Joonas Ilvonen

CI/CD-järjestelmä Azure DevOps -ympäristöllä

Opinnäytetyö
Tieto- ja viestintätekniikan koulutus

2019

Kaakkois-Suomen ammattikorkeakoulu

Kaakkois-Suomen ammattikorkeakoulu

Tekijä/Tekijät	Tutkinto	Aika
Joonas Ilvonen	Insinööri (AMK)	Toukokuu 2019
Opinnäytetyön nimi		30 sivua
CI/CD-järjestelmä Azure DevOps -ympäristöllä		
Toimeksiantaja		
GoodLife Technology Oy		
Ohjaaja		
Lehtori Niina Mässeli		

Tiivistelmä

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa toimeksiantajalle DevOps-toimintamallin mahdollistava jatkuvan toimittamisen järjestelmä yrityksen tuotteiden pilvipäivitysjärjestelmän tueksi. Järjestelmän tulisi automatisoidusti koostaa ja asettaa tuotteiden uudet versiot julkaisuvalmiuteen pilvipäivitysjärjestelmään.

Työ alkoi selvittämällä ohjelmistotuotannon automatisointiin liittyviä työkaluja ja menetelmiä. Jotta järjestelmä pystyttäisiin rakentamaan, täytyi ensin ymmärtää, mitä järjestelmältä vaadittiin. Taustatutkimuksen jälkeen valittiin lukuisista automatisointijärjestelmistä se, joka kaikkein todennäköisimmin on yhteensopiva toimeksiantajan olemassa olevan infrastruktuurin kanssa. Päädyttiin Microsoftin tekemään Azure Pipelines -palveluun. Valinta tuntui loogiselta sen ollessa osa Azure-tuoteperhettä, jota toimeksiantaja käyttää pilvipalveluratkaisunaan.

Työn aikana toteutettiin testiympäristössä järjestelmä, joka automatisoidusti koostaa ja lähettää tuotteen toimeksiantajan pilvipäivitysjärjestelmään. Tämän perusteella pystyttiin todistamaan Azure Pipelines -palvelun toimivuus toimeksiantajan ympäristössä. Jatkokehitysideana olisi järjestelmän käyttöönotto tuotantoympäristöön kokonaisuudessaan ja toimeksiantajan muiden tuotteiden lisäys.

Asiasanat

Azure DevOps, DevOps, jatkuva integraatio, jatkuva toimittaminen

South-Eastern Finland University of Applied Sciences

Author (authors)	Degree	Time
Joonas Ilvonen	Bachelor of Engineering	May 2019
Thesis title CI/CD system with Azure DevOps		30 pages
Commissioned by GoodLife Technology Oy		
Supervisor Niina Mässeli Senior Lecturer		

Abstract

The objective of this thesis was to design and implement a continuous delivery system based on DevOps operating model to support a cloud-based production update system. The system would be able to automate the compile and release process of software products into the commissioner's update system.

The work began by researching software production automation related systems and methods. For the implementation of the system to be successful, requirements for it had to be understood. After the background research, an automation system was selected based on which of the options would be the most compatible with the commissioner's environment. Azure Pipelines was chosen as a base of the system due to it being part of Azure product family which the commissioner relies upon with its cloud service solution.

As a result, the system was implemented in a sandbox environment, which builds and sends product updates to the update system. This proved that Azure Pipelines service could work well in the commissioner's environment. For the future development the system could be integrated into production in its entirety, and the commissioner's other products could be added.

Keywords

Azure DevOps, continuous delivery, continuous integration, DevOps

SISÄLLYS

TERMIT JA LYHENTEET

Agent-kone Tietokone esiasennetulla ympäristöllä, jolla ajetaan Azure Pipelines -töitten työvaiheita.

Azure DevOps Microsoftin palvelu, joka sisältää ohjelmistokehitykseen tarkoitettuja työkaluja.

Azure Marketplace Azure pilvipalvelun sisältämä kauppa, josta pystyy hankkimaan työkaluja ja palveluita.

Azure Pipelines Microsoftin ohjelmistokehitykseen tarkoitettu automatisointijärjestelmä Azure DevOps -palvelun sisällä.

Build Pipeline Automatisoidun koostamisen määrittelevä työ Azure Pipeline -järjestelmässä.

C++ Ohjelmointikieli.

C# Ohjelmointikieli.

CI/CD Continuous Integration / Continuous Delivery tai Continuous Deployment, kuvaa koko automatisoitua putkea koostamisesta ja testauksesta julkaisuun ja toimittamiseen.

Continuous Delivery Ohjelmistokehityskäytäntö, jolla pyritään mahdollistamaan ohjelmiston automatisoitu julkaisu.

Continuous Deployment
Ohjelmistokehityskäytäntö, jolla pyritään automatisoidusti julkaisemaan ohjelma jokaisen muutoksen jälkeen.

Continuous Integration
Ohjelmistokehityskäytäntö, jolla pyritään automatisoimaan ohjelmiston koostaminen ja testaus.

DevOps	Ohjelmistokehityksessä esiintyvä termi, jolla usein viitataan yrityksen toiminnan tehostamisessa käytettyihin kulttuuri muutoksiin, toimintatapoihin ja työkaluihin.
Koostaminen	Ohjelman lähdekoodin muuttamista tietokoneen ymmärtämään muotoon tuottaen ajettavan ohjelman.
Microsoft Azure	Microsoftin pilvipalvelu tuoteperhe.
MSBuild	Microsoft Build Engine, Microsoftin tekemä koostamistyökalu, jolla Visual Studio -projektit koostetaan.
Release Pipeline	Automatisoidun julkaisun määrittelevä työ Azure Pipelines -järjestelmässä.
SVN	Subversion, versionhallintajärjestelmä.
Visual Studio	Microsoftin tekemä ohjelmistokehitysympäristö.

1 JOHDANTO

Opinnäytetyön toimeksiantaja on GoodLife Technology Oy, joka on keskittynyt tuottamaan terveydenhuollon kuntoutuksen tarpeisiin suunnattuja ratkaisuja. Toimipisteet yrityksellä on Kotkassa ja Helsingissä, ja työntekijöitä on opinnäytetyön toteutuksen aikana noin kymmenen. Remote Trainer, TV Trainer ja Kiosk Trainer lukeutuvat yrityksen moninaisiin tuotteisiin. GoodLife Technology on perustettu vuonna 2013 vastaamaan kasvavan terveysteknologian digitalisaation tarpeisiin. (GoodLife Technology s.a.)

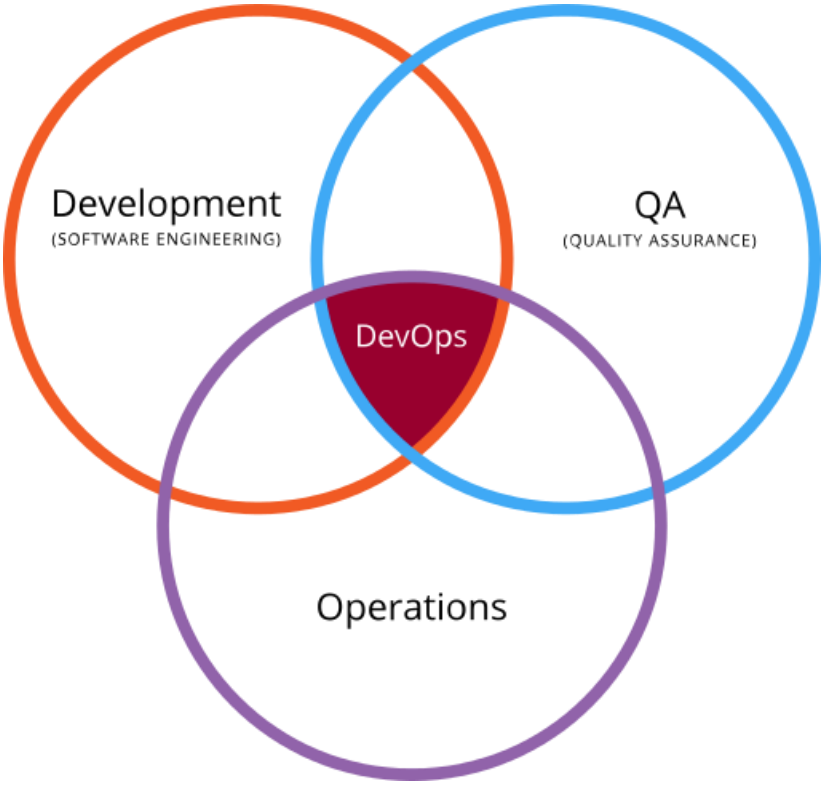
Tämän opinnäytetyön tarkoituksena on tuottaa toimeksiantajalle jatkuvan toimittamisen järjestelmä helpottamaan ja yksinkertaistamaan yrityksen tuotteiden etäpäivittämistä pilven kautta. Suunnitellun järjestelmän tarkoitus on automatisoida tuotteiden uusien versioiden koostamista ja niiden asettamista ladattavaksi toimeksiantajan olemassa olevaan pilvipäivitysjärjestelmään. Sen tehtävänä on toimia rajapintana, josta yrityksen tuotteet pystyvät kysymään ja lataamaan uusia versioita.

Opinnäytetyö koostuu viidestä osasta. Ensimmäisessä osassa käsitellään DevOps-termiä ohjelmistotuotannon automatisoinnin yhteydessä. Toisessa osassa käsitellään työn suunnittelua, vaatimuksia ja CI/CD-ratkaisua. Kolmannessa osassa käsitellään valittua CI/CD-ratkaisua tarkemmin. Neljännessä osassa kuvataan opinnäytetyön toteutuksen vaiheet. Viimeinen luku käsittelee toteutuksen onnistuneisuutta, jatkokehitysideoita ja pohdintaa työn aikana opituista asioista.

2 DEVOPS

DevOps-terminä on yhdistelmä sanoja development ja operations. Termille ei ole vakiintunutta käsitettä, mutta alkuperäisessä tarkoituksessa sillä tarkoitettiin ohjelmistokehittäjien ja -ylläpitäjien yhteistyön lisäämistä ohjelmistokehityksen eri vaiheissa (kuva 1). Nykypäivänä DevOps on yleistettynä yhdistelmä kulttuuria, toimintatapoja ja työkaluja, joilla tehostetaan yrityksen toimintaa (Guckenheimer 2018). Esimerkiksi Toivanen (2017) määrittelee termin toimintamallina seuraavasti: ”DevOps on toimintamalli, jossa ohjelmistojen rakentaminen, testaaminen ja julkaisu tapahtuu nopeasti ja luotettavasti mahdollisim-

man pitkälle automatisoidusti. Avaintermejä toiminnassa ovat jatkuva integraatio (Continuous Integration) ja jatkuva toimittaminen (Continuous Delivery). Lisäksi oleellisissa rooleissa on testauksen automatisointi (Testing Automation) ja ympäristöjen automatisoitu konfigurointi.”



Kuva 1. DevOps Venn -diagrammi (Devops s.a.)

Tämän luvun alakappaleissa käydään tarkemmin läpi DevOps-toimintamallin avaintermejä opinnäytetyön kannalta.

2.1 Jatkuva integraatio

Jatkuva integraatio on ohjelmistokehityskäytäntö, missä kehittäjätiimin jäsenet integroivat koodiansa versionhallinnan pääversioon säännöllisesti, vähintään päivittäin, mikä johtaa useisiin integraatioihin päivässä. Jokaisen integraation toimivuus varmistetaan ohjelmiston automatisoidulla koostamisella ja testauksella. Käytännön tavoitteena on nopeuttaa ohjelmiston kehitystä ja vähentää kehitykseen liittyviä ongelmia. (Fowler 2006.)

Jatkuvan Integraation yleistymiseen vaikutti tarve helpottaa erillään kehittäjätiimistä työskentelevien kehittäjien muutosten integraatiota versionhallintaan. Muutosten integraatioiden aikavälin välillä ollessa päiviä tai jopa viikkoja, luo

odottelu konflikteja integroidessa, vaikeasti korjattavia ohjelmointivirheitä, eriäviä lähdekoodityylejä ja turhaan toistettua työtä. (Guckenheimer 2017.)

Jatkuvan integraation käytännön toteuttamisessa ei ole yhtä ainutta tiettyä tapaa, vaan toteutus määräytyy kehittäjätiimin ja kehitysympäristön tarpeiden mukaan. Seuraavaksi on lueteltu käytäntöjä, joita jatkuvan integraation toteutukset usein sisältävät. (Fowler 2006.)

Versionhallinnassa kuuluisi ylläpitää yhtä ainutta versiota pääohjelmistosta

Versionhallinnan kuuluisi sisältää kaikki koostamiseen vaaditut tiedostot, mutta ei koostamisen tuottamaa tulosta. Koostamisen pitäisi pystyä tekemään minimaalisella työllä uudella koneella, johon on koostamiseen vaadittu ympäristö asennettu. (Fowler 2006.)

Vaikka jotkin nykyaikaiset versionhallintajärjestelmät tukevat haaroitettuja versioita, on tärkeää ylläpitää yksittäistä pääversiota. Usein haaroitettujen versioiden liika käyttö tuottaa ongelmia, kun niitä aletaan yhdistämään takaisin pääversioon. Pääversioon pitäisi integroida suurin osa kehittäjien tekemistä muutoksista. (Fowler 2006.)

Koostamisen automatisointi

Ohjelman koostaminen on usein monimutkainen prosessi sisältäen useita eri vaiheita, mutta sen ollessa yleensä muuttumaton, pystyy koostamisen automatisoimaan. Automatisoinnin hyötynä on inhimillisten virheiden vähentäminen ja kehittäjien toistuvien työvaiheiden vähentäminen. (Fowler 2006.)

Koostamisprosessiin automatisoitujen testien sisällyttäminen

Pelkkä koostaminen ei takaa ohjelman toimintaa. Vaikka ohjelma koostuisi virheittä, ei se takaa toimivuutta ajettaessa. Tästä syytä ohjelman koostamiseen on hyvä myös lisätä testejä, millä testataan ohjelman eri osien toimivuutta. Ku-

ten koostaminen, tulisi myös testien ajaminen olla automatisoitu. Duvallin, Matyasin ja Gloverin (2007) mukaan jatkuvassa integraatiossa automatisoidut testit ovat yhtä tärkeitä kuin automatisoitu koostaminen.

Kehittäjät integroivat muutoksensa pääversioon vähintään päivittäin

Kun kehittäjät lisäävät muutoksiansa pääversioon mahdollisimman useasti vähintään päivittäin, saavat kehittäjät nopeasti palautetta. Palautteen avulla pystytään nopeammin huomaamaan ja reagoimaan virheisiin. Tiheä muutosten lisäystahti myös vähentää mahdollisen päällekkäisen koodin syntymistä ja helpottaa paikantamaan ongelmia versionhallinnan historian avulla. (Duvall ym. 2007.)

Jokainen muutos versionhallinnassa pitää pystyä koostamaan integraatiokoneella

Kehittäjien tallentaessa muutoksiansa versionhallintaan päivittäin, saadaan koostamisen testausta säännölliseksi. Tämän pitäisi tarkoittaa, että pääversion koostaminen tapahtuisi ongelmitta, mutta virheitä silti tapahtuu. Yleisiä syitä on kurin puutteesta johtuvat yhteensopivuusongelmat muutoksien tallennuksessa ja kehitysympäristöjen eroavaisuudet. Tämänkaltaisen ongelmien välttämiseksi koostamisen onnistuminen pitää varmistaa säännöllisesti integraatiokoneella. (Fowler 2006.)

Integraatiokoneen tarkoitus on toimia esimerkkiympäristönä, missä koostamisen ja testien on onnistuttava pelkällä pääversion päivittämisellä versionhallinasta. Ympäristön tarkoituksena on löytää muutosten tallennuksessa tapahtuneita ongelmia, jotka aiheuttaisivat virheitä. Nykypäivänä integraatiokoneena käytetään yleisesti CI-palvelimia (Continuous Integration), joilla automatisoidaan integraatiokoneella koostaminen. (Fowler 2006.)

2.2 Jatkuva toimittaminen ja jatkuva julkaisu

Jatkuva toimittaminen on ohjelmistokehityskäytäntö, missä ohjelmaan tehdyt korjaukset, päivitykset ja uudet ominaisuudet automaattisesti valmistellaan jul-

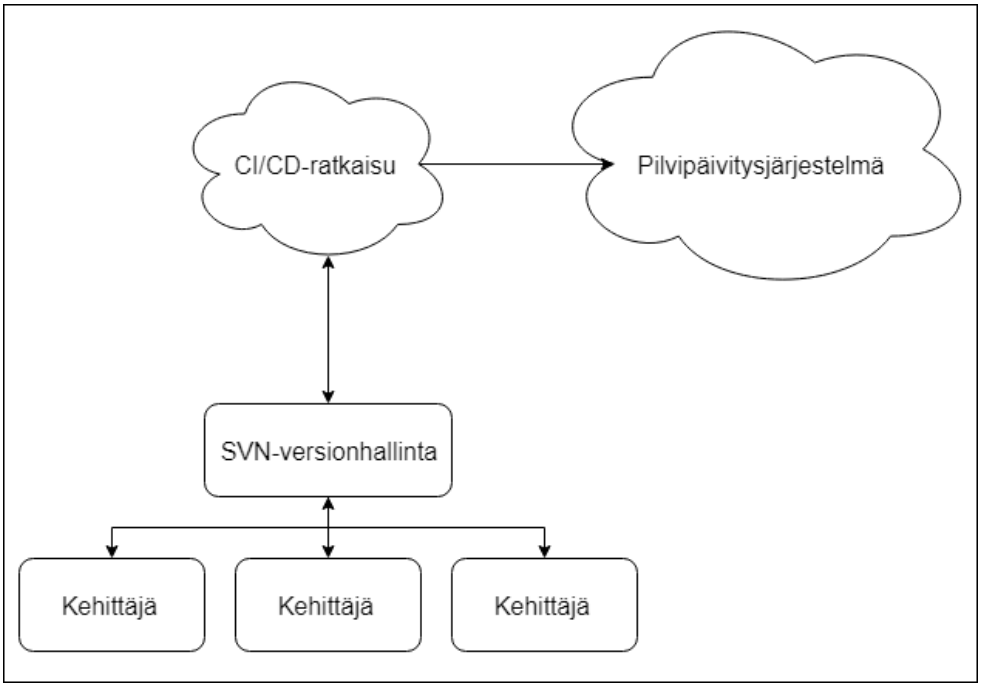
kaisua varten (What is Continuous Delivery? s.a.). Käytäntö on jatkoa jatkuvalle integraatiolle automatisoimalla julkaisu. Tavoitteena on, että ohjelmasta on aina olemassa julkaisukelpoinen versio (Fowler 2013).

Jatkuva julkaisu (Continuous Deployment) sekoitetaan ajoittain jatkuvaan toimittamiseen. Siinä missä jatkuvassa toimittamisessa ohjelman julkaisu tapahtuu manuaalisesti, jatkuvassa julkaisussa jokainen muutos ohjelmasta julkaistaan automaattisesti. (Fowler 2013.)

3 SUUNNITTELU

Ensimmäinen vaihe automatisoinnin suunnittelussa on miettiä, miten CI/CD-ratkaisu sijoittuu päivitysten tuottamisen prosessiin. Kuvassa 2 näkyy pelkistetysti päivitysten tuottamisen prosessi ja mihin kohtaa CI/CD-ratkaisu sijoittuisi prosessissa. Uusi päivitysten tuottamisen prosessi noudattaisi Duvall ym. (2007) laatimaa jatkuvan integraation yleistä toimintaa mukaillen:

1. Kehittäjä lisää tekemänsä muutoksen versionhallintaan. Samaan aikaan CI/CD-ratkaisu tarkistaa versionhallintajärjestelmästä tietyn väliajan välein muutosten tapahtumista.
2. Muutoksen tallennuksen jälkeen, CI/CD-ratkaisu huomaa muutoksen versionhallinnassa ja hakee versionhallinnasta uusimman version. Ratkaisu ajaa koostamiskomentosarjan (build script), joka integroi muutoksen ohjelmaan.
3. Ratkaisu tuottaa integraatiosta palautteen ja esittää sen kehittäjille, esimerkiksi sähköpostin välityksellä.
4. Onnistuneen koostamiskomentosarjan ajon jälkeen ratkaisu aloittaa jatkuvan toimituksen automatisoinnin, joka valmistelee koostamiskomentosarjan tuottaman ohjelman julkaisun. Onnistuessa julkaisu jää odottamaan käyttäjän hyväksyntää.
5. CI/CD-ratkaisu jatkaa versionhallinnan muutosten tarkistamista.



Kuva 2. Pilvipäivitysjärjestelmän suunniteltu prosessi

3.1 Vaatimukset

CI/CD-ratkaisun valinnan seuraavana vaiheena on vaatimusten määrittely. Vaatimuksien määrittelyllä pyritään selvittämään mitä kaikkea CI/CD-ratkaisulta vaaditaan. Määrittelyn tuloksena syntyi seuraavat vaatimukset, joista ratkaisun tulisi mahdollisimman monta täyttää:

- täytyy tukea SVN-tyyppistä versionhallintajärjestelmää.
- täytyy tukea Visual Studio -ohjelmistokehitysympäristön projektien koostamista.
- täytyy tukea Xamarin-mobiilisovelluskehityskehyksen projektien koostamista, Android ja iOS-alustalle.
- ratkaisun kuuluisi mieluiten olla pilvipalvelu, omille laitteille asentamista vältettävä.
- täytyy olla helppokäyttöinen ja edullinen.

Ratkaisun tulisi myös noudattaa yrityksessä olevia yleisiä vaatimuksia, mitä vaaditaan joka projektilta:

- teknisesti laadukas
- hyvä käytettävyys
- toimintavarma
- integroitavuus jo olemassa olevaan infrastruktuuriin.

3.2 CI/CD-ratkaisu

CI/CD on tapa jatkuvasti toimittaa tuotteita asiakkaille, automatisoinnin lisäämisellä ohjelmistokehityksen eri vaiheisiin. Termi tulee lyhenteistä continuous integration (CI) ja continuous delivery tai continuous deployment (CD). Tarkentaen CI/CD esittelee automatisoinnin ja jatkuvan valvonnan kehitettävän ohjelman koko kehityskaarelle aina integraatiosta ja testauksesta toimittamiseen ja julkaisuun (kuva 3). Näiden vaiheiden kokonaisuutta usein nimitetään CI/CD-putkeksi. Putkien automatisointiin käytetään usein CI/CD-ratkaisuja, esimerkiksi Jenkins-työkalua. (What is CI/CD? s.a.)

Kuva 3. CI/CD-putki (What is CI/CD? s.a.)

Sopivimman ratkaisun etsiminen aloitettiin toimeksiantajan ehdotuksesta Jenkins-työkalusta. Jenkins on Java-ohjelmointikielellä tehty avoimen lähdekoodin automatisointipalvelin. Sillä pystytään automatisoimaan koostamista, testaamista ja ohjelman toimittamista ja julkaisua. Sen helppokäyttöisyys ja yksinkertainen käyttöliittymä tuottavat matalan oppimiskynnyksen. Toiminnallisuutta saa laajennettua huomattavasti moninaisten avoimen lähdekoodin liitännäiskomponenttien avulla, joita tulee jatkuvasti lisää ja tarvittaessa pystyy itse kehittämään. (Jenkins User Documentation s.a.; Smart 2011.)

Liitännäiskomponenttien avulla Jenkins saisi täytettyä suuren osan CI/CD-ratkaisulle asetetuista vaatimuksista. Se olisi myös saatavilla valmiina levykuvana Azure Marketplace -palvelusta, mikä helpottaisi käyttöönottoa. Käyttöönoton osalta heräsi kuitenkin joitakin huolia:

- Vaikka Azure Marketplace -palvelusta on saatavilla valmis levykuva, sen käyttöönotto vaikutti monimutkaiselta.
- Levykuvan virtuaalikoneen käynnissä pitämisen hinta.
- Liitännäiskomponenttien ohjelmointikielen ollessa Java, mitä yrityksessä ei käytetä, voisi se tuottaa ongelmia, jos tarve itse tekemälle komponentille todetaan.

- Automatisointiin käytettävien orjakoneiden hinnasta Azure-pilvipalvelun sisällä ei ollut varmuutta.

Jenkins-työkalun yhteensopivuuden selvittämisen aikana Microsoft ilmoitti uudesta Azure DevOps -palvelusta, joka sisälsi Azure Pipelines -automatisointijärjestelmän (Coole 2018). Jenkins-selvitystyö keskeytyi Azure Pipelines -palvelun yhteensopivuuden varmentamisen ajaksi. Azure Pipelines on pilvipalvelu, jolla pystytään automatisoimaan ohjelmistoprojektien koostamista, testaamista ja julkaisua. Palvelun aloittaminen onnistuu helposti Azure DevOps -ympäristön käyttöönottamisella. Palvelun toiminnallisuutta pystytään laajentamaan liitännäisillä Azure Marketplace -palvelusta tai tekemällä itse. Palvelu tarjoaa Agent-koneet Windows-, Linux- ja MacOS-käyttöjärjestelmillä, mikä mahdollistaisi myöskin iOS Xamarin -projektien lisäämisen tulevaisuudessa. Windows Agent -koneilla on esiasennettuna Visual Studio -projektien koostamiseen tarvitut ohjelmat ja palvelut. Palvelussa on ilmainen taso yksityisille projekteille, mikä sisältää 1 800 minuuttia automatisointiputkien ajoaikaa ilman rinnakkaisuutta. (What is Azure Pipelines? 2018.)

Azure Pipelines päätettiin valita CI/CD-ratkaisuksi sen täyttäessä kaikki määritellyt vaatimukset. Myöskin sen ollessa osa Azure-tuoteperhettä, jota toimeksiantaja käyttää pilvipalveluratkaisunaan vaikutti merkittävästi päätökseen.

Myös muita automatisointipalveluita ja -palvelimia tutkittiin, kuten Travis CI, BamBoo, GitLab, CircleCI ja Codeship. Mutta näiden tutkinta jäi vähäiselle Azure Pipelines täyttäessä toimeksiantajan määrittelemät vaatimukset hyvin.

4 AZURE DEVOPS

Azure DevOps -palvelu tarjoaa ohjelmistokehityspalveluita muun muassa toiminnanohjaukseen, versionhallintajärjestelmien ylläpitoon ja CI/CD-putkien automatisointiin. Palvelusta on olemassa kaksi versiota: Microsoftin ylläpitämä pilvipalvelu Azure DevOps Services vanhalta nimeltään Visual Studio Team Services (VSTS) ja paikallisesti asennettava Azure Devops Server vanhalta nimeltään Team Foundation Server (TFS). Palvelun tarkoituksena on tuottaa sisällöllään kattava ja yhtenäinen ympäristö ohjelmistotuotannon tarpeisiin.

Palvelut on suunniteltu toimimaan keskenään ja niitä pystyy ottamaan ja poistamaan käytöstä tarpeiden mukaan. (What is Azure DevOps? 2019; Azure DevOps Services… 2019.)

4.1 Azure Artifacts

Azure Artifacts -palvelu on Azure DevOps -ympäristön sisäinen paketinhallintajärjestelmä. Paketinhallintajärjestelmä tukee muun muassa Maven-, npm-, NuGet- ja Python-paketteja. Järjestelmään voi lisätä niin omia paketteja kuin julkisista palveluista löytyviä paketteja. Paketit tallentuvat järjestelmään ja niitä voidaan käyttää esimerkiksi Azure Pipelines -automatisointiputkissa tai Visual Studio -ohjelmistokehitysympäristössä pakettien lähteenä. (Azure Artifacts s.a.)

4.2 Azure Boards

Azure Boards -palvelu tarjoaa ominaisuudet ohjelmistotuotannon projektien toiminnanohjaukseen. Palvelu sisältää kattavan listan erilaisia ominaisuuksia, kuten tuen Scrum- ja Kanban-ketterille ohjelmistokehitysmenetelmille, muokattavia raporttinäkymiä ja mahdollisuuden tuottaa raportteja toiminnanohjauksen palvelun sisällöstä. (Bjork 2018.)

4.3 Azure Pipelines

Kuten aiemmassa luvussa kerrottiin, Azure Pipelines on pilvipalveluna toimiva automatisointipalvelu, jolla pystytään automatisoimaan ohjelmistoprojektien koostamista, testaamista ja julkaisua. Palvelu tukee liki kaikkia ohjelmointikieliä ja projektityyppejä. Palvelussa on mahdollista käyttää automatisointiputkien ajamiseen paikallisia koneita, joille on asennettu Agent-sovellus tai Microsoftin ylläpitämiä Agent-virtuaalikoneita. Microsoft tarjoaa ylläpidettyjä Agent-koneita Windows-, Linux- ja MacOS-käyttöjärjestelmillä. Virtuaalikoneille on esiasennettuna yleisiä koostamiseen tarvittuja työkaluja, kuten Visual Studio -projektien koostamisessa käytetty MSBuild-ohjelma. Palvelussa on ilmainen taso yksityisille projekteille, joka sisältää 1 800 minuuttia automatisointiputkien ajoaikaa ilman rinnakkaisuutta. (What is Azure Pipelines? 2018.)

4.4 Azure Repos

Azure Repos -palvelu tarjoaa kehittäjille mahdollisuuden luoda ja hallinnoida Git- ja TFVC-tyyppisiä versionhallintajärjestelmiä. Muutoksien tapahtuessa versionhallintajärjestelmissä ilmestyvät ne myös DevOps-portaaliin. Näihin tapahtumiin pystytään viittamaan pilviportaalin sisällä, esimerkiksi kommentoimalla muutoksen sisältöä tai linkittämällä muutos Azure Boards -palvelun työtehtävään. (What is Azure Repos? s.a.)

4.5 Azure Test Plans

Azure Test Plans -palvelu tarjoaa kehittäjille kattavat mahdollisuudet luoda, hallita, valvoa ja toteuttaa erilaisia testejä koko kehitysprosessin elinkaarelle. Palvelun avulla pystytään esimerkiksi tuottamaan ja seuraamaan jatkuvan integraation testien tulosta, tekemään käyttäjätestejä tietyistä järjestelmän osista ja määrittelemään näille käyttäjätesteille tekijät. Testien tuloksista tuotetaan kattavat tulossivut Azure DevOps -portaalissa, jotka sisältävät testien tulokset. (Azure Test Plans Documentation s.a.)

5 TOTEUTUS

Järjestelmän toteutuksen tavoitteena oli varmistaa Azure Pipelines -palvelun yhteensopivuus toimeksiantajan olemassa olevan ympäristön kanssa ja varmistuttaessa ottaa se käyttöön. Yhteensopivuuden varmistaminen tehtäisiin lisäämällä toimeksiantajan tuote järjestelmään aiemmin esitetyllä CI/CD-putkella. Toteutusta varten toimeksiantajan pilvipäivitysjärjestelmästä tehtiin kopio, jota käytettiin testiympäristönä. Näin ympäristössä pystyttäisiin tekemään isojakin muutoksia ilman pelkoa siitä, että tehdyt muutokset vaikuttaisivat käytössä oleviin järjestelmiin.

Järjestelmän toteutus alkoi uuden ympäristön luomisella Azure DevOps Services -palveluun. Ympäristön luomisen jälkeen tapahtui tuotteen CI/CD-putken automatisoinnin tarvitsemat vaiheet.

5.1 Azure DevOps -palvelun käytön aloittaminen

Käytön aloittaminen tapahtuu kirjautumalla Azure DevOps -palveluun. Kirjautuminen vaatii Microsoft-tilin, jonka pystyy myös luomaan kirjautumisvaiheen aikana. Kirjautumisen jälkeen käyttäjälle luodaan automaattisesti organisaatio, jos tiliä ei ole jo liitetty organisaatioon. Käytön aloittaminen on myös mahdollista tehdä sähköpostitse lähetettävän kutsun kautta liittymällä toisen henkilön organisaatioon.

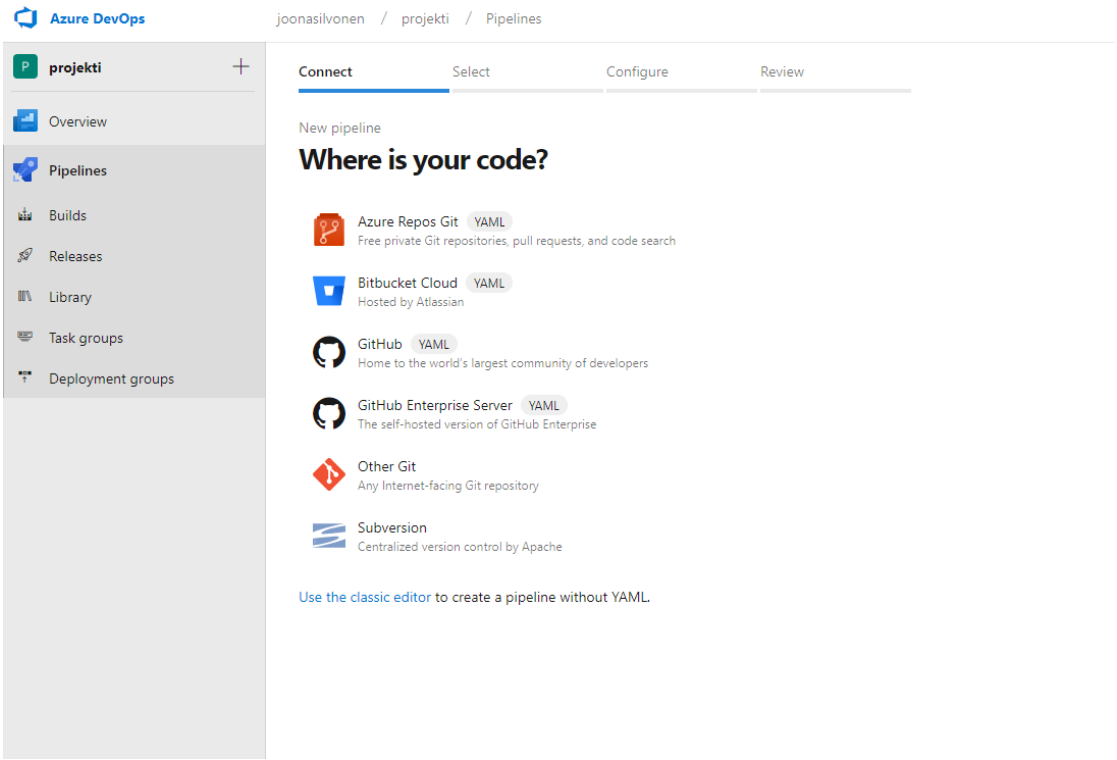
Organisaatio on itsessään pelkkä työympäristö mikä sisältää organisaation asetukset ja organisaatio kohtaiset projektit. Yksi Microsoft-tili voi olla liitettynä useisiin organisaatioihin.

5.2 Projektin luominen

Koska organisaatio itsessään ei sisällä toiminnallisuutta, seuraavana vaiheena oli projektin luominen organisaatioon. Yksittäinen projekti sisältää Azure DevOps -palvelun toiminnalliset ominaisuudet, kuten projektinhallintajärjestelmäpalvelu Boards, versionhallintapalvelu Repos tai automatisointipalvelu Pipelines. Toiminnallisten ominaisuuksien käyttö määritellään projektikohtaisesti. Opinnäytetyötä varten luotiin uusi projekti, jossa otettiin käyttöön pelkästään Pipelines muiden palveluiden ollessa tarpeettomia opinnäytetyön toteutuksen kannalta.

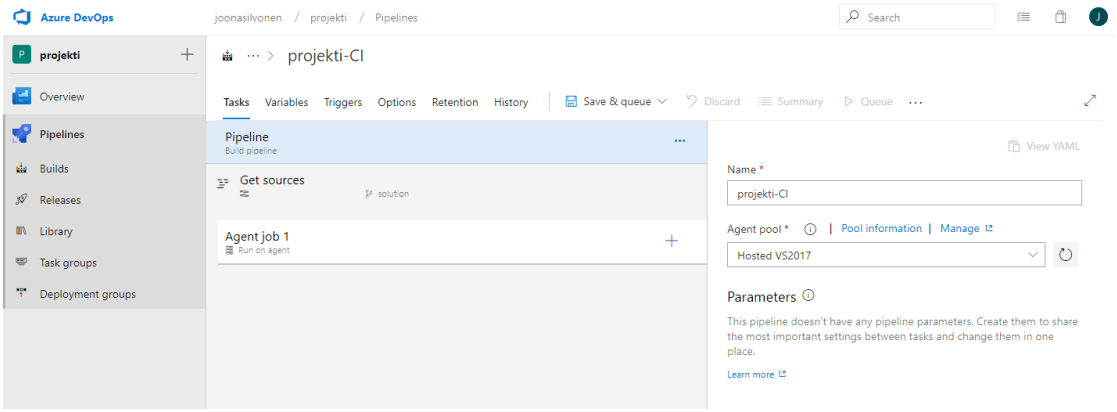
5.3 Tuotteen lisääminen järjestelmään

Seuraavaksi luotiin uusi Build Pipeline -työ, jonka tarkoituksena on automatisoidusti koostaa tuote. Kuvassa 4 näkyy luonnin ensimmäinen vaihe, jossa käyttäjää pyydetään valitsemaan versionhallintajärjestelmä, josta projektin tiedostot haetaan. Versionhallinnan asettamisen jälkeen on mahdollista valita mallipohja. Mallipohja luo automaattisesti valitun tyyppisen projektin yleisiä työvaiheita. Tuotteen ohjelmointikielen ollessa C++ ei mallipohjaa pystytty käyttämään, jolloin jouduttiin jatkamaan ilman mallipohjaa.



Kuva 4. Build Pipeline -luonnin versionhallinnan kysymisen vaihe

Kuvassa 5 on tyhjän Build Pipeline -työn pohja, mikä sisältää vain versionhallinnasta tiedostojen hakemisen työvaiheettomalle Agent-koneelle. Agent-kone on vastine orjakoneelle palvelun sisällä. Seuraavaksi alkoi työvaiheiden lisääminen Agent-koneen sisään automaattista koostamista varten.



Kuva 5. Tyhjä Build Pipeline

Työvaiheisiin on mahdollista käyttää Azure Pipelines -palvelun sisältämiä muuttujia, jotka työn suorituksen alkaessa muutetaan oikeiksi arvoiksi (kuva 6). Muuttujien käyttö mahdollistaa työvaiheiden modulaarisen käytön, kun vaiheiden määrittelyssä ei tarvitse tietää Agent-koneen sisältämää kansiorakennetta.

Display name *

Copy Files to: $(Build.StagingDirectory)

Source Folder

$(Build.SourcesDirectory)/Snapshot

Contents *

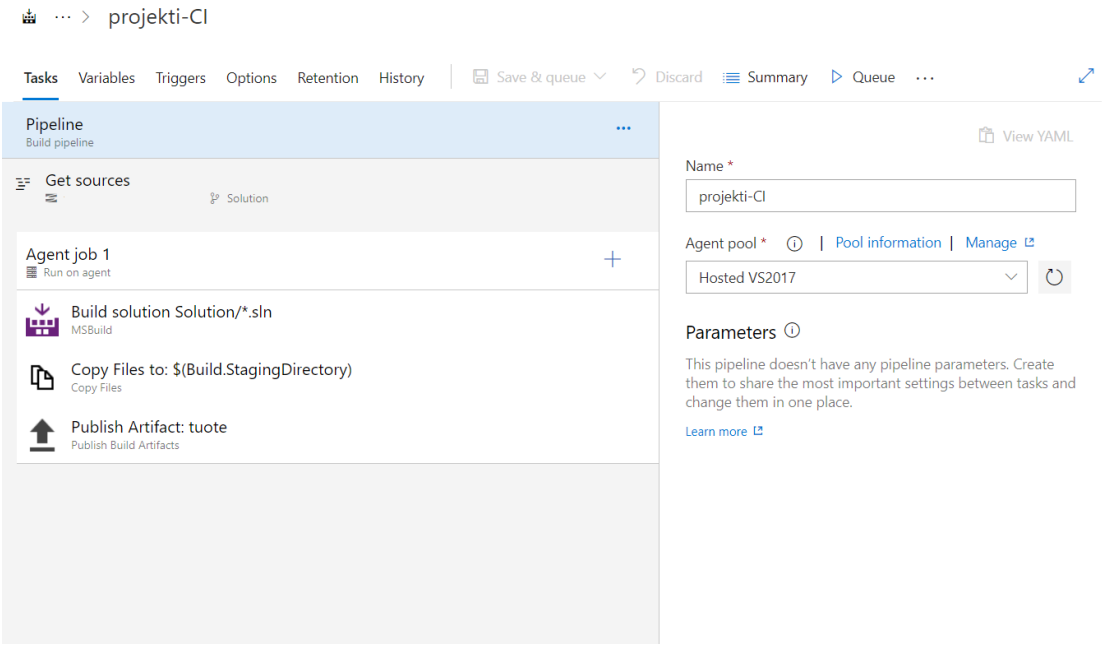
/Data/
*.exe

Target Folder *

$(Build.StagingDirectory)

Kuva 6. Työvaiheessa muuttujien käyttö

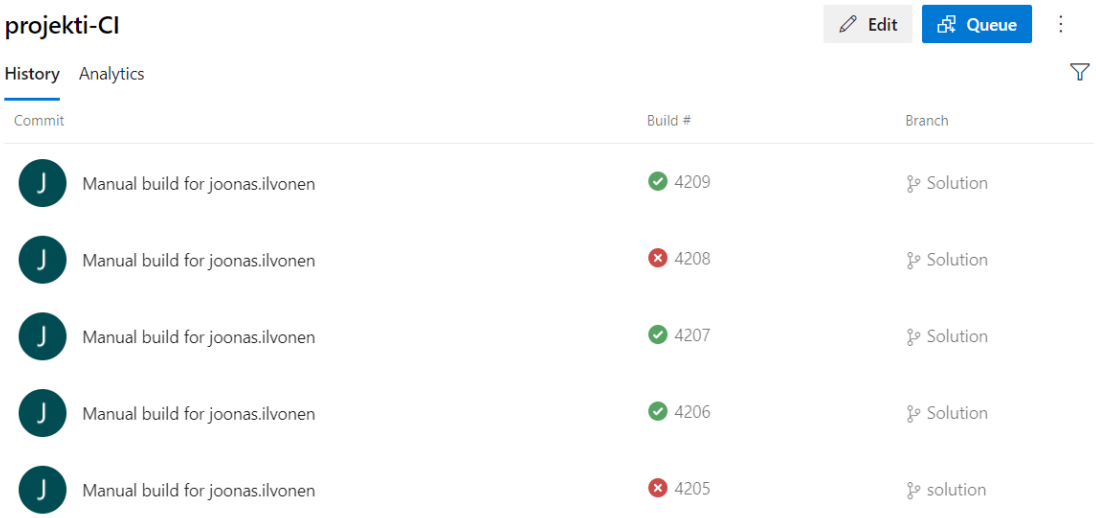
Minkä tahansa projektin koostamisen yleinen ensimmäinen vaihe on valmistella ympäristö koostamista varten, esimerkiksi tarvittavien kirjastojen lataamisella. Tuotteen koostamisen vaatiessa vain Windows-järjestelmän sisältämiä kirjastoja ei ollut tarvetta esivalmistelun työvaiheille, joten ensimmäiseksi työvaiheeksi pystyttiin suoraan lisäämään tuotteen koostaminen. Koostaminen tapahtui MSBuild-työvaiheen avulla, jolle asetettiin kohteeksi tuotteen projektitiedosto. Koostamisen jälkeen seuraavana vuorossa on ladata koostamisen luomat tiedostot koneelta Azure Pipelines -palveluun. Lataamista varten täytyi esivalmisteluna tiedostot siirtää Azure Pipelines -palvelun ennalta tekemään Staging-kansioon. Tiedostojen siirtäminen toteutettiin kopiointityövaiheella, jolla ne kopioitiin Staging-kansioon. Viimeisenä vaiheena lisättiin artefaktien lataaminen Azure Pipelines -palveluun. Tähän toimintoon löytyy valmis työvaihe Agent-koneelle, mikä lataa Staging-kansion sisällön ja asettaa sen artefaktiksi kyseiselle Build Pipeline -työn ajolle. Kuvassa 7 näkyy tuotteen valmis Build Pipeline -työ.



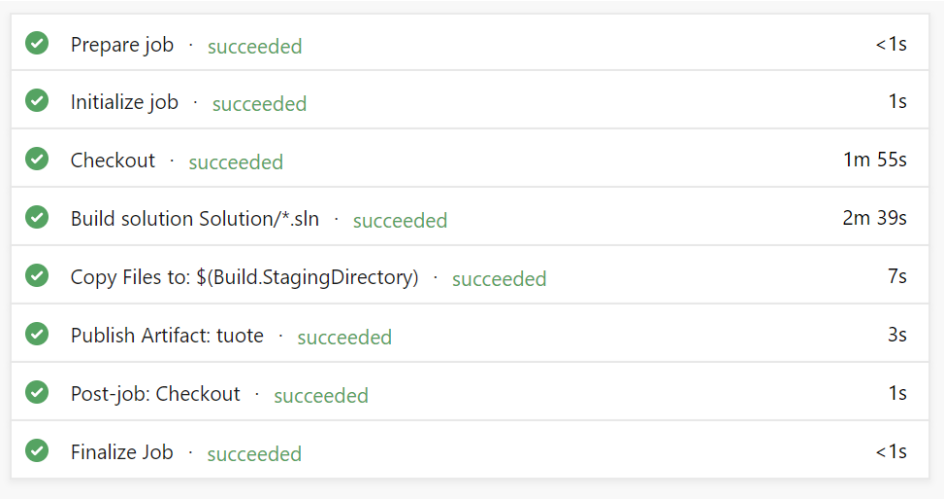
Kuva 7. Tuotteen Build Pipeline

Kun tuotteen Build Pipeline -työ oli valmis, seuraavana vuorossa oli ajaa se. Ajamisen pystyy tehdä manuaalisesti tai sen voi automatisoida muutamalla eri tavalla. Ensimmäisenä tapana on ajoitettu ajaminen, mikä käynnistää ajon tietyin aikavälein. Toisena tapana on versionhallinnan tarkkaileminen muutosten varalta, missä ajo käynnistyy palvelun huomatessa muutos versionhallinnassa. Viimeisenä tapana ajo käynnistyy, kun jokin toinen Build Pipeline -työ on ajettu. Ajamiseen määriteltiin automaattinen ajo muutoksen tapahtuessa versionhallinnassa, sen ollessa jatkuvan integraation kannalta normaali tapa automaattisen koostamisen ajolle. Työ ajettiin ensimmäisen kerran manuaalisesti, millä varmistettiin työn onnistuminen.

Jokaisesta työn ajosta tallentuu Azure Pipelines -palveluun merkintä ajon onnistumisesta (kuva 8) ja ajon sisältämien työvaiheiden lokiraportit (kuva 9). Jokainen ajon tekemä työvaihe luo lokiraportin kyseisen ajon tulokseen. Näihin lokiraportteihin tallentuu koneella ajetun komennon tuottamat lokirivit ja komentorivin sisältö. Tästä sisällöstä kuitenkin on suodatettu työvaiheiden asetuksista löytyvät arkaluontoiset tiedot kuten versionhallintaan liittyvät tunnukset.



Kuva 8. Tuotteen Build Pipeline -ajojen historia



Kuva 9. Suoritetun Build Pipeline -työn lokinäkymä

5.4 Package publisher

Kun tuotteen automatisoitu koostaminen oli tehty, seuraavaksi vuorossa oli julkaisu pilvipäivitysjärjestelmään. Release Pipeline -työn sisältämistä työvaiheista löytyi työvaiheita, millä pystyttäisiin lähettämään Azure-pilveen, mutta näillä työvaiheilla ei onnistuttaisi lähettämään päivityspakettia halutussa muodossa. Tämän takia todettiin tarve ohjelmalle, jonka tehtävänä olisi Release Pipeline -työssä lähettää pakattu tuote pilvipäivitysjärjestelmään.

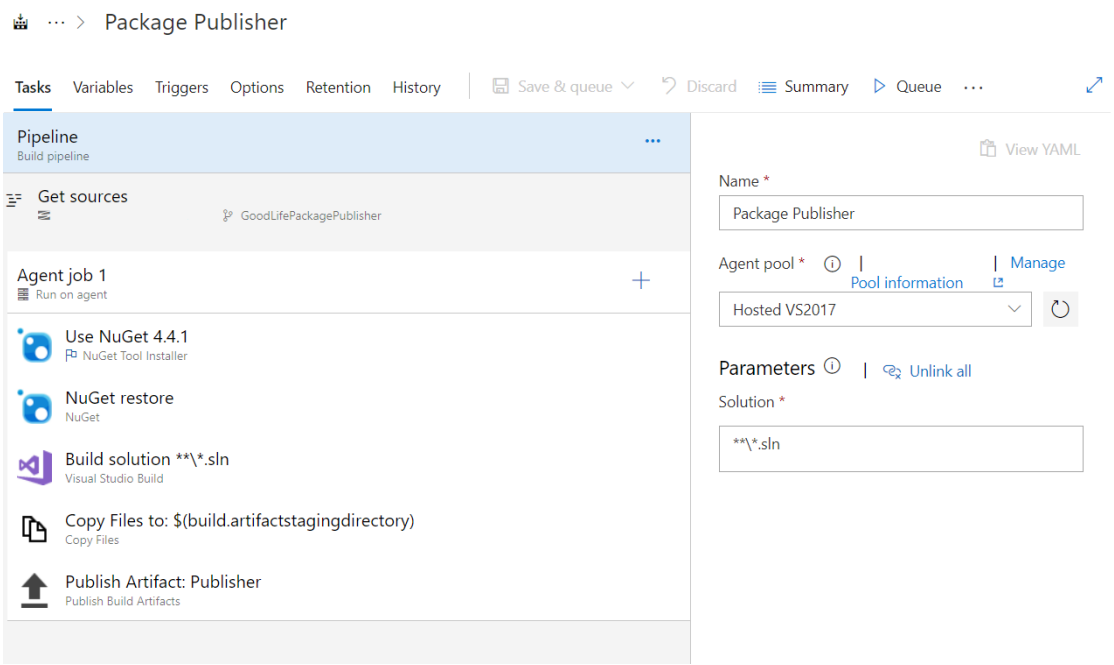
Ohjelman pohjana pystyttiin käyttämään yrityksen sisäiseen käyttöön tarkoitettua Package Manager -työkalua, jolla hallinnoidaan pilvipäivitysjärjestelmän paketteja. Package Manager on tehty C#-ohjelmointikielellä, jota käytetään myös Package Publisher -ohjelman kielenä, kielen vaihdosta aiheutuvan ylimääräisen työn välttämiseksi. Package Publisher toimisi seuraavasti: haetaan

Agent-koneesta Azure Pipelines asettamia ympäristömuuttujia (kuva 10), ja näitä muuttujia käyttäen määritetään parametrit pilvipäivitysjärjestelmään tehtävää julkaisua varten. Ohjelman toteutuksessa piti myös ottaa huomioon sen palautusarvo. Ohjelman ajon epäonnistuessa täytyisi sen palauttaa ohjelman epäonnistumista vastaava palautusarvo, jotta julkaisuputken ajaminen keskeytyy tapahtuneen virheen takia.

```
//Azure Release pipeline environment variables
string version = Environment.GetEnvironmentVariable("BUILD_SOURCEVERSION");
string projectName = Environment.GetEnvironmentVariable("BUILD_PROJECTNAME");
string releaseType = Environment.GetEnvironmentVariable("RELEASETYPE");
string releaseId = Environment.GetEnvironmentVariable("RELEASE_RELEASEID");
```

Kuva 10. Esimerkki ympäristömuuttujien hausta

Ohjelman tekemisen jälkeen sille tehtiin Build Pipeline -työ. Työn tekeminen tapahtui päämääräisesti samalla tavalla kuin tuotteen lisäämisessä aikaisemmassa kappaleessa. Package Publisher -ohjelman lähdekoodin kielen ollessa C# pystyttiin työn luomisessa hyödyntämään mallipohjaa. Mallipohja loi kuvan 11 mukaisen työn, mikä oli suoraan toimiva ilman muutoksia.



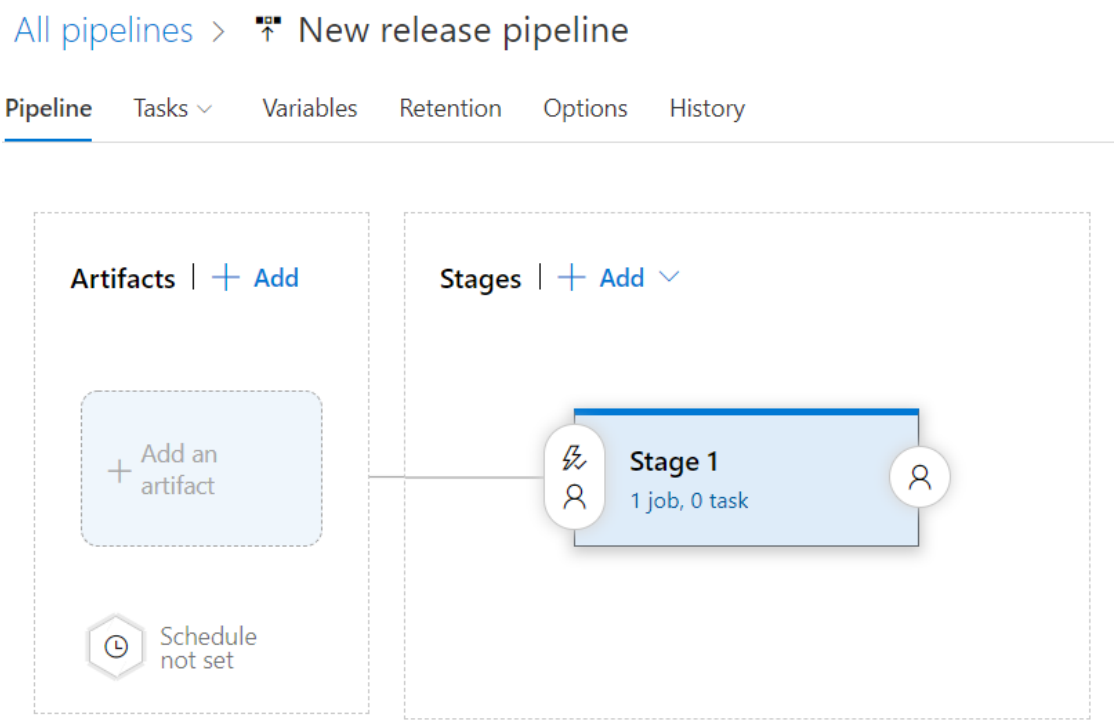
Kuva 11. Package Publisher Build Pipeline

5.5 Release Pipeline

Kun julkaisuun vaadittavien Build Pipeline -töiden lisäys järjestelmään oli valmis, voitiin Release Pipeline -työn tekeminen aloittaa. Työn tarkoituksena on

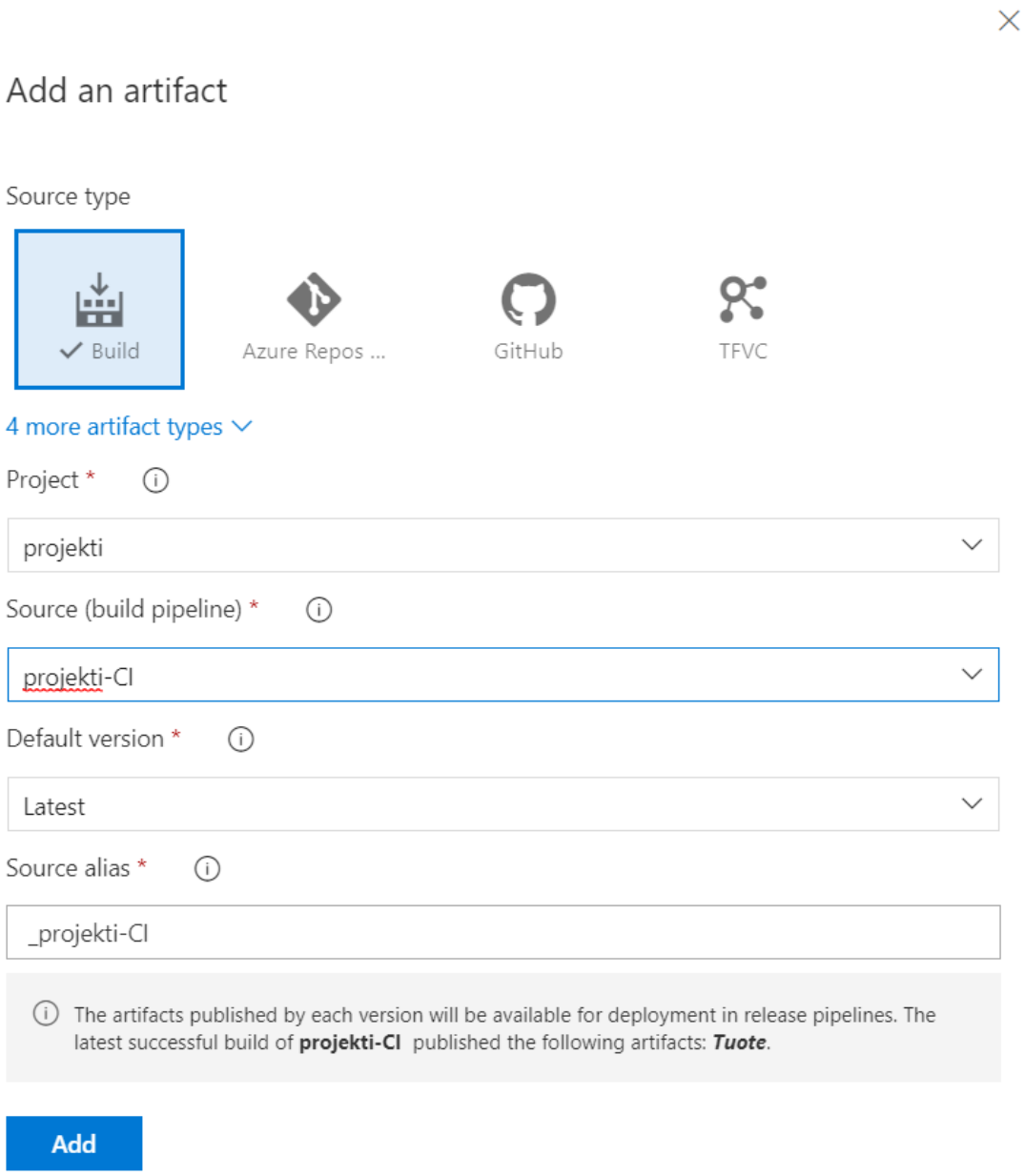
tuottaa tuotteen jatkuva toimittaminen pilvipäivitysjärjestelmään. Lopputuloksena työn pitäisi lähettää tuotteesta päivityspaketti toimeksiantajan pilvipäivitysjärjestelmään.

Kuten aiemmin Build Pipeline -työn tekemisessä, myös Release Pipeline -työn tekemisessä on mahdollista käyttää mallipohjaa. Näitä ei kuitenkaan voitu käyttää julkaisukohteen ollessa omatekoinen. Kuvassa 12 on tyhjän Release Pipeline -työn ympäristö, mikä sisältää vain työvaiheettoman Agent-koneen vaiheen (Stage) sisällä.



Kuva 12. Tyhjä Release Pipeline

Ensimmäisenä työhön täytyi lisätä artefaktit. Artefakteja käytetään julkaisun eri vaiheiden tiedostojen lähteinä, esimerkki artefaktista on Build Pipeline -työn koostama versio tuotteesta. Artefaktin lisääminen tapahtui kuvassa 12 näkyvässä ”Add an artifact” -kohdan painamisella. Tämän jälkeen avautuu kuvan 13 mukainen ikkuna, mistä artefaktien lisäys työhön tapahtuu. Työn artefakteihin lisättiin tuotteen Build Pipeline -työn tuottamat artefaktit ja määriteltiin se käyttämään viimeisintä onnistunutta koostamista. Tuotteen julkaisua pilvipäivitysjärjestelmää varten täytyi lisätä myös Package Publisher -ohjelman Build Pipeline -työn artefaktit.



Kuva 13. Artefaktin lisäys

Kun artefaktit oltiin saatu lisättyä, seuraavaksi täytyi työhön tehdä vaiheet. Yksittäisen vaiheen tarkoitus on tuottaa julkaisu tiettyyn ympäristöön, esimerkiksi testausympäristöön. Palvelun testauksen kannalta yksi vaihe riitti ja sen tarkoitus olisi lähettää päivityspaketti testiympäristössä olevaan pilvipäivitysjärjestelmään. Vaihe sisältää vaiheen ajoa säätelevät ehdot ja Pipeline -työn. Pipeline -töiden tarkoitus on ajaa julkaisuun vaadittavat työvaiheet. Vaiheen luomisen jälkeen täytyi sen sisältämään Pipeline-työhön lisätä tuotteen julkaisun työvaiheet. Julkaisun ensimmäinen työvaihe Agent-koneella on pakata tuotteen artefaktit lähettämisen helpottamiseksi. Pakkaamisen jälkeen oli vuorossa tuotteen pakatun tiedoston lähettäminen pilvipäivitysjärjestelmään. Tätä varten Package Publisher -ohjelma täytyy suorittaa Agent-koneella. Kuvassa

14 on PowerShell-työvaiheen komento, jolla Package Publisher -ohjelma suoritetaan.

```
&"${env:AGENT_RELEASEDIRECTORY}\_${env:RELEASE_ARTIFACTS__PACKAGE
_PUBLISHER_DEFINITIONNAME}\Publisher\GoodLifePackagePublisher.exe"

exit $LASTEXITCODE
```

Kuva 14. Package Publisher -ohjelman PowerShell -komento rivitettynä

Vaiheen lisäyksen jälkeen Release Pipeline oli sisällöltään valmis. Työn ajoa kokeiltiin vielä ja se todettiin toimivaksi. Näin Azure DevOps -testaus saatiin toteutettua.

Onnistuneen testauksen jälkeen aloitettiin toimeksiantajan tuotantoympäristöön tarkoitetun version teko. Tekemisen aikana Azure Pipelines -palvelussa tapahtui virhetilanne, jossa tuotteen Build Pipeline -työn automaattisen ajamisen laukaisin jäi virhetilaan, mikä toistuvasti käynnisti uuden ajon. Virhetilanne tapahtui viikonlopun aikana, jolloin se huomattiin vasta viikon alussa. Kun tilanne saatiin hallintaan poistamalla käytöstä laukaisin, oli palvelusta jo käytetty ilmaisen tason sallima töiden ajoaika. Opinnäytetyön aikataulutukseen asetettujen rajojen ylittyessä virheen takia päätettiin toimeksiantajan ja tekijän yhteispäätöksellä opinnäytteen työvaihe päättyneeksi.

6 PÄÄTELMÄT JA JATKOKEHITTÄMINEN

Alkuperäisen suunnitelman mukainen järjestelmän käyttöönotto epäonnistui virhetilanteen aiheuttaman aikatauluviiveen takia. Vaikkakaan lopulliseen työn tavoitteeseen ei päästy, pystyttiin työllä todistamaan Azure Pipelines -palvelun hyvä sopivuus toimeksiantajan ympäristöön. Näkisin ensimmäisenä jatkokehityskohteena käyttöönoton toimeksiantajan ympäristössä. Azure DevOps -palvelusta voitaisiin myös ottaa muitakin osia käyttöön kuin pelkkä Azure Pipelines.

Työn kirjoitusvaiheessa löysin tavan lisätä omia työvaiheita järjestelmään. Tämän ominaisuuden avulla voisi Package Publisher -ohjelmaa jatkokehittää. Sen ollessa työvaiheena järjestelmässä ei tarvitsisi sitä enää lisätä tuotteiden Release Pipeline -töihin artefaktina.

Opinnäytetyössä tehdyn työn perusteella uskon automatisoinnin pystyvän parantamaan yrityksien ohjelmistotuotantoa huomattavasti. Automatisoimalla toistuvia työvaiheita, jää enemmän aikaa itse kehittämiseen. Myöskin automatisoimalla voidaan saada korjausten, uusien ominaisuuksien ja päivitysten toimittamisen aikaväliä lyhennettyä huomattavasti. Pelkkä automatisointi itsessään ei kuitenkaan tuota optimaalista tulosta, vaan myös kehittäjien toimintatapojen on mukailtava automatisoidun ohjelmistokehityksen toimintatapoja. Automatisoitu koostaminen ja testaaminen ei tuota tulosta, jos ne epäonnistuvat kehittäjän toiminnan takia. Automatisointi huonosti toteutettuna voi myös tuhlata enemmän aikaa kuin mitä se vähentää, minkä takia automatisoinnin käyttöönotto on tehtävä kunnolla yrityksessä.

Opin tekemisen ja suunnittelun aikana todella paljon ohjelmistotuotannon automatisoinnista. Aikaisempaa kokemusta minkäänlaisen automatisoinnin käytöstä ei ollut. Alkuperäinen suunnitelmani oli tehdä itse automatisointijärjestelmä, millä jatkuva toimittaminen toteutettaisiin, mutta onneksi työnantaja kertoi termejä, joilla asiaa kannatti lähteä tutkimaan. Työn tekemisen tuottaman tietotaidon kanssa koen ymmärtäväni ohjelmistotuotannon automatisoinnin periaatteet ja uskon pystyväni hyödyntämään niitä tarvittaessa.

LÄHTEET

Azure Artifacts s.a. Microsoft. WWW-dokumentti. Saatavissa: https://azure.microsoft.com/en-us/services/devops/artifacts/ [viitattu 18.5.2019].

Azure DevOps Services vs. Azure DevOps Server. 2019. Microsoft. WWW-dokumentti. Päivitetty 22.4.2019. Saatavissa: https://docs.microsoft.com/en-us/azure/devops/user-guide/about-azure-devops-services-tfs?view=azure-devops [viitattu 15.5.2019].

Azure Test Plans Documentation s.a. Microsoft. WWW-dokumentti. Saatavissa:https://docs.microsoft.com/en-us/azure/devops/test-plans/?view=azure-devops [viitattu 18.5.2019].

Bjork A. Deep dive into Azure Boards. 2018. Blogi. Saatavissa: https://azure.microsoft.com/en-us/blog/deep-dive-into-azure-boards/ [viitattu 15.5.2019].

Coole J. 2018. Introducing Azure DevOps. WWW-dokumentti. Saatavissa: https://azure.microsoft.com/en-us/blog/introducing-azure-devops/ [viitattu 8.11.2018].

Devops s.a. Wikipedia. WWW-dokumentti. Päivitetty 27.7.2018. Saatavissa: https://fi.wikipedia.org/wiki/Devops [viitattu 14.5.2019].

Duvall P., Matyas S, Glover A. 2007. Continuous Integration: Improving Software Quality and Reducing Risk. E-kirja. Boston: Addison-Wesley Professional. Saatavissa: https://learning.oreilly.com/home/ [viitattu 5.5.2019].

Fowler M. 2006. Continuous Integration. WWW-dokumentti. Saatavissa: https://martinfowler.com/articles/continuousIntegration.html [viitattu 14.4.2019].

Fowler M. 2013. ContinuousDelivery. WWW-dokumentti. Saatavissa: https://martinfowler.com/bliki/ContinuousDelivery.html [viitattu 5.5.2019].

GoodLife Technology s.a. WWW-dokumentti. Saatavissa: http://www.goodlife.technology/ [viitattu 2.5.2019].

Guckenheimer S. 2017. What is Continuous Integration? WWW-dokumentti. Päivitetty 4.4.2017. Saatavissa: https://docs.microsoft.com/en-us/azure/devops/learn/what-is-continuous-integration [viitattu 12.11.2018].

Guckenheimer S. 2018. What is DevOps? WWW-dokumentti. Päivitetty 4.1.2018. Saatavissa: https://docs.microsoft.com/en-us/azure/devops/learn/what-is-devops [viitattu 12.5.2019].

Jenkins User Documentation s.a. Jenkins. WWW-dokumentti. Saatavissa: https://jenkins.io/doc/ [viitattu 14.5.2019].

Smart J. 2011. Jenkins: The Definitive Guide. E-kirja. Kalifornia: O'Reilly Media, Inc. Saatavissa: https://learning.oreilly.com [viitattu 14.5.2019].

Toivanen I. 2017. DevOps jatkuvan kehittämisen tukena. Blogi. Saatavissa: https://www.octo3.fi/devops-jatkuvan-kehittamisen-tukena/ [viitattu 21.4.2019].

What is Azure Pipelines? 2018. Microsoft WWW-dokumentti. Päivitetty 6.7.2018 Saatavissa: https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops [viitattu 26.4.2019].

What is Azure Repos? s.a. Microsoft. WWW-dokumentti. Saatavissa: https://docs.microsoft.com/en-us/azure/devops/repos/get-started/what-is-repos?view=azure-devops [viitattu 13.5.2019].

What is Continuous Delivery? s.a. Amazon. WWW-dokumentti. Saatavissa: https://aws.amazon.com/devops/continuous-delivery/ [viitattu 22.4.2019].

What is CI/CD? s.a. Red Hat. WWW-dokumentti. Saatavissa: https://www.red-hat.com/en/topics/devops/what-is-ci-cd [viitattu 14.5.2019].

KUVALUETTELO