Workshop Containers - day 1

In this document you find the container workshop text.

It has the following parts:

[Workshop environment & check 2](#_Toc19174439)

[Containers 5](#_Toc19174440)

[Registry 11](#_Toc19174441)

[Container images 13](#_Toc19174442)

[Portainer in a container 15](#_Toc19174443)

[Dockerfile 16](#_Toc19174444)

[Assignment 1 17](#_Toc19174445)

For this workshop you need:

* Internet connection
* PC with Windows 10 professional
* Chrome browser
* VS Code
* (git)

# Workshop environment & check

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

We will:

* create a Docker account
* install Visual Code
* install Docker Desktop

## Create a Docker account

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

Please keep in mind that the user-id is used as the first part of your repository name.

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

Note:

1) You need a Docker account to download Docker Desktop

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

## Install Visual Code

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

Open Visual code in admin mode (for Kubernetes extension and powershell commands).

For this workshop install the extensions Docker and Kubernetes.

Open Manage/extensions:

* Search for Docker (Microsoft) and install;
* Search for Kubernetes (from Microsoft) and install.

## Install Docker Desktop

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

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

You can install Docker from the DockerHub

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

This may take a while. If it is finished you will have a Docker icon on your desktop.

## Docker Desktop

Start Docker Desktop with the Docker Desktop icon or in the windows menu. You will see a Docker whale icon in the taskbar . When the containers on the whale are not moving the Docker Desktop is ready to use.

* Open Docker Desktop

With Docker Desktop you have installed the following (About Docker Desktop):

* Docker engine
* Docker-compose
* Kubernetes
* *Notary – not in this workshop*
* *Credential helper - not in this workshop*
* *Machine - not in this workshop*



*Settings/General*

You can tick the check box if you want to start Docker when you start your laptop.

Please note that you can switch this off in the windows start menu.

If you want notification of new updates tick the box.

Recommendation: use TLS

*Settings/Resources*

In Advanced tab you can give Docker Desktop more resources for better use.

In the file sharing option you must share your disk – so your containers can use the Windows filesystem.

In the Kubernetes option tick the box enable Kubernetes

## Is Docker working?

We will check if our Docker engine CE on your DockerHost (laptop) is working.

In this workshop we will use the 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 running in admin mode.

To check the Docker version, you give

docker version

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

With the command:

docker info

you will get a lot of Docker 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 it will run it as 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/>

## Extra: 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.

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

# Containers

In this workshop section we will learn how to work with a Docker container. As example we will have:

* Alpine container
* Aspnet container

## Alpine container

In this part we learn some “docker container” commands to work with containers.

Open a VS Code Powershell window or PowerShell.

For showing the container basic operation we will use a small Linux container called alpine. We will run command ‘ls’ in the container. To run a container to execute ls command in the container from a container image named alpine, give:

docker container run alpine ls

You will see the directory listing from the container. After the ls command is executed the container will stop.

If you want to see all running containers you can list them:

docker container ls

You will not see the alpine container running, because it has done its job. With option -a you also see the stopped containers

docker container ls -a

You will see that the container gets an id and (generated) name. Please note that the id and name are unique in your DockerHost. You need the id or name to do a container operation. Note there is also an image name.

You can start a stopped container with:

docker container start <container name> OR <container-id>

You can delete a stopped container with:

docker container rm <container name> OR <container-id>

Here is an example with command ping 8.8.8.8. This container will run forever until you stop it.

You can also name your alpine container:

docker container run --name myalpine alpine ping 8.8.8.8

Open another PowerShell window.

You can list the properties of a specific container with:

docker container ls -f name=myalpine

You can format the output with –format. Example:

docker container ls --format "table {{.ID}}\t {{.Names}}\t {{.Size}}\t {{.Image}}"

You can also pipe the output with command

docker container ls | findstr <string>

You can log the output with:

docker container logs myalpine

You can inspect the container with

docker container inspect myalpine

You can stop the container with

docker container stop myalpine

You can delete a stopped container with:

docker container rm myalpine

You can delete a running container with -f option (force)

docker container rm myalpine -f

You can delete all stopped containers with:

docker container prune -f

You can delete all running containers with (use with care):

docker container rm -f $(docker container ls -aq)

## ASPNET container

In this part you will learn, besides working with containers, that you can run many containers on one DockerHost. We will use an aspnet container as example.

In the container we have a webserver and an aspnet website.

We like to start the website in a browser. Default a container is not connected to the outside world. So we have to link to an available port on DockerHost.

We can run the container with:

docker run -it --rm -p 8000:80 --name myaspsite sim007/aspnetappdemo

In a browser you can see the website with <http://localhost:8000>.

We added the following in the command docker run:

-it interactive so you stop the website in the command line;

--rm After you stop the container it will be deleted;

-p You have to explicitly open a port to access the container. In this case the app in the container listens on port 80. On the server we will open 8000 for this. So port 80 from the container links to port 8000 on the DockerHost (localhost:8000)

Stop the website with ctrl-c and you will not find the container (running or stopped) with the docker container ls command.

As said we can start several containers on the same Docker Host.

Start in a new CLI:

docker run -it --rm -p 8001:80 --name myaspsite1 sim007/aspnetappdemo

Now you have two sites without changing any code and configuration. You can reach the sites with localhost:8000 and localhost:8001.

So it is easy to run for example .Net Core 2.2 and .Net Core 3.0 on the same Dockerhost.

Open a new PowerShell.

You can also stop the container with Docker CLI.

Do not use –rm, so the container is not deleted and you can start it again.

We can also use -d (detached) – so we still have our CLI

docker run -d -p 8002:80 --name myaspsite2 sim007/aspnetappdemo

Do the following:

* Stop the container – check if the site is available
* List stopped containers
* Start the site – see the site

docker container stop myaspsite2

docker container ls -a -f name=myaspsite2

docker container start myaspsite2

docker container ls -a -f name=myaspsite2

We can go into the running container with:

docker exec -it myaspsite2 sh

Now you are in the container and can give Linux commands.

Note: in the container world it is not best practice to change things in the container. You have to ask / make a new image.  
You exit the container (i.e. the shell) with ‘exit’.

Stop the container and delete the container

docker container stop myaspsite2

docker container rm myaspsite2

## Containers are immutable and isolated

In this part we learn that a container is immutable and isolated. You can do this part with an alpine container and/or PowerShell container

## Alpine container

Start an alpine container

docker container run -it alpine sh

Add a file Centric in home directory

echo “I added this file” >> centric

Use ls and cat to verify the file.

Go out of the interactive container with exit.

Case 1: start a new container

docker container run -it alpine sh

Search for file: Centric. It is not there.

Case 2: start the old container

Search the stopped container you started

docker container ls –a (find the container-id)

Start the container

docker container start <id>

docker exec -it <id> sh

You will see the Centric file.

Do you understand what is happening?

## PowerShell

Start a PowerShell 7.0 preview container based on an alpine container

docker container run -it mcr.microsoft.com/powershell:preview-alpine-3.8

Add a file Centric in the home directory

cd home

echo “I added this file” >> centric

Use dir and type to verify the file.

dir

type centric

Stop the container with exit.

exit

Case 1: start a new container in a new PowerShell window

docker container run -it mcr.microsoft.com/powershell:preview-alpine-3.8

Search for file: Centric. It is not there.

Case 2: start the old container

Search the stopped container you started

docker container ls –a (find the container-id)

Start the container

docker container start <id>

docker exec -it <id> sh

You will see the Centric file.

Do understand what is happening?

Question: what do you get with:

docker container run -it mcr.microsoft.com/powershell:preview-alpine-3.8 sh

You can also use PowerShell in a windows container image. But this container is much bigger.

We will explain later what mcr.microsoft.com is. For now it is a part of repo(name) of a container image.

## Passing env variables on runtime

You want to have the env YOURAPP=centric in the container.

docker container run --rm -it -e YOURAPP=centric alpine sh

In the shell give:

Printenv

Or

set

You see that the env YOURAPP is available in the container.

Leave the shell with exit.

# Registry

In this part we learn what a registry is and which commands there are.

Docker will default search in the DockerHub for images.

Open a CLI.

You can list your container images on the DockerHub with:

docker search <your docker-id>

Browse on <https://hub.docker.com/> to your own repo.

You can reach the hello-world image on the DockerHub on internet with:

<https://hub.docker.com/_/hello-world/>

You will see:

* Version per OS
* Tag

We have also used the following official image:

* <https://hub.docker.com/_/alpine/>

For official images the security scanning per tag is on.

See <https://hub.docker.com/r/library/alpine/tags/>

You can pull a hello-world docker image:

docker pull hello-world

If you don’t provide a tag then you will get the latest tag.

Note: the “tag latest” has no sematic value! It’s just a tag-name!

If the image is already on your DockerHost it will not download again.

Every container image has a unique digest. In fact each image layer has a unique digest.

You can delete an image from your DockerHost with:

docker image rm hello-world

You will see it deletes layers.

You can pull images without login.  
If you want to push containers you have to log in. You can do that with:

docker login

Enter docker-id and password.

You can add the URL of your own registry to login.

You have to tag your image before you can push it to your repo in the registry.

Tag and push the hello-world image to your repo with:

docker pull hello-world

docker tag <repo>/hello-world <repo>/hello-world:latest

docker push <repo>/hello-world:latest

Go to the Docker Hub and check if your image is there. You have now tagged your image as a ‘latest version’.

You can also check with:

docker search <repo: your Docker-id>

e.g. docker search centricms for repo with name centricms.

Note: if you don’t give a <registry name> the default is the DockerHost (docker.io).

You can also save your image on W10 filesystem with:

docker save -o hello-world.tar hello-world

where <image.tar> is the name of the tar-file you create (e.g. mycontainer.tar) and <image> is the name of your image (e.g. microsoft/aspnetcore).

You can load it with (make sure your current directory contains hello-world.tar):

docker load -i hello-world.tar

# Container images

In this part you learn the docker container image commands.

docker images is short for docker image ls

You list the images on your DockerHost with:

docker image ls

You will default see the following:

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

If you find a specific image you can give

docker image ls alpine

or

docker image ls | findstr "alpine"

Note that the image-id is not completely displayed.

docker image ls --no-trunc

Pull the alpine if it is not available.

If you like to know the complete id, you can also do the following:

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.

You can format the output with –format. Example:

docker image ls --format "table {{.Repository}}\t{{.ID}}\t {{.Size}}"

You can also pipe the output with Linux-like or PowerShell command

docker image ls | findstr <string>

docker image ls | Select-String <string>

You can inspect the content of a container image by:

docker image inspect mcr.microsoft.com/dotnet/core/samples:aspnetapp

# Portainer in a container

In this part we start a Docker UI - called portainer - in a container.

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

You can 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

--restart says it will also start the container if it is not there.

The -v (volume) we will discuss later in the workshop.

# Dockerfile

## Staticws

Dockerfile is a text file which Docker uses to make a container 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 a 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 we install a webserver named nginx. This 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 inc (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 at the end (.)! 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.

## Angular app

In this part we will dockerize an Angular app. You can use this when you are developing the app.

Goals of this part are:

* How to build a DockerFile
* How to build a container image
* How to run a container

We will use an example app of Angular 8 to show this.

See readme.md from <https://github.com/Sim007/vbangularindocker.git>

# Assignment 1

You can find the assignment in the workshop sheets.

Code & solution you can find on github:

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