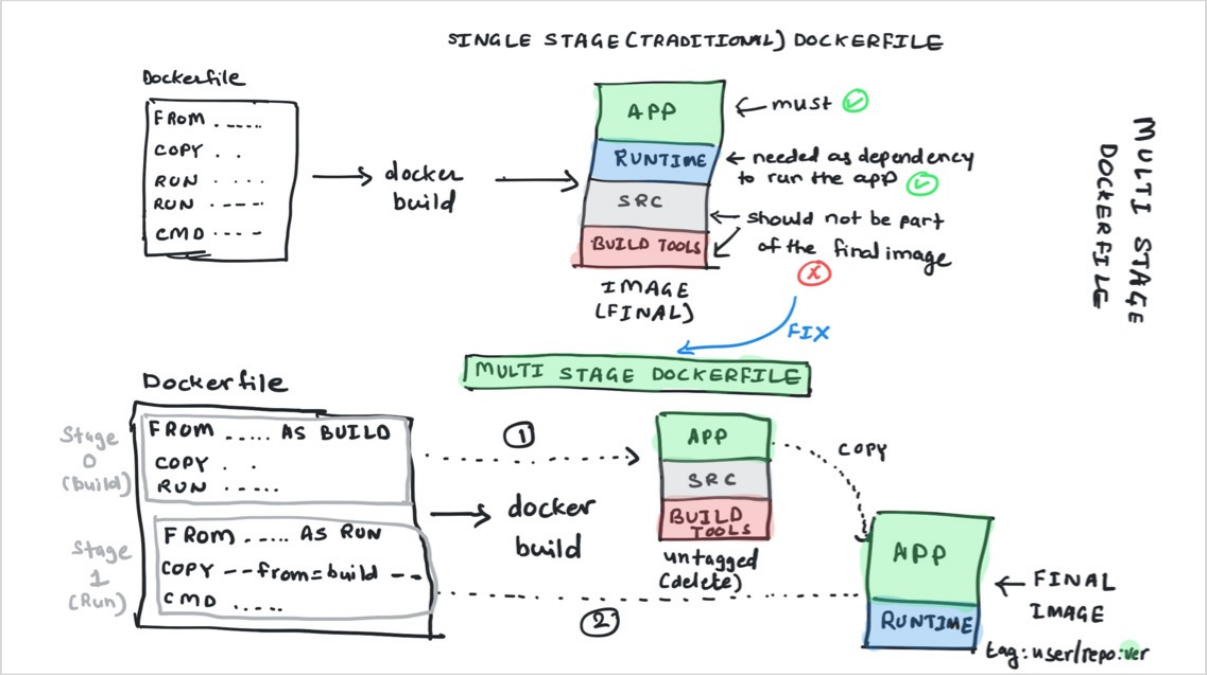
Lab 3 - Multi Stage Docker Build

You have learnt how to build an image with Dockerfile. However, the one you have built has atleast two issues,

1. The final image also contains the build tools (e.g. SDK) which is not necessary to run the application. Only a minimal runtime would have done. Packaging tooling along with application increases the size of the image.
2. Final image also contains the source code, which is definitely not desired. Only the final application and its runtime should be part of what you distribute to client/deploy to production.

How do we fix this ? Well, this is where multi stage dockerfile comes in handy and solves the problem. Following diagram depicts the difference between the two.



The above digram shows two stages, however, you could add as many as necessary. In this example, with two stage Dockerfile this is what happens

- The first stage, aliased as BUILD (also stage #0) as per above docker file uses an image with all the build tools, SDK etc. It would create a intermediate container to run the build.
- Intermediate container created for the build stage is where the source gets copied. This is where it is compiled to create the application artifacts which are deploy ready.
- Second stage, aliases here as RUN (also stage #1), launches another container. This time it uses only a minimalistic image only with runtime, with only the components needed to run the application.

- To the second stage, copy logic is added, which would fetch the deploy ready artifacts from build stage by referent it using `--from` option to COPY instruction. This ensures only the application binaries and not the source code is added to the final image.
- Second stage being the final stage in this file, an image is created by packaging the run time and application alone, and is then tagged using the option that you provide with `docker build` command.

Nano Project 301 : Refactor ASP .NET Core App with Multi Stage Dockerfile

Study multi stage Dockerfile provided here dotnetcore-samples/Dockerfile at master · initcron/dotnetcore-samples · GitHub and then refactor the Dockerfile that you have created for aspnetapp to create two stages BUILD and RUN on the similar lines. Do not look at the solution

Reference: GitHub - initcron/sysfoo: Sample java webapp with maven which prints system info

Solution

```
FROM mcr.microsoft.com/dotnet/core/sdk:3.1 AS BUILD
WORKDIR /source
COPY . .
RUN dotnet restore
WORKDIR /source/aspnetapp
RUN dotnet publish -c release -o /app

FROM mcr.microsoft.com/dotnet/core/aspnet:3.1 AS RUN
WORKDIR /app
COPY --from=build /app .
EXPOSE 80
CMD dotnet aspnetapp.dll
```

Research Project :

As part of the same repository, exists another application which includes multiple projects including a test project. Its available in dotnetcore-samples/complexapp at master · initcron/dotnetcore-samples · GitHub

Your tasks are

1. Follow the instructions given in the README for this application, and build and image for the complexapp.
2. Test the image by launching it with adequate options and then validating that you are able to connect to the web app launched.
3. Analyse the Dockerfile and find out how its designed to run tests using two different approaches.

Reference: Read the description about how this Dockerfile is designed this way and why, specially the testing part, here dotnet-docker/samples/complexapp at master · dotnet/dotnet-docker · GitHub

Packaging a .NET Framework application with Docker

.NET Framework may differ from .NET Core in terms of the functionalities, purpose and architecture it supports (e.g. .NET Framework can run only on windows) . However, when it comes to packaging it, core concepts related to building a docker image for it and writing Dockerfiles, the commands used to build and publish the image remain same.

What may differ is

- Based images used with FROM instructions would change e.g. dotnet/framework/sdk instead of dotnet/core/sdk.
- The actual commands used to build the application with RUN instruction.
- The way CMD is written to launch the application with IIS. Possibly just copy over the files and keep it in the `/inetpub/wwwroot` directory.

Research Exercise: Analyse Dockerfiles for .NET Framework Apps

You have learnt enough about writing Dockerfiles and building images with it. Now its time for you to get used to examine a Dockerfile, understand it and see why and how it works.

As part of this practice activity, you are going to analyse the following this repository

dotnet-framework-docker/samples at master · microsoft/dotnet-framework-docker · GitHub

This repository contains following four different types of .NET Framework applications, with source code and Dockerfiles for each.

- .NET Core Console App
- ASP .NET legacy App that runs with IIS
- ASP .NET MVC App
- WCF App

Your task is to,

- Read the Dockerfile for aspnetapp and dotnetapp
- Understand how its written and why its written that way
- Build an image by cloning the code and running `docker build` command for each app.
- Do a test run by launching a container using the images that you build.
- Analyse each step and learn.

Some questions you may want to find answers for

- Which image is being used for each stage why ?
- How many stages are in this Dockerfile, what's the purpose of each ?
- Which instructions are used in each stage ? How and Why ?

The best way to learn is by experimenting , failing, analysing and fixing it ! its exiting ! Go ahead and try it out.

#docker/windows/labs