美国国家安全局（NSA）

网络安全与基础设施安全署（CISA）

网络安全技术报告

Kubernetes 加固指南

Translated by青藤云安全

2021年8月

S/N U/OO/168286-21

PP-21-1104

Version 1.0

通知与变更历史

文件变更历史

日期	版本	说明
2021年8月	1.0	初次发布

关于担保和背书的免责声明

本文件中的信息和意见是“按原文”提供的，不代表任何保证或担保。本文件提及的任何具体商标名称、商标、制造商或其他名称下的具体商业产品、流程、或或服务，并不一定构成或暗示美国政府对其的认可、推荐或赞许，而且本指南不得用于广告或产品背书目的。

商标认可

Kubernetes 是 Linux 基金会的注册商标。▪ SELinux 是美国国家安全局的注册商标。▪ AppArmor 是 SUSE LLC 的注册商标。▪ Windows 和 Hyper-V 是微软公司的注册商标。▪ ETCD 是 CoreOS的注册商标。▪ Syslog-ng 是 One Identity Software International Designated Activity 公司的注册商标。▪ Prometheus 是 Linux 基金会的注册商标。▪ Grafana 是 Raintank, Inc.dba Grafana Labs 的注册商标。▪ Elasticsearch 和 ELK Stack 是 Elasticsearch B.V 的注册商标。

版权确认

本文件中的信息、示例和数字基于 Kubernetes 作者以Creative Commons Attribution 4.0 license发布的 Kubernetes 文档。

Translated by 青藤云安全

出版信息

作者

美国国家安全局 (NSA)

网络安全理事会

终端安全

网络安全与基础设施安全署 (CISA)

联系信息

客户需求/一般网络安全咨询：

网络安全需求中心：410-854-4200；Cybersecurity_Requests@nsa.gov

媒体咨询 / 新闻中心：

媒体关系：443-634-0721, MediaRelations@nsa.gov

关于关于事件响应资源，请联系 CISA： CISAServiceDesk@cisa.dhs.gov

宗旨

国家安全局（NSA）和网络安全与基础设施安全署（CISA）制定本文件是为了促进其各自的网络安全，包括其制定和发布网络安全规范和缓解措施的责任。该信息可以广泛分享，以便影响所有适当的利益相关者。

Translated by青藤云安全

执行摘要

Kubernetes® 是一个开源系统，可以自动部署、扩展和管理容器化应用，并且通常托管在云环境中。与传统的单体软件平台相比，使用这种类型的虚拟化基础设施可以提供一些灵活性和安全性方面的效益。然而，安全地管理从微服务到底层基础设施的方方面面，会让事情变得非常复杂。本报告中详述的加固指南旨在帮助企业处理相关风险并享受使用这种技术发展带来的益处。

Kubernetes 中三个常见的入侵来源是供应链风险、恶意行为者和内部威胁。

缓解供应链风险往往很有难度，通常是在容器构建周期或基础设施采购中出现。恶意行为者可以利用 Kubernetes 架构的组件中的漏洞和错误配置，如控制平面、工作节点或容器化应用。内部威胁可以是管理员、用户或云服务提供商。对组织机构的 Kubernetes 基础设施有特殊访问权的内部人员可能会滥用其特权。

本指南描述了与设置和保护 Kubernetes 集群有关的安全挑战，包括加固策略，以避免常见的错误配置，也可指导国家安全系统的系统管理员和开发人员如何部署 Kubernetes，并提供了建议的加固措施和缓解措施的配置示例。本指南详细介绍了以下缓解措施：

- 扫描容器和 Pod 的漏洞或错误配置。
- 以最低权限运行容器和 Pod。
- 使用网络隔离来控制失陷可能造成的损害程度。
- 使用防火墙来限制不必要的网络连接，并使用加密技术来保护机密性。
- 使用强认证和授权来限制用户和管理员的访问，并缩小攻击面。

Translated by 青藤云安全

- 使用日志审计，以便管理员可以监控活动并收到有关潜在恶意活动的警报。
- 定期审查所有 Kubernetes 设置，并通过漏洞扫描来确保考虑到了所有适当相应风险并应用安全补丁。

有关其他安全加固指南，请参见互联网安全中心 Kubernetes 基准、Docker 和 Kubernetes 安全技术实施指南、网络安全和基础设施安全局（CISA）分析报告以及 Kubernetes 文档 [1], [2], [3], [6]。

Translated by青藤云安全

目录

Translated by 青藤云安全

简介

Kubernetes，经常缩写为 "K8S"，是一个开源的容器或编排系统，用于自动部署、扩展和管理容器化应用。它管理着构成集群的所有元素，从应用中的每个微服务到整个集群。与单体软件平台相比，将容器化应用作为微服务，提高了灵活性和安全性，但也可能引入其他风险。

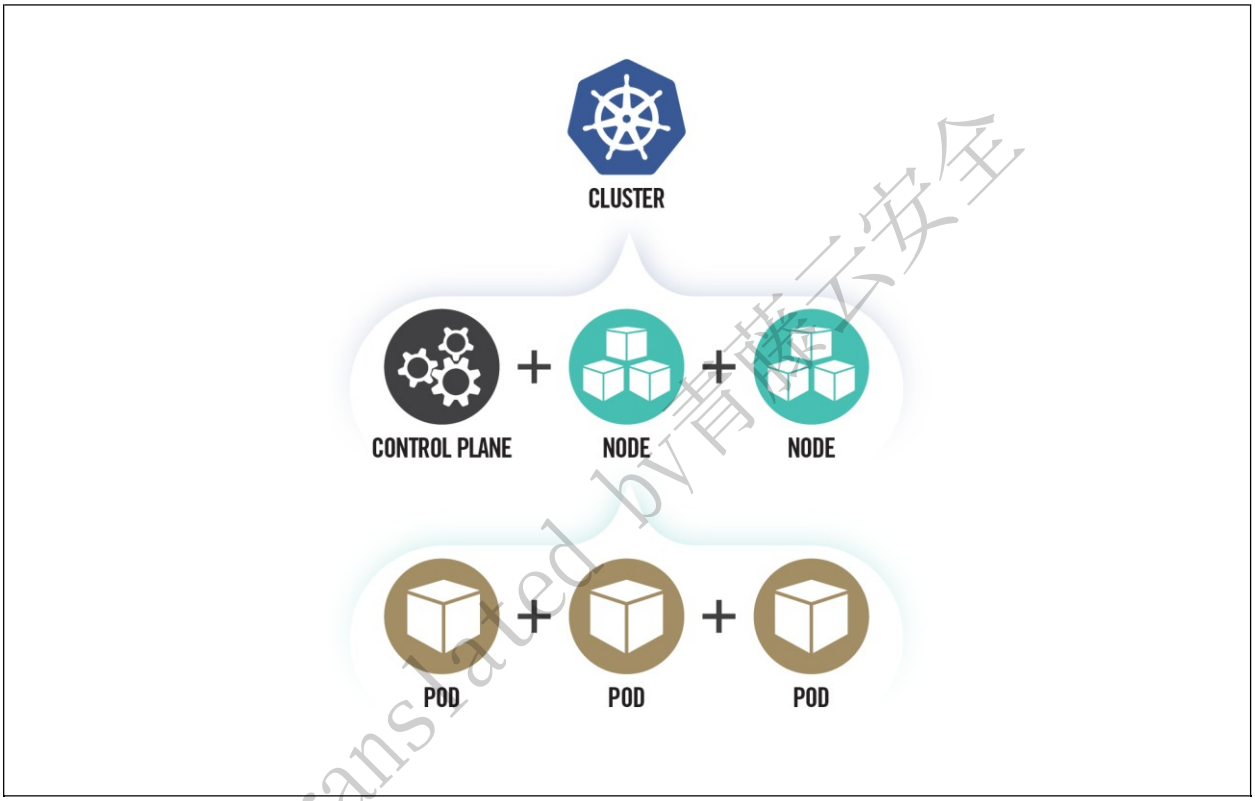


图 1：Kubernetes 集群组件视图

Translated by 青藤云安全

本指南重点关注安全挑战，并尽可能提出适用于国家安全系统和关键基础设施管理员的加固策略。尽管本指南是针对国家安全系统和关键基础设施组织的，但也鼓励联邦和州、地方、地区和部落（SLTT）政府网络的管理员实施所提供的建议。Kubernetes 集群的安全问题可能很复杂，而且经常因错误配置而发生漏洞利用，造成失陷。以下指南提供了具体的安全配置，可以帮助建立更安全的 Kubernetes 集群。

建议

每个部分的主要建议总结如下

- Kubernetes Pod安全
  - 在使用容器时，以非 root 用户身份来运行应用
  - 尽可能用不可变的文件系统运行容器
  - 扫描容器镜像，发现可能存在的漏洞或错误配置
  - 使用 Pod 安全策略强制执行最低水平的安全，包括：
    - 防止出现特权容器
    - 拒绝启用 hostPID、hostIPC、hostNetwork、allowedHostPath 等经常发生漏洞利用的容器功能
    - 拒绝以root用户身份或提升为root用户身份来运行容器
    - 使用SELinux®、AppArmor® 和 seccomp安全服务，加固应用程序，防止发生漏洞利用
- 网络隔离与加固
  - 使用防火墙和基于角色的访问控制（RBAC）锁定对控制平面节点的访问
  - 进一步限制对 Kubernetes etcd 服务器的访问
  - 配置控制平面组件，使用传输层安全（TLS）证书进行认证、加密通信
  - 设置网络策略来隔离资源。除非执行其他隔离措施（如网络策略），否则不同命名空间的 Pod 和服务仍然可以相互通信
  - 将所有凭证和敏感信息放在 Kubernetes Secrets 中，而不是放在配置文件中，并用增强式的加密方法对 Secrets 进行加密
- 认证与授权
  - 禁用匿名登录（默认启用）
  - 使用增强式用户认证
  - 制定 RBAC 策略，限制管理员、用户和服务账户活动
- 日志审计
  - 启用审计记录（默认为禁用）

Translated by青藤云安全

  - 在出现节点、Pod或容器级故障的情况下，持续保存日志以确保可用性
  - 配置一个 metric logger
- 升级和应用安全实践
  - 立即应用安全补丁和更新
  - 定期进行漏洞扫描和渗透测试
  - 不再需要某些组件时，将其从环境中移除

架构概述

Kubernetes 采用的是集群架构。Kubernetes 集群由一系列控制平面和一个或多个物理或虚拟机组成（称为工作节点）。工作者节点承载 Pod，其中包含一个或多个容器。容器是包含软件包及其所有依赖项的可执行镜像。**见图2：Kubernetes 架构。**

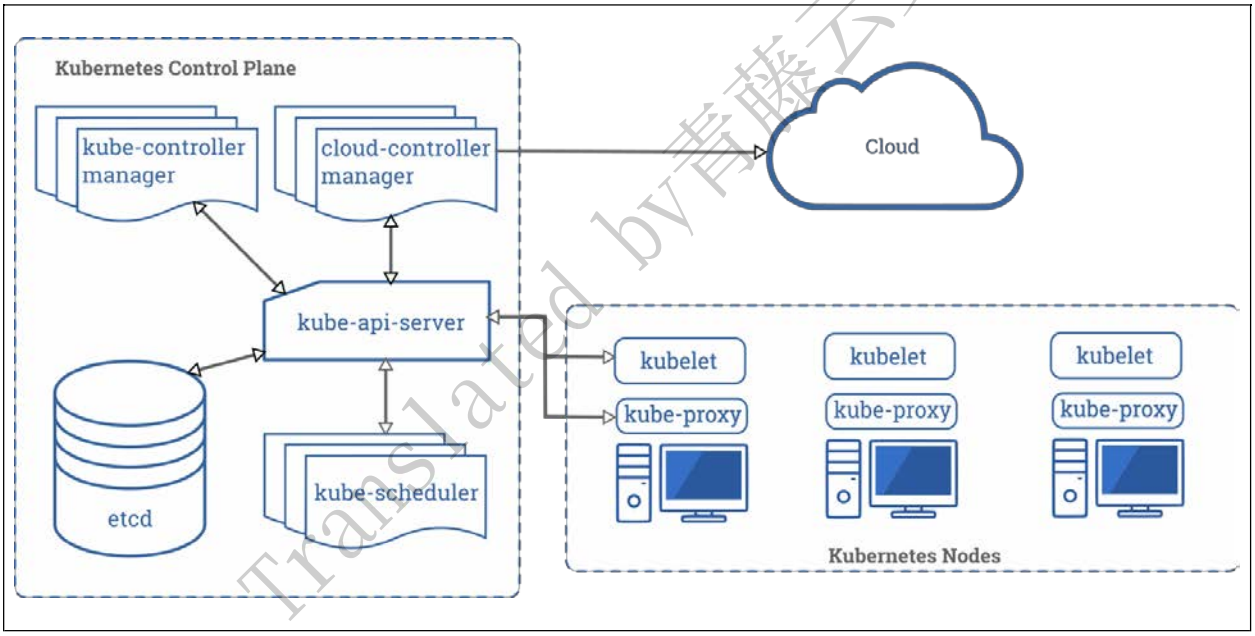


图2：Kubernetes 架构

[1] *Kubernetes Components* by *SupriyaSurbi* and *Fale* used under *CC BY 4.0*

Translated by青藤云安全

控制平面对集群进行决策。这包括调度容器的运行，检测故障情况并对故障做出响应，并在部署文件中指定的副本数没有满足要求时启动新的 Pod。控制平面包含以下逻辑组件：

- **Controller manager（默认端口：10252）**——监控 Kubernetes 集群，以便检测和维护 Kubernetes 环境，包括将 Pod 加入到服务中，确保一组Pods数量正确，并对节点丢失做出响应。
- **Cloud controller manager（默认端口：10258）**——这是一个可选组件，用于基于云的部署。云控制器与云服务提供商接口，以管理集群的负载均衡器和虚拟网络。
- **Kubernetes API Server（默认端口：6443 或 8080）**——管理员操作 Kubernetes 的接口。因此，API 服务器通常暴露在控制平面之外。API 服务器是可以扩展的，可以在多个控制平面节点上都有。
- **Etcd®（默认端口范围：2379-2380）**——用于持久化的备份存储，所有关于集群状态的信息都保存在这里。不建议直接操作 Etcd，而应该通过 API 服务器来管理。
- **Scheduler（默认端口：10251）**——追踪工作节点的状态并决定在哪里运行 Pod。只有控制平面内的节点可以访问Kube-scheduler。

Kubernetes 工作节点是专门为集群运行容器化应用的物理机或虚拟机。除了运行容器引擎外，工作节点还承载了负责从控制平面进行协调的以下两个服务：

- **Kubelet（默认端口：10250）**——运行在每个工作节点上，负责协调和验证 Pod 的执行情况。
- **Kube-proxy**——这是一个网络代理，使用主机的数据包过滤能力，确保 Kubernetes 集群中数据包的正确路由。

集群通常托管在云服务提供商（CSP）的 Kubernetes 服务中或企业内部。在设计 Kubernetes 环境时，组织机构应了解他们在安全维护集群方面的责任。虽然 CSP 管理大部分的 Kubernetes 服务，但组织机构也需要负责维护某些内容，如认证和授权。

Translated by 青藤云安全

威胁模型

Kubernetes 可以成为数据窃取和 / 或计算能力窃取的重要目标。虽然数据盗窃是传统上的主要动机，但寻求计算能力（通常用于加密货币挖掘）的网络攻击者也被吸引到 Kubernetes 来利用其底层基础设施。除了窃取资源，网络攻击者还可能针对 Kubernetes 造成拒绝服务。下面的威胁代表了 Kubernetes 集群最可能的入侵来源。

- **供应链风险**——对供应链的攻击向量是多种多样的，并且很难缓解。供应链风险是指攻击者可能颠覆构成系统的任何元素的风险，包括帮助提供最终产品的产品组件、服务或人员。这可能包括用于创建和管理 Kubernetes 集群的第三方软件和供应商。供应链的潜在威胁会在多个层面上影响 Kubernetes，包括：
  - **容器 / 应用层面**——在 Kubernetes 中运行的应用及其第三方依赖项的安全性，这依赖于开发者的可信度和开发基础设施的防御能力。第三方的恶意容器或应用可以为网络攻击者在集群中提供一个立足点。
  - **基础设施**——托管 Kubernetes 的底层系统有其自身的软硬件依赖项。系统作为工作节点或控制平面一部分，任何潜在威胁都可能为网络攻击者在集群中提供一个立足点。
- **恶意威胁行为者**——恶意行为者经常利用漏洞从远程位置获得访问权限。Kubernetes架构暴露了几个 API，网络攻击者有可能利用这些 API 进行远程漏洞利用。
  - **控制平面**——Kubernetes 控制平面有各种组件，通过通信来跟踪和管理集群。网络攻击者经常会利用缺乏适当访问控制措施的控制平面组件。
  - **工作节点**——除了运行容器引擎外，工作者节点还承载着 kubelet 和 kube-proxy 服务，这些都有可能被网络攻击者利用。此外，工作节点存在于被锁定的控制平面之外，可能更容易被网络攻击者访问。

Translated by 国家安全局

  - **容器化应用**——在集群内运行的应用是攻击者的常见目标。通常在集群之外就可以访问应用，这就让远程网络攻击者也可以接触到。然后，网络攻击者可以从已经失陷的应用出发，或者利用应用内部可访问的资源在集群中提升权限。
- **内部威胁**——威胁行为者可以利用漏洞或使用个人在组织机构内工作时获得的特权。组织机构内部的个人被赋予特殊的知识和特权，可能会给 Kubernetes 集群造成威胁。
  - **管理员**——Kubernetes 管理员对运行中的容器有控制权，包括在容器化环境中执行任意命令的能力。Kubernetes 强制执行的 RBAC 授权可以通过限制对敏感能力的访问来降低风险。然而，由于 Kubernetes 缺乏双人制的完整性控制措施，即必须有至少一个管理账户才能够获得集群的控制权。管理员通常有对系统或管理程序的物理访问权，这也可能会对 Kubernetes 环境造成入侵。
  - **用户**——容器化应用的用户可能有知识和凭证来访问 Kubernetes 集群中的容器化服务。这种程度的访问可以让其有足够多的手段对应用本身或其他集群组件进行漏洞利用。
  - **云服务或基础设施供应商**——对管理着 Kubernetes 节点的物理系统或管理程序的访问权限可被用来入侵 Kubernetes 环境。云服务提供商通常有多层技术和管理控制措施，保护系统免受特权管理员的危害。

Translated by 青藤云安全

Kubernetes Pod 安全

Pod 是 Kubernetes 中最小的部署单元，由一个或多个容器组成。Pod 通常是网络攻击者在进行容器漏洞利用时的初始执行环境。鉴于此，应该加固Pod，让网络攻击者难以进行漏洞利用，并限制成功入侵所造成的影响范围。

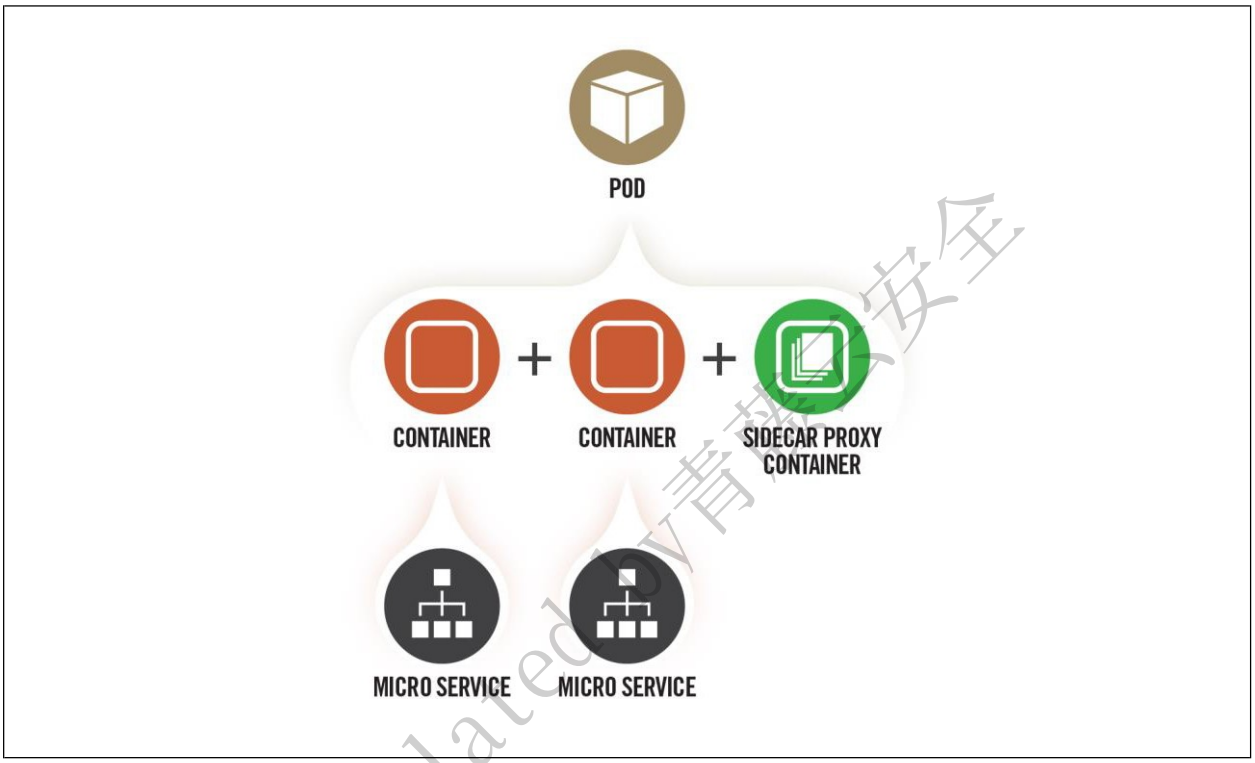


图3：Pod 组件（以sidecar代理作为日志容器）

“非 root”容器和“无 root”容器引擎

默认情况下，许多容器服务以有特权的 root 用户身份运行。即便应用程序不需要以有特权的用户身份运行，但在容器内仍会以 root 用户身份运行。

可以使用非 root 容器或无 root 容器引擎来防止以 root 用户身份运行容器行，从而限制容器失陷带来的影响。这两种方法都会对运行时环境产生重大影响，因此应该对应用进行全面测试，确保兼容性。

非 root 容器：容器引擎允许容器以非 root 用户和非 root 组成员身份运行应用。通常，这种非默认设置是在构建容器镜像时配置的。**《附录 A：非 root 应用的 Dockerfile 示例》**展示了一个 Dockerfile 示例，它就是以非 root 用户身份运行应用的。或者，Kubernetes 可以在 SecurityContext:runAsUser 指定一个非零用户的情况

Translated by 青藤云安全

下，将容器加载到 Pod 中。虽然 runAsUser 指令在部署时可以有效地强制以非 root 身份执行，但 NSA 和 CISA 鼓励开发者构建容器应用时，以非 root 用户身份执行。在构建时集成非 root 用户执行，可以更好地保证应用在没有 root 权限的情况下正常运行。

无 root 的容器引擎：一些容器引擎可以在无特权的上下文中运行，而不是使用以 root 身份运行的守护程序。在这种情况下，从容器化应用的角度来看，似乎是使用 root 用户执行，但被重新映射到主机上的引擎用户上下文。虽然无 root 容器引擎增加了一个有效的安全层，但许多引擎目前是作为实验性发布的，不应该在生产环境中使用。管理员应该了解这一新兴技术，并在供应商发布与 Kubernetes 兼容的稳定版本时寻求采用无 root 容器引擎。

不可变容器文件系统

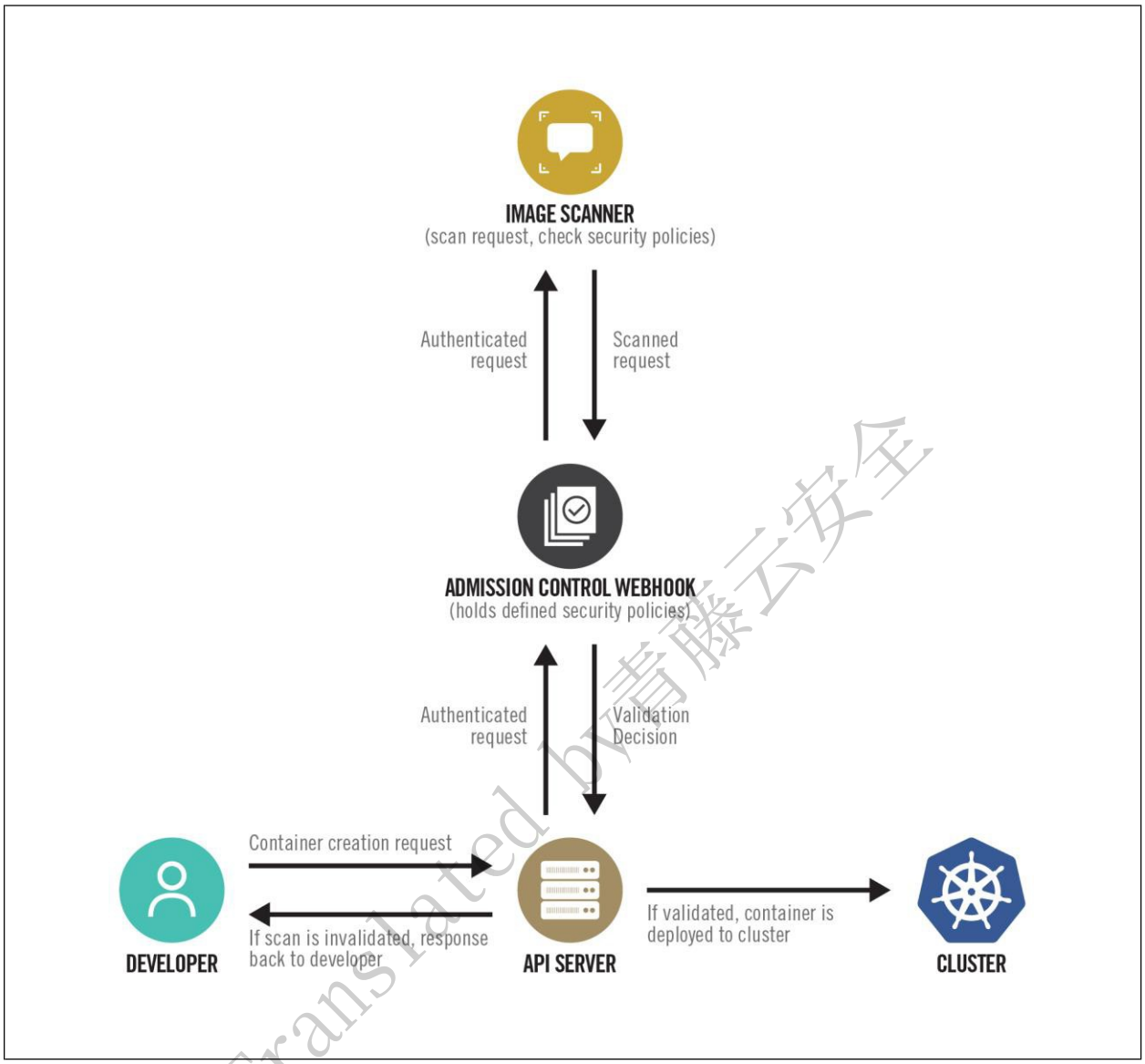
默认情况下，容器在自己的上下文中可以不受限制地执行。在容器中获得执行权限的网络攻击者可以在容器中创建文件、下载脚本和修改应用。Kubernetes 可以锁定一个容器的文件系统，从而防止了发生漏洞利用后的许多活动。

然而，这些限制也会影响合法的容器应用，并可能导致崩溃或异常行为。为了防止损害合法的应用，Kubernetes 管理员可针对需要具有写入权限的应用，在特定目录挂载二级读 / 写文件系统。**《附录 B：只读文件系统的部署模板示例》**展示了一个具有写入权限目录的不可变容器示例。

构建安全的容器镜像

容器镜像通常是通过从头开始构建容器或在从存储库中提取的现有镜像基础上创建的。除了使用可信的存储库来构建容器外，镜像扫描是确保容器安全的关键所在。在整个容器构建工作流程中，应该对镜像进行扫描，识别过时的存储库、已知的漏洞或错误配置，如不安全的端口或权限。

Translated by 青藤云安全



Translated by 青藤云安全

图4：容器构建的工作流程（用 webhook 和准入控制器进行优化）

实现镜像扫描的一种方法是使用准入控制器。准入控制器是 Kubernetes 的原生功能，可以在对象持久化之前、验证和授权请求之后，拦截和处理对 Kubernetes API 的请求。可以实施一个自定义或专有的 webhook，以便在集群中部署任何镜像之前进行镜像扫描。如果镜像不符合 webhook 配置中定义的组织的安全策略，准入控制器可以阻止镜像部署[4]。

Pod 安全策略

Pod 安全策略（PSP）是一个集群策略，它规定了 Pod 在集群内执行的安全要求 / 默认值。虽然安全机制通常是在 Pod/Deployment 配置中指定的，但 PSP 建立了一个所有 Pod 必须遵守的最低安全阈值。

Pod 的创建应遵循最小授权原则。

一些 PSP 字段提供默认值，在 Pod 的配置省略某个字段时使用。其他 PSP 字段用于拒绝创建不符合要求的 Pod。PSP 是通过 Kubernetes 准入控制器执行的，所以 PSP 只能在 Pod 创建期间执行要求。PSP 并不影响已经在集群中运行的 Pod。

PSP 是很有用的技术控制措施，可以在集群中强制执行安全措施。PSP 对于由具有分层角色的管理员管理的集群特别有效。在这些情况下，顶级管理员可以制定默认值，对低层级的管理员强制执行要求。NSA 和 CISA 鼓励企业根据自己的需要调整**《附录 C：Pod 安全策略示例》**中的 Kubernetes 加固 PSP 模板。下表介绍了一些广泛适用的 PSP 组件。

表 1: Pod 安全策略组件[2]

字段名称	使用方法	建议
privileged	控制 Pod 是否可以运行特权容器。	设置为false。
hostPID, hostIPC	控制容器是否可以共享主机进程命名空间。	设置为false。
hostNetwork	控制容器是否可以使用主机网络。	设置为false。
allowedHostPaths	将容器限制在主机文件系统的特定路径上。	使用一个“假”路径名称（比如 /foo 标记为只读）。省略这个字段是不对容器准入做出限制。
readOnlyRootFilesystem	需要使用一个只读的根文件系统。	尽可能设置为true。
runAsUser, runAsGroup, supplementalGroups, fsGroup	控制容器应用是否能以 root 权限或 root 组成员身份运行。	将 runAsUser 设置为 MustRunAsNonRoot。 将 runAsGroup 设置为非零（参见《附录 C：Pod 安全策略》中的示例）。

[2] https://kubernetes.io/docs/concepts/policy/pod-security-policy

Translated by青藤云安全

字段名称	使用方法	建议
		将 supplementalGroups 设置为非零（见附录 C 的示例）。将 fsGroup 设置为非零（参见《附录 C：Pod 安全策略》中的示例）。
allowPrivilegeEscalation	限制提升到 root 权限。	设置为false。为了有效地执行 runAsUser: MustRunAsNonRoot 设置，必须采取这一措施。
seLinux	设置容器的 SELinux 上下文。	如果环境支持 SELinux，可以考虑添加 SELinux 标签，进一步加固容器。
AppArmor 注释	设置容器所使用的 AppArmor 配置文件。	在可能的情况下，通过 AppArmor 加固容器化应用，限制漏洞利用。
seccomp注释	设置用于沙箱容器的 seccomp 配置文件。	在可能的情况下，使用 seccomp 审计配置文件来确定运行应用所需的系统调用；然后启用 seccomp 配置文件来阻止所有其他系统调用。

注意：由于以下原因，PSP 不会自动适用于整个集群：

- 首先，在应用 PSP 之前，必须为 Kubernetes 准入控制器启用 PodSecurityPolicy 插件，这是 kube-apiserver 的一部分。
- 第二，策略必须通过 RBAC 授权。管理员应从其集群组织内的每个角色中验证已实施的 PSP 的正确功能。

在有多个 PSP 的环境中，管理员应该谨慎行事，因为 Pod 的创建会遵守最小限制性授权策略。以下命令展示了给定命名空间的所有 Pod 安全策略，这有助于确定存在问题的重叠策略。

```
kubectl get psp -n <namespace>
```

保护 Pod 服务账户令牌

默认情况下，Kubernetes 在创建 Pod 时自动提供一个服务账户，并在运行时在 Pod 中挂载该账户的密钥令牌。许多容器化应用不需要直接访问服务账户，因为 Kubernetes 的编排工作是在后台公开进行的。

Translated by 青藤云安全

如果一个应用发生了入侵，Pod 中的账户令牌都可能被网络攻击者收集走并用于进一步破坏集群。当应用不需要直接访问服务账户时，Kubernetes 管理员应确保在 Pod 规范中禁用正在加载的密钥令牌。这可以通过 Pod 的 YAML 规范中的 automountServiceAccountToken: false 指令来完成。

加固容器引擎

一些平台和容器引擎提供了其他方案来加固容器化环境。一个有力证明就是使用管理程序来提供容器隔离。管理程序依靠硬件而不是操作系统来确定虚拟化边界。管理程序隔离比传统的容器隔离更安全。在 Windows® 操作系统上运行的容器引擎可以配置为使用内置的 Windows 管理程序 Hyper-V®来增强安全性。此外，一些注重安全的容器引擎将每个容器部署在一个轻量级的管理程序中，以实现纵深防御。由管理程序支持的容器可以减少容器失陷。

Translated by青藤云安全

网络隔离与加固

集群网络是 Kubernetes 的一个核心概念，其中必须要考虑到容器、Pod、服务和外部服务之间的通信。默认情况下，几乎没有网络策略来隔离资源，防止集群失陷时的横向移动或权限提升。资源隔离和加密是限制网络攻击者在集群内移动和提升权限的有效方法。

要点

- 使用网络策略和防火墙来隔离资源。
- 确保控制平面的安全。
- 对流量和敏感数据（例如 Secret）进行静态加密。

Translated by 青藤云安全

命名空间

Kubernetes 命名空间是在同一集群内的多个个人、团队或应用之间划分集群资源的一种方式。默认情况下，命名空间不会自动隔离。但命名空间确实为一个范围分配了一个标签，这可以用来通过 RBAC 和网络策略指定授权规则。除了网络隔离之外，策略可以限制存储和计算资源，以便在命名空间层面上对 Pod 进行更有效的控制。

默认有三个命名空间，并且是不能被删除的：

- kube-system（用于 Kubernetes 组件）
- kube-public（用于公共资源）
- default（针对用户资源）

用户 Pod 不应该放在 kube-system 或 kube-public 中，它们是专为集群服务预留的。可以用 YAML 文件（如**《附录 D：命名空间示例》**中所示）来创建新的命名空间。除非有其他隔离措施，如网络策略，否则不同命名空间中的 Pod 和服务仍然可以相互通信。

网络策略

网络策略控制 Pod、命名空间和外部 IP 地址之间的流量。默认情况下， Pod 或命名空间没有网络策略，导致 Pod 网络内的入口和出口流量不受限制。通过适用于 Pod 或 Pod 命名空间的网络策略，可以对 Pod 进行隔离。一旦根据网络策略选中了一个 Pod，它就会拒绝适用策略对象所不允许的任何连接。

网络策略检查清单

- ✓ 使用支持 NetworkPolicy API 的 CNI 插件
- ✓ 制定策略，使用 podSelector 和/或 namespaceSelector 来选择 Pod
- ✓ 使用默认策略拒绝所有入口和出口流量。确保未选择的 Pod 与除 kube-system 之外的所有命名空间隔离
- ✓ 使用 LimitRange 和 ResourceQuota 策略来限制命名空间或 Pod 层面的资源

要创建网络策略，需要一个支持 NetworkPolicy API 的网络插件。使用 podSelector 和 / 或 namespaceSelector 选项来选择 Pod。**《附录 E：网络策略示例》**中展示了一个网络策略示例。网络策略的格式可能有所不同，这取决于集群使用的容器网络接口（CNI）插件。管理员应该使用选择所有 Pod 的默认策略来拒绝所有入口和出口流量，并确保将任何未选择的 Pod 隔离开来。对于允许的链接，可采取其他策略放松限制。

外部 IP 地址可以使用 ipBlock 在入口和出口策略中使用，但不同的 CNI 插件、云提供商或服务实现可能会影响 NetworkPolicy 处理的顺序和集群内地址的重写。

资源策略

除了网络策略，LimitRange 和 ResourceQuota 是两个可以限制命名空间或节点资源使用的策略。LimitRange 策略限制了特定命名空间内每个 Pod 或容器的单个资源，例如，强制执行最大计算和存储资源。每个命名空间只能创建一个 LimitRange 约束，如**《附录 F：LimitRange 示例》**中所示。Kubernetes 1.10 及以上版本默认支持 LimitRange。

Translated 腾讯云

与 LimitRange 策略不同的是，ResourceQuotas 是对整个命名空间的资源使用总量的限制，例如对 CPU 和内存使用总量的限制。如果用户试图创建一个违反 LimitRange 或 ResourceQuota 策略的 Pod，则 Pod 创建失败。**《附录 G： ResourceQuota示例》**中展示了一个 ResourceQuota 策略的示例。

控制平面加固

控制平面是 Kubernetes 的核心，让用户能够在集群中查看容器，调度新的 Pod，读取 Secret，执行命令。由于这些敏感的功能，控制平面应受到高度保护。除了 TLS 加密、RBAC 和强认证方法等安全配置外，网络隔离有助于防止未经授权的用户访问控制平面。

Kubernetes API 服务器运行在 6443 和 8080 端口上，这些端口应该有防火墙加以保护，只接受预期内的流量。默认情况下，8080 端口可以在没有 TLS 加密的情况下从本地机器访问，请求绕过认证和授权模块。

保护控制平面安全的步骤

1.设置TLS加密

2.设置强认证方式

3.禁止访问互联网和不必要的或不受信的网络

4.使用RBAC策略限制访问

5.使用身份验证和 RBAC 策略保护 etcd 数据存储

6.保护kubeconfig文件不会进行未经授权的修改

不安全的端口可以使用 API 服务器标签 --insecure-port=0 来禁用。Kubernetes API 服务器不应该暴露在互联网或不受信的网络中。网络策略可以应用于 kube-system 命名空间，限制互联网对 kube-system 的访问。如果对所有命名空间实施默认的拒绝策略，kube-system 命名空间仍然必须能够与其他控制平面和工作节点进行通信。

下表列出了控制平面的端口和服务。

表 2：控制平面端口

协议	方向	端口范围	目标
TCP	Inbound	6443 或 8080 （如果没有禁用）	Kubernetes API server
TCP	Inbound	2379-2380	etcd server client API
TCP	Inbound	10250	kubelet API
TCP	Inbound	10251	kube-scheduler
TCP	Inbound	10252	kube-controller-manager
TCP	Inbound	10258	cloud-controller-manager (optional)

Translated by青藤云安全

Etcd

etcd 后端数据库存储着状态信息和集群 Secret。它是一个关键的控制平面组件，如果网络攻击者获得对 etcd 的写入权限，就获得了对整个集群的 root 权限。Etcd 只能通过 API 服务器访问，因为集群的认证方法和 RBAC 策略可以限制用户访问。etcd 数据存储可以在一个单独的控制平面节点上运行，允许防火墙限制对 API 服务器的访问。

管理员应该设置 TLS 证书，强制执行 etcd 服务器和 API 服务器之间的 HTTPS 通信。etcd 服务器应被配置为只信任分配给 API 服务器的证书。

etcd 后端数据库是一个关键的控制平面组件，也是集群中最重要的安全部分。

Kubeconfig 文件

kubeconfig 文件包含关于集群、用户、命名空间和认证机制的敏感信息。Kubectl 使用存储在工作节点和控制平面本地机器 $HOME/.kube 目录下的配置文件。网络攻击者可以利用对该配置目录的访问，获得并修改配置或凭证，从而进一步破坏集群。应该对配置文件加以保护，以防止意外变更，还应阻止未经认证的非 root 用户访问这些文件。

工作节点划分

工作节点可以是一个虚拟机或物理机，这取决于集群的实现。由于节点运行微服务并承载集群的网络应用，因而往往成为被攻击的目标。如果一个节点失陷，管理员应将该工作节点与其他不需要与该工作节点或 Kubernetes 服务通信的网段隔离，主动缩小攻击面。防火墙可用于将内部网段与面向外部的工作节点或整个 Kubernetes 服务分开，这取决于网络的情况。需要与工作节点的可能攻击面分离开来的内容包括机密数据库以及不需要互联网访问的内部服务。

下表列出了工作节点的端口和服务：

Translated by 腾讯云安全

表 3：工作节点端口

协议	方向	端口范围	目的
TCP	Inbound	10250	kubelet API
TCP	Inbound	30000-32767	NodePort Services

加密

管理员应配置 Kubernetes 集群中的所有流量 —— 包括组件、节点和控制平面之间的流量（使用 TLS 1.2 或 1.3 加密）。

加密可以在安装过程中设置，也可以在安装后使用 TLS 引导（详见 Kubernetes 文档）来创建并向节点分发证书。对于所有的方法，必须在节点之间分发证书，以便安全地进行通信。

Secrets

Kubernetes Secret 维护敏感信息，如密码、OAuth 令牌和 SSH 密钥。与在 YAML 文件、容器镜像或环境变量中存储密码或令牌相比，将敏感信息存储在 Secrets 中的访问控制效果更好。默认情况下，Kubernetes 将 Secret 存储为未加密的 base64 编码字符串，任何有 API 权限的人都可以检索到。可以通过对 secret 资源实施 RBAC 策略来限制访问。

可以通过在 API 服务器上配置静态数据加密或使用外部密钥管理服务（KMS）来对 Secrets进行加密，KMS服务可以由云提供商提供。要启用使用 API 服务器的 Secret 数据静态加密，管理员应修改 kube-apiserver 清单文件，使用 --encryption-provider-config 参数执行。**《附录 H：加密示例》**中显示了一个 encryption-provider-config 的示例。

使用 KMS 提供商可以防止原始加密密钥存储到本地磁盘上。如果通过 KMS 提供商来加密 Secret，应该在encryption-provider-config 文件中指定 KMS 提供商，如**《附录 I：KMS 配置示例》**中所示。

默认情况下，Secret 存储为未加密的 base64 编码的字符串，任何有 API 权限的人都可以检索到。

Translated by 青藤云安全

在应用了 encryption-provider-config 文件后，管理员应该运行以下命令来读取和加密所有的 Secret。

```
kubectl get secrets --all-namespaces -o json | kubectl replace -f -
```

保护敏感的云基础设施

Kubernetes 通常部署在云环境中的虚拟机上。因此，管理员应该仔细考虑 Kubernetes 工作节点所运行的虚拟机的攻击面。在许多情况下，在这些虚拟机上运行的 Pod 可以在不可路由的地址上访问敏感的云元数据服务。这些元数据服务为网络攻击者提供了关于云基础设施的信息，甚至可能是云资源的短期凭证。网络攻击者滥用这些元数据服务进行权限提升[5]。Kubernetes 管理员应通过使用网络策略或通过云配置策略防止 Pod 访问云元数据服务。由于这些服务根据云供应商的不同而不同，管理员应遵循供应商的指导来加固这些访问对象。

Translated by青藤云安全

认证和授权

认证和授权是限制访问集群资源的主要机制。如果集群配置错误，网络攻击者可以扫描常用的 Kubernetes 端口，不需要经过认证即可访问集群的数据库或进行 API 调用。但用户认证不是 Kubernetes 的一个内置功能。然而，有几种方法可以让管理员在集群中添加认证。

认证

Kubernetes 集群有两种类型的用户：服务账户和普通用户账户。服务账户代表 Pod 处理 API 请求。认证通常由 Kubernetes 通过 ServiceAccount Admission Controller 使用接口令牌自动管理。接口令牌安装在 Pod 中的常用位置，如果令牌不安全，集群外的用户也可能会使用。正因如此，应限制有查看需求的人按照 Kubernetes RBAC 的要求才能访问 Pod Secret。对于普通用户和管理员账户，没有自动的用户认证方法。管理员必须在集群中添加一个认证方法来实现认证和授权机制。

Kubernetes 假设由一个独立于集群的服务来管理用户认证。Kubernetes 文档中列出了几种实现用户认证的方法，包括客户端证书、接口令牌、认证插件和其他认证协议。至少应该采用一种用户认证方法。若实施多种认证方法，第一个成功认证请求的模块会缩短评估的时间。

管理员不应使用静态密码文件等弱认证方式。弱认证方式可能会导致网络攻击者冒充合法用户进行认证。

管理员必须在集群中添加一个认证方法来实现认证和授权机制。

匿名请求是被其他配置的认证方法拒绝的请求，与任何个人用户或 Pod 无关。在一个设置了令牌认证并启用了匿名请求的服务器中，没有令牌的请求将作为匿名请求执行。在 Kubernetes 1.6 及以上的版本中，匿名请求是默认启用的。当启用 RBAC 时，匿名请求需要 system:anonymous 用户或 system:unauthenticated 组的明确授权。匿名请求应该通过向 API 服务器传递 --anonymous-auth=false 选项来禁用。启用匿名请求可能会允许网络攻击者在没有认证的情况下访问集群资源。

Translated by 奇藤云安全

基于角色的访问控制

RBAC 是根据组织机构内个人的角色来控制集群资源访问的一种方法。在 Kubernetes 1.6及以上版本中，RBAC 是默认启用的。要使用 kubectl 检查集群中是否启用了 RBAC，执行 kubectl api-version。如果启用了，应该会列出rbac.authorization.k8s.io/v1 的 API 版本。云 Kubernetes 服务可能有不同的方式来检查集群是否启用了 RBAC。如果没有启用 RBAC，在下面的命令中用 --authorization-mode 标记启动 API 服务器。

```
kube-apiserver --authorization-mode=RBAC
```

留下授权模式标记，如 AlwaysAllow，会允许所有的授权请求，有效地禁用所有的授权，限制了执行最小权限的访问能力。

可以设置两种类型的权限：Roles 和 ClusterRoles。Roles 为特定命名空间设置权限，ClusterRoles 则为整体集群资源设置权限，而不考虑命名空间。Roles 和 ClusterRoles 只能用于添加权限。没有拒绝规则。如果一个集群配置为使用 RBAC，并且禁用了匿名访问，Kubernetes API 服务器将拒绝没有明确允许的权限。**《附录 J：pod-reader RBAC 角色示例》**中显示了一个 RBAC 角色的示例。

一个 Role 或 ClusterRole 确定一个权限，但并没有将该权限与一个用户绑定。RoleBindings 和 ClusterRoleBindings 用于将一个 Roles 或 ClusterRoles 与一个用户、组或服务账户联系起来。RoleBindings通过Roles 或ClusterRoles对特定命名空间中的用户、组或服务账户进行授权。ClusterRoles 是独立于命名空间而创建的，然后可以使用 RoleBinding 限制命名空间的范围，对个人进行授权。ClusterRoleBindings 授予用户、群组或服务账户不同集群资源的 ClusterRoles。**《附录 K：RBAC RoleBinding 和 ClusterRoleBinding 示例》**中展示了RBAC RoleBinding 和 ClusterRoleBinding 的一个示例。

Translated by 青藤云安全

要创建或更新 Roles 和 ClusterRoles，用户必须在同一范围内拥有新角色所拥有的权限，或者拥有对 rbac.authorization.k8s.io API 组中的 Roles 或 ClusterRoles 资源进行权限提升的明确权限。绑定后，Roles 或 ClusterRoles 是不可改变的。若要更换角色，必须删除该绑定。

分配给用户、组和服务账户的权限应该遵循最小权限原则，只给资源授予所必须的权限。用户或用户组可以限制在所需资源所在的特定命名空间内。默认情况下，为每个命名空间创建一个服务账户，以便 Pod 访问 Kubernetes API。可以使用 RBAC 策略来指定允许每个命名空间内服务账户进行哪些操作。对 Kubernetes API 的访问限制是通过创建 RBAC 角色或 ClusterRoles 来实现的，因为这些角色具有完成某个行为所需的适当 API 请求权限和所需的资源。有一些工具可以通过打印用户、组和服务账户及其相关分配的 Roles 和 ClusterRoles 来帮助审计 RBAC 策略。

Translated by青藤云安全

日志审计

日志记录了集群中的活动。很有必要进行审计日志，这不仅是为了确保服务按预期运行和配置，也是为了确保系统的安全。系统性的审计要求包括始终对安全设置进行彻底的检查，以识别潜在威胁。Kubernetes 能够捕获集群操作的审计日志，并监控基本的 CPU 和内存使用信息；然而，它并没有提供深入的监控或警报服务。

要点

- 在创建时建立 Pod 基线，以便能够识别异常活动。
- 在主机层面、应用层面和云端（如适用）进行日志记录。
- 整合现有的网络安全工具，进行综合扫描、监控、警报和分析。
- 设置本地日志存储，以防止在通信失败的情况下发生日志丢失。

日志记录

系统管理员在 Kubernetes 中运行应用时应该为其环境建立一个有效的日志记录、监控和警报系统。仅仅记录 Kubernetes 事件还不足以了解系统上发生的行动的全貌，还应该在主机层面、应用层面和云端（如适用）进行日志记录。而且，可以将这些日志与任何外部认证和系统日志关联起来，以提供整个环境所采取的行动的完整视图，供安全审计人员和事件响应人员使用。

在 Kubernetes 环境中，管理员应监控 / 记录以下内容：

- API 请求历史
- 性能指标
- 部署情况
- 资源消耗情况
- 操作系统调用情况
- 协议、权限变更情况
- 网络流量
- Pod 扩容情况

Translated by 青藤云安全

当创建或更新一个 Pod 时，管理员应该捕获网络通信、响应时间、请求、资源消耗和任何其他相关指标的详细日志以建立一个基线。如上一节所详述的，应禁用匿名账户，但日志策略仍应记录匿名账户采取的行动，以确定是否有异常活动。

应定期审计 RBAC 策略配置，并在组织机构的系统管理员发生变更时进行审计。这样做可以确保访问控制的调整符合基于角色的访问控制部分中概述的 RBAC 策略加固指南。

审计应包括将当前日志与正常活动的基线测量进行比较，确定任何日志指标和事件是否发生了重大变化。系统管理员应对重大变动情况进行调查——例如，应用使用的变化或恶意程序（如挖矿程序）的安装，以确定其根本原因。应该对内部和外部流量日志进行审计，以确保已正确配置了对所有预期连接的安全限制，并运行状况良好。

管理员还可以在系统发展过程中使用这些审计，以确定何时不再需要外部访问并对外部访问加以限制。

日志可以导入外部日志服务，确保集群外的安全专业人员可以使用日志，尽可能近实时地识别异常情况，并在发生入侵时保护日志不被删除。

默认情况下，Kubernetes 审计功能是禁用的，如果不制定审计策略，就不会进行日志记录。

如果使用这种方法，日志应该在传输过程中用 TLS 1.2 或 1.3 进行加密，以确保网络攻击者无法在传输过程中访问日志并获得关于环境的宝贵信息。在利用外部日志服务器时，要采取的另一项预防措施是在 Kubernetes 内配置日志转发器，只对外部存储进行追加访问。这有助于保护外部存储的日志不被删除或被集群内的其他日志覆盖。

Translated by青藤云安全

Kubernetes 原生审计日志配置

kube-apiserver保留在 Kubernetes 控制平面上，作为前端处理集群的内外部请求。每个请求，无论是由用户、应用还是控制平面产生的，在其执行的每个阶段都会产生一个审计事件。当审计事件注册时，kube-apiserver 检查审计策略文件和适用规则。如果存在这样的规则，服务器会按照第一个匹配的规则所定义的级别记录该事件。Kubernetes 的内置审计功能默认是不启用的，所以如果没有制定审计策略，就不会进行任何日志记录。

集群管理员必须制定一个审计策略 YAML 文件，建立规则，并指定所需的审计级别，以记录每种类型的审计事件。然后，将这个审计策略文件加上适当的标记，传递给 kube-apiserver。一个规则要被认为是有效的，必须将其指定为以下四个审计级别中的一个：none、Meatadataa、Request 或 RequestResponse。**《附录 L：审计策略》**展示了一个审计策略文件的内容，该文件记录了 RequestResponse 级别的所有事件。**《附录 M：向 kube-apiserver 提交审计策略文件的标记示例》**展示了 kube-apiserver 配置文件的位置，并提供了将审计策略文件传递给 kube-apiserver 时的标记示例。附录 M 还提供了关于如何挂载卷以及在必要时配置主机路径的指南。

kube-apiserver 包括用于审计日志的可配置日志和 webhook 后端。日志后端将指定的审计事件写入日志文件，webhook 后端可以配置为将文件发送到外部 HTTP API。附录 M 中的示例中设置的 --audit-log-path 和 --audit-log-maxage 标记是可以用来配置日志后端的两个示例，可以据此将审计事件写到一个文件中。log-path 标记是启用日志记录的最小配置，也是日志后端唯一需要的配置。这些日志文件的默认格式是 JSON，必要时也可以改变。日志后端的其他配置选项可以在 Kubernetes 文档中找到。

Translated by青藤云安全

为了将审计日志推送给组织机构的 SIEM 平台，可以通过提交给 kube-apiserver 的 YAML 文件手动配置 webhook 后端。webhook 配置文件以及如何将该文件传递给 kube-apiserver 可以在**《附录 N：webhook 配置示例》**中查看。可以在 Kubernetes 文档中查看在 kube-apiserver 中设置 webhook 后端配置选项的详尽列表。

工作节点和容器的日志记录

在 Kubernetes 架构中，有很多方法可以配置日志功能。在日志管理的内置方法中，每个节点上的 kubelet 负责日志管理。它根据其对单个文件长度、存储时间和存储容量的策略，在本地存储和轮转日志文件。这些日志是由 kubelet 控制的，可以通过命令行访问。下面的命令打印了一个 Pod 中的容器日志。

```
kubectl logs [-f] [-p] POD [-c CONTAINER]
```

如果要对日志进行流处理，可以使用 -f 标记；如果存在并需要容器先前实例的日志，可以使用 -p 标记；如果 Pod 中有多个容器，可以使用 -c 标记来指定一个容器。如果发生错误导致容器、Pod 或节点死亡，Kubernetes 中的本地日志解决方案并没有提供一种方法来保存存储在失效对象中的日志。NSA 和 CISA 建议配置一个远程日志解决方案，以便在节点失效时保存日志。

远程日志选项	使用的理由	配置实现
在每个节点上运行一个日志代理，将日志推送到后端。	赋予节点展示日志或将日志推送到后端的能力，在发生故障的情况下将其保存在节点外部。	在 Pod 中配置一个独立容器作为日志代理运行，让它访问节点的应用日志文件，并将其配置为将日志转发到组织机构的 SIEM中。
在每个 Pod 中使用一个 sidecar 容器，将日志推送到一个输出流中	将日志推送到独立的输出流。当应用容器写入不同格式的多个日志文件时，这可能是一个有用的选项。	为每种日志类型配置 sidecar 容器，并将这些日志文件重定向到它们各自的输出流，然后由 kubelet 进行处理。然后，节点级的日志代理可以将这些日志转发给 SIEM 或其他后端。

Translated by 青藤云安全

在每个 Pod 中使用一个日志代理 sidecar，将日志推送到后端	当需要比节点级日志所能提供的更多灵活性时，可以通过Agent来实现。	对每个 Pod 进行配置，将日志直接推送到后端。这是连接第三方日志代理和后端的常用方法。
从应用中直接向后端推送日志	捕获应用的日志。Kubernetes 没有内置的机制直接将日志展示给或推送到后端。	各组织将需要在其应用中建立这一功能，或附加一个信誉良好的第三方工具来实现这一功能。

Sidecar 容器与其他容器一起在 Pod 中运行，可以被配置为将日志流向日志文件或日志后端。Sidecar 容器也可以配置为作为另一个标准功能容器的流量代理进行打包和部署。

为了确保这些日志代理在工作节点之间的连续性，通常将它们作为 DaemonSet 运行。为这种方法配置 DaemonSet，可以确保每个节点上都有一份日志代理的副本，而且对日志代理所做的任何改变在集群中都是一致的。

Seccomp：审计模式

除了上述的节点和容器日志外，记录系统调用也是非常有用的。在 Kubernetes 中审计容器系统调用的一种方法是使用安全计算模式（seccomp）工具。这个工具默认是禁用的，但可以用来限制容器的系统调用能力，从而降低内核的攻击面。Seccomp 还可以通过使用审计配置文件记录正在进行的调用情况。

自定义 seccomp 配置文件可以用于定义允许哪些系统调用，以及对未指定调用该执行哪些默认操作。为了在 Pod 中启用自定义 seccomp 配置文件，Kubernetes 管理员可以将他们的 seccomp 配置文件 JSON 文件写入到 /var/lib/kubelet/seccomp/ 目录，并将 seccompProfile 添加到 Pod 的 securityContext。自定义的 seccompProfile 还应该包括两个字段——Type: Localhost 和 localhostProfile: myseccomppolicy.json。记录所有的系统调用可以帮助管理员了解标准操作需要哪些系统调用，有助于他们进一步限制 seccomp 配置文件而不会失去系统功能。

Translated by 青藤云安全

SYSLOG

Kubernetes 默认将 kubelet 日志和容器运行时日志写入 journald（若该服务可用）。如果组织机构希望让默认情况下不使用syslog的系统使用 syslog 工具，或者从整个集群收集日志并将其转发到 syslog 服务器或其他日志存储和聚合平台，他们可以手动配置该功能。Syslog 协议定义了一个日志信息格式化标准。Syslog 消息包括一个头部——由时间戳、主机名、应用名称和进程 ID（PID）组成，以及一个以明文书写的消息。Syslog 服务，如 syslog-ng® 和 rsyslog，能够以统一的格式收集和汇总整个系统的日志。许多 Linux 操作系统默认使用 rsyslog 或 journald。 journald是一个事件日志守护程序，它优化了日志存储并通过 journalctl 输出 syslog 格式的日志。在运行某些 Linux 发行版的节点上，syslog 工具默认记录其操作系统层面的事件。运行这些 Linux 发行版的容器，默认也会使用 syslog 收集日志。由 syslog 工具收集的日志存储在每个适用节点或容器的本地文件系统中，除非配置了一个日志聚合平台来收集日志。

SIEM 平台

安全信息和事件管理（SIEM）软件从整个组织机构的网络中收集日志。SIEM 软件将防火墙日志、应用程序日志等汇集在一起；将它们解析出来，提供了一个集中的平台，让分析人员可以从这个平台上监控系统安全。SIEM 工具在功能上有差异。一般来说，这些平台提供日志收集、威胁检测和警报功能。有些包括机器学习功能，可以更好地预测系统行为并帮助减少错误警报。在其环境中使用这些平台的组织机构可以将SIEM与 Kubernetes 集成，以更好地监测和保护集群。用于管理 Kubernetes 环境日志的开源平台可以作为 SIEM 平台的替代方案。

容器化环境在节点、Pod、容器和服务之间有许多相互依赖的关系。在这些环境中，Pod 和容器不断地在不同的节点上关闭和重启。这给传统的 SIEM 带来了额外挑战，因为它们通常使用 IP 地址来关联日志。即使是下一代的 SIEM 平台也不一定适合复杂的 Kubernetes 环境。

Translated by 青藤云安全

然而，随着 Kubernetes 成为最广泛使用的容器编排平台，许多开发 SIEM 工具的组织机构已经开发了专门用于 Kubernetes 环境的变种产品，为这些容器化环境提供全面的监控解决方案。管理员应该了解他们平台的能力，并确保他们的日志充分捕捉到环境中的状况，为未来的事件响应提供有力支撑。

警报

Kubernetes 本身并不支持警报功能；然而，一些具有警报功能的监控工具与 Kubernetes 兼容。如果 Kubernetes 管理员选择配置一个警报工具在 Kubernetes 环境中工作，有几个指标是管理员应该监控和配置警报的。

可能触发警报的示例包括但不限于：

- 环境中的任何机器上磁盘空间过低
- 记录卷上可用存储空间过少
- 外部日志服务脱机
- 一个Pod 或应用以 root 权限运行
- 一个账户对他们没有权限的资源提出请求
- 一个匿名账户正在使用或获得特权
- Pod 或工作节点的 IP 地址被列为 Pod 创建请求的源 ID
- 系统调用异常或 API 调用失败
- 用户 / 管理员的行为不正常（包括在不寻常的时间或从不寻常的地点），以及
- 明显偏离标准操作指标基线

当存储不足时发出警报，可以帮助避免因资源有限而导致性能问题以及日志丢失，并有助于识别恶意的加密劫持行为。可以调查有特权的 Pod 执行案例，以确定实管理员犯了错误，还是一个真实的用例需要权限升级，或者是恶意行为者部署了一个有特权的 Pod。可疑的 Pod 创建源 IP 地址可能表明，恶意网络攻击者已经攻陷了容器并试图创建一个恶意 Pod。

Translated by青藤云安全

将 Kubernetes 与企业现有的 SIEM 平台整合，特别是那些具有机器学习 / 大数据功能的平台，可以帮助识别审计日志中的违规行为并减少错误警报。如果配置这样的工具与 Kubernetes 一起工作，它应该被配置为这些情况和任何其他适用于用例的情况被配置为触发警报。

当疑似入侵发生时，能够自动采取行动的系统有可能被配置为在管理员对警报作出反应时采取步骤以减轻损害。在 Pod IP 被列为 Pod 创建请求的源 ID 的情况下，一个可以实施的缓解措施是自动驱逐 Pod，以保持应用程序的可用性，但暂时停止对集群的任何损害。这样做将允许一个干净的 Pod 版本被重新安排到一个节点上。然后，调查人员可以检查日志，以确定是否发生了漏洞，如果是的话，调查恶意行为者是如何执行潜在威胁的，以便可以部署一个补丁。

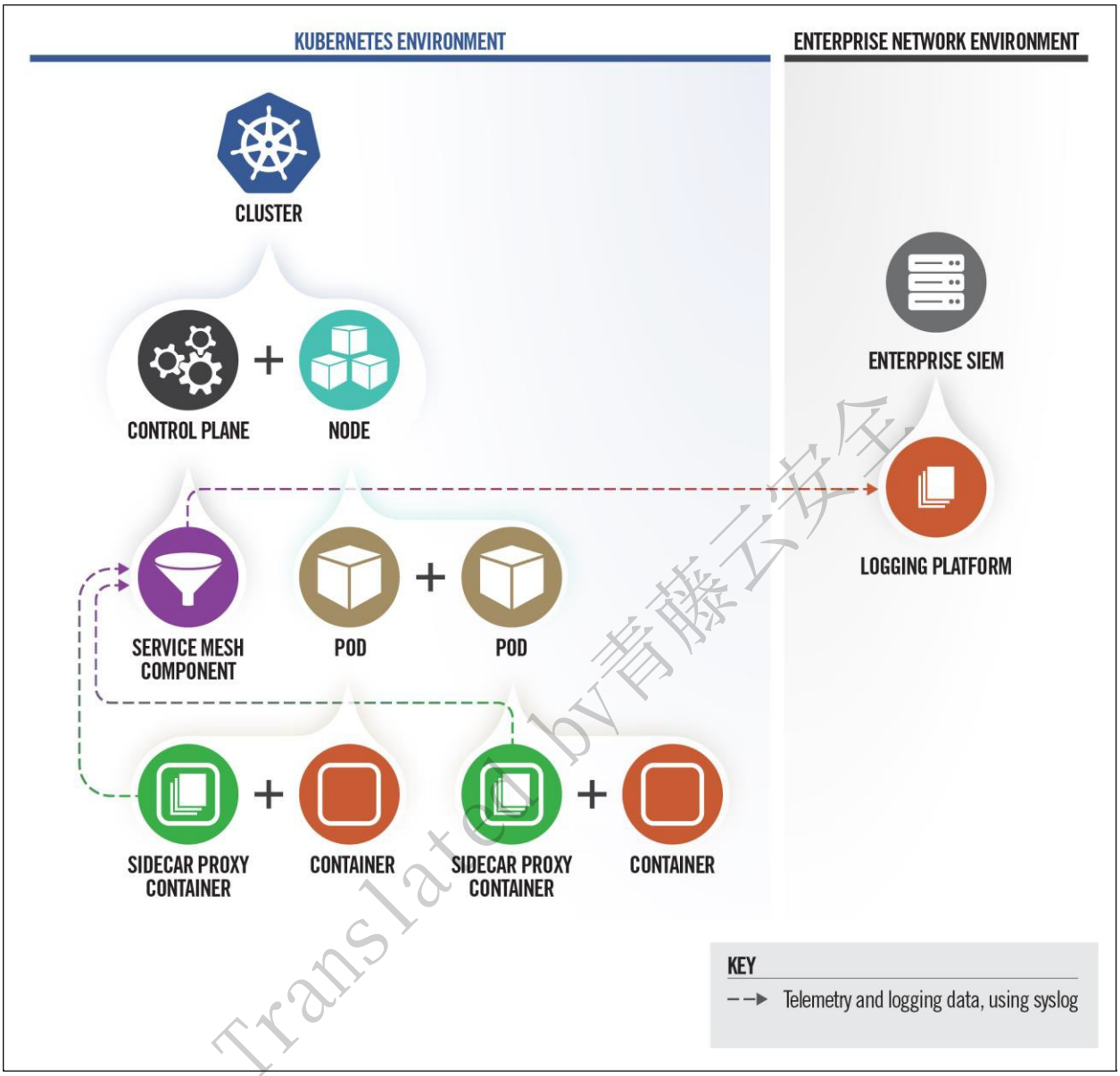
服务网格

服务网格是一个平台，由于可以将通信逻辑编码到服务网格中，而不是每个微服务中，因而简化了应用中的微服务通信。将这种通信逻辑编码到各个微服务中是很难扩展的，发生故障时也很难调试，而且很难保证安全。使用服务网格则可以简化开发人员的工作。服务网格可以：

- 在一个服务中断时，重新定向流量。
- 收集性能指标优化通信。
- 管理服务与服务之间的通信加密。
- 收集服务间的通信日志。
- 收集每个服务的日志。
- 帮助开发者诊断微服务或通信机制的问题和故障。

服务网格还有助于将服务迁移到混合或多云环境中。虽然服务网格不是必须的，但这是一种高度适合 Kubernetes 环境的方案。托管的 Kubernetes 服务通常包括他们自己的服务网格。然而，也有其他几个平台也是可用的，如果需要，是可以高度定制的。其中一些平台包括一个生成和轮换证书的证书颁发机构，可以实现服务之间进行安全的 TLS 认证。管理员应该考虑使用服务网格来加强 Kubernetes 集群的安全性。

Translated by 腾云安全



Translated by青藤云安全

图5：集群通过服务网格将日志与网络安全结合起来

容错性

组织机构应该制定容错策略，以确保日志服务的可用性。这些策略可以根据具体的 Kubernetes 用例而有所不同。一个可以实施的策略是，如果在存储容量超标的情况下，绝对有必要允许新的日志覆盖最旧的日志文件。

如果日志被发送到外部服务，应该建立一种机制，以便在发生通信中断或外部服务故障时将日志存储在本地。一旦与外部服务的通信恢复，应按照策略，将本地存储的日志推送到外部服务器上。

工具

Kubernetes 不包括广泛的审计功能。然而，该系统是可扩展的，允许用户自由开发自己的定制解决方案，或选择适合自己需求的现有附加组件。最常见的解决方案之一是添加其他审计后端服务，它可以使用 Kubernetes 记录的信息，并为用户执行其他功能，如扩展搜索参数、数据映射功能和警报功能。已经使用 SIEM 平台的企业可以将 Kubernetes 与这些现有的功能进行整合。

开源监控工具，如 Cloud Native Computing Foundation 的 Prometheus®、Grafana Labs 的 Grafana® 和 Elasticsearch 的 Elastic Stack (ELK)®，都可以用于进行事件监控、运行威胁分析、管理警报，以及收集资源隔离参数、历史使用情况和运行容器的网络统计数据。在审计访问控制和权限配置时，扫描工具可以通过协助识别 RBAC 中的风险权限配置而发挥作用。NSA 和 CISA 鼓励组织机构，若在现有环境中使用了入侵检测系统（IDS），可以考虑将该服务也整合到他们的 Kubernetes 环境中。整合后，企业能够监测并杀死有异常行为迹象的容器，从而让容器能够从最初的干净镜像中重新启动。许多云服务提供商也为那些希望得到更多管理和可扩展解决方案的人提供容器监控服务。

Translated by 信息安全

升级和应用安全实践

按照本文件中所述的加固指南行事是确保容器通过Kubernetes编排时，容器中应用的安全的一个重要步骤。然而，安全是一个持续的过程，及时打补丁、更新和升级也是至关重要的。具体的软件组件因个人配置的不同而不同，但整个系统的每个组件都应尽可能保持安全。这包括更新：Kubernetes、管理程序、虚拟化软件、插件、环境运行的操作系统、服务器上运行的应用，以及 Kubernetes 环境中托管的任何其他软件。

互联网安全中心（CIS）发布了保护软件安全的基准。管理员应遵守 Kubernetes 和任何其他相关系统组件的 CIS 基准。管理员应定期检查，以确保其系统的安全性符合当前安全专家对最佳实践的共识。应定期对各种系统组件进行漏洞扫描和渗透测试，主动寻找不安全的配置和零日漏洞。任何问题都应在被潜在的网络攻击者发现和利用之前及时补救。

软件更新后，管理员也应该从环境中删除任何不再需要的旧组件。使用Kubernetes 托管服务可有助于自动升级和修补 Kubernetes、操作系统和网络协议。然而，管理员仍然必须为他们的容器化应用打补丁和升级。

Translated by青藤云安全

参考文件

[1] Center for Internet Security, "Kubernetes," 2021. [Online]. Available: https://cisecurity.org/resources/?type=benchmark&search=kubernetes.

[2] DISA, "Kubernetes STIG," 2021. [Online]. Available: https://dl.dod.cyber.mil.wp-content/uploads/stigs/zip/U_Kubernetes_V1R1_STIG.zip. [Accessed 8 July 2021]

[3] The Linux Foundation, "Kubernetes Documentation," 2021. [Online]. Available: https://kubernetes.io/docs/home/. [Accessed 8 July 2021].

[4] The Linux Foundation, "11 Ways (Not) to Get Hacked," 18 07 2018. [Online]. Available: https://kubernetes.io/blog/2018/07/18/11-ways-not-to-get-hacked/#10-scan-images-and-run-ids. [Accessed 8 July 2021].

[5] MITRE, "Unsecured Credentials: Cloud Instance Metadata API." MITRE ATT&CK, 2021. [Online]. Available: https://attack.mitre.org/techniques/T1552/005/. [Accessed 8 July 2021].

[6] CISA, "Analysis Report (AR21-013A): Strengthening Security Configurations to Defend Against Attackers Targeting Cloud Services." 网络安全与基础设施安全署, 14 January 2021. [Online]. Available:https://us-cert.cisa.gov/ncas/analysis-reports/ar21-013a [Accessed 8 July 2021].

Translated by 青藤云安全

附录 A：非 root 应用的 Dockerfile 示例

下面的示例是一个 Dockerfile，它以非 root 用户和非 group 成员身份运行一个应用程序。下面标红的行是使用非root权限所特有的。

```
FROM ubuntu:latest

#Update and install the make utility
RUN apt update && apt install -y make

#Copy the source from a folder called "code" and build the application with
the make utility
COPY . /code
RUN make /code

#Create a new user (user1) and new group (group1); then switch into that
user's context
RUN useradd user1 && groupadd group1
USER user1:group1

#Set the default entrypoint for the container
CMD /code/app
```

Translated by青藤云安全

附录 B：只读文件系统的部署模板示例

下面是一个使用只读根文件系统的 Kubernetes 部署模板示例。下文标红色的行是确保容器文件系统为只读的命令，标蓝色的行是如何为需要这种功能的应用创建一个可写卷。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: web
  name: web
spec:
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
      name: web
    spec:
      containers:
      - command: ["sleep"]
        args: ["999"]
        image: ubuntu:latest
        name: web
        securityContext:
          readOnlyRootFilesystem: true
        volumeMounts:
          - mountPath: /writeable/location/here
            name: volName
      volumes:
      - emptyDir: {}
        name: volName
```

Translated by青藤云安全

附录 C：Pod 安全策略示例

下面是一个 Kubernetes Pod 安全策略示例，它为集群中运行的容器强制执行了强增强式的安全要求。下面是一个基于官方的 Kubernetes 文档的示例。我们鼓励管理员对该策略进行修改，以满足他们组织机构的特定要求。

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames:
'docker/default,runtime/default'
    apparmor.security.beta.kubernetes.io/allowedProfileNames:
'runtime/default'
    seccomp.security.alpha.kubernetes.io/defaultProfileName:
'runtime/default'
    apparmor.security.beta.kubernetes.io/defaultProfileName:
'runtime/default'
spec:
  privileged: false # Required to prevent escalations to root.
    allowPrivilegeEscalation: false
  requiredDropCapabilities:
    - ALL
  volumes:
    - 'configMap'
    - 'emptyDir'
    - 'projected'
    - 'secret'
    - 'downwardAPI'
    - 'persistentVolumeClaim' # Assume persistentVolumes set up by admin
are safe
  hostNetwork: false
  hostIPC: false
  hostPID: false
  runAsUser:
    rule: 'MustRunAsNonRoot' # Require the container to run without root
  seLinux:
    rule: 'RunAsAny' # This assumes nodes are using AppArmor rather than
SELinux
  supplementalGroups:
    rule: 'MustRunAs'
    ranges: # Forbid adding the root group.
      - min: 1
        max: 65535
  runAsGroup:
    rule: 'MustRunAs'
    ranges: # Forbid adding the root group.
      - min: 1
        max: 65535
  fsGroup:


    rule: 'MustRunAs'
    ranges: # Forbid adding the root group.
      - min: 1
        max: 65535
readOnlyRootFilesystem: true
```

Translated by青藤云安全

附录 D：命名空间示例

在下面的示例中，我们为每个团队或用户组，使用 kubectl 命令或 YAML 文件创建一个 Kubernetes 命名空间。应避免使用任何带有 kube 前缀的名称，因为这可能与 Kubernetes 系统保留的命名空间相冲突。

用Kubectl 命令来创建一个命名空间。

```
kubectl create namespace <insert-namespace-name-here>
```

要使用 YAML 文件创建命名空间，要创建一个名为 my-namespace.yaml 的新文件，内容如下：

```
apiVersion: v1
kind: Namespace
metadata:
  name: <insert-namespace-name-here>
```

要应用命名空间：

```
kubectl create -f ./my-namespace.yaml
```

要在现有的命名空间创建新的 Pod，需要切换到所需的命名空间：

```
kubectl config use-context <insert-namespace-here>
```

应用新的部署：

```
kubectl apply -f deployment.yaml
```

或者，也可以用以下方法将命名空间添加到 kubectl 命令中：

```
kubectl apply -f deployment.yaml --namespace=<insert-namespace-here>
```

或者在 YAML 声明中的元数据中指定 namespace：<insert-namespace-here>。

创建后，不能在命名空间之间移动资源。必须删除该资源，才能在新的命名空间中重新创建。

Translated by青藤云安全

附录 E：网络策略示例

网络策略根据使用的网络插件而定。下面是一个网络策略的示例，参考了 Kubernetes 文档，限制只有带标签的Pod方能访问 nginx 服务。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
 name: example-access-nginx
 namespace: prod #this can any namespace or be left out if no
namespace is used
spec:
    podSelector:
        matchLabels:
            app: nginx
    ingress:
    -from:
        -podSelector:
            matchLabels:
                access: "true"
```

应用新的 NetworkPolicy：

```
kubectl apply -f policy.yaml
```

一个默认的拒绝所有入口的策略：

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
    name: deny-all-ingress
spec:
    podSelector: {}
    policyType:
    - Ingress
```

一个默认拒绝所有出口的策略：

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
    name: deny-all-egress
spec:
    podSelector: {}
    policyType:
    - Egress
```

Translated by青藤云安全

附录 F：LimitRange 示例

在 Kubernetes 1.10 及以上版本中，默认启用LimitRange。下面的 YAML 文件为每个容器指定了一个 LimitRange，其中包含一个默认的请求和限制，以及最小和最大的请求。

```
apiVersion: v1
kind: LimitRange
metadata:
   name: cpu-min-max-demo-lr
spec:
   limits
   -  default:
         cpu: 1
      defaultRequest:
         cpu: 0.5
      max:
         cpu: 2
      min:
         cpu 0.5
      type: Container
```

将LimitRange应用于命名空间：

```
kubectl apply -f <example-LimitRange>.yaml --namespace=<Enter-Namespace>
```

在应用了这个 LimitRange 示例配置后，如果没有特殊指定，命名空间中创建的所有容器都会被分配到默认的 CPU 请求和限制。命名空间中的所有容器的 CPU 请求必须大于或等于最小值，小于或等于最大 CPU 值，否则无法实现容器实例。

Translated by青藤云安全

附录 G：ResourceQuota 示例

通过将 YAML 文件应用于命名空间或在 Pod 的配置文件中指定要求来创建 ResourceQuota 对象，以限制命名空间内的整体资源使用情况。下面是一个基于 Kubernetes 官方文档的示例：

一个命名空间的配置文件示例：

```
apiVersion: v1
kind: ResourceQuota
metadata:
    name: example-cpu-mem-resourcequota
spec:
    hard:
        requests.cpu: "1"
        requests.memory: 1Gi
        limits.cpu: "2"
        limits.memory: 2Gi
```

应用 ResourceQuota：

```
kubectl apply -f example-cpu-mem-resourcequota.yaml --
namespace=<insert-namespace-here>
```

这个 ResourceQuota 对所选择的命名空间做出了以下限制：

- 每个容器都必须有一个内存请求、内存限制、CPU 请求和 CPU 限制。
- 所有容器的总内存请求不应超过 1 GiB
- 所有容器的总内存限制不应超过 2 GiB
- 所有容器的 CPU 请求总量不应超过 1 个 CPU
- 所有容器的总 CPU 限制不应超过 2 个 CPU

Translated by青藤云安全

附录 H：加密示例

要对Secret静态数据进行加密，下面的加密配置文件提供了一个示例，示例指定了所需的加密类型和加密密钥。将加密密钥存储在加密文件中只能略微提高安全性。将Secret加密，但可以在 EncryptionConfiguration 文件中访问。下面是基于Kubernetes 官方文档的一个示例。

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
   -  resources:
      -  secrets
      providers:
      -  aescbc:
            keys:
            -  name: key1
               secret: <base 64 encoded secret>
      -  identity: {}
```

要使用该加密文件进行静态加密，要在重启 API 服务器时设置 --encryption-provider-config 标记，并注明配置文件的位置。

Translated by 青藤云安全

附录 I：KMS 配置示例

要用密钥管理服务（KMS）提供商插件来加密 Secret，可以使用以下加密配置 YAML 文件示例来设置提供商的相关参数。下面是基于 Kubernetes 官方文档的一个示例。

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
      - secrets
    providers:
      - kms:
          name: myKMSPlugin
          endpoint: unix://tmp/socketfile.sock
          cachesize: 100
          timeout: 3s
      - identity: {}
```

使用 KMS 提供商时要配置 API 服务器，需要设置 --encryption-provider-config 标记与配置文件的位置，并重新启动 API 服务器。

要从本地加密提供者切换到 KMS，请将 EncryptionConfiguration 文件中的 KMS 提供者部分添加到当前加密方法之上，如下所示。

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
      - secrets
    providers:
      - kms:
          name: myKMSPlugin
          endpoint: unix://tmp/socketfile.sock
          cachesize: 100
          timeout: 3s
      - aescbc:
          keys:
            - name: key1
              secret: <base64 encoded secret>
```

重新启动 API 服务器并运行下面的命令来重新加密所有与 KMS 供应商的 Secret。

```
kubectl get secrets --all-namespaces -o json | kubectl replace -f -
```

Translated by 青藤云安全

附录 J：pod-reader RBAC 角色示例

要创建一个 pod-reader 角色，需要创建一个 YAML 文件，内容如下：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
 namespace: your-namespace-name
 name: pod-reader
rules:
-  apiGroups: [“”]   # “” indicates the core API group
   resources: [“pods”]
   verbs: [“get”, “watch”, “list”]
```

应用角色：

```
kubectl apply --f role.yaml
```

创建一个 global-pod-reader ClusterRole：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata: default
 # “namespace” omitted since ClusterRoles are not bound to a
namespace
 name: global-pod-reader
rules:
-  apiGroups: [“”] # “” indicates the core API group
   resources: [“pods”]
   verbs: [“get”, “watch”, “list”]
```

应用角色：

```
kubectl apply --f clusterrole.yaml
```

Translated by青藤云安全

附录K：RBAC RoleBinding和ClusterRoleBinding示例

要创建一个 RoleBinding，需要创建一个 YAML 文件，内容如下：

```
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "jane" to read Pods in the "your-
namespace-name"
# namespace.
# You need to already have a Role names "pod-reader" in that
namespace.
kind: RoleBinding
metadata:
    name: read-pods
    namespace: your-namespace-name
subjects:
# You can specify more than one "subject"
-  kind: User
   name: jane # "name" is case sensitive
   apiGroup: rbac.authorization.k8s.io
roleRef:
   # "roleRef" specifies the binding to a Role/ClusterRole
   # kind: Role # this must be a Role or ClusterRole
   # this must match the name of the Role or ClusterRole you wish to
  bind
   # to
   name: pod-reader
   apiGroup: rbac.authorization.k8s.io
```

应用RoleBinding：

```
kubectl apply --f rolebinding.yaml
```

要创建一个ClusterRoleBinding，需要创建一个 YAML 文件，内容如下：

```
apiVersion: rbac.authorization.k8s.io/v1
# This cluster role binging allows anyone in the "manager" group to
read
# Pod information in any namespace.
kind: ClusterRoleBinding
metadata:
    name: global-pod-reader
subjects:
# You can specify more than one "subject"
-  kind: Group
   name: manager # Name is case sensitive
   apiGroup: rbac.authorization.k8s.io
roleRef:
   # "roleRef" specifies the binding to a Role/ClusterRole
   kind: ClusterRole # this must be a Role or ClusterRole
   name: global-pod-reader # this must match the name of the Role or
ClusterRole you wish to bind to
```

Translated by 清藤云安全

```
  apiGroup: rbac.authorization.k8s.io
```

应用RoleBinding：

```
kubectl apply --f clusterrolebinding.yaml
```

Translated by青藤云安全

附录 L：审计策略

下面是一个按最高级别记录所有审计事件的审计策略：

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
   -  level: RequestResponse
# This audit policy logs all audit events at the RequestResponse
level
```

这种审计策略按照最高级别记录所有事件。如果一个组织机构有可用的资源来存储、解析和检查大量的日志，那么按最高级别记录所有事件是个良好管理，可以确保事件发生时，能在日志中找到所有必要的背景信息。如果资源消耗和可用性是一个问题，那么可以建立更多的日志规则来降低非关键组件和常规非特权操作的日志级别，只要满足系统的审计要求即可。可以在 Kubernetes 官方文档中找到如何建立这些规则的示例。

Translated by青藤云安全

附录 M：向 kube-apiserver 提交审计策略文件的标记示例

在控制平面中，用文本编辑器打开 kube-apiserver.yaml 文件。编辑 kube-apiserver 配置需要管理员权限。

```
sudo vi /etc/kubernetes/manifests/kube-apiserver.yaml
```

在 kube-apiserver.yaml 文件中添加以下文字：

```
--audit-policy-file=/etc/kubernetes/policy/audit-policy.yaml
--audit-log-path=/var/log/audit.log
--audit-log-maxage=1825
```

audit-policy-file 标记应该设置为审计策略的路径，而 audit-log-path 标记应该设置为审计日志预期写入的安全位置。还有一些其他的标记，比如这里显示的 audit-log-maxage 标记，它规定了日志保存的最大天数，还有一些标记用于指定要保留的最大审计日志文件的数量、最大的日志文件大小（兆字节）等等。启用日志记录的唯一必要标志是 audit-policy-file 和 audit-log-path 标记。其他标记可以用来配置日志，以符合组织机构的策略。

如果用户的 kube-apiserver 是作为 Pod 运行的，那么就有必要挂载卷，并配置策略和日志文件位置的 hostPath 以保留审计记录。这可以通过在 Kubernetes 官方文档中指出的 kube-apiserver.yaml 文件中添加以下内容来实现：

```
volumeMounts:
 -mountPath: /etc/kubernetes/audit-policy.yaml
  name: audit
  readOnly: true
 -mountPath: /var/log/audit.log
  name: audit-log
  readOnly: false

volumes:
- hostPath:
    path: /etc/kubernetes/audit-policy.yaml
    type: File
 name: audit

- hostPath:
   path: /var/log/audit.log
   type: FileOrCreate
name: audit-log
```

Translated by 腾云安全

附录 N：Webhook 配置

YAML 文件示例：

```
apiVersion: v1
kind: Config
preferences: {}
clusters:
    - name: example-cluster
      cluster:
        server: http://127.0.0.1:8080
        #web endpoint address for the log files to be sent to
       name: audit-webhook-service
    users:
    - name: example-users
      user:
        username: example-user
        password: example-password
    contexts:
    - name: example-context
      context:
        cluster: example-cluster
        user: example-user
    current-context: example-context
#source: https://dev.bitolog.com/implement-audits-webhook/
```

由 webhook 发送的审计事件是以 HTTP POST 请求的形式发送的，请求体中包含 JSON 审计事件。指定的地址应该指向一个能够接受和解析这些审计事件的端点，无论是第三方服务还是内部配置的端点都可以。

提交到kube-apiserver中的配置文件的标记示例：

在控制面板上编辑 kube-apiserver.yaml 文件

```
sudo vi /etc/kubernetes/manifests/kube-apiserver.yaml
```

将下列文本添加到kube-apiserver.yaml文件中

```
--audit-webhook-config-file=/etc/kubernetes/policies/webhook-
 policy.yaml
--audit-webhook-initial-backoff=5
--audit-webhook-mode=batch
--audit-webhook-batch-buffer-size=5
```

audit-webhook-initial-backoff 标记决定了在初次请求失败后重新请求时要等待多长时间。可用的 webhook 模式有batch、blocking和blocking-strict三种模式。使用batch模式时，可以配置最长等待时间、缓冲区大小等。Kubernetes 官方文档包含有关其他配置选项的更多详细信息。

Translated by青藤云安全