**HRM CODE TUTORIAL COLLECTION**

**Collection By:** R&D Team

Contents

[Docker vs IIS 3](#_Toc65072056)

# Docker overview

<https://www.clariontech.com/blog/docker-for-windows-an-overview-to-dockerize-a-.net-core-app>

Why do we need Docker, Container and more?

Imagine you have to develop an app, but don’t know who will use it and on which platforms it would run. There are various apps that can run on different OSs. Compliling the code for the most popular OS is one of the ways to solve the issue. However, there is a big maintenance problem in terms of making changes in the app – you have to recompile the code repeatedly for all platforms

Creating an application within the container comes as a solution. It is an ideal way to wrap an app into an isolated package. The app will not be affected in its container by any processes or apps that exists outside of the container. The container holds everything that the app depends on to function successfully. Wherever, the container moves, the needs of the app will always be sastisfied, as it is packaged with everything that is required to run including a library, runtimes, dependencies, and so on.

Docker is the standard to build and run containerized applications. It offers a container environment to host any app of your choise. Docker containers are lightweight and allow you to move apps as well as services seamlessly between your host servers. It also enhances performance as well as lowers cost while taking less time to start and offering appropriate resource management

In addition, you no longer require pre-allocating RAM to each Docker container. The main purpose of using Docker container is to deploy apps to the production environment. Docker comes in two editions: Docker Enterprise Edition (EE) and Docker Community Edition (EC). CE is free to use and an ideal choice for small teams and development environment. EE is a subscription-based edition and suitable for running mission-critical applications. Furthermore, Docker software comes with two separate programs: Docker engine or Docker Daemon and Docker Client

# Docker vs IIS

ASP.NET Apps là các Commandline Programs. Kestrel là một built-in Library triển khai một Web Server. Chúng ta có thể run ASP.NET Core App trên Windows và Linux một cách độc lậ và access thông qua Browser. Việc cần làm là build và run file exe

Khi chúng ta run ASP.NET trong một Container, chúng ta đơn giản là run file exe và expose port nó đang listening

Run một ASP.NET Core độc lập có thể nhanh hơn việc hosting trên IIS hoặc Nginx nhưng sẽ dẫn tới việc thiếu các Features mà IIS hoặc Nginx cung cấp. Ví dụ như IIS có Static File Caching và host nhiều Applications trên cùng Port

Self Contained nghĩa là tất cả các .net Runtimes DLL đã được bao gồm trong Container thay vì sử dụng chung với Development Computer

Note: Thông thường chúng ta sẽ pull một Docker Container với .Net Core được cài đặt sẵn, vì vậy chúng ta không cần self contained deployment

ASP.Net Core luôn luôn run trong Kestrel (ngoại trừ IIS in-process hosting nơi mà IISHttpServer được sử dụng)

Vì vậy đối với một Linux Docker Container, HTTP Request được đưa tới Kestrel và sau đó được dispatch tới các middleware behind

IIS là optional khi chúng ta host Apps trên Windows, tuy nhiên nó có cung cấp thêm các Feature thực sự tốt (như Application Pool)

Chúng ta có thể pull một Linux Container với Nginx hoặc Apache, và pull một Windows Container có IIS được cài đặt sẵn. Khi run ASP.NET Core trong một Windows Container, chúng ta có thể lựa chọn việc hosting với IIS hoặc không

# .NET SDK vs .NET runtime

.net core guide <https://docs.microsoft.com/en-us/dotnet/fundamentals/>

Thông qua .NET Core guide, .NET Core bao gồm các phần sau

* .NET Runtime – cung cấp một Type System, Assembly Loading, một Garbage Collector, Native Interop và các Basic Services
* Một set Framework Libraries – cung cấp các Primitive Data Types, App Composition Types và các Fundamental Utilities
* Một set SDK Tool và Language Compilers
* ‘dotnet’ App Host, được sử dụng để lunch .NET Core apps. Nó chọn Runtime và Host the Runtime, cung cấp một Assembly Loading Policy và lunch Apps

SDK là tất cả những gì chúng ta cần để phát triển một .NET Core Application một cách dễ dàng, như là CLI và Complier

Runtime là một “virtual machine” sử dụng để host/run Application và abstract tất cả sự tương tác với Base OS

SDK cần thiết trong quá trình develop Apps, Runtime cần thiết trong quá trình run App

<https://dotnet.microsoft.com/download>

Runtime được chứa trong SDK

# DockerFile

COPY bin/Release/net5.0/publish/ App/

WORKDIR /App

ENTRYPOINT ["dotnet", "NetCore.Docker.dll"]

The COPY command tells Docker to copy the specified folder on your computer to a folder in the container. In this example, the *publish* folder is copied to a folder named *App* in the container.

The WORKDIR command changes the **current directory** inside of the container to *App*.

The next command, ENTRYPOINT, tells Docker to configure the container to run as an executable. When the container starts, the ENTRYPOINT command runs. When this command ends, the container will automatically stop.

## From

FROM mcr.microsoft.com/dotnet/aspnet:5.0 AS base

WORKDIR /app

EXPOSE 80

EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build

WORKDIR /src

COPY ["SampleOne/SampleOne.csproj", "SampleOne/"]

RUN dotnet restore "SampleOne/SampleOne.csproj"

COPY . .

WORKDIR "/src/SampleOne"

RUN dotnet build "SampleOne.csproj" -c Release -o /app/build

### dotnet/sdk

Ví dụ trên sử dụng dotnet/sdk Image để build App, Image này chứa .NET SDK (chứa các Command Line Tools CLI). Image này được tối ưu cho môi trường Development, Debugging và Unit Testing. Tool được cài đặt cho quá trình Development và Compliation làm cho Image có kích thước tương đối lớn

### dotnet/aspnet

Imange này sử dụng để để có thể run App, trong Image chứa ASP.NET Core Runtime, các Libraries và được tối ưu để running Apps trong môi trường Production. Được thiết kế để có thể tăng tốc độ Deployment và App Startup, kích thước Image này tương đối nhỏ, vì vậy Network Performance từ Docker Registry tới Docker Host được tối ưu. Chỉ có các Binraries và Content cần thiết để run App được copy tới Container

# Launchsetting.json

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/environments?view=aspnetcore-5.0>

* Chỉ được sử dụng duy nhất tại Local Development Machine
* Không được deploy
* Chứa Profile Settings

## Profiles

IIS Express – Default Profile được sử dụng để lunch một App khi sử dụng Visual Studio. “commandName” key có value là “IISExpress”, vì vậy IISExpress là WebServer

Trường hợp “commandName” có value là “Project” thì Web Server được sử dụng là Kestrel

# Web Deploy vs Web Deploy Package

## Web Deploy

Nếu chúng ta có IIS tại Server Machine đang chạy và được cấu hình để nhận Web Deploy requests, nó sẽ gửi tất cả request bằng IIS files. IIS sẽ ngay lập tức running một Web Page mới

## Web Deploy Package

Nếu chúng ta có IIS tại Server Machine đang chạy và được cấu hình để nhận một Web Deploy Package nó sẽ đóng gói toàn bộ Web Page, và sẽ upload tới Server Machine. IIS sẽ ngay lập tức running một Web Page mới

## FTP

FTP sẽ upload files tới FTP Server (có thê là bất kỳ OS nào có thể handle FTP), nhưng cần nhận thức được việc này KHÔNG an toàn và toàn bộ quá trình upload có thể bị captured và compromised

## File System

File System sẽ deploy toàn bộ items được yêu cầu để lunching một Web Service, với bất kỳ method nào được hỗ trợ bởi cấu hình hiện tại của Project vào Directory tại Machine của chúng ta.

# Choosing between .NET 5 and .NET Framework for Docker Containers

<https://docs.microsoft.com/en-us/dotnet/architecture/microservices/net-core-net-framework-containers/general-guidance>

## General guidance

Sử dụng .NET 5 với Linux hoặc Windows Containers cho Containerized Docker Server Application khi

* Chúng ta cần cross-platform, ví dụ chúng ta muốn sử dụng cả Linux và Windows Containers
* Application Architecture dựa trên kiến trúc Microservices
* You need to start containers fast and want a small footprint per container to achieve better density or more containers per hardware unit in order to lower your cost

Nói một cách ngắn gọn, khi tạo một Containerized .NET Application, chúng ta cần cân nhắc việc .NET 5 là lựa chọn mặc định. Nó sẽ mang lại nhiều lợi ích và phù hợp nhất với Containers Philosophy và Style of Working

Một lợi ích nữa khi sử dụng .NET 5 là chúng ta có thể run side-by-side các phiên bản .NET cho các Application bên trong cùng Machine. Lợi ích này là quan trọng cho Servers hoặc VMs không sử dụng Containers, bởi vì Container isolate các phiên bản của .NET mà Application cần

Sử dụng .NET Framework cho Containerized Docker Server Application khi

* Application của chúng ta hiện tại đang sử dụng .NET Framework mà có phụ thuộc nhiều vào Windows (has strong dependencies on Windows)
* Chúng ta cần sử dụng các Windows APIs không hỗ trợ .NET 5
* Chúng ta cần sử dụng third-party Libraries hoặc NuGet Packages không available trên .NET 5

Sử dụng .NET Framework trên Docker có thể cải thiện Deployment Experiences bởi việc tối thiểu hóa Deployment Issues

## When to choose .NET for Docker Containers

Tính Modularity và Lightweight của .NET 5 khiến nó trở nên hoàn hảo cho Containers. Khi chúng ta deploy và start một Container, Image .NET 5 có kích thước nhỏ hơn rất nhiều so với .NET Framework. Ngược lại, khi sử dụng .NET Framework cho một Container, chúng ta phải dựa trên Windows Server Core Image, việc này khiến nó nặng hơn rất nhiều so với Windows Nano Server hoặc Linux Images được sử dụng với .NET 5

Thêm nữa, .NET 5 là cross-platform, vì vậy chúng ta có thể deploy Server Apps với Linux hoặc Windows Container Images. Tuy nhiên, nếu chúng ta sử dụng .NET Framework, chúng ta chỉ có thể deploy Image dựa trên Windows Server Core

### Developing and deploying cross platform

Một cách rõ ràng, nếu mục đích của chúng ta là có một Application (Web App hoặc Service) có thể chạy trên đa nền tảng được hỗ trợ bởi Docker (Linux hoặc Windows), lựa chọn chính xác là sử dụng .NET 5, vì .NET Framework chỉ hỗ trợ trên Windows

.NET 5 cũng hỗ trợ macOS như là một Development Platform, tuy nhiên khi chúng ta deploy Containers tới một Docker Host, Docker Host đó phải (hiện tại) phải dựa trên Linux hoặc Windows

Visual Studio cung cấp một Intergrated Development Environement (IDE) cho Windows và hỗ trợ Docker Development

### Using Containers for new (“green-field”) Projects

Containers thường được sử dụng kết hợp với Microservices Architecture, mặc dù nó cũng có thể được sử dụng để containerize Web Apps hoặc Services đi theo các kiến trúc khác. Chúng ta có thể sử dụng .NET Framework với Windows Containers, nhưng Modularity và Lightweight Nature của .NET 5 khiến nó hoàn hảo với Containers và Microservices Architecture. Khi chúng ta tạo và deploy một Container, .NET 5 Images sẽ nhẹ hơn rất nhiều so với .NET Framework

### Create and deploy Microservices on Containers

Chúng ta có thể sử dụng .NET Framework để xây dụng Microservices-based Applications (không cần Containers) bằng cách sử dụng Plain Processes, bởi vì .NET Framework đã được cài đặt và chia sẻ giữa các Processes (Processes are light and fast to start)

Tuy nhiên nếu chúng ta sử dụng Containers, vì Image của .NET Framework dựa trên Windows Server Core vì vậy sẽ làm cho kích thước của nó vô cùng lớn cho một Microservices-on-containers Approach. Tuy nhiên, việc này đang được cải thiện đối với .NET Framework, vì hiện tại, kích thước của Windows Server Core Container Images đã được giảm đi 40% (<https://devblogs.microsoft.com/dotnet/we-made-windows-server-core-container-images-40-smaller/>)

Ngược lại đối với .NET 5 là lựa chọn tốt nhất chúng ta triển khai một Microservices-oriented System dựa trên Containers, bởi vì .NET 5 lightweight. Thêm nữa, nó liên quan tới Container Images, cho cả Linux và Windows Nano Server, cả hai đều “lean and small” khiến Containers “light and fast to start”

Một Microservice có nghĩa là chúng cần phải “to be as small as possible”: to be small when spinning up, to have a small footprint, to have a small Bounded Context (<https://en.wikipedia.org/wiki/Domain-driven_design>) to represent a small area of concerns, and to be able to start and stop fast. Vì những đòi hỏi trên, chúng ta sẽ mong muốn sử dụng một “small and fast-to-instantiate” Container Image như .NET 5 Container Image

Microservice Architecture cũng cho phép chúng ta sử dụng nhiều Technologies qua các Service Boundary. Phương pháp này cho pháp việc chuyển dịch dần dần sang .NET 5 cho các Microservices mới

### Deploying high density in scalable systems

## When to choose .NET Framework for Docker Containers

Trong khi .NET 5 cung cấp các lợi ích to lớn cho những Application mới và Application Pattern, .NET Framework vẫn sẽ là một lựa chọn tốt cho nhiều kịch bản đang tồn tại

### Migrating existing applications directly to a Windows Server Container

Chúng ta có thể mong muốn sử dụng Docker Containers chỉ để đơn giản hóa quá trình Deployment, kể cả trường hợp cúng ta không tạo các Microservices. Ví dụ có thể chúng ta mong muốn cả thiện DevOps Workflow với Docker – Containers có thể cung cấp việc Isolated Test Environment tốt hơn và có thể loại bỏ các Deployment Issues gây ra bởi việc thiếu các Dependencies khi chúng ta chuyển sang Production Environment. Trong các trường hợp như vậy, thậm chí việc chúng ta sử dụng một Monolithic Application, sẽ vẫn là hiệu quả nếu sử dụng Docker và Windows Containers cho .NET Framework Application

Trong hầu hết trường hợp thuộc kịch bản này, chúng ta sẽ không cần thiết migrate Applications hiện tại sang .NET 5; chúng ta có thể sử dụng Docker Containers chứa .NET Framework. Tuy nhiên, một phương pháp được khuyến khích sử dụng là sử dụng .NET 5 như là phần mở rộng tiếp theo của Application hiện tại, như là viết các Services mới sử dụng ASP.NET Core

### Using third-party .NET libraries or NuGet packages not available for .NET 5

Third-party Libraries thông thường sử dụng .NET Standard, việc đó cho phép code có thể được chia sẽ giữa các phiên bản .NET, bao gồm cả .NET 5. Với .NET Standard 2.0 hoặc các phiên bản sau này, API Surface tương thích với các Framework khác nhau trở nên lớn hơn rất nhiều. Thậm chí với .NET Core 2.x và các Application mới hơn có thể trực tiếp references tới các .NET Framework Libraries (.NET Framework 4.6.1 supporting .NET Standard 2.0 <https://github.com/dotnet/standard/blob/master/docs/planning/netstandard-2.0/README.md#net-framework-461-supporting-net-standard-20>)

Thêm nữa, Windows Compability Pack mở rộng API Surface available cho .NET Standard 2.0 trên Windows. Pack này cho phép recompling hầu hết code sang .NET Standard 2.x với sự thay đổi rất nhỏ hoặc không thay đổi gì để chạy trên Windows.

Tuy nhiên, thậm chí với sự phát triển vượt trội đó cho .NET Standard 2.0 và .NET Core 2.1 hoặc các phiên bản mới hơn, việc các trường hợp nơi mà các NuGet Packages cụ thể cần Windows để có thể chạy có thể không hỗ trợ .NET Core hoặc các phiên bản khác. Nếu các Packages đó là cần thiết đối với Application của chúng ta, chúng ta sẽ phải sử dụng .NET Framework trên Windows Containers

### Using .NET technologies not available for .NET 5

Một vài .NET Framework Technologies không available đối với các phiên bản hiện tại của .NET (hiện tại là .NET 5.0). Một vài Technologies đó vó thể available trong các bản Realease sau này, hoặc cũng có thể không bao giờ được hỗ trợ,…

Danh sách các công nghệ không available đối với .NET 5

* ASP.NET Web Forms
* WCF Services
* Workflow-related Services

### Using a platform or API that doesn’t support .NET 5

Một vài Platforms của Microsoft và Third-party không hỗ trợ .NET 5

## Decision table: .NET Frameworks to use for Docker

<https://docs.microsoft.com/en-us/dotnet/architecture/microservices/net-core-net-framework-containers/container-framework-choice-factors>