jamk

Liikkuvan kaluston ohjelmiston hallinta DevOps-menetelmillä

Matias Ruha

Opinnäytetyö, AMK
Maaliskuu 2021
Tietojenkäsittelyn ja tietoliikenteen koulutusala
Tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Matias Ruha

Liikkuvan kaluston ohjelmiston hallinta DevOps-menetelmillä

Jyväskylä: Jyväskylän ammattikorkeakoulu. Maaliskuu 2021, 41 sivua.

Tietojenkäsittelyn ja tietoliikenteen koulutusala. Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Verkkojulkaisulupa myönnetty: kyllä

Tiivistelmä

Toimeksiantaja Devecto Oy:n asiakas tarvitsi liikkuvan kaluston ohjelmiston hallinnan. Aiemmin ohjelmistot ovat päivitetty USB-muistitikkujen avulla ja käytössä olevista ohjelmistoversioista on pidetty kirjaa manuaalisesti. Uuden sukupolven liikkuva kone on kehityksessä ja niissä olevien ohjelmoitavien logiikkaohjaimien ohjelmiston hallinta halutaan toteuttaa pilvipalveluun. Liikkuvaan kalustoon kuuluvat koneet tulevat olemaan työmailla ympäri maailmaa. Haasteena oli se, että koneet saattavat olla verkossa vain muutamina päivinä kuukaudessa.

Opinnäytetyön tavoitteena oli suunnitella ja kehittää asiakkaan vaatimukset täyttävä toteutus liikkuvan kaluston ohjelmoitavien logiikkaohjaimien ohjelmiston hallintaan. Asiakas haluaa määritellä kullekin koneelle käytettävän ohjelmistoversion ja automatisoida ohjelmiston päivittämisen sekä käytössä olevien ohjelmistoversioiden kirjanpidon. Asiakas vaati DevOps-menetelmien hyödyntämistä.

Työn toteutus oli kolmivaiheinen. Vaiheet olivat suunnittelu, käytännön toteutus ja testaaminen. Käytännön toteutus aloitettiin ottamalla jatkuvan integroinnin ja jatkuvan toimittamisen ohjelmistokehityskäytäntöjä käyttöön ohjelmoitavan logiikkaohjaimen ohjelmiston kehityksessä. Ohjelmistokehittäjät käyttivät git-versionhallintaa ja integroivat muutoksensa päähaaraan. Muutokset laukaisevat CI/CD-putken, joka koostaa, testaa ja siirtää ohjelmiston pilvipalveluun. Pilvipalveluun kehitettiin kaksi ohjelmointirajapintaa, joilla ohjelmistoversion kysely ja lataus pilvipalvelun ulkopuolelta hoidetaan. Liikkuvassa koneessa on Linux-pohjainen mobiilitietokone, johonka on kehitetty sovellus, joka käyttää ohjelmointirajapintoja. Sovellusta suoritetaan Docker-kontissa. Ladattu ohjelmistoversio siirretään ohjelmoitavalle logiikkaohjaimelle FTP-palvelimen kautta, jota suoritetaan myös Docker-kontissa.

Työn tuloksena toimeksiantajan asiakas sai toimivan, tietoturvallisen ja heidän vaatimuksensa täyttävän kokonaisuuden ohjelmoitavan logiikkaohjaimen ohjelmiston hallintaan. Pilvipalvelusta voidaan määritellä kullekin liikkuvalle koneelle käytettävä ohjelmistoversio ja voidaan nähdä kunkin liikkuvan koneen käyttämä ohjelmistoversio.

Avainsanat (asiasanat)

DevOps, jatkuva integrointi, jatkuva toimitus, pilvipalvelu, liikkuva kalusto, ohjelmoitava logiikkaohjain, ohjelmointirajapinta

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Ruha, Matias

Software management of mobile machines with DevOps methods

Jyväskylä: JAMK University of Applied Sciences, March 2021, 41 pages.

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

Permission for web publication: Yes

Language of publication: Finnish

Abstract

The customer of the assignor Devecto Oy needed software management for mobile machines. In the past, software has been updated using USB memory sticks and a record of the software versions in use has been kept manually. A new generation mobile machine is in development and the software management of the programmable logic controllers in them is to be implemented in the cloud service. The machines will be in use on construction sites around the world. The challenge was that the machines might only be online a few days a month.

The purpose of the thesis was to design and develop software management for the programmable logic controllers of mobile machines that meets the customer's requirements. The customer wants to define the software version to be used for each machine and automate the software update as well as keep a record of the software versions in use. The customer required the use of DevOps methods.

The development work was three-phase. The phases were design, implementation and testing. The practical implementation started with the introduction of continuous integration and continuous delivery software development practices in the development of programmable logic controller software. Software developers used Git version control and integrated their changes into the main branch. The changes trigger a CI/CD-pipeline that compiles, tests, and transfers the software to the cloud service. Two application programming interfaces were developed for the cloud service, which are used to query and download the software version from outside the cloud service. The mobile machine includes a mobile computer with Linux operation system for which an application has been developed using those application programming interfaces. The application runs in the Docker container. The downloaded software version is transferred to the programmable logic controller via an FTP server, which is also run in the Docker container.

As a result of the thesis, the customer got a functional and secure programmable logic controller software management that meets their requirements. The cloud service can be used to define and view the software version for each mobile machine.

Keywords/tags (subjects)

DevOps, continuous integration, continuous delivery, cloud service, mobile machine, programmable logic controller, application programming interface

Miscellaneous (Confidential information)

Sisältö

Kuviot

1 Johdanto

1.1 Yleistä

DevOps-menetelmien käyttäminen ohjelmistokehitysprojekteissa on suuressa, jatkuvasti kasvavassa roolissa ohjelmistoalalla. Ne automatisoivat ja nopeuttavat monia eri työvaiheita ja vähentävät huomattavan paljon ohjelmistokehityksen aikana tehtyjä virheitä. DevOps-menetelmät tukevat ketterän kehityksen projekteja ja yhä useampi asiakas haluaa sovelluksilleen jatkuvan toimituksen.

1.2 Opinnäytetyön tausta ja tavoite

Opinnäytetyön aiheena on suunnitella ja toteuttaa toimeksiantajan asiakkaalle DevOps-menetelmiä käyttäen kokonaisuus, jolla voidaan päivittää liikkuvassa kalustossa olevien ohjelmoitavien logiikkaohjaimien ohjelmisto. Kyseinen kokonaisuus on vain osa suurempaa ohjelmistokehitysprojektia. Haluttu lopputulos on se, että ennen uuden ohjelmistoversion julkaisua asiakas valitsee, mitkä koneet saavat hakea verkosta uuden ohjelmistoversion ja asentaa sen. Asiakas näkee kaikki koneet, koneisiin valitut ja niihin asennetut ohjelmistoversiot pilvipalvelusta. Haasteena on muun muassa se, että laitteet saattavat olla ympäri maailmaa ja ilman verkkoyhteyttä jopa viikkoja. Kokonaisuuteen kuuluu myös muita asiakkaan asettamia vaatimuksia, jotka tulisi täyttää. Opinnäytetyössä käsitellään ohjelmistoversion siirtymistä ohjelmistokehittäjän työkoneelta aina ohjelmoitavaan logiikkaohjaimeen asti.

Opinnäytetyön tavoitteena on suunnitella ja toteuttaa toimeksiantajan asiakkaalle vaatimusten mukainen kokonaisuus liikkuvassa kalustossa olevien ohjelmoitavien logiikkaohjaimien ohjelmistoversioiden hallintaan ja päivittämiseen käyttäen DevOps-menetelmiä. Henkilökohtaisena tavoitteena on oppia enemmän DevOps-menetelmistä ja niiden hyödyntämisestä osana ohjelmistokehitystä.

Konkreettisena tuloksena on tavoitteena saada tietoturvallinen kokonaisuus, joka sisältää liikkuvan kaluston ohjelmoitavien logiikkaohjaimien ohjelmistoversioiden hallinnan ja viemisen halutuille laitteille verkkoyhteyden yli. Tämä tuottaa lisäarvoa asiakkaalle, sillä ohjelmistoversiot voidaan

testata halutuissa yksilöissä ja sitten asettaa ohjelmistoversio saataville kaikille muille koneille. Lisäksi kaikki tämä tapahtuu pilvipalveluun rakennetun käyttöliittymän kautta.

1.3 Aiheen rajaus

Aihe on rajattu siten, että tässä työssä keskitytään ohjelmoitavien logiikkaohjaimien ohjelmiston hallintaan ja ohjelmistoversioiden viemiseen ohjelmistokehittäjän paikalliselta tietokoneelta pilvipalveluun ja sieltä tietoturvallisesti liikkuvassa koneessa olevaan mobiilitietokoneeseen, josta ohjelmoitava logiikkaohjain käy lukemassa ohjelmistoversion, jonka jälkeen konetta huoltava henkilö voi painiketta painamalla päivittää ohjelmoitavan logiikkaohjaimen ohjelmiston uuteen versioon. Sisällytän työhön vain olennaisimmat tiedot DevOpsista, DevOps-menetelmistä, työkaluista ja palveluista, joita tässä kokonaisuudessa käytetään. Jätän työn ulkopuolelle itse ohjelmiston sisällön sekä komponenttien, työkalujen ja palveluiden tarjoajan valinnat, koska toimeksiantaja ja asiakas ovat ne jo päättäneet ohjelmistokehitysprojektin alkaessa kesällä 2020.

1.4 Toimeksiantaja Devecto Oy

Opinnäytetyön toimeksiantaja on Devecto Oy, joka on perustettu Jyväskylässä vuonna 2014. Devecton pääkonttori sijaitsee Jyväskylän Tourulassa. Jyväskylän lisäksi toimistoja löytyy Suomen muista kaupungeista kuten Espoosta, Tampereelta, Kajaanista ja Oulusta. Devecton liikevaihto ylsi 6 miljoonaan euroon vuonna 2019. Devecto työllistää yli 80 henkilöä ja se on myös laajasti henkilökunnan omistama. Kuviossa 1 on Devecton logo. (Devecto n.d.)

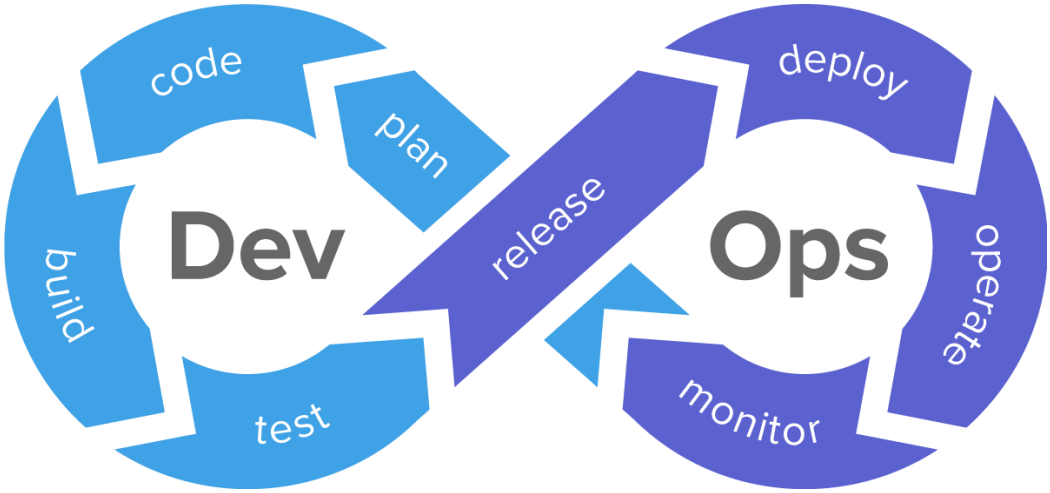
Devecto on erityisesti älykkäiden laitteiden ja koneiden ohjelmistosuunnitteluun ja niihin liittyviin testausjärjestelmiin erikoistunut tuotekehitystalo. Liikkuvan kaluston ja sulautettujen järjestelmien lisäksi Devecto tarjoaa myös digitalisaatio, web- ja mobiiliratkaisuja sekä mallipohjaista suunnittelua. Asiakkaina on laajalti eri kokoisia yrityksiä alansa johtavista kansainvälisistä yrityksistä kehittyviin startuppeihin asti. (Devecto n.d.)

Kuvio 1. Devecto Oy logo (Devecto n.d.)

2 DevOps

2.1 Yleistä

DevOps-termi muodostuu sanoista development eli ohjelmistokehitys ja operations eli IT-palvelutoiminnot. DevOps on laaja toimintamalli, jonka alkuperäinen tarkoitus on ollut lisätä ohjelmistokehittäjien ja ylläpitoon liittyvien IT-palvelutoiminnoista huolehtivien tiimien yhteistyötä ohjelmistokehityksen eri vaiheissa. Tiimien vähäinen yhteistyö on aiheuttanut ristiriitoja, joka on näkynyt yrityksen heikkoina tuloksina. Nykypäivänä DevOps-toimintamalli on yhdistelmä yrityskulttuuria, käytäntöjä ja työkaluja. Hitaita prosesseja kuten testaamista ja koodin integrointia pyritään automatisoimaan työkalujen avulla ja ohjelmistokehityksessä käytetään käytäntöjä, jotka nopeuttavat uuden ohjelmistoversion julkaisemista. Ohjelmistokehittäjä saattaa työskennellä koko sovelluksen tai palvelun elinkaaren ajan kehitystyöstä käyttöönottoon ja operointiin asti. Lisäksi suoritetaan sellaisia tehtäviä yksin, jotka aiemmin ovat vaatineet muiden apua, kuten julkaisun tekeminen. DevOps-toimintamalli lisää yrityksen kykyä toimittaa sovelluksia ja palveluita nopeammin kuin perinteinen ohjelmistokehitys ja infrastruktuurinhallinta. Se parantaa myös sovelluksien ja palveluiden luotettavuutta ja skaalautuvuutta. Kuviossa 2 on esitetty DevOps-toimintamallin keskeisimmät vaiheet. (Guckenheimer 2018; What is DevOps n.d.)



Kuvio 2. DevOpsin vaiheet (DevOps and Application Security n.d)

DevOps-toimintamallin käytäntöjä on useita. Jatkuva integrointi (*Continuous Integration*), jatkuva toimittaminen (*Continuous Delivery*), jatkuva käyttöönotto (*Continuous Deployment*), versionhallinta, ketterä kehitys ja lean-menetelmät, monitorointi ja lokitus, pilvipalveluiden käyttäminen, infrastruktuurin ilmaiseminen koodina sekä mikropalveluiden ja konttiteknologian käyttäminen. (Guckenheimer 2018; What is DevOps n.d.)

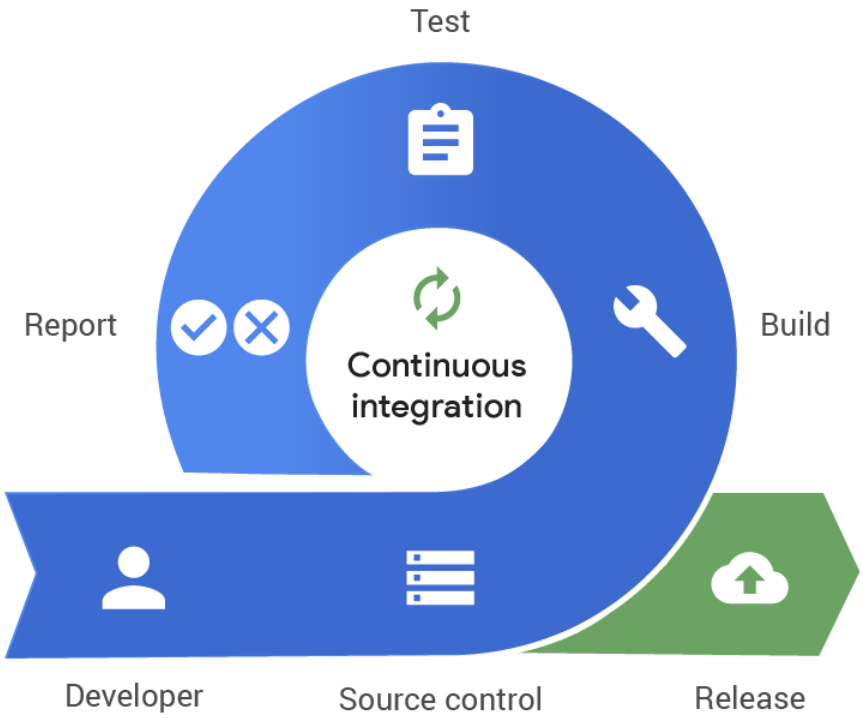
2.2 Jatkuva integrointi

Jatkuva integrointi on ohjelmistokehityskäytäntö, jossa kehitystiimin jäsenet integroivat koodinsa vähintään kerran päivässä versionhallinnan päähaaraan. Päähaaraan tulee useita integraatioita päivittäin. Integraatio testataan automatisoidulla koonnilla virheiden havaitsemiseksi. Kuviossa 3 esitetään karkeasti jatkuvan integroinnin eri vaiheet. Tämä on perinteistä integrointia nopeampi tapa kehittää yhtenäisiä ohjelmistoja ja pienentää integroinnissa esiintyviä ongelmia. (Guckenheimer 2017; Fowler 2006)

Jatkuvaan integrointiin liittyy muutamia käytäntöjä. Lähdekoodia on hyvä säilyttää yhdessä paikassa ja sen tulisi sisältää kaikki koostamiseen tarvittavat tiedostot, testiskriptit, tietokantamallit ja kirjastotiedostot, mutta ei koostettuja versioita. Tavoitteena on se, että koostamisen voi tehdä uudella tietokoneella mahdollisimman pienellä vaivalla. Lisäksi koostaminen tulisi automatisoida, sillä se vähentää toistuvaa työtä ja inhimillisten virheiden määrää. Koostamiseen tulisi sisällyttää automatisoituja testejä, koska koostettu ja ajettava

ohjelmisto ei tarkoita virheetöntä ohjelmistoa. Ohjelmiston toiminnallisuus on testattava loogisten virheiden havaitsemiseksi. Automatisoitu testaus on nopea ja tehokas tapa löytää bugit ajoissa. Koostamisen tulisi epäonnistua, jos ohjelmisto ei läpäise toiminnallisia testejä. Jokainen päähaaraan tullut muutos on koostettava integraatiokoneella tai koontipalvelimella. Näin virheet voidaan jäljittää jopa tuntien tarkkuudella ja se nopeuttaa virheiden löytämistä. Integrointikoneella ja -palvelimella puolestaan varmistetaan se, että ohjelmisto ei ole ohjelmistokehittäjän kehitysympäristöstä riippuvainen vaan se toimii myös muissa kehitysympäristöissä. Mikäli päähaaran koostaminen ei onnistu, se on korjattava korkeimmalla prioriteetilla. Koostaminen on pidettävä mahdollisimman nopeana, koska jatkuvan integroinnin pääpointti on nopeuttaa integrointia ja palautteen antamista ohjelmistokehittäjälle. (Guckenheimer 2017; Fowler 2006)

Nopeutta voidaan lisätä koostamisputkella, jolla voidaan pilkkoa kriittisemmät testit tehtäväksi ennen kuin muutokset lisätään päähaaraan. Hitaammat testit kuten tietokantaan liittyvät testit ja end-to-end testit voidaan suorittaa vasta sen jälkeen. Testaamisen jälkeen viimeisin ajettava versio ohjelmistosta tulisi asettaa helposti saataville. Käytäntöjä seuraamalla integrointi ja testaaminen nopeutuu, virheiden löytäminen nopeutuu, virheistä aiheutuvat kustannukset pienenevät, koontiversioiden laatu pysyy tasaisena ja tuotantoversioon päässeiden virheiden määrä vähenee. (Guckenheimer 2017; Fowler 2006)



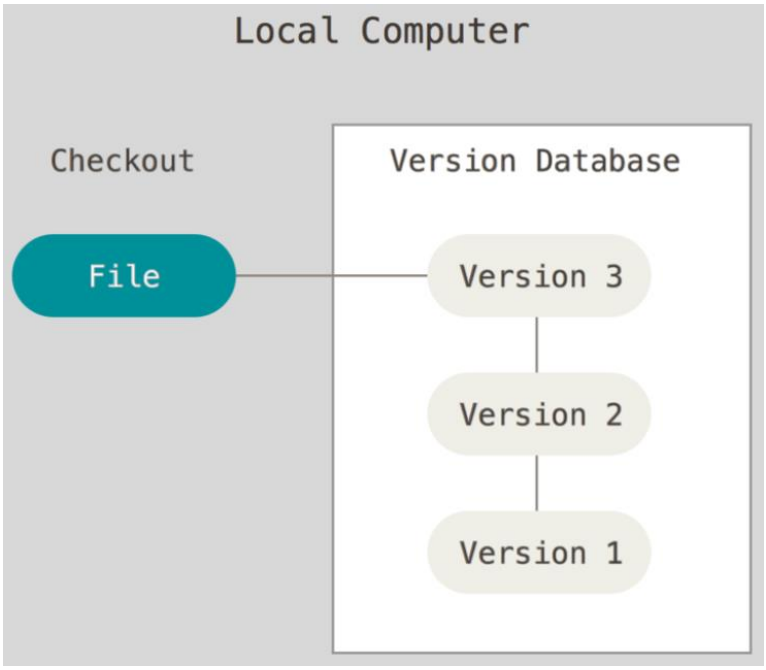
Kuvio 3. Jatkuva integrointi vaiheina (What is Continuous Integration n.d.)

2.2.1 Versionhallinta

Versionhallinnalla tarkoitetaan yleensä järjestelmää, jolla voidaan seurata ja hallita tiedostoihin tehtyjä muutoksia. Jokaisella tiedostolla on oma pitkäaikainen muutoshistoria. Versionhallinta auttaa ohjelmistokehittäjää ja ohjelmistokehittäjistä koostuvaa tiimiä hallitsemaan muutoksia ajan myötä ja suojaamaan lähdekoodia sekä inhimillisiltä että tahattomilta virheiltä. Se tuo läpinäkyvyyttä kehitystyöhön, koska ohjelmistokehittäjät näkevät mitä muutoksia muut ohjelmistokehittäjät ovat tehneet ja halutessaan muutokset voidaan yhdistää projektinhallinta- ja vikaseurantatyökaluihin. Tiedostoja voidaan verrata niiden aiempiin versioihin ja tarvittaessa tiedosto tai ohjelmavaraston kaikki tiedostot voidaan palauttaa takaisin aiempaan versioon. Lisäksi muutoshistorian läpikäyminen ja versioiden vertaaminen toisiinsa nopeuttavat virheiden paikallistamista. (About version control n.d.; What is version control n.d.)

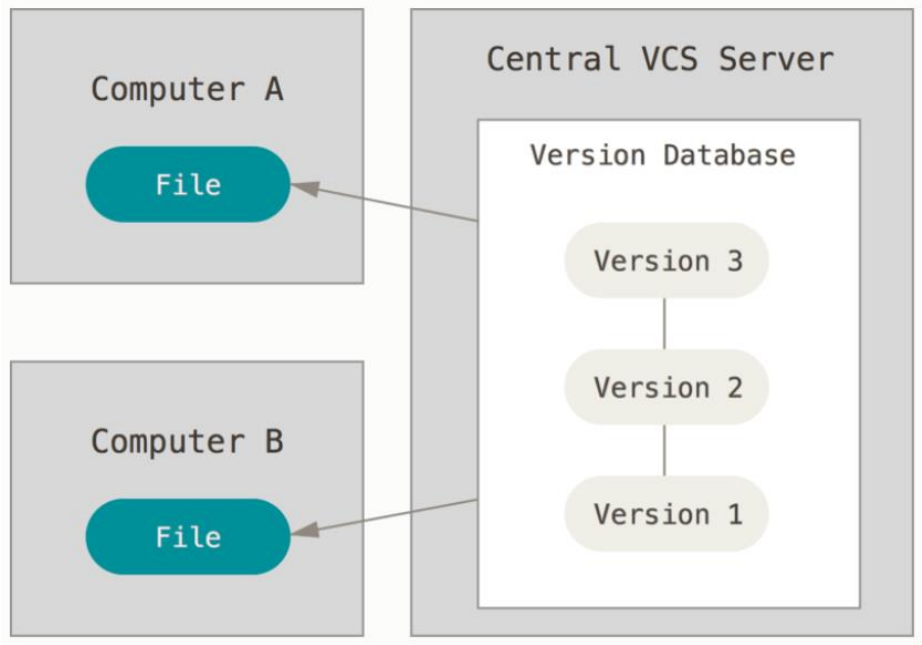
Versionhallinta voidaan karkeasti jaettuna toteuttaa kolmella eri tavalla: paikallinen versionhallinta, keskitetty versionhallintajärjestelmä tai hajautettu versionhallintajärjestelmä. Paikallinen versionhallinta on vain kehittäjän omalla tietokoneella (ks. kuvio 4). Jos paikallisesta

versionhallinnasta ei ole varmuuskopiota, on riskinä menettää projektin tiedostot ja historiatiedot kovalevyn hajotessa.



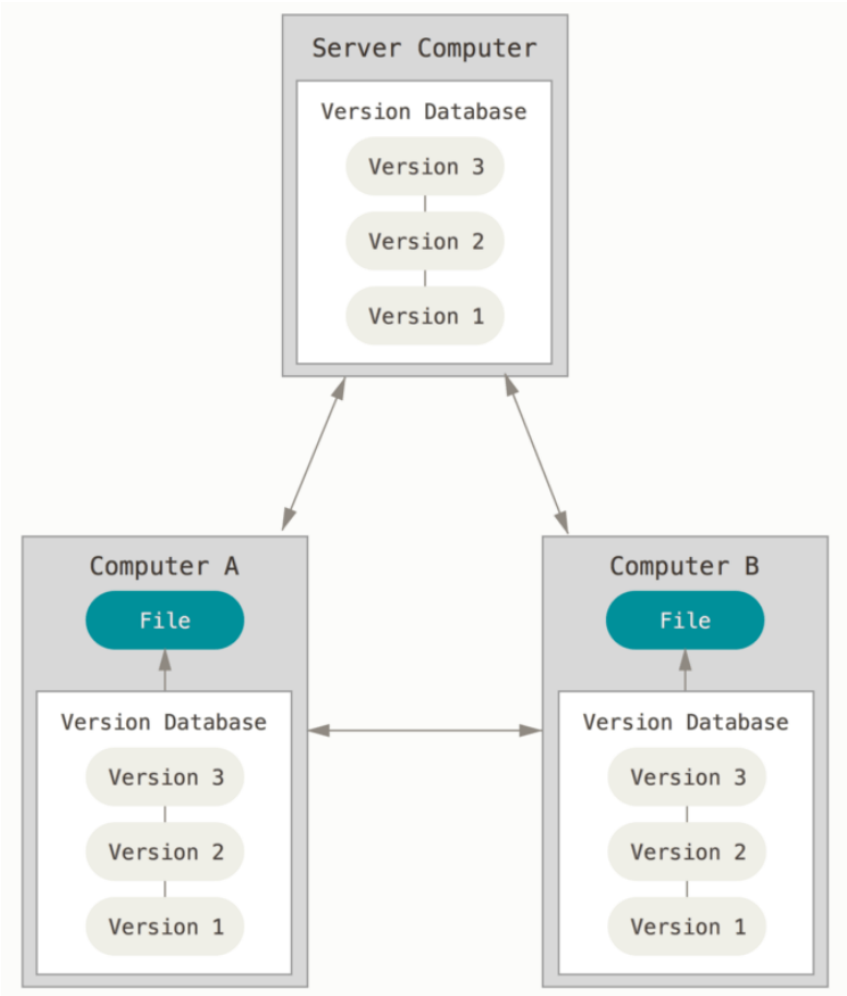
Kuvio 4. Paikallinen versionhallinta (About version control n.d.)

Keskitetty versionhallintajärjestelmä on palvelin, jossa kaikki projektin tiedostot ovat versioituna (ks. kuvio 5). Se mahdollistaa yhteistyön ohjelmistokehittäjien välillä. Ohjelmistokehittäjät hakevat projektin tiedostoista tilannekuvan omalle tietokoneelleen. Keskitetyssä versionhallintajärjestelmän käyttöön liittyy ainakin kaksi isoa riskiä. Palvelimen yhteyden katketessa ohjelmistokehittäjät eivät voi tallentaa muutoksiaan ja palvelimen kovalevyn hajotessa projektin tiedostot ja historiatiedot saatetaan menettää kokonaan. (About version control n.d.; What is version control n.d.)



Kuvio 5. Keskitetty versionhallintajärjestelmä (About version control n.d.)

Hajautetulla versionhallintajärjestelmällä päästään riskistä menettää projektin tiedostot ja historiatiedot kovalevyn tuhoutuessa. Hajautettu versionhallintajärjestelmä muodostuu siten, että ohjelmistokehittäjä ei lataa palvelimelta projektin tiedostoista tilannekuvaa, vaan täydellisen kloonin tiedostoista ja niiden muutoshistoriasta (ks. kuvio 6). Jos palvelimella olevat tiedostot tuhoutuva, ohjelmistokehittäjän tietokoneelta voidaan palauttaa kaikki tiedostot ja niiden historiatiedot. Esimerkiksi Git-versionhallinta on toteutettu tällä tavoin. (About version control n.d.; What is version control n.d.)



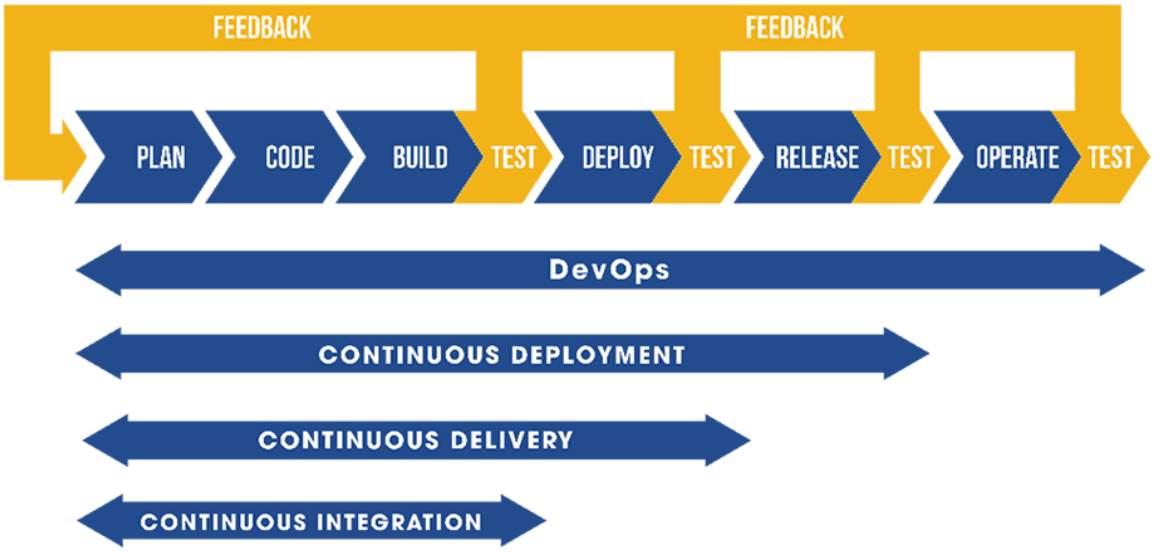
Kuvio 6. Hajautettu versionhallintajärjestelmä (About version control n.d.)

2.3 Jatkuva toimitus

Jatkuva toimitus on ohjelmistokehityskäytäntö, jossa ohjelmistoa kehitetään siten, että se voidaan julkaista tuotantoon koko sen elinkaaren ajan. Tarvittaessa voidaan toimittaa mikä tahansa versio mihinkä tahansa toimintaympäristöön. Se priorisoi ohjelmiston toimittamisen uusien ominaisuuksien kehittämisen sijasta. Jatkuvan toimittamisen toteuttaminen edellyttää jatkuvan integroinnin ohjelmistokehityskäytännön toteuttamista. Jatkuva integrointi toteuttaa jatkuvan toimituksen ensimmäiset vaiheet eli koostaa ohjelmiston ja suorittaa sille yksikkö- ja integrointitestejä. Sen jälkeen ohjelmisto voidaan toimittaa testausympäristöön automaattisesti ja suorittaa sille vaativampia testejä, kuten käyttöliittymä- ja rasitustestejä. (Fowler 2013; Pitter n.d.; What is Continuous Delivery n.d)

Jatkuvan toimituksen etuina ovat pienempi käyttöönottoriski, valmiiksi merkattujen ominaisuuksien käyttöön ottaminen ja nopean ja jatkuvan palautteen saaminen oikeilta käyttäjiltä esimerkiksi uudesta ominaisuudesta. Pienempi käyttöönottoriski muodostuu siitä, että käyttöön otetaan jatkuvasti pieni määrä muutoksia. Jos virheitä ilmenee, niiden paikallistaminen on nopeaa. Lisäksi tarjolla on aina käyttöönottovalmis versio. (Fowler 2013; Pittet n.d.; What is Continuous Delivery n.d.)

Jatkuva toimitus sekoitetaan usein jatkuvaan käyttöönottoon (*Continuous deployment*) ohjelmistokehityskäytäntöön. Erona on se, että jatkuvassa käyttöönotossa kaikki muutokset julkaistaan automaattisesti tuotantoon. Jatkuvassa toimituksessa on tietoisesti tehty valinta, ettei näin haluta tapahtuvan. Jatkuvan käyttöönoton toteuttaminen edellyttää jatkuvan toimituksen toteuttamista. Kuviossa 7 on esitetty jatkuvan integroinnin, jatkuvan toimittamisen ja jatkuvan käyttöönottamisen vaiheet. (Fowler 2013; Pittet n.d.; What is Continuous Delivery n.d)



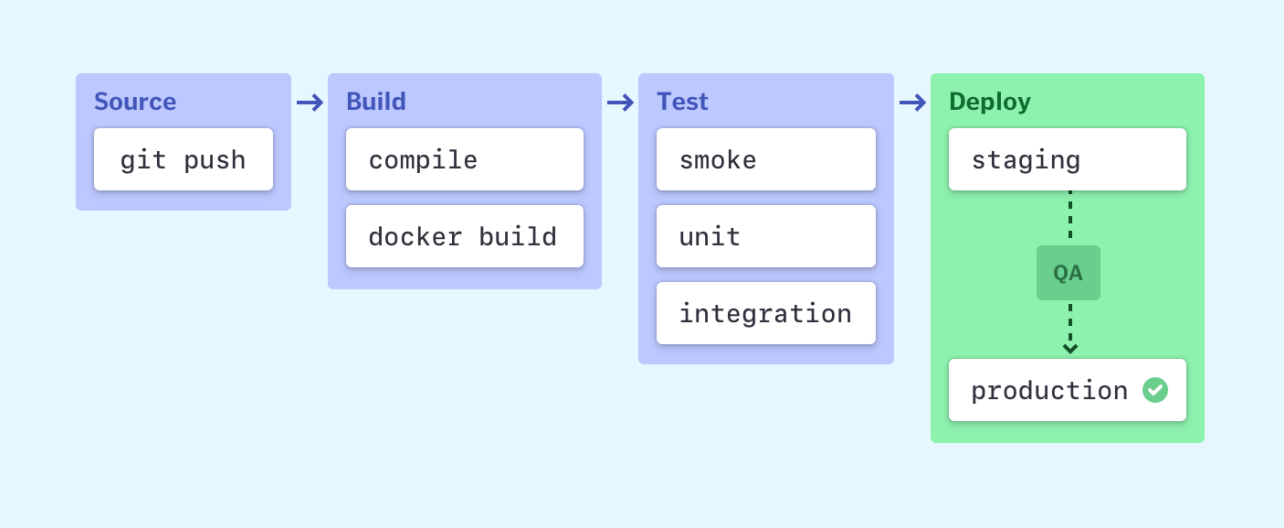
Kuvio 7. Jatkuvan integroinnin, jatkuvan toimittamisen ja jatkuvan käytöönottamisen vaiheet (Continuous Delivery 2018, muokattu)

2.4 CI/CD-putki

CI/CD-putkella (*CI/CD-pipeline*) automatisoidaan ja monitoroidaan jatkuvan integroinnin (CI) ja jatkuvan toimittamisen (CD) työvaiheita. Se on sarja työvaiheita, jotka on suoritettava

ohjelmistoversion toimittamiseksi. Putkilla parannetaan ohjelmistojen toimitusta, kun käytössä ovat DevOps-menetelmät. (What is a CI/CD pipeline n.d.a)

CI/CD-putki voidaan jakaa karkeasti neljään eri vaiheeseen (ks. kuvio 8). Ensimmäinen vaihe on muutoksen huomaaminen lähdekoodissa (source), jossa havaittu lähdekoodin muutos laukaisee CI/CD-putken suorittamisen. Toisena vaiheena on koostaminen (build), jossa sovellus koostetaan ajettavaksi. Kolmantena vaiheena on testaaminen (test), jossa ajetaan automatisoidut testit koodin oikeellisuuden ja ohjelmiston käyttäytymisen varmistamiseksi. Viimeisenä vaiheena on julkaisu (release), jossa sovellus, joka on läpäissyt testit, siirretään ohjelmavarastoon saataville. Vaihtoehtoisesti viimeinen vaihe voi olla myös käyttöönotto (deploy), joka sisältää julkaisuvaiheen toimenpiteet ja ottaa sovelluksen käyttöön automaattisesti tuotantoympäristössä. (Goyal 2020; What is a CI/CD pipeline n.d.b)



Kuvio 8. CI/CD-putken vaiheet (CI/CD Pipeline 2019)

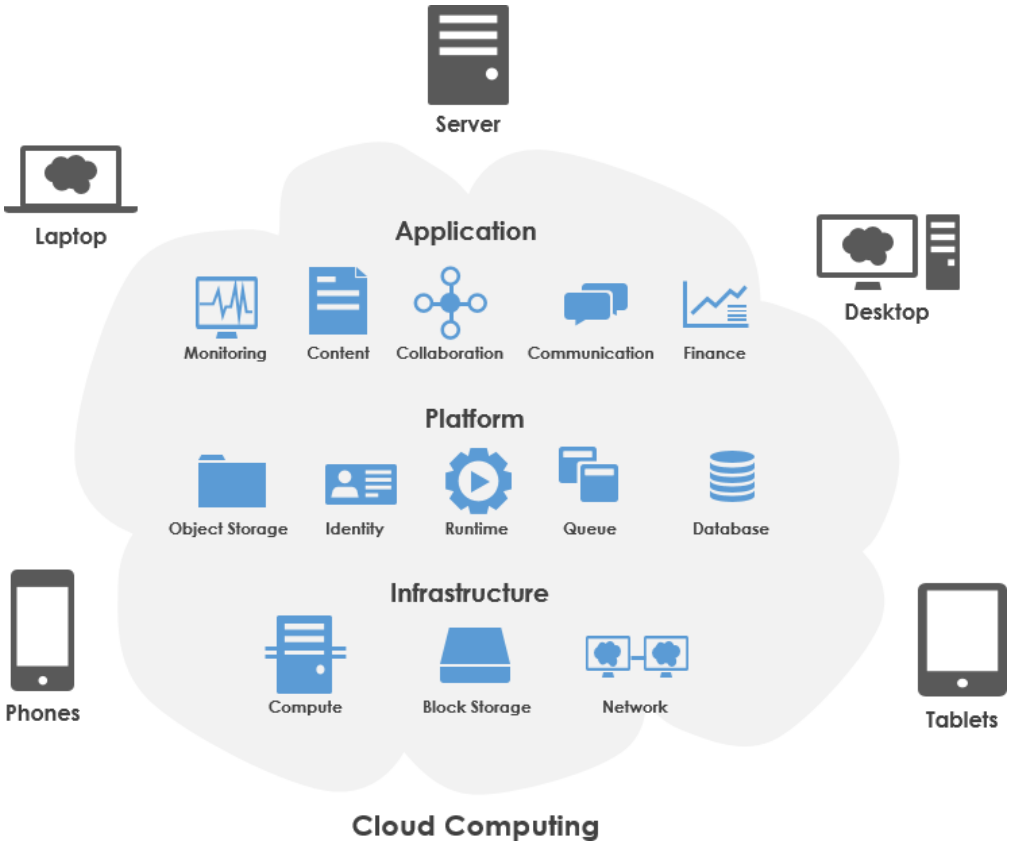
3 Pilvipalvelu

3.1 Yleistä

Pilvipalvelut ovat infrastruktuureja, alustoja ja ohjelmistoja, joita kolmannen osapuolen palveluntarjoaja tarjoaa yksityisille henkilöille ja yrityksille verkon yli. Yksinkertaisimmillaan pilvipalvelujen käyttöön tarvitaan vain tietokone, jossa on toimiva verkkoyhteys. Kaikkia infrastruktuureja, alustoja ja ohjelmistoja, joihin käyttäjät pääsevät verkon yli ilman

ohjelmistolatauksia, voidaan pitää pilvipalveluina. Pilvipalveluntarjoajat tarjoavat siis käyttäjilleen heidän palvelinkeskuksistaan IT-resursseja kuten palvelimia, laskentatehoa, tietokantoja ja tallennustilaa maksua vastaan (ks. kuvio 9). (What are cloud services n.d.; What is cloud computing n.d.)

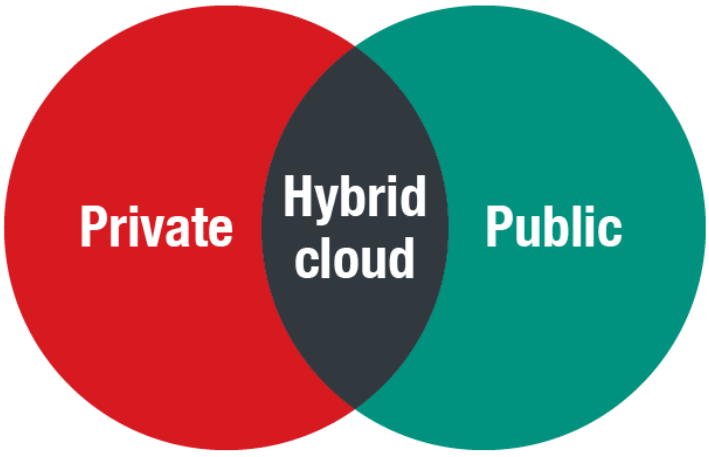
Pilvipalvelujen käyttämisessä on paljon etuja. Aloituskustannukset ovat pienet, sillä oman palvelimen tai palvelinkeskuksen ostamisen ja ylläpitämisen sijaan voidaan ostaa IT-resursseja palveluntarjoajalta ja maksaa vain resursseista, joita käytetään. Skaalautuvuus on joustavampaa, kuin perinteisellä laitteistolla. Lisäresursseja voidaan hommata maantieteellisesti oikeasta sijainnista. Korkean suorituskyvyn takaa se, että suurimmat pilvipalvelut toimivat maailmanlaajuisessa suojattujen palvelinten verkossa. Pilvipalvelut tarjoavat paljon eri vaihtoehtoisia käytäntöjä ja tekniikoita, jotka auttavat suojaamaan tietoja, sovelluksia ja infrastruktuuria. Lisäksi pilvipalvelut tarjoavat luotettavuutta tekemällä varmuuskopioita ja tarjoamalla redundanttisuutta. (What is cloud computing n.d.)



Kuvio 9. Pilvipalvelun arkkitehtuuri (Practical AWS Diagram tutorial and examples n.d.)

3.2 Pilvityypit

Pilvet voidaan luokitella kolmeen kategoriaan: julkinen pilvi, yksityinen pilvi ja näiden yhdistelmä eli hybridipilvi (ks. kuvio 10). Julkiset pilvet omistavat ja niitä ylläpitävät kolmannen osapuolen pilvipalveluntarjoajat. Kaikki julkisen pilven ylläpitämiseen tarvittava infrastruktuuri, laitteistot ja ohjelmistot ovat palveluntarjoajan omistuksessa ja hallinnassa. Yksityinen pilvi on pilvi, jonka palveluita yksinomaan käyttää jokin yritys tai organisaatio. Infrastruktuuri pilven ylläpitämiseen voi sijaita yrityksen palvelinkeskuksessa. Yritys voi myös maksaa kolmannen osapuolen pilvipalveluntarjoajalle yksityisen pilven pitämisestä. Palvelut ja infrastruktuuri ylläpidetään yksityisessä verkossa. Hybridipilvi on puolestaan julkisen pilven ja yksityisen pilven yhdistelmä, joka mahdollistaa sovellusten jakamisen niiden välillä. Se tarjoaa suuremman joustavuuden kuin pelkkä julkinen tai yksityinen pilvipalvelu. (What are cloud services n.d.; What is cloud computing n.d.)

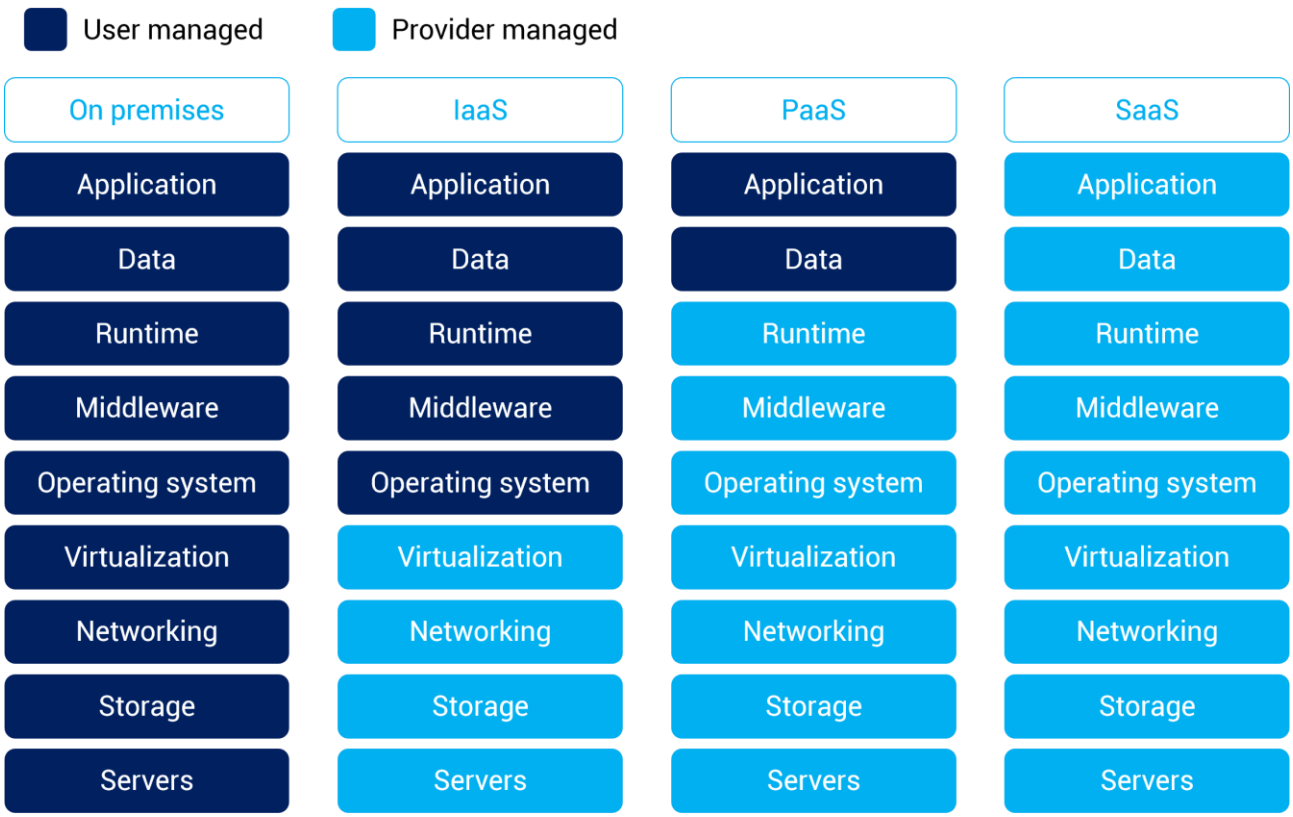


Kuvio 10. Pilvet (Key strategies for securing the hybrid cloud 2018)

3.3 Pilvipalvelu tyypit

Pilvipalvelut jaetaan yleisesti kolmeen eri tyyppiin: infrastruktuuri, alusta ja ohjelmisto. Infrastruktuuri palveluna (*Infrastructure-as-a-Service tai IaaS*) tarjoaa käyttäjille laskenta-, verkko ja tallennusresursseja. Se on tietojenkäsittelyinfrastruktuuri, jota hallinnoidaan verkon yli.

Kustannukset muodostuvat vain resursseista, joita käytetään. Se on helposti skaalautuva ja sen avulla voidaan välttää omien palvelimien ja palvelinkeskusten ostaminen ja ylläpitäminen. Palvelu soveltuu hyvin esimerkiksi testaus- ja kehitysympäristön luomiseen tai varmuuskopioiden säilyttämiseen. (What is IaaS n.d.a) Alusta palveluna (*Platform-as-a-Service tai PaaS*) puolestaan tarjoaa käyttäjälle infrastruktuuripalvelun lisäksi kehitystyökaluja, tietokantojen hallintajärjestelmiä ja virtuaalikoneiden käyttöjärjestelmiä. Alustapalvelu soveltuu parhaiten kehitys- ja käyttöönottoympäristöksi. Palvelun resurssien avulla voidaan toimittaa monimutkaisempiakin pilvisovelluksia. (What is PaaS n.d.) Ohjelmisto palveluna (*Software-as-a-Service tai SaaS*) tarjoaa käyttäjälle mahdollisuuden käyttää pilvipohjaisia sovelluksia, kuten sähköpostia, kalenteria ja muita toimistotyökaluja. Se on valmis ohjelmistoratkaisu, jota käytetään internetin välityksellä. Palveluntarjoaja hallinnoi taustalla olevaa infrastruktuuria ja sovellusta kokonaisuudessaan. (What is SaaS n.d.) Kuviossa 11 havainnollisestaan, mitä kaikkea missäkin palvelutyypissä on käyttäjän ja palveluntuottajan vastuulla ja hallinnassa.



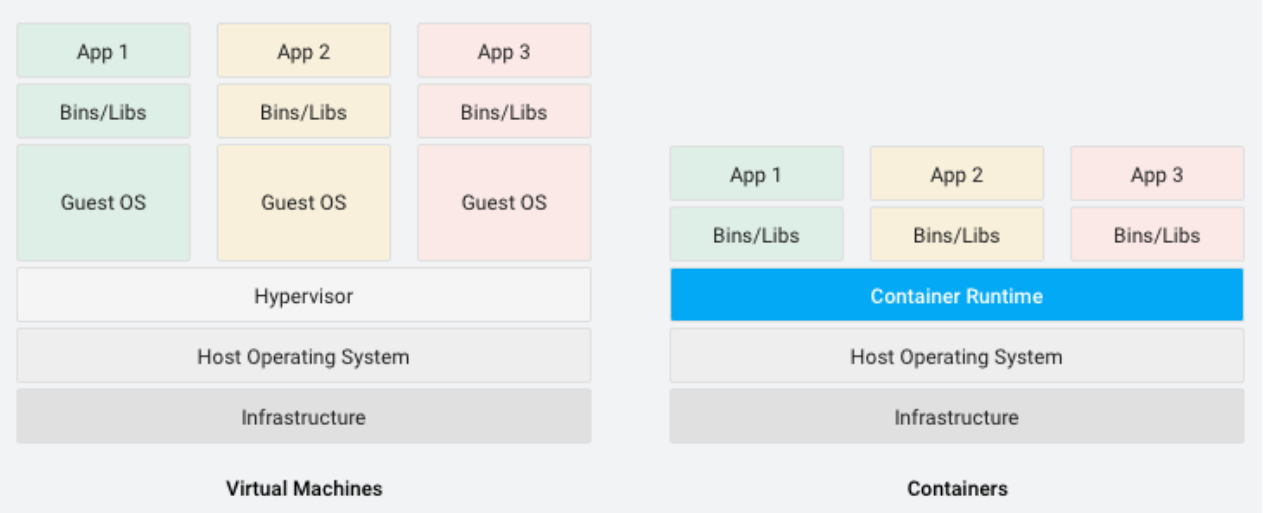
Kuvio 11. Pilvipalvelutyyppien eroavaisuudet (What is IaaS n.d.b)

Näiden palvelutyyppien lisäksi on vielä neljäs, palvelimeton tietojenkäsittely (*serverless computing*). Palvelu tarjoaa esimerkiksi verkkosovelluksille alustan, jonka infrastruktuurista ei tarvitse huolehtia lainkaan. Palveluntarjoaja huolehtii skaalautumisesta ja hallinnoi käytettävien resurssien määrää sovelluksen käytön perusteella. (Serverless computing n.d.)

4 Konttiteknologia

4.1 Yleistä

Kontti on standardisoitu ohjelmistoyksikkö, joka pakkaa koodin ja kaikki sen riippuvuudet siten, että sovellus toimii luotettavasti eri ympäristöissä. Täten konttipohjaiset sovellukset voidaan ottaa käyttöön helposti riippumatta kohdeympäristöstä. Kontteja verrataan usein virtuaalikoneisiin, joissa käyttöjärjestelmä toimii isäntäkäyttöjärjestelmän päällä ja joilla on virtualisoitu pääsy taustalla olevaan laitteistoon. Konteilla voidaan virtuaalikoneiden tapaan pakata sovellus kirjastojen ja muiden riippuvaisuuksien kanssa erilliseen ympäristöön. Kontit ovat kevyempiä yksiköitä kuin virtuaalikoneet, sillä kontit virtualisoivat käyttöjärjestelmätasolla. Ne eivät siis virtualisoi laitteistoa, kuten virtuaalikoneet. Kontit ovat nopeampia ja ne eivät käytä niin paljon muistia kuin virtuaalikoneet. Ne mahdollistavat muista sovelluksista eristettyjen ympäristöjen luomisen. Kontit toimivat kukin erillisinä prosesseina ja useat kontit voivat toimia samalla laitteella. Kuviossa 12 esitetään virtuaalikoneen ja kontin rakenne. (Containers at Google n.d.; What is a Container n.d.)

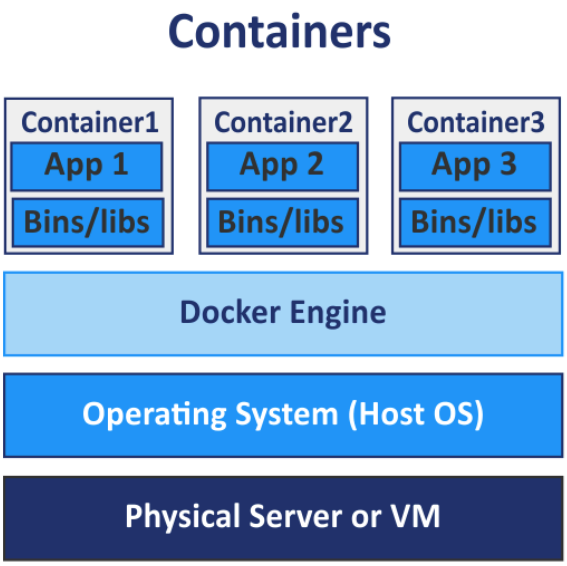


Kuvio 12. Virtuaalikoneen ja kontin rakenne (Containers at Google n.d.)

Konteissa on myös muita etuja. Niiden avulla kehittäjät voivat keskittyä sovelluksiin ja sen riippuvaisuuksiin. IT-operatiiviset tiimit puolestaan voivat keskittyä konttien käyttöönottoon ja hallintaan. Konttien avulla luotu standardisoitu ympäristö sovelluksille vähentää virheenkorjaukseen ja ympäristöerojen diagnosointiin käytettävää aikaa. Ne poistavat ongelmia, jotka aiheutuvat yleensä testausympäristön ja tuotantoympäristön eroavaisuuksista. (Containers at Google n.d.; What is a Container n.d.)

4.2 Docker

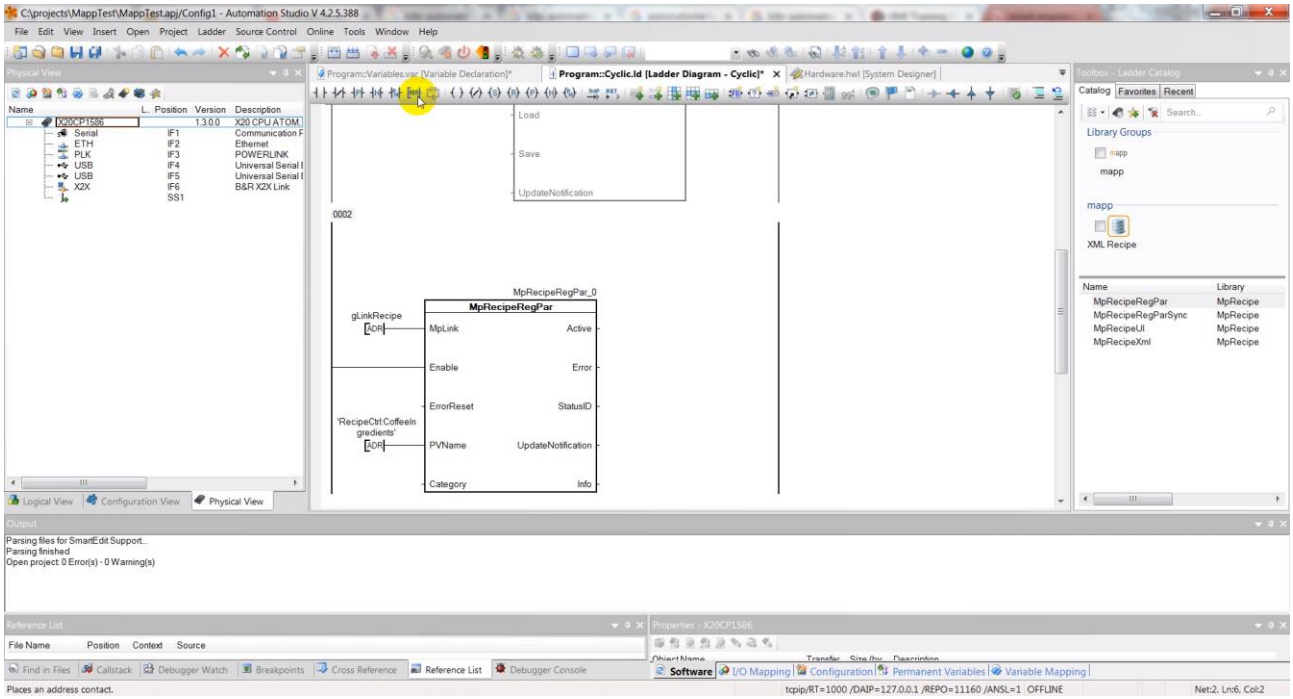
Docker on avoimen lähdekoodin alusta sovellusten kontittamiseen, konttien toimittamiseen ja käyttämiseen. Se tarjoaa työkalut ja alustan konttien elinkaaren hallinnalle. Dockerilla voidaan pakata sovellus ja käyttää sitä eristetyssä ympäristössä eli kontissa. (ks. kuvio 13). Docker-kontti eristää sovelluksen täysin infrastruktuurista. Kontit ovat kevyitä ja ne sisältävät kaiken sovelluksen suorittamiseen tarvittavan. Docker-kontit nopeuttavat ohjelmistojen toimittamista ja käyttöönottoa, sillä käyttöönotto tehdään samalla tavalla ohjelmistokehittäjän tietokoneella kuin paikallisella palvelimella tai pilvipalvelussa. Docker-kontit sopivat hyvin CI/CD-työkulkuun. Docker käyttää Linux-ydintä ja ytimen ominaisuuksia, jotta kontteja voidaan ajaa itsenäisinä prosesseina. Docker tarjoaa myös kuvapohjaisen käyttöönottomallin ja se automatisoi sovelluksen käyttöönoton konttiympäristössä. (Docker overview n.d.; What is Docker n.d.)



Kuvio 13. Docker-ympäristö (Bose 2019, muokattu)

5 B&R Automation Studio -ohjelmistokehitysympäristö

B&R Automation Studio on itävaltalaisen automaatio- ja prosessinohjaustekniikan yrityksen B&R:n tarjoama integroitu ohjelmistokehitysympäristö (ks. kuvio 14), joka sisältää työkaluja projektin kaikkiin vaiheisiin. Ohjaimet, taajuusmuuttajat, kommunikaatio ja visualisointi voidaan konfiguroida samassa ohjelmistokehitysympäristössä. B&R Automation Studio tarjoaa kielivaihtoehtoina IEC 61131-3 standardin mukaiset ohjelmointikielet eli käskylista (IL), rakenteinen teksti (ST), tikapuukaavio (LD), toimilohkokaavio (FBD) ja vuokaavio (SFC). Lisäksi vaihtoehtona on myös ANSI-C-ohjelmointikieli. Ohjelmistokehitysympäristöllä kommunikaatio laitteiden välillä voidaan muodostaa useilla eri kenttäväylillä. Se tarjoaa myös mahdollisuuden etäyhteyden luomiseen ja sitä voi käyttää hyödyksi esimerkiksi ajon aikaisen prosessin hallinnassa, diagnosoinnissa tai etäohjelmoinnissa. (Automation Studio 2014, 8, 20, 32; Automation Studio n.d.)



Kuvio 14. B&R Automation Studio -ohjelmistokehitysympäristö (Cammarano 2017, 8:40)

B&R Automation Studiossa käytetään B&R:n mapp-teknologiaa. Mapp-komponentit ovat valmiiksi rakennettuja konfiguroitavia komponentteja, joiden avulla ohjelmistokehittäjät voivat keskittyä ennemmin prosessin ohjauksen kehittämiseen. Ohjelmoinnin sijasta konfiguroidaan valmiita

komponentteja, joilla voidaan rakentaa esimerkiksi täysin toimiva hälytysjärjestelmä ja käyttäjähallinta. Mapp-komponentit jaetaan viiteen eri kategoriaan: palvelut, ohjaus, näkymä, turvallisuus ja liike. Palvelu eli mappServices-komponentit tarjoavat valmiita ratkaisuja reseptin hallintaan, käyttäjien ja oikeuksien hallintaan, auditointiin ja moneen muuhun. Ohjaus eli mappControl-komponentit tarjoavat valmiita komponentteja eri ohjaustilanteisiin kuten hydrauliikkaohjauksiin. Näkymä eli mappView-komponenteilla voidaan rakentaa HTML5, CSS ja JavaScript -pohjainen käyttöliittymä valmiilla komponenteilla (ks. kuvio 15). Tarjolla on valmiita painikkeita, numeron- ja tekstinsyöttökenttiä, taulukoita ja paljon muita komponentteja. Turvallisuus eli mappSafety-komponenteilla voidaan kehittää turva-automaatioratkaisuja. Liike eli mappMotion-komponenteilla puolestaan kehitetään monimutkaisempia liikeratoja vaativia toteutuksia kuten robotteja ja CNC-sovelluksia. (Automation Studio 2014, 21; Automation Studio n.d.; Mapp Technology n.d.)

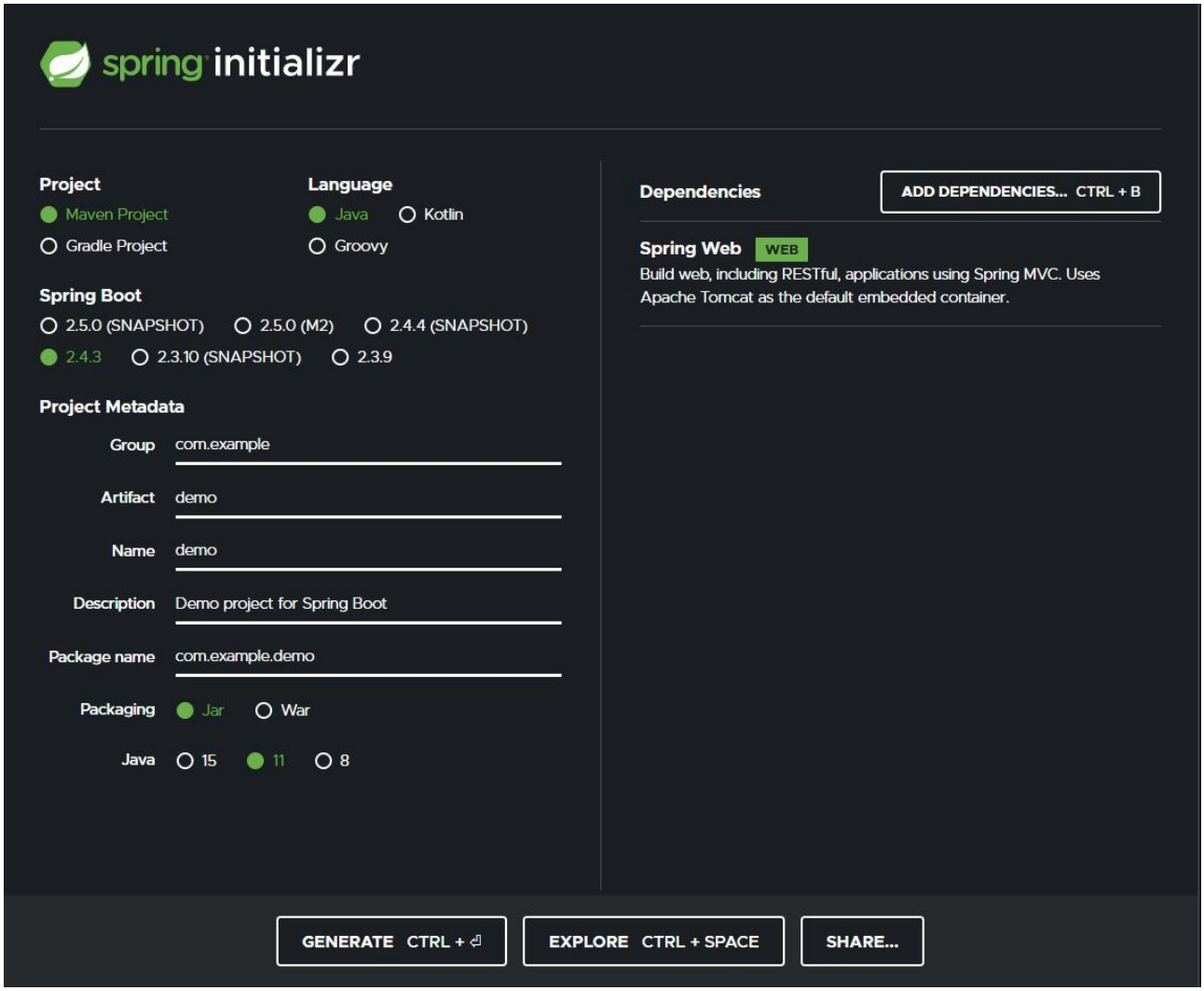
Kuvio 15. mappView-komponenteilla rakennettu testinäkymä (Web meets automation n.d.)

6 Java Spring Boot

Java Spring Framework on suosittu avoimen lähdekoodin yritystason ohjelmistokehys itsenäisten, tuotanto tason sovellusten luomiseen. Java Spring Boot puolestaan on työkalu, joka tekee verkkosovellusten ja mikropalvelujen kehittämisen Spring Framework -ohjelmistokehyksen avulla

helpommaksi ja nopeammaksi. Työkalun hyötyjä ovat automaattinen konfigurointi, harkitseva lähestymistapa konfigurointiin ja kyky luoda itsenäisiä sovelluksia. Spring Boot -työkalulla voidaan tehdä Spring-pohjainen sovellus mahdollisimman pienellä konfiguroinnilla. Se ei kuitenkaan rajoita Spring Framework -ohjelmistokehyksen käyttöä. Se on yhtä kattava, mutta sovellusten konfigurointi, asentaminen ja käyttöönotto vaatii vähemmän aikaa. (Java Spring Boot n.d.; Spring Boot n.d.)

Automaattinen konfigurointi tarkoittaa sitä, että sovellukset alustetaan ennalta asetetuilla riippuvaisuuksilla. Spring Boot -työkalussa on sisäänrakennetut automaattiset määritysominaisuudet, joilla määritetään sekä taustalla olevat Spring Framework -ohjelmistokehyksen että kolmannen osapuolen paketit asetuksiin. Oletusasetukset voidaan ohittaa, kun alustus on valmis. Harkittu lähestymistapa konfigurointiin tarkoittaa sitä, että Spring Boot -työkalu lisää ja määrittää käynnistysriippuvaisuuksia sovelluksen tarpeen mukaan. Spring Boot valitsee asennettavat paketit ja käytettävät oletusarvot ohjelmistokehittäjän sijaan. Ohjelmistokehittäjän ei tarvitse tehdä päätöksiä paketeista tai määritellä niitä. Ohjelmistokehittäjä voi määritellä sovelluksen tarpeet alustusprosessin (Spring Initializr) aikana, jonka aikana voidaan valita monista käynnistysriippuvaisuuksista (Spring Starter), jotka kattavat tyypillisimmät käyttötapaukset (ks. kuvio 16). Spring Boot sisältää yli 50 käynnistysriippuvaisuutta ja lisäksi valikoiman kolmannen osapuolen käynnistimiä. Spring Boot -työkalun kyky luoda itsenäisiä sovelluksia puolestaan tarkoittaa sitä, että sovelluksia voidaan käyttää alustasta riippumatta. Ulkoisen web-palvelimen sijasta upotetaan web-palvelin sovellukseen. Sovelluksia on mahdollista käyttää missä vain pelkällä suorita-komennolla. (Java Spring Boot n.d.; Spring Boot n.d.)



Kuvio 16. Spring Initializr alustustyökalu (Spring Initializr n.d.)

7 Toteutuksen eteneminen

7.1 Suunnittelu

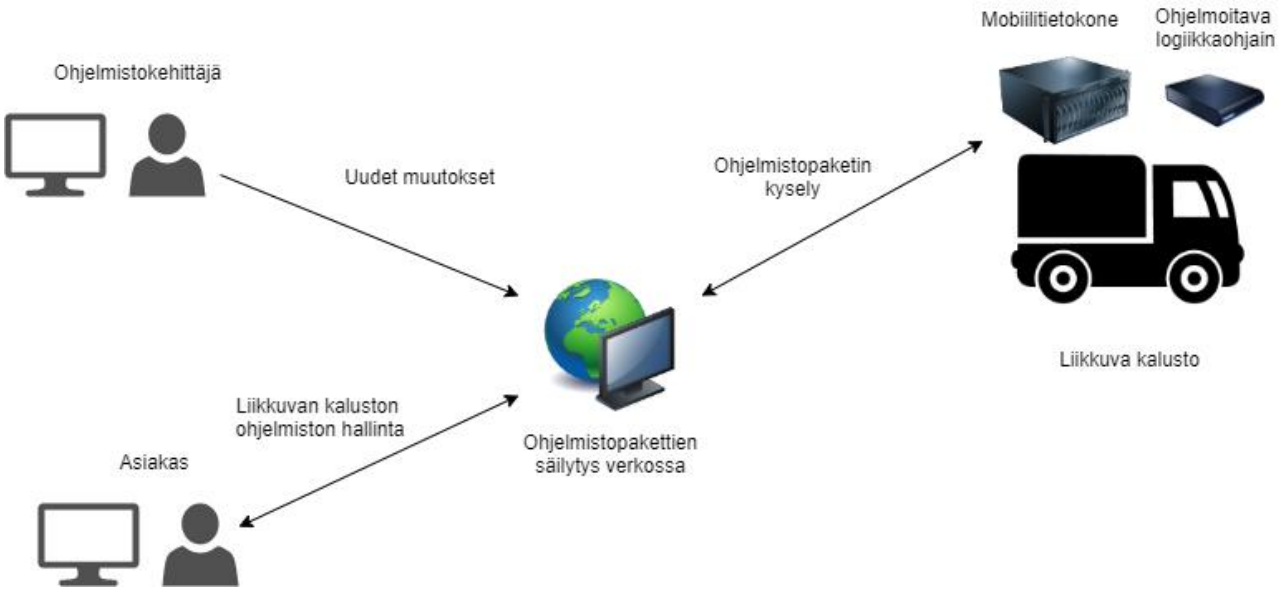
Toteutus työstäminen aloitettiin kattavalla suunnittelulla. Ohjelmiston hallintaa suunniteltiin ja vaatimuksia määriteltiin asiakkaan kanssa kahden viikon aikana useassa, tunteja kestävissä palavereissa. Vaatimuksien pohjalta kirjoitettiin karkea toteutusehdotus, joka hyväksytettiin asiakkaalla.

Ohjelmistosta on pidettävä kokoajan saatavilla toimitettava versio, joka edellyttää tuotantoversion pitämistä erillään kehitysversiosta. Ohjelmistoa kehittää useampi ohjelmistokehittäjä, joten muutoksia tulee päivittäin. Uusi tuotantoversio pitää testata automaattisesti ja olla julkaisun

jälkeen saatavilla verkossa. Lisäksi on asiakkaan päätettävissä, mitä ohjelmistoversiota mikäkin liikkuva kone käyttää.

Liikkuvassa koneessa on ohjelmoitava logiikkaohjain ja ajoneuvoihin suunniteltu mobiilitietokone, jossa on Linux-käyttöjärjestelmä. Ohjelmoitava logiikkaohjain kyselee mobiilitietokoneelta uutta ohjelmistoversiota tietyin aikavälein. Mobiilitietokone puolestaan kyselee tietyin aikavälein uutta ohjelmistoversiota verkon yli. Kysely on tehtävä tietoturvallisesti eli mobiilitietokone on todennettava. Mobiilitietokone muodostaa verkkoyhteyden langattomasti 4G:llä. Sillä ei kuitenkaan aina ole verkkoyhteyttä käytettävissä, koska kone saattaa olla 4G-verkon kuuluvuuden ulkopuolella. Liikkuvan kaluston verkossa olemista seurataan näiden kyselyjen avulla. Mobiilitietokoneen tulee lisäksi säilyttää neljä aiemmin ladattua versiota, jotta ohjelmoitavalle logiikkaohjaimelle voidaan tarvittaessa asentaa vanhempi versio.

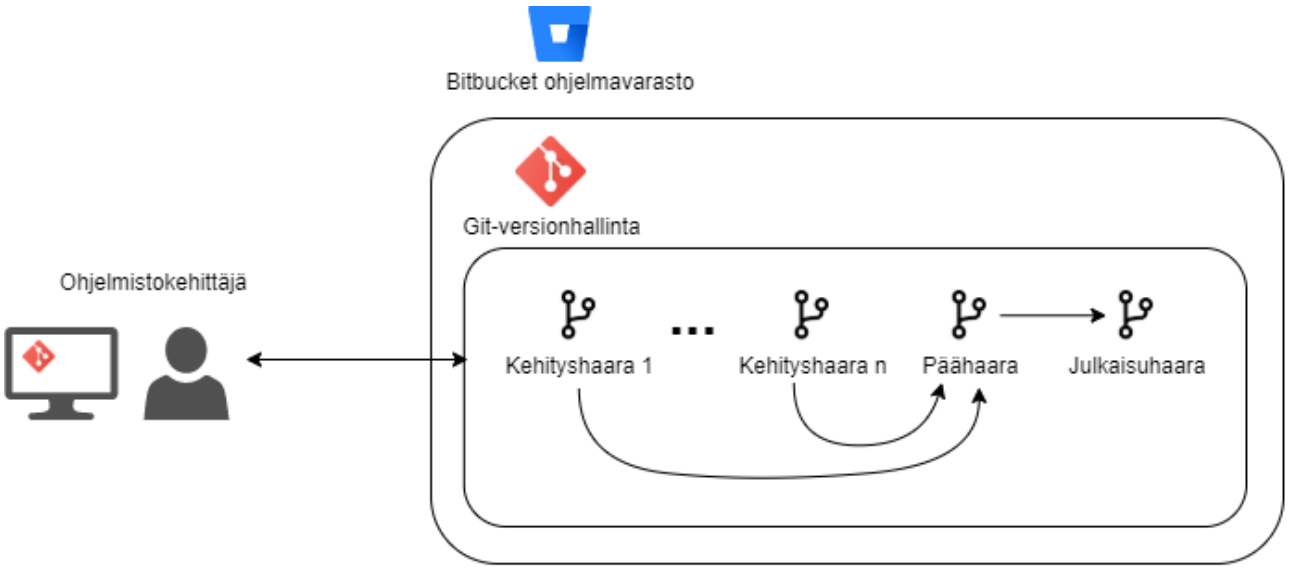
Asiakas haluaa tietää, mikä ohjelmistoversio ohjelmoitavalle logiikkaohjaimelle on asennettu. Ohjelmoitava logiikkaohjain ilmoittaa mobiilitietokoneelle, mikä versio siihen on asennettuna. Mobiilitietokone ilmoittaa asennetun version verkon yli aina, kun se kyselee uutta versiota. Tämä tieto tulee tallentaa siten, että asiakas pääsee siihen käsiksi. Kuviossa 16 on esitetty hahmotelma laitteista ja niiden välisistä yhteyksistä.



Kuvio 17. Hahmotelma laitteista ja niiden välisistä yhteyksistä

7.2 Toteutus

Ohjelmistokehityksessä käytettiin jatkuva integrointi ja jatkuva toimittaminen -ohjelmistokehityskäytäntöjen menetelmiä. Versionhallinta toteutettiin Git-versionhallinnalla ja ohjelmistovarastoksi valittiin Atlassian Bitbucket -palvelu. Hajautettu versionhallintajärjestelmä mahdollisti usean eri ohjelmistokehittäjän samanaikaisen ohjelmistokehityksen ja useiden eri haarojen käyttämisen. Ohjelmistokehittäjät integroivat muutoksia versionhallinnan päähaaraan päivittäin. Uusia ominaisuuksia kehitettiin suoraan päähaaraan ja erillisissä kehityshaaroissa, jotka yhdistettiin lopulta päähaaraan. Uuden ohjelmistoversion julkaisu tehtiin erillisestä julkaisuhaarasta, koska kaikkia päähaaran versioita ei haluttu julkaista liikkuvan kaluston käyttöön (ks. kuvio 18).

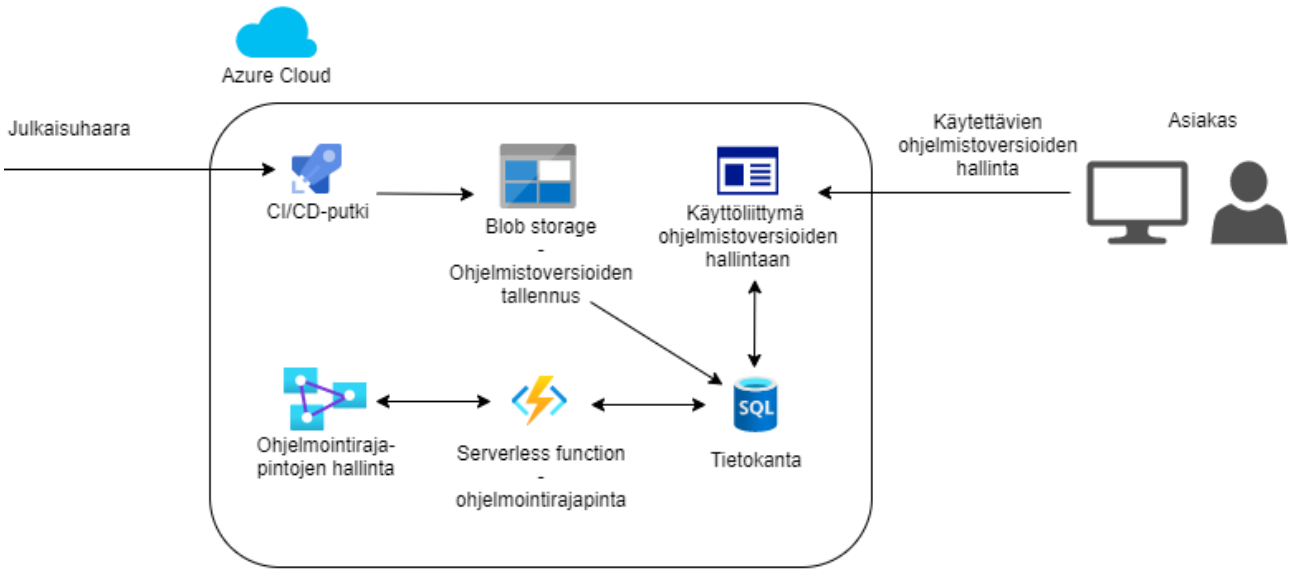


Kuvio 18. Versionhallinnan ja ohjelmavaraston käyttäminen

Päähaaraan tuleva muutos laukaisee CI/CD-putken käyntiin, jolla automatisoidaan ohjelmiston koostaminen ja testaaminen. CI/CD-putkia tarvittiin kaksi: päähaaralle ja haaralle, josta tehdään julkaisut. CI/CD-putket kehitettiin Microsoft Azure -pilvipalveluun. Päähaaran muutoksista laukeaa käyntiin CI/CD-putki, joka hakee päähaarasta lähdekoodin, koostaa sen ja ajaa automatisoidut yksikkötestit. Julkaisuhaaran muutoksista laukeaa käyntiin CI/CD-putki, joka edellä mainittujen vaiheiden lisäksi julkaisee koostetun ohjelmistoversion Azure-pilvipalvelun Blob Storage -palveluun, jota käytetään niiden säilyttämiseen. Kun Blob Storage -palveluun lisätään uusi ohjelmistoversio, sen Blob Storage -tunniste lisätään relaatiotietokantaan. Tietokantaan

tallennetaan kaikkien Blob Storage -palvelussa olevien ohjelmistoversioiden tunnisteet, mitä ohjelmistoversioita halutaan liikkuvan kaluston ohjelmoitavien logiikkaohjaimien käyttävän ja mitä ohjelmistoversioita niihin on asennettuna.

Liikkuvan kaluston mobiilitietokoneen kyselyt uusista ohjelmistoversioista menevät Azure-pilvipalveluun ohjelmointirajapintahallinnan kautta ohjelmointirajapinnoille. Ohjelmointirajapinta tallentaa kyselyn mukana tulleen tiedon ohjelmoitavaan logiikkaohjaimeen asennetusta ohjelmistoversiosta relaatiotietokantaan ja sen jälkeen tarkistaa samasta tietokannasta, onko kyseiselle laitteelle tarjolla uudempaa ohjelmistoversiota ladattavaksi. Jos uusi ohjelmistoversio löytyy, ohjelmointirajapinta palauttaa Blob Storage -tunnisteen kyselyn vastauksessa mobiilitietokoneelle (ks. kuvio 19).

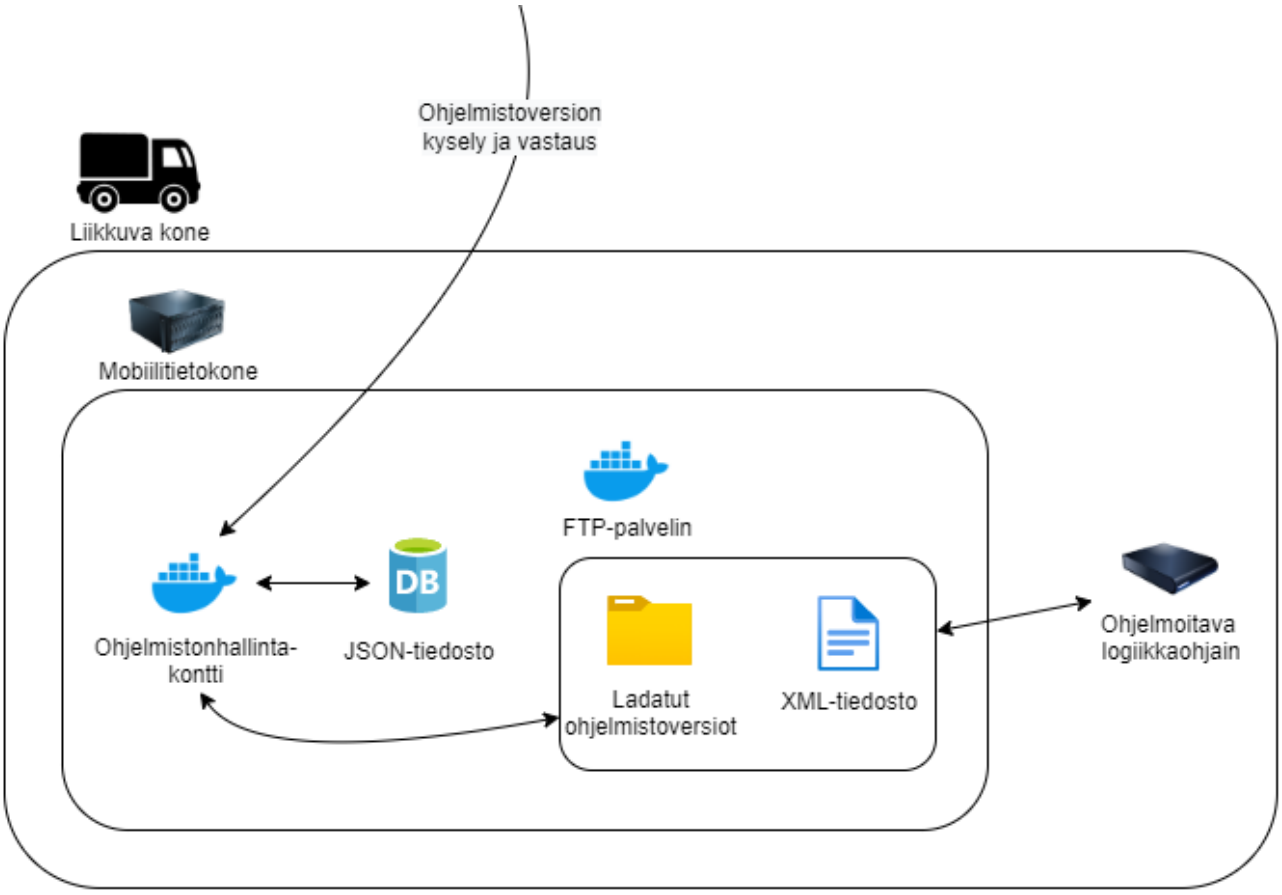


Kuvio 19. Azure-pilvipalvelun komponentit

Mobiilitietokoneen täytyy tunnistautua Azure-pilvipalveluun, tehdä säännöllisiä pyyntöjä ohjelmointirajapinnalle, ladata ohjelmistoversioita, käsitellä ladattuja ohjelmistoversioita ja olla tietoinen, mikä ohjelmistoversio on ohjelmoitavaan logiikkaohjaimeen asennettuna. Kyseisten tehtävien suorittamiseksi kehitettiin sovellus, jota ajetaan Docker-kontissa. Sovellus tunnistautuu Azure-pilvipalveluun. Se tekee tietyin aikavälein HTTP-pyyntöjä ohjelmointirajapinnalle. Kun ohjelmointirajapinta palauttaa Blob Storage -tunnisteen uuteen ohjelmistoversioon, se ladataan toisella ohjelmointirajapinnalla ja tieto ladatusta ohjelmistoversiosta tallennetaan paikalliseen

JSON-tiedostoon. Tämän tiedoston avulla ylläpidetään paikallista ohjelmistovarastoa, jossa halutaan pitää neljä uusinta versiota. Itse ohjelmistoversiot puolestaan tallennetaan FTP-palvelimelle, jota ajetaan erillisessä Docker-kontissa.

FTP-palvelin toimii mobiilitietokoneen ja ohjelmoitavan logiikkaohjaimen välikätenä, koska ohjelmoitava logiikkaohjain voi lukea tiedostoja FTP-palvelimelta. XML-tiedostoa lukemalla Docker-kontissa pyörivä sovellus tietää, mikä ohjelmistoversio ohjelmoitavaan logiikkaohjaimeen on asennettuna. Kun uusi ohjelmistoversio on siirretty FTP-palvelimelle, ohjelmoitava logiikkaohjain saa tiedon tästä XML-tiedoston elementistä, jolla indikoidaan ohjelmistopäivityksen tarvetta. Uuden ohjelmistoversion asentaminen ohjelmoitavaan logiikkaohjaimeen aloitetaan kuitenkin käsin sen käyttöliittymästä ja huoltohenkilön toimesta. Asennuksen jälkeen ohjelmoitava logiikkaohjain kirjoittaa XML-tiedostoon asennetetun ohjelmistoversion ja sitä edeltävän ohjelmistoversion palautusversioksi sekä resetoi elementin, joka indikoi ohjelmistopäivityksen tarpeesta (ks. kuvio 20).



Kuvio 20. Mobiilitietokoneen tehtävät

7.3 Testaaminen

Kehityksen aikainen testaaminen suoritettiin toimistolla. Testaamiseen käytettiin ohjelmistokehittäjän tietokonetta, Bitbucket-palvelua, Azure-pilvipalvelua, Linux-käyttöjärjestelmällä olevaa tietokonetta ja ohjelmoitavaa logiikkaohjainta. Kyseisillä laitteilla pystyttiin simuloimaan tilanne, jossa ohjelmistokehittäjä teki uuden ohjelmistoversion julkaisun ja mobiilitietokoneelta lähetettiin HTTP-pyyntöjä ohjelmointirajapinnalle. Lopullinen testaaminen tehtiin siten, että liikkuvan koneen verkkoyhteyttä katkottiin ja testattiin eri skenaarioita mm. latauksen keskeytyminen ja ohjelmistoversion korruptoituminen.

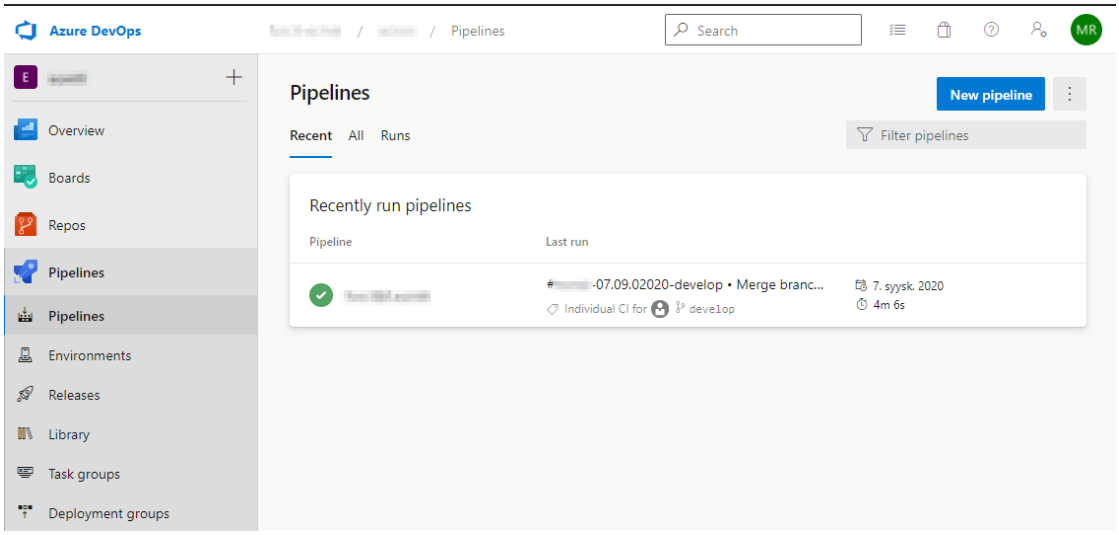
8 Toteuttaminen käytännössä

8.1 Kehitystyön vaiheet

Kehitystyö jaettiin kolmeen eri vaiheeseen. Ensimmäisenä kehitettiin CI/CD-putki, jossa ohjelmistoversio koostetaan, testataan ja siirretään Azure-pilvipalvelun Blob Storage -palveluun. Seuraavaksi kehitettiin ohjelmointirajapinta Azure-pilvipalveluun, joka palauttaa pyyntöjä lähettävälle ohjelmistoversion ja sen Blob Storage -tunnisteen. Kolmantena kehitettiin mobiilitietokoneelle sovellus, jota ajetaan Docker-kontissa. Se hoitaa HTTP-pyyntöjen lähettämisen ohjelmointirajapinnoille ja ohjelmistoversioiden hallinnan mobiilitietokoneella. Lisäksi mobiilitietokoneelle pystytettiin FTP-palvelin, jota ajetaan erillisessä Docker-kontissa.

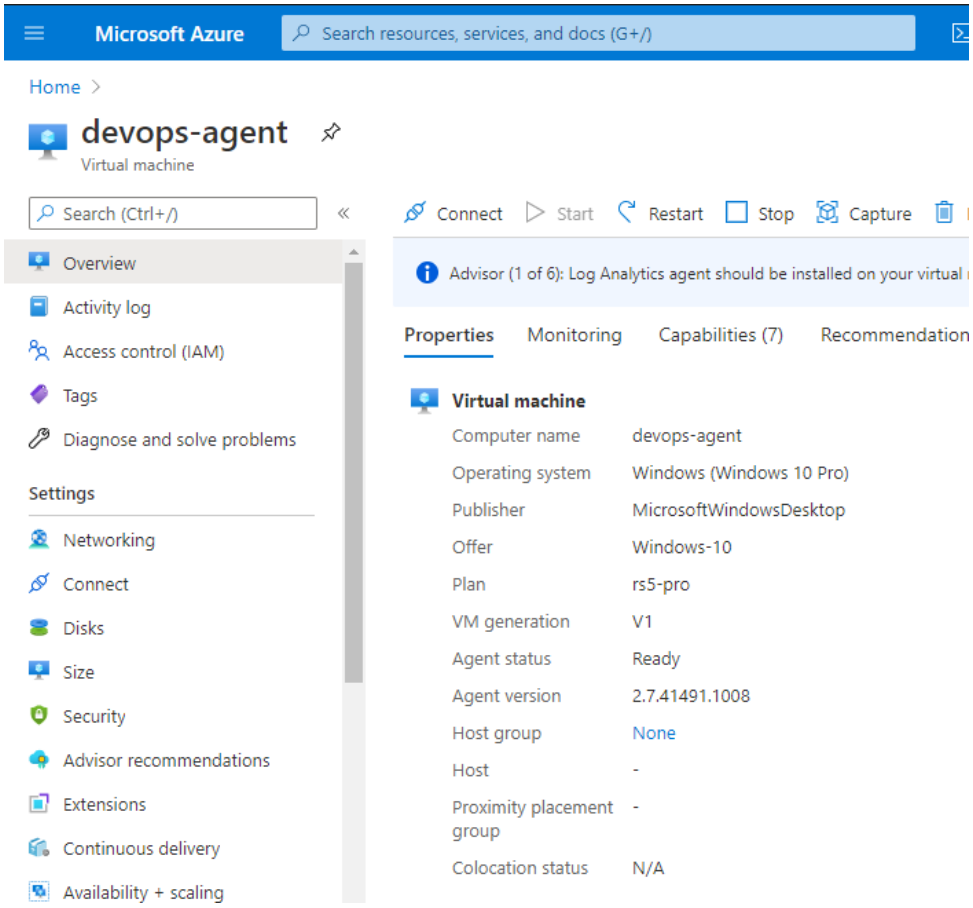
8.2 CI/CD-putken tekeminen

CI/CD-putki tehtiin Pipeline-palvelulla, johon erilaisia putkia voitiin määritellä YAML-tiedostoilla. Uutta putkea tehdessä valittiin palvelu, jossa lähdekoodia ja YAML-tiedostoa säilytetään. Tässä tapauksessa lähdekoodi ja YAML-tiedosto ovat Bitbucket-palvelun ohjelmavarastossa. Kuviossa 21 on Pipeline-palvelun sivu, jossa voidaan lisätä, monitoroida ja muokata putkia.



Kuvio 21. Pipeline-palvelu

Putken suorittamista varten tarvitaan resursseja eli agentti ja agenttia varten tarvitaan agenttivaranto. DevOps-palvelussa luotiin ensimmäisenä agenttivaranto. Agenttivarannon konfigurointisivulta ladattiin tiedosto, jolla virtuaalikone konfiguroitiin agentiksi. Sen jälkeen Azure-pilvipalvelussa luotiin Windows 10 -käyttöjärjestelmällä oleva virtuaalikone, jota käytetään putkien agenttina. Virtuaalikoneelle siirrettiin agenttivarannon konfigurointisivulta ladattu tiedosto, joka ajettiin komentokehotteella. Tiedosto konfiguroi virtuaalikoneen agentiksi ja sen jälkeen virtuaalikone on osa agenttivarantoa. Kuviossa 21 on virtuaalikone, jota käytetään CI/CD-putken agenttina.



Kuvio 22. CI/CD-putken agentti

YAML-tiedostossa määritellään mitä putki tekee ja mikä agentti sitä ajaa. Tiedostossa on määritelty putken toiminnan laukaisijaksi ohjelmavaraston julkaisuhaara ja agenttivarannoksi aikaisemmin luotu agenttivaranto, joka sisältää yhden agentin. Seuraavaksi tiedostossa on määritelty putken toiminnallinen osuus (ks. kuvio 23). Putki koostuu askeleista, joissa osassa ajetaan PowerShell-skriptejä agentilla. Ensimmäiset kolme askelta ovat PowerShell-skriptejä: ohjelmiston koostaminen, ohjelmiston käyttöönotto simulaattorilla ja yksikkötestien ajaminen. Yksikkötestien jälkeisessä askeleessa testitulokset julkaistaan määriteltyyn kansioon JUnit-formaatissa. Lopuksi tiedostot kopioidaan Azure Blob Storage -palveluun. Päähaaran muutoksesta laukaistava CI/CD-putki vastaa julkaisuhaaran muutoksesta laukaistavaa CI/CD-putkea, mutta tiedostoja ei kopioida Azure Blob Storage -palveluun.

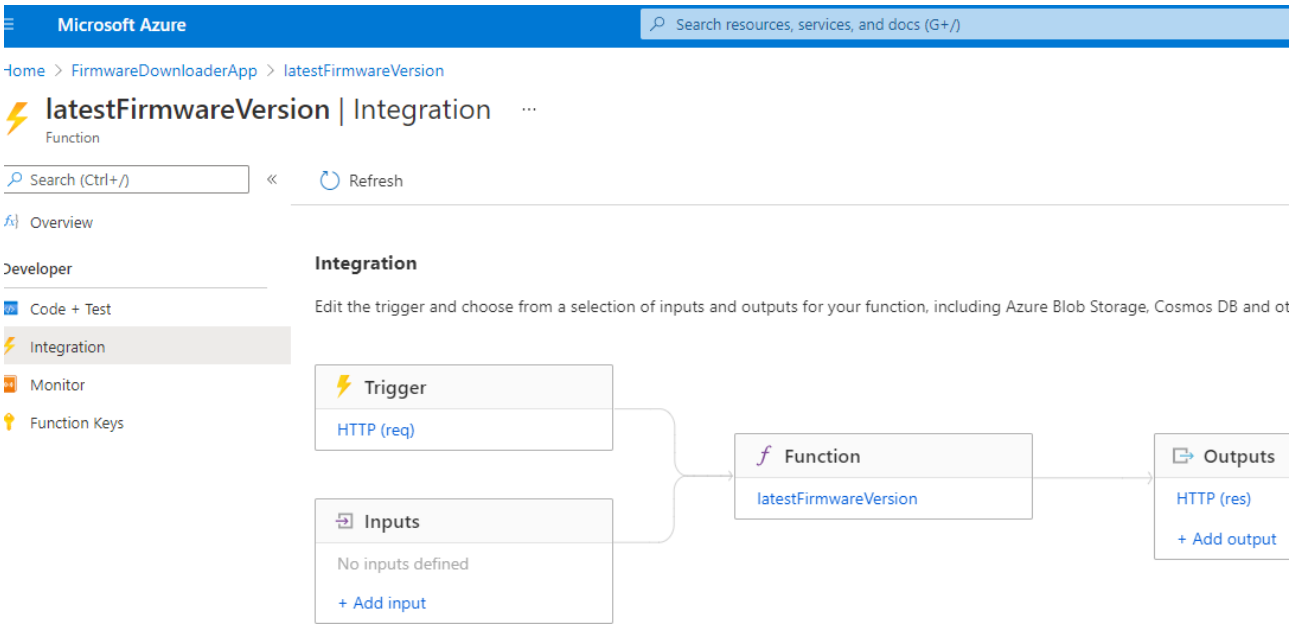
```yaml
steps:
  - checkout: self
    clean: true

  - task: PowerShell@2
    inputs:
      targetType: filepath
      filePath: "scripts/build_[redacted].ps1"
      arguments: "-c development"
      displayName: 'Build [redacted]'

  - task: PowerShell@2
    inputs:
      targetType: filepath
      filePath: "scripts/deploy_to_sim.ps1"
      displayName: 'Deploy package to simulator'

  - task: PowerShell@2
    inputs:
      targetType: filepath
      filePath: "scripts/run_tests_after_deploy.ps1"
      arguments: "-s 40"
      displayName: 'Run tests'

  - task: PublishTestResults@2
    inputs:
      testResultsFormat: 'JUnit'
      testResultsFiles: 'src/[redacted]_application/build/test-results.xml'
      failTaskOnFailedTests: true
      testRunTitle: '[redacted]-Tests'
      displayName: 'Publish test results'
```

Kuvio 23. Osa CI/CD-putken toiminnallisuudesta

8.3 Ohjelmointirajapinnat

Ohjelmistoversion kyseleminen ja lataaminen Azure-pilvipalvelusta tehtiin kahdella ohjelmointirajapinnalla, joita käytetään HTTP GET-pyynnöillä. Azure-pilvipalvelun API Management -palvelussa määriteltiin kaikille ohjelmointirajapinnoille yhteisen yhdyskäytävän URL-osoite. Sen jälkeen määriteltiin ohjelmointirajapinta FirmwareDownloadApp ohjelmistoversion kyselylle ja DownloadAPI lataamiselle. Molemmille ohjelmointirajapinnoille on määritelty oma polku, kyselyparametrit ja pyyntöjen ja vastausten prosessointi.

Liikkuvan koneen uuden ohjelmistoversion selvittäminen toteutettiin Azure-pilvipalvelussa palvelimettomalla funktiolla, joka tehtiin Azure Function -palvelulla. Funktio on nimetty samalla nimellä kuin ohjelmointirajapinta eli FirmwareDownloadApp. Funktion ajoympäristöksi on määritetty Node.js. Funktion laukaisijaksi puolestaan on määritetty HTTP GET-pyyntö (ks. kuvio 24). Parametreina funktiolle annetaan liikkuvan koneen tunniste, jonka asiakas on määritellyt kaikille koneille, ja ohjelmoitavaan logiikkaohjaimeen asennetun ohjelmistoversio. Funktioon on määritelty ympäristömuuttujina Azure PostgreSQL -relaatiotietokannan yhteys- ja tunnistetiedot. Funktiossa käytetään node-postgres -kirjastoa tietokannan yhteyden luomiseen ja kyselyihin. Tietokantaan tehdään kysely käyttäen HTTP-pyynnön mukana tullutta liikkuvan koneen tunnistetta ja asennettua ohjelmistoversiota. Funktio kirjoittaa relaatiotietokantaan asennetun ohjelmistoversion ja vertailee sitä uusimpaan saatavilla olevaan ohjelmistoversioon. Jos uusi ohjelmistoversio on saatavilla kyseiselle liikkuvalle koneelle, funktio palauttaa vastauskoodin 200 ja vastauksen rungossa ladattavan ohjelmistoversion ja sen Azure Blob Storage -palvelun tunnisteen. Mikäli uutta ohjelmistoversiota koneen tunnisteella ei löydy, palautetaan vastauskoodi 404.



Kuvio 24. Ohjelmistopaketin version kysely -ohjelmointirajapinta Azure Function -palvelussa

Ohjelmistoversion kysely -ohjelmointirajapintaan tulevat HTTP GET-pyynnöt välitetään FirmwareDownloadApp funktiolle. Ohjelmistoversion kysely -ohjelmointirajapintaan määritellään

poluksi /FirmwareDownloaderApp/latestFirmwareVersion. Kyselyparametreiksi on asetettu machineId eli liikkuvan koneen tunniste ja installedVersion eli ohjelmoitavaan logiikkaohjaimeen asennettu ohjelmistoversio.

Ohjelmistoversion lataus -ohjelmointirajapinnalle ei määritelty erillistä polkua. Pyynnön parametriksi on määritetty firmwareId eli ohjelmistoversion Azure Blob Storage -palvelun tunniste, joka saadaan ohjelmistoversion kysely -ohjelmointirajapinnalta vastaukseksi, jos uusi ohjelmistoversio on ladattavissa. HTTP GET-pyyntö ohjataan Azure Blob Storage -palveluun, joka lataa kyseisen ohjelmistoversion mobiilitietokoneelle.

Ohjelmointirajapintojen tietoturvallisuus on toteutettu avaimella ja todentamisella. Ohjelmointirajapintojen HTTP-pyynnöissä lähetetään avain, jonka API Management -palvelu tunnistaa. Lisäksi liikkuva kone tunnistautuu Azure-palveluun OAuth2.0-todentamisella. Tunnistautuminen tietoturvallisuuden lisäämisen lisäksi mahdollistaa liikkuvan kaluston ohjelmointirajapintojen HTTP-pyyntöjen monitoroinnin API Management -palvelusta yksilö tasolla. HTTP-pyynnöillä seurataan liikkuvan kaluston verkkoyhteydessä olemista.

8.4 Ohjelmiston hallinta ja asentaminen

Ohjelmoitavan logiikkaohjaimen ohjelmistoversioiden hallinta mobiilitietokoneella ja niiden asentaminen ohjelmoitavaan logiikkaohjaimeen on toteutettu sovelluksella, jota ajetaan Docker-kontissa. Sovellus on kehitetty Java-ohjelmointikielellä käyttäen Sprint Framework -ohjelmistokehystä Spring Boot -työkalulla. Kehitysympäristönä käytettiin JetBrainin kehittämää IntelliJ IDEA:ta.

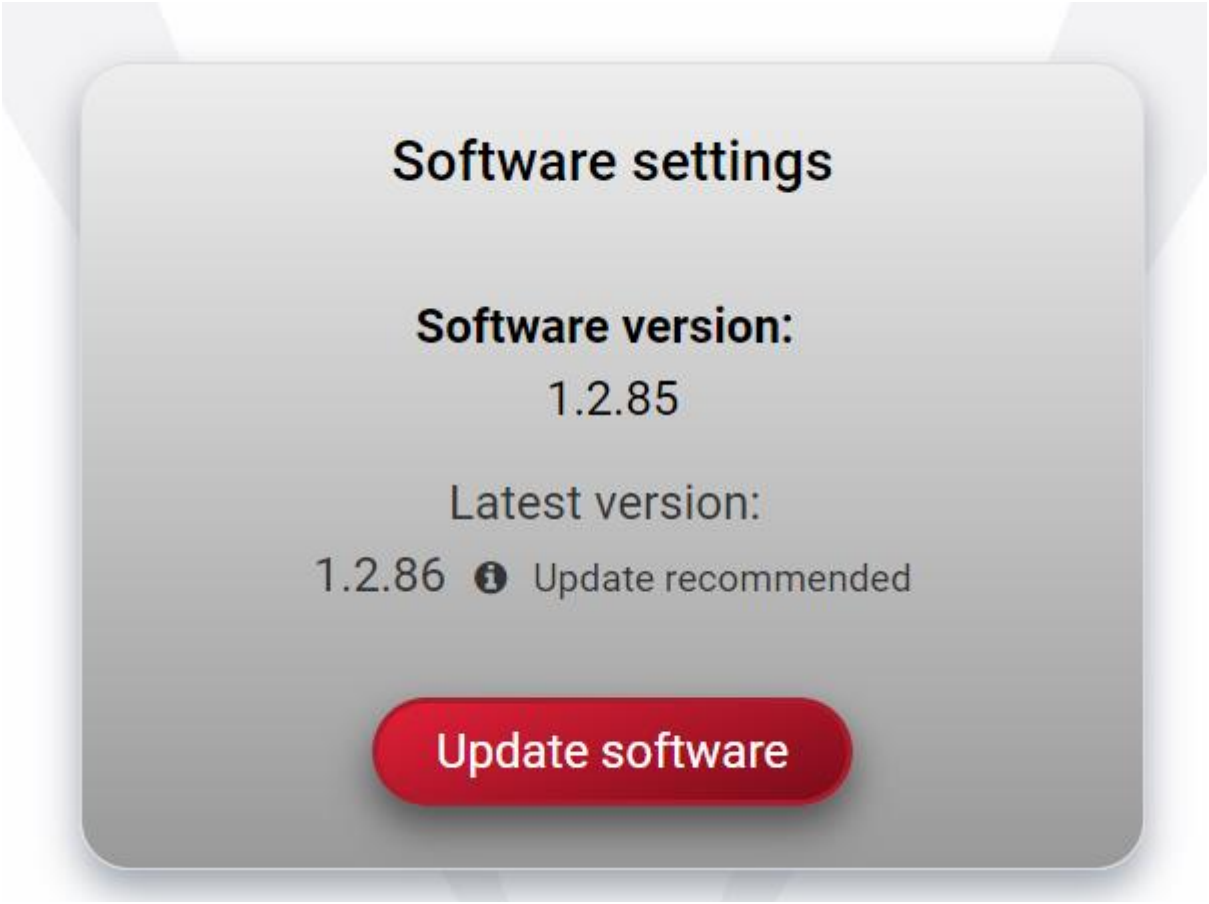
Sovellusta suoritetaan aina 15 minuutin välein. Ensimmäisenä se lukee FTP-palvelimelta XML-tiedostosta ohjelmoitavaan logiikkaohjaimeen asennetun ohjelmistoversion. Sen jälkeen tehdään HTTP-pyyntö ohjelmistoversion kysely -ohjelmointirajapintaan. Ohjelmointirajapinta palauttaa sovellukselle uuden ohjelmistoversion ja sen Azure Blob Storage -tunnisteen, jos sellainen on ladattavissa. Muuten funktio on suoritettu loppuun. Mobiilitietokone tarkastaa JSON-tietokannasta, onko ohjelmistoversio ladattu. Jos ohjelmistoversiota ei ole ladattu, luodaan mobiilitietokoneelle väliaikaistiedosto ladattavan ohjelmistoversion nimellä. Sen jälkeen ohjelmistoversion lataus väliaikaistiedostoon aloitetaan lähettämällä HTTP-pyyntö

ohjelmistoversion lataus -ohjelmointirajapintaan. Kuviossa 25 on ohjelmistoversion latauksen tekevä funktio. Onnistuneen latauksen jälkeen väliaikaistiedostosta tehdään kopio ja se siirretään FTP-palvelimelle. Väliaikaistiedosto poistetaan ja paikalliseen JSON-tietokantaan merkataan ladattu ohjelmistoversio. Jos JSON-tietokannassa on yli neljä versiota, kaikista vanhin versio poistetaan tietokannasta ja sen jälkeen FTP-palvelimelta. Lopuksi ohjelmoitavalle logiikkaohjaimelle ilmoitetaan XML-tiedoston avulla ohjelmistopäivityksen saatavuudesta.

```
35 @    public void downloadLatestFirmware(Firmware latestFirmwareVersion, File destinationFile) throws IOException {
36          String firmwareDownloadUrl = "https://firmware-downloader-api.azure-api.net/" + latestFirmwareVersion.getStorageIdentifier();
37
38          logger.info("Starting to download file '{}' as temp file", latestFirmwareVersion.getStorageIdentifier());
39          URL url = new URL(firmwareDownloadUrl);
40          URLConnection urlConnection = url.openConnection();
41          urlConnection.setRequestProperty("Ocp-Apim-Subscription-Key", "8a6a8255e54c4c20bf28005d7c600212");
42          urlConnection.setRequestProperty("User-Agent", "Mozilla/5.0 ( compatible ) ");
43          urlConnection.setRequestProperty("Accept", "*/*");
44          ReadableByteChannel readableByteChannel = Channels.newChannel(urlConnection.getInputStream());
45
46          FileOutputStream fileOutputStream = new FileOutputStream(destinationFile);
47          fileOutputStream.getChannel().transferFrom(readableByteChannel,  position: 0, Long.MAX_VALUE);
48
49          logger.info("Downloaded file '{}' as temp file", latestFirmwareVersion.getStorageIdentifier());
50      }
```

Kuvio 25. Ohjelmistoversion latauksen tekevä funktio

Ohjelmoitavan logiikkaohjaimen toiminnallisuus kuten myös koko ohjelmistoversio on kehitetty B&R Automation Studio -ohjelmistokehitysympäristössä. Ohjelmoitavalle logiikkaohjaimelle määriteltiin yhteys FTP-palvelimelle ja kehitettiin funktio, jota suoritetaan 15 minuutin välein. Funktiossa luetaan XML-tiedostoa FileIO-kirjaston FileOpen-, FileRead- ja FileClose-toimilohkoilla ja tarkistetaan XML-tiedoston elementistä, onko ohjelmistopäivitystä saatavilla. Kun ohjelmistopäivitys on saatavilla, ohjelmoitavan logiikkaohjaimen käyttöliittymässä olevalla päivityssivulla indikoidaan käyttäjää näyttämällä asennettu ohjelmistoversio ja sen alla suositus uuden ohjelmistoversion asentamista (ks. kuvio 26). Ohjelmistopäivityksen saa kuitenkin aloittaa vain käyttäjä, jolla on huoltohenkilön oikeudet. Ohjelmistopäivityksen asentaminen toteutetaan myös valmiilla mappServicen MpBackupCore-komponentilla. Komponentille annetaan polku FTP-palvelimen kansioon, jossa ohjelmistoversioita säilytetään ja asennettavan ohjelmistoversiotiedoston nimi merkkijonona. Kun ohjelmistoversion asennus on valmis, ohjelmoitava logiikkaohjain merkkaa XML-tiedostoon uuden version ja resetoi ohjelmistopäivitys elementin.



Kuvio 26. Ohjelmoitavan logiikkaohjaimen käyttöliittymä

9 Pohdinta

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa toimeksiantajan asiakkaalle vaatimusten mukainen kokonaisuus liikkuvassa kalustossa olevien ohjelmoitavien logiikoiden ohjelmiston hallintaan ja päivittämiseen käyttäen DevOps-menetelmiä. Henkilökohtaisena tavoitteena oli oppia enemmän DevOps-menetelmistä ja niiden hyödyntämisestä osana ohjelmistokehitystä.

Lopputuloksena käytännön työstä syntyi kokonaisuus, jolla hallitaan liikkuvan kaluston ohjelmoitavien logiikkaohjaimien ohjelmistoversioita pilvipalvelussa ja paikallisesti mobiilitietokoneissa. Mobiilitietokoneet voivat toteuttaa HTTP-pyyntöjä, jotka yksilöidään liikkuvan koneen tunnuksella, ja ladata niille määritellyn ohjelmistoversion tietoturvallisesti. Asiakkaalle tästä on suuri hyöty, sillä uudet ohjelmistoversiot voidaan testata halutuissa yksilöissä ja sitten vasta asettaa kyseinen ohjelmistoversio saataville koko liikkuvalle kalustolle.

Ohjelmistoversioiden asentamiseen ei tarvita enää USB-muistitikkuja tai muita lisävälineitä, kuten aiemman sukupolven liikkuvassa kalustossa.

Työstä syntynyt kokonaisuus täyttää asiakkaan sille asettamat vaatimukset. Ohjelmistokehityksessä sovellettiin jatkuva integrointi ja jatkuva toimittaminen DevOps-menetelmiä. Asiakas voi hallita liikkuvan kaluston ohjelmistoversioita verkossa ja määrittää kullekin liikkuvalle koneelle haluamansa version. Liikkuvassa kalustossa olevat mobiilitietokoneet lataavat pilvipalvelusta ohjelmistoversion tietoturvallisesti mobiilitietokoneelle, josta ohjelmoitava logiikkaohjain hakee sen asennusprosessin aikana.

Opinnäytetyöprosessi tarjosi erinomaisen mahdollisuuden tutustua DevOps-toimintamalliin, jonka suosio alalla on jatkuvassa kasvussa. Lisäksi pääsin olemaan mukana ketterän kehityksen projektitoiminnassa, josta tulee olemaan myös merkittävää hyötyä jatkossa.

Ohjelmiston hallintaan liittyvä kokonaisuus tarjosi hyvän mahdollisuuden olla mukana alusta loppuun eli suunnittelusta testaukseen asti. Suunnitteluvaiheen asiakaspalaverit, joissa vaatimuksia määriteltiin ja käytiin läpi eri riski- ja ongelmatilanteita, olivat erittäin hyödyllisiä ja silmiä avaavia. Toteutusvaiheen aikana pääsin perehtymään Azure-pilvipalvelun eri palveluihin, Git-versionhallintaa sekä muutamaan eri ohjelmointikieleen ja kehitystyökaluun.

Jatkokehityksen aikana tullaan asiakkaalle kehittämään käyttäjäystävällisempi käyttöliittymä ohjelmistoversioiden hallintaan Azure-pilvipalvelussa. Nyt toteutettu käyttöliittymä on MVP-toteutus, joka tulostaa relaatiotietokannan taulukot näkyviin ja käyttöliittymässä olevaan kenttään voidaan syöttää SQL-käskyjä, joilla lisätään ja muokataan taulukkojen sisältöä.

Jatkokehitysideana voisi olla toteuttaa mobiilitietokoneilla olevien Docker-konttien hallinta samalla tavalla kuin ohjelmoitavan logiikkaohjaimen ohjelmistopaketin versioiden hallinta. Jos mobiilitietokoneella olevat Docker-kontit tarvitsevat muutoksia, uudet versiot Docker-konteista voitaisiin lisätä CI/CD-putkella Azure Blob Storage -palveluun ja ohjelmointirajapintojen avulla mobiilitietokoneeseen. Tällä hetkellä Docker-kontit asennetaan mobiilitietokoneisiin tehtaalla ja ne päivitetään manuaalisesti etäyhteydellä.

Lähteet

About version control. N.d. Getting Started – About Version Control. Git-versiohallinnan dokumentaatio. Viitattu 11.2.2021.
https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control.

Automation Studio. N.d. Tuotteen esittelysivu valmistajan verkkosivuilla. Viitattu 14.2.2021.
https://www.br-automation.com/en/products/software/automation-studio/.

Automation Studio. 2014. Esittelylehti. Viitattu 17.2.2021.
https://www.br-automation.com/index.php?eID=dumpFile&t=f&f=2%3A%2FBRP4440000000000000289549&token=78172c5eb338a9595baadcbc5527918806c10269.

Bose, M. 2019. Kubernetes vs Docker – What Is the Difference? Artikkeli Nakivo yrityksen verkkosivuilla. Viitattu 14.2.2019.
https://www.nakivo.com/blog/docker-vs-kubernetes/.

Cammarano, A. 2017. Mapp Tutorial B&R Automation Studio. Video. Viitattu 17.2.2021.
https://www.youtube.com/watch?v=y_KycoSoCpY.

CI/CD Pipeline. 2019. A Gentle Introduction. Semaphore yrityksen artikkeli Medium-julkaisualustalla. Viitattu 14.2.2021.
https://semaphoreci.medium.com/ci-cd-pipeline-a-gentle-introduction-3c340f578a72.

Containers at Google. N.d. A better way to develop and deploy applications. Artikkeli Google Cloud verkkosivulla. Viitattu 14.2.2021.
https://cloud.google.com/containers.

Continuous Delivery. 2018. Continuous Delivery: Everything you need to know. Artikkeli LogiGear-yrityksen blogissa. Viitattu 11.2.2021.
https://www.logigear.com/blog/continuous-delivery-devops/continuous-delivery-everything-you-need-to-know/.

Devecto. N.d. Devecto Oy yrityksen verkkosivut. Viitattu 11.2.2021.
https://devecto.com/devecto/.

DevOps and Application Security. N.d. Netsparker yrityksen verkkosivut. Viitattu 11.2.2021.
https://www.netsparker.com/devops-security-tools/.

Docker overview. N.d. Docker ohjelmiston dokumentaatio. Viitattu 14.2.2021.
https://docs.docker.com/get-started/overview/.

Fowler, M. 2006. Continuous Integration. Artikkeli. Viitattu 11.2.2021.
https://martinfowler.com/articles/continuousIntegration.html.

Fowler, M. 2013. Continuous Delivery. Artikkeli. Viitattu 11.2.2021. https://martinfowler.com/bliki/ContinuousDelivery.html.

Goyal, A. 2020. What is CI/CD Implementation? Artikkeli Octal Sotware yrityksen verkkosivuilla. Viitattu 11.2.2020. https://www.octalsoftware.com/blog/steps-to-implement-ci-cd-pipeline.

Guckenheimer, S. 2017. What is Continuous Integration? Microsoft Azure DevOps dokumentaatio. Viitattu 11.2.2021. https://docs.microsoft.com/en-us/azure/devops/learn/what-is-continuous-integration.

Guckenheimer, S. 2018. What is DevOps? Microsoft Azure DevOps dokumentaatio. Viitattu 11.2.2021. https://docs.microsoft.com/en-us/azure/devops/learn/what-is-devops.

Java Spring Boot. N.d. Artikkeli IBM yrityksen verkkosivuilla. Viitattu 24.2.2021. https://www.ibm.com/cloud/learn/java-spring-boot.

Key strategies for securing the hybrid cloud. 2018. Artikkeli Trend Micro yrityksen verkkosivuilla. Viitattu 14.2.2021. https://www.trendmicro.com/vinfo/us/security/news/virtualization-and-cloud/key-strategies-for-securing-the-hybrid-cloud.

Mapp Technology. N.d. Mapp-teknologian esittelysivu. Viitattu 17.2.2021. https://www.br-automation.com/en-gb/products/software/mapp-technology/.

Pittet, S. N.d. Continuous integration vs. continuous delivery vs. continuous deployment. Artikkeli. Viitattu 11.2.2021. https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment.

Practical AWS Diagram tutorial and examples. N.d. Opas Visual Paradigm yrityksen verkkosivuilla. Viitattu 14.2.2021. https://www.visual-paradigm.com/guide/cloud-services-architecture/what-is-aws-architecture/.

Serverless computing. N.d. An introduction to serverless technologies. Artikkeli Microsoft Azure verkkosivuilla. Viitattu 14.2.2021. https://azure.microsoft.com/en-us/overview/serverless-computing/.

Spring Boot. N.d. Spring Boot dokumentointi. Viitattu 24.2.2021. https://spring.io/projects/spring-boot#overview.

Spring Initializr. N.d. Spring Boot alustustyökalu. Viitattu 24.2.2021. https://start.spring.io/.

Web meets automation. N.d. Artikkeli. Viitattu 17.2.2021. https://www.br-automation.com/en-gb/about-us/press-room/technology-highlights/mapp-technology-highlights/web-meets-automation/.

What are cloud services? N.d. Artikkeli RedHat-yrityksen verkkosivuilla. Viitattu 14.2.2021. https://www.redhat.com/en/topics/cloud-computing/what-are-cloud-services.

What is a CI/CD pipeline? N.d.a Artikkeli RedHat-yrityksen verkkosivuilla. Viitattu 11.2.2021. https://www.redhat.com/en/topics/devops/what-cicd-pipeline.

What is a CI/CD pipeline? N.d.b. Opetus artikkeli Guru99 verkkosivulla. Viitattu 11.2.2021. https://www.guru99.com/ci-cd-pipeline.html.

What is a Container? N.d. A standardized unit of software. Docker yrityksen verkkosivut. Viitattu 14.2.2021.
https://www.docker.com/resources/what-container.

What is cloud computing? N.d. A beginner's guide. Opas Microsoftin Azure-tuotteen verkkosivuilla. Viitattu 14.2.2021.
https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/.

What is Continuous Delivery? N.d. Amazon AWS -pilvipalvelun verkkosivut. Viitattu 11.2.2021. https://aws.amazon.com/devops/continuous-delivery/.

What is Continuous Integration? N.d. Artikkeli PagerDuty yrityksen verkkosivuilla. Viitattu 11.2.2021.
https://www.pagerduty.com/resources/learn/what-is-continuous-integration/.

What is DevOps? N.d. Amazon AWS -pilvipalvelun verkkosivut. Viitattu 11.2.2021. https://aws.amazon.com/devops/what-is-devops/.

What is Docker? N.d. Artikkeli RedHat-yrityksen verkkosivulla. Viitattu 14.2.2021. https://www.redhat.com/en/topics/containers/what-is-docker.

What is IaaS? N.d.a. Overview of infrastructure as a service. Artikkeli Microsoft Azure verkkosivuilla. Viitattu 14.2.2021.
https://azure.microsoft.com/en-us/overview/what-is-iaas/.

What Is IaaS? N.d.b. Artikkeli Alibaba Cloud verkkosivulla. Viitattu 14.2.2021. https://www.alibabacloud.com/knowledge/what-is-iaas.

What is PaaS? N.d. Overview of platform as a service. Artikkeli Microsoft Azure verkkosivuilla. Viitattu 14.2.2021.
https://azure.microsoft.com/en-us/overview/what-is-paas/.

What is SaaS? N.d. Overview of software as a service. Artikkeli Microsoft Azure verkkosivuilla. Viitattu 14.2.2021.
https://azure.microsoft.com/en-us/overview/what-is-saas/.

What is version control? N.d. Atlassian Bitbucket -versionhallinnan opas. Viitattu 11.2.2021. https://www.atlassian.com/git/tutorials/what-is-version-control.