Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

ILKKA SIRNIÖ

Sovelluskontti web-sovellukselle

TIETOJENKÄSITTELYN KOULUTUSOHJELMA

2020

Tekijä(t)
Sirniö, Ilkka-Kristian

Julkaisun laji
Opinnäytetyö, AMK

Päivämäärä
Toukokuu 2020

Sivumäärä
33

Julkaisun kieli
Suomi

Julkaisun nimi
Sovelluskontti web-sovellukselle

Tutkinto-ohjelma
Tietojenkäsittelyn koulutusohjelma

Tässä opinnäytetyössä luotiin Bootstrapilla pieni web-sovellus, joka laitettiin Docker-sovelluskonttissa suoritukseen Googlen pilvialustalla olevaan Cloud Shelliin.

Opinnäytetyön toteutuksessa käytettiin seuraavia työkaluja ja palveluita: CSS, Docker, HTML ja Google Cloud.

Asiasanat haetaan asiasanaluettelosta, mutta siihen ei tehdä linkitystä

Docker, Google Cloud, Sovelluskontti

Author(s) Sirniö, Ilkka-Kristian	Type of Publication Bachelor's thesis	Date May 2020
	Number of pages 33	Language of publication: Finnish
Title of publication **Docker container for web application**		
Degree programme Programme in Business Information Systems		
The purpose of this thesis was to create a small web application with Bootstrap framework and containerize it with Docker and run it on Google Cloud Shell. Some used techniques and services: CSS, Docker, Google Cloud and HTML.		
Key words search from key word list but not link Container, Docker, Google Cloud		

SISÄLLYS

TERMIT JA LYHENTEET

API-Key	Application programming key on uniikki tunniste, jolla autentikoidaan käyttäjä tai kutsuva ohjelma ohjelmistorajapintaan. Useimmiten käytetään autentikoimaan projektia kuin henkilöä.
CLI	Command Line Interface on komentorivi käyttöjärjestelmien ja ohjelmien käyttöliittymäksi.
Container	Container on Bootstrapin perus asetteluelementti, joka mahdollistaa lokerikkojärjestelmän (grid system) käytön.
Front-end	Front-end tarkoittaa asiakkaalle näkyvää koodia, jolla on tuotettu esimerkiksi web-sivut ja sovellusten käyttöliittymät käyttäen HTML, CSS ja JavaScript-kieliä.
Localhost	Localhost tarkoittaa paikallista konetta, jolla ohjelmaa suoritetaan. Localhost käyttää IP-osoitetta 127.0.01.
Hypervisor	Hypervisor on ohjelma, joka mahdollistaa useamman virtuaalikoneen käytön laiteinfrastruktuurissa.
PowerShell	Windows PowerShell on komentorivityökalu, jolla voidaan automatisoida tehtäviä ja suorittaa skriptejä.

1 JOHDANTO

Sovellustuotanto on siirtymässä yhä enemmän ja enemmän mikropalveluiden suuntaan. Ajan henki on luoda ketteriä ja hyvin skaalautuvia sovelluksia, joita voidaan päivittää pala palalta. Aiemmin sovellustuotannossa sovellukset noudattivat monoliittistä arkkitehtuuria. Monoliittisellä arkkitehtuurilla toteutetut sovellukset ovat yleensä isoja, kankeita ja vaikeasti hallittavia. Monoliittiset sovellukset ovat haastavampia ylläpitää ja ne aiheuttavat käyttäjille käyttökatkoja, koska sovellus voidaan joutua ottamaan pois käytöstä esim. päivitettäessä sovellusta.

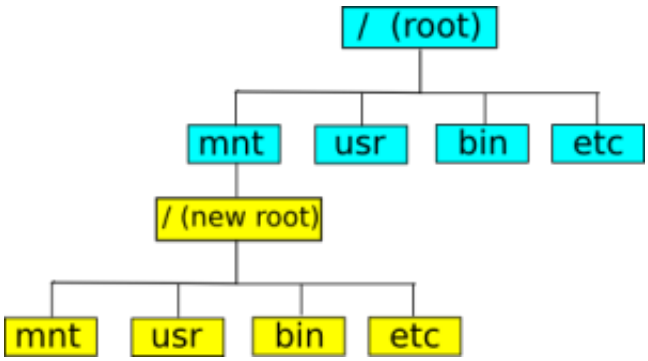
Uudet menetelmät vaativat uusia työkaluja. Jotta mikroarkkitehtuuria voitaisi hyödyntää sovellustuotannossa ja saada tasalaatuista koodia, joka käyttäytyy samalla tavalla kehitys- ja tuotantovaiheessa, niin tarvitaan uusi menetelmiä sovellustuotantoon. Docker-sovelluskontti on oiva ratkaisu sovellustuotantoon, joka mahdollistaa sovelluksen pilkkomisen pienempiin osiin. Käyttämällä sovelluskonttia voidaan olla varmoja siitä, että sovellus tai sen osa käyttäytyy samalla tavalla kehitys- ja tuotantoympäristössä. Kehittäjä voi asentaa Docker- ohjelman omalle tietokoneelle ja tuottaa koodia ja siirtää sen vaivatta palvelimelle tai esim. pilviympäristöön, joka on voitu valmiiksi optimoida sovelluskonteille kuten esim. Kubernetes.

Tässä opinnäytetyössä käydään läpi tekniikoita, joilla tuotetaan pieni web-sovellus, joka laitetaan sovelluskonttiin ja kontti suoritukseen Google Cloud pilvialustalle. Sovellus tehdään Bootstrap-kirjastolla. Ensin tutustutaan Dockeriin ja sen historiaan. Sen jälkeen tulee pieni esittely käytetyistä tekniikoista, josta siirrytään itse sovelluksen kontittamiseen, sovelluksen tallentamisen Docker Hubiin ja lopuksi sovelluksen käynnistämiseen Google Cloud Shellillä. Aivan viimeiseksi seuraa omaa pohdintaa projektista.

2 DOCKER

2.1 Dockerin historia

Sovelluskonteissa (application containers) käytettävä idea eristää sovelluksia ja prosesseja ei ole uusi, vaikkakin se tällä hetkellä on kuuma puheenaihe monessa sovellusten toteuttajien kahvipöydässä. Vuonna 1979 otettiin ensimmäisiä askelia kohti jaettuja ja eristettyjä ympäristöjä. Unix-käyttöjärjestelmän komento chroot (change root) teki mahdolliseksi eristää järjestelmän prosesseja (system process) omiksi tiedostojärjestelmiksi, mikä mahdollisti testaamisen ilman vaikuttamista globaaliin ympäristöön. Kuvassa yksi on havainnollistettu eristetty tiedostojärjestelmä. (Marquez 2018.)



Kuva 1. Chroot havainnekuva. (Go4it2day Web Services 2017.)

Solomon Hykes perusti vuonna 2008 ystäviensä Kamel Founandin ja Sebastien Pahlin kanssa toisen sukupolven platform-as-a-service (PaaS) yrityksen nimeltään dotCloud. Heidän tarkoituksensa oli luoda ohjelmointityökaluja, joita kuka tahansa voisi käyttää. He alkoivat kehittää konttiteknologiaa (containers). (Hykes 2018.)

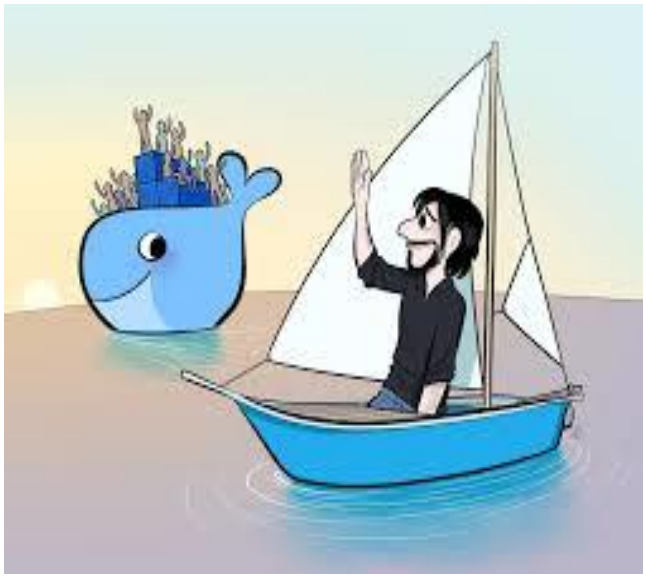
Vuonna 2013 Hykes alkoi työstää projektia nimeltään Docker yhdessä Francois-Xavier Bourletin, Andrea Luzzardin ja useiden muiden dotCloudissa työskentelevien insinöörien kanssa. Maaliskuussa 2013 Docker virallisesti lanseerattiin ja Hykes julisti, että dotCloudin pääfokus oli Dockerissa. Syyskuussa 2013 Red Hat ja dotCloud julistivat teknisen yhteistyön alkaneeksi. Samoihin aikoihin jo 10 000 kehittäjää oli kokeillut Dockeria. 2014 heinäkuuhun mennessä Dockeria oli ladattu 2,75 miljoonaa kertaa.

Ajoitus ei olisi voinut olla parempi, koska yhä useampi yritys oli investoimassa pilveen ja Red Hat ja Amazon tarjosivat kaupallista tukea. (Red Hat 2013)

Seuraavien vuosien aikana Docker sai merkittäviä sijoituksia. Monet rahoituskierrokset tuottivat kymmeniä miljoonia dollareita. Rahoituskierroksia johtivat sellaiset yritykset kuin Sequoia Capital, Greylock Partners ja Insight Venture Partners.

Docker teki merkittäviä yritysostoja, joiden avulla se paransi valmiuksiaan kehittää tuotettaan ja vahvistaakseen asemaansa markkinoilla. Ostettuihin yrityksiin kuuluivat Tutum, Kitematic, Unikernel Systems, SocketPlane, Koality, Orchard ja Conductant. Marraskuuhun 2015 mennessä Dockeria oli ladattu yli miljardi kertaa. (Cleverism 2013.)

Solomon Hykes, joka oli yksi alkuperäisistä perustajista ilmoitti 28.3.2018 astuvansa sivuun päivittäisjohtamisesta (kuva 2). Lähtöön ei liittynyt dramatiikkaa, vaan hän halusi keskittyä perheeseensä, ystäviinsä ja sijoittamiinsa yrityksiin. Hän pysyy edelleen aktiivisena johtokunnan jäsenenä ja merkittävänä osakkeenomistajana. (Hykes 2018.)



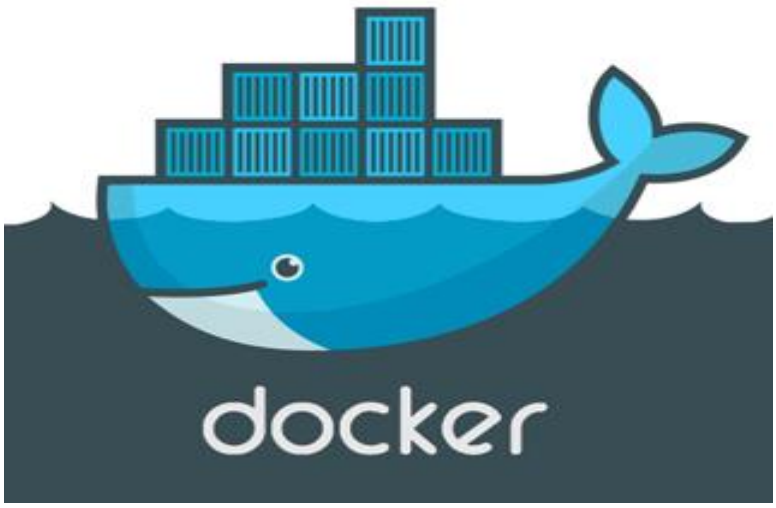
Kuva 2. Docker Founder, dotcloud, Solomon Hykes (Au Revoir 2018.)

2.2 Yleistä Dockerista

Virtualisoinnin tarve on lisääntynyt merkittävästi viime vuosina. Käyttäjien määrät pilvipalveluissa ovat kasvaneet siksi, että käyttäjät ja yritykset ovat siirtäneet toimintojaan pilviympäristöön. Aiemmin tuotanto- ja pilviympäristöissä virtuaalikoneet ovat muodostaneet selkärangan, mutta rinnalle on noussut konttiteknologia, joka mahdollistaa kevyemmät ratkaisut.

It-maailmassa on tapahtumassa perusteellinen muutos siinä, miten sovelluksia kehitetään, jaetaan ja ylläpidetään eri palvelinjärjestelmissä. Konttiteknologia on ideana samanlainen kuin tavaraliikenteen rahtikontti. Sovellus pakataan konttiin kaikkine riippuvuuksineen ja laitetaan suoritukseen esimerkiksi pilvialustalle.

Vertaus rahtikonttiin on todella osuva (Kuva 3). Kontteja on helppo ottaa lisää kyytiin ja poistaa tarpeen mukaan. Tietokoneen konttijärjestelmä toimii tismalleen samalla lailla. (Kotilainen 2017.)



Kuva 3. Docker whale Moby Dock. (Call Me Moby Dock 2013.)

Ohjelmistokonteilla on yhteinen alusta, Docker-ohjelma, jossa ajetaan käyttöjärjestelmästä riippuen Windowsin tai Linuxin ydintä, kerneliä. Alustan päällä ajetaan kontteja, jotka toimivat itsenäisesti ja riippumatta toisistaan. Kuvassa kolme on Dockerin logo, joka kuvaa konttiteknologiaa.

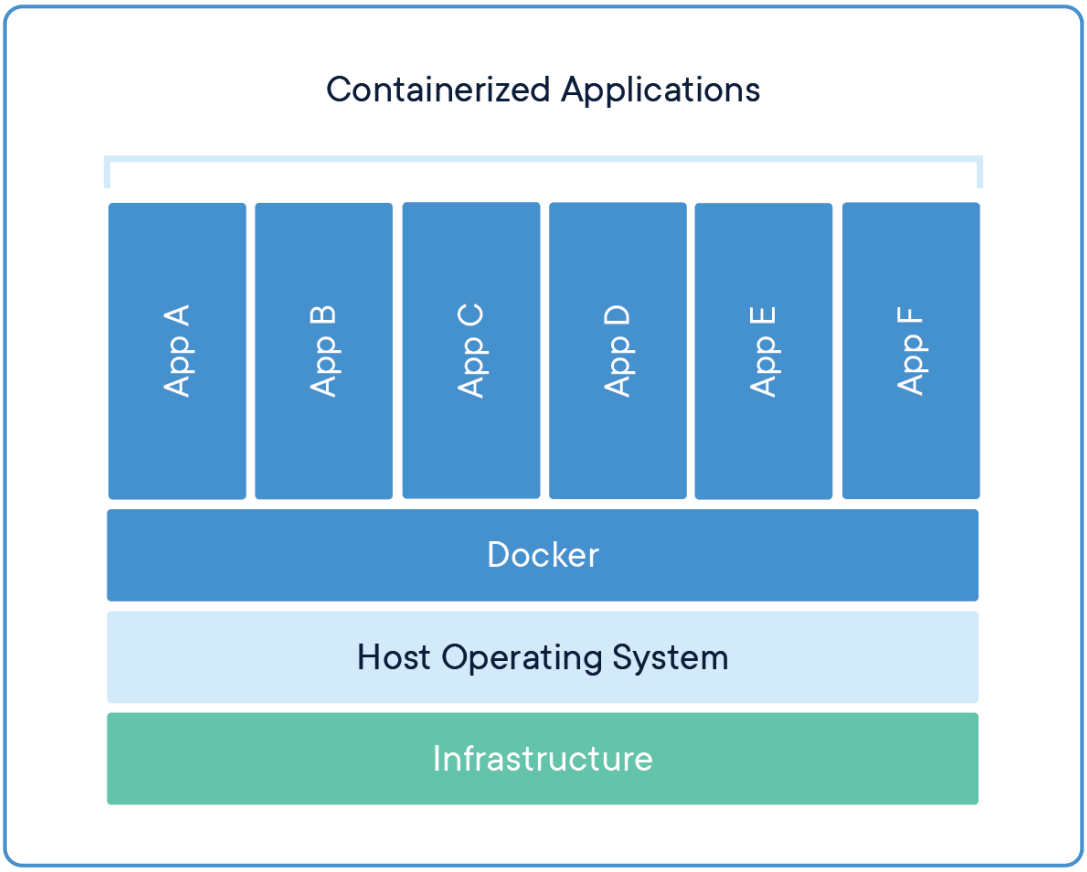
Konttiin on pakattu kaikki tarpeellinen lukuun ottamatta käyttöjärjestelmän ydintä. Kontissa on riisuttu versio käyttöjärjestelmän muista osista ja konttiin laitetaan mukaan tarvittavat ohjelmakirjastot, alustapalvelut ja sovellus. (Kotilainen 2017)

Normaalisti jokainen sovellus, joka lisätään osaksi käyttöjärjestelmään asentaa omia asetuksiaan ja osiaan. Tämä vaikuttaa käyttöjärjestelmään ja toisiin sovelluksiin, eikä ole mitenkään epätavallista saada käyttöjärjestelmä niin sekaisin, että ainoa toimiva toimenpide on käyttöjärjestelmän puhdas asennus. (Kotilainen 2017)

Nykyään palvelimissa käytetään virtualisointia ratkaisuna edellä mainittuun ongelmaan. Käytettäessä esimerkiksi VMwaren tuotteita, joihin asennetaan virtuaalisia tietokoneita, joissa on omat alustapalvelut, käyttöjärjestelmät ja sovellukset. Tällä tavalla sovellukset voidaan eristää toisistaan tehokkaasti. (Kotilainen 2017)

Virtuaalikoneilla on omat haasteensa. Jokainen virtuaalikone tarvitsee oman käyttöjärjestelmän, mikä vaatii suorituskykyä ja muistia. Virtuaalikoneita on helppo ostaa eri pilvipalveluiden tarjoajilta esim. Amazon, Microsoft Azure tai Google Cloud, mutta jokainen kone maksaa erikseen ja vaati resursseja niiden päivittämiseen, käyttöjärjestelmien asentamiseen ja muuhun ylläpitoon, joka itsessään aiheuttaa kustannuksia. (Kotilainen 2017)

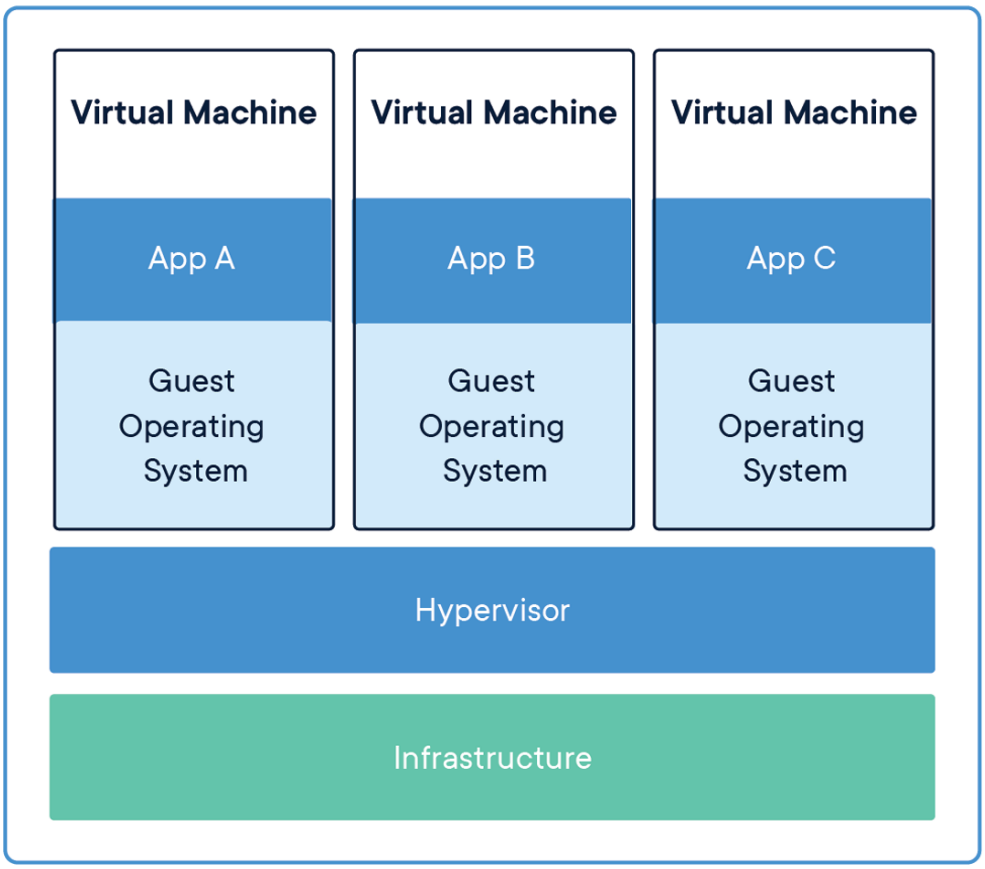
Konttitekniikan suurimpia etuja on konttien mutkaton ja nopea käsittely. Kontit saadaan nopeasti tuotantoon ja ongelmien ilmetessä ne saadaan nopeasti sammutettua. Konttien käynnistyminen ja sammuttaminen vie sekunteja, kun taas virtuaalikoneissa puhutaan helposti minuuteista tai kymmenistä minuuteista. Devops-malliin tällainen joustavuus luo uusia mahdollisuuksia. Kyseisessä mallissa pyritään nopeuttamaan ja tehostamaan sovelluskehityksen (development) ja tuotannon (operations) yhteistoimintaa ja konttiteknologia tukee tätä merkittävästi. (Kotilainen 2017.)



Kuva 4. Containers (What is a Container 2019.)

Kuvassa neljä esitetään kuusi kontitettua sovellusta. Kuvasta voi nähdä erilaiset kerrokset, jotka koostuvat infrastruktuurista, käyttöjärjestelmästä, Docker-ohjelmasta ja sen päällä olevista sovelluksista. Sovellukset ovat erillään toisistaan ja niitä voidaan tarpeen mukaan poistaa ja käynnistää ilman, että ne vaikuttavat muihin ajossa oleviin sovelluksiin. Tyypillisesti kontit vaativat vähemmän tilaa kuin virtuaalikoneet. Konttien imaget (container images) ovat normaalisti kymmeniä megatavua (MB) suuruudeltaan. (What is Container 2019.)

Jos verrataan kuvia neljä ja viisi niin huomataan, että kuvassa neljä laiteinfrastruktuurin päällä on muutama kerros vähemmän kuin kuvassa viisi, mikä mahdollistaa nopeamman suorituskyvyn.



Kuva 5. Virtual Machines (What is a Container 2019)

Kuvassa viisi on kuvattu virtuaalikoneita palvelimella. Yhdelle palvelimelle on asennettu kolme erillistä virtuaalipalvelinta. Hypervisor mahdollistaan useamman virtuaalikoneen toimimisen palvelimella. Jokaisessa virtuaalikoneessa on oma käyttöjärjestelmä, vaadittavat kirjastot ja itse sovellus vieden kymmeniä gigatavuja (GB). Virtuaalikoneiden käynnistäminen voi olla myös hidasta.

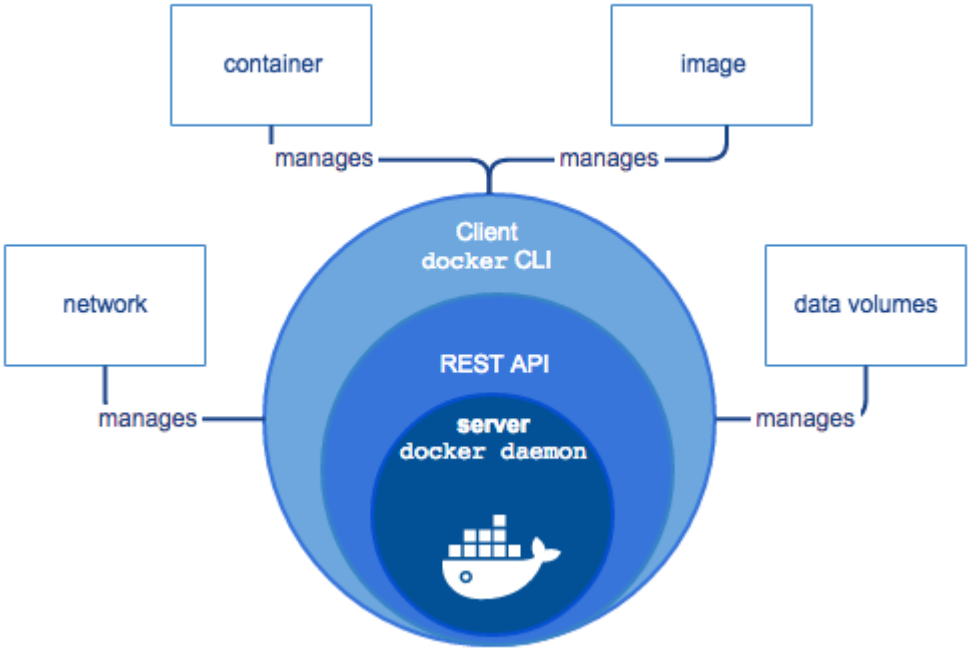
2.3 Docker Engine

Docker Engine (kuva 6) on asiakas-palvelinsovellus (client server application), joka koostuu seuraavaista komponenteista:

1. REST API määrittää ne rajapinnat, joiden avulla voidaan operoida daemon-komponentin kanssa (dockerd) ja antaa käskyjä sille.

2. Command line interface (CLI) on komentojen rajapinta, jonka avulla keskustellaan daemonin-komponentin kanssa ja jolla hallitaan kontti-instansseja.

3. Docker daemon on jatkuva taustaprosessi, joka hoitaa kontteja, verkkoja, Docker-imageja ja datavarastoja. Se kuuntelee jatkuvasti REST API- pyyntöjä ja prosessoi niitä.
   (Docker overview 2019.)



Kuva 6. Docker Engine (Docker overview 2019)

2.4 Dockerin arkkitehtuuri

Docker perustuu asiakas-palvelinarkkitehtuuriin (kuva 7). Docker client (asiakas) keskustelee palvelimen Docker daemonin kanssa, joka tekee kaiken mitä tarvitaan Docker-ympäristön suorittamiseksi.

Docker client (asiakas) mahdollistaa kommunikoinnin Dockerin (palvelin) kanssa. Docker client (asiakas) voi olla samassa koneessa (host) tai ottaa yhteyttä etänä olevaan koneeseen (remote host). Docker client (asiakas) tarjoaa komentorivin, jolla voi antaa käskyjä Docker daemonille (palvelin). Daemon rakentaa, ajaa ja pysäyttää sovelluksia. Pääasiassa komentorivin tarkoitus on tarjota mahdollisuus, jolla vetää imageja rekistereistä ja ajaa niitä Dockerissa (palvelin). Yleisimpiä käskyjä ovat: docker build, docker pull ja docker run.

Docker host tarjoaa ympäristön suorittaa ja ajaa sovelluksia. Se sisältää Docker daemonin, imaget, storaget, kontit ja verkot, jotka käsitellään kohdassa Docker-objektit. Daemon rakentaa kontti-imaget Docker clientin (asiakas) pyynnöstä. Vetäessään pyydetyn imagen, se rakentaa toimivan kontin mallin käyttäen hyväksi ohjeita, jotka on tallennettu ns. rakennustiedostoon (build file). Rakennustiedosto voi sisältää myös ohjeita daemonille muiden komponenttien lataamiseen etukäteen.

Docker-objekteja ovat seuraavat:

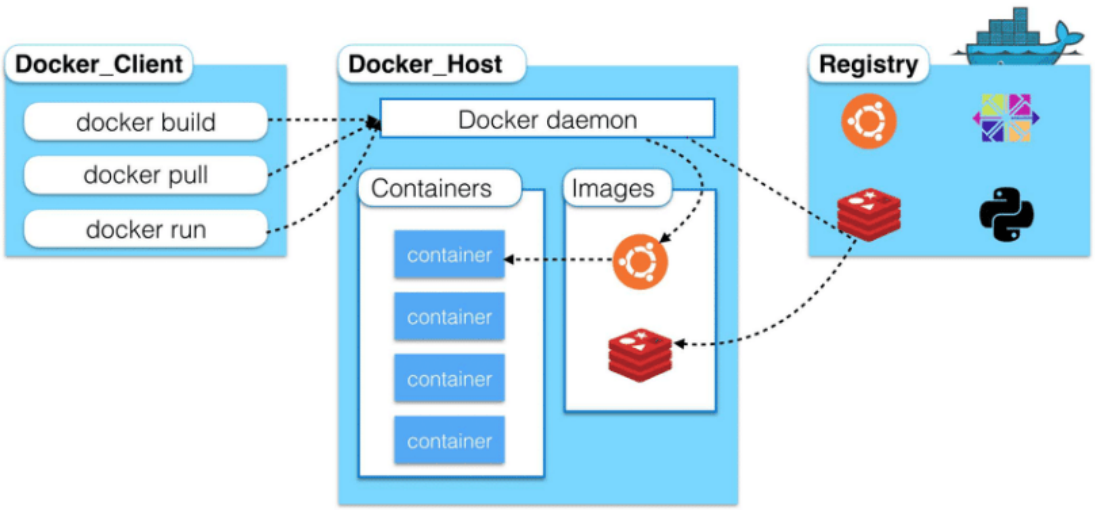
1. Imaget ovat luettavia (read-only) binäärimallinteita, joilla rakennetaan kontteja. Imaget sisältävät metadataa, joka kuvailee kontin kapasiteetin ja tarpeet. Imageja käytetään sovellusten säilömiseen ja ne mahdollistavat sovellusten ja niiden riippuvuuksien helpon siirtelyn. Imageja voidaan jakaa esim. yrityksen sisällä privaatisti rekisterin (registry) kautta tai niitä voidaan jakaa globaalisti julkisen rekisterin, kuten esim. Docker hubin. Imaget mahdollistavat yhteistyön kehittäjien kesken tavalla, joka ei ollut mahdollista ennen.

2. Kontit ovat kapseloituja ympäristöjä, joissa sovelluksia ajetaan. Image on määritellyt kontin ja muut lisäasetukset käynnistyksen yhteydessä esim. Storage-vaihtoehdot ja verkot.

3. Verkot, joita on pääasiassa kahta eri tyyppiä: käyttäjän määrittelemät ja oletuksena olevat Dockerin verkko. Oletuksena on kolme eri vaihtoehtoa Dockerin asennuksessa: -none, bridge ja host. None- ja host-verkot ovat osa Dockerin verkkopinoa. Bridge-verkko luo automaattisesti yhdyskäytävän ja IP-aliverkon. Kaikki kontit, jotka kuuluvat tähän verkkoon voivat kommunikoida

IP-osoitteiston avulla. Tämä verkko ei ole yleisesti käytössä, koska se ei skaalaudu hyvin ja sillä on rajoituksia palvelun löydettävyydessä ja verkon käytettävyydessä. Käyttäjän määrittelemissä verkoissa on kolme mahdollisuutta. Bridge, joka on kuin edellä, mutta sillä ei ole tarvetta porttien uudelleenojaukselle konteille, jotka ovat verkossa. Toinen ero on verkon automaattinen löytäminen. Overlay-verkkoa käytetään, jos halutaan eri palvelimissa olevien konttien kommunikoivan keskenään. Swarm-asetus täytyy silloin olla päällä Docker Enginessä mahdollistaen liittymisen samaan klusteriin. Macvlan-verkko poistaa sillan (bridge) ja ilman porttiohjausta paljastuu ulkoisille verkoille käyttäen IP-osoitteen asemesta MAC-osoitteita.

4. Storage mahdollistaa datan tallennuksen kirjoitettavaan kerrokseen kontissa, mutta se vaati ns. storage-ajurin. Koska storage ei ole pysyvää, niin data häviää aina, kun kontti ei ole käynnissä. Lisäksi näitä tietoja ei ole helppo siirtää. Docker tarjoaa neljä pysyvää tapaa; joita ovat data volumes, data volume container, directory mounts ja storage plugins. Data volumes tarjoaa mahdollisuuden luoda pysyviä storageja, joissa on mahdollisuus nimetä uudelleen, listata tietomäärät ja listata kontti, joka on yhteydessä dataan. Tietovolyymit ovat palvelimen tiedostojärjestelmässä, konttien kopion ulkopuolella kirjoitusmekanismilla ja ovat melko tehokkaita. Data volume Container on vaihtoehtoinen tapa, jossa dedikoitu kontti toimii datan isäntänä ja on kytketty toisiin kontteihin. Datakontti on siis itsenäinen ja se voidaan jakaa useamman kontin kesken. Directory mounts on yksi vaihtoehto liittää palvelimen paikallinen hakemisto konttiin. Edellä mainitussa tapauksessa datan tulisi olla Docker volumes kansiossa, mutta Directory mounts-tapauksessa mikä tahansa hakemisto palvelimessa käy datan lähteeksi. Storage pluginit tarjoavat kyvyn liittyä ulkoisiin storage-alustoihin. Nämä pluginit kartoittavat tallennusta isännästä ulkoiseen lähteeseen, kuten tallennusryhmään tai laitteeseen.
5. Docker Registry on palvelu, joka tarjoavat paikkoja, mihin säilöä ja ladata imageja. Toisin sanoen Docker Registry sisältää säilytyspaikkoja (repositories), jotka isännöivät yhtä tai useampaa Docker imagea. Julkisiin Docker Registry-objekteihin kuuluu Docker Hub ja Docker Cloud, mutta myös yksityisiä säilytyspaikkoja voi kanssa käyttää. (Docker Architecture 2019)



Kuva 7. Dockerin arkkitehtuuri. (Docker Architecture 2019)

3 KÄYTETTÄVÄT TEKNIIKAT

3.1 Bootstrap

Bootstrap on maailman eniten käytetty käyttöliittymäkirjasto (front-end). Bootstrap on avoimeen lähdekoodin perustuva komponenttikirjasto, joka perustuu HTML, CSS ja JavaScript-kieliin. Bootstrap käyttää hyväkseen lokerikkojärjestelmää, jossa ruudun leveys on jaettu kahteentoista yhtä suureen osaan. (Bootstrap 2020)

Bootstrap kehitettiin vuonna 2010 Mark Otton ja Jacob Thorntonin toimesta, jotka työskentelivät Twitterillä. Virallinen julkaisu oli 19.8.2011 ja sen jälkeen julkaisuja on ollut yli kaksikymmentä. Bootstrap on uudelleen kirjoitettu kolmeen kertaan: v2, v3 ja v4. (Bootstrap About 2020)

```
k! --Google Map and Contact info -->
<div class="container-fluid">
    <div class="row">
        <div class="col-sm-6">
            <iframe width="600" height="400" frameborder="0" style="border:0"
        </div>
        <div class="col-sm-6" style="background-color: #95999c">
            <h2 style="">Yhteystiedot</h2>
            <adress>
                <strong>Kennel Hardheads</strong> <br>
                Ilkka Sirniö<br>
                Tuomelanpolku 6<br>
                28660 Pori<br>
                Finland<br>
                <abbr title="phone">P:</abbr>0458027375
            </adress>
        </div>
    </div>
</div>
```

Kuva 8. Bootstrap.

Kuvassa kahdeksan on toteutettu Bootstrapilla yhteystiedot. Bootstrapissa on valmiiksi olemassa luokkia, joita on helppo ottaa käyttöön HTML-elementeissä. Kuvan Container pitää sisällään kaksi yhtä suurta Div-elementtiä, joista toisessa on toteutettu Google Map kartta, josta on rajattu vaadittava API-key pois. Toisessa taas on yhteystiedot harmaalla pohjalla. Div-elementtien class-attribuuttien arvo col-sm-6 tarkoittaa, että yksi div-elementti vie tilaa 6 lokeroa Bootstrapin lokerikkojärjestelmä kahdestatoista lokerosta eli ruutu jakautuu kahteen yhtä suureen div-elementtiin yllä olevassa kuvassa kahdeksan.

3.2 CSS

CSS on kieli, jolla määritellään HTML-dokumentin käyttämät tyylit. CSS kuvaa, miltä HTML-elementtien tulisi näyttää esimerkiksi: fonttien koot, kuvien koot, käytettävät värit ja käytettävät fontit.

Tyylit voidaan lisätä kolmella eri tavalla. Esimerkiksi kuvassa yhdeksän ne on lisätty suoraan HTML-tiedostoon. Kuvassa 10 ne ovat erillisessä tiedostossa, johon viitataan kuvassa 11 näkyvässä head-elementissä. Kuvassa 12 tyylit ovat kirjoitettu HTML-elementtiin. (w3schools 2020.)

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        body {
            background-color: aqua;
        }

        h1 {
            color: whitesmoke;
            margin-left: 40px;
        }
    </style>
</head>
<body>
<h1>Heading</h1>

</body>
</html>
```

Kuva 9. CSS määritelty HTML-tiedostoon.

```css
body
{
    background-color: red;
}
```

Kuva 10. CSS erillisessä tiedostossa.

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet" type="text/css" href="site.css">
</head>
<body>

</body>
</html>
```

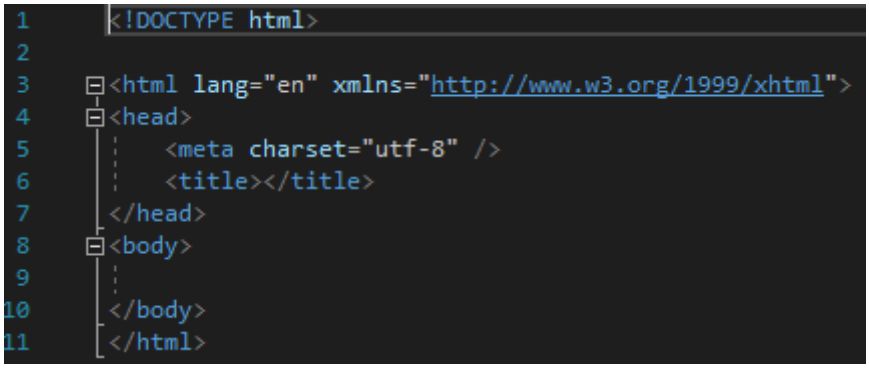
Kuva 11. Erillinen tiedosto sisällytetään HTML-dokumenttiin.

Kuva 12. CSS määritelty suoraan HTML-elementtiin.

3.3 HTML

HTML tulee sanoista Hypertext Markup Language, jonka kehitti Sir Tim Berners-Lee loppuvuodesta 1991, mutta virallisesti HTML otettiin yleiseen käyttöön vuonna 1995 versiona HTML 2.0. (HTML History 2020.)

HTML on merkkauskieli, jolla kuvaillaan web-sivun rakenne. HTML elementit kertovat selaimelle, miten sisältö tulisi näyttää. HTML elementit esitetään tageina. Kuvassa 13 on havainnollistettu web-sivun rakenne. (HTML Introduction 2020)



Kuva 13. HTML sivun rakenne.

3.4 Google Cloud

Google on tuttu yrityksenä monelle maailman suosituimmasta hakukoneesta ja esimerkiksi sähköposti Gmailista. Larry Page ja Sergey Brin perustivat Googlen vuonna 1998

saatuaan rahoitusta 100 000 dollaria yhdeltä Sun Microsystemsin perustajista Andy Bechtolsheimilta. (From the garage to the googleplex 2020.)

Tasan kymmenen vuotta siitä Google lanseerasi Google Cloud pilvipalvelualustan, jossa kehittäjät pystyivät ajamaan web-sovelluksia Googlen infrastuktuurissa. Tarkoitus oli helpottaa uuden web-sovelluksen alkuun pääsyä ja lisätä kapasiteettia siinä vaiheessa, kun sovelluksen käyttäjämäärät ja liikenne kasvavat merkittävästi. (Google App Engine Blog 2008.)

Tässä projektissa sovelluskontti otetaan suoritukseen Google Cloud Shellillä. Cloud Shell on maksuton ja vaati ainoastaan tunnusten luomisen Google Cloudiin.

Cloud Shell on debian-pohjainen virtuaalikone, jossa on 5 GB ilmaista tallennustilaa $HOME-hakemistossa. Levylle tallennetut tiedot ovat käyttäjän omia, eikä niihin pääse käsiksi muut käyttäjät. Cloud Shelliin on valmiiksi asennettu paljon nykyaikaisia työkaluja, joita käytetään yleisesti sovellustuotannossa. (Google Cloud 2020)

4 KÄYTÄNNÖN TOTEUTUS

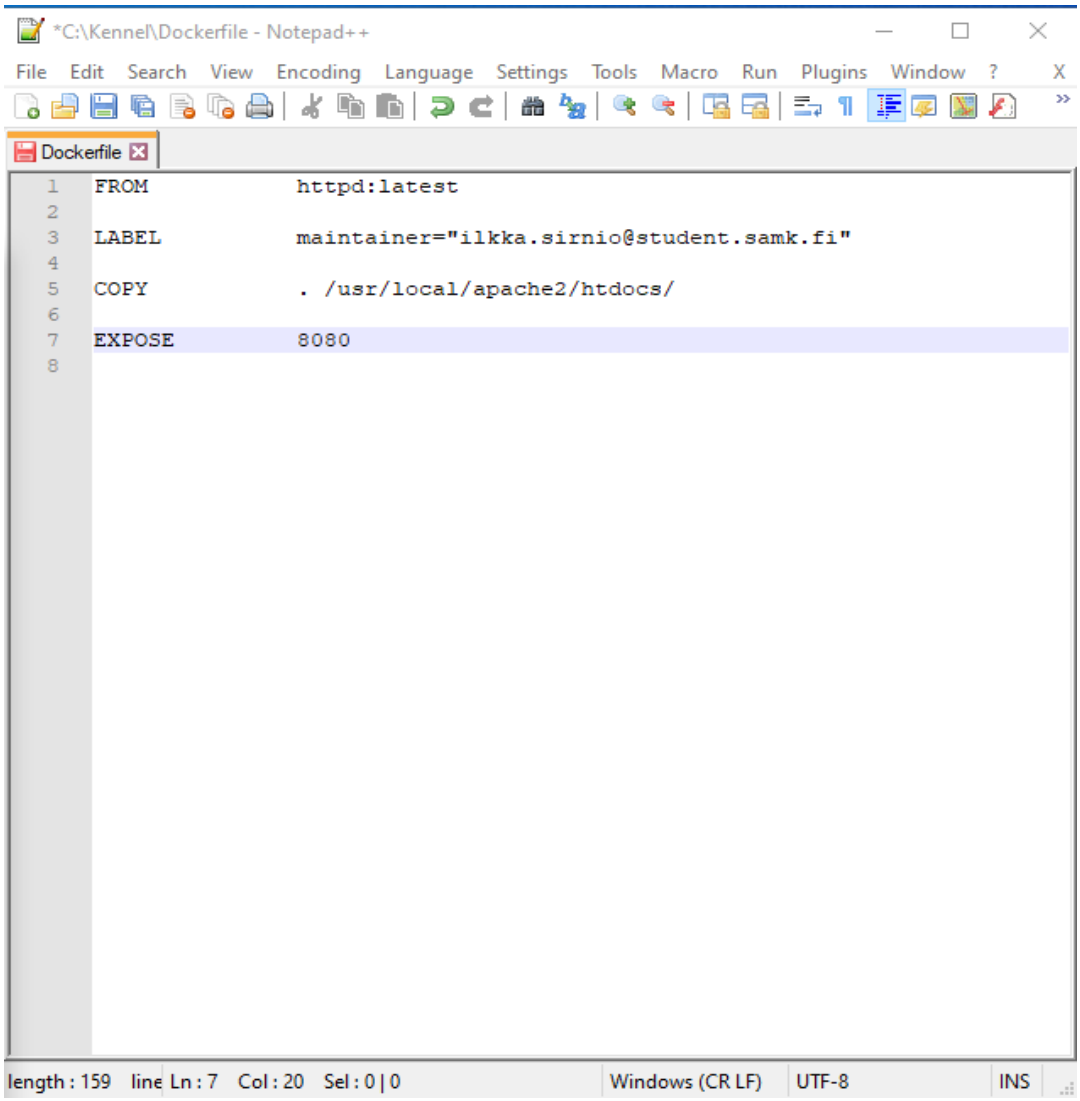
4.1 Docker Imagen rakentaminen

Käytännön toteutus aloitetaan lataamalla ja asentamalla Docker työasemalle ja sen asennus on nopeaa, helppoa ja suoraviivasta. Sen saa ladattua eri käyttöjärjestelmille osoitteesta: https://docs.docker.com/get-docker/ .

Aiemmin projektissa tehtiin Bootstrapillä yksinkertainen web-sovellus, jolla on tarkoitus havainnollistaa Dockerin käyttöä. Seuraavaksi on tarkoitus rakentaa kansiosta, joka sisältää projektin HTML, CSS- ja kuvastiedostojen image, josta myöhemmässä vaiheessa saadaan itse kontti, jota voidaan suorittaa esim. localhostissa.

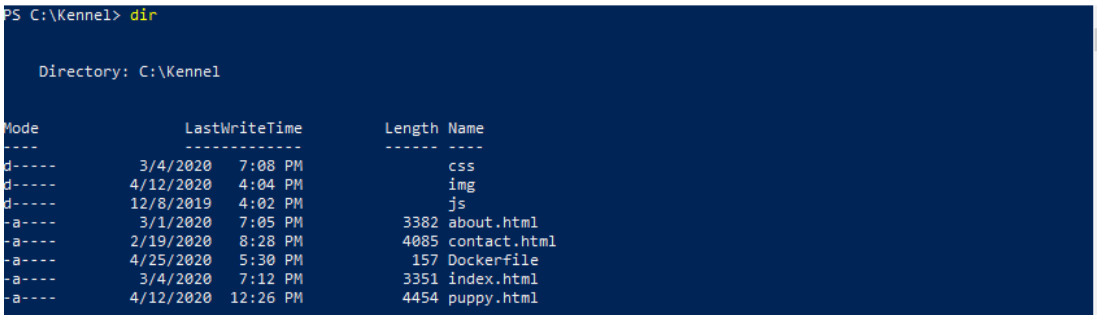
Web-sivun esittämiseen työasemalla tarvitaan Xampp tai palvelimella web-palvelin esimerkiksi Linux Apache2. Sovelluskontti tarvitsee ensimmäiseksi jotain, jolla sivusto saadaan näkymään. Tähän projektiin valitsin base imageksi Apache Httpd:n (Apache http Server). Imaget kannattaa hankkia jostain tunnetusta säilytyspaikasta, jotta voidaan varmasti varmistaa niiden aitous ja niistä saadaan myös aina tuorein versio tai jos tarvitaan esimerkiksi jokin tietty versio. Erilaisia säilytyspaikkoja on monia esimerkiksi: Docker Hub, Google Container Registry, Amazon Elastic Container Registry ja Azure Container Registry. Tähän projektiin valitsin Docker Hubin, jota tullaan myöhemmässä vaiheessa hyödyntämään valmiin imagen säilömiseen.

Imagen rakentaminen alkaa Dockerfileksi (kuva 14) kutsutun tekstitiedoston kirjoittamisella. Windows ympäristössä sen voi kirjoittaa Notepad++ tai vaikka ihan Notepadilla. Linuxilla VIM-editorilla tai jollakin toisella tekstieditorilla. Dockerfile koostuu avain-arvopareista, joista avaimet kirjoitetaan isoilla kirjaimilla ja arvot kirjoitetaan pienillä kirjaimilla. Dockerfile tulee suunnitella huolella, koska siitä rakennetaan varsinaisia imageja. Dockerfile sisältää tietoja imagen pohjakuvasta ja sen käytettävästä versiosta. Imageen on hyvä myös laittaa merkintä, kuka sen on tehnyt tai kuka sitä ylläpitää. Dockerfileen määritellään myös portti, jota käytetään. Esimerkiksi Apache2 käyttää porttia 8080. Dockerfile tallennetaan päätteettömänä eli siihen ei tule mitään päätettä, kuten normaalisti olisi esim. .txt tai vaikka .html.



Kuva 14. Dockerfile.

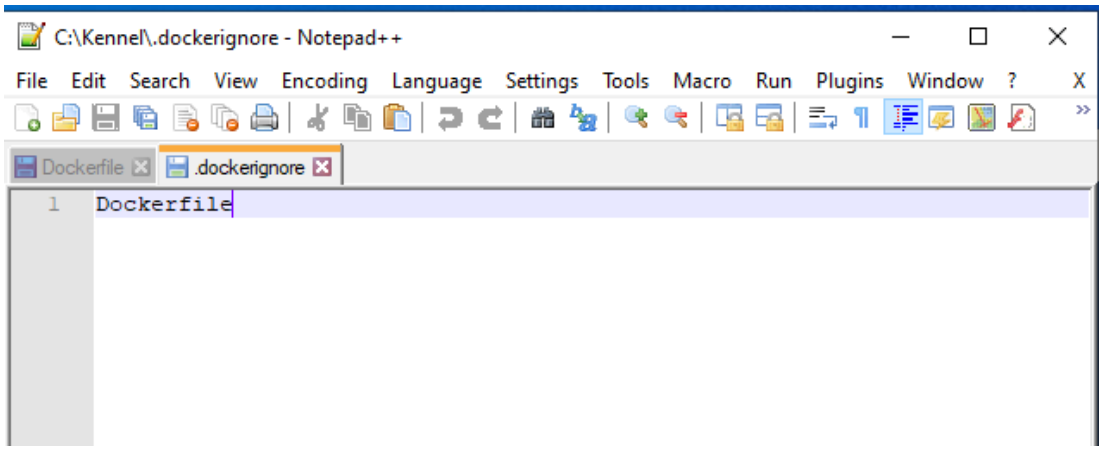
Otetaan listaus kansion tiedostoista (kuva 15) ja huomataan, että Dockerfile on siellä. Laitoin sen tarkoituksella sinne vaikkakin se kuuluisi juureen tallentaa.



Kuva 15. Kansion listaus.

Rakennettaessa imageja kaikki ylimääräiset tiedostot pyritään ottamaan pois sieltä, koska tarkoitus on saada niistä mahdollisimman pieniä kooltaan. Imagen rakennusvaiheessa Docker CLI tarkistaa, että löytyykö .dockerignore-niminen tiedosto ennen, kuin se lähettää sen Docker daemonille, joka rakentaa imagen. Tällä erityisellä tiedostolla voidaan poistaa kaikki ylimääräinen, jotta rakennettavasta imagesta saataisiin optimaalisen kokoinen ja pois kaikki sellaiset tiedot, jotka voivat pitää sisällään avaimia tai salasanoja. Imagen koolla on suoraan vaikutusta latausnopeuteen ja pienempi image vie vähemmän resursseja esimerkiksi palvelimella.

Aiemmin tallensin Dockerfilen kansioon ja seuraavaksi poistetaan se sieltä .dockerignore tiedostolla.



Kuva 16. dockerignore

Kuvassa 16 on lisätty poistettava tiedosto, joka tässä tapauksessa oli Dockerfile. Seuraavaksi rakennetaan image ja tarkistetaan, että Dockerfilea ei löydy projektin tiedostojen listauksesta. Imagen rakennus aloitetaan komennolla ”docker image build -t kennel .”, jolloin varsinainen rakentaminen käynnistyy. Imagen valmistuttua voidaan todentaa listaamalla imaget komennolla: docker images. Tällöin saadaan näkyviin kaikki imaget, jotka tässä tapauksessa ovat yksi httpd-image, joka toimi pohjakuvana äsken rakennetulle imagelle, joka nimettiin edellä kuvatussa komennossa kennel-nimiseksi (-t kennel). Jokaisella imagella on autenttinen Image ID, jota seuraavaksi käytetään käynnistettäessä kennel-image ja samalla tarkistetaan, että Dockerfilea ei löydy imagen tiedostoista, jotka sijaitsevat htdocs-kansiossa. Kirjoitetaan käsky ”docker run -it 61b88f5b0cdd sh”. Komento avaa shellissa listauksen kansioista. Seuraavaksi

annetaan käsky ”ls htdocs”, joka listaa kansion sisällön (Kuva 17) ja sieltä ei löydy Dockerfilea.

```
PS C:\Kennel> docker image build -t kennel .
Step 1/4 : FROM           httpd:latest
 ---> b2c2ab6dcf2e
Step 2/4 : LABEL          maintainer="ilkka.sirnio@student.samk.fi"
 ---> Using cache
 ---> 530692a6a1cc
Step 3/4 : COPY           . /usr/local/apache2/htdocs/
 ---> Using cache
 ---> 7e83563b59fd
Step 4/4 : EXPOSE         8080
 ---> Using cache
 ---> 61b88f5b0cdd
Successfully built 61b88f5b0cdd
Successfully tagged kennel:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker ho
PS C:\Kennel> docker images
kennel              latest              61b88f5b0cdd        About an hour ago   183MB
httpd               latest              b2c2ab6dcf2e        2 days ago          166MB
PS C:\Kennel> docker run -it 61b88f5b0cdd sh
bin  build  cgi-bin  conf  error  htdocs  icons  include  logs  modules
# ls htdocs
about.html  contact.html  css  img  index.html  js  puppy.html
#
```

Kuva 17. Listaus imagesta.

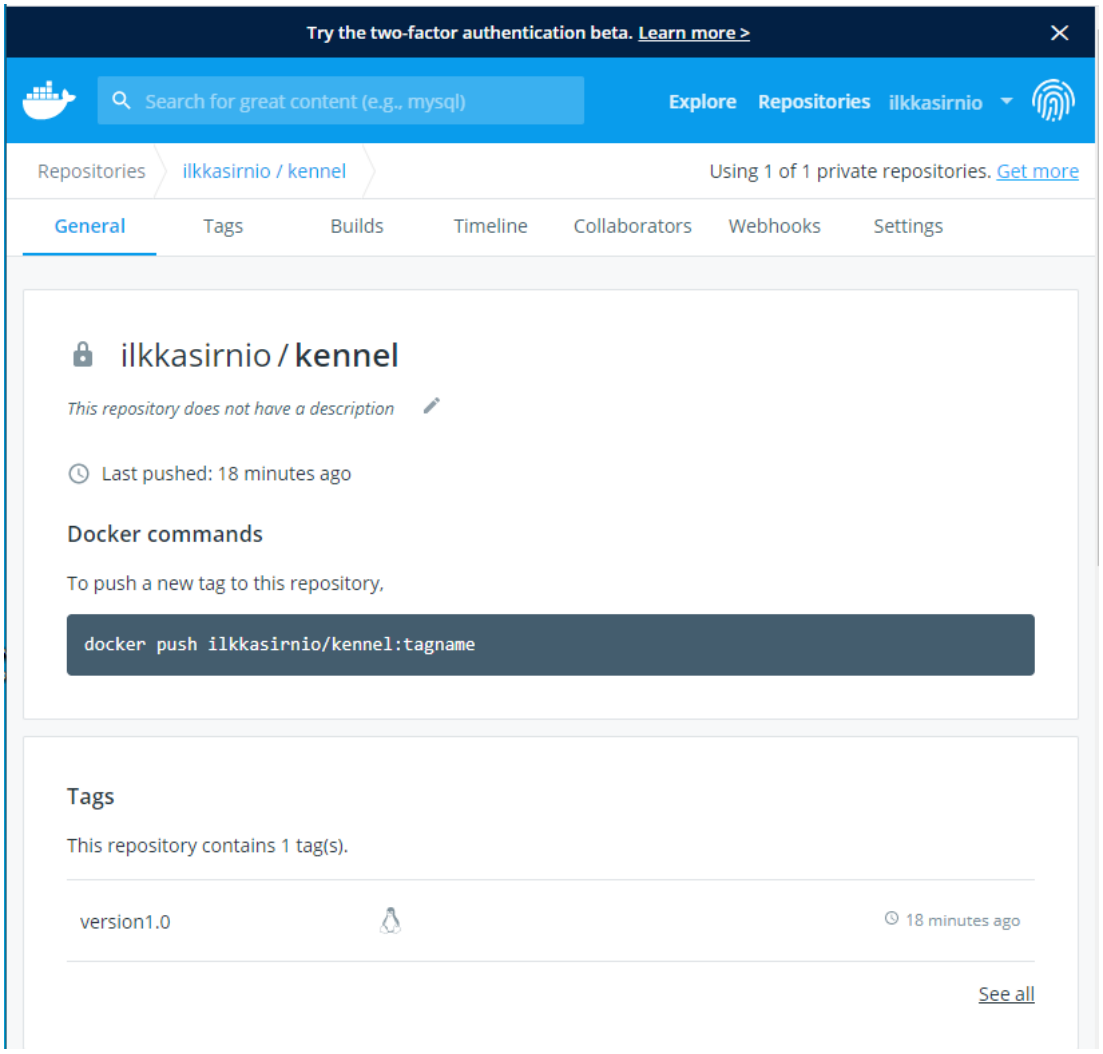
4.2 Docker Hub

Seuraavaksi rakennettu image siirretään Docker Hubiin. Oman säilytyspaikan luominen on nopeaa ja Docker Hubiin saa yhden ilmaisen yksityisen säilytyspaikan (repository). Luodun säilytyspaikan jälkeen itse siirto on nopea tehdä Powershellillä. Imagen puskeminen aloitetaan kirjautumalla komentorivillä. Sen jälkeen otetaan listaus imageista ja merkitään image komennolla ”docker tag 61b88f5b0cdd ilkkasirnio/kennel:version1.0”, jossa on autenttinen Image ID ja säilytyspaikan nimi ja kaksoispisteellä tagname. Sen jälkeen itse puskeminen tehdään komennolla: ”docker push ilkkasirnio/kennel:version1.0”. Kuvassa 18 image on merkitty ja puskettu Docker Hubiin.

```
PS C:\Kennel> docker login --username=ilkkasirnio
Password:
Login Succeeded
PS C:\Kennel> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
kennel              latest              61b88f5b0cdd        2 hours ago         183MB
httpd               latest              b2c2ab6dcf2e        2 days ago          166MB
PS C:\Kennel> docker tag 61b88f5b0cdd ilkkasirnio/kennel:version1.0
PS C:\Kennel> docker push ilkkasirnio/kennel:version1.0
The push refers to repository [docker.io/ilkkasirnio/kennel]
68160d6f1257: Pushed
35ca97a06fb3: Pushed
701ef2ccb5d3: Pushed
81b4f0dc1e64: Pushed
3e944ab7641d: Pushed
c2adabaecedb: Pushed
version1.0: digest: sha256:2f33a361674dda15c15a0c9accaf89c9aec283560ced78f8661e6163ab80f9c9 size: 1579
PS C:\Kennel>
```

Kuva 18. Image merkitty ja puskettu Docker Hubiin.

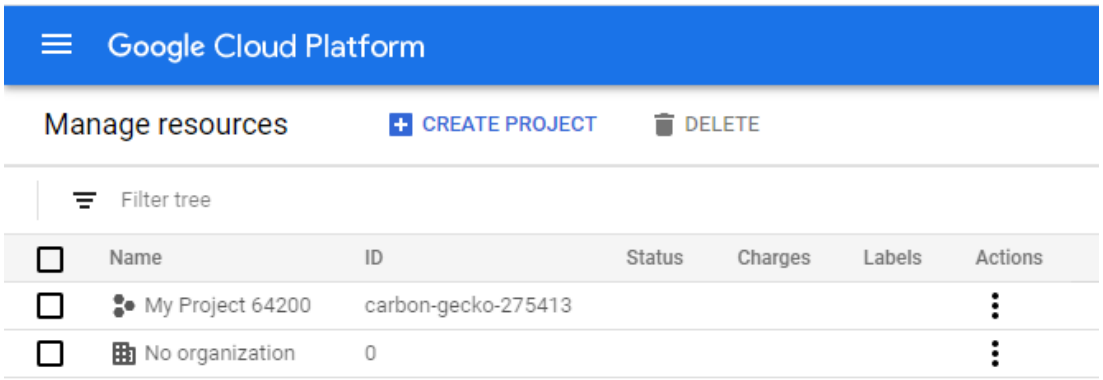
Tarkistetaan vielä kirjautumalla selaimella Docker Hubiin, että image on siellä (kuva 19).



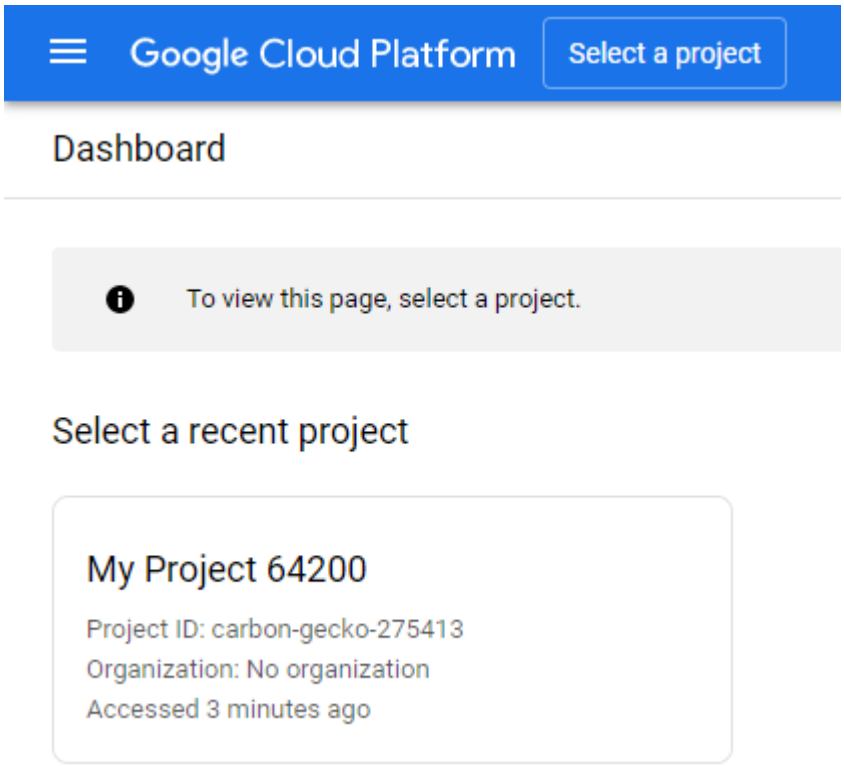
Kuva 19. Docker Hub.

4.3 Google Cloud

Seuraavaksi tarkoitus on siirtää image Docker Hubista Googlen pilvialustalle Google Cloudiin. Aluksi luodaan tunnukset tai kirjaudutaan sisälle ja luodaan projekti. Projektin loppuosuus on tarkoitus suorittaa Google Cloud Shellillä. Cloud shell on ilmainen ja ei vaadi mitään muuta, kuin tunnusten luomisen Google Cloudiin. Kuvassa 20 projekti on luotu.



Kuva 20. Projekti luotu.

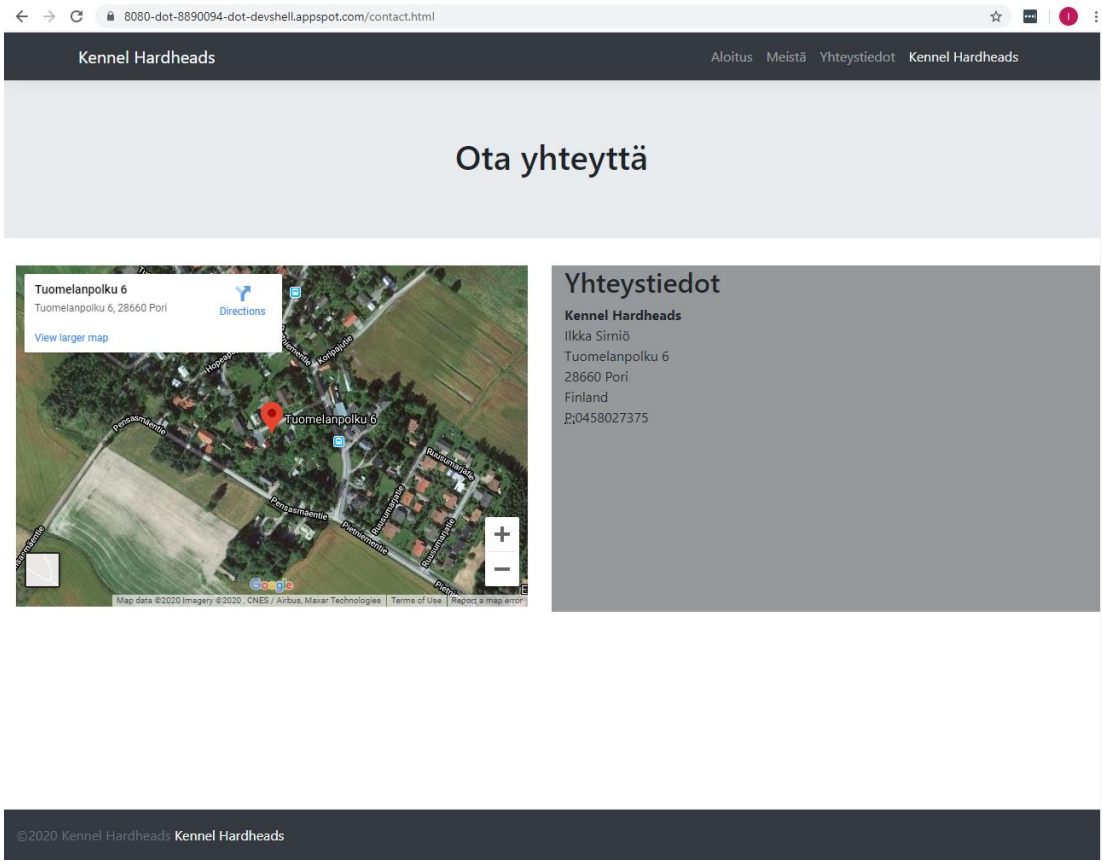


Kuva 21. Valittava projekti.

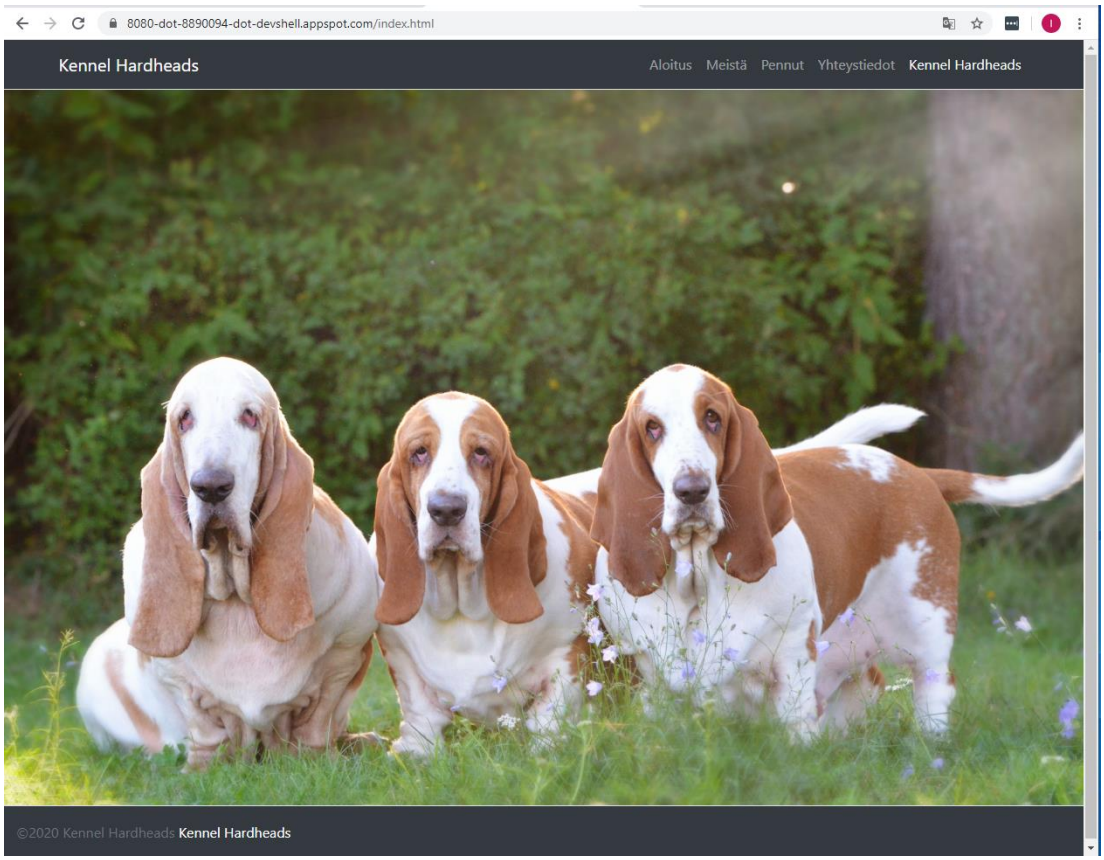
Valitun projektin jälkeen (kuva 21) voidaan avata Cloud Shell ja valita projekti. Cloud Shell on debian-pohjainen, jolloin komennoissa täytyy käyttää sudo-komentoa edessä Docker komennoissa, koska debian on Linux-pohjainen.

Aluksi täytyy kirjautua sisälle Dockeriin komennolla ”sudo docker login”. Onnistuneen kirjautumisen jälkeen voidaan alkaa siirtämään projektin tiedostoja Docker Hubin säilytyspaikasta komennoilla ”Sudo docker pull ilkkasirnio/kennel:version10”.

Latauksen jälkeen voidaan ottaa listaus imageista komennolla ”sudo docker ls”. Huomataan, että image löytyy ja se voidaa käynnistää sen komennoilla ”sudo docker run -d -p 8080:80 61b88f5b0cdd”, jossa -d tarkoittaa, että kontin käynnistyttyä komentorivi jää, eikä katoa. -p 8080:80 tarkoittaa portteja host 8080 ja guest 80. Numerosarja porttien perässä on Image ID. Komennolla ”sudo docker ps” nähdään kontit, jotka ovat käynnissä. Listauksessa näkyy portit, joita kontti käyttää. Avattaessa Cloud Shellistä web Preview, josta valitaan portti 8080, niin saadaan kuva (kuva 22 ja kuva 23) kontin sisällöstä.

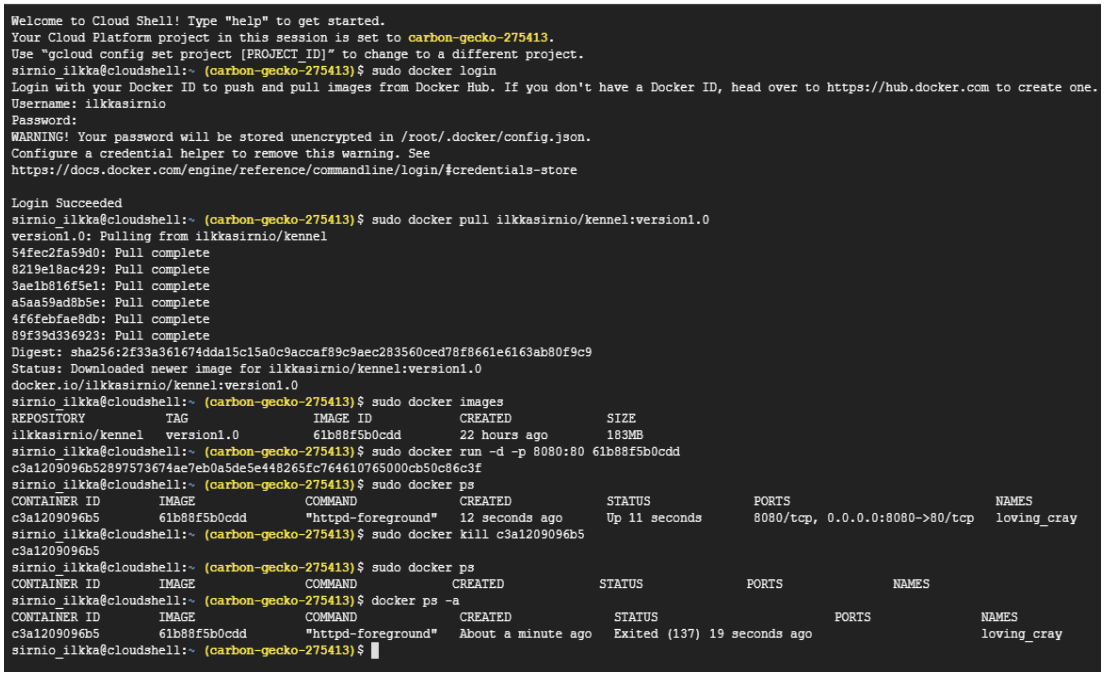


Kuva 22. Projektin sivu Web Preview.



Kuva 23. Web Preview.

Sovelluskontti saadaan sammutettua komennolla ”sudo docker kill c3a1209096b5”, jossa viimeinen numerosarja on Container ID, eikä Image ID. Lopuksi voi vielä varmistaa, että kontti on sammunut komennolla ”sudo docker ps -a” ja huomataan kontin sammutetun. Kuvassa 24 on käytetyt komennot eri vaiheista.



Kuva 24. Komennot eri vaiheista.

5 OMAA POHDINTAA

Docker on mullistanut tavan, jolla sovelluksia kehitetään ja otetaan käyttöön. Googlen tapaiset yritykset ovat vuosia sitten siirtyneet jatkuvaan julkaisuun, mikä nykyään on mahdollista konttitekniikan avuilla kaikenkokoisilla yrityksillä.

Kontit ovat virtualisoinnin tapainen pysyvä ilmiö, jonka mahdollisuuksia ei ole edes vielä nähty. Täysin virtuaalikoneita niillä ei pystyä korvaamaan, mutta se ei ole tarkoituskaan.

Opinnäytetyö oli projektina mielenkiintoinen toteuttaa, koska opittavaa ja haasteita riitti aivan loppumetreille asti. Projekti muuttui koko tekovaiheen ajan ja välillä oli ongelmia löytää aikaa jatkaa projektia.

Aiheena Docker on kiehtova ja aloittaessani tekemään tätä opinnäytetyötä, niin minulla oli käsitys Dockerista teoriatasolla, joka helpotti toteuttamista huomattavasti.

HTML ja CSS ovat entuudestaan tuttuja ja Bootstrap-kirjastoa olen käyttänyt jonkun verran. Kaikki käyttämäni tekniikat koen mieluisiksi ja tulen käyttämään niitä tulevissa projekteissa ja siksi Docker aiheena oli mielestäni oikea valinta.

LÄHTEET

Bootstrap, Viitattu 12.4.2020. Saatavissa: https://getbootstrap.com/

Bootstrap, Viitattu 12.4.2020. Saatavissa: https://get-bootstrap.com/docs/4.4/about/overview/

Cleverism. Docker. Viitattu 7.3.2019. Saatavissa: https://www.cleverism.com/company/docker/

From the garage to the Googleplex. Viitattu 17.5.2020. Saatavissa: https://about.google/our-story/

Google App Engine Blog. Viitattu 17.5.2020. Saatavissa: http://googleappengine.blogspot.com/2008/04/introducing-google-app-engine-our-new.html

Google Cloud. Viitattu 28.4.2020. Saatavissa: https://cloud.google.com/shell/docs/how-cloud-shell-works?hl=fi

HTLM History. Viitattu 16.5.2020. Saatavissa: https://www.w3schools.in/html-tutorial/history/

Hykes S. 2018. Au Revoir. Viitattu 7.3.2019. Saatavissa: https://blog.docker.com/2018/03/au-revoir/

Kotilainen S. 2017. Koodi sujahtaa konttiin – sovellusten kehittäminen mullistuu. Viitattu 16.3.2019. Saatavissa: https://www.tivi.fi/Kaikki_uutiset/koodi-sujahtaa-konttiin-sovellusten-kehittaminen-mullistuu-6674085

Marquez E. 2017. The History of Container Technology. Viitattu 7.3.2019. Saatavissa: https://linuxacademy.com/blog/containers/history-of-container-technology/

Red Hat 2013. Press Release. Viitattu 7.3.2019. Saatavissa: https://www.redhat.com/en/about/press-releases/red-hat-and-dotcloud-collaborate-on-docker-to-bring-next-generation-linux-container-enhancements-to-openshift

W3schools, CSS Tutorial, Viitattu 12.4.2020. Saatavissa: https://www.w3schools.com/css/

W3schools, HTML Tutorial, Viitattu 12.4.2020. Saatavissa: https://www.w3schools.com/html/html_intro.asp

KUVALÄHTEET

Call Me Moby Dock. Viitattu 16.3.2017. Saatavissa: https://blog.docker.com/2013/10/call-me-moby-dock/

Containers. Viitattu 17.3.2019. Saatavissa: https://www.docker.com/resources/what-container

Docker Architecture. Viitattu 19.3.2019. Saatavissa: https://www.aquasec.com/wiki/display/containers/Docker+Architecture

Docker Engine. Viitattu 17.3.2019. Saatavissa: https://docs.docker.com/engine/docker-overview/#docker-architecture

Go4it2day Web Services. 2017. Viitattu 7.3.2019. Saatavissa: https://www.go4it2day.com/news/chroot-jekyll-users.html

Virtual Machines.Viitattu. 17.3.2019. Saatavissa: https://www.docker.com/resources/what-container