jamk

Novi-ohjelmiston automaattinen julkaisu CI/CD-putken avulla

Hannu Kujanpää

Opinnäytetyö, AMK
Marraskuu 2022
Tietojenkäsittely ja tietoliikenne
Insinööri (AMK), Tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Kujanpää, Hannu

Novi-ohjelmiston automaattinen julkaisu CI/CD-putken avulla

Jyväskylä: Jyväskylän ammattikorkeakoulu. Marraskuu 2022, 35 sivua

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

Julkaisun kieli: suomi

Verkkojulkaisulupa myönnetty: kyllä

Tiivistelmä

Automatisointi on iso osa nykyaikaista ohjelmistokehityksen prosessia ja sillä voidaan vähentää kehittäjien käyttämää aikaa automatisoitavissa oleviin tehtäviin. Usein samalla kaavalla toistuvat asiat, kuten ohjelmiston testaus ja julkaisu, voidaan nykyään tehdä automaattisesti. Automatisoinnista puhuttaessa termit jatkuva integraatio, jatkuva julkaisu ja DevOps ovat esillä useasti ja ne ovat iso osa nykyaikaista ohjelmistokehitystä.

Opinnäytetyön toimeksiantaja oli Pinja Operational Excellence Oy, joka on osa Pinja Group Oy:tä. Työn tavoitteena oli parantaa Novi-ohjelmiston kehitysprosessia tekemällä automaattinen julkaisuputki, jonka avulla voitaisiin esimerkiksi testata uusia ominaisuuksia uuden version kehityksen aiemmassa vaiheessa.

Työ toteutettiin tuotteen kehittämistyönä. Tietoperustassa tarkasteltiin mitä DevOps, jatkuva integraatio, jatkuva julkaisu ja julkaisuputki ovat teoriassa. Julkaisuputken toteutuksessa toteutustavoiksi valittiin GitHub Actions ja Octopus Deploy, joiden avulla voitiin automatisoida uuden version julkaisu ja ohjelmiston asennus palvelimelle. Työssä keskityttiin automatisoimaan julkaisuprosessia Novi desktop -verkkosovellukselle, mikä on yksi kolmesta Novin osasta.

Työn tuloksena saatiin toimiva julkaisuputki Novi desktopille. Toteutettu julkaisuputki käynnistyy, kun kehittäjä lataa koodimuutokset GitHubissa olevaan kehityshaaraan. Ohjelmisto koostetaan GitHub Actionsin avulla ja se ladataan Octopus Deployn rekisteriin. Octopus Deployn avulla testaajat voivat asentaa viimeisimmän kehityksessä olevan version testipalvelimelle. Tuloksena saatiin myös tietoa DevOpsista, CI/CD:stä ja julkaisuputkista sekä saatiin lisättyä prosessin läpinäkyvyyttä uusilla kehitystikettien tiloilla.

Opinnäytetyön tavoitteet saavutettiin ja syntyneellä julkaisuputkella voitiin testata ohjelmiston uusimpia ominaisuuksia heti kun kehittäjä lataa koodimuutoksensa versionhallintaan. Julkaisuputki koettiin hyvänä lisänä Novin kehitysprosessiin ja julkaisuputkelle kirjattiin jo joitain jatkokehitysideoita.

Avainsanat (asiasanat)

DevOps, CI/CD, automaatio, ohjelmistokehitys, jatkuva julkaisu, julkaisuputki, Pinja

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Kujanpää, Hannu

Automatically publishing Novi using a CI/CD pipeline

Jyväskylä: JAMK University of Applied Sciences, November 2022, 35 pages.

Engineering and technology. Degree Programme in Information and Communications Technology. Bachelor's thesis.

Permission for web publication: Yes

Language of publication: Finnish

Abstract

Automatizing software development steps is a big part of the modern software development process. By automatizing recurring tasks, such as creating releases and testing the software, developers can focus on more important things. DevOps, continuous integration and continuous delivery are continuously mentioned in modern software development material as essential parts of the process.

The client of the thesis was Pinja Operational Excellence Oy, which is a part of Pinja Group Oy. The goal of the thesis was to create a CI/CD pipeline for Novi and thus improve the development process. As an example, testers could test new features earlier in the process using the delivery pipeline.

The thesis was carried out as development of a product. The theoretical basis is an overview of DevOps, continuous integration, continuous delivery and CI/CD pipeline. The practical implementation was created using GitHub Actions and Octopus Deploy. The pipeline was created for only Novi desktop, which is a web app and one of the three main Novi components.

The result of the thesis was a functional CI/CD pipeline, which is run every time developer pushes code to GitHub. The software is built and released using GitHub Actions, which uploads the package to Octopus Deploy. Octopus Deploy was used to enable testers to install the newest version to a test environment. Another result was theoretical knowledge of DevOps and CI/CD. The pipeline and new ticket statuses also increased the transparency of the development process.

The main goal of the thesis was achieved. After the deployment of the pipeline, testers could test the latest features in development. The pipeline had a great reception from the Novi team and already has some further improvements planned for it.

Keywords/tags (subjects)

DevOps, CI/CD, automation, software development, continuous delivery, CI/CD pipeline, Pinja

Miscellaneous (Confidential information)

Sisältö

Kuviot

1 Johdanto

1.1 Opinnäytetyön tausta

Ohjelmistokehityksen vaiheiden automatisointi vaikuttaa olevan nykyään monessa projektissa arkipäivää. Artikkeleita ja blogikirjoituksia automatisoinnin hyödyllisyydestä ja kannattavuudesta löytyy internetistä huomattava määrä. Monet isot yritykset hyödyntävät automaatiota erilaisin keinoin ja myös tarjoavat ohjeita automatisoinnin toteuttamiseen. Mitä eri paikoissa vilisevät termit, kuten DevOps, jatkuva integraatio ja jatkuva julkaisu, oikeasti tarkoittavat ja miten ne liittyvät automatisointiin ja ohjelmiston kehitykseen?

Automatisoinnin avulla yritys voi vähentää työntekijöiden käyttämää aikaa tehtäviin, joita tietokone voisi tehdä myös ilman käyttäjää. Tällöin työntekijät voivat keskittyä tärkeämpiin tehtäviin ja yrityksen prosesseihin voidaan saada lisää nopeutta. Kun tehtäviä automatisoidaan, sekä tuottavuus että prosessin luotettavuus kasvavat. (Understanding automation 2018). Yksi tapa, jolla voidaan lisätä automatisointia ja nopeutta ohjelmistokehitykseen on DevOps. DevOpsia hyödyntävillä tiimeillä voi olla jopa kymmeniä, ellei satoja kertoja nopeampi prosessi kehittää ohjelmistoa. (Buchanan n.d.)

Tämän opinnäytetyön toimeksiantaja on Pinja Operational Excellence Oy, joka on osa Pinja Group Oy:tä. Pinja aloitti toimintansa 1990-luvulla nimellä Protacon. Vuosien saatossa yritys kasvoi ja vuonna 2020 se sai brändiuudistuksen myötä uudeksi nimekseen Pinja. Yritys myytiin 2021 marraskuussa pääomasijoitusyhtiö Norvestorille. Pinjan liikevaihto oli 42,7 miljoonaa euroa vuonna 2021. Pinja työllistää noin 450 IT-alan asiantuntijaa ja sen ohjelmistoja käytetään kolmessakymmenessä maassa. Pinja tarjoaa erilaisia digitaalisia ratkaisuja esimerkiksi valmistavan teollisuuden prosessien tehostamiseen ja tiedolla johtamiseen. (Historia n.d.; Toimialat n.d.; Palvelut n.d.; Ohjelmistoyhtiö Pinjalle... 2021; Pinja Monthly tammikuu 2021.)

1.2 Opinnäytetyön tavoitteet ja toteutus

Opinnäytetyön toimeksiantona oli toteuttaa automaattinen ohjelmiston julkaisuputki kehityksessä olevien ominaisuuksien testausta varten. Työn yhtenä tavoitteena oli saada kehitettyä vähintään

pohja julkaisuputkelle, jota voitaisiin jatkossa hyödyntää esimerkiksi uusien ominaisuuksien testaamisessa. Toisena työn tavoitteena oli tutkia mitä eri termit, kuten DevOps, tarkoittavat ja mitä ne pitävät sisällään. Opinnäytetyön aihe syntyi tarpeesta parantaa Novi-ohjelmiston kehitys- ja julkaisuprosessia ja tutkijan mielenkiinnosta aihetta kohtaan.

Opinnäytetyö toteutettiin tuotteen kehittämistyönä, jonka tarkoitus oli ratkaista käytännön ongelmia sekä parantaa ohjelmiston kehitysprosessia. Opinnäytetyössä on ensiksi teoriaosa, jossa on käsitelty teknisen toteutuksen pohjalla olevia käsitteitä. Teknisen toteutuksen osassa toteutettiin tarvittu ratkaisu. Tietoa opinnäytetyöhön hankittiin kirjoista, työpaikalla käydyistä keskusteluista ja verkosta löytyvistä materiaaleista, kuten artikkeleista ja blogikirjoituksista. Työn toteutuksessa voitiin myös hyödyntää tutkijan aikaisempaa kokemusta aiheesta.

1.3 Tutkimusasetelma

Opinnäytetyön aihetta rajattiin seuraavilla kolmella tutkimuskysymyksellä:

1. Mitä DevOps on?
2. Mitä jatkuva integraatio ja jatkuva julkaisu tarkoittavat?
3. Mikä julkaisuputki on?
4. Millainen julkaisuputki on käytännössä?

Koska työn toimeksiantona oli rakentaa automaattinen julkaisuputki Novi-ohjelmistolle, yhdeksi opinnäytetyötä rajaavaksi tutkimuskysymykseksi valittiin tutkia miten sellainen toteutetaan käytännössä. Käytännön toteutus vaati tietoa teoriasta, joten toiseksi tutkimuskysymykseksi valittiin tutkia mitä julkaisuputki tarkoittaa. Julkaisuputken teoriaa tutkiessa esiin nousi myös termit jatkuva integraatio ja jatkuva julkaisu. Yhdeksi tutkimuskysymykseksi valittiin tutkia mitä näillä termeillä tarkoitetaan. Alustavaa tutkintaa tehdessä esiin nousi myös termi DevOps, joka liittyi julkaisuputkiin, jatkuvaan integraatioon ja jatkuvaan julkaisuun. Koska DevOps liittyi vahvasti muihin tutkimuskysymyksiin, valittiin yhdeksi tutkimuskysymykseksi selvittää mitä DevOps on.

2 Käytettävät ohjelmistot

2.1 Novi by Pinja -ohjelmisto

Novi on Pinjan kehittämä kunnossapitojärjestelmä, jolla voidaan muun muassa toteuttaa ennakoivaa kunnossapitoa, hallita materiaaleja ja resursoida kunnossapidon töitä (Novi by Pinja, n.d.). Novi rakentuu kolmesta osasta, jotka ovat Novi desktop, Novi mobile ja Novi backend. Novi desktop -nimi viittaa työpöytäsovellukseen, mutta tämä Novin osa on ASP.NET Web Forms -verkkosovellus. Novi desktop on yhdessä SQL-tietokannan kanssa Novin pääkomponentti. Sen avulla hallitaan muun muassa asetuksia, konfiguraatioita ja käyttäjiä. Noviin kuuluu myös mobiililaitteille kehitetty React-verkkosovellus, Novi mobile, joka hyödyntää Novi backend APIa. Myös tietokoneella käytettävään Noviin on tehty moduuleja, jotka hyödyntävät Novi backendia. Novi backend on ASP.NET Core API ja se on toteutettu C#-ohjelmointikielellä.

2.2 Visual Studio 2022 -kehitysympäristö

Visual Studio on Microsoftin kehittämä integroitu kehitysympäristö. Visual Studiolla voidaan esimerkiksi kirjoittaa koodia, testata koodia ja julkaista ohjelmisto. Visual Studio pitää sisällään paljon erilaisia työkaluja ohjelmistojen kehittämiseen, kuten IntelliSense, Git-versionhallinta sekä erilaiset graafiset työkalut. Visual Studiosta on olemassa eri versioita: Community, Professional ja Enterprise. (Welcome to the Visual Studio IDE, 2022)

2.3 GitHub-verkkosivusto

2.3.1 GitHub yleisesti

GitHub on vuonna 2008 lanseerattu sivusto, joka tarjoaa pääasiassa käyttäjilleen keskitettyä versionhallintaa hyödyntäen Git-versionhallintatyökalua. Se tarjoaa myös erilaisia työkaluja jatkuvaan integraatioon, jatkuvaan julkaisuun ja lähdekoodin turvallisuuden ylläpitoon. GitHubilla on myös työpöytä- ja mobiilisovellus, sekä monia ohjelmistokehitystä helpottavaa työkalua. GitHubilla on yli 73 miljoonaa käyttäjää ja sivustolla on yli 200 miljoonaa repositoriota. (What Is GitHub? 2021; GitHub n.d.; Preston-Werner 2008)

2.3.2 GitHub Actions

GitHubin tarjoama jatkuvan integraation ja jatkuvan julkaisun palvelu on nimeltään GitHub Actions. Actionsin avulla käyttäjät voivat automatisoida esimerkiksi ohjelmiston koostamisen ja testaamisen. Käyttäjät voivat luoda erilaisia workfloweja, joihin voidaan määritellä workflowin vaiheet ja vaiheen askeleet. Määrittelyt voidaan tehdä YAML-syntaksilla kirjoitetuilla tiedostoilla lisäämällä ne .github/workflows -kansioon. Workflow voi pitää sisällään useita vaiheita ja vaihe voi pitää sisällään useita askelia. Workfloweja voidaan ajaa joko manuaalisesti tai erilaisten tapahtumien, kuten koodimuutoksen tai uuden pull requestin, sattuessa. Workflowien suorittamiseen GitHub tarjoaa Linux-, Windows- ja macOS -virtuaaliympäristöjä, mutta käyttäjä voi halutessaan käyttää suoritukseen omaa palvelintaan. (Understanding GitHub Actions n.d.; Automate your workflow… n.d.)

2.4 Octopus Deploy -ohjelmisto

2.4.1 Octopus Deploy yleisesti

Octopus Deploy on vuonna 2012 perustettu yritys, joka kehittää samannimistä automatisointityökalua. Sitä voidaan käyttää esimerkiksi ohjelmiston julkaisuun ja sen pystytykseen palvelimelle. Octopus Deployssa on useita työkaluja, joilla voidaan esimerkiksi suorittaa automatisoituja toimenpiteitä ympäristöissä tai hallinnoida käyttäjien rooleja. Ohjelmiston toimitusprosessiin voidaan määritellä vaiheita, joilla voidaan automatisoida koko ohjelmiston pystytysprosessi aina infrastruktuurin hallinnasta ohjelmiston konfigurointiin. Octopus Deploy tarjoaa valmiiksi tehtyjä mallivaiheita, joihin käyttäjät voivat itse määritellä muuttujia, kohdeympäristöjä ja muita ohjelmiston pystytykseen tai konfigurointiin vaadittuja asioita. Octopus Deployta voidaan käyttää joko Octopus Deployn tarjoamassa pilviympäristössä tai omille palvelimille asennettuna. Jos käyttäjä haluaa käyttää Octopus Deployn tarjoamaa verkkopalvelua, pystytettävät ohjelmistot voidaan kuitenkin asentaa käyttäjän omille palvelimille. (Complex deployments made easy n.d.; Deployment process n.d.; Getting started n.d.)

2.4.2 Octopus Deploy Tentacle -ohjelmisto

Jos käyttäjä haluaa pystyttää Octopus Deployn avulla ohjelmiston omalle palvelimelleen, sinne täytyy asentaa Octopus Deploy Tentacle. Tentacle on ohjelmisto, jonka kanssa keskustelemalla Octopus Deployn palvelin voi suorittaa toimenpiteitä käyttäjän palvelimella. Tentacle on yksi asennuskohde (engl. deployment target), johon ohjelmisto voidaan asentaa. Asennuskohteita voidaan järjestellä ja ryhmitellä niille asetettavien roolien perusteella ja asennuskohteella tulee olla vähintään yksi rooli. Windows-ympäristössä Tentaclen voi konfiguroida kahdella tavalla: Listening Tentacle, joka kuuntelee käskyjä Octopus Deployn palvelimelta ja Polling Tentacle, joka puolestaan kyselee Octopus Deployn palvelimelta suoritettavia tehtäviä. (Windows targets n.d.; Deployment targets n.d.; Tentacle communication modes n.d.)

2.4.3 Octopus Deployn projektit ja prosessit

Octopus Deployssa voidaan luoda ja hallita projekteja, jotka pitävät sisällään esimerkiksi ohjelmiston pystytysprosessin, julkaisut ja muita ohjelmiston hallintaan liittyviä asioita. Ohjelmisto voidaan pystyttää erilaisiin ympäristöihin projektien avulla. Projektien pystytysprosessit ovat projektikohtaisesti määritettävissä olevia prosesseja, joilla määritellään mitä toimenpiteitä suoritetaan ohjelmistoa pystytettäessä. Prosessit koostuvat erilaisista askeleista, joita käyttäjä voi luoda ja konfiguroida vastaamaan pystytettävän ohjelmiston tarpeita. Octopus Deploy myös tarjoaa erilaisia askeleiden mallipohjia. (Projects n.d.; Deployment process n.d.)

2.5 IIS-palvelinohjelmisto

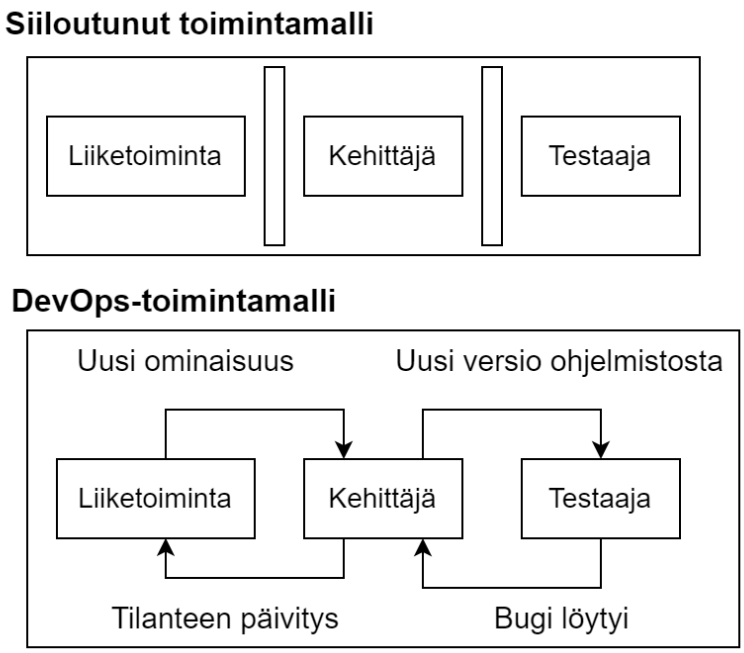
IIS, eli Internet Information Services, on Microsoftin kehittämä pääosin Windows-alustoilla käytettävä verkkopalvelinohjelmisto, jota ajetaan .NET-alustalla. Yksi yleisimmistä IIS:n käyttötarkoituksista on isännöidä ASP.NET -verkkosovelluksia. Sitä voidaan käyttää myös esimerkiksi isännöimään staattisia verkkosivuja tai FTP-palvelimena. IIS:ssä on paljon ominaisuuksia, jotka ovat omina komponentteinaan ja niitä voidaan asettaa päälle tai pois päältä tarpeen mukaan. IIS mahdollistaa myös esimerkiksi Windows autentikoinnin verkkosivuilla. IIS on Windowsin ominaisuus, joka voidaan asettaa päälle Windowsin asetuksista. (Vuollet 2018; Volodarsky 2022.)

3 DevOps, CI/CD ja julkaisuputki

3.1 DevOps käsitteenä

Käsitettä DevOps on vaikea määritellä tarkasti. Sekä Hüttermann (2012) että Mueller (2010) määrittelevät sen kuitenkin lähestulkoon samalla tavalla. Se on heidän mukaansa kokoelma erilaisia ohjelmistokehityksen ajattelutapoja sekä toimintamalleja. Mueller kertoo DevOpsin olevan eräänlainen jatke ketterälle kehitykselle. Hänen mukaansa sen periaatteita ja siihen kuuluvia asioita laajennettaessa sekä hieman muokatessa saadaan melko hyvä määritelmä DevOpsille. Myös Hüttermann vertaa DevOpsia laajennettuun versioon ketterästä kehityksestä. Hüttermannin ja Muellerin mukaan DevOpsiin kuuluu olennaisesti automatisointi, läpimenoajan lyhentäminen ja iteroinnin nopeutus, eri tiimien välinen vuorovaikutus sekä ohjelmiston monitorointi. (Hüttermann 2012; Mueller 2010.)

Hallin (n.d.b.) mukaan DevOps ei ole kuitenkaan ainoastaan toimintamalli, vaan sitä voidaan ajatella omana kulttuurinaan. Hän toteaa DevOps-kulttuurin tuovan tiimille enemmän autonomiaa sekä se jakaa vastuun julkaistavasta tuotteesta koko tiimin kesken. Näiden asioiden lisäksi sekä Hall että Hüttermann (2012) määrittelevät DevOps-kulttuurin lisääntyneellä, aiemmin vahvasti siiloutuneen, projektitiimin henkilöiden välisellä vuorovaikuttamisella, läpinäkyvyydellä ja yhteistyöllä (ks. Kuvio 1). Hallin mukaan myös nopea palautteen saanti on olennainen osa DevOps-kulttuuria, sillä tällä tavoin ohjelmistoa voidaan iteroida ja parantaa nopeammin. Molemmat ovat sitä mieltä, että myös automaatio on yksi olennaisimmista DevOps-kulttuurin asioista, sillä se nopeuttaa ja vakauttaa ohjelmiston kehitysprosessia sekä mahdollistaa nopean palautteen saamisen. (Hall n.d.b.; Hüttermann 2012.)

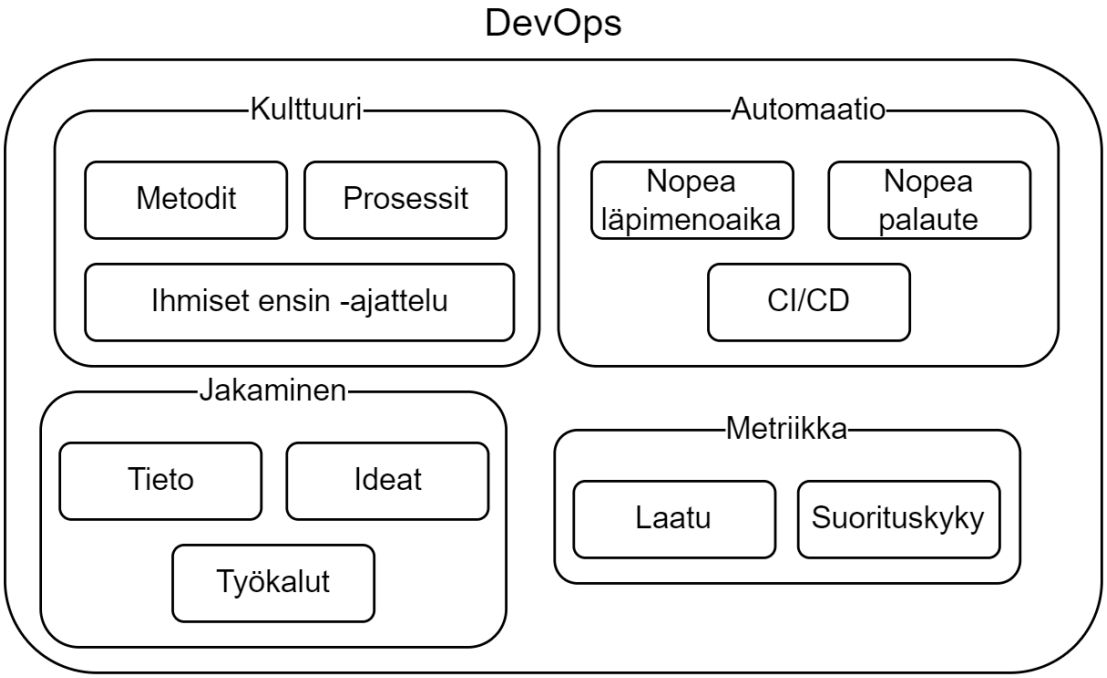


Kuvio 1. Siiloutuneen toimintamallin ja DevOps-toimintamallin erot kuvattuna.

DevOps-toimintamalli on isoksi osaksi erilaisten ihmisten yhteistyötä, mutta sen avulla voidaan myös korjata kestämättömiä toimintatapoja automatisoimalla kehitys- ja julkaisuprosessin vaiheita. Kun prosesseissa toistuvat vaiheet automatisoidaan mahdollisimman pitkälle, työntekijöiden työpanos voidaan kohdistaa paremmin sinne missä sitä enemmän tarvitaan. Tällä tavoin työn tekemisestä saadaan työntekijän näkökulmasta mielekkäämpää ja luodaan parempaa työympäristöä koko tiimille. Automatisointi ja versioiden nopeampi iterointi nopeuttaa myös uuden ohjelmistoversion julkaisua tuotantoon, mikä tuo tiimille näkyvää tehdyn työn tulosta. Tuotannosta tulevan palautteen ansiosta tiimi voi oppia virheistään ja jalostaa sekä omaa prosessiaan että kehitettävää ohjelmistoa entistä paremmaksi. (Abildskov 2013; What is DevOps? N.d.)

DevOps voidaan jakaa Hüttermannin (2012) mukaan neljään eri peruspilariin: kulttuuri, automaatio, mittaus ja jakaminen (ks. Kuvio 2). Sekä Hüttermann että Gene, Humble, Debois ja Willis (2016) ovat sitä mieltä, että ihmiset ensin -kulttuuri on tärkeää DevOpsissa, sillä se vähentää työntekijöiden kokemaa uupumusta, turhautumista ja burnouteja. Genen ja muiden mukaan esimerkiksi turhien prosessien, tehtävien turhien keskeytyksien ja turhan odottelun karsiminen on tärkeä osa DevOps-toimintamallia hyödyntäessä. Gene ja muut painottavat läpimenoajan ja suorituste-

hon kehittämisen tärkeyttä, sillä yksi DevOpsin tarkoituksista on saada kehittäjille nopeaa ja jatkuvaa palautetta. Sekä Mueller (2010) että Hüttermann korostavat myös työkalujen tärkeyttä DevOpsissa. Muellerin ja Hüttermannin mukaan tilanteisiin sopivien työkalujen tehokas hyödyntäminen muun muassa automaatiossa, tiedon jakamisessa, metriikan keräämisessä ja projektia hallinnoidessa on tärkeää. Hüttermann kuitenkin väittää selkeästi määritellyn ja nopean vauhdin mahdollistavan prosessin olevan tärkeämpi kuin pelkät työkalut. Hüttermann toteaa, että vasta sen jälkeen, kun prosessi on määritelty hyvin ja sekä kehittäjät että ylläpitäjät noudattavat sitä, voidaan valita prosessiin sopivat työkalut. (Gene, Humble, Debois & Willis 2016; Hüttermann 2012; Mueller 2010.)



Kuvio 2. Esimerkki millaisista osista DevOps koostuu.

3.2 Jatkuva integraatio, jatkuva julkaisu ja jatkuva käyttöönotto

3.2.1 CI/CD osana DevOps-toimintamallia

DevOps on toimintamallin ja kulttuurin lisäksi muutakin. DevOps-toimintamalliin kuuluu läheisesti käsite CI/CD eli **jatkuva integraatio** (engl. continuous integration) ja **jatkuva julkaisu** (engl. continuous delivery). Jatkuvalla integraatiolla tarkoitetaan prosessin vaihetta, jossa ohjelmiston koodia kehitetään, katselmoidaan ja testataan. Tähän vaiheeseen kuuluu erilaisia automaatioita, kuten

esimerkiksi ohjelmiston koostaminen ja erilaisten testien ajo. Jatkuvalla julkaisulla tarkoitetaan prosessin vaihetta, jossa koodista tehdään julkaistava koosteversio automaattisesti. Koska DevOps perustuu vahvasti automaatioon ja läpimenoajan nopeuttamiseen, jatkuvan integraation ja jatkuvan julkaisun toteuttaminen on olennainen osa sitä. (Hüttermann 2012; What is CI/CD? N.d.; What is DevOps? N.d.)

3.2.2 Jatkuva integraatio

Jatkuvan integraation tarkoitus on pitää versionhallinnan haarat eheinä ja ajan tasalla. Koska yleensä ohjelmistoa kehitetään tiimeissä, muutosten integroiminen päähaaraan tulisi tehdä usein ja automatisoidusti. Kun kehittäjä työntää koodimuutoksensa versionhallintaan, versionhallinnan työkalut ajavat staattista analyysiä, koostavat ohjelmiston ja ajavat yksikkötestejä. Tällöin mahdolliset ohjelmiston bugit löytyvät aikaisessa vaiheessa uuden version julkaisuprosessia. (What is CI/CD? N.d.; CI/CD explained n.d.)

Jatkuvan integraation käyttöön ottamiseksi on olemassa perusasioita, joiden tulisi olla kunnossa ennen kuin se voidaan ottaa projektissa käyttöön. Yksi CI:n käyttöönoton tärkeä perusasia on yhteensopiva versionhallinta. Projektin versionhallinnan tyylin tulisi noudattaa päähaaraan kehitystä (engl. trunk-based development), jolloin kehittäjät työntävät muutoksensa päähaaraan usein ja versionhallinnassa on ainoastaan lyhytikäisiä haaroja. Versionhallintaan tulisi myös laittaa kaikki ne tiedostot, jolla toimiva ohjelmisto saadaan pystytettyä kehittäjän koneelle – aina lähdekoodista dokumentaatioon sekä tietokantaskripteihin. Ellei versionhallinta ole kunnossa, oikeaa jatkuvaa integraatiota ei voida toteuttaa. (Humble & Farley 2010, 56–59; Hall n.d.a.; What is CI/CD? N.d.)

Humblen ja Farleyn (2010) mukaan jatkuvan integraation perustana toimivat oikeanlaisen versionhallinnan lisäksi ohjelmiston koostaminen komentoriviltä, automaattiset testit ja tiimin omistautuminen. Kun ohjelmisto koostetaan automaattisesti komentoriviltä, voidaan varmistaa, että ohjelmisto koostetaan joka kerta samalla tavalla ja jatkuvan integraation ympäristössä voidaan tarkastella esiin nousseita ongelmia. Humblen ja Farleyn mukaan automaattiset testit ovat erittäin tärkeitä, koska ilman testejä voidaan olla varmoja ainoastaan siitä, että ohjelmisto kääntyy ja koostuu oikein. Testeillä varmistetaan, että viimeisin koodimuutos ei hajottanut jotain osaa koodista. Yhtenä tärkeänä asiana Humble ja Farley pitävät myös tiimin omistautumista yhteisille säännöille.

Ilman tiimin itsekuria ja omistautumista jatkuva integraatio ei välttämättä tuo kaikkea potentiaalista hyötyä prosessiin. (Humble & Farley 2010, 56–59.)

3.2.3 Jatkuva julkaisu

Kun koodia ja ohjelmistoa on testattu jatkuvan integraation vaiheessa, eikä siitä löydy ohjelmistoa rikkovia bugeja, siitä voidaan tehdä julkaisuversio. Myös julkaisuvaiheessa ajetaan automaatiotestejä. Näihin automaatiotesteihin kuuluu kattavampia testejä, kuten käyttöliittymätestit, integraatiotestit ja muut mahdolliset automaattiset testit. Jatkuvan julkaisun prosessiin kuuluu myös yleensä ohjelmiston pystyttäminen testiympäristöön. Jatkuvan julkaisun perimmäinen tarkoitus on pitää ohjelmistosta mahdollisimman toimivaa ja kattavasti testattua versiota valmiina julkaistavaksi tuotantoympäristöön. (Hall n.d.a.; What is Continuous Delivery? N.d.)

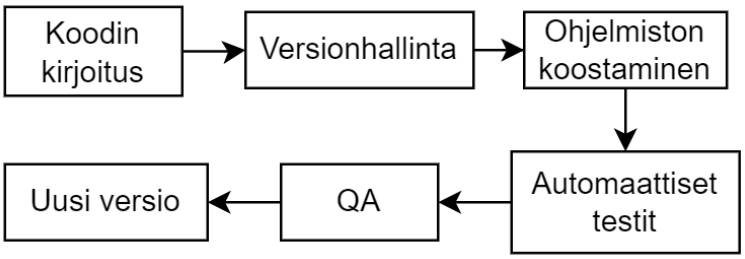
Jatkuvasta julkaisusta on konkreettisia hyötyjä tiimille. Kun uusien versioiden julkaisu on automatisoitu, ne voidaan viedä testiympäristöihin ja jopa tuotantoon asti paljon helpommin ja nopeammin. Kun julkaisun tekoon kuluva aika minimoidaan ja prosessi automatisoidaan, kehittäjät voivat kohdistaa työpanoksensa merkittävämpiin asioihin. Ohjelmisto saadaan testaukseen nopeammin jatkuvan julkaisun avulla, mikä tarkoittaa nopeampaa palautetta ohjelmiston tilasta. Kun ohjelmiston tilasta saadaan nopeammin palautetta, siihen voidaan myös reagoida nopeammin, mikä tarkoittaa nopeampaa toimivan ohjelmistoversion iterointia. Näistä jatkuvan julkaisun asioista seuraa, että myös kehityksen hinta pienenee. Kun monet testit ja ohjelmiston julkaisu on automatisoitu, tietokoneet voivat hoitaa nämä toistuvat tehtävät ja kehittäjille jää aikaa korjata bugeja sekä kehittää uusia ominaisuuksia. Nämä kaikki asiat auttavat omilta osiltaan tuomaan tiimeille positiivisempaa työilmapiiriä ja stressivapaita julkaisuja. (What is Continuous Delivery? N.d.; Humble & Farley 2010, 17–21)

3.2.4 Jatkuva käyttöönotto

Jatkuva käyttöönotto (engl. continuous deployment) voi olla CI/CD-putken osa, mutta sitä ei käytetä kovin yleisesti. Sen tarkoitus on viedä aiemmissa vaiheissa koostettu ja testattu ohjelmisto tuotantoympäristöön automaattisesti. Täysin automaattinen putki koodin kirjoituksesta aina ohjelmiston tuotantoympäristöön viemiseen asti vaatii täsmällistä ja erittäin kattavaa automaattisten testien kokonaisuutta. (Hall n.d.a.; CI/CD explained n.d.)

3.3 Julkaisuputki ohjelmiston kehityksessä

Jatkuva integraatio ja jatkuva julkaisu voivat yhdistyä muiden ohjelmiston kehitykseen liittyvien asioiden kanssa muodostaen kokonaisuuden, jota voidaan kutsua julkaisuputkeksi. Julkaisuputki on siis ohjelmiston matka aina ideasta koodiksi ja sen jälkeen erilaisten testien kautta julkaistavaksi versioksi (ks. Kuvio 3). Myöskään julkaisuputkeen ei ole yhtä ainoaa oikeaa tapaa toteuttaa sitä, vaan toteutustavan ja siinä käytettävien työkalujen valinta jää projektitiimin tai yrityksen vastuulle. Tällaisen julkaisuputken avulla pyritään tuomaan ohjelmiston kehitysprosessiin muun muassa lisää nopeutta, kattavampaa testausta ja jatkuvaa palautteen saamista. (Hall n.d.a.; Humble & Farley 2010 106–107.)



Kuvio 3. Yksinkertainen malli siitä, millainen julkaisuputki voi olla.

Humble ja Farley (2010) määrittelevät joitain hyviä käytänteitä julkaisuputken rakentamiseen, kuten ohjelman koostaminen vain kerran. He perustelevat tätä sillä, että joka kerta ohjelmistoa uudelleen koostaessa siihen voi tulla jokin virhe, joka saattaa hajottaa ohjelmiston. Koostamalla ohjelmiston vain kerran, voidaan olla varmempia siitä, että ohjelmisto toimii niin kuin sen pitääkin. Humble ja Farley pitävät yhtenä hyvänä käytänteenä myös ohjelmiston julkaisua tuotantoympäristön kaltaiseen ympäristöön ja ohjelmiston julkaisua samalla tavalla joka kerta. Kun ohjelmiston julkaisuputki on muuttumaton ja ohjelmisto julkaistaan tuotantoympäristöä vastaavaan ympäristöön usein, oikeaan tuotantoympäristöön julkaisu pitäisi onnistua yhtä helposti. Automaattiset hyväksyntä- ja muut regressiotestit ovat heidän mukaansa tärkeä osa julkaisuputkea. Tutkiva testaus ja käyttöliittymän ulkonäön arviointi jää silloin oikeille ihmisille. (Humble & Farley 2010 113–128.)

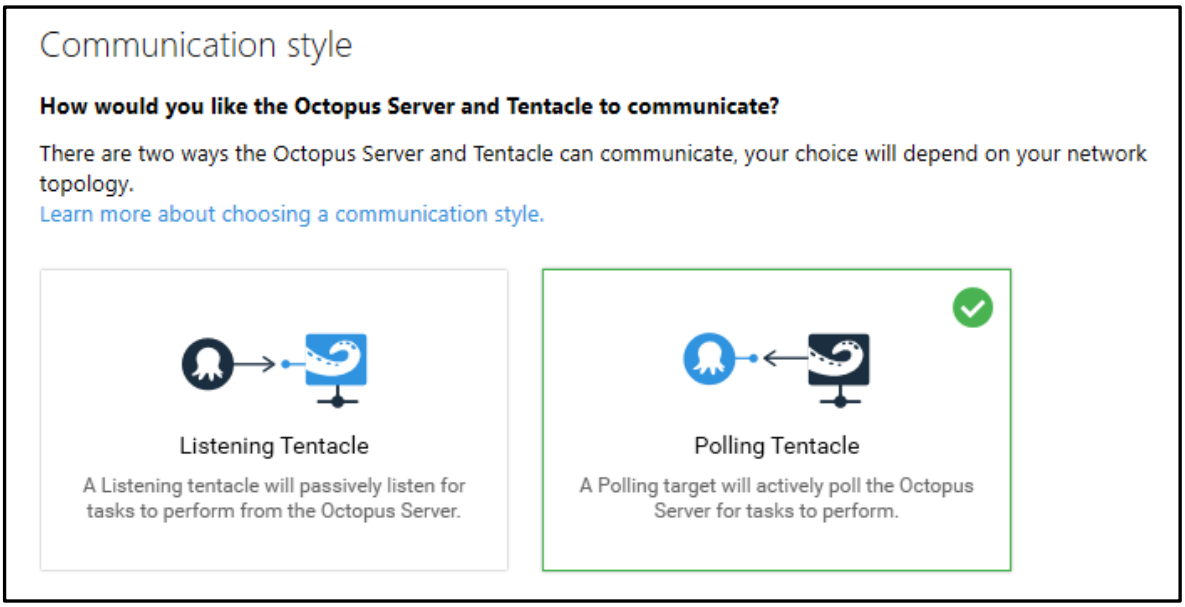
4 Julkaisuputken toteutus

4.1 Toteutustavan valinta

Toteutustapoja julkaisuputken pystyttämiseen on useita, joten niistä täytyi valita sopiva. Muissa yrityksen projekteissa oli käytössä GitHub Actions ja Ocotpus Deploy, joten ne olivat ensimmäisiä vaihtoehtoja. Alkuvaiheessa harkittiin myös Jenkinsiä ja Microsoftin tarjoamia palveluita, kuten Azure Pipelinesia. Vaihtoehtojen tutkinnan ja Octopus Deployn kokeilemisen jälkeen toteutustavaksi valikoitui GitHub Actions ja Octopus Deploy. GitHubin workfloweja oli helppo lähteä tekemään, koska Novin koodia ylläpidetään GitHubissa. Octopus Deploy myös ylläpitää valmiita Actioneita, jotka helpottivat toteutuksen tekemistä. Sekä GitHubin workfloweihin että Octopus Deployn käyttöön löytyi kattavat dokumentaatiot ja esimerkkejä, mikä helpotti toteutustavan suunnittelua. Molempia käytettiin jo yrityksen toisessa projektissa, josta myös pystyi katsomaan apua toteutukseen.

4.2 Octopus Deploy Tentaclen konfigurointi

Jotta ohjelmiston voi asentaa Octopus Deploylla joko automaattisesti tai nappia painamalla, palvelimelle täytyi ensin asentaa Octopus Deploy Tentacle -ohjelmisto ja konfiguroida se. Tentacle asennettiin palvelimelle kuten mikä tahansa muu ohjelmisto käyttämällä Octopus Deployn tarjoamaa asennusohjelmistoa. Kun Tentacle oli asennettu, se täytyi konfiguroida oikeantyyppiseksi Tentacleksi ja sille täytyi antaa Octopus Deployn palvelimen osoite sekä API-avain. Novin tapauksessa palvelimelle konfiguroitiin Polling Tentacle (ks. Kuvio 4), koska palvelin on saavutettavissa ainoastaan VPN:n kautta. Octopus Deployn palvelimelta kysely ei vaatinut palomuurin konfigurointia, kun taas Listening Tenant vaatisi palvelimelle avoimen portin ulkoverkkoon, mitä kautta Octopus Deployn palvelin voisi lähettää käskyjä Tentaclelle. Tentacle toimii aina tietyssä määritellyssä ympäristössä, joten asennetulle Tentaclelle annettiin ympäristöksi test-niminen ympäristö, joka luotiin web-käyttöliittymän kautta. Kun Tentacle oli konfiguroitu, asennus voitiin suorittaa loppuun ja uusi Tentacle tuli näkyviin web-käyttöliittymään. Tentaclelle asetettiin rooliksi novi-test. Rooli tarvittiin, että pystytysprosessia luodessa ajettava prosessin askel voidaan kohdistaa vähintään yhteen ympäristöön käyttämällä roolia tunnisteena.

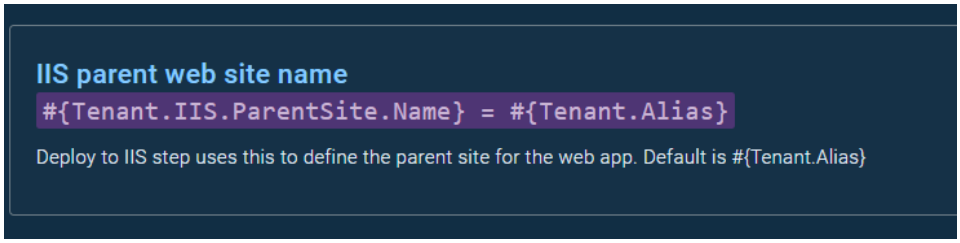


Kuvio 4. Polling Tentaclen konfigurointi.

4.3 Novi desktop projektin ja prosessin luominen Octopus Deployhin

4.3.1 Novi desktopin projekti

Octopus Deployhin täytyi määritellä Novi desktop -projekti, että sille voidaan julkaisuputken aikana luoda julkaisuja, jotka voidaan myöhemmin asentaa palvelimelle. Projektiin täytyi aluksi määritellä muuttujia, sillä julkaisua varten luotava prosessi tarvitsee niitä. Octopus Deployhin luotiin uusi kokoelma muuttujien malleja, johon määriteltiin muun muassa asiakkaan nimi ja ohjelmiston asennuspolku (ks. Kuvio 5). Mallit määriteltiin Octopus Deploy tilakohtaiseksi, jolloin niitä voidaan hyödyntää kaikissa projekteissa. Muuttujille voitiin antaa vakioarvoja ja esimerkiksi IIS:n Parent Site -muuttujalle asetettiin vakioarvoksi asiakkaan nimi. Novi desktop -projektille asetettiin myös muutama projektikohtainen malli, kuten asennuspolku ja IIS:n virtuaalipolku. Kaikkia malleissa määriteltyjä muuttujia voidaan hallita myöhemmin asiakaskohtaisesti. Projektiin luotiin myös Latest-julkaisukanava, jonka avulla voidaan rajoittaa mihin julkaistava versio voidaan asentaa. Projektin asetuksista asetettiin vielä Multi-tenant Deployments vaatimaan ohjelmistolle tenantin, joka voidaan ajatella yhtenä asiakkaana, sitä asennettaessa.



Kuvio 5. Esimerkki muuttujan mallipohjasta, jolla on vakioarvo.

4.3.2 Pystytysprosessin askeleet

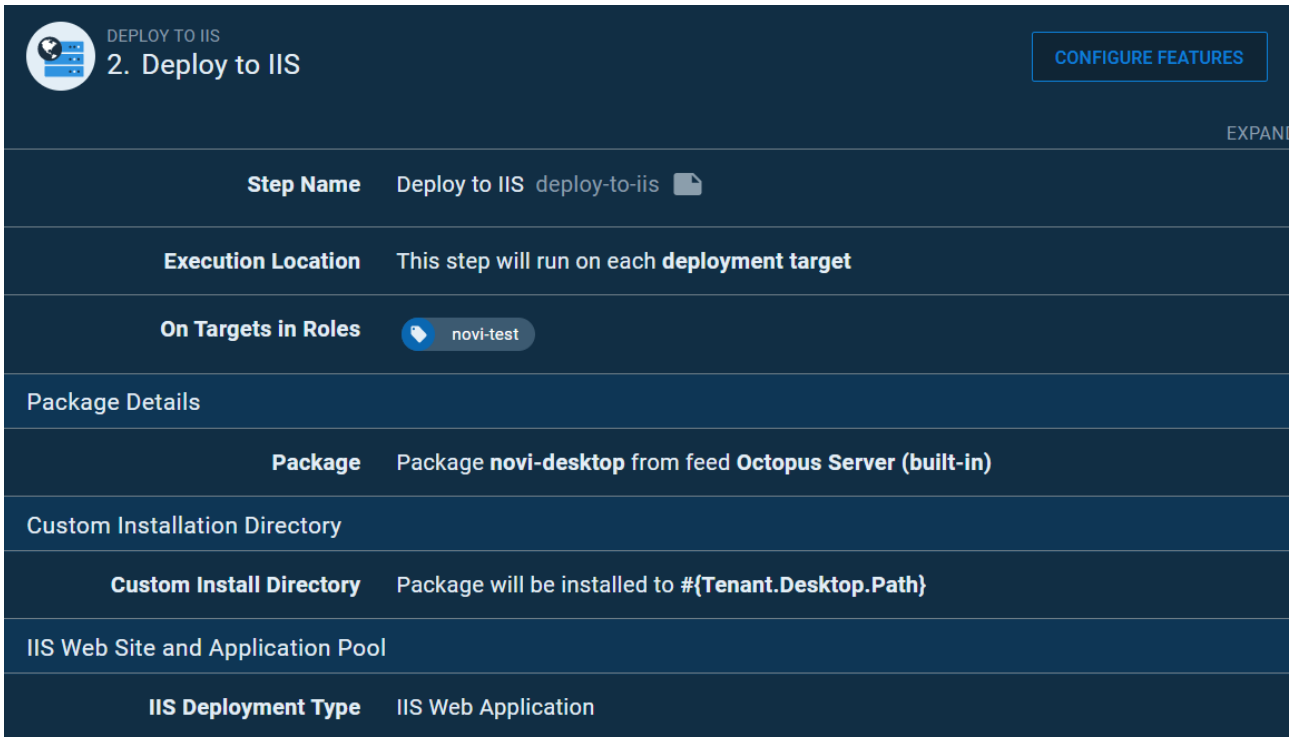
Pystytysprosessin ensimmäinen askel on luoda palvelimelle uudelleenohjaava sivu, jonka tarkoitus on ainoastaan uudelleenohjata käyttäjä juurihakemistosta Novi desktopin alihakemistoon. Uudelleenohjaussivun asennettava paketti on index.html-sivu ja uudelleenohjaava web.config-tiedosto (ks. Kuvio 6) ja se pystytetään palvelimelle kuten mikä tahansa muu verkkosivu. Oikea alihakemisto voidaan myöhemmin määritellä asiakaskohtaisesti muokkaamalla web.config-tiedostoa palvelimella. Askeleelle valittiin pohjaksi Octopus Deployn tarjoama Deploy to IIS -mallipohja, jota muokattiin käyttämään projektin muuttujia tiedostojen sijainnille ja IIS Application Poolin nimelle. Askeleelle määriteltiin myös ympäristöjen roolit, joissa se voidaan ajaa. Koska muita ympäristöjä ei vielä ole, askeleelle asetettiin ympäristöksi aiemmin luotu novi-test-ympäristön rooli. Uudelleenohjaussivun askel jätettiin valinnaiseksi. Koska askel on valinnainen, uutta versiota asentaessa käyttäjä voi itse päättää ohitetaanko uudelleenohjaussivun luonti.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <system.webServer>
        <httpRedirect enabled="true" childOnly="true">
            <add wildcard="/" destination="/desk" />
        </httpRedirect>
    </system.webServer>
</configuration>
```

Kuvio 6. Uudelleenohjaava web.config-tiedosto.

Prosessin toinen askel on Novi desktopin asentaminen palvelimelle. Koska Novi desktop on verkkosivu, myös tälle askeleelle valittiin pohjaksi Deploy to IIS -mallipohja, jota muokattiin käyttämään

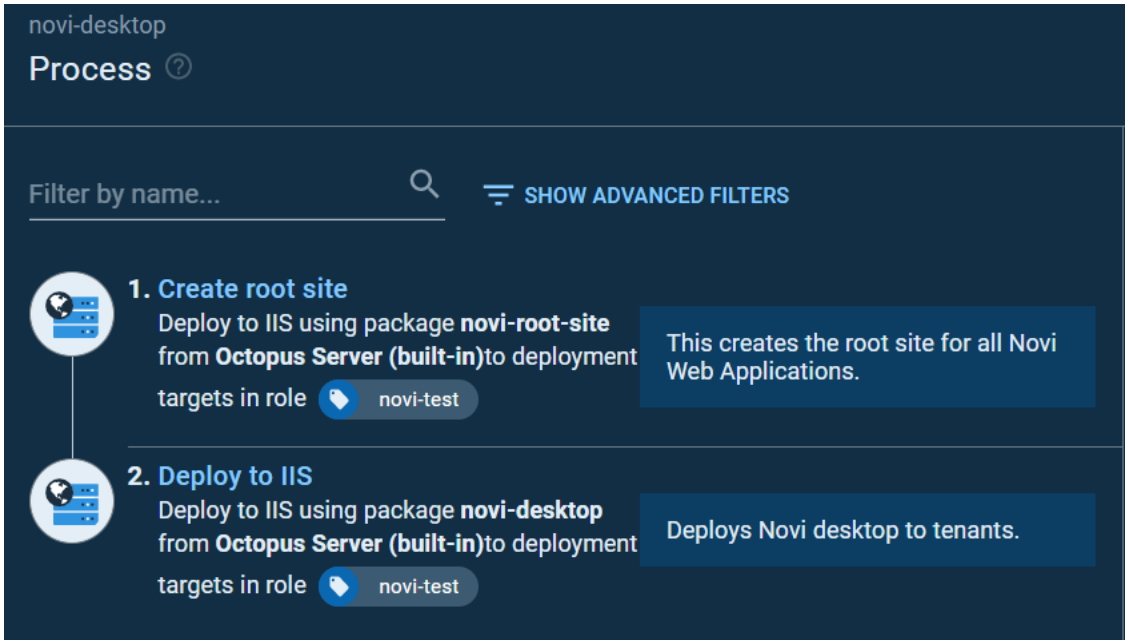
asiakaskohtaisesti määritettävissä olevia muuttujia (ks. Kuvio 7). Toinen askel konfiguroitiin käyttämään samaa IIS Application Poolia uudelleenohjaavan sivun kanssa. Novin asennuskansio konfiguroitiin käyttämään eri muuttujaa kuin uudelleenohjaava sivu, jotta tiedostot voidaan asentaa eri sijainteihin palvelimella. Novin asennusaskeleelle annettiin sama ympäristön rooli kuin uudelleenohjaussivulle. Asennusaskel asetettiin pakolliseksi, jolloin sitä ei voida vahingossa ohittaa prosessia suorittaessa.



Kuvio 7. Osa Deploy to IIS -askeleen konfiguraatiosta, jossa käytetään muuttujia.

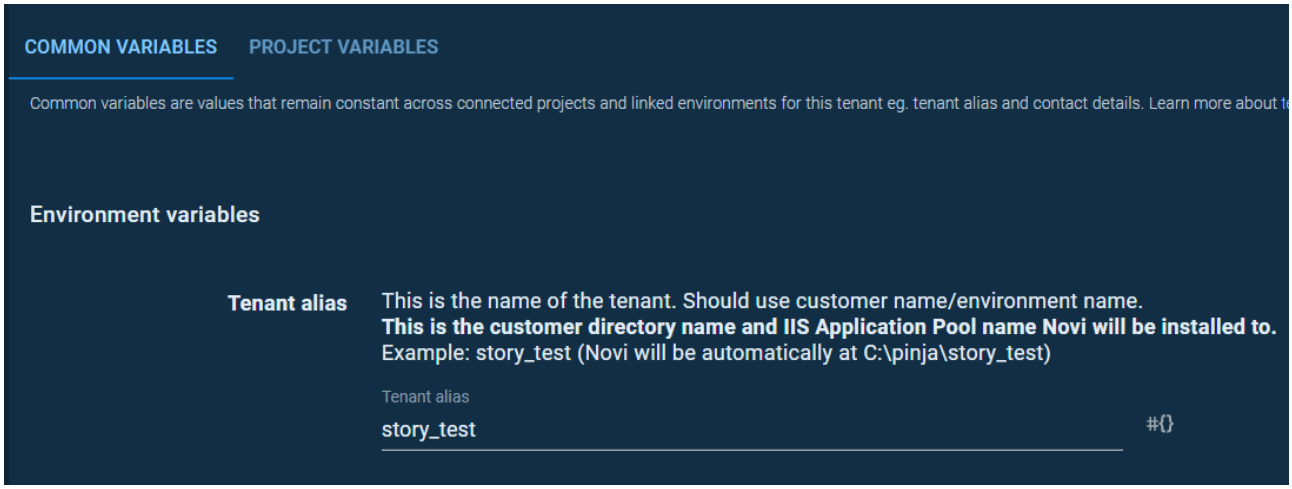
4.3.3 Valmis pystytysprosessi ja Tenantin konfigurointi

Valmiissa Novi desktopin prosessissa (ks. Kuvio 8) on juuri luodut kaksi askelta ja niitä voidaan tarkastella projektin prosessin sivulta. Luodussa prosessissa sallitaan asennukset kaikille novi-test-roolilla oleville ympäristöille, eli uusia asiakasympäristöjä voidaan pystyttää toistaiseksi ainoastaan niille ympäristöille. Novia ei voitu kuitenkaan pystyttää pelkästään konfiguroiduilla askelilla. Koska projektille asetettiin Multi-tenant Deployments -asetus päälle, Octopus Deployhin täytyi luoda ja konfiguroida tenant, jolle Novi desktop voidaan asentaa.



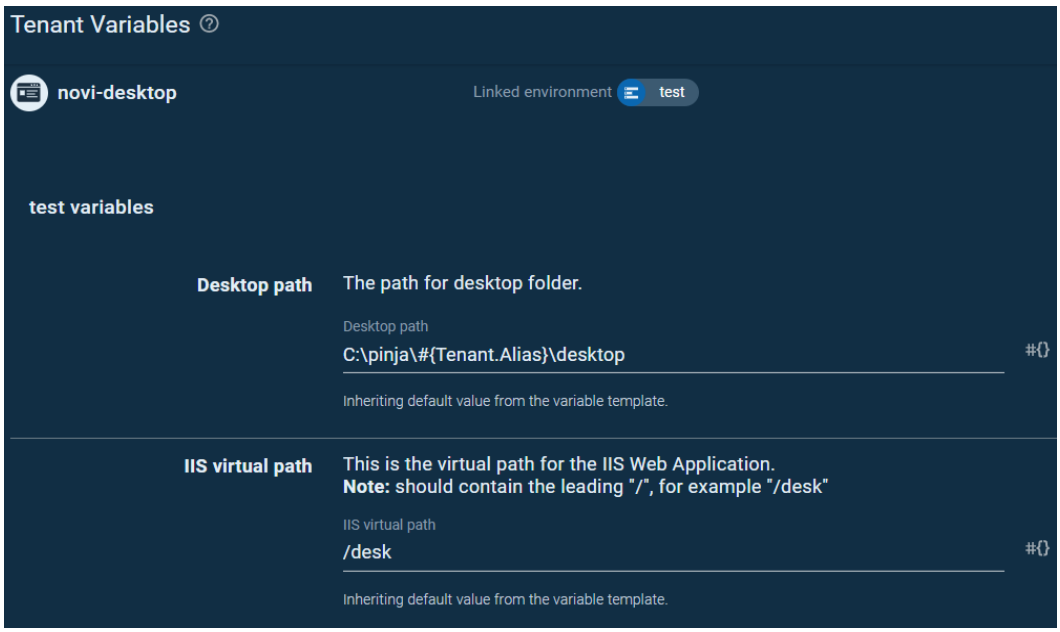
Kuvio 8. Novi desktop -projektin valmis prosessi Octopus Deployssa.

Uusia tenanteja voidaan luoda Octopus Deployn ympäristön Tenants-valikosta, jolloin tenantit näkyvät myös mahdollisille muille ympäristössä oleville projekteille. Uudelle tenantille annettiin nimeksi story-test. Jotta monia ympäristöjä voitaisiin hallita kerralla, luotiin testiympäristöjä varten Octopus Deployn ympäristöön rc-server-niminen Tag Set kertomaan käyttäjälle millä palvelimella tenant sijaitsee. Uuteen Tag Setiin luotiin story-test-tunniste, joka asetettiin juuri luodulle tenantille. Jotta tenant saatiin näkyviin asennusvalikkoon, se täytyi yhdistää aiemmin luotuun projektiin käyttäen tiettyä ympäristöä. Ympäristöksi valittiin test-ympäristö, sillä se oli ainoa ympäristö, mikä luotiin aiemmin. Kun tenant yhdistettiin projektiin käyttäen ympäristöä, aiemmin luodut ympäristökohtaiset muuttujat tulivat näkyviin tenantin konfigurointiin (ks. Kuvio 9). Ympäristökohtaisissa muuttujissa oli pakollisia täytettäviä kenttiä, kuten tenantin nimi (Tenant alias), mitä käytetään automaattisesti projektikohtaisissa muuttujissa.



Kuvio 9. Tenantin ympäristökohtaiset muuttujat.

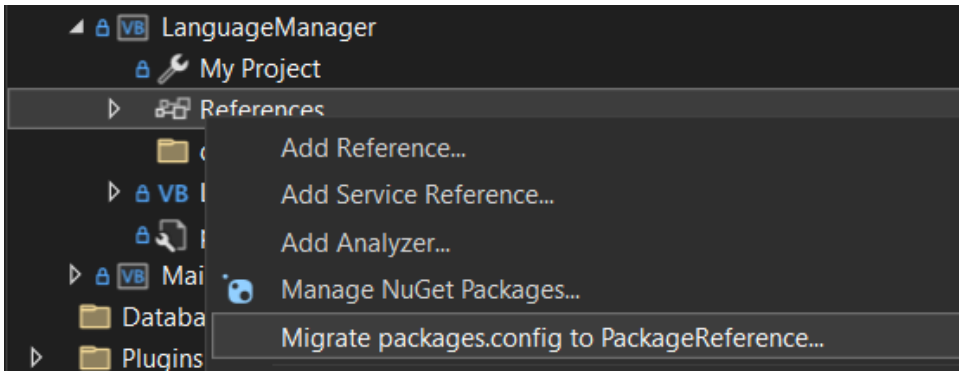
Projektin yhdistämisen jälkeen myös projektikohtaiset muuttujat tulivat näkyviin tenantin konfiguroinnissa (ks. Kuvio 10). Projektikohtaiset muuttujat voitiin erotella vielä ympäristökohtaisesti, mutta niitä oli ainoastaan aiemmin luotu test-ympäristö. Projektikohtaiset muuttujat määriteltiin niitä luodessa sillä tavoin, ettei niitä ole pakollista muokata tenantia konfiguroidessa. Esimerkiksi Novi desktopin polku palvelimella määriteltiin käyttämään tenantin nimeä, mikä on pakollinen tieto. Kun tenantin ympäristö- ja projektikohtaiset muuttujat konfiguroitiin, Octopus Deployn prosessi oli valmis.



Kuvio 10. Tenantin projektikohtaiset muuttujat.

4.4 Novi desktopin koostaminen ja julkaisu komentoriviltä

Koska Novi desktopin kehityksessä ei ollut käytössä minkäänlaista jatkuvaa integraatiota tai jatkuvaa julkaisua, ensimmäinen haaste oli koostaa se komentoriviltä. Versioiden koostaminen ja julkaisu täytyi tehdä käsin Visual Studion kautta. Visual Studio käytti sisäisesti MSBuild-työkalua koostamiseen ja julkaisuun. MSBuild oli ensimmäinen työkalu, jota kannatti lähteä kokeilemaan. Novi desktopissa on useampi projekti, joissa on viitteitä toisiinsa. Koska ne ovat yhdessä solutionissa, projektit joudutaan koostamaan tietyssä järjestyksessä. Kutsumalla MSBuildia komentoriviltä, se ei aluksi pystynyt koostamaan projekteja automaattisesti oikeassa järjestyksessä, joten järjestys täytyi määrätä itse. Projektien järjestyksen määräämiseksi tuli tarkistaa, että .vbproj-tyyppiä olevat projektitiedostot pitävät sisällään viitteet oikeisiin projekteihin. Projektien NuGet-viitteet vaihdettiin samalla packages.config-tyyppisestä hallinnasta PackageReference-tyyppiseen hallintaan. Visual Studio pystyy tekemään migraation pakettienhallinnan tyylien välillä napin painalluksella (ks. Kuvio 11). Novi desktop pystyttiin koostamaan ja julkaisemaan komentoriviltä viitteiden korjauksen jälkeen.



Kuvio 11. Migraatio pakettienhallinnan tyylien välillä Visual Studiossa.

Komentoriviltä koostamista helpottaa myös Visual Studion kautta luotu julkaisuprofiilitiedosto (ks. Kuvio 12). Julkaisuprofiilitiedostoon määriteltiin, että ohjelmisto julkaistaan paikalliseen kansioon ja Any CPU -alustalle. Kansioksi asetettiin Novi desktopin repositorion juurikansiossa oleva novi-release -kansio. Jos kansiossa on tiedostoja edellisen julkaisun jäljiltä, ne asetettiin poistettavaksi uuden julkaisun yhteydessä. Jos kansiota ei ole, se luodaan automaattisesti julkaisun yhteydessä.

Luotu julkaisutiedosto voidaan antaa parametriksi MSBuild-komennolle. Se helpottaa automaattista julkaisua, koska tällä tavoin julkaisutiedostoa voidaan muokata tarvittaessa ilman, että myöhemmin luotavan GitHubin workflowin komentoa tarvitsee muokata.

```
<?xml version="1.0" encoding="utf-8"?>
<!--
This file is used by the publish/package process of your Web project.
You can customize the behavior of this process by editing this MSBuild file.
In order to learn more about this please visit https://go.microsoft.com/fwlink/?LinkID=208121.
-->
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <WebPublishMethod>FileSystem</WebPublishMethod>
    <LastUsedBuildConfiguration>Release</LastUsedBuildConfiguration>
    <LastUsedPlatform>Any CPU</LastUsedPlatform>
    <SiteUrlToLaunchAfterPublish />
    <LaunchSiteAfterPublish>True</LaunchSiteAfterPublish>
    <ExcludeApp_Data>False</ExcludeApp_Data>
    <publishUrl>..\..\..\novi-release</publishUrl>
    <DeleteExistingFiles>True</DeleteExistingFiles>
  </PropertyGroup>
</Project>
```

Kuvio 12. Novi desktopissa käytettävä julkaisutiedosto.

4.5 GitHubin automaattinen workflow

Komentoriviltä koostamisen puute oli ainoa este GitHub workflowin tekemiselle. Kun se ratkaistiin, voitiin lähteä toteuttamaan workflowia, joka koostaa Novi desktopin, tekee siitä zip-tiedoston ja lisää sen Octopus Deployn pakettirekisteriin aina kun kehittäjä lataa koodimuutoksia GitHubiin. Novi desktopin workflow-tiedoston alussa (ks. Kuvio 13) valmistellaan GitHub Actions -virtuaaliympäristö build-nimiselle työvaiheelle. Riveillä 3–6 määritellään, että workflow ajetaan aina kun koodi muuttuu develop-haarassa. Riveillä 8–22 määritellään järjestyksessä workflowin osan nimi, virtuaaliympäristön käyttöjärjestelmä, asetetaan versionhallinnan haara, asetetaan MSBuild PATH-muuttujaan ja asetetaan NuGet PATH-muuttujaan. Näiden toimintojen jälkeen virtuaaliympäristö on valmis ohjelmiston koostamista varten.

```yaml
name: Build and upload to registry

on:
  push:
    branches:
      - develop

jobs:
  build:
    name: Build MaintWebApp
    runs-on: windows-2022

    steps:
      - uses: actions/checkout@v2
        with:
          ref: ${{ github.ref_name }}

      - name: Add MSBuild to PATH
        uses: microsoft/setup-msbuild@v1.1

      - name: Add NuGet to PATH
        uses: NuGet/setup-nuget@v1.0.5

```

Kuvio 13. GitHub Actions -ympäristön valmistelu.

Esitoimenpiteiden jälkeen ohjelmiston koostaminen voidaan suorittaa muutaman komennon avulla. Workflow-tiedoston seuraavassa osiossa (ks. Kuvio 14) määritellään riveillä 24–25 ensin ajettavaksi nuget restore -komento Visual Studion solutioniin. Tämä komento varmistaa, että ohjelmistoa koostaessa kaikkien projektien riippuvuudet ovat saatavilla. Riveillä 27–28 määritellään askel, joka koostaa Novi2Client-projektin. Koska MSBuild ei osannut automaattisesti koostaa Novi2Client-projektia ennen muita, sen koostaminen täytyi määritellä omaksi askeleekseen ennen muita. Riveillä 30–31 koostetaan ja julkaistaan Maint2013 solution, joka pitää sisällään kaikki Novi desktopin projektit. Julkaisu käyttää aiemmin luotua ci_release-julkaisuprofiilia, jossa projektit on määritelty julkaistavaksi release-kansioon repositorion juuripolkuun. Ennen kuin julkaistusta kansiosta voidaan tehdä zip-paketti, sieltä täytyy poistaa kehitykseen tarkoitetut konfiguraatiot ja pohjat. Riveillä 33–37 ajetaan release-kansiossa konfiguraatiot poistava skripti, minkä jälkeen riveillä 39–43 release-kansio asetetaan GitHub Actioneiden välimuistiin. Kun kansio asetetaan välimuistiin, se on saatavilla myös muille askeleille.

```
24        - name: Restore NuGet packages
25          run: nuget restore Maint2013.sln
26
27        - name: Build Novi2Client
28          run: msbuild -restore:False -p:Configuration=Release .
            \Novi2Client\Novi2Client.csproj
29
30        - name: Publish MaintWebApp
31          run: msbuild -m -restore:False -p:DeployOnBuild=True
            -p:PublishProfile=ci_release -p:Configuration=Release .
            \Maint2013.sln
32
33        - name: Delete config files
34          run: |
35            cd .\release\
36            .\000_delete_config_files_from_publish.bat
37            cd ..
38
39        - name: Upload cache
40          uses: actions/cache@v2.1.7
41          with:
42            path: release
43            key: desktop-cache.${{ github.run_number }}
44
```

Kuvio 14. Novi desktopin koostaminen virtuaaliympäristössä.

Seuraavan työvaiheen ympäristö täytyy valmistella ennen kuin paketti voidaan tehdä ja julkaista. Seuraavaksi workflow-tiedostossa (ks. Kuvio 15) määritellään riveillä 45–52 tälle vaiheelle nimi, käyttöjärjestelmä ja käytettävä komentotulkki, joka on tässä tapauksessa bash. Tämä vaihe määritetään olevan riippuvainen edellisestä vaiheesta. Vaihe suoritetaan siis ainoastaan silloin, kun edellinen vaihe on suoritettu kokonaan onnistuneesti. Lisäksi vaihe asetetaan lukemaan salaisia muuttujia (engl. secrets) development-ympäristöstä. Riveillä 54–64 asennetaan virtuaaliympäristöön Octopus Deployn komentorivityökalu ja ladataan aiemmin välimuistiin asetettu release-kansio uuteen ympäristöön.

```yaml
45    upload:
46      name: Package MaintWebApp and upload to registry
47      runs-on: windows-2022
48      needs: build
49      environment: development
50      defaults:
51        run:
52          shell: bash
53
54      steps:
55        - name: Install Octopus CLI
56          uses: OctopusDeploy/install-octopus-cli-action@v1.2.0
57          with:
58            version: 9.0.0
59
60        - name: Download cache
61          uses: actions/cache@v2.1.7
62          with:
63            path: release
64            key: desktop-cache.${{ github.run_number }}
65
```

Kuvio 15. Toisen virtuaaliympäristön valmistelu.

Kun virtuaaliympäristö on valmisteltu, release-kansio voidaan pakata zip-tiedostoon. Workflow-tiedoston loppupuolella (ks. Kuvio 16) luetaan riveillä 66–70 Novi desktopin versiotieto release-kansiossa olevasta version.txt-tiedostosta. GitHub Actions tukee askeleilta saatujen arvojen lukemista, kun vaiheelle asetetaan esimerkiksi rivin 70 kaltainen koodinpätkä. Tämän askel lukee version tiedostosta, poistaa ensimmäisen kirjaimen versiotiedosta ja asettaa versiotiedon askeleen arvoksi. Koska halutun versionumeron edessä on v-kirjain, ensimmäinen kirjain täytyy poistaa myöhempiä askeleita varten. Riveillä 72–74 ajetaan Octopus Deployn komentorivityökalun pack-komento. Komennolla luodaan zip-tiedosto release-kansiosta. Komennon id ja version -asetukset asettavat luodulle paketille nimen, jossa on novi-desktop, aiemmin muodostettu kehityksessä olevan version numero ja uniikki GitHub Actionin suorituksen numero. Komento määritellään tulostamaan tiedot json-muodossa komentotulkkiin, josta tiedot tallennetaan octo-pack-output.json-tiedostoon. Viimeiseksi riveillä 76–80 luetaan luodun paketin nimi json-tiedostosta askeleen arvoksi Pythonia hyödyntäen.

```
66        - name: Read Novi desktop version
67          id: get_novi_version
68          run: |
69            novi_version=$(cat release/version.txt | sed 's/^.//')
70            echo "::set-output name=version::$novi_version"
71
72        - name: Package app
73          run: |
74            octo pack --format=zip --outputFormat=json --id=novi-desktop
              --version=${{ steps.get_novi_version.outputs.version }}.${{
              github.run_number }} --basePath=release >> octo-pack-output.json
75
76        - name: Read Octopus Package version
77          id: get_package_version
78          run: |
79            package_version=$(cat octo-pack-output.json | python -c "import
              sys, json; print(json.load(sys.stdin)['Version'])")
80            echo "::set-output name=version::$package_version"
81
```

Kuvio 16. Novi desktopin pakkaaminen zip-tiedostoon.

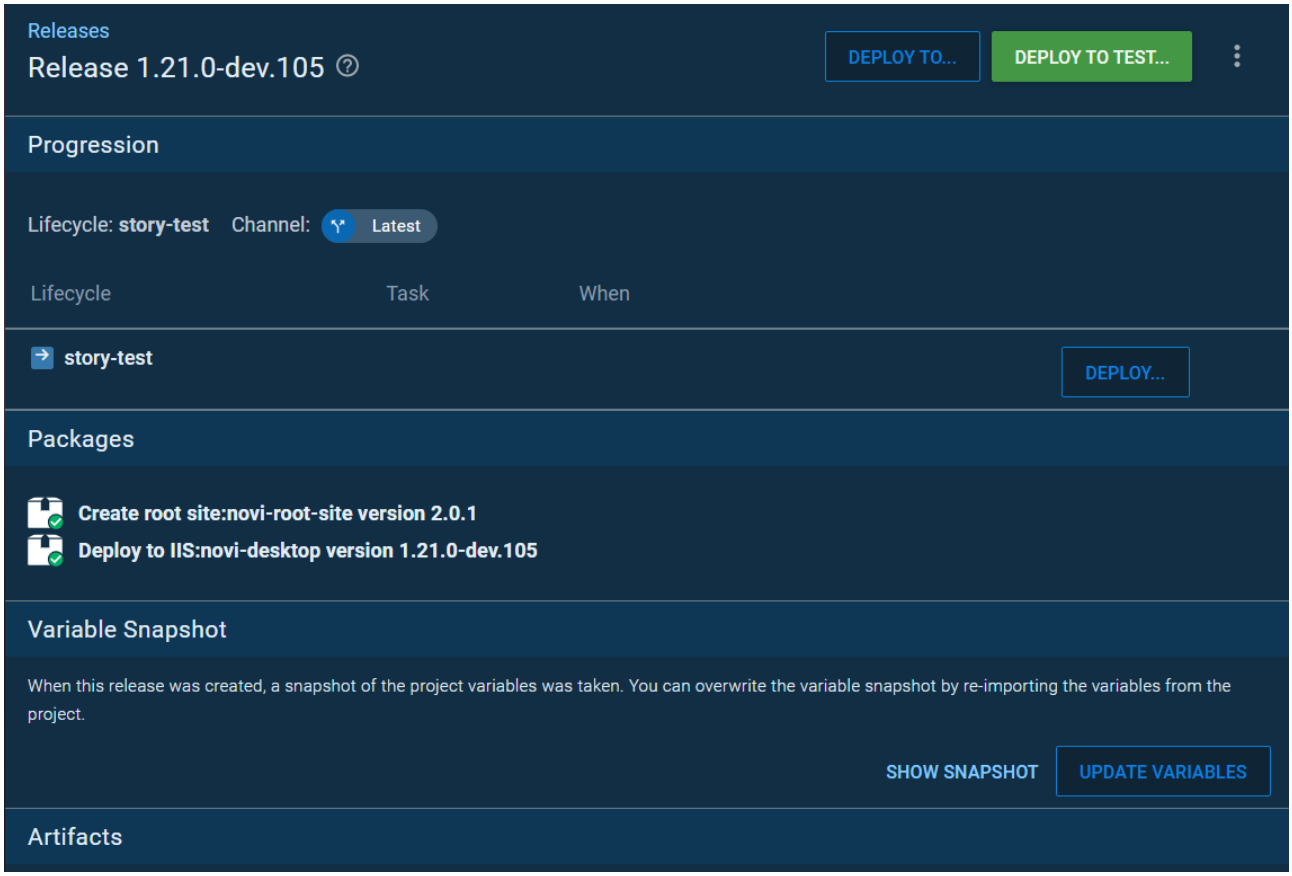
Kun Novi desktop on pakattu zip-tiedostoon, se voidaan ladata Octopus Deployn rekisteriin ja tehdä paketista julkaisu. Octopus Deploy tarjoaa GitHub Actioneita, joilla voidaan tehdä nämä toiminnot helposti. Workflow-tiedoston lopussa (ks. Kuvio 17) ladataan riveillä 82–88 edellisessä askeleessa luotu paketti Octopus Deployn rekisteriin. Octopus Deployn tarjoamalle Actionille täytyy määritellä palvelin, jossa rekisteri sijaitsee, Octopus Deployn ympäristö, API-avain ja ladattavan paketin nimi. Rivillä 89 määritellään, että mikäli samanniminen paketti on jo olemassa, se ylikirjoitetaan. Kun paketti on ladattu rekisteriin, riveillä 91–100 tehdään paketista julkaisu, joka voidaan asentaa haluttuun ympäristöön. Myös jälkimmäinen Octopus Deployn tarjoama Action tarvitsee parametreja: API-avaimen, palvelimen, projektin, Octopus Deployn ympäristön ja paketin, josta julkaisu tehdään. Lisäksi rivillä 99 määritellään, että Action näyttää julkaisun etenemisen GitHubin konsolissa ja rivillä 100 määritellään, että julkaisun numero tulee olla sama kuin paketin versionumero. Osa askeleissa käytettävistä muuttujista on määritelty GitHubin Secretien avulla, sillä esimerkiksi API-avainta ei tule ladata versionhallintaan. GitHub Secretejä käyttäessä arvot voidaan määrittää GitHubiin ja Actionia suoritettaessa muuttujaa voidaan käyttää, mutta sen arvo ei näy suorituslokissa.

```
82        - name: Push Package to Octopus Deploy
83          uses: OctopusDeploy/push-package-action@v1.1.2
84          with:
85            server: ${{ secrets.OCTO_SERVER_URL }}
86            space: ${{ secrets.OCTO_SPACE }}
87            api_key: ${{ secrets.OCTO_API_KEY }}
88            packages: "novi-desktop.${{ steps.get_package_version.outputs.
              version }}.zip"
89            overwrite_mode: OverwriteExisting
90
91        - name: Create and deploy release
92          uses: OctopusDeploy/create-release-action@v1.2.0
93          with:
94            api_key: ${{ secrets.OCTO_API_KEY }}
95            server: ${{ secrets.OCTO_SERVER_URL }}
96            project: ${{ secrets.OCTO_PROJECT }}
97            space: ${{ secrets.OCTO_SPACE }}
98            packages: novi-desktop:${{ steps.get_package_version.outputs.
              version }}
99            progress: true
100           release_number: ${{ steps.get_package_version.outputs.version }}
```

Kuvio 17. Paketin lataaminen Octopus Deployn rekisteriin ja julkaisun teko.

4.6 Novi desktopin asennus palvelimelle

Kun Octopus Deployhin saatiin julkaistua Novi desktopin versio, se voitiin asentaa palvelimelle. Aiemmin konfiguroidun pystytysprosessin ajamiseksi täytyi etsiä Novi desktopin julkaisuista viimeisin julkaisu ja avata se klikkaamalla sitä. Kun julkaisu on avattu, se voidaan asentaa eri ympäristöihin, joita toteutushetkellä oli vain yksi. Pystytysprosessi voitiin suorittaa helposti klikkaamalla julkaisulla näkyvää vihreää nappia (ks. Kuvio 18), joka täytti automaattisesti aiemmin luodun ympäristön pystytyksen kohteeksi. Kun pystytysprosessi aloitettiin, Octopus Deploy näytti joitain lokitietoja pystytykseen liittyvistä asioista ja lopuksi onnistumisen. Palvelimelta tarkistettaessa voitiin huomata, että pystytys oli mennyt kuten konfiguroitu: IIS Application Pool, kansiot sekä Novi desktopin tiedostot löytyivät palvelimelta oikeista paikoista. Koska Novi desktop tarvitsee toimiakseen myös tietokannan, sellainen täytyi luoda pystytetylle ympäristölle. Tässä työssä kopioitiin jo olemassa olevan testiympäristön tietokanta uudelle ympäristölle, sillä testidataa luovia skriptejä tai ohjelmia ei ollut käytettävissä.



Kuvio 18. Novi desktopin release Octopus Deployssa.

5 Tulokset

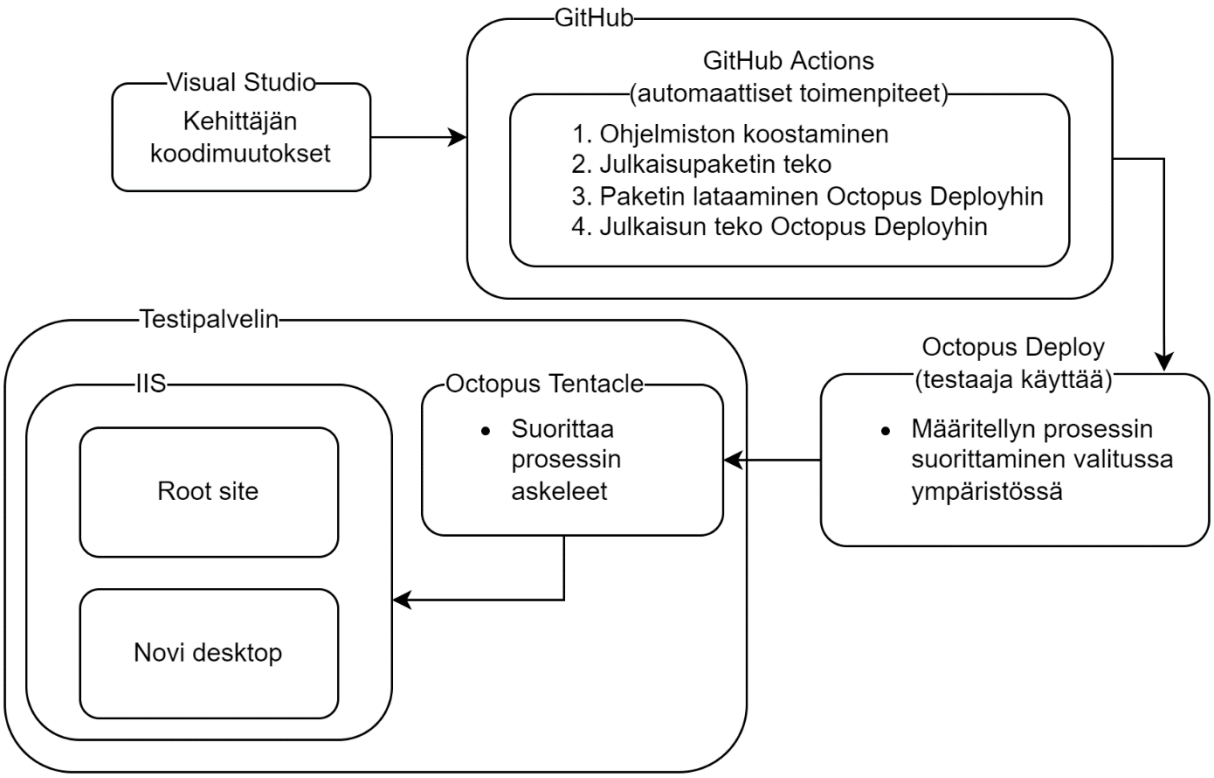
Työn tulokseksi saatiin vastattua alussa esitettyihin tutkimuskysymyksiin ja käytännön toteutus yhdenlaisesta julkaisuputkesta. Työn tuloksena saatiin myös kehitettyä toimeksiantajan ohjelmiston kehitysprosessia ja lisättiin tiedon liikkumista kehittäijen ja testaajien välillä.

Tuloksena saatiin tietoa mitä DevOps ylipäätään tarkoittaa ja mitä se pitää sisällään. DevOps on vaikeasti määriteltävissä oleva kokoelma erilaisia toimintamalleja. Yksi tärkeimmistä DevOpsin asioista on automaatio ja työkalujen tehokas hyödyntäminen. DevOpsissa kiinnitetään huomiota myös läpimenoajan lyhentämiseen ja prosessin kehittämiseen. DevOps on myös ihmiskeskeinen tapa kehittää ohjelmistoa, millä voidaan lisätä tiedon liikkumista eri tiimien välillä ja vähentää ihmisten kokemaa stressiä prosessista. DevOpsia hyödyntäessä ei ole yhtä oikeaa tapaa toteuttaa sitä, vaan tiimi voi itse valita sopivat prosessit ja työkalut.

Tuloksena saatiin myös vastaus siihen mitä jatkuva integraatio ja jatkuva julkaisu tarkoittavat. Jatkuva integraatio auttaa kehittäjiä pitämään versionhallinnan haaran ehjänä. Jatkuvan integraation keskeisin asia on ajaa automaattitestit haarassa olevaan koodiin ja koostaa ohjelmisto. Jatkuva julkaisu on jatkuvan integraation jälkeen tapahtuvat asiat, eli koostetun ohjelmiston julkaisu. Jatkuvan julkaisun vaiheessa ohjelmistoa voidaan testata lisää. Jatkuvan integraation ja jatkuvan julkaisun tarkoituksena on saada DevOps-periaatteiden mukaisesti nopeaa palautetta kehittäjälle ohjelmiston tilasta.

Yksi tutkimuskysymyksistä oli selvittää mikä julkaisuputki on. Työn tuloksena saatiin tietoa, että julkaisuputkella tarkoitetaan jatkuvan integraation ja jatkuvan julkaisun osien yhdistämistä yhdeksi kokonaisuudeksi. Julkaisuputken tarkoitus on automatisoida koodin koostaminen, testaaminen ja julkaisu, mikä nopeuttaa ohjelmiston kehitysprosessia ja läpimenoaikaa. Julkaisuputkella voidaan tarkoittaa uuden ohjelmistoversion luomista automaattisesti, mutta siinä voidaan hyödyntää myös jatkuvaa käyttöönottoa, jolloin ohjelmisto myös otetaan käyttöön automaattisesti.

Tutkimuskysymyksenä oli myös selvittää millainen julkaisuputki on käytännössä. Käytännön toteutuksen tuloksena saatiin julkaisuputki (ks. Kuvio 19), joka käynnistyy aina kun GitHubissa olevaan kehityshaaraan tulee muutoksia. Näin saadaan ohjelmistosta versio testaajille mahdollisimman aikaisin ennen julkaisua, jolloin bugien tai huomioiden korjaaminen on helpompaa. Julkaisuputki toteutus otettiin käyttöön toimeksiantajalla ja toteutuksesta saatiin lähes heti palautetta kehittäjätiimiltä ja testaajilta. Saatu palaute julkaisuputkesta oli pääosin positiivista, mutta joitain korjattavia asioita löytyi ja ne kirjattiin ylös jatkokehitystä varten. Koska testaajat saavat koodimuutokset testaukseen nopeammin, kehittäjät muistavat muutokset paremmin. Julkaisuputki myös tasaa testaajien kuormaa, sillä testaukseen tulee kerralla paljon vähemmän asioita kuin julkaisuvaiheessa olevasta versiosta tulisi. Julkaisuputken käyttöönotossa saatiin myös parannettua kehittäjien ja testaajien välistä kommunikointia ja parannettiin kehitysprosessia.



Kuvio 19. Työssä toteutetun julkaisuputken kuvaus.

Julkaisuputken käyttöönoton yhteydessä kehityksessä olevat tiketit saivat uusia tiloja: Waiting for PO, Ready for QA, QA Testing ja Fix it. Näiden uusien tilojen avulla saatiin tieto testaajille siitä, että onko tiketti jo valmis testattavaksi koodikatselmoinnin jälkeen. Tiloilla saatiin myös tieto kehittäjille siitä, missä vaiheessa tiketin testaus on ja onko toteutuksessa jotain korjattavaa tai huomioitavaa. Waiting for PO -tilalla voidaan viestiä tarkemmin, että tiketillä tai toteutuksessa on jotain, joka vaatii tuoteomistajan huomiota. Näiden uusien tilojen avulla saatiin lisättyä tiedon siirtymistä kehittäjien, testaajien sekä muiden prosessiin osallistuvien välillä.

6 Pohdinta

Opinnäytetyön tavoitteena oli vastata tutkimuskysymyksiin ja toteuttaa vähintään pohja julkaisuputkelle Novi-projektiin. Julkaisuputkea tai sen pohjaa voitaisiin käyttää avuksi ohjelmiston testauksessa tai pohjaa voitaisiin jatkokehittää toimivaksi. Yleisellä tasolla tavoite oli luoda automatisointia ja tutkia sen hyötyjä. Tuloksena saatu julkaisuputki sekä tieto DevOpsista ja CI/CD:stä olivat

hyödyllisiä toimeksiantajalle. Koska palaute julkaisuputkesta oli pääosin positiivista, toteutus koettiin hyvänä asiana Novi-tiimissä.

Julkaisuputken toteutukseen ei tehty tarkempaa suunnitelmaa tai kuvausta, vaan julkaisuputken vaatimuksia käytiin läpi vain pääkohtien avulla. Ainoat määrittelyt ja vaatimukset julkaisuputkelle olivat, että testaajille täytyy saada viimeisin kehityksessä oleva versio silloin, kun kehittäjä lataa koodimuutoksensa GitHub-repositorion develop-haaraan. Aluksi vaihtoehtona pohdittiin myös oman URL:n luomista pystytettävälle ympäristölle, mikäli ympäristö pystytettäisiin kehittäjän luomasta pull requestista GitHubissa. Ennen toteutusta päätettiin kuitenkin, että yksi ympäristö testaajalle riittää. Octopus Deployn käytöstä tai sen vaatimuksista ei ollut tietoa ennen toteutuksen aloitusta, joten tarkemman suunnitelman laatiminen olisi ollut hankalaa.

Vaikka toisessa yrityksen projektissa käytettiin jo Octopus Deployta, sen käyttöönotto Novi-projektiin oli melko työlästä. Alkuvaiheessa kului paljon aikaa Novin koostamiseen komentoriviltä ja Octopus Deployhin perehtymiseen. Alkuvaikeuksien jälkeen toteutus alkoi helpottumaan ja ensimmäisiä versioita julkaisuputkesta voitiin testata jo melko nopeasti. Julkaisuputken toteutuksen aikana löytyi kuitenkin monia huomioita ja ongelmia, joita käytiin muun muassa kokeneempien kehittäjien kanssa läpi. Osa näistä ongelmista kirjattiin ylös tiketeille jatkoa varten, koska ne eivät suoranaisesti liittyneet toimeksiantoon. Jatkokehitystä on jo suunnitteilla nousseiden ongelmien ratkaisemiseksi, sillä syntynyttä julkaisuputkea tullaan varmasti hyödyntämään jatkossakin.

Käytännön prosessi eroaa jonkin verran siitä mitä se teoriassa voisi ja mahdollisesti kannattaisi olla. Automaation pohjaksi kuului muun muassa päähaaraan tapahtuva kehitys ja erilaiset automaattiset testit, joita Novi desktopissa ei vielä toteutettu. Myös Novin pystytysprosessissa on automaation ansiosta toteutushetkellä eroja. Koska testiympäristöön Novi voidaan pystyttää Octopus Deployn avulla, tuotantoympäristöön pystyttämistä ei voida harjoitella ja varmentaa prosessin toimintaa. Yhtäläisyyksiäkin teoriaan löytyy, sillä julkaisuputken käyttöönotto lisäsi prosessin läpinäkyvyyttä ja informaation liikkumista eri projektissa olevien henkilöiden välillä. Projektissa huomattiin myös automaation ja tarinatestauksen hyötyjä, sillä nyt esimerkiksi uusista ominaisuuksista löytyviä huomioita voitiin korjata paljon ennen julkaisuversioita.

Koska julkaisuputkea on tarkoitus käyttää aktiivisesti tulevaisuudessakin, siihen liittyviä jatkokehitysideoita kirjattiin opinnäytetyön toteutuksen aikana ja käyttöönoton jälkeen. Heti Novi desktopin julkaisuputken käyttöönoton jälkeen huomattiin, että samanlainen julkaisuputki kannattaa toteuttaa myös Novi backendiin sekä Novi mobileen. Tällöin kaikki Novin osat voidaan pystyttää testiympäristöön Octopus Deployn avulla lukuunottamatta tietokantaa. Myös erityisesti automaattitestit sekä varmuuskopioiden ottaminen koettiin tärkeiksi jatkokehitysideoiksi.

Toteutuksen alkuvaiheessa tiedostettiin, että yksi toteutuksen ongelmista on testiympäristön tietokannan hallinta. Koska migraatiotiedostoja tehdään lisäämällä tarvittavat SQL-kyselyt käsin tiedostoon, niissä ei toteutushetkellä ollut tietoa versiosta tai kehitystiketistä, jolla tiedosto on luotu. Migraatiotiedostot yhdistetään uutta versiota luodessa, jolloin migraatiotieto lisätään jokaiseen tiedostoon. Jatkokehityksenä tietokannan hallintaan on pohtia miten migraatiotiedostoja jatkossa toteutetaan sillä tavoin, että ne voidaan ajaa Octopus Deployn avulla sekä testi- että tuotantoympäristöjen tietokantoihin. Testiympäristön tietokannan hallintaan pohdittiin myös automaattisen datan generointia koodilla. Silloin testaaja voisi konfiguroida testiympäristön vielä tarkemmin haluamallaan tavalla.

Julkaisuputken toteutuksen aikana huomattiin myös muutamia pienempiä jatkokehityskohteita, kuten Octopus Deployn muuttujien hyödyntämistä uudelleenohjaavan sivun tiedostoissa. Muuttujat helpottavat pystytystä jatkossa, sillä silloin uudelleenohjaava sivu ja Novi desktop käyttävät varmasti samaa alihakemistoa. Koska Novin julkaisuja luodaan GitHubiin, jatkokehityksenä pohdittiin automaattisesti koostetun ohjelmiston paketin lisäämistä juuri luodulle GitHubin julkaisulle. Automaattinen paketin lisäys vähentäisi huomattavasti kehittäjän tekemän virheen mahdollisuutta ja vähentäisi myös hieman julkaisun tekoon kuluvaa aikaa.

Octopus Deploy koettiin tarpeelliseksi ja työtä helpottavaksi työkaluksi, minkä takia Octopus Deploy olisi tarkoitus ottaa käyttöön myös tuotantoympäristöihin. Octopus Deployn avulla tuotantoympäristöjen päivitys helpottuisi ja ennen kaikkea nopeutuisi huomattavasti. Ennen kuin Octopus Deploy voidaan kuitenkaan ottaa käyttöön tuotantoympäristöihin, täytyy ensin ratkaista huomatut haasteet ja toteuttaa ainakin osa jatkokehitysideoista, jotta voidaan olla varmoja prosessin toimivuudesta tuotantoympäristöissä.

Opinnäytetyön aihe oli hyvin valittu, sillä DevOps on ajankohtainen aihe ja tuloksena syntynyt julkaisuputki oli toimeksiantajan mukaan tarpeellinen ja hyvin tehty. Julkaisuputken toteutus oli tavoite, sillä sille oli aidosti tarvetta ja sen toteutus oli mielenkiintoista. Vaikka projektissa ei tällä hetkellä toimita kaikkien DevOps ja CI/CD -periaatteiden mukaisesti, tieto teoriasta on varmasti hyödyllistä jatkossa. Myös aiheen rajaus onnistui hyvin, sillä työmäärä oli sopiva. Tulevaisuudessa olisi hyvä tarkastella kehitysprosessia ja pohtia kannattaisiko työssä etsitystä tiedosta ottaa asioita myös Novin kehitysprosessiin.

Lähteet

Abildskov, J. 2013. Mitä on DevOps? Blogikirjoitus DevOpsista Eficode Oy:n sivuilla. Viitattu 22.1.2022. https://www.eficode.com/fi/blog/mita-on-devops.

Automate your workflow from idea to production. N.d. GitHub Actionsin esittelysivu GitHubin verkkosivuilla. Viitattu 6.8.2022. https://github.com/features/actions.

Buchanan, I. N.d. Benefits of DevOps. Artikkeli DevOpsin hyödyllisyydestä Atlassianin verkkosivuilla. Viitattu 23.4.2022. https://www.atlassian.com/devops/what-is-devops/benefits-of-devops.

CI/CD explained. N.d. GitHubin artikkeli, jossa kerrotaan jatkuvasta integroinnista, jatkuvasta julkaisusta ja jatkuvasta toimituksesta. Viitattu 5.2.2022. https://resources.github.com/ci-cd/.

Complex deployments made easy. N.d. Esittely Octopus Deploysta ja sen työkaluista Octopus Deployn verkkosivuilla. Viitattu 6.8.2022. https://octopus.com/.

Deployment process. N.d. Dokumentaatiota ohjelmiston pystytysprosessiin Octopus Deployn verkkosivuilla. Viitattu 6.8.2022. https://octopus.com/docs/projects/deployment-process.

Deployment targets. N.d. Asennuskohteiden dokumentaatio Octopus Deployn verkkosivuilla. Viitattu 6.8.2022. https://octopus.com/docs/infrastructure/deployment-targets.

Gene, K., Humble, J., Debois, P. & Willis, J. 2016. The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations. Portland: IT Revolution Press. Viitattu 29.10.2022. https://janet.finna.fi/, Skillsoft Books ITPro.

Getting started. N.d. Yleistä dokumentaatiota Octopus Deploysta. Viitattu 6.8.2022. https://octopus.com/docs/getting-started.

GitHub. N.d. GitHubin verkkosivut. Viitattu 19.2.2022. https://github.com/.

Hall, T. N.d.a. DevOps Pipeline. Artikkeli DevOps-putkesta Atlassianin verkkosivuilla. Viitattu 6.2.2022. https://www.atlassian.com/devops/devops-tools/devops-pipeline.

Hall, T. N.d.b. What is DevOps Culture? Artikkeli DevOps-kulttuurista Atlassianin verkkosivuilla. Viitattu 27.2.2022. https://www.atlassian.com/devops/what-is-devops/devops-culture.

Historia. N.d. Artikkeli Pinjan historiasta Pinja Group Oy:n verkkosivuilla. Viitattu 28.3.2022. https://pinja.com/pinja/historia.

Humble, J. & Farley, D. 2010. Continuous Delivery: reliable software releases through build, test and deployment automation. Pearson Education.

Hüttermann, M. 2012. DevOps for Developers. Berkeley: Apress. Viitattu 16.3.2022. https://janet.finna.fi/, Skillsoft Books ITPro.

Mueller, E. 2010. What Is DevOps? Blogikirjoitus The Agile Admin -verkkosivulla. Yksityistä verkkosivua julkaisevat asiantuntijat Ernest Mueller, James Wickett, Karthik Gaekwad ja Peco Karayanev. Viitattu 22.1.2022. https://theagileadmin.com/what-is-devops.

Novi by Pinja. N.d. Novin esittely Pinja Group Oy:n verkkosivuilla. Viitattu 19.2.2022. https://pinja.com/palvelut/valmistava-teollisuus/novi.

Ohjelmistoyhtiö Pinjalle uusi kansainvälinen omistaja. 2021. Tiedote omistajanvaihdoksesta Pinja Group Oy:n verkkosivuilla. Viitattu 28.2.2022. https://pinja.com/uutiset/ohjelmistoyhtio-pinjalle-uusi-kansainvalinen-omistaja.

Palvelut. N.d. Esittely Pinja Oy:n tarjoamista palveluista sen verkkosivuilla. Viitattu 11.4.2022. https://pinja.com/palvelut.

Pinja Monthly tammikuu. 2022. Pinjan sisäinen kuukausipalaveri.

Preston-Werner, T. 2008. GitHub Turns One! Blogikirjoitus GitHub:in blogissa. Viitattu 27.2.2022. https://github.blog/2008-10-19-github-turns-one/.

Projects. N.d. Projektien dokumentaatiota Octopus Deployn verkkosivuilla. Viitattu 28.8.2022. https://octopus.com/docs/projects.

Tentacle communication modes. N.d. Octopus Deploy Tentaclen kommunikaatiotyyppien dokumentaatio Octopus Deployn verkkosivuilla. Viitattu 6.8.2022. https://octopus.com/docs/infrastructure/deployment-targets/windows-targets/tentacle-communication.

Toimialat. N.d. Pinjan eri toimialojen esittelyä Pinja Group Oy:n verkkosivuilla. Viitattu 28.3.2022. https://pinja.com/toimialat.

Understanding automation. 2018. Artikkeli automatisoinnista Red Hatin verkkosivuilla. Viitattu 23.4.2022. https://www.redhat.com/en/topics/automation.

Understanding GitHub Actions. N.d. GitHub Actionsin perusteita GitHubin verkkosivuilla. Viitattu 6.8.2022. https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions.

Volodarsky, M. 2022. IIS Web Server Overview. Joitain yleistietoja IIS-palvelinohjelmistosta Microsoftin sivuilla. Viitattu 8.10.2022. https://learn.microsoft.com/en-gb/iis/get-started/introduction-to-iis/iis-web-server-overview.

Vuollet, P. 2018. What is IIS? Artikkeli IIS:stä Stackifyn verkkosivuilla. Viitattu 6.8.2022. https://stackify.com/iis-web-server/.

Welcome to the Visual Studio IDE. 2022. Esittely Visual Studiosta Microsoftin verkkosivuilla. Viitattu 28.5.2022. https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022.

What is CI/CD? 2018. Artikkeli jatkuvasta integroinnista, jatkuvasta julkaisusta ja jatkuvasta toimituksesta Red Hatin verkkosivuilla. Viitattu 6.2.2022. https://www.redhat.com/en/topics/devops/what-is-ci-cd.

What is Continuous Delivery? N.d. Kirjoitus jatkuvasta toimituksesta Continuous Delivery -verkkosivulla. Sivustoa ylläpitää Jez Humble. Viitattu 8.2.2022. https://continuousdelivery.com/.

What is DevOps? N.d. Artikkeli GitHubin sivuilla. Viitattu 6.2.2022. https://resources.github.com/devops.

What Is GitHub? A Beginner's Introduction to GitHub. 2021. Artikkeli GitHubista Kinstan verkkosivuilla. Viitattu 19.2.2022. https://kinsta.com/knowledgebase/what-is-github/.

Windows targets. N.d. Windows-asennuskohteiden dokumentaatio Octopus Deployn verkkosivuilla. Viitattu 6.8.2022. https://octopus.com/docs/infrastructure/deployment-targets/windows-targets.