

Beskrivelse:

Good practice er, som jeg har nævnt før, KUN at have en service pr. container. Det har vi jo ikke i vores crypto container fra forrige opgave. Så det går ikke. Det må vi hellere få rettet op på. Så her i opgave 2. vil vi stadig have vores crypto side, men med php og nginx webserver i hver sin container!

Kapitel 1: Installér Docker.

Først er det en god ide at lave et underbibliotek, hvor vi kan have alle vores data. Så kan vi holde det separat fra den forrige opgave. Det er også best-practice at have hver enkel opgave man skal lave med docker, i hver sit underbibliotek.

Det første vi skal, er at lave underbiblioteket, vi vil kalde det cryptosep, for crypto sepperat. HUSK at stå i din brugers home bibliotek, inden du starter (dette kan gøres ved at skrive `cd ~`) vi sætter begge kommandoer ind samtidig:

```
mkdir cryptosep  
cd cryptosep
```

Forklaring:

Første kommando laver biblioteket. Den anden kommando hopper ned i biblioteket.

Eksempel output:

```
dtmek@docker2:~$ mkdir cryptosep  
cd cryptosep  
dtmek@docker2:~/cryptosep$ █
```

Vi fortsætter med at lave forberedelser. Nu skal vi lige have lavet et underbibliotek hvori vi placerer konfigurationen for nginx (Det er en webserver):

```
mkdir nginx  
cd nginx
```

Eksempel output:

```
dtmek@docker2:~/cryptosep$ mkdir nginx  
cd nginx  
dtmek@docker2:~/cryptosep/nginx$ █
```

Et par konfigurations filer skal der også laves. Den første fil vi laver her ved hjælp af en teksteditor (nano), er konfigurationsfilen til webserveren (nginx), og da det ikke er et kursus i nginx, vil jeg ikke gennemgå filen:

```
nano default.conf
```

Indhold af filen på næste side:

Filen skal indeholde følgende (Kopier teksten herunder, og sæt det ind i dit SSH vindue ved at højreklikke i vinduet):

```
##### Default.conf start #####
server {

    listen 80 default_server;
    root /var/www/html;
    index index.html index.php;

    charset utf-8;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location = /favicon.ico { access_log off; log_not_found off; }
    location = /robots.txt { access_log off; log_not_found off; }

    access_log off;
    error_log /var/log/nginx/error.log error;

    sendfile off;

    client_max_body_size 100m;

    location ~ .php$ {
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass php:9000;
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_read_timeout 300;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_intercept_errors off;
        fastcgi_buffer_size 16k;
        fastcgi_buffers 4 16k;
    }

    location ~ /\.ht {
        deny all;
    }
}
##### default.conf slut #####
```

Gem filen ved at trykke "Ctrl+x" -> trykke "Y" -> Tryk Enter ved filnavn. Og du er tilbage til normal prompt på Ubuntu.

Ingen Eksempel output.

Nu skal vi lave en fil vi har stiftet bekendtskab med før. Nemlig en Dockerfile (med stort D!). så vi starter lige nano editoren igen:

nano Dockerfile

Der skal selvfølgelig også noget indhold i denne fil:

```
##### Dockerfile Start #####  
FROM nginx  
COPY ./default.conf /etc/nginx/conf.d/default.conf  
##### Dockerfile Slut #####
```

Gem filen ved at trykke "Ctrl+x" -> trykke "Y" -> Tryk Enter ved filnavn. Og du er tilbage til normal prompt på Ubuntu.

Ingen Eksempel output.

Vi har jo set på en Dockerfile i opgave 1 før. Så ud fra det kan vi se at vores parrent image, er et image der hedder **nginx** (og her skal det indskydes igen at standard vil den søge efter det image på hjemmesiden docker.io, og det er versionen latest den søger efter. For docker ser det ud som om der står **FROM nginx:latest**).

Derefter bliver filen **default.conf** kopieret over til imaget. Det er faktisk en nginx config file, den fil vi lavede lige før.

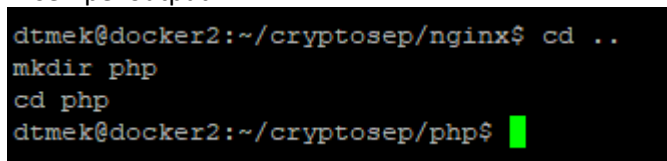
Så er vi færdig med forberedelserne til nginx imaget. Nu skal have lavet forberedelserne til php imaget. Vi starter med at hoppe et trin tilbage og lave et nyt bibliotek. (3 kommandoer på en gang denne gang):

```
cd ..  
mkdir php  
cd php
```

Forklaring:

Første kommando hopper et bibliotek op, anden kommando laver biblioteket, tredje kommando hopper ned i biblioteket

Eksempel output:



```
dtmek@docker2:~/cryptosep/nginx$ cd ..  
mkdir php  
cd php  
dtmek@docker2:~/cryptosep/php$
```

Vi skal også her lave en Dockerfile (HUSK Stort D!!):

nano Dockerfile

Indholdet af filen på næste side.

Indholdet i denne fil er:

```
##### Dockerfile Start #####  
FROM php:8.2-fpm  
RUN docker-php-ext-install mysqli pdo pdo_mysql  
RUN docker-php-ext-enable mysqli  
#####s Dockerfile Slut #####
```

Gem filen ved at trykke "Ctrl+x" -> trykke "Y" -> Tryk Enter ved filnavn. Og du er tilbage til normal prompt på Ubuntu.

Dette er også en Dockerfile, og herfra kan vi se at vores parrent image er: **php:8.2-fpm** og i dette image bliver der udført nogen kommandoer (RUN).

Så har vi forberedt de 2 konfigurationer, der skal bruges til en webserver (nginx) og php. Vi skal dog også lige have kopieret vores Site over til vores nye projekt. Husk at hoppe et bibliotek tilbage fra php, inden du fortsætter (**cd . .**):

```
cp -r ../crypto/Site/ ./Site/
```

Forklaring:

Kommando kopierer vores "Site" over til aktuel underbibliotek (cryptosep). **-r** = rekursiv, tager alt med i underbiblioteket.

Eksempel output:

```
dtmek@docker2:~/cryptosep$ cp -r ../crypto/Site/ ./Site/  
dtmek@docker2:~/cryptosep$ █
```

Har computeren mon gjort det? Eller har den i det hele taget lavet noget? Lad os lige se hvad der ligger i vores bibliotek:

```
ls -l
```

Forklaring:

Viser indhold af aktuel bibliotek. Og ja det er små L'er begge streger

Eksempel output:

```
dtmek@docker2:~/cryptosep$ ls -l  
total 12  
drwxrwxr-x 2 dtmek dtmek 4096 Nov  8 11:22 nginx  
drwxrwxr-x 2 dtmek dtmek 4096 Nov  8 12:05 php  
drwxrwxr-x 2 dtmek dtmek 4096 Nov  8 12:17 Site  
dtmek@docker2:~/cryptosep$ █
```

Yep! Det er blevet kopieret over. Vi skal lige have sat noget sikkerhed på vores nye "Site", vi lige har kopieret over. Sidst blev sikkerheden jo sat da vi kopierede "Site" over i vores image. Men det gør vi ikke dengang. Denne gang er "Site" placeret på vores host. Så vi sætter lige sikkerheden. Normalt vil man selvfølgelig kun sætte rettigheder på de filer der skal bruge de respektive rettigheder, men det er nemmest at sætte rettighederne til alle filer til windows ækvivalenten "everyone full control":

```
chmod 777 -R Site
```

Forklaring:

Linux commando der, i windows udtryk, sætter fuld kontrol til alle (-R = rekursiv, også undermapper, og filer).

Eksempel output:

```
dtmek@docker2:~/cryptosep$ chmod 777 -R Site
dtmek@docker2:~/cryptosep$ █
```

Vi må hellere lige se om det kan ses på vores list kommando. Det kan det faktisk godt. På farve, og yderst til venstre i sikkerhed.

```
ls -l
```

Eksempel output:

```
dtmek@docker2:~/cryptosep$ ls -l
total 12
drwxrwxr-x 2 dtmek dtmek 4096 Nov  8 11:22 nginx
drwxrwxr-x 2 dtmek dtmek 4096 Nov  8 12:05 php
drwxrwxrwx 2 dtmek dtmek 4096 Nov  8 12:17 Site
dtmek@docker2:~/cryptosep$ █
```

Slut på Kapitel 1: Installér Docker.

Kapitel 2: Opret docker-compose.yml.

Nu skal vi bare finde ud af hvordan man så laver flere images på en gang, og måske også starter alle containere på samme tid? Selvfølgelig kan det lad sig gøre med Docker.

Vi bruger bare en fil der hedder **docker-compose.yml** og til afveksling SKAL alle bogstaver være med små denne gang... JA. Jeg ved det godt... men sådan er Linux!...

Jeg vil også denne gang gennemgå hver kommando for sig, og til sidst i gennemgangen kommer oprettelsen af hele filen.

Lad os starte med gennemgang af filen (Læg mærke til indrykningerne. De SKAL være der):

services:

Forklaring:

Dette indikerer at vi vil oprette en liste med services (images).

nginx:

Forklaring:

Dette giver et navn på det afsnit vi kommer til (det enkelte image, hvis vi ikke specifik ændrer navnet, som i næste kommando, bliver dette også navnet på imaget):

image: ip_Registry_Server:5000/web:latest

Forklaring:

Her sætter vi vores image navn. Vi kan med det samme indikere hvilken registry server den skal ligge på og hvad navnet skal være. Dvs. vi tagger allerede vores image. Vi kunne bare have skrevet "web" så havde imaget heddet: "dit_maskinenavn-web". Men hvis vi ikke sætter vores registry server ind i imagenavnet, kan vi ikke pushe imaget til vores registry.

build: ./nginx/

Forklaring:

Hvor ligger vores byggevejledning til imaget? Det er det samme som placeringen af vores Dockerfile til vores nginx image. (det lagde vi jo i underbiblioteket nginx, da vi forberedte.)

container_name: web

Forklaring:

Hvad vil vores container hedde? (når vi kigger på den med docker ps)

hostname: web

Forklaring:

Hvad vil vores container hedde på vores interne docker netværk (se opgave 3).

```
ports:
  - 80:80
```

Forklaring:

Hvilke porte skal være åbne i containeren? Her er Port 80 på Host overført til port 80 på Container, efter formatet Port_på_Host:Port_På_Container. Man kan sagtens have flere porte åben, de listes så bare under de allerede nævnte porte (dette svarer til `-p 80:80` på `docker run`).

```
volumes:
  - ./Site:/var/www/html/
```

Forklaring:

Dette mounter (Tilslutter) stien `./Site/` på Host til stien `/var/www/html/` på container, efter formatet Sti_påHost:Sti_på_container.

```
depends_on:
  - php
```

Forklaring:

Dette fortæller at dette image skal bruge det image vi kalder php. Det er defineret under php: (Lige herunder).

```
php:
```

Forklaring:

Dette giver et navn på det afsnit vi kommer til (det enkelte image, hvis vi ikke specifik ændrer navnet som i næste linje, bliver dette også navnet på imaget) Dette afsnit hedder `php`, det image vi skal bruge til det vi lige har defineret under `depends_on` i forrige afsnit:

```
image: 192.168.1.131:5000/php:latest
```

Forklaring:

Her sætter vi vores image navn. Vi kan med det samme indikere hvilken registry server den skal ligge på ligesom sidst og hvad navnet skal være. Dvs. vi tagger allerede vores image igen.

```
build: ./php/
```

Forklaring:

Hvor ligger vores byggevejledning til php imaget? Det samme som placeringen af vores Dockerfile til vores php image. (det lagde vi jo i underbiblioteket php, da vi forberedte.)

```
container_name: php
```

Forklaring:

Hvad vil vores container hedde? (når vi kigger på den med `docker ps`)

```
hostname: php
```

Forklaring:

Hvad vil vores container hedde på vores interne docker netværk (se opgave 3)

```
expose:  
- 9000
```

Forklaring:

Hvilke porte skal være åbne i containeren? Her er Port 9000 åben på container, men ikke mappet til host, så det er kun vores anden container der kan se denne container. Vi kan ikke kommunikere med denne container direkte fra hosten.

```
volumes:  
- ./Site/:/var/www/html/
```

Forklaring:

Dette mounter (Tilslutter) stien **./Site/** på Host til stien **/var/www/html/** på container, efter formatet **Sti_påHost:Sti_på_container**. Læg mærke til at det samme bibliotek der blev mountet under "web:" afsnittet. Det er så det er muligt for php container kan få adgang til den php kode der skal udføres, der er på vores webside.

Det var så vores docker-compose.yml gennemgangen. Nå må vi hellere også oprette filen på vores server (HUSK alle bogstaver med småt).

nano docker-compose.yml

Indholdet til filen på næste side.

Indholdet i denne fil er (HUSK at ændre `ip_Registry_server` til din registry maskines ip adresse (Fremhævet) Og **VIGTIGT!!!** At alle indrykninger skal være som herunder!!! Evt. kopier filen fra det materiale der kan downloades:

```
##### docker-compose.yml start #####
#"Script" Der laver web container, og PHP container.

# Hvilke Services skal laves
services:
  # Lav Web server (nginx)
  nginx:
    #Tag på dette image (Skal bruges til Docker Compose Push/pull)
    image: ip_Registry_server:5000/web:latest
    # Hvor ligger "byggevejledning" (Dockerfile)
    build: ./nginx/
    # Navne
    container_name: web
    hostname: web
    # Hvilke porte skal bruges efter formatet HostPort : ContainerPort
    ports:
      - 80:80
    # Hvilke directories skal mountes til Container efter formatet
    HostDirectory : ContainerDirectory
    volumes:
      - ./Site/:/var/www/html/
    depends_on:
      - php

  # Lav en PHP service
  php:
    #Tag på dette image (Skal bruges til Docker Compose Push/pull)
    image: ip_Registry_server:5000/php:latest
    # Hvor ligger "byggevejledning" (Dockerfile)
    build: ./php/
    # Navne
    container_name: php
    hostname: php
    # Hvilke porte skal være åben på Container (Kan IKKE ses på Host,
    kun Docker netværk)
    expose:
      - 9000
    # Hvilke directories skal mountes til Container efter formatet
    HostDirectory : ContainerDirectory
    volumes:
      - ./Site/:/var/www/html/

##### docker-compose.yml slut #####
```

Gem filen ved at trykke "Ctrl+x" -> trykke "Y" -> Tryk Enter ved filnavn. Og du er tilbage til normal prompt på Ubuntu.

Ingen Eksempel output.

Slut på Kapitel 2: Opret docker-compose.yml.
Kapitel 3: Start vores containere.

Nu har vi lavet et "script" der viser computeren, hvordan den skal lave vores 2 containere. Men vi skal jo også lige fortælle computeren, at den skal "bygge" vores containere! Vi bygger vores containere ved at skrive:

```
docker compose up -d
```

Forklaring:

Docker compose: Dette er kommandoen til at bygge.

Up: Betyder start imagerne når de er bygget, så de bliver containere.

-d: Detach = frigiv SSH prompt efter udførelse.

Eksempel output:

Der kommer meget skriv, og der kommer et par advarsler i begyndelsen. Dette betyder bare at den ikke kan finde vores images på den server vi har specificeret under **image**:

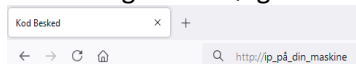
```
! nginx Warning manifest for 192.168.1.131:5000/web:latest not found: manifest unknown: manifest unknown
! php Warning manifest for 192.168.1.131:5000/php:latest not found: manifest unknown: manifest unknown
```

Derefter kommer der en hel masse mere tekst, det skulle gerne ende ca. sådan her, som man kan se er de to containere startet, og vores interne netværk som de skal køre på er oprettet, og også aktiv. Netværk kommer vi til i opgave 3:

```
=> => naming to 192.168.1.131:5000/web:latest
=> [nginx] resolving provenance for metadata file
[+] Running 3/3
✔ Network cryptosep_default Created
✔ Container php Started
✔ Container web Started
dtmek@docker2:~/cryptosep$
```

Det må jo betyde at vores containere kører, og at vi kan se vores hjemmeside. Lad os besøge vores, efterhånden velkendte hjemmeside. I din browser skriver du **http://ip_på_worker_machine**

Du skulle gener få følgende side frem:



TOP SECRET PAGE!!!

Tryk på en af følgende opgaver:

- [1. Kod din besked](#)
- [2. DEkod din besked](#)

Lad os se hvad der kører af containere.

```
docker ps
```

Eksempel output:

```
dtmek@docker2:~/cryptosep$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
fe7635c4dd59   192.168.1.131:5000/web:latest       "/docker-entrypoint..." 30 minutes ago Up 30 minutes 0.0.0.0:80->80/tcp, :::80->80/tcp   web
79577d33cf7a   192.168.1.131:5000/php:latest       "docker-php-entrypoi..." 30 minutes ago Up 30 minutes 9000/tcp                               php
dtmek@docker2:~/cryptosep$
```

Der kører 2 containere, og de hedder jo det vi bedte om at de kom til at hedde, nemlig

```
ip_Registry_server:5000/web:latest og ip_Registry_server:5000/php:latest
```


Skulle vi ikke lige se, om der ikke er et par images også:

```
docker image ls -a
```

Eksempel output:

```
dtmek@docker2:~/cryptosep$ docker image ls -a
REPOSITORY          TAG          IMAGE ID       CREATED        SIZE
192.168.1.131:5000/web  latest      b555959952fe  40 minutes ago  192MB
192.168.1.131:5000/php  latest      507301ef0134  40 minutes ago  493MB
dtmek@docker2:~/cryptosep$
```

Der er de jo! De images vi forventer der er der. Og som vi kan se af navnet er, de allerede tildelt vores registry. Skulle vi så ikke prøve at pusher dem ud på vores registry server? Der burde jo ikke komme nogen fejl, vi har jo allerede indikeret hvilket registry de skal bruge. Nemlig det samme registry som sidst.

Slut på Kapitel 3: Start vores containere.

Kapitel 4: Pushing og Pulling.

Vi behøver ikke at pushe hvert image for sig. Det er der nemlig også blevet tænkt over i docker. Vi skriver:

```
docker compose push
```

Forklaring:

Docker compose: Dette er kommandoen, vi kigger på docker-compose.yml som en samlet enhed.

Push: Skub data ud til vores registry.

Eksempel output:

```
dtmek@docker2:~/cryptosep$ docker compose push
[+] Pushing 9/21
  ✓ Pushing 192.168.1.131:5000/web:latest: 067dlb3db653 Pushed
  ✓ Pushing 192.168.1.131:5000/web:latest: e4e9e9ad93c2 Pushed
  ✓ Pushing 192.168.1.131:5000/web:latest: 6ac729401225 Pushed
  ✓ Pushing 192.168.1.131:5000/web:latest: 8ce189049cb5 Pushed
  ✓ Pushing 192.168.1.131:5000/web:latest: 296af1bd2844 Pushed
  ✓ Pushing 192.168.1.131:5000/web:latest: e3d70e983cd5 Pushed
  ✓ Pushing 192.168.1.131:5000/web:latest: b33db0c3c3a8 Pushed
  ! Pushing 192.168.1.131:5000/web:latest: 98b5f35ea9d3 Mounted from crypto
  ✓ Pushing 192.168.1.131:5000/php:latest: d6ae079b4a0f Pushed
  ✓ Pushing 192.168.1.131:5000/php:latest: b2c980019c2a Pushed
  ! Pushing 192.168.1.131:5000/php:latest: d6493f599638 Mounted from crypto
  ! Pushing 192.168.1.131:5000/php:latest: 5f70df18a086 Mounted from crypto
  ! Pushing 192.168.1.131:5000/php:latest: 935f085de609 Mounted from crypto
  ! Pushing 192.168.1.131:5000/php:latest: 324ad0c90d72 Mounted from crypto
  ! Pushing 192.168.1.131:5000/php:latest: abe69ea2e98f Mounted from crypto
  ! Pushing 192.168.1.131:5000/php:latest: 2951bbe4b953 Mounted from crypto
  ! Pushing 192.168.1.131:5000/php:latest: 39e20ac1f5ff Mounted from crypto
  ! Pushing 192.168.1.131:5000/php:latest: 8db5f32ea418 Mounted from crypto
  ! Pushing 192.168.1.131:5000/php:latest: efe825alle22 Mounted from crypto
  ! Pushing 192.168.1.131:5000/php:latest: 024a1ae60757 Mounted from crypto
  ! Pushing 192.168.1.131:5000/php:latest: 98b5f35ea9d3 Mounted from web
dtmek@docker2:~/cryptosep$
```

Som man kan se er det faktisk kun vores nyeste image "nginx" der er blevet overført. Php havde vi jo allerede fra vores sidste crypto, og da vi jo ændrede vores php image ved at tilføje layers, er vores originale uændrede image stadig på registry. Så vi udnyttede at vi kun behøvede at meddele server hvad vi evt. har ændret, og det er igen kun det den overfører.

Vi skal jo slette alt igen fra vores worker for at prøve at pulle. Kan vi så også stoppe vores containere af en omgang? Yep! det kan vi:

```
docker compose down
```

Forklaring:

Lukker alle containere defineret i docker-compose.yml ned, og lukker også det oprettede netværk ned.

Eksempel output:

```
dtmek@docker2:~/cryptosep$ docker compose down
[+] Running 3/3
  ✓ Container web          Removed
  ✓ Container php         Removed
  ✓ Network cryptosep_default Removed
dtmek@docker2:~/cryptosep$
```

Hvad nu? Computeren skriver ikke kun stoppet, den skriver removed? Har den mon fjernet vores containere også? Lad dos straks se efter:

```
docker container ls -a
```

Eksempel output:

```
dtmek@docker2:~/cryptosep$ docker container ls -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
dtmek@docker2:~/cryptosep$
```

YES! Alle containere er også væk. Det gør det jo nemmere for os, når vi nu skal have alt væk, for at kunne prøve at pulle. Så vi nøjes med at slette images, og prune systemet. Det kan gøres på følgende måde: Kopier begge kommandoer over i et notepad dokument. Ret de steder der skal rettes. Kopier alt igen. Og indsæt alt det rettede. (Husk at rette det fremhævede.):

```
docker image rm ip_Registry_server:5000/web ip_Registry_server:5000/php
docker system prune -af
```

Forklaring:

Første linje slette de to images vi har generet med vores docker-compose.yml fil.

Anden linje renser systemet. (-af: -a=all. -f = force= tager alt.)

Eksempel output

```
dtmek@docker2:~/cryptosep$ docker image rm 192.168.1.131:5000/web 192.168.1.131:5000/php
docker system prune -af
Untagged: 192.168.1.131:5000/web:latest
Deleted: 192.168.1.131:5000/web@sha256:243344772151400762f6df612ccc4312fe6ebc296f1627910d9ec2d9886464e
Deleted: sha256:855395932f490df22eefc23883f63999707b26fc8ab4327707bd0e0955
Deleted: sha256:5a41d8dc705e770183e9d8fba22ec6782e732b09346d06742a04933aebd1ed
Deleted: sha256:3e8a4396bcb26aeb916e1e4c6f4500038080839f049c490c256742dd842334
Deleted: sha256:8dd6a711fbd252eba01f96d30aa132c4b4e96961f9010fbbdb11869865f994
Deleted: sha256:93605219820e9f897f3ce4770afdbabb3b1d4120a39e6d8681e468f375d
Deleted: sha256:4e894c975b2d807053675d7698806736ce94604c650aac5f885172ab000c8
Deleted: sha256:ee43330e8b1583bee637fac3b1e66c994bb2c0cab7b2372d2584171d1c93d9
Deleted: sha256:fbc611fa4a4ff4c70f9d963c49e2c416f18047c9f94c2dc9328d3b833f1118d
Untagged: 192.168.1.131:5000/php:latest
Deleted: 192.168.1.131:5000/php@sha256:9de503fa1e67f817af9b0c63b1fe54f5744c701393243ed9442fb33fa899736
Deleted: sha256:507301ef013461eeef4038ced70361c70a57aaff350980c302be9586a6a98b
Deleted: sha256:7b9f98dafa20b9841c217df16a22edd3ce27970d0d67a0ddee4f96345012885
Deleted: sha256:8e194c943c24f6700e417ef1e927c24e478a1f74de2378586ee97699beb
Deleted: sha256:c447e3c1a30b61d089e148129aa804f62c143ba51cc5b13ef93ba6a334c4789
Deleted: sha256:220532426789170ac4b4d0360ccaef1f84cf9e3b0a49f7d93b9b0e3476de32
Deleted: sha256:18abe98d3ad08a1dccc74addef2c8a6f759d0b0c776e7b546a655bd73a3e
Deleted: sha256:6d27494c6d6c7c1f198d3b1b0ceee480ced46448254cd0b97ad2020c4
Deleted: sha256:6614480e107bde46ad2944f959909f4529049cbca81a319e194e58a357833
Deleted: sha256:9dcb23d67d2cb8525af31a3f30f938cc07d4460343c3bb357d495b4543c24
Deleted: sha256:482a2cdf9d0c874af7f3ad48d73d0539f3601b3f1e9a32ce0f707f731d3ed5
Deleted: sha256:3d4254bc7d0766f9d31ae35d4f4727ab693e239789261008f4e08604
Deleted: sha256:ee33463738ce64e77d53c0072c3e0e94e3bd120b06a36c38be4f6
Deleted: sha256:0b24856256ee8db887607911ef6eb7762f56db4ef159a1f5eeabef304045
Deleted: sha256:98b5f35e8d3ce6ed1881b5f5d1e02024e1450822879e4c13bb48e9396d0ad
Total reclaimed space: 0B
dtmek@docker2:~/cryptosep$
```

Så er at væk, og vi er klar til at pulle. Som man kan se er "Total reclaimed space: 0B" det betyder at vi faktisk har ryddet alt der var defineret i vores docker-compose.yml fil, Der kan dog ske at der er enkelte cashede objecter der bliver slettet. Uden vi behøver at rydde alt igen derefter.

Vi prøver at pulle med vores `docker-compose.yml`, det kan man selvfølgelig også:

```
docker compose pull
```

Forklaring:

Der skal ikke stå mere i kommandoen. Hvor, og hvad, den skal hente er jo defineret i vores `docker-compose.yml` fil. Inclusive navnet på den registry den skal hente det fra.

Eksempel output:

```
dtmek@docker2:~/cryptosep$ docker compose pull
[+] Pulling 22/22
  ✔ php Pulled
  ✔ a480a496ba95 Pull complete
  ✔ a47a1de29151 Pull complete
  ✔ a0821bbadde4 Pull complete
  ✔ 6be174f186fb Pull complete
  ✔ bd4787ab9f9a Pull complete
  ✔ 741ae76dd1e2 Pull complete
  ✔ 7e6f32953ef1 Pull complete
  ✔ 15b2e1adddd6 Pull complete
  ✔ 6c8a58b73ea3 Pull complete
  ✔ 4f4fb700ef54 Pull complete
  ✔ 79e003456a94 Pull complete
  ✔ 6ba217340f80 Pull complete
  ✔ 2c3e38f70930 Pull complete
  ✔ nginx Pulled
  ✔ f3ace1b8ce45 Pull complete
  ✔ 11d6fdd0e8a7 Pull complete
  ✔ f1091da6fd5c Pull complete
  ✔ 40eea07b53d8 Pull complete
  ✔ 6476794e50f4 Pull complete
  ✔ 70850b3ecceb2 Pull complete
  ✔ 4165791b2583 Pull complete
dtmek@docker2:~/cryptosep$
```

Det virkede jo fint, vi starter vores system med:

```
docker compose up -d
```

Eksempel output:

```
dtmek@docker2:~/cryptosep$ docker compose up -d
[+] Running 3/3
  ✔ Network cryptosep_default Created
  ✔ Container php Started
  ✔ Container web Started
dtmek@docker2:~/cryptosep$
```

Det virkede jo også fint, den skulle heller ikke "bygge" vores containere, så den startede bare containerne.

Kan i huske den fejlmelding vi fik lidt højere oppe, da vi skulle bygge vores containere første gang, hvor de ikke eksisterede i vores registry?

```
! nginx Warning manifest for 192.168.1.131:5000/web:latest not found: manifest unknown: manifest unknown
! php Warning manifest for 192.168.1.131:5000/php:latest not found: manifest unknown: manifest unknown
```

Nu er de images den prøvede at hente, jo på vores registry. Så hvad vil der mon ske hvis vi kun skriver `docker compose up -d`, uden pull først? Lad os prøve, men først skal vi slette alt igen. Husk at ændre de fremhævede til din registry IP:

```
docker compose down
```

```
docker image rm ip_Registry_server:5000/web ip_Registry_server:5000/php
```

```
docker system prune -af
```

Eksempel output:

```
dmek@docker2:~/cryptosep$ docker compose down
docker image rm 192.168.1.131:5000/web 192.168.1.131:5000/php
docker system prune -af
Untagged: 192.168.1.131:5000/web:latest
Deleted: sha256:385a496e0d62eab916c1e1c7c64822e6e42983917c02c5abfa32707b0c0958
Deleted: sha256:5a41d8dc709a7701893d8fba22cc782ea732b89346d06742a849333aebd1ed
Deleted: sha256:3e8a496e0d62eab916c1e1c7c64822e6e42983917c02c5abfa32707b0c0958
Deleted: sha256:9368c2188f80c9fb7fc3eaf770afb7abb3bdf4120a8defd8a81a6df3754
Deleted: sha256:4e834c575b72d507053675d760880673ee946e40c50aac5f8eb517ab008c8
Deleted: sha256:6e43330e0b1512be0e37fecb1e6e998bb20eab7b3722284171d1c948
Deleted: sha256:7bc611fa4e4ff4cf0bd9693c9e2c41eff8047c9f94c2dc932a3b033f118d
Untagged: 192.168.1.131:5000/php:latest
Deleted: 192.168.1.131:5000/php:sha256:94e503fa1c07f217aff80c3b1f64e5744c701393243ed942Eb33cfae99738
Deleted: sha256:307301ef01361eeef093ced70361c70a7aaff35980ec302b558e6a698
Deleted: sha256:7b6f9dafa20989a1c217df1ea22edd3ce27970dcd67a0dee4f96345012885
Deleted: sha256:0a84e943045f870e837e2e4f70a24e74a170dbd37836eeea76c9c0eb
Deleted: sha256:c476e1a30b61d089e140125ae04f62c439e84c0c0b18e2038ea6334c4789
Deleted: sha256:22053242679170ac4bdf0360ccaf1f84cf9a3b0a97d93b9b0e3f7606de32
Deleted: sha256:19ab65038ad0b8a1d8cc74adef7f0c0cc776e7b54e653b7a3a5e
Deleted: sha256:6dd3494c84c0c41f18d8c301b0eeee40eeee4448b4c4c9b97d85020c4
Deleted: sha256:a616480e107b6e16ad29af0999d9f3290f54bcb81a319194e9a357833
Deleted: sha256:9dcb23d67dcb8522a231a3f30f838007d44603430b357d49a5b4593c54
Deleted: sha256:49282c0f8a0c4e7728d4e8738093963b3418a32c0e0797713d3e03
Deleted: sha256:3b42d84bc7d076f9d8d31a35d9f04727ab693ea23b798210b8f4e09e404
Deleted: sha256:3e83346d739c069e77d9c90c72c3e3e40ae9d4e8d120b00fa83f0b0ceef
Deleted: sha256:3b849210e9e80887407911efefeb0702166bae119a1f6eeae39a048
Deleted: sha256:98b5f35ea93eca6ed181b5fed1e0202e145092279e4c13bb4b093860ad
Total reclaimed space: 0B
dmek@docker2:~/cryptosep$
```

Så er det slettet, så lad os prøve kun at skrive:

```
docker compose up -d
```

Eksempel output:

```
dmek@docker2:~/cryptosep$ docker compose up -d
[+] Building 0.0s [0/0]
php Pulled
  a480a956ba95 Pull complete
  a4781de93151 Pull complete
  49281bbdd4e4 Pull complete
  6be174f186fb Pull complete
  bd4787ab9fa Pull complete
  71aa76dd1c2 Pull complete
  7e4f92953ef1 Pull complete
  15b2e1add4d6 Pull complete
  668a5b73ea3 Pull complete
  441e700e54 Pull complete
  79e003456a94 Pull complete
  6ba217340f80 Pull complete
  2c3e38f70930 Pull complete
nginx Pulled
  f3ace1b0ce45 Pull complete
  11deff0e9a7 Pull complete
  f1091d0cfc0c Pull complete
  40ea0705309 Pull complete
  6476794e50f4 Pull complete
  70850b3e6b2 Pull complete
  4165791b2503 Pull complete
[+] Building 0.0s [0/0]
Network cryptosep_default Created
Container php Started
Container web Started
dmek@docker2:~/cryptosep$
```

Det virkede jo!! Så vi behøver ikke at bruger **docker compose pull** mere eller hvad? Der er en stor forskel på de to måder, læg især mærke til de nederste linjer da vi brugte **docker compose up**. Den starter containerne MED DET SAMME. Det gør **docker compose Pull** ikke! Så det er forskellen. Hvis man f.eks. udvikler og skal ændre noget i images, er det måske ikke en god ide at køre containerne med det samme.

Så er vi færdig med opgave 2, men ikke helt, vi skal lige rydde op efter os, så det hele er klar til opgave 3. Vi kører alle kommandoer på en gang. Husk at ændre det fremhævede til din registry IP:

```
docker compose down
docker image rm ip_Registry_server:5000/web ip_Registry_server:5000/php
docker system prune -af
```

Slut på Kapitel 4: Pushing og Pulling.

Så er Opgave 2 færdig, og du kan gå videre til opgave 3. Husk ikke at slette noget, da data fra opgaven skal bruges i opgave 4.