jamk

DevOpsin CI/CD-menetelmien käyttöönotto web-sovelluksessa

Taavi Pajari

Opinnäytetyö, AMK
Syyskuu 2022
Tietojenkäsittely ja tietoliikenne
Insinööri (AMK), tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Pajari, Taavi

DevOpsin CI/CD-menetelmien käyttöönotto web-sovelluksessa

Jyväskylä: Jyväskylän ammattikorkeakoulu. Syyskuu 2022, 57 sivua.

Tietojenkäsittely ja tietoliikenne. Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Verkkojulkaisulupa myönnetty: kyllä

Tiivistelmä

DevOpsin CI/CD-menetelmät lisäävät ketteryyttä ohjelmistokehityksen vaiheisiin. Opinnäytetyö toteutettiin tutkimuksellisena kehittämistyönä, tavoitteena ottaa jatkuvan integraation ja jatkuvan julkaisun menetelmät käyttöön toimeksiantajan jo olemassa olevaan web-sovellukseen ja asentaa agenttisovellus palvelimelle. Kehittämistyön tarkoitus oli mahdollistaa, että web-sovellus koottaisiin CI-menetelmään kuuluvalla koonti putkella automaattisesti uusien koodimuutosten lisäyksen jälkeen ja asennettaisiin ja otettaisiin käyttöön palvelimelle CD-menetelmiin kuuluvan julkaisu putken avulla automaattisesti. Opinnäytetyön toimeksiantajana toimi Profit Software Oy. DevOpsin CI/CD-menetelmien käyttöönotto toteutettiin toimeksiantajalle syksyn 2022 aikana.

Kehittämistyön tutkimusosuudessa perehdyttiin DevOps-käsitteeseen, DevOpsin historiaan, CI/CD-menetelmiin ja Azure DevOps -ympäristöön. Kehittämistyön käytännön osuudessa otettiin käyttöön Microsoftin Azure Pipelines -työkaluun kuuluvat CI -ja CD-putket sekä luotiin ja asennettiin CD-putken toiminnan mahdollistava agenttisovellus toimeksiantajan palvelimelle. Suuria haasteita käyttöönoton toteutuksessa ei ollut ja vastaan tulleet hidasteet ratkesivat nopeasti. DevOpsin CI/CD-menetelmien käyttöönotto vähentää jatkossa manuaalisia työvaiheita integroimalla koodimuutokset automaattisesti lähdekoodiin sekä suorittamalla koodin koontikäännöksen ja asennuksen palvelimelle automaattisesti, helpottaen kehittäjien työtä.

Tuloksena todettiin, että DevOpsin CI/CD-menetelmien käyttöönotto Azure DevOps -ympäristön avulla käy Microsoftin dokumentaation ja käyttöliittymän helppokäyttöisyyden ansiosta helposti. Menetelmien käyttöönottoa suunnitellessa on kuitenkin hyvä varmistaa, että kaikki sovelluksen osat toimivat ja toteutuksen tekijällä on tarvittavat oikeudet. Vaikka kyseessä oli pitempi ikäinen sovellus, DevOpsin CI/CD-menetelmät saatiin lisättyä pääosin sujuvasti. Jos CI/CD-menetelmien käyttöönotto olisi haastavampaa, voidaan silti todeta, että DevOpsin CI/CD-menetelmien hyödyt, eli automatisoinnin lisääminen kehitystyöhön, ovat merkityksellisyydessään mahdollisen käyttöönoton vaivan arvoisia.

Avainsanat (asiasanat)

Tutkimuksellinen kehittämistyö, ohjelmistokehitys, ohjelmistosuunnittelu

Muut tiedot (salassa pidettävät liitteet)

-

jamk

Description

Pajari, Taavi

Implementing DevOps CI/CD methods in a web application

Jyväskylä: JAMK University of Applied Sciences, September 2022, 57 pages.

Engineering and technology. Degree Programme in Information and Communication Technology. Bachelor´s thesis.

Permission for web publication: Yes

Language of publication: Finnish

Abstract

The CI/CD methods of DevOps add agility to the software development. The thesis was implemented as a research-based development assignment, with the aim of introducing the methods of continuous integration and continuous delivery to the client's web application and installing the agent application to the server. The purpose of the development assignment was to make it possible for the web application to be built automatically using the CI method's build pipeline after the addition of new code changes and install and deploy application to the server automatically using the CD method's release pipeline. The client of the thesis was Profit Software Oy. The DevOps' CI/CD methods was implemented for the client during the fall of 2022.

In the research part of the development assignment the DevOps concept, the history of DevOps, CI/CD methods and the Azure DevOps was introduced. In the practical part of the development assignment, the CI and CD pipelines belonging to Microsoft's Azure Pipelines were implemented, and an agent application enabling the operation of the CD pipeline was created and installed to the client's server. There were no major challenges in the implementation and the encountered delays were resolved quickly. The introduction of DevOps CI/CD methods will reduce manual work steps in the future by automatically integrating code changes into the source code and automatically compiling and installing the code to the server, making the work of developers easier.

It was concluded as a result, that the implementation of DevOps CI/CD methods using the Azure DevOps is easy due to the ease of use of Microsoft's documentation and Azure DevOps's user interface. However, when planning the implementation of the methods, it is good to make sure that all parts of the application work and that the developer of the implementation has enough rights. Even though the application in the assignment was a longer-lived application, the CI/CD methods of DevOps were mostly added effortless. If the implementation of CI/CD methods would be more challenging, it can still be stated that the benefits of DevOps' CI/CD methods are worth the effort.

Keywords/tags (subjects)

Research-based development assignment, software development, software design

Miscellaneous (Confidential information)

-

Sisältö

Kuviot

Käytetyt termit ja lyhenteet

Agile	Tarkoittaa ketterää, käytetään kuvaamaan ohjelmistokehityksen työmenetelmää
Artefakti	CI-menetelmään kuuluvan build-putken lopputuote, esim. toimiva sovellus tai pakattu lähdekoodi
ASP.NET	Microsoftin kehittämä ohjelmistokehys web-sovellusten tekoa varten
Backend	Ohjelmistokehitystyö, mikä ei näy loppukäyttäjälle; esimerkiksi palvelimen tai tietokantojen ylläpito
Eräajo	Tietojenkäsittelyä, joka tapahtuu yleensä automattisesti; käyttäjältä vaaditaan lähtötiedot, mutta muuten eräajon vaiheet suoritetaan itsenäisesti
Frontend	Ohjelmistokehitystyö, mikä näkyy loppukäyttäjälle; esimerkiksi sovelluksen käyttöliittymä
Full Stack	Backendin ja frontendin yhdistelmä, eli kehitystyö mikä voi sisältää mitä tahansa ohjelmiston kehitys- tai ylläpitotehtäviä
Integraatiotestaus	Suomennos englannin kielen sanoista ”integration testing”. Testaustapa ohjelmiston osien tai rajapintojen välisiä integraatioita varten
IIS	IIS on Microsoftin palvelinohjelmisto, nimi on akronyymi englannin kielen sanoista ”Internet Information Services”
ITIL	Akronyymi englannin kielen sanoista ”Information Technology Infrastructure Library”. ITIL on kokoelma IT-alaan liittyviä käytäntöjä
Kanban	Ohjelmistotuotannossa kanban on eräänlainen taulu, missä näkyy ohjelmistoprojektiin kuuluvat työtehtävät. Taulussa on yleensä erilaisia kategorioita muun muassa tekemättömille ja työn alla oleville tehtäville
Ketterä kehitys	Kattotermi kehitystyölle, jossa on mahdollista palata vaivattomasti kehitystyön aikaisempaan vaiheeseen

Komentokehote	Ohjelma, jolla voidaan ajaa skriptejä. Windows-käyttöjärjestelmän komentokehote on cmd.exe
Lean	Johtamisfilosofia, mikä keskittyy vähentämään kehityksen tai yrityksen tuottamatonta toimintaa
Lähdekoodi	Sovelluksen, tai muun ohjelmiston jollain ohjelmointikielellä kirjoitettu tekstimuotoinen kuvaus
NuGet	Microsoftin pakettienhallintaohjelma
PowerShell	Komentokehotteen kaltainen ohjelma, jolla voidaan ajaa skriptejä. PowerShell sisältää enemmän ominaisuuksia kuin komentokehote
Pull	Versionhallinnan komento, jolla muutokset haetaan kehittäjän omaan kehitysympäristöön
Pull Request	Tai Merge Request, tehdään kun ohjelmistokehittäjä haluaa yhdistää tekemänsä muutokset kehitettävän ohjelmiston pääkoodiin
Push	Versionhallinnan komento, jolla kehittäjän tekemät muutokset ”työnnetään” palvelimelle
Päästä päähän -testaus	Suomennos englannin kielen sanoista ”end-to-end testing”. Testaustapa, millä varmistetaan, että ohjelmisto vastaa käyttäjän odotuksia
read-only	Muuttuja, jonka arvon voi lukea mutta siihen ei voi asettaa tai muuttaa aiempaa arvoa
Repositorio	Lyhennettynä repo, englanniksi repository. Suomeksi tietovarasto, tarkoittaa yleensä pilvessä olevaa säiliötä, jossa ohjelmiston lähdekoodi ja versionhallinta on
Roundhouse	Muotoiltuna ”RoundhousE”, on avoimen lähdekoodin työkalu, jota käytetään tietokantojen käyttöönottoon ja siirtoon
SaaS	Akronyymi englannin kielen sanoista ”Software as a Service”. SaaS-palvelussa ohjelmisto on ylläpitäjän palvelimilla ja asiakkaat käyttävät sitä internetin välityksellä, yleensä lisenssin kautta

Scrum	Ketterän kehityksen projektinhallintapa, jossa Scrum-tiimi tekee ennalta sovitun ajan (sprint) aikana kehitystyötä mahdollisimman pitkälle
Sprint	Scrumiin kuuluva määritelty lyhyt ajanjakso, jonka aikana kehitystyön vaiheet viedään läpi nopealla aikataululla
Tenant	Sovelluksella voi olla useampi asiakasorganisaatio, jolloin asiakkaille voidaan jakaa omat ympäristöt, eli tenantit
Tutkiva testaus	Suomennos englannin kielen sanoista ”exploratory testing”. Testaustapa, jolla testataan ohjelmistoa käyttäjän näkökulmasta ja tutkitaan, kuinka ohjelmisto toimii
Vaatimusmäärittely	Yleensä erillinen dokumentti, missä kuvataan kehitystyön vaatimukset ja mitä valmiin ohjelmiston toiminnalta vaaditaan
Vesiputousmalli	Ohjelmistokehityksen menetelmä, jossa edetään vaihe vaiheelta vaatimusmäärittelystä valmiin ohjelmiston ylläpitoon, eikä edelliseen vaiheeseen ole suotavaa palata
Web-sovellus	Sovellus, jota käyttäjä käyttää selaimen kautta
VB.NET	Tai Visual Basic, ohjelmointikieli, lyhennetty sanasta Visual Basic.NET
Visual Studio	Microsoftin ohjelmistokehitysympäristö
Yksikkötestaus	Suomennos englannin kielen sanoista ”unit testing”. Testaustapa, mikä painottuu koodipuolelle; testataan lähdekoodin osia, esimerkiksi funktioiden toimintaa
Zip	Pakatun kansion tiedostomuoto

1 Johdanto

Jo yli 20 vuotta sitten ohjelmistokehityksessä todettiin perinteisen vesiputousmallin riittämättömyys ja ketterän ohjelmistokehityksen hyötyjä tuotiin esille. Abildskovin (2013) mukaan DevOps on haastaja juuri perinteisille kehitysprosesseille (Abildskov 2013). Opinnäytetyössä esitellään sekä DevOpsia sen käsitteiden ja historian kautta, että perehdytään CI/CD-menetelmiin teorian avulla toteuttaen menetelmien käyttöönotot toimeksiantajan web-sovellukseen. Toimeksiantajan CI/CD-menetelmien käyttökokemuksen lisäksi sen perusteella, että uusimmassa, vuonna 2019 ilmestyneessä ITIL 4:sessa puhutaan ketteryyden, DevOpsin, Agilen ja Leanin käyttöönoton hyödyistä (ITIL 4 on täällä – ja se on ketterämpi kuin koskaan n.d.) voidaan todeta käyttöönoton olevan toimeksiantajalle eduksi.

Opinnäytetyön tarkoitus oli toteuttaa toimeksiantajan web-sovellukseen DevOps-menetelmiin kuuluvien CI, eli jatkuvan integraation, ja CD, eli jatkuvan toimituksen ja julkaisun -menetelmien käyttöönotot. Toimeksiantajan tavoite oli, että kun tiimin jäsen suorittaa Pull Requestin, CI/CD-menetelmät kääntävät koodin automaattisesti ja suorittavat asennukset. CD-menetelmiin kuuluvan release-putken käyttöä varten toimeksiantaja halusi oman agentin ja sen asennuksen toimeksiantajan palvelimelle. Toimeksiantajalla oli käytössä Azure DevOps -ympäristö, joten CI/CD-menetelmien käyttöönotot edellyttivät Azure Pipelines -työkaluun kuuluvien build- ja release-putkien käyttöönottoa. CI/CD-menetelmien käyttöönotto lisää ohjelmistokehitykseen ketteryyttä ja mahdollistaa ohjelmistokehityksen elinkaaren vaiheiden automatisoinnin. Toimeksiantajan web-sovellusta kehitetään jatkuvasti, joten toimeksiantaja oli todennut CI/CD-menetelmien tarpeen sekä niiden hyödyn ja lisäarvon sovelluskehityksessä.

DevOpsin CI/CD-menetelmien käyttöönotto ja agentin luonti ja asennus palvelimelle toteutettiin vaiheittain; ensin luotiin ja otettiin käyttöön CI-menetelmän build-putki, seuraavana luotiin ja asennettiin agenttisovellus toimeksiantajan palvelimelle ja lopuksi toteutettiin CD-menetelmiin kuuluvan release-putken luonti ja käyttöönotto. Toimeksiantajayrityksen työelämäohjaaja oli toteuttanut CI/CD-menetelmien käyttöönoton web-sovelluksen toiseen versioon, joten toimeksiantajalla oli tiedossa mahdollisia haasteita, joihin tuli kiinnittää huomiota toteutuksessa.

Opinnäytetyön tutkimusmenetelmä on tutkimuksellinen kehittämistoiminta. Tutkimuksellinen osuus painottuu DevOps-käsitteeseen ja sen historiaan, käyttöönotettavien menetelmien määritykseen sekä Azure DevOps -ympäristön käyttöönotettavien työkalujen kuvaamiseen. Kehittämistyön käytännön toteutus käsittää Azure Pipelines -työkalun build-putken luonnin ja käyttöönottoon, agentin luonnin ja asennuksen palvelimelle ja release-putken luonnin ja käyttöönoton.

Toimeksiantaja

Kehittämistyön toimeksiantaja on finanssialalle järjestelmäratkaisuja ja konsulttipalveluita toimittava Profit Software Oy. Profit Softwarella on toimistoja Suomessa, Virossa ja Ruotsissa. (Tietoa meistä n.d.) Profit Software tarjoaa erityisesti pankkiliiketoimintaan, perintään, henki- ja eläkevakuuttamiseen sekä varainhoitoon liittyviä palveluita; asiantuntijapalveluita, ohjelmistokehitystä, kokonaispalvelun- ja projektinhallintaa, analytiikkaa, palveluintegraatioita, tuotteistettuja ratkaisuja sekä käyttäjäkokemusta (Palvelut n.d.). Tuotteiden ja palveluiden lisäksi Profit tarjoaa analytiikan ja tiedolla johtamisen asiantuntemusta ja palveluita (Analytiikka n.d.).

2 Kehittämistyön tietoperusta

Teoreettinen viitekehys

Toimeksiantaja halusi DevOpsin CI/CD-menetelmät web-sovellukseen lisätäkseen automatisaatiota ja koska menetelmät olivat käytössä web-sovelluksen toisessa versiossa, sekä muutenkin osittain käytössä toimeksiantajalla. Toimeksiantajan toive oli, että web-sovelluksen koonti ja asennus palvelimelle tapahtuu oman agenttisovelluksen avulla. Koska toimeksiantajalla oli käytössä Azure DevOps -ympäristö ja DevOpsin CI/CD-menetelmät käytössä muissa sovelluksissa, ei ollut tarpeen tehdä erillistä vaatimusmäärittelyä käyttöönotosta tai määritellä tarkemmin, mitä Azure DevOps -ympäristön työkaluja haluttiin käyttöön.

Kehittämistyön tarkoitus on vastata kysymykseen, miten DevOpsin CI/CD-menetelmien käyttöönotto toteutetaan web-sovellukseen, kuinka agentti luodaan Azure DevOps -ympäristössä ja miten luotu agentti asennetaan palvelimelle. DevOpsin CI/CD-menetelmien käyttöönotto on melko suoraviivaista, mutta kehittämistyön kannalta mielenkiintoista on samalla selvittää, vaikuttaako menetelmien käyttöönottoon web-sovelluksen ikä, tekniikka tai ylipäätänsä se, että sovellus on jo käytössä ja sisältää paljon dataa ja jos vaikuttaa, kuinka paljon.

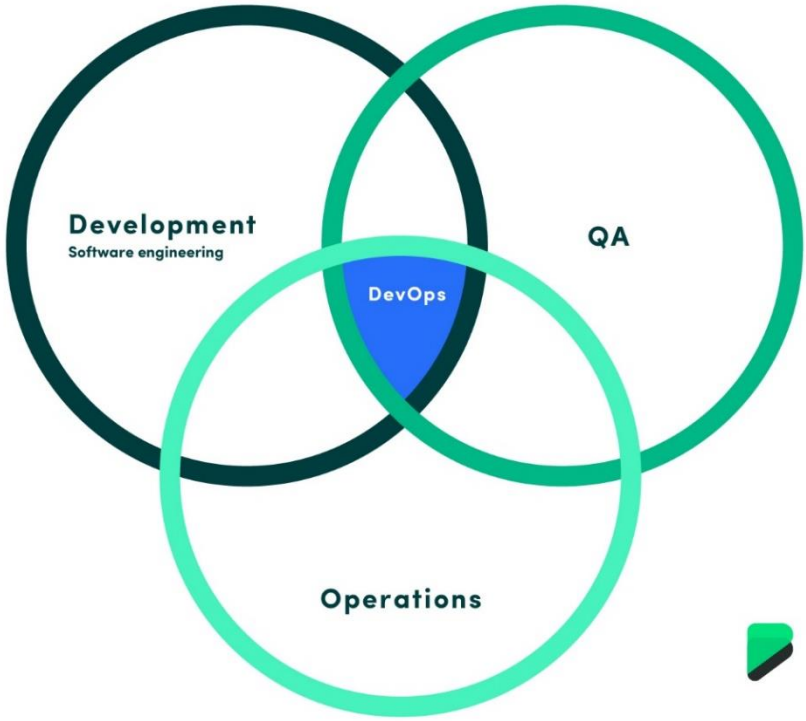
DevOpsin CI/CD-menetelmien käyttöönoton edellytyksenä on kehittämistyön tapauksessa Azure Pipelines -työkaluun kuuluvien build (koonti)- ja release (julkaisu)-putkien käyttöönotot, jotka mahdollistavat jatkuvan integraation sekä jatkuvan toimituksen ja jatkuvan julkaisun. Agentin asennusta varten tarvitaan palvelin. Ketteryyden lisäyksen lisäksi CI/CD-menetelmien käyttöönoton tavoite on luoda pohja automatisoidun testauksen käyttöönotolle. Koska käyttöönotetut CI/CD-menetelmät tuovat kehitystyöhön ketteryyttä, voidaan Lean ja Agile -menetelmien katsoa täydentävän DevOpsin ketteriä CI/CD-menetelmiä.

2.1 DevOpsin määrittely

DevOps on yhdistelmä englannin kielen sanoista development ja operations (Ilvonen 2019). Nimensä mukaisesti DevOps yhdistää kehityksen (development) ja tuotannon (operations), vaikkakaan Ilvosen mukaan termille ei vakiintunutta käsitettä (Ilvonen 2019). Myöskään vakiintunutta suomennosta tai suomenkielistä termiä DevOpsille ei ole, mutta toisaalta DevOps ei olekaan yksi, määritetty konkreettinen asia tai tekniikka, vaan enemmänkin abstrakti käsite tai ajattelumalli.

Australialainen ohjelmistoyritys Atlassian avaa verkkosivuillaan DevOpsin käsitettä seuraavasti; DevOps on ajatustapa, jossa tiimit omaksuvat uusia tapoja työskennellä. ”DevOps-kulttuurissa” lähtökohtana on yhteistyö, kehitystyö on yleensä yhden tiimin toteuttamaa full stack -kehitystä, joka mahdollistaa ohjelmistokehityksen tuomisen lähemmäs käyttäjiä ja heidän tarpeitaan. (DevOps principles n.d.)

Heikkisen (2021) mukaan DevOpsin keskeisin asia on ”kulttuurin luominen jatkuvalle oppimiselle kokeilun kautta, jotta asiakas saa parhaan mahdolliseen tuotteen nopeasti ja luotettavasti” (Heikkinen 2021). Kuten todettu, DevOps yhdistää kehitys- ja tuotantotiimit yhdeksi tiimiksi, jolla on yhteiset tavoitteet ja jonka toiminta näkyy asiakkaalle nopeutena ja luotettavuutena (kuvio 1).



Kuvio 1. DevOpsin suhteet (Introduction to DevOps: What it is, concepts, and objectives 2021)

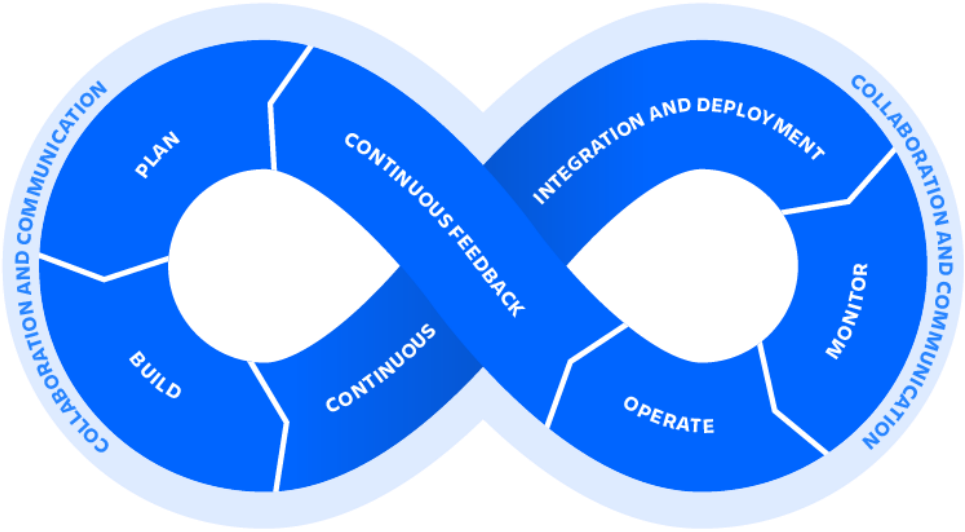
Kuviosta voidaan todeta DevOpsin olevan yhdistelmä ”devin” ja ”opsin” lisäksi laadunvarmistusta (QA). Laadunvarmistuksen voidaan todeta kuvion mukaan olevan olennainen osa DevOps-menetelmiä ja Atlassianin mukaan laadunvarmistus näkyy DevOps-menetelmien tavoitteessa, joka on

toteuttaa kehitystä nopeammin ja laadukkaammin kuin perinteisillä kehitysmenetelmillä on mahdollista. Nopeuden ja ketteryyden takaamiseksi automatisointi on tärkeässä osassa DevOps-menetelmissä, koska tiimin tuottavuus lisääntyy, kun kaikkia työvaiheita ei suoriteta manuaalisesti. (DevOps principles n.d.)

2.1.1 DevOpsin toiminnallisuus

Forsellin (2020) mukaan DevOpsin periaatteisiin kuuluu, että työvaiheet jaetaan itsenäisiksi osiksi ja organisointi tehdään työn etenemisen seuraaminen mahdollistaen. Työn seurattavuus helpottaa ennakointia, joka voi näkyä kiireen vähenemisenä. (Forsell 2020.) Pihlajamäen (n.d.) mukaan DevOps on ”joukko käytäntöjä ja kulttuurifilosofiaa, joita on mahdollista toteuttaa työkaluilla ja tekniikoilla, jotka automatisoivat ja integroivat ohjelmistokehityksen ja ylläpitotiimien välistä viestintää ja yhteistyötä.” (Pihlajamäki n.d.)

Alla olevassa kuviossa (kuvio 2) kuvataan DevOps-menetelmien toiminnallisuus. Kuten edellisessä kappaleessa Forsellin mukaan, myös kuvion mukaan DevOpsin työvaiheet on jaettu selkeiksi osiksi. Lisäksi kuten Pihlajamäki edellisessä kappaleessa, myös alla oleva kuvio korostaa yhteistyön roolia DevOps-menetelmissä.



Kuvio 2. DevOpsin työvaiheet (DevOps principles n.d.)

Kuten todettu, DevOps kehitystyössä yhteistyö on tärkeässä roolissa; kuvion mukaan yhteistyö on tärkeintä, sillä jatkuvassa yhteistyön ja kommunikoinnin yhdistelmän (collabration and communication) sisällä tapahtuu kaikki muu kehitys. Tuotannosta (operate) saa jatkuvaa palautetta (continuous feedback), mitä käytetään suunnittelussa apuna (plan). Suunnittelusta päästään toteutus vaiheeseen (build), mikä johtaa CI/CD-menetelmiin eli jatkuvaan integraatioon ja jatkuvaan toimitukseen sekä mahdollisesti jatkuvaan julkaisuun (continuous integration and deployment). CI/CD-menetelmiä seuraa monitorointi (monitor) mikä johtaa jälleen tuotantoon (kuvio 2).

2.1.2 DevOpsin historia

Ohjelmistokehityksen ensimmäisen kehitystavan, vesiputousmallin teoria esiteltiin jo vuonna 1970. Vesiputousmalli on vahvasti jakautunut tiimien välisiin osa-alueisiin ja siinä jokainen kehitysvaihe suoritetaan loppuun ennen seuraavaa. (Ilves & Luukkainen 2021.) Vuonna 2001 julkaistiin ketterän ohjelmistokehityksen julistus, Agile manifesto (kuvio 3), jossa puhutaan ketteryyden (agile) puolesta ohjelmistokehityksessä, tosin Blomqvistin (2019) mukaan ketteriä menetelmiä käytettiin kehitystyössä jo ennen agile-termin keksimistä (Blomqvist 2019).

Kuvio 3. Agile manifeston ydinsanoma (Ketterän ohjelmistokehityksen julistus 2001)

Kuvion mukaan ketterässä kehittämistyössä arvostetaan eniten yksilöitä ja kanssakäymistä, toimivaa ohjelmistoa, asiakasyhteistyötä sekä vastaamista muutokseen (kuvio 3). Ketterän ohjelmistokehityksen julistuksen mukaan ketterän kehityksen periaatteen mukaan muun muassa ketterät menetelmät, yksinkertaisuus, useat toimitukset ja lyhyet aikavälit ovat tärkeitä (Julistuksen takana olevat periaatteet 2001), ja kyseisten periaatteiden voidaan katsoa kuvaavan myös DevOpsin periaatteita. Vuonna 2003 ohjelmistokehitykseen tuotiin Lean-filosofia, mikä muun muassa mahdollisti uusien ohjelmistoversioiden julkaisut nopeammalla tahdilla ja lyhensi ohjelmiston kehityselinkaarta. (Waterfall vs. Agile vs. DevOps vs. Lean n.d.)

Varsinaisen DevOpsin voidaan Buchananin (n.d.) mukaan katsoa syntyneen vuosina 2007–2008, jolloin DevOpsin luvussa 2.1 määritetyt DevOpsin osapuolet toivat julki huolensa perinteisen ohjelmistokehityksen ongelmista; kehittäjät tekivät työtä omissa tiimeissä ja tuotantopuoli hoidettiin omassa tiimissä, jolloin merkittävä ongelma oli ”devin” ja ”opsin” irrallisuus toisistaan tiimien huolehtiessa vain omista osa-alueistaan. (Buchanan n.d.) Sekä Buchanan että Abildskov (2013) koros-

tavat kehityksen ja tuotannon ristiriitaisia tavoitteita. Lisäksi Abildskov mainitsee osapuolten ristiriitojen jopa ”kroonistuneen” ja ristiriitaisuuksien heijastuneen tuloksiin. Ristiriitojen lisäksi heikkojen tulosten katsottiin johtuvan yleisestikin perinteisistä ohjelmistokehitysprosesseista. (Abildskov 2013.)

Vaikka ajatus DevOpsista oli alkanut jo muotoutua tai ainakin tarve sille oli todettu aiemmin, tarkemmin DevOpsin voidaan katsoa syntyneen vasta vuonna 2009, jolloin pidettiin 10+ Deploys per Day: Dev and Ops Cooperation at Flickr -esitelmä, jossa todettiin ainoan järkevän tavan edistää ohjelmistokehitystä noudattavan DevOpsin periaatteita, eli ”devin” ja ”opsin” saumatonta yhteistyötä (Mezak 2018). DevOps-termin keksi Mezakin (2018) mukaan belgialainen Patrick Debois, joka ei päässyt 10+ Deploys per Day: Dev and Ops Cooperation at Flickr -esitelmään (Mezak 2018). Paulin (2014) mukaan Debois päätti järjestää oman tapahtuman Belgiassa ja keksi tapahtumalle nimen DevOpsDays, josta hän lyhensi sanan DevOps (Paul 2014).

2.1.3 DevOpsin hyödyt

DevOpsin tärkein hyöty toimeksiantajayrityksen asiantuntijoiden (2022) mukaan on juuri ketteryys ja sen mahdollistama nopeus. Myös luvussa 2.1 kuvattu yhden tiimin malli, jossa samat henkilöt hoitavat sekä kehittämisen että toiminnallisuuksien asennukset korostuvat DevOps-menetelmien hyödyissä myös toimeksiantajayrityksen asiantuntijoiden mukaan. (DevOpsin hyödyt korostuvat tuotantoonviennissä – valttina ketteryys ja nopeus 2022.)

DevOps on vakiinnuttanut paikkansa yrityksissä. Sekä Heikkinen (2021) että Halttunen (2022) korostavat DevOps-menetelmien suosiota yritysmaailmassa. Heikkisen mukaan Atlassianin 2020 DevOps Trends Survey kyselyssä todettiin, että suurin osa vastaajista koki DevOps-menetelmillä olleen positiivinen vaikutus, ja suurimman vaikutuksen olleen tuotteiden laadun parantuminen (Heikkinen 2021). DevOps-menetelmien suosio perustuu Halttusen mukaan sen ”helppoon omaksumiseen ja muovattavuuteen”, jolloin yritys voi toteuttaa DevOps-menetelmien käyttöönoton halutulla tavalla; jotkut haluavat lisätä automatisaatiota, toiset omaksua enemminkin DevOps-filosofian ja lisätä tiimien yhteistyötä, jotkut taas haluavat sekä että. (Halttunen 2022; Heikkinen 2021).

2.2 CI/CD-menetelmät

Kuten todettu luvussa 2.1.1, CI/CD-menetelmät ovat vain osa DevOpsin toiminallisuutta. Termit jatkuva integraatio, jatkuva toimitus ja jatkuva julkaisu menevät helposti sekaisin, minkä vuoksi on tärkeää varmistaa, että kaikki asianomaiset käyttävät samoja määritelmiä menetelmistä (Rossel 2017, 8). Lähteestä riippuen termejä on käytetty eri tavoin ja oman hankaluutensa termien määrittelylle tuo niiden suomentaminen, jolloin termin merkitys saattaa muuttua. Alla olevassa kuviossa (kuvio 4) on kuvattu DevOpsin CI/CD-menetelmien vaihteet Red Hat -verkkosivun mukaan.

Kuvio 4. CI/CD-menetelmien toiminta (What is CI/CD? n.d.)

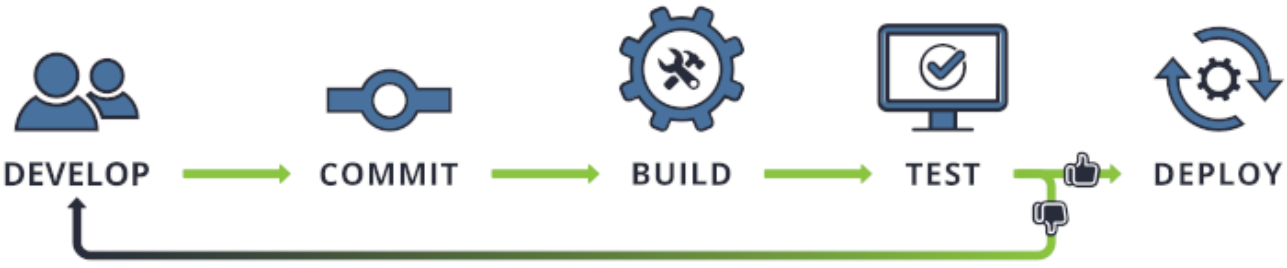
Kuviosta voidaan todeta kuinka CD-menetelmä tarkoittaakin kahta menetelmää, jatkuvaa toimitusta ja jatkuvaa julkaisua (continuous delivery, continuous deployment). Yleisesti CI/CD-menetelmien toiminnasta puhutaan putkina (pipeline), ja CI/CD pipeline on toiminallisuudeltaan data pipeline. Data pipeline tarkoittaa Hazelcast verkkosivuston artikkelin What Is a Data Pipeline? (n.d.) mukaan tietojenkäsittelyn vaiheiden esittämistä sarjoitetusti. Data pipeline -putkessa jokaisen vaiheen lopputulos on seuraavan vaiheen syöte. (What Is a Data Pipeline? n.d.)

Aiemmasta kuviosta (kuvio 4) voidaan todeta CI/CD-menetelmien toiminnan vastaavan data pipeline -putken määritelmää, eli vaiheet käydään sarjana läpi yksitellen ja edellisen vaiheen lopputulema vaikuttaa seuraavan vaiheen suoritukseen. Seuraavissa alaluvuissa kuvataan tarkemmin CI/CD-menetelmien periaatteista.

2.2.1 Jatkuva integraatio (CI-menetelmä)

CI (continuous integration), eli jatkuvan integraation menetelmä on DevOps-menetelmien prioriteetti, mikä automatisoi useamman kehittäjän koodimuutosten integroinnit (Rehkopf n.d.). Fowlerin (2006) mukaan jatkuvan integraation suurin hyöty on riskien pieneneminen; jos integraatio tehdään vasta valmiin ohjelmistoprojektin lopussa, on vaikea arvioida työn ja ajan määrää mikä integraatioon menee. Jatkuva integraatio poistaa arvioinnin riskin kokonaan ja kehittäjä pysyy ajan tasalla kehitystyön tilanteesta. Myös virheiden korjaus helpottuu, kun niitä ei tarvitse etsiä kaukaa koodista vaan versionhallinnan avulla muutoksia voi verrata aiempaan, virheettömään versioon. (Fowler 2006.) Edellisen luvun kuvion (kuvio 4) mukaan jatkuva integraatio on pohjana jatkuvalle toimitukselle ja jatkuvalle julkaisulle.

Rosselin (2017) mukaan jatkuvassa integraatiossa on perustana ohjelmiston toiminnan turvaaminen uuden koodin lisäyksen jälkeen. Jatkuvan integraation (sekä jatkuvan toimituksen ja julkaisun) tavoite on ohjelmistokehitysvaiheiden automatisointi sekä saada ohjelmistokehityksen vaiheet automatisoitua siihen pisteeseen, että periaatteessa kuka tahansa tiimin jäsen voi julkaista valmiin ohjelmiston. CI/CD-menetelmien käyttöönotto voi olla työlästä, etenkin jos kyseessä on jo olemassa oleva projekti, mutta se on vaivan arvoista, koska pidemmällä tähtäimellä menetelmien käyttöönotto näkyy ohjelmiston parempana laatuna ja ohjelmistovirheiden vähenemisemä. (Rossel 2017, 7.) Jatkuvan integraation vaiheet voidaan kuvata omana putkena (kuvio 5).



Kuvio 5. Jatkuvan integraation vaiheet (Reece 2019)

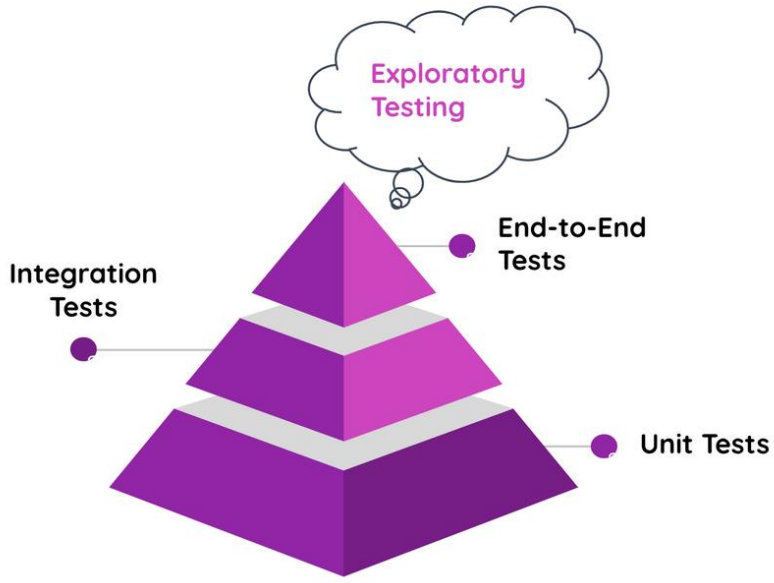
Reecen (2019) mukaan (kuvio 5) jatkuvan integraation vaiheet ovat kehitys (develop), muutosten lisäys (commit), koonti (build) ja testaus (test). Jos testit menevät läpi, voidaan siirtyä julkaisuun.

Jos testausvaiheessa ilmenee virheitä, siirrytään takaisin kehitysvaiheeseen. (Reece 2019.) Testauksen mennessä läpi CI-menetelmä integroi kehittäjän tekemät muutokset lähdekoodiin automaattisesti ja CD-menetelmät niin sanotusti jatkavat siitä, mihin jatkuva integrointi loppuu.

2.2.2 Automatisoitu testaus

Automatisoitu testaus, eli testausautomaatio on ohjelmiston automaattista tarkistamista ja jatkuvaa validointia. Testausautomaation tarkoitus on pitää huolta, että ohjelmiston koodi ja itse ohjelmisto vastaavat laatuvaatimuksia. (Hristov n.d.) Sekä edellisessä luvussa esitetty kuvio (kuvio 5) että Hristov esittävät testauksen osaksi jatkuvaa integraatiota. Hristov korostaa, että testaukseen kuuluvat tehtävät tulee automatisoida ja kohdistaa jatkuvan integraation vaiheeseen. (Hristov n.d.)

Microsoftin dokumentaation Run quality tests in your build pipeline by using Azure Pipelines (n.d.) Learn-oppimisalustan kurssin materiaali korostaa hyvän testin teon noudattavan seuraavia ohjeita; testien tulee olla tiettyyn tarkoitukseen kirjoitettu, testien tulee olla lyhyitä, jotta sujuvuus säilyy, testien tulee olla toistettavissa, testien tulee kattaa vain kehittäjän oma koodi ja testien tulee testata vain yhtä osaa kerrallaan. (Run quality tests in your build pipeline by using Azure Pipelines n.d.) Erilaiset testaustavat voidaan jakaa neljään osaan (kuvio 6).



Kuvio 6. Erilaisia testaustapoja (Hristov n.d.)

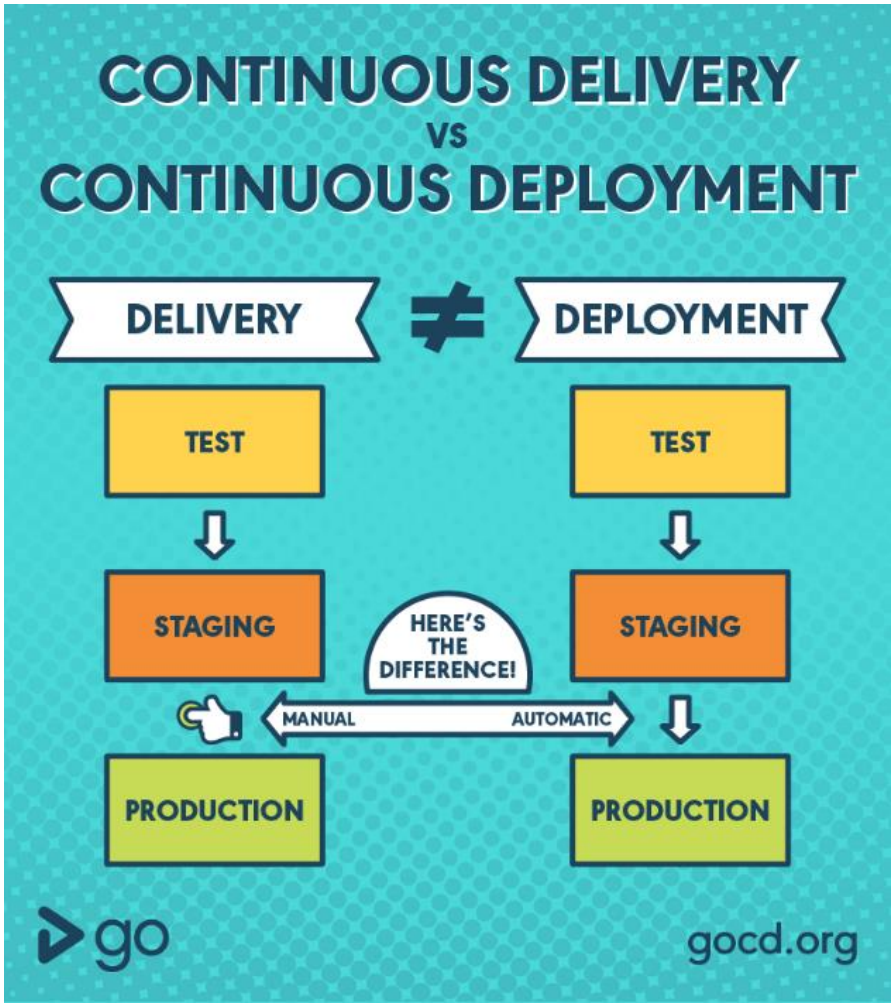
Hristovin (n.d.) mukaan testaustavat, joita voidaan automatisoida ovat yksikkötestaus, integraatiotestaus ja päästä päähän -testaus (kuvio 6). Kuvion mukaisesti eniten suoritetaan yksikkötestejä (unit tests) eli koodin testausta ja vähiten päästä päähän (end-to-end tests) -sekä tutkivia testejä (exploratory testing), eli käyttöliittymään liittyviä testejä.

Testauksen tärkeys korostuu Klemetin (2019) mukaan sen roolista ohjelmistokehityksessä sekä tuotannossa mahdollisesti ilmenevien häiriöiden korjauksessa. Testausautomaation hyödyiksi DevOps-menetelmissä Klemetti mainitsee testausautomaation mahdollistaman mahdollisuuden, millä ”pyritään vähentämään aikaa järjestelmämuutosten ja tuotantoon viennin välillä.” (Klemetti 2019.) Myös Hristov (n.d.) korostaa testausautomaation lisäävän nopeutta, DevOps-menetelmien etuja ja käytön tehokkuutta. Hristovin mukaan DevOps saavuttaa ”täyden potentiaalinsa” testausautomaation käyttöönoton avulla. (Klemetti 2019; Hristov n.d.)

2.2.3 Jatkuva toimitus ja jatkuva julkaisu (CD-menetelmät)

CD-menetelmä tarkoittaa kahta asiaa, jotka ovat jatkuva toimitus (continuous delivery) ja jatkuva julkaisu (tai käyttöönotto suomennoksesta riippuen) (continuous deployment). Montonen (2020) toteaa jatkuvan toimituksen olevan laajennettu versio jatkuvasta integroinnista, sillä jatkuva toimitus valmistelee jatkuvan integroinnin vaiheen koodimuutokset tuotantoa varten (Montonen 2020).

Siirtymä jatkuvasta integraatiosta jatkuvaan toimitukseen on yleensä automatisoitu, mutta päätös jatkuvaan julkaisuun siirtymisestä on kehittäjän päätöksen takana. Jatkuva julkaisu automatisoi muutosten käyttöönoton tuotantoon ja sitä voidaan pitää ohjelmistoyritysten päätavoitteena. (Continuous Delivery vs. Continuous Deployment: Where to draw the line? n.d.) Jatkuva toimitus ja jatkuva julkaisu eroavat toisistaan vain tuotantoon viennin osalta (kuvio 7).



Kuvio 7. Jatkuvan toimituksen ja jatkuvan julkaisun ero (Peña 2017)

Kuvion mukaan molemmat CD-menetelmät sisältävät vaiheet testaus (test), demoympäristöön vienti (staging) ja tuotantoon vienti eli julkaisu (production). Kuvion mukaan ainoa ero menetelmillä on, että jatkuvassa toimituksessa julkaisu tehdään manuaalisesti, kun jatkuvassa julkaisussa julkaisu on automatisoitu. Myös Koriseva (2018) tähdentää, että automaattinen julkaisu tehdään jatkuvan julkaisun menetelmällä, ”mutta muutokset voidaan myös julkaista manuaalisesti, mikä on tavanomaista jatkuvassa toimituksessa.” (Koriseva 2018; Peña 2017.)

2.3 Azure DevOps -ympäristö

Toimeksiantajan käyttämä DevOps-työkalu oli Microsoftin Azure DevOps. Azure DevOps on SaaS-pohjainen palvelu, jota voidaan käyttää ohjelmointikielestä, alustasta tai pilvestä riippumatta. (What is Azure DevOps? n.d.) Azure DevOps -ympäristön käyttöönottoa varten Microsoftilta löytyi runsaasti dokumentaatiota. Microsoftin Learn-alustalla oli opintokokonaisuus Azure DevOps -ympäristön käyttöönotosta, lisäksi Microsoftin Documentation-osiossa oli yksittäisten työkalujen, kuten Azure Pipelines -työkalun, käyttöönotto ohjeistuksia ja dokumentaatiota. Seuraavissa alaluvuissa käsitellään tarkemmin Azure Pipelines -ympäristön työkalujen toimintaa.

2.3.1 Azure Pipelines -työkalu

Luvussa 2.2 kerrottiin CI/CD-menetelmien toimivan CI/CD-putkina, ja Azure Pipelines on työkalu, jolla CI/CD-putkia luodaan ja muokataan. Koska kehittämistyön tarkoitus oli nimenomaan CI/CD-menetelmien käyttöönotto, kehittämistyön toteutuksen kannalta olennaisin työkalu oli Azure Pipelines. Azure Pipelines suorittaa koonnin ja automatisoidun testauksen mahdollistaen jatkuvan integraation ja jatkuvan toimituksen; jatkuva integrointi tuottaa artefaktin, mitä voidaan testata tai ottaa käyttöön jatkuvan toimituksen mukaisesti (What is Azure Pipelines? 2022). Azure Pipeline putken määrityksessä on mahdollista hyödyntää ennalta määritettyjä muuttujia, jotka ovat pääasiassa read-only muodossa (Use predefined variables 2022). Azure Pipelines -työkalussa CI-putki on koonti putki (build pipeline) ja CD-putki julkaisu putki (release pipeline).

YAML

Azure Pipelines -työkalulla koonti putkea hallitessa YAML-kieli on tärkeässä osassa. Red Hatin What is YAML? (2021) artikkelin mukaan YAML on datan käsittelyyn liittyvä kieli, jota käytetään yleensä konfiguraatiotiedostoissa (What is YAML? 2021), ja myös koonti putken hallinnassa YAML-tiedostoa käytetään juuri konfiguraatiossa. YAML-tiedoston tiedostopääte on yml.

2.3.2 Azure Repos -työkalu

Azure Repos tarjoaa työkaluja koodin versionhallintaan. Versionhallinnan avulla kehittäjä tai tiimi pysyvät koodista ajan tasalla ja se mahdollistaa koodiin tehtyjen muutosten tarkastelun sekä palaamisen aiempaan versioon tarvittaessa. (What is Azure Repos? 2022.) Azure Repos on graafinen käyttöliittymä versionhallintaan, eli versionhallintapalvelu.

Versionhallinta

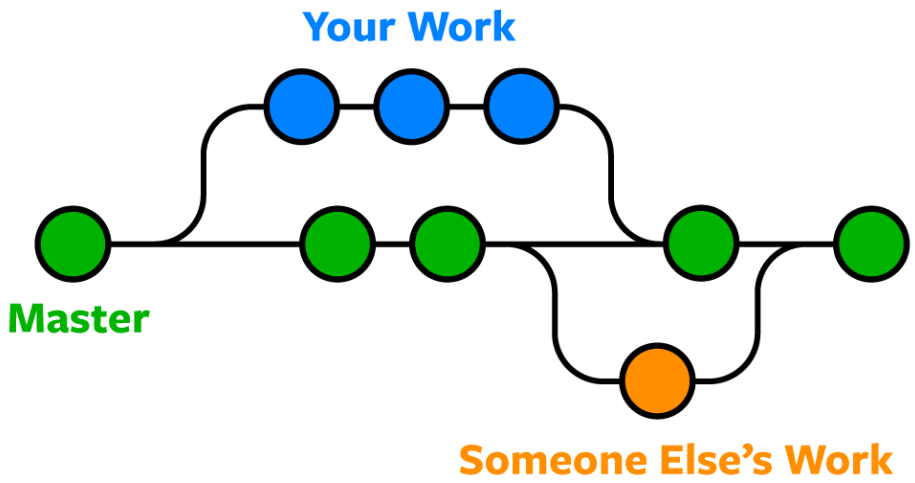
Kuten aiemmassa kappaleessa sivuttiin, versionhallinta on auttaa kehittäjiä koodimuutosten tarkastelussa. Vanhatapio (2021) lisää, että versionhallinta käyttää metadataa versioiden erojen kuvaamiseen. Versionhallinnasta voi todeta millaisia muutoksia on tehty ja milloin, kenen toimesta ja mihin tiedostoon. (Vanhatapio 2011.) Sen lisäksi, että versionhallinta mahdollistaa tehtyjen muutosten tarkastelun, se myös säilöö koodia mahdollistaen varmuuskopioinnin sekä koodin jakamisen (Karhusaari 2022).

Forsellin (2020) mukaan versionhallinta on ”ensimmäinen askel kohti DevOpsin kaltaista toimintamallia”. Forsell myös toteaa DevOps-työkalujen tuen muille kuin Git-versionhallintajärjestelmälle olevan niin surkea, että DevOps-menetelmiä käyttöönotettaessa Git on käytännössä ainoa vaihtoehto versionhallintajärjestelmäksi. (Forsell 2020.)

Git

DevOps-menetelmien käyttöönottoa, sekä versionhallintaa, varten toimeksiantajalla oli käytössä Git-versionhallinta. Kuten aiemmassa kappaleessa todettiin, Git on käytännössä ainoa vaihtoehto DevOps-menetelmiä käyttöönottaessa, ja Git onkin Atlassianin What is Git -tutoriaalin (n.d.) mukaan käytetyin versionhallintajärjestelmä. Git on hajautettu versionhallinajärjestelmä, eli jokaisen kehittäjän työkopio on oikeastaan koko repositorio, joka voi sisältää koko muutoshistorian. (What is Git n.d.) Git-komentojen ajamiseen voidaan hyödyntää esimerkiksi Visual Studio Code -koodieditoriin sisäänrakennettua komentoriviä, mutta kehittämistyössä käytettiin Git-komentojen ajoon erillistä Git Bash -komentorivisovellusta.

Versionhallinnan käyttö mahdollistaa tiimin kehittäjille paikalliset kopiot päärepositoriosta. Versionhallinnan avulla kehittäjä voi tehdä omasta lokaalista kopiostaan haaran (branch), jolloin kehittäjän tekemät koodimuutokset eivät mene heti tiimin muille kehittäjille. Tästä on hyötyä muun muassa silloin, jos kehittäjän tekemät muutokset aiheuttavat virheen; kuten aiemmassa kappaleessa todettu, versionhallintaan jää jälki muutosten tekijästä ja muutosten laadusta, jolloin mahdollisen virheen aiheuttavan koodin paikannus on helpompaa. Alla olevassa kuviossa (kuvio 8) on esitelty versionhallinnan haaran toimintaa.



Kuvio 8. Git haaran toiminta (Git Branches: List, Create, Switch to, Merge, Push, & Delete 2021)

Versionhallinnan toiminta perustuu kuvion mukaan haarojen luomiseen ja sulauttamiseen (merge). Kehittäjä kloonaa päähaarasta (master) lokaalin kopion repositoriosta, johon tekee muutoksia. Ennen muutosten aloitusta kehittäjä luo uuden haaran muutoksia varten. Samanaikaisesti toinen kehittäjä voi tehdä muutoksia omaan haaraan omassa lokaalissa lähdekoodi kopiossa. Kun kehittäjän tekemät muutokset ovat valmiit, hän voi työntää (push) muutokset päähaaraan. Kun muutokset on sulautettu (merge) päähaaraan, ja lähdekoodiin, muut kehittäjät voivat hakea muutokset (pull). Ideana on, että päähaaraan ei tehdä suoraan muutoksia, vaan aina erillisen haaran kautta.

2.3.3 Agent pools -työkalu

Agent pools ei ole samanlainen työkalu kuin aiemmissa luvuissa mainitut Azure Pipelines ja Azure Repos. Agent pools on Azure DevOpsin asetus, jossa on listattuna Azure DevOps -organisaation agentit. Asetusten alta agentteja pystyy lisäämään, mikä voi tulla tarpeen isommassa organisaatiossa, tai kuten kehittämistyössä, jos haluaa luoda oman agentin.

Agentti

Microsoftin Learn oppimisalustan Host your own build agent in Azure Pipelines (n.d.) kurssin materiaalin mukaan agentti on systeemi, joka suorittaa koontitehtäviä. Agentti voi olla asennettuna virtuaalikoneella tai omassa järjestelmässä, kuten palvelimella. Oman agentin (self-hosted agent) hyötyjä ovat muun muassa koonnin kestoajan väheneminen, mahdollisuus suurempaan levytilaan

ja suorittimen ja muistin määrään sekä tiedostojen jakamisen helpottuminen. (Host your own build agent in Azure Pipelines n.d.) Agentti voi suorittaa vain yhden tehtävän kerrallaan.

2.3.4 Muut Azure DevOps -ympäristön työkalut

Azure Boards

Azure Boards on ohjelmistoprojektinhallintatyökalu, joka helpottaa tehtävien ajoitusta. Se sisältää muun muassa tuen kanban-taululle sekä Agile ja Scrum -menetelmien käyttöön. (What is Azure Boards? 2022.)

Azure Test Plans

Azure Test Plans tarjoaa työkaluja laadun ja yhteistyön parantamiseen. Azure Test Plans on selain-pohjainen palvelu, joka tukee sekä manuaalisia että automatisoituja testejä. Kehittämistyön kan-nalta olennaista on, että työkalu on integroitu Azure Pipelines -työkalun kanssa ja tukee näin CI/CD-menetelmiä. (What is Azure Test Plans? 2022.)

Azure Artifacts

Azure Artifacts on työkalu, mikä helpottaa koodin jakamista ja pakettien hallintaa. Artefaktien jako on mahdollista myös julkisesti, joten kehittäjätkin voivat käyttää julkisia paketteja, esimerkiksi NuGet-paketteja. (Azure Artifacts overview 2022.) Vaikka CI-putki luo artefaktin, se ei näy auto-maattisesti täällä.

3 Kehittämistyön toteutus

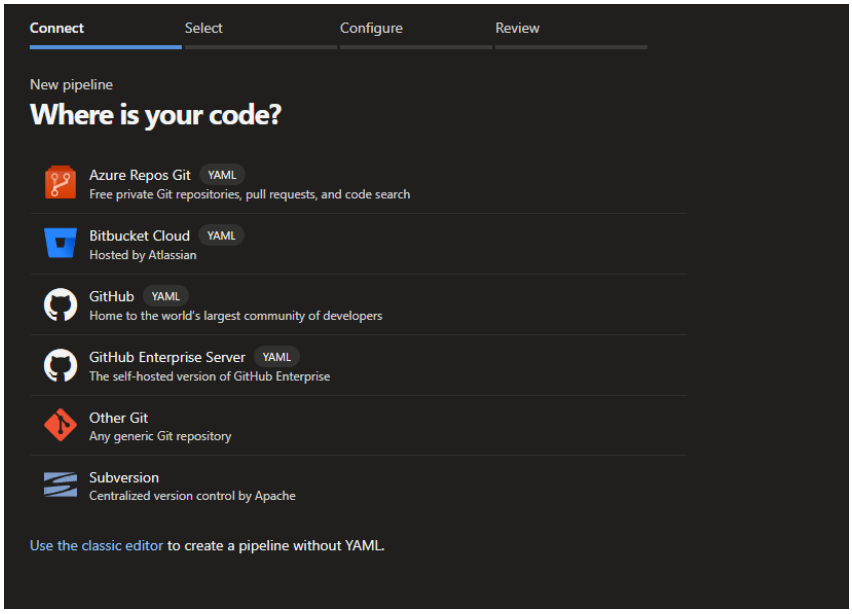
Kehittämistyön toteutuksen tavoite oli saada CI-putkesta ulos web-sovelluksen asennustiedostot sisältävä artefakti, joka purettiin ja asennettiin CD-putkella toimeksiantajan palvelimelle käyttäen omaa agenttia. Jatkossa toimeksiantaja voi hyödyntää artefaktia myös asiakasympäristöasennuksiin. Luvun 2.3.4 mukaisesti Azure DevOps tarjoaa tuen myös kanban-taululle, mutta kehittämistyön toteutuksessa ei ollut tarvetta eritellä käyttöönoton vaiheita tehtäväkorteille.

Kehittämistyön toteutuksen työvaiheet olivat:

1. CI, eli koonti putken luonti ja siihen jatkuvan integraation käyttöönotto käyttäen YAML-dokumenttia
2. Tehtävien lisäys koonti putkeen käyttäen YAML-dokumenttia
3. Agentille alustan (agent pool) luominen, agenttisovelluksen asennus toimeksiantajan palvelimelle
4. CD, eli julkaisu putken luonti, tehtävien määritys ja putkeen jatkuvan julkaisun käyttöönotto käyttöliittymän kautta

3.1 CI-putken luonti (Build Pipeline)

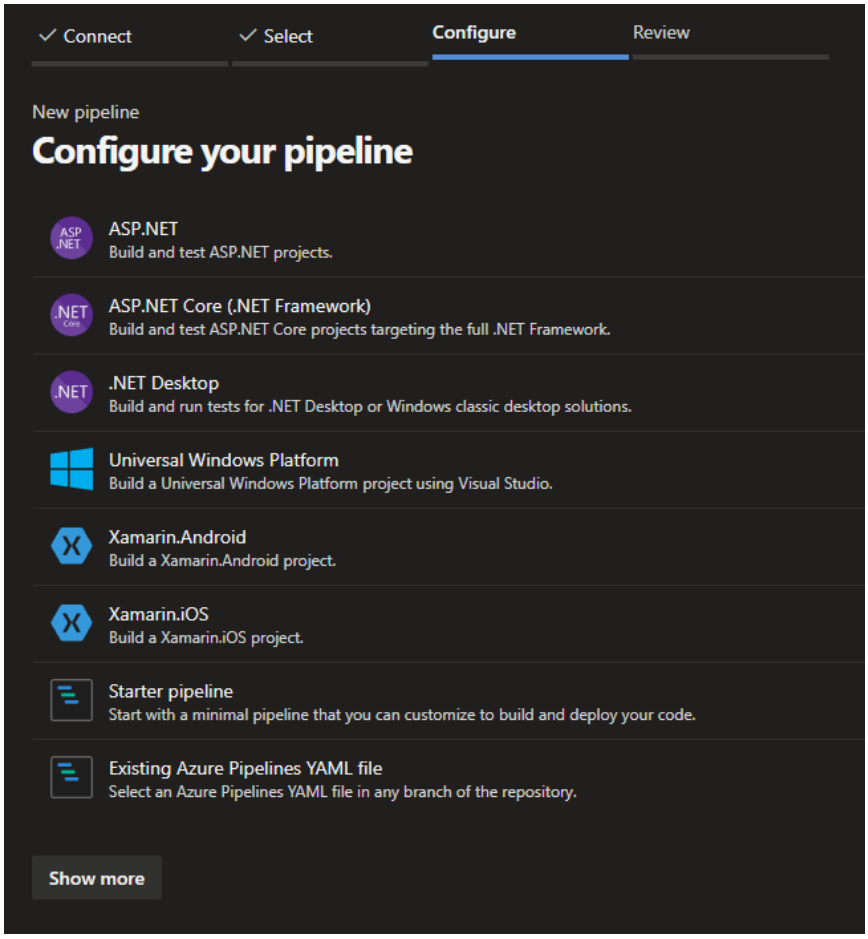
CI-putken toiminta myötäilee luvussa 2.2.1 esitetyn kuvion 5 toiminallisuutta. Ensin on kehitys, jonka jälkeen koodimuutokset lisätään lähdekoodiin versionhallinnan avulla. Lähdekoodin käännös, eli koontivaiheen hoitaa CI-putki, jonka tarkoitus oli kääntää web-sovellus ja luoda pakattu kansio, joka sisälsi web-sovelluksen asennustiedostot. CI-putken tehtäviin voi lisätä testejä, mutta vaikka testejä ei lisäisikään, putken koonnin onnistuminen on edellytys seuraavaan vaiheeseen siirtymisessä. Itse CI-putken luonti oli kohtalaisen suoraviivaista, luontia varten tarvittiin yhteys koodiin (connect), versionhallinnan ja projektin valinta (select), konfigurointi (configure) ja valintojen tarkastaminen (review) (kuvio 9).



Kuvio 9. Kuvakaappaus Azure DevOps -ympäristön tukemista versionhallintapalveluista

Kehittämistyön versionhallinnassa käytettiin Gitiä, ja versionhallintapalveluna Azure Repos -työkalua, joten lähdekoodin sijainniksi valittiin Azure Repos Git. Kuten kuviosta voidaan todeta, versionhallinnoista Azure Repos Git, Bitbucket Cloud, GitHub sekä GitHub Enterprise Server käyttivät YAML-tiedostoa koonnin määritykseen, mutta putki oli mahdollista luoda myös ilman YAML-tiedostoa (kuvio 9).

Putken konfigurointivaiheessa valittiin alla olevan kuvion (kuvio 10) mukaisesti konfigurointiasetukset. Konfigurointiasetus määrittää pohjan YAML-tiedostolle, mutta sitä pystyy muuttamaan tarvittaessa. Valmiita malleja oli esimerkiksi ASP.NET:ille, .NET Desktopille, Universal Windows Platformille. Ohjelmointikieli ja tekniikka, joilla sovellus on toteutettu, vaikuttavat mallin valintaan.



Kuvio 10. Kuvakaappaus konfigurointipohjan valinnasta Azure DevOps -ympäristössä

Kehittämistyössä putken konfigurointipohjaksi tuli ASP.NET ohjelmistokehys (kuvio 10), koska kyseessä oli pääosin VB.NET-kielellä toteutettu web-sovellus. Seuraavassa vaiheessa pystyi halutessaan muokkaamaan valitun pohjan mukaista YAML-tiedostoa, joka kehittämistyössä oli käytännössä pakollista, tai koota ja suorittaa (build and run) putken Azure Pipelines -työkalun generoimalla oletustiedostolla. Vaikka putken pystyisi suorittamaan oletustiedostolla, tulisi sitä todennäköisesti joka tapauksessa jossain vaiheessa muokata paremmin sovelluksen tarpeisiin sopivaksi. Koonti suorittaa YAML-tiedostossa olevat määritykset, ja agentti suorittaa siinä asetetut tehtävät. Alla on ASP.NET ohjelmistokehykseen muodostettu YAML-tiedostopohja.

```
# ASP.NET
# Build and test ASP.NET projects.
# Add steps that publish symbols, save build artifacts, deploy, and more:
# https://docs.microsoft.com/azure/devops/pipelines/apps/aspnet/build-aspnet-4
```

```
trigger:
- master

pool:
vmImage: 'windows-latest'

variables:
solution: '**/*.sln'
buildPlatform: 'Any CPU'
buildConfiguration: 'Release'

steps:
- task: NuGetToolInstaller@1

- task: NuGetCommand@2
inputs:
restoreSolution: '$(solution)'

- task: VSBuild@1
inputs:
solution: '$(solution)'
msbuildArgs: '/p:DeployOnBuild=true /p:WebPublishMethod=Package /p:PackageAsSingle-
File=true /p:SkipInvalidConfigurations=true /p:PackageLocation="$(build.artifactStag-
ingDirectory)"'
platform: '$(buildPlatform)'
configuration: '$(buildConfiguration)'

- task: VSTest@2
inputs:
platform: '$(buildPlatform)'
configuration: '$(buildConfiguration)'
```

YAML-tiedostossa voi hyödyntää muuttujia, joilla voi esimerkiksi määritellä Visual Studio Solution -tiedoston (.sln-tiedostopääte), joka kootaan. Kuten aiemmassa kappaleessa todettiin, kehittämistyön web-sovelluksen kanssa ei ollut mahdollista suorittaa putkea oletusasetuksilla, koska se ei sisältänyt solution-tiedostoa (mikä määritellään muuttujissa oletuksena) versionhallinnassa, eikä

käytössä ollut NuGet-paketteja; kuten esimerkki koodista voidaan todeta, niiden määrittely oli mukana automaattisesti.

CI-putken käyttöönotossa hyödynnettiin versionhallintaa tekemällä käyttöönottoa varten oma haara, johon muutokset ajettiin. YAML-tiedoston muutokset pystyi tekemään Azure Pipelines -käyttöliittymän kautta tai erillisellä koodieditorilla, jolloin muutokset tuli manuaalisesti työntää (push) Azure Repos -työkaluun Git Bash -sovelluksen avulla. Muutosten teon jälkeen putki tuli suorittaa (run) manuaalisesti niin kauan, kunnes jatkuva integraatio otettiin käyttöön; jos muutokset olisi tehty suoraan master-haaraan, jatkuva integrointi olisi ollut oletuksena käytössä ja putki olisi suoritettu automaattisesti heti.

Pääasiassa kaikki CI-putkeen käyttöönottoon liittyvät määritykset toteutettiin samalla kaavalla:

1. Tehtävien lisäys, joko YAML-tiedostoon koodieditorilla tai käyttöliittymän kautta, tai hakemalla käyttöliittymän kautta
2. Muutosten lisäys ja työntö Azure Repos -työkaluun luotuun haaraan Git Bashilla
3. Putken suoritus (myöhemmin suoritus tapahtui automaattisesti jatkuvan integraation ansiosta)
4. Lopputuloksena syntyi web-sovelluksen asennustiedostot sisältävä pakattu kansio (artefakti)

3.1.1 Tehtävien lisäys (Build Tasks)

Kun putki oli luotu, seuraava vaihe oli koontitehtävien (build tasks) lisäys. Kuten edellisen luvun koodista voidaan todeta, YAML-tiedostolle voi antaa useita erilaisia määreitä tai tehtäviä suoritettavaksi. YAML-tiedostossa määritetään, millaista agenttia koontitehtävien suorittamiseen käytetään (pool), ja annetaan mahdolliset muuttujat (variables). Tehtävät (tasks) kuitenkin määrittävät mitä YAML-tiedoston tulee tehdä. Tehtäviä pystyy lisäämään myös käyttöliittymän kautta.

Kuten luvussa 2 todettiin, CI/CD-menetelmät olivat jo toimeksiantajalla käytössä web-sovelluksen toisessa versiossa. Koontitehtävien määrittelyä helpotti se, että ne oli jo kerran määritetty toista versiota varten, ja toisen version YAML-tiedostosta pystyi tarkentamaan, mitä tehtäviä toimeksiantaja oli aiemmin ottanut käyttöön, jotta artefakti muodostui oikein. Aiempaa YAML-tiedostoa ei kuitenkaan voinut suoraan kopioida kehittämistyötä varten, sillä toinen web-sovellusversio oli Web Application -projekti, kun kehittämistyö oli Web Site -projekti.

Web-sovellus oli niin laaja kokonaisuus, että yksinkertaisinta oli lisätä yksi tehtävä kerrallaan YAML-tiedostoon ja suorittaa putki ennen uuden tehtävän lisäystä. Kuten luvussa 2.2.1 kuvattiin jatkuvan integroinnin hyödyistä, jo jatkuvan integroinnin käyttöönotossa oli järkevin toimintatapa integroida jokainen muutos erikseen, jolloin mahdollisen virheen jäljitys oli helpompaa.

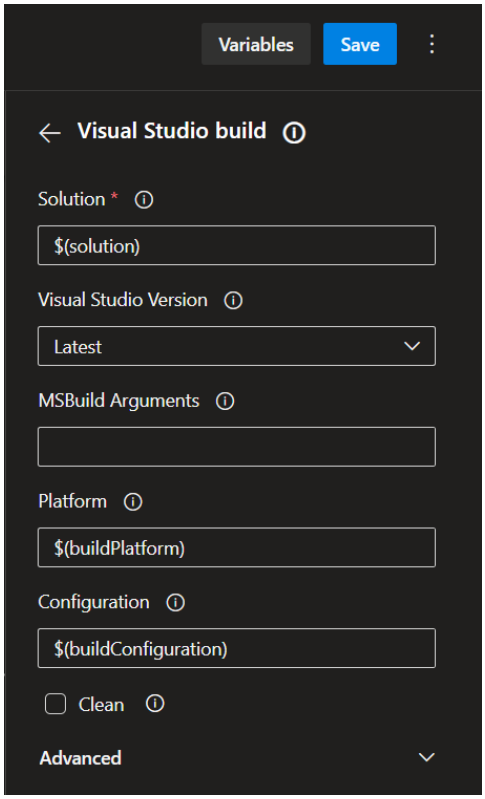
Visual Studio Build

VSBuild-tehtävä kokoaa (build) sovelluksen YAML-tiedoston muuttujien pohjalta. VSBuild käyttää pohjana MSBuild-tehtävää, eli MSBuild suoritetaan joka tapauksessa; VSBuild lisää käännökseen Visual Studion version. Kuten aiemmin todettu, web-sovelluksella ei ollut versionhallinnassa solution-tiedostoa, vaan solution-tiedostoksi asetettiin website.publishproj-tiedosto. Kehittämistyössä VSBuild-tehtävää käytettiin web-sovelluksen käyttöliittymän koonnissa. Alemmassa kuviossa (kuvio 11) on kuvattu YAML-koodi määrityksistä, joita VSBuild-tehtävään voidaan asettaa.

```
# Visual Studio Build
# Build with MSBuild and set the Visual Studio version property
- task: VSBuild@1
  inputs:
    #solution: '**\*.sln'
    #vsVersion: 'latest' # Optional. Options: latest, 16.0, 15.0, 14.0, 12.0, 11.0
    #msbuildArgs: # Optional
    #platform: # Optional
    #configuration: # Optional
    #clean: false # Optional
    #maximumCpuCount: false # Optional
    #restoreNugetPackages: false # Optional
    #msbuildArchitecture: 'x86' # Optional. Options: x86, x64
    #logProjectEvents: true # Optional
    #createLogFile: false # Optional
    #logFileVerbosity: 'normal' # Optional. Options: quiet, minimal, normal, detailed, diagnostic
```

Kuvio 11. Visual Studio build -muuttujat (Visual Studio Build task 2022)

Kuten kuviostakin voidaan todeta, VSBuild-tehtävä mahdollistaa Visual Studio version määrittämisen (vsVersion), mutta se mahdollistaa myös MSBuild-muuttujat (msbuildArgs). Kuten tehtävät, myös tehtävien muuttujat voidaan määrittää joko YAML-dokumentilla tai käyttöliittymän kautta (kuvio 12).



Kuvio 12. Kuvakaappaus muuttujien lisäyksestä Azure DevOps -ympäristössä

Kuviosta voidaan todeta, että olennaisimmat muuttujat VSBuild-tehtävässä ovat solution-tiedosto (solution), Visual Studion versio (Visual Studio Version), MSBuild-muuttujat (MSBuild Arguments), alusta (platform) ja konfiguraatio (configuration), vaikkakin ainoat pakolliset tiedot ovat solution-tiedosto ja Visual Studio versio, mikä on oletuksena uusin versio (latest). Luvun 3.1 koodista todettiin, että alusta ($(buildPlatform)) ja konfiguraatio ($(buildConfiguration)) määritettiin erillisissä muuttujissa, jotta niitä voidaan kutsua myöhemmin YAML-dokumentissa (kuvio 13). Konfiguraatioksi oli määritetty julkaisu (release), koska CI-putken artefaktista oli tarkoitus luoda julkaisu CD-putkella.

```
- task: VSBuild@1
  inputs:
    solution: '$(solution)'
    msbuildArgs: '/p:DeployOnBuild=true'
    platform: '$(buildPlatform)'
    configuration: '$(buildConfiguration)'
  displayName: Build UI
```

Kuvio 13. Kuvakaappaus Visual Studio Build -koodista Azure DevOps -ympäristössä

Myös solution-tiedostoon viitataan muuttujan nimellä, eli sitä ei tarvitse enää erikseen kirjoittaa VSBuildin solution-kohtaan. Yllä olevassa kuviossa on hyvin yksinkertaistettu VSBuild-tehtävän määritys. Kuvion argumenttien lisäksi tarvittiin myös /p:DeployDefaultTarget, /p:WebPublishMethod ja /p:PublishUrl, ja koska kehittämistyössä oli kyseessä Web site -projekti, msbuildArgs vaati enemmän argumentteja, että käännöksen sai toimimaan. Web site -projekti tarvitsi toimiakseen lisäksi Publish Profile -määrityksen ja polun sinne. Publish Profile -määrityksen argumentit olivat /p:PublishProfileRootFolder ja /p:PublishProfile.

ArchiveFiles

ArchiveFiles-tehtävä luo lähdekoodista pakatun kansion, joka sisältää asennustiedostot, jotka julkaistaan myöhemmin artefaktina. Tiedostojen pakkaamista varten tulee määrittää juurihakemiston (rootFolderOrFile) polku. Polun nimessä käytetään ennalta määritettyä Build.BinariesDirectory-muuttujaa, johon luodaan web-sovelluksen nimellä alihakemisto. Tehtävässä määritetään myös tiedostomuoto (archiveType), joka kehittämistyössä oli zip. Tiedoston sijainnin (archiveFile) määrityksessä käytetään niin ikään ennalta määritettyä Build.ArtifactStagingDirectory-muuttujaa. Määritys kertoo, mihin tiedostot kopioidaan ennen niiden artefaktiksi pakkaamista tai muuta käyttöä.

Publish Build Artifacts

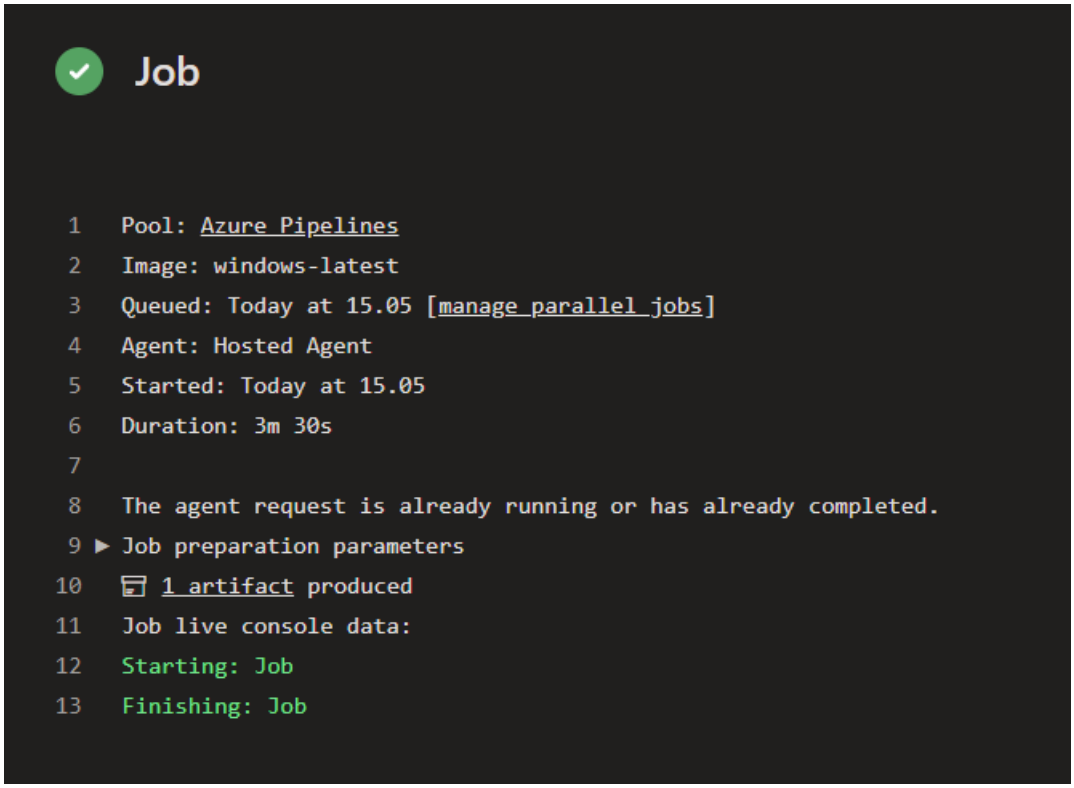
Toimeksiantajan web-sovelluksen aiemmassa CI-putken käyttöönotossa käytettiin PublishBuildArtifacts -tehtävää artefaktin luonnissa. Vaikka Microsoftin (Publish Build Artifacts task 2022) suositus on käyttää Pipeline Artifacts -tehtävää PublishBuildArtifacts -tehtävän sijaan, kehittämistyössä

päädyttiin käyttämään samaa tehtävää kuin web-sovelluksen toisessakin versiossa, jotta sovellusversioiden CI/CD-putket säilyvät yhteneväisinä. PublishBuildArtifacts -tehtävässä määritetään, miten ja mihin ArchiveFiles-tehtävässä pakattu kansio julkaistaan.

Kehittämistyössä artefakti julkaistiin ennalta määritetyn Build.ArtifactStagingDirectory-muuttujan polkuun. Tehtävässä määritetään myös artefaktin nimi ja julkaistaanko artefakti säiliöön (container) vai annetaanko sille tiedostosijainti (filePath). Kehittämistyössä artefakti julkaistiin säiliöön, koska CI-putken ajamiseen voitiin käyttää mitä vain agenttia, eikä sillä olisi pääsyä toimeksiantajan palvelimelle, minne tiedostosijainnin olisi voinut asettaa.

Jobs

Jobs, eli “työt” valinnan alta pystyi tarkkailemaan, miten koonti eteni YAML-tiedostossa määritettyjen vaiheiden mukaisesti. Seuraavassa kuviossa (kuvio 14) on kuvattu onnistuneen koonnin vaiheet.



Kuvio 14. Kuvakaappaus onnistuneesta koonnista Azure DevOps -ympäristössä

Onnistunut koonnin suoritus näyttää statistiikkaa koonnista, kuten millaista agenttia käytettiin ja kauan CI-putken suorittamisessa kesti. Putken suoritusta voi seurata reaaliajassa, ja todentaa kuinka sovellus hakee lähdekoodin ja suorittaa YAML-tiedostossa määritetyt työvaiheet yksi kerrallaan. Kuten kuviosta voidaan todeta, onnistuneen työn alta pääsee siirtymään CI-putken luomaan artefaktiin (kuvio 14, rivi 10, 1 ”artifact produced”).

3.1.2 Jatkuvan integroinnin lisäys

Azure Pipelines -työkalun luomassa YAML-tiedoston pohjassa jatkuva integraatio on oletuksena päällä master-haaraan. Tiedoston triggers-osiossa voidaan määrittää jatkuvan integraation käytöstä muissa haaroissa. Luvun 3.1 koodiesimerkin mukaan trigger-osiossa on oletuksena ”master”, eli kun muutos lisätään master-haaraan, putki suoritetaan automaattisesti, jolloin muutokset integroituvat automaattisesti lähdekoodiin.

Niin kuin useaa muutakin ominaisuutta, myös trigger-osiota pystyi hallitsemaan käyttöliittymän puolelta. Jos trigger-osiota muutetaan käyttöliittymän kautta, siellä määritetyt muutokset yliajavat, eli korvaavat mahdollisen YAML-tiedoston trigger-osion. Kuten YAML-tiedostossa, myös käyttöliittymän kautta voi määritellä, jos haluaa asettaa jatkuvan integroinnin muulle haaralle kuin master-haaralle. Kuten luvussa 3.1 todettiin, kehittämistyötä ei tehty suoraan master-haaraan, joten kehittämistyön toteutusta varten jatkuva integrointi piti ottaa manuaalisesti käyttöön haaraan toteutuksen ajaksi.

3.2 Oman agentin luonti

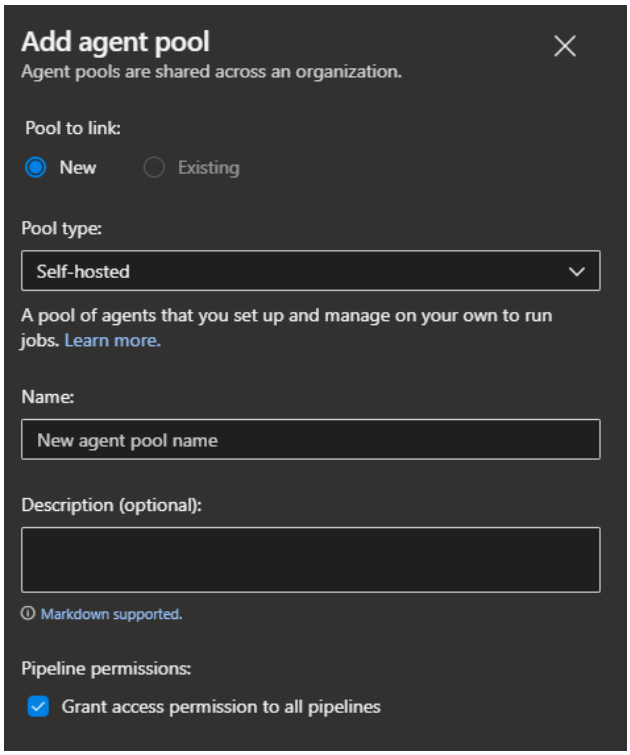
Kehittämistyön seuraava vaihe oli oman agentin luonti, joka sisälsi agentille alustan luonnin, agentin asennuksen palvelimelle ja käyttöönoton. Kuten todettu luvussa 2.3.3, oman agentin käyttö ei ole pakollista, mutta siitä on hyötyä joissain tapauksissa.

3.2.1 Alustan luonti agentille

Ensimmäinen vaihe oli luoda agenttia varten ”alusta” (agent pool). Kehittämistyössä luotu agentti oli toimeksiantajan palvelimella (self-hosted agent). Oman agentin olisi voinut määritellä valmiille default, eli oletusalustalle, mutta koska agentin ei ollut tarkoitus tulla toimeksiantajalle kuin yhden

sovelluksen käyttöön, oli parempi luoda oma alusta agentille; näin käyttöoikeuksiakaan Azure DevOps -ympäristöön ei tarvinnut määritellä niin laajasti.

Uuden alustan lisääminen kävi yksinkertaisesti Azure DevOps -ympäristön käyttöliittymän kautta. Lisäysvaiheessa alustalle määritettiin tyyppi (omalla palvelimella tai virtuaalikoneella, vai Microsoftin ylläpitämä) ja annettiin nimi sekä määritetiin oikeudet putkiin (kuvio 15).



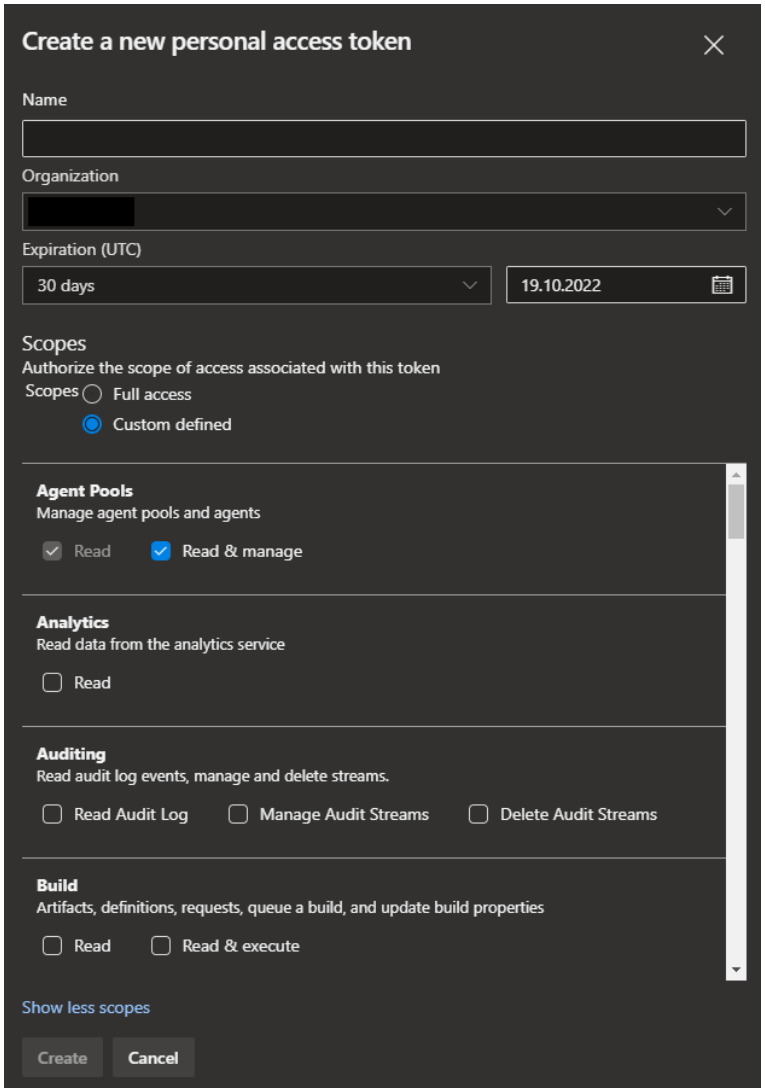
Kuvio 15. Kuvakaappaus agenttien alustan lisäyksestä Azure DevOps -ympäristössä

Alustalle pystyi antamaan luontivaiheessa määrityksen, jolla mikä tahansa putki voi käyttää altaan agentteja (grant acces permission to all pipelines -valinta) (kuvio 15). Kehittämistyössä asetusta ei laitettu päälle, vaan putken erikseen kysyessä lupaa alustan agentin käyttöön se myönnettiin.

3.2.2 Agentin asennus palvelimelle

Jotta agentin pystyi asentamaan palvelimelle, tuli ensin ladata agenttisovelluksen asennustiedosto. Agentin asennuspaketin sai ladattu Azure DevOps -ympäristön käyttöliittymän kautta, kun agentin alusta oli luotu. Agentin asennustiedoston hankkiminen oli yksinkertaisesti tiedoston lataaminen omalle koneelle, ainoa olennainen asia oli valita oikean käyttöliittymän asennuspaketti.

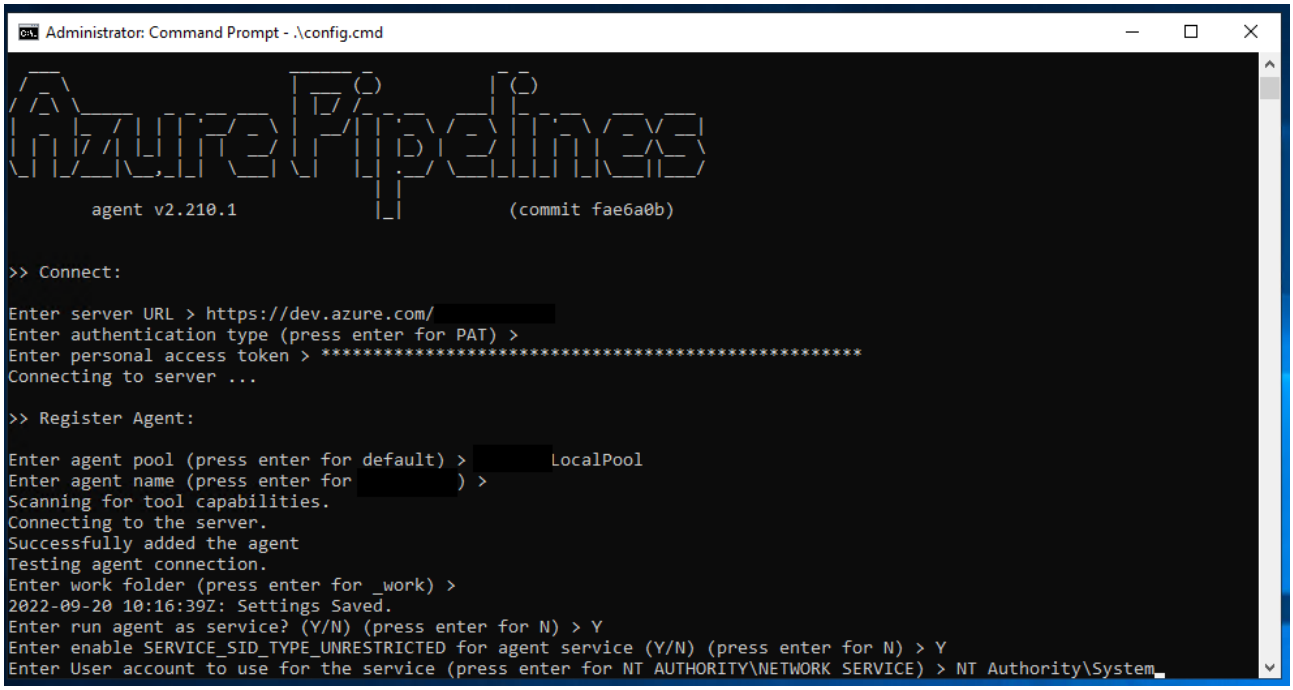
Toimeksiantajan palvelimen käyttöjärjestelmä oli Windows, joten agentin asennustiedosto valittiin sen mukaisesti Windows-käyttöjärjestelmälle sopivana. Varsinainen asennustiedosto oli pakattu kansio (zip-tiedosto), mutta ennen asennuksen toteutusta tarvittiin vielä generoitu käyttöoikeus-tunnus, Personal Access Token (PAT), agentin konfigurointia varten (kuvio 16).



Kuvio 16. Kuvakaappaus personal access token generoinnista Azure DevOps -ympäristössä

Agentin käyttötarkoitus vaikutti oikeuksien laajuuteen (scopes). Kehittämistyön käyttöoikeustun-nusta luodessa olennaisinta oli valita laajuudeksi agent pools -valinnan hallintaoikeus, sekä käyt-töönoton (deployment) hallintaoikeus. Muita oikeuksia käyttöoikeustunnukseen ei tarvinnut valita agentin asennusta varten.

Kun agentin asennuspaketti oli ladattu ja käyttöoikeustunnus generoitu, siirryttiin toteutuksessa palvelimen puolelle. Asennuspaketti siirrettiin ja purettiin palvelimelle. Varsinainen tiedosto, joka käynnisti agentin asennuksen oli config.cmd. Tiedosto ajettiin Windowsin komentokehotteen avulla, jolloin ohjelma käynnisti ohjatun agentin asennuksen (kuvio 17).

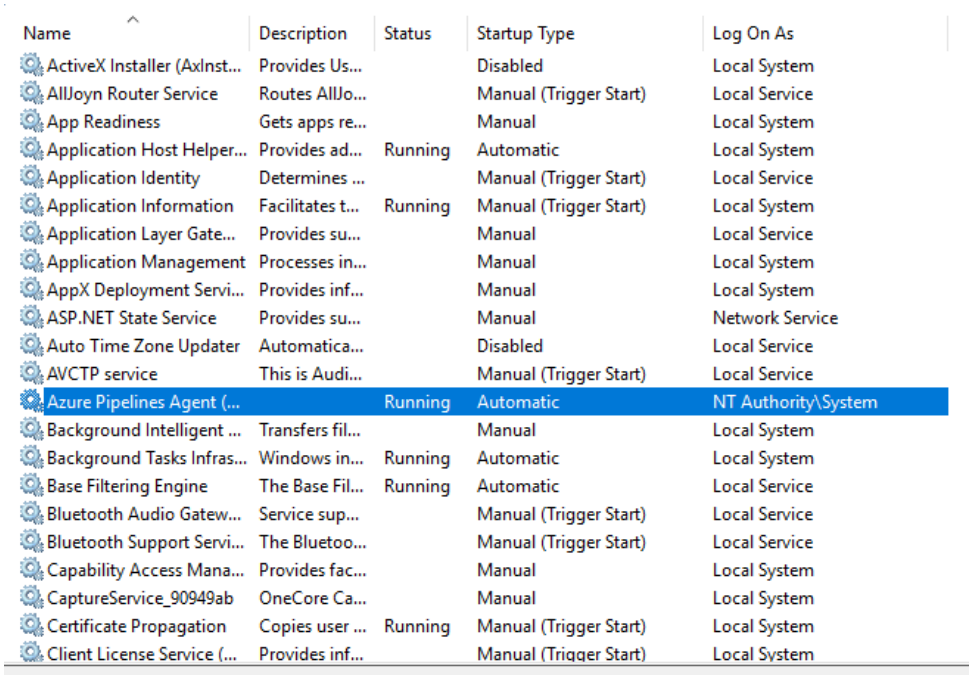


Kuvio 17. Kuvakaappaus agentin rekisteröinnistä palvelimella

Asennusohjelma oli todella pitkälti ohjattu. Ensimmäiseksi luotiin yhteys (connect) Azure DevOps -ympäristön palvelimelle palvelimen osoitteen ja aiemmin generoidun käyttöoikeustunnuksen (PAT) avulla. Kun yhteys oli onnistuneesti luotu palvelimelle, tuli agentti rekisteröidä luodulle agentti alustalle (agent pool) ja agentille annettiin nimi (kehittämistyössä nimeksi tuli palvelinkoneen nimi). Nimen annon jälkeen ohjelma yhdisti palvelimelle ja lisäsi agentin alustalle, jonka jälkeen konfiguroitiin asetuksia.

Ohjelma tarvitsi tiedon työkansiosta (work folder), joka kehittämistyössä oli asennusohjelman tarjoama oletuskansio. Seuraavaksi ohjelma halusi tiedon, käytetäänkö agenttia palveluna (as service). Kehittämistyössä agenttia käytettiin palveluna, kuten Microsoft (2022) suositteli Self-hosted Windows agents -ohjeistuksessaan (Self-hosted Windows agents 2022). Lopuksi piti määritellä

millä käyttäjätunnuksella palvelua voidaan käyttää. Koska agenttia käytettiin palveluna, onnistuneen agentin rekisteröinnin jälkeen käytiin palvelimen palvelut-asetuksista (services) vielä muuttamassa agentin asetuksia niin, että se käynnistyi automaattisesti (kuvio 18).



Name	Description	Status	Startup Type	Log On As
ActiveX Installer (AxInst...	Provides Us...		Disabled	Local System
AllJoyn Router Service	Routes AllJo...		Manual (Trigger Start)	Local Service
App Readiness	Gets apps re...		Manual	Local System
Application Host Helper...	Provides ad...	Running	Automatic	Local System
Application Identity	Determines ...		Manual (Trigger Start)	Local Service
Application Information	Facilitates t...	Running	Manual (Trigger Start)	Local System
Application Layer Gate...	Provides su...		Manual	Local Service
Application Management	Processes in...		Manual	Local System
AppX Deployment Servi...	Provides inf...		Manual	Local System
ASP.NET State Service	Provides su...		Manual	Network Service
Auto Time Zone Updater	Automatica...		Disabled	Local Service
AVCTP service	This is Audi...		Manual (Trigger Start)	Local Service
Azure Pipelines Agent (...		Running	Automatic	NT Authority\System
Background Intelligent ...	Transfers fil...		Manual	Local System
Background Tasks Infras...	Windows in...	Running	Automatic	Local System
Base Filtering Engine	The Base Fil...	Running	Automatic	Local Service
Bluetooth Audio Gatew...	Service sup...		Manual (Trigger Start)	Local Service
Bluetooth Support Servi...	The Bluetoo...		Manual (Trigger Start)	Local Service
Capability Access Mana...	Provides fac...		Manual	Local System
CaptureService_90949ab	OneCore Ca...		Manual	Local System
Certificate Propagation	Copies user ...	Running	Manual (Trigger Start)	Local System
Client License Service (...	Provides inf...		Manual (Trigger Start)	Local System

Kuvio 18. Kuvakaappaus palveluista palvelimella

Konfiguroinnin jälkeen aiemmin luodusta agentti alustasta pystyi käydä todentamassa, että sinne oli ilmestynyt agentti juuri annetuilla tiedoilla ja että agentin tila oli käynnissä (online). Agentin käynnissä olo -tilan lisäksi agentista näkyi, milloin sillä oli viimeksi ajettu putki (last run), oliko agentilla töitä kesken (current status) ja agenttisovelluksen versio (agent version). Agentin pystyi myös poistamaan käytöstä (enabled).

3.3 CD-putken luonti (Release Pipeline)

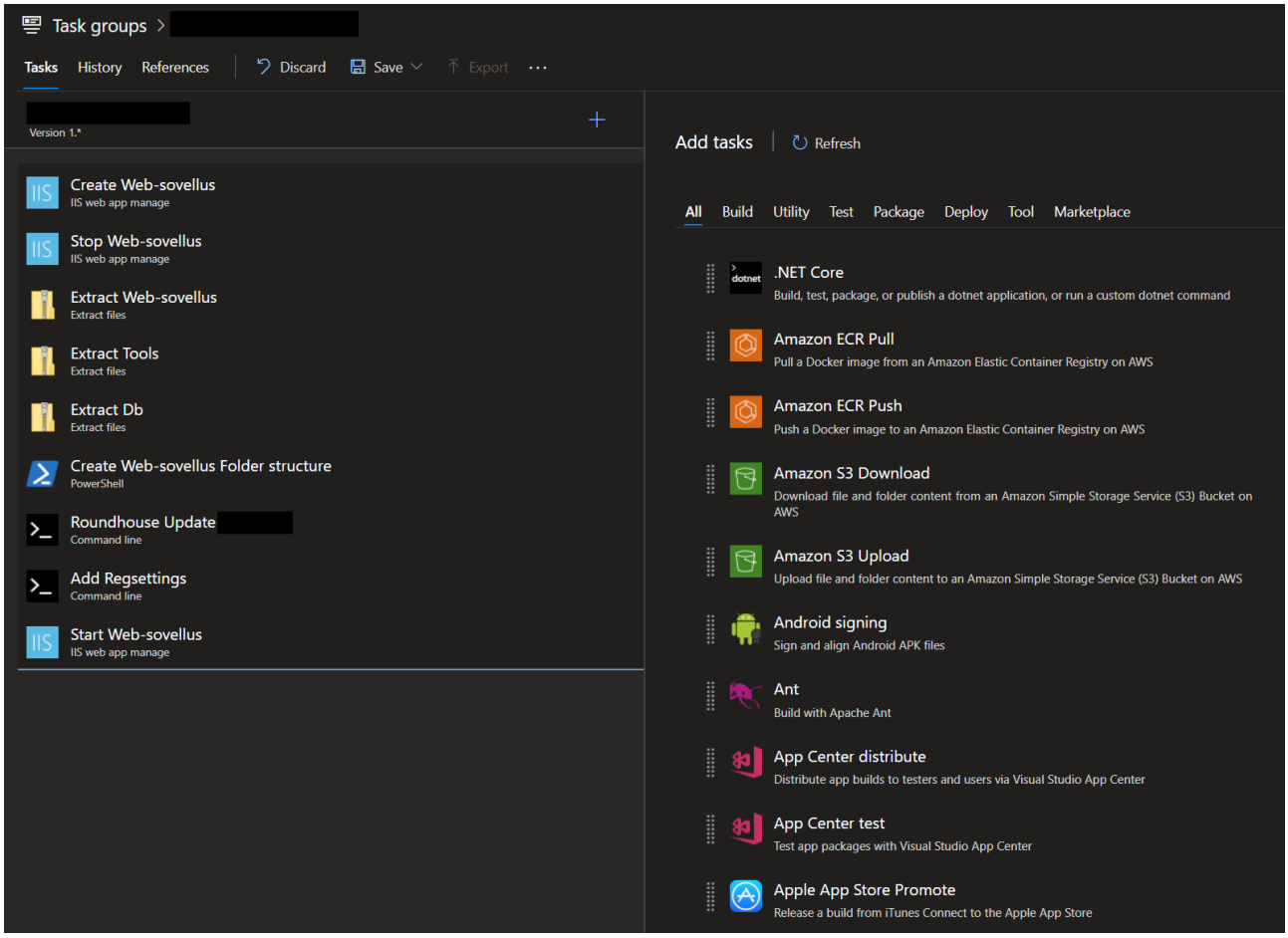
Kuten luvuissa 2.2 ja 2.2.1 kuvattiin data putken toimintaperiaatetta sekä CI-menetelmän asemaa CD-menetelmien perustana, CD-menetelmät jatkavat siitä mihin CI-menetelmä loppuu, eli kun CI-putkesta tulee artefakti ulos, sitä hyödynnetään CD-putken määrityksessä. Kun lukujen 3.1 ja 3.2 vaiheet oli toteutettu ja CI-putki suoritettu, luotu artefakti voitiin purkaa ja asentaa agentin avulla palvelimelle. Asennus tapahtui CD-putken (release pipeline) avulla, joka toteuttaa jatkuvan toimi-

tuksen ja jatkuvan käyttöönoton menetelmät. Kuten muutkin DevOpsin CI/CD-menetelmien käyttöönoton vaiheet, myös CD-putken luonti oli kohtalaisen yksinkertaista graafisen käyttöliittymän avulla.

CD-putken luontia varten tarvittiin aiemmin koottu artefakti, mutta kuten todettu oma agentti ei ollut pakollinen. CD-putkea luodessa artefaktin valinnan lisäksi pakollinen valinta on vaiheiden (stage) lisäys, joiden alle lisättiin samankaltaisesti tehtäviä kuin luvussa 3.1.1 CI-putkeen. Kehittämistyössä CD-putkeen riitti yksi vaihe, jonka alle tehtävät lisättiin.

3.3.1 Tehtävien lisäys (Task Group)

Kun CD-putki oli luotu, sille voitiin lisätä tehtävät, mitkä purkavat CI-putken tuloksena syntyneen artefaktin palvelimelle ja käyttöönottavat sen niin, että lopputulos on toimiva web-sovellus, jonka saa avattua selaimen kautta. CD-putken tehtävät lisättiin graafisen käyttöliittymän kautta, ei YAML-tiedostolla kuten CI-putkessa. Kehittämistyön toteutuksen aikana todettiin, että suoritettaville tehtäville oli kätevämpää luoda erillinen tehtäväjoukko (task group), kuin lisätä jokainen tehtävä suoraan putkeen suoritettavaksi (kuvio 19). Tehtäväjoukon luontia puolsi se, että tehtäväjoukkoa hyödynnettiin myös toimeksiantajan web-sovelluksen toisen version CD-putkessa, lisäksi omien muuttujien määrittely oli yksinkertaisempaa.



Kuvio 19. Kuvakaappaus tehtävien lisäämisestä CD-putkeen Azure DevOps -ympäristössä

Tehtäväjoukon käytön hyvä puoli kehittämistyön tapauksessa oli myös, että jos toimeksiantaja ottaisi joskus CD-putken käyttöön web-sovelluksen muuhun versioon, samaa tehtäväjoukkoa voisi mahdollisesti käyttää sielläkin, eikä samoja tehtäviä tarvitsisi määritellä uudestaan. Yllä olevasta kuviosta voidaan tarkastella, millaisia tehtäviä tarvitaan, että artefaktista saadaan web-sovellus käyttöön palvelimelle.

Onnistunutta julkaisua varten määritettiin seuraavat tehtävät;

IIS web app manage

IIS web app manage -tehtävät luovat, pysäyttävät ja käynnistävät IIS-palvelun. Jotta web-sovellus voidaan avata selaimen kautta palvelimelta, tulee siitä luoda IIS verkkosivu. IIS verkkosivun luontia varten määritetään tehtävä (esimerkiksi luominen, pysäytys, käynnistys), määritetään polku, missä web-sovelluksen tiedostot sijaitsevat palvelimella, annetaan nimi sovellukselle ja IIS verkkosivulle.

Luomisen jälkeen web-sovellus pysäytetään siksi aikaa, että tiedostot puretaan ja asennetaan. Kun web-sovelluksen asennus on valmis, palvelu käynnistetään, joka näkyy käyttäjälle niin, että web-sovelluksen saa auki selaimen kautta.

Extract files

Extract files -tehtävät purkavat CI-putkella luodun artefaktin sisältämät kansiot. Tehtävän asetuksissa määritetään, mihin hakemistoon tiedostot puretaan. Tiedostoja purkaessa on hyvä ottaa huomioon, haluaako purun yliajavan hakemistossa mahdollisesti jo olevat tiedostot, vai haluaako aiemmat tiedostot säilyttää ja vain lisätä artefaktista puretut tiedostot. Kehittämistyön artefakti sisältää pääkansion, jonka alla on seuraavat alikansiot; käyttöliittymän koodi, tietokannan luonti tiedostot ja Roundhouse-työkalu sekä ympäristön luontiin tarvittavat skriptit sisältävä kansio.

PowerShell

PowerShell tehtävällä voidaan ajaa määritettyjä PowerShell skriptejä. Kehittämistyössä tehtävissä käytetään skriptiä, joka luo web-sovellukseen halutun kansiorakenteen. Tehtävä hakee artefaktista puretusta skripti kansiosta PowerShell skriptin ja ajaa sen, luoden palvelimelle halutun kansiorakenteen.

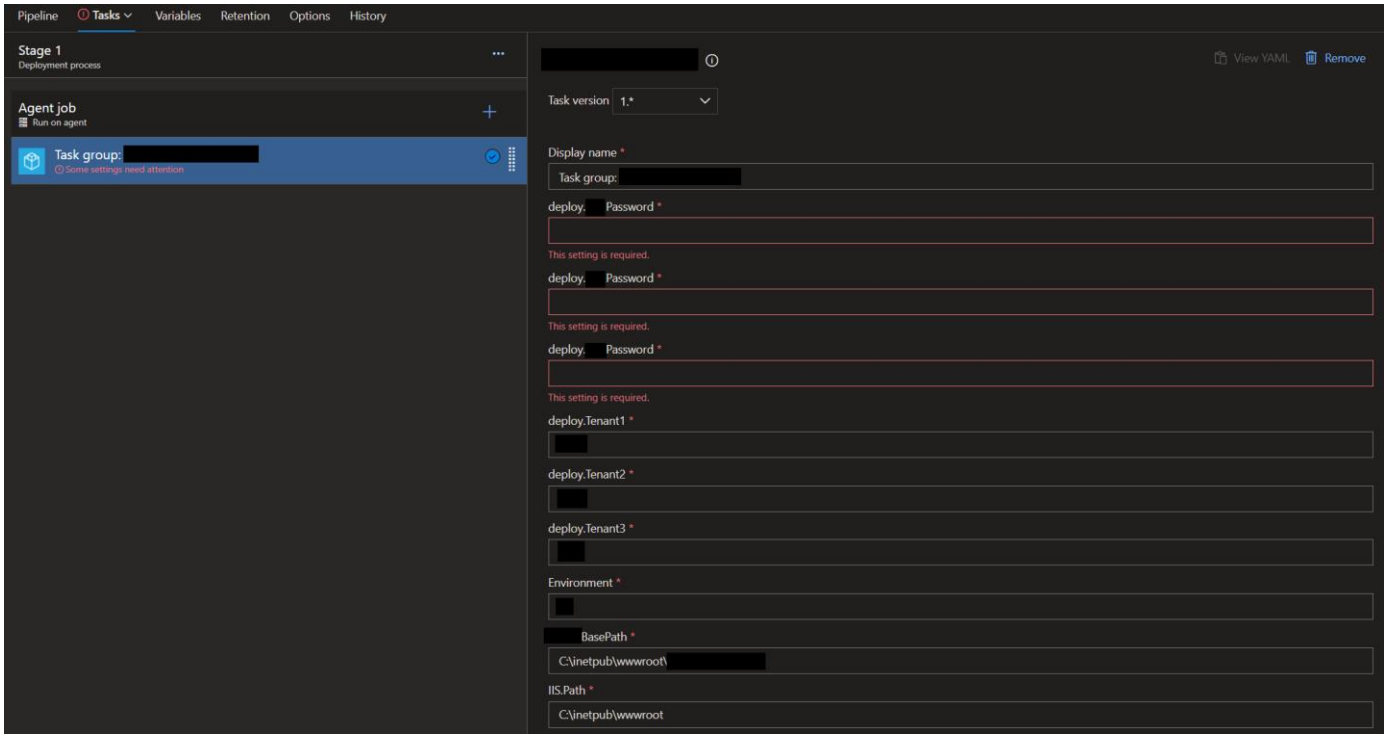
Command line

Komentokehote (command line) tehtävä toimii samantapaisesti kuin PowerShell tehtävä, määritetyt skriptit vain ajetaan komentokehoitteella. Kehittämistyössä komentokehote skriptit hakevat artefaktista puretun tietokanta kansion ja skripti kansion. Komentokehote ajaa tilanteesta riippuen tietokannan luovan tai päivittävän Roundhouse-sovelluksen ja lisää rekisteriasetukset palvelimen rekisterieditoriin.

File Transform

Aiemmassa kuviossa (kuvio 19) tehtäväryhmässä ei ole kuvattuna File Transform -tehtävää, mutta kehittämistyön kannalta se oli olennainen tehtävä. File Transform -tehtävällä voi muokata esimerkiksi konfiguraatiotiedostoja; tehtävän ja muuttujien avulla määritellään, mikä osa konfiguraatiotiedostossa korvataan toisella osalla, esimerkiksi hakemistopolku. Tehtävä helpottaa jatkossa CI/CD-putkien ajamista esimerkiksi asiakkaan palvelimelle.

Kun tehtävät oli määritetty tehtäväryhmään, voitiin koko ryhmä lisätä yhtenä suoritettavan tehtävänä CD-putken vaiheiden alle. Tehtävien määrityksissä voitiin käyttää omia muuttujia. Halutessaan muuttujan pystyi kirjoittamaan suoraan tehtävän määrityksiin, jolloin tehtäväryhmä loi automaattisesti muuttujan, jolle pystyi asetuksissa antamaan halutun arvon (kuvio 20).

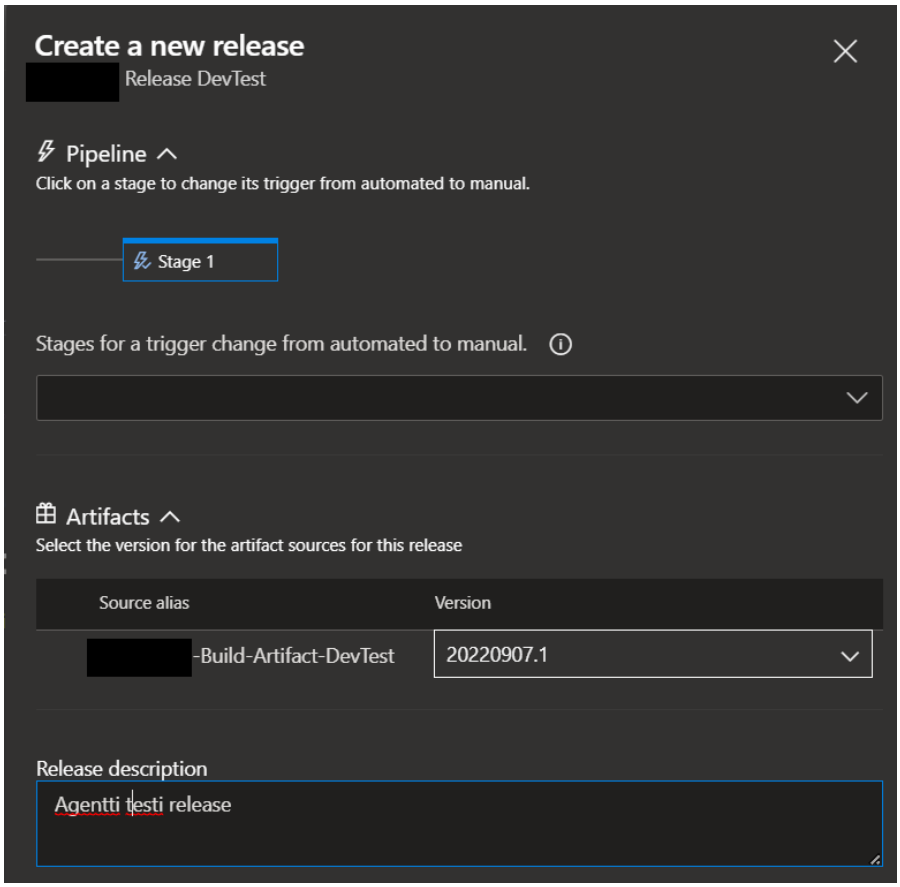


Kuvio 20. Kuvakaappaus muuttujien lisäämisestä CD-putkeen Azure DevOps -ympäristössä

Muuttujalle pystyi antamaan oletusarvon tehtäväryhmän asetuksissa, tai määrittämään CD-putkikohtaiset arvot CD-putken asetusten alla (kuvio 20). Muuttujia käytettiin etenkin tenant asetusten, kuten tenantin nimen ja salasanan, määrittämiseen. Vaikka kehittämistyön kohteena oli vain yksi versio web-sovelluksesta, muuttujien arvot määritettiin CD-putkikohtaisesti, jolloin tehtäväryhmän alla olevat muuttujien oletusarvot jätettiin tyhjiksi. Tämä mahdollisti, että jos samaa CD-putkea käytetään joskus web-sovelluksen toisessa versiossa, oletusarvoista ei tarvitse välittää. Tämänkin takia edellä mainittu File Transform -tehtävä oli tärkeä olla CD-putkessa; konfiguraatiota voidaan muuttaa myös web-sovellusversiosta riippuen.

3.3.2 Julkaisun luonti (Create release)

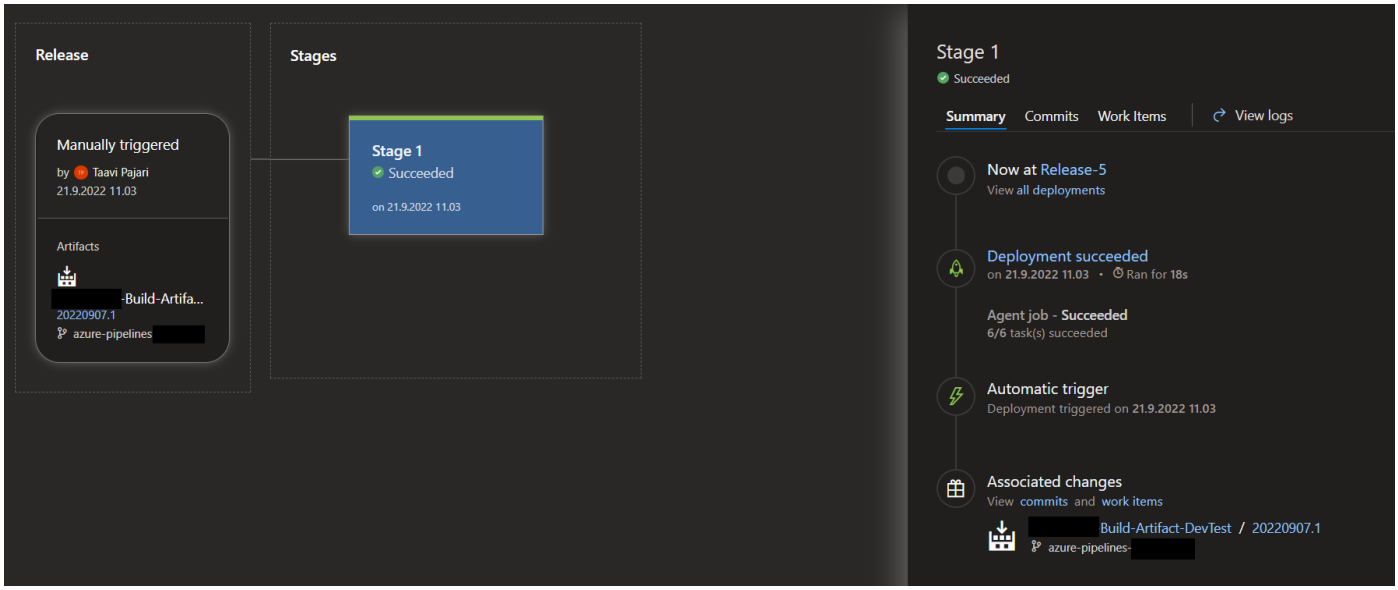
Kun putken toiminta oli määritetty tehtävien avulla, voitiin luoda julkaisu (release). Julkaisu valitsee automaattisesti putken kaikki vaiheet (stage), tai jos putki sisältää useamman vaiheen, voi halutessaan valita vain osan vaiheista julkaisuun (kuvio 21).



Kuvio 21. Kuvakaappaus uuden releasen luomisesta Azure DevOps -ympäristössä

Julkaisun luonnissa määritetään, mitä versiota CI-putken artefaktista käytetään (artifacts). Käyttöliittymä tarjoaa oletuksena artefaktin uusinta versiota. Jokainen CI-putken ajo luo artefaktista uuden version, jonka huomaa versionumeron päivittymisestä (kuvio 21). Kun julkaisu on luotu, CD-putken edistymisen tilan voi halutessaan tarkistaa agentin asetuksista; agentin tilassa näkyy, että agentti on suorittamassa CD-putken vaiheita. Kehittämistyön kannalta agentin kautta ei ollut tarvetta tarkistaa putken edistymistä, sillä kuten mainittu luvussa 3.3, kehittämistyön CD-putkessa oli vain yksi vaihe.

Kehittämistyön CD-putken tilan seuraamisen kannalta olennaisempaa informaatiota pystyi tarkastelemaan CD-putken tietojen alta. CD-putken edetessä vaiheen alta näkyy, missä tehtävässä putken suoritus on menossa. CD-putken suorituksen jälkeen näkyy yhteenvetona statistiikkaa putken suorituksesta, muun muassa suorituksen aloitus päivämäärä ja kellonaika, kuinka kauan suoritus kesti, mitkä tehtävät onnistuivat ja julkaisun päivämäärä ja kellonaika (kuvio 22).

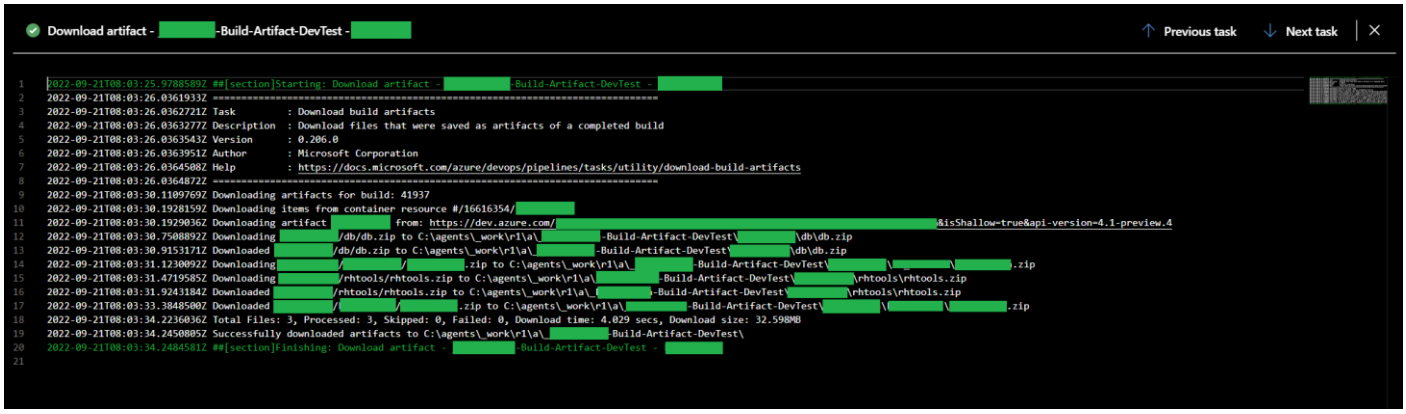


Kuvio 22. Kuvakaappaus onnistuneesta julkaisu Azure DevOps -ympäristössä

Yllä olevassa kuviossa näkyy onnistuneen julkaisun yhteenvetoa. Artefaktin tiedoista näkyy myös, mitä versionhallintahaaraa CI-putki on käyttänyt. Yhteenvedosta voidaan myös todeta, että toistaiseksi päätös julkaisun luonnista on tehty manuaalisesti (manually triggered), joten kyseessä on jatkuvan toimituksen CD-putki niin kauan, kunnes julkaisut luodaan automaattisesti (kuvio 22).

Jobs

Kuten CI-putkessa luvussa 3.1.1, myös CD-putkessa suoritettujen tehtävien lokeja voi tarkastella jälkikäteen (kuvio 23).



Kuvio 23. Kuvakaappaus onnistuneesta työtehtävästä Azure DevOps -ympäristössä

Jos agentti kohtaa tehtävän suorituksessa virheen, suoritus keskeytyy ja mahdolliset jäljellä olevat tehtävät ohitetaan. Esimerkkinä on onnistuneen artefaktin tiedostojen purkutehtävän loki (kuvio 23). Kuviosta voidaan todeta, että tehtävän suorituksen alussa näky informaatiota tehtävästä itsestään, jonka jälkeen näkyy varsinainen tehtävän suoritus; tehtävä on ensin ladannut artefaktin, jonka jälkeen pakatut zip-tiedostot on ladattu väliaikaisiin hakemistosijainteihin. Voidaan myös todeta, että eri hakemistopolkujen määrityksissä voi päästä helpommalla, jos käytätä Azure DevOps -työkalun valmiita sijainteja silloin kun niitä pystyy hyödyntämään, etenkin väliaikaisissa tiedostojen säilytyksissä.

3.3.3 Jatkuvan julkaisun käyttöönotto

Kuten luvussa 2.2.3 Montonen totesi, jatkuva toimitus valmistelee jatkuvan integraation vaiheen koodimuutokset, eli CI-putken tuottaman artefaktin, jatkuvaa tuotantoa varten. Kehittämistyön tarkoitus oli käyttöönottaa nimenomaan jatkuvan julkaisun menetelmä, missä muutokset julkaistaan automaattisesti (toimeksiantajan palvelimelle). Teoriassa, jos jatkuvaa julkaisua ei kytkisi CD-putken asetuksista päälle, kyseessä olisi jatkuvan toimituksen CD-putki, koska kuten todettu luvussa 2.2.3, CD-menetelmien ainoa ero on automaation puuttuminen jatkuvassa toimituksessa.

Kuten CI-putkessa, myös CD-putkessa automaation saa käyttöliittymän kautta kytkettyä päälle. Vaikka kehittämistyö tehtiin omaan haaraan, eikä master-haaraan, jatkuvan julkaisun olisi voinut kytkeä päälle. Kehittämistyössä jatkuva julkaisu jäi kuitenkin kytkemättä päälle, koska putkien luomisen jälkeen web-sovellukseen haluttiin lisätä vielä tehtäviä ja eräajot.

3.4 Haasteet

Koska CI/CD-menetelmät olivat jo käytössä toimeksiantajan web-sovelluksen yhdessä versiossa, toimeksiantajalla oli tiedossa aiemmin vastaan tulleita haasteita, jotka tuli ottaa huomioon kehittämistyön toteutuksessa. Kehittämistyön toteutuksen aikana ei tullut vastaan suurempia uusia haasteita, pieniä hidasteita aiheutti lähinnä oikeuksia lisäyksen odottaminen. Vaikka isoja haasteita ei ollut, joitain selkeitä haasteita kuitenkin tuli vastaan;

Projektin käännös

Tämä haaste toimeksiantajalla oli tiedossa ennen kehittämistyön aloitusta. Web-sovellus on toteutettu ASP.NET Web site -projektina, kun toimeksiantajan aiempi CI/CD-menetelmien käyttöönotto oli tehty Web application -projektina, joten versiot toimivat hieman eri tavoin. Haaste tuli vastaan CI-putken luonnissa; kuten luvuissa 3.1. ja 3.1.1 mainittiin, YAML-tiedosto tarjosi oletuksena solution-tiedostoa koontiin ja kehittämistyön web-sovelluksen tapauksessa solution-tiedostoa ei oltu viety versionhallintaan. Kehittämistyössä päädyttiin hyödyntämään sen sijaan website.publishproj, eli Publish Profile -tiedostoa muuttamalla YAML-tiedoston tarjoama oletusasetus luvun 3.1.1 mukaisesti solution-tiedostosta website.publishproj-tiedostoksi ja lisäämällä MSArgs-muuttujiin julkaisuun liittyviä määrityksiä, jotta koonnin sai toimimaan.

Publish profile CI-putkessa

Luvun 3.1.1 mukaisesti YAML-tiedostoon piti lisätä julkaisuun liittyviä määrityksiä liittyen Publish Profile-määreeseen. Koska toimeksiantajan aiempi Web application -projekti ei kyseistä määritystä tarvinnut, piti kehittämistyön toteutuksen aikana selvittää, miksi CI-putki ei meinannut mennä läpi. Koska kehittämistyön koonti putkessa käytettiin Publish Profile -tiedostoa, Web site -, eli ASP.NET-projekti tarvitsi toimiakseen Publish-ominaisuuden, tarkemmin Publish Profile-määrityksen, jonka määritykset annettiin VSBuild-tehtävässä. Publish Profile on pubxml-päätteinen tiedosto (Publish Procjet File), joka olisi pitänyt luoda web-sovelluksen projektiin, mutta koska profiili oli jo luotu, riitti sen määrittely VSBuild-tehtävään ja sen hakemistosijainnin lisäys argumentteihin.

Tietokannan päivitys Roundhouse-työkalulla

Web-sovellus oli ennestään asennettuna toimeksiantajan palvelimelle, ja se sisälsi vuosien aikana muodostuneen tietokannan. CI/CD-putket toivat kuitenkin web-sovellukseen Roundhouse-työkalun, ja koska palvelimella olevaa tietokantaa ei haluttu yliajaa, piti työkalun antaa ymmärtää, että se olisi ajan tasalla palvelimella. Tämän sai toteutettua generoimalla skriptin kehittäjän koneelta sovelluksen tietokannasta, johon Roundhouse oli ajettu, ja ajamalla se toimeksiantajan palvelimen tietokantaan, jolloin CD-putken ajamat Roundhouse skriptit olettivat sen olevan ajan tasalla palvelimella.

Sovelluksen konfigurointi

Sovelluksessa oli hakemistopolkuja, esimerkiksi raporttien loki kansio, jonka sijainnissa oli eroja riippuen oliko sovellus asennettu kehittäjän omalle koneelle vai toimeksiantajan palvelimelle; kehittäjän koneella sovelluksen tiedostot olivat C-asemalla, palvelimella D-asemalla. File Transform -tehtävällä hakemistopolkuja sai muutettua asennuksen ympäristön mukaan. Haasteeksi tuli konfigurointi järkevällä tavalla; kaikkia konfigurointeja ei voitu muuttaa File Transform -tehtävällä, koska vaikka osa hakemistopoluista oli samoja eri konfiguraatiotiedostoissa lokaalissa asennuksessa, niitä ei haluttu muuttaa samaan hakemistosijaintiin palvelinasennuksessa. File Transform -tehtävän rinnalla käytettiin skriptejä, jotka tekivät lähinnä yksittäisiä konfiguraatiomuutoksia ja huomioon tuli ottaa, etteivät skriptit olleet keskenään ristissä.

Raportointi

Web-sovelluksessa oli käytössä SQL Server Reporting Services (SSRS) raportointiominaisuus. Raportointiominaisuus saattaa aiheuttaa virheen CI-putken suorittamisessa, ellei käytössä ole vähintään Microsoft Reporting Services Projects for Visual Studio (1.18) -versiota ominaisuudesta. Kehittämistyössä tätä haastetta ei kuitenkaan tullut vastaan.

4 Kehittämistyön tulokset

Kehittämistyön tuloksia teoriaosuuden osalta on vaikea mitata, mitkä asiat itse kokee tärkeäksi teorian kannalta DevOpsin ja CI/CD-menetelmien kanssa, riippuu osittain kehittämistyön tekijästä. Kehittämistyön teoriaosuudessa päädyttiin määrittämään DevOps-käsite, hieman sen historiaa ja CI/CD-menetelmät. Muuta tarpeelliseksi katsottua teoriaa katsottiin olevan Azure DevOps -ympäristön tärkeimmät ominaisuudet, versionhallinta ja yleisesti YAML-kieli, koska ilman versionhallintaa ja YAML-kieltä CI/CD-putkien käyttöönotto ei onnistu niin helposti.

Kehittämistyön toteutuksessa onnistuttiin luomaan ja käyttöönottamaan automatisoitu CI-putki, jonka todisteena on luotu artefakti. Myös CD-putken luonti ja suoritus onnistui, josta taas on todisteena toimiva web-sovellus palvelimella. CD-putkesta jäi puuttumaan jatkuva julkaisu, eikä joitain DevOps-menetelmille olennaisia ominaisuuksia, kuten testausautomaatiota, kehittämistyössä otettu käyttöön lainkaan. Automatisoidun testauksen käyttöönottomahdollisuus oli kuitenkin toimeksiantajan jatkokehitys tavoite, ja senkin tavoitteen kehittämistyö täytti.

Kehittämistyössä onnistuttiin asentamaan agentti toimeksiantajan palvelimelle ja suorittamaan CD-putki luodulla agentilla. Vaikka omaa agenttia voisi käyttää sekä CI-, että CD-putken suorittamisessa, kehittämistyössä omaa agenttia käytettiin vain CD-putken suorittamisessa; oman agentin käyttöä CI-putkella varten toimeksiantajan palvelimelle olisi pitänyt asentaa CI-putken tehtävissä agentin käyttämiä sovelluksia, kuten Visual Studio.

Toimeksiantajalle voidaan katsoa olevan hyötyä kehittämistyöstä. Oma agentti lyhentää putken suoritusaikaa, tämän huomasi vertaamalla CI-putken ja CD-putken suoritusaikoja; CI-putkea suorittaessa jonotusaika ennen putken suorituksen alkua oli toisinaan pitkä, kun taas CD-putki, joka käyttää palvelimella olevaa omaa agenttia, aloitti putken suorituksen heti. Toimeksiantajan web-sovellusta kehitetään jatkuvasti, joten ketteryyden ja automaation lisääminen kehitystyöhön CI/CD-menetelmien keinoin helpottaa kehitystyötä jatkossa. Lisäksi mahdollisuus käyttää artefaktia asiakasympäristöasennuksissa on mainittava hyöty.

Tuloksena todettiin myös, että varsinaisesti web-sovelluksen ikä tai vanhempi ohjelmointikieli, kehittämistyön tapauksessa enimmäkseen käytetty VB.NET, ei vaikuta juurikaan DevOpsin CI/CD-

menetelmien käyttöönottoon tai sen haastavuuteen. Enemmän käyttöönottoon vaikuttaa sovellukseen kuuluvat, mahdollisesti käytöstä poistetut, ominaisuudet tai alisovellukset, koska niiden tila pitää selvittää toteutuksen aikana, jos niitä ei ole tarkastettu ennen käyttöönoton aloitusta.

Jatkokehitys

Tärkein jatkokehitys koskee jatkuvan julkaisun käyttöönottoa CD-putkeen, koska sen käyttöönotto oli kehittämistyön tavoite, jota tukee luvun 2.2.3 toteamus, että jatkuvan julkaisun voidaan katsoa olevan ohjelmistokehityksen päätavoite. Jatkuva julkaisu olisi teoriassa ollut jo mahdollista ottaa käyttöön, vaikka web-sovellukseen lisättäisiinkin ominaisuuksia, koska CD-putken toiminta oli jo testattu toimivaksi. Todennäköisesti jatkuvan julkaisun menetelmän käyttöönotto toteutuu vielä syksyn 2022 aikana.

Toinen olennainen jatkokehitys on automatisoidun testauksen käyttöönotto CI-putkeen, koska luvun 2.2.2 mukaisesti automatisoitu testaus on olennainen osa, ja hyöty, DevOps-menetelmissä, erityisesti jatkuvassa integraatiossa. Testit tulisi kirjoittaa projektiin, jonka jälkeen testausautomaatiota voisi yksinkertaisimmillaan liittää CI-putken toimintaan määrittämällä testi tehtäviä YAML-tiedostoon. Kolmas selkeä jatkokehitys on CI-putken YAML-tiedoston muokkaus niin, että PublishBuildArtifacts-tehtävä vaihdetaan Pipeline Artifacts -tehtävään Microsoftin suosituksen mukaisesti.

Neljäs olennainen jatkokehitys on web-sovelluskokonaisuuteen liittyvien eräajojen lisäys sekä CI-putkeen koottavaksi, että CD-putkeen julkaisuun. Sovelluksen eräajojen koonti suoritetaan yhdellä projektilla, johon on lisätty kaikki web-sovelluksen eräajot. Kaikki ajot sisältävää projektia ei oltu ennen kehittämistyön toteutusta päivitetty pitkään aikaan eikä sen käännös CI-putkessa onnistunut, joten jatkokehityksenä projekti pitäisi luoda kokonaan uudestaan ja testata ja kääntää edelleen käytössä olevat eräajot, jotta ne voidaan lisätä projektiin.

Kehittämistyön toteutuksessa on otettu huomioon CI/CD-putkien muunneltavuus ja sopivuus web-sovelluksen muihin versioihin (esimerkiksi muuttujissa, ja välttämällä web-sovelluksen nimen kovakoodaamista tehtäviin), jolloin jatkokehityksenä voi olla myös CI/CD-menetelmien käyttöönotto muihin sovellusversioihin.

5 Pohdinta

DevOpsin CI/CD-menetelmien käyttöönottoa suunnitellessa on hyvä sisäistää, että ennen käyttöönoton toteutusta tulee omaksua DevOps-filosofia ja -kulttuuri, minkä lisäksi on hyvä varmistaa luvun 2.2 mukaisesti, että kaikilla tiimissä on yhteneväinen näkemys CI/CD-menetelmien tarkoituksesta; kehittämistyön kannalta C/CD-menetelmien käsitteiden avaamisen tärkeys korostui kehittämistyön toteutuksen edetessä, kun selveni, että toimeksiantaja halusi CD-menetelmistä myös jatkuvan julkaisun käyttöön.

Toinen huomioon otettava asia ennen CI/CD-menetelmien käyttöönottoa, etenkin jos kyseessä on jo olemassa oleva sovellus, on huolehtia, että sovelluksen kaikki osa-alueet sekä mahdolliset alisovellukset ja projektit toimivat käyttöönoton alkaessa. Kuten edellisessä luvussa mainittu, kehittämistyön toteutusvaiheessa tuli vastaan tilanne, että web-sovelluksen eräajot kokoava projekti oli ollut pitkään päivittämättä ja näin ollen CI-putken kokoaminen ei mennyt läpi, koska eräajojen käännös ei onnistunut. Kaikki eräajot kokoava projekti jouduttiin jättämään kehittämistyön toteutuksesta pois aikataulullisista syistä. Kolmas etukäteen huomioon otettava asia on käyttöoikeudet, kehittämistyön aikana tarvittavien oikeuksien lisäämiset kävivät nopeasti, mutta jossain muualla tilanne voi olla toinen ja silloin lisäyksien odottaminen voi viedä aikaa toteutuksen ajasta. Oman haasteen toteutuksen aikatauluun voi tuoda myös jos palvelimella olevalla web-sovelluksella on käyttöä CI/CD-menetelmien käyttöönoton aikana, silloin luonnollisesti ei voi suorittaa web-sovelluksen asennusta palvelimella olevan version päälle.

Vaikka luvun 2.2.1 mukaan CI-menetelmän käyttöönotto voi olla vaivalloista jo olemassa olevaan sovellukseen, kehittämistyön toteutuksen perusteella voidaan kuitenkin todeta sekä CI-, että CD-menetelmien käyttöönoton olevan melko vaivatonta myös jo olemassa olevaan sovellukseen. Käyttöönottoa toteuttaessa on kuitenkin hyvä ottaa huomioon ohjelmointikieli ja tekniikka; mitä pitkäikäisempi sovellus, sitä vanhempaa kieltä tai tekniikkaa voi olla sovelluksessa mukana. Lähdekoodin mukana voi olla myös jo käytöstä poistuneita ominaisuuksia, ja pitkään kehitetyssä sovelluksessa voi olla muutenkin enemmän selvitystä vaativaa koodia mukana.

Vaikka käyttöönotto oli pääosin sujuvaa, pitkäikäisessä sovelluksessa voi tulla isojakin yhteensopivuusongelmia, joten uutta sovellusta luodessa DevOps-menetelmien käyttöönottoa kannattaa harkita jo vaatimusmäärittelyvaiheessa. Kehittämistyössä ei oteta kantaa, mikä DevOps-menetelmien

työkalu tai ympäristö kannattaisi valita, mutta tulosten perusteella voidaan todeta Azure DevOps -ympäristön käyttöönoton olleen helppoa kattavan dokumentaation ja ohjeistuksen ansiosta. Azure DevOps -ympäristön hyväksi voidaan myös todeta, että se sisältää DevOps-menetelmien kannalta olennaisimmat työkalut, mikä mahdollisesti yksinkertaistaa DevOps-menetelmien käyttöä, kun ei tarvitse koota palettia yksittäisistä DevOps-työkaluista.

Vaikka ohjelmistokehitysmenetelmistä puhuttaessa asetetaan yleensä vastakkain vesiputousmalli ja ketterä kehitys, ja ketterää kehitystä pidetään enimmäkseen parempana kuin vesiputousmallia, kuten luvussa 2.1.2 kuvataan vesiputousmallin olevan tarkasti rajattu tehtävällisesti myös DevOps-kehitystyössä työvaiheet ovat tarkasti rajattuja. Tulkinnasta riippuen DevOps-menetelmien voidaan katsoa olevan looginen jatkumo vesiputousmalli-ketterä kehitys -janalle, tai täydentävän ketterää kehitystä. Lopputulemana DevOps-menetelmät ovat hyödyllisiä, mutta varmasti kuten vesiputousmalli ja ketterä kehitys tuntuvat nyt osin etäisiltä, myös DevOps saattaa joskus saada tilalleen vielä edistyneemmän version itsestään.

DevOpsin suosio ohjelmistokehityksessä on kiistaton kuten luvussa 2.1.3 todettiin, ja vaikka jotkut ehkä yrittävät kaupallistaa DevOpsin ja ketterän kehityksen menetelmiä, ketterä kehitys on pääosin vakiintunut ohjelmistokehityksen pääkäytänteeksi joka tapauksessa. DevOps-menetelmiin kuuluva, mahdollisimman useiden työvaiheiden automatisointi, hyödyttää kehitystyössä laajasti säästäen työresursseja ja mahdollistaen resurssien panostamisen muihin kehitystehtäviin. Myös virheiden määrä vähenee, kun ihminen ei suorita kaikkia työvaihetta. Myös DevOpsin perusta, eli yhteistyö tiimien välillä on tärkeää, aikoi DevOps-menetelmiä ottaa käyttöön tai ei.

Lähteet

Abildskov, J. 2013. Mitä on DevOps? Eficoden verkkosivut. Päivitetty kesäkuussa 2021. Viitattu 17.5.2022. https://www.eficode.com/fi/blog/mita-on-devops.

Analytiikka. N.d. Profit Softwaren verkkosivut. Viitattu 17.5.2022. https://profitsoftware.com/analytiikka/?lang=fi.

Azure Artifacts overview. 2022. Microsoftin dokumentaatio. Julkaistu 29.7.2022. Viitattu 6.9.2022. https://docs.microsoft.com/en-us/azure/devops/artifacts/start-using-azure-artifacts?view=azure-devops.

Blomqvist, H. 2019. AGILE – Mikä? Milloin? Miksi? Oppia.fi verkkosivuston blogi 2.4.2019. Viitattu 18.5.2022. https://blog.oppia.fi/2019/04/02/agile-mika-milloin-miksi/.

Buchanan, I. N.d. History of DevOps. Atlassianin verkkosivut. Viitattu 20.5.2022. https://www.atlassian.com/devops/what-is-devops/history-of-devops.

Continuous Delivery vs. Continuous Deployment: Where to draw the line? N.d. Katalonin verkkosivut. Viitattu 2.9.2022. https://katalon.com/resources-center/blog/continuous-delivery-vs-continuous-deployment.

DevOps principles. N.d. Atlassianin verkkosivut. Viitattu 19.5.2022. https://www.atlassian.com/devops/what-is-devops.

DevOpsin hyödyt korostuvat tuotantoonviennissä – valttina ketteryys ja nopeus. 2022. Profit Softwaren verkkosivuston uutiset. Julkaisu 1.2.2022. Viitattu 16.9.2022. https://profitsoftware.com/devopsin-hyodyt-korostuvat-tuotantoonviennissa-valttina-ketteryys-ja-nopeus/?lang=fi.

Forsell, J. 2020. DevOps-toimintamalli. Opinnäytetyö, AMK. Tampereen ammattikorkeakoulu, tietojenkäsittelyn tutkinto-ohjelma. Viitattu 18.9.2022. https://urn.fi/URN:NBN:fi:amk-2020120726476.

Fowler, M. 2006. Continuous Integration. Martin Fowlerin verkkosivut 1.5.2006. Viitattu 2.9.2022. https://martinfowler.com/articles/continuousIntegration.html.

Git Branches: List, Create, Switch to, Merge, Push, & Delete. 2021. Noble Desktopin verkkosivut. Julkaistu 6.10.2021. Viitattu 7.9.2022. https://www.nobledesktop.com/learn/git/git-branches.

Halttunen, J. 2022. DevOps ja tuotannon monitorointiratkaisu. Opinnäytetyö, AMK. Jyväskylän ammattikorkeakoulu, tieto- ja viestintätekniikan tutkinto-ohjelma. Viitattu 16.9.2022. https://urn.fi/URN:NBN:fi:amk-2022052311262.

Heikkinen, J. 2021. Jatkuva toimittaminen Azure DevOps -ympäristössä. Opinnäytetyö, AMK. Jyväskylän ammattikorkeakoulu, tieto- ja viestintätekniikan tutkinto-ohjelma. Viitattu 16.9.2022. https://urn.fi/URN:NBN:fi:amk-2021060113127.

Host your own build agent in Azure Pipelines. N.d. Microsoftin Learn -alustan verkkosivut. Viitattu 20.9.2022. https://learn.microsoft.com/en-us/training/modules/host-build-agent/.

Hristov, A. N.d. How automated testing enables DevOps. Atlassianin verkkosivut. Viitattu 10.6.2022. https://www.atlassian.com/devops/devops-tools/test-automation.

Ilves, K. & Luukkainen, M. 2021. Ohjelmistotuotanto 2021 Osa 1. Helsingin yliopiston ohjelmistotuotantokurssin materiaali. Viitattu 17.6.2022. https://ohjelmistotuotanto-hy.github.io/osa1/.

Ilvonen, J. 2019. CI/CD-järjestelmä Azure DevOps -ympäristöllä. Opinnäytetyö, AMK. Kaakkois-Suomen ammattikorkeakoulu, tieto- ja viestintätekniikan koulutus. Viitattu 2.9.2022. https://urn.fi/URN:NBN:fi:amk-2019052712085.

Introduction to DevOps: What it is, concepts, and objectives. 2021. Bixlabs verkkosivuston blogi. Julkaistu 25.3.2021. Viitattu 14.9.2022. https://www.blog.bixlabs.com/post/devops-introduction.

ITIL 4 on täällä – ja se on ketterämpi kuin koskaan. N.d. Avoset Oy blogi. Viitattu 23.5.2022. https://www.avoset.fi/blogi/itil-4-on-taalla-ja-se-on-ketterampi-kuin-koskaan.

Julistuksen takana olevat periaatteet. 2001. Ketterän ohjelmistokehityksen julistus. Viitattu 27.5.2022. https://agilemanifesto.org/iso/fi/principles.html.

Karhusaari, M. 2022. Tietokone työvälineenä Osa 2 - Versionhallinta: Git ja Github. Helsingin yliopiston avoimen yliopiston materiaali. Viitattu 12.9.2022. https://tkt-lapio.github.io/git/.

Ketterän ohjelmistokehityksen julistus. 2001. Ketterän ohjelmistokehityksen julistus. Viitattu 30.5.2022. https://agilemanifesto.org/iso/fi/principles.html.

Klemetti, H. 2019. DevOps-malli yrityksen ICT-yksikössä, TyTA-kokonaisuus. Opinnäytetyö, AMK. Hämeenlinnan korkeakoulukeskus, tietojenkäsittelyn koulutusohjelma. Viitattu 10.6.2022. https://urn.fi/URN:NBN:fi:amk-2019112221945.

Koriseva, L. 2018. Jatkuvan toimituksen käyttöönotto ohjelmistoprojektissa. Diplomityö. Tampereen teknillinen yliopisto, tietotekniikan koulutusohjelma. Viitattu 3.9.2022. https://urn.fi/URN:NBN:fi:tty-201811212645.

Mezak, S. 2018. The Origins of DevOps: What's in a Name? DevOps.com verkkosivut 25.1.2018. Viitattu 12.9.2022. https://devops.com/the-origins-of-devops-whats-in-a-name/.

Montonen, J. 2021. DevOps-käytäntöjen hyödyntäminen Microsoft Azure -pilvialustalla. Opinnäytetyö, AMK. Jyväskylän ammattikorkeakoulu, tieto- ja viestintätekniikan tutkinto-ohjelma. Viitattu 6.9.2022. https://urn.fi/URN:NBN:fi:amk-2020060115767.

Palvelut. N.d. Profit Softwaren verkkosivut. Viitattu 17.5.2022. https://profitsoftware.com/palvelut/?lang=fi.

Peña, L. 2017. Infographic - What's the difference between Continuous Delivery and Continuous Deployment? GoCD CI/CD-serverin verkkosivut 17.10.2017. Viitattu 2.9.2022. https://www.gocd.org/2017/10/17/difference-between-continuous-delivery-continuous-deployment-infographic/.

Pihlajamäki, J. N.d. Mitä on DevOps? Otaverkon verkkosivujen artikkelit. Viitattu 17.5.2022. https://otaverkko.fi/mita-on-devops/.

Paul, F. 2014. The Incredible True Story of How DevOps Got Its Name. New Relic verkkosivuston blogi 16.5.2014. Viitattu 12.9.2022. https://newrelic.com/blog/nerd-life/devops-name.

Publish Build Artifacts task. 2022. Microsoftin dokumentaatio. Julkaistu 9.9.2022. Viitattu 13.9.2022. https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/utility/publish-build-artifacts?view=azure-devops.

Reece, S. 2019. Best Practices for Performance Testing in Continuous Integration. Inedo verkkosivuston blogi 31.7.2019. Viitattu 2.9.2022. https://blog.inedo.com/continuous-integration-performance-testing-best-practices.

Rehkopf, M. N.d. What is continuous integration? Atlassianin verkkosivut. Viitattu 23.5.2022. https://www.atlassian.com/continuous-delivery/continuous-integration.

Rossel, S. 2017. Continuous integration, delivery, and deployment : reliable and faster software releases with automating builds, tests, and deployment. Birmingham, England; Mumbai, [Maharashtra]: Packt. Viitattu 23.5.2022. https://janet.finna.fi, Ebook Central Academic Complete International Edition.

Run quality tests in your build pipeline by using Azure Pipelines. N.d. Microsoftin Learn -alustan verkkosivut. Viitattu 26.8.2022. https://docs.microsoft.com/en-us/learn/modules/run-quality-tests-build-pipeline/.

Self-hosted Windows agents. 2022. Microsoftin dokumentaatio. Julkaistu 21.4.2022. Viitattu 26.9.2022. https://learn.microsoft.com/en-us/azure/devops/pipelines/agents/v2-windows?view=azure-devops#choose-interactive-or-service-mode.

Tietoa meistä. N.d. Profit Softwaren verkkosivut. Viitattu 17.5.2022. https://profitsoftware.com/tietoa-meista/?lang=fi.

Use predefined variables. 2022. Microsoftin dokumentaatio. Julkaistu 30.8.2022. Viitattu 14.9.2022. https://docs.microsoft.com/en-us/azure/devops/pipelines/build/variables?view=azure-devops&tabs=yaml.

Vanhatapio, J. 2021. Versionhallinta ja Git WordPress-sivustolla. Zonerin verkkosivut 18.3.2021. Viitattu 7.9.2022. https://www.zoner.fi/wordpress/versionhallinta-git/.

Visual Studio Build task. 2022. Microsoftin dokumentaatio. Julkaistu 9.9.2022. Viitattu 13.9.2022. https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/build/visual-studio-build?view=azure-devops.

Waterfall vs. Agile vs. DevOps vs. Lean. N.d. Professional DevOps.com verkkosivut. Viitattu 18.5.2022. https://www.professional-devops.com/waterfall-agile-devops-lean.html.

What is Azure Boards? 2022. Microsoftin dokumentaatio. Julkaistu 3.8.2022. Viitattu 12.8.2022. https://docs.microsoft.com/fi-fi/azure/devops/boards/get-started/what-is-azure-boards?view=azure-devops.

What is Azure DevOps? N.d. DevOpsGroupin verkkosivut. Viitattu 12.8.2022. https://www.devopsgroup.com/insights/resources/tutorials/all/what-is-azure-devops/.

What is Azure Pipelines? 2022. Microsoftin dokumentaatio. Julkaistu 12.8.2022. Viitattu 3.9.2022. https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops.

What is Azure Repos? 2022. Microsoftin dokumentaatio. Julkaistu 27.6.2022. Viitattu 6.9.2022. https://docs.microsoft.com/en-us/azure/devops/repos/get-started/what-is-repos?view=azure-devops.

What is Azure Test Plans? 2022. Microsoftin dokumentaatio. Julkaistu 9.9.2022. Viitattu 13.9.2022. https://docs.microsoft.com/en-us/azure/devops/test/overview?view=azure-devops.

What is CI/CD? N.d. Red Hatin verkkosivut. Päivitetty toukokuussa 2022. Viitattu 23.5.2022. https://www.redhat.com/en/topics/devops/what-is-ci-cd.

What Is a Data Pipeline? N.d. Hazelcast verkkosivusto. Viitattu 14.9.2022. https://hazelcast.com/glossary/data-pipeline/.

What is Git. N.d. Atlassianin tutoriaalit. Viitattu 12.9.2022. https://www.atlassian.com/git/tutorials/what-is-git.

What is YAML? 2021. Red Hatin verkkosivut. Julkaistu 18.6.2021. Viitattu 5.9.2022. https://www.redhat.com/en/topics/automation/what-is-yaml.