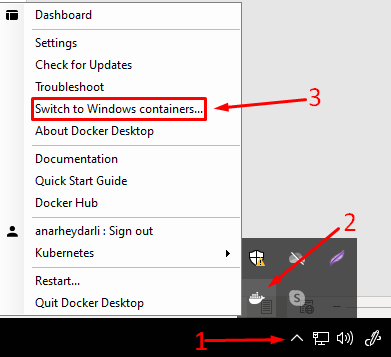
**Docker Tasks**

Task requirements:

**Make sure you already have below requirements for docker task completion.**

* Windows 10 OS
* Installed [Docker Desktop](https://docs.docker.com/docker-for-windows/install/) for Windows 10
* [Switched to **Linux** Containers](https://docs.docker.com/docker-for-windows/#switch-between-windows-and-linux-containers)



* Login to the hub.docker.com

ENTRYPOINT and CMD in Dockerfile.

**Briefly:**

* An [ENTRYPOINT](https://docs.docker.com/engine/reference/builder/#entrypoint) allows you to configure a container that will run as an executable
* The main purpose of a [CMD](https://docs.docker.com/engine/reference/builder/#cmd) is to provide defaults for an executing container
* If you would like your container to run the same executable every time, then you should consider using ENTRYPOINT in combination with CMD
* ENTRYPOINT/CMD has 2 forms:
  + **exec** form: ["echo", "hello", "world"] - preferred form
  + **shell** form: echo hello world - supports ENV Vars resolution

**Understand how CMD and ENTRYPOINT work**

Both CMD and ENTRYPOINT instructions define what command gets executed when running a container. There are few rules that describe their co-operation.

1. Dockerfile should specify at least one of [CMD](https://docs.docker.com/engine/reference/builder/#cmd) or [ENTRYPOINT](https://docs.docker.com/engine/reference/builder/#entrypoint) commands.
2. [ENTRYPOINT](https://docs.docker.com/engine/reference/builder/#entrypoint) should be defined when using the container as an executable.
3. [CMD](https://docs.docker.com/engine/reference/builder/#cmd) should be used as a way of defining default arguments for an ENTRYPOINT command or for executing an ad-hoc command in a container.
4. [CMD](https://docs.docker.com/engine/reference/builder/#cmd) will be overridden when running the container with alternative arguments.

**Examples:**

**FROM** ubuntu

**ENTRYPOINT** ["top", "-b"]

**CMD** ["-c"]

**FROM** debian:stable

**RUN** apt-get update && apt-get install -y --force-yes apache2

**EXPOSE** 80 443

**VOLUME** ["/var/www", "/var/log/apache2", "/etc/apache2"]

**ENTRYPOINT** ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]

Task 1

**Expected result:**

Deploy “ping” application in docker container locally and be able to execute commands.

**Steps to verify the result:**

Without passing an argument to the image it should display the help menu of the ping command. Otherwise, it should ping domain addresses you specify.   
(To see the difference between ENTRYPOINT and CMD perform the next actions)

* docker run aheydarli/ping
* docker run aheydarli/ping google.com
* docker run aheydarli/ping -c5 google.com

**Task:**

Create project folder C:\docker-labs\task1\.

**NOTE: All with the lowercase name image (repository name) as the first letter of your name, and your full surname. In my case, it is: aheydarli**

Create Dockerfile and build image with the requirements as below:

* Image name: aheydarli/ping
* Base image: alpine
* ENTRYPOINT: ping
* CMD: --help

**Documentation:**

* <https://docs.docker.com/engine/reference/builder/>
* <https://docs.docker.com/engine/reference/commandline/build/>
* <https://docs.docker.com/engine/reference/commandline/image/>
* <https://docs.docker.com/engine/reference/commandline/images/>
* <https://docs.docker.com/engine/reference/builder/#entrypoint>
* <https://docs.docker.com/engine/reference/builder/#cmd>

Task 2

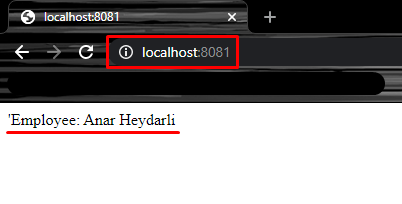
**Expected result:**

Build and deploy a static web application to localhost with docker and be able to open it locally.

The page should show Author parameter value.

**Steps to verify the result:**

Open your browser and go the URL **localhost:8081**



**Task:**Create project folder C:\docker-labs\task2\.

Place all configuration files into this folder:

* Dockerfile
* index.html

**Dockerfile Details**:

* [Base image](https://docs.docker.com/engine/reference/builder/#from): centos:7
* [Label](https://docs.docker.com/engine/reference/builder/#label): AUTHOR={Your Name and Surname}
* Make sure all necessary packages (httpd, web-assets-httpd) installed, [clean yum cache](https://www.hivelocity.net/kb/how-to-clear-the-yum-cache/)
* [Copy](https://docs.docker.com/engine/reference/builder/#copy) (or [Add](https://docs.docker.com/engine/reference/builder/#add)) index.html to httpd’s default location (/var/www/html/)
* Httpd home page should contain 'Employee: {Your Name and Surname}' line
* [Expose](https://docs.docker.com/engine/reference/builder/#expose) 80th port
* [Main container process](https://docs.docker.com/engine/reference/builder/#cmd) should start httpd in [foreground mode](https://www.techietown.info/2016/10/how-to-start-httpdapache-in-foreground/)

**Image Requirements**:

* Build an image with [docker build](https://docs.docker.com/engine/reference/commandline/build/) command.
* Image name: myweb:0.1

**Container Requirements**:

* Run your image with docker run -d -p 8081:80 myweb:0.1 command

**Documentation:**<https://docs.docker.com/engine/reference/builder/>  
<https://docs.docker.com/engine/reference/commandline/build/>  
<https://docs.docker.com/engine/reference/commandline/image/>  
<https://docs.docker.com/engine/reference/commandline/images/>

Task 3

**Expected result:**

Build 2 docker images with Java applications with different Java SDK versions.

**Steps to verify the result:**

C:\docker-labs\task3> docker run --rm c8j11 java -version

openjdk version "11.0.8" 2020-07-14 LTS

OpenJDK Runtime Environment 18.9 (build 11.0.8+10-LTS)

OpenJDK 64-Bit Server VM 18.9 (build 11.0.8+10-LTS, mixed mode, sharing)

C:\docker-labs\task3> docker run --rm c7j180 java -version

openjdk version "1.8.0\_252"

OpenJDK Runtime Environment (build 1.8.0\_252-b09)

OpenJDK 64-Bit Server VM (build 25.252-b09, mixed mode)

**Task:**

Create Dockerfile in C:\docker-labs\task3\with the requirements below.

Dockerfile has two (ARG) arguments:

* **CENTOS\_IMAGE** - base image without default value
* **JAVA\_VERSION** - openjdk to be installed, default to 11

Using this Dockerfile build two images (just by providing --build-arg <varname>=<value>):

**First Image:**

* Image name: c8j11
* CENTOS\_VERSION: 8
* JAVA\_VERSION: 11

**Second Image:**

* Image name: c7j180
* CENTOS\_VERSION: 7
* JAVA\_VERSION: 1.8.0

**Hint**:  
 To install openjdk N (where N stands for its version) on CentOS use following command:

yum install java-N-openjdk

**Documentation:**   
<https://docs.docker.com/engine/reference/builder/#arg>  
<https://docs.docker.com/engine/reference/builder/>  
<https://docs.docker.com/engine/reference/commandline/build/>  
<https://docs.docker.com/engine/reference/commandline/image/>  
<https://docs.docker.com/engine/reference/commandline/images/>

Task 4

**Expected result:**

...

**Steps to verify the result:**

Perform a few requests and check container logs. You should see the following output:

C:\docker-labs\task4> (Invoke-WebRequest http://localhost:18084).Content

hostname: 17cf4561310b

ip address: 172.17.0.4

C:\docker-labs\task4> (Invoke-WebRequest http://localhost:18084).Content

hostname: 17cf4561310b

ip address: 172.17.0.4

C:\docker-labs\task4> (Invoke-WebRequest http://localhost:18084).Content

hostname: 17cf4561310b

ip address: 172.17.0.4

C:\docker-labs\task4> docker logs go\_multi

2021/01/03 14:27:43 Getting request from 172.17.0.1:35746

2021/01/03 14:29:21 Getting request from 172.17.0.1:35746

2021/01/03 14:29:22 Getting request from 172.17.0.1:35746

Now just compare sizes of go\_simple and go\_multi images! Incredible!

C:\docker-labs\task4> docker image ls | Select-String ^go\_

go\_multi latest 1cf3116028e0 36 seconds ago 5.65MB

go\_simple latest af2fc8af024a 2 hours ago 879MB

Create Dockerfile in C:\docker-labs\task4\with the requirements below.

**Task:**

In the above directory create the following files with the below contents:

**web.go:**

package main

import (

"fmt"

"log"

"net/http"

"os"

"net"

)

func handler(w http.ResponseWriter, r \*http.Request) {

ip, port, \_ := net.SplitHostPort(r.RemoteAddr)

log.Printf("Getting request from %s:%s", ip, port)

hostname, \_ := os.Hostname()

ipaddress, \_ := net.LookupHost(hostname)

fmt.Fprintf(w, "hostname: %s\nip address: %s\n", hostname, ipaddress[0])

}

func main() {

http.HandleFunc("/", handler)

log.Fatal(http.ListenAndServe(":8080", nil))

}

**go\_multi.Dockerfile:**

FROM ...

...

## Compiling \*.go file

RUN go build -a ./web.go

FROM scratch

...

## Define container process

CMD ["./web"]

**go\_simple.Dockerfile:**

FROM golang

ENV GOOS=linux

ENV GOARCH=386

COPY web.go .

## Compiling \*.go file

RUN go build -a ./web.go

## Define container process

CMD ["./web"]

You are given Dockerfiles and web.go (simple webserver written on golang) files above place them all in C:\docker-labs\task4\ directory.

**Build C:\docker-labs\task4\go\_simple.Dockerfile with tag go\_simple and run it:**

docker build ...

docker run -d -p 8084:8080 --name=go\_simple go\_simple

Make a few requests and examine logs of container. You should see the following output:

C:\docker-labs\task4> (Invoke-WebRequest http://localhost:8084).Content

hostname: f08efe950b6c

ip address: 172.17.0.3

C:\docker-labs\task4> (Invoke-WebRequest http://localhost:8084).Content

hostname: f08efe950b6c

ip address: 172.17.0.3

C:\docker-labs\task4> (Invoke-WebRequest http://localhost:8084).Content

hostname: f08efe950b6c

ip address: 172.17.0.3

C:\docker-labs\task4> docker logs go\_simple

2021/01/03 14:19:13 Getting request from 172.17.0.1:45576

2021/01/03 14:19:18 Getting request from 172.17.0.1:45576

2021/01/03 14:19:20 Getting request from 172.17.0.1:45576

One of advantages of go applications is compilation opportunity. We can create a binary for various platforms and run it without any supplementary environments such as java.  
Moreover we can even use scratch as base image for the 2nd stage. We will get the container with only web server application (there will be no bash even!).

Let’s do this!

**Task:**

Edit **C:\docker-labs\task4\go\_multi.Dockerfile** and build **go\_multi** image:

The 1st stage (golang image) is about creating binary (executable) file web (go build -a /web.go)

The 2nd build stage:

* copy artifact (web executable) from the 1st stage;
* set 8080 port as exposed;
* CMD should operate with web binary file

**Build edited C:\docker-labs\task4\go\_multi.Dockerfile  with tag go\_multi and run it:**  
docker run -d -p 18084:8080 --name=go\_multi go\_multi

**Documentation:**

* <https://docs.docker.com/engine/reference/builder/>
* <https://medium.com/@tonistiigi/advanced-multi-stage-build-patterns-6f741b852fae>
* <https://docs.docker.com/engine/reference/commandline/build/>
* <https://docs.docker.com/engine/reference/commandline/image/>

Task 5

**Expected result:**

...

**Steps to verify the result:**

Pulling the image from the Docker hub (Optionally):

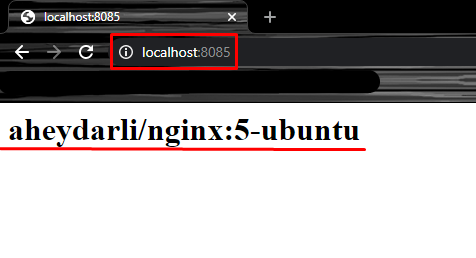
docker pull aheydarli/nginx:5-ubuntu

Run web service from your image:

docker run -d --name nginx-ubuntu-5 -p 8085:80 `

aheydarli/nginx:5-ubuntu

Check service in browser (example output):



Or using CLI:

C:\docker-labs\task5> (Invoke-WebRequest http://localhost:8085).Content  
aheydarli/nginx:5-ubuntu

Create project folder C:\docker-labs\task5\.

**NOTE: All with the lowercase name image (repository name) as the first letter of your name, and your full surname. In my case, it is: aheydarli**

**Task:**

Create Dockerfile. Build Custom Docker Images. Push it into your registry on Docker Hub. Run Container and Check that it’s working as expected.

docker build -t aheydarli/nginx:5-ubuntu .

# Optionally

docker push aheydarli/nginx:5-ubuntu

**Build:**

* Base image: ubuntu
* Image name: aheydarli/nginx:5-ubuntu
* **Nginx** web server should be installed
* Image should have 80 port defined as “exposed”
* **Nginx** web server should return image name (aheydarli/nginx:5-ubuntu) as the only content on home page.

**Runtime:**

* Container name: nginx-ubuntu-5
* Host/Container port: 8085 -> 80

**Hint:**

To find default index.html location, search for “include” statement in /etc/nginx/nginx.conf

**Documentation:**

* <https://docs.docker.com/engine/reference/builder/>
* <https://docs.docker.com/engine/reference/commandline/build/>
* <https://docs.docker.com/engine/reference/commandline/image/>
* <https://docs.docker.com/engine/reference/commandline/images/>

Task 6

**Expected result:**

...

**Steps to verify the result:**

Pulling the image from the Docker hub (Optionally):

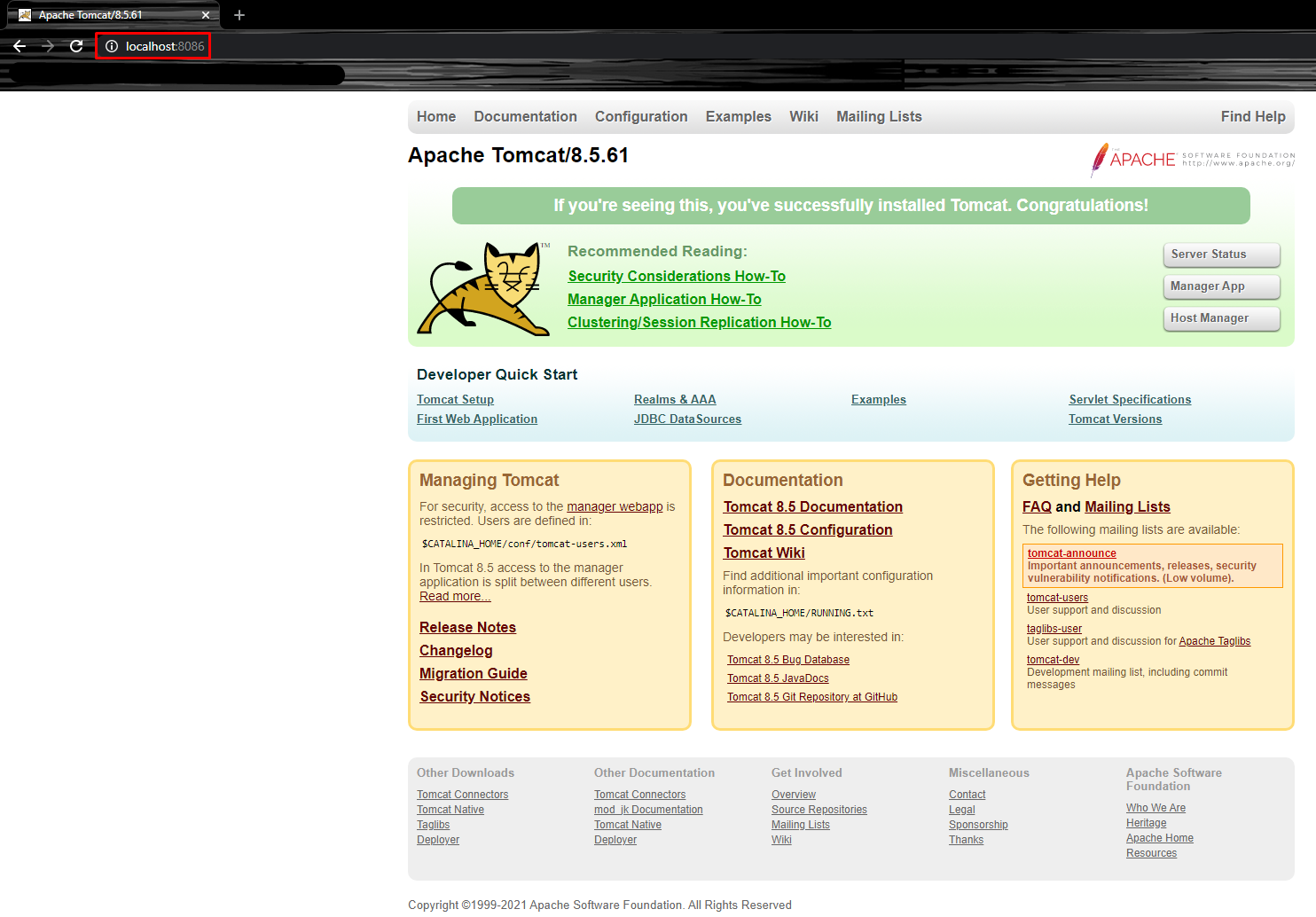
docker pull aheydarli/tomcat:6-centos

Run tomcat service from your image:

docker run -d --name tomcat-centos-6 -p 8086:8080 `

aheydarli/tomcat:6-centos

Check service in browser (example output):



Or using CLI:

C:\docker-labs\task6> (Invoke-WebRequest http://localhost:8086).Content

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>Apache Tomcat/8.5.61</title>

…………

Create project folder C:\docker-labs\task6\.

**NOTE: All with the lowercase name image (repository name) as the first letter of your name, and your full surname. In my case, it is: aheydarli**

**Task:**

Create Dockerfile. Build Custom Docker Images. Push it into your registry on Docker Hub. Run Container and Check that it’s working as expected.

docker build -t aheydarli/tomcat:6-centos .

# Optionally

docker push aheydarli/tomcat:6-centos

**Build:**

* Base image: centos:7
* Image name: aheydarli/tomcat:6-centos
* Image should have 8080 port defined as “exposed”
* **Tomcat** app server should be installed and configured to run as the main caontainer’s process

**Runtime:**

* Container name: aheydarli/tomcat:6-centos
* Host/Container port: 8086 -> 8080

**Hint:**

On CentOS you should consider installing tomcat-webapps along with tomcat package.

**Documentation:**

* <https://docs.docker.com/engine/reference/builder/>
* <https://docs.docker.com/engine/reference/commandline/build/>
* <https://docs.docker.com/engine/reference/commandline/image/>
* <https://docs.docker.com/engine/reference/commandline/images/>

Task 7  
  
**Expected result:**

...

**Steps to verify the result:**

Inspect statuses of created containers.

restarter\_2 will be exited after 7 restarts, however, restarter\_1 will be continuously restarted.

C:\docker-labs\task7> docker container ls -a `

--format "table {{.ID}}\t{{.Names}}\t{{.Status}}" `

| Select-String NAMES,restarter\_

CONTAINER ID NAMES STATUS

97c04db01d89 restarter\_2 Exited (1) 3 minutes ago

de7bde64e544 restarter\_1 Restarting (0) 26 seconds ago

Create project folder C:\docker-labs\task7\.

This task should be completed with the docker run commands.

After formulating the command save it to any file in the above directory.

**Let’s consider following problems:**

* What happens if the main process inside container fails (exits) accidently?
* How can we make sure that docker daemon restarts container automatically?
* Or, if the host reboots, how can we bring all the containers into running state back?
* How to tell Docker daemon not to restart container if it constantly fails?

We’re going to simulate the situation where our container exits in a few seconds after its start and learn docker’s restarting policies.

**Task:**

Create two containers with the commands which work for a short period of time.  
Run these containers so that:

* The 1st one will be restarted regardless of the exit status
* The 2nd one will be restarted only if the container exits with a non-zero exit status and has 7 restart retries

C:\docker-labs\task7> docker container run --help | Select-String `

'--(restart|name)', 'container exits'

--name string Assign a name to the container

--restart string Restart policy to apply when a

container exits (default "no")

* **Container #1**:
  + Runs in **detach** mode
  + Should restart regardless of the exit status
  + Container name: restarter\_1
  + Image name: busybox
  + Command: sleep 3
* **Container #2**:
  + Should run in **detach** mode
  + Should restart only if the container exits with a non-zero exit status
  + Should have 7 restart retries
  + Container name: restarter\_2
  + Image name: busybox
  + Command: sleep -3

**Documentation:**

* <https://docs.docker.com/engine/reference/run/>
* <https://docs.docker.com/config/containers/start-containers-automatically/>
* <https://www.getconvey.com/blog/docker-restart-policies/>

Task 8

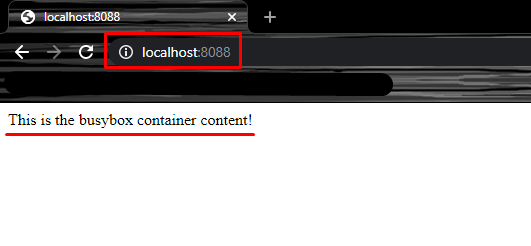
**Expected result:**

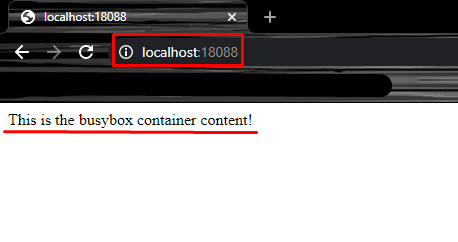
...

**Steps to verify the result:**

C:\docker-labs\task8> (docker inspect html\_data | ConvertFrom-Json).Mounts.Source

C:\docker-labs\task8\index.html

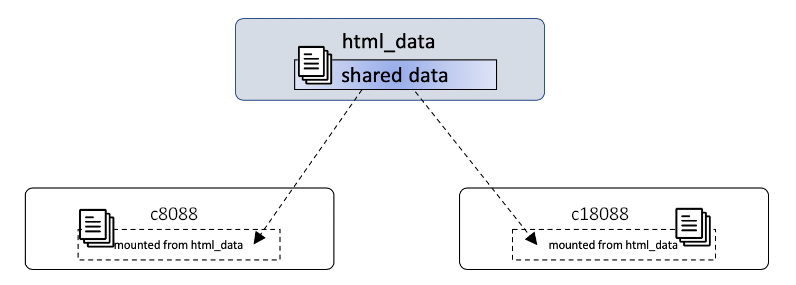




Create project folder C:\docker-labs\task8\.

This task should be completed with the docker run commands.

After formulating the command save it to any file in the above directory.



C:\docker-labs\task8> docker container run --help `

| Select-String volumes -Context 0,1

> --volumes-from list Mount volumes from the specified

container(s)

**Task:**

Run containers with the requirements below:

* **html\_data** container:
  + should run in detached mode;
  + should have name: html\_data;
  + should mount C:\docker-labs\task8\index.html file from the host to /usr/share/nginx/html/index.html into the container;
  + Insert following text “This is the busybox container content!” to your html file.
  + used image: busybox;
* **Web Container #1**:
  + container should run in detached mode;
  + container should expose 80 container port to c8088 host port;
  + container should use volume from html\_data container;
  + container name: c8088;
  + used image: nginx.
* **Web container #2**:
  + container should run in detached mode;
  + container should expose 80 container port to c18088 host port;
  + container should use volume from html\_data container;
  + container name: c18088;
  + used image: nginx.

**Documentation:**

* <https://docs.docker.com/storage/volumes/>

Task 9

**Expected result:**

...

**Steps to verify the result:**

C:\docker-labs\task9> docker container exec -it nginx-aheydarli-bridge `

sh -c "apt-get update && apt-get install -y iputils-ping"

C:\docker-labs\task9> docker container exec -it nginx-aheydarli-bridge `

ping -c 3 tomcat-aheydarli-bridge

PING tomcat-aheydarli-bridge (123.45.1.3) 56(84) bytes of data.

64 bytes from tomcat-aheydarli-bridge.aheydarli-bridge (123.45.1.3): icmp\_seq=1 ttl=64 time=0.161 ms

64 bytes from tomcat-aheydarli-bridge.aheydarli-bridge (123.45.1.3): icmp\_seq=2 ttl=64 time=0.083 ms

64 bytes from tomcat-aheydarli-bridge.aheydarli-bridge (123.45.1.3):

**OR**

C:\docker-labs\task9> docker network inspect aheydarli-bridge `

| Select-String Containers -Context 0,15

> "Containers": {

"554b717e1f7abcc001d289eab4322835352cd495612ad9978ede427373d09103": {

"Name": "tomcat-aheydarli-bridge",

"EndpointID": "53621ac885eeb61e5c6007afca748d9dabbf4e70acb2b86f4ced933e0750e023",

"MacAddress": "02:42:7b:2d:01:03",

"IPv4Address": "123.45.1.3/24",

"IPv6Address": ""

}, "ce3b11974579bc9e88bf2413dd74836e2888a385cbd4fd82593093a459846b71": {

"Name": "nginx-aheydarli-bridge",

"EndpointID": "ceab2200fda56d5ba1188530330e823d008a47379609d1e1e4413dd2dc750a95",

"MacAddress": "02:42:7b:2d:01:02",

"IPv4Address": "123.45.1.2/24",

"IPv6Address": ""

}

},

**NOTE: Change container name, network name and their labels according to your name and surname. In my case, it is: aheydarli** and **Anar Heydarli**

Check the links on the bottom and create custom network with the following requirements:

**Task:**

**Parameters:**

* **Network name**: aheydarli-bridge
* **Driver**: bridge
* **Subnet**: 123.45.1.0/24
* **IP Range**: 123.45.1.0/24
* **Label**: createdby=Anar\_Heydarli

**Hint:**

You can check all configuration parameters by inspecting this network

Once, you created aheydarli-bridge network… Let’s use it!

Run Nginx and Tomcat in Custom Network, specify custom label

 You don’t need to configure proxy on nginx to tomcat.

**Requirements:**

**Container #1**:

* container should run in detached mode
* container name: nginx-aheydarli-bridge
* container should be attached to the aheydarli-bridge network
* container should have the same label as aheydarli-bridge network
* image name: nginx

**Container #2**:

* container should run in detached mode
* container name: tomcat-aheydarli-bridge
* container should be attached to the aheydarli-bridge network
* container should have the same label as aheydarli-bridge network
* image name: tomcat

Inspect the aheydarli-bridge network and answer the question below.

**Documentation:**

* <https://docs.docker.com/network/bridge/#differences-between-user-defined-bridges-and-the-default-bridge>
* <https://docs.docker.com/v17.09/engine/userguide/networking/#default-networks>
* <https://docs.docker.com/engine/reference/commandline/network_create/>
* <https://docs.docker.com/engine/reference/commandline/network/>
* <https://docs.docker.com/v17.09/engine/userguide/networking/#default-networks>
* <https://docs.docker.com/engine/reference/commandline/network/>

Task 10

**Expected result:**

...

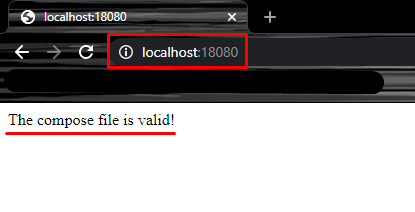
**Steps to verify the result:**

C:\docker-labs\task10> docker-compose ps

Name Command State Ports

----------------------------------------------------------------------------------------------------

nginx\_web /docker-entrypoint.sh nginx ... Up 0.0.0.0:50000->50000/tcp, 0.0.0.0:18080->80/tcp



You’re given the following command:

docker run -d `

--env-file C:\docker-labs\task10\nginx\_env `

--publish 18080:80 `

--publish 50000:50000 `

--volume C:\docker-labs\task10\index.html:/usr/share/nginx/html/index.html `

--name nginx\_web `

nginx

**Task:**

* Create docker-compose.yml file and reproduce **all options** from the command above, do follow the same names.
* Insert following text “The compose file is valid!” to your html file for nginx.
* Start the stack (docker-compose up -d)

**Documentation:**

* <https://docs.docker.com/compose/>
* <https://docs.docker.com/compose/compose-file/>
* <https://docs.docker.com/compose/reference/overview/>

Clean up resources:

C:\> docker-compose -f C:\docker\task10\docker-compose.yml down

C:\> docker container ls -q | ForEach-Object { docker container stop $\_ }

C:\> docker system prune -a

C:\> docker volume prune -f