Beskrivelse:

Vi skal lave en Docker container der skal kunne vise en webside der indeholder PHP kode. Dette betyder at vi både skal have PHP og WEBserver til rådighed i den container. Normalt vil man have de to services i separate containere. Dette laver vi i opgave 2. Vi starter med at klargøre linux til at kunne køre docker containere, og lave docker images.

Forudsætninger:

Jeg gentager lige: Det er en STOR fordel at have indsigt i, og prøvet Ubuntu eller tilsvarende Linux.
Ved hver indtastning vil der være et eksempel output. Diverse tal. F.eks. Container ID vil være forskellige, men output burde være meget lig det viste.

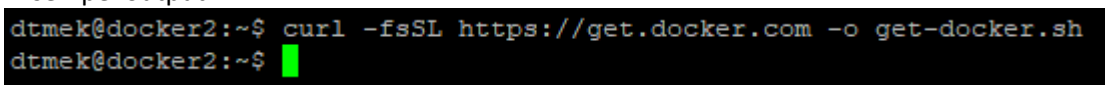
Kapitel 1: Installér Docker.

Så skal der installeres Docker. Først henter vi et script til at instalere docker. Det vil ikke beskrives i dybden, da dette ikke er et linux kursus:

```
curl -fsSL https://get.docker.com -o get-docker.sh
```

Forklaring:
Denne linje henter et script til at installere docker. Scriptet bliver hentet fra nævnte webadresse.

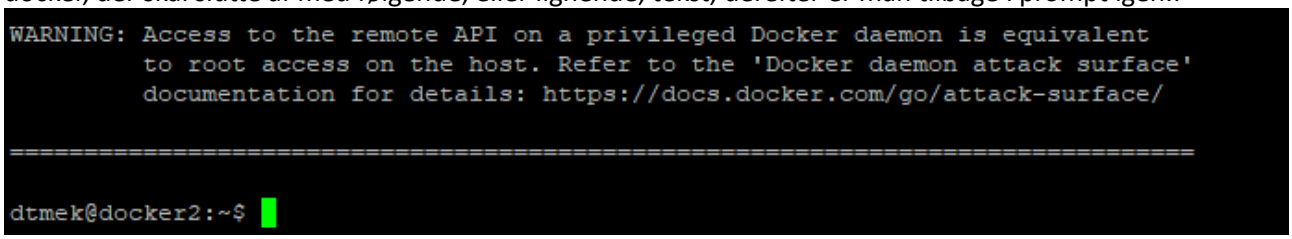
Eksempel output:




```
sudo sh get-docker.sh
```

Forklaring:
Dette starter en shell (sh) og kører scriptet heri. **Vær opmærksom på at der skal indtastes dit password**, for at kunne køre denne linje. (sudo betyder at det kører som super user)

Eksempel output:
Billede er for stort til at gengive her, men der kommer meget tekst, og til sidste kommer der versionen af docker, der skal slutte af med følgende, eller lignende, tekst, derefter er man tilbage i prompt igen.:



Efter installation kan der verificeres at docker er installeret ved at skrive:

```
docker -v
```

Forklaring:
Dette skulle gerne udskrive versionen af docker.

Eksempel output (Version kan være forskellig):

```
dtmek@docker2:~$ docker -v
Docker version 27.3.1, build ce12230
dtmek@docker2:~$
```

Vi skal lige have sat lidt op for at kunne køre docker uden at skulle skrive sudo foran hele tiden. Det gør det nemmere. Normalt vil følgende adgansgruppe være oprettet, når vi har kørt scriptet, og der vil komme en fejl ved oprettelse, men vi gør det lige for en sikkerheds skyld. Begge kommandoer kan køres samtidig, eller hver for sig. **Brugernavn** vil normalt være den bruger man bruger i linux, HUSK at ændre det! (Den du er logget ind i linux som lige nu.)

```
sudo groupadd docker
sudo usermod -aG docker brugernavn
```

Forklaring:
første linje tilføjer en sikkerhedsgruppe der hedder docker (hvis der kommer en fejl, så er den oprettet).
Anden linje tilføjer brugernavn til gruppen.

Eksempel output (Hver kommando for sig):

```
dtmek@docker2:~$ sudo groupadd docker
groupadd: group 'docker' already exists
dtmek@docker2:~$ sudo usermod -aG docker dtmek
dtmek@docker2:~$
```

GENSTART PC!!!!!!!!! (Også selvom det er en virtuel pc!) Gøres ved at skrive **Sudo reboot**

Dette er et godt tidspunkt at lave et snapshot, hvis du er på en virtuel maskine. Det gør det nemmere i allersidste opgave!

Husk også at installere docker på den anden maskine. Det foregår på samme måde (Husk genstart til sidst!). Når du har gjort det er to maskiner klar. Den ene skal, som nævnt før, bruges til at have et docker registry (Lager, eller repository) på. Den anden maskine, er den du bruger til at udføre eksemplerne på. Snapshot kan bruges hvis der sker noget og du skal begynde forfra.

Noter IP adresser på maskinerne, da alle eksempler kræver SSH forbindelse til maskinerne.

Slut for kapittel 1 Installere Docker.

Kapittel 2: Oprettelse af image.

Det efterfølgende foregår på din arbejdsmaskine, den der ikke kaldes registry. Jeg vil gøre tydeligt opmærksom på det, når der skal laves noget på registry maskinen.

Så er systemet klar til at der kan oprettes containere. Vi skal have et image inden vi kan lave en container. Et image er en template til at starte en container. Det image, og container, vi skal lave nu, kalder vi crypto.

Først skal vi finde ud af hvad der er nemmest. Installere php på en eksisterende nginx installation, eller installere en nginx webserver på en eksisterende php installation.

Nginx er nemmest at installere i forhold til en php opsætning. Så i denne opgave installerer vi nginx på et php grund image. (Et grund image, det er det image vi starter med. Det image bliver også kaldt for et parrent image).

Først skal vi have lavet et underbibliotek hvor vi placerer det vi skal bruge for at kunne oprettet imaget, og hoppe ned i det underbibliotek. Vi kører begge kommandoer samtidig.

```
mkdir crypto
cd crypto
```

Forklaring:
Linje et laver underbiblioteket crypto
Linje 2 hopper ned i biblioteket crypto

Eksempel output (Hver kommando for sig):

```
dtmek@docker2:~$ mkdir crypto
dtmek@docker2:~$ cd crypto
dtmek@docker2:~/crypto$
```

Vi skal nu oprette filen der skal bruges til konfiguration af en webside i Nginx. Da dette ikke er et Nginx kursus, vil jeg ikke komme nærmere ind på indholdet i denne konfigutaionsfil.

Opret og rediger filen med redigeringsværktøjet nano:

```
nano nginx-site.conf
```

Indholdet af filen er på næste side.

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

```
################### nginx-site.conf start ##########################
server {
    root    /var/www/mysite;

    include /etc/nginx/default.d/*.conf;

    index app.php index.php index.html index.htm;

    client_max_body_size 30m;

    location / {
        try_files $uri $uri/ /app.php$is_args$args;
    }

    location ~ [^/]\.php(/|$) {
        fastcgi_split_path_info ^(.+?\.php)(/.*)$;
        # Mitigate https://httpoxy.org/ vulnerabilities
        fastcgi_param HTTP_PROXY "";
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_index app.php;
        include fastcgi.conf;
    }
}
################### nginx-site.conf slut ###########################
```

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

Ingen eksempel output.

Nu skal vi lave scriptet, der skal bruges når vores container starter, ellers kan vi ikke få startet alt det andet vi har ”proppet” ind i det der standard image, vi bygger vores image på. Og meget passende kalder vi vores script for entrypoint.sh.

`nano entrypoint.sh`

Filen skal indeholde følgende (Kopier KUN det fremhævede!!):

```
#################### entrypoint.sh start ############################

#!/usr/bin/env bash
service nginx start
php-fpm

#################### entrypoint.sh slut #############################
```

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

Ingen eksempel output

Så skal vi have lavet "bygge vejledningen" til vores image. Jeg vil først gennemgå de forskellige ting filen indeholder. Til sidst i dette punkt kommer filen i sin helhed, og kan kopieres ind i editoren. Du behøver ikke kopiere hver enkel kommando ind. Filen oprettes efter gennemgang af indholdet.

```
FROM php:8.2-fpm
```

Forklaring:
FROM skal altid skrives med stort. Dette bestemmer hvad vores parrent image er. Det der det image vi starter med, og som en anden har lavet for os, og som vi kan hente gratis. Vi kan se at det er imaget php, og versionen på imaget er 8.2-fpm (formatet for image navnet er: image:version). Imaget vil normalt, hvis vi ikke specificere det nærmere, blive hentet på en side der hedder docker.io. her kan der hentes en del standard images. Hvis image ikke findes, vil der komme en fejl.

```
RUN apt-get update -y \
    && apt-get install -y nginx
```

Forklaring:
RUN skal altid være med stort, og kører kommandoer i vores image under generation af image. De aktuelle Kommandoer der kører her er standard linux kommandoer. Den første kører en opdatering af software repositories på linux (-y betyder Yes, og så spøges der ikke "er du sikker?"). Hvis denne kommando slutter uden fejl (**&&**) forsætter udførslen af den næste. Den næste kommando installerer nginx (-y er igen for yes, så der ikke spørges "er du sikker?").

```
ENV PHP_CPPFLAGS="$PHP_CPPFLAGS -std=c++11"
```

Forklaring:
ENV skal altid være med stort, og sætter enviroment variable . Denne variabel skal bruges af PHP.

```
RUN docker-php-ext-install pdo_mysql \
    && docker-php-ext-install opcache \
    && apt-get install libicu-dev -y \
    && docker-php-ext-configure intl \
    && docker-php-ext-install intl \
    && apt-get remove libicu-dev icu-devtools -y
```

Forklaring:
RUN skal altid skrives med stort. Her bliver der kørt flere kommandoer i imaget. Der bliver installeret forskelligt der skal bruges sammen med PHP og webserver.

```
RUN { \
        echo 'opcache.memory_consumption=128'; \
        echo 'opcache.interned_strings_buffer=8'; \
        echo 'opcache.max_accelerated_files=4000'; \
        echo 'opcache.revalidate_freq=2'; \
        echo 'opcache.fast_shutdown=1'; \
        echo 'opcache.enable_cli=1'; \
    } > /usr/local/etc/php/conf.d/php-opocache-cfg.ini
```

Forklaring:
Her bliver der lavet en fil ved at bruge linux komandoen echo (det er en kommando der sender teksten efter kommandoen til standard out enhed) disse kommandoer bliver kørt i imaget. Her bliver det udskrevne sendt til filen "**/usr/local/etc/php/conf.d/php-opocache-cfg.ini**" Dette skal også bruges i samarbejde mellem nginx, og php.

```
COPY nginx-site.conf /etc/nginx/sites-enabled/default
```

Forklaring:
COPY skal altid skrives med stort. Kommandoen kopierer data ind i imaget efter formatet **COPY fra_host Til_Image**. Her bliver nginx konfigurationen lagt ind i imaget på det sted hvor nginx forventer sin konfigurationfil efter formatet.

```
COPY entrypoint.sh /etc/entrypoint.sh
```

Forklaring:
Her kopieres vores startscript ind i imaget.

```
RUN chmod +x /etc/entrypoint.sh
```

Forklaring:
Der bliver udført en linux kommando i imaget der gør at vi kan køre scriptet entrypoint.sh.

```
WORKDIR /var/www/mysite
```

Forklaring:
WORKDIR skal altid skrives med stort. Her defineres det sted hvor, hvis der ikke specificeres et specifikt bibliotek, vil blive arbejdet i vores image.

```
COPY --chown=www-data:www-data --chmod=777 ./Site/* /var/www/mysite
```

Forklaring:
Her kopierer vi Igen data ind i vores image. Denne gang er det vores hjemmeside vi kopierer ind. Men samtidig laver vi lige et par ting. **-- chown** sætter hvilken bruger der ejer filerne. Her er det www-data. **--chmod** ændrer access beskyttelserne til filerne, her sættes everyone full access. Det er nemmest. Sikkerhed mægssigt. Normalt vil man finde ud af hvilke filer er skal have hvilke adgange. Efter det angives hvorfra filerne skal tages, og hvor de skal placeres i vores image.

```
EXPOSE 80
```

Forklaring:
EXPOSE skal altid være med stort. Dette fortæller at vi skal have port 80 åben ind til Containeren når den er startet, da vi jo ellers ikke kan få fat i vores hjemmeside. Denne port er ikke standard for dette image, da det er et php image (her er kun standard port 9000 åben, da denne port skal bruges til php kommunikation.). Men vi skal jo bruge port 80 for at kunne vide vores webside.

```
ENTRYPOINT ["/etc/entrypoint.sh"]
```

Forklaring:
ENTRYPOINT skal altid være med stort. Dette fortæller at der skal udføres en fil når vi starter imaget som en container. I dette tilfælde vores script entrypoint.sh, som vi jo gjorde executable længere oppe.

Så er gennemgangen af vores Dcokerfile færdig, og vi vil oprette filen (**filen SKAL være med stort D**).

```
nano Dockerfile
```

Indholdet til filen er på næste side.

Filen skal indeholde følgende:

```
################### Dockerfile start ##########################
# Hvilket image er vores start images
FROM php:8.2-fpm

# Kommandoer der skal køres i Image, for instalation af NginX
RUN apt-get update -y \
    && apt-get install -y nginx

# Sætter en Enviroment variable (ligsom Path på Windows)
# PHP_CPPFLAGS are used by the docker-php-ext-* scripts
ENV PHP_CPPFLAGS="$PHP_CPPFLAGS -std=c++11"

# Kør flere kommandoer der skal bruges af php eller NginX
RUN docker-php-ext-install pdo_mysql \
    && docker-php-ext-install opcache \
    && apt-get install libicu-dev -y \
    && docker-php-ext-configure intl \
    && docker-php-ext-install intl \
    && apt-get remove libicu-dev icu-devtools -y

#Lav filen /usr/local/etc/php/conf.d/php-opocache-cfg.ini
RUN { \
        echo 'opcache.memory_consumption=128'; \
        echo 'opcache.interned_strings_buffer=8'; \
        echo 'opcache.max_accelerated_files=4000'; \
        echo 'opcache.revalidate_freq=2'; \
        echo 'opcache.fast_shutdown=1'; \
        echo 'opcache.enable_cli=1'; \
    } > /usr/local/etc/php/conf.d/php-opocache-cfg.ini

# Kopier NginX konfig filen.
COPY nginx-site.conf /etc/nginx/sites-enabled/default

# Kopier Start Filen til Image
COPY entrypoint.sh /etc/entrypoint.sh

#Gør entrypoint.sh executable
RUN chmod +x /etc/entrypoint.sh

# Det er her alt hvad vi laver med COPY osv. bliver placeret
WORKDIR /var/www/mysite

# kopier vores rå site ind i image, og sætter sikkerhed
COPY --chown=www-data:www-data --chmod=777 ./Site/* /var/www/mysite

# Åben for port 80
EXPOSE 80

# dette udføres hver gang Container startes
ENTRYPOINT ["/etc/entrypoint.sh"]

################### Dockerfile slut ###########################
```

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

Ingen eksempel output.

Kopier biblioteket ”Site” over til din arbejdscomputer. ”Site” kan findes i eksempelfilen til opgave1 (Opgave1Materiale.zip) **”Site” skal ligge under crypto biblioteket**, samme sted hvor vi oprettede ovenstående filer. Vær opmærksom på at ”Site” SKAL hedde ”Site” (Med stort S!!). Biblioteket kan kopieres over med f.eks. WinSCP fra en windows maskine. Se indledning for kort vejledning i Winscp. Opgave1Materiale.zip kan findes på det website refereret til i indledningen.

Der skal ikke laves nogen ændring af sikkerhed på Site. Dette gøres når vi bruger kommandoen COPY i Dockerfile, som forklaret før.

Så er vi klar til at ”bygge” vores image.

For at bygge imaget skal vi stå i crypto underbiblioteket. Følgende kommando køres:

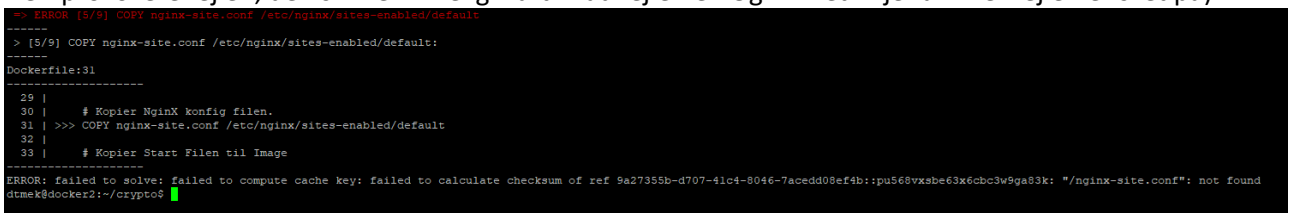
```
docker build -t crypto .
```

Forklaring:
`docker build`: Dette er kommandoen.
`-t crypto`: Dette giver imaget et navn, eller et ”tag” (ligesom når man ved graffiti tagger sit værk)
`.` : Hvor finder kommandoen sine filer. Her er det Present directory = det aktive bibliotek du befinder dig i. Det er her de filer der skal bruges kan findes. Det er også derfor at Site skal være placeret her. Kan f.eks. også være **`/path/til/filer`**

Computeren vil arbejde lidt (Det kan tage flere minutter, alt efter hvor mange ressourcer man har på work PC’en), og imaget er genereret.

Eksempel Output:
Der vil komme en hel masse tekst. Slutningen skulle gerne se sådan her, eller lign., ud (Ved mig er skriften svær at læse da den er mørkeblå på sort):

Ved fejl vil der komme en besked frem som den herunder (Fejlen her: Jeg har omdøbt den ene fil for at fremprovokere fejlen, den skriver rimelig klart hvad fejlen er. Og hvilket linjenummer fejlen er sket på):



For at se om der ligger et image, nu vi har bygget det, kan man køre følgende kommando:

`docker image ls -a`

Forklaring:
`docker image`: Kommandoen for at behandle images.
`ls`: Lister de images der er tilstede på computeren
`-a`: Er ikke altid nødvendig, det betyder bare Vis alle, også inaktive images.

Eksempel output:

```
dtmek@docker2:~/crypto$ docker image ls -a
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
crypto       latest    0ab11fa57db4   4 minutes ago   523MB
dtmek@docker2:~/crypto$
```

Slut for kapitel 2 oprettelse af image.

Kapitel 3: Bearbejd image, og kør container.

Vi vil nu køre det image vi har lavet, og idet man kører et image bliver det til en Container. Man kan lave lige så mange containere man vil af ét image, eller nok mere præcis, lige så mange containere man har ressourcer til, i det system man har adgang til.

For at køre vores image udføres følgende kommando:

```
docker run --name crypto -p 80:80 -td crypto
```

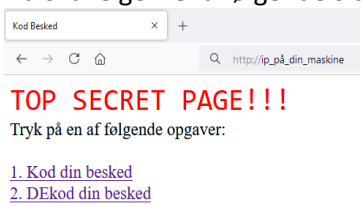
Forklaring:
`docker run`: Dette er kommandoen der skal til for at køre et image.
`--name crypto`: Dette giver containeren et navn. Her er navnet: crypto.
`-p 80:80`: Dette sender port 80 på hosten ind til port 80 på den kørende container, efter følgende syntax: Port på HOST:Port på CONTAINER.
`-td`: -t betyder output fra Container bliver sendt til en pseudo TTY (Terminal). -d betyder detatched fra aktiv SSH sesion. Altså når container er startet kan man skrive videre i sin åbne SSH sesion igen.
`crypto`: er navnet på det image der skal bruges som template.

Eksempel output (Det lange nummer er et unikt nummer for kun denne container):

```
dtmek@docker2:~/crypto$ docker run --name crypto -p 80:80 -td crypto
a7e8edfa47f41488248743fd3e378fb273e072ede2b17bafb37764fe378a7aca
dtmek@docker2:~/crypto$ ^C
dtmek@docker2:~/crypto$
```

Prøv at åbne en webbrowser på din pc, og gå til ip-adressen på din worker maskine, hvor du har startet din container. I din browser skriver du **`http://ip_på_worker_makine`**

Du skulle gerne få følgende side frem:



Så nu har du lavet en hjemmeside der bliver vist via en container. Vi skal jo ikke kun kunne køre en container, vi skal jo også kunne stoppe en container. Der findes to forskellige måder at stoppe en container på.

1. Kommandoen "**`docker stop Container_navn`**" Denne kommando stopper en container, ved at lukke den ned på den "rigtige" måde. Hvis det f.eks. er en SQL-base container, søger kommandoen for at flushe data, lukke database, og afbryde forbindelse på den rigtige måde, inden containeren lukkes ned-.
2. Komandoen "**`docker Kill Container_navn`**" Denne kommando Stopper container her og nu! Uden at "ryde op" før den lukkes ned.

Prøv nu at stoppe din container, du må selvom hvilken af kommandoerne du bruger (Hvis der udskrives container navnet, så er container stoppet/killet):

`docker stop crypto` eller **`docker kill crypto`**

Ingen eksempel output

Prøv evt. At gå til websiden for din container igen (**`http://ip_på_worker_makine`**). Den kommer ikke frem, da Containeren jo ikke kører.

Vi starter lige containeren igen:

`docker run --name crypto -p 80:80 -td crypto`

eksempel output:

```
dtmek@docker2:~/crypto$ docker run --name crypto -p 80:80 -td crypto
docker: Error response from daemon: Conflict. The container name "/crypto" is al
ready in use by container "e48a1e055398790bf9a1b56046e18ef0bd59e3e71a9c3819df103
e2c3503664a". You have to remove (or rename) that container to be able to reuse
that name.
See 'docker run --help'.
dtmek@docker2:~/crypto$
```

Åhhh Nej!! det virker ikke! Hvordan starter vi nu en container??? Nåååhhh ja... der findes selvfølgelig også en kommando for det:

`docker start crypto`

Forklaring:
`docker start`: Dette er kommandoen der starter den specificerede container.
`crypto`: dette er navnet på containeren.

Eksempel output:

```
dtmek@docker2:~/crypto$ docker start crypto
crypto
dtmek@docker2:~/crypto$
```

Prøv evt. At gå til websiden i din container igen (**`http://ip_på_worker_makine`**). Du får nu websiden frem igen.

Nu kan vi starte og stoppe en container. Men der findes et par andre vigtige kommandoer der kan give os nogle oplysninger omkring containere eller images. Dem gennemgår vi lige inde vi fortsætter.

Her er kommandoerne:

1. Vise aktivt kørende containere:
   `docker ps`

2. Vise alle containere
   `docker container ls`

3. Vise alle containere, også de inaktive (-a for ALL)
   `docker container ls -a`

4. Vise alle images
   `docker image ls`
5. Vise alle images, også de inaktive (-a for ALL).
   `docker image ls -a`

6. fjerne en container (rm for remove).
   `docker container rm container_navn`

7. fjerne et image
   `docker image rm image_navn`

8. Hvis der opstår et problem ved fjernelse (-f for FORCE)
   `docker image rm -f image_navn`

Man kan ALTID skrive **`--help`** efter en kommando for at få en oversigt over de kommandoer der kan bruges (f.eks. **`docker image --help`**, man kan også skrive **`docker --help`** for hjælp til docker hoved kommandoen)

Vi må hellere komme videre. Ville det ikke være interessant at få at vide hvor mange ressourcer en container bruger? Bare rolig... der findes også en kommando for det. Prøv at skrive:

`docker stats`

Eksempel output:

```
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O       BLOCK I/O   PIDS
e48a1e055398   crypto    0.00%     13MiB / 7.757GiB    0.16%     1.09kB / 0B   0B / 0B     9
```

Her kan man se hvor meget processor power, ram, osv. en given container bruger. Udskriften er dynamisk, dvs. den opdaterer hele tiden forbruget. For at afslutte denne visning trykkes på **`CTRL + C`**

Nu har vi jo startet vores container. Men vi skal jo også, sommetider, genstarte vores server f.eks., efter en opdatering. Det første vi gør er at se om containeren stadig kører, inden vi genstarter. (gå til **`http://ip_på_worker_makine`** og se om siden kommer frem). Hvis den ikke gør, så start din container med **`docker start crypto`**.

Genstart din docker server (**`sudo reboot`** Og indtaste dit password). Derefter log på server igen med SSH. Vi skal nok lave flere ting om lidt.

Efter genstart prøv at gå til hjemmesiden på din container igen: **`http://ip_på_din_maskine`** Det var ikke så godt... Der er ingen kontakt? Vi kigger lige efter om din container kører:

`docker ps`

eksempel output:

```
dtmek@docker2:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
dtmek@docker2:~$
```

Det gør den ikke! Men det er jo ikke godt? Hvad nu hvis vi har en hele masse forskellige containere kørende, og så skal starte alle containere hver for sig igen, efter vi har genstartet hosten? Men fortvivl ikke... der findes selvfølgelig en løsning på det.

Først skal vi have den container der ikke kører fjernet (rm for ReMove):

`docker container rm crypto`

Eksempel output:

```
dtmek@docker2:~$ docker container rm crypto
crypto
dtmek@docker2:~$
```

Nu når det kun er imaget der er tilbage igen, og det er jo dét vi skal bruge, så vi behøver ikke at bygge det igen med docker build. Vi skal jo bare starte containeren så den fortsætter med at køre efter en genstart af vores server. Dette gøres ved at køre følgende kommando, der er meget lig den forrige vi brugte til at starte en container første gang, men vi har tilføjet et enkelt ekstra flag **`--restart unless-stopped`**:

`docker run --restart unless-stopped --name crypto -p 80:80 -td crypto`

forklaring:
`docker run`: Dette er kommandoen der skal til for at køre et image.
`--restart unless-stopped`: Dette sørger for at en container ”overlever” en genstart af en server. Altså containeren starter automatisk.
`--name crypto`: Dette giver containeren et navn. Dette navn er: crypto.
`-p 80:80`: Dette sender port 80 på hosten ind til port 80 på den kørende container, efter følgende syntax: Port på HOST:Port på CONTAINER.
`-td`: -t betyder output fra Container bliver sendt til en pseudo TTY (Terminal) -d betyder detatched fra aktiv SSH sesion. Altså når container er startet kan man skrive i sin åbne SSH sesion igen.
`Crypto`: er navnet på det image der skal bruges som template.

Eksempel output:

```
dtmek@docker2:~$ docker run --restart unless-stopped --name crypto -p 80:80 -td crypto
86946a317921e436cdc7647a419030e11855bc50a641cb0fcc39272c6e98e348
dtmek@docker2:~$
```

Der findes forskellige indstillinger af indstillingen **`--restart`** her er de vigtigste:

`no`: Genstart ikke automatisk. Dette er standard indstillingen.
`Allways`: Hvis container stopper, genstarter den. Også hvis man kører en docker stop/kill.
`Unless-stoppet`: Hvis en container stopper, genstarter den, undtagen hvis du har kørt en docker kill eller docker stop

Prøv at genstarte din server (**`sudo reboot`** Og indtast password), og gå til websiden igen (http://ip_på_din_maskine). Websiden burde være tilstede og virke, selv efter genstart. Husk også at logge på din server igen med SSH.

Slut på kapitel 3 Bearbejd image, og kør container.

Kapitel 4: Pushe image til Registry.

Nu har vi jo et image, der kan køre som en container. Ville det ikke være en fordel, hvis vi kunne gemme vores images et eller andet sted, og kunne bruge dem på f.eks. andre servere? Det er jo det er styrken ved docker. At du kan køre en container hvor som helst med Docker installeret. En docker container der er lavet, vil jo opføre sig ens uanset om den container kører på ubuntu, en anden linux, eller for den sags skyld docker for windows. Det jeg vil gennemgå i dette kapitel, er hvordan man kan lave sit eget "lagerhotel" eller repository. Et repository kaldes for et "Registry" i docker. Så i dette afsnit vil vi oprette et Registry, og vi vil gemme et image (Pushe).

Vi skal nu også til at bruge den anden server. Den vi kaldte for registry. HUSK: Registry computeren skal have docker installeret også.

De følgende kommandoer skal udføres på den maskine der er kaldt registry.
##

Som man kan se herunder er et registry bare en docker container der bliver startet:

```
docker run --restart unless-stopped -d -p 5000:5000 --name local-registry registry:2
```

Forklaring:
`docker run`: Dette er kommandoen der skal til for at køre et image.
`--restart unless-stopped`: Dette sørger for at en container "overlever" en genstart af en server. Altså containeren starter automatisk.
`-d`: betyder detatched fra aktiv SSH sesion.
`-p 5000:5000`: Dette sender port 5000 på hosten ind til port 5000 på den kørende container, efter følgende syntax: Port på HOST:Port på CONTAINER
`--name local-registry`: Dette giver containeren et navn. Her er navnet: local-registry
`registry:2` : Henter imaget registry, og det er version 2 vi henter (formatet er image:version)

Eksempel Output:

```
dtmek@registry:~$ docker run --restart unless-stopped -d -p 5000:5000 --name local-registry registry:2
Unable to find image 'registry:2' locally
2: Pulling from library/registry
1cc3d825d8b2: Pull complete
85ab0942le5a: Pull complete
40960af72c1c: Pull complete
e7bb1dbb377e: Pull complete
a538cc9blae3: Pull complete
Digest: sha256:ac0192b549007e22998eb74e8d8488dcfe70f1489520c3b144a6047ac5efbe90
Status: Downloaded newer image for registry:2
7d3ce18494ca200f1d231284a096f57f176a2e13468a426a6638ec9db6c6bc00
dtmek@registry:~$
```

Når dette er udført, prøv at køre den kommando, så du kan se at docker registry containeren er kørende.

```
docker ps
```

Eksempel output:

```
dtmek@registry:~$ docker ps
CONTAINER ID   IMAGE        COMMAND                  CREATED          STATUS          PORTS                                       NAMES
7d3ce18494ca   registry:2   "/entrypoint.sh /etc…"   16 minutes ago   Up 16 minutes   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   local-registry
dtmek@registry:~$
```

Husk at notere IP adressen på denne server, da den skal bruges senere!!

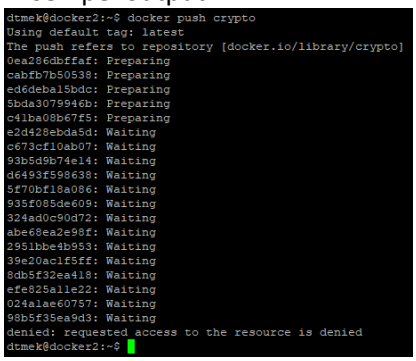
Du kan nu afslutte din SSH til din Registry server. Det gør du ved at skrive **`exit`**, og så lukke SSH vinduet.

##
Vi skifter tilbage til den maskine du kører eksemplerne på!! Din worker maskine!

Nu har vi jo oprettet et registry, så må vi hellere få pushet vores image ud til denne server:

`docker push crypto`

Eksempel output:



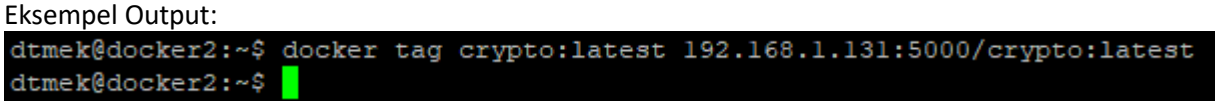
Åhh.. Det virkede ikke. Hvorfor nu ikke det? Jo... Standard vil docker pushe dine images til websiden, og registrysiden, docker.io. Men her skal du først oprettes som bruger for at kunne pushe, og du får kun lov til at pushe et projekt gratis. Ellers skal du betale. Det gider vi ikke betale for, vi har jo også lige lavet vores eget registry, hvor vi kan placere vores images. Men det vi først skal, er at fortælle docker at vi vil pushe til vores eget Registry.

Det gør vi ved at tagge vores image (ligesom en grafiti kunstner tagger sine værker). Det gør vi lige:

`docker tag crypto:latest ip_Registry_server:5000/crypto:latest`

Forklaring:
`docker tag`: Kommandoen der tagger dit image.
`Crypto:latest` : Dette er navnet og versionen efter dette format: Navnet:Version (kan også være f.eks. crypto:2.0).
`ip_Registry_server:5000/:` Dette angiver registry IP, og hvilken port den skal bruge til komunikation.
`crypto:latest`: Dette er navnet, og versionen imaget gemmes under på Registry server, efter dette format: Navn:Version.

Eksempel Output:



Vi må lige kigge lidt på hvad der ligger under images nu:

`docker image ls -a`

Eksempel Output:

```
dtmek@docker2:~$ docker image ls -a
REPOSITORY                    TAG       IMAGE ID       CREATED        SIZE
192.168.1.131:5000/crypto     latest    0ab11fa57db4   2 hours ago    523MB
crypto                        latest    0ab11fa57db4   2 hours ago    523MB
dtmek@docker2:~$
```

Her vil man nu se 2 stk. ”images” men læg mærke til at det felt, der hedder ”IMAGE ID” er der samme! Det er samme image, men med 2 forskellige navne.

Nu kan vi pushe det image der hedder **`ip_Registry_server:5000/crypto:latest`** til vores registry server:

`docker push ip_Registry_server:5000/crypto:latest`

Eksempel Output:

```
dtmek@docker2:~$ docker push 192.168.1.131:5000/crypto:latest
The push refers to repository [192.168.1.131:5000/crypto]
Get "https://192.168.1.131:5000/v2/": http: server gave HTTP response to HTTPS client
dtmek@docker2:~$
```

ÅÅhh nej.. Der var en fejl igen!
Hvad mon den mener med det? Jo.. den mener bare at det er en registry server der ikke har nogen https reply, men bare rolig! Inden der er nogen der går i panik! Vi kan selvfølgelig også løse dette. Vi bliver nødt til at fortælle vores worker server at den der Registry server er OK, og alt i orden. Der gør vi ved at tilføje den til en speciel fil vi lige skal lave:

Først starter vi en editor (Der skal indtastes password, da der skrives ”sudo” foran, indsæt tekst ved at højreklikke i editor):

`sudo nano /etc/docker/daemon.json`

Filen skal indeholde følgende (Kopier KUN det fremhævede!! Og HUSK at indtaste IP på DIN registry server!!):

`################### deamon.json start ############################`

```
{
"insecure-registries": ["http://ip_Registry_server:5000"]
}
```

`################### daemon.json slut ############################`

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

Ingen eksempel output
Så har vi lavet en fil, men for at det virker bliver vi nødt til at genstarte docker servicen, du vil blive promptet for at indtaste password på din bruger i linux (Det kan tage lidt tid at genstarte denne service):

`sudo systemctl restart docker.service`

Eksempel output:

```
dtmek@docker2:~$ sudo systemctl restart docker.service
dtmek@docker2:~$
```

Vi må hellere lige kontrollere at servicen er startet ordentligt op igen:

`systemctl status docker.service`

Eksempel output:

```
dtmek@docker2:~$ systemctl status docker.service
● docker.service - Docker Application Container Engine
     Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: enabled)
     Active: active (running) since Thu 2024-11-07 11:24:55 CET; 1min 57s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 1465 (dockerd)
      Tasks: 28
     Memory: 26.1M (peak: 26.7M)
        CPU: 470ms
     CGroup: /system.slice/docker.service
             ├─1465 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
             ├─1644 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 80 -container-ip 172.17.0.2 -container-port 80
             └─1651 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port 80 -container-ip 172.17.0.2 -container-port 80

Nov 07 11:24:54 docker2 dockerd[1465]: time="2024-11-07T11:24:54.066589495+01:00" level=info msg="[graphdriver] using prior storage driver: overlay2"
Nov 07 11:24:54 docker2 dockerd[1465]: time="2024-11-07T11:24:54.069622222+01:00" level=info msg="Loading containers: start."
Nov 07 11:24:54 docker2 dockerd[1465]: time="2024-11-07T11:24:54.599149814+01:00" level=info msg="Default bridge (docker0) is assigned with an IP address 172.17.0.0/16. Daemon option --bip can be used to set a preferred IP address"
Nov 07 11:24:54 docker2 dockerd[1465]: time="2024-11-07T11:24:54.933275635+01:00" level=info msg="Loading containers: done."
Nov 07 11:24:54 docker2 dockerd[1465]: time="2024-11-07T11:24:54.950898817+01:00" level=warning msg="WARNING: bridge-nf-call-iptables is disabled"
Nov 07 11:24:54 docker2 dockerd[1465]: time="2024-11-07T11:24:54.950929950+01:00" level=warning msg="WARNING: bridge-nf-call-ip6tables is disabled"
Nov 07 11:24:54 docker2 dockerd[1465]: time="2024-11-07T11:24:54.950952122+01:00" level=info msg="Docker daemon" commit=41ca978 containerd-snapshotter=false storage-driver=overlay2 version=27.3.1
Nov 07 11:24:54 docker2 dockerd[1465]: time="2024-11-07T11:24:54.950998086+01:00" level=info msg="Daemon has completed initialization"
Nov 07 11:24:55 docker2 dockerd[1465]: time="2024-11-07T11:24:55.051599015+01:00" level=info msg="API listen on /run/docker.sock"
Nov 07 11:24:55 docker2 systemd[1]: Started docker.service - Docker Application Container Engine.
dtmek@docker2:~$
```

Der skulle gerne et eller andet sted i toppen stå: Active: active (running) hvor det sidste er grønt. (Tryk evt. på **`CTRL + C`** for at komme tilbage til prompt, hvis skærmstørrelse ikke tillader at vise alt output).

Så prøver vi at pushe vores vores image... Igen:

`docker push ip_Registry_server:5000/crypto:latest`

Eksempel output:

```
dtmek@docker2:~$ docker push 192.168.1.131:5000/crypto:latest
The push refers to repository [192.168.1.131:5000/crypto]
0ea286dbffaf: Pushed
cabfb7b50538: Pushed
ed6deba15bdc: Pushed
5bda3079946b: Pushed
c41ba08b67f5: Pushed
e2d428ebda5d: Pushed
c673cf10ab07: Pushed
93b5d9b74e14: Pushed
d6493f598638: Pushed
5f70bf18a086: Pushed
935f085de609: Pushed
324ad0c90d72: Pushed
abe68ea2e98f: Pushed
2951bbe4b953: Pushed
39e20ac1f5ff: Pushed
8db5f32ea418: Pushed
efe825a11e22: Pushed
024a1ae60757: Pushed
98b5f35ea9d3: Pushed
latest: digest: sha256:011b8ef9a5a58f7a66ff6a813034c9095a442d557ae05e9400093abfb245178f size: 4284
dtmek@docker2:~$
```

Man kan nu se at der bliver overført en hel masse ting til Registry serveren.

Nu har vi lagt noget op på vores Registry server. Men hvordan finder vi ud af hvad der ligger, eller om der overhovedet ligger noget på den server? Det gør vi ved nedenstående kommando:

`curl -X GET http://ip_Registry_server:5000/v2/_catalog`

Forklaring:
`Curl`: Dette er en linux kommando, der f.eks. kan hente websider.
`-X Get`: Hvad skal vi lave? I dette tilfælde HENTER vi data (På engelsk Get).
`http://ip_Registry_server:5000/v2/_catalog`: Protokol (http) og hvor henter vi data?

Eksempel output:

```
dtmek@docker2:~$ curl -X GET http://192.168.1.131:5000/v2/_catalog
{"repositories":["crypto"]}
dtmek@docker2:~$
```

Som man kan se ligger der kun ét repository, og det er det der hedder crypto... (der kommer flere senere...)

Men hvor mange versioner ligger der så af det der "crypto" repository? Jo det finder vi med denne kommando. Det er faktisk kun adressen der er ændret lidt, ellers er kommandoen den samme:

`curl -X GET http://ip_Registry_server:5000/v2/crypto/tags/list`

Eksempel Output:

```
dtmek@docker2:~$ curl -X GET http://192.168.1.131:5000/v2/crypto/tags/list
{"name":"crypto","tags":["latest"]}
dtmek@docker2:~$
```

Her kan man se at der kun er et tag under crypto, og det er "Latest".

Hvad nu hvis vi sætter endnu et tag på vores image? Det er jo nemt. Det gør vi li.. Stop! Vi gør det lige lidt anderledes, bare for at lære lidt også :) Vores image har også et unikt ID, dette skal vi lige finde:

`docker image ls`

Eksempel output:

```
dtmek@docker2:~$ docker image ls
REPOSITORY                    TAG       IMAGE ID       CREATED         SIZE
192.168.1.131:5000/crypto     latest    0ab11fa57db4   3 hours ago     523MB
crypto                        latest    0ab11fa57db4   3 hours ago     523MB
dtmek@docker2:~$
```

For at finde image ID kigger vi i kolonnen "IMAGE ID" (lyder logisk.. ikke?) her er image id: 0ab11fa57db4, og det bruger vi lige til at tagge vores image med et nyt tag:

`docker tag IMAGE_ID ip_Registry_server:5000/crypto:2.0`

Eksempel Output:

```
dtmek@docker2:~$ docker tag 0ab11fa57db4 192.168.1.131:5000/crypto:2.0
dtmek@docker2:~$
```

Ligesom sidst vi taggede et image er der ikke noget at se endnu. Men vi kunne jo prøve at se på hvad der ligger af images på vores maskine nu:

`docker image ls`

Eksempel output:

```
dtmek@docker2:~$ docker image ls
REPOSITORY                    TAG       IMAGE ID       CREATED         SIZE
192.168.1.131:5000/crypto     2.0       0ab11fa57db4   3 hours ago     523MB
192.168.1.131:5000/crypto     latest    0ab11fa57db4   3 hours ago     523MB
crypto                        latest    0ab11fa57db4   3 hours ago     523MB
dtmek@docker2:~$
```

Nu ligger der et nyt image der har tag 2.0, dog stadig med samme IMAGE ID. Det må jo betyde at vi har lavet en eller anden ændring i vores image, og har lavet en version 2.0 af vores image. Den ændring må vi hellere lige sørge for at få pushed ud til vores Registry (Læg mærke til allersidst i kommandoen. Her er **`crypto:latest`** skiftet til **`crypto:2.0`**):

`docker push ip_Registry_server:5000/crypto:2.0`

Eksempel Output:

```
dtmek@docker2:~$ docker push 192.168.1.131:5000/crypto:2.0
The push refers to repository [192.168.1.131:5000/crypto]
0ea286dbffaf: Layer already exists
cabfb7b50538: Layer already exists
ed6deba15bdc: Layer already exists
5bda3079946b: Layer already exists
c41ba08b67f5: Layer already exists
e2d428ebda5d: Layer already exists
c673cf10ab07: Layer already exists
93b5d9b74e14: Layer already exists
d6493f598638: Layer already exists
5f70bf18a086: Layer already exists
935f085de609: Layer already exists
324ad0c90d72: Layer already exists
abe68ea2e98f: Layer already exists
2951bbe4b953: Layer already exists
39e20ac1f5ff: Layer already exists
8db5f32ea418: Layer already exists
efe825a11e22: Layer already exists
024a1ae60757: Layer already exists
98b5f35ea9d3: Layer already exists
2.0: digest: sha256:011b8ef9a5a58f7a66ff6a813034c9095a442d557ae05e9400093abfb245178f size: 4284
dtmek@docker2:~$
```

Det tog jo ingen tid, i forhold til første gang? Og der står en hel masse med ”Layer already exists”. Hvad mener den med det? Jo.. for at få forklaringen skal vi lige lidt tilbage i tiden, helt tilbage til hvor vi lavede ”byggevejledningen”. Den der hed Dockerfile. Hver eneste kommando i denne fil laver et lag (layer) mere. Dvs. hvis vi ændrer lidt i en enkel linje, eller tilføjer en linje. Så er det kun ændringen der bliver overført og evt. efterfølgende ændringer, som følge af den ene ændring vi lavede, der vil blive overført. Det sparer plads, og tid. Så det eneste der havde ændret sig denne gang, det var jo versionen. Så det var kun versionsnummeret der blev overført.

Men hvad ligger så egentlig på vores Registry nu? Vi kigger lige efter:

`curl -X GET http://ip_Registry_server:5000/v2/_catalog`

Eksempel output:

```
dtmek@docker2:~$ curl -X GET http://192.168.1.131:5000/v2/_catalog
{"repositories":["crypto"]}
dtmek@docker2:~$
```

Hmmmm… Der er stadig kun en post i vores repositiry… Nåh ja.. vi må hellere lige kigge i vores repository ”crypto” og se om der ligger flere versioner:

`curl -X GET http://ip_Registry_server:5000/v2/crypto/tags/list`

Eksempel output:

```
dtmek@docker2:~$ curl -X GET http://192.168.1.131:5000/v2/crypto/tags/list
{"name":"crypto","tags":["2.0","latest"]}
dtmek@docker2:~$
```

Her kan vi se at der er kommet en version ”2.0” og der er også stadig en version ”latest”.

Nu har vi pushet det hele ud på en server, så nu vil vi slette ALT der ligger på denne vores worker server, så vi kan se om vi ikke kan få vores image tilbage igen..

Først skal vi stoppe vores crypto Container (JA!.. jeg bruger Kill, det er hurtigst, og vi har ikke nogen database..):

`docker container kill crypto`

Eksempel output:

```
dtmek@docker2:~$ docker container kill crypto
crypto
dtmek@docker2:~$
```

Derefter skal vi slette vores containere:

`docker container rm crypto`

Eksempel output:

```
dtmek@docker2:~$ docker container rm crypto
crypto
dtmek@docker2:~$
```

Vi må hellere huske at slette imaget også, ellers er det snyd. Hvis der er flere containere, kan de alle slettes ved at separere de enkelte navne med mellemrum, og det kan man også ved images, men det er måske en ide at finde image navnene, inden vi begynder at slette dem. Det gør vi her:

`docker image ls -a`

Eksempel output:

```
dtmek@docker2:~$ docker image ls
REPOSITORY                   TAG       IMAGE ID       CREATED        SIZE
192.168.1.131:5000/crypto    2.0       0ab11fa57db4   4 hours ago    523MB
192.168.1.131:5000/crypto    latest    0ab11fa57db4   4 hours ago    523MB
crypto                       latest    0ab11fa57db4   4 hours ago    523MB
dtmek@docker2:~$
```

Her er der 3 images, dem sletter vi lige, hvis der er flere, tilføjes de bare linjen (se eksempel output):

`docker image rm image1 image2 image3`

Eksempel output:

```
dtmek@docker2:~$ docker image rm crypto:latest 192.168.1.131:5000/crypto:latest 192.168.1.131:5000/crypto:2.0
Untagged: crypto:latest
Untagged: 192.168.1.131:5000/crypto:latest
Untagged: 192.168.1.131:5000/crypto:2.0
Untagged: 192.168.1.131:5000/crypto@sha256:011b8ef9a5a58f7a66ff6a813034c9095a442d557ae05e9400093abfb245178f
Deleted: sha256:0ab11fa57db4338e05cae4200c9abe284f84315bc6a808b2576dd84f09302d62
dtmek@docker2:~$
```

Vi må hellere lige sørge for at systemet er helt ryddet op, det gør følgende kommando:

`docker system prune`

Eksempel output:

```
dtmek@docker2:~$ docker system prune
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all dangling images
  - unused build cache

Are you sure you want to continue? [y/N] y
Deleted build cache objects:
nvrtyazlntcmd3fetj9mo9n87
xdzilltlm3hpz6v0ksoc0dkch
jllrwhl4p8p4hr3lweokfnl6d
irizu7peatjyfa3di8chab6l9
mu4ohsx7sobd6yglf6wrbaym5
v0tm7g5y7u6y5cj8w43nbqwhc
s5nycf9elqu8zho6mky5ikqn2
z92mu6sh6rgwbl8uppgszwlei
8pszsqlbcv86xel9r0ukk62hv
a79f54yz8jh9n7isq07nyfaao
h2rnj7smnkyuosbjahqbsycc1
2w77zt87pttm3f8n2thfs2ktj
q8w96uf85wu20uytut6c4s6b2
ijxo56f2bljle609s66pub0ta
talre83ov3fzm6ptsy7brt0og
ytvcgw2dewv7hnyw6rwap2f5k
p4fmluxrdp3irbtzv5ch6f4ht
fz5y6bpkzvvt2bsnqnqqguh9t
umjac2g369ze55rr7s4335kzp
sp7jo46mmlssxlxbfkie8my9a
gvmbhlh9w3a5v3rllhh6uvnle
kluliklveo8kuban26jlbebgq

Total reclaimed space: 31.1MB
dtmek@docker2:~$
```

Der var lige et par ekstra ting der skulle slettes, det var faktisk alle de ”lag” vi havde lavet som var tilføjet det parrent image vi brugte, og vores parrent image (php:8.2-fpm) blev også slettet.

Nu er vi klar til at prøve at Pulle fra vores Registry.

Slut på Kapitel 4: Pushe image til Registry.

Kapitel 5: Pulle et image fra Registry.

Nu har vi jo gemt vores image, og vi har slettet alt på vores maskine. Så mon ikke det er god ide og se om det image vi har gemt, også kan hentes tilbage? Vi prøver! Det gør vi med følgende kommando:

`docker pull crypto`

Eksempel output:

```
dtmek@docker2:~$ docker pull crypto
Using default tag: latest
Error response from daemon: pull access denied for crypto, repository does not exist or may require 'docker login': denied: requested access to the resource is denied
dtmek@docker2:~$
```

En Fejl? Igen? Nåååh ja... vi skal huske at specificere vores registry. Så nu må vi hellere pulle fra vores registry:

`docker pull ip_Registry_server:5000/crypto`

Maskinen arbejder i noget tid med at hente, og også at pakke det hentede ud. Læg mærke til at maskinen i eksemplet bruger default tag der hedder latest. Det vil maskinen altid hvis man ikke specificere hvad man vil hente. Man kan altid efter **`crypto`** lave et kolon (:) og skrive version f.eks. **`crypto:2.0`** så vil maskinen hente den specificerede version, hvis den er tilstede, ellers kommer der en fejl.

Eksempel output:

```
dtmek@docker2:~$ docker pull 192.168.1.131:5000/crypto
Using default tag: latest
latest: Pulling from crypto
a480a496ba95: Pull complete
a47a1de29151: Pull complete
a0821bbad4e4: Pull complete
6be174f186fb: Pull complete
bd4787ab9f9a: Pull complete
741ae76ddle2: Pull complete
7e6f32953ef1: Pull complete
15b2eladddd6: Pull complete
6c8a58b73ea3: Pull complete
4f4fb700ef54: Pull complete
79e003456a94: Pull complete
25ca86c99b56: Pull complete
80e5be375403: Pull complete
f9c3aa6dade3: Pull complete
a9ed6492cace: Pull complete
56a92f01235d: Pull complete
87e6fd0ae349: Pull complete
0069f2b0a7c2: Pull complete
69fc7d3b7b79: Pull complete
Digest: sha256:011b8ef9a5a58f7a66ff6a813034c9095a442d557ae05e9400093abfb245178f
Status: Downloaded newer image for 192.168.1.131:5000/crypto:latest
192.168.1.131:5000/crypto:latest
dtmek@docker2:~$
```

Vi kan prøve at hente en version der ikke er der, for at se hvordan reaktionen er:

`docker pull ip_Registry_server:5000/crypto:Findesikke`

Eksempel output:

```
dtmek@docker2:~$ docker pull 192.168.1.131:5000/crypto:findesikke
Error response from daemon: manifest for 192.168.1.131:5000/crypto:findesikke not found: manifest unknown: manifest unknown
dtmek@docker2:~$
```

Nu skulle vi jo have hentet vores image. Skulle vi ikke lige se om den nu også er blevet hentet?

`docker image ls -a`

Eksempel output:

```
dtmek@docker2:~$ docker image ls -a
REPOSITORY                   TAG       IMAGE ID       CREATED         SIZE
192.168.1.131:5000/crypto    latest    0ab11fa57db4   25 hours ago    523MB
dtmek@docker2:~$
```

YES! Imaget ligger jo hvor den skal ligge, og det ser ud til at det er vores image.. Men det er jo fjollet at hente imaget, og ikke starte det. Så lad os starte imaget:

```
docker run --restart unless-stopped --name crypto -p 80:80 -td ip_Registry_server:5000/crypto
```

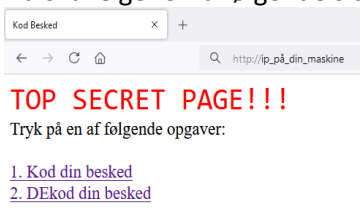
Forklaring:
-td : -t betyder at den ikke sender output til vores terminal, men til en pseudo terminal, og -d betyder som beskrevet før detached, altså vores ssh terminal bliver frigivet efter start af container.

Eksempel output:

```
dtmek@docker2:~$ docker run --restart unless-stopped --name crypto -p 80:80 -td 192.168.1.131:5000/crypto
c5ce90d23aa40de5da3dd0142545027eda20e662dba4025441b8f9d71dc3f3ad
dtmek@docker2:~$
```

Prøv nu at åbne en webbrowser på din pc, og gå igen til ip-adressen på den maskine hvor du har startet din container. I din browser skriver du **http://ip_på_worker_makine**

Du skulle gener få følgende side frem igen. Hvis siden kommer frem, er container startet igen.:



Slut på Kapitel 5: Pulle et image fra Registry.

Og det var så det sids.....

STOP! Vent lige lidt!

Vi skal lige have ryddet lidt op inden vi starter på opgave 2. Vi skal bruge nogen af de data der ligger på din maskine endnu (f.eks. biblioteket Site.). Men vi skal lige have stoppet containeren, slettet containeren, og fjernet imaget. Ellers vil der jo f.eks. være en container der spærrer for port 80. Det med at stoppe og slette har vi jo prøvet før, så lad os gøre det ved at kopiere alle tre kommandoer herunder over i vores ssh vindue på en gang!:

```
docker kill crypto
docker container rm crypto
docker image rm ip_Registry_server:5000/crypto
```

Output på næste side:

Eksempel output:

```
dtmek@docker2:~$ docker kill crypto
docker container rm crypto
docker image rm 192.168.1.131:5000/crypto
crypto
crypto
Untagged: 192.168.1.131:5000/crypto:latest
Untagged: 192.168.1.131:5000/crypto@sha256:011b8ef9a5a58f7a66ff6a813034c9095a442d557ae05e9400093abfb245178f
Deleted: sha256:0ab11fa57db4338e05cae4200c9abe284f84315bc6a808b2576dd84f09302d62
Deleted: sha256:60ff59535325ad34991f89617fe1b9a3629215b28d408303f1a617deca20fba1
Deleted: sha256:a3431ef2fedcb5bb4eef285ca9467e8b1fc47fa81faf2a278596593a44b83020
Deleted: sha256:6059232fb84d0c71f7aed52fd1d91157defc36ec4c7092fdd61a778712ec806b
Deleted: sha256:2e1ffd4728dda27cf70e936b8dc68ab994ef1dcd7619c74921600189adfea9f7
Deleted: sha256:4a3e6bdd0f0b78383a0dcce58035eafa810a9998158ce6e0e8c783ae897efbbe
Deleted: sha256:4db31b324634cfc553f3b68e20fc953bfeadff345d800ef5d95967f4457e491d
Deleted: sha256:e6d704a1f261a81223f63058c19fb77a8c05345987d5e0ddab4f285237635412
Deleted: sha256:22f494b198f59413c7ab7df819ff2dc189bd51d0bc27513cff77c90de2e3a7d8
Deleted: sha256:c447e3e1a30b61d0896148129aa804f62c143ba91ce5cb13ef83ba6a334c4789
Deleted: sha256:220532426789170ac4b4df0360ccaf1f84cf9e3b0a49f7d93b9b0e3f7606de32
Deleted: sha256:18abe958d3ad0b8a1d4cc74addefffc8a6f758cdbcc776e7b546a655bd7a3a5e
Deleted: sha256:6d224f94c6cb6c7c41f19d9c8301b0ecee480ceed4644b254e2dbb97d25020c4
Deleted: sha256:a6416480e107bd6e16ad294afd999d9ff5290f84bcba81a319f194e98a357833
Deleted: sha256:9dcb23d67d2cbd85225af31a3f30f838c0c7d4460343c3bb357d49a5b4543c24
Deleted: sha256:482a2cdf9da0c874af7f3adf48d73d0539f3601b3f1a9a32ce0f707f731d3ed5
Deleted: sha256:3b42d54bc7df0766f9dd3f1ae35dd9fc4727ab693ea28b7982610b8f4e09e604
Deleted: sha256:3ee33463d739ec069e77d92c90c72c3e3e60ae9d4e9bd120b00fa83fc9bbce6f
Deleted: sha256:0b2485625c6ee8bdb887607911ef6febb7762f56db4ef159a1feeeabef304045
Deleted: sha256:98b5f35ea9d3eca6ed1881b5fe5d1e02024e1450822879e4c13bb48c9386d0ad
dtmek@docker2:~$
```

Vi skal også lige huske en sidste oprydnings kommando:

`docker system prune`

Eksempel output:

```
dtmek@docker2:~$ docker system prune
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all dangling images
  - unused build cache

Are you sure you want to continue? [y/N] y
Total reclaimed space: 0B
dtmek@docker2:~$
```

Hvad nu?? Den lavede jo ikke noget nu! Bare rolig, det er der ikke noget galt med. Vi har jo pullet dette image, dvs. computeren har hentet alt den skulle bruge fra vores Registry server. Så der lå ikke noget andet og ”fyldte” på vores server, og alt, der var pullet fra registry var tilknyttet vores image. Se resultat ovenover da vi slettede vores image, alle layers blev slettet her!

NU kan jeg så skrive:

Slut på Kapitel 5: Pulle et image fra Registry.

Og det var så også det sidste i Opgave 1.