Docker

Docker cli

## What is docker

* Docker is an open-source project for automating the deployment of applications as portable, self-sufficient and can run on everywhere
* Docker use the host OS for its containers, makes them light and increases the boot time
  + Light in weight
  + Faster boot time

## Basic concepts

* **Image**: is an immutable file that contains the source code, libraries, dependencies… needed for an application to run. They represent an application and its virtual environment at a specific point in time.
* **Container**: An instance of a Docker image. Containers are isolated from each other and from the host system, which allows them to run consistently across different environments.
* **Layer architecture**:  
  + A Docker image is typically multi-layered. On top of the base layer, other layers are added one by one
  + Basically, a *image layer* is a change on an image
  + If we make a change to Dockerfile, docker will rebuild **only** the layer that was changed and the ones after that → layer caching
  + Whenever a container is created, a writable layer is also created. This layer is known as the container layer and it stores all the changes done to the running container.
* **Dockerfile**: A text file that contains instructions for building a Docker image. The result image can then be ran or used as base of other image
* **Volume**: a way to store and manage data in Docker containers. They are used to persist data generated by and used by Docker containers
* **Compose**: A cli tool and YAML file with metadata for defining and running multi-container applications

## Build a Docker image

* A Dockerfile is a script that contains instructions for building a image. Each instruction is represented by a keyword, which specifies the action to be taken
  + FROM: defines the base image used to start the build process
  + ENTRYPOINT: Sets the default application to be used every time a container is created with the image
  + CMD: Specifies the default command to run when the container starts
  + ENV: Sets environment variables within the container
  + ADD: Copies files from a source on the host into the container’s filesystem (can extract files, download or copy files from remote location)
  + COPY: Copies files from the build context into the container’s filesystem (faster than ADD)
  + RUN: Executes a command within the container
  + EXPOSE: Informs Docker that the container listens on the specified network ports at runtime
  + USER: sets the UID (or username) which is to run the container
  + VOLUME: Creates a mount point for externally mounted volumes or other containers
  + WORKDIR: Sets the working directory for any RUN, CMD, ENTRYPOINT, COPY, and ADD instructions that follow it
  + LABEL**:** add a label to docker image
  + MAINTAINER**:** defines a full name and email address of the image creator
* Create a docker image for [asp.net](http://asp.net) project
* FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base  
  WORKDIR /app  
  EXPOSE 80  
  EXPOSE 443  
    
  FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build  
  WORKDIR /src  
  COPY ["Lab2.csproj", "./"]  
  RUN dotnet restore "Lab2.csproj"  
  COPY . .  
  WORKDIR "/src/."  
  RUN dotnet build "Lab2.csproj" -c Release -o /app/build  
    
  FROM build AS publish  
  RUN dotnet publish "Lab2.csproj" -c Release -o /app/publish  
    
  FROM base AS final  
  WORKDIR /app  
  COPY --from=publish /app/publish .  
  ENTRYPOINT ["dotnet", "Lab2.dll"]  
  Copy
  1. Use aspnet 7 image as base and exposes 80 and 443 port
  2. Use dotnet 7 image to build app.
     1. Copy csproj file to /src in container
     2. run dotnet restore to download any needed dependencies
     3. copy all other files
     4. build the app with Release configuration in /app/build
  3. Publish the app from app/build to app/publish
  4. copy publish files to /app and set the entry point for the container
* Build with this Dockerfile
* docker build . -t Lab2:dev  
  Copy
* Generate certificate and configure local machine
* dotnet dev-certs https -ep ${HOME}/.aspnet/https/aspnetapp.pfx -p <CREDENTIAL\_PLACEHOLDER>  
  dotnet dev-certs https --trust  
  Copy
* Run the image:
* docker run --rm -it -p 5001:80 -p 5000:443 -e ASPNETCORE\_URLS="<https://+>;<http://+>" -e ASPNETCORE\_HTTPS\_PORT=5000 -e ASPNETCORE\_Kestrel\_\_Certificates\_\_Default\_\_Password="Admin123" -e ASPNETCORE\_Kestrel\_\_Certificates\_\_Default\_\_Path=/https/aspnetapp.pfx -v ${HOME}/.aspnet/https:/https/ lab2:dev  
  Copy

## Docker compose

* Docker Compose is a tool for defining and running multi-container Docker applications
* Some Docker-compose keywords:
  + services: used to define the different services that make up your application
  + networks: used to define the networks that your services will use
  + volumes: used to define the volumes that your services will use
  + build: used to specify how to build a service’s image
  + image: used to specify the image to use for a service
  + ports: used to map ports between the host and the container
  + environment: used to set environment variables for a service
* Dockerize [Asp.net](http://Asp.net) application and MSSQL Server
* services:  
   lab2.api:  
   image: lab2  
   build:  
   context: .  
   dockerfile: Lab2/Dockerfile  
   ports:  
   - "5001:80"  
   - "5000:443"  
   environment:  
   - ASPNETCORE\_ENVIRONMENT=Development  
   - ASPNETCORE\_URLS=https://+:443;<http://+:80>  
   - ASPNETCORE\_Kestrel\_\_Certificates\_\_Default\_\_Password=Admin123  
   - ASPNETCORE\_Kestrel\_\_Certificates\_\_Default\_\_Path=/https/lab2.pfx  
   - ConnectionStrings\_\_DefaultConnection=Server=lab2.db;Database=CRM3DB;User=sa;Password=Admin123;TrustServerCertificate=True  
   volumes:  
   - ~/.aspnet/https:/https:ro  
   depends\_on:   
   - lab2.db  
   lab2.db:  
   image: mcr.microsoft.com/mssql/server  
   environment:  
   - ACCEPT\_EULA=Y  
   - SA\_PASSWORD=Admin123  
   ports:  
   - "1433:1433"  
   volumes:  
   - lab2data:/var/opt/mssql  
  volumes:  
   lab2data:  
  Copy