CSA GCR cloud security alliance GREATER CHINA REGION

CSA cloud security alliance®

实现安全应用容器架构的最佳实践

在实现可信任安全系统时融入集成应用容器安全的考量

摘要

在DevOps等一些敏捷软件开发方法的设计、开发和部署应用阶段往往会采用应用容器和微服务架构。这些软件开发方法里需要嵌入安全。本文从开发者、运营者和架构师的视角，提出了一些建议和最佳实践，解决在可信赖的安全系统工程中保护应用程序容器所面临的挑战。

序言

为深入贯彻落实国家关于建设网络强国、数字中国、智慧社会的战略部署，在各行业主管部门的监督和政策指引下，信息化水平快速提升，构筑全方位的网络安全体系也成为网络安全核心任务。

众所皆知，云计算/云原生技术因能极大地提高云上资源利用率以及应用交付效率而被广泛采用。然而，云计算/云原生技术的发展也让用户遭受了更多高级威胁与攻击。如何构建有效的云原生安全管理体系应对层出不穷的安全威胁这一问题也一直受到千行百业用户的关注与热议。

此次发布的《实现安全应用容器架构的最佳实践》充分考虑了这些年安全应用容器架构的技术发展和行业需求，更好地满足数字化安全的未来发展。

如今，在DevOps等一些敏捷软件开发方法的设计、开发和部署应用阶段往往会采用应用容器和微服务架构。这些软件开发方法里需要嵌入安全。安全的DevOps要求安全人员、开发者和运营者协同工作，方可设计、构建出值得信赖的安全系统。

本文从开发者、运营者和架构师的视角着手，通过通俗易懂的语言提出了一些紧贴落地实践的建议和最佳实践（包括了跨环境的代码加固、主机安全、来自平台/主机的容器持续性监控、容器间的网络通信、验证镜像的完整性和安全性、容器取证、平台管理、容器管理、容器加密等17个方面），希望读者朋友在阅毕后可对安全应用容器架构有更为清晰的理解，能更快、更好地开展应用实践。

李雨航 Yale Li

CSA 大中华区主席兼研究院院长

致谢

《实现安全应用容器架构的最佳实践（Best Practices for Implementing a Secure Application Container Architecture）》由CSA工作组专家编写，CSA大中华区秘书处组织翻译并审校。

中文版翻译专家（排名不分先后）：

组　长：刘文懋

翻译组：陈俊杰　郭嘉伟　刘连杰　刘文懋　李娜容
马维士　申屠鹏会　杨玉欢　郑剑锋

审校组：王　亮　陆　琪　刘文懋　姚　凯

研究协调员：刘连杰

感谢以下单位的支持与贡献：

安易科技（北京）有限公司　北京华云安信息技术有限公司
北京网御星云信息技术有限公司　绿盟科技集团股份有限公司
厦门服云信息科技有限公司　深信服科技股份有限公司
腾讯云计算（北京）有限责任公司　中国工商银行股份有限公司

英文版本编写专家

应用容器和微服务工作组主席包括：

Anil Karmel　Andrew Wild

主要作者：

John Kinsella　Cem Gurkok　Frank Geck

参与编著者：

Jeff Barnes	Randall Brooks	Ramaswamy Chandramouli	Madhav Chablani
Atul Chaturvedi	Aradhna Chetal	Joshua Cuellar	Joshua Daniel
Shyamkant Dhamke	Michele Drgon	Michele Drgon	Frank Geck

Michael Green	Cem Gurkok	Amir Jerbi	Amir Jerbi
Juanita Koilpillai	Yin Lee	Aaron Lippold	Vishwas Manral
James McCloskey	James McCloskey	Lloyd Osafo	Lloyd Osafo
Alex Rebo	Michael Roza	Ed Santiago	Ed Santiago
Shankar	Ken Stavinoha	Shanthi Thomas	David Wayland
Shawn Wells	John Wrobel	Mark Yanalitis	John Osborne
Ashish Kurmi	James Yaple	Vrettos Moulos	

CSA人员：

Hillary Baron　Marina Bregu

在此感谢以上专家。如译文有不妥当之处，敬请读者联系CSA GCR秘书处给与雅正!

联系邮箱：research@c-csa.cn；国际云安全联盟CSA公众号。

目录

内容提要

根据NIST SP 800-180中的定义，应用容器和微服务架构用于设计、开发和部署应用程序。因为利用了敏捷的软件开发方法，如DevOps，所以应用组件的安全需要纳入软件全生命周期中考虑。NIST 800-160系统安全工程，定义了基于广泛利益相关者需求的值得信赖的安全系统需求 。

本文旨在成为《保护应用容器和微服务所面临的挑战》文档的配套文档。《保护应用容器和微服务所面临的挑战》文档从开发者、运营者和架构师的视角定义了保护应用容器和微服务所面临的挑战，而本文则提供了应对挑战的建议和最佳实践。

本文包含的建议和最佳实践源自信息安全、运营、容器开发和微服务方面拥有丰富知识和实践经验的不同团队的广泛合作。这些建议和最佳实践是面向开发者、运营者和架构师的。

1 介绍

1.1 目的和范围

本文目的是提供实现安全应用容器架构的最佳实践和建议，适用于各种角色，包括开发者、运营者和架构师。保护DevOps的安全要求安全人员、开发者和运营者协同工作，设计出值得信赖的安全系统。

此文的范围仅限于应用容器。

1.2 文档结构

此文档由以下部分和附录组成：

- 第2节介绍应用容器
- 第3节总结保护应用容器安全的建议和最佳实践
- 附录A提供此文档中常见缩略词
- 附录B提供从文档中选出来的词汇表
- 附录C提供此文档中引用的文献列表

1.3 读者

此文档面向应用开发者、应用架构师、系统和安全管理者、安全项目经理、信息系统安全官和其他相关责任人或对软件开发生命周期中的应用容器安全感兴趣的人员。

本文假设读者具有一定操作系统、网络和安全方面的专业知识，以及应用容器、微服务和敏捷应用程序开发方法(如DevOps)方面的专业知识。由于应用容器技术不断变化的特性，我们鼓励读者利用其他资源(包括本文中列出的资源)获取更新的更详细的信息。

2 应用容器

正如在《保护应用容器和微服务的挑战》一文所述，利益相关方的责任是保护所有的应用容器和微服务，而这些容器和服务具有独特的特质，因而会带有不同的安全后果。因此，应用容器和微服务的特质给开发者、运营者和（微服务的）架构师所带来的挑战已被识别。最佳实践中存在的其他挑战在下表列出。

表1-应用程序容器的最佳实践

最佳实践	视角
3.1.1 跨环境（开发、质量保证、测试、生产）的代码提升	
开发者应该建立信任基本源。	开发者
运营者应利用容器管理平台安全地跨环境移动镜像。	运营者
3.1.2 主机安全	
必须通过容器技术（例如标签）记录多租户或敏感数据需求，允许自动化部署和审计容器。	运营者
容器运行时与远程服务（容器仓库、容器编排）间必须使用加密通信。	运营者
除非绝对必要，容器不能运行在特权模式下。	运营者
除非绝对必要，容器运行时仅允许容器内部挂载数据卷（例如不挂载/proc、/etc、运行时套接字等）。	运营者
用户应移除所有功能，仅显式保留进程必需项。使用白名单而非黑名单机制。	运营者
3.1.3 平台或宿主机侧的容器持续监控	
开发者需要使用清晰的版本模式识别运行容器	开发者

中的应用程序版本。	
运营者应配置容器运行时向中心化日志系统传输日志。	运营者
对于运营者，监控服务的容器不能运行在特权模式下，也不能访问宿主机。	运营者
3.1.4 容器网络：宿主机与容器通信	
建议应用程序使用安全的网络通信协议。	开发者/运营者
运营者应提供鉴别和鉴权的基础设施。	运营者
运营者应提供网络监控的基础设施。	运营者
3.1.5 容器网络：容器间通信	
应用程序必须使用安全的网络通信协议。	开发者/运营者
运营者必须提供鉴别和鉴权的基础设施。	运营者
运营者必须提供网络监控的基础设施。	运营者
运营者必须使用容器化解决方案支持不同粒度的开放服务访问控制。	运营者
3.1.6 验证镜像的完整性和安全性	
开发者应在镜像构建过程中对镜像签名，并在使用前验证镜像。	开发者
开发者应在开发过程中使用漏洞扫描工具。	开发者/运营者
开发者应在镜像中仅包含必要的组件。	开发者
运营者应在基础设施设计中确保仅允许使用已签名和已验证的镜像。	运营者
运营者应在基础设施设计中保障可持续识别新	运营者

公布的漏洞。

3.1.7 容器取证

对于新应用程序，开发者应在计划和设计阶段建立和保持包含日志功能在内的编码策略。	开发者
对于现有应用程序，开发者应实现捕获应用程序日志（从鉴别日志开始）的计划。	运营者
运营者应将容器环境的取证功能引入规划。	运营者
对于新的基础设施，运营者应实现规划好的取证功能。	运营者
对于现有的基础设施，运营者应从捕获数据（即网络流量、磁盘和内存数据）开始，迭代输出新的功能。	运营者

3.1.8 跨容器信任链

开发者应确保镜像签名是构建镜像过程的一部分，并在使用前验证镜像。	开发者/运营者
开发者应在开发过程中使用漏洞扫描工具	开发者/运营者
开发者应确保在镜像中仅包含必要的组件。	开发者
运营者应根据3.1.6 (验证镜像的完整性和安全性)的建议，设计镜像完整性校验、漏洞扫描和修补漏洞的基础设施。	运营者
运营者应使用安全加固过的、安全配置过容器引擎的可信宿主机存储镜像或运行容器。	运营者
运营者应使用鉴别和鉴权机制（即双向认证的TLS），确保容器间、容器与管理平台间的可信通信。	运营者

运营者应基于用户角色限制容器和镜像的访问。	运营者
运营者应监控容器配置参数和运行时的完整性。	运营者
3.1.9 容器卷管理	
开发者应得到足够的培训，确保开发应用程序时尽可能不出现使用共享容器卷和非必要的宿主机目录访问。	开发者
运营者应提供支持资源管理、访问控制和监控容器卷的基础设施。	运营者
运营者应提供隔离不同用户和不同应用程序间网络通信的基础设施。	运营者
运营者应提供接口支持同一实体（如工作负载）的容器间通信。	运营者
3.1.10 容器密钥管理	
为开发者提供培训和最佳实践指南。	开发者
为开发者建立通用软件库，处理后端应用程序代码中的敏感数据和密钥。	开发者
运营者应通过集成部署工具或脚本，以及容器编排工具设计密钥管理架构。	运营者
3.1.11 平台管理—生命周期事件通知	
容器的生命周期（启动/停止/扩展）由CMP管理。从开发者视角，如果容器化的应用程序无法感知容器的状态变化，应用程序将会处于非安全状态。	开发者

3.1.12 平台管理—资源请求

开发者应通过平衡系统资源，确保可持续的系统性能。	开发者
在多租户的环境中，运营者应确保平衡的资源占用。	开发者
在多租户的环境中，运营者应确保CMP优化集群的资源消耗。	运营者

3.1.13 平台管理—容器资源管理

运营者应找到和实现CMP与单个应用程序资源管理目标间可接受的平衡。	运营者

3.1.14 容器管理—扩展容器资源

当建立容器部署配置时，开发者应利用CMP的资源控制特性，协调容器内部资源利用率、优先级和分配阈值。	开发者
运营者需要与开发者协作以理解容器化应用程序的资源需求，合理设置约束，监控可能的性能挑战。	运营者

3.1.15 容器管理—数据备份和复制

运营者必须确保数据备份系统可以在容器销毁前备份容器中的任何新数据。	运营者

3.1.16 容器管理—CMP间的容器重部署

对于使用敏感数据的传统应用程序的运营者来说，需要建立和管理容器在重新部署状态下，自动导出和导入敏感数据的机制。	运营者

3.1.17 容器加密	
开发者与运营者应一致使用标准、通用的身份鉴别系统。	开发者/运营者
开发者需要识别可使用的静态数据解决方案。考虑该方案是否已被一个公正的第三方验证，以及该方案是否在期望的运行环境中可行。	开发者
开发者应加密敏感的应用程序，然后建立容器入口（entrypoint）应用程序解密和执行主程序。	开发者

3 应对应用容器挑战的缓解措施

3.1.1 跨环境的代码提升

针对跨环境（开发、质量保证、测试、生产）的代码提升所固有的挑战，建议的缓解措施包括以下几点：

1. 开发者应建立信任根

公钥基础设施（PKI）允许创建和管理用于加密和数字签名的数字证书。组织中可能已经存在PKI——由此系统产生的证书可以用于容器镜像的签名，以及将来确认容器镜像来源和完整性的验证。

为了确保构建链的完整性，开发者需具备对在环境之间流转或流转到下一个阶段的代码、二进制文件进行数字签名并提供签名信息的能力。

通过用PKI的证书对容器镜像签名，CMP的用户可以很容易地验证镜像的来源。这一点很重要，因为一旦确定了镜像的来源，用户就可以确保该镜像在被推到生产环境使用之前已经经过了适当的审查。

2. 运营者应利用容器管理平台安全地跨环境移动镜像。

在一个生产级别的容器环境中，除了管理应用程序代码和容器镜像外，还需要管理

许多组件。CMP如能支持组织在一个或多个环境中伸缩，应具备安全同步容器镜像的能力。

为了确保构建链的完整性，运营者必须有能力利用开发者或编排引擎提供的数字证书和/或数字签名对跨环境和跨阶段使用的代码或二进制文件进行验证和确认。

为了确保由构建链部署的应用程序的一致性，运营者还必须有能力在应用程序运行或被使用的所有环境中同步应用程序代码。

通过使用CMP管理容器镜像，平台用户可以相信有合适的容器镜像版本适用于他们的环境。这最大限度地减少了在不同环境中运行的应用程序的版本混乱。当结合使用了信任根（root of trust）的最佳实践时，平台的用户就可以确保容器镜像在环境中的来源和完整性。

3.1.2 主机安全

针对主机安全所面临的挑战提出的缓解措施包括以下观点：

1. 多租户或数据敏感性需求必须通过容器技术（如标签）记录，实现容器部署和审计的自动化。

标签对于区分一个容器或一组容器内存储或处理的数据类型是非常重要的。这些标签可以被CMP用于将容器自动放置到与其中存储的数据的安全状况匹配的环境中。这些标签还具有允许审计引擎验证容器内数据的功能。

通过对容器应用标签，CMP可以确保容器放置在具有适当安全性的主机集群上，保护存储在所述容器中的数据。

标签可以为审计引擎提供上下文，并对宿主机安全状况的审查提供信息，确保此宿主机上的容器处理的数据得到保护。

2. 容器运行时与远程服务（容器仓库、容器编排）间必须使用加密通信。

任何由一个以上的主机组成的容器生态系统都需要节点之间的通信管理容器和资源。加密可以确保这种通信的网络传输的保密性。有些容器运行时默认启用了加密功能；有些则没有，但仍提供对加密传输的支持。

虽然应用程序的网络传输的安全性是留给开发者的一个挑战，但提供一个安全的容器托管平台需要运营者确保CMP内的通信不会被恶意实体所操纵。加密通信可以解决这一难题。

3. 除非绝对必要，容器不能运行在特权模式下。

给容器授予管理员级别的权限，实质上是授予容器对容器宿主机的管理员级的访问。这就要求有特权的容器最终被主机和可能在该主机上运行的所有其他容器所信任。这种授予必须在容器启动时向容器运行时提出请求；一些CMP现在默认不允许有特权的容器。

通过禁止这种特权授予，管理员可以减少特权容器被破坏的风险，从而减少主机及其他主机上的容器被破坏的风险。通过默认禁止运行特权容器，组织不得不考虑在特权模式下运行某个容器的风险和回报。

4. 容器运行时应该只在绝对必要时挂载容器内的数据卷（例如，不要挂载/proc、/etc、运行时套接字等）。

作为对一些需要特权访问的容器的部分缓解措施，一些应用容器被设计成需要从主机系统读取和/或读写文件或访问目录。虽然这比授予容器管理权限要好一些，但这种方法已被证明有允许恶意用户获得对主机系统的非预期访问的缺陷。

通过禁止对主机文件系统和服务的访问，主机/容器资源实现分离，主机的安全得到保证。

5. 用户应该删除所有功能，仅显式保留进程必需项。使用白名单而非黑名单机制。

默认情况下，容器启动时的Linux功能是受限的。使用这些功能，进程不再需要以root权限在特定领域内运行。容器支持通过添加或删除功能覆盖默认值，可能会由于增加的功能而使容器变得不那么安全。

在通常需要root权限的地方限制Linux内核功能（不需要进程以root身份运行），可以减少攻击面，保证主机安全。

在“传统”主机加固指南中，运营者了解哪些应用程序正在主机上运行以及为达到加固状态需要重新配置的参数。对于应用容器的主机，运营者不一定提前知道在主机上运行的应用，所以加固过程必须考虑可能在主机上运行的应用容器以及它们将合法消耗

的资源。

3.1.3 平台或宿主机侧的容器持续监控

在平台或宿主机侧的容器持续监控中，推荐采取这些缓解措施应对挑战。

1. 开发者需要使用清晰的版本模式以识别运行容器中的应用程序版本

通过定义和使用一个已接受的版本计划，开发团队可以将版本号作为构建过程的一部分自动实现。通过将该版本应用于容器名称、标签或标识，开发者就能轻松识别容器中运行的应用程序版本。

2. 对运营者而言，应配置容器运行时，向中心化日志系统传输日志

容器的短生命周期特性提高了集中式日志的需求；当容器被终止时（无论是由于故障还是用户的目的），CMP会迅速从主机上清理容器部件。为了在容器运行时和退出后都能查看日志，日志需要被运送并存储到一个安全的集中位置。

容器的审计和应用日志可以在容器运行时和终止后提供故障排除和取证价值。有必要信任这些日志，因为它们可能是容器终止后留下的唯一标识。由于这种需要，这些日志需要以安全、可信任的方式存储。

由于应用程序是由更多的细化组件组成的，对于运营者或开发者来说，排除故障会变得更加困难。运营者还需要以一致的时间戳提供所有可使用的日志。

通过提供一个具有安全存储途径的集中式日志系统，运营者既能实现安全的日志存储，又能在监控、故障排除或取证时检索和/或审查应用程序或审计日志。

3. 对于运营商而言，监控服务的容器不能运行在特权模式下，也不能访问宿主机

默认情况下，容器运行时不会授予运行中的容器对主机和其他容器的完全访问权限。然而用户或第三方在某些情况下会建议以 "特权 "模式运行他们的容器，这样他们就可以轻松访问系统资源。这样做的风险是，容器可能获得对资源的非预期的访问权限，或者如果容器被破坏，恶意行为者将获得对主机的访问，从而获得对其他容器和资源的访问。

例如，CSP上的两个租户可能是一个组织内的不同部门或公有云中的两个不同客户。

在任何情况下，租户务必不能访问对方的数据或元数据。这给运营者带来的挑战是，要确保在多租户环境中不能运行特权容器，因为特权容器会允许跨租户访问。此外，以root权限运行的容器在单租户场景中也可能会带来安全问题。

通过禁止特权授予，管理员可以降低特权容器未经授权访问资源或数据的风险。当默认情况下不允许时，组织就不得不考虑以特权模式运行特定容器的风险和/或回报。

3.1.4 容器网络-宿主机与容器通信

建议通过以下措施缓解宿主机与容器通信的挑战：

1. 建议应用程序使用安全的网络通信协议

通过使用适当的加密协议(即SSL/TLS、IPSec)保证容器产生的网络流量在应用和管理面的机密性。

选择支持安全通信的库提供加密并保持通信的机密性，防止信息泄漏和网络通信内容被篡改。网络通信被篡改可能会导致系统受损。

2. 运营者应提供鉴别和鉴权的基础设施

运营者不仅应提供加密通信的途径，还应鉴别参与者的身份，同时限制其可以采取的行动。

使用双向鉴权的TLS等解决方案可以确认通信的机密性和参与者的身份。鉴权机制将通过基于角色的访问控制（RBAC）如LDAP，来提供，它利用身份鉴别机制认证身份。

3. 运营者还应提供网络监控的基础设施

监控网络通信和配置提供了监测网络通信和配置信息被篡改，以及潜在恶意通信的途径。

提供网络传输（即网络流量）和配置更改的记录可确保在给定环境中只发生预期的行为和操作。这有助于维持事件响应和监测能力。

3.1.5 容器网络-容器间的通信

针对容器间通信存在的挑战，提出以下建议的缓解措施：

1. 应用程序必须使用安全的网络通信协议

通过使用适当的加密协议(即SSL/TLS、IPSec)保持应用程序和管理间通信网络传输的机密性。

选择支持安全通信的库来提供加密并保持通信的机密性，防止信息泄漏和网络通信内容被篡改。网络通信被篡改可能会导致系统受损。

2. 运营者必须提供鉴权与授权基础设施

运营者不仅应提供加密通信的途径，还应鉴别参与者的身份，同时限制其可以采取的行动。

使用双向身份鉴权（authentication）的TLS等解决方案可以确认通信的机密性和参与者的身份。授权（authorization）机制将通过LDAP等基于角色的访问控制（RBAC）提供，利用鉴权机制识别身份。

3. 运营者必须提供网络监控基础设施

监控网络通信和配置提供了监测网络通信和配置信息被篡改，以及潜在恶意通信的途径。

通过为容器定义网络监控的标准语义和语法，指定“必须报告”以达到合规性要求的事件，并为容器解决网络活动/事件与网络健康状况的准确性或相关性，可以缓解网络监控方面的挑战。

提供网络流量记录（即netflow）和配置更改的记录可确保在给定环境中只发生预期的行为和操作。这有助于维持事件响应和检测能力。

4. 运营者必须使用容器化解决方案支持不同粒度的开放服务访问控制。

运营者应提供支持不同粒度的开放服务访问控制。对容器使用细粒度访问控制等解决方案可以降低安全风险。

3.1.6 验证镜像的完整性和安全性

针对验证镜像的完整性和安全性方面存在的挑战，建议的缓解措施包括：

1. 开发者应在镜像构建过程中对镜像签名，并在使用前校验镜像

镜像应当作为构建过程的一部分签名，并在使用前校验。签名和验证可以通过镜像内容上的（GPG）签名或通过类似方式实现。

在镜像生成时对其内容进行数字签名，并在使用前验证签名数据，确保镜像的数据在构建阶段和运行阶段之间不会被篡改。

2. 开发者应在开发过程中使用漏洞扫描工具

开发者应使用漏洞扫描工具作为开发过程和CI管道的一部分，并将漏洞评估集成到构建过程中。如果漏洞评估失败，他们也会考虑让构建失败。一旦发现漏洞，应修补程序漏洞并重建镜像。

漏洞扫描将识别并警示使用中的具有已知安全漏洞的第三方组件。将漏洞扫描作为开发周期的一部分，可以提高镜像的安全质量，因为在软件进入运行时环境之前，已知的漏洞就会被识别和修补。

3. 开发者应该在镜像中仅包含必要的组件

开发者应该精简镜像，只包含必要的组件。在基线镜像中推荐使用尽可能少的软件包而不是全部操作系统软件包。

从镜像中删除不必要的组件将减少可能过期的或未修补的软件包的数量，从而减少安全漏洞的数量。

4. 运营者应在基础设施设计中确保仅允许使用已签名和已验证的镜像。

运营者应设计一个只接受已签名和已验证的镜像的基础架构。镜像验证可以与组织的软件验收政策（例如，质量、安全性、稳定性等）或任何适用的验收标准相关联。运营者应能够阻止虽已正确签名，但在其他方面不符合验收标准的镜像被实例化成容器。

检查签名验证可以确保镜像数据在构建和镜像运行期间不被篡改，为在运行镜像之前防止使用未经授权或不符合验收标准的镜像（即使已签名）进一步做验证。

5. 运营者应在基础设施设计中保障可持续识别新公布的漏洞。

运营者应提供一个基础设施，在后面提及的情况下持续识别镜像漏洞：存储在镜像仓库中的镜像，存储在主机本地的镜像，以及作为容器运行的镜像。一旦发现新的漏洞，应根据漏洞的严重性及其对应用程序的影响采取缓解措施。修复正在运行的容器，不如

在新镜像版本上修补程序，并将修补后的新镜像部署到容器运行时环境。

随着时间的推移，新的漏洞会不断地被发现和公布，因此需要持续扫描镜像。 扫描应包括镜像仓库中的所有镜像、本地计算机上的镜像（等待运行）以及已经作为容器运行的镜像。修补程序通过构建新的镜像版本而非更新正在运行的容器，将提供更好的流程能见度，并确保容器与其镜像之间的一致性。

3.1.7 容器取证

针对容器取证中存在的挑战提出以下几点缓解措施：

1. 对于新应用程序，开发者应创建并遵循编码规范，在规划和设计阶段涵盖日志记录功能

应用程序应提供有关鉴权、授权、操作和失败的日志。开发者应该将此功能作为计划和设计阶段的一部分。

应用程序的鉴权、授权、操作和失败日志，为进行中的调查和根本原因确立提供了证据链。

2. 对于现有应用程序，开发者应该从鉴权日志开始，实现记录应用程序日志的计划

现有应用程序应提供有关鉴权、授权、操作和失败的日志。如果这些日志记录项中的任何一项尚未实现，开发者应在维护阶段提供这些功能。

应用程序的鉴权、授权、操作和失败日志，为进行中的调查和根本原因确立提供了证据链。

3. 运营者应将容器环境的取证功能引入规划

取证功能是任何事件响应和缓解活动的基础。取证可以提供必要的证据确定事件的根本原因，并提高处置速度。取证规划应包括仪器、人力资源、证据源（即网络、磁盘、内存）、流程和程序。容器环境的易变性要求有一个更灵活的框架来捕获和分析证据。

将取证功能集成到事件响应计划和程序中，使获取和处理证据有了途径，减少了用于确定根本原因的时间，并最大限度地降低了系统受损后的风险。

4. 对于新的基础设施，运营者应实现已规划的取证能力

已规划取证能力的设计和实施应与其他基础设施建设一起进行，并应与公司事件响应、业务连续性和灾难恢复计划相结合。

取证能力的设计和实施将有助于验证计划，并在需要适应变化时修改设计。这些能力将提供获取和处理证据的方法，减少确定根本原因的时间，并最大限度地减少系统暴露后的风险。

5. 对于现有的基础设施，运营者应该从捕获数据（即网络流量、磁盘和内存工件）开始，逐步具备相应的能力

将取证能力集成到现有基础设施中应成为运营商维护和变更管理活动的一部分。这应该作为尽职调查和遵守合规的一部分，同时也满足规则和条例的要求。

取证能力的集成将帮助运营者满足尽职调查要求、合规措施、规则和条例。这些能力将提供获取和处理证据的方法，减少确定根本原因的时间，并最大限度地减少系统失陷后的风险。

3.1.8 跨容器的信任链

以下是通过跨容器信任链的建议缓解措施：

1. 开发者应确保镜像签名是镜像构建过程的一部分，并应在使用前验证镜像

镜像签名应作为构建过程的一部分，并在使用前验证。签名和验证可以通过在镜像内容上使用GPG签名或通过类似的方法实现。开发者应确保在构建过程中对镜像签名。运营者应确保在部署之前能够验证镜像签名。

在构建时对镜像内容进行数字签名，并在使用前验证签名数据，确保镜像数据在生成和到运行时这段时间内不会被篡改。

2. 开发者应在开发过程中使用漏洞扫描工具

开发者应使用漏洞扫描工具作为开发过程和CI的一部分，并将漏洞评估集成到构建过程中。如果漏洞评估失败，开发者可能会考虑使构建失败。一旦发现漏洞，他们应该对易受攻击的组件应用进行安全修复并重新构建镜像。

漏洞扫描程序将识别并警告使用具有已知安全漏洞的第三方组件。将漏洞扫描作为

开发周期的一部分，可以提高镜像的安全质量，因为在软件进入运行时环境之前，已知的漏洞就会被识别和修复。

3. 开发者应确保镜像中只包含必要的组件

开发者应该精简镜像，只包含必要的组件。最好使用具有最少软件包集合的基础镜像。

从镜像中删除不必要的组件将减少可能过期或未修复的软件包，从而减少安全漏洞的数量。

4. 在设计镜像完整性验证、漏洞扫描和修复的基础设施时借鉴3.1.6（验证镜像的完整性和安全性）重向运营者的建议

5. 运营者应使用安全加固的受信任的宿主机存储镜像和运行容器，其中宿主机的容器引擎被安全地配置

在存储镜像和启动容器之前，运营者需要加固主机所在的操作系统和容器引擎。只在主机上部署必要的包，并定期应用安全更新和内核补丁，这一点很重要。运营者希望根据供应商的安全最佳实践强化容器引擎。

确保主机操作系统和容器引擎被正确加固和安全配置将减少容器的攻击面和暴露于已知漏洞的情况。

6. 运营者应使用鉴权和授权机制（即基于TLS的双向鉴权），以确保容器本身之间以及管理平台与容器之间的信任

配置具有双向鉴权的容器管理平台和容器端点，端点之间（例如，双向TLS认证）。如果不支持双向鉴权，请考虑使用容器大使模式创建安全的代理通信。

建议对所有通信应用双向鉴权，防止数据篡改、中间人攻击和敏感信息泄露。

7. 运营者应根据用户角色限制对容器和镜像的访问

运营者应实施用户访问控制机制，以确保只有经过鉴权和授权的用户才能访问容器和镜像。实施用户访问控制机制将防止未经授权的用户访问镜像和容器。

8. 运营者应监控容器配置参数和运行时完整性

运营者应监控容器配置中的非安全设置，并应检查正在运行的容器中配置文件和进程的完整性

使用非安全设置运行容器可以获得对主机和网络资源的无限制访问。容器进程和配置文件中的更改可能表明容器已被篡改。

3.1.9 容器卷管理

针对容器卷管理中存在的挑战提出的缓解措施包括以下建议：

1. 开发者应接受足够的培训，确保开发的应用程序能够最大限度地减少使用共享容器卷并且不需要访问主机目录

2. 运营商应提供支持资源管理、访问控制和容器卷监控的基础设施

卷资源的管理包括以下卷管理服务活动：

- 创建、调配和更新卷资源
- 注册、访问和限制卷资源
- 停用和终止卷资源
- 解决卷资源故障
- 监控、报告、审核、清点、验证和验证卷资源
- 迁移卷资源
- 加密卷资源
- 恢复卷资源
- 性能调整卷资源

资产（即人员、组、容器、网络、主机和卷）之间的关系是明确已知的，并映射到卷管理服务活动：

- 容器访问映射到数据卷

  - 资产容器（x）...

- 授予对数据卷的访问权限（x）
- 如果卷位于同一VM主机上，则被授予完全访问权限

- 角色访问映射到数据和操作卷：
  - 卷角色：
    - 数据卷始终映射到专用存储
    - 运行时操作卷
  - 组角色：
    - 开发者角色...
      * 被授予对数据卷的访问权限（x）
      * 被授予对操作卷的访问权限（x）
    - 运营者角色…
      * 被授予对数据卷的访问权限（x）
      * 被授予对操作卷的访问权限（x）
    - 用户角色…
      * 授予对数据卷的访问权限（x）
      * 授予对操作卷的访问权限（x）

具体而言，“资源管理”的粒度细分被确定为对批量资源管理活动进行一致管理的潜在标准实践。

3. 运营者应提供隔离不同用户和应用程序网络流量的基础设施

不同用户和应用程序的网络流量需要隔离，以限制潜在被入侵用户或应用程序的横向移动。网络隔离限制了横向移动的威胁向量。

4. 运营者应提供一个接口，允许一个实体（例如，工作负载）拥有的容器之间进行网络通信

需要一个接口确保包含实体（例如工作负载）的容器能够有效地相互通信。外部实体可以监视此接口，以确保其不被篡改。

与监控平台相结合的接口可确保容器可以进行正常通信，但也可以与其他用户和应用程序隔离。

3.1.10 容器秘密（Secret）管理

在容器秘密管理中，建议针对挑战提出或采用以下缓解措施：

1. 为开发者提供培训和最佳实践指导

后端开发者应该通过培训了解公共云上多租户带来的威胁。他们还需要接受培训获得如何在应用程序中使用秘密管理功能的模板，以避免在应用程序代码中硬编码静态秘密的做法。

从服务器端被认为是可信环境的前提，后端开发者就习惯于将敏感信息编写到服务器代码中。提高对当前公共云环境中存在的细微差别和威胁的认识有助于推动避免此类做法的发生。

2. 为开发者创建公共库，以处理后端应用程序代码中的敏感数据和秘密

以一致的方式处理后端中的敏感数据和秘密应用程序代码，应该提供一组公共的库给开发者。

应用程序开发者不一定是安全性开发者，应该花时间开发应用程序功能，而不是安全性。应用程序中所需的安全性应该易于使用和明确，这样可以尽可能少地投入精力，并实现应用程序之间的一致性。

3. 运营者应通过集成部署工具或脚本以及容器编排工具设计秘密管理体系结构

密码管理系统、部署时间和用户输入的初始种子设定可以通过将其与特定容器编排脚本以及由人工用户触发部署的场景中的相应部署脚本紧密集成的方式处理。在完全自动化部署的情况下，应使用密钥管理服务器（KMS）或硬件安全模块（HSM）引导秘密管理服务器，并提供所需的初始引导秘密。例如，已绑定到特定角色的虚拟机（VM）元数据可用于向HSM进行鉴权，以在VM启动期间从HSM获取秘密。

秘密的自启动总是一个鸡和蛋的问题，利用一些自动生成的数据，例如作为VM元数据，从HSM/KMS获取第一个秘密是克服此问题的一种方法。

一旦秘密管理系统使用部署时间或启动时间进行秘密引导，就可以处理秘密的动态生成。例如，为了动态生成应用程序的数据库帐户凭据，需要使用部署时获得的数据库管理员凭据授权创建新的用户帐户，然后将这些凭据传递给需要使用它的应用程序。

3.1.11 平台管理-生命周期事件通知

针对生命周期事件通知平台管理中存在的挑战提出的缓解措施包括以下建议：

1. 容器的生命周期（开始/停止/伸缩）由CMP管理。从开发者的角度来看，如果容器化应用程序不知道容器的状态转换，那么应用程序可能处于未知状态

CMP应该为封装的应用程序提供一个机会，以执行到已知的安全状态。传统的解决方案是允许将容器生命周期事件通知应用程序。移除容器后释放的资源应以已知状态释放回池，并且应允许容器记录任何状态更改事件。

必须将容器的生命周期事件通知应用程序，以便应用程序可以采取适当的操作来确保安全启动和关闭

3.1.12 平台管理-资源请求

建议针对这些挑战建议采取如下方案：

1. 开发者应通过平衡系统资源帮助确保可持续的系统性能

开发者应与运营者和架构师合作，确保二进制文件和库经过优化以运行在容器化基础设施。

2. 在多租户环境中，运营者应保证资源消耗均衡

应始终以牺牲多租户为代价来维护服务质量，实现至少一项服务水平协议 (SLA)。应该由开发者或供应商，或两方合作，共同优化资源请求。

3. 在多租户环境中，运营者应确保 CMP 优化集群资源消耗

平衡资源消耗有助于可持续执行，从而最大限度地减少资源竞争。

3.1.13 平台管理-容器资源管理

建议针对这些挑战建议采取如下方案：

1. 运营者应在 CMP 和单个应用程序资源管理对象之间找到并实现可接受的平衡。

CMP 应坚持应用程序制定的资源管理策略。将确定的结果写在具有约束力的 SLA 中。

CMP 应尝试在设定的时间间隔内分配预定资源。 反过来，应用程序应该在约定时间以预定义的方式使用这些资源。如果发生违规行为（包括无条件终止申请），一方可以按照一些预订条款处理。

3.1.14 容器管理-容器资源伸缩

针对容器管理中存在的挑战建议采用如下方式伸缩容器资源：

1. 在创建容器部署配置时，开发者或运营者应利用其 CMP 的资源控制功能编排容器内资源利用率、优先级，并分配阈值

应用程序开发者必须接受正在开发的应用程序不能占用生产环境中可用的全部计算资源。即使存在自动伸缩，其功能也有限制，并不会无限扩展。开发者应该期望存在水平基础设施扩展能力，但应该确保应用程序在配置管理团队建立的参数内运行来避免垂直扩展的需要。

开发者必须获取资源利用率和加固参数，并将这些配置整合到软件开发环境和测试脚本中。如果不在定义的生产参数内测试应用程序的功能，应用程序性能下降的风险就会攀升而非下降。根据使用中的 CMP 正确配置资源和安全功能，可确保容器在发生资源故障甚至失控时，不会影响其他平行容器。

2. 运营者需要与开发者合作，了解容器化应用的资源需求，适当设置约束，并监控性能挑战

通过合作，开发者和运营者可以综合了解应用程序的资源需求。 资源消耗是一个运行时事件，但需要在容器启动前配置控制措施，以防止容器失控和耗尽CPU、内存和IO资源。

运营者需要向开发者传达应用程序的当前资源使用情况、安全参数和必要的日志，以便开发者做出合理的测试和QA决策，确保开发中的容器化应用程序可以在给定资源内运行在生产环境中。

3.1.15 容器管理-数据备份与复制

在适用于数据备份和复制的容器管理中，建议采用以下措施应对挑战：

1. 运营者必须确保数据备份系统能够在容器内的任何新数据被销毁之前备份

对于新的容器化应用程序，不应将持久数据存储在容器内。 意识到某些遗留应用程序可能在应用程序容器中运行，运营者必须与开发者合作，了解数据的保存位置，并确保定期备份这些数据，且在容器销毁之前备份这些数据。一些CMP会在容器关闭/销毁之前发送通知；如果可以，应利用此功能来确保在没有备份的情况下不会破坏任何数据。

开发者视角：

1. 正确记录应用程序备份和恢复应用程序的要求。该文档必须包括可以备份和恢复应用程序的状态。对于某些应用程序，利用存储介质的简单快照就可以；而对于其他应用程序，则需要运行特定工具。如果应用程序需要离线或耗尽任何容量的活动用户/会话，则必须记录。
2. 如果应用程序需要离线以进行备份、恢复，那么记录该应用程序上下游依赖的服务至关重要。通过记录这些服务，运维人员可以确保其他容器使用蓝绿部署等技术临时运行，确保不会发生级联故障。
3. 如果开发者对应用的上游或下游依赖有影响，使用云原生最佳实践至关重要。例如，通过在应用程序中内置服务依赖重试（service dependency retries）和幂(idempotency)等实现为恢复而设计的服务。这些最佳实践将确保应用程序在备份/恢复期间不可用时，上下游依赖项将能够恢复并进一步避免级联系统故障。
4. 开发人员需要坚持单服务责任、接口分离、避免传递服务依赖等关键微服务实践。在处理不可变的基础设施时，开发者需要应用网关服务或服务助手模式来将后端操作流程从容器中卸载。开发者应避免依赖 Web 应用程序存档（WAR

包文件）和放置在文件系统中的大量 JAR 文件来增强或塑造客户端应用程序运行时环境。开发者还需要构建能够容忍动态部署模式的应用程序，这些部署模式容易出现可变延迟并且不知道工作负载在哪里执行。

5. 开发者应提供独立于工作负载执行的文件系统管理数据和元数据的软件功能。开发者必须将容器执行视为“处理单元”，消耗的数据存在于容器外部，但可通过逻辑卷引用访问。使用开发者的代码存储库作为还原点的设计是一个错误的假设。构建良好且执行良好的备份API不仅能够恢复数据和元数据；它还应该能够独立于计算堆栈恢复到某个时间点。

运营者视角：

1. 运营者应使用开发者和组织政策规定的标准决定如何在系统上备份和恢复应用程序。这些策略可能包括对备份数量或频率的要求。
2. 对于存储快照足够的简单用例，运营者应根据应用程序和组织要求安排备份。
3. 某些环境可能包括创建卷并将其动态映射到存储设备的能力；这通常称为动态配置。如果运营者正在使用快照策略，则可能无法使用动态配置完成此操作，因为许多存储介质在创建卷之前不允许设置快照计划。如果这不能正确完成，运营者应禁用动态配置。
4. 对于非简单用例，运营者将负责创建可以备份应用程序的安全状态，并负责运行备份程序所需的特定工具。这可能包括在应用程序脱机时建立临时应用程序实例以处理应用程序依赖项的工作负载。这可能不是小事。例如，运营者可能需要：
    a. 通过蓝绿部署，在应用程序脱机时卸载流量。
    b. 为应用程序的开发者或供应商列出备份/恢复创建的安全状态。这可能包括耗尽会话或关闭所有交易。在容器场景中实现这一点可能会用到边车模式（在数据库旁边运行同一容器的代理）或使用其他cron工具。
    c. 需要运行特定工具来备份应用程序。对于Web服务器，这可能需要将所有

事件复制到辅助集群；对于数据库，这可能需要运行特定工具，如果该工具创建了数据存档，则需要将数据安全地备份到存储介质中。这可能需要自定义脚本才能将数据存档移动到安全位置。

3.1.16 容器管理-不同容器管理平台主机间容器迁移

容器管理过程中，为了应对容器在主机间迁移这个挑战，并通过容器管理平台进行容器的主机重新托管，推荐按照以下步骤执行：

1. 作为拥有敏感数据的合法应用运营者，需要做一些工作来创建和管理实用工具，这些工具在主机迁移的过程中可以帮助自动导出或导入与容器有关的敏感数据

目前还没有通用的标准方法在不同的容器管理平台之间导出和导入平台或容器的安全配置。这些数据迁移依赖容器管理平台成熟度或云代理功能规范，同时运营者需要应用管理工具提供容器迁移的能力以及相关的安全组件能力。

这些应对方法依赖于以下手动操作过程：在目标云服务提供商主机上创建具有安全配置的空容器，迁移容器本身，然后在源云服务提供商主机删除容器及其安全配置。手动执行此操作可能会导致意外，因此收集整理一组可以自动执行此过程的实用工具是一项非常值得投入时间的工作。

3.1.17 容器加密

针对容器加密中存在的挑战提出的应对措施包括以下一些建议：

1. 开发者和运营者应同意使用标准的、通用的鉴权系统

通过使用通用的鉴权方法，可跨容器管理平台使用加密密钥，确保鉴权和授权能力的行为一致性。当数据或应用程序在容器中加密时，需要有一种可靠的方法授权使用加密密钥。进一步，应用程序必须支持多种鉴权方法，减少不必要的复杂性。

2. 开发者需要确定他们将使用哪些静态数据（DAR）解决方案，并考虑该解决方案是否经过公正的第三方验证，以及该解决方案在预期的操作环境中是否操作

开发者应尽可能使用经过FIPS 140-2验证的DAR解决方案。开发者还应分析应用程序编程接口（API）和兼容的密钥管理系统（KMS）。使用经过FIPS验证的解决方案和兼容

的KMS，构建于可靠的第三方验证，证明该解决方案是安全的，具有有更高级别的保证。

开发人员要使用最新版AES加密方法，并确保密钥足够长，确保数据的机密性。

3. 对于敏感应用程序，开发者应该加密敏感应用程序，然后通过解密创建容器入口点应用程序，并执行主应用程序

通过加密敏感应用，可以减轻敏感应用的意外暴露的风险。这允许存储、传输或共享容器映像（和敏感应用程序），而不会发生未经授权的泄露。应用程序的加密和解密类似其他类型数据加密/解密过程。

附录A-缩略语

本文件中使用的部分缩略语和缩略语定义如下

ACM	应用容器和微服务
CMP	容器管理平台
CSP	云服务提供商
HSM	硬件安全模块
IAM	身份管理与访问控制
KMS	密钥管理系统
OS	操作系统
RBAC	基于角色的访问控制
SLA	服务水平协议
SSL	安全套接字层
VM	虚拟机

附录B-词汇表

应用容器 [NIST SP800-180]	应用容器是一种在共享操作系统上打包和运营应用或其组件的架构。应用应用容器与其他应用容器隔离，并共享底层操作系统的资源，从而允许在不同的云之间高效重启、纵向扩展或横向扩展应用程序。应用容器通常包含微服务。
容器管理平台	容器管理平台是一个应用程序，旨在管理容器及其各种操作，包括但不限于部署、配置、调度和销毁。
容器生命周期事件	容器生命周期中的主要事件是创建容器、运行容器、暂停容器、取消暂停容器、启动容器、停止容器、重启容器、杀死容器、销毁容器。
开发者 [NIST SP 800-64 Rev 2/ ISO/IEC 17789:2014]	云服务开发者是云服务合作伙伴的子角色，负责设计、开发、测试和维护云服务的实施。这可能涉及从现有服务实施完成当前服务的实施。 云服务开发者的云计算活动包括： • 设计、创建和维护服务组件（条款 8.4.2.1） • 编写服务（条款 8.4.2.2） • 测试服务（条款 8.4.2.3） 注1-云服务集成商和云服务组件开发者描述云服务开发者子角色，其中云服务集成商处理来自其他服务的服务的组成，其中云服务组件开发者处理单个服务组件的设计，创建，测试和维护。 注2-这包括涉及与对等云服务提供商交互的服务实现和服务组件。
主机	支持容器环境的操作系统
横向移动	来自受感染内部主机的横向操作，加强攻击者在组织网络内的

[LIGHTCYBER 2016]	立足点，控制其他计算机，并最终控制战略资产。
微服务 [NIST SP800-180]	微服务是一个基本元素，它是将应用程序组件的体系结构分解为松散耦合的模式的结果，这些模式由自包含的服务组成，这些服务使用标准通信协议和一组定义明确的 API 相互通信，独立于任何供应商、产品或技术。微服务是围绕功能而不是服务构建的，建立在SOA之上，并使用敏捷技术实现。微服务通常部署在应用程序容器内。
企业操作者	负责部署和管理IT服务流程的个人或组织。它们可确保支持向内部和外部客户部署应用程序的基础结构，确保操作环境的平稳运行，包括网络基础结构、服务器和设备管理、计算机操作、IT 基础结构库（ITIL）管理和帮助台服务。[1]
企业架构	负责战略设计建议的个人或组织。他们通过应用云，容器和微服务组件的知识来确定满足业务战略需求的最佳架构。 此外，他们还开发和维护解决方案路线图，并与开发者和操作员合作监督其采用情况，以确保高效和有效的解决方案实施。
容器资源	容器运行所需的四种资源是 CPU、内存（+虚拟交换）、磁盘（空间 + 速度）和网络。
容器资源申请	考虑到资源限制，系统将分配给容器的 CPU、内存（+虚机交换）和磁盘（空间 + 速度）的数量。
容器资源限制	系统将允许容器使用的最大资源量（CPU、内存（+虚机交换）和磁盘（空间 + 速度））。

[1] Wikipedia. Information technology operations. Retrieved from https://en.wikipedia.org/wiki/Information_technology_operations.

附录C-参考

[SP 800-180]	NIST Special Publication (SP) 800-180 (Draft), NIST Definition of Microservices, Application Containers and System Virtual Machines, National Institute of Standards and Technology, Gaithersburg, Maryland, February 2016, 12pp. http://csrc.nist.gov/publications/drafts/800-180/sp800-180_draft.pdf
[SP 800-160]	NIST Special Publication (SP) 800-160, Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems, National Institute of Standards and Technology, Gaithersburg, Maryland, November 2016, 257pp. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160v1.pdf
[SP 800-160]	NIST Special Publication (SP) 800-123, Guide to General Server Security, National Institute of Standards and Technology, Gaithersburg, Maryland, July 2008, 53pp. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-123.pdf
[NIST SP 800-64 Rev 2]	NIST Special Publication (SP) 800-64 Rev 2, Security Considerations in the System Development Life Cycle, National Institute of Standards and Technology, Gaithersburg, Maryland, October 2008, 67pp. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-64r2.pdf
[ISO/IEC 17789:2014]	International Organization for Standardization/International Electrotechnical Commission, Information Technology – Cloud Computing – Reference Architecture, ISO/IEC 17789:2014, 2014. https://www.iso.org/standard/60545.html [accessed 5/11/17]
[LIGHTCYBER 2016]	Cyber Weapons Report 2016, LightCyber, Ramat Gan, Israel, 2016, 14pp. http://lightcyber.com/cyber-weapons-report-network-traffic-analytics-reveal sattacker-tools/ [accessed 5/11/17]