VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Jussi Perälä

OHJELMISTORIIPPUVUUKSIEN AUTOMAATTINEN YLLÄPITO

Tekniikka
2023

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Jussi Perälä
Opinnäytetyön nimi	Ohjelmistoriippuvuuksien automaattinen ylläpito
Vuosi	2023
Kieli	suomi
Sivumäärä	32
Ohjaaja	Mikael Jakas

Tämän opinnäytetyön tarkoituksena on tutkia Renovatebotin käyttöönottoa ja räätälöintiä niin, että sitä voidaan hyödyntää ohjelmistokehitysprosessissa. Ensimmäinen ja tärkein tavoite on tutustua työkaluun, saada täysi käsitys sen ominaisuuksista ja sitten konfiguroida se niin, että se voidaan käyttöönottaa erilaisiin projekteihin.

Työssä aluksi tutustuttiin Renovatebotin toimintaan ja ominaisuuksiin. Tämän jälkeen tutkimme erilaisia konfigurointivaihtoehtoja, mukaan lukien konfiguraation periytyminen, oletustarkastajat ja tiettyjen hakemistojen huomioimatta jättäminen. Renovatebotin toimintaa testattiin tietovarastokopioissa, jonka jälkeen se otettiin käyttöön organisaation tietovarastoihin.

Opinnäytetyön tulokset korostavat automaatiotyökalun merkitystä nykyaikaisessa ohjelmistokehityksessä ja rohkaisevat senkaltaisten sovellusten käyttöönottoon ja räätälöintiin.

Avainsanat: automaatio, ohjelmistoriippuvuudet, ohjelmistokehitys, konfigurointi, Renovatebot

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Jussi Perälä
Title	Automatic Maintenance of Software Dependencies
Year	2023
Language	Finnish
Pages	32
Name of Supervisor	Mikael Jakas

The purpose of this thesis was to investigate the process of introducing Renovatebot and customizing it so that it may be utilized in the software development process. The first and most important goal was to become familiar with the tool, provide a full grasp of its features, and then configure it so that it can be used for a variety of projects.

To begin, the functionality of Renovatebot's various features and components were studied. After that, a variety of configuration choices, were looked into, including configuration inheritance, default reviewers, and ignoring particular directories, among others. After establishing Renovatebot abilities in forked repositories, it was finally enabled and started to operate on the primary repositories.

The findings of the thesis highlight the significance of automation tools in modern software development and provide encouragement for using these kinds of applications as well as their individual customization.

Keywords: automation, software dependencies, software development, configuration, Renovatebot

SISÄLLYS

LYHENTEET

CI Continous Integration, jatkuva integraatio

CD Continous Deployment, jatkuva julkaiseminen

CD Continous Delivery, jatkuva toimitus

CVCS Centralized Version Control System, keskitetty versionhallintajärjestelmä

DVCS Distributed Version Control System, hajautettu versionhallintajärjestelmä

1 JOHDANTO

Ohjelmistoriippuvuudet ovat tärkeä osa nykypäivän ohjelmistokehitystä. Riippuvuuksien käyttö nopeuttaa ja helpottaa kehitystyötä sen sijaan, että kehittäjät kirjoittaisivat kaiken koodin alusta itse.

Riippuvuuksien jatkuvasti kasvava määrä nykyaikaisissa ohjelmistoprojekteissa johtaa siihen, että ne jäävät helposti päivittämättä. Vanhentuneet riippuvuudet voivat aiheuttaa tietoturva-aukkoja, mitkä voivat johtaa tietomurtoihin ja muihin tietoturvahäiriöihin. Lisäksi vanhentuneet riippuvuudet voivat aiheuttaa suorituskykyongelmia.

Riippuvuuksien manuaalinen päivittäminen on aikaa vievää ja työlästä ohjelmistokehityksessä. Ongelman ratkaisemiseksi on kehitetty automaatiotyökaluja, joilla riippuvuuksien ylläpito ja päivittäminen on helppoa.

Renovatebot on yksi avoimeen lähdekoodiin perustuva ohjelmistoriippuvuuksien päivityksen automatisoimiseen tähtäävä työkalu, jonka käyttö on kasvattanut suosiota ohjelmistokehitysyhteisössä.

Tämän opinnäytetyön tarkoituksena on tutkia Renovatebotin käyttöönottoa ja käyttöä organisaatiossa sekä antaa kattava käsitys sen ominaisuuksista ja arvioida sen vaikutuksia ohjelmistokehitysprosessiin ja toimintaan. Insinöörityö toteutettiin Wärtsilä Oyj:lle. Wärtsilä on vuonna 1834 perustettu suomalainen konepajateollisuutta harjoittava pörssiyhtiö. /22/

2 OHJELMISTOPROJEKTIT

Ohjelmistoprojektit ovat vallanneet modernin digitaalisen maailman ja niiden avulla on rakennettu kaikki henkilökohtaisista laitteista kehittyneimpiin liiketoimintajärjestelmiin. Kun innovatiivisten ja tehokkaiden ohjelmistoratkaisujen kysyntä kasvaa jatkuvasti, vaikuttaa se ohjelmistoprojektien laajuuteen.

Ohjelmistokehityksen aikana käytetään erilaisia työkaluja, menetelmiä ja lähestymistapoja. Ohjelmistoprojektit koostuvat erilaisista komponenteista, jotka vaikuttavat niiden yleiseen rakenteeseen ja toimivuuteen. Näiden komponenttien hallitseminen on olennaista ohjelmistokehityksessä.

2.1 Versionhallinta

Versionhallinta on nykyaikaisen ohjelmistokehityksen yksi tärkeimpiä työvälineitä. Versionhallinta on järjestelmä, jonka avulla kehittäjät voivat seurata projektin muutoksia ajan mittaan. Järjestelmä luo muutoksista versiota, jonka avulla kehittäjät voivat tarvittaessa palata projektin aikaisempaan versioon. /7/

Yksinkertaisimpia versionhallintatapoja on kopioida tiedostoja toiseen hakemistoon. Tämä lähestymistapa on yleinen, mutta hyvin virhealtis. /7/

Ensimmäiset versionhallintajärjestelmät olivat paikallisia. Kehittäjillä oli tietokoneella yksinkertainen tietokanta, johon tallennettiin versioita projektista. Yksi isoimmista ongelmista, jonka kehittäjät kohtasivat, oli yhteistyö muiden kehittäjien kanssa. /7/

Suosioon nousi tämän jälkeen keskitetyt versionhallintajärjestelmät. Nämä mahdollistivat tehokkaamman kehittäjien välisen yhteistyön. CVCS, kuten Subversion ja Perforce, luotti yhteen keskitettyyn palvelimeen, joka tallensi projektiin tehtyjen muutosten historian. Vaikka CVCS paransi yhteistyötä ja

mahdollisti tehokkaamman muutosseurannan, oli yksi keskitetty palvelin mahdollinen haavoittuvuus ja kompastuskivi. /7/

Lopuksi kehitettiin hajautetut versionhallintajärjestelmät, joka ratkaisi CVCS:n puutteet. DVCS, kuten Git ja Mercurial, antoivat kehittäjille mahdollisuuden pitää useita erillisiä tietovarastoja (engl. repository), jotka voitiin synkronoida keskenään. Jokaisella kehittäjällä on kopio tietovarastosta, mukaan lukien koko tietovaraston historia. DVCS:llä on useita etuja, mukaan lukien nopeus, paremmat yhdistämisominaisuudet sekä riippumattomuus yhdestä tietovarastosta. /7/

Viime vuosina versionhallinta on kehittynyt jatkuvasti uusien työkalujen ja työnkulkujen (engl. workflow) myötä. Jatkuvasta integroinnista ja julkaisemisesta (CI/CD) on tullut vakiokäytäntö ohjelmistokehityksessä, ja versionhallinta on ratkaisevassa roolissa näissä työnkuluissa. Pilvipohjaisista versionhallinta-alustoista, kuten GitHub, GitLab ja BitBucket, on tullut suosittuja, sillä ne tarjoavat kehittäjille erilaisia yhteistyö- ja automaatiotyökaluja.

2.2 Ohjelmistoriippuvuudet

Useimmat ohjelmistoprojektit ovat nykyään riippuvaisia ulkoisista kirjastoista toimiakseen – tätä yhteyttä kutsutaan ohjelmistoriippuvuudeksi. Riippuvuudet ovat keskeinen osa nykyajan ohjelmistokehitystä ja oikeiden riippuvuuksien käyttö takaa ohjelmiston sujuvan toiminnan, sekä niiden avulla kehittäjien ei tarvitse hukata aikaa tietyn ominaisuuden kehittämiseen, että ohjelmisto toimii.

Proseduuriohjelmoinnin ja ohjelmointikielten, kuten FORTRAN, COBOL ja Algol, kehitys 1950-luvun lopulla ja 1960-luvulla vaikutti merkitsevästi uudelleenkäytettävän koodin ja kirjastojen käsitteen popularisointiin. Näillä ohjelmointikielillä oli ominaisuuksia, kuten alirutiineja ja proseduureja, jotka auttoivat modulaaristen ja uudelleenkäytettävien ohjelmistojen luomisessa. /10/

Olio-ohjelmoinnin yleistyessä, ohjelmistoriippuvuudet tulivat yhä merkitsevämmiksi. Oliolähtöinen suunnittelu mahdollisti

ohjelmistokomponenttien luomisen, joita voitiin käyttää uudelleen eri projekteissa korostaen riippuvuuksien merkitystä ohjelmistokehityksessä. /11/

Ohjelmistoriippuvuudet ovat siis tärkeä osa nykyaikaista ohjelmistokehitystä, koska ne mahdollistavat koodin uudelleenkäytön. Näiden riippuvuuksien hallinta on erittäin tärkeää ohjelmistoprojektien turvallisuuden ja ylläpidettävyyden kannalta.

3 DEVOPS

DevOps-toimintamalli juontaa juurensa 2000-luvun loppupuolelle, jolloin IT-toiminnot ja ohjelmistokehitysyhteisöt olivat huolissaan alan ”tappavan tason toimintahäiriöstä”. Ongelmana oli perinteinen ohjelmistokehitys, joka erotti kehittäjät ja heitä tukevat työntekijät. DevOps-liikkeen alkaessa, tavoitteena oli yhdistää nämä toiminnot, sekä virtaviivaistaa ja optimoida ohjelmistokehityksen eri vaiheita. /20/

Kehitys yleensä sisältää ohjelmoinnin, testauksen ja laadunvarmistuksen, kun taas operaatioihin kuuluvat käyttöönotto, järjestelmän- ja tietokantojenhallinta. Ilmaus ”DevOps” on yhdistelmä sanoista ”Development” ja ”Operations”. /21/

Toimintamallin perustarkoituksena on parantaa ohjelmistojen toimituksen tehokkuutta kehitystyöstä tuotantoon.

Jatkuvat operaatiot

Jatkuvat operaatiot sisältävät menetelmiä, jotka perustuvat iteratiiviseen ohjelmistokehityksen lähestymistapaan, jossa muutoksia tehdään jatkuvasti ja viedään tuotantoon heti, kun ne ovat valmiita. /13/

Jatkuva integrointi tarkoittaa, että kehittäjät pitävät ohjelmiston ajan tasalla ja julkaisuvalmiina niin usein kuin mahdollista. Tämä voi sisältää uusien ominaisuuksien tai korjausten kehittämisen sekä niiden integroinnin ohjelmistoon kokonaisuutena. Ohjelmiston eri osien validoimiseksi suoritetaan perusteellinen testaaminen, mukaan lukien yksikkö-, integraatio- ja toimintatestit. Nämä testit eivät ainoastaan varmista, että yksittäiset komponentit toimivat oikein, vaan myös, että koko järjestelmä toimii tarkoitetulla tavalla, kun komponentit yhdistetään. /13/

Jatkuva toimitus ja julkaiseminen ovat läheisesti toisiinsa liittyviä prosesseja ohjelmistokehityksessä, ja ne nähdään usein kahtena erillisenä, mutta toisiinsa

liittyvänä vaiheena. Molemmat prosessit sisältävät siirtymisen kehitysympäristöstä testaukseen ja lopulta tuotantoon. /13/

Jatkuvassa toimituksessa otetaan julkaisuvalmis ohjelmisto manuaalisesti käyttöön tuotantoon perusteellisen tarkistamisen jälkeen. Manuaalinen tarkistus on hyödyllinen sellaisten ominaisuuksien kohdalla, joita on vaikea automaattisella testaamisella testata, kuten käyttöliittymän käytettävyyttä. Tämä kuitenkin lisää kehittäjien työmäärää, mikä tekee jatkuvasta julkaisemisesta paremman vaihtoehdon joissakin olosuhteissa. /13/

4 RIIPPUVUUKSIEN HALLINTA

Ohjelmistoriippuvuuksien hallinta on olennainen osa nykyaikaista ohjelmistokehitystä. Tehokas riippuvuuden hallinta varmistaa, että ohjelmistoprojektit pysyvät vakaina, turvallisina ja ylläpidettävinä, ja samalla kehittäjät voivat hyödyntää olemassa olevaa koodia ja kirjastoja tuottavuuden parantamiseksi.

4.1 Haavoittuvuudet

Ohjelmistokehityksen haavoittuvuudet ovat ohjelmisto- tai laitteistokoodin puutteita tai haavoittuvuuksia, joilla voi olla negatiivinen vaikutus tietoturvaan ja eheyteen. Haavoittuvuudet ovat merkittävä huolenaihe avoimen lähdekoodin ohjelmistoprojekteissa. Pahimmassa tapauksessa haavoittuvuuksia sisältävä riippuvuus altistaa ohjelmiston hyökkäyksille. /9/

Ohjelmistoriippuvuudet voivat sisältää haavoittuvuuksia vuosia ennen kuin ne huomataan ja korjataan. Ongelman monimutkaisuus ja vakavuus vaikuttaa korjaamiseen. /9/

Log4j

Joulukuussa 2021, Apache Software Foundationin kehittämässä lokikirjasto Log4j:ssä huomattiin merkittävä haavoittuvuus. CVE-2021-44228, joka tunnettiin nimellä ”Log4Shell”, oli haavoittuvuus, joka salli koodin etäsuorittamisen. Hyökkääjät pystyivät hyödyntämään haavoittuvuutta ja mahdollisesti saada hallintaansa kirjastoa käyttäviä järjestelmiä. /12/

Log4Shell oli haavoittuvuus Log4j:n versioissa 2.0–2.14.1. Ongelma johtui siitä, miten kirjasto käsitteli JNDI (Java Naming and Directory Interface) -hakuja lokiviesteissä. Hyökkääjät pystyivät luomaan syötemerkkijonon, joka laukaisi JNDI-haun, mikä johti koodin etäsuorittamiseen, kun kirjasto käsitteli lokiraporttia. /12/

24. Marraskuuta 2021 Alibaba Cloud Security -tiimin asiantuntijat löysivät haavoittuvuuden ja ilmoittivat siitä Apache Software Foundationille. Pian sen jälkeen kyberturvallisuusyrityksen Lunasec asiantuntijat paljastivat haavoittuvuuden 9. Joulukuuta 2021. /12/

Saatuaan haavoittuvuusraportin Apache Software Foundation ryhtyi kehittämään korjausta ongelman korjaamiseksi. 10. Joulukuuta 2021 julkaistiin Log4j-versio 2.15.0, joka sisälsi korjaustoimen Log4Shell-haavoittuuteen. /12/

Tämä on täydellinen esimerkki mahdollisista vaaroista, joita voi syntyä avoimen lähdekoodin laajalti käytettyjen kirjastojen haavoittuvuuksista ja kuinka tärkeää on että projektien riippuvuudet pidetään ajan tasalla.

4.2 Automaattiset ohjelmistoriippuvuuksien hallintatyökalut

Riippuvuuksien ajan tasalla pitämisen tärkeyttä ohjelmistoprojekteissa ei voi vähätellä, sillä se takaa ohjelmiston vakauden, turvallisuuden ja ylläpidettävyyden. Automaattiset riippuvuuden hallintatyökalut ovat nousseet pakolliseksi ratkaisuksi, joka tarjoaa kehittäjille etuja, jotka auttavat ylläpitämään ohjelmistoprojektien riippuvuuksia.

Renovatebot

Renovatebot on yksi tällainen työkalu, joka auttaa ohjelmistoriippuvuuksien hallinnassa. Renovatebot on avoimeen lähdekoodiin perustuva automaattinen työkalu, jonka tarkoituksena on helpottaa ohjelmistoprojektien riippuvuuksien päivittämistä ja ylläpitoa. Sen päärooli on seurata projektiriippuvuuksia, löytää vanhentuneita tai haavoittuneita kirjastoja ja lähettää pyyntöjä niiden päivittämiseksi automaattisesti. /3/

Renovatebot tarjoaa useita haluttuja ominaisuuksia, jotka vastaavat nykyaikaisten ohjelmistokehitysprojektien tarpeisiin. Joitakin Renovatebotin merkittäviä ominaisuuksia ovat mm. /3/

- Automaattiset riippuvuuksien päivitykset.
- Konfiguroitavia ominaisuuksia, mm. Päivitysstrategia.
- Tuki laajalle valikoimalle ohjelmointikieliä ja paketinhallintaohjelmia.
- Integrointi suosittujen CI/CD-alustojen kanssa.

Renovatebot ei ole ainoa työkalu markkinoilla riippuvuuksien hallintaan, joten muihin vertaaminen on myös tärkeää. Joitakin suosittuja vaihtoehtoja olivat mm. Dependabot ja Snyk.

Dependabot

Githubin vuonna 2019 hankkima riippuvuuden hallintatyökalu, joka tarjoaa riippuvuuspäivityksiä eri kielille ja paketinhallintaohjelmille. Vaikka Dependabot on suosittu ja jakaa samoja ominaisuuksia Renovatebotin kanssa, siitä puuttuu tiettyjä konfigurointivaihtoehtoja, tuettuja kieliä ja paketinhallintaohjelmia. /14/

Snyk

Tietoturvaan keskittyvä riippuvuuden hallintatyökalu, joka tarjoaa automaattisia päivityksiä ja haavoittuvuustarkistuksia. Snyk on erinomainen työkalu, mutta valitettavasti se on vähemmän monipuolinen kuin Renovatebot konfigurointivaihtoehtojen suhteen ja jotkut ominaisuudet vaativat maksullisen tilauksen. /15/

Renovatebot tunnistaa projekteissa käytetyt riippuvuudet ja niiden versiot. Tämä prosessi vaihtelee ohjelmointikielen ja paketinhallinnan mukaan. Skannaamisen aikana Renovatebot etsii tiettyjä tiedostoja, nimeltään ”package files”, jotka sisältävät erilaista tietoa projektista, mm. Riippuvuuksista. Esimerkkejä eri ohjelmointikielten pakettitiedostoista: /3/

- JavaScript: package.json (npm/Yarn).
- Java: pom.xml (Maven) tai build.gradle (Gradle).
- Python: requirements.txt (pip) tai Pipfile (Pipenv).

- Ruby: Gemfile (Bundler).
- PHP: composer.json (Composer).

Renovatebot tukee myös Dockeria, Microsoftin kehittämää Bicepia ja Googlen kehittämää Bazelia.

Kun projektissa olevat riippuvuudet ja niiden versiot on analysoitu, Renovatebot etsii uusimmat saatavilla olevat versiot riippuvuuksista. Prosessi lähettää kyselyn asianmukaiseen pakettirekisteriin tai rajapintaan pakettitietojen saamiseksi. Tämän jälkeen jäsennetään saadut tiedot määrittääkseen kunkin riippuvuuden uusimman version ottaen huomioon tekijät, kuten julkaisua edeltävät versiot, vanhentuneet versiot ja versiointimallit (esim. semanttinen versio). /3/

Renovatebot luo vetopyynnöt jokaiselle riippuvuuspäivitykselle määritettyä päivitysstrategiaa ja konfigurointia noudattaen.

Haaroitus

Branching, eli haaroittuminen, on versionhallintajärjestelmien perusominaisuus. Se mahdollistaa itsenäisten kehityspolkujen luomisen joiden avulla kehittäjät voivat työskennellä eri ominaisuuksien, virheenkorjausten tai kokeilujen parissa vaikuttamatta ”master” haaraan. /7/

”Master” haara, jota yleisesti kutsutaan myös ”main” haaraksi, on oletus- ja ensisijainen haara versionhallintajärjestelmissä. Se tyypillisesti sisältää uusimman ja vakaimman version projektista. Päähaara on lähtökohta kaikille myöhemmille haaroille, jotka muodostetaan ohjelmistokehitysprosessin aikana. /7/

Muutosten sisällyttämistä yhdestä haarasta toiseen kutsutaan yhdistämiseksi (engl. merging). Kun ominaisuus- tai virheenkorjaushaara katsotaan valmiiksi ja vakaaksi, se voidaan yhdistää takaisin päähaaraan integroimalla muutokset koodikantaan. Tämä vaihe voi myös sisältää ristiriitojen ratkaisemisen, jos samoja koodin osia on muokattu molemmissa haaroissa. /7/

Vetopyyntö

Pull request, eli vetopyyntö on ominaisuus, jolla ehdotetaan ja arvioidaan muutoksia koodikantaan ennen muutosten sisällyttämistä kohdehaaraan. Vetopyynnöt auttavat ylläpitämään koodin laatua ja helpottamaan tiimiviestintää ja yhteistyötä.

Versionhallintajärjestelmissä vetopyynnöt toimivat vertaamalla ehdotettujen muutosten haaraa, päähaaraan. Näiden haarojen väliset erot näkyvät, kun kehittäjä tekee vetopyynnön, jossa korostetaan päähaarassa tehtyjä lisäyksiä, poistoja tai muutoksia.

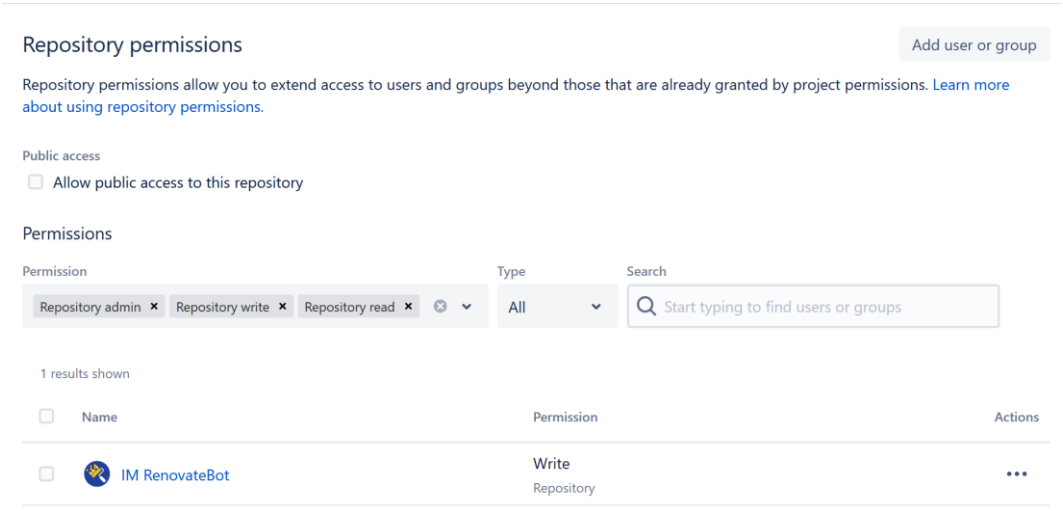
Renovatebotin käyttöönotto

Renovatebot on monipuolinen työkalu, joka tarjoaa monenlaisia tapoja käyttöönottamiseen haluamastasi alustasta, hostausympäristöstä ja työnkulusta riippuen. Jos käytössä on esimerkiksi Github, helpoin tapa aloittaa Renovatebotin käyttö on asentaa Renovate Github -sovellus tietovarastoon.

Voit myös hostata Renovatebottia itse omassa infrastruktuurissa, mikä tarjoaa enemmän hallintaa ja joustavuutta sen toiminnalle. Tämä vaihtoehto sopii parhaiten esimerkiksi BitBucket Serverille tai Azure DevOpsille. Voit hostata Renovatebottia itse käyttämällä virallista Docker -näköistiedostoa tai asentaa sen paketinhallintaohjelma npm:n avulla. Konfigurointi tehdään config.js tiedostolla tai ympäristömuutujia käyttäen ja Renovatebotin suorittaminen säännöllisin väliajoin onnistuu mm. Cronjobilla. /3/

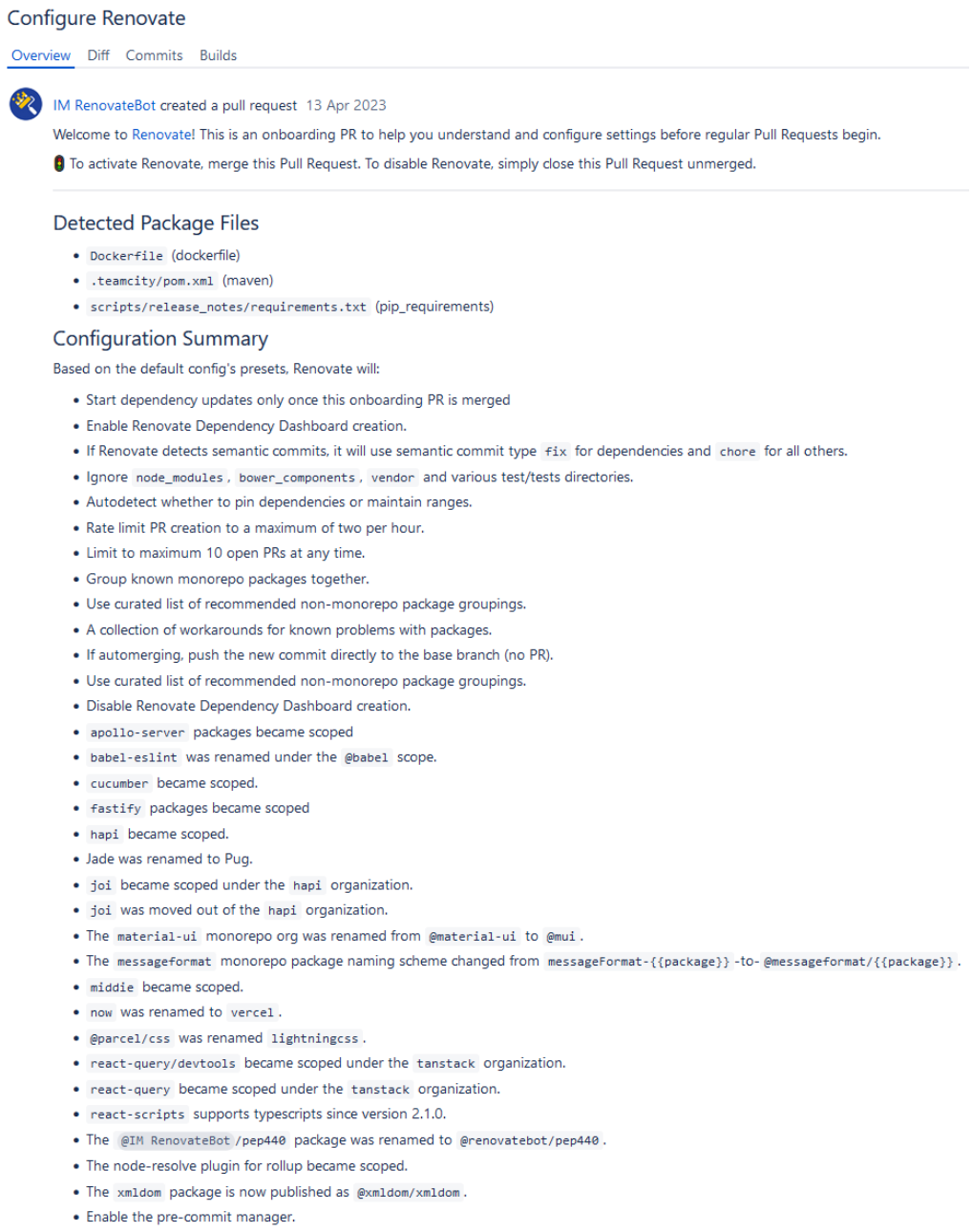
Cronjob, on aikapohjainen työkalu, jota käytetään yleisesti Unix-tyyppisissä käyttöjärjestelmissä. Cronia käytetään automatisoimaan toistuvia tehtäviä, kuten komentosarjojen suorittamista tai tietojen varmuuskopiointia suorittamalla komentoja tietyin väliajoin. /18/

Kun Renovatebot on käytössä, tarvitsee se kirjoitusoikeudet projektiin käytettävässä versionhallintajärjestelmässä. Kuvassa 1 on Renovatebotille annettu kirjoitusoikeudet projektiin BitBucket-versionhallintajärjestelmässä.



Kuva 1. Renovatebotin oikeudet projektissa

Käyttäjän valittua projektit, joihin haluaa Renovatebotin integroituvan, avaa Renovate ensimmäisen vetopyynnön asetusten määrittämistä varten. Kuvassa 2 olevassa vetopyynnössä näkyy, mitä pakettitiedostoja Renovate on tunnistanut ja minkälaisia muutoksia se aikoo tehdä konfiguroinnin perusteella.



Configure Renovate

Overview Diff Commits Builds

IM RenovateBot created a pull request 13 Apr 2023

Welcome to Renovate! This is an onboarding PR to help you understand and configure settings before regular Pull Requests begin.

To activate Renovate, merge this Pull Request. To disable Renovate, simply close this Pull Request unmerged.

Detected Package Files

- `Dockerfile` (dockerfile)
- `.teamcity/pom.xml` (maven)
- `scripts/release_notes/requirements.txt` (pip_requirements)

Configuration Summary

Based on the default config's presets, Renovate will:

- Start dependency updates only once this onboarding PR is merged
- Enable Renovate Dependency Dashboard creation.
- If Renovate detects semantic commits, it will use semantic commit type `fix` for dependencies and `chore` for all others.
- Ignore `node_modules`, `bower_components`, `vendor` and various test/tests directories.
- Autodetect whether to pin dependencies or maintain ranges.
- Rate limit PR creation to a maximum of two per hour.
- Limit to maximum 10 open PRs at any time.
- Group known monorepo packages together.
- Use curated list of recommended non-monorepo package groupings.
- A collection of workarounds for known problems with packages.
- If automerging, push the new commit directly to the base branch (no PR).
- Use curated list of recommended non-monorepo package groupings.
- Disable Renovate Dependency Dashboard creation.
- `apollo-server` packages became scoped
- `babel-eslint` was renamed under the `@babel` scope.
- `cucumber` became scoped.
- `fastify` packages became scoped
- `hapi` became scoped.
- Jade was renamed to Pug.
- `joi` became scoped under the `hapi` organization.
- `joi` was moved out of the `hapi` organization.
- The `material-ui` monorepo org was renamed from `@material-ui` to `@mui`.
- The `messageformat` monorepo package naming scheme changed from `messageFormat-{{package}}` -to- `@messageformat/{{package}}`.
- `middie` became scoped.
- `now` was renamed to `vercel`.
- `@parcel/css` was renamed `lightningcss`.
- `react-query/devtools` became scoped under the `tanstack` organization.
- `react-query` became scoped under the `tanstack` organization.
- `react-scripts` supports typescripts since version 2.1.0.
- The `@IM RenovateBot/pep440` package was renamed to `@renovatebot/pep440`.
- The node-resolve plugin for rollup became scoped.
- The `xmldom` package is now published as `@xmldom/xmldom`.
- Enable the pre-commit manager.

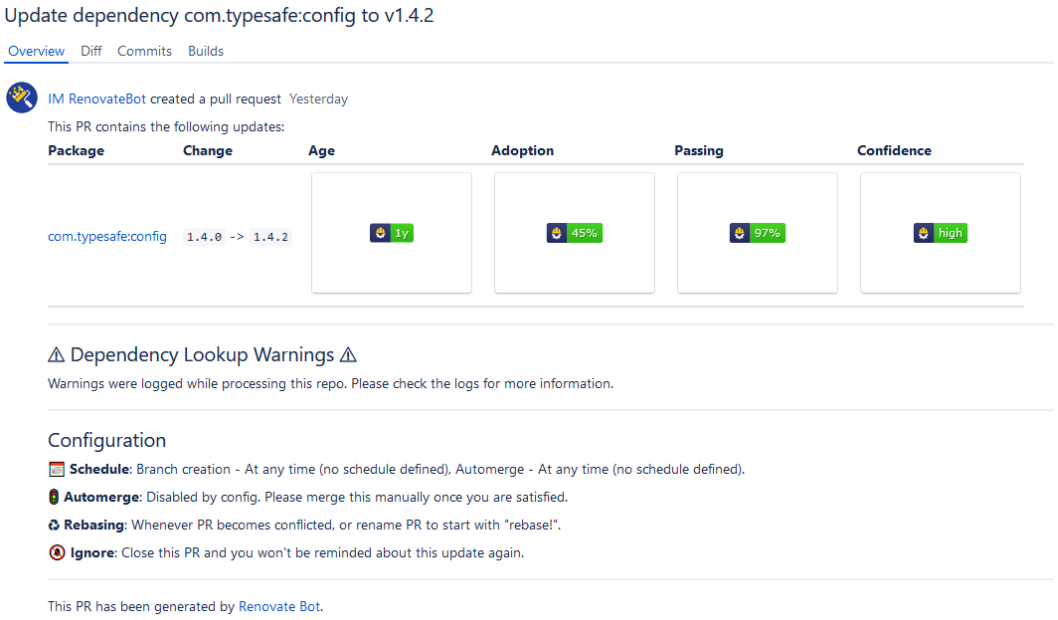
Kuva 2. Renovatebotin ensimmäinen vetopyyntö

Ohjelmistoprojektien versiointi

Semanttinen versiointi, lyhennettynä SemVer, on versiointijärjestelmä, joka käyttää jäsenneltyä muotoa ilmaisemaan ohjelmistojulkaisujen välisten muutosten laajuuden ja yhteensopivuuden. SemVer käyttää muotoa: MAJOR.MINOR.PATCH, esimerkiksi 3.2.1. Jokaisella osalla on merkitys:

MAJOR-komponentilla tarkoitetaan merkittäviä muutoksia, jotka saattavat rikkoa yhteensopivuuden aikaisempien pääversioiden kanssa. Päivitys uuteen pääversioon saattaa edellyttää muutoksia koodiin. MINOR-komponentilla tarkoitetaan rikkoutumattomia, taaksepäin yhteensopivia muutoksia, jotka tarjoavat uusia tai parantavat olemassa olevia ominaisuuksia. Päivityksellä ei pitäisi olla vaikutusta olemassa oleviin ominaisuuksiin. Lopuksi PATCH-komponentti määrittää taaksepäin yhteensopivia virheenkorjauksia tai pieniä muutoksia, jotka korjaavat ohjelmistovirheet ilman uusia ominaisuuksia. Korjauspäivitykset eivät saa vaikuttaa olemassa olevaan koodiin. /16/

Kuten kuvassa 3 huomataan, Renovatebot teki vetopyynnön, jossa haluaa päivittää riippuvuuden, joka käyttää perustana 1.4.0-versiota. Koska päivitys on versiosta 1.4.0 versioon 1.4.2, tällä tarkoitetaan patch päivitystä, eli päivittäminen ei vaatisi muutoksia projektin koodiin.



Kuva 3. Renovatebotin vetopyyntö tunnistetuista riippuvuuksista

Renovatebotin konfigurointi tapahtuu renovate.json tiedostossa, joka sijoitetaan projektin juurihakemistoon.

Renovate antaa käyttäjille paljon vapauksia asetusten määrittämisessä. Yleisimpiä asetuksia ovat mm. Jaetun konfiguraatiotiedoston periminen: jos käyttäjällä on monta projektia, joihin halutaan samoja asetuksia, voidaan tehdä jaettu konfiguraatiotiedosto ja periä se eri tietovarastoissa. Käyttäjä voi myös määrittää aikataulun, milloin Renovatebotin tulisi juosta, esimerkiksi työaikojen ulkopuolella tai ainoastaan viikonloppuisin. Kuvassa 4 on esimerkki tyypillisestä tiedostorakenteesta.

```
{
  "extends": ["config:base"],
  "schedule": ["after 10pm", "before 5am"],
  "automerge": true,
  "automergeType": "branch",
  "packageRules": [
    {
      "updateTypes": ["major"],
      "automerge": false
    }
  ]
}
```

Kuva 4. Tyypillinen renovate.json asetustiedosto

Renovatebotin konfigurointi tehdään jokaiselle tietovarastolle erikseen. Jos tietovarastoja on useampia, on viisasta tutustua Renovatebotin tarjoamiin ominaisuuksiin, kuten jaetun konfiguraatiotiedoston periminen.

Renovatebotin jaetun konfiguraation avulla voit käyttää yhteisiä asetuksia useissa tietovarastoissa. Tämä on erityisen hyödyllistä, kun hallitaan useita tietovarastoja, joilla on samanlaiset riippuvuudet tai käytännöt, koska se auttaa säilyttämään johdonmukaisuuden ja vähentämään kunkin tietovaraston määrittämiseen tarvittavaa vaivaa.

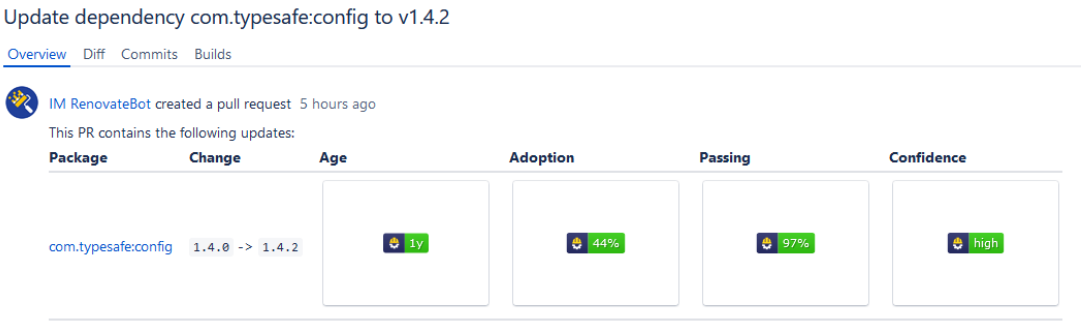
Jaettu konfiguraatio yksinkertaistaa asetusten ylläpitoa ja päivitystä. Kun jaettuun konfiguraatioon tehdään muutos, se päivittyy automaattisesti kaikkiin tietovarastoihin, jotka viittaavat kyseiseen konfiguraatioon.

Kuvassa 4 olevassa esimerkki konfiguraatiotiedostossa, ”extends”-kenttä on konfigurointivaihtoehto, jonka avulla perintä yhdestä tai useammasta jaetusta konfiguraatiosta onnistuu. Renovatebot tarjoaa useita sisäänrakennettuja esiasetuksia, kuten ”config:base”, joka koostuu suositeltavista oletusasetuksista. Kuvassa 5 on esimerkki, kuinka viitataan omaan jaettuun konfiguraatiotiedostoon.

```
"extends": [
  "devop/renovatebot-config:automerge"
],
```

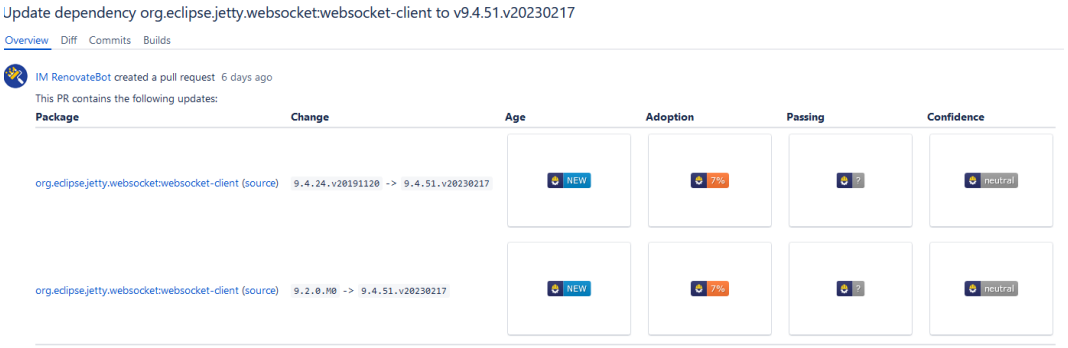
Kuva 5. Viittaus omaan jaettuun konfiguraatiotiedostoon

Yksi uusista Renovatebotin ominaisuuksista on nimeltään ”Merge Confidence”, joka on suunniteltu auttamaan kehittäjiä tekemään tietoisia päätöksiä riippuvuuspäivitysten yhdistämisestä niiden mahdollisen vaikutuksen perusteella projektin vakauteen. Kuvasta 6 huomataan kirjaston Typesafe Configin ikä, adaptointi, testien läpäisyprosentti ja Renovatebotin luottamus päivitykseen. /3/



Kuva 6. Merge Confidence

Jos Renovatebot tunnistaa, että tietovarastossa on useita projekteja, jotka käyttävät samoja riippuvuuksia, ryhmittelee se päivitykset samaan vetopyyntöön. Kuvassa 7 on vetopyyntö, jossa on 2 eri projektia samassa tietovarastossa, jotka käyttävät samaa vanhentunutta riippuvuutta. /3/



Kuva 7. Päivitysten ryhmittely

Renovatebot on hyödyllinen ja monipuolinen riippuvuuden hallintatyökalu, joka on suunniteltu helpottamaan ohjelmistoriippuvuuksien päivittämistä projekteissa. Se on suunniteltu kehittäjille ja organisaatioille, joiden on ylläpidettävä monia tietovarastoja. Renovate säästää aikaa ja vähentää inhimillisten virheiden vaaraa riippuvuuksien päivityksessä automatisoimalla tämän prosessin, mikä johtaa luotettavampiin ja turvallisempiin ohjelmistoprojekteihin.

5 RENOVATEBOTIN KÄYTTÖ ORGANISAATIOSSA

5.1 Automaatiotyökalun tarve

Iso osa organisaation projekteista käyttää riippuvuuksia tuomaan ohjelmistoihin ominaisuuksia, tehostamaan projektien toiminnallisuutta ja nopeuttamaan kehitystä. Projektien jatkuva kehitys ja riippuvuuksien kasvava määrä johtaa ns. Tekniseen velkaan (engl. technical debt). Kun riippuvuuksia alkaa olemaan huomattava määrä, niiden manuaalinen päivittäminen on hidasta, joten se jätetään yleensä tekemättä. Insinöörityön tarkoituksena oli käyttöönottaa automaatiotyökalu riippuvuuksien hallintaan ja räätälöidä se eri projektien tarpeisiin.

5.2 Renovatebotin käyttöönotto ja konfigurointi

Ensimmäisenä tehtävänä oli ratkaista, millä projektilla olisi helppoa testata Renovatebotin toimivuutta ja ominaisuuksia. Päädyimme valitsemaan projektin, jossa oli käytössä useita eri pakettitiedostoja, jotta voisimme myös testata konfigurointia.

Projektissa oli käytössä Web-serveri nginx, joka oli tällä hetkellä ainut mitä halusimme päivittää kyseisestä tietovarastosta. Konfiguroinnin avulla pystyimme kertomaan Renovatebotille, että mitään muita riippuvuuksia emme halunneet päivittää paitsi ne, joissa mainitaan ”nginx”. Kuvassa 8 on Renovatebotin asetukset kyseiselle projektille.

```
{
  "$schema": "https://docs.renovatebot.com/renovate-schema.json",
  "extends": [
    "devop/renovatebot-config:automerge"
  ],
  "packageRules": [
    {
      "matchPackagePatterns": ["*"],
      "enabled": false
    },
    {
      "matchPackagePatterns": ["nginx*"],
      "enabled": true
    }
  ]
}
```

Kuva 8. Renovatebotin asetukset projektissa

Renovatebotin suoraan käyttöönottaminen projekteihin ilman testaamista on mahdollisesti vaarallista ja voi aiheuttaa odottamattomia ongelmia. Testataksemme Renovatebotin käyttöönottoa projekteissa, pystyy versionhallintajärjestelmissä ottamaan oman kopion tietovarastosta, tätä kutsutaan nimellä ”forking”.

Haarautuminen

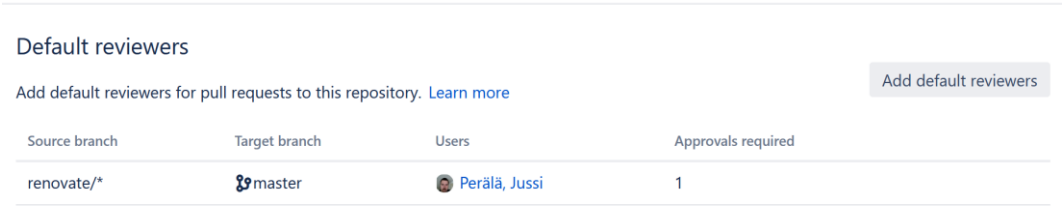
Versionhallintajärjestelmissä, forking eli haarautuminen tarkoittaa erillisen kopion luomisen olemassa olevasta tietovarastosta. Tämän avulla kehittäjät voivat työskennellä uuden kopion parissa itsenäisesti vaikuttamatta alkuperäiseen lähdekoodiin tai projektiin. Haarautuminen tarjoaa turvaverkon kehittäjille, varsinkin kun työskennellään kokeellisten tai riskialttiiden muutosten kanssa. Haarautumisen ansiosta pystytään testaamaan Renovatebottia tuottamatta ongelmia pääprojektiin. /7/

Omien kopioiden luominen BitBucketissa oli hyvin yksinkertaista, valitsemalla tietovarasto ja tietovaraston sivulta ”Create fork”. Seuraavassa vaiheessa annettaan mahdollisuus synkronoida alkuperäinen ja kopio, joka tarkoittaa sitä, että alkuperäisen projektin kehittyessä, kopio voi kuitenkin vanhentua ja jättää

huomiotta tärkeitä päivityksiä. Tämä pitää kopiosi ajan tasalla. Kyseisellä ominaisuudella ei ole käyttöä tässä tapauksessa.

Konfigurointi

Renovatebot perusasetuksilla tekee vetopyynnön riippuvuuden päivityksestä ja riippuen projektin asetuksista, vaatii tämä yhden tai useamman kehittäjän manuaalisen tarkastuksen. Tämä tarkoittaa sitä, että kehittäjä joutuu ensin etsiä vetopyynnön ja lisätä itsensä tarkastajaksi, jonka jälkeen hän voi joko hylätä tai hyväksyä vetopyynnön yhdistämisen päähaaraan. Yksi halutuista ominaisuuksista oli se, että jokaiseen Renovatebotin tekemään vetopyyntöön lisättäisiin kehittäjät automaattisesti tarkastajiksi. Tämän pystyy tehdä Renovatebotin konfiguraatiotiedostoon tietovarasto kohtaisesti asetuksella ”reviewers” tai käyttää BitBucketin ”Default Reviewers” ominaisuutta. Kuva 9 havainnollistaa, miten oletustarkastajat on lisätty tietovarastoon.



Kuva 9. Tietovaraston oletustarkastajat

Kaikki vetopyynnöt, jotka tehdään renovate etuliitteellä olevista haaraumista, asettaa BitBucket määritetyt käyttäjät oletustarkastajiksi. Sama onnistuu myös Renovatebotin konfiguroinnin kautta, käyttämällä yllä mainittua ”reviewers” asetusta.

```
{
    "$schema": "https://docs.renovatebot.com/renovate-schema.json",
    "reviewers": ["user1", "user2", "user3"]
}
```

Kuva 10. Reviewers asetuksen käyttö konfiguroinnissa

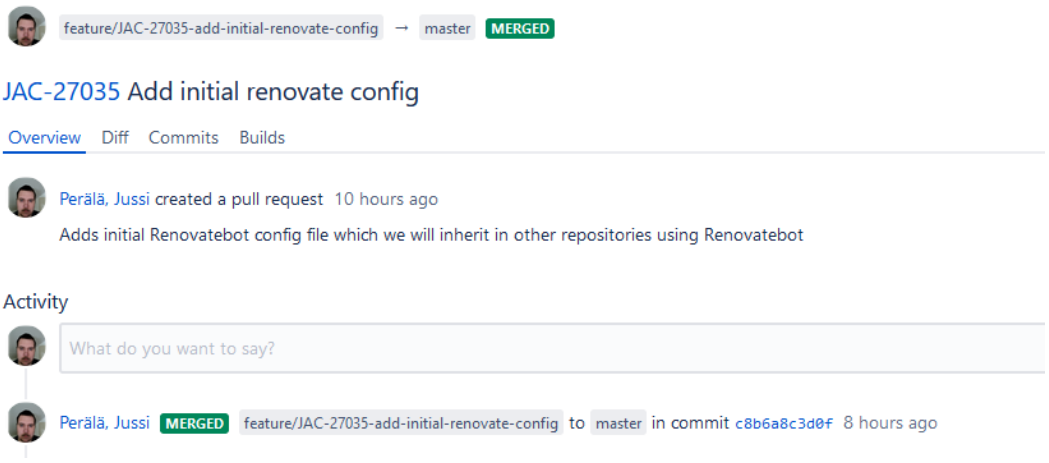
Kuvassa 10 on esimerkki, kuinka asetusta voi käyttää. Kummankin ominaisuuden käyttäminen vaatii manuaalista työtä. BitBucketin tapauksessa, joudutaan jokaiseen projektiin erikseen määritellä oletustarkastajat. Jos käytämme Renovatebotin asetusta, voimme hyödyntää jaettua konfiguraatiota. Jokaisessa projektissa mihin otamme Renovatebotin käyttöön, peritään jaettu konfiguraatiotiedosto, jossa oletustarkastajat on määritelty valmiiksi.

Koska Renovatebot tulee käyttöön useampaan tietovarastoon, on niiden manuaalinen konfigurointi hidasta. Joten tässä tapauksessa on järkevintä käyttää jaettua konfiguraatiota, sillä sen pystyy perimään niin moneen tietovarastoon kuin haluaa ja muutokset tarvii tehdä vain yhteen paikkaan. Helpoin tapa on tehdä uusi tietovarasto, joka koostuu vain konfiguraatiotiedostosta.

Jotta kehittäjätiimit pysyvät ajan tasalla tiimiläisten työskentelystä, on suositeltavaa käyttää tähän suunniteltua ohjelmistoa tai alustaa. Jira on Atlassianin kehittämä tehokas projektinhallinta- ja työnseurantaohjelmisto, jonka avulla tiimit pystyvät organisoimaan työtään tehokkaasti. Jiran avulla voidaan jakaa suuret projektit pienempiin, paremmin saavutettavissa oleviin tehtäviin. Jira tarjoaa mahdollisuuden jakaa tehtäviä, hallita tehtävien suorittamista ja seurata tiimin jäsenten työkuormia. Kuvassa 11 on lippu Renovatebotin tietovaraston tekemiseen. /23/

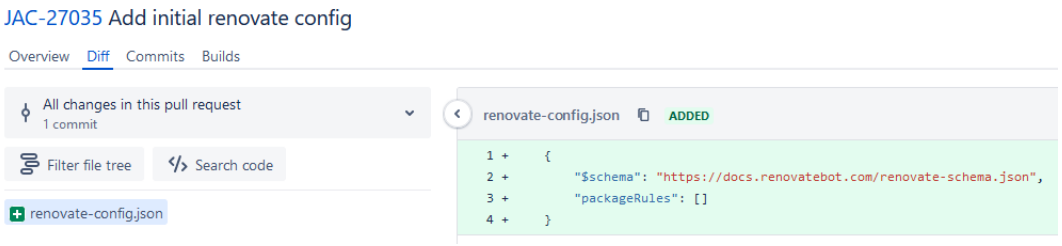
Kuva 11. Jira lippu tietovaraston tekoon

Jokaisella Jira projektilla on yksilöllinen avain, joka toimii kyseisen projektin tunnisteena. Esimerkiksi kuvassa 11 olevassa Jira-lipussa avaintunniste ”JAC-27035” tarkoittaa, että kyseiseen projektiin on luotu 27035 lippua. Tällä lipulla viitataan mahdolliseen ongelmaan, tehtävään tai ominaisuuteen. Kyseistä avainta käytetään haaraumien nimissä, jotta lippu ja haarauman tuomat ominaisuudet voidaan yhdistää. Kuvasta 12 huomataan, että on tehty uusi haara, jossa halutaan tuoda uusi ominaisuus päähaaraan.



Kuva 12. Vetopyyntö haarauman yhdistämiseen

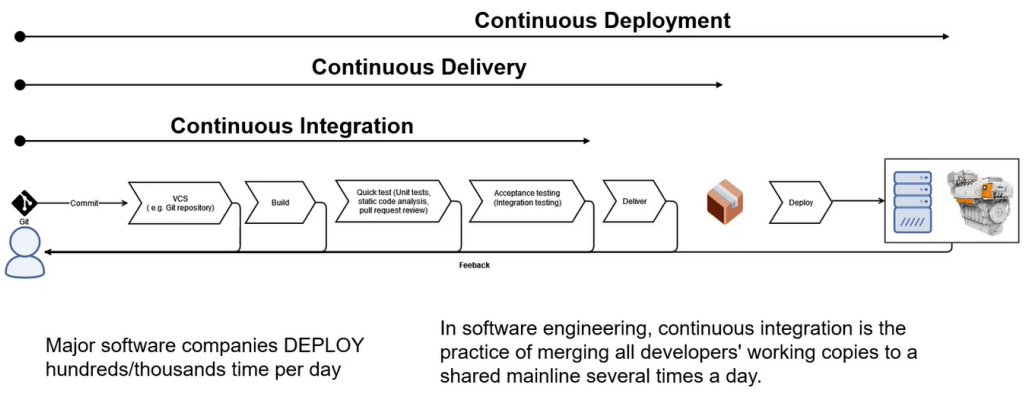
Kuvasta 13 erityisesti nähdään eroavaisuus ominaisuus- ja päähaaran välillä ja mitä muutoksia niiden yhdistäminen mahdollisesti toisi päähaaraan. Tässä solmitussa muutoksessa ainut ”ominaisuus” on Renovatebotin konfiguraatiotiedoston luominen projektin juurihakemistoon.



Kuva 13. Eroavaisuus päähaaraan

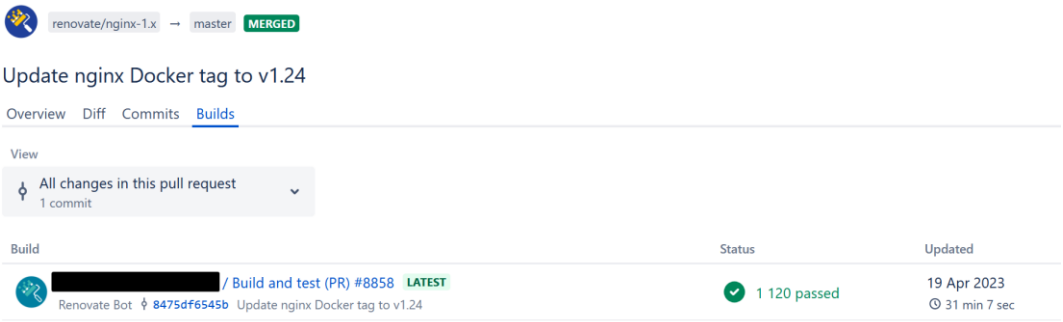
CI/CD Pipeline

CI/CD-putkissa kehittäjien yhdistäessä koodimuutokset tietovarastoihin, automaattiset koontiversiot ja testit käynnistyvät. Kuvassa 14 on illustraatio, kuinka putkiston sisällä oleva prosessi etenee.



Kuva 14. Illustraatio putkilinjasta

Renovatebot luottaa jatkuvan integroinnin putkilinjaan. Jos automaattiset koonnit tai testit epäonnistuvat, vetopyyntöä ei voi yhdistää. Tämä osoittaa, että päivitys saattaa aiheuttaa yhteensopivuusongelmia.

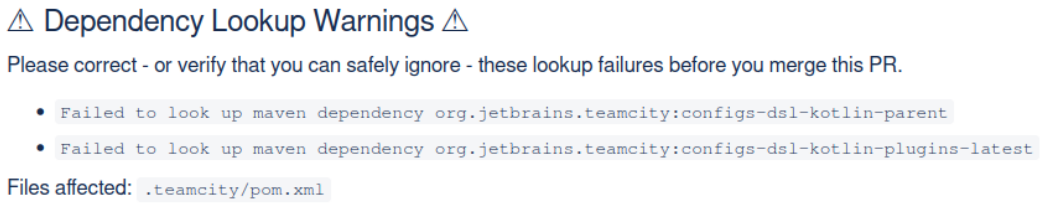


Kuva 15. Automaattiset vetopyynnön testit

Kun mahdolliset ongelmat on ratkaistu ja testit on läpäisseet, vetopyyntö yhdistetään joko manuaalisesti tai automaattisesti konfiguraatiosta riippuen. Kuvassa 15 on esimerkki läpäistyistä koonneista ja testeistä.

Yhdeksi ongelmaksi nousi Renovatebotin ilmoittama ns. Riippuvuushaun varoitus. Kuvasta 16 huomataan, että Renovatebot epäonnistui löytämään configs-dsl-kotlin-parent ja configs-dsl-kotlin-plugin-latest kirjastot. Tämä johtuu siitä, että Wärtsilän käyttämä TeamCity tarjoaa nämä kyseiset kirjastot, eikä niitä löydy julkisesta pakettirekisteristä. Koska TeamCity pyörii omassa virtuaalisessa

pilviympäristössä johon Renovatebotilla ei ole oikeutta, emme pysty kertomaan Renovatebotille mistä nämä kirjastot löytyvät.



Kuva 16. Riippuvuushaun varoitus

Tutkiessa .teamcity/pom.xml tiedostoa huomattiin, että nämä kirjastot käyttävät suoraan viimeisintä julkaistua versiota, joten niiden päivittämiseen ei Renovatebottia tarvitse.

Ongelman ratkaisemiseksi voimme siis täysin sivuuttaa projekteissa olevan .teamcity kansion. Tämä onnistuu Renovatebotissa asetuksella ”IgnorePaths”, jolla voi valikoivasti sivuuttaa pakettitiedostot, jotka Renovate tavallisesti löytäisi ja päivittäisi. Kuvassa 17 on esimerkki, kuinka kyseistä asetusta voidaan käyttää.

```
{
    "$schema": "https://docs.renovatebot.com/renovate-schema.json",
    "ignorePaths": [
        "**/.teamcity/**"
    ]
}
```

Kuva 17. ignorePaths asetus

Kaksoistähteä (**) käytetään ns. Jokerimerkkinä, joka vastaa mitä tahansa hakemistotasoa, kun taas yksi tähti (*) vastaa minkä tahansa tiedoston tai kansion nimeä.

Kun jaettu konfiguraatiotiedosto oli luotu ja Renovatebottia testattu ”forkatuissa” tietovarastoissa, voitiin aloittaa integroiminen oikeisiin tietovarastoihin. Konfiguraatiossa määritettiin oletustarkastajat jokaiseen Renovatebotin tekemään vetopyyntöön, sekä tietyn kansion sivuuttamisen. Kuvassa 18 on lopullinen versio konfiguraatiosta.

```
{
    "$schema": "https://docs.renovatebot.com/renovate-schema.json",
    "ignorePaths": ["**/.teamcity/**"],
    "reviewers": [[redacted] "jpe078", [redacted]],
    "packageRules": [
        {
            "matchUpdateTypes": ["minor", "patch", "pin", "digest"],
            "automerge": false
        }
    ]
}
```

Kuva 18. Lopullinen konfiguraatiotiedosto

6 YHTEENVETO

Insinöörityössä perehdyttiin Renovatebotin käyttöön organisaatiossa ja miten sillä pystytään ylläpitämään ohjelmistoriippuvuuksia. Työssä tutustuttiin myös ohjelmistoprojekteihin, mistä ne koostuvat ja minkälaisia työkaluja ohjelmistokehityksessä käytetään, kuten versionhallintaa. Tutkimustyön avulla pystyttiin suorittamaan projekti, jossa saatiin Renovatebot käyttöön useaan tietovarastoon.

Projektin alkuvaiheissa aikaa käytettiin Renovatebotin ominaisuuksien tutkimiseen ja ymmärtämiseen. Tämän jälkeen työkalua testattiin yksittäisessä projektissa, jonka jälkeen alkoi käyttöönotto isommalla mittakaavalla ja konfigurointi eri projektien tarpeisiin.

Analysoidessa tuloksia, Renovatebot oli ehdottanut 132 uutta vetopyyntöä. Koska BitBucket ei tue vetopyynnöissä ns. Etikettejä, on vaikea arvioida kuinka moni näistä päivityksistä korjasi esim. haavoittuvuuksia. Isoimmassa projektissa Renovatebot ehdotti 85 vetöpyyntöä, joista 15 oli MAJOR, 60 MINOR ja 10 PATCH -päivitys. Huomasimme kuitenkin, että yksi näistä MINOR-päivityksistä korjasi haavoittuvuuden Apache Commons Text -kirjastossa.

Apache Commons Text on suosittu Java-kirjasto tekstinkäsittelyyn, joka sisältää haavoittuvuuden versioissa 1.5–1.9 sen oletusarvoisen muuttujien interpolointiominaisuuden vuoksi. Ongelma johtuu tietyistä interpolaattoreista, mukaan lukien ”script”, ”dns” ja ”url”, jotka voivat johtaa mielivaltaiseen koodin suorittamiseen tai tahattomaan kommunikointiin etäpalvelimien kanssa, kun käytetään petollisia määritysarvoja. Ongelman korjaamiseksi suositellaan päivittämistä versioon 1.10, joka poistaa nämä ongelmalliset interpolaattorit käytöstä. /19/

Käytössä oleva Orca Security, tietoturvatyökalu, joka tarjoaa kattavan uhkien havaitsemisen julkisissa pilvialustoissa, kuten AWS ja Microsoft Azure, ilmoitti

haavoittuneiden ohjelmistojen vähentyneen Renovatebotin käyttöönoton jälkeen.

Työn tuloksesta voidaan päätellä, että automaatiotyökalun käyttöönottaminen parantaa organisaation ohjelmistokehitystä. Säännölliset riippuvuuspäivitykset auttavat korjaamaan tietoturva-aukkoja, suojaamaan ohjelmistoja mahdollisilta riskeiltä ja pitämään riippuvuudet ajan tasalla. Renovatebot säästää kehittäjien aikaa automatisoimalla riippuvuuspäivitykset, jolloin he voivat käyttää aikansa paremmin ohjelmistokehityksen eri vaiheisiin.

Hyödyistä huolimatta on automaatiotyökalun käyttämisessä myös omat huonot puolensa. Riippuvuuspäivitykset ei välttämättä pääse läpi testeistä tai aiheuttaa koontivirheitä, jotka vaativat manuaalisia toimenpiteitä ja vianmääritystä. Vaikka Renovatebot automatisoi päivitysprosessin, vetopyynnöt on silti manuaalisesti tarkistettava ja hyväksyttävä, mikä on taas ristiriidassa automatisoinnin kanssa.

Automaatiotyökalun käyttöönotto riippuvuuksienhallintaan voi nopeuttaa ohjelmistokehitysprosessia ja parantaa kehittäjien tuottavuutta vähentämällä manuaalista työtä. On kuitenkin tärkeää huomioida myös huonot puolet ja räätälöidä automaatiotyökalu vastaamaan organisaation tarpeita.

LÄHTEET

/1/ Itewiki. Ohjelmistokehitys – Ite wikin digitalisoinnin opas 2018. Luettu 15.2.2023. https://www.itewiki.fi/opas/ohjelmistokehitys/

/2/ IBM Corporation. What is software development. Luettu 16.2.2023. https://www.ibm.com/topics/software-development

/3/ White Source, Ltd. Renovate Docs. Viitattu 15.4.2023. https://docs.renovatebot.com/

/4/ Github – Renovatebot. Luettu 13.2.2023 https://github.com/renovatebot/renovate

/5/ Itewiki. DevOps – Ite wikin digitalisoinnin opas 2018. Luettu 3.4.2023. https://www.itewiki.fi/opas/devops/

/6/ About – Git. Luettu 16.03.2023. https://git-scm.com/about

/7/ Pro Git, 2014. Chacon, Scott & Straub, Ben. E-Kirja, Apress. Viitattu 15.4.2023. https://git-scm.com/book/en/v2

/8/ Snyk Limited. Software dependencies: How to manage dependencies at scale. Luettu 16.4.2023. https://snyk.io/series/open-source-security/software-dependencies/

/9/ JFrog. 2021. What is a Software Vulnerability. Viitattu 21.4.2023. https://jfrog.com/devops-tools/article/software-vulnerability/

/10/ Encyclopaedia Britannica, Inc. Computer – FORTRAN, COBOL, and ALGOL. Viitattu 25.4.2023. https://www.britannica.com/technology/computer/IBM-develops-FORTRAN

/11/ Cheng, J. 1993. Improving the software reusability in object-oriented programming. Viitattu 25.4.2023. https://dl.acm.org/doi/10.1145/163626.163636

/12/ TechTarget, Inc. Log4j explained: Everything you need to know. Viitattu 23.4.2023. https://www.techtarget.com/whatis/feature/Log4j-explained-Everything-you-need-to-know

/13/ Amazon Web Services, Inc. What is DevOps – DevOps Models Explained. Viitattu 25.4.2023. https://aws.amazon.com/devops/what-is-devops/

/14/ Mullans, A. 2020. Keep all your packages up to date with Dependabot. Viitattu 25.4.2023. https://github.blog/2020-06-01-keep-all-your-packages-up-to-date-with-dependabot/

/15/ Snyk Limited. Snyk User Docs. Viitattu 25.4.2023. https://docs.snyk.io/getting-started/introducing-snyk

/16/ Preston-Werner, T. Semantic Versioning. Viitattu 1.5.2023. https://semver.org/

/17/ Git- ja Github-sanasto. Luettu 16.03.2023. https://www.cs.helsinki.fi/u/hisahi/sanastot/git_github.html

/18/ The Linux Foundation. Cronjob. Viitattu 9.5.2023. https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/

/19/ MITRE Corporation. CVE-2022-42889 – Apache Text Commons Vulnerability. Viitattu 10.5.2023. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-42889

/20/ Atlassian, Inc. History of DevOps. Viitattu 19.4.2023. https://www.atlassian.com/devops/what-is-devops/history-of-devops

/21/ Atlassian, Inc. What is DevOps. Viitattu 19.4.2023.
https://www.atlassian.com/devops/what-is-devops

/22/ Wärtsilä Oyj. History. Viitattu 3.4.2023
https://www.wartsila.com/about/history

/23/ Wärtsilä Oyj. Introduction to Jira – sisäinen asiakirja 2023. Viitattu 8.5.2023.