QINGTENG
青藤云安全

云原生安全白皮书

目录

执行摘要

目标

当前，采用云原生的开发和部署模式已日渐成为技术行业的发展趋势。同时，整个生态系统，包括技术、产品、标准和解决方案，也正在不断扩展，要求决策者要时刻了解复杂的设计。特别是CISO，要在这个动态发展的领域，实践业务价值。同时，云原生模式还鼓励改变消费模式，并采用现代化工作流（例如敏捷方法和 DevOps 流程），这就要求将安全集成进来。

问题分析

由于传统的基于边界的安全已不再可行，同时由于云原生明确关注快速开发与部署，在这种情况下，安全问题也就变得很复杂。这就要求转变安全范式来保护应用，将基于边界的安全方式转变为更接近动态工作负载的安全方式。动态工作负载是基于属性和元数据（例如标签）来确定的。通过这种方法，可以识别并保护工作负载，满足云原生应用的规模化需求，同时还能适应不断变化的需求。

安全范式的转变要求提高在应用安全生命周期中的自动化程度，并通过设计架构（例如零信任）来确保其安全。将容器化作为云原生环境中的一项核心变化，也需要新的最佳实践。容器化的安全实施会涉及组织机构内的多个相关方，并且会对追求业务目标的开发人员和运维人员的工作效率产生重大影响。云原生应用仍然需要开发、分发、部署和运维，但是这种范式就决定了，要有效实现这些目标，需要新的安全机制。

可以在应用生命周期的不同阶段（“开发”、“分发”、“部署”和“运行时”），对云原生开发进行建模。云原生安全与传统安全方法的不同之处在于，可以确保在生命周期的不同阶段将安全嵌入进来，而不是通过单独管理的安全干预措施来确保生命周期的安全。

要注意，如果不对这些概念、工具和流程的使用和集成进行长期的教育和培训，云原生的落地和应用可能就会难以难以为继，甚至被打回原形。

全生命周期的不同阶段

开发阶段

云原生工具旨在在应用生命周期的早期阶段引入安全。安全测试需要及早发现不合规情况和配置错误，以便缩短可行性反馈的周期，进行持续改进。这种方法可以确保，可以按照针对管道中其他问题而提出的类似工作流（例如 bug 修复或 CI 故障）解决安全故障问题。通常，需要先解决好这些安全问题，才能推动软件在管道中的进一步操作。

这种模式的现代化安全生命周期是围绕着代码开发而展开的，代码开发遵循的是推荐的设计模式（例如 12-Factor），并确保了所交付工作负载的完整性。从概念上来讲，云原生与基础架构即代码（IaC）密切相关，旨在确保通过早期安全检查集成，让控件能够按预期状况运行。通过这些控件和集成可以尽早识别错误配置并实施 IaC 和编排清单中的最佳实践，以减少长期成本并提高安全价值。

分发阶段

在支持软件快速迭代的模型中，软件供应链安全尤其重要。云原生应用生命周期需要采用一些方法，不仅可以验证工作负载本身的完整性，还可以验证工作负载的创建过程和运维方式。加之需要一直使用开源软件和第三方运行时镜像（包括上游依赖项的层），面临的挑战就更大了。

生命周期管道中存在的工件（例如容器镜像）需要进行连续的自动扫描和更新，确保免受漏洞、恶意软件、不安全编码方法和其他不当行为的侵害。完成这些检查后，重要的是对工件进行加密签名以确保完整性并强制执行不可否认性。

部署阶段

在整个开发和分发阶段集成了安全后，可以实现实时、持续验证候选工作负载属性（例如，验证签名的工件，确保容器镜像和运行时安全策略以及验证主机的适用性）。与工作负载一起部署安全工作负载的可观测性功能，可以以高度信任的方式监控日志和可用指标，对集成安全进行补充。

运行时阶段

预计云原生环境在设计时就会提供策略实施和资源限制功能。运行时工作负载的资源限制（例如 Linux 内核 cgroup 隔离）是在云原生环境中集成到应用生命周期的更高级别时的限制性和可观测性示例。可以将云原生运行时环境本身分解为相互关联的组件层，这些组件具有不同的安全问题（例如，硬件、主机、容器镜像运行时、编排）。[1]

在云原生运行时环境中，全球各行各业及各类组织机构都采用了微服务架构用于各类应用。应用通常由几个独立的、单一用途的微服务组成，这些服务通过容器编排层进行服务层抽象后进行通信。确保这种相互关联的组件体系结构安全的最佳实践包括：确保只有经过批准的进程才能在容器命名空间内运行，防止和通知未授权的资源访问情况，监控网络流量来检测恶意工具的活动。服务网格是另一种常见的抽象，它为编排服务提供了整合和补充的功能，而不会对工作负载软件本身进行任何更改（例如 API 流量日志、传输加密、可观测性标记、身份验证和授权）。

建议

云原生安全旨在确保实现像传统安全模型一样的谨慎性、完整性、信任和威胁防护，同时整合了临时性、分布式和不变性等现代概念。这些瞬息万变的环境支持故障转移，以便进行快速迭代。在这种环境中，需要在开发管道中实现自动化，以此来确保最终结果的安全性。组织机构必须认真分析并应用这些核心安全概念，缩短对加固和环境控制的延迟，并且需要让参与的第三方遵循相同的标准，同时平衡与云功能相关的、针对其安全支持者的持续教育和培训。

由于还要考虑到其他层的复杂性，而且还需要关注广泛的组件网格，因此，必须通过在整个生命周期内以及在运行时环境中集成安全，来防止未经授权的访问。强烈建议组织机构根据相关攻击框架[2]来评估安全防御堆栈，明确防御堆栈能够涵盖哪些威胁。此外，组织机构需要采用一些新的

[1]另一个要考虑的模型是云、群集、容器和代码：
https://kubernetes.io/docs/concepts/security/overview/
[2]示例——MITRE ATT&CK 框架（Kubernetes）

方式和方法将安全左移，扩大 DevOps 的能力。对于在生命周期内进行的创新，需要在管道之前、之中、之后持续妥善地验证所有组件。[3]

结论

在整个组织机构中有策略地执行云原生安全时，可以大规模地实现高可用性、保证、弹性和冗余，从而确保客户和开发人员能够按预期的速度安全访问所需资源。安全本身仍然是一个跨学科领域，不能与开发生命周期隔离开来，也不能视为纯粹的技术领域。开发人员、运维人员和安全人员必须加强交流和合作，才能继续推动该领域和行业的发展。与任何技术创新一样，人、激情以及整个发展过程共同造就了云原生社区，并实现了云原生安全。

[3]安全左移后，通常会让组织机构忽略对运维安全的监控。但安全存在于生命周期的所有阶段，并且组织机构必须不断评估其业务和技术流程的方方面面，这样才可能超越现代安全范式，从而形成一种文化和习惯。

引言

本文旨在让组织机构及其技术领导者对云原生安全、如何在生命周期流程中嵌入安全以及确定最合适的应用时的考虑因素有一个清晰的了解。云原生安全是一个多目标、多约束的问题领域，涵盖了许多专业知识和实践领域。从身份管理到存储解决方案，几乎所有的 Day 1 和 Day 2 运维都与安全领域有所重叠。但是，云原生安全不仅仅包括这些领域。它也是一个关于人的问题领域，包含了个人、团队和组织机构。它是人和系统与之交互并更改云原生应用和技术的机制、流程和意图。

目标受众

我们的目标受众是希望提供安全的云原生技术生态系统的私营企业、政府机构或非营利组织的首席安全官（CSO）、首席信息安全官（CISO）或首席技术官（CTO）。其他相关者可能包括负责设计和实施安全云原生产品和服务的项目经理、产品经理、计划经理和架构师。除此之外，对云原生安全有浓厚兴趣的任何人都可以参考此文档。

云原生目标

容器和微服务架构的采用和创新带来了许多挑战。在现代化的组织机构中，缓解网络安全漏洞这一需求的优先级已经明显上升。随着围绕上云的创新在不断加速，威胁形势也随之加剧。安全负责人有责任采取措施来预防、检测和响应网络威胁，同时满足严格的合规性要求，从而保护人类和非人类4资产。过去人们常说，安全的实施会阻碍 DevOps 团队的速度和敏捷性。因此，安全负责人必须让 DevOps 团队更紧密的集成在一起，加强相互理解，共同对网络风险负责。

组织机构必须要共享要采用的云原生安全采用模式及架构，确保行业以高优先级实施安全措施，并将这些安全措施整合到整个现代化应用开发生命周期中来。最重要的是，突出安全体系架构与安全负责人之间的协同作用，并符合组织机构在漏洞管理、零信任、云安全和 DevSecOps 等方面的安全目标，这些都应该是最高优先级的事项。

本文通篇所描述的概念并非是说某种产品或组件优于另一种产品或组件，而是无论选择哪些服务，都可以使用。

云原生的层次结构

[4] 人力资本是任何组织成功所必需的重要资产，由此带来的相应知识产权和关系资本同样需要受到保护。

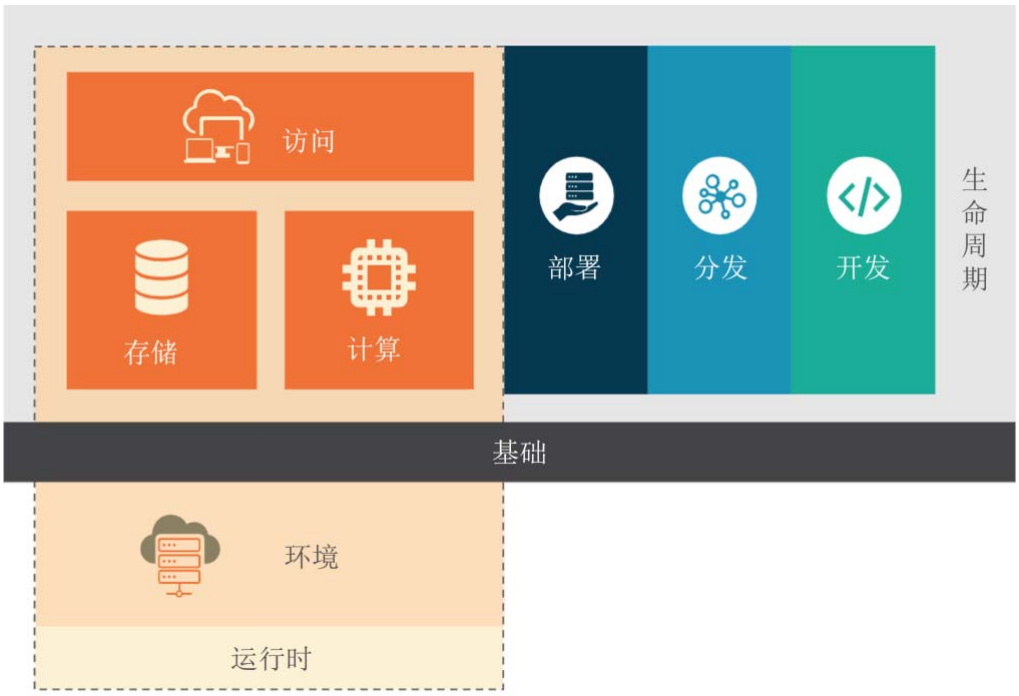


图 1

云原生堆栈由基础层、生命周期层和环境层组成。云原生堆栈可以使用不同的部署模型（IaaS、PaaS、CaaS 和 FaaS）来部署。每个部署模型都提供了更多的抽象，以简化云原生环境的管理和运维。由于其中一些模型是众所周知的并且已经使用了多年，因此，我们将重点介绍云原生的特有模型。

容器即服务（CaaS）模型可以让用户通过利用基于容器的虚拟化平台、应用编程接口（API）或 Web 门户管理接口来编排和管理容器、应用和集群。CaaS 通过将安全策略作为代码嵌入，以此来帮助用户构建可扩展的容器化应用，并在私有云、本地数据中心或公有云上运行。CaaS 有助于简化容器的构建过程。通过微服务编排和部署，CaaS 可以帮助企业加快软件的发布速度，并可以在混合环境和多云环境之间移植，从而降低基础架构的成本和运营成本。CaaS 模型可节省成本，因为它可以帮助企业简化容器管理，同时让企业可以仅仅支付希望使用的 CaaS 资源的成本费用。CaaS 将容器作为其基本资源，而对于 IaaS 环境，则使用虚拟机（VM）和裸机硬件主机系统。

功能即服务（FaaS）是另一种云原生部署模型，这是一种云服务，可以让企业用户通过执行代码对事件作出响应，而无需关心通常与构建和启动微服务相关的复杂基础架构。在云中托管软件应用通常需要配置和管理虚拟环境，管理操作系统和 Web 组件等。使用 FaaS 服务后，物理硬件、虚拟机操作系统和 Web 服务器软件管理都由云服务商自动处理。因此，用户可以专注于微服务代码中的各个功能，同时仅为所使用的资源付费，并可以充分利用云提供的资源弹性。

生命周期

云原生环境中的生命周期是指让弹性、可管理和可观测的工作负载在云环境中运行的技术、实践和流程。如图 1 所示，生命周期由四个连续的阶段组成：开发、分发、部署和运行时。每个阶段都是对前一个阶段的拓展和延伸，同时允许并支持工作负载的安全执行。

生命周期流程

供应链的管理和适用的安全基准对于安全实施至关重要。

供应链

组织机构有责任确保，可以在生命周期过程中对他们正在开发的工作负载供应链进行可行的安全分析。供应链安全可以分为两部分：提供环境创建工作负载的工具和服务的安全（例如开发人员工具）以及构成工作负载本身的组件的安全（例如库、依赖项和镜像）。供应链实施后，需要确保供应链本身的完整性可以验证，并且可以对软件供应链产生的工件进行签名以验证来源。因此，组织机构在使用依赖项时必须谨慎行事，因为上游依赖程序包可能不可避免地包含安全漏洞。验证所使用的第三方程序包的真实性和完整性对于确保依赖项按预期运行，且不会受到入侵至关重要。

云原生应用的一个主要特征是软件复用，这些软件可能以开源软件包和容器镜像的方式，通过开源仓库进行构建和分发。因此，对于开发人员、运维人员和安全人员而言，重要的一点是要确保应用中的工件和依赖项不包含已知的恶意软件和漏洞来源。容器镜像中存在恶意软件是运行时环境中的重要攻击向量[5]。必须在 CI 管道以及容器镜像仓库中对容器镜像和软件包定期或按需进行漏洞扫描。

通过这些方法可以确保软件分发和持续运行的安全性及可验证性。将漏洞扫描整合到工作负载生成管道中来，这样组织机构就可以加强对开发团队的反馈，并能够进一步阻止分发或部署不安全或易受攻击的更新软件。定期扫描软件还会发现现有软件中新发现的漏洞升级。

安全基准

安全基准（例如 NIST 应用容器安全指南、互联网安全中心（CIS）、NIST 微服务安全策略和 OpenSCAP）可为开发团队和组织机构提供创建“默认安全”的工作负载的指南。这些基准的采用和实施可以让团队开展测试，强化安全基准。但是，这并没有考虑到数据流和测试平台的自定义使用。安全从业人员应将其作为一项指南，而不是一个检查清单。

接下来的几节将详细分析在整个应用生命周期中集成安全的含义、工具、机制和最佳实践。

开发阶段

[5]https://blog.aquasec.com/malicious-container-image-docker-container-host

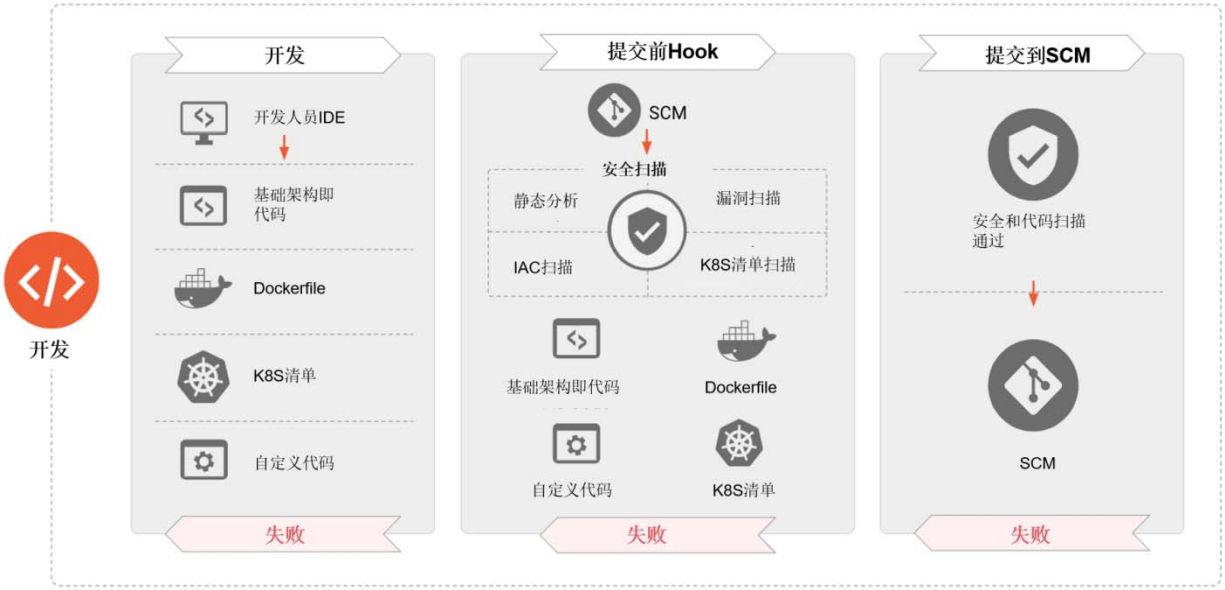


图 2

需要确保云原生应用在整个生命周期中的安全性。“开发”阶段是生命周期的第一个阶段，包括创建工件（例如，基础架构即代码，应用程序和容器清单等），用于部署和配置云原生应用。事实证明，这些工件是许多攻击向量的源头，会在运行时阶段发生漏洞利用。接下来的几节将详细介绍此阶段需要创建的各种安全工具、流程和检查，以显著缩小在运行时中部署的应用的攻击面。

开发阶段的安全检查

开发阶段中的安全加固是应用部署阶段的一个关键部分。这意味着，必须在软件开发的早期就引入安全需求，并用与其他任何需求相同的方式来处理安全需求。这些安全需求通常是围绕风险和合规进行的。在早期阶段解决这些需求可防止在生命周期的后期重新处理，而重新处理则会延缓 DevOps 流程，并增加总体成本[6]。DevOps 团队还必须利用专门构建的工具在部署应用之前就识别安全配置错误和漏洞。另一个重点在于，这些工具可以无缝集成到 DevOps 团队的现有和熟悉的工具中，以增强敏捷性和安全性，而不是对其造成妨碍。例如，这些工具需要在开发人员 IDE 中或发出“拉取请求”时，对基础架构即代码模板以及应用清单进行扫描，并提供丰富的上下文相关的安全信息，根据这些信息在开发阶段的早期快速、轻松地进行安全处理。采取这些措施可确保消除已知漏洞或高风险配置。云原生组件应由 API 驱动，让复杂的调试工具与依赖编排工具的原始工作负载进行交互。

各团队应部署专用的开发、测试和生产环境，以便为基础架构和应用的开发人员提供独立的环境来开发、测试和部署各个系统和应用、容器根镜像、VM 黄金镜像以及非功能性测试。一些组织机构可能发现，利用金丝雀部署、蓝绿部署或红黑部署以及其他部署模型可以提高动态和交互式测试和可行性研究的效率。

测试开发

开发人员、运维人员和安全人员应针对关键业务级、高威胁类型、频繁变更或具有 bug 历史记录

[6]根据 Applied Software Measurement（Capers Jones，1996 年）的研究，并将通货膨胀考虑进来，约 85%的缺陷是在编码过程中引入的，每个漏洞的修复成本为 41 美元，而发布后的修复成本为 26,542 美元。

的代码和基础架构构建测试。威胁建模可以识别高风险和高影响力的代码热点，对这些热点代码进行测试可以实现高投资回报率（ROI）。测试可能涵盖了部署、操作系统、基础架构和数据库加固、应用程序测试（静态和动态源代码测试、容器配置）、集成或系统测试（应用和基础架构组件的验收及其交互）以及冒烟测试（部署后对实时系统进行检查）。测试编写者应可以访问开发和测试综合环境，以便他们能够快速地开发测试，同时缩短持续集成（CI）反馈循环。应该为测试编写者提供系统测试包，以便在本地运行，也可以在共享测试环境中运行。

代码审查

对工作负载或已部署了工作负载的基础架构进行的细微变更，也可能会产生深远的安全问题。为了降低产生意外后果的风险，建议各团队先根据“四眼”原则进行代码审查，然后再将先前的变更整合到代码库中（例如，在 git 工作流中实现拉取请求）。

分发阶段

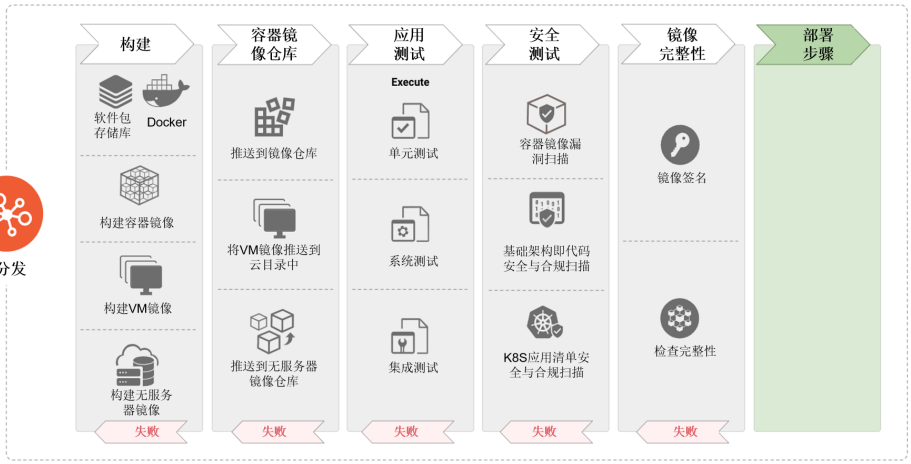


图 3

“分发”阶段的主要任务是使用镜像定义和规范来构建下一个阶段的工件，例如容器镜像、VM 镜像等等。按照现代的持续集成和持续部署范式，“分发”阶段包括系统的应用测试，以识别软件中的 bug 和错误。但是，采用开源软件和可重复使用的软件包会导致将漏洞和恶意软件引入到容器镜像中。因此，有必要执行一些安全措施，例如扫描镜像，检查是否存在上述威胁向量，以及验证镜像的完整性，防止篡改。下文将详细介绍安全最佳实践，帮助开发人员和运维人员识别并保护容器镜像免受威胁，并确保有适当的技术和工具可以保护整个 CI/CD 管道和基础架构的安全。此外，如果需要保密，组织机构可能希望对软件工件进行加密。

如果软件工件由于入侵或其他事件而变得不可信，相关团队应吊销签名密钥以确保废止。

构建管道

持续集成（CI）服务器应是独立的，并仅限于具有相似安全性或敏感性的项目使用。基础架构构建若需要提升权限，则应在独立的专用 CI 服务器上运行。构建策略应在 CI 管道中以及编排工具的准入控制器中执行。

供应链工具可以收集构建管道元数据并进行签名。然后，后续阶段可以验证签名，以验证管道的必要阶段已经在运行。

读者应确保 CI 和持续交付（CD）基础架构尽可能安全。例如，应优先考虑尽快安装安全更新版本，并应通过使用硬件安全模块（HSM）或凭据管理器来保护加密密钥不被泄露。

镜像扫描

扫描容器镜像是在整个生命周期中保护容器应用的一个重要步骤。很重要的一点是，先在 CI 管道中进行扫描，然后再将镜像部署到生产中去。这一功能可确保开发人员、运维人员和安全专业人员可以详细了解所有已知漏洞以及诸如严重性、CVSS 分数和是否具有缓解措施/修复程序之类的详细信息。将容器镜像的漏洞扫描与管道合规性规则相结合，可确保仅将妥善修补的应用部署到生产环境中，从而减少了潜在的攻击面。扫描容器镜像还有助于识别来自开源镜像仓库的开源软件包或根镜像层中是否存在恶意软件。通过容器镜像扫描，各团队可以了解是否存在漏洞或恶意软件，但并不能预防漏洞或恶意软件。在选择进行镜像扫描、设置机制根据扫描信息采取措施以及执行组织机构的合规性规则时，组织机构应谨慎行事。

镜像加固

容器镜像是构建管道的一级输出。因此，容器镜像必须进行安全加固，加固时要考虑到缓解威胁，同时允许在运行时阶段进行一些即时配置，以便与生态系统的其他部分相适应。

关于安全保证目标，应评估以下问题：

- 执行环境是否应限于特定用户？
- 是否应该限制资源的访问？
- 是否应限制内核级的进程执行？

容器应用清单扫描

应用清单列出了部署容器化应用所需的配置。如“安全基准”部分中所述，NIST 800-190 等指南和建议推荐了应用容器安全的最佳实践和配置。因此，重要的一点是要使用工具扫描 CI/CD 管道中的这些应用清单，找出可能导致不安全部署状态的配置。

容器应用清单加固

对于容器镜像，可以在构建阶段以及运行时阶段考虑并执行容器应用清单加固。

关于安全保证目标，应评估以下问题：

- 运行时执行生态系统应遵守哪些最小约束？

测试

云原生应用应遵循与传统应用相同的套件和质量测试标准，包括整洁代码、测试金字塔、通过静态应用安全性测试（SAST）进行应用安全性扫描和 lint 扫描、依赖性分析和扫描、动态应用安全性测试（DAST）（例如模拟试验）、应用工具以及开发人员可以在本地工作流程中全套测试基础设施。自动化测试结果应按照二重证据法（开发人员和工具）的要求进行映射，以向安全和合规团队提供实时安全保证。

一旦确定了安全 bug（例如，错误的防火墙或路由规则），如果通过根本原因分析确定了该 bug 有可能再次出现，那么开发人员应编写自动测试以防止重现。在测试失败时，团队会收到反馈以

对该 bug 进行修复，并且在下一次合并时，通过测试（假定 bug 已修复）。这样做可以防止将来变更代码时回退到最初的代码。

基础架构的单元测试是一种预防性控制，其目标是基础架构即代码（IaC）配置中定义的实体和输入。对已构建的基础架构进行安全测试是一种检测性控制，结合了安全保证、历史回归和非预期配置检测（公开的防火墙规则、IAM 特权策略、未认证的端点等）。

应有全面的测试套件来支持对基础架构和工作负载进行加固，这可以随着系统的成熟而逐步加固。在构建阶段应该进行测试，确认已经进行了加固，但也应在部署阶段时执行测试，进而评估整个生命周期中可能发生的任何变更或回归。

静态分析和安全测试

对 IaC、应用清单和软件代码的静态分析可能包括 lint 扫描、识别错误配置和漏洞扫描。IaC 代码应遵循和应用工作负载相同的管道策略。

IaC 越来越受到青睐，有越来越多的组织机构在执行 IaC，用于部署云和容器基础架构。因此，这些模板中的不安全配置可能导致攻击向量的暴露。

在部署应用和基础架构工件之前，应使用自动化工具对这些模板进行扫描，以查找不安全配置和其他安全控制。需要注意的关键性错误配置包括：

- 应用清单中所述镜像中所包含的缺陷。
- 配置设置，例如，可以提升权限的容器。
- 识别有哪些安全性上下文和系统调用会导致系统入侵。
- 资源限制设置。

动态分析

对已部署的基础架构进行的动态分析可能包括检测 RBAC 和 IAM 配置漂移，验证预期的网络攻击面以及确保 SOC 可以在专用测试环境中检测异常行为进而为生产环境配置警报信号。动态分析被认为是测试的一部分，但是，也可以在非生产运行时环境中进行。

安全测试

对应用和基础架构进行自动安全测试是安全团队中应该是不可或缺的重点内容。测试套件应不断更新，按组织威胁模型复制威胁，并可随系统的发展重新用于安全性回归测试。自动化安全测试项消除了手动安全门（例如，在单个检查点进行验证和手动控制操作，既耗时又不充分）来提高安全性和发布速度。自动化安全测试可根据需要明确尝试执行威胁，证明按需控制的有效性，从而实时改善系统的安全性以及对任何嵌入式合规要求的遵从性。

工件和镜像

镜像仓库暂存

由于通常使用从公共资源中获取的开源组件，因此，组织机构应在其管道中创建多个用于暂存的镜像仓库。只有经授权的开发人员才能访问公共镜像仓库并拉取根镜像，然后将其存储在内部镜像仓库中以供组织内部使用。另外，建议组织机构建立单独的私有镜像仓库，用于存储每个团队或每个小组的开发工件，最后，还应建立一个暂存或生产前的镜像仓库，以备生产阶段使用。这

样就可以更严格地控制开源组件的来源和安全性，同时可以对 CI/CD 链中的各个阶段进行不同类型的测试。

对于任何已使用的镜像仓库，必须通过专用的身份验证和权限模型来实现访问控制。（在架构内的交互中）对所有镜像仓库之间的连接，使用相互认证的 TLS。

签名、可信任及完整性

在构建阶段对镜像内容进行数字签名，并在使用前对签名数据进行验证，以防止镜像数据在构建阶段和运行时阶段之间被篡改，从而确保工件的完整性和来源。验证应从说明工件已经过审核和批准这一流程开始。可信性验证还包括验证工件具有有效签名。在最简单的情况下，每个工件可以由一位签名者签名，以表明该工件经过了一个测试和验证过程。但是，在大多数情况下，软件供应链更为复杂，创建一个工件需要多个验证步骤，因此这取决于多个实体的可信性。例如：

- 容器镜像签名——容器镜像清单的签名过程
- 配置签名——配置文件（即应用配置文件）的签名：在 GitOps 方法中最常见，其中可能包括一个验证和检查配置的过程。
- 软件包签名——对工件软件包（如应用软件包）进行签名。

对于诸如库或 OCI 工件之类的通用软件工件，要对这些工件上进行签名，表明它们已被组织机构批准使用。验证这些工件对于确保仅允许使用授权的工件同样至关重要。强烈建议在对存储库中的镜像进行变更或将代码提交到存储库中时，对存储库进行相互认证。

加密

容器镜像加密即对容器镜像进行加密，使其内容保持机密。容器镜像内容经过加密后，可确保内容从构建阶段到运行时阶段都可以保持机密性。如果在分发时遭到入侵，镜像仓库的内容仍可以保持机密，这有助于解决诸如保护商业秘密或其他机密材料之类的用例。

容器镜像加密的另一个常见用途是强制执行容器镜像授权。当镜像加密与密钥管理认证和/或授权和凭证分发相结合时，可能会要求容器镜像只能在特定平台上运行。容器镜像授权对于合规用例（例如，区域限制或出口控制以及数字版权媒体管理）很有用。

部署阶段

图 4

“部署”阶段的主要任务是整合一系列运行前检查，确保要在运行时环境中部署的应用符合并遵守全组织机构范围内的安全性和合规性策略。

运行前部署检查

部署之前，组织机构应验证以下各项是否存在、是否使用及其当前状态：

- 镜像签名和完整性
- 镜像运行时策略（例如，不存在恶意软件或严重漏洞）
- 容器运行时策略（例如，不存在多余的特权）
- 主机漏洞和合规性控制
- 工作负载、应用和网络安全策略

可观测性和衡量指标

将可观测性和衡量指标纳入云原生架构可提供安全见解，这样，相关方就可以解决和减少报告时出现的异常情况；这方面的工具可以有助于收集和可视化此类信息。通过使用行为和启发式分析，团队可以检测并升级异常数据、可疑事件以及对相关方不明原因的调用。鼓励使用人工智能（AI）、机器学习（ML）或统计建模来协助进行行为和启发式分析开发。

响应和调查

一款应用应提供有关身份验证、授权、操作和失败的日志。开发人员应将此功能纳入计划和设计阶段。当进行调查并需要确定根本原因时，这些要素就提供了可循的证据。

取证能力是任何事件响应和缓解活动不可或缺的一个组成部分。该能力为确定事件的根本原因提供了证据，并为要采取的任何缓解措施提供了反馈。容器环境的短暂特性要求使用更敏捷的工具集来捕获和分析任何证据。将取证能力集成到事件响应计划和程序中，将为获取和处理证据提供

方法，缩短确定根本原因的时间，并最大程度地降低入侵风险。

运行时环境

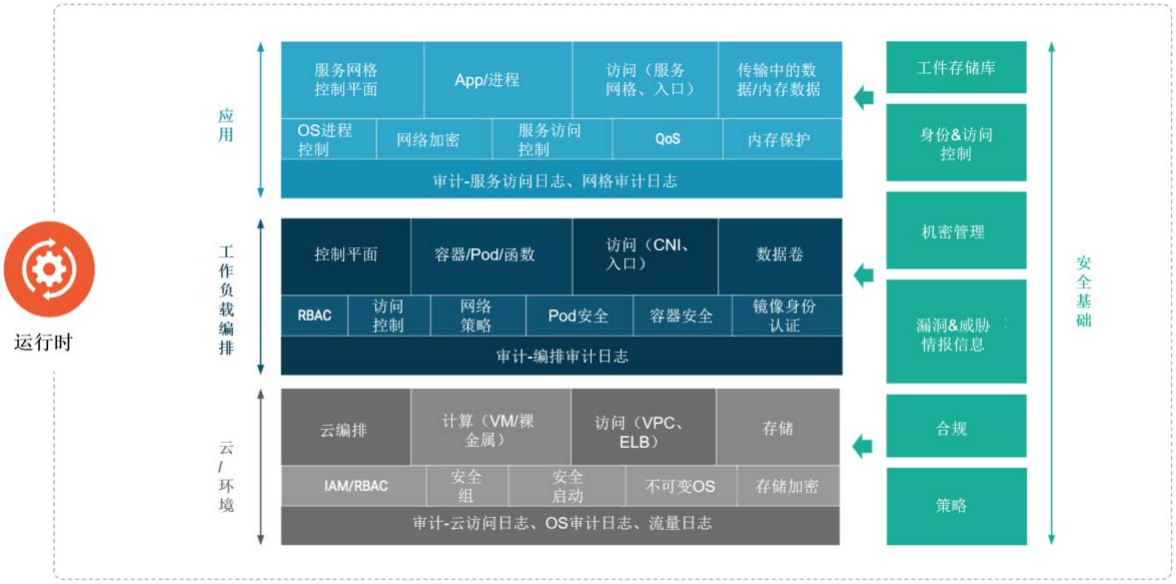


图 5

运行时阶段包括三个关键领域：计算、访问和存储。尽管运行时环境取决于开发、分发和部署阶段的成功完成，但是运行时的安全性取决于先前各阶段安全实践的之有效性。下文详细介绍了每个关键组件的安全要求及其意义。

计算

云原生计算是一个非常复杂且不断发展的结构。若没有核心组件来提高计算利用率，组织机构将无法确保工作负载的安全性。

考虑到容器为共享主机上的多租户应用提供了基于软件的虚拟化，因此，使用容器专用的操作系统非常重要。通常，容器专用操作系统是一个只读操作系统（OS），禁用了其他服务。使用这种 OS 有助于减少攻击面，还提供了独立性和资源限制，让开发人员能够在共享主机内核上运行独立的应用。为了实现深度防御，还建议不要在相同的 OS 内核上运行不同的数据敏感度的工作负载。

为了保障容器平台和服务所有层的安全，可以使用基于可信平台模块（TPM）或 vTPM 的硬件信任根。可以将源于硬件的信任链扩展到 OS 内核及其组件，以对可信启动、系统镜像、容器运行时和容器镜像等进行加密验证。

操作系统提供基本的系统组件，例如用于远程连接的加密库以及用于进程启动和管理等的内核功能。这些操作系统可能存在漏洞，并且由于它们为容器提供了底层计算基准，因此可能会影响在主机上运行的所有容器和应用。同时，容器配置不当会影响主机内核的安全性，从而影响在该主机上运行的容器中的所有服务。更多信息，请参阅“分发阶段”了解详情。

编排

任何编排工具都有多个组件，可以分成不同层，例如控制层和数据层。有时，需要具有一个较高层次的多部署管理结构，负责维护多个彼此独立并存的不同控制层。

任何编排系统都面临诸多威胁，影响部署的整体安全性以及运行时的持续安全性。为控制集群、拦截控制层流量而恶意访问编排工具的 API、对键值存储和编排工具仪表板进行未经授权的访问以及更改、滥用 API、拦截应用流量等都是潜在威胁。对于任何编排工具而言，由于存在多种威胁，使用最佳实践和配置加固以防止遭受威胁是很重要的7。监控和检测在运行时阶段对初始配置所做的任何更改对于确保集群的持续安全状态也很重要。其他安全的最佳实践，如最大限度地降低对控制层的管理访问权限、职责划分和最小权限原则，均应予以执行。

安全策略

必须考虑编排工具的安全特性和各种配置方案，以控制在容器运行时生成容器的安全特权。使用较高级别的策略和治理结构可能会加强这些安全防护。

资源请求和限制

通过 cgroup 应用不同的对象级别和资源请求及限制，有助于防止由于一个有意（例如，fork 炸弹攻击或加密货币挖矿）或无意的（例如，读取内存中的大文件而没有输入验证，水平自动定量以耗尽计算资源）问题导致工作负载出现问题，耗尽节点和集群资源。

审计日志分析

审计日志分析是识别和关联系统入侵、滥用或配置错误的最成熟的方法之一。审计日志分析和关联的持续自动化对于安全团队而言至关重要，因为与传统系统相比，云原生架构能够为工作负载生成更细粒度的审计配置和过滤功能。此外，通过云原生日志的互操作性，可以进行高级过滤，防止下游处理中的过载问题。与传统的日志分析一样，此处至关重要的是生成可操作的审计事件，将日志中的数据关联起来/形成上下文，形成驱动决策树/事件响应的“信息”。

根据一组预先配置的规则来检测不合规的违规行为，以过滤违反组织机构策略的违规行为。

为了能够使用集群来审计实体的操作行为，启用 API 审计功能非常重要，其中要包含安全团队、集群管理人员或其他学科团队感兴趣的针对一组特定 API 组或动词的过滤器。将日志直接转发到通过集群级凭据无法访问的位置，也可以挫败攻击者通过禁用日志或删除其活动日志来掩盖其踪迹的企图。处理警报的系统应定期调整为误报，以避免在系统未检测到安全事件后出现警报洪泛、疲劳和误报。

控制平面身份验证和根信任证书

除了加固现有控制平面之外，编排工具管理人员还应配置所有编排工具控制平面组件，如控制器——管理器、调度器、API 服务器和 kubelet（如适用），以便通过定期轮换的证书完成相互认证和证书验证，进行通信。发出命令的 CA 可以是默认编排工具 CA 或外部 CA。管理人员应特别注意保护 CA 的私钥。有关扩展或建立信任的更多信息，请参阅本文的“身份认证”部分。

机密数据加密

---

[7]CIsecurity.org 维护了用于加固的基准测试清单

通过使用外部机密管理工具或在本地使用编排工具的机密数据，可以在容器编排或部署环境中管理机密数据。使用本机机密存储区时，注意有几种不同的保护方法可用：

- 使用外部密钥管理存储（KMS）进行加密
    - 使用 KMS 是保护编排工具机密存储区中机密数据的一种安全方法，其中，外部 KMS 中的密钥加密将对数据加密密钥（DEK）进行加密，而该 DEK 将对 etcd 中的静态机密数据进行加密。此方法确实可以在内存中缓存 DEK，减少对外部 KMS 的依赖，并在 Pod 创建期间更快地解密机密数据。
- 由编排工具管理的加密
    - 这种方法会对存储在编排工具中的机密数据进行加密，但是加密密钥也由编排工具管理（例如，编排工具的配置文件）
- 不加密
    - 例如，对于某些编排工具，机密数据是 base64 编码的，并且在默认情况下以明文形式存储在键值存储区中

使用外部机密数据管理工具可以限制使用未加密机密数据带来的风险，并降低了密钥管理的复杂性。大多数情况下，这些工具是作为控制器或运算器提供的，可以在运行时注入机密数据并透明地进行轮换。

容器

运行时环境

需要从进程、文件和网络的角度监控和保护容器的运行时环境。主机操作系统只应允许在容器中执行或调用经过批准的功能和系统调用（例如 seccomp 过滤器）。应监控并阻止对关键挂载点和文件的变更。必须通过配置防止更改二进制文件、证书和远程访问配置。还必须通过配置阻止容器的出入网络访问，只访问需要操作的内容。此外，应检测并拒绝恶意域名的网络流量。

微服务架构和消除绝对信任

以微服务架构部署的容器化应用，其边界即微服务本身。因此，有必要定义好策略，限制仅经批准的微服务对之间方可进行通信。在微服务架构中纳入零信任，则可以在微服务架构被入侵时，通过防止横向移动来减小影响范围。运维人员应确保他们正在使用诸如网络策略之类的功能，以确保容器部署内的东西网络通信仅限于授权访问的范围。NIST SP 800-204 文件为微服务安全提供了一些策略，已经做了一些初步工作，并且可以作为实现安全微服务体系结构的指南。

镜像信任和内容保护

利用策略代理来强制执行或控制已签名授权的容器镜像，这样组织机构便能够保证工作负载的镜像来源。此外，引入加密容器可以保护容器内存在的敏感源、方法或数据。

服务网格

服务网格将各项服务连接起来，以此实现流量控制、服务发现、负载平衡、弹性、可观测性、安全性等。服务网格能够让微服务架构从应用库中卸载这些功能，并让开发人员能够专注于区分业务逻辑。为了有效地确保云原生环境中各服务之间的安全通信，组织机构应执行服务网格以消除

其 Pod 内部和不同工作负载之间通过动态数据加密实现的绝对信任。使用服务网格还解决了身份问题，其中，传统的第 3 层和第 4 层身份（即 IP 地址）不再明确地映射到工作负载。服务网格不仅提供网络级别的独立性和安全性，而且还提供网络级别的弹性功能，例如，重试、超时和实现各种断路功能。流处理平台可以通过使用工作负载级别的授权来设置主题或代理的访问规则，从而从服务网格中受益，并提高安全性。

值得注意的是，服务网格的执行可以有助于减少云原生部署的攻击面，并提供用于构建零信任应用网络的关键框架。

运行时阶段的检测

监控已部署的工作负载应为团队提供验证，以确保真实的操作状态是预期的状态。组织机构不能放弃在其环境中进行定期安全扫描和监控，而不将其工作负载变成攻击者的无监督场所。应该使用用于检测、跟踪、汇总和报告系统调用的组件以及来自容器的网络流量，来查找意外或恶意行为。

尽管回归测试和安全测试项有利于防止将已知的预期问题转移到生产环境中，但它们无法阻止这一切的发生。应对工作负载进行动态扫描，以检测尚无已知事件的恶意或阴险行为。在大多数环境中，预计不会发生扩充休眠命令之类事件，在工作负载已运行 X 天后从 etcd 进行数据渗出，因此，这些事件未包含在安全测试中。关于工作负载可能具有时间或事件延迟的特洛伊木马这一方面，只能通过对比基准预期行为与通常在细致的活动和扫描监控期间发现的行为才能检测到。

此外，工作负载在部署之时或之后将变得易受攻击。组织机构应不断扫描其环境，以检测哪些工作负载现在易受攻击。了解每个工作负载的组成或构成清单有助于组织机构快速确定漏洞的位置。有关这些漏洞的其他信息，例如，漏洞利用成熟度和使用时易受攻击的路径，对于确定工作负载的实际风险至关重要，并且可以帮助组织机构对有风险的应用进行优先更新。

函数

无服务器函数易受到各种攻击，因此，需要得到适当的保护。进程必须只能执行在允许列表中明确定义的函数。此外，不允许函数更改关键文件系统的挂载点。

这些函数必须具有限制条件，仅允许通过网络限制或权限模型中的最小权限来访问已批准的服务。此外，出口网络连接必须由管理人员进行监控，以检测并阻止（在可能的情况下）对 C＆C（命令和控制）和其他恶意网络域的访问。还必须考虑进行入口网络检查，以检测和删除可在数据渗出中使用的恶意载荷和命令。例如，可以通过检查来检测 SQL 注入攻击。

无服务器函数有多种威胁，而可供租户使用的控件是有限的。其中一些问题，如身份验证失效和 API 与相关服务的集成不安全。确保所有无服务器函数都在基于租户的资源中运行，或者通过性能隔离对类似的数据分类可能有助于解决此问题，但是，由于隔离环境可用的地址空间有限，它们可能会对性能产生影响。

启动程序

需在计算节点中启动信任，以确保工作负载和配置在正确的节点上运行。启动程序可确保计算处于正确的物理和逻辑位置，并具有对自身进行身份验证的能力。这些步骤通常是云提供商部署服务的一部分。但是，可以使用一些方法来验证信任，而不用太多地依赖于第三方。

存储

云原生存储涵盖很多技术，包括两类存储方式，一类是可用于工作负载（例如数据卷）的存储，包括块存储、文件系统和共享文件系统；另一类是访问存储，可通过应用 API 访问，包括对象存储、键值存储和数据库。

存储系统包含一个数据访问接口，定义了应用或工作负载如何存储或使用由存储系统或服务保留的数据。可以通过访问控制、身份验证、授权以及传输过程中的潜在加密来保护此接口。

存储系统还包含一个控制平面管理接口，通常是受身份验证和 TLS 保护的 API，不过也可以使用更细粒度的访问。一般来说，编排工具或服务代理只能通过服务帐户访问控制界面。

存储堆栈

任何存储解决方案都由多层功能组成，这些功能定义了如何存储、检索、保护数据以及数据如何与应用、编排工具和/或操作系统进行交互。每层功能都有可能影响存储系统的安全性。一个常见的示例是，通过一个文件系统将文件或块持久保存到对象存储中。保护拓扑结构中的每一层都同等重要，而不仅仅是访问数据的顶层。

编排

大多数编排的系统都能实现各种抽象和虚拟化层，其中可能包括文件系统（例如绑定挂载）、数据卷管理器以及基于编排工具策略，具有用户或组级别权限的应用。与容器化和微服务体系结构的许多组件一样，对数据卷和存储的保护将始终依赖对其他功能的保护。如果用户能够在编排工具或容器运行时当中将其权限升级为根用户，则其会在环境中造成严重破坏。零信任、最小权限以及访问控制和强制执行的实现是成功保护云原生体系结构中存储安全的关键所在。

系统拓扑结构和数据保护

了解系统的存储拓扑结构是确保分布式拓扑结构中存储系统的数据访问路径和节点内通信安全的关键。

常见的拓扑结构包括所有计算节点都访问中央存储服务的集中式模型，在多个节点上分布功能的分布式模型以及应用和存储工作负载整合于同一节点上的超融合模型。系统根据所使用的拓扑结构选择特定的、分层的安全机制来保护存储中的数据以及在存储位置之间传输的数据。

任何存储系统的一个关键功能是为存储在系统或服务中的数据提供保护。首先，通过授权用户使用数据来实现此保护，并且该保护应作为系统中透明的一层。这可以包括诸如奇偶校验或镜像、纠删码或副本之类的技术。接下来，进行保护的是完整性，其中存储系统将为块、对象或文件添加哈希和校验码，旨在检测并恢复已破坏的数据，但还可以添加一层保护以防止数据被篡改。

缓存

缓存层通常是完全成熟的独立系统，旨在提高存储系统（尤其是文件系统、对象和数据库）的性能。需要将适当的访问控制和安全策略应用于缓存层，因为缓存发生在对实际存储后端进行访问之前。

数据服务

存储系统通常提供许多数据服务，这些数据服务通过提供可以在堆栈的不同层上实现的附加功能，补充核心存储功能，这些附加功能还可能包括复制和快照（数据的时间点副本）。这些服务

通常用于将数据副本移动到远程位置，且重要的是，要确保将相同的访问控制和安全策略应用于处于远程位置的数据。

物理或非易失层

云原生存储安全不仅仅限于虚拟的云原生架构，因为云原生功能可以在本地部署，甚至虚拟产品都具有物理存在形式。重要的是要记住，存储系统最终会将数据持久保存在某种形式的物理存储层上，该层通常是非易失的。诸如 SSD 之类的现代物理存储通常支持安全功能（例如根据 OPAL 标准的自加密）和快速/安全擦除功能。在包含数据的设备需要离开安全的物理位置时（例如在出现故障后退还给供应商），安全擦除非常重要。

存储加密

存储系统可以提供通过数据加密来确保数据机密性的方法。可以对传输中的数据或静态数据执行数据加密，并且在存储系统中使用数据加密时，可以确保单独对应用实现加密功能。

加密可能会影响性能，因为它隐含了计算开销，但是许多系统上都提供了加速选项，可以减少开销。在为数据选择加密类型时，请考虑数据路径、大小和访问频率，以及可能需要使用更安全算法的任何规定或其他安全保护。此外，团队在考虑其架构的加密需求时，不应忽略使用缓存。

可为传输中的数据（保护网络中的数据）和静态数据（保护磁盘上的数据）执行加密服务。加密可以在存储客户端或存储服务器中实现，并且加密的粒度将视系统变化而变化（例如，按数据卷、按组或全域密钥）。在许多系统中，传输中的数据用 TLS 保护（TLS 的另一个优势在于通过证书8提供身份验证层）。较旧的协议（例如 iscsi）可能更难保护传输中的数据安全（但可以使用更复杂的解决方案，如 IPsec 或加密的 VPN9）。通常可以使用标准对称加密算法（例如 AES）来保护静态数据，并且可以使用特定的加密模式（例如用于块设备的 XTS）来部署静态数据。

加密功能通常取决于与密钥管理系统的集成。

持久卷保护

数据卷存取保护对于确保仅授权的容器和工作负载可以利用所提供的数据卷至关重要。必须为命名空间定义信任边界，以限制对数据卷的访问。利用现有安全策略或创建新的安全策略，防止容器组访问工作节点上的数据卷挂载，并确保只有适当的工作节点才能访问数据卷。这一点尤其重要，因为特权容器可以访问其他名称空间中挂载的数据卷，所以需要采取额外的预防措施。

为数据卷指定 UID 或 GID 后，同一名称空间内的容器仍可以访问，但不提供数据保护。网络文件系统版本 3（NFSv3）假定客户端已经执行了身份验证和授权，且未进行验证。在执行保护时，必须考虑在何处进行身份验证和授权，以及是否存在对该操作进行确认，这一点至关重要。

工件存储库

存储库应提供用于签署和验证 OCI 工件的技术。还有一点很重要，要确保缓存和分发工具也提供签名、加密功能和校验码功能，进而确保缓存层可以检测到篡改或破坏数据集的行为。

[8]务必注意，虽然可以使用身份验证，但是相互认证是首选机制，不仅可以验证客户端，还可以验证服务器（外部人员与内部人员）。
[9]使用 VPN 并不能保证加密。

《CNCF 存储白皮书》介绍了有关云原生存储的概念、术语、使用模式和技术类别的更多信息。

访问

身份和访问管理

云原生架构的综合性身份和访问管理（IAM）解决方案至少应包括服务身份。维护或运营本地私有云或混合云的组织机构需要用户和设备身份管理。对于跨多云环境分布的应用和工作负载，身份联合对于 IAM 的成功执行至关重要。

应明确授权应用和工作负载使用相互认证进行彼此通信。由于云计算的短暂性，密钥轮换频率要频繁，使用期限要短暂，以维持对高速功能的需求，并控制和限制受损凭证的影响范围。

使用云提供商提供的身份管理服务要取决于行业特定的用例。独立于云提供商的用户应生成和管理用于敏感工作负载（例如，健康或财务信息）的凭证和密钥。

为了使客户端和服务器能够通过密码双向验证身份，所有工作负载必须利用双向身份验证。

身份验证和授权必须在环境内和整个环境中各自确定（决策点）并强制实施（实施点）。理想情况下，应实时地确认所有工作负载的安全操作，并在可能的情况下验证更新的访问控制和文件权限，因为可能通过缓存实现未经授权的访问（如果访问被吊销且从未经过验证）。根据已为工作负载分配的属性和角色/权限对工作负载进行授权。强烈建议组织机构同时使用基于属性的访问控制（ABAC）和基于角色的访问控制（RBAC）在所有环境以及整个工作负载生命周期中提供细粒度的授权执行。这种做法可以实现纵深防御，其中所有工作负载都可以接受、使用和转发最终用户的身份以进行上下文或动态授权。这可以通过使用身份证明文件和令牌得以实现。若不强制执行此操作，组织机构对系统到系统以及服务到服务的调用执行最小权限访问控制的能力就会受到限制。

值得注意的是，应用或服务身份在微服务架构的环境中也不可或缺，在微服务架构中，应用的身份主要受到恶意服务的欺骗和模仿。利用强大的身份框架和服务网格有助于克服这些问题。

必须对所有人员集群和非人员集群以及工作负载操作人员进行身份验证，且必须根据访问控制策略对其所有的操作进行评估，评估每个请求的上下文、目的和输出。为了简化身份验证过程，可以将身份联合配置为允许使用企业功能，例如多重身份验证。接下来，必须使用本节中提到的访问控制机制来执行授权。

凭证管理

硬件安全模块（HSM）

只要有可能，读者应尽可能使用诸如 HSM 之类的技术，通过加密密钥对加密机密进行物理保护，因为加密密钥永远不会离开 HSM 保护边界。如果无法做到这一点，则应使用基于软件的凭证管理器。

凭证管理周期

加密机密应在 HSM 或基于软件的机密管理系统中安全生成。

机密的有效期限或存在时间应尽可能短，在此之后它们将变得毫无用处。机密管理应该是高度可

用的，且易于生成，因为这些特征是机密保持短暂性的前提。但是，如果组织机构使用的是使用期长的机密，则应建立适当的流程和指南进行定期轮换或撤销（但不建议这样做），特别是在偶然泄露机密的情况下。所有机密必须通过安全的信道在传输过程中进行分发，并应根据其所保护的访问或数据级别进行相应的保护。

不管怎样，应在运行时通过非持久性机制将机密信息注入工作负载中，这些机制可以防止通过日志、审计或系统转储（即内存中共享的数据卷而非环境变量）而造成机密泄漏。

可用性

拒绝服务（DoS）和分布式拒绝服务（DDoS）

云原生应用环境下的拒绝服务攻击（DoS 攻击）是指一类网络攻击。攻击者试图让云原生应用暂时或长期无法供其目标用户（人工或自动化）使用。攻击者可以通过破坏关键的云原生应用组件（例如微服务架构），破坏负责保持微服务架构运行的编排层或破坏负责应用扩容的运行状况监控系统来实现攻击。DoS 通常是通过向关键微服务架构或资源注入过多的请求来使系统超负荷工作，并阻止某些或所有合法请求实现攻击。

分布式拒绝服务攻击（DDoS 攻击）通常涉及大量传入流量，这些流量会对云原生应用服务或其所依赖的上游网络造成洪泛攻击。通常情况下，攻击的来源各不相同。在攻击到达云原生应用之前对其进行检测和分流，可以缓解大部分攻击。

安全保证

从根本上讲，安全是一个风险管理过程，旨在识别和解决对系统构成的风险。根据组件或组织机构的风险状况和容忍度，对系统迭代进行永久加固可缓解、降低或转移风险。加固这一概念虽然其核心是传统概念，但仍可以根据最小但灵活的功能单元，评估组件及其构成，然后将其应用于前向安全团队。例如，当团队确定更新后的根镜像时，应检查随更新一起添加进来的其他端口、权限和程序包，并对其予以接受、更改或限制。

相比之下，合规性标准构成了控制原则，以确定或创建需求定义，据此来评估系统。评估结果为二选一（通过或未通过），但可能包含类型 1（误报）或类型 2（漏报）错误，应对 CI/CD 管道的测试结果进行评估，类似于管道中的任何测试结果。因此，合规性和安全性保证是互补的过程，但不可互换。系统合规并不能保证是安全的，安全的系统也不能保证是合规的。

威胁建模

对于采用云原生的组织机构来说，识别风险以及由此产生的控制和缓解措施的主要机制是对应用、数据流以及支持的流程和基础架构进行威胁建模。完成此任务的方法与典型的威胁建模大同小异。以下指南是针对云原生功能的 OWASP 四步威胁建模方法的改进版本。

端到端架构

对组织机构或个人的云原生体系结构有清晰的了解后，应该可以得出数据影响指导和分类。这有助于团队在架构内组织数据分发以及后期的其他保护机制。云原生图表和文档应该不仅包括整个系统设计的核心组件，还应考虑源代码的位置，正在使用的存储桶和其他存储机制以及软件开发

周期的任何其他内容。这些都是在对云原生进行威胁建模时必须考虑到的。

威胁识别

在考虑针对组织机构的云原生功能的威胁时，建议利用成熟、使用方便的威胁模型，例如 STRIDE 或 OCTAVE。组织机构针对其云原生体系结构可能考虑的常见威胁包括但不限于：

- 通过社工攻击窃取身份验证凭证来假冒集群管理人员
- 篡改 API 服务器配置文件或证书可能会导致 API 服务器重启失败或 TLS 相互验证失败
- 对禁用 API 或配置错误的 API 进行审计，可能导致攻击者否认其行为，从而导致缺乏潜在攻击证据
- 如果攻击者入侵了正在运行的工作负载并能够将数据泄露到外部实体，则可能致使信息泄漏
- 由于 Pod 没有资源限制而导致拒绝服务（DoS），因此消耗了整个节点的 CPU 和内存，并导致工作节点丧失
- 如果 pod 运行时的安全策略不受限制或权限较高，或者修改 pod 或容器的安全上下文，则可能会发生权限提升

对于云原生安全，要考虑的威胁主体需与现有的威胁模型保持一致：

- 恶意内部人员
- 不知情的内部人员
- 恶意外部人员
- 不知情的外部人员

建议组织机构利用云原生环境中的现有资源来获取有关云原生架构所面临威胁的更多信息。

使用管道和基础设施即代码（IaC）可以为某些威胁提供补偿控制或缓解控制，或降低攻击成功或发生的可能性。

与任何云原生流程一样，重要的一点是要进行迭代并提供反馈信息。就威胁建模来说，这意味着在架构不断变化的情况下，要重新评估现有措施、机制和矩阵是否准确反映了运行状态。

威胁情报

从设计和目的来看，云原生应用是受到第一方和第三方代码和工具威胁的多个动态组件之集合，这意味着威胁情报必须应用于网络活动和云原生应用组件。网络威胁情报是有关威胁和威胁主体的信息，有助于缓解有害事件。云原生系统中的威胁情报会利用在网络或主机上观察到的指标，例如 IP 地址、域名、URL 和文件哈希，这些指标都有助于识别威胁。行为指标，例如威胁主体的战术、技术和步骤，也可以用于识别云原生组件中威胁主体的活动。MITRE ATT＆CK 云框架包括云原生策略和技术，可用作建立和验证可观测性的基础。

应急响应

若组织机构已有现有的应急响应和分类工作流，应特别注意如何将其应用于云原生工作负载，这些工作负载可能并不总是符合某些有关节点隔离（新的 Pod 实例可以在其他服务器上运行）、网络（例如动态分配 IP 地址）和不变性（例如，对容器的运行时更改在重新启动后将不复存在）的基本假设。因此，重新审视这些假设并根据需要重新应用或更新应急响应手册非常重要。可观测性和取证工具需要了解云原生的特殊结构（例如，Pod 和容器），以便可以维护或恢复被攻击系统的状态。在目的明确的编排工具中，有时，可能会无意地造成证据处理不当，编排工具将工作负载视为大批量的任务，不会那么小心谨慎。另外，从头开始构建应急响应和分类策略（尽管管可能）不在本文件的讨论范围之内。

安全堆栈

环境

（工作负载）运行前的安全工具

工作负载前的安全工具应能最大程度地进行加固，并确保遵循最佳的安全实践，同时最大程度地降低与主机环境、网络和编排工具层有关的权限。工具还应确保合在运行时阶段不会出现违规。

计算和节点检查

在将资源标记为可以接受工作负载之前，组织机构应先利用工具进行加固，并确保计算安全。为此，建议使用主机漏洞扫描器和 CIS 基准扫描器。

运行上下文

用于工作负载前安全运维习惯检查防护面的安全工具最适合作为 CI 管道的一部分，用以扫描文件、工件（例如容器镜像）和 IaC。在 CD 管道中运行的安全工具更适合在特定环境的上下文中运行，并考虑到指定环境的特定配置。若云原生编排工具支持准入时机检查，则组织机构在应用工具时可以进行准入机制设置，弥补早期阶段未发现的问题。

运行中的安全工具

工作负载和主机运行时安全

运行时安全工具可分为四类，用于保护四个关键领域：

- 进程、容器或系统级安全性
- 网络安全
- 数据安全
- 应用安全

针对组织机构关注领域的每个防护面，应使用多种安全工具。策略驱动的工具可以执行基于规则的策略，无论是手动编写还是通过推荐系统编写。应用之后，这些策略驱动的工具将提供可预测的结果，并且可以在仅监视或强制执行模式下应用。

威胁和漏洞信息可以确保安全工具能够拦截来自未知威胁和已识别威胁的异常行为和安全事件。这些信息通常会定期和频繁地更新。使用这些信息相当于拥有一个防御层，可以补充策略驱动工具，并且能够应用到负责大多数防护面的工具中。团队应该关注信息中已知命令与控制（C&C）服务器、加密域名、恶意软件文件校验码等网络威胁情报，以帮助更新策略工具。

尽管现有工具可以提供各种机制，管理因误报和漏报问题产生的噪声并处理已知威胁，并使用策略驱动的防护措施来规范操作，但基于机器学习（ML）的安全工具提供了一个已知和未知威胁的检测层，超出了可预测工具所能建立的范围。例如，对身份授权日志进行基于行为的分析以检测内部威胁和破坏，或者对编排工具审计日志进行自适应分析以检测利用企图或服务帐户盗用。ML 驱动的主机系统调用模式分析可用于检测容器逃逸或主机利用企图。

监控和跟踪各种云原生编排安全的编排安全工具通常作为针对特定域名的商业产品来提供，功能广泛，包括精细的策略控制、合规性检查、基于 AI/ML 的异常检测和良好的集成面。

与任何云原生工作负载一样，用于监控、报告或控制云原生环境安全的工具也应该是云原生的，从而便于使用、管理和部署。

零信任架构

零信任架构通过细粒度的细分、微界限以及通过验证和执行策略消除对数据、资产、应用和服务（DAAS）的绝对信任，从而减少了网络内横向移动的威胁。零信任架构最常见的实现方式是通过加密概念来创建零信任。这主要得益于能够在硬件或令牌中保护特定密钥材料并并进行妥善管理，将其安全地传达到平台。

零信任架构的基本构成要素通常包括以下几个方面：

- 每个实体都可以创建身份证明文件
- 各实体能够独立地验证其他身份（例如公钥基础设施）
- 实体之间的通信保持机密且不受干扰

零信任框架通过利用强大的信任根来创建零信任构成要素：将防篡改信任绑定到实体或流程中是零信任基础的构建要素。接下来，零信任框架需要证明其具有证明、验证和证实实体身份的能力。拿容器服务示例来说，如何检查此容器是否符合其所声称的身份。这就需要通过编排工具来验证，但是要信任该编排工具，我们需要确保其运行不受干扰，而这一点只有在运行受信任的 OS、BIOS 时才能得到保障。证明流程通常也是一个链条。

零信任还需要实体之间的安全通信。虽然网络分段为零信任架构提供了价值，应该予以考虑，但这并非实现零信任的全部解决方案。编排工具网络策略以及服务网格的使用都是综合零信任解决方案的组件。在网上，可以获得有关零信任概念的更多信息。

最小权限

最小权限也很重要，或者，也许是云原生体系结构最重要的一个方面，在做出身份验证或授权决策涉及的堆栈中的所有部分，都应考虑到这一点。传统上，最小权限是在帐户层予以考虑，无论该帐户是人员帐户还是服务帐户。

在云原生环境中，必须在堆栈的每一层中应用最小权限。在评估负责完成每一层执行过程的特定工具时，也应考虑最小权限。在探索各种产品和功能时，组织机构可能会发现，许多容器部署具有默认权限或操作所需的根权限。因此，可能需要采取其他措施来将那些提升的权限与其余工作负载隔离开。组织机构应考虑需要在其工作负载和部署中采用隔离和最小权限的所有领域；从运行时环境中的 cgroup 和系统调用到工件管理和无根构建都应考虑在内。

为了不断减少潜在的攻击面和相应的影响范围，组织机构需要在其体系结构的每个级别上执行最小权限原则。这不仅适用于在其职责范围内执行各种功能的个人，还适用于在给定环境中执行的服务和工作负载。无根服务和容器对于确保如果攻击者确实进入组织机构的环境中，它们便无法轻易地在获得访问权限的容器与底层主机或其他主机上的容器之间穿梭至关重要。

强制访问控制（MAC）的实现（例如，SELinux 和 AppArmor）可以限制为容器或名称空间设置的权限以外的权限，并在主机级别提供额外的容器隔离，以防止容器失陷或从一个容器转移到另一个容器来实现权限提升。

角色与职责

说到云原生体系结构和部署时，组织机构应该期待对传统安全角色和职责进行调整，并创建云特有的新安全角色。随着现代开发方法的迅速兴起以及 IT 活动与业务需求的更好契合，安全性必须是自适应的、与实际风险相称且是透明的。希望开发人员和运维人员成为安全专家是不切实际的。安全从业人员需要与开发人员、运维人员和项目生命周期中的其他人员合作，让安全性和合规性与流程现代化和开发生命周期完全集成。这样做意味着，开发人员通过在习惯性解决问题时使用的工具实时报告调查结果，类似于如何在发现问题后立即解决构建失败问题。

涉及到云原生环境中的安全管理时，DevOps 环境中经常出现界线模糊的问题，但这时仍应坚持明确的职责划分（SoD）。尽管开发人员会更多地参与实施和执行安全措施，但他们无需设置策略，也不必了解角色所不需要的领域等。根据组织机构的风险容忍度和业务实践，组织机构应该按照角色以及不同产品和应用团队进行职责划分。众所周知，在规模较小的组织机构中，一人需要身兼数职，以保持业务蓬勃发展，这时职责划分就变得困难。然而，随着组织机构的不断发展，在认知上迫使个人在活动执行中发生心理变化，这时，进行明确的角色界定有助于实现职责划分。最后，允许将重新确定后的角色重新分配给新的个人，但不会提高新任命的访问权限。

随着产品和服务迁移到云计算上来，组织机构将需要重新评估其资产风险。因为所使用的技术及其部署堆栈的所有权和管理方面发生变化，管理人员应该预料到风险态势会发生重大变化。服务提供商和团队之间需要进行责任共担，这就要求改变风险承受、转移、威胁缓解新机制的阈值。

合规性

在系统设计中纳入一组安全控制措施，来解决法规和合规性指南问题，这样可以让云原生资源更加安全。这样做还可以让相关监管机构和审核机构的认证更加容易，尤其是在系统的设计和规划完成后，可以通过插件模型实现对各种监管机构的自动合规。尽管为了实现合规，通常需要通过满足安全基准来提高安全性和配置管理，例如互联网安全中心（CIS）基准，但必须注意，建议使用机器可读的合规性控制框架和语言。

监管审计

许多金融、医疗、政府和其他实体需要遵守一系列特定的要求才能保护其系统。用户相信系统会保护他们的交互隐私和安全。每个组织机构都应评估适用于它们的法规标准（例如，PCI-DSS、HIPAA、FedRAMP、GDPR 等）。然后，它们应该确定这些特定要求如何应用于其云原生系统以及如何实现这些标准的实际实施。在可能的情况下，这种证明遵守特定标准的证据收集机制要有不可否认性保证，以实现自动化。

角色和用例

重点是尽可能地关注安全性、保护、检测和自动响应。重点也不一定只是开发工具，而是可以透明地集成到开发流程中的安全工具，以便在收到快速反馈、采取及时的补救措施后能够执行安全策略。有关云原生安全用例的详细信息，请参阅 SIG-Security 用例列表。

行业

企业

企业采用云原生模型的核心关注领域是在满足业务目标的同时，能够保持当前流程和事项。当在整个组织机构中引入新的标准和实践时，将互操作性、数据丢失或泄漏以及安全风险保持在最低水平。

小企业

小企业采用云原生模型的核心关注领域是专注于是否有能力实现短期目标并促进创新以应对激烈竞争。但由于资源、预算、技术深度和最佳实践，这阻碍了它们适应云原生解决方案的能力。小企业需要使用可重复的模式、降低 IT 影响来解决这些挑战。

金融

金融行业是否能够成功采用云原生应用的核心关注领域是，未经授权的信息披露、欺诈和资金可用性。欺诈会直接影响到资金的可用性，使得金融交易的完整性变得至关重要。

医疗

医疗行业是否能够成功采用云原生应用的核心关注领域是，未经授权的信息披露、及时性和记录的可用性以及准确性。由于医疗行业的性质和惯例，记录及其相关内容的可用性是做出医疗决策的基础。在没有此类信息的情况下，需要进行新的记录。

学术与教育

对于教育机构而言，成功采用云原生应用的核心关注领域可能取决于目标最终用户。对面向未成年人的机构可能有其他的法律要求，以保护未成年人的机密，这时，访问控制就变得至关重要。除此之外，教育机构还应把重点放在是否有教育内容可供最终用户使用。

公共部门

公共部门的组织机构是否能够成功采用云原生应用的核心关注领域是安全性、数据主权、合规性和供应商锁定。其面临的障碍来自为保护公众利益而制定法规的机构。在公共部门，保持公众与政府实体之间的和谐与信任至关重要。此外，是否能够及时部署并提供相关功能也可能是一个重要的考虑因素。采用云原生应用以及现代方法可以提高组织速度，这对于公共部门的许多领域都尤为重要。

云原生安全的演变

容器技术是一个不断发展的领域，使用日益普及。云原生技术所面临的威胁以及在减少和解决这些威胁时面临的相应安全挑战也在不断发展。除了安全的容器平台的复杂生态系统外，还需要制定周密、经过深思熟虑的安全策略，以及技术控制和自动化措施，用于安全策略执行、响应和运维领域。

如果使用正确，容器将带来巨大的安全优势。容器的透明度和模块化程度更高，攻击面更小，应用组件的更新也更为便捷，并为应用组件提供了统一的运行环境。正是由于这种统一性，才能够在开发、测试和生产运行时环境中实现并行安全的蓬勃发展。在应用之间建立适当的隔离（实际上是在可能使用实行扁平网络的企业中实现微隔离分），作为分层纵深防御安全策略的一部分时，容器还可以减少企业级安全事件的影响。

面对当前安全方面的所有挑战，所需安全工具的数量以及市场上技能和人才的短缺，确保容器平台的安全是一项巨大的挑战。随着云提供商的容器服务产品变得越来越成熟，并且通过兼容规范与更多的云原生安全工具和智能工具集成在一起，我们预计将有更多的人使用云技术。作为责任共担模型的一部分，这些产品减少了企业的开销。

因此，采用容器以及云原生将继续推动企业的数字化转型。企业已经在某些服务上使用了无服务器架构和设计，但是考虑到要将多个功能编排成一个业务功能会降低可见性，以及目前尚不清的诸多安全隐患，使用无服务器架构来构建整个业务功能仍在发展之中。简而言之，随着服务提供商安全控制措施像现有容器生态系统一样降低了消费者的开销，预计在云原生架构中采用无服务器架构将随着时间的推移而增加。

然而，威胁态势依然如故，同一批威胁主体不断在利用最主要的弱点进行攻击。我们看到，最明显的变化体现在攻击者攻击云原生组织和应用的方式和机制上。针对容器编排工具和部署的攻击也在增加，通过镜像渗透或特洛伊木马镜像进行的加密挖矿攻击也在增加。就像任何创新技术开始达到市场饱和一样，恶意主体利用一些容易利用的漏洞只是时间问题。

随着攻击变得更加普遍、更加复杂，且不断增多，云原生安全必须发展，将其更多的关注点放在企业和 DevOps 团队上，而不是裹足不前。我们看到，安全策略即代码越来越受欢迎，但是在安全策略的执行、检测和响应方面还有很大的发展空间和提高自动化的空间。显然，即时而自动的安全情报和响应对于阻止攻击甚至是自我修复至关重要。甚至可能在攻击出现时调整和集成防回归[10]措施。

容器取证工具及技术也需要不断发展，才能跟上云原生技术的发展步伐。这一点尤其重要，因为

[10]防回归的概念可以解释为技术环境中抗脆弱性行为的一个方面。在面临不利条件和攻击时，不是让技术保持韧性和强健，相反地，当面临不利条件和受攻击时，技术可以主动适应并蓬勃发展。

在基础架构即服务和其他即服务模型的背景下，事件的数量和复杂性都在增加。

总结

在过去的十五年中，云原生社区的云服务和技术快速普及，并且最近在云原生模型方面取得了重大进展。与安全行业中的任何新商品一样，创新机构也在不断推动技术向前发展，以便尽早得到应用和测试。

处于技术鸿沟边缘的组织机构，或者处于技术早期的绝大多数组织机构，必须认真分析并应用核心安全概念，以减轻加固和环境控制方面的滞后，这一点非常重要。

尽管对于当今和未来的大多数创新来说，可能还没有特定的安全指南和控制措施，但在设计、开发和部署新功能时，可以坚持在应用云原生体系结构中落实核心安全概念。

这些核心安全概念包括：

- 防止未经授权的访问（个人和非个人实体）——短暂性，通过不断地对已知的良好状态进行调整，可以减少未经授权实体对资产的接触。
- 不变性，保持内容和代码的完整性。
- 服务、工具和内容的可用性——分发可提供弹性和冗余。
- 审计和责任认定——提供一种机制，以确保未发生任何违规情况，并跟踪已授权的变更。

缩略词和词汇表

ABAC — 基于属性的访问控制
IAM — 身份和访问管理
RBAC — 基于角色的访问控制
SOC — 安全运营中心
IaC — 基础架构即代码
CI — 持续集成
CD — 持续部署
HSM — 硬件安全模块

参考文献

NIST 800-204 基于微服务的应用系统的安全策略
NIST 800-190 应用容器安全指南
https://www.cisecurity.org/benchmark/Kubernetes/
威胁建模：12 种可用方法
https://owasp.org/www-community/Application_Threat_Modeling
NIST 应用容器安全指南、互联网安全中心(CIS)、NIST 微服务安全策略及 OpenSCAP 基准均适

用于 Docker、Kubernetes 及多个托管 Kubernetes 发行版本。
MITRE ATT&CK 矩阵（Kubernetes）

致谢

本白皮书是 CNCF Security-SIG 成员共同努力的成果。感谢大家的杰出贡献。特别感谢 Emily Fox 和 Jeyappragash JJ。

编著：

- Aradhna Chetal - TIAA
- Brandon Lum - IBM
- Chase Pettet - Mirantis (Chase.Pettet@mirantis.com)
- Emily Fox - US National Security Agency (NSA)
- Gadi Naor - Alcide
- Harmeet Singh - IBM
- Jeff Lombardo - Independent
- Jeyappragash JJ - Tetrate IO
- Pushkar Joglekar - Visa
- Rowan Baker - ControlPlane
- Andrew Martin - ControlPlane
- Trishank Karthik Kuppusamy - Datadog
- Vinay Venkataraghavan -Prisma Cloud (Palo Alto Networks)
- Wayne Haber - GitLab
- Mark Bower
- Alex Chircop - StorageOS

审核：

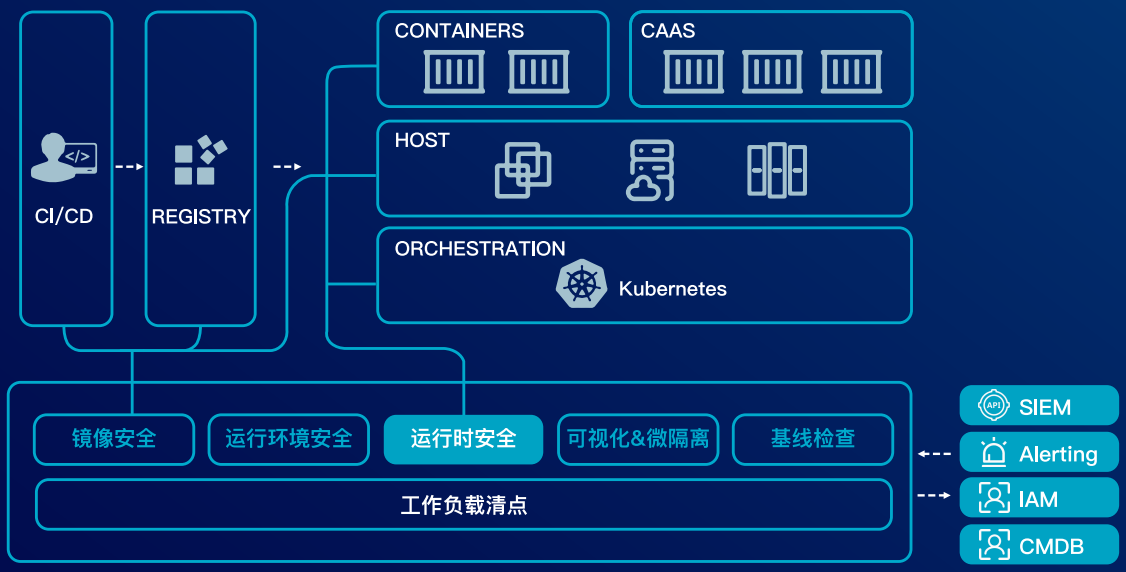
- @justincappos
- @lumjjb
- @whaber
- @craigbox
- @anvega
- @magnologan
- Alok Raj - XenonStack (alok@xenonstack.com)
- @nyrahul
- @ranio1

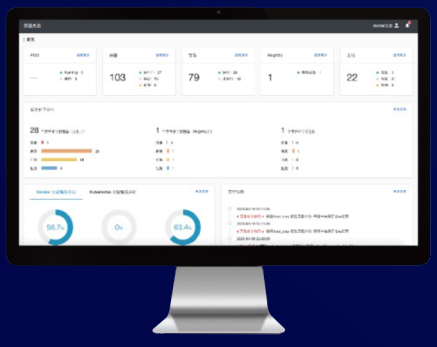
关于青藤蜂巢·容器安全平台

青藤蜂巢专注于容器安全领域，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。

产品架构

青藤蜂巢与云原生环境深入集成；提供工作负载清点、镜像安全、运行环境安全、运行时安全、可视化&微隔离、基线检查等功能，实现容器安全预测、防御、检测和响应的安全闭环。





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