LAB University of Applied Sciences

REST-rajapinnan pilotointi konttiteknologia-ympäristössä

LAB-ammattikorkeakoulu
Insinööri (YAMK), IoT:stä tekoälyyn
2024
Esa Suhonen

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Esa Suhonen	Opinnäytetyö, YAMK	2024
	Sivumäärä	
	42	

Työn nimi

REST-rajapinnan pilotointi konttiteknologiaympäristössä

Tutkinto ja koulutusala

Insinööri (YAMK), IoT:stä tekoälyyn

Toimeksiantajaorganisaatio

ISS Palvelut Oy

Tiivistelmä

Opinnäytetyössä pilotoitiin REST-rajapinnan toimintaa konttiteknologiaympäristössä. Tavoitteena oli selvittää, kuinka konttiteknologiaympäristö soveltuu REST-rajapinnan kyselyjen tekemiseen ja voidaanko tätä hyödyntää tuotantoympäristössä.

Opinnäytetyö perustuu toiminnalliseen rakennemalliin sekä konstruktiiviseen tutkimusotteeseen. Työssä pystytettiin konttiteknologiaympäristö ohjelmakoodin suorittamista varten. Tätä varten kirjoitettiin ohjelmakoodi, jolla tehtiin kyselyjä REST-rajapintaan ja testattiin kokonaisuuden toimivuutta.

Työn tulokset osoittavat, että konttiteknologiaympäristö tarjoaa tehokkaan ja joustavan ratkaisun REST-rajapintojen kanssa työskenneltäessä. Konttiteknologiaympäristö mahdollistaa ohjelmakoodin helpon ja tehokkaan suorittamisen. Näin ollen konttiteknologiaympäristön käyttöä voidaan suositella tuotantoympäristössä käytettäväksi, sillä se parantaa skaalautuvuutta, suorituskykyä, ohjelmistokehitystä ja ylläpidettävyyttä ohjelmistoprojekteissa.

Asiasanat

konttiteknologia, REST, python, rajapinta, tietokanta

Abstract

Author(s)	Type of Publication	Published
Esa Suhonen	Master's Thesis	2024
	Number of Pages	
	42	

Title of Publication

Piloting a REST API in a container technology environment

Degree, Field of Study

Master of Engineering, From IoT to AI

Organisation of the client

ISS Services Ltd.

Abstract

The thesis piloted the operation of a REST interface in a container technology environment. The aim was to determine how container technology environment is suitable for making queries to a REST interface and whether this can be utilized in a production environment.

The thesis is based on a functional structural model and a constructive research approach. In the job, a container technology environment was set up for executing program code. For this purpose, code was written to query the REST interface and test the functionality of the whole.

The results of the work indicate that the container technology environment provides an efficient and flexible solution when working with REST interfaces. The container technology environment enables easy and efficient execution of the code. Therefore, the use of container technology environments can be recommended for use in a production environment, as it enhances scalability, performance, software development, and maintainability in software projects.

Keywords

container technology, REST, python, interface, database

Sisällys

1 Johdanto

1.1 Toimeksiantajan esittely

ISS Palvelut on alan johtava yritys Suomessa, joka tarjoaa kiinteistö- ja toimitilapalveluita (ISS Yritysvastuuraportti 2024b). Samalla Yritys on Suomen suurimpia yksityisiä työnantajia ja sillä on tärkeä rooli yhteiskunnassa. Yrityksessä työskentelee 7 613 ammattilaista, jotka edustavat 107 eri kansalaisuutta. (ISS Yritysvastuuraportti 2024c.) Yritys toimii Suomessa noin 300 kunnassa ja liikevaihto vuonna 2023 oli 445 miljoonaa euroa. Suomen pääkonttori sijaitsee Helsingissä. (ISS Yritysvastuuraportti 2024b.)

ISS Palvelut on osa kansainvälistä ISS-konsernia, joka on perustettu Tanskassa vuonna 1901. Konsernin pääkonttori on Kööpenhaminassa ja ISS:n maailmanlaajuinen liikevaihto vuonna 2023 oli 10,6 miljardia euroa. ISS työllisti vuonna 2023 maailmanlaajuisesti 30 maassa noin 350 000 ammattilaista. (ISS Yritysvastuuraportti 2024b.)

ISS Palvelut pyrkii rakentamaan asiakkailleen kokonaisvaltaisen palvelukokemuksen, joka edistää heidän liiketoimintaansa luomalla toimivan, älykkään ja tuottavan ympäristön. Yrityksen kokonaispalvelut koostuvat siivous-, kiinteistö-, ravintola- ja työpaikkapalveluista. (ISS Yritysvastuuraportti 2024b.) Yrityksen päämääränä on tehdä maailmasta toimiva luomalla tiloja, joissa on hyvä olla. Yritys huolehtii toimistoista, teollisuuskiinteistöistä, kouluista ja lentoasemista sekä tiloja käyttävistä ihmisistä. ISS Palvelut pyrkii vaikuttamaan ympäristön hyvinvointiin palveluosaamisellaan ja auttamalla asiakkaitaan vastuullisuustavoitteiden saavuttamisessa sekä pitämällä oman liiketoiminnan läpinäkyvänä. (ISS Yritysvastuuraportti 2024c.) ISS:n arvot ovat yhtenäisyys, rehellisyys, vastuullisuus, yrittäjyys ja laatu, jotka ohjaavat jokaisen ISS:läisen päivittäistä toimintaa (ISS Yritysvastuuraportti 2024a).

1.2 Työn tausta ja tavoitteet

Yrityksissä tapahtuvan digitalisaation kasvaessa yhä useammat yritykset ovat ryhtyneet hyödyntämään erilaisia teknologisia ratkaisuja liiketoimintaprosessien tehostamiseksi. Yksi tärkeä kehitysaskel tässä on datan hyödyntäminen REST-rajapintojen avulla. Tämä lähestymistapa mahdollistaa entistä paremman integraation eri järjestelmien välillä ja tarjoaa samalla reaaliaikaista ja tarkkaa tietoa päätöksenteon tueksi.

Yrityksessämme tunnistettiin tarve saada dataa luettua REST-rajapinnasta ja integroitua se osaksi omaa järjestelmäämme. Nykyisessä järjestelmässämme REST-rajapintaa ei voitu hyödyntää halutulla tavalla riittävän tehokkaasti. Tämä johtui nykyisen järjestelmämme teknisistä rajoituksista, joka hyödynsi kolmannen osapuolen tekemää suljettua toteutusta.

Tästä syystä päätettiin tutkia vaihtoehtoisia tapoja, joilla voisimme saavuttaa tavoitteemme ja hyödyntää REST-rajapintoja tehokkaammin. Samalla pystyisimme tehostamaan toimintaamme ja hyödyntämään uutta teknologiaa prosessiemme parantamiseksi. Erityisesti konttiteknologiaympäristöstä suoritettavat kyselyt REST-rajapintoihin tarjoavat joustavan tavan saada dataa REST-rajapinnoista, mikä tekee niistä houkuttelevan vaihtoehdon yrityksellemme.

Opinnäytetyön tavoitteena on toteuttaa tämä konttiteknologiaympäristö käytännössä. Pyrimme selvittämään, miten voimme hyödyntää konttiteknologiaympäristöä, jossa tehtyä REST-rajapintaohjelmakoodia voidaan pilotoida ja testata. Tämän lisäksi pystytetään tietokanta, jonka avulla voimme testata REST-rajapinnasta saatavan datan kirjoittamista tietokantaan. Näin voimme testata tehokkaasti kokonaisuuden ja vastata yrityksemme tarpeisiin. Opinnäytetyön avulla pyrimme saamaan laajemman ymmärryksen siitä, miten REST-rajapintoja voidaan hyödyntää konttiteknologiaympäristössä.

1.3 Tutkimuskysymykset ja rajaus

Opinnäytetyöllä pyritään selvittämään vastaukset seuraaviin tutkimuskysymyksiin:

- Kuinka konttiteknologiaympäristö soveltuu ohjelmakoodin suorittamiseen?
- Kuinka konttiteknologiaympäristö soveltuu REST-rajapinnan kyselyjen tekemiseen ohjelmakoodilla?
- Miten konttiteknologiaympäristö soveltuu hyödynnettäväksi tuotantoympäristössä?

Toteutus REST-rajapinnan osalta rajataan yhteen REST-rajapinta kokonaisuuteen ja testaus suoritetaan tällä laajuudella konttiteknologiaympäristössä. Kehittämistiimin yksikössä oli aikaisemmin linjattu, että tämän tutkimuksen keskeinen lähtökohta on konttiteknologiaympäristö, joka tulee ohjaamaan kehitystiimin yksikön kaikkea kehitystyötä ja siihen liittyviä kokonaisuuksia.

1.4 Tutkimusmenetelmät

Opinnäytetyö perustuu toiminnalliseen rakennemalliin sekä konstruktiiviseen tutkimusotteeseen, jossa pyritään ratkaisemaan reaalimaailman ongelmia. Tutkimuksessa kehitetty ratkaisu on vastaus havaittuun ongelmaan. Tämä ratkaisu käsittää myös käytännön kokeilun sen toimivuuden arvioimiseksi. (Lukka 2001.)

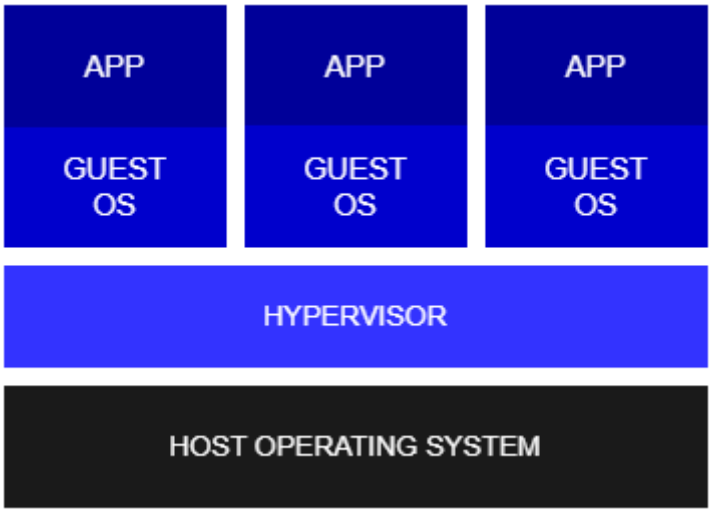
Konstruktiivista tutkimusta kuvaillaan tutkimusprosessina, johon kuuluu 7 erilaista vaihetta (Lukka 2001). Tässä opinnäytetyössä relevanttina ongelmana on REST-rajapinnan

hyödyntäminen konttiteknologiaympäristössä. Tutkimusyhteistyön mahdollisuudet kartoitetaan yrityksen kanssa ja päätetään lähteä etsimään ongelmaan ratkaisua tämän opinnäytetyön avulla. Tutkimus aloitetaan selvittämällä teoreettista osuutta, etsimällä tietoa REST-rajapinnoista ja erilaisista konttiteknologiaympäristöistä. Opinnäytetyön teoreettisen tiedon lähteinä käytettään luotettavia verkkolähteitä ja asiantuntijoiden blogikirjoituksia. Tutkimuksessa innovoidaan ratkaisumalli, jossa tehdään ohjelmakoodi, joka on yhteydessä REST-rajapintaan sekä pystytetään konttiteknologiaympäristö ohjelmakoodin suorittamista varten. Opinnäytetyö toteutetaan ratkaisumallin mukaisesti sekä testataan todellisella datalla. Tutkimuksessa tarkastellaan myös, voidaanko ratkaisua soveltaa muissa vastaavissa tilanteissa. Opinnäytetyön tavoitteena on saada tietoa, kuinka konttiteknologiaympäristö soveltuu REST-rajapinnan kyselyjen tekemiseen ja näiden hyödyntämiseen käytännössä.

2 Virtualisointi ja konttiteknologia

2.1 Virtualisointi

Virtualisointi on tekniikka, joka on osoittautunut tehokkaaksi teknologiaksi. Virtualisointi mahdollistaa usean virtuaalisen ympäristön suorittamisen fyysisen laitteiston päällä. Tietokoneiden virtualisoinnissa lisätään laitteiston ja käyttöjärjestelmän väliin virtualisointikerros. Tämä virtuaalikerros mahdollistaa useiden käyttöjärjestelmäinstanssien samanaikaisen suorituksen virtuaalikoneissa yhdellä tietokoneella. Virtuaalikerros jakaa käytettävissä olevan fyysisen laitteiston resurssit dynaamisesti, kuten prosessorin, tallennustilan, muistin ja I/O-laitteet. (VMware 2008.) Kuvassa 1 on kuvattu virtualisointi.

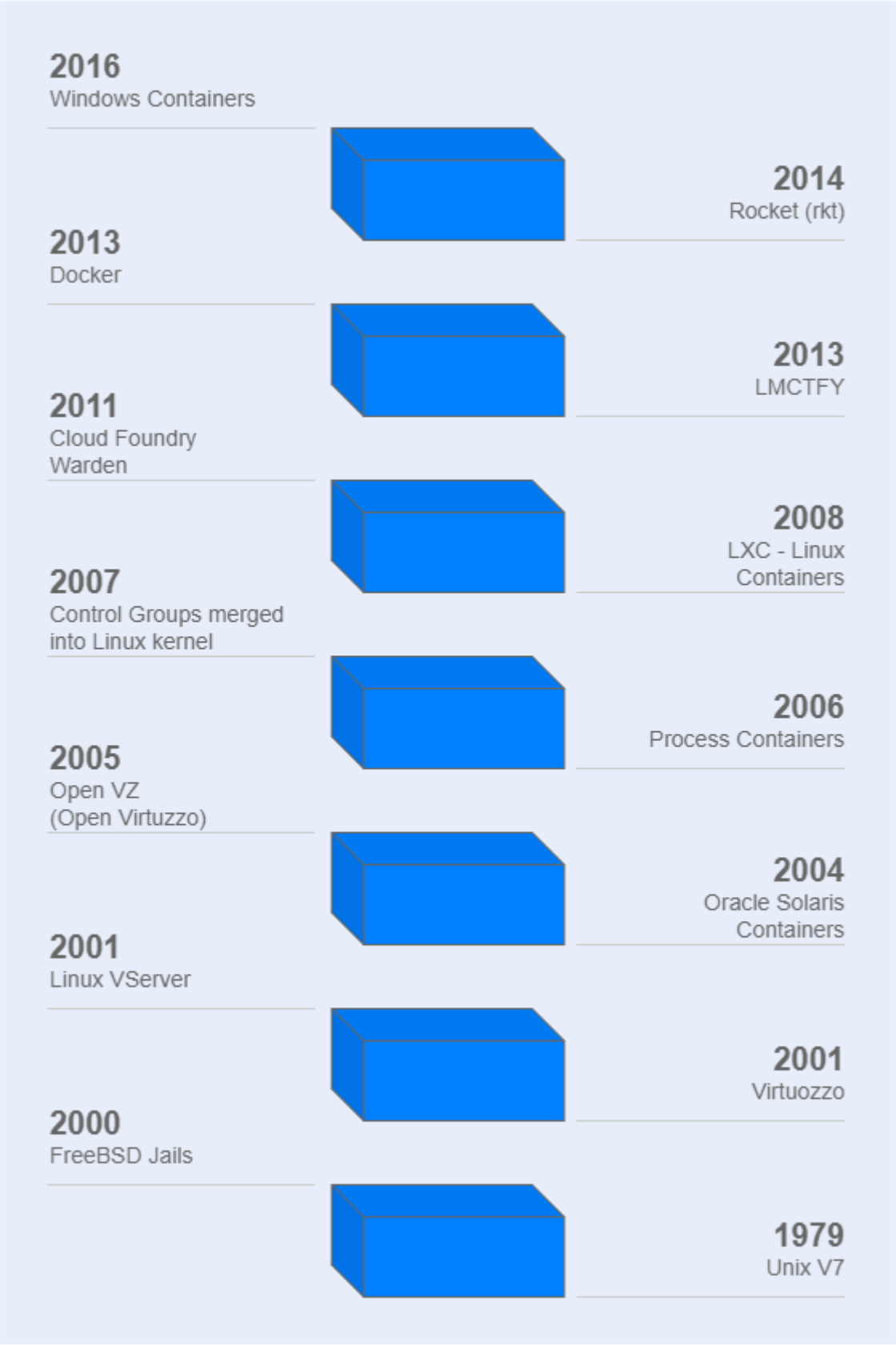


Kuva 1. Virtualisointi (mukailtu Red Hat 2018a)

Virtualisointitekniikoita on useita erilaisia eli virtualisointitekniikkaa on mahdollista hyödyntää todella monipuolisesti eri osa-alueilla. Näitä virtualisointitekniikoita ovat esimerkiksi työpöytävirtualisointi, sovellusvirtualisointi, verkkovirtualisointi ja tallennusvirtualisointi. Erilaisilla virtualisointitekniikoilla pystytään saamaan merkittäviä etuja yrityksille. (IBM.)

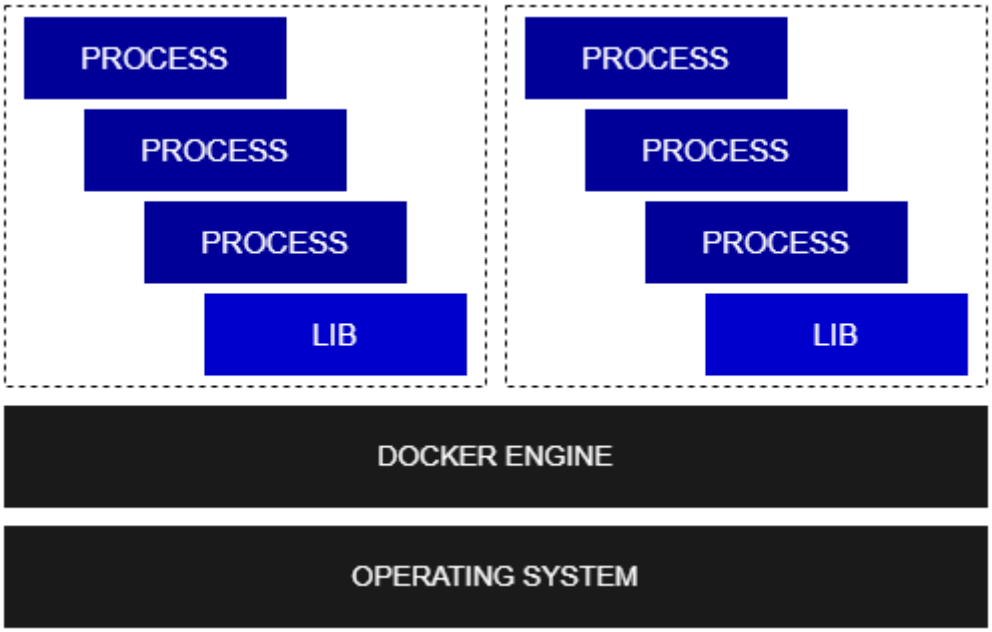
2.2 Konttiteknologia

Konttiteknologia on tekniikka, joka on kovaa vauhtia yleistymässä ja konttiteknologiasta puhutaan yleisesti enemmän ja enemmän. Ennen kuin konttien käyttö yleistyi, yleinen menetelmä sovellusten suorittamiseen oli hyödyntää virtuaalipalvelimia. Vuonna 2001 Virtuozzo toi markkinoille ensimmäisen kaupallisesti saatavilla olleen konttiteknologian (Strotmann 2016). Kuvassa 2 on kuvattu konttiteknologian historia.



Kuva 2. Konttiteknologian historia (mukailtu Strotmann 2016)

Konttiteknologian avulla voidaan pakata erilaisia ohjelmistoja, kehitystä ja käyttöönottoa varten. Kontti sisältää ohjelmiston koodin ja kaikki sen tarvitsemat riippuvuudet, jotta sovellus toimii tehokkaasti ja luotettavasti. Kontit eristävät ohjelmiston ympäristöstään ja varmistavat, että ne toimivat samalla tavalla oli kyseessä kehitys- tai testausympäristö. Kontit toimivat siis aina samalla tavalla eli eivät ole ympäristö tai alusta riippuvaisia. (Docker 2024b.) Kuvassa 3 on kuvattu konttiteknologia.



Kuva 3. Konttiteknologia (mukailtu Red Hat 2018b)

2.3 Virtualisointi vs. konttiteknologia

Konttien ja virtuaalikoneiden suurin ero on niiden arkkitehtuurissa. Virtuaalikoneissa on isäntäkäyttöjärjestelmä ja jokaisella virtuaalikoneella on oma käyttöjärjestelmä, kun taas kontit käyttävät yhteistä isäntäkäyttöjärjestelmää. Tämä tekee konteista kevyempiä ja nopeampia käynnistää, kun taas virtuaalikoneissa jokaisen täytyy käynnistyä omalla käyttöjärjestelmällään. Tästä syystä kontit ovat käteviä monien sovellusten ajamiseen saman käyttöjärjestelmän ytimen päällä, kun taas virtuaalikoneet soveltuvat paremmin tilanteisiin, joissa tarvitaan eri käyttöjärjestelmiä. (Cloud Academy Team 2023.)

Virtuaalikoneiden ja Dockerin välillä on eroja myös turvallisuudessa. Virtuaalikoneet ovat itsenäisiä omalla ytimellään ja turvallisuusominaisuuksillaan toimivia, joten ne soveltuvat sellaisille sovelluksille, jotka tarvitsevat enemmän oikeuksia ja turvallisuutta. Kontit jakavat isäntäytimen ja tästä syystä sovelluksille ei suositella annettavaksi pääkäyttäjäoikeuksia ja sovelluksien suorittamista pääkäyttäjän oikeuksilla. (Cloud Academy Team 2023.)

Dockerin ja virtuaalikoneiden välillä on merkittävä ero siirrettävyydessä. Virtuaalikoneet ovat sidoksissa omaan käyttöjärjestelmäänsä, mikä vaikeuttaa niiden siirtämistä eri alustoille ilman yhteensopivuusongelmia. Kontit sen sijaan ovat itsenäisiä ja voivat suorittaa sovelluksia missä tahansa ympäristössä, koska niillä ei tarvitse olla erillistä käyttöjärjestelmää. Tämän ansiosta kontit voidaan helposti siirtää eri alustoille ja niitä voidaan käynnistää ja pysäyttää hyvin nopeasti verrattuna virtuaalikoneisiin. (Cloud Academy Team 2023.)

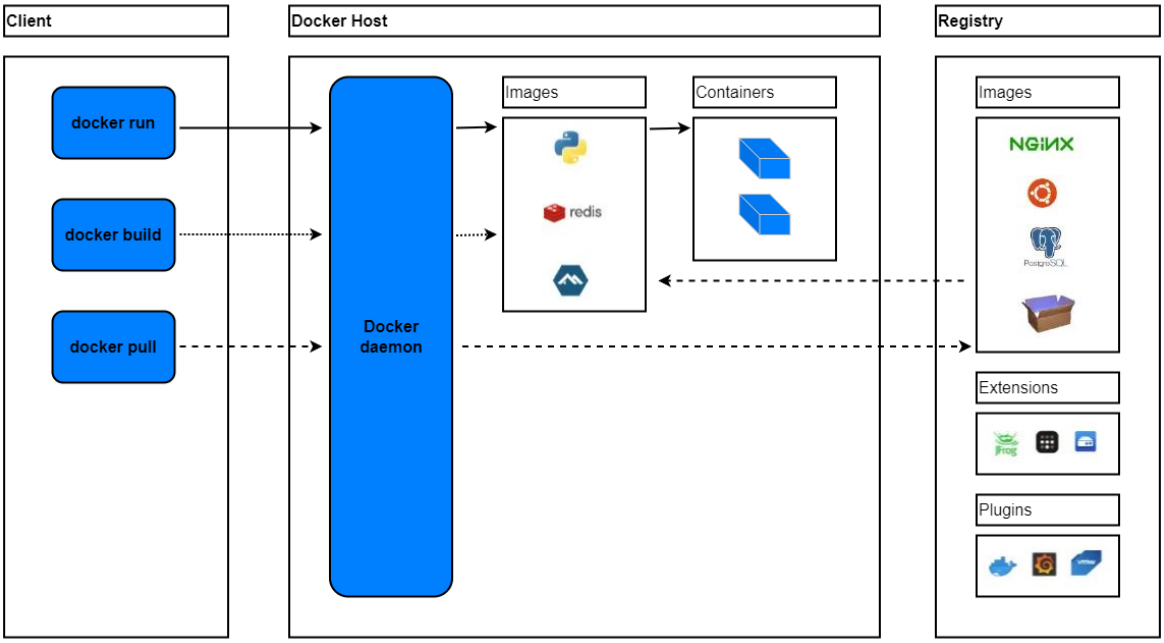
Suorituskyvyssä on myös eroja konttien ja virtuaalikoneiden välillä. Virtuaalikoneet ovat resurssien suhteen vaativampia, koska niiden on ladattava koko käyttöjärjestelmä käynnistyksen yhteydessä. Kontit ovat kevyempiä ja vaativat vähemmän resursseja ja näin ollen käynnistyvät huomattavasti nopeammin. Konttien kopiointi on helppoa verrattuna virtuaalikoneisiin, koska niihin ei tarvitse asentaa käyttöjärjestelmää erikseen. Docker image voi olla jopa kymmenen kertaa pienempi kuin virtuaalikoneen image. (Cloud Academy Team 2023.)

3 Docker

3.1 Yleistä

Docker on avoin alusta, joka helpottaa sovellusten kehittämistä, siirtämistä ja suorittamista. Hyödyntämällä Dockeria ohjelmakoodin testauksessa ja käyttöönotossa, voidaan huomattavasti lyhentää aikaa, joka kuluu ohjelmakoodin siirtämiseen tuotantoympäristöön. Docker mahdollistaa sovelluksen pakkaamisen ja käynnistämisen eristetyssä ympäristössä eli kontissa. Tämä mahdollistaa useiden konttien samanaikaisen ajon samalla isäntäkoneella. Kontit ovat kevyitä ja sisältävät kaiken tarvittavan sovelluksen ajamiseen eli sovellukset eivät ole riippuvaisia isäntäkoneelle asennetuista ohjelmistoista tai käyttöjärjestelmästä, koska kaikki tarvittava on pakattu konttiin. (Docker docs 2024f.)

Dockerin toiminta perustuu asiakas-palvelin-arkkitehtuuriin. Docker-asiakas keskustelee Docker-daemonin kanssa, joka hoitaa Docker-konttien rakentamisen, käynnistämisen ja jakelun. (Docker docs 2024f.) Kuvassa 4 on kuvattu Docker arkkitehtuuri.



Kuva 4. Docker arkkitehtuuri (mukailtu Docker docs 2024f)

3.2 Dockerfile

Dockerfile on tekstitiedosto, jonka avulla rakennetaan Docker image. Dockerfile koostuu muun muassa ADD, CMD, COPY, FROM, RUN komennoista. (Docker docs 2024h.) Jokainen Docker komento luo imageen uuden kerroksen. Kun Dockerfilea muutetaan ja image uudelleen rakennetaan, vain muuttuneet tasot rakennetaan uudelleen. Tämä tekee

imageista pieniä, kevyitä ja nopeita muihin virtualisointiteknologioihin verrattuna. (Docker docs 2024f.)

3.3 Docker Image

Docker Image on vain luku tyyppinen, joka sisältää ohjeet Docker kontin luomiseen. Usein Docker Imaget käyttävät toisia imageja pohjalla, jonka päälle lisättään haluttuja ominaisuuksia. (Docker docs 2024f.) Docker Imageen on sisällytetty riippuvuudet, määritykset ja ympäristömuuttujat eli kaikki tarvittava sovelluksen suorittamiseen. Docker Imagesta tulee kontti, kun se suoritetaan Docker ympäristössä. (Docker docs 2024n.) Valmiita Docker Imageja voidaan ladata Docker rekisteristä (Docker docs 2024f).

3.4 Docker Compose

Docker Compose on työkalu, jolla pystytään määrittelemään ja jakamaan monikonttisovelluksia. Docker Composea käyttämällä voidaan määritellä halutut palvelut YAML tiedoston määrittelyjen perusteella, jonka jälkeen näiden palveluiden käynnistämisen ja sammuttaminen on mahdollistaa yhdellä komennolla. Suuri etu Docker Composen käytössä on, että tätä käyttämällä voidaan määritellä sovelluskokonaisuuksia yhteen tiedostoon ja pitää se projektin juuressa. Tämä mahdollistaa muiden helpon osallistumisen projektiin, koska muiden tarvitsee vain kloonata repository ja käynnistää sovellus käyttämällä Docker Composea. (Docker docs 2024o.)

3.5 Docker Hub

Docker Hub on Dockerin tarjoama alusta, joka mahdollistaa Docker imageiden etsimisen ja jakamisen. Se on globaalisti suurin Docker imageiden paikka, joka kokoaa yhteen erilaisia sisältölähteitä, kuten konttiyhteisön kehittäjiä, avoimen lähdekoodin projekteja ja itsenäisiä ohjelmistotoimittajia, jotka jakavat ja rakentavat konteissa käytettävää koodia. Docker Hub alusta tarjoaa myös mahdollisuuden muokata Docker-tilin asetuksia ja suorittaa hallinnollisia tehtäviä. (Docker docs 2024m.)

3.6 Docker Desktop

Docker Desktop on ohjelma, jonka avulla voidaan jakaa, rakentaa ja ajaa konttisovelluksia ja mikropalveluita. Siinä on graafinen käyttöliittymä muun muassa konttien, sovelluksien ja imageiden hallintaan suoraan omalta tietokoneelta. Docker Desktop säästää aikaa, kun monimutkaiset asetukset voidaan määrittää tämän avulla. Lisäksi Docker Desktop huolehtii muun muassa porttien määrityksistä, tiedostojärjestelmästä ja muista oletusasetuksista.

Docker Desktopia päivitetään säännöllisesti virheenkorjaus päivityksillä sekä tietoturvapäivityksillä. Docker Desktop on saatavana Linux, Windows ja Mac pohjaisille käyttöjärjestelmille. (Docker docs 2024k.)

Docker Desktop sisältää seuraavanlaisia kokonaisuuksia: Docker Engine, Docker CLI client, Docker Scout, Docker Build, Docker Extensions, Docker Compose, Docker Content Trust, Kubernetes ja Credential Helper (Docker docs 2024k).

Docker Engine on avoimen lähdekoodin konttiteknologia, jonka avulla voidaan rakentaa ja kontittaa sovelluksia. Docker Engine toimii asiakas-palvelinsovelluksena, jossa palvelinpuolella on jatkuvasti käynnissä oleva Dockerd daemon prosessi. Docker Engineen kuuluu myös API-rajapinnat, joiden kautta ohjelmat voivat kommunikoida Dockerd daemonin kanssa ja antaa sille komentoja. Lisäksi Docker Enginessä on komentorivikäyttöliittymä, joka käyttää näitä API-rajapintoja Docker daemonin hallintaan, joko suoraan komentojen kautta tai skriptien avulla. Monet muut Docker sovellukset käyttävät taustalla tätä API-rajapintaa ja komentorivikäyttöliittymää. Docker daemon hallitsee ja luo Docker objekteja, kuten imageja, kontteja, verkkoja ja volyymejä. (Docker docs 2024e.)

Docker CLI client on työkalu, jonka avulla voidaan suoraan komentoriviltä hallita Docker kontteja ja konttijärjestelmän eri osa-alueita. Docker CLI client avulla voidaan helposti suorittaa tehtäviä, kuten konttien luomista, käynnistämistä, pysäyttämistä ja poistamista sekä hallita konttien imageja, verkkoja ja volyymejä. Docker CLI client mahdollistaa joustavan ja tehokkaan tavan hallita kontteja ja niiden resursseja. (Docker 2024a.)

Docker Scout on ratkaisu, joka parantaa ennakoivasti ohjelmiston toimitusketjun turvallisuutta. Docker Scout analysoi imageja, jotka koostuvat tasoista ja ohjelmistopaketeista, jotka voivat sisältää haavoittuvuuksia, jotka voivat vaarantaa konttien ja sovellusten turvallisuuden. Docker Scout kokoaa näistä komponenttiluettelon, jota kutsutaan ohjelmistomateriaaliluetteloksi. Ohjelmistomateriaaliluetteloa verrataan jatkuvasti päivittyvään haavoittuvuustietokantaan, jotta voidaan tunnistaa turvallisuusheikkoudet. Docker Scout on erillinen palvelu ja alusta, jonka kanssa voit olla vuorovaikutuksessa Docker Desktopin, Docker Hubin, Docker CLI:n ja Docker Scout Dashboardin kautta. Docker Scout mahdollistaa myös integroinnit kolmannen osapuolen järjestelmien, kuten konttirekisterien ja CI-alustojen, kanssa. (Docker docs 2024g.)

Docker Build on yksi Docker Enginen käytetyimmistä ominaisuuksista. Aina kun luodaan uutta imagea, käytetään Docker Buildia. Docker Build on olennainen osa ohjelmistokehitystä, koska se mahdollistaa koodin paketoimisen ja niputtamisen sekä sen lähettämisen, minne tahansa. Docker Build on muutakin kuin vain komento imageiden rakentamiseen.

Docker Build on kokonainen työkalujen ja ominaisuuksien kokoelma, joka tarjoaa ratkaisuja monimutkaisiin ja vaikeisiin ongelmiin. (Docker docs 2024j.)

Docker Extensions mahdollistaa kolmannen osapuolen työkalujen käytön Docker Desktopissa laajentaen sen toimintoja. Docker Extensionsin avulla voidaan helposti integroida halutut kehitystyökalut osaksi sovelluskehityksen ja käyttöönoton prosesseja. Näitä ovat esimerkiksi vianetsintä, testaus, turvallisuus ja verkkotoiminnot sekä itse luodut mukautetut lisäosat, käyttämällä Extensions SDK:ta. Kuka tahansa voi hyödyntää Docker Extensionseja, eikä niiden asennusten määrällä ole rajoituksia. (Docker docs 2024l.)

Docker Compose on työkalu, jolla voi määrittää ja ajaa useita kontteja sisältäviä sovelluksia, joka mahdollistaa sovellusten sujuvan ja tehokkaan kehittämisen ja käyttöönoton. Docker Compose yksinkertaistaa koko sovelluspinon hallinnan ja yhdestä YAML-määritystiedostosta voidaan hallita palveluita, verkkoja ja volyymejä. YAML-määritystiedoston määrittämisen jälkeen yhdellä komennolla voidaan luoda ja käynnistää kaikki määritystiedostossa määritetyt palvelut. Docker Composea voidaan käyttää kaikissa ympäristöissä, kuten tuotannossa, välivaiheessa, kehityksessä, testauksessa sekä jatkuvan integroinnin tehtävissä. Docker Composella on myös komentoja, joilla voidaan hallita koko sovelluksen elinkaarta, kuten palveluiden käynnistys, pysäytys ja uudelleen rakennus, käynnissä olevien palveluiden tilan tarkastelu, käynnissä olevien palveluiden lokien seuranta sekä yksittäisen komennon suorittaminen palvelussa. (Docker docs 2024d.)

Docker Content Trust mahdollistaa digitaalisten allekirjoitusten käyttämisen dataa lähetettäessä ja vastaanotettaessa Docker etärekistereistä. Nämä digitaaliset allekirjoitukset sallivat tiettyjen imageiden eheyden ja julkaisijan tarkistuksen asiakkaan päässä tai suoritusaikana. Docker Content Trustin avulla imagen julkaisijat voivat allekirjoittaa imaget ja imagen käyttäjät voivat varmistaa, että heidän lataamansa imaget ovat allekirjoitettuja. Julkaisijat voivat olla yksityishenkilöitä tai yrityksiä, jotka joko manuaalisesti allekirjoittavat sisällön tai käyttävät automatisoituja ohjelmistojen toimitusketjuja sisällön allekirjoittamiseen osana julkaisuprosessiaan. (Docker docs 2024c.)

Docker Content Trust on hyödyllinen, kun tietoa siirretään verkossa järjestelmien välillä, jolloin luotettavuus on tärkeää. Erityisesti, kun tietoa siirretään internetin kaltaisten epäluotettavien verkkojen kautta, on kommunikoitaessa tärkeää varmistaa kaiken järjestelmän käyttämän datan eheys ja alkuperä. Docker Engineä käytettäessä imageiden siirtämiseen julkiseen tai yksityiseen rekisteriin, Docker Content Trust mahdollistaa varmistamaan julkaisijoiden luotettavuuden. Sisällön luotettavuus antaa mahdollisuuden tarkistaa, että kaikki rekisteristä saatu data on eheyden ja sen julkaisijan osalta luotettavaa. (Docker docs 2024c.)

Kubernetes on avoimen lähdekoodin järjestelmä konttisovellusten hallintaan useilla palvelimilla. Kubernetes tunnetaan myös nimellä K8s. Kubernetes tarjoaa perusmekanismit sovellusten käyttöönottoon, ylläpitoon ja skaalaukseen. Kubernetes pohjautuu Googlen yli 15 vuoden kokemukseen tuotantokuormien hallinnasta laajassa mittakaavassa, hyödyntäen Borg nimistä järjestelmää, joka yhdistää parhaat yhteisön ideat ja käytännöt. Kubernetesia ylläpitää Cloud Native Computing Foundation. (GitHub 2024b.)

Credential Helper on kokoelma ohjelmia, jotka käyttävät alkuperäisiä varastoja pitääkseen Dockerin tunnistetiedot turvassa (GitHub 2024a). Käyttääksesi tunnistetietoja tarvitset ulkoisen apuohjelman, joka osaa kommunikoida tietyn avainnipun tai muun ulkoisen varaston kanssa. (Docker docs 2024b). Credential Helper protokolla on vahvasti Gitin inspiroima. (Docker docs 2024a).

4 Rajapinta

4.1 Yleistä

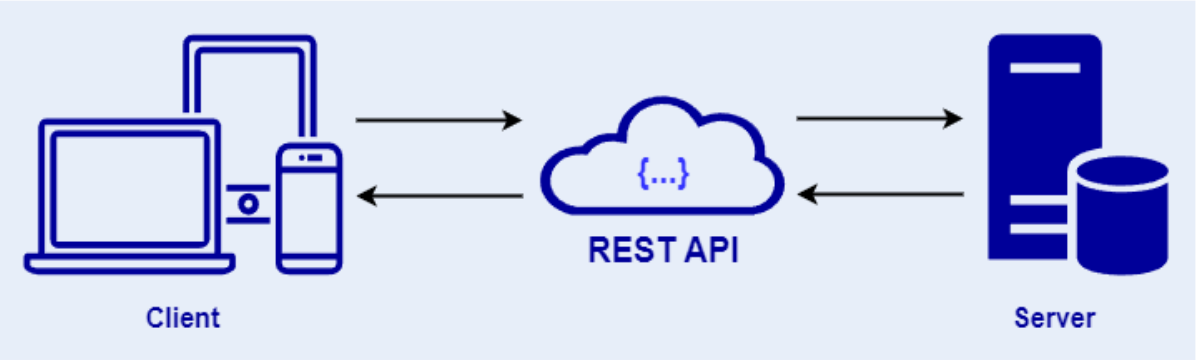
Monet ohjelmistot hyödyntävät rajapintoja, jotka mahdollistavat tietojen hakemisen, lähettämisen tai tietyn toiminnon suorittamisen palvelimella Internetin välityksellä. Palveluiden rajapintojen käyttö on erittäin yleistä modernissa ohjelmistokehityksessä. Rajapinnat mahdollistavat monimutkaisten ja dynaamisten sovellusten luomisen, joilla on pääsy moniin erilaisiin tietolähteisiin ja palveluihin. Palveluiden rajapinnat tarjoavat kehittäjille standardoidun tavan kommunikoida erilaisten verkkopalveluiden kanssa.

On olemassa valtava määrä erilaisia rajapintoja eri käyttötarkoituksiin ja teknologioihin. Näitä rajapintoja voi olla tarjolla esimerkiksi sosiaalisen median alustoilla, pilvipalveluissa, maksujärjestelmissä, IoT-laitteissa, tekoälyratkaisuissa ja monissa muissa sovellusympäristöissä. Nämä rajapinnat mahdollistavat eri järjestelmien välisen kommunikaation ja integraation, mikä edistää tehokkuutta, skaalautuvuutta ja innovaatiota digitaalisissa ympäristöissä.

4.2 REST

REST (Representational State Transfer) on ohjelmointirajapintojen toteuttamiseen tarkoitettu arkkitehtuurityyli, jota käytetään ohjelmointirajapintojen toteuttamiseen. Roy Fielding esitteli REST:n ensimmäisen kerran vuonna 2000 väitöskirjassaan. (Gupta 2023b.)

REST-rajapinnat yleensä välittävät tietoa JSON- tai XML-formaatissa, jolloin dataa voidaan siirtää helposti palvelimelta toiselle. JSON (JavaScript Object Notation) on näistä yleisempi käytettävä formaatti modernissa ohjelmistokehityksessä. REST tarjoaa yksinkertaisen, skaalautuvan, suorituskykyisen ja joustavan lähestymistavan järjestelmien rakentamiseen ja integrointiin. Nämä ominaisuudet korostavat REST:n monipuolisuutta ja sen tarjoamia etuja modernissa ohjelmistokehityksessä ja integroinnissa. Kuvassa 5 on kuvattu REST-rajapinta.



Kuva 5. REST-rajapinta (mukailtu Xu 2022)

Yleisemmin käytettävät HTTP-menetelmät REST:n yhteydessä ovat GET, POST, PUT, DELETE ja PATCH. Harvemmin käytettävät HTTP-menetelmät ovat puolestaan HEAD, OPTIONS, TRACE ja CONNECT.

GET hakee resurssin tiedot palvelimelta. GET-pyyntö ei muuta resurssin tilaa tai dataa, vaan palauttaa vain resurssin tiedot. Koska GET-pyyntö ei muuta resurssin tilaa, vaan lukee, niin tätä voidaan kuvata turvalliseksi menetelmäksi. (Gupta 2023a.) Tämän metodin avulla voidaan kysyä esimerkiksi haluttujen tietojen arvoja.

POST luo uuden resurssin palvelimelle. POST-pyyntö lähettää dataa palvelimelle, joka sitten käsittelee tiedot ja luo uuden resurssin. Koska POST luo uusia resursseja, niin tätä menetelmää ei voida kuvata turvalliseksi. (Gupta 2023a.) Tällä menetelmällä voidaan luoda palvelimelle esimerkiksi erilaisia tilauksia ja ilmoituksia.

PUT päivittää olemassa olevan resurssin tiedot palvelimella. PUT-pyyntö lähettää koko resurssin tiedot, jotka korvaavat olemassa olevan resurssin tiedot. Tätä menetelmää ei voida kuvata turvalliseksi, koska tämä päivittää tietoja. (Gupta 2023a.) Tämä menetelmä mahdollistaa esimerkiksi luodun tilauksen uusimisen ennen kuin se kerkeää vanhenemaan määritetyn ajan kuluessa, uusinnan yhteydessä voidaan määrittää, kuinka kauan tämä tilaus on jatkossa voimassa.

DELETE poistaa olemassa olevan resurssin palvelimelta (Gupta 2023a). Tätä menetelmää ei voida kuvata turvalliseksi, koska tämä poistaa tietoja (McKenzie 2023). Tämä menetelmä mahdollistaa esimerkiksi luotujen ilmoituksien poistamisen.

PATCH päivittää osan olemassa olevan resurssin tiedoista. PATCH-pyyntö lähettää vain muuttuneet tiedot, jotka päivitetään resurssiin. (Gupta 2023a.) Koska PATCH päivittää osan resurssin tiedoista, niin tätä menetelmää ei voida kuvata turvalliseksi (McKenzie 2023).

HEAD palauttaa resurssin metadataa, kuten otsikot ja olemassaolon tarkistuksen, mutta ei itse resurssin sisältöä (McKenzie 2023). Tätä voidaan kuvata turvalliseksi menetelmäksi, koska ei tee palvelimelle muutoksia, kuten päivitä tai poista tietoja (Gupta 2023a).

OPTIONS palauttaa tiedot resurssin tukemista HTTP-metodeista ja muista toiminnoista (McKenzie 2023). Tätä voidaan kuvata turvalliseksi menetelmäksi, koska ei tee palvelimelle muutoksia, kuten päivitä tai poista tietoja (Gupta 2023a).

TRACE menetelmää käytetään virheenkorjauksessa ja vianetsinnässä (McKenzie 2023). Koska tämä ei tee palvelimelle muutoksia, niin tätä voidaan kuvata turvalliseksi menetelmäksi (Gupta 2023a).

CONNECT menetelmää käytetään yleensä yhteyden ottamiseen välityspalvelimeen, joka mahdollistaa tunnelin avaamiseen ja pääsyn ulos paikallisesta verkosta. Tätä menetelmää ei voida kuvata turvalliseksi, koska tämä mahdollistaa muutoksien tekemisen palvelimelle. (McKenzie 2023.)

5 REST-rajapinta ohjelmakoodin testaus konttiteknologiaympäristössä

5.1 Ohjelmakoodi

5.1.1 Ohjelmointikielen valinta

Opinnäytetyön ohjelmakoodin toteuttamisen alkuvaiheessa oli ratkaistava, millä ohjelmointikielellä työ toteutetaan. Tämä valinta oli tärkeä, sillä valittu kieli tulisi vaikuttamaan opinnäytetyön tekniseen toteutukseen sekä lopputuloksen laatuun. Useita vaihtoehtoja harkittiin perusteellisesti ja lopulta päädyttiin valitsemaan Python-ohjelmointikieleksi, koska yhtenäistä käytäntöä ohjelmointikielen osalta ei ollut ennestään kehittämistiimin yksikössä linjattu.

Python valittiin sen monipuolisuuden ja laajan käytettävyyden vuoksi. Python tarjoaa selkeän ja helposti luettavan syntaksin, mikä helpottaa ohjelmakoodin ymmärtämistä ja ylläpitoa. Tämä on erityisen tärkeää opinnäytetyössä, jossa ohjelmakoodin selkeys ja ylläpidettävyys ovat keskeisiä tekijöitä. Tämä mahdollistaa, että ohjelmakoodin ymmärtäminen, omaksuminen ja muokkaaminen on sujuvaa myös muille käyttäjille.

5.1.2 Python-ohjelmointikieli

Python-ohjelmointikielen loi Guido van Rossum niminen henkilö. Kun hän aloitti Pythonin toteuttamisen, hän luki samanaikaisesti "Monty Python’s Flying Circus" nimisen BBC:n komediasarjan julkaistuja käsikirjoituksia. Hän halusi keksiä nimen, joka olisi lyhyt, uniikki ja hieman salaperäinen, joten hän päätyi nimeämään kielen Pythoniksi. (Python docs 2024b.)

Python on olio-ohjelmointikieli, joka tarjoaa monipuoliset ominaisuudet ja mahdollisuudet ohjelmoinnin eri osa-alueilla. Se on tulkattava kieli, mikä tarkoittaa, että ohjelmakoodia ei tarvitse erikseen kääntää konekieleksi, vaan se suoritetaan suoraan ohjelmakooditiedostosta. Python on myös interaktiivinen, mikä mahdollistaa ohjelmakoodin suorittamisen ja tulosten näkemisen välittömästi, mikä on hyödyllistä ohjelmakoodin testauksessa ja kehityksessä. Python toimii Linux, Mac ja Windows käyttöjärjestelmissä. (Python docs 2024a.)

Pythonilla on laaja suosio ja aktiivinen kehittäjäyhteisö. Python tarjoaa valtavan valikoiman erilaisia kirjastoja, jotka ovat valmiiksi hyödynnettävissä. Lisäksi Pythonin saatavilla olevat dokumentaatiot ovat erinomaisia, mikä tekee kielen oppimisesta ja hallinnasta sujuvaa ja vaivatonta. Python tarjoaa vahvan perustan opinnäytetyön vaatimille teknisille ominaisuuksille ja mahdollistaa sujuvan kehitysprosessin sekä lopputuloksen.

5.1.3 Ohjelmakoodin määrittäminen

Ennen ohjelmakoodin määrittämistä tutustuttiin käytettävään REST-rajapintaan sekä sen perustoimintoihin. Tämän lisäksi REST-rajapintoihin liittyvää tietoa etsittiin verkosta, mikä tarjosi hyödyllistä taustatietoa toteutusta suunniteltaessa. Käytettävään REST-rajapintaan tehtiin testikyselyjä Swagger UI käyttöliittymästä. Testikyselyjen avulla selvitettiin, miten tietoja voidaan hakea ja millä tavalla REST-rajapinta reagoi erilaisiin kyselyihin sekä saatiin käsitys, kuinka voitiin kysyä käytettävästä rajapinnasta usean pisteen tiedot yhdellä kyselyllä. Tämä antoi hyvän käsityksen siitä, millainen ohjelmakoodin tulisi olla, jotta se ottaisi huomioon mahdollisimman laajasti erilaiset tilanteet.

Ohjelmakoodin määrittelyssä pyrittiin huomioimaan useita erilaisia toiminnallisuuksia, jotka toteutetaan erillisinä funktioina ohjelmassa. Ohjelmakoodissa tarvittiin toiminto, joka hakee REST-rajapinnasta tokenin. Tokenia tarvittiin, jotta voitiin tehdä HTTP-pyyntöjä palvelimen REST-rajapintaan. Tokenin hakeminen oli olennainen osa ohjelmakoodin toimintaa, koska ilman voimassa olevaa tokenia REST-rajapintakyselyjä ei voitaisi suorittaa, sillä palvelin hyväksyy REST-rajapintakyselyt ainoastaan tunnistautuneilta käyttäjiltä. Kuviossa 1 on kuvattu työnkulun prosessin eri vaiheet, joista nähdään selkeästi, miten kukin vaihe etenee ja linkittyy seuraavaan vaiheeseen.

Ohjelmakoodiin tarvittiin myös ominaisuus, joka lukee tiedostoa, jonne käyttäjä voi määrittää REST-rajapinnasta haettavat pisteet. Samaan tiedostoon määritettiin myös pisteen yksilöllinen tunnus. Tiedostosta luettavat pisteet ovat niitä pisteitä, joiden arvoja halutaan kysyä REST-rajapinnan avulla. Tiedostossa olevaa yksilöllistä tunnusta varten tuli kirjoittaa funktio, joka tarkisti, että yksilöllinen tunnus oli oikean mittainen sekä sisälsi vain tiettyjä kirjaimia. Tämän tavoitteena oli varmistaa, ettei myöhemmässä vaiheessa synny inhimillisen virheen seurauksena ongelmia.

Kun kaikki pisteet oli luettu tiedostosta, tuli REST-rajapintaan luoda tilaus, jolle annettiin parametreina edellisessä kohdassa tiedostosta luetut pisteet. Tilausta käytettiin, koska tarkoituksena oli saada yhdellä kyselyllä mahdollisimman monen pisteen tiedot samalla kertaa. Tämä mahdollisti sen, että tehdyt kyselyt eivät kuormittaneet palvelimen REST-rajapintaa, kun kaikki pisteet kysyttiin yhdellä kyselyllä eikä yksitellen.

Kun tilaus oli luotu, tuli REST-rajapintaan luoda ilmoitus, jolle annettiin parametrina tilaus kohdassa palautuksena saatu ID. Kun ilmoitus oli valmis, voitiin aloittaa pisteiden arvojen kysely käyttämällä HTTP GET menetelmää, antamalla tämän kyselyn parametreiksi tilaus kohdassa palautuksena saatu ID. Kun pisteiden arvot saatiin luettua ohjelmakoodiin, tuli ne tulostaa käyttäjälle järkevässä muodossa.

Ohjelmakoodissa tulostamisen tuli tapahtua siten, että jokaisen pisteen kohdalla tulostettiin aikaleima, pisteen nimi, pisteen arvo, pisteen laatu sekä yksilöllinen tunnus. Tulostuksen tiedot erotettiin käyttämällä #-merkkiä. Tämä teki saaduista tiedoista helposti luettavia ja ymmärrettäviä käyttäjälle. Lisäksi pisteiden tiedot olivat näin tehtynä jatkossa helppo viedä esimerkiksi tietokantaan tai muuhun järjestelmään.

Kun arvot oli tulostettu, tuli ilmoitus poistaa. Tämä tuli tehdä, kun arvoja ei enää tarvittu ja ohjelmakoodi on saanut kaikki tarvittavat tiedot sekä käsitellyt nämä tiedot. Lopuksi tuli uusia tilaus, jotta tilaus ei päässyt vanhenemaan.

Ohjelmakoodin tuli kirjoittaa virheilmoitukset erilliseen tiedostoon. Tämä mahdollisti ohjelmakoodin sekä REST-rajapinnan toiminnan seuraamisen tehokkaasti yhden tiedoston avulla, kun kaikki virheilmoitukset olivat yhdessä tiedostossa. Tätä varten kirjoitettiin oma Python kirjasto, joka mahdollisti tämän kirjaston hyödyntämisen myös tulevissa ohjelmakoodeissa, kun kirjasto sisällytetään osaksi uutta ohjelmakoodia.

Kuvio 1. Työnkulun prosessikaavio

5.1.4 Python HTTP-kirjaston valinta

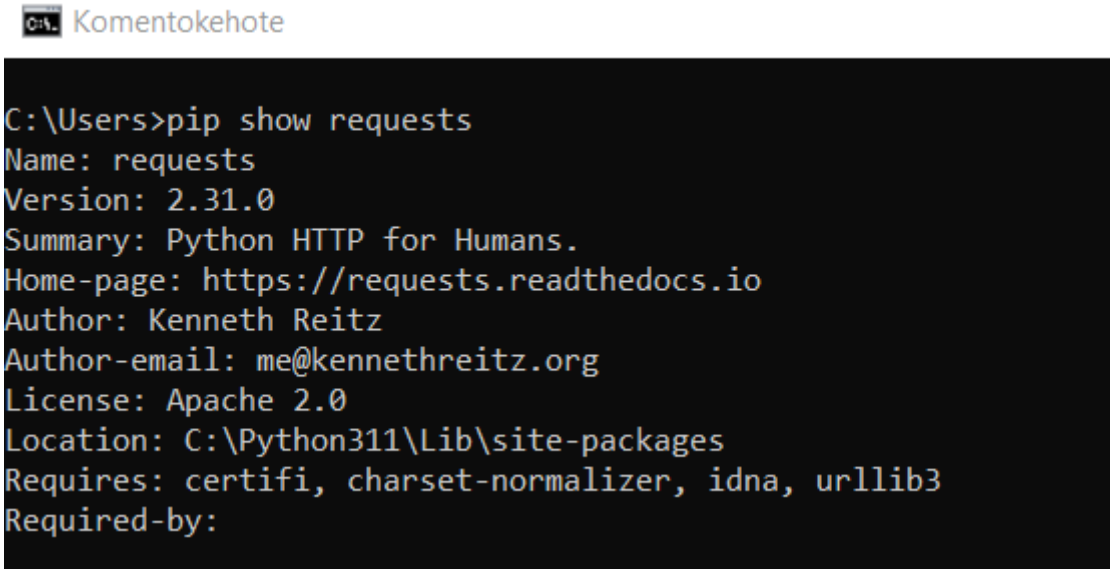
Python tarjoaa monipuolisen ja laajan valikoiman erilaisia kirjastoja kehitykseen. Ohjelmakoodin määrittämisen yhteydessä vertailtiin erilaisia kirjastoja, joiden avulla pystyttiin

kommunikoimaan REST-rajapinnan kanssa. Perusteellisen harkinnan jälkeen useista vaihtoehdoista päädyttiin hyödyntämään Requests nimistä kirjastoa, joka on yksi suosituimmista kirjastoista HTTP-pyyntöjen tekemiseen Pythonissa. Tämä kirjasto tarjosi yksinkertaisen ja selkeän tavan tehdä HTTP-pyyntöjä eri verkkoresursseihin, kuten REST-rajapintoihin.

Requests kirjaston avulla pystyttiin helposti luomaan ja lähettämään HTTP-pyyntöjä REST-rajapintaan Python ohjelmakoodissa. Pythonin ja Requests kirjaston yhdistelmä tarjoaa tehokkaan ja joustavan tavan kommunikoida REST-rajapinnan kanssa. Tämä mahdollistaa sujuvan integraation ulkoisten palveluiden kanssa ja on keskeisessä osassa tämän opinnäytetyön teknisessä toteutuksessa.

5.1.5 Python HTTP-kirjaston asennus

Pythonin Requests kirjasto asennettiin Pythonin PIP paketinhallintatyökalun avulla ”pip install requests” komennolla. Kirjaston onnistunut asennus varmistettiin ”pip show requests” komennolla kuvan 6 mukaisesti. Tiedoista nähtiin muun muassa paketin versio, sijainti ja lisenssitiedot.



Kuva 6. Requests kirjaston tietoja

5.1.6 Visual Studio Code tekstieditori

Ohjelmakoodin kirjoittamiseen valittiin Visual Studio Code niminen ohjelma, joka on kehittäjien suosima avoimen lähdekoodin tekstieditori. Visual Studio Code tarjoaa monipuolisen valikoiman työkaluja eri ohjelmointikielten kehittämiseen, kuten JavaScript, Python, C++ ja moniin muihin. Tämä kehitysympäristö on suosittu erityisesti sen monipuolisten

ominaisuuksien ansiosta. Sen käyttöliittymä on yksinkertainen ja helppokäyttöinen, mikä tekee siitä hyvän valinnan niin aloittelijoille kuin kokeneillekin kehittäjille. Visual Studio Code tarjoaa tehokkaan ohjelmakoodieditorin, jossa on automaattinen täydennys, syntaksin korostus ja monirivimuokkausmahdollisuus, mikä tehostaa ohjelmakoodin kirjoittamista ja auttaa virheiden löytämistä.

Visual Studio Code tukee laajaa valikoimaa laajennuksia ja lisäosia, jotka mahdollistavat sen muokkaamisen omiin tarpeisiin sopivaksi. Kehittäjät voivat integroida Visual Studio Codeen erilaisia työkaluja ja palveluita, kuten versionhallintaa, testausta ja vianetsintää. Visual Studio Code on erittäin suorituskykyinen ja kevyt. Visual Studio Code on saatavana Windows, Linux ja Mac pohjaisille käyttöjärjestelmille.

5.1.7 Ohjelmakoodin kirjoittaminen

Ohjelmakoodin kirjoittaminen toteutettiin luvun 5.1.3 Ohjelmakoodin määrittäminen -luvun mukaisesti. Ohjelmoinnin alkuvaiheessa keskityttiin ohjelmakoodin kirjoittamiseen, jonka avulla pystyttiin kysymään yksittäisten pisteiden arvoja REST-rajapinnasta. Ensimmäisen vaiheen keskeisenä tavoitteena oli varmistaa, että yksittäisten pisteiden kyselyt saatiin toimimaan luotettavasti ja tehokkaasti, sillä tämä oli perusta myöhemmille vaiheille. Tässä vaiheessa ohjelmakoodiin ohjelmoitiin myös pisteen yksilöllisen tunnuksen tarkistuksen ominaisuus. Yksilöllisen tunnuksen avulla pystyttiin varmistamaan mistä pisteestä on kyse ja tällä tavalla varmistettiin, ettei pisteet sekoitu myöhemmin keskenään.

Kun nämä vaiheet oli saatu toteutettua ohjelmakoodissa onnistuneesti, siirryttiin seuraavaan vaiheeseen, jossa keskityttiin kysymään REST-rajapinnasta usean pisteen tiedot yhdellä kyselyllä. Tämän vaiheen pääasiallisena tavoitteena oli optimoida REST-rajapinnalle tehtävien kyselyjen määrä ja varmistaa, että kyselyt eivät kuormittaisi turhaan palvelimen REST-rajapintaa. Lisäksi pyrittiin siihen, että kyselyjen suorittaminen REST-rajapintaan olisivat mahdollisimman tehokkaita.

Usean pisteen kyselyn toteutuksen yhteydessä ohjelmakoodiin lisättiin ominaisuus, jonka avulla käyttäjä voi määrittää REST-rajapinnasta haettavat pisteet ja niiden yksilölliset tunnukset erillisessä tiedostossa. Tämä ominaisuus helpottaa huomattavasti REST-rajapinnasta haettavien pisteiden hallintaa, sillä käyttäjän ei tarvitse määrittää haettavia pisteitä varsinaiseen ohjelmakoodiin. Käyttäjä voi näin ollen hallinnoida haettavia pisteitä tehokkaasti ja joustavasti, mikä mahdollistaa nopean ja vaivattoman järjestelmän päivittämisen ja laajentamisen ilman, että itse ohjelmakoodia tarvitsee muuttaa. Tämä erottelu tekee myös testauksesta ja vianmäärityksestä helpompaa sekä vähentää inhimillisen virheen

mahdollisuuksia ohjelmakoodin osalta, kun käyttäjä voi määrittää haettavat pisteet suoraan yhdessä erillisessä tiedostossa.

Usean pisteen kyselyjä tehtäessä REST-rajapintaan huomattiin ongelma, jossa palautuksena ei saatu mittauksen laatua ja pisteen nimeä. Tämä havaittiin, kun verrattiin tilannetta yksittäisten pisteiden kyselyihin, joissa kyseiset tiedot saatiin ongelmitta. Ongelma ratkaistiin määrittämällä mittauksen laatu ja pisteen nimi samaan tiedostoon, johon oli jo aikaisemmin määritelty REST-rajapinnasta haettavat pisteet sekä kunkin pisteen yksilöllinen tunnus. Ohjelmakoodiin tehtiin tarvittava muutos, joka mahdollisti tiedostosta löytyvien tietojen hakemisen ja niiden yhdistämisen kuhunkin pisteeseen kyselyjen yhteydessä. Tämä muutos ei ainoastaan ratkaissut alkuperäistä ongelmaa, vaan myös tehosti tiedonkäsittelyä, sillä kaikki tarvittavat tiedot olivat nyt keskitetysti yhdessä paikassa. Kuvassa 7 on kuvattu ohjelmakoodiin tulostus, jossa on yhdistetty kaikki pisteen tiedot.

```
2024-04-25 18:26:24#302.TE16.3.011.1#21.277778625488281#°C#FIACAEAEAAAAAC
2024-04-25 18:26:24#302.QE16.3.011.1#409#ppm#FIACAEAEAAAAAC
2024-04-25 18:26:24#302.TE16.3.011.2#21.888889312744141#°C#FIACAEAEAAAAAC
2024-04-25 18:26:24#302.QE16.3.011.2#409#ppm#FIACAEAEAAAAAC
2024-04-25 18:26:24#302.TE16.03.012#22.277776718139648#°C#FIACAEAEAAAAAC
2024-04-25 18:26:24#302.QE16.03.012#424#ppm#FIACAEAEAAAAAC
2024-04-25 18:26:24#302.TE16.03.013#22.722223281860352#°C#FIACAEAEAAAAAC
2024-04-25 18:26:24#302.QE16.03.013#410#ppm#FIACAEAEAAAAAC
2024-04-25 18:26:24#302.TE16.03.015#21.500001907348633#°C#FIACAEAEAAAAAC
2024-04-25 18:26:24#302.QE16.03.015#416#ppm#FIACAEAEAAAAAC
2024-04-25 18:26:24#302.TE16.03.016#21.388889312744141#°C#FIACAEAEAAAAAC
2024-04-25 18:26:24#302.QE16.03.016#406#ppm#FIACAEAEAAAAAC
2024-04-25 18:26:24#302.TE16.3030#22.659999847412109#°C#FIACAEAEAAAAAC
2024-04-25 18:26:25#301.TE16.1004#21.111110687255859#°C#FIACAEAJACAAAA
2024-04-25 18:26:25#301.QE16.1004#409#ppm#FIACAEAJACAAAA
2024-04-25 18:26:25#301.TE16.1005#20.944446563720703#°C#FIACAEAJACAAAA
2024-04-25 18:26:25#301.TE16.1006#21.277778625488281#°C#FIACAEAJACAAAA
2024-04-25 18:26:25#301.QE16.1006#402#ppm#FIACAEAJACAAAA
2024-04-25 18:26:25#301.TE16.1007#20.888887405395508#°C#FIACAEAJACAAAA
2024-04-25 18:26:25#301.QE16.1007#405#ppm#FIACAEAJACAAAA
```

Kuva 7. Ohjelmakoodiin tulostus

Ohjelmakoodia tehdessä huomattiin, että vastauksen saaminen REST-rajapinnasta saattoi ajoittain kestää poikkeuksellisen pitkään. Tämä ongelma ratkaistiin lisäämällä ohjelmakoodiin aikakatkaisutoiminto, joka esti ohjelmakoodin jumiutumisen yhteen kohtaan liian pitkäksi aikaa. Ongelma johtui palveluntarjoajan REST-rajapinnan hitaudesta ja tätä hitautta esiintyi aina ajoittain. Ohjelmakoodi kirjoittaa tällaisesta aikakatkaisusta virheilmoituksen erilliseen tiedostoon, josta on helppo seurata REST-rajapinnan toimintaa ja sen ajoittaista hitautta.

Ohjelmakoodiin lisättiin ominaisuus, joka vertailee haettavien pisteiden määrää REST-rajapinnasta saatuihin pisteiden määrään. Tämä on kuvattu kuvio 1:ssä ”Tarkista haettavien pisteiden määrä” kohdassa. Tällä haluttiin varmistaa, että oli saatu REST-rajapinnasta vastaukset kaikille haettaville pisteille, jotka oli määritetty haettavaksi erillisessä tiedostossa. Mikäli pisteiden määrissä ilmenee poikkeamia, ohjelmakoodi kirjoittaa tästä virheilmoituksen erilliseen tiedostoon, joka helpottaa ongelman paikantamista ja ymmärtämistä, missä haettavassa pisteessä ongelma on.

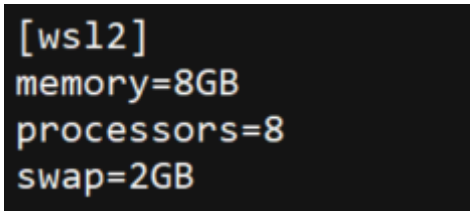
Mahdollisimman moneen kohtaan ohjelmakoodia tehtiin aina odottamattoman tilanteen sattuessa virheilmoituksien kirjoitus erilliseen tiedostoon. Tämä mahdollistaa ohjelmakoodin ja REST-rajapinnan helpon seurannan virheiden havaitsemiseksi testaus vaiheessa sekä myöhemmin lopullisessa tuotantoympäristössä. Tämä avulla pystyttiin tarkkailemaan järjestelmän toimintaa ja ratkaisemaan mahdolliset ongelmat nopeasti ja tehokkaasti. Tätä varten kirjoitettiin oma Python kirjasto.

5.2 Konttiteknologiaympäristö

5.2.1 Docker Desktop asennus ja käyttöönotto

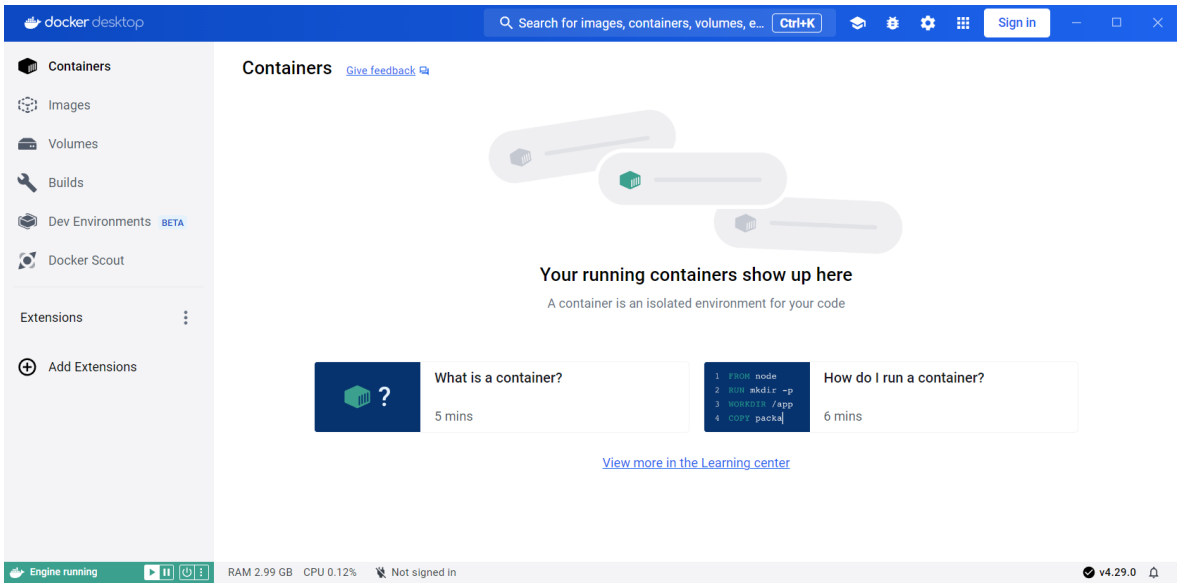
Docker Desktop asennettiin paikallisesti omalle tietokoneelle ja asennus aloitettiin selvittämällä laitteistovaatimukset Dockerin viralliselta sivustolta. Windows 10 käyttöjärjestelmälle vaatimuksena on, että laitteiston tulee olla 64-bittinen sekä Windows 10 version tulee olla vähintään 21H2. Keskusmuistia tulee olla vähintään 4GB sekä virtualisoinnin tuki tulee olla otettu käyttöön BIOS-asetuksista. (Docker docs 2024i.)

Dockerin viralliselta sivustolta ladattiin Docker Desktopin asennuspaketti. Latauksen yhteydessä varmistettiin, että asennuspaketti ladattiin oikealle käyttöjärjestelmälle eli tässä tapauksessa asennuspaketti ladattiin Windows käyttöjärjestelmälle. Asennuksen jälkeen Docker Desktop käyttöönotettiin määrittelemällä Docker Desktopille käytettävät resurssit, kuten muistin määrä, prosessorien määrä ja swap-muistin määrä. Nämä määriteltiin erillisestä asetustiedostosta kuvan 8 mukaisesti.



Kuva 8. Docker Desktop resurssien määritys

Asennuksen ja resurssien määrityksen jälkeen Docker Desktop käynnistetään Docker Desktop käynnistyskuvakkeesta. Kuvassa 9 on kuvattu Docker Desktop aloitusnäkymä, kun Docker Desktop on käynnistetty Windows 10 käyttöjärjestelmässä. Docker Desktop aloitusnäkymän vasemmasta reunasta löytyy keskeiset navigointi kohdat muun muassa kontit, imaget, volumes ja lisäosa kokonaisuuksiin. Aloitusnäkymän yläpalkista löytyy imageiden hakutoiminto sekä asetukset kuvake, josta pääsee määrittelemään Docker Desktop asetuksia.



Kuva 9. Docker Desktop aloitusnäkymä

5.2.2 Konttiteknologiaympäristön pystyttäminen

Konttiteknologiaympäristön pystytys paikallisesti omalle tietokoneelle aloitettiin luomalla Dockerfile, kuvan 10 mukaisesti. Dockerfilen komentojen avulla rakennettiin Docker Image, joka sisälsi kaikki ohjelmakoodin suorittamiseen tarvittavat tiedostot ja riippuvuudet. Kun Docker Image suoritettiin, siitä luotiin Docker kontti, joka mahdollisti ohjelmakoodin suorittamisen konttiteknologiaympäristössä.

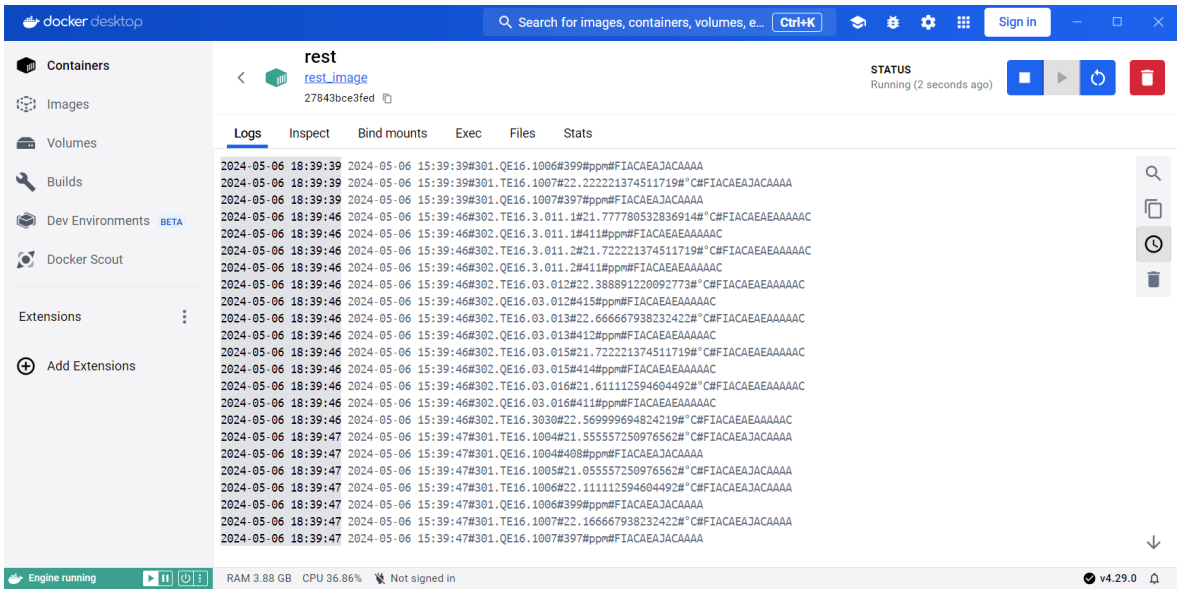
```
# Python.
FROM python:3.11

# Tiedot.
LABEL maintainer="Esa Suhonen <esa.suhonen@student.lab.fi>"

# Päivitykset ja asennukset.
RUN apt-get update && apt-get -y --no-install-recommends install \
nano \
net-tools \
iputils-ping && \
python -m pip install --upgrade pip \
pip install requests==2.31.0 \
pip install mysql-connector-python

# Työskentely hakemisto.
WORKDIR /home/projects

# Kopioidaan kansiot.
COPY /projects/app/ /home/projects

# Käynnistetään ohjelmakoodi.
CMD ["python","/home/projects/request.py"]
```

Kuva 10. Dockerfile pääohjelmakoodin kontista

5.2.3 Ohjelmakoodin suorittaminen konttiteknologiaympäristössä

Konttiteknologiaympäristön pystyttämisen jälkeen ohjelmakoodin toiminta testattiin konttiteknologiaympäristössä. Testauksen aikana havaittiin, että ohjelmakoodi ei käynnistynyt oikein suoritettavassa kontissa. Tämä johtui siitä, että suoritettavassa kontissa ei ollut Pythonin Requests kirjastoa asennettuna. Requests kirjaston tuli olla asennettuna konttiin Pythonin PIP paketinhallintatyökalun avulla käyttämällä ”python -m pip install --upgrade pip” ja ” pip install requests==2.31.0” komentoja, kuten kuvassa 10 on esitetty.

Tämän korjauksen jälkeen suoritettiin perusteellinen testaus, jotta voitiin varmistua, että ohjelmakoodi toimi odotetusti konttiteknologiaympäristössä. Tarkistettiin, että kaikki tarvittavat tiedostot ja riippuvuudet asentuivat oikein konttiin ja ohjelma käynnistyi ilman ongelmia. Kuvasta 11 nähdään selkeästi, että ohjelmakoodi toimii odotetusti konttiteknologiaympäristössä ja konttiteknologiaympäristö on määritetty oikein.



Kuva 11. Ohjelmakoodin suoritus konttiteknologiaympäristössä

5.3 Tietokanta

5.3.1 Tietokannan valinta

Kun saatiin ohjelmakoodi toimimaan konttiteknologiaympäristössä, tuli tehdä päätös käytettävästä tietokannasta. Tietokannan valinta on tärkeä vaihe, koska tietokanta tulee toimimaan keskeisessä osassa koko järjestelmää, datan varastointi paikkana. Perusteellisen harkinnan jälkeen päädyttiin valitsemaan MySQL-tietokanta.

MySQL-tietokanta valittiin sen monipuolisuuden ja laajan käytettävyyden vuoksi. MySQL-tietokanta tarjoaa selkeät ja helpot tietokantakäskyt, jotka ovat yleisiä ja tuttuja monille entuudestaan. Tämä mahdollistaa, että tietokannan tietojen hakeminen, lisääminen, päivittäminen, poistaminen ja muut toiminnot ovat helposti ja nopeasti omaksuttavissa myös muiden käyttäjien toimesta.

5.3.2 MySQL-tietokanta

MySQL-tietokanta on yksi suosituimmista avoimen lähdekoodin relaatiotietokannan hallintajärjestelmistä maailmassa. MySQL-tietokannan kehitti ruotsalainen yritys MySQL AB vuonna 1995. Kehitystyöstä vastasivat Michael Widenius (Monty), David Axmark ja Allan Larsson. Tavoitteena oli tarjota koti- ja ammattikäyttäjille tehokkaat ja luotettavat datanhallintamahdollisuudet. (Rieuf 2016.) MySQL-tietokanta on nimetty perustajajäsen Michael Wideniuksen tyttären My:n mukaan (MySQL 2024b).

MySQL-tietokanta on suunniteltu erittäin nopeaksi, monisäikeiseksi ja usean käyttäjän käytettäväksi tietokannaksi. MySQL-tietokanta sopii kriittisiin ja voimakkaasti kuormitettuihin tuotantojärjestelmiin sekä MySQL-tietokanta voidaan integroida laajasti käytettyihin ohjelmistoihin. (MySQL 2024a.) MySQL-tietokannoissa dataa säilytetään erillisissä taulukoissa, mikä mahdollistaa nopean ja tehokkaan tietojen käsittelyn. MySQL-tietokanta käyttää SQL-kieltä tiedon lisäämiseen, hakemiseen ja käsittelyyn. (MySQL 2024d.)

MySQL-tietokanta on kirjoitettu C ja C++ kielillä ja testattu monilla eri kääntäjillä. MySQL-tietokanta toimii Linux, Mac ja Windows pohjaisissa käyttöjärjestelmissä. MySQL-tietokanta on testattu muistivuotojen varalta kaupallisella Purify työkalulla sekä GPL lisensoidulla Valgrindilla. MySQL-tietokanta käyttää monikerroksista palvelinrakennetta, jossa on itsenäisiä moduuleja. MySQL-tietokanta osaa hyödyntää useita suorittimia, jos niitä on saatavilla. (MySQL 2024c.)

MySQL-tietokannalla on laaja suosio ja aktiivinen käyttäjäyhteisö. MySQL-tietokannasta on saatavilla hyvät dokumentaatiot, mikä tekee MySQL-tietokannan käyttämisestä ja hallinnasta sujuvaa ja vaivatonta. MySQL-tietokanta tarjoaa vahvan perustan opinnäytetyön vaatimille teknisille ominaisuuksille ja mahdollistaa sujuvan kehitysprosessin sekä lopputuloksen.

5.3.3 Tietokannan pystyttäminen konttiteknologiaympäristöön

Konttiteknologiaympäristöön pystytettiin paikallisesti myös toinen kontti, jonne määritettiin MySQL-tietokanta. Tämä tehtiin luomalla Dockerfile, kuvan 12 mukaisesti. Dockerfilen komentojen avulla rakennettiin Docker Image, joka suorittamalla luotiin MySQL-tietokanta Docker kontti. Tämä MySQL-tietokantakontti toimi REST-rajapinnasta saatavan datan varastointi paikkakana. Lisäksi luotiin erillinen .env tiedosto, jonne määritettiin tarvittavat ympäristömuuttujat MySQL-tietokantaa varten.

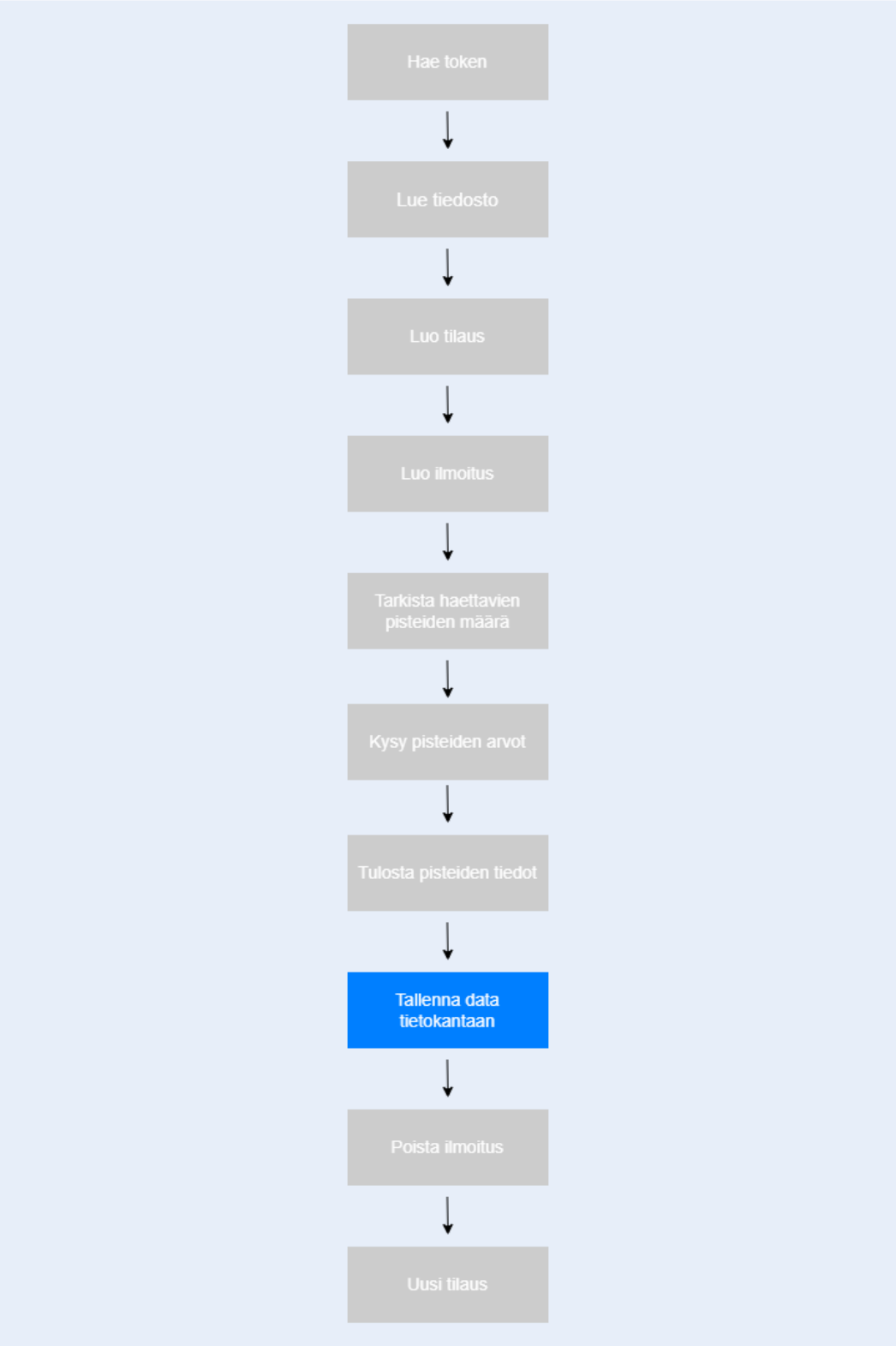
```
1  # MySQL.
2  FROM mysql:8.3.0
3
4  # Tiedot.
5  LABEL maintainer="Esa Suhonen <esa.suhonen@student.lab.fi>"
6
7  # Ilmoitetaan mitkä portit on tarkoitus julkaista ulospäin.
8  EXPOSE 3306
```

Kuva 12. Dockerfile MySQL-tietokantakontista

Kun MySQL-tietokantakontti oli saatu käyntiin, kirjauduttiin MySQL-tietokantakehotteeseen. MySQL-tietokantakehotteessa luotiin tietokanta nimeltä data ja tähän tietokantaan luotiin

taulu nimeltä measurements. Tauluun määritettiin id, timestamp, point_name, point_value, point_quality ja point_measurement_id kentät, joihin REST-rajapinnasta saatava data tallennettiin.

Seuraavana kirjoitettiin erillinen ohjelmakoodi, jolla testattiin pääohjelmakoodin kontista, että Python-ohjelmointikoodilla voidaan kirjoittaa testidataa MySQL-tietokantakontissa olevaan tietokantaan ja sinne tehtyyn tauluun. Kun oli todettu, että MySQL-tietokantaan kirjoitus onnistuu, lisättiin tämä MySQL-tietokantaan datan kirjoitus kokonaisuus myös varsinaiseen REST-rajapintaohjelmakoodiin. Kuviossa 2 on kuvattu päivitetty työnkulun prosessikaavio, josta nähdään datan tallennus tietokantaan kokonaisuus ja mihin vaiheeseen prosessia tämä kokonaisuus sijoittuu. Samalla varmistettiin, että data tallentuu oikein ja oikeaan muotoon MySQL-tietokannan taulussa oleviin kenttiin. Tämän jälkeen todettiin, että datan tallentaminen tietokantaa sujuu moitteettomasti.



Kuvio 2. Työnkulun prosessikaavio, jossa päivitetty tietokanta kokonaisuus

5.4 Ohjelmakoodin testaus konttiteknologiaympäristössä

Testausta varten pystytettiin konttiteknologiaympäristö paikallisesti omalle tietokoneelle aiempien vaiheiden mukaisesti. Ympäristön pystytys vaati jonkin verran ylimääräistä työtä,

mutta tämä osoittautui lopulta hyväksi ratkaisuksi. Tällainen lähestymistapa tarjosi mahdollisuuden testata ja varmistaa ohjelmakoodin toiminta turvallisesti, ennen sen myöhempää käyttöönottoa tuotantoympäristössä.

Testien suorittaminen paikallisesti omalla tietokoneella ennen tuotantoympäristöön siirtämistä oli tärkeä vaihe. Suorittamalla testit paikallisesti omalla tietokoneella ennen ohjelmakoodin tuotantoympäristöön siirtämistä, varmistettiin ohjelmakoodin vakaus ja luotettavuus. Tämä lähestymistapa mahdollisti ongelmien havaitsemisen ja korjaamisen ennen kuin ne olisivat voineet vaikuttaa tuotannossa oleviin toimintoihin. Testaus paikallisesti omalla tietokoneella ennen tuotantoympäristöön siirtymistä auttaa varmistamaan sujuvan ja virheettömän käyttöönoton myöhemmin tuotantoympäristössä.

Testatessa esille tuli, että REST-rajapinnan palveluntarjoaja oli saattanut poistaa julkaistuja pisteitä rajapinnasta ilman, että oli muistanut informoida osapuolia pisteiden poistoista. Pisteiden poisto johti siihen, että ohjelmakoodissa tapahtui odottamaton virhe, joka esti tästä pisteestä eteenpäin olevien pisteiden käsittelyn. Tämä ongelma ratkaistiin lisäämällä ohjelmakoodiin ehtolause, joka ohitti tällaiset virheet ja jatkoi muiden pisteiden osalta toimintaa normaalisti. Tällaisesta tapahtumasta ohjelmakoodi kirjoittaa virheilmoituksen erilliseen tiedostoon, josta on helppo seurata ohjelmakoodin toimintaa. Kuvassa 13 on kuvattu ohjelmakoodiin lisätty ehtolause kokonaisuus.

```
518  # Käydään Measurements avaimet läpi.
519  for key in Measurements.keys():
520      # Tarkistetaan, että "key" löytyy "nameValuePairs" sanakirjasta.
521      if key in nameValuePairs:
522          # Tarkistetaan, että ID määritys on oikeanlainen.
523          if checkID(Measurements[key]["info"]):
524              # Jos pisteellä on sanakirjassa arvo.
525              if nameValuePairs[key] is not None:
526                  # Jos pisteellä ei ole laatua.
527                  if Measurements[key]["unit"] == "" or Measurements[key]["unit"].lower() == "nan":
528                      Measurements[key]["unit"] = "NaN"
529                  else:
530                      pass
531                  # Lisätään aikaleima.
532                  export = timestamp + "#" + Measurements[key]["description"] + "#" + nameValuePairs[key] + "#" + \
533                  Measurements[key]["unit"] + "#" + Measurements[key]["info"]
534                  print(export)
535              else:
536                  # Kirjoitetaan logia tiedostoon.
537                  my_logging.writeLog("Value is none: " + key + ";" + Measurements[key]["description"] + ";" + \
538                                      Measurements[key]["unit"] + ";" + Measurements[key]["info"])
539          else:
540              # Kirjoitetaan logia tiedostoon.
541              my_logging.writeLog("Check ID Error: " + key + ";" + Measurements[key]["description"] + ";" + \
542                                  Measurements[key]["unit"] + ";" + Measurements[key]["info"])
543      else:
544          # Kirjoitetaan logia tiedostoon.
545          my_logging.writeLog("Point not found: " + key + ";" + Measurements[key]["description"] + ";" + \
546                              Measurements[key]["unit"] + ";" + Measurements[key]["info"])
```

Kuva 13. Ohjelmakoodiin lisätty ehtolause

Testauksen aikana havaittiin, että REST-rajapinta saattoi palauttaa jostain syystä haettavan pisteen ilman arvoa, joka aiheutti odottamattoman tilanteen ohjelmakoodissa. Ongelman

ratkaisemiseksi ohjelmakoodia muutettiin siten, että ohjelmakoodi tarkkaili REST-rajapinnasta saatua palautusta ja varmisti, että palautuksessa oli pisteellä arvo. Mikäli kysyttävälle pisteellä ei ollut arvoa, ohjelmakoodi huomioi tämän ja ohitti kyseisen pisteen ja jatkoi seuraavaan pisteeseen. Ohjelmakoodi tallentaa tapahtumasta virheilmoituksen erilliseen tiedostoon, joka mahdollistaa ohjelmakoodin toiminnan helpon tarkkailun tämän ongelman osalta.

Testatessa havaittiin myös, että ohjelmakoodi ei hakenut kaikkia pisteitä onnistuneesti, vaikka nämä pisteet oli määritelty erillisessä tiedostossa haettaviksi. Ongelma osoittautui hankalaksi selvitettäväksi, sillä ohjelmakoodi toimi moitteettomasti pienillä haettavilla pistemäärillä. Tämä aiheutti haasteita, sillä kaikkien pisteiden odotettiin olevan saatavilla samanaikaisesti yhdellä REST-rajapinta kyselyllä, mutta kaikkia pisteitä ei saatu kuitenkaan kysyttyä kerralla. Ajatus siitä, että tätä varten olisi tarvinnut tehdä ylimääräinen kysely REST-rajapintaan, ei tuntunut järkevältä ratkaisulta. Ongelma saatiin lopulta ratkaistua, kun löydettiin REST-rapapinta kyselylle lisämääritys, joka mahdollisti suuremman pistemäärän hakemisen yhdellä REST-rajapinnan kyselyllä.

Näiden tehtyjen korjauksien jälkeen ohjelmakoodin annettiin pyöriä pystytetyssä konttiteknologiaympäristössä ja tarkkailtiin, että ohjelmakoodi sekä konttiteknologiaympäristö toimivat suunnitellusti. Ohjelmakoodin toimintaa seurattiin tätä varten tehdystä erillisestä tiedostosta, johon kaikki ohjelmakoodin virheilmoitukset tallentuivat automaattisesti. Lisäksi toimintaa tarkkailtiin MySQL-tietokantakontin avulla vertaamalla siellä olevan measurements taulun arvoja REST-rajapinnasta saatuun dataan. Näin varmistettiin, että data oli tietokannassa oikein ja vastasi todellisia arvoja. Tämä mahdollisti perusteellisen ohjelmakoodin toiminnan seuraamisen ja virheiden havaitsemisen, ennen kuin ohjelmisto siirrettäisiin myöhemmin tuotantoympäristöön. Kuvassa 14 on esitetty MySQL-tietokantakonttiin tallennettua dataa, joka on haettu REST-rajapinnan kautta.

```
mysql> SELECT * FROM measurements ORDER BY id DESC LIMIT 20;
+-----+---------------------+------------------+-------------+---------------+----------------------+
| id  | timestamp           | point_name       | point_value | point_quality | point_measurement_id |
+-----+---------------------+------------------+-------------+---------------+----------------------+
| 936 | 2024-05-28 17:14:11 | 301.QE16.1007    |      397.00 | ppm           | FIACAEAJACAAAA       |
| 935 | 2024-05-28 17:14:11 | 301.TE16.1007    |       23.56 | °C            | FIACAEAJACAAAA       |
| 934 | 2024-05-28 17:14:11 | 301.QE16.1006    |      388.00 | ppm           | FIACAEAJACAAAA       |
| 933 | 2024-05-28 17:14:11 | 301.TE16.1006    |       24.06 | °C            | FIACAEAJACAAAA       |
| 932 | 2024-05-28 17:14:11 | 301.TE16.1005    |       23.28 | °C            | FIACAEAJACAAAA       |
| 931 | 2024-05-28 17:14:11 | 301.QE16.1004    |      392.00 | ppm           | FIACAEAJACAAAA       |
| 930 | 2024-05-28 17:14:11 | 301.TE16.1004    |       23.28 | °C            | FIACAEAJACAAAA       |
| 929 | 2024-05-28 17:14:10 | 302.TE16.3030    |       23.96 | °C            | FIACAEAEAAAAAC       |
| 928 | 2024-05-28 17:14:10 | 302.QE16.03.016  |      403.00 | ppm           | FIACAEAEAAAAAC       |
| 927 | 2024-05-28 17:14:10 | 302.TE16.03.016  |       24.11 | °C            | FIACAEAEAAAAAC       |
| 926 | 2024-05-28 17:14:10 | 302.QE16.03.015  |      402.00 | ppm           | FIACAEAEAAAAAC       |
| 925 | 2024-05-28 17:14:10 | 302.TE16.03.015  |       23.89 | °C            | FIACAEAEAAAAAC       |
| 924 | 2024-05-28 17:14:10 | 302.QE16.03.013  |      409.00 | ppm           | FIACAEAEAAAAAC       |
| 923 | 2024-05-28 17:14:10 | 302.TE16.03.013  |       24.28 | °C            | FIACAEAEAAAAAC       |
| 922 | 2024-05-28 17:14:10 | 302.QE16.03.012  |      422.00 | ppm           | FIACAEAEAAAAAC       |
| 921 | 2024-05-28 17:14:10 | 302.TE16.03.012  |       24.28 | °C            | FIACAEAEAAAAAC       |
| 920 | 2024-05-28 17:14:10 | 302.QE16.3.011.2 |      402.00 | ppm           | FIACAEAEAAAAAC       |
| 919 | 2024-05-28 17:14:10 | 302.TE16.3.011.2 |       23.83 | °C            | FIACAEAEAAAAAC       |
| 918 | 2024-05-28 17:14:10 | 302.QE16.3.011.1 |      407.00 | ppm           | FIACAEAEAAAAAC       |
| 917 | 2024-05-28 17:14:10 | 302.TE16.3.011.1 |       23.89 | °C            | FIACAEAEAAAAAC       |
+-----+---------------------+------------------+-------------+---------------+----------------------+
20 rows in set (0.00 sec)
```

Kuva 14. MySQL-tietokantakonttiin tallennettua dataa

6 Yhteenveto ja pohdinta

Opinnäytetyön tarkoituksena oli pilotoida REST-rajapinnan toimintaa konttiteknologiaympäristössä. Keskeisenä tavoitteena oli selvittää, kuinka hyvin konttiteknologiaympäristö soveltuu ohjelmakoodin suorittamiseen sekä soveltuuko konttiteknologiaympäristö REST-rajapinta kyselyjen tekemiseen ohjelmakoodilla. Opinnäytetyön tavoitteena oli myös arvioida, miten konttiteknologiaympäristö soveltuu hyödynnettäväksi tuotantoympäristössä.

Opinnäytetyön olennainen osa oli konttiteknologiaympäristön pystyttäminen, joka mahdollisti ohjelmakoodin suorittamisen konttiteknologiaympäristössä. Tämä vaati huolellista suunnittelua ja konfigurointia, jolla varmistettiin, että konttiteknologiaympäristö toimi odotetusti ja tarjosi oikeanlaisen ympäristön ohjelmakoodin suorittamiselle. Lisäksi olennainen osa tätä prosessia oli myös ohjelmakoodin kehittäminen, joka mahdollisti kyselyjen tekemisen REST-rajapintaan ohjelmakoodilla. Tämä edellytti tarkkaa ymmärrystä REST-arkkitehtuurista ja sen käytännön toteutuksesta sekä kykyä luoda toimiva ja tehokas koodi, joka kykenee kommunikoimaan REST-rajapinnan kanssa. Näiden REST-rajapinta kyselyjen avulla oli mahdollista testata konttiteknologiaympäristön ja ohjelmakoodin toimivuutta ja varmistaa kokonaisuuden toiminta.

Opinnäytetyössä oleva testausvaihe oli tärkeä osa opinnäytetyön kokonaisuutta, sillä se mahdollisti perusteellisen ohjelmakoodin ja konttiteknologiaympäristön testauksen. Testausvaihe mahdollisti ohjelmakoodin muokkaamisen sellaiseksi, että datan siirto toimi luotettavasti kaikissa tilanteissa ja kommunikointi REST-rajapinnan kanssa toimi oikein ja tehokkaasti, jotka olivat tämän opinnäytetyön kannalta erityisen tärkeitä kokonaisuuksia. Testausvaiheessa tuli esille erilaiset ohjelmakoodiin tehtävät muutostarpeet, joita ei välttämättä olisi muuten havaittu. Testausvaihe paljasti ennakoimattomat ongelmat, jotka olivat olennaisia ohjelmakoodin ja konttiteknologiaympäristön toimivuuden kannalta.

Kokonaisuudessaan konttiteknologiaympäristön pystyttäminen ja ohjelmakoodin kehittäminen sekä tämän kokonaisuuden testaus olivat olennaisia vaiheita tässä opinnäytetyössä. Nämä vaiheet mahdollistivat tutkimuksen tavoitteiden saavuttamisen ja vastasivat monipuolisesti tutkimuskysymyksiin. Tämän avulla saatiin syvällinen ymmärrys siitä, kuinka konttiteknologiaympäristö soveltuu REST-rajapinnan kyselyjen tekemiseen ohjelmakoodilla ja mitkä tekijät vaikuttavat tähän kokonaisuuteen.

Ensimmäisenä tutkimuskysymyksenä oli, kuinka konttiteknologiaympäristö soveltuu ohjelmakoodin suorittamiseen. Tähän vastauksena saatiin konttiteknologiaympäristöä pilotoitaessa, että konttiteknologiaympäristö tarjoaa erinomaisen ympäristön ohjelmakoodin suorittamiseen. Ohjelmakoodia voidaan suorittaa konttiteknologiaympäristössä lähes samalla

tavalla, kuin sitä suoritettaisiin jossain muussa ympäristössä. Erona on, että Docker kontteja varten tulee pystyttää Docker ympäristö, jossa Docker kontteja voidaan ajaa. Docker konttiin paketoidaan suoritettava ohjelmakoodi, ohjelmakoodin riippuvuudet, ympäristömuuttujat ja tarvittavat muut määritykset. Docker kontti luodaan Docker Imagesta, joka luodaan Dockerfile tiedoston avulla. Docker konttien etuna on, että ne ovat helposti siirrettävissä ja skaalattavissa eri alustoilla.

Lisäksi konttiteknologiaympäristö mahdollistaa ohjelmakoodin eristämisen, mikä parantaa järjestelmän turvallisuutta ja luotettavuutta. Tämä eristäminen tarkoittaa, että ohjelmakoodin suorittaminen yhdessä kontissa ei vaikuta muiden konttien toimintaan, mikä vähentää riippuvuuksien ja versio-ongelmien riskiä. Eristäminen helpottaa myös vianmääritystä ja ylläpitoa, koska jokainen kontti toimii omassa eristetyssä ympäristössä ja mahdolliset ongelmat voidaan rajata tiettyyn konttiin ilman, että ongelmat vaikuttavat koko järjestelmän toimintaan.

Konttiteknologiaympäristö käyttää myös vähemmän resursseja ja parantaa suorituskykyä. Koska kontit jakavat käyttöjärjestelmän ytimen, ne ovat kevyempiä ja käynnistyvät nopeammin kuin perinteiset virtuaalikoneet. Konttiteknologiaympäristö mahdollistaa tehokkaan resurssien käytön ja vähentää tietojärjestelmien ja ohjelmistojen ylläpitämiseen tarvittavia investointeja ja henkilöstökuluja. Konttiteknologiaympäristön avulla voidaan ajaa useita instansseja samasta sovelluksesta rinnakkain, mikä parantaa sovelluksen käytettävyyttä ja suorituskykyä, joka korostuu erityisesti palveluiden kuormitustilanteissa.

Näin ollen voidaan todeta, että konttiteknologiaympäristö tarjoaa monia etuja ohjelmakoodin suorittamiseen. Näitä etuja on siirrettävyys, skaalautuvuus, eristäminen ja resurssien tehokas hyödyntäminen. Näiden etujen ansiosta konttiteknologiaympäristö soveltuu erinomaisesti erilaisiin sovellusympäristöihin ja auttaa parantamaan ohjelmakoodin tekemistä ja testausta sekä valmiin ohjelmakoodin käyttöönottoa ja ylläpitoa.

Toisena tutkimuskysymyksenä oli, kuinka konttiteknologiaympäristö soveltuu REST-rajapinnan kyselyjen tekemiseen ohjelmakoodilla. Tähän vastauksena saatiin, että konttiteknologiaympäristö soveltuu erinomaiseksi ratkaisuksi REST-rajapinnan kyselyjen tekemiseen ohjelmakoodilla, ilman merkittäviä haasteita. REST-rajapinnan kyselyjen tekemistä ohjelmakoodilla pilotoitaessa huomattiin, että ohjelmakoodi toimi konttiteknologiaympäristössä samalla tavalla, kuin ohjelmakoodia suoritettaisiin paikallisesti omalla tietokoneella.

Konttiteknologiaympäristön lisäksi olisi ollut useita muita vaihtoehtoja REST-rajapinnan kyselyjen tekemiseen ohjelmakoodilla. Näitä vaihtoehtoja olisi ollut muun muassa virtuaalikoneet, fyysiset palvelimet ilman virtualisointia sekä pilvipalvelut. Jokaisella näistä

vaihtoehdoista on omat hyvät ja huonot puolet, jotka tulee arvioida huolellisesti aina tapauskohtaisesti jokaisen tehtävän kokonaisuuden osalta.

Virtuaalikoneet tarjoavat korkean eristyneisyyden, mikä lisää tietoturvaa. Lisäksi virtuaalikoneet mahdollistavat useiden eri käyttöjärjestelmien ajamisen samalla laitteistolla. Virtuaalikoneet kuitenkin ovat raskaampia kuin konttiteknologiaympäristön ratkaisut, koska vaativat enemmän resursseja, kuten muistia, prosessoritehoa ja tallennustilaa. Virtuaalikoneiden käynnistysaika on hitaampi verrattuna konttiteknologiaympäristöön ja ylläpito on yleensä myös monimutkaisempaa, kuin konttiteknologiaympäristön ylläpito.

Fyysiset palvelimet ilman virtualisointia mahdollistavat suoritustehon tarjoamisen ilman virtualisointikerrosta ja mahdollistavat näin ollen täyden laitteiston resurssien käytön. Fyysisien palvelimien huonona puolena on, että ne skaalautuvat huonosti sekä niiden kyky mukautua nopeasti ja tehokkaasti muuttuviin tarpeisiin ja olosuhteisiin on heikkoa. Fyysiset palvelimet eivät ole joustavia, koska jokainen palvelin on kiinteästi sidottu tiettyyn laitteistoon ja käyttöjärjestelmään, mikä tekee resurssien hallinnasta ja optimoinnista monimutkaista ja hidasta. Fyysiset palvelimet vaativat enemmän manuaalista konfigurointia ja ylläpitoa sekä eivät tarjoa yhtä hyvää eristyneisyyttä kuin virtualisointiteknologiat.

Pilvipalvelut mahdollistavat yksinkertaisen sovellusten käyttöönoton ja hallinnan tarjoamalla valmiita ympäristöjä erilaisiin tarpeisiin. Pilvipalvelut tarjoavat myös hyvän integraation muiden pilvipalveluiden kanssa. Näiden etujen rinnalla on kuitenkin palveluntarjoajaan ”lukkiutumisen” riski, joka tarkoittaa sitä, että vaihtoehtoisia vaihtoehtoja ei ole halutessa saatavilla tai vaihtaminen toiseen pilvipalveluun on hankalaa tai kallista. Konttiteknologiaympäristöön verrattuna pilvipalveluissa on vähemmän joustavuutta eli järjestelmien mukauttaminen omiin tarpeisiin voi olla rajallista verrattuna paikallisesti asennettuihin järjestelmiin. Lisäksi pilvipalvelun käyttäjän on luotettava siihen, että pilvipalveluntarjoaja huolehtii palvelun toimivuudesta ja turvallisuudesta, eikä käyttäjällä itsellään välttämättä ole täyttä valvontaa näissä asioissa. Myös kustannukset pilvipalveluiden käytössä voivat nousta, jotka riippuvat siitä, miten paljon palveluita käytetään.

Näiden vaihtoehtojen tarkastelu osoittaa, että jokaisella vaihtoehdolla on omat hyvät ja huonot puolet. Konttiteknologiaympäristö erottuu erityisesti sen keveyden, nopeuden ja skaalautuvuuden ansiosta edukseen. Konttiteknologiaympäristö tarjoaa hyvän tasapainon suorituskyvyn, hallittavuuden ja eristyneisyyden välillä, mikä tekee siitä hyvän vaihtoehdon useissa tapauksissa. Erityisesti kun tarvitaan nopeaa ja joustavaa sovelluskehitystä ja käyttöönottoa, konttiteknologiaympäristö tarjoaa merkittäviä etuja. Koska kontit ovat riippumattomia alustasta ja ympäristöstä, sovelluksia voidaan kehittää ja testata paikallisesti ja siirtää sitten saumattomasti eri ympäristöihin. Konttiteknologiaympäristö mahdollistaa sovellusten

nopean käynnistyksen ja sammuttamisen, mikä auttaa optimoimaan resurssien käytön ja vähentämään hukkaan meneviä resursseja. Lisäksi konttiteknologiaympäristön avulla sovelluksia voidaan skaalata automaattisesti vastaamaan vaihtelevia käyttäjämääriä ja kuormia, mikä takaa joustavan ja tehokkaan suorituskyvyn kaikissa tilanteissa. Näin ollen konttiteknologia tarjoaa vankan perustan nopealle ja joustavalle sovelluskehitykselle ja käyttöönotolle, mikä on erityisen tärkeää nykypäivän dynaamisessa ja nopeasti muuttuvassa toimintaympäristössä. Lisäksi kehittämistiimin yksikössä oli aikaisemmin linjattu, että kaikki kehitystiimin yksikön kehitystyöt ja niihin liittyvät kokonaisuudet tullaan toteuttamaan konttiteknologiaympäristössä.

Kolmantena tutkimuskysymyksenä oli, miten konttiteknologiaympäristö soveltuu hyödynnettäväksi tuotantoympäristössä. Tähän vastauksena saatiin opinnäytetyön pilotoidun kokonaisuuden perusteella, että konttiteknologiaympäristö soveltuu erinomaisesti hyödynnettäväksi tuotantoympäristössä. Konttiteknologiaympäristö mahdollistaa ohjelmistojen ja niiden riippuvuuksien pakkaamisen yhdeksi kokonaisuudeksi, joka helpottaa sovellusten siirtämistä eri ympäristöjen välillä.

Konttiteknologiaympäristö mahdollistaa nopean ja luotettavan käyttöönoton, koska konttiteknologiaympäristön avulla kehittäjät voivat luoda kehitysympäristöjä, jotka jäljittelevät tarkasti tuotantoympäristöä. Tämä auttaa vähentämään tuotantoympäristössä ilmeneviä ongelmia ja lisää järjestelmän luotettavuutta. Konttiteknologiaympäristö mahdollistaa isäntäkoneen resurssien tehokkaan hyödyntämisen, koska kontit jakavat isäntäkoneen resurssit tehokkaasti. Konttiteknologiaympäristössä yhden kontin kaatuminen ei vaikuta suoraan muihin kontteihin, koska kontit ovat eristettyjä, jolloin järjestelmän luotettavuus myös paranee.

Lisäksi konttiteknologiaympäristön käyttöönotto ja sen hallinta on helppoa, sillä nykyaikaiset konttiteknologiat, kuten Kubernetes ja Docker tarjoavat tehokkaita työkaluja tähän tarkoitukseen. Nämä työkalut ovat hyvin dokumentoituja ja niillä on laaja yhteisötuki, mikä helpottaa niiden oppimista ja käyttöä. Esimerkiksi Kubernetesin avulla voidaan automatisoida sovellusten orkestrointi, mikä vähentää manuaalisen työn tarvetta ja mahdollistaa sovellusten nopean ja luotettavan käyttöönoton.

Opinnäytetyön tuloksien perusteella voidaan todeta, että konttiteknologiaympäristö tarjoaa tehokkaan ja joustavan ratkaisun REST-rajapintojen kanssa työskenneltäessä. Konttiteknologiaympäristö mahdollistaa ohjelmakoodin helpon, tehokkaan ja luotettavan suorittamisen, mikä tekee siitä erinomaisen vaihtoehdon myös tuotantoympäristöä ajateltaessa. Konttiteknologiaympäristössä ohjelmakoodin helppo suorittaminen perustuu siihen, että kontit

tarjoavat yhtenäisen ja eristetyn konttiteknologiaympäristön, jossa ohjelmakoodi voi toimia, vaikka taustajärjestelmät ja muut sovellukset muuttuisivat.

Konttiteknologiaympäristössä ohjelmakoodin tehokkuus tulee siitä, että kontit jakavat käyttöjärjestelmän resurssit ja mahdollistavat sen, että useita kontteja voidaan ajaa rinnakkain ilman, että ne kuluttavat merkittävästi ylimääräisiä resursseja. Tämä optimoi resurssien käytön ja parantaa suorituskykyä. Konttiteknologiaympäristössä ohjelmakoodin tehokkuus on erityisen tärkeää tuotantoympäristöissä, joissa korkea suorituskyky ja resurssien optimaalinen käyttö ovat merkittäviä tekijöitä.

Konttiteknologiaympäristössä ohjelmakoodin luotettavuus saavutetaan konttien kyvyllä eristää sovellukset toisistaan sekä muusta ympäristöstä, jossa niitä suoritetaan. Tämä vähentää riskejä, jotka liittyvät sovellusten keskinäisiin riippuvuuksiin ja ympäristön muutoksiin. Lisäksi kontit ovat helposti siirrettävissä ja toistettavissa, mikä tekee järjestelmästä vakaamman ja auttaa järjestelmää palautumaan nopeasti ongelmatilanteissa, tarjoten luotettavan toimintaympäristön tuotantoympäristöön.

Konttiteknologiaympäristön käyttö parantaa skaalautuvuutta, suorituskykyä ja ylläpidettävyyttä ohjelmistoprojekteissa, mikä luo vankan perustan nykyaikaiselle ohjelmistokehitykselle ja toiminnalle. Konttiteknologiaympäristö mahdollistaa ohjelmiston osien pakkaamisen yhtenäisiin ja itsenäisiin yksiköihin. Tämä mahdollistaa sovellusten joustavan skaalautumisen tarpeen mukaan lisäämällä tai vähentämällä kontteja. Näin ollen konttiteknologiaympäristössä sovellukset voivat mukautua joustavasti erilaisiin kuormitusvaatimuksiin ilman suurta järjestelmän muutostarvetta.

Suorituskyky parantuu konttiteknologiaympäristöä käytettäessä, koska kontit eristävät sovelluksen ja sen riippuvuudet muusta ympäristöstä. Lisäksi kontit toimivat kevyesti ja tehokkaasti omassa eristetyssä ympäristössään ilman tarvetta jakaa resursseja muiden sovellusten kanssa, joka parantaa merkittävästi suorituskykyä. Konttiteknologiaympäristö tarjoaa yhdenmukaisen ja toistettavan kehitys- ja tuotantoympäristön. Tämä helpottaa ohjelmistojen siirtämistä kehityksestä tuotantoon ja mahdollistaa nopeat ja luotettavat käyttöönotot. Lisäksi kontit ovat helppoja hallita ja päivittää, mikä tehostaa ylläpitotoimia ja vähentää mahdollisia virheitä ja ongelmia. Näin ollen kaikkien yläpuolen seikkojen perusteella konttiteknologiaympäristön käyttöä voidaan suositella tuotantoympäristössä käytettäväksi.

Opinnäytetyön tulokset tarjoavat arvokasta tietoa konttiteknologiaympäristöstä ja REST-rajapintojen kyselyjen toteuttamisesta ohjelmakoodin avulla. Havaitut lähestymistavat ja tehokkaat menetelmät tarjoavat arvokasta tietoa myös muille ohjelmistokehittäjille ja alan ammattilaisille, jotka työskentelevät REST-rajapintojen ja ohjelmakoodien kanssa konttiteknologiaympäristössä. Opinnäytetyössä pilotoitua konttiteknologiaympäristöä ja toteutettua

ohjelmakoodia, joka käsittelee REST-rajapintakyselyjä, voidaan hyödyntää myös muiden vastaavien kokonaisuuksien yhteydessä, joka mahdollistaa datan hankkimisen myös muista REST-rajapinnoista. Näin ollen tämän opinnäytetyön tulokset eivät pelkästään rajoitu tähän opinnäytetyöhön, vaan niitä voidaan soveltaa myös laajemmin.

Tutkimuksen tulosten jakaminen ja niiden hyödyntäminen muissa yhteyksissä edistävät alan tietotaitoa ja voivat auttaa parantamaan ohjelmistokehitysprosesseja sekä vauhdittamaan innovaatiota. Mahdollisuus hyödyntää opinnäytetyössä käytettyjä menetelmiä ja ratkaisuja muissa projekteissa lisää niiden arvoa ja vaikutusta alan käytänteisiin. Lisäksi avoimen tiedonjakamisen kautta voidaan edistää oppimista, mikä voi johtaa uusiin oivalluksiin ja parhaiden käytäntöjen leviämiseen.

Opinnäytetyön tulokset eivät vain tarjoa ymmärrystä tutkituista aiheista, vaan ne myös kannustavat jatkotutkimuksiin ja kehitystyöhön. Esimerkiksi opinnäytetyön havaintojen perusteella voidaan suunnitella jatkotutkimuksia, jotka syventävät ymmärrystä konttiteknologiaympäristöistä ja REST-rajapinnoista. Näin ollen opinnäytetyön tulokset eivät ainoastaan vastaa tutkimuskysymyksiin, vaan tulokset myös avaavat uusia mahdollisuuksia ja innostavat jatkokehitykseen alan ammattilaisten keskuudessa.

Jatkotoimenpiteenä on suunnitella ja toteuttaa tässä opinnäytetyössä pilotoitu konttiteknologiaympäristön kokonaisuuden siirtäminen tuotantoympäristöön. Tavoitteena on hyödyntää opittuja käytäntöjä ja ottaa konttiteknologiaympäristö osaksi varsinaista tuotantoympäristöä. Konttiteknologiaympäristön pystyttäminen tulee suunnitella ja toteuttaa siten, että kokonaisuuden pystyttäminen tuotantoympäristöön tapahtuu sujuvasti ja tehokkaasti, aiheuttamatta häiriötä nykyisiin olemassa oleviin toteutuksiin.

Tässä yhteydessä on olennaista ottaa huomioon isäntäkoneen resurssien suunnittelu, jotta konttiteknologiaympäristön käyttö olisi optimaalista tuotantoympäristössä. Lisäksi tulee suunnitella jatkuvan seurannan ja ylläpidon prosessit siten, että konttiteknologiaympäristö toimii optimaalisesti tuotantoympäristössä. Tämä sisältää esimerkiksi prosessien valvonnan, varmuuskopioinnin ja palautusprosessien varmistamisen.

Tämän kokonaisuuden siirtymisessä pilotointivaiheesta tuotantoympäristöön on otettava huomioon myös tekniset, että yrityksen sisäiset näkökulmat varmistaen, että siirto tapahtuu mahdollisimman sujuvasti. Tämä edellyttää tiivistä yhteistyötä eri sidosryhmien välillä ja huolellista suunnittelua kaikilla tasoilla. Tämän lisäksi on tärkeää varmistaa, että käyttöönottoon liittyvät koulutusresurssit, dokumentaatiot ja opetusmateriaalit ovat riittävät, jotta henkilöstö voi omaksua uudet käytännöt ja konttiteknologiaympäristön tehokkaasti.

Lisäksi on tärkeää huomioida, että konttiteknologiaympäristön siirtäminen tuotantoympäristöön voi vaatia muutoksia nykyisiin prosesseihin ja käytäntöihin. Onkin tärkeää varata riittävästi aikaa ja resursseja muutosten suunnitteluun ja seuraamiseen sekä käyttöönoton edistämiseen. Lisäksi on suositeltavaa tehdä kattava riskianalyysi ja varautua mahdollisiin ongelmiin ja haasteisiin siirtymävaiheessa. Kaiken kaikkiaan konttiteknologiaympäristön siirtäminen tuotantoympäristöön on kokonaisvaltainen prosessi, joka vaatii huolellista suunnittelua, yhteistyötä eri osapuolien välillä ja jatkuvaa seurantaa ja ylläpitoa, joilla varmistetaan onnistunut käyttöönotto ja hyvä toiminnallisuus.

Lähteet

Cloud Academy Team. 2023. Docker vs. Virtual Machines: Differences You Should Know. Blogikirjoitus. Cloud Academy. Viitattu 5.4.2024. Saatavissa https://cloudacademy.com/blog/docker-vs-virtual-machines-differences-you-should-know/

Docker. 2024a. Docker CLI. Viitattu 12.6.2024. Saatavissa https://www.docker.com/products/cli/

Docker. 2024b. Use containers to Build, Share and Run your applications. Viitattu 5.4.2024. Saatavissa https://www.docker.com/resources/what-container/

Docker docs. 2024a. Credential helper protocol. Viitattu 14.6.2024. Saatavissa https://docs.docker.com/reference/cli/docker/login/

Docker docs. 2024b. Credential stores. Viitattu 14.6.2024. Saatavissa https://docs.docker.com/reference/cli/docker/login/

Docker docs. 2024c. Content trust in Docker. Viitattu 13.6.2024. Saatavissa https://docs.docker.com/engine/security/trust/

Docker docs. 2024d. Docker Compose overview. Viitattu 13.6.2024. Saatavissa https://docs.docker.com/compose/

Docker docs. 2024e. Docker Engine overview. Viitattu 12.6.2024. Saatavissa https://docs.docker.com/engine/

Docker docs. 2024f. Docker overview. Viitattu 12.4.2024. Saatavissa https://docs.docker.com/get-started/overview/

Docker docs. 2024g. Docker Scout. Viitattu 12.6.2024. Saatavissa https://docs.docker.com/scout/

Docker docs. 2024h. Dockerfile reference. Viitattu 12.4.2024. Saatavissa https://docs.docker.com/reference/dockerfile/

Docker docs. 2024i. Install Docker Desktop on Windows. Viitattu 4.5.2024. Saatavissa https://docs.docker.com/desktop/install/windows-install/

Docker docs. 2024j. Overview of Docker Build. Viitattu 12.6.2024. Saatavissa https://docs.docker.com/build/

Docker docs. 2024k. Overview of Docker Desktop. Viitattu 12.4.2024. Saatavissa https://docs.docker.com/desktop/

Docker docs. 2024l. Overview of Docker Extensions. Viitattu 12.6.2024. Saatavissa https://docs.docker.com/desktop/extensions/

Docker docs. 2024m. Overview of Docker Hub. Viitattu 12.4.2024. Saatavissa https://docs.docker.com/docker-hub/

Docker docs. 2024n. Overview of the get started guide. Viitattu 12.4.2024. Saatavissa https://docs.docker.com/get-started/

Docker docs. 2024o. Use Docker Compose. Viitattu 12.4.2024. Saatavissa https://docs.docker.com/get-started/08_using_compose/

GitHub. 2024a. Docker credential helpers. Viitattu 14.6.2024. Saatavissa https://github.com/docker/docker-credential-helpers/

GitHub. 2024b. Kubernetes. Viitattu 14.6.2024. Saatavissa https://github.com/kubernetes/kubernetes/

Gupta, L. 2023a. HTTP Methods. Viitattu 29.3.2024. Saatavissa https://restfulapi.net/http-methods/

Gupta, L. 2023b. What is REST. Viitattu 29.3.2024. Saatavissa https://restfulapi.net/

IBM. What is virtualization. Viitattu 5.4.2024. Saatavissa https://www.ibm.com/topics/virtualization

ISS Yritysvastuuraportti. 2024a. ISS:n arvot. Viitattu 10.5.2024. Saatavissa https://issyritysvastuuraportti.fi/iss-yrityksena/

ISS Yritysvastuuraportti. 2024b. ISS lyhyesti. Viitattu 10.5.2024. Saatavissa https://issyritysvastuuraportti.fi/iss-yrityksena/

ISS Yritysvastuuraportti. 2024c. Luomme kestävää hyvinvointia. Viitattu 10.5.2024. Saatavissa https://issyritysvastuuraportti.fi/iss-yrityksena/

Lukka, K. 2001. Konstruktiivinen tutkimusote. Viitattu 1.4.2024. Saatavissa https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/

McKenzie, C. 2023. HTTP request methods explained. Blogikirjoitus. TechTarget. Viitattu 29.3.2024. Saatavissa https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/HTTP-methods

MySQL. 2024a. General Information. Viitattu 23.5.2024. Saatavissa https://dev.mysql.com/doc/refman/8.0/en/introduction.html

MySQL. 2024b. History of MySQL. Viitattu 23.5.2024. Saatavissa https://dev.mysql.com/doc/refman/8.0/en/history.html

MySQL. 2024c. The Main Features of MySQL. Viitattu 23.5.2024. Saatavissa https://dev.mysql.com/doc/refman/8.0/en/features.html

MySQL. 2024d. What is MySQL. Viitattu 23.5.2024. Saatavissa https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

Python docs. 2024a. General Python FAQ - What is Python. Viitattu 17.4.2024. https://docs.python.org/3/faq/general.html#what-is-python

Python docs. 2024b. General Python FAQ - Why is it called Python. Viitattu 17.4.2024. https://docs.python.org/3/faq/general.html#why-is-it-called-python

Red Hat. 2018a. Understanding virtualization. Viitattu 5.4.2024. Saatavissa https://www.redhat.com/en/topics/virtualization

Red Hat. 2018b. What is Docker. Viitattu 5.4.2024. Saatavissa https://www.redhat.com/en/topics/containers/what-is-docker

Rieuf, E. 2016. History of MySQL. Blogikirjoitus. TechTarget. Viitattu 23.5.2024. Saatavissa https://www.datasciencecentral.com/history-of-mysql/

Strotmann, J. 2016. A Brief History of Containerization. Blogikirjoitus. Plesk. Viitattu 5.4.2024. Saatavissa https://www.plesk.com/blog/business-industry/infographic-brief-history-linux-containerization/

VMware. 2008. Understanding Full Virtualization, Paravirtualization, and Hardware Assist. Viitattu 5.4.2024. Saatavissa https://www.vmware.com/techpapers/2007/understanding-full-virtualization-paravirtualizat-1008.html

Xu, A. 2022. Why is RESTful API so popular. Blogikirjoitus. ByteByteGo. Viitattu 29.3.2024. Saatavissa https://blog.bytebytego.com/p/why-is-restful-api-so-popular