Jatkuvan integraation optimointi Frends:lle

Riku Virtanen

Haaga-Helia ammattikorkeakoulu

Tradenomi

Opinnäytetyö

2022

Tiivistelmä

Tekijä(t)
Riku Virtanen

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Jatkuvan integraation optimointi Frends:lle

Sivu- ja liitesivumäärä
25 + 6

Opinnäytetyön tavoitteena on tutkia CI:n eli Jatkuvan Integraation toimintamalleja ja parhaita käytäntöjä. Työssä selitetään jatkuvan integraation tarkoitus käsitteenä sekä avataan sen toimintaperiaatetta. Työssä myös tutustutaan kolmeen CI-järjestelmään ja niiden toimintaperiaatteita vertaillaan keskenään.

Työn toiminnallinen osuus toteutetaan toimeksiantona Frends:lle ja se koostuu kahdesta osasta. Ensimmäisessä osassa toteutetaan nykytilakartoitus, jonka tavoitteena on laatia dokumentaatio toimeksiantajalle sen projektissa käytettävästä CI-järjestelmästä sekä siinä käytettävistä toimintamalleista ja käyttötapauksista. Toiminnallisen osuuden toisessa osassa toteutetaan kehitysehdotus CI-järjestelmän käytöstä. Kehitysehdotuksessa peilataan tietoperustasta löytyneitä toimintamalleja sekä käytetään CI-järjestelmien vertailutuloksia hyödyksi.

Työn lopputuloksena saadaan luotua toimeksiannolle dokumentaatio CI:n käytöstä sekä luodaan valmis kehitysehdotus, jonka perusteella toimeksiantaja pystyy muokkaamaan toimintaansa ja kehittämään projektin tehokkuutta sekä saavuttamaan standardisoitu toimintamalli.

Asiasanat
CI, CT, CD, Jenkins, TeamCity, Github Actions, Integraatio, testaus, automatisaatio, DevOps, Docker

Sisällys

1 Johdanto

Ohjelmistokehityksessä projektien edetessä tuottajan koodikanta kasvaa jatkuvasti ja koodin hallinnoinnin merkitys nousee merkittävämmäksi. Kehittyvät menetelmät auttoivat nopeuttamaan ohjelmointisykliä ja tutustuttivat tasaisen sekä luotettavan vauhdin ohjelmistojen tuottamiselle. CI (Continuous Integration) ja CD (Continuous Delivery) lisäsivät vielä tehokkuutta tuotantoon erilaisten järjestelmien ja automaation avulla. Automaattisten testien sekä useasti tehtävien koodin työntämisen avulla pystytään tehokkaaseen ja nopeaan työskentelyyn, jossa CI-järjestelmät hoitavat vikojen ja yhteensopivuusongelmien havaitsemisen. (Altexsoft 2022.)

Opinnäytetyön toimeksiantajan Frends:n projektissa käytetään tällä hetkellä GitHub Actions jatkuvan integraation järjestelmää. Järjestelmän käyttöä ja toimintamenetelmiä ei olla dokumentoitu eikä standardisoitu. Tämä luo haavoittuvuuksia järjestelmän käyttöön sekä esteitä tehokkuuden maksimointiin. Näiden ongelmien vuoksi myös uusien työntekijöiden perehdyttäminen projektin CI-järjestelmän käyttöön on hitaampaa.

Opinnäytetyö käsittelee CI:n eli Jatkuvan Integraation toimintamallien käyttöä ohjelmistotuotannossa sekä vertaa kolmea erilaista CI-järjestelmää keskenään. Työ tehdään toiminnallisena, jonka tietoperustan tavoitteena on syventää opiskelijan tietotaitoa CI:n maailmaan ja toiminnallisen osuuden tavoitteena on toteuttaa toimeksiantajalle eli Frends:lle kehitysehdotus CI-järjestelmän käyttöön sekä dokumentoida nykyiset toimintamallit ja käyttötapaukset. Toiminnallisen osuuden kehitysehdotuksessa peilataan CI:n yleisiä parhaita käyttö- ja toimintamalleja sekä verrataan nykyistä alustaa kolmeen eri järjestelmiin ja otetaan huomioon, miten niillä pystytään menettelemään yleisiä parhaita käytänteitä.

1.1 Tavoitteet

Opinnäytetyön tavoitteena on selvittää epäkohdat CI-järjestelmän käytössä toimeksiantajalla ja luoda dokumentaatio sekä kehitysehdotus. Kehitysehdotuksen lähtökohtana on parantaa nykyisen järjestelmän käyttöä tai suositella uuden järjestelmän käyttöönottoa.

Opinnäytetyön tekijän oppimistavoitteena on syventää tietotaitoa sekä käytännön osaamista CI-järjestelmiin ja niiden toimintaan. Erityisesti työnkulkukaavioiden luomiseen sekä niiden käyttöperiaatteisiin halutaan syventyä opinnäytetyön aikana. Lisäksi tavoitteena on tutustuttaa opiskelija erilaisiin CI-järjestelmiin ja niiden käyttötarkoituksiin sekä eroavaisuuksiin nykyisestä Frends:llä käytettävään järjestelmään.

1.2 Rajaukset

Työssä luodaan dokumentaatio CI:n käytön nykytilanteesta ja se rajataan koskemaan ainoastaan käyttötapauksia ja -tarkoituksia liittyen työn toimeksiantajan Frends:n virallisien taskien kehitysprojektiin. Taskit ovat koodipalikoita, jotka ovat käyttövalmiita palasia. Taskit ovat valmiiksi koodattuja virallisien taskien kehitysryhmän toimesta. Työn toiminnallisessa osiossa tarkastellaan kolmea erilaista CI-järjestelmää, joista yksi on tällä hetkellä käytössä oleva järjestelmä. Työssä käsiteltävät CI-järjestelmät ovat Github Actions, Jenkins ja TeamCity. Työssä tullaan myös keskittymään ohjelmoinnin osalta .NET ohjelmointialustaan sekä C# ohjelmointikieleen. Toiminnallisen osuuden dokumentointi osiossa keskitytään nykyiseen toimintamalliin. Työssä versionhallintaa koskevissa kohdissa peilataan tällä hetkellä Frends:llä käytössä olevaan GitHub. Muita versionhallintajärjestelmiä ei sisällytetä työhön. Vaikkakin työssä sivutaan DevOps:n muita kehityskanavia, kuten CD ja CT (Continuous Testing), työssä tullaan keskittymään CI;n käyttöön ja sen ominaisuuksiin.

1.3 Toimeksiantajan kuvaus

Frends eli Front End Dialing System kehitettiin vuonna 1988 palvelemaan Suomessa toimivia öljynjalostamoita, kuten Neste, Shell ja Teboil. 30 vuoden jälkeen nämä yritykset kuuluvat edelleen Frends:n asiakaskuntaan. Frends nimi syntyi sen käyttötarkoituksesta yhdistää Suomen bensa-asemilla käytetty data käyttäen dial-up modeemeja. Nykyään Frends on kansainvälisen vuonna 1995 perustetun HiQ:n omistuksessa. Tällä hetkellä Frendsin kehitystiimi koostuu kymmenestä pääkehittäjästä, joita tukevat yli 100 henkilön konsultointi ja tuki ryhmät. (Frends 2022.)

1.4 Käytetyt käsitteet

Osuvampi kuvaavampi

.NET — Microsoft:n kehittämä avoimeen lähdekoodiin perustuva ohjelmointi alusta, jolla pystytään rakentamaan erilaisia ohjelmia C# kielellä. (Microsoft, 2022.)

C# — Moderni olio-ohjelmointiin perustuva ohjelmointikieli, joka mahdollistaa kehittäjiä luomaan monen tyyppisiä suojattuja ja vakaita ohjelmia, joita ajetaan .NET alustalla.

Jatkuva integraatio (CI) — Jatkuva integraatio on työn pääkäsite. Käsitteellä tarkoitetaan DevOps toimintamallia, jonka päätarkoitus on

	tehostaa ohjelmistokehitysprosessia automatisoimalla prosessin eri osa-alueita.
Jatkuva toimitus (CD)	DevOps-mallin kehityskanava, jonka päätavoitteena on automatisoida tuotoksen julkaisua pääkäyttäjille.
Jatkuva testaus (CT)	CI:n vaihe, joka tarkoituksena on automatisoida testausta ja siten vähentää uusien bugien muodostuminen ja varmistaa koodien yhteensopivuus tietovaraston kanssa.
DevOps	Yhdistelmä toimintamalleja, käytänteitä ja työkaluja, joiden avulla pysytään tehostamaan organisaation kykyä toimittaa applikaatioita ja palveluja.
Docker	Ohjelmointia helpottamaan luotu alusta, jonka avulla pystytetään virtuaalisia ja eristettyjä ympäristöjä esimerkiksi testausta varten.
Frends	IPaaS integraatioalusta, jonka tarkoituksena on nopeuttaa ja helpottaa erilaisten järjestelmien integroitumista toisiinsa.
iPaaS	Alusta, joka yhdistää erilliset järjestelmät yhtenäiseksi ratkaisuksi asiakkaalle. (Frends, 2022.)
Test Driven Development (TDD)	Kehitysmalli, jonka tavoitteena on aloittaa kehitystyö luomalla kattava testikanta, jota käytetään hyödyksi perustelemalla ohjelman tavoitteiden saavuttaminen.
Yksikkötestaus (Unit Testing)	Yksikkötestauksella tarkoitetaan koodin tavoitteellista testaamista, jonka päätarkoituksena on varmistaa, että loppukäyttäjälle halutut toiminnot toimivat oikealla tavalla.
Kehityskanava (Pipeline)	Yhdistelmä automatisoituja prosesseja ja työkaluja, joiden tarkoitus on mahdollistaa kehittäjien yhteistyö tuotantoympäristössä.

Versionhallinta	Järjestelmä, joka seuraa ohjelmistoprojektin tiedostojen versioita. Versionhallinnan avulla pystytään tekemään muutoksia ilman, että aikaisempi versio häviää.
Ajaja (Runner)	GitHub Actions:n palvelin. Jokainen ajaja voidaan ylläpitää lokaalisti tai antaa GitHub:n hoitaa. GitHub ylläpitämät palvelimet voivat olla Ubuntu Linux, Windows tai macOS käyttöjärjestelmiin perustuvia. Ajaja ajaa työkulkukaavioissa määriteltyjä töitä ja tehtäviä.
Palautus (Roll back)	Toiminto, jossa projektin aikaisempi versio palautetaan työversioksi.
Työnkulku (Workflow)	CI-järjestelmissä käytetty työnkulkua kuvaava tiedosto, joka sisältää komennot CI:lle.
Tapahtuma (Event)	CI:n osa, jonka tarkoituksena on käynnistää työnkulku.
Tehtävät (Actions)	Komentoja, joita ajaja suorittaa työnkulun määrittelemällä tavalla.
Työt (Jobs)	Kokoelma työvaiheita, jotka suoritetaan saman ajajan päällä.
Työvaiheet (Steps)	Yksilöllisiä tehtäviä, joita CI-järjestelmän työt toteuttaa. Työvaiheet voivat olla joko tehtäviä tai shell komentoja.

1.5 CI sanastoa

Opinnäytetyössä käsitellään versionhallintaa ja versionhallinnan käsitteitä peilataan GitHub:n sanastoon. GitHub sanaston suomennoksissa tukeudutaan Ohjelmistokehityksen menetelmät sivuilta löytyvään Git sanastoon. Kuten sivuillakin lukee, sanastolle ei ole virallisia suomenkielisiä käännöksiä, mutta yhtenäisyyden vuoksi tuota sanastoa käytetään tässä opinnäytetyössä. Taulukosta 1 nähdään työssä käytetty sanasto. Taulukko on luotu mukaillen Ohjelmistokehityksen menetelmät internetsivuilta löytyvään GitHub sanastoon.

Taulukko 1. Sanasto Git (mukaillen Ohjelmistokehityksen menetelmät)

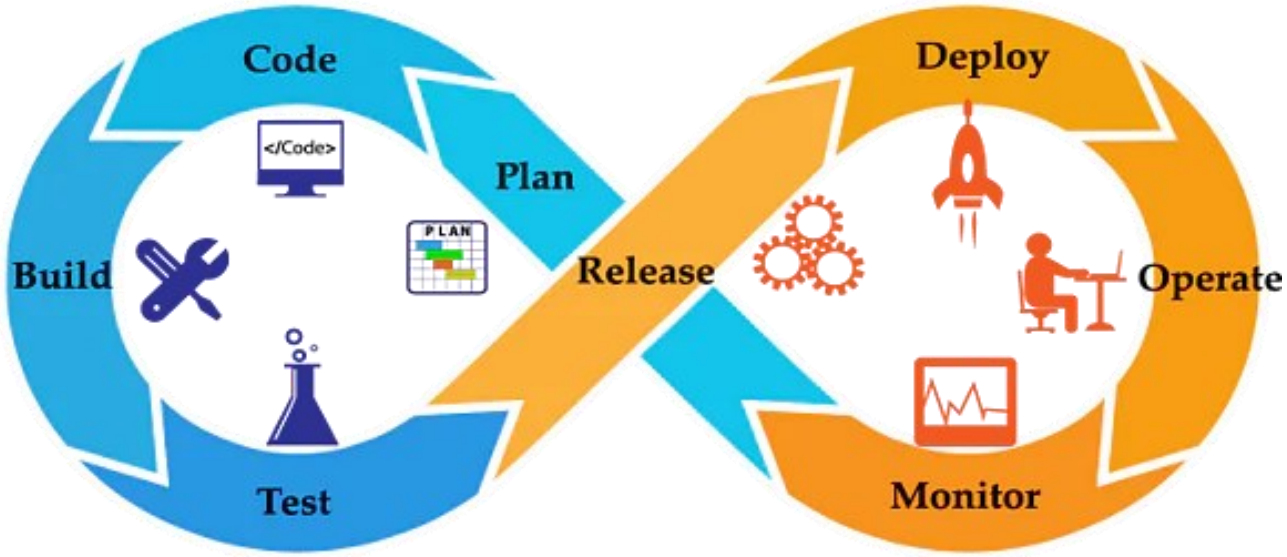
EN	FI	Selitys
Repository	Tietovarasto	Tarkoitetaan lähdekoodille luotua varastoa. Voidaan puhua myös eräänlaisesta "git hakemistosta". Sisältää versionhallinnan luoman muutoshistorian lähdekoodista.
Remote repository	Etätietovarasto	Tarkoitetaan sellaista olemassa olevaa tietovarasto, joka ei sijaitse kehittäjän työasemalla. Sijaitsee usein erillisellä palvelimella tai palvelussa. (Esim. Github)
Local repository	Paikallinen tietovarasto	Tarkoitetaan sellaista olemassa olevaa tietovarastoa, joka sijaitsee kehittäjän työasemalla.
Working directory	Työhakemisto	Tarkoitetaan sitä kansiota missä sillä hetkellä on komentorivillä. Esimerkiksi polku nykyiseen työhakemistoon Linux ympäristöissä selviää komennolla `pwd`
Branch	haara, kehityshaara	Tarkoitetaan versionhallinnassa päähaarasta tehtyä sivuhaaraa. Haara voi olla esimerkiksi toisen ominaisuuden takia tehty kopio nykyisestä versiosta, jota kehitetään itsenäisenä kokonaisuutena. Haara voidaan yhdistää takaisin päähaaraan.
Conflict	Konflikti	Tarkoittaa sellaista tilannetta, kun työkalu ei osaa yhdistää haluttuja muutoksia esimerkiksi kahden eri kehityshaaran kohdalla. Tällöin syntyy konflikti, jonka kehittäjän on selvitettävä itse ja muokattava alkuperäinen tiedosto haluttuun lopputilaan, joka vahvistetaan.
Master / main	Päähaara	Termi tarkoittaa tietovaraston päähaaraa, joka oletuksena syntyy aina kun Git -versionhallinta otetaan käyttöön. Usein se mitä pääosan ajasta käytetään mutta riippuu työtavoista. *(Katso myös kehityshaara)*
Commit	Pysyvä muutos	Tarkoitetaan muutosten vahvistamista, jotta ne tulevat voimaan ja näkyvät versiohistoriassa omalla

		tunnisteella. Muutokset on ensin pitänyt lisätä valmistelualueelle `add` komennolla.
Pull	Veto	Git komento, jolla etätietovarastossa olevat tiedot päivitetään paikalliseen tietovarastoon.
Pull Request (PR)	Vetoehdotus	Aktiviteetti, jossa ehdotetaan oman työversion yhdistämistä yhteiseen koodikantaan.
Push	Työntö	Pull komennon vastakohta eli työntää paikallisessa tietovarastossa olevat uudet pysyvät muutokset etätietovarastoon.

2 Jatkuva Integraatio

CI eli Jatkuva integraatio on DevOps malli, jonka ajatuksena on ajaa kehitystiimiä tuottamaan jatkuvasti pieniä muutoksia pohjakoodiin automaattisen versionhallinnan avulla. CI mahdollistaa automaattisen tavan rakentaa ohjelmistopaketteja ja testata muutoksien yhteensopivuutta sekä toimivuutta. Johdonmukainen integraatio prosessi rohkaisee kehittäjiä tekemään pysyviä muutoksia useammin, jonka takia parannetaan yhteistyön toimivuutta ja koodin laatua. (Sacolick 2022.)

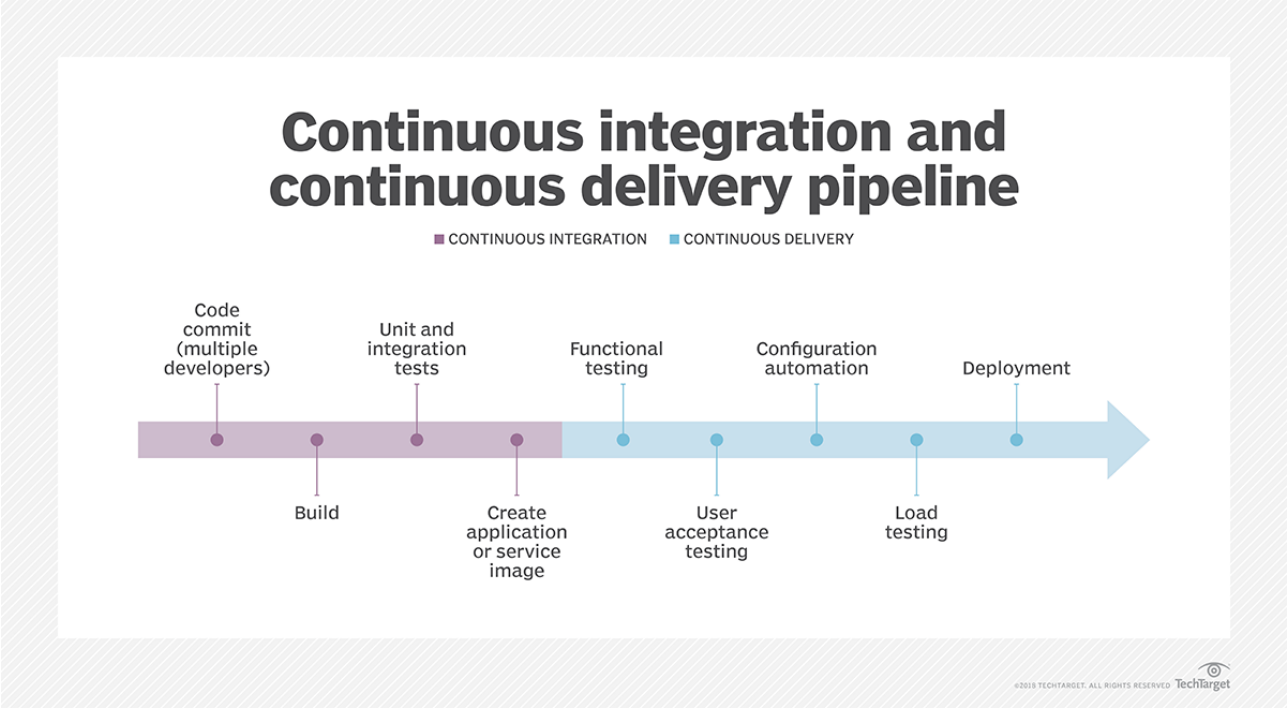
CI on kehitetty seuraamalla ketterien menetelmien ja DevOps:n parhaita käytäntöjä, jossa koodin testaaminen, versionhallinta, rakennus ja julkaisu on automatisoitu. Rehkopf kuvaa näitä CI:n pilareiksi, joiden tavoitteena on luoda automatisoitu ympäristö ohjelmistokehitykselle. (Rehkopf 2022.) Kuvasta 1 voidaan havaita DevOps:n toimintamalli, jossa sinisellä värjätty alue lukeutuu CI:n toimintaan ja keltaisella värjätty alue lukeutuu CD:n toimintaan. Kuvassa lisäksi DevOps:iin kuuluvat operointi ja monitorointi vaiheet.



Kuva 1. DevOps toimintamalli, Gaba 2022.

CI:n tavoitteena on minimoida työn määrää ja koodin yhdistämisen kompleksisuutta. Tämä saavutetaan toteuttamalla pieniä muutoksia pohjakoodiin ja puskemalla nämä muutokset yhteiseen koodikantaan nopealla aikavälillä. Kun muutoksien vaikutukset on saatu yhdistettyä ja testattua, jokaisen ryhmäjäsenen oma koodi saadaan päivitettyä ajan tasalle usein tapahtuvalla koodinkannan hakemisella. Näin saadaan luotua ympäristö, jossa jokainen työryhmän jäsen työskentelee ajantasaisella koodikannalla ja muutoksien tekeminen ja pohjakoodin kehittäminen helpottuu. (Roddewig 2022.)

Kun CI-järjestelmä saadaan toimimaan optimaalisella tavalla, pystytään helpommin löytämään koodin rikkovat bugit sekä muutokset, jotka eivät toimi koodikannan kanssa. Tämä saadaan mahdollistettua automaattisella testauksella, kun koodia yritetään työntää koodikantaan tai kun tarkistettu koodi lopulta yhdistetään. CI-järjestelmä myös ilmoittaa kehittäjille saman tien, mikäli heidän työssään on havaittu bugi tai sopimaton muutos. CI:n käytössä tavoite tilanne on saada muutoksia nopeasti ja tehokkaasti tehtyä. CI:n avulla tavoitellaan sitä, että kehittäjät tekevät muutoksia useasti yleensä vähintään kerran päivässä. Koodin automaattinen validointi mahdollistaa tämän nopean työrytmin eikä kehittäjien tarvitse odottaa koodausvaiheen loppuun asti ennen kuin työversio yhdistetään koodikantaan. (Roddewig 2022.) Kuva 2 havainnollistaa CI ja CD kehityskanaville suunnatut työnvaiheet. Kuva käsittää myös CT:n vaiheita, jotka on integroitu molempiin ylempiin malleihin.



Kuva 2. CI/CD malli kehityskanavasta. (Terrell 2022.)

CI toimii kehityskanavien avulla, joihin on annettu lista komentoja, joita CI:n halutaan toteuttavan. CI validoi tulevat muutokset automatisoitujen testien avulla ja ajaa tuotteeseen luodut validointi- sekä integrointitestit. Mikäli virheitä havaitaan CI keskeyttää ajon ja ilmoittaa kehittäjälle virheestä. CI siis paikallistaa virheen ja ilmoittaa siitä kehittäjälle. Tämä mahdollistaa virheiden havaitsemisen nopeammin ja poistaa integroitumispuutteiden syntymistä. Näin ollen tuote saadaan nopeammin valmiiksi ja julkaistua asiakkaille. (Ruck 2021.)

2.1 Versionhallinta

Versionhallinnalla tarkoitetaan projektin eri työvaiheiden tallentamista ja käsittelyä. Jotta tämä on mahdollista pitää projektin tiedostoihin tehtyjä muutoksia seurata. Versionhallinnan avulla pystytään tallentamaa jokainen tiedostoon tai useisiin tiedostoihin tehdyt muutokset. Versionhallinta luo version jokaisesta muutoksesta, joka tietovarastoon tehdään ja mahdollistaa palautuksen tekemisen, mikäli se on tarpeellista. Versionhallintaa pystytään hallinnoimaan siihen suunnatun työnkalun eli versionhallintajärjestelmän avulla. Versionhallintajärjestelmät mahdollistavat versioiden tarkastelun ja navigaation versioiden välillä. Näiden järjestelmien avulla pystytään nopeasti navigoitumaan projektin eri versioon. (Tsitoara 2020 Luku 1)

Versionhallinta on yksi CI/CD järjestelmän päätukipilareista. Versionhallinnan tarkoituksena on tukea projektin toimintaa ja helpottaa eri osien integroituminen toisiinsa. Max Rehkopfin mukaan suosituimmat versionhallintajärjestelmät ovat tällä hetkellä Git, Subversion ja Perforce. Näiden työkalujen avulla pystytään helpommin tekemään muutoksia yhteiseen tietovarastoon käyttämällä erilaisia komentoja. Versionhallintajärjestelmät mahdollistavat myös koodin eri versioiden hallinnointia. (Rehkopf 2022.)

Kehitysvaiheessa kehittäjät tekevät muutoksen yhteiseen tietovarastoon. Olkoon muutos uusi toiminto, päivitys tai korjaus, nämä kaikki seuraavat samaa mallia. Kehittäjä puskee muutokset yhteiseen koodikantaan, joka sijaitsee organisaation yhteisessä tietovarastossa. GitHub ja Atlanssian Bitbucket ovat yleisimmin organisaatioiden käytössä. Versionhallinnan työkalut mahdollistavat kehitystyön tekemisen ilman häiriöitä muiden kehittäjien työhön. (Levan 2020.)

2.2 Jatkuva testaus

CT tai Jatkuva testaus on jatkuvan kehityksen vaihe, jonka tavoitteena on automatisoida testausta ja toimia kehityksen tukena nopeuttamalla kehitystä ja päivityksien integroitumista koodikantaan. CT:n alussa tehdään staattinen koodin tarkistus, jonka aikana tarkistetaan syntaksi virheitä tai yleisiä haavoittuvuuksia. Liitännäisien (Plugins) avulla pystytään tekemään staattinen analyysi käyttäen hyväksi erilaisia työkaluja kuten SonarQube, Veracode tai Codacy. (Levan 2020.)

CT:n ensimmäisen vaiheen jälkeen tulee yksikkötestivaihe. Yksikkötestien avulla pystytään varmistamaan yksittäisten toiminnallisuuksien toiminta ja että nämä toimivat tarkoituksenmukaisesti. CT:n viimeinen vaiheen tavoitteena on testata koko koodin toiminnallisuus. Tämä tehdään rakentamalla ohjelma CI palvelimen avulla. Toiminnallisuustestien avulla varmistetaan, että koko ohjelma toimii tarkoituksenmukaisesti ja se toimii vahtina, jottei tuotantoon julkaista sellaista versiota, joka rikkoo toteutuksen. (Levan 2020.)

CI ei itse korjaa koodin virheitä, mutta sen avulla pystytään helpommin havaitsemaan ja korjaamaan niitä. Automaattiset virheidentarkistus ajot mahdollistavat virheiden havaitsemisen nopealla aikavälillä. (Gaba 2022.)

2.3 Rakennus

Rakennusvaiheen tavoitteena on yhdistää kehittäjien omat valmiit versiot yhteiseen koodikantaan ja varmistaa kehittäjien koodien yhteensopivuus ja tuotteen laadun korkeatasoisuus. Automatisoidulla kehityskanavalla pystytään paremmin varmistamaan näiden tavoitteiden saavutettavuus ja nopeuttaa työskentelyä projektin sisällä. Kehityskanavan integroiminen CI kehityskanavaksi parantaa ja automatisoi kehittäjien työtä jakaa koodia keskenään sekä tarjoaa työkaluja, jotta pystytään varmistamaan tuloksen standardisoitu laatu. (Fosco 2022.)

Rakennusvaiheessa ajatusmalli ”Fail fast” eli epäonnistu nopeasti toimii hyvänä lähtökohtana kehityksessä. CI:n tulee antaa nopeasti palautetta epäonnistuneista rakennuksista, jotta kehittäjät pystyvät nopeammin korjaamaan virheet. Kun palaute tulee nopeasti, on koodi vielä kehittäjän mielessä ja korjaus on useimmiten nopeampaa ja helpompaa. Ajatusmalli myös vähentää kehittäjien kontekstin vaihtoa, jolla saadaan parannettua DevOps toimintaa. (Silverthorne 2022.)

2.4 Julkaisu

Julkaisu on CI:n vaihe, jonka tavoitteena on siirtää projektin julkaisuversio tuotantoon tai muihin ympäristöihin. CI:n kehityskanavaa voidaan konfiguroida ja automatisoida julkaisemaan tuotantoversiota aikataulullisesti joko kaikille asiakkaille samanaikaisesti tai vain tietylle ryhmälle. Kehityskanavan kautta pystytään myös tekemään version palautus, mikäli komplikaatioita havaitaan. Kehityskanavien konfiguroimismahdollisuus varmistaa sen, että pystytään päättämään millä tavalla tuote päätyy asiakkaalle. (Fosco 2022.)

Automatisoimalla kehityksen julkaisuvaihe, pystytään saavuttamaan julkaisu ajankohdat nopeammin. Koska julkaisu tehdään CI:n toimesta, pystytään käyttämään kehityksessä olevat resurssit muualle. Nopean kehitystahdin vuoksi pystytään havaitsemaan virheitä nopeammin ja pitämään tuote julkaisu valmiissa tilassa jatkuvasti. Tämä ominaisuus varmistaa sen, ettei projekti jää jälkeen aikataulusta ja asiakkaat sekä käyttäjät saavat uusia toiminnallisuuksia tai parannuksia nopeammin. (InvoiceNinja 2022.)

2.5 Jatkuvan kehityksen tärkeys

Jatkuva kehitys parantaa organisaation kykyä kehittää ohjelmia ja tehdä muutoksia, uusia toiminnallisuuksia ja korjata olemassa olevaa nopealla aikataululla. CI mahdollistaa useiden kehittäjien

helpon yhteistyön ja lisää projektille läpi- ja pitkänäköisyyttä. CI ei ainoastaan paranna kehittäjien työympäristöä parantamalla työkaluja ja yhteistyötä, mutta se lisää tehokkuutta myös organisaatiotasolla. (Gaba 2022.)

Automaattisen testauksen ja nopean julkaisun avulla pystytään taklaamaan koodissa tapahtuvia virheitä ja alentaa projektin riskitasoa. Kun julkaisuja tehdään myös pienien koodinmuutoksien ja päivityksien aikana, löytyvät bugit ja virheet myös helpommin ja niiden korjaaminen pystytään tekemään heti. Tämä mahdollistaa nopeamman aikataulun ja projektista tulee kokonaisuudessaan halvempi. Nopea työvauhti mahdollistaa nopeamman palautteen saannin, joka helpottaa korjaustyötä ja tekee yhteistyöstä ja kommunikaatiosta tehokkaampaa. (Gaba 2022.)

Kun CI-järjestelmä on optimoitu toimimaan CD järjestelmän kanssa käyttäen samoja työnkulku työkaluja, helpottuu koodin jakaminen ja standardisointi. Tämä mukauttaa projektin läpinäkyvämmäksi ja parantaa ryhmäjäsenien yhteistyötä. Pitkällä aikavälillä tämä tehostaa kommunikaatiota mahdollistaen sen, että kaikki organisaatiossa on samalla tasolla. (Gaba 2022.)

Pienien muutoksien tekeminen ja optimoitu testaus kanava takaavat paremman laadun ja tekevät virheiden havaitsemisesta tehokasta. Mikäli muutos ei saavuta standardisoitua tasoa tai virheitä löytyy koodista, ilmoitetaan siitä suoraan tekijöille esimerkiksi sähköpostin tai SMS viestien avulla. Jatkuva palautteen anto myös auttaa kehittäjiä kehittymään ohjelmoinnissa. (Gaba 2022.)

Yhteistyön parantuminen ja työn tehostuminen edesauttaa projektia vähentämään odotusaikaa muiden projektin osuuksien osalta. CI mahdollistaa projektin prosessien jatkuvuuden tapahtui mitä tahansa. (Gaba 2022.)

2.6 CI:n yleisiä toimintamalleja

CI-järjestelmien päätavoitteena on nopeuttaa ohjelmistotuotantoa tiheästi tapahtuvien päivityksien avulla. CI-järjestelmien avulla pystytään tuottamaan tuloksia tehokkaammin ja julkaista niitä käyttäjille nopeammalla tahdilla. CI/CD kehityskanavien tuottamat prosessit muuttavat kehityksen kulkua ja tekevät muutoksien tuottamisesta systemaattista. Näiden kehityskanavien avulla viikkoja tai kuukausia kestävät muutokset pystytään tekemään päivissä tai jopa tunneissa. (Kaftzan 2022.) Tähän kappaleeseen on kerätty useimmiten toistuvat toimintamallit, jota monet kehittäjät ovat suositelleet käytettäviksi.

2.6.1 Optimoitu kehitysnopeus

GitLab (2022) esittelee yhden useimmin esiintyvän toimintamallin ”Commit early, commit often”, joka tarkoittaa sitä, että muutoksia tulisi tehdä usein siihen tietovarastoon, jossa CI toimii. Pienien

muutoksien ja usein tapahtuvan koodin puskemisen avulla pystytään käynnistämään CI:n tarkastukset usein ja saadaan nopeasti palautetta, mikäli virheitä havaitaan. Tämä nopeuttaa näiden virheiden korjaamisen ja estää virheiden kehittymisen koodin mukana. (GitLab 2022.)

Yleisesti CI:n tarkoitus on nopeuttaa kehitystä, mutta CI myös itse tarvitsee nopean kehitysnopeuden toimiakseen tarkoituksenmukaisesti. Nopea kehityskanava varmistaa nopeamman palautteen projektin kehitysprosessista. Tämä helpottaa kehittäjien työtä ja mahdollistaa muutoksien työntämistä tietovarastoon nopeammalla aikataululla. Tämä skenaario mahdollistaa nopeamman virheiden havaitsemisen ja korjaamisen. (Gaba 2022.)

Optimoidulla kehitysnopeudella pystytään säästämään kehittäjien aikaa, kun CI-järjestelmä hoitaa osan kehitysprojektista automaatioiden avulla. Kehittäjien aikaa pystytään säästämään vielä enemmän käyttämällä hyödyksi samankaltaisia konfiguraatioita. Useat CI-järjestelmät mahdollistavat samojen konfiguraatiotiedostojen käytön useissa eri projekteissa. (Ben 2022.) Ziolkowski (2022) myös CI kehityskanavan tehokkuuden tärkeyteen. Mikäli CI:n kehitykanavat ovat hitaita, nousee riski, että kehittäjät alkavat yrittää kiertää sitä aina kun mahdollista. Mitä useampi kehittäjä yrittää kiertää kehityskanavaa, sitä suuremmaksi riski kasvaa, että virheitä päätyy tuotantokoodiin. (Ziolkowski 2022.)

2.6.2 Tietoturvallinen CI/CD järjestelmä

CI-järjestelmän käytössä on useita tietoturvariskejä, jotka pitää ottaa huomioon, kun suunnitellaan kehityskanavia. Kun ohjelmisto saavuttaa tuotantovaiheen, se on useimmiten käynyt lukemattoman määrän kehityssyklejä läpi, joissa ohjelmistoa on kehitetty, testattu ja hienosäädetty. Tuotantovaiheessa koodi on suhteellisen vakaata. Tässä vaiheessa nousevat tietoturvariskit yleensä johtuvat väärin konfiguroiduista komponenteista tai ulkoisista riskeistä. CI/CD Security on ideologia, jonka avulla pystytään varmistamaan CI/CD kehityskanavien tietoturvallisuutta. (Kaftzan 2022.)

Myrkytetty kehityskanava (PPE) on haavoittuvuus, joissa kehityskanavan suojaamattomuutta voidaan käyttää hyväksi syöttämällä haitallista koodia. Haitallisen koodin tarkoituksena voi olla esimerkiksi aiheuttaa CI-prosessin kaatuminen tai kaapata koko prosessi. Tältä tietoturvariskiltä pysytään suojautumaan välttämällä ei luotettuja syötteitä (input) sekä ulostuloja (output). (Kaftzan 2022.)

Projektissa käytettävät kolmannen osapuolen tuottamat palvelut ja ohjelmat voivat luoda tietoturvariskejä, mikäli niiden hallinnointia ei ole tehty organisoidusti. Esimerkkinä kehittäjä voi käyttää kolmannen osapuolen kehittämää koodikirjastoa, jossa ei ole samoja turvallisuuskäytänteitä käytössä kuin sisäisessä projektissa. Näin ollen koodi kirjaston sisällä voi olla vähemmän turvallista ja se vois sisältää haavoittuvuuksia. Hallitun hallinnan avulla pysytään taklaamaan tällaisia

tietoturvariskejä. Organisaation sisälle rakennettujen käytänteiden myötä voidaan esimerkiksi käyttää ainoastaan sellaisia kolmannen osapuolen tuottamia kirjastoja, jotka täyttävät organisaation sisäisesti sovitut tietoturvakäytänteet. Lisäksi näiden kirjastojen päivittäminen tulisi tehdä aikataulutetusti ja useasti. (Kaftzan 2022.)

Riippuvuuksien ja muiden ohjelmiston komponenttien hallinnointi sekä seuraus on tärkeää, kun taistellaan tietoturvariskejä vastaan. Esimerkiksi web applikaatiolla voi olla käytössä tietokanta, jota kyberhyökkääjät voivat käyttää hyväksi. Tietokannan avulla voidaan web applikaatioon syöttää vahingollista koodia, jonka avulla pystytään ottamaan yhteys itse applikaatioon. Tämänlaisiin riskeihin pysytään varautumaan käytänteillä ja prosesseilla, joiden tarkoituksena on varmistaa riippuvuuksien tietoturvallisuus ja monitoroida mahdollisia haavoittuvuuksia. (Kaftzan 2022.)

2.6.3 Testilähtöinenkehitys

Beck:n (2002, Luku Preface) mukaan testilähtöisenkehitys koostuu kahdesta säännöstä: kirjoitetaan uutta koodia ainoastaan, jos projektiin luodut testit epäonnistuvat ja poistetaan toistuvuus koodista. Näiden sääntöjen mukaisesti testilähtöinenkehitys aloitetaan luomalla testit ensimmäiseksi. Tämän jälkeen kirjoitetaan koodi, mutta vain siltä osin, että testit saadaan menemään läpi. Lopuksi koodista poistetaan toistuvuus eli se refaktoroidaan. (Beck 2002, Luku Preface.)

Testilähtöisen kehityksen tavoitteena on luoda testikanta projektin alussa, jota käytetään hyödyksi kehityksessä. Projekti aloitetaan luomalla tarkka määritelmä sekä listaamalla toiminnallisuudet, jotka ohjelman tulee täyttää. Tämä malli varmistaa tuotteen yksinkertaisuuden sekä tarkoituksenmukaisuuden ja vähentää bugeja kehityksen aikana. Virheen löytäminen valmiista ohjelmasta on yleensä paljon haastavampaa. Lisäksi virhe, joka on luotu projektin alkuvaiheessa saattaa laajentua projektin edetessä, tehden sen korjaamisesta vaikeampaa ja aikaa vievää. (Philip 2022.)

Silverthorne:n (2002) mukaan testien kattavuuden sekä toiminnan tehokkuuden tasapainottamisessa on hiuksenhieno raja, milloin kannattaa mieluummin suosia tehokkuutta ja milloin testien kattavuutta. Esimerkiksi mikäli testaukseen menee todella paljon aikaa, voi kehittäjät yrittää kiertää testausprosessia. CI-järjestelmän tulee voida tukea kehitysprosessia, mutta sen pitää pystyä toimimaan tehokkaasti ja antaa palautetta nopeasti. Tähän pystytään vaikuttamaan konfiguroimalla ja optimoimalla kehityskanavia tarkoituksenmukaisiksi. (Silverthorne 2022.)

2.6.4 Muita toimintamalleja

Vetoehdotuksien avulla kehittäjän koodi yhdistetään yhteiseen koodikantaan. Vetoehdotus ilmoittaa kaikille projektissa oleville henkilöille, että uusi muutos on valmiina yhdistettäväksi. Vetoehdotuksen myötä useimmiten käynnistetään automatisoituja testejä ja hyväksymisaskeleita.

Manuaalinen hyväksyminen on myös yleensä vaadittu, jotta muutos voidaan yhdistää yhteiseen koodikantaan. Tässä joku muu projektin henkilö käy muutokset läpi ja tekee koodintarkistuksen (code review). Tämän tarkistuksen avulla pystytään takaamaan parempi lopputulos ja ohjelman toiminnallisuuksien toimivuus verrattuna sen tavoitteisiin. (Gaba 2022.)

CI kehityskanavat sisältävät töitä ja vaiheita, joita CI-järjestelmä toteuttaa. Työt toteutetaan tietyssä vaiheessa ja kun kaikki vaiheen työt on toteutettu, CI jatkaa seuraavaan vaiheeseen. Optimoimalla nämä vaiheet pystytään tehostamaan virheiden löytämistä, kun palaute saadaan nopeammin CI-järjestelmästä. Kehityskanavan vaiheet voidaan helposti organisoida sisältämään samankaltaisia töitä, mutta jotkut näistäkin töistä voidaan ajaa aikaisemmissa vaiheissa. Näiden töiden ajaminen aikaisemmissa vaiheissa nopeuttaa virheiden löytämistä, mikäli niitä havaitaan. (GitLab 2022). Tästä voidaan todeta, että vaikka jossain tapauksissa töiden organisoiminen samankaltaisien töiden kanssa tuntuisi luontevalta, voi useimmiten olla parempi vaiheistaa työt sen mukaan, koska ne on mahdollista ajaa.

3 CI-järjestelmät

Tässä kappaleessa käydään läpi kolme CI-järjestelmää ja verrataan niiden toimintaa keskenään. CI-järjestelmien vertailu toteutetaan ottamalla GitHub Actions järjestelmä pohjaksi ja kahta muuta järjestelmää verrataan siihen.

CI-järjestelmät koostuvat kehityskanavista, jotka ovat järjestelmän toiminnallisia työkaluja. CI työnkulut ovat orkestraatioita, jotka mahdollistavat ohjelman ajamisen. Työnkulut jakavat CI:n työt eri osiin, mikä helpottaa virheiden etsintään. CI:n töiden ajovaihetta voidaan seurata reaaliajassa ja siitä nähdään helposti ja nopeasti missä kohtaa virhe tapahtuu. Tämä nopeuttaa virheenkorjausta, kun tiedetään saman tien missä kohtaan työnkulku on päätynyt virheeseen. Työnkulun töitä pystytään useimmiten myös ajamaan erikseen, jolloin pystytään helposti varmistamaan, että korjaus on toiminut. Työt ovat kokoelma työnvaiheita, jotka CI työnkulku toteuttaa. Työnkulun avulla pystytään määrittelemään säännöt näille vaiheille ja töille sekä määritellä ajojärjestys. (Fosco 2022.)

3.1 Github Actions

Github Actions on CI-järjestelmä, joka toimii rinnakkain GitHub tietovaraston kanssa. Github Actions on natiivi CI/CD työkalu, joka toimii webalustalla ja sen käyttöönotossa, ei tarvita erillisiä asennuksia. Tämä tekee järjestelmästä äärimmäisen kevyen. GitHub Actions rakentuu ajajista, tapahtumista, töistä, työvaiheista ja tehtävistä. Näiden osasien toiminta kirjoitetaan työnkulkukaavioon. (GitHub Docs 2022). Kuvasta 3. nähdään millä tavalla työnkulku rakentuu. Kaaviosta voidaan havaita työ, jonka työnkulku toteuttaa. Työ on jaettu eri osiin, joissa eritellään käyttöjärjestelmä ja työn vaiheet. Tässä esimerkissä käyttöjärjestelmä on Ubuntu ja työnkulussa on vain yksi vaihe nimeltä Rick Roll. Tässä tapauksessa työ käyttää valmista ohjelmaa, jota työnkulku kutsuu työn käynnistyessä.

```yaml
on:
  issues:
    types: [opened]

jobs:
  comment:
    runs-on: ubuntu-latest
    steps:
     - name: Rick Roll
      uses: TejasvOnly/random-rickroll@v1.0
      with:
        percentage: 100
```

Kuva 3. Esimerkki työnkulku. (Scarlett 2022)

Github Actions järjestelmän käyttöönotto on helppoa. Aluksi projektin tietovaraston juureen luodaan /.github-niminen hakemisto, jonka alle luodaan /workflows niminen alihakemisto. Näiden hakemistojen tarkoitus on saada CI:n työnkulut näkyviksi GitHub Actions järjestelmälle. Tähän hakemistoon luodaan YAML muotoinen tiedosto, joka on itse työnkulku. (GitHub Docs 2022.) YAML on datan serialisointikieli, jota yleisesti käytetään konfiguraatiotiedostoissa (Red Hat 2021). Kyseinen hakemisto voi sisältää useita työnkulku tiedostoja, jotka voidaan toteuttaa projektin eri vaiheissa. Työnkulku sisältää komentoja ja ohjeita, joita CI-järjestelmän halutaan toteuttaa. Viimeiseksi muutokset tulee työntää tietovaraston päähaaraan. (GitHub Docs 2022.)

Tapahtumat ovat toimintoja GitHub Actions järjestelmässä, joiden tehtävänä on käynnistää työnkulun ajo. Tapahtumat voidaan konfiguroida alkamaan esimerkiksi, kun GitHub koodikantaan tehdään vetoehdotus, avataan kehitysehdotus (issue) tai työnnetään pysyviä muutoksia. Työnkulut voidaan asettaa alkamaan myös aikataulutetusti joko manuaalisesti tai käyttämällä webhook-tapahtumia. (GitHub Docs 2022.)

3.2 Jenkins

Jenkins kehitettiin vuonna 2004 automatisoimaan eri ohjelmistokehityksen kehitysvaiheita. Jenkins on epäitsenäinen palvelimessa suoritettava ohjelmisto (servlet), jonka avulla pysytään luomaan automatisoituja kehityskanavia kehityksen tueksi. Monien liitännäisten avulla Jenkins pystyy suoriutumaan monesta eri kehitysvaiheesta. (Dingare 2022. Luku 2)

Jenkins toimii skriptien avulla, joita Jenkins kutsuu kehityskanaviksi. Nämä ovat tehtävien ja alitehtävien ketjuja, joita käytetään jokaisessa rakennusvaiheessa. Nämä rakennusvaiheet on suunniteltu siten, että jokainen vaihe ottaa edellisen vaiheen tuotoksen ja käyttää sen määritellyn prosessin läpi ennen kuin antaa sen seuraavalle vaiheelle. Mikäli virheitä havaitaan missään rakennusvaiheessa, prosessi pysäytetään ja järjestelmä tulostaa virheviestin kehittäjän nähtäväksi. (Dingare 2022. Luku 2)

Jenkins:n teknilliset vaatimukset asennettavalle järjestelmälle on esitetty taulukossa 2. Kuten taulukosta 2 voidaan havaita, Jenkins on suhteellisen kevyt järjestelmä ja se vaatii ainoastaan 256 MB RAM muistia sekä 1 GB verran tallennustilaa. Näiden lisäksi Jenkins tarvitsee tietyn JDK version Java:sta. (Dingare 2022. Luku 3.)

Taulukko 2. Jenkins:n vaatimukset asennettavalle järjestelmälle.

Vaatimus	**Selitys**
Muisti (RAM)	256 MB
Tallennustila	1 GB (jos asennettuna Docker konttiin, suositellaan vähintään 10 GB)
Software	Java: JDK 8 tai JDK 11

Jenkins:n asennus tehdään suoraan koneelle. Koska järjestelmä on palvelimessa suoritettava ohjelma, tarvitaan tietokone, johon ohjelma asennetaan. Jenkins voidaan asentaa Windows-koneelle lataamalla ja ajamalla MSI asennuspaketti. MSI tiedosto on Windows-paketti, joka sisältää tarvitun informaation asennusta varten. Toinen vaihtoehto on ladata Jenkins:n .WAR (Web Application Resource) tiedosto ja tekemällä asennus sitä kautta. WAR tiedostoon säilötään applikaatioon tarpeelliset tiedot. Se voi sisältää JAR tiedostoja, JavaServer sivuja, Java Servlet:ä, Java luokkia, xml tiedostoja ja muita resursseja, joita verkossa toimiva applikaatio tarvitsee. Jenkins on myös mahdollista asentaa käyttäen hyödyksi Docker ohjelmaa. Kuten taulukosta 1 voidaan nähdä, mikäli halutaan asentaa Jenkins Docker konttiin suositellaan vähintään 10 GB verran tallennustilaa. (Dingare 2022. Luku 3.)

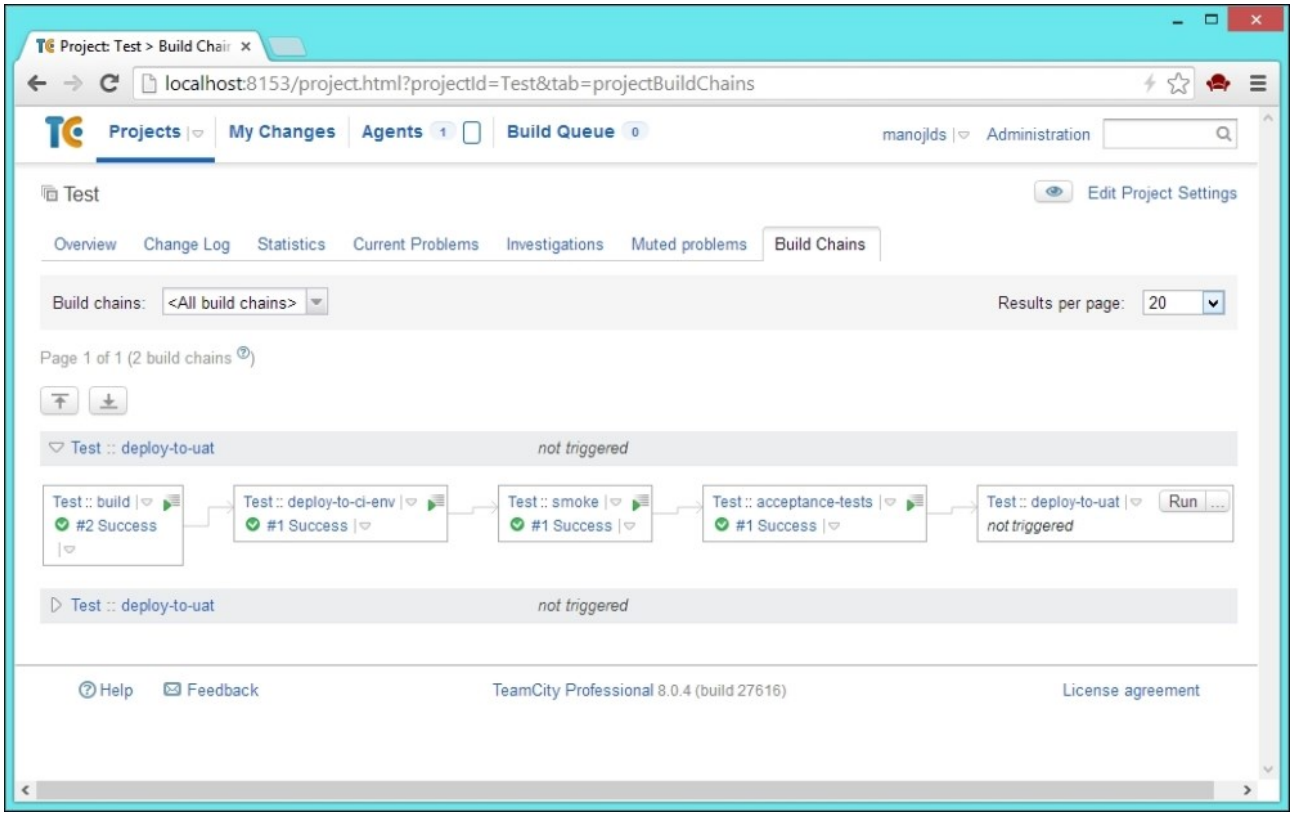
3.3 TeamCity

TeamCity on JetBrains:n vuonna 2006 kehittämä CI-järjestelmä, jonka toimintamalli perustuu ’out-of-the-box’ ajatusmalliin. Tällä tarkoitetaan sitä, että järjestelmä on valmiiksi konfiguroitu sisältämään yksikkötestailun, koodin laadun tarkistuksen ja aikaisen rakennus virheiden havaitsemisen. TeamCity:n käyttöönotto tapahtuu minuuteissa, koska asennusprosessista on luotu mahdollisimman yksinkertainen ja automaattinen. Aikaisempiin CI-järjestelmiin verrattuna TeamCity on helpompikäyttöinen ja se sisältää kaikki lisävarusteet eikä näin ollen tarvitse erillisiä asennuksia, kuten Jenkins. Näiden toiminnallisuuksien takia, TeamCity ei perustu, aikaisempien järjestelmien kaltaisesti, avoimeen lähdekoodiin. (Kumar 2022.)

TeamCity on maksullinen CI-järjestelmä, jossa on mahdollisuus ilmaiseen tasoon (tier) kunhan järjestelmää käyttävä projekti on tarpeeksi suppea. Koska TeamCity perustuu suljettuun lähdekoodiin, myös yhteisötuki on pienempää kuin aikaisemmin mainituilla järjestelmillä. (Kumar 2022.)

TeamCity:n käyttöönotto on nopeaa ja helppoa, koska erillisiä työnkulkukonfiguraatiotiedostoja ei tarvita. TeamCity on kuitenkin Jenkins:n tavoin ladattava aluksi lokaalille koneelle enne kuin järjestelmää voidaan käyttää. Työnkulkujen konfigurointi onnistuu suoraan järjestelmän käyttöliittymän avulla ja TeamCity tarjoaa loistavan dokumentaation käytön tueksi. (TeamCity.)

TeamCity:n toiminta keskittyy näyttävyyteen ja toimintojen seuraamiseen visuaalisesti. Mahalingam:n (2014) mukaan TeamCity tarjoaa hienon käyttöliittymän, jonka avulla projektien ja rakennuskonfiguraatioiden tarkkailu on helppoa. TeamCity tarjoaa myös käyttäjähallintaan liittyviä toimintoja, joiden avulla pystytään myös seuraamaan mitä kukakin projektin henkilöistä teki. Käyttäjät pystyvät kommunikoimaan mitä he ovat tekemässä ja muut käyttäjät voivat antaa eri tehtäviä muille käyttäjille. TeamCity tarjoaa kehityskanavien rakentamisen ja konfiguraation suoraan sen käyttöliittymästä. Kuva 4 havainnollistaa kehityskanavan visuaalisen ilmeen. (Mahalingam 2014 Luku 1.)



Kuva 4. Kehityskanavan visuaalinen toteutus TeamCity:llä. (Mahalingam 2014 Luku 1)

3.4 Vertailu

Kaikissa CI-järjestelmissä on omat hyvät ja huonot puolensa. CI-järjestelmien vertailu toteutetaan vertaamalla kahta muuta järjestelmää, Jenkins ja TeamCity, toimeksiantajalla nykyisin käytössä olevaan CI-järjestelmään, GitHub Actions:iin.

GitHub Actions järjestelmän käyttöönotto on helppoa eikä erillisiä asennuksia tarvita, koska GitHub Actions toimii pilvessä ja kaikki toiminnot pystytään näkemään webalustan avulla. GitHub Actions on kuitenkin mahdollista asentaa lokaalille koneelle. Tämä tarjoaa myös mahdollisuuden, mikäli yritys haluaa toimia täysin On-Premise toimintamallilla. On-Premise tarkoittaa sitä, että IT-infrastruktuuri ylläpidetään paikallisesti (SoftCo 2022). Jenkins tarvitsee erillisen asennuksen palvelimelle samoin kuin TeamCity. Tämä hidastuttaa käyttöönottoa, kun verrataan järjestelmään, jossa erillistä asennusta ei tarvita.

Molemmat GitHub Actions ja Jenkins käyttävät työnkulkukaavioita, jotka konfiguroidaan joko YAML tiedostojen tai scriptien avulla. TeamCity:n avulla pystytään käyttäjäystävällisemmin konfiguroimaan työnkulkuja suoraan käyttöliittymästä. TeamCity tarjoaa kehitysystävällisemmän tavan konfiguroida kehityskanavia, koska käyttäjien ei tarvitse opetella uutta kieltä tai scriptaamisen taitoa.

TeamCity:n käyttöliittymä on näistä kolmesta parhaimman näköinen ja se on helppokäyttöinen uusille kehittäjille. Jenkins:n käyttöliittymä on vanhoillinen eikä se ole kaunis. Käyttöliittymässä on myös paljon parannettavaa. GitHub Actions:n käyttöliittymä on lähempänä TeamCity:n käyttöliittymää ja sen avulla pystytään näkemään työkulkukaavioiden toteutuksen ja vaiheet.

TeamCity:n tietoturvallisuus on kahteen muuhun järjestelmään parempi. Tähän vaikuttaa se, että TeamCity on kaupallinen, kun taas GitHub Actions ja Jenkins perustuu avoimeen lähdekoodiin. Voidaan olettaa, että mikäli haavoittuvuuksia löytyy järjestelmästä, TeamCity tarjoaa nopeammin korjaukset näihin kuin avoimeen lähdekoodiin perustuvat kilpailevat järjestelmät. Taulukosta 3 voidaan havaita CI-järjestelmien vertailuntulos. Taulukkoon on kirjattu jokaisen järjestelmän hyvät ja huonot puolet tiettyyn kategoriaan verrattuna.

Taulukko 3. CI-järjestelmien vertailuntulos.

	GitHub Actions	Jenkins	TeamCity
Tietovarasto	+ Sisään rakennettu - Pakottaa GitHub tietovaraston käytön	- Joudutaan erikseen migroimaan + Voidaan käyttää mitä tahansa tietovarastoa	- Joudutaan erikseen migroimaan + Voidaan käyttää mitä tahansa tietovarastoa
Käyttöliittymä	+ Helppo kun opittu käyttämään	- Vanha käyttöliittymä	+ Helppo käyttöliittymä

	- Käyttöliittymä tarjoaa vain osa toiminnallisuuksista	- Toiminta maksuton, joten kehityksiä tuskin on tulossa UI:hin	+ Käyttöliittymä mahdollistaa lähes kaiken.
Asennus	+ Webalusta, ei tarvita erillistä asennusta	- Nopea asennus palvelimelle ja suhteellisen kevyt.	- Nopea asennus palvelimelle ja suhteellisen kevyt.
Kehityskanavien konfiguraatio	- Konfiguraatiotiedostojen kirjoittamiseen tarvitaan erillinen serialisointikieli YAML	- Konfiguraatiot tehdään scriptien avulla	+ Konfiguraatiot pystytään tekemään suoraan käyttöliittymästä
Yhteisön tuki	+ Open source, joten yhteisön tuki on suurta	+ Open source, joten yhteisön tuki on suurta	- Maksullinen ja yhteisön tuki on pienempää + Järjestelmällä on laajat dokumentaatiot järjestelmän toiminnasta
Maksullisuus	Ilmainen	Ilmainen	Maksullinen, mutta ilmainen versio saatavilla

4 Kehitysehdotus CI-järjestelmän toimintamalleihin

Opinnäytetyön toiminnallisessa vaiheessa toteutetaan toimeksiannolle nykytilakartoitus sekä kehitysehdotus CI:n käytöstä Frends:n virallisien Taskien kehityksessä. Taskit terminä tarkoittavat koodipalikoita, joita käytetään Frends alustalla ja ne on koodattu valmiiksi käyttäjien käytettäviksi. Taskit ovat valmiita palikoita, joita tarvitsee ainoastaan konfiguroida käyttäjän tarkoituksiin sopiviksi. Taskeille ei ole suomenkielistä vastinetta, joten työssä päädyttiin käyttämään tätä termiä, koska se on Frends:n puolella hyväksytty termi. Frends:n puolella Taskit kirjoitetaan isolla alkukirjaimella, jotta se pystytään eristämään yleisesti käytössä olevasta termistä ’task’ eli tehtävä.

4.1 Toimeksiannon kuvaus

Työn projekti toteutetaan toimeksiantona Frends:lle. Projektin tavoitteena on toteuttaa nykytilakartoitus Github Actions CI-järjestelmän käytöstä ja nykyisistä toimintamalleista. Lisäksi vertaillaan nykyisiä käyttömalleja tietoperustassa havaittuihin parhaisiin toimintamalleihin ja luodaan vertailun tuloksien pohjalta kehitysehdotus CI-järjestelmän käyttöön. Vertailussa otetaan myös huomioon, miten toimintaa voidaan parantaa nykyisen tai vaihtoehtoisen CI-järjestelmän avulla. Projektin tuloksena tulee valmis kehitysehdotus, jonka avulla nykyistä toimintaa voidaan parantaa. Projekti rajataan kattamaan Frends:n virallisien Taskien kehitykseen käytettävä CI-järjestelmä sekä nykyiset toimintamallit.

4.2 Nykytilakartoitus

Frends virallisien Taskien kehityksen CI:n käytön nykytilakartoitus esitellään liitteessä 2. Nykytilakartoitus. Liitteen kaksi mukaisesti projektin CI-järjestelmänä käytetään GitHub Actions järjestelmää. Projektin koodikirjastoon on luotu viisi työnkulkua, joita käytetään perustana jokaisen Taskin omassa työnkulussa. Luettavuuden parantamiseksi luettelen pohjatyökulkujen tiedoston nimen lisäksi keksityn nimen, jota käytetään työnkulkujen selityksissä. Projektin pohjatyönkulut ovat:

- build_test.yml – Windows-rakennustyönkulku
- linux_build_test.yml – Linux-rakennustyönkulku
- build_main.yml – Windows-yhdistämistyönkulku
- linux_build_main.yml – Linux-yhdistämistyönkulku
- release.yml - Julkaisutyönkulku

Nämä työnkulut on luotu automatisoimaan eri kehitysvaiheen testausta. Sekä Windows-, että Linux-rakennustyönkuluja käytetään testaamaan puskemisvaiheessa olevia haaroja ja yhdistämistyönkulkuja käytetään testaamaan yhdistämisvaiheessa olevia haaroja. Julkaisuvaiheen aikana testausta ei tehdä vaan julkaisutyönkulku on luotu automatisoimaan uuden Taskin tai uuden

version julkaisua Task-feed:iin. Task-feed:n avulla pystytään toimittamaan Taskit asiakkaille ja käyttäjille suoraan heidän Frends-alustaansa. Julkaisutyönkulku kuitenkin käynnistetään manuaalisesti, jolloin pystytään varmistamaan, että tuotantoon menee ainoastaan valmis tuote.

Ennen julkaisua käynnistyvien työnkulkujen toiminta on suhteellisen samankaltaista. Työnkulun alussa alustetaan ympäristömuuttujat sekä esikomento. Tämä jälkeen listataan työnkulun työt, jotka ajetaan työnkulun mukaan joko Linux tai Windows-alustalla .Net ohjelmointialustaa vasten. Tämän jälkeen Linux pohjaisissa työnkuluissa asennetaan tarvittu työkalu xmlstarlet 'apt-get' komennon avulla ja ajetaan työnkulun esikomento, jos sellainen on annettu. Seuraavaksi Taski rakennetaan, ajetaan testit ja luodaan NuGet paketti, jota pystytään käyttämään hyväksi julkaisussa. Lopuksi työnkulku laskee testien kattavuusprosentin ja tallentaa sen API:a hyväksi käyttäen Taskin GitHub koodikirjastoon. Lisäksi Taskin NuGet paketti pusketaan testi feediin.

Projektin työnkulut testaavat ainoastaan Taskiin luotuja yksikkötestejä. CI:n avulla ei pystytä testaamaan Taskien toimivuutta Frends alustalla, mutta tätä varten on luotu erillinen testialusta, jonka toiminta perustuu manuaalitestaukseen.

4.3 Kehityskohteet

Projektille on rakennettu pohjatyönkulut sekä Windows-, että Linux-pohjaisille testialustoille. Tämä tarkoittaa sitä, että Frends-Taskeja pystytään testaamaan sekä Linux-, että Windows-alustoilla. Taskien testaamista ei kuitenkaan ole standardisoitu vaan kehittäjä voi itse valita kummalla alustalla testit halutaan suorittaa. Testausalusta useimmiten valitaan sen perusteella, kummalla on helpompi Taskeja testata. Tämä luo riskejä, kun Taski ajetaan testaamattomalla alustalla ensimmäistä kertaa. Tosi asia on kuitenkin se, ettei perus-Taskien toiminta eroa paljon, vaikka se ajettaisiin molemmilla alustoilla, mutta standardisointi loisi lisäturvaa ja antaisi myös lisäarvoa projektin tuotoksille. Silverthornen (2022) mukaan testauksen toteuttamisessa tulee kuitenkin ottaa huomioon testauksen ja toiminnan tehokkuuden tasapaino (Silverthorne 2022). Tästä voidaan todeta, ettei testauksen toteuttaminen molemmilla alustoilla ole jokaisessa tilanteessa välttämätöntä. Esimerkiksi mikäli on jo tiedossa, että Taskin koodi tulee toimimaan molemmilla alustoilla täysin samalla toimintaperiaatteella.

Taskien julkaisu tehdään nykyisellä tasolla manuaalisesti ja ilman testaamista. Todellisuudessa Taskit on kyllä kehitys- ja yhdistämisvaiheessa testattu kahteen kertaan ja Taskin toimivuus vielä varmistetaan manuaalitestauksella Frends alustalla. Tähän pitäisi pystyä luomaan automaattinen julkaisuversion testaus, jonka aikana varmistetaan Taskin tai muutoksien toimivuus sekä yhteensopivuus. Lisäksi Taskit pitäisi automaattisesti testata Frends alustalla nopeuttamaan ja tehostamaan toimintaa. Lopuksi Taskin julkaiseminen, eli Taskin NuGet paketin lisääminen NuGet feediin tulisi

automatisoida, jotta minimoidaan riskiä, että kehittäjä unohtaa uuden version julkaisun. Kuten InvoiceNinja (2022) toi esille työn kappaleessa 2.4, julkaisun automatisaatiolla pystytään pitämään projektin tuote julkaisuvalmiissa tilassa ja siten nopeuttaa tuotteen siirtymistä asiakkaan käyttöön (InvoiceNinja 2022).

CI-järjestelmän testien aikana käytetään testien kattavuuslaskelmaa, joka esitetään jokaisen Taskin GitHub tietovarastossa Tähän ei kuitenkaan olla asetettu standardisoitua tasoa, jonka jokaisen Taskin tulisi saavuttaa. Taskien kehityksen aikana testien kattavuudelle tulisi asettaa standardisoitu prosentuaalinen taso, joka on sama jokaisella Taskilla. Byron Skoutaris:n mukaan testilähtöinen toimintamalli tukee CI:n toimintaa vähentämällä virheitä ja luomalla hyvän testauspohjan heti projektin alusta alkaen (Skoutaris 2019). Ottamalla testilähtöinen kehittäminen standardisoiduksi tavaksi luoda Taskeja, voisi parantaa kattavuuslaskelman tuottamaa prosentuaalista merkitystä sekä tehostaisi, että yhtenäistäisi projektin toimintaa.

CI-järjestelmien avulla pystytään tarkastelemaan myös koodin toimivuuden lisäksi koodin laatua. Tällä hetkellä koodin laatu tarkistetaan manuaalisesti toisen kehittäjän toimesta. Manuaalityö ei ole aina huono asia, eikä sitä saa poistaa koskaan kokonaan. Automatisoinnilla pystyttäisiin kuitenkin nopeuttamaan manuaalitarkastuksia, kun koodin laatu on jo kertaalleen tarkistettu CI-järjestelmän toimesta. CI-järjestelmiin pystytään erilaisten liitännäisten avulla saamaan esimerkiksi laadun testaus. Yksi tällainen liitännäinen on SonarQube, joka pystytään integroimaan CI:n toimintaa suoraan työnkulkukaavioon. SonarQube tukee monia ohjelmointikieliä mukaan lukien C# ja siinä on sisäänrakennettu tuki .NET alustalle (SonarQube Docs 2022).

Taskien kehityksessä käytetään erilaisia nimiä, kun luodaan Taskien työnkulkukaavioita. Useimmat kehittäjät noudattavat tiettyä nimeämissääntöä, mutta mikäli projektin kehittäjien määrä kasvaa tai henkilöstö muuttuu, tulisi projektille asettaa standardisoitu nimeämissääntö helpottamaan työnkulun tiedostojen tarkastelua. Tämä helpottaisi näiden tiedostojen nimeämistä sekä loisi standardoidun tavan kehittää projektin sisälle.

4.4 Lopputulos

CI-järjestelmien vertailutuloksien perusteella voidaan todeta, ettei toimeksiannon projektiin tarvita uutta CI-järjestelmää. Uuden järjestelmän myötä kehittäjiltä vaaditaan aikaa uuden työkalun opetteluun sekä mikäli toinen vertailukohteena oleva järjestelmä valittaisiin, tarvittaisiin asentaa työkalu jokaisen projektin kehittäjän koneelle. Nykyinen CI-järjestelmä GitHub Actions toimii tarkoituksenmukaisesti ja tarjoaa tarpeellisen määrän toimintoja projektin tuotoksien testaamiseen. Nykyistä järjestelmää verraten Jenkins järjestelmään voidaan todeta, ettei tämä tuo lisäarvoa vaan vaatisi enemmän kehittäjiltä sekä loisi haasteita projektin henkilöstön kasvattamiseen sekä

perehdyttämiseen. TeamCity varmasti helpottaisi kehityskanavien luomista sekä poistaisi tarpeen opetella nykyisin käytössä oleva YAML kieli, mutta se ei tuo paljon muuta projektin käytettäväksi. Projektin tarkoituksena on luoda avoimeen lähdekoodiin perustuvia tuotteita, jolloin samaan perustuva CI-järjestelmä sopii hyvin käyttötarkoituksiin.

Kaikkien kehityskohteiden ja CI-järjestelmien vertailun lopputuloksena todettakoon, ettei nykytilaan tarvita tehdä montaakaan muutosta tehostamaan projektin toimintaa. Nykyinen CI-järjestelmä GitHub Actions toimii loppujen lopuksi tarpeisiin sopivalla tasolla eikä toinen järjestelmä toisi paljoa lisäarvoa projektin toiminnalle. Projektiryhmä on nykyisellä tasolla pieni ja muodostuu vain muutamasta kehittäjästä. Tämän takia toimintamallien standardisoimisella sekä muutamien käyttömallien muuttamisella pystyttäisiin helposti tehostamaan riittävästi toimintaa.

5 Pohdinta

Opinnäytetyön tavoitteena oli luoda nykytilakartoitus toimeksiannolle sekä kehitysehdotus CI-järjestelmän käytöstä. Työn tavoitteet saavutettiin ja opinnäytetyön lopputuloksena syntyi kehitysehdotus CI-järjestelmän käyttöön ja nykyisiin toimintamalleihin. Lisäksi nykyiset toimintatavat saatiin dokumentoitua toimeksiannon käyttöön.

Aihe on selvästi ajankohtainen, koska CI:n käyttö ja tärkeys nousee jatkuvasti kehitysprojekteissa. CI:n avulla pystytään toteuttamaan projekteja nopeammin ja säilyttämään laadukas tuote. Nykytilakartoituksessa sekä kehitysehdotuksessa kerätyt tiedot tulevat auttamaan toimeksiannon projektin toimintaa.

Työn aihealue on todella laaja ja tässä työssä esiin tuodut kohdat ovat pintaraapaisuja koko aiheeseen verrattuna. Aiheen suomennokset ovat myös vakiintumattomia, joten kaikkien termien suomennokset eivät vastaa aina samaa englanninkielistä termiä. CI-järjestelmien vertailu osoittautui haasteelliseksi, koska eri järjestelmiä on sen verran monta ja kaikki toimivat suhteellisen samankaltaisesti. Vertailun tuloksien pohjalta pystytään tekemään johtopäätös, ettei CI-järjestelmän valinnalla ole paljon vaikutusta, kun projektilta löytyy tarpeeksi asiantuntijuutta käsitellä järjestelmää. Monet maksulliset CI-järjestelmät eroavat maksuttomista ainoastaan sillä, että järjestelmä on valmiiksi konfiguroitu ja järjestelmän käyttöliittymä on helpompi ohjattava ja selkeämpi. Toimintatavat ja parhaat käytänteet ovat kuitenkin samoja kaikkien järjestelmien kanssa.

Projektin aikana minulle tuli kiire, koska motivaation ja ajan vähentyminen lisäsi stressitilaa ja vaikeutti projektin etenemistä. Tämä oli yksi projektiin listattu riski, joka toteutui, vaikka yritin tätä riskiä välttää suunnittelemalla projektin huolellisesti ja asettamalla itselleni tavoitteita ja määräaikoja. Tästä huolimatta sain työni suoritettua tavoite ajassa.

Opinnäytetyön aikana opiskelija pääsi syventymään CI:n toimintamalleihin ja käytäntöihin. Opiskelija oppi myös paljon eri CI-järjestelmistä sekä sai uusia näkökulmia Frends projektin käyttämään CI-järjestelmään. Opinnolliset tavoitteet täyttyivät myös, koska opiskelija pääsi syventymään CI:n toimintaan paljon enemmän ja oppi tuntemaan CI toimintamalleja. Opinnäytetyön aikana opiskelija oppi tulkitsemaan paremmin CI-järjestelmän työnkulkukaavioita sekä ajattelemaan CI:n toimintatapojen mukaisesti.

Lähteet

Altexsoft. 31.1.2022. CI/CD Tools Comparison: Jenkins, TeamCity, Bamboo, Travis CI, and More. Luettavissa: https://www.altexsoft.com/blog/engineering/cicd-tools-comparison/. Luettu: 15.9.2022.

Beck, K. 2002. Test Driven Development: By Example. Addison-Wesley Professional. E-kirja. Luettu: 6.11.2022.

Ben, N. 29.4.2022. Top 5 CI/CD best practices. Circleci Blog. Luettavissa: https://circleci.com/blog/top-5-ci-cd-best-practices/?utm_source=google&utm_medium=sem&utm_campaign=sem-google-dg--emea-en-dsa-maxConv-auth-brand&utm_term=g_-_c__dsa_&utm_content=&gclid=CjwKCAjwtp2bBhAGEiwAOZZTuKhKE-4qj81lq5l5dCS6sgoRp6Vxt4MQRvS_BVo23xmBoNIa0wFoCRoCk5IQAvD_BwE. Luettu: 6.11.2022.

Dingare, P. P. 2022. CI/CD Pipeline Using Jenkins Unleashed: Solutions While Setting up CI/CD Processes. Apress L. P. Berkeley, CA. E-kirja. Luettu: 15.9.2022.

Fosco, M. 6.10.2022. What is a CI/CD pipeline?. Circleci Blog. Luettavissa: https://circleci.com/blog/what-is-a-ci-cd-pipeline/. Luettu: 10.10.2022.

Frends. 2022. What is and iPaas?. Luettavissa: https://frends.com/platform/what-is-an-ipaas. Luettu: 29.10.2022.

Gaba, I. 16.6.2022. What is Continuous Integration and Why it is Important?. Simplilearn. Luettavissa: https://www.simplilearn.com/tutorials/devops-tutorial/continuous-integration. Luettu: 25.9.2022.

GitHub Docs. 2022. Understanding GitHub Actions. Luettavissa: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions. Luettu: 25.10.2022.

GitLab. 2022. Continuous integration best practices. Luettavissa: https://about.gitlab.com/topics/ci-cd/continuous-integration-best-practices/. Luettu: 6.11.2022.

InvoiceNinja. 2022. Why Software Development Companies Should Use CI/CD. Luettavissa: https://invoiceninja.com/why-software-development-companies-should-use-ci-cd/. Luettu: 1.11.2022.

Kaftzan, J. 28.9.2022. CI/CD Security: How to Keep your CI/CD Pipelines Secure. Armo. Luettavissa: https://www.armosec.io/blog/ci-cd-security/. Luettu: 2.10.2022.

Kumar, R. 6.4.2022. What is TeamCity and How it works? An Overview and Its Use Cases. DevOpsSchool. Luettavissa: https://www.devopsschool.com/blog/what-is-teamcity-and-how-it-works-an-overview-and-its-use-cases/. Luettu: 25.10.2022.

Levan, M. 9.10.2020. How to put CI, CT and CD together in a DevOps pipeline. TechTarget. Luettavissa: https://www.techtarget.com/searchsoftwarequality/tip/How-to-put-CI-CT-and-CD-together-in-a-DevOps-pipeline. Luettu: 28.9.2022.

Mahalingam S., M. 2014. Learning continuous integration with TeamCity. Packt Publishing. E-kirja. Luettu: 6.11.2022.

Microsoft. 2022. What is .NET?. Luettavissa: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet. Luettu: 29.10.2022.

Ohjelmistokehityksen menetelmät. Sanasto Git. Luettavissa: https://book.sovelluskontti.com/versionhallinta/sanasto. Luettu 29.10.2022.

Philip, R. 24.6.2022. What Is Test-Driven Development And What Are Its Benefits?. ThinkPalm. Luettavissa: https://thinkpalm.com/blogs/what-is-test-driven-development-and-what-are-its-benefits/. Luettu: 20.9.2022.

Red Hat. 18.6.2021. What is YAML?. Luettavissa: https://www.redhat.com/en/topics/automation/what-is-yaml. Luettu: 6.11.2022.

Rehkopf, M. 2022. Continuous integration tools. Atlassian. Luettavissa: https://www.atlassian.com/continuous-delivery/continuous-integration/tools. Luettu: 16.9.2022.

Roddewig, S. 11.2.2022. Continuous Integration: What It Is and How to Achieve It. HubSpot. Luettavissa: https://blog.hubspot.com/website/continuous-integration. Luettu: 25.9.2022.

Ruck, D. 17.5.2021. What Is Continuous Integration. Earthly. Luettavissa: https://earthly.dev/blog/continuous-integration/. Luettu: 5.9.2022.

Sacolick, I. 15.4.2022. What is CI/CD? Continuous integration and continuous delivery explained. InfoWorld. Luettavissa: https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html. Luettu: 25.9.2022.

Scarlett, R. 3.6.2022. A beginner’s guide to CI/CD and automation on GitHub. GitHub Blog. Luettavissa: https://github.blog/2022-06-03-a-beginners-guide-to-ci-cd-and-automation-on-github/. Luettu: 25.9.2022.

Silverthorne, V. 3.2.2022. How to keep up with CI/CD best practices. GitLab. Luettavissa: https://about.gitlab.com/blog/2022/02/03/how-to-keep-up-with-ci-cd-best-practices/. Luettu: 1.11.2022.

SoftCo. 2022. On-Premise Software. Luettavissa: https://softco.com/glossary/on-premise/. Luettu: 8.11.2022.

SonarQube Docs. 24.8.2022. SonarScanner for .NET. Luettavissa: https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-msbuild/. Luettu: 1.11.2022.

TeamCity. Building .NET Projects Using TeamCity. JetBrains. Luettavissa: https://www.jetbrains.com/teamcity/tutorials/dotnet-build-configure-test/. Luettu: 6.11.2022.

Terrell, H. 2022. What is a build?. TechTarget. Luettavissa: https://www.techtarget.com/searchsoftwarequality/definition/build. Luettu: 12.9.2022.

Tsitoara, M. 2020. Beginning Git and GitHub: A comprehensive guide to version control, project management, and teamwork for the new developer. Apress LLC. E-kirja. Luettu: 6.11.2022.

Ziolkowski, D. 13.7.2022. 6 CI/CD best practices you need to know. Architect. Luettavissa: https://www.architect.io/blog/2022-07-13/six-cicd-best-practices/. Luettu: 5.10.2022.

Liitteet

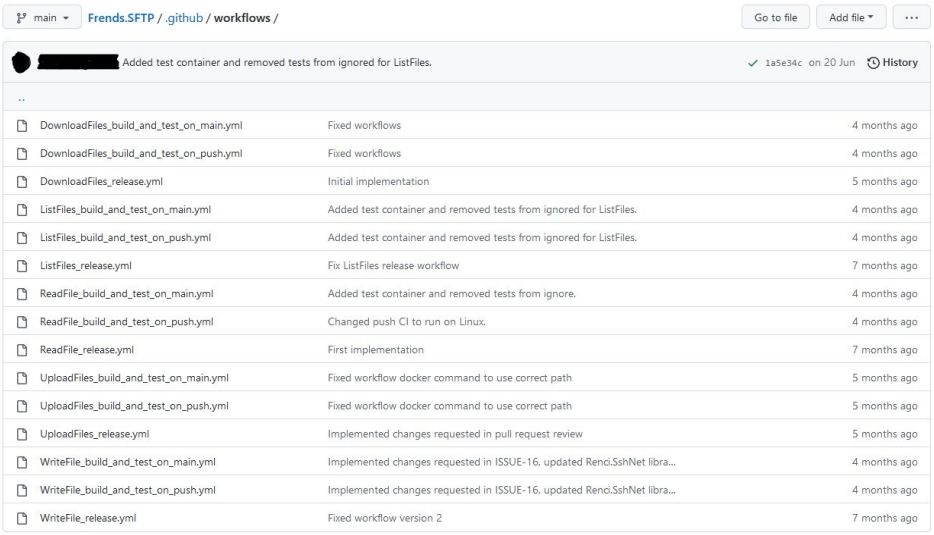
Liite 1. Nykytilakartoitus

CI/CD usage documentation

Frends Tasks project uses GitHub Actions CI system to handle automated Task testing to prevent build fails in the main branch. Every Task needs to have three individual workflows which handles the build testing when pushing separate remote branches, merging the remote branches to the main / master branch and when releasing the Task to the public Task feed. Workflows needs to be added to the root of the Tasks' main repository. The Task's main repository should have all the Tasks' workflows in /.github/workflows/ folder.

There is no systematic naming convention for the Tasks' individual workflows but generally used conventions are:

- <Task name>_build_and_test_on_push.yml
- <Task name>_build_and_test_on_main.yml
- <Task name>_release.yml

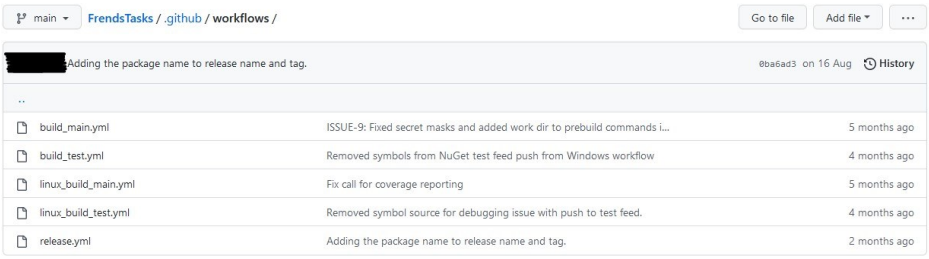


- Base workflows
- CI Architecture overview
- Example use-cases
    - Build Docker container using prebuild command in the workflow
    - Build Docker container using docker-compose file
    - Build docker container with a script

Base workflows

Frends Tasks project's CI is constructed of three individual base workflows which are in the public FrendsTasks repository.

Here are all the base workflows. There are separate workflows for Linux and Windows build for the build_main and build_test workflows.



Base workflows are responsible of setting environment variables for the workflow, running the prebuild command set in the Task's own individual workflow, building and testing the Task, collecting test coverage results and packing the release version of the task and sending it to either to the test feed or production feed.

Base workflows can be used by creating individual workflow for the task and setting build job for it using the base workflow, see image below.

In this example the Linux based version of the base workflow is used. This will enable e.g., Linux based docker image to be used when testing the Task. Prebuild command can be set as in the example image. Example shows how to set up a docker container as a testing platform.

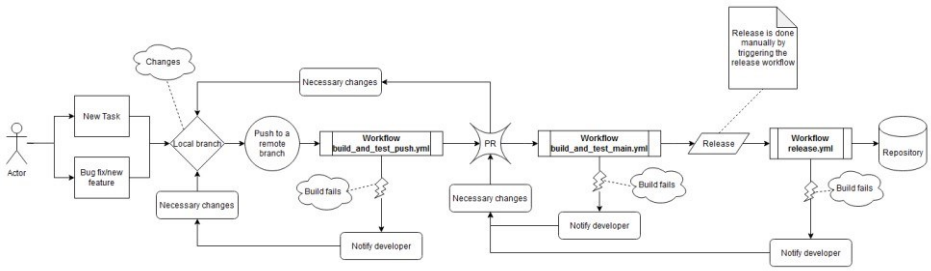
The release base workflow does not handle testing of the Task but only the release. This is because the testing has been done when changes has been pushed to a remote branch and when the remote branch has been merged to the main branch.

CI Architecture overview

CI workflows are triggered in three places:

- local branch is pushed to a remote branch
- PR is accepted and the branch is merged to the main branch
- Task is released to the Task feed

Frends Tasks project's CI is built in a way that the build_test and build_main workflows are automatically triggered, when necessary, event has happened. The first workflow build_and_test_push.yml is triggered when local branch is push to a remote branch which is not the main/master branch. The second workflow build_and_test_main.yml is triggered when the remote branch is merged to the main/master branch. Finally, the release workflow needs to be triggered manually from the GitHub actions tab when the Task is ready to be published. This is done so to ensure that the Task feed only gets the Tasks that are ready for production. Below you can see architecture diagram which shows the places where the CI workflows are triggered.



Example use-cases

Build Docker container using prebuild command in the workflow

For this use case you need to write the docker building command straight to the workflow's prebuild_command:

Example:

```
jobs:
  build:
    uses: FrendsPlatform/FrendsTasks/.github/workflows/linux_build_main.yml@main
    with:
      workdir: Frends.PostgreSQL.ExecuteQuery
      prebuild_command: docker run -p 5432:5432 -e POSTGRES_PASSWORD=mysecretpassword -d postgres
```

This can be used whenever the build command is not too long, and complicated or additional configurations are not needed.

Build Docker container using docker-compose file

For this you can use docker-compose file with or without DockerFile or script that is run in the prebuild_command of the workflow.

Example:

```

jobs:
  build:
    uses: FrendsPlatform/FrendsTasks/.github/workflows/linux_build_test.yml@main
    with:
      workdir: Frends.SFTP.DownloadFiles
      prebuild_command: docker-compose -f ./Frends.SFTP.DownloadFiles.Tests/docker-compose.yml up -d
```

Here a docker-compose file is used which is then called in the prebuild command. Dockercompose file has all the instructions to build the docker container.

main Frends.SFTP / Frends.SFTP.DownloadFiles / Frends.SFTP.DownloadFiles.Tests / **docker-compose.yml**

RikuVirtanen PR review changes ✓

1 contributor

16 lines (13 sloc) | 294 Bytes

```
1  version: '3'
2
3  # Usage https://github.com/atmoz/sftp
4
5  services:
6   sftp_server:
7    build:
8      context: .
9    ports:
10    - "2222:22"
11   volumes:
12    - ./Volumes/ssh_host_rsa_key.pub:/home/foo/.ssh/keys/ssh_host_rsa_key.pub:ro
13    - ./Volumes/share:/home/foo/share
14   command:
15    - "foo:pass:::upload"
16
```

Here is an example of a docker-compose file which uses Dockerfile to create the container. Dockerfile can have specific instructions which are implemented after the container is up and running.

main Frends.SFTP / Frends.SFTP.DownloadFiles / Frends.SFTP.DownloadFiles.Tests / **Dockerfile**

RikuVirtanen Fixed mistake in Dockerfile ✓

1 contributor

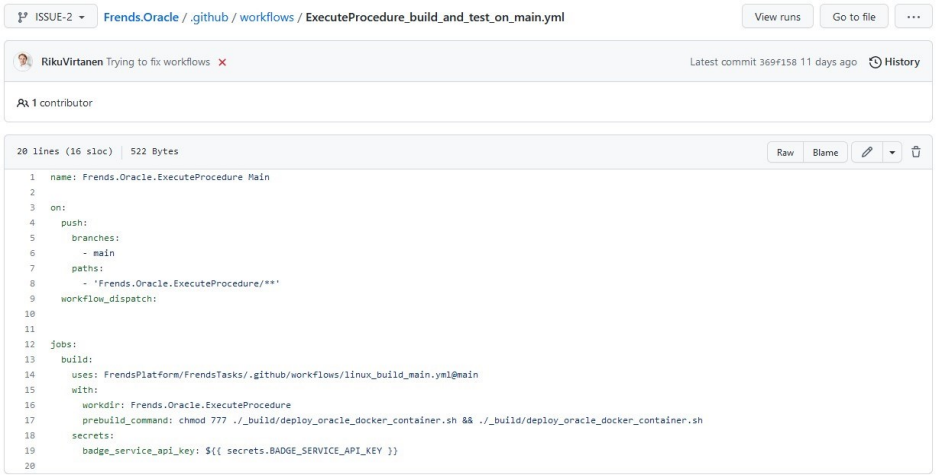
5 lines (3 sloc) | 101 Bytes

```
1  FROM atmoz/sftp
2
3  ARG DEBIAN_FRONTEND=noninteractive
4
5  COPY ./Volumes/sshd_config /etc/ssh/sshd_config
```

In this example Dockerfile instructs the docker-compose build the container using atmoz/sftp image and copy sshd_config file from local directory to that container.

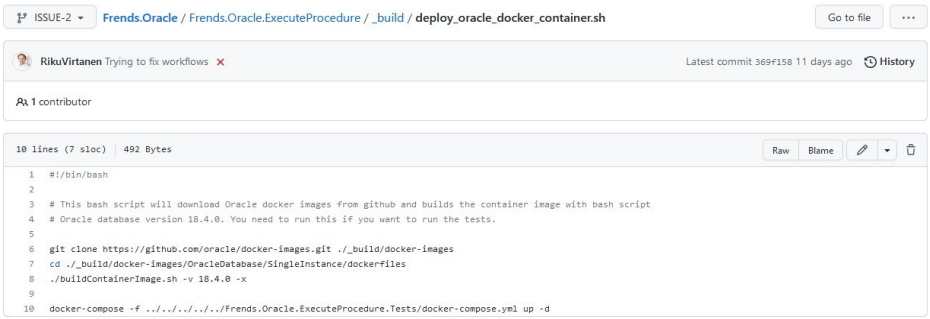
Build docker container with a script

Scripts can also be added to the workflow's prebuild command:



Here a script is used with a command 'chmod 777' to give the CI permissions to the file in question after which the script file is run.

Scripts are great when additional operations are needed when building docker containers.



In this example a script handles the downloading of the folder which holds the docker image and the running of the script inside that folder. Finally, the script runs docker-compose file which builds the actual container. This is usually needed when the Docker Hub doesn't have the needed image.

This example can also be handled by running the docker commands in the script file itself without needing the docker-compose file.