

Multi-Stage Dockerfiles

In this lesson, we will learn how to handle large images using multi-stage Dockerfiles.

The problem with the image we created above is that it's massive; it's **1730 MB**! This is because it contains the build tools we don't need, tools like *dotnet restore* and *dotnet publish*. Also, it contains the source code and intermediate build artifacts.

We could use the *RUN* command to try and clean the image; delete intermediate build artifacts, uninstall build tools, and delete source code, but that would be tedious. Remember that containers are like cheap, disposable machines; let's dispose of the build machine and grab a brand new one that has only the runtime installed!

Docker has a neat way to do this; use a single *Dockerfile* file with distinct sections. An image can be named simply by adding *AS* at the end of the *FROM* instruction. Consider the following simplified *Dockerfile* file:

```
FROM fat-image AS builder
...

FROM small-image
COPY --from=builder /result .
...
```



It defines two images, but only the last one will be kept as the result of the *docker build* command. The filesystem that has been created in the first image, named *builder*, is made available to the second image thanks to the `--from` argument of the *COPY* command. It states that the */result* folder from the *builder* image will be copied to the current working directory of the second image.

This technique allows you to benefit from the tools available in *fat-image* while getting an image with only the environment defined in the *small-image* it's based on. Moreover, you can have many stages in a *Dockerfile* file when

necessary.

Here is an actual example. I just improved the *Dockerfile* file shown in the preceding chapter, so that this time it uses a multi-stage build:

Dockerfile

```
FROM microsoft/dotnet:2.2-sdk AS builder
WORKDIR /app

COPY *.csproj .
RUN dotnet restore

COPY . .
RUN dotnet publish --output /out/ --configuration Release

FROM microsoft/dotnet:2.2-aspnetcore-runtime-alpine
WORKDIR /app
COPY --from=builder /out .
EXPOSE 80
ENTRYPOINT ["dotnet", "aspnet-core.dll"]
```

In the first part, I use the full and fat image that contains the whole SDK in order to build my application to an */out* directory. Then this image is trashed. The contents of its */out* directory are copied to the */app* directory of a second image. The second image is based on a runtime image, much smaller than the SDK image. In order to make it even lighter, I used an alpine-based image.

When I build an image from that multi-stage definition, I get an image that weights only **161 MB**. That's a 91% improvement over the image size!

You want to produce small images for the reasons we saw in the [Size Matters](#) lesson, so if you plan to generate artifacts inside Docker, make sure to use multi-stage *Dockerfile* files.

In the next chapter, we will look at sample Docker files for different technologies.

