jamk

Sähköpostin seurannan toteutus ja toiminta

Jonne Lustila

Opinnäytetyö, AMK
Joulukuu 2023
Insinööri (AMK), Tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Lustila, Jonne

Sähköpostin seurannan toteutus ja toiminta

Jyväskylä: Jyväskylän ammattikorkeakoulu. Joulukuu 2023, 29 sivua.

Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Opinnäytetyön tavoitteena oli luoda palvelu, jonka avulla on mahdollista selvittää sähköpostiviestin avaamisaika. Motivaationa työn tekemiselle toimi kiinnostus aihetta kohtaan sekä mahdollisuus lisätä lukijan tietoisuutta sähköpostin seurannan teknisestä toteutuksesta. Työ pyrkii myös parantamaan lukijan ymmärrystä keinoista, joilla yritykset seuraavat kuluttajien liikkeitä ja kohdentavat markkinointia mahdollisiin asiakkaisiin.

Toteutetun palvelun tuottamiseen käytettiin Pythonia. Python-ohjelmista rakennettiin kontit Dockeriin, jonne myös luotiin MariaDB-tietokanta. Palvelut saatiin toimimaan keskenään käyttäen Docker Composea ja reititys ulkoverkosta toteutettiin käänteisellä Nginx-välityspalvelimella. Palvelut luotiin virtuaalipalvelimelle, joka tehtiin Oraclen Oracle Cloud Infrastructure -palveluun.

Työn tuloksena syntyi toimiva palvelu, jonka avulla sähköpostin avaamisajan selvittäminen on mahdollista. Palvelulla on mahdollista seurata usean sähköpostin avaamisaikoja. Palvelu onnistuttiin tuottamaan käyttäen ilmaisia resursseja, jonka tarkoituksena on mahdollistaa vastaavien toteutuksien tekeminen ilman kustannuksia.

Avainsanat (asiasanat)

Sähköposti, Docker, pilvipalvelut

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Lustila, Jonne

Implementation and functionality of email tracking

Jyväskylä: JAMK University of Applied Sciences, December 2023, 29 pages.

Degree Programme in Information and communication Technology. Bachelor´s thesis.

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

The aim of the thesis was to create a service that allows determining the opening time of an email. The motivation for the work was the interest in the subject and the opportunity to increase the reader's awareness of the technical implementation of email tracking. The work also seeks to improve the reader's understanding of the methods companies use to track consumer behavior and target marketing to potential customers.

Python was used to implement the service. Python programs were containerized into Docker, where a MariaDB database was also created. The services were made to work together using Docker Compose, and external network routing was implemented with Nginx reverse proxy server. The services were deployed on a virtual server created on Oracle's Oracle Cloud Infrastructure service.

As a result of the work, a functional service was created that enables determining the opening time of an email. The service allows tracking the opening times of multiple emails. The service was successfully produced using free resources, with the aim of enabling similar implementations to be made without costs.

Keywords/tags (subjects)

Email, Docker, cloud services

Miscellaneous (Confidential information)

Sisältö

Kuviot

1 Johdanto

1.1 Yleistä

Nykyään internetissä ei tapahdu mitään ilman, että käyttäjän liikkeitä seurataan ja analysoidaan. CRM-ohjelmistojen tavoitteena onkin kerätä ja hyödyntää tätä tietoa. Sähköpostikampanjoiden ja mainostamisen osalta tämä tarkoittaa, että CRM kerää mahdollisimman paljon dataa käyttäjästä viestin lukutapahtuman yhteydessä. Yritystä kiinnostaa esimerkiksi klikkaako käyttäjä viestin linkkejä, kuinka nopeasti vastaanottamisen jälkeen viesti avataan, millä laitteella käyttäjä lukee viestin ja mihin kellonaikoihin käyttäjä on aktiivinen. Näistä syistä on tärkeää, että jokainen ymmärtää joitain perusperiaatteita ja toimintamalleja, joilla yritykset tätä dataa keräävät. Opinnäytetyössä toteutettiin palvelu, jonka avulla on mahdollista seurata, koska sähköpostiviesti on avattu.

Motivaatio työn tekemiselle oli ottaa käyttöön sähköpostin avausajan seurannan mahdollistava palvelu, eikä verkosta löytynyt omaan käyttöön sopivaa toteutusta. Palvelun haluttiin olevan mahdollisimman avoimesti toteutettava ja tukevan ohjelmointirajapintaa, ettei käyttö olisi rajoitettu valmiiksi tehtyyn kankeaan käyttöliittymään. Verkosta löytyvät ilmaiset palvelut eivät vastanneet vaatimuksia, joten palvelu päätettiin tuottaa itse.

1.2 Tavoite

Opinnäytetyön tavoitteena oli tuottaa palvelu käyttäen nykyaikaisia menetelmiä ja järjestelmiä ilmaiseksi saatavilla olevin resurssein. Tavoitteena oli tuottaa palvelu käyttäen julkiseen pilveen luotua virtuaalikonetta, jossa tuotettua palvelua ylläpidetään Docker-virtualisointialustalla. Palvelu ohjelmoitiin käyttäen Pythonia ja tietokantana käytettiin MariaDB:tä.

Tuotetun palvelun tavoitteena oli mahdollistaa lähetetyn sähköpostiviestin avausajan selvittäminen muiden oleellisten tietojen ohessa, jotka ovat seurantaan käytettävällä menetelmällä mahdollista kerätä. Ohjelmisto tuotettiin yksityiskäyttöä silmällä pitäen, joten palvelun ei tarvitse kestää raskaita kuormia. Palvelut haluttiin kuitenkin toteuttaa helposti skaalattavalla tavalla, että myös suurempia kuormia olisi tulevaisuudessa mahdollista tukea skaalaamalla toteutusta horisontaalisesti.

1.3 Tutkimusasetelma

Tutkimuksessa käytetään kehittämistutkimuksen menetelmiä. Kanasen (2015) mukaan kehittämistutkimus ei itsessään ole erillinen tutkimusmenetelmä, vaan joukko tutkimusmenetelmiä, joita käytetään riippuen tutkimuksen kohteesta. Kehittämistutkimus pyrkii muuttamaan tutkimuksen kohteena olevaa asiaa ja siksi kehittämistutkimuksen tyyli mielestäni sopii tähän työhön. Tutkimuksessa keskitytään luomaan uutta teoriaa samalla hyödyntäen vanhaa. (Kananen 2015.)

Tutkimuksessa edetään tavoitteiden määrittämisestä ja tutkimuskysymysten luomisesta kohti kehityksen kohteena olevan tuotteen valmistumista. Olennaisessa osassa on aiemmin kerätty tieto, johon viittaamalla sitä käytetään hyödyksi. Lähteinä käytetään verkko- ja kirjalähteitä, joissa on mahdollisimman ajankohtaista tietoa. Lähteiden sisältöä tarkastellessa pyritään olemaan kriittinen ja ottamaan huomioon julkaisupaikka.

Opinnäytetyö rajattiin käsittelemään toteutetun palvelun suunnittelua, testausta ja toteutusta. Palvelun toiminnallisuudet rajattiin perustoimintoihin, joita edistynyt yksityiskäyttäjä voisi tarvita käyttäessään palvelua. Palvelulle ei myöskään toteuteta käyttöliittymää osana tutkimusta. Tutkimuksen tarkoituksena on selvittää, kuinka helppoa on luoda ilmaisia resursseja käyttäen sähköpostin seurantapalvelu ja millaisen toteutuksen se vaati. Seuraavien tutkimuskysymysten avulla voidaan luoda palvelu ja tarkastella onko se verrattavissa kaupallisesti saatavilla oleviin ratkaisuihin:

1. Miten saada selville sähköpostin avaamisen ajankohta?
2. Kuinka hallitaan eri henkilöille lähetettyjen viestien seurantaa?
3. Miten palvelu eroaa kaupallisista ratkaisuista?

2 Ympäristö ja teknologiat

2.1 Oracle Cloud Infrastructure

Oracle Cloud Infrastructure on Oraclen pilvipalvelu. Oracle tarjoaa palvelussaan aina ilmaisia resursseja ja palveluita, joiden avulla pilveen voi tutustua. Ilmaisen käyttäjätunnuksen luodessaan käyttäjä saa käyttöönsä sekä ”Always Free”-tason palvelut, että 300 Yhdysvaltain dollaria, jotka voi käyttää palveluihin ja resursseihin kuukauden sisällä tunnuksen luomisesta. Aina ilmaisiin palveluihin sisältyy käyttörajoituksia, esim. verkon kuormantasaajia voi luoda yhden ja se tarjoaa enimmillään 10 Mbps läpisyötön. Oraclen verkosta ulospäin voi myös siirtää enintään 10 tera bittiä dataa kuukaudessa. (Oracle Cloud Free Tier n.d.)

Compute-palvelu on tärkeä pilvipalvelu, se tarjoaa mahdollisuuden luoda ja ottaa käyttöön virtuaali- ja fyysisiä instansseja. Ennen instanssin luomista käyttäjällä on oltava virtuaalinen verkko ja ssh-avaimet, pitää myös tietää missä fyysisessä sijainnissa haluaa resurssin sijaitsevan sekä palvelimen käyttöjärjestelmä, että muoto. Käyttöjärjestelmä kuvaa valittaessa vaihtoehtoina on muun muassa Oraclen tarjoamia levykuvia, yhteistyökumppaneiden levykuvia ja mahdollisuus käyttää omaa levykuvaa. Instanssin muoto määrittää prosessorin tyypin ja käytettävien ytimien määrän sekä keskusmuistin ja tallennustilan määrän, että verkotustietoja. (Png & Demanche 2020, 57-60.)

2.2 Nginx

Nginx on avoimen lähdekoodin ohjelma, joka mahdollistaa monia eri toimintoja. Yleisimmät käyttökohteet ovat yksinkertainen web-palvelin, välityspalvelin, sisällön tallentamisen välimuistiin nopeuttaakseen palvelun toimintaa ja sähköpostin välityspalvelin. Nginx:n ohjelmoi alun perin Igor Sysojev ratkaistakseen C10K-ongelman. (What Is NGINX? N.d.) Valappilin mukaan C10K-ongelmassa yritetään mahdollistaa kymmenen tuhannen yhtäaikaisen yhteyden ottaminen samaan palveluun ja haasteen keksi Dan Kegel vuonna 1999 lisääntyvän internetin käytön takia (Valappil 2022). Nginx:n tavoitteena on alusta asti ollut olla maailman nopein web-palvelin ja sitä se palvelun oman sivuston mukaan onkin (What Is NGINX? N.d).

Nginx:n asentaminen useimpiin järjestelmiin on helppoa, sillä käyttöjärjestelmien paketinhallinta ohjelmat sisältävät valmiiksi kootun paketin myös Nginx:lle. Ubuntu järjestelmään nginx asennetaan komennolla ”apt install nginx” (Installing NGINX Open Source n.d.). Nginx konfiguroidaan käyttäen paketin latauksen yhteydessä luotua tiedostorakennetta ja sen konfiguraatiotiedostoja. Kun Nginx:ää käytetään välityspalvelimena, tulee käyttää ”location”-moduulia, johon määritellään sijainti, josta liikenne ohjataan ja sijainti johon liikenteen tulisi seuraavaksi mennä.

2.3 Docker

Docker on konttiteknologia, joka tarjoaa muista konteista ja järjestelmistä eristetyn alustan sovelluksille (Wallenius 2022). Docker on etäisesti verrattavissa virtuaalipalvelimiin, jotka ovat aiemmin olleet suosittu ratkaisu IT-alalla. Virtuaalipalvelimet ja Docker eroavat toisistaan siinä, että kontit eivät tarvitse omaa käyttöjärjestelmää eikä kerneliä, vaan käyttävät yhteistä Linux kerneliä, joka on valmiiksi päällä, kun Docker on käynnistetty. Virtuaalipalvelimella on aina oma kerneli ja käyttöjärjestelmä, jotka kuluttavat enemmän isäntäkoneen resursseja ja tallennustilaa. Docker mahdollistaa todella nopeat käynnistysajat palveluille sekä paljon virtuaalikonetta pienemmän resurssien kulutuksen myös, koska levykuvissa ei yleensä ole turhia paketteja ja ne perustuvat kaikista pienimpiin käyttöjärjestelmiin. (Ghosh 2020, Chapter 1 Introduction to Containerization and Docker.) Myös Vohra (2016) kirjoittaa, että Docker nopeuttaa ja yksinkertaistaa sovelluskehitystä. Hän toteaakin, että Dockerfile, jota käytetään konttien määrittämiseen aiemmin määriteltyjen konttien päälle mahdollistaa tärkeiden parametrien määrittämisen selkeästi (Vohra 2016, Chapter 1: Hello Docker).

Kun Docker otetaan käyttöön, ohjelma tai konfiguraatiot kontin funktion mukaan siirretään konttiin käyttäen Dockerfilea (ks. Kuvio 1). Dockerin konfiguraatiotiedostossa määritellään, myös muun muassa avattavat portit, ympäristömuuttujat ja kontti, johon luotava kontti perustuu. Dockerfilen avulla voi myös asentaa ohjelmia ja paketteja, mikäli pohja kuva ei sisällä kaikkea tarvittavaa. Bhat (2021) myös suosittelee, että kontit eivät kirjoittaisi mitään kontin tiedostojärjestelmään, vaan kaikki data jonka on tarkoitus olla pysyvää tulisi kirjoittaa docker volumeille tai kontin ulkopuoleisiin tiedostojärjestelmiin. Myös kerrosten määrä tulisi hänen mukaansa minimoida. Kerroksia luo käskyt ”RUN”, ”COPY” ja ”ADD”. (Bhat 2021, Chapter 2: Docker 101) Kokemukseni mukaan minimointia voi suorittaa esimerkiksi pakettien asennusvaiheessa yhdistämällä monta erillistä komentoa yhdeksi käyttäen Linux-komentokehotteen syntaksia. Dockerfilen luomisen jälkeen levykuva tulee luoda ”docker build”-komennolla. Levykuvan luomisen jälkeen kontti voidaan luoda ”docker container run”-komennolla. (Stoneman 2022, Chapter 3 Building Your Own Docker Images)

```
FROM ubuntu:latest
LABEL author="sathyabhat"
LABEL description="An example Dockerfile"
RUN apt-get install python
COPY hello-world.py
CMD python hello-world.py
```

Kuvio 1. Esimerkki Dockerfilestä. (Bhat 2021, Chapter 2: Docker 101).

Kun sovellus pilkotaan useaan konttiin joista jokainen on tarpeellinen sovelluksen toiminnan kannalta koko applikaation kontit voi sisällyttää Docker Composen compose-tiedostoon. Siinä määritellään Dockerfileä vastaavalla tavalla kontteja eroina se, että samassa tiedostossa määritellään useita kontteja, ja niiden väliset yhteydet. Dockerfilen voi käsittää enemmänkin

sarjana ohjeita yhdelle kontille ja compose-tiedosto voi suurimmillaan olla määritystiedosto kokonaiselle ympäristölle. Kumpaakin kuitenkin tarvitaan, koska niillä on eri funktiot Dockerin ekosysteemissä. Bhat (2021) toteaakin Composen tavoitteena olevan yksinkertaistaa monimutkaisten sovellusten käynnistämistä. (Bhat 2021, Chapter 7: Understanding Docker Compose.) Dockerin avulla luotiin ja testattiin opinnäytetyössä tuotettuja palveluita.

Docker Compose on erillinen ohjelmisto, joka ladataan palvelimelle, jolle on jo asennettu Docker. Compose käyttää yaml-muotoista tiedostoformaattia, josta määritykset luetaan ja oletuksena tiedosto on nimeltään docker-compose.yaml, mikäli tämä halutaan yliajaa, tulee käyttää -f argumenttia. Docker-compose.yaml-tiedostossa määritellään palvelut käyttäen ”services:” lohkoa ja levyt käyttäen ”volumes:” lohkoa vastaavat lohkomäärityket on olemassa myös verkkojen määrittelyä varten. Kuvion esimerkissä määritellään kolme konttia ja yksi levy. Levy määritellään, ettei tiedot häviäisi kontin uudelleen luonnin yhteydessä. (Ks. Kuvio 2.) Opinnäytetyön palvelun käynnistämisessä ja luomisessa hyödynnetään Docker Composea. Sillä on myös mahdollista sammuttaa ja uudelleen käynnistää palvelu.

```
services:
    database:
      image: mysql
      environment:
        MYSQL_ROOT_PASSWORD: dontusethisinprod
      volumes:
        - db-data:/var/lib/mysql
    webserver:
      image: 'nginx:alpine'
      ports:
        - 8080:80
      depends_on:
        - cache
        - database
    cache:
      image: redis

volumes:
    db-data:
```

Kuvio 2. Esimerkki Docker Compose -tiedostosta (Bhat 2021, Chapter 7 - Understanding Docker Compose).

2.4 Python

Python on aloittelijaystävällinen ohjelmointikieli, joka on helppo oppia. Se on myös lähdekoodiltaan avoin, joka tarkoittaa, että se on kaikkien saatavilla ja muokattavissa. Pythonin koodi on lisensoitu OSI-lisenssiä käyttäen ja sitä hallinnoi Python Software Foundation. Python sovelluksia kehitetään kolmansien osapuolten kehittämiä moduuleja hyväksi käyttäen. Nämä

moduulit ovat saatavilla Python Package Indexistä. Ulkoiset moduulit mahdollistavat nopeamman sovelluskehityksen. (About Python n.d.)

Pythonin on kehittänyt Guido Van Rossum vuonna 1991 ABC-ohjelmointikielen pohjalta. Pythonin suosio on noussut 2010-luvulla, koska Pythonin kolmas täysversio julkaistiin 2008. Python 3 ja Python 2 eivät ole täysin yhteensopivia keskenään, vaikkakin erot näiden välillä ovat ohjelmoijan näkökulmasta melko pieniä. Peltomäen (2023) mukaan Pythonin tärkeimpiä ominaisuuksia ovat; hyvin jäsennelty ja helppolukuinen koodi, koodin siirrettävyys, ohjelman suoritettavuus Python-tulkilla ja monipuoliset kirjastot, joilla on mahdollista laajentaa ohjelmaa. Koodia voidaan kirjoittaa ja ajaa kahdella eri tavalla. Ensimmäinen ja suoraviivaisempi tapa on käyttää Python-konsolia, jossa komento ajetaan aina heti kun se on syötetty kehotteeseen. Yleensä koodi kuitenkin kirjoitetaan tiedostoon käyttäen IDE-sovelluskehitintä. Sovelluskehittimet sisältävät hyödyllisiä työkaluja koodin muotoilua, kirjoittamista ja testausta ajatellen. (Mts. 6-8.)

2.5 Flask

Flask on python moduuli, joka mahdollistaa dynaamisten verkkosivujen luomisen käyttäen Jinja2-formaattia ja toimii verkkopalvelimena. Se on koodipohjaltaan pieni ohjelmistokehys, mutta se ei kuitenkaan tarkoita, että se olisi huonompi kuin muut verkkopalvelinmoduulit. Flaskin on kehittänyt Armin Ronacher. (Grinberg 2018, 11) Flask tekee verkkopalvelimen luomisesta helppoa juuri aiemmin mainitun Jinja2 formaatin ja helposti luotavien funktioiden takia. Se myös madaltaa kynnystä tehdä verkkopalvelin käyttäen Pythonia, koska se ei sisällä yhtä paljoa pakollista pohjakoodia, kuin muut vaihtoehtoiset moduulit.

Flask applikaatio luodaan koodissa tuomalla Flask-moduuli käyttäen import komentoa. Tämän jälkeen määritellään olio luokasta ”Flask”, joka vaatii vain yhden argumentin. Argumentti määrittää olion käyttämän päämoduulin tai paketin ja yksinkertaisissa tapauksissa ”__name__” riittää. Flask käyttää tätä tietoa määrittääkseen luotavan sovelluksen mallien ja muiden tiedostojen sijainnin. Flaskissa reitit ja URL:t määritetään koristefunktioiden avulla. Koristefunktioita käytetään yleisesti Pythonissa rekisteröimään tapahtumia, jotka edelleen käynnistävät käsittelijäfunktion.

2.6 MariaDB

MariaDB syntyi yrityskauppojen seurauksena, kun Sun Microsystems hankki MySQL:n vuonna 2008, jonka jälkeen Oracle hankki Sun Microsystemsin vuonna 2010 ja kauppaan sisältyi MySQL. MariaDB:n loi MySQL:n perustaja Michael Widenius ”forkkaamalla” eli kopioimalla lähdekoodin MySQL:stä. Estääkseen MySQL:n kohtalon Widenius perusti MariaDB foundation-yhdistyksen vuonna 2012, jonka tarkoituksena on tukea MariaDB:n jatkuvuutta sekä tarjota yhteyspiste avoimelle yhteistyölle eri tahojen välillä. (What is MariaDB n.d.) MariaDB:n oman verkkosivuston mukaan Oraclen hankinta tapahtui vuotta aiemmin 2009. Sivustolla myös sanotaan, että

kummatkin tietokannan hallinta järjestelmät ovat nimetty Wideniuksen lasten mukaan, he ovat My ja Maria. (MariaDB in brief n.d.)

Petersonin (2023) vertailun mukaan MariaDB on nopeampi kuin sitä vanhempi MySQL. MySQL:ssä on myös suljetun lähteen koodia yrityksille tarkoitetussa Enterprise-versiossa. Näistä syistä moni suosiikin MariaDB:tä, vaikka MySQL tarjoaa mahdollisuuden dynaamisiin sarakkeisiin ja datan maskeeraukseen. MySQL ja MariaDB käyttävät samaa syntaksia ja ovat muutenkin hyvin samanlaiset, koska MariaDB on kehitetty perustuen MySQL:ään. Kumpikin on käytössä pienistä harrasteprojekteista suurten yritysten tuotantoympäristöihin, joten suurta merkitystä näiden välillä normaalissa käytössä ei ole.

3 Sähköpostin seuranta

Yritykset markkinoivat aktiivisesti sähköpostin välityksellä, koska sillä saavutetaan helposti laaja kuluttajaryhmä, jotka ovat joko osoittaneet kiinnostuksensa yritystä ja sen tuotteita kohtaan tai ovat jo asiakkaita. CRM-ohjelmistojen avulla voidaan myös jakaa käyttäjiä erilaisiin ryhmiin, joille räätälöidään omanlaisensa viestit esim. aiemmin katsottujen tai tilattujen tuotteiden perusteella. Viesteissä käytetään erilaisia seurannan keinoja, kuten seurantapikseliä ja seurantalinkkejä. Näiden avulla kerätään dataa käyttäjän interaktioista markkinointiviestin sekä yrityksen sivuston tai vaikka verkkokaupan kanssa. Tämä data on tärkeää yritykselle, koska sen avulla yritys voi parantaa markkinointiaan, asiakasviestintäänsä ja mahdollisesti myös tuotteitaan.

Gordonin (2023) mukaan käyttäjänhallintaohjelmistot auttavat yritystä ymmärtämään asiakkaan tarpeita paremmin ja hallitsemaan asiakassuhteita eivätkä siten ole pelkkä tapa hallita asiakkaiden yhteystietoja. Hänen mielestään tärkeimpiä ominaisuuksia ovat markkinoinnin automatisointi ja yhteystietojen hallinta, jotka kytkeytyvät tiiviisti sähköpostin seurantaan.

On olemassa myös kuluttajille suunnattuja sähköpostin seuranta ohjelmistoja, jotka toimivat usein esim. Gmailissa selainlisäosan avulla. Esimerkkejä tällaisista palveluista ovat Boomerang ja Mailtrack. Kummatkin tarjoavat mahdollisuuden lähettää käyttäjälle ilmoituksen, kun vastaanottaja avaa viestin ja kun viestin sisällön mikä tahansa linkki avataan. Kummassakin palvelussa on ilmainen versio saatavilla ja lisäksi Mailtrackilla on myös kaksi eri tason kuukausimaksullista tilausvaihtoehtoa. Tilausvaihtoehtojen välillä on raportoinnin laajuuden suhteen eroja, jotka saattavat houkutella käyttäjiä valitsemaan kalliimman vaihtoehdon (ks. kuvio 3).

REPORTING	FREE	PRO	ADVANCED
Daily Report	✓	✓	✓
Activity Dashboard	Only first open	Full metrics	Full metrics
Email tracking history	Only first open	Full metrics	Full metrics
Email Productivity	✓	✓	✓
Export data to CSV	×	×	✓
Email Delivery Certificate (PDF)	×	×	✓

Kuvio 3. Mailttrack-ohjelmiston tilausvaihtoehtojen eroja (Choose the plan that best fits your needs n.d).

4 Tietoturva

Tietoturvalla tarkoitetaan yrityksen omien tietojen suojaamista ja sen tavoitteena on turvata yrityksen oma toiminta, kun taas tietosuoja on henkilötietojen suojaamista. Henkilötietojen suojaamisen kohteena on ihminen ja tavoitteena on turvata henkilön yksityisyyttä. Nämä eroavat toisistaan myös periaatteellisista näkökulmista. Omia tietojaan yritys saa käsitellä vapaasti ja päättää miten niitä käytetään, mutta henkilötietojen käsittely on oletuksena kiellettyä ja sallitaan vain perustelluista syistä. Syitä henkilötietojen käsittelyyn ei kuitenkaan ole vaikea löytää. Yleinen syy henkilötietojen keräämiseen ja käyttämiseen on asiakassuhde, joka velvoittaa yrityksen keräämään ja ylläpitämään ajantasaisia toiminnan kannalta tarpeellisia tietoja. Vuodesta 2018 asti kaikki EU-maat ovat seuranneet henkilötietojen osalta samoja lakeja, jotka määrittävät EU:n tietosuoja-asetus GDPR. (Järvinen 2022, 26.)

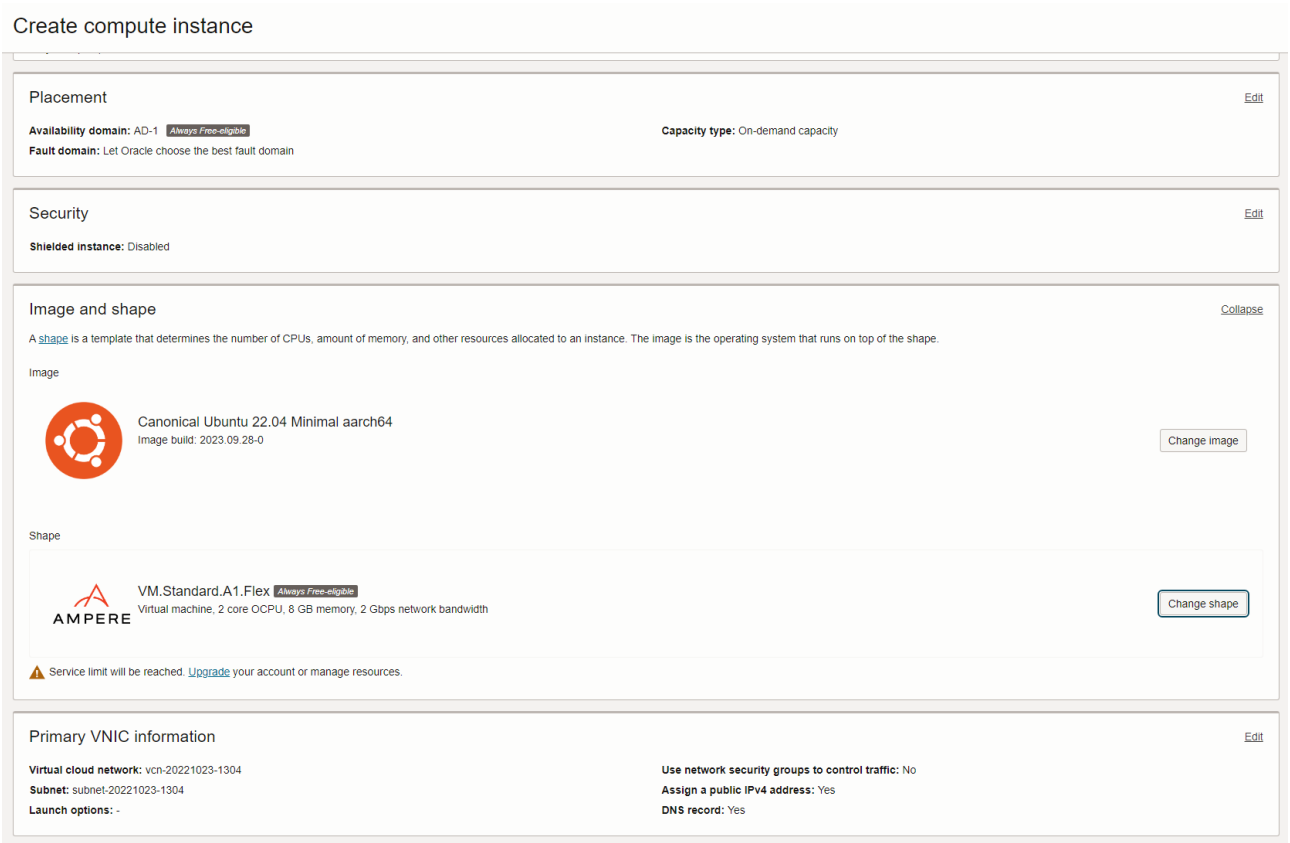
Järvisen (2022) mukaan päivitykset ovat modernin maailman pahin vitsaus. Päivitykset työllistävät käyttäjää ja ylläpitäjää sekä usein keskeyttävät muun toiminnan päivityksen ajaksi. Ne kuitenkin korjaavat palveluissa olevia tietoturva-aukkoja ja etenkin kriittisiksi merkityt päivitykset tulisi asentaa mahdollisimman pikaisesti. Tavallisetkin päivitykset tulisi hänen mukaansa asentaa aina heti erityisesti julkiseen verkkoon auki olevien palvelinten ja palveluiden osalta. Tästä syystä olisikin hyvä ottaa automaattiset päivitykset käyttöön. Käyttöjärjestelmä päivitykset tulisi suorittaa vain, kun niihin on varattu riittävästi aikaa, koska niissä saattaa kestää pidempään.

Verkkoon kytketyissä laitteissa oman kokemuksen mukaan on tärkeää, että pääsyä hallitaan monella eri tasolla ja sallitaan pääsy pelkästään tarvittaviin järjestelmiin tai järjestelmän osiin. Esimerkiksi verkkosivustolla tämä voi tarkoittaa sitä, että tietyt sivustot näkyvät vain jos käyttäjä on kirjautunut sisään ja omaa tietyt oikeudet. Verkko-tasolla liikennettä voidaan sallia ja estää perustuen porttimäärityksiin, asiakkaan IP-osoitteeseen tai esimerkiksi fyysiseen sijaintiin. Näitä keinoja yhdistelemällä on mahdollista rakentaa tietoturvallisia järjestelmiä, joiden väärinkäyttö on mahdollisimman vaikeaa.

5 Seurantapalvelun tekeminen

5.1 Järjestelmien pystytys ja ensimmäiset testit

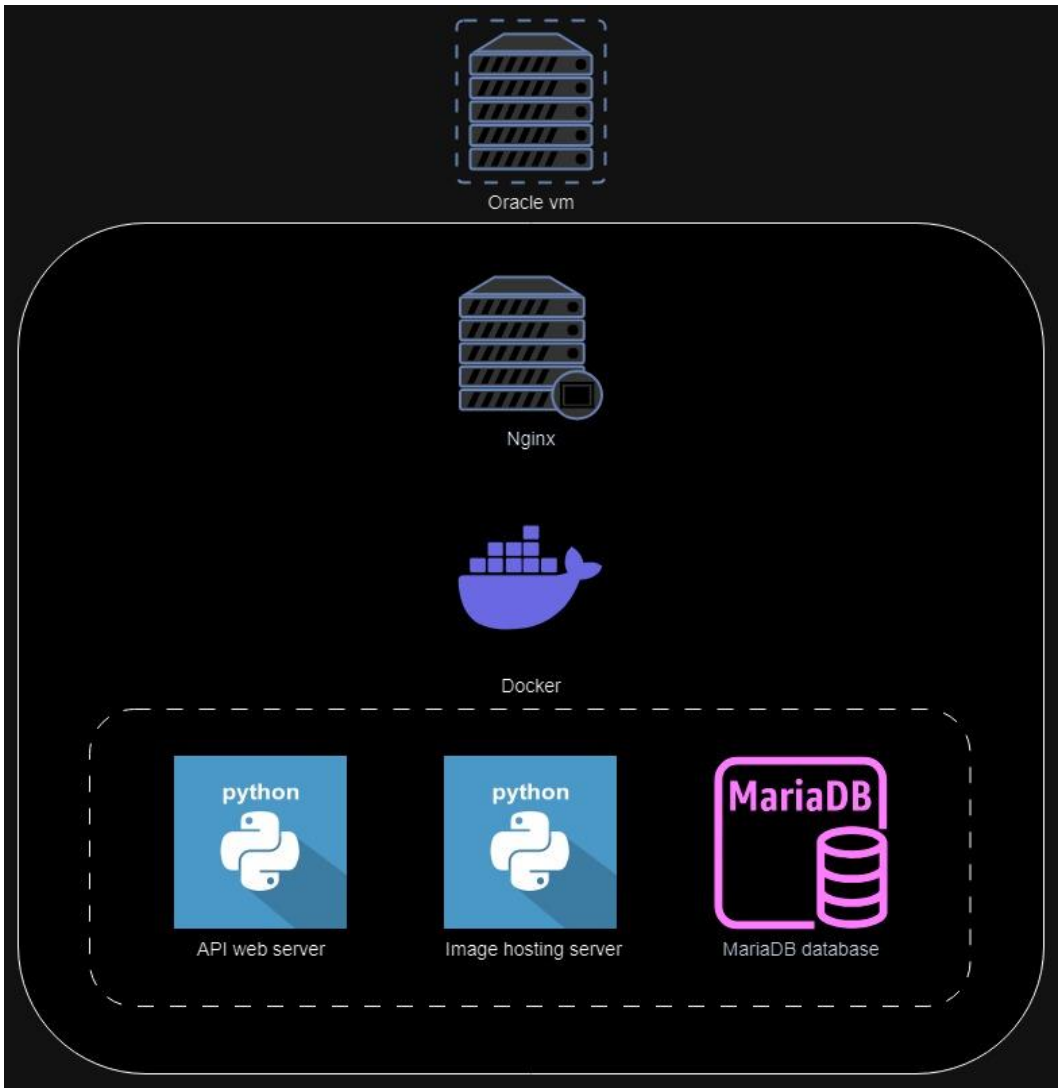
Suunnitteluvaiheen alkaessa pohdittiin, olisiko palvelu mahdollista tuottaa ilman erillistä palvelinta, mutta se todettiin vaikeaksi toteuttaa halutun ajan ja resurssien puitteissa. Pilvipalveluun päädyttiin punnitsemalla kotiverkosta ylläpidettävän palvelimen hyötyjä ja haittoja. Suurimmiksi haitoiksi havaittiin porttien avaaminen kotiverkkoon, palvelimen ylläpitämisestä nousevat kustannukset sähkön kulutuksessa sekä dynaaminen IP-osoite. Ylläpitämällä palvelua pilvessä kaikilta näiltä ongelmilta voidaan välttyä, kun kyseessä on pienen skaalan toteutus, joka mahtuu Oraclen Always Free-tason rajojen sisälle. Virtuaalipalvelimelle saa staattisen IP-osoitteen ja kunhan instanssin koko on tarpeeksi pieni, se ei maksa mitään. Instanssin kokoa valittaessa päädyttiin testauksen myötä kahteen ARM-ytimeen ja 8GB keskusmuistiin. Tämä mahdollistaa muiden resurssien käyttämisen toisiin projekteihin tai esimerkiksi kahden eri version testauksen samaan aikaan ja on tarpeeksi tehokas hallitsemaan kohtalaisen käytön yksityiskäytössä. Testauksessa todettiin, että muodon VM.Standard.E2.1.Micro virtuaalipalvelimet, joita ilmaiseen pakettiin kuuluu kaksi, olivat liian hitaita erilaisen prosessori arkkitehtuurin ja keskusmuistin vähäisyyden takia. Virtuaalipalvelin provisioitiin halutuilla parametreillä Oraclen verkkosivujen kautta (ks. kuvio 4).



Kuvio 4. Virtuaalipalvelimen provisiointi.

Verkkopalvelimen käyttöjärjestelmäksi valittiin Ubuntu server 22.04, koska Dockerilla on parempi tuki aarch64-arkkitehtuurilla kyseiselle distribuutiolle kuin muille käyttöjärjestelmille. Provisioinnin jälkeen palvelimelle ajettiin päivitykset komennoilla ”apt update” ja ”apt upgrade”. Tämän jälkeen asennettiin tarvittavat ohjelmistot käyttäen apt-paketinhallinta ohjelmaa. Asennetut ohjelmat olivat Docker ja siihen liittyvät paketit, bash-completition, joka helpottaa komentorivin käytössä ja Nginx, joka vastaa liikenteen ohjauksesta oikeaan konttiin.

Suunnitteluvaiheessa pyrittiin siis luomaan stabiili pohja, jolle olisi helppo rakentaa halutunlainen toteutus. Kuviossa 5 on kuvattu karkea suunnitelma toteutuksen rakenteesta ja tarvittavista komponenteista. Siinä virtuaalikone sisältää Nginx:n liikenteen ohjausta varten ja Dockerin, johon sijoitetaan muut tarvittavat palvelut. Nginx päätettiin asentaa ohjelmana suoraan palvelimelle, koska konfiguraatioita tarvitsi muokata testausten edetessä useasti.



Kuvio 5. Suunniteltu virtuaalikoneen sisältö.

Python valittiin ohjelmointikieleksi sen helppokäyttöisyyden ja keveyden vuoksi. Se mahdollisti myös nopean testauksen eri versioiden välillä, koska ohjelmaa ei tarvinnut erikseen koota ennen ajoa. Python tarjosi myös palvelun toteuttamiseen vaadittavia teknologioita moduulien muodossa, joka nopeutti ohjelmointia entisestään. Python ohjelmat toimivat Docker konteissa, jonka vuoksi niitä on helppo hallita. Pythonin valintaan vaikutti myös aiempi kokemus kielen käytöstä.

Testausvaihe aloitettiin tekemällä kaksi Docker-konttia ja pyrittiin saamaan niiden välinen kommunikaatio toimimaan halutulla tavalla. Alussa vaikeuksia tuotti se, että samaan aikaan piti saada toimimaan pyynnön lähettäminen yhdestä kontista ja vastaanottaminen toisessa kontissa. Näitä asioita oli vaikea testata, koska testin onnistumiseksi vaadittiin, että kummatkin palvelut toimivat, eikä kummankaan toiminnasta voinut varmistua aiemmin. Tämän jälkeen toistettiin sama testi ”POST”-tyyppisillä pyynnöillä niin, että lähetettiin dataa pyynnön mukana. Testauksen viimeiset merkittävät vaiheet olivat tietokantahakujen tekeminen kontissa ajettavasta Python ohjelmasta sekä kuvan ylläpitäminen verkkopalvelimelta niin, että se latautuu sieltä viestin vastaanottajan avatessa sähköpostiviestin.

Tietokanta alustettiin luomalla skeema, johon tehtiin kaksi taulua. Tauluille annettiin nimet ”users” ja ”accesses”. Kumpaankin tauluun tehtiin tarvittavat sarakkeet. Liitteessä 1 on tietokantakyselyt, joilla taulut luotiin (ks. liite 1). Tietokanta käyttää tietojen tallentamiseen compose-tiedostossa määritettyä kansiota isäntäkoneelta (ks. liite 2). Tällä tavoin tietokanta ei menetä alustettua eikä tallennettua tietoa kontin sammutuksen tai uudelleenkäynnistyksen yhteydessä. Tietokannan portti 3306 avattiin isäntäkoneelle auki muuttamalla compose-tiedoston expose-määritys ports-määritykseen ja omalta koneelta otettiin SSH-yhteys, jonka yli siirrettiin avattu 3306 myös avoimeksi omalle tietokoneelle tietokannan alustamista varten.

Dockerin compose-tiedosto luotiin käyttäen hyödyksi Dockerfileissä määriteltäviä uusia kontteja. Valmista esirakennettua konttia käytettiin vain MariaDB-tietokannan luomiseen. Compose-tiedostossa määriteltiin palvelut eli kontit. Kaikille konteille määriteltiin verkko ja alias verkossa, että palvelun sisäisissä kutsuissa voi käyttää IP-osoitteiden sijasta palvelun nimeä. (Ks. liite 2.) Dockerfileissä määriteltiin web-palvelin konteille asennettavat Python-moduulit ja kopioitava app.py tiedosto. Viimeisenä määritettiin ajettava komento, joka käynnistää web-palvelimen. Api- ja tracker-konttien Dockerfile-tiedostot ovat lähes samanlaiset, koska kummassakin käytetään Pythonia ja Flask web-palvelinta. (Ks. liite 3 ja liite 4.)

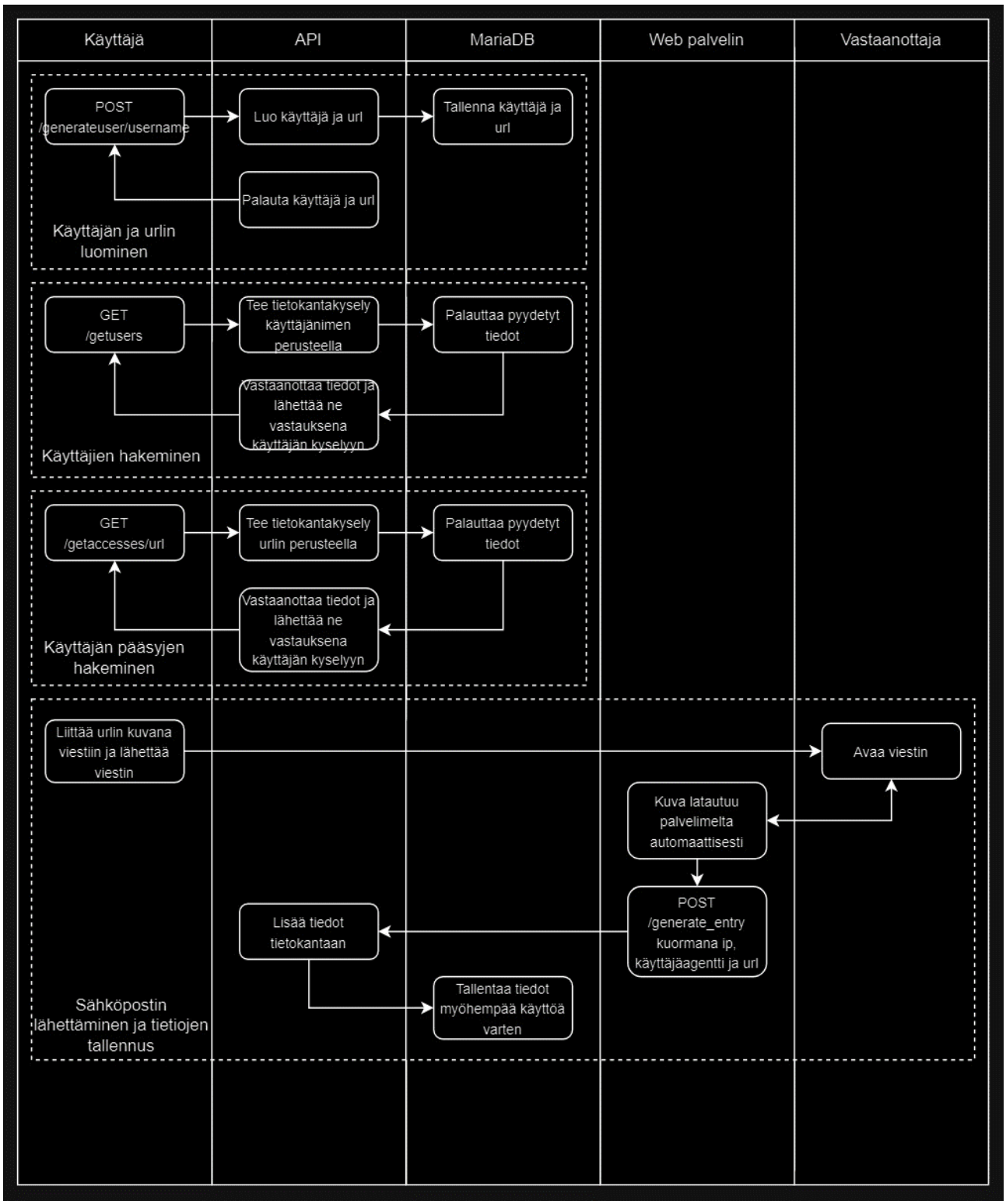
Testausten, konttien ja tietokannan pystyttämisten jälkeen python-sovellusten ohjelmoiminen oli mahdollista. Ensin tehtiin tracker-sovellus, joka palauttaa kuvan, kun pyyntö lähetetään virtuaalipalvelimen osoitteeseen hakemistoon ”/trackme/<image>”. (Ks. liite 5.) ”image” Määrittää, että paikalla voi olla mitä tahansa merkkejä ja ne lähetetään edelleen pyynnön käsittelevälle funktiolle. Ennen tämän ratkaisun löytymistä harkittiin uuden kuvan luomista jokaiselle käyttäjälle, mutta tämä lähestymistapa, olisi täyttänyt virtuaalikoneen tallennustilaa, siten rajoittaen enimmäiskäyttäjä määrää merkittävästi. Funktio lähettää tiedot pyynnön tekijästä API-palvelimelle urliin ”/generate_entry”, joka tekee lisäyksen tietokantaan insert-kyselyllä.

Ohjelmointia jatkettiin tekemällä käyttäjäkohtaisten osoitteiden luontifunktio ja yhdistämällä se API-palvelimen ”/generateuser”-päätepisteeseen. Myös tietokanta kyselyjä tekevät API-palvelimen päätepisteet ohjelmoitiin. ”/getusers” ja ”/getaccesses” hakevat tietoa MariaDB tietokannasta ja muokkaavat sen JSON-formaattiin, jonka jälkeen se palautetaan käyttäjälle. (Ks. liite 6.) Tietokantakyselyiden ja API-päätepisteiden luominen oli nopeaa, kun ensimmäiset funktiot oli saatu luotua.

Kun palvelut oli saatu toimimaan keskenään sisäverkossa, niitä testattiin luomalla erilaisia kyselyitä palveluihin samalta virtuaalikoneelta, jonka jälkeen tehtiin Nginx:lle konfiguraatio, jossa se toimii käänteisenä välityspalvelimena tuotetuille palveluille ja mahdollistettiin liikenne ulkoverkosta palvelimille. (Ks. liite 7.) Tässä yhteydessä varattiin myös ilmainen DNS-nimi suomalaisesta dy.fi-palvelusta. Välityspalvelin konfiguroitiin ohjaamaan liikenne oikeaan konttiin alemman tason domainin perusteella.

5.2 Toiminnot

Toiminnot, joita palvelulla on mahdollista tehdä, tukevat sähköpostiviestin avaamisen seurantaa. Toiminnallisuudet, jotka luotiin ovat kuvattuna kuviossa 6. Ensimmäisenä palvelua käyttävän käyttäjän tulee luoda käyttäjä lähettämällä API-palveluun ”POST”-pyyntö sivuston sijaintiin ”/generateuser/käyttäjän_nimi”. Tämän jälkeen palvelun käyttäjä voi hakea kaikkien käyttäjien tiedot tekemällä pyynnön ”/getusers” sijaintiin. API-palvelu vastaa lähettämällä tietokannasta haetut käyttäjien tiedot JSON-muodossa. Aiempi kysely palauttaa käyttäjille satunnaisesti luodut sijainnit. Käyttäjän on mahdollista tehdä myös pyyntö sijaintiin ”/getaccesses/käyttäjän_url”, jolloin API-palvelin tekee tietokantakyselyn ja palauttaa tiedot kaikista kerroista, kun kuva on ladattu verkosta kyseisen urlin kautta. ”/generate_entry” on sijainti, joka luotiin, että kuvaa ylläpitävän verkkopalvelin voi luoda rivejä tietokantaan. Verkkopalvelin lähettää sijaintiin ”POST”-pyynnön, joka sisältää tarvittavat tiedot kuvan lataajasta ja API-palvelin siirtää tiedot tietokantaan.

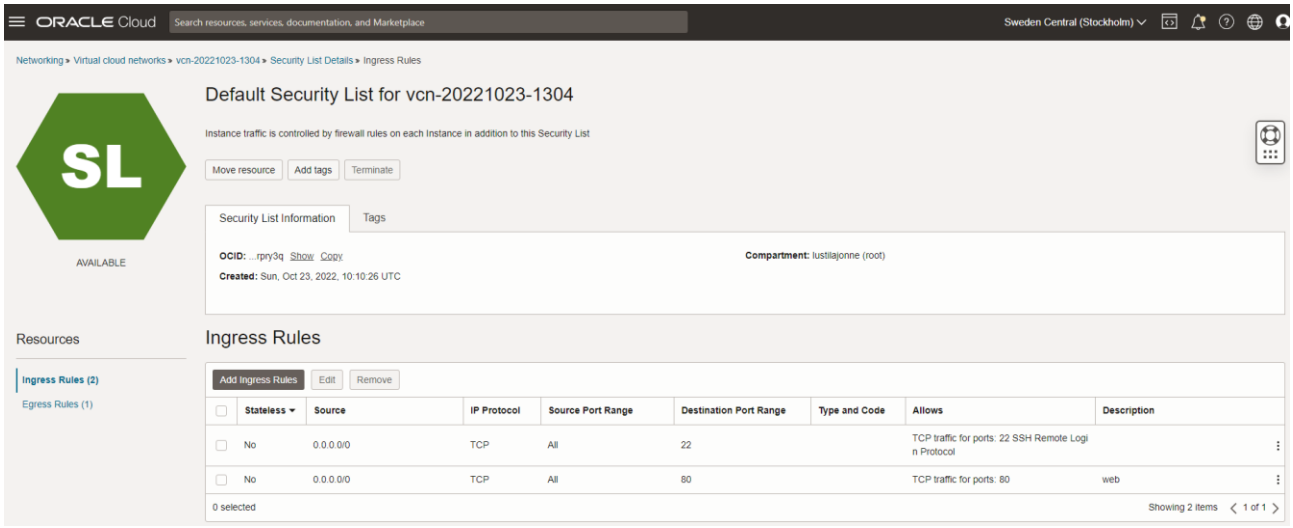


Kuvio 6. Uimaratakaavio palvelun toiminnoista.

5.3 Toiminta ja käyttö

Palvelu toimii yhdellä virtuaalipalvelimella. Kaikki HTTP-protokollan liikenne tulee palvelimelle porttiin 80. Nginx ohjaa liikenteen alemman tason domainista riippuen joko API-palvelimelle tai

seurantapalvelimelle (ks. liite 7). MariaDB ei ole saatavissa ulkoverkosta. Pelkästään portit 80 ja 22 on avoinna ulkoverkkoon Oracle Cloud Infrastructuren käyttöliittymästä (ks. kuvio X.). Porttia 22 tarvitaan SSH-yhteyden luomiseksi palvelimelle. Palvelimet vastaavat pyyntöihin ja tarvittaessa tekevät pyyntöjä tai hakuja muihin palvelimiin, jotka ylläpidetään Dockerissa. Palvelimet vastaavat kaikkiin pyyntöihin, jotka ne vastaanottavat.



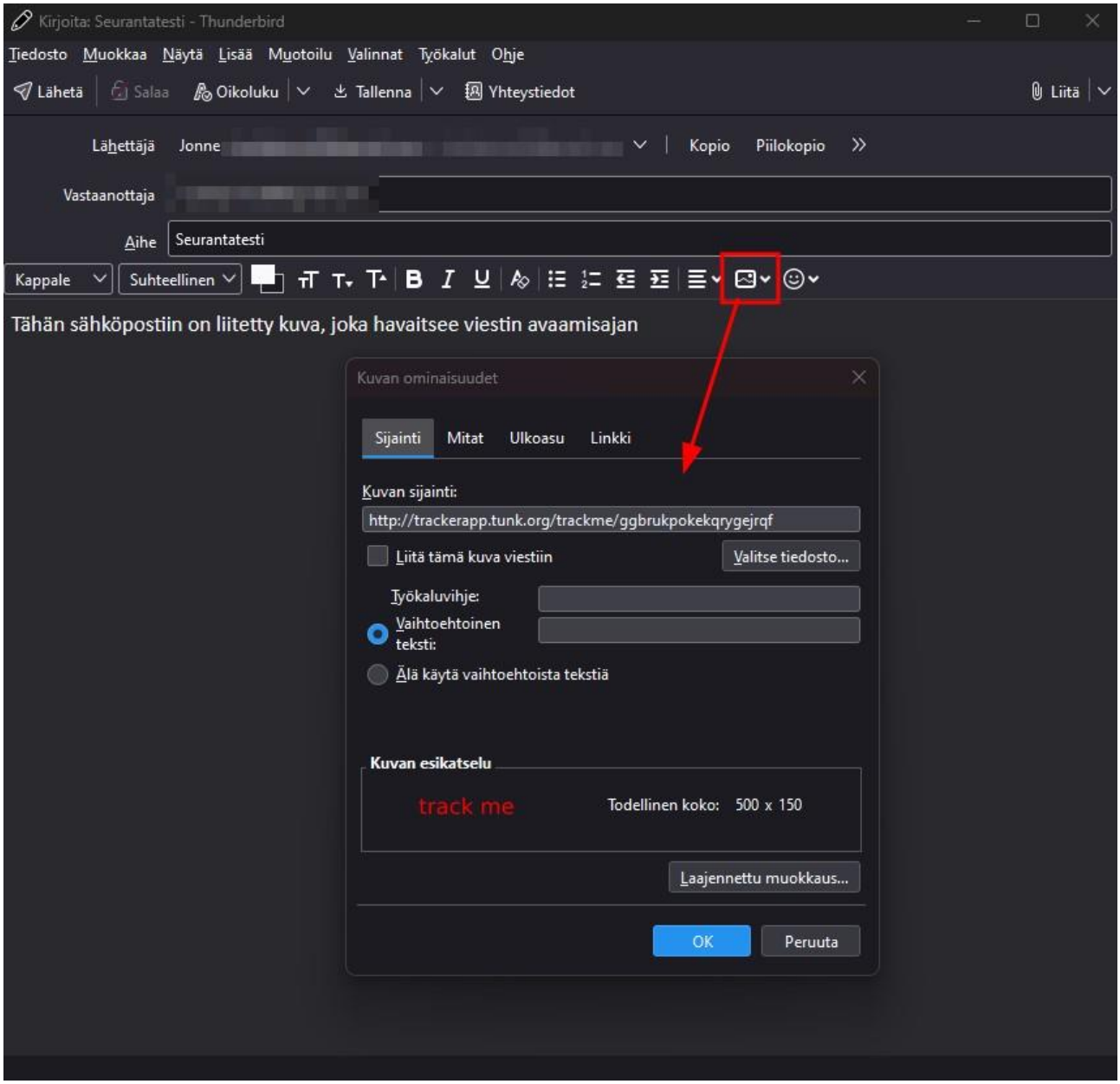
Kuvio 7. Avoimet portit OCI-palvelussa.

Palvelua käytetään komentoriviltä ja siihen on mahdollista tehdä aiemmassa luvussa esiteltyjä pyyntöjä. Kuviossa kahdeksan luodaan kaksi käyttäjää ja sen jälkeen pyydetään käyttäjien tiedot JSON-muodossa. Kun käyttäjä on luotu, palvelu palauttaa käyttäjän nimen ja url:n, jonka voi liittää sähköpostiviestiin. Kaikki aikaleimat tallennetaan UTC-ajassa. Viimeisenä ajettu ”/getusers”-pyyntö ohjataan jq-ohjelmaan, joka muotoilee JSON-tulosteen helpommin luettavaan muotoon. (Ks. Kuvio 8.)

```
jonne@pc:~$ curl -X POST http://api.trackerapp.tunk.org/generateuser/testiteppo
User: 'testiteppo' created, image url is: 'http://trackerapp.tunk.org/trackme/ggbrukpokekqrygejrqf'
jonne@pc:~$ curl -X POST http://api.trackerapp.tunk.org/generateuser/massimatti
User: 'massimatti' created, image url is: 'http://trackerapp.tunk.org/trackme/gwjorihgpjxxemrpmsuo'
jonne@pc:~$ curl -s http://api.trackerapp.tunk.org/getusers | jq
[
  {
    "created": "25-11-2023 19:34:53",
    "id": 1,
    "url": "ggbrukpokekqrygejrqf",
    "username": "testiteppo"
  },
  {
    "created": "25-11-2023 19:34:56",
    "id": 2,
    "url": "gwjorihgpjxxemrpmsuo",
    "username": "massimatti"
  }
]
```

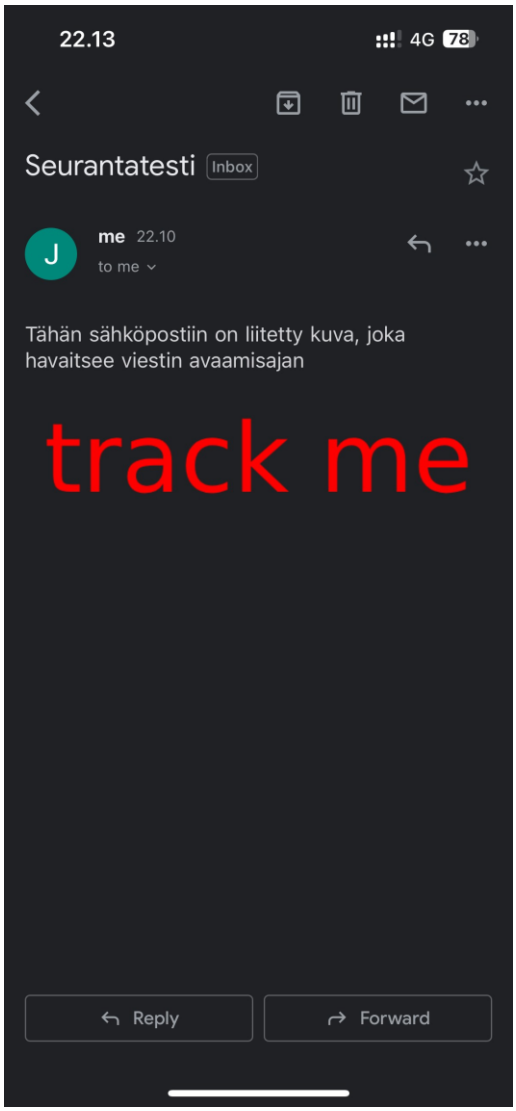
Kuvio 8. Esimerkki käyttäjän luomisesta.

Sähköpostiviesti on helpointa luoda käyttäen esimerkiksi Mozilla Thunderbird-sähköpostiohjelmaa. Lähettäminen onnistui testauksen aikana myös Gmail-verkkosovelluksella, mutta oli työlästä. Alla olevassa kuviossa on näytetty kuvan liittäminen Thunderbirdillä (ks. Kuvio 9). Kuva lisättiin sähköpostiviestiin ja sijainniksi asetettiin ohjelman aiemmin luoma osoite. ”Liitä tämä kuva viestiin”-kohta tulee jättää valitsematta, koska muuten kuvaa ei ladata verkosta lähetyksen jälkeen. Testauksen aikana käytettiin taustaltaan läpinäkyvää kuvaa, jossa lukee punaisella ”track me”. Tämä tehtiin siksi, että olisi helpompaa seurata, mihin viesteihin kuva on liitetty sähköpostisovelluksessa. Kuva voisi kuitenkin yhtä hyvin olla yhden pikselin kokoinen ja väriltään läpinäkyvä tai esimerkiksi palvelua käyttävän henkilön logo, mikäli seuranta haluttaisiin salata viestin saajalta.



Kuvio 9. Kuvan liittäminen sähköpostiin.

Kuvan lähettämisen jälkeen odotettiin muutama minuutti, jonka jälkeen viesti avattiin älypuhelimella (ks. Kuvio 10). Kuvakaappaus otettiin heti, kun viesti oli avattu, jotta aikaleimaa voitaisiin verrata. Älypuhelin oli yhdistettynä eri verkkoon testin aikana, jotta laitteella olisi eri julkisen verkon ip kuin tietokoneella, josta viesti lähetettiin. Kuvakaappauksessa näkyy viestinä ollut teksti, sekä liitetty kuva.



Kuvio 10. Avattu viesti puhelimella.

Viestin avaamisen jälkeen komentoriviltä tehtiin pyyntö ”/getaccesses/ggbrukpokekqrygejrqf”-urliin (ks. Kuvio 11). Kuten kuviosta 11 nähdään, kuva on ladattu neljä kertaa, eikä yhtä, kuten odotettu. Tarkastelemalla käyttäjäagenttia ja ip-osoitteita, huomattiin, että kuva latautuu palvelimelta kolme kertaa, kun se lähetetään Thunderbirdillä. JSON-syötteen viimeinen tietokenttä näyttää kuitenkin tallentaneen viestin avausajan ja muut tiedot oikein.

```
jonne@pc:~$ curl -s http://api.trackerapp.tunk.org/getaccesses/ggbrukpokekqrygejrqf | jq
[
  {
    "accessedAt": "25-11-2023 20:09:19",
    "id": 1,
    "ip": "80.223 [redacted]",
    "url": "ggbrukpokekqrygejrqf",
    "useragent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:115.0) Gecko/20100101 Thunderbird/115.4.2"
  },
  {
    "accessedAt": "25-11-2023 20:09:30",
    "id": 2,
    "ip": "80.223 [redacted]",
    "url": "ggbrukpokekqrygejrqf",
    "useragent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:115.0) Gecko/20100101 Thunderbird/115.4.2"
  },
  {
    "accessedAt": "25-11-2023 20:09:30",
    "id": 3,
    "ip": "80.223 [redacted]",
    "url": "ggbrukpokekqrygejrqf",
    "useragent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:115.0) Gecko/20100101 Thunderbird/115.4.2"
  },
  {
    "accessedAt": "25-11-2023 20:13:10",
    "id": 4,
    "ip": "66.102 [redacted]",
    "url": "ggbrukpokekqrygejrqf",
    "useragent": "Mozilla/5.0 (Windows NT 5.1; rv:11.0) Gecko Firefox/11.0 (via ggpht.com GoogleImageProxy)"
  }
]
```

Kuvio 11. Kuvan lataukset viestin avaamisen jälkeen.

6 Tulokset

Opinnäytetyön tarkoituksena oli selvittää, olisiko mahdollista toteuttaa sähköpostin seurantapalvelu käyttäen ilmaisia resursseja. Työssä käytettiin OCI:n ilmaisen tason resursseja, joilla tehtiin virtuaalikone. Virtuaalikoneelle luotiin palveluita käyttäen avoimen lähdekoodin sovelluksia. Toteutus oli täysin ilmainen ja kenellä tahansa on mahdollisuus toteuttaa vastaava palvelu käyttäen samoja resursseja ilmaiseksi. Tutkimuksen tuloksena syntynyt palvelu ei kuitenkaan ole valmis tuote, vaan enemmänkin konseptitoteutus, jossa pyrittiin muuttamaan teoriatason ajatukset käytännöksi. Tutkimuksen aikana jouduttiin ratkaisemaan myös sellaisia ongelmia, joita ei ollut osattu etukäteen ennakoida.

Ensimmäinen tutkimuskysymys oli miten saada selville sähköpostin avaamisen ajankohta. Opinnäytetyössä avaamisajan selvittäminen ratkaistiin luomalla palvelin, jonka verkko-osoitteesta ladattava kuva liitetään viestiin. Viestin auettua kuva latautuu palvelimelta automaattisesti ja se tallentaa tiedon tietokantaan. Tutkimuksen aikana ei löydetty muuta yhtä joustavaa keinoa saada selville viestin avaamisen ajankohtaa.

Toisena tutkimuskysymyksenä esitettiin, kuinka hallitaan eri henkilöille lähetettyjen viestien seurantaa. Työssä useiden käyttäjien seuraaminen ratkaistiin luomalla tietokantaan käyttäjätaulu, joka ylläpitää listausta käyttäjistä ja heille luoduista satunnaisista osoitteista. Tällä tavoin yhdelle käyttäjälle tai viestille voi luoda oman osoitteen ja niitä voi seurata ilman riskiä, että tulokset sekaantuvat. Tutkimuksen alkuvaiheessa pohdittiin myös uuden kuvan luomista jokaiselle käyttäjälle, mutta se olisi rajoittanut käyttäjien enimmäismäärää virtuaalikoneen tallennustilan täyttyessä. Lopullinen toteutus ratkaisi ongelman jakamalla saman kuvan kaikista eri osoiteyhdistelmistä, jolloin kuvien korkea määrä ei rasita palvelinta.

Viimeinen tutkimuskysymys oli, miten palvelu eroaa kaupallisista ratkaisuista. Omasta mielestäni isoimmat erot kaupallisiin ratkaisuihin ovat rajallisemmat toiminnallisuudet, mutta samaan aikaan varmuus siitä, että käyttäjä hallitsee itse kaikkea keräämäänsä dataa. Myös kaupalliset ratkaisut käyttävät kuvan lataamista merkkinä viestin avaamisesta, mutta ohjelmiston logiikka on viety pidemmälle. Tutkimuksessa tuotetussa palvelussa ei myöskään ole samaa toimintavarmuutta, kuin kaupallisissa ratkaisuissa, koska käyttäjä on lähtökohtaisesti myös palvelun ylläpitäjä.

7 Pohdinta

Opinnäytetyössä saatiin kehitettyä halutunlainen palvelu ennalta määritetyssä ajassa. Myös kaikki toiminnot, joita palveluun haluttiin, toteutuivat. Työn aikana myös osaaminen käytetyiden teknologioiden osalta kehittyi. Palvelua olisi mahdollista jatkokehittää esimerkiksi lisäämällä web-käyttöliittymä ja mahdollisuus lisätä erilaisia kuvia seurantaa varten. Työssä haluttiin kuitenkin rajata laajuus perustoiminnallisuuksiin aikarajoitteiden takia. Nykyinen toteutus vaatii edelleen komentorivin käyttöä, jotta API-palvelimelle voidaan tehdä halutunlaiset kutsut, joten palvelu ei sovellu jokaisen käyttöön ja sitä on epäkäytännöllistä käyttää mobiililaitteella.

Työn alkuvaiheessa oletuksena oli, että useat kaupallisten CRM-ohjelmistojen toiminnallisuudet olisivat olleet helposti tehtävissä ja lisättävissä myöhemmässä vaiheessa tämän työn aikana kehitettyyn palveluun, mutta kuten useassa kehitystyössä vaadittu laajuus selvenee vasta kun työtä aletaan suunnittelemaan ja toteuttamaan saatavilla olevin resurssein. Tästä syystä olikin hyvä, että alusta asti työn rajaus oli tarpeeksi kapea. Tämä oli myös suuri vaikuttava tekijä aikataulussa pysymiseen. On mahdollista, että työn ratkaisu ei ole yhtä optimaalinen kuin kaupallisissa ratkaisuissa, mutta perusajatus ja toimintamalli ovat vastaavat kuin kaupallisissa ratkaisuissa. Työ mahdollistaakin sekä ratkaisun toteutuksen tarkastelun matalammalla tasolla, että toimintaperiaatteen ymmärtämisen, jotka olivat tutkimusta tehdessä tärkeässä roolissa. Työssä myös haluttiin löytää ratkaisu ongelmaan itse, eikä käyttää valmiita ohjelmia tai niiden osia.

Lähteet

About Python. N.d. Tietosivu pythonin verkkosivuilta. Viitattu 1.10.2023. https://www.python.org/about/.

Bhat, S. 2022. Practical Docker with Python : build, release, and distribute your Python app with Docker. 2. p. New York: Apress. https://janet.finna.fi/, Skillsoft Books ITPro.

Choose the plan that best fits your needs. N.d. Mailtrack hinnoittelun verkkosivu. Viitattu 2.10.2023. https://mailtrack.io/en/pricing.

Ghosh, S. 2020. Docker Demystified. Delhi: BPB Publications. https://janet.finna.fi/, Skillsoft Books ITPro.

Gordon, W. 2023. What can CRM do for my business? Blogi artikkeli Nutshell-sivustolla. Julkaistu 23.08.2023. Viitattu 2.10.2023. https://www.nutshell.com/blog/what-does-crm-do.

Grinberg, M. 2018. Flask Web Development. Sebastopol: O'Reilly Media. Viitattu 1.10.2023. https://books.google.com/.

Installing NGINX Open Source. N.d. Nginx dokumentaatio. Viitattu 25.11.2023. https://docs.nginx.com/nginx/admin-guide/installing-nginx/installing-nginx-open-source/.

Järvinen, P. 2022. Yrityksen tietoturvaopas. Helsinki: Kauppakamari. https://janet.finna.fi/, KauppakamariTieto.

Kananen, J. 2015. Kehittämistutkimuksen kirjoittamisen käytännön opas: Miten kirjoitan kehittämistutkimuksen vaihe vaiheelta. Jyväskylä: Jyväskylän ammattikorkeakoulu. https://janet.finna.fi/, Booky.

MariaDB in brief. N.d. MariaDB:n verkkosivut. Viitattu 16.11.2023. https://mariadb.org/en/.

Oracle Cloud Free Tier. N.d. Sivusto Oraclen ilmaisista palveluista. Viitattu 1.10.2023. https://www.oracle.com/cloud/free/.

Peltomäki, J. 2023. Python 3 -ohjelmoinnin perusteet. Helsinki: BoD - Books on Demand. https://janet.finna.fi/, Ellibslibrary.

Peterson, R. 2023. MariaDB vs MySQL - Difference Between Them. Blogipostaus Guru99 verkkosivustolla. Viitattu 28.11.2023. https://www.guru99.com/mariadb-vs-mysql.html.

Png, A. & Demanche, L. 2020. Getting started with Oracle Cloud free tier: Create modern web applications using always free resources. Apress. https://janet.finna.fi/, Skillsoft Books ITPro.

Stoneman, E. 2020. Learn Docker in a Month of Lunches. Shelter Island: Manning. https://janet.finna.fi/, Skillsoft Books ITPro.

Vohra, D. 2016. Pro Docker. Berkeley: Apress. https://janet.finna.fi/, Skillsoft Books ITPro.

Wallenius, N. 2022. Mikä on Docker ja mitä hyötyä siitä on?. Julkaistu 23.2.2022. Viitattu 16.11.2023. https://niklaswallenius.fi/mika-on-docker/ .

What is NGINX? N.d. Nginx:n verkkosivut. Viitattu 25.11.2023. https://www.nginx.com/resources/glossary/nginx/.

Liitteet

Liite 1. tietokannan alustuskyselyt

```
use tracker;
create table if not exists users(
    id int auto_increment,
    username varchar(40) not null,
    created timestamp default current_timestamp,
    url varchar(100),
    primary key(id)
);

create table if not exists accesses(
    id int auto_increment,
    accessedAt timestamp default current_timestamp,
    ip varchar(50),
    useragent varchar(200),
    url varchar(100) not null,
    primary key(id)
);
```

Liite 2. compose.yaml

compose.yaml

```yaml
services:
  api:
    build: ./api/
    ports:
      - "5000:5000"
    networks:
      trackernetwork:
        aliases:
          - api
    depends_on:
      - db
      - tracker
  tracker:
    build: ./tracker/
    ports:
      - "5001:5001"
    networks:
      trackernetwork:
        aliases:
          - tracker
    depends_on:
      - db
  db:
    image: mariadb
    networks:
      trackernetwork:
        aliases:
          - db
    environment:
      - MYSQL_ROOT_PASSWORD=my-secret-pw
      - MYSQL_DATABASE=tracker
    volumes:
      - ./db/data:/var/lib/mysql
    ports:
      - "3306:3306"
networks:
  trackernetwork:
```

Liite 3. api/Dockerfile

api > Dockerfile > ...

```
1  FROM python:3
2  WORKDIR /code
3  COPY requirements.txt .
4  RUN pip install -r requirements.txt
5  EXPOSE 5000
6  COPY . .
7  CMD ["flask", "run", "--host=0.0.0.0"]
```

Liite 4. tracker/Dockerfile

tracker > Dockerfile > ...

```
1  FROM python
2  WORKDIR /code
3  COPY requirements.txt .
4  COPY www/trackme.png /www/trackme.png
5  RUN pip install -r requirements.txt
6  EXPOSE 5001
7  COPY . .
8  CMD ["flask", "run", "--host=0.0.0.0", "--port=5001"]
```

Liite 5. api/app.py

api > app.py > ...

```python
from flask import Flask, jsonify, request
import mariadb
import random
import string
import json

app = Flask(__name__.split(".")[0])

try:
    conn = mariadb.connect(
        user="root",
        password="my-secret-pw",
        host="db",
        port=3306,
        database="tracker"
    )
except mariadb.Error as e:
    print(f"Error connecting to db: {e}")
conn.autocommit = True
cur = conn.cursor()

def create_url():
    return ''.join(random.choice(string.ascii_lowercase) for i in range(20))

@app.route('/')
def get_line():
    return '', 204

@app.route('/getaccesses/<url>')
def get_urlaccesses(url):
    retlist=[]
    cur.execute("SELECT * FROM accesses WHERE url LIKE ?", (url,))
    for (id, accessedAt, ip, useragent, url) in cur:
        retlist.append({'id': id, 'url': url, 'accessedAt': accessedAt.strftime("%d-%m-%Y %H:%M:%S"), 'ip': ip, 'useragent': useragent})
    return jsonify(retlist)

@app.route('/getusers')
def get_users():
    retlist=[]
    cur.execute("SELECT * FROM users")
    for (id, username, created, url) in cur:
        retlist.append({'id': id, 'username': username, 'created': created.strftime("%d-%m-%Y %H:%M:%S"), 'url': url})
    return jsonify(retlist)

@app.route('/generateuser/<username>', methods=['POST'])
def generate_user(username):
    urls=[]
    cur.execute("SELECT url FROM users")
    for url in cur:
        urls.append(url)
    possible_url = create_url()
    if possible_url not in urls:
        cur.execute("INSERT INTO users (username, url) VALUES (?, ?)", (username, possible_url))
        return f"User: '{username}' created, image url is: 'http://trackerapp.tunk.org/trackme/{possible_url}' \n"
    else:
        return '', 409

@app.route('/generate_entry', methods=['POST'])
def generate_entry():
    jsondata = request.get_json()[0]
    ip = jsondata["ip"]
    ua = jsondata["useragent"]
    url = jsondata["url"]
    cur.execute("INSERT INTO accesses (ip, useragent, url) VALUES (?,?,?)", (ip, ua, url))
    return '', 204

```

Liite 6. tracker/app.py

tracker > app.py > ...

```python
from flask import Flask, jsonify, request, send_file
import requests
import sys

app = Flask(__name__.split(".")[0])

@app.route('/trackme/<image>')
def get_image(image):
    ip = request.environ.get('HTTP_X_FORWARDED_FOR')
    if ip is None:
        ip = request.environ['REMOTE_ADDR']
    ua = request.headers.get('User-Agent')
    url = request.url.split("/")[-1]
    jsondata = [{"ip": ip, "useragent": ua, "url": url}]
    try:
        r = requests.post('http://api:5000/generate_entry', json=jsondata)
    except:
        app.logger.error(f"Writing to db failed: URL: {url}, IP: {ip}, User agent: {ua}")
    return send_file("/www/trackme.png", mimetype="image/gif")

@app.route('/')
def get_index():
    return '',204

```

Liite 7. Nginx konfiguraatio

```
server {
        listen          80;
        server_name     api.trackerapp.tunk.org;
        location        / {
                proxy_set_header Host                   $host;
                proxy_set_header X-Real-IP              $remote_addr;
                proxy_set_header X-Forwarded-For        $proxy_add_x_forwarded_for;
                proxy_pass http://127.0.0.1:5000;
                proxy_redirect off;
        }
}

server {
        listen          80;
        server_name     trackerapp.tunk.org;
        location        / {
                proxy_set_header Host                   $host;
                proxy_set_header X-Real-IP              $remote_addr;
                proxy_set_header X-Forwarded-For        $proxy_add_x_forwarded_for;
                proxy_pass http://127.0.0.1:5001;
                proxy_redirect off;
        }
}
```