From Zero to Docker

Training | 2019.05.16 | Mário Dagot, Jorge Dias

Docker is an open platform for developing, shipping, and running applications. Through the course of this training we will guide you to the most common feature and use cases of docker. Take this as an introduction and an opportunity to dive into the docker world.

AGENDA

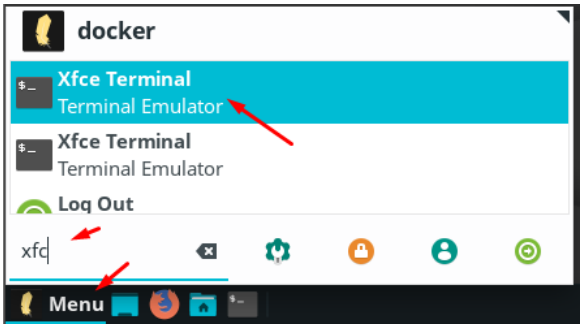
- 01 - Install Vim and Terminator and VSCode
- 02 - Install Docker CE for Ubuntu
- 03 - Hello from Busybox
- 04 - Webapp with Docker
- 05a - Webapp with Docker - My first Dockerfile – Nginx
- 05b.1 - Webapp with Docker - My first Dockerfile - Dotnet Core
- **05b.2 - Webapp with Docker - My first Dockerfile MultiStage - Dotnet Core**
- 06 - Save and Restore and Push to Docker Hub
- 07a - Webapp with database integration - My first network – SpringBoot
- 07b - Webapp with database integration - My first docker-compose – SpringBoot

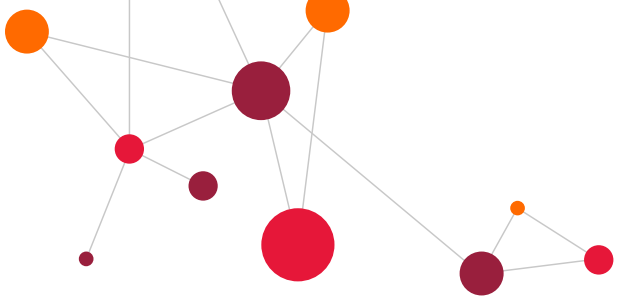
05B.2 - WEBAPP WITH DOCKER - MY FIRST DOCKERFILE MULTISTAGE - DOTNET CORE

Objective

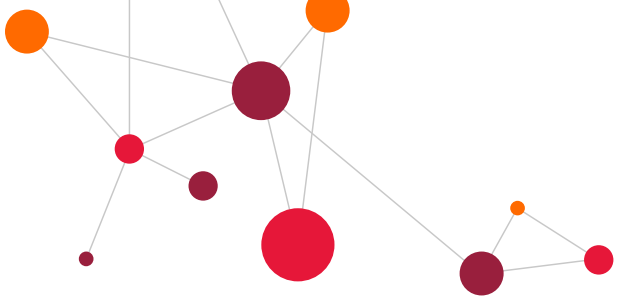
- Reuse our existing C# webapp and build a multi-stage Dockerfile
    a. HINT: SDK vs Runtime – we use the SDK to build from source. If we already have the binaries we just need the runtime.
- Learn why this is a good idea

Step by Step

Open a terminal windows	



Go to our webapp folder and create a new Dockerfile. Let's call it Dockerfile.enhanced. We can have multiple FROM named section. We can copy files from one FROM section to another. Each FROM section can use its own docker image. The last FROM section will be used as the base image of the image we are generating.	docker ~ cd ~/workspace/myapp/ docker ~ > workspace > myapp sudo vi Dockerfile.enhanced [sudo] password for docker: docker ~ > workspace > myapp cat Dockerfile.enhanced FROM mcr.microsoft.com/dotnet/core/sdk:2.2 AS build WORKDIR /app # copy csproj and restore as distinct layers COPY *.csproj /app RUN dotnet restore # copy everything else and build app COPY . /app WORKDIR /app RUN dotnet publish -c Release -o publish FROM mcr.microsoft.com/dotnet/core/aspnet:2.2 AS runtime WORKDIR /app COPY --from=build /app/publish ./ ENTRYPOINT ["dotnet", "myapp.dll"]
Let's build it. Take some time to look at the output produced.	docker ~ > workspace > myapp docker build . --tag myapp-enhanced -f Dockerfile.enhanced Sending build context to Docker daemon 3.79MB Step 1/11 : FROM mcr.microsoft.com/dotnet/core/sdk:2.2 AS build ---> 61da26769572 Step 2/11 : WORKDIR /app ---> Using cache ---> 88b0779c5fc4 Step 3/11 : COPY *.csproj /app ---> 8298e738e93d Step 4/11 : RUN dotnet restore ---> Running in a75c1dac7a61 Restore completed in 18.69 sec for /app/myapp.csproj. Removing intermediate container a75c1dac7a61 ---> 9f61745aa24a



```
Step 5/11 : COPY . /app
 ---> d0fd286acd62
Step 6/11 : WORKDIR /app
 ---> Running in a1987283a1e4
Removing intermediate container a1987283a1e4
 ---> d6e25f50b35f
Step 7/11 : RUN dotnet publish -c Release -o publish
 ---> Running in 6f6c04fe9637
Microsoft (R) Build Engine version 16.0.450+ga8dc7f1d34 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

  Restore completed in 820.5 ms for /app/myapp.csproj.
  myapp -> /app/bin/Release/netcoreapp2.2/myapp.dll
  myapp -> /app/publish/
Removing intermediate container 6f6c04fe9637
 ---> 72a92bf4ed34
Step 8/11 : FROM mcr.microsoft.com/dotnet/core/aspnet:2.2 AS runtime
2.2: Pulling from dotnet/core/aspnet
743f2d6c1f65: Already exists
393d3bd273fb: Pull complete
c9d85f416025: Pull complete
cbb29aea03dd: Pull complete
Digest: sha256:e1a94fd298a9e9bc05bb9fa4cb1af0953006095973484c51ad3fac0bce2b84bc
Status: Downloaded newer image for mcr.microsoft.com/dotnet/core/aspnet:2.2
 ---> ce06b36fcba4
Step 9/11 : WORKDIR /app
 ---> Running in 4998ccda1ef3
Removing intermediate container 4998ccda1ef3
 ---> 23408a0d79b0
Step 10/11 : COPY --from=build /app/publish ./
 ---> bc3d6b5f67df
Step 11/11 : ENTRYPOINT ["dotnet", "myapp.dll"]
 ---> Running in af274aae0021
Removing intermediate container af274aae0021
 ---> 193765e77fee
Successfully built 193765e77fee
Successfully tagged myapp-enhanced:latest
```

Let's compare the two images. The size is significant less now.	docker ~ > workspace > myapp docker images | grep myapp myapp-enhanced latest 193765e77fee About a minute ago 262MB myapp latest 0b8583945316 10 minutes ago 1.78GB
Does it behaves the same way? Go ahead and try. Did you get a "port is already allocated" error? Try listing the running containers and stop/remove it. Alternatively you could also use a new port here.	docker ~ > workspace > myapp docker run --rm -d -p 8080:80 myapp-enhanced d10a85ccb3a6c62ab54f34f1cee66b01f7aa5487ce33b6f3d82eb46f13e96709 docker ~ > workspace > myapp docker ps CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES d10a85ccb3a6 myapp-enhanced "dotnet myapp.dll" 4 seconds ago Up 3 seconds 0.0.0.0:8080->80/tcp hungry_heisenberg docker ~ > workspace > myapp docker logs hungry_heisenberg warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35] No XML encryptor configured. Key {f237a20e-d455-40d6-91ea-e71d323bb0e8} may be persisted to storage in unencrypted form. Hosting environment: Production Content root path: /app Now listening on: http://[::]:80 Application started. Press Ctrl+C to shut down. docker ~ > workspace > myapp curl http://localhost:8080/api/values ["value1","value2"]

Lessons learned

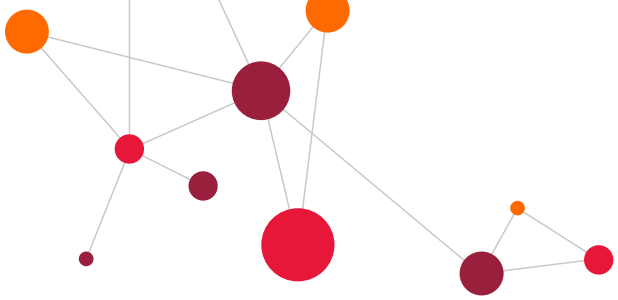
Docker is cool. But that we already knew. ☺

We should try to reduce as much as possible our docker images. Some options:

- Use a different flavors of the base image - many images are based on Alpine which is significant smaller in size than Ubuntu, for example
- Use multistage docker files and try to build the code using the SDK but only ship the image with the runtime
- Try to reduce the number of writes on the Docker file – for example, each RUN command will produce a different layer adding up to the final size. If we call run and pass along several commands it counts as only one layer.
    - For example:

Standard way	Optimal way (but less readable if we add many commands)
RUN apt-get install vim -y RUN apt-get install terminator -y	RUN apt-get install vim -y && apt-get install terminator -y

Some reasons for us to want to reduce the image size:



1. Faster builds
2. Reduced image size – faster to share
3. Reduced attack surface – the less stuff our image has the less is the probability of having security vulnerabilities that can be exploited

Revision History

Version	Date	Author	Description
1.0	2019.05.01	Mário Dagot, Jorge Dias	Initial Version