迁移至容器的真正优势 - 第 1 部分

了解容器化的优势

作者：Kim Clark、Callum Jackson

更新日期：2019 年 6 月 14 日 | 发布日期：2019 年 6 月 14 日

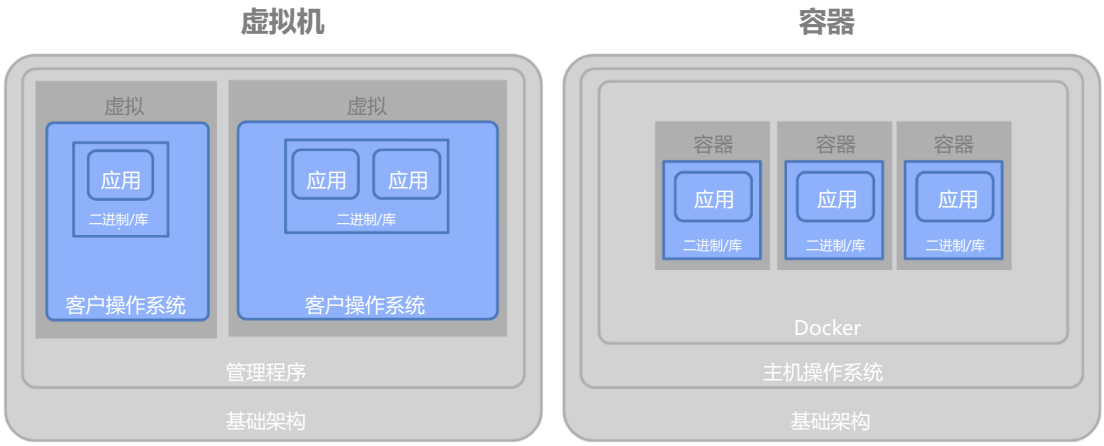
大多数企业已经或者想要制定一个战略，来将部分 IT 资产迁移到容器。这是“云迁移”战略的一个重要分支。企业面临的主要挑战在于，容器化往往被视为基础架构升级，就好像过去的硬件升级一样，甚至会与过去十年向虚拟机的迁移进行比较。对容器的这种观点常常会导致企业采用“直接迁移”的方法，而这将会错过要通过使用容器来实现的所有初衷。在最好的情况下，这会导致容器化运用的执行不力。在最糟糕的情况下，这会导致整个现代化项目失败。

本文探讨了容器化所带来的各种潜在优势。为了获得这些优势，您将学习迁移到容器基础架构之后还需要采取的其他措施。

对容器的理解过于简单导致您误解了容器化的潜在优势

容器化指的是在容器技术（比如 Docker）上运行您的组件，并且大多数情况下，利用容器编排技术（比如 Kubernetes）管理这些容器。

如今，大多数企业都在虚拟机上（VM）上运行很大一部分 IT 资产。思考一下虚拟机与容器的差别。虚拟机映像里包含整个操作系统。相反，容器映像假设大多数操作系统功能都来自机器上的主机操作系统。容器映像只需要包含额外的部分，比如语言运行时的二进制和库，以及应用代码本身。



显然，容器映像比虚拟机映像更轻巧。它占用的磁盘和内存空间更少，能够更快速地启动和停止运行，因为它不会启用另一个操作系统。因此，您很容易得出一个结论：优化基础架构是迁移到容器所带来的主要优势。更糟糕的是，“轻量级容器”理念可能会使得您认为容器本身就有更高的性能。但是，这些理念很容易导致错误的假设，使得您忽略全局。

事实上，如果您的主要目标是尽可能提高某个应用的性能，那么直接在裸机上的主机操作系统上采用传统的安装方式就能提供最佳性能，因为它摒弃了架构中的多个层次。

此外，只有当您在同一资源上部署了多个异构工作负载时，您才能真正做到优化基础架构，因为某个工作负载闲置时，另一个工作负载可能处于繁忙状态，这时您就能平衡两者。对于虚拟机基础架构，您也可以提出同样的观点。因此，如果您已经在采用成熟的虚拟机环境处理大量不同的工作负载，那么简单地将这些工作负载直接迁移到容器中可能无法帮助您大幅提高效率，除非您另外采取一些其他的措施。

为了确保成功实施容器化计划，您需要了解容器化的各种优势，容器如何改变企业开发和构建解决方案的方式，以及除了“直接迁移”的方法外，您还需要采取哪些其他的措施。

一个广泛的容器化战略能带来哪些潜在优势？

通过采取正确的方法，迁移到容器能为您提供一个革新机会，具体来说，容器化战略能在以下领域提供更广泛的优势：

1. 敏捷性和生产力：加快开发速度，提高跨环境的一致性，助力自主管理型团队提高生产力和质量。
2. 细粒度弹性：单独部署高度可用的组件，以消除单点故障。
3. 可扩展性和基础架构优化：实现细粒度动态扩展和最大化的组件/资源密度，从而充分利用基础架构资源。
4. 运营一致性：以统一方式管理异构组件，从而减少运营环境所需的各种技能集。
5. 组件可移植性：利用跨节点、环境和云的可移植性，确保选择正确的平台。

下面，我们将详细探讨每一个领域。

优势 1：敏捷性和生产力

容器部署的轻量级属性有助于团队大幅提高速度，让团队能够以可靠的方式让新节点或工具从开发阶段进入生产阶段。但是，这种方法需要您改变组件的粒度和组成，以及组件的构建和部署流程。此外，开发团队还需要加快行动速度。

容器技术本身就能直接提供生产力优势，尤其是在配置新资源和部署代码的速度方面。比如，您可以思考一下配置、启动和扩展新容器映像的速度；通常只需要几秒钟。相比之下，由于复杂的企业程序和资源分配的控制，配置新的虚拟机通常需要几个小时、几天甚至几周。但是，如果您想获得容器化的所有优势，容器技术本身提供的优势还只能算是冰山一角。

您需要将大型的代码孤岛分解成细粒度组件。借此，不论是在哪个特定的发布周期，您需要构建、测试和部署的代码都要更少，如果设计合理，您还能减少对其他组件的依赖。这样，构建流程变得更简单、更快速，也更容易理解。您可以更快速地测试变更，诊断问题，然后交付修复方案。

随着组件数量的增加，Kubernetes 这类容器编排平台无疑变得更加关键，以便您能够统一处理所有组件，而不需要一个个处理它们。

借助企业分散化模式，团队自行拥有代码，这样，他们就能更快速地将要求与组件进行映射。他们尽可能独立、自由地构建组件。他们可以独立部署组件，并且确信它们不会影响其他组件。他们不需要与其他团队合作安排部署、共享环境，也不需要担心共享软件库的版本。

基于映像的部署大幅提高了一致性和交付内容的质量。运行时语言和操作系统的代码与版本之间的任何重要的依赖关系都会有效地融入映像内。该结构大幅减少了因传统共享服务器环境中出现的配置差异所引发的问题。

当然，即使再多的技术也不能更快地将功能投入生产，除非流程得到了妥善简化。您必须采用敏捷开发方法，以减少不同团队之间的工作移交和相互依赖。

通过结合利用敏捷方法与管道自动化技术，您能够实现持续集成和部署(CI/CD)，因为您需要采用高效的机制来构建和部署代码的定期迭代。

总而言之，您可以在敏捷性和生产力方面获得下文所述的各项优势。但是，除了迁移到容器基础架构外，您还需要采用企业分散化模式，重新打包细粒度组件，利用容器编排平台管理组件，并转而采用敏捷开发方法，后者用自动化管道和基于映像的部署进行了优化。借助这些方法，您可以获得以下优势：

- 加快构建速度
- 缩短维护周期
- 实现跨环境的一致性

- 独立部署组件
- 简化独立组件的测试
- 异构运行时版本
- 标准化构建管道
- 标准化部署机制
- 以自助方式配置资源
- 通过分散控制，赋予团队自主权
- 清晰的组件范围和所有权
- 自由选择技术

优势 2：细粒度弹性

如果只是简单地将孤岛式应用迁移到容器中，您不能获得多少额外的弹性。一个功能的内存泄露可以会危害整个应用。通过利用细粒度组件，您有机会更分散地提供弹性。您可以安全部署新功能，或者变更现有功能，而不危害或影响另一个功能。即使一个组件出现故障，它也不会影响其他组件。而且，细粒度组件包含的代码和库更少，借此，您能够显著缩短重启时间。

您只需使用这种快速恢复功能，就可以提供高可用性，而无需在其他专用的冷或热服务器上浪费任何资源。如果不想以非常基本的方式提供这种恢复功能，您无疑需要容器编排功能，从而采用更先进的恢复策略，其中包括在需要时的位置限制，确保跨越物理节点和地区合理分配副本集。

要利用编排功能实施清洁的恢复，容器必须当作可支配组件实施，让您可以随意停止、启动和迁移容器。

因此，要想获得细粒度弹性的所有下述优势，您必须将应用分解成细粒度组件，通过对这些可支配组件的容器化编排来实现管理。

- 快速重启，从而进行恢复，而非使用高可用性（HA）结对。
- 安全独立地部署，避免给现有组件带来不稳定的风险。
- 利用可支配组件，支持快速启动/停止，实现简单的 HA 和扩展。
- 实用的分散型拓扑。
- 基于复制策略和内置的恢复功能，实现高可用性。
- 利用滚动升级、金丝雀发布（canary release）和 A/B 测试，实现细粒度推广。

优势 3：运营一致性

通过采用容器，您将获得统一的组件类型，从而在运行时管理组件。外面的容器看起来大同小异。

您不再需要详细了解产品或运行时环境特定的命令行工具，就能执行构建和部署。您不需要构建和维护复杂的运行时特定拓扑，就能实现负载平衡和高可用性。不论容器内有什么功能，也不论您采用什么方式构建、部署、升级和提供高可用性，您都能以完全相同的方式完成扩展。这就是运营一致性。您不再需要成为产品或运行时专家，就能监控和管理运行时环境。您只需要了解通用容器编排平台（如 Kubernetes）的工作原理。

要想实现这一目标，您需要采用全新的方法来处理部署。借助基于映像的部署方法，您可以将在单一轻量级映像内实例化和运行组件所需的一切进行封装，包括功能本身与运行功能的运行时环境。借助这种完全封装的组件，您不需要了解任何有关运行时环境或环境内代码的知识，就能基于任意映像或集群内的任意节点，利用 Kubernetes 这类平台快速构建、启动和扩展容器。

通过将您的拓扑配置与容器（基础架构即代码）紧密关联，操作人员不需要了解拓扑结构的具体特点（比如可用性、扩展、节点关联性），就能部署拓扑。比如，

Helm Chart 中嵌入的基础架构要求使得容器编排平台能够设置拓扑，进而以标准化方式提供所需的非功能型特点。

设计可支配组件至关重要，这样，容器编排平台能够高效恢复容器，确保可用性；引进新容器，实现弹性的扩展；并执行清洁的升级和回滚。借此，操作人员能够使用标准的机制执行所有上述操作，而非使用不同的技术处理单个运行时环境。

借助容器编排平台，您可以利用基于映像的部署，实现统一的构建和部署；提供容器和基础架构即代码，以一致的方式构建拓扑，并自动维护拓扑；确保您设计可支配组件，实现标准化扩展和高可用性。因此，您可以在运营一致性方面获得以下优势。

- 基于标准化容器平台的部署、负载平衡，高可用性、扩展、滚动升级和回滚、日志记录、监控、使用情况指标、端点暴露和组件间安全性等等。
- 统一管理异构节点。
- 跨越所有部署的产品和运营时环境的统一基础架构平台技能集。
- 采用更简单的流程，将技能从一个运行时环境传输到另一个运行时环境。
- 轻松获取实现端到端 DevOps 所需的技能。
- 提高跨环境的一致性。
- 自动构建、部署和管理拓扑。

优势 4：可扩展性和基础架构优化

使用虚拟机时，您需要启动整个操作系统，才能开始处理任何工作。在运行的操作系统中启动和停止容器，这意味着，凭借容器的轻量级属性，您可以在数秒内构建和摧毁容器。这种结构有利于您获得动态的可扩展性，进而实时应对工作负载变更。

借助容器编排平台，比如 Kubernetes，您可以指定复制策略，然后根据需要向上和向下扩展容器。结果就是，每个容器只能使用它所需要的资源，从而解放任意闲置资源，供平台上的其他容器使用。

只有将应用分解成细粒度组件，您才能快速启动，进而实现弹性的扩展。此外，细粒度组件还允许容器平台向上扩展欠载的功能，而非复制整个应用和所有资源。

此外，容器上运行的组件必须设计成可支配组件，这样，编排平台就能在不产生负面影响的情况下，随意引入和移除容器。

总而言之，如果您将组件设计成可支配组件，则只有平台能够高效扩展迁移到容器中的组件。此外，它们将继续使用现有的方式利用当前效率低下的资源扩展组件，除非应用能够分解成细粒度组件，您才能实现真正差异化的扩展。当然，扩展和优化本身主要由重要的容器编排平台来提供。您将获得以下优势：

- 细粒度扩展每个功能。
- 实现最大化的组件/资源密度。
- 相比使用虚拟机，隔离开支更低。
- 动态、弹性地配置资源（CPU、内存和持久卷）。

优势 5：组件可移植性

可移植性指的是将组件迁移到不同平台的便捷性。但是，“迁移”可能是复制或者重构，这是两种目的不同、形式不同的可移植性。

复制可移植性指的是，在保持不变的前提下，组件在另一个终点运行。创建容器映像后，容器平台可以轻松将映像复制到平台上的任意节点上，并启动创建容器。但是，如果只是在容器内运行现有的软件，您不一定需要这种可移植性。从根本

上来说，获得复制可移植性意味着，迁移到基于映像的部署，将容器映像当作部署单元处理。

要想实现上述可移植性，您必须依赖小型映像（从而更快速地生成文件副本），并实现快速启动。要实现这种可移植性，迁移到容器基础架构不是唯一的方式。您还可以重构更小型的细粒度组件，将它们设计成可支配组件，从而允许编排引擎随意启动和停止组件，实现扩展或再分配。

拓扑的特点还必须指明为基础架构即代码，以确保编排引擎能理解有关容器安置位置的限制条件。比如，您需要知道最少需要多少副本；是否需要跨地区分配副本，以获取额外的弹性；或者是否有任何地理隐私问题。

但是，在容器的复制可移植性方面，您需要注意一点。容器映像可以自由地在同类型的操作系统（比如在不同的 Linux 系统之间，或者在不同的 Windows 系统之间）和硬件之间迁移，前提是容器映像基于同样的核心架构构建（比如，在 x86 架构之间迁移或者在 AMD 之间移动）。通常，这种方法适用于云平台，这样，您就可以在所有类型相似的环境中选择所有节点。但是，如果跨越这些界限迁移，映像就不能按原样运行。作为替代，可通过为新的基础操作系统或硬件重构映像来获得可移植性。组件依然可移植，因为在重构映像后，您没有做出任何实际的代码变更，即可在新环境中运行组件。然而，由于您在所有环境中运行的不再是完全相同的映像，因此，这种方法在一致性方面无法做出相同的保证。在该场景下，您能够期待的最佳结果就是通过自动化管道确保至少构建的流程是相同的，来确保实现一致性，即使最终映像存在平台差异。

容器通常有良好的可移植性。但是，要真正获得这些优势，您至少需要转而采用基于映像的部署，从而以更简单的方式将容器从一个平台迁移或复制到另一个平台。如果您想要快速、动态地迁移或复制容器，那么您需要将应用重构成细粒度组件，从而缩小映像规模，加快启动速度。您需要将它们设计成可支配组件，确保您可以高效地将组件从之前的位置移除。您可能想利用容器编排平台将映像移

植到新节点，这种情况下，基础架构即代码方法能确保您的容器编排平台能清晰地指导将容器移植到何处。最后，如果是在不同的操作系统或硬件架构之间移植容器，那么自动化管道能帮助您确保所有部署的一致性。如果您采取了所有上述其他措施来迁移到容器平台，您将获得以下优势：

- 在特定的云环境内跨节点动态地重新分配容器。
- 您可以在任意容器平台上构建和运行映像。
- 您专注于开放的容器化标准，比如 Docker 和 Kubernetes。
- 您可以采用多云场景。

您还需要采取哪些其他措施，确保成功实现容器化？

在上面的章节中，我们介绍了您需要采取哪些额外的措施，以赢得容器化优势。在迁移到容器架构的同时，您需要实施以下变更：

- 细粒度组件：将现有组件分解成更细粒度元素，从而以完全独立地方式一个个设计、部署、扩展和维护这些元素。
- 容器编排：在您转而采用大量更细粒度组件后，您必须使用 Kubernetes 这样的编排平台来管理这些组件，从而自动执行生命周期管理、扩展、负载平衡、资源配置和其他详细操作。
- 可支配组件：设计无状态且可立即停止的轻量级容器，让编排平台能够处理整个生命周期，而不需要了解有关容器内部工作原理的任何信息。
- 管道自动化：借助细粒度容器，您可以更独立、更快速地实施变更。通过自动构建管道，您可以更快速地进行迭代，实施变更。
- 基于映像的部署：相比部署代码以运行服务器，容器具有轻量级特点，让您可以交付整个堆栈，包括代码、运行时环境和配置。这种交付方法简化了部署和管理，提高了跨环境的一致性。

- 基础架构即代码：您可以在容器平台上为每个部署的容器有效构建独特的拓扑。文件将以声明的方式指明该拓扑的特点，包括扩展、负载平衡、跨区分配、路由和安全性。这些文件必须有效纳入容器内组件的代码库中。这种方法能确保在每个环境中以一致的方式将组件交付至专为满足需求量身定制的拓扑。
- 企业分散化模式：容器编排平台提供的细粒度部署和运营一致性能减少构建和维护解决方案所需的入门知识。这种方法能减少企业对以技术为中心的中央团队的需求。现在，业务团队能够拥有自己的技术团队（影子 IT），更快速地推出新创新成果。
- 敏捷开发方法：显然，敏捷方法与容器之间有协同效应。通过结合利用敏捷方法和容器，拥有自主权（分散式）的团队能构建与业务更契合的快速变更周期。
- 获得自助服务开发者体验：我们必须支持团队独立、轻松地设置所需的资源。当然，这些体验包括团队能够在容器编排平台上配置容器，还包括各种团队资源，比如源代码库、构建自动化功能以及映像库。

如需了解其他措施的具体信息，请参见本系列文章的第 2 部分。

结语

您不应该只是将容器化当作基础架构升级，也不能只专注于资源优化。事实上，单纯采用“直接迁移”方法来实现容器化很少提供有价值的结果。相反，除了迁移到容器外，您还需要采取一系列具体的措施。通过采取这些措施，您将获得更多优势。

有关这些措施的更多信息，请参见本系列文章的第 2 部分。

致谢

非常感谢 Brian Petrini 在我们编写本系列文章时提出的宝贵意见。