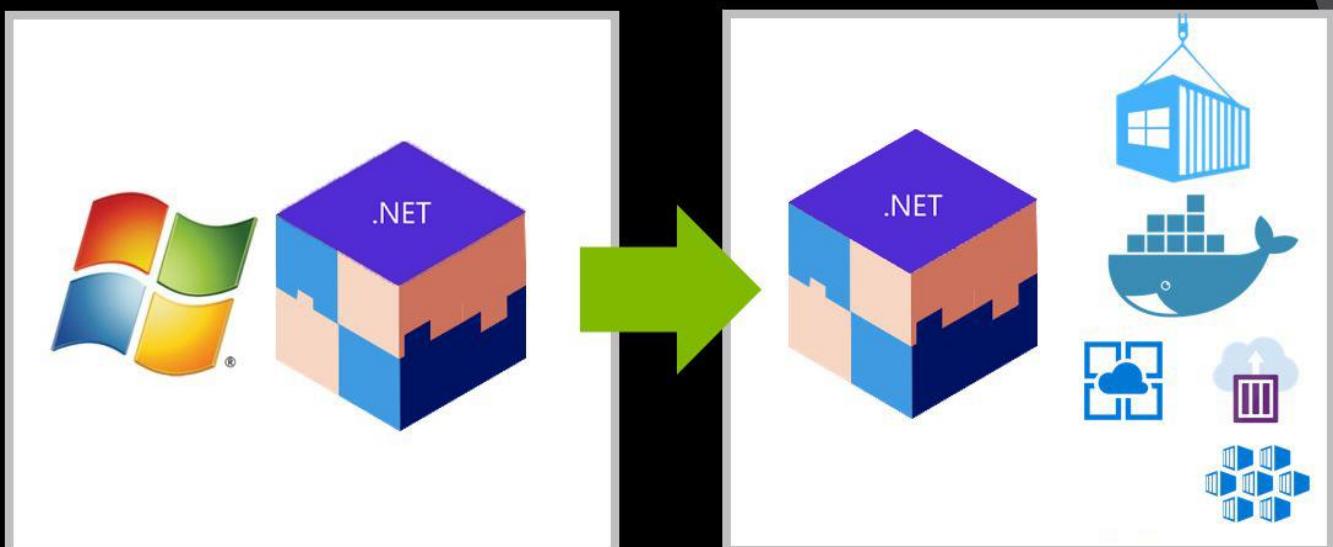


Modernize existing .NET applications with Azure cloud and Windows Containers



Cesar de la Torre
Microsoft Corp.



EDITION v6.0

Refer [changelog](#) for the book updates and community contributions.

PUBLISHED BY Microsoft Press and Microsoft DevDiv Divisions of Microsoft Corporation One Microsoft Way Redmond, Washington 98052-6399

Copyright © 2022 by Microsoft Corporation

All rights reserved. No part of the contents of this book may be reproduced in any form or by any means without the written permission of the publisher.

This book is available for free in the form of an electronic book (e-book) available through multiple channels at Microsoft such as <https://dot.net/architecture>.

If you have questions related to this book, email at dotnet-architecture-ebooks-feedback@service.microsoft.com.

This book is provided "as-is" and expresses the author's views and opinions. The views, opinions, and information expressed in this book, including URL and other Internet website references, may change without notice.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

Microsoft and the trademarks listed at <https://www.microsoft.com> on the "Trademarks" webpage are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

Author: > **Cesar de la Torre**, Sr. PM, .NET Product Team, Microsoft Corp.

Participants and reviewers: > **Scott Hunter**, Partner Director PM, .NET team, Microsoft > **Paul Yuknewicz**, Principal PM Manager, Visual Studio Tools team, Microsoft > **Lisa Guthrie**, Sr. PM, Visual Studio Tools team, Microsoft > **Ankit Asthana**, Principal PM Manager, .NET team, Microsoft > **Unai Zorrilla**, Developer Lead, Plain Concepts > **Javier Valero**, Chief Operating Officer at Grupo Solutio > **Steve Smith**, Architect, NimblePros

Introduction

When you decide to modernize your web applications or services and move them to the cloud, you don't necessarily have to fully rearchitect your apps. Rearchitecting an application by using an advanced approach like microservices isn't always an option because of cost and time restraints.

Depending on the type of application, rearchitecting an app also might not be necessary. To optimize the cost-effectiveness of your organization's cloud migration strategy, it's important to consider the needs of your business and the requirements of your apps. You'll need to determine:

- Which apps require a transformation or rearchitecting.
- Which apps need to be only partially modernized.
- Which apps you can "lift and shift" directly to the cloud.



About this guide

This guide focuses primarily on the initial modernization of existing Microsoft .NET Framework web or service-oriented applications, meaning the action of moving a workload to a newer or more modern environment without significantly altering the application's code and basic architecture.

This guide also highlights the benefits of moving your apps to the cloud and partially modernizing apps by using a specific set of new technologies and approaches, like Windows Containers and related compute-platforms in Azure supporting Windows Containers.

Path to the cloud for existing .NET applications

Organizations typically choose to move to the cloud for the agility and speed they can get for their applications. You can set up thousands of servers (VMs) in the cloud in minutes, compared to the weeks it typically takes to set up on-premises servers.

There isn't a single, one-size-fits-all strategy for migrating applications to the cloud. The right migration strategy for you will depend on your organization's needs and priorities, and the kind of applications you are migrating. Not all applications warrant the investment of moving to a platform as a service ([PaaS](#)) model or developing a [cloud-native](#) application model. In many cases, you can take a phased or incremental approach to invest in moving your assets to the cloud, based on your business needs.

For modern applications with the best long-term agility and value for the organization, you might benefit from investing in *cloud-native* application architectures. However, for applications that are existing or legacy assets, the key is to spend minimal time and money (no rearchitecting or code changes) while moving them to the cloud, to realize significant benefits.

Figure 1-1 shows the possible paths you can take when you move existing .NET applications to the cloud in incremental phases.

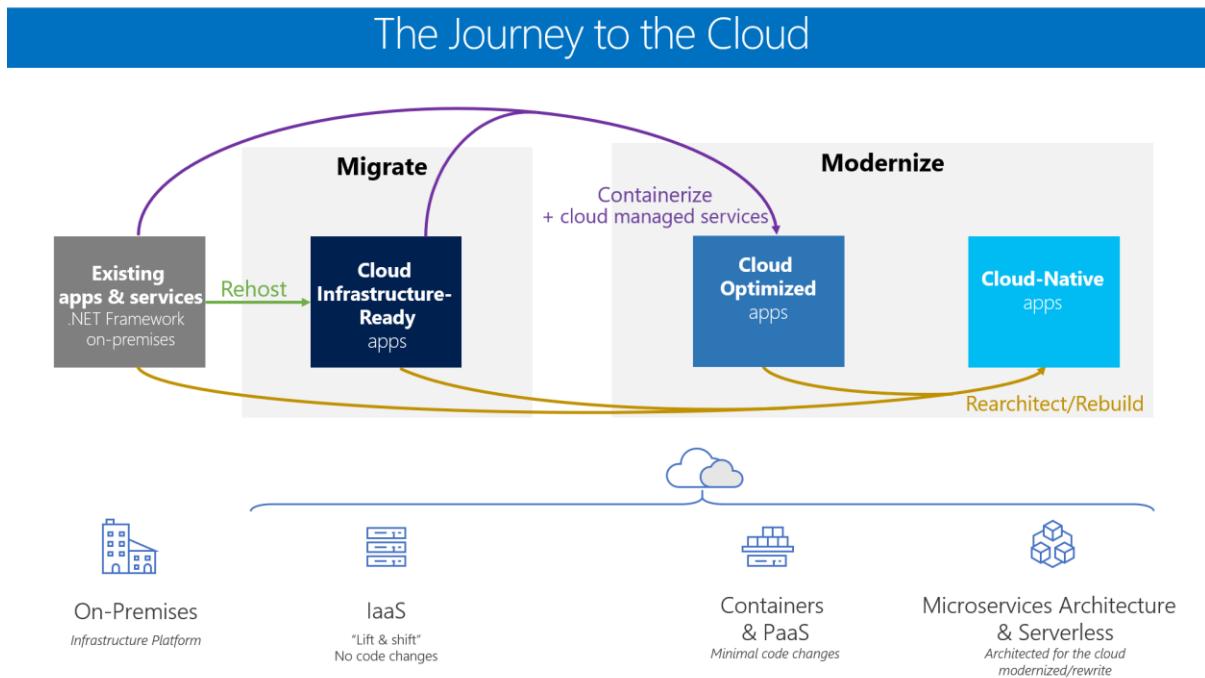


Figure 1-1. Modernization paths for existing .NET applications and services

Each migration approach has different benefits and reasons for using it. You can choose a single approach when you migrate apps to the cloud, or choose certain components from multiple approaches. Individual applications aren't limited to a single approach or maturity state. For instance, a common hybrid approach would have certain on-premises components plus other components in the cloud.

The definition and short explanation for each application maturity level are the following:

Level 1: Cloud Infrastructure-Ready applications: In this migration approach, you just migrate or rehost your current on-premises applications to an infrastructure as a service ([IaaS](#)) platform. Your apps have almost the same composition as before, but now you deploy them to VMs in the cloud. This simple type of migration is typically known in the industry as "Lift & Shift."

Level 2: Cloud Optimized applications: At this level and still without rearchitecting or altering significant code, you can gain additional benefits from running your app in the cloud with modern technologies like containers and additional cloud-managed services. You improve the agility of your applications to ship faster by refining your enterprise development operations (DevOps) processes. You achieve this functionality by using technologies like Windows Containers, which is based on Docker Engine. Containers remove the friction that's caused by application dependencies when you deploy in multiple stages. In this maturity model, you can deploy containers on IaaS or PaaS while using additional cloud-managed services related to databases, cache as a service, monitoring, and continuous integration/continuous deployment (CI/CD) pipelines.

The third level of maturity is the ultimate goal in the cloud, but it's optional for many apps and not the main focus of this guide:

Level 3: Cloud-Native applications: This migration approach typically is driven by business need and targets modernizing your mission-critical applications. At this level, you use PaaS services to move your apps to PaaS computing platforms. You implement [cloud-native](#) applications and microservices architecture to evolve applications with long-term agility, and to scale to new limits. This type of modernization usually requires architecting specifically for the cloud. New code often must be written, especially when you move to cloud-native application and microservice-based models. This approach can help you gain benefits that are difficult to achieve in your monolithic and on-premises application environment.

Table 1-1 describes the main benefits of and reasons for choosing each migration or modernization approach.

Cloud Infrastructure-Ready Lift and shift	Cloud-Optimized Modernize	Cloud-Native Modernize, rearchitect, and rewrite
Application's compute target		
Applications deployed to VMs in Azure	Monolithic or N-Tier apps deployed to Azure App Service, Azure Container Instance (ACI), VMs with containers, or AKS (Azure Kubernetes Service)	Containerized microservices on Azure Kubernetes Service (AKS) and/or serverless microservices based on Azure Functions.
Data target		
SQL or any relational database on a VM	Azure SQL Database Managed Instance or another managed database in the cloud.	Fined-grain databases per microservice, based on Azure SQL Database, Azure Cosmos DB, or another managed database in the cloud
Advantages		
<ul style="list-style-type: none"> • No rearchitecting, no new code • Least effort for quick migration • Least-common denominator supported in Azure • Basic availability guarantees • After moving to the cloud, it's easier to modernize even more 	<ul style="list-style-type: none"> • No rearchitecting • Minimal code/config changes • Improved deployment and DevOps agility to release because of containers • Increased density and lower deployment costs • Portability of apps and dependencies • Flexibility of host targets: PaaS approaches or IaaS 	<ul style="list-style-type: none"> • Architect for the cloud, you get the best benefits from the cloud but new code is needed • Microservices cloud-native approaches • Modern mission-critical applications, cloud-resilient hyper-scalable • Fully managed services • Optimized for scale • Optimized for autonomous agility by subsystem • Built on deployment and DevOps
Challenges		
<ul style="list-style-type: none"> • Smaller cloud value, other than the shift in operating expense or closing datacenters 	<ul style="list-style-type: none"> • Containerizing is an additional step in the learning curve for developers and IT Operations • DevOps and CI/CD pipelines are 	<ul style="list-style-type: none"> • Requires rearchitecture for cloud-native apps and microservice architectures and usually requires significant code refactoring or rewriting when modernizing (increased time and Budget)



Cloud Infrastructure-Ready Lift and shift	Cloud-Optimized Modernize	Cloud-Native Modernize, rearchitect, and rewrite
<ul style="list-style-type: none">• Little is managed: No OS or middleware patching; might use infrastructure solutions, like Terraform, Spinnaker, or Puppet	usually 'a must' for this approach. If not currently present in the culture of the organization, it might be an additional challenge	

Table 1-1. Benefits and challenges of modernization paths for existing .NET applications and services

Key technologies and architectures by maturity level

.NET Framework applications initially started with the .NET Framework version 1.0, which was released in late 2001. Then, companies moved towards newer versions (such as 2.0, 3.5 and .NET Framework 4.x). Most of those applications ran on Windows Server and Internet Information Server (IIS), and used a relational database, like SQL Server, Oracle, MySQL, or any other RDBMS.

Most existing .NET applications might nowadays be based on .NET Framework 4.x, or even on .NET Framework 3.5, and use web frameworks like ASP.NET MVC, ASP.NET Web Forms, ASP.NET Web API, Windows Communication Foundation (WCF), ASP.NET SignalR, and ASP.NET Web Pages. These established .NET Framework technologies depend on Windows. That dependency is important to consider if you are simply migrating legacy apps and you want to make minimal changes to your application infrastructure.

Figure 1-2 shows the primary technologies and architecture styles used at each of the three cloud maturity levels:

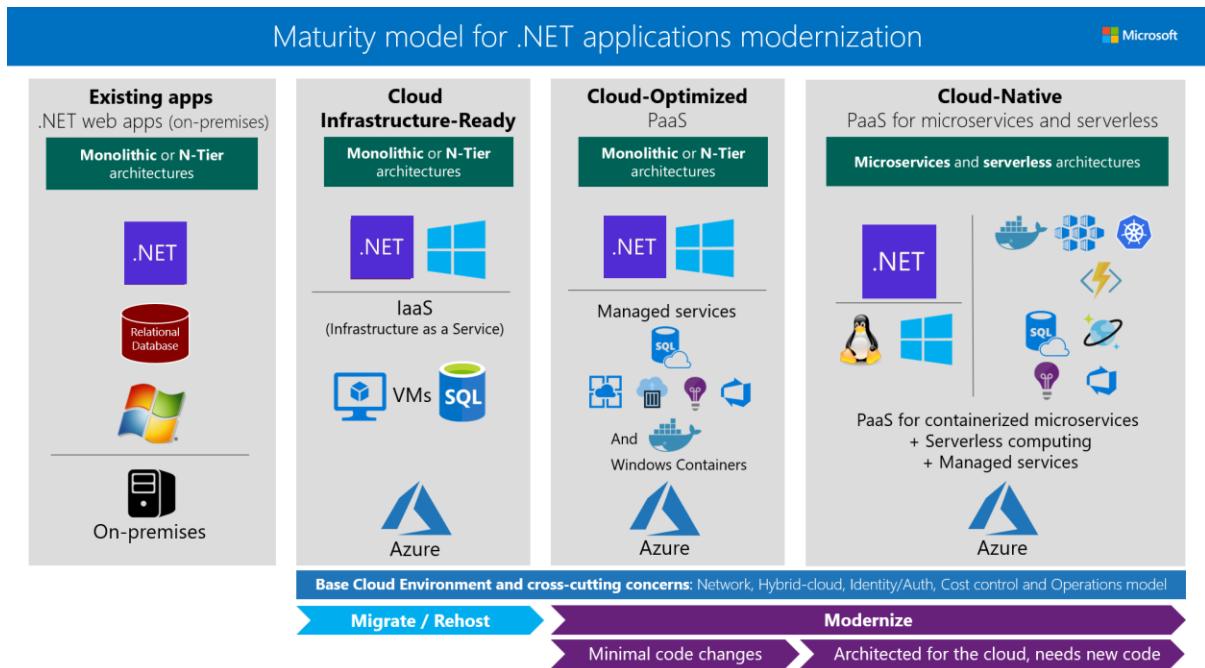


Figure 1-2. Primary technologies for each maturity level for modernizing existing .NET web applications

Figure 1-2 highlights the most common scenarios, but many hybrid and mixed variations are possible when it comes to architecture. For example, the maturity models apply not only to monolithic architectures in existing web apps, but also to service orientation, N-Tier, and other architecture style variations. The higher focus or percentage on one or another architecture type and related technologies determines the overall maturity level of your applications.

Each maturity level in the modernization process is associated with the following key technologies and approaches:

- **Cloud Infrastructure-Ready** (rehost or basic lift & shift): As a first step, many organizations want only to quickly execute a cloud-migration strategy. In this case, applications are rehosted. Most rehosting can be automated by using [Azure Migrate](#), a service that provides the guidance, insights, and mechanisms needed to assist you in migrating to Azure based on cloud tools like [Azure Site Recovery](#) and [Azure Database Migration Service](#). You can also set up rehosting manually, so that you can learn infrastructure details about your assets when you move legacy apps to the cloud. For example, you can move your applications to VMs in Azure with little modification—probably with only minor configuration changes. The networking in this case is similar to an on-premises environment, especially if you create virtual networks in Azure.
- **Cloud-Optimized** (Managed Services and Windows Containers): This model is about making a few important deployment optimizations to gain some significant benefits from the cloud, without changing the core architecture of the application. The fundamental step here is to add [Windows Containers](#) support to your existing .NET Framework applications. This important step (containerization) doesn't require touching the code, so the overall lift and shift effort is light. You can use tools like [Image2Docker](#) or Visual Studio, with its tools for [Docker](#). Visual Studio automatically chooses smart defaults for ASP.NET applications and Windows Containers images. These tools offer both a rapid inner loop, and a fast path to get the containers to Azure. Your

agility is improved when you deploy to multiple environments. Then, moving to production, you can deploy your Windows Containers to [Azure Web App for Containers](#), [Azure Container Instances \(ACI\)](#), and Azure VMs with Windows Server 2016 and containers if you prefer an IaaS approach. For more complex multi-container applications, consider using orchestrators like [Azure Kubernetes Service \(AKS/ACS\)](#).

During this initial modernization, you can also add assets from the cloud, such as monitoring with tools like [Azure Application Insights](#); CI/CD pipelines for your app lifecycles with [Azure DevOps Services](#); and many more data resource services that are available in Azure. For instance, you can modify a monolithic web app that was originally developed by using traditional [ASP.NET Web Forms](#) or [ASP.NET MVC](#), but now you deploy it by using Windows Containers. When you use Windows Containers, you should also migrate your data to a database in [Azure SQL Database Managed Instance](#), all without changing the core architecture of your application.

- **Cloud-Native:** As introduced, you should think about architecting [cloud-native](#) applications when you are targeting large and complex applications with multiple independent development teams working on different microservices that can be developed and deployed autonomously. Also, due to granularized and independent scalability per microservice. These architectural approaches face very important challenges and complexities but can be greatly simplified by using cloud PaaS and orchestrators like [Azure Kubernetes Service \(AKS/ACS\)](#) (managed Kubernetes), and [Azure Functions](#) for a serverless approach. All these approaches (like microservices and Serverless) typically require you to architect for the cloud and write new code—code that is adapted to specific PaaS platforms, or code that aligns with specific architectures, like microservices.

Figure 1-3 shows the internal technologies that you can use for each maturity level:

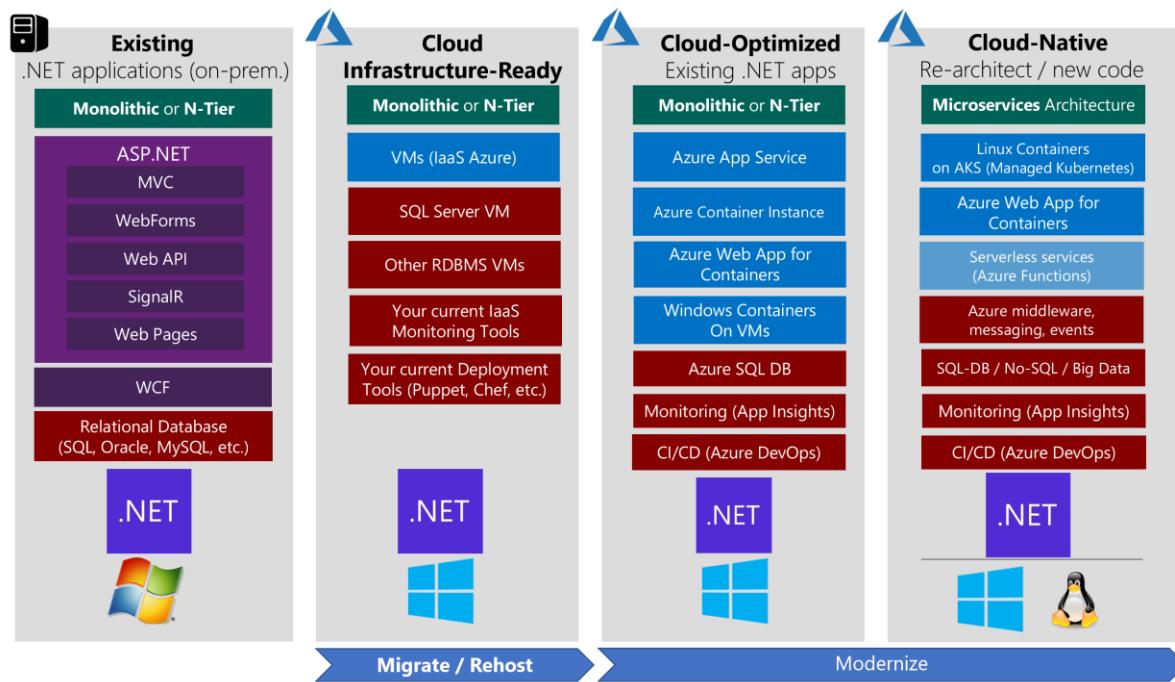


Figure 1-3. Internal technologies for each modernization maturity level

Lift and shift scenario

For lift and shift migrations, keep in mind that you can use many different variations of lift and shift in your application scenarios. If you only rehost your application, you might have a scenario like the one shown in Figure 1-4, where you use VMs in the cloud only for your application and for your database server.

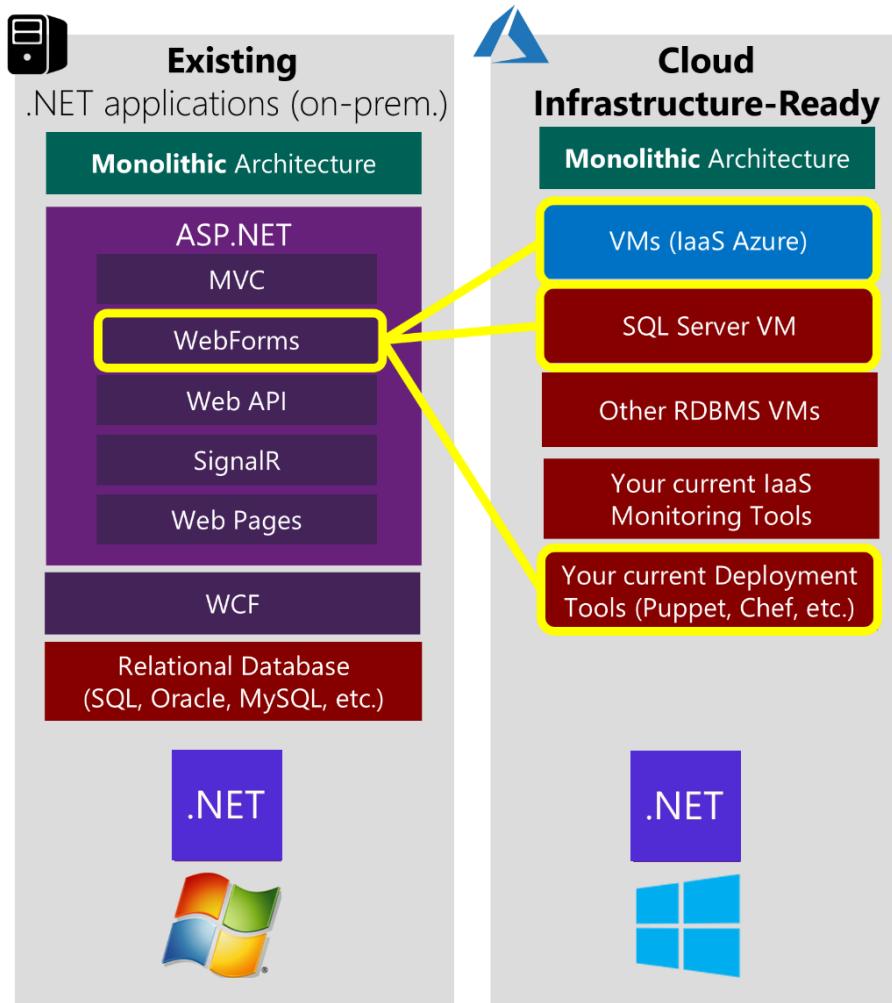


Figure 1-4. Example of a pure IaaS scenario in the cloud

Modernization scenarios

For modernization scenarios, you might have a pure Cloud-Optimized application that uses elements only from that maturity level. Or, you might have an intermediate-state application with some elements from Cloud Infrastructure-Ready and other elements from Cloud-Optimized (a “pick and choose” or mixed model), like in Figure 1-5.

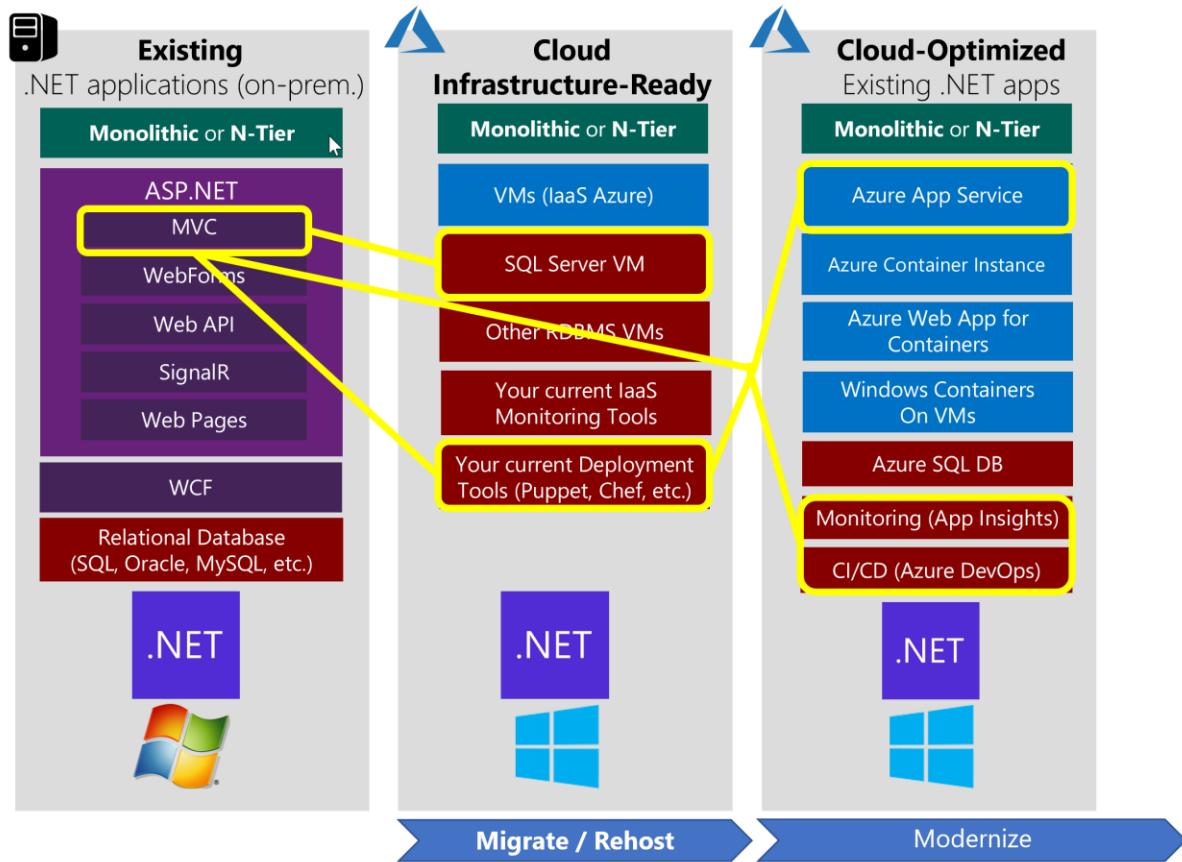


Figure 1-5. Example “pick and choose” scenario, with database on IaaS, DevOps, and containerization assets

Next, as the ideal scenario for many existing .NET Framework applications to migrate, you could migrate to a Cloud-Optimized application, to get significant benefits from little work. This approach also sets you up for Cloud-Native as a possible future evolution. Figure 1-6 shows an example.

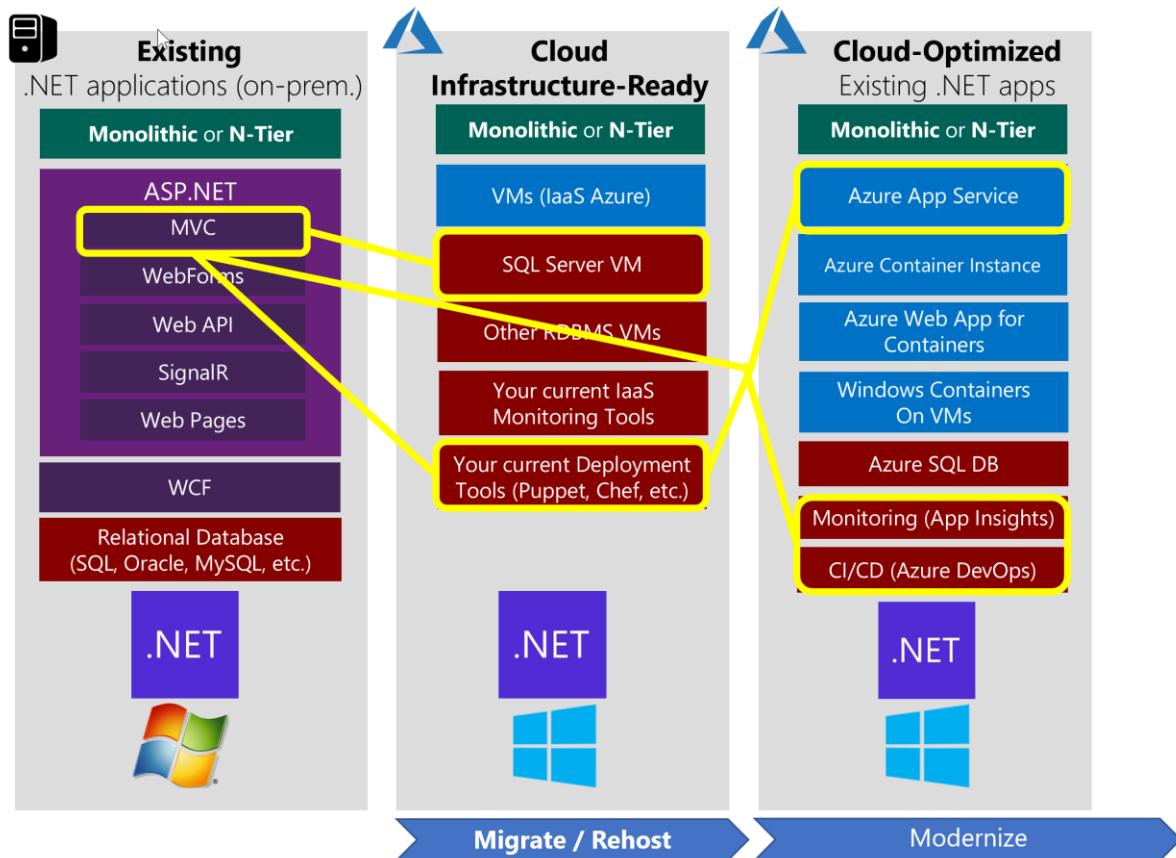


Figure 1-6. Example Cloud-Optimized apps scenario, with Windows Containers and managed services

Going even further, you could extend your existing Cloud-Optimized application by adding a few microservices for specific scenarios. This approach would move you partially to the level of Cloud-Native model, which is not the main focus of the present guidance.

What this guide does not cover

This guide covers a specific subset of the example scenarios, as shown in Figure 1-7. This guide focuses only on the lift and shift scenarios, and ultimately, on the Cloud-Optimized model. In the Cloud-Optimized model, a .NET Framework application is modernized by using Windows Containers, plus additional components like monitoring and CI/CD pipelines. Each component is fundamental to deploying applications to the cloud, faster, and with agility.

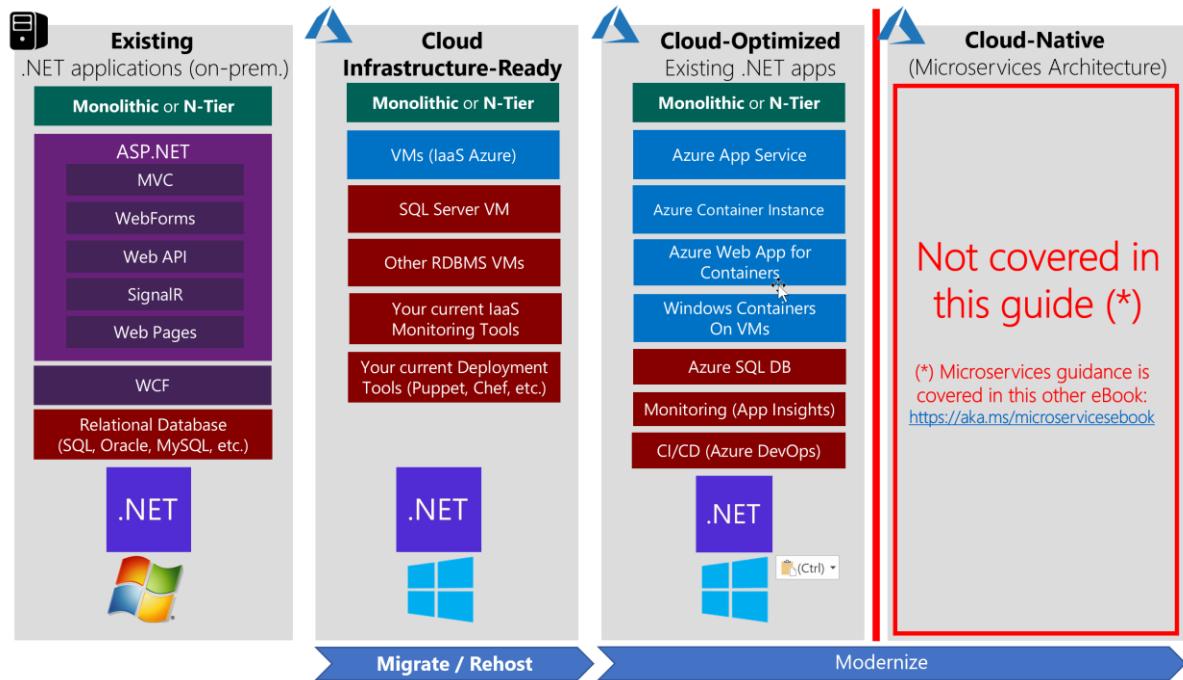


Figure 1-7. Cloud-Native is not covered in this guide

The focus of this guide is specific. It shows you the path you can take to achieve a lift and shift of your existing .NET applications, without rearchitecting, and with no code changes. Ultimately, it shows you how to make your application Cloud-Optimized.

This guide doesn't show you how to create Cloud-Native applications, such as how to evolve to a microservices architecture. To rearchitect your applications or to create brand-new applications that are based on microservices, see the e-book [.NET Microservices: Architecture for containerized .NET applications](https://aka.ms/microservicesebook).

Additional resources

- **Containerized Docker application lifecycle with Microsoft platform and tools** (downloadable e-book)
<https://aka.ms/dockerlifecyclebook>
- **.NET Microservices: Architecture for containerized .NET applications** (downloadable e-book)
<https://aka.ms/microservicesebook>
- **Architecting modern web applications with ASP.NET Core and Azure** (downloadable e-book)
<https://aka.ms/webappebook>

Who should use this guide

This guide was written for developers and solution architects who want to modernize existing ASP.NET web applications or WCF services that are based on the .NET Framework, for improved agility in shipping and releasing applications.



You also might find this guide useful if you are a technical decision maker, such as an enterprise architect or a development lead/director who just wants an overview of the benefits that you can get by using Windows Containers, and by deploying to the cloud when using Microsoft Azure.

How to use this guide

This guide addresses the "why"-why you might want to modernize your existing applications, and the specific benefits you get from using Windows Containers when you move your apps to the cloud. The content in the first few chapters of the guide is designed for architects and technical decision makers who want an overview, but who don't need to focus on implementation and technical, step-by-step details.

The last chapter of this guide introduces multiple walkthroughs that focus on specific deployment scenarios. This guide offers shorter versions of the walkthroughs, to summarize the scenarios and highlight their benefits. The full walkthroughs drill down into setup and implementation details, and are published as a set of [wiki posts](#) in the same public [GitHub repo](#) where related sample apps reside (discussed in the next section). The last chapter and the step-by-step wiki walkthroughs on GitHub will be of more interest to developers and architects who want to focus on implementation details.

Sample apps for modernizing legacy apps: eShopModernizing

The [eShopModernizing](#) repo on GitHub offers two sample applications that simulate legacy monolithic web applications. One web app is developed by using ASP.NET MVC; the second web app is developed by using ASP.NET Web Forms and the third app is an N-Tier app with a WinForms client desktop app consuming a WCF service backend. All these apps are based on the traditional .NET Framework. These sample apps don't use .NET Core/.NET 6 or ASP.NET Core as they are supposed to be existing/legacy .NET Framework applications to be modernized.

These sample apps have a second version, with modernized code, and which are fairly straightforward. The most important difference between the app versions is that the second versions use Windows Containers as the deployment choice. There also are a few additions to the second versions, like Azure Storage Blobs for managing images, Azure Active Directory for managing security, and Azure Application Insights for monitoring and auditing the applications.

Send your feedback

This guide was written to help you understand your options for improving and modernizing existing .NET web applications. The guide and related sample applications are evolving. Your feedback is welcome! If you have comments about how this guide might be more helpful, please send them to dotnet-architecture-ebooks-feedback@service.microsoft.com.

Contents

Lift and shift existing .NET apps to Azure IaaS (Cloud Infrastructure-Ready)	1
Why migrate existing .NET web applications to Azure IaaS	2
When to migrate to IaaS instead of to PaaS.....	2
Use Azure Migrate to analyze and migrate your existing applications to Azure	2
Use Azure Site Recovery to migrate your existing VMs to Azure VMs	3
Additional resources	4
Migrate your relational databases to Azure.....	6
When to migrate to Azure SQL Database Managed Instance	7
When to migrate to Azure SQL Database.....	8
When to move your original RDBMS to a VM (IaaS)	8
When to migrate to SQL Server as a VM (IaaS)	9
Use Azure Database Migration Service to migrate your relational databases to Azure	9
Additional resources.....	10
Modernize existing .NET apps to Cloud-Optimized applications	11
Reasons to modernize existing .NET apps to Cloud-Optimized applications.....	12
Cloud-Optimized application principles and tenets	12
Benefits of a Cloud-Optimized application	14
Microsoft technologies in cloud-optimized applications	14
Monolithic applications <i>can</i> be Cloud-Optimized	15
What about Cloud-Native applications?	16
Cloud-native applications details	17
What about microservices?	18
Deploy existing .NET apps as Windows containers.....	19
What are containers? (Linux or Windows).....	19
Benefits of containers (Docker Engine on Linux or Windows)	19
What is Docker?	20
Benefits of Windows Containers for your existing .NET applications	21
Choose an OS to target with .NET-based containers	22

Windows container types	23
The container ecosystem in Azure	24
When not to deploy to Windows Containers	26
When to deploy Windows Containers in your on-premises IaaS VM infrastructure	26
When to deploy Windows Containers to Azure VMs (IaaS cloud).....	26
When to deploy Windows Containers to Azure Container Instances (ACI).....	27
When to deploy Windows Containers to Azure Kubernetes Service (AKS)	28
Choosing Azure compute platforms for container-based applications.....	28
Build resilient services ready for the cloud: Embrace transient failures in the cloud.....	29
Handling partial failure	29
Modernize your apps with monitoring and telemetry.....	31
Monitor your application with Application Insights.....	31
Monitor your Docker infrastructure with Log Analytics and its Container Monitoring solution.....	32
Modernize your app's lifecycle with CI/CD pipelines and DevOps tools in the cloud	34
Migrate to hybrid cloud scenarios	35
Azure Stack	35
Walkthroughs and technical get started overview.....	38
Technical walkthrough list.....	38
Walkthrough 1: Tour of eShop legacy apps.....	39
Technical walkthrough availability	39
Overview	39
Goals.....	39
Scenario 1: ASP.NET Web apps	39
Scenario 2: WCF service and WinForms client app (3-Tier app).....	40
Benefits.....	41
Next steps	41
Walkthrough 2: Containerize your existing .NET applications with Windows Containers.....	41
Overview	41
Goals.....	41
Scenario 1: Containerized ASP.NET web apps.....	42
Scenario 2: Containerized WCF service	42
Benefits.....	43

Next steps	43
Walkthrough 3: Deploy your Windows Containers-based app to Azure VMs.....	44
Technical walkthrough availability	44
Overview	44
Goals.....	44
Scenarios.....	44
Azure VMs for Windows Containers.....	46
Benefits.....	46
Next steps	46
Walkthrough 4: Deploy your Windows Containers-based apps to Azure Container Instances (ACI).	47
Technical walkthrough availability	47
Overview	47
Goals.....	47
Scenarios.....	47
Benefits.....	47
Considerations.....	48
Next steps	48
Walkthrough 5: Deploy your Windows Containers-based apps to Kubernetes in Azure Container Service	48
Technical walkthrough availability	48
Overview	48
Goals.....	49
Scenarios.....	49
Benefits.....	50
Next steps	51
Walkthrough 6: Deploy your Windows Containers-based apps to Azure App Service for Containers	51
Technical walkthrough availability	51
Overview	51
Goals.....	51
Scenario	52
Benefits.....	52

Next steps	52
Conclusions	53

Lift and shift existing .NET apps to Azure IaaS (Cloud Infrastructure-Ready)

Vision: As a first step, to reduce your on-premises investment and total cost of hardware and networking maintenance, simply rehost your existing applications in the cloud.

Before getting into *how* to migrate your existing applications to the Azure infrastructure as a service (IaaS) platform, it's important to analyze the reasons *why* you'd want to migrate directly to IaaS in Azure. The scenario at this modernization maturity level essentially is to start using VMs in the cloud, instead of continuing to use your current, on-premises infrastructure.

Another point to analyze is *why* you might want to migrate to pure IaaS cloud instead of just adding more advanced managed services in Azure. Determine what cases might require IaaS in the first place.

Figure 2-1 positions Cloud Infrastructure-Ready applications in the modernization maturity levels:

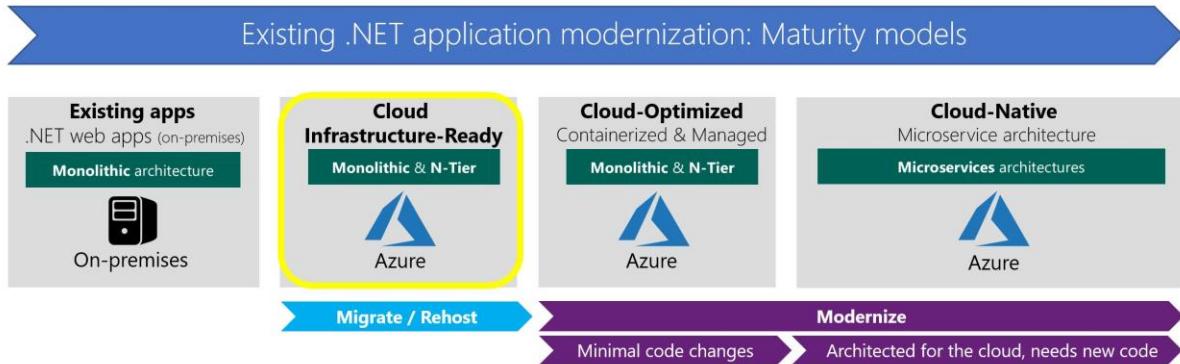


Figure 2-1. Positioning Cloud Infrastructure-Ready applications

Why migrate existing .NET web applications to Azure IaaS

The main reason to migrate to the cloud, even at an initial IaaS level, is to achieve cost reductions. By using more managed infrastructure services, your organization can lower its investment in hardware maintenance, server or VM provisioning and deployment, and infrastructure management.

After you make the decision to move your apps to the cloud, the main reason why you might choose IaaS instead of more advanced options like PaaS is simply that the IaaS environment will be more familiar. Moving to an environment that's similar to your current, on-premises environment offers a lower learning curve, which makes it the quickest path to the cloud.

However, taking the quickest path to the cloud doesn't mean that you will gain the most benefit from having your applications running in the cloud. Any organization will gain the most significant benefits from a cloud migration at the already introduced Cloud-Optimized and Cloud-Native maturity levels.

It also has become evident that applications are easier to modernize and rearchitect in the future when they are already running in the cloud, even on IaaS. Application data migration has already been achieved. Also, your organization will have gained the skills required for working in the cloud and made the shift to operating in a "cloud culture."

When to migrate to IaaS instead of to PaaS

The next sections discuss Cloud-Optimized applications that are mostly based on PaaS platforms and services. These apps give you the most benefits from migrating to the cloud.

If your goal is simply to move existing applications to the cloud, first, identify existing applications that would not require substantial modification to run in Azure App Service. These apps should be the first candidates for Cloud-Optimized.

Then, for the apps that still cannot move to Windows Containers and PaaS such as App Service or orchestrators like Azure Kubernetes Service, migrate those apps to simple plain VMs (IaaS).

But, keep in mind that correctly configuring, securing, and maintaining VMs requires much more time and IT expertise compared to using PaaS services in Azure. If you are considering Azure Virtual Machines, make sure that you take into account the ongoing maintenance effort required to patch, update, and manage your VM environment. Azure Virtual Machines is IaaS.

Use Azure Migrate to analyze and migrate your existing applications to Azure

Migrating to the cloud doesn't have to be difficult. But many organizations struggle to get started - to get deep visibility into the environment and the tight interdependencies between applications, workloads, and data. Without that visibility, it can be difficult to plan the path forward. Without detailed information on what's required for a successful migration, you can't have the right

conversations within your organization. You don't know enough about the potential cost benefits, or whether workloads could just lift-and-shift or would require significant rework to migrate successfully. No wonder many organizations hesitate.

[Azure Migrate](#) is a new service that provides the guidance, insights, and mechanisms needed to assist you in migrating to Azure. Azure Migrate provides:

- Discovery and assessment for on-premises virtual machines
- Inbuilt dependency mapping for high-confidence discovery of multi-tier applications
- Intelligent right sizing to Azure virtual machines
- Compatibility reporting with guidelines for remediating potential issues
- Integration with Azure Database Management Service for database discovery and migration

Azure Migrate gives you confidence that your workloads can migrate with minimal impact to the business and run as expected in Azure. With the right tools and guidance, you can achieve maximum return on investment while assuring that critical performance and reliability needs are met.

Figure 2-2 shows you the built-in dependency mapping for all server and application connections performed by Azure Migrate.

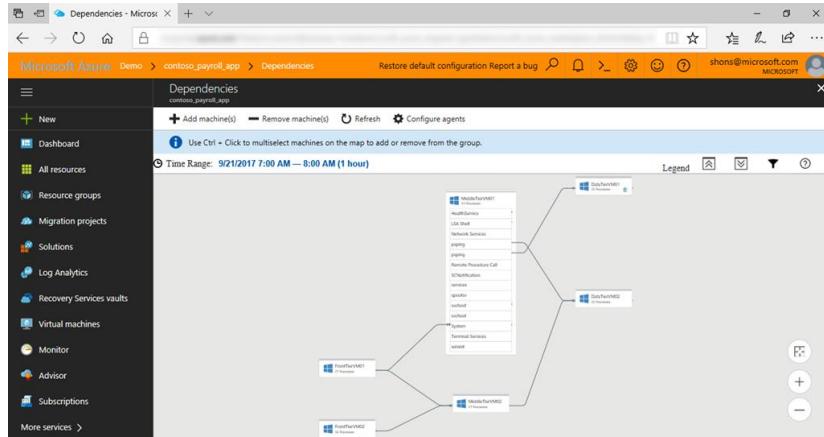


Figure 2-2. Positioning Cloud Infrastructure-Ready applications

Use Azure Site Recovery to migrate your existing VMs to Azure VMs

As part of the end-to-end [Azure Migrate](#), [Azure Site Recovery](#) is a tool that you can use to easily migrate your web apps to VMs in Azure. You can use Site Recovery to replicate on-premises VMs and physical servers to Azure, or to replicate them to a secondary on-premises location. You can even replicate a workload that's running on a supported Azure VM, on an on-premises *Hyper-V* VM, on a *VMware* VM, or on a Windows or Linux physical server. Replication to Azure eliminates the cost and complexity of maintaining a secondary datacenter.

Site Recovery is also made specifically for hybrid environments that are partly on-premises and partly on Azure. Site Recovery helps ensure business continuity by keeping your apps that are running on VMs and on-premises physical servers available if a site goes down. It replicates workloads that are running on VMs and physical servers so that they remain available in a secondary location if the primary site isn't available. It recovers workloads to the primary site when it's up and running again.

Figure 2-3 shows the execution of multiple VM migrations by using Azure Site Recovery.

The screenshot shows the 'Replicated items' dashboard for 'ContosoRecoveryServices'. The table lists five replicated items:

NAME	HEALTH	STATUS	ACTIVE LOCATION	REPLICATION POLICY
AWSServer2012	● OK	Unplanned failover comp...	Microsoft Azure	AWSReplicationPolicy
AWSWordpressRedHat	● OK	Protected	AWSGATEWAY	AWSReplicationPolicy
FabrikamMarketing	● OK	Unplanned failover comp...	Microsoft Azure	ContosoReplicationPolicy
FabrikamFinance	● OK	Protected	CONTOSOGATEWAY	ContosoReplicationPolicy
▶ ContosoReplicationGr...	-	-	-	-

A context menu is open over the last row, with the 'Complete Migration' option highlighted with a red box. Other options in the menu include: Pin to dashboard, Unplanned Failover, Test Failover, Change PIT, Commit, Re-protect, Resynchronize, Error Details, and Delete.

Figure 2-3. Positioning Cloud Infrastructure-Ready applications

Additional resources

- **Azure Migrate Datasheet**
<https://aka.ms/azuremigration\ datasheet>
- **Azure Migrate**
<https://aka.ms/azuremigrate>
- **Azure migration and modernization center**
<https://azure.microsoft.com/migration/>
- **Migrate to Azure with Site Recovery**
<https://docs.microsoft.com/azure/site-recovery/site-recovery-migrate-to-azure>
- **Azure Site Recovery service overview**

<https://docs.microsoft.com/azure/site-recovery/site-recovery-overview>

- **Migrating VMs in AWS to Azure VMs**

<https://docs.microsoft.com/azure/site-recovery/site-recovery-migrate-aws-to-azure>

Migrate your relational databases to Azure

Vision: Azure offers the most comprehensive database migration.

In Azure, you can migrate your database servers directly to IaaS VMs (pure lift and shift), or you can migrate to Azure SQL Database, for additional benefits. Azure SQL Database offers the managed instance and full database-as-a-service (DBaaS) options. Figure 3-1 shows the multiple relational database migration paths available in Azure.

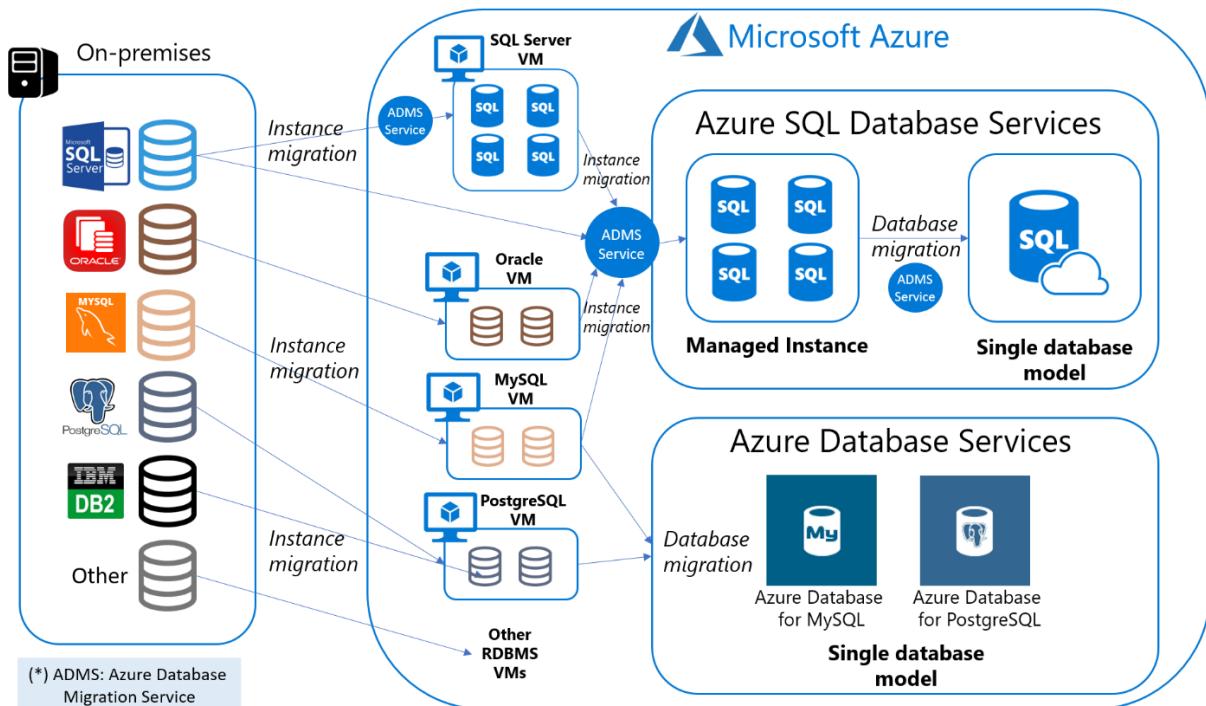


Figure 3-1. Database migration paths in Azure

When to migrate to Azure SQL Database Managed Instance

In most cases, Azure SQL Database Managed Instance will be your best option to consider when you migrate your data to Azure. If you are migrating SQL Server databases and need nearly 100% assurance that you won't need to rearchitect your application or make changes to your data or data access code, choose the Managed Instance feature of Azure SQL Database.

Azure SQL Database Managed Instance is the best option if you have additional requirements for SQL Server instance-level functionality, or isolation requirements beyond the features provided in a standard Azure SQL Database (single database model). This last one is the most PaaS-oriented choice, but it doesn't offer the same features as that of a traditional SQL server. Migration might surface frictions.

For example, an organization that has made deep investments in instance-level SQL Server capabilities would benefit from migrating to SQL Managed Instance. Examples of instance-level SQL Server capabilities include SQL common language runtime (CLR) integration, SQL Server Agent, and cross-database querying. Support for these features is not available in standard Azure SQL Database (a single-database model).

An organization that operates in a highly regulated industry, and which needs to maintain isolation for security purposes, also might benefit from choosing the SQL Managed Instance model.

Managed Instance in Azure SQL Database has the following characteristics:

- Security isolation through Azure Virtual Network
- Application surface compatibility, with these features:
 - SQL Server Agent and SQL Server Profiler
 - Cross-database references and queries, SQL CLR, replication, change data capture (CDC), and Service Broker
- Database sizes up to 35 TB
- Minimum-downtime migration, with these features:
 - Azure Database Migration Service
 - Native backup and restore, and log shipping

With these capabilities, when you migrate existing application databases to Azure SQL Database, the Managed Instance model offers nearly 100% of the benefits of PaaS for SQL Server. Managed Instance is a SQL Server environment where you continue using instance-level capabilities without changing your application design.

Managed Instance is probably the best fit for enterprises that currently are using SQL Server, and which require flexibility in their network security in the cloud. It's like having a private virtual network for your SQL databases.

When to migrate to Azure SQL Database

As mentioned, the standard Azure SQL Database is a fully managed, relational DBaaS. SQL Database currently manages millions of production databases, across 38 datacenters, around the world. It supports a broad range of applications and workloads, from managing straightforward transactional data, to driving the most data-intensive, mission-critical applications that require advanced data processing at a global scale.

Because of its full PaaS features, better pricing-and ultimately lower cost-you should move to the standard Azure SQL Database as your "by-default choice" if you have an application that uses basic, standard SQL databases, and no additional instance features. SQL Server features like SQL CLR integration, SQL Server Agent, and cross-database querying are not supported in the standard Azure SQL Database. Those features are available only in the Azure SQL Database Managed Instance model.

Azure SQL Database is the only intelligent cloud database service that's built for app developers. It's also the only cloud database service that scales on-the-fly, without downtime, to help you efficiently deliver multitenant apps. Ultimately, Azure SQL Database leaves you more time to innovate, and it accelerates your time to market. You can build secure apps and connect to your SQL database by using the languages and platforms that you prefer.

Azure SQL Database offers the following benefits:

- Built-in intelligence (machine learning) that learns and adapts to your app
- On-demand database provisioning
- A range of offers, for all workloads
- 99.99% availability SLA, zero maintenance
- Geo-replication and restore services for data protection
- Azure SQL Database Point in Time Restore feature
- Compatibility with SQL Server 2016, including hybrid and migration

The standard Azure SQL Database is closer to PaaS than Azure SQL Database Managed Instance. Prefer the standard Azure SQL Database because you'll get more benefits from a managed cloud. However, Azure SQL Database has some key differences from regular and on-premises SQL Server instances. Depending on your existing application's database requirements, and your enterprise requirements and policies, it might not be the best choice when you are planning your migration to the cloud.

When to move your original RDBMS to a VM (IaaS)

One of your migration options is to move your original relational database management system (RDBMS), including Oracle, IBM DB2, MySQL, PostgreSQL, or SQL Server, to a similar server that's running on an Azure VM. If you have existing applications that require the fastest migration to the cloud with minimal changes, or no changes at all, a direct migration to IaaS in the cloud might be a

fair option. It might not be the best way to take advantage of all the cloud's benefits, but it's probably the fastest initial path.

Currently, Microsoft Azure supports up to [331 different database servers](#) deployed as IaaS VMs. These include popular RDBMS like SQL Server, Oracle, MySQL, PostgreSQL, and IBM DB2, and many other NoSQL databases like MongoDB, Cassandra, DataStax, MariaDB, and Cloudera.

Note

Although moving your RDBMS to an Azure VM might be the fastest way to migrate your data to the cloud (because it is IaaS), this approach requires a significant investment in your IT teams (database administrators and IT pros). Enterprise teams need to be able to set up and manage high availability, disaster recovery, and patching for SQL Server. This context also needs a customized environment, with full administrative rights.

When to migrate to SQL Server as a VM (IaaS)

There might be a few cases where you still need to migrate to SQL Server as a regular VM. An example scenario is if you need to use SQL Server Reporting Services. In most cases, though, Azure SQL Database Managed Instance can provide everything you need to migrate from on-premises SQL servers, so migration to a SQL Server VM should be your last resort to try.

Use Azure Database Migration Service to migrate your relational databases to Azure

You can use Azure Database Migration Service to migrate relational databases like SQL Server, Oracle, and MySQL to Azure, whether your target database is Azure SQL Database, Azure SQL Database Managed Instance, or SQL Server on an Azure VM.

The automated workflow, with assessment reporting, guides you through the changes you need to make before you migrate the database. When you are ready, the service migrates the source database to Azure.

Whenever you change an original RDBMS, you might need to retest. You also might need to change the SQL sentences or Object-Relational Mapping (ORM) code in your application, depending on testing results.

If you have any other database (for example, IBM DB2) and you opt for a lift and shift approach, you might want to continue using those databases as IaaS VMs in Azure, unless you are willing to perform a more complex data migration. A more complex data migration will require additional effort because you'd be migrating to a different database type with the new schema and different programming libraries.

To learn how to migrate databases by using Azure Database Migration Service, see [Get to the cloud faster with Azure SQL Database Managed Instance and Azure Database Migration Service](#).

Additional resources

- **Choose a cloud SQL Server option: Azure SQL Database (PaaS) or SQL Server on Azure VM (IaaS)**
<https://docs.microsoft.com/azure/sql-database/sql-database-paas-vs-sql-server-iaas>
- **Get to the cloud faster with Azure SQL DB Managed Instance and Database Migration Service**
<https://channel9.msdn.com/Events/Build/2017/P4008>
- **SQL Server database migration to SQL Database in the cloud**
<https://docs.microsoft.com/azure/sql-database/sql-database-cloud-migrate>
- **Azure SQL Database**
<https://azure.microsoft.com/services/sql-database/?v=16.50>
- **SQL Server on virtual machines**
<https://azure.microsoft.com/services/virtual-machines/sql-server/>

Modernize existing .NET apps to Cloud-Optimized applications

Vision: Modernize your existing .NET Framework applications to Cloud-Optimized applications to drastically improve your deployment agility, so you can ship faster and lower application's delivery costs.

To take advantage of the benefits of the cloud and new technologies like containers, you should at least partially modernize your existing .NET applications. Ultimately, modernizing your enterprise applications will lower your total cost of ownership.

Partially modernizing an app doesn't necessarily mean a full migration and rearchitecture. You can initially modernize your existing applications with important but easy to do modernization. You can maintain your current code base, written in existing .NET Framework versions, with any Windows and IIS dependencies. Figure 4-1 highlights how Cloud-Optimized apps are positioned in Azure application modernization maturity models.

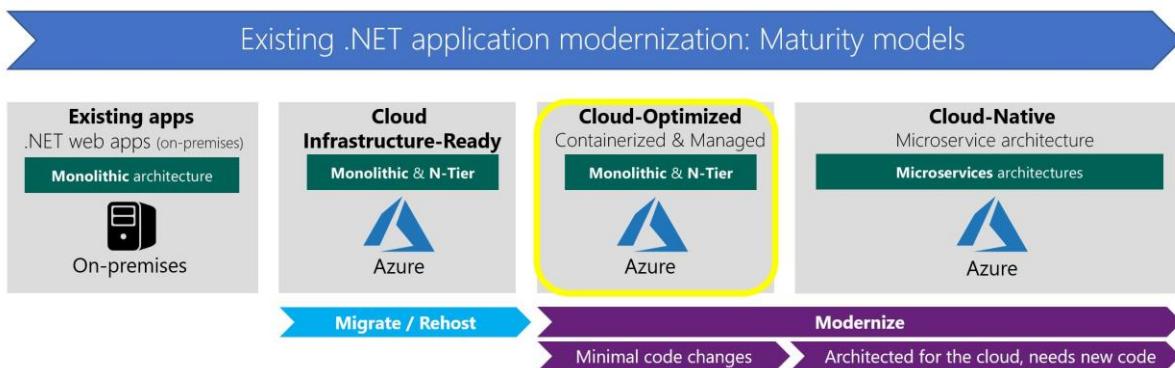


Figure 4-1. Positioning Cloud-Optimized applications

Reasons to modernize existing .NET apps to Cloud-Optimized applications

With a Cloud-Optimized application, you can rapidly and repeatedly deliver reliable applications to your customers. You gain essential agility and reliability by deferring much of the operational complexity of your app to the platform.

If you can't get your applications to market quickly, by the time you ship your app, the market you were targeting will have evolved. You might be too late, no matter how well the application was architected or engineered. You might be failing or not reaching your full potential because you can't sync app delivery with the needs of the market.

The need for continuous business innovation pushes development and operations teams to the limit. The only way to achieve the agility you need in continuous business innovation is by modernizing your applications with technologies like containers and specific Cloud-Optimized application principles.

The bottom line is that when an organization builds and manages applications that are Cloud-Optimized, it can put solutions in the hands of customers sooner and bring new ideas to market when they are relevant.

Cloud-Optimized application principles and tenets

Improvements in the cloud are mostly focused on meeting two goals: Reduce costs and improve business growth by improving agility. These goals are achieved by simplifying processes and reducing friction when you release and ship applications.

Your application is Cloud-Optimized if you can-in an agile manner-develop your app autonomously from other on-premises apps, and then release, deploy, autoscale, monitor, and troubleshoot your app in the cloud.

The key is *agility*. You can't ship with agility unless you reduce to an absolute minimum any deployment-to-production issues and dev/test environment issues. Containers (specifically, Docker, as a de facto standard) and managed services were designed specifically for this purpose.

To achieve agility, you also need automated DevOps processes that are based on CI/CD pipelines that release to scalable platforms in the cloud. CI/CD platforms (like Azure DevOps Services or Jenkins) that deploy to a scalable and resilient cloud platform (like Azure App Service or Azure Kubernetes Service) are key technologies for achieving agility in the cloud.

The following list describes the main tenets or practices for Cloud-Optimized applications. Note that you can adopt all or only some of these principles, in a progressive or incremental approach:

- **Containers.** Containers give you the ability to include application dependencies with the application itself. Containerization significantly reduces the number of issues you might encounter when you deploy to production environments or test in staging environments. Ultimately, containers improve the agility of application delivery.

- **Resilient and scalable cloud.** The cloud provides a platform that is managed, elastic, scalable, and resilient. These characteristics are fundamental to gain cost improvements and ship highly available and reliable applications in a continuous delivery. Managed services like managed databases, managed cache as a service (CaaS), and managed storage are fundamental pieces in alleviating the maintenance costs of your application.
- **Monitoring.** You can't have a reliable application without having a good way to detect and diagnose exceptions and application performance issues. You need to get actionable insights through application performance management and instant analytics.
- **DevOps culture and continuous delivery.** Adopting DevOps practices requires a cultural change in which teams no longer work in independent silos. CI/CD pipelines are possible only when there is an increased collaboration between development and IT operations teams, supported by containers and CI/CD tools.

Figure 4-2 shows the main optional pillars of a Cloud-Optimized application. The more pillars you implement, the readier your application will be to succeed in meeting your customers' expectations.

Cloud-Optimized application

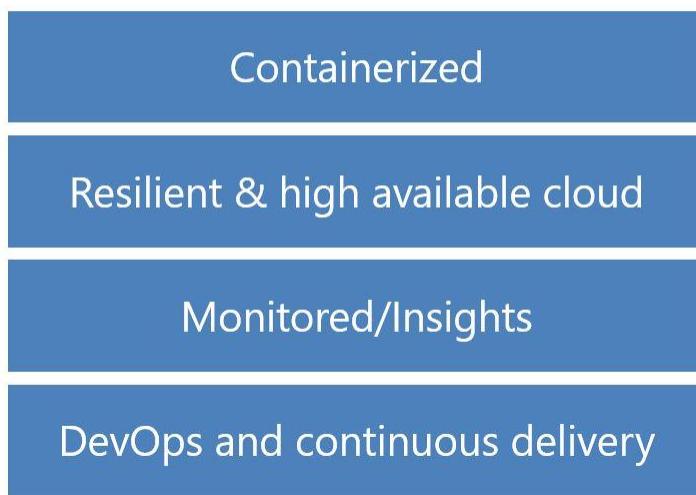


Figure 4-2. Main pillars of a Cloud-Optimized application

To summarize, a Cloud-Optimized application is an approach to building and managing applications that takes advantage of the cloud computing model, while using a combination of containers, managed cloud infrastructure, resilient application techniques, monitoring, continuous delivery, and DevOps, all without the need to rearchitect and recode your existing applications.

Your organization can adopt these technologies and approaches gradually. You don't have to embrace all of them, all at once. You can adopt them incrementally, depending on enterprise priorities and user needs.

Benefits of a Cloud-Optimized application

You can get the following benefits by converting an existing application to a Cloud-Optimized application (without rearchitecting or coding):

- **Lower costs, because the managed infrastructure is handled by the cloud provider.** Cloud-Optimized applications get the benefits of the cloud by using the cloud's out-of-the-box elasticity, autoscale, and high availability. Benefits are related not only to the compute features (VMs and containers), but also depend on the resources in the cloud, like DBaaS, CaaS, and any infrastructure an application might need.
- **Resilient application and infrastructure.** When you migrate to the cloud, you need to embrace transient failures; failures will occur in the cloud. Also, cloud infrastructure and hardware are "replaceable," which increases opportunities for transient downtime. At the same time, inner cloud capabilities and certain application development techniques that implement resiliency and automate recovery make it much easier to recover from unexpected failures in the cloud.
- **Deeper insights into application performance.** Cloud monitoring tools like Azure Application Insights provide visualization for health management, logging, and notifications. Audit logs make applications easy to debug and audit, fundamental for a reliable cloud application.
- **Application portability, with agile deployments.** Containers (either Linux or Windows containers based on Docker Engine) offer the best solution to avoiding a cloud-locked application. By using containers, Docker hosts, and multi-cloud orchestrators, you can easily move from one environment or cloud to another. Containers eliminate the friction that typically occurs in deployments to any environment (stage/test/production).

All of these benefits ultimately provide key cost reductions for your end-to-end application lifecycle.

In the following sections, these benefits are explained in more detail, and are linked to specific technologies.

Microsoft technologies in cloud-optimized applications

The following list describes the tools, technologies, and solutions that are recognized as requirements for Cloud-Optimized apps. You can adopt Cloud-Optimized elements selectively or gradually, depending on your priorities.

- **Cloud infrastructure:** The infrastructure that provides the compute platform, operating system, network, and storage. Microsoft Azure is positioned at this level.
- **Runtime:** This layer provides the environment for the application to run. If you are using containers, this layer usually is based on [Docker Engine](#), running either on Linux hosts or on Windows hosts. ([Windows Containers](#) are supported beginning with Windows Server 2016. Windows Containers is the best choice for existing .NET Framework applications that run on Windows.)

- **Managed cloud:** When you choose a managed cloud option, you can avoid the expense and complexity of managing and supporting the underlying infrastructure, VMs, OS patches, and networking configuration. If you choose to migrate by using IaaS, you are responsible for all of these tasks, and for associated costs. In a managed cloud option, you manage only the applications and services that you develop. The cloud service provider typically manages everything else. Examples of managed cloud services in Azure include [Azure SQL Database](#), [Azure Redis Cache](#), [Azure Cosmos DB](#), [Azure Storage](#), [Azure Database for MySQL](#), [Azure Database for PostgreSQL](#), [Azure Active Directory](#), and managed compute services like [VM scale sets](#), [Azure App Service](#), and [Azure Kubernetes Service](#).
- **Application development:** You can choose from many languages when you build applications that run in containers. This guide focuses on [.NET](#), but, you can develop container-based apps by using other languages, like Node.js, Python, Spring/Java, or Go.
- **Monitoring, telemetry, logging, and auditing:** The ability to monitor and audit applications and containers that are running in the cloud is critical for any Cloud-Optimized application. [Azure Application Insights](#) and [Microsoft Operations Management Suite](#) are the main Microsoft tools that provide monitoring and auditing for Cloud-Optimized apps.
- **Provisioning:** Automation tools help you provision the infrastructure and deploy an application to multiple environments (production, testing, staging). You can use tools like Chef and Puppet to manage an application's configuration and environment. This layer also can be implemented by using simpler and more direct approaches. For example, you can deploy directly by using Azure command-line interface (Azure CLI) tooling, and then use the continuous deployment and release management pipelines in [Azure DevOps Services](#).
- **Application lifecycle:** [Azure DevOps Services](#) and other tools, like Jenkins, are built automation servers that help you implement CI/CD pipelines, including release management.

The next sections of this chapter, and the related walkthroughs, focus specifically on details about the runtime layer (Windows Containers). The guidance describes the ways you can deploy Windows Containers on Windows Server 2016 (and later versions) VMs and Azure Container Instances. It also covers more advanced PaaS platforms like Azure App Service and orchestrator like Azure Kubernetes Service.

Monolithic applications can be Cloud-Optimized

It's important to highlight that monolithic applications (applications that are not based on microservices) *can* be Cloud-Optimized applications. You can build and operate monolithic applications that take advantage of the cloud computing model by using a combination of containers, continuous delivery, and DevOps. If an existing monolithic application is right for your business goals, you can modernize it and make it Cloud-Optimized.

Similarly, if monolithic applications can be Cloud-Optimized applications, other, more complex architectures like N-Tier applications can also be modernized as Cloud-Optimized applications.

What about Cloud-Native applications?

Although [Cloud-Native](#) applications are not the main focus of this guide, it's helpful to have an understanding of this modernization maturity level, and to distinguish it from Cloud-Optimized applications.

Figure 4-3 positions Cloud-Native apps in the application modernization maturity levels:

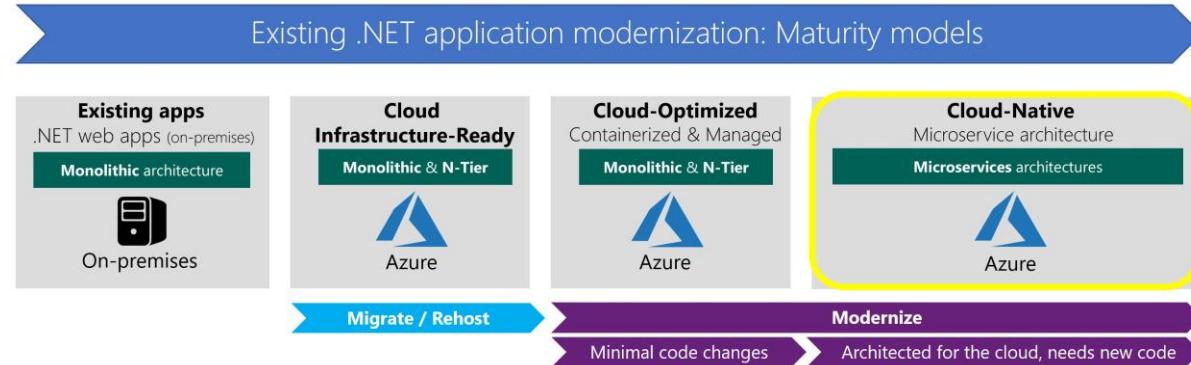


Figure 4-3. Positioning Cloud-Native applications

The Cloud-Native modernization maturity level usually requires new development investments. Moving to the Cloud-Native level typically is driven by business need to modernize applications as much as possible to drastically improve scale in large applications by creating autonomous subsystems (microservices) that can be deployed and scale independently from other areas of the application while lowering costs in the long term and increase evolution agility of those autonomous app's parts that provide significant competitive advantages.

The main pillars of Cloud-Native applications are based on microservices architecture approaches, which can evolve with agility and scale to limits that would be difficult to achieve in a monolithic architecture, deployed to either on-premises or cloud environment.

Figure 4-4 shows the main characteristics of the Cloud-Native model.



Cloud-Native in Azure

Main characteristics

Cloud-native

Microservices

Docker containers

Orchestrators (AKS/Kubernetes, Service Fabric)

Serverless (Azure Functions)

Cloud-resilient

.NET Core (cross-platform, Linux/Windows)

Other languages

Figure 4-4. Cloud-Native characteristics

In addition, you can extend basic modern web apps and cloud-native apps by adding other services, like artificial intelligence (AI), machine learning (ML), and IoT. You might use any of these services to extend any of the possible Cloud-Optimized approaches.

The fundamental difference in applications at the Cloud-Native level is in the application architecture. Cloud-native applications are, by definition, apps that are based on microservices. Cloud-native apps require special architectures, technologies, and platforms, compared to a monolithic web application or traditional N-Tier application.

Cloud-native applications details

Cloud-Native is a more advanced or mature state for large and mission-critical applications. Cloud-Native applications usually require architecture and design that are created from scratch instead of by modernizing existing applications. The key difference between a Cloud-Native application and a simpler Cloud-Optimized web app is the recommendation to use microservices architectures in a cloud-native approach. Cloud-Optimized apps can also be monolithic web apps or N-Tier apps.

The [Twelve-Factor App](#) (a collection of patterns that are closely related to microservices approaches) is also considered a requirement for cloud-native application architectures.

The [Cloud Native Computing Foundation \(CNCF\)](#) is a primary promoter of cloud-native principles. Microsoft is a [member of the CNCF](#).

For detailed guidance on how to design and develop cloud-native applications, read the following free e-books:

- [Architecting Cloud-Native .NET Applications for Azure](#)
- [.NET Microservices: Architecture for containerized .NET applications](#).

The most important factor to consider if you migrate a full application to the cloud-native model is that you must rearchitect to a microservices-based architecture. This approach clearly requires a significant investment in development because of the large refactoring process involved. This option usually is chosen for mission-critical applications that need new levels of scalability and long-term agility. But, you could start moving toward cloud-native by adding microservices for just a few new scenarios, and eventually refactor the application fully as microservices. This step is an incremental approach that is the best option for some scenarios.

What about microservices?

Understanding microservices and how they work is important when you are considering cloud-native applications for your organization.

The microservices architecture is an advanced approach that you can use for applications that are created from scratch or when you evolve existing applications toward cloud-native applications. You can start by adding a few microservices to existing applications to learn about the new microservices paradigms. But clearly, you need to architect and code, especially for this type of architectural approach.

However, microservices are not mandatory for any new or modern application. Microservices are not a "magic bullet," and they aren't the single, best way to create every application. How and when you use microservices depends on the type of application that you need to build.

The microservices architecture is becoming the preferred approach for distributed and large or complex mission-critical applications that are based on multiple, independent subsystems in the form of autonomous services. In a microservices-based architecture, an application is built as a collection of services that can be independently developed, tested, versioned, deployed, and scaled. This approach can include any related, autonomous database per microservice.

For a detailed look at a microservices architecture that you can implement by using .NET, see the downloadable PDF e-book [.NET microservices: Architecture for containerized .NET applications](#). The guide also is available [online](#).

But even in scenarios in which microservices offer powerful capabilities-independent deployment, strong subsystem boundaries, and technology diversity-they also raise many new challenges. The challenges are related to distributed application development, such as fragmented and independent data models; achieving resilient communication between microservices; the need for eventual consistency; and operational complexity. Microservices introduce a higher level of complexity compared to traditional monolithic applications.

Because of the complexity of a microservices architecture, only specific scenarios and certain application types are suitable for microservice-based applications. These include large and complex applications that have multiple, evolving subsystems. In these cases, it's worth investing in a more complex software architecture, for increased long-term agility and more efficient application maintenance. But for less complex scenarios, it might be better to continue with a monolithic application approach or simpler N-Tier approaches.

As a final note, even at the risk of being repetitive about this concept, you shouldn't look at using microservices in your applications as "all-in or nothing at all." You can extend and evolve existing monolithic applications by adding new, small scenarios based on microservices. You don't need to start from scratch to start working with a microservices architecture approach. In fact, we recommend that you evolve from using an existing monolithic or N-Tier application by adding new scenarios. Eventually, you can break down the application into autonomous components or microservices. You can start evolving your monolithic applications in a microservices direction, step by step.

In any case, the rest of this present guidance focuses most of all on "no microservices-based apps" because this guidance is mainly targeting the modernization of existing apps that usually have monolithic or N-Tier architectures.

Deploy existing .NET apps as Windows containers

Deployments that are based on Windows Containers are applicable to Cloud-Optimized applications and Cloud-Native applications.

However, in this guide and especially in the following sections, it mostly focuses on using Windows Containers for *Cloud-Optimized* applications where you don't need to rearchitect your application.

What are containers? (Linux or Windows)

Containers are a way to wrap up an application into its own isolated package. In its container, the application is not affected by applications or processes that exist outside of the container. Everything the application depends on to run successfully as a process is inside the container. Wherever the container might move, the requirements of the application will always be met, in terms of direct dependencies, because it is bundled with everything that it needs to run (library dependencies, runtimes, and so on).

The main characteristic of a container is that it makes the environment the same across different deployments because the container itself comes with all the dependencies it needs. You can debug the application on your machine, and then deploy it to another machine, with the same environment guaranteed.

A container is an instance of a container image. A container image is a way to package an app or service (like a snapshot), and then deploy it in a reliable and reproducible way. You could say that Docker is not only a technology—it's also a philosophy and a process.

As containers daily become more common, they are becoming an industry-wide "unit of deployment."

Benefits of containers (Docker Engine on Linux or Windows)

Building applications by using containers—which also might be defined as lightweight building blocks—offers a significant increase in agility for building, shipping, and running any application, across any infrastructure.

With containers, you can take any app from development to production with little or no code change, thanks to Docker integration across Microsoft developer tools, operating systems, and the cloud.

When you deploy to plain VMs, you probably already have a method in place for deploying ASP.NET apps to your VMs. It's likely, though, that your method involves multiple manual steps or complex automated processes by using a deployment tool like Puppet, or a similar tool. You might need to perform tasks like modifying configuration items, copying application content between servers, and running interactive setup programs based on .msi setups, followed by testing. All those steps in the deployment add time and risk to deployments. You will get failures whenever a dependency is not present in the target environment.

In Windows Containers, the process of packaging applications is fully automated. Windows Containers is based on the Docker platform, which offers automatic updates and rollbacks for container deployments. The main improvement you get from using the Docker engine is that you create images, which are like snapshots of your application, with all its dependencies. The images are Docker images (a Windows container image, in this case). The images run ASP.NET apps in containers, without going back to source code. The container snapshot becomes the unit of deployment.

Many organizations are containerizing existing monolithic applications for the following reasons:

- **Release agility through improved deployment.** Containers offer a consistent deployment contract between development and operations. When you use containers, you won't hear developers say, "It works on my machine, why not in production?" They can say, "It runs as a container, so it will run in production." The packaged application, with all its dependencies, can be executed in any supported container-based environment. It will run the way it was intended to run in all deployment targets (dev, QA, staging, production). Containers eliminate most frictions when they move from one stage to the next, which greatly improves deployment, and you can ship faster.
- **Cost reductions.** Containers lead to lower costs, either by the consolidation and removal of existing hardware, or from running applications at a higher density per unit of hardware.
- **Portability.** Containers are modular and portable. Docker containers are supported on any server operating system (Linux and Windows), in any major public cloud (Microsoft Azure, Amazon AWS, Google, IBM), and in on-premises and private or hybrid cloud environments.
- **Control.** Containers offer a flexible and secure environment that's controlled at the container level. A container can be secured, isolated, and even limited by setting execution constraint policies on the container. As detailed in the section about Windows Containers, Windows Server 2016 and Hyper-V containers offer additional enterprise support options.

Significant improvements in agility, portability, and control ultimately lead to significant cost reductions when you use containers to develop and maintain applications.

What is Docker?

[Docker](#) is an [open-source project](#) that automates the deployment of applications as portable, self-sufficient containers that can run in the cloud or on-premises. Docker is also a [company](#) that promotes and evolves this technology. The company works in collaboration with cloud, Linux, and Windows vendors, including Microsoft.

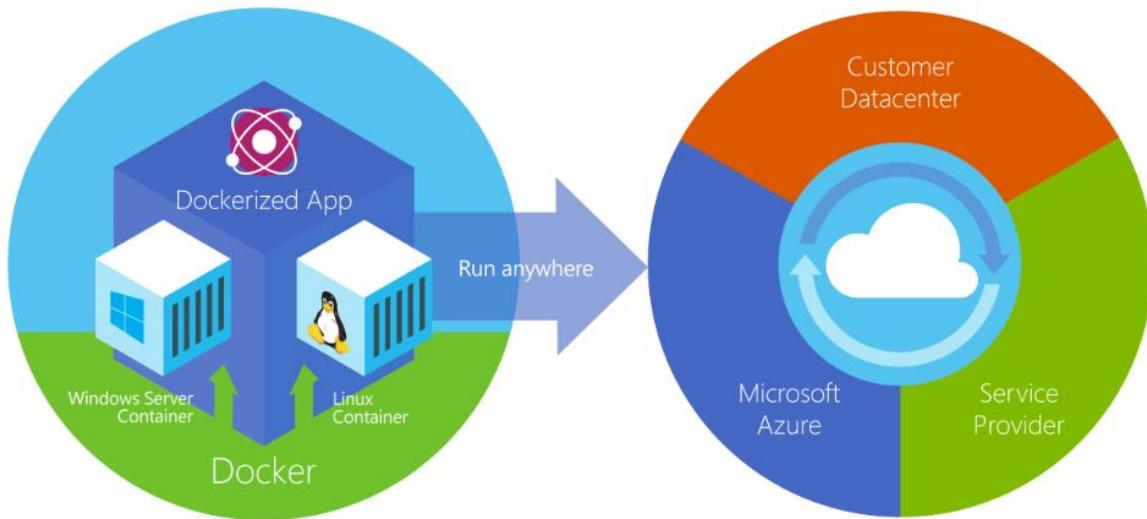


Figure 4-6. Docker deploys containers at all layers of the hybrid cloud

To someone familiar with virtual machines, containers might appear to be remarkably similar. A container runs an operating system, has a file system, and can be accessed over a network, just like a physical or virtual computer system. However, the technology and concepts behind containers are vastly different from virtual machines. From a developer point of view, a container must be treated more like a single process. In fact, a container has a single entry point for one process.

Docker containers (for simplicity, *containers*) can run natively on Linux and Windows. When running regular containers, Windows containers can run only on Windows hosts (a host server or a VM), and Linux containers can run only on Linux hosts. However, in recent versions of Windows Server and Hyper-V containers, a Linux container can also run natively on Windows Server by using the Hyper-V isolation technology that currently is available only in Windows Server Containers.

In the near future, mixed environments that have both Linux and Windows containers will be possible and even common.

Benefits of Windows Containers for your existing .NET applications

The benefits of using Windows Containers are fundamentally the same benefits you get from containers in general. Using Windows Containers is about greatly improving agility, portability, and control.

Existing .NET applications refer to those applications that were created using the .NET Framework. For example, they might be traditional ASP.NET web applications—they don't use .NET Core or .NET 5+, which is newer and runs cross-platform on Linux, Windows, and MacOS.

The main dependency in the .NET Framework is Windows. It also has secondary dependencies, like IIS, and System.Web in traditional ASP.NET.

A .NET Framework application must run on Windows, period. If you want to containerize existing .NET Framework applications and you can't or don't want to invest in a migration to .NET Core or later ("If it

works properly, don't migrate it"), the only choice you have for containers is to use Windows Containers.

So, one of the main benefits of Windows Containers is that they offer you a way to modernize your existing .NET Framework applications that are running on Windows-through containerization. Ultimately, Windows Containers gets you the benefits that you are looking for by using containers-agility, portability, and better control.

Choose an OS to target with .NET-based containers

Given the diversity of operating systems that are supported by Docker, as well as the differences between .NET Framework and .NET Core, you should target a specific OS and specific versions based on the framework you are using.

For Windows, you can use Windows Server Core or Windows Nano Server. These Windows versions provide different characteristics (like IIS versus a self-hosted web server like Kestrel) that might be needed by .NET Framework or .NET applications.

For Linux, multiple distros are available and supported in official .NET Docker images (like Debian).

Figure 4-7 shows OS versions that you can target, depending on the app's version of the .NET Framework.

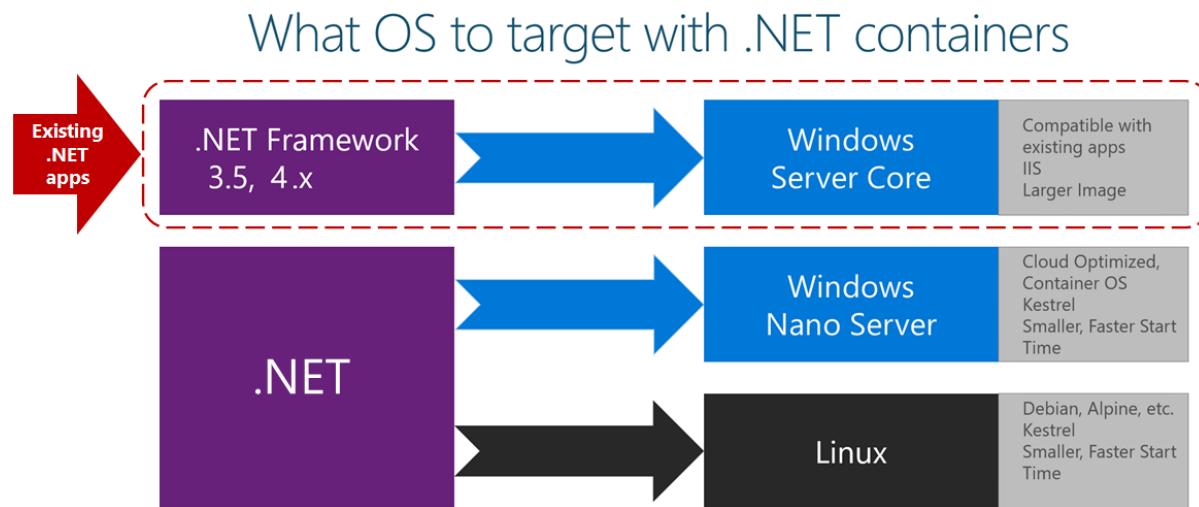


Figure 4-7. Operating systems to target based on .NET Framework version

In migration scenarios for existing or legacy applications that are based on .NET Framework applications, the main dependencies are on Windows and IIS. Your only option is to use Docker images based on Windows Server Core and the .NET Framework.

When you add the image name to your Dockerfile file, you can select the operating system and version by using a tag, as in the following examples for .NET Framework-based Windows container images:

Tag	System and version
mcr.microsoft.com/dotnet/framework/runtime:4.x-windowsservercore-20H2	.NET Framework 4.x on Windows Server Core
mcr.microsoft.com/dotnet/framework/aspnet:4.x-windowsservercore-20H2	.NET Framework 4.x with additional ASP.NET customization, on Windows Server Core

For .NET (cross-platform for Linux and Windows), the tags would look like the following:

Tag	System and version
mcr.microsoft.com/dotnet/runtime:5.0	.NET runtime-only on Linux
mcr.microsoft.com/dotnet/runtime:5.0-nanoserver-20H2	.NET runtime-only on Windows Nano Server

Multi-arch images

Since 2017, Docker has had a feature called [multi-arch](#) images. .NET Docker images can use multi-arch tags. Your Dockerfile files no longer need to define the operating system that you are targeting. The multi-arch feature allows a single tag to be used across multiple machine configurations. For instance, with multi-arch, you can use one common tag: **mcr.microsoft.com/dotnet/runtime:5.0**. If you pull that tag from a Linux container environment, you get the Debian-based image. If you pull that tag from a Windows container environment, you get the Nano Server-based image.

For .NET Framework images, because the traditional .NET Framework supports only Windows, you cannot use the multi-arch feature.

Windows container types

Like Linux containers, Windows Server containers are managed by using Docker Engine. Unlike Linux containers, Windows containers include two different container types, or run times-Windows Server containers and Hyper-V isolation.

Windows Server containers: Provides application isolation through process and namespace isolation technology. A Windows Server container shares a kernel with the container host and all containers that are running on the host. These containers do not provide a hostile security boundary and should not be used to isolate untrusted code. Because of the shared kernel space, these containers require the same kernel version and configuration.

Hyper-V isolation: Expands on the isolation provided by Windows Server Containers by running each container on a highly optimized VM. In this configuration, the kernel of the container host is not shared with other containers on the same host. These containers are designed for hostile multitenant hosting, with the same security assurances of a VM. Because these containers don't share the kernel with the host or other containers on the host, they can run kernels with different versions and configurations (with supported versions). For example, all Windows containers on Windows 10 use Hyper-V isolation to utilize the Windows Server kernel version and configuration.

Running a container on Windows with or without Hyper-V isolation is a run-time decision. You might choose to create the container with Hyper-V isolation initially, and at run time, choose to run it as a Windows Server container instead.

Additional resources

- **Windows Containers documentation**
<https://docs.microsoft.com/virtualization/windowscontainers/>
- **Windows Containers fundamentals**
<https://docs.microsoft.com/virtualization/windowscontainers/about/>
- **Infographic: Microsoft and containers**
<https://info.microsoft.com/rs/157-GQE-382/images/Container%20infographic%201.4.17.pdf>

The container ecosystem in Azure

In previous sections, it's been explained what the benefits of Docker containers are as well as details on the specific container images for .NET applications. All that generic information is fundamental in order to develop or containerize an application. However, when thinking about the production deployment environment or even QA and Dev/Test environments, Microsoft Azure provides an open and broad variety of choices, a full container ecosystem in the cloud (shown in the diagram below). Depending on your specific application's needs, you should choose one or another Azure product.

Azure container ecosystem

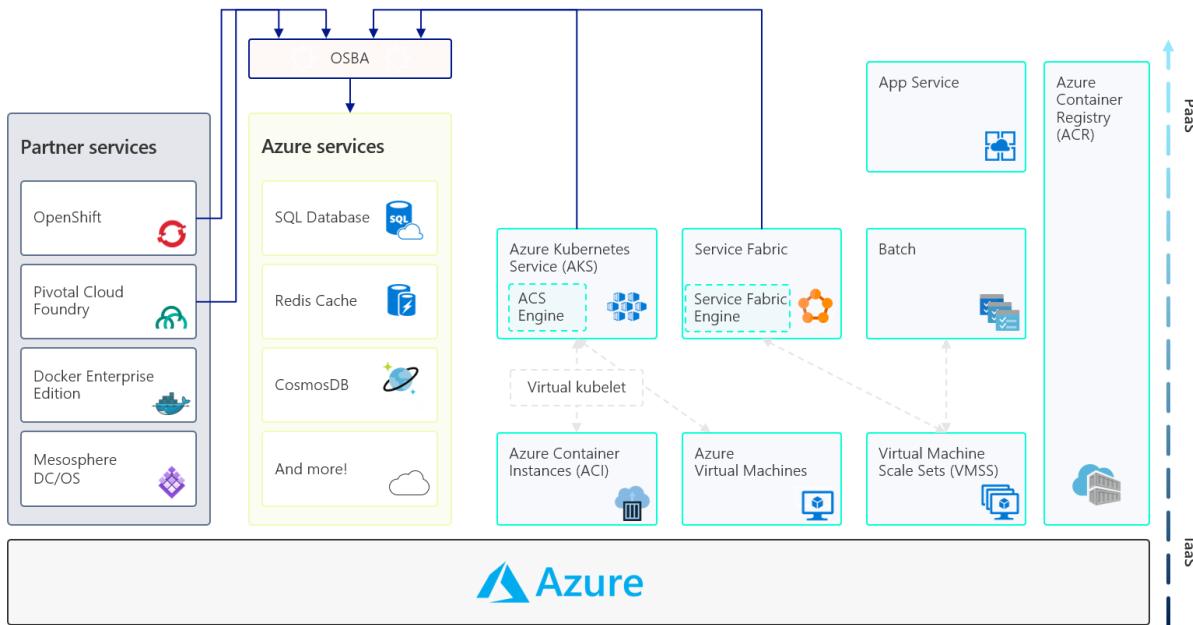


Figure 4-7.5. The container ecosystem in Azure

From the container ecosystem in Azure, the following products supporting containers that are considered infrastructure:

- **Azure Container Instances (ACI)**
- **Azure Virtual Machines** (With container's support)
- **Azure Virtual Machine Scale Sets** (With container's support)

From those three, ACI provides a great benefit, which is the fact that you don't need to maintain the underlying OS, no need for you to upgrade/patch, etc. but ACI still is positioned in the infrastructure level, as better explained in the upcoming sections of this book.

The products in Azure supporting containers that are at the same time positioned more in the PaaS (Platform as a Service) level are:

- **Azure App Service**
- **Azure Kubernetes Service (AKS and ACS)**
- **Azure Batch**

Then, Azure Container Registry is a high scalable container registry hosted in Azure that you can use from all the previous products when registering and deploying your custom container images.

In addition, from your containers, you can consume other managed services in Azure like Azure SQL Database, Azure Redis cache, Azure Cosmos DB, etc. plus there are third-party solutions/platforms

available in Azure Marketplace like Cloud Foundry and OpenShift where you can also use containers within Azure.

In the next sections, you can explore Microsoft's recommendations on when to use each of those Azure products and solutions specifically when targeting Windows Containers.

When not to deploy to Windows Containers

Some Windows technologies are not supported by Windows Containers. In those cases, you still need to migrate to the standards VMs, usually with just Windows and IIS.

For more information, see the list of [Applications not supported by Windows containers](#).

Additional resources

- **Virtual machines and containers in Azure**

<https://azure.microsoft.com/overview/containers/>

When to deploy Windows Containers in your on-premises IaaS VM infrastructure

- Your organization might not be ready to move to the cloud, or it might not be able to move to the cloud for a business reason. But, you can still get the benefits of using Windows Containers in your own datacenters.
- You might have other artifacts that are being used on-premises, and which might slow you down when you try to move to the cloud. For example, security or authentication dependencies with on-premises Windows Server Active Directory, or any other on-premises asset.
- If you start using Windows Containers today, you can make a phased migration to the cloud tomorrow from a much better position. Windows Containers is becoming a unit of deployment for any cloud, with no lock-in.

When to deploy Windows Containers to Azure VMs (IaaS cloud)

If your organization is using Azure VMs, even if you are also using Windows Containers, you are still dealing with IaaS. That means that dealing with infrastructure operations, VM OS patches, and infrastructure complexity for highly scalable applications when you need to deploy to multiple VMs in a load-balanced infrastructure. The main scenarios for using Windows Containers in an Azure VM are:

- **Dev/test environment:** A VM in the cloud is perfect for development and testing in the cloud. You can rapidly create or stop the environment depending on your needs.

- **Small and medium scalability needs:** In scenarios where you might need just a couple of VMs for your production environment, managing a few VMs might be affordable until you can move to more advanced PaaS environments, like orchestrators.
- **Production environment with existing deployment tools:** You might be moving from an on-premises environment in which you have invested in tools to make complex deployments to VMs or bare-metal servers (like Puppet or similar tools). To move to the cloud with minimal changes to production environment deployment procedures, you might continue to use those tools to deploy to Azure VMs. However, you'll want to use Windows Containers as the unit of deployment to improve the deployment experience.

When to deploy Windows Containers to Azure Container Instances (ACI)

The main value proposition of Azure Container Instances is that you can right away deploy containers to it and you don't need to maintain that environment, you don't need to upgrade/patch the underlying operating system or VMs, all that is transparent and you just deploy containers into a ready-to-use environment.

The reasons and scenarios when you would want to use ACI are similar to the main scenarios when you use Azure VMs with containers, so basically, the main scenarios for using Azure Container Instances are:

- **Dev/Test scenarios**
- **Task automation**
- **CI/CD agents**
- **Small/scale batch processing**
- **Simple web apps**

The simple web apps scenario is a fair scenario for ACI but take into account that since in ACI you can only have a single container instance per container image, you won't have high availability and only have limited scalability.

However, even when ACI is considered infrastructure because it just provides single container instances, there is a huge benefit compared to regular Azure VMs with Windows Server. With ACI, you just deploy the containers into a self-maintained environment and you just pay for those containers. You don't need to maintain/update/patch VMs, so it is a much better platform for most scenarios where you might be using VMs with containers. Using ACI is straight forward, you just deploy a container, there's no need to create a VM environment you just deploy containers.

The main benefits of Azure Container Instances (ACI) are:

- Run containers without managing servers
- Increase agility with containers on demand
- Deploy containers to the cloud with unprecedented simplicity and speed—with a single command.

- Secure applications with hypervisor isolation

In short, with ACI you can develop apps fast without managing virtual machines or having to learn new tools. It's just your application, in a container, running in the cloud.

When to deploy Windows Containers to Azure Kubernetes Service (AKS)

Azure Kubernetes Service (AKS) simplifies deploying a managed Kubernetes cluster in Azure by offloading the operational overhead to Azure. You get an open solution that offers portability both for your containers and for your application configuration. You select the size, the number of hosts, and the orchestrator tools. Azure Kubernetes Service handles the infrastructure for you. It also supports Windows Server containers alongside Linux containers. For more details on Windows OS support and the feature differences between Windows and Linux, read the [FAQ](#).

The main benefits of Azure Kubernetes Service (AKS) are:

- Identity and security management
- Integrated logging and monitoring
- Cluster node and pod scaling.
- Ingress with HTTP application routing
- AKS supports the Docker image format. For private storage of your Docker images, you can integrate AKS with Azure Container Registry (ACR).

If you are already working with open-source orchestrators like Kubernetes, Docker Swarm, or DC/OS, you don't need to change your existing management practices to move container workloads to the cloud. Use the application management tools that you're already familiar with and connect via the standard API endpoints for the orchestrator of your choice.

Choosing Azure compute platforms for container-based applications

As you have noticed after reading the previous sections, Azure is an open cloud that offers multiple choices. You can use the best fit for your needs, however, it also surfaces questions about what product/technology you should use for your containerized applications.

As a *by-default* recommendation, the following is the main criteria recommended in this guidance:

- **Single monolithic app:** Choose Azure App Service
- **N-Tier app:** Choose orchestrators such as Azure Kubernetes Service (AKS) or App Service if you have a single or a few back-end services
- **Microservices:** Choose AKS or Azure Web Apps for Containers
- **Serverless functions & event handlers:** Choose Azure Functions
- **Large-scale Batch:** Choose Azure Batch

However, this recommendation should be taken with a pinch of salt, as the product's selection will depend on your specific application's needs and characteristics. Not all applications are the same even when initially they might look similar types.

After a deeper analysis of the application's needs, the product selected could be different. But, as a starting point, it is good to have initial guidance from where you can start evaluating and testing based on certain priority.

Additional resources

- **Choose an Azure compute service for your application**

<https://docs.microsoft.com/azure/architecture/guide/technology-choices/compute-decision-tree>

Build resilient services ready for the cloud: Embrace transient failures in the cloud

Resiliency is the ability to recover from failures and continue to function. Resiliency is not about avoiding failures, but accepting the fact that failures will occur, and then responding to them in a way that avoids downtime or data loss. The goal of resiliency is to return the application to a fully functioning state after a failure.

Your application is ready for the cloud when, at a minimum, it implements a software-based model of resiliency, rather than a hardware-based model. Your cloud application must embrace the partial failures that will certainly occur. Design or partially refactor your application to achieve resiliency with expected partial failures. It should be designed to cope with partial failures, like transient network outages and nodes, or VMs crashing in the cloud. Even containers being moved to a different node within an orchestrator cluster can cause intermittent short failures within the application.

Handling partial failure

In a cloud-based application, there's an ever-present risk of partial failure. For instance, a single website instance or a container might fail, or it might be unavailable or unresponsive for a short time. Or, a single VM or server might crash.

Because clients and services are separate processes, a service might not be able to respond in a timely manner to a client's request. The service might be overloaded and respond slowly to requests, or it might not be accessible for a short time because of network issues.

For example, consider a monolithic .NET application that accesses a database in Azure SQL Database. If the Azure SQL database or any other third-party service is unresponsive for a brief time (an Azure SQL database might be moved to a different node or server, and be unresponsive for a few seconds), when the user tries to do any action, the application might crash and show an exception at the same moment.

A similar scenario might occur in an app that consumes HTTP services. The network or the service itself might not be available in the cloud during a short, transient failure.

A resilient application like the one shown in Figure 4-9 should implement techniques like “retries with exponential backoff” to give the application an opportunity to handle transient failures in resources. You also should use “circuit breakers” in your applications. A circuit breaker stops an application from trying to access a resource when it’s actually a long-term failure. By using a circuit breaker, the application avoids provoking a denial of service to itself.

Resilient communication in cloud applications

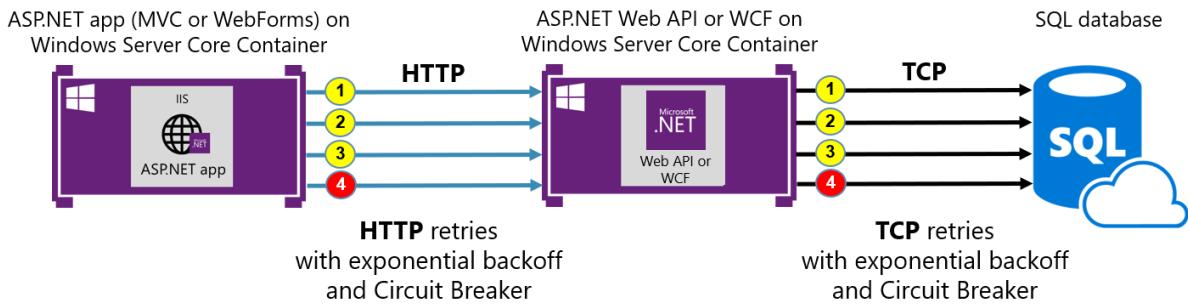


Figure 4-9. Partial failures handled by retries with exponential backoff

You can use these techniques both in HTTP resources and in database resources. In Figure 4-9, the application is based on a 3-tier architecture, so you need these techniques at the services level (HTTP) and at the data tier level (TCP). In a monolithic application that uses only a single app tier in addition to the database (no additional services or microservices), handling transient failures at the database connection level might be enough. In that scenario, just a particular configuration of the database connection is required.

When implementing resilient communications that access the database, depending on the version of .NET you are using, it can be straightforward (for example, [with Entity Framework 6 or later](#). It's just a matter of configuring the database connection). Or, you might need to use additional libraries like the [Transient Fault Handling Application Block](#) (for earlier versions of .NET), or even implement your own library.

When implementing HTTP retries and circuit breakers, the recommendation for .NET is to use the [Polly](#) library, which targets .NET Standard 1.1 (coverage: .NET Core 1.0, Mono, Xamarin, UWP, WP8.1+) and .NET Standard 2.0+ (coverage: .NET Core 2.0+, .NET Core 3.0, and later Mono, Xamarin and UWP targets). The nuget package also includes direct targets for .NET Framework 4.6.1 and 4.7.2.

To learn how to implement strategies for handling partial failures in the cloud, see the following references.

Additional resources

- **Implementing resilient communication to handle partial failure**

<https://docs.microsoft.com/dotnet/architecture/microservices/implement-resilient-applications/partial-failure-strategies>

- **Entity Framework connection resiliency and retry logic (version 6 and later)**
<https://docs.microsoft.com/ef/ef6/fundamentals/connection-resiliency/retry-logic>
- **The Transient Fault Handling Application Block**
- [https://docs.microsoft.com/previous-versions/msp-n-p/hh680934\(v=pandp.50\)](https://docs.microsoft.com/previous-versions/msp-n-p/hh680934(v=pandp.50))
- **Polly library for resilient HTTP communication**
<https://github.com/App-vNext/Polly>

Modernize your apps with monitoring and telemetry

When you run an application in production, it's critical that you have insights about how your application is performing. Is it performing at a high level? Are users getting errors, or is the application stable and reliable? You need rich performance monitoring, powerful alerting, and dashboards to help ensure that your application is available and performing as expected. You also need to be able to see quickly if there's a problem, determine how many customers are affected, and perform a root-cause analysis to find and fix the issue.

Monitor your application with Application Insights

Application Insights is an extensible Application Performance Management (APM) service for web developers who work on multiple platforms. Use it to monitor your live web application. Application Insights automatically detects performance anomalies. It includes powerful analytics tools to help you diagnose issues, and to help you understand what users actually do with your app. Application Insights is designed to help you continuously improve performance and usability. It works for apps on a wide variety of platforms, including .NET, Node.js, and J2EE, whether hosted on-premises or in the cloud. Application Insights integrates with your DevOps processes, and has connection points to various development tools.

Figure 4-10 shows an example of how Application Insights monitors your application and how it surfaces those insights to a dashboard.

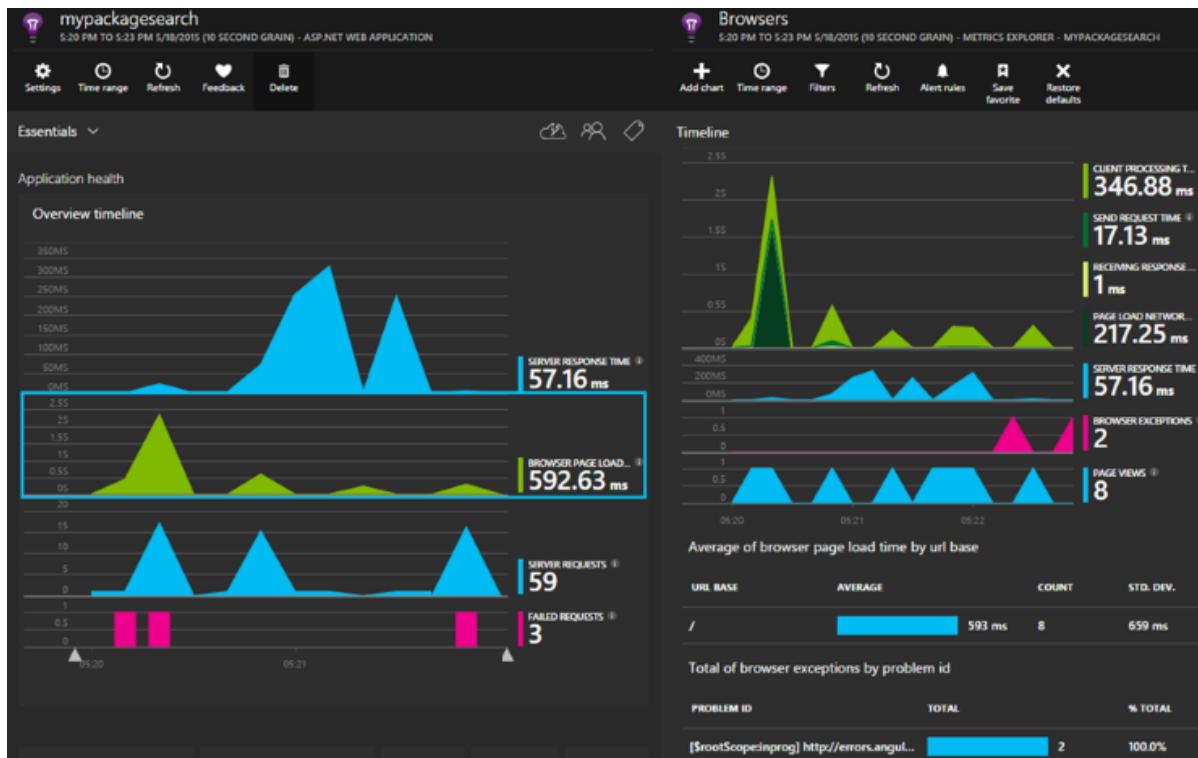


Figure 4-10. Application Insights monitoring dashboard

Monitor your Docker infrastructure with Log Analytics and its Container Monitoring solution

Azure Log Analytics is part of the [Microsoft Azure overall monitoring solution](#). It's also a service in [Operations Management Suite \(OMS\)](#). Log Analytics monitors cloud and on-premises environments (OMS for on-premises) to help maintain availability and performance. It collects data generated by resources in your cloud and on-premises environments and from other monitoring tools to provide analysis across multiple sources.

In relation to Azure infrastructure logs, Log Analytics, as an Azure service, ingests log and metric data from other Azure services (via [Azure Monitor](#)), Azure VMs, Docker containers, and on-premises or other cloud infrastructures. Log Analytics offers flexible log search and out-of-the box analytics on top of this data. It provides rich tools that you can use to analyze data across sources, it allows complex queries across all logs, and it can proactively alert based on specified conditions. You can even collect custom data in the central Log Analytics repository, where you can query and visualize it. You can also take advantage of the Log Analytics built-in solutions to immediately gain insights into the security and functionality of your infrastructure.

You can access Log Analytics through the OMS portal or the Azure portal, which run in any browser, and provide you with access to configuration settings and multiple tools to analyze and act on collected data.

The [Container Monitoring solution](#) in Log Analytics helps you view and manage your Docker and Windows Container hosts in a single location. The solution shows which containers are running, what

container image they're running, and where containers are running. You can view detailed audit information, including commands that are being used with containers. You can also troubleshoot containers by viewing and searching centralized logs, without needing to remotely view Docker or Windows hosts. You can find containers that might be noisy and consuming excess resources on a host. Additionally, you can view centralized CPU, memory, storage, and network usage, and performance information, for containers. On computers running Windows, you can centralize and compare logs from Windows Server, Hyper-V, and Docker containers. The solution supports the following container orchestrators:

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

Figure 4-11 shows the relationships between various container hosts and agents and OMS.

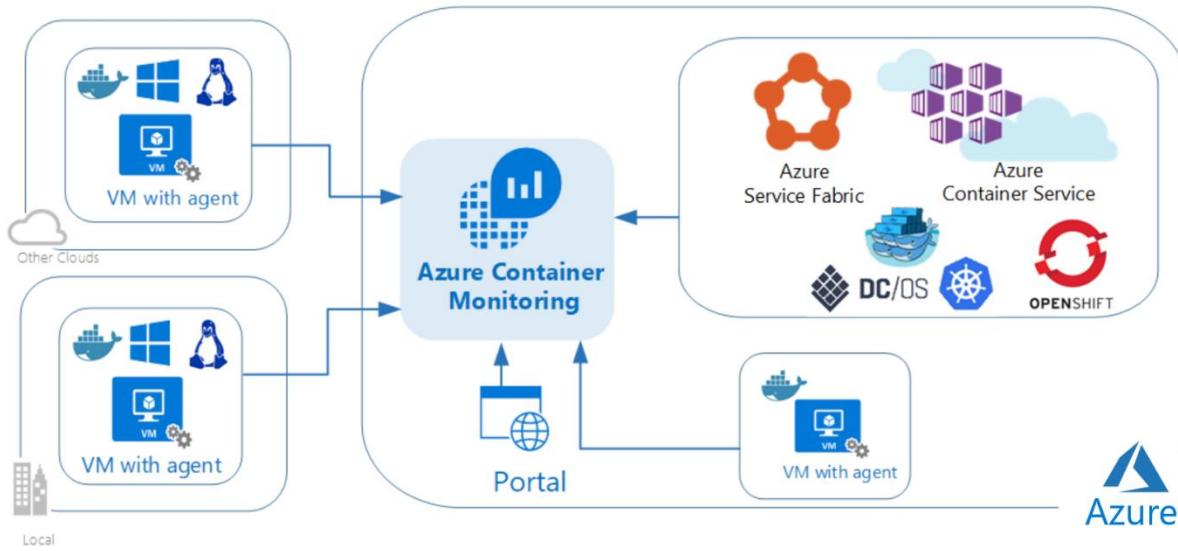


Figure 4-11. Log Analytics Container Monitoring solution

You can use the Log Analytics Container Monitoring solution to:

- See information about all container hosts in a single location.
- Know which containers are running, what image they're running, and where they're running.
- See an audit trail for actions on containers.
- Troubleshoot by viewing and searching centralized logs without remote login to the Docker hosts.
- Find containers that might be “noisy neighbors,” and be consuming excess resources on a host.
- View centralized CPU, memory, storage, and network usage, and performance information, for containers.

Additional resources

- **Overview of monitoring in Microsoft Azure**

<https://docs.microsoft.com/azure/azure-monitor/overview>

- **What is Application Insights?**

<https://docs.microsoft.com/azure/azure-monitor/app/app-insights-overview>

- **What is Log Analytics?**

<https://docs.microsoft.com/azure/log-analytics/log-analytics-overview>

- **Container Monitoring solution in Azure Monitor**

<https://docs.microsoft.com/azure/azure-monitor/insights/containers>

- **Overview of Azure Monitor**

<https://docs.microsoft.com/azure/azure-monitor/overview>

- **What is Operations Management Suite (OMS)?**

<https://docs.microsoft.com/azure/operations-management-suite/operations-management-suite-overview>

Modernize your app's lifecycle with CI/CD pipelines and DevOps tools in the cloud

Today's businesses need to innovate at a rapid pace to be competitive in the marketplace. Delivering high-quality, modern applications requires DevOps tools and processes that are critical to implement this constant cycle of innovation. With the right DevOps tools, developers can streamline continuous deployment and get innovative applications into the hands of users more quickly.

Although continuous integration and deployment practices are well established, the introduction of containers introduces new considerations, particularly when you are working with multi-container applications.

Azure DevOps Services supports continuous integration and deployment of multi-container applications to various environments through the official Azure DevOps Services deployment tasks:

- [Deploy to an Azure Web App for Containers](#)
- [Deploy to Azure Kubernetes Service](#)

But you can also deploy to [Docker Swarm](#) or DC/OS by using Azure DevOps Services script-based tasks.

To continue facilitating deployment agility, these tools provide excellent dev-to-test-to-production deployment experiences for container workloads, with a choice of development and CI/CD solutions.

Figure 4-12 shows a continuous deployment pipeline that deploys to a Kubernetes cluster in Azure Container Service.

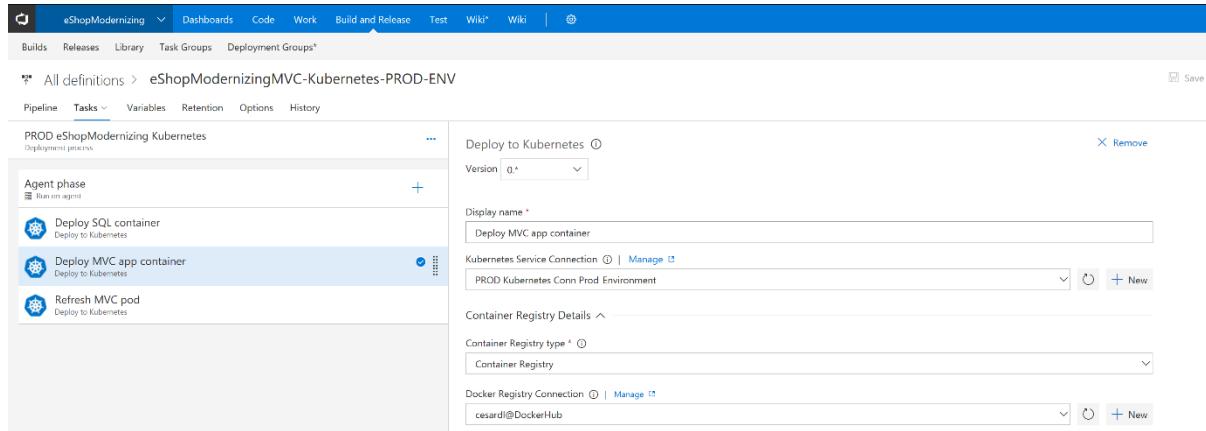


Figure 4-12. Azure DevOps Services continuous deployment pipeline, deploying to a Kubernetes cluster

Migrate to hybrid cloud scenarios

Some organizations and enterprises can't migrate some of their applications to public clouds like Microsoft Azure or any other public cloud due to regulations or their own policies. However, it's likely that any organization might benefit from having some of their applications in the public cloud and other applications on-premises. But a mixed environment can lead to excessive complexity in environments due to different platforms and technologies used in public clouds versus on-premises environments.

Microsoft provides the best hybrid cloud solution, one in which you can optimize your existing assets on-premises and in the public cloud, while you ensure consistency in an Azure hybrid cloud. You can maximize existing skills, and get a flexible, unified approach to building apps that can run in the cloud or on-premises, thanks to Azure Stack (on-premises) and Azure (public cloud).

When it comes to security, you can centralize management and security across your hybrid cloud. You can get control over all assets, from your datacenter to the cloud, by providing single sign-on to on-premises and cloud apps. You accomplish this functionality by extending Active Directory to a hybrid cloud, and by using identity management.

Finally, you can distribute and analyze data seamlessly, use the same query languages for cloud and on-premises assets, and apply analytics and deep learning in Azure to enrich your data, regardless of its source.

Azure Stack

Azure Stack is a hybrid cloud platform that lets you deliver Azure services from your organization's datacenter. Azure Stack is designed to support new options for your modern applications in key scenarios, like edge and unconnected environments, or meeting specific security and compliance requirements.

Figure 4-13 shows an overview of the true hybrid cloud platform that Microsoft offers.

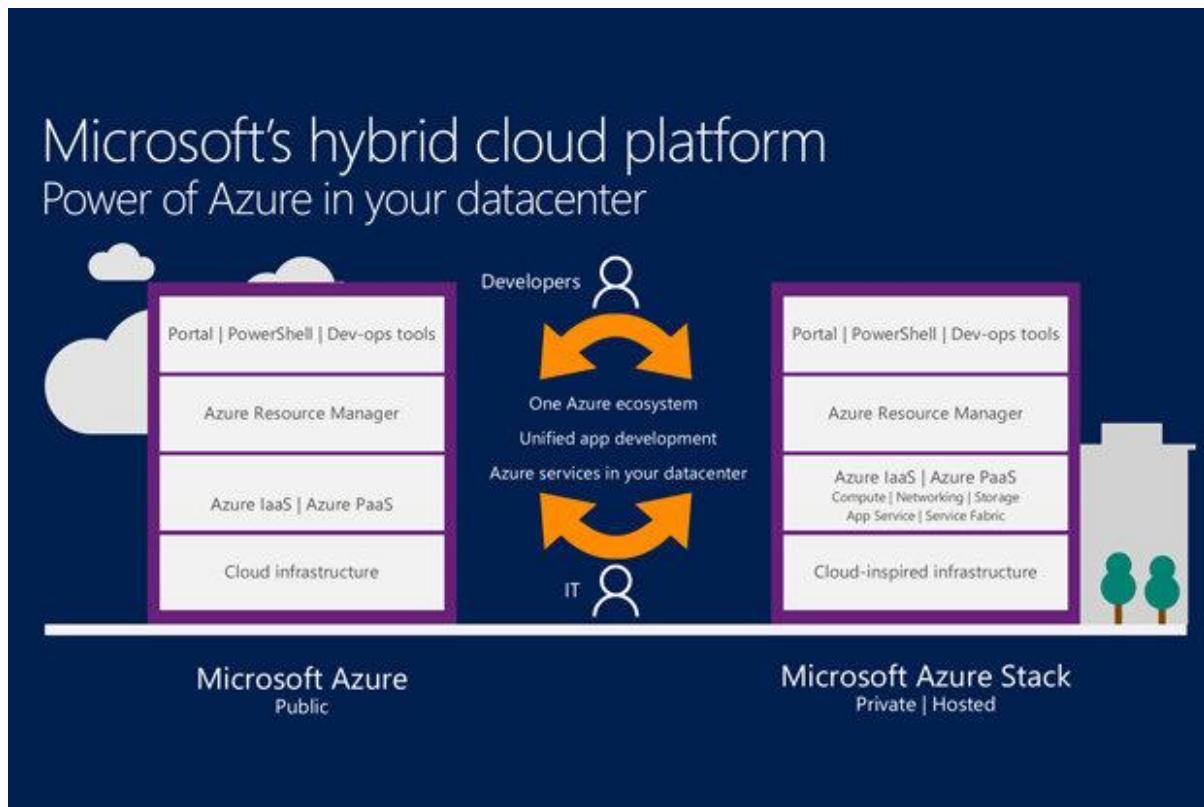


Figure 4-13. Microsoft hybrid cloud platform with Azure Stack and Azure

Azure Stack is offered in two deployment options, to meet your needs:

- Azure Stack integrated systems
- Azure Stack Development Kit

Azure Stack integrated systems

Azure Stack integrated systems are offered through a partnership of Microsoft and hardware partners. The partnership creates a solution that offers cloud-paced innovation that is balanced with simplicity in management. Because Azure Stack is offered as an integrated system of hardware and software, you get the right amount of flexibility and control, while still adopting innovation from the cloud. Azure Stack integrated systems range in size from 4 to 12 nodes, and are jointly supported by the hardware partner and Microsoft. Use Azure Stack integrated systems to implement new scenarios for your production workloads.

Azure Stack Development Kit

Microsoft Azure Stack Development Kit is a single-node deployment of Azure Stack, which you can use to evaluate and learn about Azure Stack. You can also use Azure Stack Development Kit as a developer environment, where you can develop using APIs and tooling that are consistent with Azure. Azure Stack Development Kit is not intended to be used as a production environment.

Additional resources

- **Azure hybrid cloud**
<https://azure.microsoft.com/overview/hybrid-cloud/>
- **Azure Stack**
<https://azure.microsoft.com/overview/azure-stack/>
- **Active Directory Service Accounts for Windows Containers**
<https://docs.microsoft.com/virtualization/windowscontainers/manage-containers/manage-serviceaccounts>
- **Create a container with Active Directory support**
<https://docs.microsoft.com/archive/blogs/containerstuff/create-a-container-with-active-directory-support>
- **Azure Hybrid Benefit licensing**
<https://azure.microsoft.com/pricing/hybrid-benefit/>

Walkthroughs and technical get started overview

To limit the size of this e-book, additional technical documentation and the full walkthroughs were made available in a GitHub repository. The online series of walkthroughs that is described in this chapter covers the step-by-step setup of the multiple environments that are based on Windows Containers, and deployment to Azure.

The following sections explain what each walkthrough is about, its objectives and high-level vision, and provides a diagram of the tasks that are involved. You can get the walkthroughs themselves in the *eShopModernizing* apps GitHub repo wiki at <https://github.com/dotnet-architecture/eShopModernizing/wiki>.

Technical walkthrough list

The following get-started walkthroughs provide consistent and comprehensive technical guidance for sample apps that you can lift and shift by using containers, and then move by using multiple deployment choices in Azure.

Each of the following walkthroughs uses the new sample eShopLegacy and eShopModernizing apps, which are available on GitHub at <https://github.com/dotnet-architecture/eShopModernizing>.

- **Tour of eShop legacy apps (baseline apps to modernize)**
- **Containerize your existing ASP.NET web apps (WebForms & MVC) with Windows Containers**
- **Containerize your existing WCF services (N-Tier apps) with Windows Containers**
- **Deploy your Windows Containers-based app to Azure VMs**
- **Deploy your Windows Containers-based apps to Kubernetes in Azure Container Service**

Walkthrough 1: Tour of eShop legacy apps

Technical walkthrough availability

The full technical walkthrough is available in the eShopModernizing GitHub repo wiki:

[eShopModernizing wiki walkthroughs](#)

Overview

In this walkthrough, you can explore the initial implementation of three sample legacy applications. The first two sample web apps have a monolithic architecture, and were created by using classic ASP.NET. One application is based on ASP.NET 4.x MVC; the second application is based on ASP.NET 4.x Web Forms. The third app is a 3-Tier app composed by a client WinForms app and a server-side [Windows Communication Foundation \(WCF\)](#) service.

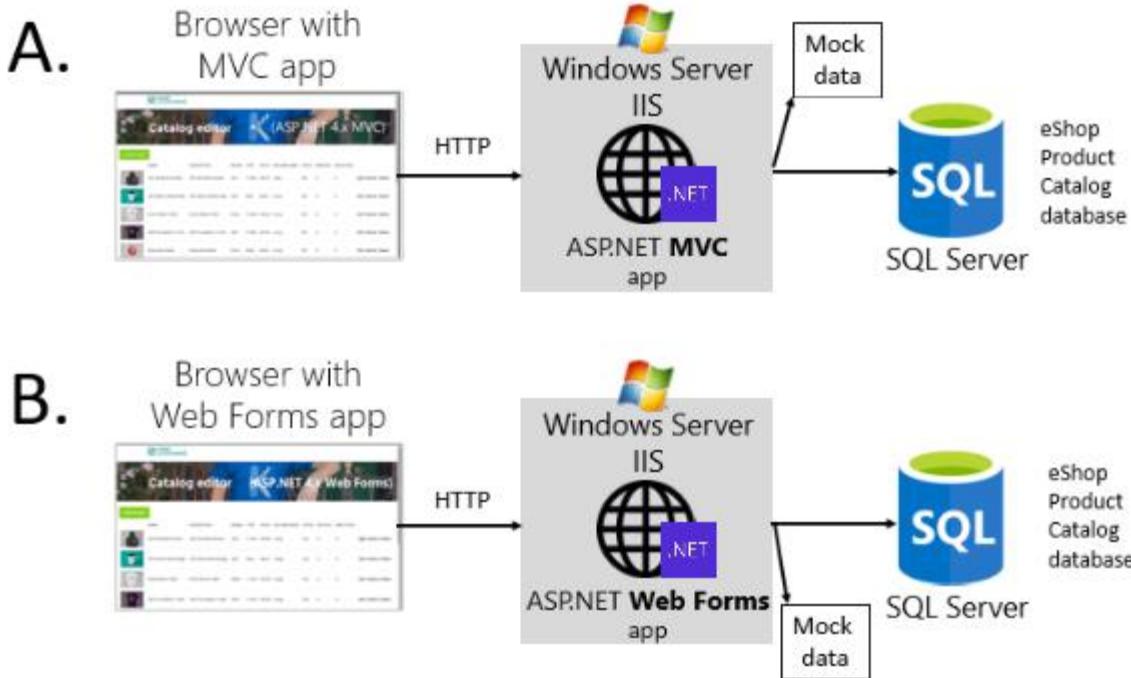
All these applications are available at the [eShopModernizing GitHub repo](#).

Goals

The main goal of this walkthrough is simply to get familiar with these apps, and with their code and configuration. You can configure the apps so that they generate and use mock data, without using the SQL database, for testing purposes. This optional config is based on dependency injection, in a decoupled way.

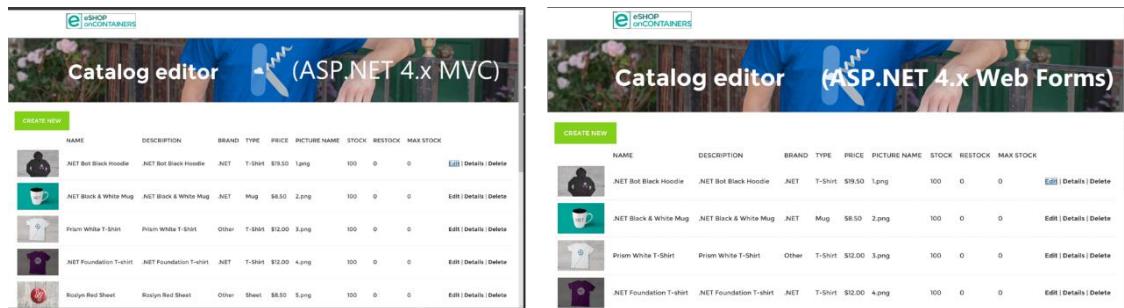
Scenario 1: ASP.NET Web apps

The figure below shows the simple scenario of the original legacy ASP.NET web applications.



From a business domain perspective, both apps offer the same catalog management features. Members of the eShop enterprise team would use the app to view and edit the product catalog.

The next figure shows the initial app screenshots.



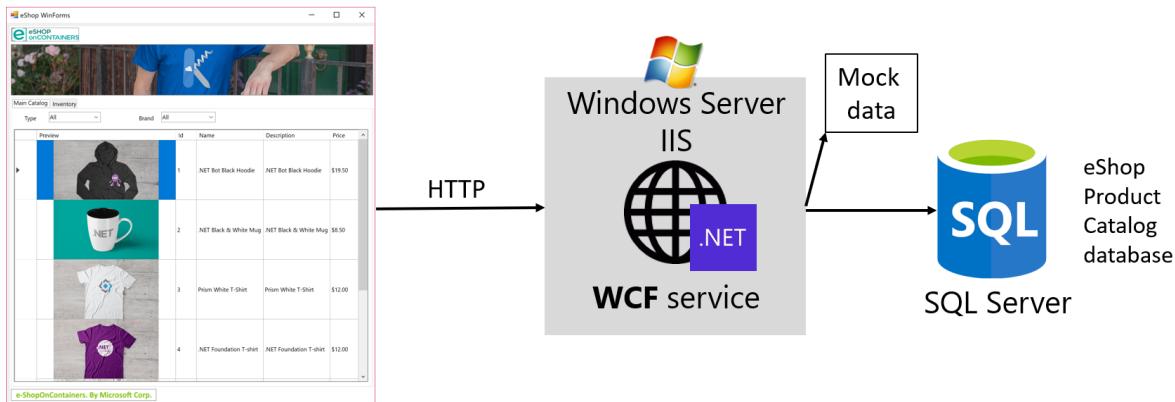
Dependencies in ASP.NET 4.x or earlier versions (either for MVC or for Web Forms) means that these applications won't run on .NET Core unless the code is fully rewritten by using ASP.NET Core MVC.

Scenario 2: WCF service and WinForms client app (3-Tier app)

The figure below shows the simple scenario of the original 3-Tier legacy application.

WinForms

Desktop app



Benefits

The benefits of this walkthrough are simple: Just get familiar with the code and initial apps.

Next steps

Explore this content more in-depth on the GitHub wiki:

- [Tour on the baseline ASP.NET MVC and Web Forms “legacy” apps](#)
- [Tour on the baseline WCF service and WinForms \(3-Tier\) “legacy” app](#)

Walkthrough 2: Containerize your existing .NET applications with Windows Containers

Overview

Use Windows Containers to improve deployment of existing .NET applications, like those based on MVC, Web Forms, or WCF, to production, development, and test environments.

Goals

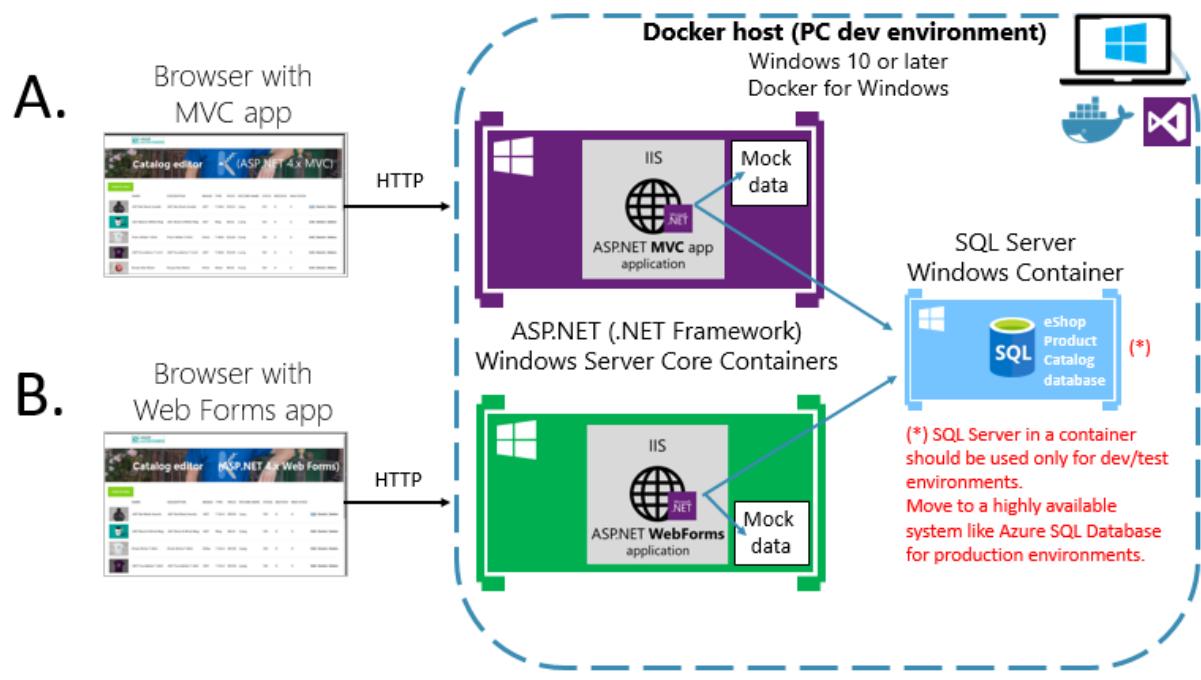
The goal of this walkthrough is to show you several options for containerizing an existing .NET Framework application. You can:

- Containerize your application by using [Visual Studio 2022 Tools for Docker](#) (Visual Studio 2017 or later versions).
- Containerize your application by manually adding a [Dockerfile](#), and then using the [Docker CLI](#).
- Containerize your application by using the [Img2Docker](#) tool (an open-source tool from Docker).

This walkthrough focuses on the Visual Studio 2022 Tools for Docker approach, but the other two approaches are fairly similar in regard to using Dockerfiles.

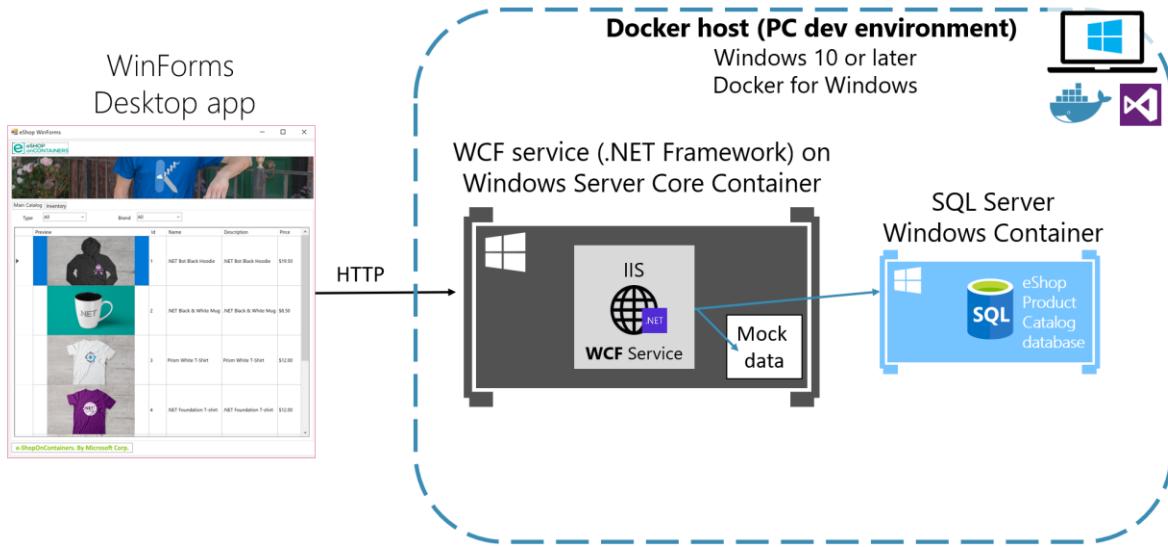
Scenario 1: Containerized ASP.NET web apps

Figure below shows the scenario for containerized eShop legacy web apps applications.



Scenario 2: Containerized WCF service

Figure below shows the scenario for a 3-Tier app with a containerized WCF service.



Benefits

There are advantages to running your monolithic application in a container. First, you create an image for the application. From that point on, every deployment runs in the same environment. Every container uses the same OS version, has the same version of dependencies installed, uses the same .NET framework version, and is built by using the same process. Basically, you control the dependencies of your application by using a Docker image. The dependencies travel with the application when you deploy the containers.

An additional benefit is that developers can run the application in the consistent environment that's provided by Windows Containers. Issues that appear only with certain versions can be spotted immediately, instead of surfacing in a staging or production environment. Differences in development environments used by members of the development team matter less when applications run in containers.

Containerized applications also have a flatter scale-out curve. Containerized apps enable you to have more application and service instances (based on containers) in a VM or physical machine compared to regular application deployments per machine. This approach translates to higher density and fewer required resources, especially when you use orchestrators like Kubernetes.

Containerization, in ideal situations, does not require making any changes to the application code (C#). In most scenarios, you just need the Docker deployment metadata files (Dockerfiles and Docker Compose files).

Next steps

Explore this content more in-depth on the GitHub wiki:

- [How to containerize the .NET Framework web apps with Windows Containers and Docker](#)
- [Adding Docker Support to a WCF service](#)

Walkthrough 3: Deploy your Windows Containers-based app to Azure VMs

Technical walkthrough availability

The full technical walkthrough is available in the eShopModernizing GitHub repo wiki:

[https://github.com/dotnet-architecture/eShopModernizing/wiki/06.-Deploying-your-Windows-Containers-based-app-into-Azure-VMs-\(Including-CI-CD\)](https://github.com/dotnet-architecture/eShopModernizing/wiki/06.-Deploying-your-Windows-Containers-based-app-into-Azure-VMs-(Including-CI-CD))

Overview

Deploying to a Docker host on a Windows Server 2016 Virtual Machine (VM) in Azure lets you quickly set up development/test/staging environments. It also gives you a common place for testers or business users to validate the app. VMs also can be valid Infrastructure as a Service (IaaS) production environments.

Goals

The goal of this walkthrough is to show you the multiple alternatives you have when you deploy Windows Containers to Azure VMs that are based on Windows Server 2016 or later versions.

Scenarios

Several scenarios are covered in this walkthrough.

Scenario A: Deploy to an Azure VM from a dev PC through Docker Engine connection

Scenario: Deploy to Azure VM through Docker Engine connection

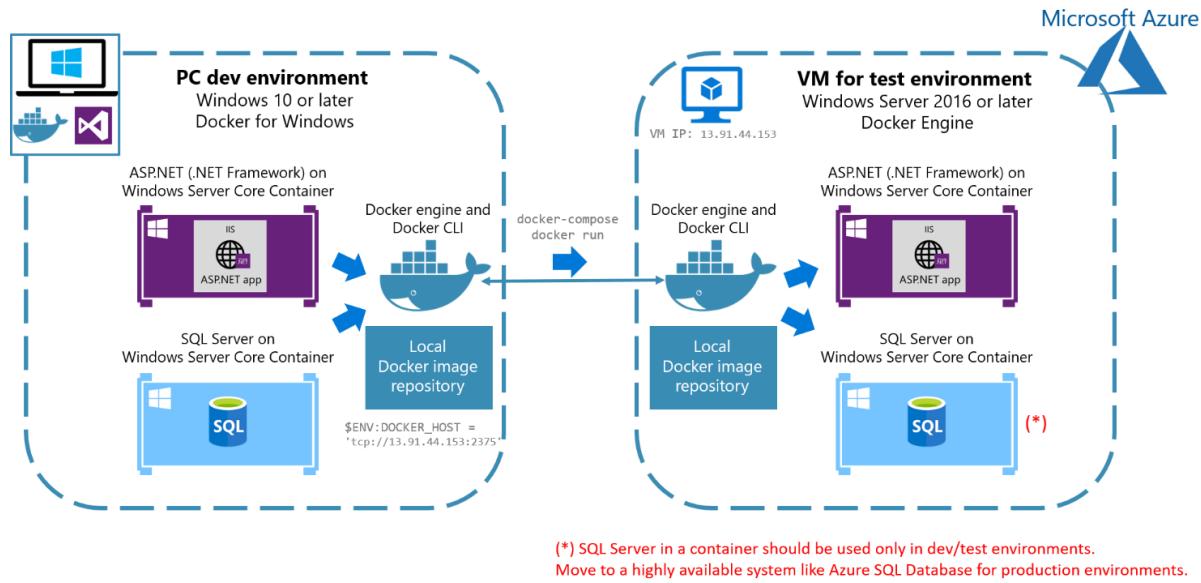


Figure 5-4. Deploy to an Azure VM from a dev PC through a Docker Engine connection

Scenario B: Deploy to an Azure VM through a Docker Registry

Scenario: Deploy to Azure VM through the Docker Registry

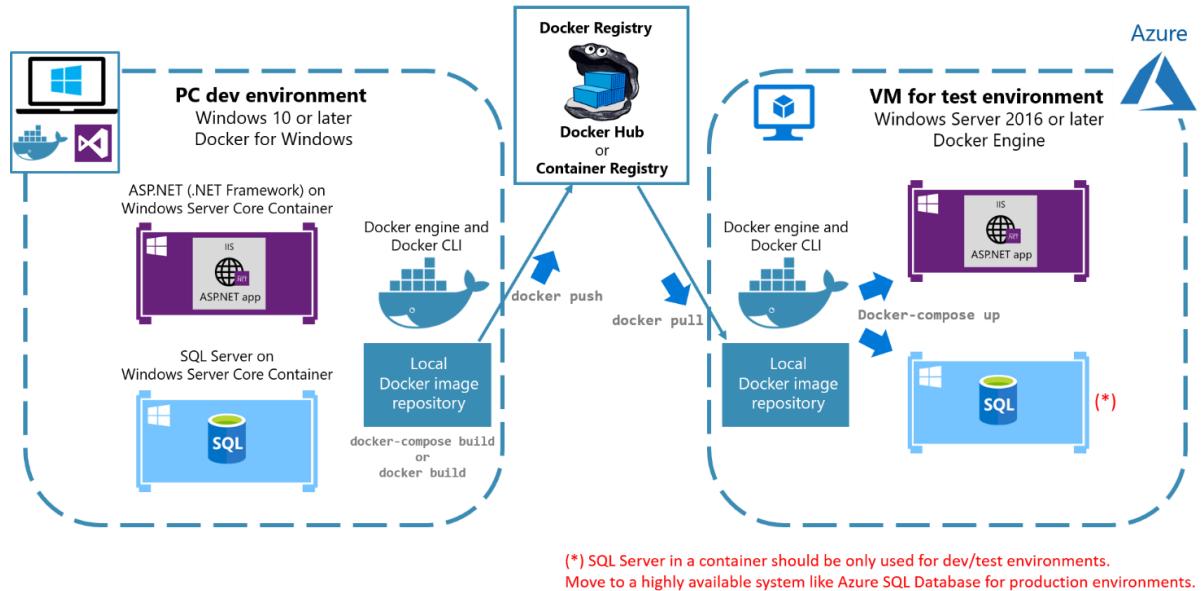


Figure 5-5. Deploy to an Azure VM through a Docker Registry

Scenario C: Deploy to an Azure VM from CI/CD pipelines in Azure DevOps Services

Scenario: Deploy to Azure VM through CI/CD pipelines

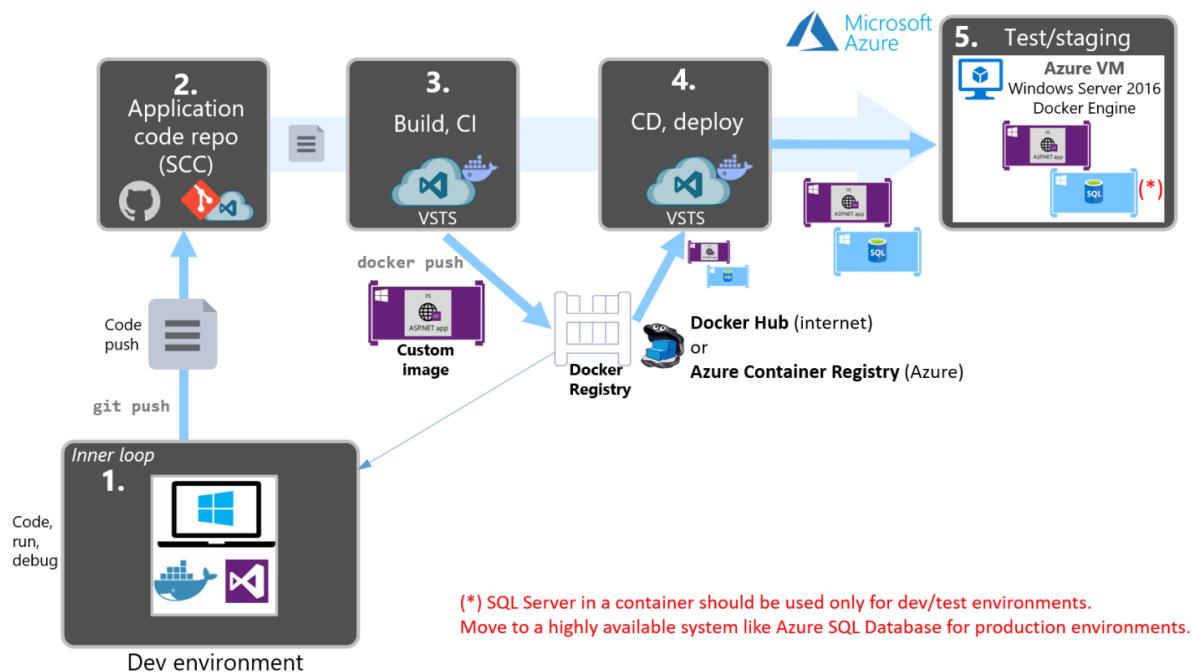


Figure 5-6. Deploy to an Azure VM from CI/CD pipelines in Azure DevOps Services

Azure VMs for Windows Containers

Azure VMs for Windows Containers are VMs based on Windows Server 2016, Windows 10, or later versions, both with Docker Engine set up. In most cases, Windows Server 2016 is used in the Azure VMs.

Azure currently provides a VM named **Windows Server 2016 with Containers**. You can use this VM to try the new Windows Server Container feature, with either Windows Server Core or Windows Nano Server. Container OS images are installed, and then the VM is ready to use with Docker.

Benefits

Although Windows Containers can be deployed to on-premises Windows Server 2016 VMs, when you deploy to Azure, you get an easier way to get started, with ready-to-use Windows Server Container VMs. You also get a common online location that's accessible to testers, and automatic scalability through Azure virtual machine scale sets.

Next steps

Explore this content more in-depth on the GitHub wiki:

[https://github.com/dotnet-architecture/eShopModernizing/wiki/06.-Deploying-your-Windows-Containers-based-app-into-Azure-VMs-\(Including-CI-CD\)](https://github.com/dotnet-architecture/eShopModernizing/wiki/06.-Deploying-your-Windows-Containers-based-app-into-Azure-VMs-(Including-CI-CD))

Walkthrough 4: Deploy your Windows Containers-based apps to Azure Container Instances (ACI)

Technical walkthrough availability

The full technical walkthrough is available in the eShopModernizing GitHub repo wiki:

[Deploying the Apps to ACI \(Azure Container Instances\)](#)

Overview

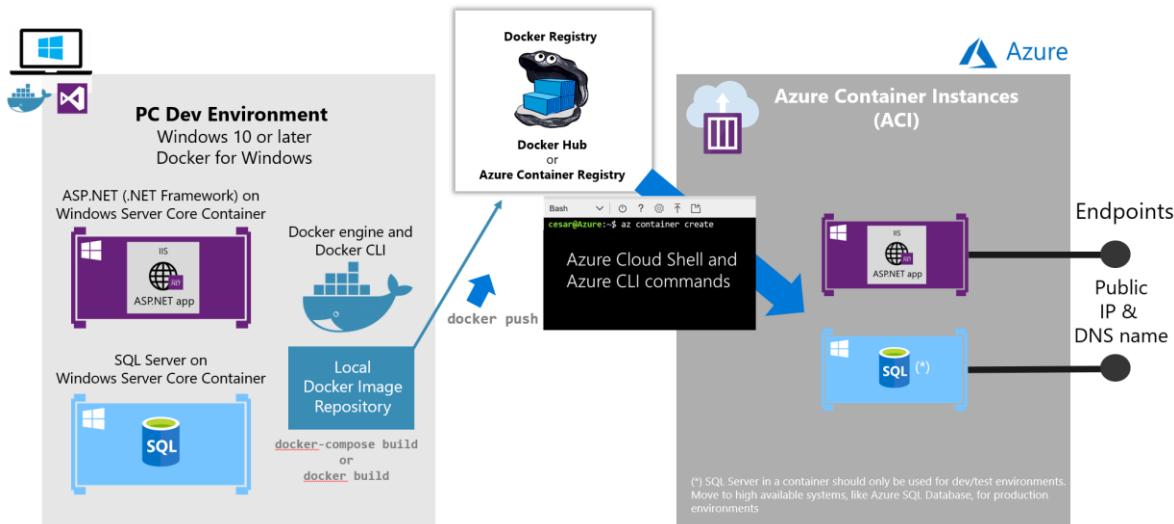
[Azure Container Instances \(ACI\)](#) is the quickest way to have a Containers dev/test/staging environment where you can deploy single instances of containers.

Goals

This walkthrough shows you the main scenarios when deploying Windows Containers to Azure Container Instances (ACI) and how you can deploy eShopModernizing Apps into ACI.

Scenarios

There can be variations about deploying the eShopModernizing apps into ACI such as deploying just one or all of the apps (MVC app, WebForms app or WCF service). In the following scenario shown below, you can see the ASP.NET MVC app plus the SQL Server container both of them deployed as containers into ACI (Azure Container Instances).



Benefits

Azure Container Instances makes it easy to create and manage Docker containers in Azure, without having to provision virtual machines or adopt a higher-level service. With ACI, you can directly deploy a Windows container in Azure and expose it to the internet with a fully qualified domain name (FQDN).

in a matter of seconds (Provided that you have the Windows Container image ready in a Docker registry like Docker Hub or Azure Container Registry).

Considerations

Deploying Windows Containers with either full .NET Framework / ASP.NET or SQL Server into Azure Container Instances (ACI) is not quite as fast as deploying to a regular Docker Host (like a Windows Server 2016 with Windows Containers) because the Docker image has to be downloaded (pulled from the Docker registry) every time and the sizes of the SQL container image (15.1 GB) and the ASP.NET container image (13.9 GB) are significantly large, however it is much cheaper than maintaining your own docker host (permanently on-line Windows Server 2016 with Windows Containers VM in Azure) not to mention a whole orchestrator like Kubernetes in Azure (AKS) which is, on the other hand, a great choice for production deployments.

As the main conclusion, using Azure Container Instances is a very compelling option for Dev/Test scenarios and for CI/CD pipelines.

Next steps

Explore this content more in-depth on the GitHub wiki:

[https://github.com/dotnet-architecture/eShopModernizing/wiki/05.-Deploying-the-Apps-to-ACI-\(Azure-Container-Instances\)](https://github.com/dotnet-architecture/eShopModernizing/wiki/05.-Deploying-the-Apps-to-ACI-(Azure-Container-Instances))

Walkthrough 5: Deploy your Windows Containers-based apps to Kubernetes in Azure Container Service

Technical walkthrough availability

The full technical walkthrough is available in the eShopModernizing GitHub repo wiki:

[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))

Overview

An application that's based on Windows Containers will quickly need to use platforms, moving even further away from IaaS VMs. This approach is needed to easily achieve high scalability and better automated scalability, and for a significant improvement in automated deployments and versioning. You can achieve these goals by using the orchestrator [Kubernetes](#), available in [Azure Container Services](#).

Goals

The goal of this walkthrough is to learn how to deploy a Windows Container-based application to Kubernetes (also called *K8s*) in Azure Container Service. Deploying to Kubernetes from scratch is a two-step process:

1. Deploy a Kubernetes cluster to Azure Container Service.
2. Deploy the application and related resources to the Kubernetes cluster.

Scenarios

Scenario A: Deploy directly to a Kubernetes cluster from a dev environment

Scenario: Direct deployment to a Kubernetes cluster in Azure Container Service

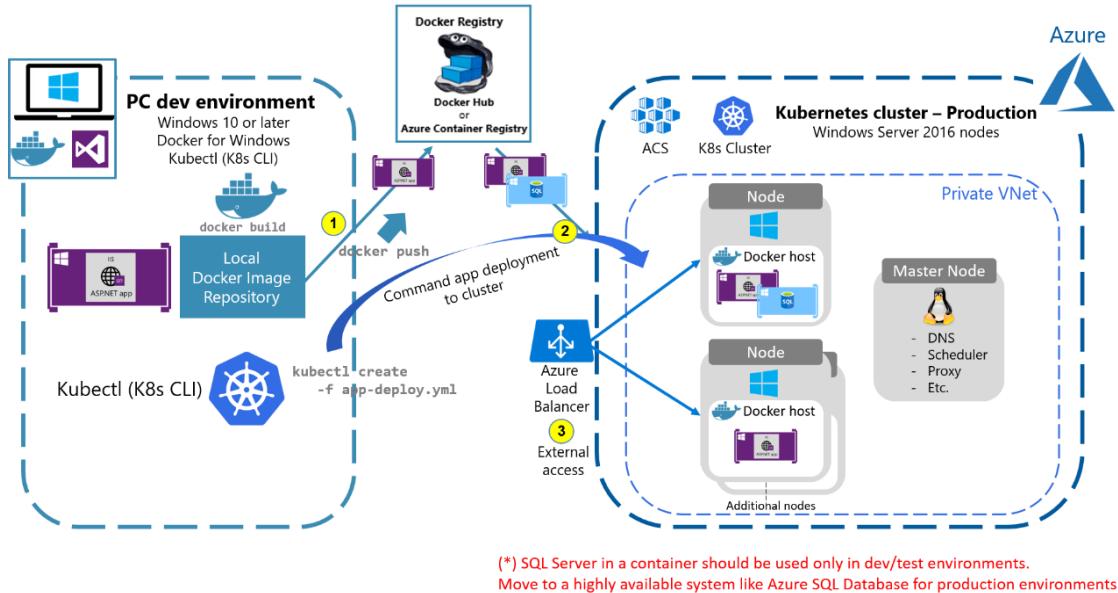


Figure 5-7. Deploy directly to a Kubernetes cluster from a development environment

Scenario B: Deploy to a Kubernetes cluster from CI/CD pipelines in Azure DevOps Services

Scenario: Deploy to Kubernetes through CI/CD pipelines

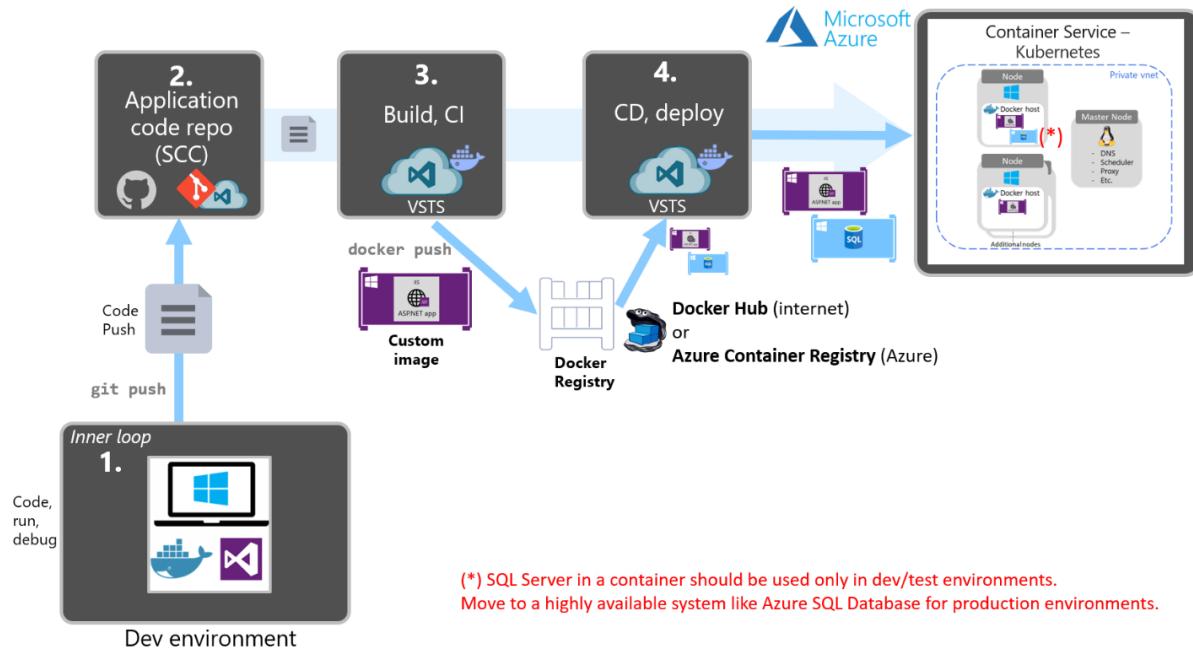


Figure 5-8. Deploy to a Kubernetes cluster from CI/CD pipelines in Azure DevOps Services

Benefits

There are many benefits to deploying to a cluster in Kubernetes. The biggest benefit is that you get a production-ready environment in which you can scale out the application based on the number of container instances you want to use (inner-scalability in the existing nodes), and based on the number of nodes or VMs in the cluster (global scalability of the cluster).

Azure Container Service optimizes popular open-source tools and technologies specifically for Azure. You get an open solution that offers portability, both for your containers and for your application configuration. You select the size, the number of hosts, and the orchestrator tools-Container Service handles everything else.

With Kubernetes, developers can progress from thinking about physical and virtual machines, to planning a container-centric infrastructure that facilitates the following capabilities, among others:

- Applications based on multiple containers
- Replicating container instances and horizontal autoscaling
- Naming and discovering (for example, internal DNS)
- Balancing loads

- Rolling updates
- Distributing secrets
- Application health checks

Next steps

Explore this content more in-depth on the GitHub wiki: [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))

Walkthrough 6: Deploy your Windows Containers-based apps to Azure App Service for Containers

Technical walkthrough availability

The full technical walkthrough is available in the eShopModernizing GitHub repo wiki:

<https://github.com/dotnet-architecture/eShopModernizing/wiki/Deploy-Windows-Container-to-Azure-App-Service>

Overview

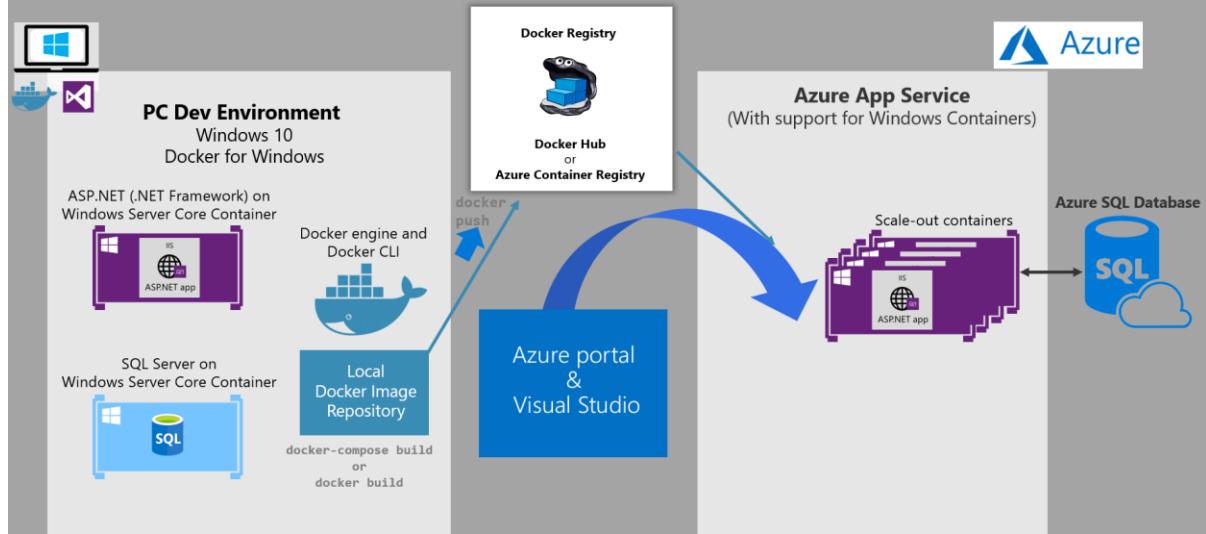
A simple containerized application using Windows Containers can easily be deployed to Azure App Service for Containers. This approach is the recommended approach for most Windows Container-based applications.

Goals

The goal of this walkthrough is to learn how to deploy a Windows Container-based application to Azure App Service for Containers from a registry (Docker Hub or Azure Container Registry).

Scenario

Scenario: Deploy to **Azure App Service** (Windows Containers)



Benefits

Deploying to Azure App Service for Containers offers the benefits of containers paired with the PaaS benefits of Azure App Service. The app service can easily be scaled both vertically and horizontally, and can be configured to autoscale to meet changing demands. Updates can be performed with zero downtime and the configuration of continuous deployment from a registry is easily configured as well.

Next steps

Explore this content more in-depth on the GitHub wiki: <https://github.com/dotnet-architecture/eShopModernizing/wiki/Deploy-Windows-Container-to-Azure-App-Service>

Conclusions

- Container-based solutions ultimately provide cost savings benefits. Containers are a solution to deployment problems because they remove the friction caused by an absence of dependencies in production environments. By removing those issues, it improves Dev/Test, DevOps, and production operations significantly.
- A Docker container is becoming the standard unit of deployment for any server-based application or service.
- For production environments, you should use an orchestrator (like Kubernetes) to host scalable containers-based applications.
- Azure VMs hosting containers are a fast and simple way to create small Dev/Test environments in the cloud.
- Azure SQL Database Managed Instance is recommended by default when migrating your relational databases from existing applications to Azure.
- Visual Studio 2022 and Image2Docker are basic tools for you to start modernizing your existing .NET applications with Windows Containers by accelerating the getting started learning curve.
- When placing containerized applications in production you will always create or adopt a DevOps culture and DevOps tools for CI/CD pipelines, like Azure DevOps Services or Jenkins.
- Microsoft Azure provides the most comprehensive and complete environment to modernize your existing .NET Framework applications with Windows Containers, cloud infrastructure and PaaS services.