# Practice M3: Advanced Docker

We assume that we are working on a machine, no matter what operating system, with VirtualBox installed

Most of the steps can be executed with little or no modification if you are using **Docker Toolbox for Windows** or **Docker Toolbox for Mac**

## Part 1: Docker Machine

### Docker Machine

The installation of **docker-machine** can be done in several ways

If we use **Docker Desktop for Windows** or **Docker Desktop for Mac**, we already have it and we can skip the following steps about client and machine installation

Because the above two require (or work by default) a different (native) virtualization from the one we are using (VirtualBox), we will do a manual installation

Another option would be to use the legacy **Docker Toolbox for Windows / Mac** with also comes prepacked with all tools and uses **VirtualBox**

#### Install Docker Client (docker)

Depending on how you would like to communicate with the **Docker** **instance**, you can install **Docker** **client** on your host, or use the one that comes with the instance

Next steps are NOT mandatory, and they are provided for completeness

##### Linux Distribution

**Docker Client** installation can be done via the package system of our Linux distribution

Alternatively, we can download the universal binary from **Docker**

To do it for version 19.03.9 (it is not the latest version), we must execute:

**$ curl -L https://download.docker.com/linux/static/stable/x86\_64/** **docker-19.03.9.tgz /tmp/docker-19.03.9.tgz && \**

**tar xzvf /tmp/docker-19.03.9.tgz && \**

**sudo mv docker/docker /usr/local/bin/docker && \**

**rm -rf docker/ && \**

**rm /tmp/docker-19.03.9.tgz**

Alternatively, we can go here: <https://download.docker.com/linux/static/edge/x86_64/> and check for the latest version

##### Windows

Navigate to the following URL:

<https://download.docker.com/components/engine/windows-server/19.03/docker-19.03.5.zip>

And save the file locally

Extract it to a folder of your choice, for example **C:\BIN\docker**

Make sure that the folder is part of your system’s path

#### Install Docker Machine (docker-machine)

Installation can vary between different host operating systems

##### Linux Distribution

In case we use any kind of Linux distribution, we can install Docker Machine with:

**$ curl -L https://github.com/docker/machine/releases/download/v0.16.2/docker-machine-`uname -s`-`uname -m` >/tmp/docker-machine &&**

**chmod +x /tmp/docker-machine &&**

**sudo cp /tmp/docker-machine /usr/local/bin/docker-machine**

Alternatively, we can use a native package for our distribution of choice, if available

More info here: <https://docs.docker.com/machine/install-machine/>

##### Windows

For Windows, the procedure can be accomplished via **git bash** (<https://gitforwindows.org/>) or manually

If you decide to go this way, after having the above package installed, open a **git bash** terminal, and execute:

**$ if [[ ! -d "$HOME/bin" ]]; then mkdir -p "$HOME/bin"; fi && \**

**curl -L https://github.com/docker/machine/releases/download/v0.16.2/docker-machine-Windows-x86\_64.exe > "$HOME/bin/docker-machine.exe" && \**

**chmod +x "$HOME/bin/docker-machine.exe"**

Let us do it manually instead

Download this archive:

<https://github.com/docker/machine/releases/download/v0.16.2/docker-machine-Windows-x86_64.exe>

For this example, I will use the folder **C:\BIN\docker** as a target (you can use whatever you like)

Extract the downloaded archive to a folder of your choice

Make sure that you added this folder to the system’s path

More information about the current version, here: <https://github.com/docker/machine/releases/>

#### Create Docker Machine

Now, that we have everything installed, we can create our first **Docker Machine**

Open a terminal window and execute:

**$ docker-machine create --driver virtualbox default**

This will create a **Docker instance** named **default**

We can check the list of available **Docker** machines with:

**$ docker-machine ls**

NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS

default - virtualbox Running tcp://192.168.99.100:2376 v19.03.5

The **URL** column gives us information which address we should use to communicate with the containers running inside the instance

#### Environment Setup

If we have a **Docker Client** installed on our host system, and we want to use it to control the **Docker Machine**, we must set up the environment first

This can be done in two steps

First, we must ask for the settings (change **default** with the name of your Docker Machine instance):

**$ docker-machine env default**

Then, we must follow the instructions which will vary between different host operating systems

For example, for a Linux distribution we will be expected to execute:

**$ eval $(docker-machine env)**

And for Windows, we must execute:

**$ @FOR /f "tokens=\*" %i IN ('docker-machine env default') DO @%i**

Now, we are ready to send commands to our **Docker instance**

For example, we can execute:

**$ docker container run shekeriev/welcome-dob:2021**

Of course, we can test with other images as well

Now, if we want to reset the environment, we can execute:

**$ docker-machine env -u**

And then follow the instructions

Should we want to use the client that comes with the instance instead, we have two options

The first one, is to send commands via the **Docker Machine**

**$ docker-machine ssh default docker image ls**

You may need to put the actual command (**docker image ls**) in single quotes

And the second option is to enter the instance, and then execute commands as we did in the previous module

**$ docker-machine ssh default**

docker@default:~$ **docker image ls**

To make sure that we are working in the same environment, until the end of this practice, we will work from within the **Docker Machine**

### Networks

Enter the Docker instance

**$ docker-machine ssh default**

Let us check what networks we have defined:

docker@default:~$ **docker network ls**

We can ask for additional information for one of the existing networks. For example, for the **bridge** one

docker@default:~$ **docker network inspect bridge**

We may see the IP address range, the reference to an existing network adapter on the system, etc.

Now, we can create our own bridge network named **dob-bridge**:

docker@default:~$ **docker network create -d bridge --subnet 10.0.0.0/24 dob-bridge**

docker@default:~$ **docker network ls**

We can ask for additional information about the network we just created with:

docker@default:~$ **docker network inspect dob-bridge**

Let us create two containers and put them on our new network

We will base the containers on the **alpine** image

docker@default:~$ **docker container run -dt --name co1 --network dob-bridge alpine sleep 1d**

docker@default:~$ **docker container run -dt --name co2 --network dob-bridge alpine sleep 1d**

Now, we can enter in both containers and test their network connectivity

docker@default:~$ **docker container exec -it co1 sh**

/ # **ping -c 4 softuni.bg**

/ # **ip a**

/ # **exit**

docker@default:~$ **docker container exec -it co2 sh**

/ # **ip a**

/ # **ping -c 4 co1**

/ # **exit**

If we ask again for detailed information about the network

docker@default:~$ **docker network inspect dob-bridge**

We can see that both containers are attached to it and we can see their IP addresses

Let us stop them

docker@default:~$ **docker container stop co1 co2**

And remove the network

docker@default:~$ **docker network rm dob-bridge**

### Volumes

We can share data with containers in several ways. Let us explore them in the next exercises

#### On the Fly

First, we will create a container to which we will attach a volume (a local folder available on the host)

docker@default:~$ **docker container run -it -v /test-vol --name c1 ubuntu /bin/bash**

Now, we should have a **/test-vol** folder inside the container

Let us exit the container with **Ctrl+p** и **Ctrl+q**

Create a second container that will inherit the volume(s) from the first one

docker@default:~$ **docker container run -it --volumes-from c1 --name c2 ubuntu /bin/bash**

Again, exit the container with **Ctrl+p** и **Ctrl+q**

Attach back to the first one

docker@default:~$ **docker container attach c1**

root@8f7010fff13d:/# **echo 'Hi from C1!' >> /test-vol/file.txt**

Exit again with **Ctrl+p** и **Ctrl+q** and attach back to the second one:

docker@default:~$ **docker container attach c2**

Add some text to the file and exit (stop) the container

root@937b91cf5b51:/# **echo 'C2 is here!' >> /test-vol/file.txt**

root@937b91cf5b51:/# **exit**

Now, attach back to the first container:

docker@default:~$ **docker container attach c1**

Check the file and exit the container

root@8f7010fff13d:/# **cat /test-vol/file.txt**

root@8f7010fff13d:/# **exit**

Let us ensure that both containers are stopped

docker@default:~$ **docker container ls**

Now, we can start third container, that will inherit the volume from **c1**

docker@default:~$ **docker container run -it --volumes-from c1 --name c3 ubuntu /bin/bash**

And check the file content and add a row to the file

root@1de7a8d4b5f2:/# **cat /test-vol/file.txt**

root@1de7a8d4b5f2:/# **echo 'C3 joined the party!' >> /test-vol/file.txt**

Exit the container with **Ctrl+p** и **Ctrl+q** , and start again the **c1** container

docker@default:~$ **docker container start -i c1**

root@8f7010fff13d:/# **cat /test-vol/file.txt**

Exit again with **Ctrl+p** и **Ctrl+q**

To check where the data is being stored in a persistent manner, execute

docker@default:~$ **docker container inspect c1 | grep -i source**

We can now see the folder on our host where the data is being stored

Alternatively, we can ask for container volumes with

docker@default:~$ **docker volume ls**

Detailed information about the volume, we can get by executing (substitute the identifier with the one you see on your screen, the whole identifier not a part of it)

docker@default:~$ **docker volume inspect f2d6e112f178b918f4e204312**

Now, we can inspect the file content from the host

Change to the **root** user with

docker@default:~$ **sudo -i**

Navigate to the folder

root@default:~# **cd /mnt/sda1/var/lib/docker/volumes/f2d6e112f178b918f4e204312/\_data**

Check the file content and exit the **root** session

root@default:/mnt/sda1/var/lib/docker/volumes/f2d6e112f178b918f4e204312/\_data# **cat file.txt**

root@default:/mnt/sda1/var/lib/docker/volumes/f2d6e112f178b918f4e204312/\_data# **exit**

As a last step we can stop both running containers

docker@default:~$ **docker container stop c1 c3**

#### Attach an Existing Folder

Let us first create the folder and a simple **index.html** file in it

docker@default:~$ **mkdir /home/docker/web**

docker@default:~$ **echo '<h2>Hello from a Docker Volume</h2>' > /home/docker/web/index.html**

Now, we can start the container with the folder attached to it

docker@default:~$ **docker container run -d -p 8080:80 --name co-apache \**

**-v /home/docker/web:/var/www/html php:7.0-apache**

Open a new browser tab on your host and enter the following URL: <http://192.168.99.100:8080>

If your **Docker** instance has another IP address, then use it instead

We can create a session to the container

docker@default:~$ **docker container exec -it co-apache bash**

And browse and explore the file

root@a255401e6b24:/var/www/html# **ls -al**

**...**

root@a255401e6b24:/var/www/html# **cat index.html**

Let us change the file being inside the container

root@a255401e6b24:/var/www/html# **echo '<br />Changed inside the container' >> /index.html**

If we refresh the browse tab, we will see the difference

Exit the container session

root@a255401e6b24:/var/www/html# **exit**

Let us change the file on the **Docker** host

docker@default:~$ **echo '<br /><br />Updated on the host' >> /home/docker/web/index.html**

If we refresh the browse tab, we will see the difference

Let’s check if there is a volume for the mounted folder

docker@default:~$ **docker volume ls**

It appears that there is not

Let us stop the container

docker@default:~$ **docker container stop co-apache**

#### Dedicated Volume

In this exercise we will create the volume upfront with

docker@default:~$ **docker volume create lv-1 --label mode=prod**

Then will list the volumes with

docker@default:~$ **docker volume ls**

Explore the volume’s details

docker@default:~$ **docker volume inspect lv-1**

Filter volumes list by label

docker@default:~$ **docker volume ls -f label=mode=prod**

Format the volumes list

docker@default:~$ **docker volume ls --format "{{.Name}}: {{.Driver}}: {{.Mountpoint}}"**

We can create a file directly into the volume

docker@default:~$ **echo '<h2>Volume created with <u>docker volume create</u></h2>' | \**

**sudo tee /mnt/sda1/var/lib/docker/volumes/lv-1/\_data/index.html**

Now, we are ready to start a new container with the volume attached

docker@default:~$ **docker container run -d -p 8000:80 --name co-apache1 \**

**-v lv-1:/var/www/html php:7.0-apache**

We can open a browser tab on the host and navigate to <http://192.168.99.100:8000> (adjust the address to match yours)

Stop the container

docker@default:~$ **docker container stop co-apache1**

#### Volume Containers

With the technique that we will explore now, we aim not only for data sharing, but for less space occupation as well

Let us create the container with

docker@default:~$ **docker container create -v /con-data --name con-store alpine /bin/true**

And check where the data is being stored

docker@default:~$ **docker container inspect con-store | grep -i source**

Now, we can add a simple **readme.txt** file which will be available for other containers

docker@default:~$ **echo 'Read Me File in a Container Volume' | \**

**sudo tee /mnt/sda1/var/lib/docker/volumes/ce46ae99493ba70dbc8c715e148/\_data/readme.txt**

Let us start a new container that is connected to the volume

docker@default:~$ **docker container run -d --volumes-from con-store --name alp1 alpine \**

**sleep 1d**

If we open a session to the container, we will see that everything is according to the plan

docker@default:~$ **docker container exec -it alp1 /bin/sh**

/ # **cat /con-data/readme.txt**

**Read Me File in a Container Volume**

/ # **exit**

Finally, we can stop the container

**docker@default:~$ docker container stop alp1**

## Part 2: Distributed Applications. Docker Compose

### Distributed Applications

When deploying distributed application in a containerized environment usually every component runs in its own container

So, we must ensure that the related components can communicate together

#### Linking

One of the options to provide containers with a way to communicate is to link them

Let us first prepare the environment

Download the accompanying archive file and extract it to a folder on your host

Copy the content of the **M3-2a** sub-folder to the **Docker instance**

**$ scp -r M3-2a/ docker@192.168.99.100:/home/docker/**

The password is **tcuser**

We can enter the instance

**$ docker-machine ssh**

Build the first image by executing the following steps

docker@default:~$ **cd M3-2a/mysql**

docker@default:~/M3-2a/mysql$ **docker image build -t img-mysql .**

Now, build the second image

docker@default:~/M3-2a/mysql$ **cd ../php**

docker@default:~/M3-2a/php$ **docker image build -t img-php .**

docker@default:~/M3-2a/php$ **cd ..**

We are ready to continue

Start the two containers

docker@default:~/M3-2a$ **docker container run -d --name c-mysql \**

**-e MYSQL\_ROOT\_PASSWORD=12345 img-mysql**

docker@default:~/M3-2a$ **docker container run -d --name c-php -p 8080:80 \**

**-v /home/docker/M3-2a:/var/www/html --link c-mysql:dob-mysql img-php**

Now, open a browser tab on your host and navigate to the following URL (adjust the address to match yours):

<http://192.168.99.100:8080>

We should see a sample **PHP** page that reads data from a **MySQL** database

Before we stop the containers, let us enter the **c-php** one and check how the link is expressed

docker@default:~/M3-2a$ **docker container exec -it c-php /bin/bash**

root@fbddfb6f95d1:/var/www/html# **cat /etc/hosts**

**…**

root@fbddfb6f95d1:/var/www/html# **exit**

Now, we can stop both containers

docker@default:~/M3-2a$ **docker container stop c-php c-mysql**

docker@default:~/M3-2a$ **cd ..**

#### Isolated Network

Alternatively, we can create an isolated network and, in this way, allow the containers to communicate

Create a new network

docker@default:~$ **docker network create --driver bridge dob-network**

Now, start the first container

docker@default:~$ **docker container run -d --net dob-network --name dob-mysql \**

**-e MYSQL\_ROOT\_PASSWORD=12345 img-mysql**

And then, the second one

docker@default:~$ **docker container run -d --net dob-network --name dob-php -p 8080:80 \**

**-v /home/docker/M3-2a:/var/www/html img-php**

If we return to the browser tab and refresh the page, we must see exactly the same result achieved in different way

Enter one of them and check the content of the **/etc/hosts** file again. There isn’t any record for the other host

Let us stop the containers

docker@default:~$ **docker container stop dob-mysql dob-php**

### Docker Compose

For the cases in which we want to manage a group of containers, we can use the **Docker Compose**

#### Installation

Again, we have multiple options to install this component

They vary between the different operating systems and methods used during the installation

More information can be read here: <https://docs.docker.com/compose/install/>

To eliminate the discrepancies caused by the different environments, we will install it on the **Docker instance**

docker@default:~$ **curl -L https://github.com/docker/compose/releases/download/1.29.2/docker-compose-`uname -s`-`uname -m` > /tmp/docker-compose && chmod +x /tmp/docker-compose && sudo cp /tmp/docker-compose /usr/local/bin/docker-compose**

There is an alternative set of two commands. First, download the binary

docker@default:~$ **sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose**

Then, make it executable

docker@default:~$ **sudo chmod +x /usr/local/bin/docker-compose**

In any way, execute it to test if it is working fine

docker@default:~$ **docker-compose**

#### Working

Return on the host and copy the next folder with exercise files

**$ scp -r M3-2b/ docker@192.168.99.100:/home/docker/**

The password is **tcuser**

We can enter the instance again

**$ docker-machine ssh**

Check the files

docker@default:~$ **cd M3-2b**

docker@default:~/M3-2b$ **ls -al**

docker@default:~/M3-2b$ **cat .env**

docker@default:~/M3-2b$ **cat docker-compose.yml**

As a last step, check the docker-compose.yml file with

docker@default:~/M3-2b$ **docker-compose config**

Start the build

docker@default:~/M3-2b$ **docker-compose build**

Start the containers

docker@default:~/M3-2b$ **docker-compose up -d**

Now, if return to the browser and refresh again, we must see the same page that we saw earlier

We can explore the status of the containers, their processes and their logs

docker@default:~/M3-2b$ **docker-compose ps**

**…**

docker@default:~/M3-2b$ **docker-compose top**

**…**

docker@default:~/M3-2b$ **docker-compose logs**

**…**

We can start or stop containers individually

Let’s stop the **com-mysql** container

docker@default:~/M3-2b$ **docker-compose stop com-mysql**

Now, return to the browser and refresh the page. No connection to the database message should be displayed

Return to the command line and start it again

docker@default:~/M3-2b$ **docker-compose start com-mysql**

Now, return to the browser and refresh the page. Everything should be just fine

Now, let us stop and delete both containers simultaneously

docker@default:~/M3-2b$ **docker-compose down**

## Part 3: Swarm Cluster

Before we continue let us free some resources

Exit from the **Docker instance** if you are still in

Stop the **Docker instance**

**$ docker-machine stop**

#### Infrastructure Preparation

Execute the following command to create our first **Docker node**

**$ docker-machine create -d virtualbox docker-1**

Repeat the command two more times but change the index at the end to 2 and 3 respectively

Check how many **Docker Machines** we have

**$ docker-machine ls**

Enter the first **Docker host**

**$ docker-machine ssh docker-1**

Initialize the node as the first node of the cluster

docker@docker-1:~$ **docker swarm init --advertise-addr 192.168.99.101**

Ask for the token that can be used to join the workers

docker@docker-1:~$ **docker swarm join-token -q worker**

Copy the result of the above command and exit the instance

docker@docker-1:~$ **exit**

Now, enter the second instance

**$ docker-machine ssh docker-2**

Execute the copied command to join the second node as a worker

docker@docker-2:~$ **docker swarm join \**

**--token SWMTKN-1-3cmw4zhu9fdep8wn162kwark8hoedtt8xw959pztgp125hlaqo-ay73hkczlbxlpmlqy3t2evdtv \**

**--advertise-addr 192.168.99.102 192.168.99.101:2377**

Then exit the session

docker@docker-2:~$ **exit**

And finally, repeat the procedure on the third node

**$ docker-machine ssh docker-3**

Join the third node to the cluster

docker@docker-3:~$ **docker swarm join \**

**--token SWMTKN-1-3cmw4zhu9fdep8wn162kwark8hoedtt8xw959pztgp125hlaqo-ay73hkczlbxlpmlqy3t2evdtv \**

**--advertise-addr 192.168.99.103 192.168.99.101:2377**

Close the session to the third node

docker@docker-3:~$ **exit**

We can return on the first host and ask for the nodes list

**$ docker-machine ssh docker-1**

docker@docker-1:~$ **docker node ls**

#### Start a Service

To start a simple service, we can execute

docker@docker-1:~$ **docker service create --replicas 1 --name pinger alpine ping softuni.bg**

We can ask for a listing of the running services

docker@docker-1:~$ **docker service ls**

To see the details of a service, execute

docker@docker-1:~$ **docker service inspect pinger**

There is an alternative more beautiful version

docker@docker-1:~$ **docker service inspect --pretty pinger**

We can see the tasks of a service and where (on which node) they are running

docker@docker-1:~$ **docker service ps pinger**

Let us increase the number of tasks to 5

docker@docker-1:~$ **docker service scale pinger=5**

And check where they are running

docker@docker-1:~$ **docker service ps pinger**

#### Node Maintenance

We may have a need to do some maintenance on a node

First, we must drain the node and then do the maintenance

Draining is done in the following way

docker@docker-1:~$ **docker node update --availability drain docker-2**

Ask for information about the drained node

docker@docker-1:~$ **docker node inspect --pretty docker-2**

Check details about the service we have

docker@docker-1:~$ **docker service ps pinger**

We can see that the requested number of copies of the service is still the same but redistributed

Bring back the host as part of the cluster

docker@docker-1:~$ **docker node update --availability active docker-2**

Ask again for detailed information for the node

docker@docker-1:~$ **docker node inspect --pretty docker-2**

Check again the service

docker@docker-1:~$ **docker service ps pinger**

As we can see, the node is back, but the tasks were not redistributed again

We can force the redistribution of the tasks

docker@docker-1:~$ **docker service update --force pinger**

The above command, archives the requested but with the price of restarting the service

There are situations in which this is unacceptable

Instead, we can modify the service definition when we start it

docker@docker-1:~$ **docker service create --replicas 1 --name pinger \**

**--update-delay 10s alpine ping softuni.bg**

We can even add a parameter stating how many tasks can be updated simultaneously with **--update-parallelism**

We can stop the service

docker@docker-1:~$ **docker service rm pinger**

#### Group of Services (Stack)

Exit on the host machine if you are still in one of the **Docker instances**

Being on the host, copy the next folder with exercise files on host #1

**$ scp -r M3-3/ docker@192.168.99.101:/home/docker/**

The password is **tcuser**

Repeat the procedure for the other two hosts as well

Create new session to host #1

**$ docker-machine ssh docker-1**

Enter the copied folder and execute the following to start the stack

docker@docker-1:~$ **cd M3-3**

docker@docker-1:~/M3-3$ docker **stack deploy -c docker-compose.yml docker-help**

List available stacks

docker@docker-1:~/M3-3$ **docker stack ls**

List the services in the **docker-help** stack

docker@docker-1:~/M3-3$ **docker stack services docker-help**

Check information about the stack

docker@docker-1:~/M3-3$ **docker stack ps docker-help**

We can open a browser tab on the host and navigate to <http://192.168.99.101:8080>

We must see the same application as before

Try with the IP addresses of the other two nodes

Finally, we can stop the stack with

docker@docker-1:~/M3-3$ **docker stack rm docker-help**

#### Sharing Information

There are several ways to share information between our host and a **Docker Machine**

* Mount a folder from the instance on the host

**docker-machine mount docker-vm:/home/vm/dir /home/user/dir**

This approach is not suitable for exchanging information between multiple machines

Furthermore, on Windows it requires external components to work

* Mount a host folder in an instance

**docker-machine create --driver virtualbox \**

**--virtualbox-share-folder C:\Temp:/home/docker/Temp default**

This way, we can share a folder on the host with multiple virtual machines

## Cleaning

All hosts (or instances) can be deleted either one by one or as a whole:

**$ docker-machine rm default docker-1 docker-2 docker-3**