# Практика: DOB Модул 2

За целите на стъпките, описани по-долу се приема, че се работи в **CentOS 7 64 бита** среда, с потребител **devops** член на **wheel** групата.

## Инсталиране и конфигуриране на Docker

**Референции**

<https://docs.docker.com/engine/installation/linux/centos/>

<https://docs.docker.com/engine/installation/linux/linux-postinstall/>

**Команди**

**version, system info, container run**

### Премахване на стари версии

$ sudo yum remove docker \

docker-common \

container-selinux \

docker-selinux \

docker-engine

### Подготовка за инсталация

$ sudo yum install -y yum-utils device-mapper-persistent-data lvm2

$ sudo yum-config-manager \

--add-repo \

https://download.docker.com/linux/centos/docker-ce.repo

### Същинска инсталация

$ sudo yum makecache fast

$ sudo yum install docker-ce

Стартираме **docker** и проверяваме дали всичко е наред:

$ sudo systemctl start docker

$ sudo systemctl status docker

При нормално работеща услуга, бихме могли да получим повече информация за инсталираните версии на компонентите и текущата конфигурация:

$ sudo docker version

$ sudo docker system info

След това можем да стартираме първия си контейнер:

$ sudo docker container run hello-world

### Допълнителни настройки

За да можем да работим с **docker** без да се налага постоянно да използваме **sudo**, можем да изпълним следната команда:

$ sudo usermod -aG docker $USER

След това, за да влезе в сила направената промяна, трябва да изпълним **log off** - **log on** процедура.

Можем да направим проверка, като изпълним:

$ docker container run hello-world

Като последна стъпка, можем да настроим **docker** да стартира автоматично при стартиране на системата:

$ sudo systemctl enable docker

### Смяна на драйвера за съхранение

Спираме docker ако работи и проверяваме дали е наличен файлът **/etc/docker/daemon.json**. Ако липсва някой от компонентите (папката или файлът), то можем да ги създадем съответно с командите:

$ sudo mkdir -p /etc/docker

$ sudo touch /etc/docker/daemon.json

Отваряме файла **daemon.json** и добавяме следните три реда:

{

"storage-driver": "devicemapper"

}

Стартираме **docker** и проверяваме информацията, за текущата конфигурация.

Ако искаме да настроим сторидж системата на **docker** в режим наподобяващ продукционна среда, можем да направим следното:

* Спираме **docker** и машината;
* Добавяме нов диск и стартираме машината;
* Премахваме или запазваме наличните контейнери и спираме **docker**;
* Променяме съдържанието на **/etc/docker/daemon.json** файла до следното:

{

"storage-driver": "devicemapper",

"storage-opts": [

"dm.directlvm\_device=/dev/sdb",

"dm.thinp\_percent=95",

"dm.thinp\_metapercent=1",

"dm.thinp\_autoextend\_threshold=80",

"dm.thinp\_autoextend\_percent=20",

"dm.directlvm\_device\_force=false"

]

}

* Трябва само да сменим **/dev/sdb** на устройството налично в нашата система;
* Запазваме файла и стартираме **docker**;

При горната конфигурация делегираме контрола върху устройството на **docker**.

Бихме могли да подходим и по друг начин - ние да контролираме устройството. Това ще ни даде възможност да направим по-сложна конфигурация, примерно с няколко диска, а полученият дял да предоставим за използване на **docker**.

## Работа с шаблони и контейнери

**Референции**

<https://hub.docker.com/explore/>

<https://store.docker.com/>

**Команди**

**search**, **image** **pull**, **image** **ls**, **image** **rm**, **container** **run**, **container** **ls**, **container** **rm**, **container** **create**, **container** **start, container attach, container stop, container prune**

### Примерни действия

Бихме могли да потърсим всички шаблони които, съдържат **ubuntu** в името си:

$ docker search ubuntu

Първият запис в списъка е с най-голям рейтинг и е маркиран като официален. Можем да го изтеглим локално, като изпълним командата:

$ docker image pull ubuntu

Всъщност горната команда е еквивалентна на тази:

$ docker image pull ubuntu:latest

Резултатът от изпълнението и на двете ще бъде изтегляне на последната версия. Ако искаме да изтеглим конкретна версия, можем да изпълним командата:

$ docker image pull ubuntu:14.10

Списък с наличните шаблони, можем да получим като изпълним следната команда:

$ docker image ls

Нека да стартираме контейнер от шаблона на ubuntu:14.10

$ docker container run -it ubuntu:14.10

Бидейки в контейнера, можем да изпълним последователно командите ls и ps ax:

root@35ac9218a880:/# **ls**

bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var

root@35ac9218a880:/# ps ax

PID TTY STAT TIME COMMAND

1 ? Ss 0:00 /bin/bash

15 ? R+ 0:00 ps ax

root@35ac9218a880:/#

Ако искаме временно да излезем от контейнера, но без да го спираме, можем да направим това с последователното натискане на клавишните комбинации **Ctrl+P** и **Ctrl+Q**.

Сега можем да видим списък на всички работещи контейнери:

$ docker container ls

Резултатът от изпълнението ѝ би трябвало да изглежда горе-долу така:

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

35ac9218a880 ubuntu:14.10 "/bin/bash" 8 minutes ago Up 8 minutes cocky\_fermat

Ако искаме да се върнем обратно в контейнера, от който излязохме по-рано, можем да го направим така:

$ docker container attach 35ac9218a880

Като **35ac9218a880** е идентификатора на контейнера (първата колонка от изхода на команда **docker container ls**). Вместо него бихме могли да използваме автоматично генерираното име (последната колонка от изхода на команда **docker container** **ls**) – **cocky\_fermat**

Бихме могли да излезем от контейнера и да преустановим изпълнението му с командата **exit**:

root@35ac9218a880:/# exit

За да видим списък на всички контейнери, включително приключилите своето изпълнение, изпълняваме:

$ docker container ls -a

Бихме могли да създадем контейнер и без да го стартираме, като изпълним командата:

$ docker container create -it --name ubuntu-14 ubuntu /bin/bash

След това можем да го стартираме като изпълним командата:

$ docker container start -ai ubuntu-14

Нека да излезем от контейнера без да го спираме с последователното натискане на клавишните комбинации **Ctrl+P** и **Ctrl+Q**. Сега бихме могли да го спрем с командата:

$ docker container stop ubuntu-14

Ако искаме да спрем всички работещи контейнери, можем да изпълним следната команда:

$ docker container stop $(docker container ls -q)

Конкретен контейнер или списък от контейнери можем да премахнем с командата:

$ docker container rm ubuntu-14

Всички спрени контейнери можем да изтрием с командата:

$ docker container prune

Можем да изтрием един или повече шаблона с командата:

$ docker image rm ubuntu

## Обмен на шаблони

**Команди**

**container run**, **container export**, **image pull**, **image import**, **image save**, **image load**

### Архив и трансфер на шаблони от контейнери

Нека да стартираме контейнер от шаблона **Alpine** (**alpine**):

$ docker container run -it --name my-alpine alpine

След това можем да изпълним командите **ls**, **echo**, **cat** и **exit**, както е показано по-долу:

/ # **ls**

bin dev etc home lib media mnt proc root run sbin srv sys tmp usr var

/ # **echo 'Hello from Alpine container!' > readme.txt**

/ # **cat readme.txt**

Hello from Alpine container!

/ # **ls**

bin etc lib mnt readme.txt run srv tmp var

dev home media proc root sbin sys usr

/ # **exit**

Този контейнер бихме могли да го съхраним като **tar** архив и да го импортираме като шаблон на друга **docker** инстанция. В случая ще използваме нашата инстанция. Командата за експорт е:

$ docker container export -o my-alpine.tar my-alpine

Процесът на импорт изисква допълнителни опции, тъй като по време на този вид експорт не се запазват **CMD** и **ENV** свойствата на контейнера. Командата е:

$ docker image import my-alpine.tar --change "CMD /bin/sh" my-new-alpine

Сега бихме могли да стартираме контейнер от новия шаблон и да проверим дали нашият потребителски **readme.txt** файл е на линия. За целта трябва да изпълним следните команди:

$ docker container run -it my-new-alpine

/ # **ls**

bin etc lib mnt readme.txt run srv tmp var

dev home media proc root sbin sys usr

/ # **cat readme.txt**

Hello from Alpine container!

/ # **exit**

### Архив и трансфер на шаблони

Бихме могли да архивираме директно шаблони и да ги местим между отделните **docker** инстанции. Нека изпълним следната поредица от команди:

$ docker image pull busybox

$ docker image save -o busybox.tar busybox

Нека сега да опитаме да импортираме шаблона от създадения архивен файл. Първо обаче трябва да изтрием стария шаблон:

$ docker image rm busybox

$ docker image load -i busybox.tar

## Създаване на шаблон от контейнер

**Команди**

**image** **ls**, **container** **run**, **container** **ls**, **container** **commit**

За да се подготвим за това упражнение, можем да стартираме контейнер от шаблона на **Ubuntu (ubuntu)**, като изпълним следната команда:

$ docker container run --name=my-ubuntu -it ubuntu

Можем да излезем от установената към контейнера връзка, без да го спираме, като натиснем последователно **Ctrl+P** и **Ctrl+Q**. Бихме могли да проверим дали контейнерът е все още в режим на изпълнение:

$ docker container ls -f name=my-ubuntu

Създаването на шаблон от контейнер (включително работещ) е изключително опростен процес и се постига при изпълнение на следната команда:

$ docker container commit --author "SoftUni Student" my-ubuntu new-ubuntu

Нека да проверим дали имаме шаблон с новото име:

$ docker image ls new-ubuntu

Като последна стъпка бихме могли да стартираме контейнер, базиран на новия шаблон и след това да го спрем:

$ docker container run -it new-ubuntu

root@fc0bc2b9b8ab:/# **exit**

## Създаване на шаблони от файлове

**Команди**

**image** **build**, **container** **run**, **container** **ls**, **container** **commit**, **image history**, **container prune**, **image rm**

### Създаване от heredoc файл

Подходът за създаване на текстови файлове в движение, т.нар. **heredoc** файлове, може да бъде приложен и при създаването на **docker** шаблони. Нека изпълним следната команда:

$ docker image build -t alp-htop - << EOF

FROM alpine

RUN apk --no-cache add htop

EOF

Сега можем да стартираме контейнера, базиран на току що създадения шаблон:

$ docker container run -it alp-htop

/ # **htop**

/ # **exit**

### Създаване от Dockerfile

Ще създадем два идентични от потребителска гледна точка шаблона, базирани на сходни **Dockerfile** файлове. Нека последователно изпълним следните действия:

[devops@ldostation2 ~]$ mkdir nginx-1

[devops@ldostation2 ~]$ cd nginx-1

[devops@ldostation2 nginx-1]$ touch Dockerfile

[devops@ldostation2 nginx-1]$ vi Dockerfile

Нека да въведем следното като съдържание на файла:

FROM ubuntu

LABEL maintainer "SoftUni Student"

RUN apt-get update

RUN apt-get install -y nginx

ENTRYPOINT ["/usr/sbin/nginx","-g","daemon off;"]

EXPOSE 80

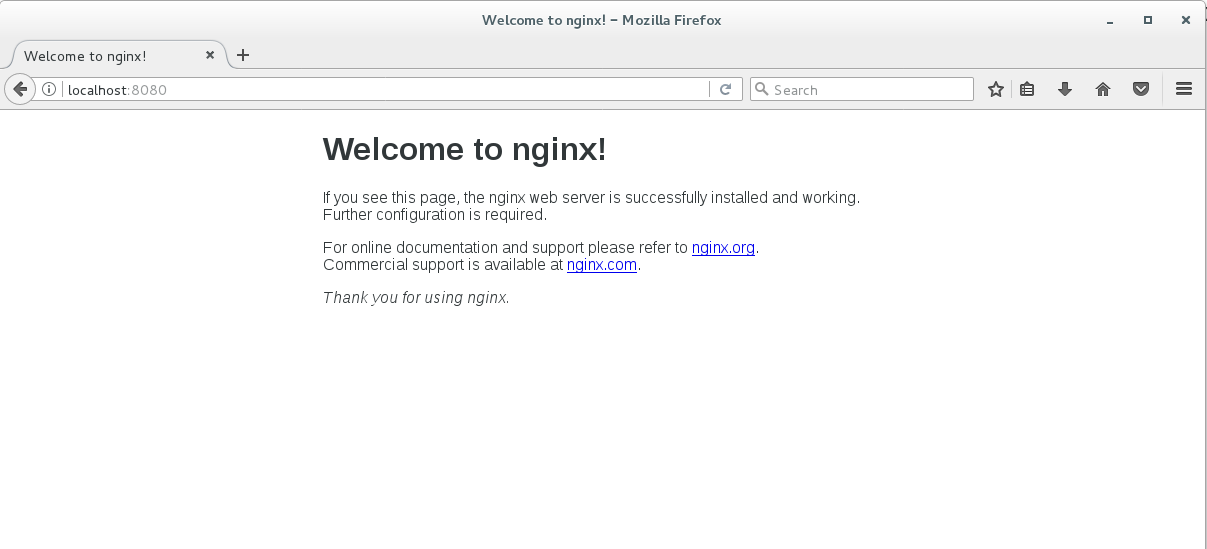
След като го запишем, нека да генерираме шаблона и да стартираме контейнер, базиран на него:

$ docker image build -t nginx-1 .

...

$ docker container run -d -p 8080:80 --name web-1 nginx-1

Сега можем да отворим браузър и да въведем следния адрес <http://localhost:8080> . В резултат на това, би трябвало да видим страницата по подразбиране на **nginx**:



Можем да го спрем като изпълним следната команда:

$ docker container stop web-1

Нека сега да създадем и втория контейнер, извършвайки следната поредица от действия:

[devops@ldostation2 nginx-1]$ cd ..

[devops@ldostation2 ~]$ mkdir nginx-2

[devops@ldostation2 ~]$ cd nginx-2/

[devops@ldostation2 nginx-2]$ touch Dockerfile

[devops@ldostation2 nginx-2]$ vi Dockerfile

Нека да въведем следните команди като съдържание на файла:

FROM ubuntu

LABEL maintainer "SoftUni Student"

RUN apt-get update && apt-get install -y nginx

ENTRYPOINT ["/usr/sbin/nginx","-g","daemon off;"]

EXPOSE 80

След като го запишем, нека да генерираме шаблона и да стартираме контейнер, базиран на него:

$ docker image build -t nginx-2 .

...

$ docker container run -d -p 8080:80 --name web-2 nginx-2

Ако отворим отново браузъра и посетим адрес <http://localhost:8080> ще се уверим, че резултатът е идентичен. От гледна точка на **docker** обаче нещата не стоят точно така. Нека да изпълним последователно следните две команди, за да се уверим в това:

[devops@ldostation2 nginx-2]$ docker image history nginx-1

IMAGE CREATED CREATED BY SIZE

e232a6ba87f2 14 minutes ago /bin/sh -c #(nop) EXPOSE 80/tcp 0 B 53cee9ce5577 14 minutes ago /bin/sh -c #(nop) ENTRYPOINT ["/usr/sbin/... 0 B 9c4a4265ca72 14 minutes ago /bin/sh -c apt-get install -y nginx 56.5 MB e7bb8c797027 14 minutes ago /bin/sh -c apt-get update 38.5 MB d26f0a8d43ae 15 minutes ago /bin/sh -c #(nop) LABEL maintainer=SoftUn... 0 B 7b9b13f7b9c0 2 weeks ago /bin/sh -c #(nop) CMD ["/bin/bash"] 0 B <missing> 2 weeks ago /bin/sh -c mkdir -p /run/systemd && echo '... 7 B <missing> 2 weeks ago /bin/sh -c sed -i 's/^#\s\*\(deb.\*universe\... 2.76 kB <missing> 2 weeks ago /bin/sh -c rm -rf /var/lib/apt/lists/\* 0 B <missing> 2 weeks ago /bin/sh -c set -xe && echo '#!/bin/sh' >... 745 B <missing> 2 weeks ago /bin/sh -c #(nop) ADD file:5aff8c59a707833... 118 MB

[devops@ldostation2 nginx-2]$ docker image history nginx-2

IMAGE CREATED CREATED BY SIZE

9540f6cd2f5f 2 minutes ago /bin/sh -c #(nop) EXPOSE 80/tcp 0 B b0e51c3cacc0 2 minutes ago /bin/sh -c #(nop) ENTRYPOINT ["/usr/sbin/... 0 B 372d0ba9f9a4 2 minutes ago /bin/sh -c apt-get update && apt-get insta... 95 MB 1fec01b2e84d 2 minutes ago /bin/sh -c #(nop) LABEL maintainer=SoftUn... 0 B 7b9b13f7b9c0 2 weeks ago /bin/sh -c #(nop) CMD ["/bin/bash"] 0 B <missing> 2 weeks ago /bin/sh -c mkdir -p /run/systemd && echo '... 7 B <missing> 2 weeks ago /bin/sh -c sed -i 's/^#\s\*\(deb.\*universe\... 2.76 kB <missing> 2 weeks ago /bin/sh -c rm -rf /var/lib/apt/lists/\* 0 B <missing> 2 weeks ago /bin/sh -c set -xe && echo '#!/bin/sh' >... 745 B <missing> 2 weeks ago /bin/sh -c #(nop) ADD file:5aff8c59a707833... 118 MB

Както може да се види от горните извадки, двата шаблона имат различно количество слоеве. Разликата се дължи на двата начина на изпълнение на **RUN** командата – в единия случай са две отделни, а в другия – една.

### Спиране и почистване на всички контейнери и шаблони

Можем да приведем системата си в първоначален вид, като спрем всички работещи контейнери и изтрием всички шаблони и контейнери. Това можем да постигнем като изпълним следната поредица от команди:

$ docker container stop $(docker container ls -q)

$ docker container prune

$ docker image rm $(docker image ls –q)