Workshop Docker 101

In this document you find the Docker 101 workshop text.

It has the following parts:

[Workshop environment & check 2](#_Toc529304979)

[Containers 4](#_Toc529304980)

[Registry 9](#_Toc529304981)

[Container images 11](#_Toc529304982)

[Dockerfile 12](#_Toc529304983)

[Examples 13](#_Toc529304984)

[Build-Ship-Run with staticws 14](#_Toc529304985)

[Some Docker commands - revisited 17](#_Toc529304986)

For this workshop you need:

* PC with Windows 10 professional
* Chrome browser
* (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 101.

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 to 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 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/>

# Containers

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

* A plain Linux container
* An aspnet container
* PowerShell container

## Alpine container

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

Open a PowerShell.

For showing the container operation we will use a small Linux container called alpine. We will run command ‘ls’ in the container. To run a container to perform 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 run <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 an another PowerShell window.

You can list the properties of a specific container with:

docker container ls -f name=myalpine

You can log output with:

docker container logs 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 a not running container with:

docker container prune -f

## ASPNET

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.

Do the following:

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

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. You can reach the site with localhost:8000 and localhost:8001.

So it is easy to run for example .net 2.0 and .net 2.1 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 in 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.

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 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.

Stop 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>

You will see the Centric file.

Do you understand what is happening?

## PowerShell

Start a PowerShell container based on an alpine container

docker container run -it microsoft/powershell

Add a file Centric in the home directory

echo “I added this file” >> centric

Use dir and type to verify the file.

Stop the container with exit

Case 1: start a new container

Search for file: Centric. It is not there.

Case 2: start the old container

docker container run -it microsoft/powershell

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>

You will see the Centric file.

Do understand what is happening?

Question: what do you get with:

docker container run -it microsoft/powershell sh

# 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 <repo=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 also used the following official images:

* <https://hub.docker.com/_/alpine/>
* <https://hub.docker.com/r/microsoft/dotnet-samples/>

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 than 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.

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 registry.

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

docker pull hell-world

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

docker push <repo>/hello-world

Go to the DockerHub and check if your image is there.

You can also check with:

docker search <repo: your Docker-id>

You can also save your image on W10 filesystem with:

docker save -o ${pwd}/<image.tar> <image>

You can load it with:

docker save -i <image.tar> <image>

# Container images

In this part you learn the 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. So you have to version your Dockerfiles.

# 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 Docker is named Dockerfile.

A dockerfile for example for a static website has only 2 lines.

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 OS Alpine and the server software. We call this a base image. The base image is used to put your artifacts on to. In this case the base image is build and maintained by official nginx guys.

The next command is the copy command: 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.

# 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.

# Build-Ship-Run with staticws

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 and run it in k8s.

## Build

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

Change the image and text in the source html.

Build the image with the provided Dockerfile or your own Dockerfile

docker image build -t mystaticws .

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>

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!

Run it on <https://labs.play-with-docker.com/> or in the cloud.

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 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.

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.

## Run in Docker play

Run the same container in play with Docker

## Run 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

## Run 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

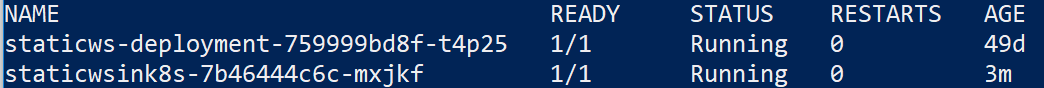
If your node is ready, we can run our staticws container with kubernetes.

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.

# 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