Jatkuvan integraation ja julkaisun prosessin luominen osaksi verkkosivujen kehittämisprosessia - case Web-veistämö

Miikka Niemeläinen

2020 Laurea

Laurea-ammattikorkeakoulu

Jatkuvan integraation ja julkaisun prosessin luominen osaksi verkkosivujen kehittämisprosessia - case Web-veistämö

Miikka Niemeläinen

Tietojenkäsittely

Opinnäytetyö

Lokakuu, 2020

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittely
Tradenomi (AMK)

Miikka Niemeläinen

Jatkuvan integraation ja julkaisun prosessin luominen osaksi verkkosivujen kehittämisprosessia - case Web-veistämö

Vuosi 2020 Sivumäärä 36

Tämän opinnäytetyön tarkoituksena oli kehittää toimiva jatkuvan integraation ja julkaisun prosessi toimeksiantona Web-veistämö Oy:lle. Tavoitteena oli luoda prosessi, joka helpottaa lähdekoodin laadunvarmistusta, toimitusta ja integroimista. Jatkuva integraation ja julkaisu on viime vuosina yleistynyt metodologia, jonka tarkoitus on automatisoida ohjelmistojen testaus- ja toimitusprosesseja. Kehittämismenetelminä käytettiin Scrumia tehtävien suunnitteluun sprinteittäin ja Kanban-taulua tehtävien etenemisen tarkkailuun ja erittelyyn.

Opinnäytetyön tuloksena oli jatkuvan integraation ja julkaisun prosessi, joka pystyy automaattisesti käynnistämään olemassa olevan verkkosivuston etäpalvelimelle lataamalla tarvittavat tiedostot koodikannasta. Prosessia voi vielä parantaa tulevaisuudessa lisäämällä tuen varmuuskopioille ja optimoimalla koodia entisestään.

Asiasanat: continuous integration, continuous delivery, docker, bash

Laurea University of Applied Sciences **Abstract**
Business Information Technology
Bachelor of Business Administration

Miikka Niemeläinen

Creation of Continuous Integration and Continuous Delivery Process as a Part of Web Development Process – Case Web-veistämö Inc.

Year 2020 Pages 36

The purpose of this bachelor's thesis was to create a working continuous integration and delivery process as an assignment for Web-veistämö Inc. The goal was to create a process that helps with source code quality assurance, delivery and integration. Continuous integration and delivery is an increasingly popular methodology which purpose is to automate software testing and deployment processes. As development methods Scrum was used for planning the tasks in sprint-to-sprint basis and Kanban to keep track of the tasks' progress and to separate the project into smaller tasks.

The product of this thesis was a continuous integration and delivery process that could automatically start an existing website by downloading the required files to a remote server from the source code management system. The process can be improved by implementing a backup support and optimising the code further.

Keywords: continuous integration, continuous delivery, docker, bash

Sisällys

1 Johdanto

Työn tavoitteena oli kehittää CI/CD-prosessi, joka vähentää manuaalisen työn määrää, parantaa toimitusvarmuutta ja jota voitaisiin skaalata/soveltaa tulevaisuudessa. Opinnäytetyö toteutettiin toimeksiantona Web-veistämö Oy:lle. Web-veistämö on 2010 perustettu verkkopalveluiden kehittämiseen erikoistunut yritys, joka sijaitsee Helsingin Vallilassa. Yritys halusi implementoida jatkuvan integraation ja jatkuvan toimituksen toimintatavat osaksi verkkosivustojen kehitysprosessiin.

Projekti osoittautui mielenkiintoiseksi, koska halusin selvittää, mitä konkreettista hyötyä yritys voisi toimivasta jatkuvan integraation ja julkaisun prosessista hyötyä. Opiskeluni aikana ohjelmoinnin lomassa törmäsin hyvin usein blogiteksteihin tai artikkeleihin, joissa kehuttiin DevOpsin ja jatkuvan integraation ja julkaisun hyötyjä ja sitä, miten ne vähentävät kehittäjien ja IT-osastoiden taakkaa. Koulussa DevOpsista ei ollut varsinaisia kursseja enkä perehtynyt niihin sen tarkemmin itsekään, sillä en välttämättä olisi omissa projekteissani saanut siitä läheskään samaa hyötyä kuin yritykset, joilla saattaa olla samaan aikaan useita projekteja työnalla. Kun ohjaajani otti jatkuvan integraation ja julkaisun puheeksi harjoittelun aikana, tuumin, että aiheen tutkiminen voisi soveltua hyvin opinnäytetyöksi ja edistää omaa osaamistani aihealueesta.

Tämän opinnäytetyön toisessa luvussa käydään läpi projektin lähtökohdat eli miksi projekti aloitettiin ja miten yritys tästä tulisi hyötymään. Kolmannessa luvussa selitetään, mitä jatkuva integraatio ja jatkuva julkaisu terminä tarkoittaa, ja mitä kaikkea se pitää sisällään. Neljännessä luvussa kerrotaan, mitä teknologioita ja työkaluja projektissa hyödynnettiin ja miksi niitä käytettiin. Viidennessä luvussa käydään läpi, mitä kehittämismenetelmiä käytettiin ja miten projektia hallinnoitiin. Kuudennessa luvussa kerrotaan, miten projekti eteni, miten se suunniteltiin, miten kehittämistä lähestyttiin ja miten prosessia testattiin. Viimeisessä luvussa vedetään yhteen projektin tulokset ja käyn läpi omaa oppimistani ja mietteitä projektista.

2 Työn lähtökohdat

Web-veistämöllä ei ole ennestään ole ollut CI/CD-prosessia verkkopalveluiden kehityksen apuna. Tällä hetkellä projektien muutokset täytyy tehdä manuaalisesti sivustoille lataamalla päivitetyt tiedostot palvelimelle. Nykyään web-kehitys on monesta eri näkökulmasta monimutkaistunut niin paljon, että pienienkin muutosten tekeminen vaatii enemmän resursseja ja on kalliimpaa toteuttaa. Samalla toimitusvarmuus ja laatu kärsii, sillä monimutkaisen

lähdekoodin muuttaminen altistaa kokonaisuuden helpommin virheille. Kun sovelluksen kokonaisuus kasvaa ja sisältää enemmän toiminnallisuutta, osapuolia ja komponentteja, on bugien aiheuttajien selvittäminen vaikeampaa ja aikaa vievää.

Yritys ei ole aiemmin toteuttanut CI/CD-prosessia, koska teknologiat, talon sisäinen osaaminen ja resurssit eivät ole olleet riittäviä. Tämä osoittautua hyväksi tilaisuudeksi tutkia, miten CI/CD-prosesseja voitaisiin hyödyntää verkkosivustojen kehityksessä, miten ne voisivat auttaa kehittäjiä poistamalla toistuvia, helposti automatisoitavia vaiheita ja miten niiden avulla laatua voitaisiin varmentaa entistä paremmin.

2.1 Tutkimuskysymykset

Ennen projektin aloittamista pidimme ohjaajan kanssa pienimuotoisen palaverin, jossa keskustelimme prosessista. Palaverin tarkoitus oli luoda konsepti prosessin toiminnasta ja selittää, mitä prosessilta odotetaan. Keskeisiksi kysymyksiksi nousivat:

- Miten jatkuvan integraation ja julkaisun prosessi kannattaa toteuttaa?
- Miten prosessin implementoiminen vaikuttaa nykyisiin työkäytäntöihin?
- Mitä hyötyä prosessista on yritykselle?

Nämä kysymykset mielessä pitäen keskustelimme keskenämme, miten tätä projektia kannattaisi lähestyä. Koska minulla ei ollut aikaisempaa kokemusta DevOps puolelta, halusin selvittää, mitkä ovat alan parhaat käytännöt.

2.2 Aihealueen rajaus

Tämä projektin tarkoitus oli luoda alustava kartoitus ja pienin toimiva tuote (MVP, Minimum Viable Product) siitä, miten CI/CD-prosessia voitaisiin hyödyntää osana verkkosivujen kehitystä. DevOps on käsitteenä hyvin laaja ja sisältää CI/CD-prosessien lisäksi paljon muita vaiheita liittyen kehittämiseen, ylläpitoon ja laadunvarmistukseen. Tästä syystä tässä projektissa keskityimme ainoastaan CI/CD-prosessin kehittämiseen, jotta yritys saisi jonkinlaisen viitekehyksen siitä, miten prosessi olisi hyödyllinen heille. Lisäksi koko DevOps elinkaaren suunnittelu ja toteutus olisi ollut projekti liian suuri toteutuakseen työharjoittelun aikana ja olisi vaatinut huomattavasti enemmän osaamista ja tietämystä aihealueesta.

2.3 Keskeiset käsitteet

DevOps on joukko tapoja, joiden avulla automatisoidaan ja integroidaan prosesseja ohjelmistokehityksen ja IT-toimintojen välillä, jotta ohjelmistojen rakentaminen, testaaminen ja julkaiseminen olisi nopeampaa ja luotettavampaa (Atlassian, n.d.).

Continuous Integration (CI) eli jatkuva integraatio tarkoittaa tuotetun koodin testaamista ja sen integroimista tai sisällyttämistä nykyiseen koodikantaan.

Continuous Delivery (CD) eli jatkuva julkaisu on automatisoitu tapa toimittaa koodia haluttuun ympäristöön.

Repository (repo) on versiohallinnassa käytetty eräänlainen kansio tai hakemisto, joka sisältää projektin eri tiedostot ja kaikkien tiedostojen muutoshistoriat.

CI/CD on CI:n ja CD:n muodostama prosessi. Kun CI:n jatkeeksi liitetään CD, saadaan CI/CD-prosessi tai -putki, jonka avulla koodin testaus ja julkaisu automatisoidaan yhdeksi prosessiksi.

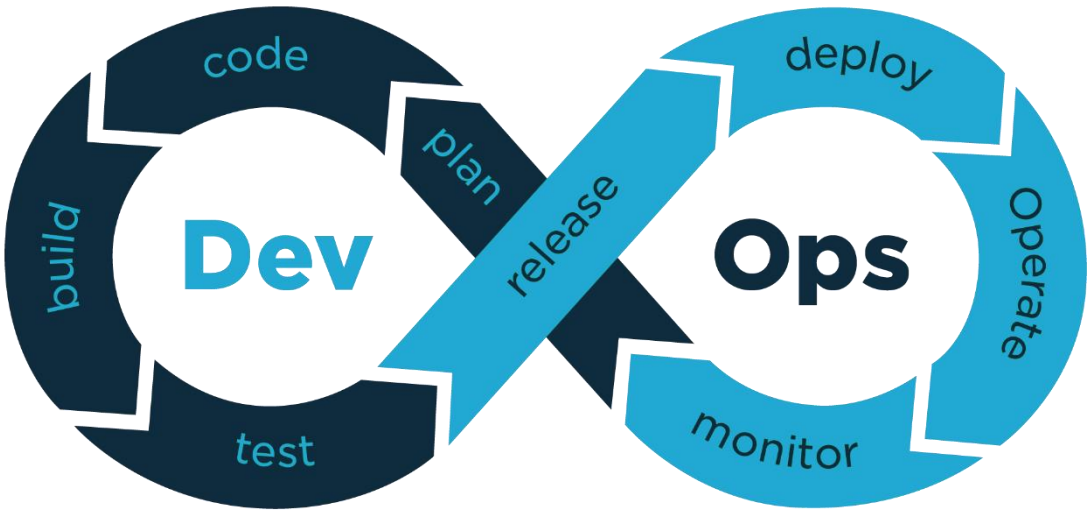
Git on projektien versiohallintaan tarkoitettu ohjelma, jota voidaan käyttää kaiken kokoisissa projekteissa (Git, n. d.)

Commit tarkoittaa Git-versiohallintaohjelmassa tapahtuvaa tallennusoperaatiota, jonka avulla haluttujen tiedostojen muutokset tallennetaan repoon. Muita projektissa mainittuja operaatioita ovat mm. **Pull**, jonka avulla repo päivitetään alkuperäisestä sijainnista ja **Push**, jonka avulla paikallisen repon muutokset tallennetaan alkuperäiseen repoon.

3 DevOps

Viime vuosina DevOps-käytännöistä on tullut yhä tärkeämpi osa yritysten toimintaa. DevOps-liike alkoi 2007 ja 2008 välillä, kun IT- ja kehittäjäyhteisöt ilmaisivat tyytymättömyytensä perinteiseen ohjelmistokehitykseen. Kehittäjillä ja IT-toimintojen työtekijöillä oli eri tavoitteet ja mittarit heidän suoriutumisensa seuraamiseen, jonka perusteella heidän tehokkuuttaan arvioitiin. Kommunikaation puute näiden tiimien välillä johti pitkiin työpäiviin, huonoihin julkaisuihin ja tyytymättömiin asiakkaisiin. Yhteisöt arvelivat, että olemassa täytyy olla parempi ratkaisu ja niinpä yhteisöt lyöttäytyivät yhteen ja alkoivat kehittää parempaa tapaa. (Atlassian, n.d.)

Devops on terminä hyvin laaja ja sitä voidaan kuvailla monella eri tavalla. Sharma (2017, 3) kuvailee DevOpsia liikkeenä, jonka perimmäinen tarkoitus on tuoda kehittäjät (Dev) ja IT-toiminnat (Ops) lähemmäksi toisiaan ja luoda enemmän kommunikaatiota, yhteistyötä ja luottamusta. DevOpsilla pyritään luomaan luotettavampia ratkaisuja ohjelmistojen kehittämiseen ja julkaisuun tuomalla nämä kaksi osa-aluetta lähemmäksi toisiaan.



Kuvio 1: DevOps kuvailtuna (medium.com)

CI/CD terminä tarkoittaa koko prosessia, jossa koodia integroidaan, testataan, rakennetaan ja julkaistaan usein ja kokonaan tai melkein kokonaan automaattisesti. CI/CD hoitaa koodin testaamisen ja julkaisun, jolloin näihin toimintoihin ei tarvitse kuluttaa ylimääräisiä resursseja.

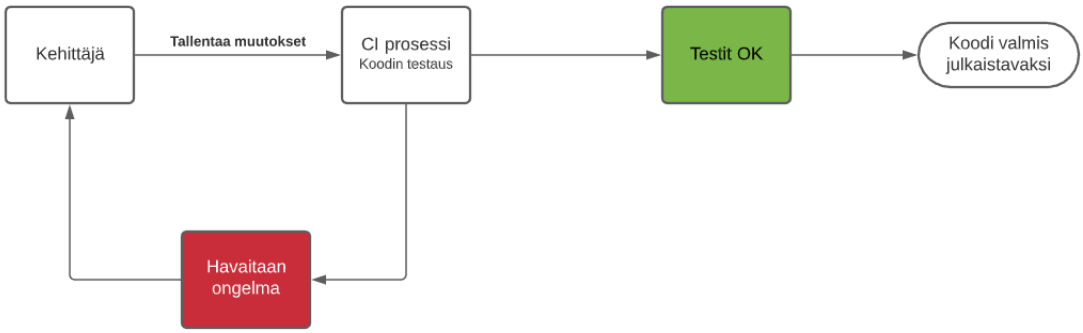
3.1 Jatkuva integraatio (CI)

CI:llä tarkoitetaan kehitysfilosofiaa ja -käytäntöjä, joissa kehitystiimi implementoi ja julkaisee muutokset lähdekoodiin pienissä erissä ja usein. CI:n tavoite on luoda automatisoitu tapa rakentaa, paketoida ja testata sovelluksia. Kun integraatioprosessissa on tietynlainen yhtenäisyys, tiimit todennäköisemmin julkaisevat muutoksensa versiohallintaan useammin, joka johtaa parempaan yhteistyöhön ja ohjelmiston laatuun. (Sacolick, 2020)

CI:n tarkoitus on siis varmistaa, että tallennettu koodi on toimivaa eikä aiheuta konflikteja olemassa olevan koodin kanssa. CI auttaa myös ratkaisemaan konflikteja tallennusvaiheessa, jos esimerkiksi kaksi kehittäjää on muokannut samaa koodia. Jatkuvan integraation etu on myös se, että automatisoitu prosessi pystyy paikantamaan virheitä, jolloin ne voidaan korjata nopeasti (Codeship, n.d).

CI prosessi pyrkii testaamaan koodia automatisoiduilla testausskripteillä tai -ohjelmilla. Sovelluksesta riippuen prosessi saattaa rakentaa ohjelmiston lähdekoodista toimivaan tilaan, testata ohjelmistoa ja tarkkailla, miten ohjelma reagoi testausohjelman antamiin syötteisiin. Jos sovellusta ei voida rakentaa ja testata automaattisesti, voidaan koodia lukea ohjelmalla, joka tarkistaa, että kirjoitettu koodi vastaa esimerkiksi yrityksen määrittelemää koodistandardia. Kun testit on suoritettu, prosessi antaa joko hyväksytyn tai hylätyn päätöksen. Jos koodi ei läpäise testejä, prosessi ilmoittaa siitä kehittäjälle, joka voi sitten korjata ohjelmiston

puutteet. Jos koodi menee läpi, prosessi ilmoittaa kehittäjälle tai antaa komennon julkaisuprosessille. (Kuvio 2)



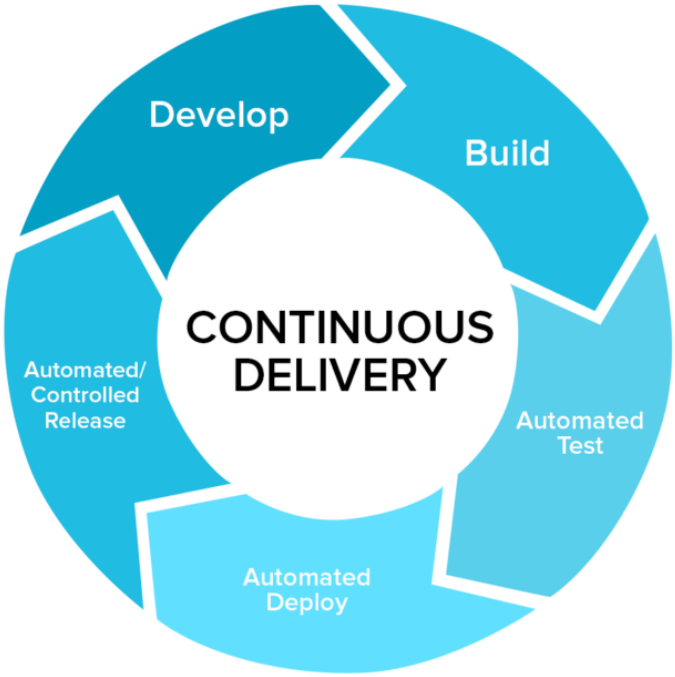
Kuvio 2: Kaavio jatkuvan integraation prosessista

3.2 Jatkuva julkaisu (CD)

CD automatisoi sovelluksen julkaisun määriteltyyn ympäristöön. Monet tiimit työskentelevät useammissa ympäristöissä kuin vain tuotannossa, kuten kehitys- tai testausympäristöissä. CD varmistaa, että näihin ympäristöihin on mahdollista toimittaa ohjelmistoa automaattisesti. (Sacolick, 2020)

Jatkuvat julkaisu mahdollistaa uusien muutosten ja ominaisuuksien viemisen suoraan haluttuun ympäristöön. Jos esimerkiksi kehittäjät ovat varmistuneet uuden ominaisuuden toiminnasta, voidaan koodi tallentaa tuotantoympäristön koodikantaan, josta uusi ominaisuus asennetaan ja se tulee näkyviin sivustolle. Jatkuva julkaisu hoitaa uuden ominaisuuden asennuksen, eikä kehittäjien tarvitse tuhlata aikaa ympäristön koodin päivittämiseen ja mahdollisiin uudelleenkäynnistyksiin.

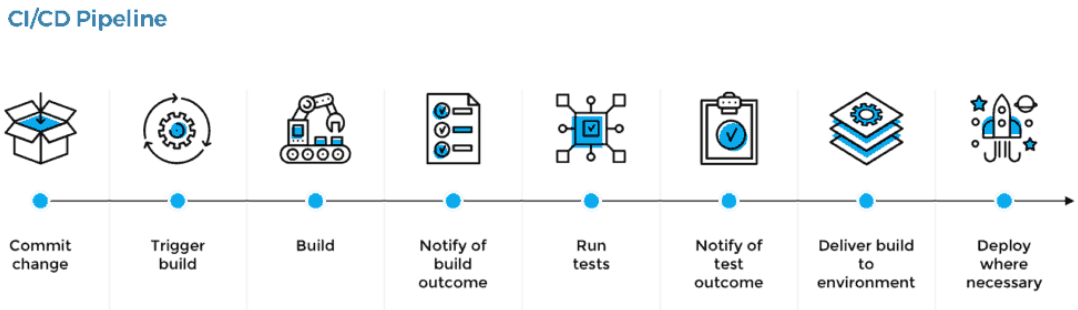
Jatkuvan julkaisun suurin hyöty on uusien päivitysten ja ominaisuuksien vieminen asiakkaalle mahdollisimman nopeasti. CD on käytännössä jatke CI:lle, sen tarkoitus on julkaista CI:n hyväksymä versio määriteltyyn ympäristöön. Kuviosta 3 selviää, miten CD liitetään CI/CD prosessiin: Kehittäjät kehittävät koodia, koodi rakennetaan ja testataan, jonka jälkeen se siirretään CD:lle. Kun koodi on julkaistu, prosessi alkaa alusta uudella iteraatiolla.



Kuvio 3: Jatkuvan julkaisun eri vaiheet (altexsoft.com)

3.3 Jatkuva integraation ja julkaisu (CI/CD)

CICD terminä tarkoittaa koko prosessia, jossa koodia integroidaan, testataan, rakennetaan ja julkaistaan usein ja kokonaan tai melkein kokonaan automaattisesti (kuvio 4). CICD hoitaa koodin tes-taamisen ja julkaisun, jolloin näihin toimintoihin ei tarvitse kuluttaa ylimääräisiä resursseja.



Kuvio 4: CI/CD prosessi kuvattuna (plutora.com)

3.4 DevOps hyödyt

DevOpsista on tutkitusti hyötyä yritysten IT-toiminnoissa. Nicole Forsgrenin pitämän esityksen vuosina 2014 ja 2015 yritykset, jotka hyödynsivät DevOpsia ja CI/CD-prosessia, julkaisivat koodia 30 kertaa enemmän kuin muut yritykset ja niiden läpimenoaika oli jopa 200 kertaa nopeampi (DevOps Enterprise Summit, 2015). Tämä itsessään parantaa yrityksen ketteryyttä ja parantaa työntekijöiden tyytyväisyyttä ja hyvinvointia: kun koodia voidaan julkaista luotettavasti ja sen toimivuudesta voidaan olla varmoja, työntekijöiden ei tarvitse stressata koodin julkaisua ja siitä aiheutuvia ongelmia. IT-alalla työuupumus on ollut huolestuttava ilmiö viime vuosin. Blindin vuonna 2018 tekemän kyselyn mukaan noin 60% IT-alalla työskentelevistä kertoi kärsineensä burnoutista eli työuupumuksesta. CI/CD:n ja DevOpsin avulla työntekijöiden stressiä voidaan vähentää poistamalla työntekijän mahdollisesti aiheuttaman virheen todennäköisyyttä automatisoimalla toistuvia toimenpiteitä, kuten koodin julkaisemista ja testaamista. Olen itse töissä ja työharjoittelun aikana julkaissut muutamia uusia verkkosivuja ja huomannut, miten paljon toistuvia ja helposti automatisoitavia vaiheita siihen sisältyy ja yhdenkin unohtaminen saattaa johtaa julkaisun epäonnistumiseen tai julkaisuun, joka epäonnistuu jollain osa-alueella.

IT-toimintojen tehokkuuteen vaikuttivat myös prosessien MTTR:n (Mean Time To Recovery eli keskimääräinen aika epäonnistuneen prosessin palautumiseen) ja muutosten onnistumisprosentin paraneminen. DevOpsia hyödyntävien yritysten MTTR oli 48 kertaa nopeampi kuin muilla yrityksillä ja muutosten onnistumisprosentti oli 3 kertaa korkeampi (DevOps Enterprise Summit, 2015). Tämä todistaa, miten paljon DevOpsin hyödyntäminen parantaa yritysten mahdollisuuksia havaita mahdolliset ongelmat prosesseissa ja parantaa toimitus- ja julkaisuvarmuutta. Kun prosesseja voidaan monitoroida monipuolisesti, mahdolliset ongelmat havaitaan nopeasti ja ne voidaan korjata. Myös automatisoitujen prosessien hyödyntäminen parantaa onnistumisprosenttia, kun julkaisu- ja testausprosessi on standardoitu eikä vaadi työntekijän manuaalista panosta samalla poistaen työntekijän aiheuttamien virheiden ilmaantumisen mahdollisuuden.

DevOps ei ole ainoastaan hyödyllinen toimintatapa sovelluskehityksen näkökulmasta vaan sillä on myös todistetusti myönteinen vaikutus koko organisaation toimintaan ja kannattavuuteen. Forsgrenin mukaan yritykset, jotka hyödyntävät DevOpsin käytäntöjä ylittivät kannattavuuden ja tuottavuuden tavoitteet kaksi kertaa todennäköisemmin ja yrityksen markkina-arvo kasvoi 50% enemmän kolmen vuoden aikana kuin muilla yrityksillä (DevOps Enterprise Summit, 2015).

DevOpsin hyödyt eivät rajaudu ainoastaan IT-toimintoihin vaan niillä on vaikutusta koko organisaation tasolla. Kun asiakkaat pysyvät tyytyväisinä ja resursseja voidaan ohjata muualle julkaisutoimenpiteistä, vaikuttaa tämä positiivisesti yrityksen talouteen. Työntekijät voivat

käyttää aikaansa enemmän olennaisiin työtehtäviin, kun koodin julkaisu ja testaus hoidetaan automaattisesti. Tämä myös vähentää aiemmin mainittua työntekijöiden kuormittumista ja työuupumusta, kun työntekijöillä on vähemmän ylläpitotehtäviä.

4 Teknologiat

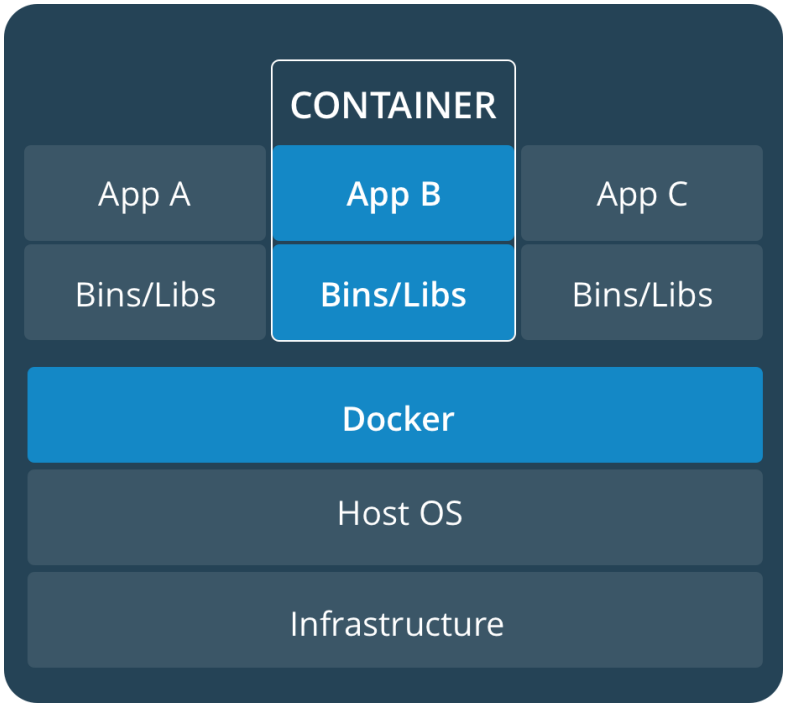
CI/CD-prosessiin käytettävät teknologiat riippuvat paljon siitä, mitä prosessilla halutaan tehdä. Jotkin ohjelmointikielet ja -kehykset sisältävät CI/CD-työkaluja, jolloin prosessin luominen niille on helpompaa. Projektin alussa tutkimme, löytyykö Wordpress-pohjaisilla sovelluksille virallisia CI/CD-työkaluja, mutta nopealla tutkimisella emme löytäneet juuri Wordpressille kehitettyjä testaustyökaluja. Päätimme siis rakentaa CI/CD-prosessin alusta asti itse hyödyntäen GitLabin tarjoamia työkaluja.

Seuraavissa aliluvuissa on kerrottu, miksi nämä kyseiset teknologiat valittiin ja miten ne ovat hyödyllisiä CI/CD prosessissa. Osa teknologioista on käytössä Web-veistämön kehitysprosessissa ja ne on todettu hyödyllisiksi, joten päätimme käyttää niitä myös tässä projektissa.

4.1 Docker

Docker on avoimen lähdekoodin alusta sovellusten kehittämiseen, toimittamiseen ja ajamiseen. Dockerin avulla sovelluksia voidaan paketoida ja ajaa löyhästi eristetyissä ympäristöissä (engl. container, myöhemmin kontti). Kontit ovat kevyitä, koska ne eivät vaadi erillistä hypervisor-ohjelmistoa, vaan ne toimivat erillisinä prosesseina, eivätkä tällöin vaadi erillistä käyttöjärjestelmää pohjalle (Kuvio 5). Kontit soveltuvat myös hyvin käytettäviksi CI/CD-prosesseissa, koska kontti voidaan paketoida Docker levykuvaksi, siirtää toiseen ympäristöön ja luoda Docker levykuvasta kontti uuteen ympäristöön. Konttien avulla samaa ohjelmistoa voidaan ajaa missä tahansa ympäristössä, oli se sitten kehittäjän tietokone, pilvipalvelin tai palvelinkeskus ja odottaa ohjelmiston käyttäytyvän samalla tavalla. (Docker Documentation, n.d.)

Dockerin avulla Wordpress-pohjaisten verkkosivustojen luominen on helppoa, koska sivuston saa konfiguroitua ja käynnistettyä yhdellä komennolla. Konfiguraatioon voidaan käyttää docker-compose.yml tiedostoa, jonne annetaan tarvittavat parametrit, kuten tietokannan nimi, salasana ja osoite, verkkosivuston URL-osoite ja tarvittaessa halutut lisäosat, jotka asennetaan ensimmäisen käynnistyksen yhteydessä.



Kuvio 5: Kaavio Dockerin toiminnasta (Docker Documentation, n.d.)

4.2 Bash

Bash on GNU-pohjaisissa käyttöjärjestelmissä käytetty komentokehote, joka toimii seuraajana SH shell komentokehotteelle. Bash on oletuskomentokehote kaikissa GNU-pohjaisissa käyttöjärjestelmissä ja se on pitkälti yhteensopiva SH shell komentokehotteen kanssa, mutta tarjoaa parannuksia interaktiivisuuteen ja ohjelmointiin. (GNU, 2019)

Koska palvelimet, joilla sivustot sijaitsevat ovat Linux-pohjaisia, käytettiin prosessin ohjelmointiin Bashia. Bash sopii ohjelmointiin erinomaisesti, sillä se tarjoaa laajan valikoiman erilaisia työkaluja esimerkiksi tiedostojen siirtämiseen, muokkaamiseen, luomiseen, paketoimiseen ja purkamiseen, joista on erityisesti hyötyä, kun sivustojen tiedostot ladataan GitLabista ja asennetaan palvelimelle. Lisäksi koska Bash on oletuskomentokehote, voimme olettaa, että Bashilla tehdyt ohjelmat toimivat kaikissa Web-veistämön palvelinympäristöissä, joka tekee prosessin implementoinnista helpompaa.

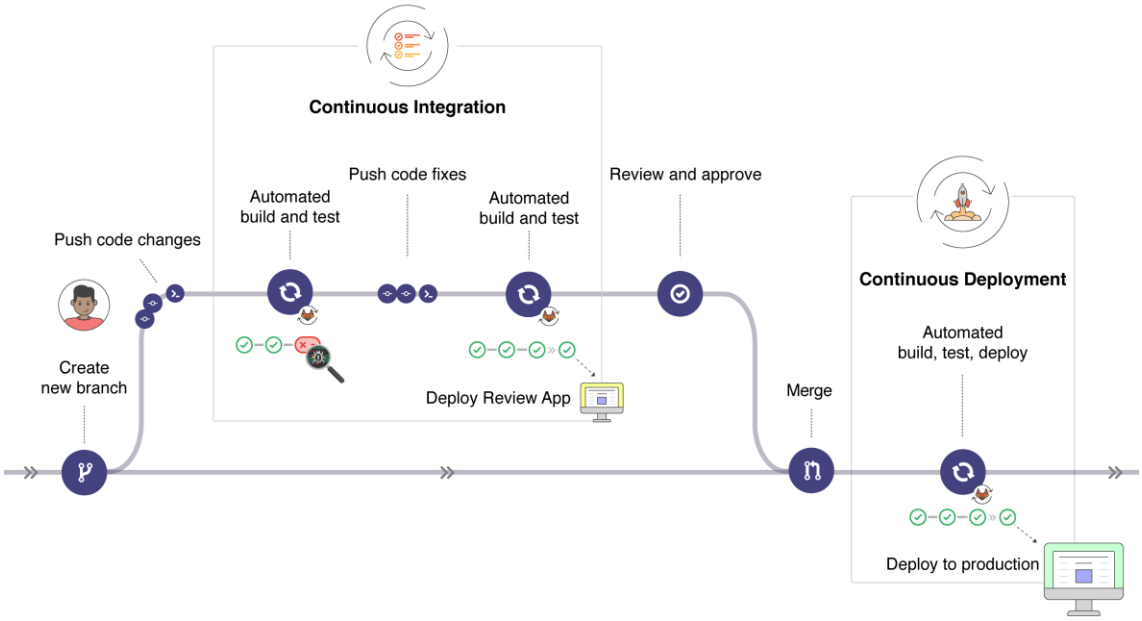
4.3 GitLab

GitLab on avoimen lähdekoodin ohjelmistokokonaisuus, joka tarjoaa työkaluja koko DevOps elinkaarelle. GitLabin tarjoamiin ominaisuuksiin kuuluu lähdekoodin hallinta (Source Code Management eli SCM), versiohallinta, erilaiset DevOps työkalut, monitorointi, tietoturvatyökalut ja paljon muuta.

Tähän mennessä Web-veistämöllä GitLabia on käytetty pääasiassa versiohallintaan ja lähdekoodin säilömiseen, mutta tässä projektissa GitLabin tarjoamista DevOps työkaluista oli todella paljon apua. Työkalut on dokumentoitu perusteellisesti ja dokumentaatio sisältää esimerkkejä niiden käytöstä, joka helpottaa niiden käytön hahmottamista.

4.3.1 Gitlab CI/CD

Gitlab CI/CD on Gitlabiin sisäänrakennettu työkalu ohjelmistokehityksen toteuttamiseen jatkuvien käytäntöjen avulla (Gitlab, n.d.). GitLab CI/CD tekee jatkuvan integraation ja julkaisun prosessien luomisesta helpompaa tarjoamalla hyvin dokumentoidun ja helppokäyttöisen työkalun. Kuvio 6 avaa työkalun toimintaa tarkemmin.



Kuvio 6: Kaavio GitLab CI/CD:n toiminnasta (GitLab, n.d.)

4.4 Repository

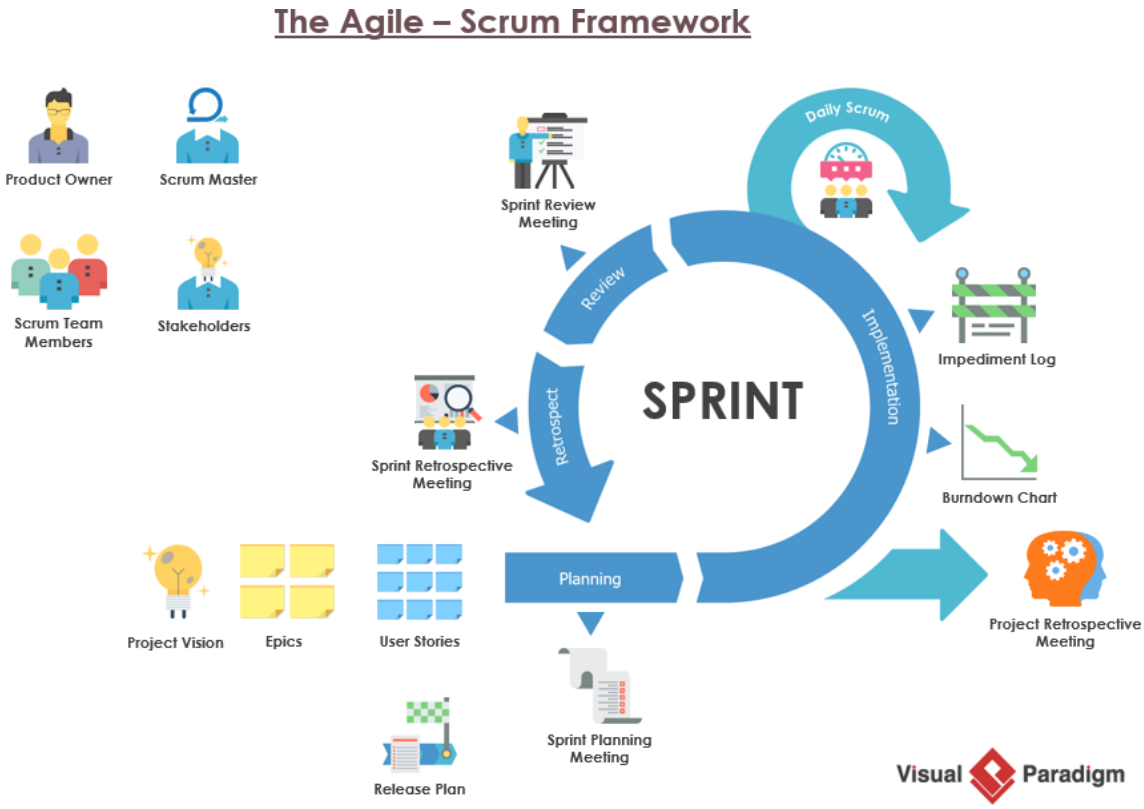
Repository (myöhemmin repo) on eräänlainen kansio tai hakemisto, jota käytetään lähdekoodin säilyttämiseen ja versiohallintaan. Repo sisältää kaikki projektin tiedostot ja tiedostojen muutoshistorian. (Github, n.d) Koska repo sisältää koko projektin muutoshistorian alusta lähtien, on helppoa palata projektissa takaisin päin, jos uusimman päivityksen mukana ilmenee kriittisiä virheitä tai vanhempaan versioon halutaan palata.

Repo on siis käytännössä projektin tiedostojen hallintaan tarkoitettu työkalu, jonka avulla projektin eri vaiheet voidaan tallentaa repoon commiteilla. Jokainen commit tallentaa repon eri tiedostojen sisällön sellaisena, kuin ne olivat commitin hetkellä. Tällöin repossa olevia tiedostoja voidaan palauttaa aiempaan revisioon, jos myöhemmin ilmenee ongelmia ja halutaan

palata toimivaan vaiheeseen. Commit historiasta myös näkee, mitä muutoksia on tehty, kuka ne on tehnyt ja milloin ne on tehty.

5 Kehittämismenetelmät

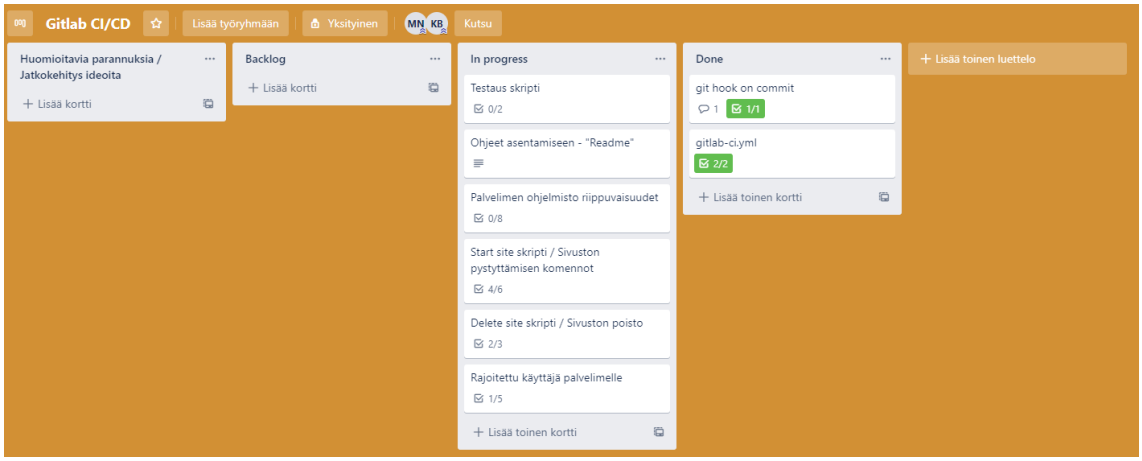
Projektin pääasiallisena menetelmänä käytettiin SCRUMin kaltaista järjestelmää. SCRUM on ketterän kehittämisen menetelmä, jossa projekti pilkotaan pienempiin tehtäviin (kuvio 7). Sprintit ovat aikarajoitettuja suunnitelmia, jotka vastaavat iteraatioita. Jokaisen sprintin lopussa pidetään palaveri, jossa käydään läpi sprintin aikana tuotettujen tulosten laatua. (Holcombe 2008, 15)



Kuvio 7: Scrum visualisoituna (visual-paradigm.com)

Sprinteille ei ollut määritelty mitään alitehtäviä, mutta selitimme aina ohjaajalle, mitä olimme saaneet sillä sprintillä tehtyä. Sovelsimme projektissa myös KANBAN-taulua, jonne merkitsimme projektin kannalta oleellisimmat tehtävät ja seurasimme edistymistä sitä kautta. Lisäksi hyödynsimme ketterän kehittämisen menetelmiä, joiden avulla kehitimme prosessin suunnitelmaa ja lisäsimme ominaisuuksia tarpeen tullen. Tämä osoittautui hyödylliseksi, koska emme olleet aikaisemmin olleet tekemisissä tällaisen järjestelmän kanssa, joten parannettavaa löytyi useita kertoja kehittämis- ja testausvaiheissa.

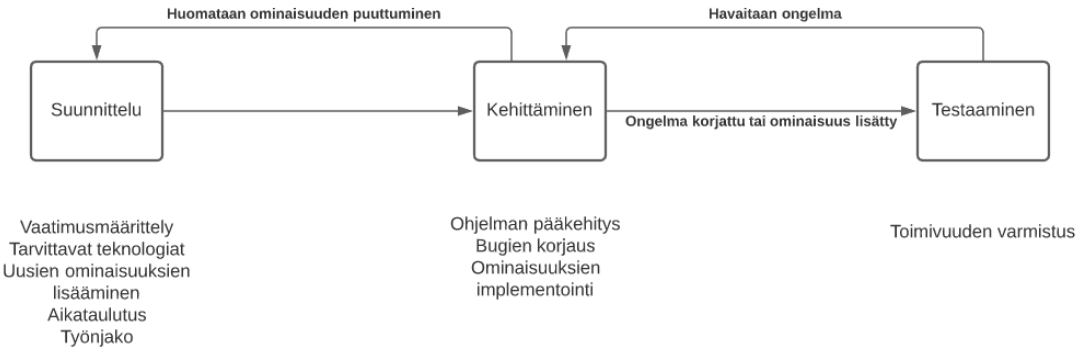
Projektin jaksottaminen kahden viikon sprintteihin oli mielestäni hyvä idea, sillä siinä ajassa kerkesimme aina hiomaan koodia tarpeeksi eteenpäin, mutta ei kuitenkaan niin paljon, että osa muutoksista pääsisi unohtumaan. Mietimme aina työkaverin kanssa, mitä haluaisimme saada aikaiseksi seuraavan sprintin aikana ja pyrimme saamaan asetetut tavoitteet maaliin. Kanban-taulu oli myös hyvä lisä projektin etenemisen seuraamiseen. Käytimme Kanban-taulun luomiseksi Trelloa. Trello tarjoaa helpon ja monipuolisen ympäristön projektin hallinnointiin. Kanban-kortteihin voi lisätä esimerkiksi tarkistuslistoja, liitetiedostoja, kuvia ja kortit voi nimittää tietyn henkilön vastuulle (kuvio 8).



Kuvio 8: Kuvakaappaus projektin Trello-taulusta

6 Projektin kulku

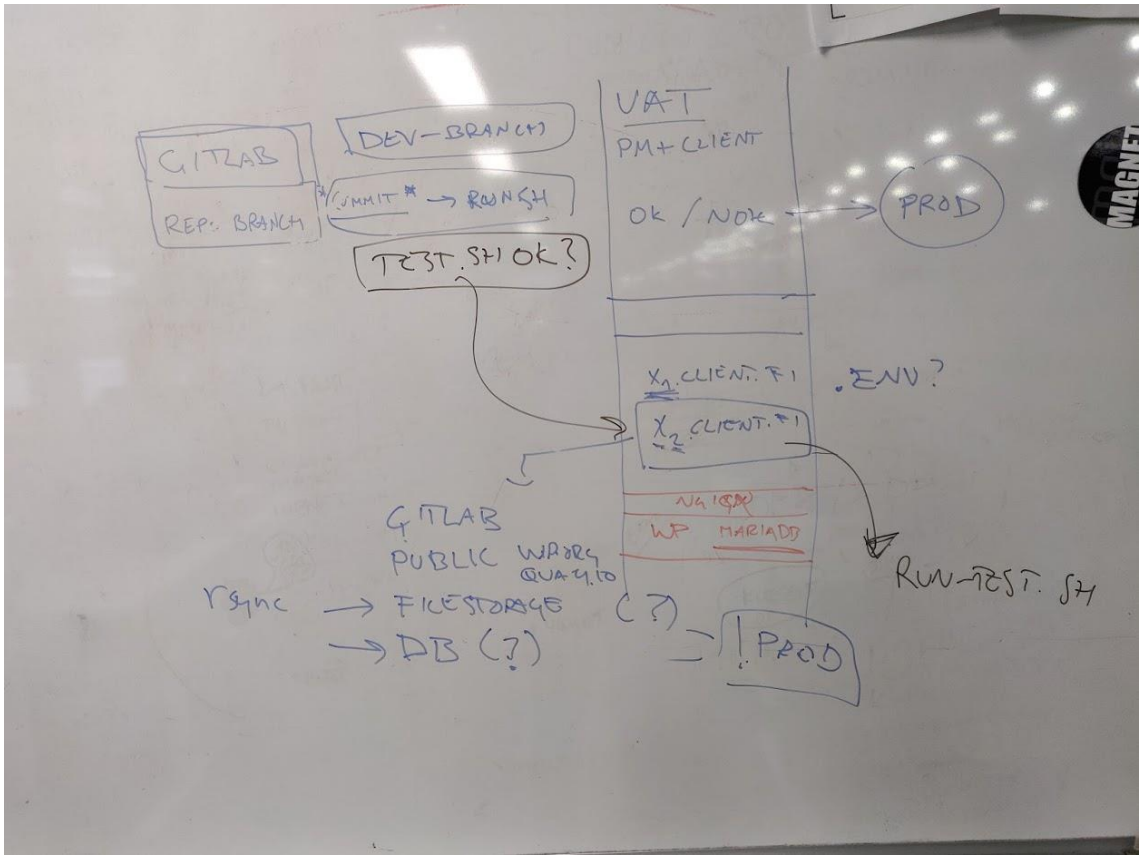
Tässä kappaleessa avataan tarkemmin projektin eri vaiheita. Vaiheet on jaettu karkeasti kolmeen eri osaan: suunnittelu, kehittäminen ja testaus. Suunnittelu-alakappaleessa käydään läpi projektin suunnittelu, vaatimusmäärittely ja aikataulu. Kehittäminen-alakappaleessa kerrotaan esimerkiksi, miten projektia kehitettiin, mitä teknologioita käytettiin ja miten ongelmat ratkaistiin. Testaus-alakappaleessa kerrotaan, miten prosessin toiminta varmistettiin, miten prosessi reagoi virhetilanteissa ja miten ilmenneet ongelmat ja bugit korjattiin. Kuvio 9 selventää, mitä kaikkea projektin eri vaiheissa tehtiin ja miten ja milloin vaiheiden välillä liikuttiin.



Kuvio 9: Kaavio projektin kulusta ja eri vaiheiden määrittelystä

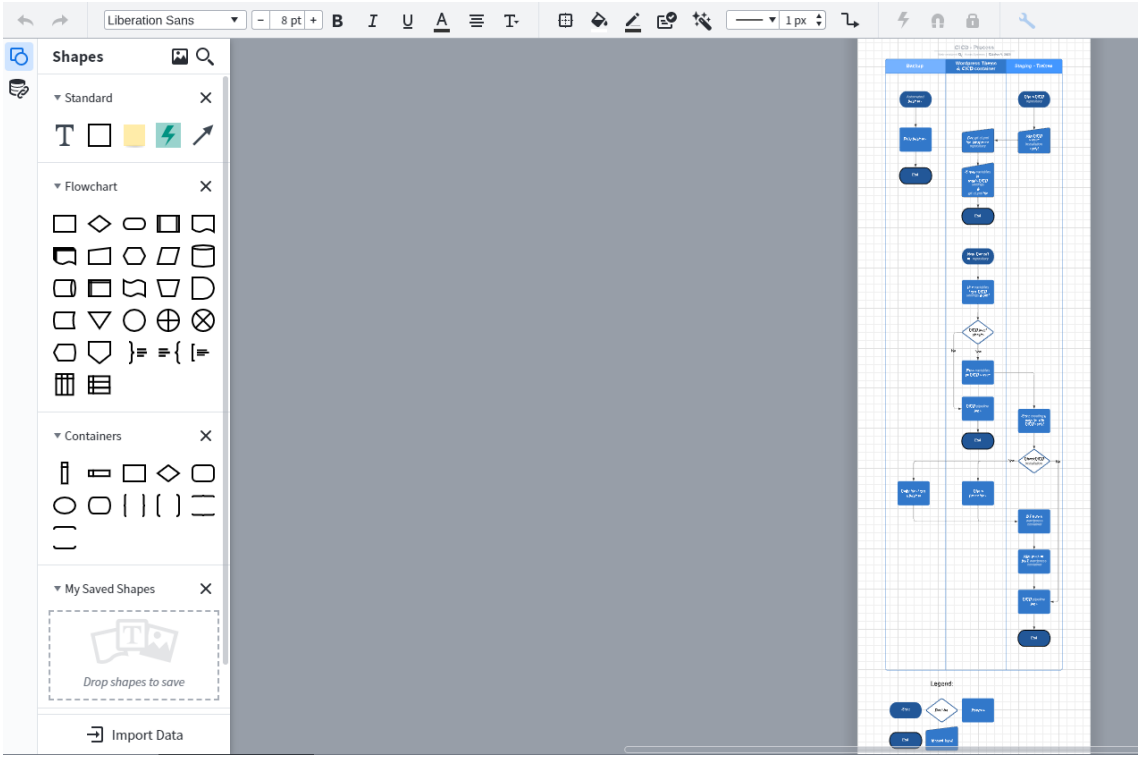
6.1 Suunnittelu

Projektin suunnittelu aloitettiin yhteisellä tapaamisella, johon osallistui lisäkseni työharjoittelun ohjaaja ja toinen harjoittelija. Ohjaaja selitti lyhyesti, mikä projektin tavoite on ja mitä vaatimuksia prosessilla on. CI/CD-prosessin tarkoitus on kuunnella GitLab repoa ja joka kerta, kun valittuun repon branchiin lisätään koodia, prosessi käynnistyy. Putkilinja ottaa SSH-yhteyden jollekin etäpalvelimelle, vetää sinne projektin tiedostot GitLabista, konfiguroi Docker kontin sivustoa varten ja nostaa kontin pystyyn. Kuviossa 10 on karkea luonnos prosessin toiminnasta.



Kuvio 10: Ensimmäinen luonnos prosessista

Tapaamisen jälkeen aloimme toisen harjoittelijan kanssa hahmottelemaan prosessia erilaisilla prosessikaavioilla. Yrityksen tarjoama Lucidchart-palvelu teki prosessikaavioiden piirtämisestä helppoa ja yksinkertaista, jolloin pystyimme luomaan pitkiäkin kaavioita nopeasti (kuvio 11). Lucidchart tarjoaa myös monia erilaisia valmiita templaatteja, jolloin kaavioita ei tarvitse itse rakentaa alusta asti. Käytimme prosessin piirtämiseen uimaratakaavioita (katso liite 1). Uimaradan avulla prosessin eri vaiheet voidaan sijoittaa eri radoille riippuen siitä, minkä prosessin alustan (GitLab, etäpalvelin) vastuulla kyseinen toiminto on. Tämä helpottaa myös ongelmatilanteiden ratkomista, kun kaaviosta voi nopealla vilkaisulla nähdä, millä alustalla virhe ilmenee.



Kuvio 11: Kuvakaappaus projektin Lucidchart-kaaviosta

6.2 Kehittäminen

Suunnittelun jälkeen aloimme pohtia, miten prosessi kannattaisi toteuttaa. Päädyimme käyttämään Bashia pääasiallisena ohjelmointikielenä projektissa, koska sivustojen pystyttäminen vaatii paljon tiedostojen siirtelyä ja jäsentelyä etäpalvelimella. Koska palvelin on Linux-pohjainen, on Bash erinomainen kielivalinta, sillä sen avulla etäpalvelimella voi tehdä melkein kaikki samat asia ohjelmallisesti kuin käyttäjä tekisi ne manuaalisesti. Lisäksi jaoimme projektin kahteen eri osaan: GitLabin puoli ja palvelimen puoli. Koska nämä alustat eivät suoraan ole liitettyinä toisiimme, tarvitsimme jonkinlaisen yhteyden GitLabin CI/CD-putken ja palvelimen välille.

6.2.1 GitLabin konfigurointi

GitLabin CI/CD-putket ovat aina projektikohtaisia, joten jokaiselle projektille täytyy luoda oma putkilinja erikseen, jos CI/CD-putkea halutaan käyttää kyseisessä projektissa. Tätä varten projektin repon juureen luodaan gitlab-ci.yml-tiedosto, johon asetetaan halutut

parametrit putkelle (kuvio 12).

```
variables:
    BACKUP_ZIP_PATH: /home/ci/http-wejazz-qa-web-veistamo-fi04-03-20.zip

default:
  image: debian:latest

stages:
  - remotesetup

remotesetup:
  only:
    - master
  stage: remotesetup
  image: kroniak/ssh-client
  before_script:
    # Setup SSH client
    - '[ -z $SSH_PRIVATE_KEY ] && echo "Check your SSH key" && exit 1'
    - '[ -z $SQL_ROOT_PASSWORD ] && echo "Please give the SQL root password" && exit 1'
    - echo "$THEME_REPOSITORY"
    - 'which ssh-agent || ( apt-get update -y && apt-get install openssh-client -y )'
    - mkdir -p ~/.ssh
    - chmod 700 ~/.ssh
    - eval $(ssh-agent -s)
    - '[[ -f /.dockerenv ]] && echo -e "Host *\n\tStrictHostKeyChecking no\n\n" > ~/.ssh/config'
    - echo "${SSH_PRIVATE_KEY}" |  tr -d '\r' | ssh-add -
  script:
    - ssh $REMOTE_HOST "startsite.sh
        -e $CI_REPOSITORY_URL
        $([ ! -z $BACKUP_ZIP_PATH ] && echo "-g ${BACKUP_ZIP_PATH}")
        $([ ! -z $WP_DATABASE_PREFIX ] && echo "-f ${WP_DATABASE_PREFIX}")
        -a $CI_PROJECT_NAME
        -b $CI_COMMIT_REF_NAME
        -c $CI_COMMIT_SHORT_SHA
        $SQL_ROOT_PASSWORD"
```

Kuvio 12: gitlab-ci.yml tiedosto

Kun tiedosto on määritetty projektin repoon, GitLab käynnistää CI/CD-putken aina, kun tiedostossa määriteltyyn branchiin (tai mihin tahansa branchiin jos sitä ei ole määritelty) tallennetaan koodia. gitlab-ci.yml, jota käytimme projektissa sisältää seuraavat määritykset:

- variables
    - Tämän avaimen alle voidaan määritellä muuttujia, joita voidaan käyttää myöhemmin osana CI/CD-putkea. Tällä hetkellä avain sisältää vain yhden arvon, joka on sivuston varmuuskopion sijainnin. Tätä ominaisuutta ei kuitenkaan ole vielä implementoitu varsinaiseen prosessiin, vaan sitä käytettiin testaustarkoituksiin.
- default: image
    - CI/CD-suorittajan käyttämä oletus Docker-kuva kontin luontia varten.
- stages
    - CI/CD-putken eri vaiheet voidaan määritellä tämän avaimen alla, jos esimerkiksi CI/CD-putki sisältää useamman suoritettavan vaiheen, kuten build (sovelluksen rakentaminen) ja testing (sovelluksen testaaminen)
- remotesetup
    - Tämä on kuviossa 12 esiintyvän vaiheen (stage) nimi. Sen sisällä määritellään esimerkiksi mitä Docker-kuvaa käytetään, missä branchissa ja mitä komentoja vaiheessa suoritetaan.
    - before_script
        - Ennen pääskriptiä before_script-avaimen avulla voidaan ajaa komentoja esimerkiksi CI/CD-putken Docker-kontin alustamiseksi. Tässä

projektissa before_script-avaimen avulla konttiin asennetaan ssh-agent ohjelma, jos sitä ei vielä ole asennettuna ja konttiin viedään SSH avain etäyhteyttä varten.

- script
  - Tämä avain sisältää komennot, joita halutaan suorittaa kyseisen vaiheen pääkomentoina. Tässä projektissa suoritetaan vain yksi komento, joka ottaa SSH-yhteyden palvelimelle ja ajaa palvelimelta löytyvän startsite.sh skriptin, joka käynnistää palvelimella sivuston rakentamisen.

Kun tiedosto on valmis, voidaan CI/CD-putkea alkaa käyttämään projektin kanssa.

6.2.2 Palvelimen konfigurointi

Käytimme verkkosivujen alustana Ubuntu-pohjaista Linux palvelinta, koska sitä on helppo käyttää ilman graafista käyttöliittymää, jolloin palvelimen ei tarvitse edes fyysisesti olla tavoitettavissa. Kaiken konfiguroinnin ja koodauksen teimmekin SSH-yhteyden avulla. SSH on verkkoprotokolla, jonka avulla voidaan kirjautua toiseen tietokoneeseen turvallisesti etänä. SSH tarjoaa monia erilaisia vahvoja tunnistautumistapoja ja verkkoliikenne suojataan vahvalla salauksella. (Ylönen 1996, 2)

Palvelin vaati muutamien ohjelmien asentamista toiminnan varmistamiseksi. Niistä tärkeimmät ovat:

- sudo
  - Rajoittaa käyttöoikeuksia palvelimella (Ohjelma ei vahingossa mene muuttamaan järjestelmän muita tiedostoja)
- openssh server
  - SSH palvelinohjelmisto, jotta palvelinta voidaan käyttää etänä SSH:n avulla
- git
  - Sivustojen tiedostojen lataamista varten GitLab reposta
- Docker ja docker-compose
  - Konttien pystyttämistä ja konfigurointia varten
- pwgen
  - Salasanojen automaattista luontia varten
- npm, NodeJS ja gulp
  - Sivustojen Wordpress-teemojen asentamista ja viimeistelyä varten
- unzip ja zip
  - Varmuuskopioiden hallintaan. Tätä ominaisuutta ei ole vielä implementoitu.

Teimme myös palvelimen konfiguroinnin helpottamiseksi bash skriptin, jonka avulla palvelimelle voidaan asentaa kaikki tarvittavat ohjelmat ja paketit, luoda käyttäjä ja antaa haluttu polku sivustojen asennuksille. Skriptin avulla palvelin voidaan konfiguroida luotettavasti niin, että CI/CD-prosessi voidaan implementoida mahdollisimman helposti uusille palvelimille.

6.2.3 Startsihe.sh -skripti

Aina kun CI/CD-prosessi käynnistyy, GitLab runner lähettää SSH:n yli komennon palvelimelle tietyillä parametreillä, jotka sisältävät esimerkiksi sivuston ja repon nimet, repon branchin, tietokannan etuliitteen ja repon URL osoitteen. Tämä skripti on vastuussa prosessin rakentamispuolesta, sen avulla sivuston sisältö ladataan palvelimelle, sivuston palvelin konfiguroidaan ja käynnistetään. Tämän skriptin kehittämiseen kului suurin osa projektin työtunneista, sillä se on kirjoitettu kielellä, josta minulla tai toisella harjoittelijalla oli vain vähän kokemusta. Koodia optimoitiin ja kirjoitettiin uudelleen useita kertoja, kun löysimme parempia ratkaisuja tiettyihin ongelmiin.

Skriptin alussa gitlab-ci.yml tiedostossa annetut parametrit asetetaan muuttujiin, jotta niitä voidaan helpommin käyttää muualla skriptissä (kuvio 13). Jos jotakin parametriä ei ole annettu gitlab-ci.yml tiedostossa, annetaan muuttujalle oletusarvo virheiden ehkäisemiseksi. Muuttujat, jotka saavat arvonsa parametrien kautta on nimetty niin, että ne alkavat alaviivalla, jotta ne erottuvat helpommin muista muuttujista, joita määritellään muualla ohjelmassa.

```
## Defining variables.
# opts
_WP_DATABASE_PREFIX=${_WP_DATABASE_PREFIX:-wp_}
_THEME_REPO_URL=${_THEME_REPO_URL:-}
_INSTALLATION_PATH=${_INSTALLATION_PATH:-$(readlink -f /docker/CICD/)}
_CI_REPO_NAME=${REPO_NAME:-"default"}
_CI_REPO_BRANCH=${COMMIT_BRANCH:-"dev"}
_CI_REPO_COMMIT_HASH=${COMMIT_HASH:-"$(date +%Y-%m-%d_%H-%M-%S)"}
_BACKUP_ZIP_PATH=${_BACKUP_ZIP_PATH:-}
# parameters
_SQL_ROOT_PASSWORD=${1:-}
```

Kuvio 13: Muuttujien alustaminen startsite.sh skriptissä

Muuttujien jälkeen skriptissä määritellään func_update_repository funktio (kuvio 14). Funktion tarkoitus on hakea sivuston pystyttämiseen vaadittavat repot, kuten Docker kuvaa varten luotu repo ja tietyissä projekteissa käytetyn Web-veistämön oman isäntäteeman repo. Funktioon annetaan parametreinä repon SSH- tai URL-osoite ja haluttu asennussijainti palvelimella. Kun funktiota kutsutaan skriptissä, repon osoite ja nimi tallennetaan muuttujiin ja funktio

tarkistaa, löytyykö kyseistä repoa jo palvelimelta. Jos repo löytyy, funktio päivittää repon sisällön ajamalla git pull -komennon. Jos repon nimistä kansiota ei löydy palvelimelta, funktio kloonaa (lataa) repon sisältöineen annettuun sijaintii ajamalla git clone -komennon. Tämän jälkeen funktio palauttaa repon sijainnin merkkijonona, joka tallennetaan muuttujaan myöhempää käyttöä varten.

```
function func_update_git_repository() {
  REPO_URL=$1
  INSTALL_PATH=$2

  if [ -n "${REPO_URL}" ]; then
    REPO_NAME=$(sed -E "s/.+\/([a-zA-Z0-9\-]+).git$/\1/g" <<< ${REPO_URL})
    REPO_PATH="${INSTALL_PATH}/${REPO_NAME}"
    if [ -d "${REPO_PATH}" ] \
    && [ $(git --git-dir="${REPO_PATH}"/.git remote get-url --push origin) == "${REPO_URL}" ]; then
      git --git-dir="${REPO_PATH}/.git" pull >&2
    else
      mkdir -p ${REPO_PATH}
      git clone ${REPO_URL} ${REPO_PATH} >&2
      #exits on git error
    fi
  fi

  #returns insstallation path
  echo $REPO_PATH
}
```

Kuvio 14: func_update_repository -lähdekoodi

Tämän jälkeen ohjelma vielä tarkistaa, löytyykö sivustolle varmuuskopioita tai teemaa annetuista parametreistä ja arvot tallennetaan muuttujiin.

Seuraavassa vaiheessa (kuvio 15) ohjelma luo projektille kansiorakenteen palvelimelle. Projektille luodaan ensin nimi, joka muodostetaan repon nimestä, branchin nimestä ja commit hashista (esim. projekti-master-abc123). Projektin polku haetaan parametreistä tai se sijoitetaan oletusarvoiseen sijaintiin. Tämän jälkeen ohjelma kopio Docker repon projektin polkuun ja kopio nimetään uudelleen projektin nimellä. Projektin teemoihin kopioidaan Web-veistämön pohjateema varmuuden vuoksi, sillä jotkut projektit tarvitsevat kyseistä teemaa toimiakseen.

Kun projektin kansiorakenne on kunnossa, voidaan projektin teema kopioida kansioon. Jos projektille ei ole vielä luotu teemaa, ohjelma kopio pohjateeman oletuslapsiteeman projektiin. Ohjelma vielä tarkistaa, löytyykö projektiin liittyvää varmuuskopiota ja kopioi sen, jos sellainen löytyy.

```
#-------------------------------------------------#
## Begin creating project directory
#Define variables
PROJECT_NAME=$_CI_REPO_NAME-$_CI_REPO_BRANCH-$_CI_REPO_COMMIT_HASH
PROJECT_PATH=$([[ -d ${_INSTALLATION_PATH} ]] && echo "${_INSTALLATION_PATH}/sites/client-${PROJECT_NAME}")

echo "Creating project directory..."
# Create base folder with docker repo
cp --recursive ${WP_DOCKER_PATH} ${PROJECT_PATH}
# Copy lankku theme to project
mkdir -p ${PROJECT_PATH}/wp-content/themes/weistamo-lankku
cp --recursive ${WP_LANKKU_PATH} ${PROJECT_PATH}/wp-content/themes/

# Theme creating logic
if [[ ! -z $HAS_THEME ]] && [[ $($HAS_THEME;echo $?) -eq 0 ]]; then
  echo "Copying theme from repo"
  # TODO improve copying path for different repository directory structures
  cp --recursive ${WP_THEME_PATH:-~}/* ${PROJECT_PATH}/wp-content/
else
  echo "Creating new lankku child-theme"
  cp --recursive ${PROJECT_PATH}/wp-content/themes/weistamo-lankku/child-theme ${PROJECT_PATH}/wp-content/themes/${THEME_NAME}
fi
# Backup handling logic
# TODO getting data from backup server
if [[ ! -z $HAS_BACKUP ]] && [[ $($HAS_BACKUP;echo $?) -eq 0 ]]; then
  # Use specified backup to get database and content files
  cp ${_BACKUP_ZIP_PATH} ${PROJECT_PATH}/data-import/
fi

## Run theme building commands
# Check for gulpfile
if [[ -f ${PROJECT_PATH}/wp-content/themes/${THEME_NAME}/gulpfile.js ]]; then
  cd ${PROJECT_PATH}/wp-content/themes/${THEME_NAME}
  npm install
  gulp build
  cd
fi

#-------------------------------------------------#
```

Kuvio 15: Kansiorakenteen luominen

Kun kaikki tarpeelliset tiedostot on kopioitu, ohjelma konfiguroi docker-compose.yml -tiedoston, jota käytetään Docker kontin käynnistämiseen annetuilla parametreillä (kuvio 16). Docker-composen käyttö helpottaa konfigurointia, kun kaikki asetukset ovat tallessa yhdessä tiedostossa, eikä niitä tarvitse kirjoittaa aina erikseen konttia käynnistäessä. Ohjelma aluksi luo kaikki tarvittavat muuttujat, kuten sivuston URL-osoitteen, tietokannan nimen, Wordpress salasanan ja docker-compose.yml -tiedoston sijainnin ja asettaa niille arvot. Tämän jälkeen ohjelma kopioi projektin kansioista löytyvät docker-compose.yml_example -esimerkkitiedoston ja alkaa muokata sitä. Sivuston URL-osoite muodostetaan samalla tavalla kuin projektin nimi yhdistämällä repon nimi branchin nimen ja commit hashin kanssa, jonka perään vielä lisätään palvelimen alidomain-osoite. Tietokannan nimi luodaan projektin repon nimestä ja commit hashista, joista väliviivat (-) korvataan alaviivoilla (_). Tietokannan ja sivuston salasana luodaan pwgen-nimisellä ohjelmalla, jonka tarkoitus on luoda satunnainen salasana. Kun kaikki nämä muuttujat on luotu, ohjelma kopioi docker-compose.yml_example -tiedoston projektin kansioon ja nimeää sen docker-compose.yml -tiedostoksi. Tämän jälkeen ohjelma avaa

tiedoston sed:llä, jonka avulla tiedostoon syötetään uudet asetukset juuri luoduista muuttujista.

```
sed -i.bak \
-e "s/testsite.com/${SITEURL}.trainee.web-veistamo.fi/g" \
-e "s/testdbname/${DB_NAME}/g" \
-e "s/wp_/${_WP_DATABASE_PREFIX}/g" \
-e "s/PASSWORD/${WP_PASSWORD}/g" \
${DOCKER_COMPOSE} && rm ${DOCKER_COMPOSE}.bak
```

Kuvio 16: docker-compose.yml asetusten muuttaminen sed:llä

Seuraavaksi ohjelma luo tietokannan edellisessä vaiheessa luoduilla muuttujilla. Ennen luomista ohjelma vielä tarkistaa, että tarvittavat muuttujat on asetettu. Jos näin ei ole, ohjelma antaa virhekoodin ja lopettaa suorituksen. Jos muuttujat ovat kunnossa, ohjelma ajaa kolme SQL-komentoa tietokantaan, joilla luodaan tietokanta, käyttäjä ja annetaan tarvittavat oikeudet käyttäjälle tietokannan muokkaamiseen. Käyttäjän luomisessa on vielä paranneltavaa, sillä emme löytäneet tapaa luoda käyttäjää ilman, että sille annetaan pääkäyttäjän oikeuksia. Pääkäyttäjän oikeuksien antaminen ohjelman käyttäjälle voi olla vaarallista, jos jokin menee pieleen ja ohjelma tekee muutoksia väärään tietokantaan. Tämän takia olisi hyvä, jos kyseisellä käyttäjällä olisi oikeudet vain ohjelman luomaan tietokantaan.

Kun tietokanta on määritelty, ohjelma tallentaa kaikki oleelliset muuttujat JSON-tiedostoon. Tällöin kaikki tarvittavat kirjautumis- ja tietokantatiedot ovat tallessa tiedostolla, jos niitä tarvitsee tulevaisuudessa. Lopuksi ohjelma vielä käynnistää sivuston kontin ja tulostaa tarvittavat tiedot, kuten sivuston URL-osoitteen ja kirjautumistiedot.

6.2.4 PHP_CodeSniffer (PHPCS)

CI/CD-prosessin osaksi kehitimme myös PHP_CodeSniffer-skriptin, joka tarkistaa koodin laatua ennalta määritellyn standardin mukaan. PHPCS muodostuu kahdesta skriptistä; phpcs-skripti tarkistaa koodin ja havaitsee mahdolliset rikkeet määriteltyyn standardiin nähden ja phpcbf-skripti korjaa automaattisesti kyseiset rikkeet. (PHP_CodeSniffer, n.d)

Koodin tarkistus toteutettiin Git-versiohallinnan hooks-ominaisuudella. Git hookit ovat skriptejä, jotka ajetaan aina, kun tietty toimenpide tapahtuu Git repossa (Atlassian, n.d.). Loimme skriptin precommit-hook -tiedostoon, joka ajetaan aina, kun uutta koodia commitoidaan repoon.

Skriptin alussa määritellään polut, joista phpcs ajetaan (kuvio 17). Phpcs asennetaan tähän polkuun composer-ohjelman avulla. Kun palvelimella on yksi globaali polku phpcs-asennukselle, on sitä helppo ajaa jokaisessa projektissa. Päätin asentaa phpcs:n palvelimen

kotihakemistoon, koska kansiorakenne on samanlainen kaikilla palvelimilla, jolloin phpcs:n käyttöönotto on vaivatonta millä tahansa palvelimella. Polut tallennetaan muuttujiin, jolloin niitä on helppo käyttää myöhemmin skriptissä. Lisäsin myös PREFIX-muuttujaan teksin ”Pre-commit: ”, joka helpottaa skriptin tulosteiden lukemista komentokehotteesta. Kaikki skriptin tulostamat tekstit alkavat tuolla kyseisellä tekstillä. Myös käytettävän standardin polku annetaan viimeiseen muuttajaan. Standardi on XML-tiedosto, jossa on määritelty, mitä standardeja vasten tiedostoja vertaillaan.

```
#!/bin/bash
PREFIX="Pre-commit: "
PHPCS_BIN=$HOME/precommithook/vendor/bin/phpcs
PHPCBF_BIN=$HOME/precommithook/vendor/bin/phpcbf
PHPCS_STANDARD=$HOME/precommithook/phpcs.xml
```

Kuvio 17: precommit-skripti, polkujen määrittely

Kun polut on määritelty, skripti käy läpi kaikki tiedostot, jotka ovat gitin staging-vaiheessa eli valmiina commitoitavaksi. Skripti tarkistaa jokaisen tiedoston päätteen kuvion 18 mukaisesti ja lisää tiedoston muuttujaan, jos tiedoston pääte sisältää tekstin ”.php”. Näin ainoastaan PHP-tiedostot lisätään listaan ja muut tiedostot, joita phpcs ei voi tarkistaa, jätetään huomioimatta.

```
## Loop all files that are about to be committed (diff of git head and staged)
echo "${PREFIX} ==> Checking for syntax standard violations..."
for FILE in $(git diff --cached --name-only); do
    case $FILE in
    *.module )
        ;&
    *.php )
        if [[ -f $FILE ]]; then
            PHP_FILES+="${FILE} "
        fi
        ;;
    esac
done
```

Kuvio 18: PHP-tiedostojen lisääminen muuttujaan

Seuraavaksi skripti ajaa phpcs-skriptin ja käy läpi kaikki tiedostot, jotka lisättiin edellisessä vaiheessa PHP_FILES-muuttujaan (kuvio 19). Ensin skripti tarkistaa, onko PHP_FILES -muuttujassa tarkastettavia tiedostoja. Jos muuttuja ei ole tyhjä, skripti ajaa phpcs:n PHPCS_BIN -muuttujasta löytyvän polun kautta, määrittelee standardin PHPCS_STANDARD -muuttujan avulla ja skannaa kaikki tiedostot, jotka ovat PHP_FILES -muuttujassa. Seuraavan vaiheen if-

konditionaali tarkistaa phpcs:n poistumiskoodin. Jos poistumiskoodi ei ole nolla, se tarkoitta, että phpcs havaitsi tiedostoissa rikkeitä. Skripti ilmoittaa käyttäjälle, että virheitä löytyi ja kysyy, haluaako käyttäjä varmasti tallentaa muutokset repoon. Jos käyttäjä vastaa kyllä (y), ohjelma poistuu loopista ja git commit -komento jatkaa ja tallentaa muutokset repoon. Jos käyttäjä vastaa ei (n), ohjelma keskeyttää git commit -komennon ja muutoksia ei tallenneta.

```
# Sniff staged files and notify user, if errors are found

if [[ ! -z $PHP_FILES ]]; then
    ${PHPCS_BIN} -s -p --colors --standard=${PHPCS_STANDARD} ${PHP_FILES} >&1
    if [[ $? -ne 0 ]]; then
        while true; do
        read -p "Errors found, do you want to commit anyway? (y/n) " yn
            case $yn in
                [Yy]* ) echo "Commiting..."
                        break;;
                [Nn]* ) exit 1;;
                * ) echo "Please answer y or n.";;
            esac
        done
    fi
fi
```

Kuvio 19: Tiedostojen tarkastaminen

Skriptiin on myös luotu vaihe, jossa PHPCBF korjaa virheet automaattisesti. Turvallisuussyistä tämä ominaisuus on toistaiseksi kommentoitu pois, koska ominaisuutta ei ole vielä kunnolla testattu asiakasprojekteilla. Jos skripti yrittää korjata koodia ja vahingossa rikkoo sen, joutuu kehittäjä ratkomaan asian itse. Ominaisuus otetaan käyttöön myöhemmin, kun sen toiminnasta on varmistuttu.

6.3 Testaus

Skriptin testaaminen tapahtui pitkälti manuaalisesti tuottamalla ohjelmalle erilaisia virhetilanteita. Yleensä ohjelmistoja testataan automatisoidulla testausskripteillä, jolloin ohjelmaa voidaan testata yhä uudelleen, kun sitä päivitetään ja ominaisuuksia lisätään. Emme kuitenkaan kokeneet automaattisten testien luomista tarpeelliseksi tässä vaiheessa, koska skriptin toiminta on hyvin suoraviivaista ja yksinkertaista. Manuaalinen testaaminen on tosin vaikeaa toteuttaa tarkasti, sillä monet virhetilanteet eivät tule edes mieleen. Varsinkin kun ohjelmaa on kehittänyt itse, syntyy helposti ”putkinäkö” ohjelman suhteen, eikä testejä osaa välttämättä ajatella tarpeeksi monesta näkökulmasta.

Heti ohjelman alussa tarkastetaan, että kaikki vaaditut globaalit muuttujat on annettu. Jos ohjelma huomaa, että jokin parametreistä puuttuu, se ilmoittaa käyttäjälle, mikä parametri

on puutteellinen, lopettaa suorituksen ja nostaa virhekoodin. Testasimme tätä toiminnallisuutta antamalla GitLabissa gitlab-ci.yml -tiedostoon virheellisiä tai puutteellisia parametrejä. Tarkistus toimi odotetusti ja antoi virheviestin ja lopetti suorituksen aina, kun huomasi virheellisen parametrin.

Toinen tärkeä tarkistus ohjelmassa on GitLab repojen tarkistaminen. Kuviossa 14 näkyvä funktio tarkistaa, että annetut git-osoitteet ovat toimivia osoitteita. Jos osoitteeseen ajettu git pull tai git clone -komento epäonnistuu, ohjelma lopettaa suorituksen. Käytännössä repojen URL-osoitteista vain Web-veistämön pääteeman osoite voi olla väärin, koska projektin repon URL-osoite saadaan automaattisesti siitä reposta, jossa CI/CD-prosessi käynnistetään. Testasimme toiminnallisuutta ja ohjelma lopetti suorituksen aina, kun git komennot epäonnistuivat.

Edellä mainitut kaksi vaihetta ovat ohjelman toiminnan kannalta tärkeimmät vaiheet tarkistaa. Jos nämä vaiheet ovat kunnossa, ohjelma tuskin törmää virhetilanteisiin suorituksen aikana. Ohjelmaan on kuitenkin asetettu muutamia ”tarkistuspisteitä” ohjelman toimivuuden tarkastamiseksi ja ohjelma lopettaa suorituksen niiden epäonnistuessa. Pyrimme testaamaan toimivuutta monipuolisesti erilaisilla virhetilanteilla, joilla suurin osa ongelmista saatiin korjattua. Todennäköistä on, että emme osanneet tuottaa kaikkia mahdollisia virhetilanteita, joihin ohjelma saattaisi normaaleissa olosuhteissa törmätä, mutta ne voidaan tulevaisuudessa korjata, kun prosessi otetaan virallisesti käyttöön. Mikään ohjelma ei ole suoraan täydellinen, vaan erilaisia bugeja ilmenee ajan mittaan ja ne voidaan jälkeenpäin korjata.

Precommit hookkia olemme testanneet asiakasprojekteissa syksyn aikana ja se on toistaiseksi toiminut odotetulla tavalla. Skripti on niin pieni, että mahdolliset ilmenevät virheet on todennäköisesti helppo korjata, jos sellaisia edes esiintyy. Skripti tullaan todennäköisesti ottamaan käyttöön nopeammin, kuin koko CI/CD-prosessia, koska se on helppo sisällyttää kaikkiin projekteihin.

7 Jatkokehitysehdotukset

Vaikka saimme kehitettyä prosessin toimivaan versioon, on siinä jonkin verran puutteita. Ensinnäkin CI/CD:n CI osuus jäi tämän projektin aikana melko vaillinaiseksi. Projekti keskittyi suurimmaksi osaksi jatkuvan julkaisun prosessin luomiseen ja se veikin melkein 90% projektiin käytetystä ajasta. Kun saimme CD prosessin tehtyä, jouduimme jättämään projektin tauolle, koska asiakasprojektit vaativat resurssejamme ja niillä oli suurempi prioriteetti. CI on tärkeä osa koko CI/CD kokonaisuutta ja sitä tulisi kehittää tulevaisuudessa ennen prosessin ottamista käyttöön tuotanto-olosuhteissa. Ilman perusteellista lähdekoodin testausta, emme voi olla

varmoja, toimiiko sivusto ongelmitta. Jatkuvan integraation tarkoitus on ytimessään parantaa toimitusvarmuutta ja laatua, joten kyseisen prosessin lisääminen prosessiin on tärkeää.

Lisäksi emme ehtineet toteuttaa varmuuskopioiden tuontia prosessiin. Jos prosessia halutaan käyttää tuotanto tai staging ympäristössä, on varmuuskopion tuominen tärkeä osa prosessia, koska se sisältää kaiken sivuston sisällön. Ilman varmuuskopiota, prosessi luo tyhjän Wordpress sivuston, jossa ei ole mitään sisältöä. Ehdimme manuaalisesti testaamaan, että prosessi kykenee tuomaan varmuuskopion mukaan sivuston luontivaiheessa, mutta varmuuskopiot pitäisi saada ladattua palvelimelle yrityksen varmuuskopiointivarastosta. Projektin alussa pidetyssä palaverissa keskustelimme siitä, miten varmuuskopiot voitaisiin ladata palvelimelle prosessin aikana, mutta emme ehtineet toteuttaa tätä.

Olisimme myös halunneet toteuttaa ominaisuuden, joka poistaa prosessilla luodun sisällön tietyn ajan päästä. Sivustot sisältöineen voivat viedä todella paljon tallennustilaa palvelimella sivuston laajuudesta riippuen, joten olisi hyvä, jos palvelin osaisi poistaa turhat sivut tietyin väliajoin tallennustilan vapauttamiseksi. Ehdimme projektin aikana tutkia, miten tämän voisi toteuttaa Linuxin cron-tehtävillä, mutta emme ehtineet sisällyttää tai kehittää ominaisuutta projektin aikana.

Olisi myös kiinnostavaa tutkia, miten Wordpress-pohjaisia verkkosivuja voisi testata. Emme projektin aikana ehtineet selvittää, miten Wordpress sivustoja yleensä testataan tai että onko niille virallista testaukseen tarkoitettuja ohjelmistokehyksiä. Koulussa suorittaessani JavaScript verkkokurssia sen aikana tutustuttiin JavaScript sovellusten yksikkötestaamiseen ja testipainotteiseen kehittämiseen (TDD, Test Driven Development). TDD:n tarkoitus on jatkuvasti testata sovellusta automatisoidulla testeillä, kun ominaisuuksia lisätään tai koodia päivitetään. Tällä pyritään vähentämään bugien ja virheiden päätymistä sovelluksen lähdekoodiin. Olisi kiinnostavaa tutkia, onko Wordpressille luotu testaamiseen tarkoitettuja ohjelmistokehyksiä.

8 Yhteenveto

Projektin aikana saimme kehitettyä toimivan konseptin, josta on helppo lähteä muovaamaan käytännöllistä CI/CD-prosessia verkkosivujen kehittämisen tueksi. Kehitetty prosessi vastaa määritelmää, jonka saimme projektin alussa ja sen pitäisi olla toimiva myös tuotantoympäristössä. Prosessissa on vielä varmasti paranneltavaa ja sitä voi optimoida, mutta projektin tarkoitus oli lähinnä tutkia, miten CI/CD-metodologiaa voitaisiin hyödyntää Wordpress-verkkosivujen kehittämisessä.

Vaikutus yrityksen työkäytäntöihin olisi melko minimaalinen, jos tämä prosessi implementoidaan osaksi kehitysprosessia. Se kuitenkin vähentäisi julkaisuihin liittyvää työmäärää

huomattavasti, koska prosessi tekee sen automaattisesti. CI/CD:n konfigurointi kuitenkin vaatii jonkin verran työpanosta, jos se halutaan sisällyttää projektiin. CI/CD prosessin käyttöönotto vaatii gitlab-ci.yml -tiedoston lisäämisen jokaiseen projektiin erikseen ja projektikohtaisten muuttujien asettamista tiedostoon.

Projektin laajuus huolestutti aluksi, koska aiempaa kokemusta tästä aihealueesta ei ollut ja en oikein tiennyt, mistä edes aloittaa. GitLab on onneksi kehittänyt todella vankan viitekehyksen CI/CD-prosessien luomiseen ja dokumentoinut sen erittäin hyvin. Dokumentaatio helpotti prosessin toiminnan opettelua ja kehityksen aloittamista paljon, kun kaikki tarvittava tieto oli yhdessä paikassa. Projekti vaati myös Bash-ohjelmoinnin harjoittelua, jota oli käsitelty koulussa vain vähän, mutta aiemmin hankittu ohjelmoinnin osaaminen auttoi, koska ohjelmointi on hyvin samankaltaista kielestä huolimatta. Bash on ohjelmointikielenä melko vanhanaikainen ja syntaksi on välillä sekavaa ja vaikealukuista, jonka takia ohjelman toimintaa sai palauttaa aina mieleen, jos projekti oli välillä tauolla.

Kesän aikana yritykseen palkattiin työtekijä, jonka vastuulla on yrityksen DevOps toimintojen kehittäminen. Ensisijaisesti DevOpsia kehitetään Avoin.Systems-aputoiminimen kehittämiin Odoo-toiminnanohjaus järjestelmiin ja myöhemmin Web-veistämön Wordpress-toteutuksiin. Tarkoitus olisi jollain tapaa integroida tämän opinnäytetyön aikana kehittämäämme CI/CD-prosessia osaksi Wordpressin DevOps-järjestelmää. Olemme syksyn aikana alkaneet panostaa koodin testaamiseen ja yritämme saada tätä CI/CD-järjestelmää osaksi kehitysprosessia joissakin projekteissa.

Kaikista ongelmista huolimatta tämä projekti oli hieno oppimiskokemus ja se auttoi ymmärtämään, miten CI/CD-prosessit käytännössä toimivat. Oma osaamiseni kehittyi paljon projektin aikana ja opin paljon uusia asioita, joista en ollut edes kuullut aikaisemmin. Jatkuvan integraation ja julkaisun menetelmien omaksuminen yrityksen toimintatapoihin on hyvä idea kaiken kokoisille yrityksille, mutta varsinkin sellaisilla, joilla on paljon asiakasprojekteja työnalla. Näiden menetelmien avulla kehittäjät voivat keskittyä kehittämiseen ja vähentää turhia ylläpitotehtäviä.

Lähteet

Sähköiset

Atlassian. N.d. DevOps. Viitattu 26.8.2020. https://www.atlassian.com/devops

Holcombe, W. 2008. Running an Agile Software Development Project. John Wiley & Sons Inc.

Sharma, S. 2017. The DevOps Adoption Playbook: A Guide to Adopting DevOps in a Multi-Speed IT Enterprise. John Wiley & Sons Inc.

Sacolick I. 2020. What is CI/CD? Continuous integration and continuous delivery explained. Viitattu 29.6.2020. https://www.infoworld.com/article/3271126/what-is-CI/CD-continuous-integration-and-continuous-delivery-explained.html

Codeship. N.d. Continuous Integration Essentials. Viitattu 17.08.2020. https://codeship.com/continuous-integration-essentials

Docker Documentation. N.d. Docker overview. Viitattu 29.6.2020. https://docs.docker.com/get-started/overview/

GNU. 2019. GNU Bash Manual. Viitattu 29.6.2020. https://www.gnu.org/software/bash/manual/html_node/What-is-Bash_003f.html

Gitlab. N.d. Simplify your workflow with GitLab. Viitattu 29.6.2020. https://about.gitlab.com/stages-devops-lifecycle/

Github. N.d. About repositories. Viitattu 21.7.2020. https://docs.github.com/en/github/creating-cloning-and-archiving-repositories/about-repositories

Ylönen T. 1996. SSH - Secure Login Connections over the Internet. Espoo: SSH Communications Security. https://pdfs.semanticscholar.org/617a/bee0122f200e4684c4b1365f2b45c3fba6fb.pdf

GitLab. N.d. Gitlab CI/CD. Viitattu 28.8.2020. https://docs.gitlab.com/ee/ci/

DevOps Enterprise Summit, 2015: DOES15 - Nicole Forsgren - How Continuous Delivery and Lean Management Make your DevOps Amazeballs, YouTube. Julkaistu 15.12.2015. https://www.youtube.com/watch?v=opEKZpAOhIk. Viitattu 28.9.2020

Github. N.d. PHP_CodeSniffer repository. Viitattu 02.10.2020. https://github.com/squizlabs/PHP_CodeSniffer

Atlassian. N.d. Git hooks. Viitattu 02.10.2020. https://www.atlassian.com/git/tutorials/git-hooks#:~:text=Git%20hooks%20are%20scripts%20that,in%20the%20development%20life%20cycle.

Git. N.d. Viitattu 9.10.2020. https://git-scm.com/

Kuviot

Liitteet

Liite 1: Uimaratakaavio prosessista