Workshop Docker 103

In this document you find the Docker 103 workshop text.

It has the following parts:

[Workshop environment & check 2](#_Toc530751014)

[Multistage build 4](#_Toc530751015)

[Angular app build example and containers 5](#_Toc530751016)

[Bonus: Angular local build examples 8](#_Toc530751017)

[Windows containers 10](#_Toc530751018)

[Docker containers examples 11](#_Toc530751019)

[Some Docker commands - revisited 12](#_Toc530751020)

For this workshop you need:

* PC with Windows 10 professional
* Chrome browser
* (Visual Studio Code)
* (git)

# Workshop environment & check

In this part of the workshop you will create your workshop environment.

We will install:

* Visual Studio Code
* Docker Desktop

## Install Visual Code

You can install Visual Studio Code here: <https://code.visualstudio.com/>

## Install Docker Desktop

Docker information you find here: <https://www.docker.com/products/docker-desktop>

We will install the Docker CE (Community Edition) stable version.

You can install Docker from the Docker Store.

<https://store.docker.com/editions/community/docker-ce-desktop-windows>

Let’s check the installation on Windows 10:

* Check if Docker is running – if not, run it
* Open the About Docker window
  + Which version of Docker is running?

## Create a Docker account

For this workshop we need a Docker account. With the account we push our images.

Please keep in mind that user-id is used as your repository name.

Create a Docker account on <https://hub.docker.com/>

Note: you don’t need a Docker account if you only want to pull (get) images. The Docker hub is the easiest way to share images with departments, colleagues and other people.

## Play with Docker site

People who cannot install Docker Desktop (because they don’t have Windows 10 professional or are not allowed to install or get errors) can use a site to follow the workshop.

You can reach the playground with: <https://labs.play-with-docker.com/>

Log in with your Docker account.

After login, create a new instance.

You will get Linux DockerHost. With this you can follow most of the workshop Docker 103.

Note: the experience on this site depends on your use case and on your internet connection speed.

## Is Docker working?

We will check if our Docker CE is working.

In this workshop we will use Docker CLI to go via the Docker API to Docker daemon. You can start the CLI in PowerShell (use PowerShell in the administrator mode) or you can open a PowerShell window in Studio Visual Code.

To check the Docker version, you give

docker version

You will get the version of the Docker client and server.

With command:

docker info

You will get a lot of info.

We like to check if our Docker engine is working well. We can check this with a “hello-world” app. At this moment you run the command. Later we explain it in more detail. You can run a hello-world app in a container. As you will see, Docker will pull the hello-world image from the DockerHub and will run it as a container.

docker container run hello-world

If you see the output we can presume that Docker works well.

Need some help, then you can use:

docker --help

or go to <https://docs.docker.com/>

# Multistage build

In this part we will use an aspnet website as example. Please note we are not learning dotnet.

Clone the repo <https://github.com/dotnet/dotnet-docker.git>

Goto directory: <your path>\dotnet-docker\samples\aspnetapp.

We build the app based on Linux. In the directory is Dockerfile.alpine.x64. This Docker file we discussed in the presentation.

docker image build -f Dockerfile.alpine-x64 -t aspnetapp .

This will take a while because the image is rather big. Or perhaps you have to do this @home.

You can run the app with:

docker container run --rm -it -p 8040:80 aspnetapp

You can see the app with: localhost:8040.

Try something more challenging? Use your own source and run your own aspnet site with above Docker commands.

It will run much faster because the base images are already on your machine!

# Angular app build example and containers

This part of the workshop is to learn how you can use Angular and containers, not to learn Angular. Angular is the object for working with containers.

In this part we will do the following:

* Angular without multistage build
* Angular production flow – multistage build
* Angular run – local, push to Docker registry, run on another host
* Working with Angular CLI in a container + test with container

There is a bonus part if you have an Angular environment on your laptop.

## Prerequisites

Local W10 professional machine with admin rights

Container: Docker

Code editor or IDE: Visual Code

Git (to do git clone)

Browser: Chrome

Some knowledge of Angular is handy.

## Angular production workflow without multistage build

In this way of working you do not need a local version of node, npm, angular packages. You will build and run your Angular app in a container. It is a working workflow but you will not get the most out of the advantages of containers (Docker).

Clone / copy the repo: <https://github.com/Sim007/My1Angular.git> where the source is.

Build the Angular app:

docker build -t my1angular.prodbig -f my1angular.prodbig.dockerfile .

Run the container:

docker container run -d --rm --name my1angularbigdocker -p 4300:4200 my1angular.prodbig

You see the Angular app on localhost:4300

## Angular production workflow with multistage build

In this way of working you do not need a local version of node, npm, angular packages. You will build your Angular app in a container. In this container your Angular application is the dist directory. For production and other test environments we will have a container with a webserver and the application from the dist directory. You can start this process with one (Docker) build command and run the application with one (Docker) run command.

Clone / copy the repo: <https://github.com/Sim007/My1Angular.git> where the source is.

Open Visual Studio Code and go to the directory. (fastest way: type *clone .*)

Open the file: my1angular.prodflow.dockerfile. Read and understand the multistage build file.

Start the multistage build with:

docker build -t my1angular.prodflow -f my1angular.prodflow.dockerfile .

Please note this process takes a long time the first time because it will install all needed artifacts.

You can run the Angular app in the container with

docker container run -d --rm --name my1angular1 -p 4200:80 my1angular.prodflow

## Angular run – local, push to Docker registry, run on another Dockerhost

With the container we have a running Angular application. The container - the container image to be precise - we can share with colleagues or give it to the production process.

The image is available on the laptop and we have to transfer it to the registry for sharing. We will use DockerHub.

We must tag the container with repo name and we can also give a version number. For example: sim007/my1angular.prod:0.1.0

With the following command we push it to the Docker Hub (docker.io).

docker push <repo>/my1angular.prod:<tag>

On any Docker system you can run the container. For this part we will use play-with-Docker environment.

To pull the image

docker pull <repo>/my1angular.prod:<tag>

To run a container from the image

docker container run -d --rm -p –name my1angular 4200:80 <repo>/my1angular.prod:<tag>

## Working with the Angular CLI in a container

The use case is that you don’t have an Angular environment or not the correct version of npm, nodejs, @angular/cli.

Clone / copy the repo <https://github.com/Sim007/MyToHAngular.git> with the extra files.

This is a fork of the angular showcase Tour of Heroes (ToH). (<https://angular.io/tutorial>)

Build the Angular CLI image:

docker image build -t angularclitoh -f mytohangular.cli.dockerfile .

Note this can take minutes. Just be patient.

Start the container and link your repo.

docker container run -it --name angularclitoh -v ${pwd}:/tmp angularclitoh sh

With this container you have an Angular CLI in a container.

You have an Angular CLI on Alpine Linux.

In the Angular CLI you can modify the sources in Visual Studio Code.

The source directory is linked with /tmp in the CLI container.

So go to the directory and start a build:

cd tmp

npm i

ng build --watch --delete-output-path=false

We build the app and then the CLI waits for changes and it will build again.

We have to start another container for the production container.

Open a new powershell in the source directory and build the container image.

docker build -t mytohangular.dev -f mytohangular.dev.dockerfile .

Start the container with:

docker container run -d --rm -p 4200:80 -v ${pwd}/dist:/usr/share/nginx/html mytohangular.dev

The Angular CLI container is linked to the source directory and the dev container is linked to the dist directory.

# Bonus: Angular local build examples

If you have a local Angular environment you can do the following parts:

* Angular workflow without Docker containers
* Angular workflow with production container
* Angular local workflow – test with container

## Angular workflow without Docker containers

Local machine (laptop) with node, npm and @angular/CLI.

You can install Angular CLI with:

npm install -g @angular/cli

Check the version with

node -v

npm -v

ng version

Way of working: code – test -code – deliver app

Clone / copy the repo: <https://github.com/Sim007/My1Angular.git>

Open VS Code and go to directory.

Build (Deliver) app:

ng build

Serve app to test

ng serve -o

Note: ng serve uses webserver in nodejs. For production this can be another webserver. Best practice to test is to use the same stack. So test your ng app in a container with webserver implementation from production.

## Angular local workflow – with production container

We want to have a container serving our application but also see our Angular changes immediately.

In production the only thing we need is a webserver and static Angular files from the dist directory. In this case you have a build app locally and serve it in a production container.

Build (Deliver) app:

ng build --watch --delete-output-path=false

Build the dev container. You only have to build the container once.

docker build -t my1angular.prod -f my1angular.prod.dockerfile .

Run the container.

docker container run -d --rm -p –name my1angular 4200:80 my1angular.prod

Verify the app.

## Angular local workflow – test with container

We want to have a container serving our application but also see our Angular changes immediately.

In production the only thing we need is a webserver and static Angular files from the dist directory. In this case you have a build app locally and serve it in a production container.

Build (Deliver) app:

ng build --watch --delete-output-path=false

Build the dev container. You only have to build the container once.

docker build -t my1angular.dev -f my1angular.dev.dockerfile .

Run the container.

docker container run -d --rm -p 4200:80 -v ${pwd}/dist/My1Angular:/usr/share/nginx/html my1angular.dev

Verify the app.

Note: ${pwd} is the powershell notation.

# Kubernetes

This part is a high level introduction of Kubernetes on Docker Desktop.

Open Powershell in admin mode.

Check kubernetes version

kubectl version

Check your kubernetes environments

kubectl config get-contexts

Choose the kubernetes of Docker Desktop

kubectl config set-context docker-for-desktop

List the nodes

kubectl get nodes

### Run staticws on k8s

Copy the yaml in a directory

apiVersion: apps/v1

kind: Deployment

metadata:

name: staticws-deployment

labels:

app: staticws

spec:

replicas: 1

selector:

matchLabels:

app: staticws

template:

metadata:

labels:

app: staticws

spec:

containers:

- name: staticws

image: sim007/staticws:latest

ports:

- containerPort: 80

---

kind: Service

apiVersion: v1

metadata:

name: staticws-service

spec:

selector:

app: staticws

ports:

- protocol: TCP

port: 4223

targetPort: 80

type: LoadBalancer

Run with:

kubectl apply -f staticws.yaml

You have also made a service loadbalancer to see the staticws. To check the service:

kubectl get service

You can find how to show the website.

Want to run 3 instances. Edit the yaml and change the replicas to 3

kubectl apply -f staticws.yaml

See that 3 pods are running

kubectl get pods

You can see the replica sets with:

kubectl get rs

# Windows containers

In this part a short introduction of Windows containers.

You can switch to windows containers in Docker Desktop menu.

If you have done this you can check in Powershell your docker version.

Docker version

Note that your client and server are Windows.

You can also check if Docker is working by running the hello-world container.

docker container run hello-world

See how big your image is.

docker image ls

Do you remember how many bytes the Linux container image was?

Check your image.

Docker image inspect hello-world

Run an aspnet website

docker run -it --rm -p 8060:80 --name aspnetcore\_sample microsoft/dotnet-samples:aspnetapp

# Docker containers examples

In this part we show some useful examples (containers) ready to use.

## Portainer

Portainer is Docker UI. For more information go to [https://portainer.readthedocs.io/en/latest/index.html#](https://portainer.readthedocs.io/en/latest/index.html)

docker run -d -p 9000:9000 --name portainer --restart always -v /var/run/docker.sock:/var/run/docker.sock -v portainer\_data:/data portainer/portainer

## Docker security bench

Information can be found here:

<https://docs.docker.com/compliance/cis/docker_ce/>

The source can be found here:

<https://github.com/docker/docker-bench-security/blob/master/docker-compose.yml>

docker run -it --net host --pid host --userns host --cap-add audit\_control -e DOCKER\_CONTENT\_TRUST=$DOCKER\_CONTENT\_TRUST -v /var/lib:/var/lib -v /var/run/docker.sock:/var/run/docker.sock -v /usr/lib/systemd:/usr/lib/systemd -v /etc:/etc --label docker\_bench\_security docker/docker-bench-security

## Extra: mssql

You can also run a database in a container.

See: <https://docs.microsoft.com/en-us/sql/linux/quickstart-install-connect-docker>

## Extra: ZAP

ZAP is dynamic security tool. You can start it with:

docker run -u zap -p 9080:8080 -p 9090:8090 -i owasp/zap2docker-stable zap-webswing.sh

Start with <http://localhost:9080/?anonym=true&app=ZAP>

Now you can scan your website.

# Some Docker commands - revisited

You can list the images in Docker environment with:

Docker image ls

In short you can use docker images.

You can show the running containers with:

docker container ls

A shorthand is docker ps.

If you also want to list the stopped containers, use:

docker container ls -a

Please note that a container has an id and a name. If you don’t give the container a name than Docker will give it a (funny) name. Use the name or the id to do operations for a container.

## Stop and start containers

Let’s stop the container:

docker container stop <container>

See that with docker container ls the container is not running. With docker container ls -a you will see stopped containers. Check if the application is still running.

You can start the container with:

docker container start <container>

The container will start from the point where it was stopped. All changes made in the container are still present in the container.

## Container log, go in the container and see information

If the container doesn’t do what you want, you like to see the logs:

docker container logs <name>

Sometimes you want to go in the container. In a normal container workprocess this is a no-go. The container workflow is: delete the container and ask for a new container. But hey, in some cases you want to go in the container and this is how:

docker container exec -it <name> sh

The option -it says interactive. And you must specify the shell. As you see you need some knowledge of what is in the container. You will get a command prompt. And now you are in the container – isolated from the “outside world”. You have to exit from the container with a command. Most times ctrl-C or “exit” will work.

There is a Docker command to know what is in the container.

docker container inspect <name>

There is lot of info here. If you forget the port number you can find IP and port here.

## Docker maintenance commands

To clean up you can use these commands:

To delete all stopped containers use

docker container prune

If you also give option -f it will also destroy the running container.

For images

docker image prune

You want to clean up containers & images do:

docker system prune

For volume

docker volume prune