

Smaller Docker Images for ASP.NET Core Apps

Reduction in size up to 70%



Ali Bahraminezhad

Follow

Jan 9 · 10 min read ★





Have you ever asked yourself why you should keep your Docker images as small as possible?

1. Larger images mean a **longer time to download**.
2. Larger images mean **more storage consumption**.
3. Larger images mean there are **unnecessary components** in your images.










So a large image will cause a delay in integration and deployment or more money for storage.

In this article, I will keep efficiency in mind to describe several ways of .NET Core deployment on Docker.

. . .

1) Scaffold the Project

To start deploying on Docker we need a sample project. I will use the default ASP.NET Core API project. You can create the project by the `dotnet new webapi -o MyWebApp` command in the terminal.

Name	Date modified	Type	Size
 Controllers	07/01/2020 14:40	File folder	
 obj	07/01/2020 14:40	File folder	
 Properties	07/01/2020 14:40	File folder	
 appsettings.Development	07/01/2020 14:40	JSON File	1 KB
 appsettings	07/01/2020 14:40	JSON File	1 KB
 MyWebApp.csproj	07/01/2020 14:40	Visual C# Project File	1 KB
 Program.cs	07/01/2020 14:40	Visual C# Source File	1 KB
 Startup.cs	07/01/2020 14:40	Visual C# Source File	2 KB
 WeatherForecast.cs	07/01/2020 14:40	Visual C# Source File	1 KB

The command will create the new ASP.NET Core Web API project

2) Publish the project as a framework- dependent app

You can publish the application as a self-contained application or framework-dependent. At this part of the article, we will publish the project as framework-dependent.

To publish, enter the command below in the terminal:

```
dotnet publish -c Release -o ./publish
```

It's important to make sure you are publishing your app with **the release profile**. The `-c Release` the option will tell the .NET publisher which profile it should use.

2.1) Create a docker image with ASP.NET Core Runtime

Create a `Dockerfile` with contents below:

```
1  # We only ship the published app in the image
2  # so we only use ASPNET runtime as the base-image
3  FROM mcr.microsoft.com/dotnet/core/aspnet:3.1 AS runtime
4
5  EXPOSE 80
6  EXPOSE 443
7
8  # Copy
9  WORKDIR /app
10 COPY ./publish ./
11
12 ENTRYPOINT ["dotnet", "MyWebApp.dll"]
```

Dockerfile hosted with ❤ by GitHub

[view raw](#)

Build the image:

```
docker build -t web1 -f Dockerfile .
```

Docker will build the image and will produce output as below:

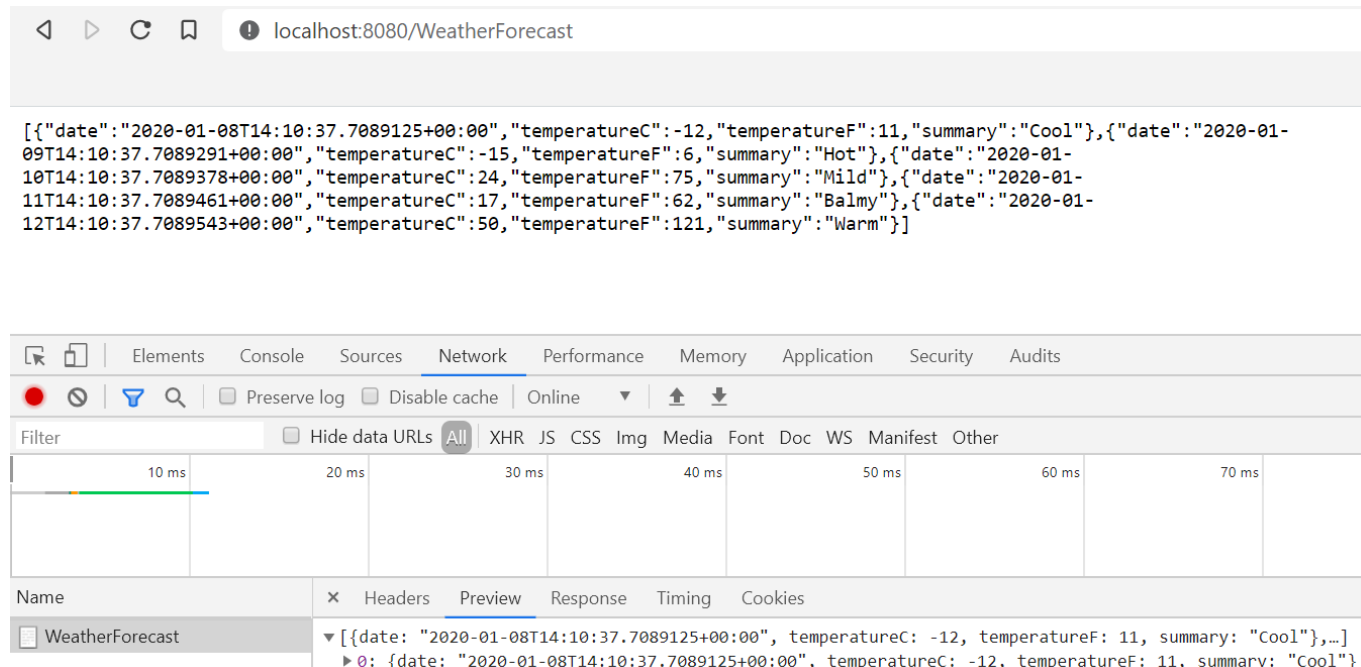
```
Sending build context to Docker daemon 1.807MB
Step 1/6 : FROM mcr.microsoft.com/dotnet/core/aspnet:3.1 AS runtime
3.1: Pulling from dotnet/core/aspnet
8ec398bc0356: Pull complete
9584d2ef7ebe: Pull complete
62b61706cd9b: Pull complete
8f13df7c0cb1: Pull complete
9c72d70b702b: Pull complete
Digest:
sha256:9f0aebb2e83a9f455e4ac123db10bc263e729e1faaf733709db04d0d6df2b7
7c
Status: Downloaded newer image for
mcr.microsoft.com/dotnet/core/aspnet:3.1
---> a843e0fbe833
Step 2/6 : EXPOSE 80
---> Running in 870640b52084
Removing intermediate container 870640b52084
---> f9a31931a780
Step 3/6 : EXPOSE 443
---> Running in 7e477e68bd5a
Removing intermediate container 7e477e68bd5a
---> 7522ce7349fa
Step 4/6 : WORKDIR /app
---> Running in 7c4d3e1490a3
```

```

Removing intermediate container 7c4d3e1490a3
---> 784d7ff4e4bd
Step 5/6 : COPY ./publish ./
---> 5a7b9b9163c0
Step 6/6 : ENTRYPOINT ["dotnet", "MyWebApp.dll"]
---> Running in ce7cdd5dd2c4
Removing intermediate container ce7cdd5dd2c4
---> bc92fbe6e1b2
Successfully built bc92fbe6e1b2
Successfully tagged web1:latest

```

📖 To make sure the web app is working fine, try to run the image by the command `docker run -p 8080:80 web1`. Then navigate to `http://localhost:8080/Weather` to see a list of JSON.



The screenshot shows a web browser window with the address bar displaying `localhost:8080/WeatherForecast`. The page content shows a JSON array of weather forecast objects. Below the browser window, the Chrome DevTools Network tab is open, showing the request to `WeatherForecast` and its response.

JSON Response:

```

[{"date": "2020-01-08T14:10:37.7089125+00:00", "temperatureC": -12, "temperatureF": 11, "summary": "Cool"}, {"date": "2020-01-09T14:10:37.7089291+00:00", "temperatureC": -15, "temperatureF": 6, "summary": "Hot"}, {"date": "2020-01-10T14:10:37.7089378+00:00", "temperatureC": 24, "temperatureF": 75, "summary": "Mild"}, {"date": "2020-01-11T14:10:37.7089461+00:00", "temperatureC": 17, "temperatureF": 62, "summary": "Balmy"}, {"date": "2020-01-12T14:10:37.7089543+00:00", "temperatureC": 50, "temperatureF": 121, "summary": "Warm"}]

```

DevTools Network Tab Details:

- Name: WeatherForecast
- Preview: `[{"date": "2020-01-08T14:10:37.7089125+00:00", "temperatureC": -12, "temperatureF": 11, "summary": "Cool"}, ...]`
- Response: `{0: {date: "2020-01-08T14:10:37.7089125+00:00", temperatureC: -12, temperatureF: 11, summary: "Cool"}}`

```

▶1: {date: "2020-01-09T14:10:37.7089291+00:00", temperatureC: -15, temperatureF: 6, summary: "Hot"}
▶2: {date: "2020-01-10T14:10:37.7089378+00:00", temperatureC: 24, temperatureF: 75, summary: "Mild"}
▶3: {date: "2020-01-11T14:10:37.7089461+00:00", temperatureC: 17, temperatureF: 62, summary: "Balmy"}
▶4: {date: "2020-01-12T14:10:37.7089543+00:00", temperatureC: 50, temperatureF: 121, summary: "Warm"}

```

Sample output

It seems the image is running fine, let's measure the size of the image. With `docker images` command, you can get the list of the images.

```

D:\MyWebApp>docker images
REPOSITORY              TAG         IMAGE ID      CREATED        SIZE
web1                    latest      bc92fbe6e1b2  16 minutes ago 208MB
mcr.microsoft.com/dotnet/core/aspnet 3.1        a843e0fbe833  10 days ago   207MB

```

207 MB for ASP.NET Core App Base Image and **1 MB** for the Web App. For a simple web application **208 MB** is big, isn't it? 🤖

Let's inspect the image, to see what made our image size 208 MB:

```
docker history web1:latest
```

```
# result
```

```

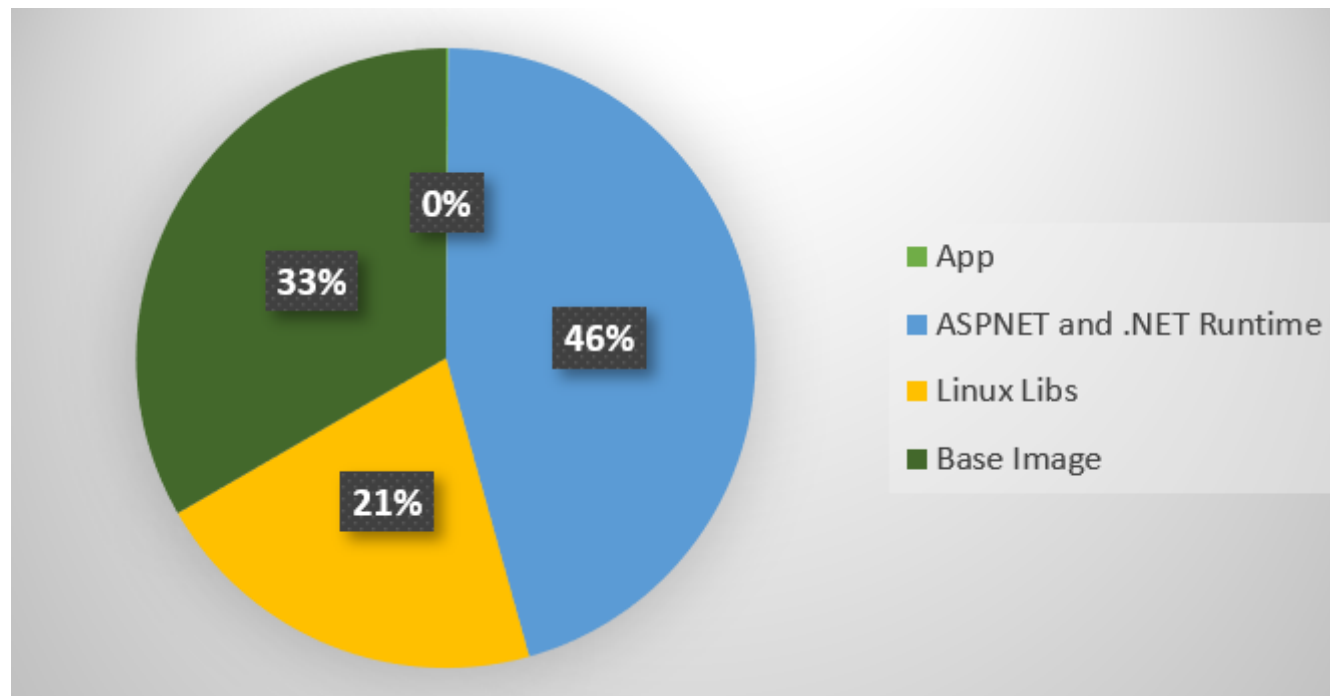
CREATED BY          SIZE
/bin/sh -c #(nop)   ENTRYPOINT ["dotnet" "MyW... 0B
/bin/sh -c #(nop)   COPY dir:36b502377fe8f29be... 289kB
/bin/sh -c #(nop)   WORKDIR /app                0B

```

```

/bin/sh -c #(nop)  EXPOSE 443                                0B
/bin/sh -c #(nop)  EXPOSE 80                                  0B
/bin/sh -c aspnetcore_version=3.1.0      && c...            17.8MB
/bin/sh -c dotnet_version=3.1.0          && curl ...         76.7MB
/bin/sh -c apt-get update                  && apt-get ins...  2.28MB
/bin/sh -c #(nop)  ENV ASPNETCORE_URLS=http:...             0B
/bin/sh -c apt-get update                  && apt-get ins...  41.3MB
/bin/sh -c #(nop)  CMD ["bash"]                                                    0B
/bin/sh -c #(nop)  ADD file:04caaf303199c81ff...           69.2MB

```



94 MB for .NET and 112MB for the base image and libs.

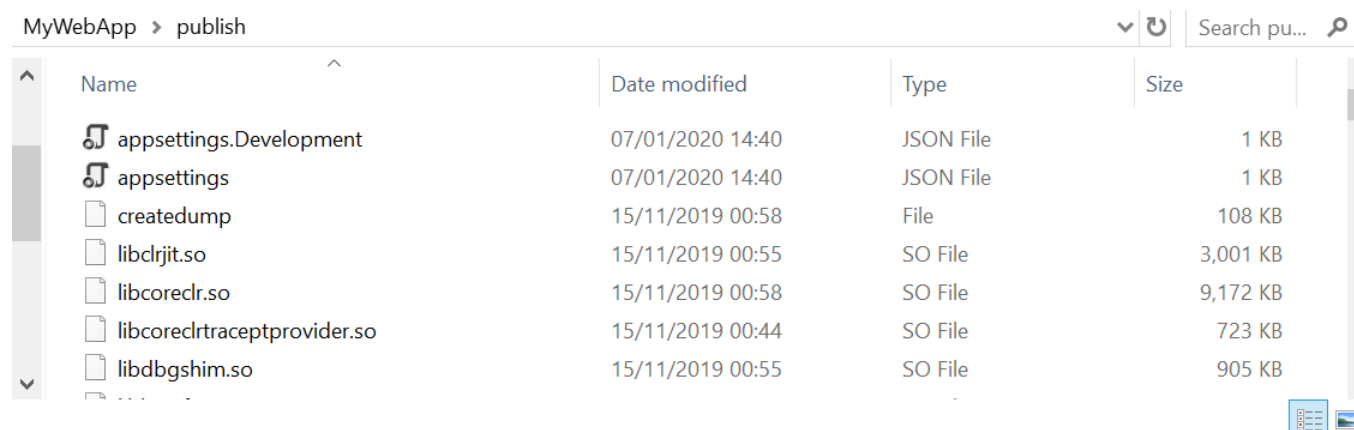
3) Publish the project as self-contained

In section 2, we published the project as framework-dependent and we saw the image size is about 208 MB.

With the self-contained deployment, you deploy your app and any required third-party dependencies along with the version of .NET Core that you used to build the app. To publish as self-contained, run the command below:

```
dotnet publish --runtime alpine-x64 -c Release --self-contained true  
-o ./publish
```

The app is going to run on top of Linux docker, so I picked `linux-x64` runtime for deployment. To see other runtimes supported by the publish command read this document.



Name	Date modified	Type	Size
appsettings.Development	07/01/2020 14:40	JSON File	1 KB
appsettings	07/01/2020 14:40	JSON File	1 KB
createdump	15/11/2019 00:58	File	108 KB
libclrjit.so	15/11/2019 00:55	SO File	3,001 KB
libcoreclr.so	15/11/2019 00:58	SO File	9,172 KB
libcoreclrtraceptprovider.so	15/11/2019 00:44	SO File	723 KB
libdbgshim.so	15/11/2019 00:55	SO File	905 KB

324 Files and 93.2 MB

It will generate **324 files (93.2 MB)** containing assemblies, etc. For running it on docker, we need a Linux base image. There are different base images such as Ubuntu, CentOS, OpenSuse, Alpine, etc.

Size, Security, Efficiency, Community are the important values to decide which one you need.

<u>ubuntu:latest</u> 188 mb Layers: 4	<u>busybox:latest</u> 2 mb Layers: 3	<u>centos:latest</u> 172 mb Layers: 3	<u>opensuse:latest</u> 82 mb Layers: 2	<u>alpine:latest</u> 5 mb Layers: 1
ADD file:c8f078961a543cd... 188 mb	MAINTAINER Jérôme Peta... 0 bytes	MAINTAINER The CentOS ... 0 bytes	MAINTAINER Flavio Castelli 0 bytes	ADD file:98d5decf83ee59e... 5 mb
RUN echo '#!/bin/sh' > /usr... 195 kb	ADD file:8cf517d90fe79547... 2 mb	ADD file:82835f82606420c... 172 mb	ADD file:30a527143b57cd1... 82 mb	
RUN sed -i 's/^#s*(deb.*u... 2 kb	CMD "/bin/sh" 0 bytes	CMD "/bin/bash" 0 bytes		
CMD "/bin/bash" 0 bytes				

Source: Docker Image Size Comparison by Brian Christner

As you can see in the image above, Alpine Linux is the lightest docker base image. It provides security, simplicity and it is resource-efficient!

Alpine Linux is an independent, non-commercial, general purpose Linux distribution designed for power users who appreciate security, simplicity and resource efficiency.

<https://alpinelinux.org/about/>

3.1) Create a docker image with Alpine base image

Create a Docker file with contents below:

```
1  FROM alpine:3.9.4
2
3  # Add some libs required by .NET runtime
4  # https://github.com/dotnet/core/blob/master/Documentation/build-and-install-rhel6-prerequisites
5  RUN apk add --no-cache \
6      openssh libunwind \
7      nghttp2-libs libidn krb5-libs libuuid lttng-ust zlib \
8      libstdc++ libintl \
9      icu
10
11  EXPOSE 80
12  EXPOSE 443
13
14  # Copy
15  WORKDIR /app
16  COPY ./publish ./
17
18  ENTRYPOINT ["/MyWebApp", "--urls", "http://0.0.0.0:80"]
```

I used alpine:3.9.4 as the base image for docker, installed some library packages in the Docker file. Now if you build the image:

```
docker build -t web2 -f Dockerfile .
```

Docker will build the image and will produce output below:

```
Sending build context to Docker daemon 296.2MB
Step 1/8 : FROM alpine:3.9.4
3.9.4: Pulling from library/alpine
e7c96db7181b: Pull complete
Digest:
sha256:7746df395af22f04212cd25a92c1d6dbc5a06a0ca9579a229ef43008d4d130
2a
Status: Downloaded newer image for alpine:3.9.4
---> 055936d39205
Step 2/8 : RUN apk add --no-cache      openssh libunwind      nghttp2-
libs libidn krb5-libs libuuid lttnng-ust zlib      libstdc++ libintl
icu
---> Running in 9471cd447603
fetch http://dl-
cdn.alpinelinux.org/alpine/v3.9/main/x86_64/APKINDEX.tar.gz
fetch http://dl-
cdn.alpinelinux.org/alpine/v3.9/community/x86_64/APKINDEX.tar.gz
(1/26) Installing libgcc (8.3.0-r0)
(2/26) Installing libstdc++ (8.3.0-r0)
```

```
(3/26) Installing icu-libs (62.1-r0)
(4/26) Installing icu (62.1-r0)
(5/26) Installing krb5-conf (1.0-r1)
(6/26) Installing libcom_err (1.44.5-r1)
(7/26) Installing keyutils-libs (1.6-r0)
(8/26) Installing libverto (0.3.0-r1)
(9/26) Installing krb5-libs (1.15.5-r0)
(10/26) Installing libidn (1.35-r0)
(11/26) Installing libintl (0.19.8.1-r4)
(12/26) Installing libunwind (1.2.1-r3)
(13/26) Installing libuuid (2.33-r0)
(14/26) Installing userspace-rcu (0.10.1-r0)
(15/26) Installing lttng-ust (2.10.1-r0)
(16/26) Installing nghttp2-libs (1.35.1-r1)
(17/26) Installing openssh-keygen (7.9_p1-r6)
(18/26) Installing ncurses-terminfo-base (6.1_p20190105-r0)
(19/26) Installing ncurses-terminfo (6.1_p20190105-r0)
(20/26) Installing ncurses-libs (6.1_p20190105-r0)
(21/26) Installing libedit (20181209.3.1-r0)
(22/26) Installing openssh-client (7.9_p1-r6)
(23/26) Installing openssh-sftp-server (7.9_p1-r6)
(24/26) Installing openssh-server-common (7.9_p1-r6)
(25/26) Installing openssh-server (7.9_p1-r6)
(26/26) Installing openssh (7.9_p1-r6)
Executing busybox-1.29.3-r10.trigger
OK: 53 MiB in 40 packages
Removing intermediate container 9471cd447603
---> 13d705c0f17d
Step 3/8 : EXPOSE 80
---> Running in 1e9629765f45
Removing intermediate container 1e9629765f45
---> 3e9b13c5f712
Step 4/8 : EXPOSE 443
---> Running in d203c44684f4
Removing intermediate container d203c44684f4
---> d28ff936ef66
Step 5/8 : WORKDIR /app
---> Running in c0e330d7f71e
Removing intermediate container c0e330d7f71e
```

```
---> 914a79994c12
Step 6/8 : COPY ./publish ./
---> f264c40e2341
Step 7/8 : RUN ["chmod", "+x", "MyWebApp"]
---> Running in a66b965fda5b
Removing intermediate container a66b965fda5b
---> 94e004ed94a7
Step 8/8 : ENTRYPOINT ["/MyWebApp", "--urls", "http://0.0.0.0:80"]
---> Running in ded6415d0e8b
Removing intermediate container ded6415d0e8b
---> 2686bfb52f0f
Successfully built 2686bfb52f0f
Successfully tagged web2:latest
```

The log is a bit longer than the previous one; because it's going to install some libraries for Alpine that are needed by .NET Core. Let's measure the image size:

```
docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
web2	latest	ced9959a36c0	39 seconds ago	147MB
alpine	3.9.4	055936d39205	8 months ago	5.53MB

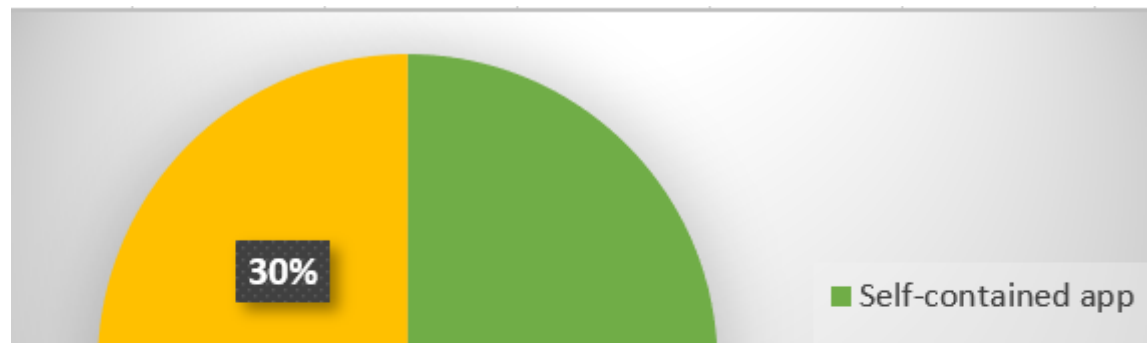
29% reduction in the size

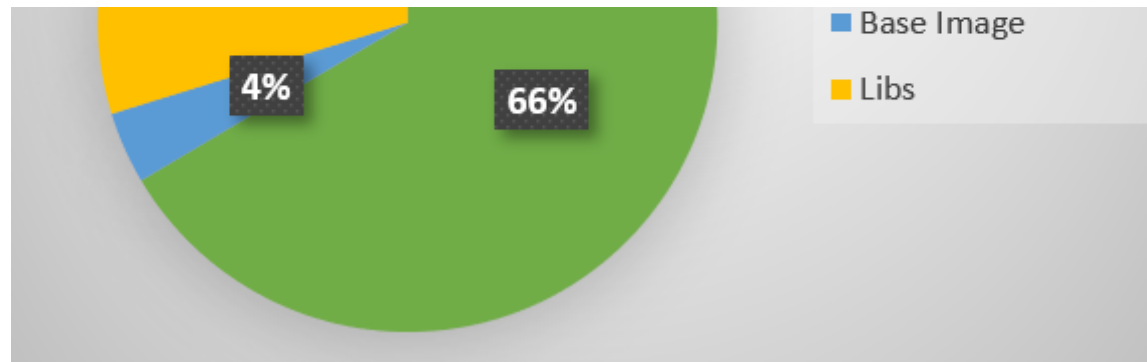
The image size is **147 MB**, **61 MB** less than the previous one. Let's inspect image layers to see what 147 MB is concluded of:

```
docker history web2:latest
```

The summarized result would be something like this:

CREATED BY	SIZE
/bin/sh -c #(nop) ENTRYPOINT ["/MyWebApp" ...	0B
/bin/sh -c #(nop) COPY dir:eec964e257409b648...	97.8MB
/bin/sh -c #(nop) WORKDIR /app	0B
/bin/sh -c #(nop) EXPOSE 443	0B
/bin/sh -c #(nop) EXPOSE 80	0B
/bin/sh -c apk add --no-cache openssh li...	43.8MB
/bin/sh -c #(nop) CMD ["/bin/sh"]	0B
/bin/sh -c #(nop) ADD file:a86aea1f3a7d68f6a...	5.53MB





49.33 MB for Base Image and Libs, 97.8 MB for App

We managed to shrink the image size by using a light base Linux image compare to the previous one. The .NET runtime is added to the self-contained app, can we make it a bit smaller?

4) Combine self-contained and IL-Linker

What's IL-Linker?

The IL Linker is a tool one can use to only to ship the minimal possible IL code and metadata that a set of programs might require to run as opposed to the full libraries.

It is used by the various Xamarin products to extract only the bits of code that are needed to run an application on Android, iOS and other platforms.

— <https://github.com/mono/linker>

IL-Linker has already included in .NET SDK, and as it says it can reduce the size of .NET Core apps to 50%.

To use il-linker, just pass the `/p:PublishTrimmed=true` flag to the publish command.

```
dotnet publish -c Release -r alpine-x64 --self-contained true  
/p:PublishTrimmed=true -o ./publish
```

It takes a bit longer, because of the IL Linker process. The result is extraordinary!

226 Files and 53.3 MB, it's 45% reduction 🙌

4–1) Dockerize it with the previous DockerFile

```
docker build -t web3 -f Dockerfile .
```

Measure the image size:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
web3	latest	96beff5758f2	7 minutes ago	105MB
alpine	3.9.4	055936d39205	8 months ago	5.53MB

```
docker history web3:latest
```

```
# result
```

```
/bin/sh -c #(nop) ENTRYPOINT ["/MyWebApp" ... 0B
/bin/sh -c #(nop) COPY dir:80b91cb154520a076... 55.9MB
/bin/sh -c #(nop) WORKDIR /app 0B
/bin/sh -c #(nop) EXPOSE 443 0B
/bin/sh -c #(nop) EXPOSE 80 0B
/bin/sh -c apk add --no-cache openssh li... 43.8MB
/bin/sh -c #(nop) CMD ["/bin/sh"] 0B
/bin/sh -c #(nop) ADD file:a86ae1f3a7d68f6a... 5.53MB
```

55.9MB for the self-contained app, 49.33 for Alpine and libs. **49.52% smaller than the default image.**

5) Remove libs and packages of Alpine

We installed libraries for Alpine that are needed by .NET Core. This documentation by Microsoft is saying these documents are only needed for

RHEL6. Since we are using Alpine, let's try to remove them and see what would happen?

```
# RUN apk add --no-cache \  
#     openssh libunwind \  
#     nghttp2-libs libidn krb5-libs libuuid lttng-ust zlib \  
#     libstdc++ libintl \  
#     icu
```

The first try:

```
Error loading shared library libstdc++.so.6: No such file or  
directory (needed by ./MyWebApp)  
Error loading shared library libgcc_s.so.1: No such file or directory  
(needed by ./MyWebApp)
```

✓ libstdc++ package is needed.

2nd try with libstdc++ lib installed:

```
Failed to load 0a)0V, error: Error loading shared library  
libintl.so.8: No such file or directory (needed by
```

```
/app/libcoreclr.so)  
Failed to bind to CoreCLR at '/app/'  
Failed to create CoreCLR, HRESULT: 0x80008088
```

✓ libintl package is needed too.

3rd try with libstdc++ and libintl installed:

```
Process terminated. Couldn't find a valid ICU package installed on  
the system. Set the configuration flag System.Globalization.Invariant  
to true if you want to run with no globalization support.
```

It seems icu package is needed. You can do two things:

1. Install the **icu** package alongside **libstdc++** and **libintl**.
2. Config your application to be invariant to cultures. (of course, if globalization is not an option for your app)

With icu package alongside those two:

```
1 FROM alpine:3.9.4
```

```
2
3 # Add some libs required by .NET runtime
4 RUN apk add --no-cache libstdc++ libintl icu
5
6 EXPOSE 80
7 EXPOSE 443
8
9 # Copy
10 WORKDIR /app
11 COPY ./publish ./
12
13 ENTRYPOINT ["./MyWebApp", "--urls", "http://0.0.0.0:80"]
```

Dockerfile hosted with ❤ by GitHub

[view raw](#)

Image size with the **required** libraries: 94.8MB. **10.2 MB** smaller than the previous one.

How about config the app to run the invariant culture mode?

```
1 FROM alpine:3.9.4
2
3 # Add some libs required by .NET runtime
4 RUN apk add --no-cache libstdc++ libintl
5
6 EXPOSE 80
7 EXPOSE 443
8
9 # Copy
```

```
10 WORKDIR /app
11 COPY ./publish ./
12
13 ENTRYPOINT ["../MyWebApp", "--urls", "http://0.0.0.0:80"]
```

Dockerfile hosted with ❤ by GitHub

[view raw](#)

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
web5	latest	d07dd3149212	About an hour ago	63MB

The image size will be **63MB** (the smallest possible size 😊). Now you need to pass an environment variable to .NET for CultureInvariant option.

Environment variable:

`DOTNET_SYSTEM_GLOBALIZATION_INVARIANT=1`

```
docker run -e DOTNET_SYSTEM_GLOBALIZATION_INVARIANT=1 -p 8080:80 web5
```

```
#result
```

```
> docker run -e DOTNET_SYSTEM_GLOBALIZATION_INVARIANT=1 -p 8080:80
web5
```

```
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://0.0.0.0:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
```

```
info: Microsoft.Hosting.Lifetime[0]  
      Hosting environment: Production  
info: Microsoft.Hosting.Lifetime[0]  
      Content root path: /app
```

Working fine and ~70% reduction in size 🙌!

. . .

Conclusion

I like Alpine Linux; because it's really light, and it makes deployment faster.

ASPNET CORE DOCKER IMAGE SIZES (MB) LOWER IS BETTER



Runtime image on Alpine and trimmed on Alpine trimmed, minimal libs on Alpine trimmed, minimal libs without ICU on Alpine

• • •






Do you want more?

The self-contained deployment will generate lots of files (+200 files). I don't consider it something really neat, on the other hand, the Docker copy command takes a little time to copy 2-3 hundred files.

The publish command supports single-file-deployment, which means it will make just a binary file for the whole application.

With the `/p:PublishSingleFile=true` flag, the publisher will only publish one single binary file.

```
dotnet publish --runtime alpine-x64 -c Release --self-contained true
-o ./publish /p:PublishSingleFile=true /p:PublishTrimmed=true
```


Name	Date modified	Type	Size
 appsettings.Development.json	07/01/2020 14:40	JSON File	1 KB
 appsettings.json	07/01/2020 14:40	JSON File	1 KB
 MyWebApp	08/01/2020 13:25	File	54,597 KB
 MyWebApp.pdb	08/01/2020 12:09	Program Debug Data...	2 KB
 web.config	08/01/2020 13:25	XML Configuration File	1 KB

One binary file + configuration files

• • •

Important note

If you are using dynamic assembly loading like

`Assymbly.Load('System.Security')` don't forget to config IL-Linker, and do not trim those assemblies.

In that case, all we need to do is to edit the **csproj** file and add:

```
<ItemGroup>
  <TrimmerRootAssembly Include="System.Security" />
</ItemGroup>
```

. . .

Update 1:

Some people mentioned me, Microsoft already provided docker images for SDK and Runtime of .NET Core based on Alpine. Why didn't I use them?

For example, this is the docker file for ASPNET Core runtime based on Alpine:

```
ARG REPO=mcr.microsoft.com/dotnet/core/runtime
FROM $REPO:3.1-alpine3.10

# Install ASP.NET Core
RUN aspnetcore_version=3.1.0 \
    && wget -O aspnetcore.tar.gz \
    https://dotnetcli.azureedge.net/dotnet/aspnetcore/Runtime/$aspnetcore_
    _version/aspnetcore-runtime-$aspnetcore_version-linux-musl-x64.tar.gz \
    && \
    aspnetcore_sha512='fa5e4ae71134a8a6db4ad6a247d3e31406673e03f0a64f7faa
    ad3d84cfb3b70d2cf69e9d9abc1f8688138907d4ddd37cd908669999d85a87892e164
    053c63847' \
    && echo "$aspnetcore_sha512  aspnetcore.tar.gz" | sha512sum -c - \
    && tar -ozxf aspnetcore.tar.gz -C /usr/share/dotnet \
    ./shared/Microsoft.AspNetCore.App \
    && rm aspnetcore.tar.gz
```

```
EXPOSE 80
EXPOSE 443

# Copy
WORKDIR /app
COPY ./publish ./

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

If I build the project framework dependant and build this docker image, the image size will **105MB**.

```
docker history alp-1
```

CREATED BY	SIZE
/bin/sh -c #(nop) ENTRYPOINT ["dotnet" "MyW...	0B
/bin/sh -c #(nop) COPY dir:36b502377fe8f29be...	289kB
/bin/sh -c #(nop) WORKDIR /app	0B
/bin/sh -c #(nop) EXPOSE 443	0B
/bin/sh -c #(nop) EXPOSE 80	0B
/bin/sh -c aspnetcore_version=3.1.0 && w...	17.8MB
/bin/sh -c dotnet_version=3.1.0 && wget ...	77.2MB
/bin/sh -c #(nop) ENV ASPNETCORE_URLS=http:...	0B
/bin/sh -c apk add --no-cache ca-certifi...	4.08MB
/bin/sh -c #(nop) CMD ["/bin/sh"]	0B
/bin/sh -c #(nop) ADD file:fe1f09249227e2da2...	5.55MB

As you can see, **95MB** for the .NET and ASPNET Core, **4.08MB** for the libs and **5.55MB** for Alpine itself. In my workaround, I only deployed part of the code and .NET is needed by the app, because IL-Linker removed unnecessary/unused parts. **94MB** or **63MB** are smaller sizes compare to **105MB**.

Of course, Microsoft Alpine-based docker image is a good choice for deployments, but it's always up to you what you choose for your project.

. . .

Good luck 😊

Thanks to Kiarash Irandoust.

[Docker](#)[Dotnet Core](#)[Aspnetcore](#)[Alpine Linux](#)[Deployment](#)

Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. Watch

Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. Explore

Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just \$5/month. Upgrade

[About](#)

[Help](#)

[Legal](#)