INTRODUCTION TO CONTAINERS

Vikash Verma

What is a Container?

Containerization is an approach to software development in which an application or service, its dependencies, and its configuration (abstracted as deployment manifest files) are packaged together as a container image.

The containerized application can be tested as a unit and deployed as a container image instance to the host operating system (OS).

Containers also isolate applications from each other on a shared OS.

Containerized applications run on top of a container host that in turn runs on the OS (Linux or Windows). Containers therefore have a significantly smaller footprint than virtual machine (VM) images.

Another benefit of containerization is scalability. You can scale out quickly by creating new containers for short-term tasks.

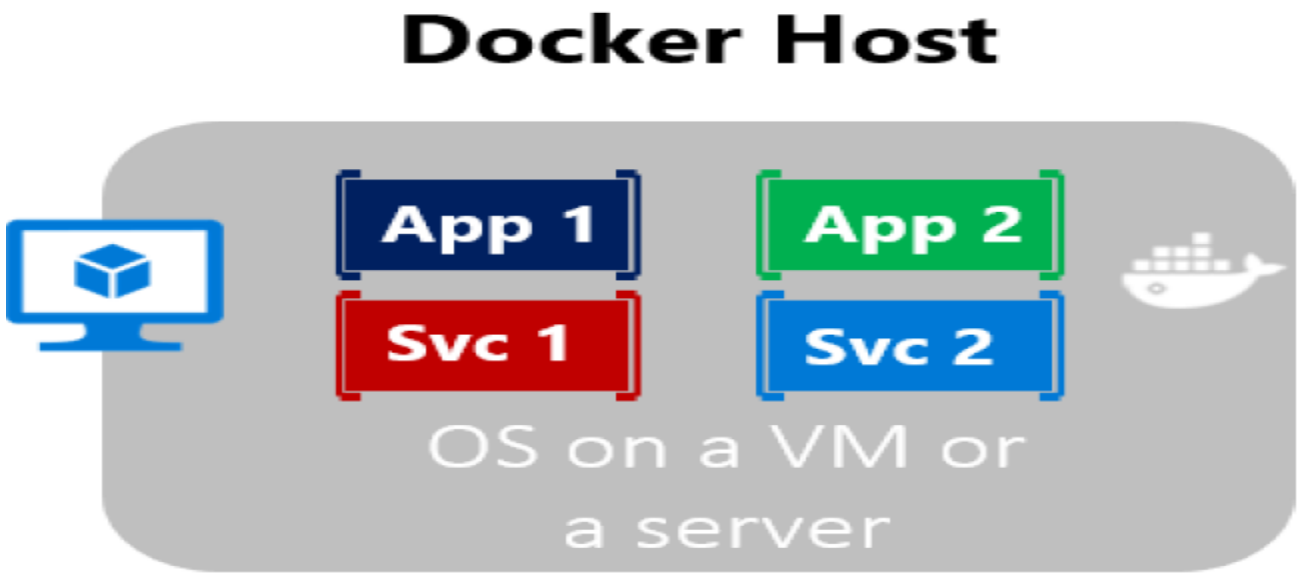
From an application point of view, instantiating an image (creating a container) is similar to instantiating a process like a service or web app.

For reliability, however, when you run multiple instances of the same image across multiple host servers, you typically want each container (image instance) to run in a different host server or VM in different fault domains.

What is a Container?

Each container can run a whole web application or a service

In this example, Docker host is a container host, and App1, App2, Svc 1, and Svc 2 are containerized applications or services.



What is Docker?

Docker is an open-source project for automating the deployment of applications as portable, self-sufficient containers that can run on the cloud or on-premises.

Docker is also a company that promotes and evolves this technology, working in collaboration with cloud, Linux, and Windows vendors, including Microsoft.

Docker image containers can run natively on Linux and Windows. However, Windows images can run only on Windows hosts and Linux images can run on Linux hosts and Windows hosts (using a Hyper-V Linux VM, so far), where host means a server or a VM.

Developers can use development environments on Windows, Linux, or macOS. On the development computer, the developer runs a Docker host where Docker images are deployed, including the app and its dependencies.

Developers who work on Linux or on the Mac use a Docker host that is Linux based, and they can create images only for Linux containers.

What is Docker?

To run Windows Containers, there are two types of runtimes:

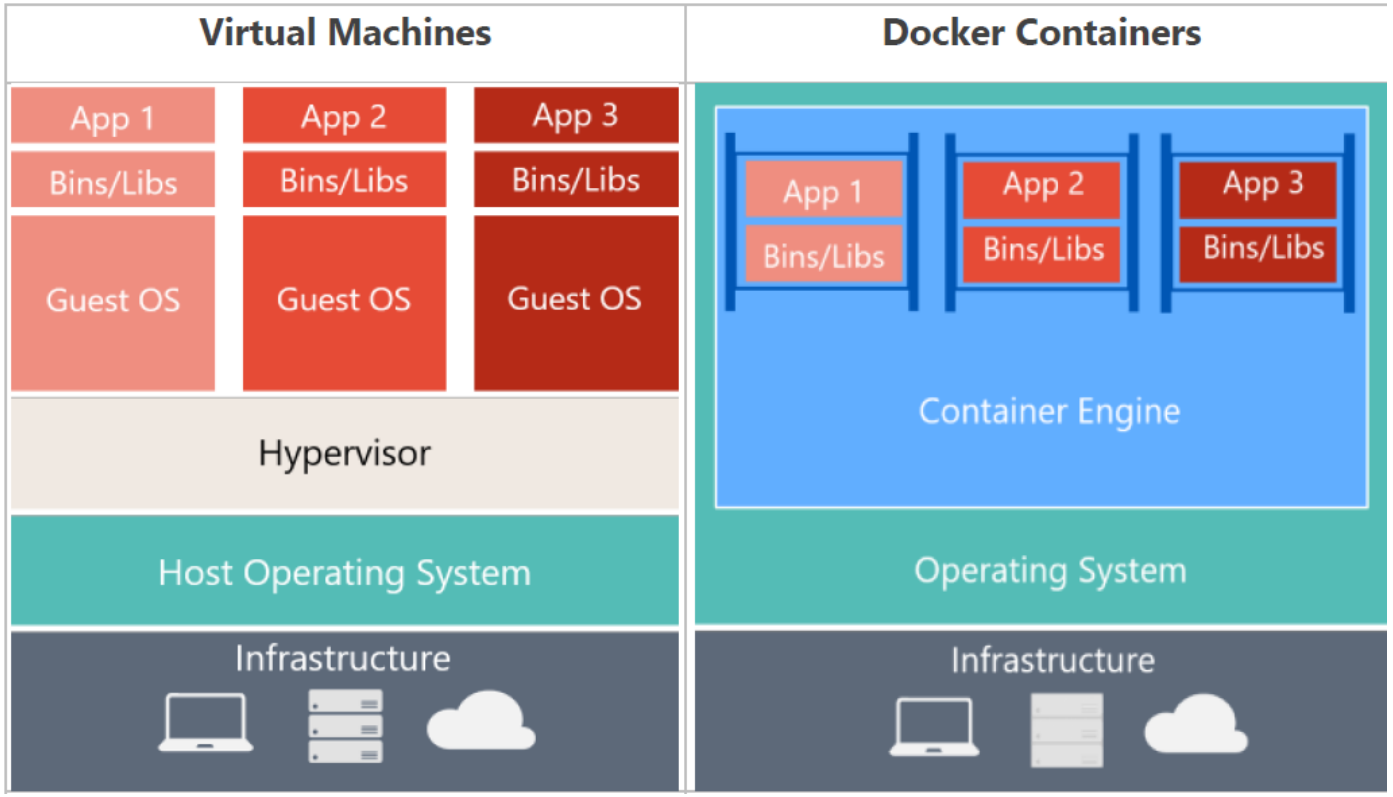
Windows Server Containers provide application isolation through process and namespace isolation technology.

A Windows Server Container shares a kernel with the container host and with all containers running on the host.

Hyper-V Containers expand on the isolation provided by Windows Server Containers by running each container in a highly optimized virtual machine.

In this configuration, the kernel of the container host isn’t shared with the Hyper-V Containers, providing better isolation

Docker containers with virtual machines



Docker containers with virtual machines

Because containers require far fewer resources (for example, they don’t need a full OS), they’re easy to deploy and they start fast. This allows you to have higher density, meaning that it allows you to run more services on the same hardware unit, thereby reducing costs.

As a side effect of running on the same kernel, you get less isolation than VMs.

The main goal of an image is that it makes the environment (dependencies) the same across different deployments. This means that you can debug it on your machine and then deploy it to another machine with the same environment guaranteed.

A container image is a way to package an app or service and deploy it in a reliable and reproducible way. You could say that Docker isn’t only a technology but also a philosophy and a process.

Docker terminology

Container image:

A package with all the dependencies and information needed to create a container. An image includes all the dependencies (such as frameworks) plus deployment and execution configuration to be used by a container runtime. Usually, an image derives from multiple base images that are layers stacked on top of each other to form the container's filesystem. An image is immutable once it has been created.

Dockerfile:

A text file that contains instructions for how to build a Docker image. It's like a batch script, the first line states the base image to begin with and then follow the instructions to install required programs, copy files and so on, until you get the working environment you need.

Build:

The action of building a container image based on the information and context provided by its Dockerfile, plus additional files in the folder where the image is built. You can build images with the Docker docker build command.

Docker terminology

Container:

An instance of a Docker image. A container represents the execution of a single application, process, or service. It consists of the contents of a Docker image, an execution environment, and a standard set of instructions. When scaling a service, you create multiple instances of a container from the same image. Or a batch job can create multiple containers from the same image, passing different parameters to each instance.

Volumes:

Offer a writable filesystem that the container can use. Since images are read-only but most programs need to write to the filesystem, volumes add a writable layer, on top of the container image, so the programs have access to a writable filesystem. The program doesn't know it is accessing a layered filesystem, it is just the filesystem as usual. Volumes live in the host system and are managed by Docker.

Tag:

A mark or label you can apply to images so that different images or versions of the same image (depending on the version number or the target environment) can be identified.

Docker terminology

Multi-stage Build:

Is a feature, since Docker 17.05 or higher, that helps to reduce the size of the final images. In a few sentences, with multi-stage build you can use, for example, a large base image, containing the SDK, for compiling and publishing the application and then using the publishing folder with a small runtime-only base image, to produce a much smaller final image

Repository (repo):

A collection of related Docker images, labeled with a tag that indicates the image version. Some repos contain multiple variants of a specific image, such as an image containing SDKs (heavier), an image containing only runtimes (lighter), etc. Those variants can be marked with tags. A single repo can contain platform variants, such as a Linux image and a Windows image.

Registry:

A service that provides access to repositories. The default registry for most public images is Docker Hub (owned by Docker as an organization). A registry usually contains repositories from multiple teams. Companies often have private registries to store and manage images they've created. Azure Container Registry is another example.

Docker terminology

Multi-arch image:

For multi-architecture, is a feature that simplifies the selection of the appropriate image, according to the platform where Docker is running, e.g. when a Dockerfile requests a base image FROM microsoft/dotnet:2.2-sdk from the registry it actually gets 2.2-sdk-nanoserver-1709, 2.2-sdk-nanoserver-1803, 2.2-sdk-nanoserver-1809 or 2.2-sdk-alpine, depending on the operating system and version where Docker is running.

Docker Hub:

A public registry to upload images and work with them. Docker Hub provides Docker image hosting, public or private registries, build triggers and web hooks, and integration with GitHub and Bitbucket.

Azure Container Registry:

A public resource for working with Docker images and its components in Azure. This provides a registry that is close to your deployments in Azure and that gives you control over access, making it possible to use your Azure Active Directory groups and permissions.

Docker terminology

Docker Trusted Registry (DTR):

A Docker registry service (from Docker) that can be installed on-premises so it lives within the organization's datacenter and network. It is convenient for private images that should be managed within the enterprise. Docker Trusted Registry is included as part of the Docker Datacenter product. For more information, see Docker Trusted Registry (DTR).

Docker Community Edition (CE):

Development tools for Windows and macOS for building, running, and testing containers locally. Docker CE for Windows provides development environments for both Linux and Windows Containers. The Linux Docker host on Windows is based on a Hyper-V virtual machine. The host for Windows Containers is directly based on Windows. Docker CE for Mac is based on the Apple Hypervisor framework and the xhyve hypervisor, which provides a Linux Docker host virtual machine on Mac OS X. Docker CE for Windows and for Mac replaces Docker Toolbox, which was based on Oracle VirtualBox.

Docker Enterprise Edition (EE):

An enterprise-scale version of Docker tools for Linux and Windows development.

Docker terminology

Compose:

A command-line tool and YAML file format with metadata for defining and running multi-container applications. You define a single application based on multiple images with one or more .yml files that can override values depending on the environment. After you have created the definitions, you can deploy the whole multi-container application with a single command (docker-compose up) that creates a container per image on the Docker host.

Cluster:

A collection of Docker hosts exposed as if it were a single virtual Docker host, so that the application can scale to multiple instances of the services spread across multiple hosts within the cluster. Docker clusters can be created with Kubernetes, Azure Service Fabric, Docker Swarm and Mesosphere DC/OS.

Orchestrator:

A tool that simplifies management of clusters and Docker hosts. Orchestrators enable you to manage their images, containers, and hosts through a command line interface (CLI) or a graphical UI.

You can manage container networking, configurations, load balancing, service discovery, high availability, Docker host configuration, and more.

An orchestrator is responsible for running, distributing, scaling, and healing workloads across a collection of nodes. Typically, orchestrator products are the same products that provide cluster infrastructure, like Kubernetes and Azure Service Fabric, among other offerings in the market.

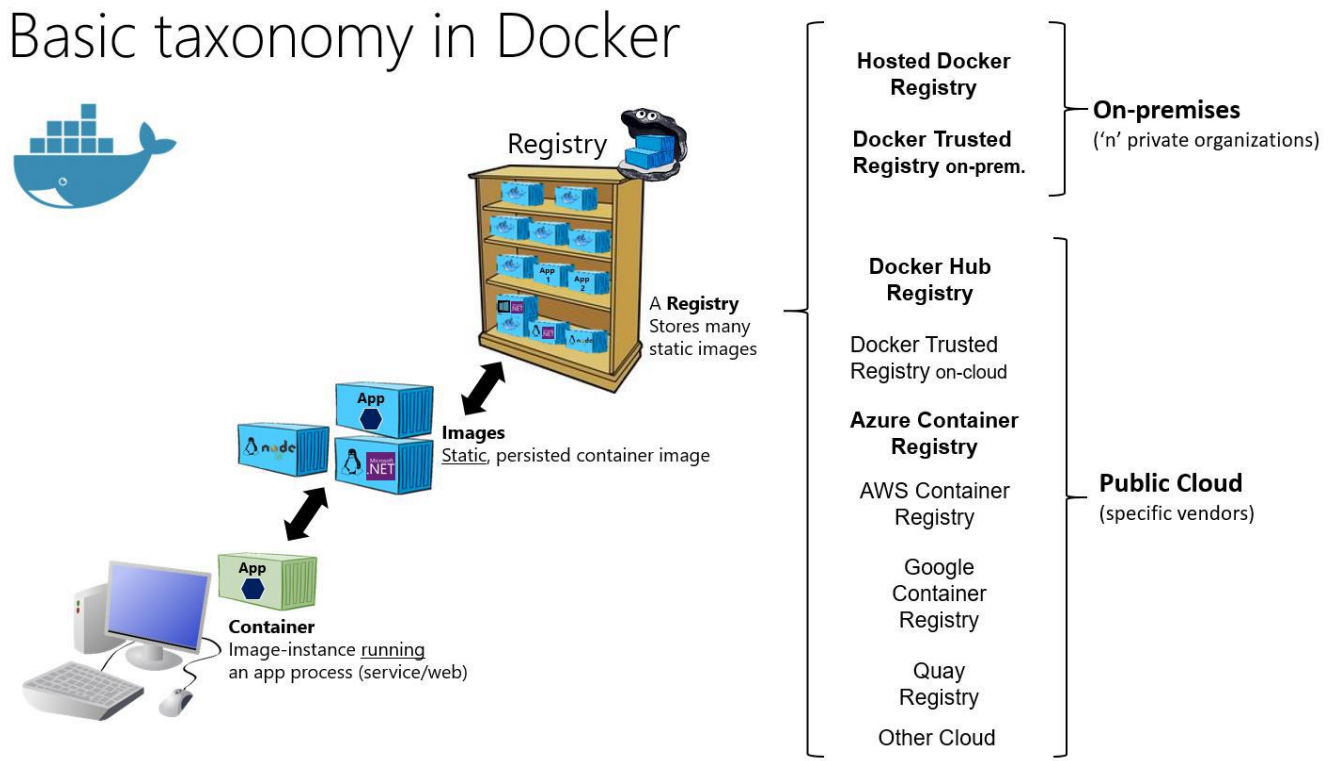
Docker containers, images, and registries

Developers should store images in a registry, which acts as a library of images and is needed when deploying to production orchestrators.

Docker maintains a public registry via Docker Hub; other vendors provide registries for different collections of images, including Azure Container Registry.

Alternatively, enterprises can have a private registry on-premises for their own Docker images.

Docker containers, images, and registries



Docker containers, images, and registries

Private image registries, either hosted on-premises or in the cloud, are recommended when:

- Your images must not be shared publicly due to confidentiality.
- You want to have minimum network latency between your images and your chosen deployment environment.

Development environment for Docker apps

Visual Studio (for Windows)

Visual Studio for Mac.

Visual Studio Code and Docker CLI

Hyper V Support

Open a PowerShell console as Administrator.

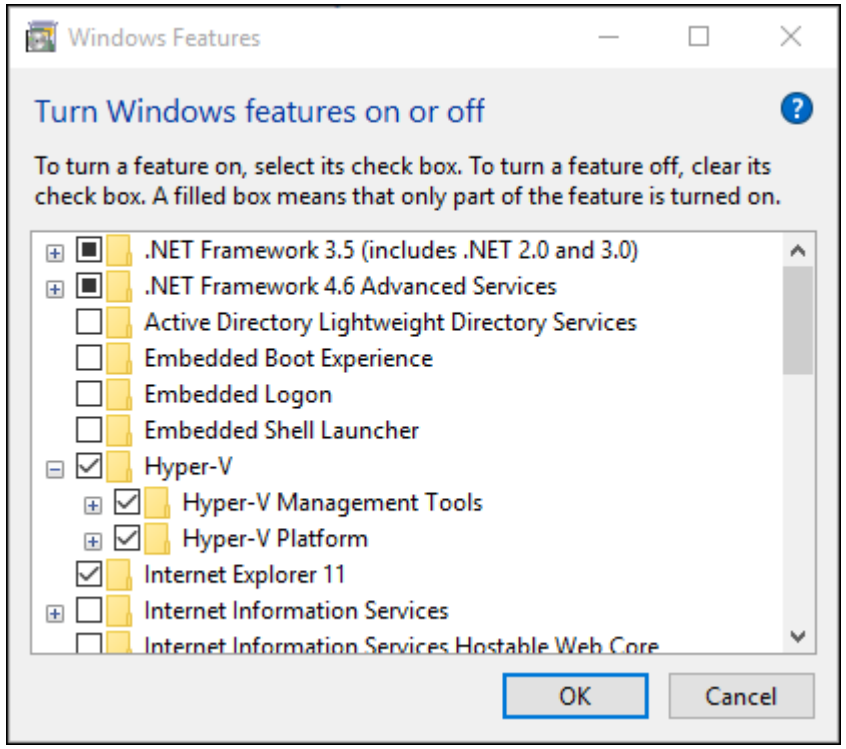
```
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V –All
```

Open up a PowerShell or CMD session as Administrator.

```
DISM /Online /Enable-Feature /All /FeatureName:Microsoft-Hyper-V
```

Hyper V Support

1. Right click on the Windows button and select ‘Apps and Features’.
2. Select **Programs and Features** on the right under related settings.
3. Select **Turn Windows Features on or off**.
4. Select **Hyper-V** and click **OK**.



How to organize the application?

The application development life cycle starts at your computer, as a developer, where you code the application using your preferred language and test it locally.

With this workflow, no matter which language, framework, and platform you choose, you're always developing and testing Docker containers, but doing so locally.

Each container (an instance of a Docker image) includes the following components:

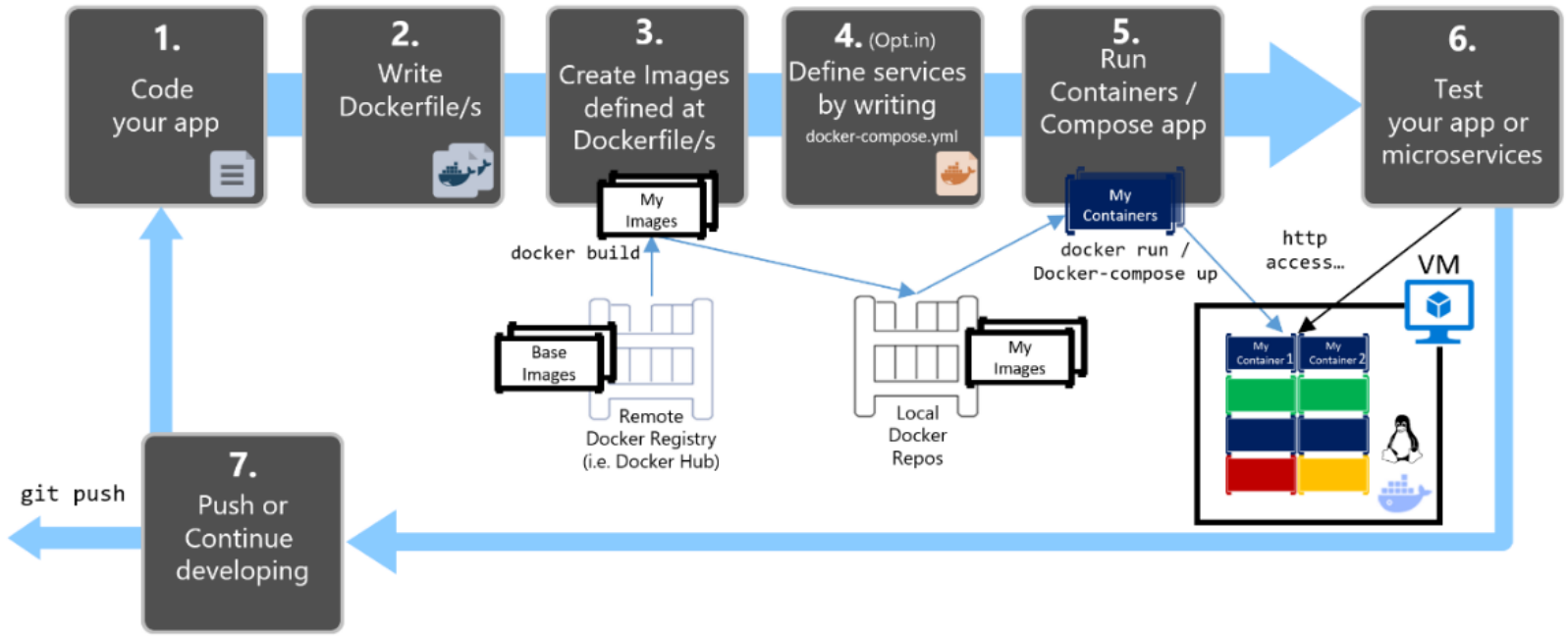
- An operating system selection, for example, a Linux distribution, Windows Nano Server, or Windows Server Core.
- Files added during development, for example, source code and application binaries.
- Configuration information, such as environment settings and dependencies.

Workflow for developing Docker container-based applications

Inner-loop development workflow

Just focuses on the development work done on the developer's computer

Inner-Loop development workflow for Docker apps



Docker Commands

Find all dangling images

docker images -f dangling=true

Purging All Unused or Dangling Images, Containers, Volumes, and Networks

docker system prune

To Remove any stopped containers and all unused images (not just dangling images:

docker system prune –a

List Docker Images

docker images –a

Remove Docker Images

docker rmi image imageidfetchedfromlistcommand

Docker Commands

Remove All Docker Images

CMD - docker rmi $(docker images -a -q)

PS - $images = docker images -a -q foreach ($image in $images) { docker image rm $image -f }

List Containers

docker ps –a

Stop all containers

docker stop $(docker ps -a -q)

Remove Container

docker rm ID_or_Name

Run and remove container on exit

docker run --rm image_name

Remove all exited containers

docker rm $(docker ps -a -f status=exited -q)

Docker Commands

List Volumes

docker volume ls

Remove Volumes

docker volume rm volume_name volume_name

Find dangling volumes

docker volume ls -f dangling=true

Remove dangling volumes

docker volume prune

Remove container and volume

docker rm -v container_name

Demo

New .Net Core application on Docker

Demo

Add docker support to existing .Net core application

Demo - Run in a Windows container

docker build -t appname .

docker run -it --rm --name sampleapp appname

Run docker ps to see the running containers.

Run docker exec aspnetcore_sample ipconfig

Demo - Run in a Linux container

docker build -t appname .

docker run -it --rm -p 5000:80 --name sampleapp appname

Run docker ps to see the running containers.

Run docker exec aspnetcore_sample ipconfig

Demo –Manual

```
FROM mcr.microsoft.com/dotnet/core/sdk:2.2 AS build
WORKDIR /app
# copy csproj and restore as distinct layers
COPY *.sln .
COPY aspnetapp/*.csproj ./aspnetapp/
RUN dotnet restore
# copy everything else and build app
COPY aspnetapp/. ./aspnetapp/
WORKDIR /app/aspnetapp
RUN dotnet publish -c Release -o out
FROM mcr.microsoft.com/dotnet/core/aspnet:2.2 AS runtime
WORKDIR /app
COPY --from=build /app/aspnetapp/out ./
ENTRYPOINT ["dotnet", "aspnetapp.dll"]
```

Build and deploy manually

dotnet publish -c Release -o published

dotnet published\aspnetapp.dll

Using multi-arch image repositories

A single repo can contain platform variants, such as a Linux image and a Windows image.

This feature allows vendors like Microsoft (base image creators) to create a single repo to cover multiple platforms (that is Linux and Windows).

Image	Target
`microsoft/dotnet:2.2-aspnetcore-runtime-stretch-slim`	OS specific: .NET Core 2.2 runtime-only on Linux
`microsoft/dotnet:2.2-aspnetcore-runtime-nanoserver-1809`	OS specific: .NET Core 2.2 runtime-only on Windows Nano Server

Using multi-arch image repositories

if you specify the same image name, even with the same tag, the multi-arch images (like the aspnetcore image) will use the Linux or Windows version depending on the Docker host OS you're deploying

Image	Target
`microsoft/dotnet:2.2-aspnetcore-runtime`	Multi-arch: .NET Core 2.2 runtime-only on Linux or Windows Nano Server depending on the Docker host OS

Multi-stage builds in Dockerfile

```
FROM microsoft/dotnet:2.2-aspnetcore-runtime AS base
WORKDIR /app
EXPOSE 80

FROM microsoft/dotnet:2.2-sdk AS build
WORKDIR /src
COPY src/Services/Catalog/Catalog.API/Catalog.API.csproj …
COPY src/BuildingBlocks/HealthChecks/src/Microsoft.AspNetCore.HealthChecks …
COPY src/BuildingBlocks/HealthChecks/src/Microsoft.Extensions.HealthChecks …
COPY src/BuildingBlocks/EventBus/IntegrationEventLogEF/ …
COPY src/BuildingBlocks/EventBus/EventBus/EventBus.csproj …
COPY src/BuildingBlocks/EventBus/EventBusRabbitMQ/EventBusRabbitMQ.csproj …
COPY src/BuildingBlocks/EventBus/EventBusServiceBus/EventBusServiceBus.csproj …
COPY src/BuildingBlocks/WebHostCustomization/WebHost.Customization …
COPY src/BuildingBlocks/HealthChecks/src/Microsoft.Extensions …
COPY src/BuildingBlocks/HealthChecks/src/Microsoft.Extensions …
RUN dotnet restore src/Services/Catalog/Catalog.API/Catalog.API.csproj
COPY . .
WORKDIR /src/src/Services/Catalog/Catalog.API
RUN dotnet build Catalog.API.csproj -c Release -0 /app

FROM build AS publish
RUN dotnet publish Catalog.API.csproj -c Release -0 /app

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

Creating your base image from scratch

Demo

Working with docker-compose.yml in Visual Studio 2017

Demo