# Practice M2: Introduction to Docker

For this practice we will use an infrastructure like this:

Graphical user interface, application

Description automatically generated

All exercises that follow assume that we are working in an environment with both **VirtualBox** and **Vagrant** installed

For the VM we will use a **CentOS Stream 9** minimal based **Vagrant** box

*Please note, that some commands end with this* ***\*** *symbol. This is because the command is long and does not fit on one line. Written this way makes it more readable. When you type in such commands, you can omit the* ***\*** *symbol and continue with the rest of the command*

## Part 1: Docker installation

**References**

<https://docs.docker.com/engine/install/centos/>

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

**Commands**

**version, system info, container run**

### Preparation

Spin up either a **CentOS Stream** or **Debian** based virtual machine

You can do it either by importing a template or using **Vagrant**

Let us go with the second approach, as it is more manageable, and this is what we are aiming for

Open a terminal session and navigate to an empty folder of your choice

Invoke the initialization command to create an empty environment for **CentOS Stream**

**vagrant init --minimal shekeriev/centos-stream-9**

Or for **Debian**

**vagrant init --minimal shekeriev/debian-11**

Now open the **Vagrant** file with a text editor of your choice

If you are working in a graphical environment of **Windows**, **Linux**, or **macOS**, you can install the **VSCode** free editor by **Microsoft** and add one of available **Vagrant** plugins *(for example the* ***Vagrantfile Support*** *by* ***Marco Stazi*** *or* ***Vagrant*** *by* ***Baptist Benoist****)*

Now, that we are in, let us add one block, between the **config.vm.box** and **end**, to set some port forwarding and change the memory size

**config.vm.network "forwarded\_port", guest: 8080, host: 8080, auto\_correct: true**

**config.vm.provider :virtualbox do |vb|**

**vb.customize ["modifyvm", :id, "--memory", "2048"]**

**end**

Save and close the file

Return to the terminal session and start the environment

**vagrant up**

And then, once the machine is up and running, establish **ssh** session to it

**vagrant ssh**

### *Remove old installations*

*This is not directly applicable to our case, as we start fresh without existing* ***Docker*** *installation*

*Should you have old installation of* ***Docker****, it is recommended to remove it first*

#### *CentOS*

*Under* ***CentOS*** *and other* ***Red Hat****-based distributions we can execute this*

***sudo dnf remove docker docker-client docker-client-latest \***

***docker-common docker-latest docker-latest-logrotate \***

***docker-logrotate docker-selinux docker-engine-selinux docker-engine***

#### *Debian*

*Under* ***Debian****-based distributions we can execute this*

***sudo apt-get remove docker docker-engine docker.io containerd runc***

### Installation preparation

First, we must make sure that we have the appropriate repository registered

#### CentOS

Add the repository

**sudo dnf config-manager \**

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

#### Debian

First, refresh package information and install the required packages

**sudo apt-get update**

**sudo apt-get install ca-certificates curl gnupg lsb-release**

Then add the repository key (change **debian** with **ubuntu** if working on **Ubuntu**)

**curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg**

Finally, add the repository (change **debian** with **ubuntu** if working on **Ubuntu**)

**echo \**

**"deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/debian \**

**$(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null**

### Docker installation

Now, we are ready to install **Docker** itself

#### CentOS

To install **Docker**, execute the following:

**sudo dnf install docker-ce docker-ce-cli containerd.io docker-compose-plugin**

Let us start the **docker** daemon and check its status

**sudo systemctl start docker**

**systemctl status docker**

#### Debian

To install **Docker**, first we must update package information

**sudo apt-get update**

And then install the necessary packages

**sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin**

Finally, check that the service started successfully

**systemctl status docker**

### Post installation test

If the service is working normally, we can receive more information about the installed versions and the current configuration with

**sudo docker version**

**sudo docker system info**

And let us start our very first container

**sudo docker container run hello-world**

Or this one:

**sudo docker container run shekeriev/welcome-do:latest**

### Additional settings

To be able to work with **docker** without the need to use always **sudo**, we must add our user to the **docker** group

**sudo usermod -aG docker $USER**

To apply the change, we must log off and then log on back again

We can check by executing any of the previous **docker** commands, for example

**docker container run hello-world**

As a last step, we can mark the **docker** service for automatic start on boot (this is not necessary on **Debian**/**Ubuntu** as it is automatically done)

**sudo systemctl enable docker**

For more post-installation settings like remote connectivity, storage options, etc. visit the following address

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

### Clean up

Close the session to the VM

And being in the folder where we started (where we have our minimal **Vagrantfile**), execute

**vagrant destroy --force**

This will stop and delete the virtual machine

### Alternative way

Besides the way we saw, there are two other ways to install **Docker**

One of them is to download the archive and deal with everything manually

You can check more on this here: <https://docs.docker.com/engine/install/binaries/>

The other way is to use the **Get Docker** script

Is simple. Just download the script

**curl -fsSL https://get.docker.com -o get-docker.sh**

Should you want, you can do a dry run with

**DRY\_RUN=1 sh ./get-docker.sh**

Or do the actual installation with

**sudo sh get-docker.sh**

That is, it. Of course, any post-installation actions are applicable here as well

### Create a complete Vagrantfile

Let us try to write an all-in-one **Vagrantfile** that can be used to spin up a VM with **docker** installed and configured

Create a new folder and switch to it

Open a new **Vagrantfile** for editing

#### CentOS

You can copy and paste what we have in our first **Vagrantfile** and extend it or start clean

Any way, you should type the following text

Vagrant.configure("2") do |config|

    config.vm.define "docker" do |docker|

        docker.vm.box="shekeriev/centos-stream-9"

        docker.vm.hostname = "docker.do1.lab"

        docker.vm.network "private\_network", ip: "192.168.99.100"

        docker.vm.network "forwarded\_port", guest: 8080, host: 8080, auto\_correct: true

        docker.vm.provision "shell", path: "docker-setup.sh"

        docker.vm.provider :virtualbox do |vb|

            vb.customize ["modifyvm", :id, "--memory", "2048"]

      end

    end

  end

Save and close the file

Now create a new **docker-setup.sh** file

Type the following (these are the steps we did manually)

#!/bin/bash

echo "\* Add hosts ..."

echo "192.168.99.100 docker.do1.lab docker" >> /etc/hosts

echo "\* Add Docker repository ..."

dnf config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

echo "\* Install Docker ..."

dnf install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin

echo "\* Enable and start Docker ..."

systemctl enable docker

systemctl start docker

echo "\* Firewall - open port 8080 ..."

firewall-cmd --add-port=8080/tcp –permanent

firewall-cmd --reload

echo "\* Add vagrant user to docker group ..."

usermod -aG docker vagrant

Save and close the file

#### Debian

You can copy and paste what we have in our first **Vagrantfile** and extend it or start clean

Any way, you should type the following text

Vagrant.configure("2") do |config|

    config.vm.define "docker" do |docker|

        docker.vm.box="shekeriev/debian-11"

        docker.vm.hostname = "docker.do1.lab"

        docker.vm.network "private\_network", ip: "192.168.99.200"

        docker.vm.network "forwarded\_port", guest: 8080, host: 8080, auto\_correct: true

        docker.vm.provision "shell", path: "docker-setup.sh"

        docker.vm.provider :virtualbox do |vb|

            vb.customize ["modifyvm", :id, "--memory", "2048"]

      end

    end

  end

Save and close the file

Now create a new **docker-setup.sh** file

Type the following (these are the steps we did manually)

#!/bin/bash

echo "\* Add hosts ..."

echo "192.168.99.200 docker.do1.lab docker" >> /etc/hosts

echo "\* Add any prerequisites ..."

apt-get update

apt-get install -y ca-certificates curl gnupg lsb-release

echo "\* Add Docker repository and key ..."

curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/debian $(lsb\_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null

echo "\* Install Docker ..."

apt-get update

apt-get install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin

echo "\* Add vagrant user to docker group ..."

usermod -aG docker vagrant

Save and close the file

### Use the Vagrantfile

Give it a try with

**vagrant up**

If everything goes according to the plan, then as a result there should be a working **docker** machine

Once, done with the exploration, do not destroy the machine. We will be using it in the next part

## Part 2: Working with images and containers

**References**

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

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

**Commands**

**search**, **image** **pull**, **image** **ls**, **image** **rm**, **container** **run**, **container** **ls**, **container** **rm**, **container** **create**, **container** **start, container attach, container stop, container prune, container export**, **image import**, **image save**, **image load**

### Working with containers

We could search for all images that contain **ubuntu** in their name

**docker search ubuntu**

The first result in the list is with the biggest rating and it is marked as an official image

We can download it locally with

**docker image pull ubuntu**

In fact, the above command is equivalent to this one

**docker image pull ubuntu:latest**

As a result of the execution of either of the above two commands, we will have the latest version downloaded locally

Should we want a particular version, we can execute

**docker image pull ubuntu:18.04**

List with the available images can be obtained with the execution of the following command

**docker image ls**

Let us start a container out of the **ubuntu:18.04** image

**docker container run -it ubuntu:18.04**

Once, we are in the container, we can execute the following sequence of commands *(just the black portion)*

**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 pts/0 Ss 0:00 /bin/bash**

**15 pts/0 R+ 0:00 ps ax**

**root@35ac9218a880:/#**

We can do the usual stuff. For example, install a package

Let’s do it by executing the following

**apt-get update**

**apt-get install inetutils-ping**

Now, we can ping, for example the address of **softuni.bg**

**ping softuni.bg**

It is working 😊 Press **Ctrl+C** to interrupt the ping command

Should we want to temporary exit the container without stopping it, we can do it by pressing and holding the **Ctrl** key, then pressing **P** and then **Q** and finally releasing the **Ctrl** key

Alternatively, we can press **Ctrl+P** and then **Ctrl+Q**

Now, we can ask for the list of all working containers

**docker container ls**

The result should look similar to

**CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES**

**35ac9218a880 ubuntu:18.04 "/bin/bash" 8 minutes ago Up 8 minutes cocky\_fermat**

Should we want to go back to the container, we can execute

**docker container attach 35ac9218a880**

Where the **35ac9218a880** sequence is the container ID (the first column of the **docker container ls** command)

In fact, we can use even shorter string if it is unique amongst the containers on the host

For example, we can use just **35** in this case

Alternatively, we can use the value from the last column - the name of the container (**cocky\_fermat**)

We could terminate the container by executing the **exit** command

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

If we ask now for the containers with

**docker container ls**

We won’s see it, as it not running anymore (we terminated the only process in the container)

In order to see the list of all containers including the stopped ones, we can execute

**docker container ls -a**

We can create a container with custom name and without starting it with

**docker container create -it --name ubuntu-22 ubuntu /bin/bash**

Then, we can start it by executing the following command

**docker container start -ai ubuntu-22**

Let us exit the container without stopping it. We can use the **Ctrl+P** and **Ctrl+Q** key combination

Then, we can stop it with

**docker container stop ubuntu-22**

Let’s start a few more containers but this time they will run in background or detached mode

Start the first one based on the **Alpine** image (it a tiny one) using this command

**docker container run -d --name dummy1 alpine sleep 1d**

Now, start the second but this time based on the **Busybox** image (even tinier) with

**docker container run -d --name dummy2 busybox sleep 1d**

Let’s check how tiny are those images and compare them to the **Ubuntu** based images

**docker image ls**

Wow, they are tiny indeed 😊

Let’s see the list of running containers

**docker container ls**

Imagine that there were many running containers and we need a way to stop them all

We will need a way to extract just their IDs for example as a list and supply it to the stop command

A list of all running containers can be extracted with

**docker container ls -q**

Now, to combine this with the stop command, we can execute the following

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

What about the stopped containers? Can we remove them somehow? Yes, we can

Let’s ask for their list with

**docker container ls -a**

Particular stopped container or list of containers, we can remove with

**docker container rm ubuntu-22**

All stopped containers can be deleted with

**docker container prune**

Now, the list of the stopped containers should be empty

**docker container ls -a**

What about the images? How can we delete them?

Let’s check again their list with

**docker image ls**

We can remove one or more images with

**docker image rm ubuntu**

And check again

**docker image ls**

Yes, the image is gone

### Archive and transfer containers

Let us start a container out of the **Alpine** (**alpine**) image

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

Then, we can execute the following commands **ls**, **echo**, **cat**, and **exit**, as shown bellow

**/ # 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**

Now the container is stopped. Check with

**docker container ls -a**

We can start it again with

**docker container start -ai my-alpine**

Check if the file is there

**cat readme.txt**

Stop the container again

**exit**

Now, we can save it as **tar** archive and then transfer it to another **docker** server and import it there

As we do not have another server, we will use our current

The export command is the following

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

Check the size of the resulting file

**ls -lh**

Now, check the current list of locally available images

**docker image ls**

The import process requires additional options because during the export the settings for **CMD** and **ENV** are lost

The actual import command may look like

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

In fact, we are importing it as an image which later will be used to spin new container

Check again the list of locally available images

**docker image ls**

Now, we can start a container out of our new image and check if our custom file is there

**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**

### Archive and transfer images

We can archive images directly and move them between **docker** servers

Remember the **Busybox** image that we downloaded earlier?

Let’s archive it

**docker image save -o busybox.tar busybox**

In order to try the import from archive, first we must remove the existing image

**docker image rm busybox**

Confirm that the image is gone

**docker image ls**

And the import the image from the archive

**docker image load -i busybox.tar**

Now, confirm that we have the image, and we can use it

**docker image ls**

We can create and run a container out of it (we can even make the container disappear once stopped)

**docker container run -it --rm busybox**

Explore it and then execute the **exit** command to stop it

Check that the container is absent as promised

**docker container ls -a**

Yes, it is gone 😊

### Working with different registry

Okay, we saw that **Docker Hub** is the default registry (**docker info**)

But how we can change it at least temporary, just for a single container?

Check this address: <https://hub.zahariev.pro/v2/_catalog>

Two of the images here are suitable for test – **alpine** and **k8s-environ**

Should we want to see what tags are available for the **k8s-environ** image, we can execute

**curl -X GET https://hub.zahariev.pro/v2/k8s-environ/tags/list**

The first is just a local (for the registry) copy of the publicly available **alpine** image

Let’s try the second one

We must execute the following command

**docker container run -d --name other -p 8080:80 hub.zahariev.pro/k8s-environ**

So, we are prefixing the image with the registry, and this is enough

Check the list of local images

**docker image ls**

And then the containers

**docker container ls**

Open a browser tab and navigate to <http://localhost:8080>

The application is working

Remove the container

**docker container rm --force other**

## Part 3: Creating images

**Commands**

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

### Create image from container

Let us create and run a container out of the **Ubuntu (ubuntu)** image first

**docker container run -it --name my-ubuntu ubuntu**

Apply the **Ctrl+P** and **Ctrl+Q** combination to close the connection without terminating the container

We can check if the container is still running, but let us apply a filter this time

**docker container ls -f name=my-ubuntu**

The process of image creation out of a container (even a running one) is very simple

The actual command is this

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

Let’s first check if the container is still running

**docker container ls -f name=my-ubuntu**

And then check if our new image is there

**docker image ls new-ubuntu**

Let us test if we can create and run a container based on it

This time we will launch the container with one additional option that will cause the container to be deleted automatically once stopped

**docker container run -it --rm new-ubuntu**

Yes, we can, and it is working. Now, close the session

**root@fc0bc2b9b8ab:/# exit**

If we check the list of stopped containers, we will not find it there as we used the **--rm** option again

**docker container ls -a**

### Create image with heredoc

Have you heard about heredoc?

It is technique for building simple multi-line text documents on the fly

It can be used to create simple **Docker** images as well by creating the **Dockerfile** on the fly

Let us execute the following command

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

**FROM alpine**

**RUN apk --no-cache add htop**

**EOF**

Do we have the new image listed?

**docker image ls**

Now, we can launch a container based on our new image

**docker container run -it alp-htop**

Check that we can use the added package

**/ # htop**

And then close the session

**/ # exit**

### Create image from Dockerfile

We will create two identical from user point of view images using the **Dockerfile** technique

Let us create a folder **nginx-1** and change to it

**mkdir nginx-1 && cd nginx-1**

Now, open an empty **Dockerfile** for editing (use **nano** if you like)

**vi Dockerfile**

Enter the following text

**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**

Save and close the file

Next step is to generate or build the image

Image building is done with

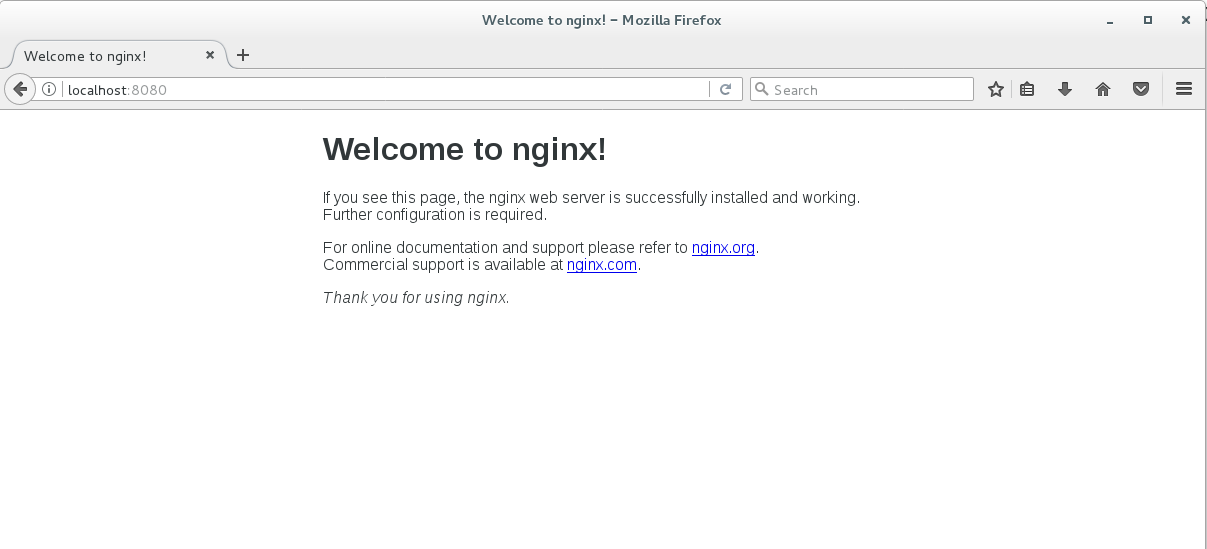
**docker image build -t nginx-1 .**

Once the image is built, spin up a container based on it with

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

Now, we can open a browser on the host and navigate to <http://localhost:8080>

If all went according to plan, we must see output similar to this



We can stop it with the following command

**docker container stop web-1**

Now, change to the upper folder

**cd ..**

Create another new folder and switch to it

**mkdir nginx-2 && cd nginx-2**

Open a new empty file for editing (use **nano** if you like)

**vi Dockerfile**

Enter the following text

**FROM ubuntu**

**LABEL maintainer="SoftUni Student"**

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

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

**EXPOSE 80**

Save and close the file

Build the new image with

**docker image build -t nginx-2 .**

Spin up a container out of the new image

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

If we open again a browser tab and visit <http://localhost:8080> we will see the same result, as expected

Now stop it with the following command

**docker container stop web-2**

So, for us, the result is one and the same but for **Docker** this is not the case

Let us inspect the first image with

**docker image history nginx-1**

The output should be similar to

**IMAGE CREATED CREATED BY SIZE COMMENT**

**b16421e0b876 3 minutes ago /bin/sh -c #(nop) EXPOSE 80 0B**

**3d5c566eb9a8 3 minutes ago /bin/sh -c #(nop) ENTRYPOINT ["/usr/sbin/ng… 0B**

**96a3afaaa16e 3 minutes ago /bin/sh -c apt-get install -y nginx 59.2MB**

**a96d134e620d 4 minutes ago /bin/sh -c apt-get update 33.7MB**

**92309843847a 4 minutes ago /bin/sh -c #(nop) LABEL maintainer=SoftUni … 0B**

**54c9d81cbb44 7 days ago /bin/sh -c #(nop) CMD ["bash"] 0B**

**<missing> 7 days ago /bin/sh -c #(nop) ADD file:3ccf747d646089ed7… 72.8MB**

And then the second one

**docker image history nginx-2**

The output should be similar to

**IMAGE CREATED CREATED BY SIZE COMMENT**

**335de904496b About a minute ago /bin/sh -c #(nop) EXPOSE 80 0B**

**ff29f1a69e9c About a minute ago /bin/sh -c #(nop) ENTRYPOINT ["/usr/sbin/ng… 0B**

**ec1107f8aad4 About a minute ago /bin/sh -c apt-get update && apt-get install… 92.8MB**

**92309843847a 4 minutes ago /bin/sh -c #(nop) LABEL maintainer=SoftUni … 0B**

**54c9d81cbb44 7 days ago /bin/sh -c #(nop) CMD ["bash"] 0B**

**<missing> 7 days ago /bin/sh -c #(nop) ADD file:3ccf747d646089ed7… 72.8MB**

As it can be seen, both images have different number of layers (seven vs. six)

The difference is because of the different number of **RUN** commands in the two **Dockerfile**s

In any case, when we check their size

**docker image ls**

It is the same. So, can we go one step further? Sure, we can

Let’s go for a third run

Now, change to the upper folder

**cd ..**

Create another new folder and switch to it

**mkdir nginx-3 && cd nginx-3**

Open a new empty file for editing (use **nano** if you like)

**vi Dockerfile**

Enter the following text

**FROM ubuntu**

**LABEL maintainer="SoftUni Student"**

**RUN apt-get update && apt-get install -y nginx && rm -rf /var/lib/apt/lists/\***

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

**EXPOSE 80**

Save and close the file

Build the new image with

**docker image build -t nginx-3 .**

Spin up a container out of the new image

**docker container run -d -p 8080:80 --name web-3 nginx-3**

If we open again a browser tab and visit <http://localhost:8080> we will see the same result, as expected

Now stop it with the following command

**docker container stop web-3**

Again, for us, the result is one and the same but for **Docker** this is not the case

Let us inspect this one as well

**docker image history nginx-3**

The output should be similar to

**IMAGE CREATED CREATED BY SIZE COMMENT**

**f9e661a365e7 2 minutes ago /bin/sh -c #(nop) EXPOSE 80 0B**

**b85ce1089c69 2 minutes ago /bin/sh -c #(nop) ENTRYPOINT ["/usr/sbin/ng… 0B**

**0b5861ca838f 2 minutes ago /bin/sh -c apt-get update && apt-get install… 92.8MB**

**92309843847a 19 minutes ago /bin/sh -c #(nop) LABEL maintainer=SoftUni … 0B**

**54c9d81cbb44 7 days ago /bin/sh -c #(nop) CMD ["bash"] 0B**

**<missing> 7 days ago /bin/sh -c #(nop) ADD file:3ccf747d646089ed7… 72.8MB**

Again, six layers as with the attempt no.2

Let’s check the size of the three images again

**docker image ls nginx-\***

Okay, we managed to save 30+ MB 😊

### Adding content during build

Now, change to the upper folder

**cd ..**

Create another new folder and switch to it

**mkdir -p nginx-4/web && cd nginx-4**

Create a custom index.html file

**echo 'It works! :)' > web/index.html**

Wait, where shall we copy this file (or the whole folder) in the image? Hm, let’s start again the last container (**web-3**)

**docker container start web-3**

Now, how can we enter? Perhaps, the attach command? Hm, no, it won’t work this time

We can execute a whole new process in the container with the following command

**docker container exec -it web-3 bash**

Check the configuration of the **NGINX** server with

**cat /etc/nginx/nginx.conf**

And then the file for the default web site

**cat /etc/nginx/sites-enabled/default**

Aha, the files are read from **/var/www/html**

So, we should place our content there

Close the session to the container

**exit**

Long way to check something simple. Is there another way?

Yes, there is. In fact, it is just a modification of what we did

Try this command

**docker container exec -it web-3 cat /etc/nginx/sites-enabled/default**

Okay, finally we can stop the container

**docker container stop web-3**

And continue with the building process

Now, open a new empty file for editing (use **nano** if you like)

**vi Dockerfile**

Enter the following text

**FROM ubuntu**

**LABEL maintainer="SoftUni Student"**

**RUN apt-get update && apt-get install -y nginx && rm -rf /var/lib/apt/lists/\***

**COPY web/ /var/www/html**

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

**EXPOSE 80**

Save and close the file

Build the new image with

**docker image build -t nginx-4 .**

Spin up a container out of the new image

**docker container run -d -p 8080:80 --name web-4 nginx-4**

If we open again a browser tab and visit <http://localhost:8080> we will see a different result 😊

Now stop it with the following command

**docker container stop web-4**

### Publishing an image

Let’s assume that we are glad by what we accomplished so far, and we want to publish our latest image (**nginx-4**)

First, we must make sure that we have access to a registry

The easiest option is to register at **Docker Hub** here: <https://hub.docker.com/>

Then, we must login and authenticate our client to the registry

**docker login**

Enter the username and the password you used for **Docker Hub**

Now, tag the image appropriately

**docker image tag nginx-4 <repository>/super-nginx:demo**

Check that it appears in the list

**docker image ls**

Pay attention to their size an ID. They are in fact two names or records for the same image

Now, push it to the registry

**docker image push <repository>/super-nginx:demo**

Return to the browser and check that you see your newly published image

We can safely delete the local copy both for the original image (**nginx-4**) and the published one

**docker image rm nginx-4**

**docker image rm <repository>/super-nginx:demo**

Check that they are gone

**docker image ls**

Now, start a container out of the published image

**docker container run -d -p 8080:80 <repository>/super-nginx:demo**

If we open again a browser tab and visit <http://localhost:8080> we will see it 😊

### ENTRYPOINT and CMD

Let us see how **ENTRYPOINT** and **CMD** work together

Return to the upper folder

**cd ..**

Create a new one

**mkdir entry-cmd && cd entry-cmd**

Open an empty **Dockerfile** for editing (or use **nano** if you like)

**vi Dockerfile**

Enter the following text

**FROM busybox**

**LABEL description="ENTRYPOINT vs CMD demo" maintainer="SoftUni Student"**

**ENTRYPOINT ["ping", "-c", "4"]**

**CMD ["www.softuni.bg"]**

Save and close the file

Build the image with

**docker image build -t pinger .**

Create and run a new container based on the image with

**docker container run --name p1 pinger**

And now run a second one but this time with modified command

**docker container run --name p2 pinger tuionui.com**

This way, by using both **ETRYPOINT** and **CMD** in their **exec** form, we got an image that when used for containers can be controlled by appending a parameter

### ENTRYPOINT and CMD extended

We can go even further

Let’s first remove all stopped containers

**docker container prune**

Now, let’s create a **pinger.sh** bash script with the following content

**#!/bin/bash**

**docker container run --rm pinger $1**

Save and close the file

Then set its execute permissions

**chmod +x pinger.sh**

And test it without arguments

**./pinger.sh**

Now, test it with an argument

**./pinger.sh abv.bg**

It acts the same just the command is shorter

Let’s check the stopped containers

**docker container ls -a**

No traces of the last two executions

This way, we can create and use container-based utilities

### Clean up

We can return our system to a clean state by removing all containers and images

First, we can stop all running containers

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

Then, we can remove all stopped containers

**docker container prune**

And finally, we can remove all images

**docker image rm $(docker image ls -q)**