迁移至容器的真正优势 - 第 2 部分

成功实现容器化所需的关键措施

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

在第 1 部分中，我们探讨了采用狭隘的“直接迁移”方法迁移到容器不太可能让您获得容器化所带来的诸多优势。本文将进一步介绍您需要采取的一些最常见的其他措施，以迁移到容器基础架构，实现容器化计划的最大价值：

1. 细粒度组件
2. 容器编排
3. 可支配组件
4. 管道自动化
5. 基于映像的部署
6. 基础架构即代码
7. 组织去中心化
8. 敏捷开发方法
9. 自助服务开发者体验

本文将依次探讨上述每个主题，以及它们如何为迁移到容器提供补充。然后，我们还总结了迁移到容器所带来的所有优势。

细粒度组件

容器很轻巧，并且具有很高的可行性，您最好是将现有组件分解成更细粒度元素，这些元素不相互依赖，可以完全独立运行。理论上，您可以利用虚拟机（VM）完成这项任务。但是事实上，如果使用虚拟机，那么资源开支、创建映射所花的时间以及整个操作系统的启动时间都成为了阻碍因素。

在应用开发领域，您可以将一个应用分解成分散、独立的功能（或服务），这是微服务架构的一个核心方面。细粒度组件的设计、部署、扩展和维护相互独立。

值得注意的是，如何确定组件的适当细粒度程度，这本身就是一个复杂的主题。相关的重构工作可能在难度上差异很大，具体取决于在当前设计模式下内部组件的解耦是否得当。

容器编排

任何容器计划都会很快衍生出比您使用的虚拟机数量更多的容器。为什么呢？因为容器的配置和扩展更简单，您可以转而采用更细粒度模式。

现在，您需要容器编排平台，这一点不言而喻，本文以使用 Kubernetes 为例子，因为 Kubernetes 是目前使用最普遍的容器编排平台。

这些平台提供了一个最小的框架，用于高效协调容器内的资源；提供了多个机制来管理容器生命周期和扩展、跨容器实现负载平衡，在容器之间进行路由，并控制容器在容器平台外的公开方式。有些平台提供标准化日志记录，比如通过 ELK 堆栈提供日志记录。最先进的容器编排平台甚至引入了服务网格，比如 Istio，以支持高级路由、嵌入式日志记录和流量管理模式（如 Circuit Breaker）。

可支配组件

在本案例中，*可支配*指的是容器的设计允许所属的编排平台（在本文的案例中为 Kubernete）随意停止容器。平台将用最少的关闭操作尽快停止容器，容器没有任何专门的状态，与相关组件也没有专门的关联性。

在可支配方面还有另一个常用的术语，那就是*无状态*，但是它仅表达可支配性的一个分支，并且经常被人们误解。可支配不仅回答了前面的“为什么”，还传达了更多信息：您必须设计可支配的组件，从而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。这样，编排平台就能以标准化方式在所有类型的容器中实施可用性和可扩展性策略。

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。

使用容器的一大差别在于，您可以轻松将容器定义为分层的文件系统。比如，Dockerfile 中用来定义和构建 Docker 映像的操作将使得现有文件系统映像中新增新文件。

现代运行时环境通常利用脚本执行安装，因此，您不需要使用专有的安装工具，只需简单地将脚本添加到自动化管道中，即可完成安装。

大多数构建都是从存储库（比如 GitHub）中复制文件，然后将文件汇总至容器映像中，而容器映像本身也保存在标准的存储库（比如 DockerHub）中。您可以在源代码库中通过操作轻松触发构建。您可以使用产生的映像存储库更新，触发自动化部署，并测试早期环境。通过使用标准命令（比如文件副本），而非复杂的专有安装、编译和部署命令，您可以更轻松地自动执行并维护构建和部署管道。

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。比如，映像中可能包含您的代码、运行时以及任何相关的配置。这种方法能大幅提升在不同环境中的部署一致性。

与虚拟机映像相比，容器映像的构建更快，启动和停止也更快，存储和移动空间更小（如果您重构为细粒度组件，您可以迁移更多映像）。您有机会以全新的方式向环境交付您的组件。过去，您会将组件部署到运行的基础架构上，比如虚拟机上运行的应用服务器。您不能关闭虚拟机，因为虚拟机中的某些组件归其他团队所有。

而通过使用容器，您可以将整个堆栈融入映像中，其中包括代码中的示例，应用服务器的特定配置，以及语言运行时环境和存储库本身。然后，您可以将其作为自包含单元部署到目标环境中，它的启动很快，并且完全不受其他任何相关因素的影响。这种结构有助于您提高在不同环境中的部署一致性，从而支持您采用标准化部署机制，且不需要掌握任何容器运行时知识，为实现标准化扩展作出贡献。

基础架构即代码

开发人员习惯于将组件部署到现有的共享基础架构上。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。

*基础架构即代码*指的是配置文件以声明的方式指明容器拓扑的所有非功能性特点（比如可用性和扩展），配置文件将与组件的功能代码一并存储。事实上，您也应该将它们视为代码的一部分。

以使用 Kubernetes 为例。*Helm Chart* 将指明何时以何种方式扩展容器（复制策略），如何跨越多个节点扩散容器，哪些组件应该保留整合状态

(Kubernetes pod)，封装界限是什么（Kubernetes 命名空间），以及其他组件可以访问哪些端口。任何可变属性在开发、测试和生产阶段之间的变化都将具体化（Kubernetes configMap 或密钥），并且可被这些环境调用（通常在部署时调用）。这种方法大幅提高了部署的一致性和可预测性，确保环境之间的最大相似性，简化了部署任务的执行，进而提高部署自动化水平。

组织去中心化

组织去中心化对于帮助团队制定自己的决策、全面实现端到端的组件所有权至关重要。您需要将严格的治理和控制解绑，否则往往会使得企业采用“一刀切”策略来简化复杂 IT 架构的管理。为了高效开展工作，团队需要针对其使用的技术、框架和方法，迅速作出自己的决策。此外，他们还需要自行配置所需的功能，这一点同样至关重要。

下面，我们将介绍如何利用架构、设计和技术的变化，支持团队的自主管理。但是您首先需要重组企业结构，以适应这一变化。根据业务需求组建小型团队，每个团队都有决策者能够自行制定决策。当然，请注意这种去中心化并不适用于每家企业。仅企业的某些部门采用这种模式，或者企业实施特定类型的计划，这两种方式可能更合适。

一方面，您需要支持企业的某些团队开展创新和探索，维持市场领导地位；另一方面，您需要确保持续不断的变化和与日俱增的差异性不会影响企业核心能力的完整性。您需要在两者之间找到一个平衡，这是您的最终目标。您需要不断提高整个企业所用方法和技术的一致性。微服务和敏捷方法通常能通过“协会”满足您的这一需求，这些协会由来自不同团队的个人组成，这些团队致力于根据资深团队的切身经验，推广（而非实施）通用方法和工具。

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（去中心化）的团队能构建与业务更契合的快速变更周期。

若想更有效地交付变更，单单使用技术还不够。敏捷方法利用迭代开发周期，并定期与企业的业务部门互动，确保项目并未偏离业务目标。如果您的团队利用自动化管道将更细粒度组件交付至自编排容器平台，他们将获得极高的敏捷性。此外，您还可聚焦基于映像的部署、基础架构和代码所提供的一致性和质量，借此进一步改进您的方法。

自助服务开发者体验

可能有人会说，除非您的基础架构能支持团队自主开展工作，并利用中央治理组织尽可能减少联络点，否则团队无法采用上述措施。团队必须能够独立、轻松地设置资源，比如源代码库、构建自动化功能和映像存储库。当然，他们还需要设置容器编排平台，以及相关的计算、内存、存储和网络资源。

打造这类自助服务体验可能很简单，比如您只需要允许团队使用公有托管服务，如 GitHub、DockerHub 和云供应商提供的容器编排功能。 但是，对于很多企业来说，公有功能可能不太适合用于处理一些资源和工作负载。要在内部支持这些功能，企业需要经过一个漫长的过程，为了简化这一流程，很多企业转而采用私有云平台，以便更快速、更一致地构建这些功能。

结语

一个广泛的容器化战略能带来哪些优势？容器化战略在以下领域提供广泛的优势：

- 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。
- 细粒度弹性：单独部署高度可用的组件，以消除单点故障。

- 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与资源密度，从而充分利用基础架构资源。
- 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。
- 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。

第 1 部分更详细地探讨了这些优势。

要想获得更广泛的优势，您必须采用正确的方法，并意识到迁移至容器将为您提供一个拓宽革新的机会。根据容器化计划的具体需求，确定本文探讨的哪些措施能为您所用。

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。