jamk.fi

# Jatkuva toimittaminen Azure DevOps -ympäristössä

Jesse Heikkinen

Opinnäytetyö
Toukokuu 2021
Tietojenkäsittely ja tietoliikenne
Insinööri (AMK), tieto- ja viestintätekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

**Kuvailulehti**

| Tekijä(t) | Julkaisun laji | Päivämäärä |
|---|---|---|
| Heikkinen, Jesse | Opinnäytetyö, AMK | Toukokuu 2021 |
| | Sivumäärä<br>31 | Julkaisun kieli<br>Suomi |
| | | Verkkojulkaisulupa myönnetty: x |

Työn nimi
**Jatkuva toimittaminen Azure DevOps -ympäristössä**

Tutkinto-ohjelma
Insinööri (AMK), tieto- ja viestintätekniikka

Työn ohjaaja(t)
Esa Salmikangas, Jouni Huotari

Toimeksiantaja(t)
Kupari Solutions Oy


Tiivistelmä

Työn tavoitteena oli suunnitella ja toteuttaa toimeksiantajalle automatisoitu jatkuvan toimittamisen järjestelmä. Järjestelmä tuli toteuttaa toimeksiantajan Azure DevOps -ympäristössä. Toteutettavan järjestelmän vaatimuksina oli koostaa ja julkaista tuotteiden uudet versiot Azuren App Service -palveluun sekä ajaa mahdolliset muutokset Azuren pilvipalvelussa olevaan tietokantaan.

Työn alkuvaiheessa tutustuttiin DevOps-käsitteeseen sekä Azure DevOps -ympäristöön tarkemmin, jonka jälkeen tutkittiin miten automaattinen jatkuvan toimittamisen järjestelmä, kannattaisi Azure DevOps -ympäristössä toteutettaa. Azure DevOps -ympäristöön paremmin tutustuttua oli selvää, että kannattaisi työn toteutuksessa käyttää ympäristön Pipelines-palvelua. Palvelussa voidaan luoda jatkuvan integraation ja jatkuvan toimittamisen putkia, joista kokonaisuutena saataisiin muodostettua automatisoitu jatkuvan toimittamisen järjestelmä.

Ensimmäisenä työn toteutusvaiheessa luotiin Azure DevOps -ympäristöön jatkuvan integroinnin putki, joka koosti tuotteen, muodosti tietokannan muutoksista ajettavan SQL-skriptin ja ajoi testit. Tämän jälkeen luotiin Azure DevOps -ympäristöön jatkuvan toimittamisen putki, joka haki uusimman version koostetusta tuotteesta ja julkaisi sen Azuren App Service -palveluun, sekä ajoi tietokanta muutokset Azuren pilvipalvelussa olevaan tietokantaan.

Työn tuloksena toimeksiantajalle toteutettin Azure DevOps -ympäristössä toimiva automatisoitu jatkuvan toimittamisen järjestelmä. Toteutettu järjestelmä automatisoidusti koostaa ja julkaisee uuden version tuotteista Azuren App Service -palveluun sekä ajaa mahdolliset muutokset Azuren pilvipalvelussa olevaan tietokantaan.



Avainsanat (asiasanat)
Azure DevOps, DevOps, jatkuva integraatio, jatkuva toimittaminen


Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

**Description**

| Author(s) | Type of publication | Date |
|---|---|---|
| Heikkinen, Jesse | Bachelor's thesis | May 2021 |
| | | Language of publication: Finnish |
| | Number of pages: 31 | Permission for web publication: x |

Title of publication

**Continuous delivery in Azure DevOps**

Degree programme

Information and Communication Technology

Supervisor(s)

Esa Salmikangas, Jouni Huotari

Assigned by

Kupari Solutions Oy


Abstract

The objective of this work was to plan and implement an automated continuous delivery system to the assignee. The system was to be done in Azure DevOps platform. The requirements for the system were that it had to build and deploy any new versions of the products to Azure App Service and apply any changes to the database in Azure cloud services.

At the start of this work, we get a better understanding of the term DevOps and Azure DevOps platform in general. After which we studied more about how an automated continuous delivery system should be implanted in Azure DevOps platform. After studying more about Azure DevOps platform, it became clear that we should use the Pipelines service. With the Pipelines service, you can create a continuous integration and continuous delivery pipelines that together would create an automated continuous delivery system.

The first thing when starting to implement the system was to create a continuous integration pipeline that built the product, formed the script from database changes, and ran the tests. After this came the creation of the continuous delivery pipeline that fetched the newest version of the built product and deployed it to Azure App Service and applied the database changes to the database in Azure cloud services.

As the result of this work, an automated continuous delivery system was implemented for the assignee. The implemented system automatically builds and deploys any new versions of the products to Azure App Service and applies any database changes to the database in Azure could services.



Keywords/tags (subjects)

Azure DevOps, DevOps, continuous integration, continuous delivery


Miscellaneous (Confidential information)

# Sisältö



**Kuviot**

# Sanasto

**.NET Core**

Microsoftin kehittämä avoimen lähdekoodin ohjelmistokehitys alusta Windowsille, Linuxille ja macOS käyttöjärjestelmille.

**Agent**

Virtuaalinen tai fyysinen kone, jolla suoritetaan tehtäviä yksi kerrallaan.

**AVUX**

Kupari Solutions Oy:n kehittämä ja tarjoama työnohjausjärjestelmä.

**Azure**

Microsoftin tarjoama pilvialusta ohjelmistojen ja palveluiden julkaisuun sekä hallintaan pilvessä.

**Azure App Service**

Palvelu, jolla voidaan julkaista web-sovelluksia.

**Azure DevOps**

Ympäristö, joka tarjoaa tarvittavat työkalut DevOps-käytänteiden käyttöönottamiseksi.

**Azure SQL Database**

Palvelu, jolla voidaan julkaista tietokantoja pilveen.

**Backend**

Ohjelmiston osa, jonka kanssa käyttäjä ei ole suoraan vuorovaikutuksessa. Vastaa tyypillisesti datan tallennuksesta ja manipuloinnista.

**DevOps**

Käsite, joka koostuu käytänteistä, jotka yhdistävät kehitys ja ylläpito tehtävät toisiinsa.

**Dotnet-komentokehote**

Komentorivi käyttöliittymä, joka tarjoaa komentoja .NET-sovelluksien kehittämiseen ja koostamiseen.

**Entity Framework Core**

Avoimen lähdekoodin tietokantarajapinta, joka mahdollistaa .NET luokkien käyttämisen tietokantojen kanssa.

**Frontend**

Ohjelmiston osa, jonka kanssa käyttäjä on suoraan vuorovaikutuksessa. Tyypillisesti tämä on ohjelmiston käyttöliittymä.

**Jatkuva integraatio**

Kuvaa lähdekoodin automaattista koostamista ja testausta, lähdekoodiin tulleiden muutosten yhteydessä. Englanniksi continuous integration.

**Jatkuva toimitus**

Kuvaa sovelluksen automaattista julkaisemista aina uuden version ollessa saatavilla. Englanniksi continuous delivery.

**Migraatio**

Tiedosto, joka muodostetaan .NET luokkiin tehdyistä muutoksista ja jonka avulla muutokset voidaan ajaa suoraan tietokantaan.

**Putki**

Kooste eri tehtäviä, jotka suoritetaan järjestyksessä yksi kerallaan. Kuten sovelluksen koostaminen, testaus ja julkaisu.

# 1 Johdanto

Työn toimeksiantajana toimii Kupari Solutions Oy, jolla on toimipisteet Espoossa ja Jyväskylässä. Kupari Solutions Oy:n historia alkaa jo vuodesta 1976, jolloin yhtiö toimi LVIS-suunnittelun ja energiakatselmustoiminnan parissa. Tietoteknisen vallankumouksen mukana yhtiö on kuitenkin kehittynyt IT yhtiöksi. Nykyään Kupari Solutions Oy tarjoaa yhtenä palvelunaan AVUX-työnohjausjärjestelmän, joka on markkinoiden käytetyin työnohjausjärjestelmä, yli 20 000 aktiivisella päivittäisellä käyttäjällä. Järjestelmän kehityksessä yhtiö panostaa tiiviiseen yhteistyöhön kaikkien asiakkaidensa kanssa. (Kupari Solutions N.d)

Yhtiön tämänhetkisestä AVUX-työnohjausjärjestelmästä on kehitteillä uusi versio, jonka kehityksessä on käytetty nykyaikaisia teknologioita. Järjestelmä tulee toimimaan Azuren pilviympäristössä ja järjestelmä tulee käyttämään DevOps-käsitteen jatkuvan integraation ja jatkuvan toimittamisen periaatteita hyödykseen. Samalla tavoitellen näistä saatavia etuja kuten päivitysten vientiä asiakkaille nopeammin ja useammin, paremman reaktiokyvyn asiakaspalautteeseen ja tiiviimmän yhteistyön asiakkaiden kanssa järjestelmän kehityksessä.

Työn tarkoituksena on luoda uuden AVUX-järjestelmän backend- ja frontend-projekteille kehitysvaiheessa toimiva automatisoitu jatkuvan toimittamisen järjestelmä, josta voidaan sitten jatkokehittää valmiin AVUX-järjestelmän kanssa toimiva ratkaisu. Jatkuvan toimittamisen järjestelmän tulisi ajaa mahdolliset tietokantamuutokset Azuren pilvipalvelussa sijaitsevaan tietokantaan, jota käytetään kehityksen aikana. Koska uusi AVUX-järjestelmä tulee toimimaan Azuren pilvipalvelussa, ja Azuren tarjoamat pilvipalvelut ovat vahvasti mukana uuden järjestelmän kehityksessä ja ylläpidossa, tulee toteutettava jatkuvan toimittamisen järjestelmä toteuttaa Azure DevOps -alustalla.

Opinnäytetyö koostuu neljästä osasta. Ensimmäisessä osassa avataan DevOps-käsitettä ja sen keskeisimpiä periaatteita, jatkuvaa integraatioita ja jatkuvaa toimittamista. Toisessa osassa tutustutaan Azuren tarjoamiin DevOps-palveluihin tarkemmin.

Kolmannessa osassa käydään läpi itse työn toteutuksen eri vaiheet ja mitä ne pitävät sisällään. Viimeisessä osassa pohditaan työn onnistumista, mitä kohtia voidaan vielä jatkokehittää ja pohditaan työn aikana opittuja asioita.

Kyseessä on toiminnallinen opinnäytetyö, jonka tuotoksena on käytännön toteutus, Azure DevOps -ympäristössä toteutetusta automatisoidusta jatkuvan toimittamisen järjestelmästä. Työn tarkoituksena oli pyrkiä säästämään kehittäjien resursseja ja tihentämään AVUX-järjestelmän päivitysväliä, hyödyntämällä DevOps-käytänteita kuten jatkuvaa integrointia ja jatkuvaa toimittamista.

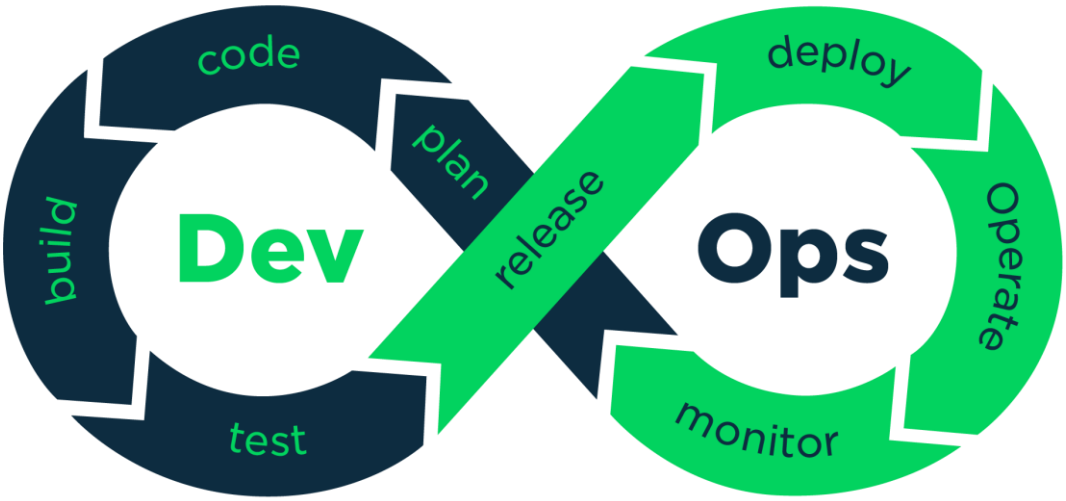
# 2 DevOps

## 2.1 DevOps:n historia

Vuonna 2009 yleiseen tietoon noussut termi DevOps kuvasi silloin uudenlaista käytännettä ohjelmisto projektien kehitykseen (Mezak 2018). Perinteinen käytänne ohjelmistokehityksessä on ollut, että kehittäjät ja ylläpitäjät toimivat erillään toisistaan. Koska kehitystiimi toimii omana yksikkönään erillään ylläpidollisista yksiköistä, niin mm. uusien versioiden vienti tuotantoon on vaatinut jonkin verran byrokratiaa ja muita ylimääräisiä askeleita. DevOps-käsite onkin tuoda näiden kahden nimikkeen alla toimivat yksiköt yhteen ja panostaa näiden yksiköiden tiiviiseen yhteistyöhön. DevOps-käsite on kasvattanut suosiotaan kovasti ja onkin jo tullut melkein yhdeksi alan vakiokäytänteeksi. (Abildskow 2020).

Alun perin DevOps-käsite on syntynyt vastauksena ketterän kehityksen puutteille. Kun yritykset alkoivat siirtyivät pois vesiputousmallista, alkoi syntyä uudenlainen ohjelmistokehitys malli, joka tunnetaan tänä päivänä ketterän kehityksen mallina. Tässä uudenlaisessa mallissa oli kuitenkin omat puutteensa, kuten kehittäjien harteille syntynyt kasvava työtaakka sekä järjestelmän ylläpitäjien lisääntynyt kyvyttömyys tukea kehittäjiä. Myös muut asiat kuten dokumentoinnin vähäisyys, kasvattivat ketterien

projektien riskiä päättyä huonosti. Vastauksena näihin ketterän kehityksen puutoksiin syntyi konsepti nimeltä DevOps, Andrew Clayn ja Patrick Debois:n keskustelun tuloksena, vuonna 2008 (Kuvio 1). Ensimmäisen DevOpsDays-tapahtuman jälkeen vuonna 2009 alkoi termi levitä myös yleiseen tietoisuuteen ohjelmistokehityksen maailmassa. (Agarwal 2019).




Kuvio 1. DevOps työnkulku (DevOps 2020.)

## 2.2 DevOps tänä päivänä

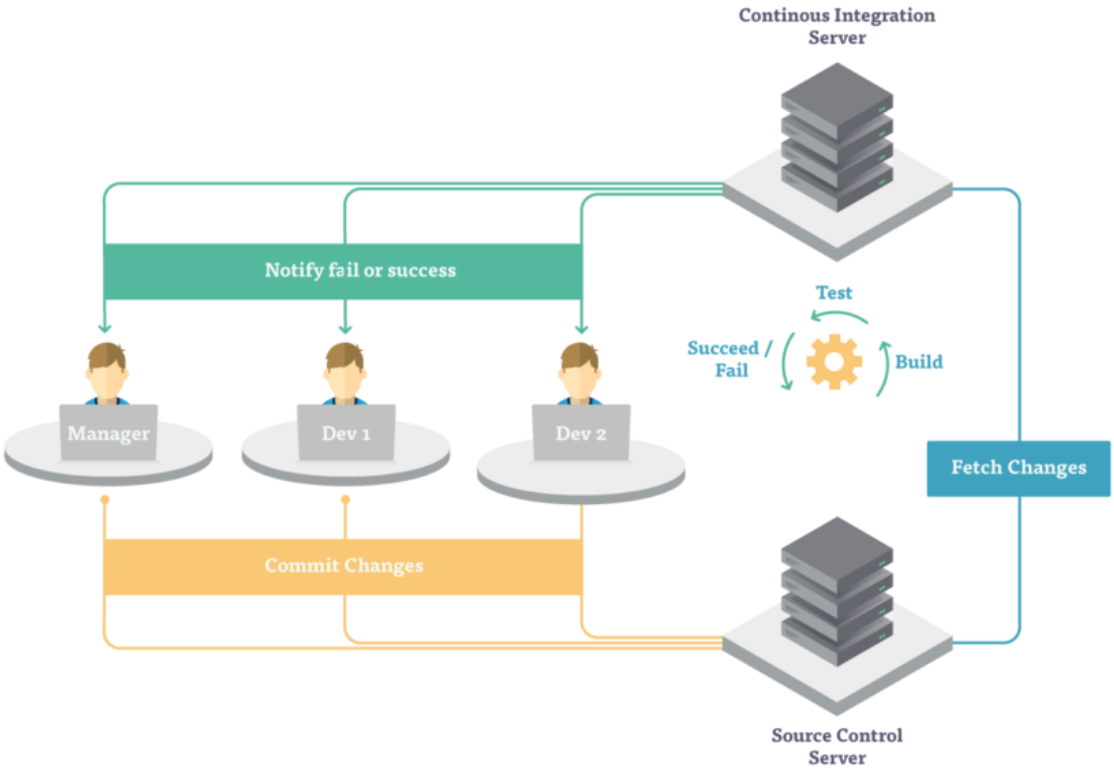
Tänä päivänä DevOps-käsite pitää kuitenkin sisällään paljon enemmän kuin vain kehittämisen ja ylläpidolliset tehtävät. Nykyään termi DevOps viittaa kaikkeen ohjelmistosuunnittelusta tuotehallintaan, palvelun luotettavuuteen ja terveeseen työkulttuuriin. DevOps:in keskeisimmäksi asiaksi onkin muodostunut kulttuurin luominen jatkuvalle oppimiselle kokeilun kautta, jotta asiakas saa parhaan mahdolliseen tuotteen nopeasti ja luotettavasti. Tähän päästään automatisoidun testauksen ja toimittamisen ansiosta, jotka luovat kehittäjille eräänlaisen turvaverkon. Kyseisen turvaverkon ansiosta voidaan kokeilla uusia asioita entistä vapaammin ilman pelkoa, että rikotaan tuotantoympäristö. Juuri tästä syystä jatkuva integraatio ja jatkuva toimittaminen ovat DevOps-käsitteen keskeisimpiä termejä. (Abildskow 2020).

CITE Research suoritti helmikuussa 2020 kyselytutkimuksen, Atlassianin puolesta, DevOps-käytänteistä yrityksissä. Melkein kaikki tutkimukseen osallistuneet kokivat, että DevOps-käytänteillä on positiivinen vaikutus heidän organisaatioissansa. Osallistujat, jotka olivat päätöksenteko asemassa tai mittasivat DevOps:in menestystä yrityksensä sisällä vastasivat todennäköisimmin, että DevOps-käytänteillä oli todella suuri positiivinen vaikutus heidän organisaatioissansa. Tutkimuksessa havaittiin myös, että osallistujat kokivat DevOps:in suurimmaksi vaikutuksesi heidän organisaatioissansa tuotteiden laadun paranemisen. Suurimmiksi esteiksi DevOps-käytänteiden käyttöönottoon organisaatioissa koettiin työntekijöiden osaamisen puute, organisaation vanha infrastruktuuri ja kulttuurin muuttaminen organisaation sisällä. (2020 DevOps Trends Survey 2020).

## 2.3 Jatkuva integraatio

Jatkuva integraatio, englanniksi continuous integration, on osa DevOps -käsitettä ja sillä kuvataan automaattista lähdekoodin koostamista ja testaamista. Jatkuvan integraation ansiosta kehittäjät pystyvät julkaisemaan muutoksiaan versionhallinnan master-haaraan useammin ja paljon nopeammin kuin ennen. Ennen jatkuvaa integraatiota kehittäjät saattoivat viettää pitkiäkin aikoja omien kehitystensä parissa, julkaisematta muutoksiaan master -haaraan. Kun muutoksia vihdoin vietiin master-haaraan, saattoi prosessi olla työläs ja hidas, luoden paljon ristiriitoja muiden kehittäjien muutosten kanssa, ennalta-arvaamattomia bugeja ja jopa päällekkäistä työtä muiden kehittäjien kanssa. (Guckenheimer 2017).

Jatkuvan integraation tehtävänä on tarkkailla versionhallintaa ja havaita aina kun versionhallinnassa on uusi versio saatavilla. Kun uusi versio on havaittu, haetaan versionhallinnasta uusin versio, koostetaan se ja ajetaan mahdolliset testit, jotka on määritetty ajettavaksi (Kuvio 2). Koska prosessi on automaattinen niin se ei vie kehittäjien aikaa ja kehittävät voivat huoletta ajaa pieniäkin tekemiään muutoksia versionhallinnassa käytettävään yhteiseen haaraan. Ajamalla muutoksia yhteiseen haaraan useammin, vältetään paljon päällekkäistä työtä, ristiriitoja muutoksissa ja muutosten aiheuttamia bugeja. (Guckenheimer 2017).




Kuvio 2. Jatkuva integraatio (Luetgebrune 2018.)

Mikäli on tarve kehittää jotakin ominaisuutta eristyksissä master-haarasta, voidaan käyttää hyväksi versionhallinnan tarjoamia haaroja ja luoda kehityksellä oma sivuhaara master-haarasta. Kehitystä voidaan tehdä tässä haarassa erillään master-haarasta ja kun kehitys on viety loppuun, voidaan haara yhdistää master-haaraan. Kuvissa 2 on havainnollistettu jatkuvan integraation toimintaa ja eri vaiheita. (Guckenheimer 2017).

## 2.4 Jatkuva toimitus

Jatkuva toimittaminen, englanniksi continuous delivery, on osa DevOps-käsitettä kuten jatkuva integraatiokin. Jatkuvalla toimittamisella tarkoitetaan tuotantoversion jatkuvaa päivittämistä aina uuden version ollessa saatavilla. Tuotantoversion päivittyessä useammin syntyy asiakkaille lisäarvoa. Jatkuva toimittaminen onkin tänä päivänä ohjelmistoyrityksille jo melkein enemmänkin vaatimus kuin vaihtoehto. (Guckenheimer 2017).

Jatkuvan toimittamisen prosessi alkaa, kun jatkuvan integraation prosessi on suoritettu onnistuneesti loppuun ja uusi koostettu versio on saatavilla. Jatkuvan toimittamisen putki koostuu yleisesti testausympäristöstä ja tuotantoympäristöä jäljittelevästä ympäristöstä. Uusi versio kulkee putken läpi ympäristöstä toiseen, joissa sitä testataan ja viimein, testien mentyä onnistuneesti läpi tai kun todetaan että versio on toimiva, tuotantoympäristön versio korvataan uudella versiolla. (Guckenheimer 2017).

# 3 Azure DevOps

Azure DevOps on Microsoftin tarjoama ja ylläpitämä ympäristö, joka mahdollistaa DevOps-käytänteiden käyttöönoton ja tarjoaa siihen kaikki tarvittavat työkalut ja palvelut. Ympäristö tukee kehittäjien, projektipäälliköiden ja mahdollisten muiden tukijoiden yhteen tuomista. Näin ollen mahdollistaen nopeamman ohjelmistokehitys prosessin kuin perinteiset ohjelmistokehityksen menetelmät. Tässä luvussa käydään lävitse eri työkaluja, joita ympäristö tarjoaa. (What is Azure DevOps 2021).

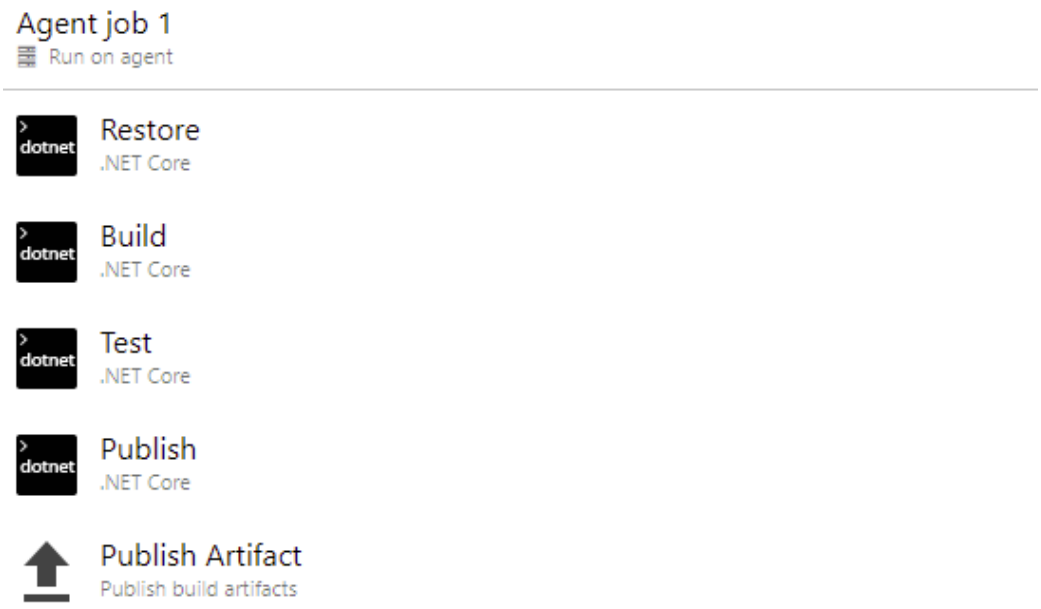
## 3.1 Azure Repos

Azure Repos on yksi Azure Devops -ympäristön tarjoamista palveluista. Palvelu koostuu erilaisista työkaluista, joilla voidaan hallita lähdekoodia. Palvelu tarjoaa tällä hetkellä kaksi eri versionhallinta järjestelmää. Toinen näistä järjestelmistä on Git, joka on niin sanottu hajautettu versionhallinta, tarkoittaen, että paikallinen kopio versionhallinnassa olevasta lähdekoodista sisältää koko muutoshistorian. Git:in lisäksi Azure Repos tarjoaa myös TFVC-versionhallinta järjestelmän, eli keskitetyn versionhallinnan. Keskitetyssä versionhallinnassa lokaali kopio lähdekoodista sisältää vain yhden version ja muutoshistoriaa säilytetään ainoastaan versionhallinnan palvelimella. (What is Azure Repos 2020).

## 3.2 Azure Pipelines

Azure Pipelines -palvelulla voidaan implementoida projektiin DevOps-käsitteen mukainen automatisoitu jatkuvan integroinnin tai jatkuvan toimittamisen putki. Palvelu työskentelee yhdessä Azure Repos -palvelun kanssa ja se voidaan konfiguroida automaattisesti reagoimaan uuden version tullessa saataville versionhallintaan, ajaen uusi versio jatkuvan integraatio -putken lävitse. Palvelu toimii myös muiden versionhallinta alustojen kanssa, kuten esimerkiksi GitHub:in kanssa. (What is Azure Pipelines 2019).

Palvelun jatkuva integraatio -putken toteutus koostuu yhdestä tai useammat tehtävästä, joista luodaan lista ja putki suorittaa tehtävät yksi kerrallaan järjestyksessä (Kuvio 3). Tehtäviä yhdistelemällä saadaan luotua valmis jatkuvan integraation putki, joka koostaa ja testaa sovelluksen automaattisesti. Palvelu tukee suurinta osaa yleisimmistä sovellustyypeistä ja palvelussa onkin valmiita tehtäviä esimerkiksi .NET Core, Java tai Node teknologioilla toteutettujen sovelluksien koostamiseen. Valmiita tehtäviä löytyy myös testien ajamiseen useilla eri testauskehyksillä tai omien komentojen ajamiseen PowerShell-, komentorivi- tai Shell-komentokehoitteilla. (What is Azure Pipelines 2019).




Kuvio 3. Esimerkki jatkuvan integraation putkesta

Jatkuva toimitus -putken toteutus Azure Pipelines -palvelussa on toteutettu jatkuva integraatio -putken tapaan. Erona vain se, että jatkuvan toimittamisen putki voi koostua monesta vaiheesta, joilla kullakin on oma tehtävälistansa, joka suoritetaan. Kunkin vaiheen tehtävälista koostuu yhdestä tai useammasta tehtävästä. Eri vaiheet voivat olla esimerkiksi tuotteen julkaisu kehitysympäristöön ja seuraavassa vaiheessa tuotantoympäristöön. Putken avulla voidaan sovellus julkaista useaan erityyppiseen kohteeseen, kuten virtuaalikoneille, kontteihin tai vaikka pilvipalveluihin. Palvelu mahdollistaa myös mobiilisovelluksen julkaisemisen suoraan sovelluskauppaan. (What is Azure Pipelines 2019).

## 3.3 Azure Boards

Azure Boards on Azure DevOps -ympäristössä toimiva toiminnanohjaus-järjestelmä projektien hallintaan. Palvelusta löytyy kattava lista ominaisuuksia ja työkaluja, jotka myös skaalautuvat tarpeen mukaan. Palvelusta löytyy mm. natiivi tuki Scrum:lle ja Kanbanille, muokattavissa olevat tehtävätaulut ja integroitu raportointi. (What is Azure Boards 2020).

## 3.4 Azure Test Plans

Azure Test Plans -palvelu tarjoaa kehittäjille selainpohjaiset työkalut testauksen hallintaan ja seurantaan. Palvelun avulla voidaan määrittää manuaalisia testejä, käyttäjätestejä, tutkivia testejä sekä kerätä palautetta sidosryhmiltä. Palvelu toimii myös yhteistyössä Azure Boards -palvelun kanssa, ja esimerkiksi manuaalisia testejä voidaan liittää suoraan Kanban-taulun tehtäviin. Kanban-taululta voidaan suoraan hallita testitapauksia sekä seurata niiden tilaa. (Azure Test Plans documentation n.d).

# 4 Toteutus

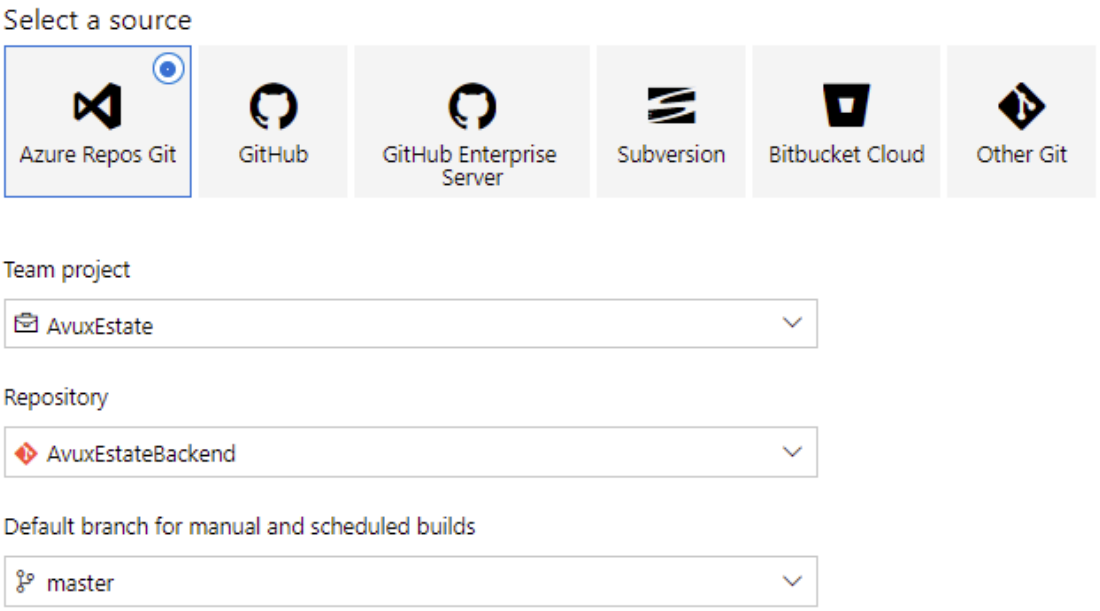
## 4.1 Lähtökohdat

Kupari Solutions Oy:n AVUX-järjestelmän nykyistä versiota päivitetään manuaalisesti viemällä tehdyt muutokset manuaalisesti palvelimille, jossa eri palvelut pyörivät. AVUX-järjestelmästä on kuitenkin uusi versio kehityksessä ja siinä halutaan välttää ongelmia ja hitautta, jotka manuaalisesti päivittäminen tuo. Toteutettavan automaattisen jatkuvan toimittamisen järjestelmän vaatimuksena on automatisoida AVUX-järjestelmän päivitysprosessi. Eli toteutettavan järjestelmän tulee reagoida versionhallinnan muutoksiin master-haarassa ja automaattisesti koostaa ja julkaista tuote. Lisäksi tietokantamuutokset tulee ajaa Azuren pilvipalvelussa olevaan tietokantaan. Järjestelmän tulee toimia kehitysvaiheen AVUX-järjestelmän kanssa ja siitä on tarkoitus myöhemmässä vaiheessa jatkokehittää tuotantoon sopiva versio.

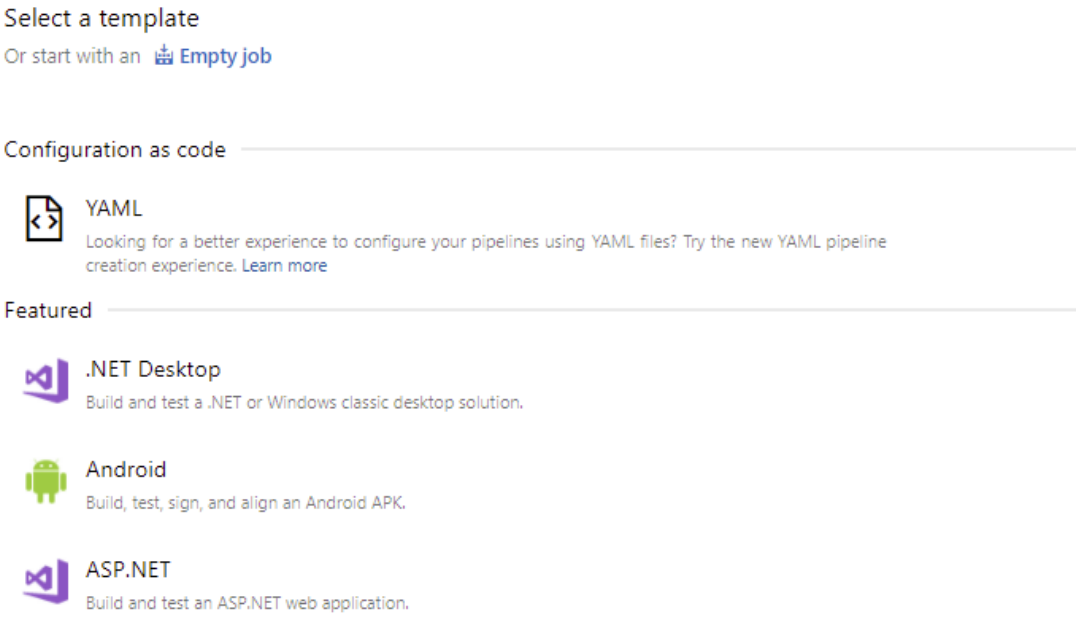
Uusi AVUX-järjestelmä koostuu kehitysvaiheessa tietokannasta, REST-rajapinnasta (backend) ja web-käyttöliittymästä (frontend). Käytännössä tämä tarkoittaa, että toteutettava jatkuvan toimittamisen järjestelmä sisältää kaksi jatkuvan integraation putkea ja kaksi jatkuvan toimittamisen putkea, backend- ja frontend-projekteille omat. Tässä käydään kuitenkin läpi vain backend-projektin putkien luominen ja käyttöönotto, koska molemmat projektit käyttävät .NET Core -kehystä, niin putket ovat melkein identtisiä. Ainoa käytännön ero putkissa on backend-projektin osalta huomioon otettavat tietokantamuutokset.

## 4.2 Jatkuva integraatio -putken käyttöönotto

Azure DevOps -ympäristössä uuden jatkuva integraatio -putken luominen onnistuu Azure Pipelines -palvelun kautta. Jatkuva integraatio -putken luomisen ensimmäinen vaihe on versionhallintajärjestelmän valinta, josta putken läpi menevä lähdekoodi löytyy (Kuvio 3). Tämän jälkeen voidaan valita valmis pohja putkelle tai luoda tyhjä putki ilman tehtäviä. Azure Pipelines -palvelu tarjoaa valmiita pohjia esimerkiksi ASP.NET, Android, Go ja Node.js sovelluksille. (Kuvio 4).

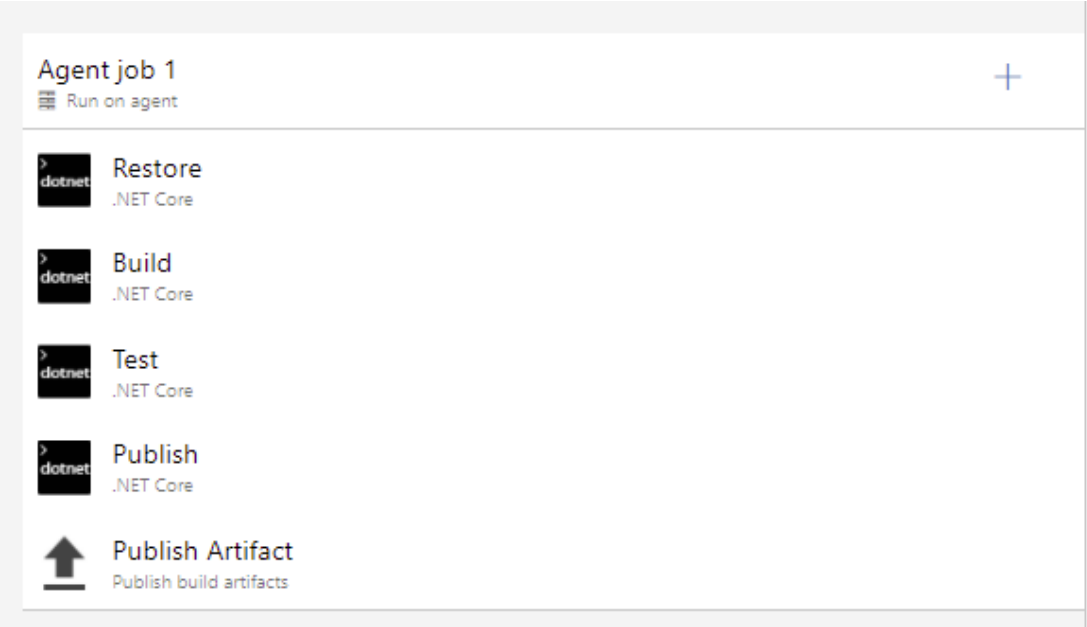



Kuvio 4. Lähteen valinta uudelle CI-putkelle




Kuvio 5. Azuren tarjoamia valmiita pohjia CI-putkeen

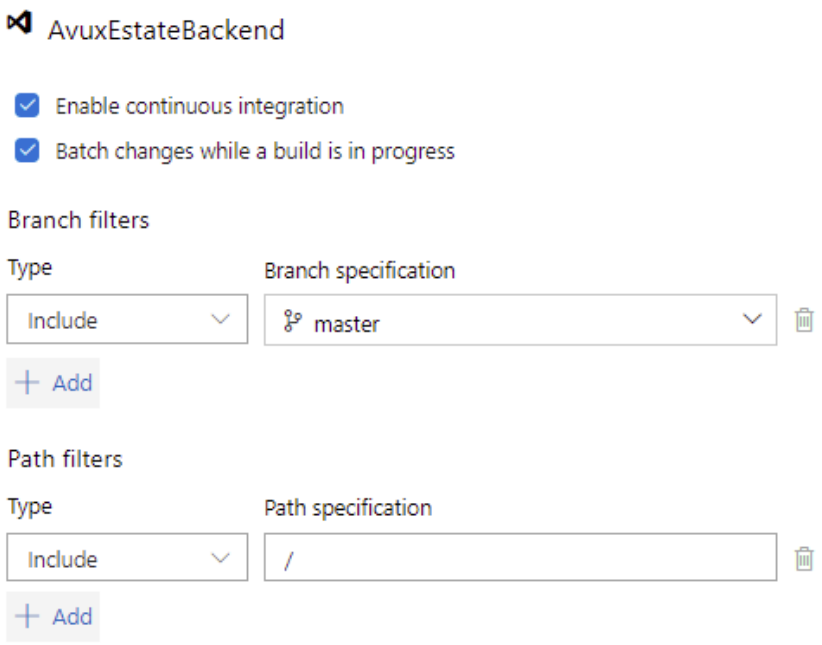
Backend-projekti on toteutettu käyttäen .NET Core kehystä, joten valitsemalla vastaava valmis pohja putkea luodessa, on putkessa valmiiksi luotuna tehtävät sovelluksen käyttämien riippuvuuksien ja työkalujen asennukseen, koostamiseen, yksikkötestien ajamiseen ja artefaktin julkaisuun. (Kuvio 5).




Kuvio 6. Azuren luoma CI-putki ASP.NET Core mallista

Halutessaan luodulle putkelle voi vaihtaa oletuksena asetetun agent-koneen. Agent on virtuaalikone, jota käytetään putkessa määritettyjen tehtävien suorittamiseen. Tarjolla on Azure Pipelines -palvelun ylläpitämiä agent-koneita, joita voi käyttää, tai sitten käyttäjä voi ylläpitää itse omaa virtuaalista agent-konetta. Itse ylläpidetyssä agent-koneessa on se etu, että käyttäjä voi itse hallita koneella olevia ohjelmia ja säädetyt asetukset säilyvät agent-koneella putken ajokertojen välillä. Tässä vaiheessa backend-projektia käyttöön tulee Azure Pipelines -palvelun ylläpitämä agent-kone. Agent-koneesta luodaan aina putken ajon alkaessa uusi instanssi. Agent-konetta voi vaihtaa itse ylläpitämään tai muuhun Azure Pipelines -palvelun ylläpitämään koneeseen vielä myöhemmin, mikäli tälle tulee tarvetta.

Triggers-välilehdellä voidaan määrittää sääntöjä sille, milloin kyseinen putki ajetaan. Ajo voidaan suorittaa esimerkiksi aina kun versionhallinnan master-haara päivittyy, tietyin aikavälein tai aina jonkin toisen jatkuvan integraatio putken ajon jälkeen. Tässä tapauksessa halutaan ajaa CI-putki aina kun master-haara päivittyy, joten putkelle täytyy valita kohta Enable continuous integration ja suodattaa pois muut kuin master-haaran muutokset (Kuvio 6).




Kuvio 7. Triggers välilehden asetukset

## 4.3 Valmis pohja .NET Core sovellukselle

Koska putkelle valittiin valmis pohja, niin on siinä jo valmiiksi määritettynä kuviossa 5 näkyvä lista tehtäviä. Käydään tässä kappaleessa lyhyesti läpi jokainen tehtävä ja mikäli tehtävä tarvitsee lisämäärityksiä.

Restore-tehtävä asentaa sovelluksen käyttämät riippuvuudet ja työkalut, käyttäen dotnet-komentokehotteen restore-komentoa. Tehtävään ei tässä tapauksessa tarvinnut tehdä muutoksia vaan oletuksena määritellyt asetukset olivat riittävät.

Build-tehtävä hoitaa sovelluksen koostamisen, käyttäen dotnet-komentokehotteen build-komentoa. Build-komennolle voidaan määritellä tehtävässä argumentteja. Configuration-argumentilla voidaan määritellä, koostetaanko sovelluksesta julkaisu versio vai esimerkiksi kehitysversio. Azure Pipelines -palvelussa on tähän jo valmiiksi määritelty muuttuja nimeltä BuildConfiguration, variables-välilehdellä. Muuttujan arvona on oletuksena Release, eli sovelluksesta koostetaan julkaisu versio kuten on tässä tapauksessa tarkoituskin.

Backend-projektiin ei vielä tässä vaiheessa kehitystä ole määritelty ajettavia testejä, joten valmiin pohjan mukana tullut Test-tehtävä on jätetty oletus arvoilleen, vielä toistaiseksi.

Publish-tehtävä hoitaa sovelluksen julkaisun, käyttäen dotnet-komentokehotteen publish-komentoa. Komento julkaisee sovelluksen ja kaikki riippuvuudet koostettuna valittuun kansioon. Myös tälle komennolle voidaan määritellä argumentteja ja ennalta komennolle on määritelty configuration-argumentti, kuten Build tehtävässä ja output-argumentti. Output-argumentti määrittää julkaistavan kansion kohde polun. Tälle valmis konfiguraatio on määrittänyt arvoksi Azuren ennalta määrittelemän muuttujan, nimeltä build.artifactstagingdirectory. Muuttujan arvona on polku Agent-koneella sijaitsevaan kansioon, johon artefaktit kopioidaan ja josta ne voidaan julkaista käyttäjän haluamaan paikkaan.

Publish Artifact -tehtävä hoitaa äskeisessä tehtävässä tehdyn artefaktin julkaisun Azure Pipelines -palveluun, jossa se on esimerkiksi jatkuvan toimittamisen putken saatavilla. Tehtävässä voidaan myös määrittää artefakti julkaistavaksi File Share -palveluun. Tässä tapauksessa artefakti kuitenkin julkaistaan Azure Pipelines -palveluun, josta se on helposti saatavilla jatkuvan toimittamisen putkessa. Polku jossa julkaistava artefakti sijaitsee saadaan käyttämällä samaa build.artifactstagingdirectory muuttujaa, joka edellisessä tehtävässä määritettiin julkaistavan artefaktin polun arvoksi.

## 4.4 Valmiin pohjan jatkokehitys

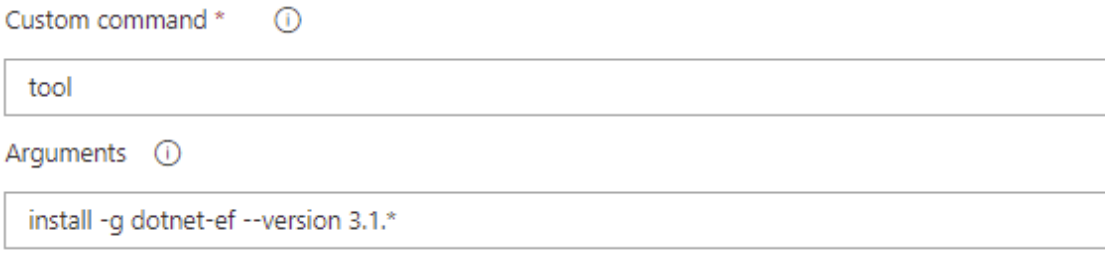
Tällä hetkellä Jatkuvan integraation putki koostaa sovelluksen ja julkaisee sen jatkuvan toimituksen putken saataville. Putkeen täytyy kuitenkin lisätä joitakin ominaisuuksia, jotta se saadaan toimimaan backend-projektin kanssa, ja täytettyä toimeksiantajan asettamat vaatimukset jatkuvan toimittamisen järjestelmälle.

Backend-projekti käyttää .NET Core versiota 3.1, joka oletuksen puuttuu luodun putken käyttämältä agent-koneelta. Putken alkuun täytyy siis lisätä Azuren tarjoama val-

mis Use .NET Core -tehtävä. Tehtävä asentaa agent-koneelle käyttäjän määrittelemän .NET Core -version ja asettaa agent-koneen käyttämään tätä versiota kaikissa lopuissa tehtävissä. Tehtävään täytyy itse vain määritellä mikä versio halutaan asentaa ja halutessaan voidaan vaihtaa asennuspolkua tai asentaa vain pelkkä runtime versio.

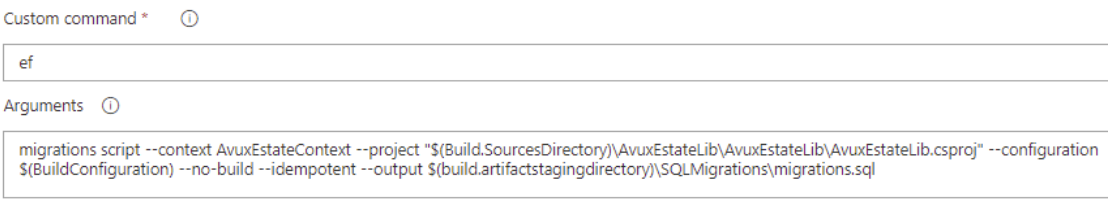
Backend-projekti käyttää myös Entity Framework Core -tietokantarajapintaa ja Code First -tietokantatoteutusmallia. Tarkoittaen, että tietokannan taulut, keskinäiset suhteet ja muutokset näihin luodaan modelien, eli C# olioiden pohjalta, käyttäen Entity Framework Core:n migraatioita. AVUX-järjestelmän kehitysvaiheessa käyttämä tietokanta on Azuren tarjoamassa SQL Database -palvelussa ja migraatioiden ajo tietokantaan tulisi automatisoida jatkuvan toimittamisen putken yhteydessä. Jatkuvan toimituksen putkessa on mahdollista luoda tehtävä, joka ajaa SQL-skriptin SQL Database -palvelun tietokantaan. Migraatioista on kuitenkin ensiksi muodostettava ajettava SQL-skripti. Tämän luominen onnistuu parhaiten osana jatkuvan integraation putkea.

Migraatioista saadaan muodostettua SQL-skripti ajamalla ef-komento ja antamalla tälle migrations script -argumentit dotnet-komentokehotteessa. Entity Framework työkalu pitää olla kuitenkin asennettuna dotnet-komentokehotteella, jotta komentoa voidaan käyttää. Koska agent-koneella ei oletuksena ole tätä asennettuna täytyy putkeen lisätä uusi .NET Core -tehtävä, jossa asennetaan kyseinen työkalu dotnet-komentokehotteeseen. .NET Core -tehtävässä voidaan ajaa dotnet-komentokehotteessa komentoja agent-koneella. Valittavissa on Azuren tarjoamia valmiita komentoja kuten restore, build ja publish. Tehtävässä on kuitenkin mahdollista myös ajaa itsemääriteltäviä komentoja valitsemalla Command-valikosta kohta custom. Entity Framework työkalun asennukseen tarvitaan komentoa tools ja argumenteiksi komennolle annetaan argumentit, joilla saadaan asennettua EF työkalun uusin päivitys versiosta 3.1, käytettäväksi järjestelmän laajuisesti (Kuvio 7).




Kuvio 8. Entity Framework toolin asennus

Seuraavaksi CI-putkeen tarvitaan vielä .NET Core -tehtävä, joka luo migraatioista SQL-skriptin ja julkaisee sen saataville samaan kansioon kuin julkaistu artefakti. Kuten aikaisempi tehtävä, myös tämä vaatii custom-valinnan Command-kohdasta ja komennoksi laitetaan äsken asennettu ef. Argumenteiksi tarvitaan aikaisemmin mainittu migrations script ja lisäksi argumenteissa täytyy määritellä käytettävä contexti, koska Backend-projekti sisältää useamman contextin. Muodostettavan skriptin on myös oltava idempotenssi, koska skripti muodostetaan jokaisella putken ajokerralla, kaikista saatavilla olevista migraatioista. Näin ollen muodostettava skripti sisältää tarkistuksen jokaiselle migraatiolle, tarkistaen onko se jo ajettu kohde tietokantaan. Näin skripti voidaan ajaa riippumatta kohde tietokantaan ajetusta viimeisimmästä migraatiosta. EF Core pitää kirjaa ajetuista migraatioista __EFMigrationsHistory-taulussa. Tästä taulusta pystytään tarkistamaan mitkä migraatiot tietokantaan on jo ajettu. Näiden lisäksi määritellään myös polku missä projektin .csproj-tiedosto sijaitsee, määritellään kooston kohde, määritellään että sovellusta ei tarvitse koostaa ja määritellään polku, johon muodostettu skripti julkaistaan. (Kuvio 8).

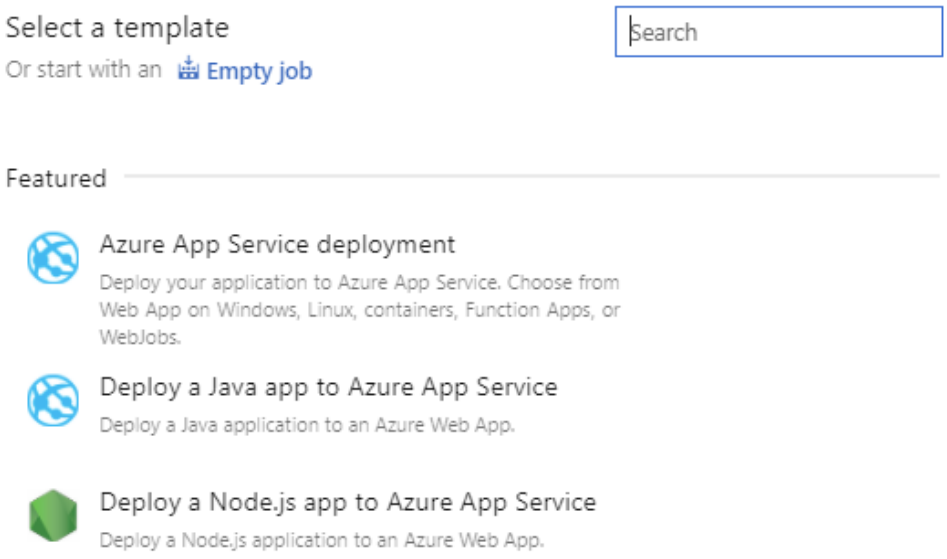



Kuvio 9. SQL-skriptin luonti migraatioista

Tässä vaiheessa jatkuvan integraation putki täyttää vaatimukset ja on valmis käytettäväksi. Kun määritetyn versionhallinnan master-haara päivittyy, niin laukaisee se putken ajon. Putki asentaa ensimmäisenä agent-koneelle oikean version .NET Core:sta, jonka jälkeen se hakee tarvittavat riippuvuudet ja koostaa sovelluksen. Seuraavaksi haetaan Entity Framwork -työkalu dotnet-komentokehotteelle ja luodaan migraatioista ajettava SQL-skripti. Kun nämä tehtävät ovat suoritettu, julkaistaan sovellus ja muodostetaan artefakti sekä siirretään se jatkuvan toimittamisen putken saataville.

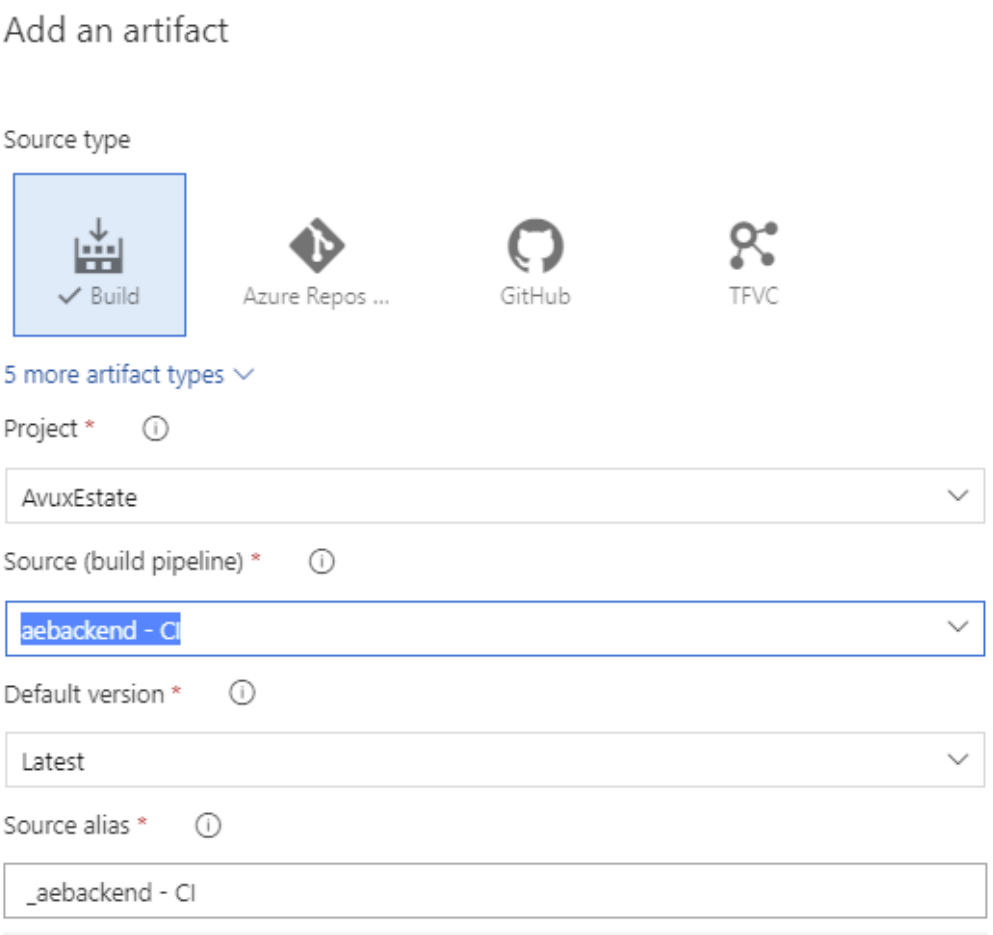
## 4.5 Jatkuva toimitus -putken käyttöönotto

Kun sovellus on mennyt jatkuvan integraatio putken läpi ja siitä on julkaistu artefakti Azure Pipelines -palveluun, niin lisätään seuraavaksi jatkuvan toimittamisen putki, joka hakee kyseisen artefaktin ja julkaisee sen Azure App Service -palveluun. Näin ollen automatisoiden sovelluksen uuden version julkaisemisen. Putkea lisättäessä voidaan valita joko kokonaan tyhjä pohja tai sitten jokin Azure Pipelines -palvelun tarjoamista valmiista pohjista, kuten jatkuvan integraation putkea luodessa (Kuvio 9). Koska backend-projekti on tarkoitus julkaista Azuren App Service:en, niin voidaan jatkuvan toimittamisen putkea luodessa käyttää Azuren tarjoamaa valmista Azure App Service deployment -pohjaa.



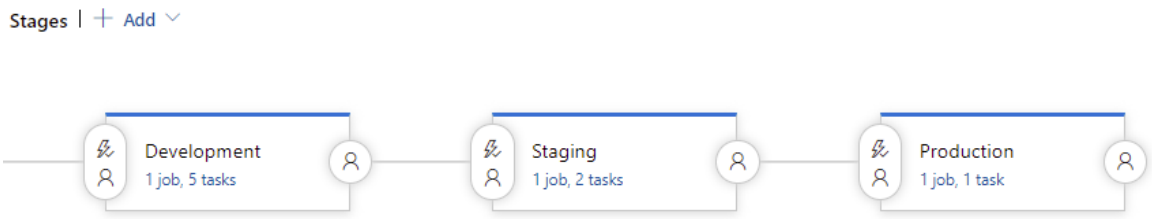

Kuvio 10. Julkaisuputken valmiita pohjia

Ensimmäisenä luodulle putkelle on määritettävä artefakti, joka julkaistaan. Tarvittaessa putkeen voi myös lisätä useamman artefaktin, mutta tässä tapauksessa sille ei ole tarvetta. Lisätään uusi artefakti ja valitaan tälle projekti sekä jatkuvan integraation putki, josta artefakti julkaistaan. Tämän jälkeen valitaan, mikä versio artefaktista halutaan ja artefaktin nimi (Kuvio 10). Kun artefakti on lisätty putkeen, sille voidaan asettaa sääntöjä, kuten milloin artefakti julkaistaan putken eri vaiheiden saataville. Tässä tapauksessa halutaan, että aina uuden artefakti version ollessa saatavilla se viedään putken vaiheiden saataville, joten laitetaan Continuous deployment trigger -valinta päälle.




Kuvio 11. Artefaktin lisäys julkaisuputkeen

Oletuksena julkaisuputki sisältää vain yhden vaiheen, julkaisuputkeen voi kuitenkin halutessaan lisätä useamman eri vaiheen. Backend-projektin julkaisuputken on tarkoitus sisältää kolme eri vaihetta, development, staging ja production. Vaiheiden välille on myös mahdollista lisätä ennen ja jälkeen ehtoja, joiden on täytyttävä, jotta artefakti siirtyy vaiheesta toiseen. Tässä vaiheessa kehitystä vaiheesta toiseen siirtyminen saa kuitenkin tapahtua vielä ilman ehtoja. (Kuvio 11).

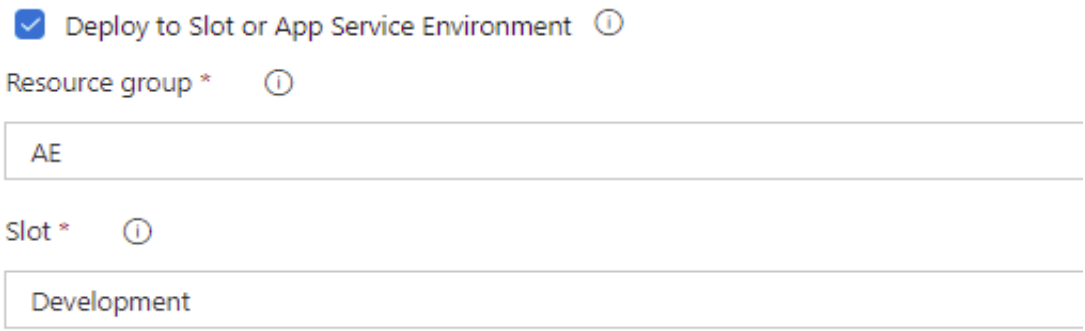



Kuvio 12. Julkaisuputken vaiheet

Jokaiseen vaiheeseen putkessa määritellään oma lista tehtäviä, jotka putki suorittaa järjestyksessä, aivan kuten jatkuvan integroinnin putkessakin. Jokaisessa vaiheessa vaatimuksena on julkaista sovelluksen uusi versio Azure App Service -palveluun ja ajaa migraatioista luotu SQL-skripti, sovelluksen käyttämään Azure SQL Database -palvelun tietokantaan. Tässä vaiheessa uuden AVUX-järjestelmän kehitystä kuitenkin jokainen Azure App Service -palvelussa oleva versio backend-projektista käyttää samaa tietokantaa. Joten tässä vaiheessa tarvitsee luotu SQL-skripti ajaa vain ensimmäisessä vaiheessa. Tarkoituksena on tulevaisuudessa luoda development, staging ja production -versioille erilliset tietokannat ja jatkokehityksenä ajaa luotu SQL-skripti myös näihin tietokantoihin.

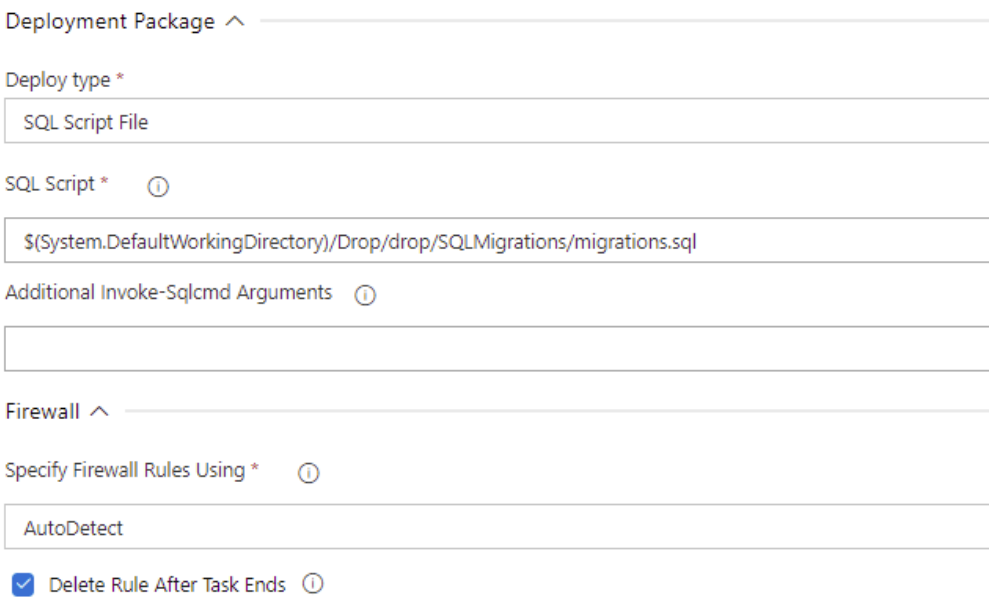
Sovelluksen julkaisu Azure App Service -palveluun onnistuu valmiissa pohjassa olevalla Azure App Service deploy -tehtävällä. Koska developement ja staging -versiot backend-projektista sijaitsevat Azure App Service Sloteissa, niin täytyy näiden vaiheiden tehtävistä valita kohta Deploy to Slot or App Service Environment ja sen alta valita käytettävä Resource group ja Slot, johon sovellus julkaistaan (Kuvio 12). Yksi

Azure App Service voi koostua monesta eri Slot:ista. Palvelussa voidaan esimerkiksi ylläpitää staging- ja production-versioita sovelluksesta. Kun halutaan viedä uusia ominaisuuksia production-versioon, voidaan ominaisuudet viedä ensiksi staging-version Slot:iin ja varmistaa niiden toimivuus siellä. Kun on varmistuttu ominaisuuksien toimivuudesta, voidaan esimerkiksi staging-version ja production-version Slot:it vaihtaa päikseen. Näin saadaan vietyä päivitykset tuotantoon ilman minkäänlaista huoltokatkoa. Slot:ien avulla voidaan myös jakaa käyttäjiä manuaalisesti eri Slot:ien välillä, ja viedä esimerkiksi päivitykset käyttäjien saataville porrastetusti. Ja voidaan myös nopeasti ohjata kaikki käyttäjät takaisin vanhaan versioon, mikäli uudessa versiossa ilmenee ongelmia käyttäjämäärän kasvaessa.




Kuvio 13. Sovelluksen julkaisu App Servicen Development Slottiin

Azure SQL Database deployment -tehtävällä voidaan ajaa SQL-skriptejä Azure SQL Database -palvelussa sijaitsevalle tietokannalle. Migraatioista luotu SQL-skripti voidaan siis ajaa tämän tehtävän avulla haluttuun tietokantaan. Tehtävään täytyy vain määrittää autentikointi tapa, millainen SQL-skripti halutaan suorittaa ja palomuuri asetukset. Tässä tapauksessa autentikoidaan käyttäjätunnuksella ja salasanalla, valitaan suoritettavaksi SQL-skriptiksi luotu SQL-skripti tiedosto, sille määritetystä polusta ja annetaan tehtävän luoda automaattisesti tarvittavat poikkeukset palomuurin asetuksiin. Valitaan lisäksi vielä kohta, joka poistaa tehdyt poikkeukset palomuurin asetuksista ajon jälkeen. (Kuvio 13).




Kuvio 14. SQL skripti tiedoston suoritus SQL Database palvelussa

Nyt jatkuvan toimittamisen putki on valmis ja kun jatkuvan integroinnin putki julkaisee uuden version valitusta artefaktista, niin jatkuvan toimittamisen putki julkaisee sen automatisoidusti vaiheittain kuhunkin määritettyyn Azure App Service -palveluun, ja ajaa migraatioista muodostetun SQL-skriptin Azure SQL Database -palvelun tietokantaan.

# 5 Pohdinta

Lähtökohtana automaattisen jatkuvan toimittamisen järjestelmän luomiseen oli toteuttaa järjestelmä, josta voidaan uuden AVUX-järjestelmän ollessa lähempänä tuotantovaihetta, jatkokehittää tuotantovaiheessa toimiva ratkaisu. Tähän tavoitteeseen työssä päästiin ja työn lopputuloksena AVUX-järjestelmällä on toimiva automaattinen jatkuvan toimittamisen järjestelmä jo kehitysvaiheessa. Järjestelmä onkin osoittautunut hyödylliseksi jo kehitysvaiheessa, sillä uuden AVUX-järjestelmän mobiilisovelluksen kehityksessä käytetään Azure App Service -palvelussa pyörivää uuden AVUX-järjestelmän REST-rajapintaa. Kehittäjien viedessä muutoksia versionhallintaan, hoitaa julkaisujärjestelmä uuden version julkaisun pilveen

automaattisesti. Tämän ansiosta säästetään kehittäjien resursseja ja voivat he keskittyä uuden AVUX-järjestelmän kehittämiseen.

Toteutettu jatkuvan toimittamisen järjestelmä toimii hyvänä pohjana, josta voidaan tulevaisuudessa jatkokehittää järjestelmä, joka toimii myös tuotantovaiheessa. Jatkokehityksenä toteutettuun järjestelmään pitäisi implementoida jatkuvan integraation putkeen mahdollisten testien ajaminen automatisoidusti. Toteutetussa järjestelmässä käytetään Azuren ylläpitämää agent-konetta putkien ajamiseen, mutta ennen tuotantovaihetta pitäisi kuitenkin tutkia lisää kannattaako tämä vaihtaa itse ylläpidettyyn agent-koneeseen. Myös SQL-skriptien ajaminen useampaan eri tietokantaan jatkuvan toimittamisen putkessa täytyy toteuttaa osana jatkokehitystä.

Työn toteutuksen jälkeen huomattiin myös, että tietokanta muutosten ajaminen täysin automatisoidusti päättyi monesti virhetilanteeseen. Yleissimmin virheet johtuivat siitä, että yritettiin poistaa tietokannan tauluista sarakkeita, joihin viitattiin muissa tauluissa. Myös tilanteet, joissa muodostetussa SQL-skriptissä oli where-ehdossa viittaus jo poistettuun sarakkeeseen aiheutti virhetilanteen SQL-skriptin ajamisessa. Koska järjestelmä on tältä osin jokseenkin epäluotettava on myös tutkittava voidaanko tietokantamuutoksien ajamisesta toteuttaa varmempi ratkaisu. Toisena vaihtoehtona on muuttaa järjestelmä tulevaisuudessa jatkuvan julkaisun järjestelmäksi ja ajaa tietokantamuutokset manuaalisesti asiakstietokantoihin. Tällöin myös uuden version julkaisu tuotanto ympäristöön käynnistettäisiin manuaalisesti.

Jos työn tuloksena toteutettua järjestelmää vertaa siihen miten perinteisesti muutokset on saatettu viedä testi tai tuotanto ympäristön palvelimille, niin tulee DevOps:n hyödyt ohjelmistokehityksessä nopeasti esille. Perinteisiä menetelmiä käyttämällä saattoi olla, että kun veit jonkin kehitetyn ominaisuuden versionhallintaan täytyi jonkun manuaalisesti käydä hakemassa versionhallinnasta uusin versio, koostaa siitä julkaisuversio ja käydä viemässä se manuaalisesti esimerkiksi testi ympäristön palvelimelle. DevOps-käytänteiden mukaisella automaattisella jatkuvan toimittamisen järjestelmellä onnistuu sama asia vain viemällä muutokset versionhallintaan. Järjestelmä ajaa automaattisesti mahdolliset testit, hoitaa julkaisuversion koostamisen ja julkaisee siitä uuden version

määritettyyn ympäristöön. Kehittäjä tai testaaja pääsee suoraan testaamaan uusia ominaisuuksia esimerkiksi testiympäristössä, ilman muiden henkilöiden manuaalista työpanosta. Näin voidaan myös ylläpitää täysin automatisoidusti esimerkiksi omaa ympäristöä, jossa mahdolliset sidosryhmät pääsevät kokeilemaan uusia vasta kehityksessä olevia ominaisuuksia. Sidosryhmiltä saatu palaute voidaan ottaa huomioon jo ominaisuuksien kehitysvaiheessa ja toimittaa asiakkaille entistä laadukkaampia tuotteita. Ilman että täytyisi ympäristöä, jossa sidosryhmät ominaisuuksia kokeilevat, ylläpitää manuaalisesti.

Ennen työn aloittamista en ollut kuin ohimennen kuullut termin DevOps, enkä osannut kertoa mitään jatkuvan integraation tai jatkuvan toimittamisen käytänteistä. Työn aikana opin kuitenkin paljon mitä kaikkea DevOps-käsite pitää sisällään ja mitä sen keskeisimmät termit tarkoittavat. Automaatisen julkaisujärjestelmän toteuttaminen ja sen käyttäminen uuden AVUX-järjestelmän kehittämisen aikana on vakuuttanut ainakin minut DevOps-käytänteiden tehokkuudesta ja jopa välttämättömyydestä nykyajan ohjelmistokehityksessä.

# Lähteet

2020 DevOps Trends Survey. 2020. Atlassian DevOps Trends Survey 2020. Kyselytutkimus. Viitattu 1.5.2021. https://www.atlassian.com/dam/jcr:a87265a6-4a4d-4905-97e9-4129ac78563c/Atlassian%20DevOps%20Trends%20Survey%202020.pdf.

Abildskov, J. 2020. What is DevOps?. Blogi. Päivitetty 14.7.2020. Viitattu 7.10.2020. https://www.eficode.com/blog/what-is-devops.

Agarwal, H. 2019. Roadway to IT Revolution: The History of DevOps. Artikkeli. Viitattu 1.5.2021. https://www.appknox.com/blog/history-of-devops.

Azure Test Plans documentation. N.d. Azure Test Plans documentation. Viitattu 1.3.2020. https://docs.microsoft.com/en-us/azure/devops/test/?view=azure-devops.

DevOps. 2020. DevOps. Viitattu 1.4.2020. https://devopedia.org/devops.

Guckenheimer, S. 2017. What is Continuous Delivery?. Viitattu 05.01.2021. https://docs.microsoft.com/en-us/azure/devops/learn/what-is-continuous-delivery.

Guckenheimer, S. 2017. What is Continuous Integration?. Viitattu 03.01.2021. https://docs.microsoft.com/en-us/azure/devops/learn/what-is-continuous-integration.

Kupari Solutions. N.d. Kupari Solutions kotisivujen yritys sivu. Viitattu 20.4.2021. https://www.kuparisolutions.fi/yritys.

Luetgebrune, J. 2018. The Expert Guide To Continuous Integration. Blogi. Viitattu 1.4.2021. https://deploybot.com/blog/the-expert-guide-to-continuous-integration.

Mezak, S. 2018. The Origins of DevOps: What's in a Name?. Artikkeli. Päivitetty 25.1.2018. Viitattu 28.10.2020. https://devops.com/the-origins-of-devops-whats-in-a-name/.

What is Azure Boards. 2020. What is Azure Boards?. Viitattu 15.2.2021. https://docs.microsoft.com/en-us/azure/devops/boards/get-started/what-is-azure-boards?view=azure-devops&tabs=agile-process.

What is Azure DevOps. 2021. What is Azure DevOps?. Viitattu 28.1.2021. https://docs.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops.

What is Azure Pipelines. 2019. What is Azure Pipelines?. Viitattu 15.2.2021. https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops.

What is Azure Repos. 2020. What is Azure Repos?. Viitattu 10.2.2021. https://docs.microsoft.com/en-us/azure/devops/repos/get-started/what-is-repos?view=azure-devops