Haaga-Helia
ammattikorkeakoulu Oy

CI/CD isossa yrityksessä

Laureat Grepi

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2020

Tiivistelmä

Haaga-Helia
ammattikorkeakoulu Oy

Tekijä(t)
Laureat Grepi

Koulutusohjelma
Tietojenkäsittely

Raportin/Opinnäytetyön nimi
CI/CD isossa yrityksessä

Sivu- ja liitesivumäärä
26

CI/CD tarkoittaa jatkuvaa integrointia ja julkaisua. Tämä on osana DevOps menetelmiä, missä pyritään automatisoimaan sovelluskehityksen ja it-toimintojen välisiä prosesseja. Tavoitteena on tutkia jatkuvan integroinnin ja julkaisun menetelmiä, niihin kuuluvia ohjelmistotyökaluja ja kyseisten järjestelmien vaikutuksia yrityksiin. Raportti tehtiin tutkimalla olemassa olevia artikkeleita aiheesta, sekä peilaamalla omia kokemuksia tämän osa-alueen osalta.

Jatkuvan integroinnin (CI) ja julkaisun (CD) järjestelmiä voidaan kehittää erilaisilla ohjelmistotyökaluilla, mitkä toimivat tähän tarkoitukseen. Kehitys- ja tuotantoympäristöt voidaan pystyttää esim. Terraform -työkalulla automaattisesti ja ne voidaan konfiguroida Ansiblella juuri sellaiseksi, kuin vaatimukset määrittelevät. Virtualisoinnilla voidaan eristää ympäristöjä ja kopioida niitä moneksi identtiseksi ympäristöksi. Tähän voidaan käyttää esimerkiksi Docker -työkalua luomaan tietynlaisia ohjelmistopaketteja ns. kontiksi ja käyttää sitä, vaikka testiympäristönä.

Tämä raportti käsittelee sovelluskehitysmaailmassa tapahtuvaa koodin jatkuvaa integroimista ja julkaisua, sekä niiden toimintaa ja vaikutusta yritystoiminnassa. Nämä kaksi asiaa ovat sovelluskehityksessä tärkeitä, sillä ne voivat vaikuttaa myös yrityksen liiketoimintaan merkittävästi. Ohjelmistoprojektien etenemisen kannalta on tärkeää pystyä implementoida ja julkaista sopivissa sykleissä, ettei liian suuria kasoja kerry yhdelle julkaisukerralle. Tämän takia jatkuvalla integroinnilla ja julkaisulla on automatisoitu näitä prosesseja. Koodia voidaan testata ja integroida vaikka päivittäin ja julkaisuja voidaan tehdä esimerkiksi kahden viikon välein. Raportissa käydään myös hieman läpi DevOpsia ja miten jatkuva integroiminen ja julkaiseminen liittyy siihen.

Yritykset ovat selkeästi hyötyneet näistä menetelmistä, niin teknisesti kuin liiketoiminnallisesti. Automatisoimalla näitä prosesseja, on voitu säästää ajassa ja rahassa. Puhumattakaan siitä, että tällä tavalla tärkeät prosessit, kuten integroiminen ja julkaisu, tapahtuvat useammin, paljon turvallisemmin ja vakaammin.

Asiasanat
Jatkuva integrointi, Jatkuva julkaisu, DevOps, Kontti, It-infrastruktuuri

Sisällys

1 Johdanto

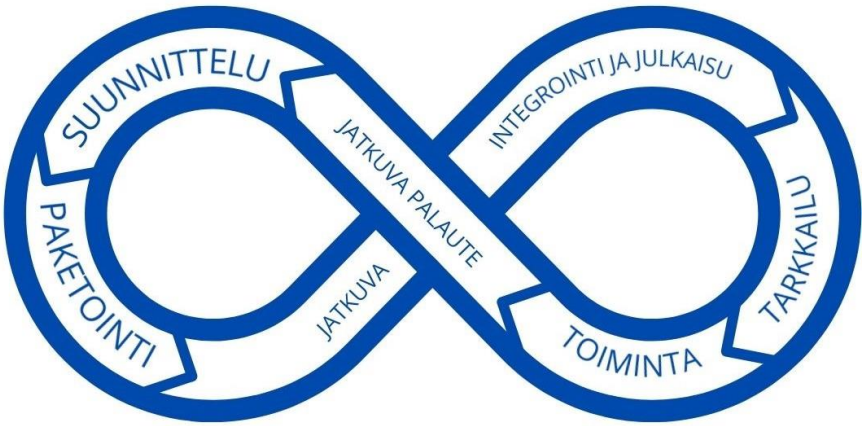
Valitsin tämän aiheen, koska se osa päivittäistä työelämääni ja koen nämä käsiteltävät aiheet erittäin tärkeäksi myös koko it-alalla. DevOps (Development Operations) on usein osana yritysten sovelluskehitysprojektissa ja olen itsekin ollut mukana projekteissa rakentamassa CI/CD (Continuous Integration/Continuous Deployment) järjestelmiä yritysten käyttöön. Tämä raportti avaa hieman DevOpsin maailmaa ja keskittyy enemmän CI/CD järjestelmiin sekä siinä tarvittaviin työkaluihin. Raportissa myös selviää, miten tällaiset järjestelmät näkyvät yrityksissä ja niiden projekteissa.

Viime aikoina DevOpsin soveltaminen it-alan yrityksissä, on nostanut CI/CD järjestelmien tärkeyttä ja muutenkin yleistä näkyvyyttä alalla. Yleisimpiä osa-alueita it-alalla voivat olla esimerkiksi backend koodaus, sovellusten käyttöliittymien kehitys, fyysiset tietotekniset laitteet, verkot ja muu it-infrastruktuuri. Itse koen DevOpsin olevan erittäin olennainen osa it-alaa. Kun tarkkailee kuinka suurta DevOps osaamisen kysyntä on nykypäivänä, voisin uskoa, että sen opetuksen lisääminen korkeakouluissa on hyvin mahdollista.

DevOps sana on lyhenne termeistä ”Development” ja ”Operations”. Tunnettu Australialainen ohjelmistoyritys Atlassian antaa verkkosivuillaan hyvän määritelmän DevOpsista:

> “Devops on joukko käytäntöjä, mitkä yhdessä automatisoivat ja integroivat prosesseja ohjelmistokehityksen ja It tiimien välillä, jotta voidaan paketoida, testata ja julkaista ohjelmistoa nopeammin ja turvallisemmin. ” (Atlassian.com, 2020)

DevOps on siis ohjelmistokehityksen eri osa-alueiden prosessien automatisointia ja toisiinsa integroimista. Nykypäivän it-alan ketterissä tuotekehityksien projektimenetelmissä tapahtuu niin paljon nopealla tahdilla. Taustalla on erittäin tärkeä olla vakaa ja luotettava menetelmä integroida (CI) uudet toiminnallisuudet ja julkaista (CD) ne samaan tahtiin. Tällaista tekeminen on DevOpsia

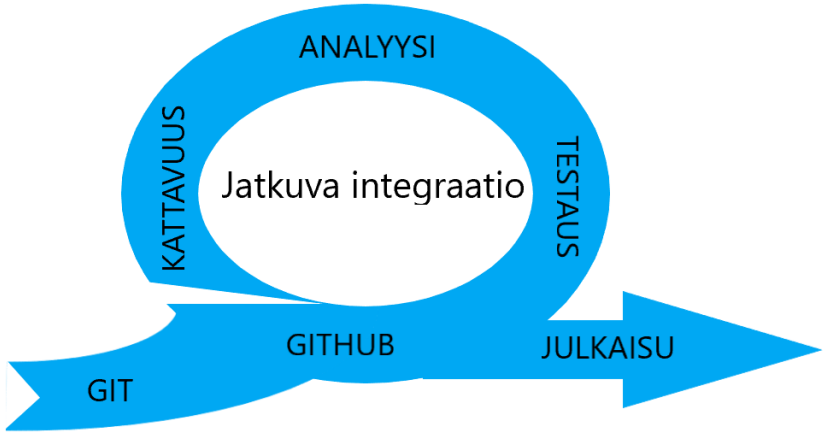


Kuva 1. DevOps symboli (mukaillen Atlassian.com, 2020.)

Kuten kuvassa 1 näkyy, DevOps kuvataan usein äärettömyyden symbolilla. Tämän tarkoitus on kuvastaa DevOpsin ”ikuisesti” jatkuvaa toimintaa ohjelmistokehityksen taustalla

Tämä raportti keskittyy kuitenkin enemmän CI/CD toimintaan. Kuten sanottu CI ja CD lyhenteet tulevat sanoista ”Continuous Integration” ja ”Continuous Deployment” ja ovat osa DevOps menetelmiä. CI on tärkeä osa erilaisissa ohjelmistokehitysprojekteissa. Sillä pyritään automatisoida uuden koodin integroimista olemassa olevaan. Usein kun integroidaan uutta koodia, halutaan esimerkiksi tietylle muutokselle ajaa joitain testejä tai isoimmissa projekteissa eräänlaisia ”end to end” testejä, missä varmistetaan, että muutos ei riko mitään prosessia tuotteen toiminnassa. CI järjestelmällä tämä kaikki voidaan automatisoida ja suorittaa säännöllisesti ja turvallisesti. CI:hin tarvitaan yleensä versionhallintatyökalua (GitHub, Gitlab jne.) jotta sen toiminnasta saadaan kaikki tarvittava irti. Ohjelmistoyritys Atlassian kuvailee CI järjestelmään seuraavilla sanoilla:

> “Jatkuva integraatio (Continuous Integration) on käytännössä sitä, että automatisoidaan useita koodimuutoksia useammalta tekijältä yhdessä ohjelmistoprojektissa. Tämä prosessi on yhdistelmä työkaluja, mitkä varmistavat koodin oikeellisuuden ennen sen integroitumista. Versionhallintatyökalu on CI järjestelmän ydin” (Atlassian.com, 2020.)



Kuva 2. CI järjestelmä (mukaillen Devonblog.com, 2019.)

Kuten kuvasta 2 näkyy, CI toiminta tapahtuu muutoksen versiohallintaan lisäämisen jälkeen ja ennen muutoksen julkaisemista. Olen itse työelämässä huomannut tämän olevan erittäin tärkeää ohjelmistokehitysprojekteissa. Tällaisella järjestelmillä voidaan esimerkiksi huomata ja korjata virheet aikaisessa vaiheessa projektin etenemisen näkökannalta.

CD on toinen osista mitä tämä raportti myös käsittelee. Se on lyhenne sanoista ”Continuous Deployment”. Tämä DevOpsin osa-alue on vastuussa jatkuvasta ohjelmiston muu-

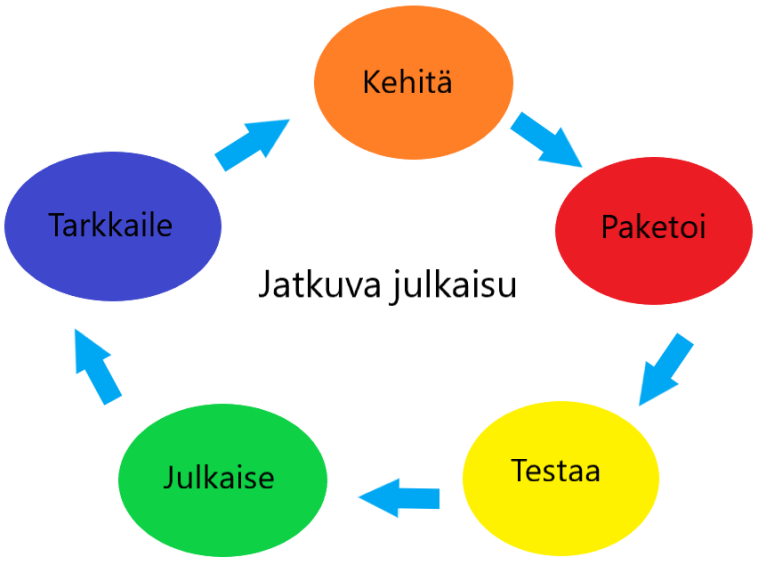
tosten julkaisemista. Tämä voi tarkoittaa julkaisua kehitysympäristöön tai itse tuotantoympäristöön. Ohjelmistoprojekteissa tuotetta kehitetään jatkuvasti, ja näitä muutoksia halutaan myös julkaista sitä mukaan, kun niitä tehdään ja todistetaan toimivaksi.

Atlassianin mukaan Continuous Deploymentin määritelmä on:

> “Jatkuva julkaiseminen (Continuous Deployment) on ohjelmiston julkaisuprosessi, mikä käyttää hyväkseen automaattista testausta validoidakseen koodin oikeellisuuden ja vakauden mahdollista julkaisua varten” (Atlassian.com, 2020.)

Ennen kuin CD prosesseja osattiin kehittää, uusien muutoksien julkaiseminen oli työlästä ja aikaa vievää.

> “Ohjelmistojulkaisun kiertokulku on kehittynyt ajan saatossa. Perinteinen tapa siirtää koodi käsin yhdestä koneesta toiseen ja tarkastaa sen oikea toiminnallisuus on ollut erittäin virhealtis ja raskas prosessi. Nykypäivänä työkalujen avulla voidaan automatisoida koko julkaisuprosessin, mikä antaa yritysten keskittyä olennaisiin tehtäviin” (Atlassian.com, 2020.)



Kuva 3. Continuous Deployment (mukaillen Devonblog.com, 2018.)

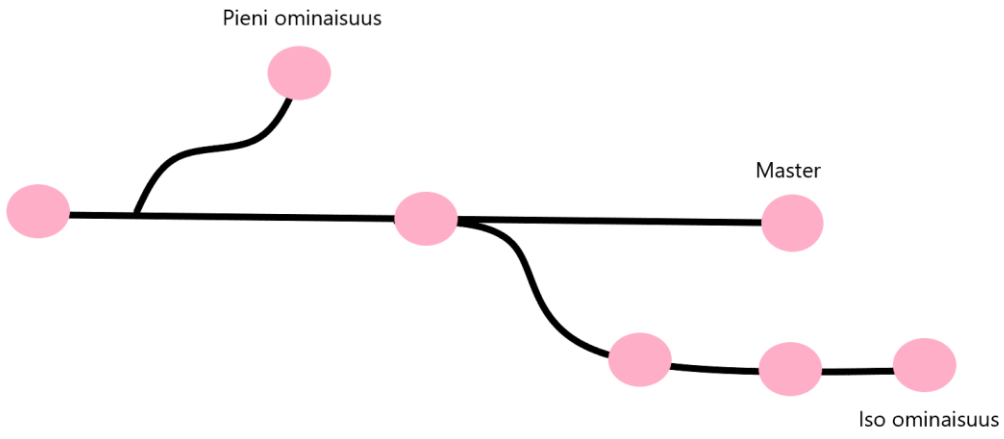
Raportin tehtävä on syventää tekijän ymmärrystä CI/CD järjestelmiin ja niiden tarkoitukseen sovelluskehityksessä. Tavoitteena on myös tutkia, kuinka CI/CD järjestelmät ovat vaikuttaneet yritysten toimintoihin ja tuloksiin. Raportissa käydään myös läpi eri ohjelmistotyökaluja, joita tyypillisesti käytetään näissä kyseisissä järjestelmissä. Tavoitteena on myös kertoa yleisesti DevOpsista käsitteellisellä tasolla. Projekti ei tuota tuloksena konkreettista CI/CD järjestelmää, vaan enemmänkin tutkii sitä ja sen toimintaa.

2 CI/CD osana DevOpsia

2.1 CI – Continuous Integration

Kuten johdannossa jo selvisi, CI/CD on osana DevOps toimintaa. DevOpsin kaltaisessa prosessissa halutaan automatisoida kaikkea mahdollista tuotekehityksestä, aivan sen julkaisemiseen saakka. Tällaisen prosessin tärkeänä osana on uuden koodin lisäämisen olemassa olevaan. Tämä voi kuulostaa yksinkertaiselta prosessilta, mutta ajan saatossa on huomattu, että uuden koodin sellaisenaan lisääminen olemassa olevaan ei ole todistunut niinkään helpoksi hommaksi. Nykypäivänä tässä asiassa meitä auttavat hienot versionhallintatyökalut.

Kuvitellaan 20 ihmistä työskentelemässä saman koodin parissa, kehittelemässä eri ominaisuuksia tuotteelle. Ilman versionhallintatyökaluja, koodihan olisi erittäin haasteellista pitää ajan tasalla, toimivana ja mahdollisimman tehokkaana. Versionhallintatyökaluilla esim. GitLab, voidaan hallita suurtenkin joukkojen työtä saman koodin ympärillä, menettämättä ideaa siitä, mikä versio koodista on jo tuotannossa, ja mitä on vielä kehitteillä. Hyvällä versionhallinta työskentelyllä esimerkiksi 10 ihmistä voi olla koodaamassa tuotteeseen uusia ominaisuuksia ja vaikkapa 10 seuraavaa ihmistä voi olla korjailemassa tuotteesta löytyneitä koodivirheitä. Tämäkin on vain kuitenkin esimerkki siitä, miten versionhallintaa voi tehdä. Kuva 4 näyttää miten versionhallintatyökalulla, koodi voi jakaantua eri ”haaroihin” eri merkityksien mukaisesti. Nämä ”haarat” kuvastavat eri versioita koodista.



Kuva 4. Git Branch (mukaillen Atlassian.com, 2020.)

Tässä kohtaa mukaan astuu automaattinen integroitumisprosessi. Kuten voidaan kuvitella, ohjelmistoprojekteissa uutta koodia tulee jatkuvasti ja tuotteelle halutaan kehittää uusia ominaisuuksia mahdollisimman nopeasti ja tehokkaasti. CI tarjoaa projekteille automatisoitua prosessia, missä muutokset integroidaan olemassa olevaan koodiin turvallisesti. Sitä mukaan, kun koodarit kehittävät uusia muutoksia, täytyy ne testata ensin, ennen kuin ne integroidaan. CI järjestelmällä voit jokaiselle muutokselle ajaa esim. niin sanotut ”porttitestit”, mitkä nimensä mukaan ovat ”portti” tähän olemassa olevaan koodiin. Tällaisilla testeillä voit esim. taata, ettei mikään muutos riko jotain olemassa olevaa koodia.
Esimerkki: Yritys A:lla on olemassa nettisivut julkaistuna internetissä, mutta niitä myös kehitetään jatkuvasti samalla. Sivujen etusivulla oikeassa yläreunassa näkyy yritys A:n logo suurella fontilla. Tämän yrityksen CI prosessissa yhtenä ”porttitestinä” voi olla se, että varmistetaan logon näkyvyys etusivulla. Tämän avulla, kaikki muutokset mitä näihin nettisivuihin ollaan tekemässä, tullaan varmuuden vuoksi tarkastamaan siitä, että löytyykö logo enää etusivulta. Joku voi esimerkiksi vahingossa muokata koodia siten, että poistaisi logon näkyvistä, jolloin CI järjestelmä varoittaa tästä, eikä anna muutoksen integroitua olemassa oleviin nettisivuihin.

Nimensä mukaisesti, CI järjestelmät pyrkivät jatkuvaan koodin integroimiseen, joten tällaisia järjestelmiä on hyvä käyttää koodin ja muutoksien testaamiseen ja paketoimiseen.

2.2 CD – Continuous Deployment

Kun ohjelmistoprojektissa on käytössä tehokas ja turvallinen integrointiprosessi, tarvitaan siihen usein myös kaveriksi turvallinen julkaisuprosessi (Continuous Deployment) julkaisemaan kaikki nämä uuden muutokset mitä jatkuvasti tulee. Sitä mukaan kuin koodiin integroidaan uusia ominaisuuksia tai muutoksia, ne halutaan julkaista myös tuotannossa tai kehitysympäristössä olevaan tuotteeseen. Esimerkki: Eräässä yritys A:n projektitiimissä kehitetään yrityksen olemassa olevia nettisivuja, ja tavoitteena on lisätä sivuihin uusi sivu, uudella sisällöllä. Uuden koodin valmistumisen ja integroimisen jälkeen, se halutaan julkaista myös internettiin ihmisten käytettäväksi. CD työkalujen avulla voi automaattisesti luoda uudesta integroidusta koodista julkaistavan version ja julkaista sen tuotantokäytössä olevilla palvelimilla. Tällaisella prosessilla taataan muutosten ja uuden ominaisuuksien jatkuva julkaiseminen automaattisesti sen sijaan, että se tehtäisiin manuaalisesti joka kerta.

Ilman julkaisuprosessia, integrointiprosessista ei välttämättä saa otettua kaikkea irti. Uudet ominaisuudet ja muutoksen tuotteesta halutaan julkaista automaattisesti ja sitä mukaan, kun niitä hyväksytään ja integroidaan.

Tästä huomaakin CI/CD:n roolin DevOpsissa ja mitä se käytännön tasolla tarkoittaa. Tällaisilla järjestelmillä voidaan automatisoida suuria, yritykselle ja tuotteelle erittäin tärkeitä ja tehokkaita prosesseja. DevOps ei voi kuitenkaan koodata ihmisen puolesta tuotteen lähdekoodia, mutta melkein kaikki prosessit sen jälkeen aivan tuotteen julkaisemiseen asti, on automatisoitavissa.

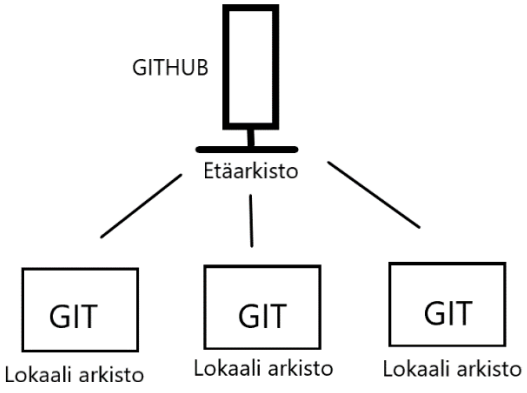
3 CI/CD Työkalut

CI/CD järjestelmä koostuu yleensä useasta eri kokonaisuudesta. Näiden kokonaisuuksien kehittäminen, yhdisteleminen ja automatisointi on DevOpsia. Tähän tarvitaan usein erilaisia ohjelmistotyökaluja, mitkä on suunniteltu niemenomaan tähän tarkoitukseen. Tässä raportissa listataan muutama yleisistä käytössä olevaa työkalua

3.1 Git – Versionhallinta

Versionhallintajärjestelmiä on useita, mutta yksi suosituimmista on tällä hetkellä Git. Versionhallintajärjestelmien avulla, viiden tai 500:an ihmisen yhteistyö ohjelmistoprojektissa on mahdollista sujuvasti ja vaivatta. Sen käyttö on niin kannattavaa projektin etenemisen kannalta, että uskon sen olevan käytössä tavalla tai toisella, jokaisella ohjelmistoyrityksellä. Tiedän että sana ”kaikilla” on hieman rohkea väite, mutta sen verran vakuuttunut olen sen hyödyistä, että uskallan sanoa niin.

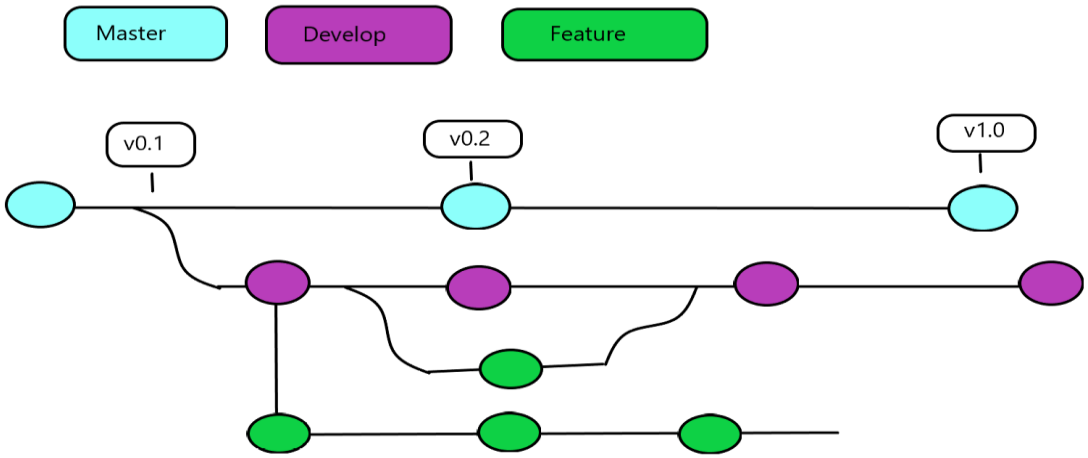
Git -työkalujen idea perustuu siihen, että samasta koodista (tai mistä ikinä tiedostostakaan) voidaan säilyttää useita eri versioita, näin ollen nimi ”versionhallinta”. Tämä toiminnallisuus mahdollistaa jopa suurien ihmismäärien samanaikainen työskentely saman koodin äärellä. Ajatuksen on, että esim. lähdekoodista on yksi ”aito” versio toiminnassa tuotannossa. Usein tätä versiota kutsutaan ”master” versioksi. Otetaan esimerkiksi vaikkapa nettisivut. Versionhallinnassa oleva data ei sijaitse fyysisesti omalla koneellasi, vaan työkalun omilla palvelimilla. Ihmiset, jotka haluavat muokata tai tarkastella tätä koodia omalta tietokoneelta, joutuvat kloonaaman tämän itselleen. Tällaista koodikokonaisuutta kutsutaan arkistoksi, engl. *repository.* Eli tässä tapauksessa muiden saatavilla oleva versio koodista sijaitsee versionhallintatyökalun palvelimilla (etäarkisto, engl. *remote repository*), ja työntekijöillä on vain klooni siitä koodista omalla tietokoneellaan (paikallisarkisto, engl. *local repository*).



Kuva 5. GitHub Repositories (mukaillen Trailhead.com, 2020.)

Kuva 5 havainnollistaa lokaali- ja etäarkistot ja niiden yhteyden toisiinsa. Kuvassa on käytetty esimerkkinä GitHub versionhallintatyökalua mikä on tällä hetkellä yksiä käytetyimpiä git -työkaluja.

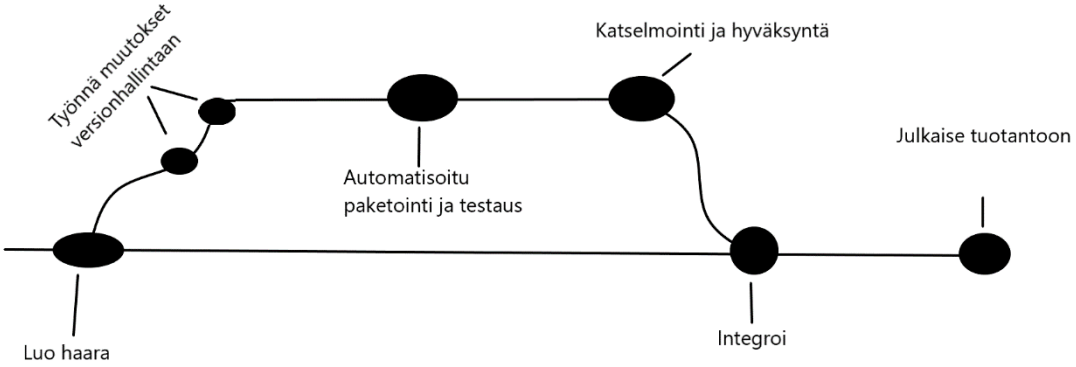
Kun ihmiset haluavat muuttaa tai lisätä uutta koodia, he tekevät sen paikallisesti omalla tietokoneellaan, ja työntävät tämän muutoksen tai lisäyksen git -työkalun avulla etäarkistoon. Yleensä on tapana työntää muutokset ensimmäiseksi, johonkin muuhun haaraan/versioon kuin tuotantoon. Tällä yritetään välttää virheellisen tai puutteellisen koodin työntämistä tuotantokoodiin.



Kuva 6. Git Flow (mukaillen Atlassian.com, 2020.)

Versioita koodista voi olla monia, mutta tässä tapauksessa ”Master” versio on niin sanotusti tärkein. Tätä versiota koodista päivitetään tietyn aikavälein ja niitä kutsutaan julkaisuksi. Continuous Deployment prosesseilla pyritään tekemään tätä julkaisua jatkuvasti, uusien muutosten ja ominaisuuksien valmistuttua.

Versionhallintatyökalut ovat usein osana CI/CD järjestelmiä ja toimii isona roolina tässä prosessissa. Vaikka versionhallintaa käytetään myös näiden järjestelmien ulkopuolella, se on erittäin olennainen osa jatkuvaa integrointia. Jokainen muutos halutaan lisätä ja julkaista sulavasti ja automaattisesti, joten kaikkia muutoksia ja versioita halutaan kontrolloida selkeästi. Kuvassa 7 näkyy, kuinka versionhallinta toimii ensiaskeleina ja laittaa niin sanotusti pyörät pyörimään, tällaisessa CI/CD järjestelmässä.

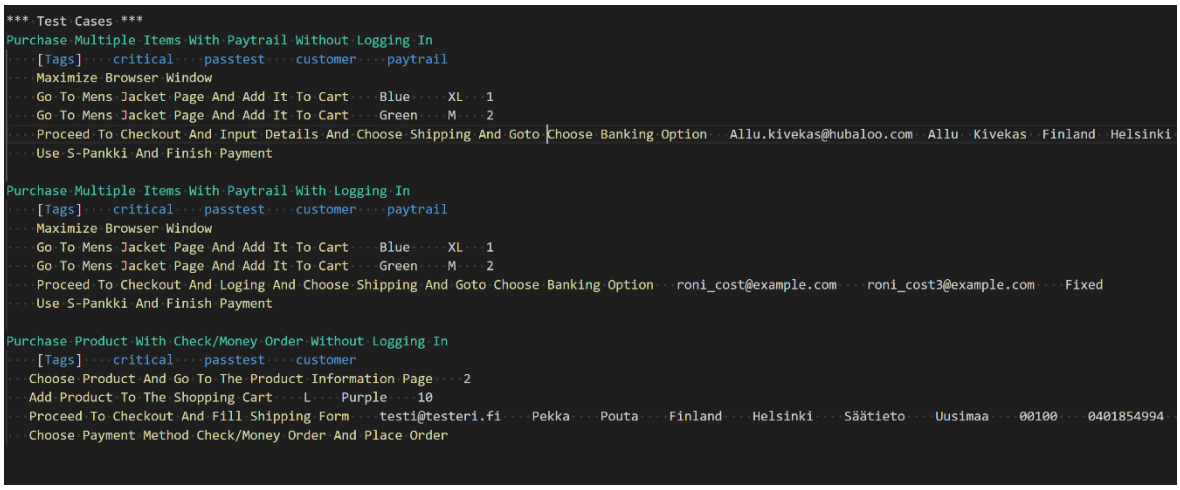


Kuva 7. Git – CI – CD prosessi (mukaillen Howtoforge.com, 2020.)

3.2 Robot Framework

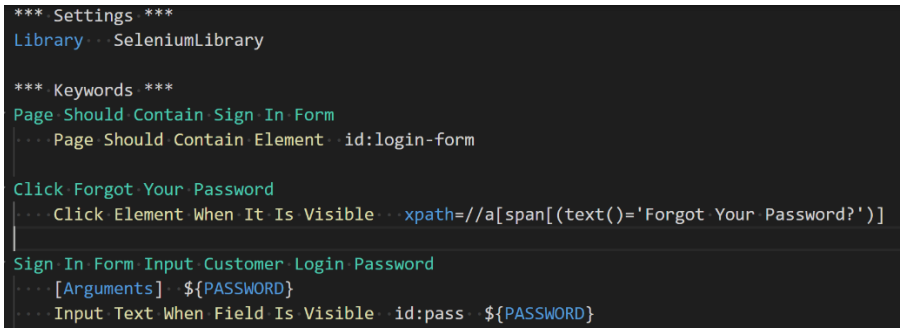
Robot Framework eli RF on testaustyökalu, jota käytetään muun muassa testauksen automatisointiin. Robot Framework on helppolukuinen testaustyökalu, jolla voidaan kehittää jopa pitkiä ja monimutkaisia testejä. RF:n kehittäjä on itseasiassa suomalainen mies nimeltä Pekka Klärck. RF on kehitetty Python ohjelmointikielellä.

Robot Frameworkin yksi käytetyimmistä ominaisuuksista on testiautomaatio. Sillä voi tehdä esimerkiksi nettisivuille tai mobiilisovellukselle testejä, mitkä suoriutuvat automaattisesti ja samalla raportoivat tuloksia niistä. Tämä on erittäin hyödyllistä siinä kohtaa, kun ollaan kehittämässä jotain uutta järjestelmää tai sovellusta. Tällaisella automatisoinnilla voidaan esimerkiksi tehostaa sovelluskehitystä siten, että kehittää RF:llä kasan niin sanottuja ”porttitestejä” ja ajaa ne testit automaattisesti ennen kuin versionhallinnassa olevat muutokset yhdistetään tuotantokoodiin. Tällä toimenpiteellä estetään esim. kehittäjiä muuttamasta tiettyjä vaatimustenmäärittelyjen mukaisia ominaisuuksia. RF:ssä käytetään termiä ”avainsana”, kuvamaan testin yksittäistä vaihetta. Avainsanoja voi luoda itse ja niitä voi yhdistellä keskenään yhdeksi avainsanaksi omien tarpeiden mukaisesti. RF:ssä on olemassa jo valmiina kirjastoja, auttamaan käyttäjiä eri tarpeisiin. Yleisin on RF:n oma sisäänrakennettu kirjasto, millä voi suorittaa yleisimpiä toimintoja.



Kuva 8. Robot Framework syntaksi.

Kuvassa 8 näkyy oma esimerkkini Robot Framework syntaksista. Kuten aikaisemmin mainittiin, se on erittäin helppolukuista ja selkeää. Kuvassa näytetään esimerkillä, miten voidaan suorittaa automatisoitua nettisivujen testausta. Siinä on omat avainsanat käytössä ja ne on muodostettu, yhdistelemällä eri avainsanoja, mitkä voidaan kokea yhdeksi suuremmaksi kokonaisuudeksi esimerkiksi ”Go To Mens Jacket Page And Add It To Cart”. Tämä avainsana on koottu pienimmistä avainsanoista, millä pyritään pitää testisyntaksi siistinä ja helppolukuisena. Nettisivupohjaisessa testauksessa, RF tarjoaa erittäin hyvää kirjastoa nimeltä ”SeleniumLibrary”. Tällä kirjastolla on erittäin hyviä avainsanoja valmiina käytettäväksi esimerkiksi ”Avaa Linkki” tai ”Paina nappia”. Samankaltaisia kirjastoja RF:lle on mahdollista kehittää itse, mikä onkin itseasiassa aika yleistä, sillä aina ei välttämättä aivan sinun tarpeisiisi sopivaa kirjastoa valmiina.



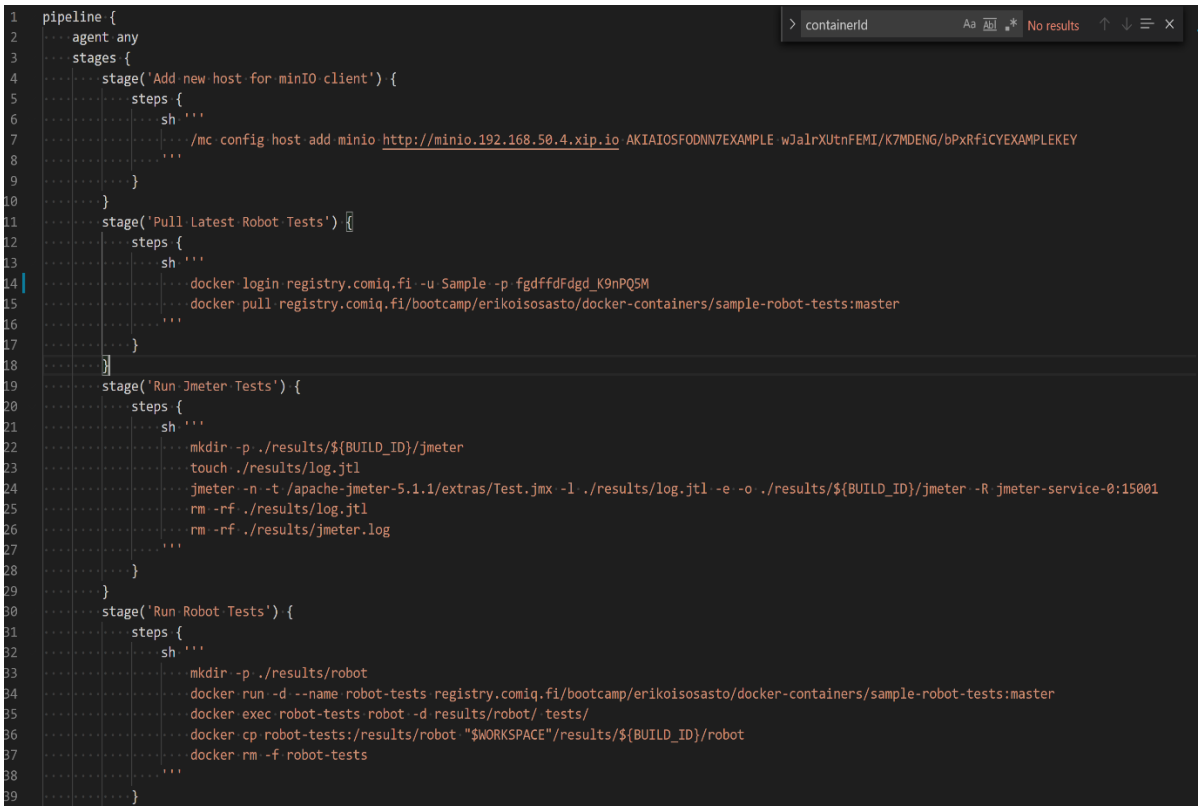
Kuva 9. SeleniumLibrary avainsanoja

3.3 Jenkins

Jenkins on automatisointityökalu, jota voidaan käyttää esimerkiksi lähdekoodin paketointiin, testien ajamiseen tai julkaisemiseen. Tätä työkalua konfiguroimalla, voidaan suorittaa erittäin tarkkaan ajoitettuja ja määriteltyjä toimenpiteitä. Jenkinsiin rakennetaan niin sanottuja ”putkistoja”, suorittamaan niille tarkoitettuja tehtäviä. Nämä tehtävät ovat yleensä aikaisemmin mainitut testaus, paketointi ja julkaiseminen. Toisin sanoen, Jenkins on siis työkalu, jota konfiguroimalla, voidaan automatisoida erilaisia prosesseja suoriutumaan tietyin väliajoin.

Jenkins asennetaan haluamallesi palvelimelle ja konfiguroidaan haluamallasi tavalla. Sille voidaan määrittää esimerkiksi git etäarkisto, minkä kanssa se on yhteydessä. Aina kun kyseiseen git arkistoon työnnetään muutoksia, se käynnistää haluamasi Jenkins putkiston. Tähän kohtaan CI/CD prosessia, voidaan esimerkiksi lisätä ”porttitestejä”, mitkä testaavat kyseisen työnnetyn muutoksen ja raportoivat sitten tulosten mukaisesti. Putkistojen asetuksia voi säädellä omien tarpeiden ja halujen mukaisesti, miten haluaa. Yksi yleinen konfigurointi putkistoille on ehto, jolloin putkiston toiminnot suoritetaan. Sen voi säätää esimerkiksi joka päivä alkamaan klo 23:00 illalla tai vaikkapa joka viikonloppu. Useat ohjelmistokehitysfirmat haluavat testata heidän ohjelmistoaan joka yö automaatiotesteillään varmistaakseen koodin laadun ja vakauden. Tähän tilanteeseen heillä on yölliset robot Frameworkillä tehdyt automaatiotestit, jotka pyörivät joka yö, raportoiden tuloksia jatkuvasti. Putkistoihin saa halutessaan Robot Framework -kytköksen, jolla RF testien tuloksia ja kestoja saa helposti ja selkeästi esiin Jenkinsin selainnäkymässä.

Putkistojen tehtäviä voidaan myös määritellä omassa tiedostossa nimeämällä sen tiedoston ”Jenkinsfileksi”. Jenkins putkiston ollessa yhteydessä git etäarkistoon, se paikantaa tämän kyseisen jenkinstiedoston siitä arkistosta ja suorittaa siinä lueteltavat asiat. Tämä tiedosto on erittäin hyödyllinen, kun halutaan määritellä näitä putkistoja erittäin yksityiskohtaisesti ja nimenomaan arkistokohtaisesti. Tässä tapauksessa voi olla esimerkiksi oma Jenkins putkisto sovelluksen lähdekoodin julkaisemiselle ja oma putkisto itse testeille.



Kuva 12. Jenkinstiedoston syntaksi.

Kuvassa 12 näkyy eräs omista jenkinstiedostoistani. Tässä suoritetaan muutama eri kokonaisuus automaatiotestejä, mukaan lukien Robot Framework testejä. Sen jälkeen, mitä kuvassa ei näy, tulokset ja raportit tallennetaan MinIO -nimiseen varastointipalveluun.

3.4 Ansible

Ansible on työkalu, millä voidaan järjestää, hallita ja konfiguroida muun muassa palvelimia ja niiden sisällä olevia järjestelmiä. Se on osana ”Infrastruktuuri koodina” käsitettä eli tietoteknisten järjestelmien infrastruktuuri kirjoitettuna koodiksi. IaC eli infrastruktuuri koodina tarkoittaa korkeatasoista kuvaavaa ohjelmointikieltä, millä automatisoidaan it-infrastruktuurin järjestely. Ennen tällaista ratkaisua, kehittäjien tai muiden tietoteknisten ihmisten täytyi itse omin käsin konfiguroida ja hallita esim. yrityksen omia fyysisiä palvelimia. Pahimmassa tapauksessa niitä on kymmeniä tai jopa satoja. Ansiblen tai muiden järjestelytyökalujen ansiosta, voidaan automaattisesti järjestää, hallita ja julkaista muun muassa palvelimia, käyttöjärjestelmä ja tietokantayhteyksiä haluamaansa tarkoitukseen. (Ibm.com, 2019.)

Ansible on helppolukuista ja kuvaavaa ”infrastruktuuri koodina” -kieltä. Sitä kirjoitetaan yksinkertaisella ”YAML” -konfigurointi kielellä. Yksinkertaistettuna Ansible toimii siten, että pelikirjoja määrittelemällä yaml-tiedostoihin, voi kertoa Ansiblelle mitä tehdä. Pelikirjaan

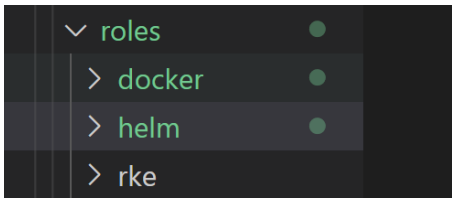
voi määritellä muuttujia, vaiheita ja periaatteessa kaikki tarvittava tieto, mitä Ansible tarvitsee konfiguroidakseen esimerkiksi palvelimia. Kuvassa 13 on esimerkkinä omasta Ansible pelikirjasta, missä on muutama tehtävä.

```
---
- hosts: all
  become: true
  become_user: root

  tasks:
  - name: Add Helm to path
    shell: echo "PATH=$PATH:/usr/bin/helm/linux-amd64" > /etc/environment
    become: true

  - name: Ensure vagrant user is in docker group
    user:
      name: "vagrant"
      groups: docker
      append: yes

  - name: Create symbolic link for config
    file:
      src: /usr/local/rke_d/kube_config_cluster.yml
      dest: /home/vagrant/.kube/config
      state: link
```

Kuva 13. Ansible pelikirja.

Ansiblessa tehtävien määrittelyyn käytetään sen omia moduuleja. Moduulit on kehitetty suorittamaan niille määriteltyjä tehtäviä. Moduulin nimi näkyy tehtävän nimen alapuolella. Kuvan 13 esimerkissä on käytetty moduuleja: ”shell”, ”user” ja ”file”. Näiden avulla Ansible-tehtävien kirjoittaminen on tehty helpommaksi ja yksinkertaisemmaksi.

Kun pelikirjasta alkaa tulla liian suuri, ja sen niin sanottu helppolukuisuus katoaa, on tehtävät hyvä jakaa erilaisiin rooleihin. Yleensä tapana on jakaa tehtävät niiden yhdistävien tekijöiden perustella. Tällä tehdään pelikirjasta luettavan näköinen ja itse tehtävät löytyvät muista tiedostoista. Kuva 14 näyttää roolien kansiorakenteen Ansiblessa ja kuva 15 osoittaa, kuinka rooleja kutsutaan itse pelikirjassa.



Kuva 14.

```
---
- hosts: all
  become: true
  become_user: root

  roles:
  - docker
  - rke
  - helm
```

Kuva 15.

Suurten järjestelmäkokonaisuuksien julkaiseminen ja hallinnointi tapahtuu yleensä ”mestari – kohde” menetelmällä. Tämä tarkoittaa käytännössä sitä, että käytössä on yksi palvelin, mikä toimii tässä tapauksessa mestarina, ja se kontrolloi etäältä näitä kohdepalvelimia. Tämä toiminta tapa edellyttää Ansiblen asentamisen vain mestari -koneeseen ja sillä sitten voi hoitaa suurempiakin kokonaisuuksia. Kohdepalvelimiin ei tarvitse asentaa Ansiblea ollenkaan. Näitä kohteita voidaan jakaa omiin ryhmiin ja niitä hallinnoidan erikseen omilla toiminnoilla. Esimerkiksi, kun testaus- ja tuotanto palvelimille halutaan tehdä erilaisia konfiguraatioita, niitä olisi hyvä pitää omissa ryhmissä. Etäkonfigurointi tapahtuu SSH:n (secure shell) avulla. Tämän avulla voidaan ottaa yhteyttää palvelimiin etäältä ja suorittaa niissä toimenpiteitä.

Ansiblen rooli DevOpsissa on selkeä. Sillä voi hallita CI/CD järjestelmien infrastruktuuria automaattisesti. Sillä voidaan myös takaa, että jokainen Ansiblella julkaistu ympäristö, oli se sitten testaus- tai tuotantoympäristö, on tismalleen samanlainen. Tällä minimoidaan tietotekniseen ympäristöön liittyviä yhteensopimattomuuksia.

3.5 Docker

Docker on virtualisointityökalu, millä voidaan eristää ennalta määriteltyjä ympäristöjä erilaisiin käyttötarkoituksiin. It-alalla virtualisointi tarkoittaa eri komponenttien luominen ohjelmoinnilla eli niiden virtualisointi. Yleisiä virtualisointeja on tietokoneet, tallennustilat ja verkot. Esimerkiksi virtualisoidut tietokoneet oven toiminnallisesta näkökulmasta aivan kuin ”normaalit” tietokoneet, mutta niillä ei ole varsinaisesti omaa fyysistä laitteistoa. Virtualisointi itsessään on luonut aivan mahdollisuuden rakentaa suuria ja ennen aikaan paljon fyysistä laitteistoa vaativia kokonaisuuksia. Jos tämän ajatuksen yksinkertaistaa, niin esimerkiksi yhden fyysisen palvelimen sisällä voi olla monta toisistaan riippumattomia virtualisoitua tietokoneita, suorittamassa niille määritettyjä prosesseja.

Dockerin tapauksessa luodaan käyttöjärjestelmätason virtualisoituja ns. ohjelmistopaketteja, mitkä ovat halutessasi täysin eristettyjä isäntäkoneesta. Isäntäkone tarkoittaa siis sitä fyysistä konetta, mikä isännöi tällaista virtualisoitua ohjelmistoa. Dockerin avulla luotuja virtuaalisia ohjelmistopaketteja kutsutaan konteiksi. Konttiteknologia perustuu siihen, että kontti luodaan yhdelle prosessille kerrallaan. Kontti siis luodaan omalla käyttöjärjestelmällä, omilla riippuvuuksilla ja halutessasi omaan virtualisoituun verkkoon. Tiettyyn pisteeseen saakka, kontit siis muistuttavat virtualisoitua tietokonetta, mutta niissä on paljon eroja kuitenkin. Virtuaaliset tietokoneet käynnistäessä, ovat päällä niin kauan kuin sinä

itse päätät sulkea sen, kun taas kontit käynnistäessä ovat päällä vain sen prosessin ajan, minkä olet sille ennakkoon määrittänyt. Kun kyseinen prosessi loppuu, kontti havainnollisesti tuhoutuu välittömästi. Prosessi voi olla mikä tahansa tietotekninen suoritus, jos vain ympärillä olevat riippuvuudet sen sallivat. Konteissa voi siis pyöriä vaikkapa nettisivut tai automaatiotestit. Automaatiotestikontti olisi käynnissä siihen asti, kunnes testit ovat ohi. Nettisivujen kohdalla, kontti olisi käynnissä siihen asti, kunnes esim. nettisivut kaatuisivat tai jotain muuta tapahtuisi mikä lopettaisi tämän prosessin. Kontit ovat yksinkertaisia, kevyitä, helposti liikutettavia ja uudelleenkäynnistettäviä virtuaalisia ympäristöjä erilaisiin käyttötarkoituksiin.

Kontit luovat siis sen hyödyn, että sinulla voi olla monta eristettyä, toisistaan riippumattomia ympäristöjä, suorittamassa erilaisia tai samanlaisia prosesseja, yhdellä fyysisellä tietokoneella tai palvelimella. Kontit ovat helposti määriteltävissä ja muokattavissa. Niitä on helppo lisätä ja helppo poistaa toiminnasta. Siihen nähden mitä ne pitävät sisällän, muun muassa käyttöjärjestelmän ja muut riippuvuudet, ne ovat erittäin vähä muistia ja tallennustilaa vieviä. Eli allokoimalla isäntäkoneen muistia ja tallennustilaa oikein, voi kontteja käyttää suuria määriä samaankin aikaan.

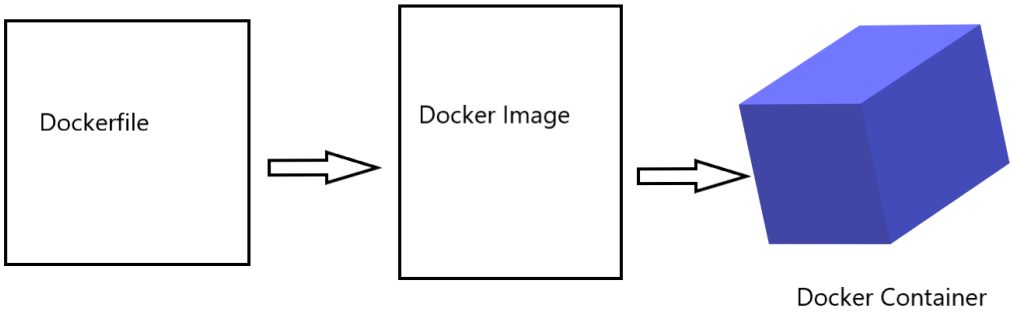
Dockerkontti luodaan siis määrittelemällä ensimmäiseksi dockertiedosto nimellä ”Dockerfile”. Kuva 16 on esimerkki omasta dockertiedostostani, millä voin käynnistää Nginx-palvelimen kontissa. Nginx on web-palvelin, minkä avulla voit esim. isännöidä nettisivuja.

```
# Operating system
FROM ubuntu:latest

# Install and configure nginx
RUN apt-get install -y nginx
RUN echo "daemon off;" >> /etc/nginx/nginx.conf

# Copy entrypoint script to container and execute it
COPY entrypoint.sh /usr/local/bin/
RUN chmod ugo+x /usr/local/bin/entrypoint.sh
ENTRYPOINT ["entrypoint.sh"]
```

Kuva 16. Dockertiedosto

Dockertiedosto on niin sanotusti ”resepti” kontin luomiseen. Tiedostossa määritetään muun muassa käyttöjärjestelmä, minkä päälle kaikki muu tulee. Kuvassa 16 näkyykin, että käytössä on Ubuntu-käyttöjärjestelmä, mikä on yksi Linuxin suosituimmista käyttöjärjestelmäjakeluista. Seuraavaksi Ubuntussa käytettävällä paketinhallinta työkalulla asennetaan Nginx palvelin ja sitä konfiguroidaan muokkaamalla tiettyä tiedostoa. Kuten aikaisemmin mainitsin, kontit tarvitsevat jonkun prosessin suoritettavaksi. Se määritellään dockertiedostossa sisääntulopisteeksi eli ”Entrypointiksi”. Kuvan 16 dockertiedostossa kopioidaan ”entrypoint.sh” niminen tiedosto isäntäkoneesta konttiin ja annetaan käyttäjälle oikeudet suorittaa kyseinen tiedosto. Tiedosto voi sisältää mitä vaan haluat suoriutuvan, oli se sitten automaatiotestit tai Nginx palvelin. Rivillä 11 määritetään kontin sisääntulopiste, ja se siis suoriutuu, kun kontti käynnistetään.

Kontin luomisprosessi kulkee siis näin: Dockertiedostosta luodaan dockerkuvake ja tästä kuvakkeesta sitten käynnistetään itse kontti.



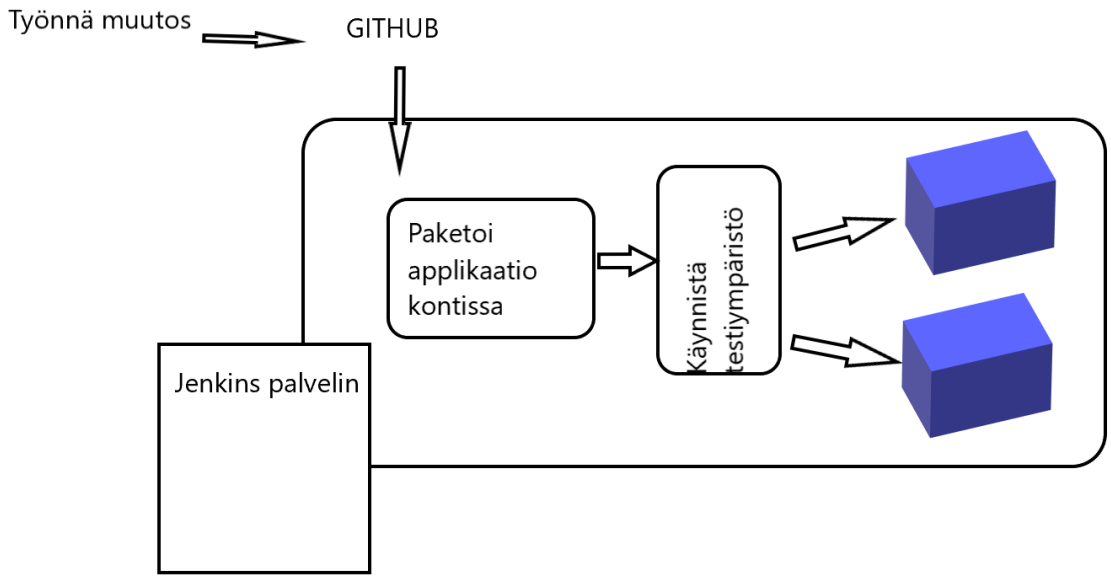
Kuva 17. Dockerkontin luominen (mukaillen Medium.com, 2020)

Kuva 17 osoittaa siis, kuinka dockertiedostosta rakennetaan dockerkuvake ja se sitten käynnistetään kontiksi. Tämä kuvakkeen rakennusvaihe sisältää toimenpiteet, mitä dockertiedostossa on määritelty. Dockertiedoston sisällöstä ja pituudesta riippuen, kuvakkeen rakentamiseen voi mennä muutamista kymmensekunneista useaan minuuttiin. Kuvakkeen valmistuttua, se voidaan vain yhdellä komennolla käynnistää yhdeksi tai useammaksi kontiksi.

Yksi tärkeä ominaisuus konttiteknologiassa on kiinnitys. Kontithan tuhoutuvat prosessi loputtua ja sen myötä kaikki sisältö katoaa kuten esimerkiksi lokitiedostot ja vaikkapa testitulokset ja -raportit. Yksi tapa ratkaista tämä, on kiinnittää konttiin kansio isäntäkoneesta. Tässä tapauksessa, kun kansiosta lisätään, muokataan tai poistetaan, se tapahtuu myös isäntäkoneessa. Esimerkiksi automaatiotestikontissa haluttaisiin kiinnittää kansio isäntä-

koneesta konttiin, ja testien loputtua testiraportit ja -tulokset tallennettaisiin tähän kiinnitettyyn kansioon ja sitä mukaa ne ovat saatavilla myös isäntäkoneessa. Tämän jälkeen, vaikka kontti tuhoutuu ja hävittää kaikki sisällään olevan tavaran, kiinnitystiedostossa oleva sisältö löytyy vielä isäntäkoneessa.

Dockerin käyttö CI/CD putkistoissa on erittäin yleistä. Sillä voi luoda nopeasti eristettyjä ja uudelleenkäytettäviä ympäristöjä moneen eri tarkoitukseen kuten lähdekoodin paketointiin tai testaamiseen. Kuva 18 havainnollistaa, kuinka CI järjestelmässä testaan muutoksia versionhallinnasta, dockerkonttien avulla.



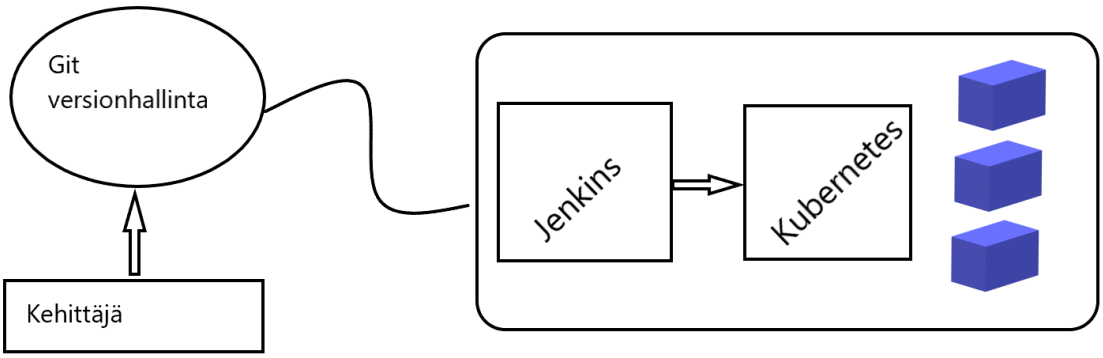
Kuva 18. Docker osana testausta CI putkistossa (mukaillen Rancher.com, 2020.)

3.6 Kubernetes

Kubernetes on konttien orkestrointityökalu. Sen avulla voidaan yhdistellä kontteja ja hallita niitä yhtenä isona klusterina.

Sillä voi julkaista ja hallita jopa suurempia ja haastavimpia konttikokonaisuuksia. Tällaisien konttikokonaisuuksien ei tarvitsisi sijaita vain yhdellä fyysisellä palvelimella, vaan kubernetes klusterin voi muodostaa useasta eri fyysisestä palvelimesta ja silti aivan kykeneväisiä kommunikoimaan keskenään. Sillä voi kontrolloida ja automatisoida applikaatioiden päivi-

tyksiä ja julkaisuja. Kontteja ja niiden resursseja voidaan skaalata tarpeen mukaan applikaation ajoaikana. Esimerkiksi jos isännöi nettisivuja kubernetes klusterissa, nettisivut skaalautuvat verkkoliikenteen mukaisesti. Jos kävijöitä lisääntyy huomattavasti, osaa kubernetes lisätä kontteja ja resursseja, jotta sivut eivät ruuhkautuisi ja pahimmassa tapauksessa lakkaisi toimimasta. Kuberneteksessä on myös seuraavanlaisia ominaisuuksia: terveystarkastukset, itsekorjaus, automaattisia uudelleenkäynnistyksiä, replikointia ja skaalausta. Tässä tapauksessa terveystarkastus tarkoittaa sitä, että kubernetes osaa itse tarkistaa, onko esim. applikaatio toiminnassa ja terve. (Redhat.com, 2020.)



Kuva 19. Kubernetes CI/CD putkistossa (mukaillen Medium.com, 2019.)

Kehittäjistä alkaen, muutos lähtee versionhallintaan. Sen jälkeen CI putken kautta se päätyy kubernetes klusteriin, missä uusin muutos julkaistaan. Kuva 19 osoittaa kuinka kubernetes klusteri on ympäristö, missä applikaatio pyörii ja mihin kaikki muutokset julkaistaan.

3.7 Terraform

Terraform on ”infrastruktuuri koodina” työkalu, jota automatisoidaan fyysisten tai virtuaalisten infrastruktuurien luonti. Se käyttää apunaan kuvaavia Terraform tiedostoja luodakseen ja päivittääkseen näitä ympäristöjä. Tiedostoihin kuvaillaan resursseja, mitä halutaan julkaista eli esimerkiksi palvelimia tai vaikkapa verkkoon liittyviä komponentteja.

Terraform -koodin ajo on helposti automatisoitavissa, ja se on aina uudelleenajettavissa. Se tallentaa aina ajon jälkeen infrastruktuurin niin sanotun ”tilan”, joten aina kun se ajetaan uudelleen, se varmistaa, että halutut resurssit löytyvät kohdeympäristöstä. Koodin uudelleenajo ei siis luo kaikkia haluttuja resursseja uudelleen, vaan varmistaa että ne ovat

olemassa. Luonnollisesti uudelleenajojen aikana, jos Terraform huomaa resurssien puuttuvan, se luo ne automaattisesti.

Terraformia käytetään paljon pilvipalvelualustoilla esimerkiksi Amazon Web Servicessä tai Microsoft Azuressa. Pilviympäristöissä on helppo luoda lisää ja poistaa vanhaa, sillä ympäristöt ovat virtuaalisia eivätkä fyysisiä. Tämä kuitenkaan ei tarkoita sitä, että Terraformilla ei voisi fyysisiä palvelimia konfiguroida. Moni pilvipalvelu tukee Terraformin käyttöä.

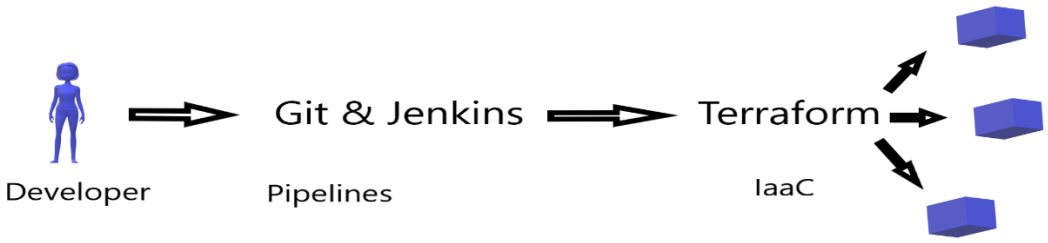
```
provider "aws" {
  region = "us-east-1"
}

resource "aws_internet_gateway" "gw" {
  vpc_id = aws_vpc.default.id
}

resource "aws_subnet" "public-subnet" {
  vpc_id     = aws_vpc.default.id
  cidr_block = "10.0.1.0/24"

  tags = {
    Name = "Main"
  }
}

resource "aws_instance" "web-server" {
  ami           = "ami-0bb3fad3c0286ebd5"
  instance_type = "t3.micro"
  subnet_id     = aws_subnet.public-subnet.id

  tags = {
    Name = "Web-Server"
  }
}
```

Kuva 20. Terraform koodia

Kuvassa 20 näkyy esimerkki Terraform koodista. Siinä käytetään Amazon Web Services alustaa kohteena rakentaa haluttu infrastruktuuri. Ensimmäiset kaksi resurssia on verkkoon ja yhteyksiin liittyviä ja kolmantena on web-palvelin, mikä siten luodaan ylempänä luotuun aliverkkoon. Tämän koodin, kun ajaa useaan kertaa, Terraform vain varmistaa, että vain kyseiset resurssit ovat olemassa halutussa ympäristössä.

Terraformin rooli DevOpsissa ei ole välttämättä oleellinen, mutta erittäin hyödyllinen. Suurten CI/CD putkistojen ympäristöt on luotava jossain vaiheessa, ja mikä sen turvallisempaa ja vakaampaa, kuin näiden infrastruktuurien luonti on automatisoitu ja dokumentoitu. Terraform koodi voi toimia siis jonkinlaisena dokumentaationa infrastruktuurille, sillä

siinä näkyy kaikki käytössä olevat osat. Ilman Terraform koodia, näitä infrastruktuureja joutuisi luoda käsin aina uudestaan.



Kuva 21. Terraform koodi osana CI putkea

4 CI/CD toiminta yrityksessä

4.1 Tutkimuskysymykset

Tutkimuksen tavoitteena oli selvittää CI/CD putkistojen käyttötarkoituksia ja niiden vaikutusta yrityksiin. Putkistojen käyttötarkoitukset tulivat ilmi kappaleessa kaksi ja kolme, missä selitettiin niiden teknisiä rakenteita ja kuinka niitä sovelletaan yrityksissä. Kysymys kuuluu, kuinka nämä CI/CD järjestelmät ovat vaikuttaneet nykypäivänä yrityksiin ja millaista oli ennen niiden käyttöä. Aineistona toimii internetistä löydetyt artikkelit aiheesta, tiedot omasta kokemuksesta sekä konkreettisempi esimerkki omasta vuoden mittaisesta projektista erään suuren kansainvälisen asiakkaan kanssa.

Facebookin kukoistaessaan 2011 luvulla, se päätti julkaista uusia ominaisuuksia internetsivuilleen. He julkaisivat sen maailmanlaajuisesti samalla kertaa eli puskivat sen suoraan kaikille tuotantopalvelimilleen samaan aikaan. Facebook ei kuitenkaan osannut odottaa sitä suosion määrää mitä uuden ominaisuudet aiheuttivat ihmisissä. Tämä aiheutti kaatumista heidän palvelimillaan. Tämän sijaan he olisivat voineet julkaista jatkuvasti pieniä määriä ja näin he olisivat huomanneet suosion nousun hyvissä ajoin. Tämän avulla he olisivat voineet reagoida tähän ja vaikkapa lisätä palvelimia tuotantoon jakaakseen tätä kuormaa. Palvelinten kaatumisen takia, he eivät myös saaneet hyvin palautetta uusista ominaisuuksista. (medium.com, 2019.)

Riskialttiita osa-alueita tuotekehityksissä voi olla useita. Esimerkiksi tuotanto- ja kehitysympäristöjen luonti ja ylläpito. On tärkeää kehittää tuotetta mahdollisimman samankaltaisessa, ellei jopa identtisessä ympäristössä, kuin tuotantoympäristö. Tai vaikkapa uuden koodin integroiminen olemassa olevaan koodiin, täytyy tapahtua ilman että siitä aiheutuu mitään ongelmia tuotteelle. Nämä osa-alueet ovat tärkeitä yrityksen toiminnan kannalta. Kun näitä osa-alueita ei automatisoida tai dokumentoida hyvin, voi pidemmällä aikavälillä seurata ongelmia. Sanotaan että eräällä mobiiliapplikaatioyrityksellä olisi vuosikausia ollut vain yksi palvelin tuotantokäytössä. Huomataan, että viime aikoina käyttäjämäärä on noussut hurjasti ja nyt palvelimien määrää täytyy nostaa, jotta enempi käyttäjä voisi käyttää heidän tuotettansa. Jos ensimmäinen palvelin on pystytetty manuaalisesti eikä sitä olla dokumentoitu hyvin, on vaikea pystyttää seuraava toimimaan täysin identtisesti. Ja syy miksi se on tärkeä luoda ne identtisiksi, on se, että heillä on jo olemassa oleva ja nimenomaan toimiva tuotantoympäristö. Olisi turha riski, alkaa pystyttää toista hieman erilaista ympäristöä samalle asialle. Tällaisen riskin saisi poistettua sillä, että luo esimerkiksi Terraform skriptit ja siihen perään Ansible konfigurointia. Tällä varmistutaan siitä, että aina kun pystytetään uusia ympäristöjä, ne tulee olemaan identtisiä.

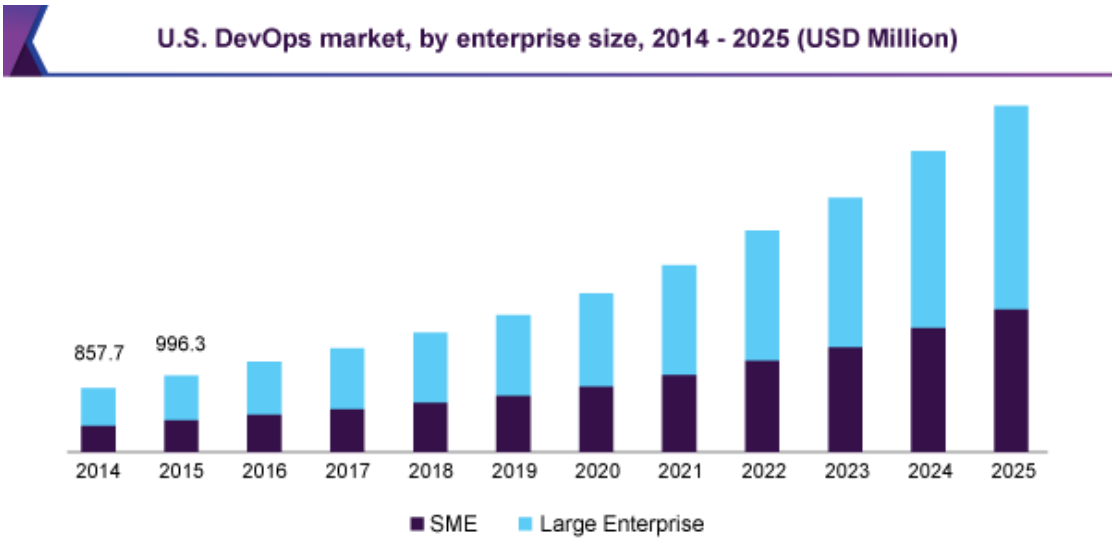
Yritykset ovat huomanneet nämä riskit ja ymmärtäneet niiden mahdolliset vaikutukset. Tänä päivänä monella yrityksellä, koosta ja resursseista riippuen, saattaa olla omistetun tiimit tällaisille CI/CD ja muihin DevOpsiin liittyviin asioihin. He varmistavat sen, että asiat kulkevat eteenpäin sulavasti aiheuttaen mahdollisimman vähän ongelmia matkassa. Toimivien CI/CD putkistojen avulla, niin sanottujen ”puhtaiden” kehittäjien ei tarvitse huolehtia asioista, jotka ei varsinaisesti liity kyseiseen tuotekehitykseen, esimerkiksi julkaisut. Yritykset todella panostavat tähän ja viime vuosikymmenen aikana siitä on alkanut kasvamaan isompi trendi alalla.

Moni yritys käyttää DevOps menetelmiä niiden toiminnassaan ja sen tulokset toiminallisuuksissa huomataan. Amazonin siirtyminen pilveen ja erilaisten tietoteknisten palvelujen virtualisointi mullisti muiden lisäksi myös DevOps maailmaa, sillä nyt melkein kaiken pystyi tehdä pilvessä eikä fyysisiä resursseja tarvittu paljon.

DevOpsin arvo liiketoiminnassa on huomattavaa. DevOps käytäntöjen ja työprosessien implementointi omaan liiketoimintaan säästää yritykselle aikaa ja rahaa, auttaa ennustamaan tuotekehityksen kulkua ja ylläpitää työntekijöiden motivaatiota. (Dzone.com, 2019.)

Adidas julkaisi merkkikengän mikä valmistettiin yhdessä amerikkalaisen artistin Kanye Westin kanssa vuonna 2015. Christopher Null kertoo artikkelissaan, kuinka kenkiä myytiin yli 2600 vain viidessä sekunnissa ja kuinka Adidas sitten hoiti tämän. Alussa se loi valtavasti ongelmia it-osastolle, sillä heillä ei eivät osanneet ennustaa tätä, ja sitä kautta heillä ei ollut myöskään tarvittavia resursseja selviytymään tällaisista käänteistä. Pahimpina aikoina, ruuhkien ja ylikuormituksien takia, kehittäjillä saattoi mennä jopa viikko käynnistää yksi virtuaalikone. Tämän jälkeen Adidas koki suuren muutoksen it-osastollaan, ja siirtyi käyttämään pilvipalvelu-pohjaista arkkitehtuuria. Käyttöön otettiin Kubernetes ohjaamaan kontteja ja muita DevOps menetelmiä. Tämän muutoksen jälkeen Adidas pääsi kuuden viikon tuotekehityksellisestä toimitussyklistä, viiteen toimitukseen yhdessä päivässä. (Techbeacon.com, 2020.)

Korkeatasoiset DevOps työntekijät ovat erittäin arvokkaita työmarkkinoilla tällä hetkellä. Sen huomaa kuinka paljon yrityksen niitä haluavat ja kuinka paljon ne ovat valmiita heille maksamaan. Tämä on täysin ymmärrettävää, sillä nuo asiat luovat yrityksille turvaa ja vakautta monessa tietoteknisessä asiassa ja se on taas suoraan yhteydessä yrityksen liiketoimintaan.



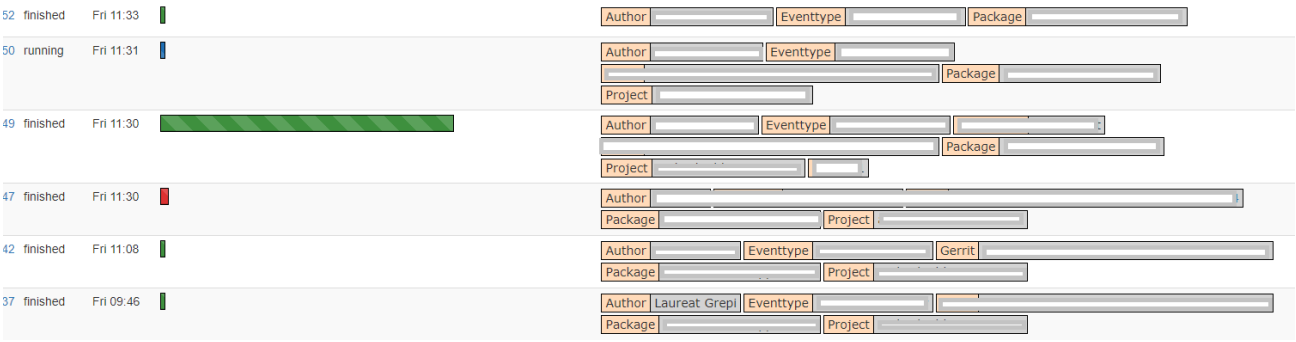
Kuva 22. DevOps-markkinat ennustettuna Yhdysvalloissa. (Grandviewresearch.com, 2018.)

4.2 Scheduler – Asiakasprojekti

Tämä kappale koskee omaa asiakasprojektiani, mikä kesti reilun vuoden verran. Tässä projektissa olin mukana sovelluskehityksessä ja erityisesti osana CI/CD tiimiä. Asiakas oli erittäin suuri, kansainvälinen ja tunnettu yritys. Salassapitovelvollisuuden vuoksi, viittaan yritykseen, projektiin ja muihin luottamuksellisiin tietoihin geneerisimmillä sanoilla. Scheduler on sen CI järjestelmän nimi, mitä tämä yritys käytti kyseisessä projektissaan. Scheduler pääsiassa on avointa koodia ja saatavilla GitHubista, mutta siihen integroidut osat, mitkä olivat oleellisia nimenomaan asiakkaalleni, ovat salaista tietoa.

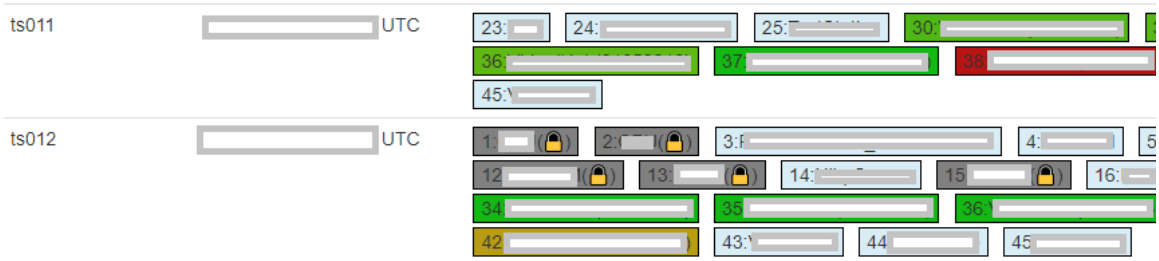
Yksinkertaistettuna, projektin tavoitteena oli tuottaa tietynlaisia fyysisiä laitteita ja niiden sisältämää ohjelmistoa. Projekti oli suuruudeltaan melko laaja ja käsitti paljon eri teknisiä osapuolia. Fyysisten laitteiden kehittämiselle oli omat tiimit, jotka hoitivat sen, kuten myös niihin liittyvää ohjelmistoa hoiti eri ihmiset. Kaiken tämän päälle oli erikseen testaukseen soveltuva tiimi, joka piti huolen testipuolesta ja muita tiimejä, jotka työskentelivät nimenomaan kehitysympäristön parissa ja sen ylläpitämisessä. Kuten aikaisemmin mainitsin, siellä oli erikseen CI/CD tiimi, joka hoiti siihen kuuluvat asiat kuten muun muassa Scheduler. CI/CD tiimin tarkoitus oli siis kehittää ja ylläpitää niitä koodeja ja ympäristöjä, mitkä piti huolen muun muassa uuden koodin jatkuvasta integroitumisesta, automaatiotestien pyörimisestä ja eri versioiden julkaisuista.

Projektissa oli kehitetty selkeät ja toimivat järjestelmät, mitkä pitivät huolen jatkuvan koodin integroimisesta ja julkaisemisesta. Asiat toimivat, aivan kuin normaalissa CI/CD järjestelmässä. Kun koodia muutettiin tai luotiin lisää, ennen kuin sen pysty integroimaan, se muutos paketoitiin, julkaistiin (vain siis kehitysympäristössä) ja testattiin.



Kuva 23. Schedulerin tehtävänäkymä.

Schedulerin tehtävänäkymässä näkyy, millaisia tehtäviä on ajoitettu suoritettavaksi, sekä lisätietoja niistä. Kuva 23 havainnollistaa siis jatkuvan koodin integroimisen ohella tapahtuvia testiajoja. Ennen muutosten testejä, muutos paketoidaan ja julkaistaan jotta se olisi saatavilla testiympäristössä.

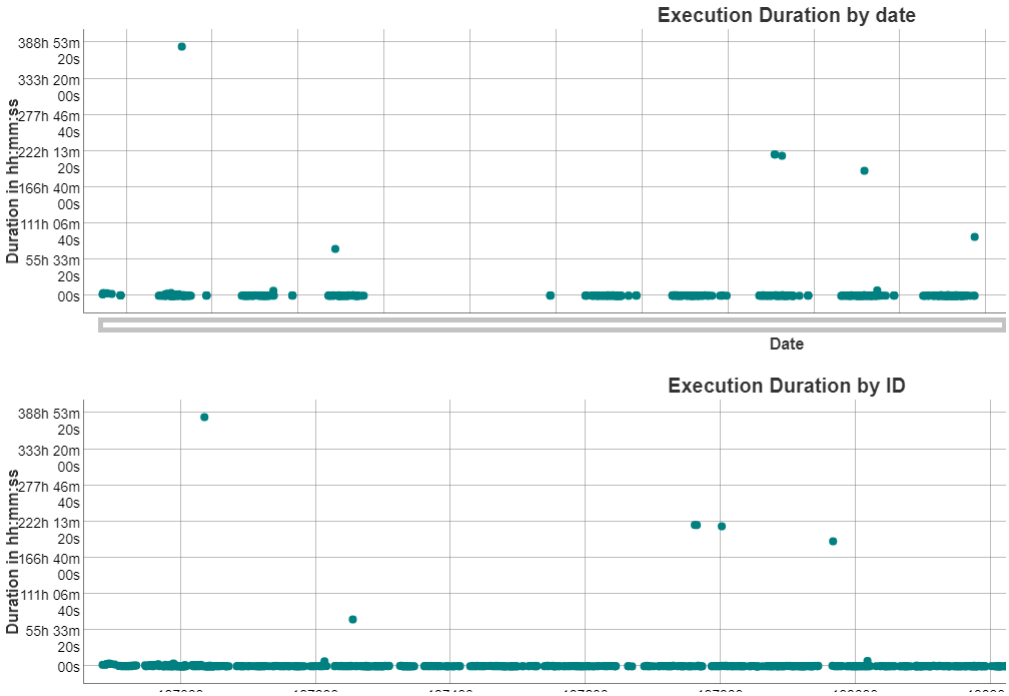


Kuva 24. Testiympäristöt ja niiden resurssit.

Kuvassa 24 näkyy muutama testiympäristö ja niissä saatavilla olevat fyysiset tai virtuaaliset resurssit. Riippuen testistä, siinä saatettiin tarvita useita erilaisia resursseja, ja Scheduler osasi siitä pitää huolen, että oikeat resurssit menivät oikealle testille. Schedulerin erikoisosaaminen on muun muassa siis osata allokoida nämä lukuisat resurssit oikein siten, että konflikteja niiden käytöstä ei tule. Tämä on siis Continuous Integration järjestelmä ja se toimii automaattisesti kaiken muun taustalla. Siksi on tärkeää, että resurssien allokoiminen onnistuu jatkuvasti ja resursseja käyttävät vain ne tehtävät, mitkä niitä tarvitsevat. Kuvasta näkyy kuinka resurssin värit vaihtelevat, riippuen niiden käytöstä ja tilasta.

Tällaisen järjestelmän avulla on ratkaistu lukemattomia ongelmia ja ennaltaehkäisty monia pullonkaulojakin. Tällaisen järjestelmän arvo asiakkaalleni oli äärimmäisen suuri, kuten se olisi ollut monelle muullekin yritykselle. Tämän avulla säästettiin paljon rahaa, aikaa sekä ihmisresursseja.

Kuva 25 näyttää kuinka Scheduler myös kerää statistiikkaa ajetuista tehtävistä. Tällaiset tilastot ovat tärkeitä CI/CD tiimille, sillä niissä näkyvät esimerkiksi järjestelmien pullonkaulat.



Kuva 25. Schedulerin statistiikkaa.

5 Pohdinta

Selkeästi CI/CD järjestelmät ovat olleet yrityksille hyödyllisiä. Ja yleistestikin DevOps on saanut paljon tuulta alleen viime vuosikymmenen aikana ja se näkyy jopa suurimpien yritysten toiminnassa. Nämä CI/CD järjestelmät ovat todistuneet erittäin arvokkaaksi yrityksille ja niiden toiminalleen. Koodia integroidaan ja testataan jatkuvasti sitä mukaa kun sitä kehitetään. Kaikki versiot paketoidaan, julkaistaan ja testataan automaattisesti. Yritysten liiketoiminnan kannalta, ei ole mitään syytä miksi tällaisia järjestelmiä ei haluttaisi omaan käyttöön.

It-alla olen huomannut sen, että kun ongelmakohtia tai pullonkauloja löytyy erilaisista toimintamenetelmistä, niihin melkein aina puututaan ja halutaan ratkaista. Mitä tämä kertoo sitten ajasta ennen CI/CD järjestelmiä? Onkohan ennen aikaan koodin integroiminen ja julkaiseminen nähty kenties hidasteena tai jopa ongelmana, kun sille ei ole ollut helppoja ja automaattisia ratkaisuja. Kun Facebookin kokoinen yritys vielä vuonna 2011 suoritti tuollaisia julkaisuja, millaisia ongelmia muilla yrityksillä on ollut ennen näitä DevOps menetelmien käyttöä. Olin hieman yllättynyt tästä Facebookiin liittyvästä tapahtumasta, ja kuinka päättivät julkaista se koko maailman käyttäjille samaan aikaan. Vuosi oli kuitenkin 2011 ja siihen aikoihin heillä oli jo suunnattoman paljon käyttäjiä. Nykypäivänä Facebook pyörii suurimmalta osalta pilvessä ja uskon, että samanlaisia virheitä he eivät tule toistamaan.

Heijastin omaa kokemustani alalta tähän tutkimukseen ja huomasin, että monet artikkelit ja statistiikka tukivat kuitenkin kokemuksiani. Tämä tutkimus on hyödyllinen niille yrityksille, jotka miettivät DevOpsin hyötyjä ja nimenomaan CI/CD näkökulmasta. Tästä tutkimuksesta ne saavat selville millaista arvoa voisi Continuous Integration ja Deployment tuottaa niille. Työkalujen esittelyssä, yritykset voivat peilata omia teknisiä tarpeitaan ja selvittää mitä he oikeasti tarvitsevat.

Lähteet

Atlassian. Git Branch. https://www.atlassian.com/git/tutorials/using-branches Luettu: 15.10.2020

Atlassian. Git Flow. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow. Luettu: 27.10.2020

Atlassian. What is Continuous Deployment?. https://www.atlassian.com/continuous-delivery/continuous-deployment. Luettu: 6.10.2020

Atlassian. What is Continuous Integration?. https://www.atlassian.com/continuous-delivery/continuous-integration. Luettu: 6.10.2020

Atlassian. What is DevOps?. https://www.atlassian.com/devops/what-is-devops Luettu: 10.6.2020

Bhagwat, Manoj. 29.6.2019. Kubernetes CI/CD using Jenkins on Google Cloud. https://medium.com/avmconsulting-blog/kubernetes-ci-cd-using-jenkins-on-google-cloud-5b10da6147a6

Dutta, Mainak. 2019, BEFORE AND AFTER OF DevOps: A PEEK INTO AGILE DevOps. https://medium.com/@mainakdutta76/before-and-after-of-devops-a-peek-into-agile-devops-3600c26129ac

Fedak, Vladimir. 2019. DevOps Business Value and Advantages. https://dzone.com/articles/devops-business-value-and-advantages

Git – CI – CD. https://www.howtoforge.com/how-to-set-up-gitlab-server-for-ci-cd-operation-on-centos/. Luettu: 27.10.2020

Github. Robot Framework plugin in Jenkins. https://github.com/jenkinsci/robot-plugin/blob/master/doc/images/robo_loglink_projectpage.png

Grand View Research. Development to Operations Market Size, Share & Trends Analysis Report. 3/2018. https://www.grandviewresearch.com/industry-analysis/development-to-operations-devops-market

IBM. What is Infrastructure as Code (IaC)?. https://www.ibm.com/cloud/learn/infrastructure-as-code. Luettu: 1.11.2020

Jayanandana, Nilesh. Build a Docker Image just like how you would configure a VM. 18.04.2018. https://medium.com/platformer-blog/practical-guide-on-writing-a-dockerfile-for-your-application-89376f88b3b5

Kulkarni, Saahas. 2019. Continuous Integration Best Practices. https://devon-blog.com/continuous-delivery/continuous-integration-best-practices/. Luettu: 10.6.2020

Kurniadi, Rori. 2018. The Journey Build Continuous Deployment. https://devon-blog.com/continuous-delivery/continuous-integration-best-practices/ Luettu: 10.6.2020

Null, Christopher. 10 companies killing it at DevOps in 2020. 10.08.2020. https://techbeacon.com/devops/10-companies-killing-it-devops-2020

Rancher. Docker-based build pipelines. https://rancher.com/docker-based-build-pipelines-part-1-continuous-integration-and-testing. Luettu 2.11.2020

RedHat. What is Kubernetes?. https://www.redhat.com/en/topics/containers/what-is-kubernetes. Luettu 2.11.2020

Trailhead. GitHub Repositories. https://trailhead.salesforce.com/en/content/learn/projects/develop-app-with-salesforce-cli-and-source-control/add-salesforce-dx-project-to-source-control. Luettu: 27.10.2020