Workshop Docker 101

In this document you find the Docker 101 workshop text.

It has the following parts:

[Workshop environment & check 2](#_Toc529977160)

[Containers 4](#_Toc529977161)

[Registry 9](#_Toc529977162)

[Examples 10](#_Toc529977163)

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 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 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 all stopped containers 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 Core 2.0 and .Net Core 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.  
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 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> sh

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> sh

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 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.

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 hellow-world.tar):

docker load -i hello-world.tar

# 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