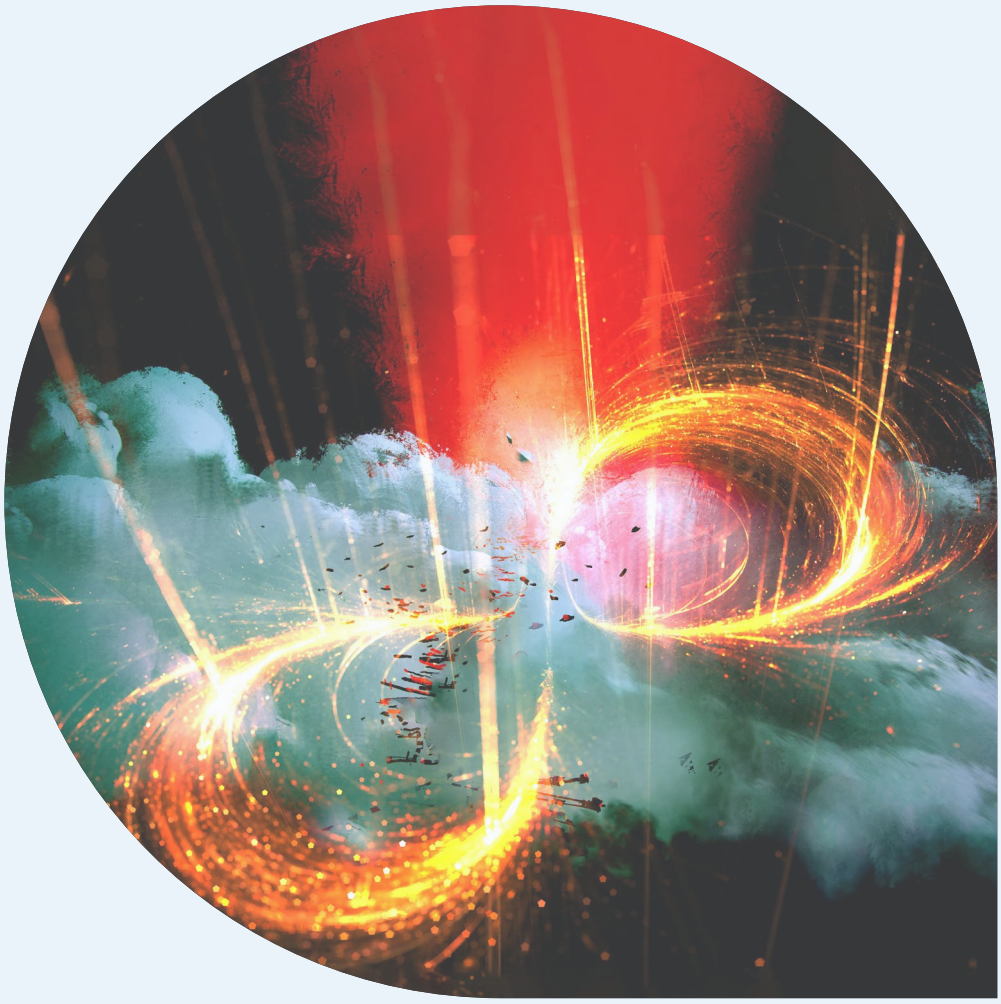
aqua

DevOps 一定要关注的 10 件要事



导言

容器技术颠覆了整个软件开发的进程，根据 Gartner 的调查，在 2017 年底，已有 65% 的企业将容器投入生产，另有 21% 的企业计划在 2018 年及以后投入生产。

容器的发展改变了各组织开发和部署应用程序的方式。就在几年前，大多数部署还是由使用 Docker 创建和管理的独立未开发应用程序所构成。

如今，组织已采用 Kubernetes 等工具将应用程序的交付流程转变为灵活、可扩展的自动化流程。

这不仅对开发和运维产生了广泛的影响，而且对安全也产生了深远的影响。像 Equifax 漏洞这样的数据泄露事件每次都能影响数百万人，对应用程序高安全性的需求从未如此重大。

对于安全部门来说，容器化带来了多方挑战：

- 容器化引入了新基础架构，本质上是开放的动态运行，拥有了更大的可能性进行容器间横向移动。
- 容器化引入了新的流程，可以加快代码在软件开发流水线的移动速度并具有更大的灵活性（和更少的监督）。
- 容器化引入了新身份，能够以开发者和 IT 运维的形式访问资源，这可能会对运行时环境产生直接影响。

从交付的角度来看，这些都是有助于减少部署时间和应用程序资源占用的因素，让运维和开发之间变得更加平滑。

但这种缩短的流水线带来的挑战之一就是它忽略了安全性。DevOps 的目标是尽快地改善部署并减少维护。因此，妨碍快速部署流程的行为被开发者视为了一种障碍。52% 的开发者担心应用程序安全会让开发进程变缓并可能导致无法按时交付，这使得安全团队很少对 DevOps 流程拥有可见性。

开发

- 对构建环境的威胁
- 平台安全
- 微服务网络拆分
- 职责分离

测试

- 加强用户访问控制
- 强化主机

运行

- 管理镜像中的漏洞
- 减少容器攻击面
- 机密管理
- 容器运行时安全

在企业环境中，仅仅相信开发者不会带来漏洞是不够的。产品必须满足合规性、报告和威胁迁移的严格要求，而安全团队验证这一切的唯一方法就是对 DevOps 流程的完全可见性。DevOps 和安全的这种结合被称为 DevSecOps，已有 63% 的组织在内部拥有正式或非正式的 DevSecOps 团队。

关于确保容器化应用程序安全，DevOps 应了解以下 10 件要事

1 对构建环境的威胁

尽管构建环境是 DevOps 流水线的重要部分，但它经常被认为是某项安全责任。传统上，构建环境的安全风险较小，因为它与生产环境并没有直接的关联。随着公司变得更加灵活，采用持续集成和持续开发 (CI/ CD) 实践，现在构建环境需要每天多次将更改部署到生产环境，而这，就会直接访问安全资源，随着攻击者发现要直接击穿高度安全的生产环境变得更加困难，就使得构建环境成为了攻击者更具价值的攻击目标。入侵到构建环境的攻击者不仅可以引入恶意代码，还可以获得特权帐户、内部服务和公司机密的访问权。

由于构建过程涉及了众多步骤，漏洞可能会在任何地方被入侵：泄露的源代码、恶意库、宽松的访问凭据或不安全的构建服务器。容器通过将应用程序与其运维环境合并，加剧了这个问题。虽然这简化了部署过程，但它创建了一个抽象概念，使安全团队更难审计底层代码。安全团队需要立即分析应用程序、容器环境、用于构建容器的镜像、构建工具和构建环境。在不显著影响周转时间的情况下，能够做到这一点的唯一方法是让安全团队自始至终深入了解构建过程。

DevOps 团队需要尽力保护在构建过程中使用的资源，以确保应用程序从源代码到最终 artifact 不受损坏。对构建环境的访问特权，应仅限于少数授权用户。需要访问代码存储库、公司服务器和其他安全资源的构建工具，也应仅限于有访问权限的用户帐户，并与其他系统隔离开来。

2 管理容器镜像中的漏洞

镜像是用于实例化容器的静态代码文件。因此，镜像中存在的任何漏洞都一定存在于镜像实例化的每个容器中。从安全的角度来看，镜像必须保证安全，否则整个应用程序都将面临着风险。在来源处保护镜像也更容易、更有效，而非追求庞大且多变的“富”容器。

镜像用在基于层级的文件系统存储。每一层都代表其下一层的更改，就像叠加文件。在镜像和容器之间可以重复利用层级，以减少下载和部署时间。然而，每一层也都增加了漏洞进入最终容器的风险。基础镜像中的漏洞可以很容易地进入众多容器中。

使用 Docker Hub 等公共镜像仓库会带来额外的风险，它允许开发者下载镜像并与数百万其他用户共享镜像。其中的许多镜像，特别是像 Ubuntu 这样的基础 Linux 镜像，构成了许多其他镜像的基础层，而这些基础层又构成其他镜像的基础。除非控制原始源的访问，否则，想要对所有这些层的进行保护就会立刻变得困难重重。

当使用公共镜像仓库时，DevOps 团队应该只从可信来源下载签名镜像，这些镜像在传输过程中不会被篡改。镜像应该经过全面的安全扫描，并固定到特定的版本，以防止未经验证的更新带来漏洞。除此之外，建议只允许指定的开发者访问公共镜像仓库，并拥有一个内部镜像仓库，该镜像仓库仅存储供其他开发者使用的可信基础镜像。

3 减少容器攻击面

尽管容器提供了一定程度上的隔离，攻击者仍然可以利用漏洞或配置不当的容器访问主机系统。例如，Docker 容器默认以 root 用户身份运行。虽然这个 root 用户的权限比主机 root 用户少，但它仍然可以访问挂载资源和主机内核。攻击者可以利用这些入口点来访问安全文件或利用漏洞来获得主机上的管理访问权限。

容器化的优点之一是将用户命名空间与主机系统隔离开来。这种隔离使要被定义和管理的用户和组，可以独立于主机中的用户和组。在容器中创建的用户帐户仍然可以访问容器资源和文件，但攻击面比 root 用户小得多。

DevOps 必须遵循最小权限原则，并且只允许运行容器化应用程序所需的权限。这样做的好处有两个：如果攻击者能够访问容器，他们与容器交互的能力将会受到限制。因此，攻击者会发现要从容器突破到主机会更加困难。

4 加强用户访问控制

默认情况下，CloudTrail 是为 AWS 账户启用的，但同样，在默认情况下，它也会限制用户对容器的访问，这是因为具有不同信任级别或敏感度的容器，以及与不同应用程序相关联的容器，可能在同一个集群或同一主机上运行。

身份验证和授权都应该受到控制，以限制哪些用户可以访问哪些组或类型的容器，以及一旦用户有了访问权限，他们的权限应该允许他们做什么。例如，构成 PCI 兼容应用程序的容器，需要与构成其他非 PCI 应用程序的容器分开来进行访问控制。此外，审计员或合规团队成员可能需要了解环境（查看正在运行的流程、查看审计日志），但又不应拥有启动或停止容器或更改 Docker 环境配置的能力。

这些控制可以通过使用外部用户访问控制系统来解决，允许为用户配置粒度权限，而非提供 root 级别的访问。例如，Docker EE[1] 和 Kubernetes[2] 都支持对各自环境中的资源进行基于角色的访问控制，但它们必须采用集中式策略进行管理。

如果没有运用集中式的方法，就很难确定分配给用户的权限是否与其职能角色一致，尤其是当这些角色会随时间发生变化的时候。

1 https://docs.docker.com/ee/ucp/authorization
2 https://kubernetes.io/docs/reference/access-authn-authz/rbac

5 机密管理

有些容器在运行期间需要访问敏感数据。这些敏感数据 (称为机密数据) 可以包括凭据、令牌和密码。安全分发机密的解决方案已经存在，但容器的短暂性使这个过程更具挑战性。

很多时候，机密被硬编码到源代码、镜像或构建过程中。虽然这便于测试，但由于无法预先知道镜像的最终位置，它也有意想不到的副作用。嵌入在镜像中的机密可以传播给任何能够访问镜像的用户，甚至当镜像存储在私有镜像仓库时也是如此。而将机密限制在构建工具中又会限制开发者在本地系统上测试应用程序的能力。

一个好的机密管理解决方案能既允许 DevOps 团队将机密存储在一个安全的集中存储库中，同时又确保相关容器能够访问到所需的机密，这些机密在其他任何地方都无法访问。这样一来，就能减少不安全容器泄漏机密的机会，或者滥用机密以获得对受限资源的访问的机会。

6 强化主机

尽管容器提供了自己独立的环境，但它们通常仍运行在主机操作系统上，共享内核资源。因此，应该强化主机以抵御来自容器或容器运行时的潜在攻击。

主机安全很大程度上取决于操作系统 (OS)。强烈建议将运行容器的主机专用于运行容器，并且不要将 容器化工作负载与非容器化工作负载混合使用。这样可以让控制访问和使用编排器变得更加容易。对于 运行容器，建议使用“精简操作系统 (thin OS)”，即为了运行容器而优化的 Linux 发行版，并且不包含完整企业 Linux 发行版中许多不必要的功能，例如 Red Hat CoreOS、Rancher OS、和 VMware 的 Photon OS。与完整操作系统相比，精简操作系统不仅减少了攻击面，而且提高了速度，降低了资源占用。某些最小化操作系统还可提供为容器量身定制的功能，例如本地集群工具、只读 root 文件系统以及对 Docker 和其他运行时的本地支持。

大多数容器运行时通过利用 Linux 访问限制过程进行额外的强化。关键组件是命名空间和控制组（cGroup）。本质上， cgroup 决定了一个容器可以使用多少共享内核和系统资源，而命名空间决定了容器可以访问哪些资源。建议使用 seccomp 配置文件，可以限制对 Linux 内核系统调用的访问，改善容器到主机的隔离，并防止对容器的内核攻击。

在生产环境中，用户访问也应严格限制在“break glass”的场景中。在正常操作下，没有任何理由让任何人以 SSH 管理员身份进入主机或手动配置它。主机应该通过配置管理模板（例如，使用 Chef 或 Ansible ）进行管理，并且只有编排器具有运行 / 停止容器的持续访问权限。

7 容器运行时安全

容器可以写入文件、启动新进程，以及增加其资源利用率。而获得容器访问权限的攻击者，则可以滥用这种资源，运行未经授权的代码，并在容器的未来实例中进行持续更改。

为了降低这种风险，容器应该保持不变。这种不变性可以防止容器在运行时被修改，如果需要进行更改，则需要部署一个全新的镜像。这确保了从同一镜像创建的所有容器在初次启动时都是相同的，并且每个容器的行为方式基本相同。它还可以防止创建具有自定义或独特配置的“ snowflake ”部署，这些部署会让 DevOps 团队的故障排除变得更加困难。

即使在确保了不变性之后，也应该监控容器的更改、暴露的端口、打开的连接以及其他的泄露迹象。

8 平台安全

容器化应用程序的安全性与容器运行时本身一样。所涉及的步骤将因运行时、运行时运行的主机操作系统以及使用的任何附加功能或插件而异。然而，一些组织已经发布了优化平台安全的指南。

其中两个比较流行的审计工具是互联网安全中心（CIS）Docker 基准 [1] 及其 Kubernetes 基准，这是一个以安全方式管理 Docker 安装和 Kubernetes 集群的行业最佳实践的步骤检查列表。每一部分不仅描述了安全漏洞的含义和风险，还描述了如何对它进行检测和修复。包括了更新操作系统组件和应用程序、设置强制访问控制策略，以及执行全面的日志记录和监控。

有一些开源工具可以针对这些基准执行检查，例如 Aqua 的 Kube-Bench 可以对 CIS Kubernetes 基准运行完整的检查列表。

9 微服务网络拆分

单体应用程序通常具有相对静态的网络部署，具有保留的 IP 地址、主机名和端口。然而，容器从编排器获取动态 IP 地址，也可能在节点上频繁出现和消失，具有几个小时的生命周期（而非几天）。容器网络并不是概念化的网络，隐藏在网关后面的物理连接；而是软件定义的网络，以容器服务环境为基础 (通常由编排器提供，或者在更高级的部署中由服务网格提供)，在逻辑连接上运行。

微分段 (Microsegmentation) 涉及将网络拆分为更小的区域，并对每个区域应用高级 IT 安全策略。这些策略可以应用于特定类型的流量，甚至可以应用于特定容器。这使 DevOps 团队能跨复杂的分布式应用程序来管理端口、解析 DNS 地址、创建负载平衡器和代理请求。

如果攻击者确实获得了对联网容器的访问权限，分段可以确保暴露给攻击者的资源数量比其他情况下要少得多。为了安全起见，预防性的微分段通常是不够的，为了不仅仅采取被动措施，想要主动监控和阻止试图穿越网络的攻击者，还应在容器化服务之间实施额外的防火墙控制。

---

1 https://kubernetes.io/docs/reference/access-authn-authz/rbac

10 职责分离

职责分离 (通常称为 SoD) 是一个安全原则，它规定任何个人或团队都不应该对系统具有无限制的访问特权。通常这意味着，管理和运行服务器的系统管理员将不负责服务器的安全，他们也没有权限更改安全设置，这项工作将由安全团队执行。

对于容器，由于 DevOps 方法的大量使用，加上有时候 DevSecOps 方法会将安全进程合并到 DevOps 中，边界通常会变得很模糊。然而，这并不意味着 SoD 不应该被严格执行，事实上，它应该比以往任何时候都更严格地执行，因为由恶意内部人员和粗心大意所造成的损失已经成倍提高了。

例如，在许多 Kubernetes 设置中，没有内置的 SoD 和集群管理是“非常强大的”。这会在企业生产环境中造成不可承受的风险，必须要采取适当的控制措施，允许安全团队监控和执行安全策略、评估风险，并防止管理人员更改集群、Kubernetes 控制台以及访问集群上的节点上的安全控制。

进一步了解 Aqua Security 的 DevSecOps 解决方案 >

Aqua Security 是全球最大的云原生安全厂商之一，总部位于以色列特拉维夫，为客户的创新以及加速数字化转型保驾护航。Aqua 平台在整个应用生命周期管理中提供预防、检测和响应自动化服务，以确保供应链、云基础架构平台和运行时工作负载的安全运行。

Aqua 的客户是全球金融服务、软件、媒体、制造和零售领域最大的企业之一，能够在广泛的云提供商和现代技术堆栈中实施，其中包括容器、无服务器功能和云虚拟机。如需了解更多信息，请访问公司官网 www.aquasec.com 或关注官方微信公众号“AquaSecurity”。