Workshop Docker 102

In this document you find the Docker 102 workshop text.

It has the following parts:

[Workshop environment & check 2](#_Toc530515580)

[Container images 4](#_Toc530515581)

[Dockerfile 5](#_Toc530515582)

[Build-Ship-Run AAA with staticws 7](#_Toc530515583)

[Volume 10](#_Toc530515584)

[Network 12](#_Toc530515585)

[Docker-compose 14](#_Toc530515586)

[Multistage build 17](#_Toc530515587)

[Angular app build example and containers 18](#_Toc530515588)

[Bonus: Angular local build examples 21](#_Toc530515589)

[Docker containers examples 23](#_Toc530515590)

[Some Docker commands - revisited 24](#_Toc530515591)

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 Code
* Docker Desktop

## Install Visual Code

You can install Visual 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 102.

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/>

# Container images

In this part you learn the docker container image command.

You list the images on your DockerHost with:

docker image ls

You will see default the following:

* Repository
* Tag
* Image-id
* Created
* Size

If you find a specific image you can give

docker image ls alpine

Note that the image-id is not completely displayed.

Pull the alpine if it is not available.

If you like to know the complete id, you have to:

docker image inspect <image-id> or <name>

In the output you find the full image-id and digests.

You can delete an image from DockerHost with:

docker image rm <image-id> or <repo>

If you like to delete the unused images you can also use:

docker image prune -f

Unused images are called dangling images.

Keep in mind that you can always (re)build your container.

# Dockerfile

Dockerfile is a text file which Docker uses to make an image. Each line is a layer in the container image. Keep in mind that a layer is only stored once. You can see the file as receipt or blueprint. Default it is named Dockerfile.

A dockerfile for a static website can be:

FROM nginx:stable-alpine

COPY index.html \*.png /usr/share/nginx/html/

With the FROM command you start with a webserver named nginx. In fact it is the OS Alpine and the server software. We call this a base image. The base image is used to put your application on top. In this case the base image is build and maintained by nginx guys (official image).

Gitrepo: <https://github.com/nginxinc/docker-nginx>

DockerHub: <https://hub.docker.com/_/nginx/>

The next command is the copy command: copy the static files to the correct library.

Best practice is to have the Dockerfile in your source directory. So the copy command is reproducible.

It is also possible to start with nothing, so-called scratch. You have to build it all by yourself.

To build an image from a Dockerfile give the command docker image build:

docker image build -t <registry>/<repo>/name:tag .

You can name your image with -t option. Don’t forget the dot (.)! It is used to specify the build directory. You can also use another name for the Dockerfile. You can specify this by starting with -f.

Clone the repo:

<https://github.com/Sim007/staticws>

Build the container with your own Dockerfile named myDockerfile.

FROM nginx:stable-alpine

COPY index.html \*.png /usr/share/nginx/html/

Build it with

docker image build -f myDockerfile -t <repo>/staticws:<tag> .

See the output. How many layers are there?

Change some static content and build it again. You can use the same image name.

Add yourself as a maintainer in the Dockerfile and build it again.

# Build-Ship-Run AAA with staticws

In this part we will do the following case: build a staticws, test it, ship it to your repo and run it. Ask your colleague to run the container. Change the source and repeat the steps. Run staticws on play-with-Docker, run it with docker-compose, run it in k8s and run it on Azure (demo).

## Build

Use the same git repo: <https://github.com/Sim007/staticws>

Change the image and text in the html.

Build the image with the provided Dockerfile

docker image build -t mystaticws -f <Dockerfile> .

Tag the image

docker tag mystaticws <repo>/mystaticws:v1.0.0

Push the image to your repo

docker push <repo>/mystaticws:v1.0.0

Run a container with:

docker container run -d -p 4200:80 <repo>/mystaticws:v1.0.0

Delete the image from your DockerHost.

docker container rm <id> -f

And run again:

docker container run -d -p 4200:80 <repo>/mystaticws:v1.0.0

Test your container. Is it ok? Give the above docker command to a colleague and ask to run your website on his/her DockerHost. Remember that a port on DockerHost is unique!

Make another change and follow the above steps to bake a new version of the image. Decide if you want to use a version tag.

Difficult? Repetitive? Automate it! This is a kind of CI/CD pipeline.

## A way to change html page – not recommended

You can change the static html on your laptop and copy the file in the container. It‘s not a recommended way of working but it is good to understand that there are files in the container and files on your W10 system.

Open VSC in the directory with the sources of staticws.

Open a PowerShell window and run the container with the static website:

docker container run --name staticws -d -p 4202:80 <repo>/staticws

Take the index.html on your laptop and make a change and copy this in the container with:

docker container cp index.html staticws:/usr/share/nginx/html/

Note it’s a look-alike with a line in the Dockerfile:

COPY index.html \*.png /usr/share/nginx/html/

## Run in Docker play

Run the same container in play-with-Docker

## Showing: running with Docker-compose

As you have seen there are things (commands and options) to remember to run containers. With Docker-compose you can start several containers with their options with only 1 command. In the repo there is an example of a Docker-compose file. In the part docker-compose you will learn more about it.

version: '3.3'

services:

webapp:

image: sim007/staticws:$WebVersion

ports:

- 4201:80

You can start a staticws container with:

docker-compose -d up

You can stop the container with:

docker-compose down

The $Webversion is defined in .env file in the directory.

## Showing: running on Azure

You can also run the same container in Azure (services) if you have an Azure account:

* Azure Web Application
* Azure Container Instance
* AKS – in Docker 103

## Showing: running in Kubernetes local

This part of the workshop is only to show that the Docker container is working in Kubernetes.

With Docker Desktop, two container orchestrations are included namely Docker Swarm and Kubernetes (k8s). We show how we can run a container in Kubernetes with CLI (Kubectl).

Start a PowerShell session with administrator rights and check if your cluster is working:

kubectl get nodes

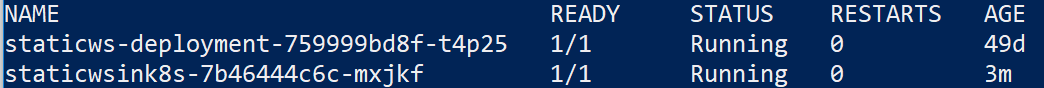
If your node is ready, we can run our staticws container with Kubernetes.

kubectl run staticwsink8s --image=<repo>/staticws --port=4203 --expose=true

Verify with:

kubectl get pods

You will get something like:



This shows that the container is running.

How to show the website is part of Docker 103.

# Volume

This part will introduce the volume in Docker.

Create a volume:

docker volume create myvolume

Check if volume is created with:

docker volume ls

You can inspect the volume with:

docker volume inspect myvolume

You can use this volume if you run a container:

docker container run -it -v myvolume:/tmp alpine

Add a file to /tmp directory

cd tmp

echo "dit is een testfile" > test.txt

ls

Open a new powershell and inspect the container

docker container inspect <container-id>

Search for a mount and you will see that the volume is mounted.

Verify that you can’t delete your volume

docker volume rm myvolume

Note in the docker “volume inspect myvolume” there is no lines that is connected to a container.

Run a new container and mount the volume

docker container run -it -v myvolume:/tmp alpine

Verify that you see the file you put in the volume and you can use it in a newly created container.

cd tmp

ls

cat test.txt

Stop de container with exit.

Go back to your other powershell with interactive alpine container.

Give exit.

Now you can delete your volume:

docker volume rm myvolume

Note that also stopped containers can have a mount to the volume.

There is also the prune command

docker volume prune

Note that Docker also uses volumes for the Docker processes.

You start Portainer with:

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

In this command example there are two mounts:

* One to directory on the DockerHost
* One to volume named portainer\_data

So if you forget your portainer password just delete the volume portainer\_data

It is also possible to mount a directory from your laptop.

In the staticws example you can mount the directory where the static files are.

Go to your directory with staticws sources.

And start the container with the mount

docker container run --name staticwsdemo -d -p 4202:80 -v ${PWD}:/usr/share/nginx/html/ nginx:stable-alpine

So open the website, change the index.html and/or picture. See that the changes are deployed directly.

Note that we just only use the official nginx image.

Above not working?

* Read the given command again!
* Did you share your drive?
* If your Windows password has changed than you have to share the drive again in Docker Desktop.

# Network

This workshop part will introduce you to docker networking.

You will create and connect and delete a docker network.

You can list the networks:

docker network ls

You will see some default networks Docker Desktop create.

Let’s make a new bridge network:

docker network create -d bridge <your network name>

With ls you can see your new network:

docker network ls

You can see the details:

docker network inspect <your network name>

You see the details of your bridge network.

You also see that there are containers in this network.

If you do docker run, your container will be attached on the default bridge network.

You run a container with alpine and connect to your network with:

docker container run -d --name myalpine1 --network <name> alpine ping 8.8.8.8

Verify that the container is connected:

docker container inspect myalpine1

Verify from the network that the container is connected:

docker network inspect <your network name>

You can also connect a running container to network bridge. This network is started by Docker engine.

docker network connect bridge myalpine1

Verify that your container is connected:

docker network inspect bridge

You can also verify this from your container

docker container inspect myalpine1

Try to delete your bridge network

docker network rm <your network>

Docker is preventing this.

Stop and delete your alpine container and then delete your network.

docker container stop myalpine1

docker container rm myalpine1

docker network rm <your network>

# Docker-compose

In this part we learn what Docker-compose is. Docker-compose is a tool that can be used to start and stop an application with one command. The tool is included in Docker Desktop.

A Docker-compose file is a yaml file.

There is a sample docker-compose file in <https://github.com/Sim007/staticws>

Open in Visual Studio Code in staticws directory and open the Docker-compose. Try to read docker compose file.

version: '3.3'

services:

staticws:

image: sim007/staticws:$WebVersion

ports:

- 4201:80

Above is the same as Docker command:

docker container run --name staticws -p 4201:80 sim007/staticws:latest

A docker-compose starts with a version.

The service is an application (composition) and port.

We add an environment variable. The variable is set in .env file

Image is the name of the image in registry. Change the repo name to your own.

And start the app with:

docker-compose up -d

Notice that docker-compose makes a network for you if not define it yourself.

You can stop the app with:

docker-compose down

The default name is Docker-compose. If you want to start another Docker-compose file you can use the -f option. Goto directory Examplesdocker-compose

docker-compose -f AnOtherName.yml up -d

You stop the app with:

docker-compose -f AnOtherName.yml down

There is another reserve name for Dockerfile: Docker-compose.override.yml. In combination with the Docker-file you can override or add lines to the Docker-file. See the Dockerfile and Docker-compose.override.yml in the editor.

You can use the docker-compose.override.yml for your environment differences.

Docker-compose config

You start the app with:

docker-compose up -d

You stop the application with:

docker-compose down

## Docker-compose example: Wordpress

In this part we see an example of the Docker-compose for Wordpress. It consist of 2 containers (Wordpress and SQL DB) and volume container. The compose file is:

version: '3.3'

services:

db:

image: mysql:5.7

volumes:

- db\_data:/var/lib/mysql

environment:

MYSQL\_ROOT\_PASSWORD: somewordpress

MYSQL\_DATABASE: wordpress

MYSQL\_USER: wordpress

MYSQL\_PASSWORD: wordpress

wordpress:

depends\_on:

- db

image: wordpress:latest

ports:

- "8000:80"

environment:

WORDPRESS\_DB\_HOST: db:3306

WORDPRESS\_DB\_USER: wordpress

WORDPRESS\_DB\_PASSWORD: wordpress

volumes:

db\_data:

For production you should not have secrets in the compose files.

## Example voting-app

Clone the repo: <https://github.com/dockersamples/example-voting-app>

Read the readme and run app with docker-compose.

# 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 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).

Build the Angular app:

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

Run the container:

docker container run -d --rm --name my1angularbigdocker -p 4300:80 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.

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 Agular 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

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.

# 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 that 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