# Практика: DOB Модул 3 – Docker-Machine

За целите на упражненията в тази практика приемаме, че работим върху физическа машина с инсталирана **Linux** дистрибуция за предпочитане **CentOS7** и инсталиран **VirtualBox**. Приемаме също, че всички разполагаме с архивния файл и сме го разархивирали в папка **DOB-Docker** в папката на нашия потребител.

Всички действия, без описаните в секция „Подготовка“, са приложими и при използването на продуктите **Docker Toolbox for Windows** и **Docker Toolbox for Mac**

## Подготовка

#### Инсталиране на docker-machine

Инсталацията на **docker-machine** можем да направим като изпълним прикачения скрипт (**install-docker-machine.sh**) или копираме и поставим в конзолата следните инструкции:

[user@host ~]$ curl -L https://github.com/docker/machine/releases/download/v0.13.0/docker-machine-`uname -s`-`uname -m` >/tmp/docker-machine && \

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

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

След успешното приключване на инсталацията, можем да пристъпим към създаване на първата (**default**) инстанция на **Docker**:

[user@host ~]$ docker-machine create --driver virtualbox default

Списъка от създадени инстанции на **Docker** можем да видим като изпълним:

[user@host ~]$ docker-machine ls

NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS

default - virtualbox Running tcp://192.168.99.100:2376 v17.10.0-ce

Колонката **URL** ни дава информация за това на кой адрес можем да комуникираме с контейнерите.

#### Инсталиране на самостоятелен docker клиент

Инсталацията на **docker** клиент можем да направим като изпълним прикачения скрипт (**install-docker-client.sh**) или копираме и поставим в конзолата следните инструкции:

[user@host ~]$ curl -L https://download.docker.com/linux/static/stable/x86\_64/docker-17.09.0-ce.tgz > /tmp/docker-17.09.0-ce.tgz && \

tar xzvf /tmp/docker-17.09.0-ce.tgz && \

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

rm -rf docker/ && \

rm /tmp/docker-17.09.0-ce.tgz

Като алтернатива можем да използваме edge канала**: https://download.docker.com/linux/static/edge/x86\_64/**

След успешното приключване на инсталацията, можем да пристъпим към тест на **docker** клиента:

[user@host ~]$ docker version

#### Инициализиране и де-инициализиране на средата

В случай, че имаме инсталиран **Docker** клиент на нашата работна машина и искаме всички команди, които подаваме през него да се изпълняват срещу конкретна инстанция на **Docker**, трябва да зададем контекст. Това става с т.нар. процес на инициализация на средата:

[user@host ~]$ docker-machine env default

[user@host ~]$ eval $(docker-machine env)

Първата команда дава информация за променливите на средата заедно със стойностите, които ще им бъдат присвоени за работа с инстанция на **Docker** с етикет **default**. Втората команда установява съответния контекст.

Сега бихме могли да използваме локалния клиент и да подадем команда към инсталираната инстанция:

[user@host ~]$ docker run shekeriev/welcome-dob-oct-17

Ако искаме да занулим установения контекст:

[user@host ~]$ docker-machine env -u

[user@host ~]$ eval $(docker-machine env -u)

[user@host ~]$ env | grep DOCKER

Първата команда връща информация за променливите, които ще бъдат занулени, втората извършва самото действие, а с третата можем да се убедим в резултата.

Ако искаме да изпълняваме команди директно в инстанцията на **Docker** без да използваме локален клиент, единият вариант е да подаваме командите така:

[user@host ~]$ docker-machine ssh default 'docker image ls'

Друг вариант е да влезем в съответната инстанция на **Docker** и да изпълняваме командите локално:

[user@host ~]$ docker-machine ssh default

docker@default:~$ docker image ls

За да си гарантираме еднаквост по отношение на средата, поведението и резултатите, за целите на тази практика ще изберем втория вариант, т.е. ще влезем в инстанцията на **Docker**.

## Networks

Нека проверим какви мрежи има дефинирани по подразбиране на нашата инстанция:

docker@default:~$ docker network ls

Сега нека да създадем наша мрежа от тип **bridge** с и ме **dob-bridge**:

docker@default:~$ docker network create -d bridge --subnet 10.0.0.1/24 dob-bridge

docker@default:~$ docker network ls

Можем да получим подробна информация за създадената мрежа:

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

Нека да създадем последователно два контейнера на базата на шаблона **alpine**:

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

Сега можем да влезем последователно в единия и в другия и да проверим мрежовата им свързаност:

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

Нека отново да изследваме мрежата, която създадохме по-рано:

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

Виждаме, че двата контейнера са закачени към мрежата и техните адреси. Сега можем да ги спрем:

docker@default:~$ docker container stop co1 co2

Нека да изтрием мрежата:

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

## Volumes

Хранилища на споделени данни можем да създаваме и предоставяме за ползване по няколко начина. Нека ги изследваме в следващите стъпки.

#### В движение

Нека да създадем един контейнер, към който да закачим хранилище на данни, т.е. папка на локалния хост, ще бъде налична и в контейнера (**/test-vol**):

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

Сега ще излезем с **Ctrl+p** и **Ctrl+q** и ще създадем втори контейнер, който ще унаследи споделената папка от първия:

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

Отново излизаме с **Ctrl+p** и **Ctrl+q** и се закачаме към първия контейнер:

docker@default:~$ docker container attach c1

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

Излизаме с **Ctrl+p** и **Ctrl+q** и се закачаме към втория контейнер, добавяме текст и спираме контейнера:

docker@default:~$ docker container attach c2

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

root@937b91cf5b51:/# exit

Сега отново се закачаме към първия контейнер, преглеждаме файла и спираме контейнера:

docker@default:~$ docker container attach c1

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

root@8f7010fff13d:/# exit

Нека се уверим, че и двата контейнера са спрени:

docker@default:~$ docker container ps

Сега ще стартираме трети контейнер, който ще наследи папката от **c1** и на свой ред ще добави запис:

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

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

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

Излизаме с **Ctrl+p** и **Ctrl+q** , стартираме отново **c1** и се закачаме към него:

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

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

Отново излизаме с **Ctrl+p** и **Ctrl+q** и изпълняваме:

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

С нея виждаме коя е папката на нашата хост система, която е закачена като споделена.

Друг вариант да получим информация за хранилищата на споделени данни е като изпълним:

docker@default:~$ docker volume ls

Детайлна информация за конкретното хранилище можем да получим като копираме идентификатора му и го подадем като аргумент в следната команда:

docker@default:~$ docker volume inspect f2d6e112f178b918f4e204312

Сега можем през хоста да изследваме съдържанието на файла, намиращ се в споделената папка:

docker@default:~$ sudo su root

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

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

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

Като последна стъпка можем да спрем двата контейнера:

docker@default:~$ docker container stop c1 c3

#### Закачване на съществуваща папка

Нека да направим папка на хоста, която ще споделим по-късно с контейнер. Веднага след това ще създадем и опростен **index.html** файл:

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

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

Сега ще стартираме контейнера и ще закачим папката:

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

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

Нека сега на нашата станция да отворим браузър и да подадем следния адрес:

<http://192.168.99.100:8080>

Където **IP** адресът е този, който сме получили при първоначалното стартиране на **Docker** инстанцията.

Сега можем да се закачим към контейнера и да разгледаме папката:

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

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

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

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

Нека сега да променим файла на **Docker** хоста:

docker@default:~$ echo '<br><br><h3>Recently updated</h3>' >> /home/docker/web/index.html

Ако опресним съдържанието на браузъра би трябвало да видим промяната, която направихме по-рано. Можем да спрем контейнера:

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

#### Създаване на хранилище за споделени данни

В това упражнение ще създадем предварително хранилището на данни и ще му зададем етикет. След това ще го изследваме по няколко начина:

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

docker@default:~$ docker volume ls

...

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

...

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

...

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

...

Нека сега да създадем и файл, който в последствие ще използваме в контейнер:

docker@default:~$ sudo su root

docker@default:~$ echo '<h2>Volume created with <u>docker volume create</u><h2>' >> /mnt/sda1/var/lib/docker/volumes/lv-1/\_data/index.html

docker@default:~$ exit

След това стартираме контейнер и закачваме хранилището:

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

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

Можем да видим ефекта в браузър на нашата станция, като подадем примерно:

<http://192.168.99.100:8000>

След това нека спрем контейнера:

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

#### Създаване на контейнери хранилища

Целта на този подход е освен да предоставим възможност за споделяне на данни, да намалим и заеманото от контейнерите място, като споделим слоевете помежду им.

Нека да създадем контейнера и да видим какъв е пътят до него:

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

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

Сега бихме могли да поставим **readme.txt** файл, който да бъде видим от следващите контейнери:

docker@default:~$ sudo su root

root@default:~# echo 'Read Me File in a Container Volume' >> /mnt/sda1/var/lib/docker/volumes/ce46ae99493ba70dbc8c715e148aac542cf444ba5/\_data/readme.txt

root@default:~# exit

Нека да стартираме контейнер, към който да закачим контейнера-хранилище:

docker@default:~$ docker container run -d --volumes-from con-store --name alp1 alpine sleep 1d

Ако установим връзка с контейнера и проверим съдържанието на файла, ще се убедим, че всичко е по план:

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

/ # cat /con-data/readme.txt

Read Me File in a Container Volume

/ # exit

Като последна стъпка можем да спрем контейнера:

docker@default:~$ docker container stop alp1

## Linking

Един от вариантите да предоставим възможност на контейнерите да обменят информация помежду си както това се случва между отделни сървъри, е да ги „свържем“. Нека първо да подготвим опитната постановка.

Първо трябва да копираме папката **w3-2** заедно с цялото ѝ съдържание на **Docker** хоста (машината):

docker@default:~$ mkdir w3-2

docker@default:~$ exit

[user@host ~]$ cd DOB-Docker

[user@host DOB-Docker]$ scp -r w3-2/\* docker@192.168.99.100:/home/docker/w3-2

Паролата за достъп е **tcuser**

Нека се върнем в **Docker** машината и последователно да създадем двата шаблона:

[user@host DOB-Docker]$ docker-machine ssh

docker@default:~$ cd w3-2/mysql

docker@default: ~/w3-2/mysql$ docker image build -t img-mysql .

docker@default: ~/w3-2/mysql$ cd ../php

docker@default: ~/w3-2/php$ docker image build -t img-php .

docker@default: ~/w3-2/php$ cd ..

Вече сме готови да продължим напред. Сега стартираме последователно двата контейнера:

docker@default: ~/w3-2$ docker container run -d --name c-mysql \

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

docker@default: ~/w3-2$ docker container run -d --name c-php -p 8080:80 \

-v /home/docker/w3-2:/var/www/html --link c-mysql:dob-mysql img-php

Сега ако отворим браузър на нашата станция и въведем следния адрес:

<http://192.168.99.100:8080>

Би трябвало да видим примерен сайт на **PHP**, който чете данни, съхранени в **MySQL** база от данни. Можем да спрем контейнерите:

docker@default: ~/w3-2$ docker container stop c-php c-mysql

docker@default: ~/w3-2$ cd ..

## Isolated Network

Алтернатива на свързването на контейнери е изолацията им в отделна мрежа. За да демонстрираме този подход на практика ще използваме вече познати техники. Първо ще създадем мрежата и след това ще стартираме два контейнера, които са част от нея:

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

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

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

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

-v /home/docker/w3-2:/var/www/html img-php

Ако сега стартираме браузър на нашата станция и въведем адреса от предходното упражнение, ще видим същия резултат, но постигнат по друг начин. Можем да спрем контейнерите:

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

## Docker-Compose single host

За случаите, в които искаме да управляваме група контейнери заедно, можем да използваме инструмента **Docker Compose**.

#### Инсталиране

Можем да инсталираме **Docker Compose** на **Docker** хоста, като изпълним следния скрипт:

docker@default: ~$ curl -L https://github.com/docker/compose/releases/download/1.17.0/docker-compose-`uname -s`-`uname -m` > /tmp/docker-compose && chmod +x /tmp/docker-compose && sudo cp /tmp/docker-compose /usr/local/bin/docker-compose

#### Упражнение

Първо трябва да копираме папката **w3-2-c** заедно с цялото ѝ съдържание на **Docker** хоста (машината):

docker@default:~$ mkdir w3-2-c

docker@default:~$ exit

[user@host ~]$ cd DOB-Docker

[user@host DOB-Docker]$ scp -r w3-2-c/ docker@192.168.99.100:/home/docker

Паролата за достъп е **tcuser**

Нека се върнем в **Docker** машината и последователно да прегледаме съдържанието на следните файлове:

[user@host DOB-Docker]$ docker-machine ssh

docker@default:~$ mkdir w3-2-com

docker@default:~$ cp w3-2-c/php/php/index.php w3-2-com/index.php

docker@default:~$ cd w3-2-c

docker@default: ~/w3-2-c$ cat .env

docker@default: ~/w3-2-c$ cat docker-compose.yml

Сега можем да пристъпим към създаване на групата от контейнери и тяхното стартиране:

docker@default: ~/w3-2-c$ docker-compose build

docker@default: ~/w3-2-c$ docker-compose up -d

Ако отворим браузър и въведем адреса от предходното упражнение, би трябвало да видим познатата страница.

Сега можем да видим състоянието на контейнерите и техните логове:

docker@default: ~/w3-2-c$ docker-compose ps

docker@default: ~/w3-2-c$ docker-compose logs

Можем да спрем контейнерите с командата **stop** и после да ги стартираме отново с командата **start**, а също така и да ги спрем и изтрием едновременно с командата **down**:

docker@default: ~/w4-2-c$ docker-compose down

## Клъстер (Swarm)

За да освободим ресурси на нашия хост, можем да излезем от текущата **Docker** машина и да я спрем:

docker@default: ~/w3-2-c$ exit

[user@host DOB-Docker]$ docker-machine stop

#### Създаване на инфраструктурата

Можем да създадем трите необходими **Docker** хоста по следния начин:

[user@host DOB-Docker]$ for i in 1 2 3; do \

docker-machine create -d virtualbox docker-$i; \

done

[user@host DOB-Docker]$ docker-machine ls

Сега можем да влезем в първия хост и да изпълним следната команда:

[user@host DOB-Docker]$ docker-machine ssh docker-1

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

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

Копираме резултата от изпълнението на последната команда и го използваме при конструиране на командите, които ще изпълним последователно на другите два хоста, за да ги включим в клъстера като работници. Първо на хост 2:

docker@docker-1:~$ exit

[user@host DOB-Docker]$ docker-machine ssh docker-2

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

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

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

docker@docker-2:~$ exit

После и на хост номер 3:

[user@host DOB-Docker]$ docker-machine ssh docker-3

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

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

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

docker@docker-3:~$ exit

Сега можем да влезем на хост номер 1 и да видим състоянието на клъстера:

[user@host DOB-Docker]$ docker-machine ssh docker-1

docker@docker-1:~$ docker node ls

#### Стартиране на услуга (Service)

За да стартираме услуга, можем да изпълним:

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

Със следващите команди можем да изследваме какви услуги се изпълняват и да получим повече детайли за услугата, която стартирахме по-рано:

docker@docker-1:~$ docker service ls

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

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

Можем да видим на кои участници (**nodes**) в клъстера се изпълнява нашата услуга:

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

Сега нека да увеличим копията на услугата на 5 и да проверим как са разпределени:

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

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

#### Освобождаване (drain) и активиране на участник в клъстера

Ако ни се налага да извършваме някакви действия с някой от участниците в клъстера, но не искаме това да доведе до отпадане на част от услугите, правилния начин да го направим е да маркираме съответния хост за освобождаване:

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

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

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

С последната команда можем да се уверим, че броя на заявените копия на услугата се поддържа същия, т.е. бройката на отпадналите вследствие оттеглянето на хост 2, се преразпределя между останалите хостове. Вече можем да маркираме хост 2 като активен и да проверим как се отразява това:

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

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

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

Както можем да видим от последната команда, въпреки че хост 2 е наличен вече работещите копия на услугата не се преразпределят повторно. Всички следващи заявки ще отчитат това, че хост 2 е вече наличен.

Има вариант за форсирано преразпределение на вече работещите копия на услугата:

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

Горната команда на практика рестартира услуга, при което нейни копия ще бъдат разпределени и на добавения хост. Трябва да имаме предвид, че това води до временно прекъсване на услугата.

Има все пак вариант да избегнем подобни нежелани ситуации. За целта по време на дефиниране на услугата трябва да подадем допълнителен параметър за поетапен ъпдейт:

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

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

В допълнение бихме могли да укажем колко задачи (копия) на услугата да бъдат ъпдейтвани едновременно с модификатора **--update-parallelism**

Можем да спрем услугата:

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

#### Група от услуги (Stack)

Трябва първо да копираме папка **w3-3** на всеки хост. Първо хост 1:

docker@docker-1:~$ exit

[user@host DOB-Docker]$ docker-machine ssh docker-1 "mkdir /home/docker/w3-3"

[user@host DOB-Docker]$ scp w3-3/docker-compose.yml docker@192.168.99.101:/home/docker/w3-3

[user@host DOB-Docker]$ scp w3-3/php/php/index.php docker@192.168.99.101:/home/docker/w3-3

А след това хост 2 и 3:

[user@host DOB-Docker]$ docker-machine ssh docker-2 "mkdir /home/docker/w3-3"

[user@host DOB-Docker]$ scp w3-3/php/php/index.php docker@192.168.99.102:/home/docker/w3-3

[user@host DOB-Docker]$ docker-machine ssh docker-3 "mkdir /home/docker/w3-3"

[user@host DOB-Docker]$ scp w3-3/php/php/index.php docker@192.168.99.103:/home/docker/w3-3

Паролата е **tcuser**

След това се връщаме в хост 1 и изпълняваме следната последователност:

[user@host DOB-Docker]$ docker-machine ssh docker-1

docker@docker-1:~$ cd w3-3

docker@docker-1:~/w3-3$ docker stack deploy -c docker-compose.yml docker-help

docker@docker-1:~/w3-3$ docker stack ps docker-help

Сега можем да отворим браузър и да посетим първо адреса на хост 1, а после и на другите. Би трябвало да се отвори познатата от предходните упражнения страница:

<http://192.168.99.101:8080>

Можем да спрем комплекта от услуги:

docker@docker-1:~/w3-3$ docker stack rm docker-help

## Почистване на инфраструктурата

Хостовете можем да изтрием накуп със следната команда:

docker@docker-1:~/w3-3$ exit

[user@host DOB-Docker]$ docker-machine rm default docker-1 docker-2 docker-3