Lab 1.2:   
Docker – Tasks

This laboratory is tackling virtualization by usage of containerization. The goal of this laboratory is to gain a basic understanding of containerized virtualization and that knowledge to get an application to run.

This laboratory focuses on the deployment of the *store-ui* and *product-microservice* from the case study into a local docker instance. The *store-ui* running in the local docker instance should connect to the services which have already been used in the VirtualBox-Laboratory.

You can find the GitLab here:   
https://git.inf.fh-dortmund.de/01/cloud-labs

Overview:

[Student Group Information 2](#_Toc146983719)

[Tasks 2](#_Toc146983720)

[1. Docker Introduction 2](#_Toc146983721)

[Images 2](#_Toc146983722)

[Container 3](#_Toc146983723)

[Custom Images 3](#_Toc146983724)

[2. Case Study 5](#_Toc146983725)

[Docker Images 5](#_Toc146983726)

[Building the Application 5](#_Toc146983727)

[Container Registries 6](#_Toc146983728)

[Networking Modes 6](#_Toc146983729)

[3. Case Study 7](#_Toc146983730)

[4. Reflective Tasks 7](#_Toc146983731)

[Conclusion 9](#_Toc146983732)

[Outlook 9](#_Toc146983733)

# Student Group Information

Enter your student group information in the following table and provide your group number.   
(This number will be important for later labs as well, please remember it!)

**GroupNumber (e.g. Group01): *Group10***

|  |  |
| --- | --- |
| **Member Name** | **Matriculation Number** |
| **Eraj Rizvi** | **7219193** |
| **Chowdhury Abida Anjum Era** | **7219089** |
| **Nimisha Vilayatarani** | **7219293** |

# Tasks

This section contains the questions required to pass the lab acceptance session, and therefore need to be answered. Please make sure to install Docker before you continue with the following tasks (document “Lab 1.2 Docker - Installation Instructions").

## Docker Introduction

The tasks in this chapter address the basic docker commands and concepts.

### Images

1. What is a docker image?

**Answer:**

Docker image is a blueprint that define the environment and configuration of an application, including its code, dependencies and libraries. Also, it is a read-only template that contains instructions for creating a DOCKER CONTAINER.

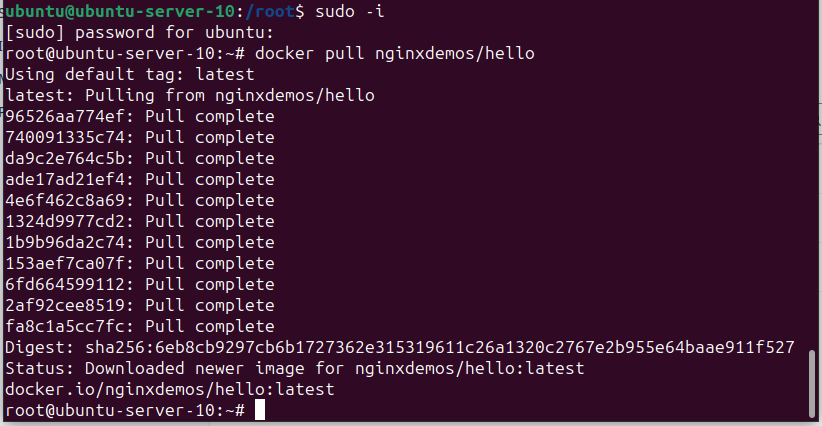
It used to execute code in a Docker container. Docker images act as a set of instructions to build a Docker container, like a template. Docker images also act as the starting point when using Docker. An image is comparable to a snapshot in virtual machine (VM)environments.

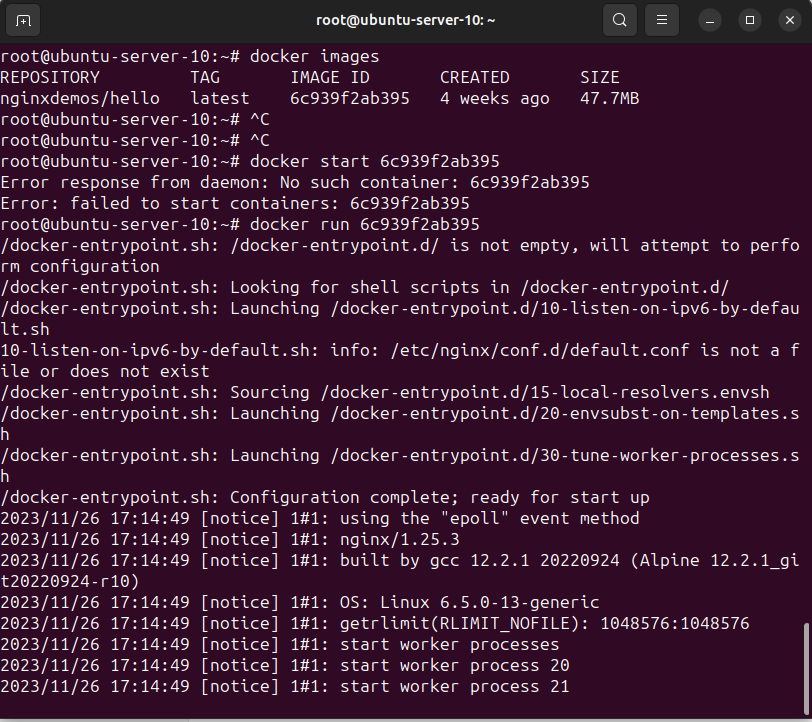
1. How can you download the image ‘*nginxdemos/hello*’ to your docker instance?

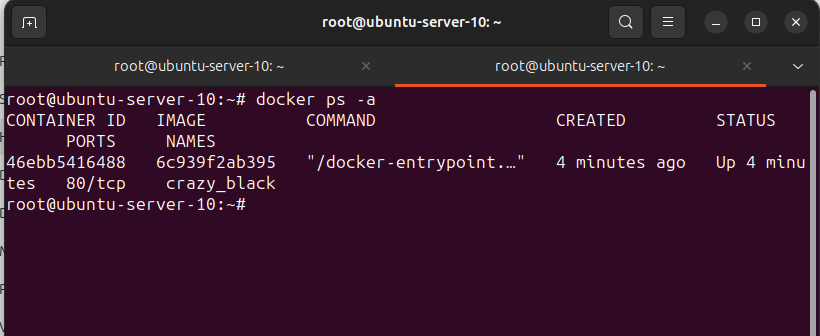
**Answer:**

To download the mention image, we can use the following command:

**docker pull nginxdemos/hello**



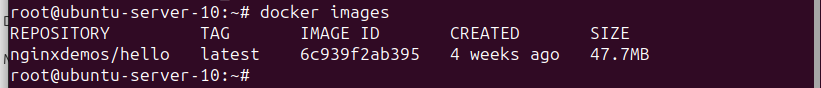
****

****

1. How can you check which images you have downloaded on your Docker instance?  
     
   **Answer:**

To check the downloaded images on your Docker instance we can use the following command:

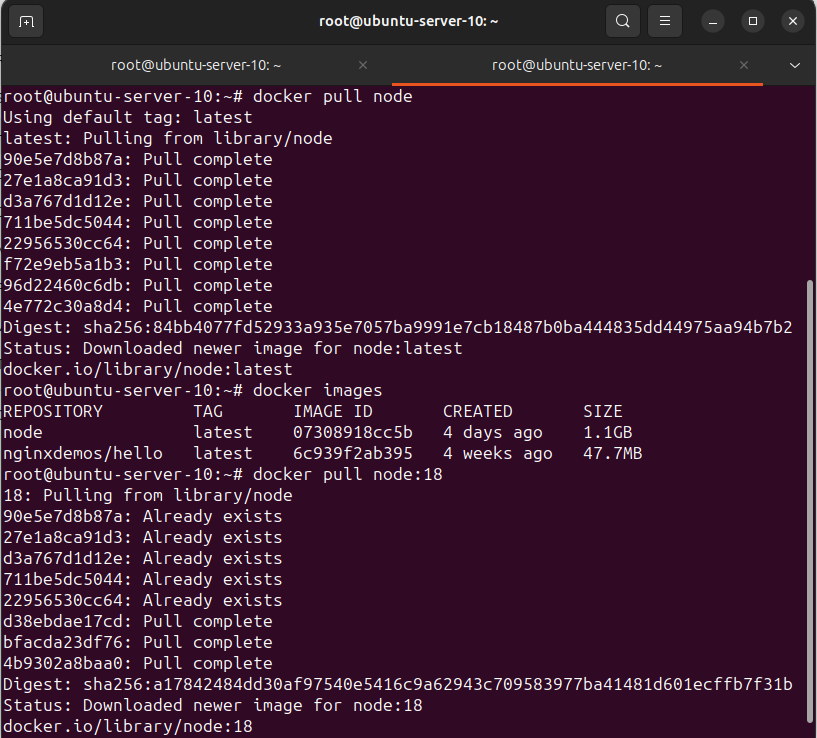
**docker images**

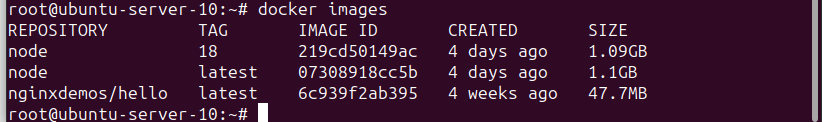
****

1. How could you download a specific version of *NodeJS* as a Docker image?  
     
   **Answer:**

To download a specific version of NodeJS as Docker image we can use following command:

**docker pull node:<version of NodeJS you want to download>**

****

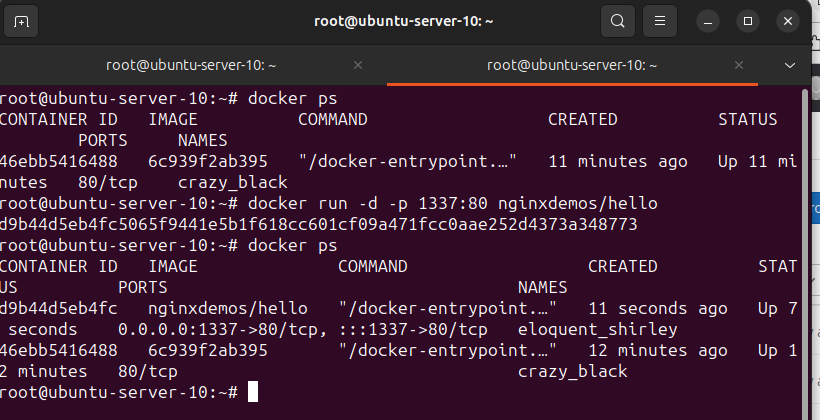
****

### Container

* + - 1. Which command is needed to run the ‘*nginxdemos/hello*’-image, and which parameters are needed to see the results in your web browser at “*http://localhost:1337*”?  
           
         **Answer:**

To run the mentioned image, we can use the following command:

**docker run -d -p 1337:80 nginxdemos/hello**



* + - 1. What is the difference between a container and an image?

**Answer:**

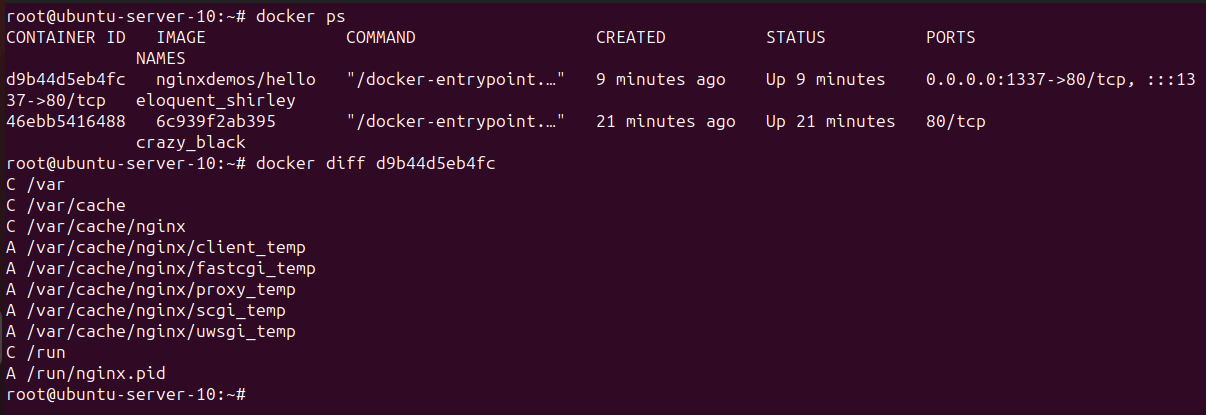
A **Docker image** is the template loaded onto the container to run it, like a set of instructions. It is a blueprint that define the environment and configuration of an application, including its code, dependencies and libraries. Also, it is a read-only template that contains instructions for creating a Docker container.

On the other hand, a **Docker container** is a self-contained, runnable software application or service. Docker container is the running instance of Docker image.

You store images for sharing and reuse, but you create and destroy containers over an application’s lifecycle.

* + - 1. How can you check the filesystem changes since the startup of a container?  
           
         **Answer:**

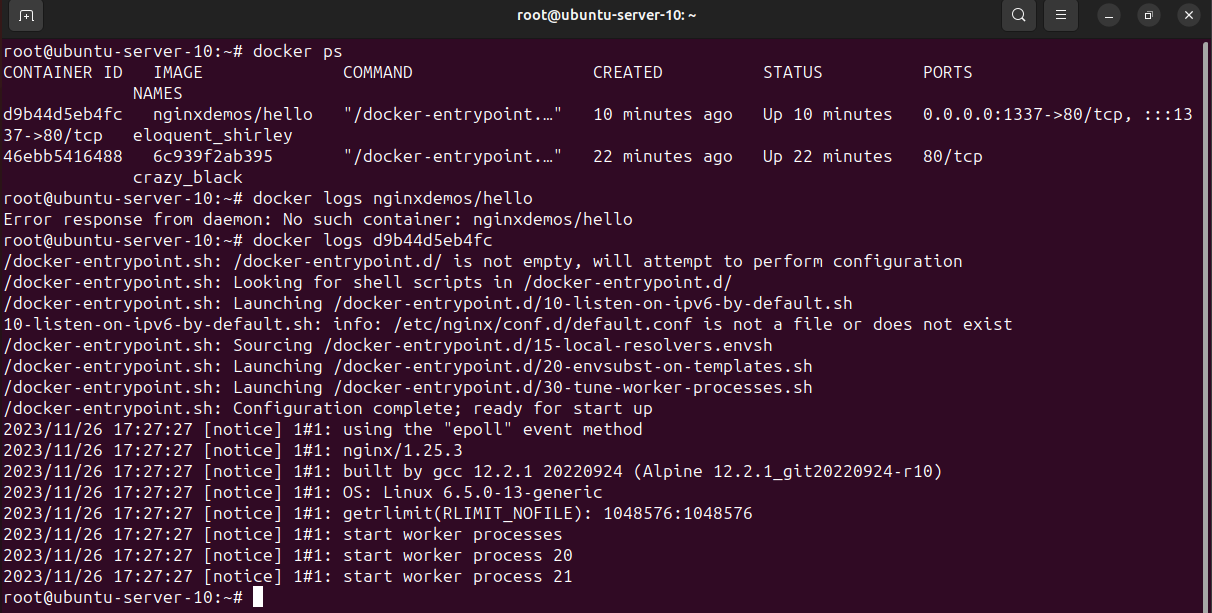
To check the filesystem changes since the startup of a container we can use the following command:

**docker diff <container-name or container-id>  
**

* + - 1. Are you able to access the container and read the standard output?  
           
         **Answer:**

Yes, we can access the container and read the standard output by using following command:

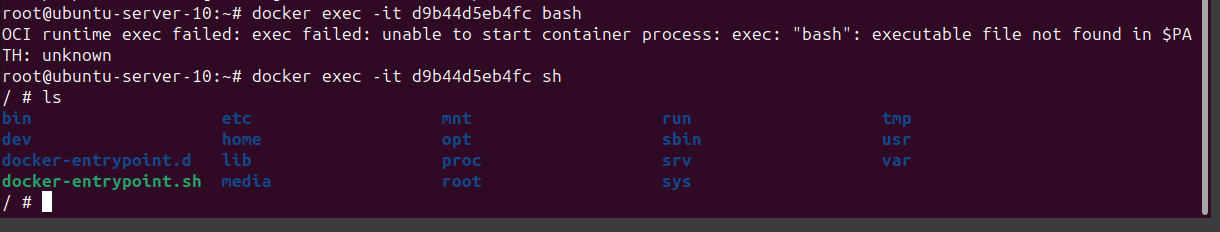
**docker logs <conatiner-name or container-id >**



* + - 1. How can you execute code inside of an already running container?  
           
         **Answer:**

We can execute the code insider of running container by following command:

**docker exec -it <container-id> bash**

****

### Custom Images

For the following tasks copy the Dockerfile from the “Docker Instructrions” document (found in the additions) in an empty folder. Then, delete all docker resources to start on a “*green field*”. (“docker system prune -a”)

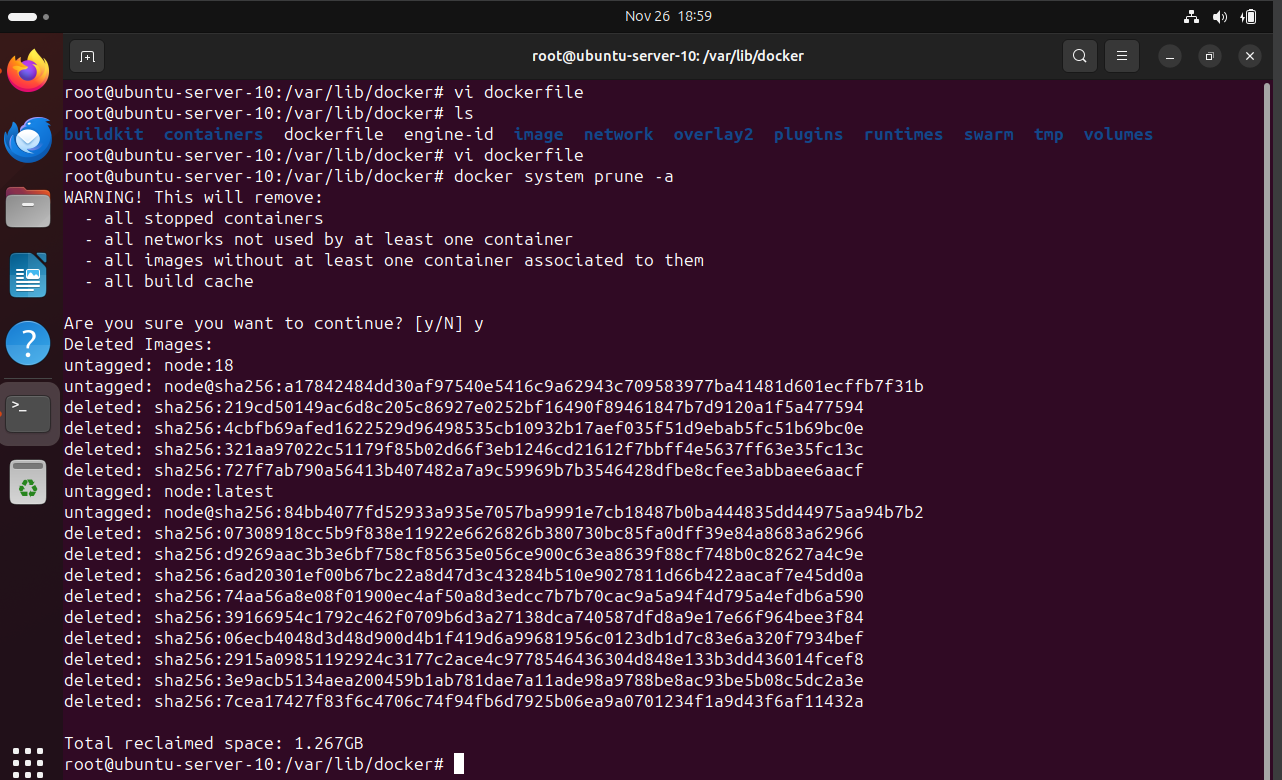
Follow the section 4: Disable Docker Build kit for Intermediate images and containers in the installation document. Then continue with the tasks.

**IMPORTANT:** Build the image by running the following command (Replace <group> with your group name)

docker build . -t <group> --rm=false --no-cache

What’s the purpose of the command you just ran?

**Answer:**

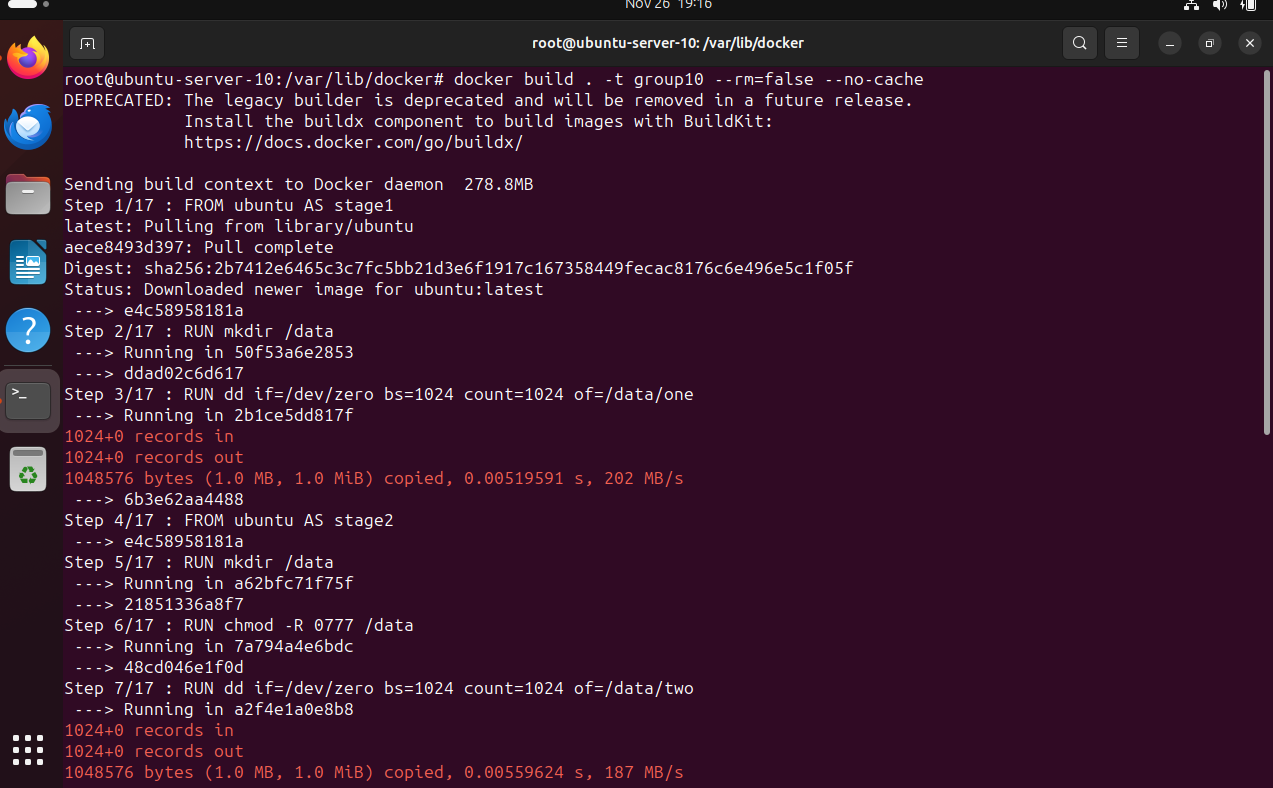


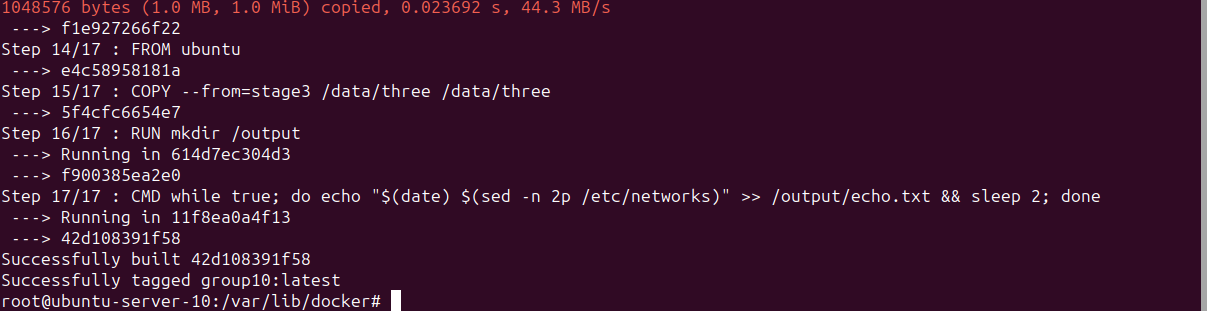
The mentioned command builds a Docker image using Docker File in the current directory,

-t flag assign a tag or name to Docker image,

--rm=false prevents the removal of container,

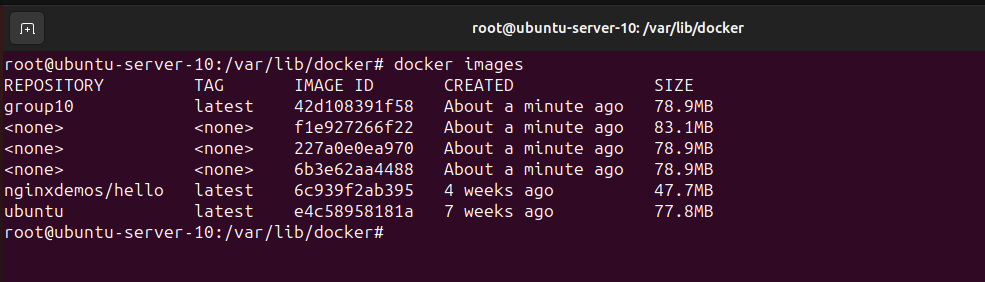
--no-cache force Docker to build image without using cached layer.



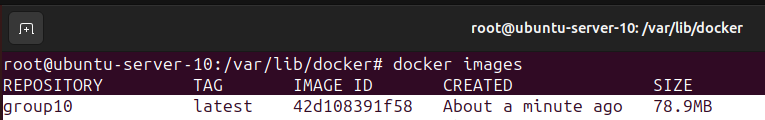


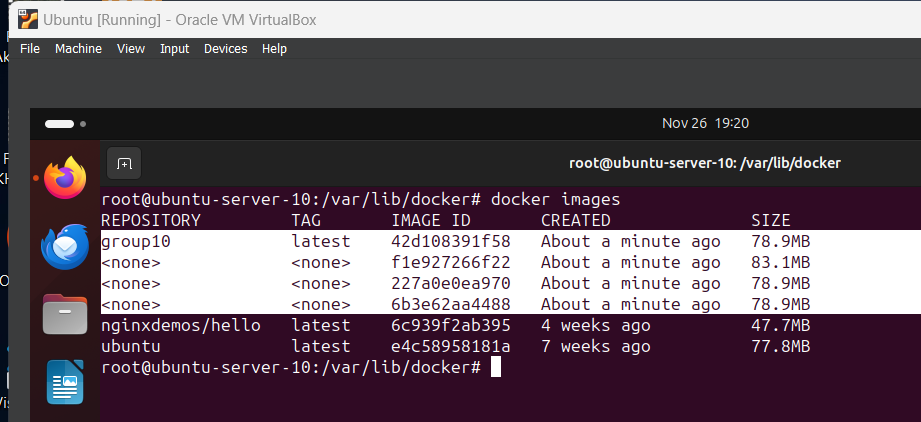
What’s the role of containers during the build of an image?   
(**Hint**: Check the IDs in the build command)

**Answer:**  
During the build of a Docker image, containers play a crucial role in executing each instruction specified in the Dockerfile. Containers are utilized in a multi-stage build process to execute each step defined in the Dockerfile. The build process involves creating an intermediate container for each instruction and committing the changes made by that instruction into a new layer. Containers are created at each stage with specific purposes, allowing for isolation of tasks and efficient image construction. These layers are then stacked to form the final image. The container IDs you see in the build command output correspond to temporary containers created during each build step.

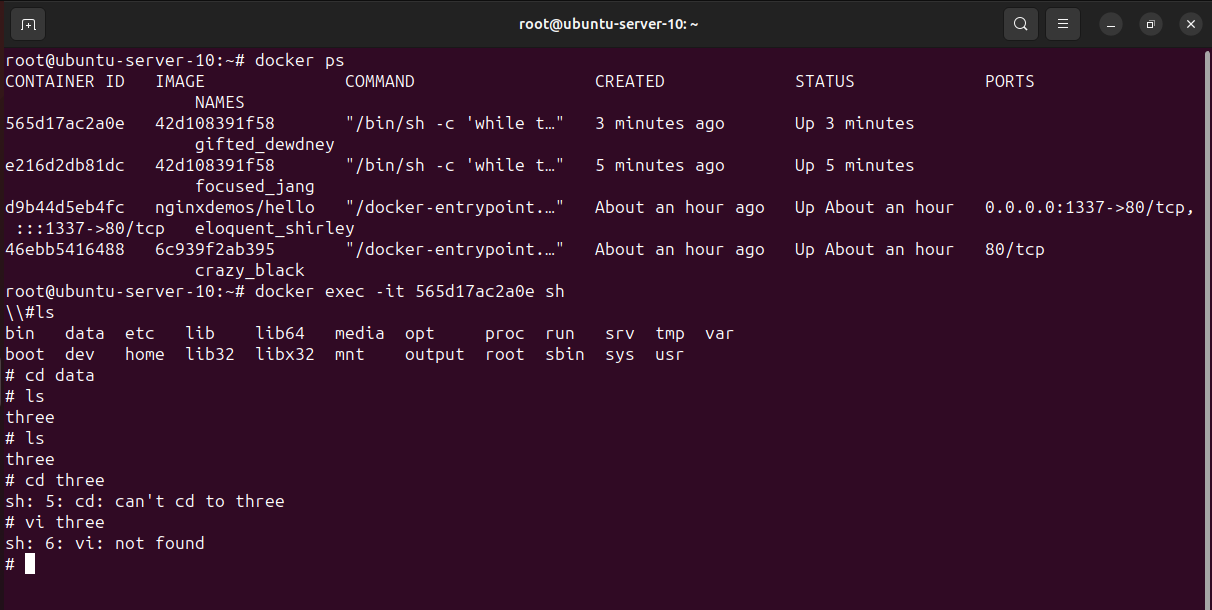


How many images does this command create?  
  
**Answer:**  
This command will create a single Docker image.





Run each image that has been created during the build in its own docker container. (Use the Image IDs from “*docker image list*” rather than tag names)   
Connect to each container and list the content of the “*/data*” folder and print “*echo.txt*”. (**Hint**: Use the -dt to keep the containers alive in parallel)  
Did you expect the result?  
  
**Answer:**

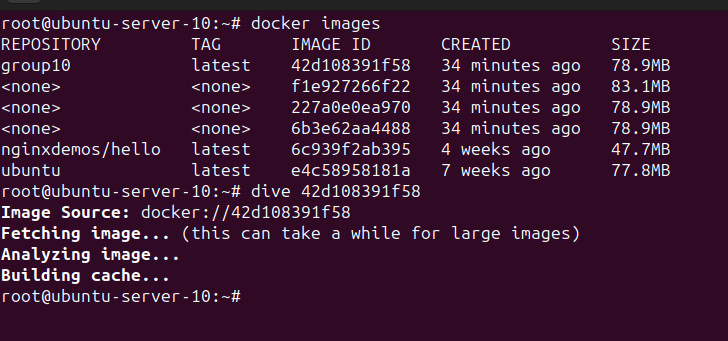


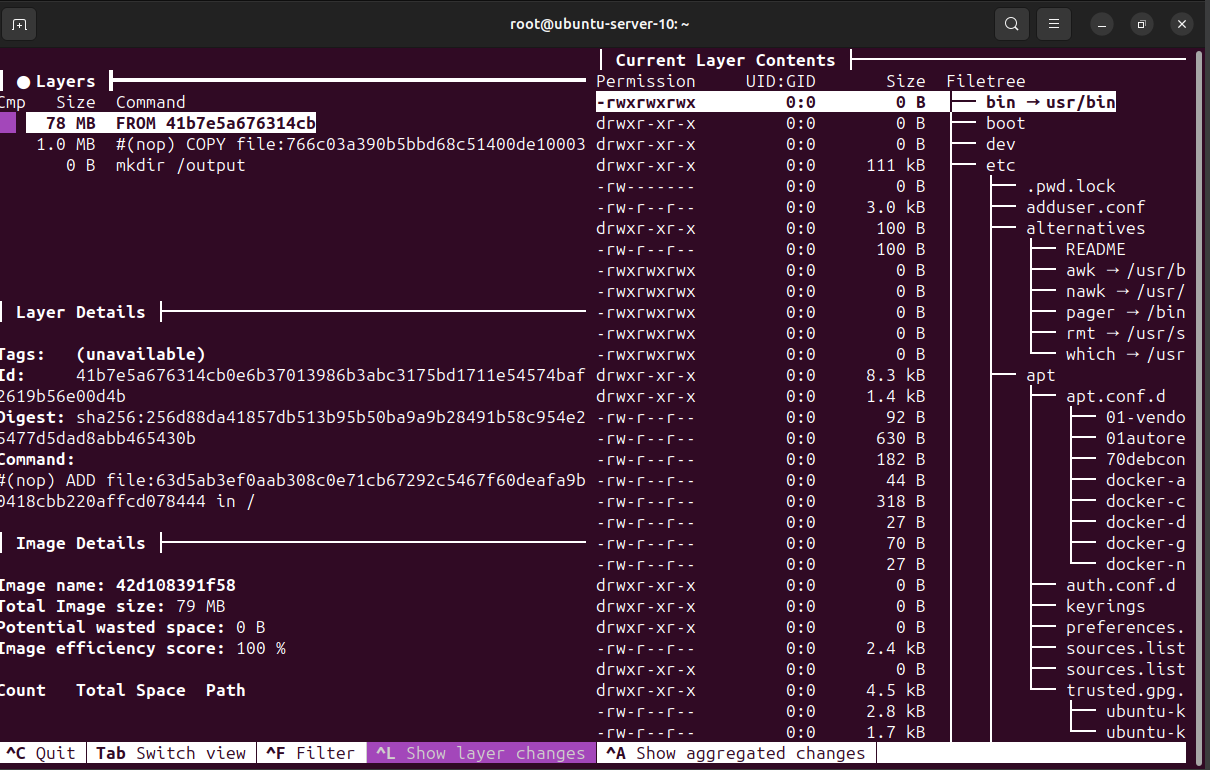
/data: no such file or directory

Run the command from the “*Container*”-Section Task 3 again and compare the results for each container you started.  
  
**Answer:**



What does the tool “*dive*” tell you about the image with your group’s tag?  
  
**Answer:**

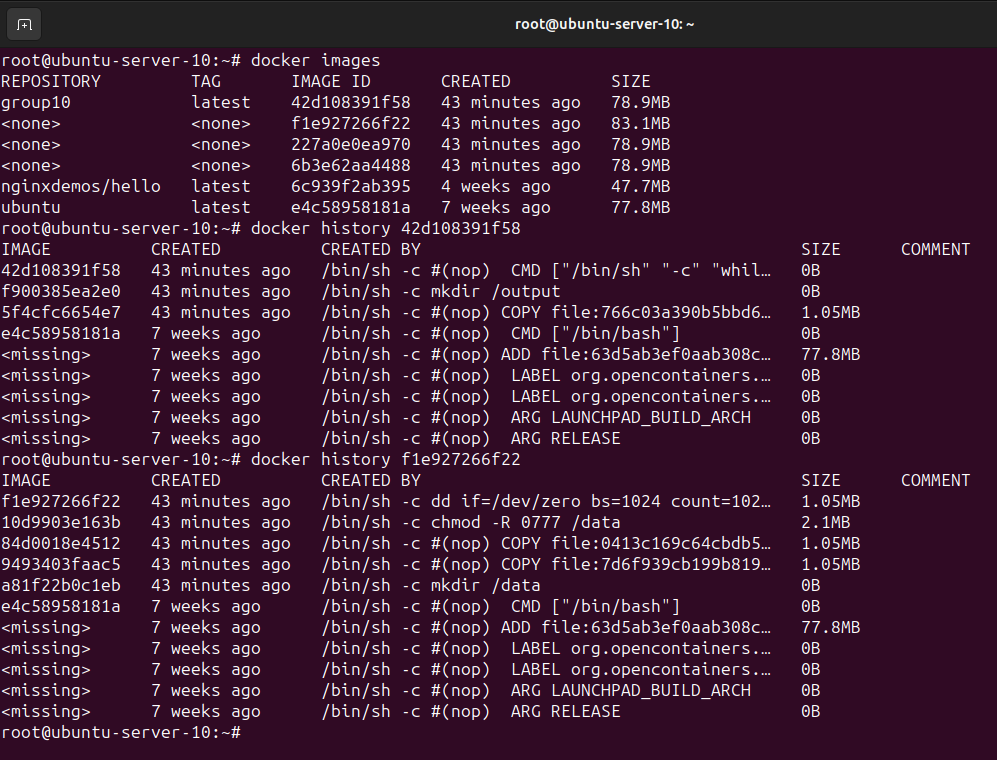
****



**Dive** is a tool that allows you to explore and analyze Docker images layer by layer. It provides insights into the contents of each layer, their sizes, and helps you identify potential issues or inefficiencies in your Docker images.

Running **dive <group>** lets you inspect the layers of an image tagged with your group name, showing details like layer sizes, file system changes, and command history. It helps identify inefficiencies and optimize Docker images for size and composition.

Which ImageID contains the changes of this command “*chmod -R 0777 /data*”?  
(You need to be able to show it, by running the image before and after the command ran; **Hint**: Try “*docker history*” command)  
  
**Answer:**

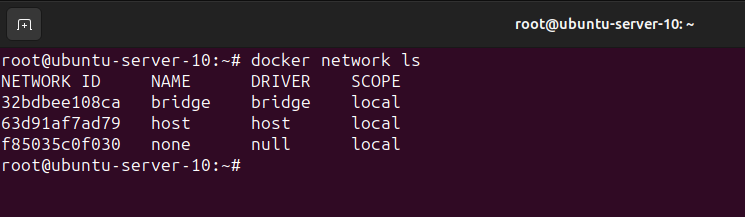


### Networking

This chapter requires you to use Edgeshark and wireshark from the instructions document.

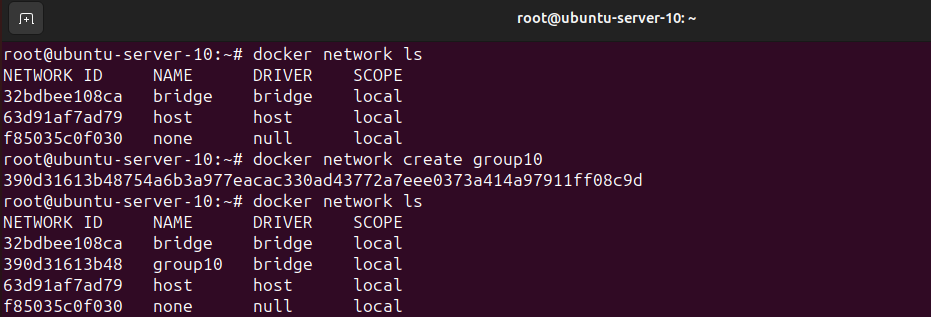
1. How can you check the networks available in docker?  
     
   **Answer:**  
   By using the command:

**docker network ls**



1. How can you create a new network for your application?  
     
   **Answer:**  
   By using following command, we can create a new network for your application:

**docker network create <network\_name>**



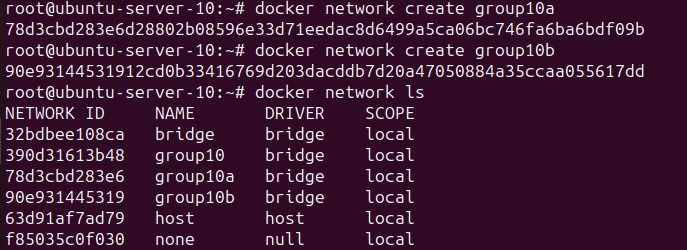
1. Create two networks and two containers (ubuntu image), connect each to a different network. Are you able to ping one container from the other one? (You probably need to install “*iputils-ping*”  
     
   **Answer:**

By using following command,

**docker network create Group10**

**docker network create Group10b**

**docker network ls**

****

By using following command, you can create containers:

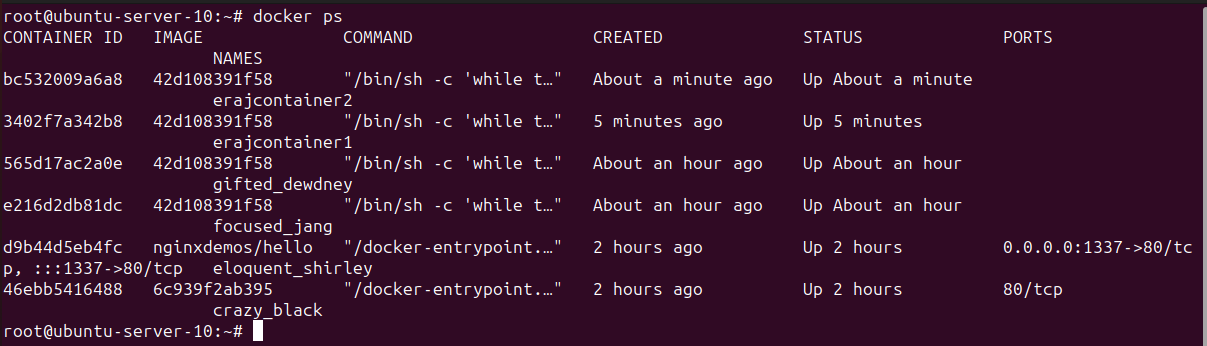
**docker run –name <containerName> --net <network name> <imageName>**

**or**

**docker run --containerName --net network name -p 1337:1337 imageName**

****

****

****

By using following command, you can ping one container to another:

**docker exec –it <containerName1> ping <containerName2>**

**docker exec –it erajcontainer1 ping erajcontainer2**

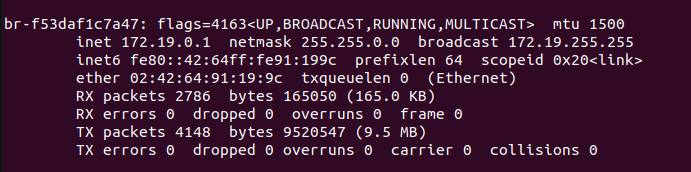
or can use IP Address as well:

**docker inspect <containerName>**

now use the below command:



1. What does the creation of a network do on your host? (Create a network -> Check ifconfig/ip 1/ipconfig; You probably want to search for the network id)  
     
   **Answer:**



1. Run Edgeshark and connect both containers you created to the same network.  
   Are you able to capture the packages of a ping from one container to the other?  
     
   **Answer:**

Yes

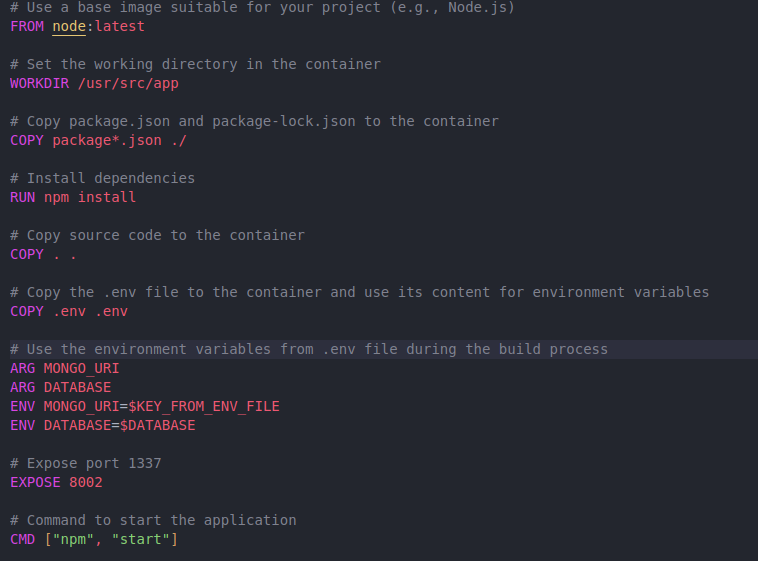
## Case Study

This Section is going to apply the concepts from the previous section (“*Docker Introduction*”) to the case study.

### Docker Images

1. How can you containerize the *store-ui* and make it available on port 1337? (Build the *store-ui* on your local machine and move the files to the image in your Dockerfile. The variables from “*.env*”-file are written to the output files during the build, there is no way to change it later on.)

**Answer:**

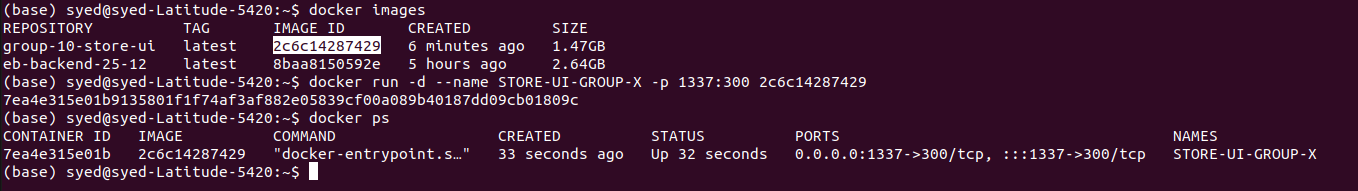
****

After setting Dockerfile, run this command to build Docker image:

**docker build --build-arg MONGO\_URI=mongodb://172.17.0.3:27017 --build-arg DATABASE=storeDB -t products .**

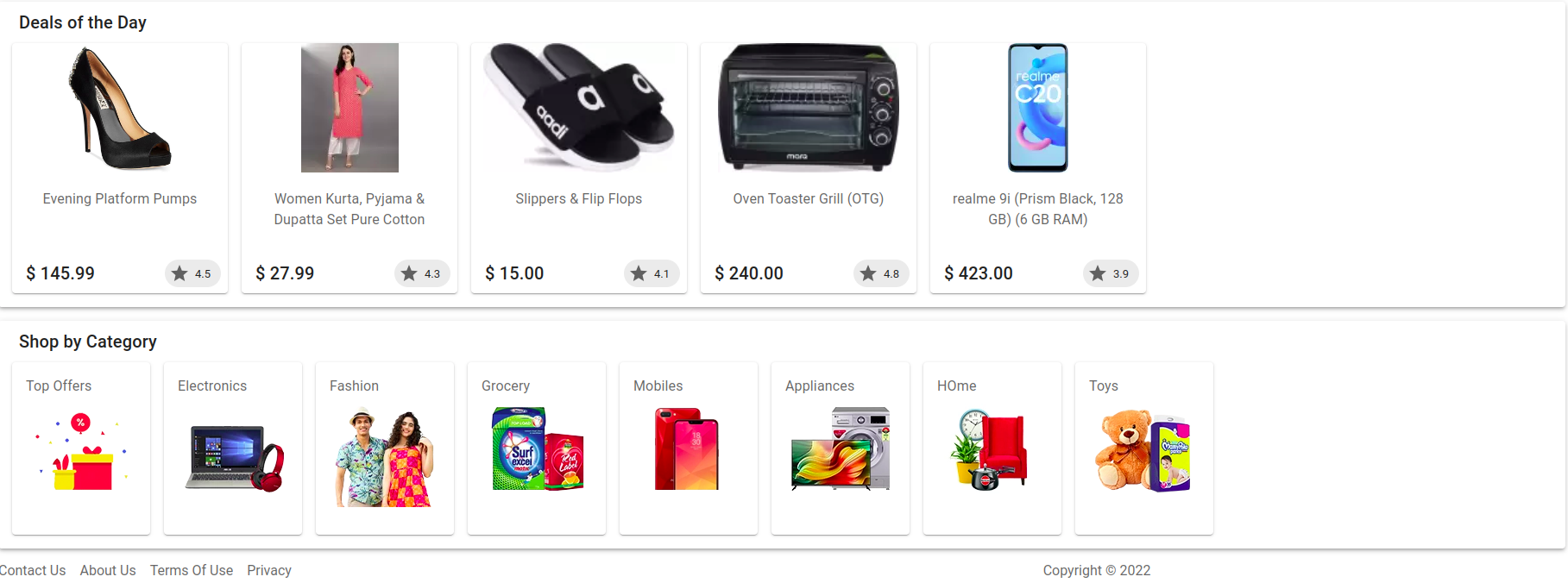
Run the Container on port 1337 by this command:

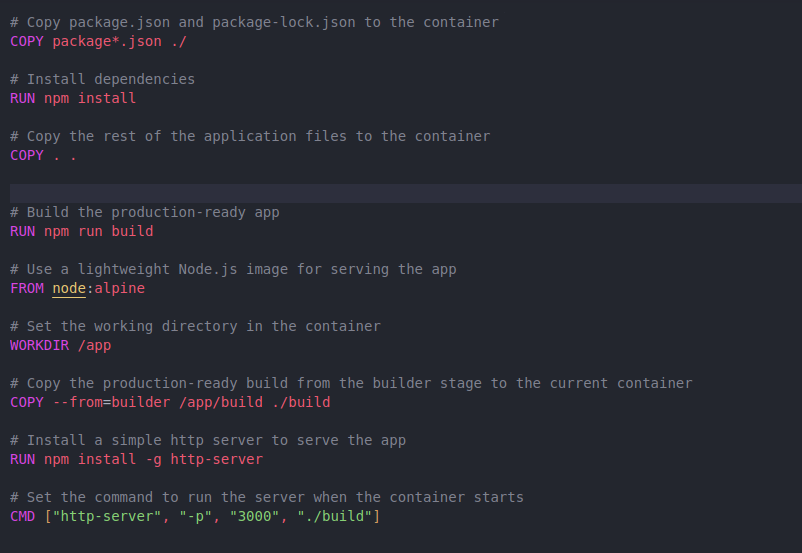
**docker run -d --name PRODUCTS-GROUP-X -p 1337:5000 products**



1. Are you able to use this image and connect your *store-ui* to the remote services?  
     
   **Answer:**

YES, see the following screenshot

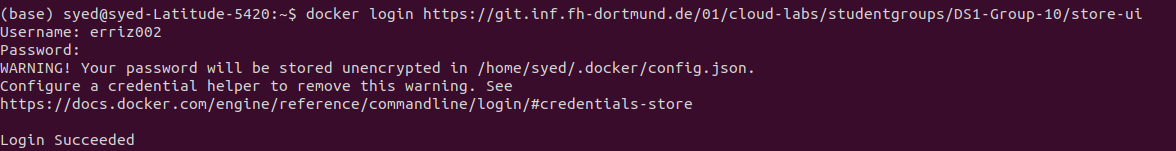


1. Create a multi-stage Dockerfile for your *store-ui* and build the application inside the first stage.  
   What problems does this approach solve?   
   Can you run it on any machine where docker is installed?  
     
   **Answer:**  
     
   

### Container Registries

1. How can you connect to the docker registry you set up during the steps for the case study from the *Docker Instructions* document?  
     
   **Answer:**  
   Run this command to login:

docker login <https://git.inf.fh-dortmund.de/01/cloud-labs/studentgroups/DS1-Group-10/store-ui>



1. How can you make your locally created image available in the container registry of your GitLab repository?  
     
   **Answer:**



A screen shot of a computer

Description automatically generated

1. How can you automatically create a new version of the application hosted in your repositories on a commit?  
     
   **Answer:**  
   To create a new version of application CICD can be used.
2. How can you use the image from your Gitlab container registry?

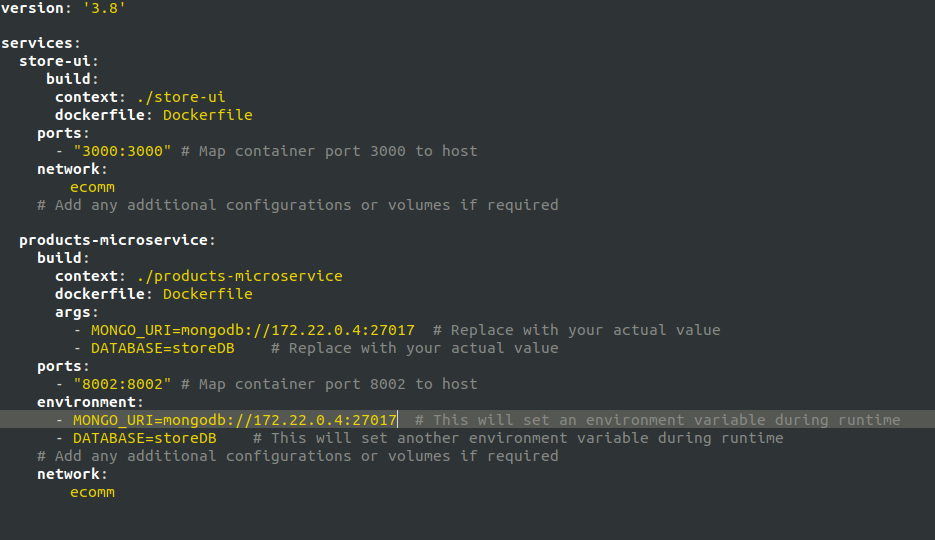
**Answer:**

A screen shot of a computer

Description automatically generated

### Deployment

To reduce the amount of docker commands needed we are using docker compose in this section.

1. How can you deploy the application (*store-ui* and *product-microservice*) using docker compose?  
     
   **Answer:**  
   

Using docker compose yml file docker compose can be configured

1. How can you create a MongoDB which your *product-microservice* requires?  
     
   **Answer:**

Using the following command pull the MongoDB image:

**docker pull mongodb/mongodb-community-server**

Run as a container:

**docker run --name mongo -d mongodb/mongodb-community-server:latest**

Connect to the MongoDB Deployment with mongosh:

**docker exec -it mongo mongosh**

1. What changes are needed to persist the database if you delete the container and run it again?  
     
   **Answer:**  
   Create a Docker volume to store MongoDB data independently of the container:

**docker volume create mongodb\_data**

-- Run the MongoDB container, mounting the volume to persist data:

**docker run -d -p 27017:27017 --name my-mongodb -v mongodb\_data:/data/db mongo**

-- Create a directory on your local machine where MongoDB data will be stored:

**mkdir -p /home/syed/ASSIGNMENT**

-- Run the MongoDB container, using a bind mount to link the local directory to MongoDB's data directory:

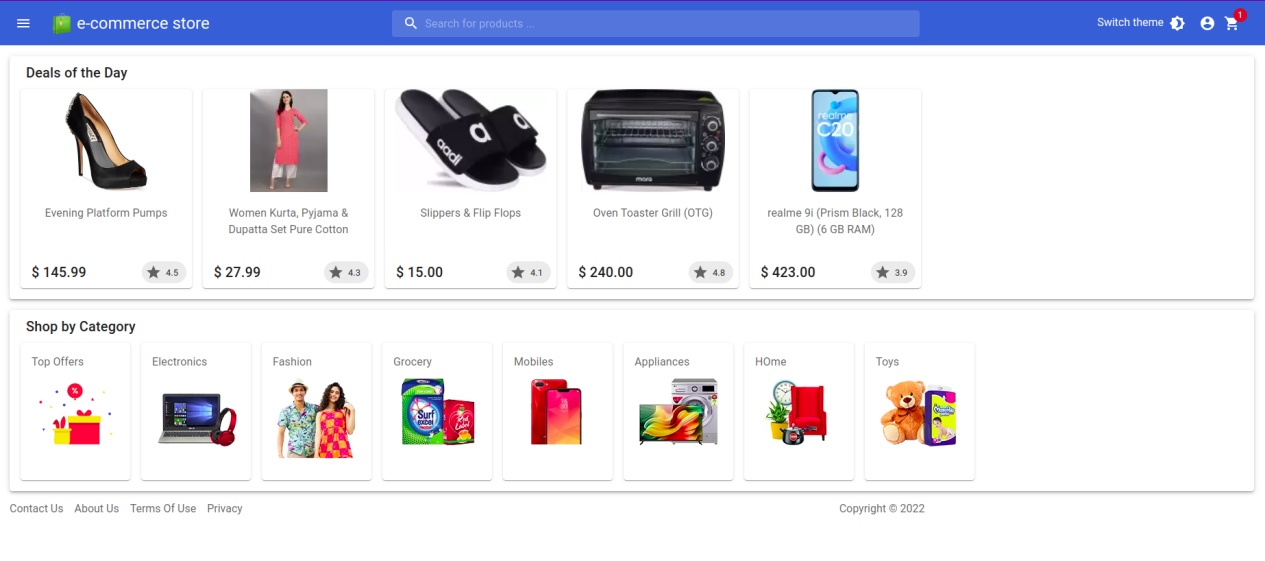
**docker run -d -p 27017:27017 --name my-mongodb -v /home/syed/ASSIGNMENT**

**:/data/db mongo**

1. Are you able to serve the *store-ui* locally? Does it fetch the products from your local *product-microservice* instance?

**Answer:**

Yes, the store-ui can be served locally and it fetches the products from product-microservices container instance successfully.



## Reflective Tasks

These tasks are (almost) the same as in the VirtualBox laboratory, but your answers will probably change. Try to compare the containerization approach with the previous approach.

1. Do you think the way the application has been created in this laboratory is efficient and worth it?

**For Development**:  
  
**Answer:**

Docker ensures consistent environments across development, testing, and production, reducing the "it works on my machine" problem by providing a unified environment for developers.  
  
**To be run as a server**:  
  
**Answer:**

Docker enables easy horizontal scaling by deploying multiple containers, allowing applications to handle increased load efficiently without extensive resource allocation.  
  
**To rent them to customers:**  
  
**Answer:**

Docker simplifies the installation process for customers by encapsulating applications and their dependencies, allowing for straightforward deployment across various environments without complex setup steps.

1. Is the used approach scalable? (vertically and horizontally)

**Answer:**

Yes, the approach is scalable. By default, a container has no resource constraints and can use as much of a given resource as the host's kernel scheduler allows. However, Docker provides ways to control how much memory, or CPU a container can use, setting runtime configuration flags, allowing vertical scalability. Although, the current setup lacks explicit provisions for horizontal scalability in the compose files, Docker inherently supports both horizontal scalability and load balancing through orchestration tools like Docker Swarm or Kubernetes.

1. Is this approach more efficient than the previous approach?

**Host system resources**:  
  
Docker containers share the host OS kernel, requiring fewer resources compared to VMs, which each run their own OS, leading to better resource utilization and efficient use of system resources.  
  
**Time to setup**:  
Docker containers typically have faster setup times compared to VMs due to their lightweight nature, allowing for quicker deployment and scaling of applications.

1. How can you migrate your application to the machine of one of your group members?  
   (Compared to the previous approach: Is this way more efficient?)  
     
   **Answer:**  
   Application Packaging -- Share the Docker image or Dockerfile along with instructions -- Upload the packaged application to a cloud service -- Share the repository -- Set Up on the New Machine

Yes, it’s more efficient compared to the previous approach.

1. Is there any way to configure resource limits for containers?  
     
   **Answer:**  
   Docker provides flags and options (like `--cpu`, `-m`, `--blkio-weight`, etc.) to define CPU, memory, I/O, and network bandwidth limits for containers, enabling fine-grained control and efficient management of resource allocation for individual containers.
2. Scenario: Your application is deployed in a docker environment which has a 100% uptime guaranteed. The application is mission-critical for your business and is not allowed to become inoperative.
   1. How could you detect failures in the application containers quickly to prevent outages? Think of a way to replace an inoperative container with a new one.

**Answer:**

By implementing monitoring and alerting for application containers, utilize Docker's health check mechanism to perform periodic checks on the containers. Use container orchestration tools like Docker Swarm, Kubernetes, or ECS to manage the deployment and scaling of application containers.

* 1. Will the new container be available with the same configuration and Network-Identity as the old one?

**Answer:**

The new container generally inherits the same configurations (based on Docker run parameters, Compose files, and image used) as the failed container, ensuring consistency in settings such as environment variables, volumes, ports, and the base software within the container.

* 1. Does docker offer any assistance for this scenario?

**Answer:**

Docker provides restart policies (--restart) that automatically restart containers in case of failures. This feature ensures that failed containers are automatically replaced with new ones, reducing manual intervention and aiding in maintaining high availability.

We have other assistance/tool: Container orchestration, Load balancing, Data persistence, Health checks, Rolling updates, Monitoring and logging.

# Conclusion

This lab should teach you the basics AND advanced techniques of containerized virtualization using docker. You should be able to set create multi-stage images, check how they have been built, run containers, capture traffic between containers and “debug” your containers by directly accessing them through cli.

# Outlook

The docker approach is a headless without unnecessary dependencies. It therefore is a lean approach for virtualization but it still lacks some functionality needed to create highly scalable applications. Therefore, the next laboratory will focus on “cloud virtualizuation” methods.