Microsoft

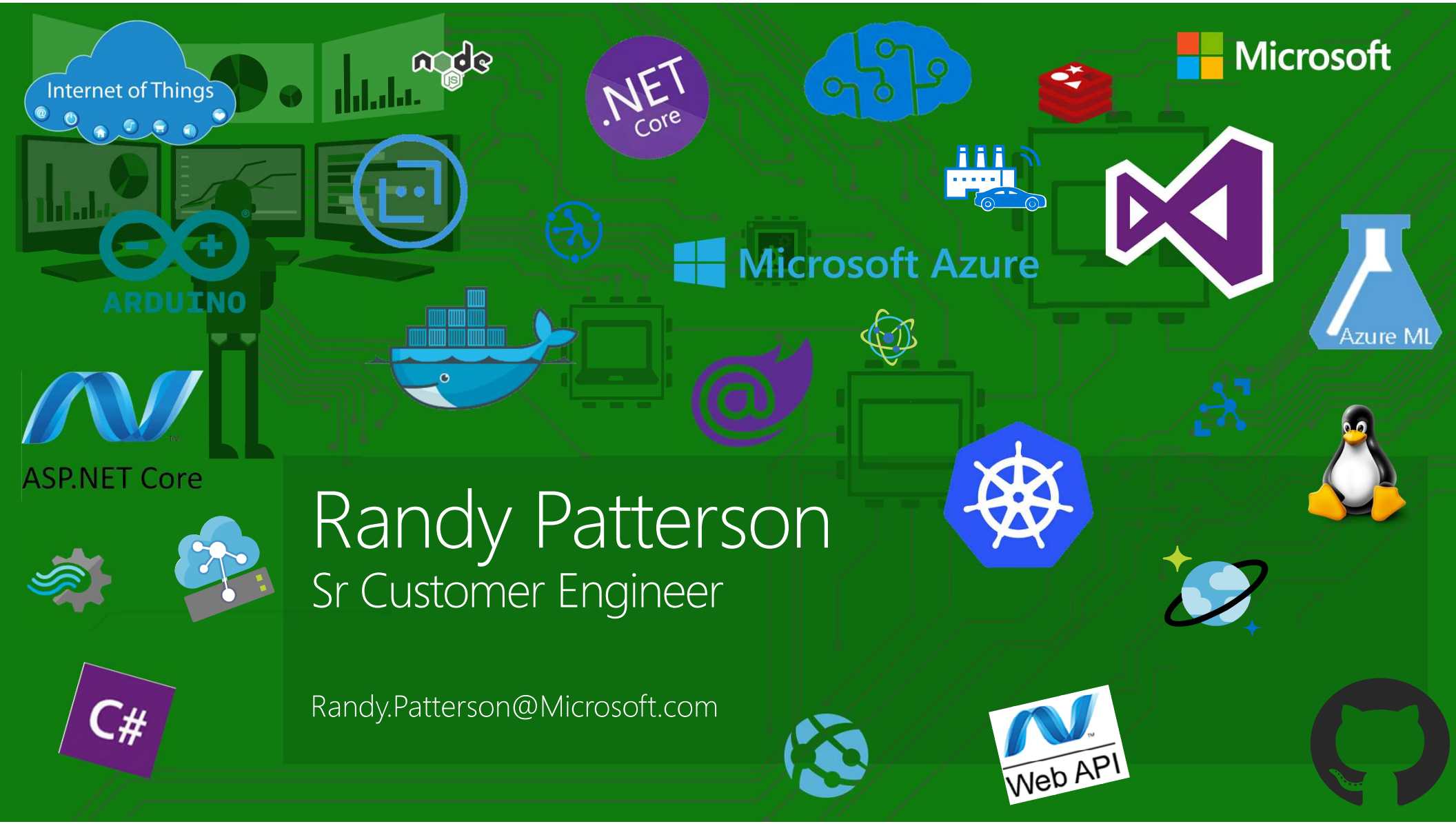
Containers as Infrastructure

Microsoft Services

Microsoft

Module 1 - Introduction to Containers

Microsoft Services



Randy Patterson

Sr Customer Engineer

Randy.Patterson@Microsoft.com

Agenda

Day 1

Introduction to Containers

Getting Started with Windows Containers

Day 2

Advanced Docker Topics

Microservices and Containers

Container Orchestrators

Day 3

DevOps with Containers

Monitoring and Troubleshooting Containers

Setup Labs

http://aka.ms/PremierEducation

Training Key: `DE2C1271004D4030`

Conditions and Terms of Use

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet website references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Copyright and Trademarks

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see **Use of Microsoft Copyrighted Content** at
https://www.microsoft.com/en-us/legal/intellectualproperty/permissions/default.aspx

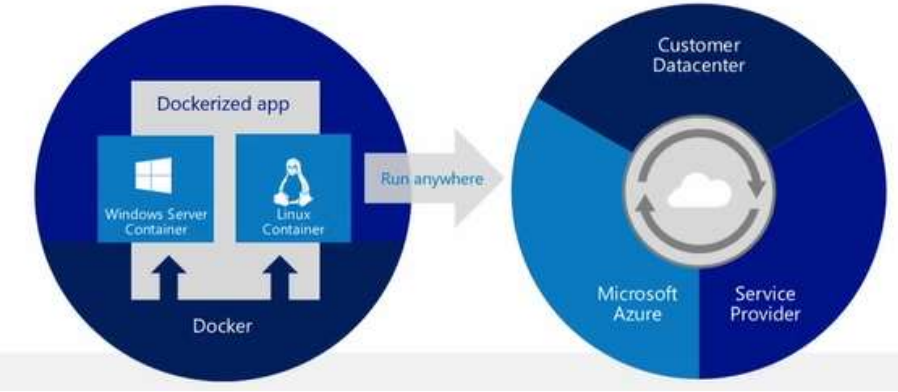
Microsoft®, Internet Explorer®, Outlook®, SkyDrive®, Windows Vista®, Zune®, Xbox 360®, DirectX®, Windows Server® and Windows® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Objectives

- Understand what Containers are
- Learn Docker Fundamentals (Docker Engine and Client)
- Understand Container Images and Docker Registry
- Learn How to Build Container Image using Dockerfile
- Learn how to Start, Stop, and Remove Docker Containers
- Understand use of Tags for Versioning Images

Why Docker?

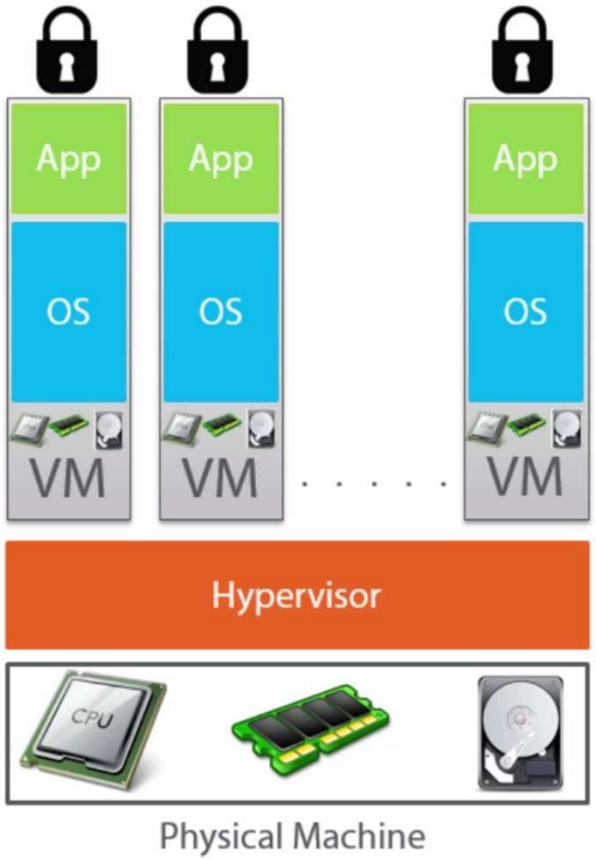
- Build any app in any language using any stack (OS)
- Dockerized apps can run anywhere on anything
- Prevents “It works on my machine”
- Applications are deployed with their dependencies installed and configured

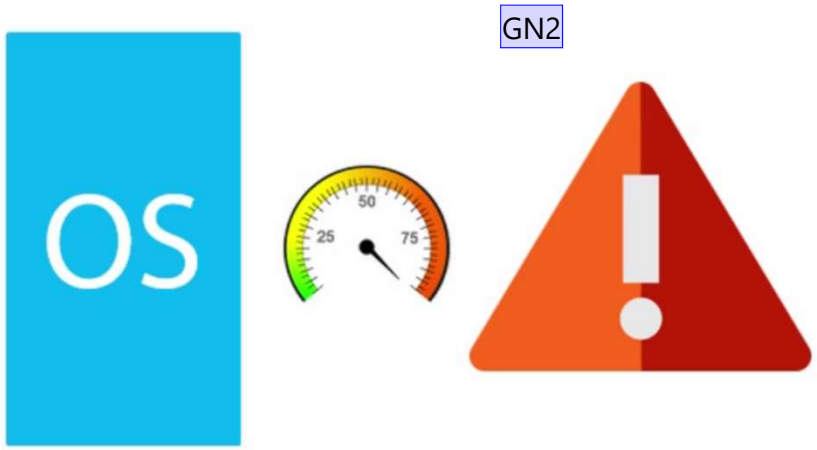


Docker Vocabulary

Host	A VM running the Docker Daemon to host a collection of Docker Containers
Client	Where docker commands are executed (client/server)
Image	An *ordered collection of filesystems (layers)* to be used when instancing a container (more on it later)
Container	A runtime instance of an image
Registry	A collection of docker images

Challenges with Virtualization





GN2

> OS != Business Value

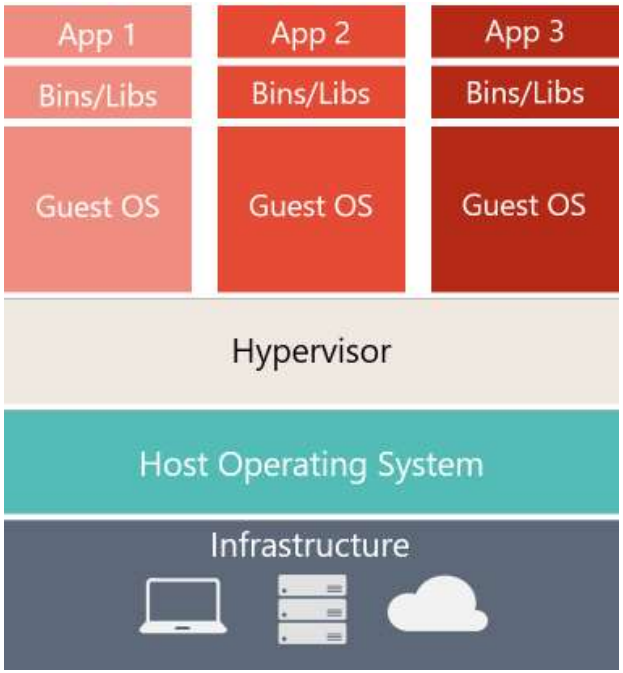
Slide 10

GN2 Maybe insert notes for the speaker to explain the 2nd challenge : here to explain that the OS could know an "over" utilization due to hosted application ?

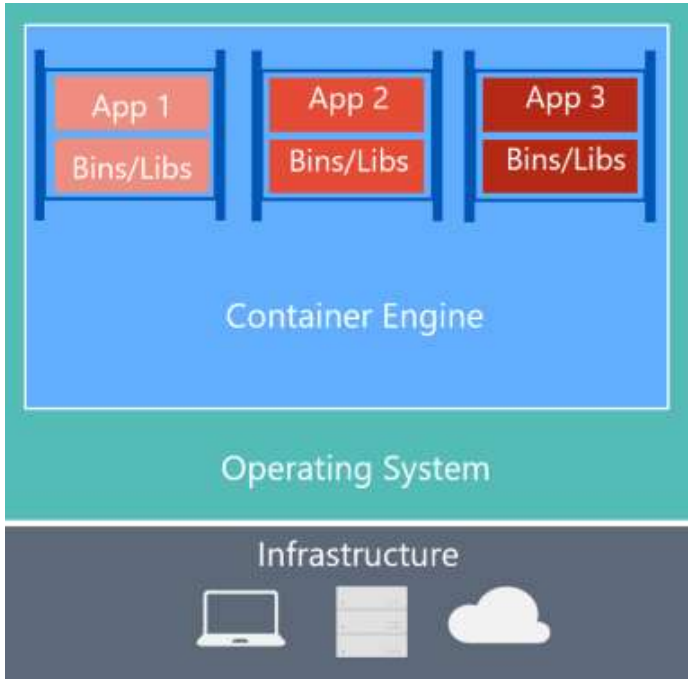
Guy Northee, 8/22/2017

Virtual Machines versus Containers

Virtual Machine



Container

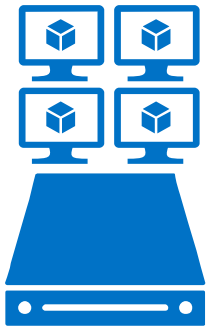


Containers

Physical

Applications traditionally built and deployed onto physical systems with 1:1 relationship

New applications often required new physical systems for isolation of resources



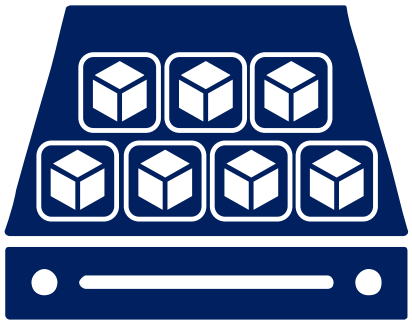
Virtual

Higher consolidation ratios and better utilization

Faster app deployment than in a traditional, physical environment

Apps deployed into VMs with high compatibility success

Apps benefited from key VM features i.e. Live migration, HA



Physical/Virtual

Package and run apps within

Containers

Key Benefits

Further accelerate of app deployment

Reduce effort to deploy apps

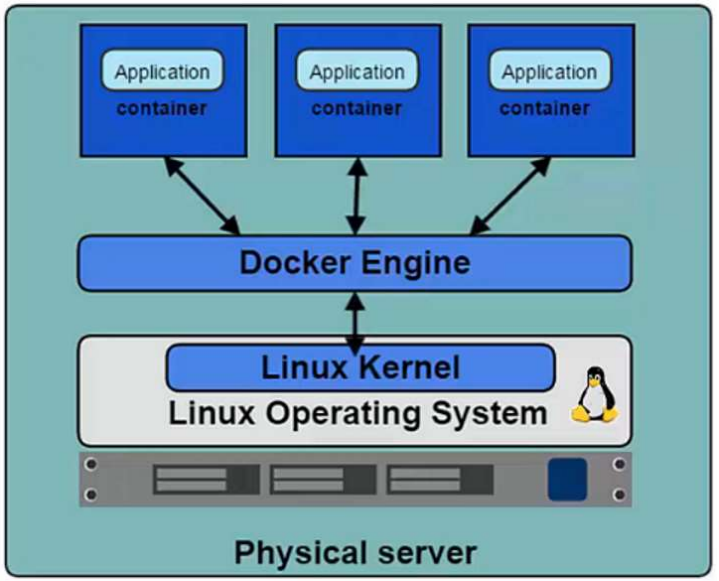
Streamline development and testing

Lower costs associated with app deployment

Increase server consolidation

Docker Platform

- Docker Engine (a.k.a. Docker Daemon)
  - The program that enables containers to be built, shipped, and run.
  - Uses Linux Kernel namespaces and control groups to give an isolated runtime environment for each application
- Docker Hub
  - A online registry of Docker images
- Docker Trusted Registry
  - Private on-site Registry for Docker images

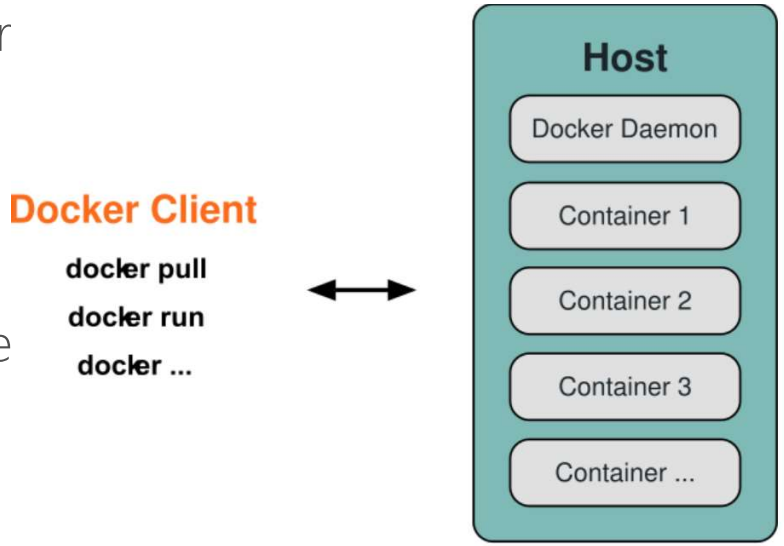


Docker Platform (Cont.)

- Docker Client
  - Takes user inputs and sends them to the Daemon.
  - Client and Daemon can run on the same host or on different hosts.

- Docker Images
  - Read-only template used to create containers.
  - Contains a set of instructions for creating the containers.

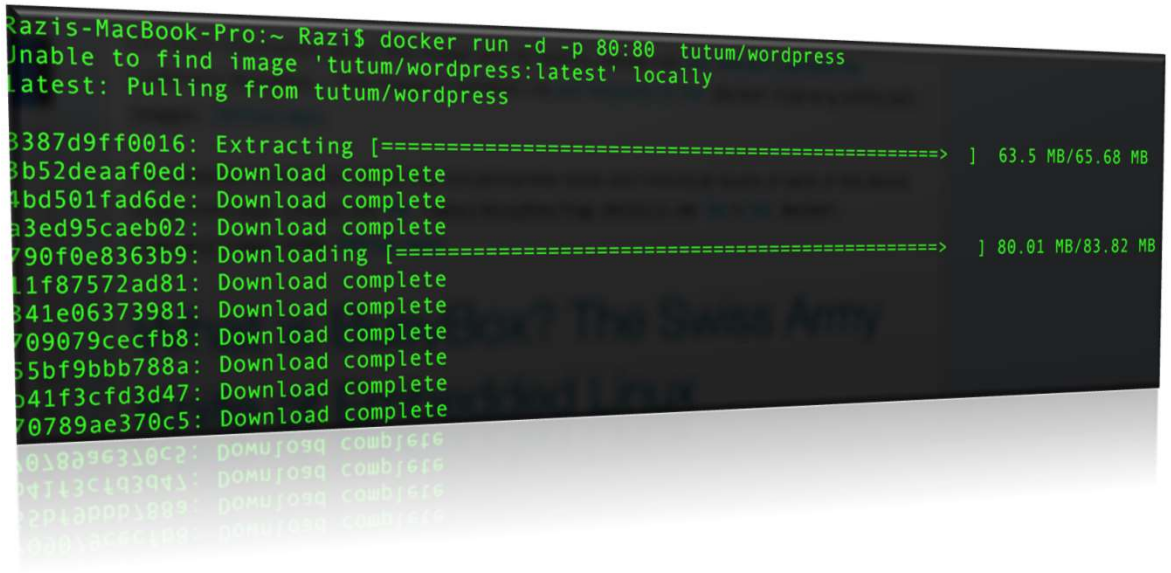
- Docker Containers
  - Isolated application platform based on one or more images.
  - Contains everything needed to run your application.



Quick Question?

How fast you can launch a fully functional WordPress blog engine?

How about multiple WordPress blog engines running side by side on same host?

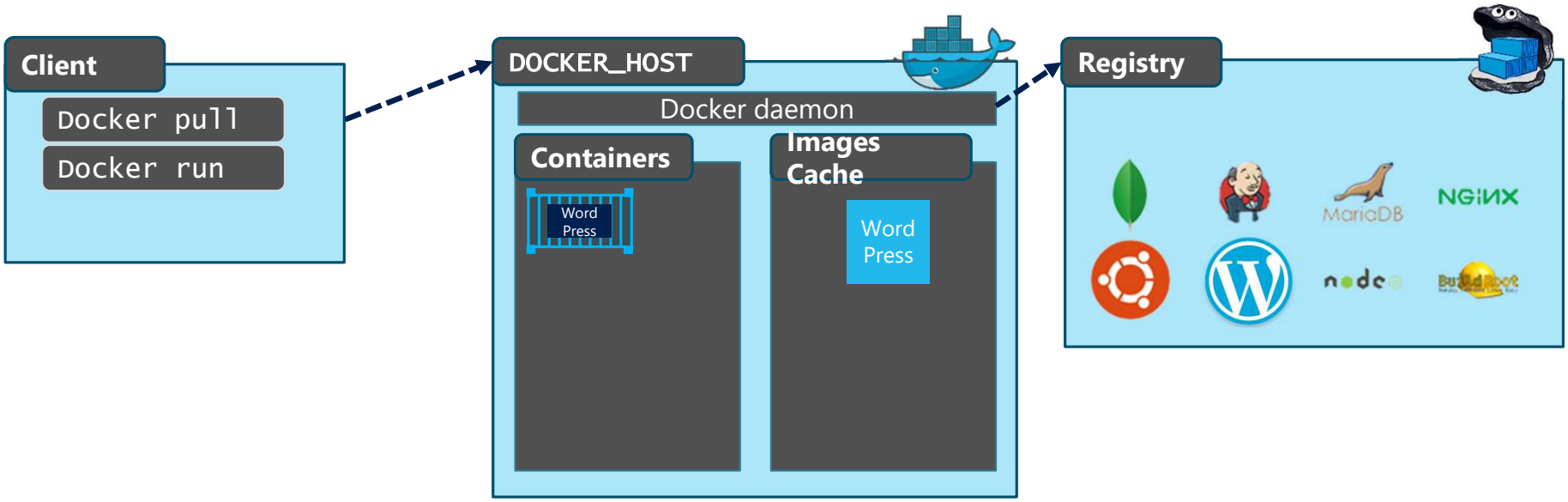


Demonstration: Running Docker Containers

Launch a single WordPress Container

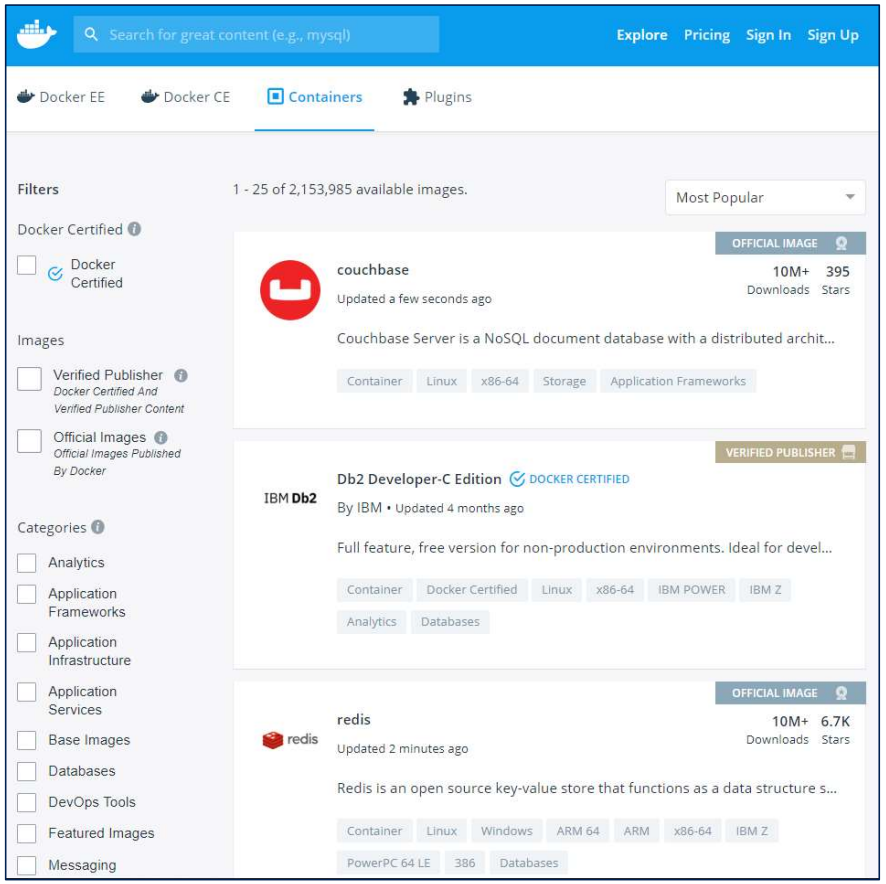
Running multiple WordPress Containers side by side

Docker In Action



Docker Registry

- Stores docker images
- Searchable
- Public Registry – hub.docker.com
- Private Registries – Instanced for you. E.g. Azure Container Registry
- The Registry is open-source under the permissive Apache License



Demonstration: Docker Registry

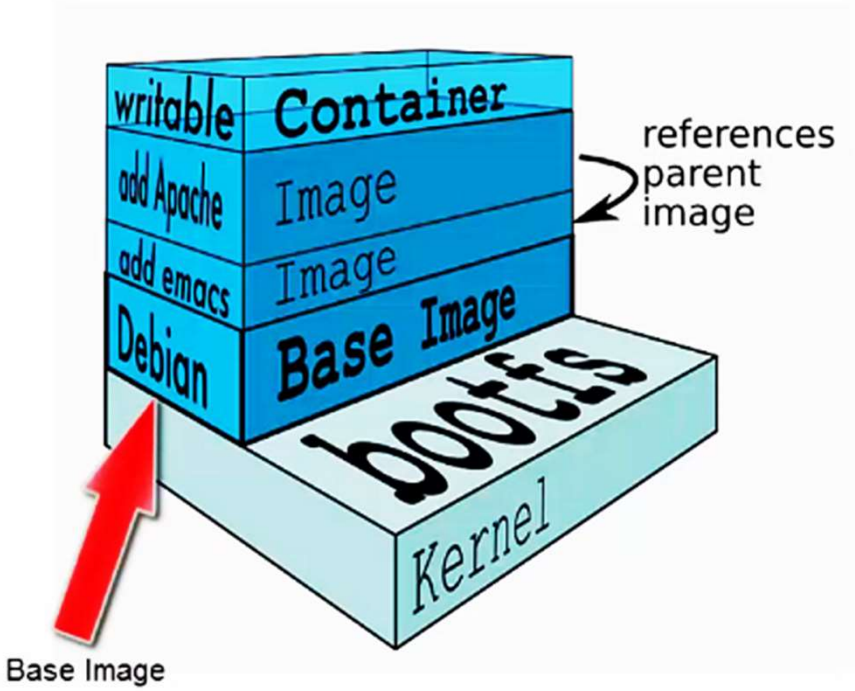
Search Docker Registry using Docker CLI

Search Images on DockerHub

Docker Image Naming Convention

Docker Images

- A Docker image is built up from a series of layers.
- Base platform OS image is provided by vendors like Microsoft for Windows OS image, Canonical for Ubuntu image etc. These images get published to DockerHub.
- Each layer represents an instruction in the image's Dockerfile.
- Each layer except the last one is read-only.



Demonstration: Docker Image Layers

List All Layers for Docker Image

Look at locally cached images

Dockerfile

- Text file with Docker commands in it to create a new image. You can think of it as a configuration file with set of instructions needed to assemble a new image.

- Docker has a docker build command that parses Dockerfile to build a new container image.

```
# Simple Dockerfile for NGINX

FROM nginx:stable-alpine

MAINTAINER Razi Rais

COPY index.html /usr/share/nginx/html/index.html

CMD ["nginx", "-g", "daemon off;"]
```

```
FROM microsoft/dotnet:1.1.0-sdk-projectjson

COPY . /app

WORKDIR /app

RUN ["dotnet", "restore"]

RUN ["dotnet", "build"]

EXPOSE 5000/tcp

CMD ["dotnet", "run", "--server.urls", "http://*:5000"]
```

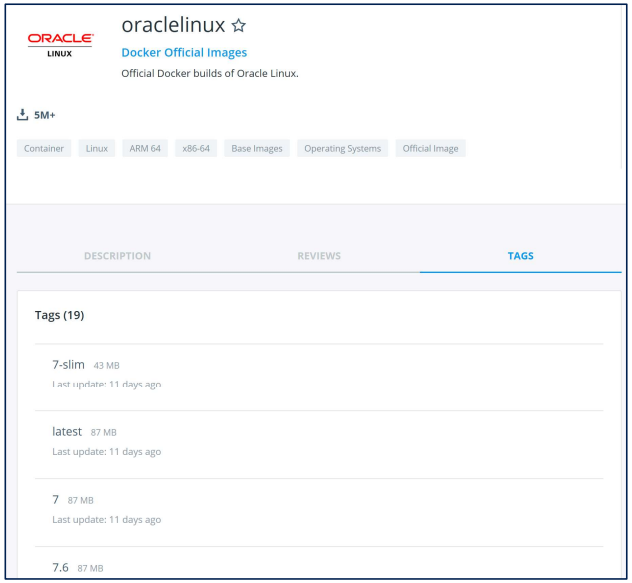
```
# Simple Dockerfile for NodeJS

FROM node:boron

MAINTAINER Razi Rais

# Create app directory
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# Install app dependencies
COPY package.json /usr/src/app/
RUN npm install

# Bundle app source
COPY . /usr/src/app

EXPOSE 8080

CMD [ "npm", "start" ]
```

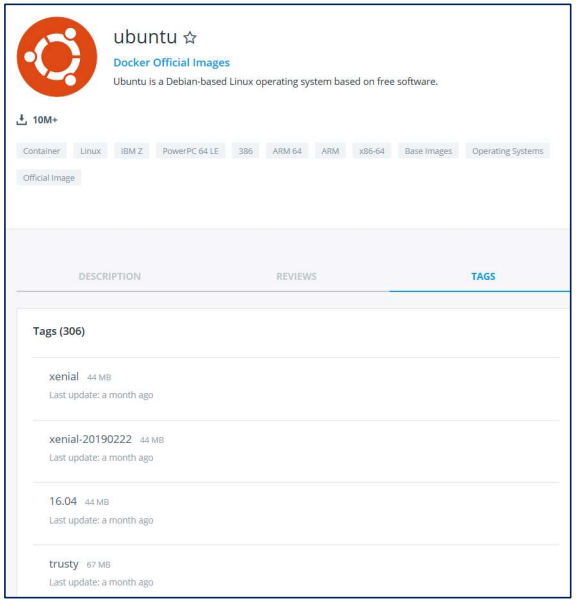
Common Dockerfile Instructions

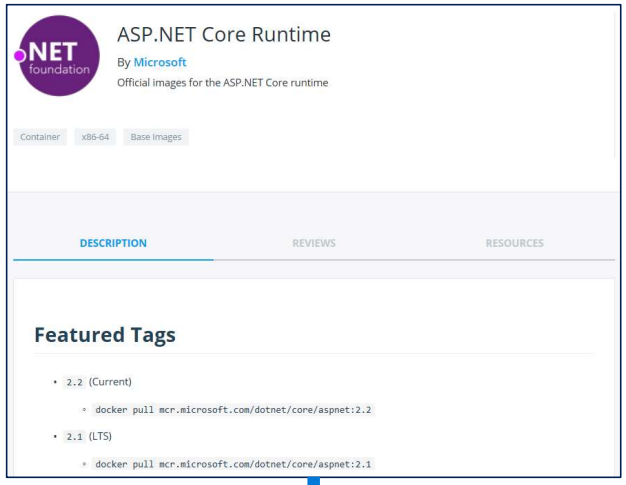
- FROM instruction initializes a new build stage and sets the Base Image for subsequent instructions.
- LABEL is a key-value pair, stored as a string. You can specify multiple labels for an object, but each key-value pair must be unique within an object.
- RUN will execute any commands in a new layer on top of the current image and commit the results.
- WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it.
- ADD instruction copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.
- COPY instruction copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.
- CMD provide defaults for an executing container. These defaults can include an executable.
- ENTRYPOINT allows you to configure a container that will run as an executable.
- EXPOSE instruction informs Docker that the container listens on the specified network port(s).

Image Tags

- A Tag name is a string value that you can use to distinguish versions of your Docker images so you can preserve older copies or variants of a primary build.
- You can group your images together using names and tags (if you don't provide any tag default value of latest is assumed)







Demonstration: Dockerfile and Docker Build

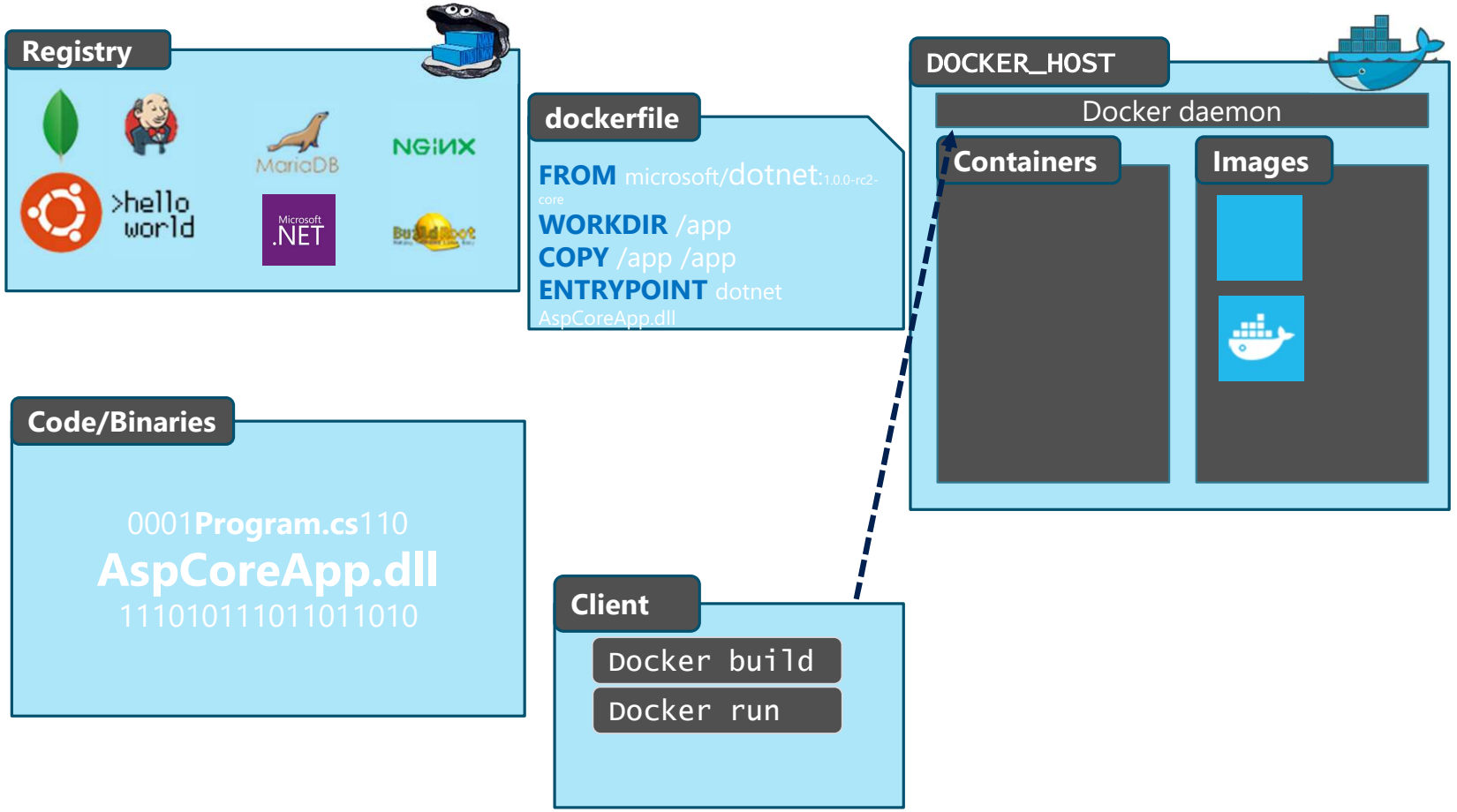
Working with Dockerfile

Build container images using Docker build command:

- NodeJs
- Nginx
- ASP.NET Core

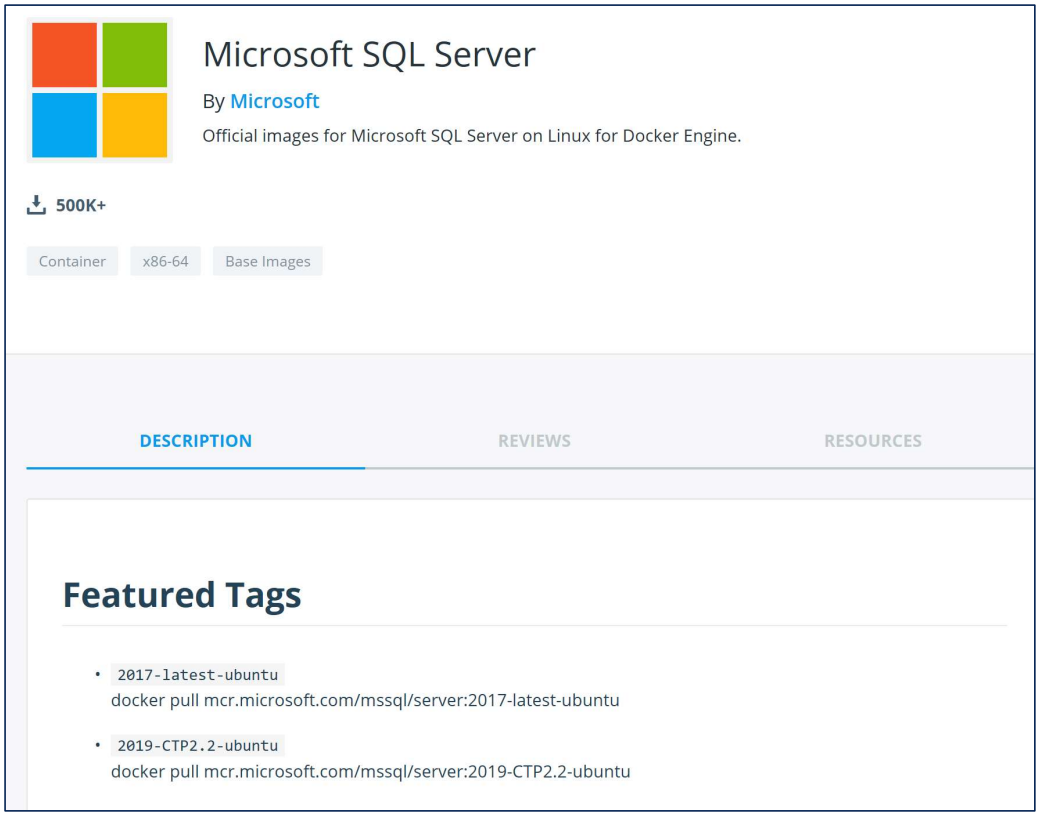
Using Image Tags

How does Docker build work?



SQL Server 2017 Container Image

- SQL Server in Linux can be packaged in a Docker container
- Can be used in automated tests to pre-populate a SQL Server instance with test data on-demand
- Can run on top of Ubuntu 16.04, RHEL and CentOS



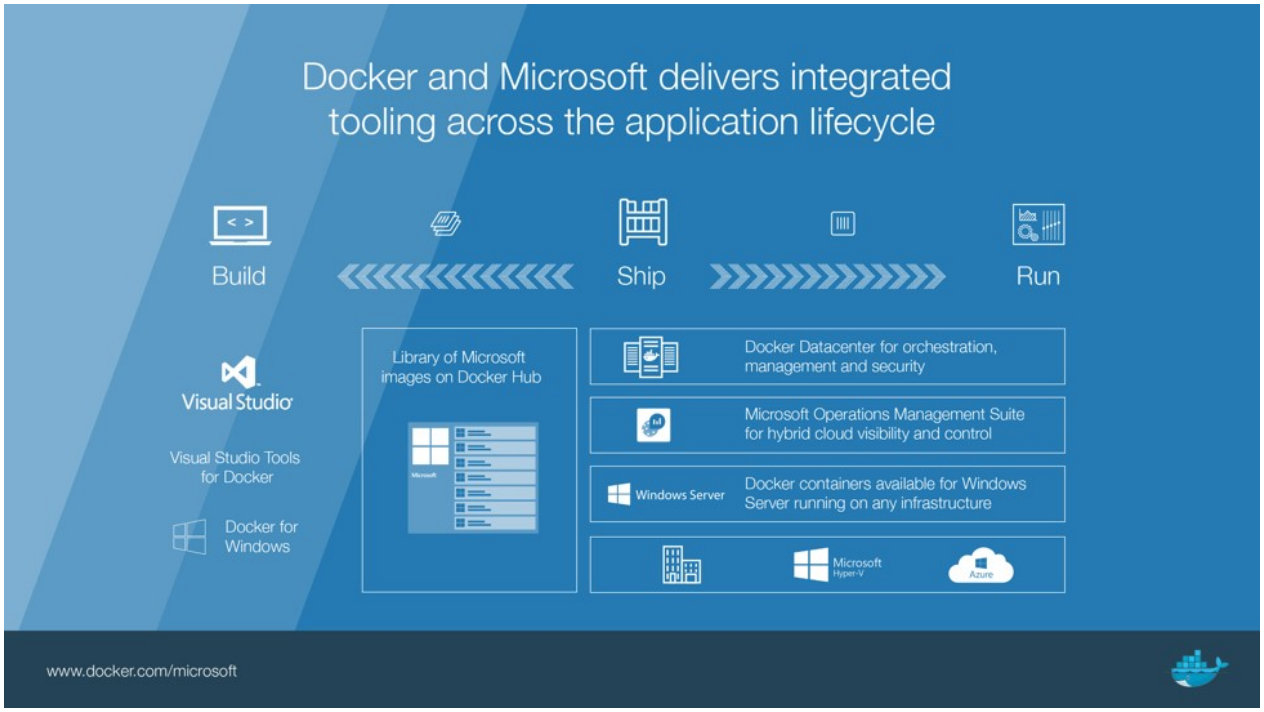
Demonstration: *Running SQL Server 2017 inside Container*

SQL Server 2017 Container Image

Using Custom Database with SQL Server 2017 running inside a Container

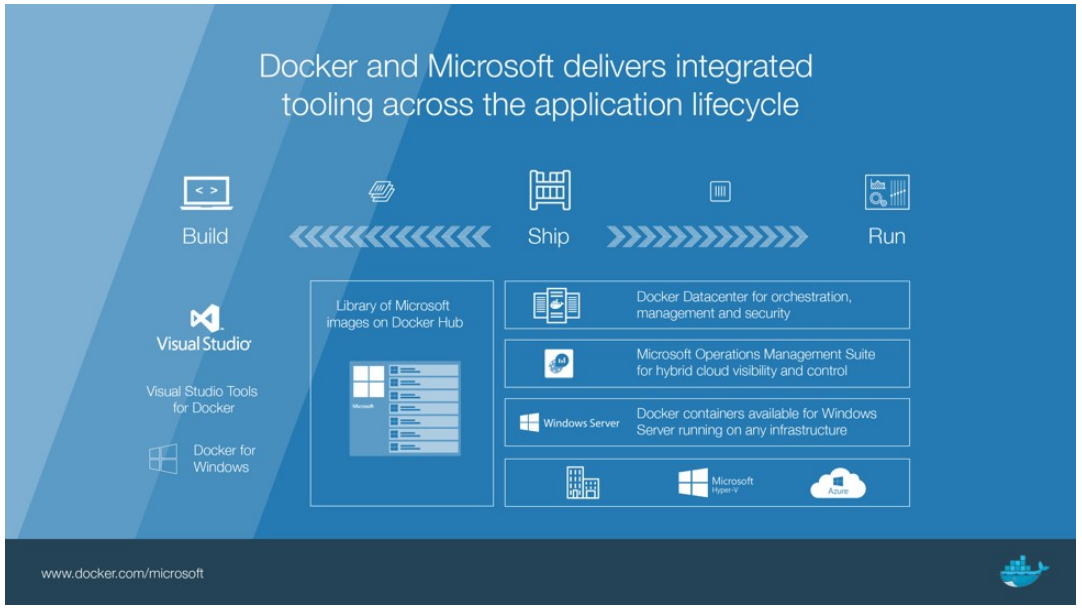
Docker and Microsoft Partnership

- Docker Engine is tested, validated, and supported on Windows Server 2016/2019 and Windows 10 at no additional cost.
- Microsoft provides Windows Server 2016/2019 customers enterprise support for Docker Enterprise Edition.
- Docker is supported throughout Microsoft Cloud and on-premises ecosystem.



Docker and Microsoft Partnership (Cont.)

- For developers, the integration of **Visual Studio Tools for Docker** and **Docker Desktop** provides complete desktop development environments for building Dockerized Windows apps
- To jumpstart app development, Microsoft has contributed **Windows Server container base images and apps to Docker Hub**
- For IT pros, **Docker Enterprise for Azure** is designed to manage Windows Server environments in addition to the Linux environments with the Docker Universal Control Plane



Notes (hidden)

NEXT: <next slide title>

Lab M01: Introduction to Containers

Running Your First Container

Working with Docker Command Line Interface (CLI)

Building Custom Container Images with Dockerfile

Interaction with a Running Container

Tagging

Windows Containers

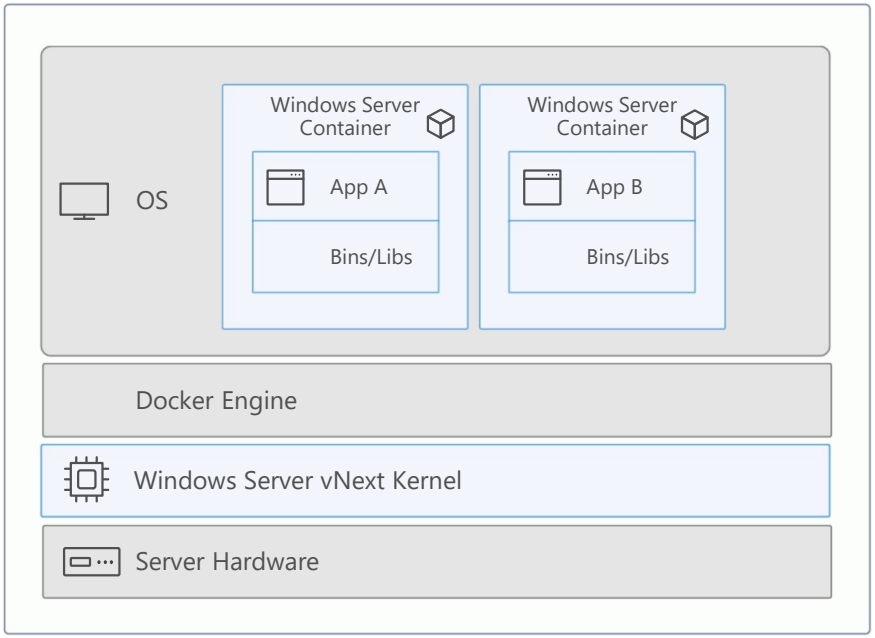
Windows Containers

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

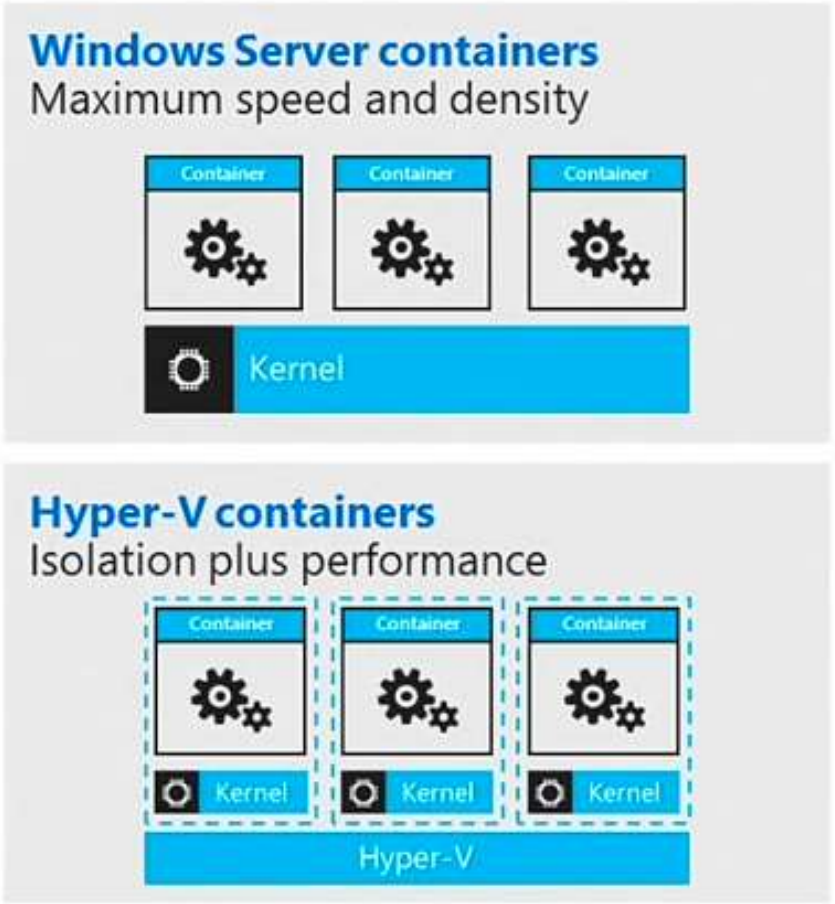
Each application/container could have it's own IP address to provide a layer of isolation so that the container doesn't have access outside of its sandboxed execution environment



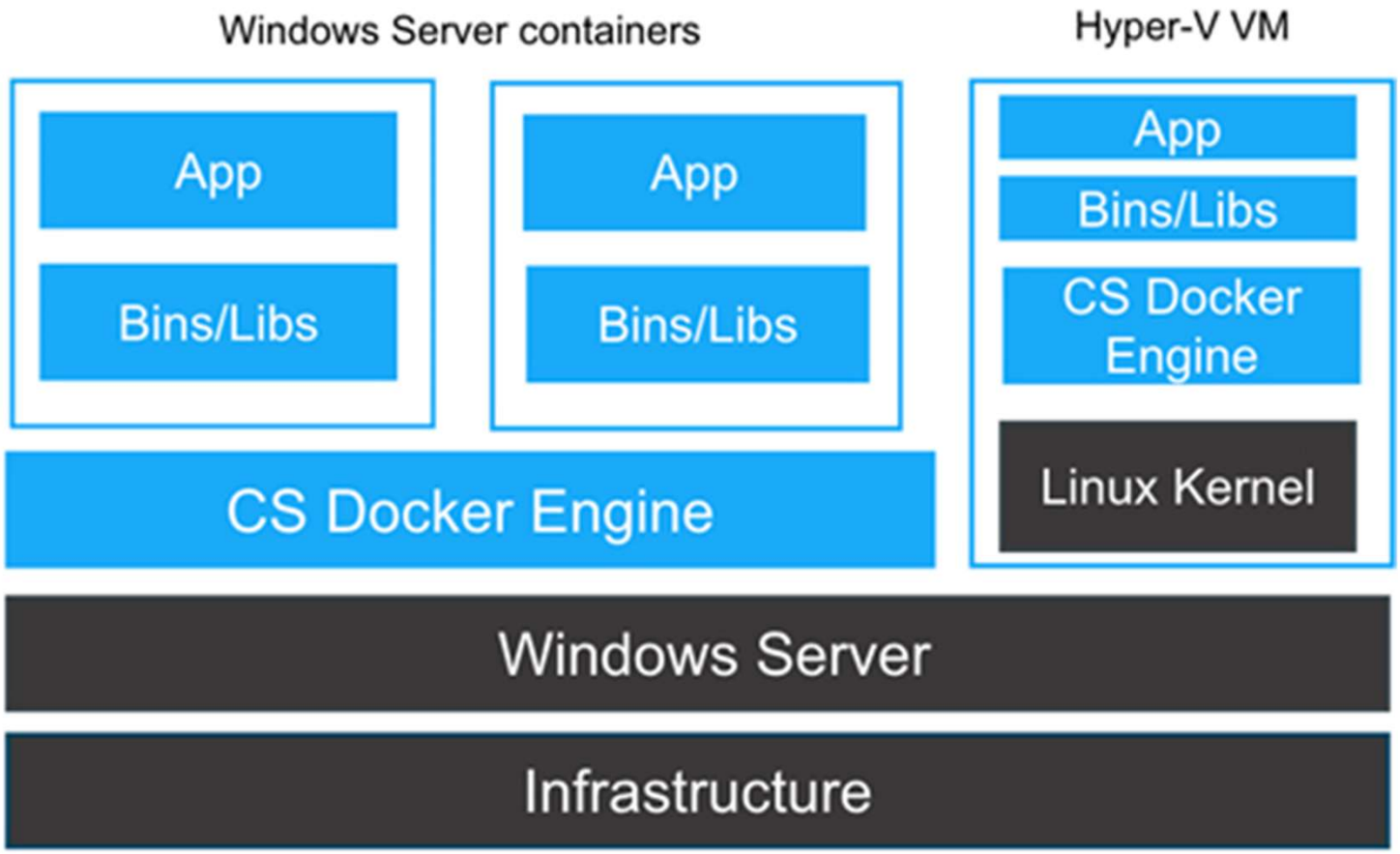
How Hyper-V Containers are Different?

- 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.



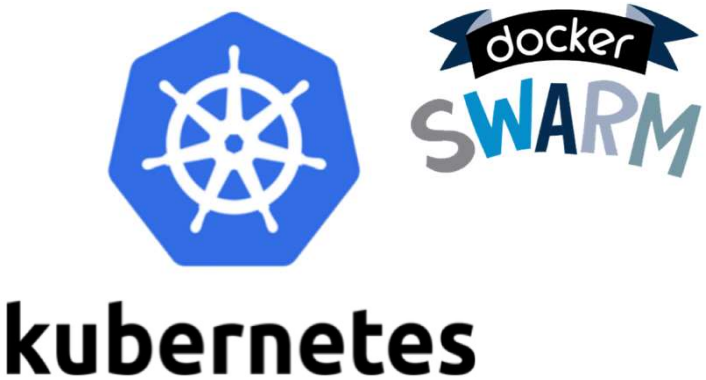
Hyper V Containers with Linux



Demonstration: Isolation

Docker Desktop for Windows

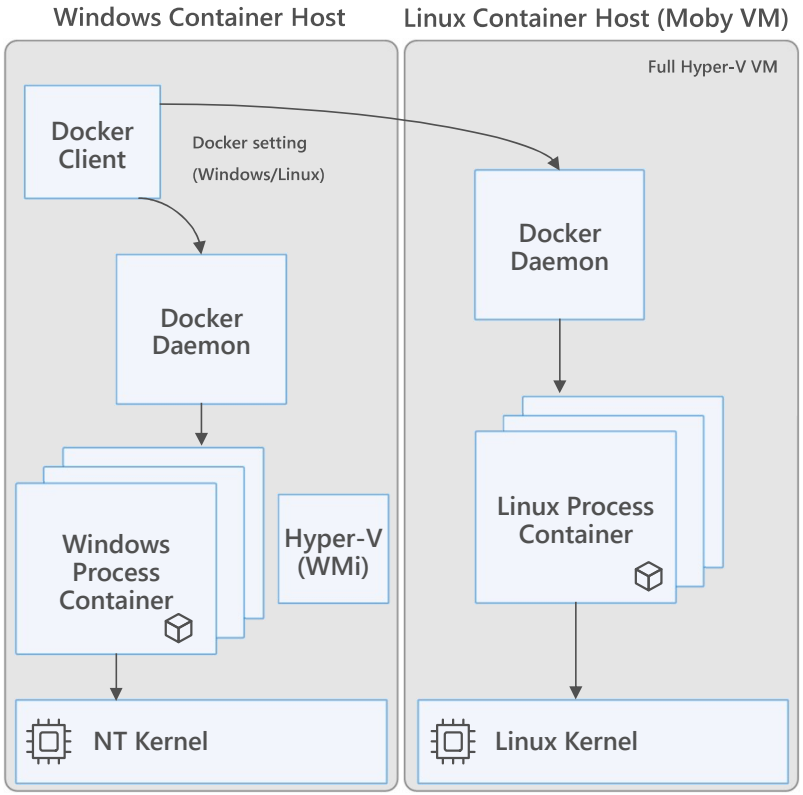
- By installing "Docker Desktop for Windows", Docker developers can use a single Docker CLI to build apps for both Windows or Linux
- Includes everything you need to build, test and ship containerized applications right from your machine
- Integrated tools including the Docker command line, Docker Compose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



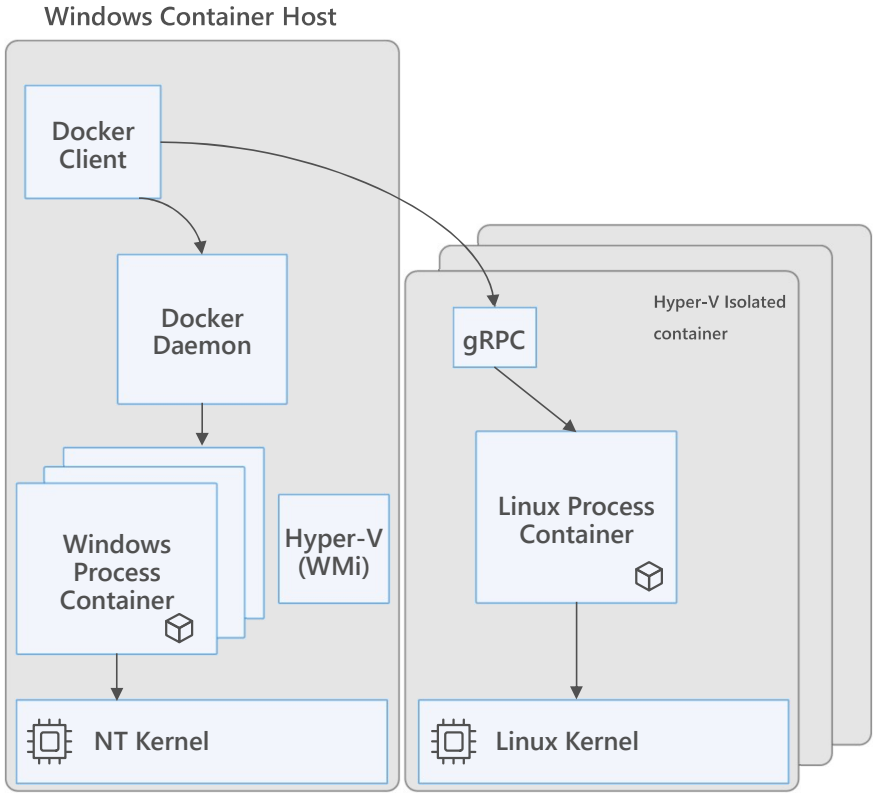
Linux Containers on Windows

Right now there are two ways to run Linux containers with Docker for Windows and Hyper-V

Run Linux containers in a full Linux VM
What Docker typically does today



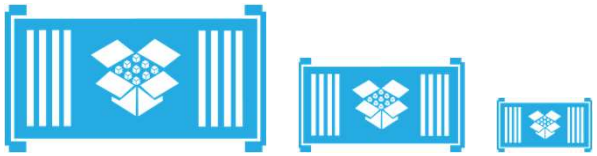
Run Linux containers with Hyper-V isolation (LCOW)
This is a new option in Docker for Windows



Server Core

Server Core Container Image

- Image size reduced by 50% in version 1709, another 30% in version 1803, and down to 1.5 GB in Windows Server 2019
- Unused optional features removed (can be added as needed) / List removed from version 1803 →
- Application compatibility improvements in each release
- Boot and run-time performance improved in each release



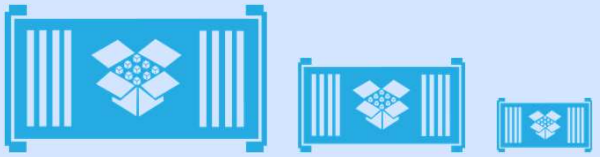
Windows Server, version 1709 and 1803

- AD Certificate Services
- AuthManager, Bitlocker
- BITS, CCFFilter, Containers
- CoreFileServer
- DataCenterBridging
- Dedup-Core
- DeviceHealthAttestationService
- DFSN and DFSR
- DirectoryServices
- DiskIo-QoS
- EnhancedStorage
- FailoverCluster
- File-Services
- FileServerVSSAgent
- FRS-Infrastructure , FSRM
- HardenedFabricEncryptionTask
- HostGuardian
- IdentityServer-SecurityTokenService
- IPAM, iSCSI, iSNS Service
- Licensing, LightweightServer
- Hyper-V, Group Policy
- Windows-FCI-Client-Package
- Windows-Subsystem-Linux
- MSRDC-Infrastructure
- MultipathIo , NetworkController
- NDES
- NetworkLoadBalancingFullServer
- NetworkVirtualization
- NFS Client and Server
- OnlineRevocationServices
- P2P-PnrpOnly, PeerDist
- Printing, QWAVE
- RasRoutingProtocols
- Remote-Desktop-Services
- RemoteAccess, ResumeKeyFilter
- RightsManagementServices-Role
- SBMgr-UI
- ServerCore-Drivers
- RSAT
- ServerMediaFoundation
- ServerMigration
- SessionDirectory
- SetupAndBootEventCollection
- ShieldedVMToolsAdminPack
- SMB1Protocol-Server
- SmbDirect
- SMBHashGeneration
- SmbWitness
- SNMP
- SoftwareLoadBalancer
- Storage-Replica-AdminPack
- Storage-Replica
- Tpm-PSH-Cmdlets
- UpdateServices-Database
- UpdateServices-Services
- UpdateServices-WidDatabase
- UpdateServices
- VmHostAgent
- VolumeActivation-Full-Role
- Web-Application-Proxy
- WebAccess
- WebEnrollmentServices
- Windows-Defender
- WindowsServerBackup
- WindowsStorageManagementService
- WINSRuntime
- WMISnmpProvider
- WorkFolders-Server
- WSS-Product-Package

Nano Server

Nano Server Container Image

- Image size reduced by over 80% in version 1709, and as of version 1803, image is below 100mb
- Optimized for .Net Core 2.0
- PowerShell Core, .NET Core and WMI no longer included by default in Container Image



Windows Server, version 1709 and 1803

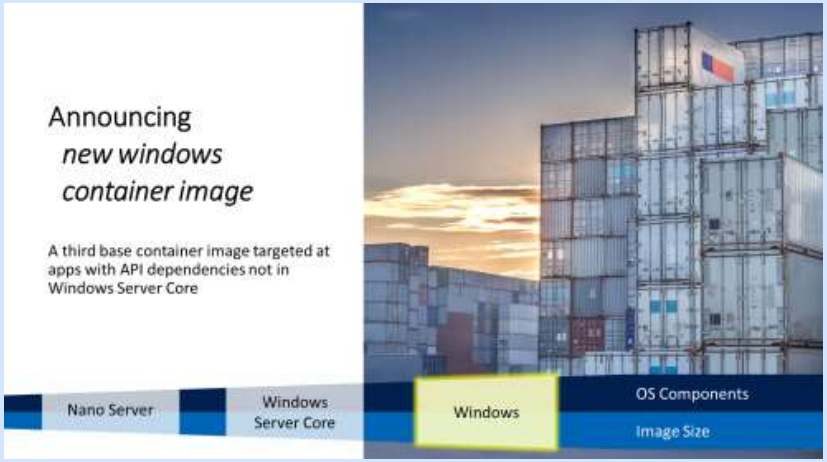
Windows

New! “Windows” Container Image

Based on customer feedback, Microsoft is introducing a third base image called “Windows” that supports applications with additional API dependencies.

Examples:

- Customers interested in moving legacy apps in containers, however some components were missing
- Customers interested in leveraging containers to run automated User Interface tests and needed graphic capabilities like DirectX



Windows Server 2019

MS Docs: aka.ms/WindowsContainer

Windows Base OS Image Summary

Nano Server	Windows Server Core	Windows
Born in the cloud applications	App Compatibility	Automation Workloads
.NET core Support	Full .NET framework support	Carries most Windows OS components
94 MB Image Size	1.4 GB Image Size	3.5 GB Image Size

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

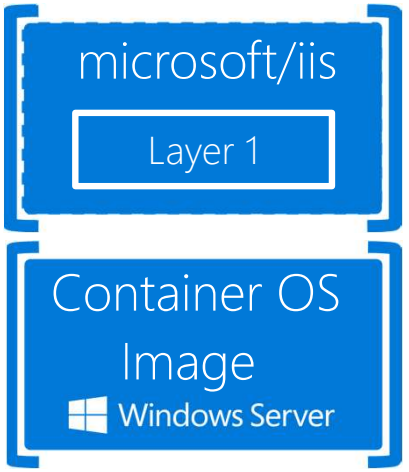
Working with Windows Server Core Container

Working with Nano Server Container

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

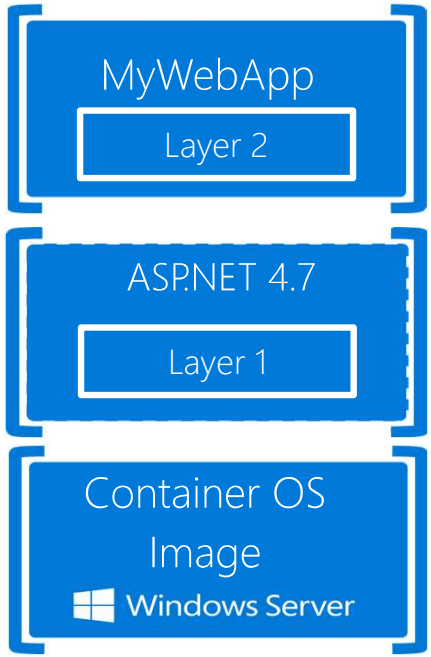
```
FROM mcr.microsoft.com/windows/servercore:ltsc2019
```

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

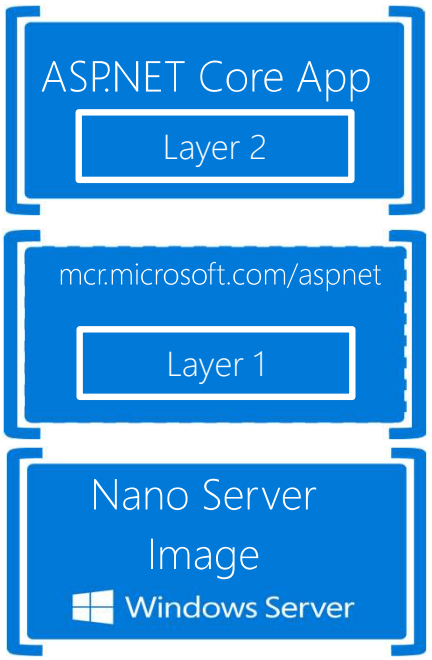
```
FROM mcr.microsoft.com/dotnet/core/aspnet:2.2.3-nanoserver-1809
```

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

Demonstration: *Building and Running IIS Server Container*

Build IIS Container Image using Dockerfile

Run IIS Container

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 2019

Debugging ASP.NET Core Application using Visual Studio

Lab M02: Getting Started with Windows Containers

Knowledge Check

- Question #1
- Question #2
- Question #3