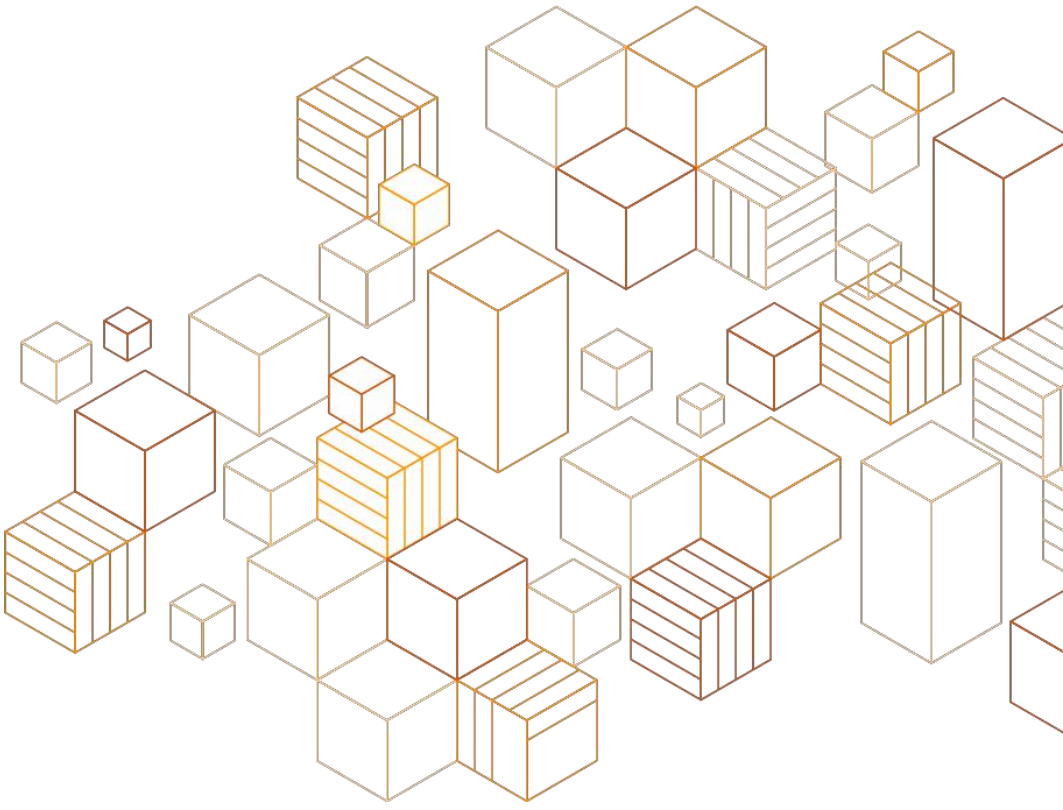
利用 Amazon EKS 打造多租户 SaaS 应用程序的安全实践

技术指南

2021 年 6 月 4 日

亚马逊云科技



公告

客户负责对本文档中的信息进行独立评估。本文档：(a) 仅供参考，(b) 代表亚马逊云科技当前的产品服务和实践，如有变更，恕不另行通知，以及 (c) 不构成亚马逊云科技及其附属公司、供应商或授权商的任何承诺或保证。亚马逊云科技产品或服务均“按原样”提供，没有任何明示或暗示的担保、声明或条件。亚马逊云科技对其客户的责任和义务由亚马逊云科技协议决定，本文档与亚马逊云科技和客户之间签订的任何协议无关，亦不影响任何此类协议。

目录

关于本指南

本技术指南说明了如何安全管理和运行 Amazon Elastic Kubernetes Service（Amazon EKS）集群上的多租户软件即服务（SaaS）应用程序。

本文档改编自 Amazon EKS 最佳实践指南。最佳实践指南将会不时进行更新。Amazon EKS 和 Kubernetes 也会持续进行升级，因此亚马逊云科技建议用户定期检查更新情况。亚马逊云科技还建议用户订阅亚马逊云科技容器博客，接收有关亚马逊云科技容器服务方面的最新信息。

概述

Amazon EKS 主要面向利用亚马逊云科技构建软件即服务（SaaS）解决方案的客户。在这些 SaaS 解决方案中，租户数据和应用程序的隔离方式可能存在一定差异。一些 SaaS 提供商依赖于独占租户模式。在该模式下，每个租户独立使用各自的资源。也有一些提供商会采用*池化*租户模式。在该模式下，各租户共享资源。

下文详细说明了在 Amazon EKS 上实现这两种模式的方法：

- 在**池模式**环境下，租户共享 EKS 资源，同时还会实施一些附加措施，确保租户无法在未经授权的情况下访问其他租户的资源。许多客户希望能够使用共享主机和公共控制平面来运行工作负载。该方法通常可以简化 SaaS 应用程序的运行流程，同时改进 SaaS 环境的敏捷性、创新性和成本模型。
- 在**独占模式**环境下，各租户享有专有的 EKS 资源。对于可能需要绝对隔离边界的租户，该模式非常适用。租户可能会出于多种原因而使用此模式（安全性、吵闹邻居等）。在 EKS 中，可以使用多种架构实现独占模式。对于从独占模式中访问的资源，可以通过*独占*或*池*模式进行部署。

这些选择并不具有非排他性。一些 SaaS 提供商可能同时支持这两种选项，具体视应用程序中的层级或服务而定。

无论采取哪种模式，都必须确保租户无法进行以下操作：

- 读写与该租户无关的任何控制平面信息
- 访问任何并非该租户所有的资源
- 获取非该租户所有的凭证
- 假冒其他租户
- 规避对该租户的计算、内存或其他资源分配方面的限制

建议

亚马逊云科技建议重点关注以下目标：

- 严格隔离租户间的控制平面数据
- 防止因租户容器导致的主机崩溃
- 防止租户容器“越狱”或访问主机上的敏感数据（例如凭证）

使用多集群隔离租户工作负载

在 EKS 上运行独占工作负载的最安全方式是，为每个租户创建独有的 EKS 集群。采用这种设计时，即使租户运行特权容器且具有主机访问权限，也不会影响其他租户。另外还须特别注意，不得公开不同集群上其他租户所关联的凭证，同时必须实施额外的亚马逊云科技安全最佳实践，例如适当的安全组规则和/或虚拟私有云（VPC）隔离策略等。

这种方法也存在一些缺点。为每个租户创建单独的集群，这将增加您环境运行流程的复杂性。尽管使用该方法时可以自动处理大多数运行体验，但仍会影响到您的 SaaS 环境的效率、敏捷性和成本模型。

使用租户专用的工作节点

针对在单个集群上托管多个租户的客户，应该将租户工作负载隔离在专用节点上。如果出现容器越狱，这有助于确保任何其他客户的 Pod 或数据不会遭遇意外访问或篡改。

AWS Fargate

实施这种限制措施的最简单的方法是，在 AWS Fargate 上运行租户 Pod。Fargate 是一种托管式计算服务，可以帮助代理运行 EKS Pod，无需用户自行管理 Amazon Elastic Compute Cloud（Amazon EC2）实例。 配置 Pod 时，可以按需分配和自定义容量，使其匹配 Pod 资源。使用 Fargate 时，不能在同一虚拟机（VM）上同时运行任何两个 Pod，以确保为租户工作负载实施 VM 级别隔离和容器隔离。

Amazon EC2 工作节点

或者，如果使用 EC2 实例，实施这种限制措施的一种方法是，在所有包含租户标识符的节点应用“污点”，如 tenantID=12345:NoSchedule。如果在租户的 Pod 规范中结合使用匹配容差，能确保仅将租户的 Pod 放置在匹配同一污点的节点上。

另一种（稍弱）限制实施方式是，使用租户标识符来标记节点，并在 Pod 规范中使用 nodeSelector 或 affinity 规则，确保仅在正确节点上配置租户 Pod。对于计划使用这种方式实施限制的客户，应使用准入控制器（此内容将于后文讨论），以确保在所有客户 Pod 中提供这些字段。

节点授权模式

作为一种缓解控制措施，在 EKS 集群中，所有节点请求都需要采取节点授权模式。在与节点本身所运行 Pod 无关的情况下，这能防止节点访问 Secrets、ConfigMaps、Persistent Volume Claims 或 Persistent Volumes。 有关更多信息，请参阅使用节点授权。

不提供 Kubernetes 或 EKS API 的直接访问权限

如果接受租户的不受信任输入，并将输入内容传输至 Kubernetes 等安全敏感系统，则可能会导致您的集群或其租户暴露于风险之中，例如未授权修改和数据访问。因此亚马逊云科技建议，在租户与运行工作负载的 EKS 集群之间创建离散管理层。与 Web 应用防火墙（WAF）类似，该层允许首先检查和过滤请求，随后再采取进一步措施。无效请求应立即予以拒绝，而对于有效的请求，则应附加身份信息和安全相关修改，然后再传输至 Kubernetes 控制平面。

使用命名空间隔离租户工作负载

Kubernetes 将命名空间作为一种逻辑分区系统，用于组织管理 Pod 和部署等对象。在 Kubernetes 基于角色的访问控制（RBAC）系统中，命名空间还可以用作特权边界。例如，在命名空间 A 中所创建的 Pod 不能访问命名空间 B 中的密钥（反之亦然）。

亚马逊云科技建议客户为每个租户分配各自唯一的命名空间。向租户分配特权时，请确保每个租户只能访问该租户指定命名空间中的 Kubernetes 对象。客户可启用 Mutating Admission Webhook 工具来自动处理此项任务，该工具要求在所有客户相关对象中提供租户特定标签，从而确保将对象放置在租户的命名空间中。

限制容器特权

租户容器默认以**非特权**模式运行。如果租户容器需要特权，则这些特权应仅限于成功运行容器的必要范围内。

特权模式在容器的 SecurityContext 中进行指定。仅能通过以下两种方式指定特权模式：

- 将 privileged（特权）属性设置为“True”（真）。这与在主机中设置 root 访问权限的方式相同。
- 在 capabilities（功能）列表中，将一个或多个功能的列表指定为 add（添加）或 drop（删除）。

在运行 Docker 容器运行时的 EKS 节点中（包括使用 EKS 优化型 Amazon Machine Image （AMI）的节点），每个容器包含以下默认功能：

```
CAP_CHOWN, CAP_DAC_OVERRIDE, CAP_FOWNER, CAP_FSETID, CAP_KILL,
CAP_SETGID, CAP_SETUID, CAP_SETPCAP, CAP_NET_BIND_SERVICE,
CAP_NET_RAW, CAP_SYS_CHROOT, CAP_MKNOD, CAP_AUDIT_WRITE,
CAP_SETFCAP
```

亚马逊云科技建议从该列表中删除所有不必要的功能，大多数软件都不需要这些功能。亚马逊云科技还建议检查 Linux 功能的完整列表，并允许每个租户仅选择您允许使用的功能。各项功能记录在 Capabilities Linux 手册页面中。

亚马逊云科技还建议，禁止运行任何 privileged（特权）属性设置为 true（真）的容器。通过授权特定功能来提供精细化特权，这种方式更具安全性。例如，对于需要绑定到低编号端口的容器，可以使用 CAP_NET_BIND_SERVICE 功能运行，而不必使用全部特权；亚马逊云科技还建议禁用 CAP_SYS_ADMIN 和 CAP_NET_ADMIN 功能，这两个功能将允许以接近特权的模式来访问主机。

准入控制器（此部分将在后文进行讨论）可帮助执行这些限制策略。

禁止以 root 用户身份运行租户容器

为了简化管理， Kubernetes 容器默认与主机共用一个用户 ID 命名空间。因此，容器内部的 UID 100 与主机上的 UID 100 相同。UID 0 同样如此（例如 root 用户）。

默认情况下，容器起初以 UID 0 （root 用户）身份运行，这会带来诸多风险。例如，如果未授权用户破坏应用程序，那么他们就能读写容器文件系统中的文件，或远程访问这些文件。如果在容器中挂载任何主机文件系统，攻击者可能会对容器中的任何文件进行读写操作。最后，如果容器以特权模式运行，未授权的攻击者用户可能会获取主机级别的访问权限。这不仅会破坏主机本身，还可能破坏控制平面。

亚马逊云科技建议，针对用于构建租户容器映像的各个 Dockerfile，都应指定一个非 root 用户或 ID 的 USER 指令。此外，亚马逊云科技建议，应该使用特定用户 ID、群组 ID 以及 Kubernetes 容器规范 SecurityContext 中的 fsGroup（相当于群组 ID）来运行每个租户的容器。

> **注意：**针对需要在服务账户中访问 Secrets 或利用 IAM 角色的 Pod，以及并非以 root 用户身份运行的 Pod，都必须在其群组 ID 对应的 securityContext 中指定一个 `fsGroup`。这将阻止与文件所有权相关的权限错误。

准入控制器（此部分将在后文进行讨论）可帮助执行这些限制策略。

限制挂载主机文件系统

容器可以将主机中的卷挂载到容器中。该功能在一些情况下非常有用，但也会带来重大风险。

首先，容器可能能够查看主机或其他容器中的 Secrets 文件。例如，如果将主机中的 /var/lib 卷挂载到容器中，其他容器中的文件（包括 Secrets）也将变为可见。

以 root 用户身份运行的容器对主机文件系统具有无限制写入权限。这可能允许非授权用户修改 kubelet 设置，创建关联到其他敏感位置目录或文件的符号链接（例如 /etc/shadow），安装 Secure Shell （SSH）密钥，损坏重要文件或执行其他恶意活动。

亚马逊云科技建议，除非绝对必要，应该禁止容器挂载主机文件系统。对软件即服务（SaaS）环境下的容器，需要访问主机的情况非常少见。如有必要，亚马逊云科技建议执行只读挂载，防止向主机写入文件。

准入控制器（此部分将在后文进行讨论）可帮助执行这些限制策略。

限制使用主机网络并阻止访问实例元数据服务

默认情况下，EC2 Instance Metadata Service （IMDS）能够访问所有 EC2 实例。 该服务可提供一些有用的自检功能，例如确定节点的可用区、实例 ID 等等。此外，IMDS 提供 IAM 凭证访问权限，允许应用程序承担实例的 IAM 角色。

默认情况下，EKS 集群中的每个 EC2 节点都会获得某些必要的特权，从而完成启动并向 Pod 分配 IP 地址。例如，节点可以连接 VPC 网络接口，并发现与所连接的 EKS 集群相关的信息。尽管仅在有特权时才能有效运行这些节点，但通常不建议节点上运行的 Pod 继承这些特权。

阻止 Pod IMDS 访问权限的一种方式是，应用 Calico 等附加组件执行网络策略，从而确保 Pod 无法访问实例元数据服务。为此，请将您的网络策略配置为阻止 169.254.0.0/16 的出口流量。

阻止 Pod IMDS 访问的另一种方式是，使用 IMDS 版本 2（IMDSv2），并将最大跳数设置为 1。如果这些 Pod 不使用主机网络，以这种方式配置 IMDS 时，Pod 向 IMDS 发出的请求将被拒绝。

此外，亚马逊云科技建议禁止不受信任 Pod 使用主机网络。准入控制器（此部分将在后文进行讨论）可执行这项限制策略。

限制使用外部 IP 地址创建服务

Kubernetes 的一项核心功能是服务抽象。在集群中创建指向 IP 地址且所有 Pod 均可见的域名系统（DNS）条目，在一定程度上能实现服务抽象。这些 IP 地址可能为 Pod，也可能为外部地址。

此外，如果任何 Kubernetes 服务使用外部 IP 地址，都会导致系统将从集群的任何 Pod 流向该地址的所有流量发送到该服务——即使该 IP 地址实际上属于第三方。

例如，请考虑地址为 1.2.3.4 的假定互联网服务情形。如果租户创建了外部 IP 地址为 1.2.3.4 的 Kubernetes 服务，则从集群内部流向 1.2.3.4 的*所有*流量都会被该服务拦截。这可能会导致严重的中间人（MITM）攻击安全风险。

亚马逊云科技建议禁止租户创建使用外部 IP 地址的服务。客户可以使用准入控制器执行这项限制策略，包括 GitHub 上提供的控制器。

此外，亚马逊云科技建议禁止租户修补任何 Kubernetes 对象的任何状态字段。通常情况下此选项禁用，但请注意，不要通过任何集群 RBAC 策略启用此选项。

应用 Seccomp 配置文件至容器

Seccomp 为 Linux 内核功能，可以限制程序作出未授权的系统调用（syscall）操作。Syscall 是程序与 Linux 内核的交互方式。例如，如果程序希望写入标准输出，则可以使用 write(2) syscall。许多 syscall 都是无害的，但也有一些 syscall 可用于升级特权，调整内核设置或执行其他非预期操作。

默认情况下，容器将会“无限制”运行，因此可以调用任何 syscall。但是，亚马逊云科技建议启用容器运行时提供的默认 Seccomp 配置文件。该配置文件允许大多数系统调用，但会排除一些被认定具有高风险的系统调用。请参阅面向 Docker 的 Seccomp 安全配置文件，了解默认允许和拒绝的 syscall 列表。

如需启用此配置文件，请在各 Pod 或容器的 SecurityContext 中，指定一个类型为 RuntimeDefault 的 seccompProfile。请参阅为容器设置 Seccomp 配置文件，了解详细信息。

用户也可使用自定义 Seccomp 配置文件来运行容器。这可用于进一步限制可能被调用的 syscall，或允许启用可能被禁止的 syscall。可使用 strace(1)或 Sysdig Inspect 等工具，确定应用程序可调用的 syscall。

将 SELinux 配置文件应用至容器

SELinux 是 Linux 中提供的一项安全增强功能。该功能最初由美国国家安全局（NSA）开发，旨在提供对操作系统的强制访问控制权限。

SELinux 远远超越了基本的 UNIX 权限模式，在进程和文件中引入了标记概念，并使用一些精细化策略来控制进程在访问文件和执行敏感操作时所需的权限类型。如果策略允许操作，则将授予相应权限。否则，即使 UNIX 权限模型允许操作，系统也会拒绝访问权限。

亚马逊云科技建议在托管多租户 EKS 工作负载的 EC2 实例中启用 SELinux。这需要启用 SELinux 的 Linux 发行版，例如 Bottlerocket、Red Hat Enterprise Linux 7 或更高版本、或者 CentOS 7 或更高版本。在非 Bottlerocket 发行版中，也需要使用启用 SELinux 的容器运行时引擎，例如 Docker CE 19 或更高版本。 SELinux 目前不适用于 Amazon Linux 2。

启用 SELinux 时，大多数非特权 Pod 将自动应用各自的多类别安全（MCS）标签。各 Pod 的 MCS 标签都是唯一的，以确保一个 Pod 中的进程无法操纵任何其他 Pod 或主机中的进程。即使经标记的 Pod 以 root 用户身份运行，且能够访问主机文件系统，也无法操纵文件，无法在主机中作出敏感系统调用，无法访问容器运行时引擎，且无法获取 Kubelet 的密钥材料。

这里提供了一个为 Pod 配置 SELinux MCS 标签的示例。在该示例中，类别 ID 为 c123 和 c456，您可与一个唯一 Pod 相关联。（SELinux 需要使用至少具有两个类别 ID 的进程）。

```
securityContext:
  seLinuxOptions:
    level: "s0:c123,c456"
```

> **注意**：亚马逊云科技建议为集群中的每个 Pod 分配唯一 MCS 标签。也存在无法自动应用 MCS 标签的特殊情况，例如容器启用 `hostPID` 标记时。
>
> 特权 Pod 进程具有 SELinux 标签：(`system_u:system_r:spc_t:s0`)，因而能够完全访问容器主机。因此，仍需要为 SELinux 补充附加控件，以阻止创建特权 Pod 或启用 `hostPID` 标记。
>
> 在运行 SELinux 的节点中，亚马逊云科技 VPC Container Networking （CNI）控制器必须以特权模式运行。

使用准入控制器执行安全策略

准入控制器是 Kubernetes 中的一项强大功能。这些控制器可拦截创建新对象或变更集群现有对象的请求，并会采取一项或多项操作。准入控制器可以修改请求，从而符合指定策略（“Mutating webhook”），也可以共同拒绝请求（“Validating webhook”）。

亚马逊云科技 VPC Container Networking （CNI）建议运行多租户集群的客户实施以下一项或两项安全策略执行机制。

Pod 安全策略（PSP）

每个 EKS 集群都内置能够执行 Pod 安全策略（PSP）的准入控制器。Pod 安全策略为集群管理员可以创建的普通 Kubernetes 对象。有关详细信息，请参阅 Pod 安全策略。

以下为禁止运行特权 Pod 的策略示例：

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: DisallowPrivilegedPods
spec:
  privileged: false
  # The rest fills in some required fields.
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
```

```
  runAsUser:
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
  volumes:
  - '*'
```

有关更复杂的策略，请参阅本文档的附录部分。此策略可以执行以下操作：

- 禁用特权 Pod
- 禁用特权升级
- 要求删除所有能力
- 禁止挂载主机卷
- 禁止使用主机网络、与主机进行进程间通信（IPC）以及使用主机进程 ID（PID）
- 禁止以 root 用户身份运行
- 要求使用默认 Seccomp 配置文件

默认情况下，EKS 会提供无限制 Pod 安全策略。亚马逊云科技建议删除应用于所有认证用户的默认集群角色绑定策略-eks.privileged。如需要进行此操作，您可编辑 eks:podsecuritypolicy:authenticated 集群角色绑定，从对象列表中删除 system:authenticated 群组。如果您已为集群创建了替代管理员群组，则可使用您的管理员群组替代 system:authenticated 群组：

```
apiVersion:
rbac.authorization.k8s.io/v1 kind:
ClusterRoleBinding
metadata:
  name: eks:podsecuritypolicy:authenticated
  annotations:
    kubernetes.io/description: 'Allow all authenticated users to
create privileged pods.'
  labels:
    kubernetes.io/cluster-service: "true"
    eks.amazonaws.com/component: pod-security-policy
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: eks:podsecuritypolicy:privileged
```

```
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    # Replace this with your administrator group
    name name: system:authenticated
```

> **警告**：在对您的集群进行上述或其他变更时，请务必谨慎操作。除非创建替代策略、适当角色和/或角色绑定，否则这可能会阻止您创建新的 Pod。

开放策略代理（OPA）

开放策略代理（OPA）是一种强大的开源通用型策略代理。OPA 的核心功能是，使用特定域语言 Rego，并根据您所定义的规则集合来评估配置。OPA 能够灵活处理几乎任何类型的结构化数据，但其最常用的功能是执行 Kubernetes 集群内部策略。

与 Pod 安全策略相比，OPA 可提供更为广泛的策略管理功能。PSP 限制应用于 Pod，而 OPA 可以管理几乎所有类型的 Kubernetes 对象。PSP 仅能向 Pod 应用受限的策略集合，而 OPA 可以向对象中的任何字段应用强大的验证器功能（例如模式匹配器）。例如，使用 OPA，您还可以请求从受信任的镜向库中提取所有容器镜像。

以下为禁止创建特权容器的 Rego 策略示例：

```
package kubernetes.admission

deny[message] {
    # match only if a Pod is being created
    input.request.kind.kind == "Pod"

    # examine each container
    container := input.request.object.spec.containers[_]
    # match if privileged is set
    container.securityContext.privileged
    message := sprintf("Container %v runs in privileged
mode.", [container.name])
}
```

OPA 正在快速发展演进。客户可选择在 EKS 集群中运行不同的实施工具。Kube-mgmt 是最早的实施工具，目前仍广泛使用。Gatekeeper 是最新的实施工具，具有强大的模板配置模型。

结论

有多种方法和途径可用以保护 Amazon EKS 集群中的多租户工作负载。确保完全隔离互不信任的工作负载的最佳方法是，为每个租户运行专有的 EKS 集群。无论使用何种方法，您都可以运用多种缓解控制措施，从而帮助提高共享集群中多租户工作负载的安全性。

多项可用于改进容器隔离的新型技术即将面世。Firecracker（亚马逊云科技内置开源轻量级虚拟机管理器）和 Bottlerocket（亚马逊云科技内置面向容器的开源 Linux 发行版）等技术也正在开发中。针对在 Kubernetes 中运行独占多租户工作负载的亚马逊云科技客户，亚马逊云科技期望这些客户最终能够在生产级别解决方案中融合利用这些技术。如有可用的解决方案，亚马逊云科技将会及时提供更新信息。

附录：面向不受信任租户的严格 Pod 安全策略

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: Tenant
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames
: 'docker/default,runtime/default'
    seccomp.security.alpha.kubernetes.io/defaultProfileName
: 'runtime/default'
spec:
  privileged: false
  # Required to prevent escalations to
  root. allowPrivilegeEscalation: false
  # This is redundant with non-root + disallow privilege
escalation,
  # but we can provide it for defense in depth.
  requiredDropCapabilities:
```

```
    - ALL
  # Allow core volume types.
  volumes:
    - 'configMap'
    - 'emptyDir'
    - 'projected'
    - 'secret'
    - 'downwardAPI'
    # Assume that persistentVolumes set up by the cluster admin
are safe to use.
    - 'persistentVolumeClaim'
  hostNetwork: false
  hostIPC: false
  hostPID: false
  runAsUser:
    # Require the container to run without root
    privileges. rule: 'MustRunAsNonRoot'
  seLinux:
    # This policy assumes the nodes are using AppArmor rather
than SELinux.
    rule: 'RunAsAny'
  supplementalGroups:
    rule:
    'MustRunAs'
    ranges:
      # Forbid adding the root group.
      - min: 1
        max:
  65535 fsGroup:
    rule:
    'MustRunAs'
    ranges:
      # Forbid adding the root group.
      - min: 1
```

编著人

本文档的编著人为：

- Michael Fischer，亚马逊云科技资深专家解决方案架构师（容器领域）
- Tod Golding，亚马逊云科技首席合作伙伴解决方案架构师（SaaS 领域）

文档修订

日期	说明
2021 年 6 月 4 日	第一版