**Introduction**

Without docker, every team member should have, for every shared application or service, the same version, configuration, environment variables, etc…

Container runs our application

Images are like blueprints for containers

Container is an isolated process, no matter what node version installed in the computer

Images are made up from several layers.

All the layers built on the parent/base layer

**Installation**

* Install Linux on Windows with WSL (Windows Subsystem for Linux)
* Prerequisites: You must be running Windows 10 version 2004 and higher (Build 19041 and higher) or Windows 11
* Run winver in your cmd to verify your Windows version
* Manual installation steps for older versions of WSL: https://learn.microsoft.com/en-us/windows/wsl/install-manual (in step 6, choose Ubuntu <version>> LTS)
* search "ubuntu" when you will click on it, you should have unix terminal opened
* If you got error "please enable the virtual machine platform windows feature and ensure virtualization is enabled in the bios", you should evable virtual machine on your windows. for that:

Reboot your computer

1. Reboot your computer
2. On load press f2 again and again you are able to enter the BIOS
3. Go to configuration tab
4. Enable virtualization; the setting may be called VT-x, AMD-V, SVM, or Vanderpool. Enable Intel VT-d or AMD IOMMU if the options are available

* Download docker desktop for Windows

**Docker Hub**

* Open docker hub, which is the default repository for docker images.
* Search for node image.
* You should see in the results, node images with different versions. for example: 19-alpine3.15 means node version 19 with linux distribution alpine version 3.15.
* By default the command "docker pull node" will install the latest node image version in docker hub.
* No matter where your command line located, docker images will be installed to a reserved location.
* Now you can see the node image in the docker desktop under “Images” tab.  
    
  Graphical user interface, text, email, website

  Description automatically generated
* You can create and run a container from this image. you can see the node image in the docker desktop under “Containers” tab.  
  you can see the node image in the docker desktop under “images” tab.  
    
  Graphical user interface

  Description automatically generated

**Dockerfile**

* Install docker extension for vs code
* Create Dockerfile and .dockerignore, see example in server-3000
* cd into server-3000 and run "docker build -t server-3000 .", such "-t" stand for the image tag name and "." is the path to the Dockerfile
* After the build we should see the image in the docker desktop

**Create and run new container**

Open docker desktop, you should see the docker image in the “images” tab.  
After clicking run, you have the option of filling a settings form.

The container port, should map to the machine port.

In our case they both are 3000.

Graphical user interface

Description automatically generated

After running, the container should appear in the “container” tab

Graphical user interface, text, application

Description automatically generated

Now you can send requests on port 3000

Graphical user interface, text, application, email

Description automatically generated

**Docker Commands**

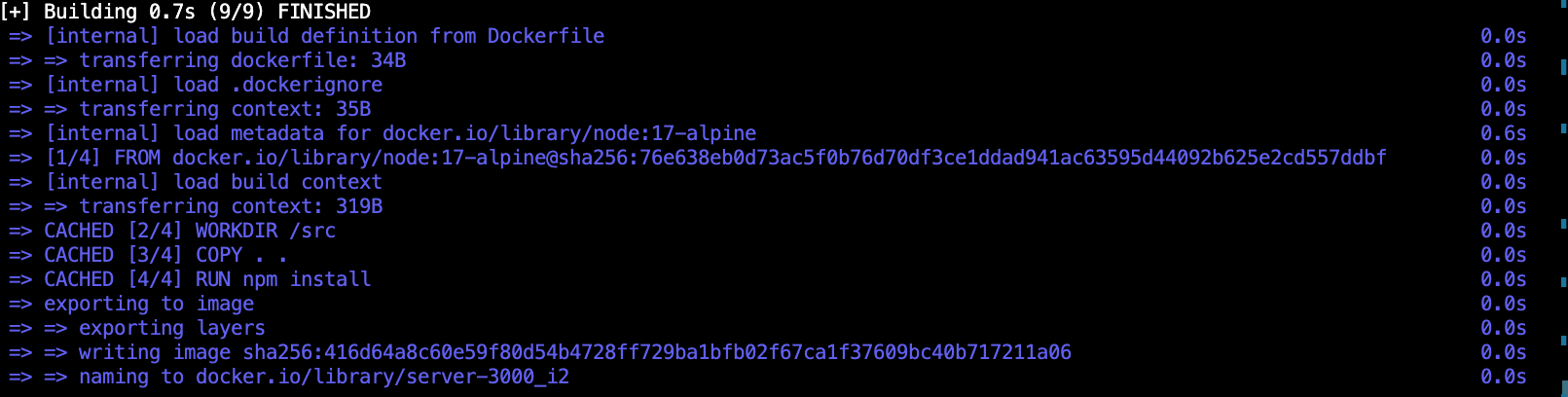
* Close the container you have created
* Run "docker images" to see the all images
* Run "docker run --name server-3000\_c2 server-3000" - (docker run --name <container name> <image bame>) - to create and run container
* You can see now that the app is running: "server is listening on port 3000"
* You won't be able to send request on port 3000 because we didn't map the container port to the machine port
* Open new terminal and run "docker ps" and you should see the running containers info (add -a to see the stopped containers info too)
* Stop the container by running "docker stop server-3000\_c2" - (docker stop <container id || container name>)
* Go back to the previous terminal and create a new container "docker run --name server-3000\_c3 -p 3005:3000 -d server-3000" (-p machine-port:container-port -d detached mode)
* Start another existing container by running "docker start server-3000 \_c1" (no need to specify -p after it configured in the first time)
* Now you are able to run the app on two ports - 3000 and 3005

**Layers Caching**

When we are building an image, the layers will be retrieved from the cache unless there is a change.  
  
When there are changes on the layers that built before RUN npm I layer, the RUN npm I layer won’t take the packages from the cache.

* Run build command “docker build -t server-3000 .”.   
  The build time:

|  |  |  |
| --- | --- | --- |
| Overall | FROM | RUN npm i |
| 0.0s | 0.0s | 0.0s |



* Change the node version in the docker file (see Dockerfile-v2), and then run the build command.   
  The build time:

|  |  |  |
| --- | --- | --- |
| Overall | FROM | RUN npm i |
| 13.7s | 8.8s | 2.8s |

Graphical user interface, text

Description automatically generated  
  
Since we are asking for a new node version, this layer won’t be taken from the cache, and it will increase the build time.

* Edit index.js and run the build command.  
  The build time:

|  |  |  |
| --- | --- | --- |
| Overall | FROM | RUN npm i |
| 3.6s | 0.0s | 2.8s |

Text

Description automatically generated

Since the COPY layer indicates for an application change, the npm install layer will re install the packages.

* To avoid this, we can copy the package.json before we are copying the all source code, and only in case of changes in the package.json, the RUN npm I layer will re install the packages (see Dockerfile-v3)  
    
  Run the build command.  
  The RUN npm I layer will take 7.3s since the COPY package.json layer has changed.  
    
  Text

  Description automatically generated
* Edit index.js and run build command.  
  The RUN npm I layer will take 0.0s since the COPY package.json layer has not changed.

Text

Description automatically generated

**Managing images and containers**

* Run “docker images” to see all images
* Run “docker ps” to see all active containers
* Run “docker ps -a” to see all containers
* Run “docker image rm <list of ids or names separated by space >” to remove image which hasn’t containers
* Run “docker image rm <id or name> -f” to remove image which has an inactive container, -f doesn’t help to remove an image which has an active container
* Run “docker start <id or name>” to activate a container
* Run “docker stop <id or name>” to deactivate a container
* Run “docker container rm <list of ids or names separated by space >” to remove container
* If you build the same image (there aren’t changes in the source code or in the Dockerfile) with different name, you will get the same id for all.
* You can’t remove image by id that references to multiple repositories
* Run “docker system prune -a” to remove all images and containers
* Run “docker build -t <image name>:<version> .” to create an image and version (docker build -t server-3000:v1 .)
* Run “docker run --name server-3000\_c4 -p 3000:3000 -d server-3000:v1” to create a container from a specific image version
* The difference between “docker start <container>” and “docker run <container name and all the rest of the command>” is that “run” creates brand new container while “start” activate an existing one
* Run “docker start -i –rm <container>”. -i to see the logs –rm will remove the container once it stopped.

**Volumes**

The container would not be updated with the changes of the source code without building its image with the changes.

Volume allows the container to get updated with new changes.

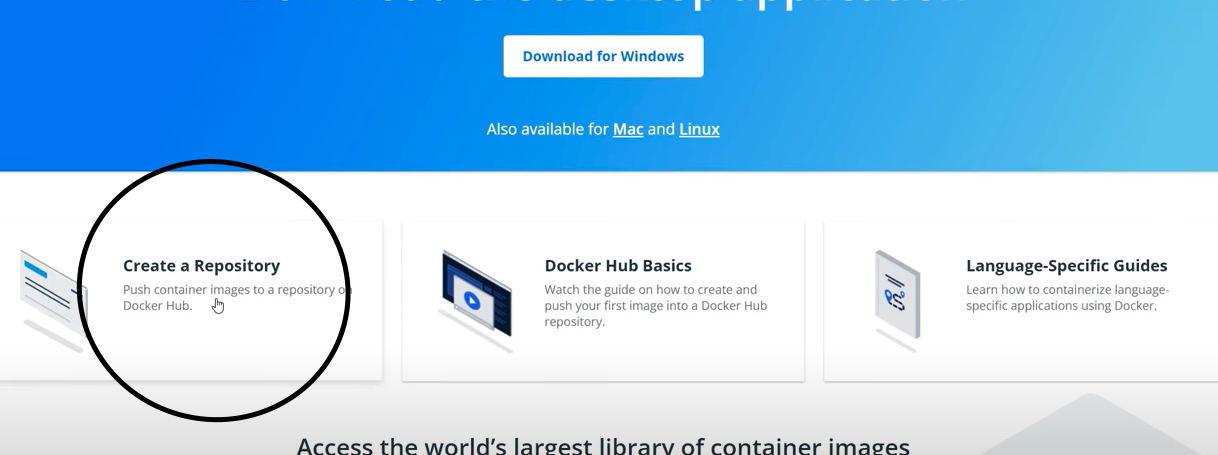
The image itself won’t be updated before a new build.

* Install nodemon globally in the container. for that add the installation command and the running command to the Dockerfile (see Dockerfile-v4)
* After editing the Dockerfile, build a new image “docker build -t server-3000:v2 .”
* Run “docker run --name server-3000\_c5 -p 3006:3000 -v /Users/Yossi.arye/lectures/full-stack-lectures/18-docker/server-3000:/src -v /src/node\_modules server-3000:v2”.   
  The first -v stands for mapping all the changes to the docker working directory.  
  The second -v is because, since we are mapping always the machine source code, we should have in the machine directory the node\_modules, so this flag indicate docker to install the node\_modules in the container to the machine directory.

**Docker Compose**

* See docker-compose.yaml
* Run “docker-compose up” to activate the container
* The image name will be the service name concatenated to the current directory name
* Run “docker-compose down” to deactivate the container
* Add the above command “--rmi all” to delete all the images that created from this docker-compose (docker-compose down –rmi all)
* Add -v flag to the above command to delete all the volumes that created from this docker-compose

**Uploading to docker hub**

* Sign in to docker hub
* Create repository   
    
  

Graphical user interface, application, email, Teams

Description automatically generated

* Build an image “docker build -t 4164/server-3000 .”
* Login to docker hub “docker login”
* Push the image “docker push 4164/server-3000” and after the pushed done you can see it in docker hub  
     
  Graphical user interface, application

  Description automatically generated
* Pull your image: click on public view and copy the command  
    
  Graphical user interface, application

  Description automatically generated

Graphical user interface, application, Teams

Description automatically generated

* Delete the image on your machine and then pull from docker hub
* Paste the copied command to your terminal and pull the image
* You should see the image in the images list