LAB University of Applied Sciences

Microsoft Dynamics -ympäristön toimituksen automatisointi

LAB-ammattikorkeakoulu
Tieto- ja viestintätekniikka, Ohjelmistotekniikka
2022
Otto Piskonen

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Piskonen, Otto	Opinnäytetyö, AMK	2022
	Sivumäärä	
	30	

Työn nimi

Microsoft Dynamics ympäristön toimituksen automatisointi

Tutkinto ja koulutusala

Insinööri (AMK), Tieto- ja viestintätekniikka

Toimeksiantajan nimi, titteli ja organisaatio (jos opinnäytetyöllä on toimeksiantaja)

Crosskey Banking Solutions Ab Ltd

Tiivistelmä

Opinnäytetyössä toteutettiin Crosskey-yritykselle soveltuvuusselvitys toimituksen automatisoinnista Microsoft Dynamics -alustalla toimivalle varainhoitojärjestelmälle. Automatisointiputkisto keskittyi varainhoitojärjestelmän versionhallinnan, yksikkötestaamisen ja ympäristön päivittämisen automatisointiin. Tavoitteena oli saada toimiva toteutus automatisoinnille.

Työssä tarkasteltiin, mitä on CI/CD ja miten sen ketteriä menetelmiä voidaan käyttää automatisoinnissa. Tutkittiin myös, mikä on Microsoft Dynamics -alusta ja miten sen ympäristöön toimitus voitaisiin automatisoida parhaiten. Automatisointi tuotettiin Azure DevOps -alustalle.

Automatisointiputkisto rakennettiin Azure DevOps -alustalle käyttäen avointa lähdekoodi kirjastoa xRM CI Frameworkia. Kirjasto mahdollisti Microsoft Dynamics -ympäristön toimituksen automatisoinnin ohjelmallisesti. Putkisto integroitiin Bitbucketiin, jotta siellä säilytettävä lähdekoodi oli putkiston käytössä.

Lopputulos on toimiva automatisointiputkisto Microsoft Dynamics -ympäristölle. Putkisto seuraa ketteriä CI/CD menetelmiä. Automatisointiputkisto on hyvä lähtökohta mahdolliselle lisäkehitykselle.

Asiasanat

Azure DevOps, Dynamics CRM, CI/CD

Abstract

Author(s)	Type of Publication	Published
Piskonen, Otto	Thesis, UAS	2022
	Number of Pages	
	30	

Title of Publication

Microsoft Dynamics build process automatization

Degree and field of study

Bachelor of Engineering, Information and communication technologies

Name, title and organisation of the client (if the thesis work is commissioned by another party)

Crosskey Banking Solutions Ab Ltd

Abstract

The thesis examines how Microsoft Dynamics platforms day to day build process was automated. The goal was to build proof of concept for Crosskey. The automatization pipeline covered version control, unit testing and environment updating.

The purpose was to examine what CI/CD is and how its agile methods could be applied to automatization. The thesis will also examine the Microsoft Dynamics platform and what was the best way to implement automatization pipeline. The pipeline was built to Azure DevOps platform.

The automatization pipeline was built using open-source software library called xRM CI Framework. Microsoft Dynamics environment was automated programmatically using the library. The pipeline was integrated with Bitbucket so source code could be used in the pipeline.

The final proof of concept is working as intended. The pipeline uses agile methods from CI/CD and is built to be configurable. The pipeline is a good starting point for possible future development for automatization.

Keywords

Azure DevOps, Dynamics CRM, CI/CD

Sisällys

1 Johdanto

Ohjelmointikehityksessä monet työnkuvaan liittyvät tehtävät vievät paljon aikaa. Nämä tehtävät ovat usein itseään toistavia ja liittyvät koodin versionhallintaan tai sen viemiseen testattavaksi eri ympäristöihin (Crowdbotics 2022). Suurin osa toistuvista tehtävistä on mahdollista automatisoida erilaisilla menetelmillä.

Crosskey on yksi suurimmista Suomessa vaikuttavista finanssialan yrityksistä, joka keskittyy ohjelmistojen suunnitteluun ja valmistukseen (Fintech Farm 2021). Crosskey kuuluu Ålandsbanken konserniin ja on sen tytäryhtiö. Ålandsbanken on Ahvenanmaalta toimiva finanssialan konserni, joka tarjoaa asiakkailleen pankki- ja varainhoitopalvelujen lisäksi it-palveluita. (Ålandsbanken 2022.) Crosskey osti vuonna 2019 IT-alan yrityksen ModelIT:n, joka kehitti OneFactor-varainhoitojärjestelmää. Crosskey jatkoi OneFactor-järjestelmän kehittämistä yhdistymisen jälkeen. OneFactor on tuotettu Microsoft Dynamics-alustalle ja sitä on muokattu toimivaksi asiakkaiden toiveiden mukaisesti. (Crosskey 2019.) OneFactor-varainhoitojärjestelmä tarvitsi kehitysympäristön automatisointia. Versionhallinta, yksikkötestaaminen ja ympäristön päivittäminen ovat kriittisimmät kohdat automatisoinnille.

Opinnäytetyön tavoitteena on tuottaa OneFactor-alustalle toimiva CI/CD-putki käyttäen moderneja DevOps-skenaarioita. Tämä CI/CD-implementaatio vapauttaa kehittäjän manuaalisista toistuvista tehtävistä, jotka vievät työpäivästä suuren osan. Toimiva CI/CD-putki tarvitsee monipuoliset yksikkötestaukset ja yhteensopivuustarkistukset, sekä automaattisen ympäristöjen päivittämisen ja sen versionhallinnoinnin (Tirupati 2020). Opinnäytetyössä tarkastellaan myös CI/CD-menetelmiä ja tutkitaan, miten niitä voidaan käyttää automaatioputkiston luomisessa OneFactor-alustalle.

Microsoftin Dynamics on asiakkuudenhallintaan tarkoitettu sovelluspaketti, jota voidaan muokata omatekoisten laajennusten avulla. Laajennukset tuotetaan käyttämällä Microsoftin palvelupaketteja ja C#-kieltä. (Microsoft 2022a.) Uusien laajennusten luominen tai jo olemassa olevien laajennusten muokkaaminen vaatii niiden päivittämistä Dynamics-ympäristöön, mikä tehdään manuaalisesti käyttäen Microsoftin omaa työkalua. Laajennuksen päivittämiseen kuluu paljon aikaa. Päivittäminen voitaisiin automatisoida, mikä vapauttaisi kehittäjältä paljon aikaa työpäivästä.

2 OneFactor-toimitus

2.1 Microsoft Dynamics

Microsoft Dynamics -alusta mahdollistaa datavetoisten ratkaisujen luomisen asiakkuudenhallinnan tarpeisiin. Microsoft Dynamicsia voi isännöidä paikallisilla palvelimia tai käyttämällä Microsoftin pilvipalvelua. Ympäristöihin voidaan ottaa yhteys verkkoselaimella, joka avaa käyttöliittymän. Ympäristöt toimivat verkkosovelluksien tavoin (Microsoft a). Ympäristöt tarvitsevat toimiakseen palvelimen, johon yhteys voidaan ottaa sekä tietokannan, jonne kaikki Dynamicsin entiteetit ja toiminnot tallentuvat. Ympäristöissä on sisäänrakennettuja ominaisuuksia kuten käyttäjäntodentaminen, liiketoimintadatan entiteetit ja valmis käyttöliittymä, jotka ovat heti käytettävissä (Microsoft b). Ympäristöjä voidaan muokata eri tarkoituksiin integroiduilla ratkaisuilla (engl. solution). Ratkaisut ylläpitävät sen toiminnallisuuden entiteettirakenteet, kustomoidut näkymät ja muut riippuvuudet. Ratkaisuja on mahdollista viedä ja tuoda eri ympäristöistä toiseen, mutta ratkaisut saattavat olla riippuvaisia toisistaan. Tämä saattaa aiheuttaa joissain tilanteissa yhteensopivuusongelmia. Ratkaisut voivat olla unmanaged tai managed tilassa. Unmanaged ratkaisu on perusarvo ratkaisulle. Ratkaisu on vielä kehityksessä ja siihen tai sen entiteetteihin voidaan tehdä muutoksia käyttöliittymästä. Managed ratkaisua käytetään, kun ratkaisu halutaan viedä eteenpäin ja se on valmis. Ratkaisu voidaan viedä ympäristöstä joko unmanaged tai managed tilassa. Vieminen ja tuominen nopeuttaa ominaisuuksien testaamista, koska uusien muutosten rekisteröiminen täytyy tehdä vain yhteen ympäristöön. (Microsoft 2022b.).

Muutokset ja kustomoidut toiminnot tuotetaan C#-kielellä ja niistä rakennetaan dynamic link library eli dll tiedostopääte tiedosto. Dynamic link library tiedosto on Microsoftin luoma tiedostorakenne Windows-käyttöjärjestelmälle. Näitä tiedostoja kutsutaan assemblyiksi Microsoft Dynamics kontekstissa. Assemblyt pitävät sisällään kaikki luokat ja niiden riippuvuudet. (Microsoft 2022c.) Näitä tiedostoja voidaan rekisteröidä Dynamics-ympäristöön käyttämällä Microsoftin tarjoamaa työkalua. Dynamics hyväksyy kahta erityyppistä toiminnallisuutta, jotka ovat: Workflow Activity tai Plugin. Workflow Activity mahdollistaa toimintojen suorittamisen entiteeteistä tai batch jobeista. Plugin suorittaa toimintoja automaattisesti, jos tietokannassa tapahtuu muutoksia. Microsoft tarjoaa tarvittavat toiminnot molemmille tyypeille omilla luokillaan, mitkä periytetään oman kustomoidun toiminnon luokkaan. Periytetyt luokat tarvitsevat Microsoftin tarjoaman ohjelmistokehityspaketin, josta löytyy kaikki tarvittavat ominaisuudet kustomoitujen Pluginien ja Workflow Activityjen luomiseen (Microsoft 2022d). Rekisteröintityökalu lataa ja tekee tarvittavat merkinnät assemblystä ympäristön tietokantaan. Rekisteröinnin jälkeen toiminnallisuudet tai niiden muutokset ovat

käytettävissä Dynamicsissa prosesseina. Assemblyjä voidaan lisätä ympäristön ratkaisuihin, jotta niitä voidaan viedä muihin ympäristöihin ilman rekisteröintiä.

Entiteettejä käytetään liiketoimintadatan mallintamiseen ja hallinnoimiseen. Entiteetit, kuten kaikki Dynamics-ympäristössä käytettävät oliot, voidaan tallentaa ratkaisuihin. Entiteetit toimivat pääsääntöisinä olioina, mitä voidaan manipuloida toiminnoissa. Entiteettien datan manipuloimiseen käytetään Microsoftin tarjoamaa ohjelmistokehityspakettia, jossa on erilaisia valmiita komentoja. (Microsoft 2022e.) Entiteettien dataa voidaan myös muokata suoraan käyttöliittymästä. Muokkaus oikeuksia voidaan myös asettaa käyttäjätasolla, mikä mahdollistaa ympäristössä toteutettavan käyttäjähierarkian (Microsoft 2022f). Entiteetit perivät itselleen näkymän, jotka määrittävät, mitä kaikkea tietoa käyttöliittymässä näytetään. Näkymiin voidaan lisätä jo valmiita kenttiä tai luoda uusia kustomoituja kenttiä. Näkymiin voidaan myös upottaa JavaScript tiedosto, jolla voidaan manipuloida näkymän kenttiä vaativammalla logiikalla.

2.2 Toimintaympäristö

Microsoft Dynamics -ympäristö sisältää kaikki siihen luodut laajennukset eri ratkaisujen sisällä. Ratkaisuja voidaan kuvitella eräänlaiseksi hakemistoiksi, joihin tallennetaan kustomoituja näkymiä, entiteettejä ja laajennuksia. Ratkaisuja voidaan tuoda ja viedä ympäristöstä toiseen, mutta tämä saattaa aiheuttaa yhteensopivuusongelmia. Yhteensopivuusongelmat johtuvat yleisesti koodissa kutsutuista entiteeteistä, joita ei ole lisätty ympäristöön. Yhteensopivuusongelmia voidaan kuitenkin minimoida ylläpitämällä tiukkaa versionhallintaa kehityksessä. (Microsoft 2022g.)

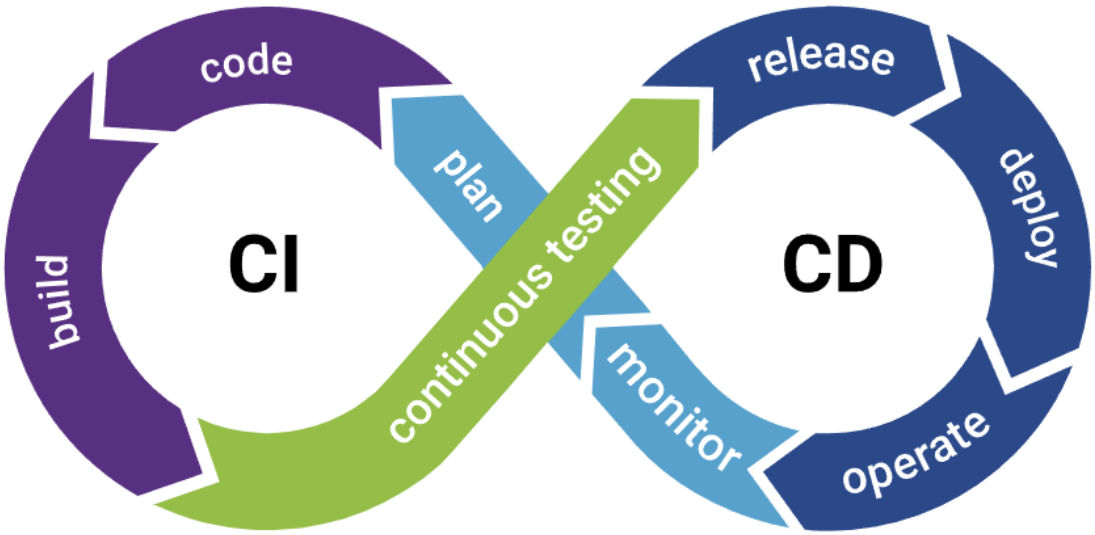
Kehitettävää koodikantaa ylläpidetään yleisessä kehitysympäristössä, mihin muutoksia ja uusia laajennuksia on helppo lisätä ilman yhteensopivuusongelmia. Tuotanto on jaettu inkrementteihin ja aktiivisen inkrementin lähdekoodin ylläpidetään kehitysympäristössä. Muutokset viedään kehitysympäristöstä testiympäristöihin, joista niitä on helpompi testaajien testata, koska testiympäristöihin on luotu riittävä määrä dataa. Hyväksymistestaaminen voidaan myös suorittaa omassa ympäristössä ja sieltä siirtää ratkaisut tuotantoon, kun ne ovat valmiita. (Piskonen 2022; Rantanen 2022.)

Muutosten rekisteröiminen ratkaisuihin ja sen vieminen eteenpäin on yksi manuaalisista tehtävistä, joita kehittäjän tarvitsee tehdä melkein päivittäin. Tämä koko prosessi voidaan automatisoida. Koodikannassa tehdyt muutokset voivat käynnistää automatisointiputkia, jotka suorittavat muutosten rekisteröinnin, ratkaisujen versionhallinnan ja niiden viemisen oikeaan ympäristöön testattavaksi. Samalla automatisointi vapauttaa kehittäjän aikaa. (Piskonen 2022.)

3 Continuous Integration/Continuous Deployment

3.1 CI/CD-Teoria

CI/CD on lyhenne sanoista ”Continuous Integration/Continuous Deployment” ja on sovelluskehityksessä käytettävä käytäntö toimituksen automatisointiin. Sovelluskehityksen manuaaliset työt, kuten versionhallinnasta koodin testaaminen ja sen käyttöönotto, voidaan haluta automatisoida. Jatkuvassa kehityksessä koodia muokataan, siihen suoritetaan yksikkötestit ja se voidaan päivittää testiympäristöön. Tämän prosessin automatisointi voi nopeuttaa kehittämistyötä ja helpottaa koodin ylläpitoa. Ilman CI/CD-putkistoa suuret projektit voivat helposti päätyä ”integrointi helvettiin”, joka pysäyttää kehityksen kokonaan (RedHat 2018). ”Integrointi helvetti” pysäyttää koko kehitystiimin työskentelyn, koska muutokset, mitkä on viety kehityshaaraan rikkovat ohjelman toiminnallisuuden. Tämä voi vaikuttaa koko projektin kulkuun, jos muutoksia minkä päälle on tehty lisää muutoksia, täytyy palauttaa (Apica 2016). Jokaisen projektin tarpeet ovat erilaiset ja myös siihen tarvittavat putkistot peilaavat sitä. CI/CD-putkistossa käytettävät menetelmät toimivat enemmän suuntaviivana, mitä kehittäjien ja muiden tiiminjäsenien pitäisi seurata. Automatisoidun putkiston tuottaminen edellyttää sen sääntöjen seuraamista, jos jatkuvasta kehittämisestä halutaan saada kaikki irti. CI/CD-filosofia ei toimi kaikissa projekteissa ja sen edut tulevat paremmin esille mitä isompi projekti on (InfoWorld 2022). Kuvassa 1 esimerkki CI/CD-putken toimintaperiaatteista.



Kuva 1. CI/CD-putken toimintaperiaate (Synopsys 2022)

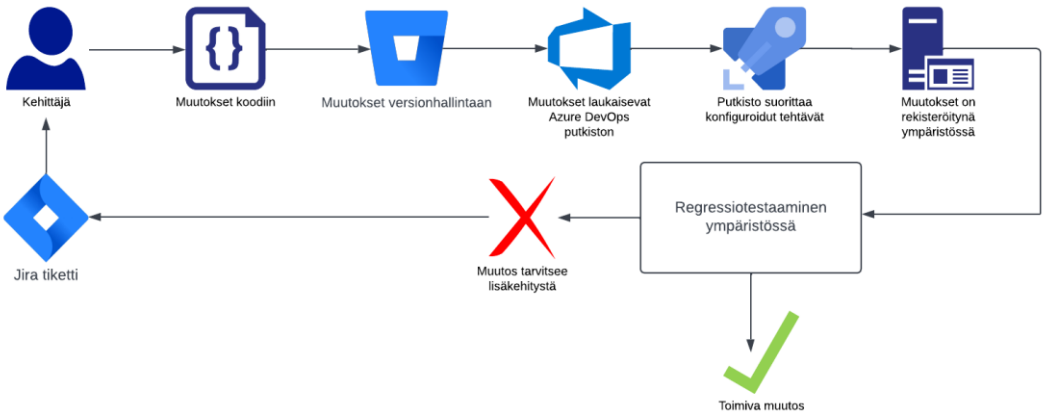
Ihanteellisessa CI/CD-putkistossa kaikki tarvittavat tehtävät on jaettu vaiheisiin. Vaiheet suorittavat oman osansa putkesta ja voivat aktivoida seuraavia vaiheita automaattisesti. Erilaisia vaiheita voi olla esimerkiksi: regressiotestien suorittaminen, koodin kääntäminen ja sen päivittäminen testiympäristöön. Jaetut vaiheet voidaan ohittaa tai niiden ulostuloja voidaan muokata tarvittaessa. Vaiheisiin jakaminen helpottaa putkiston toimintaa ja sen muokkaamisesta tulee helppoa. Toimivan putkiston luominen vaatii paljon etukäteistä työtä, kuten yksikkötestien ja automatisoinnin luomista. Oikein tehtynä putkisto ei tarvitse paljon ylläpitoa tulevaisuudessa.

3.2 Azure DevOps

Azure DevOps on Microsoftin tarjoama verkkosovellusalusta, joka tarjoaa ohjelmisto työkalut yhteiseen kehittämistyöhön. Azure DevOps -alustalla tiimit voivat ylläpitää tuotannon toimintaympäristöjä, automatisoida toimituksia, testata kehitettäviä sovelluksia ja ylläpitää versionhallintaa. (Microsoft c.) Azure DevOps -rakenne perustuu organisaatioihin, joihin lisätään käyttäjiä ja heille oikeuksia. Organisaatiolla voi olla monta Azure DevOps -projektia ja yksi projekti pitää sisällään sen Azure DevOps -ominaisuudet kuten tehtävätaulu sekä putkisto. Microsoft kutsuu tuotannon toimintaympäristöään Azure Boardiksi ja siitä löytyy sisäänrakennettuna ominaisuuksia tuotannon ajoitusmenetelmiin (Microsoft 2022h). Azure Boardista löytyy ketterät työkalut kuten: Kanban-taulut, backlogit ja scrum-toiminnallisuus, joita kehittäjät käyttävät moderneissa kehitysympäristöissä. Azure Board voidaan myös integroida jo valmiiden sovelluksien kanssa, kuten Github tai Bitbucket. Microsoft tarjoaa myös kustomoituja lisäosia Azure DevOps markkinapaikassa.

Automatisointi voidaan toteuttaa Azure DevOps -alustalla käyttäen Azure Pipeline -toimintoa ja sen tarjoamia työkaluja toimituksen automatisointiin. Azure Pipeline tukee kaikkia suuria kehittämisympäristöjä ja koodikieliä, mikä tekee siitä suositun vaihtoehdon CI/CD-alustaksi. (Codilime 2021; Microsoft 2022i.) Azure Pipelineä voidaan käyttää Azure DevOps -alustan versionhallinnan kanssa tai se on mahdollista integroida kolmannen osapuolien sovellusten kanssa. Azure Pipeline tarvitsee toimiakseen palvelimen, joka voi olla paikallinen tai Microsoftilta vuokrattu. Kaikki automatisointiputkiston toiminnallisuudet suoritetaan valitulla palvelimella, mikä mahdollistaa myös sen skaalattavuuden (Microsoft 2022j). Automatisoitu putkisto voi koota lähdekoodin versionhallinnasta ja käynnistää sille yksikkötestejä. Putkisto voidaan konfiguroida suorittamaan eri toimintoja tiettyjen parametrien perusteella ja automatisointi voidaan keskeyttää, jos jokin askel jäi kesken. Azure Pipeline -työt voidaan integroida Microsoft DevOps markkinapaikasta löytyvien lisäosien kanssa. Tämä mahdollistaa erillisten kolmannen osapuolen sovelluksien käytön automatisoinnissa.

Esimerkki Azure DevOpsilla luodusta automaatiosta voidaan nähdä kuvion 1 prosessikaaviossa. Kuvion lähtökohta on Jira tiketti, josta kehittäjän tekemät muutokset johtavat automatisointiputkistoon.



Kuvio 1. Automaation prosessikaavio

Azure Pipeline luodaan selaimen käyttöliittymästä ja yksi Pipeline suorittaa sille konfiguroidut tehtävät, joita kutsutaan askeleiksi. Askeleet voivat olla yksikkötestejä, koodin kääntämistä tai muutosten rekisteröintiä ympäristöihin. Pipelinen tehtäviä voidaan konfiguroida käyttämällä Azure DevOps käyttöliittymää tai versionhallintaan tallennettavaa YAML-tiedostoa. YAML-tiedostot pitävät sisällään YAML-formaatilla kirjoitettua logiikkaa, joka kertoo Azure Pipelinelle, mitä tehdä (YAML 2022). Azure Pipeline luo käyttöliittymässä konfiguroiduista putkista valmiin YAML-tiedoston, joka voidaan viedä käyttöliittymästä toiseen palveluun. Putkistoja voidaan suorittaa sarja-ajona, joka mahdollistaa monimutkaisten, toisistaan riippuvaisten, putkistojen luomisen. Sarja-ajo mahdollistaa modulaarisen jatkuvan integraatioputkiston, jossa putkiston kohtia voidaan ohittaa tai olla suorittamatta ollenkaan (Youtube 2022). Azure Pipelinen suorittamia tehtäviä ja niiden kulkua voidaan seurata käyttöliittymästä. Putkiston tehtävistä on tarjolla tiivistelmäsivu, mihin mahdolliset virheet tai ongelmat tulostetaan. Tiivistelmästä voidaan avata tarkempi seurantasivu, jossa on palvelimesta peilattu komentokehote. Komentokehotteeseen tulostuu toimintojen kulku ja siihen voidaan aktivoida system debug -tuloste lisäominaisuutena. Näin on mahdollista nähdä lisää palvelimen tapahtumia. Putkistoa voidaan konfiguroida käyttäen Azure DevOpsin tarjoamia valmiita muuttujia. Valmiisiin muuttujiin tallentuu taustalla tapahtuvien toimintojen kuten kääntämis-, putkisto- tai systeemitason tehtävien arvoja. Valmiit muuttujat ovat automaattisesti käytettävissä putkistossa ja joidenkin muuttujien arvot muuttuvat dynaamisesti putkiston suorituksen aikana (Microsoft 2022m).

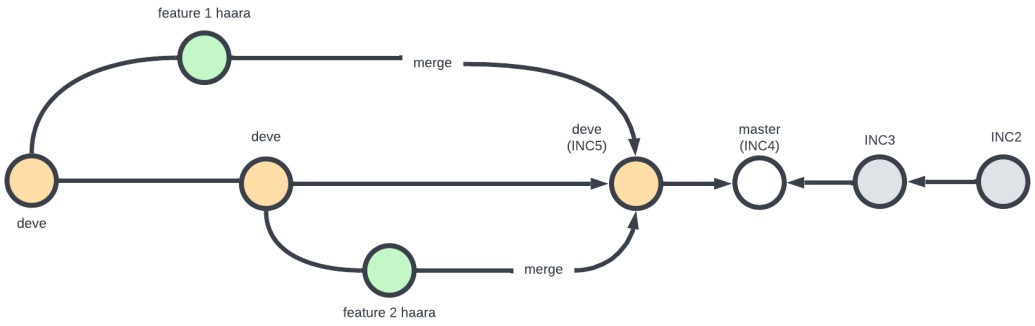
Azure DevOps tarjoaa testaamiseen tarkoitetut työkalut. Test Plans välilehdestä löytyvät työkalut mahdollistavat yksikkötestien suorittamiseen laadittavan suunnitelman tekemisen ja yksittäisten testitehtävien seuraamisen. Putkistossa tapahtuvat testitehtävät luovat raportin, josta ilmenee oleelliset tiedot testatuille testeille. Raportin avulla voidaan seurata, kuinka monta testiä suoritettiin ja jos suoritetuista tehtävistä jotkut epäonnistuivat. Epäonnistuneista testeistä saatavat stack trace ja error message -viestit auttavat kehittäjää löytämään ongelman.

4 Toteutus

4.1 OneFactorin tuotanto

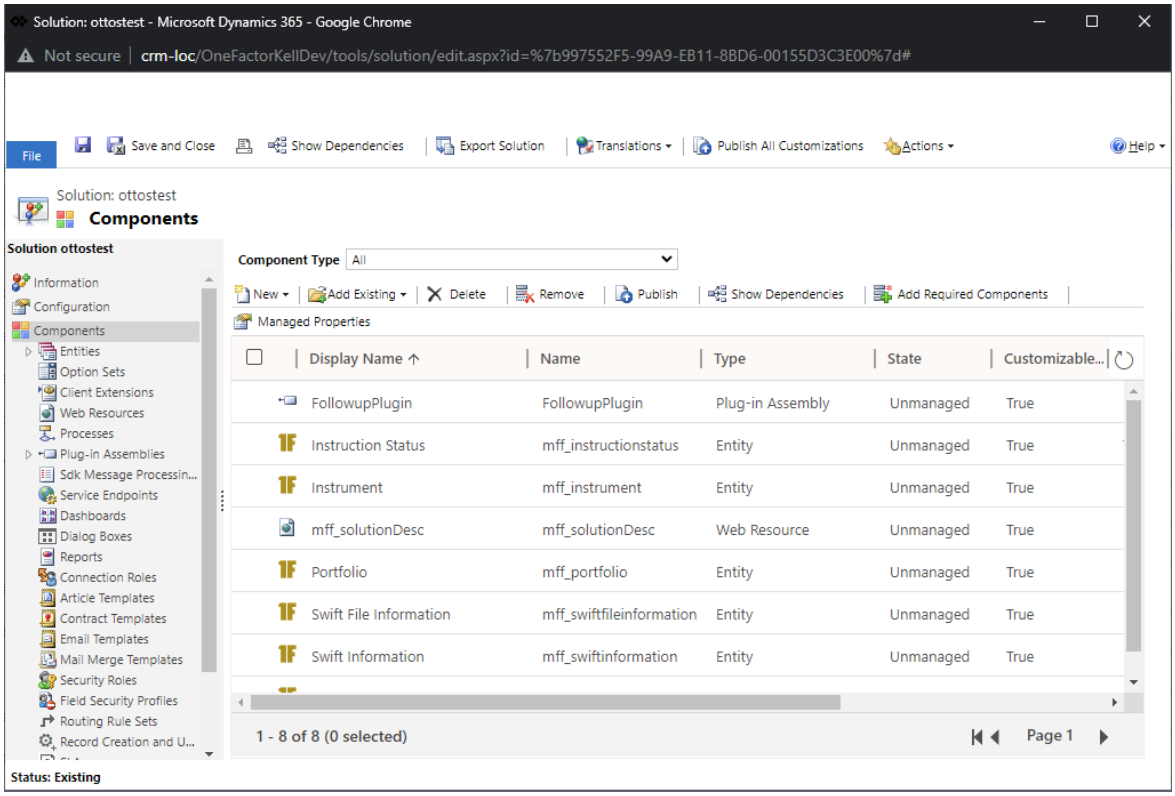
Finanssialan yritys Crosskey tarvitsi varainhoitojärjestelmänsä OneFactorin toimituksen automatisointia. Automatisointi tehtiin helpottamaan kehittäjien päivittäisiä toistuvia työtehtäviä. OneFactor on toteutettu Microsoft Dynamics -alustalle ja Crosskey tarjoaa sitä varainhoitojärjestelmäpakettina asiakkailleen. OneFactoria voidaan muokata asiakkaan tarpeiden mukaisesti ja tehdyt muokkaukset voidaan lisätä olemassa olevaan OneFactor-järjestelmään. Pienet asiakkaat eivät tarvitse muutoksia OneFactorin monipuolisuuden takia, mutta isot asiakkaat voivat tarvita ominaisuuksia, jotka toimivat juuri heidän teknisten spesifikaatioiden mukaan.

OneFactor-tuotanto on jaettu inkrementteihin. Vuodessa inkrementtejä on viisi kappaletta ja viimeinen inkrementti jatkuu viikon ajan seuraavan vuoden tammikuulle. Inkrementin päätyttyä lähdekoodi jäädytetään ja se siirretään kehityshaarasta, master-haaraan. Master-haarassa pidetään aina edesmenneen inkrementin lähdekoodi. Kuviossa 2 on esitetty esimerkki Git-rakenteesta.



Kuvio 2. Esimerkki Git-rakenteesta

Jokaiselle päättyvälle inkrementille tehdään oma inkrementti-haara, mihin master-haara yhdistetään. Inkrementin aikana OneFactoria voidaan tuottaa monen asiakkaan teknisten spesifikaatioiden perusteella saman aikaisesti. Kehitystä tehdään inkrementin aikana kehityshaaraan, josta muutokset rekisteröidään kehityshaaralle tehtyyn ympäristöön. Kehitysympäristön tarkoitus on ylläpitää tuoreinta versiota lähdekoodista ja sinne on myös helpoin tehdä käyttöliittymä muutokset (Rantanen 2022). Rekisteröidyt muutokset tallentuvat OneFactorin käyttämiin ratkaisuihin ja ratkaisuja voidaan viedä eteenpäin testiympäristöihin. Kuvassa 2 on esitetty käyttöliittymän ratkaisuikkunan perusnäkymä.



Kuva 2. Ratkaisun perusnäkymä

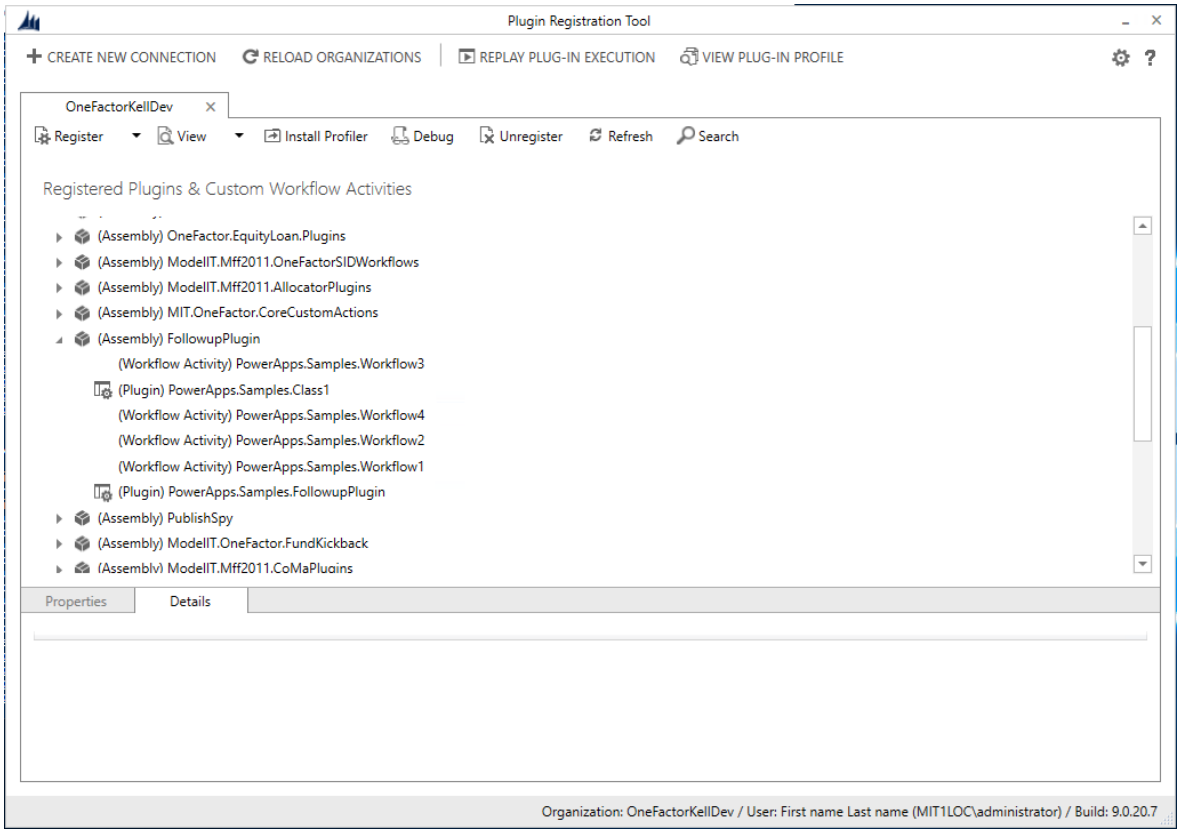
Asiakkaalla voi olla monta testiympäristöä, mutta yleisesti seurataan ketteriä menetelmiä ja jokainen testivaihe on oma ympäristönsä. Lähdekoodin jäädytyksen jälkeen tulevia muutoksia ei kuitenkaan enää kehitetä kehityshaaraan vaan erilliseen ylläpitohaaraan, joka tehdään asiakkaalle tarvittaessa. Asiakkaan testiympäristöt saavat muutoksensa tästä ylläpitohaarasta, mikä vähentää yhteensopivuusongelmia. Pienet korjaustyöt voidaan kuitenkin suorittaa inkrementille tehtyyn haaraan.

Päivittäisessä kehitystyössä kehittäjä tekee muutokset lähdekoodiin, kääntää koodin, rekisteröi muutokset kehitysympäristöön, ylläpitää versionhallintaa OneFactorin ratkaisuissa joihin muutokset on tehty ja, vie muutetun ratkaisun eteenpäin testattavaksi. Tässä opinnäytetyössä tämä prosessi automatisoidaan käyttäen Microsoftin Azure DevOpsia ja Dynamics CRM SDK -paketteja. OneFactor-toimituksessa ei käytetä automatisointia. Ainoa automatisointi, jota OneFactorin tuotannossa käytetään, on pohjaprojektin, johon on sisälletty tarvittavat Dynamics CRM SDK:n toiminnallisuudet, rakentaminen ja NuGettien päivittäminen palvelimelle, josta niitä voidaan käyttää muissa projekteissa. Tämä tehdään Jenkins-palvelun avulla. Automatisoinnilla tulisi ratkaista muutosten rekisteröinti, ratkaisujen versionhallinta sekä niiden vieminen testiympäristöön. Automatisoinnin olisi hyvä olla myös

modulaarinen, jotta sen vaiheita voitaisiin peruuttaa tai ohittaa. Tuotettu automatisointiputkisto on soveltuvuusselvitys automatisoinnin tarpeista toimitukseen. Työn yhteydessä laadittiin myös sääntöjä päivittäiseen toimitukseen, mitkä auttaisivat automatisoinnin luonnissa.

4.2 Ympäristön päivittämisen riippuvuudet

OneFactorin automatisointiputkisto rakennettiin Azure DevOps -alustalle, koska Microsoftin tuotteet toimivat keskenään sulavasti. Microsoft Dynamics -assemblyt rekisteröidään manuaalisesti käyttäen Microsoftin omaa Plugin Registration -työkalua, mutta Microsoft mahdollistaa rekisteröinnin ohjelmallisesti käyttäen Dynamics CRM SDK:ta, jolla voidaan myös manipuloida ympäristön entiteettejä koodissa. Kuvassa 3 näkyy Microsoftin tarjoama Plugin Registration -työkaluohjelma.



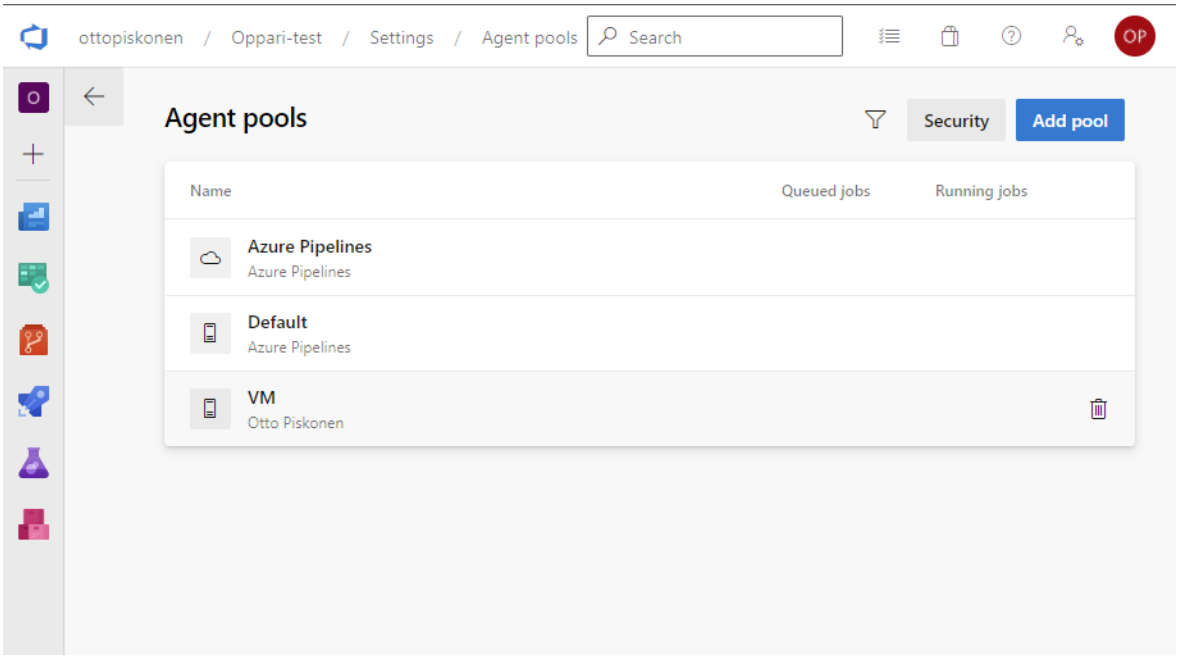
Kuva 3. Plugin Registration -työkalu

Rekisteröinti on kaikessa yksinkertaisuudessaan kootun koodin luoman dynamic link library tiedoston tallentamista tietokantaan. Microsoft DevOps -markettipaikasta löytyy käyttäjän Wael Hamzen luoma xRM CI Framework kirjasto, johon on rakennettu Dynamics CRM

SDK:ta käyttäen valmiita skriptejä, jotka suorittavat SDK-komentoja. xRM CI Framework on avointa lähdekoodia ja käyttää MIT License -ohjelmistolisenssiä. OneFactoriin tuotettiin automatisointiputkisto xRM CI Frameworkin avulla. xRM CI Frameworkista löytyy valmiita Azure DevOps -tehtäviä, joita voidaan lisätä suoraan käyttöliittymästä. Enemmän xRM CI Frameworkin ominaisuuksia voi ladata PowerShell Galleriasta, joka on Microsoftin ylläpitämä PowerShell-ohjelmistopakettien säilytyspaikka. PowerShell Galleriasta ladatut paketit ja niiden sisältämiä ominaisuuksia voidaan suorittaa suoraan PowerShellista. Azure DevOpsista putkistossa voidaan suorittaa kustomoituja skriptejä ja valmiita Azure DevOps toiminnallisuuksia sekaisin. Kustomoidut skriptit suoritetaan palvelimen PowerShellista, jossa putkistoa suoritetaan.

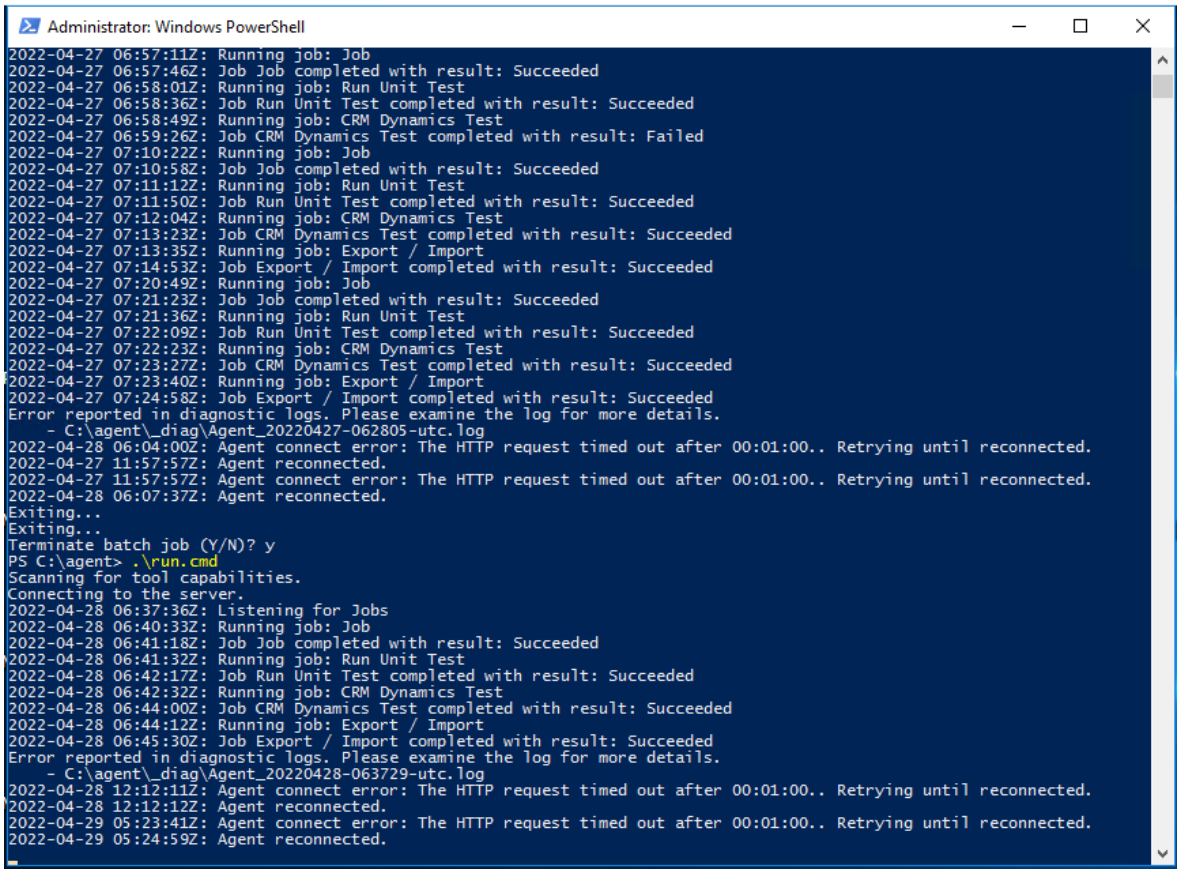
Asennusvaiheessa on hyvä tarkistaa Microsoft Dynamics -alustan ja xRM CI Frameworkin versiot. Uusin xRM CI Framework tarvitsee toimiakseen Microsoft Dynamics CRM SDK:sta version 9 tai sitä uudemman. Tämä SDK versio pitää olla asennettuna palvelimelle, josta toimintoja suoritetaan. Jos toimintoja suoritetaan palvelimelta, jossa ylläpidetään Dynamics-ympäristöä, täytyy tarkistaa, mitä versiota Dynamics CRM SDK:sta on käytetty yhteensopivuuden säilyttämiseksi. Palvelimeen, jossa tehtävät suoritetaan, asennettiin PowerShell Galleriasta xRM CI Framework. Tehtävät käyttävät markkinapaikasta asennettua ohjelmistopakettia ja kustomoituja skriptejä, jotka käyttävät PowerShell Galleriasta ladattua versiota.

Azure DevOpsista voidaan vuokrata lisämaksua vastaan palvelin, joka suorittaa putkiston tehtäviä. Tehtävien suorittamiseen voidaan myös käyttää omia palvelimia asentamalla Windows Agentti. Microsoft kutsuu ohjelmistoa, joka suorittaa tehtäviä palvelimessa, Agentiksi. Yksi Agentti suorittaa yhtä tehtävää samanaikaisesti, joten jos automatisointiputkisto on monimutkainen tai tarvitsee rinnakkaista ajoa, tarvitaan useampi Agentti. Rinnakkainen ajo voidaan valita jokaiselle tehtävälle erikseen. Useita Agentteja voidaan luoda Agenttipooliin, joka voidaan valita putkiston konfiguraatiossa suorittimeksi. Kuvassa 4 näkyy Azure DevOps käyttöliittymässä olevia Agenttipooleja.



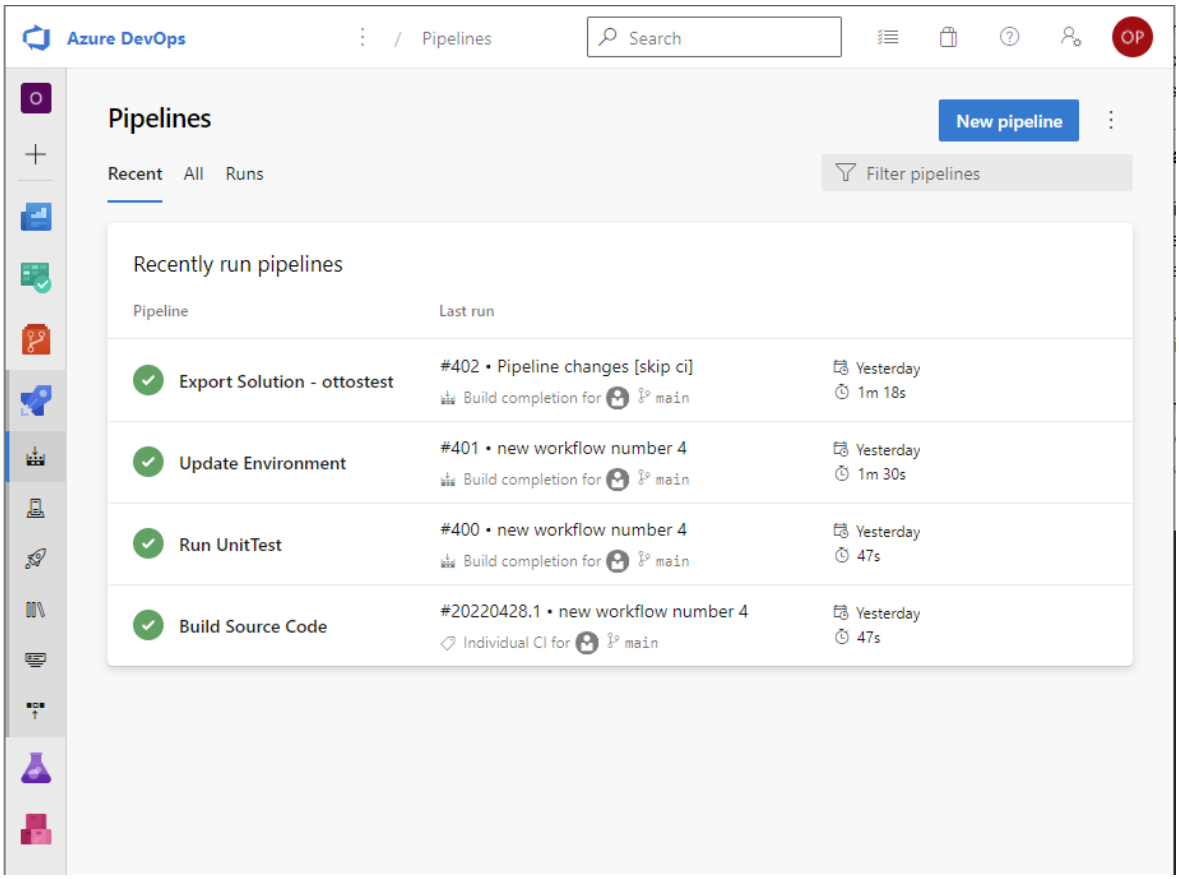
Kuva 4. Azure DevOps -Agenttipooli

Agentin asennus omalle palvelimelle tehdään käyttöliittymästä. Asennuksen aikana käyttöliittymä ohjeistaa Agentin lataamisen ja sen alustamisen palvelimeen. Ennen tehtävien suorittamista Agentti täytyy aktivoida PowerShellista, jonka jälkeen se kuuntelee tulevia tehtäviä automaattisesti. Aktivointi tapahtuu Agentin hakemistosta suorittamalla run.cmd skripti. Kuvassa 5 näkyy PowerShell-ikkuna, jossa suoritetaan Agenttia.



Kuva 5. PowerShell-ikkuna, jossa suoritetaan Agenttia

OneFactorin putkistoagentti asennettiin virtuaalikoneelle, jossa käyttöjärjestelmänä toimi Windows Server 2016. Virtuaalikoneella ylläpidetään myös Dynamics CRM -ympäristöä, jota käytetään kehittäjän omana ympäristönä. Agentille täytyy varata tallennustilaa, koska mahdolliset koodin kääntämiset tapahtuvat palvelimessa ja koodi pitää tuoda sinne versionhallinnasta. Jotkut putkiston tehtävät tarvitsevat myös lähdekoodia suorittamaan askeleitaan, joten niistä haetaan lähdekoodi komennoilla Git-checkout ja Git-fetch. OneFactorin automatisointiputki on jaettu neljään erilliseen tehtävään, jotka suoritetaan sarjana. Tehtävät aktivoituvat vasta, kun edellinen tehtävä on suoriutunut onnistuneesti. Erilliset tehtävät voidaan myös halutessa deaktivoida, jos niitä ei haluta suorittaa. Kuvassa 6 nähdään Azure DevOpsin käyttöliittymä putkistolle.



Kuva 6. Azure Pipeline käyttöliittymä

4.3 Koodin kääntäminen

Ensimmäinen tehtävä OneFactorin automatisointiputkistossa on koodin kääntäminen. Azure Pipeline -tehtävä, joka kääntää koodin, on konfiguroitu käyttämään MSBuild-alustaa. MSBuild on Microsoftin tarjoama ohjelmistomoottori, jolla voidaan kääntää Visual Studio -projekteja, jotta saadaan ympäristöjen päivittämiseen tarvittavat assemblyt. MSBuild voidaan asentaa suoraan järjestelmään asentamalla .NET SDK-ohjelmistopaketti tai asentamalla Visual Studio -ohjelmistokehitystyökalu, joka sisältää .NET SDK:n. Kääntötehtävä käyttää Microsoftin valmiita Azure DevOps -komentoja: NuGetToolInstaller, NuGetCommand, VSBuild ja PublishBuildArtifacts. NuGetToolInstaller hakee järjestelmään asennetun NuGet ohjelmistopakettiasennustyökalun ja asentaa sen, jos sitä ei ole vielä järjestelmässä (Microsoft 2022k). Tätä tarvitaan kääntöprosessissa, jotta assemblyn kaikki riippuvuudet ja paketit voidaan ladata ohjelmistokirjastoista. NuGetCommand hakee annetusta projektipolusta package.config tiedoston (Microsoft 2022l). Package.config -tiedostoon on kirjattu kaikki ohjelmistopaketit, joita projekti tarvitsee toimiakseen oikein. .NET Framework projektiin asennettavat ohjelmistopaketit asennetaan käyttämällä NuGet kirjastoa. NuGetteja

voidaan myös asentaa omasta palvelimesta, koska NuGetit voivat olla omia ohjelmistopaketteja. Löydettyään package.config tiedoston, NuGetCommand asentaa siitä löytyvät ohjelmistopaketit.

VSBuild-komento käyttää MSBuild-ohjelmistomoottoria ja kääntää annetusta polusta projektin ja luo siitä assemblyn eli dynamic link library -tiedoston. Microsoft suosittelee käyttämään VSBuild-ominaisuutta MSBuild-ominaisuuden sijaan, jos käännettävä koodi on ratkaisu. VSBuild antaa automaattisesti kääntämismoottorille parametrejä, jotka helpottavat kääntämisprosessia. Kääntöprosessin aikana suoritetaan ILMerge-skripti, joka yhdistää OneFactorissa kehitettävien NuGettien ohjelmistopaketit käännettävään assemblyyn. Suoritus on lisätty projektitiedostoon ja se vaatii, että ILMergen-ohjelma on asennettuna palvelimella, jossa kääntöprosessi tapahtuu. Viimeisenä PublishBuildArtifacts toiminto hakee annetusta polusta käännetyn assemblyn ja lataa sen joko ajettavan tehtävän konttiin tai annettuun tiedostopolkuun. Artefakti voi olla mikä tahansa tiedosto ja artefakteja voidaan käyttää myöhemmin putkiston elinkaaren toisissa tehtävissä. Tämä vähentää putkiston kuormaa, koska koodi voidaan kääntää kerran ja käyttää kaikissa putkiston vaiheissa ilman uudelleen kääntämistä. Kuvassa 7 näkyy ensimmäisessä tehtävässä käytettävä YML-tiedosto.

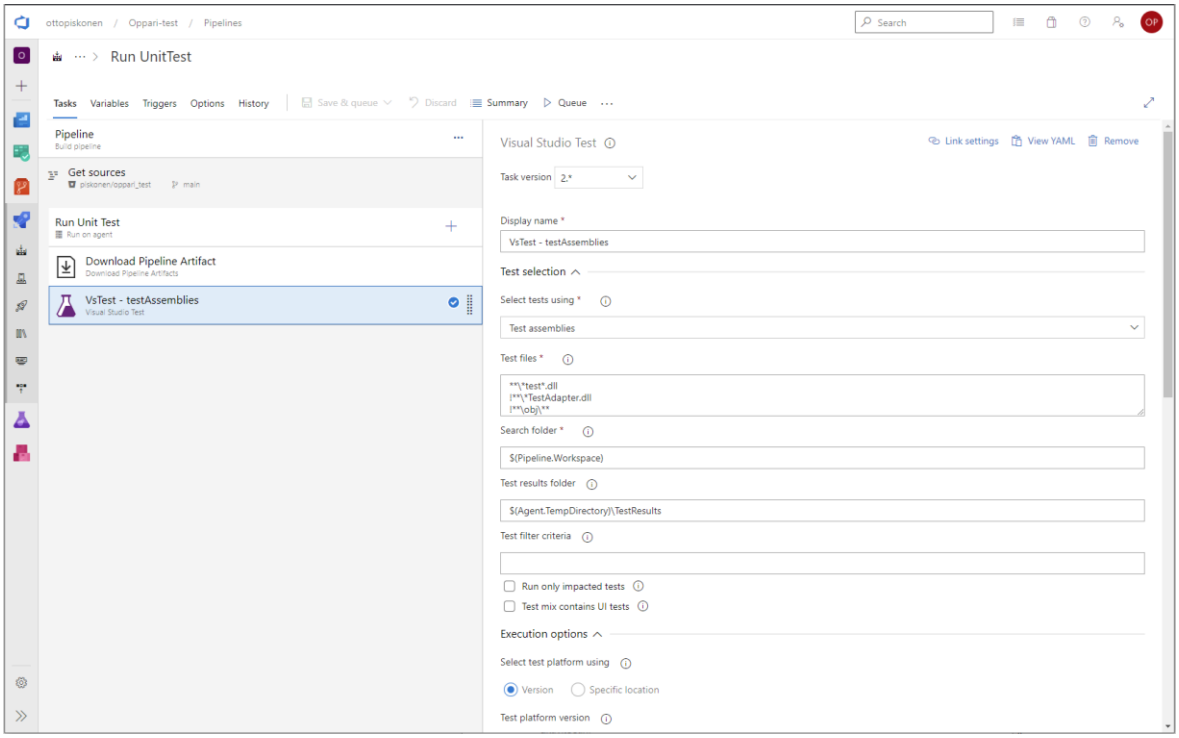
```yaml
# ASP.NET Core (.NET Framework)
# Build and test ASP.NET Core projects targeting the full .NET Framework.
# Add steps that publish symbols, save build artifacts, and more:
# https://docs.microsoft.com/azure/devops/pipelines/languages/dotnet-core

trigger:
- refs/heads/main
pr: none


pool:
  name: 'VM'
  demands:
  - msbuild
  - powershell

variables:
  solution: '**/PowerApps-Samples/cds/orgsvc/C#/FollowupPlugin/*.sln'
  buildPlatform: 'Any CPU'
  buildConfiguration: 'Release'
  artifact: 'PowerApps-Samples/cds/orgsvc/C#/FollowupPlugin/FollowupPlugin/FollowupPlugin.dll'
  testArtifact: 'PowerApps-Samples/cds/orgsvc/C#/FollowupPlugin/UnitTest/bin/Release/net462/UnitTest.dll'

steps:
- task: NuGetToolInstaller@1

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

Kuva 7. Kääntötehtävän YML-tiedoston esimerkki

Tässä putkistossa käännetty assembly tallennetaan sen hetkisen ajon konttiin, josta se haetaan muihin vaiheisiin. Artefakti nimetään tallennusvaiheessa, jotta se on helpompi hakea myöhemmissä tehtävissä. Jos putkistossa tarvitaan useampia artefakteja, täytyy lisätä useampi PublishBuildArtifacts-toiminto putkistoon. Tässä putkistossa käännettiin workflow- ja yksikkötestiprojektit. Näistä käännetyistä projekteista tallennettiin artefakti.

4.4 Testien suorittaminen

Toinen tehtävä putkistossa, mikä aktivoituu kääntötehtävän onnistuttua, on testien suorittaminen. Testien suorittamistehtävä suorittaa projektin yksikkötestit ja OneFactorille tehty kustomoitu API-tason yksikkötestiratkaisun testit. Testitehtävä käyttää Azure DevOpsin valmiita komentoja DownloadPipelineArtifact ja VSTest. DownloadPipelineArtifact -komento hakee joko sen hetkisestä suorituksesta tai aikaisemmasta putkiston tehtävästä luotuja artefakteja nimeltä. Artefakti haetaan aikaisemmasta kääntötehtävästä ja tallennetaan testien testaamistehtävän hakemistoon käyttäen Azure DevOpsin valmista muuttujaa Pipeline.Workspace. Tämä valmis muuttuja tarjoaa putkiston käyttöön putkiston kotihakemiston, jossa sen hetkinen tehtävä suoritetaan ja tarvittavat tiedot säilytetään. Seuraava komento VSTest mahdollistaa yksikkötestien suorittamisen käyttäen Visual Studio Test Runneria. VSTest -komento voidaan konfiguroida suorittamaan yksikkötestit assemblystä, Azure Pipelinessä luodusta testisuunnitelmasta tai luodusta ympäristöstä. Microsoft ei suosittele käyttämään viimeistä vaihtoehtoa suorittamaan CI/CD-automaatioputkistoon tarkoitettuja yksikkötestejä. Tässä putkistossa yksikkötestit suoritetaan käyttämällä artefaktia, joka ladattiin aikaisemmin. Kuvassa 8 nähdään yksikkötestien testaamiseen tarkoitetun tehtävän käyttöliittymäsivu konfiguroinnille.

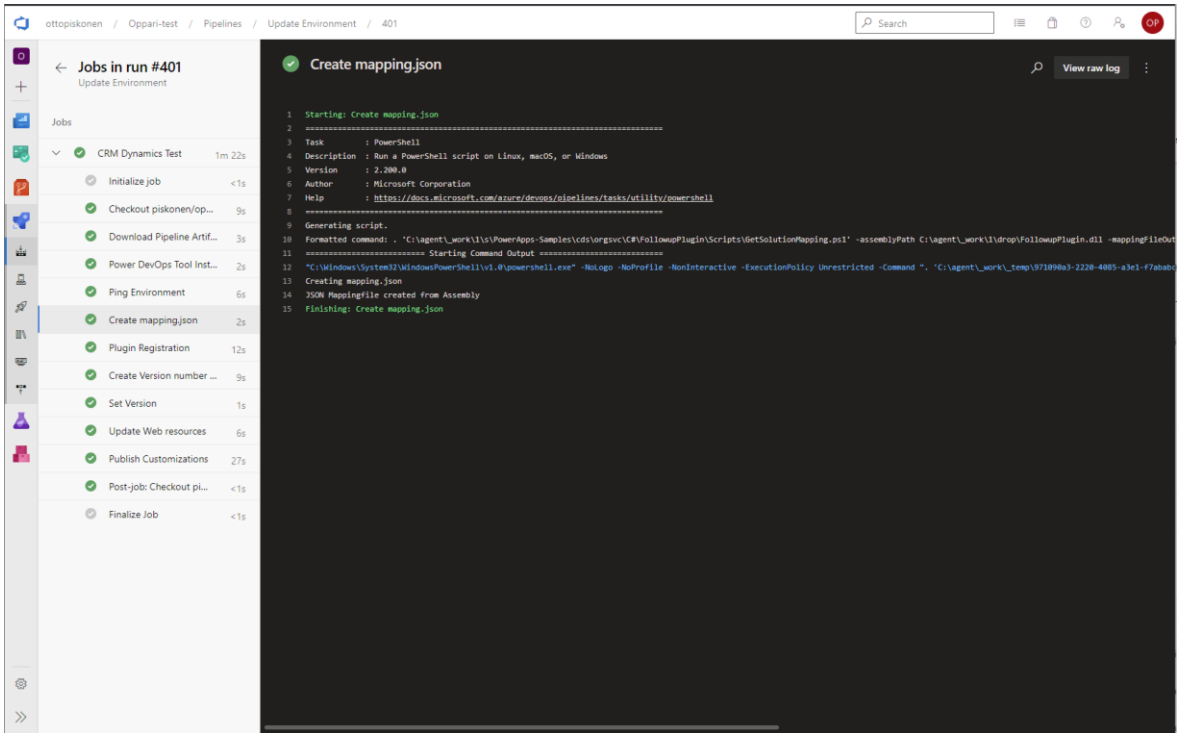


Kuva 8. Yksikkötesti tehtävän konfigurointi sivu

VSTest-toiminto hakee annetusta hakemistosta artefaktin käyttäen minimatch-malleja. Minimatch-malli käyttää erilaisia säännöllisiä lausekkeita hakuehtoina ja sillä voidaan dynaamisesti hakea tiedostoja. Minimatch-malli hakee hakemistosta kaikki tiedostot, jotka päättyvät esimerkiksi samalla tiedostopäätteellä. Käyttämällä valmista muuttujaa Pipeline.Workspace ja VSTestin käyttämää mimimatch mallinnusta voidaan tehtävän hakemistosta hakea kaikki tiedostot, joiden nimessä esiintyy sana ”test”. Voidaan lisätä myös haku vain tiedostoille, jotka ovat dynamic link library -tiedostoja. Tällä tavalla käännetty assembly saadaan haettua yksikkötesteihin. C# -projektissa voidaan suorittaa monella eri ohjelmistopaketilla yksikkötestejä. VSTest suorittaa oletuksena yksikkötestit käyttäen MSTest-ohjelmistokirjastoa. VSTest-toimintoon voidaan kuitenkin tuoda useampi testiadapteri, kuhan ne on lisätty projektiin ennen kääntämistä. Jos halutaan käyttää muita testiadaptereita, annetaan VSTest-toiminnolle hakemisto, josta adapterien assemblyt löytyvät. VSTest-toiminnossa voidaan käyttää hakemistoa, johon kääntötehtävä kääntää yksikkötestiprojektin tai tuoda tarvittavat adapterit artefakteina testitehtävään. Koska OneFactor putkisto suoritetaan yhdeltä palvelimelta tässä putkistossa, voidaan käyttää kääntötehtävän hakemistoa. VSTest-toiminto hakee kaikki annetussa hakemistossa löytyvät testiadapterit ja suorittaa assemblyn yksikkötestit niitä käyttäen.

4.5 Dynamics-ympäristön päivittäminen

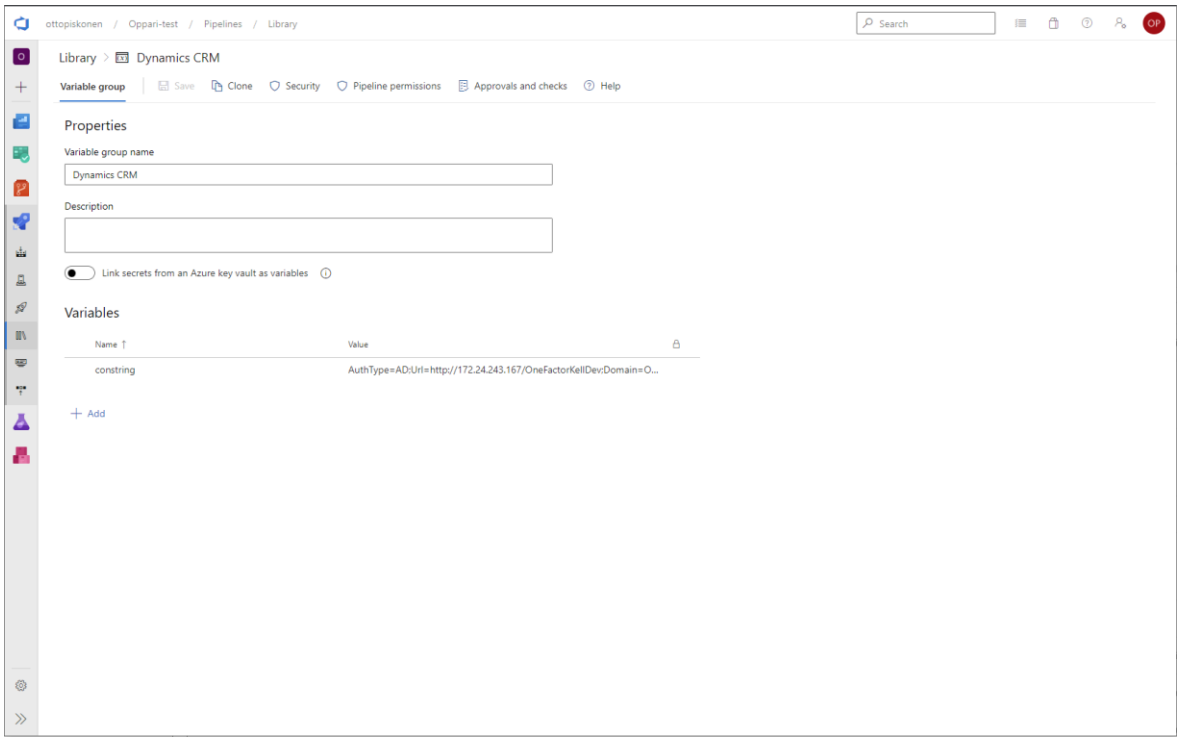
Kolmas vaihe putkistossa on Dynamics CRM -ympäristön päivittäminen. Tässä tehtävässä käytetään asennettua xRM CI Frameworkia. Tehtävä lataa aikaisemmasta putkiston tehtävästä käännetyn assembly artefaktin ja tallentaa sen tehtävähakemistoon käyttäen valmista muuttujaa Pipeline.Workspace. Ennen kuin xRM CI Frameworkia voidaan käyttää tehtävässä, täytyy sille tarkoitettu asennusvaihe lisätä. Asennusvaihe hakee markkinapaikasta ohjelmistopaketin, ja sen riippuvuuksien tiedostot, ja asentaa ne tehtävän suoritushakemistoon. Muut xRM CI Frameworkin toiminnot tarvitsevat niitä toimiakseen. Kuvassa 9 näkyy suoritettu päivitystehtävä ja sen komentokehote käyttöliittymässä.



Kuva 9. Päivitystehtävän suoritus käyttöliittymässä

Ennen kuin muutoksia tehdään ympäristöön, otetaan siihen yhteys käyttäen Ping-toimintoa. Ping-toiminto ottaa yhteyden annettuun ympäristöön käyttäen yhteysmerkkijonoa. Yhteysmerkkijono rakentuu muuttujista, jotka kertovat toiminnolle tarvittavat tiedot yhteyden ottamiseksi. Yhteysmerkkijonoja on monenlaisia ja oikean valitseminen riippuu yhteyden konfiguraatiosta. (Microsoft 2022n.) Ympäristö, johon yhteys tehdään, on on-premises versio Dynamicsista, joten yhteysmerkkijonoon määritetään muuttujat AuthType, URL, Domain, Username ja Password. AuthType kertoo, millä keinolla yhteys luodaan Dynamics-ympäristöön. Käyttämällä Authtypeä AD, voidaan yhteys muodostaa käyttämällä AD- tunnuksia.

URL-muuttujalle annetaan Dynamics-ympäristön käyttöliittymän osoite. Domain kertoo, mille Dynamics-ympäristön organisaatiolle otetaan yhteyttä, koska yhdellä palvelimelle asennetulla Dynamics-ympäristöllä voi olla monta organisaatiota. Organisaatiot mahdollistavat yhdeltä palvelimelta usean ympäristön käyttämisen samanaikaisesti. Lopuksi tarvitaan käyttäjätunnus ja salasana AD-tunnuksille, jolla yhteys luodaan ympäristöön. Yhteysmerkkijonomuuttuja voidaan tallentaa Azure Pipelinen kirjastoon, jonne muuttujaryhmä luodaan. Muuttujaryhmiä voidaan lisätä putkistoihin, mikä mahdollistaa yhtenäisten muuttujien käyttämisen useassa putkistossa. Kuvassa 10 näkyy käyttöliittymästä käytetty muuttujaryhmä.



Kuva 10. Muuttujaryhmä ja sen muuttujat käyttöliittymässä

Ping-toiminto mahdollistaa päivittämistehtävän keskeyttämisen, jos palvelimen yhteydessä on jotain vikaa. Putkisto siirtyy eteenpäin onnistuneen yhteyden luotuaan Ping-askeleessa.

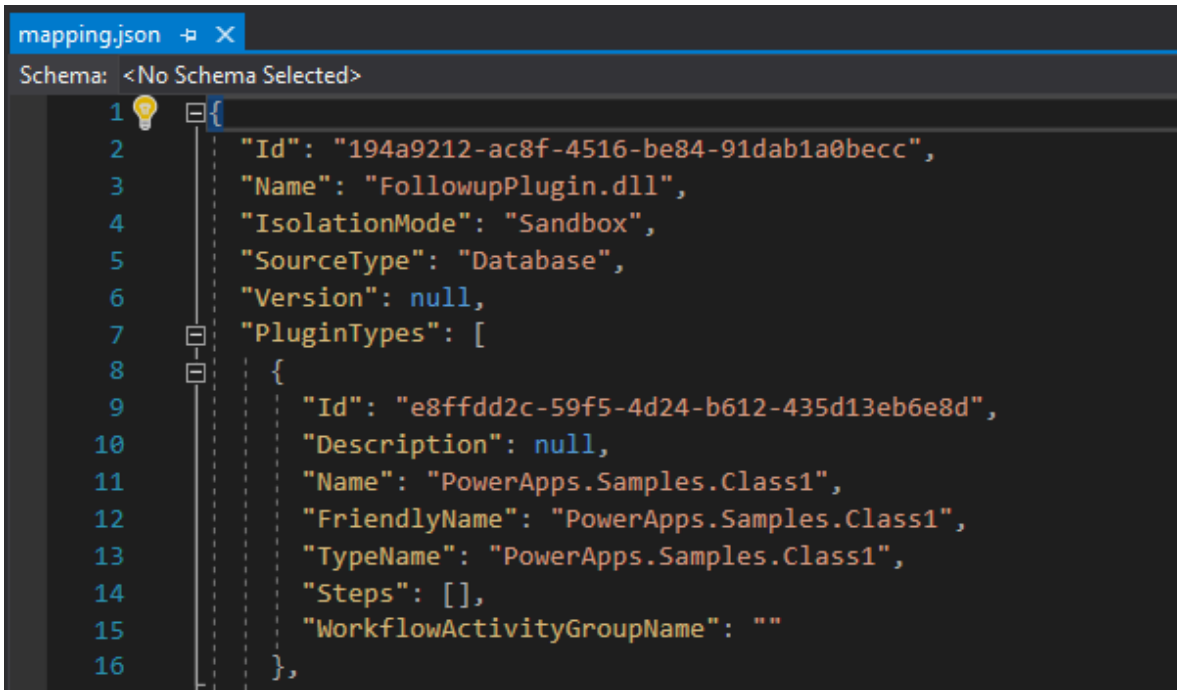
Seuraava vaihe on luoda mapping-tiedosto ladatulle assemblylle. Mapping-tiedosto luodaan JSON muodossa. Tämä voidaan suorittaa palvelimelle asennetun xRM CI Frameworkin lisäominaisuuksien avulla. Mapping-tiedoston luominen edellyttää kustomoituihin luokkiin attribuuttien lisäämistä. Attribuutteihin annetaan olion muodostaja, joka saadaan käyttämällä xRM CI Framework komentoa Get-XrmPluginRegistrationClass. Get-XrmPluginRegistrationClass PowerShell-komento tulostaa tarvittavan luokan. Luokka pitää sisällään

kaikki tarvittavat muuttujat ja olion muodostajat mapping-tiedoston luomiseen. Tulostettu luokka tallennetaan omana tiedostonaan projektiin, jonka jälkeen sen sisältämiä olion muodostajia voidaan käyttää attribuuteissa. Komento Get-XrmPluginRegistrationFromAssembly hakee annetusta polusta tiedoston ja luo siitä mapping-tiedoston haluttuun polkuun. Polussa täytyy olla jo olemassa oleva JSON-tiedosto, jotta komento toimii. Komento tarvitsee myös aikaisemmat luokkiin lisätyt attribuutit tai muuten mapping-tiedostoa ei voida luoda. Kuvassa 11 näkyy, luokkaan lisätty attribuutti käyttäen XrmCiPluginRegistration olion muodostajaa.

```
[XrmCiPluginRegistration(null, PluginIsolationMode.Sandbox, PluginSourceType.Database, "Workflow4", "FollowupPlugin (1.1.0.0)")]
– references | piskonen, 1 day ago | 1 author, 1 change
public class Workflow4 : CodeActivity
{
    [Input("BatchType")]
    [ReferenceTarget(ModelIT.Mff2011.XrmBase.Utils.MffEntities.MFFBatchType)]
    – references | piskonen, 1 day ago | 1 author, 1 change
    public InArgument<EntityReference> BatchType { get; set; }
```

Kuva 11. Luokkaan lisätty attribuutti

Päivittämistehtävään luotiin kustomoitu PowerShell-skripti, jolle annetaan polku ja ulostulopolku. Skripti suoritetaan Azure Pipelinen valmiilla toiminolla, joka suorittaa PowerShell-skriptejä. Käyttöliittymästä voidaan antaa myös argumentteja dynaamisesti, joita skripti tarvitsee. Skripti suorittaa Get-XrmPluginRegistrationFromAssembly -komennon ja päivittää jo olemassa olevan mapping-tiedoston, joka on tallennettu versionhallintaan projektin hakemistoon. Mapping-tiedoston luotua tehtävän seuraava vaihe on rekisteröidä assembly ympäristön tietokantaan. Kuvassa 12 näkyy esimerkki mapping-tiedostosta.



Kuva 12. Esimerkki mapping.json tiedostorakenteesta

Assemblyn rekisteröinti tehtiin xRM CI Frameworkin markkinapaikasta löytyvistä toiminnoista. Rekisteröintitoiminto tarvitsee aikaisemman yhteysmerkkijonon, rekisteröintityypin ja polun, josta toiminto löytää rekisteröitävän assemblyn. Toiminto saa yhteysmerkkijonon luodusta muuttujaryhmästä. Rekisteröintityyppi kertoo toiminnolle, minkälainen toiminto suoritetaan. Vaihtoehtoina on suorittaa Upsert, Reset tai Delsert. Upsert päivittää olemassa olevan assemblyn tietokantaan tai luo sellaisen, jos kyseessä on uusi assembly. Reset, poistaa edellisen rekisteröinnin ja asentaa assemblyn uudelleen tietokantaan. Delsert poistaa annetun assemblyn tietokannasta kokonaan. Rekisteröintitoiminto konfiguroitiin käyttämään Upsert-vaihtoehtoa. Assembly, jota käytetään rekisteröinnissä, on aikaisemmin ladattu artefakti, joka löytyy tehtävän hakemistosta. Toiminnolle voidaan antaa vaihtoehtoisia tietoja, kuten: ympäristössä olevan ratkaisun nimi, mihin rekisteröitävä assembly lisätään ja aikaisemmin luodun mapping-tiedoston polku, jota käytetään rekisteröinnissä. Rekisteröintitoiminto ottaa yhteyden ympäristöön ja päivittää annetun assemblyn tietokantaan.

Seuraavaksi tehtävä suorittaa ympäristössä olevan ratkaisun versionhallinnan. Ratkaisuun tallennetaan uusi versiokentän arvo ja ratkaisussa oleva HTML-tiedosto päivitetään uudella versioarvolla, Bitbucketista saadulla hash-arvolla ja haaran nimellä, josta muutokset tulivat. Näin muutoksia voidaan seurata tarvittaessa. Tähän tarkoitukseen luotiin kustomoitu PowerShell-skripti. Skripti käyttää xRM CI Frameworkin komentoja. Skripti ottaa dynaamisesti argumentteina versionhallintaan tallennetun HTML-tiedoston polun, Bitbucketin antaman

hash-arvon, haaran nimen, josta yhdistäminen tapahtui, ratkaisun nimen sekä yhteysmerkkijonon. Tehtävät suorittavat komennot Git-checkout ja Git-fetch jokaisen tehtävän alussa. Näillä komennoilla saadaan versionhallinnasta tiedostot tehtävän hakemistoon käytettäväksi. Tämä mahdollistaa versionhallintaan tallennetun HTML-tiedoston manipulointia putkistossa. Kuvassa 13 näkyy tehty kustomoitu PowerShell-skripti.

GetVersion.ps1

```
param(
    [string] $pathToHtml,
    [string] $gitHash,
    [string] $gitName,
    [string] $solutionName,
    [string] $constring
)

git config user.email "otto.piskonen@profitsoftware.com"
git config user.name "piskonen"
git checkout main
git pull

$date = Get-Date -Format "yy.MM.dd"
$content = Get-XrmSolution -UniqueSolutionName: $solutionName -ConnectionString: $constring
$increment = $content.Version.Substring(2, 2)

if($content.Version.Substring(5, 5) -ine $date.Substring(3,5)) {
    $monthAndDate = $date.Substring(3,5)
    $year = $date.Substring(0, 2)
    $version = "${year}${increment}.${monthAndDate}.0"
}
else {

    $int = $content.Version.Substring(11, 1) -as [int]
    $version = $content.Version.Substring(0, 11) + ++$int
}


@("<dt>${version}: ${gitHash} ${gitName}</dt>") + (Get-Content $pathToHtml) | Set-Content $pathToHtml

Write-Host "##vso[task.setvariable variable=version;isOutput=true;]$version"
Write-Host "Version control done.. Next merging changes"

git add -A
git commit -m "Pipeline changes [skip ci]"

git push origin HEAD:main
```

Kuva 13. PowerShell-skripti ratkaisun versionhallinnalle

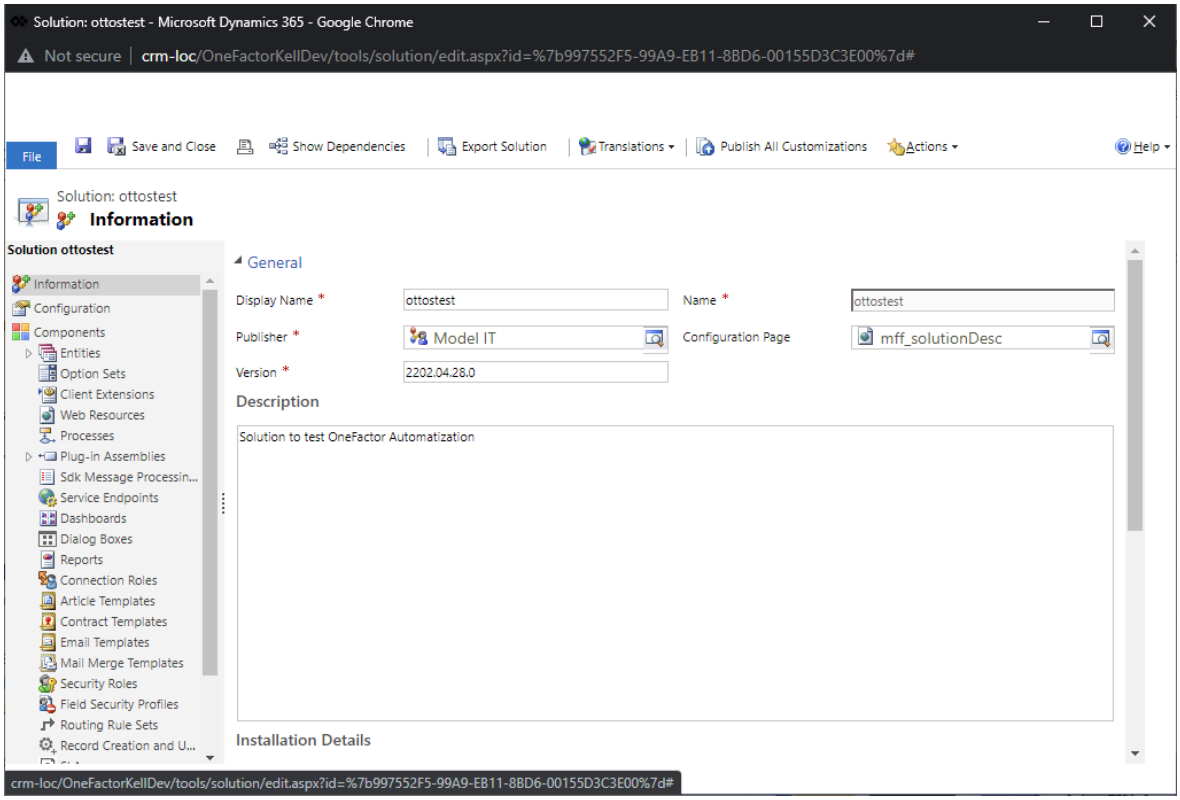
Git hash -arvo saadaan käyttämällä valmista muuttujaa Build.SourceVersion, jota käytetään ratkaisun versionhallinnassa. Haaran nimi saadaan valmiista muuttujasta System.PullRequest.SourceBranch. Saatu arvo on haara, josta muutokset yhdistettiin kehityshaaraan. Ratkaisun nimi voidaan antaa argumenttina käyttöliittymässä ja yhteysmerkkijono käyttää muuttujaryhmästä saatua arvoa. Skripti hakee kehityshaarasta uusimmat tiedot käyttämällä komentoa Git-checkout ja Git-pull, koska HTML-tiedostoon tehdyt muutokset täytyy tallentaa versionhallintaan. Käyttämällä xRM CI Frameworkin komentoa Get-XrmSolution, saadaan ympäristöstä tietyn ratkaisun tiedot. Tämä komento tarvitsee argumentteina ratkaisun nimen ja yhteysmerkkijonon. Ratkaisun tiedot saatuaan skripti tarvitsee suoritushetken päivänmäärän, joka saadaan käyttämällä komentoa Get-Date. Get-Date-komento argumentoidaan Format-muuttujalla, jolla arvot voidaan muokata halutulla tavalla. Tähän toteutukseen tarvittiin päiväarvo, josta ilmenee vuosi, kuukausi ja päivä. Seuraavaksi skripti vertaa

ratkaisun versiosta löytyvää arvoa haettuun päivänmääräarvoon. Ratkaisun versio -kentän arvo pitää ensin jäsentää, koska siitä löytyy myös tämänhetkisen inkrementin arvo. Käyttämällä Substring funktiota saadaan versiosta kuukausi ja päivänmäärä. Tätä arvoa verrataan haetun päivän kuukauteen ja päivänmäärään. Jos arvot eivät ole samat luodaan uusi versiokentän arvo. Versioon tulee sen hetkinen vuosi ja inkrementtiarvo, joiden jälkeen pisteellä eroteltuna kuukausi, päivä ja aktiivinen versio. Aktiivinen versio kasvaa yhdellä, jos ratkaisuun tehdään muutoksia saman päivän aikana. Jos verrattavat päiväarvot eivät ole samat, asetetaan aktiivinen versio nollaksi. Uusi versiokentän arvo, argumenteista saatu hash-arvo ja haaran nimi lisätään HTML-tiedostoon käyttämällä komentoa Get-Content ja Set-Content. Näin saadaan uusi arvo lisättyä tiedoston ylimmälle riville. Muutosten jälkeen uusi versiokentän arvo tallennetaan putkiston muuttujaksi, koska sitä tarvitaan vielä myöhemmin. Muuttujia voidaan tallentaa suoritettavaan putkistoon tulostamalla se PowerShellista. Se on esitetty kuvassa 14.

```
Write-Host "##vso[task.setvariable variable=version;isOutput=true;]$version"
Write-Host "Version control done.. Next merging changes"
```

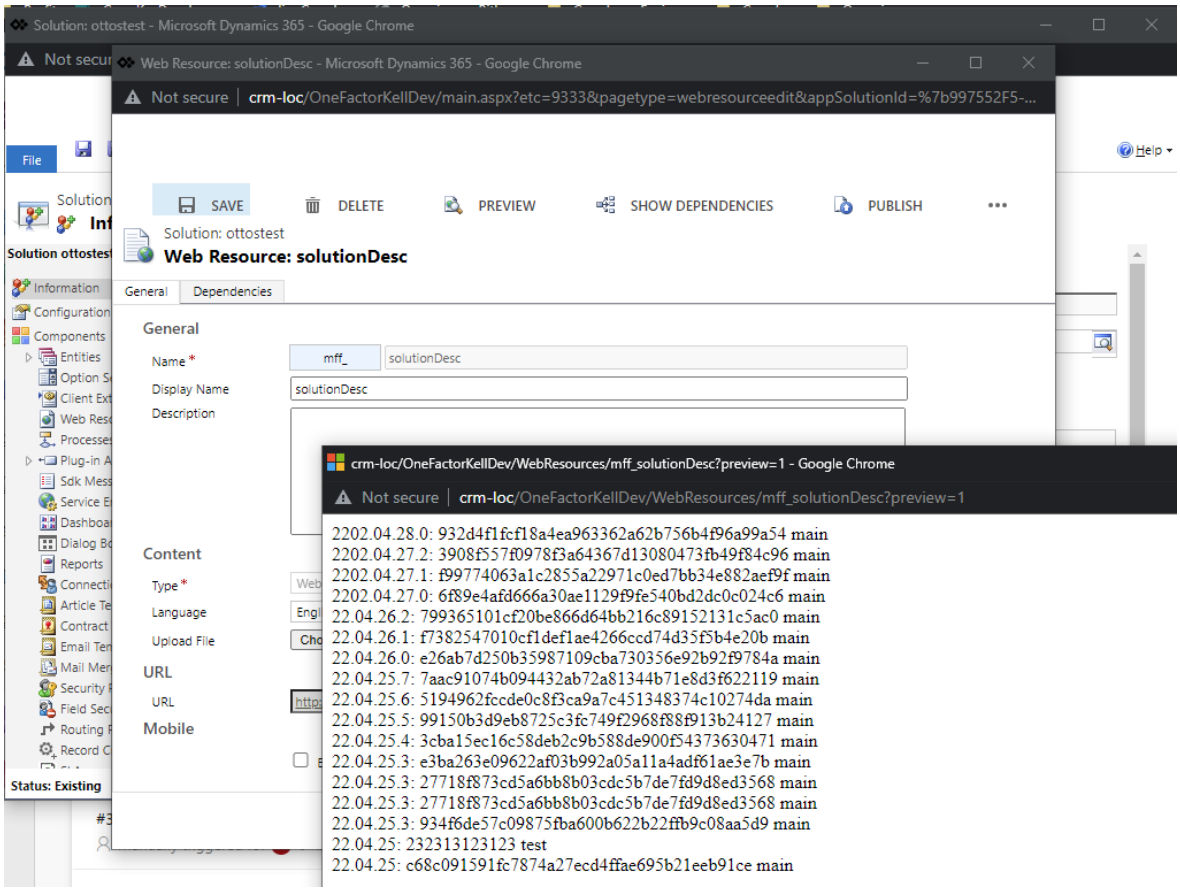
Kuva 14. Muuttuja $version tulostus PowerShell-skriptissä

Lisäämällä askeleelle ulostuloreferenssi, voidaan muuttujaa kutsua sen avulla myöhemmin tehtävän toiminnoissa. HTML-tiedosto tallennetaan versionhallintaan, jotta sitä voidaan ylläpitää automaatiossa käyttämällä komentoja Git-add, Git-commit ja Git-push. Git-commit viestiin lisäämällä [skip ci] voidaan kertoa Bitbucketille, että tästä muutoksesta ei haluta käynnistää automaatioputkistoa. Lopuksi tehtävä asettaa ratkaisun version käyttämällä arvoa, mikä saadaan tallennetusta muuttujasta. Kuvassa 15 näkyy OneFactor käyttöliittymässä ratkaisun versiotiedot.



Kuva 15. Ratkaisu ikkunan versiotiedot

Kaikki viimeiset tehtävän askeleet käyttävät xRM CI Frameworkin toimintoja. Toiminnolle annetaan yhteysmerkkijono, ratkaisun nimi ja versionumero. Ratkaisun versionhallinta käyttää muokattua HTML-tiedostoa, johon on tallennettu tarvittavat tiedot. HTML-tiedosto voidaan upottaa ratkaisun konfiguraatio sivu -kenttään ja tallentaa ratkaisun web-resursseihin. Ratkaisun versionhallinta on parasta suorittaa HTML-tiedostoon, koska ratkaisusta löytyvä kuvaus kenttä antaa lisätä vain 2000 merkkiä. Toiminto, joka päivittää web-resurssin, tarvitsee yhteysmerkkijonon, polun HTML-tiedostoon ja ratkaisun nimen, jossa web-resurssi on tallennettu. Toiminto etsii annetusta polusta tiedostoja, jotka voidaan määrittää filttereiden avulla. Löydettyään tiedoston, toiminto ottaa yhteyden ympäristöön ja päivittää kyseisen tiedoston tietokantaan. Kuvassa 16 nähdään HTML-tiedosto ja miten sitä voidaan katsella käyttöliittymästä.



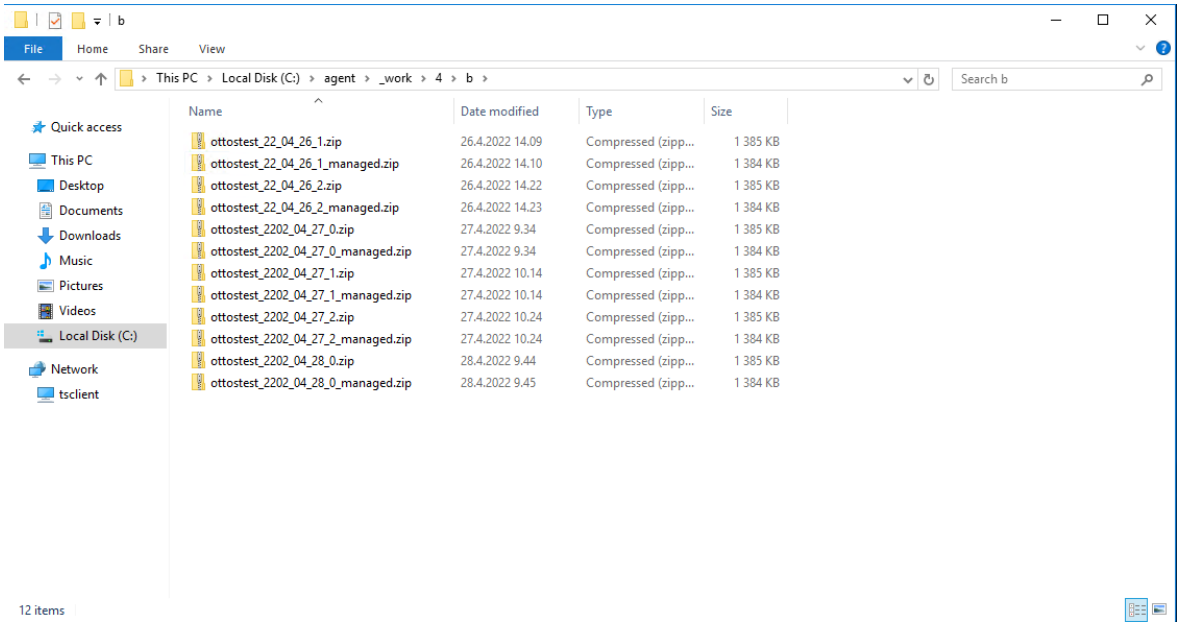
Kuva 16. Avattu HTML-tiedosto, ratkaisun configuration page -kentästä

Tehtävän viimeinen askel on suorittaa kaikkien muutosten julkaisuympäristöön. Tämä toiminto tarvitsee vain yhteysmerkkijonon ja se suorittaa Dynamics-ympäristössä Publish Customatization -toiminnon, joka voidaan suorittaa myös ympäristön käyttöliittymästä. Tämä toiminto julkaisee muutokset kaikkien käyttäjien nähtäväksi.

4.6 Ratkaisun vieminen ympäristöstä

Putkiston viimeinen tehtävä on hakea ympäristöstä päivitetyt ratkaisut ja viedä ne ulos ympäristöstä. Ratkaisun vienti ja tuonti onnistuu käyttämällä markkinapaikasta löytyvää xRM CI Frameworkia. Tehtävässä asennetaan ohjelmistopaketti samaa askelta käyttäen kuin ympäristön päivittämistehtävässä. Ohjelmistopaketin asennettua tehtävä käyttää Export Solution -askelta, jolle annetaan yhteysmerkkijono, ratkaisun nimi ja ulostulopolku. Valinnaisena valintana voidaan päättää, halutaanko ratkaisu viedä managed vai unmanaged -tilassa. Askeleesta voidaan myös viedä ratkaisu molemmissa tiloissa ulos ympäristöstä. Ulostulopolku määrittää, mihin viety ratkaisu tallentuu. Tehtävässä käytettiin valmista

muuttujaa Build.Binariesdirectory, josta saadaan tehtävän hakemiston valmiin kansion polku. Kuvassa 17 näkyy hakemisto, mihin viedyt ratkaisut tallentuvat palvelimella.



Kuva 17. Vietyjen ratkaisujen hakemisto

5 Yhteenveto ja pohdinta

Opinnäytetyön tavoitteena oli toteuttaa OneFactor-varainhoitojärjestelmälle soveltuvuusselvitys päivittäisen toimituksen automatisoinnista. Työssä onnistuttiin toteuttamaan automatisoituputkisto Azure DevOps -alustalla. Putkisto seuraa ketteriä CI/CD-menetelmiä ja on räätälöity OneFactor-alustan tarpeisiin. Putkiston automatisointia helpottamaan luotiin päivittäiseen toimitukseen sääntöehdotuksia. Sääntöjä luotiin käyttämällä yleisiä ketteriä menetelmiä ohjelmistokehityksessä. Tuotettu putkisto esiteltiin lopuksi Crosskeyn kehittäjille.

Toimituksen automatisoinnissa käytetty ohjelmistopaketti xRM CI Framework mahdollisti kaikki tarvittavat toiminnot Microsoft Dynamics -alustan muokkaamiseen ohjelmallisesti. Ohjelmistopaketin käyttöönotto tuotti ongelmia. Ohjelmistopaketti on avointa lähdekoodia ja sen tarjoamia toimintoja ei ollut dokumentoitu kunnolla. Ympäristössä käytettävä Microsoft Dynamics CRM SDK täytyi olla yhteensopiva xRM CI Frameworkin kanssa. Oikean version löytäminen vaati tutkimustyötä. Ohjelmistopaketista löytyvä Azure DevOps -markettipaikkaversio ei tarjonnut kaikkia ohjelmistopaketin toimintoja suoraan käyttöliittymästä. Ohjelmistopaketti tuli asentaa palvelimelle PowerShell Galleriasta niin, että edistyksellisemmät toiminnot saatiin käyttöön. Olemattoman dokumentoinnin takia joidenkin edistyksellisten toimintojen käyttöönotto vaati paljon työtä. Ohjelmistopaketin käyttö putkistossa onnistui lopulta täysin.

Tuotettu automatisointiputkisto on valmiina käyttöönottoon OneFactor-alustalle. Putkistossa käytetyt menetelmät ja ohjelmistopaketit mahdollistavat sen käyttöönoton, millä tahansa vain alustalla, joka tukee YML-tiedostoja. Standardit voidaan ottaa käyttöön automatisoinnin rinnalla, mutta ne voivat vaatia tulevaisuudessa muokkaamista tarpeiden mukaan. Jatkokehitystä työlle on ottaa putkisto käyttöön useassa ympäristössä samanaikaisesti. Automatisointiputki olisi mahdollistaa tuottaa myös inkrementtien ylläpitämiseen ja lähdekoodin yhdistämiseen. Lisäksi tietyt putkiston tehtävät tarvitsivat versionhallinnasta löytyviä tiedostoja askeleiden suorittamiseen. Jokainen tehtävä oletusarvoisesti suoritti Git-komentoja, jotka latasivat lähdekoodin tehtävän hakemistoon käytettäväksi. Tämä vei todella paljon tallennustilaa palvelimelta. Lähdekoodin haku voitaisiin poistaa tehtävistä, jotka sitä eivät tarvitse ja se voisi tapahtua vain yhdessä tehtävässä, josta kaikki muut tehtävät voisivat sen hakea. Päivittämistehtävässä käytettyjen skriptien syntaksia voitaisiin parantaa tulevaisuudessa, vaihtamalla tapaa, jolla versio- kentän arvot haetaan muuttujasta. Skriptissä voitaisiin käyttää säännöllisiä lausekkeita, mikä vähentäisi käyttäjävirheitä dynaamisten arvojen antamisessa. Crosskeylle esitetty automaatioputkisto saavutti odotukset ja on hyvä lähtökohta OneFactorin toimituksen automatisoinnille tulevaisuudessa.

Lähteet

Apica. 2016. Continuous Integration: 5 Tips to Avoid “Integration Hell”. Viitattu 16.5.2022. Saatavissa https://www.apica.io/top-5-tips-avoid-integration-hell-continuous-integration/

Be A Better Dev. 2022. The IDEAL & Practical CI / CD Pipeline - Concepts Overview. Viitattu 29.4.2022. Saatavissa https://www.youtube.com/watch?v=OPwU3UWCxhw

Codilime. 2021. Best CI/CD pipeline tools you should know. Viitattu 29.4.2022. Saatavissa https://codilime.com/blog/best-ci-cd-pipeline-tools-you-should-know/

Crosskey. 2019. Crosskey ja Model IT yhdistyvät. Viitattu 29.4.2022. Saatavissa https://www.crosskey.fi/fi/crosskey-ja-model-it-yhdistyvat/

Crowdbotics. 2022. Fully Automated Development: Where We Are Today and Where We’re Headed. Viitattu 16.5.2022. Saatavissa https://www.crowdbotics.com/blog/fully-automated-development-where-we-are-today-and-where-we-are-heading

Fintech Farm. 2021. Biggest fintech companies and startups in Finland. Viitattu 6.5.2022. Saatavissa https://www.helsinkifintech.fi/news/biggest-fintech-companies-and-startups-in-finland/

InfoWorld. 2022. What is CI/CD? Continuous integration and continuous delivery explained. Viitattu 29.4.2022. Saatavissa https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html

Microsoft. a. Mikä on CRM-ohjelmisto? Viitattu 29.4.2022. Saatavissa https://dynamics.microsoft.com/fi-fi/crm/crm-software/

Microsoft. b. Mitä on CRM? Viitattu 29.4.2022. Saatavissa https://dynamics.microsoft.com/fi-fi/crm/what-is-crm/

Microsoft. c. Azure DevOps. Viitattu 29.4.2022. Saatavissa https://azure.microsoft.com/en-us/services/devops/#overview

Microsoft. d. Azure Boards. Viitattu 29.4.2022. Saatavissa https://azure.microsoft.com/en-us/services/devops/boards/

Microsoft. 2022a. Use plug-ins to extend business processes. Viitattu 29.4.2022. Saatavissa https://docs.microsoft.com/en-us/power-apps/developer/data-platform/plug-ins

Microsoft. 2022b. Introduction to solutions. Viitattu 29.4.2022. Saatavissa https://docs.microsoft.com/en-us/dynamics365/customerengagement/on-premises/developer/introduction-solutions?view=op-9-1

Microsoft. 2022c. What is a DLL? Viitattu 29.4.2022. Saatavissa https://docs.microsoft.com/en-us/troubleshoot/windows-client/deployment/dynamic-link-library

Microsoft. 2022d. Sample: Create a custom workflow activity. Viitattu 29.4.2022. Saatavissa https://docs.microsoft.com/en-us/power-apps/developer/data-platform/workflow/sample-create-custom-workflow-activity

Microsoft. 2022e. Introduction to entities. Viitattu 29.4.2022. Saatavissa https://docs.microsoft.com/en-us/dynamics365/customerengagement/on-premises/developer/introduction-entities?view=op-9-1

Microsoft. 2022f. Authenticate users in Dynamics 365 Customer Engagement (on-premises). Viitattu 29.4.2022. Saatavissa https://docs.microsoft.com/en-us/dynamics365/customerengagement/on-premises/developer/authenticate-users?view=op-9-1

Microsoft. 2022g. Ympäristöjen yleiskatsaus. Viitattu 12.5.2022. Saatavissa https://docs.microsoft.com/fi-fi/power-platform/admin/environments-overview

Microsoft. 2022h. Overview of extensions for Azure Boards. Viitattu 29.4.2022. Saatavissa https://docs.microsoft.com/en-us/azure/devops/boards/extensions/?view=azure-devops

Microsoft. 2022i. What is Azure Pipelines? Viitattu 29.4.2022. Saatavissa https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops

Microsoft. 2022j. Azure Pipelines agents. Viitattu 29.4.2022. Saatavissa https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=azure-devops&tabs=browser

Microsoft. 2022k. NuGet Tool Installer task. Viitattu 29.4.2022. Saatavissa https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/tool/nuget?view=azure-devops

Microsoft. 2022l. NuGet task. Viitattu 29.4.2022. Saatavissa https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/package/nuget?view=azure-devops

Microsoft. 2022m. Use connection strings in XRM tooling to connect to Dynamics 365 Customer Engagement (on-premises). Viitattu 29.4.2022. Saatavissa https://docs.microsoft.com/en-us/dynamics365/customerengagement/on-premises/developer/xrm-tooling/use-connection-strings-xrm-tooling-connect?view=op-9-1

Microsoft. 2022n. Use predefined variables. Viitattu 29.4.2022. Saatavissa https://docs.microsoft.com/en-us/azure/devops/pipelines/build/variables?view=azure-devops&tabs=yaml

Piskonen, O. 2022. OneFactor Development Documentation. Wikisivu.

Rantanen, O. 2022. Ympäristöarkkitehti. Crosskey Oy. Haastattelu 28.4.2022.

RedHat. 2018. What is CI/CD? Viitattu 29.4.2022. Saatavissa https://www.redhat.com/en/topics/devops/what-is-ci-cd/

Synopsys. Why is CI/CD important? Viitattu 6.5.2022 Saatavissa https://www.synopsys.com/glossary/what-is-cicd.html

Tirupati. 2020. Mikä on CI / CD-putki? Viitattu 29.4.2022. Saatavissa https://tirupati-tour-packages.com/fi/mika-on-ci-cd-putki/

YAML. YAML: YAML Ain't Markup Language™. Viitattu 29.4.2022. Saatavissa https://yaml.org/

Wael Hamze. xrm-ci-framework. Viitattu 29.04.2022. Saatavissa https://github.com/Wael-Hamze/xrm-ci-framework/wiki

Ålandsbanken. Ålandsbankenin kehitys jatkuvaa ja vakaata. Viitattu 6.5.2022. Saatavissa alandsbanken.fi/meista/alandsbankenista