

CL-Exam-23e

UNDERVISER: KENNETH JEPSEN KLAUSEN

FORFATTER: MICHAEL RULLE

Contents

Problem.....	2
Redegørelse.....	2
Docker	2
Docker Swarm	2
Docker volumes.....	2
Analyse	3
Backend.....	3
Frontend.....	4
Valgt fremgangsmåde	4
Dockerfiles.....	4
Database	4
Backend.....	4
Frontend.....	4
Implementering af stacken	5
Frontend.....	6
Backend.....	6
Runonce service.....	7
Database	7
Deployment af stack	8
Skalering af services	9
Brug af logs.....	9
Konklusion.....	10

CL exam 23e

Underviser: Kenneth Jepsen Klausen

Problem

Der skal laves en containeriseret løsning til given kode til en webshop.

Krav er at alle tre komponenter skal køres i containers i en docker swarm, samt at der anvendes docker-volume til persistens for databasen.

Redegørelse

Opgaven er låst, i forhold til docker, swarm og anvendelse af volume.

I dette afsnit følger en beskrivelse af de valgte teknologier, samt fordelene ved dem.

Docker

Docker er en container teknologi, som muliggør kørsel af softwareapplikationer, i et miljø der er isoleret og uafhængigt af det underliggende operativsystem.

for at køre applikationer i docker, skal man bruge et image, som er bygget til applikationen. Dette image fungerer som skabelon til at skabe en container, hvori applikationen køres.

Docker Swarm

Er en orkestreringsværktøj til kørsel af containers i stil med eks. kubernetes. Det bruges til at konfigurere og køre services baseret på dockerimages.

Nogle af fordelene ved at anvende swarm, er at swarm gør det lettere at skalere services i forhold til belastningen, eks. kan man skalere en webshop op ved black Friday, og nedskalere, når trafikken til shoppen igen er normaliseret.

Der er indbygget service discovery, hvilket gør kommunikation mellem services lettere, og uafhængig af at kende ip-adresser for services.

Indbygget sikkerhed; kommunikation mellem noder internt i en docker swarm er krypteret med TLS som standard.

Flere fysiske eller virtuelle maskiner kan kobles på samme swarm.

Docker volumes

Docker volumes er til for at skabe et persistenslag, da data inde i containers forsvinder, sammen med containeren.

Flere containers kan mounte samme volume, for at dele data.

Der kan også deles data med hostsystemet gennem et bind-mount.

Volumes gør det også nemmere at lave backup af data.

Analyse

Da nodejs er anvendt til både frontend og backend, afslører en søgning efter `process.env` de steder, hvor der er anvendt miljøvariabler.

Miljøvariablerne skal/kan bruges til konfiguration af de enkelte services. Alle steder er der angivet standardværdier, hvis der ikke blev fundet tilsvarende miljøvariabler i værtssystemet.

Her ses alle de anvendte miljøvariabler på figuren til højre.

For frontend er der ingen, det kommer vi tilbage til.

Backend

For backend er der, som vist på figuren, en hel del.

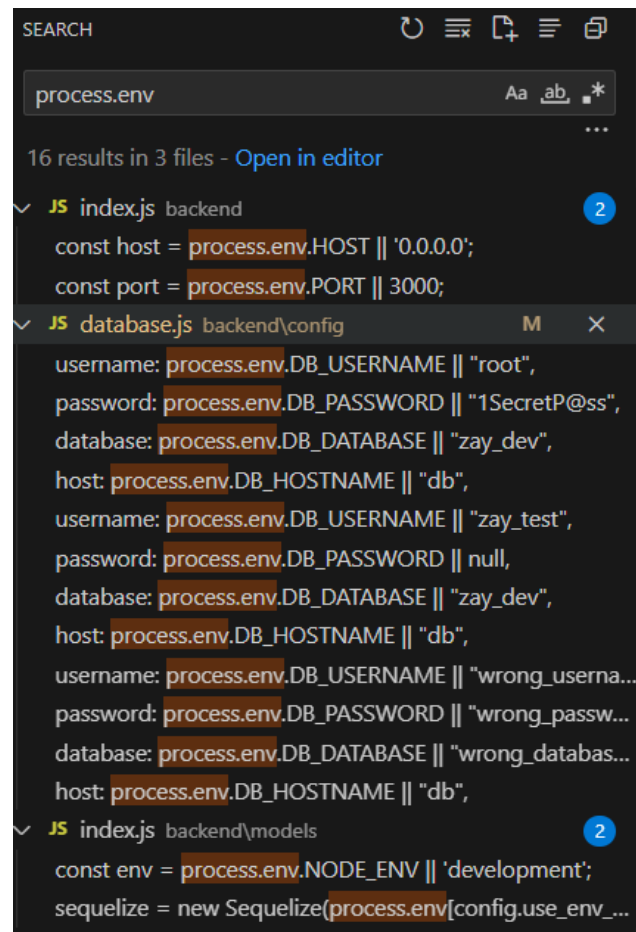
Af disse er `DB_USERNAME`, `DB_PASSWORD` og `DB_DATABASE` helt centrale, da det skal stemme overens med konfiguration af databasen.

`HOST` og `PORT` behøver vi ikke ændre på, da de kun er åbnet på selve containeren.

I `package.json` findes der information relevant for både bygge og deployment process.

```
6  "scripts": {
7    "dev": "nodemon ./",
8    "db:start": "docker run --rm -d -p 3306:3306 -e MARIADB_ROOT_PASSWORD=root -e MARIADB_DATABASE=zay_dev --name zay_db mariadb:latest",
9    "db:stop": "docker stop zay_db",
10   "db:restart": "npm run db:stop && npm run db:start",
11   "db:migrate": "npx sequelize-cli db:migrate --debug",
12   "db:reset": "npx sequelize-cli db:migrate:undo:all && npm run db:migrate && npm run db:seed",
13   "db:seed": "npx sequelize-cli db:seed:all --debug",
14   "start": "node ./",
15   "test": "echo \"Error: no test specified\" && exit 1"
```

Her er angivet scripts til forskellige handlinger, såsom start og stop af database, migrate, seed og hvordan applikationen startes.



Frontend

Konfiguration af frontend er hardcoded i filen `./frontend/src/main.js`, hvilket ikke er god praksis at gøre. Dette kan ændres, ved at indlæse miljøvariabler på samme måde som backend løsningen.

Ud fra strategien om at foretage mindst mulig ændring af kildekoden, er de hardcodede værdier ændret, så de stemmer overens med de porte der senere vil blive åbnet i docker konfigurationen.

Ligesom for backend, findes der i **package.json** findes der information relevant for både bygge og deployment process.

I backend er der også en `./backend/config/database.js` fil, der giver information omkring miljøvariabler der anvendes til forbindelse til databasen, ligesom der er angivet hvilken type af database der er anvendt.

Valgt fremgangsmåde

Ud fra den indledende analyse vil løsningen bestå af 3 images der skal bygges og køres i en docker stack, hvoraf kun frontend og backend vil have redundans og skalerbarhed.

Det vil kræve betydeligt mere arbejde, at kunne skalere databasen i forhold til konfiguration.

Der er dog stadig fordele, ved at køre databasen i en container, blandt andet kan der laves netværkssegregering, for at isolere databasen fra de udstillede containers frontend og backend, ligesom den kan overvåges og automatisk genstartes, hvis den fejler.

Dockerfiles

Der skal skrives to stk Dockerfile, en til frontend, og en til backend.

Databasen kræver kun miljøvariabler, for at kunne køres og bruges af backend.

Database

Kræver intet nyt bygget oven på image fra https://hub.docker.com/_/mariadb

Backend

Backend bliver bygget ovenpå en **node:20.3.0** med en kopi af kildekoden og **package.json**.

Som vist på figuren til højre, er det valg af base image, kopier kildekode, sæt working directory, installer dependencies, åbn port, som angivet som standard i `./backend/index.js` og kørs applikationen.

```
FROM node:20.3.0
COPY . ./app
WORKDIR /app
RUN ["npm", "install"]
EXPOSE 3000
CMD [ "npm", "run", "start" ]
```

Backend image bygges med følgende kommando: ``docker build . -t backend:0.0.1``

hvor `.` angiver mappen hvor den givne Dockerfile findes, `-t` angiver "tag" for det resulterende image, med efterfølgende versionsnummer.

Frontend

Frontend komponenten vil blive bygget som et **multistage-build**, da vite build processen omdanner kildekoden til statiske filer, som egner sig til hosting gennem en reverse proxy.

Reverse proxy

Det sker ved at bygge 2 images med hvert sit base-image, i samme fil. Det første image er baseret på **node:20.3.0** og anvendes til at bygge produktionskoden. Det resulterende image bygger på **nginx:1.25.0-alpine**, hvor den resulterende `./app/dist` mappe fra første image vil blive kopieret til.

Under processen med at bygge og teste frontend løsningen, opstod der et problem, der fremstod som et CORS relateret problem. Men efter en undersøgelse af backend koden og requests i browseren, så det ud til at frontend forsøgte at kontakte backend på en forkert adresse. Det viste sig at der manglede configuration for produktionsmiljøet i `./frontend/src/main.js`, og efter den tilføjelse, virkede forbindelsen.

```
# build builder image
FROM node:20.3.0 AS builder
COPY . ./app
WORKDIR /app
RUN ["npm", "install"]
RUN ["npm", "run", "build"]

# build image
FROM nginx:1.25.0-alpine
COPY --from=builder /app/dist /usr/share/nginx/html
EXPOSE 80
CMD [ "nginx", "-g", "daemon off;" ]
```

Frontend image bygges med følgende kommando: ``docker build . -t frontend:0.0.1``

Implementering af stacken

Sideløbende med configuration af images, bliver **compose filen** bygget, med konfiguration i form af servicenavne, navne på images, miljøvariabler, port-forwarding og netværkssegregering, for at teste stacken undervejs.

Al configuration af komponenterne foretages gennem **PROD_stack.yml**.

For både frontend og backend, gælder det, at replicas er sat til 1 som default ved deployment.

Under udviklingen af compose filen, dukkede denne note op, i dockers dokumentation:

- There are several things to be aware of when using `depends_on` :**
- `depends_on` does not wait for `db` and `redis` to be "ready" before starting `web` - only until they have been started. If you need to wait for a service to be ready, see [Controlling startup order](#) for more on this problem and strategies for solving it.
 - The `depends_on` option is ignored when [deploying a stack in swarm mode](#) with a version 3 Compose file.

Figur 1 (Docker.com, 2023)

Det viste sig, at man ikke kan tvinge services til at starte op i en bestemt rækkefølge, når man kører dem i en swarm, hvilket ellers var en del af planen.

Frontend

Frontend krævede ikke noget særlig konfiguration, udover den der var hardcoded i `./src/main.js`

På figuren vises udsnit af compose filen, hvor frontend konfigurationen.

Her udstilles port 8000 på host systemet, og forwardes internt til frontend på port 80.

Her skal bemærkes, at frontend kun er tilsluttet det interne docker netværk med navnet **public**.

Dett er for at undgå, at frontend og database er på samme netværk.

```
frontend:
  image: "frontend:0.0.1"
  ports:
    - "8000:80"
  deploy:
    mode: replicated
    replicas: 1
    restart_policy:
      condition: on-failure
      delay: 5s
      max_attempts: 5
      window: 10s
  networks:
    - public
```

Backend

Konfiguration af backend er som vist på figuren.

```
backend:
  image: "backend:0.0.1"
  environment:
    DB_USERNAME: zay_prod
    DB_PASSWORD: user123
    DB_DATABASE: zay_prod
    DB_HOSTNAME: db
  command: '/bin/sh -c "node index.js"'
  deploy:
    mode: replicated
    replicas: 1
    restart_policy:
      condition: on-failure
      delay: 5s
      max_attempts: 5
      window: 10s
  ports:
    - "8080:3000" # for at teste det udefra
  networks:
    - public
    - private
```

Her sættes DB_USERNAME og DB_PASSWORD, som skal være identisk med MARIADB_USERNAME og MARIADB_PASSWORD i konfiguration for databasen.

Backend er på to separate interne netværk, for at kunne kontakte databasen på netværket **private**, og kontaktes af frontend på netværket **public**.

Eksterne port på host systemet er 8080, som mapper til port 3000, som fundet i `./backend/index.js`, på det interne netværk.

Runonce service

For at initialisere databasen med tabeller og data, er der lavet en runonce service, der skal køres en gang, og **KUN** en gang, da der ellers vil opstå duplikering af data i databasen.

Den er konfigureret til ikke at være replikeret fra start, ligesom den heller ikke må genstartes automatisk.

```
runonce:
  image: "backend:0.0.1"
  environment:
    DB_USERNAME: zay_prod
    DB_PASSWORD: user123
    DB_DATABASE: zay_prod
    DB_HOSTNAME: db
    NODE_ENV: production
  command: '/bin/sh -c "npm run db:migrate && npm run db:seed"'
  deploy:
    mode: replicated
    replicas: 0
    restart_policy:
      condition: none
  networks:
    - private
```

Database

For at kunne bruge databasen, skal der angives 4 miljøvariabler:

MARIADB_DATABASE, MARIADB_ROOT_PASSWORD, MARIADB_USER og MARIADB_PASSWORD

Den første skal være identisk med DB_DATABASE i konfigurationen for backend, ligesom de to sidste skal være identiske med hhv. DB_USERNAME og DB_PASSWORD i konfiguration for backend.

Når de miljøvariabler er sat, bliver schema oprettet af **docker-entrypoint.sh** script, som er angivet som **ENTRYPOINT** i mariadb image.

Og da backend koden har både migrate og seed funktionalitet indbygget, er der ikke behov for at kopiere sql-script ind, for at lave tabellerne.

```
# IMPORTANT: may NOT be scaled, because instances will share the same docker-volume for
persistence

db:
  image: "mariadb:10.6.14"
  ports:
    - "3306:3306"
  networks:
    - private
  volumes:
    - db_persistence:/var/lib/mysql:rw
  environment:
    MARIADB_DATABASE: zay_prod
    MARIADB_ROOT_PASSWORD: root123
    MARIADB_USER: zay_prod
    MARIADB_PASSWORD: user123
  healthcheck:
    test: mysqladmin ping -h 127.0.0.1 -u $$MARIADB_USER --password=$$MARIADB_PASSWORD
    start_period: 5s
    interval: 5s
    timeout: 5s
    retries: 55
  deploy:
    # since there is only one node in the current swarm, this prevents scaling of db
    # and in the future, it prevents multiple databases to be run on same node in the swarm

    mode: global
    restart_policy:
      condition: on-failure
      delay: 10s
      max_attempts: 5
      window: 40s
```

Databaseservicen er kun på det interne docker netværk med navnet **private**, for at databasen ikke kan tilgås ude fra, derfor er der heller ikke mappet nogle porte ind til databasen fra det underliggende hostsystem.

Deployment af stack

For at kunne deploy som en stack, skal der oprettes en swarm.

Forudsat at der kører en docker service på maskinen (eks. docker-ce eller docker desktop), initieres en swarm med denne kommando: **`docker swarm init`**.

Nu er docker swarm oprettet, og maskinen kommandoen blev kørt på, er en manager node.

Der kan tilføjes flere nodes, ved brug af den token som den ovennævnte kommando returnerer.

For at deploy de konfigurerede services køres følgende kommando: **`docker stack deploy -c {sti til compose fil} {navn på stack}`**
 eksempelvis **`docker stack deploy -c ./PROD_stack.yml exam`**, hvilket giver en stack med navnet **exam** dette skal bruges ved evt. senere skalering.

Systemet er dog ikke funktionelt endnu, da databasen ikke er initialiseret endnu.

For at initialisere databasen, skaleres runonce service først op til 1, for derefter at skalere den ned til 0 igen. Denne process sikrer, at databasen kun bliver initialiseret en gang.

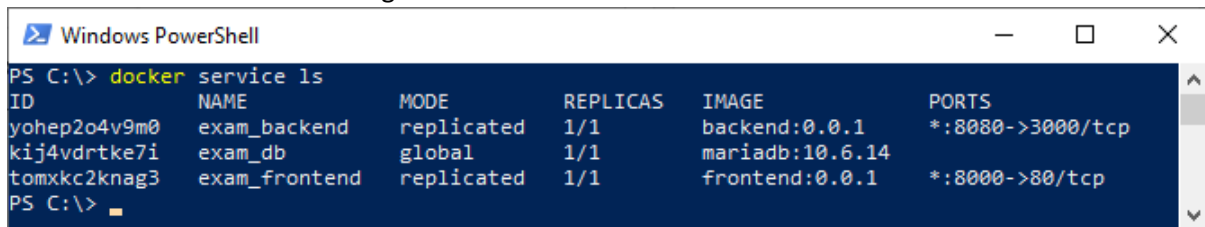
Skalering af services

I tilfælde af man har brug for flere instanser af frontend eller backend, kan man skalere dem med følgende kommando på en manager node:

`docker service scale {navn på service}={antal ønskede instanser}`

eksempelvis: **`docker service scale exam_frontend=3`**, det vil (fra default 1), øge antallet af frontend instanser til 3.

Man kan finde services ved brug af **`docker service ls`**, der kan man blandt andet se navn på services, som de skal skrives ved skalering osv.



```

PS C:\> docker service ls
ID                NAME                MODE                REPLICAS    IMAGE                PORTS
yohep2o4v9m0     exam_backend        replicated          1/1         backend:0.0.1       *:8080->3000/tcp
kij4vdr7ke7i     exam_db             global              1/1         mariadb:10.6.14
tomxkc2knag3     exam_frontend       replicated          1/1         frontend:0.0.1      *:8000->80/tcp
PS C:\>
  
```

Brug af logs

I tilfælde af man ikke bruger docker desktop, kan logs for de enkelte services tilgås vha.

Kommandoen **`docker service logs -f {navn på service}`**

Konklusion

Den valgte fremgangsmåde virker som den skal, dog med den begrænsning, at databasen ikke er skalerbar. Hvis den skales vil det medføre uhensigtsmæssige konsekvenser, eksempelvis race-conditions i forhold til skrivning til den delte docker-volume.

Der er testet for skalerbarhed fra 1 til 3 replicas for både frontend og backend, hvilket har været succesfulde.

Der er testet, at metoden med runonce servicen initialiserer databasen, og at den kun kører en gang, såfremt den beskrevne fremgangsmåde følges.

Der findes løsninger til skalering af databaser, men indenfor den fastsatte tidsramme, har det ikke været muligt at gennemføre den nødvendige research og efterfølgende implementering.

~~Der skal bygges endnu en container, som en "run-only-once", til at køre migrate og seed af databasen. Som det er nu, vil de to tasks blive kørt, hver gang backend replicas skales. Docker-volume for databasen skal ligeledes slettes, hvis man redeployer stacken, da migrate og seed tasks køres, når man redeployer løsningen.~~

Man bør også overveje at undersøge muligheden, for at anvende eks. docker-config og docker-secrets. Det har heller ikke været muligt indenfor den givne tidsramme.

I produktion skal konfiguration af frontend ændres, sådan at værdien for **api** sættes til url for det domæne, eller den IP, som løsningen deployes til, for at der er kontakt mellem frontend og backend.