Opinnäytetyö (AMK)

Tieto- ja viestintätekniikka

2023

Riko Saarinen

MyE.Way CI/CD-putki

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja viestintätekniikka

2023 | 34 sivua

Riko Saarinen

MyE.Way CI/CD-putki

Tämä opinnäytetyö käsittelee jatkuvan integroinnin ja jatkuvan toimittamisen (CI/CD) putken toteuttamista GitLab-alustalla olemassa olevaan ohjelmistoprojektiin nimeltä MyE.Way, joka on sähköinen urheiluvalmennuksen tukijärjestelmä. Työn tekemisen syy oli tarve automatisoida testausprosessi, ja julkaisuprosessi projektissa, jotta voitaisiin olla varmoja siitä, että palvelimelle ei mene rikkinäistä koodia ja että julkaisuprosessi olisi yksinkertaisempi.

Teoriaosuudessa annettaan yleiskatsaus CI/CD:n käsitteistä, hyödyistä ja mahdollisista huonoista puolista. Osuudessa käydään myös läpi erilaisia CI/CD-työkaluja, joita projektissa käytettiin ja DevOps metodologiaa.

Opinnäytetyön käytännöllisessä osassa luodaan CI/CD-putki MyE.Way projektiin käyttäen GitLab alustaa ja sen CI/CD ominaisuuksia. Osassa käydään läpi, miten CI/CD-putki rakennetaan GitLab alustaa käyttäen, kun projektin koodi on jaettu kahteen eri repositorioon ja annetaan lyhyt kuvas MyE.Way projektista, johon CI/CD-putki tehtiin.

Lopputuloksena on CI/CD-putki, joka helpottaa ohjelmistokehittäjien työtä, mahdollistaa ohjelman ja koodin automaattisen testaamisen ennen kuin koodi lähetetään testipalvelimelle automaattisesti.

Asiasanat:

CI/CD, DevOps, GitLab, MyE.Way, GitLab Runner, Docker

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2023 | 34 pages

Riko Saarinen

MyE.Way CI/CD-pipeline

The objective of this thesis was to implement a continuous integration and continuous delivery (CI/CD) pipeline on the GitLab platform for an existing software project called MyE.Way, which is a electronic sport coaching support system. The reason for carrying out this work was the need to automate the testing and deployment process in the project, in order to be sure that no broken code is sent to the server and that the deployment process is simplified.

The theoretical part of the thesis gives an overview of CI/CD concepts, benefits and possible disadvantages. The section also covers various CI/CD tools that were used in the project and DevOps methodology.

In the practical part of the thesis, the CI/CD pipeline was integrated into the MyE.Way project using the GitLab platform and it’s CI/CD tools. This section details the process of building a CI/CD pipeline on the GitLab platform when the project uses multiple repositories and provides a brief overview of the MyE.Way project, for which the CI/CD pipeline was developed.

The end result was a ready-to-use CI/CD pipeline that helps developers with their work by automating testing of the software and code before the code is automatically sent to the test server.

Keywords:

CI/CD, DevOps, GitLab, MyE.Way, Gitlab Runner, Docker

Sisältö

Kuvat

Käytetyt lyhenteet tai sanasto

E2e	End to end
NPM	Node Package Manager
CI	Continuous integration
CD	Continuous deployment / delivery

1 Johdanto

Ohjelmistokehityksessä testaaminen on tärkeä osa uuden ohjelman rakentamista. Testauksen avulla voidaan varmistaa ohjelman laadun ja toiminnan. Ilman säännöllisen ja automatisoidun testauksen toteutusta, ohjelmistoprojektien virheiden ja ongelmien määrä voi kasvaa merkittävästi varsinkin suurissa ohjelmissa, mikä voi viivästyttää uuden ohjelman tai ohjelmaan päivityksen rakentamista ja lisätä projektin kustannuksia. CI/CD-putki on yksi apu tähän haasteeseen, CI/CD koostuu jatkuvasta integraatiosta (CI) ja jatkuvasta toimituksesta / käyttöönotosta (CD). CI/CD-putki tarjoaa säännöllisiä ja automatisoituja testauksia ja automaattisen julkaisuprosessin ohjelmistoprojektiin.

Turun ammattikorkeakoulun terveysteknologian testialustaympäristössä Health Tech Labissa rakennetaan uutta versiota MyE.Way palvelusta, MyE.Way on sähköinen urheiluvalmennuksen tukijärjestelmä, jonka omistajana on Eerikkilä Sport & Outdoor Resort (MyEway - Eerikkilä, ei pvm.). MyE.Way:n uusi versio pyrkii etenkin vastaamaan joustavasti monenlaisten käyttäjäryhmien erilaisiin ja muuttuviin tarpeisiin. Tähän uuteen versioon tulee olemaan paljon helpompi tehdä mahdollisia muutoksia, joita Eerikkilä Sport & Outdoor Resort haluaa, mutta mitä enemmän muutoksia ja päivityksiä tähän uuteen versoon tehdään, sitä vaikeammaksi uusien versioiden testaaminen menee. CI/CD-putki auttaa tässä tilanteessa hyvin paljon ohjelmistokehittäjien työtä automatisoimalla projektin testausprosessin, jolloin kehittäjät voivat keskittyä enemmän uusien ominaisuuksien tekemiseen. Julkaisuprosessin automatisointi helpottaa, ja yksinkertaistaa myös kehitystiimin ja ylläpitotiimin työtä projektissa huomattavasti vähentämällä manuaalista ihmisen väliintuloa julkaisussa.

Opinnäytetyön päätehtävänä on luoda CI/CD-putki käytettäväksi MyE.Way projektiin. Ensin opinnäytetyössä käydään läpi mitä CI/CD on, mitä DevOps tarkoittaa ja mitä työkaluja projektissa käytetään. Tämän jälkeen alkaa soveltava osuus, jossa rakennetaan tämä CI/CD-putki, sen toimintaa testataan ja se on lopussa valmis käyttöön MyE.Way projektissa.

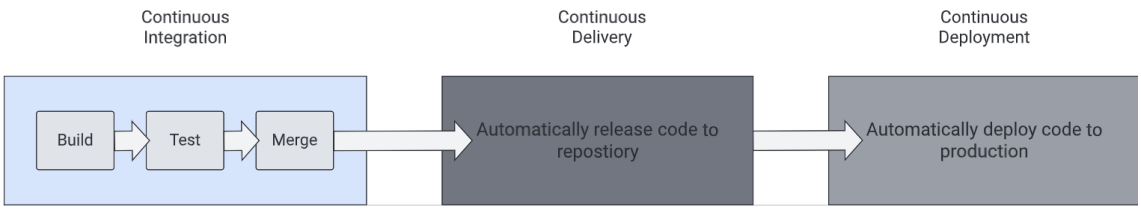
2 CI/CD:n yleistiedot, työkalut ja DevOps

2.1 CI/CD:n tarkoitus

CI/CD-putki on sarja tehtäviä, jotka suoritetaan uuden ohjelmiston version tuottamisen yhteydessä ohjelmistokehityksessä. CI/CD kuuluu DevOps-metodologian alle ja sen päätehtävä on automatisoida suurin osan kaikista manuaalisista ihmisen väliintulosta, jota perinteisesti tarvitaan uuden koodin lähettämisessä tuotanto / testipalvelimelle ja koodin testaamisessa. (What is CI/CD? | GitLab, ei pvm.)

Akronyymillä CI/CD on pari eri tarkoitusta. CI akronyymissä CI/CD tarkoittaa jatkuvaa intergraatiota, joka on automaatio prosessi. Oikein rakennettu CI tarkoittaa sitä, että uudet muutokset sovellukseen rakennetaan ja testataan automaattisesti käytetyn alustan avulla (GitLab tässä projektissa). (What Is CI/CD? | RedHat, 2022)

CD akronyymissä CI/CD tarkoittaa jatkuvaa toimitusta / jatkuvaa käyttöönottoa. Molempia termejä ”jatkuva toimitus” ja ”jatkua käyttöönotto” käytetään usein sekaisin, mutta niitä voi myös käyttää erikseen esittämään kuinka paljon automatisointia on CI/CD-putken loppuvaiheessa tapahtumassa. Erikseen ”jatkuva toimitus” tarkoittaa yleensä koodin automaattista testaamista ja testatun koodin lähettämistä alustalle, jossa koodia ylläpidetään (kuten GitLab). ”Jatkuva käyttöönotto” tarkoittaa taas testatun koodin lähettämistä tuotanto / testipalvelimelle, jossa muut käyttäjät pystyvät testaamaan uusia muutoksia. (What Is CI/CD? | RedHat, 2022)



Kuva 1. CI/CD-putken vaiheet.

On hyvä pitää mielessä, että CD:n kaksi eri vaihetta usein tarkoittaa eri asioita riippuen keneltä kysyy ja joskus CD on ainoastaan yksi vaihe eikä kaksi erillistä vaihetta. Yleisesti projekteissa aloitettaan rakentamalla CI osa ja vasta sen jälkeen CD osa, koska on helpompi rakentaa CD viimeisenä, kun koodin testaus ja rakentaminen on jo toteutettu CI osassa. (What Is CI/CD? | RedHat, 2022)

2.2 DevOps

DevOps on IT-alalla käytetty metodologia ja koostuu ohjelmistokehityksestä (dev) ja tuotannosta (ops). DevOps keskittyy nopeaan pienien päivitysten tuottamiseen, integroimaan ohjelmistokehitystiimien ja operaatiotiimien työskentelyn edistämällä yhteistyökulttuuria ja yhteisvastuuta. (What Is DevOps? | GitLab, ei pvm.)

DevOps metodologia koostuu neljästä keskeisestä periaatteesta, jotka ohjaavat ohjelmistokehityksen ja käyttöönoton tehokkuutta:

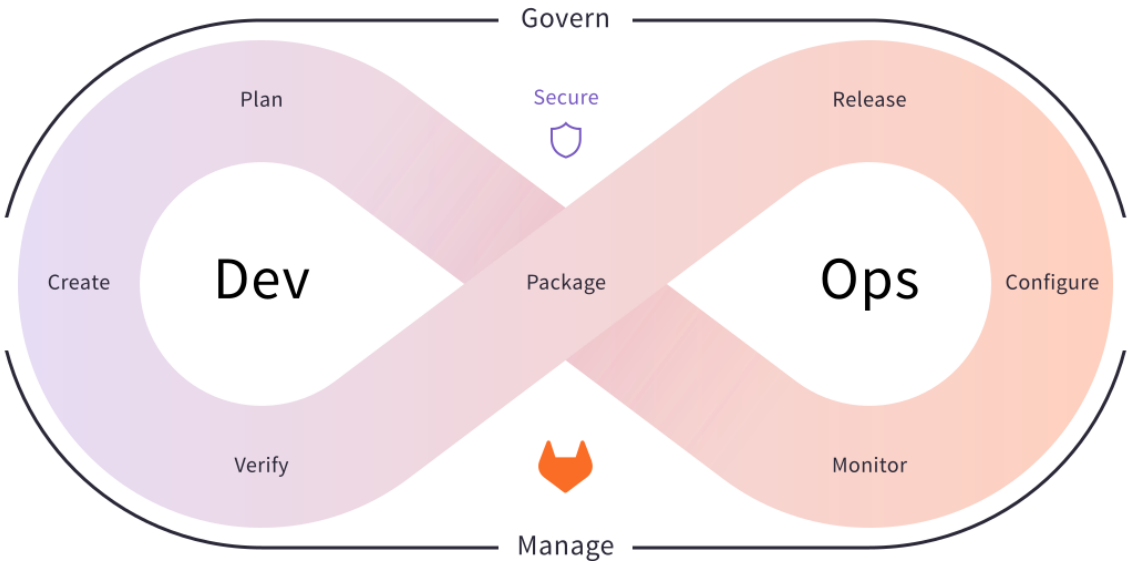
1. **Ohjelmistokehityksen elinkaaren automatisointi.** Tämä sisältää automaattisen testauksen, koontiversioiden luomisen, julkaisemisen ja muita manuaalisia tehtäviä, jotka hidastavat päivitysten tekemistä tai aiheuttaa inhimillisiä virheitä ohjelmistokehityksessä.
2. **Yhteistyö ja viestintä.** Hyvässä DevOps-tiimissä on automatisointia, mutta myös tehokasta yhteistyötä ja viestintää.
3. **Jatkuva parannus ja jätteen (eng. waste) minimointi.** Tehokkaat DevOps-tiimit etsivät jatkuvasti alueita, joita voitaisiin parantaa, esimerkiksi toistuvien tehtävien automatisointia, keinoja lyhentää julkaisemisaikoja tai keskimääräistä palautumisaikaa.
4. **Keskittyminen käyttäjien tarpeisiin lyhyellä palautesilmukalla.** DevOps-tiimit pystyvät keskittymään siihen, mitä asiakkaat haluavat ja miten se voidaan toteuttaa käyttäen automatisointia, parempaa kommunikointia ja yhteistyötä sekä jatkuvaa parannusta.

(What Is DevOps? | GitLab, ei pvm.)

DevOps elinkaari ulottuu ohjelmistokehityksen alusta toimitukseen, ylläpitoon ja turvallisuuteen. DevOpsin elinkaaren osat ovat:

- **Suunnittele:** Järjestä työtehtävät tärkeysjärjestykseen ja seuraa niiden valmistumista.
- **Luo:** Kirjoita, suunnittele, kehitä ja hallitse projektitietoja turvallisesti tiimisi kanssa.
- **Vahvista:** Varmista, että koodisi toimii oikein ja noudattaa laatustandardeja ihanteellisesti automaattisen testauksen avulla.
- **Paketoi:** Pakkaa sovelluksesi ja riippuvuutesi, hallitse kontteja ja rakenna esineitä (eng. artifact).
- **Suojaa:** Tarkista mahdolliset haavoittuvuukset staattisten ja dynaamisten testien, fuzz-testauksen ja riippuvuustarkistuksien avulla.
- **Julkaise:** Julkaise ohjelmisto loppukäyttäjien käyttöön.
- **Määritä:** Halinnoi ja määritä tarvittavat infrastruktuurit sovellusten tukemiseksi.
- **Valvo:** Seuraa suorituskykymittareita ja virheitä vähentääksesi pahoja tilanteita.
- **Hallita:** Hallitse tietoturva-aukkoja, käytäntöjä ja vaatimustenmukaisuutta koko organisaatiossa.

(What Is DevOps? | GitLab, ei pvm.)



Kuva 2. DevOps-elinkaari (What Is DevOps? | GitLab, ei pvm.).

2.3 CI/CD:n hyvät ja huonot puolet

CI/CD:n käytössä on monia hyviä puolia kuten:

- Nopeampi päivitysten tuottaminen ja lähettäminen tuotanto- ja testipalvelimelle.
- Vähentää mahdollisuuksia ihmisen tekemille virheille päivitysten lähettämisessä tuotanto- ja testipalvelimelle.
- Koodin laatu paranee, kun mahdolliset virheet, ja ristiriidat voidaan löytää ja korjata ajoissa.
- Organisaation tuottavuus paranee.
- Julkaisu- ja testaus prosessi yksinkertaistuu.

CI/CD:n käytössä on kuitenkin myös mahdollisia huonoja puolia:

- CI/CD:n rakentaminen vie paljon aikaa, resursseja ja tietoa.
- CI/CD:n ylläpito vie myös paljon resursseja, joita varsinkin pienillä tiimeillä ei ole paljoa.
- CI/CD:n käyttö nostaa mahdollisia turvallisuusriskejä, varsinkin jos sitä ei ole toteutettu turvallisuutta huomioon ottaen.

2.4 GitLab

CI/CD-putken rakentamiseen on monia eri alustoja kuten Jenkins, CircleCI, GitLab ja Github. MyE.Way-projektin koodi ylläpidetään GitLab repositoriossa, joten tässä työssä käytetään GitLabia CI/CD-putken rakentamiseen.

GitLab on web-pohjainen Git-repositorio, joka tarjoaa ilmaisia avoimia ja yksityisiä repositorioita koodin tallentamiseen ja ylläpitoon. GitLab sisältää myös ongelmanseurantaohjelman, integroitu kehitysympäristön, wikitoiminnallisuuden ja CI/CD-ominaisuuksia, joita tässä työssä käytetään. Nämä toiminnot tekevät GitLabista DevOps-alustan, joka mahdollistaa ammattilaiset suorittamaan kaikki

projektin tehtävät projektin suunnittelusta lähdekoodien hallintaan, seurantaan ja turvallisuuteen. Lisäksi se mahdollistaa tiimien yhteistyön ja paremman ohjelmiston kehittämisen. (Karin Kelley, 2023)

GitLabin CI/CD-putket sisältää:

- **Töitä (eng. jobs)**, jotka määrittävät mitä tehdään. Esimerkiksi työt, jotka rakentavat tai testaa koodin.
- **Vaiheita (eng. stages)**, jotka määrittä milloin työt suoritetaan. Esimerkiksi vaihe, jossa testataan koodi ja jälkeen tulee toinen vaihe, missä lähetetään koodi testipalvelimelle.

(CI/CD Pipelines | GitLab, ei pvm.)

GitLabilla on avoimen lähdekoodin ohjelma nimeltä GitLab Runner, jota käytetään GitLabin CI/CD-putken töiden suorittamiseen. GitLab Runnerin pystyy asentamaan omalle tietokoneelle, virtuaalikoneeseen, Docker konttiin ja pilvi infrastruktuuriin. GitLab Runner on tuettu Linux, Windows, macOS ja freeBSD käyttöjärjestelmissä. GitLab Runnereita on kolmea eri tyyppiä, jotka määrittävät millä tasolla GitLab Runner on käytettävissä GitLabissa:

- **Jaettu** GitLab Runner on käytössä kaikille projekteille kaikissa ryhmissä samassa GitLab instanssissa.
- **Ryhmä** GitLab Runner on käytössä kaikille projekteille samassa ryhmässä.
- **Projekti** GitLab Runner on käytössä yhdessä projektissa.

(Tech and Beyond With Moss, 2022)

GitLab Runnerin asentamisessa täytyy valita mitä suorittajaa käytät GitLab Runnerin kanssa. Suorittaja määrittää ympäristön missä CI/CD-putki suoritetaan ja vaihtoehtoja suorittajaksi on:

- **Virtuaalikone**
- **Shell**
- **Remote SSH**

- **Docker**
- **Kubernetes**
- **Mukautettu**

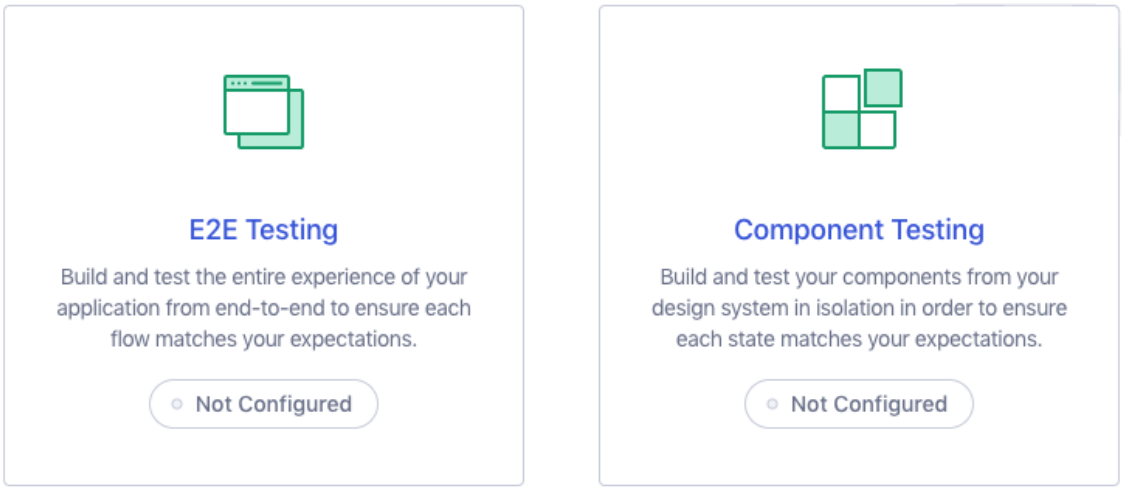
(Tech and Beyond With Moss, 2022)

2.5 CI/CD työkalut

Tämä osa käy läpi työkaluja, joita käytettiin CI/CD-putken tekemisessä apuna GitLab alustan lisäksi.

2.5.1 Cypress

Cypress on testaustyökalu, jonka avulla pystyy testaamaan nykyaikaisia verkkosovelluksia, korjaamaan mahdollisia virheitä visuaalisesti ja automaattisesti ajamaan testit CI-työkalujen kanssa. Cypressilla pystyy tekemään kahden tyyppisiä testejä, e2e-testejä ja komponenttitestejä. (Cypress Documentation, 2023)



Kuva 3. Cypress testaustyypit. (Opening the App | Cypress Documentation, ei pvm.)

E2e-testeissä Cypress menee nettisivulle ja tekee asioita käyttöliittymässä juuri niin kuin normaali käyttäjä tekisi, kuten esimerkiksi lomakkeiden lähettäminen ja tapahtumien lisääminen kalenteriin. Komponenttitesteillä, niin kuin nimi kertoo, testataan komponentteja nykyaikaisissa ohjelmistokehyksissä React, Angular, Vue ja Svelte. Komponenttitesteissä testataan tiettyjen elementtien toimintaa, tyylejä ja ulkonäköä, kuten nappeja, tekstikenttiä ja valintaruutuja tai suurempia osia nettisivusta, kuten lomakkeita, laatikoita ja navigointipalkkeja. (Cypress Documentation, 2023)

Cypress sisältää myös monia hyödyllisiä ominaisuuksia, jotka helpottaa nettisivun testaamista. Cypress lataa automaattisesti videopätkät ja kuvankaappaukset testeistä, joiden avulla pystyy katsomaan, mikä testeistä ei mennyt läpi ja mikä kohta nettisivusta on rikki. Cypressin avulla pystyy myös testaamaan monilla eri selaimilla nettisivua varmistaakseen, että se toimii kaikilla samalla tavalla, kuten Firefox- ja Chrome-pohjaiset selaimet (mukaan lukien Edge ja Electron). (Cypress Documentation, 2023)

2.5.2 ESLint

ESLint on avoimen lähdekoodin projekti, joka auttaa löytämään ja korjaamaan mahdollisia ongelmia JavaScript-koodissa. ESLintillä myös varmistetaan, että koodissa on kaikkialla samanlainen tyyli, joka helpottaa ohjelmistokehittäjien työtä. (ESLint - Pluggable JavaScript Linter, 2023)

2.5.3 Node Package Manager (NPM)

NPM on ohjelmistorekisteri, jossa on yli 800 000 koodipakettia. Avointa lähdekoodia rakentavat kehittäjät käyttävät sitä jakamaan ohjelmistoa ja monet yritykset käyttävät sitä myös yksityisen kehityksen hallintaan. (What Is Npm, ei pvm.)

NPM koostuu kolmesta osasta:

- Verkkosivu
- Komentorivikäyttöliittymä (CLI)
- Rekisteri

NPM verkkosivua voi käyttää pakettien etsimiseen, profiilien luomiseen ja muiden NPM-kokemuksen osa-alueitten hallitsemiseen. CLI toimii terminaalin kautta ja sen avulla ohjelmistokehittäjät käyttävät NPM:ää. Rekisteri on suuri julkinen tietokanta JavaScript-ohjelmistoa ja niihin liittyvää metatietoa. (About Npm | Npm Docs, ei pvm.)

NPM työkalua käytetään tässä työssä pakettien asentamiseen ja käyttöön, kuten ESLint ja Cypress.

2.5.4 Docker

Docker on virtualisointi ohjelma, joka helpottaa ohjelmiston kehittämistä ja julkaisua. Docker paketoi ohjelman pakettiin nimeltä kontti, jossa on kaikki mitä tämä ohjelmaa tarvitsee toimiakseen, eli se ohjelma itse, ja sen ajoympäristö ja tämän kontin voi sitten helposti jakaa eteenpäin. (TechWorld with Nana, 2023) On olemassa myös Docker kuva, joka on tiedosto, jota käytetään koodin suorittamiseen kontin sisällä. Docker kuva toimii ohjeena kontin rakentamiseen. (Alexander S. Gillis, 2021)

Ennen Dockerin käyttöä ohjelmistokehityksessä kehittäjät joutuivat asentamaan omalle koneelle jokaisen palvelun ja paketin, mitä ohjelmaan tarvittiin ja usein kehittäjille on eri versiot näistä palveluista, joka johtaa monenlaisiin virheisiin. Dockerin avulla näin ei tarvitse tehdä, koska kontti sisältää aina tietyn version palvelusta, ohjelman suorittamiseen kehittäjän täytyy ainoastaan asentaa Docker koneelleen ja suorittaa yksi komento ”docker run <image_name>”. Tämä komento hakee netistä kontin paketin, käynnistää sen koneelle ja tämä komento on aina sama riippumatta siitä mikä käyttöjärjestelmä kehittäjällä on ja mitä palvelua haetaan. (TechWorld with Nana, 2023)

Dockeria voi käyttää GitLab Runnerin suorittajana, ja se on yksi yleisemmistä suorittajista GitLab Runnereissa. GitLabin omat ylläpitämät Runnerit käyttävät Dockeria suorittajana. Dockerin avulla kaikki CI/CD-putken työt suoritetaan kontin sisällä, joten muita ohjelmistoja ei tarvitse asentaa koneelle, jossa GitLab Runner on asennettuna. Lisäksi Dockerin avulla on mahdollista valita, mitä palveluita tietyillä töillä käytetään. (TechWorld with Nana, 2022)

Jos valitsee Dockerin GitLab Runnerin suorittajaksi, täytyy myös antaa oletus Docker kuva mitä tämä GitLab Runner käyttää, kun se suorittaa CI/CD-putken työt. Esimerkiksi MyE.Way projektin GitLab Runnerit, joilla on Docker suorittajan käyttää Node:a oletuskuvana. Tämä tarkoittaa sitä, että jos CI/CD-putken työssä ei ole mainittu mitään Docker kuvaa erikseen, siinä käytetään tuota oletuskuvaa ja se antaa työlle käyttöön kaikki mitä tulee Node Docker kuvan mukana kuten NPM.

3 CI/CD-putken rakentaminen MyE.Way projektiin

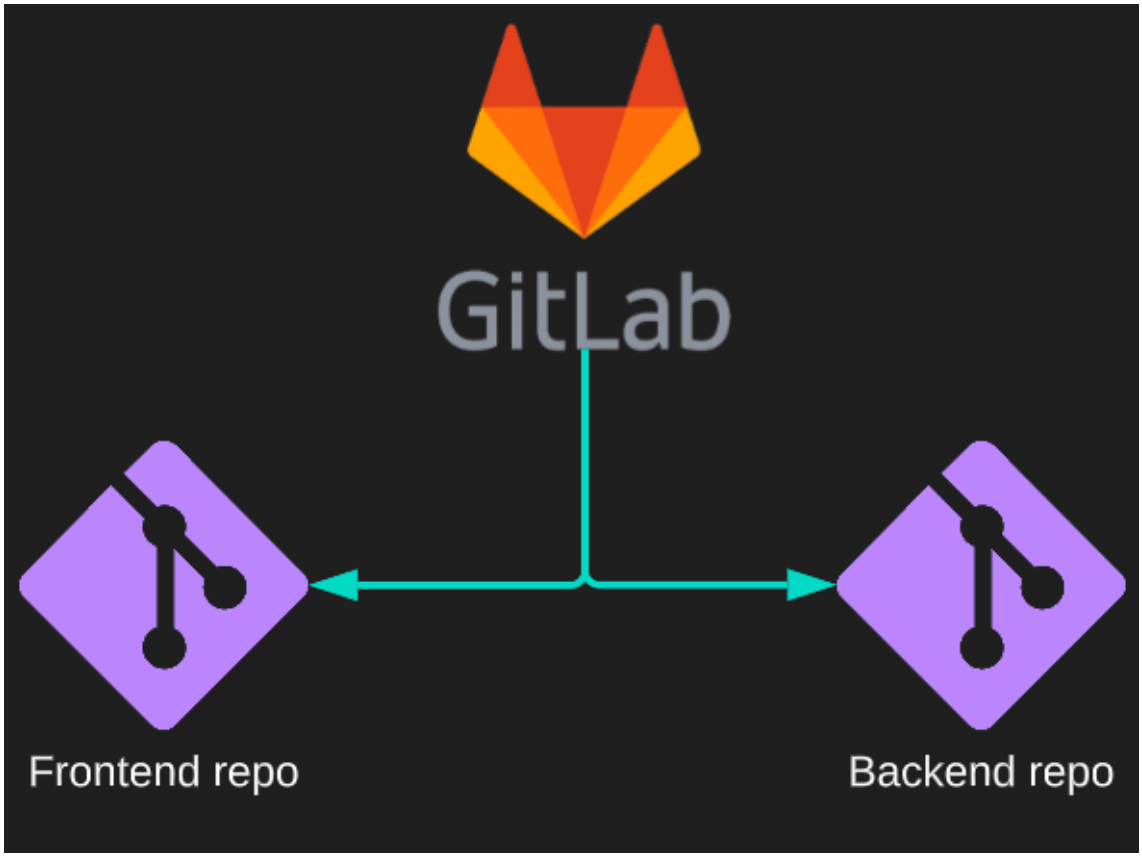
Tässä osassa käydään läpi MyE.Way projekti yleisesti, sen rakenne, asennetaan kaikki tarvittavat työkalut ja rakennetaan CI/CD-putki käytettäväksi projektiin.

3.1 MyE.Way yleisesti & projektin rakenne

MyE.Way on valmennuksen ja valmentajakoulutuksen lähtökohdista rakennettu sähköinen urheiluvalmennuksen tukijärjestelmä. (MyE.Way - Eerikkilä, ei pvm.)

MyE.Way on Turun ammattikorkeakoulun ylläpitämä ja kuuluu Eerikkilä Sport & Outdoor Resortin omistukseen. MyE.Way tukee pelaajien kehitystä, ja sitä käytetään Eerikkilän omassa toiminnassa ja monet urheiluseurat käyttävät sitä heidän toiminnassaan. MyE.Way-tietojärjestelmään kerätään tietoa pelaajien suorituksista ja valmentajat voivat hyödyntää näitä tietoja tehdäkseen parempia päätöksiä pelaajien valmentamisensuunnitelman suhteen. MyE.Way-tietojärjestelmän päätehtävä on auttaa erityisesti pelaajia, mutta myös valmentajia, kehittymään kohti kansainvälistä huipputasoa. (Tietojärjestelmän Avulla Kohti Urheilun Huippua - Eerikkilä, ei pvm.)

MyE.Way projekti ylläpidetään Turun ammattikorkeakoulun GitLab instanssissa. Projektin koodi on jaettu kahteen eri repositorioon, toisessa on käyttöliittymän (eng. frontend) koodi ja toisessa on taustajärjestelmän (eng. backend) koodi.



Kuva 4. MyE.Way projektin rakenne.

Jako kahteen repositorioon vaikeuttaa projektiin CI/CD-putken rakentamista. Tämä johtuu siitä, että kaikki testausvaiheet ja koodin julkaisu on käytävä läpi aina, kun jommankumman repositorion koodia päivitetään.

Tämän haasteen ratkaisemiseksi on useita eri tapoja, joista yksi on käyttää ”usean projektin putkia” (eng. downstream pipeline). Tässä ratkaisussa käytetään yhtä CI/CD-putkea toisen käynnistämiseen ja tätä käytettiin tässä projektissa. Projektissa voi siis tehdä niin, että yhdessä repositorioista on virallinen CI/CD-putki missä on kaikki testausvaiheet ja uuden koodin julkaisu. Sitten meillä on toinen CI/CD-putki toisessa repositoriossa, jonka ainoat tehtävät ovat käynnistää tämä virallinen CI/CD-putki ja varmistaa oikean Git haaran käyttö.

Testien ja julkaisun suorittavan CI/CD-putken laitettiin taustajärjestelmän repositorioon ja käynnistävän CI/CD-putken käyttöliittymän repostorioon.

Järjestely tehtiin näin, koska jos halutaan laajentaa CI/CD-putken toimimaan MyE.Wayn mobiilisovelluksen kanssa, on yksinkertaisempaa yhdistää mobiilisovelluksen CI/CD-putki taustajärjestelmän repositorioon kuin käyttöliittymän repositorioon, koska käyttöliittymä ja mobiilisovellus molemmat käyttävät samaa taustajärjestelmää.

3.2 Projektin tyylien testaaminen ESLintillä

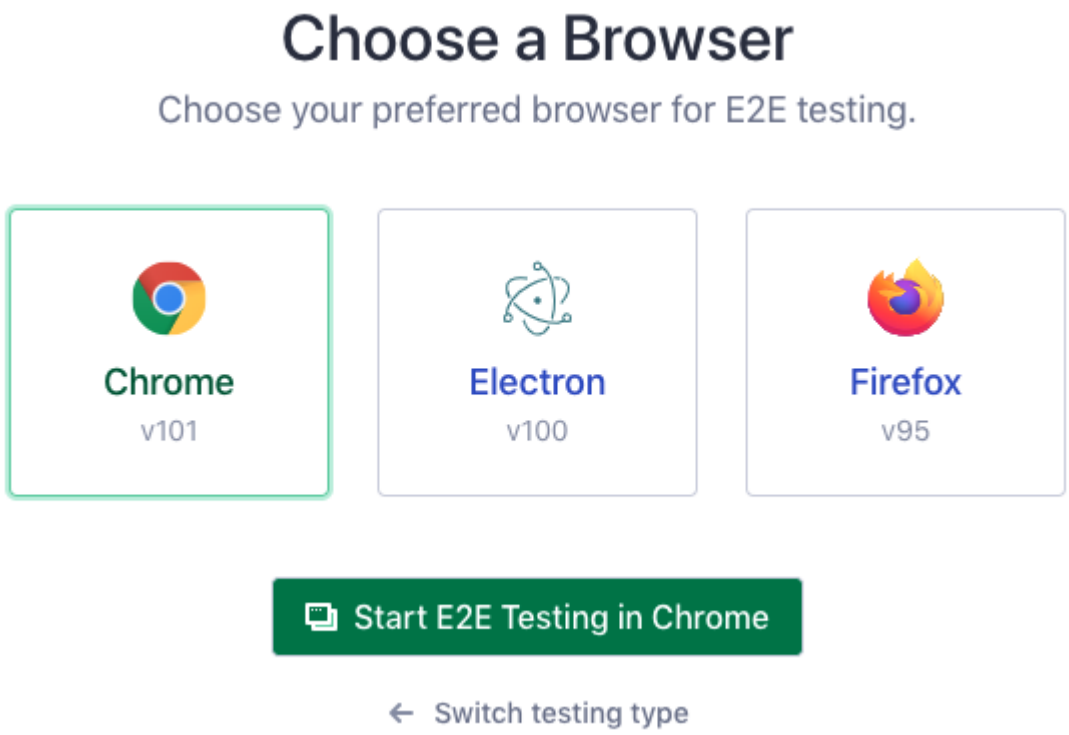
Projektiin ennen CI/CD-putken rakentamisen aloittamista molempiin repositorioista asennettiin tarvittavat paketit testaamista varten, joista ensimmäinen oli ESLint. ESLint paketin saa asennettua käyttäen NPM työkalua ja komentoa ”npm install --save-dev eslint”. Asennuksen jälkeen alustavan ESLint-konfiguraation saa muodostettua käyttäen komentoa ”npx eslint --init” ja vastaamalla annettuihin kysymyksiin. Molemmissa repositorioista käytettiin aluksi vähän eri tyyliasetuksia. Käyttöliittymän repositoriossa valittiin itse perustyylit mitä koodi seuraa ja taustajärjestelmän repositoriossa käytettiin ESLintin valmista tyylipohjaa. Myöhemmin kuitenkin taustajärjestelmän repositoriosta poistettiin ESLintin valmiin tyylipohjan, koska sen käyttö aiheutti vaikeuksia ja se laitettiin käyttämään aivan samoja tyyliasetuksia kuin käyttöliittymä.

Alustuksen jälkeen linttauksen voi suorittaa käyttäen npm-ohjelmakoodia (eng. npm-script). Molempiin repositorioihin luotiin npm-ohjelmakoodi nimeltä ”lint”, jonka voi suorittaa käyttäen komentoa ”npm run lint” ja se testaa kaikki repositorion JavaScript tiedostojen koodin. Komentoriviltä suoritettiin komento molemmissa repositoriossa ja kaikki tyylivirheet korjattiin mitä tuli konsoliin. Osa ESLintin säännöistä kuitenkin laitettiin pois päältä, koska se oli liian tarkka nykyisen koodin tarkistamisessa ja on pari asiaa mitä täytyy koodista muuttaa ennen kaikkien ESLint sääntöjen käyttöä.

3.3 Projektin koodin testaaminen käyttäen e2e-testejä

Toinen työkalu mikä asennettiin testejä varten oli Cypress käyttöliittymän repositorioon e2e-testejä varten, jotka testaa nettisivun toimivuutta. Cypress paketin saa asennettua käyttäen komentoa ”npm install --save-dev cypress”. Asennuksen jälkeen projektiin tehtiin kaksi npm-ohjelmakoodia:

- **cy:run** testaa koodin käyttäen tehtyjä e2e-testejä.
- **cy:open** avaa Cypree laukaisualustan (eng. launchpad), mistä pystyy valitsemaan testaustyypin (komponentti testit tai e2e-testit) ja käynnistämään selaimen missä testit suoritetaan.



Kuva 5. Selaimen valitseminen Cypress laukaisualustalla. (Opening the App | Cypress Documentation, ei pvm.)

CI/CD-putken e2e-testityötä varten tehtiin yhden yksinkertaisen e2e-testin, joka tarkistaa, että selain käynnistyy ja saa yhteyden nettisivulle osoitteessa http://localhost:8001/. Osoite on määritetty cypress.config.js tiedostossa, joka

määritetään projektiin Cypress laukaisualustan käynnistyessä ensimmäisen kerran.

```
describe("launch spec", () => {
    it("passes", () => {
        cy.visit("/")
    })
})
```

Kuva 6. E2e-testi nettisivun käynnistymistä varten.

```
const { defineConfig } = require("cypress")

module.exports = defineConfig({
    e2e: {
        baseUrl: "http://localhost:8001",
    },
})
```

Kuva 7. cypress.config.js tiedosto.

3.4 Gitlab Runnerien asentaminen

Ennen kuin Gitlab Runnereita asennettiin koulun palvelimelle, GitLab Runnerin testausta varten virtuaalikone luotiin käyttäen VirtualBox ohjelmaa ja Ubuntu käyttöjärjestelmää. GitLab Runner- ohjelma asennettiin tähän virtuaalikoneeseen, rekisteröitiin GitLab repositorioon ja sen toimintaa testattiin.

Testauksien onnistumisen jälkeen työtä varten koulun lähiverkkoon asennettiin palvelin Ubuntu-käyttöjärjestelmällä. Asennuksen jälkeen PuTTY nimisellä työkalulla otettiin yhteys tälle Ubuntu-palvelimelle yhdeltä Health Tech Labin koneista ja sinne asennettiin yksi **Projekti** tyypin GitLab Runner. Tämä GitLab Runner yhdistettiin MyE.Wayn taustajärjestelmän repositorioon ja sen suorittajana oli Docker.

Myöhemmin projektissa, kun käyttöliittymän repositorion CI/CD-putkea rakennettiin, sitä varten asennettiin **Ryhmä** tyypin GitLab Runnerin, jonka

suorittajana oli Docker. GitLab Runnerista tehtiin **Ryhmä** tyyppinen, koska sitä voidaan käyttää kaikissa MyE.Way ryhmän alla olevissa repositorioissa ja tämä helpottaa mahdollisien tulevien CI/CD-putkien rakentamista. MyE.Way ryhmään lisättiin viimeiseksi vielä yksi **Ryhmä** tyyppinen GitLab Runner, jolle annettiin suorittajaksi Shell, eli se suorittaa kaikki tehtävät Ubuntu-palvelimen sisällä eikä erillisessä kontissa ja tämä kolmas GitLab Runner on tarkoitettu koodin julkaisuvaiheita varten.

Yksi muutos mitä GitLab Runnereihin tehtiin rekisteröinnin jälkeen oli merkkien (eng. tags) käyttöönotto. Merkkien avulla pystytään määrittämään tietyt GitLab Runnerit tekemään tiettyjä töitä CI/CD-putkessa. Tämän avulla voidaan estää esimerkiksi GitLab Runnerit, joilla on suorittajana Docker tekemästä töitä, jotka on tarkoitettu ainoastaan Shell-suorittajille ja toisin päin. GitLab Runnereille annettiin kaksi merkkiä ”test” ja ”deploy”. Shell-suorittajalla toimivaan GitLab Runneriin annettiin ”deploy” merkki ja muille GitLab Runnereille annettiin ”test” merkki. Lopputulos on, että ”test” merkinnän alla olevat työt suorittaa GitLab Runnerit, joilla on ”test” merkintä ja nämä GitLab Runnerit käyttävät Docker suorittajaa. Sitten ”deploy” merkinnän alla olevat työt suorittaa ainoastaan Shell-suorittajalla toimivat GitLab Runnerit, kun niillä on ”deploy” merkintä. CI/CD-putken töissä saa määritettyä merkit käyttäen ”tags” avainsanaa.

```
job:
  tags:
    - test
```

Kuva 8. Esimerkki merkkien käytöstä CI/CD-putken työssä.

3.5 Taustajärjestelmän repositorion CI/CD-putken rakentaminen

Ensimmäisenä projektissa aloitettiin CI/CD-putken tekeminen, missä on kaikki testausvaiheet ja koodin julkaisu. CI/CD-putken saa luotua GitLabissa tiedostoon nimeltä ”.gitlab-ci.yml”, jonne luodaan kaikki vaiheet ja tehtävät mitä CI/CD-putken täytyy tehdä. Ensin projektissa määritettiin tähän tiedostoon eri

vaiheet mitä CI/CD-putkessa on, näitä vaiheita on neljä ja ne suoritetaan tässä järjestyksessä ylhäältä alas:

1. **Install**
2. **Test**
3. **Package**
4. **Deploy**

3.5.1 **Install** vaihe

Ensimmäinen vaihe **Install** sisältää yhden työn jonka nimi on **front-end-install-job** ja tälle työlle on määritetty ”test” merkki. Tämän työn tehtävänä on ensin kloonata käyttöliittymän repositorion koodi käyttäen Git-komentoa nimeltä ”git clone” ja GitLabin ympäristömuuttujaa nimeltä CI_JOB_TOKEN, joka on ainutlaatuinen merkki minkä GitLab automaattisesti luo ennen kuin CI/CD-putki käynnistyy.

```
- git clone https://gitlab-ci-token:${CI_JOB_TOKEN}@gitlab.instance/group/project.git
```

Kuva 9. Esimerkki GitLab repositorion kloonaamisesta CI/CD-putkessa. (*Gitlab - How to Access Multiple Repositories in CI Build? - Stack Overflow*, 2015)

Toinen asia minkä tämä työ tekee on oikean Git-haaran valitseminen käyttöliittymän koodille, koska kun koodin kloonaa, normaalina Git-haarana on repositorion oletus Git-haara ja jossain tilanteissa meidän täytyy testata toisen Git-haaran koodia. Git-haaran vaihtamisen toteutettiin käyttämällä ”git checkout <branch>” -komentoa, jossa ”<branch>” on se Git-haara, johon halutaan vaihtaa. Itse Git-haaran valitseminen työssä tapahtuu käyttämällä ennalta määritettyjä muuttujia, jotka ovat käytössä jokaisessa CI/CD-putkessa GitLabissa ja joita käytettiin konditionaalilauseissa. Konditionaalilauseet tarkistavat käytettävissä olevat muuttujat ja valitsevat oikean Git-haaran niiden perusteella. Käytetyt muuttujat ovat alla:

- **CI_PIPELINE_SOURCE** kertoo miten CI/CD-putki käynnistettiin.

- **UPSTREAM_CI_PIPELINE_SOURCE** kertoo miten käyttöliittymän CI/CD-putki käynnistettiin ja tämä on käytössä CI/CD-putken käynnistyessä käyttöliittymän CI/CD-putken kautta.

- **UPSTREAM_CI_MERGE_REQUEST_SOURCE_BRANCH_NAME** kertoo Git-haaran nimen jolta ollaan siirtämässä koodia toiselle, kun käyttöliittymän CI/CD-putki käynnistettiin yhdistyspyyntönä (eng. merge request) ja tämä on käytössä CI/CD-putken käynnistyessä käyttöliittymän CI/CD-putken kautta.

- **UPSTREAM_CI_COMMIT_BRANCH** kertoo sitoutus (eng. commit) Git-haaran nimen käyttöliittymän repositoriossa ja tämä on käytössä CI/CD-putken käynnistyessä käyttöliittymän CI/CD-putken kautta.

(Predefined Variables Reference | GitLab, ei pvm.)

```
job:
  script:
    - >
      if [ "$CI_PIPELINE_SOURCE" == "push" ]; then
        echo "Push event"
      elif [ "$CI_PIPELINE_SOURCE" == "pipeline" ]; then
        echo "Downstream pipeline"
      else
        echo "Something else"
      fi
```

Kuva 10. Konditionaalilauseen käyttö CI/CD-putken työssä.

Viimeisenä työssä tallennetaan tämä kloonattu käyttöliittymä esineenä, jotta tulevat vaiheet pystyvät käyttämään sitä töissään.

3.5.2 **Test** vaihe

Toisessa vaiheessa **Test** on kolme eri työtä **front-end-lint-test-job** , **back-end-lint-test-job** ja **e2e-test-job**. Jokaiselle näistä töistä on määritetty ”test” merkki käyttöön. **Front-end-lint-test-job** testaa käyttöliittymän koodin laadun käyttäen ESLint pakettia ja komentoa ”npm run lint”. **Back-end-lint-test-job** tekee saman asian kuin **front-end-lint-test-job**, mutta tekee sen taustajärjestelmän koodille.

Viimeinen työ **e2e-test-job** testaa ohjelman toiminnan e2e-testeillä, jotka kirjoitettiin aikaisemmin käyttöliittymän repositorioon käyttäen Cypress pakettia. Työssä ei käytetä oletus GitLab Runnerin Docker kuvaa vaan käytetään kuvaa ”cypress/base”, joka antaa kaikki tarvittavat riippuvuudet Cypress testien suorittamiseen työlle. Lisäksi käytössä on toinen kontti missä on tietokanta ja sen saa linkattua automaattisesti tähän testauskonttiin käyttäen ”services” avainsanaa.

```
services:
  - mongo:6
```

Kuva 11. Esimerkki MongoDB tietokannan lisäämisestä CI/CD-putken työhön.

Tämä toinen kontti mahdollistaa uusien testien tekemisen, jossa otetaan taustajärjestelmä enemmän huomioon ja voidaan testata asioita joihin tarvitsee tietokanta yhteyden kuten kirjautuminen, profiilin katsominen ja monia muita. Itse työn vaiheet ovat hyvin yksinkertaiset, siinä asennetaan kaikki tarvittavat paketit, konfiguroidaan tarvittavat asiat, käynnistetään käyttöliittymä, taustajärjestelmä ja viimeiseksi itse e2e-testit. Työn valmistumisen jälkeen tallennetaan videot testeistä esineenä, jotta ohjelmistokehittäjät pystyvät virhetilanteissa katsomaan, mikä testeistä ei mennyt läpi ja missä kohtaa testi ei mennyt läpi.

```
artifacts:
  when: always
  paths:
    - cypress/videos/**/*.mp4
    - cypress/screenshots/**/*.png
  expire_in: 1 day
```

Kuva 12. Esimerkki Cypress testien videoiden ja kuvien tallentamiseen esineinä CI/CD-putken työssä.

Test-vaiheen toimimisen jälkeen tein pari uutta e2e-testiä, jotka hyödynsivät tietokantaa ja taustajärjestelmää.

3.5.3 **Package** vaihe

Kolmannessa vaiheessa **Package** on yksi työ nimeltä **build-image**, jossa rakennetaan Docker kuva ja siirretään tämä kuva GitLab instanssin konttirekisteriin. Tässä työssä käytetään merkkiä ”deploy”. Tämä vaihe suoritetaan ainoastaan silloin, kun uuttaa koodia pusketaan yhteen kahdesta GitLabin repositorioista ”dev” haaraan ja se on määritetty käyttäen ”rules” avainsanaa työssä.

```
rules:
  - if: '$CI_PIPELINE_SOURCE == "push" && $CI_COMMIT_BRANCH == "dev"'
```

Kuva 13. Esimerkki sääntöjen käytöstä työssä CI/CD-putkessa.

Työssä ensin kirjaudutaan sisään GitLabin konttirekisteriin Docker kuvan puskemista ja rakentamista varten käyttäen komentoa ”docker login” ja paria valmiiksi luota muuttujaa:

- **CI_REGISTRY** kertoo rekisterin osoitteen.
- **CI_REGISTRY_USER** on käyttäjänimi konttien puskemista varten rekisteriin.
- **CI_REGISTRY_PASSWORD** on salasana konttien puskemista varten rekisteriin.

(Predefined Variables · Variables · Ci · Help · GitLab, ei pvm.)

Kirjautumisen jälkeen uusi Docker kuva luodaan ja pusketaan GitLabin konttirekisteriin käyttöliittymän koodista tai taustajärjestelmän koodista riippuen kummassa repositoriossa päivitys tapahtui. Docker kuvan luonti tapahtuu käyttäen komentoa ”docker build”, puskeminen rekisteriin käyttäen komentoa ”docker push” ja oikean koodin valitseminen tapahtuu käyttäen konditionaalilauseita. Kuvan luomisessa ja puskemisessa käytetään muuttujaa **CI_REGISTRY_IMAGE**, joka kertoo projektin konttirekisterin osoitteen (Predefined Variables · Variables · Ci · Help · GitLab, ei pvm.), konditionaalilauseissa käytetään muuttujaa **CI_PIPELINE_SOURCE**, jonka avulla tarkistetaan kumpaan repositorioon päivitys tuli.

3.5.4 **Deploy** vaihe

Viimeisessä vaiheessa **Deploy** on yksi työ **deploy-to-test-server** ja sillä on käytössä ”deploy” merkki. Työssä ensin kirjaudutaan sisään GitLab konttirekisteriin käyttäen komentoa ”docker login” ja samoja muuttuja mitä viimevaiheessa käytettiin. Sitten käyttäen konditionaalilauseita testipalvelimella pysäytetään vanha kontti käyttäen komentoa ”docker stop” ja poistetaan tämä kontti käyttäen komentoa ”docker rm”. Viimeiseksi luodaan uusi kontti Docker kuvasta ja käynnistetään se. Luonti ja käynnistys onnistuu käyttäen komentoa ”docker run”.

3.6 Käyttöliittymän repositorion CI/CD-putken rakentaminen

Käyttöliittymän repositorion CI/CD-putkessa on kaksi vaihetta, jotka suoritetaan tässä järjestyksessä ylhäältä alas:

1. **Branching**
2. **Trigger**

CI/CD-putkessa on valmiiksi luotu muuttuja **BRANCH_TO_USE**, mille on annettu oletusarvoksi ”dev”. Tämä kertoo CI/CD-putkelle, että jos ensimmäisessä vaiheessa ei ohiteta tuota muuttujaa toisella, käynnistetään toinen CI/CD-putki ”dev” Git-haarassa, joka on oletushaara. Kaikissa töissä on myös ”test” merkki käytössä.

3.6.1 **Branching** vaihe

Ensimmäisessä vaiheessa **Branching** on yksi työ **branch-job**, jonka tehtävänä on valita oikea Git-haara käyttöön taustajärjestelmän repositorion CI/CD-putkelle. Työssä käytetään konditionaalilauseita ja GitLabin valmiiksi luotuja muuttujia katsomaan, miten CI/CD-putki käynnistettiin ja mitä Git-haaraa pitäisi käyttää tai jos käytetään valmiiksi määritettyä **BRANCH_TO_USE** muuttujaa.

Jos toinen Git-haara valitaan, valitun Git-haaran nimi tallennetaan muuttujaan **BRANCH_TO_USE**, tämä muuttuja tallennetaan tiedostoon nimeltä ”build.env” ja tämä tiedosto tallennetaan esineenä, jotta se ohittaa seuraavassa työssä **BRANCH_TO_USE** muuttujan oletusarvon. Tähän on käytetty avainsanoja ”reports” ja ”dotenv”, joiden avulla CI/CD putki tallentaa ”build.env” tiedostosta muuttujat käytettäväksi tulevissa töissä.

```
echo "BRANCH_TO_USE=branch" >> build.env
```

Kuva 14. Esimerkki CI/CD-putkessa työssä tiedostoon kirjoittamisesta.

```
artifacts:
  reports:
    dotenv: build.env
```

Kuva 15. Esimerkki muuttujien tallentamisesta esineinä seuraavia töitä varten tiedostosta ”build.env”.

3.6.2 **Trigger** vaihe

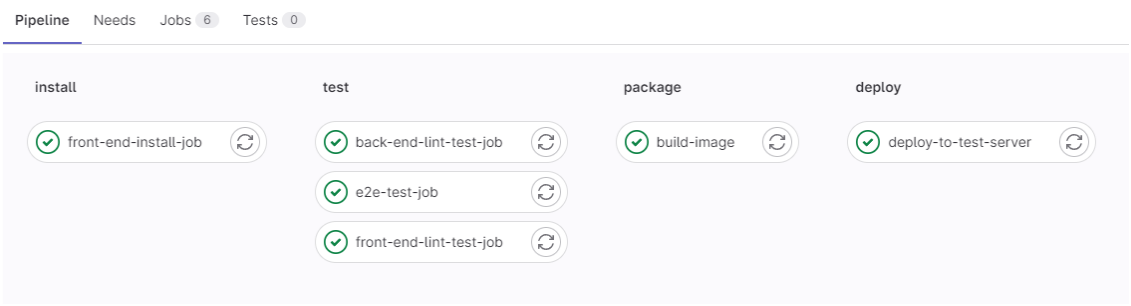
Toisessa vaiheessa **Trigger** on myös yksi työ **trigger-job**, jolla käynnistetään virallinen CI/CD-putki, joka on taustajärjestelmän repositoriossa. Työssä käytetään muuttujaa BRANCH_TO_USE oikean Git-haaran valitsemiseen taustajärjestelmän CI/CD-putkelle ja samalla luodaan käynnistetylle CI/CD-putkelle muuttujia, joita se voi käyttää hyväksi töissään käyttäen ”variables” avainsanaa. Työssä myös käytetään arvoa ”strategy: depend”, minkä avulla jos käynnistetty CI/CD-putki epäonnistuu, myös käynnistävä CI/CD-putki epäonnistuu.

```
trigger-job:
  variables:
    UPSTREAM_CI_PIPELINE_SOURCE: $CI_PIPELINE_SOURCE
  trigger:
    project: group/project
    branch: main
    strategy: depend
```

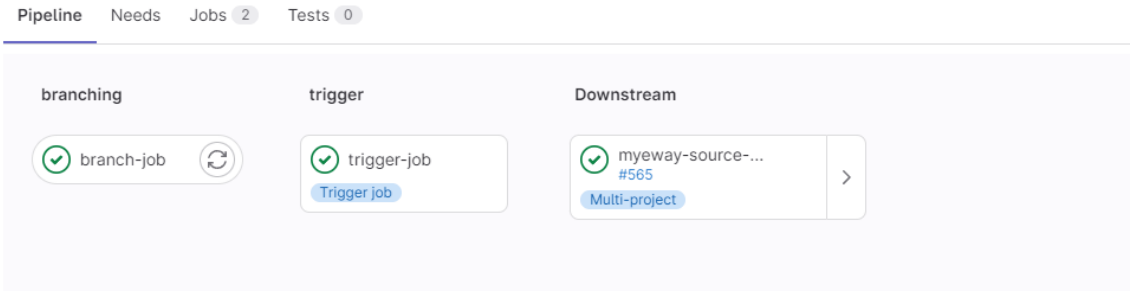
Kuva 16. Esimerkki CI/CD-putken työstä toisen CI/CD-putken käynnistämiseen.

4 Tulokset ja pohdinta

CI/CD-putken tekeminen onnistui odotuksien mukaisesti ilman vaikeuksia. CI/CD-putkien toimintaa ensin testattiin omassa Git-haarassaan käyttäen testipalvelinta, ensimmäisten testauksien ja parin korjauksen jälkeen tämä CI/CD-putki voidaan siirtää omalta Git-haaraltaan oletus Git-haaraan, jossa testausta jatketaan. Viimeiseksi vaihdetaan CI/CD-putki käyttämään oikeaa palvelinta testipalvelimen sijaan, kun ollaan varmoja siitä, että kaikki toimii niin kuin pitäisi. CI/CD-putken käyttöönoton jälkeen ohjelmistokehittäjät MyE.Way projektissa voivat keskittyä enemmän koodin kirjoittamiseen, kun testaaminen ja koodin lähettäminen testipalvelimelle on automatisoitu.



Kuva 17. Valmis virallinen CI/CD-putki taustajärjestelmän repositoriossa.



Kuva 18. Valmis CI/CD-putki toisen käynnistämiseen käyttöliittymän repositoriossa.

Työn aikana kävi ilmi kuinka paljon eri Linux-distribuutiot vaikuttaa automaattisen julkaisun toteuttamiseen CI/CD-putkessa. Alun perin automaattinen julkaisu piti toteuttaa samanlaiselle palvelimelle, mitä projektissa

käytetään tuotantopalvelimena, mutta Dockerin ja järjestelmän ongelmien takia päätetiin siirtää julkaisu väliaikaisesti Ubuntu-palvelimelle. Tulevaisuudessa automaattinen julkaisu todennäköisesti siirretään tuotantopalvelimen tyyppiseen, kun kaikki ongelmat on selvitetty.

Konttien käyttö helpotti e2e-testien rakentamista paljon, kun yritettiin testata asioita, johon tarvittiin tietokanta yhteys. Suunnitelmana oli ensin luoda verkkoon tietokanta, johon otettaisiin testaustyössä yhteys, se tyhjennettäisiin aina ennen testejä ja näin siellä ei olisi vanhoista testeistä tietoa. Konttien kanssa näin ei kuitenkaan tarvinnut tehdä, kun pystyttiin vaan linkkaamaan tämän tietokantakontin testauskonttiin ja se luotiin aina uudestaan ennen testejä, joten ei tarvinnut toteuttaa mitään tyhjennys operaatiota.

GitLabin konttirekisteri helpotti myös paljon julkaisuprosessissa. Alun perin testasin julkaisua käyttäen Dockerhub alustaa, mutta GitLabin oman konttirekisterin käyttö oli kuitenkin helpompaa. Konttirekisterissä oli valmiiksi annettu tarvittavat komennot, muuttujat yhteyden ottoon ja pakettien siirtoon. Tämä helpotti julkaisun tekoa ja konttirekisterin oleminen samalla alustalla missä koodi ylläpidetään on myös iso plussa.

5 Yhteenveto

Opinnäytetyön tarkoituksena oli rakentaa CI/CD-putki MyE.Way projektiin, joka automatisoi koodin testauksen ja koodin siirtämisen testipalvelimelle.

Opinnäytetyön teoriaosuudessa käsiteltiin CI/CD:n perustietoja, CI/CD:n hyviä ja huonoja puolia, DevOpsia, GitLabia ja muita työkaluja, joita käytettiin CI/CD-putken rakentamisessa.

Ensin työtä varten testasin omassa repositoriossa GitLabin CI/CD-putkien rakentamista, GitLab Runnerin asentamista ja yhdistämistä tähän repositorioon käyttäen virtuaalikonetta, jossa oli Ubuntu käyttöjärjestelmä. Testauksien jälkeen GitLab Runnerien asentaminen hoidettiin koulun palvelimelle, joita voidaan käyttää kaikissa MyE.Way ryhmän projekteissa GitLabissa. Asennuksien jälkeen CI/CD-putket rakennettiin MyE.Way projektin GitLab repositorioon ja niiden toimintaa testattiin ”dev-ci-cd-pipeline” nimisessä Git-haarassa molemmissa projektin repositoriossa.

Työn lopputuloksena on CI/CD-putki, joka helpottaa MyE.Way projektin ohjelmistokehittäjien työtä ja yksinkertaistaa koodin julkaisuprosessin testipalvelimelle. Nykyinen julkaisuprosessi rajattiin vain testipalvelimelle, mutta tulevaisuudessa CI/CD-putkeen voi lisätä automaattisen julkaisun tuotantopalvelimelle.

Jatkossa CI/CD-putkea varten voisi myös tehdä lisää e2e-testejä projektiin, jotta MyE.Wayn ominaisuuksia testattaisiin mahdollisimman laajasti ja huomattaisiin mahdolliset virheet, mitä päivityksien tekemisessä saattaa tulla. Projektin molemmista repositorioista otettiin työn alussa pois käytöstä osa ESLintin säännöistä, tulevaisuudessa nämä säännöt laitetaan päälle uudestaan koodin läpikäynnin ja muutosten tekemisen jälkeen. MyE.Way:n mobiilisovellukseen voisi myös tulevaisuudessa luoda CI/CD-putken samalla tavalla, kun MyE.Way:n nettisivuun on luotu tässä työssä.

Lähteet

About npm | npm Docs. (n.d.). NPM Docs. Retrieved April 3, 2023, from https://docs.npmjs.com/about-npm

Alexander S. Gillis. (2021). *What is a Docker Image? Introduction and use cases*. TechTarget. https://www.techtarget.com/searchitoperations/definition/Docker-image

CI/CD pipelines | GitLab. (n.d.). GitLab. Retrieved March 29, 2023, from https://docs.gitlab.com/ee/ci/pipelines/

Find and fix problems in your JavaScript code - ESLint - Pluggable JavaScript Linter. (2023). OpenJS Foundation. https://eslint.org/

GitLab. (n.d.). *What is CI/CD? | GitLab*. GitLab. Retrieved March 23, 2023, from https://about.gitlab.com/topics/ci-cd/

gitlab - How to access multiple repositories in CI build? - Stack Overflow. (2015). Stack Overflow. https://stackoverflow.com/questions/32995578/how-to-access-multiple-repositories-in-ci-build

Karin Kelley. (2023, January 16). *What is GitLab and How to Use It? [2023 Edition] | Simplilearn*. Simplilearn. https://www.simplilearn.com/tutorials/git-tutorial/what-is-gitlab#what_is_gitlab

MyEway - Eerikkilä. (n.d.). EERIKKILÄ SPORT & OUTDOOR RESORT. Retrieved March 24, 2023, from https://eerikkila.fi/en/sport/myeway/

MyE.Way - Eerikkilä. (n.d.). Eerikkilä Sport & Outdoor Resort. Retrieved April 13, 2023, from https://eerikkila.fi/urheilu/myeway/

Opening the App | Cypress Documentation. (n.d.). Cypress. Retrieved March 28, 2023, from https://docs.cypress.io/guides/getting-started/opening-the-app

Predefined variables · Variables · Ci · Help · GitLab. (n.d.). TurkuAMK GitLab Instance. Retrieved March 27, 2023, from https://git.dc.turkuamk.fi/help/ci/variables/predefined_variables.md

Predefined variables reference | *GitLab*. (n.d.). GitLab. Retrieved March 27, 2023, from https://docs.gitlab.com/ee/ci/variables/predefined_variables.html

Tech and Beyond With Moss. (2022, April 4). *(12) GitLab CI/CD* | *GitLab Runner Introduction* | *2022 - YouTube*. Tech and Beyond With Moss. https://www.youtube.com/watch?v=-CyVpfDQAG0

TechWorld with Nana. (2022, June 9). *GitLab CI CD Tutorial for Beginners [Crash Course] - YouTube*. TechWorld with Nana. https://www.youtube.com/watch?v=qP8kir2GUgo

TechWorld with Nana. (2023, February 15). *Docker Crash Course for Absolute Beginners - YouTube*. TechWorld with Nana. https://www.youtube.com/watch?v=pg19Z8LL06w

Tietojärjestelmän avulla kohti urheilun huippua - Eerikkilä. (n.d.). EERIKKILÄ SPORT & OUTDOOR RESORT. Retrieved March 24, 2023, from https://eerikkila.fi/tietojarjestelman-avulla-kohti-urheilun-huippua/

What is CI/CD? (2022). RedHat. https://www.redhat.com/en/topics/devops/what-is-ci-cd

What is DevOps? | *GitLab*. (n.d.). GitLab. Retrieved March 24, 2023, from https://about.gitlab.com/topics/devops/

What is npm. (n.d.). W3Schools. Retrieved April 3, 2023, from https://www.w3schools.com/whatis/whatis_npm.asp

Why Cypress? | *Cypress Documentation*. (2023). Cypress.Io. https://docs.cypress.io/guides/overview/why-cypress