QINGTENG
青藤云安全

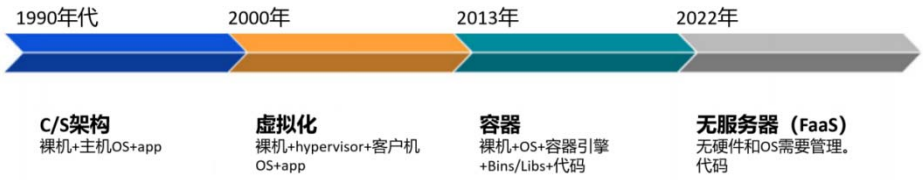
云原生环境下

如何打造一站式容器安全

目录

第 1 章　云原生下的容器安全

现代计算的发展可以分为四次浪潮，每次浪潮的到来都促进了 IT 技术的快速发展。伴随效率的提升，也引发了新的安全挑战。



第一次浪潮：发生在 1990 年代之前，最明显的特征是在裸机服务器上运行 C/S 架构。通常，服务器上只有一个操作系统，也只运行一个应用。

第二次浪潮：始于 2000 年左右，功能完整的操作系统虚拟化技术的出现。虚拟化的发展改变了服务器市场，迎来了虚拟计算的时代。

第三次浪潮：以云和容器技术为主要特征。容器能够得到广泛采用，主要是有两个推动因素：在 DevOps 的推动下，需要加快产品的上市时间；以及各类应用要实现在不同云端之间的可移植性。

第四次浪潮：功能即服务（FaaS），现在也正在快速发展，但是尚未得到企业的广泛应用。功能即服务（FaaS）不受硬件和操作系统影响，对计算能力进行了更高层次的抽象。

目前，我们处于第三次浪潮的快速发展时期，以容器为代表的云原生得到了市场的高度认可，目前正在成为新一代主流 IT 基础设施。

多年来，开发人员已经对反复处理操作系统和应用依赖问题不胜其烦。容器具有很好的可移植性，通过将所有文件打包在一起，实现了“一次打包，随处运行”，有效解决了操作系统和应用依赖的打包问题。虽然容器使用率越来越高，但是相关安全建设却处于落后阶段。

1. 容器安全成为云安全第一战场

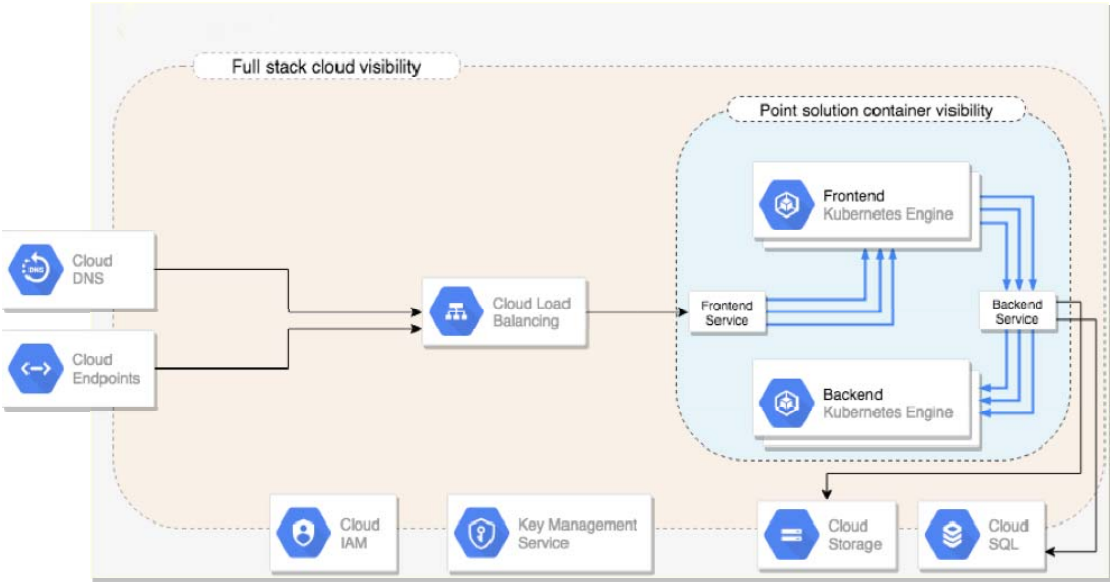
容器是一种轻量级的虚拟化技术，主要致力于提供一种可移植、可重用且自动化的方式来打包和运行应用。容器这一术语是对船运集装箱的一个类比，它提供了一个标准化方式，将不同内容组合在一起，同时又将它们彼此隔离开来。对于容器技术：

- 您可以在云中部署容器。通常还可以使用开源工具来管理云中的容器。这意味着，容器的使用很大程度地提高了在云上的移动性。
- 可以使用容器在云中部署应用程序，而不必纠结是选择云服务提供商的虚拟服务器还是选择计算实例。
- 云供应商可以使用容器来构建其他类型的服务，例如无服务器计算。
- 容器为在云中运行的应用程序提供安全优势。容器应用程序和主机环境之间增加了另一层隔离，而无需再运行整个虚拟服务器。

因此，容器的应用大大简化了云应用的部署。在云原生领域中，容器和云齐头并进，共同发

展，可以说容器技术是云原生应用发展的基石。而云和容器之间也有着本质的联系，云安全离不开容器安全，不可能脱离容器安全而单独去讲云安全。

虽然现在市场上有各类开源和商业容器安全产品，但很多点状产品只是解决了容器某一特定方面的安全挑战，完全没有把握容器安全的整体状况。例如，在容器上开发的大多数应用都是在使用各类开源和商业的 IaaS 和 PaaS 服务。但是由于缺乏对 IaaS 和 PaaS 环境的可见性，导致企业在云端面临一些风险。如果没有完整的可见性，并采取控制措施限制横向移动，则很难检测到攻击者在容器中的横向移动。



了解容器的整体堆栈可以有效缓解风险，而只注重于解决容器安全某一特定方面的点状产品则可能会因为没有全局观而捕捉不到这一风险。例如，当 CVE 最新公告公布了容器最新漏洞，了解整个容器的堆栈情况则可以解决这一漏洞所有信息，包括受影响的资产范围、镜像补丁的修复状态等。

2. DevSecOps：安全左移和自动化是全生命周期容器安全的前提

鉴于容器和云的运行速度非常快，DevSecOps 是安全团队唯一可行的途径。DevSecOps 将 DevOps 团队和安全团队整合在一起，尽早在容器生命周期中引入安全。这种方法将安全嵌入到整个容器生命周期中：构建、部署和运行。

在当今快节奏的容器 DevOps 中，部署容器时，安全人员必须注意容器共享 OS 内核带来潜在风险、镜像漏洞、错误配置、容器间网络流量问题。不管是防火墙还是入侵防御系统，并不适用于容器环境。

此外，随着现代环境越来越多地由软件控制，并实现了自动化，停止作业进行安全评估已不再可行。标准的做法是尽可能地将安全融入生产，并且通过微服务进行快速迭代。

安全左移

现在容器镜像是由开发人员构建的，因此，开发人员承担了许多其他职责。开发人员对以前由其他团队处理的活动所具有的控制权越来越多，包括一部分测试和运营。

无论是在测试环境中还是在生产环境中，由开发人员构建的容器镜像的运行方式都是相同的。因此，容器镜像必须是独立的而且适配各类环境，只需配置好与容器相关的安全和加固策略，适当调整可与生产资源连接的基本操作系统即可。

因此，容器安全必须左移，从代码编写和编译时就要确保容器安全。如果在生产过程中解决安全问题必然会产生大量浪费，因为修复发现的问题就要终止创新管道，并将容器返回到开发阶段。

自动化

容器是 DevOps 实践最佳承载方式之一，微服务事实上成为新应用程序体系结构。借助容器化微服务，不同的开发团队通过为每个微服务建立并行开发管道，可将创新速度提高十倍以上。

当这些微服务投入生产时，编排软件可以对其进行部署和管理。编排软件是基于策略来运行的，从而让容器部署实现远超人工可以实现的效果。这就有效地让人员从容器化生产环境运营中抽出身来，让他们能够进行策略制定和监控异常。这就意味着支持自动化、API 和策略运行，从而让容器具有自行评估和采取措施的能力。

3. 全生命周期解决方案

（1）构建阶段

◆ 安全镜像扫描

通常，容器镜像是从根镜像的基础上构建出来的，根镜像提供了操作系统组件和应用程序中间件（例如 node.js 或 Tomcat）。然后，开发人员通过自己的代码对根镜像进行拓展，形成微服务的应用程序结构。一般情况下，无需修改即可直接使用 Kafka 或 Vertica 等应用中间件。

但是，如果是从 Docker Hub 等公共仓库获取根镜像时，开发人员无法了解那些未经测试和未经验证的根镜像究竟会带来哪些安全风险。因此，需要通过容器镜像扫描，检测是否包含了常用漏洞和风险（CVE），减少最终容器镜像的攻击面。

正确的镜像扫描应包括以下几个级别：

- 进行镜像扫描，检查根镜像，检测开源镜像库中是否有已知的第三方漏洞。
- 对配置和部署脚本进行静态扫描，及早发现错误配置问题，并对已部署的镜像进行动态基础架构加固扫描。

◆ 对受信镜像进行签名和注册

在查看容器镜像时，确保已对其进行了扫描和测试以确保安全。了解这一点很重要，因为这会影响下一个阶段（例如转移到生产环境中）的其他检查点。

在成功进行镜像扫描和创建安全评分后，可以对容器镜像重新签名，包括安全评分和测试结果，标明该容器镜像已经过测试并达到特定的安全状态等级。

◆ 观察应用程序行为

当开发人员通过许多容器化的微服务形成其工作负载和应用程序时，网络将成为应用程序结构。网络动态绑定所有微服务。以前，所有逻辑都是在编译时绑定的。现在，微服务是解绑的，并根据需要在运行时与其他服务形成连接。

当然，判断正常行为和异常行为并非易事。将一个应用程序分解为可服务多个应用程序的微服务时，威胁建模要困难得多。建议在开发和集成时观察微服务架构，了解哪些是正常行为，这有助于威胁建模。在生产中，可通过威胁模型检测异常行为，进行隔离。

（2）分发阶段

◆ **审核已知内容**

随着容器镜像从一个镜像仓库迁移到另一个镜像仓库（不管是内部还是外部运行的），遇到包含未知漏洞的镜像风险都会增加。容器安全系统需要在通过容器镜像仓库时验证容器镜像，一旦发现不合规情况，就要拦截和隔离相关镜像。

每次将容器升级到新状态时（例如，从开发到测试或从测试到生产），都应执行额外的强制措施，以确保在上一阶段为了方便进行调试/监视/基准测试而添加的任何配置不会随容器本身而进入下一个阶段。

◆ **审核风险评分**

容器和镜像层的安全策略非常广泛，因此，很难设置一个人为管理的统一且易于维护的安全策略。对安全策略进行编码，对每个检查点的每个容器镜像生成一个风险评分，这样就可以实现容器生命周期的标准化，并对每个重要的检查点中设置最低安全阈值，如果没有达到最低水平，将对容器生命周期进行控制。

风险评分还有助于促进 Dev、Sec 和 Ops 的协同合作，因为这个统一的风险评分是对这三方的综合评分，这有助于促进不同团队和专业人士保持统一行为。

（3）部署阶段

◆ **自动部署**

开发人员正在以不断加快的速度创建容器格式的微服务。不仅 DevOps 管道难以管理，而且在生产环境中，在调度和编排方面，人工操作要让位于机器操作，因为编排和调度程序可以实现容器化微服务的自动化部署。编排程序可以比人做出更好的布局和扩展决策，因此，可以统一稳定地实施安全策略。

为了将管理良好的安全状态始终维持在一个可接受的水平上，要将容器安全软件与部署系统联系起来，以便您按统一方式遵守安全策略。

基于基础架构即代码（IaC）原则，要对所编写的、支持自动化部署任务的代码进行扫描和验证，按照应用程序代码级别，发现代码中存在的漏洞。

◆ **安全基础架构**

在主机上部署干净无漏洞的容器仍然会有安全风险，需要实施相关的加固最佳实践，否则，针对流氓容器的保护就太少了。例如，以 CIS 基准或公司加固策略为基准，查看与加固最佳实践存在哪些偏差；向管理员发出警报，并为容器化的基础架构提供安全评分。

◆ **用户和机器审计**

通过完整的审核跟踪，团队可以调查导致安全事件的原因，并据此采取补救措施并实施新的安全策略。这就需要知道谁做了什么，哪个容器编排程序将哪个镜像部署到了物理或虚拟主机，或者为什么阻止了容器镜进行部署或访问给定资源。

容器安全系统需要与人为操作系统和机器操作系统连接起来，对容器基础设施上发生的所有事件创建准确的审计跟踪，并记录其自身的行动和活动。

（4）运行阶段

◆ **秘钥管理**

在很多系统中，密码和安全令牌之类的秘钥是安全系统的一个重要组成部分。在主机上部署容器镜像或访问基于网络的资源时需要这些秘钥。若将秘钥存储在容器或环境变量中，所有有权访问该容器的人都会可以看到。

容器系统要求在进行操作（例如容器部署）时使用秘钥。因此，容器安全系统通常需要访问秘钥系统，在容器命令中注入正确的秘钥进行部署和运行。因此，容器的秘钥管理需要采用专门解决方案。例如，与 HashiCorp 的 Vault 之类的秘钥系统集成后，仅特定用户和容器可以访问特定秘钥。

◆ **与主机隔离**

在主机上运行的容器可能会访问主机资源，例如计算资源、存储资源和网络资源。此外，由于容器通常包含微服务，因此从本质上讲，容器应仅限于一些特定任务，并且每个容器通常负责的任务只有一个。

一旦流氓容器获得对主机资源（尤其是网络）的访问权限，就可以从网络上获取更多资源，进一步渗透到其他主机和系统。应该限制正在运行的容器的访问权限，并且这些容器只能使用已批准的特定主机资源，从而限制其对主机和网络的影响。

◆ **容器网络安全**

一台主机上有多个容器，每个容器都与同一主机或基于网络的服务上的相邻容器进行交互，仅依靠网络安全措施是不够的，因为主机内部发生的一切对于这些解决方案都是不可见的。

容器安全解决方案需要更靠近主机上发生事件的地方。比如，通过容器化的 agent，它会监控所有主机网络活动，主机上运行的容器的流入和流出，并观察容器是如何与不同主机的另一个网络进行交互的。了解了网络交互情况之后，可以通过网络流量加固活动来阻止任何异常活动。

应根据在构建阶段进行的威胁建模，预先规划适当的网络分段，并对其进行实施和验证，确保失陷容器尽可能少地影响其他网段，而不会对整个网络造成重大影响系统。

◆ **应用程序配置文件加固**

应用程序架构是由众多微服务构建而成，每个微服务都具有一个或多个实例，可以实现弹性扩展。应用程序架构中的容器在不断发生变化，主要由编排产品来实现的。

因此，想要维持正确的应用程序拓扑和观察异常行为也愈发困难。例如，异常行为可能是由

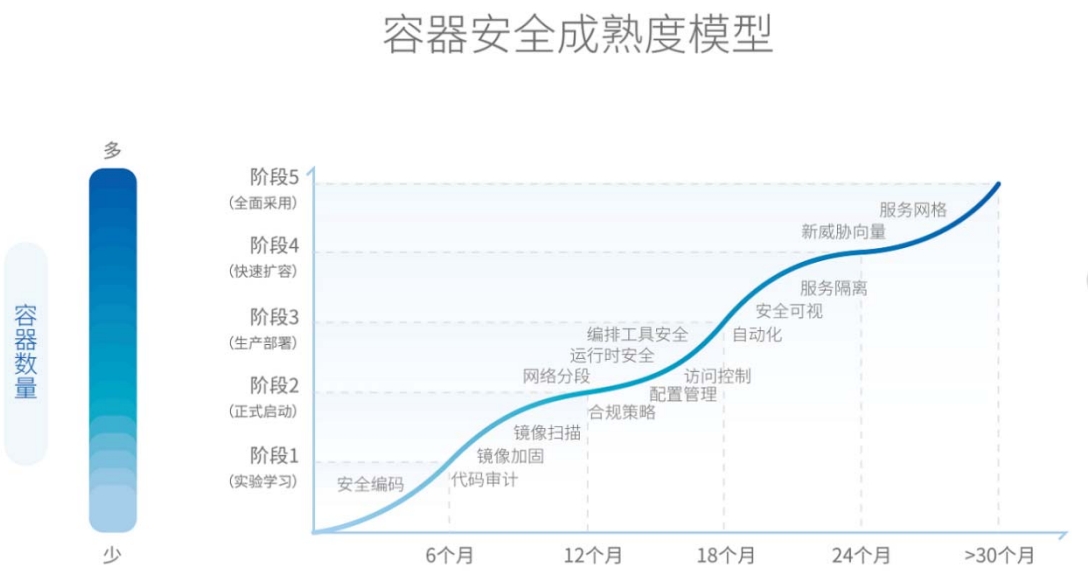
软件缺陷带到了生产环境中造成的，也可能是通过第三方开放源代码库渗透的恶意代码引起的。

建议，在开发阶段捕获正常的应用程序拓扑和行为，然后将其用于与生产环境的实际行为进行对比，从而发现异常情况。另外，可以先让应用程序运行一段时间，确定在生产环境中的预期应用程序，然后再捕获哪些属于正常网络行为，并将其另存为正常应用程序行为配置文件。

第 2 章　容器安全成熟度

虽然越来越多的企业开始采用容器化的云原生应用，但传统的安全措施却无力应对新技术发展带来的挑战。当下云原生虽说异常火热，却没有一个清晰的路线图，能够指明在容器化过程中，各个阶段该如何做好安全。由于容器化的每个阶段都会带来新的安全挑战，企业必须在做好基础架构的同时，确保每个阶段的安全。可以预见，从第一个容器化应用开发到管理数千个容器的过程中，安全需求也在不断变化。

在很多情况下，企业并不完全了解容器化架构所面临的安全挑战，更不用说如何主动应对这些挑战。这时，常见的做法就是将之前的安全策略应用到容器中来，但这并不一定适合新的应用体系架构和开发工作流程。总得来说，就是容器使用率不断提高，但安全总是落后一步。容器安全成熟度模型的发布，可以帮助企业更好了解需要部署哪些重要工具，采取哪些措施，来满足当前和未来容器安全需求。



1. 第 1 阶段：实验学习

在这个阶段，主要是开发人员自己学习如何在机器上使用容器。例如，通过一些不进入生产系统的项目做练习。在这一阶段，所涉及的应用大多是一些基本应用。

这阶段可能持续数月之久。由于只有个别开发人员在学习和测试容器，因此只要项目不是用

于生产，就不需要专用的安全工具，也不需要改变以往安全策略。与往常一样，开发人员只需确保安全编码。但是，这个阶段安全也是声明式的和自动化的，是开发工作流程的一部分，而不是在完成应用构建后才解决安全问题。

虽说，在该阶段安全并不太重要，也不会犯什么错误。但是，如果组织机构打算扩展容器化项目，则需要确保让每个人都了解，安全风险和复杂性会在扩展后迅速增加。在第 1 阶段和第 2 阶段之间的安全挑战存在巨大差异的，这会让许多组织机构措手不及。

2. 第 2 阶段：正式启动

在这个阶段，容器化已不再是个别开发人员的业余项目，而是由团队或业务部门开展一部分正式项目。这时，开发团队会用容器创建一个用于生产的新应用或将现有应用容器化。

大多数组织机构在开发其第一个容器化应用时，可能会使用下列其中一种方法：

- 将现有应用容器化
- 将现有应用的一部分内容容器化
- 从头开始在容器中开发新应用

无论开发团队采取哪种方式，这一阶段属于概念验证阶段，主要了解云原生技术会带来效益，以及应用是如何在容器中运行的。

开展一个正式项目，就需要多人合作，这时就有必要建立一个镜像仓库了。组织机构可以创建私有镜像仓库，通过策略规定谁可以访问镜像或镜像仓库，确保镜像来源可信。毕竟攻击者通常会通过镜像仓库中的受损镜像，获得对某个应用的初始访问。

这个阶段的第一个核心问题是认知偏差。从开发人员到习惯使用传统工具的安全专家，几乎没有人完全了解容器中可能出现潜在安全问题。这种知识或技能上差距会导致公司“所采取的安全”措施与“应该采取的安全措施”之间会出现差距。

这一阶段的第二个核心问题是没有做好未来规划。只考虑当前阶段所需的工具和安全需求，而没考虑容器化项目扩大使用范围之后所需的安全能力。公司在此阶段必须考虑以下注意事项：

合规策略。容器安全不只是使用安全工具就可以彻底解决，还必须制定一些策略，这样才能够确保在整个容器生命周期中落实合规要求。虽说大多数公司早已经制定了一套安全策略来管理应用开发，但也需要根据容器化应用的特定环境，修改安全策略，满足容器安全需求。

漏洞管理。在开发过程中，漏洞管理最重要的环节就是镜像扫描。不同的扫描工具，其扫描的粒度也不同。有些扫描工具能够发现操作系统漏洞和某些特定语言才会有的漏洞，而有些却无法扫描每个镜像层或某些开源软件包。

配置管理。在该阶段，DevOps 和安全团队可以采用 CIS 基准中针对 Docker 和 Kubernetes 的配置指南。在该阶段，团队可能仍会手动进行配置管理，但是自动执行配置检查的工具将改善安全状况并减少运营工作量。

3. 第 3 阶段：生产部署

到这个阶段，组织的第一个应用已投入生产运行。编排工具开始成为容器化应用生产部署的

一个关键部分。使用 Kubernetes、MESOS、Swarm 等工具可以自动执行与容器运行相关的大多数操作任务。以 Kubernetes 为例，当容器在 Kubernetes 中运行时，它们被打包在 Pod 单元中。Pod 可以包含一个或多个容器，但是同一 Pod 中的所有容器将共享相同的资源和本地网络。

虽然编排工具可以让容器应用更具弹性并易于扩展，但编排工具的使用也带来了其自身的威胁向量。编排工具本身就会引入漏洞，而且其配置也会对组织机构的整体安全状况产生很大影响。此外，编排工具的默认配置并不安全，如果忽略这些默认配置会带来巨大的安全风险。例如，Kubernetes 的 Kubeconfig 文件是攻击者获得对应用的初始访问的一种常用方法。

在此阶段，要实现容器和编排工具的安全性，团队必须注意事项：

扩展配置管理，在此阶段，还需要加强容器和编排工具的配置管理。对于容器，需要做到以下几点：

- 确保实现正确的容器配置，不会产生计划之外的特权用户
- 以非 root 用户身份运行容器
- 在容器中使用只读根文件系统
- 使用秘钥管理工具，而不要将秘钥存储在镜像中
- 调整容器的权限，确保访问限制尽可能严格
- 设置容器的内存和 CPU 限制，让其能够满足功能需要求，但不能再执行其他操作

对于编排工具（以 Kubernetes 为例）：

- 使用命名空间隔离资源
- 限制 Pod 之间的通信
- 使用 Pod 安全策略限制 Pod 可以实现的功能

设置运行时安全。应用投入生产后，必须能够检测到应用中出现异常行为，这可能是发生安全事件的前兆，包括：

- 容器中运行了哪些进程
- 是否有足够的信息来评估每个容器的风险并相应地确定修复的优先级
- 是否存在异常网络流量，是否存在策略太宽松，导致发生安全事件后影响范围过大

设置网络分段。如何管理网络分段将取决于具体应用，但是网络分段的原理没有变化。应允许每个 Pod 仅与执行其功能所需的内部或外部资源进行通信。

满足合规需求。合规要求取决于应用的功能以及所在的行业。

4. 第 4 阶段：快速扩容

在第一个应用投入生产取得成功后，组织机构就会将更多的应用容器化，然后投入生产。这时，容器化所涉及的工程师和团队的数量增加了。

在这一阶段，复杂性成为主要问题。复杂性不仅会给安全带来挑战，还会给事件响应和合规带来挑战。在该阶段，组织机构的治理策略对于确保开发、测试、部署和运行应用的标准工作流程至关重要。这些策略包括安全防护，确保在整个组织机构中统一安全处理，并最大程度地降低个人错误的风险。

在这一阶段，应主要注意以下方面：

自动化至关重要。随着公司处理的集群越来越多，手动进行配置管理或手动筛选原始事件数据来分析运行时行为，已不太可行。自动化变得尤其重要，通过自动化工具实现统一行安全策略是唯一可行方法。

更复杂的合规要求。在此阶段，重要的是要追踪，哪些应用或微服务要满足哪些合规要求，并通过合规检查来查看它们是否满足合规要求。

服务隔离和分段。随着服务数量的增加以及合规和安全生态系统变得越来越复杂，在服务之间进行适当的流量隔离和分段至关重要。

5. 第 5 阶段：在全组织机构范围内采用容器

这个阶段，所有新应用都在容器中开发，并且还可能将现有应用移入容器。组织机构将在基础结构和安全方面遇到新问题。在该阶段，大多数组织机构将考虑不只是使用编排工具来进行编排。他们将使用服务网格之类的技术来管理服务之间的交互方式，并了解各个服务的行为以及服务之间的通信。

随着将更多层（例如 Kubernetes 和服务网格）添加到堆栈中，有太多的按钮需要控制，已无法再用手动控制，因此自动化变得至关重要。在该阶段，优化和自动化至关重要。组织机构应不断寻找简化开发和运营的流程，简化与编排工具的交互界面，并实现重复性工作的自动化运行。

例如，容器和编排工具的默认配置一般不太安全，因此公司需要依靠自动化来确保，在公司范围内统一实行符合其标准和配置的安全规定。在第 5 阶段新加入的其他技术层也有一套针对他们自身的配置、隔离和漏洞管理的最佳实践办法。

在容器化进程中，这一阶段成熟度最高，所有新开发都是在容器中完成的，因此，每个人都会步入一个未知的世界，出现第二次技能差距。每次将新的层或技术添加到堆栈中时，都会产生新的学习曲线。同时，随着应用变得越来越复杂，新的威胁向量可能会出现。组织机构需要确保他们拥有的安全工具能够随着它们添加这些新技术，增强容器安全。因为随着应用控件的复杂性增加，可见性和自动配置管理的重要性成倍增加。

第 3 章　镜像安全检查清单

容器镜像是云原生环境中各类应用的标准交付格式。由于容器镜像需要大量分发和部署，因此，需要确保容器镜像在构建、分发和运行全生命周期内的安全。镜像扫描是检查操作系统和安装包中是否存在已知漏洞的一项基本措施。除此之外，还有很多措施可以加强容器镜像的安全。如下表所示，基于镜像安全 4 个阶段，11 个要求，28 项检查点，全面检测容器生命周期各个阶段的镜像风险，确保容器镜像的安全。

容器镜像安全检查清单

阶段	要求	描述
构建	构建/CI基础设施的安全	限制对构建基础设施的管理访问权限
		仅允许使用规定的网络入口
		仅授予完成必要工作所需的最低权限
		仔细审查第三方站点是否存在漏洞
		确保扫描工具的安全性和可靠性
	采用最小根镜像	对于第三方根镜像上，要考虑来源及其可信度、更新频率、默认安装的软件
		使用最小根镜像，比如Google Distroless、Red Hat通用根镜像或自己创建
	删除不必要的软件	将镜像限制为二进制文件、库和配置文件
		避免安装软件包管理器、Unix Shell、编译器和调试器
		创建临时调试镜像
	容器构建与运行时	不要想着对运行中的容器进行打补丁或更改容器
		使用多级Dockerfile，在运行时镜像以外的地方进行软件编译
	镜像中不要嵌入秘钥	切勿将任何秘钥嵌入镜像
		仅在运行时才提供敏感数据
		使用Kubernetes或其他秘钥管理系统
	镜像扫描	不安全的镜像不得推送到镜像仓库中去
		确保镜像扫描工具能够提供您所需的覆盖范围
		确定可接受的风险级别
分发	镜像仓库	使用内部的私有镜像仓库可最大限度地确保安全和配置
		谨慎管理镜像仓库的基础架构和访问控制
	镜像控制	对镜像添加不可变标签，防止对不同版本的镜像重复使用同一个标签
		通过镜像签名增强安全防护
运行	镜像扫描	使用自定义或第三方访问控制器来避免部署不安全的容器镜像
	镜像仓库	在客户端强制执行针对镜像仓库的保护措施
		部署一个Kubernetes访问控制器，验证Pod使用的是受信任的镜像仓库
维护	漏洞管理	根据漏洞严重性、数量、是否有可用的补丁或修补程序、是否影响部署，确定漏洞管理标准
		制定服务级别目标和处理这些镜像的流程
		制定一些程序，确定如何处理部署了有问题镜像的容器

1. 构建阶段

（1）基础设施的安全

为了从源头开始确保镜像安全，实现安全左移，需要确保镜像构建、CI/CD 基础设施的安全，防止将外部漏洞引入到镜像当中来。

下列措施有助于确保构建基础设施和管道的安全：

- 限制对构建镜像相关的基础设施的管理访问权限
- 仅允许使用规定的网络入口
- 谨慎管理各类秘钥，并且仅授予完成必要工作所需的最低权限
- 从第三方站点拉取源文件或其它文件时，仔细审查文件是否存在漏洞，可使用白名单，仅允许受信站点访问
- 确保扫描工具的安全性和可靠性，才能获得可靠的扫描结果

（2）根镜像

在现有的第三方根镜像上构建镜像，则需要考虑到以下因素：

- **根镜像来源和托管机构的可信度。**镜像是来自知名公司还是开源组织？其维护的镜像仓库是否拥有良好的信誉？镜像中所有组件的 Dockerfile 和源代码是否可用？
- **更新频率。**避免使用不经常更新的镜像，尤其是不对相关漏洞做出披露的镜像。
- **默认安装的软件。**从最基本的根镜像开始，有选择性地安装应用所需的工具和库，这要比从现成镜像中找出需删除的软件包，更安全可靠。

下面是一些常见选择根镜像的方案：

- Google Distroless——Google 针对几种常见语言预先构建的最小根镜像。镜像中没有软件包安装程序，如果需要其他软件，可以将文件复制到镜像中。
- Red Hat 通用根镜像（UBI）——基于 RHEL 的镜像。其镜像分为三层——最小根镜像、标准平台和多服务。它还支持多种语言的镜像。
- 自己从头开始构建根镜像。

（3）删除不必要的软件

如果恶意攻击者确实访问了运行中的容器，那么也该确保攻击者进入容器后，容器内不会有其它地方存在漏洞成为黑客进一步攻击的抓手。使用最小根镜像就是一个很好的入手点，但是如果 Dockerfiles 安装了很多工具，就不如使用最小根镜像那么安全了。使用最小镜像的另一个好处就是，可以减少遇到 0day 漏洞的可能性，从而减少了对镜像打补丁和进行维护的需求，也加快了镜像的存储和拉取速度。

将镜像限制为二进制文件、库和配置文件，可提供最佳保护。特别是，尽量要避免安装以下工具：

- 软件包管理器：例如 apt、yum、apk
- Unix shell：例如 sh、bash。删除 shell 程序后，在运行时无法再使用 shell 脚本，但可以使用编译语言。
- 编译器和调试器。只有在构建和开发容器时，才使用编译器和调试器，而在生产容器中则绝不使用。

如果在生产镜像中需要安装这些工具进行应用调试，建议创建临时调试镜像。Kubernetes 目前也支持临时容器，可以将容器放置在现有的 pod 中进行调试。

（4）容器的构建

在生产系统上运行时，用于生成和编译应用的构建工具会被黑客利用。应该将容器视为短暂存在的临时实体。不要想着对运行中的容器进行打补丁或更改容器。只能通过构建一个新镜像，然后替换过时的容器部署。使用多级 Dockerfile，在运行时镜像以外的地方进行软件编译。

（5）秘钥

切勿将任何秘钥嵌入镜像，即便只是供内部使用，也不建议这样做，因为任何能够拉取镜像

的人都可以提取秘钥。秘钥包括 TLS 证书秘钥、云提供商的凭据、SSH 私钥和数据库密码等等。只在运行时提供敏感数据，这还可确保能够在不同的运行时环境中使用同一镜像。这样，无需重新构建镜像，就能够简化更新过期秘钥或吊销秘钥的流程。

不使用嵌入式秘钥时，还可以向 Kubernetes Pod 提供秘钥作为 Kubernetes 秘钥，或使用其他秘钥管理系统。

（6）镜像扫描

若镜像中所含的软件存在安全漏洞，则容器在运行时就更容易受到攻击。因此，通过 CI/CD 构建镜像时，镜像扫描是一个必然要求。不安全的镜像不能推送到镜像仓库中去，以免在生产中误用了存在漏洞的镜像。

目前，有一些开源的镜像扫描工具可供使用，但它们提供的覆盖范围不尽相同。有些扫描工具只是扫描已安装的操作系统软件包，有些会扫描已安装的运行时存储库，而另一些则可能会提供二进制指纹验证或文件内容扫描。

具体选择什么样的扫描工具，取决于所需要扫描的范围（编程语言等），以及对漏洞风险的接受度。对于那些安全要求高的企业，选择商用的镜像扫描，例如青藤蜂巢产品，可以实现更高质量镜像扫描。

2. 分发阶段

（1）选择镜像仓库

构建好容器镜像后，需要将其进行存储。使用内部的私有镜像仓库可最大限度地确保安全和配置，但是也需要谨慎管理镜像仓库的基础架构和访问控制。大多数云提供商还提供托管镜像仓库服务，其通常会自带云访问管理功能。

使用镜像仓库，可以减轻许多管理开销。安全工程师和开发人员将需要根据其安全要求和基础架构资源，为其组织选择最佳解决方案。

（2）镜像控制

镜像仓库支持对镜像添加不可变标签，防止对不同版本的镜像重复使用同一个标签，从而能够执行确定的镜像运行时。例如，要想能够准确地知道要在什么时间，部署某个应用的、某个镜像的、某个版本，在这种情况下，如果每个镜像都标记“最新”标签，就会造成混乱。

镜像签名也可以增强安全防护。通过镜像签名，镜像仓库会生成标记镜像内容的校验和，然后使用私钥来创建含有镜像元数据的加密签名。客户端仍然可以在不验证签名的情况下，拉取并运行镜像，但是在安全环境中，运行时应支持镜像验证要求。镜像验证使用签名密钥的公钥，来解密镜像签名的内容，然后可以将其与提取的镜像进行比较，以确保镜像内容未被篡改。

3. 运行阶段

（1）镜像扫描

在镜像构建时进行镜像扫描，并不意味着运行时就不需要镜像扫描了。相反，对于可能使用的任何第三方镜像和自己的镜像（其中可能包含新发现的安全漏洞），在运行时进行镜像扫描更为重要。可以在 Kubernetes 集群中使用自定义或第三方访问控制器来避免部署不安全的容器镜像。

尽管某些扫描工具支持将扫描结果存储在数据库或缓存中，但用户需要权衡信息失效和每次拉取镜像进行扫描所引起的延迟。

（2）镜像仓库和镜像信任

在“分发”部分，我们讨论了选择镜像仓库的标准以及某些镜像仓库支持的一些其他安全功能。尽管确保镜像配置正确，且使用了安全的镜像仓库很重要，但是如果不在客户端强制执行这些保护措施，安全依然无法得到保障。

Kubernetes 本身并不提供安全镜像的拉取服务。这就需要部署一个 Kubernetes 访问控制器，验证 Pod 使用的是受信任的镜像仓库。为了支持签名镜像，控制器需要能够验证镜像的签名。

4. 后期维护

在容器的全生命周期内，镜像扫描是至关重要的，这时就需要权衡组织机构的风险承受能力和交付速度。组织机构需要制定对应策略和流程来处理镜像安全和漏洞管理。

对于镜像中存在的漏洞，应根据以下指标来确定漏洞管理标准：

- 漏洞严重性
- 漏洞数量
- 这些漏洞是否具有可用的补丁或修补程序
- 漏洞是否影响配置错误的镜像的部署

基于漏洞严重性、漏洞数量、是否具有补丁等标准，首先，可以帮助用户决定是否允许存在问题（不严重）镜像进行部署上线。其次，发现新的漏洞后，是否要阻止使用现有镜像进行部署，最后，确定对于已经部署了有问题镜像的容器，该如何处理。

第 4 章　安全使用 Kubernetes

近年来，以 Kubernetes 为代表的安全编排工具让企业实现了应用的自动化部署，给企业带来了巨大的业务收益。但是，和传统环境下一样，这些部署也很容易受到黑客和内鬼的攻击和利用，Kubernetes 的安全也因此成为容器使用中重点保护对象。

2018 年黑客入侵了特斯拉在亚马逊上的 Kubernetes 容器集群。由于该集群控制台未设置密码保护，黑客便得以在一个 Kubernetes pod 中获取到访问凭证，然后据此访问其网络存储桶 S3，通过 S3 获取到了一些敏感数据，比如遥测技术，并且还在特斯拉的 Kubernetes pod 中进行挖矿。

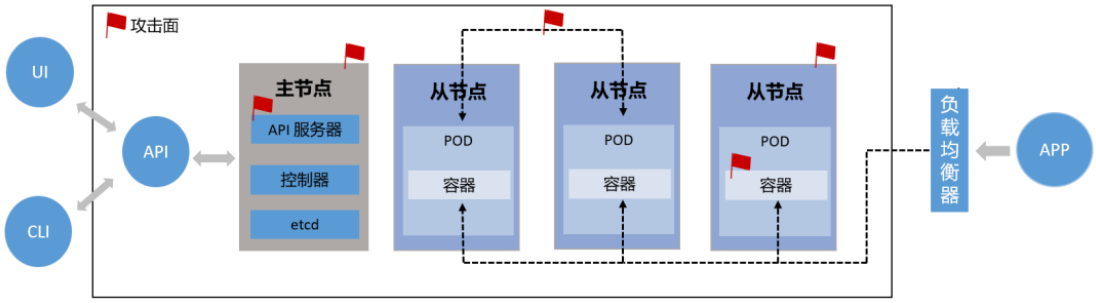
在我们正式讨论 Kubernetes 安全之前，首先让我们来了解一下 Kubernetes 的工作流程。

1. Kubernetes 工作流程

Kubernetes 是一个容器编排工具，可自动执行容器的部署、更新和监控。所有主要的容器管理和云平台（例如 Red Hat OpenShift、Docker EE、Rancher、IBM Cloud、AWS EKS、Azure、SUSE CaaS 和 Google Cloud）都支持 Kubernetes。以下是有关 Kubernetes 的一些基础知识：

- **主节点。**主节点服务器管理着 Kubernetes 工作节点集群，并在工作节点上部署 Pod。
- **从节点。**也称为工作节点，通常运行应用容器和其他 Kubernetes 组件，例如 agent。
- **Pod。**Pod 是 Kubernetes 中的部署和可寻址单位。每个 pod 都具有自己的 IP 地址，并且可以包含一个或多个容器（通常是一个）。
- **服务。**服务充当底层 Pod 的代理，并且可以对不同的 Pod 进行负载均衡。

系统组件。用于管理 Kubernetes 集群的关键组件包括 API 服务器、Kubelet 和 etcd。这些组件都是潜在的攻击目标。在上文提到的特斯拉事件中，黑客就是攻击了没有密码保护的 Kubernetes 控制台来安装加密挖矿软件。



在 Kubernetes 网络连接中，每个 Pod 都有自己的可路由 IP 地址。Kubernetes 的网络插件负责将主机之间的所有请求从内部路由到相应的 Pod。可以通过服务、负载均衡或入口控制器来提供对 Kubernetes pod 的外部访问请求，Kubernetes 会将其路由到适当的 pod。

Pod 通过这些叠加网络，也就是“包内之包”相互通信，并进行负载均衡和 DNAT 来与相应的 Pod 进行连接。可以使用适当的报头对数据包进行封装，将它们发送到相应的目的地，然后在目的地解除封装。

Kubernetes 会动态处理所有这些叠加网络，因此，监控网络流量非常困难，确保网络安全更是难上加难。

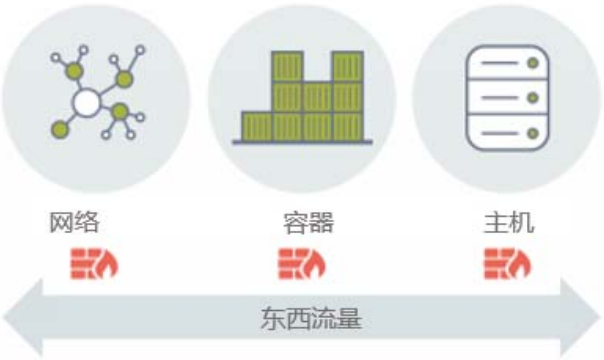
对 Pod 中运行的容器进行攻击，既可以通过网络从外部进行，也可以由内鬼在内部进行，例如，黑客通过网络钓鱼攻击，让受害者的系统成为内部攻击的跳板。针对 Kubernetes 的漏洞和攻击向量包括：

- **容器失陷。**若存在应用配置错误或漏洞，攻击者就能够进入容器中，探测网络、进程控件或文件系统中的弱点。
- **Pod 之间未经授权的连接。**失陷容器可能会与同一主机或其他主机上的其他正在运行的 Pod 进行连接，以进行探测或发动攻击。尽管第 3 层网络控件可以将 Pod IP 地址列入白名单，提供了一些保护，但是只有通过第 7 层网络过滤才能检测到对受信任 IP 地址的攻击。

- **Pod 的数据渗透。**数据窃取通常是使用多种技术组合来完成的，包括在 pod 中进行反弹 shell 连接，连接到 CC 服务器和网络隧道，从而隐藏机密数据。

2. 实时增强 Kubernetes 的运行时安全

在容器投入生产运行后，保护 Kubernetes 安全最重要的三个安全向量分别是：主机安全、容器检查和网络过滤。



主机安全

如果容器运行所在的主机（例如 Kubernetes 工作节点）遭到入侵，则可能会导致：

- 通过提权，实现 root 用户权限
- 窃取用于访问安全应用或基础结构的秘钥
- 更改集群管理员权限
- 主机资源损坏或劫持（例如挖矿软件）
- 影响关键编排工具基础架构，例如 API Server 或 Docker 守护程序

像容器一样，需要对主机系统的可疑活动进行监控。因为容器可以像主机一样运行操作系统和应用，所以，需要像监视容器进程和文件系统活动一样监控主机。总之，综合进行网络检查、容器检查和主机安全，形成了从多个向量来检测 Kubernetes 攻击的最佳方法。

容器检查

攻击经常利用提权和恶意进程来实施或传播攻击。Linux 内核（如 Dirty Cow）、软件包、库或应用程序本身的漏洞利用，都可能导致容器被攻击。

检查容器进程和文件系统活动并检测可疑行为是确保容器安全的一个关键要素。应该检测所有可疑进程，例如端口扫描和反向连接或提权。内置检测和基线行为学习过程应结合在一起，可以根据以前的活动识别异常进程。

如果容器化应用是根据微服务原则设计的，容器中的每个应用功能有限，并且是使用所需的软件包和库来构建的，那么，检测可疑进程和文件系统活动将更加容易和准确。

网络过滤

容器防火墙是一种新类型的网络安全产品，可以将传统的网络安全技术应用到新的云原生 Kubernetes 环境中。有不同的方式可以确保容器网络的安全：

- 基于 IP 地址和端口的 3/4 层过滤。该方法需要制定 Kubernetes 网络策略，以动态方式更新规则，在容器部署发生变更和扩容时提供保护。简单的网络隔离规则并不能提供关键业务容器部署所需的强大监控、日志记录和威胁检测功能，仅提供针对未授权连接的某种保护。
- Web 应用程序防火墙（WAF）攻击检测可以使用检测常见攻击的方法来保护面向 Web 的容器（通常是基于 HTTP 的应用）。但是，这种保护仅限于通过 HTTP 进行的外部攻击，而且还缺少内部流量通常所需的多协议过滤。
- 第 7 层容器防火墙。具有第 7 层过滤功能和 pod 间流量深度数据包检查功能的容器防火墙可使用网络应用协议保护容器。这是基于应用程序协议白名单以及对基于网络的常见应用程序攻击（例如 DDoS、DNS 和 SQL 注入）的内置检测。容器防火墙还有一个独特功能，可以将容器进程监控和主机安全纳入监控的威胁向量中。

深度数据包检测（DPI）技术对于容器防火墙中的深度网络安全至关重要。漏洞利用通常使用可预测的攻击向量：恶意 HTTP 请求，或在 XML 对象中包含可执行 shell 命令。基于第 7 层 DPI 的检测可以识别这些方法。使用这些技术的容器防火墙可以确定是否应允许每个 Pod 连接通过，或者确定是否是应该拦截的潜在攻击。

考虑到容器和 Kubernetes 联网模型的动态性，传统的用于网络可见性、取证分析的工具可能就不再适用了。简单的任务（例如进行数据包捕获，用于调试应用或调查安全事件）也不再那么简单。需要新的 Kubernetes 和容器感知工具来执行网络安全、检查和取证任务。

确保 Kubernetes 系统与资源的安全

如果不对 Kubernetes 进行安全保护，Kubernetes 以及基于 Kubernetes 的管理平台可能很容易受到攻击。这些漏洞暴露了容器部署的新攻击面，很可能被黑客利用。为了保护 Kubernetes 和管理平台自身不受攻击，需要做的一个基本步骤就是正确配置 RBAC，确保用户获得适当的系统资源。同时，还需要审查和配置以下方面的访问控制权限。

- 保护 API 服务器。为 API 服务器配置 RBAC 或手动创建防火墙规则，防止未经授权的访问。
- 限制 Kubelet 权限。为 Kubelet 配置 RBAC，并管理证书轮换以保护 Kubelet。
- 要求对所有外部端口进行身份验证。查看所有可从外部访问的端口，并删除不必要的端口。需要对所需的外部端口进行身份验证。对于未经身份验证的服务，则仅限白名单访问。
- 限制或删除控制台访问。除非通过正确配置，可以让用户通过强密码或双因素身份认证进行登录，否则不允许访问控制台/代理。

如前所述，结合功能强大的主机安全，锁定工作节点，可以保护 Kubernetes 部署基本架构免受攻击。但是，还建议使用监控工具来跟踪对基础结构服务的访问，检测未经授权的连接和潜在攻击。

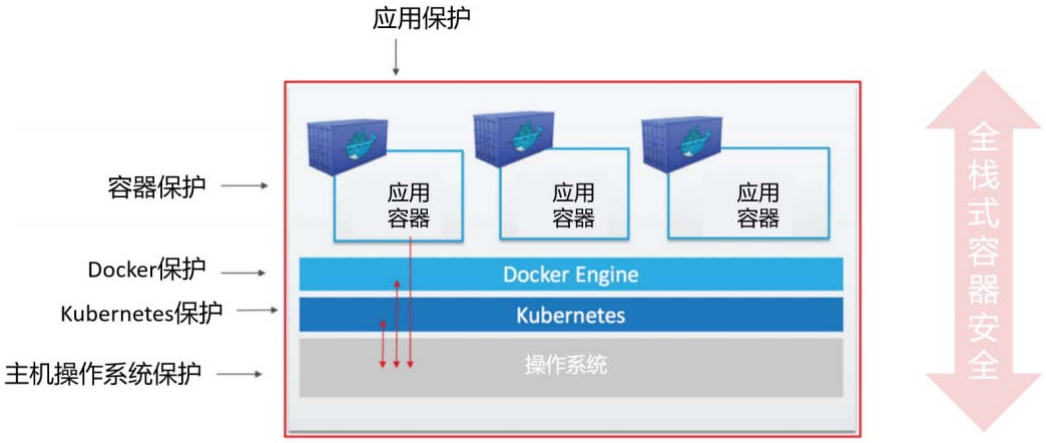
我们仍然以特斯拉 Kubernetes 控制台漏洞利用为例。在黑客攻陷了工作节点后，就会建立一个外部连接，控制加密货币挖矿软件。对容器、主机、网络和系统资源进行基于策略的实时监控，可以检测到可疑进程以及未经授权的外部连接。

第 5 章　全面打造一站式容器安全解决方案

由于容器底层的基础架构非常复杂，稍有不慎，就可能会带来严重的安全后果。因此，在享受容器带来便捷性优势的同时，也要确保容器的安全，这是一个持续的过程。像其他测试和质量控制措施一样，应该将安全集成到开发过程中来，尽可能实现自动化，减少手动处理的工作量，同时，安全还应该扩展到底层基础架构的运行与维护当中来。

青藤蜂巢·容器安全平台，在对容器安全进行深入研究之后，正式提出了容器安全“五个保护”：

- 保护容器主机的安全
- 保护容器网络流量安全
- 保护容器中应用的安全
- 保护容器管理技术堆栈
- 保护构建管道的完整性



1. 保护容器主机的安全

要保护主机安全，首先要保护主机操作系统。应尽可能使用经过优化的分布式操作系统来运行容器，例如 Container Linux、RancherOS、Ubuntu Core、Google 的 Container-Optimized OS 或 RedHat Atomic 等发行版本。这些发行版本都已经删除了可有可无的服务，优化了性能并减少了攻击面。如果使用常规 Linux 发行版或 Windows 系统，则通常需要禁用或删除不必要的服务并加固操作系统。

然后添加主机安全产品，确保主机按预期状况运行，实时监控主机风险状况包括漏洞、不合规配置、弱密码等等，还需要实时进行入侵检测，在入侵行为尚未造成危害之前及时发现和响应。

2. 监控容器网络流量安全

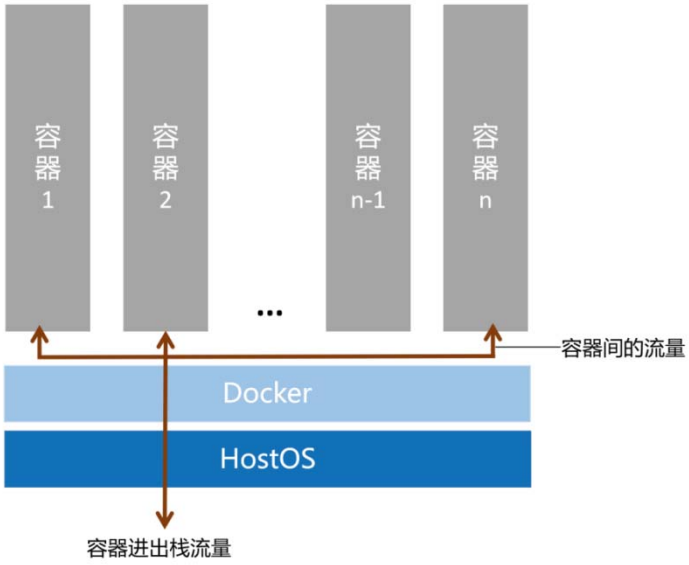
容器投入生产后，需要与其他容器和资源进行交互。进入网络的流量通常会受到其他安全控

制措施的保护。部署在网络边界处的 Web 网关和 IPS 具有一定的过滤功能，可以防止发生直接攻击或尝试进行水坑攻击（先攻击某项服务，然后攻击其访问者）。

关于容器流量的另一项安全挑战是监控和保护容器之间的流量。攻击者在系统中找到一个立足点后，就会迅速进行横向行动，扩大影响范围。比如主机 A 的 Apache 向主机 B 的 MongoDB 请求数据；同时也有可能是主机内部容器之间的流量，比如主机 A 内部的 Apache 服务器请求主机 A 内部的 MongoDB 的流量。

为应对这类威胁，目前有两种入侵防御方案，一种是在网络边界处部署 IPS 设备，另一种是以软件形式在物理主机上部署入侵防御软件。但是部署在边界处的 IPS 无法监控主机内部容器之间的流量，部署在主机内部 IPS 软件，也无法对跨主机的容器之间的流量进行入侵防御。

针对容器东西向流量入侵防御，以 docker 为例，目前有一种方式是在物理主机内部连接容器的网桥上设置 Netfilter 钩子，利用钩子截获所有进出容器的东西向流量。对截获的东西向流量进行清洗，判断所述东西向流量中的报文是否属于恶意流量。若属于恶意流量，则阻止相关连接。



3. 保护容器管理技术堆栈

在容器部署问题，通常会使用两个关键基础设施，即私有容器镜像仓库和容器编排工具 Kubernetes。

容器镜像仓库是用于搜索、存储和分发容器的，简化了容器的共享工作，并帮助团队充分利用彼此的工作成果，提高了工作效率。为了确保每个容器都符合开发和安全基准，需要对容器镜像实时进行漏洞扫描。在将容器镜像推送到镜像仓库中去之前，要先对容器镜像进行漏洞扫描，查看是否存在已知漏洞、恶意软件以及明文秘钥，这样有助于减少下游的安全问题。此外，还需要确保镜像仓库本身也采取了良好的保护措施。镜像仓库应该运行在加固系统或信誉良好的云服务之上。

Kubernetes 是一款复杂的容器编排工具，让企业实现了应用的自动化部署，给企业带来了巨大的业务收益。从安全角度来看，Kubernetes 还提供了实现许多运营和安全控制的功能。实

施 Pod（集群级别的资源）和网络安全策略后，可以采用不同方案来满足风险承受能力。

4. 保护容器中的应用安全

在构建一款应用时，开发人员很少会从头开始。通常的做法是从一个现有容器着手，例如像 Ubuntu 或 CentOS 这样的低层容器，或者像 node 或 nginx 这样的更高层容器，或者是组织机构自定义的容器。不管选择在哪个容器上构建数据，关键是要确保构建应用的底层容器基础是安全的，不会给数据造成风险。

从首选镜像仓库中拉取镜像部署容器时，也是如此。需要有一个相应的工作流程，确保使用的容器安全可靠，可抵御常见威胁。为此，需要进行容器镜像扫描，扫描容器的内容，查看是否存在任何问题，然后再用于构建应用。在投入生产之前，还要进行最后的安全检查。

进行镜像扫描主要是为了查找恶意软件、已知漏洞以及任何秘钥或机密信息（例如应用程序编程接口（API）密钥和令牌），以便在继续开发应用之前降低风险。这一步可确保在安全容器基础之上构建应用。

镜像扫描功能不仅限于确定使用的容器基础是安全可靠的，还应该成为编码流程不可或缺的一部分。这是一个完全自动化的过程，可以快速轻松地识别开发应用和容器时遇到的任何问题。在开发过程的早期阶段捕获漏洞、恶意软件或明文秘钥之类的问题，更易于解决并节省您的时间。

虽然一旦将应用构建到容器中并部署到生产环境中，就不会再发生变化。但是，容器投入生产运行后，是会实时发生变化的。容器会产生应用数据，生成日志文件、缓存文件等等。

安全监控软件，包括有助于确保在容器中开展的活动中是否存在一些风险。例如，如果存在漏洞，则安全引擎可以检测是否有人在尝试进行漏洞利用，并通过丢弃数据包来保护应用。

5. 确保构建管道的完整性

除了容器环境之外，容器构建管道也日益成为攻击者的目标。攻击者已开始将攻击转移到持续集成/持续交付的早期阶段（CI/CD）管道。如果攻击者攻陷了容器构建服务器、代码存储库或开发人员工作站，那么攻击者就可以在组织机构的环境中驻留很长时间。因为大多数安全程序不会主动监控这些关键资源。

对此，第一步是确保这些系统具有强大的安全控制措施，并进行随时更新。组织机构的代码通常是其最具价值的资产之一，因此，需要保护代码的安全。

第二步是在整个管道中实施强大的访问控制策略。这可以从代码存储库和分支策略着手，一直扩展到容器存储库。这需要确保实施最小特权原则（仅提供完成任务所需的访问权限）并定期审核该访问情况。

客户案例：物流企业如何守护云上的“集装箱”安全？

你知道吗？集装箱的发明曾经改变了这个世界，因为集装箱让物流运输实现了标准化，大大降低了运输成本。如今，全球 90%的货物都是通过集装箱运输到世界各地的。《经济学人》杂志曾指出：如果没有集装箱，就不会有全球化。对于这一点，作为物流企业的 A 公司有

着深刻体会。

而容器，就好比是 IT 技术领域的集装箱。有人断言，容器的广泛应用也将改变各个行业，甚至改变世界。

有数据表明，企业对容器的采用率正在逐渐上升。Gartner 在 2019 年发布的一份报告显示，到 2023 年，全球将有 70%以上的企业采用容器化应用，而且，这些企业将在生产中运行 3 个以上的容器化应用。

1. 容器带来的六大好处

据了解，A 公司的云原生已经规模化落地，而且采用了容器，应用了 kubernetes 编排技术。IT 架构师张晓丹（曾担任久游网、聚尚网等企业技术负责人）向记者介绍了容器带来的 6 大好处：

其一，部署方便。

过去 IT 团队进行编程时，往往要耗费几个小时搭建环境，而且出现问题时总是需要很久才能解决，还得向其他成员求助。有了容器之后，这些都变得非常容易，开发环境只是一个或几个容器镜像的地址，最多再需要一个控制部署流程的执行脚本，或者进一步将环境镜像以及镜像脚本放入一个 git 项目，发布到云端，需要的时候再将它拉到本地。

"目前使用主流容器技术的团队都是用这种方案搭建本地开发环境，并将其整理成相关标准，慢慢沉淀成关于容器技术的财富了。"

其二，部署安全。

过去研发人员收到 bug 反馈时，心里的第一反应都是"在我本地是好的啊！"其实导致这种情况的是环境不一致，在开发过程中的调试往往不能保证在其他环境里也没有问题。有了容器以后，可以通过容器技术让开发环境、测试环境以及生产环境保持版本上的统一，保证代码在一个高度统一的环境中执行，这样也能解决 CI 流程对环境的要求。

"在分布式技术和扩容需求日益增长的今天，如果运维人员能使用容器技术进行环境部署，不仅能节省部署时间，还能把很多因人工配置环境产生的失误降到最低。"

其三，隔离性好。

无论是用于开发还是生产，一台机器上往往需要跑多个服务，而且服务各自需要的依赖配置不尽相同，如果两个应用需要使用同一个依赖，或者两个应用需要的依赖之间有一些冲突，就很容易出现问题，所以最好把同一台服务器上不同应用提供的不同服务隔离起来。而容器在这方面有天生的优势，每一个容器就是一个隔离环境，对容器内部提供服务的要求，容器可以自依赖的全部提供。这种高内聚的表现可以快速分离有问题的服务，在复杂系统中实现快速排错和及时处理。

其四，能快速回滚。

在容器之前的回滚机制，一般需要基于上个版本的应用重新部署，替换问题版本；在最初时

是用一套完整的从开发到部署的流程，而执行这一套流程往往需要很长时间；在基于 git 的环境中，需要回退某个历史提交，然后重新部署。与容器技术相比，这些方式都不够快，而且可能会引起新的问题。容器技术天生带有回滚属性，对每个历史或镜像都会保存，而替换容器的某个历史镜像是非常快速简单的。

其五，使用成本低。

张晓丹表示，"这是最明显和有用的优点了。"在容器出现之前，构筑一个应用往往需要一台新的服务器或一台虚机，而服务器的购置成本和运维成本都很高，虚机则需要占用很多不必要的资源。相比之下，容器技术就小巧轻便得多，只要给一个容器内部构建应用需要的依赖就可以了，这也是容器技术发展迅速的最主要原因。

其六，管理成本更低。

随着容器技术不断普及和发展，容器管理和编排技术也同样得到发展。如 Docker Swarm、Kubernetes、Mesos 等编排工具也在不断迭代更新，这让容器技术在生产环境中拥有了更多的可能性和更大的发挥空间。随着大环境的发展，Docker 等容器的使用和学习成本也在降低，成为更多开发者和企业的选择。

简而言之，通过"云化"和"容器化"，可实现快速构建和维护新服务、新应用，达到降本增效和信息资源整合的目标。

2. 保障容器安全的三大策略

但是，容器本身也存在着重大的安全风险，如不安全的镜像、容器逃逸攻击、运行环境安全问题、编排安全问题等等，这意味着保护容器安全也是一项持续的挑战。针对这些挑战，传统的防火墙、漏扫类安全产品，并不能解决容器内东西向流量问题，更无法对容器分层镜像进行扫描。据 Gartner 统计，2019 年，容器安全产品的覆盖率只占 5%-20%。

对此，张晓丹表示，目前很多技术团队对容器安全技术的了解都不是很透彻，市场也未能提供丰富的技术手段，但是容器安全领域处于蓬勃的发展期，新技术总是要经历这样的过程才会逐步成熟。

在他看来，保护容器安全在未来将会出现三种主要策略：

第一，安全策略即代码

Kubernetes ConfigMaps 和自定义资源（CRD）等工具将把安全产品、配置和规则自动化到 CI/CD 和 DevOps 环境中。DevOps 团队可以通过分析应用程序、应用行为，在标准的 YAML 文件中声明所有新的工作负载都要部署安全策略，从而使安全过程高效集成且自动化。传统的安全团队也可以使用相同的工具，将全球安全策略注入到整个环境中，让企业体验到来自云原生的现代化安全。

第二，安全网格化服务

越来越多的企业开始在服务体系架构中增加安全网格化功能，来阻止潜在的网络攻击风险，让应用程序具备可感知能力。当黑客绕过传统网络和主机安全技术，开始攻击容器编排解决

方案，必要的安全保障措施也要及时跟上。因为即时和自动化的安全情报响应体系是解决 Kubernetes 和容器 API 漏洞的最有利措施，也是解决黑客攻击的必要手段。

第三，容器安全动态调整

很多企业在容器开发后才通过安全措施来加固，防范风险，比如：寻找攻击的薄弱点，通过环境模拟来寻找未知的漏洞。而有些安全意识比较强的企业会在容器开发之初就格外重视安全问题，为了确保云迁移万无一失，必须把安全策略贯穿于容器编排平台的整个生命周期，包括构建之初，平台运行过程中，此外也要关注运行之后的变化。

3. 企业的容器安全之选

信息安全专家任祖森（曾担任起亚、吉利等知名汽车主机厂信息安全负责人）指出，为确保网络安全，公司有必要建立一套网络安全体系，而网络安全体系是一项复杂的系统工程，需要把安全组织体系、安全技术体系和安全管理体系等手段进行有机融合，构建一体化的整体安全屏障。目前网络安全防护有几个比较经典的架构，包括 PDRR 模型、P2DR 模型、IATF 框架和黄金标准框架。

理论模型--网络安全体系构建--演习实战的验证，会让网络空间安全体系不断优化运行。演习覆盖了演习规划、方案设计、实战和演习后优化等步骤，以此发现网络安全体系在防御、检测、响应、修复环节中的问题和待优化的地方，验证有效性。

任祖森进一步表示，在选择容器安全产品时，要从容器安全的全生命周期防护和持续的监控与分析功能考虑，并形成闭环。

在全生命周期防护方面，平台在容器和 kubernetes 的安全性方面建立了标准，可以在容器应用程序的整个生命周期（构建、分发、运行）中保护容器环境安全。

在持续的监控与分析方面，要持续性监测容器的安全状况，可视化展现企业的风险场景。为安全决策者动态展示企业容器环境安全指标变化、安全走势分析，使容器安全清晰可衡量。

为此，A 公司使用了青藤蜂巢 • 容器安全平台。该平台专注于容器安全领域，能提供强大的实时监控和响应能力，帮助企业发现和解决风险，保障企业的容器环境安全。

据任祖森介绍，使用青藤蜂巢 • 容器安全平台后，在以下方面得到了加强：

首先是资产清点：在发生安全事件时，能全面及时的进行资产数据支持，大大缩短了排查问题的时间周期，减少了企业损失。此外，资产信息与补丁、入侵功能协同联动，能帮助用户快速定位问题容器，并及时查看到容器内的进程、应用详情信息。

其次，安全补丁解决了 Docker 补丁管理难的问题：通过建立一个智能应用补丁扫描工具，可以为安全运维人员提供补丁管理、补丁检测以及自动化补丁修复建议。

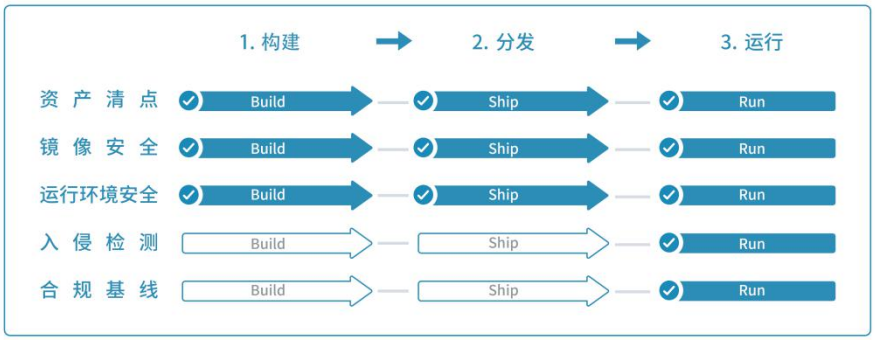
第三，实现了入侵检测：视角从了解黑客的攻击方式，转化成对内在指标的持续监控和分析，无论多么高级的黑客，其攻击行为都会触发内部指标的异常变化，从而被迅速发现并处理。

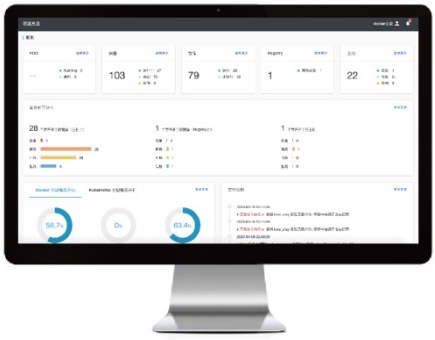
第四是达到了合规基线：紧跟监管政策，不断更新等级保护、CIS 标准对应的基线。可以一键进行自动化检测，提供可视化基线检查结果和代码级的修复建议。

张晓丹向记者表示，"青藤蜂巢•容器安全产品提供了精准、高效、可扩展的主机安全产品和专业服务；以服务器安全为核心，构建了基于业务端的安全联动平台，为企业提供了稳固、持续性的安全防护，保障了容器对业务的快速响应。"

关于青藤蜂巢 • 容器安全平台

青藤蜂巢 •容器安全平台是青藤经过多年研究潜心打造的一款容器安全产品。青藤蜂巢通过对容器安全状况的持续监控与分析，可视化展现企业风险场景，为安全决策者动态展示企业容器环境安全指标变化、安全走势分析，在容器应用的整个生命周期（构建、分发、运行）内保护容器环境安全，实现了安全预测、防御、检测和响应的安全闭环。青藤蜂巢主要提供容器资产清点、镜像安全、运行时安全、合规基线等功能。





现在就去试一试吧？

点击下方按钮，即刻部署青藤蜂巢，为您的容器增加一层防弹衣，让您的容器不再裸奔。

申请试用