导者。他是 Kubernetes 的产品经理，共同领导产品管理特别兴趣小组，专注于

关于增强 Kubernetes 作为开源产品。此外，他还作为功能负责人参与了 Kubernetes 发布过程。

# 云原生 应用模式

通过 JANAKIRAM MSV

操作。他们的队伍中充斥着创建在复杂的底层基础设施技术上运行的微服务的人。开发人员被容器中的实用程序和打包功能所吸引,这些功能可以提高效率并适应现代应用程序开发实践。

开发人员使用具有容器技术的应用程序架构来开发反映组织目标的服务。Kubernetes 在应用程序架构下运行,用于跨多个集群运行服务,提供编排的抽象来管理遵循 DevOps 实践的微服务。当今在由软件管理的基础设施上运行的云原生现代服务的灵活性为开发人员提供了以前无法实现的功能。开发人员现在拥有通过集成到声明性基础设施中来连接更多端点的资源。动态的、声明性的基础设施现在是开发的基础,并将越来越多地作为现代应用程序架构中事件驱动自动化的资源。

在所有这些中,容器可以被定义为一个单元。每个单元都包含代码,这是一种有效载荷,在更复杂的操作中,它会跨分布式架构和由云服务管理的基础设施进行编排。

现在,越来越多的开发人员正在测试如何优化不同类型基础架构上的可用资源,以利用容器和虚拟化带来的优势。

云原生是一个用于描述基于容器的环境的术语。

云原生技术用于开发使用打包在容器中的服务构建的应用程序,部署为微服务,并通过敏捷的 DevOps 流程和持续交付工作流在弹性基础设施上进行管理。重要的是要注意,在这个等式中,允许提高速度和敏捷性的团队和流程与技术本身一样重要。

“Cloud Native 正在构建团队、文化和技术,以利用自动化和架构来管理复杂性和释放速度。”

- Heptio 的首席技术官兼联合创始人 Joe Beda。

在运营团队手动管理基础设施资源分配给传统应用程序的地方,云原生应用程序部署在抽象底层计算、存储和网络原语的基础设施上。处理这种新型应用程序的开发人员和运营商不会直接与基础架构提供商公开的应用程序编程接口 (API) 进行交互。

相反,编排器自动处理资源分配,

根据 DevOps 团队制定的政策。控制器和调度器是编排引擎的基本组件，负责处理资源分配和应用程序的生命周期。云原生平台，如 Kubernetes，暴露了一个覆盖在现有网络拓扑和云提供商原语上的胖网络。类似地，本机存储层通常被抽象为暴露与容器集成的逻辑卷。运营商可以分配存储配额和网络策略，供开发人员和资源管理员访问。基础架构抽象不仅解决了跨云环境的可移植性需求，还允许开发人员利用新兴模式来构建和部署应用程序。

编排管理器成为部署目标，而不管可能基于物理服务器或虚拟机、私有云或公共云的底层基础设施。

Kubernetes 是运行设计为云原生应用程序的当代工作负载的理想平台。它已成为事实上的云操作系统，就像 Linux 是底层机器的操作系统一样。本章的作者是 Janakiram MSV，他是 Janakiram & Associates 的首席分析师，也是国际信息技术学院的兼职教员。他是 Kubernetes 开源项目之家 Cloud Native Computing Foundation 的大使，也是首批获得认证的 Kubernetes 管理员和获得认证的 Kubernetes 应用程序开发人员之一。在这里，他利用自己丰富的专业知识为组织提供云战略和 Kubernetes 实施方面的咨询服务，以将云原生应用程序的构建块与 Kubernetes 的构造和原语进行映射。本章将指导您为成熟的云原生应用程序的每个组件选择正确的 Kubernetes 对象。

只要开发人员遵循设计和开发的最佳实践

软件作为一组包含云原生应用程序的微服务,DevOps 团队将能够在 Kubernetes 中打包和部署它们。

本章旨在帮助指导 DevOps 团队在 Kubernetes 中部署云原生应用程序。

## 云原生的 10 个关键属性应用

在纵观云原生应用设计的大局之前,让我们回顾一下云原生应用的十大关键属性。

1. 打包为轻量级容器:云原生应用程序是打包为轻量级容器的独立和自治服务的集合。与虚拟机不同,容器可以快速横向扩展和收缩。由于缩放单元转移到容器,因此优化了基础架构利用率。
2. 使用最佳语言和框架开发:云原生应用程序的每项服务都是使用最适合功能的语言和框架开发的。云原生应用程序是多语言的;服务使用各种语言、运行时和框架。例如,开发人员可能会构建一个基于 WebSockets 的实时流服务,在 Node.js 中开发,同时选择 Python 和 Flask 来公开 API。开发微服务的细粒度方法让他们可以为特定工作选择最佳语言和框架。
3. 设计为松耦合微服务:属于同一应用程序的服务通过应用程序运行时相互发现。它们独立于其他服务而存在。弹性基础架构和应用程序架构在正确集成后,可以高效和高性能地进行横向扩展。

松散耦合的服务允许开发人员独立对待每个服务。通过这种解耦,开发人员可以专注于每个服务的核心功能,以提供细粒度的功能。这种方法可以对整个应用程序进行有效的生命周期管理,因为每个服务都是独立维护的并且具有明确的所有权。

#### 4. 以用于交互和协作的 API 为中心:云

本机服务使用基于代表性状态传输 (REST)、Google 的开源远程过程调用 (gRPC) 或 NATS 等协议的轻量级 API。 REST 用作通过超文本传输协议 (HTTP) 公开 API 的最低公分母。

出于性能考虑,gRPC 通常用于服务之间的内部通信。 NATS 具有发布-订阅功能,可以在应用程序内实现异步通信。

#### 5. 将无状态和有状态服务完全分开进行架构:持久和持久的服务遵循不同的模式,以确保更高的可用性和弹性。无状态服务独立于有状态服务而存在。这里与存储如何影响容器使用有关。持久性是一个必须越来越多地在状态、无状态和 (有些人会争辩)微存储环境的背景下考虑的因素。

#### 6. 与服务器和操作系统依赖隔离:

云原生应用程序对任何特定的操作系统或个人机器没有亲和力。它们在更高的抽象级别上运行。唯一的例外是当微服务需要某些功能时,包括固态驱动器 (SSD) 和图形处理单元 (GPU),这些功能可能仅由一部分机器提供。

7. 部署在自助式、弹性的云基础设施上:云原生应用部署在虚拟的、共享的、弹性的基础设施上。它们可能与底层基础设施保持一致以动态增长和收缩调整自己以适应不断变化的负载。

## 8. 通过敏捷的 DevOps 流程进行管理:

云原生应用程序经历一个独立的生命周期,通过敏捷的 DevOps 流程进行管理。多个持续集成/持续交付 (CI/CD) 管道可以协同工作以部署和管理云原生应用程序。

## 9. 自动化能力:云原生应用可以高度

自动化。他们很好地运用了基础架构即代码的概念。

事实上,仅需要一定程度的自动化来管理这些大型和复杂的应用程序。

## 10. 定义的、策略驱动的资源分配:最后是云原生

应用程序与通过一组策略定义的治理模型保持一致。它们遵守中央处理器 (CPU) 和存储配额等政策,以及为服务分配资源的网络政策。例如,在企业场景中,中央 IT 可以定义策略来为每个部门分配资源。每个部门的开发人员和 DevOps 团队都可以完全访问和拥有他们共享的资源。

# 云原生应用概述 设计

云原生应用程序由各种逻辑层组成,根据功能和部署模式进行分组。每一层都运行旨在执行细粒度任务的特定微服务。一些

# 将层映射到云原生工作负载

云原生应用程序模式



资料来源:Janakiram MSV

© 2018 THE NEW STACK

图 2.1:当今的现代应用程序架构与单一的遗留系统相连。

这些微服务是无状态的,而其他微服务是有状态且持久的。

应用程序的某些部分可能作为批处理运行。代码片段可以部署为响应事件和警报的函数。

此处的描述试图识别云原生应用程序的层。虽然它们被组合在一起进行表示,但每一层都是独立的。与按层次结构堆叠的传统三层应用程序不同,云原生应用程序在胖结构中运行,每个服务都公开一个 API。

可扩展层运行公开 API 和用户体验的无状态服务。该层可以根据运行时的使用动态扩展和收缩。在横向扩展操作期间,运行更多服务实例时,底层基础设施也可能会横向扩展以满足 CPU 和内存要求。自动缩放

实施政策以评估执行横向扩展和横向扩展操作的必要性。

持久层具有由多语言持久性支持的有状态服务。它是多语言的,因为可用于持久性的数据库种类繁多。有状态服务依赖于传统的关系数据库、NoSQL 数据库、图形数据库和对象存储。每个服务都会选择一个与存储数据结构一致的理想数据存储。这些有状态服务公开了可扩展层和持久层中的服务都使用的高级 API。

除了无状态和有状态层之外,还有计划作业、批处理作业和并行作业被归类为可并行化层。例如,计划作业可以每天运行一次提取、转换、加载 (ETL) 任务,以从存储在对象存储中的数据中提取元数据并填充 NoSQL 数据库中的集合。对于需要科学计算来执行机器学习训练的服务,计算是并行运行的。这些作业与底层基础设施公开的 GPU 交互。

为了触发由平台中任何服务引发的事件和警报引起的操作,云原生应用程序可以使用一组部署在事件驱动层中的代码片段。与其他服务不同的是,运行在这一层的代码并没有被打包成一个容器。而是直接部署用 Node.js 和 Python 等语言编写的函数。

该层托管事件驱动的无状态功能。

云原生应用程序还可以与遗留层的现有应用程序进行互操作。遗留的、单一的应用程序 例如企业资源规划、客户关系管理、供应链管理、人力资源和内部业务线应用程序 由服务访问。

企业将采用微服务来构建与现有应用程序互操作的 API 层和用户界面 (UI) 前端。在这种情况下,微服务增强和扩展了现有应用程序的功能。例如,他们可能必须与为业务线应用程序提供支持的关系数据库对话,同时交付部署为微服务的弹性前端。

云原生应用程序的每个服务都公开了一个定义良好的 API,供其他服务使用。对于服务内通信,首选 gRPC 或 NATS 等协议,因为它们具有高效的压缩和二进制兼容性。 REST 协议用于公开与外部世界交互的服务。

DevOps 团队将部署和通信模式与 Kubernetes 等云原生平台公开的原语进行映射。

他们需要打包、部署和管理在生产环境中运行的这些服务。下一部分帮助将工作负载模式与 Kubernetes 原语对齐和映射。

## 将云原生工作负载映射到 Kubernetes 对象

Kubernetes 不仅仅是一个容器管理器。它是一个旨在处理打包在任意数量的容器和组合中的各种工作负载的平台。 Kubernetes 中内置了多个控制器,它们映射到云原生架构的各个层。

DevOps 工程师可以将 Kubernetes 控制器视为一种手段,用于规定您的团队正在运行的各种工作负载的基础设施需求。他们可以通过声明性方法定义所需的配置状态。例如,作为 ReplicationController 的一部分部署的容器/pod 保证始终可用。一个

打包为 DaemonSet 的容器保证在集群的每个节点上运行。声明式方法使 DevOps 团队能够利用基础设施即代码等范例。下面讨论的一些部署模式遵循不可变基础设施的原则，其中每个新的推出都会导致原子部署。

Kubernetes 的控制平面不断跟踪部署，以确保它们符合 DevOps 定义的所有配置状态。

Kubernetes 中的基本部署单元是 Pod。它是 Kubernetes 的基本构建块，是 Kubernetes 对象模型中最小、最简单的单元。一个 pod 代表集群上正在运行的进程。无论服务是有状态的还是无状态的，它总是被打包并部署为一个 pod。

控制器可以在集群内创建和管理多个 pod，处理在集群范围内提供自我修复功能的复制。

例如，如果一个节点发生故障，控制器可能会通过在不同节点上安排相同的替换来自动替换 pod。

Kubernetes 带有多个控制器来处理所需的 pod 状态。ReplicationController、Deployment、DaemonSet 和 StatefulSet 是控制器的一些示例。Kubernetes 控制器使用提供的 pod 模板来创建它负责维护所需状态的 pod。Pod 与其他 Kubernetes 对象一样，在 YAML 文件中定义并提交到控制平面。

在 Kubernetes 中运行云原生应用程序时，运维人员需要了解控制器处理的用例，以充分利用平台。这有助于他们定义和维护应用程序的所需配置状态。

上一节中解释的每个模式都映射到特定的 Kubernetes 控制器,这些控制器允许以自动化方式更精确、更细粒度地控制 Kubernetes 上的工作负载。

Kubernetes 的声明式配置鼓励不可变的基础架构。部署由控制平面跟踪和管理,以确保在整个应用程序生命周期中保持所需的配置状态。与基于虚拟机的传统部署相比,DevOps 工程师将花费更少的时间来维护工作负载。有效的 CI/CD 策略利用了 Kubernetes 原语和部署模式,使操作员从执行平凡的任务中解放出来。

可扩展层:无状态工作负载无状态工作负载被打包并部署为 Kubernetes 中的 ReplicaSet。ReplicationController 构成了 ReplicaSet 的基础,它确保指定数量的 pod 副本始终在任何给定时间运行。换句话说,ReplicationController 确保一个 pod 或一组同构的 pod 始终启动并可用。

如果 pod 太多,ReplicationController 可能会终止额外的 pod。如果太少,ReplicationController 会继续启动额外的 pod。与手动创建的 pod 不同,由 ReplicationController 维护的 pod 在失败、删除或终止时会自动替换。在中断性维护(例如内核升级)后,在节点上重新创建 Pod。因此,即使应用程序只需要一个 pod,也建议使用 ReplicationController。

一个简单的用例是创建一个 ReplicationController 对象以可靠地无限期地运行 pod 的一个实例。一个更复杂的用例是运行多个相同的横向扩展服务副本,例如 Web 服务器。

在 Kubernetes 中部署时,DevOps 团队和操作员将无状态工作负载打包为 ReplicationController。

在最近的 Kubernetes 版本中,ReplicaSets 取代了 ReplicationControllers。它们都针对相同的场景,但 ReplicaSet 使用基于 [集合的标签选择器](#) 这使得使用基于注释的复杂查询成为可能。此外,Kubernetes 中的部署依赖于 ReplicaSet。

部署是 ReplicaSet 的抽象。当在 Deployment 对象中声明了期望的状态时,Deployment 控制器以受控的速率将实际状态更改为期望的状态。

强烈建议使用部署来管理云原生应用程序的无状态服务。虽然服务可以部署为 pod 和 ReplicaSet,但部署可以更轻松地升级和修补您的应用程序。 DevOps 团队可以使用 Deployment 就地升级 Pod,而 ReplicaSet 无法做到这一点。这使得以最少的停机时间推出新版本的应用程序成为可能。部署为应用程序管理带来了类似平台即服务 (PaaS) 的功能。

[持久层:有状态工作负载](#) 有状态工作负载可以分为两类需要持久存储的服务 (单实例) 和需要以高可靠和可用模式运行的服务 (复制的多实例)。需要访问持久存储后端的 pod 与一组为关系数据库运行集群的 pod 非常不同。前者需要长期、持久的持久性,而后者需要工作负载的高可用性。 Kubernetes 解决了这两种情况。

单个 pod 可以由向服务公开底层存储的卷提供支持。该卷可以映射到任意节点上

预定哪个 pod。如果多个 pod 被调度到集群的不同节点并且需要共享后端，则在部署应用程序之前手动配置分布式文件系统，例如网络文件系统 (NFS) 或 Gluster。云原生生态系统中可用的现代存储驱动程序提供容器原生存储，其中文件系统本身通过容器公开。当 pod 只需要持久性和持久性时使用此配置。

对于需要高可用性的场景，Kubernetes 提供了 StatefulSets 一组专门的 pod，可保证 pod 的顺序和唯一性。这在运行主/从（以前称为主/从）数据库集群配置时特别有用。

与 Deployment 一样，StatefulSet 管理基于相同容器规范的 pod。与 Deployment 不同，StatefulSet 为其每个 pod 维护一个唯一标识。这些 pod 是根据相同的规范创建的，但不可互换：每个 pod 都有一个持久标识符，它在任何重新调度期间都会维护该标识符。

StatefulSets 对于需要以下一项或多项的工作负载很有用：

- 稳定、唯一的网络标识符。
- 稳定、持久的存储。
- 有序、优雅的部署和扩展。
- 有序、优雅的删除和终止。
- 有序的自动滚动更新。

Kubernetes 对待 StatefulSets 的方式不同于其他控制器。当一个 StatefulSet 的 pod 被 N 个副本调度时，它们是按顺序创建的，从 0 到 N-1。当 StatefulSet 的 pod 被访问时

删除后,它们会以相反的顺序终止,从 N-1 到 0。在将扩展操作应用于 pod 之前,其所有前驱必须运行并准备好。Kubernetes 确保在终止一个 pod 之前,它的所有后继者都完全关闭。

当服务需要运行集群时,建议使用 StatefulSets

Cassandra、MongoDB、MySQL、PostgreSQL 或任何具有高可用性要求的数据工作负载。

并非每个持久性工作负载都需要是 StatefulSet。某些容器依赖持久存储后端来存储数据。为了向这些类型的应用程序添加持久性,pod 可能依赖于由基于主机的存储或容器原生存储后端支持的卷。

**可并行化层:批处理** Kubernetes 具有用于批处理的内置原语,这对于执行运行到完成作业或计划作业非常有用。

Run to completion 作业通常用于运行需要执行操作并退出的进程。在处理数据之前一直运行的大数据工作负载就是此类作业的一个示例。另一个例子是处理队列中的每条消息直到队列变空的作业。

Job 是一个控制器,它创建一个或多个 pod 并确保指定数量的 pod 成功终止。当 pod 成功完成时,作业会跟踪成功完成的情况。当达到指定的成功完成次数时,作业本身就完成了。

删除作业将清理它创建的 pod。

一个 Job 还可以用于并行运行多个 pod,这使其成为机器学习训练作业的理想选择。作业还支持并行处理一组独立但相关的工作项。

当 Kubernetes 在带有 GPU 的硬件上运行时,机器学习训练可以利用 Jobs。Kubeflow 等新兴项目 一个致力于使机器学习在 Kubernetes 上的部署变得简单、可移植和可扩展的项目 将公开原语以将机器学习训练打包为作业。

除了运行并行作业外,可能还需要运行计划作业。Kubernetes 公开了可以在指定时间点运行一次或在指定时间点定期运行的 CronJobs。Kubernetes 中的一个 CronJob 对象类似于 Unix 中的一行 crontab (cron 表)文件。它按照给定的时间表定期运行作业,以 cron 格式编写。

CronJobs 对于安排定期作业 (例如数据库备份或发送电子邮件) 特别有用。

**事件驱动层 :无服务器计算**是指构建和运行不需要服务器管理的应用程序的概念。它描述了一种更细粒度的部署模型,其中将捆绑为一个或多个功能的应用程序上传到平台,然后根据当前所需的确切需求执行、扩展和计费。

函数即服务 (FaaS) 在无服务器计算的上下文中运行以提供事件驱动的计算。开发人员使用由事件或 HTTP 请求触发的功能来运行和管理应用程序代码。开发人员将小的代码单元部署到 FaaS,这些代码作为离散操作根据需要执行,无需管理服务器或任何其他底层基础设施即可扩展。

尽管 Kubernetes 没有集成的事件驱动原语来响应其他服务引发的警报和事件,但人们正在努力引入事件驱动的功能。[云原生计算基金会](#),

作为 Kubernetes 的托管人,有一个无服务器工作组专注于这些工作。开源项目,例如 Apache OpenWhisk,裂变,无管, OpenFaaS 和 Oracle 的 Fn 可以作为事件驱动的无服务器层在 Kubernetes 集群中运行。

部署在无服务器环境中的代码与打包为 pod 的代码有着根本的不同。它由自主功能组成,这些功能可以连接到一个或多个可能触发代码的事件。

当事件驱动计算(无服务器计算)成为 Kubernetes 不可或缺的一部分时,开发人员将能够部署响应 Kubernetes 控制平面生成的内部事件以及应用程序服务引发的自定义事件的函数。

遗留层:无头服务即使您的组织定期使用微服务架构构建应用程序并将其部署到云上的容器中,也可能有一些应用程序继续存在于 Kubernetes 之外。云原生应用程序和服务必须与那些传统的单体应用程序进行交互。

遗留层的存在是为了实现互操作性,以公开一组指向单体应用程序的无头服务。无头服务允许开发人员以自己的方式自由进行发现,从而减少与 Kubernetes 系统的耦合。Kubernetes 中的 Headless 服务不同于 ClusterIP、NodePort 和 LoadBalancer 类型的服务。

它们没有分配给它们的互联网协议 (IP) 地址,但有一个域名系统 (DNS) 条目指向外部端点,例如 API 服务器、Web 服务器和数据库。遗留层是一个逻辑互操作层,用于维护众所周知的外部端点的 DNS 记录。

微服务应用程序的每一层都可以映射到其中一个

Kubernetes 的控制器。根据他们希望部署的模式,DevOps 团队可以选择合适的选项。

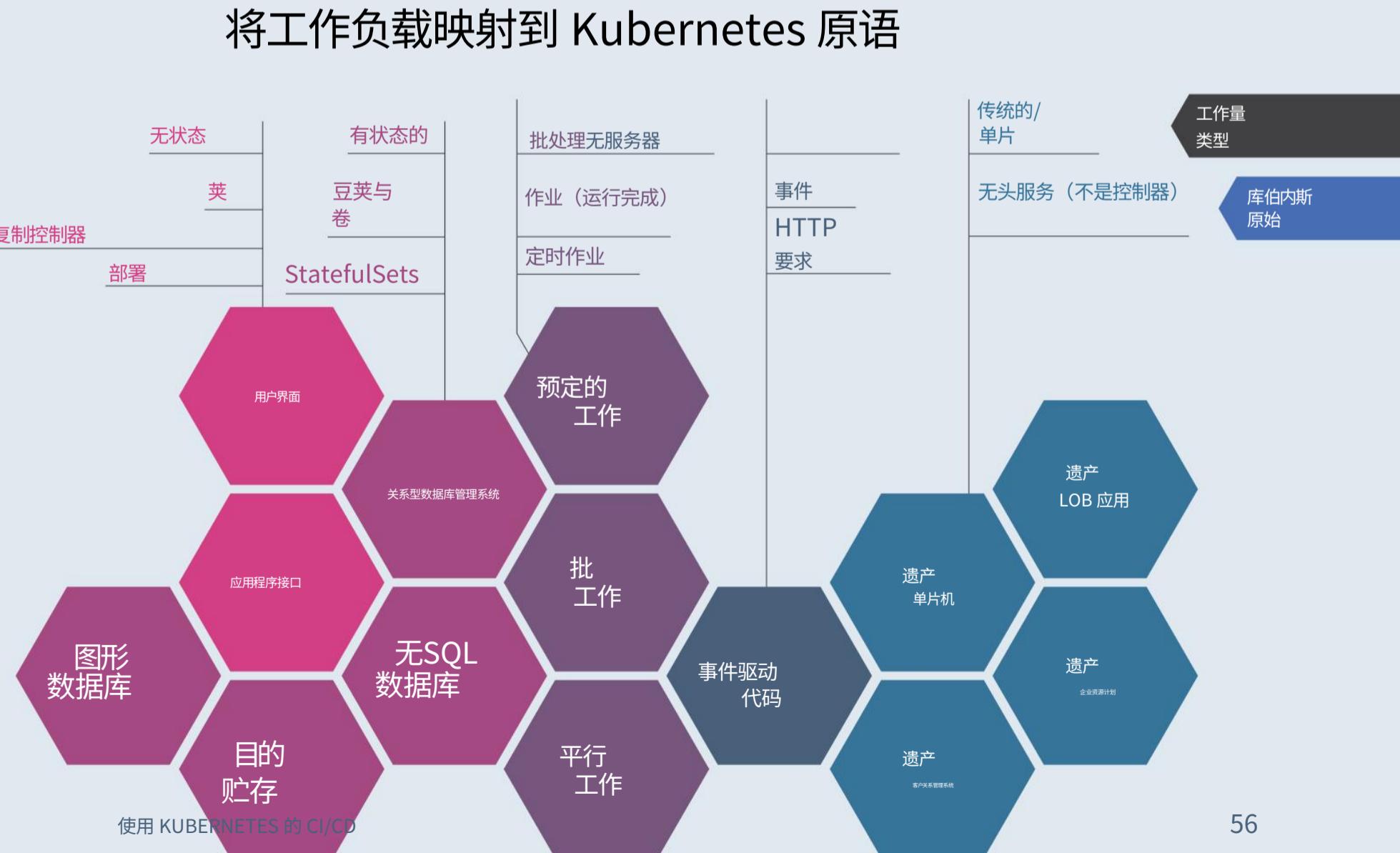
下面的描述 (图 2.2) 将各个层映射到具有 Kubernetes 原语的云原生应用程序堆栈。

## 部署云原生的最佳实践 Kubernetes 中的应用

了解云原生应用程序和 Kubernetes 原语之间的映射对于 DevOps 工程师来说很重要。但在部署这些工作负载时遵循一些公认的最佳实践同样重要。

以下是部署和运行云原生的 10 个最佳实践

图 2.2: DevOps 工程师可以通过声明性方法定义所需的配置状态 每个工作负载都映射到一个控制器。



Kubernetes 中的应用程序。这些技术帮助 DevOps 团队在 Kubernetes 上部署应用程序时获得最佳性能。

1. 永远不要部署“裸”Pod。裸 pod 是不属于 ReplicaSet、ReplicationController 或 Deployment 的 pod。由于简单,开发人员和运维团队的常见做法是将容器打包为简单的 pod 并将其部署在 Kubernetes 中。

裸 pod 会遭受单点故障,因为当节点出现故障时 Kubernetes 将无法重新调度它们。始终将 pod 打包为 ReplicationController 或 Deployment。

2. 为工作负载选择合适的控制器。Kubernetes 有各种映射到特定工作负载的控制器。根据工作负载的类型,从 ReplicationController、Deployment、StatefulSet、DaemonSet 和 Job 控制器中进行选择。

3. 使用初始化容器确保应用程序是  
初始化。Pod 是一个或多个容器的集合。Pod 中的 Init 容器在应用程  
序容器启动之前运行。

初始化容器类似于常规容器,但它们总是运行到完成,并且按顺序运行。如果一  
个 pod 有多个 Init 容器,则每个容器都必须成功完成,然后才能启动下一个  
容器。

Init 容器用于填充数据库表,将文件从远程位置下载到本地卷并检查其他  
pod 是否可用。以 Kubernetes 为目标时,避免使用边车容器。打包在同一个  
pod 中的 Sidecar 容器被 Kubernetes 视为普通容器。与 Init 容器不同,它  
们与 pod 的其他容器一起启动,因此很难确保在启动其余 pod 之前完成初始  
化。

#### 4. 在工作负载之前启动服务。先创建服务

在需要访问它的任何工作负载之前创建其相应的后端工作负载,例如 Deployments 或 ReplicaSets。当 Kubernetes 启动容器时,它会提供指向容器启动时正在运行的所有服务的环境变量。

当一个 pod 被调度时,它会自动填充同一个命名空间中创建的服务的环境变量。

#### 5. 使用部署历史回滚和前滚版本。

使用 Deployments 而不是 naked pod 和 ReplicaSets 的优势之一是能够回滚和前滚版本。始终使用部署记录来执行更轻松的回滚。此功能模仿处理平台即服务 (PaaS) 的便利性,例如用于部署的 Heroku 或 Engine Yard。

#### 6. 使用 ConfigMaps 和 Secrets。秘密安全地存储敏感

信息,例如密码、OAuth 令牌和安全外壳 (SSH) 密钥。将此信息转移到秘密中比将其嵌入 pod 定义或将其烘焙到容器镜像中更安全、更灵活。

ConfigMaps 允许开发人员将配置工件与图像内容分离,以保持容器化应用程序的可移植性。它们很好地替代了通过外部文件提供的硬连线配置。

#### 7. 为 Pod 添加 Readiness Probe 和 Liveness Probe。这

Kubernetes 控制平面依赖于就绪探测来了解容器何时准备好开始接受流量。当所有容器都准备就绪时,一个 pod 就被认为准备就绪。此信号的一种用途是控制哪些 pod 用作服务的后端。当 pod 未就绪时,它会从服务负载均衡器中删除。

Kubernetes 依赖于 liveness probe 来决定何时重启容器。当 liveness probe 遇到类似死锁的情况,应用程序正在运行但无法取得进展时,Kubernetes 会重启容器以重置状态。

## 8. 定义容器的 CPU 和内存资源限制

豆荚。 Pod 规范允许为每个容器指定资源限制。一个容器保证有它所请求的内存,但不允许使用超过定义限制的内存。 CPU 限制也是如此。 CPU 资源是

以 CPU 单位衡量,通常转换为虚拟 CPU (vCPU) 或相关基础架构的核心。

## 9. 定义多个命名空间以限制服务范围的默认可见性。 Kubernetes 支持由称为命名空间的同一物理集群支持的多个虚拟集群。对于 Kubernetes 集群中创建的每个服务,都有一个对应的 DNS 条目。此条目的格式为<service-name>.<namespace name>.svc.cluster.local,这意味着如果容器仅使用<service-name>,它将解析为命名空间本地的服务。这有助于在多个命名空间(例如开发、登台和生产)中使用相同的配置。

要从其他名称空间访问服务,可以使用完全限定域名 (FQDN)。

## 10. 配置 Horizontal Pod Autoscaling 以实现无状态工作负载的动态扩展。

Horizontal Pod Autoscaler (HPA) 根据观察到的 CPU 利用率或自定义指标支持自动缩放 ReplicationController、Deployment 或 ReplicaSet 中的 Pod 数量。 HPA 由 Kubernetes 控制平面管理。资源消耗影响控制器的行为。 controller 周期性的调整replicas的数量

ReplicationController 或 Deployment 以将观察到的平均 CPU 利用率与用户指定的目标相匹配。这种配置提供了无状态服务的最佳性能。

## 结论

Kubernetes 取得成功的原因之一是它为开发人员和运维人员提供的灵活性和控制力。开发人员专注于交付微服务，无需担心部署环境。DevOps 工程师获取软件，将层映射到适当的原语并将其部署在 Kubernetes 中。这种工作流以及软件设计和部署的解耦是 Kubernetes 的独特之处。

本章重点介绍了云原生应用程序的属性、将云原生工作负载类型映射到 Kubernetes 原语以及在 Kubernetes 中部署和运行应用程序的最佳实践。在下一章中，我们将探讨如何构建自动化部署的 CI/CD 管道。

# 提高安全性 自动图像 通过 CI /CD扫描



通过 CI/CD 管道实现自动化是保护部署在 Kubernetes 上的应用程序的关键。使用直接连接到 Jenkins 或您最喜欢的 CI/CD 工具的云原生安全工具,企业安全团队可以为正在构建容器映像的开发人员设置策略。该管道通过在构建过程中对每个映像进行自动漏洞扫描来执行这些策略。开发人员只部署安全团队确信的图像,因为它们已经过扫描。

“CI/CD 自动化是关键,因为规模很大,” [Liz Rice 说, Aqua Security 的技术传播者](#)。“当您一天可能运送数百或数千个部署时,您不可能手动检查所有这些不同的图像。”

在此播客中,了解分布式架构如何改变企业的安全方法、自动化如何大规模提高安全性以及 Aqua 的工具 (免费和商业企业版)如何帮助提高整个应用程序生命周期的安全性。[在 SoundCloud 上收听»](#)



Liz Rice 是容器安全专家Aqua Security 的技术传播者,她还从事与容器相关的开源项目,包括[kube-bench](#)和[宣言](#)。今年,她是 CNCF 在哥本哈根、上海和西雅图举行的[KubeCon + CloudNativeCon](#)活动的联合主席。

# 持续交付 带大三角帆

克雷格·马丁

而且,最重要的是,无聊。这让每个人都感到高兴。开发现代应用架构发布应该频繁、快速。

围绕 DevOps 概念组织软件团队和技术的不断发展的技术运动引起了人们对持续交付 (CD) 平台的极大兴趣。

DevOps 本身承诺了黄金机会:更快地部署需要新功能的市场,通过自动化提高部署的准确性,通过迭代更快地从客户那里获得反馈,以及更高的工作满意度,因为团队合作更快地获得功能和修复。 DevOps 是一段旅程,而不是目的地。这意味着建立具有共同目标的跨职能团队,围绕架构调整组织 [颠覆康威定律](#)- 并创造持续改进的文化。 DevOps 旅程中更高层次的成就之一是持续交付。

---

ThoughtWorks 封装了理想的 CD 思维模式。他们是这样描述的:

“连续的交付是持续集成的自然延伸,在这种方法中,团队确保对系统的每个更改都是可发布的,并通过按下按钮发布任何版本。持续交付旨在让发布变得乏味,这样我们就可以频繁交付并快速获得用户关心的反馈。”

单独实施 Kubernetes 并不能神奇地为您的组织实现 CD。然而,Kubernetes 在模块化、可用工具和不可变基础设施方面提供的特性无疑使 CD 更容易实施。尽管 Kubernetes 将帮助您定义容器部署和管理实例,但如何将这些部署自动化到环境中,则由您自行决定。

在本章中,The New Stack 问[Craig Martin](#),[Kenzan](#) 工程高级副总裁,讨论 Spinnaker 及其如何说明 CI/CD 实践为云原生架构演变的方式。Kenzan 是开源 Spinnaker 项目的主要贡献者,并使用它构建了自己的 Kubernetes 部署开源框架。在这里,Martin 借鉴了他与组织合作通过在 Kubernetes 和 Spinnaker 之上构建大规模微服务应用程序来实施数字化转型的经验来解释:

- 一种针对云原生架构出现的CI/CD 新方法。
- Spinnaker 作为基于 “状态”管理的 CD 工具的出现。
- 这种尚未得到广泛采用的新工具的优点和缺点。
- 在 Kubernetes 上使用 Spinnaker 设置 CI/CD 管道的最佳实践。
- 云原生堆栈上的CI/CD 新策略。

最终,公司如何在 Kubernetes 上使用 CD 对于实现按钮式、无聊的部署的理想非常重要。开发团队通常使用他们熟悉的既定工具 (即 Jenkins) 来回答 CD 问题。[The New Stack 的 Kubernetes 调查](#)中约有 45% 的受访者说他们使用 Jenkins 将应用程序部署到 Kubernetes。这是完全有道理的。

---

公司历来首先通过 Jenkins 中的版本控制代码和可重复构建来解决持续集成 (CI) 的问题。他们很自然地会扩展 Jenkins 的作业运行 CI 功能,然后解决自动化部署的问题。精通 CD 的开发人员甚至开始熟悉新的 Jenkins 2.0 管道。在 2.0 中,一个包含的管道插件允许您创建 Groovy 脚本来协调对构建、测试和部署阶段的细粒度控制。Blue Ocean 等其他插件允许您在 Jenkins 中可视化这些 CI/CD 管道。

虽然这些高度可定制的 Jenkins 管道可以通过复杂的脚本来构建,但它仍然引出了一个问题,即 Jenkins 是否是 CD 工作的正确平台,或者实际上哪个平台是云上 Kubernetes 部署的最佳工具.

## 进入大三角帆

CD 不同于 CI。CI 是一种持续合并和测试代码更改的机制,通常由 Jenkins 等工具实现。CD 是加速和自动化部署的尝试,运营商可以在一周内跨众多服务推出多个部署,并了解部署过程中应用程序和基础设施的确切状况。CI 工具无法提供的持续交付真正需要的是“状态”机。这样的状态机将能够将环境从一种状态变为另一种状态,直到它使

一直到生产。该机器将以自动化方式将 Docker 容器等环境转移到生产环境,甚至能够执行回滚、金丝雀部署和扩展实例等操作。这允许实现敏捷、按钮式、自动化的部署,这是理想的 CD 思维方式所推动的。

[大三角帆](#)提供了这样一个状态机。 Spinnaker 是由 Netflix 开发的开源持续交付平台,用于在其云网络上处理大规模 CD 操作。它是一种云原生管道管理工具,支持集成到所有主要的云提供商,即亚马逊网络服务 (AWS)、Azure、谷歌云平台和 OpenStack。它原生支持 Kubernetes 部署。

Spinnaker 不同于大多数其他 CI 工具。 CI 技术构建代码并对其运行测试。 CD 技术更侧重于显示环境的当前状态。 CD 通过自动测试软件并使用金丝雀测试和蓝绿测试等技术将其推入生产,使软件交付更进一步。

Spinnaker 不会取代 CI 工具。它与 Jenkins 久经考验的主力一起工作:Jenkins 作业仍然可以处理构建和存储 CI 部分的工件,并且一旦完成,该作业可以触发 Spinnaker 管道,将应用程序部署到 Kubernetes 或任何地方 CD 部分。

我们与几家公司合作,将 Jenkins 与 Spinnaker 并排实施,并且有时会收到皱眉,质疑最初的建议。为什么要费尽心思学习 Spinnaker 作为辅助 CD 工具并与之集成?

## 大三角帆特点

单独使用 Jenkins 来实现对管道的细粒度控制,这些管道可以执行自动化测试、回滚、可视化和模板化等操作

重用将需要相当多的使用 Jenkins 2.0 管道的自定义 Groovy 代码。完成所有这些工作后,您仍然没有真正的“状态”管理工具。

Spinnaker 已经掌握了这种状态管理,并提供了开箱即用的上述管道功能,远远超出了 Jenkins 等工具。

有许多功能赋予它这种精通:

- 云原生交付工具: Spinnaker 是云原生的。它是为云和在云中构建的。这意味着所有的好处

云 自动弹性、自动配置和自动缩放 是 Spinnaker 的原生功能。

- 基于微服务: Spinnaker 遵循微服务架构

模式。除了利用微服务,每个组件都是为了解决其领域的需求而构建的,利用领域驱动设计。这使 Spinnaker 具有高水平的模块化和可扩展性,可以满足您的特定需求。

- 交付和基础架构可见性: Spinnaker 提供了一个用户界面 (UI),允许您查看基础架构并准确查看代码所在的位置。这使您可以查看负载均衡器、区域、入口 IP 等。查看代码部署和基础设施的重要性怎么强调都不为过,因为它将简化您对部署的看法。

- 条件流水线:这是让流水线自旋的能力

当满足某些标准时,例如:基础设施、负载、区域和故障类型。虽然这可以通过其他工具实现,但我们发现 Spinnaker 管道最容易和最直观地配置条件逻辑。

·强大的 API： Spinnaker 被构建为一个消费应用程序编程接口 (API) 并且非常彻底,允许根据您的部署需要进行大量自定义。

·扩展能力:由于 Spinnaker 使用的微服务架构模式（下一节中有更多详细信息）,它可以实现高水平的可扩展性。每个组成 Spinnaker 的微服务都能够水平扩展,并且每个组件都内置了相当多的弹性。如果您需要高可扩展性,那么 Spinnaker 是黄金标准。

·内置部署策略:开箱即用,Spinnaker 提供了许多常用的部署策略,包括 Highlander、红/黑和滚动红/黑。您可以将它们放入您的管道中,并在您看到的任何地方使用它们。它允许操作员直接在 Spinnaker UI 中回滚失败的部署。

·部署方式的灵活性： Spinnaker 不仅适用于部署应用程序。它旨在涵盖堆栈的所有方面,包括基础架构组件和配置的部署。通过这种方式,您可以部署基础架构组件,然后再部署应用程序。

·多区域： Spinnaker 原生支持可以跨区域并行部署的交付管道。它甚至可以支持多云设置。

· Kubernetes 和容器感知： Spinnaker 中的管道可轻松与 Kubernetes 容器部署集成。 UI 甚至可以用来管理实例。您可以在高峰期从 UI 扩展部署的 pod,或根据需要终止它们。

·部署目标的灵活性： Spinnaker 支持多种

部署目标,包括主流云和容器平台,如 Amazon Web Services、Azure、Cloud Foundry、DC/操作系统、谷歌计算引擎 (GCE)、Kubernetes 和 OpenStack。无论目标环境如何,部署体验始终是一致的。

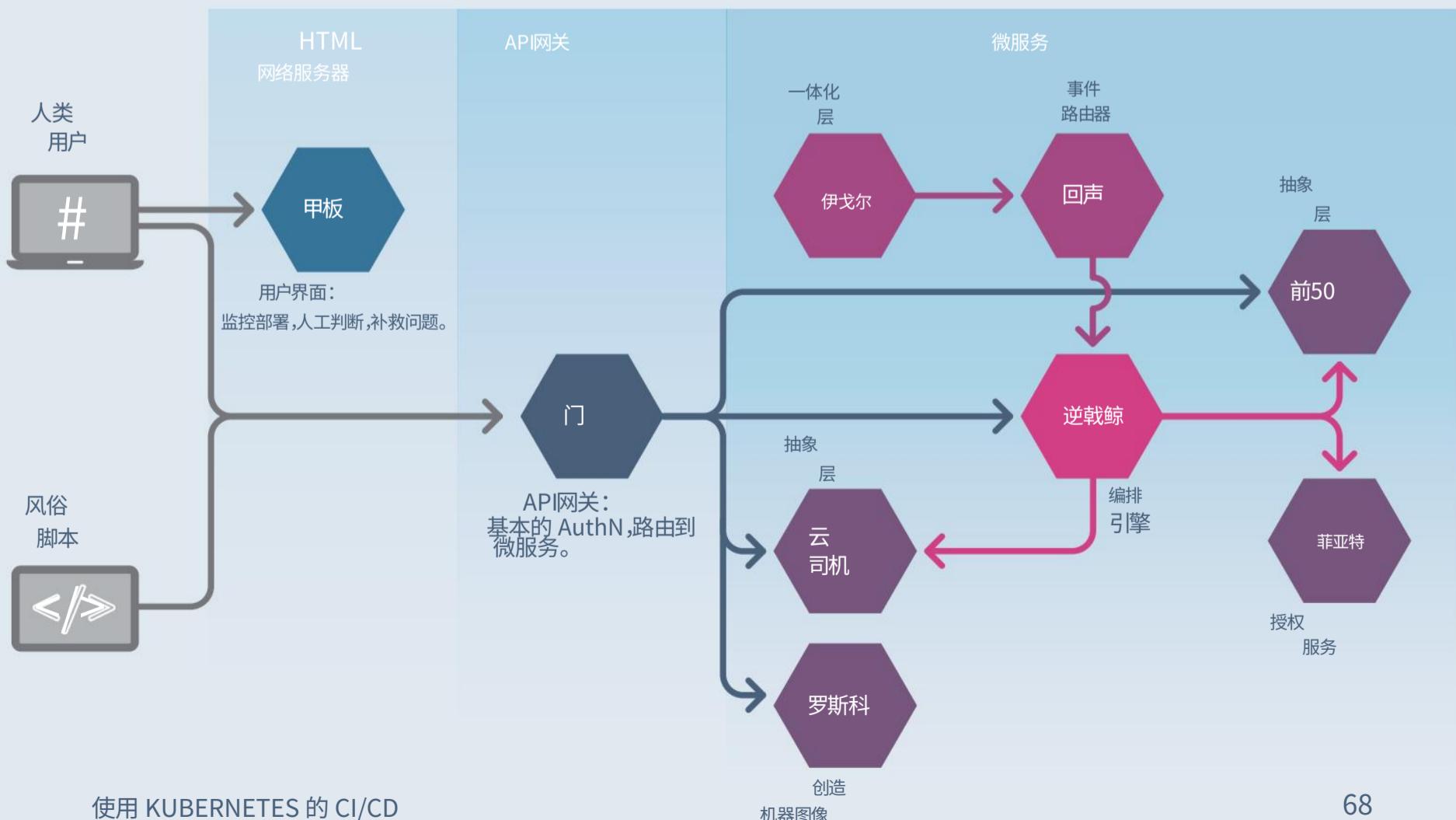
## 大三角帆架构

了解 Spinnaker 的组件架构对于了解其优势很重要。典型的 Spinnaker 安装包括许多微服务,它们协同工作以创建和管理基于管道的部署。

每个组件在其域内承担特定的角色和职责。

图 3.1: Spinnaker 模块化架构中的每个组件都有自己的职责。

### Spinnaker 组件及其作用



·甲板:用于监控部署、进行手动判断和修复问题的用户界面。

·门:用于执行基本 AuthN 和路由到微服务的 API 网关。

·逆戟鲸:用于实际执行的编排引擎

管道任务并执行工作。它使用Redis在执行时持久化数据。——

·云驱动: Spinnaker 使用的抽象层

与云供应商沟通。还存储已部署资源的本地缓存。

·前50持久化元数据的抽象层,配置,

通知等。默认情况下,它使用 Cassandra,但可以使用许多不同的数据存储。

·罗斯科:这是用于创建机器映像的微服务,例如:AWS Amazon 机器映像 (AMI)、Azure 虚拟机 (VM) 映像和 Google 计算引擎 (GCE) 映像。

·伊戈尔:持续集成工具中的集成层,例如 Jenkins 和 Travis;和 Git 存储库,例如 Bitbucket、GitHub 和 Stash。

·回声:这是一个事件路由器,用于传达所有关键信息

将事件发送给正确配置的侦听器,例如 Slack 和短消息服务 (SMS)。

·菲亚特: Spinnaker 的授权服务。它处理用户、应用程序和服务帐户的身份验证和授权。

·Halyard Spinnaker 的配置服务,用于安装、维护和升级 Spinnaker 本身。吊索是一个重要的

“类 Terraform”工具,为 Spinnaker 创建黄金部署配置,可用于恢复或生成额外的实例。

Spinnaker 具有智能组件架构,其中每个微服务域都有自己的 API,并且可以以模块化方式扩展。这允许更大的灵活性、更多的功能以及未来与任意数量技术的集成。

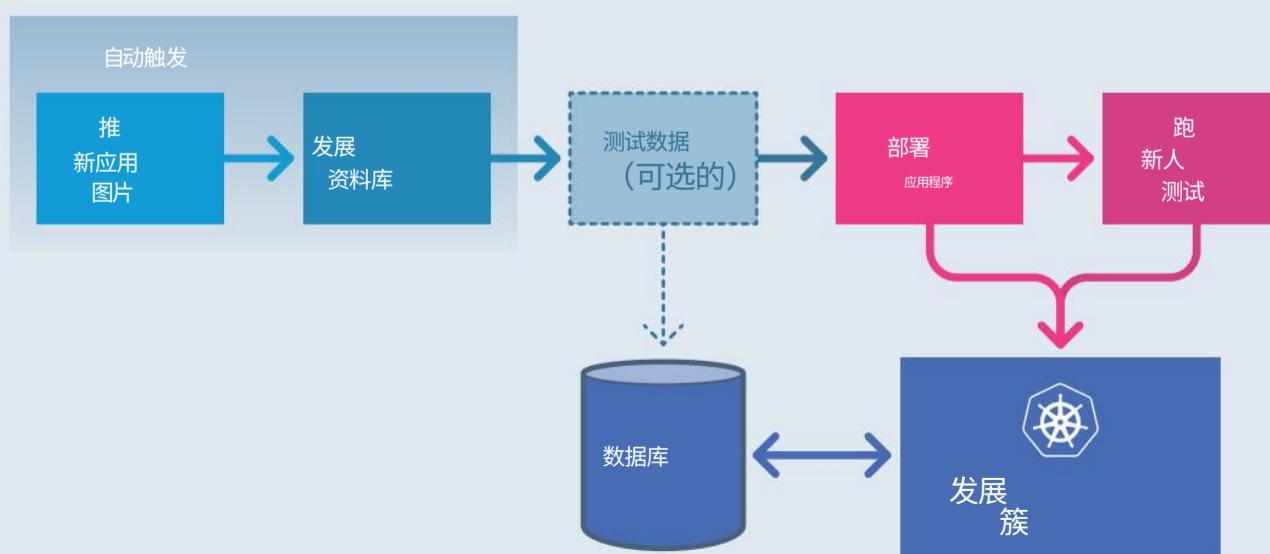
## 管道之美

虽然我们已经在 Spinnaker 中提到了管道的概念,但我们还没有描述它们的作用或看到它们的作用。通过在各个阶段编排可重复的部署,管道是 CD 功能的核心。

Spinnaker 中的每个管道都以一个特定的触发器开始。这可以是 Git 提交、手动启动、Jenkins 构建、推送到 Docker 存储库或另一个条件触发器。流水线通过几个用户定义的阶段进行。这些阶段可能涉及实际部署、针对特定环境运行自动化测试,甚至在冒烟测试失败时回滚部署。你想完成什么取决于

图 3.2:开发存储库中的新图像触发 Spinnaker 开发 (dev) 管道将应用程序部署到 Kubernetes 开发集群。

## 大三角帆开发管道



环境,因此通常会根据您部署的环境创建单独的管道。

## 开发环境

例如,在开发 (dev) 环境中,您可能会设置一个非常简单的 Spinnaker 部署,它可以运行。触发将新图像推送到容器图像存储库,它将应用程序部署到开发 Kubernetes 集群,然后在部署后使用 Newman 脚本针对应用程序的 REST API 运行许多卷曲。

## 暂存环境

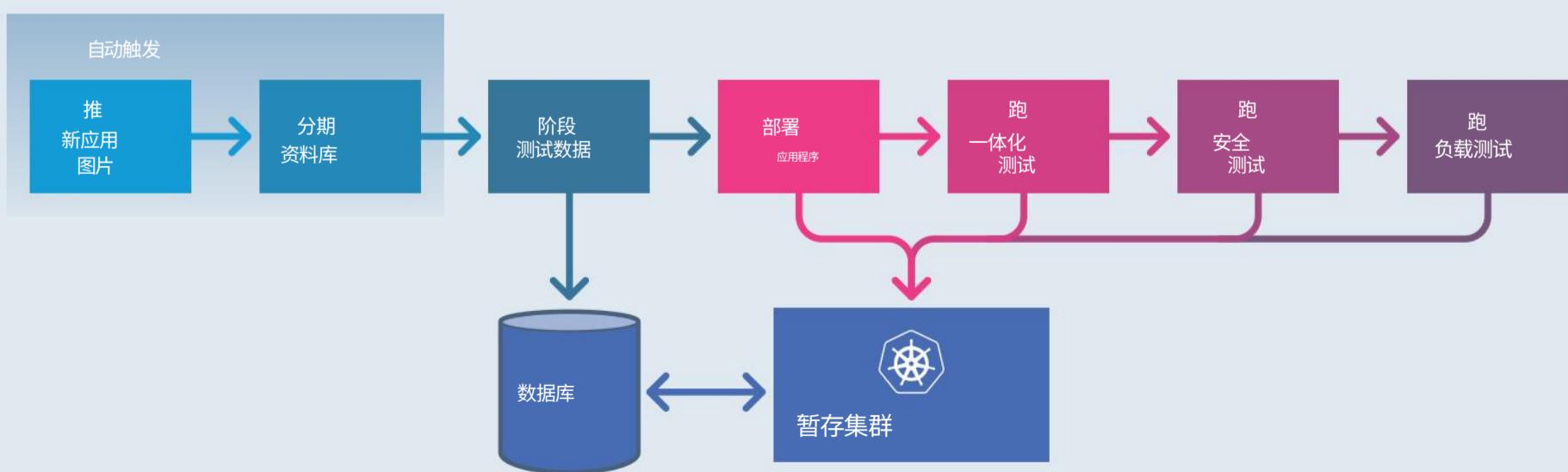
在暂存环境中,您可能会设置另一个稍微复杂一些的管道。触发推送到临时存储库的新图像,您首先使用数据库插入创建一些新的测试数据。接下来的几个阶段部署应用程序,利用测试数据运行集成测试,执行安全渗透测试,最后进行负载测试以模拟峰值。

## 生产环境

在生产 (prod) 环境中,我们允许操作手动启动管道,而不是触发 Jenkins。使用三角帆的

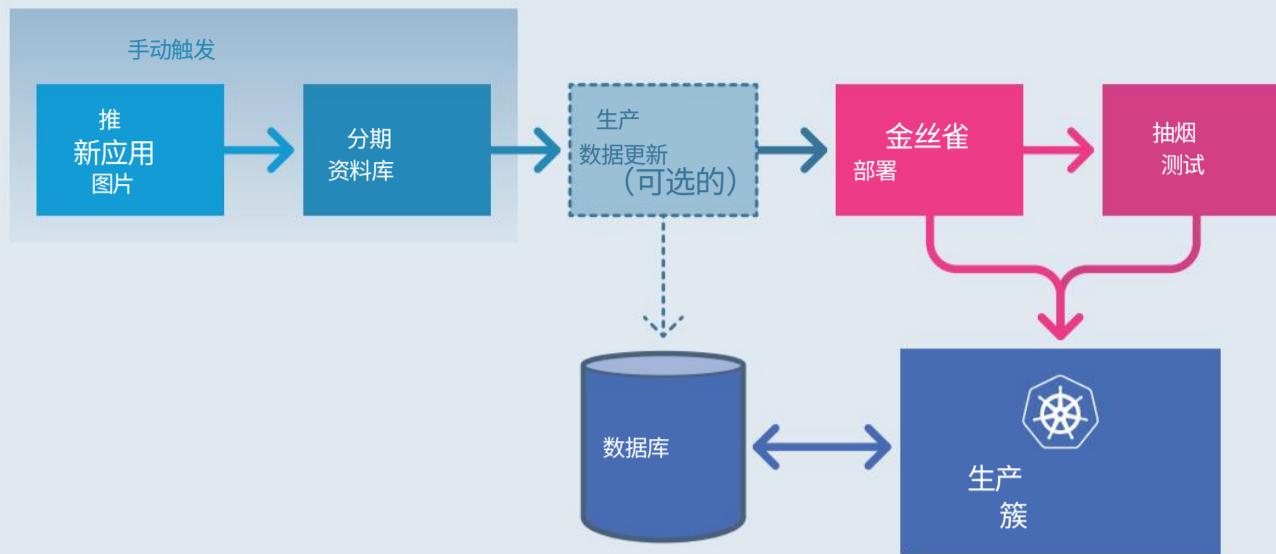
图 3.3:暂存存储库中的新图像触发 Spinnaker 暂存管道将应用程序部署到暂存集群。

### Spinnaker 暂存管道



# 大三角帆生产流水线

使用 SPINNAKER 持续交付



资料来源:肯赞

© 2018 THE NEW STACK

图 3.4:运营团队手动触发生产管道以将应用程序部署到 Kubernetes 上的生产环境。

内置功能,我们设置了金丝雀部署,逐渐将请求路由到应用程序的新版本,还包括一个执行冒烟测试的阶段。有关 Canary 部署的更多信息,请参阅[最佳实践](#)。

---

在这些不同的管道中,Spinnaker 的基础架构视图提供了有关管道中发生和未发生的事情、通过或失败的测试以及部署的版本的可见性。

拥有一个提供应用程序基础设施视图的 CD 编排工具可以支持运行中的应用程序更容易操作,提供一定程度的洞察力和控制力来创建跨环境的按钮部署。

上述所有管道都是在同一个 Spinnaker UI 中构建的。

尽管可以为更复杂的操作编写脚本管道,但 Spinnaker UI 中的模板使得将管道阶段串在一起相当容易,其中大多数工具和选项都是稳健部署所需的。根据我们的经验,管道及其管道模板的概念是 Spinnaker 真正闪耀的领域之一。创建[自定义管道模板](#)是创建可重复使用的管道模块的简单方法

应用于不同的情况。我们发现这是一种非常有用的方法,可以让每个团队控制他们的管道,但要确保单个组件只构建一次并正确构建。这些模板使重用变得自然,而不是需要最佳实践或冗长的文档,而自定义代码模板可能需要这些繁琐的工作。

虽然上述管道展示了在典型的开发、暂存和生产环境中部署可能是什么样子,但 CD 对快速部署的最终关注应该会让您质疑拥有这么多环境。能否在更少的环境中完成上述所有阶段?部署策略 (如金丝雀) 与 Spinnaker 在其管道阶段提供的自动化和准确性相结合,可以很好地提供一种使代码通过更少环境的途径,从而以更少的步骤将其投入生产。

## 示例实现

查看一些实际实现有助于了解 Spinnaker 的灵活性和强大功能。在 Kenzan, 我们帮助客户实施了以下许多 Spinnaker 用例。

### 案例 1:IaaS 的一切

对于云之旅不受支持遗留基础架构和应用程序负担的组织,他们可以选择利用基础架构即服务 (IaaS) 功能来构建应用程序。动机是 IaaS 现在提供各种经典的持续集成功能,允许组织使用他们的服务而不是实施特定的构建工具。例如,[谷歌云容器构建器](#)- 也用于创建用于发布的 Spinnaker 容器镜像 - 是一种谷歌云服务,可用于执行各种构建和低级集成操作。

Spinnaker 与 Google Cloud 和 Kubernetes 集成。在 Google Cloud Container Builder 中构建的容器镜像可以充当 Spinnaker 的管道触发器,然后 Spinnaker 可以将您的容器部署到 Kubernetes 集群上。使用这种云优先、支持 IaaS 方法的组织可以快速构建和部署具有所有适当系统开发生命周期 (SDLC) 功能的容器化应用程序。这为组织提供了不必承诺使用多个独立且不同的 SDLC 工具的优势。此外,通过现有的自定义 Spinnaker 阶段仍然保留灵活性,允许与各种 SDLC 工具集成。

该方法的结果是,组织可以将更多时间用于支持应用程序,而将更少时间用于应用程序工具。

## 案例 2:生成开发人员环境

Spinnaker 能够部署代码和基础设施。从零开始,它可以部署和管理整个堆栈:包括 Spinnaker 本身的功能环境。

为了展示这种能力,让我们假设我是一家基础设施即代码、微服务和容器化的公司。我的电子商务平台由 100 种不同的服务和这些服务的各种实例组成。我想启用以下 SDLC 条件和做法:

1. 我想让从事功能开发的开发人员能够获得整个平台环境的最后一次已知良好 (LKG) 的副本,除了他们正在增强的服务。
2. 我想要一次性的集成环境。
3. 我想生成一次性性能和笔测试环境。

这些是许多组织梦寐以求的目标,但由于复制环境的复杂性和这样做的成本,它们往往无法执行。

Spinnaker 通过基础架构感知和自我复制能力,为潜在的复杂问题提供了一个优雅的解决方案。我们实施的一个解决方案是拥有一个“Environment Creator Spinnaker”实例,它可以为

开发商。首先是部署 Kubernetes。然后使用预定义的 Halyard 定义将一个新的 Spinnaker 实例部署到 Kubernetes 集群上。然后管道定义以适当的顺序运行,以重新部署所有应用程序及其基础架构的所需版本。 Spinnaker 部署 Kubernetes 和 Spinnaker,运行管道以创建一个完整的一次性环境。通过这种方式,可以为开发、集成测试和性能测试创建单独的环境。

您可能会想,这很好,但是这样的环境的成本呢?您可能会在不知不觉中启动大量云资源并将它们留在原处,但后来才发现您的使用费用增加了两倍。为了帮助解决这个问题,可以在 Environment Creator 管道的末尾放置一个手动决策阶段,创建一个手动暂停,以便有足够的时间进行测试,然后通过破坏整个环境来完成管道。虽然没有办法避免创造这种一次性环境的全部成本,但 Spinnaker 可以通过减少他们的生存时间来提供帮助。

## 案例 3:使用 Spinnaker 进行数据库迁移

Spinnaker 不仅仅是一个用于部署应用程序的工具。它可用于自动化与基础设施相关的任务。例如,使用 Spinnaker 管道,您可以完成非常困难的模式迁移任务 通常称为数据库迁移。最多

组织将数据库迁移作为一个伪手动过程来执行。

有些人可能会尝试将架构更改作为应用程序启动的一部分。但是,当您有一个正在运行的应用程序尝试金丝雀发布时,这可能会导致不确定的数据表示。因此,数据库迁移被认为是奇怪的特殊情况,通常会导致“其中之一”部署。

Spinnaker 通过[启用数据库迁移](#)解决了这个问题直接在发布管道中运行。在应用程序的部署阶段之前,您可以使用一个阶段来执行数据迁移任务,这是数据库管理员 (DBA) 通常必须执行的任务。您甚至可以在每个版本的部署管道中包含一个“检查模式更改”步骤。这使得操作成本高昂的晦涩难懂的一次性任务与部署新容器镜像一样便宜和容易。

以这种方式自动化数据库迁移还提供了一定程度的可重复可靠性;替代方案 自定义脚本或手动步骤 通常更脆弱。

## 大三角帆作为标准

我们经常会问 Spinnaker 是否会成为未来标准 CD 工具。这很难预测,但这并没有阻止我们根据经验和我们在软件开发中看到的一般情况发表意见。简短的回答是,我们相信它会成为卓越的持续交付工具,尽管这需要一些时间。

Spinnaker 已经得到一些具有前瞻性思维的大型公司的支持,这些公司是该项目的主要贡献者,包括谷歌、微软、Netflix、甲骨文和 Pivotal。它是首批从一开始就构建的 CD 工具之一,可让 DevOps 团队真正利用云原生架构来提高效率、速度和敏捷性。但它

仍处于早期阶段。我们已经看到其他工具开始发展以包括云原生 CD 功能，并且还会有更多工具跟进。此外，该领域的任何竞争者都需要解决与构建工具集成或合并的需求，以实现完整的应用程序生命周期管理。 Spinnaker 是我们所见过的最接近于提供云原生持续交付工具所需的扩展能力和特性的工具，但仍需要进行大量开发工作才能使其成为交钥匙的企业就绪状态。

## 交付的文化转变

大多数公司才刚刚开始向基于 DevOps 的组织转型。对于这些公司而言，交付仍未被视为真正的优先事项或业务差异化因素。在 Kenzan，我们推动部署的速度和准确性对于产品与任何实际功能一样重要的概念。

公司通常很难接受这种心态。

对于涉及组织变革的公司来说，专注于部署是一种文化转变。它涉及改变团队结构、改变流程和改变文化，所有这些都需要更长的时间来扎根和成长。虽然公司可能意识到通过 Jenkins 等 CI 工具实现自动化部署的能力，但他们最初可能会忽视实现更快、自动化部署的长期战略计划，以及 Spinnaker 等功能齐全的 CD 工具可以提供的好处。正因为如此，采用像 Spinnaker 这样真正以部署为中心的平台可能在一段时间内都无法实现。

## 工具的复杂性转变

在等式的技术方面，Spinnaker 的初始版本被证明过于复杂并且涉及精心设计的设置。 Spinnaker 团队现在非常努力地简化 Spinnaker 的设置，但早期版本确实降低了社区采用它的难度。

幸运的是,当前的 Kubernetes 安装和设置是通过 Helm 可能是迄今为止配置和设置 Spinnaker 最简单的方法。这使得那些与 Kubernetes 集成的人更容易上手,但 Spinnaker 的一般复杂性仍然是许多组织的界限。

总的来说,我们已经逐渐看到复杂性从代码的构建和组装转向发布的编排。构建工具开始商品化并变得更加简单,例如 Travis CI 和 Jenkins。这种商品化是由使用不可变基础设施的简化推动的,特别是 Docker 容器,构建可以直接在它将运行的容器环境中进行,然后通过管道提升整个容器。随着越来越多的组织习惯于使用容器和其他不可变结构构建自定义代码,他们将花费更少的时间来构建该代码,并转向解决编排发布的问题。我们正在帮助我们的许多客户在考虑软件时看到“编排优先”的心态。在进行新的开发时,我们首先要考虑如何部署和自动化,然后再考虑它是如何构建的,甚至是功能中的架构。从这个角度来看,构建代码只是另一个实现决策。我们最终预见到大多数组织都会采取这种观点,一旦转变发生,Spinnaker 将非常适合需要 Spinnaker 提供的编排原生功能的新一波用户。还有其他工具,例如 Jenkins Pipelines,可以进行基本的编排,但它们无法扩展或几乎不具备 Spinnaker 的功能。

## 在人群中脱颖而出

总的来说,Spinnaker 已经进入了一个看似拥挤的 CI/CD 市场。它包括许多替代方案以及工具

就像前面提到的 Jenkins 2.0。大多数解决方案都是 CI 工具已通过 CD 功能进行了扩展,其他一些仅专注于 CD。

- [AWS 代码管道](#)是专门为 AWS 构建的 CD 服务。它允许创建管道以使用 AWS CodeBuild 构建和测试代码,然后部署应用程序。它还与 AWS CloudFormation 集成,允许您部署完整的应用程序堆栈。

它为部署提供单一 UI,但您需要使用其他 AWS 功能才能获得完整的可见性。

- [CircleCI](#)是一个以 CI 为中心的工具,已扩展到管道能力。它可以托管和自托管。它的最大功能在于不试图模仿 Jenkins 等遗留 CI 工具的架构。能够支持托管配置但仍然隔离构建运行时,使 CircleCI 成为运行时隔离很重要的企业级解决方案。除此之外,CircleCI 使 CI/CD 的入门变得容易。

[亚搏体育应用CI](#)是组织可以自行部署的领先 GitHub 克隆。如果您正在寻找一个自托管的 Git 并且不选择[Bitbucket](#), GitLab 是一个不错的选择,尤其是 GitLab 8+ 提供的 [CI/CD](#) 功能。顾名思义,GitLab CI 更多的是关于 CI,但是可以使用 GitLab 的流水线功能来完成 CD 任务。使用这些管道,可以对代码构建、Docker 构建和将映像部署到 Kubernetes 进行排序。然而,这些都是基于文件的配置,需要熟悉 Kubernetes 清单。

- [线条](#)是一种软件即服务 (SaaS) 持续交付工具可以集成到您的自定义环境中。 Harness 的最新添加使其能够开始利用人工智能

(AI) 预测和查看环境中的异常情况。 SaaS 的好处是它将抽象出建立您自己的 CD 工具的复杂性,但您可能会受限于对特定工具的定制。

·[Jenkins X](#)是一个令人兴奋的全新开源 CI/CD 工具集,于 2018 年 3 月发布。

它采用了 Jenkins 的许多核心功能,并对其进行了增强,使其成为云原生的。它是为

Kubernetes 并针对部署到该环境中进行了设计和优化。 Jenkins X 试图通过自动生成一些预定义的东西 Jenkinsfles、Dockerfles、Helm Charts、Kubernetes 集群、命名空间甚至环境来简化 CI/CD 的过程。它还使用预定义的自动化从 Git 提交触发构建和部署。虽然 Jenkins X 显示出很大的希望并非常快速地添加功能,但作为一种工具,它仍然在不断成熟。最值得注意的是,它只能从命令行运行,还没有用于管理部署的 UI,并且不容易用于管理 Kubernetes 集群底层的代码和基础设施（例如负载均衡器、图像和 DNS）。

上述工具包括市场上最流行和最相关的替代品;还有许多我们没有提到的其他 CI/CD 解决方案。重要的是要认识到,最初作为 CI 解决方案的工具现在正在适应 Spinnaker 从一开始就要做的事情:专注于基于云的部署。虽然一些 CI 工具可能会为部署提供基本级别的功能,但它们可能无法提供 Spinnaker 已经证明自己的可伸缩性、可扩展性和社区支持。最重要的是,这些工具可能需要很长时间才能与 Spinnaker 作为真正的状态机的功能相匹配:允许操作员自动更改、执行金丝雀部署、回滚和扩展实例,同时具有高

哪些应用程序和基础架构就位的级别视图。在检查任何工具的 CD 功能时,重要的是要记住理想的 CD 目标,即拥有一个可以为您的组织实现快速部署的状态机。

作为 Spinnaker 的一部分构建工具Spinnaker 社区可能会尝试推出自己的 CI 构建工具,该工具将与 Spinnaker 并存或成为其一部分。它是否取代 Jenkins 或其他工具并不像它所创造的机会那么重要。使用两种工具进行构建和编排通常是采用 Spinnaker 的障碍,人们最终会回到他们在 Jenkins 等工具中所知道的东西。如果将 CI 解决方案内置到 Spinnaker 中,它将打开使用一套工具完成整个构建和编排过程的大门。

## 大三角帆最佳实践

多年来,我们吸取了很多经验教训,我们尝试利用这些经验教训将客户转移到使用 Spinnaker 和 Kubernetes 的 CD 管道。

1. 确保弹性与基础设施中的任何应用程序一样,确保 Spinnaker 堆栈的弹性非常重要。

确保部署工具的正常运行时间与任何功能或应用程序一样重要。幸运的是,Spinnaker 在构建时已经考虑到了弹性,通常只需要配置即可满足您的特定需求。我们建议以下设置:

- 多区域**:如果您有一个多区域设置,则确保每个集群具有全套 Spinnaker 组件。

- 数据复制**:取决于您使用的数据存储

对于您的组件 Cassandra 和 Redis 配置它们以跨区域和数据存储进行复制非常重要。我们通常会发现开箱即用的数据复制配置,例如主动主动、主动被动和最终一致性,往往可以满足我们的安装需求。

#### ·混沌工程： Spinnaker 与混沌完美集成

猴子,我们发现这在我们的应用程序管道中非常有用,以确保它们的开发具有适当的弹性水平。

我们建议您也对 Spinnaker 设置执行混沌工程和测试。这将表明它可以处理故障,因此您可以确保您的部署管道不会受到中断的影响。

## 2. 使用命名空间

我们通常将 Spinnaker 安装在 Kubernetes 中的一个单独的命名空间中。这使我们能够在整个命名空间级别调整 Spinnaker 资源的大小,并且还可以防止 Spinnaker 资源从其他命名空间中夺走,反之亦然。对 Spinnaker 进行分区后,我们就可以密切监控部署微服务所需的资源。

从它自己的命名空间中,应该注意的是,Spinnaker 可以部署到其他特定的命名空间中。这是一项非常好的功能,允许部署一次只针对一个名称空间。

## 3. 监控 Spinnaker

监控 Spinnaker 与监控基础设施中的任何应用程序一样重要。 Spinnaker 目前支持三种主要的监控系统 Datadog、Prometheus 和 Stackdriver 但其他的可以相对容易地添加。我们建议使用与其他基础设施相同的监控工具。下一章会去

更详细地了解如何使用 Prometheus 进行监控。

·**了解监控:** Spinnaker 的设计具有非常强大的监控功能,但了解其工作原理至关重要。

每个微服务都被用来捕获和公开事件和指标,同时存在单独的守护进程来收集这些数据并将其推送到所选的监控工具。如需更深入的了解,请参阅[监控在 Spinnaker 中的工作原理](#)。

---

·**仪表板:**我们通常使用提供的仪表板

由我们的监控工具(例如 Prometheus)提供,但重点关注 Spinnaker 的两个主要视图。首先是 Spinnaker 的整体健康状况。这包括来自运行 Spinnaker 的集群、节点和命名空间的数据,以及任何集成点(例如 Jenkins)的健康状况。第二种观点是单独监控每个微服务。

我们通常会在微服务中寻找错误率和延迟。

·**权衡比率的价值差异:**我们从 Spinnaker 团队那里得到的一个监控实践是寻找数据趋势,而不是太在意当前的比率或计数。事实证明,这是非常好的建议,因为交付平台中负载的波动性非常高。

4. 版本化你的 ConfigMaps 和 Secrets如果你使用 Docker 容器之外的 ConfigMaps 和 Secrets,那么版本化它们是非常重要的。如果不对它们进行版本控制,如果配置证明不好,您将无法在 Spinnaker 管道中回滚。它可能会导致需要新构建和新配置的“前滚”情况。

5. 使用金丝雀部署虽然 Spinnaker 支持管道内的许多部署策略,但 Kenzan 热衷于使用金丝雀策略。这让我们可以慢慢滚动

输出该功能并直接在生产中进行测试。我们通常使用某种形式的自动金丝雀分析 (ACA) 来监控部署的运行状况，并将新部署的日志文件与之前的部署进行比较。幸运的是，现在谷歌和 Netflix 已经开源了[Kayenta](#)，这变得更容易实现，它与 Spinnaker 集成以监控金丝雀部署的质量。该版本于 2018 年 4 月发布，将使实现持续交付 然后是持续部署 比以往任何时候都更容易。

## 6. 使用特征标记

我们倾向于使用特征标记作为应用程序的设计实践。

这提供了在准备好后仍可交付该功能的能力，但在以后将其打开，并且不会创建等待通过 Spinnaker 部署的项目积压。

## 7. 使用流水线模板

使用[Spinnaker 流水线模板](#)允许您标准化一组管道组件，但仍然给每个团队一些灵活性来决定他们需要哪些部分，甚至是事件的顺序。这是一种非常强大的方法，可以确保在所有管道中进行重用。

# 更广泛的 CD 最佳实践

使用 Spinnaker 的转变不仅仅是将一项技术应用到位。它可能会与组织在思考和制定一些关键 CD 实践方面的转变齐头并进。

## 1. Mindset Shift 整体组织

需要支持 CD。这意味着业务、开发和运营需要调整他们的实践以支持持续交付。多年来，我们已经看到几个共同的主题出现，它们是这一旅程的关键：

- 关注发布代码而不是构建代码:这一点已被提及多次,但可能值得再次强调。重要的是,每个人都将交付质量和速度视为任何应用程序最重要的方面。
- 将资金作为产品而非项目:太多的组织仍在围绕项目提供资金和建立预算。每个项目可以有很多功能,有时这些预算不允许功能独立发布。为了确保优先发布,资金需要整体流入产品。
- 建立自动化文化:您的组织应该支持自动化文化。这涉及寻找新的自动化技术,但也避免任何无法自动化的基础设施或应用程序代码。这将扼杀管道的自动化。
- 您构建您拥有它:在我们的经验中,与构建代码的团队建立所有权是最重要的。每一段代码 微服务、管道和基础设施 都应该有一个明确的所有者,我们发现最好不要在管理团队和构建代码的团队之间创建分离。这些类型的分离通常最终会产生切换,从而停止或阻碍 CD 管道固有设计的速度和自动化。

## 2. 基础架构即代码如果您的基础架构

作为代码和配置进行管理,那么自动化一切都会变得更加容易。这将使在所有环境中创建一致性和奇偶性变得更加容易。实现这一点取决于您的具体实现,但我们通常使用某种脚本,例如使用 Terraform,并且还使用专用的基础设施管道来自动化部署。这确保了

如果没有适当的检查和平衡,个人永远不会手动调整环境。

3. 测试金字塔早期失败 自动化测试和早期失败对于加速交付管道很重要。在 Kenzan, 我们大量使用测试金字塔, 其中大部分测试发生在单元测试和集成测试的基础上, 然后我们使用更少的端到端和 UI 测试来进行基本的冒烟测试。

端到端和 UI 测试往往是最慢和最脆弱的。通过在运行速度更快的单元测试和集成测试中运行复杂的逻辑, 这使我们能够快速失败并在较低的环境中更早地发现错误。

4. 一致的分支模型 虽然 Spinnaker 确实能够处理多个 Git 分支模型, 但我们发现, 如果所有微服务和应用程序都在所有代码库中使用一致的分支和版本控制模型, 那么管理起来会更容易。如果没有一致的分支模型, 您最终会在 Spinnaker 中为不同的应用程序添加不同的分支触发器, 甚至可能为不同的版本控制和标记添加不同的分支触发器。根据我们的经验, 这会给开发带来不必要的复杂性。

5. 向后兼容的更改 每个更改都必须向后兼容! 时期。一旦您开始发布不向后兼容的功能, 回滚策略和耦合就会变得非常困难。这永远不应该发生。

6. 从小处着手 你不会在一周期内达到完整的 CD。该过程将需要时间来转移所有项目。在组织中实施 CD 时, 我们通常从单个应用程序或组开始非常小。这使您可以在较小规模上证明想法和组织变革, 然后将它们推广到更大的团队。

# 大三角帆教程

我们希望本章已经激起了您对 Spinnaker 作为 CD 平台的兴趣。如果您想开始使用该工具,Kenzan 有一个开源存储库,我们使用该存储库使用在 Google Kubernetes Engine (GKE) 中运行的 Spinnaker (与 Jenkins)设置一个功能齐全的 CD 环境。该工具使用一些简单的 Terraform 脚本来设置和配置您的环境,是开始动手检查 Spinnaker 的好方法。在以下位置查看: <https://github.com/kenzanlabs/capstan>

---

# 一种新的方法 使用SPINNAKER 进行开发 在 KUBERNETES 上



当组织将 DevOps 视为实现数字化转型的一种手段时,他们意识到他们必须加速端到端的软件交付过程,并使其更安全。

“这看起来像是一个沸腾的问题。这让你很难弄清楚你将如何逐步从中获取价值,”谷歌云平台 DevOps 部门的产品经理安德鲁菲利普斯说。

通过将开发人员反馈周期与推出过程分开,组织可以找到一个可管理的起点。Phillips 说,像 Spinnaker 这样的工具就是为此目的而构建的,“以提供一种抽象,以便开发团队可以获得简化的体验,同时仍然为运营团队提供管理、调整和定义它的能力对组织最有意义。”[在 SoundCloud 上收听»](#)



Andrew Phillips 是[Google Cloud Platform](#)的项目经理并且经常为持续交付和 DevOps 领域做出贡献。

他曾是一名软件工程师、团队负责人、基础设施建设者(又名管道胶带负责人)和社区布道者,并为许多开源项目做出了贡献。他是 ContainerDays Boston 和 NYC 的定期演讲者、作者和共同组织者。

# 监测中 云原生时代

作者：IAN CROSBY、MAARTEN HOOGENDOORN、THIJS SCHNITGER 和  
艾蒂安特雷梅尔

深入了了解云原生系统意味着知道要做什么，观察比什么都重要。这不仅仅是关于监控、警报、标记和指  
标。相反，这四种功能的结合使 DevOps 工程师可以观察在  
Kubernetes 等编排环境中的容器上运行的横向扩展应用程序。

一些云原生技术的早期采用者将可观测性称为新的监控。对可观测性不断增长的需求是真实存在的，它来自于理解复杂基础设施以及运行和构成云原生应用程序的众多组件所产生的原始数据的合理需求。随着越来越多的组织部署云原生应用程序，它将变得更加突出。对底层微服务架构的细致理解将使组织接受并接受与可观测性相关的概念。这种兴趣将推动对使用时间序列数据库收集数据的更深入功能的需求，这反过来又会带来更好的可观测性。

监控在今天具有不同的含义。过去,开发人员构建应用程序。部署由管理生产应用程序的运营团队完成。服务的健康状况主要取决于客户反馈和硬件指标,例如

磁盘、内存或中央处理器 (CPU) 使用情况。在云原生环境中,开发人员越来越多地参与监控和运营任务。监控工具已经出现,供使用它们的开发人员设置他们的标记并微调应用程序级指标以满足他们的兴趣。这反过来又使他们能够更快地检测到潜在的性能瓶颈。

开发人员正在应用持续集成/持续交付 (CI/CD) 等技术来优化可编程和不可变的基础架构。随着新基础设施的日益普及,对 DevOps 专业人员和[站点可靠性工程人员](#)的需求也在增加(SRE) 经验。可观察性为工程师提供了使系统和应用程序架构更稳定和更具弹性所需的信息。反过来,这为开发人员提供了一个[反馈循环](#),允许快速迭代和适应不断变化的市场条件和客户需求。如果没有这些数据和反馈,开发人员就会盲目行事,更有可能破坏事物。有了数据,开发人员可以更快、更有信心地行动。

将图形数据库集成到云原生监控工具 (如 Prometheus) 中,为此类工具提供了相当大的持久力。

通过捕获可以被视为与时间相关的图表的数据,此类工具允许开发人员以更精细的细节观察应用程序。

图形数据库将越来越多地作为获得更深入可见性的一种方式,并支持任意数量的监控用例。结果是应用程序架构的效率更高。组织可以

开始在不同的基础设施环境上构建自我修复的应用程序架构。

在这个新的云原生时代,对基础设施健康状况的持续了解决定了应用程序的构建、部署和管理方式。它确定组件如何以弹性方式自动修改 向上或向下 取决于负载。它告诉操作员如何做出有关故障转移或服务回滚的决定。就其本质而言,云原生系统是短暂且短暂的。它们随时可能失败,触发新的事件。它们扩展迅速,需要新的监控功能来涵盖各种情况。云原生监控不能独立对待任何特定组件,而是关注这些组件一起应该执行的聚合功能。

可观察性的主题是相当新的,但高度相关。我们的作者是软件工程师,他们研究了云原生架构中出现的新监控方法。 Ian Crosby、Maarten Hoogendoorn、Thijs Schnitger 和 Etienne Tremel 是 Kubernetes 容器解决方案 应用程序部署专家,为进行云迁移的客户提供支持的咨询组织。这些工程师在使用 Prometheus (已成为 Kubernetes 最流行的监控工具)以及作为可视化仪表板的 Grafana 进行监控方面拥有丰富的经验。

云原生环境中的监控需要超越对资源状态的检查;除了“我的 HTTP 服务正在响应”或“我的磁盘容量为 60%”之外,它还必须考虑其他因素。云原生监控环境必须提供对服务状态如何与其他资源状态相关的洞察。反过来,这必须指向系统的整体状态,该状态反映在包含多种考虑的错误消息中,例如

因为“HTTP 服务响应时间增加超过阈值,我们无法扩展,因为我们已经达到 CPU 资源限制。”正是这些类型的洞察力使系统能够根据明智的决策进行更改。这些见解定义了可观察性,并推动 DevOps 团队在通往真正的 CI/CD 反馈循环的道路上进一步前进。没有更深入的洞察力,问题可能就看不到了。在这个考虑可扩展性的云原生时代,监控是更广泛的可观察性实践带来的众多因素之一。

## 可观察性与上下文有关

云原生意味着不仅仅是在具有无限计算能力的云上托管服务。云原生系统由组装为微服务的应用程序组成。这些松散耦合的服务跨全球数据中心的云提供商运行。可能有数百万个这样的服务同时运行,产生大量的交互。在这样的规模下,不可能单独监控每个人,更不用说他们的相互依赖性和沟通了。

在规模上,上下文很重要。它显示了系统中的独立事件如何相互关联。对这种相互关系的理解是构建模型的基础,该模型有助于确定系统如何以及为何以特定方式运行。这不仅仅是收集尽可能多的数据的问题,而是收集有意义的数据以增加对行为的理解。可视化数据和指标,并在事件从一个组件流向下一个组件时跟踪事件,现在已成为监控微服务环境的现实。如果监控是观察系统随时间的状态,那么可观察性更广泛地说,是深入了解系统以某种方式运行的原因。

---

可观察性源于控制理论,它用来衡量系统的内部状态可以从其知识中推断出来的程度。

外部输出。对于现代 SRE,这意味着能够通过查看系统通过指标和日志公开的参数来了解系统的行为方式。它可以看作是监控的超集。

根据网络规模计算和微服务领域的先驱公司之一 Twitter 的说法,可观察性有四大支柱:

1. 记录。
2. 监控和指标。
3. 追踪。
4. 警报和可视化。

收集、存储和分析这些新型应用程序性能数据提出了新的挑战。

## 应用性能管理

云原生系统的动态特性对应用程序性能管理 (APM) 提出了新的挑战。首先,从定义上讲,云原生系统比传统系统更短暂、更复杂。构成系统的组件不再是静态的,而是短暂的 按需出现,不再需要时消失。需求的突然增加可能导致某些组件被大量扩展。任何 APM 解决方案都需要能够适应这些快速和大量的变化。

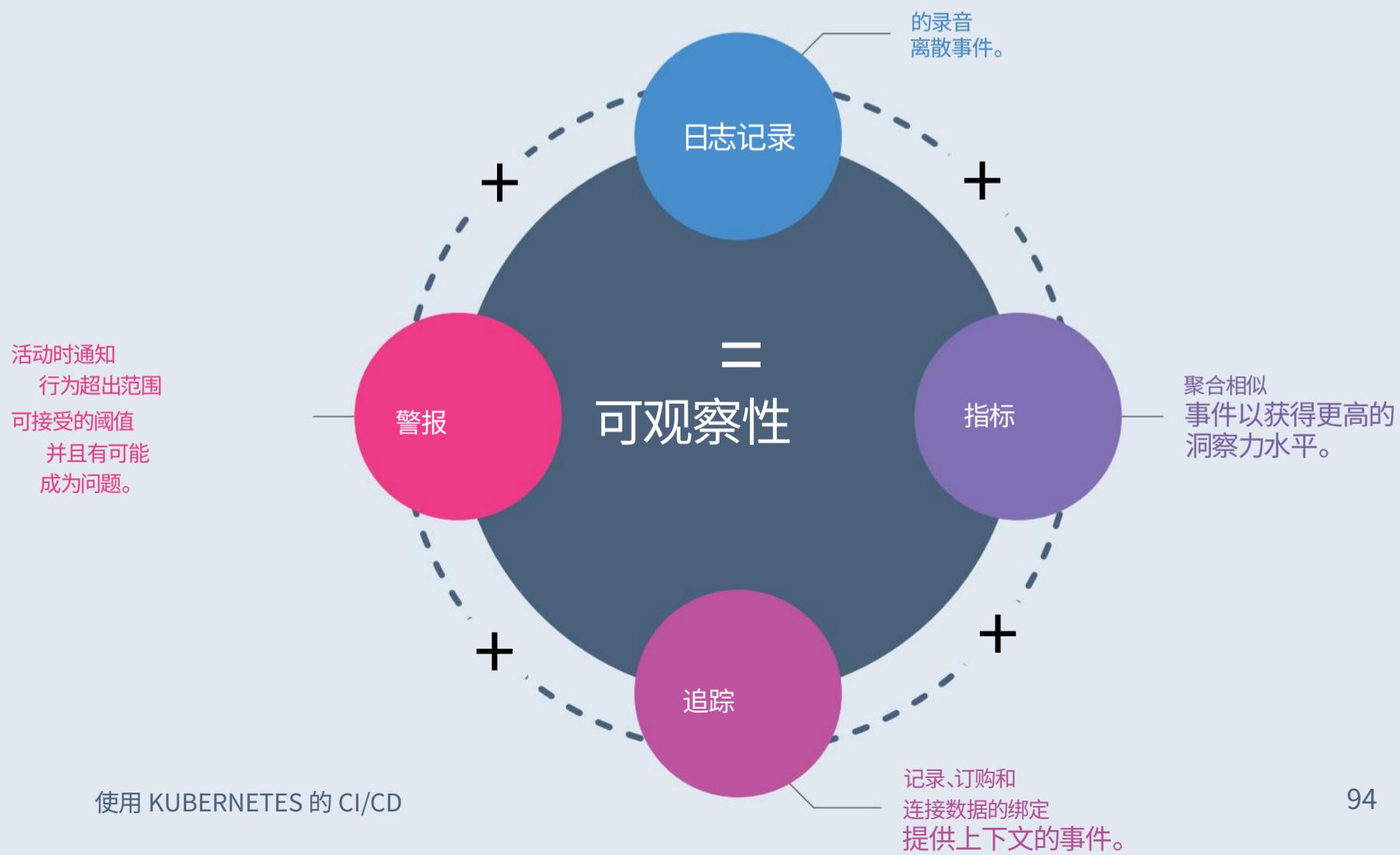
随着持续部署技术的使用增加,云原生系统中的组件也往往会更频繁地发生变化。这就需要将日志记录和指标不仅与特定时间点的系统状态相关联,而且还与导致该状态的软件更改相关联。此外,组件数量大大增加

增加记录的数据和指标的数量。在分析这些数据和指标时,这会增加对存储和处理能力的需求。这两个挑战都导致使用时间序列数据库,这些数据库专门用于存储按时间戳索引的数据。使用这些数据库可以减少处理时间,从而更快地得出结果。

这些大量数据还允许通过应用人工智能和机器学习的原理来获得洞察力。这些技术可以提高性能,因为它们允许系统通过学习先前变化的影响来适应其变化的方式,以响应它收集的数据。这反过来又导致了预测分析的兴起,它使用过去事件的数据来预测未来,从而防止错误和停机。

图 4.1:在云原生系统中,可观察性是新的监控。

## 可观察性的四大支柱



# 四大支柱

可观察性及其产生的洞察力来自日志记录、指标、跟踪和警报。这些可观察性支柱的价值来自于使用定义明确的术语并清楚地确定每个支柱的目的。从每个支柱中获取数据并用于以后的评估。让我们举一个简单的例子，一个 500 错误，看看 DevOps 工程师如何通过每个镜头获得洞察力。

## 日志记录

最简单意义上的日志就是记录离散的事件。这是任何新开发人员接触到的第一种监视形式，通常以打印语句的形式出现。在现代系统中，每个应用程序或服务都会在事件发生时记录它们，无论是标准输出、系统日志还是文件。然后，日志聚合系统将集中所有日志，以便根据需要查看或搜索。在我们发生 500 错误的示例中，这将被一个服务或多个服务可见，记录导致 500 状态代码的错误。这个错误可以通过对其他三个支柱的评估来破译。

## 指标

相比之下，指标是来自测量多个事件的数据的组合。云原生监控工具通过计数器、仪表、直方图和仪表等各种指标来满足不同类型的测量。

· 计数器：计数器是一个只能增加的累积指标。例如，服务的请求、完成的任务和发生的错误。

它不应该用于也可能下降的指标，例如线程数。

· Gauge（量规）：量规是可以任意上下的度量。为了

例如,温度、内存使用情况和实时用户数。

·直方图:直方图衡量一组事件的统计分布;例如,请求持续时间和响应大小。直方图跟踪观测值的数量和观测值的总和,允许用户查看观测值的平均值。

·计量器:测量事件发生的速率。利率可以

在不同的时间间隔内测量。平均速率涵盖应用程序的整个生命周期,而一分钟、五分钟和三分钟的速率通常更有用。

这个想法是聚合相似的事件以获得更高层次的洞察力。

指标通常是基于时间的,因此我们通常会定期收集指标,例如每秒一次。在我们的 500 错误示例中,我们可以看到特定服务忽略 500 错误的比率。如果我们有一个一致的 500 个错误率,这将指向一个与 500 秒的突然峰值不同的问题。

## 追踪

跟踪是关于记录和排序连接的事件。通过将唯一 ID 注入初始请求并将该 ID 传递给系统中的所有其他事件,所有数据事务或事件都被绑定在一起。在分布式系统中,单个调用最终会经过多个服务。跟踪提供了应用程序级别的完整画面。再次回到我们的 500 错误响应示例,我们可以看到导致 500 的特定请求的整个流程。通过查看请求通过了哪些服务,我们获得了有价值的上下文,这将使我们能够找到根源原因。

## 警报

警报使用模式检测机制来发现可能存在问题的异常。通过从以下位置创建事件来发出警报

通过日志记录、指标和跟踪收集的数据。一旦工程师确定了一个事件或一组事件，他们就可以根据潜在问题的严重程度创建和修改警报。回到我们的例子：我们如何开始调试 500 错误的过程？

建立阈值以定义构成警报的内容。在这种情况下，阈值可以由特定时间段内 500 个错误的数量来定义。五分钟内出现 10 个错误意味着对 Container Solutions 管理的操作发出警报。警报被发送到相应的团队，标志着调试和解决过程的开始。考虑到构成警报的内容还取决于系统的正常状态。

通过建立四个数据支柱，系统及其运行的云原生应用程序获得了可观察性。随着系统的开发，复杂性只会增加，需要更多的可观察性，即以可以存储和分析的方式收集更多数据，为更深入的优化和对应用程序的适当洞察提供反馈循环。

## 监控模式

在四大支柱中，指标提供了对应用程序性能的最深入了解。没有指标，就不可能判断应用程序的行为是否符合服务级别目标。有不同的策略用于收集和分析指标，以报告云原生系统的健康状况，这是最重要的问题。

黑盒和白盒监控是用于报告系统健康状况的两种不同策略。两者都依赖于不同的技术，这些技术结合起来可以增强报告的可靠性。

黑盒监控是一种无需访问应用程序内部结构即可确定系统状态的方法。收集的指标类型提供有关硬件的信息,例如磁盘、内存和 CPU 使用率或探测器 传输控制协议 (TCP)、互联网控制消息协议 (ICMP)、超文本传输协议 (HTTP) 等。

健康检查是黑盒监控的典型例子。它通过使用特定协议 (如 TCP 或 ICMP) 探测不同的端点来确定系统的状态。如果探测成功,则应用程序处于活动状态,否则我们可以假设系统已关闭而不知道确切原因。

与黑盒监控相比,白盒监控更为复杂,它依赖于遥测来收集应用程序行为指标,例如 HTTP 请求和延迟的总数,或者通过接口 (如 Java 虚拟机分析接口) 指定的错误或运行时的数量 (JVMPRI)。为了正确监控应用程序,必须指定此信息,并且由开发人员使用正确的指标对其进行检测。

黑盒和白盒监控是两种相互补充的模式,可以报告系统的整体健康状况。它们在云原生系统中发挥着重要作用,现代 SRE 解释这些指标以识别服务器性能下降并尽早发现性能瓶颈。

## 性能指标和方法

在云原生环境和复杂的分布式系统中,发现导致故障的原因需要时间和精力。只有少数几种方法,它们旨在简单快速地帮助 SRE 得出结论。每种方法都依赖于以下关键指标之一:

- 错误:产生错误事件的比率。
- 延迟:请求的持续时间。
- 利用率:系统的繁忙程度。
- 饱和度:服务无法处理额外的阈值工作。
- 吞吐量:请求系统的速率或数量。

根据这些指标,您可以应用以下四种方法之一来确定系统的性能:

- USE (利用率、饱和度和误差) :该技术由[Brendan Gregg 开发](#),是一种面向资源的方法,旨在检测负载下系统中的资源瓶颈。它依赖于三个指标:利用率、饱和度和错误。
- TSA (线程状态分析) :此方法是USE 方法的补充。也由[Brendan Gregg 开发](#),它专注于线程而不是资源,并试图找出哪个状态花费的时间最多。

它依赖于性能问题的六个关键来源:执行、可运行、匿名分页、休眠、锁定和空闲。

- RED (速率、错误和持续时间) :此方法针对请求驱动的服务。与TSA一样,RED 方法是 USE 方法的补充,它依赖于三个关键指标:速率、错误和持续时间。
- Golden signals:该方法由[谷歌SRE团队](#)推广并依靠四个关键指标来确定系统的状态:延迟、吞吐量、错误和饱和度。

在任何给定系统中,如果收集了正确的指标,工程师 即使他们不了解他们使用的系统的整个架构 也可以

应用其中一种方法可以快速找出系统的哪一部分有可能成为性能瓶颈并导致故障。

## 异常检测

在现代生产系统中,可观察性是检测和排除任何类型故障所需的核心功能。它帮助团队对可操作的项目做出决策,以便将系统恢复到正常状态。为解决故障而采取的所有步骤都应仔细记录并通过事后分析共享,以后可以使用它来加快重复发生事件的解决时间。

过去由运营团队处理的恢复过程现在由容器编排器处理。当恢复过程更加复杂时,可以开发额外的工具来自动执行恢复步骤,从而使系统恢复到正常状态。可以根据指标和阈值或机器学习等其他一些预测机制来触发恢复决策。

实现基于反复出现的问题的自我修复能力是使系统具有弹性的第一步。事后分析中描述的观察和解决方案可以转化为可操作的项目,供未来决策使用。

分析可以揭示很多关于系统行为的信息。根据历史数据,可以在潜在趋势成为问题之前对其进行预测。这就是机器学习发挥作用的地方。机器学习是一组算法,可以逐步提高特定任务的性能。从观察到的行为来解释系统的特征是有用的。有了足够的数据,发现不符合“正常”行为模型的模式是一个优势,可用于减少误报并帮助决定将采取的行动。

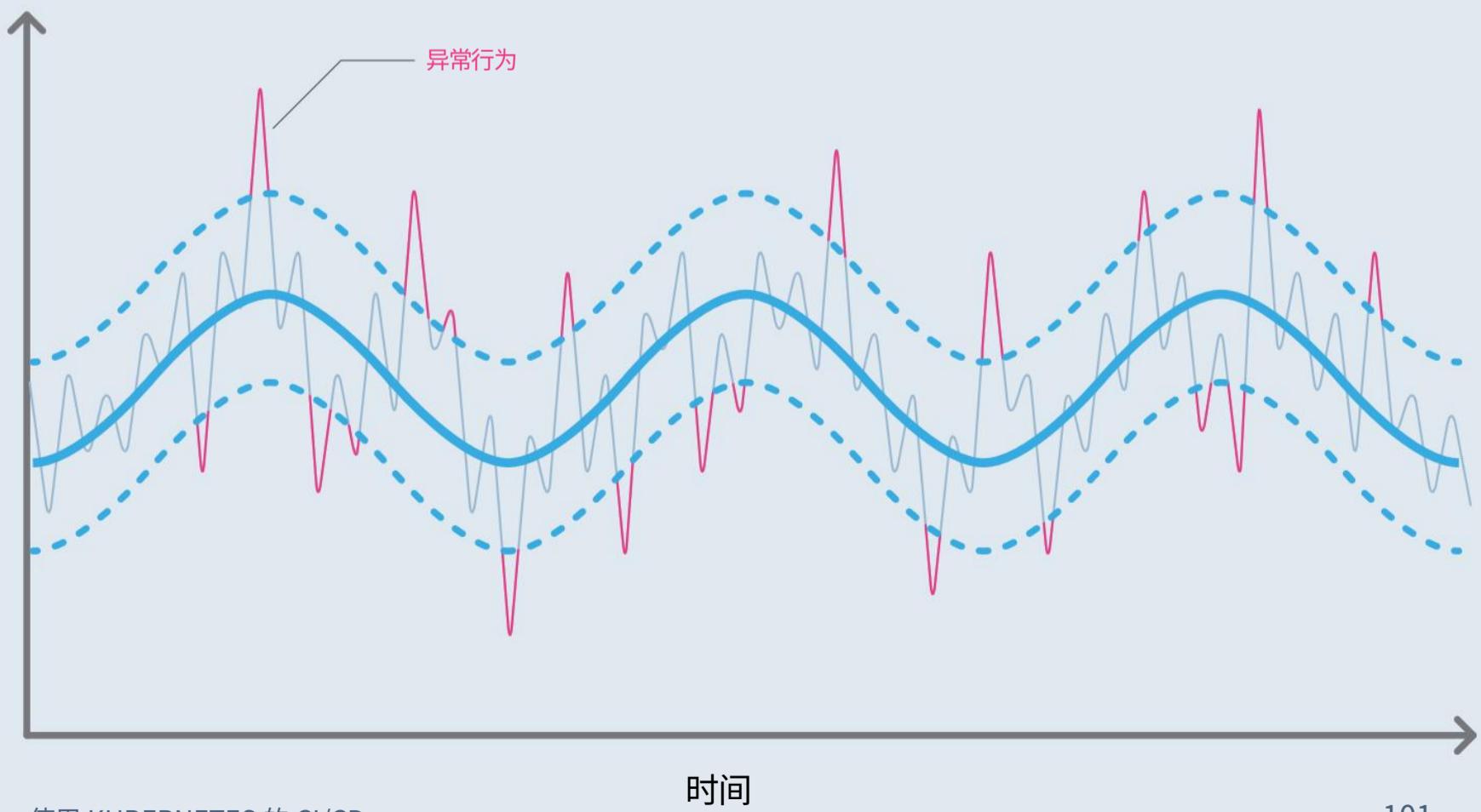
尝试使系统恢复到正常状态。

监督和非监督是异常检测的两种类型。标记为正常或异常的模型称为监督模型。当数据集用于训练模型然后随后被标记时,它就得名了。它有局限性,因为贴标签既困难又昂贵。相反,无监督检测无法识别错误的数据集。基于很少发生或不重复的事件,可以使用贝叶斯网络中的标准推理将它们归类为异常,并使用概率计算排名。

由于将其集成到监控系统中可能很复杂,因此更简单的方法是使用三重指数平滑法,也称为 Holt-Winters 方法。这有可能提供准确的预测,因为它将季节性波动纳入模型。这是

图 4.2: Holt-Winters 方法有可能提供准确的预测,因为它结合了季节性波动来预测一系列随时间变化的数据点。

## 异常检测的 Holt-Winters 方法



可用于预测一系列数据点随时间变化的众多方法之一。图 4.2 概述了 Holt-Winters 方法评估的内容。

## 推送与拉取数据收集

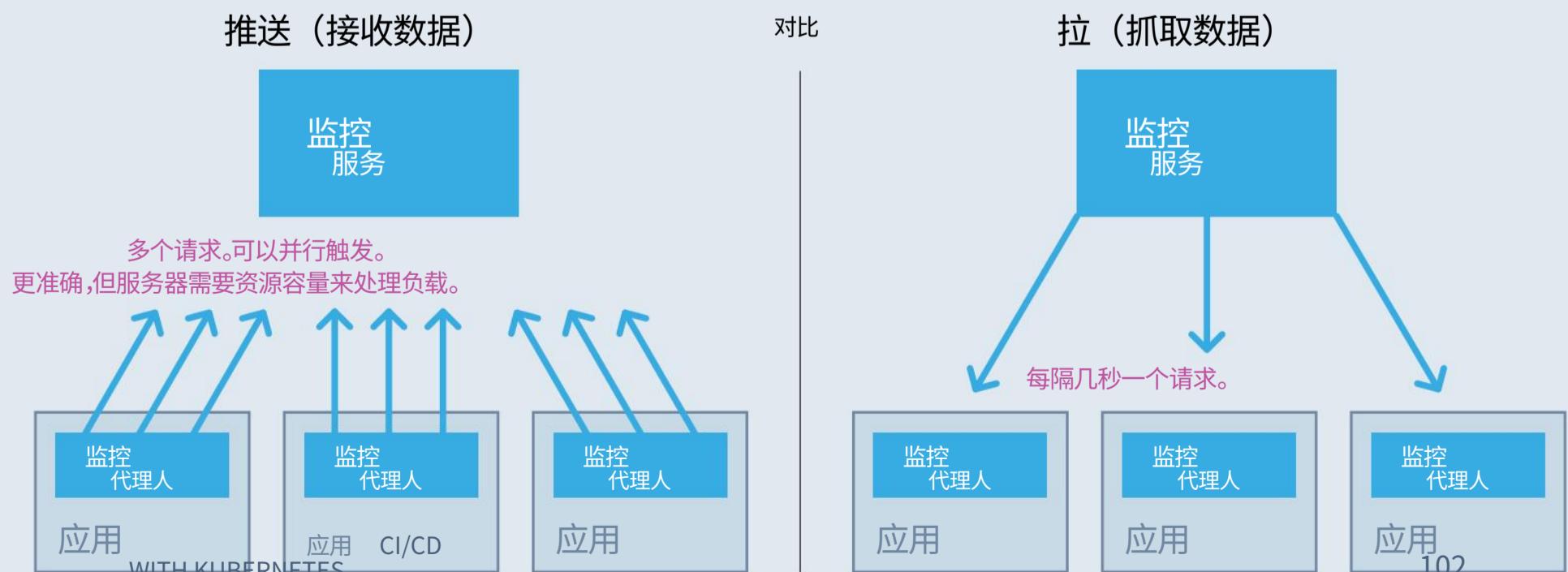
在从应用程序收集指标时,我们可以区分两种模型:推送和拉取。许多监控解决方案都希望得到数据,这就是所谓的推送模型。其他人接触服务并抓取数据,这就是所谓的拉动模型。在这两种情况下,开发人员都需要检测其应用程序的特定部分以测量其性能 执行任务的时间、外部请求所花费的时间等 并在以后对其进行优化。根据用例,一种方法可能比另一种方法更好。

推送模型最适合事件驱动的时间序列数据集。它更准确,因为每个事件都是在源头触发时发送的。

使用推送模型,需要一些时间来判断服务是否不健康,因为实例健康状况基于它接收到的事件。推动

图 4.3:许多监控解决方案都希望传递数据,这被称为推送模型。其他人则接触服务并抓取数据,这被称为拉模型。

### 收集数据的两种方法



当无法访问实例以拉取指标时,模型会假设该实例不健康。

每个都有不同的目的。对于大多数用例来说,拉模型是一个不错的选择,因为它通过使用标准语言来强制执行约定,但它确实有一些限制。从物联网 (IoT) 设备或浏览器事件中提取指标需要付出很多努力。相反,推送模型更适合此用例,但需要固定配置来告诉应用程序将数据发送到哪里。

在云原生环境中,由于拉动模型的简单性和可扩展性,公司倾向于青睐拉动模型而不是推动模型。

## 大规模监控

可观察性在任何大型分布式系统中都扮演着重要的角色。随着容器和微服务的兴起,当您开始抓取如此多需要横向扩展的容器时会发生什么?你怎样才能让它高可用?

有两种方法可以解决这个大规模监控的问题。第一个是使用联合监控基础设施的技术解决方案。

联合允许监控实例从其他监控实例收集选定的指标。另一种选择是一种组织方法,通过采用 DevOps 文化来改进监控,并通过为团队提供自己的监控工具来赋予他们权力。

这种重组可以进一步划分为领域 前端、后端、数据库等 或产品。拆分有助于解决团队按角色拆分时可能出现的隔离和耦合问题。通过提前确定角色,您可以防止出现“我将忽略该前端警报,因为我目前正在处理后端”这样的情况。第三种选择,也是最好的,是两者的混合

方法:采用 DevOps 并联合一些指标,从各种监控实例中提取一些顶级服务水平指标。

## 联邦

当一组应用程序在多个数据中心或气隙集群上运行时,一种常见的方法是为每个数据中心运行一个监控实例。拥有多个服务器需要一个“全局”监控实例来聚合所有指标。这称为分层联合。

很久以后,由于系统负载过高,您可能会发展到抓取速度太慢的地步。发生这种情况时,您可以启用分片。分片包括跨多个服务器分布数据以分散负载。这仅在一个监控实例处理数千个实例时才需要。通常,建议避免这种情况,因为它会增加监控系统的复杂性。

## 高可用性

高可用性 (HA) 是一种分布式设置,它允许一个或多个服务出现故障,同时保持服务始终正常运行。一些监控系统,如 Prometheus,可以通过同时运行两个监控实例来实现高可用性。它抓取目标并将指标存储在数据库中。如果一个出现故障,另一个仍然可以被抓取。

但是,在高可用性系统上发出警报可能很困难。DevOps 工程师必须提供一些逻辑来防止警告被 fred 两次。显示仪表板也可能很棘手,因为您需要一个负载均衡器在一个出现故障时将流量发送到适当的实例。由于每个实例可能在不同的时间收集数据,因此存在显示略有不同的数据的风险。启用“粘性会话”

负载均衡器可以防止这种不同步时间序列的闪烁显示在仪表板上。

## 用于云原生监控的 Prometheus

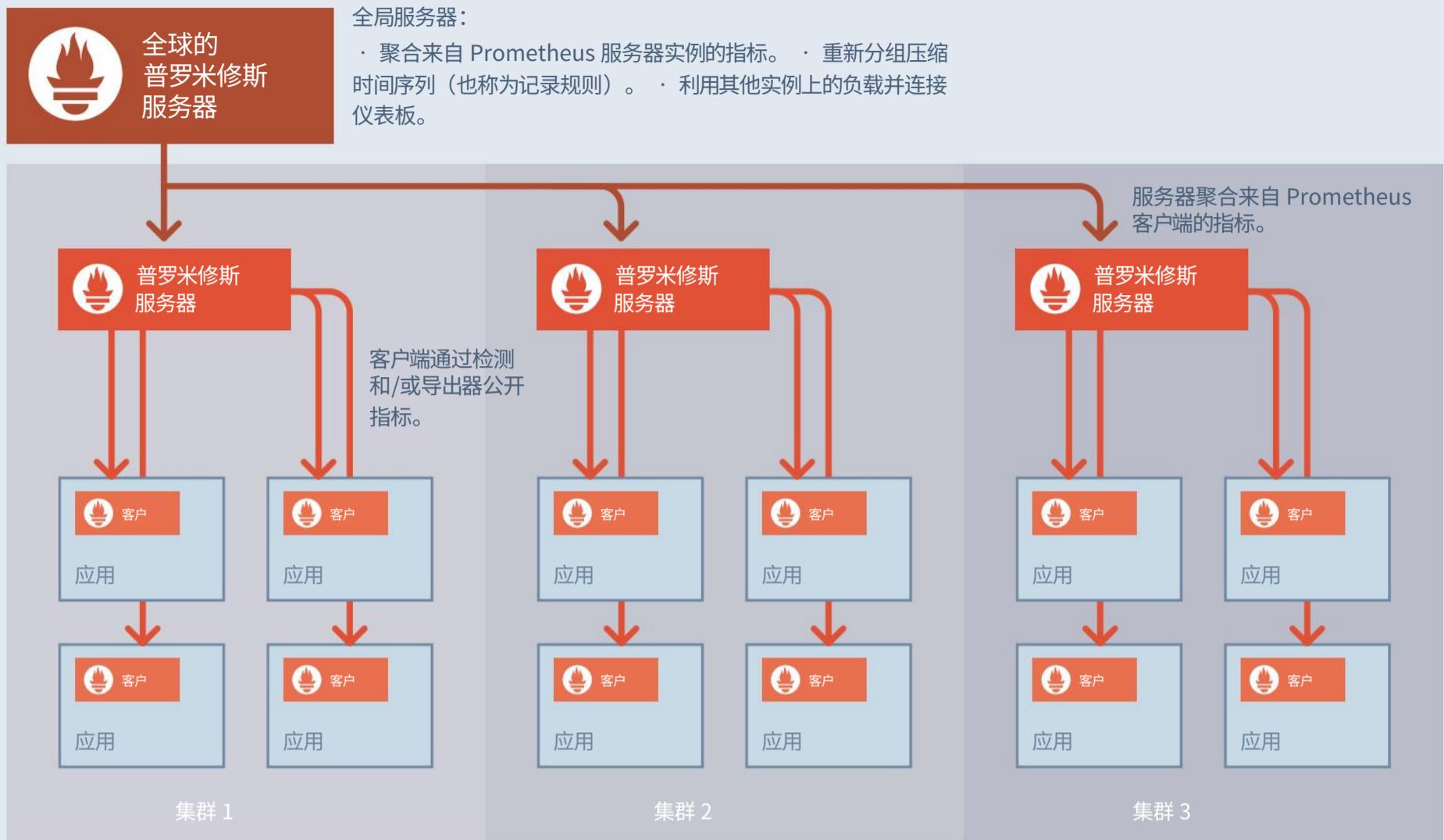
企业越来越多地转向基于微服务的系统来优化应用程序基础设施。当大规模完成时,这意味着对数据有一个细粒度的理解以提高可观察性。

由于 Kubernetes 架构的互连特性,运行微服务的应用程序非常复杂。微服务需要监控、跟踪和日志记录以更好地衡量整体基础设施性能,并且需要对原始数据有更深入的了解。传统的监控工具更适合通过配置节点的检测来监控的遗留应用程序。在微服务上运行的应用程序是使用在不可变基础架构中的容器上运行的组件构建的。它需要将复杂的软件转化为复杂的系统。服务级别域的复杂性意味着传统监控系统不再能够确保可靠运行。

Prometheus 是解决该问题的一个简单但有效的开源解决方案。它的核心是一个时间序列数据库,但关键特性在于它使用了拉模型。它从服务中抓取和提取指标。仅此一项就使其健壮、简单和可扩展,与微服务架构完美契合。 Prometheus 最初由 SoundCloud 开发供内部使用,是一种基于 Google Borgmon 思想的分布式监控工具,它使用时间序列数据和指标让管理员了解他们的操作执行情况。它成为[云原生计算基金会](#)采用的第二个项目 (CNCF) 在 Kubernetes 之后,它允许项目社区之间进行一些有益的协调。

# 作为服务而非机器进行监控

云原生时代的监控



资料来源:<https://www.slideshare.net/brianbrazil/prometheus-overview>

© 2018 THE NEW STACK

图 4.4: Prometheus 在分层、联合架构中的表示。

普罗米修斯的主要特点是：

- 简单。
- 从服务中提取数据,服务不推送到Prometheus。
- 不依赖分布式存储。
- 没有复杂的可扩展性问题。
- 通过服务发现或静态配置发现目标。
- 称为PromQL 的强大查询语言。

Prometheus 在微服务架构中运行良好。它处理

简单高效地处理多维数据。它也是关键任务系统的一个很好的选择。当系统的其他部分出现故障时,Prometheus 仍会运行。

Prometheus 也有一些缺点：准确性就是其中之一。

Prometheus 会抓取数据，但不保证会发生此类抓取。如果您有需要准确性的服务，例如按使用量计费，那么 Prometheus 不是一个好选择。它也不适用于非 HTTP 系统。HTTP 是 Prometheus 的主要编码，因此如果您不使用 HTTP，而是使用 Google 远程协议过程 (gRPC)，例如，您将需要添加一些代码来公开指标（请参阅[go-grpc-prometheus](#)）。

---

## 普罗米修斯的替代品

根据[CNCF 2017 年秋季社区调查](#)，Grafana 和 Prometheus 是 Kubernetes 用户首选的监控工具。

---

64% 使用 Kubernetes 管理容器的组织使用开源数据可视化工具 Grafana，Prometheus 紧随其后，占 59%。这两个工具是互补的，用户数据显示它们最常被一起使用。大约 67% 的 Grafana 用户也使用 Prometheus，75% 的 Prometheus 用户也使用 Grafana。

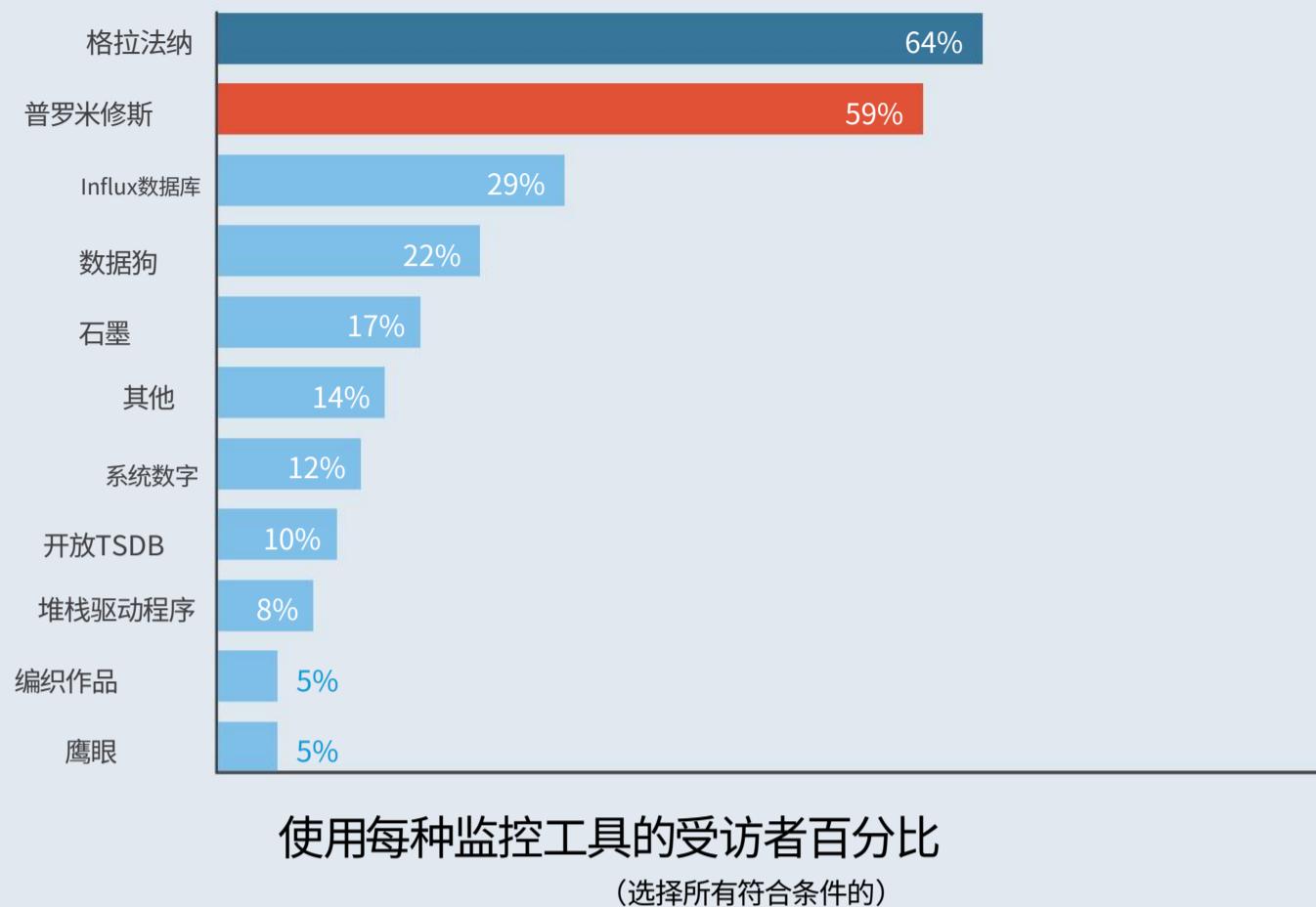
根据 CNCF 的调查，由于不同程度的重叠功能，Kubernetes 用户经常同时使用多个监控工具。例如，Grafana 和 Graphite 主要是可视化工具。Prometheus 可以设置为提供类似于时间序列数据库的功能，但它不一定会取代对时间序列数据库的需求。在使用 Prometheus 的 Kubernetes 商店中，[InfluxDB 的](#)采用率略有增加，同时[OpenTSDB 的](#)

---

使用下降了几个百分点。CNCF 没有询问许多监控供应商的产品，例如 Nagios 和 New Relic。然而，在所有提供“其他”答案的受访者中，有 20% 提到了 New Relic。（参见本系列的第二本电子书，[Kubernetes 部署和安全模式](#)，进行更详细的分析。）

---

## Grafana和Prometheus是最多的 云原生时代的监控 Kubernetes 用户中广泛使用的监控工具



资料来源:2017 年秋季进行的云原生计算基金会调查的新堆栈分析。  
问:您目前使用什么监控工具?请选择所有适用项。英语 n=489;普通话,n=187。  
请注意,图表中仅包含使用 Kubernetes 管理容器的受访者。

© 2018 THE NEW STACK

图 4.5: Grafana 和 Prometheus 是最常用的监控工具,InfluxDB 位居第三。

根据我们在 Container Solutions 的经验,以下是我们对一些 Prometheus 替代方案的看法:

- 石墨是一个时间序列数据库,而不是一个开箱即用的监控解决方案。通常只存储聚合,而不是原始时间序列数据,并且对到达时间的期望在微服务环境中并不合适。

- 汇入数据库与 Prometheus 非常相似,但它带有一个用于缩放和集群的商业选项。它比 Prometheus 更擅长事件记录,也更复杂。

- Nagios是一种基于主机的开箱即用的监控解决方案。每个主机可以有一个或多个服务,每个服务可以执行一次检查。它没有标签或查询语言的概念。不幸的是,它是

不太适合微服务,因为它使用了一种黑盒监控形式,在大规模使用时可能会很昂贵。

·[新遗迹](#)专注于业务方面,并且可能比 Nagios 具有更好的功能。大多数功能都可以用开源等效项复制,但 New Relic 是付费产品,其功能比 Prometheus 单独提供的要多。

·[开放TSDB](#)基于 Hadoop 和 HBase,这意味着它在分布式系统上变得复杂,但如果用于监控的基础设施已经在基于 Hadoop 的系统上运行,则可以作为一个选项。与 Graphite 一样,它仅限于时间序列数据库。它不是开箱即用的监控解决方案。

·[堆栈驱动程序](#)是 Google 的日志记录和监控解决方案,集成与谷歌云。它提供与 Prometheus 类似功能集,但作为托管服务提供。它是一种付费产品 尽管谷歌确实提供了一个基本的免费层级。

## 组件和架构概述

Prometheus 生态系统由多个组件组成,其中一些组件是可选的。在其核心,服务器使用上述拉模型通过遥测端点接触服务并抓取数据。

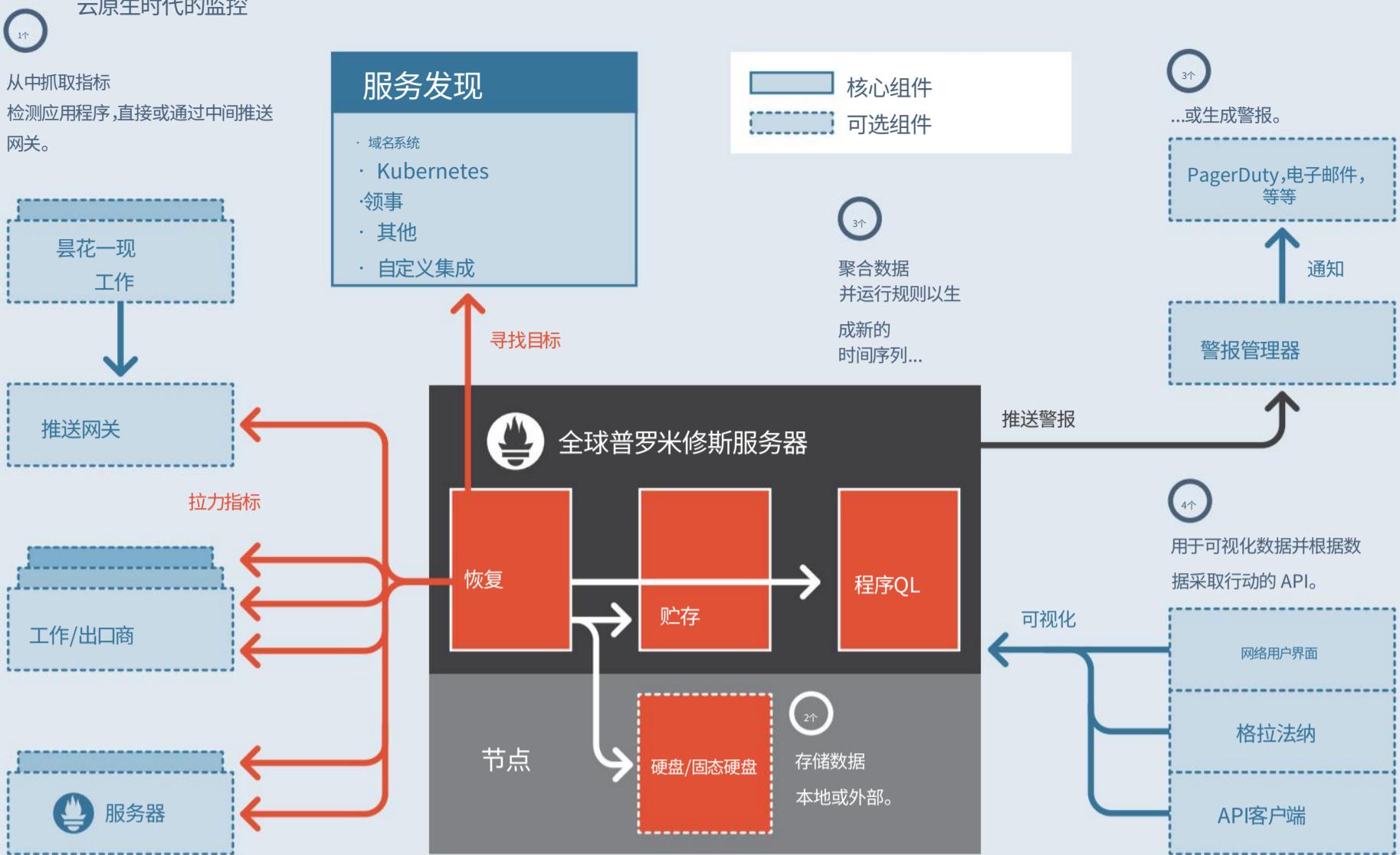
Prometheus 本身提供的基本功能包括

·直接或通过中间推送网关从已检测的应用程序中抓取指标。

·存储数据。

·聚合数据并运行规则以生成新的时间序列或  
生成警报。

# Prometheus 生态系统组件



资料来源:<https://prometheus.io/docs/introduction/overview/>

© 2018 THE NEW STACK

图 4.6: Prometheus 核心之外的组件提供补充功能来抓取、聚合和可视化数据,或生成警报。

·通过应用程序编程可视化数据并根据数据采取行动  
接口 (API) 消费者。

其他组件提供补充功能。这些包括:

·**推送网关:** 支持短期工作。这是用作作品  
让应用程序推送指标而不是被拉取指标。一些示例是来自 IoT 设备的事件、发  
送浏览器指标的前端应用程序等。

·**警报管理器:** 处理警报。

·**出口商:** 将不兼容的 Prometheus 指标转换为  
兼容格式。一些示例是 Nginx、RabbitMQ、系统指标等。

·[格拉法纳](#):分析仪表板以补充普罗米修斯表达式浏览器,这是有限的。

## Prometheus 概念

Prometheus 是一项专门为容器设计的服务,它提供了有关这个新的云原生时代的数据密集度的视角。即使是互联网规模的公司也不得不调整他们的监控工具和实践来处理这些系统生成和处理的大量数据。以这样的规模运行需要了解数据的维度、扩展数据、拥有查询语言并使其全部易于管理以防止服务器过载并允许提高可观察性和持续改进。

## 数据模型

Prometheus 将它收集的所有数据存储为时间序列,表示带有时间戳的离散测量或指标。每个时间序列都由一个指标名称和一组键值对(也称为标签)唯一标识。

通过将数据流识别为键值对,Prometheus 聚合并过滤指定的指标,同时允许进行细粒度查询。它的函数式表达语言称为 PromQL,允许用户使用 Prometheus 用户界面 (UI) 实时选择和聚合时间序列数据。其他服务(例如 Grafana)使用 Prometheus HTTP API 来获取要在仪表板中显示的数据。

其成熟、可扩展的数据模型允许用户将任意键值维度附加到每个时间序列,关联的查询语言允许您进行聚合和切片和切块。这种对多维数据收集和查询的支持是一种优势,尽管不是按请求计费等用途的最佳选择。

Prometheus 的一个常见用例是在以下情况下广播警报

某些查询通过了阈值。 SRE 可以通过定义警报规则来实现这一点 ,然后定期评估这些规则。默认情况下 ,Prometheus 每分钟处理一次这些警报 ,但这可以通过将 Prometheus 配置键更改为 global.evaluation\_interval 来调整。

每当警报表达式在给定时间点产生一个或多个向量元素时 ,Prometheus 就会通知一个名为 Alertmanager 的工具。

Alertmanager 是一个小型项目 ,具有三个主要职责 :

1. 存储、聚合和去重警报。
2. 抑制和消音警报。
3. 将警报推送和路由到外部来源。

借助 Alertmanager ,通知可以按团队、层级等分组 ,并在接收者之间分发 :Slack 、电子邮件、PagerDuty 、 WebHook 等。

Prometheus 优化如果密集使用 ,

Prometheus 服务器可能会很快过载 ,具体取决于要评估的规则数量或针对服务器运行的查询。当许多团队使用查询繁重的仪表板时 ,会发生这种情况。但是 ,有几种方法可以利用服务器上的负载。第一步是设置录制规则。

记录规则预先计算经常需要或计算量大的表达式 ,并将结果保存为一组新的时间序列 ,这对仪表板很有用。

运行电子商务网站的公司采用的一种常见设置不是运行需要大量内存和 CPU 的单个大型 Prometheus 服务器 ,而是为每个产品团队提供一个内存和 CPU 很少的 Prometheus 服务器 搜索、结账、支付等。

- 每个实例都在其中抓取自己的一组应用程序。这样的设置可以很容易地转换为分层联合架构,其中全局 Prometheus 实例用于抓取所有其他 Prometheus 实例并吸收业务使用的查询密集型仪表板的负载,而不会影响主要抓取器的性能。

## 安装 Prometheus 安装

Prometheus 及其组件非常简单。每个组件都是一个二进制文件,可以安装在任何流行的操作系统上,例如 Unix 和 Windows。安装 Prometheus 最常见的方法是使用 Docker。官方镜像可以从[Docker Hub prom/prometheus 拉取](#)。Prometheus 网站上提供了安装 Prometheus 的分步指南。

---

在云原生基础架构中,有一个名为 Operators 的概念,由[CoreOS 在 2016 年引入](#)。[Operator](#) 是一种应用程序,它能够设置、升级和恢复应用程序,以减少繁重的脚本编写或手动重复性任务(通常由站点可靠性工程师定义)以使其正常工作。在 Kubernetes 中,Operators 通过 CustomResourceDefinition 扩展 Kubernetes API,使用户可以轻松创建、配置和管理复杂的应用程序。

[普罗米修斯算子](#) 同样由 CoreOS 团队开发 使 Prometheus 配置成为 Kubernetes 原生。它管理和运行 Prometheus 和 AlertManager 集群。一个称为[Kube Prometheus](#)的补充工具,在 Prometheus Operator 之上使用,以帮助开始监控[Kubernetes](#)。它包含一系列清单 Node Exporter、Kube State Metrics、Grafana 等 以及使用单个命令部署整个堆栈的脚本。[项目存储库](#)中提供了安装 Prometheus Operator 的说明。

---

## 结论

云原生系统由小型独立服务组成,旨在通过可预测的行为最大限度地提高弹性。在公共云基础设施中运行容器并利用容器编排器来自动化一些操作例程只是迈向云原生的第一步。

系统已经发展,并带来了比几十年前更复杂的新挑战。可观察性 这意味着监控、日志记录、跟踪和警报 在克服新的云原生架构带来的挑战方面发挥着重要作用,不应被忽视。无论您最终投资的监控解决方案是什么,都需要具备云原生监控系统的特性,能够实现可观察性和可扩展性,以及标准的监控实践。

采用云原生态度是一种文化变革,涉及大量的努力和工程挑战。通过使用正确的工具和方法来应对这些挑战,您的组织将通过提高效率、加快发布周期以及通过反馈和监控持续改进来实现其业务目标。

# CI/CD与 Jenkins X 和Kubernetes



Kubernetes 使应用程序生命周期管理变得更加容易。在 Kubernetes 之前,只有持续集成和持续交付 (CI/CD) 的 CI 一半是自动化的。光盘

是用脚本、管道、元数据和配置手工组装的。Kubernetes 支持 CD 自动化,而像 Jenkins X 这样的工具使得在 Kubernetes 上部署变得简单。

“CI/CD 应该越来越像一个只为你做 CI 和 CD 的设备。这不是只有专家才能弄清楚的忍者脚本,”CloudBees 的首席架构师 James Strachan 说。“Jenkins X 并没有隐藏 Kubernetes,也没有隐藏 CI、CD 和管道,而是将其自动化。”

在此播客中,CloudBees 联合创始人兼工程经理 Michael Neale 和 Strachan 讨论了 Kubernetes 如何改变 CI/CD、发展 GitOps 等工作流,以及 CloudBees 如何努力改善开发人员在 Kubernetes 上部署应用程序的体验。[在 SoundCloud 上收听»](#)



Michael Neale 是 CloudBees 的联合创始人兼开发经理。Michael 是一位开源多语言开发人员。由于他的错误,他在 JBoss 工作了一段时间,从事 Drools 项目,然后是 Red Hat。他在澳大利亚悉尼郊外的蓝山生活和工作。



James Strachan 是 Jenkins X 及其商业版本 Kube CD 的首席架构师。他还是 Groovy 编程语言和 Apache Camel 的创建者。James 积极参与 Jenkins 社区。在加入 CloudBees 之前,他曾在 Red Hat 和 FuseSource 工作。

# 关闭

关于 Kubernetes 中持续集成和持续交付的叙述从 DevOps 开始。它包含用于更快和持续部署的新驱动力,以及对如何管理在微服务上运行的组件的更深入理解。向面向应用程序的现代架构的过渡不可避免地导致组织寻找具有管理 Kubernetes 和相关云原生服务所需的 DevOps 经验的人员。

随着团队的成长,我们现在看到更多对声明式基础架构的需求。基于 DevOps 实践构建的应用程序架构工作得更好并且运行起来摩擦更少,但最终它们只会让基础架构变得乏味。如果它很无聊,那就太好了 它很管用。然后开发人员对自己的资源有更多的控制权,应用程序的性能成为首要关注点。性能越好,最终用户越高兴,用户和开发人员之间的反馈循环也就越统一。通过这种方式,Kubernetes 等云原生技术可以提供改变游戏规则的商业价值。

出色的执行力可以带来出色的结果。但范围发生了变化。阻碍 Kubernetes 使用的历史障碍需要克服,即当来自不同背景和公司经验的人们进入一个开源项目并一起工作时出现的社会问题。Kubernetes 社区正在成熟,定义价值观已成为他们努力加强项目核心的优先事项。尽管如此,在考虑长期业务和技术目标时,确实必须考虑到 Kubernetes 的缺点。随着 Kubernetes 项目的成熟,必须信任它。会有矛盾和固执。这一切都将深入到项目中,影响测试和最终向 Kubernetes 引擎交付更新。这取决于开源社区如何解决

[关闭](#)

THE NEW STACK

委员会和特别兴趣小组结盟以推动项目向前发展。这是一个不会消失的问题。在这里，反馈循环在用户和 Kubernetes 社区之间也变得至关重要。

这带来了一些智慧，可以了解持续交付的本质以及它如何随着 CI/CD 平台的发展而变化，以及 Git 管理 Kubernetes 操作的新用途。这些努力包括关于安全性、身份、服务网格和无服务器方法的讨论，以进一步抽象的方式使用资源。

只有当抽象变得功能失调时，真正的变化才会到来。例如，在整个构建和部署周期中必须有一个持续的反馈循环，以便更好地了解时间序列信息的比较如何显示异常。新兴模式成为寻找问题线索的最佳方式。反馈循环也是开发人员体验的关键，这对于他们以最佳方式构建自己的图像至关重要。在编辑这本电子书时对我们来说显而易见的是，这个反馈循环必须存在于用户和 Kubernetes 社区本身之间，以及在其之上构建和部署其应用程序的组织内部。

The New Stack 的下一个主题是我们开发电子书的新方法。今年寻找有关微服务和无服务器的书籍，其中包含相应的播客、深入的帖子和世界各地提供煎饼的活动。

谢谢，很快再见到你。

亚历克斯·威廉姆斯  
创始人、总编辑  
新堆栈

# 披露

除了我们的电子书赞助商之外,以下公司也是 The New Stack 的赞助商:

Alcide, AppDynamics, Aspen Mesh, Blue Medora, Buoyant, CA Technologies, Chef, CircleCI, Cloud Foundry Foundation, {code}, InfluxData, LaunchDarkly, MemSQL, Mesosphere, Microsof, Navops, New Relic, OpenStack Foundation, PagerDuty, Pivotal, Portworx、Pulumi、Puppet、Raygun、Red Hat、Rollbar、SaltStack、Stackery、StackRox、Linux 基金会、Tigera、Twistlock、Univa、VMware、Wercker 和 WSO2。

