Containerization & Linux eksamen 2023

Sebastian Theodor Laursen - [stla26845@edu.ucl.dk](mailto:stla26845@edu.ucl.dk)

Indholdsfortegnelse

[Problemformulering 3](#_Toc137608507)

[Docker 3](#_Toc137608508)

[Docker Swarm 3](#_Toc137608509)

[Docker volumes 3](#_Toc137608510)

[Linux (CLI) 3](#_Toc137608511)

[Løsning: 4](#_Toc137608512)

[Step Guide til processen 4](#_Toc137608513)

[Containerization 4](#_Toc137608514)

[Swarm 4](#_Toc137608515)

[Tilføjelse af nye produkter 5](#_Toc137608516)

[Konklusion 6](#_Toc137608517)

[Guide til at køre applikationen 6](#_Toc137608518)

[Link til gihub repository: 7](#_Toc137608519)

# Problemformulering

Hvordan kan man bruge Docker til at containerize en allerede eksisterende applikation?

# Docker

Docker er en open source platform der er lavet til at styre containers. Containere er lavet til at løse ”der virkede på min computer” problemet hvor forskelle på det system der kører en applikation kan føre til at den ikke virker ordentligt. Containere er en pakke der indeholder alting som skal bruges for at køre den, og dette er ting som libraries, dependencies osv. Containere har dermed kun de ting der er strengt nødvendige for at køre dem så de er meget lightweight da de for eksempel ikke skal have fulde operativsystemer med sig. Docker benytter sig af images, som er deres form for blueprint, som der kan indeholde applikationer eller isolerede features osv. Disse images er det som en container kører på. En af de mest relevante fordele for containers, udover hvor lightweight de er, er at de kan sættes op til at genstarte hvis de af uhensigtsmæssige årsager stopper, sådan at de altid er live.

# Docker Swarm

Docker Swarm er et værktøj der benyttes til at orkestrere arbejdet for Dockerized applikationer. En swarm opsættes på et cluster af flere maskiner der snakker sammen. En swarm består af en swarm manager der uddeligerer arbejdet, og resten af maskinerne i dets cluster, nodes, udfører arbejdet. En af de største fordele ved swarm er at man kan få udført load balancing. Hvis man har flere servere der skal udføre samme service så kan swarm uddelegere arbejdet ligeligt til alle servere, sådan at man ikke risikerer at en server får alt trafikken og at den potentielt ikke kan udføre sit arbejde ordentligt, mens at de andre servere ikke laver noget.

# Docker volumes

Docker volumes er det som benyttes for persistens i Docker. Disse volumes bruges til at persistere data udenfor containeren, sådan at hvis containeren lukkes ned, og startes op igen, så vil man have gemt sin data. Dette ses især hvis man har sat en container op til at genstarte, og så er det en kæmpe fordel at have mountet en volume til den, sådan at den kan fortsætte med den samme data den benyttede sig af før containeren døde. Et godt eksempel er når man bruger Docker swarm med replicas og en service dør, hvorefter en anden replica tager dens plads.

# Linux (CLI)

Command-line interface (CLI) er det jeg har brugt til at udføre kommandoer for Docker i dette projekt. Jeg bruger en Windows computer med Docker Desktop og The Windows Subsystem for Linux (WSL) til at tilgå en Ubuntu terminal, for at udføre Linux kommandoer. Hvilke kommandoer jeg har brugt i terminal og hvordan beskrives i løsningen.

# Løsning:

Til at starte med clonede jeg repositoriet med opgaven og begyndte at kigge applikationens filer igennem. Da applikationen var opdelt i 2 dele, frontend og backend, valgte jeg at lave en Dockerfil for begge. Man kunne godt bruge en enkelt container for hele applikationen i stedet for 2 som jeg gør, men ved at splitte den op i både frontend og backend containere kunne de isoleres fra hinanden. Dette gør for eksempel at hvis backend containeren er nede, og eventuelt i gang med at genstarte, så vil man stadigvæk kunne tilgå selve hjemmesiden. Man ville ikke kunne se produkterne fra databasen men det er trodsalt bedre at man kan komme ind og se på forsiden, end at man overhovedet ikke kan komme ind på hjemmesiden.

Da begge Dockerfilerne blev færdige, kørte jeg Docker build for at bygge de images der skulle bruges til at køre applikationen. Der blev lavet en composefil der skulle starte de to images, samt et allerede eksisterende image for MariaDB databasen og en Docker volume for persistens i forhold til databasen.

Til sidst kørte jeg en Docker swarm for at se om jeg kunne påvirke databasen live, mens swarmen var kørende.

# Step Guide til processen

## Containerization

Der blev først lavet Dockerfiler for henholdsvist frontend og backend. Herefter byggede jeg images på baggrund af disse med:

* Docker build -t frontend .
* Docker build -t backend .

Hvor jeg så først brugte ”cd” kommandoen med en path til frontend mappen og byggede det image, og så derefter gjorde det samme for backend.

Et billede, der indeholder tekst, kvittering, skærmbillede, Font/skrifttype

Automatisk genereret beskrivelse

Da begge images var bygget, kunne der så blive lavet en Docker-compose.yml fil der skal beskrive instruktionerne for at køre alle 3 images.

For at se om hjemmesiden virkede brugte jeg Docker compose kommandoen:

* Docker compose up --detach

Da jeg kunne komme ind på hjemmesiden og at der blev vist produkter i shoppen, vidste jeg at alle 3 images virkede og snakkede sammen.

## Swarm

For at fortælle Docker at jeg vil arbejde med en swarm kørte jeg:

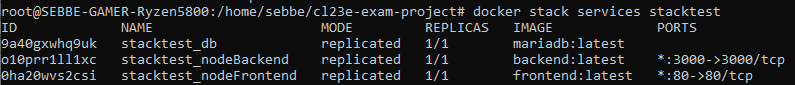
* Docker swarm init

Herefter kan der køres ”stack -deploy” kommandoen for at applikationen bliver deployet i swarmen:

* Docker stack deploy --compose-file Docker-compose.yml stacktest

For at se om applikationen blev deployet ordentligt, bruges kommandoen:

* Docker stack services stacktest



Som det kan ses på ovenstående billede, så blev alle dele af applikation deployet succesfuldt.

Hvis man gerne vil stoppe swarmen kan man bruge kommandoen:

* Docker stack rm stacktest

## Tilføjelse af nye produkter

For at teste om der kan tilføjes produkter til databasen imens at swarmen køre, valgte jeg at attache terminalen direkte til MariaDB databasen. Hvis jeg kunne inserte et nyt testprodukt mens at swarmen kørte og at siden opdaterede produkterne live uden at swarmen skulle genstartes, ville det virke.

For at connecte til vores MariaDB database bruges kommandoen:

* Docker exec -it “containername” mariadb --user root -ppassword[[1]](#footnote-1)

Her stødte jeg ind i det problem at jeg ikke kunne connecte til databasen og fik en ”Access denied for user” error. Jeg fandt så ud af at der ikke skulle være et space imellem -p og ens password, og kunne derefter connecte til databasen hvor jeg så udførte sql commands for at tilføje et nyt testprodukt.

For at tilgå databasen brugte jeg:

* use sample[[2]](#footnote-2);

For at fremvise data fra databasen brugte jeg:

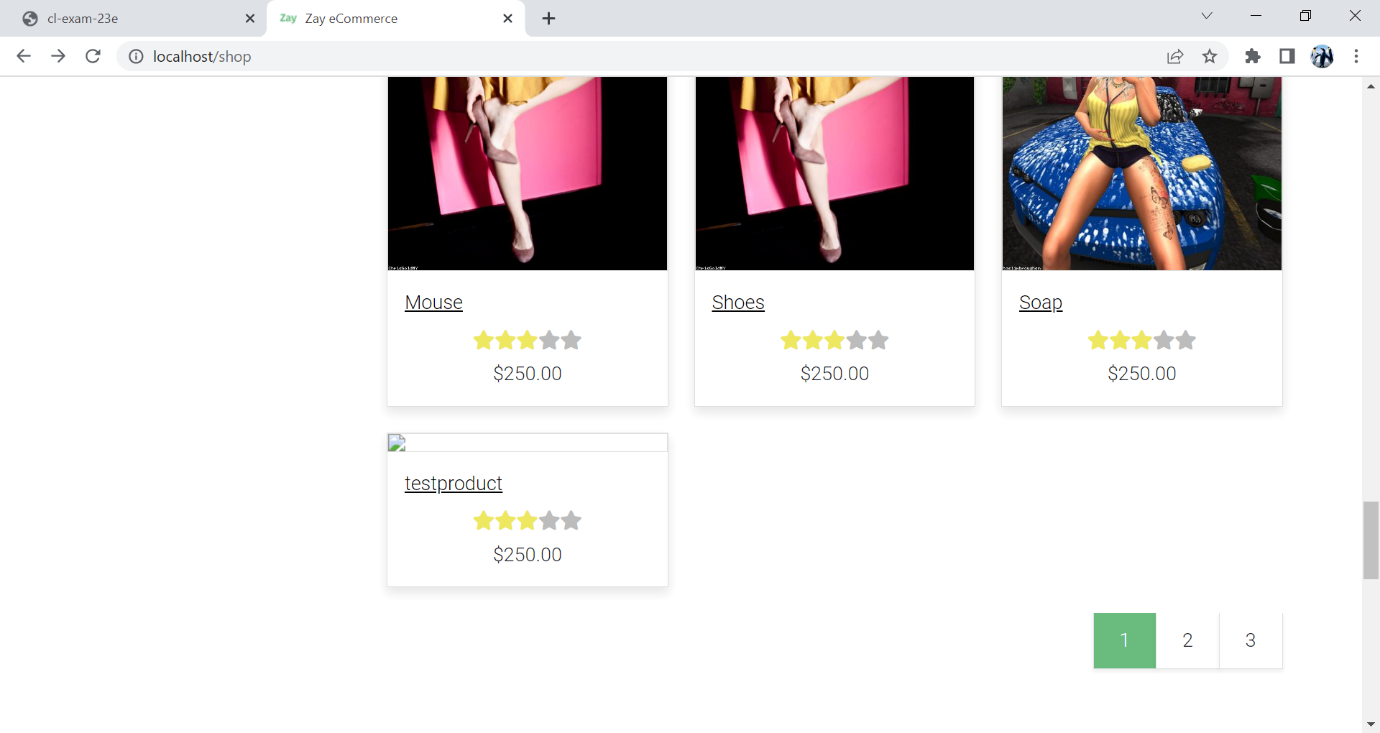
* Select \* from sample;

For at tilføje et nyt produkt brugte jeg:

* insert into Products values (25, 'testproduct', 1337, 5, 'testcompany', 'testdescription', 'XL', 123456789, 7, 'na', '2000-01-01', '2010-02-02');

Hvor de forskellige datatyper for et produkt kunne aflæses i det givne projekt.

Jeg gik så derefter ind på hjemmesiden for at se om den også opdaterede kataloget med produkter, ved at refreshe hjemmesiden:



Som det kan ses på ovenstående billede, er der blevet tilføjet det testprodukt jeg insertede direkte i databasen. Hermed blev der oprettet et produkt i databasen live, imens at Docker swarm var oppe at køre og testen var en succes.

# Konklusion

Igennem dette projekt har jeg lært processen fra at have en applikation som ikke bruger Docker, til at kunne containerize den. Containerization er et ekstremt brugbart redskab der kan bruges til at isolere dele af applikationer. Containere er kun udstyret med det absolut nødvendige filer og indstillinger så de er ekstremt lightweight hvilket gør at man nemt kan bruge dem. Da de selv kommer med deres eget miljø, kan containere køres på alle maskiner. Denne sidste pointe er især vigtig for hvorfor man skulle bruge docker fremadrettet. Hvis jeg skal udvikle et produkt, vil jeg altid kunne have containerization i mente da det kan bruges til at undgå mange af de typiske problemer man støder ind i, og det gælder både i udviklingen og i produktionen.

Docker swarm kan bruges til orchestration, nemlig at man kan uddelegere arbejdet til flere nodes på samme tid, sådan at det ikke kun er en server ikke laver alt arbejdet. Docker swarm vil jeg især kunne benytte mig af hvis jeg arbejder med hjemmesider der skal behandle trafik for mange forskellige bruger på samme tid. Ligesom standard containere vil docker swarm kunne hjælpe mig med at bygge en skalarbar struktur for det projekt jeg arbejder på.

# Guide til at køre applikationen

Byg images for frontend og backend med:

* Docker build -t frontend .
  + I frontend folderen
* Docker build -t backend .
  + I backend folderen

Et billede, der indeholder tekst, kvittering, skærmbillede, Font/skrifttype

Automatisk genereret beskrivelse

Brug en af de nedenstående kommandoer til at køre applikation

* Docker compose up --detach
* Docker stack deploy --compose-file Docker-compose.yml stacktest

# Link til gihub repository:

<https://github.com/sebbe120/cl23e-exam-project>

1. Passworded er defineret i ”Docker-compose.yml” filen. [↑](#footnote-ref-1)
2. ”sample” er navnet på den database der indeholder produkter - også defineret i composefilen. [↑](#footnote-ref-2)