Microsoft

Modernizing Applications with Containers and Orchestrators

Microsoft Services

Microsoft

Module 2 – Getting Started with Windows Containers

Microsoft Services

Objectives

- Introduce Windows Containers
- Understand the Microsoft and Docker Partnership
- Explore Windows Container Images
- Work with Dockerfiles
- Build and run a variety of Windows containers
- Learn about Visual Studio Support for Docker
- Understand how to perform container updates

Application innovation with Windows containers

Windows Server 2016

Initial launch of containers

Process and Hyper-V isolation

Docker EE Basic Included at no additional cost

Windows Server, version 1709

Optimized container images for Nano Server and Server Core

Platform level support for Linux containers

Windows Subsystem for Linux

Networking enhancements for overlays and SDN

Windows Server, version 1803

Optimized Server Core image

App compat improvements

Native command line tools—curl.exe, tar.exe and SSH

Enhancements to the Windows Subsystem for Linux

Networking enhancements for greater density and quicker endpoint creation

Improved network security with Calico Open source storage plugins for Kubernetes

Platform functionality required for Kubernetes conformance

Windows Server 2019

Optimized Server Core image

App compat improvements

Enhanced Group Managed Service Account support

Platform functionality for Kubernetes and Microsoft Service Fabric

Performance and density improvements

Platform and open source work on CNI networking plugins such as Calico and Flannel

Enhancements to the Windows Subsystem for Linux

...you will have to wait

Docker and Microsoft delivers integrated tooling across the application lifecycle

Build

Ship

Run

Visual Studio

Visual Studio Tools for Docker

Docker for Windows

Library of Microsoft images on Docker Hub

Docker Datacenter for orchestration, management and security

Microsoft Operations Management Suite for hybrid cloud visibility and control

Windows Server

Docker containers available for Windows Server running on any infrastructure

Microsoft Hyper-V

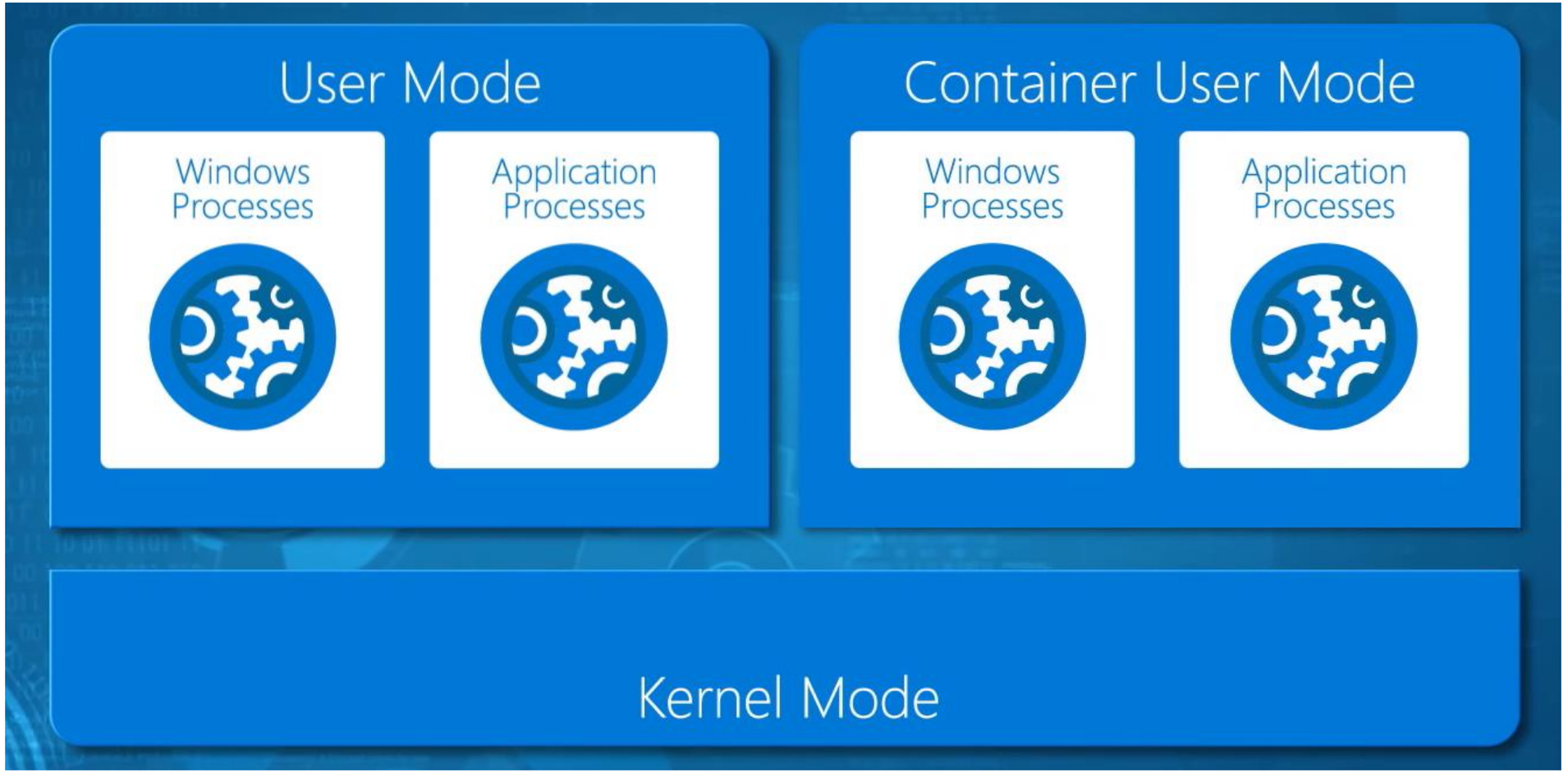
Azure

www.docker.com/microsoft

Docker and Microsoft Partnership

- Docker Engine is tested, validated, and supported on Windows Server 2016 and 2019/Windows 10 customers at no additional cost
- Microsoft will provide Windows Server 2016 and 2019 customers enterprise support for CS Docker Engine, backed by Docker, Inc.
- Integration between Visual Studio Tools for Docker and Docker for Windows
- Windows Server container base images discoverable on Docker Hub

Windows Containers Shared Kernel Model



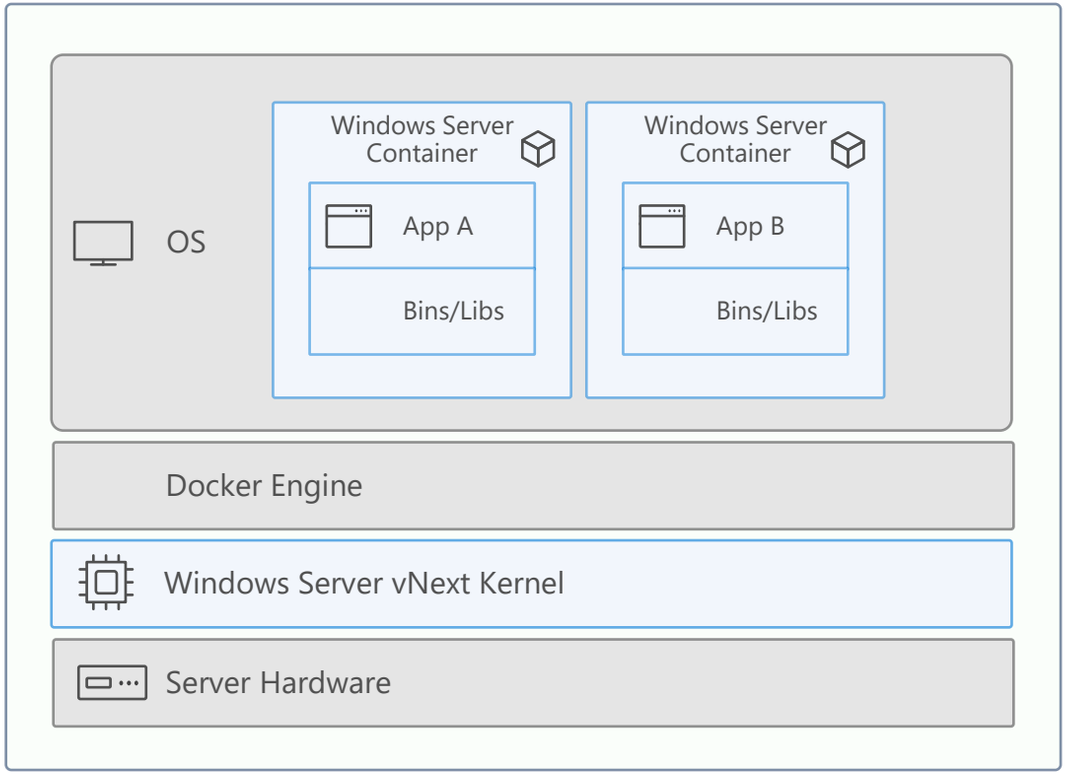
Windows Containers

Namespace and Resource Isolation

Container sees it's own file system and registry and can be told how much process, memory, and CPU it can use.

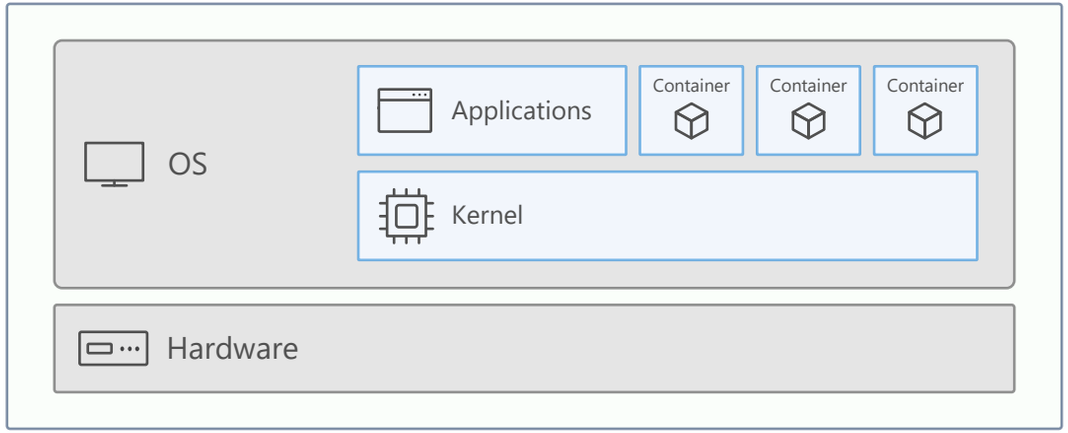
Network virtualization

Each application/container could have it's own IP address to provide a layer of isolation so that the container doesn't have access outside of its sandboxed execution environment



Types of Windows Containers

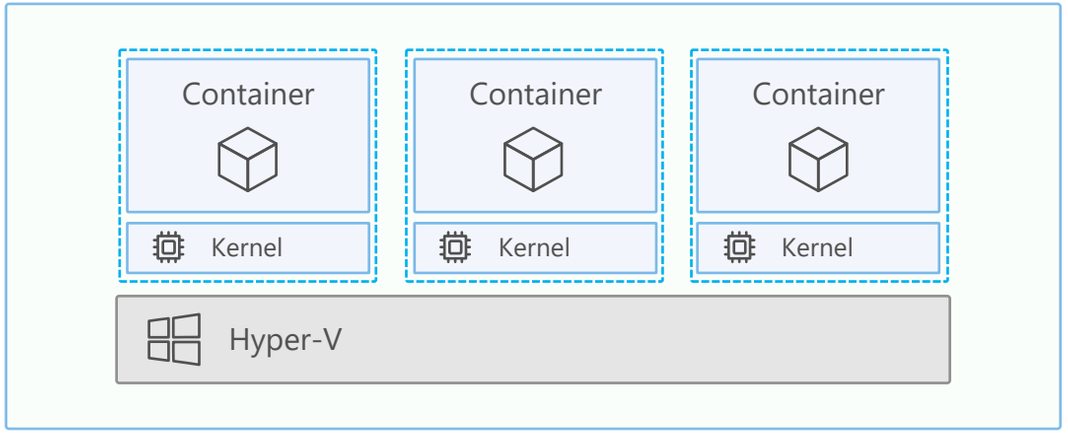
Windows Server containers: maximum speed and density



Namespace, resource control and process isolation

Shared host kernel

Hyper-V containers: isolation plus performance



Run inside a light-weight, utility VM

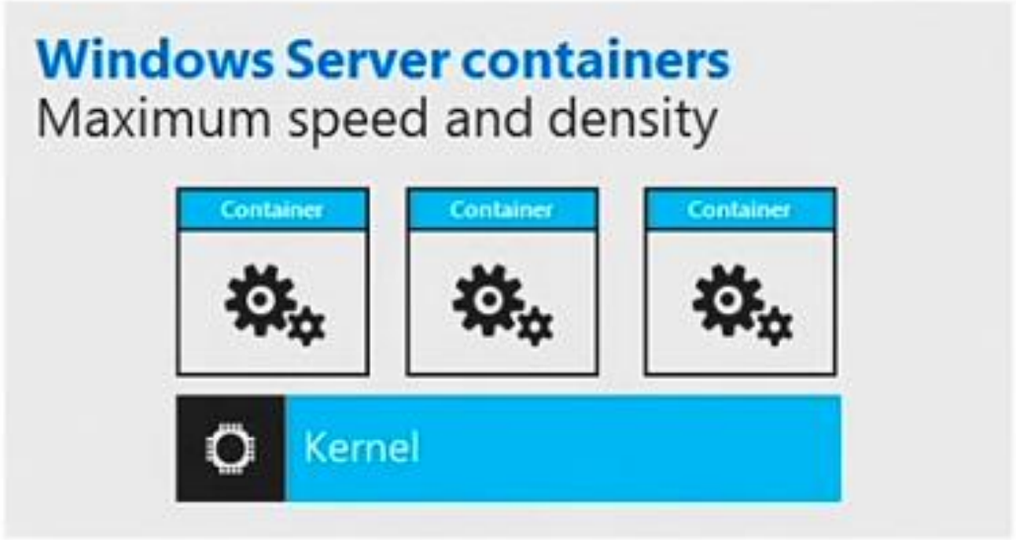
Kernel-level isolation

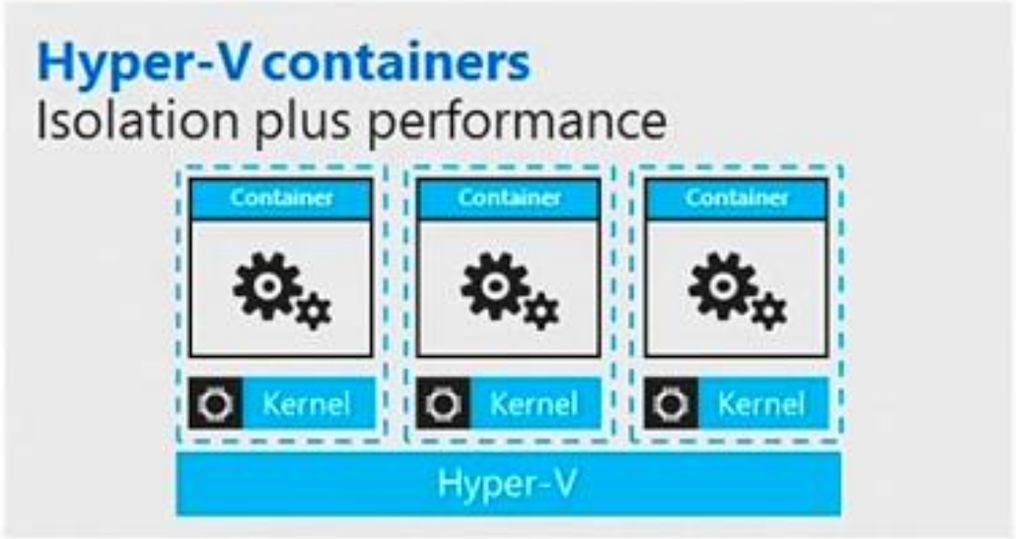
Isolation level can be specified at runtime --isolation=process or hyperv

How Hyper-V Containers are Different?

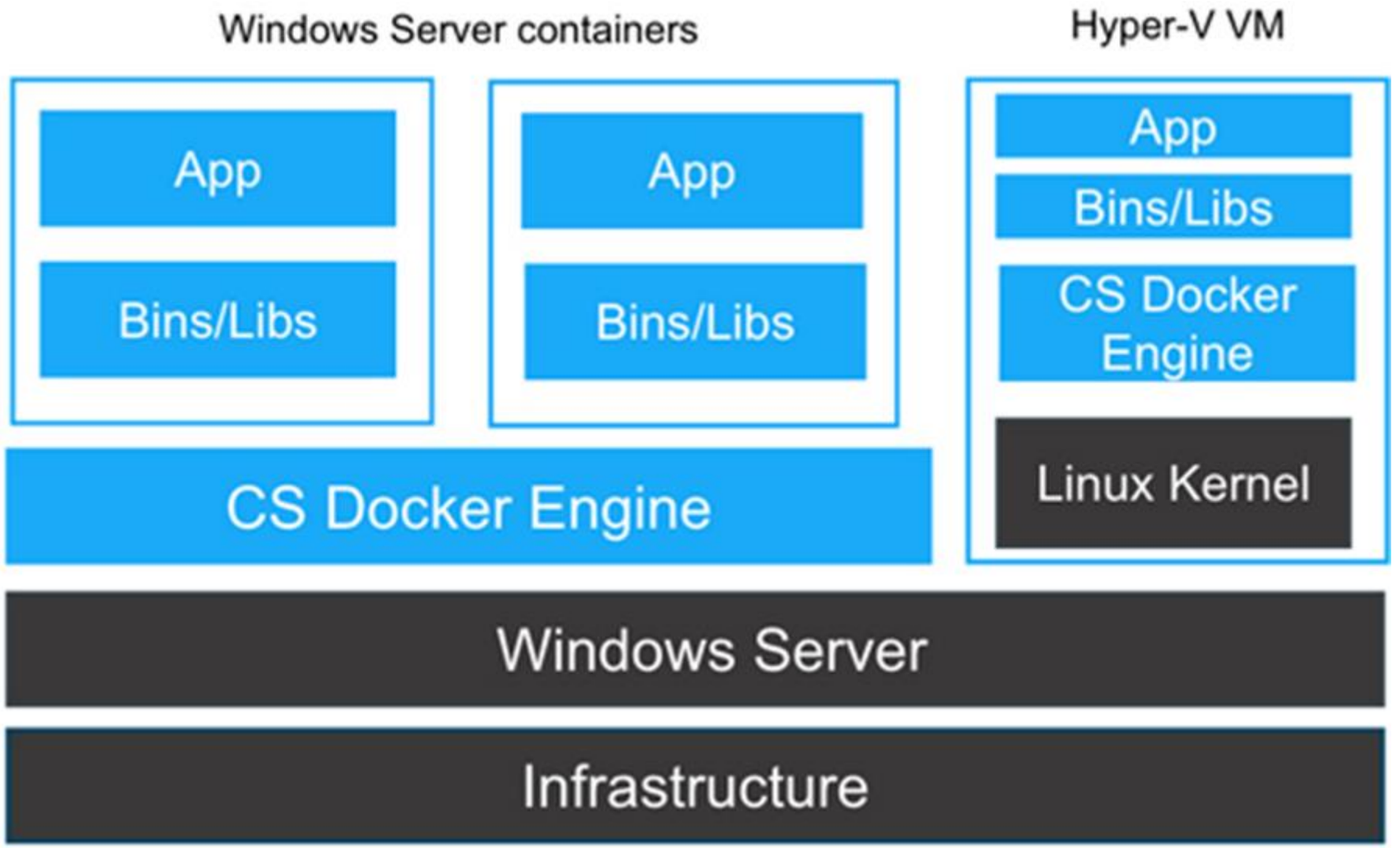
- Windows Server Container applications that are pushed or pulled from the Docker Hub or local repository can be placed in either a regular Windows Server Container or a Hyper-V Container without any modification.

- Hyper-V Containers offer both OS virtualization (container) and machine virtualization (VM) in a slightly lighter-weight configuration than a traditional VM.





Hyper V Containers with Linux



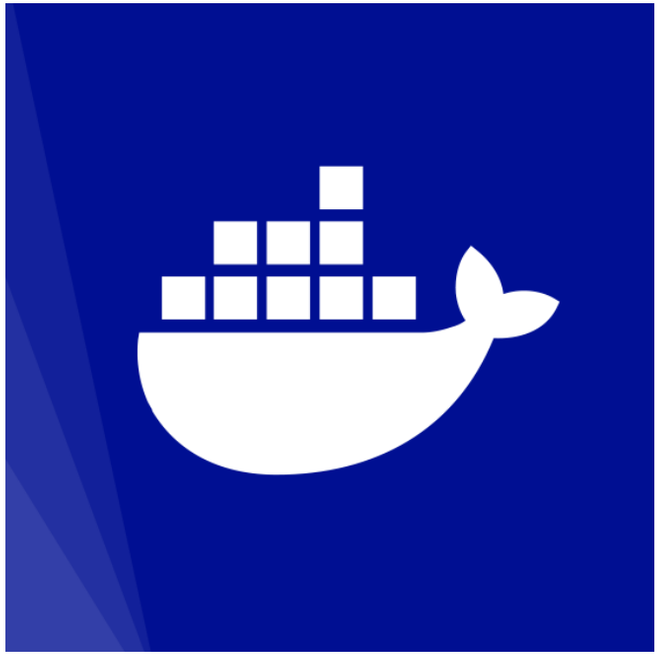
Docker Desktop for Windows

- By installing “Docker Desktop for Windows”, Docker developers can use a single Docker CLI to build apps for both Windows or Linux
- Includes everything you need to build, test and ship containerized applications right from your machine
- Integrated tools including the Docker command line, Docker Compose and kubectl command line
- Docker Desktop allows you to develop applications locally with either Docker Swarm or Kubernetes

kubernetes

Windows 10 vs. Windows Server

Docker Desktop for Windows



Docker on Windows Server

Windows Container Base OS Images

- Base OS image is the first layer in potentially many image layers that make up a container
- Container OS Base Image is immutable (read-only)
- 4 options as of Windows Server 2019:
  - Nano server
  - Server core
  - Windows
  - IOT core
- Hosted in Microsoft Container Registry and discoverable via existing channels (i.e. Docker Hub)

Windows Container Base OS Images

Windows (https://hub.docker.com/_/microsoft-windows) *New in Windows Server 2019

- Automation workloads
- Carries most Windows OSS components

3.5 GB

Windows Server Core (https://hub.docker.com/_/microsoft-windows-servercore)

- Minimal installation of Windows Server 2016
- Contains only core OS features
- Command-line access only

1.4 GB

Nano Server (https://hub.docker.com/_/microsoft-windows-nanoserver)

- Available only as container base OS image (no VM support)
- 20 times smaller than Server Core
- Headless – no logon or GUI
- Optimized for .NET Core applications

94 MB

*Sizes based on Windows Server 2019

Demonstration: Nano Server and Windows Server Core

Working with Windows Server Core Container

Working with Nano Server Container

Container Image

An immutable, file-based template for a container that is created one of three ways:

- Manual via a Docker commit
- Automated with a Dockerfile
- Pulled from a registry

```
.
├── 47bcc53f74dc94b1920f0b34f6036096526296767650f223433fe65c35f149eb.json
├── 5f29f704785248ddb9d06b90a11b5ea36c534865e9035e4022bb2e71d4ecbb9a
│   ├── VERSION
│   ├── json
│   └── layer.tar
├── a65da33792c5187473faa80fa3e1b975acba06712852d1dea860692ccddf3198
│   ├── VERSION
│   ├── json
│   └── layer.tar
├── manifest.json
└── repositories
```

Image Layering

Base OS Image

- System Binaries
- C:\Windows

Choices

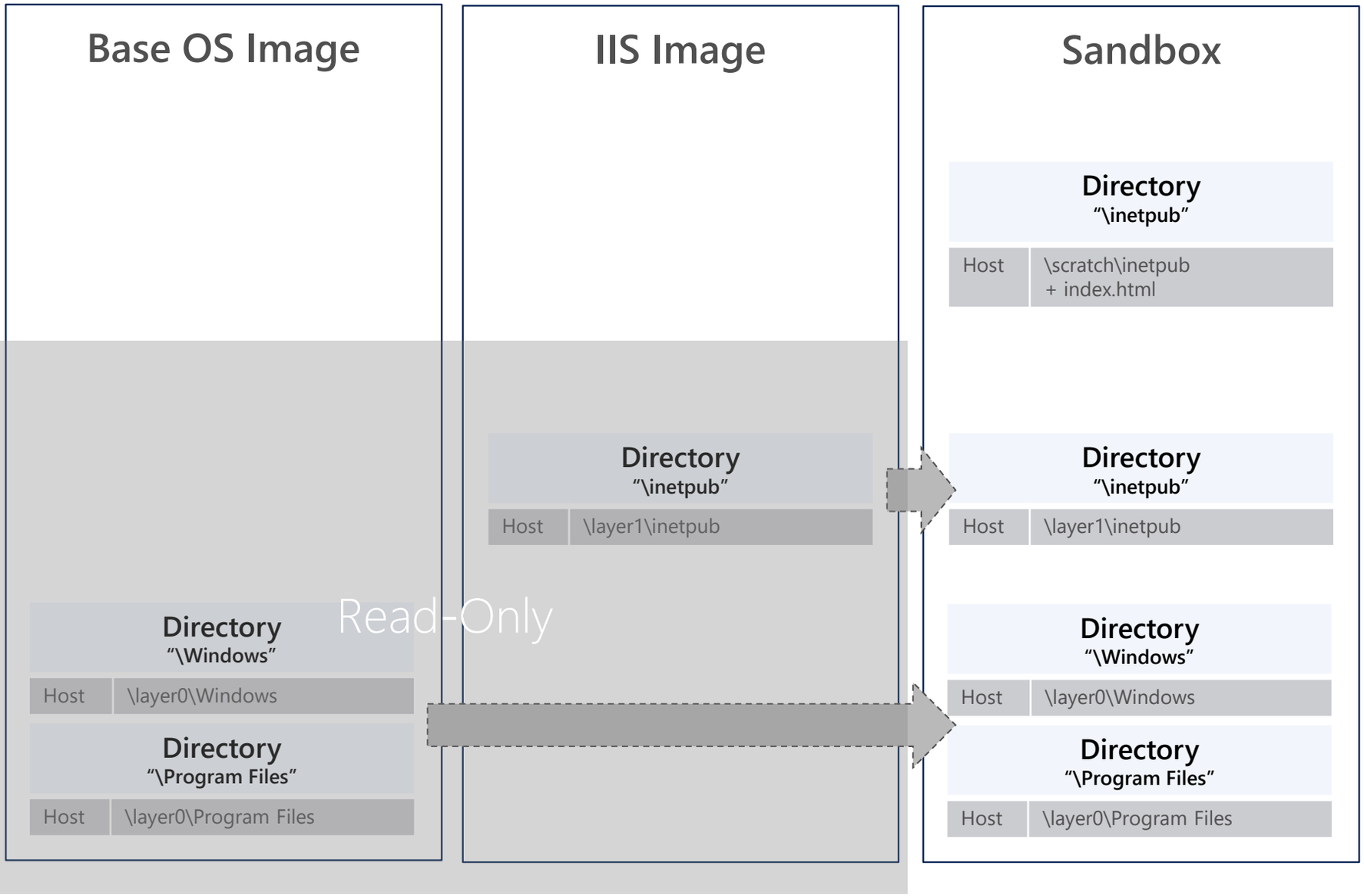
- Windows
- Windows Server Core
- Nano Server

Availability

- Obtainable through Container Image provider
- Published by Microsoft to the MCR

Image Layering

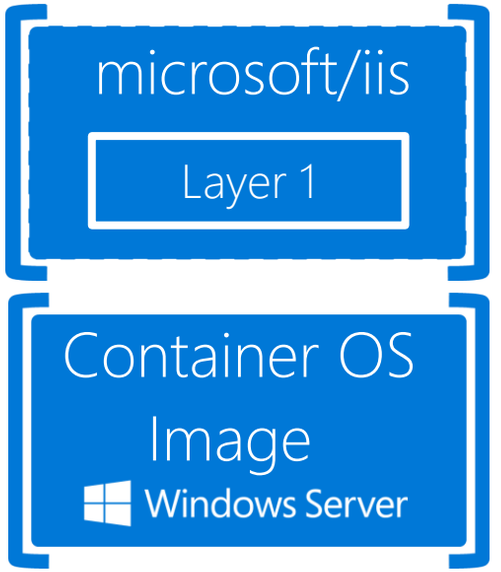
Running Container



Dockerfile – Build IIS Server Container Image

- Method for automated container image build
- Consumed when running “docker build”
- Enables automated builds via Docker Hub
- Caches unchanged commands

```
FROM windowsservercore
RUN powershell –command
Add-WindowsFeature Web-Server
```



Demonstration: *Building and Running IIS Server Container*

Build IIS Container Image using Dockerfile

Run IIS Container

Dockerfile – Build ASP.NET 4.7 Container Image

Leverage Windows Server Core 2019 Container Image

```
FROM mcr.microsoft.com/windows/servercore:ltsc2019
```

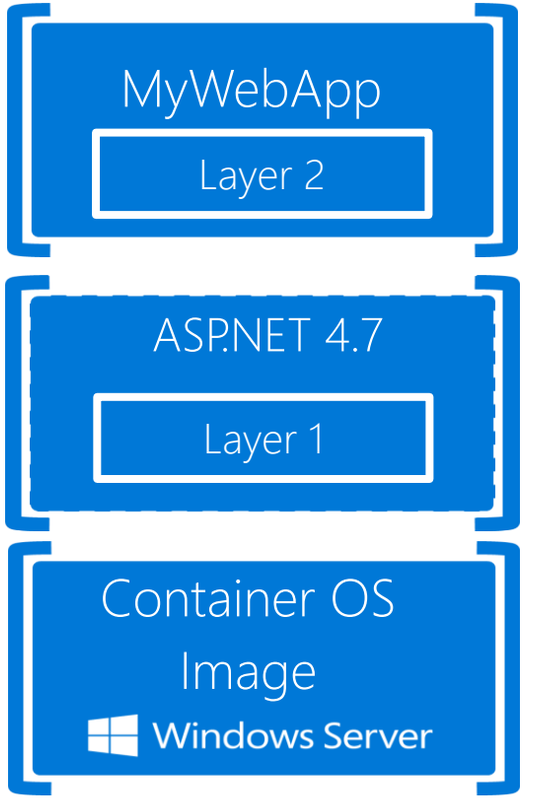
Install .NET and ASP.NET 4.7

```
ENV COMPLUS_NGenProtectedProcess_FeatureEnabled 0

RUN \Windows\Microsoft.NET\Framework64\v4.0.30319\ngen uninstall "Microsoft.Tpm.Commands,
    && \Windows\Microsoft.NET\Framework64\v4.0.30319\ngen update `
    && \Windows\Microsoft.NET\Framework\v4.0.30319\ngen update
```

Copy MyWebApp to Container

```
COPY MyWebApp /inetpub/wwwroot
```



Demonstration: *Package ASP.NET 4.7 Web Application as Container*

Containerized ASP.NET 4.7 Web Application

Run ASP.NET 4.7 Web Application as Container

Dockerfile – Build ASP.NET Core Container Image

Leverage IIS Container Image

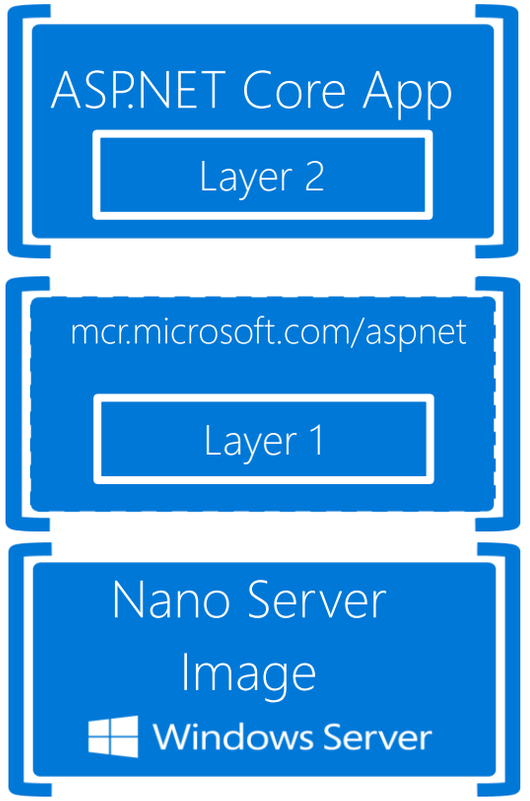
```
FROM mcr.microsoft.com/dotnet/core/aspnet:2.2.3-nanoserver-1809
```

Copy ASP.NET Core App to Container

```
COPY published ./
```

Entrypoint set to Application

```
ENTRYPOINT ["dotnet", "mywebapp.dll"]
```



Demonstration: *Package ASP.NET Core Web Application as Container*

Containerized ASP.NET Core Web Application

Run ASP.NET Core Application as Container

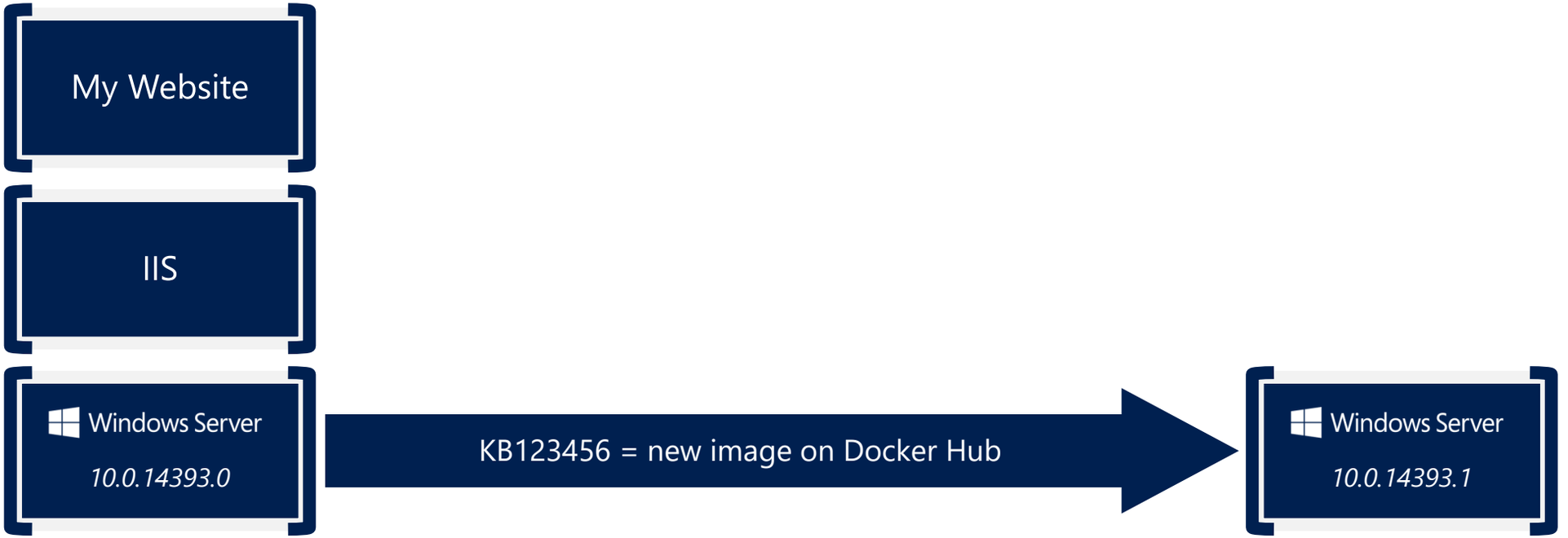
Initiating an update of the Base OS Image

- Rebuild containers using Dockerfile
- Pull updated base image

Update Container OS Image

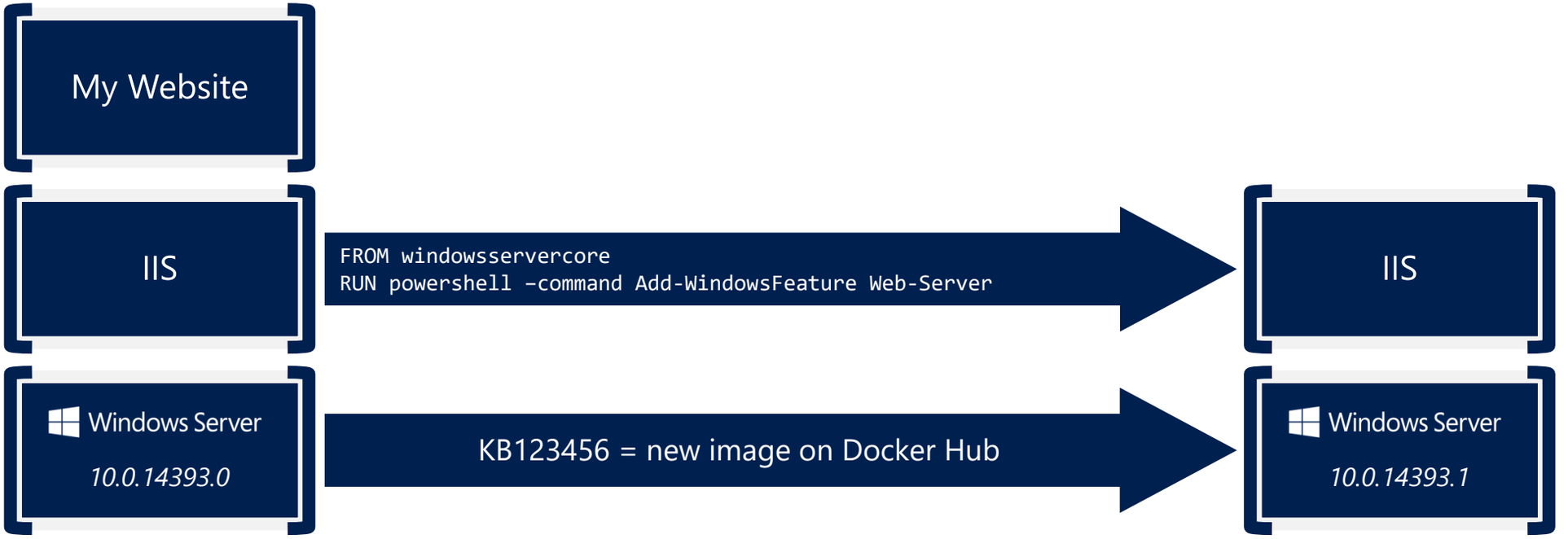
Pull updated base image

https://hub.docker.com/_/microsoft-windows-nanoserver



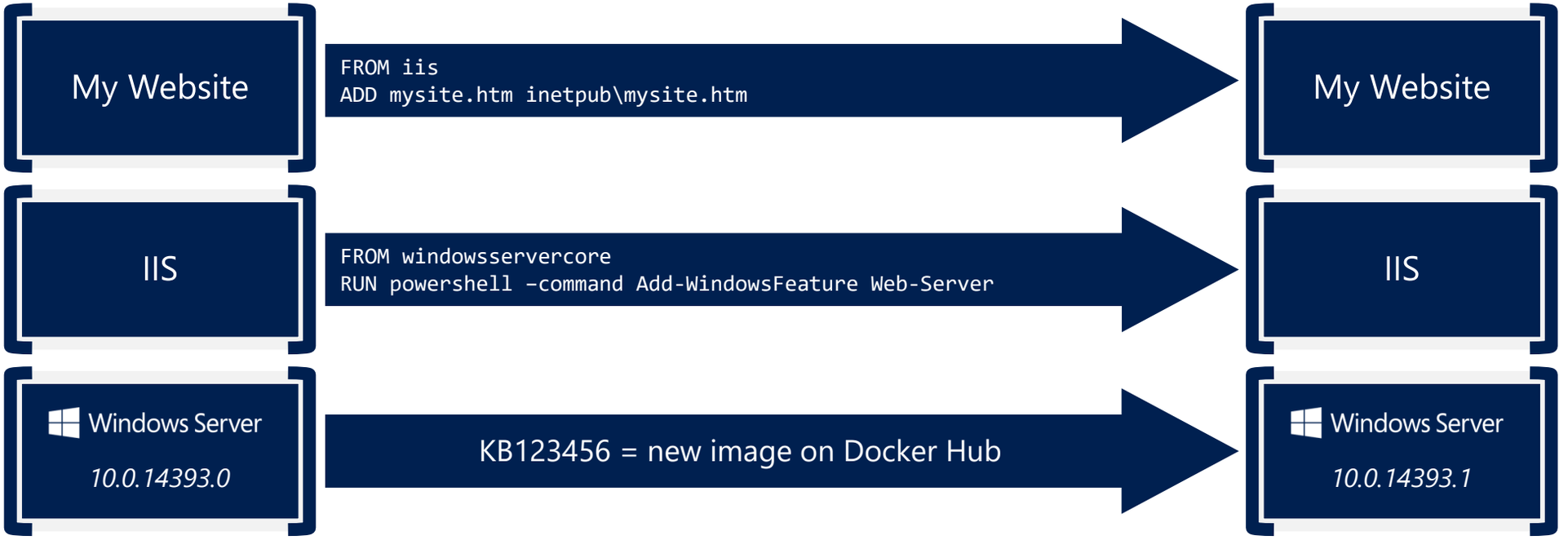
Update Base OS Image

Create new image using Dockerfile



Update Base OS Image

Create new image using Dockerfile



Update as New Layer

Download update in container

When container is stopped update is applied as a new layer

Not a recommended practice

Update as New Layer

Download update in container

When container is stopped update is applied as a new layer

Update as New Layer

Download update in container

When container is stopped update is applied as a new layer

Update as New Layer

Download update in container

When container is stopped update is applied as a new layer

Pre-Multistage Dockerfile

1. Dockerfile.{purpose} to use for environments
2. Dockerfile: must compile application first

```
PS C:\temp\app> dotnet publish -o published -c release app.csproj
Microsoft (R) Build Engine version 16.3.0+0f4c62fea for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

  Restore completed in 15.22 ms for C:\temp\app\app.csproj.
  app -> C:\temp\app\bin\release\netcoreapp3.0\app.dll
  app -> C:\temp\app\bin\release\netcoreapp3.0\app.Views.dll
  app -> C:\temp\app\published\
PS C:\temp\app>
PS C:\temp\app>
PS C:\temp\app> docker build -y myapp .
```

Dockerfile > ...

```
1  FROM mcr.microsoft.com/dotnet/core/aspnet:3.0-buster-slim
2  WORKDIR /app
3  COPY published .
4  ENTRYPOINT [ "dotnet","myapp.dll" ]
5
```

Multistage Dockerfile (Docker 17.5)

- Use multiple FROM statements in your Dockerfile
- Each FROM begins a new stage of the build and can use a different base image
- Selectively copy artifacts from one stage to another

```
FROM mcr.microsoft.com/dotnet/core/aspnet:3.0-buster-slim AS base
WORKDIR /app
EXPOSE 80

FROM mcr.microsoft.com/dotnet/core/sdk:3.0-buster AS build
WORKDIR /src
COPY ["myapp/myapp.csproj", "myapp/"]
RUN dotnet restore "myapp/myapp.csproj"
COPY . .
WORKDIR "/src/myapp"
RUN dotnet build "myapp.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "myapp.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "myapp.dll"]
```

Target a specific build stage

- Debug a specific build stage
- Use a debug stage with all debugging symbols or tools enabled, and a lean production stage
- Use a testing stage in which your app gets populated with test data, but building for production using a different stage which uses real data

```
FROM mcr.microsoft.com/dotnet/core/aspnet:3.0-buster-slim AS base
WORKDIR /app
EXPOSE 80

FROM mcr.microsoft.com/dotnet/core/sdk:3.0-buster AS build
WORKDIR /src
COPY ["myapp/myapp.csproj", "myapp/"]
RUN dotnet restore "myapp/myapp.csproj"
COPY . .
WORKDIR "/src/myapp"
RUN dotnet build "myapp.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "myapp.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "myapp.dll"]
```

```
docker build --target build -t myrepo/myapp:latest
```

Visual Studio Tools for Docker

- Microsoft Visual Studio 2017 and 2019 provide integrated developer experiences for Docker
- Leverage VS Code using the Docker extension

Demonstration: *Visual Studio and Docker*

Building ASP.NET Core Application using Visual Studio 2017/2019

Debugging ASP.NET Core Application using Visual Studio

Group Managed Service Accounts for Windows containers

- Although Windows containers cannot be domain joined, they can still use Active Directory domain identities to support various authentication scenarios

- You can configure a Windows container to run with gMSA, which is a service account designed to allow multiple computers to share an identity without needing to know its password

- Support for scheduling Windows containers with gMSAs in Kubernetes is currently in Public preview. See Configure gMSA for Windows pods and containers for the latest information about this feature and how to test it in your Kubernetes distribution

Lab: Getting Started with Containers

http://aka.ms/PremierEducation

Sign in with a Microsoft Account (Live/outlook or personal ID).

Navigate to **: WorkshopPLUS -> My Training -> Redeem Training Key**.

Training Key: **14AF9E739B574ABF**

Launch Windows VM (UPD20211105)

```
docker run -p 8888:8888 jupyter/minimal-notebook
```

```
docker run -d -p 8080:8080 springio/gs-spring-boot-docker
```

© 2015 Microsoft Corporation. All rights reserved.