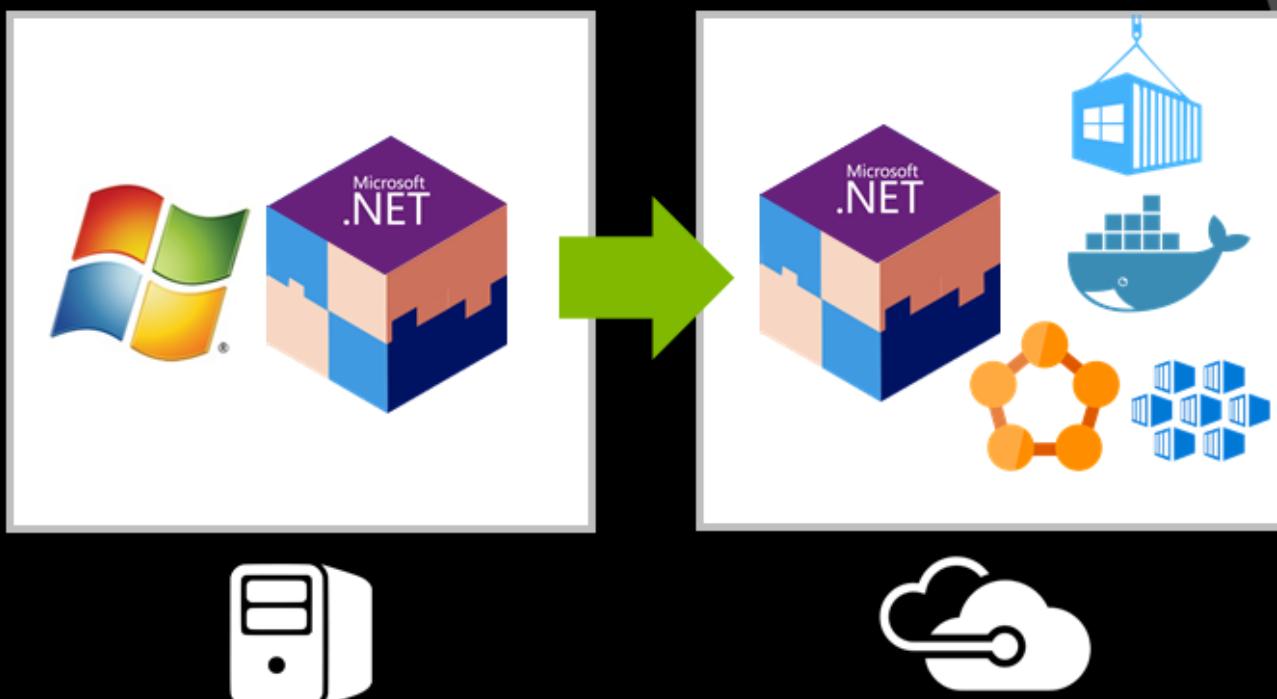


借助Azure云和 Windows容器让现 有.NET应用程序实现 现代化革新



Cesar de la Torre
Microsoft Corp.

借助 Azure 云和 Windows 容器让现有.NET 应用程序实现现代化革新（第一版）

出版商

Microsoft Press 和 Microsoft DevDiv

Microsoft Corporation 下设部门

One Microsoft Way

Redmond, Washington 98052-6399

Copyright ©. 2017 by Microsoft Corporation

保留所有权利。未经出版商的明确书面许可，本书的任何部分不得以任何形式或任何方式转载或传播。

本书以电子图书（电子书）的形式通过微软的多种渠道免费提供，如：

<http://dot.net/architecture>。

如果您就本书有任何问题，请发邮件至：

dotnet-architecture-ebooks-feedback@service.microsoft.com。

本书“依原样”提供，仅代表作者本人的观点和看法。本书所表达的观点意见以及所提供的信息，包括 URL 和引用的其他互联网站点，若有更改恕不另行通知。

本书提供的示例仅限示意和说明，均为虚构。与现实中的任何实体无关，也不应做此推断。

Microsoft 以及 <http://www.microsoft.com> “商标”网页列出的商标均为微软集团公司的商标。所有其他商标皆属于各自所有者的财产。

作者：

Cesar de la Torre, 资深产品经理, .NET 产品组, 微软公司。

合作者与审阅者：

Scott Hunter, 合作伙伴总监产品经理, 微软.NET 团队。

Paul Yuknewicz, 首席产品经理, 微软 Visual Studio Tools 团队。

Lisa Guthrie, 资深产品经理, 微软 Visual Studio Tools 团队。

Ankit Asthana, 首席产品经理, 微软.NET 团队。

Unai Zorrilla, Plain Concepts 开发者主管。

Javier Valero, Grupo Solutio 首席运营官。

目录

第 1 章：简介	7
关于本指南.....	7
现有.NET 应用程序的上云之路.....	7
不同成熟度级别涉及的关键技术和架构	10
平移场景	12
本文未涉及哪些话题	15
本指南的目标读者	16
如何使用本指南.....	17
遗留应用现代化所用的范例应用：eShopModernizing	17
我们希望听到您的反馈！	17
第 2 章：将现有.NET 应用平移至 Azure IaaS（云基础架构就绪）	18
为何将现有.NET Web 应用程序迁移到 Azure IaaS	18
何时迁移至 IaaS 而非 PaaS	19
使用 Azure Migrate 分析现有应用并将其迁入 Azure	19
使用 Azure 站点恢复将现有虚拟机迁移为 Azure 虚拟机.....	20
第 3 章：将关系型数据库迁移至 Azure.....	22
何时迁移至 Azure SQL 数据库托管实例	22
何时迁移至 Azure SQL 数据库.....	23
何时将原先的 RDBMS 迁移至虚拟机（IaaS）	24
何时将 SQL Server 迁移为虚拟机（IaaS）	24
使用 Azure Database Migration Service 将关系型数据库迁入 Azure	24
第 4 章：将现有.NET 应用平移为云 DevOps 就绪的应用程序	26
将现有.NET 应用平移为云 DevOps 就绪应用程序的理由	26
云 DevOps 就绪应用程序的基本原则和宗旨	27

云 DevOps 就绪应用程序可提供的收益	28
云 DevOps 就绪应用程序中的微软技术.....	29
单体式应用程序也可以云 DevOps 就绪.....	30
云优化应用程序又该怎么办？	30
云原生和云优化应用程序.....	31
微服务是怎么回事？	32
何时使用 Azure 应用服务对现有.NET 应用实现现代化	33
如何将现有.NET 应用部署至 Azure 应用服务	35
使用 Azure App Service Migration Assistant 评估网站并迁移至应用服务.....	35
将现有.NET 应用部署为 Windows 容器	36
容器是什么？ (Linux 或 Windows)	36
容器的收益 (Windows 或 Linux 上使用的 Docker Engine)	37
Docker 是什么？	38
通过 Windows 容器运行现有.NET 应用程序的收益	39
为.NET 容器选择目标操作系统.....	39
Windows 容器的类型.....	40
何时不应部署到 Windows 容器	41
何时将 Windows 容器部署到本地 IaaS 虚拟机基础架构.....	42
何时将 Windows 容器部署至 Azure 虚拟机 (IaaS 云)	42
何时将 Windows 容器部署至 Service Fabric	42
何时将 Windows 容器部署至 Azure Container Service (例如 Kubernetes)	43
构建云就绪的弹性服务：适应云端的暂时故障.....	44
应对局部故障.....	44
借助监视和遥测实现应用的现代化.....	45
使用 Application Insights 监视应用程序	46
使用 Log Analytics 及容器监视解决方案监视 Docker 基础架构	46
使用 CI/CD 流程和云端 DevOps 工具实现应用生命周期的现代化.....	48

迁移至混合云场景	49
Azure Stack	49
第 5 章：步骤和技术准备概述	52
技术步骤清单	52
场景 1：eShop 遗留应用概览	52
可用的技术步骤	52
概述	52
目标	53
场景	53
收益	54
下一步	54
场景 2：使用 Windows 容器实现现有.NET 应用程序的容器化	54
可用的技术步骤	54
概述	54
目标	54
场景	55
收益	55
下一步	55
场景 3：将基于 Windows 容器的应用部署至 Azure 虚拟机	56
可用的技术步骤	56
概述	56
目标	56
场景	56
收益	58
下一步	58
场景 4：将基于 Windows 容器的应用部署至 Azure Container Service 中的 Kubernetes 集群	58

可用的技术步骤	58
概述	58
目标	58
场景	59
收益	59
下一步	60
场景 5：将基于 Windows 容器的应用部署至 Azure Service Fabric	60
可用的技术步骤	60
概述	60
目标	61
场景	61
收益	62
下一步	63
第 6 章：结论	64
主要结论	64

第 1 章：简介

当你打算对 Web 应用程序进行革新，并将其迁入云端运行，此时未必需要彻底重构现有应用。由于成本和时间等条件的约束，使用诸如微服务等先进方式重构应用并非总是可行。取决于具体类型，应用可能并不需要重构。为了优化整个组织上云迁移策略的成本效益，必须首先考虑应用所承担的业务要求和技术需求。此时需要决定：

- 哪些应用需要转换或重构。
- 哪些应用只需要对部分组件实现现代化。
- 哪些应用可以直接“平移（Lift and shift）”上云。

关于本指南

本指南主要侧重于“平移”场景，最初主要考虑对基于 Microsoft .NET Framework 的现有 Web 或面向服务的应用程序进行现代化。平移是指在不改变应用程序代码和基础架构的前提下，将工作负载转移到更新，或更现代化的环境中运行的做法。

本指南将介绍如何将基于.NET Framework 的现有服务器端应用程序的不同方面进行现代化改造，随后直接迁入云端运行，但并不重构整个应用程序或更改其代码。

本指南还将重点介绍使用 Windows 容器和 Azure 中的编排引擎（Orchestrator）等新技术和方法，将应用迁入云端或对应用程序的部分组件进行现代化改造等做法所能获得的收益。

现有.NET 应用程序的上云之路

组织选择上云，通常是为了让自己的应用程序更加敏捷快速。毕竟用户只需要几分钟时间即可在云中创建数千台服务器（虚拟机），而类似的本地服务器部署往往需要数周时间。

但不同应用程序的上云迁移并非只有这一种“均码”的做法。怎样的迁移策略最适合，这主要取决于组织的需求和工作优先级，以及要迁移的应用程序所属的类型。并非所有应用程序都值得斥资迁移至平台即服务（PaaS）模式，或专门开发云原生模式应用程序。很多情况下，可以采取阶段式或循序渐进式的方法，根据业务需求逐渐将应用迁入云端。

只有可以在长期范围内为组织带来敏捷性和价值的现代化应用程序，才可以从云优化和云原生应用程序架构的投资中获益。然而对于现有的或遗留的应用程序，重点在于如何只付出最少量的时间和金钱（不重构或更改代码）将其迁入云端，并获得巨大的收益。

图 1-1 展示了在以循序渐进的方式将现有.NET 应用程序迁入云端时可行的路径。

现有 .NET 应用程序实现现代化的方法

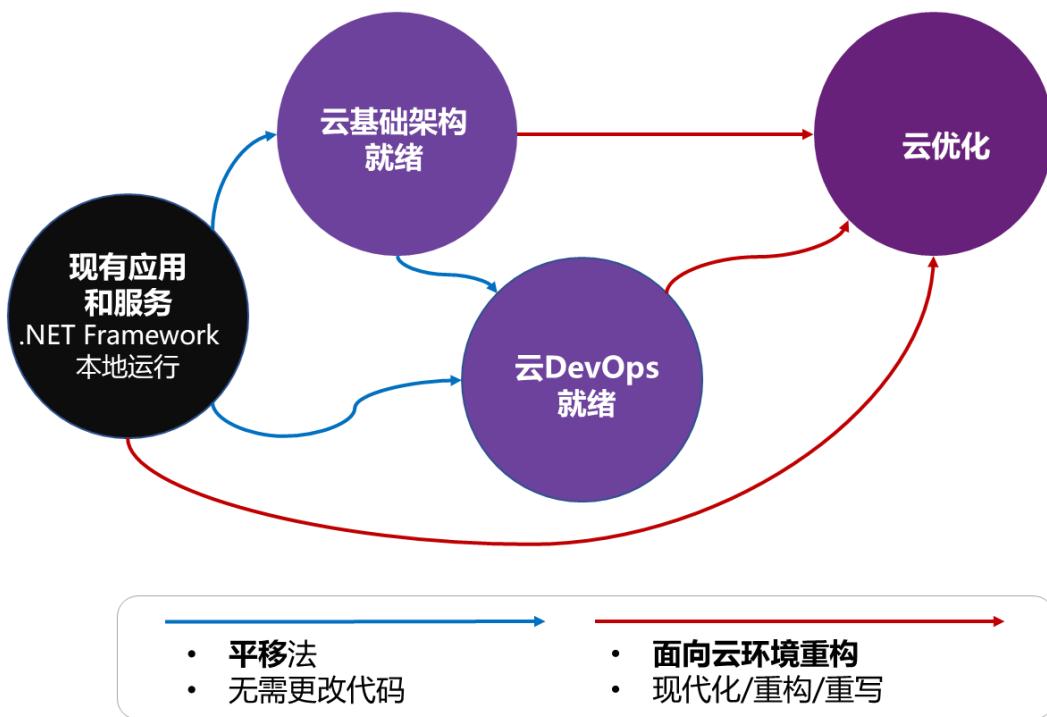


图 1-1：现有 .NET 应用程序和服务实现现代化的路径

每种迁移方式可提供不同收益，适合不同场景。在将应用迁往云端时，可选择使用一种方式，或为不同组件选择不同方式。每个应用程序并非只能使用一种方式，各自的成熟度也各不相同。例如，有一种比较常用的混合方法会将某些组件放置在本地，同时将一些组件放在云端。

在前两种迁移程度下，只需将应用程序平移到云端：

第 1 级：云基础架构就绪：在这种迁移方式中，只需简单地将目前在本地运行的应用程序重新托管或迁移至基础架构即服务 ([IaaS](#)) 平台，应用的组成组件与迁移前几乎完全相同，但迁移后需要部署在云端虚拟机中。

第 2 级：云 DevOps 就绪：在这种迁移方式中，首先需要进行平移，此时依然无需重构或修改代码，这样便可通过云端运行获得更多收益。随后可通过重塑企业开发运维 (DevOps) 流程，以更快的速度发布应用程序，借此改善敏捷性。为此可使用诸如 Windows 容器等技术，这是一种基于 Docker Engine 的技术。在多阶段部署过程中，容器可以避免因为应用程序依赖性造成的阻力。同时容器技术还可以充分运用与数据、监视、持续集成/持续部署 (CI/CD) 流程有关的其他云端托管服务。

第三级成熟度是上云的最终目标，但对很多应用来说非必需，同时也不是本文的重点：

第 3 级：云优化：这种迁移方法通常受到业务需求的推动，目标在于关键业务应用程序的现代化。在这个级别中，我们可以使用 PaaS 服务将应用迁移至 PaaS 计算平台。我们可以实施[云原生](#)应用程序和微服务架构，借此实现应用程序的革新，并在长期范围内获得敏捷性，以及更进一步的伸缩能力。此类现代化措施通常需要专为云环境设计的架构，并且通常需要编写新的代码，在实现云原生应用和基于微服务的模型时更是如此。这种方法可以帮助我们获得传统单体式（Monolithic）应用和本地应用程序环境中无法实现的优势。

表 1-1 列出了每种迁移或现代化方式的主要收益和选择每种方式的原因。

云基础架构就绪	云 DevOps 就绪	云优化
平移		现代化/重构/重写
应用程序计算工作所处位置		
应用程序部署在 Azure 虚拟机中	容器化的单体式应用程序或 N 层应用，部署在虚拟机、Azure Service Fabric，或 Azure Container Service (如 Kubernetes) 中	容器化的微服务或常规 PaaS 应用程序，位于 Azure 应用服务、Azure Service Fabric、Azure Container Service (如 Kubernetes) 中
数据所处位置		
虚拟机中的 SQL 或任何关系型数据库中	Azure SQL 数据库托管实例	Azure SQL 数据库、Azure Cosmos DB 或其他 NoSQL
优势		
<ul style="list-style-type: none">• 无需重构或编写新代码• 付出最少精力即可快速迁移• Azure 可支持的最基本用法• 可保证基本的可用性• 上云后可以很容易实现进一步的现代化	<ul style="list-style-type: none">• 无需重构或编写新代码• 相比虚拟机，容器可支持更细化的增量式迁移• 通过容器改善版本发布时的部署和 DevOps 敏捷性• 提高密度降低部署成本• 应用和依赖项可获得移植能力	<ul style="list-style-type: none">• 云专用架构，需要重构并编写新代码• 基于微服务的云原生方法• 新的 Web 应用、单体式应用、N 层应用、获得云弹性和云优化• 全面托管的服务• 补丁安装自动化

	<ul style="list-style-type: none"> 借助 Azure 容器服务（如 Kubernetes）和 Azure Service Fabric 实现高可用性和编排引擎能力 可为 Service Fabric 的节点/虚拟机打补丁 灵活选择托管位置：Azure 虚拟机或虚拟机规模集、Azure 容器服务（如 Kubernetes）、Service Fabric，以及以后新发布的容器技术 	<ul style="list-style-type: none"> 针对伸缩性优化 通过子系统提高应用的敏捷性 基于 DevOps 进行构建 增强的 DevOps，如插槽和部署策略 PaaS 和编排引擎的运行位置：Azure 应用服务、Azure 容器服务、Azure Service Fabric，以及以后新发布的容器化 PaaS
挑战		
<ul style="list-style-type: none"> 除了成本转变为运营开销或关闭本地数据中心，可获得的其他云价值较小 托管式组件很少：需要手工为操作系统或中间件打补丁，通常还需要依赖不可变的基础设施解决方案，如 Terraform、Spinnaker 或 Puppet 	<ul style="list-style-type: none"> 容器化的实现需要额外的学习 	<ul style="list-style-type: none"> 可能需要大幅重构或重写代码（需要更多时间和预算）

表 1-1：现有.NET 应用程序和服务实现现代化的过程所面临的收益和挑战

不同成熟度级别涉及的关键技术和架构

.NET Framework 应用程序最初始于 2011 年底发布的 1.0 版.NET Framework，随后很多公司开始陆续升级为后续的新版本（如 2.0、3.5 和.NET 4.x）。大部分此类应用程序运行在 Windows Server 和 Internet Information Server (IIS) 基础上，并会使用关系型数据库，例如 SQL Server、Oracle、MySQL 或其他 RDBMS。

目前的大部分现有.NET 应用程序可能基于.NET Framework 4.x 甚至.NET Framework 3.5，并会使用诸如 ASP.NET MVC、ASP.NET Web Forms、ASP.NET Web API、Windows Communication Foundation (WCF)、ASP.NET SignalR 以及 ASP.NET Web Pages 等 Web 框架。这些成熟的.NET Framework 技术都基于 Windows。如果希望直接迁移遗留应用，并且

希望只对应用程序的基础架构进行最少量的改动，就必须充分考虑到这种依赖性的存在。

图 1-2 展示了三种云成熟度级别中主要会用到的技术和架构风格：

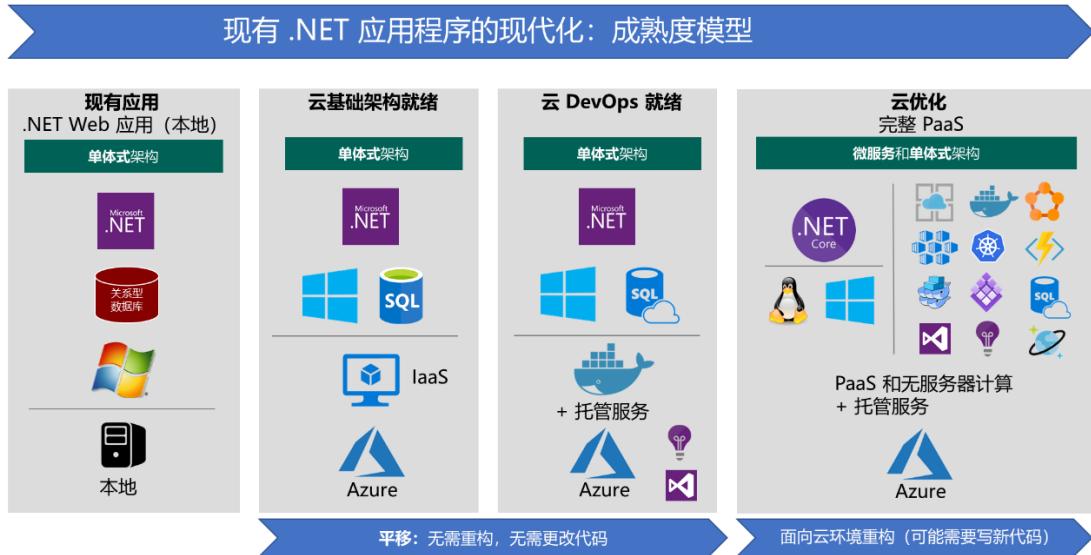


图 1-2：现有.NET Web 应用程序现代化过程中，每种成熟度级别使用的主要技术

图 1-2 展示了最常见的场景，但在具体架构方面，也有很多基于这些场景演变的混合 (Hybrid) 与杂合 (Mixed) 场景。例如该成熟度模型不仅适用于单体式架构的现有 Web 应用，同样适用于面向服务的、N 层，以及其他架构风格的变体。

现代化过程中的每个成熟度级别都可对应下列重要技术与方法：

- **云基础架构就绪**（重新托管或基本的平移）：作为第一个步骤，很多组织只希望快速执行云迁移策略。此时可直接重新托管应用程序。大部分重新托管工作可通过 [Azure Migrate](#) 自动进行，这种服务提供了相关工具，帮助用户借助诸如 [Azure 站点恢复](#) 和 [Azure 数据库迁移服务](#) 等云工具，迁移至 Azure 所需的指南、见解和机制。用户也可以手工执行重新托管的操作，借此在将遗留应用迁移到云端的过程中，更进一步了解有关基础架构的各种细节。例如，用户只需少量改动即可将应用程序迁移至 Azure 虚拟机中运行，这一过程可能只需要对配置进行少量改动。此时所用的网络环境与本地环境非常类似，尤其是在 Azure 中创建虚拟网络的话更是如此。
- **云 DevOps 就绪**（更完善的平移）：该模式需要进行少量重要的部署优化，借此从云中获得一些重要收益，但依然不需要更改应用程序的核心架构。此时最重要的步骤在于为现有 .Net Framework 应用程序增加对 [Windows 容器](#) 的支持，这个操作（容器化）并不需要修改代码，因此整个平移过程也很快捷。例如可以使用 [Image2Docker](#) 或 Visual Studio 等工具，以及 [Docker](#) 的相关工具。Visual Studio 可以智能地自动选择 ASP.NET 应用程序的默认值和 Windows 容器镜像。这些工具还可提供快速的内循环 (Inner loop)，并快速将容器部署到 Azure。如果要部署到多个环境，此时还可以更敏捷。随后即可在生产环境中将 Windows 容器部署到诸如 [Azure Service Fabric](#) 或 [Azure 容器服务](#) (Kubernetes、DC/OS 或 Swarm) 等编排引擎中。在现代化过程的初期，还可从云中

添加资产，例如使用 [Azure Application Insights](#) 等工具进行监视，或通过 [Visual Studio Team Services](#) 为应用的生命周期创建 CI/CD 流程，甚至可以添加 Azure 可支持的各种数据资源服务。例如，我们可以直接修改最初使用传统 [ASP.NET Web Forms](#) 或 [ASP.NET MVC](#) 开发的单体式 Web 应用，随后使用 Windows 容器部署。在使用 Windows 容器后，还可将数据迁移至 [Azure SQL 数据库托管实例](#) 数据库，这一切工作都不需要更改应用程序的核心架构。

- **云优化：**如上文所述，应用程序的上云现代化，最终目标就在于将整个系统托管到诸如 [Azure 应用服务](#) 等 PaaS 平台。PaaS 平台侧重于现代化的 Web 应用程序，并可通过诸如[无服务器计算](#)（如 [Azure Functions](#)）等平台进一步扩展应用。这种成熟度级别下，另一种更先进的场景是微服务架构和[云原生](#)应用程序，这种场景通常会使用诸如 [Azure Service Fabric](#) 或 [Azure 容器服务](#)（Kubernetes、DC/OS 或 Swarm）等编排引擎。这些编排引擎专为微服务和多容器应用程序打造。所有这些方法（例如微服务和 PaaS）通常都需要编写新代码，这些代码需要明确适配具体的 PaaS 平台，或者可以针对诸如微服务等特定的架构来编写。

图 1-3 展示了每种成熟度级别可使用的内部技术：

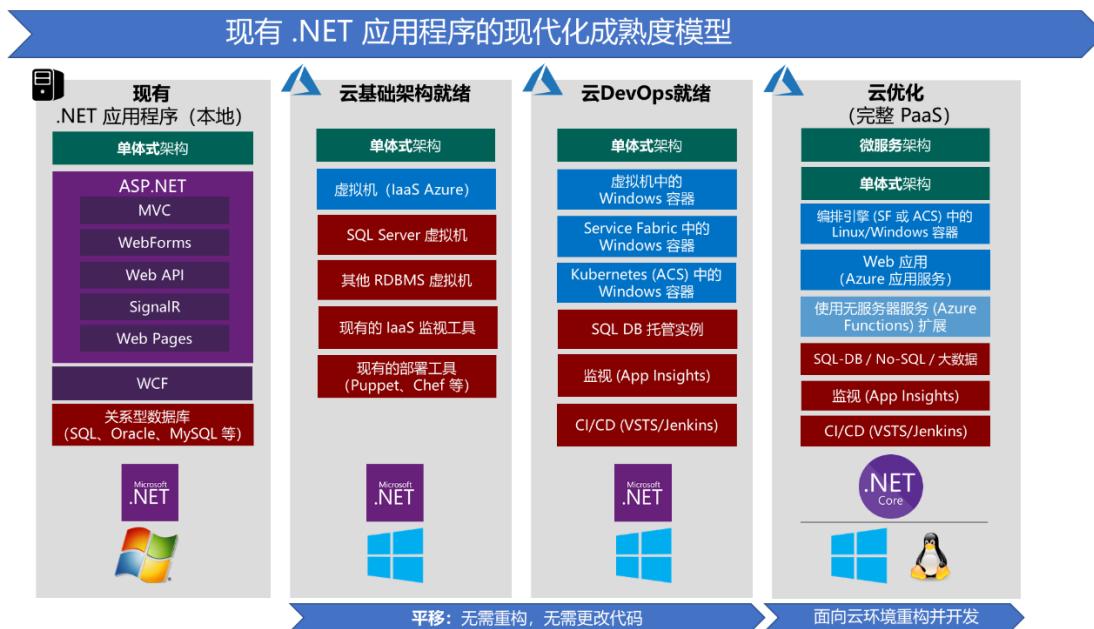


图 1-3：现代化过程中每种成熟度级别所用的内部技术

平移场景

对于平移迁移需要注意的是，具体应用程序场景中可以使用基于平移演变出的各种变体。如果只是重新托管应用程序，那么可能会面对类似图 1-4 所示的场景，只将云中的虚拟机用于应用程序和数据库服务器。

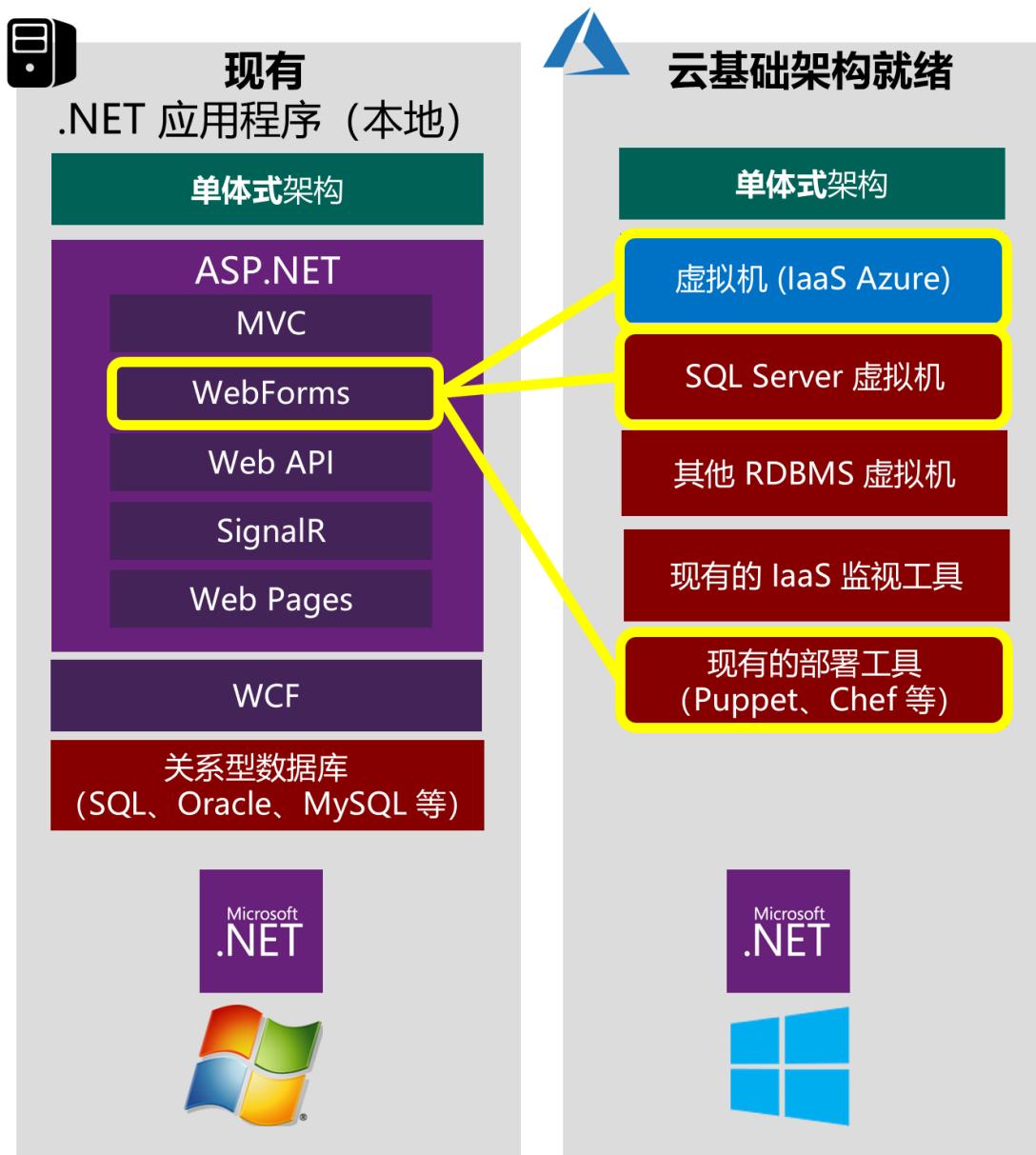


图 1-4：云中的 IaaS 场景范例

随后还可以只使用该成熟度级别的组件搭建纯粹的云 DevOps 就绪应用程序，或者可能遇到处于中间状态的应用程序，其中一些组件来自云基础架构就绪级别，另一些组件来自云 DevOps 就绪级别（“挑选并选择”或杂合模式），例如图 1-5 所示。

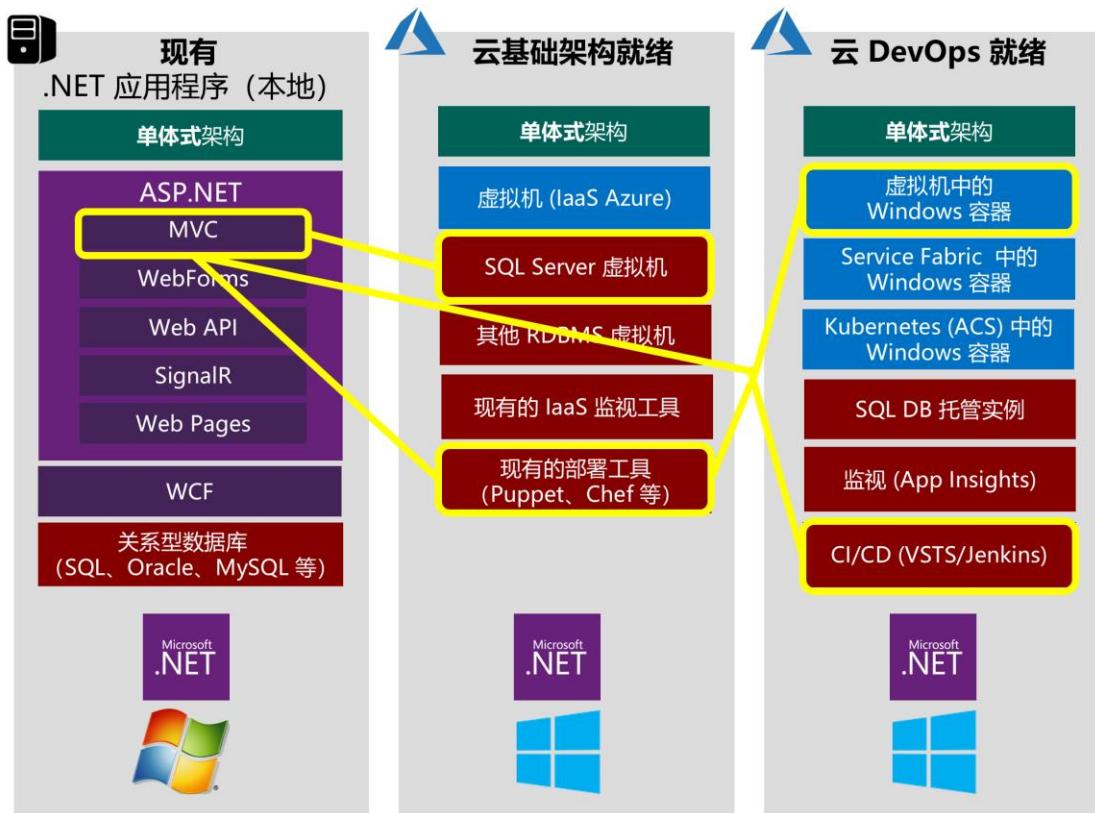


图 1-5：“挑选并选择” 场景范例，使用了 IaaS 数据库，以及 DevOps 和容器化的组件

接下来，作为现有.NET Framework 应用程序最理想的迁移场景，可以将其迁移为云 DevOps 就绪应用程序，借此只需少量工作即可获得巨大收益。这种方法还可以帮助我们面对未来的云优化做好充分准备。图 1-6 展示了一个范例。

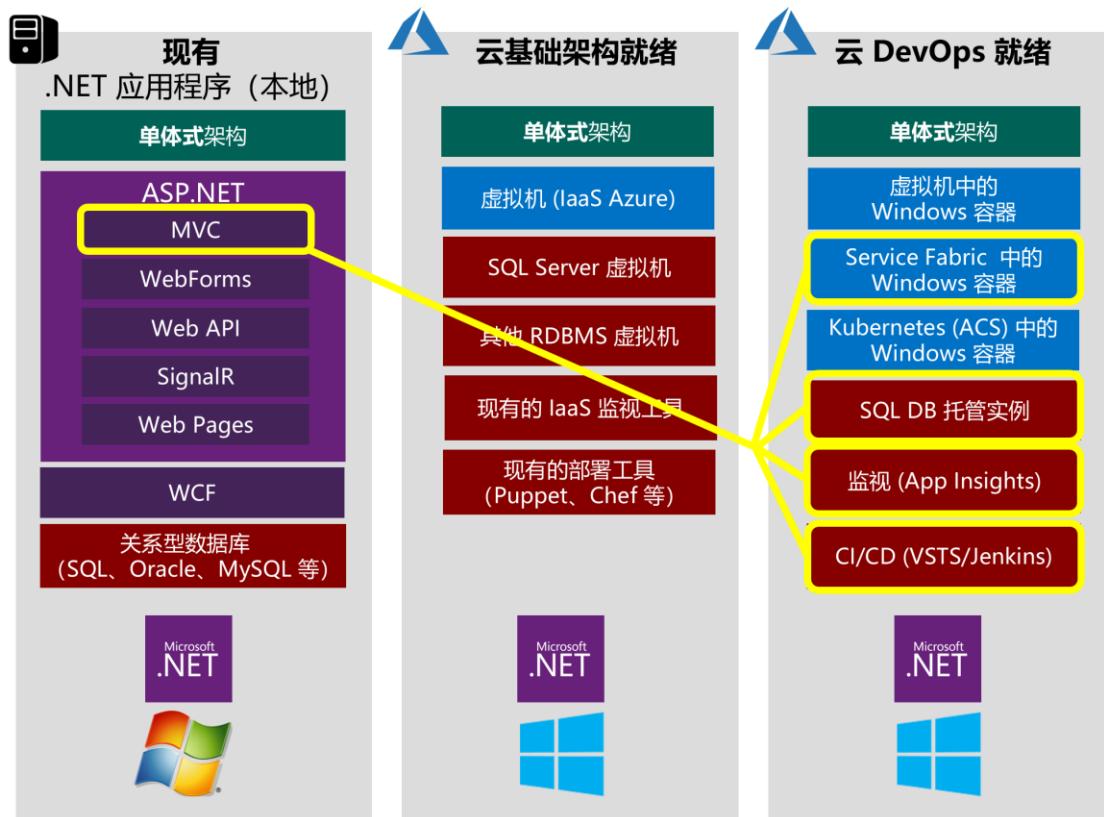


图 1-6：云 DevOps 就绪应用场景范例，其中使用了 Windows 容器和托管服务

如果要进一步完善，还可将现有的云 DevOps 就绪应用程序进一步扩展，针对具体场景添加少量微服务。这样便可在一定程度上实现云优化模式中的云原生场景，但这不是本文的重点。

本文未涉及哪些话题

本指南涵盖了上述范例场景中的几个特定子集，如图 1-7 所示。本指南主要专注于平移场景以及最终所要实现的云 DevOps 就绪模型。在云 DevOps 就绪模型中，可使用 Windows 容器外加诸如监视和 CI/CD 流程等额外组件实现 .NET Framework 应用程序的现代化。每个组件都是用快速敏捷的方式将应用程序部署到云端所必不可少的。

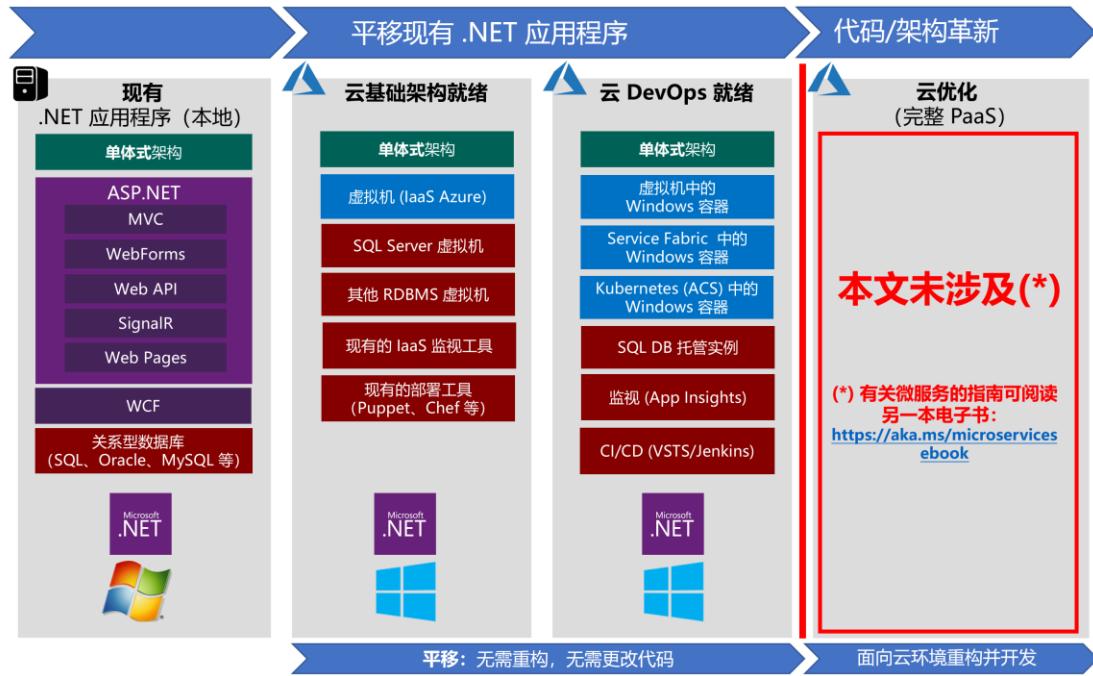


图 1-7：通过平移和初步的现代化打造云 DevOps 就绪应用程序

本指南的重点非常具体。下文会介绍在无需重构，无需改动代码的情况下，对现有.NET 应用程序进行平移的不同方法。最后，下文还将介绍如何让应用程序面对云 DevOps 做好准备。

本指南不涉及打造云原生应用程序的方法，例如：如何演变为微服务架构。如果要重构应用，或基于微服务开发全新应用，可参阅电子书：[.NET 微服务：容器化.NET 应用架构指南](#)。

参考资源

- 使用微软平台和工具实现 Docker 应用程序生命周期的容器化（可下载电子书）：
<https://aka.ms/dockerlifecyclebook>
- .NET 微服务：容器化.NET 应用程序的架构（可下载电子书）：
<https://aka.ms/microservicesebook>
- 使用 ASP.NET Core 和 Azure 构建现代化 Web 应用程序的架构（可下载电子书）：
<https://aka.ms/webappebook>

本指南的目标读者

本指南适合于那些希望对基于.NET Framework 的现有 ASP.NET 应用程序进行现代化，借此改善应用程序交付与发布过程敏捷性的开发者和解决方案架构师。

如果您是希望概括了解使用 Windows 容器，以及使用 Microsoft Azure 将应用部署到云端所获收益的技术决策者，例如企业架构师或开发主管/总监，也将从本指南中获益。

如何使用本指南

本指南回答了“为什么”：为什么要将现有应用程序现代化，以及使用 Windows 容器将应用迁入云端所能获得的具体收益。本指南前几章的内容主要面向希望概括了解相关内容，但无需了解具体实现方式和技术细节的架构师和技术决策者。

本指南的最后一章通过多种方法介绍了具体的部署场景。在本指南中，我们仅提供了各种方法的简要介绍，借此总结每个场景并强调各自的收益。深入到安装和实现细节的完整方法已作为一系列[维基页面](#)发布到公开的 [GitHub 代码库](#)，此外这里还提供了相关的范例应用（下一节将详细介绍）。最后一章内容，以及 GitHub 上的循序渐进操作步骤维基页面更适合希望了解具体实现细节的开发者和架构师。

遗留应用现代化所用的范例应用：eShopModernizing

GitHub 上的 [eShopModernizing](#) 代码库提供的两个范例应用模拟了遗留的单体式 Web 应用程序。其中一个 Web 应用使用 ASP.NET MVC 开发，另一个使用 ASP.NET Web Forms 开发。这两个 Web 应用都基于传统.NET Framework。这些范例应用并未使用.NET Core 或 ASP.NET Core，因为它们都是需要进行现代化的现有/遗留.NET Framework 应用程序。

这两个范例应用都有另外一个版本，其中包含了现代化后的代码，内容也非常简单直接。两个版本应用最重要的差异在于，第二版可使用 Windows 容器部署。此外第二版还增加了少量内容，例如使用 Azure Storage Blob 管理镜像，使用 Azure Active Directory 实现安全性，并使用 Azure Application Insights 实现应用程序的监视和审核。

我们希望听到您的反馈！

撰写本指南是为了帮您理解和完善现代化现有的.NET Web 应用程序时可用的选项。本指南和相关范例应用程序依然还在不断改善。欢迎您反馈自己的意见！如果您对本指南的完善有自己的建议和意见，请发送至：

dotnet-architecture-ebooks-feedback@service.microsoft.com。

第 2 章: 将现有.NET 应用平移至 Azure IaaS (云基础架构就绪)

愿景：作为第一步，只需将现有应用程序重新托管到云端，便可降低本地投入以及硬件和网络维护的总成本。

在介绍如何将现有应用程序迁移至 Azure 基础架构即服务 (IaaS) 平台之前，首先需要分析将应用直接迁移至 Azure IaaS 的原因。对于这种成熟度的现代化场景，基本上可以直接使用云端的虚拟机，借此取代目前使用的本地基础架构。

此外还要分析为何要迁移至纯粹的 IaaS 云平台，而非使用 Azure 中其他更先进的托管服务。因此需要确定是由于哪些情况导致必须在一开始使用 IaaS。

图 2-1 展示了云基础架构就绪应用程序在现代化成熟度级别中的定位：

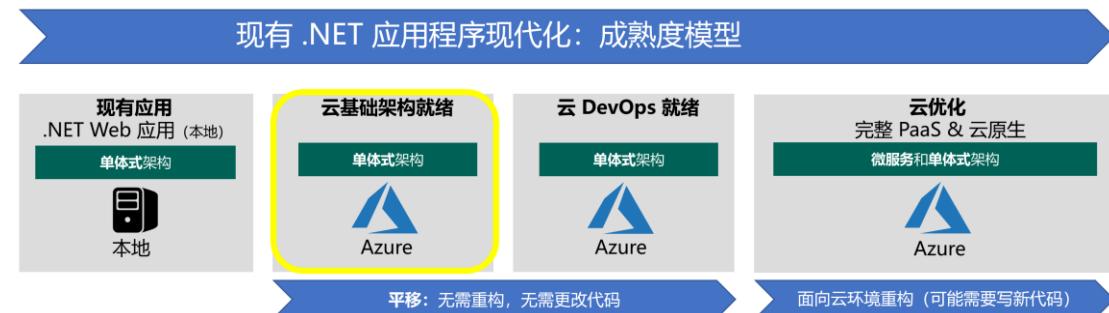


图 2-1：云基础架构就绪应用程序的定位

为何将现有.NET Web 应用程序迁移到 Azure IaaS

最初以 IaaS 的形式迁入云端的主要原因在于降低成本。通过使用托管的基础架构服务，组织可以降低在硬件维护、服务器、虚拟机配置与部署，以及基础架构管理等方面的投入。

在决定将应用迁入云端后，选择 IaaS 而非 PaaS 等更先进的服务，主要原因可能在于对 IaaS 环境更熟悉。迁移到与当前本地环境更相似的环境，可以降低学习曲线，让上云过程更顺利。

然而这样“走捷径”上云并不意味着可以立刻获得通过云环境运行应用程序所能实现的全部收益。任何组织的云迁移只要能达到云 DevOps 就绪和 PaaS (云优化) 的成熟度级别，均能进一步获得更大的收益。

很显然，当应用程序已经在云中运行时（哪怕只是 IaaS），对于后续的进一步现代化甚至重构工作都会变得更容易。部分原因是由于应用程序数据的迁移已经完成。此外组织也可借此获得使用云环境所需的技能，让未来的“云中运行”变得更容易实现。

何时迁移至 IaaS 而非 PaaS

下一节将探讨主要基于 PaaS 平台和服务的云 DevOps 就绪应用程序。这些应用可以提供上云所能获得的大部分收益。

如果您的目标仅仅在于将现有应用程序迁入云端，那么一开始可以找出需要进行大量改动才能通过 Azure 应用服务运行的现有应用程序，这些应用将是首选的迁移目标。

随后，如果不希望或者依然无法改为使用 Windows 容器或通过 Azure Service Fabric 以及 Kubernetes 等编排引擎运行，那么可以考虑直接使用虚拟机（IaaS）。

然而要注意，相比使用 Azure 提供的 PaaS 服务，虚拟机的妥善配置、保护和维护需要投入更多时间和 IT 技能。如果考虑使用 Azure 虚拟机，请确保考虑了后期持续不断的维护工作量，例如需要打补丁、更新、管理虚拟机环境。毕竟 Azure 虚拟机是一种 IaaS 服务。

使用 Azure Migrate 分析现有应用并将其迁入 Azure

上云过程并不难。但很多组织最初的脚步都迈得很艰难，往往难以深入了解整个环境，以及应用程序、工作负载、数据等内容之间的依赖性。缺乏这样的了解就很难为未来的工作做好规划。如果无法细致了解成功迁移所要满足的条件，整个组织范围内的迁移工作都将无法顺利实施。可能会无法全面了解潜在的成本收益，无从判断工作负载是否可以平移上云还是需要大量改动才能成功迁移。难怪很多组织对上云依然抱有抗拒的态度。

[Azure Migrate](#) 是一种全新的服务，可以提供必要的指导、见解和机制，帮助用户轻松迁移 到 Azure 环境。Azure Migrate 可提供：

- 发现并评估本地虚拟机
- 内建发现多层应用程序依赖性映射的功能
- 智能地确定所需 Azure 虚拟机的规模
- 通过兼容性报表和指南为潜在问题提出补救意见
- 与 Azure Database Management Service 集成实现数据库发现和迁移

Azure Migrate 可以帮助用户自信地找出适合迁移的工作负载，并在确保对业务产生的影响最小的前提下顺利迁移，转为在 Azure 中运行。在恰当的工具和指南帮助下，用户可在获得最大化投资回报的同时满足对性能与可靠性的严格要求。

图 2-2 展示了通过 Azure Migrate 对所有服务器和应用程序相互之间的连接情况创建的依赖性映射。

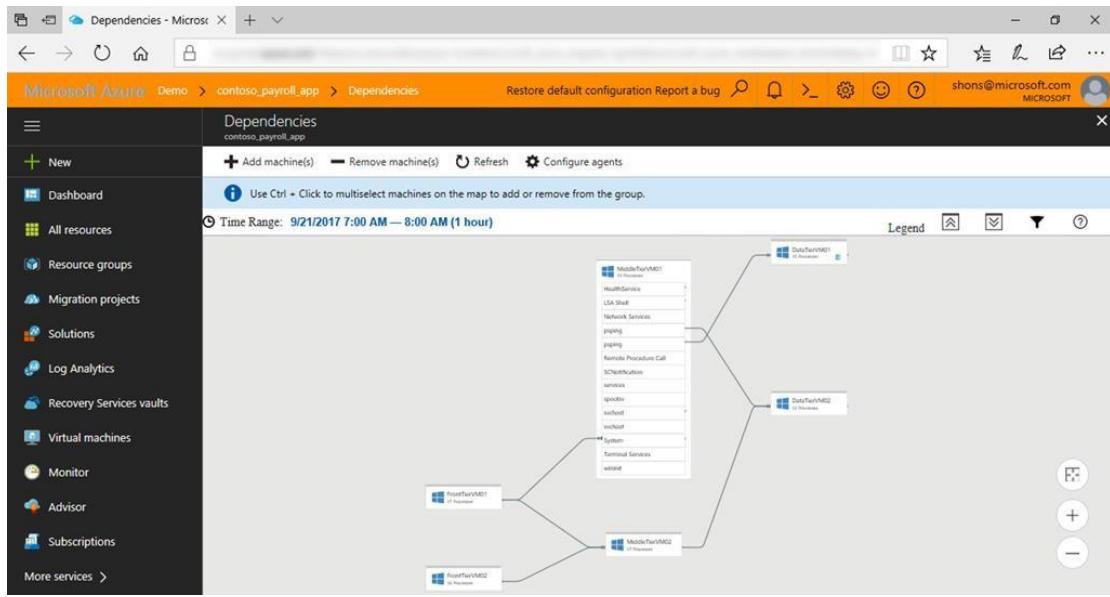


图 2-2：Azure Migrate 创建的依赖性映射图

使用 Azure 站点恢复将现有虚拟机迁移到 Azure 虚拟机

作为端到端 [Azure Migrate](#) 服务的一部分，用户可以使用 [Azure 站点恢复工具](#)轻松地将 Web 应用程序迁移到 Azure 虚拟机中。用户可以使用站点恢复将本地虚拟机和物理服务器复制到 Azure，或将其复制到另一个本地位置。用户甚至可以复制受支持的 Azure 虚拟机、本地 Hyper-V 虚拟机、VMware 虚拟机、以及 Windows 或 Linux 物理服务器中运行的工作负载。复制到 Azure 有助于消除管理第二个数据中心所造成的技术债务。

站点恢复还有助于创建横跨本地和 Azure 的混合环境。通过在站点故障后确保虚拟机和本地物理服务器中运行的应用不受影响，站点恢复有助于保障业务连续性。该技术可复制虚拟机和物理服务器中运行的工作负载，并在主要站点故障后让工作负载可以在辅助位置继续运行。当主站点恢复运行后，还可将工作负载重新转入主站点。

图 2-3 展示了使用 Azure 站点恢复执行的多虚拟机迁移过程。

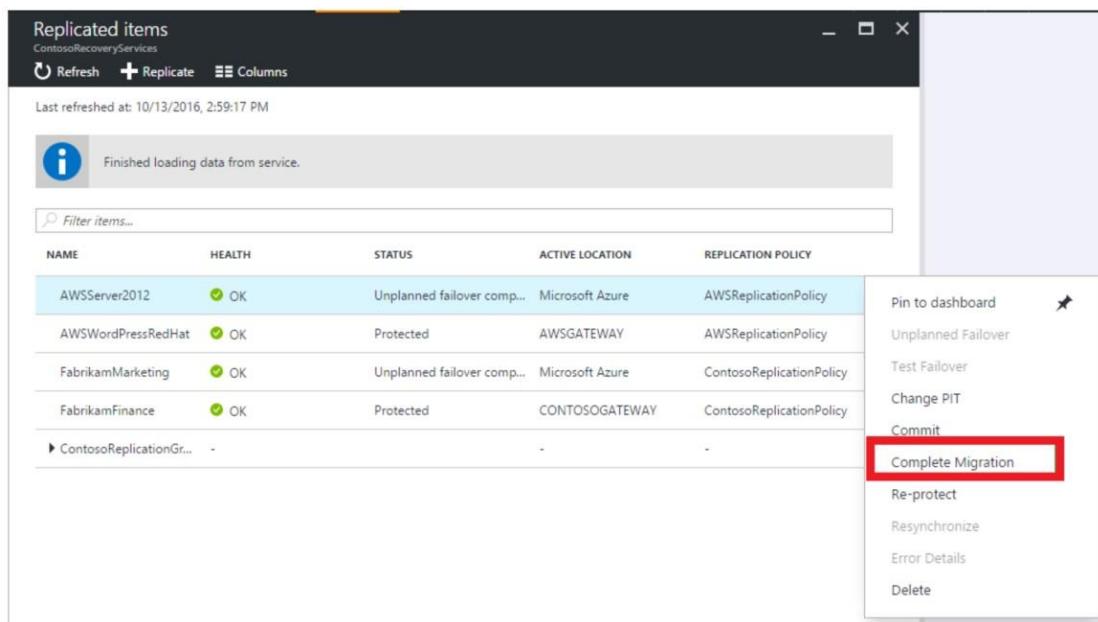


图 2-3：使用 Azure 站点恢复迁移多个虚拟机

参考资源

- **Azure Migrate 产品单页**
https://aka.ms/azurermigration_datasheet
- **Azure Migrate**
<http://azurermigrationcenter.com/>
- **使用站点恢复迁移至 Azure**
<https://docs.microsoft.com/en-us/azure/site-recovery/site-recovery-migrate-to-azure>
- **Azure 站点恢复服务概述**
<https://docs.microsoft.com/en-us/azure/site-recovery/site-recovery-overview>
- **将 AWS 虚拟机迁移为 Azure 虚拟机**
<https://docs.microsoft.com/en-us/azure/site-recovery/site-recovery-migrate-aws-to-azure>

第 3 章：将关系型数据库迁移至 Azure

愿景：Azure 提供了最全面的数据库迁移服务。

通过使用 Azure，用户可将数据库服务器直接迁移至 IaaS 虚拟机（纯粹的平移），或迁移至 Azure SQL 数据库以获得更多收益。Azure SQL 数据库提供了托管实例和完整的数据库即服务（DBaaS）选项。图 3-1 展示了 Azure 可支持的多种关系型数据库迁移路径。

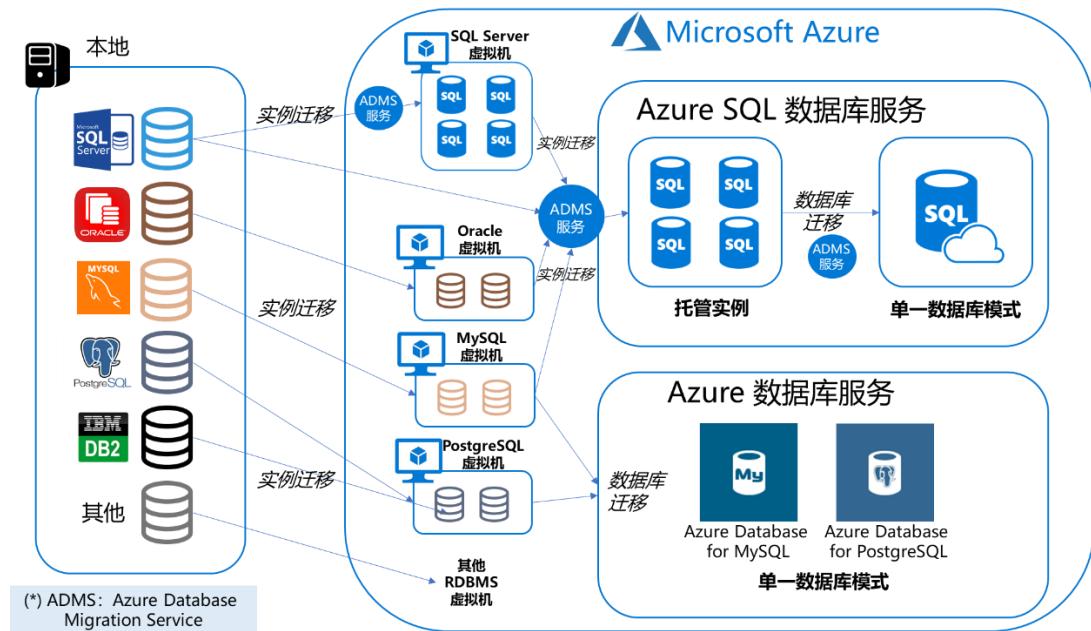


图 3-1：Azure 支持的数据库迁移路径

何时迁移至 Azure SQL 数据库托管实例

大部分情况下，如果要将数据迁入 Azure，Azure SQL 数据库托管实例将会是用户的最佳选择。如果用户自行管理了 SQL Server 数据库，需要 100% 确保无需重构应用程序，无需更改数据，或无需更改数据访问代码，即可选择 Azure SQL 数据库托管实例。

除了 Azure SQL 数据库提供的标准功能（单一数据库模型），如果针对 SQL Server 实例级别的功能或隔离程度有额外需求，Azure SQL 数据库托管实例也将是最适合的选项。Azure SQL 数据库标准服务是一种 PaaS 服务，但无法提供与传统 SQL Server 完全一致的功能，并可能导致迁移工作无法顺利进行。

例如，如果某家组织在实例级别的 SQL Server 功能方面进行了大量深入投入，即可迁移至 SQL 托管实例并从中获益。实例级别 SQL Server 功能有很多，例如 SQL 通用语言运行时（CLR）集成、SQL Server Agent，以及跨数据库查询。Azure SQL 数据库标准服务（单一数据库模型）不支持这些功能。

处于高度管控行业，以及出于安全目的需要实现隔离的组织，也能从 SQL 托管实例中获益。

Azure SQL 数据库托管实例具备下列特征：

- 通过 Azure 虚拟网络实现安全隔离
- 可通过下列功能获得进一步的应用程序兼容性：
 - SQL Server Agent 和 SQL Server Profiler
 - 跨数据库引用和查询、SQL CLR、复制、更改数据捕获 (CDC)，以及 Service Broker
- 数据库规模最大可达 35 TB
- 通过下列功能将迁移过程中的停机时间降至最低：
 - Azure Database Migration Service
 - 原生备份和还原，以及日志发送

在这些功能的帮助下，当用户将现有应用程序数据库迁移至 Azure SQL 数据库时，托管实例模式可提供近乎完全等同于 PaaS 模式 SQL Server 的全部功能。托管实例是一种 SQL Server 环境，可在无需更改应用程序设计的前提下继续使用实例级别的功能。

对于目前正在使用 SQL Server，需要在云中继续获得灵活的网络安全性的组织，托管实例可能是最适合的选项。这种方式类似于为用户的 SQL 数据库建立了一个私有的虚拟网络。

何时迁移至 Azure SQL 数据库

如上文所述，标准的 Azure SQL 数据库是一种托管式关系型 DBaaS。SQL 数据库目前已经跨越全球 38 个数据中心管理着数百万个生产数据库，为不同类型的应用程序和工作负载提供着支持，从普通事务型数据的管理，到需要在全球范围内执行高级数据处理任务的数据密集型关键业务应用程序，均在使用 Azure SQL 数据库。

由于这是一种真正的 PaaS 服务，并且价格极富吸引力（最终可以帮助用户降低成本），如果用户的应用程序只使用了最基本的 SQL 数据库功能，无需实例级别的功能，那么建议将标准的 Azure SQL 数据库作为“默认选择”。标准版 Azure SQL 数据库不支持诸如 SQL CLR 集成、SQL Server Agent、跨数据库查询等功能，若要使用这些功能，只能选择 Azure SQL 数据库托管实例。

Azure SQL 数据库是唯一面向应用开发者打造的智能云数据库服务，同时也是唯一可以不停机即时伸缩的云数据库服务，借此可以帮助用户高效率地交付多租户应用。此外，Azure SQL 数据库可以帮助用户将更多时间用于业务创新，并能加快产品投放市场的速度。用户可以选择自己惯用的语言和平台构建安全可靠的应用程序，并将其连接到 SQL 数据库。

Azure SQL 数据库可提供下列收益：

- 内建机器学习能力，可学习并适应用户的应用
- 按需配置数据库

- 面向各类工作负载提供了不同类型的服务
- 99.99%高可用 SLA，零停机
- 通过地域复制和还原服务提供数据保护
- Azure SQL 数据库时间点还原功能
- 兼容 SQL Server 2016，包括混合功能和迁移

标准的 Azure SQL 数据库比 Azure SQL 数据库托管实例更像是 PaaS。因此如果可行，建议用户始终尝试首先使用标准服务，借此可从托管云服务中获得更多收益。然而 Azure SQL 数据库也与普通的本地 SQL Server 实例有一些关键的差异。取决于现有应用程序对数据库的需求，以及企业需求和策略，在规划上云项目时，该服务可能并非总是最佳选择。

何时将原先的 RDBMS 迁移至虚拟机 (IaaS)

迁移过程中，用户也可以将原先的关系型数据库管理系统 (RDBMS)，如 Oracle、IBM DB2、MySQL、PostgreSQL 或 SQL Server 迁移到 Azure 虚拟机中运行。如果现有应用程序必须在只进行最少量改动（甚至完全不进行任何改动）的情况下以最快速度迁入云端，那么可以考虑直接将其迁入云中的 IaaS 服务。这种方式也许无法获得上云的全部收益，但却是速度最快的方法。

目前 Microsoft Azure 支持通过 IaaS 虚拟机部署最多 [331 种不同的数据库服务器](#)，其中包括流行的 RDBMS，例如 SQL Server、Oracle、MySQL、PostgreSQL 和 IBM DB2，以及各种 NoSQL 数据库，例如 MongoDB、Cassandra、DataStax、MariaDB 和 Cloudera。

但是要注意，虽然将 RDBMS 迁移到 Azure 虚拟机可能是数据上云的最快方法（毕竟这是 IaaS），但这种方式需要对 IT 团队进行大量投入（数据库管理员和 IT 专业人员）。企业团队需要自行配置并管理高可用性、灾难恢复、SQL Server 补丁安装等工作。此外这种做法还要求具备有完整管理权限的可定制环境。

何时将 SQL Server 迁移为虚拟机 (IaaS)

有些情况下，用户可能依然需要将 SQL Server 迁移为普通虚拟机，例如需要使用 SQL Server Reporting Services。然而大部分情况下 Azure SQL 数据库托管实例即可提供原本在本地 SQL Server 上可以获得的一切，因此将 SQL Server 迁移为虚拟机应该视作最后一种选择。

使用 Azure Database Migration Service 将关系型数据库迁入 Azure

无论迁移的最终目标是 Azure SQL 数据库、Azure SQL 数据库托管实例，或 Azure 虚拟机中的 SQL Server，用户都可以使用 Azure Database Migration Service 将 SQL Server、Oracle 和 MySQL 等关系型数据库迁入 Azure。

自动化工作流和评估报表可以引导用户完成迁移前所必须做的改动，准备就绪后即可帮助用户将数据库迁移至 Azure。

如果原先的 RDBMS 有改动，可能还需要重新测试。此外取决于测试结果，可能还要更改应用程序中的 SQL 语句或 Object-Relational Mapping (ORM)。

如果还有其他数据库（如 IBM DB2）希望通过平移的方式迁移，除非愿意进行复杂的数据迁移工作，否则可能需要继续在 Azure 中通过 IaaS 虚拟机的方式运行这些数据库。更复杂的数据迁移工作往往需要付出更多精力，因为这种情况实际上是要将数据迁移到使用新数据库模式和编程类库的不同种类数据库中。

如果希望了解使用 Azure Database Migration Service 迁移数据库的更多信息，可参阅[使用 Azure SQL 数据库托管实例和 Azure Database Migration Service 更快速地上云](#)。

参考资源

- **选择一种云端 SQL Server 选项：Azure SQL 数据库 (PaaS) 或 Azure 虚拟机中的 SQL Server (IaaS)**
<https://docs.microsoft.com/en-us/azure/sql-database/sql-database-paas-vs-sql-server-iaas>
- **使用 Azure SQL 数据库托管实例和 Database Migration Service 更快速地上云**
<https://channel9.msdn.com/Events/Build/2017/P4008>
- **将 SQL Server 数据库迁移至云中 SQL 数据库**
<https://docs.microsoft.com/en-us/azure/sql-database/sql-database-cloud-migrate>
- **Azure SQL 数据库**
<https://azure.microsoft.com/en-us/services/sql-database/?v=16.50>
- **虚拟机中的 SQL Server**
<https://azure.microsoft.com/en-us/services/virtual-machines/sql-server/>

第 4 章: 将现有.NET 应用平移为云 DevOps 就绪的应用程序

愿景: 将现有.NET Framework 应用程序平移为云 DevOps 就绪的应用程序, 大幅提高部署敏捷性, 提高发布速度并降低交付成本。

为了充分运用云计算和容器等新技术的收益, 用户至少应该将现有.NET 应用程序的部分组件进行现代化。最终, 将整个企业的所有应用程序全面现代化也可进一步降低总体拥有成本。

对应用的部分组件进行现代化, 并不一定意味着全面迁移和重构。最初我们可以通过简单快速的平移方式实现现有应用程序的现代化, 并继续维护通过现有版本.NET Framework 开发, 依赖 Windows 和 IIS 的代码。图 4-1 凸显了云 DevOps 就绪应用在 Azure 应用程序现代化成熟度模型中的定位。

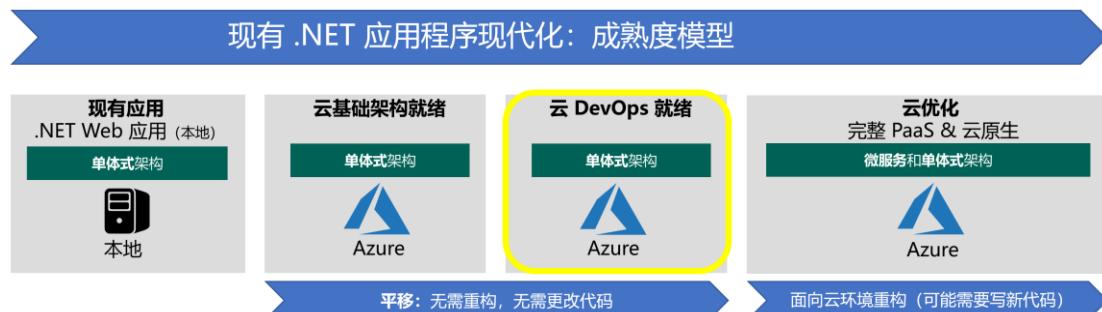


图 4-1: 云 DevOps 就绪应用程序的定位

将现有.NET 应用平移为云 DevOps 就绪应用程序的理由

对于云 DevOps 就绪的应用程序, 我们可以用快速、可重复的方式向客户交付可靠的应用, 并将复杂的运维工作交给运行应用的平台来承担, 借此提高敏捷性和可靠性。

如果无法以足够快的速度将应用程序投放市场, 那么在最终发布应用时, 您的目标市场可能已经产生了翻天覆地的变化。无论您的应用在架构或工程方面做得多出色, 可能也已经于事无补。您的业务可能失败, 或者无法发挥出应有的潜力, 而这一切只因为应用的发布速度跟不上市场需求的步伐。

对业务持续创新的需求为开发和运维团队造成了巨大的压力。为了实现持续不断的业务创新, 获得所需敏捷性的唯一方法就是使用容器等技术以及云 DevOps 就绪应用程序这一基本原则, 实现应用程序的现代化。

当一家组织可以构建并管理云 DevOps 就绪的应用程序时, 至少可以更快速将解决方案交付给客户, 并及时将相关想法投放市场。

云 DevOps 就绪应用程序的基本原则和宗旨

云技术的完善主要围绕两个目标：降低成本，通过更高灵活性促进业务增长。通过简化发布和交付应用程序的流程并减少过程中的阻力，即可实现这些目标。

对于真正云 DevOps 就绪的应用程序，您将可以（用非常敏捷的方式）独立于其他本地应用，自主地开发您自己的应用，随后在云端发布、部署、自动伸缩、监视并排错您的应用。

敏捷是关键。但除非能将从开发到投产全过程，以及开发/测试环境中遇到的问题数量降至最低，否则敏捷发布根本无从谈起。容器（尤其是已成既定标准的 Docker）和托管服务就是为了实现这一目的而设计的。

为实现敏捷性，用户还需要基于 CI/CD 流程打造自动化的 DevOps 过程，并通过可伸缩的云平台来运行。将 CI/CD 平台（例如 Visual Studio Team Services 或 Jenkins）部署到可缩放、高弹性的云平台（例如 Azure Service Fabric 或 Kubernetes），才是借助云环境实现敏捷性的关键。

下文列表描述了云 DevOps 就绪应用程序的主要原则或实践。请注意，您可以通过渐进的，或者逐渐改进的方式，采纳下列部分或全部基本原则：

- **容器。** 容器可供您将应用程序本身以及依赖项打包到一起。这种做法可大幅降低在生产环境中部署或在暂存（Staging）环境中测试时问题的出现。此外容器还能提高应用程序交付过程的敏捷性。
- **弹性可伸缩的云。** 云能提供托管的、弹性的、易伸缩、可复原的平台。这些特征是以持续交付方式发布高可用、可靠的应用程序，并降低成本的关键。为降低应用程序的维护成本，免不了要使用诸如托管数据库、托管缓存即服务（CaaS）、托管存储等托管式服务。
- **监视。** 如果不能以足够好的方式检测并诊断应用程序异常或与性能有关的问题，就无法交付可靠的应用程序。为此我们需要通过应用程序性能管理和即时分析获得可行的见解。
- **DevOps 文化和持续交付。** 采纳云 DevOps 就绪实践还要求对团队文化进行改革，团队应避免“各自为战”。只有加强开发和 IT 运维团队之间的协作，并得到容器和 CI/CD 工具的支持，CI/CD 流程才会变得可行。

图 4-2 展示了云 DevOps 就绪应用程序主要的实现方向。实现过程中涵盖的方向越全面，应用程序成功满足客户预期的效果就会越出色。

云 DevOps 就绪应用程序



图 4-2：云 DevOps 就绪应用程序的主要实现方向

简而言之，云 DevOps 就绪应用程序是一种构建和管理应用程序的方法，这种应用程序可以充分利用云计算模式，并结合使用容器、托管云基础架构、高弹性应用程序技术、监视、持续交付和 DevOps 等技术，而这一切并不需要重构或重写您的现有应用程序。

您的组织可以逐步采用这些技术和方法，并不需要一次性全部采纳。取决于工作的优先级和用户需求，您也可以循序渐进地采纳。

云 DevOps 就绪应用程序可提供的收益

通过将现有应用程序（不重构，不改代码）转换为云 DevOps 就绪应用程序，可获如下收益：

- **降低成本，因为托管基础架构由云提供商负责管理。** 云 DevOps 就绪应用程序可以获得云平台拆箱即用的弹性、自动伸缩能力，以及高可用性，借此从云中获益。这些收益不仅来自计算能力（虚拟机和容器），而且可能来自云端的资源，如 DBaaS、CaaS 以及应用程序所需的其他基础架构。
- **高弹性的应用程序和基础架构。** 迁移上云后，依然可能遇到短暂的故障，云平台也会出故障。此外云基础架构和硬件是可以“重新安置”的，这也增加了短暂停机的概率。但同时云平台固有的能力和某些应用程序开发技术可通过实现高弹性和自动化恢复能力，使得应用可以更容易从云的故障中恢复。
- **深入了解应用程序性能。** 而如 Azure Application Insights 等云端监视工具可以对应用程序的运行状况管理、日志以及通知提供可视化功能。审核日志使得用户可以更容易地调试和审核应用程序。这些都是塑造可靠云应用程序的基础。
- **应用程序可移植能力和敏捷开发能力。** 容器（无论基于 Docker Engine 的 Linux 或 Windows 容器）为避免应用程序受制于特定云平台提供了最佳解决方案。通过使用容

器、Docker 主机以及支持多种云的编排引擎，即可轻松在不同环境或云平台之间切换。容器技术同时还消除了部署到不同环境（暂存/测试/生产）过程中常见的阻力。

所有这些收益最终将能在应用程序的完整生命周期内大幅降低成本。

下文将详细介绍这些收益，以及可能用到的具体技术。

云 DevOps 就绪应用程序中的微软技术

下文列出了云 DevOps 就绪应用所需的工具、技术和解决方案。取决于工作优先级，用户可以有选择地，或者逐步采纳云 DevOps 就绪应用。

- **云基础架构：**提供计算平台、操作系统、网络和存储的基础架构，Microsoft Azure 是这种场景的首选。
- **运行时：**为应用程序提供了运行所需的环境。如果使用容器，这一层通常会基于运行在 Linux 或 Windows 主机上的 [Docker Engine](#)。（对 [Windows 容器](#)的支持始于 Windows Server 2016，Windows 容器是在 Windows 上运行的现有 .NET Framework 应用程序的最佳选择）。
- **托管云：**如果选择托管云，就无需面对管理和支持底层基础架构、虚拟机、操作系统补丁、网络等工作的开销和复杂度。如果决定迁移至 IaaS，则需自行负责所有这些任务，并承担相关的成本。在托管云方式中，用户只需要管理自己部署的应用程序和服务。通常由云服务提供商管理其它一切。例如 Azure 提供了这些托管云服务：[Azure SQL 数据库](#)、[Azure Redis 缓存](#)、[Azure Cosmos DB](#)、[Azure 存储](#)、[Azure Database for MySQL](#)、[Azure Database for PostgreSQL](#)、[Azure Active Directory](#)，以及其他托管的计算服务，如[虚拟机规模集](#)、[Azure Service Fabric](#)、[Azure 应用服务](#)和[Azure 容器服务](#)。
- **应用程序开发：**构建在容器中运行的应用程序时，可使用多种语言。本文主要专注于[.NET](#)，但用户也可使用其他语言，例如 Node.js、Python、Spring/Java 或 GoLang 开发基于容器的应用。
- **监视、遥测、日志和审核：**对任何云 DevOps 就绪应用程序来说，云中运行的应用程序和容器的监视和审核能力都是至关重要的。[Azure Application Insights](#) 和 [Microsoft Operations Management Suite](#) 是微软为云 DevOps 就绪应用程序提供的主要监视和审核工具。
- **配置：**自动化工具可以帮助用户配置基础架构并将应用程序部署到多种环境（生产、测试、暂存）。用户可以使用诸如 Chef、Puppet 等工具管理应用程序的配置和环境。这一层还可以通过更简单直接的方式实现，例如直接使用 Azure 命令行接口（Azure CLI）工具部署，随后使用 [Visual Studio Team Services](#) 中的持续部署和发布管理流程。
- **应用程序生命周期：**[Visual Studio Team Services](#) 以及 Jenkins 等工具构建的自动化服务器可以帮助用户实现 CI/CD 流程，包括发布管理。

下一节以及相关过程展示将侧重于运行时（Windows 容器）的细节。相关指南介绍了在 Windows Server 2016（以及后续版本）虚拟机中部署 Windows 容器的方法，同时还将介绍其他更先进的编排引擎层，例如 Azure Service Fabric、Azure 容器服务（如 Kubernetes）。编排引擎层的设置是现有.NET Framework（基于 Windows）应用程序实现现代化，以及打造云 DevOps 就绪应用程序的基本要求。

单体式应用程序也可以云 DevOps 就绪

有一个不容忽略的重点：单体式应用程序（并非基于微服务的应用程序）同样可以成为云 DevOps 就绪应用程序。我们构建并运维的单体式应用程序一样可以借助云计算模型，通过配合使用容器、持续交付和 DevOps 赢得。如果现有的单体式应用程序已经可以满足业务目标，那么即可将其现代化，使其变得云 DevOps 就绪。

同理，如果单体式应用程序可以成为云 DevOps 就绪应用程序，那么 N 层应用程序这种更复杂的应用程序一样可以现代化成为云 DevOps 就绪应用程序。

云优化应用程序又该怎么办？

虽然云优化和云原生应用程序并非本文的重点，但依然有必要了解它们对应的现代化成熟度，并理解它们与云 DevOps 就绪的差异。

图 4-3 展示了云优化应用在应用程序现代化成熟度中的定位：

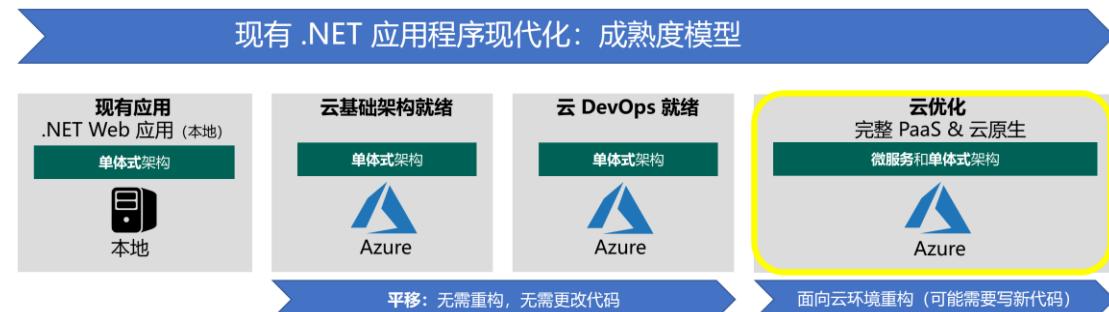


图 4-3：云优化应用程序的定位

在现代化过程的不同成熟度中，云优化这一程度通常需要在开发方面进行新的投入，升级至云优化级别，这通常是由业务需求推动的，业务希望将应用程序尽可能现代化，以降低成本，改善敏捷性，获得竞争优势。这些目标可通过云端 PaaS 获得最大化效果。这意味着不仅需要使用诸如 DBaaS、CaaS 和存储即服务（STaaS）等 PaaS 产品，而且需要将现有应用程序和服务迁移至 PaaS 计算平台，例如 Azure 应用服务，或使用编排引擎。

此类现代化通常需要重构或编写新的代码，这些代码需要针对云端 PaaS 平台（例如 Azure 应用服务）进行优化。甚至可能需要明确针对云环境调整架构，尤其是打算迁移为基于微服务的云原生应用模型时。相比无需重构或编写新代码的云 DevOps 就绪，这是个很大差异。

在某些高级场景中，我们可能需要根据微服务架构创建云原生应用程序，这种方式可以提供

本地环境部署的单体式应用程序完全无法比拟的敏捷性和伸缩能力。

图 4-4 展示了云优化模型中可以部署的应用程序类型。此时有两个基本选择：现代化 Web 应用程序，以及云原生应用程序。

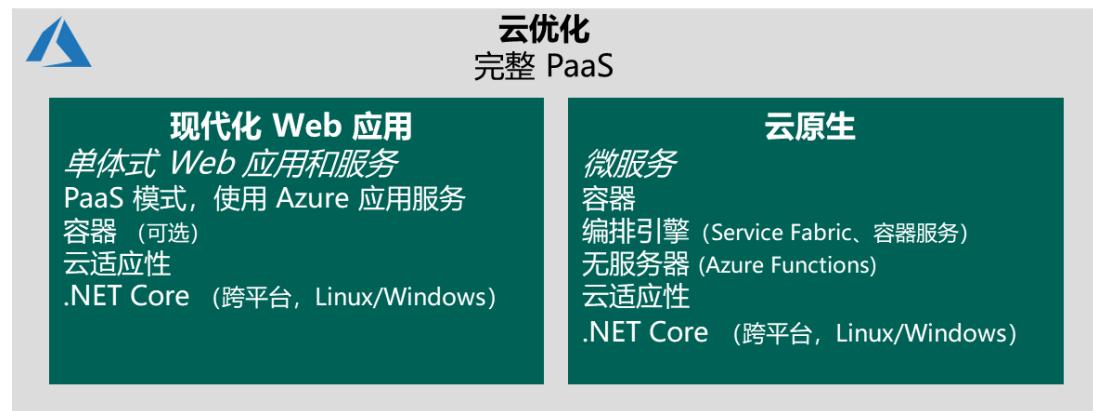


图 4-4：云优化级别下的应用类型

通过添加诸如人工智能 (AI)、机器学习 (ML) 以及物联网等服务，还可对已经基本实现现代化的 Web 应用和云原生应用进行扩展。这些服务均可用来扩展各种云优化应用。

云优化级别的应用程序最基本的不同在于应用程序架构。按照定义，[云原生](#)应用程序是一种基于微服务的应用。相比单体式 Web 应用程序或传统的 N 层应用程序，云原生应用需要专门的架构、技术和平台。

对于新创建的应用，在很多场景中，不使用微服务也是一种合理做法。很多新出现的并且依然足够现代化的场景中，基于微服务的方法可能有些大材小用，某些情况下，我们可能只是需要构建一个简单的单体式 Web 应用，或为原有的 N 层应用添加一些简单的服务。此时依然可以借助云 PaaS 的所有能力，例如 Azure 应用服务提供的各项功能。这种方式依然有助于降低维护工作量。

另外，因为要在云优化场景中开发新代码（开发完整应用程序，或开发某个子系统），在编写新代码时，可以使用新版本的.NET ([.NET Core](#)，尤其是 [ASP.NET Core](#))。如果要创建微服务或容器，尤其建议考虑使用新版，因为.NET Core 是一种更精益也更快速的框架。磁盘占用越少，在容器中的启动速度越快，可以进一步提高应用程序的性能。这种方法可以很好地满足对微服务和容器的需求，同时依然可以从跨平台框架中获益，在 Linux、Windows Server 和 Mac (Mac 仅限开发环境) 上运行同一个应用程序。

云原生和云优化应用程序

对大型和关键业务应用程序来说，[云原生](#)是一种更先进，或者说更成熟的状态。云原生应用程序通常需要从零开始全新创建架构和设计，不能通过对现有应用程序现代化获得。云原生应用程序，相对更为简单，部署到 PaaS 平台的云优化 Web 应用程序，两者之间最大的差异在于，云原生方法更推荐使用微服务的架构。云优化应用也可以是部署到 Azure 应用服务等

PaaS 云平台上的单体式 Web 应用程序或 N 层应用程序。

[Twelve-Factor App](#) (一系列与微服务方法密切相关的开发模式) 也被视作[云原生](#)应用程序架构的一种前提要求。

[Cloud Native Computing Foundation \(CNCF\)](#)是云原生原则的主要推动者，微软是[CNCF 成员](#)。

如果希望通过范例进一步了解该概念的定义，或想了解有关云原生应用程序不同特征的细节信息，请参阅 Gartner 文章：[如何架构并设计云原生应用程序](#)。如果希望了解微软针对云原生应用程序实现方法提供的指南，请参阅：[.NET 微服务：容器化.NET 应用程序的架构](#)。

如果要将完整的应用程序迁移为[云原生](#)模式，最重要的问题在于必须全面考虑采用一种全新的，基于微服务的架构。由于需要重构，这无疑需要在开发方面进行大量投入。因此通常只有关键业务应用程序需要获得全新程度的缩放能力和长期敏捷性时，才会考虑这样做。但我们也一样可以先只为少数几个新的使用场景添加微服务，借此向着云原生迈出第一步，最终将整个应用程序全面重构为微服务。对某些场景来说，这种循序渐进的方式最为适合。

微服务是怎么回事？

在为组织考虑使用云原生应用时，一定要首先理解微服务以及微服务的工作原理。

微服务架构是一种先进的应用程序开发方法，全新开发应用，或向着云原生方向完善现有应用程序时均可考虑（无论本地应用或云 DevOps 就绪应用）。首先可以为现有应用程序添加少量微服务，借此了解微服务的全新使用范式。但毫无疑问，我们需要架构和工具，尤其是在面对这种全新架构的方法时。

然而并非任何新的或现代化应用就必须使用微服务。微服务并不是“万灵药”，也不是开发每个应用程序时唯一可选的最佳方法。如何使用，何时使用微服务，这取决于想要构建的应用程序所属的类型。

微服务架构已成为分布式应用，以及以多个自主服务方式连接的相互独立子系统的大型或复杂关键业务应用程序首选的方法。在基于微服务的架构中，应用程序实际上是由一系列分别开发、测试、版本控制、部署、伸缩的服务组成的。每个微服务均可包含任何相关的自主数据库。

如果希望详细了解可以通过.NET Core 实现的微服务架构详情，推荐阅读可下载的 PDF 版电子书：[.NET 微服务：容器化.NET 应用程序的架构](#)，相关指南也可[在线](#)阅读。

但就算可以通过微服务的强大功能（独立部署、强大的子系统边界、技术多样性）获益的场景，同样会遇到很多新挑战。这些挑战主要来自分布式应用程序的开发，例如碎片化并且相互独立的数据模型，微服务之间的弹性通信，对最终一致性的需求，以及复杂的运维工作。相比传统的单体式应用程序，微服务让复杂性问题愈加突出。

由于架构复杂，只有特定场景和某些类型的应用程序适合使用基于微服务的架构。例如使用

多个快速演进的子系统的大型分布式应用程序。这种情况下，值得投资开发一种更复杂的软件架构，借此在长期范围内获得敏捷度并实现更高效的应用程序维护。但对不那么复杂的场景，继续使用单体式应用程序方法或更简单的 N 层方法也许是更好的选择。

最后要注意，尽管一直在反复强调，但为应用程序使用微服务时，不应“全盘使用或全盘不使用”。更好的方法是陆续添加基于微服务的全新小规模场景，借此对现有的单体式应用程序进行扩展和革新，并不需要使用微服务架构的方式从零开始构建全新的应用程序。实际上，建议的做法是先增加使用场景，借此革新现有的单体式或 N 层应用程序。最终即可将应用程序拆解为相互自主的组件或微服务。单体式应用程序向着微服务方向的演变可以循序渐进地执行。

何时使用 Azure 应用服务对现有.NET 应用实现现代化

在将现有 ASP.NET Web 应用程序现代化至云优化成熟度级别的过程中，因为 Web 应用程式是使用.NET Framework 开发地，因此可能主要会依赖 Windows，并很可能依赖 Internet Information Server (IIS)。在使用和部署基于 Windows 以及 IIS 的应用程序时，可以直接将其部署至 [Azure 应用服务](#)，或首先使用 Windows 容器技术将应用程序容器化。如果要容器化，可将应用程序部署至 Windows 容器主机（基于虚拟机）或支持 Windows 容器的 Azure Service Fabric 集群中。

如果使用 Windows 容器，即可获得容器化所能提供的全部收益。应用的发布和部署过程会更加敏捷，环境（暂存、开发/测试、生产）问题的阻力也会更小。下文将详细介绍使用容器可获得的收益。

截止撰写本文时，Azure 应用服务尚不支持 Windows 容器，但可支持 Linux 容器。因此我们可能会问：“Azure 应用服务和 Windows 容器之间该如何选择？”

基本上，如果 Azure 应用服务可以支持你的应用程序，并且没有因为与服务器或自定义依赖项有关的问题妨碍使用应用服务，即可考虑将现有.NET Web 应用程序迁移至应用服务。这是维护应用程序最简单高效的做法。迁移到 Azure 后应用程序的维护工作也会更简单，因为 DBaaS、CaaS 和 STaaS 等 PaaS 服务可以简化很多工作。

然而如果应用程序对服务器或自定义组件有一定的依赖性，且 Azure 应用服务无法支持，那么可能需要考虑使用 Windows 容器。例如对服务器或自定义组件的依赖可能包括：需要在服务器上安装第三方软件或.msi 文件，而 Azure 应用服务无法支持。另外还有一个例子，必要的某些服务器配置可能不被 Azure 应用服务支持，例如需要使用 Global Assembly Cache (GAC) 程序或 COM/COM+组件。不过好在有了 Windows 容器镜像，我们可以将所有自定义依赖项直接包含到同一个“部署单位”中。

或者可以对应用程序中不被 Azure 应用服务支持的部分进行重构。取决于重构需要的工作量，可能需要慎重评估是否值得这样做。

迁移至 Azure 应用服务可获得的收益

Azure 应用服务是一种全面的托管式 PaaS 产品，可以帮助用户轻松构建业务流程所需的 Web 应用程序。在使用应用服务后，可省略升级和维护本地 Web 应用所需的基础架构管理成本。尤其是可大幅缩减在本地运行 Web 应用所需的硬件和许可成本。

如果 Web 应用程序适合迁移至 Azure 应用服务，最主要的收益在于可以用更短时间完成迁移工作。应用服务提供了一个非常简单易于上手的环境。

Azure 应用服务是大部分 Web 应用的最佳选择，是 Azure 中运行 Web 应用最简单的 PaaS 服务。该平台已经集成了部署和管理能力，网站可快速伸缩以应对激增的流量负载，同时内建的负载均衡和流量管理器功能可保障高可用性。

在 Azure Application Insights 的帮助下，Web 应用的监视工作也变得更简单。Application Insights 免费包含在应用服务中，无需在应用程序中编写任何特殊代码即可使用。直接在应用服务中运行 Web 应用，无需额外操作即可获得一个完善的监视系统。

用户还可在应用服务中直接使用来自 Azure Web 应用程序库的大量开源应用(如 WordPress 或 Umbraco)，或使用各种框架和工具自行新建网站，例如可以使用 ASP.NET。应用服务的 WebJobs 功能可以帮助我们轻松地为应用服务中运行的 Web 应用增加后台作业处理能力。

使用 Azure 应用服务的 Web 应用功能迁移 Web 应用可获得的主要收益包括：

- 自动伸缩以满足忙时需求，并在闲时降低成本。
- 通过自动网站备份保护所做改动和数据。
- Azure PaaS 平台提供的高可用性和高弹性。
- 为开发和暂存环境提供部署槽，并可用于测试多个网站。
- 负载均衡和分布式拒绝服务 (DDoS) 保护。
- 通过流量管理将用户定向至地理位置最近的部署。

应用服务可能是新 Web 应用的最佳选择，对于现有应用程序，如果它的应用依赖能被应用服务支持，那么应用服务也将成为迁移时的首选。

参考资源

- **Azure 应用服务兼容性分析**
<https://www.migratetoazure.net/Resources>

迁移至 Windows 容器可获得的收益

使用 Windows 容器的主要收益在于，相比非容器化应用，可以获得更可靠和完善的部署体验。此外使用容器实现应用程序的现代化，还可以让应用程序面向其他可支持 Windows 容

器的平台和云环境做好充分准备。下文将详细介绍迁移至 Windows 容器可获得的收益。

Azure 提供的，可支持 Windows 容器的主要计算环境（截至 2017 年中期正式发布的）包括 Azure Service Fabric 和基本的 Windows 容器主机（Windows Server 2016 虚拟机）。本文也将主要介绍这些基础架构场景的使用环境。

此外还可将 Windows 容器部署到其他编排引擎，如 Kubernetes、Docker Swarm 或 DC/OS。目前（2017 年初秋），在 Azure 容器服务中这些平台对 Windows 容器的支持处于预览阶段。

如何将现有.NET 应用部署至 Azure 应用服务

Azure 应用服务的 Web 应用功能是一种托管式计算平台，并为网站和 Web 应用的托管进行了优化。Microsoft Azure 提供的这个 PaaS 产品可供我们专注于业务逻辑，基础架构的运行和应用的伸缩可交由 Azure 处理。

使用 Azure App Service Migration Assistant 评估网站并迁移至应用服务

如果使用 Visual Studio 新建应用程序，将其迁移至应用服务的过程其实非常简单。然而如果打算将现有应用程序迁移至应用服务，首先需要评估应用程序的所有依赖项是否都可兼容应用服务。例如对服务器操作系统，以及对服务器上安装的任何第三方软件的依赖。

此时可使用 [Azure App Service Migration Assistant](#) 分析网站并将其从 Windows 或 Linux Web 服务器迁移至应用服务。迁移过程中，该工具会按需在 Azure 中创建 Web 应用和数据库，发布内容，并发布数据库。

Azure App Service Migration Assistant 支持从 Windows Server 上运行的 IIS 迁移至云端。应用服务可支持 Windows Server 2003 以及后续版本。

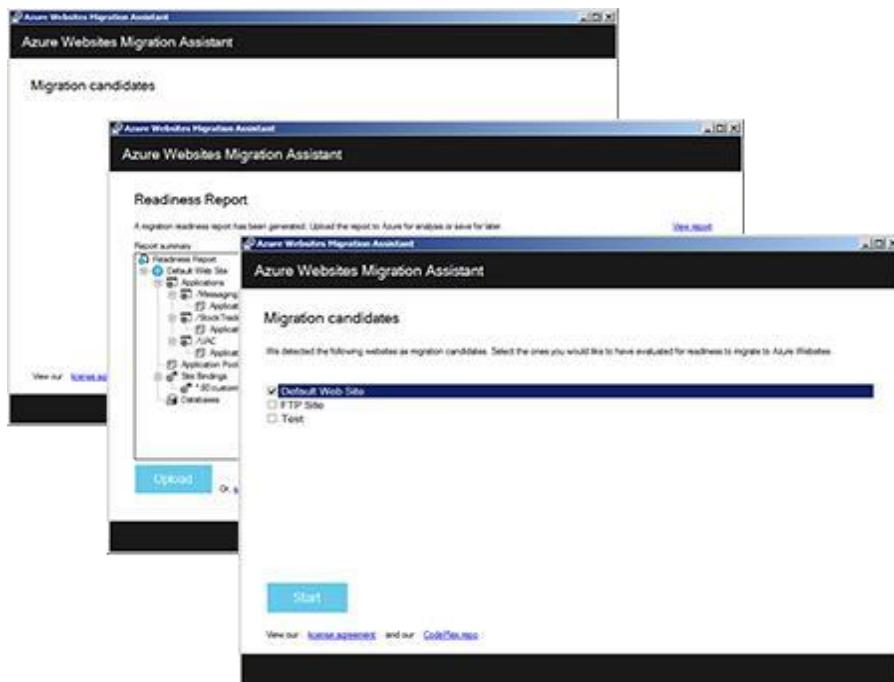


图 4-5：使用 Azure App Service Migration Assistant

App Service Migration Assistant 可帮助用户将网站从自己的 Web 服务器迁移到 Azure 云中。

网站迁移到应用服务后，即可获得安全高效运行所需的一切支持。网站会自动在 Azure 云 PaaS 服务（应用服务）中设置完成并顺利运行。

该工具可分析网站并针对迁移至应用服务过程中的兼容性创建报表。如果分析结果显示一切正常，即可让 App Service Migration Assistant 自动迁移内容、数据和设置。如果存在兼容性问题，迁移工具会显示该如何调整才能顺利完成迁移。

参考资源

- **Azure App Service Migration Assistant**
<https://www.migratetoazure.net/>

将现有.NET 应用部署为 Windows 容器

云优化应用程序、云原生应用程序，以及云 DevOps 就绪应用程序都适合基于 Windows 容器的部署方式。

下文将侧重于在平移现有.NET 应用程序时，使用 Windows 容器部署云 DevOps 就绪应用程序的做法。

容器是什么？（Linux 或 Windows）

容器是一种将应用程序打包为独立程序包的做法。位于容器中的应用程序不会受到容器之外

应用程序或进程的影响。应用程序成功运行所需的一切均以进程的形式运行在容器内部。无论将容器移动到哪里，应用程序直接依赖的其他内容也可同时移动，因为应用程序运行所需的一切（依赖的库、运行时等）都打包在一起了。

容器的主要特征在于，可以跨越不同部署维持环境的一致，因为容器本身就包含了所有必须的依赖项。这意味着我们可以在自己的计算机上调试应用程序，随后部署到另一台计算机，不同计算机均可保证提供完全相同的环境。

容器可以看作容器镜像的一个实例。容器镜像是一种对应用或服务打包（类似快照），随后以可靠、可再现的方式部署的方法。可以说 Docker 已不仅仅是一种技术，而成为了一种方法论甚至流程。

容器日渐普及，已经成为行业通用的“部署单位”。

容器的收益(Windows 或 Linux 上使用的 Docker Engine)

使用容器（也可以看作一种轻量级的构建块）构建应用程序可大幅改善跨越任何基础架构，构建、发布、运行任何应用程序过程中的敏捷性。

通过使用容器，只需少量更改代码，甚至完全无需更改，即可将任何应用从开发环境直接发布到生产环境，这一切都要归功于微软开发者工具、操作系统以及云平台实现对 Docker 的紧密集成。

在直接部署到虚拟机时，可能已经具备了将 ASP.NET 应用部署到虚拟机的标准做法。这样的做法很可能涉及大量手工操作，或使用诸如 Puppet 或其他类似部署工具执行复杂的自动化流程。修改配置项，在服务器之间复制应用程序内容，通过.msi 文件运行交互式的安装程序，执行测试，这一过程中可能需要执行很多此类任务。部署过程中所有这类任务都会延长部署所需时间并导致风险增加。如果目标环境缺少依赖项，最终部署将会失败。

在 Windows 容器中，应用程序的打包过程将完全自动化。Windows 容器基于 Docker 平台，可为容器的部署实现自动化更新和回滚。使用 Docker 引擎获得的主要收益在于：可以直接创建镜像，镜像类似于应用程序快照，其中包含了所有依赖项。这是一种 Docker 镜像（本例则为 Windows 容器镜像），镜像无需获得源代码即可在容器内部运行 ASP.NET 应用。容器快照目前已成为部署的最基本单位。

很多组织正在对现有单体式应用进行容器化，这样做的主要原因包括：

- **改善部署过程实现敏捷发布。**容器可为开发和运维人员提供一致的部署规则。如果使用容器，就不会听到开发者说：“我在计算机上可以运行，生产环境中为什么就不行了？”他们反而会说：“它们是在容器中运行的，因此肯定可以在生产环境中运行。”将应用程序和所有依赖项打包在一起，即可在任何受支持的容器环境中运行。无论部署到任何目标位置（开发、测试、镜像、生产），均能以一致的方式运行。容器可以消除应用程序在不同环境间移动时可能遇到的大部分阻碍，借此大幅改善部署和发布速度。

- **降低成本。**通过整合并减少所需硬件，以及让运行应用的硬件实现更高密度运行，容器可大幅降低成本。
- **可移植性。**容器是模块化、可移植的。Docker 容器已得到所有服务器操作系统（Linux 和 Windows）、各大主要公有云平台（Microsoft Azure、Amazon AWS、Google、IBM），以及本地部署和私有/混合云环境的支持。
- **控制。**容器提供了灵活安全，可在容器层面控制的环境。通过设置执行限制策略，可以保护、隔离，甚至限制容器。正如 Windows 容器一节所述，Windows Server 2016 和 Hyper-V 容器还可提供更丰富的企业支持选项。

在敏捷性、可移植性，以及控制能力方面的大幅改进使得用户使用容器开发和维护应用程序时可以大幅降低成本。

Docker 是什么？

Docker 是一个[开源项目](#)，能通过可移植、自包含的容器将应用程序自动部署至云端或本地环境。Docker 同时也是一[家公司](#)，该公司在积极推动并完善这一技术。该公司已经与很多云平台、Linux 以及 Windows 供应商，包括和微软建立紧密的合作关系。

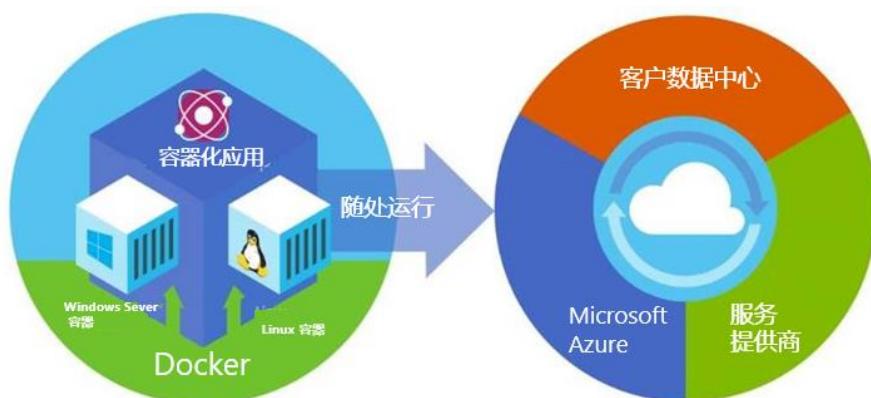


图 4-6：Docker 可将容器部署到混合云的所有层面上

在一些熟悉虚拟机的人看来，容器可能与虚拟机非常类似。容器运行了操作系统，有自己的文件系统，可通过网络访问，看似与物理或虚拟机系统无异。然而容器背后的技术和概念与虚拟机有很大差别。从开发者的角度来看，容器其实更像是一个进程。实际上，容器为每个进程都提供了一个入口点。

Docker 容器(简称为容器)可在 Linux 和 Windows 上原生运行。运行普通的容器时，Windows 容器只能在 Windows 主机(主机服务器或虚拟机)中运行，Linux 容器只能在 Linux 主机中运行。然而在最新版 Windows Server 和 Hyper-V 容器中，Linux 容器也可以在 Windows Server 中使用 Hyper-V 隔离技术原生运行，这项技术目前只支持 Windows Server 容器。

在不远的未来，同时包含 Linux 和 Windows 容器的混合环境不仅可行，而且会极为普遍。

通过 Windows 容器运行现有.NET 应用程序的收益

Windows 容器同样可以提供常规意义上容器技术所能提供的全部收益，使用 Windows 容器同样可以改进敏捷性、可移植性，以及控制能力。

对于现有.NET 应用程序，主要是指使用.NET Framework 创建的传统应用程序。例如可能是传统的 ASP.NET Web 应用程序，但并非可以在 Linux、Windows 和 macOS 上跨平台运行的新版.NET Core。

.NET Framework 主要依赖 Windows，此外可能还依赖其他组件，如 IIS，以及传统 ASP.NET 中的 System.Web。

.NET Framework 应用程序只能在 Windows 上运行。如果想容器化现有.NET Framework 应用程序，同时无法或者不愿意迁移至.NET Core（“如果可以正常运行，就不要迁移”），此时 Windows 容器将成为唯一可用的容器技术。

因此可以说，Windows 容器的主要收益之一就是：可以帮助用户通过容器技术对运行在 Windows 上的现有.NET Framework 应用程序实现现代化。最终，Windows 容器也能帮助用户获得容器技术所应提供的收益：敏捷、可移植、更好的控制能力。

为.NET 容器选择目标操作系统

考虑到 Docker 可支持多样化的操作系统，并且.NET Framework 和.NET Core 之间存在巨大的差异，因此用户需要结合自己使用的框架确定最适合的操作系统和版本。

对于 Windows，可以使用 Windows Server Core 或 Windows Nano Server。这些 Windows 版本提供了.NET Framework 或.NET Core 应用程序可能需要的不同特征（如 IIS，以及 Kestrel 等自托管 Web 服务器）。

对于 Linux，官方提供的.NET Docker 镜像已可以使用并支持多种发行版（如 Debian）。

图 4-7 展示了取决于应用程序所用的.NET Framework 版本，用户可以选择的操作系统版本。

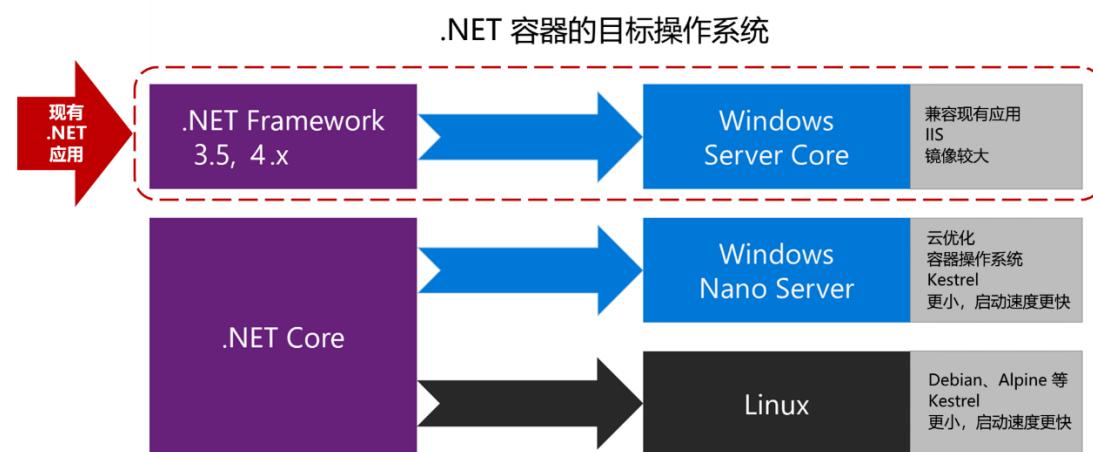


图 4-7：根据.NET Framework 版本选择目标操作系统

在迁移基于.NET Framework 的现有或遗留应用程序时，主要依赖项包括 Windows 和 IIS。此时只能选择基于 Windows Server Core 和.NET Framework 的 Docker 镜像。

在将镜像名称加入 Dockerfile 文件的过程中，可以使用标签选择操作系统与版本，例如通过下列标签可选择基于.NET Framework 的 Windows 容器镜像：

标签	系统和版本
microsoft/dotnet-framework:4.x-windowsservercore	Windows Server Core 上的.NET Framework 4.x
microsoft/aspnet:4.x-windowsservercore	Windows Server Core 上的.NET Framework 4.x，以及额外的 ASP.NET 定制

对于.NET Core (Linux 和 Windows 跨平台)，可使用下列标签：

标签	系统和版本
microsoft/dotnet:2.0.0-runtime	Linux 上的.NET Core 2.0 运行时
microsoft/dotnet:2.0.0-runtime-nanoserver	Windows Nano Server 上的.NET Core 2.0 运行时

多架构镜像

从 2017 年中期开始，我们已经可以开始使用 Docker 新增的[多架构 \(Multi-arch\)](#) 镜像功能。.NET Core Docker 镜像可以使用多架构标签，Dockerfile 文件不再需要定义目标操作系统。多架构功能可以为多种计算机配置使用同一个标签。例如，通过使用多架构功能，只需要使用一个通用的标签：**microsoft/dotnet:2.0.0-runtime**。从 Linux 容器环境拉取该标签可获得基于 Debian 的镜像，从 Windows 容器环境拉取可获得基于 Nano Server 的镜像。

对于.NET Framework 镜像，由于传统.NET Framework 仅支持 Windows，因此无法使用多架构功能。

Windows 容器的类型

与 Linux 容器类似，Windows Server 容器也可通过 Docker Engine 管理。但与 Linux 容器的不同之处在于，Windows 容器提供了两种不同类型容器（或称运行时）：Windows Server 容器，以及 Hyper-V 隔离。

Windows Server 容器：可通过进程和名称空间隔离技术实现应用程序的隔离。Windows Server 容器会在主机以及主机上运行的所有容器之间共享同一个内核，这些容器无法面向恶意行为提供安全边界，不应用于隔离不受信任的代码。由于内核空间是共享的，因此容器需要使用相同的内核版本和配置。

Hyper-V 隔离：通过在深度优化的虚拟机中运行每个容器，可实现比 Windows Server 容器

更进一步的隔离能力。这种情况下，容器主机的内核不会与同一主机运行的其他容器共享。这种容器在设计上主要面向可能存在恶意行为的多租户托管环境，可提供与虚拟机一致的安全保障。由于这种容器不与主机或主机上的其他容器共享内核，因此可运行（受支持的）不同版本与配置的内核。例如 Windows 10 中的 Windows 容器就是通过 Hyper-V 隔离技术实现 Windows Server 不同版本和配置的内核同时运行的。

在 Windows 上运行容器时是否使用 Hyper-V 隔离，要在运行过程中决定。例如最开始可能选择通过 Hyper-V 隔离创建容器，并在运行的时候选择通过 Windows Server 容器来运行。

参考资源

- **Windows 容器文档**
<https://docs.microsoft.com/en-us/virtualization/windowscontainers/>
- **Windows 容器的基本原理**
<https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/>
- **信息图：微软和容器**
<https://info.microsoft.com/rs/157-GQE-382/images/Container%20infographic%201.4.17.pdf>

何时不应部署到 Windows 容器

一些 Windows 技术目前尚不被 Windows 容器支持。此时用户依然需要迁移到标准的虚拟机，通常虚拟机中可能运行了 Windows 和 IIS。

截止 2017 年中期，Windows 容器暂无法支持的功能包括：

- Windows 容器目前不支持 Microsoft Message Queuing (MSMQ)。
 - [UserVoice 请求论坛](#)
 - [讨论论坛](#)
- Windows 容器目前不支持 Microsoft Distributed Transaction Coordinator (MSDTC)。
 - [GitHub 问题](#)
- Microsoft Office 目前不支持容器。
 - [UserVoice 请求论坛](#)

有关社区提供的目前无法支持的其他场景与请求，请参阅 UserVoice 上的 Windows 容器论坛：<https://windowsserver.uservoice.com/forums/304624-containers>。

参考资源

- **Azure 中的虚拟机和容器**
<https://docs.microsoft.com/en-us/azure/virtual-machines/windows/containers>

何时将 Windows 容器部署到本地 IaaS 虚拟机基础架构

出于很多原因，用户可能需要将 Windows 容器部署到本地基础架构（虚拟机或裸机服务器）：

- 组织可能没有准备好上云，或者出于业务原因不能上云。但此时依然可以在自己的数据中心中通过 Windows 容器获益。
- 可能有一些组件需要在本地运行，如果将其迁入云端可能会影响速度。例如出于安全或身份验证的需求要使用本地 Windows Server Active Directory 或其他本地系统。
- 如果现在开始使用 Windows 容器，未来即可分阶段上云并取得更好的效果。Windows 容器已成为所有云的部署单位，不会造成技术锁定。

何时将 Windows 容器部署至 Azure 虚拟机（IaaS 云）

如果用户已经在使用 Azure 虚拟机，哪怕同时使用了 Windows 容器，依然会面对 IaaS 环境。这意味着如果需要将多个虚拟机部署到负载均衡的基础架构借此打造更高缩放能力的应用程序，此时需要面对基础架构运维、虚拟机补丁安装，以及与基础架构有关的其他复杂问题。在 Azure 虚拟机中使用 Windows 容器的主要场景包括：

- **开发/测试环境**：云中虚拟机很适合基于云环境的开发和测试。用户可以根据需求快速创建或停用这样的环境。
- **中小规模的弹性需求**：一些情况下用户可能只需要通过少量虚拟机搭建生产环境，再迁移至更先进的 PaaS 环境，例如使用编排引擎，少量虚拟机管理起来可能较为轻松。
- **包含现有部署工具的生产环境**：用户可能在本地环境投资了各种向虚拟机或裸机服务器执行复杂部署任务的工具（如 Puppet 或其他类似工具），并希望从这样的本地环境中迁移。为了在上云迁移的同时将对本地生产环境中部署流程的变动降至最低，可以继续使用这些工具执行面向 Azure 虚拟机的部署。然而这一过程中，用户可能依然会希望使用 Windows 容器作为部署单位，以便进一步完善部署体验。

何时将 Windows 容器部署至 Service Fabric

基于 Windows 容器的应用程序很快将需使用比 IaaS 虚拟机更先进的平台。这主要是为了实现自动缩放、获得更高伸缩能力，并进一步改善部署、升级、版本控制、回滚以及运行状况监视过程中的管理体验。为此可使用 Microsoft Azure 云提供的编排引擎分布式系统平台：Azure Service Fabric，同时也可以在本地甚至其他云平台中实现类似的目标。

很多将现有单体式应用程序平移至容器的组织，这样做主要出于两个原因：

- 降低成本，例如整合或减少了现有硬件数量，或可以提高应用程序运行密度。
- 在开发和运维人员之间建立一致的部署规程。

对于降低成本的诉求很好理解，这可能是所有组织的目标。一致的部署往往很难评估，但其实同样重要。一致的部署规程使得开发者可以自由选择最适合的技术，随后运维团队可以通过同样的方式部署并管理应用程序，这种方式可以缓解运维团队面对不同技术，却只能使用特定技术的复杂性。最终，每个应用程序都被容器化到自包含的部署镜像中。

一些组织会继续通过逐渐添加微服务的方式实现现代化（云优化和云原生应用程序）。但也有很多组织会止步于此（云 DevOps 就绪）。如图 4-8 所示，这些组织不打算迁移至微服务架构，主要原因在于可能并不需要。此时他们通过使用容器外加 Service Fabric 已经获得了所需的收益：部署、升级、版本控制、回滚以及运行状况监视的完善管理体验。

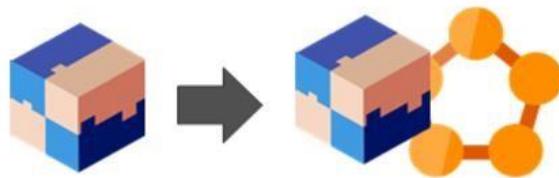


图 4-8：将应用程序平移至 Service Fabric

使用 Service Fabric 的重点在于重用现有代码并简化平移。因此用户可以使用 Windows 容器迁移现有的.NET Framework 应用程序，将其部署至 Service Fabric。随后即可陆续增加新的微服务，更容易将现代化继续深入下去。

Service Fabric 与其他编排引擎相比，最大的特点在于 Service Fabric 对运行基于 Windows 的应用程序和服务方面极为成熟。微软多年来一直通过 Service Fabric 运行各种基于 Windows 的服务和应用程序，甚至包含第一方和关键业务产品。这也是面向 Windows 容器正式上市的首个编排引擎（2017 年 5 月）。诸如 Kubernetes、DC/OS 和 Docker Swarm 等其他编排引擎在 Linux 世界中更成熟，但面对基于 Windows 的应用程序以及 Windows 容器，成熟度方面相比 Service Fabric 还有所欠缺。

Service Fabric 的最终目标是通过微服务的方式降低构建应用程序工作的复杂性。对于某些类型的应用程序，这也是用户最终的目标：避免重新设计产生的高昂成本。用户可以从小处着手，按需缩放，逐渐废弃旧的服务并加入新的服务，随着客户的使用逐渐演进自己的应用程序。当然，为了让微服务更适合更多开发者，还有很多其他问题有待解决，如果用户目前只是使用 Windows 容器平移现有应用程序，但计划以后增加基于微服务的容器，那么依然适合选择 Service Fabric。

何时将 Windows 容器部署至 Azure 容器服务（例如 Kubernetes）

Azure 容器服务针对流行开源工具的配置和 Azure 特有技术进行了深入优化，用户可以通过这样开放式解决方案在容器以及应用程序配置方面获得方便的可移植能力，只需决定规模、主机数量和编排引擎工具即可。Azure 容器服务可以代为管理基础架构。

如果用户已经在使用开源编排引擎，例如 Kubernetes、Docker Swarm 或 DC/OS，无需改变现有管理实践即可将工作负载迁移至云。随后可使用自己已经熟悉的应用程序管理工具，并通过标准的 API 终结点连接至所选的编排引擎。

如果用户已经在使用 Linux Docker 容器，所有这些编排引擎工具可提供成熟的支持，但从 2017 年开始，这些工具也可以支持 Windows 容器（取决于编排引擎工具，具体时间有先后差别）。

例如 Kubernetes 对容器提供了原生支持（“一等公民”），因此用户可以通过 Kubernetes 高效可靠地运行 Windows 容器（目前为预览版，2017 年秋季正式发布）。

构建云就绪的弹性服务：适应云端的短暂故障

弹性是指从故障中恢复正常运行的能力。弹性的目的不在于避免故障，而在于承认并接受故障时有发生，但遇到故障后可在不停机并且不丢失数据的情况下顺利恢复。弹性的目标在于遇到故障后将应用程序恢复至正常运行状态。

应用程序至少实现了基于软件的弹性模型（而非基于硬件的模型）后，便可认为这个应用程序已经云就绪。云应用程序必须能够适应最终肯定会遇到的局部故障。如果需要对可能出现的局部故障获得弹性，就必须妥善设计应用程序，甚至重构部分组件。在设计上必须能处理诸如网络和节点的暂时故障，或云端虚拟机崩溃等情况。甚至将容器移动到同一个编排工具集群中的不同节点，也可能导致应用程序遇到短暂的间歇性故障。

应对局部故障

在云端应用程序中，始终存在局部故障的风险。例如，某个网站实例或容器可能故障，在短时间内不可用或不响应，甚至某个虚拟机或服务器可能崩溃。

由于客户端和服务器是相互独立的进程，某个服务可能无法及时响应客户端的请求。服务可能已经超负荷运行，请求的响应速度极慢，或者可能因为网络问题在短时间内无法访问。

例如，假设某个单体式.NET 应用程序需要访问 Azure SQL 数据库。如果 Azure SQL 数据库或任何其他第三方服务在短时间内无法响应（例如 Azure SQL 数据库可能被移动到不同的节点或服务器，会在几秒钟内无法响应），此时当用户试图执行任何操作时，应用程序可能会在那一刻崩溃或抛出异常。

使用 HTTP 服务的应用也可能遇到类似情况。云中的网络或服务本身可能会在短时间内不可用，产生短暂的故障。

类似图 4-9 所示的弹性应用程序应实现类似“包含指数退避（Exponential backoff）重试”的技术，借此让应用程序有机会处理响应过程中短暂的故障。此外还可以在应用程序中使用“断路器（Circuit breaker）”，在遇到长期故障时，断路器可以阻止应用程序尝试访问资源。通过使用断路器，即可避免应用程序引发面向自身的拒绝服务“攻击”。

云应用程序之间的弹性通信

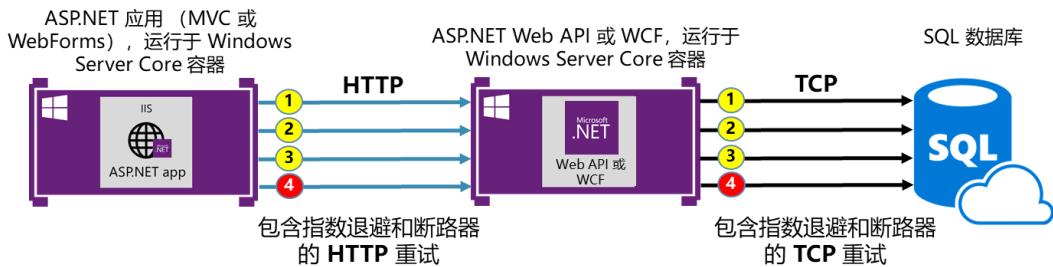


图 4-9：通过包含指数退避的重试应对局部故障

这些技术可同时运用于 HTTP 资源和数据库资源。图 4-9 所示的应用程序基于一种三层架构，因此需要在服务层（HTTP）和数据层（TCP）使用这些技术。在单体式应用程序中，由于除了数据库之外只使用了一个应用层（无其他服务或微服务），仅在数据库连接的层面上处理短暂的故障就足够了，这种情况下只需要对数据库连接进行必要的配置即可。

如果要对数据库访问实现弹性通信，取决于所用的.NET 版本，这个过程可能相当简单（例如，如果使用 [Entity Framework 6 或后续版本](#)，只需要配置数据库连接），或者可能需要其他库，如 [Transient Fault Handling Application Block](#)（针对老版本.NET），甚至使用自行实现的库。

在实现 HTTP 重试和断路器时，建议为.NET 使用 [Polly](#) 库，该库主要面向.NET 4.0、.NET 4.5 和.NET Standard 1.1，并可支持.NET Core。

有关在云中处理局部故障的具体策略和实现，请参阅下列资源。

参考资源

- **实现弹性通信以处理局部故障**
<https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/implement-resilient-applications/partial-failure-strategies>
- **Entity Framework 弹性连接和重试逻辑（6.0 和后续版本）**
[https://msdn.microsoft.com/en-us/library/dn456835\(v=vs.113\).aspx](https://msdn.microsoft.com/en-us/library/dn456835(v=vs.113).aspx)
- **Transient Fault Handling Application Block**
[https://msdn.microsoft.com/en-us/library/hh680934\(v=pandp.50\).aspx](https://msdn.microsoft.com/en-us/library/hh680934(v=pandp.50).aspx)
- **适用于弹性 HTTP 通信的 Polly 库**
<https://github.com/App-vNext/Polly>

借助监视和遥测实现应用的现代化

在生产环境中运行应用程序时，需要密切关注应用程序的性能情况。性能是否足够高？用户是否遇到错误，或应用程序本身是否稳定可靠？我们需要丰富的性能监视、强大的警报，以及仪表板，以确保应用程序的运行和性能符合预期。此外还需要能快速了解是否存在故障，判断受影响的用户数量，通过根源分析快速找到并修复问题。

使用 Application Insights 监视应用程序

Application Insights 是一种可扩展的 Application Performance Management (APM) 服务，主要面向需要使用多种平台的 Web 开发者。通过该服务可实时监视 Web 应用程序。Application Insights 可自动检测性能异常，并通过强大的分析工具帮助我们诊断问题，理解用户在通过我们的应用做些什么。Application Insights 在设计上可以帮助我们持续改善性能与可用性，适用于在本地或云中托管，基于.NET、Node.js 和 J2EE 等不同平台的应用。Application Insights 可与 DevOps 流程集成，并针对各种开发工具提供了连接点。

图 4-10 展示了使用 Application Insights 监视应用程序，并通过仪表板提供见解的范例。

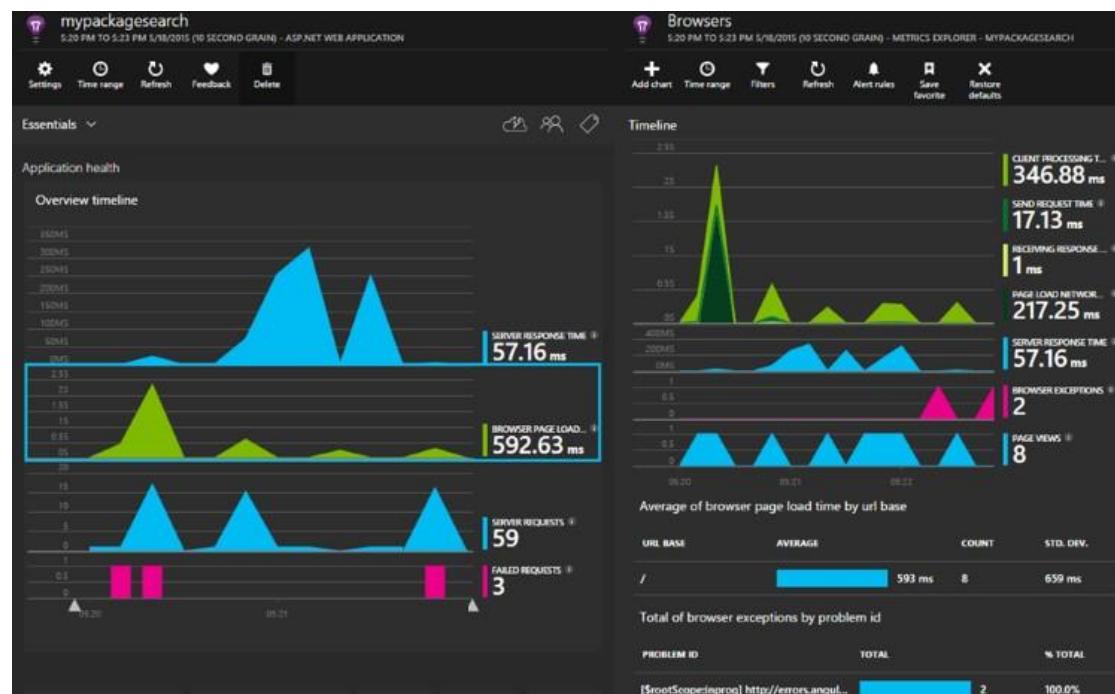


图 4-10：Application Insights 监视仪表板

使用 Log Analytics 及容器监视解决方案监视 Docker 基础架构

[Azure Log Analytics](#) 是 Microsoft Azure 整体监视解决方案的一部分，同时也已作为一项服务包含在 [Operations Management Suite \(OMS\)](#) 中。Log Analytics 可监视云环境和本地环境（使用 OMS 监视本地环境），借此保障可用性与性能。它可以收集云端和本地环境中不同资源生成的数据，同时可结合其他监视工具生成的数据执行跨越不同数据源的分析。

对于和 Azure 基础架构有关的日志，Log Analytics 作为一项 Azure 服务，可以获取来自其他 Azure 服务（通过 [Azure Monitor](#)）、Azure 虚拟机、Docker 容器，以及本地或云端其他基础架构的日志和指标（Metric）数据。Log Analytics 提供了灵活的日志搜索功能，并可基于这些数据提供拆箱即用的分析能力。该服务可通过丰富的工具帮助我们分析不同来源的数

据，针对所有日志执行复杂的查询，并可根据预设条件主动发出警告。我们甚至可以将自定义数据收集到 Log Analytics 仓库中集中存储，并统一查询和可视化。通过 Log Analytics 内建的解决方案即可针对基础架构的安全和功能运行情况立即获得直观的见解。

我们可以通过 OMS 门户或 Azure 门户访问 Log Analytics，这些门户可通过所有浏览器访问，并能借此访问配置选项，通过不同工具分析收集的数据，进而做出反应。

Log Analytics 提供的[容器监视解决方案](#)可以帮助我们在一个位置查看并管理 Docker 和 Windows 容器主机。该解决方案可显示运行中的容器，容器运行的镜像，以及容器的运行位置。此外还可以看到详细的审核信息，包括配合容器使用的命令。

通过查看并搜索集中存储的日志，无需远程查看 Docker 或 Windows 主机，即可直接对容器问题排错。在这里可以找到可能有问题，消耗过多主机资源的容器。此外还可以集中查看容器的 CPU、内存、存储和网络用量与性能信息。对于运行 Windows 的计算机，还可集中存储并对比来自 Windows Server、Hyper-V 和 Docker 容器的日志。该解决方案支持下列容器编排引擎：

- Docker Swarm
- DC/OS
- Kubernetes
- Service Fabric
- Red Hat OpenShift

图 4-11 展示了不同容器主机和代理与 OMS 之间的关系。

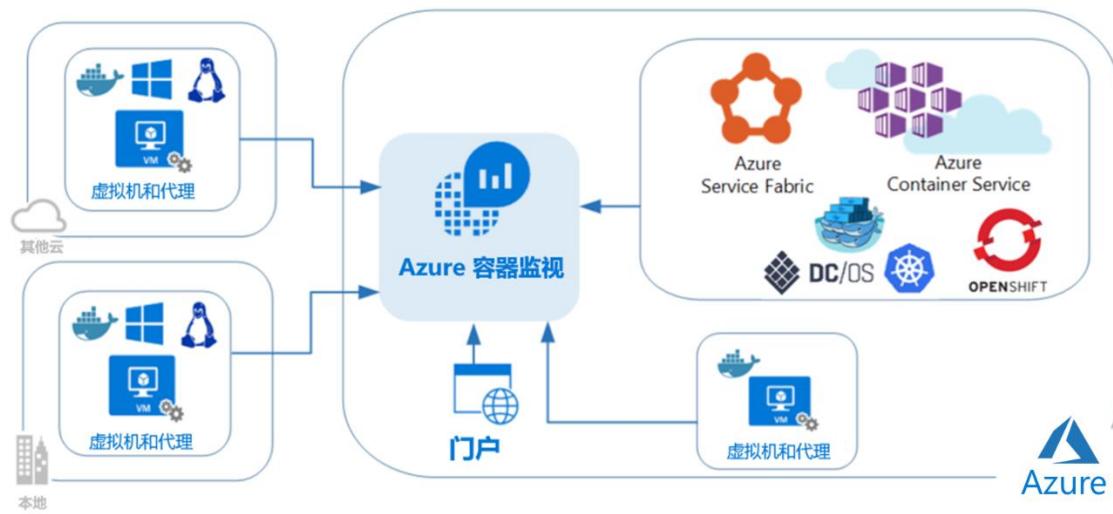


图 4-11：Log Analytics 容器监视解决方案

我们可以通过 Log Analytics 容器监视解决方案：

- 在一个位置查看所有容器主机的信息。

- 了解运行中的容器，所运行的镜像，以及容器的运行位置。
- 查看针对特定容器所执行操作的审核轨迹。
- 无需远程登录 Docker 主机即可查看并搜索集中存储的日志进而排错。
- 查找可能影响其他容器，或者过多消耗了主机资源的容器。
- 集中查看容器的 CPU、内存、存储和网络用量，以及性能信息。

参考资源

- **Microsoft Azure 监视功能概述**
<https://docs.microsoft.com/en-us/azure/monitoring-and-diagnostics/monitoring-overview>
- **Application Insights 是什么？**
<https://docs.microsoft.com/en-us/azure/application-insights/app-insights-overview>
- **Log Analytics 是什么？**
<https://docs.microsoft.com/en-us/azure/log-analytics/log-analytics-overview>
- **Log Analytics 中的容器监视解决方案**
<https://docs.microsoft.com/en-us/azure/log-analytics/log-analytics-containers>
- **Azure Monitor 概述**
<https://docs.microsoft.com/en-us/azure/monitoring-and-diagnostics/monitoring-overview-azure-monitor>
- **Operations Management Suite (OMS) 是什么？**
<https://docs.microsoft.com/en-us/azure/operations-management-suite/operations-management-suite-overview>
- **使用 MOS 监视 Service Fabric 中的 Windows Server 容器**
<https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-diagnostics-containers-windowsserver>

使用 CI/CD 流程和云端 DevOps 工具实现应用生命周期的现代化

当今的业务要求企业必须以足够快的速度创新才能维持竞争力。高质量、现代化应用程序的交付需要具备 DevOps 工具和相关流程，这些条件已成为持续不断创新的关键。借助适合的 DevOps 工具，开发者可以更流畅地持续部署，更快速地将创新的应用程序交付给客户。

虽然持续集成与持续部署的实践早已创建，但面对容器依然有新问题需要考虑，对于使用多种容器的应用程序更是如此。

Visual Studio Team Services 支持通过持续集成与持续部署的方式将使用多种容器的应用程序部署到不同环境，为此可使用下列官方推荐的 Team Services 部署任务：

- [部署到独立的 Docker 主机虚拟机](#) (Linux 或 Windows Server 2016 及后续版本)

- [部署到 Service Fabric](#)
- [部署到 Azure 容器服务 – Kubernetes](#)

此外也可以使用 Team Services 基于脚本的任务部署到 [Docker Swarm](#) 或 DC/OS。

为了进一步提高部署敏捷性，这些工具还为容器工作负载提供了卓越的“开发到测试再到生产”部署体验，并可灵活选择开发和 CI/CD 解决方案。

图 4-12 展示了部署到 Azure 容器服务中 Kubernetes 集群的持续部署流程。

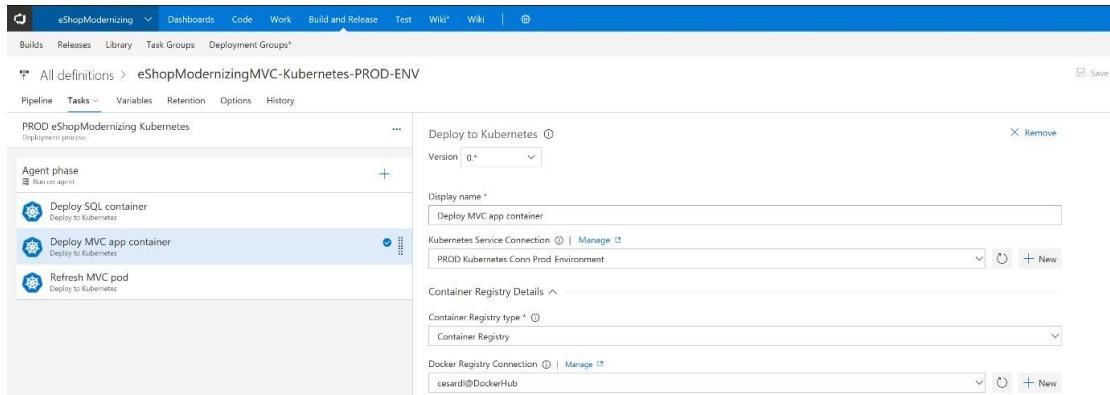


图 4-12：Visual Studio Team Services 持续部署流程，可部署至 Kubernetes 群集

迁移至混合云场景

由于制度或策略要求，一些组织或企业无法将自己的某些应用程序迁移至诸如 Microsoft Azure 或其他供应商的公有云平台。然而通过在公有云中运行一部分应用程序，并在本地环境运行另一部分应用程序，任何组织都可从中获益。但由于公有云和本地环境使用了不同的平台和技术，这样的杂合环境可能导致管理工作变得极为复杂。

微软提供了最佳混合云解决方案，借此可对本地的现有资产和公有云中的资产进行优化，同时确保整个 Azure 混合云的一致。通过这种方式可以充分利用现有技能，通过灵活且一致的方法构建同时在云端和本地运行的应用，当然这一切都离不开 Azure Stack (本地) 和 Azure (公有云)。

在安全方面，可以跨越混合云集中管理并保障安全性。从数据中心到云端，所有资产均可集中控制，一次登录即可访问本地和云端的应用。为此只需要将 Active Directory 扩展到混合云，并使用相应的身份认证机制即可。

最后，我们还可以无缝地分发和分析数据，针对云端和本地资产使用相同的查询语言，通过在 Azure 中运行分析和深度学习技术进行数据增强，而无需考虑数据的来源是什么。

Azure Stack

Azure Stack 是一种混合云平台，可帮助我们通过组织自有数据中心交付 Azure 服务。Azure Stack 按照设计可为不同的应用程序现代化新场景提供支持，例如边缘计算和断网环境，此

外也可帮助我们更好地满足安全与合规方面的需求。

图 4-13 展示了微软提供的真正混合云平台的概况。

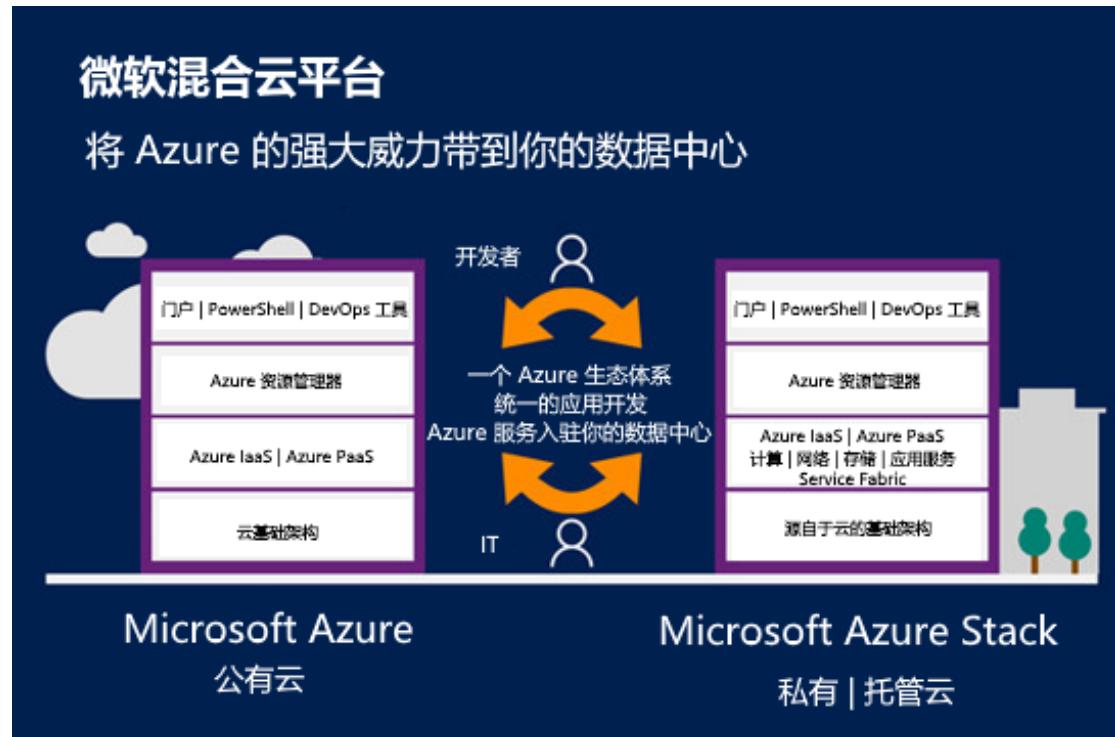


图 4-13：通过 Azure Stack 和 Azure 打造的微软混合云平台

为满足不同需求，Azure Stack 提供了两种部署选项：

- Azure Stack 集成式系统
- Azure Stack Development Kit

Azure Stack 集成式系统

Azure Stack 集成式系统由微软与硬件合作伙伴联手提供。通过这种合作打造的解决方案可提供云节奏 (cloud-paced) 的创新以及更简化的管理。由于 Azure Stack 是以集成式的软硬件方式联合提供，用户可获得足够的灵活性和控制力，同时继续获得云平台源源不断的创新。Azure Stack 集成式系统的规模和包含 4-12 个节点，并可得到硬件合作伙伴与微软的联合支持。用户可通过 Azure Stack 集成式系统为生产环境中的工作负载打造全新使用场景。

Azure Stack Development Kit

Microsoft Azure Stack Development Kit 是一种单节点部署的 Azure Stack，可用来评估或学习 Azure Stack。用户也可将 Azure Stack Development Kit 用作开发环境，使用与 Azure 一致的 API 和工具进行开发。Azure Stack Development Kit 不能用于生产环境。

参考资源

- Azure 混合云

<https://www.microsoft.com/en-us/cloud-platform/hybrid-cloud>

- **Azure Stack**

<https://azure.microsoft.com/en-us/overview/azure-stack/>

- **适用于 Windows 容器的 Active Directory 服务帐户**

<https://docs.microsoft.com/en-us/virtualization/windowscontainers/manage-containers/manage-serviceaccounts>

- **创建支持 Active Directory 的容器**

<https://blogs.msdn.microsoft.com/containerstuff/2017/01/30/create-a-container-with-active-directory-support/>

- **Azure 混合权益许可**

<https://azure.microsoft.com/en-us/pricing/hybrid-use-benefit/>

第 5 章：步骤和技术准备概述

受制于篇幅，我们已将详细的技术文档和完整的步骤描述发布到 GitHub 代码库。本章所涉及的步骤相应的在线文档涵盖了基于 Windows 容器的不同环境循序渐进的搭建步骤，以及部署到 Azure 的详细做法。

下文将简要介绍每个场景的大致步骤，主要目标和愿景，并会简要描述所涉及的任务。我们可以通过 GitHub 代码库 eShopModernizing 应用的维基页面查看详情：

<https://github.com/dotnet-architecture/eShopModernizing/wiki>。

技术步骤清单

下列场景概述提供了完善的技术指南，可以帮助我们使用容器平移范例应用程序，并使用不同的 Azure 服务进行部署以最终实现应用的迁移。

每个场景均使用了 eShopLegacy 和 eShopModernizing 这两个范例应用，这些应用已发布至 GitHub：<https://github.com/dotnet-architecture/eShopModernizing>。

- **eShop 遗留应用概览**
- **使用 Windows 容器实现现有.NET 应用程序的容器化**
- **将基于 Windows 容器的应用部署至 Azure 虚拟机**
- **将基于 Windows 容器的应用部署至 Azure 容器服务中的 Kubernetes 集群**
- **将基于 Windows 容器的应用部署至 Azure Service Fabric**

场景 1：eShop 遗留应用概览

可用的技术步骤

完整的技术步骤可访问 eShopModernizing 在 GitHub 上的代码库维基页：

<https://github.com/dotnet-architecture/eShopModernizing/wiki/01.-Tour-on-eShopModernizing-apps-implementation-code>

概述

本场景将简要了解两个遗留范例应用程序最初的实现方式。这两个范例应用均采用了单体式架构，使用传统的 ASP.NET 创建。其中一个应用程序基于 ASP.NET 4.x MVC，另一个基于 ASP.NET 4.x Web Forms。两个应用程序都已发布至 [eShopModernizing GitHub 代码库](#)。

这两个范例应用均可容器化，具体方法与传统 [Windows Communication Foundation \(WCF\)](#) 应用的容器化方法类似，随后即可作为桌面应用程序使用。具体范例可参阅 [eShopModernizingWCFWinForms](#)。

目标

该场景主要目标在于熟悉这些应用，及其代码与配置。为了方便测试，这些应用通过配置可直接生成并使用样本数据，无需使用 SQL 数据库。可选配置主要基于通过依赖注入实现解耦。

场景

图 5-1 展示了遗留应用程序最初简单的架构场景。

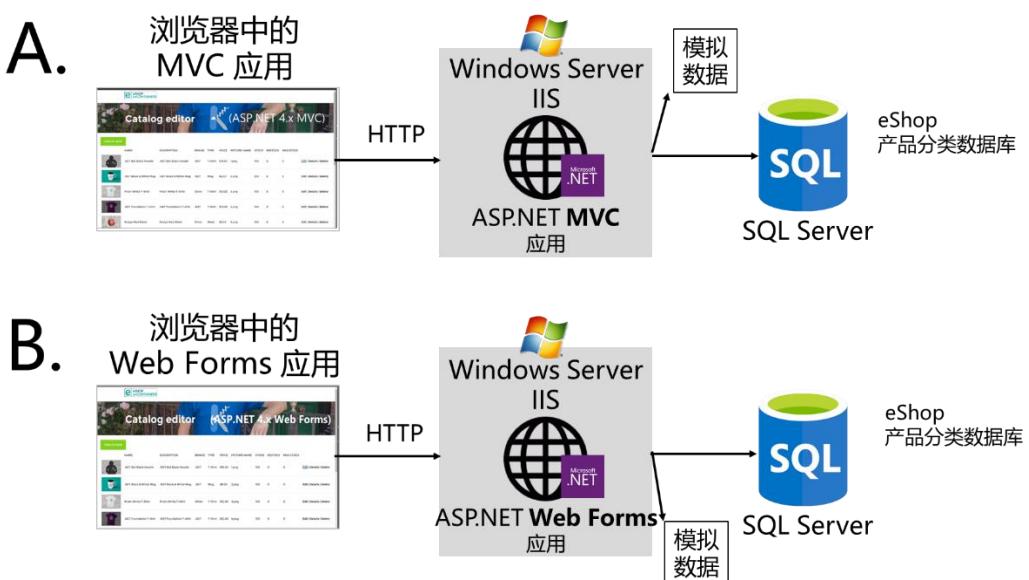


图 5-1：遗留应用程序最初简单的架构场景

从业务的角度来看，这两个应用提供了相同的分类管理功能。eShop 企业团队成员可以使用该应用查看并编辑产品分类。图 5-2 展示了最初版本的应用截图。

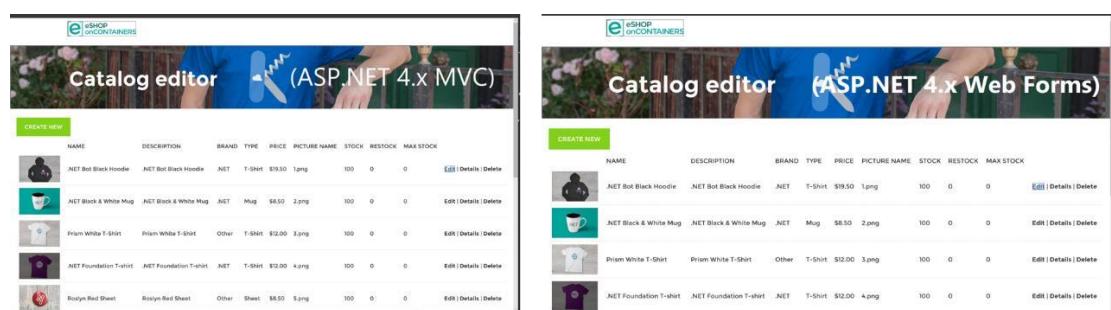


图 5-2：ASP.NET MVC 和 ASP.NET Web Forms 应用程序（现有/遗留技术）

这些都是用于浏览和修改目录项的 Web 应用程序。两个应用都提供了相同的业务/功能，因此可以方便地进行比较。使用 ASP.NET MVC 和 ASP.NET Web Forms 框架创建的应用也可以通过类似的过程实现现代化。

ASP.NET 4.x 或更早版本（无论 MVC 或 Web Forms）的依赖性意味着，除非使用 ASP.NET

Core MVC 彻底重写，否则这些应用程序无法通过.NET Core 运行。这也证明了如果不想重构或重写代码，并继续使用相同的.NET 技术和相同的代码，可以选择将现有的应用程序容器化。下文将介绍如何在不改动遗留代码的前提下通过容器运行此类应用程序。

收益

这个场景的收益很简单：根据依赖项注入熟悉代码以及应用程序的配置。随后即可进行实验，通过这种方法将应用容器化，并部署到不同环境中。

下一步

通过 GitHub 维基页面进一步了解相关内容：

<https://github.com/dotnet-architecture/eShopModernizing/wiki/01.-Tour-on-eShopModernizing-apps-implementation-code>

场景 2：使用 Windows 容器实现现有.NET 应用程序的容器化

可用的技术步骤

完整的技术步骤可访问 eShopModernizing 在 GitHub 上的代码库维基页：

<https://github.com/dotnet-architecture/eShopModernizing/wiki/02.-How-to-containerized-the-.NET-Framework-web-apps-with-Windows-Containers-and-Docker>

概述

使用 Windows 容器改善基于 MVC、Web Forms 或 WCF 的现有.NET 应用程序在生产、开发和测试环境中的部署。

目标

该场景的目标在于介绍对现有.NET Framework 应用程序实现容器化的不同选项，借此可以：

- 使用 [Visual Studio 2017 Tools for Docker](#) (Visual Studio 2017 或后续版本) 实现应用程序容器化。
- 手工添加 [Dockerfile](#) 并使用 [Docker CLI](#) 实现应用程序的容器化。
- 使用 [Img2Docker](#) 工具 (Docker 提供的开源工具) 实现应用程序的容器化。

本场景侧重于 Visual Studio 2017 Tools for Docker 方法，但在 Dockerfile 的使用方面，另外两种方法也基本类似。

场景

图 5-3 展示了对 eShop 遗留应用程序实现容器化的场景。

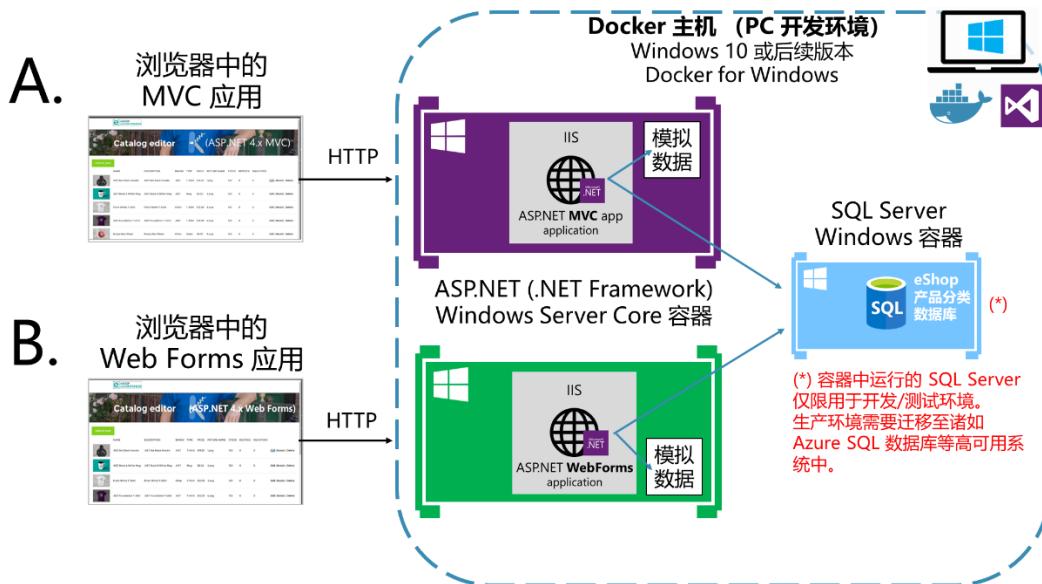


图 5-3：开发环境中容器化应用程序架构的简化示意图

收益

在容器中运行单体式应用程序可以获得很多收益。首先，只需为应用程序创建一个镜像。随后每次部署均可在相同环境中运行，每个容器可使用相同版本的操作系统，装有相同版本的依赖项，使用了相同版本的.NET Framework，可使用相同流程构建。基本上通过一个 Docker 镜像即可控制应用程序的依赖项，并且依赖项可以与应用程序一起通过容器部署。

另一个收益在于，开发者可以通过 Windows 容器提供的一致环境运行自己的应用程序。由于版本不统一出现的问题可以彻底解决，绝对不会出现在暂存或生产环境中。当通过容器运行应用程序时，开发团队成员所用开发环境与生产环境是否有差异将无关紧要。

容器化应用程序的缩放曲线也更平坦。相比普通方式将应用程序部署到每台计算机，容器化的应用程序可以让我们在一个虚拟机或物理机中（使用容器）运行更多应用程序和服务实例。进而可以获得更高密度，资源消耗更少，在使用 Kubernetes 或 Service Fabric 等编排引擎时更是如此。

理想情况下，容器化并不需要改动应用程序代码（C#）。大部分情况下，只需要一个 Docker 部署元数据文件（Dockerfile 和 Docker Compose 文件）。

下一步

通过 GitHub 维基页进一步了解相关内容：

<https://github.com/dotnet-architecture/eShopModernizing/wiki/02.-How-to->

场景 3：将基于 Windows 容器的应用部署至 Azure 虚拟机

可用的技术步骤

完整的技术步骤可访问 eShopModernizing 在 GitHub 上的代码库维基页：

[https://github.com/dotnet-architecture/eShopModernizing/wiki/03.-How-to-deploy-your-Windows-Containers-based-app-into-Azure-VMs-\(Including-CI-CD\)](https://github.com/dotnet-architecture/eShopModernizing/wiki/03.-How-to-deploy-your-Windows-Containers-based-app-into-Azure-VMs-(Including-CI-CD))

概述

将 Docker 主机部署到运行 Windows Server 2016 的 Azure 虚拟机，可供我们快速搭建开发/测试/暂存环境，同时有助于为测试人员或业务用户提供一个应用验证的共用场所。虚拟机也可以成为实用的 IaaS 生产环境。

目标

本场景的目标在于展示将 Windows 容器部署到运行 Windows Server 2016 或后续版本的 Azure 虚拟机时，可用的不同选项。

场景

本节涉及多种不同场景。

场景 A：从开发计算机通过 Docker Engine 连接部署至 Azure 虚拟机

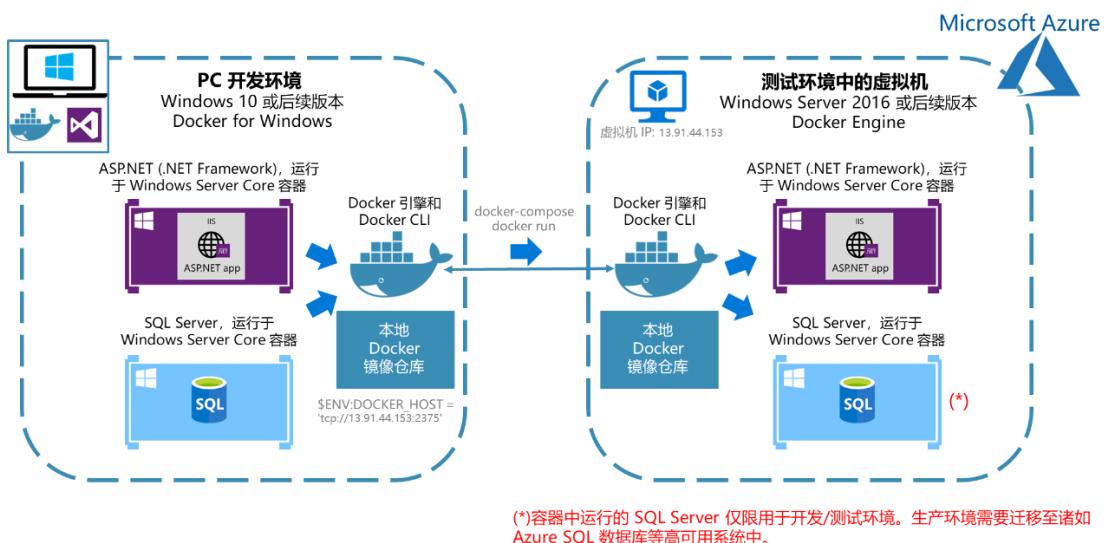


图 5-4：从开发机器通过 Docker Engine 连接部署至 Azure 虚拟机

场景 B：通过 Docker 注册表部署至 Azure 虚拟机

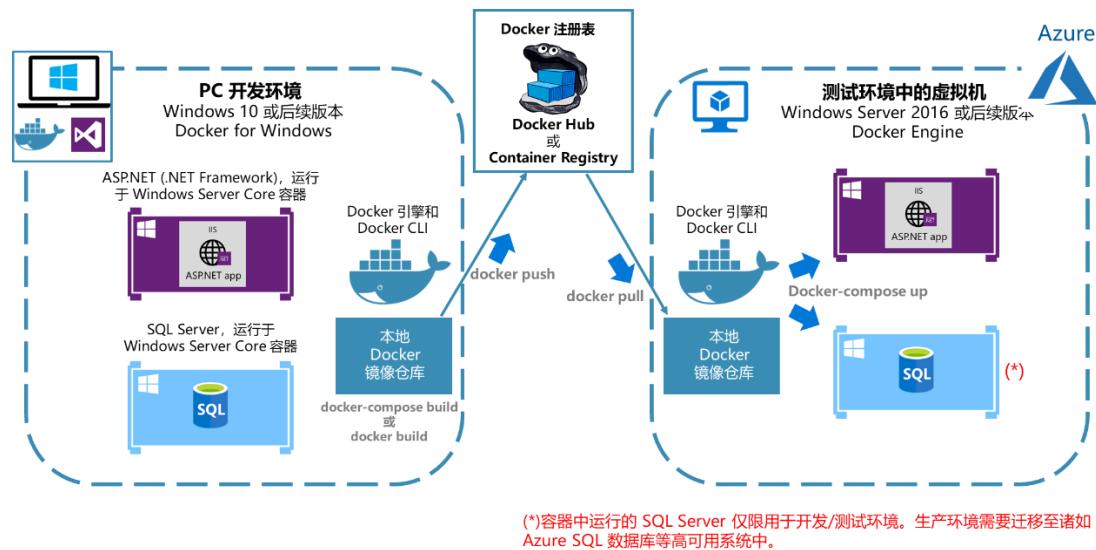


图 5-5：通过 Docker 注册表部署至 Azure 虚拟机

场景 C：在 Visual Studio Team Services 中通过 CI/CD 流程部署至 Azure 虚拟机

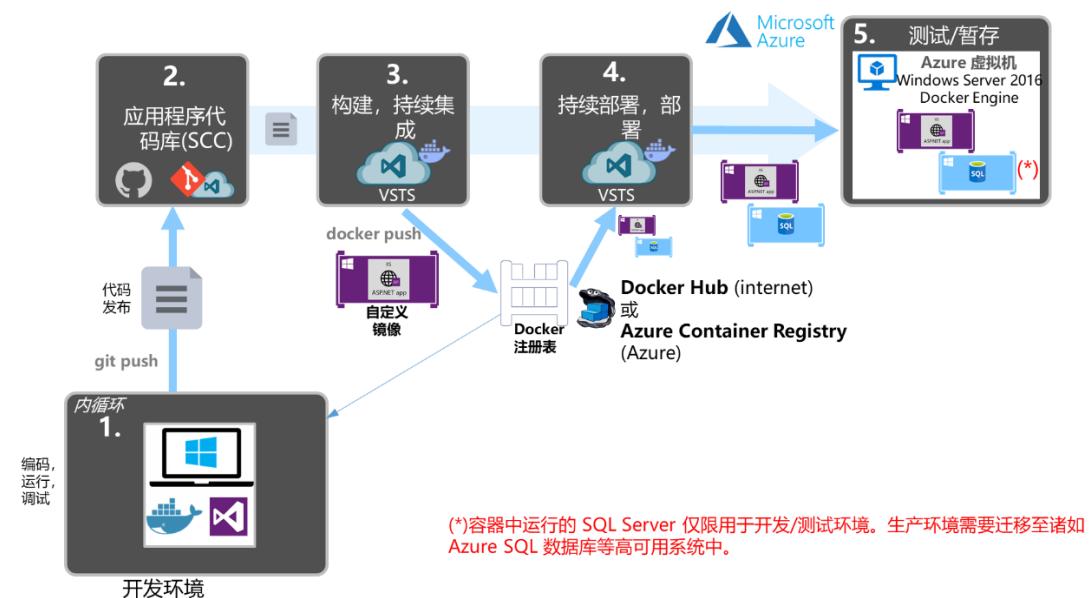


图 5-6：在 Visual Studio Team Services 中通过 CI/CD 流程部署至 Azure 虚拟机

适用于 Windows 容器的 Azure 虚拟机

适用于 Windows 容器的 Azure 虚拟机其实就是一个运行 Windows Server 2016、Windows 10 或后续版本，并配置了 Docker Engine 的虚拟机。大部分情况下可使用运行 Windows Server 2016 的 Azure 虚拟机。

Azure 目前提供了名为 **Windows Server 2016 with Containers** 的虚拟机。我们可以使用该虚拟机，通过 Windows Server Core 或 Windows Nano Server 尝试 Windows Server 容器的新功能。容器操作系统镜像已安装，随后即可在此虚拟机上部署并运行 Docker 应用。

收益

虽然 Windows 容器可部署到本地 Windows Server 2016 虚拟机中，但如果部署到 Azure 虚拟机，可以更快速轻松地上手，并获得立即可用的 Windows Server 容器虚拟机。此外还可获得一个可供测试人员随时访问的通用在线位置，并能通过 Azure 虚拟机规模集轻松扩展。

下一步

通过 GitHub 维基页进一步了解相关内容：

[https://github.com/dotnet-architecture/eShopModernizing/wiki/03.-How-to-deploy-your-Windows-Containers-based-app-into-Azure-VMs-\(Including-CI-CD\)](https://github.com/dotnet-architecture/eShopModernizing/wiki/03.-How-to-deploy-your-Windows-Containers-based-app-into-Azure-VMs-(Including-CI-CD))

场景 4：将基于 Windows 容器的应用部署至 Azure 容器服务中的 Kubernetes 集群

可用的技术步骤

完整的技术步骤可访问 eShopModernizing 在 GitHub 上的代码库维基页：

[https://github.com/dotnet-architecture/eShopModernizing/wiki/04.-How-to-deploy-your-Windows-Containers-based-apps-into-Kubernetes-in-Azure-Container-Service-\(Including-C-CD\)](https://github.com/dotnet-architecture/eShopModernizing/wiki/04.-How-to-deploy-your-Windows-Containers-based-apps-into-Kubernetes-in-Azure-Container-Service-(Including-C-CD))

概述

基于 Windows 容器的应用程序很快将需要通过平台演进为比 IaaS 虚拟机更先进的场景。借此可以轻松实现更高伸缩性，并更好地实现自动化伸缩，同时可进一步改善自动化部署和版本控制机制。为实现该目标，可使用 [Azure 容器服务](#) 提供的编排引擎 [Kubernetes](#)。

目标

该场景目标在于了解如何将基于 Windows 容器的应用程序部署到 Azure 容器服务中的 Kubernetes 集群（也叫做 K8s）。从零开始部署到 Kubernetes 集群需要两个步骤：

1. 将 Kubernetes 集群部署至 Azure 容器服务。
2. 将应用程序和相关资源部署至 Kubernetes 集群。

场景

场景 A：从开发环境直接部署至 Kubernetes 集群

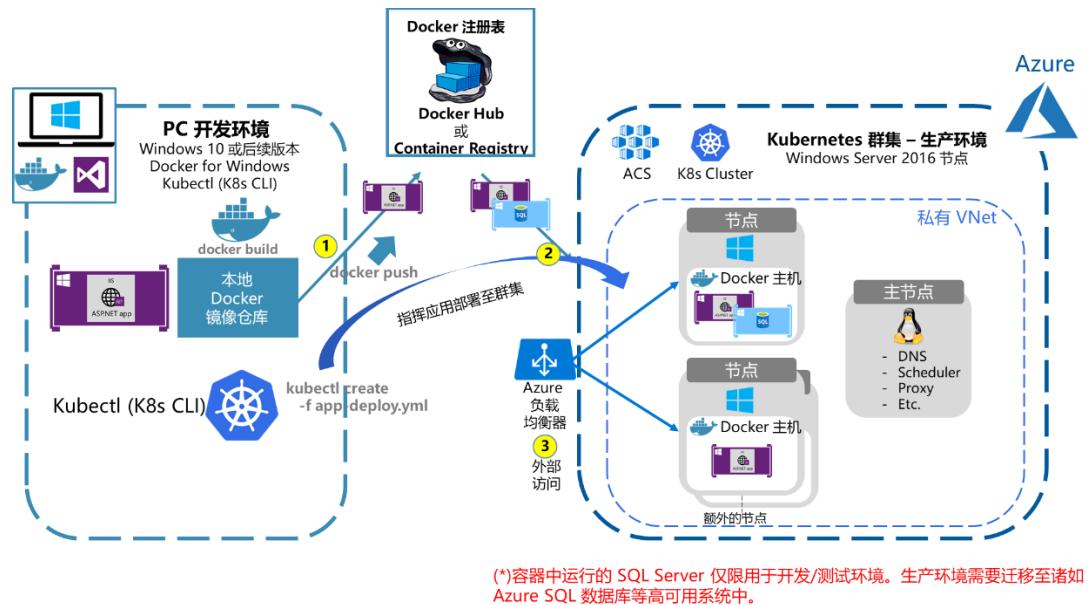


图 5-7：从开发环境直接部署至 Kubernetes 集群

场景 B：在 Team Services 中通过 CI/CD 流程部署至 Kubernetes 集群

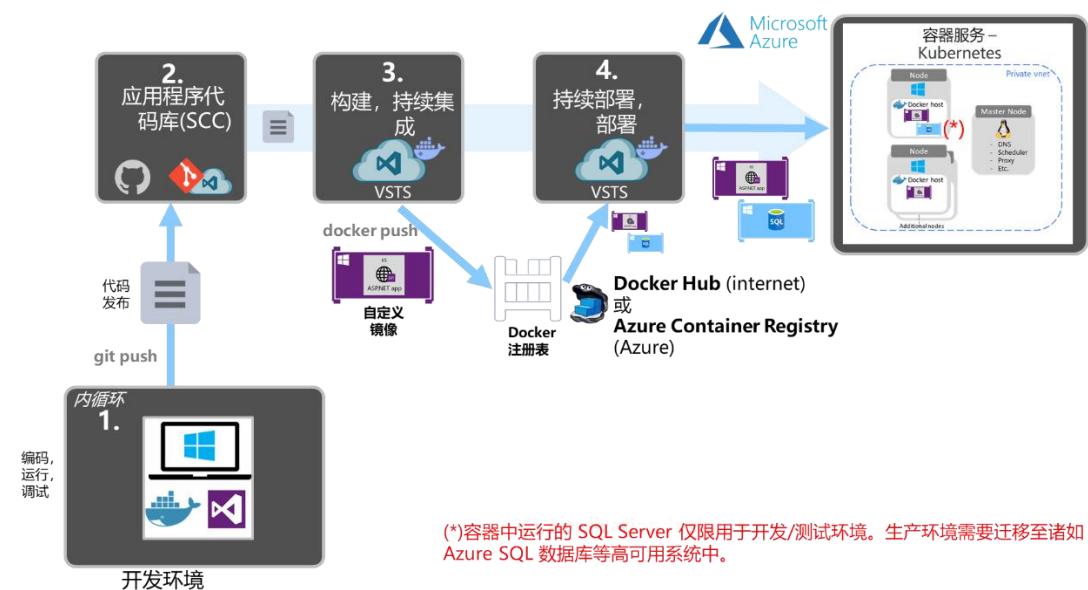


图 5-8：在 Team Services 中通过 CI/CD 流程部署至 Kubernetes 集群

收益

部署至 Kubernetes 集群可获得多项收益。最大的收益在于可以获得一个生产就绪的环境，

并根据要使用的容器实例数量对应用程序进行横向扩展（现有节点的内部伸缩），并可根据集群中节点或虚拟机的数量进行扩展（集群的全局伸缩）。

Azure 容器服务针对流行的开源工具和 Azure 专有技术进行了深度优化，用户可获得一套无论容器或应用程序配置均可灵活移植的开放式解决方案。选择虚拟机大小，节点数量以及编排引擎工具即可，其他事情可交由容器服务处理。

在 Kubernetes 的帮助下，开发者可以不再关注物理机和虚拟机，而是转为考虑以容器为中心的基础架构的规划工作，进而可以获得下列能力：

- 使用多容器应用程序
- 复制容器实例并横向自动伸缩
- 命名和发现（例如内部 DNS）
- 负载均衡
- 滚动式更新
- 分布式密文密码（Secret）处理
- 应用程序运行状况检查

下一步

通过 GitHub 维基页进一步了解相关内容：

[https://github.com/dotnet-architecture/eShopModernizing/wiki/04.-How-to-deploy-your-Windows-Containers-based-apps-into-Kubernetes-in-Azure-Container-Service-\(Including-CI-CD\)](https://github.com/dotnet-architecture/eShopModernizing/wiki/04.-How-to-deploy-your-Windows-Containers-based-apps-into-Kubernetes-in-Azure-Container-Service-(Including-CI-CD))

场景 5：将基于 Windows 容器的应用部署至 Azure Service Fabric

可用的技术步骤

完整的技术步骤可访问 eShopModernizing 在 GitHub 上的代码库维基页：

[https://github.com/dotnet-architecture/eShopModernizing/wiki/05.-How-to-deploy-your-Windows-Containers-based-apps-into-Azure-Service-Fabric-\(Including-CI-CD\)](https://github.com/dotnet-architecture/eShopModernizing/wiki/05.-How-to-deploy-your-Windows-Containers-based-apps-into-Azure-Service-Fabric-(Including-CI-CD))

概述

基于 Windows 容器的应用程序很快将需要通过平台演进为比 IaaS 虚拟机更先进的场景。借此可以轻松实现更高伸缩性，并更好地实现自动化伸缩，同时可进一步改善自动化部署和版本控制机制。为实现该目标，可使用 Azure 云提供的编排引擎 - Azure Service Fabric，该服

务还可用于本地环境，甚至其他公有云平台。

目标

本场景的目标在于了解如何将基于 Windows 容器的应用程序部署至 Azure Service Fabric。从零开始部署至 Service Fabric 只需要两个步骤：

1. 在 Azure (或其他环境中) 部署 Service Fabric 集群。
2. 将应用程序和相关资源部署至 Service Fabric 集群。

场景

场景 A：从开发环境直接部署至 Service Fabric 集群

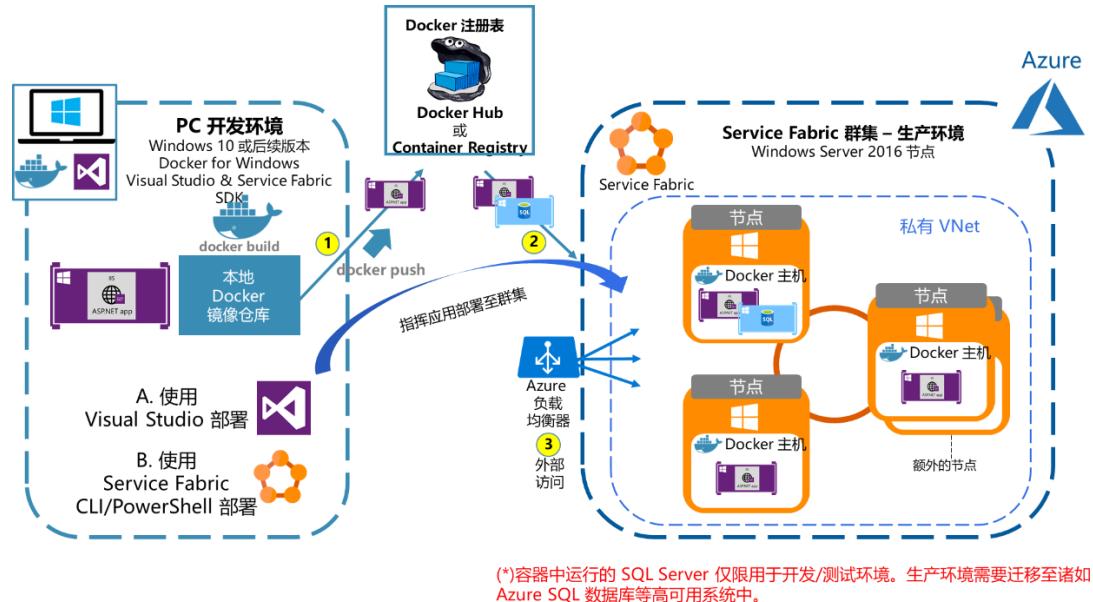


图 5-9：从开发环境直接部署至 Service Fabric 集群

场景 B：在 Team Services 中通过 CI/CD 流程部署至 Service Fabric 集群

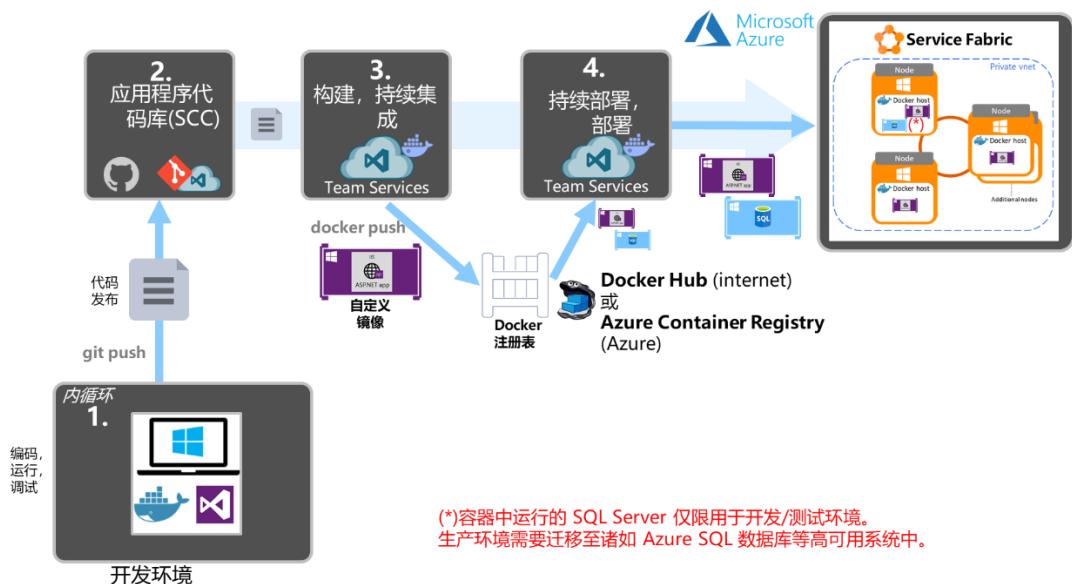


图 5-10：在 Team Services 中通过 CI/CD 流程部署至 Service Fabric 集群

收益

部署至 Service Fabric 集群所能获得的收益与使用 Kubernetes 的收益类似，但有一点差异：相比 Kubernetes，Service Fabric 对生产环境中运行的 Windows 应用程序提供了更成熟的支持，而 Kubernetes 对 Windows 容器的支持截止 2017 年初秋还处于预览阶段（对 Linux 来说，Kubernetes 是一种更成熟的环境）。

使用 Azure Service Fabric 的主要收益在于，可以获得生产就绪的环境，并根据要使用的容器实例数量对应用程序进行横向扩展（现有节点的内部伸缩），并可根据集群中节点或虚拟机的数量进行扩展（集群的全局伸缩）。

Azure Service Fabric 可以让容器以及应用程序配置获得可移植能力。我们可以在 Azure 中运行 Service Fabric 集群，或将其安装在本地数据中心，甚至可以将 Service Fabric 集群安装到其他云平台，例如 [Amazon AWS](#) 中。

在 Service Fabric 的帮助下，开发者可以不再关注物理机和虚拟机，而是转为考虑以容器为中心的基础架构的规划工作，进而可以获得下列能力：

- 使用多容器应用程序
- 复制容器实例并横向自动伸缩
- 命名和发现（例如内部 DNS）
- 负载均衡

- 滚动式更新
- 分布式密文密码 (Secret) 处理
- 应用程序运行状况检查

(相比其他编排引擎) Service Fabric 可提供下列专有能力：

- 通过 Reliable Services 应用程序模型提供有状态 (Stateful) 服务能力
- 通过 Reliable Actors 应用程序模型提供 Actors 模式
- 除了 Windows 或 Linux 容器，还可部署 Bare-bone 进程
- 高级滚动式更新和运行状况检查

下一步

通过 GitHub 维基页进一步了解相关内容：

[https://github.com/dotnet-architecture/eShopModernizing/wiki/05.-How-to-deploy-your-Windows-Containers-based-apps-into-Azure-Service-Fabric-\(Including-Cl-CD\)](https://github.com/dotnet-architecture/eShopModernizing/wiki/05.-How-to-deploy-your-Windows-Containers-based-apps-into-Azure-Service-Fabric-(Including-Cl-CD))

第6章：结论

主要结论

- 基于容器的解决方案最终可帮助用户降低成本。容器是一种解决部署问题的方案，这种技术可消除由于生产环境缺乏依赖项所造成的阻力。通过解决这些问题，可大幅促进开发/测试、DevOps 以及生产运维。
- Docker 正在成为容器行业的既定标准。Linux 和 Windows 生态的大部分重要供应商，包括微软均已支持 Docker。未来，该技术还将提供最完善和全面的环境，帮助用户使用 Windows 容器和 Azure 基础架构即服务实现现有.NET Framework 应用程序的现代化。Docker 容器已成为所有服务器应用程序和服务的标准部署单位。
- 在生产环境中，可以使用编排引擎(如 Service Fabric 或 Kubernetes)托管基于 Windows 容器的可伸缩应用程序。
- Azure 虚拟机托管的容器是一种在云中打造小规模开发测试环境最快速简单的方法。
- 如果要将现有应用程序使用的关系型数据库迁移至 Azure，Azure SQL 数据库托管实例将是推荐的“默认”选择。
- Visual Studio 2017 和 Image2Docker 等重要工具可以帮助我们着手使用 Windows 容器实现现有.NET 应用程序的现代化，并可加速上手过程。
- 在生产环境中使用容器化应用程序时，首先需要确立 DevOps 文化，并通过 DevOps 工具实现 CI/CD 流程，例如可使用 Visual Studio Team Services 或 Jenkins。
- 在使用 Windows 容器、云基础架构和 PaaS 服务为现有.NET Framework 应用程序实现现代化的过程中，Microsoft Azure 提供了全面且完善的环境。