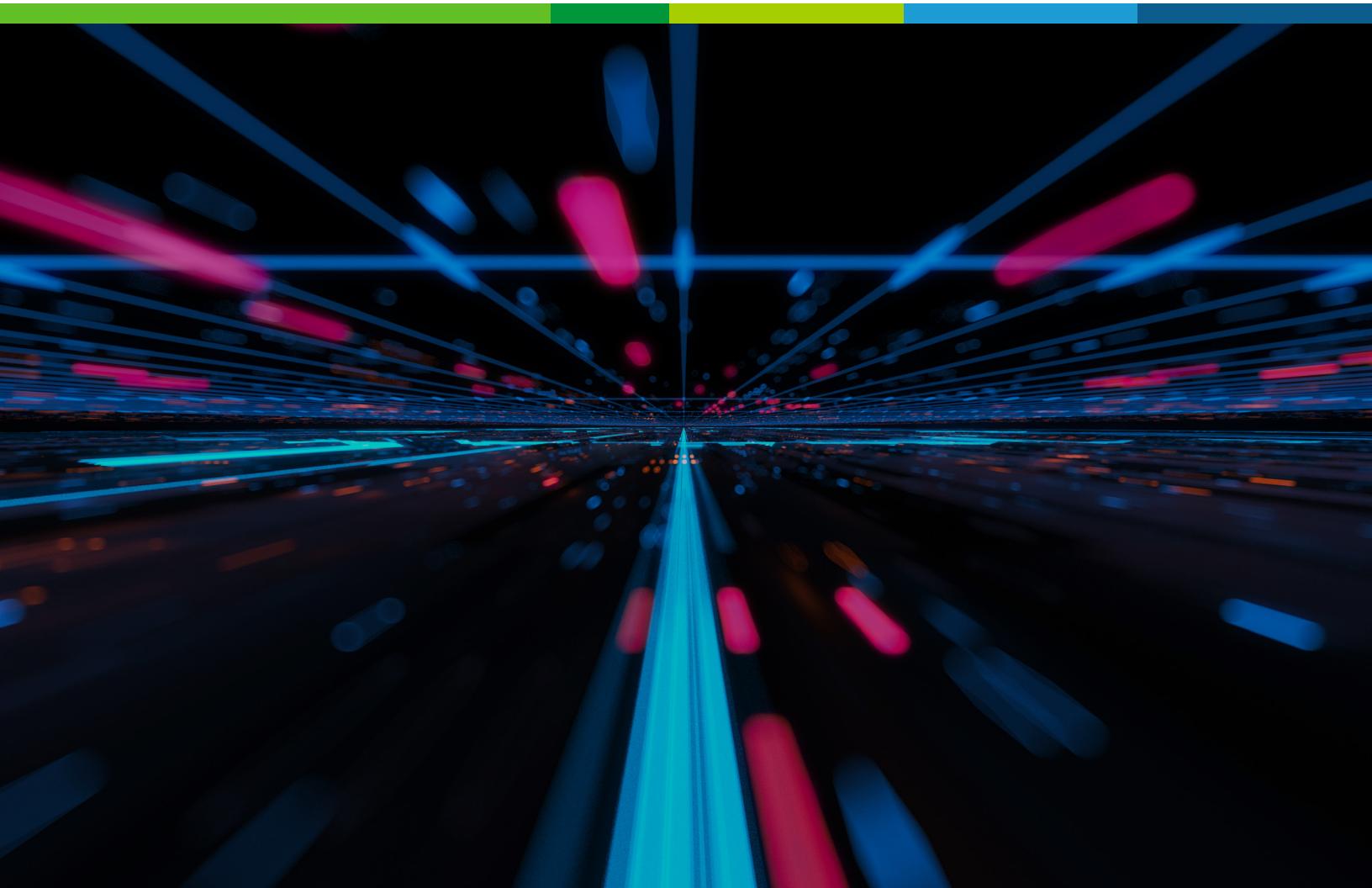


Kubernetes：从测试到生产

通过多种流量管理工具提升弹性、可视性和安全性

作者: Jenn Gile
产品营销经理
F5 NGINX



目录

前言	4
第一部分：	
Kubernetes 流量管理工具助力应对 Kubernetes 挑战.....	5
1. 生产级 Kubernetes 助您降低复杂性.....	5
现代的转变和趋势	5
解决方案：生产级 Kubernetes.....	8
Kubernetes 优先事项：弹性、可视性和安全性.....	11
2. 提高弹性的六种方法.....	12
高级流量管理用例	13
防止服务接收过多请求	15
避免级联故障.....	15
在生产环境中测试新版本.....	16
确保新版本稳定	17
了解客户更喜欢新版本还是当前版本.....	18
在无需停机的条件下将用户迁移到新版本.....	19
使用 NGINX 进行高级 Kubernetes 流量管理	20
借助高级定制实现更复杂的流量控制和流量精分	22
3. 更出色的可视化能够帮助您解决两个问题.....	23
如何通过可视化获取洞察	23
发现并解决两个 Kubernetes 问题	24
Kubernetes 问题 1：应用运行缓慢.....	25
Kubernetes 问题 2：资源耗尽.....	26
4. 提高安全性的六种方法.....	27
安全防护及身份验证的相关术语.....	27
让安全成为每个人的责任	29
快速解决 CVE 漏洞，有效避免网络攻击.....	29
抵御 OWASP 十大安全漏洞和 DoS 攻击.....	31
从应用中卸载身份验证和授权.....	34
设置有“防护栏”的自助服务	35
实施端到端加密	36
确保客户端使用可信的强密码.....	37

第二部分：	
选择最适合您需求的 Kubernetes 流量管理工具	40
5. Ingress controller 选型指南 —— 确定需求	40
什么是 Ingress controller?	41
为什么需要 Ingress controller?	41
Ingress controller 有何作用?	41
您希望 Ingress controller 解决什么问题?	43
如何为 Ingress controller 分配资源?	44
6. Kubernetes Ingress controller 选型指南 —— 评估风险和技术前瞻性	47
Ingress controller 风险.....	47
面向未来的 Ingress controller	50
7. Ingress controller 选型指南 —— 开源、默认和商用版本能力对比	53
开源 Ingress controller	53
默认的 Ingress controller.....	55
商用 Ingress controller	56
8. Ingress controller 选型指南 —— NGINX Ingress controller 选项	58
NGINX 版与 Kubernetes 社区版 Ingress controller	58
NGINX 开源版与 NGINX Plus —— 为什么要升级到我们的商用版?	60
9. 如何选择 service mesh	64
您做好使用 service mesh 的准备了吗?	66
如何选择适合您的应用的 service mesh	68
F5 NGINX Service Mesh.....	70
没有准备好使用 service mesh?	73
附录：	
三种 NGINX Ingress Controller 选择的性能测试	74
静态部署的延迟结果.....	75
动态部署的延迟结果.....	76
动态部署中的超时和错误结果	77
结语	78
术语	79

前言

自 2014 年推出以来，开源项目 Kubernetes 席卷了云基础架构市场，迅速成为有史以来最热门、部署最广泛的基础架构管理解决方案之一。Kubernetes 催生了一整套全新的工具和平台——包括 Red Hat OpenShift 和 SUSE Rancher 等热门托管平台以及 Amazon Elastic Kubernetes Service、Azure Kubernetes Service、Google Kubernetes Service 等云供应商平台——这些工具和平台旨在提高应用部署的可用性、易用性和适用性。然而即便这些企业级的平台铺平了道路，用户想要真正从 Kubernetes 中获取价值仍需经历陡峭的学习曲线。

Kubernetes 带来了许多新概念，特别是关于网络和流量管理的概念。伴随着这些新概念的出现，一系列专为轻量、容器化且分布式应用部署而设计的全新工具也诞生了。特别是 Ingress controller（Ingress 控制器）和 service mesh（服务网格），在 Kubernetes 时代之前，这些工具并不存在，四层和七层的协议和流量通常也不是由同一个控制平面管理的。从更细的层面来说，Kubernetes 还新引入了安全和管理方面的复杂性。在完全使用临时的基础架构并且经常不断变动的环境中，即使是像负载均衡这样的简单任务都与传统环境有很大不同——无论是用新的 IP 地址设置新实例，还是在全球各地到处迁移。

通过更大的弹性、更高的性能和更出色的安全性，这些流量管理的新概念和新工具可以大幅改善开发人员体验，并加快应用开发周期和交付周期。但当流量管理的重要性被忽视或低估时，企业就会遇到一些问题，导致其难以获得 Kubernetes 应该带来的价值，甚至还会因此面临风险。为了充分发挥 Kubernetes 的价值，不同的团队都必须以 Kubernetes 的原生方式去理解和管理流量。

本电子书将为读者介绍 Kubernetes 从测试到投入生产的整个过程。从概念验证到灰度部署，从蓝绿部署到全面投入生产——每一章都提供了清晰明确的解释，从而帮助读者了解 Kubernetes 策略，并为读者主动有效管理流量提供参考。通过了解实用的决策框架，您还可以确定哪些流量管理解决方案最适合（或不适合）特定的应用和情形。本电子书还包含参考架构和即时可用的代码示例，可以帮助任何 Kubernetes 技能水平的读者进一步了解 Kubernetes 流量管理的工具和概念。

本电子书包含了丰富清晰的基础知识，非常值得阅读。希望它能够成为您学习 Kubernetes 过程中值得信赖的伙伴。

Jenn Gile
产品营销经理
F5 NGINX

第一部分

Kubernetes 流量管理工具 助力应对 Kubernetes 挑战

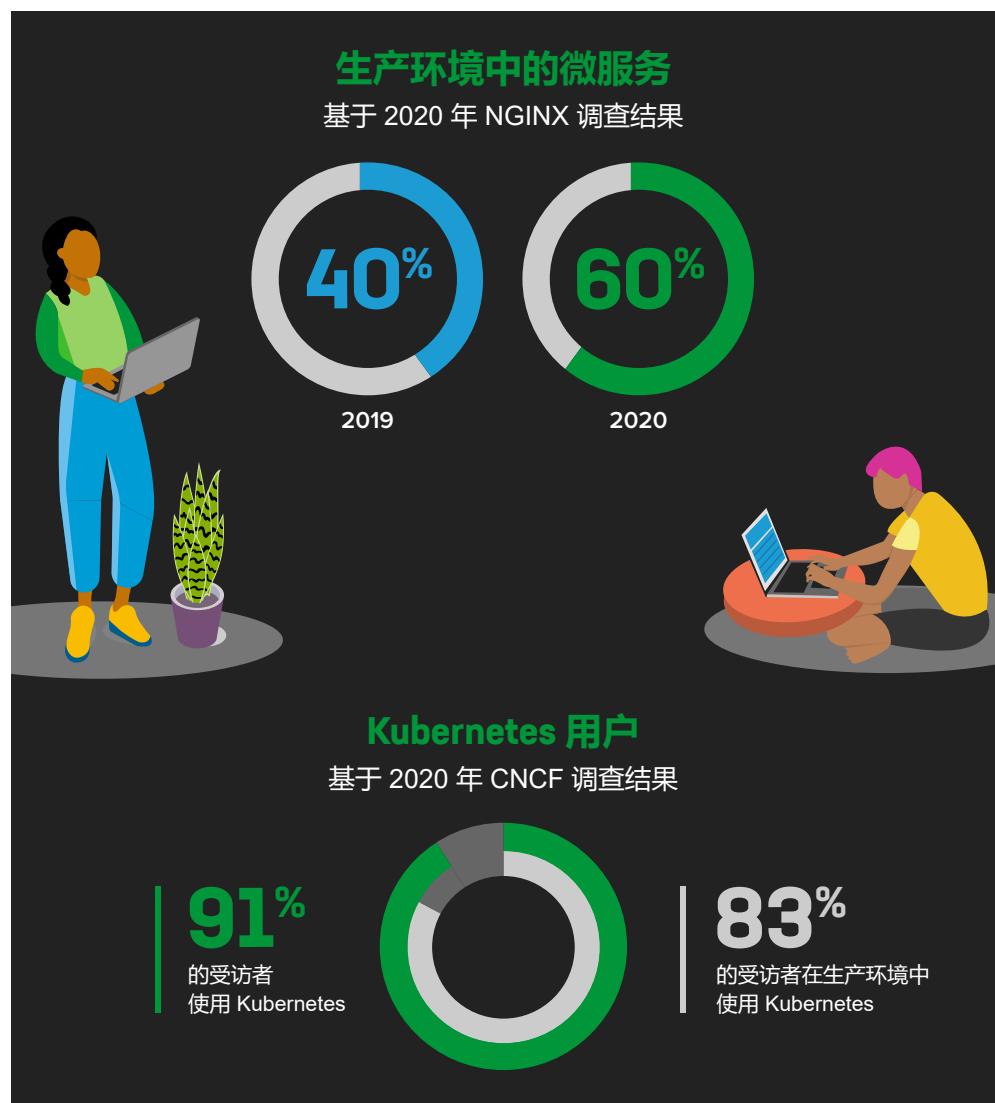
1. 生产级 Kubernetes 助您降低复杂性

现代的转变和趋势

2020 年是令人刻骨铭心的一年。学校、企业和公共服务的突然停摆让我们与社会隔绝开来，也让我们的人身安全和财务稳定性充满了不确定。现在我们想象一下，如果疫情发生在 2000 年，甚至是 2010 年，会和现在有什么不同？答案是技术。如果没有医疗、流媒体视频、远程协作工具等我们习以为常的高质量数字服务做保障，疫情可能就会是另外一番景象了。是什么让 2020 年的技术与过去几十年如此不同？答案是容器和微服务。

微服务架构通常会与容器和 Kubernetes 技术结合，能够缩短上市时间、提升数字体验，进而推动业务增长和技术创新。无论是与传统架构搭配部署还是独立使用，这些现代应用技术都可以显著改善扩展性、灵活性和部署速度，甚至能够节省成本。

在 2020 年之前，我们发现大多数客户已开始采用微服务作为其数字化转型战略的一部分，而疫情又极大地推动了应用的现代化步伐。我们在 2020 年对 NGINX 用户进行的一项调查发现，60% 的受访者在生产环境中使用微服务，较 2019 年（40%）有所上升。而容器的受欢迎程度是其他现代应用技术的 2 倍多。



Kubernetes 是公认的容器化应用管理标准。云原生计算基金会 (CNCF) 在 2020 年的一项调查中发现，91% 的受访者正在使用 Kubernetes，其中的 83% 已经在生产环境中使用。许多企业在采用 Kubernetes 时已经做好了进行重大架构变更的准备，但大规模运行现代应用技术对企业带来的影响却是他们没有想到的。

企业对于系统架构的设计往往反映了该企业自身的沟通形态

不同的团队选择不同的策略来满足相同的要求

容器化应用的分布式特性意味着攻击面要大得多

如果您正在运行 Kubernetes，您可能已经遇到了以下三个攸关业务的障碍：

• 文化

即使应用团队采用了敏捷开发和 DevOps 这样的现代方法，他们通常也摆脱不了康威定律，即“企业对于系统架构的设计往往反映了该企业自身的沟通形态”。换句话说，分布式应用由独立运转但资源共享的分布式团队开发。虽然这种沟通形态能够有效地提高团队的工作效率，但同时也容易形成孤岛，进而出现沟通低效（这将导致其他负面结果）、安全漏洞、工具蔓延、自动化实践不一致以及团队冲突等问题。

• 复杂性

企业要想实施企业级的微服务技术，就必须将一套关键组件组合在一起，以实现可视化、安全性和流量管理。通常，团队会使用基础架构平台、云原生服务和开源工具来满足这一需求。虽然每个策略都各有所用，但也都各有不足，可能会造成复杂性。同一企业结构内的不同团队常常会选择不同的策略来满足相同的要求，从而导致“运营债务”。此外，各个团队一旦在某个时间点选择了某个流程和工具，那么无论使用容器部署和运行现代微服务应用的需求如何变化，他们都会继续使用下去。

CNCF Cloud Native Interactive Landscape（云原生基金会的交互全景图）清楚地说明了支持微服务应用的必要基础架构的复杂性。企业需要精通各种不同的技术，这会造成基础架构锁定、IT 无所不入、工具蔓延以及基础架构维护人员的学习难度加大等问题。

• 安全性

云原生应用和传统应用的安全防护需求存在明显不同，因为 Kubernetes 中不存在“围栏（ringfenced）”策略。庞大的生态系统和容器化应用的分布式特性意味着攻击面更为广泛，而对外部 SaaS 应用的依赖则意味着员工和外部人员注入恶意代码或泄露信息的机会大幅增加。此外，上文“文化”和“复杂性”部分（尤其是工具蔓延）中提到的后果将直接影响现代应用的安全性和弹性。在生态系统中使用不同的工具解决相同的问题不仅效率低下，而且给 SecOps 团队带来了巨大的挑战，因为他们必须学习如何正确地配置每个组件。

解决方案：生产级 Kubernetes

与大多数企业结构问题一样，解决 Kubernetes 挑战的方法是将技术和流程相结合。本书重点介绍了如何使用流量管理工具来解决这些问题，内容涵盖了从测试到生产的整个过程。

Kubernetes 是一种开源技术，实现生产级 Kubernetes 的方式有很多。虽然一些企业更喜欢使用原生的 Kubernetes，但还有许多企业认为托管平台通过出色的灵活性、规范性及强大的支持提供了更多价值。

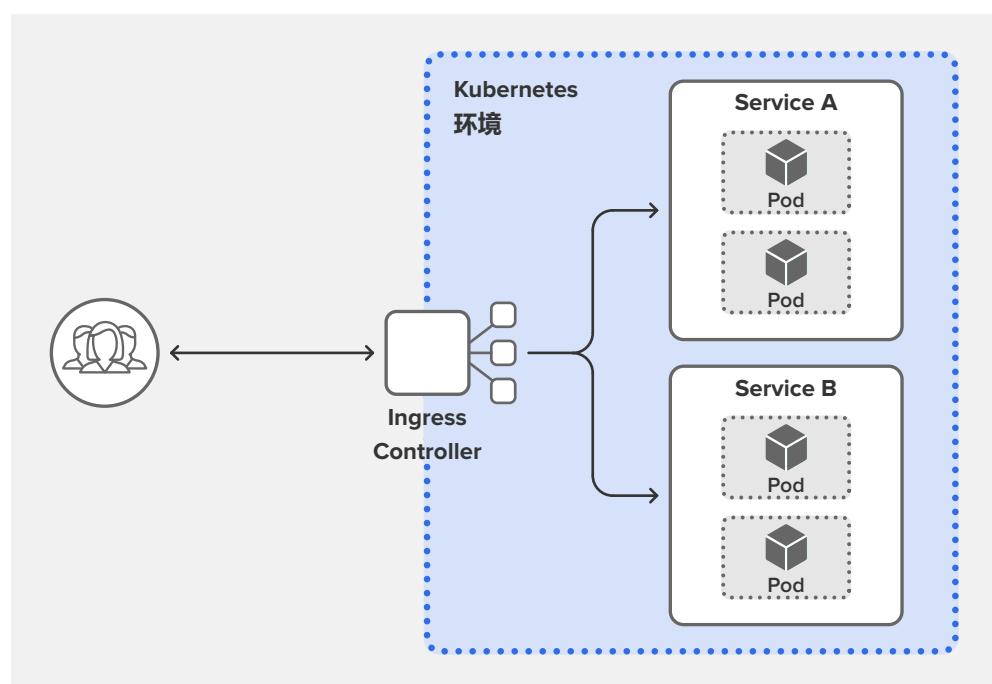
各种 Kubernetes 平台可以轻松启动和运行，但是它们所关注的是服务广度而非深度。您可以一站式获齐所需的所有服务，但无法获得大规模生产所需的所有功能。也就是说，它们不关注进阶的网络功能和安全性，而这正是许多客户对 Kubernetes 不满意的地方。

要实现生产级 Kubernetes，您需要按照以下顺序添加三个组件：

1. 可扩展的出入向层，用于控制进出集群的流量

这是通过 **Ingress controller** 实现的。Ingress controller 是一个专用负载均衡器，能够将 Kubernetes 网络的复杂性抽象出来，并在 Kubernetes 集群内部的 service 和外部的 service 之间建立一座桥梁。如果该组件包含有助于提高弹性（例如高级健康检查和 Prometheus 指标）、快速扩展（动态重新配置）和自助服务（基于角色的访问控制（RBAC））的特性，它就变成了生产级组件。

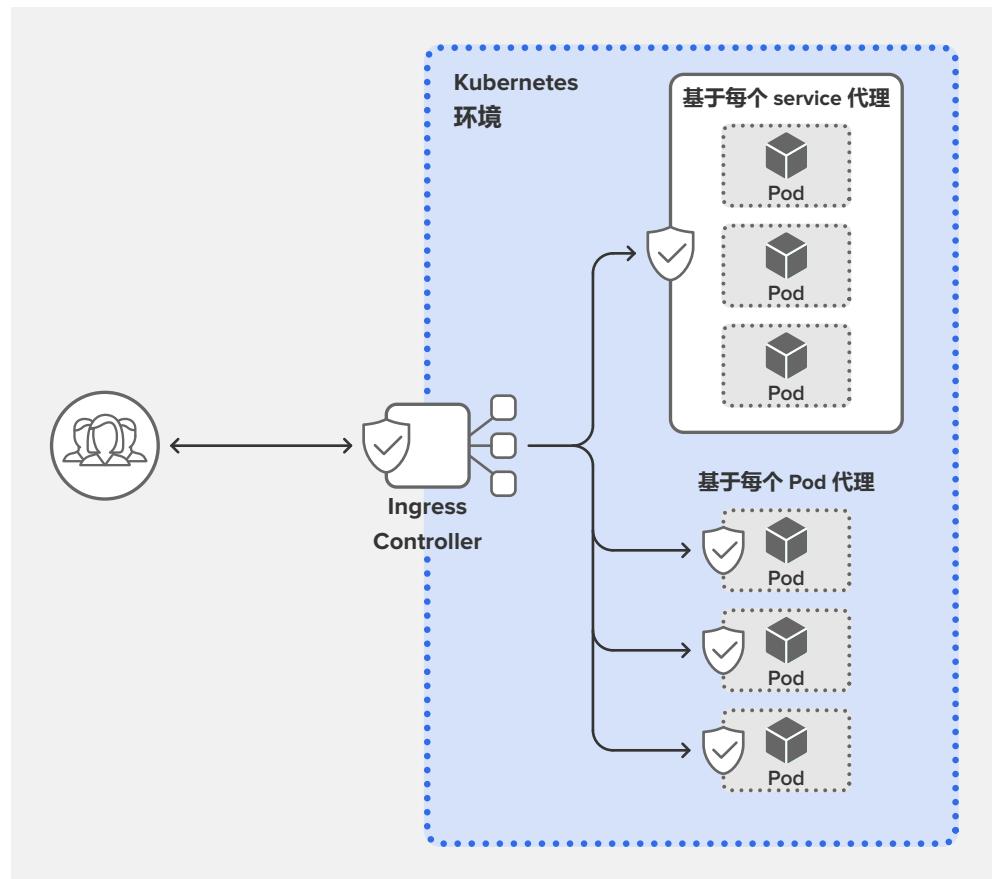
图 1：生产级 Kubernetes 部署的第一部分是一个可扩展的出入向层



2. 内置安全防护，防范整个集群中的威胁

虽然集群外部可能只要有“粗粒度”的安全防护就够了，但集群内部必须要有“细粒度”安全防护。根据集群的复杂程度，您可能需要在以下三个位置部署灵活的 [web 应用防火墙 \(WAF\)](#)：在 Ingress controller 上，以及基于每个 service 或每个 pod 进行代理。您可以对敏感应用实施更严格的控制（例如在计费方面），并对低风险应用放宽控制。

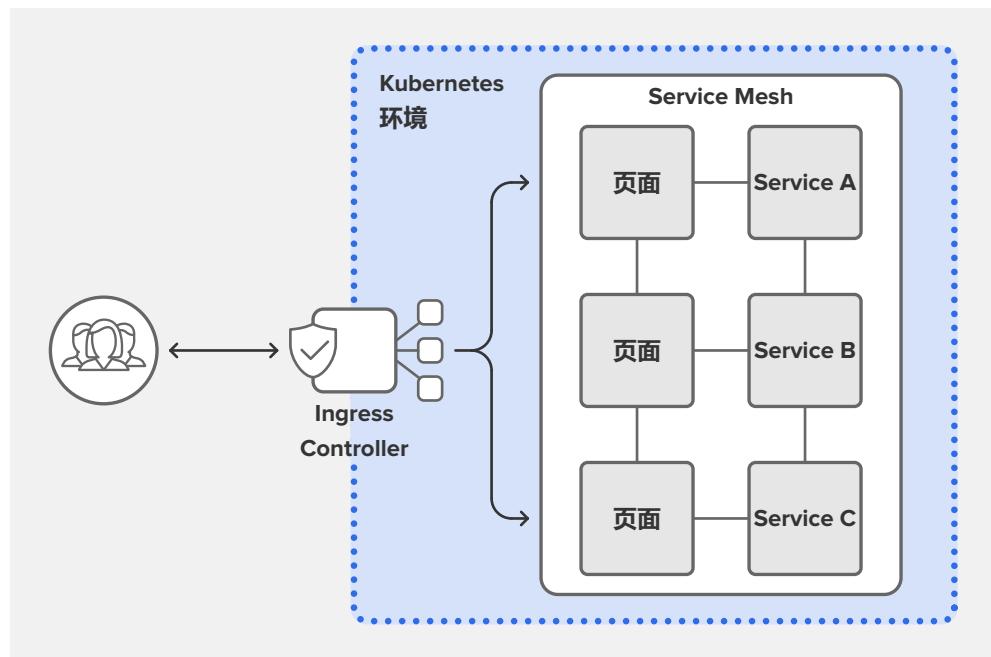
图 2：部署生产级 Kubernetes 的第 2 部分是内置的安全防护，例如灵活的 WAF



3. 可扩展的东西向流量层，用于优化集群内的流量

一旦 Kubernetes 应用的复杂性和规模超出基本工具的处理能力范围，就需要使用这第三个组件。在该阶段，您需要一个 **service mesh**，这是一种编排工具，可为集群内的应用服务提供更细粒度的流量管理和安全性。Service mesh 通常负责管理容器化应用之间的应用路由，提供和实施自动的 service 到 service 的双向 TLS (mTLS) 策略，并让应用的可用性和安全性变得可视化。若要了解更多关于“何时应该考虑使用 service mesh”的相关信息，请参阅第 9 章。

图 3：部署生产级 Kubernetes 的第 3 部分是可扩展的东西向流量层



Kubernetes 优先事项：弹性、可视性和安全性

在本书第一部分的剩余部分，我们将探讨生产级流量管理工具能够帮助您解决的三大 Kubernetes 问题：

- **弹性**

正常运行时间之所以是一个标准的 SLA，原因在于客户的心情总是变化无常，他们会很快放弃不可靠或不可用的产品。在[第 2 章](#)中，我们分享了六种高级流量管理方法，用于提高 Kubernetes 应用和基础架构的弹性。

- **可视化**

可视化的重要性再怎么强调也不足为过。如果没有它，您的团队根本无法获得所需的洞察，也就无法先客户一步检测到并预防问题的发生。在[第 3 章](#)中，我们将介绍如何使用从流量管理工具和集成中获得的洞察来解决两个常见的 Kubernetes 问题。

- **安全性**

对 Kubernetes 环境和应用保护不足会导致企业无法实现 Kubernetes 的相关目标。安全漏洞可能导致客户流失、敏捷性降低以及数字化转型停滞。在[第 4 章](#)中，我们将介绍六个使用案例，这些案例可以在避免降低开发人员工作效率的前提下，提高您 Kubernetes 环境和应用的安全性。

2. 提高弹性的六种方法

如何判断一家公司是否成功使用了现代应用开发技术呢？很简单，看看客户有没有在社交媒体上抱怨就知道了。客户可能会抱怨新片看不了，网银登不上去，购物车超时无法下单。



即使他们没有公开抱怨，也不代表就没有问题。我们的一位客户——一家大型保险公司曾告诉我们，如果他们未能在短短的 3 秒内加载出主页，便会面临客户流失的问题。

用户对性能差或宕机问题的所有抱怨都指向了一个共同的元凶：缺乏弹性。微服务技术（包括容器和 Kubernetes）好就好在它能提高应用弹性，显著改善客户体验。为什么呢？这一切得从架构说起。

微服务架构与单体架构存在本质上的区别。打个比方，在老式的节日灯串中，只要有一个灯坏了，整串就都不亮了。如果不换灯泡，那就只能将整串丢掉。单体应用就像这种老式灯串一样，其组件紧密耦合在一起，并且一损俱损。



照明行业和软件行业都发现了这个痛点——所以如果现代灯串上的某个灯泡发生故障，其他灯泡仍会继续照明。同样，设计良好的微服务应用，即使在某个服务实例出问题时，也会继续运行。

高级流量管理用例

容器具有轻便、可移植和易于扩展的特点，非常适合使用小型独立组件构建应用的场景，是微服务架构中颇受欢迎的一种选择。[Kubernetes](#) 是约定俗成的容器编排标准，但若投入生产环境，还面临着许多挑战。要想增强 Kubernetes 应用的可控性和弹性，成熟的流量管理策略是一个重要因素，它能够让您控制服务而不是数据包，并动态地或使用 Kubernetes API 调整流量管理规则。流量管理在任何架构中都很重要，但对于高性能应用来说，有两个流量管理工具是必不可少的：[流量控制](#)和[流量精分](#)。

- **流量控制**（有时称为“流量路由”或“流量整形”）是指控制流量的去向及其传输方式的行为。这是在生产环境中运行 Kubernetes 的必要条件，因为它可以让基础架构和应用免遭攻击和流量激增问题。在应用开发过程中，您需要采用“速率限制”和“断路”这两种技术。
- **流量精分**（有时称为“流量测试”）是流量控制的一个子类，目的是为了控制传入流量的比例，这些流量会被导向到在环境中同时运行的不同版本的后端应用（通常为当前的生产版本和更新版本）。流量精分是应用开发周期中的重要一环，允许团队在不影响客户的情况下测试新特性和新版本的功能和稳定性。实用的部署场景包括“调试路由、灰度部署、A/B 测试及蓝绿部署”。（业界对这四个术语的使用存在很大分歧。此处先按我们理解的定义来使用它们。）

在本章中，我们首先探讨可以使用流量控制和流量整形技术来解决的用例。然后我们将解释如何使用 NGINX 解决方案实现几种技术。这六个用例是：

1. 防止服务接收过多请求
2. 避免级联故障
3. 在生产环境中测试新版本
4. 确保新版本稳定
5. 了解客户更喜欢新版本还是当前版本
6. 在无需停机的条件下将用户迁移到新版本

速率限制技术可限制用户在给定时间段内的请求数量

实现弹性的用例 1: 防止服务接收过多请求

解决方案：速率限制

无论 HTTP 请求是恶意的（例如暴力密码破解和 DDoS 攻击）还是正常的（例如顾客蜂拥抢购），大量的 HTTP 请求都会导致服务瘫痪和应用崩溃。速率限制技术可限制在给定时间段内的用户请求数量。这些请求可能非常简单，例如访问网站主页的 GET 请求，或是登录表单上的 POST 请求。举例来说，当受到 DDoS 攻击时，您可以使用速率限制将传入的请求速率限制为真实用户的典型值。

实现弹性的用例 2:

避免级联故障

解决方案：断路器

当服务不可用或出现高延迟时，传入请求的超时时间以及客户端收到错误响应的时间可能很长。长超时可能会造成级联故障，即一个服务中断导致其他服务超时，最终引发整个应用故障。

断路器可监控服务故障，防止发生级联故障

断路器可监控服务故障，防止发生级联故障。当服务请求的失败次数超过预先设定的阈值时，将触发断路器，断路器一收到请求就向客户端返回错误响应，从而对服务进行限流。

断路器将持续拦截和拒绝请求，等过了指定的时长后再放行有限数量的请求以作测试。如果这些请求成功，断路器将停止限流。如果不成功，则开始重新计时，期间断路器继续拒绝请求。

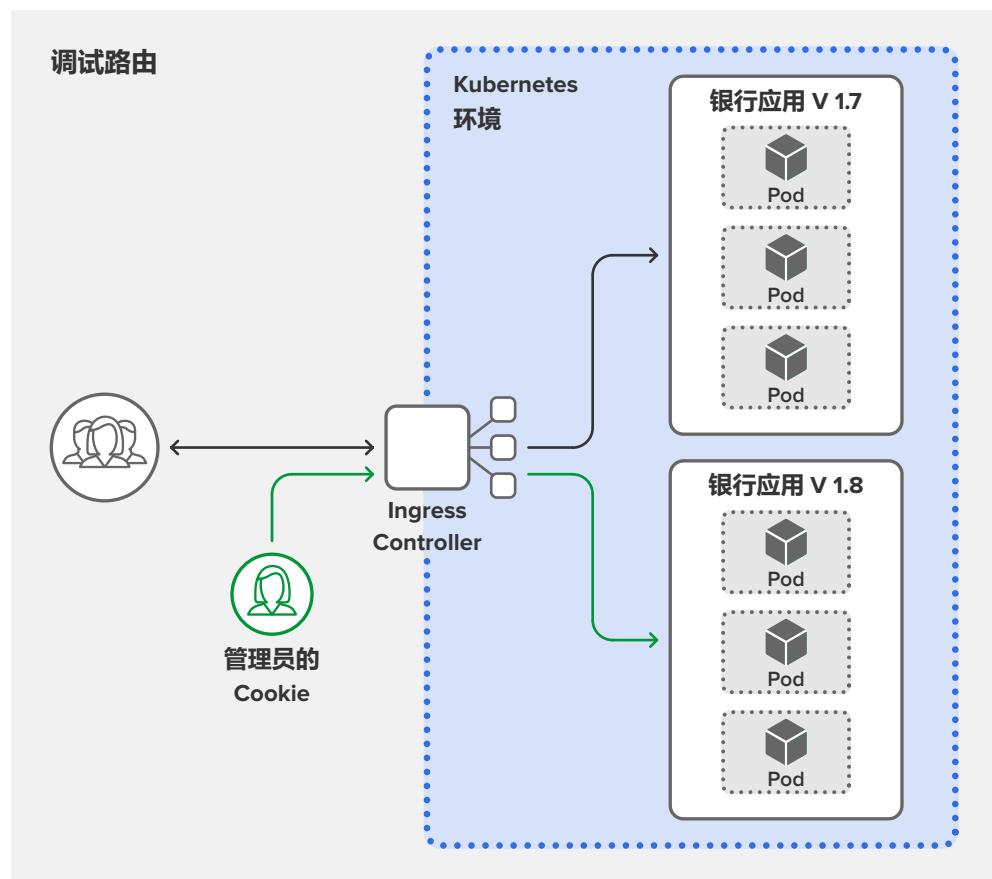
调试路由允许您公开部署应用，同时又向大部分用户“隐藏”它

实现弹性的用例 3： 在生产环境中测试新版本

解决方案：调试路由

假设您有一个银行应用，现在您要为其添加一个信用评分功能。在进行客户测试之前，您可能想要看一下它在生产环境中的表现。调试路由（也称为“条件路由”）允许您公开部署新功能，同时又向真正的用户“隐藏”它。根据会话 cookie、会话 ID 或组 ID 等七层属性，调试路由可以只允许特定用户访问它。例如，您可以只允许拥有管理会话 cookie 的用户访问——他们的请求将被路由到具有信用评分功能的新版本，而其他用户则继续访问稳定版。

图 4：Kubernetes 环境中的调试路由



灰度部署为测试新特性或新版本的稳定性提供了一种安全、敏捷的方法

实现弹性的用例 4:

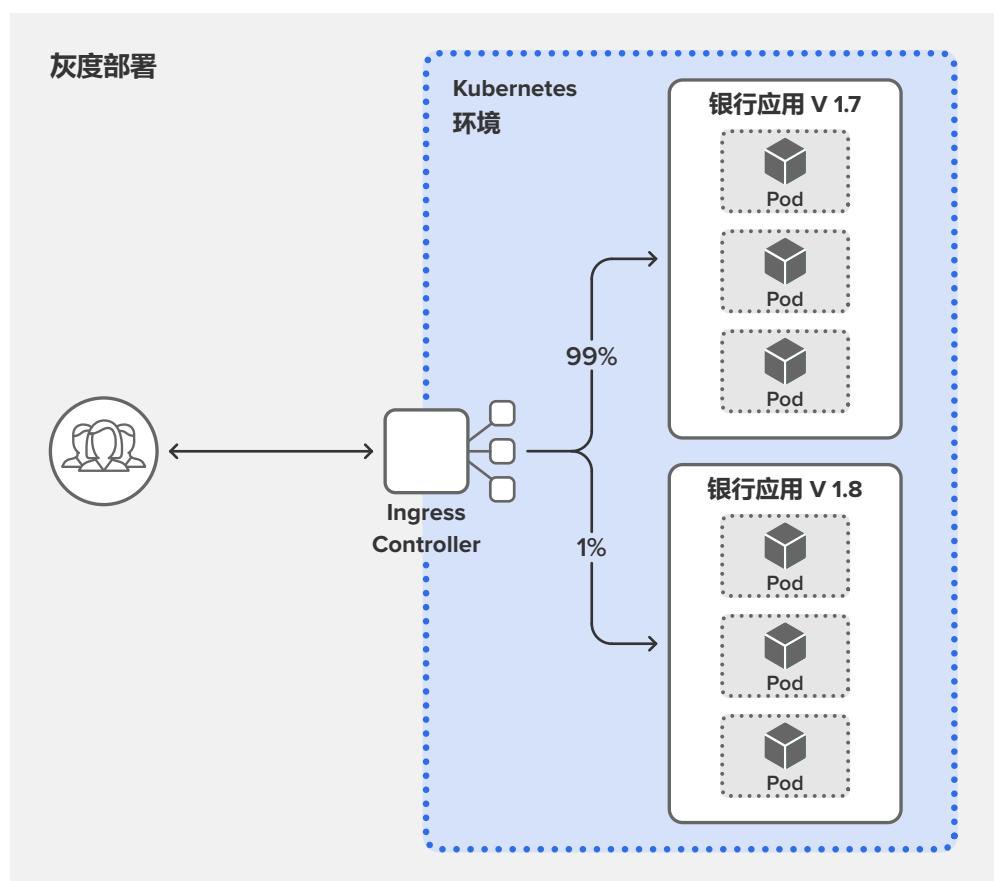
确保新版本稳定

解决方案：灰度部署（金丝雀部署）

灰度部署的概念来自一个历史悠久的采矿实践，当时的矿工将金丝雀装在笼子里带入矿井，一旦发现金丝雀中毒就紧急撤离，因此金丝雀是“瓦斯报警鸟”。在应用世界里，不会再有牺牲品了。灰度部署为测试新特性或新版本的稳定性提供了一种安全、敏捷的方法。典型的灰度部署是，先让绝大多数（比如 99%）用户使用稳定版，然后将一小部分用户（剩余的 1%）转移到新版本。如果新版本出现问题（例如崩溃或向客户端返回错误），您可以立即将测试用户转移回稳定版。如果新版本顺利运行，您可以一次性或以可控的方式逐步（更为常见）将用户从稳定版迁移到新版本。



图 5：Kubernetes 环境中的灰度部署

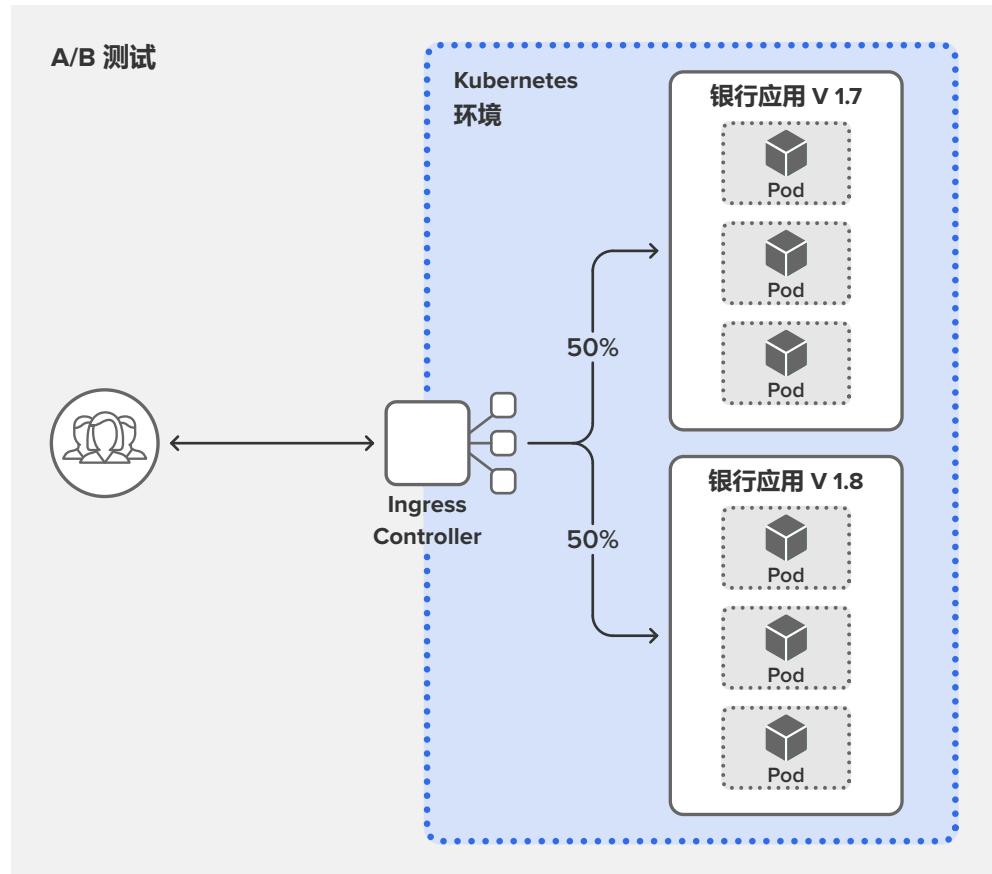


A/B 测试是一种许多行业用来衡量和比较用户行为的测试流程

实现弹性的用例 5： 了解客户更喜欢新版本还是当前版本 解决方案：A/B 测试

确认新特性在生产环境中运行无误后，您可能还希望了解客户对该特性的反馈，包括点击量、重复用户或显式评分级等关键性能指标（KPI）。许多行业都使用 A/B 测试流程来衡量和比较用户行为，目的是确定不同产品或应用版本在客户群体中的受欢迎程度。在典型的 A/B 测试中，50% 的用户访问版本 A（当前的应用版本），剩余 50% 的用户访问版本 B（包含稳定的新功能的版本）。KPI 综合得分最高的版本将胜出。

图 6：Kubernetes 环境中的 A/B 测试



实现弹性的用例 6： 在无需停机的条件下将用户迁移到新版本

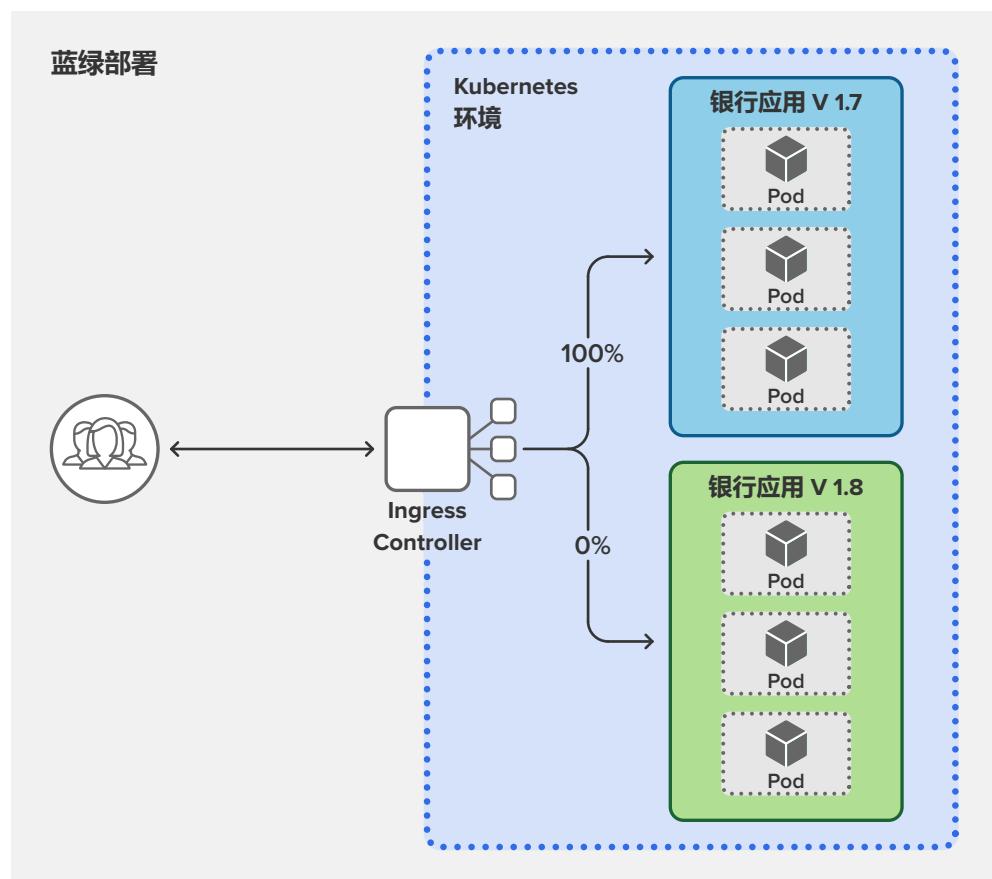
解决方案：蓝绿部署

蓝绿部署极大地减少甚至消除了
升级的停机时间

现在，假设您的银行应用即将进行重大版本变更，那么，恭喜您！过去，版本升级通常意味着用户停机，因为您必须先从生产环境中移除旧版本，然后才能再推送新版本。但在当今竞争激烈的环境中，大多数用户都无法接受停机升级。蓝绿部署极大地减少甚至消除了升级的停机时间。您可以继续在生产环境中运行旧版本（蓝），同时在该生产环境中部署新版本（绿）。

大多数企业都不愿意将所有用户一次性从蓝色转移到绿色，毕竟，如果绿色版本发生故障怎么办？！解决方案是使用灰度部署，以最符合您的风险规避策略的增量方式转移用户。如果新版本是一场灾难，您只需敲击几下键盘，便可以轻松地将每个人转移回稳定版。

图 7：Kubernetes 环境中的蓝绿部署



使用 NGINX 进行高级 Kubernetes 流量管理

大多数 [Ingress controller](#) 和 [service mesh](#) 都可以帮助您实现高级流量控制和精分。使用哪种技术取决于您的应用架构和用例。例如，Ingress controller 适用于以下三种场景：

- 您的应用只有一个端点，就像您迁移到 Kubernetes 的简单应用或单体应用一样。
- 集群中没有 service 到 service 通信。
- 集群中有 service 到 service 通信，但您还没有使用 service mesh。

如果您的部署复杂到要使用 service mesh 的地步，一个常见的用例是通过分割服务流量来测试或升级各个微服务。举例来说，在移动端前端后台，您可能想在两个不同版本的地理位置微服务 API 之间进行灰度部署。

然而，有些 Ingress controller 和 service mesh 在设置流量精分时不仅非常耗时，而且容易出错，原因有很多：

- 不同厂商的 Ingress controller 和 service mesh 具有不同的 Kubernetes 功能实施方式。
- Kubernetes 不是专为管理和理解七层流量而生。
- 有些 Ingress controller 和 service mesh 不支持复杂的流量管理。

在几秒钟内轻松配置稳健的流量路由和流量精分策略

借助 [F5 NGINX Ingress Controller](#) 和 [F5 NGINX Service Mesh](#)，您可以通过更简单的配置、高级自定义功能和改进的可视化，在几秒钟内轻松配置稳健的流量路由和流量精分策略。

Annotations、ConfigMap 和自定义模板缺乏细粒度控制，并且有不安全、易出错和难使用的问题

NGINX Ingress 资源和 SMI 规范助您简化配置

以下 NGINX 功能简化了配置流程：

- **面向 NGINX Ingress Controller 的 NGINX Ingress 资源** —— 虽然标准的 Kubernetes Ingress 资源可简化 SSL/TLS 终止、HTTP 负载均衡和七层路由的配置，但它不具备断路、A/B 测试和蓝绿部署所需的定制功能。因此，非 NGINX 用户必须求助于 Annotation、ConfigMap 和自定义模板，但它们都缺乏细粒度控制，并且有不安全、易出错和难使用的问题。

NGINX Ingress Controller 自带 **NGINX Ingress 资源**，作为标准 Ingress 资源（同时也支持该资源）的替代方案。该资源采用了一种原生、类型安全的缩进式配置风格，可简化 Ingress 负载均衡的实施。对现有 NGINX 用户来说，NGINX Ingress 资源还有一个额外的好处：它们可以简化非 Kubernetes 环境中负载均衡配置的再利用，从而支持所有 NGINX 负载均衡器使用相同的负载均衡配置。

- **遵循 SMI 的 NGINX Service Mesh** —— NGINX Service Mesh 能够实施 Service Mesh Interface (SMI)，SMI 是一个规范，定义了在 Kubernetes 上运行的 service mesh 的标准接口，具有 TrafficSplit、TrafficTarget 和 HTTPRouteGroup 等类型化资源。借助标准的 Kubernetes 配置方法，NGINX Service Mesh 和 **NGINX SMI 扩展程序** 可简化流量精分策略（如灰度部署）的部署，并最大限度地减少对生产流量的中断。以下是使用 NGINX Service Mesh 定义灰度部署的示例：

```
apiVersion: split.smi-spec.io/v1alpha2
kind: TrafficSplit
metadata:
  name: target-ts
spec:
  service: target-svc
  backends:
    - service: target-v1-0
      weight: 90
    - service: target-v2-0
      weight: 10
```

我们的教程《[如何使用 NGINX Service Mesh 进行流量精分](#)》介绍了使用到流量精分的部署模式示例，包括灰度部署和蓝绿部署。

先进的断路器能够更快速地检测故障并进行故障转移，从而帮助您节省时间并提高弹性

借助高级定制实现更复杂的流量控制和流量精分

NGINX 的以下功能能够以更高级的方式简化流量控制和流量精分：

- **面向灰度部署的键值存储** —— 当您执行 A/B 测试或蓝绿部署时，您可能希望按特定的增量（例如 0%、5%、10%、25%、50% 和 100%）将流量过渡到新版本。大多数工具都需要很多人工操作，因为您必须为每个增量编辑百分比并重新加载配置文件。面对不小的工作量，您可能会冒险直接从 5% 过渡到 100%。然而，借助基于 NGINX Plus 的 NGINX Ingress Controller，您可以[利用键值存储更改百分比，且无需重载配置文件](#)。
- **通过 NGINX Ingress Controller 断路** —— 先进的断路器能够更快速地检测故障和故障转移，甚至针对不健康的上游激活定制的格式化错误页面，从而帮助您节省时间和提高弹性。举例来说，搜索服务的断路器可能会返回一组格式正确但内容为空的搜索结果。为了达到这种效果，基于 NGINX Plus 的 NGINX Ingress Controller 采用了[主动健康检查](#)，主动监控 TCP 和 UDP 上游服务器的健康状况。由于监控是实时的，您的客户端遭遇应用错误的概率将大大降低。
- **通过 NGINX Service Mesh 断路** —— NGINX Service Mesh [断路器规范](#)具有三个自定义字段：
 - Errors —— 断路器触发前的错误数
 - timeoutSeconds —— 断路器触发前发生错误的窗口，以及断路器关闭前的等待时间
 - fallback —— 断路器触发后，流量被重新路由到的 Kubernetes service 的名称和端口

errors 和 timeoutSeconds 是标准的断路器功能，而 fallback 支持您定义备份服务器，进一步提高了弹性。如果您的备份服务器响应是独一无二的，它们可以作为集群故障的早期指示器，让您第一时间开启故障排除。

解析流量精分结果

现在，您已实现了流量精分……接下来该做什么呢？接下来我们应该分析流量精分结果。这可能是最难的一个环节，因为许多企业都缺乏对 Kubernetes 流量和应用运行情况的关键洞察。如欲了解有关“如何通过改进可视化获取洞察”的更多信息，请继续阅读[第 3 章](#)。

您的架构可能会给您带来更大
风险

3. 更出色的可视化能够帮助您解决两个问题

虽然采用微服务可以改善数字体验，但微服务架构也会让这些体验变得更加脆弱。在开发人员紧锣密鼓地推出新应用的同时，您的架构可能会带来更大的中断风险和安全隐患，您可能还要耗费时间进行低效的故障排除或修复可预防的问题。在本章中，我们将探讨在微服务环境中，如何通过流量可视化来降低复杂性并提高安全性。

如何通过可视化获取洞察

首先，我们来看几个定义。

- **可视化** —— 能够看到或被看到的状态
- **洞察** —— 对人或物拥有深刻的理解

在 Splunk 2021 年的一项调查中，81% 的受访者认为数据“非常”或“极其”有价值。我们也认为数据十分重要，尤其是在很难知道部署了哪些内容的 Kubernetes 环境中。然而，在 F5 的《2021 年应用策略现状》报告中，95% 的受访者表示，尽管他们拥有丰富的数据，但对于保护和发展基础架构和业务所需的应用的性能、安全性和可用性，他们仍然无法获取足够的洞察。洞察为何如此重要？如何获得洞察？

洞察可帮助您：

- | | |
|--|---|
| <ul style="list-style-type: none">✓ 检测漏洞和潜在攻击向量，增强安全性与合规性✓ 先于客户发现问题，减少中断和停机时间✓ 查找应用问题的根源，提高故障排除效率✓ 确认流量正确路由 | <ul style="list-style-type: none">✓ 准确了解 Kubernetes 环境中的运行项目及其是否得到合理配置和保护✓ 根据延迟和性能历史记录确定您是否使用了适当的资源数量✓ 基于过去的流量模式预测季节性需求✓ 测量响应时间以追踪性能并和服务等级协议（SLA）进行比较，同时用作预警系统，从而在问题影响用户体验之前就能及时识别 |
|--|---|

出于组织架构方面的原因，人们未能获得宝贵的洞察

要获得洞察，您需要两种类型的可视化数据：实时数据和历史数据。实时数据可帮助您诊断当前问题的来源，而历史数据可帮助您辨别正常与异常。这两种类型的可视化来源相结合，可提供关于应用和 Kubernetes 性能的关键洞察。

与其他技术投资一样，您还需要制定策略来帮助您实现相关收益。[F5 报告](#)还指出，出于企业方面的原因，例如招聘和员工发展、战略和流程以及数据用途、使用情形和使用人员的分歧，人们未能获得宝贵的洞察。调查结果显示：



发现并解决两个 Kubernetes 问题

大多数 Kubernetes 部署已经有一个监控工具了，不再需要其他工具了。但是您是否知道可以使用生产级流量管理工具（特别是 Ingress controller 和 service mesh）获取洞察？由于能接触到 Kubernetes 集群中的所有的流量，Ingress controller 和 service mesh 有可能提供重要的数据，帮助您优化正常运行时间的 SLA。

- **了解出入向（南北）流量**

Ingress controller 能够帮助您了解进出 Kubernetes 集群的流量。

- **了解 service 到 service（东西）流量**

Service mesh 能够帮助您了解容器化的多个应用之间的流量。

为了在 NGINX 工具中实现这种可视化水平，我们提供了 [F5 NGINX Plus API](#)，既可帮您轻松导出指标，也可与 [OpenTracing](#) 以及 [Grafana](#) 和 [Prometheus](#) 等热门工具相集成，从而为您提供集群内性能的全面洞察。通过深度跟踪，您可以有针对性地获得应用性能和可用性的洞察，从而了解请求在微服务应用中的处理过程。

在本章的剩余部分中，我们将借助 [F5 NGINX Ingress Controller](#) 和 [F5 NGINX Service Mesh](#) 通过六种方法来解决两个常见 Kubernetes 问题：

1. 应用运行缓慢
2. 资源耗尽

Kubernetes 问题 1：应用运行缓慢

[您只有在找到问题所在之后才能着手解决问题](#)

您的应用是否运行缓慢...甚至完全崩溃？您有没有想过可能遭到了 DDOS 攻击？用户是否报告了网站的错误？您只有在找到问题所在之后才能着手解决问题。根据应用的复杂性以及您使用的 NGINX 工具，您可以通过三种方式来获取实时指标，以便可以即时诊断当前出现的问题。

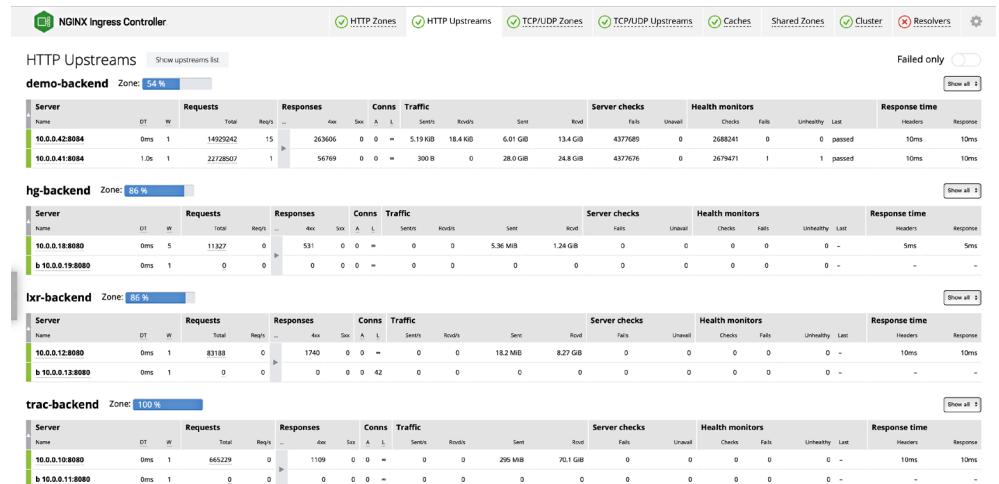
- **选项 1：借助 NGINX Ingress Controller 进行实时监控**

基于 NGINX Plus 的 NGINX Ingress Controller 提供了一个实时活动监控仪表盘（由 NGINX Plus API 提供支持），可显示数百个关键负载和性能指标。仪表盘可以提供细粒度信息，甚至细化到单个 pod 级别，可帮助您快速、轻松地衡量应用响应时间，并诊断问题的来源。随着您的 Kubernetes 环境不断扩展，每个新增的 NGINX Ingress Controller 实例都会自动获得新仪表盘。

例如，**HTTP 上游 (HTTP Upstreams)** 选项卡上的两列直观地显示了应用和基础架构状态：

- **请求** —— 如果每秒请求数 (**Req/s**) 降至给定应用标准以下（例如，每秒 5 个请求，而正常请求数为 40），则表示 NGINX Ingress Controller 或应用的配置可能不正确。
- **响应时间** —— 如果响应时间为 10 毫秒 (ms) 或更少，则表示运行状况良好。延迟超过 30-40 毫秒表示上游应用出现问题。

图 8：NGINX Ingress Controller 实时监控仪表盘



503 和 40x 错误表示存在资源问题，而 502s 表示配置变更不成功

- **选项 2：NGINX Ingress Controller 的 stub status**

NGINX 开源版 NGINX Ingress Controller 提供了一个[状态页面](#)，展示了八个基本指标。

- **选项 3：结合使用 NGINX Service Mesh 和 OpenTracing**

NGINX Service Mesh 已通过[NGINX OpenTracing 模块](#)支持 OpenTracing。在撰写本文时，NGINX OpenTracing 模块已支持 DataDog、LightStep、Jaeger 和 Zipkin。

Kubernetes 问题 2：资源耗尽

出现了 HTTP 错误？503 和 40x 错误表示存在资源问题，而 502s 表示配置变更不成功。要诊断可能出现资源耗尽问题的位置，您需要能够将指标导出到可视化工具，且这种工具需要能够绘制随时间变化的值。除了发现问题的根源之外，历史数据还可以帮助您预测何时会出现流量激增，以便您做好扩展的准备。

- **选项 1：借助 NGINX Ingress Controller 进行日志记录**

诊断网络问题的第一步是查看[NGINX Ingress Controller 日志](#)，其中每个日志条目（包括有关错误的日志条目）都标识了关联的 Kubernetes service。日志包含了流经 NGINX Ingress Controller 的所有流量的详细信息，包括时间戳、源 IP 地址和响应状态码。您还可以将日志导出到流行的聚合器中，例如 DataDog、Grafana 和 Splunk。

- **选项 2：Prometheus 指标**

NGINX Ingress Controller 最受欢迎的特性之一是其不断扩展的[Prometheus 指标](#)列表，其中包含网络性能和 Ingress controller 流量指标。基于 NGINX Plus 的 NGINX Ingress Controller 能够[导出一系列相关指标](#)，涵盖连接、缓存、由 NGINX worker 组（[在内存区共享数据](#)）处理的 HTTP 和 TCP/UDP 流量、由[后端服务器组](#)处理的 HTTP 和 TCP/UDP 流量等。

NGINX Service Mesh 使用 NGINX Plus API 从 NGINX Service Mesh sidecar 和 NGINX Ingress Controller pod 获取指标。其中包含[Prometheus scrape 配置](#)，因此您可以自定义 Prometheus 配置文件中所需的指标。

- **选项 3：Grafana 仪表盘**

我们为[NGINX Ingress Controller](#) 和[NGINX Service Mesh](#) 提供了官方 Grafana 仪表盘，能够将 Prometheus Exporter 公开的指标可视化。用户非常重视数据的粒度，包括精确到毫秒的详细信息、逐日覆盖和流量峰值。例如，NGINX Service Mesh 仪表盘可以显示任何一个服务或 pod 的流量以及被监控的活动 pod 的数量，以标示 pod 的运行负载情况。

4. 提高安全性的六种方法

我们曾在《[在不影响速度的同时保护云原生应用](#)》一文中探讨过，有三个因素使[云原生应用](#)比传统应用更难以保护：

1. 云原生应用交付导致工具泛滥以及不一致的企业级服务
2. 云原生应用交付成本可能非常高且不可预测
3. SecOps 团队难以保护云原生应用，并与 DevOps 存在分歧

虽然这三点都会影响安全性，但也许是由于第三点涉及到人的因素最多，它可能是最棘手的问题。如果 SecOps 没有能力或没有足够的权限保护云原生应用，有些后果可能很明显（漏洞和违规），有些后果却很隐蔽，例如敏捷性降低、数字化转型停滞不前等。

当 Kubernetes 环境发生安全事件时，多数企业又会将他们的 Kubernetes 部署挪到生产环境之外

我们来深入探讨一下这些隐性代价。企业之所以选择 [Kubernetes](#)，是因为它有望提高敏捷性并降低成本。但是当 Kubernetes 环境发生安全事件时，[多数企业又会将他们的 Kubernetes 部署挪到生产环境之外](#)。且不说这会减缓事关企业未来发展的数字化转型的步伐，就连投入的工程设计和资金也付诸东流了。从逻辑上来说，如果您希望 Kubernetes 完成从测试到生产的旅程，那么就必须将安全性视为重要的战略组成部分，让企业中的每个人都予以高度重视。

安全工具生态系统是一个巨大的（并且不断发展的）行业，科技巨头和初创公司都加入了解决 Kubernetes 特有问题的行列。但即使采用了安全工具，例如漏洞扫描器和 WAF，企业仍然容易受到攻击。其实就像您的可视化和故障排除策略一样，您可以利用 Kubernetes 流量管理工具来优化和简化您的安全策略。

安全防护及身份验证的相关术语

在介绍用例之前，我们先快速了解一下您将在本章中遇到的安全防护和身份验证术语。您已经是安全专家了？[点击此处跳转到用例](#)。

- **身份验证和授权** —— 用于确保只有“正确”的用户和服务可以访问后端或应用组件，是系统必备的功能：
 - **身份验证** —— 验证“身份”，确保发出请求的客户端与自己声称的身份相符。通过 ID 令牌实现，例如密码或 JSON Web Tokens (JWTs)。
 - **授权** —— 验证资源或功能的访问“权限”。通过访问令牌实现，例如会话 cookie、会话 ID、组 ID 或令牌内容等七层属性。

- **通用漏洞披露 (CVE)** —— 这是一个漏洞数据库，它公开披露了“软件、固件、硬件或服务组件中可能会遭到利用的漏洞缺陷，这些缺陷会破坏受影响的一个或若干组件的机密性、完整性和可用性”。¹ CVE 可能由管理工具的开发人员、渗透测试人员、用户或客户或社区人员（例如漏洞猎人）发现。在公开披露漏洞之前，披露者通常会给软件所有者一定的时间来开发补丁，以免给黑客带来可乘之机。
- **拒绝服务 (DoS) 攻击** —— 一种网络攻击，恶意请求 (TCP/UDP 或 HTTP/HTTPS) 犹如洪水般涌向受害网站，目的是让网站瘫痪。DoS 攻击会影响可用性，因此它们造成的主要后果就是受害者的声誉受损。**分布式拒绝服务 (DDoS) 攻击**是指多个攻击源瞄准同一个网络或服务进行攻击，由于攻击者的潜在网络规模较大，防御起来也更加困难。DoS 防护需要一种可以自适应识别和防止攻击的工具。更多信息请参阅《[分布式拒绝服务 \(DDoS\) 详解](#)》。
- **端到端加密 (E2EE)** —— 对从用户传输到应用和后端的数据全程进行完全加密。E2EE 需要 SSL/TLS 证书，并可能也需要 mTLS。
- **双向 TLS (mTLS)** —— 一种对客户端和主机都进行身份验证（通过 SSL/TLS 证书）的实践。使用 mTLS 还可以保护在客户端和主机之间传输的数据的机密性和完整性。mTLS 的实施可以细化到 Kubernetes pod 级别、同一集群的两个 service 之间。有关 SSL/TLS 和 mTLS 的详细介绍，请参阅 F5 Labs 的《[什么是 mTLS?](#)》。
- **单点登录 (SSO)** —— SSO 技术（包括 SAML、OAuth 和 OIDC）可以简化身份验证和授权的管理。
 - **简化的身份认证** —— SSO 让用户省去了针对每个不同的资源或功能设置唯一 ID 令牌的麻烦。
 - **标准化授权** —— SSO 有助于根据角色、部门和资历设置用户访问权限，无需为每个用户单独配置权限。
- **SSL (安全套接字层) /TLS (传输层安全)** —— 一种用于在联网计算机之间建立经过身份验证和加密的链接的协议。SSL/TLS 证书能够验证网站的身份并建立加密连接。虽然 SSL 协议已经在 1999 年弃用并被 TLS 协议取代，但人们通常仍然将这些相关技术称为“SSL”或“SSL/TLS”——为了保持文本内容的一致，本章均使用“SSL/TLS”一词。
- **Web 应用防火墙 (WAF)** —— 一种[反向代理](#)，能够检测和拦截针对应用和 API 的复杂攻击（包括 [OWASP 十大安全漏洞](#)和其他高级威胁），同时让“安全”的流量顺利通过。Web 应用防火墙能够识破黑客的“奸计”，防止他们窃取敏感数据或劫持系统。有些厂商将 WAF 和 DoS 防护整合到了一种工具中，有些厂商则提供独立的 WAF 和 DoS 防护工具。

1. 来源：[CVE 计划](#)。

- **零信任** —— 一种经常用于高等安全企业的安全概念，但与每个人息息相关，数据必须在存储和运输的所有阶段都得到保护。零信任意味着企业决定在默认情况下不“信任”任何用户或设备，且所有流量都必须经过全面检查。零信任架构通常会包括身份验证工具、授权工具以及 mTLS，并且企业实施端到端加密的可能性很大。

让安全成为每个人的责任

本章介绍了您可以使用 Kubernetes 流量管理工具解决的六个安全用例，该工具可以使 SecOps 与 DevOps 和 NetOps 更好地协作，从而共同保护云原生应用和 API。您可以搭配使用这些技术以构建一套全面的安全战略，从而在保护应用和 API 的同时，最大限度地减少对客户的影响。

1. 快速解决 CVE 漏洞，有效避免网络攻击
2. 抵御 OWASP 十大安全漏洞和 DoS 攻击
3. 将身份验证和授权从应用层卸载下来
4. 设置有“防护栏”的自助服务
5. 实施端到端加密
6. 确保客户端使用可信的强密码

安全用例 1：

快速解决 CVE 漏洞，有效避免网络攻击

解决方案： 使用能够发出及时主动的补丁通知的工具

波耐蒙研究所的一项研究指出，在 2019 年，企业平均有 43 天的“宽限期”来开发和发布重大漏洞或高优先级漏洞的补丁，否则就会有攻击者利用漏洞图谋不轨。F5 NGINX 发现，这个窗口在接下来的几年里明显缩短了（2021 年的 Apple iOS 15 事件甚至连一天也没有），也是出于这个原因，我们建议尽快打补丁。但是，如果在 CVE 发布后的几周甚至几个月内，您的流量管理工具都没有可用的补丁，怎么办？

Jose Rodriguez @VBarraquito · Sep 15
In hopes Apple realizes that is being tightwad rewarding security bug reports, and reconsider the bounties.

Jose Rodriguez @VBarraquito · Sep 15
I will giveaway a very minor Lock Screen Bypass working in iOS 14.8 / 15RC
Apple values reports of issues like this with UP TO \$25,000 but for reporting a more serious issue I was awarded with \$5,000
I will send in private a PoC video to who asks for it when iOS 15 is public.

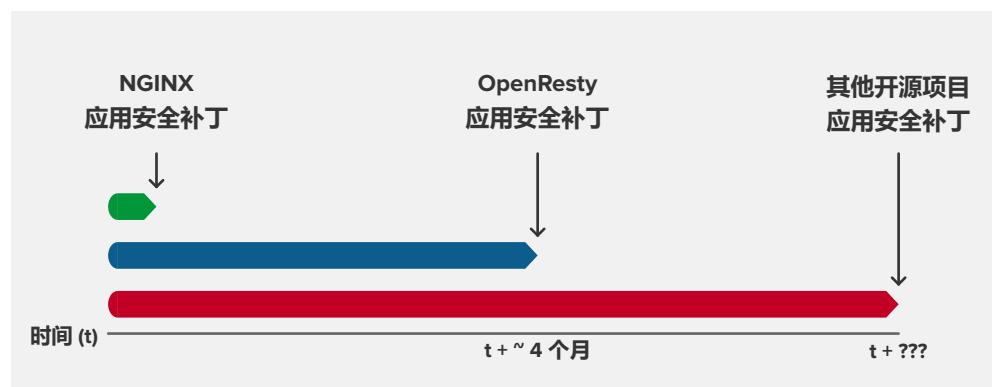
Physical Access to Device: Lock Screen Bypass
“Sensitive data” access includes gaining a small amount (i.e., one or two items), partial access (i.e., some large number), or broad access (i.e., the full database) from Contacts, Mail, Messages, Notes, Photos, or real-time or historical precise location data — or similar user data — that would normally be prevented by the system.

10:53 AM · Sep 15, 2021 · Twitter for iPhone

在之前的一个案例中，OpenResty 花了四个月的时间来应用与 NGINX 相关的安全补丁

由社区贡献者（而不是专门的工程团队）开发和维护的工具可能会在 CVE 发布以后滞后几周或几个月，导致企业难以在 43 天内完成打补丁。在之前的一个案例中，OpenResty 花了四个月的时间来应用与 NGINX 相关的安全补丁。这导致使用基于 OpenResty 的 Ingress controller 的任何人至少在四个月的时间里存在风险敞口。而对于依赖其他开源项目的软件来说，通常还要再等一段时间才能获得补丁。

图 9：实施安全补丁的延迟时间示例



要以最快的速度获得 CVE 漏洞补丁，请在选择流量管理工具时注意以下两个特性：

- **专门的工程团队** —— 如果有一支工程团队（而非社区志愿者）负责工具开发，那么您就知道有一群人会全力保障该工具的健康状况，并会优先考虑尽快发布补丁。
- **集成的代码库** —— 只要不依赖外部项目（就像我们在 OpenResty 案例中讨论的那样），打补丁就快速多了。

有关 CVE 漏洞补丁的更多信息，请参阅我们的博文《[NGINX Plus 助您快速轻松地缓解安全漏洞](#)》。

通过使用相同的工具支持所有部署，您可以重复使用策略，并降低 SecOps 团队的学习难度

安全用例 2：

抵御 OWASP 十大安全漏洞和 DoS 攻击

解决方案：部署对 Kubernetes 友好且灵活的 WAF 和 DoS 防护解决方案

在为 Kubernetes 应用选择合适的 WAF 和 DoS 防护解决方案时，有两个必不可少的考虑因素（除特性外）：

- **灵活性** —— 某些情况下，在 Kubernetes 内部署工具是最好的选择，这样工具就可以在 Kubernetes 内外灵活运行，而不会受基础架构的限制。通过使用相同的工具支持所有部署，您可以重复使用策略，并降低 SecOps 团队的学习难度。
- **体量** —— 一款优秀的 Kubernetes 工具通常体量也小，资源消耗恰到好处，能够最大程度地减少对吞吐量、每秒请求和延迟的影响。DevOps 团队通常会拒绝使用安全工具，因为他们先入为主地认为这会降低应用速度，而选择一款小体量的高性能工具无疑会增加使用概率。

一款融合了 WAF 和 DoS 防护的工具看似更有效，实则在 CPU 使用率（由于体量大）和灵活性方面不尽人意，并且即使您不需要，也永远都是两个同时部署。最终，这两个问题会抬升 Kubernetes 部署的总体拥有成本，同时为其他基本工具和服务带来预算方面的挑战。



2021 OWASP 十大 Web 应用安全风险

1. 访问控制中断	6. 易受攻击且过时的组件
2. 加密失败	7. 识别和认证失败
3. 注入漏洞	8. 软件和数据完整性问题
4. 不安全的设计	9. 安全日志记录和监控故障
5. 安全配置错误	10. 服务器端请求伪造 (SSRF)

来源：OWASP 基金会

安全工具不仅要选得好，还要用得好。企业通常可以在以下四个位置部署应用服务，保护 Kubernetes 应用：

- **前门**（在 F5 NGINX Plus 或 F5 BIG-IP 等外部负载均衡器上）—— 适用于粗粒度的全局保护，因为它可以跨多个集群应用全局策略
- **边缘**（在 Ingress controller 上，比如 F5 NGINX Ingress Controller）—— 适用于在一个集群上提供标准的细粒度保护
- **service**（在 NGINX Plus 等轻量级负载均衡器上）—— 当集群中的少量服务对独特的策略有着共同需求时，这种方法必不可少
- **pod**（作为应用的一部分）—— 一种高度自定义的方法，适用于策略具有应用针对性的情形

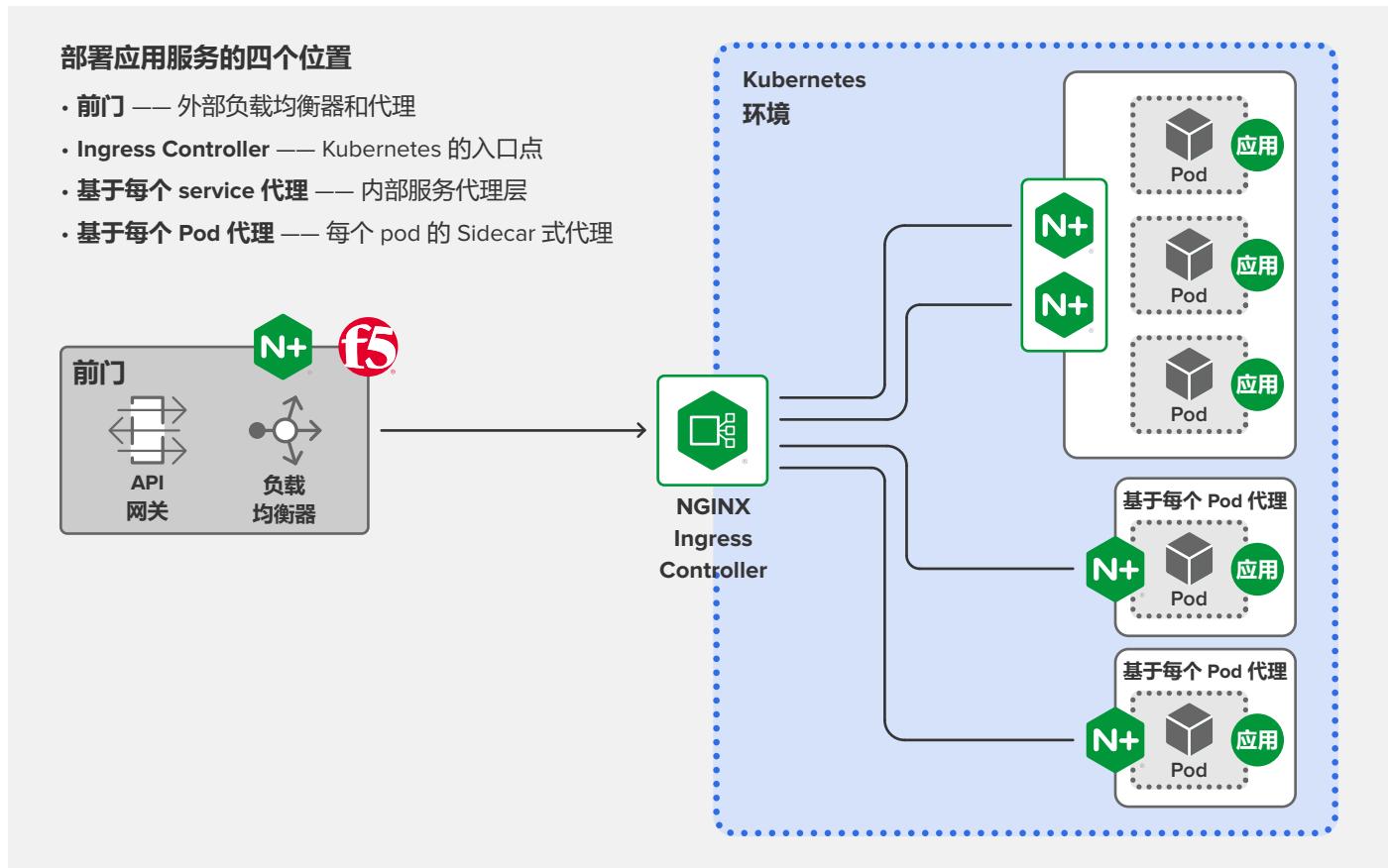


图 10：部署应用服务的四个位置

那么四个选项中，哪个最好呢？嗯...这要看情况！

WAF 的部署位置

WAF 的各个部署选项具有更细微的差别，我们不妨先来看看。

- **前门和边缘** —— 如果企业更倾向于采用“深度防御”安全策略，那么我们建议在外部负载均衡器和 Ingress controller 上部署 WAF，以有效平衡全局保护和自定义的保护。
- **前门或边缘** —— 如果不采用“深度防御”策略，那么也可以只部署在二者其一的位置上，具体取决于所有权。当传统 NetOps 团队负责管理安全性时，他们可能习惯将 WAF 部署在传统代理（外部负载均衡器）上。但是，熟悉 Kubernetes 的 DevSecOps 团队（更喜欢将安全配置放在集群配置附近）可能会选择在 Ingress controller 层面上部署 WAF。
- **基于每个 service 或 pod** —— 如果您的团队对服务或应用有着特定的要求，则可以按列表部署额外的 WAF。但要注意：这些位置的部署成本更高一些。这种方法不仅会拖慢开发速度并增加云预算，而且还会提高与故障排除工作相关的运营成本（例如在排查意外拦截流量的 WAF 时）。

DoS 防护工具的部署位置

DoS 攻击防御起来更直接一些，只需将工具部署在一个位置即可——要么在前门，要么在 Ingress controller。如果您在前门和边缘都部署了 WAF，那么我们建议您在前门 WAF 的前面部署 DoS 防护工具，因为它的覆盖面最广。如此一来，那些非法流量还没到达 WAF 就被清除了，能够让您更有效地利用计算资源。

有关这些方案的更多信息，请参阅我们的博文《[在 Kubernetes 中部署应用服务，第二部分](#)》。

数据用例 3： 从应用中卸载身份验证和授权

解决方案：在入口处集中实施身份验证和授权

身份验证和授权是应用和服务中一个常见的非功能性要求。在不需要频繁更新应用的小规模部署环境中，这种做法虽然会增加一定的复杂性，却也是可以控制和接受的。但是在应用发布速度更快、规模更大的环境中，将身份验证和授权集成到应用中是行不通的。让每个应用维护相对应的访问协议可能会让应用的业务逻辑变得分散甚至被忽视，并导致信息泄露。虽然 SSO 技术通过一组凭证消除了单独设置各个用户名和密码的需要，提高了安全性，但是开发人员仍然必须在应用中写入与 SSO 系统交互的代码。

对此，我们有更好的解决办法：将身份验证和授权卸载到 Ingress controller。

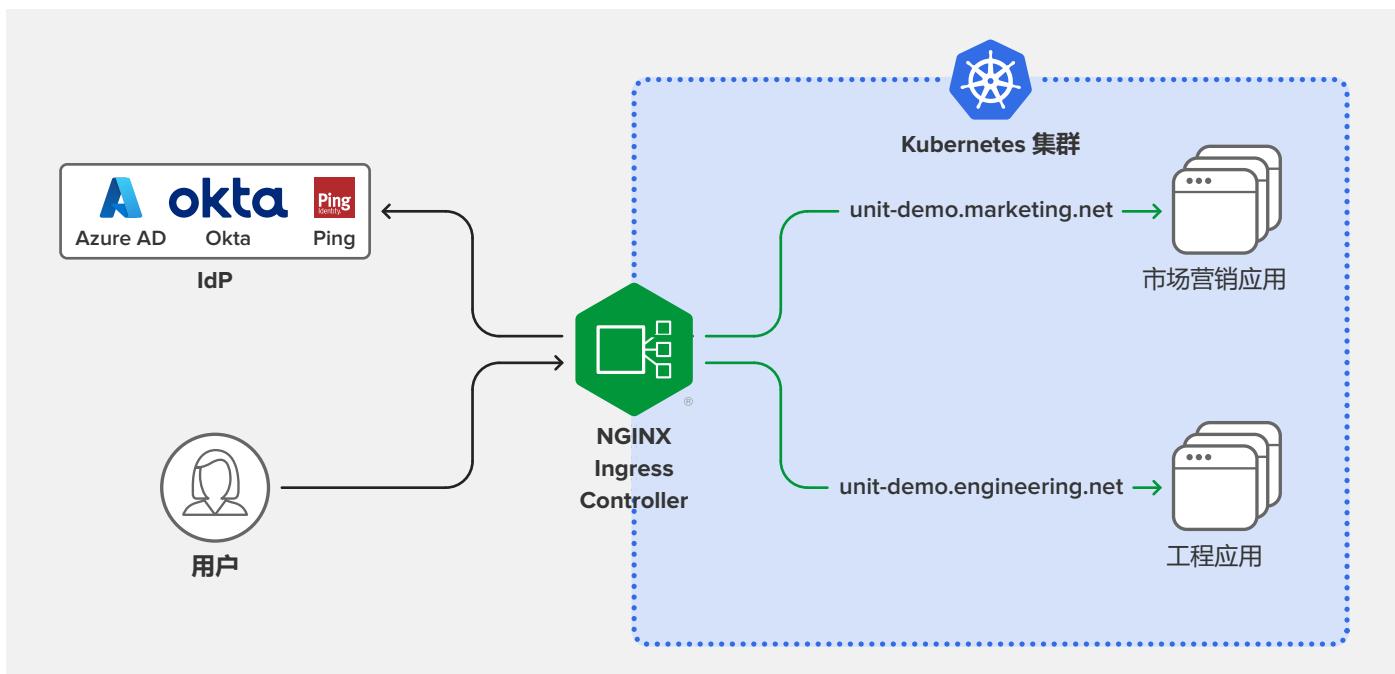


图 11：使用 Ingress controller 进行身份验证和授权

由于 Ingress controller 可以检查进入集群的所有流量并将其路由到相应的服务，在这里集中进行身份验证和授权是一个明智的选择。开发人员不仅可以摆脱构建、维护和复制应用代码逻辑的麻烦，而且还可以使用原生的 Kubernetes API 在入口层快速地用上 SSO 技术。

有关该主题的更多信息，请参阅我们的博文《[借助 Okta 和 NGINX Ingress Controller 实现 Kubernetes OpenID Connect 身份验证](#)》。

有了受控的权限访问，用户无需提交工单就可使用到他们完成工作所需的功能

安全用例 4： 设置有“防护栏”的自助服务

解决方案：实施基于角色的访问控制（RBAC）

Kubernetes 使用 RBAC 来控制不同类型的用户可用的资源和操作。这是一项重要的安全措施，因为它允许管理员或超级用户确定用户或用户组如何与集群中的任何 Kubernetes 对象或特定命名空间进行交互。

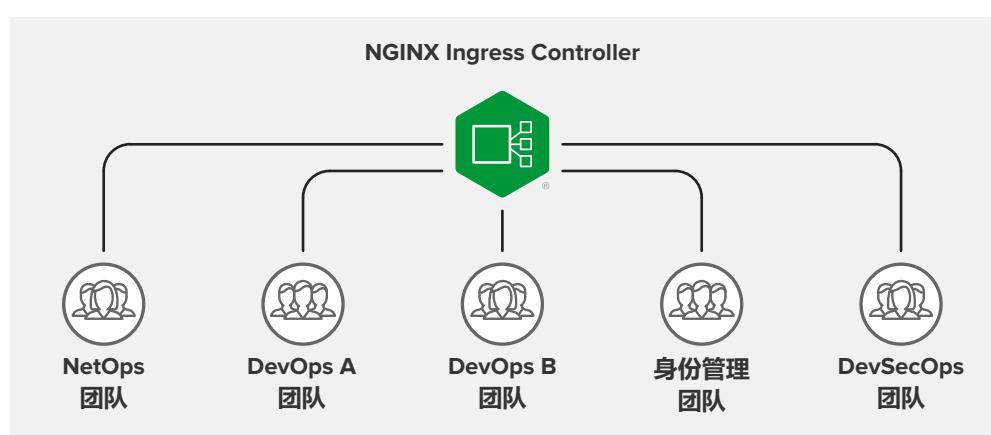
Kubernetes RBAC 在默认情况下呈启用状态，需要注意的是，您的 Kubernetes 流量管理工具也默认启用 RBAC，能够与企业的安全需求保持一致。借助 RBAC，用户无需提交工单，即可利用受控的权限访问完成工作所需的功能。如果不配置 RBAC，用户可能会获得他们不需要或无权使用的权限，而权限滥用就会导致出现漏洞。

配置好 RBAC 的工具可以为多个人员和团队提供服务，Ingress controller 就是一个典型的例子。通过使用 Ingress controller 进行细粒度的访问管理（甚至细微到一个命名空间），您可以使用 RBAC 以多租户模式高效利用资源。

例如，以下多个团队可能会使用 Ingress controller：

- **NetOps 团队** —— 配置应用的外部入口点（如主机名和 TLS 证书），并将流量控制策略委派给不同的团队
- **DevOps A 团队** —— 配置 TCP/UDP 负载均衡和路由策略
- **DevOps B 团队** —— 配置速率限制策略，防止服务请求过多
- **身份管理团队** —— 管理身份验证和授权组件，同时将 mTLS 策略配置为端到端加密策略的一部分
- **DevSecOps 团队** —— 设置 WAF 策略

图 12：多个团队管理 Ingress Controller 的示例



安全用例 5： 实施端到端加密

解决方案：使用流量管理工具

在需要处理敏感信息或个人信息的企业中，端到端加密（E2EE）成为了一个越来越普遍的要求。无论是财务数据还是社交媒体消息，消费者对隐私保护的期望加之 GDPR 和 HIPAA 等法规的实施都拉升了对这类保护的需求。实现 E2EE 的第一步是让您的后端应用在架构上能够接受 SSL/TLS 流量，或者使用工具将 SSL/TLS 管理从应用上卸载下来（这是分离安全功能、性能、密钥管理等工作负载的首选方法）。然后，您再根据环境的复杂性配置流量管理工具。

最常见的方案：使用 Ingress controller 实现 E2EE

当您的应用只有一个端点（简单的应用或者您直接迁移到 Kubernetes 的单体应用）或没有 service 到 service 通信时，您可以在 Kubernetes 内使用 Ingress controller 实现 E2EE。

第一步：确保您的 Ingress controller 只允许使用服务端证书或 mTLS 证书加密的 SSL/TLS 通过，理想情况下出入向流量均如此。

第二步：解决典型的默认设置，让 Ingress controller 在将流量发送给应用之前必须先解密并重新加密流量。实现方法有多种，具体取决于您的 Ingress controller 和要求：

- 如果您的 Ingress controller 支持 SSL/TLS 流量直通，那么它可以根据服务名称指示（SNI）标头路由 SSL/TLS 加密连接（无需解密流量，也不要求访问 SSL/TLS 证书或密钥）。
- 或者，您可以设置 SSL/TLS 终止，即由 Ingress controller 终止流量，然后将其代理到后端或上游——要么以明文的形式，要么通过 mTLS 或服务端 SSL/TLS 对流量重新加密，然后再将其转发到您的 Kubernetes 服务。

在 E2EE 中，service mesh 的角色是确保只有特定的服务能够安全地与其他服务通信

不太常见的方案：使用 Ingress controller 和 service mesh 实现 E2EE

如果您的集群中存在 service 到 service 通信，那么您需要在两个平面上实施 E2EE：使用 Ingress controller 处理出入向流量，并使用 service mesh 处理 service 到 service 流量。在 E2EE 中，service mesh 的角色是确保只有特定的服务能够安全地与其他 service 通信。为 E2EE 设置 service mesh 时，您应该通过两个因素建立零信任环境：service 之间的 mTLS（设置为需要证书）和 service 之间的流量访问控制（指定有权通信的 service）。理想情况下，您还可以在应用之间实施 mTLS（由 service mesh 和出入向流量控制器管理），以便在整个 Kubernetes 集群中实现真正的 E2EE 安全性。

有关加密网络通信数据的更多信息，请参阅我们的博文《NGINX Service Mesh 中的 mTLS 架构》。

不管您处于哪个行业、哪个地区，遵守 FIPS 都是明智之举，因为 FIPS 也是全球加密领域的基准

安全用例 6：

确保客户端使用可信的强密码

解决方案：遵守联邦信息处理标准 (FIPS)

在软件行业中，FIPS 通常是指密码学的专刊 [FIPS 140-2 《加密模块的安全要求》](#)。该文件是美国和加拿大合作的产物，为两国联邦机构都能接受的加密模块（目的是保护敏感信息）制定了测试和认证标准。“等等！”您可能会说，“我不在乎 FIPS，我又不与北美政府机构合作。”不管您处于哪个行业、哪个地区，遵守 FIPS 都是明智之举，因为 FIPS 也是全球加密领域的基准。

遵守 FIPS 并非难事，只是您的操作系统和相关流量管理工具必须也得以 FIPS 模式运行。人们普遍存在一种误解，认为只要在 FIPS 模式下运行操作系统就可以实现 FIPS 合规性。然而即使在 FIPS 模式下运行操作系统，客户端在与 Ingress controller 的通信时可能也不使用强密码。在 FIPS 模式下运行时，您的操作系统和 Ingress controller 可以只使用典型 SSL/TLS 密码套件的一个子集。

您可以按照以下四个步骤为您的 Kubernetes 部署设置 FIPS：

第一步：将您的操作系统配置为 FIPS 模式

第二步：验证操作系统和 OpenSSL 是否处于 FIPS 模式

第三步：安装 Ingress controller

第四步：执行 FIPS 状态检查，验证是否符合 FIPS 140-2 标准

在下面的示例中，操作系统和 OpenSSL（使用 NGINX Ingress Controller 实现）启用 FIPS 模式后，所有出入 NGINX Ingress Controller 的最终用户流量都使用经过验证且支持 FIPS 的加密引擎进行了解密和加密。

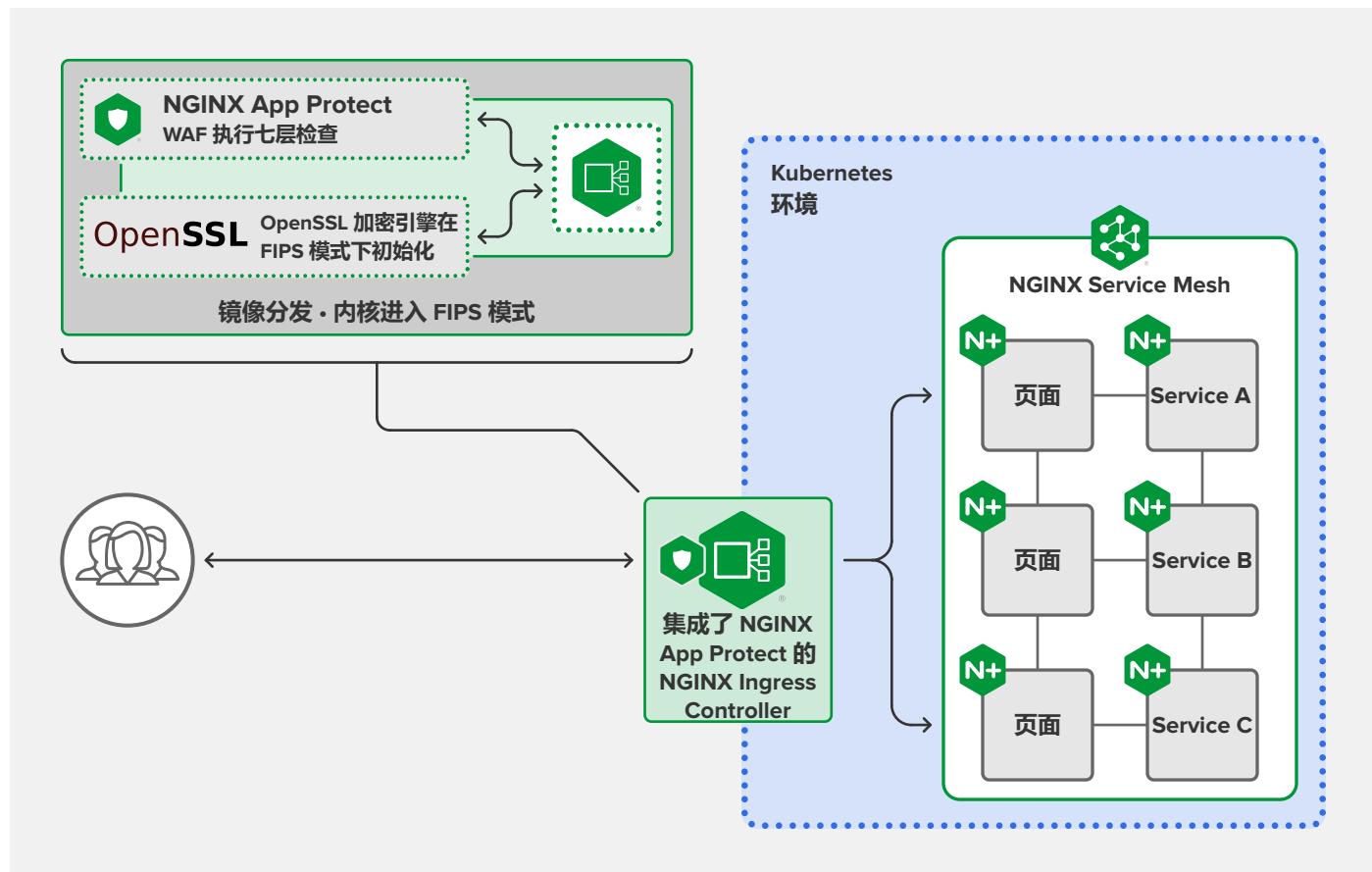


图 13：Kubernetes 环境中的 FIPS

有关 FIPS 的更多信息，请参阅我们的博文 [《通过 NGINX Plus 实现 FIPS 合规性》](#)。

使用 NGINX 提高 Kubernetes 弹性、可视性和安全性

如果您准备实施本电子书中讨论的部分（或全部）方法，那么您需要先确保您的工具具有完成用例所需的合适特性和功能。NGINX 拥有一套生产级 Kubernetes 流量管理工具，能够助您一臂之力：



NGINX Ingress Controller —— 面向 Kubernetes 的 NGINX Plus 版 Ingress controller，支持高级流量控制和整形、监控和可视化、身份验证和单点登录，并且能够充当 API 网关。

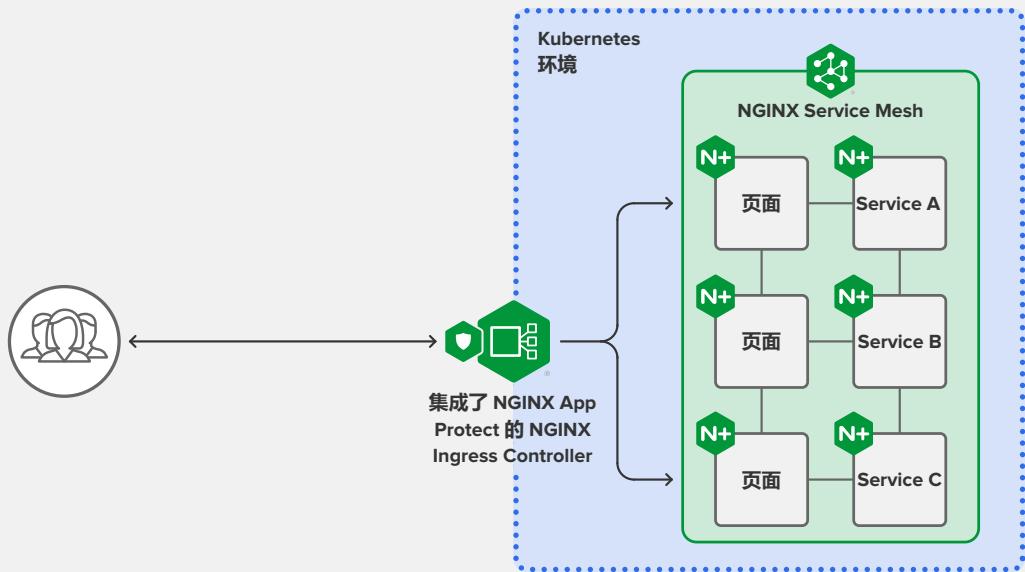


NGINX App Protect —— 建立在 F5 领先市场的安全技术之上，集成了 NGINX Ingress Controller 和 NGINX Plus，能够为现代应用和 API 提供全面的保护。采用模块化方法来实现部署方案的灵活性和资源的最佳利用：

- **NGINX App Protect WAF** —— 一种功能强大的轻量级 WAF，能够抵御 OWASP 十大安全漏洞和其他威胁，并帮助企业遵守 PCI DSS 标准
- **NGINX App Protect DoS** —— DoS 行为检测和防护，横跨各种云环境和各种架构，提供一致的自适应保护



NGINX Service Mesh —— 一种对开发人员友好的、轻量级的、开箱即用的 service mesh，能够将 NGINX Plus 用作企业级 sidecar。



立即申请 NGINX Ingress Controller 30 天免费试用版（含 NGINX App Protect WAF 和 DoS），并[下载](#)始终免费的 NGINX Service Mesh。更多信息请访问 nginx-cn.net。

第二部分

选择最适合您需求的 Kubernetes 流量管理工具

但随着规模的不断扩大，您对
Ingress controller 的选择将变
得愈发重要

5. Ingress controller 选型指南 —— 确定需求

首次使用 Kubernetes 的企业通常不会在 Ingress controller 的选择上花太多心思。他们可能认为 Ingress controller 都大同小异，为了快速启动和运行，直接使用当前 Kubernetes 框架默认的 Ingress controller 是最方便的了。的确，几乎任何 Ingress controller 在测试环境或小批量生产环境中都表现还行。但随着规模的不断扩大，您对 Ingress controller 的选择将变得愈发重要。这是因为 Ingress controller 不仅可以提供将流量从 A 点移动到 B 点的基本功能，而且还可提供高级流量管理、可视化及内置安全性，是您的 Kubernetes 堆栈中最强大的工具之一。

事实上，F5 NGINX 认为 Ingress controller 是任何生产级 Kubernetes 部署的基础。但是，许多开发人员和平台运营团队都没有认识到 Ingress controller 的全部功能，也没有意识到选择一个无法扩展的控制器所带来的后果。如果您选择的 Ingress controller 无法有效地扩展或保护复杂的环境，那么您的 Kubernetes 可能无法从测试环境顺利进入生产环境。在本电子书的第二部分中，我们将向您介绍 Ingress controller 的基础知识，以及如何明智地选择符合现在和未来功能与安全性需求的 Ingress controller。

什么是 Ingress controller?

Ingress controller 是一种专用负载均衡器，用于管理进入以及可能离开 Kubernetes 集群的**四层和七层**流量。明确这一点后，我们来看看 NGINX 使用的术语（与业务所用术语基本相同）：

- **入向流量** —— 进入 Kubernetes 集群的流量
- **出向流量** —— 离开 Kubernetes 集群的流量
- **南北向流量** —— 进入和离开 Kubernetes 集群的流量（也称为“出入向流量”）
- **东西向流量** —— 在 Kubernetes 集群内的 service 之间移动的流量（也称为“service 到 service 流量”）
- **service mesh** —— 一种用于路由和保护 service 到 service 流量的流量管理工具

为什么需要 Ingress controller?

默认情况下，Kubernetes pod
(容器) 中运行的应用无法通过
外部网络来访问

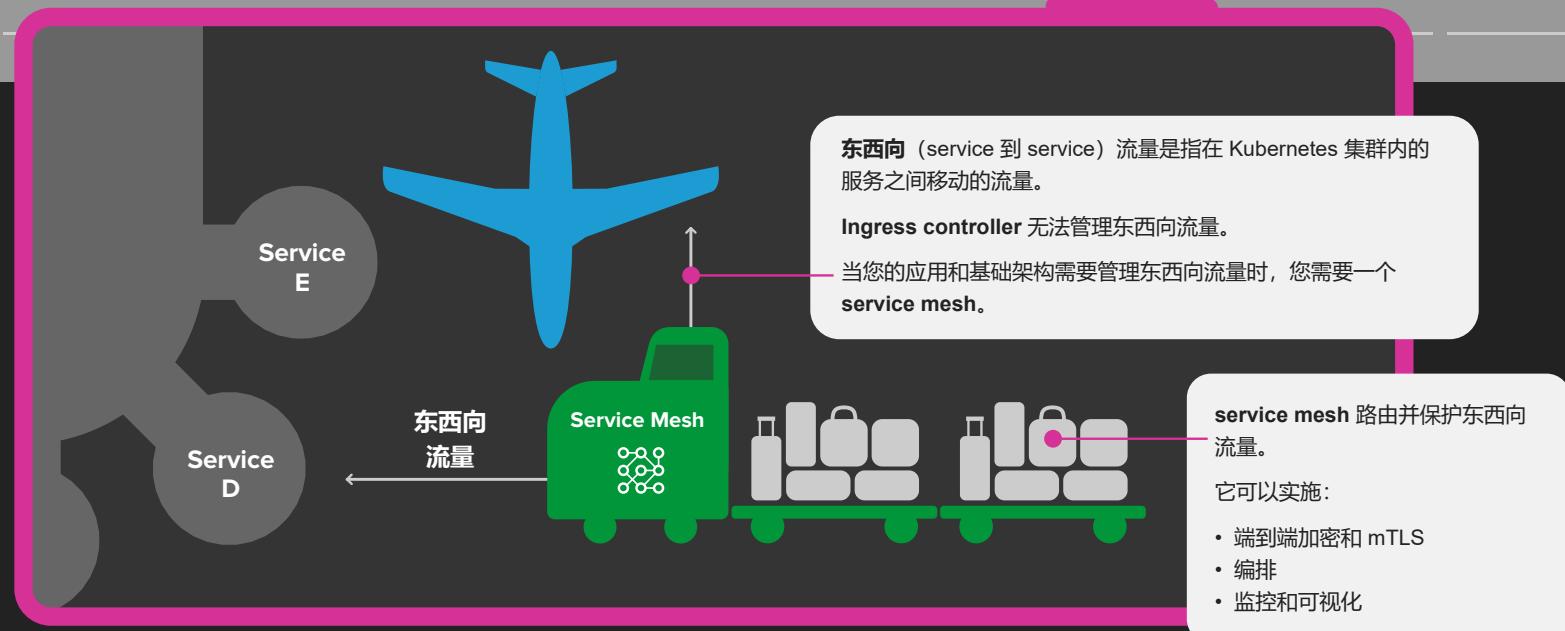
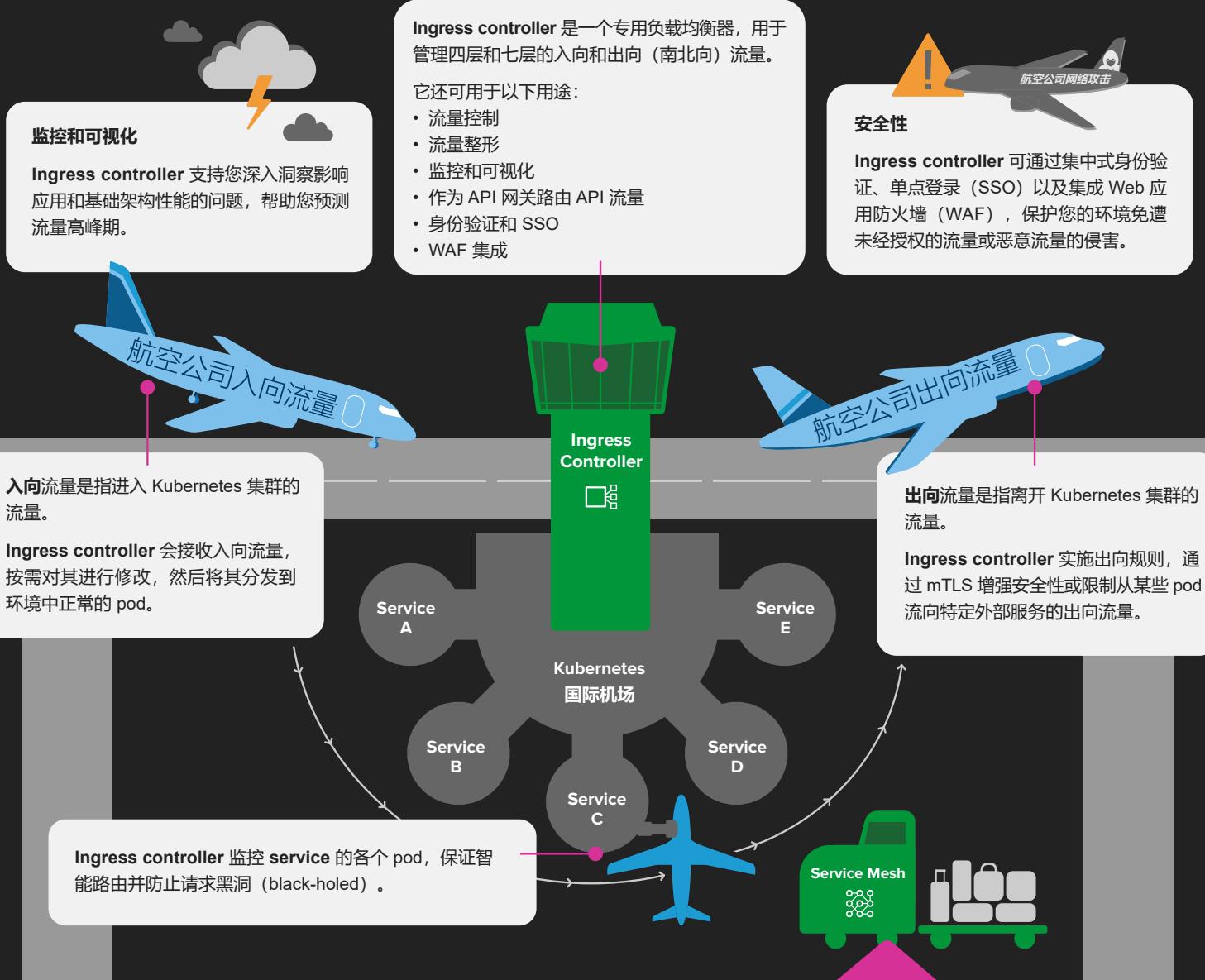
默认情况下，Kubernetes pod（容器）中运行的应用无法通过外部网络来访问，只能通过 Kubernetes 集群内的其他 pod 访问。Kubernetes 有一个用来保持 HTTP 负载均衡的内置配置对象，即为 Ingress。Ingress 定义了 Kubernetes 集群外部实体连接代表一个或多个 Kubernetes service 的 pod 的方式。

当您需要提供对 Kubernetes service 的外部访问时，您可以创建一个 Ingress 资源来定义连接规则，包括 URI 路径、支持服务名称及其他信息。然而，**Ingress 资源**本身不执行任何操作。您必须通过部署和配置 Ingress controller 应用（使用 Kubernetes API）来实施 Ingress 资源中定义的规则。

Ingress controller 有何作用?

- Ingress controller 会接受来自 Kubernetes 环境外部的流量，可能对其进行修改（整形）并将其分发到在 Kubernetes 环境内部运行的 pod。它取代了默认的 kube-proxy **流量分发模式**，为您提供了额外的控制，例如应用交付控制器（ADC）和反向代理在非 Kubernetes 环境中提供的控制。
- 监控 service 的各个 pod，确保智能路由，防止请求“**迷路 (black holed)**”。
- 实施出向规则，通过双向 TLS (mTLS) 增强安全性或限制从某些 pod 流向特定外部服务的出站流量。

Ingress controller 有何作用？



但是大多数 Ingress Controller 可以做的不仅仅是管理流量，我们没必要就盯着这一项

在选择 Ingress controller 时，您可能会先看功能，但这很容易导致最终选择的 Ingress controller 徒有花哨的功能，却不能满足您的业务需求。因此，您必须要了解影响您的团队和应用能否有效使用 Ingress controller 的两个要素：用例（要解决的问题）和资源（投入的成本）。接下来我们将重点讨论这两个主题。

您希望 Ingress controller 解决什么问题？

Ingress controller 的核心用例是流量管理，因此您可能想要 Ingress controller 处理以下一个或多个常见用例：

- 负载均衡 (HTTP2、HTTP/HTTPS、SSL/TLS 终止、TCP/UDP、WebSocket、gRPC)
- 流量控制 (速率限制、断路、主动运行状况检查)
- 流量精分 (调试路由、A/B 测试、灰度部署、蓝绿部署)

但是大多数 Ingress controller 可以做的不仅仅是管理流量，我们没必要就盯着这一项。Ingress controller 能够解决多种问题，不仅可以帮助您减少堆栈的规模和复杂性，而且还可以将非功能需求从应用卸载到 Ingress controller 上。我们来看看四个非传统的 Ingress controller 用例，它们可以帮助您提高 Kubernetes 部署的安全性、敏捷性和可扩展性以及资源的利用率。

监控和可视化

Kubernetes 集群可视性的缺乏是生产环境面临的最大挑战之一，增加了故障排除和恢复的难度。由于 Ingress controller 在 Kubernetes 集群的边缘运行并处理所有流量，它们能够很好地提供一些数据来帮助您解决（甚至避免）两个常见的问题：Kubernetes 集群或平台中的应用运行缓慢和资源耗尽问题。要提高可视化水平，Ingress controller 需要：

- 实时提供指标，以便您可以即时诊断“当前”出现的问题
- 能够将指标导出到常用的可视化工具（例如 Prometheus 和 Grafana），绘制随时间变化的值，以帮助您预测流量激增及其他趋势

API 网关

除非您希望在 Kubernetes 中执行请求-响应操作，否则 Ingress controller 很可能会兼作您的 API 网关。Ingress controller 能够提供核心 API 网关功能，包括 TLS 终止、客户端身份验证、速率限制、细粒度访问控制以及四至七层的请求路由，具体取决于它的功能特性。

身份验证和单点登录

通过将登录凭证的身份验证从 Kubernetes service 卸载到 Ingress controller，您可以解决两个问题：

- 实施采用 OpenID Connect (OIDC) 的单点登录 (SSO)，允许用户使用一组凭证登录多个应用。
- 消除为每个应用构建身份验证功能的需求，让您的开发人员专注于应用的业务逻辑。

Web 应用防火墙集成

我们不是说 Ingress controller 可以充当 Web 应用防火墙 (WAF)，而是它们两个可以集成。尽管 WAF 可以部署在 Kubernetes 外部和内部的许多地方，但对于大多数企业而言，最高效且最有效的部署位置是 Ingress controller 所在的 pod。此用例非常适合安全策略由 SecOps 或 DevSecOps 指导的情况，以及需要细粒度、按 service 或按 URL 方法的案例。这意味着您可以使用 Kubernetes API 来定义策略并将它们与 service 相关联。此外，Ingress controller 基于角色的访问控制 (RBAC) 功能支持 SecOps 按监听器实施策略，并支持 DevOps 用户为每个 Ingress 资源设置策略。

如何为 Ingress controller 分配资源？

每个 Ingress controller 都需要成本，即使那些免费的开源控制器也不例外（您可能听说过这个比喻：“就像养一只免费的小狗一样”）。有些成本是预算中可预测的成本，有些成本则取决于您的团队需要花费多少时间来处理您所选的 Ingress controller 带来的一系列问题（增加的复杂性、缺乏可移植性等）。我们来看看在确定 Ingress controller 预算时需要考虑的主要成本（包括时间成本和资本成本）。



时间成本预算

确定人员预算可能比确定固定的项目成本更难，尤其是当您尝试为不熟悉的领域分配项目资源。您需要思考几个问题：

- 谁负责配置和管理 Ingress controller？这是他们的专职工作（例如作为平台运营团队的成员）还是兼任的职责（作为开发人员）？
- 您的员工是否有时间接受正规培训？您是否要求工具必须简单易学，以便快速轻松地用于工作？
- 您是否准备好让员工在需要新功能时进行修补，或者在出现问题时花大量时间进行解决？或者您是否需要他们来提供其他业务价值？

关于 Kubernetes 工具所有权的说明

我们观察到业界有一种趋势，即在 NetOps 团队的领域内整合工具和所有权。虽然这有助于简化堆栈和提高安全性，但我们提倡根据团队目标考虑选择的工具。NetOps 团队专注于更广泛的平台，因此让他们管理边界工具（如外部负载均衡器）合情合理，而 DevOps 团队最关心靠近应用代码部署的服务，并且需要比 NetOps 工具更高的敏捷性和更细粒度的控制。DevOps 选择 Kubernetes 工具（包括 Ingress Controller）最有可能取得成功。这并不是说您必须完全放手让开发人员自由选择工具！Kubernetes 中一些严苛的工具标准化依然是最佳实践。

资本成本预算

首次使用 Kubernetes 的企业通常不会在工具或支持上投入太多预算。如果您拥有足够的人力资源来支持需要更多“人工”的 Ingress controller，那么初期可以不考虑资本成本预算。但随着 Kubernetes 投资的增加，您可能会发现工具的功能不够用，或者想要将团队划拨到其他优先事项。这时，大家开始购买更易于使用、更稳定且具有企业级功能和支持的工具来进行扩展。

请注意，到了花钱这一步，许可模式就变的很重要。Ingress controller 采用的是不同于 Kubernetes 的传统定价模式，通常“按实例”或“按 Ingress 代理”定价。有时也可考虑“按集群”付费，这意味着您将根据应用租赁而不是“代理数量”购买 Ingress controller 许可。

以下是我们针对不同场景的建议：

- **Kubernetes 新手？选择按集群许可。**

如果您缺乏 Kubernetes 使用经验，便很难准确预测您需要的 Ingress controller 实例的数量。如果非要选“按实例”的话，那么您可能会低估（从而难以实现您的目标）或高估实例数量，浪费了本可以在 Kubernetes 项目的其他部分发挥作用的资金。为“小集群”争取一个相对较低的固定价格将会增加您成功的机会。

- **扩展 Kubernetes？选择按集群许可。**

您可能很难预测 Kubernetes 资源（突发和崩溃）的调整规模和频率。在按实例模式下，您的团队必须要设定任意阈值，确保不超出许可上限。而在按集群许可中，您不需要跟踪各个 Ingress 容器，并且流量突发支持也可能无需额外付费，具体取决于厂商（例如 NGINX）。

- **高级部署还是静态部署？选择按实例许可。**

如果您对 Kubernetes 足够精通，准确地知道将如何使用 Ingress controller，并计划在每个集群中只使用少数几个 Ingress 代理（通常少于三个），而且不需要随工具附带的任何服务，那么按实例购买可能是一个不错的选择。

下一步行动：风险承受能力和未来需求洞察

现在您已经了解了自己的需求，接下来是判断整个团队对 Ingress controller 的风险承受能力，并弄清楚随着 Kubernetes 部署的增长，您需要如何扩展 Ingress controller。我们将在[下一章](#)中讨论这些主题。

6. Kubernetes Ingress controller 选型指南 —— 评估风险和技术前瞻性

欢迎阅读 Ingress controller 选型指南的第二部分。您现在可能已经确定了要求，但还不是测试产品的时候！在本章中，我们将向大家说明不符合您需求的 Ingress controller 为什么会降低发布速度，甚至对您的客户造成损失。同任何工具一样，Ingress controller 也存在风险并影响未来的可扩展性。下面我们来看看如何避免做出弊大于利的选择。

Ingress controller 风险

现在的消费者都很“薄情”，只要感到数字体验不好，那么无论功能多么花哨，他们也会一去不回头

在选择 Kubernetes 流量管理工具时，您应考虑三个特定风险：复杂性、延迟和安全性。这些风险盘根错节，往往牵一发而动全身。Ingress controller 可能会引发其中任何一个问题，能否容忍就要看企业自己了。现在的消费者都很“薄情”，只要感到数字体验不好，那么无论功能多么花哨，他们也会一去不回头。

Ingress controller 风险

新工具会带来新风险，可能让您得不偿失。不符合需求的 Ingress controller 可能会带来以下三大风险。

01 复杂性 Ingress controller 是否与微服务架构的目标背道而驰？ 复杂性是使用和部署容器面临的主要挑战之一。 ¹ Ingress controller 选择不当可能会加剧复杂性，从而不仅限制您横向扩展部署的能力，而且还会对应用性能产生负面影响。	02 延迟 Ingress controller 是否会减缓应用速度？ 企业和机构采用 Kubernetes 是为了能够更快地部署新应用。 ² 但是一款因错误、超时和重载而增加延迟的 Ingress controller 可能反而会降低您的应用速度。	03 安全性 Ingress controller 是否为黑客提供了可乘之机？ 超过一半的企业和机构出于容器或 Kubernetes 的安全考量，推迟或放慢了将应用部署到生产环境的速度。 ³ 注意不要选择 CVE 补丁发布缓慢的 Ingress controller，也不要过于依赖公共论坛的支持。
--	---	---

1. 2020 年云原生计算基金会 (CNCF) 调查
2. 2021 年 Kubernetes 采用率调查
3. 红帽《Kubernetes 安全状态报告》

我们通常可以通过大小来看复杂性：Ingress Controller 占用空间越大就越复杂

复杂性

Ingress controller 是否与微服务架构的目标背道而驰？

能够满足微服务架构目标（设计简单、使用便捷）的工具才是一款好的 Kubernetes 工具。企业可以开发一种功能丰富且遵循这些原则的 Ingress controller，但有时却事与愿违。有些设计人员要么添加过多的功能，要么使用复杂脚本来扩展底层引擎的非原生功能，导致 Ingress controller 膨胀不堪。

为什么必须要降低复杂性？在 Kubernetes 中，过于复杂的工具不仅会影响应用性能，而且还会限制您横向扩展部署的能力。我们通常可以通过大小来看复杂性：Ingress controller 占用空间越大就越复杂。

延迟

Ingress controller 是否会减缓应用速度？

Ingress controller 可能会因资源使用、错误和超时而增加延迟。我们需要了解静态和动态部署中增加的延迟，并根据您的内部要求消除延迟过长的选项。有关重新配置对应用速度的影响的更多详细信息，请阅读附录中的《[性能测试三种不同的 NGINX Ingress Controller 选项](#)》。

安全性

Ingress controller 是否为黑客提供了可乘之机？

在当今的互联网中，通用漏洞披露（CVE）信息库中的漏洞不断增加，Ingress controller 提供商提供 CVE 补丁的速度决定了 Ingress controller 的安全性能。根据企业的风险承受能力，您可能想要弃用补丁发布时间超过几天（或最多几周）的解决方案。

除了 CVE 中的漏洞之外，一些 Ingress controller 还会带来另一个潜在漏洞。假设您为一家在线零售商工作，您的开源 Ingress controller 的配置出现了问题，亟需别人的帮助。由于没有购买商业支持服务，您将问题发布到了 Stack Overflow 等论坛。有人向您伸出了援手，他想要看一下是否是 Ingress controller 及其他 Kubernetes 组件的配置和日志文件出现了问题。迫于尽快解决问题的压力，您共享了文件。

企业选择购买支持服务的主要原因之一：它能保证机密性

这个“好心人”帮您解决了问题，但六个月后公司遭到入侵了，客户记录里的信用卡号码被盗了。这是怎么回事？原来是您共享的文件中包含一些信息，被居心叵测的攻击者用来入侵应用了。这也恰好反映了企业选择购买支持服务的主要原因之一：它能保证机密性。

关于基于 OpenResty 的 Ingress controller 的说明

OpenResty 是一个基于 NGINX 开源软件构建的 Web 平台，它包含了 LuaJIT、Lua 脚本和第三方 NGINX 模块，可扩展 NGINX 开源软件的功能。市面上有几种 Ingress controller 是基于 OpenResty 构建的，我们认为与我们基于 NGINX 开源软件和 F5 NGINX Plus 构建的 Ingress controller 相比，它们可能会额外带来两个风险：

- **超时** —— 如上文所述，OpenResty 使用 Lua 脚本来实现高级功能，例如基于商用 NGINX Plus 的 Ingress controller 中的功能。其中一项功能是动态重新配置，这消除了 NGINX 开源软件中原本会降低可用性的一项要求，即当 service 端点发生变化时必须重新加载 NGINX 配置。如要使用 OpenResty 完成动态重新配置，Lua handler 会选择将请求路由到哪个上行 service，从而消除了重新加载 NGINX 配置的需要。

Lua 必须持续检查后端的变更情况，十分消耗资源

但是，Lua 必须持续检查后端的变更情况，十分消耗资源。处理入站请求的时间延长，导致部分请求被搁置，从而增加了超时的可能性。随着扩展到更多用户和服务，每秒入站请求的数量与 Lua 可以处理的请求数量之间的差距呈指数级扩大，最终导致延迟、复杂性和成本的增加。

如欲了解 Lua 具体的延迟情况，请阅读附录中的《[性能测试三种不同的 NGINX Ingress Controller 选项](#)》。

- **CVE 补丁延迟** —— 与 NGINX Ingress Controller 相比，CVE 补丁推送到基于 OpenResty 等工具（这些工具又基于 NGINX 开源软件）的 Ingress controller 中的时间更长，这一点无法避免。我们曾在博文《[NGINX Plus 助您快速轻松地规避安全漏洞](#)》中详细说过，当在 NGINX 中发现 CVE 漏洞时，我们作为厂商通常会在 CVE 公开披露之前得到通知。因此，我们能够在 CVE 发布后立即发布 NGINX 开源软件和 NGINX Plus 的补丁。

在此之前，使用 NGINX 开源软件的技术群体可能不会知晓这个 CVE 漏洞的存在。根据我们的经验，OpenResty 补丁的发布时间比我们晚得多（[最近一次晚了四个月](#)）。基于 OpenResty 的 Ingress controller 的补丁发布时间不可避免地需要更久，这为攻击者提供了充分的可乘之机。

面向未来的 Ingress controller

即使您只是刚开始使用 Kubernetes，您可能也希望有朝一日将其投入生产。随着时间的推移，您在基础架构、安全性、支持和多租户等四个主要领域的需求可能会有所增长。

面向未来的 Ingress controller

您在以下四大领域的需求
可能会随时间增长。

01

基础架构

**您是否会在混合或多云环境中使
用 Kubernetes？**

很少有企业会一直完全使用一种环境。
您不妨从一开始就选择一种不受基础
架构限制的 Ingress Controller，以便
在所有环境中使用相同的工具。

02

安全性

**您将如何从内部保护
Kubernetes？**

Kubernetes 应用只有在靠近安全措施
(包括身份验证和授权) 时才能获得
最好的保护。在入口处集中实施安全
措施 (身份验证、授权、DoS 防护、
Web 应用防火墙) 有助于降低成本和
提高效率。

03

支持

您“自力更生”的能力有多强？

当您运行小型部署环境时，自己寻找变通
方法和等待社区支持还尚可，但一旦转入
生产环境，这种方法就行不通了。您可以
选择一种允许您在未来添加支持服务的
Ingress Controller，或者选择一种价格经
济并且可以随着扩展而升级的支持等级。

04

多租户

**多个团队和应用如何安全可靠地共享
容器环境？**

随着服务和团队规模及复杂性的增加，您
可能需要转向使用多租户来保证最高效率。
一些 Ingress controller 可以通过多种功能
和概念 (支持设置 RBAC 的多个入口、类
别、命名空间和限定资源) 帮助您划分这
些集群。



基础架构

您是否会在混合或多云环境中使用 Kubernetes?

很少有企业会一直完全使用一种环境。反而是本地和云端的混合环境更为常见。这种环境一般包含私有云、公有云、混合云和多云。（如欲深入了解这些环境之间的差异，请阅读《[多云和混合云的区别是什么？](#)》）

我们在[第 5 章](#)中提到过，企业一般会选择每种环境默认的工具，但默认的 Ingress controller 本身存在许多问题。我们将在[第 7 章](#)讨论这样做的所有弊端，但与未来前瞻性最有关系的问题是厂商锁定，即您无法在所有环境中使用同一个由特定云厂商提供的 Ingress controller。使用云平台默认的工具将会影响您的扩展能力，因为您只有两个毫无吸引力的替代方案：

1. 尝试让现有的云满足所有需求
2. 在每个新环境中重写 Ingress controller 的所有配置，不仅包括负载均衡，而且还包括安全性！

从一开始就选择一种不受基础架构限制的 Ingress controller

第一种方案出于商业原因通常不可行，第二种同样也很棘手，因为它会导致工具无序蔓延、引发新的安全漏洞，并需要员工学习大量新知识。第三种也是最有效的替代方案是从一开始就选择一种独立于基础架构的 Ingress controller，以便在所有环境中使用相同的工具。

涉及到基础架构，我们还要考虑认证这个因素。我们以红帽 OpenShift 容器平台（OCP）为例。如果您是 OCP 用户，那么您可能已经知道 Red Hat Marketplace 为 OCP 提供认证操作符，包括 [NGINX Ingress Operator](#)。红帽[认证标准](#)意味着该工具适用于您的部署环境，甚至提供红帽与厂商联合支持，从而让您高枕无忧。出于安全性和稳定性的原因，许多企业都要求使用经过认证的工具，因此即使您目前只处于测试阶段，牢记公司的生产环境要求也是有好处的。

安全性

您将如何从内部保护 Kubernetes?

Kubernetes 应用只有在靠近安全措施时才能获得最好的保护

仅靠边界安全就足以保护应用与客户安全的日子已经成为过去式。现在，只要在靠近 Kubernetes 应用的地方实施安全防护（包括身份验证和授权），才能为其提供最强大的保护。随着越来越多的企业在 Kubernetes 中强制采用端到端加密方法和零信任网络模型，service mesh 可能将会成为您未来计划的一部分。

[符合您的安全策略的 Ingress controller 可防止您在应用开发过程中遇到重大问题。](#)

所有这些又与 Ingress controller 有什么关系呢？关系很大！从成本和效率的角度来看，在 Ingress 中集中管理安全防护措施（身份验证、授权、DoS 防护、Web 应用防火墙）具有重要意义。虽然大多数 service mesh 能够与大多数 Ingress controller 集成，但它们的集成方式很重要。符合您的安全策略的 Ingress controller 可防止您在应用开发过程中遇到重大问题。

有关云原生应用交付风险的更多信息，请阅读《[在不影响速度的同时保护云原生应用](#)》；有关决定安全工具最佳位置的因素的更多信息，阅读《[在 Kubernetes 中部署应用服务第二部分](#)》。

支持

您“自力更生”的能力有多强？

对于首次使用 Kubernetes 的团队来说，无论是社区支持还是公司支持通常都不是最重要的。如果您的团队时间充足，能够制定自己的解决方案和应变方法，这没有问题，但进入生产环境后就行不通了。即使您现在不需要支持，您也可以选择一种允许您在未来添加支持的 Ingress controller，或者选择一种价格经济并且可以随着您的扩展而升级的支持等级。

多租户

多个团队和应用如何安全可靠地共享容器环境？

“企业发展之初，都是只有一个团队和一个应用，难道不是吗？”随着这个团队成功开发出一个 Kubernetes 应用，企业开始在 Kubernetes 上运行更多服务。当然，更多的服务意味着更多的团队以及更高的复杂性。

为了最大限度地提高效率，企业采用多租户和 Kubernetes 模型，以支持业务流程所需的访问和隔离，同时提供运营商所需的安全性和控制力。一些 Ingress controller 可以通过多种功能和概念（支持设置基于角色的访问控制（RBAC）的多个入口、类别、命名空间和限定资源）帮助您划分这些集群。

下一步行动：锁定选择范围

既然您已确定了要求、风险承受能力和未来防范能力，那么这说明您掌握了足够的信息，可以开始缩小 Ingress controller 的选择范围了。您可以按类别划分范围，这有助于快速完成选择，在[第 7 章](#)中，我们将探讨三种不同类别的 Ingress controller 以及它们各自的优缺点。

7. Ingress controller 选型指南 —— 开源、默认和商用版本能力对比

恭喜您！读完第 5 章和第 6 章后，您差不多就可以开始选择 Ingress controller 了。我们来回顾一下往期的内容：

- [第 5 章](#)讨论了如何确定您在性能、预算、用例、架构和所有权方面的要求。
- [第 6 章](#)讨论了 Ingress controller 选择不当可能会引发的风险，介绍了让您的选择具有前瞻性的几个关键领域。

Ingress controller 分为三类：开源、默认和商用。它们各有各的用例，您需要在选择之前先明确自己的短期和长期需求。本章将会介绍它们各自的优缺点。

开源 Ingress controller

虽然有些开源 Ingress controller 有专门的工程团队，但多数都是由用户和志愿者开发人员社区来维护。目前最流行的两个开源 Ingress controller 都是基于 NGINX 构建的——一个由 Kubernetes 社区维护，一个由核心 NGINX 工程团队领导和开源。有关基于 NGINX 的 Ingress controller 的进一步比较，请参阅[第 8 章](#)。

开源的 Ingress controller	
虽然有些开源 Ingress controller 有专门的工程团队，但多数都是由用户和志愿者开发人员社区来维护。	
优点	缺点
<p>选择开源 Ingress controller 的主要原因</p> <ul style="list-style-type: none">▲ 无需金钱投资（免费！）▲ 社区驱动▲ 功能发布速度快 <p>适用场景：您刚刚开始使用 Kubernetes，处于测试或小批量生产阶段。</p>	<p>不选择开源 Ingress controller 的主要原因</p> <ul style="list-style-type: none">▼ 需要花费更多时间▼ 存在不稳定、不安全和不可靠的风险▼ 几乎没有支持 <p>您可以使用“默认”或“商用”版 Ingress controller 来消除这些缺点。</p>

能依赖的除了自己以外就是文档而已

开源 Ingress controller：优点和缺点

- **优点：**

- **免费使用，社区驱动** —— 许多企业和机构选择开源项目不仅是因为其无与伦比的价格（免费！），而且还在乎他们更喜欢社区开发的技术。
- **功能更新换代快** —— 这类 Ingress controller 更有可能走在功能创新的最前沿。

- **缺点**（开源项目普遍存在的问题）：

- **成本（时间）** —— 它们缺乏“开箱即用”的工具，因而无法轻松设置和扩展，最终您还是要花时间根据特定需求进行自定义和变通。
- **风险** —— 可能缺乏稳定性、安全性和可靠性（原因在于过于重视功能发布速度，再加上更多的是由社区自愿贡献）。CVE（通用漏洞披露）漏洞的补丁可能永远不会出现，或者可能在 CVE 公开披露数月后才出现，漫长的时间窗口给黑客带来了充分的可乘之机。
- **几乎没有支持** —— 出现问题时多数需要自己解决，最多查查文档资料。如果遇到自己无法解决的问题，则很难甚至根本无法获得帮助——唯一的选择就是在社区论坛上晒出自己的问题，寄望于社区的其他成员能够（a）好心回应和（b）知道解决办法。
- **总结：**由于有文档、上手快且免费，首次试水 Kubernetes 的企业和机构通常会选择开源 Ingress controller。这在起步阶段、测试环境或小批量生产环境中是一种不错的选择。

默认的 Ingress controller

虽然许多默认的 Ingress controller 也基于开源技术构建，但它们由提供完整 Kubernetes 平台（并且通常提供管理支持）的公司开发和维护，因此我们单独拿出来介绍。公有云 Ingress controller、Rancher 和红帽 Hat OpenShift 路由器都属于这一类。

默认的 Ingress controller	
默认的 Ingress controller 由提供完整 Kubernetes 平台的公司开发和维护（并且通常提供管理支持）。	
优点	缺点
选择默认 Ingress controller 的主要原因	不选择默认 Ingress controller 的主要原因
<ul style="list-style-type: none">▲ 免费或低成本▲ 可靠▲ 有支持服务	<ul style="list-style-type: none">▼ 基础架构锁定▼ 只有基础功能▼ 扩展时间或金钱成本不可预测
适用场景：您刚刚开始使用 Kubernetes，处于测试或小批量生产阶段。	您可以使用“开源”或“商用”版 Ingress controller 来消除这些缺点。

默认 Ingress controller：优点和缺点

• 优点：

- **免费或低成本** —— 低廉的价格是这类产品非常有吸引力的一个宣传点。它们已经集成到平台中，对新手来说是当之无愧的省时神器。
- **可靠且有支持** —— 它们由专门的工程团队来维护，比社区维护的 Ingress controller 更可靠一些。它们通常自带或额外销售商业支持。

• 缺点：

- **基础架构锁定** —— 默认的 Ingress controller 受基础架构的限制，您无法在云间迁移它们或它们的配置。这意味着每个部署环境都需要不同的 Ingress controller，从而导致工具无序蔓延，对团队知识储备的要求加大，并使 Ingress controller 更加难以保护。
- **基本功能** —— 它们通常缺乏大规模部署所需的高级流量管理和安全功能。

您无法在不同的云平台之间迁移默认的 Ingress Controller 以及你对它们所做的配置

- **不可预测的成本（时间和金钱）** —— 虽然起步成本很低或直接没有，但可能会随着应用的增长而急剧攀升且不可预测。您可能需要花费时间将 Ingress controller（功能匮乏）无法实现的功能构建到应用中 —— 当然了，每次更新都还伴随着回归测试。有些默认工具还有一个缺点：随着应用变得越来越流行，起初看似不起眼的吞吐量费用将会让您的云账单大幅增加。
- **总结：**对于刚接触 Kubernetes 并使用托管平台（例如亚马逊 Elastic Kubernetes Service (EKS)、Google Kubernetes Engine (GKE)、微软 Azure Kubernetes Service (AKS)、Rancher 和红帽 OpenShift Container Platform 的团队来说，默认的 Ingress controller 是一种比较受欢迎的选择。随着应用的成熟和团队的壮大，企业和机构通常会选择在堆栈中添加企业级 Ingress controller，而不是替换默认工具。

商用 Ingress controller

商用 Ingress controller 是许可产品，旨在支持大型生产部署。基于 NGINX Plus 的 [F5 NGINX Ingress Controller](#) 就是一个典型的例子，我们将在[第 8 章](#)详细讨论。

商用 Ingress controller	
商用 Ingress controller 是旨在支持大型生产部署的许可产品。	
优点	缺点
选择商用 Ingress controller 的主要原因	不选择商用 Ingress controller 的主要原因
<ul style="list-style-type: none">▲ 功能丰富▲ 可扩展且省时▲ 可靠且有支持服务	<ul style="list-style-type: none">▼ 功能发布缓慢▼ 需要金钱投资
适用场景：您刚刚开始使用 Kubernetes，处于测试或小批量生产阶段。	您可以使用“开源”或“默认”版 Ingress controller 来消除这些缺点。

商用 Ingress controller：优点和缺点

- **优点：**

- **功能丰富** —— 商用 Ingress controller 包含丰富、强大的功能，能够为大型部署环境提供高级流量管理和可扩展性。它们可能还会集成其他生产级产品，例如 WAF 或 service mesh。
- **可扩展** —— 企业和机构发现这类产品可以节省时间，因为它们往往有更多“开箱即用”的功能，不需要自定义和变通。它们可以轻松添加到自动化流水线中，允许基础架构按需扩展。
- **可靠且有支持** —— 商用产品的主要优势之一是稳定，这意味着每个版本都经过了广泛测试，并定期进行软件更新，按需实施安全补丁。商用产品一般会提供各个级别的全面商业支持，如果您遇到严重问题，通常几分钟或几小时内就能以保密的方式获得帮助。

- **缺点：**

- **开发略慢** —— 由于稳定性是商用 Ingress controller 的重要因素，它们的功能发布速度可能比开源竞品稍微慢一些。
- **成本（金钱）** —— 商用产品有一个不得不面对的现实：费钱。如果开发周期长，现金回流慢，那么除非这种情况得以改变，否则成本就是交易路上的绊脚石。

一旦企业呈现高度复杂性，商用 Ingress controller 就变得有意义
• **总结：**随着企业和机构的扩展，根据团队和应用的复杂性来选择 Ingress controller 变得愈发重要。一旦企业呈现高度复杂性，商用 Ingress controller 就变得有意义，因为它可以降低管理复杂性并加快新产品功能的上市速度。

下一步行动：评估您的选择

到了这个阶段，您就可以划掉一些不符合需求的选项，只锁定一部分 Ingress controller 了。

在研究 Ingress controller 的过程中，您可能会注意到许多选择都是基于 NGINX 的。阅读第 8 章，了解基于 NGINX 的 Ingress controller 选择。

8. Ingress controller 选型指南 —— NGINX Ingress controller 选项

根据 2020 年云原生计算基金会 (CNCF) 的调查，NGINX 是 Kubernetes [Ingress controller](#) 中最常用的数据平面 —— 但您知道吗？我们不止有一个 “NGINX Ingress Controller”。

本章内容在 2018 年还有一个版本，标题是《NGINX Kubernetes Ingress Controller 版本辨析》。文章的灵感来自与一位社区成员对两个基于 NGINX 的热门 Ingress controller 的探讨。



Matt Oswalt
@Mierdin

回复 @pandom_ @nginx @LindsayofSF

...我大概知道这两种不同的 Ingress controller,
因为我见过它们的容器镜像在不同的位置。
但我应该没听 NGINX 的员工介绍过它们。

2018 年 11 月 16 日，下午 4:48

我们不难理解为什么人们会有困惑（现在仍然如此）。两个 Ingress controller 都有以下特点：

- 名称是 “NGINX Ingress Controller”
- 开源
- 托管在 GitHub 上，具有非常相似的代码库名称
- 均为在大约同一时间启动的项目的成果

当然，最大的共同点是它们实现了相同的功能。

NGINX 版与 Kubernetes 社区版 Ingress controller

为了清晰起见，我们这样区分两个版本：

- **社区版：**社区版 Ingress controller 以 NGINX 开源版为基础（文档参见 [Kubernetes.io](#)），可在 GitHub 的 [kubernetes/ingress-nginx](#) 中找到。它由 Kubernetes 社区维护，并且 [F5 NGINX 承诺](#)帮助管理该项目。

- **NGINX 版本**: NGINX Ingress Controller 由 NGINX 开发和维护 (文档见 [docs.nginx.com](#)) , 可在 GitHub 的 [nginxinc/kubernetes-ingress](#) 中找到。它有两个版本:
 - 基于 NGINX 开源版 (开放的开源版本)
 - 基于 F5 NGINX Plus (商用版本)

市面上还有一些其他基于 NGINX 的 Ingress controller, 比如 Kong, 不过好在它们的名字很好区分。如果您不确定使用的是哪个 NGINX Ingress Controller, 可以查看正在运行的 Ingress controller 的容器镜像, 然后将 Docker 镜像名称与上面列出的代码库进行比较。

NGINX Ingress Controller 的目标和优先级

NGINX Ingress Controller 和社区版 Ingress controller (以及基于 NGINX 开源版的其他 Ingress controller) 之间的主要区别在于它们的开发和部署模式, 而这些模式又基于不同的目标和优先级。

所有 NGINX 项目和产品的首要任务都是提供一个具有长期稳定性和一致性的快速、轻量级的工具

- **发展理念** —— 所有 NGINX 项目和产品的首要任务都是提供一个具有长期稳定性和一致性的快速、轻量级的工具。我们努力避免版本之间有任何机理上变化, 尤其是那些破坏向后兼容性的机理。我们保证您在升级时不会得到任何意外 “惊喜” 。我们也注重可选择性, 因此我们的所有解决方案都可以部署在任何平台上, 包括裸机、容器、虚拟机以及公有云、私有云和混合云。
- **集成式代码库** —— NGINX Ingress Controller 使用 100% 纯 NGINX 开源版或 NGINX Plus 实例实施负载均衡, 使用仅包含原生 NGINX 功能的最佳实践配置。它不依赖于未接受 NGINX 互操作性测试的任何第三方模块或 Lua 代码。我们的 Ingress controller 不是利用各种第三方代码库拼凑而成, 我们自主开发和维护负载均衡器 (NGINX 和 NGINX Plus) 及 Ingress controller 软件 (一种 Go 应用) 。我们是 NGINX Ingress Controller 所有组件的唯一权威所有者。
- **高级流量管理** —— 标准 Kubernetes Ingress 资源的一大局限性在于, 您必须使用 Annotation、ConfigMaps 和客户模板等辅助特性来定制它的高级功能。[NGINX Ingress Resource](#) 采用了一种原生的、类型安全的、缩进式的配置风格, 简化了 Ingress 负载均衡功能的实施, 包括 TCP/UDP、断路、A/B 测试、蓝绿部署、HTTP 请求头修改、双向 TLS 认证 (mTLS) 和 Web 应用防火墙 (WAF) 。
- **持续生产就绪性** —— 每个版本都按照合理的生产标准构建和维护。无论您使用的是基于 NGINX 开源版的版本还是基于 NGINX Plus 的版本, 您都可以从这种 “企业级” 特性中受益。NGINX 开源版用户可以在 [GitHub](#) 上咨询我们的工程团队, NGINX Plus 用户则可以获得一流的[支持](#)。无论哪种方式, NGINX 开发人员都将为效劳!

NGINX 开源版与 NGINX Plus —— 为什么要升级到我们的商用版？

既然说到这了，我们就来回顾一下基于 NGINX Plus 的 NGINX Ingress Controller 的一些主要优势吧。我们在第 7 章中说过，开源和商用 Ingress controller 之间存在实质性差异。如果您计划在生产环境中部署大规模 Kubernetes 和复杂的应用，那么我们的商用 Ingress controller 可以帮助您在某些关键领域节省时间和金钱。

安全性与合规性

许多企业和机构未能在生产中交付 Kubernetes 应用的一大原因是难以保障它们的安全性和合规性。基于 NGINX Plus 的 NGINX Ingress Controller 可以从如下五个至关重要的方面保证您的应用安全和客户安全（关于这些用例及更多内容请参见第 4 章）。

使用 NGINX Ingress Controller 提高安全性与合规性

基于 NGINX Plus 的 NGINX Ingress Controller 可以从如下五个
至关重要的方面保证您的应用安全和客户安全。



- 01 为边缘提供防护
- 02 集中管理身份验证和授权
- 03 实施端到端加密
- 04 获得及时主动的补丁通知
- 05 遵循 FIPS

阅读《NGINX 助力奥迪进行应用创新，打造面向未来的技术愿景》，了解德国汽车巨头奥迪是如何保护他们的红帽 OpenShift 应用的。



- **保护边缘** —— 在架构合理的 Kubernetes 部署中，对于数据平面的流量来说，Ingress controller 是流向在 Kubernetes 中运行的服务的唯一入口点，因而是部署 WAF 的理想位置。F5 NGINX App Protect WAF 集成了 NGINX Ingress Controller，可保护您的 Kubernetes 应用免受 OWASP 十大风险和许多其他漏洞的侵害，确保 PCI DSS 合规性，性能完胜 ModSecurity。
- **集中管理身份验证和授权** —— 您可以使用 OpenID Connect (OIDC，构建在 OAuth 2.0 框架之上) 和 JSON Web 令牌 (JWT) 身份验证在入站点实施一层身份验证和单点登录 (SSO)。
- **实施端到端加密** —— 如果需要保护 service 之间的流量，您可以选择使用 service mesh。F5 NGINX Service Mesh (始终免费) 可以无缝集成 NGINX Ingress Controller，让您以低于其他网格的延迟，有效控制入向和出向的 mTLS 流量。
- **及时获取打补丁预警通知** —— 一旦曝出 CVE 漏洞，用户就会收到预警信息，并快速获得补丁。用户可以立即应用补丁，降低被黑客利用的风险，而不必再巴巴地期盼 GitHub 上的更新，也不必再苦等数周甚至数月发布补丁。
- **符合 FIPS 的要求** —— 您可以启用 FIPS 模式，确保与 NGINX Plus 通信的客户端使用可信的强密码。



应用性能和弹性

正常运行时间和应用速度通常是开发人员和平台运营团队的关键性能指标（KPI）。基于 NGINX Plus 的 NGINX Ingress Controller 可以从五个至关重要的方面保证您的应用的正常运行时间和速度，帮助您实现 Kubernetes 的承诺（更多内容已经在[第 2 章](#)和[第 3 章](#)进行了介绍）。

**使用 NGINX Ingress controller
提高应用性能和弹性**

基于 NGINX Plus 的 NGINX Ingress Controller 可以从五个至关重要的方面保证您的应用的正常运行时间和速度，帮助您实现 Kubernetes 的承诺。

- 01 实时监控
- 02 更快地检测和解除故障
- 03 无需重启即可重新配置
- 04 全面测试新功能和部署
- 05 快速提供支持

 阅读《借助 NGINX 增强亚马逊 EKS 的安全性和流量可视性》，了解商务短信公司 Zipwhip 是如何实现 SaaS 应用 99.99% 的正常运行时间的。



- **实时监控** —— NGINX Plus 仪表板可显示数百个关键负载和性能指标，帮助您快速找到应用运行缓慢（或崩溃！）的原因。
- **更快地检测和解除故障** —— 实施主动检查运行状况的断路器，主动监控 TCP 和 UDP 上游服务器的运行状况。
- **无需重启即可重新配置** —— 更快速的无中断重新配置能够让您以**低于**开源替代方案的延迟，交付具有一致性能和资源使用率的应用。
- **全面测试新功能和部署** —— 利用键值存储更改百分比，避免了重新加载，有效简化了 A/B 测试和蓝绿部署的执行。
- **快速提供支持** —— 对于那些等不及社区回答问题或不想承担敏感数据暴露风险的企业和机构来说，保密的商业支持很重要。NGINX 提供了多个层级的支持，能够满足您多方面的需求，涵盖安装、部署、调试和纠错等。哪怕只是看起来不对劲儿，您也可以找我们获得帮助。



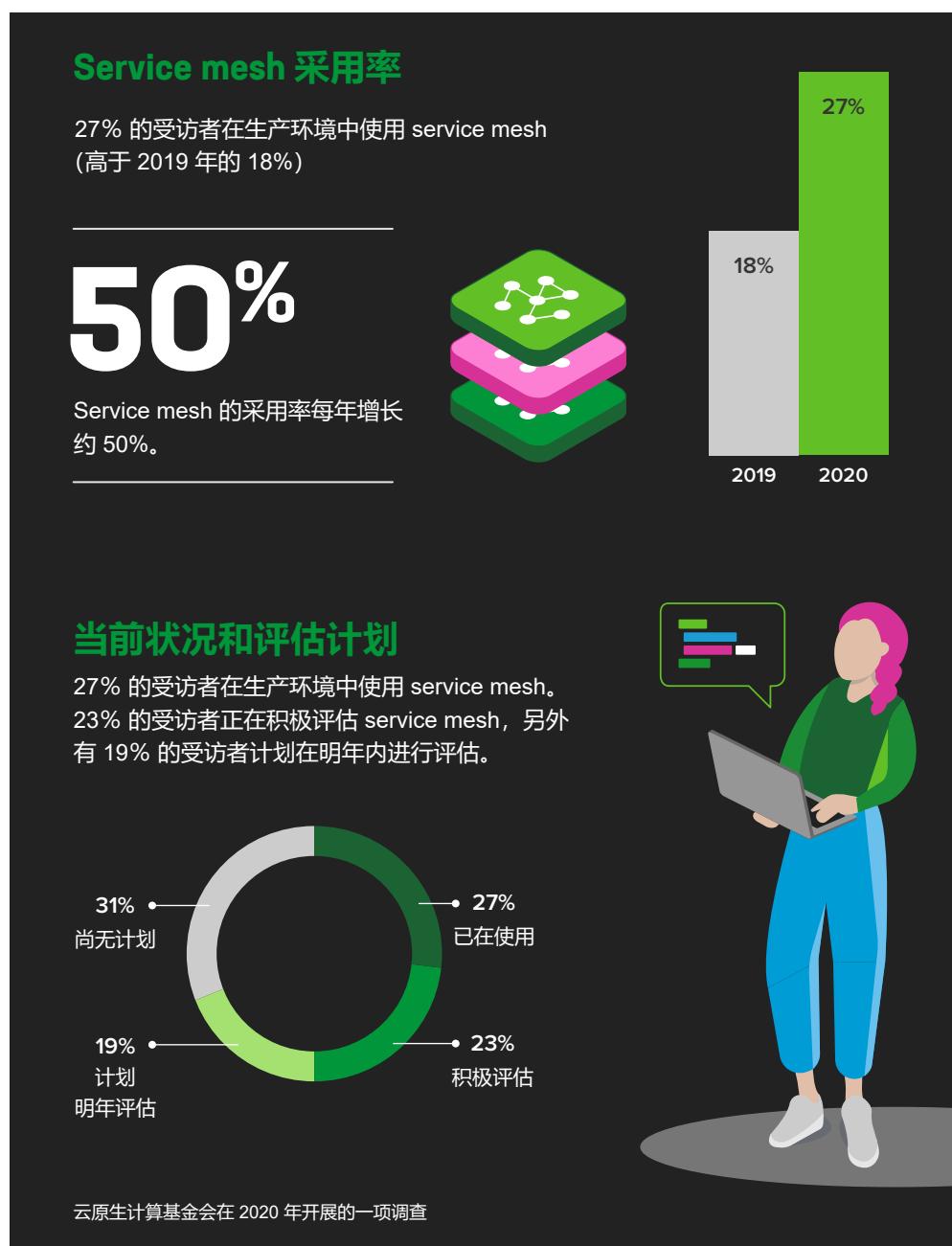
如果您认为开源 Ingress controller 对您的应用来说是最好的选择，那么您可以利用我们的 GitHub 代码库快速上手。

如果是大型生产部署，您不妨试试基于 NGINX Plus 的商用 Ingress controller。它提供 **30 天免费试用版**，其中包括 NGINX App Protect。

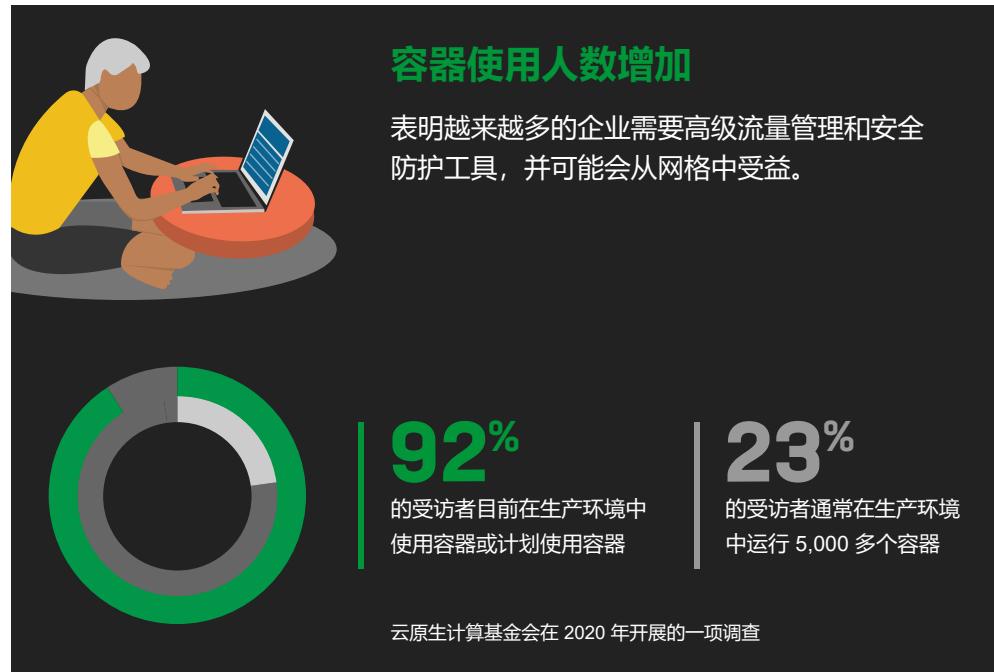
9. 如何选择 service mesh

近年来，随着企业不断加大对微服务和容器化应用的投资，service mesh 已从前沿技术稳步转向主流应用。根据云原生计算基金会在 2020 年开展的一项关于云原生技术使用情况的调查，我们得出了以下结论。

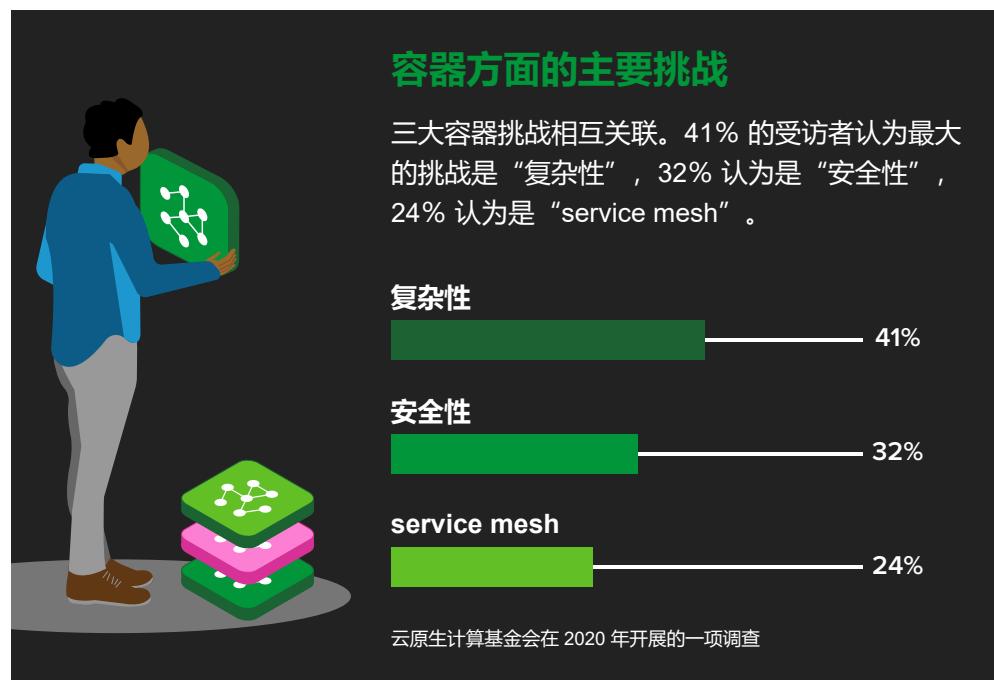
要点 1：service mesh 的使用人数迅速增长。



要点 2：容器使用人数的增加表明，越来越多的企业需要高级流量管理和安全防护工具，并可能会从网格中受益。



要点 3：三大容器挑战相互关联。



已经不是“我是否必须使用 service mesh？”的二元问题

您做好使用 service mesh 的准备了吗？

我们认为，这已经不是“我是否必须使用 service mesh？”的二元问题，而是“我何时做好使用 service mesh 的准备？”我们认为，对于在生产环境中部署容器并使用 Kubernetes 进行编排的任何人而言，其应用和基础架构设施的成熟度水平都足以让 service mesh 的价值体现出来。

但与任何技术一样，在实际需要 service mesh 之前就提前部署的做法弊大于利，它带来的风险和开销要超出可为您的企业所带来的潜在好处。对于想要采用 service mesh 的客户，我们根据以下 6 点来确定其准备情况并了解其现代化进程。您对以下陈述的肯定回答越多，service mesh 为您创造的价值就越大。

1：Kubernetes 是您生产环境中的唯一平台。

无论您已将生产应用迁移到 Kubernetes，还是刚开始测试将应用迁移到容器工作负载，Kubernetes 都在您的长期应用管理路线图中。

2：您需要一个零信任的生产环境，并需要 service 之间配置双向 TLS (mTLS)。

您已为生产应用采用零信任模型，并需要在容器化环境中保持这种级别的安全性，或者您正在使用迁移作为强制功能来提供您的服务级别安全性。

3：无论在数量还是服务深度上，您的应用都很复杂。

您拥有一个大型的分布式应用。它有多个 API 依赖项，并很可能需要外部依赖项。

4：您拥有一个成熟的生产 CI/CD 流水线。

“成熟”取决于您的企业。我们将该术语应用到以编程方式部署 Kubernetes 基础架构和应用的过程，可能使用的工具包括 Git、Jenkins、Artifactory 或 Selenium。

5：您经常在生产环境中进行部署——至少每天一次。

我们发现，对于这个问题，大多数人的回答都是“否”——尽管他们已将应用迁移到了生产 Kubernetes 中，但他们还没有使用 Kubernetes 来支持持续的版本修订。

6：您的 DevOps 团队已准备好开始使用强大的工具部署超现代应用！

即使 service mesh 为 NetOps 团队拥有，但通常由 DevOps 团队在集群内进行管理，后者需要准备好将网格添加到其堆栈中。

对上述 6 个陈述的回答不都是“是”？没关系！请继续阅读，了解您在做好准备之后，接下来该做的工作，包括您可以做些什么来让您的团队做好使用 service mesh 的准备。

您做好使用 service mesh 的准备了吗？

根据以下 6 点确定您的就绪度 —— 您对以下陈述的肯定回答越多，service mesh 为您创造的价值就越大。

01



Kubernetes 是您生产环境中的唯一平台。

02

你需要一个零信任的生产环境，并需要服务之间配置双向 TLS (mTLS)。



无论在数量还是服务深度上，您的应用都很复杂。

03

您拥有一个成熟的生产 CI/CD 流水线。



04

您经常在生产环境中进行部署 —— 至少每天一次。



06

您的 DevOps 团队已准备好开始使用强大的工具部署超现代应用！



Istio 既不是唯一的选择，也不是适用于每个人或每个用例的选择

如何选择适合您的应用的 service mesh

在您准备好使用 service mesh 之后，您还需要从众多 service mesh 选项中做出选择。就像 Kubernetes 已成为事实上的容器编排标准一样，Istio 通常被视为事实上的 service mesh 标准。事实上，Istio 很容易被认为是最适合的选择，不仅因为它很流行，还因为它的目标是解决 Kubernetes 网络环境中的所有问题。尽管有宣传自家产品的嫌疑，但我们还是要在这里告诉您，Istio 既不是唯一的选择，也不是适用于每个人或每个用例的选择。Istio 让您的环境变得非常复杂，可能需要集整个工程师团队的力量来运行，而最终带来的问题通常比解决的问题还要多。

为了明确说明哪种 service mesh 最适合您的应用，我们建议您与您的团队和利益相关者举行一次战略规划会议。下面是一个对话指南，可帮助您推动这些讨论。

第 1 步：您为何想要使用 service mesh？

换句话说，您需要 service mesh 解决什么问题？举例来说，您的企业可能要求在 service 之间配置 mTLS，或者您需要“端到端”加密，包括从边缘进入（针对入向流量）和从网格内流出（针对出向流量）的流量。或者，您可能需要为您的全新 Kubernetes service 配备企业级流量管理工具。

第 2 步：如何使用 service mesh？

这取决于您的角色。

- 如果您是开发人员：
 - 您是否计划提高正在迁移到 Kubernetes 的传统应用的安全性？
 - 在将应用重构为原生 Kubernetes 应用时，您是否打算将安全性纳入其中？
- 如果您负责平台和基础架构：
 - 您是否打算将 service mesh 添加到 CI/CD 流水线中，以便在每个新集群中自动部署和配置，并在开发人员启动新实例时可用？

第 3 步：影响您选择的因素有哪些？

您的 service mesh 是否需要不受基础架构的限制？是否需要与您的可视化工具兼容？是否是 Kubernetes 原生工具？是否需要易于使用？您是否预见到未来要使用与网格中的 service 到 service（东西向）流量管理工具相同的工具在边缘管理出入向（南北向）流量？

响应缓慢的数据平面会降低整个 Kubernetes 引擎的速度，并且会影响系统性能

第 4 步：评估 service mesh 选项

回答完这些问题之后，您将会获得明确的需求（以作为您的评估选项）清单。在此过程中，还有两个额外的领域需要评估：service mesh 的数据平面和 service mesh 可能带来的隐藏成本。

数据平面 —— 数据平面直接影响着客户的性能体验，因为响应缓慢的数据平面会降低整个 Kubernetes 引擎的速度，并且会影响系统性能。根据下列问题来评估待选 service mesh 的数据平面是否可以满足您的需求。

- 这个数据平面已经上市多少年了？
- 它的容量是多少？
- 它是否具有你将来需要且想要的集成功能？
- 它如何衡量并提供可观察性？
- 它能否从灾难性故障中动态恢复？

如欲了解更多信息，请阅读《[数据平面并非商品](#)》。

部署和运营 service mesh 的任何成本可能都并非显而易见

隐藏成本 —— 部署和运营 service mesh 的任何成本可能都不是显而易见的，一旦您超出了更新换代的范围，成本就会激增。用以下问题来准确了解您的 service mesh 选择可能带来的任何隐藏成本。

- 运行控制平面需要多少个容器镜像？每个镜像必须有多大？
- 您的 service mesh 的 Ingress controller 的容量是多少？
- 您的 Sidecar 可以跟上您的服务需求吗？
- 您会运行多个集群或多租户吗？
- 您的 service mesh 需要多少 Kubernetes CustomResourceDefinitions (CRDs)？
- 您是否需要专门的人员来运行 service mesh？如果需要，需要多少人员？
- 您所选的 service mesh 是否将您限制在一个特定的软件或云平台？

如欲了解更多信息，请阅读《[service mesh 的隐藏成本](#)》。

F5 NGINX Service Mesh

NGINX Service Mesh 于 2020 年作为开发版推出，是为开发人员优化的免费的轻量级服务网格，它提供了最简单的方法来实现 mTLS，以及 Kubernetes 中东西向（service 到 service）流量和南北向（入向和出向）流量的端到端加密。为了以一种侵入性最小但仍提供高级灵活性和关键洞察力的方式让您完全控制应用数据平面，我们构建了自己的 service mesh。

我们相信，如果您需要具有以下特性的网格，您一定会对 NGINX Service Mesh 一见钟情：



关于 NGINX Service Mesh 架构

NGINX Service Mesh 有两个主要的组件。

- **控制平面**

我们构建了一个轻量级的控制平面，可与 Kubernetes API 服务器搭配使用，动态支持和管理应用。它会随着应用的扩展和部署而做出响应和更新，以便让每个工作负载实例自动得到保护，并与其他应用组件集成——实现“一次设置，自动运行”，从而让您专注于处理业务。

- **数据平面**

我们的数据平面将企业流量管理、性能和可扩展性提升到了市场最高水平

NGINX Service Mesh 的真正厉害之处是全面集成的高性能数据平面。我们的数据平面利用 F5 NGINX Plus 的强大功能来运行高可用和可扩展的容器化环境，将企业流量管理、性能和可扩展性提升到了市场最高水平。它提供了生产级 service mesh 部署所需的无缝且透明的负载均衡、反向代理、流量路由、身份和加密功能。当与基于 NGINX Plus 版本的 F5 NGINX Ingress Controller 搭配使用时，它还可提供可通过单个配置管理的统一的数据平面。

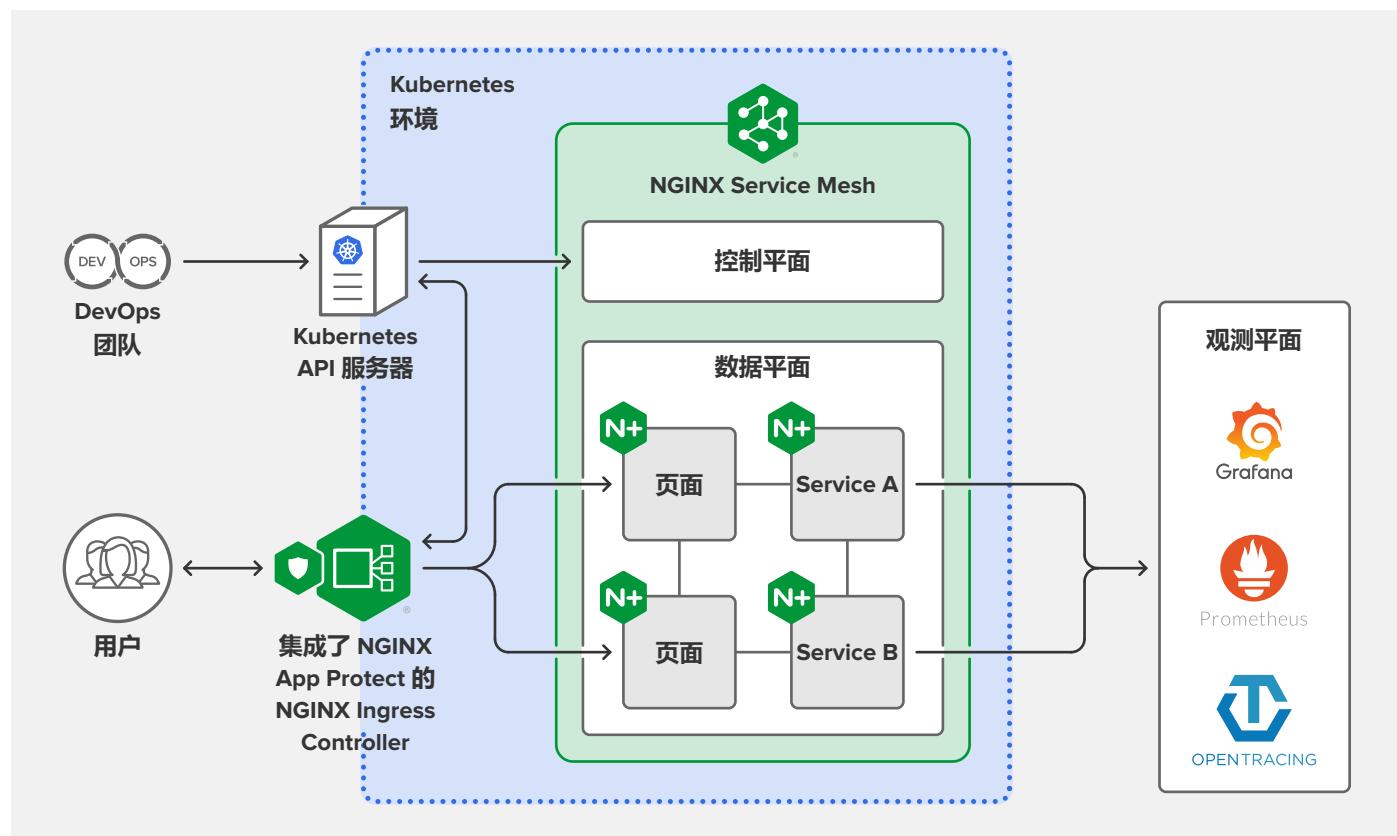


图 14: NGINX Service Mesh 架构

NGINX Service Mesh 的优势

您可从 NGINX Service Mesh 获得的几项优势：

- **较低的复杂性**

NGINX Service Mesh 易于使用且不受基础架构的限制。它实现了 Service Mesh Interface (SMI) 规范，该规范定义了 Kubernetes service mesh 的标准接口，并提供了 SMI 扩展，在部署新应用时，可极大地减少人工操作和生产流量中断。NGINX Service Mesh 还与 NGINX Ingress Controller 进行了原生集成，创建了一个统一的数据平面，支持您在边缘位置集中和简化出入向（南北向）流量管理以及 service 到 service（东西向）反向代理 sidecar 流量管理的配置。与其他网格不同，NGINX Service Mesh 不需要将 sidecar 注入 NGINX Ingress Controller，因此不会增加 Kubernetes 环境的延迟和复杂性。

- **更高的弹性**

借助我们的智能容器流量管理功能，您可以指定策略来限制流向新部署的 service 实例的流量，并随着时间的推移逐渐放宽限制。借助速率限制和断路器等功能，您可以完全控制通过 service 的流量。您可以利用各种强大的流量分配模型，包括速率整形、服务质量 (QoS)、服务节流、蓝绿部署、灰度发布、断路器模式、A/B 测试和 API 网关功能。

- **细粒度的流量洞察**

NGINX Service Mesh 可用于使用 Opentrocing 和 Prometheus 进行指标收集和分析。NGINX Plus API 通过 NGINX Service Mesh sidecar 和 NGINX Ingress Controller pod 生成指标。借助内置的 Grafana 仪表盘，您可以查看各种指标（指标信息可精确到毫秒、每日对比和流量峰值），从而进一步了解应用和 API 性能。

- **保护容器化应用**

将 mTLS 加密和七层防护扩展到各个微服务，并利用访问控制来定义描述应用拓扑的策略——让您能够细粒度控制哪些 service 被授权进行相互通信。NGINX Service Mesh 支持高级安全功能（包括配置控制和治理），以及对出入向和 service 到 service 流量的放行列表支持。借助基于 NGINX Plus 版本的 NGINX Ingress Controller，您还可以默认拦截访问内部 service 的南北向流量，并将 NGINX App Protect 用作边缘防火墙。

没有准备好使用 service mesh?

如果您还没有准备好使用 service mesh，那么您可能对 Kubernetes 不熟悉，或者正在努力克服大型生产部署所面对的障碍。现在正逢其时，您可以采用生产级 Ingress controller 和内置安全防护来解决复杂性、安全性、可视化和可扩展性等常见的 Kubernetes 挑战。

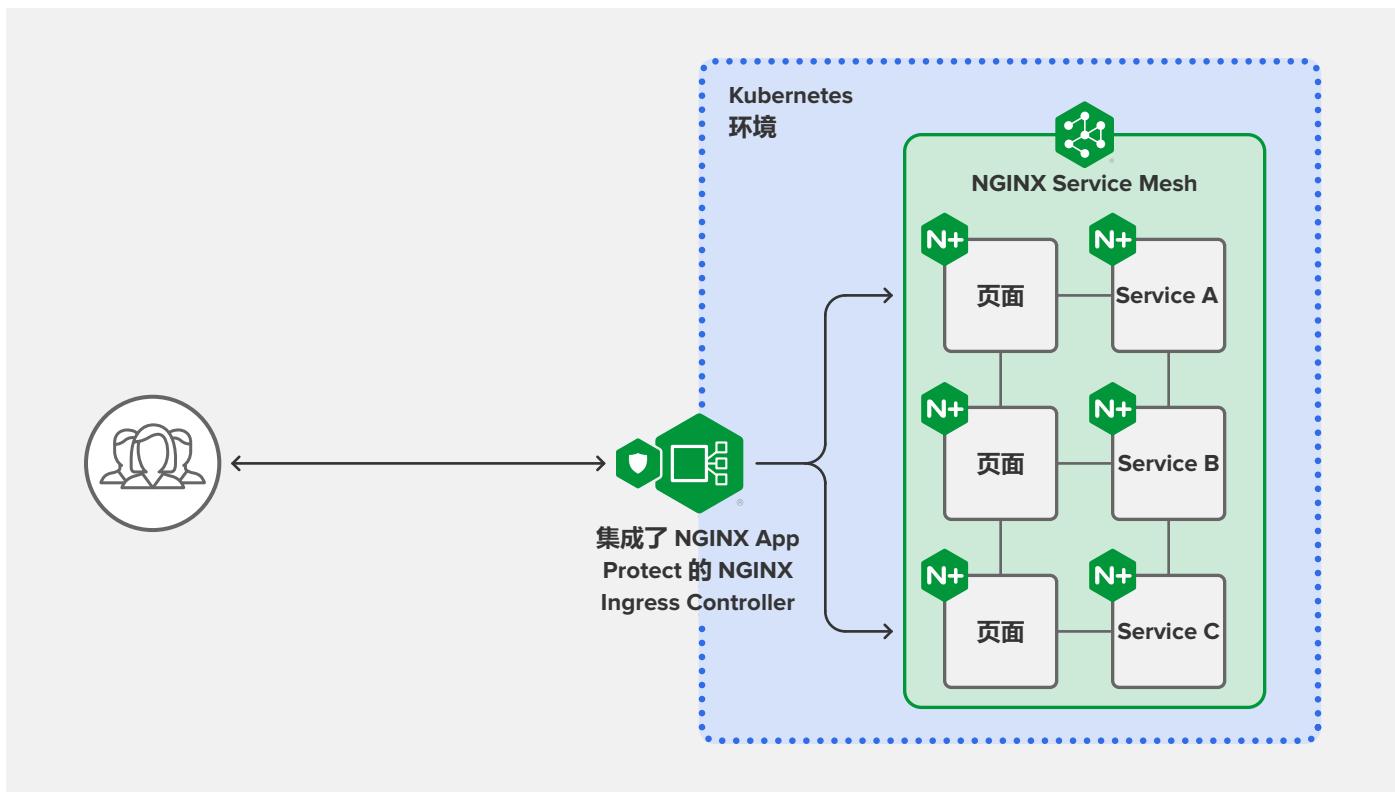


图 15：使用 NGINX 的生产级 Kubernetes

立即试用

NGINX Service Mesh 完全免费，您可[立即下载](#)并在 10 分钟内完成部署！您可查看我们的[文档以获取更多信息](#)。我们希望了解您的使用体验，欢迎您在 [GitHub](#) 上发表您的反馈意见。

如果事实证明 Istio 最能满足您的需求，请查看 F5 的 [Aspen Mesh](#)。它是一个建立在开源 Istio 之上的企业级 service mesh，拥有实时流量 GUI，是寻求交付 5G 服务的提供商的完美之选。

运行缓慢的应用会迅速导致客户流失

附录

三种 NGINX Ingress Controller 选择的性能测试

以下测试结果最初发布在《[NGINX Ingress Controller 在动态 Kubernetes 云环境中的性能测试](#)》博文中。请阅读博文获取完整的详细信息，包括测试方法和规格。

选择流量管理工具（尤其是对于敏捷 Kubernetes 应用而言）时，该工具可能增加的延迟量是最重要的因素之一。毕竟，运行缓慢的应用会迅速导致客户流失。我们比较了三个基于 NGINX 的 Ingress controller 在真实多云环境中的性能，并测量了客户端连接在互联网中的延迟情况：

- 基于 NGINX 开源版的 [NGINX Ingress Controller](#)，由 Kubernetes 社区维护。我们在这里将其称为“社区版”。我们从 [Google Container Registry](#) 中提取了其 0.34.1 版本的镜像，用于此次测试。
- 基于 [NGINX 开源版的 NGINX Ingress Controller 1.8.0](#)，由 NGINX 维护。我们在这里将其称为“NGINX 开源版”。
- 基于 [F5 NGINX Plus 的 NGINX Ingress Controller 1.8.0](#)，由 NGINX 维护。我们在这里将其称为“NGINX Plus 版”。

我们生成了稳定的客户端流量，并使用这些流量对 Ingress controller 进行压力测试，收集了以下性能指标：

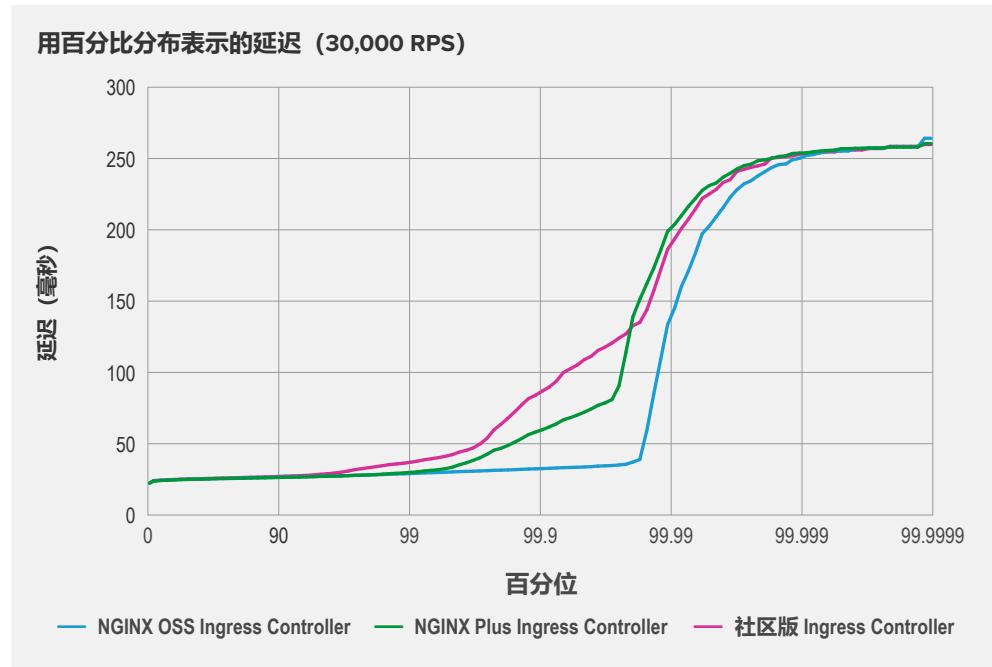
- **延迟** —— 客户端从生成请求到接收响应所用的时间。我们在报告中用百分位分布的形式来表示延迟。例如，如果有 100 个延迟测试样本，则第 99% 个的值是 100 次测试中仅次于最慢值的响应延迟。
- **连接超时** —— 由于 Ingress controller 在特定时间内无法响应请求而被静默断开或丢弃的 TCP 连接。
- **读取错误** —— 尝试读取连接失败，因为来自 Ingress controller 的套接字已关闭。
- **连接错误** —— 客户端和 Ingress controller 之间未建立的 TCP 连接。

我们在两种情况下进行了测试。在“静态部署”中，后端 pod 副本的数量在整个测试运行期间保持不变，一直是五个。在“动态部署”中，我们使用一个脚本定期将副本数量扩展到七个，然后再恢复回五个。正如以下结果所示，我们发现只有 NGINX Plus 版本在 pod 副本数量上升和下降时不会产生高延迟。

静态部署的延迟结果

如图所示，在静态部署后端应用时，三个 NGINX Ingress Controller 具有相似的性能。考虑到它们都基于 NGINX 开源版构建，并且静态部署不需要从 Ingress controller 重新配置，这一结果也很合理。

图 16：静态部署的延迟结果



动态部署的延迟结果

该图显示了在定期将后端应用从五个 Pod 副本扩展到七个又减少回五个的动态部署下，每种 NGINX Ingress Controller 产生的延迟。

很明显，只有 NGINX Plus 版在这种环境中表现良好

很明显，只有 NGINX Plus 版在这种环境中表现良好，一直到 99.99% 百分位都几乎没有任何延迟。社区版和 NGINX 开源版虽然具有截然不同的延迟模式，但都在很低的百分位上产生了明显的延迟。

- **社区版：**延迟呈缓慢但稳定的上升状态，在第 99 个百分位达到大约 5000 毫秒（5 秒）并在之后趋于稳定。
- **NGINX 开源版：**延迟呈急剧攀升状态，在第 99 个百分位达到大约 32 秒，到第 99.99 个百分位又变成了 60 秒。

社区版和 NGINX 开源版所经历的延迟是由 NGINX 配置更新和重新加载（以响应后端应用不断变化的端点）后出现的错误和超时引起的，具体内容我们将在下文“[动态部署中的超时和错误结果](#)”部分进行进一步讨论。

图 17：动态部署的延迟结果

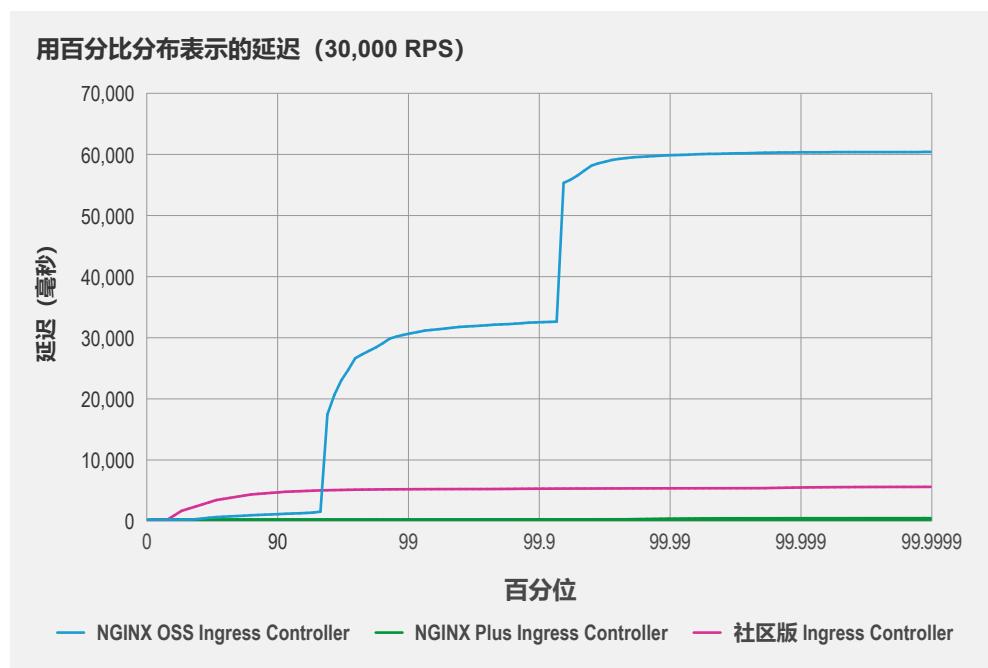
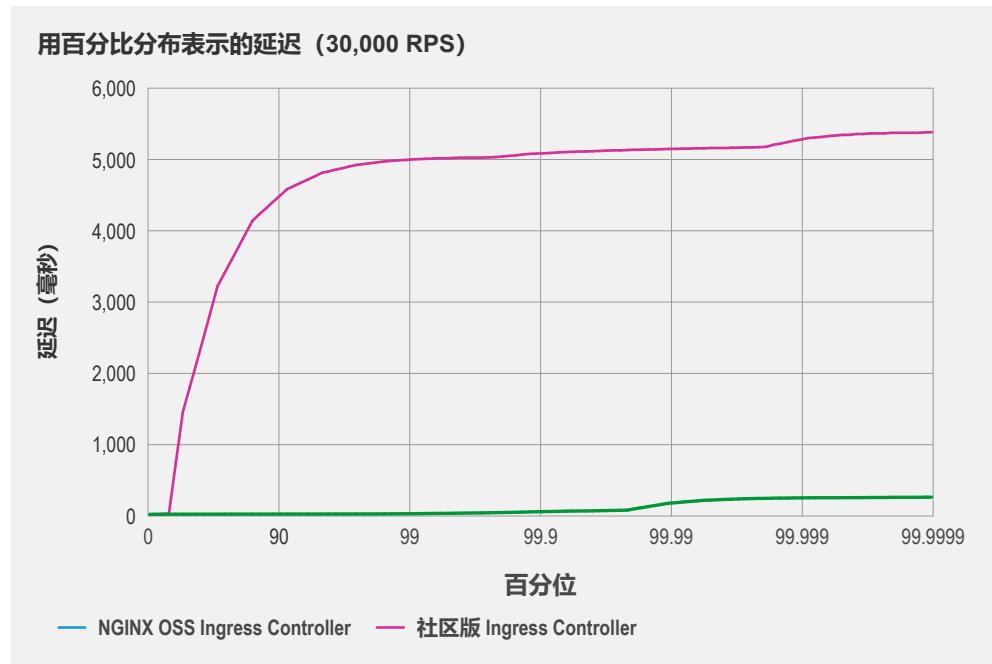


图 18：动态部署延迟结果的细粒度视图

这是一个更细粒度的视图，展示了社区版和 NGINX Plus 版在与上图相同的测试条件下得出的结果。NGINX Plus 在第 99.99 个百分位之前几乎没有延迟，并从第 99.9999 个百分位开始向 254 毫秒攀升。社区版 Ingress controller 的延迟在第 99 个百分位稳定增长到 5000 毫秒，此后趋于平稳。



动态部署中的超时和错误结果

此表更详细地显示了引起延迟的原因。

	NGINX 开源版	社区版	NGINX PLUS
连接错误	33,365	0	0
连接超时	309	8,809	0
读取错误	4,650	0	0

使用 NGINX 开源版 Ingress controller 时，每次更改后端应用端点后都需要更新并重新加载 NGINX 配置，这会导致许多连接错误、连接超时和读取错误的问题。当客户端尝试连接不再分配给 NGINX 进程的套接字时，系统会在 NGINX 重新加载的短时间内发生连接/套接字错误。当客户端与 Ingress controller 建立连接但后端端点不再可用时，会发生连接超时。错误和超时会严重影响延迟，导致延迟在第 99 个百分位激增到 32 秒，然后在第 99.99 个百分位变成 60 秒。

使用社区版 Ingress controller 时，端点随着后端应用的调整而更改，引发了 8809 连接超时。社区版 Ingress controller 使用 Lua 代码来避免在端点变更时重新加载配置。结果显示，在 NGINX 内部运行的检测端点变更的 Lua 处理程序解决了 NGINX 开源版 Ingress controller 的一些性能限制问题——这些限制是由每次更改端点后都重新加载配置引起的。尽管如此，连接超时仍然会发生，导致更高百分位的显著延迟。此外，正如我们在[第 6 章](#)中所讨论的，在基于 OpenResty 的 Ingress controller（包括社区版本）中使用 Lua 可能会带来意想不到的风险。

而使用 NGINX Plus 版时没有错误或超时——动态环境对性能几乎没有影响。这是因为 NGINX Plus 使用了 NGINX Plus API 在端点变更后动态更新 NGINX 配置。如上文所述，它的最高延迟为 254 毫秒，发生在第 99.9999 个百分位。

结语

性能结果表明，要想完全消除动态 Kubernetes 云环境中的超时和错误，Ingress controller 就必须动态适应后端端点的变更，且无需使用事件处理程序或重新加载配置。根据这些结果，NGINX Plus API 可以说是在动态环境中重新配置 NGINX 的最佳解决方案。在我们的测试中，只有基于 NGINX Plus 的 NGINX Ingress Controller 在高度动态的 Kubernetes 环境中实现了符合用户需求的完美性能。

术语

A A/B 测试 (A/B Testing)：通过衡量和比较用户行为来确定不同产品或应用版本在客户群体中的受欢迎程度。

(第 18 页)

应用层 (Application Layer)：参见[七层](#)。

身份验证和授权 (Authentication and Authorization)：用于确保只有“正确”的用户和 service 可以访问后端或应用组件，是系统必备的功能：

- **身份验证 (Authentication)**：验证“身份”，确保发出请求的客户端与自己声称的身份相符。
- **授权 (Authorization)**：验证资源或功能的访问“权限”。通过访问令牌实现，例如会话 cookie、会话 ID、组 ID 或令牌内容等七层属性。

(第 34 页)

B 蓝绿部署 (Blue-Green Deployment)：一种可以缩短甚至消除升级停机时间的技术，支持在同一生产环境中继续运行旧版本（蓝）的同时部署新版本（绿）。

(第 19 页)

C 灰度部署 (Canary Deployment)：一种安全、敏捷的方法，用于测试新特性或新版本的稳定性。大多数用户会继续使用稳定版，而少数用户迁移到新版本。

(第 17 页)

断路器 (Circuit Breaker)：一种通过监控服务故障来防止级联故障的方法。

(第 15 页)

云原生 (Cloud-Native)：一种应用开发和交付方法，支持企业使用容器、service mesh、微服务、不可变基础架构和声明式 API 在现代动态环境（例如公有云、私有云和混合云）中构建和运行可扩展的应用。

条件路由 (Conditional Routing)：参见[调试路由](#)。

容器 (Containers)：一种虚拟化技术，旨在创建和支持应用的便携式标准包，从而可以轻松地将应用部署在各种不同的平台上。容器将应用的所有需求（应用代码本身、应用需要运行的库等依赖项，以及应用及其依赖项的运行时环境）打包成一个可以跨平台独立传输和运行的外形。

通用漏洞披露（CVE, Critical Vulnerabilities and Exposures）：这是一个漏洞数据库，它公开披露了“软件、固件、硬件或服务组件中可能会遭到利用的漏洞缺陷，这些缺陷会破坏受影响的一个或若干组件的机密性、完整性或可用性”。¹

(第 29 页, 第 48 页, 第 61 页)

D 调试路由/条件路由（Debug Routing/Conditional Routing）：一种公开部署应用但对拥有访问权限的用户之外的其他用户进行“隐藏”的技术，基于第七层属性（例如 session cookie、session ID 或 group ID）。

(第 16 页)

拒绝服务（DoS）攻击（Denial-of-Service（DoS）Attack）——一种网络攻击，恶意请求（TCP/UDP 或 HTTP/HTTPS）犹如洪水般涌向受害网站，目的是让网站瘫痪。

(第 31 页)

分布式拒绝服务（DDoS）攻击（Distributed Denial-of-Service（DDoS）Attack）：一种 DoS 攻击，多个攻击源瞄准同一个网络或服务进行攻击，由于攻击者网络规模可能较大，防御难度比标准的 DoS 攻击更大。

E 东西向流量（East-West Traffic）：在 Kubernetes 集群内的 service 之间移动的流量（也称为“service 到 service 流量”）。

出向流量（Egress Traffic）：离开 Kubernetes 集群的流量。

端到端加密（E2EE, End-to-End Encryption）：对从用户传输到应用和后端的数据全程进行完全加密。E2EE 需要 SSL/TLS 证书并可能需要 mTLS，并且是零信任策略和环境的关键组成部分。

(第 36 页)

I Ingress controller（Ingress 控制器）：用于 Kubernetes（和其他容器化）环境的专用负载均衡器。

(第 8 页, 第 41 页)

入向流量（Ingress Traffic）：进入 Kubernetes 集群的流量。

洞察（Insight）：对人或物拥有深刻的理解。

K Kubernetes：一种开源容器编排系统，为管理和扩展部署在容器中的应用提供了一个完整的平台。通常缩写为“K8s”（发音为“kates”）。

1. 来源：[CVE 计划](#)。

- L 四层 (Layer 4)** : OSI 模型中的处于中间位置的“传输层”，负责处理消息的传递，而不考虑消息的内容。
- 七层 (Layer 7)** : OSI 模型中处于上层位置的“应用层”，处理每个消息的实际内容。
- M 微服务 (Microservices)** : 一种软件架构方法，将多个执行单一功能（例如身份验证、通知或支付处理）的小组件构建成一个大型复杂应用，也是意指这些小组件本身的术语。每个微服务都是软件开发项目中的独特单元，拥有自己的代码库、基础架构和数据库。微服务协同工作，通过 web API 或消息队列进行通信以响应传入事件。
- 双向 TLS (mTLS, Mutual TLS)** : 一种对客户端和主机都进行身份验证（通过 TLS 证书）的实践。[什么是 mTLS?](#) F5 Labs 提供了关于 TLS 和 mTLS 的绝佳入门介绍。
(第 35 页, 第 72 页)
- N 南北向流量 (North-South Traffic)** : 进入和离开 Kubernetes 集群的流量（也称为“出入向流量”）。
- R 速率限制 (Rate Limiting)** : 限制用户在给定时间段内可以发出的请求数量。
(第 14 页)
- S Service mesh (服务网格)** : 一种用于路由和保护 service 到 service 流量的流量管理工具。
(第 10 页, 第 64 页)
- Sidecar**: 一个单独的容器，与 Kubernetes pod 中的应用容器并列运行。通常，sidecar 负责从应用卸载 service mesh 中所有应用所需的功能（SSL/TLS、mTLS、流量路由、高可用性等），并实施部署测试模式，例如断路器、灰度和蓝绿部署。
- 单点登录 (SSO, Single Sign-On)** : 要求用户只需进行一次身份验证即可访问多个应用和服务的做法。SSO 技术（包括 SAML、OAuth 和 OIDC）可帮助简化身份验证和授权的管理。
(第 34 页)
- **简化的身份认证 (Simplified Authentication)** : SSO 让用户省去了针对每个不同的资源或功能设置唯一 ID 令牌的麻烦。
 - **标准化授权 (Standardized Authorization)** : SSO 有助于根据角色、部门和职位级别设置用户访问权限，无需为每个用户单独配置权限。

安全套接字层 (SSL, Secure Sockets Layer) / 传输层安全 (TLS, Transport Layer Security)：一种用于在联网计算机之间建立经过身份验证和加密的链接的协议。（尽管 SSL 协议在 1999 年被弃用，但仍将这些相关技术称为“SSL”或“SSL/TLS”。）SSL 证书能够验证网站的身份并建立加密连接。

(第 36 页)

T 流量控制/流量路由/流量整形 (Traffic Control/Traffic Routing/Traffic Shaping)：控制流量去向及其传输方式的行为。

(第 14 页)

流量精分/流量测试 (Traffic Splitting/Traffic Testing)：流量控制的一个子类，其中传入流量被划分为不同的流，这些流定向到在环境中同时运行的后端应用的不同版本（通常为当前的生产版本和更新版本）。定向到更新版本的流量比例通常会逐渐增加，直到该版本接收到所有流量。

(第 14 页)

传输层 (Transport Layer)：参见四层。

V 可视化 (Visibility)：能够看到或被看到的状态。

(第 23 页, 第 72 页)

W Web 应用防火墙 (WAF, Web Application Firewall)：一种反向代理，能够检测和拦截针对应用和 API 的复杂攻击（包括 OWASP 十大安全漏洞和其他高级威胁），同时让“安全”的流量顺利通过。

(第 9 页, 第 31 页, 第 44 页, 第 61 页)

Z 零信任 (Zero Trust)：一种用于高安全性企业的安全概念，但与每个人息息相关。“零信任”环境意味着数据必须在存储和运输的所有阶段都得到保护，并意味着企业决定在默认情况下不“信任”任何用户或设备，且所有流量都必须经过全面检查。零信任架构通常会包括一组身份验证工具和 mTLS，并且企业实施端到端加密的可能性很大。

(第 37 页, 第 66 页, 第 72 页)

F5 NGINX 市场销售热线: 400 991 8366
F5 NGINX 售后支持电话: 400 815 5595, 010-5643 8123
与我们在线沟通: contactme_nginxapac@f5.com



**NGINX 官方社区
微信公众号**

(产品信息、解决方案、活动资源)

**NGINX 开源社区
微信公众号**

(技术资料、活动信息、社区动态)

**立即加入
NGINX 社区微信群**

(答疑互动、行业交流、活动提醒)

F5 NGINX 北京办公室

地址: 北京市朝阳区建国路 81 号华贸中心 1 号
写字楼 21 层 05-09 室
邮编: 100025
电话: (+86) 10 5643 8000
传真: (+86) 10 5643 8100
<https://www.nginx-cn.net/>

F5 NGINX 上海办公室

地址: 上海市黄浦区湖滨路 222 号企业天地
1 号楼 1119 室
邮编: 200021
电话: (+86) 21 6113 2588
传真: (+86) 21 6113 2599
<https://www.nginx-cn.net/>

F5 NGINX 广州办公室

地址: 广州市天河区珠江新城华夏路 10 号
富力中心写字楼 1108 室
邮编: 510623
电话: (+86) 20 3892 7557
传真: (+86) 20 3892 7547
<https://www.nginx-cn.net/>