Labo 4: Docker

Voorbereiding

Eerst en vooral moeten we Docker installeren. Voor wie MacOS of Linux gebruikt is dit heel eenvoudig, zij kunnen dit gewoon op hun pc installeren. Voor de Windows-gebruikers komt er iets meer bij kijken. Oftewel moet je WSL (Windows Subsystem for Linux) installeren, oftewel, en dat is de optie die we in dit labo verkiezen, maken we een nieuwe VM in VMWare, waarin we een Linux Server installeren. Deze kan je downloaden via deze link.

Wanneer je dit hebt geïnstalleerd kan je Docker installeren. Gebruik hiervoor de documentatie van Docker zelf. (Linux, MacOS).

Je eerste container

Als de installatie van Docker geslaagd is, zou je het volgende commando moeten kunnen uitvoeren.

```
$ docker run hello-world
```

Zoals het commando doet vermoeden zal dit commando een container opstarten van de image 'Hello-world'. Eerst en vooral zal Docker vaststellen dat hij die image zelf niet heeft, en deze downloaden. Wanneer deze gedownload is krijg je een bericht dat Docker correct geïnstalleerd is, war extra info, en sluit het proces af.

Natuurlijk heeft deze container niet veel meer nut dan een sanity-check, dus we gaan iets ambitieuzer werken.

```
$ docker run ubuntu ls
```

Dit commando start een container van de Ubuntu-image. In deze image voeren we het commando 'ls' uit. Wat valt je op aan de output? Probeer hierna eens een Ubuntu-container te starten en de inhoud van het bestand /etc/passwd te tonen.

We kunnen een container interactief gebruiken. Hiervoor gebruiken we de opties -it (Interactive, TTY). Als je deze opties combineert met een proces dat toelaat om met de terminal te interageren (een shell, bijvoorbeeld sh of bash) kan je een terminal krijgen om zo commando's uit te voeren. Deze terminal blijft open tot je de shell afsluit, meestal door het commando "exit"

```
$ docker run -it ubuntu bash

root@0ce7733bf23c:/# ls
root@0ce7733bf23c:/# exit

$
```

Applicaties in Docker

Natuurlijk is dit leuk, maar heeft dit weinig nut. Bij containers draait alles om applicaties. We gaan proberen om een applicatie op te starten met behulp van Docker. Het IT-team van Cosci heeft van zijn website een Docker-image gemaakt. We nemen deze als een voorbeeld.

```
$ docker run -d -p 8080:80 coscicorp/website
```

Dit commando zal een wordpress-container opstarten. De optie -d zal ervoor zorgen dat deze in de achtergrond blijft draaien (Daemonized). De optie -p is een heel interessante. Iedere container krijgt van de Docker Engine zijn eigen IP-adres. Applicaties draaien zoals je geleerd hebt in Computernetwerken op een poort, websites bijvoorbeeld op poort 80 (HTTP-protocol). Met de optie -p [HOSTPORT]:[CONTAINERPORT] map je de containerport op een hostport. Dit wil zeggen dat alles wat op die container op poort 80 draait, ook op de host op poort 8080 draait. Dit betekent dat wanneer je het commando hierboven hebt uitgevoerd, en daarna surft naar http://HOST_IP:8080 surft, je normaal een instantie van de Cosci-website moet zien.

Je eigen Docker-Image

Dockerfiles

Natuurlijk is het leuk om applicaties snel te kunnen opstarten van een image, maar het is nog veel leuker om zelf je eigen applicaties te verpakken. Om eigen Docker-Images te maken, gaan we gebruik maken van Dockerfiles.

Eerst en vooral hebben we een bestand index.html nodig. Maar er een aan, zet hierin de tekst 'Hello World'. Vervolgens maken we in dezelfde map de volgende Dockerfile aan. Deze begint van de standaard-image Nginx, en kopieert ons html-bestand in de container.

```
FROM nginx
COPY index.html /usr/share/nginx/html/
```

We maken de image aan met het volgende commando

```
$ docker build -t myfirstweb .
```

Dit moeten we uitvoeren vanuit de map waarin zowel de Dockerfile als het html-bestand staan. De image krijgt als naam (tag) "myfirstweb". Wanneer je het commando 'docker images' uitvoert zal je deze ook kunnen terugvinden.

Probeer nu van deze image een container op te starten, zodat als je naar http://[HOST-IP]:8081 surft, je deze website ziet.

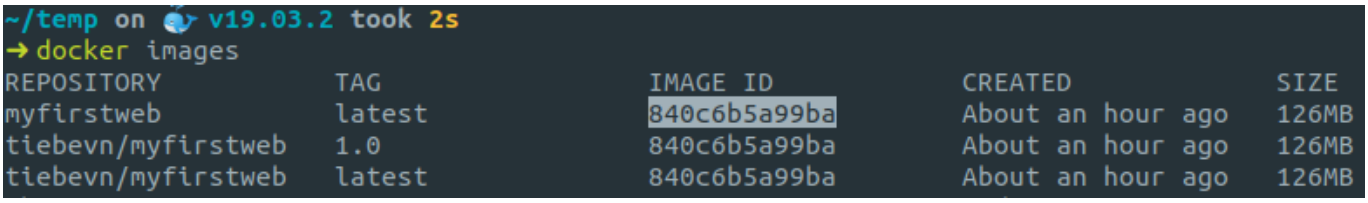
DockerHub

We hebben al gezien hoe een image vaak automatisch gedownloadet wordt. Dit komt omdat deze allemaal ergens online opgeslagen worden, in een Docker Registry. Je kan je eigen Docker Registry gebruiken, of je kan de publieke Docker Registry van Docker zelf gebruiken, DockerHub.

We beginnen eerst en vooral met het aanmaken van een account. Wanneer je dit gedaan hebt maak je een repository aan met de naam 'myfirstweb'.

Hierna moet je in de terminal een aantal dingen doen. Eerst en vooral wil je via de terminal aanmelden op DockerHub. Daarvoor moet je niet meer doen dan het commando 'docker login' uit te voeren.

Vervolgens willen we een bepaalde tag toekennen aan onze image. Om dit te doen moeten we eerst en vooral de ID van onze image hebben.



Met deze ID kan je de image manipuleren.

```
$ docker tag [IMAGE_ID] [USERNAME]/[REPOSITORY_NAME]:[VERSION]

$ docker tag 840c6b5a99ba tiebevn/myfirstweb:1.0
$ docker tag 840c6b5a99ba tiebevn/myfirstweb:latest

$ docker push tiebevn/myfirstweb
```

Hier krijgt de image twee tags met verschillende versies. Dit is omdat Docker standaard de image met de versie 'latest' gaat zoeken als de versie niet bepaald wordt.

Probeer nu een aanpassing te doen aan de image (HTML wijzigen), een nieuwe image te builden, deze te pushen als versie 1.1, en de tag latest ook aan de nieuwe image te geven. Controleer dan of je beide versies kan binnenhalen.

Oefening 1: Tomcat-WAR Image

Natuurlijk gaat dit niet alleen over HTML. Er bestaan ook Docker-images voor bijvoorbeeld Tomcat. Probeer nu je project voor Web3 eens in Docker te steken: Schrijf een Dockerfile waarmee je een WAR-bestand naar de juiste plaats in Tomcat schrijft, en zet deze image op DockerHub.

Volumes

Stel nu dat we in plaats van onze website te publiceren, we graag de Docker Image van Nginx willen gebruiken om te testen terwijl we aan deze website werken. In dat geval is het niet handig om voor iedere aanpassing opnieuw Docker Build te runnen, een nieuwe container te maken.

Hier kunnen we een ander feature van Docker benutten, volumes. Met volumes kunnen we bepaalde mappen of bestanden van en naar ons Host OS linken. Alle aanpassingen die we dan op onze host doen komen zo ook terecht in de container en andersom.

```
$ docker run -d -v /path/to/html:/usr/share/nginx/html/ -p 80:80 nginx
```

Wanneer je nu aanpassingen doet in je index.html, en nadien de webpagina vernieuwt, zou je de aanpassingen moeten zien.

Microservices

VIDEO: What are Microservices

In Docker gebruiken we Docker Compose voor het beschrijven van Microservice architecturen. Docker Compose vertekt altijd vanuit een docker-compose.yml bestand, en kan eruit zien zoals hieronder.

```
version: '3.7'

services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
       WORDPRESS_DB_NAME: wordpress
volumes:
    db_data: {}
```

In dit voorbeeld maken we 2 containers. Een database-container, en een Wordpress-container. Eerst beschrijven we de database-container.

We bepalen eerst dat de database een MySQL-image wordt. Met een volume zorgen we ervoor dat moest de container crashen, de data niet verloren gaat, doordat er een kopie op de host staat. De restart-policy is 'always', dus als de container zou crashen, wordt er automatisch een nieuwe gemaakt. Daarnaast geven we enkele variabelen mee die we nadien nodig hebben om de applicatie te koppelen.

In de applicatie beginnen we met een depends-on statement. Dit zegt Docker dat de db-container moet bestaan voordat hij aan de wordpress mag beginnen. Hier zie je ook bij de variabelen staan dat de DB_HOST ingesteld staat als 'db:3306'. Het deel 'db' verwijst naar de andere container. Dit kan, doordat Docker achterliggend zijn eigen netwerken gebruikt met DNS, dus je hoeft je geen zorgen te maken over het IP van de container, de naam volstaat.

Oefening 2

Probeer nu, met bovenstaande voorbeeld als inspiratie, je eigen web3-project met database in een Docker-compose file te steken.