THENEWSTACK

KUBERNETES生态环境

THE STATE OF THE KUBERNETES ECOSYSTEM

The New Stack
The State of the Kubernetes Ecosystem
Alex Williams, Founder & Editor-in-Chief
Core Team:
Bailey Math, AV Engineer
Benjamin Ball, Marketing Director
Gabriel H. Dinh, Executive Producer
Judy Williams, Copy Editor
Kiran Oliver, Associate Podcast Producer
Krishnan Subramanian, Technical Editor
Lawrence Hecht, Research Director
Scott M. Fulton III, Editor & Producer

目录

前言

在我看来整个系统在物理意义上最基本的概念是，不仅要包括复杂的有机体， 而且包括形成我们称之为环境的复杂的整体物理因素...虽然有机体可能是我们的主要兴趣，但当我们试图从根本上考虑这个问题时，我们不能将它们与形成它们的一个特殊环境的物理系统分开。 从生态学家的角度来看，正是这样形成的系统是地球上基本的自然基础单位。 这些都是生态系统。

-1935 年，英国生态学家，亚瑟·乔治·坦斯利爵士

《The Use and Abuse of Vegetational Concepts and Terms》

我们使用的术语“基础设施”越来越多地被定义成信息技术支持系统。 无论我们的应用程序做什么，是为我们的客户创造价值，或为自己创造收入，现在都在用 IT 基础设施来支撑。所有的东西都被封装起来。它们是技术的一部分，当它正常工作，或者表现得像我们期望的时候，我们不会非要花很多时间注意它，也不会在晚上闲着没事的时候讨论它们。

在我们现在工作依赖的一堆技术栈中，有越来越多的属于底层技术。 使用现代超级服务器将计算，存储和内存资源转换集中到巨大的资源池中，异构技术的网络将这些资源池组合成一整套物理基础设施。

而在现代分布式计算网络中，即使云计算只能部分在云中，使应用程序可部署，可管理和可扩展的支持结构已成为我们的虚拟基础架构。是的，它同样被封装起来，只有它的技术栈顶部接口是开放的。

这本书的内容是关于虚拟基础设施的一种非常新的实现方法 - 由于 Google 需要在大规模网络上运行云原生应用程序而出现。 Kubernetes 并不是一个真正的操作系统，我们曾经考虑 Windows Server 或许多企业级 Linux 的方案，但是在越来越多的团队中，它已

经在运营商和开发人员的观念中取代了操作系统。 它的目标是作为在容器中运行的应用程序（我们以前称为“Linux 容器”，尽管其形式和格式已经扩展到 Linux 之外）的资源提供者，并确保这些应用程序的性能达到指定的服务级别。 所以 Kubernetes 在这方面确实取代了操作系统。

这本书的标题是 Kubernetes 的生态系统。 这是一个不寻常的定义。 第一个软件生态系统是由程序员，培训师和经销商组成，他们互利互益的一起进行工作。从本质上说，这也是 Kubernetes 生态系统正在尝试的事情。 它构建了一个环境，让其参与者利用开源软件开发过程及其开放的开源精神，这创建了一个经济体系，使得参与者可以从彼此的依存中获得收益。

很难这样认为——Kubernetes 实际上是或者应该是这个生态系统的中心。就如 Linux 不再是 Linux 本身所产生的这个生态系统的焦点。分布式计算环境由几十个组件组成，其中一些是开源的，一些是商业的，其中也有许多两者都是。Kubernetes 可能会产生一个这样的组件协同工作的愿景，但即使如此，它也只是一个组件。在一个不断发展出创新想法（远远超越对专利被侵权的担心）的市场上，这个组件也可能被新技术所取代。

本书的目的是为您提供理解它的生态环境的简洁方法，为您呈现分布式系统的经济和技术环境以及 Kubernetes 在该环境中的最清晰的实时简介。 我们在六个赞助商的帮助和指导下向您提供本书，我们非常感激：

- Cloud Native Computing Foundation（CNCF），一个 Linux 基金会项目; Kubernetes 开源项目和许多特别兴趣小组的的管理者; 还是 Fluentd，linkerd，Prometheus，OpenTracing，gRPC，CoreDNS，containerd，rkt 和 CNI 的管理者。
- Codeship，一个持续集成平台提供商，提供 Docker 和 Kubernetes 的集成。
- CoreOS，构造商业平台的开发商，将上游的 Kubernetes 作为其调度引擎，提供企业级的功能特性。
- Powered by Kubernetes，在 Google 云平台上的 Google 容器引擎托管环境，用于部署容器化应用程序。

- Red Hat，OpenShift 云原生应用平台的开发商，它使用 Kubernetes 作为其调度引擎。
- Twistlock，提供一个自动化容器安全平台，目的在与 Kubernetes 进行集成。

本书的部分内容来自以下公司软件工程师的贡献：

- Kenzan，一家为企业定制 IT 部署和管理解决方案的专业服务公司。

我们很高兴您选择先阅读这部分内容，它属于关于 Kubernetes 的三卷系列丛书，希望能帮助您接触到在企业部署，管理和扩展企业应用程序方面的最新变化。

赞助商

我们非常感谢基金会赞助支持我们的电子书：

同时感谢我们的赞助商：

Kubernetes 及容器编排的总体介绍

作者：**JANAKIRAM MSV** *和* **KRISHNAN SUBRAMANIAN**

仅仅在几年前，无论是旧的 `cgroup` 还是流行的 Docker 或者 CNCF rkt，最可能发挥 Linux 容器功能的地方还是在一些开发人员笔记本上的隔离的沙盒环境中。通常它还只是一个实验环境，最多是个开发平台，根本就不是数据中心的一部分。

而今天，容器已经成为在生产环境中部署新的、基于云的应用程序的实际选择。在 3 到 4 年的时间内，现代应用程序部署的方式已经从虚拟的基于机器的云平台转变为有规模的有组织的容器群。

在本章中，我们将讨论在容器生态系统中的调度编排器（包括 Kubernetes），介绍市场上的一些主要的编制工具，并描述它们的各种优点。

Kubernetes 的来历

容器化的想法并不新鲜。某种形式的虚拟隔离，无论是出于安全性还是多租户的目的，自上世纪 70 年代以来就一直被用于数据中心里。

从 `chroot` 系统调用的出现开始，首先是在 Unix，后来在 BSD 中，容器化的思想已经成为企业 IT 的通行做法的一部分。从 FreeBSD 的 Jails 到 Solaris 的 zones 到 Warden 到 LXC，容器一直在不断发展，所有的过程都越来越接近成为采用的主流方案。

在容器在开发人员中流行之前，谷歌在 Linux 容器中运行它的一些核心 web 服务。2014 年，Kubernetes 创始人之一的 Joe Beda 在 GlueCon 的一次演讲中声称，谷歌在一周内启动了超过 20 亿个容器。谷歌管理容器的能力的秘密在于它的内部数据中心管理工具：Borg。

谷歌将 Borg 改造成一个通用的容器编排调度器，于 2014 年将其发布到开源社区，并将其捐赠给 2015 年 Linux 基金会的云计算基础（CNCF）项目。Red Hat、CoreOS、微

软、中兴、Mirantis、华为、富士通、Weaveworks、IBM、Engine Yard 和 SOFTICOM 都是该项目的主要贡献者。

在 2013 年 Docker 出现以后，容器的采用率发生了爆炸式增长，成为了那些想要实现 IT 基础设施现代化的企业的焦点。这种爆发的趋势有四个主要原因：

- **封装：**Docker 解决了容器的用户体验问题，使它们更容易打包应用程序。在 Docker 之前，处理容器是非常困难的（Warden 是个例外，它是由云计算平台抽象出来的）。
- **分发：**自从云计算的出现以来，现代应用程序体系结构逐渐变得更加分散。无论是初创公司还是大型组织机构，都受到 DevOps 这种新方法和合作精神的启发。近年来它们都已将注意力转向了微服务架构。容器比迄今为止的其他各种体系结构设计得更模块化，更适合于支持微服务。
- **可移植性：**开发人员喜欢在任何地方构建应用程序并运行它——将代码从他们的笔记本电脑推向生产环境，并期望它们在没有重大修改的情况下以完全相同的方式工作。随着 Docker 周围广泛积累了更多的工具，其功能的广度和深度更有助于开发人员采用容器。
- **速度：**尽管在 Docker 之前已经存在了容器化的形式，但它们在最初的实现中却启动非常缓慢——在 LXC 的情况下，经常会花费几分钟，而使用 Docker 则把时间缩短为几秒钟。

自 2015 年 7 月发布以来，Kubernetes 已经成为最受欢迎的容器编排引擎。四大公共云服务提供商中有三家——谷歌、IBM 和微软——都提供了一个基于 Kubernetes 的服务平台(CaaS)平台。而第四名亚马逊，则刚刚加入了 CNCF，并有自己的计划来支持这个平台。尽管 Amazon 以 EC2 容器服务的形式拥有自己的托管容器平台，但是 AWS 以运行了最多 Kubernetes 集群而闻名。大型企业，如教育出版商 Pearson，飞利浦的物联网设备部门，TicketMaster，eBay 和纽约时报等公司都正在生产环境中运用了 Kubernetes 技术。

什么是编排？

虽然容器有助于提高开发人员的生产力，但编排工具为寻求优化 DevOps 和 Ops 投资的团队提供了许多好处。容器编制的一些好处包括:

- 高效的资源管理。
- 服务的无缝扩展。
- 高可用性。
- 在规模上操作开销较低。
- 一个声明式模型(对于大多数编制工具)减少了对更加自治化管理的阻碍。
- 操作式基础设施即服务(IaaS)，但又具备类似平台即服务(PaaS) 可管理的功能。

容器解决了开发人员的生产力问题，使 DevOps 工作流变得可以无缝连接。开发人员可以创建 Docker 镜像、运行 Docker 容器并在该容器中开发代码。然而，对开发人员生产力的提升并不能自动转化为生产环境中的效率。

从开发人员的笔记本电脑的本地环境中分离出的生产环境比原有的规模要大得多。无论您运行成规模的多级应用程序或基于微服务的应用程序，管理大量的容器和支持它们的节点集群并非易事。编制是实现规模所需的组件，因为规模需要自动化。

云计算的分布式特性带来了我们如何感知虚拟机基础结构的典型转换。“牛与宠物”的观点——把一个容器当做牲畜类似的使用单元，而不是最喜欢的动物——帮助重塑了人们对基础设施根本的观念。把这个观念付诸实践，容器在规模上扩展和完善了收缩和资源可用性的定义。

典型的容器编排平台的基本特征包括:

- 调度
- 资源管理
- 服务发现
- 健康检查
- 自动伸缩
- 更新和升级

目前容器编排市场由开源软件主导。在本文出版的时候，Kubernetes 依然是领导者。但在我们深入研究 Kubernetes 之前，我们应该花一点时间将其与市场上其他主要的编制工具进行比较。

Docker Swarm

Docker 公司是负责最受欢迎的容器产品的公司，它提供 Docker Swarm 作为容器的编制工具。在 Docker 中，所有容器都是标准化的。在操作系统级别的每个容器的执行由 runc 处理，这是遵循 Open Container Initiative（OCI）规范的实现。Docker 与另一个开源组件“containerd”一起工作，以管理由 runc 在特定主机上执行的容器的生命周期。在一个主机操作系统上，Docker、containerd 和 runc 执行程序负责处理容器操作。

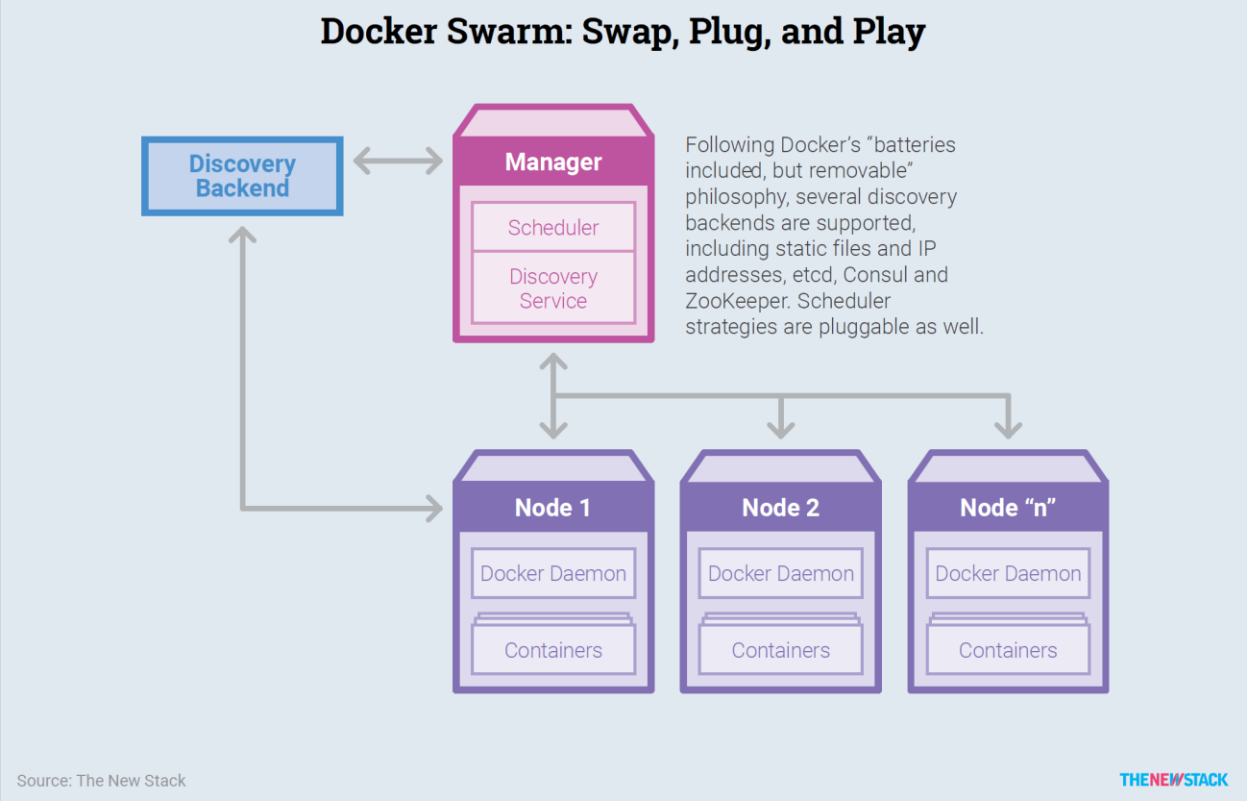


图 1.1：*典型的 Docker Swarm 主节点与工作节点之间的配置关系。*

简单地说，Swarm——由 Docker Swarm 编排——是一组运行 Docker 的节点。如图 1.1 所示。Swarm 中的一个节点充当其他节点的管理器，并包括调度程序和服务发现组件的容器。

Docker 的理念要求在容器级别进行标准化，并使用 Docker 应用程序编程接口（API）来处理业务流程，包括底层基础设施的使用。Docker Swarm 使用现有的 Docker API 和网络框架，而不扩展它们，结合 Docker Compose 工具而更适合构建多容器应用程序。它使开发人员和操作人员更容易将一个应用程序从 5 个或 6 个容器扩展到数百个。

注意：Docker Swarm 使用 Docker API，使它可以轻松地适应现有的容器环境。采用 Docker Swarm 可能意味着要在 Docker 上孤注一掷，目前，Swarm 的调度器选项是有限的。

Apache Mesos

Apache Mesos 是一个开放源码的集群管理器，它的出现早于 Docker Swarm 和 Kubernetes。再加上 Marathon，一个用于基于容器的应用程序的编排框架，它为 Docker Swarm 和 Kubernetes 提供了一个有效的替代方案。Mesos 也可以使用其他框架来同时支持容器化和非容器化的工作负载。

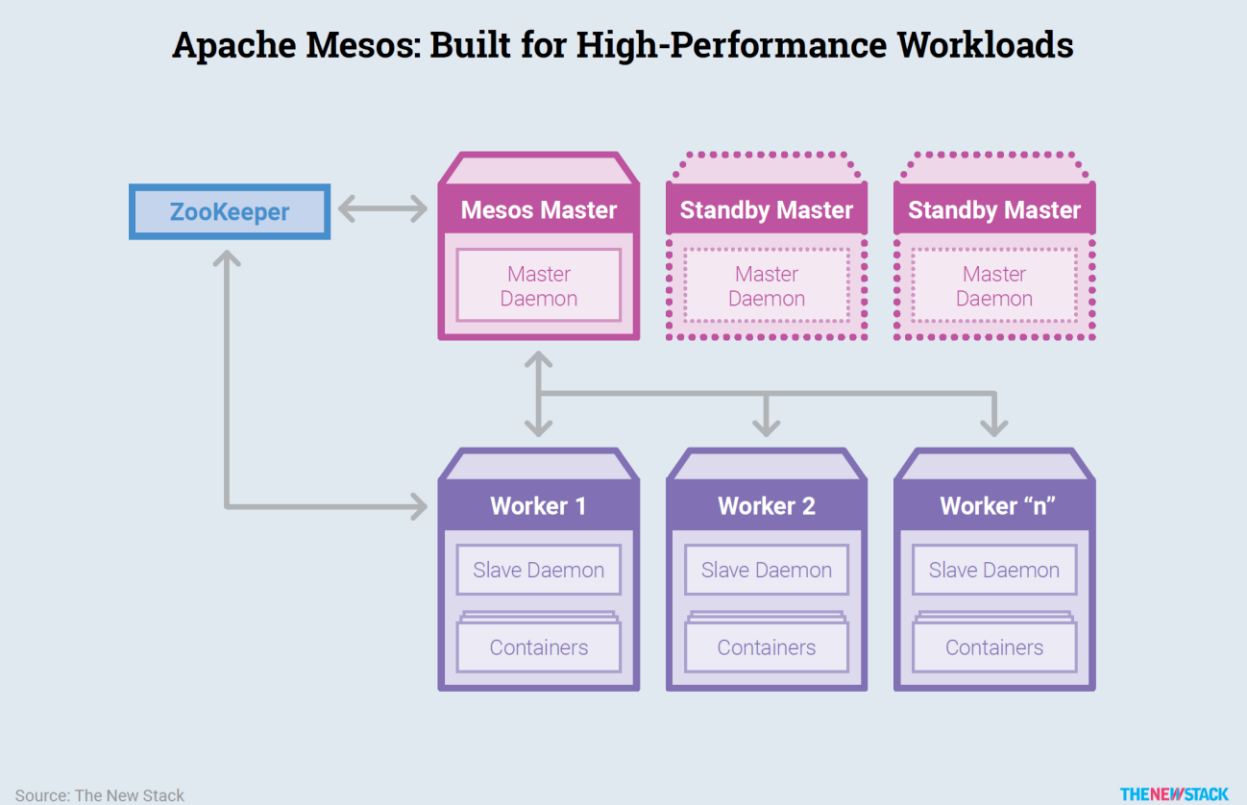


图 1.2：*Apache Mesos，构建多种多样、高性能的工作负载。*

Mesos 的平台，如图 1.2 所示，显示了节点之间的主/从关系。在这个方案中，分布式应用程序通过一个称为 ZooKeeper 的组件在集群之间进行协调。这个 ZooKeeper 的工作就是为一个集群选择控制节点，或者分配备用控制节点，并用代理逐步分配给每个其它节点。这些代理建立了主/从关系。在 Master 中，主守护进程建立了所谓的“框架”，它像桥梁一样，在主节点和工作节点之间延伸。运行在这个框架上的调度器确定了哪些工作节点可以接受资源，而守护进程则设置了要共享的资源。这是一个复杂的方案，但它的优点是能够适应多种类型和分布式负载的标准，而不仅仅是容器。

与 Docker Swarm 不同，Mesos 和 Marathon 都有自己的 API，这使得它们比其他编配工具更复杂。 然而，Mesos 在支持 Docker 容器和虚拟机(如 VMware vSphere 和 KVM)方面更加灵活。Mesos 还支持支持大数据和高性能工作负载框架。

注意: Apache Mesos 是混合环境的完美编配工具，它包含容器和非容器的工作负载。尽管 Apache Mesos 是稳定的，但许多人认为它为容器用户提供了更陡峭的学习曲线。

Kubernetes

Kubernetes 最初是由谷歌发起的开源项目，现在是云计算基础（CNCF）的一部分，它可以用很低的运行开销对容器进行互联网级规模的无缝管理。

Kubernetes 并不固定要求容器的格式，并且使用它自己的 API 和命令行接口（CLI）来进行容器编排。

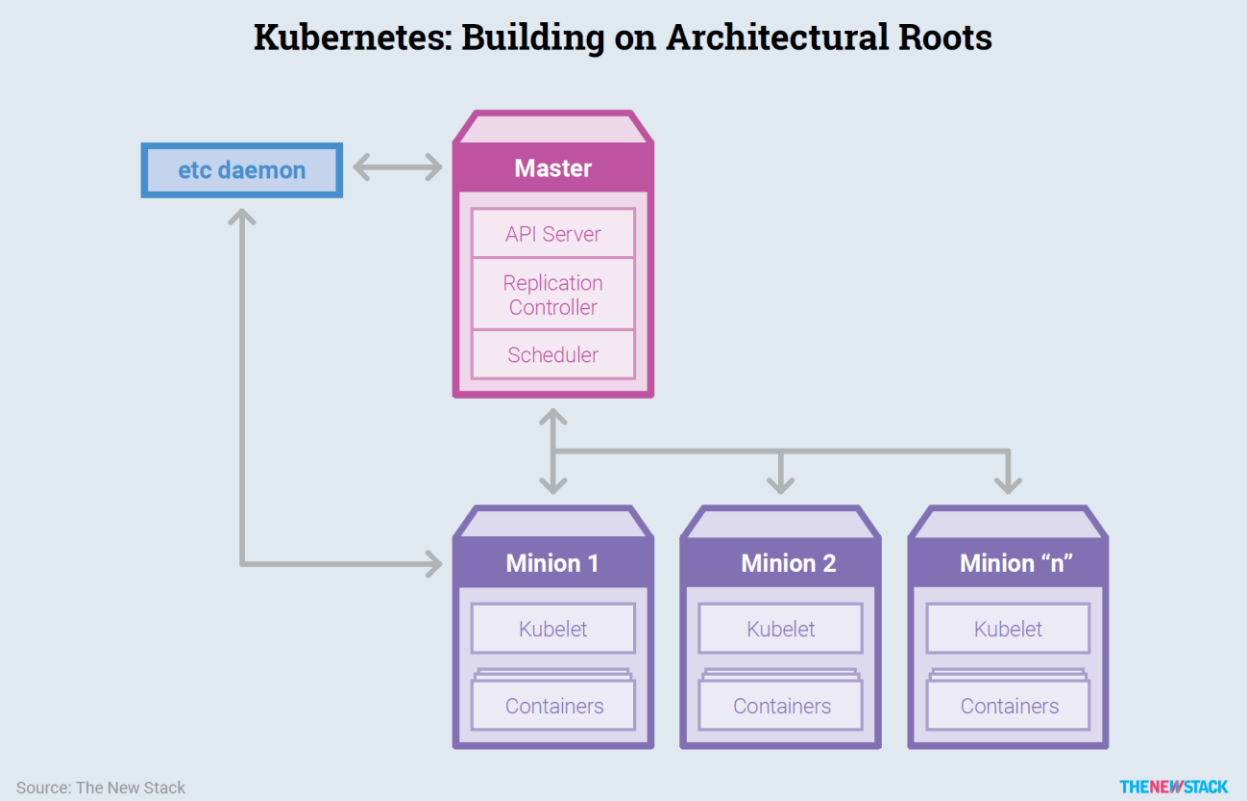


图 1.3：*Kubernetes 的控制器与节点之间的关系，以前也称为“minions”。*

它支持多种容器格式，不只是包括 Docker 的，也支持 rkt——最初由 CoreOS 创建，现在是 CNCF 托管的项目。该系统也是高度模块化且易于定制的，允许用户选择任何调度器、网络系统、存储系统和监控工具集。它以单个集群开始，并可以无缝地扩展到大规模的 Web 应用。

我们之前提到的一个编排器的六个主要特征体现在 Kubernetes 中的以下方面中：

- **调度：**Kubernetes 调度程序确保在任何时候都可以满足对基础设施上的资源的需求。
- **资源管理：**在 Kubernetes 的环境中，资源是编排器可以实例化和管理的逻辑结构，例如服务或应用程序部署。
- **服务发现：**Kubernetes 支持共享系统的服务，可以通过名称来发现。这样，包含服务的 pods 可以在整个物理基础结构中分布运行，而不必保留网络服务来定位它们。
- **健康检查：**Kubernetes 利用称为“活性探针”和“就绪探针”的功能，给编排器提供应用程序状态的周期性的指示。
- **自动伸缩：**使用 Kubernetes，当一个 pod 的指定 CPU 资源未被充分利用时，pod 自动伸缩器能自动生成更多的副本。
- **更新/升级：**一个自动化的滚动升级系统使每个 Kubernetes 的部署保持更新和稳定。

注意：Kubernetes 是由一个非常活跃的社区建立的。它为用户提供了更多的选择来扩展编排引擎以满足他们的需求。由于它使用了自己的 API，所以更熟悉 Docker 的用户将会遇到一些学习难点。

Kubernetes 体系结构

现在一个打包为一组容器的应用程序需要一个足够健壮的基础设施来处理集群的需求和动态编排的压力。这样的基础设施应该为跨主机的调度、监视、升级和重新定位容器提供支撑。它必须将底层的计算、存储和网络原语作为资源池来处理。每个容器的工作负载应该能够充分利用提供给它的资源，包括 CPU 内核、存储单元和网络。

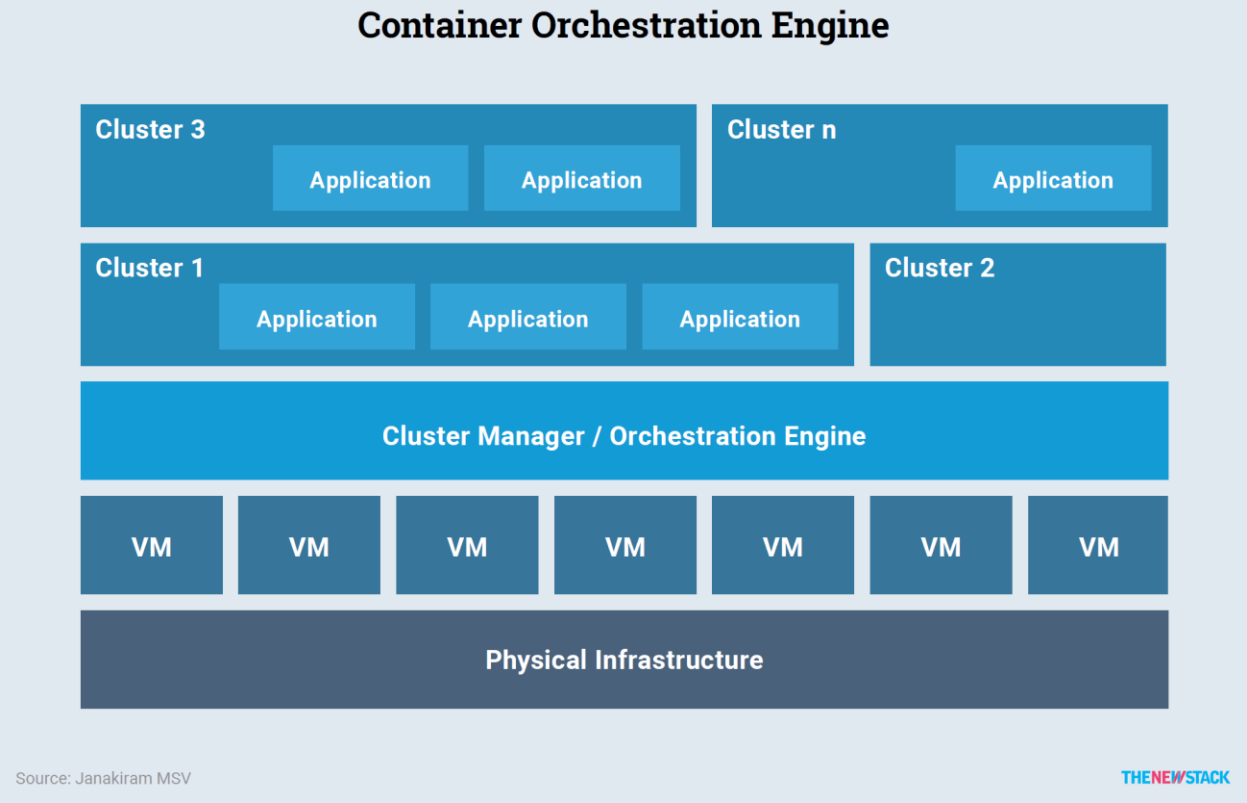


图 1.4：*从容器编制引擎的角度来看的系统资源层。*

Kubernetes 是一个开放源码的集群管理器，它封装了底层的物理基础设施，使其更容易在很大规模上运行容器应用程序。一个应用程序，通过 Kubernetes 的整个生命周期管理，是由一组 pod 成集合作并协调成一个单元工作的。一个高效的集群管理器层让 Kubernetes 能够有效地将这个应用程序与它的支持基础结构分离开来，如图 1.4 所示。一旦 Kubernetes 基础结构被完全配置好，DevOps 团队就可以专注于管理已部署的工作负载，而不是管理底层资源池。

Kubernetes API 可以用来创建提供微服务的关键构建块的组件。这些组件是自主的，这意味着它们独立于其他组件存在。它们被设计为松散耦合、可扩展和适用于各种工作负载。API 为内部组件提供了这种扩展性，以及在 Kubernetes 上运行扩展组件和容器。

Pod

Pod 是 Kubernetes 的工作负载管理的核心单元，它充当了共享相同运行环境和资源的容器的逻辑边界。将相关的容器分组到 pod 中，就可以弥补由于容器化取代了第一代虚拟化所带来的配置挑战，从而使其能够一起运行多个相互依赖的进程。

每个 pod 是一个或多个使用远程过程调用（RPC）进行通信的容器的集合，共享存储和网络堆栈。适用的场景是一些容器需要耦合和共存的应用——例如，一个 web 服务器容器和一个缓存容器。它们很容易被包装在一个 pod 里。一个 pod 可以手动伸缩，也可以通过称为水平 pod 自动伸缩（HPA）的特性来定义。通过这种方法，pod 内包装的容器数量按比例增加。

Pod 支持开发和部署之间的功能分离。当开发人员专注于他们的代码时，操作人员可以将注意力集中在更广阔的视角上，考虑相关容器可能被组合成一个功能单元。这样有助于达到可移植性的最佳数量，因为一个 pod 只是多个容器镜像像管理的一个清单。

服务

Kubernetes 的服务模型依赖于微服务的最基本、最重要的方面：发现机制。

一个单独的 pod 或一个副本集（稍后解释）可以通过服务暴露给内部或外部客户，该服务将一组带有特定标准的 pod 联起来。任何其标签匹配选择器的 pod 将自动被服务发现。该体系结构提供了一种灵活的、松散耦合的服务发现机制。

当一个 pod 被创建时，它被分配一个只有在集群内访问的 IP 地址。但不能保证 pod 的 IP 地址在整个生命周期中保持不变。Kubernetes 可以在运行时迁移或重新实例化 pod，从而为 pod 提供一个新的 IP 地址。

为了解决这种不确定性，服务能确保路由到集群中适当的 pod，而不考虑它的调度节点，每个服务公开一个 IP 地址，也可能公开一个 DNS 地址，这两个入口都不会改变。需要与一组 pod 进行通信的内部或外部用户将使用该服务的 IP 地址，或者更常见的 DNS 地址。这样，服务就充当了连接 pod 与其他 pod 的粘合剂的角色。

服务发现

Kubernetes 中的任何 API 对象，包括一个节点或一个 pod，都可能具有与其相关的键值对--用于标识和组合共享一个公共特征或属性的对象的附加的元数据。Kubernetes 引用这些键值对作为标签。

选择器是用来查询与标签值匹配的 Kubernetes 对象的一种标准。这种强大的技术支持对象的松散耦合。可以生成新的对象，其标签与选择器的值相匹配。标签和选择器形成了 Kubernetes 的主要分组机制，用于标识操作应用的组件。

一个副本集依赖于标签和选择器来决定哪个 pod 将参与缩放操作。在运行时，可以通过副本集对 pod 进行缩放，确保每次部署都运行所需数量的 pod。每个副本集始终维护一个预定义的 pod 集。

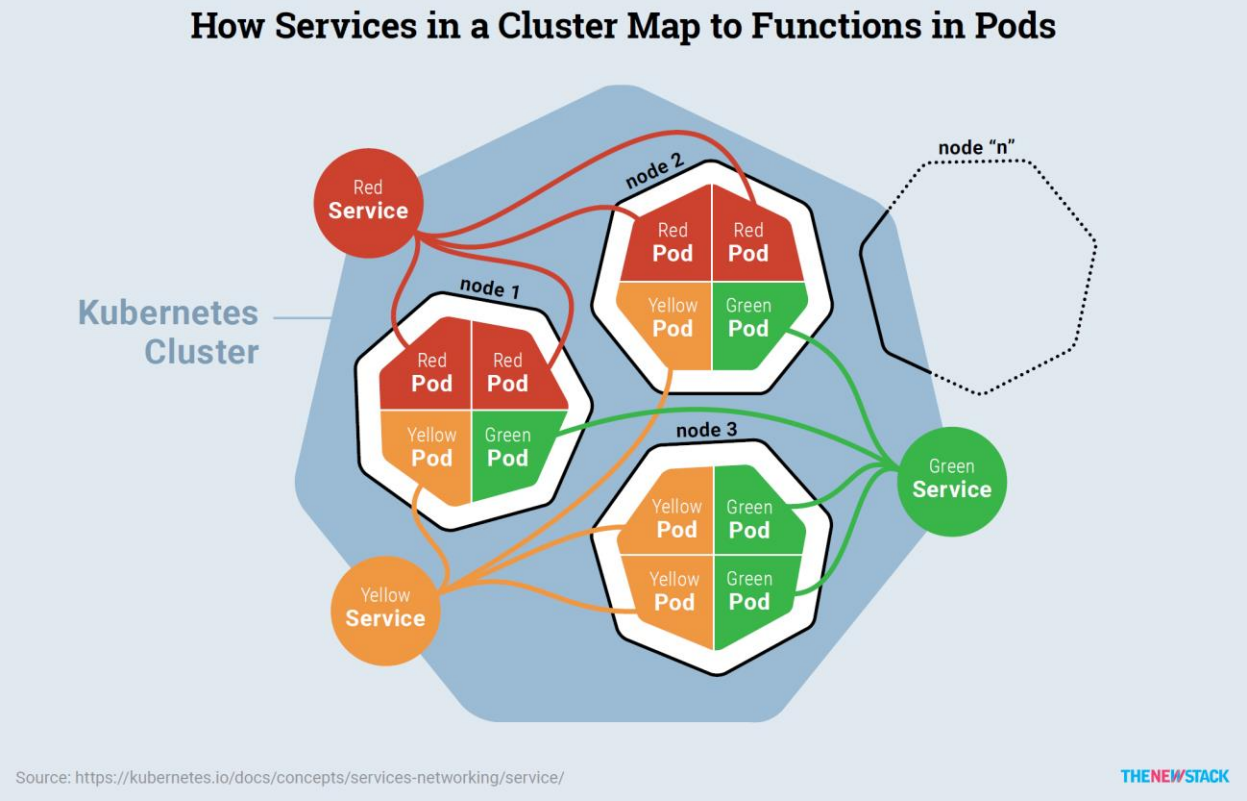


图 1.5：*Kubernetes 集群关注的是 pod，而它们以服务的方式提供给外面的世界。*

任何标签与服务定义的选择器相匹配的 pod 将可以被终端公开访问。当一个缩放操作由一个副本集发起时，由该操作创建的新 pod 将立即开始接收流量。然后，服务通过在匹配的 pod 中路由流量提供基本的负载平衡。

图 1.5 描述了服务发现如何在 Kubernetes 集群中工作。这里有三种类型的 pod，由红、绿、黄相间的盒子代表。一个副本控制器已经扩展了这些 pod，以在所有可用节点上运行实例。每个类的 pod 都通过一个由彩色圆圈表示的服务向客户发布。假设每个 pod 都有 `color=value` 的标签，它的相关服务将有一个匹配它的选择器。

当客户到达红色服务时，请求被路由到任何与标签 `color=red` 匹配的 pod 中。如果一个新的红色 pod 被缩放调度器产生，那么它将立即被服务发现，因为它的具有匹配的标签和选择器。

可以将服务配置为将 pod 公开给内部和外部的使用者。一个暴露给内部的服务可以通过一个 `ClusterIP` 地址获得，这个地址只能在集群中路由。不需要暴露给外部的数据库 pod 和其他敏感资源，可以配置为使用 `ClusterIP`。当一个服务需要被外部访问时，它可能通过每个节点的特定端口来发布，这被称为 `NodePort`。在公共云环境中，Kubernetes 可以提供自动配置的负载平衡器，以便将流量路由到相应的节点。

Master 控制节点

与大多数现代分布式计算平台一样，Kubernetes 利用了一个主/从架构。 如图 1.6 所示，master 封装了从 API 中运行应用程序的节点，这些节点是与编排调度器进行通信的。

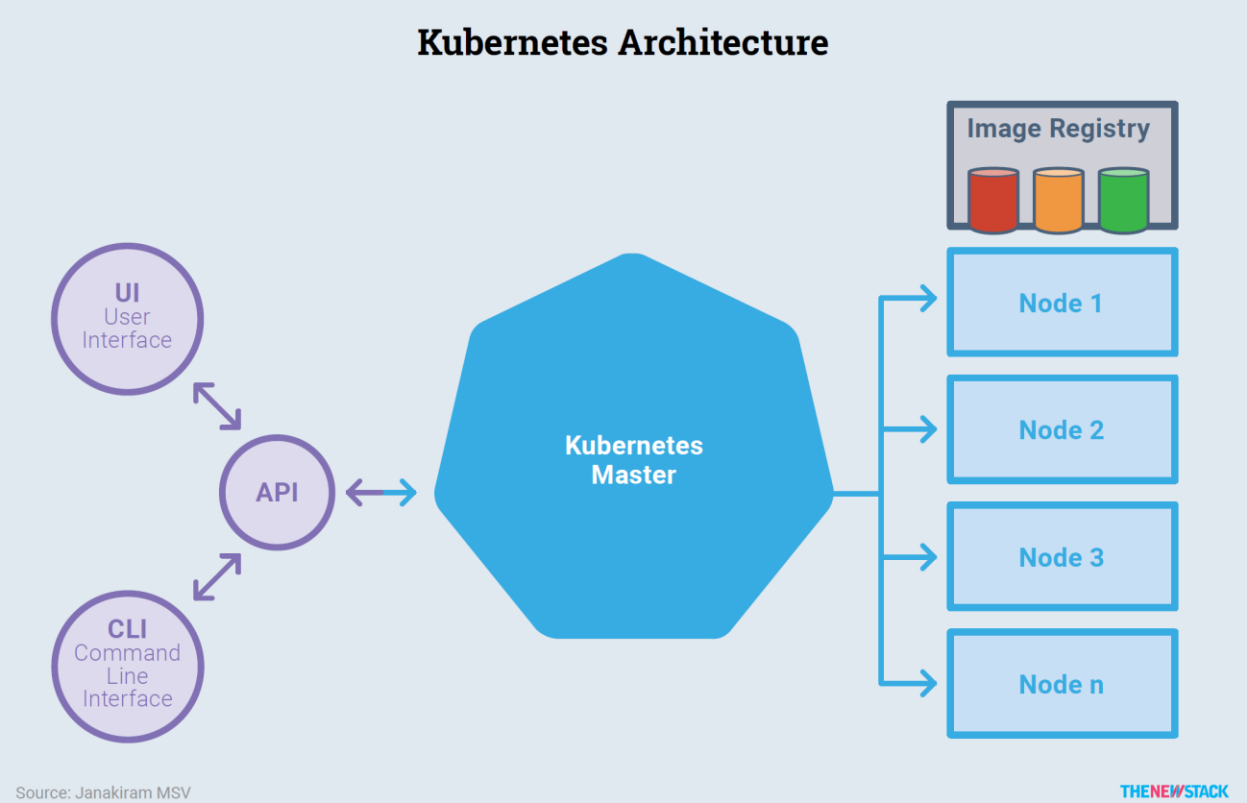


图 1.6：*Master 在 Kubernetes 体系结构中的位置。*

Master 负责提供 Kubernetes API，调度工作负载的部署，管理集群，并控制整个系统的通信。如图 1.6 所示，Master 监视每个节点上运行的容器以及所有注册节点的健康状态。容器镜像作为可部署的构件，必须通过私有或公共的镜像（image）仓库让 Kubernetes 集群可以访问使用。负责调度和运行应用程序的节点从镜像仓库中获得应用服务的镜像。

如图 1.7 所示，Kubernetes Master 运行以下组件，它们组成了控制面板：

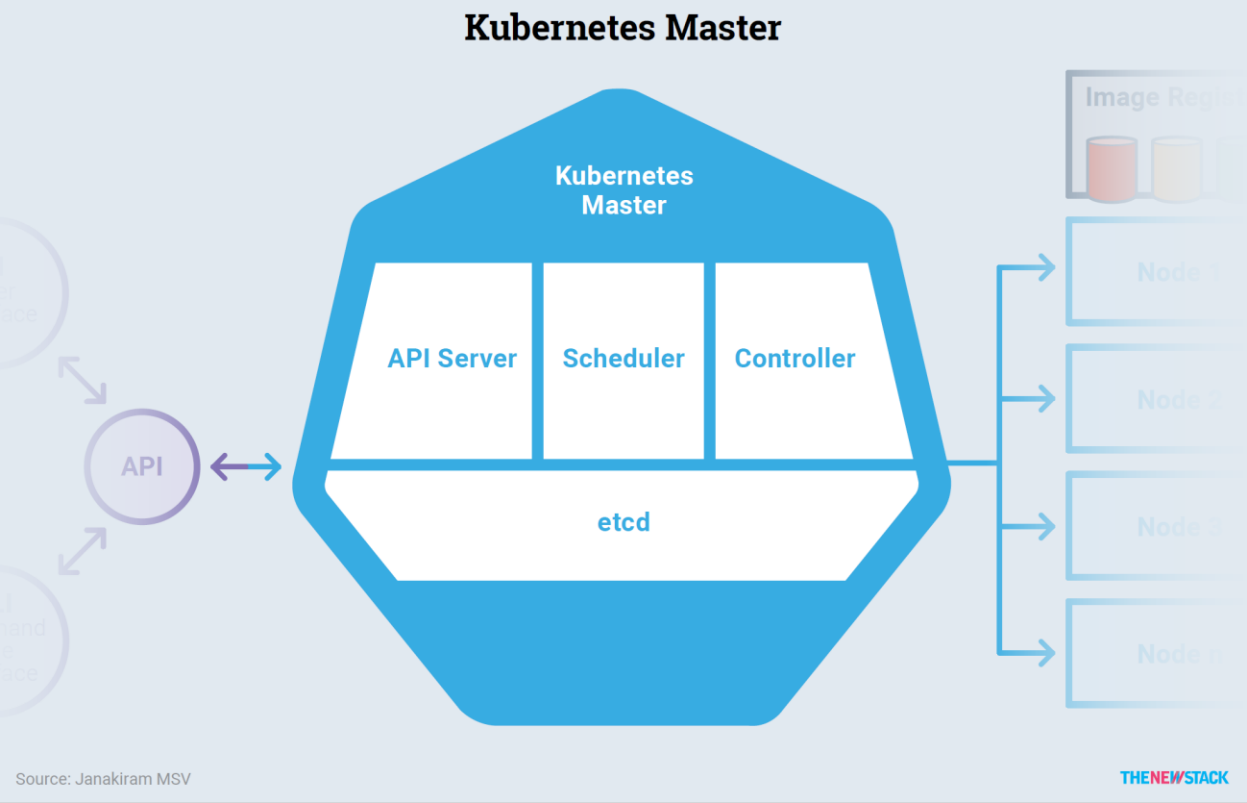


图 1.7：*Kubernetes Master 的组件。*

etcd

etcd 是由 CoreOS 开发的，它是一个持久的、轻量级的、分布式的、键值的数据存储服务，它维护集群的配置数据。它表示在任何时间点上集群的总体状态，作为唯一的事实数据来源。其他各种组件和服务监视对 etcd 存储进行更改，以保持应用程序达到期望的状态。该状态由声明式策略（实际上是一个声明该应用程序的最佳环境的文件）定义，因此编排调度器可以通过一系列工作来获得该环境。该策略定义了编排调度器如何处理应用程序的各种属性，例如实例数量、存储需求和资源分配。

API Server

API 服务器通过 HTTP 协议以 JSON 方式发布 Kubernetes API，为编排调度器的内部和外部端点提供 REST 接口。CLI、web UI 或其他工具可能向 API 服务器发出请求。服务器处理并验证请求，然后更新 etcd 中 API 对象的状态。这使得客户终端能够跨工作节点来配置工作负载和容器。

Scheduler 调度器

调度器基于对资源可用性评估来为每个 pod 选择运行的节点，然后跟踪资源利用率，以确保 pod 不会超过它分配的限额。它维护和跟踪资源需求、资源可用性以及各种其他用户提供的约束和策略指标；例如，服务质量(QoS)、亲和/反亲和性需求和局部性数据。操作团队可以声明式的定义资源模型。调度器将这些声明解释为为每个工作负载提供和分配正确资源集的指令。

Controller 控制器

控制器是 Kubernetes 体系结构的一部分，是 Master 的一部分。控制器的职责是确保集群始终保持节点和 pod 的期望状态。通过我们所说的期望状态，达到由 pods 的 YAML 配置文件所声明和请求使用的资源和系统的当前需求和约束的平衡。

控制器通过不断地监视集群的健康状况和部署在集群上的工作负载来维护节点和 pod 的稳定状态。例如，当一个节点变得不健康时，在该节点上运行的那些 pod 可能无法访问。在这种情况下，控制器的任务是在不同的节点中调度相同数量的新 pod 提供服务。这个活动确保集群在任何时间点都保持预期状态。

Kubernetes 控制器在生产中运行容器化的工作负载时起着至关重要的作用，这使得一个团队可以部署和运行容器化的应用程序，这些应用程序复杂程度远远超出了典型的无状态和扩展的场景。控制器管理者负责管理核心 Kubernetes 控制器：

- `ReplicationController`（`ReplicaSet`）维护集群中特定部署的 pod 数目。它保证任何时候都能有指定数量的 pod 在系统的节点上运行。
- `StatefulSet` 有状态集类似于副本集，是对于需要持久性和定义良好的标识符的 pod 的定义。
- `DaemonSet` 确保包在一个 pod 里的一个或多个容器被运行在集群的每个节点中。这是一种特殊类型的控制器，它强制一个 pod 在每个节点上运行。作为 `DaemonSet` 的一部分运行中的 pod 的数量与节点的数量成正比。
- `Job` 和 `cron job` 控制器处理后台进程和批处理进程。

这些控制器与 API 服务器通信，以创建、更新和删除它们管理的资源，如 pod 和服务节点。

关键型应用程序需要更高级别的资源可用性。通过使用一个副本集，Kubernetes 确保了预定义的 pod 数量一直在运行。但这些 pod 是无状态的，短暂的。由于以下原因，很难运行有状态的工作负载，例如数据库集群或大数据栈：

- 每个 pod 在运行时分配一个任意名称。
- 在任何可用节点上都可以安排一个 pod，除非使用关联规则，在这种情况下，pod 只能在有特定标签或在规则中指定的标签的节点上调度。
- 在任何时间点都可以重新启动和重新定位。
- 一个 pod 可能永远不会被它的名称或 IP 地址直接引用。

从 1.5 版本开始，Kubernetes 引入了有状态集(由 `StatefulSets` 表示)的概念，用于运行高可用的工作负载。参与有状态集的有状态 pod 具有以下属性：

- 一个稳定的主机名，它将永远由 DNS 解析。
- 一个序数索引号，用来表示副本集中的 pod 的顺序位置。
- 与主机名和序号相关联的稳定存储。

稳定的主机名及其序号索引号使一个 pod 能够以可预测和一致的方式与另一个 pod 进行通信。这是无状态 pod 和有状态 pod 之间的根本区别。

Node

节点是 Kubernetes 集群的工作站，负责运行容器化的工作负载；日志、监视和服务发现；和可选的附加组件。它的目的是向应用程序公开计算、网络和存储资源。每个节点包括一个容器运行时，例如 Docker 或 rkt，以及与控制器 Master 通信的代理。节点可以是在云中运行的虚拟机(VM)，也可以是数据中心内的实体机服务器。

如图 1.8 所示，每个节点包含以下内容:

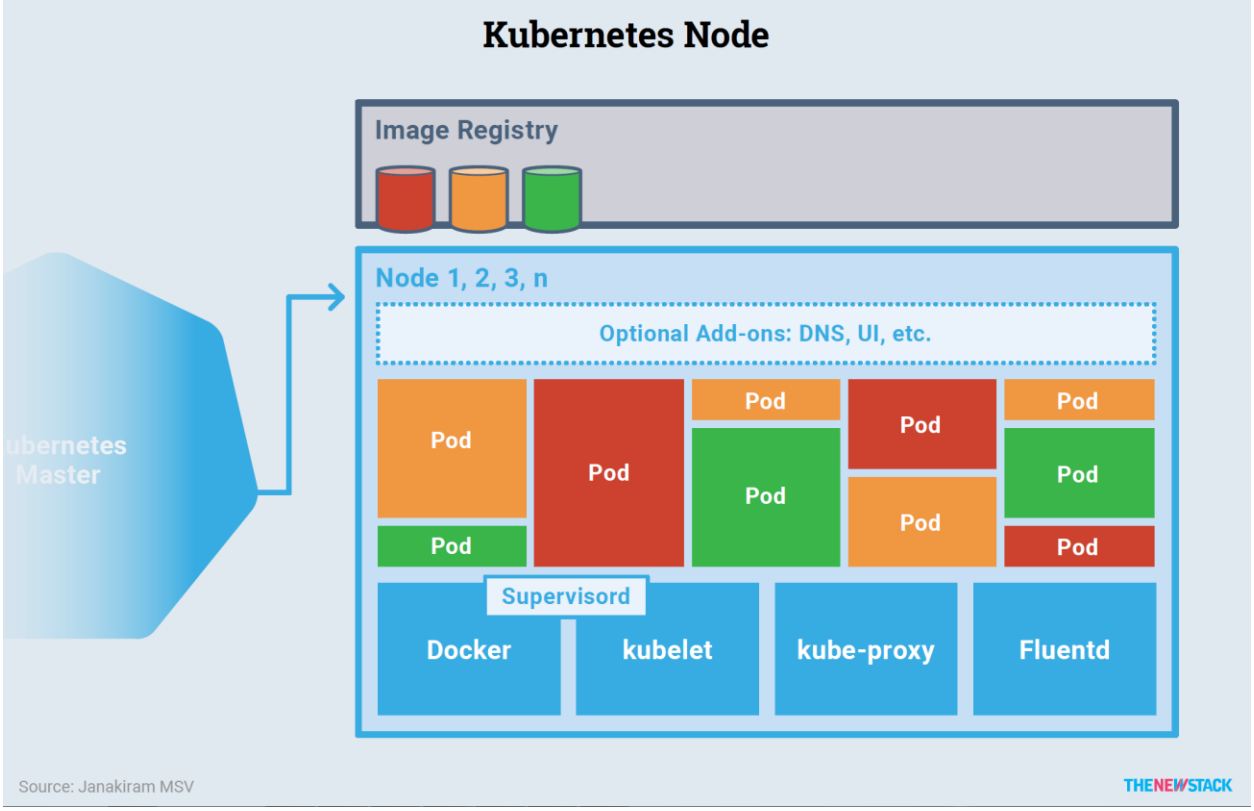


图 1.8：*显示了 Kubernetes 节点中的大量组件的一个分解视图。*

容器运行器

容器运行器负责在节点中运行的每个容器的生命周期管理。在节点上安排了一个 pod 之后，容器运行器就会从镜像仓库中拉取出 pod 所指定的镜像。当一个 pod 被终止时，运行器将杀死属于该 pod 的容器。Kubernetes 可以与任何兼容 OCI 的容器运行器进行通信，包括 Docker 和 rkt。

Kubelet

kubelet 是确保节点上所有容器都健康运行的组件。它与容器运行器进行通信，以执行诸如启动、停止和维护容器等操作。

每个 kubelet 还监测 pod 的状态。当一个 pod 不满足副本控制器定义的理想状态时，它可能在相同的节点上被重新启动。节点的状态每隔几秒通过心跳消息传递给控制器。如果

控制器检测到节点故障，副本控制器会观察到这个状态的变化，并将这些 pod 安排到其他健康的节点上。

Kube-proxy

`kube-proxy` 组件是作为一个网络代理和负载均衡器来实现的。它根据 IP 地址和传入请求的端口号将流量路由到合适的容器。它还利用了基于 os 的特定网络功能，通过操作 `iptables` 定义的策略和规则。每个 `kube-proxy` 都可以与特定容器的网络层集成，如 flannel 和 Calico。

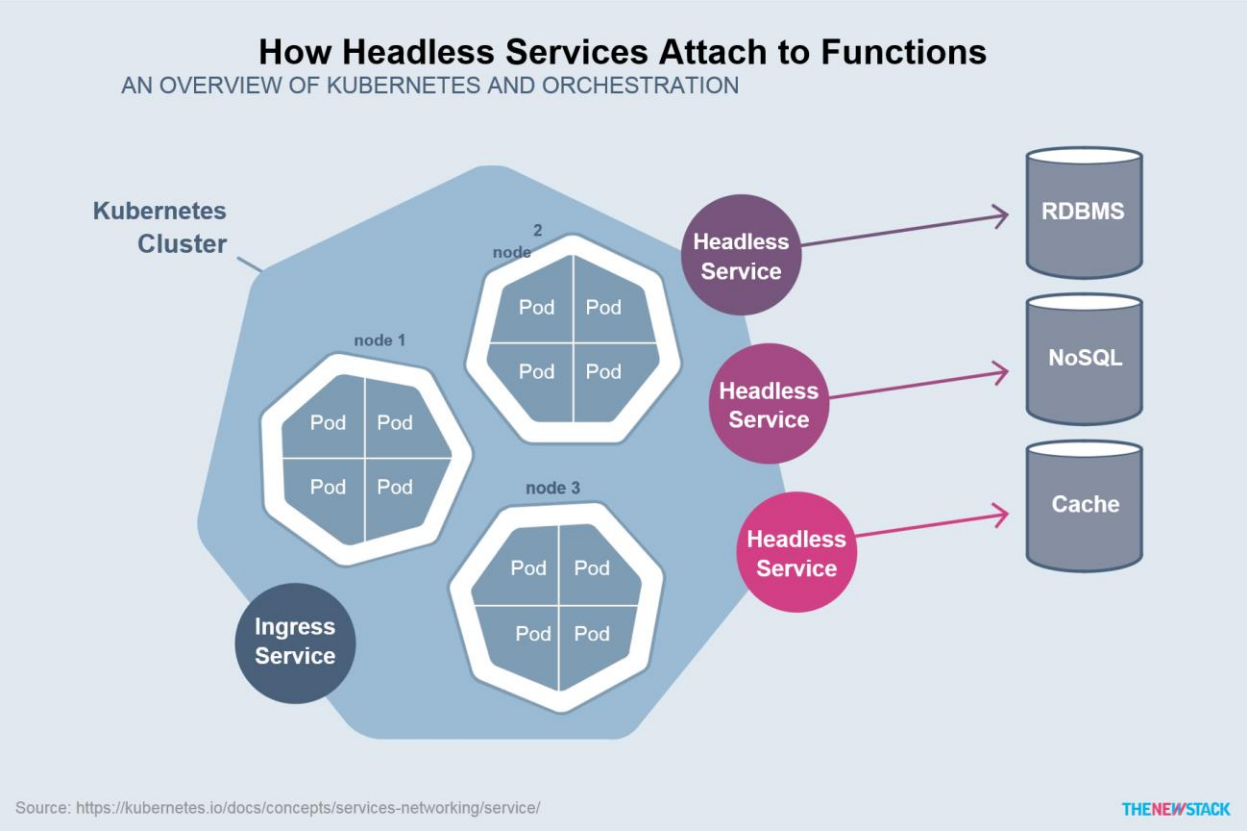


图 1.9：*headless 服务提供状态数据库和服务之间的连接点。*

可以使用一个名为 headless 服务的东西直接将 pod 用于外部服务，如缓存、对象存储和数据库。如图 1.9 所示，这本质上与服务是一样的，但不需要使用 `kube-proxy` 或负载平衡。对 headless 服务的调用将解析为服务专门选择的集群的 IP 地址。通过这种方式，您可以使用自定义的逻辑来选择 IP 地址，绕过正常的路由。

Logging Layer

编排调度器经常使用日志记录作为在每个节点上收集资源使用和性能指标的方法，例如CPU、内存、文件和网络使用。CNCF 定义了一个统一的日志层，用于 Kubernetes 或其他的编排调度器，称为 Fluentd。该组件产生的原因是 Kubernetes 主节点控制器需要跟踪可用的集群资源，以及整个基础设施的健康状态。

Add-Ins

Kubernetes 支持附加插件形式的服务。这些可选的服务（如 DNS 和仪表板）与其他应用程序一样部署，但与诸如 Fluentd 和 `kube-proxy` 等节点上的其他核心组件集成。例如，仪表板外接程序从 Fluentd 提取指标，以充分的显示资源利用率。DNS 插件通过名称解析扩展了 `kube-proxy`。

区分各种 Kubernetes 平台

自从云计算成为现代企业的规范以来，系统架构师一直困惑如何平衡云平台应该采用的封装方式。适当的封装可以提高操作效率，同时提高开发人员的生产力。

在云计算的早期，虚拟机是计算的基本单元。最终用户被迫在所谓的 IaaS+平台之间进行选择，比如 AWS，以及像 Heroku、Engine Yard、Cloud Foundry 和 OpenShift 这样的 PaaS 平台。封装级别决定了开发人员对平台底层基础结构的直接控制程度。更深层次的封装意味着开发人员需要使用平台提供的 API 来驱动编码。

现在，容器作为计算的基本单元出现了，用户也面临着类似的问题，关于选择正确封装级别的需求。正如下面的表格所示，在容器编排领域有许多不同类型的 Kubernetes 发行版。用户的需求——包括工作环境、专业知识的可用性以及用户正在处理的特定场境——将确定是把容器作为服务（CaaS）还是封装的平台的重要考虑因素。没有一个简单明了的框架能够保证完美的满足一切需求。但表 1.1 可能是一个开始。

发行版类型				
描述	**社区支持**	**发行商（无增值）**	**发行商（有增值）**	**应用平台 / PaaS**
开发者封装	无	无	无	有
目标用户	集群管理员	集群管理员/IT	集群管理员/IT	集群管理员/IT/开发者
使用成熟度	准备阶段/PoC	产品化	产品化	产品化
开发者工具/中间件	无	无	也许有	有
DevOps 工具	无	无	也许有	有
Vanilla 发行版	有	也许有	无	无
控制环境	有	有	有	无
优点	资源	运营产品	运营产品	运营和开发
企业支持	没有，只有社区支持	有	有	有

表 1.1：*Kubernetes 发行类型的比较，从完全社区生产到完全商业化*

一个 CaaS 平台包括 Kubernetes 项目，和为其部署和管理提供所需的额外工具。一个封装好的平台，在规模化的另一端，远远超出了 CaaS 提供的运营效率，专注于提高开发人员的生产力。在 CaaS 中，开发人员需要将他们自己的代码打包到一个容器中，以便它可以部署在集群中。尽管基于 docker 的容器在开发人员的名义上解决了这个打包问题，封装好的应用程序平台还可以在内部完全封装构建容器映像的过程——自动化过程，而不是由开发人员那样手动控制。一旦他把他的代码 Push 到像 GitHub 这样的源代码控制工具上，或者像 Jenkins 这样的持续集成/持续交付(CI / CD)系统里，开发人员的任务就停止了，而交给平台做其余的事情。

通过设计，CaaS 与 Kubernetes 开源项目紧密地结合在一起，帮助它大规模的运行和管理容器。但 CaaS 模型希望开发人员能够处理所有应用程序的依赖项。从文化的角度来看，DevOps 模型遵循了 Dev 和 Ops 一起与跨功能知识协同工作的文化。在这个场景中，两个团队都知道的是完成一件事是需要依赖很长的列表。

封装好的应用程序平台使用 Kubernetes 作为核心组件，帮助它以比 CaaS 更少的开销运行容器。开发人员不必担心管理运行时或任何应用程序依赖项。相反，他们可以专注于编写应用程序代码并将其推送到源代码控制存储库或 CI/CD 系统。封装好的平台提高了开发人员的生产力，同时削减了底层组件的控制。可以说 DevOps 仍然是这个模型的中

心。但由于封装，并不需要这种跨功能的知识——它变成了冗余的繁杂工作。开发人员不需要了解 Kubernetes 的基础，也不需要了解如何管理它。

正如我们前面提到的，在 CaaS 和封装好平台之间选择不存在简单的框架。您的选择取决于您的团队希望达到的开发生产力水平和您所能预见的团队的特定需求集合。

开始使用 Kubernetes

这里有一些方法可以开始 Kubernetes 的部署：

1. 注册一个托管的 CaaS 服务。
2. 下载并安装一个单处理器的安装程序，用于在单个 PC 上测试 Kubernetes，比如 CoreOS' Tectonic Sandbox。
3. 通过克隆 Kubernetes GitHub repo 建立一个本地集群
4. 尝试一下 Kubernetes 集群，Play with Kubernetes，这将给你一个完整的 Kubernetes 环境，你可以从你的浏览器运行。

包括 IBM Bluemix、谷歌云平台和微软 Azure 在内的公共云服务提供商提供了托管的 Kubernetes 作为服务。通过免费的积分和试用优惠，我们很容易就能把一个小的 Kubernetes 集群用来进行测试。有关在 Kubernetes 部署第一个应用程序的详细信息，请参阅您选择的服务的文档。

Minikube 是一个单节点 Kubernetes 集群，用于学习和探索环境。它是强烈推荐给过去没有经验的初学者。

在功能更强大的主机上，Kubernetes 可以被配置为基于 Vagrant 的多节点集群。GitHub repo 有关于设置 Vagrant boxes 的说明。

规划 Kubernetes 路线图

在这段 podcast 中，Google 集合了监督和贡献项目路线图的三名成员。在一个容器化的环境里面， 一直存在操作系统，并且我们一直可以把“Linux 容器”进程当作一个服务的工具来调用。但是， 应用程序间通信(IAC)不再是由底层操作系统平台所推动的。作为一个调度者，Kubernetes 帮助组件之间搭建好了网络。如何实现这一便利，将为那些实现 Kubernetes 路线图的人提供了 一个重要的话题。社区中的 PM 团体，该团体维护开放源码的路线图，并且是 CNCF 管理委员会的成员。**播放 Podcast**。

***Aparna Sinha** 在谷歌管理 Kubernetes 的产品组。 她开始并共同领导 Kubernetes 社区 PM SIG，该团队维护开源路线图，是 CNCF 理事会的成员。*

***Eric Brewer** 领导 Google 的计算基础架构设计，包括 Google 云平台。 作为加利福尼亚大学伯克利分校计算机科学长期教授，他领导了可扩展服务器，网络基础设施，物联网和 CAP 理论的项目。*

***Ihor Dvoretskyi** 是 CNCF 的大使，Kubernetes 社区的产品经理和 OpenStack SIG 负责人以及 Mirantis 的项目经理。 他的重点是 Kubernetes 和 OpenStack 之间的紧密集成。*

Kubernetes 1.7 和可扩展性

离开了 Kubernetes 的人，也许整个编排的概念永远不会实现，他不相信 Kubernetes 是真正的新兴生态系统的中心，显然这样的人是不存在的；是的，我们称之为 Kubernetes 生态系统， 但谷歌的 Tim Hockin 解释说，他认为这个平台是一个更大的生态系统的中心。Hockin 说， Kubernetes 在开始时真正想做的，是提供生态系统的中心。这里有网络生态系统、存储生态系统和安全生态系统。

在这段音频中，将学习到平台的新扩展模型如何能够支持与安全策略的集成，和如何使用您手边的任何策略模型。**播放 Podcast**。

***Tim Hockin** 是谷歌的一名工程师，从事 Kubernetes 和谷歌容器引擎（GKE）的工作。 他是 Kubernetes 项目的联合创始人，负责网络、存储、节点、联合、资源隔离和集群 共享等主题。*

Kubernetes 生态系统图

作者：**SCOTT M. FULTON III**

“生态系统”这个词在 20 世纪 70 年代末首次应用于计算机，用来解释当时新兴领域苹果公司 Apple II 的软件环境。那个时候，几乎所有的软件都是由电脑制造商独家发布的。但是苹果公司却选择了接受独立软件供应商的产品，其中一些软件厂商将他们的软盘包装打包， 并且使用了彩色印刷的说明书。当计算机制造商意识到所有各方之间的关系，对包括自己在内的所有人都有利时，苹果公司在我们的小小世界中开启了第一个生态系统。

将近 40 年过去了，大多数的计算平台都已经创造自己的生态系统——一种结合生产和经济目标的系统，那些创造并支持该产品的人，反过来又得到了它的支持。Kubernetes 社区许多领导者的观念准则是与强制排他性支持对立的。

重新发明轮子没有意义

“Kubernetes 是一个平台，它不应该定义你使用什么样的监控解决方案。” Tim Hockin， 谷歌的首席工程师，他说： “我们应该这样做，并且我一直致力于在这样的决定中保持中立。 我认为采纳下面的方式是站不住脚的，他举例说，’ 请不但要安装 Kubernetes（这本身就是一堆东西），而且还要通过 Prometheus 改变你的监控，而且还要通过 Fluentd 改变你的所有日志，这样，你还要使用 gRPC 和 OpenTracing。’ ，这将无法工作。”

“Kubernetes 是一个支持插入的系统，这非常有意义” ，Hockin 继续说到，” 这样你就能把多个解决方案整合到它里面。现在像 Prometheus 成为了家庭的一部分，我希望人们把 Prometheus 看成是原生云工作的方式的一种。如果他们没有现成的监控解决方案，他们也许会关注 CNCF[原生云计算基金会]正在做的其他事情，并且考虑采用这些技术。但我的意思是‘考虑’ ，而不是‘被迫接受’ “。 CNCF 也有自己的生态系统图，叫做“Cloud Native Landscape” 。在这张生态图上，Docker 很容易被找到， 而且你可以在"调度和编排"表中找到 Kubernetes，以及 Docker Swarm、 Mesos、 Nomad 和亚马逊的弹性容器化服务(现在是 EC2 容器化服务)。 但我们在这一章的目的是为了描

绘进展的状态，在本书出版时，迄今为止，这一生态系统已经与 Kubernetes 有了明确的结合。首先一个有说服力的论据是，容器生态系统（关于 The New Stack 已经出版了电子书）几乎完全由 Docker 推动。尽管许多组织机构声称已经为划分命名空间或可移植的工作负载的概念上做出了贡献，但只有这个标志的工作才会很容易的被认出来：一条蓝色的大鲸鱼。

Kubernetes 并没有真正创造编排和适配工作负载。但它首次做出了一个统一的方法概念，就是在某个级别上启用了进程间通信和可伸缩性，并且易于实现，这个是 Docker 并没有单独实现的部分。

对于大多数企业来说，接受 Kubernetes 作为一个平台，在更广泛的程度上，作为一种方法论，而并非仅仅是一种工具，虽然它确实需要被看作是一套广泛可互换工具的先驱。Tim Hockin 的观点是，平台不能被认为是通过一个单一的工具集而试图建立一种单一的工作方式。就像是一个操作系统，如果它打算这么做的话，那么就会有认证的软件和授权的扩展库。

Laura Frank，CI/CD 平台供应商的工程主管，使这个想法更进一步：她认为，一个生态系统的关键点是不能指望开发者先去提供资源给它，即使他们有自主研发的工具，要让他们在这之前已经能够从中获益。“我的实用主义思想总是让我使用一种存在的工具，而不是试图编写自己的工具，” Frank 告诉我们。“坦率的讲，和 Kubernetes 的滚动升级、自愈性的特点相比，我不能写出比它更好的解决方案。我认为尝试重新发明轮子是没有意义的，依赖 Kubernetes 做类似的事情是非常棒的，相反的是，我们可以尝试自动化它们，甚至自己去重写这些部分；我认为依靠工具很棒，我认为关于这点：‘你不是 Google，你不存在 Google-sized 的问题’，这个确实不再是个问题了。“

“Kubernetes 拥有如此丰富的生态系统，围绕它有开源社区，这个不仅仅代表了一家公司的利益，” 她继续说到，“它实际上是贯穿许多不同行业的一个合作，并且有相当规模的工程师团队和组织聚集一起，创造一种能以最好的方式一起工作的东西，以及大量的使用案例。”

DevOps 流水线

图 2.1 展示了我们对 Kubernetes 生态系统现状的描述。这张图看起来有点儿不平衡。这并不是偶然的，这些遗漏并不是因为我们匆忙出版这本电子书导致的。虽然许多人很快就会有所发现，并宣称 Kubernetes 是一个完整的 DevOps 生命周期管理框架，当我们划分 DevOps 流水线时，以开发为中心的工具放在左边，以操作为中心的工具放在右边，生态系统工具的则倾向于右边。

这不是某种生态系统缺陷。实际上使用 Kubernetes 去创建和部署一个微服务，相当于使用一个煎饼锅去煎一个鸡蛋一样。为实现这个功能，存在大量的独立的开发环境。此外，Docker 还可以把这种开发的产品打包到便携的容器中。Kubernetes 社区正在研发一种叫做 CRI-O 的工具集，是为了将预先做好的容器放到标准引擎中。但它不能取代 Docker，CRI-O 不能创建容器镜像。流水线列表中的“Create”列下面的项目中，包括 Telepresence：一个为在 Kubernetes 搭建微服务的本地开发环境。和 rkt 和 containerd 的运行时一样，它们现在都是 CNCF 的工程了；Codeship 出现在流水线图中的“Package”和“Release”的阶段，可在你持续部署模式整合微服务的时候使用它。

在“Configure”列中，出现了许多管理平台：这些系统并不像 OpenShift 那样提供端对端生命周期管理，但是他们向用户提供了各种超越了 Kubernetes 本身的增值服务。还有就是“Monitor”列显示了许多的日志，监控，监视工具，而且这些平台能够融入到 Kubernetes 系统中（例如 Fluentd 中统一的日志层），或者作为插件集成到 Kubernetes 中（例如：New Relic APM，AppDynamics，Dynatrace，Sysdig，VMware Wavefront）。许多监控平台都是在 Kubernetes 诞生之前的几年前发布的，可是他们已经被带入了这个平台的影响范围。 在图 2.1 的底部是支持 Kubernetes 的基础设施服务，比如 Quay 容器部署平台、 Nuage Networks 的 SDN、还有 Twistlock 的容器安全平台。

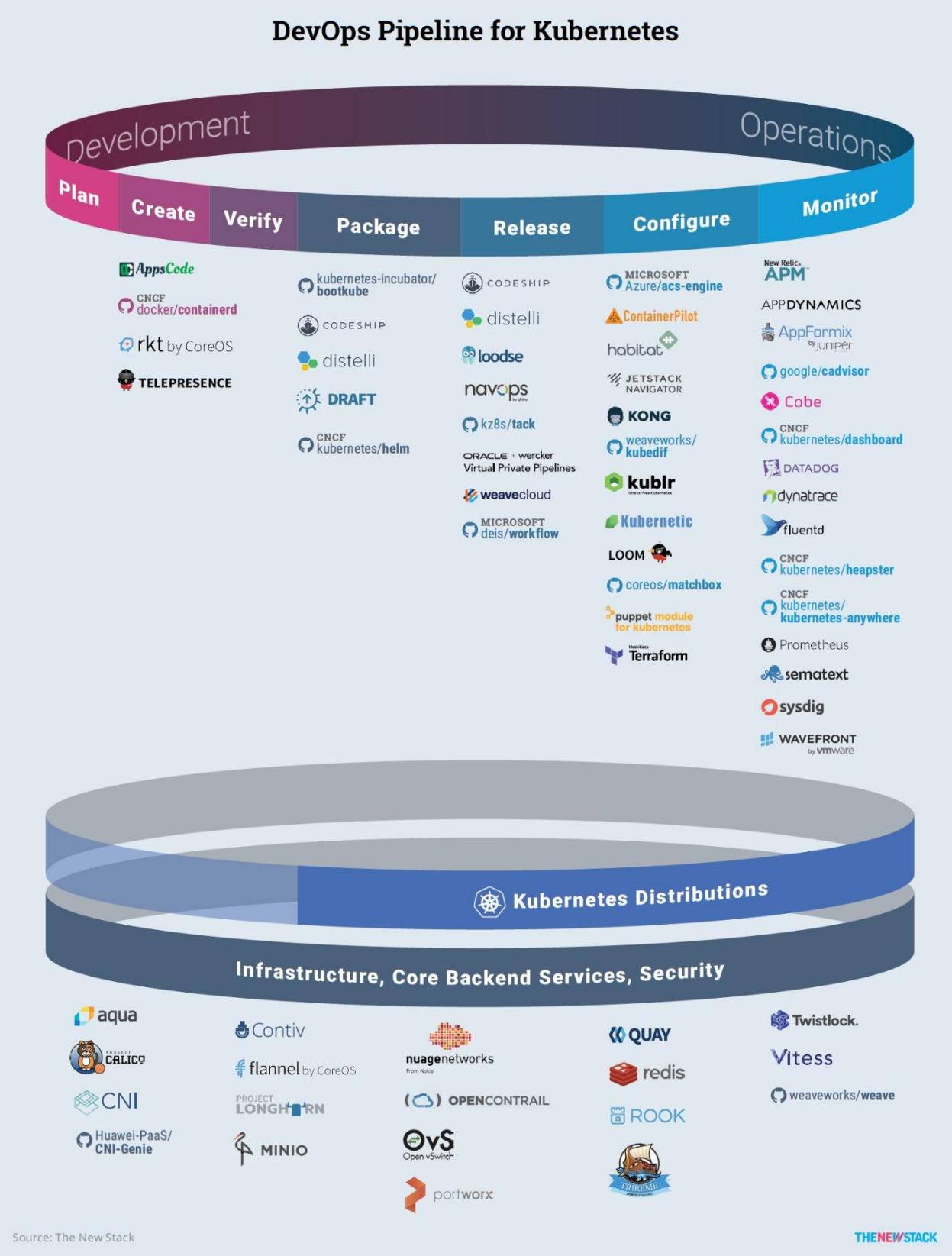


图 2.1：*现在的 Kubernetes DevOps 流水线。*

“这里确实存在一些差异，保护容器和保护虚拟机之间的基础设施是不同的。”Twistlock 的首席执行官 John Morello 说到。“有许多错误的文章，描述容器是虚拟机的下一个演变。虽然它们表明上显示出了许多相似之处，但使用方式和核心技术都有着显著的区别。如果你把容器看作是一种小型的虚拟机，那么这将会对容器带来一系列的偏见和假设，显然这些都是不正确的。”

Morello 发现，一个容器化的、分布式的应用程序，是一个丰富的拥有许多对象的小型实体。与之形成对比的是，来自开发者的观点认为，目前构造的应用程序仍然是一个单一的实例。这也许可以解释为什么今天的 Kubernetes DevOps 流水线出现了一些不平衡的现象：当应用程序到达生产阶段时，采用会构建越来越少而且不再重要了。

支持层级

图 2.2 列出了 Kubernetes 中的包和服务，你可以直接获得，或者购买，或者购买服务；我们分为如下几类：

- **社区支持：**发布了许多免费且开源的包，通过这些包，你自己可以尝试部署这个平台。Minikube 是一个真正的本地 Kubernetes 环境，可以在单机上进行实验性部署。
- **供应商发行版(没有增值)：**这类的公司能够交付软件的解决方案，或者云基础平台，其中包括真正 Kubernetes 的售后支持。
- **供应商发行版(有增值)：**这类的公司的系统以 Kubernetes 为核心，展示了更完整的环境，包括调度、开发和生命周期管理。
- **APP 平台/PaaS：**展示了作为服务的完整平台，包括 Red Hat OpenShift，能够有效地将 Kubernetes 的管理和维护封装成为端到端，自动化开发和部署环境。

对于那些想获得详细信息的人，可以关注这个由社区管理，实时更新的 Kubernetes 发行列表。

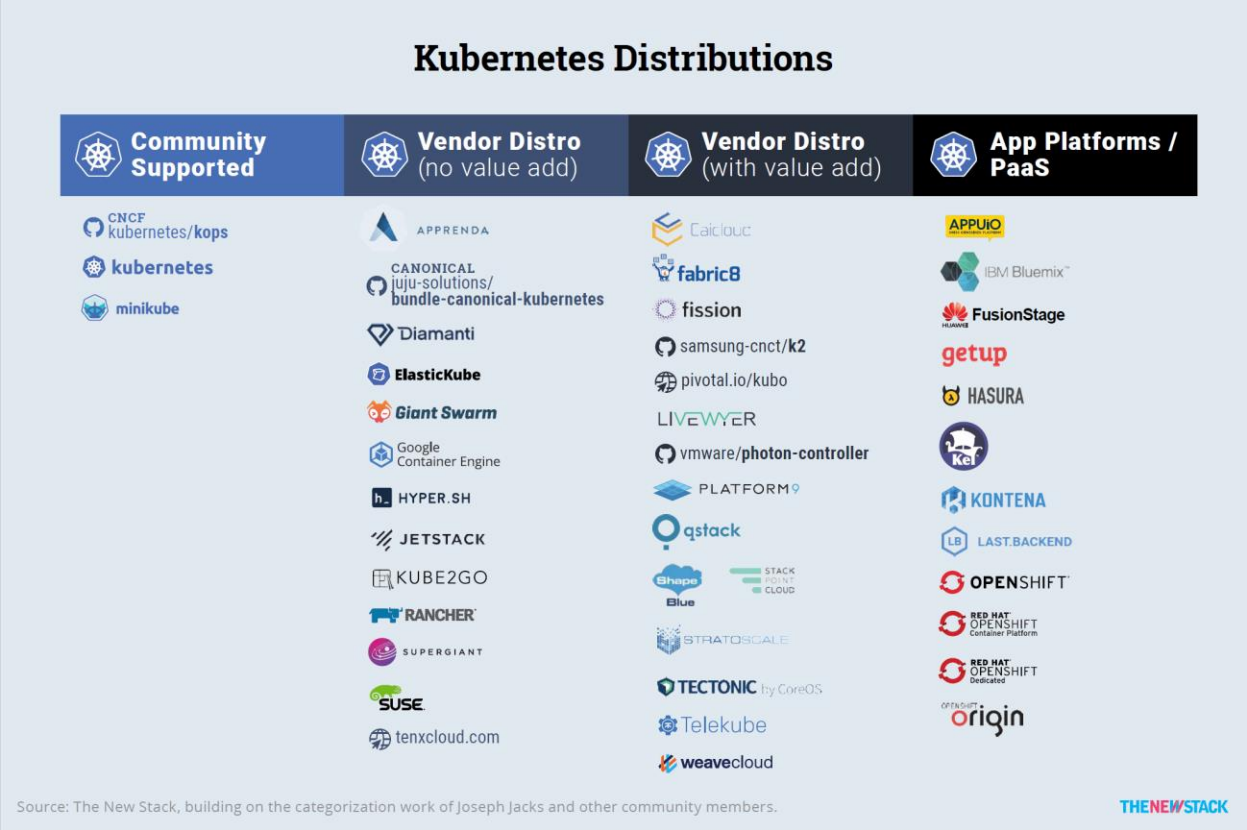


图 2.2：*Kubernetes 发行商的不同分类。*

Red Hat 的 Kubernetes 和 OpenShift 产品经理 Brian Gracely 告诉我们，他期望用户对于 Kubernetes 的认知应该减少，而去使用包含它的平台（例如：它自身具备的）的势头应该持续增长。“最终，它的一种结果，是你是否围绕这个想法终于建立了一个商业模式，这开始变得没有意思？” Gracely 问到。“这个时候，你有一个选择，在它的上面或者围绕它，实现一些与众不同之处。我们现在看到的是，每一个健康的生态系统都围绕着大量的公司，想成为数字变革时代的一部分——云原生应用——这太奇妙了。我们看到人们从不同的角度去接近它。Red Hat 作为一种软件实现；我们也可以通过 OpenShift Online 作为服务实现它；我们看到主要的云供应商，把它制作成了服务，任何人都可以使用，这是一个新的商业模式，一个伟大的商业模式。”

“我认为 Kubernetes 的生态系统处于早期阶段看起来是相当不错的，” 他继续说到。"还有很多正在进行的创新，以及非常酷的想法。如果你与正在这些事情做出奉献的大量工程技术人才交流的话——Red Hat，Microsoft，Google，Intel，华为，IBM——会发现最终我们的期望是核心技术保持稳定。人们会以不同的方式进入这个市场。是的，五年

后，也许我们不把它叫做 ‘Kubernetes 生态系统’ 。但如果我是一个关注 Kubernetes 的人，当人们在说 ‘嗯，它是一种枯燥的技术’ ，这是很好的事情。商业用户的描述更贴切一些： ‘这是我不需要考虑的事情 ‘” 。

编排和开发者的文化

对于很年轻的在容器业务来说，将微服务思想与企业开发和部署流水线相结合的最成功的从业者之一，是 Berlin CI / CD 平台提供商 Codeship 的工程总监 Laura Frank。

Frank 作为 Docker 团队的领导者接触的企业越多，她越了解当今组织机构中开发者所面临的文化问题，以及像 Kubernetes 这样的平台在多大程度上也许无法实现革命性的变革。 在这段 Podcast 中，听到更多关于“领队” Frank 的企业经验，试图在这些新的和仍未知的领域提供一些指导。**播放 Podcast**。

***Laura Frank** 是 Codeship 工程总监和 Docker 领队。 她的主要工作重点是加强 Docker 的基础设施，并改善其整体 CI / CD 体验。 在加入 Docker 之前，她参与了 HPE 的公共云产品和 OpenStack 项目。*

用户体验调查

作者：**LAWRENCE HECHT**

有些人推测 Kubernetes 已经赢得了容器编排的战争。如果确实有这样的战争，除非出现在任何情况下都与市场兼容的技术，那么 Kubernetes 的优势很难否认。这样的结论已经被许多早期使用者得出：Kubernetes 平台特性与社区结合使它比其他容器编排器更有优势。

《The New Stack 的用户体验调查》并没有最终证明 Kubernetes 已经成为了我们调查的组织或团队中事实上的容器编排工具。尽管如此，它提供了一个清晰的场景，人们如何评估和部署 Kubernetes，他们面临的挑战，以及他们对一些有竞争力的产品的看法。

重要发现

- **生产现状：**随着越来越多的容器应用程序进入生产，Kubernetes 被采用也越来越多。
- **竞争的大门仍然敞开：**可能还有 Kubernetes 的竞争对手的机会。无论是 Docker 集群、AWS EC2 容器服务(ECS)、中间层企业 DC / OS，还是其他任何产品，实际上都将可能在这一领域成功地挑战 Kubernetes，这很大程度上取决于 Kubernetes 用户的整体满意度，以及其他因素，如技术优势和商业价值。
- **更多希望：**你会发现用户经常抱怨平台的复杂性，执行困难，以及维护头疼等问题。分布式系统本质上是复杂的，Kubernetes 当然也不例外。但至少就目前而言，用户普遍对 Kubernetes 的做法感到满意。
- **商业平台比比皆是：**受访的 Kubernetes 用户中有 45%已经实施了供应商分销，无论是将其集成到平台中，还附加了额外的软件，或者只是捆绑在企业支持上。无论是作为服务（PaaS）的平台，还是作为服务（CaaS）的容器，OpenShift（可以说是 PaaS）和 Google 容器引擎（CaaS 平台）等平台正变得越来越流行。提供标准的支持实现或管理平台的供应商可能会面临更少的成功，除非他们选择更加独特和差异化的产品。

- **越简单越好**：在我们的调研中，Flannel 是针对 Kubernetes 的软件定义网络的主要网络层。集群运营商大量采用 Project Calico 项目，它是一个开放源码的边界网关协议（BGP）路由器项目。他们也刚刚开始在生产中实现 Kubernetes。
- **新的成熟的监测方法**：在我们的调查对象中，Prometheus 是最常用于监测 Kubernetes 集群的工具。然而，Heapster 在我们调查的团队中也得到了显著的应用。传统的监控供应商不太乐观，尽管当它们被集成到一个更大的自定义的监视平台时，使用级别似乎在增加。
- **将容器化带给运营商**：在我们调查的组织或团队中，集群运营商的工作角色是最有可能实施 Kubernetes 的组织机构。虽然应用程序操作员的参与程度较低，但是当平台作为持续部署策略的一部分部署时，最终可能会获得最大的收益。

样本与方法

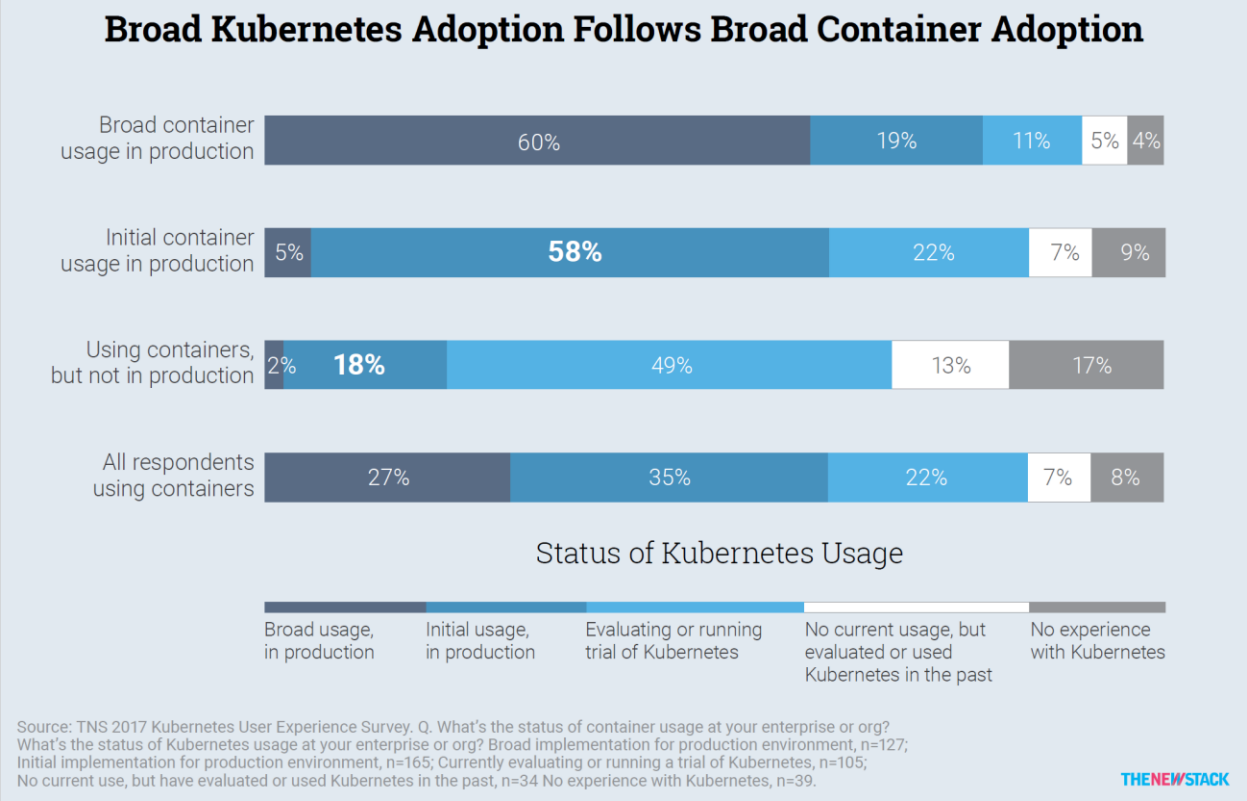
从 2017 年 5 月 15 日至 5 月 29 日，由 New Stack 进行的基于网络的调查，得出了本章的观察和结论。我们的报告基于来自 470 个人的数据子集，这些人认为他们所在的组织机构或团队是容器用户。有实践经验的人被问及很多问题以确定他们的意见和观察，尽管这些问题被用来问其他 Kubernetes 的评估者。 我们作出了额外的努力来区分实际的 Kubernetes 最终用户与基础设施和工具市场成员的反应。如果受访者的雇主提供 PaaS，基础架构即服务（IaaS）或软件部署工具，我们要求该人限制对雇主公司使用或评估 Kubernetes 的回应。 供应商的回应可能会影响我们的调查结果。

我们的样本中有一半以上是由拥有超过 100 名员工的企业雇佣或签约的员工组成的，其中 24%的人在拥有 1000 多名员工的公司工作。我们还会告诉你，公司规模可能对我们的研究结果产生重大影响。 在我们的网络调查中，受访者主要是虚拟基础设施和分布式系统平台的用户，或可能是更广泛的 IT 市场的代表。我们提供了一个干净的数据版本 供社区评审。

谁是 Kubernetes 用户

正如我们在 The New Stack 的 2016 年调查中所报告的(参见“The Present State of Container Orchestration”)，直到已经将第一个容器化的应用程序部署到生产环境中，

人们才意识到他们所在的的组织或团队需要进行容器编制。 这个调查似乎成了被调查者判断的依据，用来部署管理他们的容器基础设施。



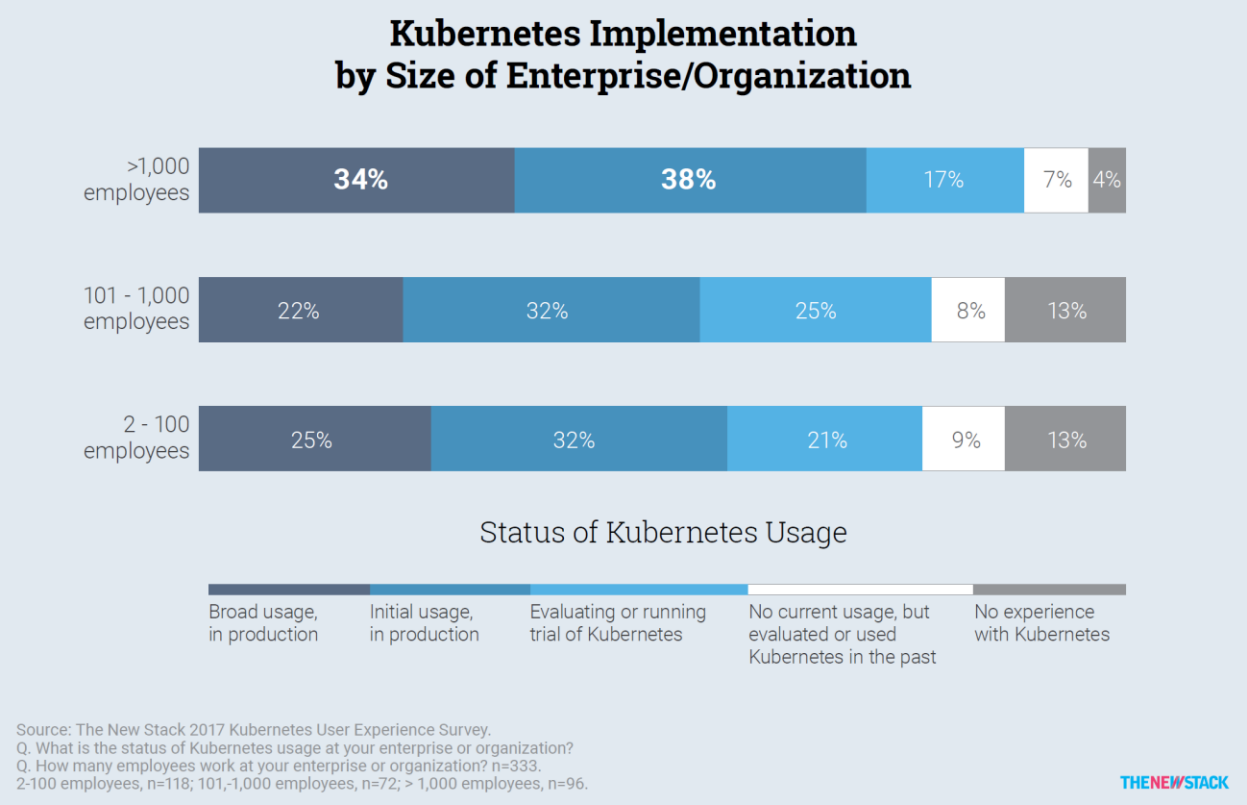
我们要求开发人员描述其组织或团队部署容器的现状。注意，所有这些开发人员的组织在某种程度上正在使用容器。在所有接受调查的开发者中，约 27%的受访者表示，他们所在的组织或团队已经广泛地采用了 Kubernetes。另有 35%的人表示，他们所在的组织或团队正处于 Kubernetes 采用的最初阶段。

然而，当我们根据他们的生产环境对这一问题进行调查时，只有 5%的被调查者在容器使用的初始阶段已经广泛地采用了 Kubernetes。然而，近五分之三的人已经开始接受他们的 Kubernetes 环境。相比之下，只有 18%使用了容器的企业，尽管还没有在产品中使用，却还处在在使用 Kubernetes 的初期阶段。这是一种明确的信号表明，在绝大多数的公司在生产过程中完全采用了容器后，开始使用 Kubernetes 了。

换句话说，如果容器没有进入生产过程中使用的阶段，不要指望 Kubernetes 会实现这个目标。

大公司更喜欢 Kubernetes？

我们给了受访者一份有 5 个短语的列表，并让他们选择一个他们认为的最能描述在自己所在的组织或团队中部署的 Kubernetes 相对状态的短语。结论是在大公司工作的受访者更有可能成为 Kubernetes 的用户。当被要求描述 Kubernetes 在他们的组织或团队中的使用情况时，超过三分之一的受访者（34%）表示他们处于部署阶段，处在初始阶段有 38%。相比之下，中等规模企业中只有 22%的受访者和约 25%的小公司的受访者普遍实施了 Kubernetes。

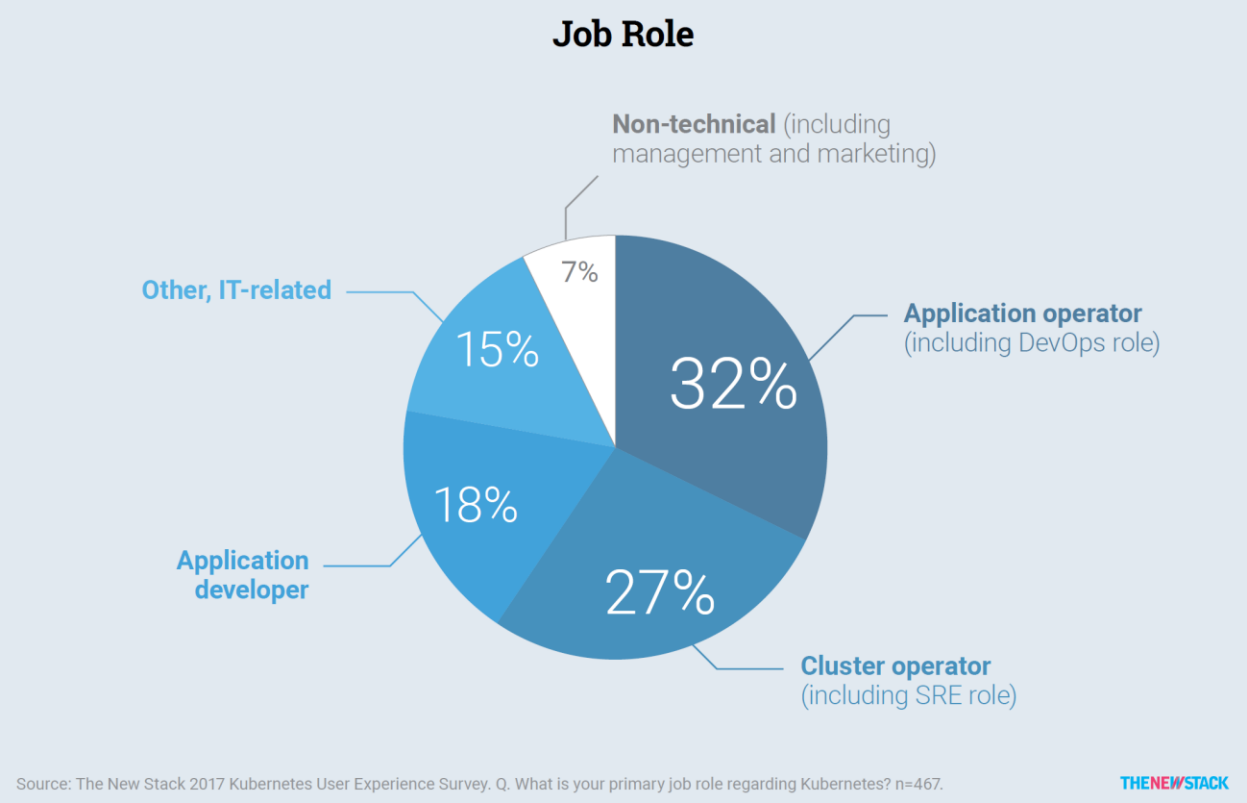


根据 Datadog 的客户分析，大型组织机构比其他人更早地开始了使用容器技术。这样说也是有道理的，大公司在采用可减轻手工操作的容器编排系统方面走的更远。

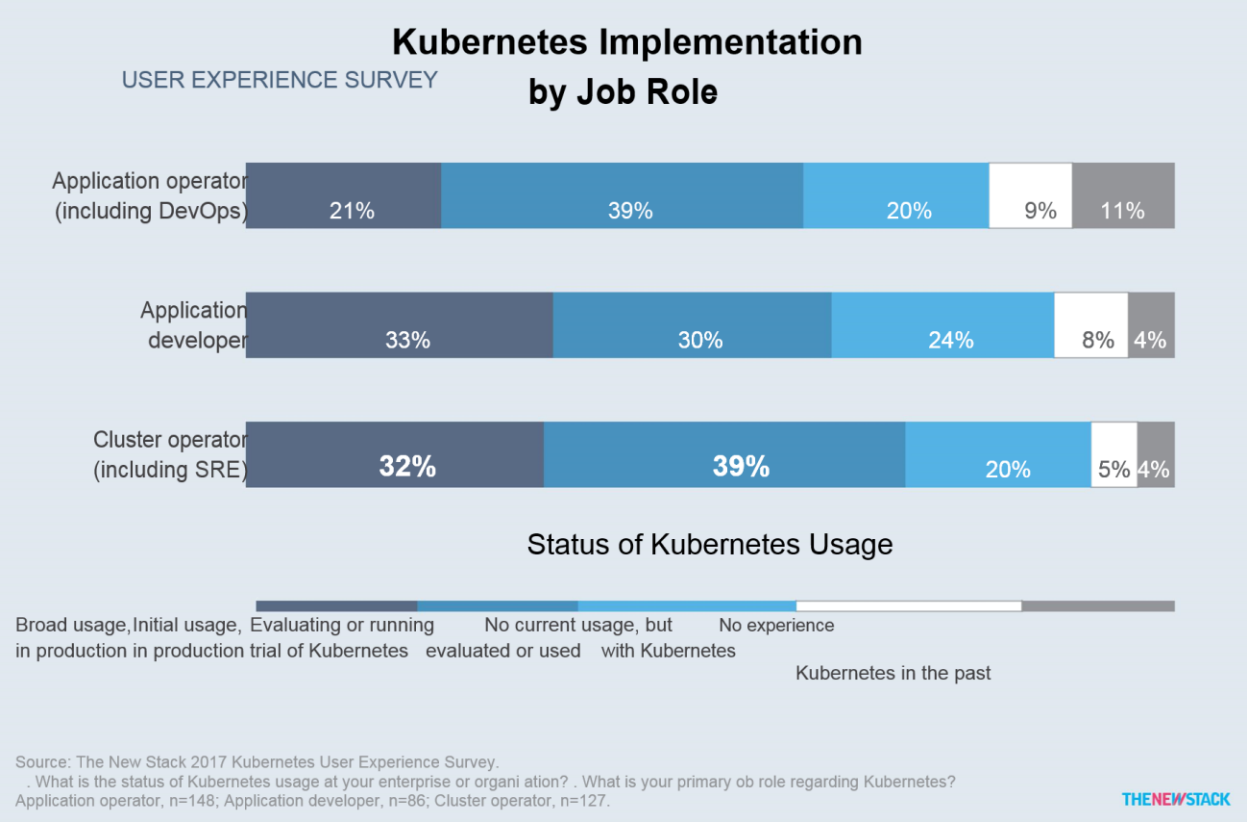
容器用户的工作需求

在我们的调查中，我们询问了所有受访者，不考虑他们所在的组织或团队对 Kubernetes 的使用情况，他们对 Kubernetes 的工作角色的描述都是一样的。我们给他们列出了四种选择——加上一项，其他与 IT 相关的——并要求他们选择最适合他们与编配器的工作关

系的角色，即使这是在初始阶段。而结论印证了我们在之前的容器调查中所看到的，相当多的(32%)的受访者自认为是一个 DevOps 的角色。在本报告中，我们定义了一个在 Kubernetes 上管理应用程序的操作的人，是一个 DevOps 角色。另外 27%的人从一个集群操作员或站点可靠性工程师（SRE）的角度来关注 Kubernetes。虽然容器的广泛使用是在应用程序开发人员中开始的，但只有 18%的受访者选择将其作为与 Kubernetes 相关的工作角色。



在集群运营商和有 SRE 角色的人群中， 71%的人所在的组织或团队把 Kubernetes 作为产品一部分——这是我们调查中最多的工作岗位。在具有应用程序操作员背景的人中，这个数字下降到 60%。集群运营商可能更可能是系统管理人员，而不是开发人员，这可能解释了为什么他们更关心 Kubernetes 的管理，以及它是如何与其他系统集成的。



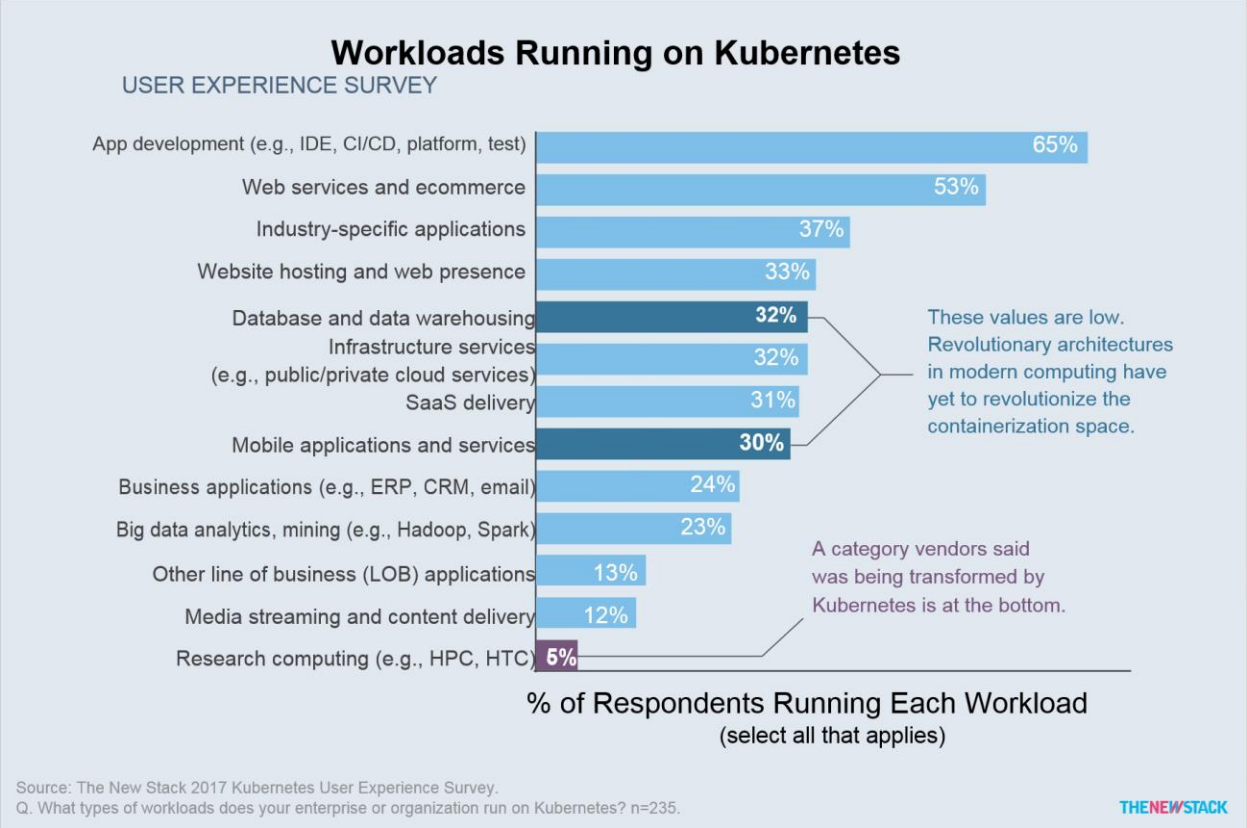
应用程序操作员和那些拥有 DevOps 角色的人并不一定对 Kubernetes 本身感兴趣，他们可能和那些尚未在生产中采用 Kubernetes 组织或团队里的人成为对 Kubernetes 感兴趣的第二梯队。

Kubernetes 是如何在生产中使用的

对于大约三分之二的组织或团队来说，Kubernetes，一个据说是由开发者主导的革命性的产物，是开发人员使用的应用程序。虽然开发人员并不是我们大多数的受访者，但他们是最初的 Docker 革命的拥护者，而且他们首先从更高效的部署环节中获益。

编排工作负载类型

我们把已经将应用程序部署到 Kubernetes 的组织或团队的受访者当成一个子集，为他们提供了 13 个关于这些应用程序的工作类型的列表，并要求它们选择何种类型的应用在自己的生产环境中使用。

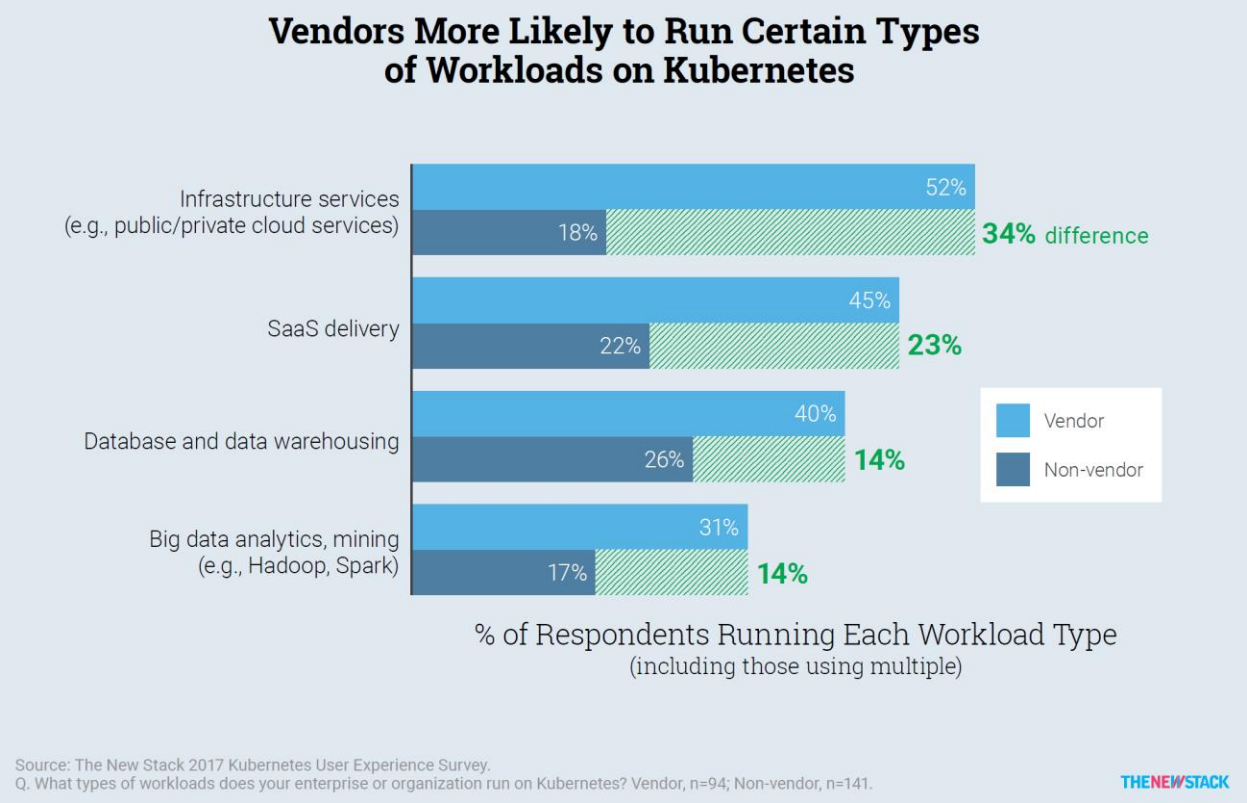


大约三分之二的 Kubernetes 的生产实例(65%)正在运行专门用于开发过程本身的应用程序。这个数字进一步证明了开发人员是编排分布式系统架构背后的驱动因素。尽管一些电子商务应用程序依赖于状态（例如，来自数据库的持久数据）来支持事务，但不依赖于状态的 web 服务是一类最早的应用程序，它们都已经被容器化，而且，可以转移到微服务体系结构中。毫不奇怪， Kubernetes 上运行的第二种可能的工作负载是 web 服务和电子商务（53%）。

只有不到三分之一的受访者(32%)表示他们的组织或团队在 Kubernetes 上运行数据库或数据仓库服务，而更少的(30%)正在运行移动应用服务器和移动服务。这些数字远非可以忽略不计，但它们的低价值表明，在现代计算中，另外两种所谓的革命架构——大数据和移动设备——尚未彻底容器化。令人惊讶的是，只有 5%的受访者表示，他们基于 Kubernetes 为基础的工作负载是基于诸如高性能计算之类的研究工作。供应商们说这是一种在 Kubernetes 历史早期就会被改变的行业。

供应商使用实例

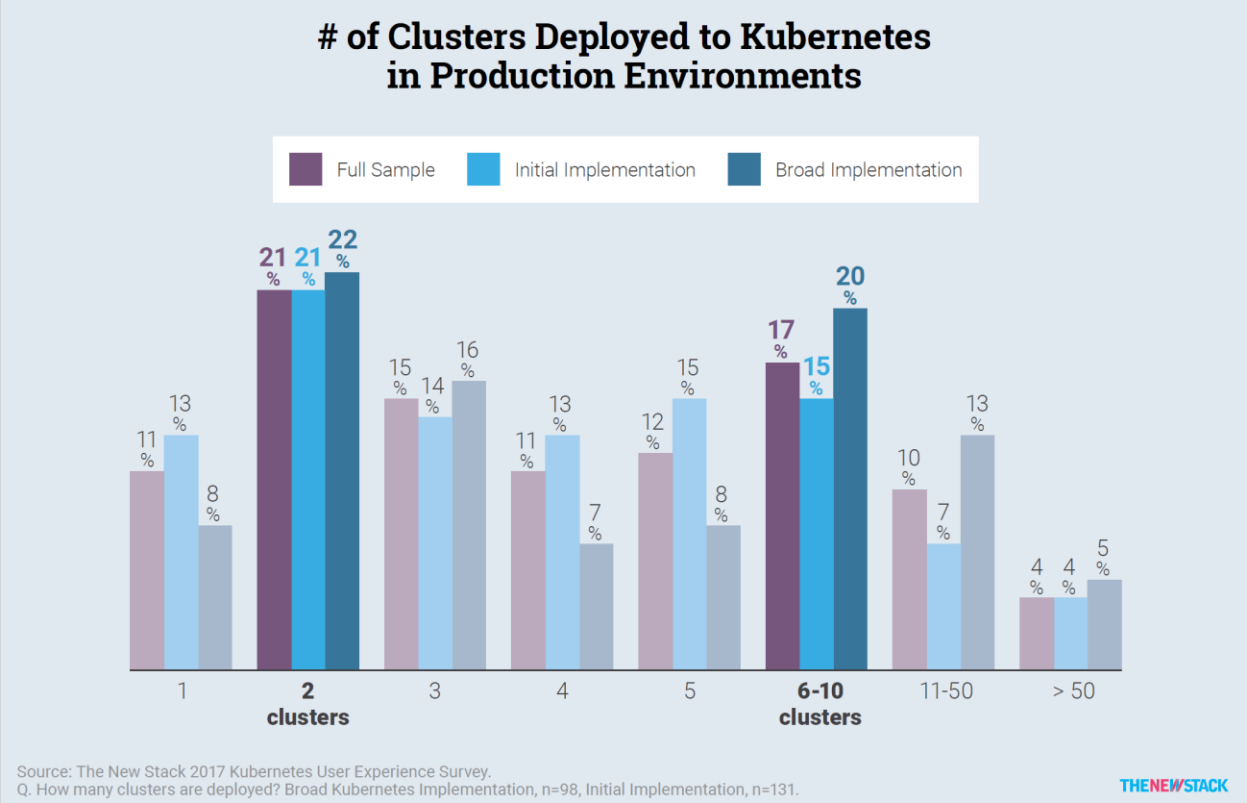
我们给了供应商和最终用户四大类服务，并要求他们选择所有应用于他们的 Kubernetes 部署的服务。与最终用户相比，供应商更有可能运行与供应云基础设施和数据库等基础服务相关的工作负载。一些非供应商的用户可能会从外部供应商那里得到这些服务，这些供应商基于 Kubernetes 提供服务。



供应商更可能说他们正在运行有状态的工作负载（传统类型）而不是无状态的服务。约有 40%的供应商表示他们在 Kubernetes 上运行数据库和数据仓库操作，而非供应商只有 26%。同样，有 31%的供应商在 Kubernetes 上运行大数据分析，而非供应商则为 17%。这些明显的差异是超级计算机和软件服务提供商的要求不同于商业企业的明确体现。

部署的规模和广泛程度

我们要求受访者对他们组织或团队当前 Kubernetes 部署的集群进行量化，然后将他们的答案分成 8 个部分。在所有受访者中，约有 21%表示他们的组织部署了两个 Kubernetes 的集群，这个数字在刚刚开始采用 Kubernetes 的参与者（21%）和那些广泛使用的人（22%）中保持稳定。



第二个峰值出现在 6 到 10 个集群范围内。当集群的数量达到两位数时，第二个“峰值”就会变得更加明显。6 到 10 个集群的 17%的数据代表了这一范围内 20%的广泛使用者和 15%的初始实现者的平均水平。这应该给您一个大致的概念，即集群正在被添加到逐步成熟的 Kubernetes 部署中。

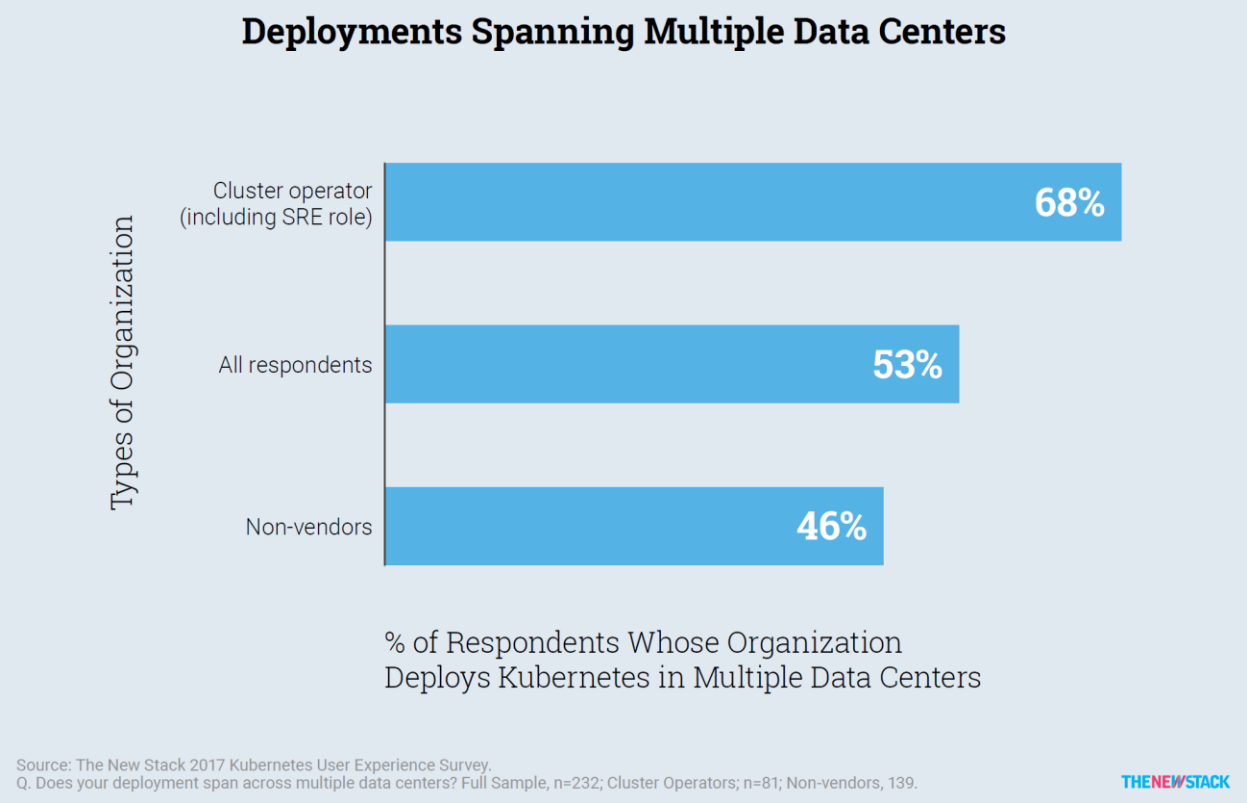
实际上，更大规模的实现还在继续，因为 38%的受访者管理着 5 个以上的集群，而在初始阶段，只有 26%的人这么做。在其他的技术市场中，大型企业通常有更大规模的部署，但在我们的参与者中还没有体现出来。

基于这些反馈，我们可以估计在 Kubernetes 上为所有实现者部署的集群的平均数量为 23 个。

多个物理主机

随着工作负载的范围越来越大，集群的数量可能会反映出增长。管理这种增长的障碍可能是跨多个云和多个站点运行集群的能力。这种障碍可以通过技术进步来克服。另一个可能的障碍是，公司构建了管理大型集群的能力，但是他们的内部开发人员并没有迅速将工作负载迁移到容器。在这两种情况下，集群运营商都将陷入管理未充分利用的基础设施的尴

尬境地。



我们询问了所有的调查参与者，他们的组织或团队的 Kubernetes 部署是否跨越多个物理数据中心，以确定虚拟数据中心是否确实超出了物理数据中心的边界。在我们的受访者中，略多于一半的部署（53%）跨越多个数据中心。不提供云或软件服务的公司不太可能有多站点部署（46%），但集群运营商更有可能（68%）。

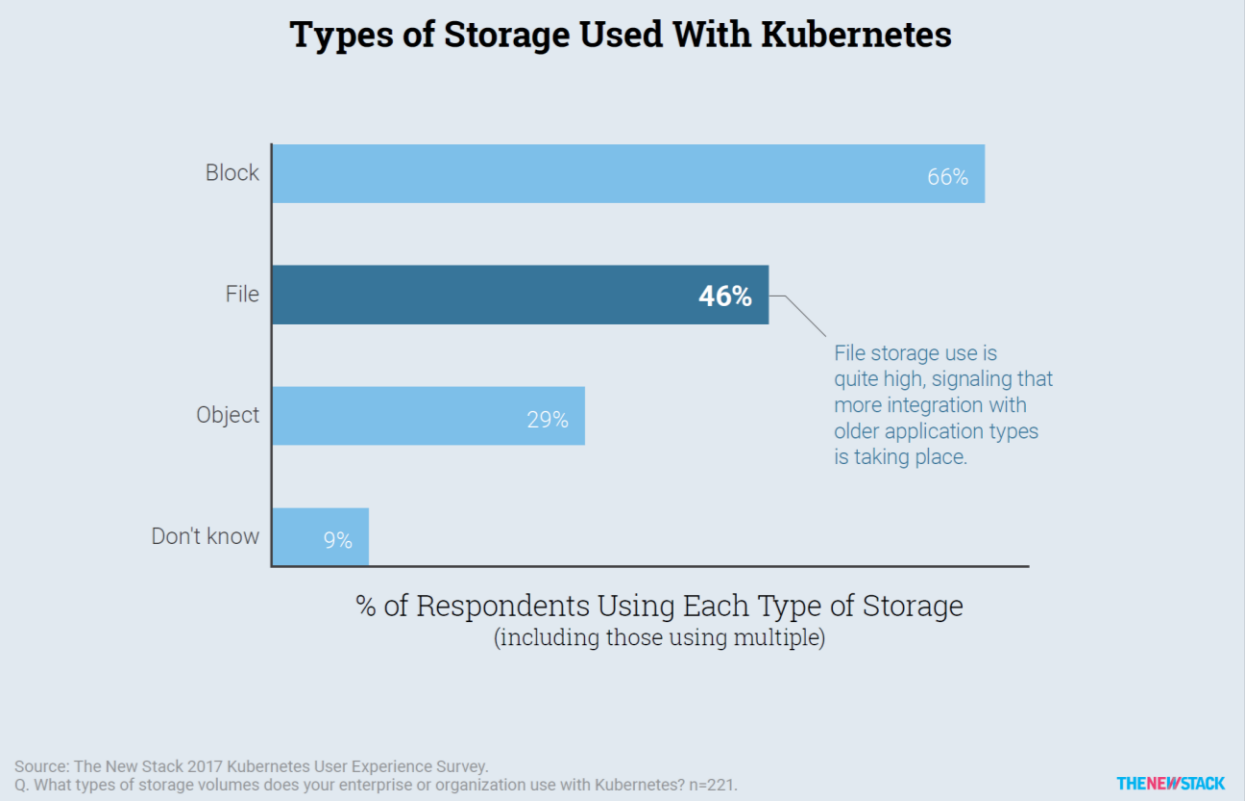
在多个数据中心运营 Kubernetes 的受访者中，97%的公司也运行不止一个集群。将虚拟数据中心的组件分别部署在多个云上可能以后会发生，但现在还没有。

Kubernetes 资源使用

无论组织或团队是否主要在公共云中或主要在本地进行应用程序的开发，显然，他们最喜欢的是与 Kubernetes 相关的资源。存储，网络和监控采用模式可以告知您选择使用哪种工具，或者在您自己的产品中支持哪些工具。

存储系统

在今天的 Kubernetes 部署中使用的逻辑存储结构的类型提供了一些更深入的信息，揭示了正在部署的工作负载的性质。块存储为王，其中三分之二(66%)的受访者认为它与 Kubernetes 实现有关。

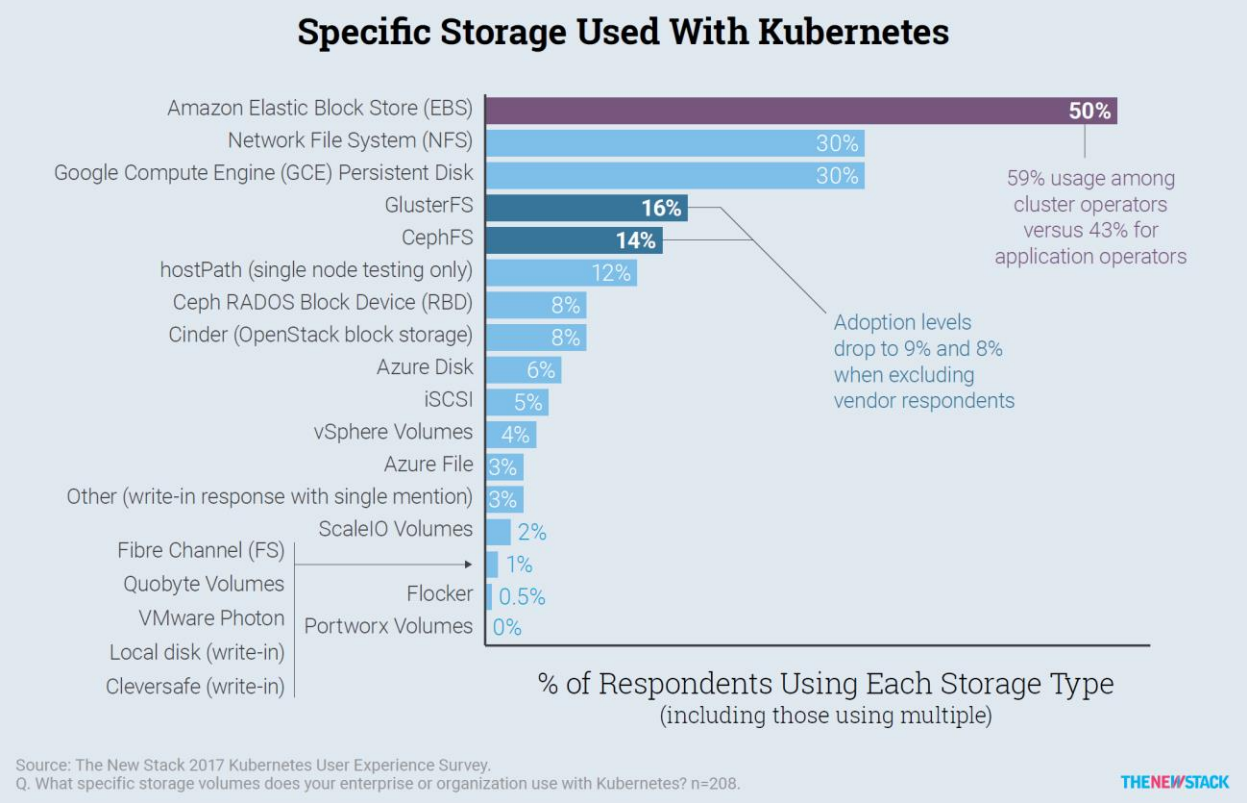


很少有部署只被降级为一种逻辑存储，因此，它告诉我们，只有不到一半的受访者(46%)引用他们正在使用的文件存储。使用微服务架构的更新的云原生应用程序，以及利用数据库或数据结构，通常不需要文件系统，因为它们不通过操作系统与数据进行交互。46%的数字相当高，这表明与旧类型的应用程序的集成正在发生。

29%的受访者使用对象存储，相对于我们过去看到的对象存储的采用率来说，对象存储相对较高。由于对象存储是可伸缩的，开发分布式系统的开发人员可能已经有了这方面经验。 此外，对象存储通常用于为网站提供静态内容，这也是 Kubernetes 的一种常见工作类型。

提供逻辑存储服务

我们给受访者提供了一个具体的逻辑存储系统品牌、项目和技术的列表，并要求引用所有在他们 Kubernetes 部署的逻辑存储系统。



我们给了他们一个机会来写我们没有列出的替代方案。其中一些是商业产品或服务，而另一些则是开源项目。

部分由于其在云市场的强势地位，在 Kubernetes 的所有实现中，亚马逊的弹性块存储被一半(50%)的受访者使用。不可否认的是，谷歌在早期的 Kubernetes 采纳者中所处的地位，可能比其在云空间中的市场份额更有说服力，推动了高百分比的受访者(30%)引用 GCE 持久性磁盘。因此，这些结果并不是一个指标，例如，谷歌拥有亚马逊五分之三的股份。

在我们的团队中，另一个指标，这可能有利于谷歌的倾斜是只有百分之七的受访者使用 Azure-branded 逻辑存储。因此，微软在云市场的份额明显不足。可能的原因是，Kubernetes 只在 2017 年 2 月成为微软的正式产品。随着它的 Azure 技术栈变得更广

泛，它的客户应该有更多的存储选择，除了 Azure 之外。因此，我们预计 Azure 开发人员将比其他云的客户更有可能利用他们直接从一个主要云服务提供商以外的来源购买的存储。

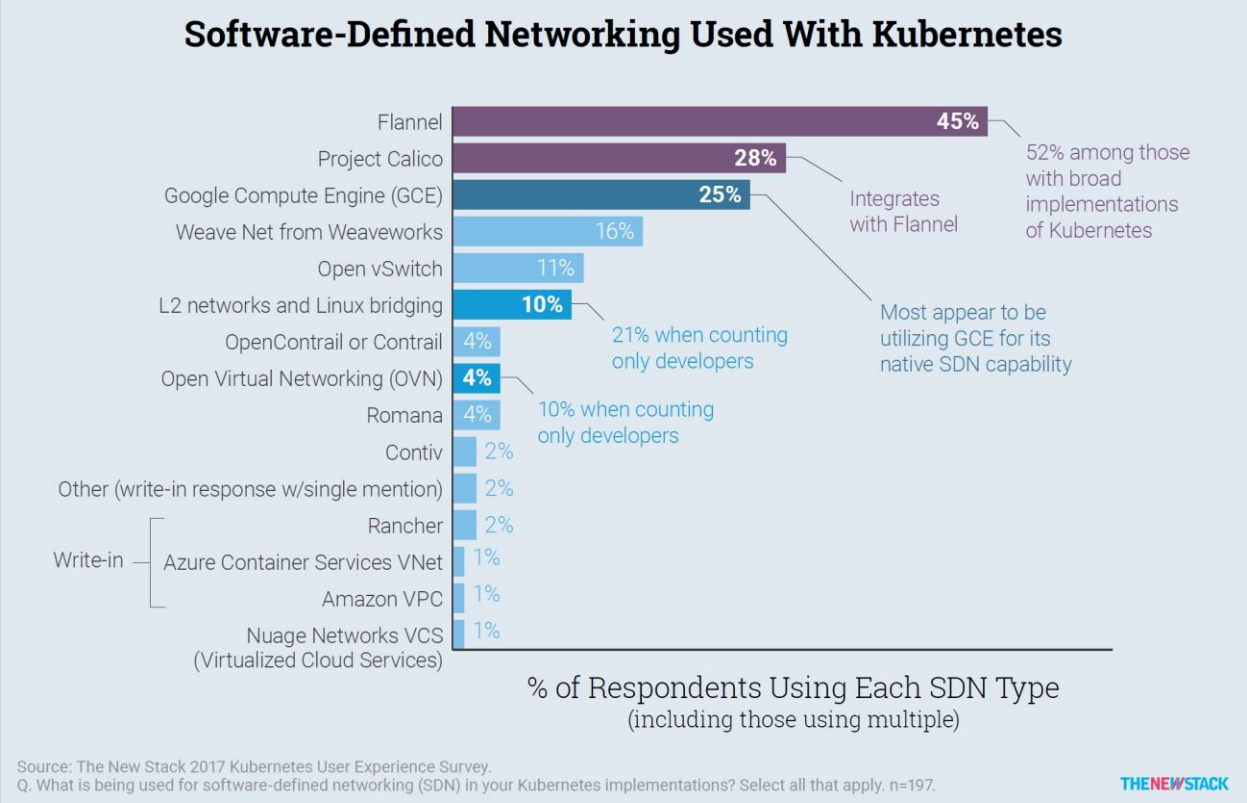
由于集群操作人员通常属于 IT 操作团队，因此在逻辑上，他们更有可能使用内部存储资源。至少在我们的调查参与者中，情况并非如此。大约 59%的集群运营商使用 AWS 存储，但只有 43%的应用程序操作人员（DevOps）也这么做。也许使用 AWS 云让 SREs 专注于基础设施监控，而不需要头痛管理硬件相关的问题。

许多受访者表示，他们使用了一个特定的基于文件的存储系统，其中最常被使用的是 NFS（30%），其次是 GlusterFS（16%）和 CephFS（14%）。然而，后者更经常被供应商使用。忽略这些供应商，剩下的只有 9%使用 GlusterFS，8%使用 CephFS。很有可能许多这样的受访者被 Red Hat 或它的众多合作伙伴之一所雇用。这与我们之前看到的一种趋势是一致的，在这一趋势中，OpenStack 供应商比其他厂商更多采用了 Red Hat 背后支持的存储标准。

特定容器存储解决方案很少被使用。也许随着更持久的工作负载转移到容器中，对专门存储选项的需求可能会增加。另一种可能与便利有关。用户可能更喜欢他们已经知道的存储空间，或者他们最容易访问的存储空间。后面一种解释对于跟随现在不存在的 ClusterHQ 的脚步的那些初创公司来说并不是个好消息。

软件定义网络

接下来，我们向受访者提供了软件定义网络产品，项目和技术的列表，并要求他们列举应用于他们用于 Kubernetes 部署的任何东西。



我们给了他们机会写下我们没有列出的替代答案。

绝大多数情况下，Flannel 是引用次数最多的 SDN 组件，45%的受访者表示引用，其次是 Project Calico 28%，引用 GCE 原生 SDN 工具的 25%。

深入研究这些数据，我们发现 CoreOS 的 Flannel 更有可能被那些广泛实施 Kubernetes 的人（52%的受访者）所使用，而最初的实施者（39%）则相反。Flannel 系统是一个明确用于容器的网络结构，它依赖于每个主机上安装的名为 flanneld 的小型代理。flannel 已经获得了“公正工作”的名声，所以目前还没有任何证据表明它的最高地位受到威胁。

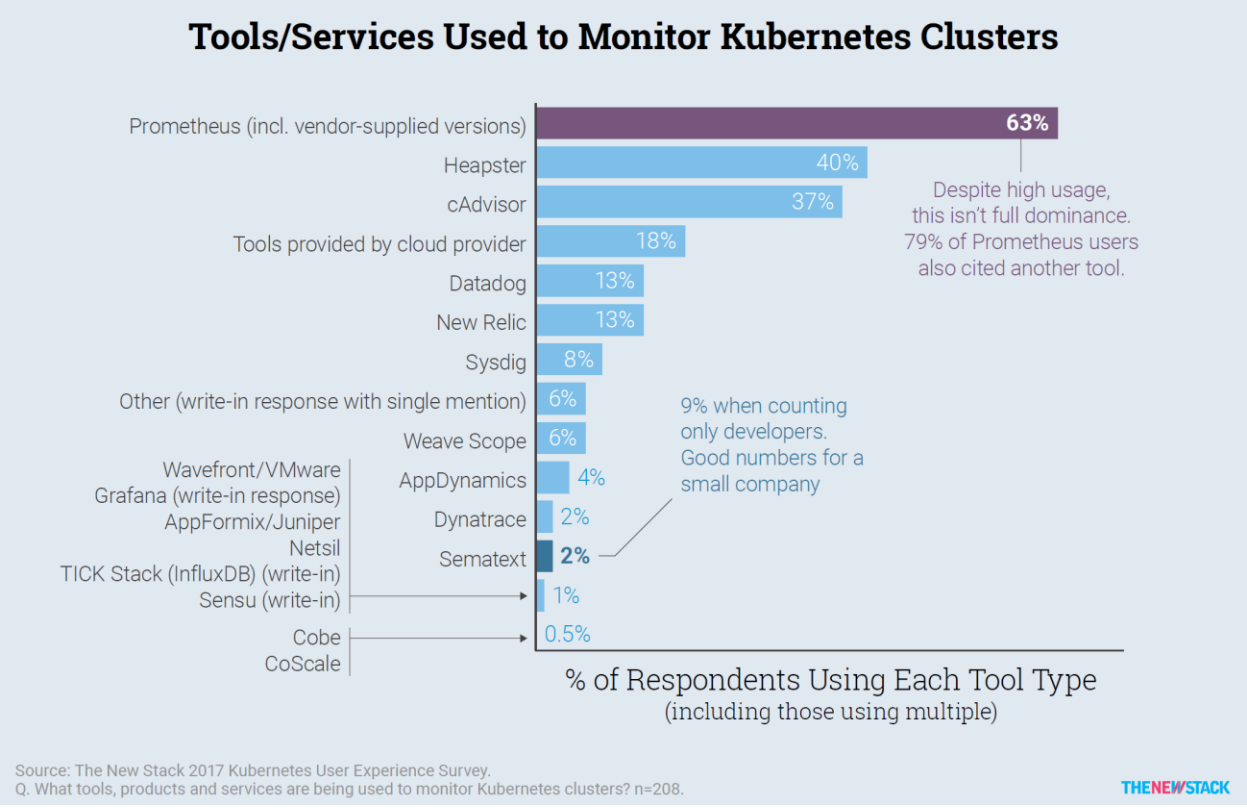
Tigera 公司是 Calico 项目的幕后支持者，该公司一直在研究与 flannel 的集成，因此，在引用 Calico 的人中，有 46%的人引用了 flannel 的说法也就不足为奇了。Calico 项目最近也在集群运营商中取得了成功。在 40 个集群运营商中，有 40%的人说他们的在部署 Kubernetes 初始阶段正在使用 Calico 项目。Tigera 能否通过他们的发展来留住这些用户是一个未知的问题。

在接受谷歌计算引擎的受访者中，有 25%的人似乎都在利用自己的 SDN 功能。一位在 2 到 100 名员工的公司工作的受访者说，GCE 拥有“卓越的容器级别网络...相比 Mesos 和 Weave 链路层网络来说“。如果别人同意，那么用户就不需要在使用谷歌服务的同时拼凑一个 SDN 解决方案。

应用程序开发人员有不同的需求，因此，他们对 SDN 方法的选择遍布地方并不奇怪。使用 L2 网络和 Linux 桥接的开发人员跃升到 21%，而在全部样本中只有 10%。与所有受访者相比，开放虚拟网络(OVN)的使用在开发人员中也更高（10%比 4%）。当然，这表明开发人员更熟悉开发工具。但它也表明，非开发人员可能不熟悉他们的开发人员正在使用的 SDN 工具。

集群监控

一个组织或团队对 Kubernetes 使用的监视工具的选择充分说明了它们的应用程序部署策略。



正如您将在本书中的其他地方看到的，一个组织或团队可能会优先监控其应用程序的基础

架构，反之亦然。 哪一个组织机构选择的目标也与谁进行监控有关：开发人员还是操作人员。

我们向 Kubernetes 用户提供了一长串各种类型的系统监视工具列表，并要求他们选择所有在他们的部署中使用的系统监控工具。我们让受访者有机会写一些我们没有列出的系统监控工具。

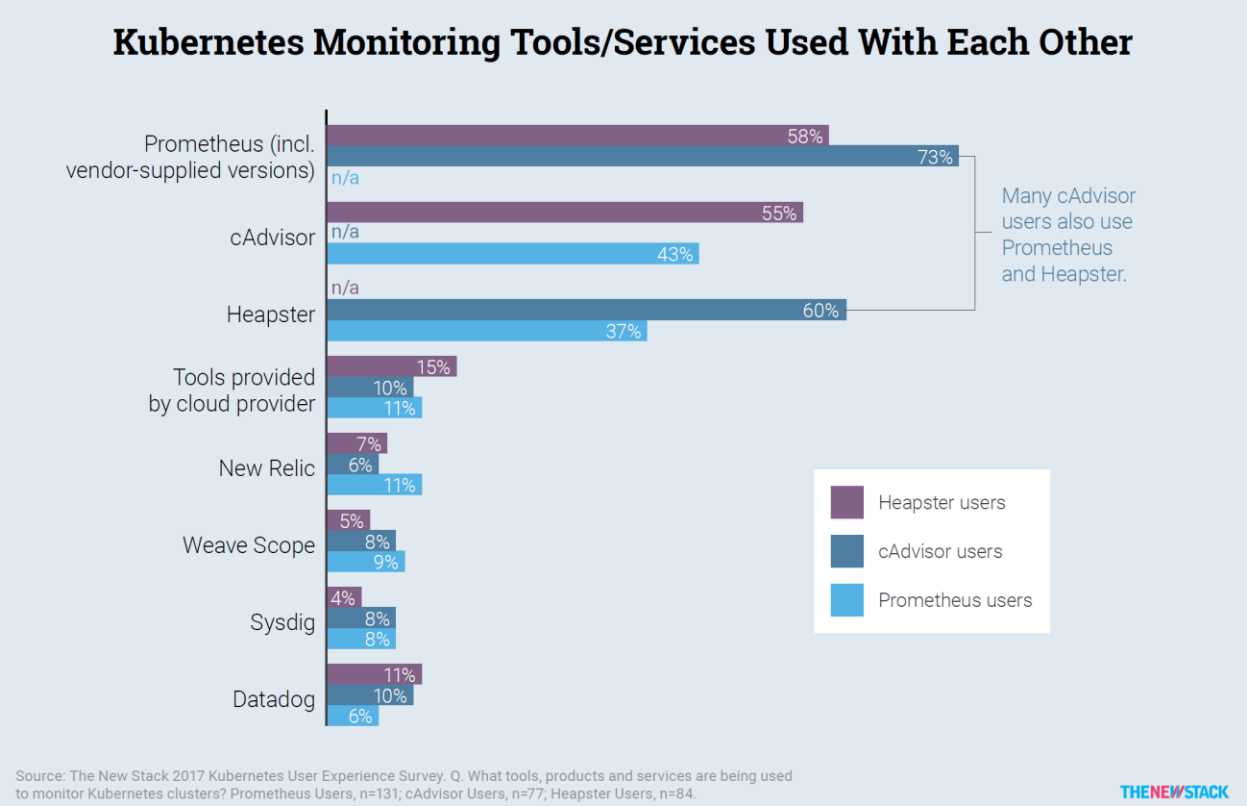
在这些回复中，大约 63%的用户使用 Prometheus 来监控他们的集群，这些服务监控组件的项目是由云计算基础设施所引导的，如 Kubernetes。但这并不意味着 Prometheus 在其他工具和策略上已经取得了主导地位，因为有 79%的人在使用 Prometheus 的同时也采用了另一种工具。现代数据中心的监控策略通常包括有意地提供一种工具的组合，或者将已经在数据中心使用多年的工具拼凑起来。事实上，我们调查的一般组织或团队使用了 2.2 种技术来监控集群。

此外，被广泛引用的是 Heapster（40%），它专门监控 Kubernetes 的资源使用事件；cAdvisor（37%），这是为监控 Docker 容器创建时的资源而使用的监视器。

Sematext 对于应用程序开发人员具有更强的可视性。大约 9%的开发商回应称，他们使用的是它的 Kubernetes 代理，这比全部样本高出 4 倍——这对一个小公司来说是个不错的数字。

组合资源监控工具

Prometheus 项目的领导者可以通过继续与其他工具集成的策略来加强其在数据中心的地位。cAdvisor 的前景尤其广阔，其中 73%的用户也使用 Prometheus，而 60%的 cAdvisor 用户也是用 Heapster。由于 cAdvisor 为其他工具提供数据输入，因此它的集成设计策略非常有意义。



在我们的调查中，DevOps 的专业人员说他们发现了一个影响更小的的方法来管理 Kubernetes，不太会去使用 Heapster 和 cAdvisor。然而，他们也不太可能使用云提供商提供的工具（6%比所有受访者的 18%）。根据定义，DevOps 应该是应用程序和系统监控的集合点。如果确实存在这样的聚合点，那么应用程序操作人员应该关注并权衡他们的组织或团队的监控策略——特别是当它们是从基于供应商的开源选项中拼凑起来的时候。

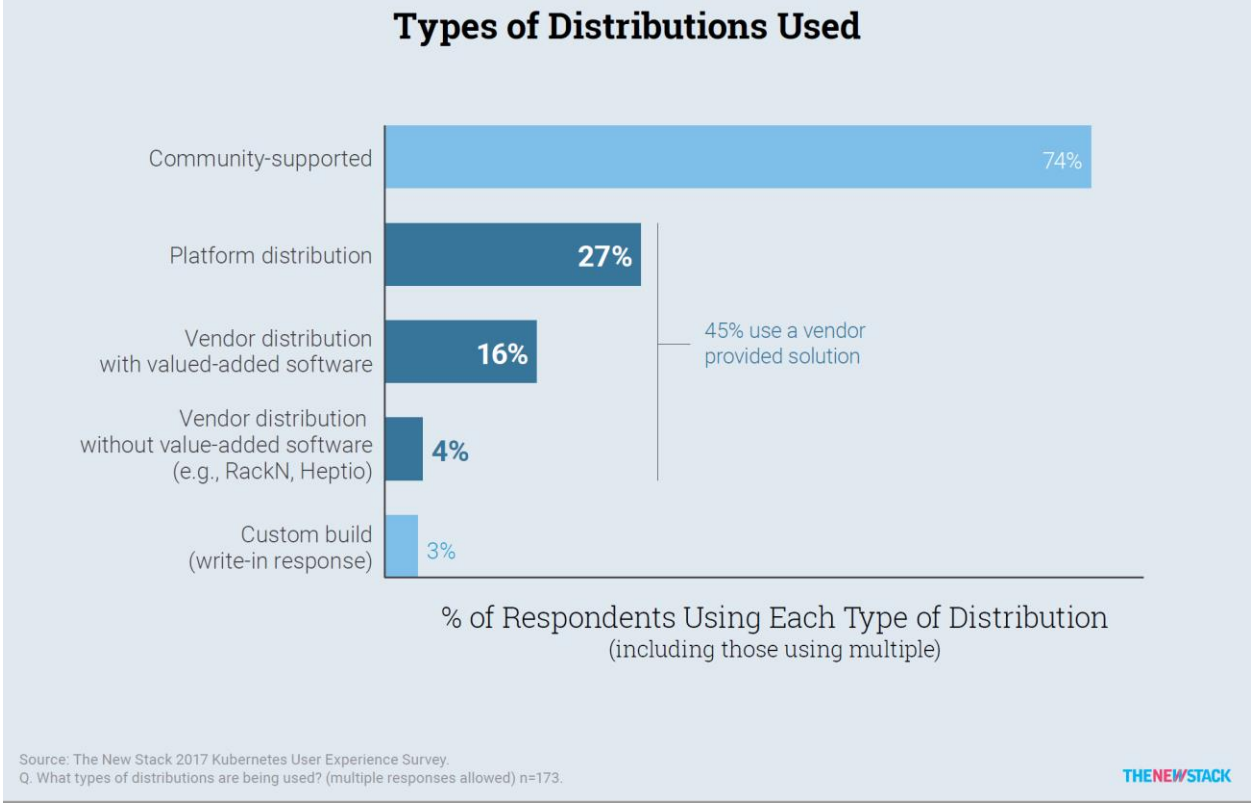
传统的应用程序性能管理（APM）和基础设施监控供应商不太可能被 Prometheus 用户引用。Datadog 监控 Kubernetes 集群，大约有四分之一的非 Prometheus 用户。如果 Datadog 的客户对他们已经使用的监控策略感到满意，他们可能最终不会使用 Prometheus。如果不这样做，现有的用户将会有更多的机会，因为一些用户会查看完整的集群监控解决方案。

社区第一,供应商第二

如果你听过 Kubernetes 的一件事，那就是它的支持社区非常强大。我们在一家中间件企业工作的一名受访者表示：“社区和其他公司的参与似乎是巨大的。” 除了它的技术优势，它的社区的力量是 Kubernetes 甚至有机会在如此广泛的企业范围内迅速采用的一个关键原因。

社区选择与供应商产品

许多观察家认为，为了让 Kubernetes 继续增长，供应商需要提供更多的支持和服务。

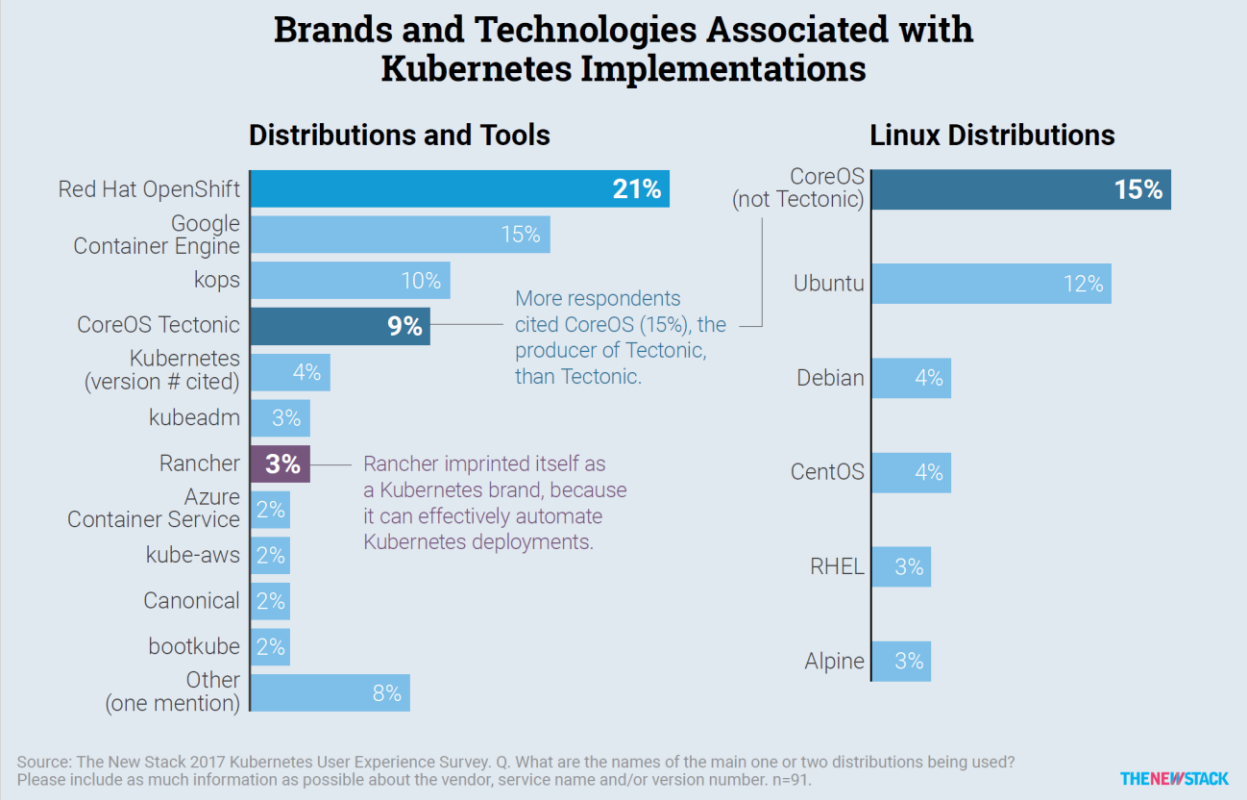


当然，最近几个月我们已经看到了大量基于供应商的选择。 但社区支持的方案的力量依然强大。目前，74%的受访者表示，他们的组织或团队使用社区支持的 Kubernetes 分布——通常是由 GitHub 推出的版本。在这个组中，只有四分之一的人使用额外的非社区支持的分发版。

Kubernetes 根本没有足够的时间来确定组织或团队如何迅速地将社区支持的 Kubernetes 转移到供应商的发行版，甚至是发生这种转变时所考虑的优先级。然而，我们知道有 45%的受访者使用了某种供应商提供的产品。大约一半的人说他们使用一个平台，另有 16%的人说他们使用某一个供应商发行版和附加的其他有价值的软件。少量受访者表示，他们使用的是提供了 Kubernetes 的支持的供应商的发行版。

品牌认知度

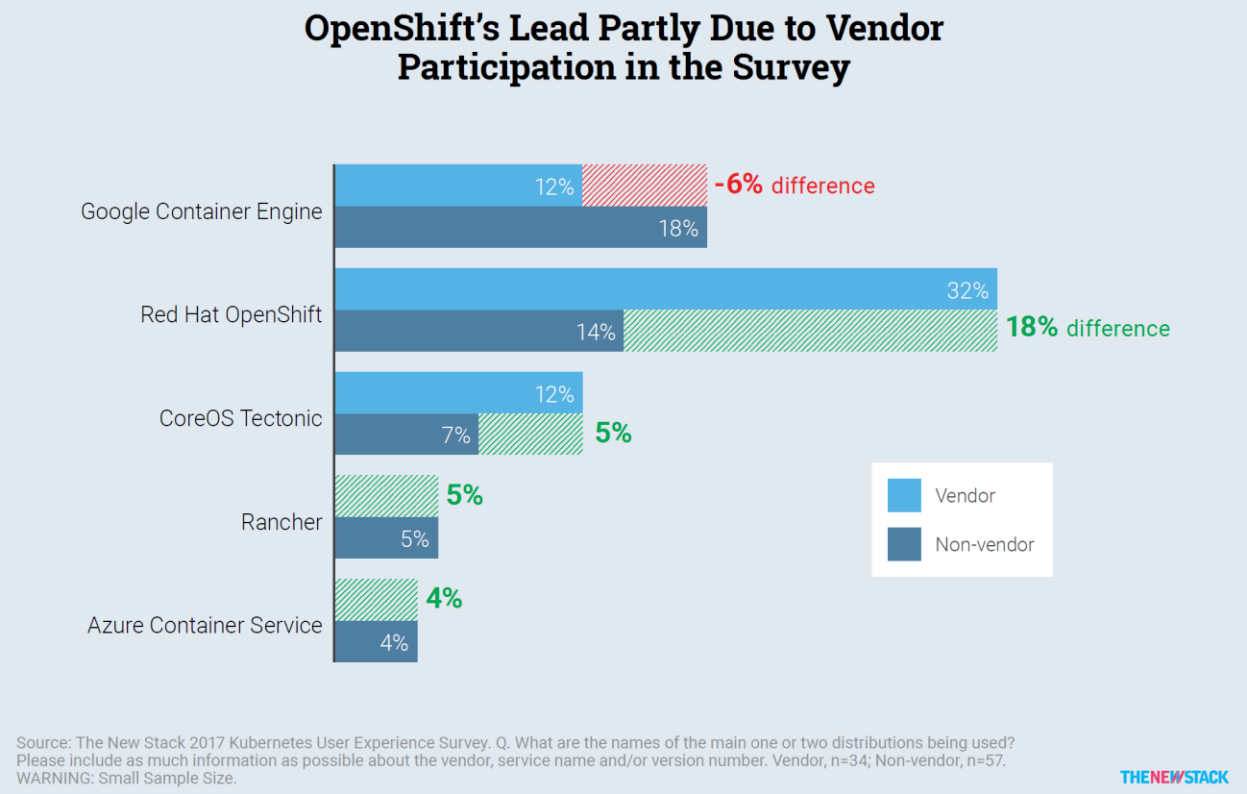
接下来的一项调查将向我们的参与者中掷出一颗手榴弹，看看他们会如何反应。我们要求调查参与者从名单中写下一两个名字，这些名字是他们认为的“发行版”的名字。这是一个措辞模糊的问题，而且是故意这么做的。



我们的目的是确定 Kubernetes 用户与 Kubernetes 之间的联系。可以想象，“发行版”可以指提供 Kubernetes 平台的供应商，或者是使用该平台发布的 Linux 的制造商，或者是协调者运行的云服务提供商。 超过五分之一的受访者（21%）写下了 Red Hat OpenShift，这是这个调查最多的答案。但是，当我们发现，在那些被供应商雇佣的参与

者中，有 32%的人提到了 OpenShift，相比之下，在不为软件供应商工作的人中只有 14%。

进一步的分析结果，我们确定 19 名被调查者中只有 12 人认为 OpenShift 是基于供应商的增值功能。

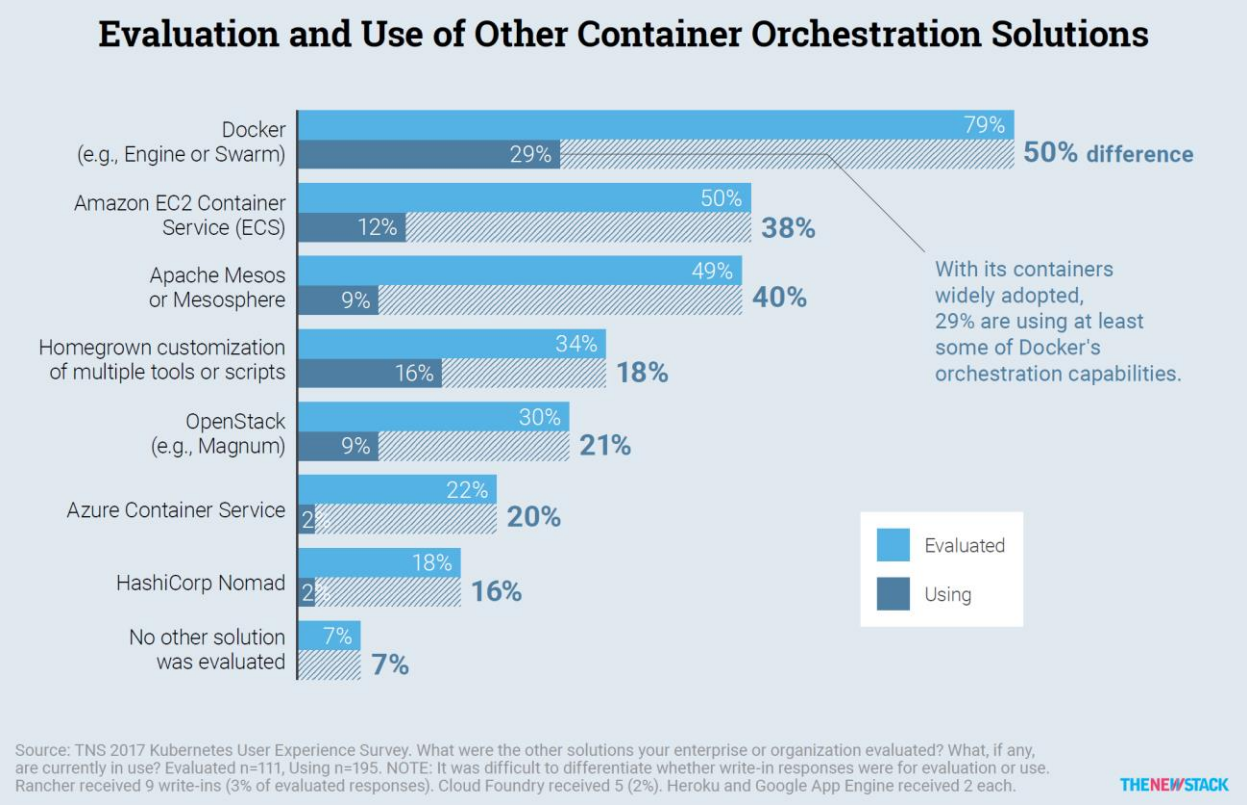


许多受访者将他们的 Linux 发行版写成 Kubernetes 提供商。事实上，即使 CoreOS 是 Tectonic 的生产者，还是有更多的受访者写下 CoreOS(15%)，比 Tectonic (9%)要多。大约 10%的受访者提到了 kops，这实际上是 Kubernetes 的一个供应和安装工具。使用社区支持的 Kubernetes 的受访者经常期待 kops 为他们提供最新的版本。

Rancher 是由 Rancher Labs 生产的一个成熟的容器管理平台。虽然它不包括 Kubernetes，但它可以非常有效地自动化部署 Kubernetes。约 3%的受访者认为 Rancher 是我们所要求的 Kubernetes“发行版”中的一员。因此，在这些人中，Rancher 将自己塑造成一个成功的 Kubernetes 品牌。

评估 Kubernetes 和竞争对手

人们不会无理由的选择 Kubernetes。93%的受访者告诉我们，他们的组织或团队在做出选择之前已经评估过 Kubernetes 的替代方案。



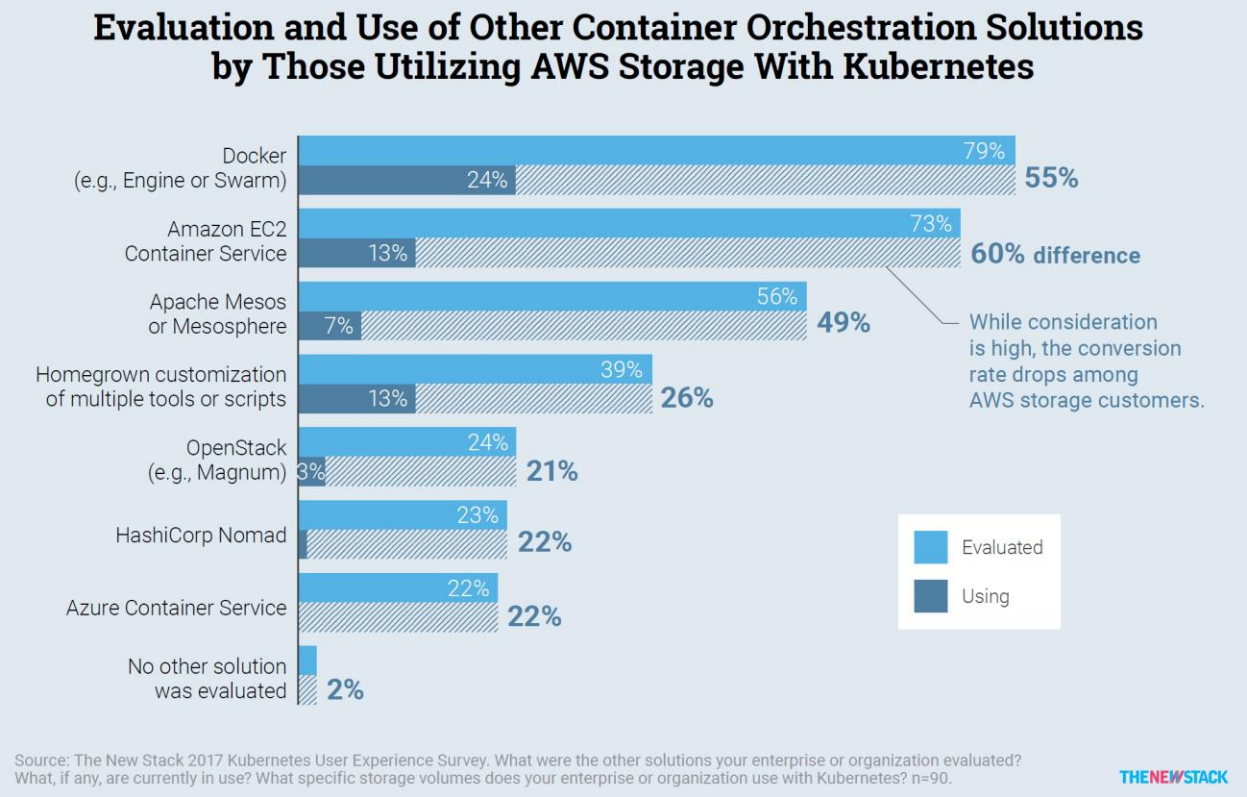
开放的替代品

更重要的是，整整 45%的受访者告诉我们，他们同时利用了其他应用平台。在那些没有使用过 Kubernetes 的人当中，有 61%的人说他们正在使用一种有竞争力的产品。

用于管理这些新的分布式环境的工具的工程仍在进行中。因此，Kubernetes 的竞争对手仍有一线希望。显然，市场本身缺乏定义，甚至许多 Kubernetes 用户也在考虑其他选择。

竞争对手观点

我们给调查对象一个列有七个容器平台的列表，加上一个没有选择的选项，并要求他们选择他们的组织或团队目前正在评估的以及他们目前正在使用的所有平台。我们看到了一组有趣的结果，由这张图表显示，在被调查者中，他们还表示正在把 Amazon AWS 存储服务与 Kubernetes 部署在一起使用。



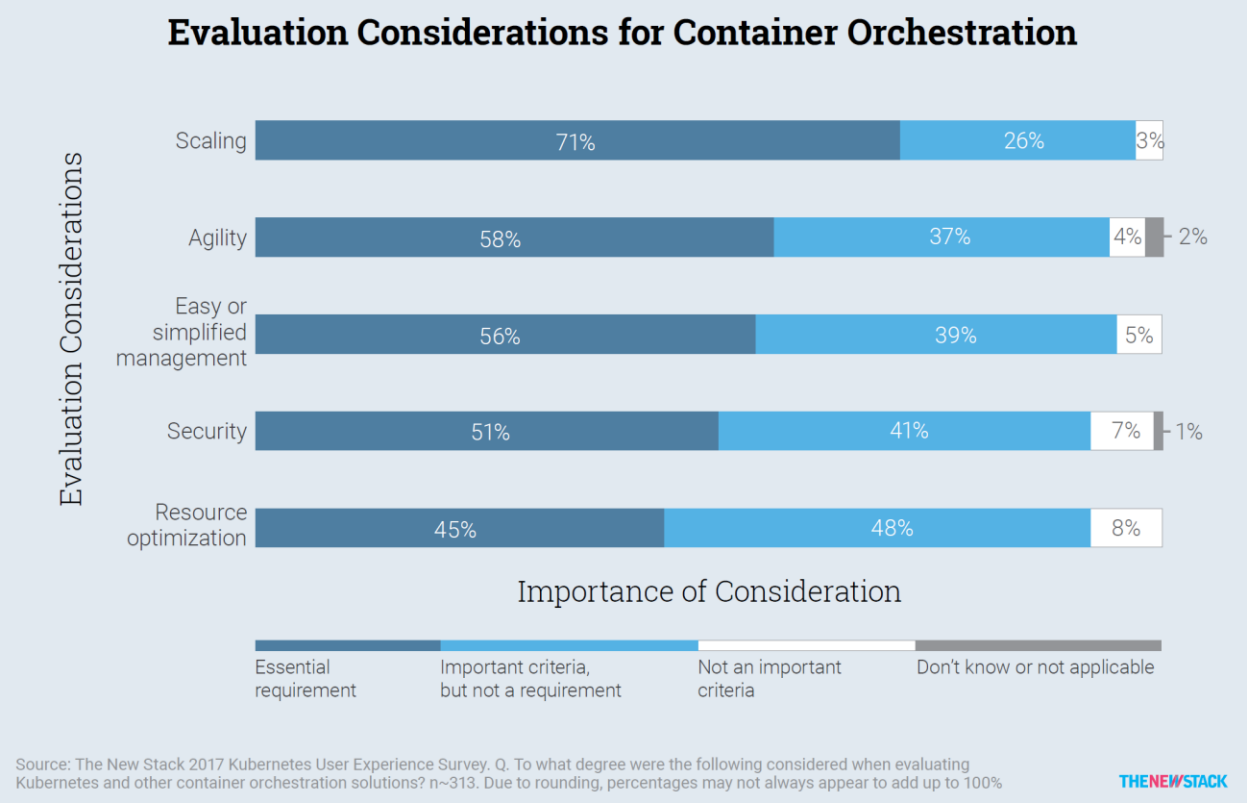
我们在 The New Stack 从书中曾经说过，Kubernetes 的兴起是由 Docker 所发起的革命实现的。2016 年，Docker 公司开始将 Swarm 编排功能绑定到 Docker 引擎上。因此，各个组织或团队有充分的机会评估 Swarm 的假设是否合理。

在对我们的调查做出回应的 Kubernetes 评估者中，有四分之三的受访者（73%）使用 AWS 存储，而 Kubernetes 则对 Amazon EC2 容器服务进行了一些考虑。这些人中很少（18%）最终使用了 ECS。更高层的考虑主要是 AWS 的整体基础设施作为服务（IaaS）的市场份额。在非供应商的受访者中，ECS 和 Mesos/Mesosphere 的使用率都略低。以下是由雇佣 100 名或更少员工企业的调查参与者的一些评论，他们对这两个平台进行了评估:

- “在 AWS 上与 K8s 进行整合是很困难的。一年半之后，一些工具就像 kube - aws 和其他类似于 kops 的工具一样，在每次更新集群时仍需要破坏集群。如果你想在 AWS 上使用 K8s，并且是安全的，那么选择正确的工具是非常重要的。
- “AWS 需要将 K8s 集成为像 GCP 这样的服务；否则，我们会迁移到别的平台上。”
- 如果您想编写自己的调度器，那么使用 Mesos 是非常痛苦的：您必须在同一版本的 libmesos 中编译您的调度器。然而，您正在运行的 libmesos 不能快速或轻松地编译。

决策标准

现在我们知道了我们的调查参与者在考虑哪些平台，让我们来看看被调查者使用哪些标准来评估容器编排服务。我们给参与者列出了五个评估编排平台标准的列表，并要求他们按四分制打分，从“基本符合”到“不适用”。



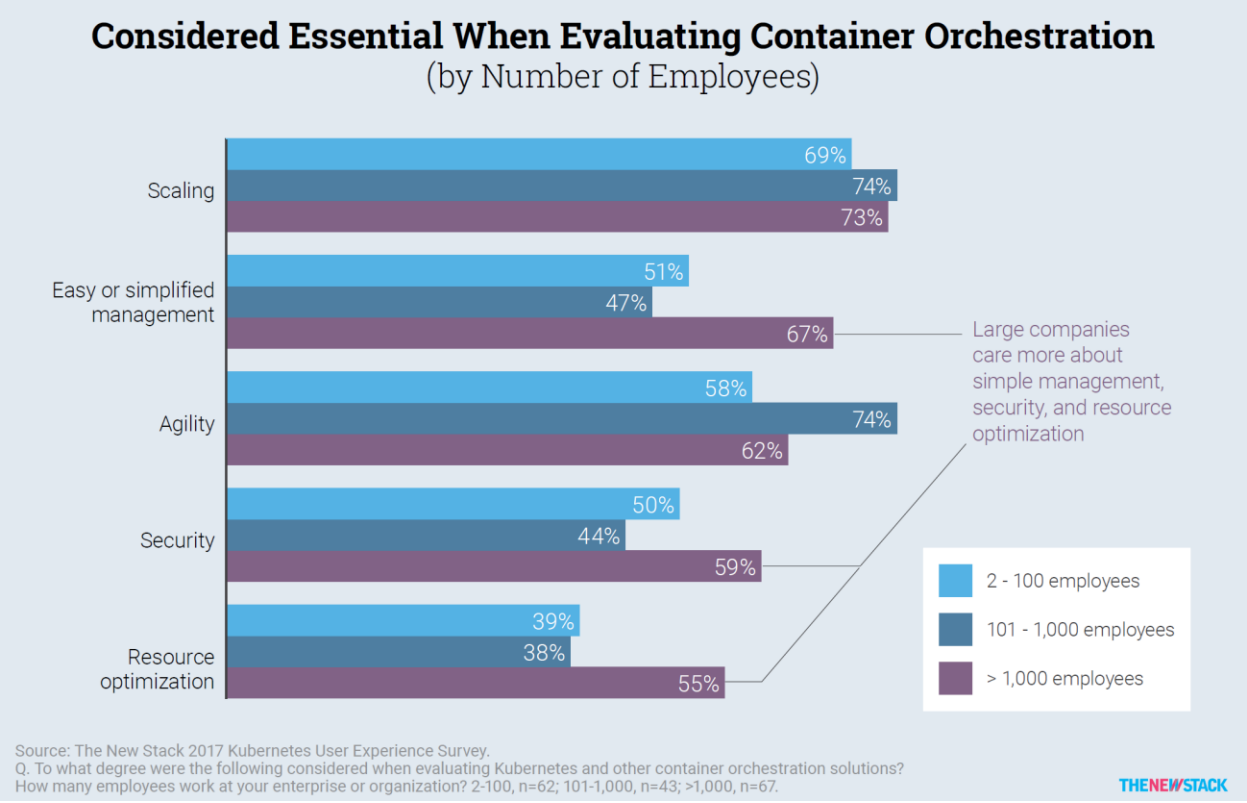
绝大多数受访者认为这 5 个因素都是重要的标准，只有资源优化“基本符合”低于一半(45%)。有 71%的受访者认为，伸缩的能力是必不可少的。

目前评估 Kubernetes 的人比使用它的人更有可能将安全视为一项基本要求——65%的是评估人员，而只有 45%的是用户。以下是来自用户那一列的两个人的评论:

- “需要一个安全的——真正安全的，而不是目前可用的或提出的方法——在集群应用程序和其他应用程序之间访问和管理数据。” [软件公司，2 - 100 名员工]
- “我们运营着一个相当大的网络商店（每年有 1 亿欧元的收入）。我们的应用是整体的，我们需要一种方法来扩大需求。K8s 提供了可伸缩性需求的基础。” [101 - 1000 人]

分解决策标准

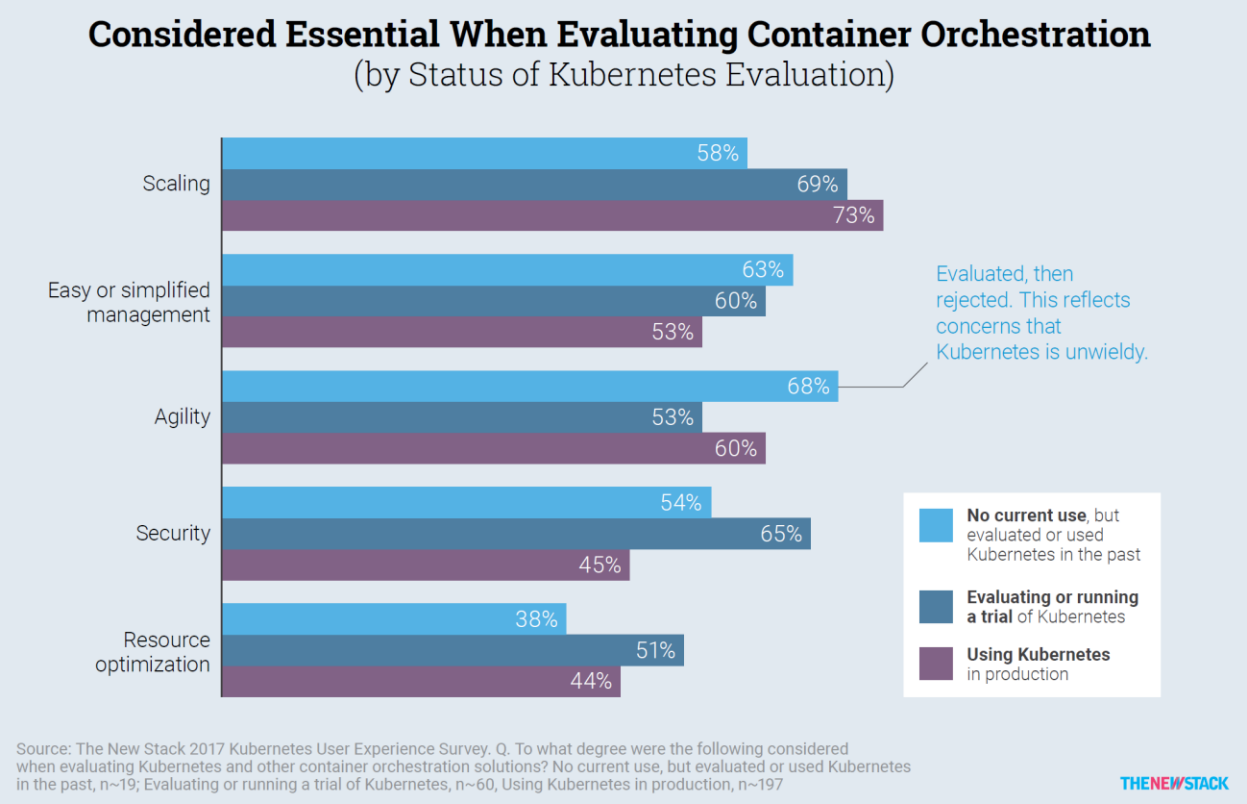
当我们在回答被调查者的雇主的决策标准问题时，我们发现他们的模式存在一些惊人的明显差异。



大型公司（超过 1000 名员工）比中小企业更重视简单的管理、安全和资源优化。特别地，简单或简化的管理在大型公司得到了 67%的受访者的支持，而其他较小规模公司的平均比例为 53%。中等公司（100 - 1000 人）的受访者认为敏捷（74%）和伸缩（74%）同样重要。

虽然简单的管理不太可能成为小公司的核心要求，但许多受访者在他们的评论中表示，Kubernetes 很容易使用。他们使用了它的 api，它的可插入功能，以及它支持在多个云中进行部署的特性。

接下来，我们以当前的 Kubernetes 部署状态对决策标准问题的回复进行了分析。在那些进行过评估并拒绝使用 Kubernetes 的少数受访者中，68%的人认为敏捷是一项基本要求。这可能反映了他们对 Kubernetes 有点笨拙的担忧，这也是我们在 IT 社区和会议上听到的观点。这仍然是一个年轻的项目，所以人们不那么惊讶于文档的不完整以及并不是所有的事情都在开始的时候就能正常工作的抱怨。



下面是与我们分享的一些关于 Kubernetes 的评论:

- 有很多快速的变化，这使得我们很难“跨越”部署一个非常大的环境，因为你必须不断进行调整。[101 - 1000 人]
- “痛点。关于安装和操作的文档很差。”组件的快速变化使之成为最佳选择。不同云提供商之间的选择大相径庭。但基于有限的信息很难进行评估。“(> 1000 名员工)

- “这些都不是 100%的生产环境。K8s 本身似乎在规模和生产上都有几个问题。在 K8s 中容器/ pod 中的应用程序需要考虑环境，在移植时需要进行调优。在大规模集群事件中，许多 K8s 组件在规模上并不能证明其健壮性，在某些情况下可能导致服务中断。这种处境需要进一步的改进和优化，经常是很棘手的，导致需要不断地修补和升级以获得必需的特性和修复。 “[> 1000 名员工]
- “有大量的宣传说这仅仅是很酷的东西，不值得那么多工程师付出的努力。Kubernetes API 中有一大批功能没有通过 OSS 命令行工具暴露出来，而且通常不具备完善的文档（如 RBAC）。如果您的目的是为了销售 SaaS 产品而赚钱，这就已经足够了。 我打赌，Kube 不会变成另一个‘不该被命名的编排系统’（*咳嗽* OpenStack *咳嗽*）。：）“[未知的公司规模]
- “很多方法做错了，没有产品级别的设置指南。 简单的工具留下很多需求。 自动化工具不适合生产。 “[2 - 100 名员工]
- “Kubernetes 充满了错误，并不是每个人都喜欢认为其是应用交付的最佳途径。” [101 - 1000 人]
- “其实 Kubernetes 是很难驾驭，我开始没意识到这些。当离开一个专家和帮助者，你的团队开始使用你所建立的集群，你就会明白我在说什么。” [101 - 1000 个名雇员]
- “需要更多的人员来操作。 由于这些平台的快速变化，需要更多的人员来保持一切更新和有效运作。 “[101 - 1,000 名员工]

使用 kubernetes 原因

除了有一般的要求，调查的参与者告诉我们为什么选择了 Kubernetes。 许多人在其关键指标中引用了它的普及性。 一位评论者对这种受欢迎程度表示了负面的评价，他说很多人“喝多了 Kubernetes Kool-Aid（Kool-Aid：一种流行饮料）” 。有些评论者强调了社区的力量，还有人赞赏来自 Google 和 Red Hat 等知名公司的强大支持。

一个新兴的主题是，Kubernetes 技术上至少在目前是领先的。 正如一位受访者所解释的：

- 它是第一个以容器为中心的编排器，提供构建微服务风格应用程序所需的全套功能。 服务发现与其他编排器相比处于领先地位，并且具有配置管理功能。 其他编排器似乎正在赶上。 “[2 - 100 名员工]

另外一些人似乎期望 Kubernetes 能够凭借其在技术上的竞争力，在编配市场保持目前的优势。 正如一些人告诉我们的：

- “Kubernetes 真棒。 Azure 很好地掌握了 K8s 在哪里先进，但是他们的 PVC 实现很糟糕，落后了。 GKE 是那里最强的实践。 AWS 落伍了。” [101 - 1,000 名员工]
- “Mesos 不提供编排层，除此之外，运行 Mesos 的开销是巨大的，不存在平台的稳定性和可扩展性。 由于 Cloud Foundry 是一个“单一供应商”开源项目，没有明确的使命，因此被淘汰了，Pivotal 似乎正在向 K8s 本身迈进。除此之外，Cloud Foundry 仅允许托管两层应用程序，因为受限于其组合模型严格使用应用程序代理绑定模型。 三大云提供商都没有提供在单个托管服务中捕获的足够好的模型。 Google 云的距离最近 。OpenShift 只是封装有限的 K8s 附加价值，但需要有一个额外的成本。 因此，我们使用 K8 的开放源代码，自行托管，将基础架构和 K8s 自动化部署到我们需要的所有平台。 “[100 - 1,000 名员工]
- “速度，可扩展性和安全性对我们非常重要。 我们从早期版本的 Docker 与 Consul 开始的自行开发的解决方案，通过桥接网络进行服务发现。 事情对我们来说并不容易。 原生解决方案难以扩展，维护是一场噩梦。我们玩过 Apache Mesos，但是这不符合我们的需求。 当 Docker Swarm 发布时，我们已经有了运行 K8s 的集群。 Docker Swarm 没有正确实现我们已经在 K8 中使用的功能。 “[101 - 1,000 名员工]
- “容器编排确实很难做到：）。 大多数解决方案要么太复杂（例如 Mesos 和 DC / OS），要么工作不正确（Rancher，Kontena 和其他小型玩家）。 像 Cloud Foundry 这样的“企业”解决方案也是不必要的复杂。 我们在物联网领域工作，我们还有一个容器编排解决方案的标准：应该支持将系统移动到裸机。 这就是为什么我们现在使用 Nomad 来做物联网，但是可能在将来对 K8s 进行评估。 “[2 - 100 名员工]
- “不是所有平台在能力，稳定性和规模上都是平等的。 VMware 错过了其他基础设施提供的核心服务。 OpenStack 不等于在版本和发行版之间的 API 兼容性方面

> Open Stack（意思存在缺陷）。 Google Cloud 和 AWS 是最稳定的；Azure 正在赶上来。 台面下面有很多的多样性，这就造成了风险。 K8s OSS 没在所有平台上进行测试，所以我们运行我们想要采用的架构进行验证，构建我们自己的版本，包括我们需要的更改并通过测试。 没有供应商提供跨平台服务！ Heptio 是一个从未实现过的承诺，太糟糕了。 “[101 - 1,000 名员工]

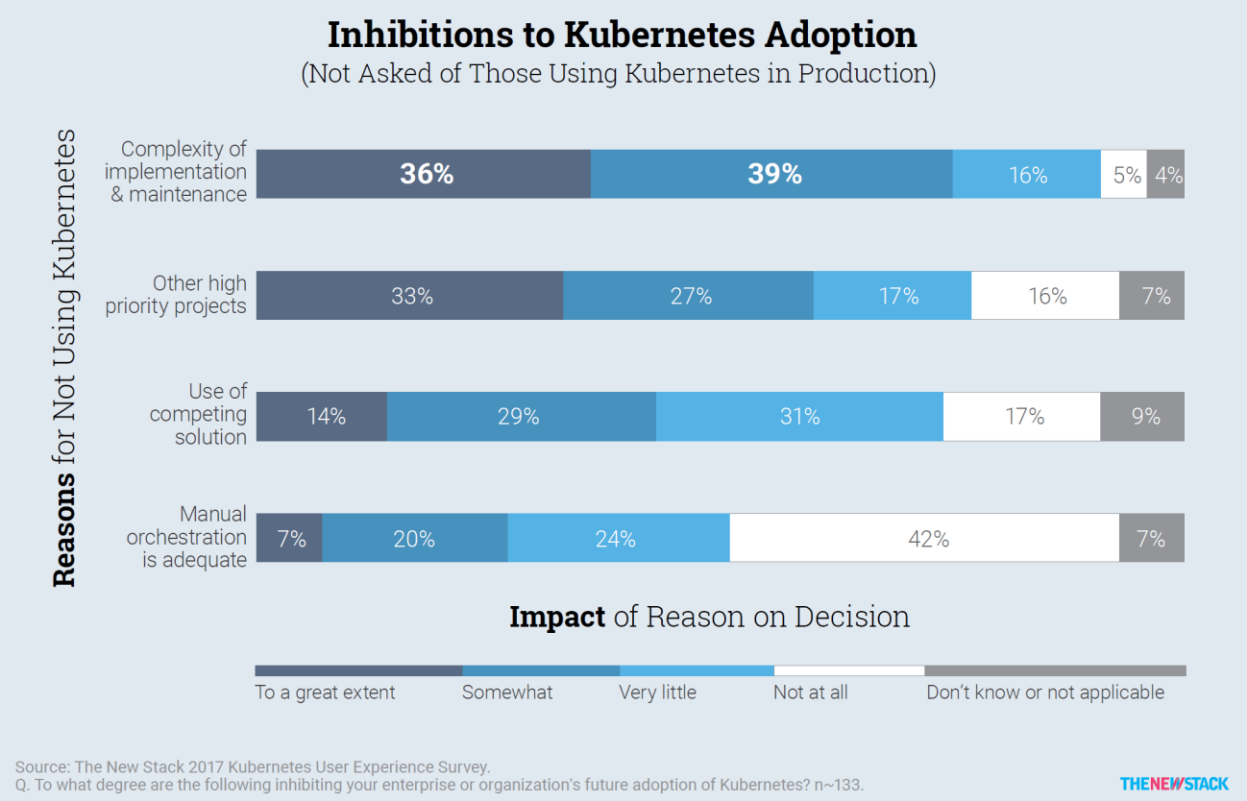
采用 Kubernetes 的障碍

Kubernetes 在容器编排市场上的优势，就像今天容器编排市场一样，既不是明确的也不是确定的。它的生存还可能取决于竞争对手满足客户期望的对编排基本要求的能力。未来，企业可能会寻求与大型平台捆绑或提供的解决方案，或者只要他们发现已经与他们投资的平台捆绑在一起，就可以简单地接受这些解决方案。

Kubernetes 开发社区需要解决它被采用的阻碍因素，特别是那些尚未做出评估的评估者遇到的阻碍因素。

负面因素和障碍

正如我们所看到的那样，对于过去评估 Kubernetes 的人来说，敏捷性更重要，但是选择了另一条路径。我们明确地询问了这个小组，以及目前正在评估 Kubernetes 但尚未投入使用的小组，这是什么阻碍了他们的使用。

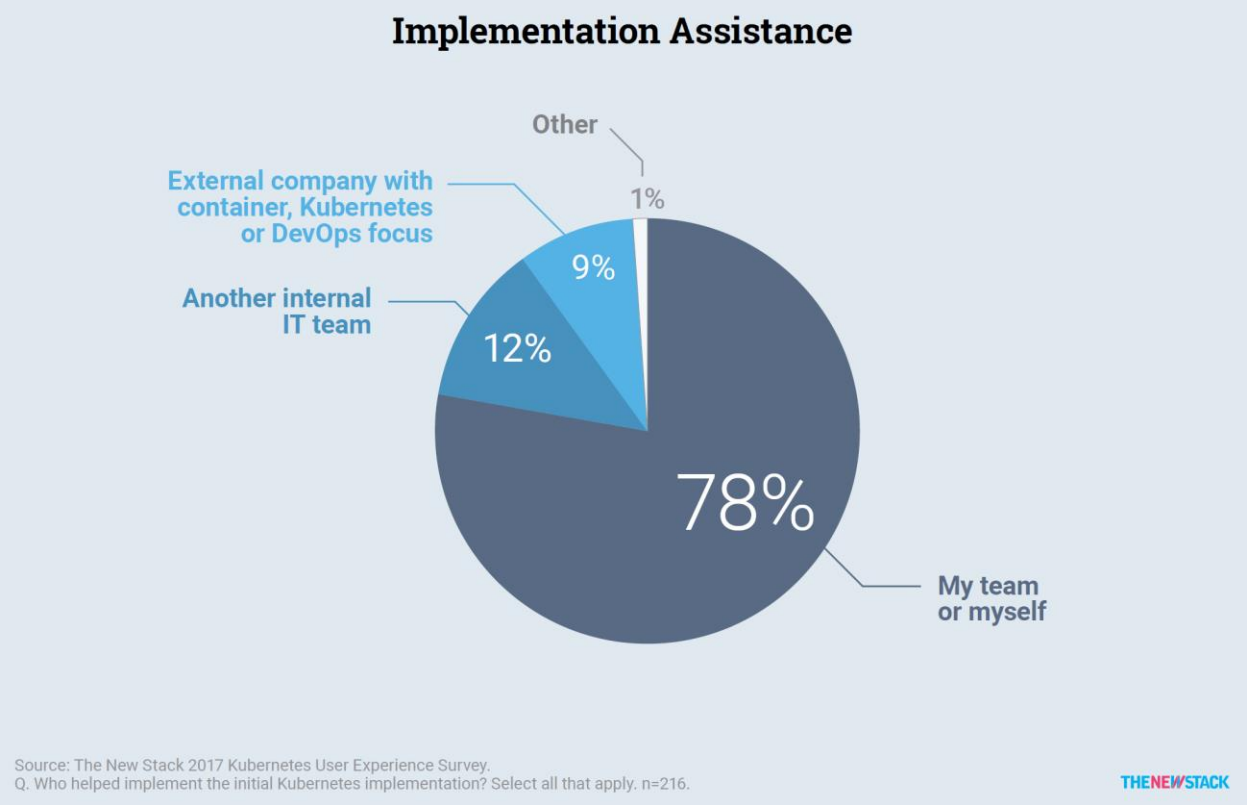


我们给这些小组列出了四类影响采用的障碍列表，并要求他们以五分的比例给每一组打分，从“很大程度上”到“不适用”。在“很大程度上“，Kubernetes 实施的复杂性阻碍了 36%的受访者的使用。在这些组中，有三名受访者认为复杂性在一定程度上是一种重要的障碍。同样，对复杂性的担忧在那些评估和拒绝 Kubernetes 的组织或团队中更为明显。 另一个大的障碍仅仅是其他项目的高优先级。五分之三的受访者表示，其他地方的高优先级项目是一个重要的问题，三分之一的受访者认为 Kubernetes 的低优先级是最高的。对于这些公司来说，Kubernetes 的采用可能需要等到其他组织或团队成功的时候。 另一个障碍是”不太可能被使用“。受访者明确表示，他们认为手工编排不是选择的一种主要障碍。只有 7%的受访者认为手工编排是他们的主要障碍。在那些不使用 Kubernetes 的组织或团队中，42%的人认为手工编排不是一种障碍。只有百分之十四的人认为他们的组织或团队使用竞争性解决方案作为是一个重要的使用障碍。

实施：是好，是坏还是差强人意

我们已经从 Kubernetes 收到的好评，尤其是之前拒绝在生产环境中部署的机构中得出此结论。那么这些分享好评的组织或团队是否已经完全接受了 Kubernetes?

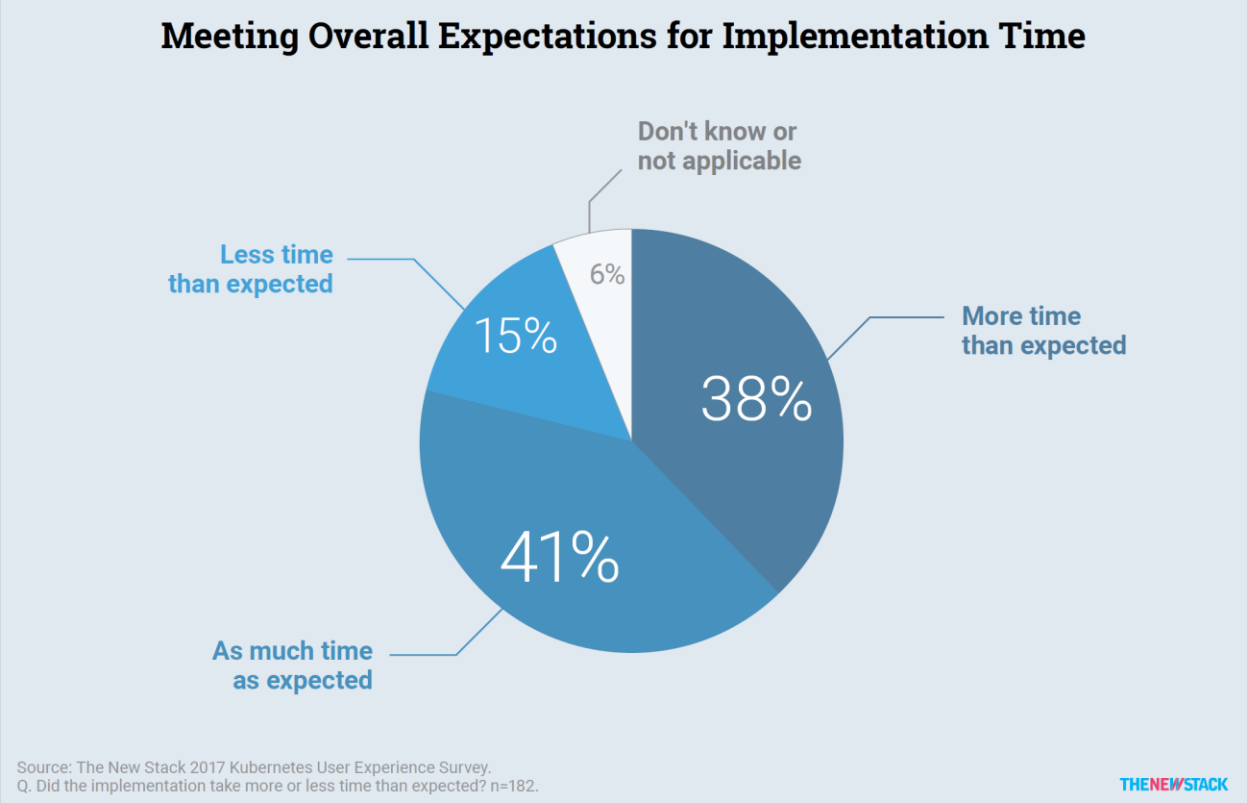
我们问了采用方一些问题得出这些结论。为了让您更好地了解这个分组与实施的实际关系，大约 78%的受访者告诉我们他们或他们的团队直接负责实施 Kubernetes，如下图所示。



实施过程耗时太长超过预期

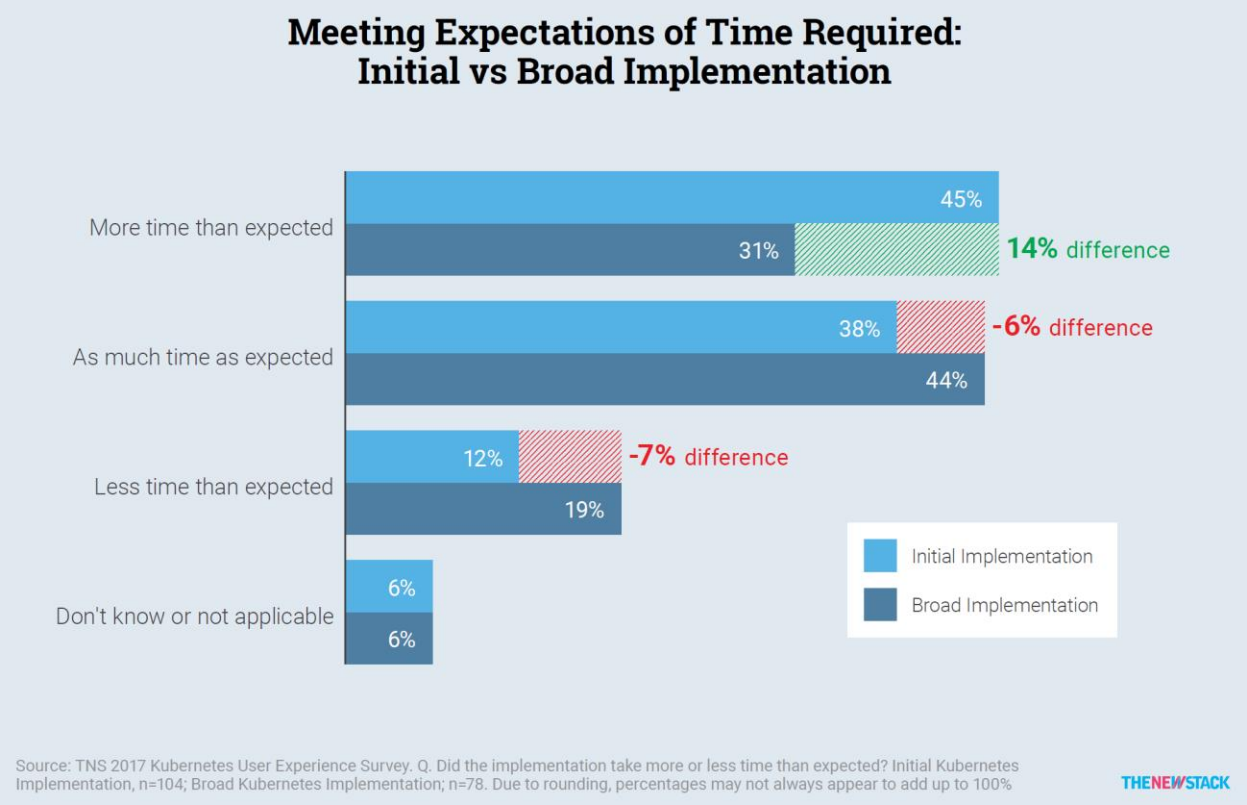
我们询问了这些实施者，他们的组织或团队实施 Kubernetes 是否超过了最初估计的时间，或者是多少时间。 约 38%的受访者表示花费的时间比预期的要长，而只有 15%的受访者表示花费更少的时间。 仅仅是这部分，就已经显示出了部署的复杂性。

对于那些说实施 Kubernetes 耗时超过预期的受访者而言，百分比的降低则源于他们在实施过程中经验的提升。在耗时更多方面，尤其那些处于广泛实施阶段的会比那些刚刚开始实施阶段的低 14 个百分点。这会给很多实施者一些良好的建议——尽管不是全部——一旦他们开始实施，就可以放心享受 Kubernetes 时间了。



一位在小型业务部门评论者给出了令人期待的结论：

- “之所以花费更多的时间而超过预期，是因为部署和管理 Kubernetes 集群的复杂度，以及一年都未更新的文档说明所导致。今天这些已经很容易了。”



即使安装和配置阶段可能会导致拖延，一些受访者这样解释，组织或团队的动态性才会造成最严重的时间拖延。这就有一些例子：

- “团队和流程上的障碍难以克服。你在公司范围发起了只有一次机会的事情。”[大于 1000 名雇员]
- “我们低估了要在全员推广持续集成的工作量，但我们必须这样做下去。”[101 ~ 1000 名雇员]

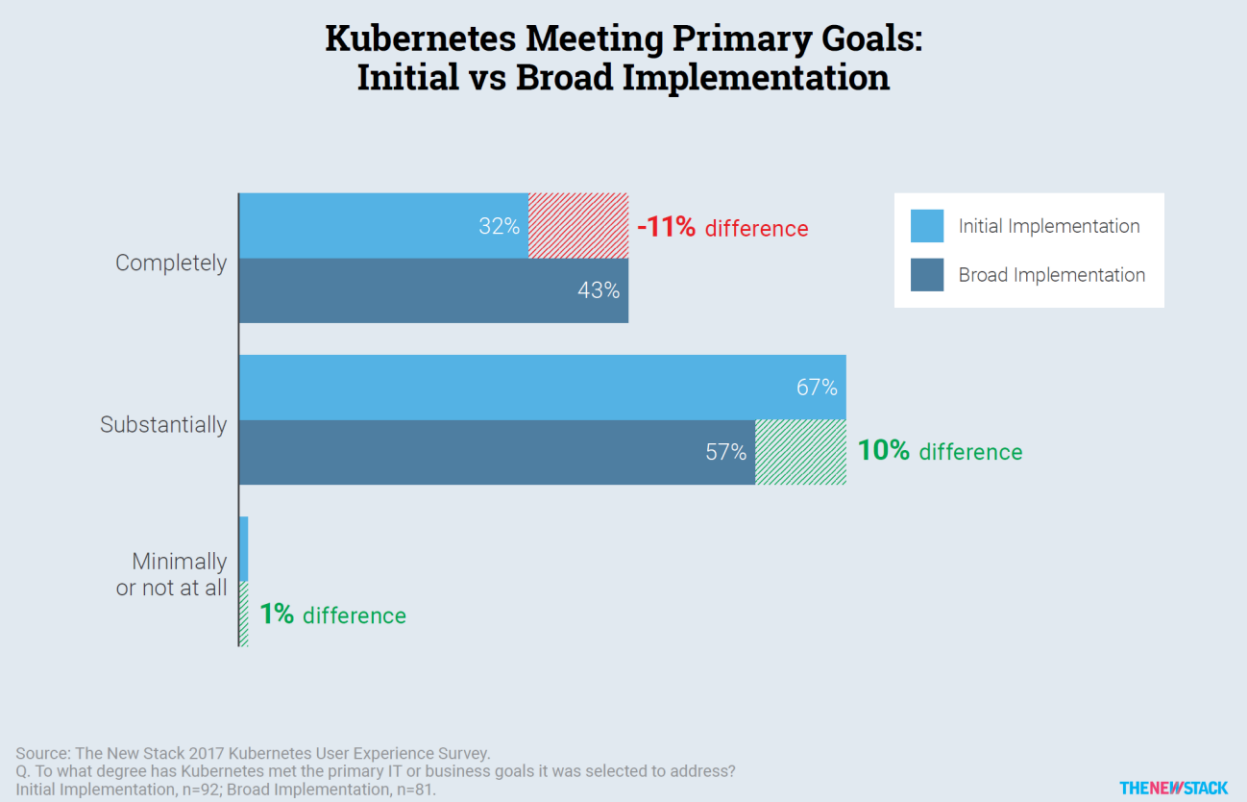
显然，迁移到微服务架构是一个过程而不是一蹴而就的。

是否在简化安装方面存在商机？

如果实施 Kubernetes 是复杂和耗时的，那么也许这是那些提供一体化服务公司的商机。也许是。但是目前，还没有。

我们同时问询了处于初步和广泛使用 Kubernetes 用户对于当前状态是否完全满足他们所有预期，基本上满足还是很少满足。只有 32%的处于初步使用 Kubernetes 的受访者表示，Kubernetes 已经完全满足了他们的公司预期。而 43%的处于广泛使用阶段的受访者

表示对 Kubernetes 完全满意。这应该是正确的方向。虽然少于半数以上的处于广泛采用 Kubernetes 群体表示完全满意，请记住，无论对于 Kubernetes 的生态或市场，刚刚两年时间的发展，比一些分析公司认为一个成熟市场的标准还年轻了整整一年。



这些人中 9%的受访者表示，他们确实需要依赖于外在公司的帮助来在组织或团队中帮助实施 Kubernetes。 有迹象表明雇佣这些外包人员实施，对这些公司存在积极影响， 只有 20%这样的受访者表示，实施需要花费了更久的时间——而之前的调查显示，超过 38%的 Kubernetes 用户抱怨在时间上的花费超出预期。

只有 31%的广泛实施阶段的人群表示，实施花费了比预计更长的时间——比另一项调查数据更乐观，这里有 45%的初步实施阶段人群抱怨有关在时间上的花费超出预期。

总体而言，Kubernetes 正在这场游戏中赢得更好的预期。通过对 173 个有生产环境要求的用户使用 Kubernetes 的调查得出结论，几乎全部的（99%）人群表示这个平台完全或是实质上满足了他们的 IT 或是业务目标。从更广泛的意义而言，这些公司经历了 Kubernetes 的进化阶段。在每一个产品的进化阶段，产品的进化会愈加逼近产品化要求的期望。

然而使用一个商业发行版会让事情变得更简单（至少是，销售商这么说），那些完全满意的用户，只有 47%的仅使用社区支持版的项目，而不采用商业版产品。这证明更多的投资会带来更好的期望。

Kubernetes 用户分享他们的体验

在这次调查所有参与者的共识是，Kubernetes 减少了代码部署次数，增加了部署频率。然而，在短期的运行中，实施阶段确实会耗费更多人力。对待这些问题，IT 组织或团队应该有所改变。

这有一些评论，分享来自那些认为 Kubernetes 对组织或团队产生影响的参与者。

- “每天 300 次部署与以往不到一次相比。团队中的测试循环花费时间从 1 小时降至几乎只要 5 分钟了。这是持续集成的关键因素。” [2-100 名雇员]
- “当评估 DevOps 路线图时，Kubernetes 带来了很多特性，使得我们只需付出很少代价就可以加速这个路线图实现，与此同时，还带给我们云相关的解决方案。它同样缩减了我们在应用部署方面的成本和资源。” [101-1000 名雇员]
- “我们 4 次削减了在基础实施上面的花费。现在它工作正常并且我们不再需要花费时间调试。开发人员可以更容易地、以清晰的方式部署应用（通过 YAML 描述）……我们已经通过 ingress 应用了 SSL。我们最终正确地处理了加密问题。产生的问题一大部分都和安全相关。” [2-100 名雇员]
- “它消除了引入额外独立服务的风险，促成微服务架构。另外，我们现在可以推出与生产环境一致的集群用于负载和性能测试，而且/或者可以去用来验证大规模的基础设施或架构改变。它真是一个功能强大的东西。” [>1000 名雇员]
- “Kubernetes 已经使我们的系统工程师可以免去项目和试验相关资源分配的负担。我们同以往需要几天时间来等待虚拟机资源比较，现在可以在几分钟内跑起来测试应用。” [>1000 名雇员]
- “和以前相比，开发人员可以更加容易的执行他们的应用。以前，了解系统部署，是管理员的必修课，这使得支持和部署应用程序的知识只能是有限几个人来掌握。现如今，很多开发者掌握了基本的 Kubernetes 概念就可以做一些部署下应用相关的事情了。” [2-100 名雇员]

- “开发工作有更多自主权：之前扩展应用需要创建一台服务器，调试它，部署最新的代码而且还需要把它添加到负载均衡器。因此它需要引入我们更多的基础设施人员，应用开发人来配合，这太痛苦了。现如今，为它们仅仅是设置副本数量然后就完成了。或者使用 HPA（Horizontal Pod Auto-scalingpod 自动伸缩）。同样，开发拥有通过网络服务器或 pod 守护监视进程改变配置文件的能力（通过 configmaps）。”之前，他们需要寻求我们帮助，因为这些改变必须经过主管来处理……实施团队有更多时间：我们不再是操作手册中列出的瓶颈，同时会给我们更多的时间去做其他事情而不是运维……我们可以很容易地利用 AWS 保存实例，因为它们都适配 Kubernetes 集群。我们之前可能做不到，因为当我们的应用负载有所改变，AWS 上面的服务器也要跟着改变并不是一件容易事。因为我们想省钱……我们收到的 90%的都是容量问题。通过使用 HPA（pod 自动伸缩）和集群自动伸缩，我们可以显著地消除或减少这些问题。举个例子，从整体上说，现在我们只会在遗留的基础设施上收到有关上述问题，而在 Kubernetes 中没有了。”[101-1000 名雇员]
- “通过 Kubernetes，我们节省了使用 Jenkins 来做 CI/CD 的每月 800 美元的花费。我们降低了 300%的部署操作时间，同时节省了 15%的用于自动伸缩的基础设施成本。”[2-100 名雇员]

虽然成功可以用 Kubernetes 带来的组织或团队结构或工作方式颠覆式的改变来衡量，但是改变来源的起始点在于安装那一刻。这有一些在核心平台成功案例和大家分享：

- “有了 kube-admin 来进行初始化安装，真得很有帮助。同样，还有云原生组织基金会在 KubeCon 大会上的视频资料。”[>1000 名雇员]
- “有了 Google 对于硬件、网络和日志等带来的支持。如果必须由我来做这些的话，我会选择放弃。”[2-100 名雇员]
- “关键的能力在于可以创建‘操作者’或是替换一部分组件。举例来说，我们不得已用一个已有的 iptables 来替换 kube-proxy。同样的事情也存在于 Ceph RBD 提供者——我在 2014 年部署实施过。然而，高品质的 API 支持简单的开发流程管理方式，来优化我们的需求（当前而言，git 的 push 和更新部署少于 10 秒钟；下一步挑战是领先于秒级）。”[2-100 名雇员]
- “许多跨团队的交流，在许多部署应用程序的往复循环中同时看到问题的出现。”[101-1000 名雇员]

- “Kubernetes 正在持续改进并且不断修复问题：并不认为我的问题会陷入黑盒而不知所措。同样，Kubernetes 用 Go 语言编写，使得打补丁更容易。”[2-100 名雇员]
- “了解 Linux， 持续获得广泛的跨团队支持，会尽早让应用成功。”[>1000 名雇员]
- “（我推荐）从 Kubernetes 上游代码构建你自己的版本，利用支持工具和自动化基础工具，Kubernetes 可以很快地重建，测试，验证。”[101-1000 名雇员]

其他参与者指出了潜在的 Kubernetes 用户有着更现实意义的期待：

- “内行的开发专家在技术底层仍是关键，即便是我们正在重度使用社区支持的系统。对于 Kubernetes 的应用设计和调优，是他们自己要面对的挑战。即使 Kubernetes 和它的技术减少了绝大部分系统复杂度，然而仍会在调试产品部署中，有一些关键细节需要理解。”[2-100 名雇员]
- “Kubernetes 不是银弹而且它仍需要更多运维。目前多种云的理想环境对供应商而言很难实现，即便是使用基于 Kubernetes 的容器编排环境。”[>1000 名雇员]
- “Kubernetes 在许多地方仍然是很年轻的。它依然有很多需要打磨的棱角，举例而言，定时任务在很多正常工作负载中很‘鸡肋’。但幸运的是，它正在被重构。”[101 ~ 1000 名雇员]

用一句很简单的话去评论 Kubernetes 生态，就像上述所说的，还没有达到理想的成熟状态，而在这方面做出的任何评测，在初期都是最有帮助的。更值得关注的是，这里涉及到了大多数世界级数据中心虚拟化组件制造商，如此众多的满足需要的实施人员，而这仅仅是经历了两年的 1.0 发布版本的产品做到的。

重新思考 DevOps 流水线

在定义应用程序的现代演进路径的事件链中——路径现在包括微服务，持久化容器，编排器，监视工具和“kubelets” ——安全部分何时开始？

“开发人员或者至少在开发人员的工作流程中负有更多的责任来保护这个应用程序。” 容器安全平台提供商 Twistlock 的首席技术官 John Morello 说：“在新的容器世界中，开发人员需要重新创建容易受到攻击的镜像。 然后他们需要部署这些新的镜像来取代原先的应用。 “

了解更多关于 Morello 关于安全性如何适应编排驱动的观点。**播放 Podcast**。

***John Morello** 是 Twistlock 的首席技术官。 作为首席技术官，John 领导与战略客户和合作伙伴的工作，推动产品路线图。 在 Twistlock 之前，John 曾是财富 500 强全球化工公司 Albemarle 的首席信息安全官，并在微软工作了 14 年。*

Kubernetes 购买者检查表

作者：**KRISHNAN SUBRAMANIAN, JANAKIRAM MSV, CRAIG MARTIN**

由 Docker 支持的“docker 化”的兴起已经改变了部署和管理软件的动力。开发人员在本地的机器上使用容器，部署各种环境且不需要任何依赖。 与此同时，企业 IT 团队在已安装依赖的基础上肯定支持所有级别的应用程序部署， 从开发人员的笔记本电脑到他们自己的生产环境。

从虚拟机（VMs）和虚拟服务器到应用程序和工作负载的部署目标，它是现代 IT 的一个关键目标。 开发人员可以专注于构建应用程序，操作人员可以切换到管理应用程序，而不是管理 VM，以便能够顺利地运行应用程序。

这是一个值得追求的目标。但是，要使这种思维模式成为一种思想，就需要组织或团队吸收和整合一个他们可能从未考虑过的概念，至少不是自愿的:协调的工作量分配。

我们已经列出了几个最重要的考虑事项列表，其中包括文化、组织或团队、实践和技术方面的重要事项，在此之前，您将做出一个关于部署 Kubernetes 的决定。为本章贡献他们的专业知识：

- **Krishnan Subramanian**，首席分析师，Rishidot Research，LLC。
- **Janakiram MSV**，首席分析师 Janakiram & Associates。
- **Craig Martin**，工程高级副总裁，Kenzan。

在选择 Kubernetes 之前的注意事项

Kubernetes 只是编排调度器的一个分支。目前，它是用户数量最多的产品之一，但这场竞争才刚刚开始。Docker Swarm 和 Mesosphere Enterprise DC/OS 是竞争对手。 如果你选择了 Kubernetes，那你将是一个需要做很多重要考虑的人。

CIO 需要考虑什么

IT 部门典型的目标是为组织或团队提供健壮可靠的应用程序。然而，当它面对当今经济现代化的挑战时，组织或团队的首席信息官不仅期望它能提供稳健性和可靠性，而且在资

源消耗方面也变得更加敏捷，更有意义。Kubernetes 帮助它优化资源，并以比以往任何时候都更大的规模运作。 现代企业成功的重点在于两个关键领域：

- IT 如何通过提供所需的基础服务来满足业务需求。
- IT 如何提供 DevOps 所需的工具和支持，从而以更高的灵活性为客户提供软件。

Kubernetes 是将 IT 从守门人转变为现代企业创新者的秘密武器。 以下是 CIO 需要考虑的如何将 Kubernetes 作为转型的工具的注意事项：

业务注意事项

- **价值评估：**重要的是 CIO 对 IT 转换的业务价值进行战略评估，以及容器和 Kubernetes 如何影响组织或团队的转换。一个组织可以从给组织增加收入的东西中获得商业价值，或者给他们提供战略竞争优势的东西。Kubernetes 并不是一种能治愈所有疾病的神奇药丸。CIO 应该得到所有利益相关者的支持，这需要对潜在的障碍进行仔细的讨论，并制定一个计划，在 Kubernetes 被推出时如何避免它们。这里可能涉及包括 IT 管理员、开发人员和业务用户等相关者。现代 IT 将这些关键人员纳入视线，将助于引领文化变革，以及现代化带来的技术和架构变革。
- **遗留评估：**Kubernetes 通过有状态的存储选项支持遗留应用程序（指那些一开始并不是容器化的应用，往往是单个复杂结构的大型程序），尽管这些通常不是 Kubernetes 的理想工作状态。CIO 应该优先考虑从技术和业务角度迁移到 Kubernetes 的高价值工作负载。对于应用程序的架构更改可能会带来成本，CIO 也应该考虑这些成本。例如，一个关键任务的应用程序可能会被中断——导致业务价值的损失——如果它被转移到 Kubernetes，那只是为了使用 Kubernetes。
- **过程评估：**使用像 Kubernetes 这样的平台可以提供敏捷性，并能帮助快速交付业务价值。CIO 应该考虑他们组织或团队的整个价值交付过程，考虑到潜在的陷阱，并决定对 Kubernetes 的投资是否是正确的。当对应用程序的体系结构更改可以编排到容器和 Kubernetes 时，可以最大限度地提高 ROI。

技术方面的考虑

- **模式转变：**使用容器和编排服务需要 IT 决策者，尤其是 CIO 的心态变化。IT 专业人士不必考虑应用程序的可靠性，而是需要考虑它们的弹性。对于 CIO 来说，为 IT 和开发人员带来这种心态变化是很重要的。
- **架构转变：**当基础架构和应用程序组件被视为牛而不是宠物时，您很快就需要重新考虑现有的应用架构。CIO 应该准备好这一点，并提前从开发人员那里得到支持。
- **部署转变：**Kubernetes 可以部署在多个云提供商上，或者使用本地基础设施。CIO 应该根据组织或团队的需求和基础设施的需求制定部署策略。
- **存储转变：**对于 CIO 来说，确定组织或团队的应用程序是否需要有状态存储非常重要。他们应该确保有状态的、面向存储的应用程序的需求——在 Kubernetes 中不会消失——将得到支持。
- **声明转变：**对于 Kubernetes，您总是需要考虑整个系统，并利用它的声明模型进行部署。这需要在基础设施和基础上改变心态应用程序的水平。Kubernetes 支持一个声明式的模型，就像它在问你，“告诉我你想要什么，我会试着去做，”而不是经典的，命令式的模型，“请告诉我，我应该如何创造你想要的。”

人，过程注意事项

- **人才考虑：**迁往 Kubernetes 需要跨职能人才。CIO 应该制定一个策略，要么雇佣新的人才，要么重新培训现有的人才，不仅要更好地理解新技术，还要接受随之而来的流程变化。
- **文化方面的考虑：**拥抱 Kubernetes 只是更大旅程的一个开始。在现代企业的运营团队中，完成划分开发团队的时间已经过去了。虽然这个概念现在有许多变体叫做 DevOps，但它们都是一个共同的概念，即它们应该分担责任，并且彼此更直接地交流。Kubernetes 并不是 Dev 和 Ops 的协作平台，也不是 Dev 和 Sec 和 Ops 的协作平台，或者是现在流行的任何流程模组的混合。但是正确地采用，并接受分布式编排系统的概念，确实需要创建应用程序的人和管理他们的人，比经常在一起吃午餐和交换故事要更频繁的交流。CIO 的职责是授权利益相关者促进这种沟通，这样这些团队就不会为了进行建设性的讨论而变成官僚机构了。
- **失败的因素：**开发者快速试验和失败的机会。CIO 应该授权这一任务，并授权利益相关者以这种方式进行实验，从而使失败成为恢复和改进的第一步。

IT 实现者需要考虑什么

Kubernetes 在容器编排方面非常强大，但对于每个开发环境来说，它并不一定完美。组织或团队中的关键利益相关者应该首先问自己这些问题：

您的应用程序是否需要一个分布式架构（比如，用于微服务）？

尽管 Kubernetes 可以在一个庞大的基础设施中工作，但它的重点是大规模地精心编排大量的微服务。 您需要考虑的是，您今天运行的服务，以及您计划在未来运行的服务，是否可以与使用它们的应用程序分离。 换句话说，可以在 API 后面部署工作的代码吗？如果是这样，您可以使用 Kubernetes 将这些服务与调用它们的客户机分开编排。这种分离对于使这些服务可伸缩至关重要。

是否有监视工具来支持 Kubernetes 的部署？

基础设施监控有助于确保其服务的应用程序的资源——存储、CPU 和网络——的健康和可用性。 如果您没有这样的工具，您应该投资一个健壮的监视机制，它可以跟踪底层节点的健康状况，以及在 Kubernetes 中运行的工作负载。 有开放源码和商业监视工具可以很好地与 Kubernetes 环境相结合。开始考虑监控工具，以及其他工具，马上，在接下来的一章中，我们更多地关注监测的考虑， 因为 Kubernetes 可以带来特别的挑战。

您的应用程序容器准备好了吗？

容器是来自虚拟机的。组织或团队中的每个人——包括开发人员、系统管理员和 DevOps 从业者——应该对容器有一个基本的了解。 如果他们还没有意识到通过集装箱化交付的业务价值(并不是每个人都在 rst 上)，他们至少应该尊重团队的领导人及其背后的理由。团队应该在非生产环境中采用容器， 例如开发、测试、QA 和展示。

你的人准备好了吗？

在过渡过程中采用 Kubernetes 比采用容器要晚得多。Docker 增加了开发/测试环境和持续集成/持续开发（CI / CD）过程的价值。 在开始使用 Kubernetes 之前，这些价值应该已经被兑现了——或者，这个问题适用于任何一个编排调度器。对容器的业务和技术价值的充分认识和确认是在您能够有效地使用 Kubernetes 来管理生产中的容器工作负载

之前的先决条件。管理层应该采用容器技术带来的好处。最重要的是，所有的利益相关者都应该接受在分布式系统环境中使用容器的培训。Google 为 Kubernetes 专业人士提供专门针对 Google 云端平台认证培训。

您是否打算将遗留应用程序迁移到 Kubernetes？

您选择遗留系统迁移的迁移方法可能具有挑战性，无论它是什么。例如，一个常见的方法是部署一个 API 网关，然后将整体分解成 Kubernetes 的一个个特性集， 并在一段时间内完成。这是一种有效且易于管理的方法，但这并不容易完成。

你是否计划从一个全新的绿色部署开始作为替代计划呢?

挑战开发和部署应用程序到全新的 Kubernetes 环境是完全不一样的。你可以更自由地承担风险，因为本质上是全新的技术。与此同时，你也会遇到和其他人一样的痛苦。

你会选择商业版还是社区版？

Kubernetes 可以作为一个开源发布的技术栈，或者作为一个有管理的商业产品 。根据您的内部 IT 团队的技术能力，您可以选择 GitHub 上可用的开源版本，或者购买一个基于 Kubernetes 的编排商业版，比如 CoreOS 和 Canonical，它们是专业服务和支持的供应商。

你准备好投入时间和精力建立你自己的容器镜像了吗？

一个容器基于预先配置好的镜像。这样的镜像通常包括基本操作系统。除非所包含的应用程序是已编译的二进制文件，否则镜像可能还包括它所依赖的库和其他依赖项。您的组织或团队可能希望投资于一个私有镜像库，它存储了基本镜像和自定义镜像。 一些商业镜像库有镜像扫描和安全功能，扫描潜在漏洞。即便如此，对 Docker Hub 镜像库进行的一项最近研究 发现，有五分之四的镜像包含了至少一个记录的安全漏洞。因此，您可以选择从二进制文件中完全编译您的镜像组件，使用您的团队了解的镜像库并信任安全。另外，您可能会考虑 CoreOS 工程师 Brian Harrington 提出的一种架构方法，称为最小容器——一个更简洁的组装容器的方法。

您的存储是否准备好用于高性能工作负载？

Kubernetes 集群可基于分布式文件系统，如 NFS，Ceph 和 Gluster。 这些文件系统可以提高吞吐量。在 Kubernetes 中运行的有状态应用程序可以利用这些底层存储，它可以为生产环境工作的后端服务提供存储。

你期望的正常工作时间？

您的客户服务水平协议（SLA）需求将对您的 Kubernetes 部署的各个方面产生重大影响：您如何控制您的环境，您如何控制每个应用程序，您能够承受多少复杂性， 您可以支持多少同时并行的部署管道……而且名单还不止于此。所有这些变量都会影响应用程序的总成本。对于一些内容， 下面的表 4.1 解释了可用性目标中每个“9”的预期停机时间（摘自 Susan J. Fowler 的书）。由您自行决定您的组织或团队所需要达到每个级别的工作量。

“可靠性 9”停机阈值				
可靠性	每年	每月	每周	每天
99.9%	8.76 hours	43.8 minutes	10.1 minutes	1.44 minutes
99.99%	52.56 minutes	4.38 minutes	1.01 minutes	8.64 seconds
99.999%	5.26 minutes	25.9 seconds	6.05 seconds	864.3 milliseconds

表 4.1：可靠性标准中的停机等级如何转化为实际时间。

你是否有一个发布管理系统?

迁移到 Kubernetes 的关键任务之一是自动化应用程序的部署。 Kubernetes 的部署支持滚动更新、补丁、灰度部署和 A / B 测试。 为了利用这些功能，您应该设置一个良好的构建自动化和发布管理平台 。Jenkins 是一个使用广泛的自动化部署工具的一个例子，它与 Kubernetes 很好地结合在一起，通过从源代码构建容器镜像，可以将其推到生产环境和非生产环境中。

你准备好所有的日志了吗?

Kubernetes 支持基于群集的日志记录，允许工作负载在集中式日志记录中记录容器活动。创建集群后，日志代理（如 Fluentd）可以从每个容器的标准输出和标准错误输出通道中获取事件。这样生成的日志可以被导入到 Elasticsearch 中，并用 Kibana 进行分析。

Kubernetes 生产需要什么

生产环境不一定总是在物理上，也许是虚拟机，并且与开发环境隔离。 也就是说，Kubernetes 对生产环境的要求超出了开发或测试环境所需要的范围。 以下是您应该评估的关键因素，以及它们之间的差异。

高可用性和弹性伸缩

“高可用性”这个短语的意思可能取决于它所处的语境。因此，在这个讨论中，我们将着重于部署 Kubernetes，以使它能够在部分子系统失败的情况下存活下来。这种生存能力被称为韧性（有时是弹性)。 记住，Kubernetes 不是整个容器系统，而是部署在数据中心的实体。与其他数据中心一样，它的操作条件和长期威胁都是一样的。当我们考虑 Kubernetes 环境中的高可用性时，我们倾向于将其划分为三个不同的方面:

- **基础设施高可用**决定了 Kubernetes 所运行的基础设施是否具有高度可用性和分布式，而不考虑环境。
- **Kubernetes 高可用**确保了环境从来没有单一的故障点，而且编制系统的组件与基础结构的相同线路的分布是合理的和比例的。
- **应用程序高可用**集中于应用程序，并确保所有的 pod 和容器都达到了正确的可用性水平。根据应用程序的需求，应用程序组件将松散地分布在 Kubernetes 环境中，该环境本身分布在整个基础设施中。

由于 Kubernetes 可以在一些公共和私有基础设施上安装作为服务(IaaS)平台，这些平台的功能和属性共同构建了 Kubernetes 安装的固有弹性支持。例如，IaaS 能够提供的分区级别是多少?

您的公共云的 IaaS 平台由细分的部署位置组成。AWS 称这些位置为“区域”，而细分为“可用性区域”，其他主要云提供商也纷纷效仿。Kubernetes 可能被部署在这些区域。但默认情况下，Kubernetes 总是部署一个单一的主程序，不管你有多少个区域。在高可用性场景中（无论如何是也是多个区域中的一个点），在单个区域中拥有主机和控制器会造成单点故障。

一些供应商，以及 Kubernetes 社区的其他参与者，已经提出了高可用性模式的安装过程，在集群中有多个主服务器，最好是每个区域一个主服务器。这样，任何位置的容器运行时在其中一个区域失败都不会有影响。因此，如果您的 IaaS 有两个区域，那么您应该保证在每个区域至少有一个高可用的主区域。

最常见的，也是通常最推荐的处理方法是使用多主机 Kubernetes 环境。在这个场景中，只有一个主机被“选”为主节点，所以如果它失效，就会选出一个新的主节点。然后使用这个新的主节点来处理 Kubernetes 环境的所有调度和管理。

Kubernetes 集群实际上是相关的工作节点集群。由于您的业务的应用在这些集群上运行，因此每个集群都部署在尽可能多的逻辑分区上是有意义的。

注意：使用您的主节点用于管理 Kubernetes 环境是最佳实践，并且只使用工作节点来托管和运行所有的 Pods。这将保证在 Kubernetes 和您的应用程序之间进行完全隔离。

使用私有 IaaS，您仍然可以将服务器分布部署到到每个区域。虽然每一个私有的 IaaS 配置可能是独一无二的，但它们都有一些共同之处。 无论您的 IaaS 是公共的还是私有的，您的存储和网络分区应该与您的计算分区相对应。在数据中心中，您将拥有冗余的存储区域网络（SAN）功能以及冗余的机架顶部和机架网络连接功能，可以对刀片机箱中的冗余进行镜像。这样，您就可以提供冗余网络、电源、计算和存储分区。使用公共 IaaS，每个服务提供商都能以不同的方式处理存储和网络功能，尽管大多数服务内置分区冗余。至于网络，您应该使用 IaaS 的负载均衡来暴露 Kubernetes 服务端点。

最终，来自 IaaS 的所有这些底层弹性特性允许 Kubernetes 支持副制集合，从而确保指定数量的 Pod 副本一致地部署在分区内。这样，Kubernetes 可以自动处理分区故障。最近的 Kubernetes 版本已经实现了一种称为部署 `Deployment` 的副本集合的超级结构（super-construct），每个副本都涉及实例化定义的部署对象。这个对象操纵“部署控制器”自动管理副本集合，并定期刷新它们，以使它们尽可能接近您声明的所需配置。

通过这种方式，您的 Pod 会在发生停电事故时在其它未受影响区域重新进行平衡，而无需操作员执行特殊操作。 更直接的说就是，可以在声明节点时将节点的 `failure-domain` 进行显式声明。 您可以根据您的特定需求定制调度程序，并确保它将跨节点或跨特定基础设施进行复制。

我们过去的一种做法是的是通过 `nodeAffinity`（正式的名称是 `nodeSelector`，在 Kubernetes 1.6 中进行了更新），通过它告诉调度程序在发生特定事件时究竟应该影响哪些节点。 这可确保应用程序层的可用性与基础架构分区保持一致，从而消除或至少减少基础架构出现故障时的停机时间。

网络

Kubernetes 网络模型取决于一个重要的事实：每个 Pod 都有自己的 IP 地址。 这个地址是 Kubernetes 集群内部网络的一部分，它允许所有的 pod 彼此间进行通信。 由于 Kubernetes 支持软件定义网络（SDN）的模块化实施，因此您可以选择多种网络方案。

虽然确实有可能在不使用 overlay 网络的情况下实现 Kubernetes 群集，但是如果不使用 overlay 网络，则可能会带来潜在的重大限制和配置开销。 在这种情况下，基础设施需要知道所有的集群 IP 地址，并能够路由到所有的集群 IP 地址。对于像 AWS 这样的 IaaS，这可能是相当重要的限制条件。

一个 overlay 网络为管理 Kubernetes 集群内的 IP 路由提供了一种更简单的方法。 这个虚拟架构通常以某种形式的封装将内部集群网络与运行 Kubernetes 的基础设施分开。为了简单起见，最常用的 overlay 网络是 CoreOS 的 Flannel。 还有其他一些组建例如 Calico 和 Weave Net，它们不仅支持 IP 地址管理（IPAM），还支持用于控制流量的网络策略。

您选择的 SDN 对群集大小和整体性能都有影响，并且可能取决于群集正在运行的基础设施提供商。 每个网络覆盖实施针对不同的基础设施类型和群集配置具有不同的模块。 每个应用程序的具体需求将有所不同，但下面这些是你需要知道的基础知识。

- **Flannel** 非常灵活，因为它支持许多不同的协议和网络模型。 和 CoreOS 捆绑在一起，Flannel 也是我们经验中最安全的，也是最容易支持分布式数据需求的网络方案。
- **Project Calico** 是三个选项中表现最好的。 我们一直可以看到，Calico 可以以最快的速度跑赢别人。 但是，您在性能方面所获得的优势，在功能上会有所损失，因为 Calico 不支持许多安全功能，协议或子网限制。

- **Canal** 是由 Calico 项目团队发起的开源混合方案，现在由 Kubernetes 平台迁移提供商 Tigera 提供支持。 它集成了 Calico 的网络策略管理功能和 Flannel 的网络连接组件。
- **Weave Net** 与 Flannel 类似，但在关键特性方面略有不同。它支持 NaCl（发音“salt”）加密，它有更好的子网限制，并支持部分连接的网络（例如穿越防火墙）。这些好处在少数领域略有不足，在分布式数据需求或者支持传输层安全（TLS）方面往往不够健壮。

存储

容器最适合于符合 12-factor 的无状态应用程序，在这些应用程序中，服务对请求作出响应，然后就可以消失不见，不留下任何东西。正如你所能想象的，这是一个非常新的概念——对许多企业来说，是提交申请的外来方式。而且，它无法转化为现有应用程序的存储模型，而这些应用程序生命周期尚未完全结束（意思是这些程序还在使用过程中）。

Kubernetes 通过增加了对持久存储卷的支持（特别是 `PersistentVolume` 子系统），支持动态配置，同时支持有状态和无状态的应用程序。 这允许为 Kubernetes 集群上的传统和其它有状态应用程序提供支持，从而使 Kubernetes 成为企业平台的有竞争力的候选人。 Kubernetes 编排器支持连接到 Pod 的卷以及外部持久化存储卷。

Kubernetes 中的卷与 Docker 中的卷概念不同。 Docker 卷直接连接到容器，Kubernetes 卷则与一个 Pod 相关联。 因此，即使 Pod 内的容器出现故障，存储卷仍然保持不变。 但是，Kubernetes 存储卷的生命周期同样是短暂的，这意味着当 Pod 终止时它将被终止。 在拥抱 Kubernetes 时，你必须在你的脑海里区分这两个相同命名的概念。

Kubernetes 提供对有状态应用程序的支持，通过对存储空间的封装，提供用户使用的 API 以及存储空间的管理。 一个存储卷通过 `PersistentVolume` 子系统在 Kubernetes 中进行外部化，`PersistentVolumeClaim` 则声明 pod 是如何以无缝方式使用 `PersistentVolume` 资源。 具体来说，它在 Pod 内的容器之间创造了一种联系，而存储卷类的生命周期将超过那些 Pod 的生命周期。

Kubernetes 为来自云供应商的各种存储产品（例如网络文件系统（NFS），GlusterFS，Ceph，Azure SMB 和 Quobytes）提供逻辑文件系统安装选项。 存储专业兴趣小组（Special Interest Group-SIG）目前正在努力通过实现插件支持的外部化来更轻松地添加新存储服务和系统的支持。

在开始将已有应用程序转换到分布式系统环境（如 Kubernetes）之前，你应该确保的它们存储需求得到支持。

安全

容器化改变了数据中心的整个安全概念。编排调度器的使用再次改变了它。 也许可以说，您已经在与您的数据中心向新的安全模型过渡，并且可能正在进行管理。 Kubernetes 在这中情形下的介入会影响这个模型，并可能会改变其中的内容。

有关在 Kubernetes 环境中运行的工作负载的安全主题分为三类：

- **应用程序级别：**会话模型，数据加密，出口限制。
- **平台级别：**密钥管理，数据存储，访问限制，分布式拒绝服务（DDoS）保护。
- **环境安全：**Kubernetes 访问，节点访问，主节点配置，etcd 中的加密密钥/值存储。

节点访问

简而言之，很少有人可以直接访问 Kubernetes 节点。 每当你给予某人这样的访问权限，你都会承担风险。 用户其实无需直接访问节点主机，只需运行 `kubectl exec` 即可查看容器及其环境，并且不会引入安全漏洞。

如果您需要细粒度的容器级安全性，操作员可以通过 Kubernetes 的授权插件对访问权限进行微调。 这些插件可以限制特定用户对某些 API 的访问，并防止意外更改系统属性，如调度程序选项。

命名空间

Kubernetes 允许应用程序安装在不同的命名空间中。 这在应用程序之间创建了一个轻量级的边界，并有助于对应用程序团队进行适当的划分。 这种边界用于防止意外部署到错误的 Pod 或集群，并且能够在每个命名空间上建立公司级别的资源约束（配额）。 例如，您可以为每个由微服务组成的应用程序提供自己的命名空间。 这允许单个 Kubernetes 环境承载多个应用程序，而没有应用程序之间的冲突（跨越命名空间）的风险。

命名空间作为工作负载的逻辑边界 - 它将应用程序的范围限制在仅适用于相同名称的系统部分。 更具体地说，命名空间表示由同一物理集群支持的多个虚拟集群。 这首先使容器化成为可能。

加密管理

Kubernetes 具有内置的加密信息存储功能——通常是与应用程序运行时相关的离散值，但不应与其他应用程序或其他人共享。 当你将某个东西定义为需要加密时，它将独立于该 Pod 存储，确保它在休眠时被加密。 一个 Pod 只能被授予访问在其 Pod 定义文件中定义的加密信息访问权限。 在容器启动时，将加密信息提供给容器，以便它们可以被适当地使用。

Kubernetes 提供了一种通过环境变量直接向容器提供加密密钥的机制。 具体而言，您可以在 pod 定义文件中定义 `secretKeyRef` 变量。

作为 Kubernetes 加密机制的补充，您可以选择使用另一个密钥管理工具，比如 Vault。专用于密钥管理的应用通常提供更强大的功能，例如密钥轮换。

Kenzan 的观点：“在 Kenzan，我们对使用 `secretKeyRef` 方法非常犹豫，因为把密钥从直接使用到写入名字和位置都知道的属性文件中并不复杂。 按理说，应用之间的加密不一定适用于人与人之间， 但是为不能被看见的东西提供指南和地图从来都不是好主意。 也许有人这么用，这是有道理的，但我们会建议慎重使用 `secretKeyRef`。 “

访问管理

除了网络分割之外，两个 Pod 之间几乎没有安全性。对于许多数据中心来说，这是一个严重的问题，因为不良的参与者可能最终访问整个群集，或至少引发网络错误，然后所有服务都将暴露给黑客。

您可能会发现最好制定一些内部安全控制级别来验证 pod A 是否可以调用 pod B，假设这些交互是由策略允许的。像 JSON Web Token（JWT）声明（用于帮助验证基于 JSON 的真实性的声明）和短期签名密钥轮换一样，在这里使用会很方便。通过在 JWT 中提供角色，这可以确保每个特定端点非常细化的安全性需求，同时也确保了频繁的 JWT 私钥轮换（我们建议每隔一两分钟）。使用这种模式，任何人如果没有经过签名的 JWT，就算可以成功地破坏防御系统进入网络也无法调用任何服务。一个签名的 JWT 只设置一两分钟的有效期。

结论是

您刚刚看到了许多与 Kubernetes 日常保养和维护有关的关键因素，以及与之交互的网络结构。 您和您的组织机构可能不会对部署做出严谨的决策承诺，但这个决定会影响应用程序的各个阶段，以及监视和运营等方面，而不是仅仅是对准备的内容快速了解一下就完事了。 在下一章我们将为您提供即将到来的产品的预览，如果您愿意的话：一个在真实世界的生产环境中，编排调度器的概况。

云原生应用导致企业整合

公司如何看待 Kubernetes 正在开始改变，就像人们开始使用平台时经常发生的那样。 红帽 OpenShift 的 Brian Gracely 认为，正是公司开发的最初应用程序，正导致企业转向采用更大规模的 Kubernetes。

通常情况下，公司会将开始使用移动应用程序作为第一步。 这个应用程序开发流程可以帮助团队了解真正的创造优雅的用户体验的必要条件。 云原生应用程序可用作与现有企业环境集成的一种方式。 现在，团队正在超越一个单一的应用程序，以更整体的观点，改变公司进入市场甚至与客户交互的方法。

了解更多关于 Gracely 和 OpenShift 团队发现的有关将现代软件开发思维与企业集成的关键点。**播放 Podcast**。

***Brian Gracely** 是 Red Hat 产品战略总监，专注于 OpenShift。 他拥有 20 年的战略，产品管理和系统工程经验。 Brian 共同主持了屡获殊荣的播客 The Cloudcast 和以 Kubernetes 为中心的播客 PodCTL。*

在产品中使用 K8s 的问题与挑战

作者：**CRAIG MARTIN**

在这一章，我们将着重于产品中部署 Kubernetes 环境的相关问题。这非常重要，需要集中的、审慎的思考。整个过程中可能会有各种错误发生，你需要对此做好心理准备。错误恢复在这里很有用。幸运的是，容器编排在设计中考虑到了这一点。

为了帮助你在自己公司的产品中部署 Kubernetes（不仅仅是在开发中），我们与软件工程服务供应商 Kenzan 的工程部高级副总裁 Craig Martin 取得了联系。Martin 和他的团队帮助多个企业建立了持续交付流程，这些流程常涉及 Jenkins，最近几年更多的涉及了 Kubernetes。

Kenzan 的团队会与你分享他们的日常工作，包括他们在自己的生产环境中部署 Kubernetes 的过程和对客户端管理的思考。下面是我们将要探讨到的内容：

- 用于监测 Kubernetes 平台， 和在 Kubernetes 平台上运行的大规模的应用的工具和方法；
- 记录系统事件发生的方法；
- Kubernetes 自动化工作所需的工具和方法；
- Kubernetes 及其应用程序之间互相需要的资源。

大规模生产环境中的 Kubernetes 监控

监控分布式系统环境和监控客户/服务器网络，是完全不同的概念，这是显而易见的：对你的公司来说，性能、弹性和安全性比某个处理器上独立运行的单元重要的多。所以，对于服务器或网络地址，传递的监控信息远远少于分布式系统和微服务。

这一事实所造成的挑战是：你需要制定一个监控策略，然后再选择能够帮助你执行这个策略的工具。Kubernetes 不能自动监控，也不包括你需要的工具。

所有的 Kubernetes 组件——容器、pod、node 和集群——都在监控范围之列。另外，正确的接受监控结果并采取适当的纠正措施也同样重要。后一项是经常被 DevOps 团队所忽视的。

选择一个监控工具集当然重要，但可能不是你想到的原因。每一个监控工具集都有利弊，我们可以说是独特的品质。当你需要同时监控多个方面的时候，你可能发现自己就像 Kenzan 内部一样，综合使用了很多工具集。事实上，关于工具集最重要的是你要坚持你选择的那套，并且连续的运用在你的 k8s 集群。

虽然有很多选择，但我们需要找到适合自己的。下面列出了一些经常被 Kubernetes 用户使用的监控工具集，包括我们团队已经使用的和推荐使用的：

- **Heapster**：可以安装为 Kubernetes 内的一个 pod，它收集集群内的容器和 pod 的数据和事件。
- **Prometheus**：开源云计算基金会（CNCF）提供的项目，有强大的查询功能、可视化和警报功能。
- **Grafana**：连同 heapster 一起，为你的 Kubernetes 环境提供可视化数据。
- **InfluxDB**：一个高可用的数据平台，用于储存所有 heapster pod 捕获的数据。

如果没有更长的生存周期，至少也应该保证监控工具的生存周期和你的应用程序一致。当你需要信息时，却由于监控工具停止运行而无法获得，没有什么比这更让人沮丧的了。在针对某个具体应用程序时，一个好的监控行为应该支持查看基础平台上的故障点，并确保任何可能发生的中断不会导致监控盲区。

大部分用于监测 Kubernetes（比如 cAdvisor，Heapster）的第三方或可添加应用可以部署在 Kubernetes 环境中。不过，要确保这些应用程序的日志记录发生在集群之外，或者它们具备失败恢复能力。值得注意的是，这种简单而重要的概念却常常被忽略。

监控容器

容器是 Kubernetes 生态系统最低层次的实体。监控这一层级不仅包括监控容器化应用是否健在，同样也要确保应用数量是否正确。大部分 Docker 提供的指标可以用于监控 Docker 容器，另外也可以利用许多传统的监控工具（如 Datadog，Heapster）。Kenzan 趋向于关注底层数据去确定每个容器个体的健康；如：

1. **CPU 利用率：**以分钟、小时或天的均值来表述。

2. **内存利用率：**以每分钟、小时或天的使用率均值来表述。

3. **网络 I/O：**确定网络中的所有主要延迟时间，因为流量高峰的影响可以被网络延迟放大。监控 I/O 也可能会帮助我们更好的把握缓存或关闭应用程序的时机。

这三个指标将显示容器是否压力过大，延迟是否发生在容器级别，和容器扩展数量是否如你所愿。

注意：Linkerd 是一个 CNCF 提供的服务发现代理，用来管理 Kubernetes 项目。利用 linkerd 作为你网络流量的负载均衡器，是实现减轻负载的网络管理方案，也可用于处理由于重连出现峰值或邻机流量过大带来的性能问题。

监测 pods

Pods 是 Kubernetes 在容器中的抽象层。这一层 Kubernetes 会自动保证 Pods 的数量规模符合预期。Pods 会定期的检查健康状态。pod 层级最有用的监测是这些 pods 的生命周期事件。在这些监测中获取的数据，对于了解 Pod 数量是否正确，数据高峰是否处理，和 Pods 错误是否修复很有帮助。

从 Kubernete 中可以获得很多数据；这份在 GitHub 的清单会提供给你完整的说明。以下是 Kenzan 团队的经验中需要去关注的事项：

- **规模事件**发生在 pods 创建和存续时期。这些事件提供了应用程序规模如何处理的更高层视图。
- **Pod 中止事件**对知道哪个 pods 被强制关闭和强制关闭的原因非常有用。有时中止是因为 Pod 数量波动，另外 Kubernete 也可能 因为一个简单的健康检查失败而强制中止 pod。区分两者是非常重要的。
- **容器重启**对于监测 pod 中的容器健康是很重要的。
- 冗长的**启动时间**是应用程序健康状况不好的常见信号。容器的启动和中止应该是很快的。

如果保持我们的监测简洁和清晰，那么管理就更加容易。更进一步的，如果需要变化的历史曲线，你可以尝试某种服务的 Dashboard，比如 Grafana。

监测集群

监测一个产品系统所涉及的过程要么彼此相似，要么逐渐变得相似，尤其是搜集数据的过程。但是监测的不同组件之间的关系会有所不同。在集群层级，我们更倾向于考虑应用的整体。我们仍然监测 CPU，内存利用率和网络 I/O，但会关注整个应用的表现。

当一个应用规模超出其配置的内存和 CPU 限额时，我们经常能看到一个集群出错。有弹性的应用面临一个独特的挑战：它将持续尝试维持预期的状态，直到无法实现。因此通常在集群层级，你得到一个唯一彻底出错的明确信号。出于这个独特的原因，密切监视每一个集群是非常重要的，要在集群发生故障前寻找预兆。

我们监测四个关键变量用作时间序列分析：

1. 整体集群 CPU 使用率；
2. 每个节点的 CPU 使用率；
3. 整体集群内存；
4. 每个节点的内存使用率；

尽管比较少见，但节点的 CPU 使用率可以反映某个节点相对于其他节点表现欠佳，而每个节点的内存使用率可以揭露路由问题或负载共享问题。使用时间序列分析，你能够在启发式图表上绘制这些变量。

The New Stack 发现了一个基于容器的，称之为 CrateDB 的分布式 SQL 数据库。它的设计融合了分布式架构的自我修复和即时故障转移的原则。CrateDB 的工程师称它易于被 Kubernetes 管理，并且最近已被用于存储机器数据——用于性能监控的一类数据，这些数据包括了 Kubernetes 集群自身，以及它们的 pods 和容器。

监测网络

随着越来越多的应用正在向微型服务的弹性应用转移，我们很容易忽视这种转变很大程度依赖于网络来保持健康和运行。在性能不佳的网络上，高弹性微服务应用永远不会流畅运行。没有任何防范方法或自动修复功能让其正确运行。

这就是我们为什么相当看重网络监控的原因。幸运的是，类似 Heapster 的工具可以捕获网络指标及其性能。虽然通常我们发现这些指标对于发现瓶颈是有用的，但它们在诊断根本原因方面做得不够。这要求使用网络特定工具进行更进一步的发掘。

我们通常监测以下项目，并且发现区分发送和接收是很有用的：

- **网络接受的字节：**显示固定时间片的字节。我们一般关注这里的尖峰。
- **网络已发送字节：**揭示传输和接收流量之间的差异，这是非常有用的。
- **网络接收的错误：**显示指定时间段中丢包或网络错误的数目。
- **网络已发送的错误：**告诉我们发送过程中发生的错误的数量。

着眼于 Kubernetes 环境的基础，将会带来与指标一样的好处。Kenzan 建议你定期的获得调查结果，并且制定一个可行的计划来解决你发现的问题。该行动计划的对象是应用程序，Kubernetes 环境和它们运行的平台。值得注意的是，很多团队忘记了这一反馈循环的重要性。

注意：目前，Kubernetes 的自动延展特性处于 alpha 版。这是一个非常令人兴奋的发展，因为这使得在自动延展中添加精细的和可定制的反馈回路成为可能。

Kubernetes 的自动化伸缩

Kubernetes 不是一个管理平台，也不应该被认为是。编排的核心是可靠地在应用程序的部署和管理中实现系统自动化，不需要每一步都进行人工干预。如果你在 Kubernetes 环境中使用的工具不能实现自动化，那么你还没有真正体会它的好处。

日志

任何 Kubernetes 生产环境都非常依赖日志。我们通常将应用程序日志和平台日志分开。这可以通过各种不同的工具和应用程序完成，或者直接对日志本身进行过滤和标记。在分布式系统中，即使是不同的微服务架构，日志也提供了重要的线索去准确追踪特定调用，来确定错误的根本原因。

以下是我们对于分布式 Kubernetes 环境中的日志的建议：

- 使用一个单一的，高度可用的**日志聚合器**，并在单个位置捕获整个环境中的数据。
- 在每个特定客户机的端对端调用中创建一个单一的**公共事务 ID**，这将使跟踪所有的线程更容易实现。
- 确保**服务名称和应用**都被记录。
- 规范整个工作栈的**日志等级**。
- 确保计划**保护的数据**不被明码记录。

除了这种高层次的记录方法，你应该理解 Kubernetes 是如何处理自己的日志和事件。

Kubernetes 节点运行在一个虚拟的 Linux 的计算平台。例如 kubelet 和 Docker 等组件在运行期内在本地 Linux 上运行，登陆到本地文件系统。Linux 日志记录配置在不同的文件夹位置，包括常见的`/var/log`。管理员应该做的第一件事是验证这些日志文件，就像其他 Linux 日志一样。Kubernetes 说明文档为文件的循环提供了很好的建议。即便你计划替换本地日志机制，日志配置也应该检查。

我们不建议你在临时的云计算环境中为虚拟计算实例保留日志。这些日志可能随时消失。现代日志和分析工具提供足够的信息和图表帮助运营商确定大的 Kubernetes 集群部署里的实际发生情况。你应为以后对日志的回顾与分析，使用一个日志聚集服务去将日志保留在 Kubernetes 环境之外。

有几种方法可以可靠的捕捉 Kubernetes Linux 日志，Kubernetes 基于容器的组件日志，和所有的应用程序容器的日志数据：

- 简单延长的几种方法现有记录能力。当日志在节点上积累和循环时，你可以运送它们去其他地方。一种流行的方法是使用一个日志容器，其目的是将日志发送到另一个系统。
- 或者，你可以使用 Fluentd 数据采集器传输日志去一个 ELK 系统（Elasticsearch, Logstash and Kibana），或其他一些日志聚合和分析系统。你可以在每个节点用 Fluentd 容器记录 Pod 日志（Kubernetes 使用 `DaemonSets` 概念使其容易）。此日志传送方法利用命令 `kubectl logs`。
- 还有一种方法是，应用程序容器在同一个 POD 中有一个日志容器，将应用程序与系统日志分离开来。

不管你选择哪种方法，日志都不能保留在某个临时节点上，而且它们必须转移到其他地方。

自我修复

我们认为，你的系统几乎不可能在没有自愈能力的情况下实现高的正常运行率，特别是在分布式环境中。Kubernetes 可以定期监测 pods 和容器的健康状况，并立即采取行动解决遇到的问题。Kubernetes 生来辨识的两种对象类型是 `podstatus` 和 `containerstatus`。这些对象可以呈现几种状态：

- **待定**：Kubernetes 接受了 pod，但尚未创建一个或多个容器的镜像。这包括在预定时间之前以及在网络上下载镜像所花费的时间，这可能需要一段时间。
- **运行**：Kubernetes 已经限制了 pod 分发到某一个节点，并创造了所有的容器。这些容器中至少有一个正在运行，或正在启动或重新启动过程中。
- **成功**：pod 中的所有容器都已成功终止，不会重新启动。
- **失败**：pod 中的所有容器都已终止，这些容器中至少有一个已退出非零状态，否则将被系统终止。
- **未知**：由于某种原因，pod 的状态无法获得，通常是由于与主机通信的错误造成的。

用下面三种不同的方法，可使运行在每个节点的 kubelet 代理能够得到更详细的健康检查信息：

1. `ExecAction`：在容器中运行一个特殊命令。
2. `TCPSocketAction`：对特定容器执行简单的传输控制协议（TCP）检查，以确保其存在。
3. `HTTPGetAction`：在容器上执行一个简单的 HTTP GET 检查，期望得到 200（OK）或 400（bad request）的响应。

Kubelet 也可以探测 pod 内容器的“活跃度”（其反应能力）和“准备度”（它做好准备的处理请求）。你可以配置一个活性检查以满足特定需求，如例 5.1 中的 YAML 文件：

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - args:
    - /server
    image: gcr.io/google_containers/liveness
livenessProbe:
      httpGet:
      # when "host" is not defined, "PodIP" will be used
      # host: my-host
      # when "scheme" is not defined, "HTTP" scheme will be
used. Only "HTTP" and "HTTPS" are allowed
      # scheme: HTTPS
      path: /healthz
      port: 8080
      httpHeaders:
        - name: X-Custom-Header
          value: Awesome
    initialDelaySeconds: 15
    timeoutSeconds: 1
  name: liveness
```

示例 5.1：用于建立活性检查的配置文件。[Courtesy Kuber-netes.io]

随着数据返回这样一个配置文件，Kubernetes 已创建启动策略的能力。这些策略会告诉 Kubernetes 在任何健康检查中失败的情况下如何去做。系统只会在 pod 级别重启，所以 pod 中所有的容器都需要重启。

你的自愈配置的细节将基于你的需求和应用程序的需求。幸运的是，Kubernetes 从许多不同的位置提供了相当多具有灵活性和可配置性的数据。

注意：自愈有很好的特点，比如每半小时重启，可能掩盖你应用程序的问题。你需要足够强健的监测和日志记录功能，以便解决可能出现的任何问题。

恢复能力

根据应用程序的需求（例如，99.999%的正常运行时间），弹性测试可以并且应该是平台的一部分。应用程序的任何级别的故障都应该恢复，这样就不会有任何停机时间。根据我们的经验，如果开发团队事先知道他们的工作将通过广泛的弹性测试，健壮的应用才是可行的。

虽然你可以通过最简单的手动方法进行一种类型的弹性测试，例如手动关闭数据库或随机地强制关闭数据库，但我们的经验证明，这些方法在自动化时更为有效。尽管 Netflix 的 Chaos Monkey 是非常强大、非常有用的运行在 AWS 上的弹性测试工具，但不幸的是它不是为 Kubernetes 建立。值得庆幸的是，在 Kubernetes sphere 还有新兴的弹性测试框架，其中有两个是 fabric8 Chaos Monkey （是 fabric8 综合发展环境的一部分) 和 kube-monkey。

例行审计

无论你落实到位多少检查和平衡，你的 Kubernetes 生产环境将从日常维护和审计中获得好处。这些审计将涵盖正常监测不涵盖的主题。传统上，审计是作为一个手动过程，但在这个领域具有功能迅速和显著改善的自动化工具。

通常，我们期望一个好的审计需要包括的标记包括：

- **容器的安全漏洞：**容器中安全漏洞的主要来源之一是包含在其中的易受攻击的相关项。应该向容器添加定期的安全补丁；确保它们升级是非常重要的，但这并不一定要像修补一个虚拟机的一部分。相反，你会发现通过更新、修补的相关条目，更容易重建易受攻击的容器镜像。
- **在特权模式下运行的容器：**虽然我们已经听说了一些容器可能需要以特权模式运行的原因，但我们尽量避免这样做，或者只允许在很短的时间内运行特权模式。对所有以特权模式运行的 pods 进行例行审计是一个有用的过程，以确保防止由于未检查权限而造成的薄弱环节。
- **无主容器：**允许容器不在本机命名空间中有所有者是一个潜在的安全漏洞。
- **镜像尺寸：**查找那些增长过大的镜像，然后重建和替换它们，记录日志文件，添加状态，然后随着时间累积，持久状态和不整洁的缓存过程都会导致庞大而难以管理的容器镜像。
- **单个复制品：**任何一个只有一个副本的 pod 可能是单个故障点的标志。

- **未经批准的容器资源库：**允许从未经注册的 registry 下载镜像是对计算机的一个重大安全隐患。
- **无效的入口：**例如，来自域或子域的“通配符入口”会使 pod 易受攻击。

注意：另一种防止引入单一故障点的方法是确保主节点被复制。

集群伸缩

对于 Kubernetes，伸缩可以说意味着三件事情：使每个节点在集群中变大（增加其计算、内存和存储），添加更多的节点到群集，或添加更多的集群。好消息是，伸缩的绝大部分是由 Kubernetes 调度处理。如果没有可用的资源，它将不会调度任何 pod。这意味着系统不会崩溃，尽管这也意味着用户可能会遇到延迟或停机时间。

在 Kubernetes 系统中，replication 控制器不断监视系统的期望状态（由其配置决定），并对系统当前状态进行更改，使其更接近所需状态。replication 控制器定义了在任何给定时间都应该运行的 pod 副本的数量。这个数字是每个 pod 根本的基础水平要求。使 Kubernetes 通过 replication 控制器管理 pod 的情况是确保所有 pod 健康运行的方法之一。

Horizontal Pod Autoscaling 是一个工具，Kubernetes 基于 cpu 利用率，用来在 replication 控制器上自动缩放 pod 数量。在写这篇文章时使用应用程序提供的指标进行伸缩的能力在 alpha 阶段，用 horizontal autoscaling，当资源被过度使用时 Kubernetes 会创建更多的 pods 应对高峰时的负荷。一般来讲，Kubernetes 会添加足够的 pods 以应对负载高峰。

伸缩在基础设施即服务（IaaS）或数据中心级通常需要分配更多的实例和更多的内存的环境，并创造更多的节点。

资源配额

在你的 Kubernetes 平台内，资源配额可以通过命令行限制一个命名空间，确保应用程序不会消耗所有的资源和影响其他应用程序，如示例 5.2 显示。

```
$    kubectl describe quota compute-resources --
namespace=quota-example

Name:compute-resources

Namespace:quota-example

Resource        Used Hard

--------        ---- ----

limits.cpu      0    2

limits.memory   0    2Gi

pods            0    4

requests.cpu    0    1

requests.memory0    1Gi
```

示例 5.2：一个样本资源配额，由 Kubectl 返回的描述配额命令。

设置资源配额可能有点挑战性。根据我们的经验，我们发现按预期负载分解名称空间，使用比率计算集群的百分比是最初的成熟方式。在这里，使用监测和审计来确定分区是否正确。

例如，假设你有 1 个 16GB 内存，8 个虚拟 CPUs，和 3 个命名空间的集群。你将需要至少 15%的内存（3GB）来运行系统守护进程，比如内核和 kubelet，用剩余的 13 GB 的分配实际负载。然后你就可以分解负荷百分率为，命名空间 A 给定负荷的 50%和 4 个 vCPUs，和命名空间 B 和 C 各自得到 25%的负载和 2 个 vCPUs。

典型的集群资源分配细目			
命名空间	**负载百分比**	**vCPU**	**Memory**
NamespaceA	50%	4	8GB
NamespaceB	25%	2	3GB
NamespaceC	25%	2	2GB

表 5.1：确定容器集群可扩展的资源分配负载，vCPUs，和内存。

容器资源限制

计算单个容器或容器需要多少资源已经成为一门艺术。从历史上看，开发团队已经使他们的估算方法比他们需要的更强大。我们尝试执行某种级别的负载测试，看看它是如何失败的，然后适当分配资源。Netflix 公司创造了这种方法“挤压测试”。

通过一个 YAML 的配置文件，一个 Kubernetes 运算提出“bids”中可用的资源，但这不是它的正式名称。示例 5.3 是 Kubernetes 的开源代码中的例子，一种容器的计算资源配置示例。这种配置定义了 pod 有两个容器，mysql 和 wordpress。这个 pod 里的两种容器均声明要求 64 mebibytes（MiB）——基本兆节，但是以 2 的次方为单位，而不是 10 的——和 vCPU 的 1/4 时间 250millicores (m)。这些要求是一般要求，但这是可以想象的，编排调度器可以超过这些要求。所以，从这个例子可以看出，在 pod 中的两容器均设置限制 128 MiB 和 500 millicores（一半 vCPU 时间）。

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
      image: mysql
      resources:
        requests:
          memory: “64Mi”
          cpu: “250m”
      limits:
        memory: “128Mi”
        cpu: “500m”
  - name: wp
    image: wordpress
    resources:
      requests:
      memory: “64Mi”
      cpu: “250m”
      limits:
        memory: “128Mi”
        cpu: “500m”
```

示例 5.3 ：为 MySQL 和 WordPress 提供一个 pod 的示例配置。

为应用程序设置适当的内存已成为一门艺术和一门科学，恰当的分配应考虑应用程序的实际需求。我们通常会考虑一些事情：

- **应用波动：**如果你预期用户数量或总的应用需求大幅度波动，那么有更多的扩展空间是一件好事。
- **高可用性要求：**如果你确实需要高可用性，并在两个节点（例如 AWS 区域）展开工作负载，那么一个节点关闭，你希望确保单个剩余节点能够处理满负荷。

通常，我们按照 60/40 规则。我们传统上将允许 60%的资源在任何给定的时间分配，留下 40%的增长空间。通过监测，我们可以观察到资源消耗高峰接近上限，并添加更多节点来补偿它。

推进 Kubernetes 和应用

关于容器化程序和容器编排系统之间的关系，有两种观点。其一是开发人员应该能够完全不关注部署应用程序的单个平台或多个平台。这是无服务器架构其中之一的标志——将开发者从配置和部署的细节中抽离出来——这种做法 Kubernetes 当然支持。

而相反有这样的说法：开发者为他们组织或团队会使用的平台建立应用的最佳方式是，使这些应用借助 APIs 检测他们自己的配置，并且动态地适应他们自身运行时的任何基础设施性能要求。Kubernetes 同样也支持这样的功能。

这是 Kubernetes 开发工程师声明该平台不自以为是的众多实例之一。但这并不是说，编排器是，或者应该是，与 pods 中包含的应用程序或工作负荷无直接关系，无论你的组织或团队会选择采用哪种方法。

相互的需求

一个健康的 Kubernetes 环境会有一个稳定的、双向的信息流从平台到应用，从应用中回到平台。这意味着这两个组件确实需要彼此联系。尽管系统层级封装的分离了应用程序和它们的主机，现在轻松地分布式应用已经成为现实，分布式架构时代创建的应用程序需要关注分布式平台。

相反地，类似 Kubernetes 的平台永远不该像对资源和连接精确配额的实体一样对待应用程序。你可能认为这是一个不言而喻的断言，但是你会惊讶于在这两个组件之间很少进行双向交流。

- **可用性：**不是每一个应用程序都需要 99.999%的可用性，但是有些肯定需要。成本和挑战与需要的可用性级别相关。你的应用程序部署平台应考虑到每个应用程序的需求。如果需要高可用性，它应该确保提供给应用程序的所有资源都是富余的——包括数据、负载平衡和服务发现。
- **可拓展性：**如果你在投入生产之前才刚刚开始评估应用程序的可拓展性需求，那就太晚了。你的开发团队可能从未考虑过它，所以很可能应用程序甚至无法处理它。考虑到可拓展性需求，将使你能够为服务设置基线，并为每个单独的 pod 设置阈值。这些指标将对监测内存和 CPU 使用情况至关重要。
- **容器镜像大小：**容器镜像有多种不同的大小。在这些容器中的服务可能会有不同的需求。精准知道什么样的服务会被使用和哪里它们开始使用（例如，内存，CPU，网络），会使你轻松合理的按大小排列容器，并且最大限度地利用基础容量。

一个健康的 Kubernetes 安装方法会为所负担的应用程序建立基线和回退模式。这将创建一致性，并确保在优化应用程序时的监测集中在正确的权值上。我们通常会遇到一些平台影响应用程序的设计的关键地方。

- **启动需求：**写的很差的 pods 不会很快启动，慢的启动不允许进入 Kubernetes 生产环境。考虑到平台的弹性本性和持续的监测，慢的应用程序可以极大地影响其健康。我们通常不允许应用程序启动超过 20 秒。此规则迫使开发人员维护轻量级服务，并确保依赖项保持在最低限度。

- **On-pod 需求与 off-pod 需求：**用 Kubernetes，你可能会发现它可以很方便的将很小的容器（称为支撑容器）当作应用嵌入到相同的 pod 里。它们往往非常小，重量轻，并且处理许多与平台相关的代码，如日志、加密和配置。

- **超高速缓存：**每一个性能良好的应用程序都会有一定程度的缓存。你将需要标准化缓存的管理方式，如果需要的话平台就可以支持分发缓存，以及监测应用程序在 pod 终止时可能需要的持久层。

- **应用日志：**我们应用多种不同的日志平台——例如，ELK stack，Splunk，Graylog，Datadog——你可以成功应用所有这些平台。你的平台团队应该负责决定日志记录应该如何发生，并设置开发人员对捕获应用程序级度量和事件的期望，如 HTTP 错误、stack 错误、资源使用级别和交易 ID。

配置管理

每个应用程序的配置都是其行为的主要驱动力。否则，很难实现快速部署，并且很难确保配置的准确性。

一个常用的方法，在管理中我们看到的是使用“外部配置”服务。除了简单的键值存储之外，现在还有微服务架构模式和解决方案，允许外部配置，例如 Spring Cloud Config。这样的配置服务通常是 pod 自身通过数据库的支持来实现，一般采用 off-pod 类型。

Kubernetes 提供其他方法去管理平台固有配置。有两种最常用方法：

- **环境变量**用来设定特定的容器级参数，这些都是直接设置在容器中，并使用 pod 级的 YAML 配置指定。
- **Config maps** 类似于环境变量，因为它们是容器级别，但具有更复杂的数据结构，如 key-value pairs。

除了拥有稳定性外，确实没有正确的方法来处理配置。我们通常使用环境变量和配置 maps 管理平台级的配置，以及外部配置服务管理应用程序级配置。

因此，你的应用程序开发人员必须从平台的需求中获得直接反馈，并且你的平台将立即受益于他们所做的更改。这是你能在 Kubernetes 环境中开发和改进应用去保证健壮性的唯一途径。不要过于坚持寻找最正确的方法，但要保持有一致性。

使能 Statefulness

Kubernetes 1.7 Beta 版引入的（因此可能不是全部可用），stateful set（使用 `StatefulSet` 对象引用）可以提供有序、稳定、优美和通常较好的存储、网络配置和部

署——非常易用、优雅的 pod 管理。对于需要以可预测的方式启动和停止服务的应用程序，这一点非常重要。

例如，应用程序 stack 可能希望数据生成和变更的服务首先在关机时停止，并在启动时最后启动。Replicated statefulsets 利用此命令保证允许支持数据共享场景，如带有读取副本容器的主数据存储容器。这使得一个有完整状态，数据持久化的应用完全适用于 Kubernetes。

注意：在开发中使用这些特性之前，一个架构应该测试和模拟启动、关闭和故障场景。这是对 Kubernetes 的要求，也是对应用操作的。Replicated statefulsets 应该被认为是一种先进的功能。

但在你考虑执行一切之前，你的应用程序需要在 Kubernetes Replicated statefulsets 中保持状态，你应该拥有一个使用公共 IaaS 的资源的完整成本评估。从本质上讲，statefulsets 是真正强大的工具，但要小心和明智的使用。

例如，对于关系型数据库管理系统（RDBMS），您可以使 stateful set，但你的 IaaS 提供了一个全面的管理系统及解决方案——甚至可以处理，或者至少能在地理上复制的，全球数据一致性问题。为了达到这个目的，一些公共 IaaS 平台甚至提供了一个 RDBMS 代理容器，或者可能是相关数据库服务供应商可以提供一个。这是一个完全隔离维护数据库的依赖应用程序代码之外有用的方法，从而减少你对 stateful set 的依赖。

协调部署

公司通常将软件开发团队与系统操作团队分开，以便确定谁对活跃代码转换负责。正因为如此，软件平台本身就有义务强制执行它支持的代码的标准。在不满足最低要求的情况下没有代码能部署在 Kubernetes 上。

这意味着应该为部署设置最低标准。下面是一些建议：

轻量的跟踪和启动时间

Pod 不应该消耗太多资源。这意味着一个轻量的解决方案体系结构（例如，非常单一），或非常少量的代码。我们建议你将内存最大限度限制在 2 GB，引导时间限制在 10 秒以内。

你可以有很好的定制的，分布式应用程序代码，但是如果你的数据团队决定直接将数据库拷贝到 pod 中，世界上最伟大的代码也无法帮助你压缩缓慢的启动时间。

足够的自动化测试覆盖率

这里的足够可能有点主观，并且可能根据你运行的工作量的要求而有所不同。也许不是所有的测试都可以或者应该是自动化的。但是，自动化测试在部署传递途径中扮演着某种角色，这一点极为重要。具体来说，你的生产线应该能够为部署的每个 pod 运行一个完整的、唯一的测试套件，越快越好。如果没有满足需求，每个测试都应该能够让部署失败。我们严重依赖于这些类型的测试：

- **单元测试**应该在构建过程中运行，并适用于最小的代码块。这里，你不用测试所有代码。你甚至不应该测试大部分代码。你只应该测试嵌入代码。作为构建过程的一部分，实现单元测试从运行它们开始。你将它们扩展到集成测试的所有依赖项和消使用者，包括功能测试，回归测试依赖的环境，以及轻量级，有针对性的端到端（E2E）测试。这使小的版本发布时有更充足的信心。
- **综合测试**适用于所有功能和模块的测试，确保它们按预期工作。在我们的测试经验中这可能是最被人误解的类型，尤其是在微服务架构环境。对我们来说，集成测试是关于集成功能和模块的。它不是关于整合微服务架构——来自功能测试的。Contract 测试（如 Pact）是一种真正强大的集成测试方法，因为它允许对正在部署的代码模块进行非常有针对性的测试。
- **功能测试**确保所有依赖的组件（通过即将部署的 pod）正常运行。这是通过测试的输入和认定的输出完成的。这是第一次可以开始将所有的上级依赖关系和——至少在理论上，所有下级依赖关系共同组合的测试。
- **端到端测试**模拟用户如何与生产部署中的工作负载交互。这些测试往往体量最轻，而且冒烟测试与其他测试一样重要。

自动化部署

手动部署通常是危险的，很少产生一致的结果。Kenzan 通常使用自动化平台部署工作负载，如 Spinnaker 和 Jenkins。通过一种称为“canary release”的技术，工作负载在几个小时内慢慢地滚动到生产中，其错误率被不间断监测。如果错误率一直很高，这个过程可以触发自动回滚。你应该避免构建任何需要手动干预来回滚的系统。

结语

选择 Kubernetes 作为你的容器编排解决方案，解决了许多组织机构如今面临的容器管理中存在的问题。本章强调了许多在生产中运营 Kubernetes 时组织机构所面临的部署和操作问题。接着讨论了如何利用 Kubernetes 使微服务器应用架构成为可能。对于容器化，云原生 apps，Kubernetes 不仅仅有容器管理的能力，还提供了大部分平台作为服务（PaaS）类的功能，再加上很好的操作弹性支持。

Kubernetes 生命周期运维

Kubernete 现在面临着以往 Docker 面对着的生命周期循环痛楚，即多并行版本问题。如今我们已经能用上 1.7 版本了，然而其他版本仍在活跃使用。伴随 Kubernetes 使用率上升，会有多种版本部署在组织或团队的 IT 项目中，一些会有 3～4 年时间。

“我们正在尝试让每一次 Kubernetes 发布版本都尽可能稳定。”，Kubernetes 项目 SIG Caleb Miles 这样说。“在新版本发布前，我们会保持代码不变和稳定周期，去除掉不成熟的新特性。”

在这段 Podcast 中，会学到如何使 SIG 项目管理变得一致和并行，而避免特立独行。**播放 Podcast**。

Cale Miles, *CoreOS 项目技术经理。他专注于为 Kubernetes 开源社区贡献者提供更佳的体验，处理发布和项目管理。在加入 CoreOS 之前，Caleb 曾工作于 Pivotal 的 Cloud Foundary，当时他关注的是运维和提升 Cloud Foundary 基础平台特性。再早之前，Caleb 曾撰写过有关 Ceph 分布式存储系统，从事有关 S3 兼容对象存储接口开发工作。*

Kubernetes 未来路线图

作者：**SCOTT M. FULTON III**

自从我们用 IBM, RCA, Sperry 和 DEC 搭建计算平台时就会被一直问到同样的问题，这甚至是从开挖我们总部大楼的地下室就开始了：当这些机器到寿命了会发生什么？他们的程序在那时会在哪里运行？这些新机器会以同样的方式替换旧机器和旧程序吗？

尽管这些设备与锅炉和洗衣房共享地下室，我们当时并没有把它称为基础设施。 在二十世纪八十年代的某个时候，英国政府的某些学者开始使用这个术语来描述作为普通商品的信息技术。这种称呼渐渐成为一种主流认知，甚至历史学家都没能记录到是谁最早提出的这个说法。

随着时间的推移，他们把这些规则编写成了一种平台语言。把信息技术基础库（ITIL）代码化到文档中，也许是最先采用这种方式的工作。这部书的主旨是：把技术作为服务来交付，把管理过程做成流水线，对制度进行持续改进。

构建一种基础设施服务（IaaS）并不像是聆听能够改变世界的 TED 演讲那么激昂，而更像是在世界上出现了一个带着病毒的猫的 GIF 图片。IaaS 实际上是一种我们一直在的某种工作，而它所带来的变革已经完成时，我们才意识到变化的事物。

“我认为编排工具真是太神奇、太棒了”，Laura Frank 说，CodeShip 持续集成平台总监。“因为它太神奇、太出色，还是一种全新的事物，（对于）工程师（而言）——也许是我们职业中的一个缺点——经常是会把事情搞得复杂。我们总想找到所有新事物并探寻到使用它们的理由。”

“我认为很多时间，工程师团队选择的工具复杂度都远远超过了实际需求，” Frank 继续说，“并且他们还经常尝试去解决他们还不确定的问题。我不认为这对那些工程师是必须的；我认为有很多人都会去做这样的事情。但只有一小部分工程师会把它说出来。”

还是感谢那些工程师的梦想和愿景，尽管它们是如此复杂，在最近七十年时间里，计算在我们身边已经变成云的方式。我们只会认为我们正在使用的超级计算机就在自己的口袋

里，但事实上它们是在云端的数据中心处理完成的。它们可能就在我们的办公大楼里，或者分布在各地方，诸如 Equinix 或是 Digital Reality，或是以虚拟设备方式来自 AWS 或 US-West-1 或 2。但他们确实是我们世界真正的一员。我们的智能手机和平板电脑正在呈现这些美好的事物，与此同时，Siri，Spotify 和 Slack、Salesforce 已经装配到我们家庭中，就像之前（智能手机和平板电脑）带给我们的一样。

“如果我们想让一个项目持续进行下去，我们就要允许灵活性、多样性和开放性，目的是让我们不再遇到之前的麻烦。”

— Brandan Burns, Microsoft

改变其实很少。

我们想要问计算平台或是我们正在调查的问题，无外乎是有关虚拟化平台的，如 vSphere, Mesosphere，OpenStack 和 Kubernetes，对比于上世纪六十年代的话就是——RCA501，IBM 701 ——或是后台为中心的上世纪 90 年代——IBM AS/400，DEC VAX-11。这些平台都经历过自己的从重视开题调研到投资建成的关键生命周期过程。不久的将来，又会有人问，我们将会用这个平台吗？

什么真正改变了——突然和激进的——才是我们答案的内容。它将解释如何并且为什么今天我们的计算基础设施在短短三年时间里就发展得如此多样，而计算机工业已经发展七十多年，而它们第一次的出现的时候就像是天外来客。

谁创造了 Kubernetes, 是真的吗？

“应该选择那些能够为应用带来价值的工具”，CoreOS 的 Calel Miles 说到。“你看到的是一些组织或团队利用 Kubernetes 为它们的工作提供了更大的价值。我们了解到的有些组织机构通过使用 Kubernetes 上游的多种多样的其他工具和项目为他们所在的组织机构带来了更多价值。”

“我相信 CNCF 在这里发挥很重要的作用，” Miles 继续说，“作为一处净土，正在协助组织或团队的云原生改造进程——衡量应该使用什么工具，去解决他们面临的挑战。提升程序人员生产力，提高他们的开发速度，改善操作透明化程度——我认为 CNCF 是组织机构（云原生）改造进程中，可信赖的合作伙伴。”

“Kubernetes,” Google 的 Tim Hockin 这样描述，“是一个由百万行代码组成的一系列代码仓库，是每周都拥有数以百计的活跃开发人员的项目。”他是 Google 的核心软件工程师，也是 Kubernetes 的联合创始人。“这几乎不可能是一个人可以胜任的，不论是涉及的各类 Pull Request 的提交和分析出现的 Bug，实际上，我们在实际分析和参与解答问题上只花费了很少的时间。”

Hockin 成名于他对 Borg 的贡献，这是一个 Google 内部使用的容器编排系统，目的是为了改进应用程序的部署，它也成为了 Kubernetes 最初的原型。结合对 Docker 的贡献，他成为了分布式容器领域有影响力的热门人物。然而，Hockin 并不满足现状，引入了当前容器系统所面临的复杂境况，开发社区里已有很多被他影响的人。

“我认为可以称之为’Kubernetes 星系’”，来自 Google Kubernetes 产品经理 Aparna Sinha 说。“因为它是一个很大的项目。从很多角度来衡量的，都是 Github 上最大的项目之一。”

Kubernetes 是云原生计算基金会（CNCF 其自身也是 Linux 基金会一部分）的一个开源项目。按照设计，CNCF 上层不会由任何一家企业领导或维护。然而来自 Google 的工程师，却在这个组织机构中扮演了重要的官方角色，Microsoft 在 2017 年 6 月也加入了 CNCF 并于 7 月成为白金成员。Cisco，DELL 科技，华为，IBM，Intel，Red Hat，CoreOS 和三星也同样是其中的高级或白金成员。

“对于新成员来讲，这些兴趣小组成员看起来有点眼花缭乱。——有二十多个 SIGs——可以试着数一下，‘我遇到一个问题，我有一个疑问，该属于哪个 SIGs（Special Interest Groups 兴趣小组）？’”，Hock 承认。“但我认为我们做了有成效的引导工作。并且随着管理结构规范化，每一个 SIG 都自成体系，都有各自的负责部分，涉及范围和与其他 SIGs 协作的切入点。”

权力中心在哪？

表面上，从整个项目来看，兴趣小组基于每个厂家所主营的业务范围，来作为结构制定的依据。然而，各类高手都在当今的 IT 领导层——数十年老练的 Windows 调试人员和数年的金牌级虚拟机专家——是这些人使得我们明确怀疑这到底是一个“开发社区”还是一个真正意义上的，更准确地讲是一个”开发管理委员会“。

但是 CNCF 毕竟不是联合国，或是公司的董事会。相反，这个组织机构尝试去区分用户对 Kubernetes 的兴趣，以及管理 Kubernetes 和其他项目的兼容性工作。广义上讲，这些用户都是典型的，或是潜在客户。然而区分用户首要需求或是亟待获得的特性是 CNCF 来驱动整个平台开发战略的目标所在，即响应不同的用户所需，现在虽然还不能自动化，但是未来却有可能做到的。

这是一种非常不同，较于某些以柔性或市场化做为目标，更能适应形式或功能变化的，可立即定制的平台。我们之前会经常看到这样的工作吗？

“在当今的云原生应用中，不是所有人都准备去实现 12-factor。我们正在尝试开发新版本的 Kubernetes 来让用户觉得值得去部署。”

——Caleb Miles, CoreOS

SIG 小组成员 Sinha 解释说，“已有的小组由来自不同公司、跨职能的个人，他们做为一个独立个体，围绕着同一个兴趣话题组织在一起。每一个小组都会编制当年兴趣小组的路线图。因此我认为，这就是有时无法看到一个的”项目级别“主题出现的原因。”

路线图在每个兴趣小组看来是日程表或是章程。它解释一个将要展现给世界的功能是如何按计划来实现的。

“你可以把这些兴趣小组看作是自己的独立星球或是一个类似太阳系的结构”，Sinha 说到。

Google 的 Tim Hockin 会融入各类项目中心做为评判成员，他十分愿意去分担一些问题——就是那些围绕 SIGs 的——将会被委派给他人去做的事情。“从这点来看，我不需要参与到每一个问题”，Hockin 说。“团队的开发成员来自 Google 和全世界，他们非常可信，我相信即便我不去关注，他们做的事情也是正确的。”

“项目大小确立是我选择的需要关注点”，他继续说，“这对于我来说有点不同，因为我愿意参与任何事情讨论，并且我愿意去理解任何正在发展的事情和那些人们正在制定和猜疑中的判断。这些问题伴随着项目发展，有的注定无法实现。”

“我们可以用一整天时间来讨论工具，框架，或是流程。实际上，这些都源自高效工程师组织或者团队，” Codeship 的 Laura Frank 如是说，” 我认为归根到底，是程序员生态。给工程师更多主导权，或者让他们感觉到主导权，超越他们（所认为的）写代码和交付的首要任务。这很关键，当你想要和程序员在代码中谈快乐，生产力，或是投资，确保（主导权感）这是最合适的。”

“很长一段时期，当人们还为 Kubernetes 复杂度而却步的时候” ，Frank 继续说，“我总是鼓励他们去想一下 Kubernetes 来自于哪里，为什么它会存在和它会去尝试解决的各类问题。并且我经常这样说如果你遇到了类似 Google 的特定问题，或类似 Netflix 的特定问题或是其他大型应用系统所面对的巨大的可控范围问题，Kubernete 的表现是很出色的。你也许很想说我有我特定的问题，也许没有。但从稍微小的和可以运行的事物开始，对于我们来说是有益的。”

通用的部分

有一个使人困惑的事：一个组织或团队总有一个自然的领导 —— 或者，更坦然地说，总有一个被这样提名的人—— 不能用以人为本的方式去领导团队。为了弄明白一个实体的可用功能，你也许不得不去对一些最不寻常的事物，针对企业级计算平台期望需求分析：解析平台里的每一个要素到一个自成容器的组件里，它们之间或许会发生交互，或许会被其他组件替代而产生比交互更好的效果。Kubernetes 成为一个构建分区块交互的平台，终极结构是它自己成为一个在不同层级的构建区块系统：一个成长中的 CNCF 主导项目列表，类似一个技术栈。

“我们已经开始拆解这个‘巨石’ kubernetes/kubernetes repo 了” ，Hockin 进一步解释到，参照这个 GitHub 链接可以定位到中央代码仓库。” 我们正在每一个仓库中注入可自成体系的事物；我们已经知道哪些应该做为新的规则引入到 Kubernetes 仓库，哪些不应该被引入；我们已经按照一些架构图和文档去设置一层结构，将做为核心的一部分，还是生态系统的一部分，这也会对我们如何模块化和代理方式访问的决策提供指引和帮助。”

这个编排器假定一个系统拥有了可交互的构建模块。同理，从另一个层面来看，也是一个完整的 CNCF 项目集合。但是整合到单一架构的本能想法，理性的进行架构可能会很重要，但有些时候，Kubernetes 架构所做得会像是反常理的。

“Kubernetes，从最基本说起”，Hockin 继续讲到，“就是许多组件在一起工作的形式，但也不像是一个庞然大物。我们已经有了 5，6 个主要概念化区分的组件，虽然存在一定数量的重叠——但他们在一起存在着开发和测试的价值。并且，他们已有不同团队的人员为之工作，这体现在我们的 SIG 结构体系里。”

Hokin 痛恨闭门造车。许多人所生活和工作的组织或团队不断隔离他们的工作过程，声称他们痛恨单干。这就是 Hockin 所阐述的，他希望各 SIG 避免完全的工作划分。但是一些隔离是必要的，他确信，只要是为了程序员和贡献代码者为了实现手头上的目标时，就存在一些他们独立工作的时期。他把这类的隔离看作是“柔性分隔”。

必须清楚，一些 Kubernetes 的代码贡献者并不是这些主要供应商、服务提供方或电信公司的工程师，但是这些“雇佣”的程序员们确实可以通过连接到云环境来构建他们的网站，也可以获取所需要的 IT 资源。

“一些网络安全供应商关注协同工作时说到，‘看吧，我们需要解决这个问题，但是我们需要用一种交互操作的方式来做这个事情。’”

——Gabe Monroy, Microsoft

这就是 Kubernetes 项目的管理结构，Hokin 提到，代码按功能划分到组，在每个组里面各自工作，而不是按照任务划分到每个成员的。各 SIG 小组有责任和主导权，并且他们都有技术主管。只有这时才会有一个决定谁将成为领导的过程。

“这个结构实际上会让程序员和项目运维人员都获益”，Hockin 说到，“这样我们可以知道哪里出现 Bug，哪里变化有冲突，还有哪里工作需要别人配合。我不期望一个最终用户——某个人并没有加入我们的社区——为我们的结构提出意见。这就是我们为什么会有一个主 kubernetes/kubernetes repo ——这是一个中心，接受 Bug 提交，答疑和发布消息。”

做为贡献者的应该是这样的，他继续说，他会直接找到合适的 SIG 或邮件列表——就是这种分散交互的形式，让 Kubernetes 贡献者知道谁是他们期望的用户，实现者和客户。

不像是操作系统或浏览器的情景，以数据为中心的部署结构不能保证安全的自动更新——不能排除平台上存在不稳定的驻留服务。因此新版本 Kubernetes 不能立即加载旧版

自动检测漏洞——也许，某些操作系统会支持。Caleb Miles, CoreOS 技术经理，是 Kubernetes 项目管理 SIG 成员，同样也是新设立的 kubernetes/sig-release 小组的核心成员。Mile 的职责对于项目而言是一个特殊的领域：他会去协助保证将要发行的平台版本的稳定性（截至写稿时，Kubernetes 当前版本为 1.7.3）同时，他实际也会维护与旧版本的持续集成。

“我们会保持一定的代码冻结和稳定的周期，在新版本发布前用来去除任何当前没有准备好的部分，” Miles 说到。 “但我们同样想让每次重要版本发布能够满足公司、用户的需求，同样，也想让社区尽快适配新版本。确定是否迁移某些重要 API 或资源从 alpha 到 beta 直至 GA（基本可用），或是标示出某些在实际项目的关键难点，这会让实际部署工作任务变得更加困难，用户现在必须去面对的……并不是所有人现在都具备条件去部署 12-factor 这样的云原生应用程序。我们尝试在新版本 Kubernetes 中让用户认为值得去部署的功能。

谁该优先转变？

实现这个目标的挑战实际来自于具体业务的需求而不是技术方面。许多人都知道他们的系统并不能满足需求。如果需要某些人做出改变，业务领导相信，一定是 IT 部门。

“我们从客户所看到的 IT 关注的话题已经转变，” Brian Gracely 讲到，RedHat OpenShift 产品战略部经理（商业化的 Kubernetes 平台）。 “人们总是说， ‘好吧，我需要一些价廉物美的东西’ ，不管是的成本驱动或计算能力驱动。在最近的日子里技术人员在一起，围绕相同的话题——不管他是不是 IT 或是业务线上人员——都一同说到， ‘这就是我们需要如何改变的业务。这是我们要如何去细分典型市场、供应链或是我们的客户交互已经发生了很大改变……我们不能继续像十年前那样去做这件事。’ ”

这里没有太多的供应链在转换或改变，Gracely 确信。事实上已经改变，好像一些人最终触动了控制室的开关而没有或是忘记发邮件给 IT 部门。

“不像是供应链中每件事情都处理的大应用 “，Gracely 说， “而是由不同的各司其职的服务组成的。而供应链就是 由 API [应用编程接口 application programming interfaces]组成。”

从软件开发人员的角度来看，API 是一个用于去连接服务的端点。用来在终端之间交换的协议是接口。从业务分析人员的角度来看，API 是供应链中要素之间的连接点。对于软件自身基于 API 来适配于特定服务的重要意义已经在业务中显露。

数字化改革，对于今天应用的软件创新如此必要，已无需言表。事实上，Kubernetes 已经紧随在角逐接下来的改革的台阶上了。

“如果你回顾一下 2017 年的全部主题，其中某一重要的主题是扩展项目，” Google 的 Aparna Sinha 说。 “这是我们桌面上的筹码，但我们要确保贡献者可以容易地贡献项目代码；为了保证这些，我们将项目进行多层架构；这样会使项目的核心会变的越来越稳定，因此我们的用户可以信赖它。”

是谁或是什么为 Kubernetes 提供安全？

Kubernetes 内核的稳定程度是其可信赖和可适应的本源所在。如果开发人员能够基于稳定的内核开发外围功能，那这些功能也可以仰仗内核对于它们的支撑。

这是一个美好的理想，但还是有一些本质内容，被观察家或历史证明：依赖去扩充的平台会抛弃与功能组件的交互，使功能组件置身于自身完整性的风险中。从更深层次说，这就是为什么 PC 工业可以屹立很久而移动或云计算工业却还达不到的原因所在。

“人们发现了平台的价值，是为了给他们的用户带来价值——同理，Linux 是一个通用系统，而不会涉及分布式方面很多内容。”

——Caleb Miles, CoreOS

“我认为我们有责任为用户提供一个安全的环境，” Brendan Burns 说，他是 Kubernetes 的核心工程师之一，现在是 Microsoft 的伙伴架构师。 “但在一些层面，安全是一个连续体。我们想保证用户不要误伤自己。但我们同样明白，这同样会依赖你正在使用的数据，同样会依赖人们想用到的工具插件，配置插件。”

“我认为我们应该这样保证，当用户应用了安全策略和 API 后，我们希望这些 API 可以正常地，为即将发布和提供给用户的服务工作，” Burns 继续说道。

人们所看到的平台

这是一个更深层意义的信息：Kubernetes 将不断在大品牌供应商的各种面向客户的平台中出现。这些供应商会向传统的企业 IT 决策者展现编排器和更多地传统业务场景，以及单一的技术支持来源。 但也可能就是这些供应商向系统提供必要的安全措施，以便企业采用，同时保持其已有的政策机制下必要的安全方法。

“Kubernetes 作为一个平台，允许很多系统去集成，” CoreOS 的 Caleb Miles 解释到。Miles 引用 GitLab 开源代码审核项作为例子来推进 Kubernete 整合， “为了提供更高层次的平台体验，团队平台使用了 GitLab。 “

” 因此 Kubernetes 平台尝试去不那么固执地要项目与 Kubernetes 进行交互， “他继续说。 “应用程序开发者可以将代码直接提交到 Git；GitLab 会选取其中的变化，开发人员可以执行定义好的测试；同时在运行完测试之后，可以将应用直接部署在 Kubernetes 上。”

Miles 所说的核心是描绘了 Kubernetes 很紧密的整合了应用程序所需要的东西，而不是其他可行方法。 “这会给第三方平台或供应商构建提供丰富的基于 Kubernetes 软件生态环境，有机会给他们的用户提供预期的体验，” 他这样说。

GitLab 吸纳 Kubernetes 作为可交互的编排工具，成为它持续集成部署（CI/CD）系统的引擎。现在 Kubernetes 工程师正在以相同手法，开始架构更大的安全系统，尤其是以当前网络安全策略模型作为参照。

人们相信的策略

虽然策略驱动安全的概念已在本世纪之前面世，但对于很多企业而言，还是个新事物。但对于 Kubernetes 而言却并不新——那些 IT 决策人员，可能已经在上周，就经意识到它的存在。这种新的安全潮流被装配到控制器功能里，作为一种门禁服务，在初代的虚拟机

器中保护处理器不会执行破坏或恶意的代码。当用 Docker 容器方式来替代虚拟机（VM）后，对于这种早期的安全系统的生命周期来说，已经不再适合了。

然而，基于 Kubernetes 环境的虚拟机可以采用管理器基础（hypervisor-based)安全保护，以同样方式保护在虚拟机中的应用程序，这同样会严重制约组织或团队在可伸缩性和跨云部署中做出选择——Kubernetes 占据首要位置的两种优点。

“Kubernetes, 起初，是许多在一起工作的组件。所以，它没有必要看起来像一座庞然大物。”

——Tim Hockin, Google

在当前 Kubernetes 的结构中，是谁或什么决定平台的安全特性和功能？或者是否有一些特定的兴趣小组被委托负责关心特定人群的安全问题？

“我想说一下首要策略，” Google 基础设施副总裁 Eric Brewer 回应，” 在大多数地方是有所谓的尝试，但安全不在讨论范围内。” 这意味着什么？ “我们这样假定，举个例子，你正在一台虚拟机中运行着你所需要的应用，” Brewer 说，“并且它已经考虑了一定范围的攻击问题。有另一种情景——一个明显的例子——好比代码确实经过仔细检查还需要同行评审。所以我想说，架构师的第一要务是要知道哪些地方需要严谨，而哪些地方可以自由发挥。一个好消息是，绝大多数情景下，你可以放手去做，这样很好。但确实有一些情形下，诸如你如何挂载一块存储卷，或是你如何去确认分区，或是你应该允许哪些作为网络介入的位置，这些是不得不去十分小心的地方。最好是让很多人关注的这些事情。”

人们需要知道的认证

认证——特别来讲，是一个数字化认证代理系统——即今天许多的 Kubernetes 安全项目所关注的内容，在一定范围内是争议的焦点（直到有其它需要关注的事情）。从版本 1.6 在 2017 年 3 月引入以来，编排器已经支持基于角色的访问控制（RBAC）了，在实例中对一些关键权限的角色描述成为了认证的方式。

所有这些说明，认证并不是 Kubernete 的原生组件。这里有架构方面的原因：认证是一种建立在假设持久化状态上的标准。在安全的基础结构中，认证对一些基本可信的现存事物是开放的。根据定义，一个容器是短暂存在的。它的持续存在是信任受限的。

说的更多些，一个认证是一种状态——一种由数据产生的特定属性或特性的展现。

“这有一堆认证和保密的工作需要我们去做，” Microsoft 的 Brendan Burns 说。 “我们真的没有完整地认证库。我们仅仅是开始去做，像为 Azure 容器服务添加 AD 整合方案；我们仅仅是刚去思考它，什么是所谓使用云端提供的保密服务，就像使用 Key Vault，为你的集群提供加密？”

在一个更容易想象的世界中，应用程序可以通过数字证书的方式向身份验证系统标识自己。 在一个由大量的微服务组成的应用程序中，许多应用程序是冗余的，分散运行在云中，故意设计为无状态的，这时候单个证书可以满足认证的需求么？

况且如果你考虑的更远一些，加密应用，从定义上讲，是一种状态。逻辑上说，这根本不是一个无状态系统；一个加密应用，是独立地维护了一个实体，它来源另一个实体，通过一些关联，使两个组件形成一个数据单元。想象一下共享于你的浏览器和网络服务器的会话。现在设想一下同时出现的、数以千计的文章在微服务中共享。如果一些服务转瞬即逝，系统如何知道哪些剩下服务正在共享什么加密服务？

“通过安全的 pod，每个 pod 都有自己的可验证的认证……思考这个问题的正确方式是，想象一个单一租户的 pod 在一个多租户节点情形。”

——Eric Brewer, Google

如果这个基础问题的答案是“不可管控” ，就是 Google 的 Eric Brewer 所建议的，那么解决方案是——不管它来自哪里——都需要被非常明确地集成到 Kubernetes 中。对待此事不能亡羊补牢；这种情形应该立即指出，并且进一步增强系统的基础架构。否则，在安全组件和编排器关联之间，总会存在最弱的短板。

“Kubernetes 提供了一定数量的、非常灵活的多种机制，” Google 的 Aparna Sinha 解释。 “你可以使用不同的实现应用到不同的云里，或是不同的特定环境。不同的机制，很显然，用于认证，授权和身份；不但适用于基于角色的访问控制，同样适用过去的，基

于属性的访问控制。因此对于安全的容器接口而言——需要通过添加安全策略来实现，无论是使用 AppArmor, 或者 SELinux 或是其他 Linux 实现。”

CoreOS 已经帮助 CNCF 编制 Kubernetes 安全发布处理方案（Kubernetes Security Release Process），这是一个由志愿工程师组成的小组，一旦由他们发现了漏洞，他们会一起配合去响应，并去负责去修复。这是一种从开始打补丁到修复的自动化工作流程。但是作为编排器的架构师已经公开提到，对于 Kubernetes 的发展之所以可以被客观地称之为一个平台，需要采纳一些基础的认证工具，这里包含可靠且唯一地识别了节点，pod 和容器的方法。

“还在不断增加中，” Sinha 补充道，” 我们正在为节点安全和 pod 级别安全做些工作。Kubernetes 的路线图是拥有可靠的、可扩展的安全能力。任何特定的实现，在特定的云环境里，会采用不同的能力层级。 “

人们对可扩展性的期待

一些原因是，围绕 Kubernetes 实现可变化能力方面，最近的主要是改变为可扩展模型。

“我们有一些旧观念，所谓第三方资源（`ThirdPartyResource`），” Brewer 解释。“这是一种人们使用的基本扩展机制——诸如 CoreOS 系统，以及其他人正在做的基于第三方 `ThirdPartyResource` 的东西，这可以让你以某种形式扩展原生的 Kubernetes API。”

“某种尝试已经失败……一些失败的尝试者表示，“他承认。 “因此已经引入了，并且我们已经有了新的替代方案，称作 ‘客户资源定义’ `CustomResourceDefinition` [CRD]。这是一种新产生的想法。”

特别地，CRD 使得外界服务能够标识出一种编排器可以像 pod 同级别对待的（也就是原生级别资源）资源。因此我们已经可以如此去思考作为一个 Kubernetes pod、或是一组容器的维护人员，凭借这种可扩展性，它可以同样实例化和编排其它底层结构。

“在一定层面，安全是一个连续体。我们要确保人们不要用它来误伤自己。”

——Brendan Burns, Microsoft

“这会提供给 Kubernetes 的第三方扩展一个新的机会，” Brewer 讲，“确实真是如此，表现得就像原生的一样。同样会给我们的模块化带来机会，同时还有供应商提供我们之前根本没有想过的创新和做有创意的事情。”

如果最终的对于编排器的安全模型确实尚待设计，那么 CRD 对每一项可设想的主意都会打开闸门去规划模型应该是什么样子。比较典型的，一个安全模型，对于一个平台而言，必须遵循平台现有的运行环境。CRD 在这个运行环境的架构层面展开，让外界组织或团队加入基本架构图的探讨，如果你愿意，同样可以用全新的想法去表达别出心裁的想法。

就像 Brewer 将要说到的，Kubernetes 已经标识出一个计算处理的安全边界——简称“计算边界”——即虚拟机。“这就是一些已经被验证和证实的跟踪记录，”他说。

“最新的进展是，Kubernetes 已经做到使一个集群成为单一实体，在某种意义上实现了安全。如果你已经进入到一个集群，你就已经进入到集群的所有节点。在 Kubernetes 1.7 版本，我们正在把节点做为一个安全边界，隔离同一集群的其他节点。良好的粒度对于许多不同的用例是重要的。但它同样基于以实体机实现的边界。”

没有人会期待的排列组合

Brewer 说到他个人比较青睐，pod 做为自然的安全边界主意。一个建议是他已经把安全的 pod 提给社区——一个扩展的基础 pod 类型，它的行为从实质上有别于基础 pod。理论上讲，他说，标识和认证基础资源应该由一个安全的 pod 来管控，它的生命周期按照自己规则来定义。

pod 会用一种新的排列方式：举例而言，嵌入的虚拟化——就是 pod 里的 pod。或者可能是一个单一 pod 可以与多个虚拟机同时存在。为什么？在一个架构师推崇的无状态环境里，认证——最终在一个系统中的实例——可能非常难商妥。而且 Kubernetes 甚至

在尝试通过 etcd 来解决有状态问题——这被一位 CoreOS 的开发人员称为”状态看守者”——标识一种不能用键值方式来展现存储结构。

“我认为，这是我长期以来一直想要的，真正的云——不是基于云中的一个数据中心到另一个数据中心搬动机器方式。”

——Eric Brewer, Google

正在实现的这个准维度模型，在哪里可以让有状态和无状态共存，需要 Kubernetes 社区这边做出决定，就像标识和定位新的安全边界一样——引用策略的实体。已经提供了，Eric Brewer 说道，边界已经从集群移到节点，虽然在将来他希望更加细粒度的控制。“这是我将努力和社区一起去努力确立的事情，” 他说道。

“对于安全的 pod，每个 pod 会有自己的可验证的认证，并且它不能干扰其他在同一物理机上的 pod……正确的思考方式是，在多租户节点上运行一个单一租户 pod。”

用户可访问性需求

在这方面的可验证性要求 Kubernetes 的工程师不需要在这里重新发明轮子。无论目前在哪个系统中认证现有数据中心的实体，该 Pod 的安全部分都需要被识别。基于此种原因，编排器的工程师们会在将来适配平台，接纳已有的网络策略，不论它们以何种方式存在。

“我认为网络策略 API（the Network Policy API）的产生过程就是一个很好的参照模型，” Gabe Monroy, Microsoft Azure 容器基础项目经理，同样也是 Deis 前任首席技术官——作为 Kubernetes 发布管理器制造商的 Helm，最近已被微软收购。

“与把所有事情运行在大而全的供应链应用程序上不同，由服务组成的应用会带给你多种不同的模块和组件。而它的供应链就是 API。”

——Brian Gracely, Red Hat

“在 Kubernetes 早于 1.3 的版本里，网络访问控制和集群的入站/出站策略默认是基本开放的，” Monroy 继续说。 “一些关注网络的安全厂商聚在一起说， ‘瞧吧，我们需

要修复这个，但我们需要一种合作方式来做。’ 这个工作的结果是产生了 Kubernetes 网络策略 API，基本上指定了一个架构和一种在一个集群中默认否定的访问策略方式；还有一种 Kubernetes 标签选择器，用来定义哪些容器可以相互交流，同样应用于网络的入站和出站。 “

Monroy 所引述的这个他所青睐的模型，与其说是策略模型，倒不如看作是协作模型更合适。参与其中的众多安全供应商，他说，使他们能够想出的不是一个最常见的政策标准的系统，而是一个具有分离和可互换组件的平台。

CoreOS’ Clair 是一个以收集容器漏洞数据为目标的项目，将数据的使用作为一种保护机制。CoreOS 是 Kubernetes 网络策略 API 的参与者之一。另一个是 Twistlock，他们的基础产品是一个商业化容器安全平台。

作为 Twistlock CTO 的 John Morello 提到，对那些公司而言，在他的位置，为 Kubernetes 编排器设计安全架构，是过于专业的，没有任何帮助的。 “就像我们为 Kubernetes 构建独立的应用，对比 Docker 的 Swarm 或对比 Mesosphere DC/OS。这里面有更多的标准化，就像是编排器所说的方式一样， ‘在我运行任何事情时，有一套标准方法使我可以去查看一些外部服务是否应该运行……’ 这使得它为客户做集成时和其他工具在一起联调更加简化。”

现在，使用 Twistlock 软件的客户可以通过安全终端发布 Kubernetespod，并且可以同时用一个特定的 DaemonSet 方式部署 Defender 代理到多个节点——即一种给编排器在每个节点运行特定 pod 授权的策略方式。但是不断完善，现在需要企业去大量采用的方法，可能更像是 Kubernetes 特定的。通过研究一套更加抽象，多层次，解耦的机制，Kubernetes 应用开发人员可以确保这些策略可一运行于外部的平台，在理论上不需要特定为 kubernetes 设计调优。

“每一个安全供应商都会为此提供不同的后台强制策略，” Microsoft 的 Gabe Monroy 说到。 “我认为这是一个值得推行的样板；也就是说，在一些实例方面，大家在增加它们后，会得到一致的结果，同时他们也可以在 Kubernetes 社区管辖的范围之外做一些事情。我认为我们也应该欢迎这些，因为这也是保证我们一同去实现目标的某种创新形式。”

平台将会在三年时间成为什么？

当命名书名为“Kubernetes 生态系统状态”时，我们清楚这样一个关于生态的假定——创建可循环的发展状态，目的是让所有参与者从中受益——围绕着 Kubernetes，我们会继续这样做下去。当然这实际上仍是一个开放的问题。

“在很长一段时间里，当人们对 Kubernetes 的复杂度而却步时，我总是鼓励他们去想一想 Kubernetes 从哪里来，为什么存在，以及会尝试解决哪些种类的问题。”

——Laura Frank, Codeship

在问及如何打造专业的 DevOps 时，人们都会认定 Linux 系统在数据中心领域是独领风骚的——通常会把 Windows Server 排除在外——很多人会告诉你在创建平台方面 Linux 会比其他系统带给你更多选择，而且那些服务器端或是云端平台，已经运行在 Linux 系统了。在历史的一个特定时点，Linux 就已经在这（服务器和云端）征途中了。

一些人也相信同样的历史会在 Docker 重演。同时很多人也以此类推到 Kubernetes 上。

“我确信，基本上，Kubernetes 是云原生革命里的一小部分，”Google 的 Tim Hockin 这么说，“希望它能做为催化剂让人们去考虑如何运行他们的应用。但是就像 Linux，这不是它自身的最终目标。现在，三年时间并不是一个很长的时期，但我认为接下来的三年，我们会沿着云原生化的道路做更多有益的事情，我们很期待更多的企业去采用容器和 Kubernetes。这些事情做起来很好，但企业中开展起来会很慢。”

Hockin 希望在 2020 年看到，命令行中定义了 Kubernetes 的缩略语，比如 `kubectl`，成为了广泛采用的引擎，客户端工具集——比如，像是 ksonnet，一种 Google JSON 模版语言实现称作 jsonnet，它编译成可用于 Kubernetes 原生 YAML 类型的，更高层面的声明化代码。他相信，随着越来越多的可供开发人员使用的客户端封装好的操作工具的实现，就会有更多的组织或团队，即便是在并不十分清楚内在机制的（Kubernetes）前提下，采用 Kubernetes 技术。

“Kubernetes 的安全能力路线图是更加健壮和广泛。任何特定实现，特定的云环境，应用这些能力也会有所不同。”

——Aparna Sinha，Google

“正如我希望的，在这三年里，Kubernetes 是按着这样的预期而存在，”Hockin 说到，“并且它成为了工具箱中，像 Linux 服务器一样的工具。是否还记得，当 Linux 服务器做为一个念头出现时，你觉得它是多么的不可信、有争议，然后把这些告诉你的老板。但是现在，当你谈论起服务器，Linux 是默认首选。而当人们告诉你去用 Windows 服务器，却成了一个例外，而不是规矩。”

来自 Red Hat 的 Brian Gracely 确信 Kubernetes 品牌在 2020 年前不会大放异彩。

“我深知，我们现在应该鼓励人们去谈论 Kubernetes。毕竟，它是这两年才出现的，”Gracely 说。“我们通过观察其他的，已存在一定时期项目，同时这些项目有社区做为技术支持。我认为 OpenShift 团队的工作应该受到特别关注，我们是如何在很短的时期帮助客户获得成功，同打造自身业务变革的？我们打算花费下一个两年，让这个平台部署到任何地方，让人们意识到他们的价值只能在这样的平台实现。于此同时，我认为我们会看到客户如此说，‘看这些含苞初放的项目。我如何进一步壮大它？’”

“我认为在这三年里，我们将会讨论如何让人们把构建的任何应用都跑在 Kubernetes 上，”来自 CoreOS 的 Caleb Miles 这样说。“我相信我们会看到令人吃惊的 TensorFlow 运行环境，开源的机器学习工具，下一代数据存储，以及传统的关系数据库。我认为在这三年里我们讨论的话题是，在现代的 Kubernetes 平台上，让一个新手去构建和部署一个复杂应用是多么的容易。”

但是注意这些，可能会导致很大开销的平台应用程序和用例的产生。

“我确信一些细节的、底层的基础设施会被淡忘，” Miles 继续说到。 “在 2017 年，我们讨论很多话题是围绕应用程序如何迁移到单一节点——单一服务器节点，桌面节点或是一个节点数据中心……我相信我会进一步看到这样的趋势：构建更多更复杂的应用程序，把它们部署到平台。并且我认为，话题将会是我们正在构建的应用程序，而很少提及平台自身。人们会从平台中获益，为此会把价值带给他们自己——就像 Linux 是一个通用系统，很少被提及到在分布式系统层面的应用。”

Ihor Dvoretskyi，负责管理 Kubernetes 产品线，Mirantis 商业化平台制造商，直到出现这些上述公开的谈话，才认同了此点。“我会拿 Kubernetes 与 Linux 做比较，并且我会以 Linux 的方式，为 Kubernetes 运行的云原生应用程序命名，” Dvoretsyi 这样说到。

“我可以看到 Kubernetes 生态将会成长壮大，但与此同时，Kubernetes 内核稳定性会更加关注，” 他补充到。“我看到很多正在活跃地应用，并作为 Kubernetes 内核商业化制造项目的供应商。同样的形式也在最终用户中：很多公司正在通过使用 Kubernetes 作为新的解决方案，已将传统技术迁移到云原生的微服务体系。我很高兴看这些。然而在这三年里，我更期待看到的是大规模扩大的市场。”

“这确实像是让你如何去挂载一块磁盘，或是让你如何去认证，或是小心区分允许谁去访问网络。”

——Eric Brewer，Google

“我认为适用性、稳定性，和健康的项目、社区对于我们规划的未来是至关重要的吗，“Google 的 Aparna Sinha 这么说。“但假定我们现在正在做的事情是正确的，我认为 Kubernetes 会成为在跨基础设施架构上的容器编排平台。而且他会成为部署最多的，事实上的标准——前提是我们能把其他三件事情做好。“

” 现如今，已有很多不同的 Kubernete 发行版，“Sinha 继续说到，“我们认为会有一些关键性的内容，用户将在多个不同的环境中使用多个不同的云中的一套发行版，能够使他们的工作负载在这些环境中移植。第二条是，Kubernetes 之上是什么。我认为是将要被开发出来的，让开发人员使用的工具，是由 Google 云平台或是社区负责的。这样会使开发人员无需去考虑 Kubernetes 自身，并且可以从运行他们的应用程序达到目标种益处——举例而言，开发人员所依赖的持续集成系统，日志，监控和其他各方面。这也现今已存在的东西——随着技术成熟，会使一少部分将来会成为最棒的工具，而这些工具也会越来越多多地采用。”

“如果我们想让一个项目持续存在下去，我们不得不去这样做：我们必须满足灵活性、多样性和开放性，” Microsoft 的 Brendan Burns 说到，“人们基于一个核心项目，无论

是在其基础上或是基于其扩展插件，为了不让我们遇到对我们已完成内容（去如何解释）的窘境，为了最终的成功，要通过社区来扩充知识广度。“

Burns 预见 Kubernetes 生态系统演进成一个（完善的）生态系统，应该是一个由很多重要组件组成的系统。

“一个编程语言拥有它的核心语法，而且它的设计者还负责维护一套标准库，但事实上这件事是由广泛的人群去完成，” Burns 继续说到。 “并且围绕着开发，会有一个生态系统形式的库。而且我认为 Kubernetes 也会如此的。我们所期待的会有一个真正简洁的，良好定义的、基本核心的，可能有一些标准形式，但确实是一个生态系统形式的库，实用包，或是工具集，人们通过将他们整合到工作中实现解决方案。这就是（让项目）持续存在下去的途径。否则，你只是搞了一个只满足于一个非常特定环境的产品。”

“Kubernetes，” Google 的 Eric Brewer 确信， “就是将要成为平台的这个平台。”

“对于任何特定的领域……你可以利用 Kubernetes 作为底层基础构建任意作为服务的平台。而且它很强大。事实上，他解决的一些平台领域特定的问题：如果它太关注特定领域了，你就会从中受益，但也会遇到困难，并且很难找到答案，也要去做不是平台应该干的事。如果你在基于 Kubernetes 上面构建一些特定领域应用，同时你想获得领域内特定的需求，你可以去从 Kubernetes 里面找到答案。而且现在你会发现，对大量广泛的问题，(Kubernetes)总有一个灵活的解决方案。”

“于此同时，退一步讲，” Brewer 继续说， “在我看来，这就是我长期以来想要的、真正的云计算——它不是建立于将你的机器从这个数据中心迁移到另一个数据中心的云。这是一个有效的转变。但是这个转变又是如此有用。这就是说，让我们一起去编写程序的服务和 API，去构建这个云计算的基础。这是比迁移虚拟机到另一处数据中心而言更强大的变革。”

每一次数据技术平台的采用和实现都会有一个转变过程。Kubernetes 的不同之处就在于，需要考虑的更少，而改变得更大。这场转变对于微服务而言，会迫使它去转换观念，从外在角度去考虑业务的处理方式——这里既有自身正在转变的内容，也有我们将要创建的新事物。我们自己会在这场转变中得到更好体验，更少的浪费——即会减少过剩的

虚拟机，更少的延迟和大大提高的性能。就在此时，我们让一些可以自动化事情变得更加值得去自动化。

结束语

容器编排是计算历史上一个全新的概念。 它在大多数的细分层次上是对 Web 服务的最新描述。 在 20 世纪 90 年代后期，当 Web 服务的概念刚刚出现时，软件开发人员和网络工程师就构建了可以通过公共接口进行远程调用的通用功能目录。 在那段时间，技术上的压力让微软和 Novell 之间进行了一场战争，来争夺接口制定标准的权力。

当基于特定知识创建的管理应用程序第一次测试普通服务器的极限时，创建了一种新类型的负载均衡方案，该方案可以监听正在接收的请求的类型和状态，并根据服务器的可用性调度到特定服务器进行响应。 当服务器进行虚拟化后，调度请求变得更加容易，并且支持软件对调度网络进行定义。

但是现在，该方案模型中的服务器已经变成了一个单独的功能：微服务。 而现在，Kubernetes 和整合了它的生态系统，使得网络中的任何地方都可以精密地响应来自地球上任意地方的远程服务请求。 调度机制已经从最初的 Web 应用程序中凸显出来，成为了一个复杂的代理方案。 维护这些独立服务有了更加有效地方法，并被逐步被实现。现在可以对这些服务本身进行单独的维护，并且可以自动化的监督部署应用程序整个生命周期。

“Kubernetes 部署模式和步骤”将是我们关于 Kubernetes 生态系统系列的第二本书。它将重点关注支持这一新类型应用程序的机制及其构建的基础架构。 在此之前，请妥善照顾好您的系统，我们会在 The New Stack 和您再见。

KUBERNETES 解决方案目录

虽然这个目录有近一百个条目，但并不意味着列出了其中全部的参与者。 我们这里列出了许多用于部署和管理 Kubernetes 及其上运行的应用程序的项目和供应商产品。 列表分为四个部分，以方便读者快速阅读。这些只是您可能想要使用或考虑的解决方案进行进行评审时的一个起点。

KUBERNETES 产品发行

平台，产品，服务和项目，包括 Kubernetes 作为发行版本。

产品/项目(公司/支持组织机构)	发行类型
APPUiO (APPUiO) 基于 Red Hat OpenShift 的 PaaS 平台（Platform as a Service. 以开发者为目标的瑞士公司。	平台
Canonical Distribution of Kubernetes (Canonical) Canonical 的发行版为客户提供了稳定的上游 Kubernetes 版本，以及获得上游 Kubernetes 开发分支的早期版本。 Canonical 已经优化了 Kubernetes，使其可以运行在现有的基础设施和 DevOps 工具上，但它也适用于所有主要公共云和私有基础设施。	平台
CloudStack Container Service (ShapeBlue) 结合了 Apache CloudStack 和 Kubernetes 的强大功能的容器即服务（CaaS）解决方案。 它使用 Kubernetes 作为底层平台，为自动化应用程序容器的部署，缩放和运营提供支撑。	平台
Container Platform (Greenqloud) 内置 Kubernetes 容器和集群引擎的云管理系统。	平台
Deis Workflow (Microsoft) 一个专注于开发者自助服务和操作灵活性的 Kubernetes 原生 PaaS 平台。 Deis Workflow 可帮助团队在任何公共云，私有云或裸机群集上快速启动并运行 Kubernetes。	平台
Diamanti (Diamanti) 专门进行构建的容器基础架构，解决了将容器部署到生产中的难题，同时可以让用户保留现有的基础架构。 它通过插入一个 CPU 总线插件支持在裸机上进行本地切换。	平台
fabric8 (Red Hat) 支持应用程序读取和写入数据到 etcd。 一个简单的用例是将数据库连接详细信息或功能标志作为键值对存储在 etcd 中。 可以监视这些值，并允许您的应用在更改时自行重新配置。	平台
Fission (Platform9) Kubernetes 上的无服务器（serverless）功能框架。	平台
FusionStage (Huawei) 一个企业级平台即服务产品，其核心是基于包括 Kubernetes 和 Docker 在内的主流开源容器技术。 它可用于公有云和企业私有数据中心的部署。	平台
Getup Cloud (Getup Cloud) 使用 Docker，Kubernetes 和 OpenShift 构建的平台。 目前，巴西创业公司在使用试用版。	平台

产品/项目(公司/支持组织机构)	发行类型
Giant Swarm (Giant Swarm)	平台
基于托管容器的解决方案，以 Kubernetes 为核心组件构建，部署和管理容器化服务。 它为客户提供完整的私有 Kubernetes 集群管理，包括管理 Master 节点和 Node 节点。 它可以作为“服务”来使用，或者可以由 Giant Swarm 本地部署和管理。	
Google Container Engine (Google)	平台
Google Container Engine 是一个集群管理和编排系统，可让用户在 Google 云平台上运行容器。	
Hasura Platform (34 Cross Systems)	平台
一个创建和部署微服务的平台。 这家初创公司的基础设施是使用 Docker 和 Kubernetes 构建的。	
Hypernetes (HyperHQ)	平台
多租户 Kubernetes 发行商。 它结合了 Kubernetes 的编排功能和 Hyper 的运行时隔离功能，构建了一个安全的多租户容器管理平台。	
IBM Bluemix Container Service (IBM)	平台
使用 IBM Containers 技术在 IBM Bluemix 上的托管云环境中运行 Docker 容器。 IBM Containers 提供 Docker 容器的全面托管和生命周期管理，以及自动的和集成的日志分析和监控，具有自动恢复功能的弹性扩展，可靠性工具，负载平衡和路由，持久化存储，安全服务和私有镜像仓库。	
K2 (Kraken 2) (Samsung CNCT)	平台
支持在 CoreOS 上使用 Terraform 和 Ansible 部署 Kubernetes 集群。	
Kel (Eldarion)	平台
一个开源的基于 Python 和 Go 的开源项目，集成基于 Kubernetes 的 PaaS 平台，可以轻松部署和托管整个软件生命周期中的 Web 应用程序。	
Kismatic Enterprise Toolkit (KET) (Apprenda)	平台
KET 是 Apprenda 商业支持的和完全开源的 Kubernetes 产品。 它提供了一组默认的集群服务，提供了在几个节点或笔记本电脑上自动化安装和运行 Kubernetes 的增强功能。	
Kontena (Kontena)	平台
一个专注于开发者自助服务和操作灵活性的 Kubernetes 原生 PaaS 平台。 Deis Workflow 可帮助团队在任何公共云，私有云或裸机群集上快速启动并运行 Kubernetes。	
Kops (Cloud Native Computing Foundation)	平台
Kubernetes Operations（kops）实现了生产级 Kubernetes 安装，升级和管理。	
KUBE2GO (Platform9)	平台
将 Kubernetes 群集部署到公共云的工具。 截至发布时，仅支持 AWS。	
Kubernetes (Cloud Native Computing Foundation)	平台
Kubernetes 是一个开源的 Docker 编排工具。 谷歌最初开发 Kubernetes 来帮助管理自己的 LXC 容器。 有对状态服务的支持是通过一个叫 Pet Set 的新对象完成的。 另外，还有很多可用的网络和数据存储卷插件。	
Kubernetes Services Managed by LiveWyer (LiveWyer)	平台
该咨询公司提供 Kubernetes 管理实施和 Kubernetes 培训。	

产品/项目(公司/支持组织机构)	发行类型
Kubo (Pivotal) 提供与 Cloud Foundry 一起使用 BOSH 部署和管理 Kubernetes 的解决方案。	服务商
Last.Backend (Last.Backend) 在 Kubernetes 之上运行构建的平台，带有一个命令行工具包和 UI，用于部署应用程序和管理基础架构。	平台
Managed Kubernetes (Kumina) 荷兰咨询公司，提供管理服务。	服务商
Minikube (Cloud Native Computing Foundation) Minikube 是一个让本地运行 Kubernetes 变得简单的工具。 Minikube 在笔记本电脑上的虚拟机（VM）内运行单节点 Kubernetes 群集。 主要面向希望尝试 Kubernetes 或者进行日常开发的用户。	
OpenShift Container Platform (Red Hat) 一个容器应用程序平台，可以跨越多种不同的基础架构。 它使用 Docker 和 Kubernetes 技术构建。	平台
OpenShift Origin (Red Hat) OpenShift Origin 是 OpenShift 的上游开源版本，旨在开发原生的云应用程序。 OpenShift 是一个构建在 Docker 容器上的 PaaS 平台，它使用 Kubernetes 进行编排。 它还具备 Atomic 和 Red Hat Linux 组件。	平台
Photon Platform (VMware) 基于容器优化的云平台，提供对 Kubernetes 集群的按需访问。	服务商
Pivotal Container Service (Pivotal) Kubo 的商业版本，可以轻松在 vSphere 或 Google 云平台的环境中部署和使用 Kubernetes。	服务商
Platform9 Managed Kubernetes (Platform9) 使用 Kubernetes 提供服务托管。 客户可以在 Platform9 的单一平台上与虚拟机（VMs）一起编排和管理容器。 换句话说，您可以使用 OpenStack 和（或者）Kubernetes 编排虚拟机（VMs）。	服务商
Rancher (Rancher Labs) Rancher 原生支持和管理 Kubernetes，Mesos 和 Swarm 集群。	服务商
Red Hat OpenShift (Red Hat) Eclipse Che 项目（收购 Codenvy）基础上，集成基于 Web 的开发人员环境，源代码库和 CI/CD 流程。 是个整合 OpenShift Online 的开发环境。	平台
Red Hat OpenShift Container Platform (Red Hat) 一个容器应用程序平台，可跨越多种基础架构（裸机，虚拟机，VMware，OpenStack，AWS，Azure 和 GCP）。 它使用 Docker 和 Kubernetes 技术构建。集成了多租户网络（SDN），多种类型的存储，容器镜像仓库，Red Hat 中间件和应用程序服务以及 Open Service Broker。 它在 RHEL 主机上运行，使用 Ansible 进行部署，并使用 CloudForms 进行管理。	平台
OpenShift Dedicated (Red Hat) 私有，高可用性 OpenShift 集群，托管在 Amazon Web 服务（AWS）或 Google 云平台上，由 Red Hat 提供云运营服务。	平台

产品/项目(公司/支持组织机构)	发行类型
Red Hat OpenShift Online (Red Hat) 世界各地的开发人员可以使用 Red Hat 的 OpenShift 公有云服务（有免费和付费的区别）。	平台
StackPointCloud (StackPointCloud) 允许用户使用自己选择的云服务提供商轻松创建，扩展和管理任何规模的 Kubernetes 集群。 其目标是成为基于 Kubernetes 的云服务通用控制面板。	服务商
Supergiant (Qbox) Supergiant 是一个运行 Docker 容器的开源框架。 它托管基于 Kubernetes 的有状态的集群应用程序。 它使用自研的结构和代码进行持久化存储和外部负载均衡。 Supergiant 由 Qbox 创建并提供商业支持。	服务商
SUSE Container as a Service Platform (SUSE) 基于容器的应用程序以及服务的开发和托管平台。 它使用 SUSE Linux Enterprise MicroOS 和 Kubernetes 技术。	服务商
Symphony (Stratoscale) 托管作为服务的 Kubernetes 管理功能。	服务商
Tectonic (CoreOS) Tectonic 提供纯粹的 Kubernetes 上游的企业级解决方案。 Tectonic 提供自动化操作，用户只需点击一次鼠标即可轻松升级到最新的 Kubernetes 版本，实现了私有云和公共云提供商之间移植，并始终保持 LDAP，RBAC 和 SAML 支持的安全性。 用户通过简单和安全的方法，可以轻松扩展应用程序，保证部署的一致性，并支持跨环境的轻松管理应用程序。 除了部署最新版本的 Kubernetes 之外，Tectonic 还包括帮助您快速启动和运行的安装程序，可视化的集群监控控制台，管理集群组件，以及提供与现有安全框架集成的安全功能。	服务商
Telekube (Gravitational) 一个用于打包，部署和远程管理遍布全球的云和内部的复杂多节点 Linux 应用的工具集。 自称是一个私有的 SaaS 平台。	服务商
TenxCloud Container Engine (TCE) (TenxCloud) 一家提供 Kubernetes 服务的中国公司（时速云）。	服务商

工具和服务

有助于实施 Kubernetes 的产品，以及在 Kubernetes 之上部署和管理的应用程序。

产品/项目(公司/支持组织机构)	专业服务类型（如适用）
AppController (Mirantis) 可以部署到 Kubernetes 集群中以创建对象并管理依赖关系的 Pod。	
Ark (Heptio) 用于管理灾难恢复的实用程序，特别适用于 Kubernetes 集群资源和持久化存储卷。	
Azure Container Service (Microsoft) Azure 容器服务简化了集群的创建和配置。 此集群的默认配置包括使用 Docker 和 Docker Swarm 以实现代码可移植性; 和 Marathon，Chronos 和 Apache Mesos 技术来确保可扩展性。	
Bootkube (N/A) 启动自托管 Kubernetes 集群的辅助工具。	
Cabin (Bitnami) 用于管理 Kubernetes 应用程序的 iOS 和 Android 应用。	
cAdvisor (N/A) cAdvisor（Container Advisor）是一个 Google 支持的项目，分析运行容器的资源使用情况和性能特征。	
Containerd (Cloud Native Computing Foundation) Containerd 是管理一台机器上的容器的守护进程。 它基于 Docker Engine 的实时容器内核，并遵循 Open Container Initiative 规范。	
ContainerPilot (Joyent) ContainerPilot 与其他调度程序一起工作 - 让他们启动和停止容器 - 而 ContainerPilot 协调其余部分。 由 ContainerPilot 编排的应用程序可以从一个调度程序移植到另一个调度程序。	
Datadog-Kubernetes Integration (Datadog) 实时收集和监控 Kubelets 的指标。 它作为 Docker 容器与现有工作负载一起部署。	
Digital Rebar (RackN) 基于容器的云和提供硬件设施的平台。	
ElasticKube (CenturyLink) ElasticKube 的服务用于连接 CI/CD 流程，配置管理工具和部署云应用程序。 它是 Kubernetes 的一个开源管理平台，可以为容器应用程序提供自助服务。	

产品/项目(公司/支持组织机构)	专业服务类型（如适用）
Endocode (Endocode) 一家德国软件工程公司，曾帮助几个与容器相关的项目，为它们提供了许多捐款。	咨询
Heapster (Heapster) 提供计算资源使用情况分析和容器集群监控。 Heapster 目前支持 Kubernetes 和 CoreOS。	
Helm (Cloud Native Computing Foundation) Kubernetes 原生包管理器，可帮助运营商声明和管理复杂的应用程序。	
Heptio Professional Services and Support (Heptio) Heptio 是由 Kubernetes 项目的创始人创立的公司，旨在支持和推动开放的 Kubernetes 生态系统。	支持
Jetstack Container Services (Jetstack) Jetstack 是一家专注于帮助企业建立容器管理基础设施的咨询公司。	支持
K8S Dashboard (Distelli) Distelli 提供了一个 Dashboard 来显示和管理应用程序。	
K8sPort (Cloud Native Computing Foundation) 具有游戏化功能的社交网络，它支持 Kubernetes 社区。	
Kolla-Kubernetes (OpenStack Foundation) 该项目提供了 Docker 容器和 Ansible 操作手册，用以在 OpenStack 上部署 Kubernetes。	
Kompose (Cloud Native Computing Foundation) 一个帮助用户从熟悉的 docker-compose 转移到 Kubernetes 的工具。	
ksonnet (Heptio) Jsonnet 是来自 Google 的开源 JSON 模板语言。使用 ksonnet-lib 和 kubecfg 为编写 Kubernetes 的复杂 YAML 配置文件提供了一个更简单的选择。	
kubeadm (Cloud Native Computing Foundation) 是 Kubernetes 发行版的一部分，可帮助安装和设置 Kubernetes 群集。	
Kubediff (Weaveworks) Kubernetes 显示运行状态和版本控制配置之间差异的工具。	
Kubeflix (Red Hat) 提供与 Netflix 开源组件（如 Hystrix，Turbine 和 Ribbon）的 Kubernetes 集成。	
Kubeless (Bitnami) Kubernetes 原生 serverless 框架。 它支持 HTTP 和基于事件的触发器，具有一个 serverless 插件，图形用户界面和并行实时运行。	
Kubermatic (Loodse) 使得部署和管理多个容器群集变得更容易。	
Kubernauts (Kubernauts) 作为非营利组织，Kubernauts 提供培训和咨询服务。 它管理 Kubernauts Worldwide Meetup。	
Kubernetes Anywhere (Kubernetes Anywhere) 一个自动化的解决方案，最终将允许用户跨多个云部署 Kubernetes 集群。	

产品/项目(公司/支持组织机构)	专业服务类型（如适用）
Kubernetes Dashboard (Kubernetes Dashboard) 基于 Web 的通用用户界面对 Kubernetes 集群进行管理。 它允许用户管理集群中运行的应用程序，进行故障排除，并提供集群本身的管理功能。	
Kubernetes service-catalog (N/A) 基于 Web 的通用用户界面对 Kubernetes 集群进行管理。 它允许用户管理集群中运行的应用程序，进行故障排除，并提供集群本身的管理功能。	
Kubernetes Support (Apprenda) 专业支持 Kubernetes 来处理原生的部署和正在进行的操作。 Apprenda 提供三层支持，包括每个事件的开销。	
Kubernetic (Harbur Cloud Solutions S.L.) 用于管理 Kubernetes 集群的桌面客户端。	
Kublr (EastBanc Technologies) 一个自动化的集群管理平台。	
Kupespray (N/A) 部署 Kubernetes 集群的工具。 可以代替 kops 和 kubeadm。	
Magnum (OpenStack Foundation) OpenStack API 服务，使得容器编排引擎（如 Docker 和 Kubernetes）成为 OpenStack 中的重要资源。	
Open Service Broker API (Cloud Foundry Foundation) 该项目为开发人员，独立软件供应商和 SaaS 供应商提供了一种向运行在云原生平台（如 Cloud Foundry，OpenShift 和 Kubernetes）上的应用程序提供服务的方法。 它适用于 Kubernetes 孵化器服务目录中的项目。	
Poseidon (University of Cambridge) Poseidon 整合了 Firmament 与 Kubernetes。	
Prometheus (Cloud Native Computing Foundation) Prometheus 是一个开源的系统监控和警报工具集，为监控系统和时间序列数据库提供服务。	
Quick Start for Kubernetes (Heptio) 提供一组模板和配置，支持 CloudFormation 和 kubeadm 在 AWS 上快速设置 Kubernetes 集群。	
ReactiveOps (ReactiveOps) 为定制化构建基于 Kubernetes 的 DevOps 平台提供咨询。	咨询
rkt (Cloud Native Computing Foundation) rkt 是用于在 Linux 上基于 App Container Specification（appc spec）运行应用程序容器的命令行工具（CLI）。	

产品/项目(公司/支持组织机构)	专业服务类型（如适用）
Sematext Kubernetes Agent (Sematext) Sematext 通过收集 Kubernetes 日志，事件和指标，提供开箱即用的指标图表，可搜索的日志以及关联日志，关键指标，警报等功能，从而提供运营分析能力。 它利用 Sematext Docker Agent 从 Docker 容器名称中提取信息，并使用命名空间，pod，容器，镜像名称和 UID 标记所有日志。	
Sonobuoy (Heptio) 一种诊断工具，通过以可访问和非破坏性的方式运行一组 Kubernetes 一致性测试，可以更容易地了解 Kubernetes 集群的状态。	
Supergiant Support (Supergiant) Supergiant 是一个运行 Docker 容器的开源框架。 它使用 Kubernetes 托管有状态的集群应用程序。 它使用自己的结构和代码进行持久化存储和外部负载均衡。 由创建者 Qbox 为 Supergiant 提供商业支持。	支持
Tack (N/A) 这是一个使用 CloudFormation 的替代方法，采用自动化的 Terraform 模块，用于在 AWS 虚拟私有云上创建基于 Container Linux 运行的高可用的 Kubernetes 集群。	
Virtual Private Pipelines (Oracle) Docker 原生，单租户和完全托管的 CI/CD 平台，针对 Kubernetes 进行了优化，并使用微服务架构。 它提供了网络隔离和灵活的并发性。	
Weave Scope (Weaveworks) Weave Scope 提供实时监控容器的解决方案。	

相关的 DEVOPS 技术

在整个 DevOps 生命周期中与 Kubernetes 协作的工具和技术。 这个部分主要涵盖创建，打包/发布，配置或监视等步骤。

产品/项目(公司/支持组织机构)	DevOps 生命周期阶段
AppDynamics (Cisco) 通过主机上安装的代理软件来收集数据应用程序和业务性能的软件。 它提供了一个扩展模块来收集来自 Docker API 的数据。	监控
AppFormix (Juniper Networks) 在任何公共，私人，多租户或混合环境中运行的云基础架构监控和分析软件。 它包括 ContainerFlow，它利用英特尔资源导向器技术强化工作负载之间的隔离，并为容器工作负载提供更好的性能支持。 该公司专注于 OpenStack 和 Kubernetes 运营商的分析。	监控
AppsCode (AppsCode) 用于容器应用的协同编码，测试和部署的集成平台。 支持将容器部署到 AWS 和 Google Cloud 平台。	创建
Clocker (Cloudsoft) Clocker 创建和管理 Docker 云基础架构。 它使用 Apache Brooklyn 的蓝图，以便部署和管理 Docker Swarms 和 Kubernetes 集群。	配置
CloudPlex (CloudPlex) CloudPlex 是一个云编排和管理平台。 它使用 Chef 部署到虚拟机，并使用 Kubernetes 部署 Docker 容器。	打包/发布
Cobe.io (Cobe.io) 为异构的基础架构提供实时拓扑显示，并在提供覆盖整个系统模型的性能指标和警报。	监控
Codeship Pro (Codeship) 它可以很容易地测试和部署你的微服务，并推送到任何镜像仓库。 如果您想要使用 Kubernetes 进行部署，它也是非常完美的，因为它提供了一个方便的原生命令行工具，可让您在本地运行构建的微服务，对环境变量进行加密，并保证开发和生产环境之间 100％的兼容。 Codeship Pro 提供免费的计划，每个月可以提交 100 个版本，没有项目，团队和用户的限制。	打包/发布
Container Builder (Google) 支持在 Google 云端平台上快速，一致，可靠的构建。	构建
Container Linux (CoreOS) CoreOS Container 是一个最小的 Linux 操作系统，支持常用的容器系统运行。 操作系统的设计是以集群方式运行。 例如，它被设计成可以方便的通过 PXE 启动和支持大多数云商。	构建

产品/项目(公司/支持组织机构)	DevOps 生命周期阶段
Draft (Microsoft) 一个支持开发人员在 Kubernetes 上创建云原生应用程序的工具。 Draft 现在是个实验性的项目。	打包/发布
Dynatrace (Dynatrace) Dynatrace 提供基于其 Ruxit 技术的的监控工具套件。 它通过注入一个容器的代理程序，可以自动发现在主机上运行的新服务，并从 Docker API 获取数据。 Dynatrace 也在开发人工智能技术来对遇到的问题进行深入分析。	监控
etcd (CoreOS) etcd 是一个分布式的键值存储系统，提供了一个高可用的方式来存储跨机器集群的数据。 它是 GitHub 上的开源项目，并且为 Kubernetes 提供关键数据存储。 etcd 在网络被隔离的情况下可进行简洁高效的主控节点选举，并容忍包括主控节点在内的机器故障。 您的应用程序可以读取和写入数据到 etcd。 一个简单的例子是将数据库连接详细信息或功能标志作为键值对存储在 etcd 中。 可以监听这些值的变化，允许您的应用在更改时自动重新配置。	配置
Fluentd (Cloud Native Computing Foundation) Fluentd 是开源的统一日志记录层的数据收集器。	监控
Forge (Datawire) 基于 Docker 和 Kubernetes 的服务构建。 使用 YAML 文件配置特定部署。	配置
gRPC (Cloud Native Computing Foundation) 开源的高性能通用远程过程调用（RPC）框架，最先使用在移动端和 HTTP/2 协议上。	支持
Istio (N/A) 一个集成微服务平台，支持跨微服务的流量管理，策略执行和遥测数据汇总。Istio 的控制架构是在底层集群管理平台上提供了一个抽象层。	监控
Kong (Mashape) Kong 是对 API 进行管理的管理层。 它提供编排 Dockerfiles 的功能。	配置
Linkerd (Cloud Native Computing Foundation) 一个微服务的进程之外网络技术栈。 它可以充当透明的 RPC 代理，处理服务间进行 RPC 安全通信一切需求，包括负载平衡，服务发现，检测和路由。 Linkerd 建立在 Finagle 框架之上。	监控
Loom (Datawire) 运行在 AWS 上 Kubernetes 环境中的自助配置微服务。 它具有预先配置好的模板，用于在 AWS 中创建 Kubernetes 开发集群。	配置
Navigator (Jetstack) 管理在 Kubernetes 上的 DBaaS，Navigator 是一个集中控制器，用于管理 Kubernetes 上的有状态服务。	配置
New Relic APM (New Relic) 应用程序性能监控是 New Relic 产品套件的核心功能，它最初被称为数字智能平台。 其基于代理技术的监控方法特别适用于定位与代码相关的应用程序性能问题。	监控

产品/项目(公司/支持组织机构)	DevOps 生命周期阶段
OpenTracing API (Cloud Native Computing Foundation) 一致的、显式的、并非基于特定供应商的 API，用于分布式跟踪和运行环境迁移。	监控
Project Atomic (Red Hat) Atomic Host 项目使用基于 RHEL，Fedora 和 CentOS 的组件在 Docker 容器中运行应用程序。 除了 Atomic Host 之外，该项目还包括 Nulecule，这是一个基于容器的应用程序规范，可以使用现有容器作为新应用程序的构建基础。	构建
Puppet Module for Kubernetes (Puppet) 使用 Puppet 部署 Kubernetes 的配置文件模板。	配置
Red Hat OpenShift Application Runtimes (RHOAR) (Red Hat) 目前处于测试阶段，RHOAR 是一套基于 Spring Boot，Eclipse Vert.x，Node.js 和 WildFly Swarm 的云原生的，容器优化的实时应用框架。 原生的集成 OpenShift 容器平台和 Kubernetes。	构建
StackState (StackState) 提供全面的容器监控解决方案。	监控
Sysdig Cloud (Sysdig) Sysdig Cloud 基于开源的 Sysdig 技术，在容器环境中进行监控，故障排除和预警。Sysdig Cloud 可以作为云服务使用，也可以作为托管软件部署在您的私有云中。	监控
Tack (N/A) 一个 Terraform 模块，用于在 AWS 虚拟私有云中创建基于 CoreOS 的在 Container Linux 上运行的 Kubernetes 集群。	打包/发布
Telepresence (Datawire) 针对远程 Kubernetes 或 OpenShift 集群，可以进行本地化开发。	构建
Terraform (HashiCorp) Terraform 是构建和启动基础架构的工具，包括支持容器。	配置
Wavefront (VMware) 使用 cAdvisor 收集容器运行指标，并与其他系统和应用的指标一起进行分析。	监控
Weave Cloud (Weaveworks) 服务平台化简化了容器和微服务的部署，监控和管理。 它与 Kubernetes 集成，并提供 Prometheus 监控服务。	监控

相关的基础设施技术

以下内容包括存储，网络，计算和其他基础设施架构技术的常见示例，支持使用像 Kubernetes 一样的云原生环境。

产品/项目(公司/支持组织机构)
CoreOS Container Linux (CoreOS) CoreOS Container Linux 是最小化的操作系统之一，支持常用的容器系统。 操作系统设计目标是以集群方式运行。 例如，它很容易通过 PXE 启动和支持大多数云服务商。
Network Interface (CNI) (Cloud Native Computing Foundation) CNI 是一个帮助 Linux 应用程序容器配置网络接口的项目。它可以帮助设置容器的网络连接，并在删除容器时删除已分配的资源。
CNI-Genie (Huawei) 使容器调度器能够无缝地连接到像 Calico，Canal，Romana 和 Weave 这样的 CNI 插件。
Container Registry (Google) 在 Google 云端平台上快速，私有的 Docker 镜像存储服务。
Contiv (Cisco) 将容器，虚拟机和裸机统一到一个网络结构中，允许容器网络在虚拟机或裸机网络中寻址。
dex (CoreOS) dex 是一个身份服务，使用 OpenID Connect 来驱动其他应用程序的身份验证。 Dex 可以在任何 Kubernetes 集群之上运行。 dex 不是一个用户管理系统，而是通过“连接器”作为其他身份提供者的门户。这使得 dex 可以将身份验证推送到 LDAP 服务器，SAML 提供商或者 GitHub，Google 和 Active Directory 等已建立的身份提供商。 客户端编写认证逻辑，一旦与 dex 交互，dex 会控制处理特定的后端协议。
flannel (CoreOS) flannel 是一个虚拟网络，它为每个主机提供一个用于容器运行时的子网。 像 Google 的 Kubernetes 这样的平台假定每个容器（pod）在集群内都有一个唯一的，可路由的 IP。 这种模式的优点是降低了端口映射的复杂度。
Open vSwitch (Linux Foundation) 采用开源 Apache 2.0 许可证的产品级的多层虚拟交换机。 它旨在通过编程扩展实现大规模的网络自动化，同时还支持标准的管理接口和协议。 此外，它还支持通过类似 VMware vNetwork 分布式 vswitch 或思科 Nexus 1000V 的设备来分布式访问多个物理服务器。

产品/项目(公司/支持组织机构)
Longhorn (Rancher Labs) 使用容器和微服务构建的分布式块存储系统。
Minio (Minio) Minio 是为云应用程序和 DevOps 构建的开源的对象存储服务器。
Nuage Networks Virtualized Cloud Services (VCS) (Nokia) 基于 Nuage Networks 虚拟化服务平台（VSP）的数据中心和云网络框架。
Nuage Networks Virtualized Services Platform (VSP) (Nokia) 为各种规模的云服务提供软件定义的网络功能。 它用来实现所有已有虚拟化和非虚拟化服务器和网络资源的无中断覆盖。 VSP 被设计可以与 Docker 容器，Kubernetes 和 Mesos 一起工作。
OpenContrail (Juniper Networks) 一个 Apache 2.0 许可的项目，使用基于标准的协议构建，为网络虚拟化提供了所有必要的组件：SDN 控制器，虚拟路由器，分析引擎和已发布的北向 API。 它具有大规模的 REST API 来配置和收集系统中的操作和分析数据。
Portworx PX-Series (Portworx) 用于持久化存储的数据层，可以使用 Kubernetes 进行管理。
Project Calico (Tigera) 为连接数据中心工作负载（容器，虚拟机或裸机）提供可扩展的网络解决方案。 它使用 3 层交换。 Calico 可以在没有封装或 overlay 的情况下进行部署，以提供大规模的高性能网络连接。
Quay (CoreOS) 可以运行在私有的服务器上的安全镜像仓库。
Redis (Redis) Redis 是内存数据库，持久化数据保留在磁盘上。数据模型是键值对，支持许多不同类型的值。
Romana (N/A) 云原生应用程序的网络和安全自动化解决方案。 Romana 可自动创建独立的云原生网络，并通过分布式防火墙保护应用程序的安全，支持将访问控制策略部署在任意位置运行的端点和服务上。
Rook (Quantum) 通过持久卷存储 Kubernetes 应用程序。
Trireme (Aporeto) 由 Aporeto 策划的一个开源库，为云原生应用程序提供隔离。
Twistlock (Twistlock) Twistlock 为现代企业提供云原生的网络安全。通过先进的智能和机器学习能力为整个开发生命周期中提供自动策略创建和执行。 与领先的 CI/CD 和编排工具的原生集成提供安全，实现持续创新并不减缓开发速度。 强大的合规性检查和可扩展性支持开发人员从工作站到生产环境进行全面的控制。
Vitess (Google) 支持扩展的 MySQL 数据库解决方案。 它可以在 Kubernetes 上运行。

产品/项目(公司/支持组织机构)
Weave Net (Weaveworks) 可以将容器连接成透明，动态和弹性的网格。 Weave Net 创建了一个虚拟网络，连接多个主机上的 Docker 容器，并支持它们的自动发现。

更多信息

本书中提及的以下公司是 The New Stack 的赞助商：Apcera，Aporeto，CA Technologies，Chef，Cloud Foundry Foundation，{code}，Containership，DigitalOcean，GoDaddy，HPE，InfluxData，Microsoft，OpenStack，Packet，PageRDuty，StackRox ，Linux 基金会，ThoughtWorks，Univa，VMware，Wercker。

华为公司是 The New Stack 的咨询分支机构。

特别感谢 Joseph Jacks 维护 Kubernetes 发行商的电子表格。

特别感谢 Inspursoft 技术中心提供翻译。

- 特别感谢 Inspursoft 技术中心王建华总经理的大力支持。
- 特别感谢 Inspursoft 技术中心的华勇，罗天（Tim），王锟（George），李耀明，李岩青（MrLi），毛泽（Robin），魏亭（Sok）的翻译工作。