jamk.fi

Suljettujen paketinhallintajärjestelmien vertailu

Työkalujen vertailu sovelluskehittäjän näkökulmasta

Jony Oinas

Opinnäytetyö
Elokuu, 2020
Liiketalouden ala, ICT-tradenomi

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Oinas, Jony	Opinnäytetyö, AMK	Elokuu 2020
	Sivumäärä **49**	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Suljettujen paketinhallintajärjestelmien vertailu.
Työkalujen vertailu sovelluskehittäjän näkökulmasta.

Tutkinto-ohjelma
Liiketalouden ala, ICT-tradenomi

Työn ohjaaja(t)
Tommi Tuikka

Toimeksiantaja(t)
Poistettu toimeksiantajan pyynnöstä

Tiivistelmä

Opinnäytetyöni tarkoituksena oli tutkia paketinhallintajärjestelmiä JavaScript – kehityksen tueksi. Tutustuin paketinhallintajärjestelmien toimintaperiaatteisiin ja historiaan, sekä vertailin suljetun paketinhallintajärjestelmän pystyttämiseen tarkoitettuja työkaluja sovelluskehittäjän näkökulmasta.

Opinnäytetyö suoritettiin osittain toimeksiantona. Toimeksiantaja - organisaatio halusi kartoittaa sisäisestihallitun NPM – pohjaisen paketinhallintajärjestelmän hyötyjä, sekä mahdollisia haittoja JavaScript – kehityksessä. Organisaatio tahtoi myös pysyä anonyyminä tässä opinnäytetyössä. Tutkimusmenetelmänä toimi vertaileva empiirinen tutkimus, sekä sen yhdistelmät. Vertailu tapahtui kehittäjän näkökulmasta, mutta toimeksiantajaorganisaation arvoja pyrittiin myös huomioimaan tutkimuksen aikana.

Opinnäytetyö havainnollisti paketinhallintajärjestelmien toimintaperiaatteita, sekä porautui niiden pitkään historiaan. Opinnäytetyön tuloksena saatiin vertailu eri työkaluista, joilla sisäisen paketinhallintajärjestelmän pystytys on mahdollista suorittaa.

Vertailu eritteli työkalujen hyviä, sekä huonoja puolia, sekä muodosti yhteenvedon työkalujen käyttökohteista, sekä mahdollisuuksista. Lisäksi opinnäytetyön aikana pystyin parantamaan tietämystäni paketinhallintajärjestelmien toimintaperiaatteista. Eri työkalujen asettaminen paremmuusjärjestykseen on melkein mahdotonta, sillä niiden käyttökohteet ovat niin moninaisia. Organisaatioiden tuleekin siksi pohtia tarkkaan tarpeensa, joita he työkalulta tarvitsevat.

Avainsanat (asiasanat)
NPM, Paketinhallintajärjestelmä, JavaScript, Node, Package Manager, Verdaccio, Nexus, Repository Manager

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Oinas, Jony	Bachelor's thesis	August 2020
		Language of publication: Finnish
	Number of pages **49**	Permission for web publication: x

Title of publication

Comparison of internally controlled package manager – solutions.

Comparison of common tools from the perspective of a programmer.

Degree programme

Business, ICT-tradenom

Supervisor(s)

Tommi Tuikka

Assigned by

Redacted by request of the assigner

Abstract

The goal of this thesis was to study and research different package managers for JavaScript development. I researched the ways package managers work, while also taking a dive into the rich history of the package managers. I compared the different package manager – tools from the perspective of a web developer.

The thesis started as an assignment from a Finnish organization. The assigner – organization wished to stay anonymous during this thesis, which is why their name has been redacted from this document. The user research method was empirical, and the research was done from the perspective of the developer. The needs of the assigner were also considered during the research.

The thesis managed to give insight into the inner workings of the package manager – technology, and shone light into their long history. The resulting data from this thesis allowed a comparison to be made between different package managers.

The thesis compares the different aspects of these tools and attempted to give an overall answer to which private package manager one should use for the given situation. During the thesis process the writer of this thesis also was able to improve their knowledge of the inner workings of the package managers. It is incredibly difficult to rank different tools, as every organization has different needs. That is why every organization must consider all the specific needs they might have before choosing a tool.

Keywords/tags (subjects)

NPM, Paketinhallintajärjestelmä, JavaScript, Node, Package Manager, Verdaccio, Nexus, Repository Manager

Miscellaneous (Confidential information)

Sisältö

Kuviot

1 Johdanto

Opinnäytetyössä tutkittiin suljettujen sekä sisäisesti hallittujen paketinhallintajärjestelmien vaikutuksia moderniin websovelluskehitykseen. Opinnäytetyö suoritettiin osittaisena toimeksiantona suomalaiselle teknologia-alan organisaatiolle, joka tahtoi pysyä anonyyminä tässä opinnäytetyössä. Opinnäytetyössä käytetään tästä lähtien toimeksiantajasta nimitystä *Organisaatio X*. Opinnäytetyön tarkoituksena oli kartoittaa sisäisestihallitun paketinhallintajärjestelmän hyötyjä sovelluskehityksessä. Opinnäytetyössä vertailtiin erilaisia pakettirekistereiden pystytykseen tarkoitettuja työkaluja.

Paketinhallintajärjestelmät ovat nousseet olennaiseksi osaksi modernia sovelluskehitystä. Avoimen lähdekoodin paketit tarjoavat merkittävät määrät resursseja, joiden avulla sovelluskehittäjät pystyvät nopeuttamaan ja tehostamaan työskentelyään. Paketinhallintajärjestelmien suuri suosio perustuu niiden helppoon käytettävyyteen sekä matalaan osallistumiskynnykseen. Avoin lähdekoodi, sekä pakettien aiheuttamat riippuvaisuussuhteet kuitenkin muodostavat myös potentiaalisia tietoturvariskejä. Vuonna 2019 kohdistetut tietoturvahyökkäykset avointen ja suojaamattomien paketinhallintajärjestelmien kautta ovat jo varsin huomioon otettavia seikkoja organisaatioiden tietoturvastrategioissa. Suorien hyökkäysten lisäksi on myös mahdollista, että kolmannen osapuolen kirjoittamassa koodissa on haavoittuvuuksia, jotka korjaamattomana aiheuttavat tietoturvariskin.

Opinnäytetyön tarkoituksena oli kartoittaa sisäisestihallitun paketinhallinta järjestelmän hyötyjä, sekä toteuttaa vertailu erilaisten työkalujen välillä, joiden avulla sisäisestihallitun suojatun paketinhallinnan pystytys nopeutuu ja yksinkertaistuu. Vertailukriteereinä toimivat mm. työkalun lähdekoodin avoimuus, hinta, dokumentaatio, yhteisön laajuus ja tuki, sekä käyttöönoton nopeus ja helppous kehittäjän näkökulmasta.

2 Tutkimusasetelma

2.1 Opinnäytetyön tavoitteet ja rajaukset

Viimevuosien aikana ovat erilaiset paketinhallintajärjestelmät ja valtavat, jopa tuhansista erilaisista pienistä sekä isoista moduuleista koostuvat pakettirekisterit nostaneet päätään web-sovelluskehityksessä. Paketinhallintajärjestelmä, joka noutaa palauttamansa moduulit julkisesta pakettirekisteristä, mahdollistaa nopeamman ja suoraviivaisemman sovelluskehityksen. Kehittäjän ei tarvitse keksiä ja kirjoittaa kaikkea sovelluslogiikkaa itse, vaan hän voi osittain siirtää vastuun ohjelman logiikan toimivuudesta kolmannen osapuolen kirjoittaman koodin varaan.

Opinnäytetyön tavoitteena oli vertailla erilaisia paketinhallintajärjestelmän pystytykseen luotuja työkaluja, sekä porautua paketinhallintajärjestelmien toimintaperiaatteisiin. Toivoin oppivani enemmän jo entuudestaankin tutuista työkaluista opinnäytetyön myötä, sekä mahdollisesti nostamaan esiin mielenkiintoisia huomioita paketinhallintajärjestelmistä tietoturvan näkökulmasta. Opinnäytetyössä ei ollut tarkoituksena luoda uutta teknistä ratkaisua tai tuotetta, jonka avulla paketinhallintajärjestelmän pystytys olisi mahdollista. Tarkoituksena oli vertailla jo ennestään luotuja työkaluja, sekä mahdollisesti luoda johtopäätös eri työkaluille sopivista käyttötarkoituksista.

2.2 Tutkimuskysymykset

Tutkimuskysymykset, joihin tämä opinnäytetyö pyrkii vastaamaan.

1. Minkälaisia työkaluja on olemassa oman paketinhallintajärjestelmän pystyttämisen helpottamiseksi?
2. Mitä paketinhallintajärjestelmää kannattaa käyttää?
3. Mitä potentiaalista hyötyä on suljetusta paketinhallintajärjestelmästä?

2.3 Tutkimusmenetelmät ja keskeiset käsitteet

Tutkimuksessa yhdisteltiin laadullista ja määrällisiä menetelmiä. Opinnäytetyö lähti liikkeelle toimeksiantona, jossa olisi ollut tarkoitus hyödyntää pääasiassa kehitystutkimusta. Kuitenkin opinnäytetyön tarkentuessa pääasialliseksi tutkimusmenetelmäksi valikoitui empiirinen menetelmä, sekä sen yhdistelmät.

2.4 Keskeiset käsitteet

Opinnäytetyöni liittyi ohjelmistonkehitykseen. Tarkemmin ilmaistuna tutkimani aihe sisälsi sekä tietoturvallisia näkökulmia, että ohjelmistotuotannon prosessien tarkastelua.

Opinnäytetyön keskeiset mielenkiinnon kohteet, eli avainkäsitteet olivat:

- NodeJS
- NPM (Node Package Manager)
- Node - moduuli
- Paketinhallintajärjestelmä
- Repositorio
- Linux – palvelin

3 Tutkimuksen toteutus

Tässä kappaleessa eritellään tutkimuksessa odotettavat tulokset, sekä tutkimuksen aikataulu.

3.1 Tutkimuksesta odotettavat tulokset

Tutkimuksen päätteeksi toivottiin, että voitaisiin ehdottaa kehittäjille yksiselitteisesti kannattavin työkalu sisäisen paketinhallintajärjestelmän pystyttämiseen. Lisäksi odotettiin, että opinnäytetyön yhteenvetona saataisiin muodostettua selkeä kuva paketinhallintajärjestelmien toimintaperiaatteista. Kokonaisuudessaan odotuksena oli, että jokaiseen tutkimuskysymykseen pystyttäisiin vastaamaan, sekä että tuloksista olisi hyötyä eri kehitystiimeille sekä organisaatioille.

3.2 Tutkimuksen aikataulu

Opinnäytetyön suoritus aloitettiin 1.11.2019. Opinnäytetyölle annettiin aikaa syksyyn 2020 saakka. Tutkimus tapahtui iltaisin ja viikonloppuisin, sekä aina kun sille vain riitti aikaa. Ongelmia aikataulussa pysymiseen tuottivat koulun ulkopuolinen täysipäiväinen työ, sekä opinnot. Keskimäärin opinnäytetyötä suoritettiin vajaa 10 tuntia per viikko n. 9 kuukauden ajan. Tiedonhakuun, muistiinpanoihin sekä aiheeseen perehtymiseen pyhitettiin n. 3 kuukautta. Loput ajasta menivät itse kirjoittamiseen, sekä muistiinpanojen kokoamiseen, läpikäymiseen ja tarkentamiseen.

4 Tutkimuksen teknologinen tausta

4.1 Paketinhallintajärjestelmien taustaa

Paketinhallintajärjestelmien pitkä historia alkaa jo 90-luvulta, ja siihen kuuluu olennaisena osana linux-käyttöjärjestelmä. David Mackenzien The GNU -projektiin vuonna 1991 luoma *autoconf*, sekä Larry Wallien kehittämä *metaconf* olivat ensimmäisten työkalujen joukossa rakentamassa tietä nykyisille paketinhallintajärjestelmille (Krutisch 2017). Ensimmäiset varsinaisiksi paketinhallintajärjestelmiksi luokiteltavat työkalut, kuten esimerkiksi *dpkg* ja *rpm*, alkoivat nousemaan suosiossa muutaman vuoden jälkeen vuonna 1993, ja osa näistä järjestelmistä on edelleen laajassa käytössä eri linux – distroissa. Vuonna 1998 julkaistun Debian – distron mukana tullut *apt-get* toi mukanaan helpotetun pakettien välisen riippuvuuksien hallinnan (Katz 2017).

Paketinhallintajärjestelmät jatkoivat kehittymistä läpi 2000-luvun alun. Vuonna 1999 julkaistiin PHP – kielelle yksilöity PEAR – niminen paketinhallintajärjestelmä. Tämän jälkeen vuonna 2003 ilmestyi Python – ohjelmointikielelle oma paketinhallintajärjestelmä nimeltään PyPI. Samana vuonna aloitti kehityksen RubyGems, joka julkaistiin vuonna 2004 (Krutisch 2017). Web 2.0 toi mukanaan käsityksen dynaamisesta ja interaktiivisemmasta internetistä. Teknologiat, kuten AJAX ja XML, mahdollistivat verkkosivujen muuntamisen käyttäjäystävällisemmiksi, sekä verkkoselainten kehittyessä myös javascriptin tehokkuus parantui merkittävästi (Wolcott 2019). Tämä johti javascript – projektien monipuolistumiseen, mikä toi mukanaan omat projektien ylläpidolliset haasteensa. Näiden haasteiden ylikäymiseksi julkaistiin vuonna 2009 NPM (lyhenne sanoista Node Package Manager) osana Node.js – kokonaisuutta (About npm 2020). Muutama vuosi myöhemmin vuonna 2014 perustettiin organisaatio nimeltään npm, Inc, jonka tehtävänä on ylläpitää ja valvoa NPM:n valtavaa pakettirekisteriä (About npm 2020).

Facebookin ja Googlen tukema Yarn julkaistiin vuonna 2016 vaihtoehtoisena NPM – klienttinä, ja se on saavuttanut suosiota kehittäjien keskuudessa (Nakazawa, McKenzie & Kyle 2016).

4.2 ECMAScript moduulit ja NodeJS

Javascriptin alkuperäinen käyttötarkoitus oli toimia verkkosivujen ohjelmointikielenä verkkoselaimeen kahlittuna. NodeJS kuitenkin muutti tämän vuonna 2009 verkkoselaimesta erillisen runtime-ympäristönsä myötä, jolloin syntyi myös tarve javascript moduuleille. Koska javascriptia oli alunperin tarkoitettu käytettäväksi pelkästään selaimessa, oli ainoa tapa yhdistää eri tiedostoissa oleva javascript samaan kontekstiin asettamalla jokainen erillinen tiedosto <script> - tagien sisään suoraan HTML – tiedostoon.

```
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>My website</title>
        <script src="js/script1.js"></script>
        <script src="js/script2.js"></script>
    </head>
    <body>
```

Kuva 1: Esimerkki js-tiedostojen liittämisestä html - tiedostoon

NodeJS mahdollistaa javascriptin ajamisen ilman verkkoselainta, mikä poistaa käytöstä myös HTML:n, sekä verkkosivun ylläpitämän kontekstin. Javascriptin modulaarisuudelle siis syntyi uusia tarpeita (Mottaz 2019).

Moduulit mahdollistavat kaksi oleellisesti erilaista ominaisuutta, jotka puuttuvat aikaisemmasta, verkkosivun omaan kontekstiin pohjautuvasta tavasta hallita koko projektin kontekstia.

- Javascript – tiedosto itsessään pystyy ilmoittamaan muille projektiin kuuluville javascript – tiedostoille, mitä funktionaalisuuksia se pystyy tarjoamaan.
- Javascript – tiedosto pystyy myös hakemaan tarvitsemansa toiminnallisuudet muista moduuleista valikoidusti, näin pitäen projektin nimiavaruuden puhtaana, sekä paremmin hallittavana.

```
const exampleModule = require('exampleModule');

let MyObject = function() {
    exampleModule.exampleFunction("This is an example object!");
}

module.exports = MyObject;
```

Kuva 2: Esimerkki moduulien hallinnasta CommonJS - syntaksilla

Oheisessa kuvassa on esitetty esimerkin kautta, kuinka moduulien hyödyntäminen tapahtuu NodeJS-ympäristössä CommonJS (CJS)-syntaksilla. Javascript-tiedostosta käsin voidaan vaatia tarvittavaa moduulia käyttöön require-funktiolla, kunhan vaadittu moduuli ensin mahdollistaa kutsuttavan sisällön käytön. Tämä tapahtuu syöttämällä ulospäin näkyvä sisältö module.exports-objektiin (Mottaz 2019).

CommonJS toi moduulien hallinnan NodeJS-ympäristöön, mutta ongelmaksi muodostui yhtenäisen standardin puuttuminen javascript moduuleista. Tämä kuitenkin muuttui ECMAScript moduulien myötä. ES-moduulit toivat moduulien hallinnan natiiviin javascriptiin, mikä mahdollisti yhtenäisen ja ECMAScript-standardiin pohjautuvat tavan hyödyntää moduuleita (Mottaz 2019). ECMAScript-spesifikaation mukainen toteutus kuva 2:n esimerkistä voitaisiin kirjoittaa oheisella tavalla.

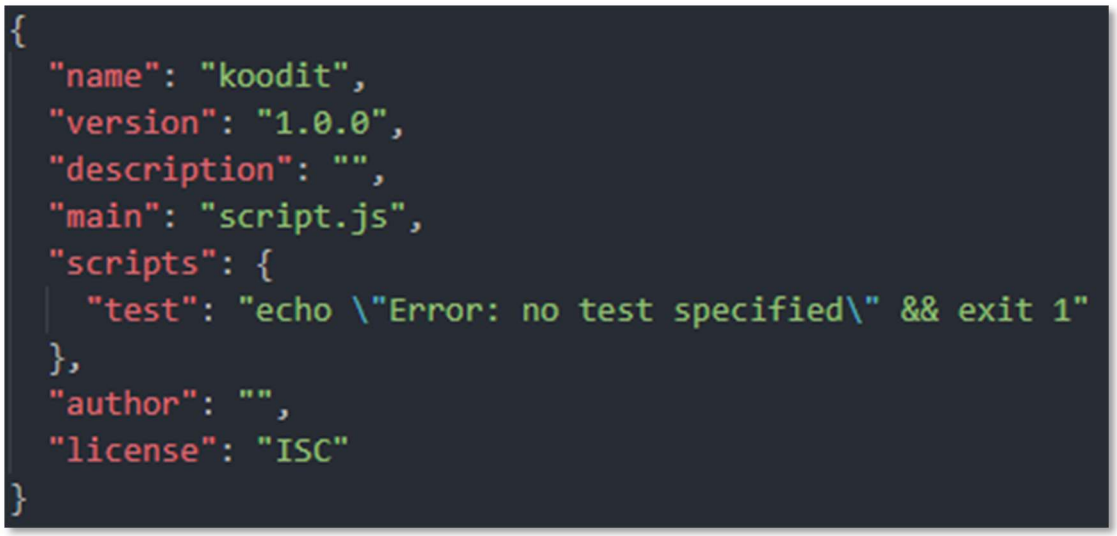
```
import {exampleModule} from '/modules/module.js';

let MyObject = function() {
    exampleModule.exampleFunction("This is an example object!");
}

export { MyObject };
```

Kuva 2: Esimerkki moduulien hallinnasta ECMAScript - syntaksilla

Natiivin moduulin hallinnan käytön tuki NodeJS-ympäristössä on tätä opinnäytetyötä kirjoittaessa vielä kokeiluvaiheessa oleva ominaisuus. ECMAScript moduulien hyödyntämiseksi NodeJS-ympäristössä, täytyy käytössä olla versio 10 tai uudempi Nodesta. Lisäksi javascript-tiedostot täytyy nimetä 'mjs'-päätteellä (Node.js v14.8.0 Documentation 2020).

4.3 Kartoitus paketinhallintajärjestelmän toiminnasta

Paketinhallintajärjestelmien tarkoitus on ylläpitää ja hallita projektien eri pakettien välisiä riippuvuuksia. Package.json-tiedostoon asetetaan projektin paketinhallinnan näkökulmasta oleelliset tiedot, kuten projektin riippuvuudet, sekä projektin nimi ja versio.

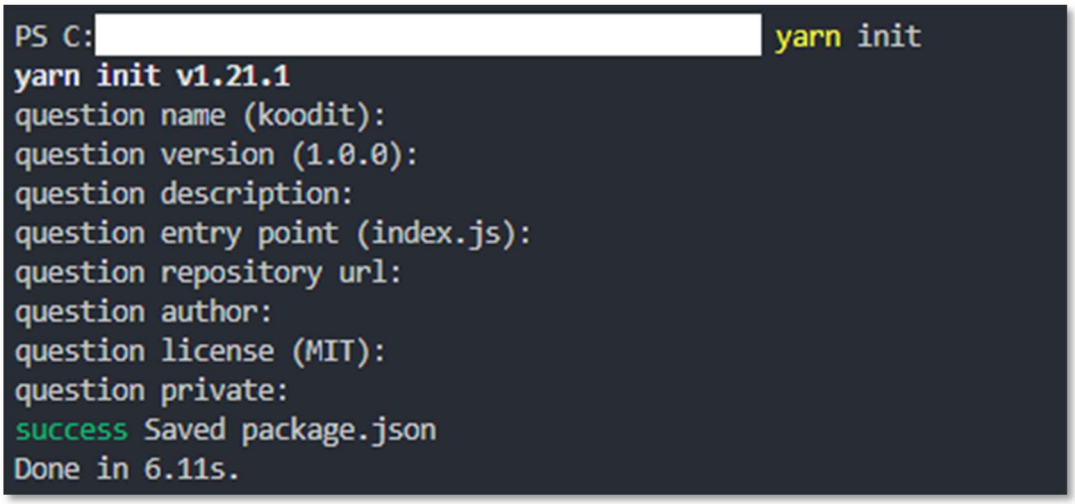


Kuva 3: Esimerkki package.json - tiedoston sisällöstä

Paketinhallintajärjestelmät hyödyntävät myös ns. lock-tiedostoa, jonka tarkoituksena on kertoa paketinhallintajärjestelmälle kaikki projektissa olevien riippuvuuksien versiotiedot. Tämän toiminnallisuuden ansiosta saavutetaan ns. determinismi projektin riippuvuuksien välillä, jolloin voidaan varmistaa, että paketinhallintajärjestelmän projektiin lataamat riippuvuudet ovat versioltaan aina oikein. Tämä helpottaa kehitystyötä merkittävästi (An introduction to how JavaScript package managers work 2016).

4.4 Yarn

Yarn on Facebookin ja Googlen kehittämä ja ylläpitämä NPM - ja Bower-pakettirekisterien kanssa yhteensopiva paketinhallintajärjestelmä. Se julkaistiin 17. kesäkuuta 2016 ratkaisemaan useita eri ongelmia, joita Facebookin ja Googlen kehitystiimit olivat kokeneet projekteissaan NPM:n kanssa. Näitä ongelmia olivat mm. projektien paisuminen kohtuuttoman valtaviksi koodipohjaltaan, sekä kehitysympäristöjen yhteensovittamisen hankaloituminen, mikä johtui NPM:n tavasta hallita projektin riippuvuuksien versiointia (Nakazawa, McKenzie & Kyle 2016).



Kuva 4: Esimerkki työtilan alustuksesta yarn init - komennolla

Yarn toi mukanaan yarn.lock-tiedoston, mikä mahdollisti projektien riippuvuuksien versioinnin paremman hallinnan. Yarn myös asentaa projektin riippuvuudet niin sanottuun offline cacheen. Offline cache on kehitysympäristön projektien riippuvuuksien lokaali varmuuskopio, josta paketinhallintajärjestelmä pystyy hakemaan tarvittavat riippuvuudet internetyhteyden puuttuessa. Yarn offline cachen lokaation pystyy hakemaan ajamalla yarn cache dir -komennon käyttöliittymän terminaalissa (Sayfan 2017).

Kuva 6: Esimerkki yarn offline cachen lokaatiosta

4.5 Bower

Bower on NPM:n kaltainen vuonna 2012 julkaistu paketinhallintajärjestelmä, jonka kehitys on tätä opinnäytetyötä kirjoittaessa jo käytännössä lopetettu. Bower-tiimi edelleen ylläpitää projektia, mutta kehottaa kehittäjiä siirtymään muiden työkalujen, kuten NPM:n, Yarnin, sekä Parcelin käyttöön uusissa projekteissa (About Bower 2020).

Vaikka Bowerin suosio on laskenut merkittävästi viime vuosien aikana, on se jättänyt jälkensä moderniin internettiin, sekä moderniin javascript-kehitykseen. Bower tarjosi ilmestyessään erilaisia uusia ominaisuuksia, kuten esimerkiksi “Flat Depedency Graph”-versionhallinnan. Bowerin mukanaan tuomat ominaisuudet on sittemmin implementoitu osaksi muita työkaluja, tehden Bowerista turhan modernissa webkehitys-stackissa (Hefetz 2017).

```
PS C:\Users\                          bower install jquery
bower jquery#*               not-cached https://github.com/jquery/jquery-dist.git#*
bower jquery#*                  resolve https://github.com/jquery/jquery-dist.git#*
bower jquery#*                 download https://github.com/jquery/jquery-dist/archive/3.4.1.tar.gz
bower jquery#*                  extract archive.tar.gz
bower jquery#*                 resolved https://github.com/jquery/jquery-dist.git#3.4.1
bower jquery#^3.4.1             install jquery#3.4.1

jquery#3.4.1 bower_components\jquery
```

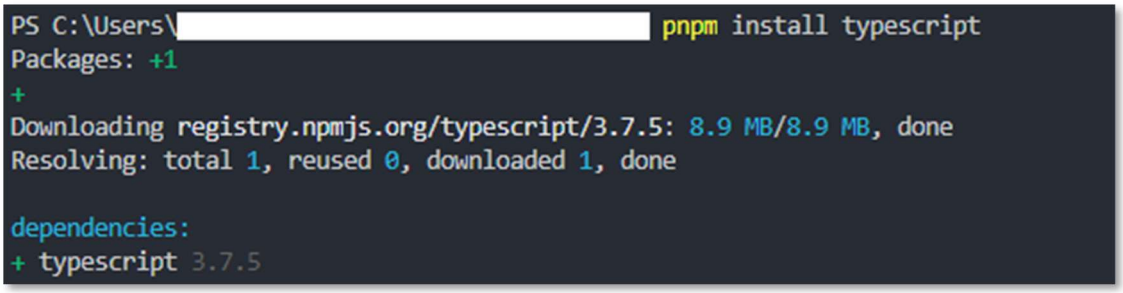
Kuva 7: Esimerkki moduulin asentamisesta Bower - paketinhallintajärjestelmällä

Vaikka Bowerin merkitys javascript – kehityksessä on ollut merkittävä ja sitä edelleen käytetään suhteellisen paljon, tulen jättämään sen pois tämän opinnäytetyön tutkimusosiosta.

Kuva 8: Diagrammi bowerin suosiosta (Lähde: https://bower.io/stats/)

4.6 PNPM

PNPM on npm – paketinhallintajärjestelmään perustuva vuonna 2017 julkaistu paketinhallintajärjestelmä. PNPM syntyi tarjoamaan ratkaisun NPM:n ja Yarnin levynkäytön ongelmiin. PNPM pyrkii myös tarjoamaan nopeamman moduulien asennuksen, kuin NPM ja Yarn. PNPM:ssä kehittäjän lataamat node – moduulit asennetaan käyttöjärjestelmässä yhteen hakemistoon, ns. global storeen, josta PNPM käy hakemassa tarvittavat riippuvuudet eri projekteihin aina tarvittaessa. Näin vältetään saman moduulin lataaminen useaan kertaan, minkä seurauksena kehittäjä voi säästää levytilaa työasemaltaan (Kochan 2017).



Kuva 9: Esimerkki moduulin asentamisesta PNPM - paketinhallintajärjestelmällä

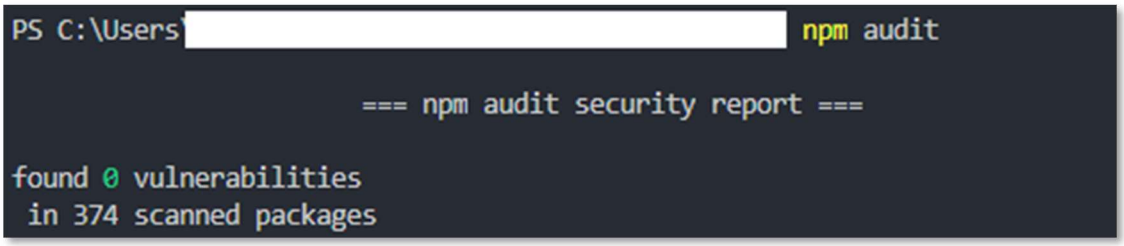
Tämä myös mahdollistaa riippuvuuksien käyttämisen julkisenverkon ulkopuolella, kunhan tarvittavat moduulit löytyvät työasemalta. Asentaminen paikallisesta lokaatiosta vaatii --*offline* – määritteen lisäämisen (Kochan 2017).

Vuonna 2018 julkaistiin PNPM versio 2.0, joka toi takaisin mahdollisuuden käyttää ns. flat dependency -rakennetta projekteissa. Uusi versio myös toi mukanaan työkalun, jonka avulla paketinhallintajärjestelmä pystyy paremmin jakamaan paikallisen node-moduuli-varaston hallintaa. Tämän pitäisi teoriassa tehdä projektien riippuvuuksien hallinnasta ja asentamisesta entistäkin nopeampaa (Kochan 2018)

4.7 Node ja tietoturva

Paketinhallintajärjestelmät syntyivät tarjoamaan kehittäjille tavan hallita moduuleita projekteissaan, sekä ylläpitämään projektien riippuvuuksien versiointia. Noden ja NPM:n julkaisu mahdollistivat kolmannen osapuolen koodin hyödyntämisen yksinkertaistamisen, sekä javascript-moduulien jakamisen standardisoimisen kansainvälisellä tasolla. Kolmannen osapuolen kirjoittaman koodin rajoittamaton hyödyntäminen kuitenkin tekee myös mahdolliseksi erilaisten projektien tietoturvaa heikentävien syntymisen. Node-moduulit muodostavat riippuvuuksien rihmaston, mikä heikentää kehittäjän kykyä ylläpitää projektin koko koodipohjaa merkittävästi (Hunter 2018).

Node-ympäristöön on viime vuosina kehitetty erilaisia tapoja parantaa projektien tietoturvaa. Kehittäjä voi ajaa komennon *$ npm audit fix,* mikä tarkistaa projektin node-moduulien versiotiedot ja lataa tarvittaessa tarvittavat päivitykset projektin moduuleille (npm-audit 2020).



Kuva 5: Esimerkki npm audit - komennon ajamisesta

Kehittäjät voivat myös hyödyntää snyk-nimistä työkalua, joka pyrkii tarkistamaan projektin riippuvuudet tunnettujen haavoittuvuuksien varalta. Kehittäjien tulee myös huomioida ns. hyvät käytänteet projektiensa node moduuliriippuvuuksien hallinnassa. Snyk tarjoaa sivuillaan kymmenen erilaista hyvää käytäntöä, jotka osaltansa lisäävät npm - projektien tietoturvaa. Näitä käytänteitä ovat mm. erilaisten API-avainten ja salasanojen npm-rekisteriin päätymisen estäminen, lock-tiedoston käyttäminen projektissa, sekä *npm audit* -, tai *snyk test* -työkalujen hyödyntäminen (Tal & Picado 2019).

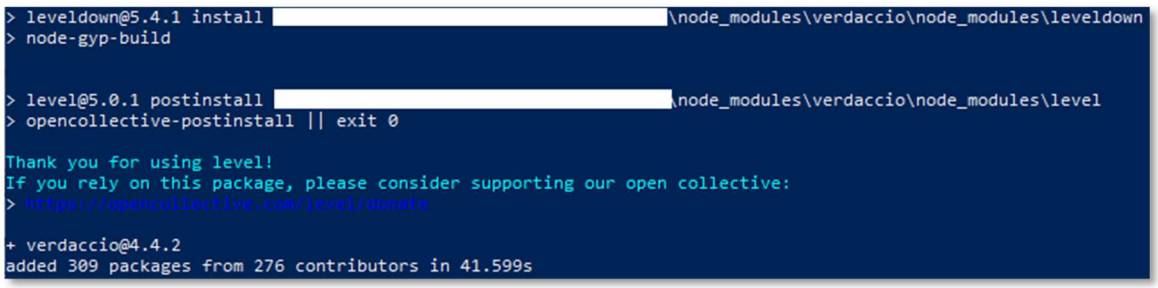
5 Tutkittavat työkalut

5.1 Verdaccio

Verdaccio on yksinkertainen avoimenlähdekoodin työkalu privaatin pakettirekisterin pystyttämiseksi. Verdaccio on suosittu ja aktiivisesti ylläpidetty ratkaisu, joka vaikuttaa tarjoavan skaalautuvuutta isommillekin tiimeille, joilla on tarve rajoittaa pakettirekisterin näkyvyyttä tiimin ulkopuolisille tahoille esimerkiksi tietoturvallisista syistä.

Verdaccio mahdollistaa NPM-rekisterin tallettamisen lokaalisti, mikä vähentää NPM-rekisteristä riippuvaisuuden muodostamia riskejä. Käyttäjät voivat myös “yliajaa” NPM:ssä saatavilla olevia paketteja tekemällä niistä lokaaleja versioita, sekä tallettamalla ne lokaaliin ympäristöön (Verdaccio – projektin viralliset github – sivut 2020).

Tulen tässä opinnäytetyössä tutkimaan näitä väitteitä, sekä vertailemaan Verdaccion hyötyjä, sekä haittoja verrattuna muihin vastaaviin ilmaisiin lokaalin pakettirekisterin pystyttämisratkaisuihin. Verdaccion asennus onnistuu NPM:n kautta erittäin helposti *npm install –global verdaccio* – komennolla.



Kuva 6: Esimerkki Verdaccion asentamisesta *npm install --global* - määreellä

Tämä asentaa Verdaccion globaalisti työasemaan, tehden sen pystyttämisestä helppoa ja lähestyttävää.

5.2 Sonatype Nexus Repository Manager 3

Nexus Repository Manager (NXRM) 3 on Sonatypen kehittämä ja ylläpitämä paketinhallintajärjestelmä, jonka avulla kehitystiimit pystyvät pystyttämään ja rakentamaan organisaation sisäisesti hallitun paketinhallintajärjestelmän. NXRM 3 tarjoaa tuen erilaisille rekisterityypeille, kuten esimerkiksi Apt, Bower, Docker, Maven, PyPi, sekä tietenkin NPM.

NXRM 3 ei ole ilmainen työkalu, mutta pienissä avoimenlähdekoodin projekteissa sitä voi käyttää OSS – lisenssin myötä ilmaiseksi. Tuottoa tavoittelevat tiimit joutuvat ostamaan PRO – lisenssin työkalun käyttöä varten. NXRM 3 on stabiili ja suosittu ratkaisu sisäisen paketinhallintajärjestelmän pystyttämiseksi, ja se tarjoaa laajan rekisteritukensa ansiosta laajennettavuutta tiimeille, sekä organisaatioille (Sonatype Repository Manager 3:n virallinen dokumentaatio 2020).

NXRM tarvitsee Java 8 suoritusympäristön (JRE) toimiakseen. Tämä toisaalta myös mahdollistaa NXRM:n pystyttämisen mihin tahansa ympäristöön, tehden siitä ideaalin vaihtoehdon isoille organisaatioille, joiden mahdollisuudet muokata ympäristöjään uuden työkalun käyttöönottamiseksi saattaa olla rajatumpaa kuin pienillä tai keskikokoisilla organisaatioilla. Sonatype myös pystyy tarjoamaan isoille organisaatioille keskitetympää teknistä tukea ongelmatilanteiden ilmaantuessa. Samaa ei voi sanoa pienemmän mittakaavan avoimenlähdekoodin projekteista, joiden ylläpito perustuu yhteisön hyväntekeväisyyteen, sekä hyväntahtoisuuteen.

Näistä syistä tulen huomioimaan Sonatype Nexus Reposiroty Manager 3:n tämän opinnäytetyön työkalujen vertailuosiossa. Vaikka suurille organisaatioille melkein poikkeuksetta vaaditaan maksullisen PRO-tason lisenssin hankkimista, tulen tämän opinnäytetyön aikana hyödyntämään pelkästään NXRM 3:n ilmaisen lisenssin antamia toiminnallisuuksia, sekä hyötyjä taloudellisista syistä.

6 Työkalujen vertailu

6.1 Vertailuperusteet

Opinnäytetyössä vertailtiin aikaisemmassa osiossa esittelemiäni työkaluja erilaisten ominaisuuksien kautta. Nämä ominaisuudet olivat Organisaatio X:n vaatimia, mutta opinnäytetyön laajuuden, sekä tarkoituksenmukaisuuden takia oli mielestäni myös tärkeää, että vertailtavat ominaisuudet olivat yleisesti hyödyllisiä erilaisille kehitystiimeille, sekä kehittäjille. Siksi vertailuun sisällytettiin seuraavat funktionaalisuudet:

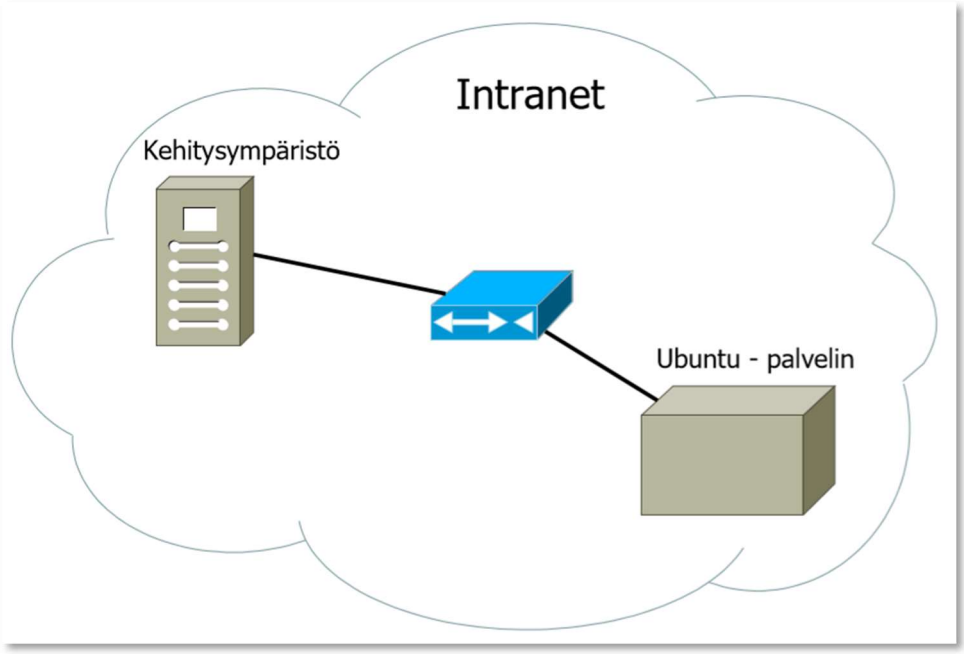
- Yhteisön koko ja saatavilla olevan tuen määrä
- Ympäristön pystytyksen kompleksisuus
- Dokumentaatio
- Maksullisuus ja lisenssit
- Työkalun resurssienkäyttö ja vaatimukset.

Vertailu tapahtui kehittäjän näkökulmasta, ja vertailtavat työkalut asennettiin mahdollisimman geneeriseen ympäristöön. Tällä pyrittiin mahdollistamaan suurin mahdollinen yleistettävyys, jotta vertailun tuloksista olisi mahdollisimman paljon suoraa hyötyä myös yleisellä tasolla, eikä pelkästään Organisaatio X:lle.

Esittelen kehitysympäristön seuraavassa kappaleessa tarkemmin. Käyn myös lävitse työasemalle, sekä virtuaaliympäristölle annetut resurssit.

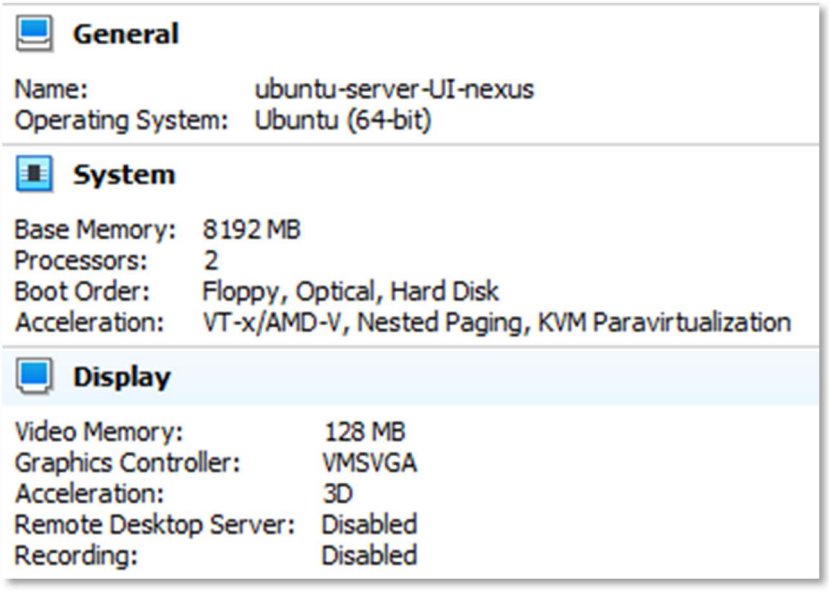
6.2 Testausympäristön esittely

Kehitysympäristössä pyrittiin ottamaan huomioon yleistettävyys, ja sen rakentamiseen ja pystyttämiseen käytettiin ilmaisia avoimenlähdekoodin työkaluja. Palvelin, johon pakettirekisteri asennettiin, asetettiin testaustyöasemasta ulkoistettuun virtualisoituun linux – ympäristöön. Virtualisointityökaluna käytettiin Oraclen VirtualBoxia.



Kuva 12: Kaavio käytetyöstä kehitysympäristön rakenteesta

Palvelimena käytin Ubuntua sen suosion, sekä sen myötä laajasti saatavilla olevan tuen takia. Muita varteenotettavia vaihtoehtoja olivat mm. Fedora ja Redhat, mutta opinnäytetyön laajuuden takia päädyin hyödyntämään mielestäni helpoiten käytettävää työkalua.

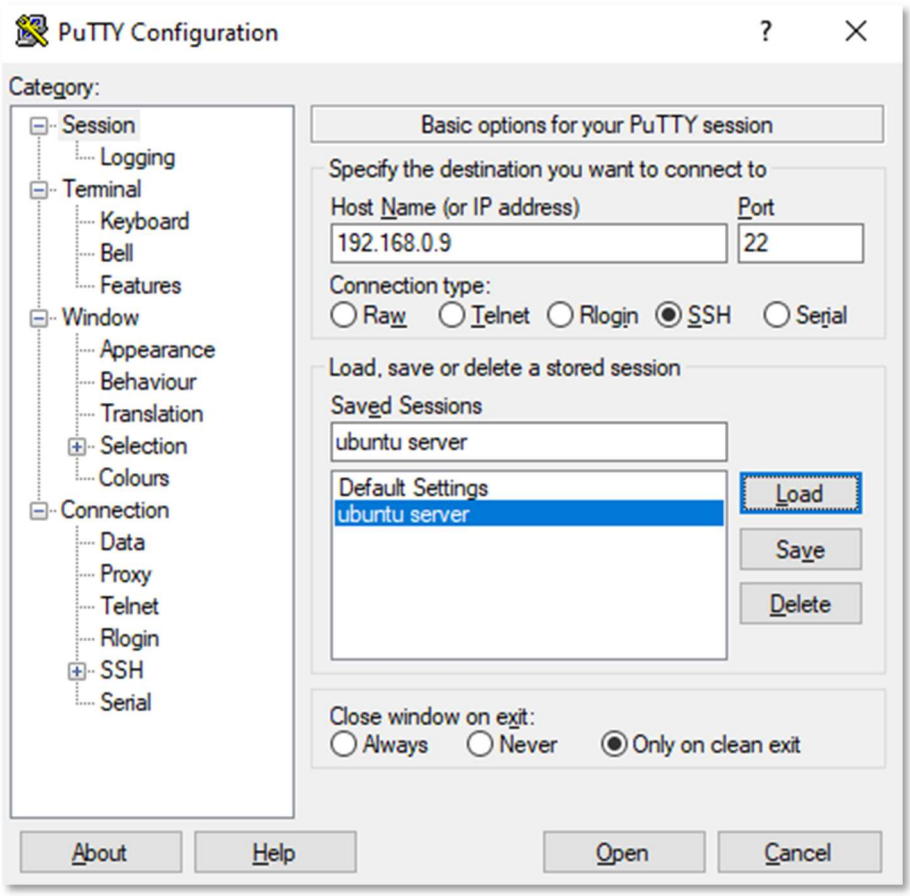


Kuva 13: Kuvakaappaus virtuaaliympäristöjen kokoonpanosta

Palvelimille on asetettu käyttöön 8192 MB muistia, 2 prosessoria, sekä 128 MB videomuistia. Työkoneessa, jossa virtuaaliasema ja testausympäristö ovat asennettuina, on 32 GB muistia, sekä Intel i9-9900K-prosessori. Käytetty Ubuntu distribuutio sisälsi myös graafisen käyttöliittymän CouchDB:n hallinnan helpottamiseksi. Tämä saattaa potentiaalisesti hidastaa palvelimen toimintaa, jos käytettävissä ei ole runsaasti resursseja palvelimen ajamiseksi. Isäntä-kokoonpanon suuren resurssimäärän vuoksi kuitenkin GUI-ominaisuuksien mukanaan tuoma ylimääräinen rasitus palvelimelle ei ole merkittävä aspekti tässä vertailussa. Isäntäkoneen käyttöjärjestelmänä oli Windows 10 Pro 64-bit.

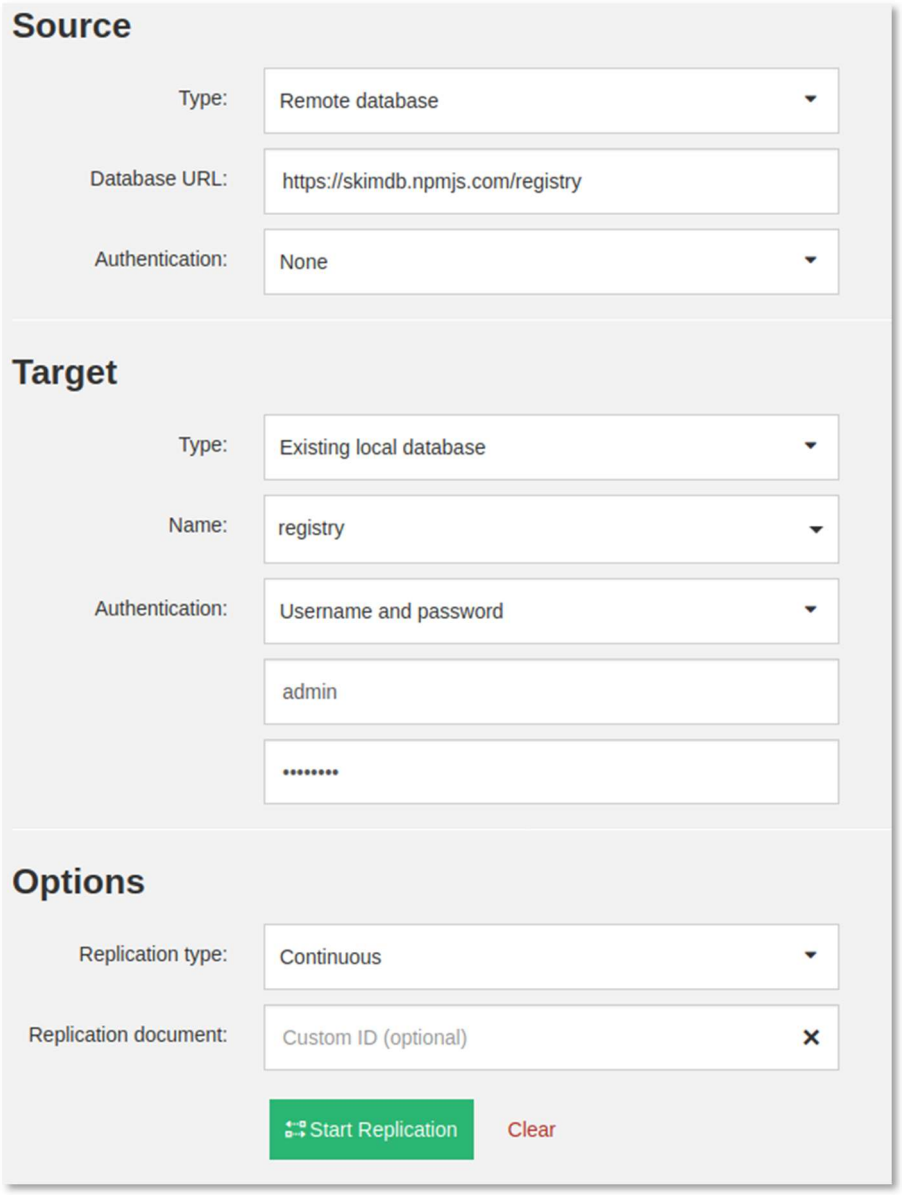
Kuva 14: Kuvakaappaus testausalustan kokoonpanosta

Ubuntu-palvelimelle, josta valitsin käytettäväksi version 19.10, asennettiin instanssi CouchDB-tietokannasta, johon pakettirekisterit asetettiin. Ubuntu-palvelimen käyttö tapahtui kätevästi isäntä – kokoonpanosta käsin PuTTY-nimisen työkalun avulla, jolla pystyy muodostamaan suojatun SSH - yhteyden palvelimiin.



Kuva 15: PuTTY mahdollistaa SSH-yhteyden muodostamisen hallintapalvelimeen

Loin CouchDB-instanssin ubuntu-palvelimelle virtuaaliseen ympäristöön. CouchDB määritettiin hakemaan NPM-rekisterin sisältö 10 minuutin väliajoin. CouchDB mahdollistaa ajoittaisen ja jatkuvan remote-tietokannan replikoinnin lokaaliin tietokantaan. Tämä toiminnallisuus on potentiaalisesti hyödyllinen organisaatioille, jos he haluavat hakea lokaaliin pakettirekisteriin aina säännöllisin väliajoin kaikki uusimmat paketit ja versiot julkaistuista node-moduuleista.



Kuva 16: NPM repositorion replikointi lokaaliin CouchDB - instanssiin

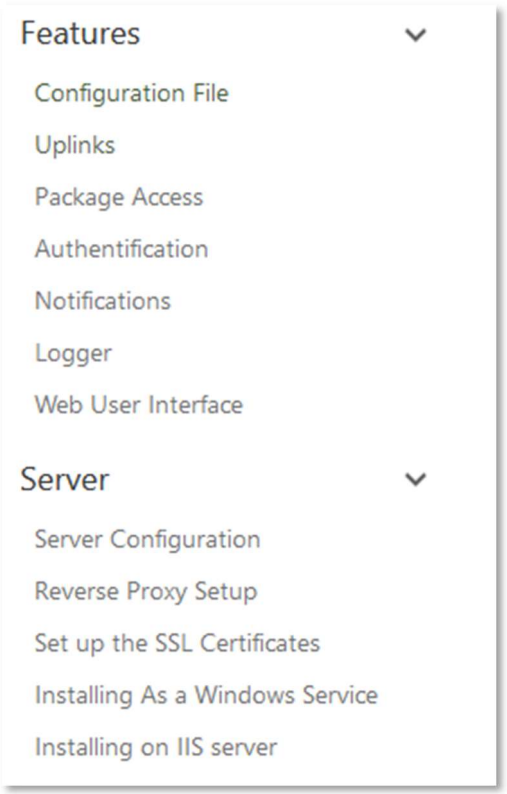
Koko NPM-rekisterin replikointi on täysin mahdollista, mutta vaatii merkittävän määrän tilaa kovalevyltä. Tämän opinnäytetyön laajuuden huomioiden päätin siksi replikoida vain pienen osan kaikista NPM:n moduuleista. Lokaalin CouchDB-tietokannan hyödyntäminen on mahdollista jokaisen vertailtavan työkalun kohdalla. Tämä ei kuitenkaan ole välttämätöntä, sillä Verdaccio ja Nexus Repository Manager tarjoavat omat sisäiset pakettirepositorioiden hallintatavat (Verdaccio Server Configuration 2020).

Jos pakettirekisterin haluaa pystyttää ilman Verdaccion tai Nexus Repositorio Managerin kaltaisia työkaluja, on NoSQL – tietokannan pystytys lokaalin repositorion talletuspaikaksi erittäin hyödyllistä. NPM:n virallinen dokumentaatio myös suosittelee tätä lähestymistapaa (npm-registry 2020).

6.3 Yhteisö ja saatavilla olevan tuen määrä

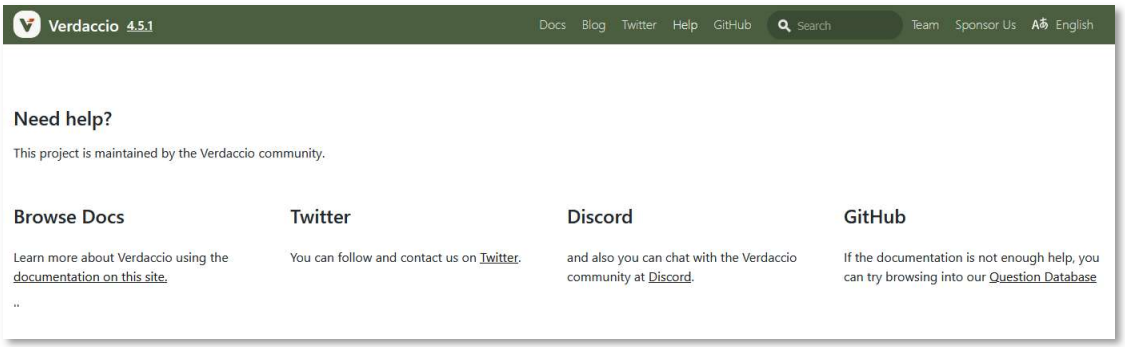
6.3.1 Verdaccio

Verdaccio on MIT – lisenssillä suojattu avoimen lähdekoodin projekti (Verdaccio – projektin viralliset github – sivut 2020). Projekti vaikuttaisi olevan melko hyvin dokumentoitu, ja dokumentaatiota päivitetään aktiivisesti. Verdaccion dokumentaatio kokonaisuudessaan löytyi tätä opinnäytetyötä kirjoittaessa osoitteesta *www.verdaccio.org* . Dokumentaatio kattaa ohjeistukset projektin asentamiselle, sekä lyhyen ohjeen Verdaccio CLI:n konfiguroinnista.



Kuva 17: Kuvakaappaus Verdaccion kotisivuilta löytyvän dokumentaation sisällöstä (verdaccio.org. Official documentation. 20.2.2020)

Pystyin ottamaan verdaccion käyttöön testiympäristössäni virallisen ohjeistuksen avulla todella helposti. Dokumentaatio on selkeästi kirjoitettua, sekä tarjoaa havainnollistavia kuvia eri suoritettavista toiminnoista. Verdaccion kotisivuilta löytyy myös linkit kotisivun ulkopuolisiin virallisiin lähteisiin, kuten Verdaccion viralliseen Discord – ryhmään, sekä Githubissa sijaitsevaan lähdekoodiin.



Kuva 18: Kuvakaappaus Verdaccion kotisivuilta löytyvästä ohjeistuksesta (verdaccio.org. Official documentation. 21.2.2020)

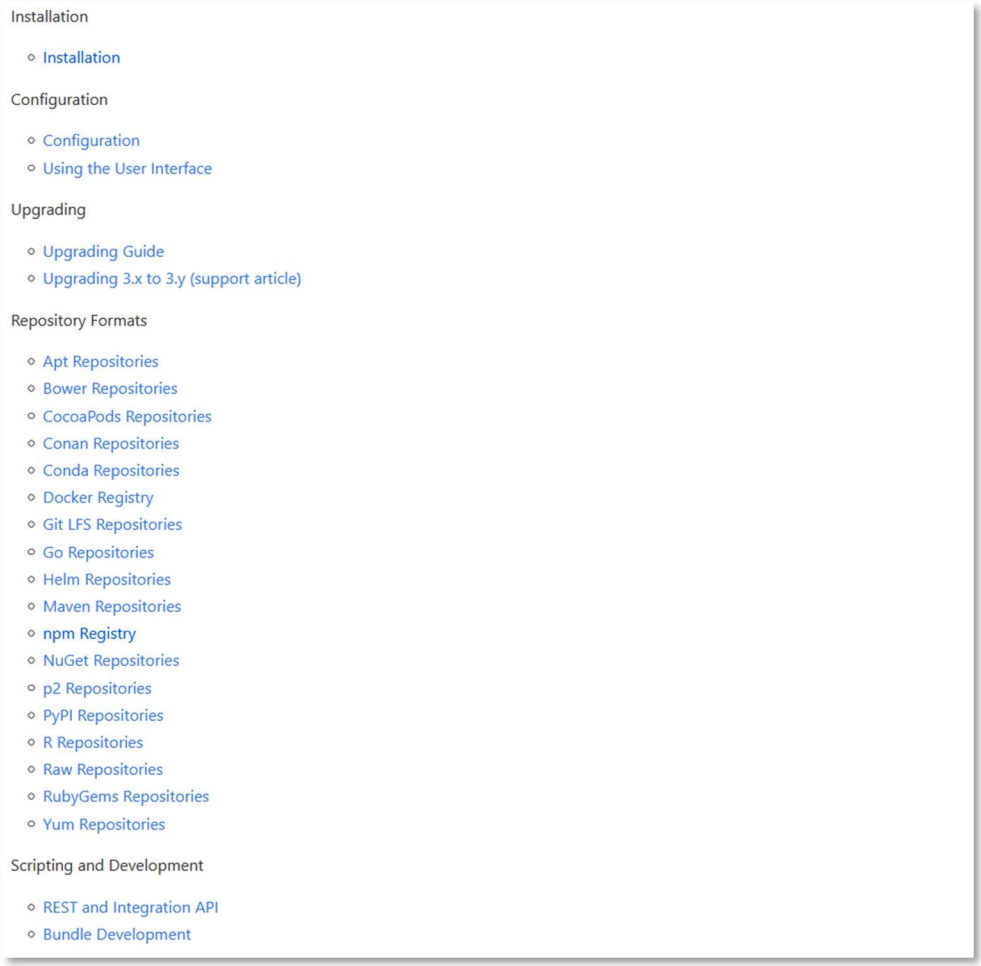
Discord on Skypeä, sekä Microsoft TeamSpeakia monelta osin muistuttava viestittelyyn tarkoitettu ohjelma ja verkkosivu, joka on erityisesti PC – pelaajien laajassa suosiossa. Discordissa on mahdollista luoda ns. discord – palvelimia, joiden avulla ylläpitäjät voivat luoda muista discord – palvelimista eristettyjä yhteisöjä, joissa on omat etikettinsä, teemansa, sekä sääntönsä (Hornshaw 2020). Verdacciolle on luotu oma virallinen discord-palvelin, jossa käyttäjät pääsevät helposti keskustelemaan verdaccion kehittäjien kanssa.

Kuva 19: Kuvakaappaus Verdaccion virallisesta discord – palvelimesta.

Verdacciolle löytyy paljon tutoriaaleja, sekä ohjeistavia artikkeleita. Osa näistä artikkeleista löytyy suoraan Verdaccion virallisten kotisivujen kautta (Verdaccio Articles 2020).

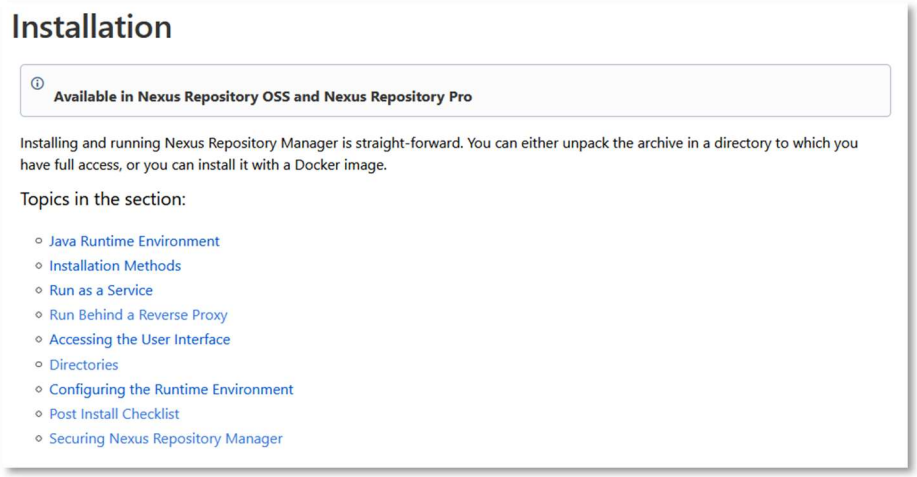
6.3.2 Nexus Repository Manager 3

Nexus Repository Manager 3 on Sonatypen kehittämä ja ylläpitämä osittain avoimen lähdekoodin projekti. Sonatype tarjoaa Nexus Repository Manager 3:sta kattavan ja keskitetyn dokumentaation kotisivuillaan osoitteessa https://help.sonatype.com/repomanager3. Dokumentaatio sisältää ohjeet projektin asentamiselle, sekä konfiguroinnille.



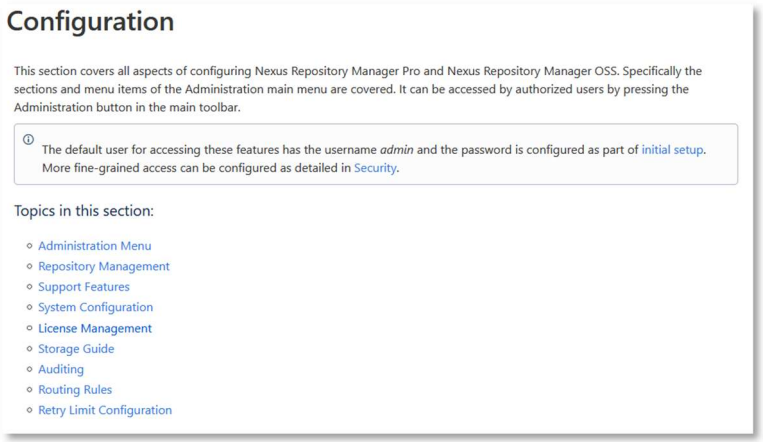
Kuva 20: Kuvakaappaus Nexus Repository Manager 3:n virallisen dokumentaation sisällöstä (help.sonatype.com/repomanager3. Official documentation. 29.2.2020)

Dokumentaatiossa on jaoteltu myös ohjeistus erilaisten pakettirekisteri – formaattien hyödyntämiselle. Myös hyödyllinen on projektin päivitystuki, jonkaavullakehittäjät voivat päivittää käytössä olevan Nexus Repository Manager – versionsa uudempaan.



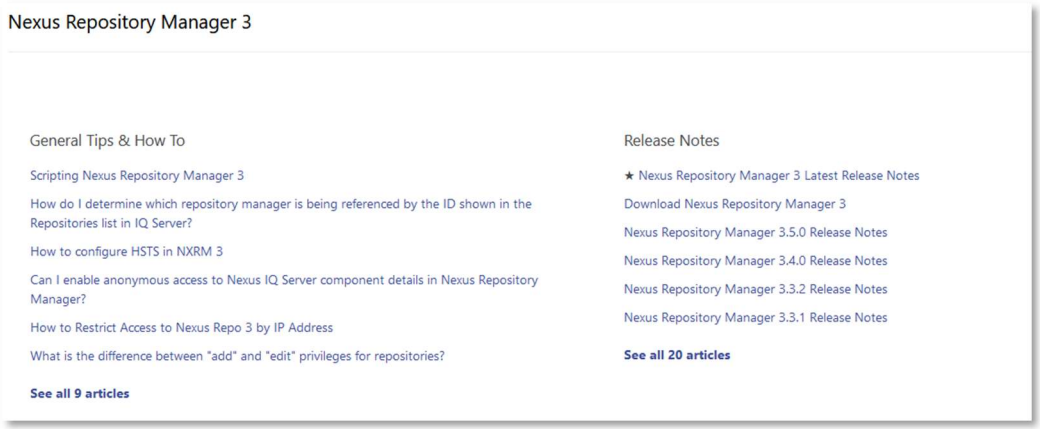
Kuva 21: Kuvakaappaus Nexus Repository Manager 3:n virallisen dokumentaation sisällöstä (help.sonatype.com/repomanager3/installation. Official documentation. 29.2.2020)

Nexus Repository Manager 3:n viralliset konfiguraatio-ohjeet vaikuttavat myös olevan todella kattavat. Ohjeiden avulla pystyin asentamaan Nexus Repository Manager 3:n Ubuntu – palvelimelleni todella helposti, eikä minun tarvinnut turvautua virallisen dokumentaation ulkopuoliseen tukeen.



Kuva 7: Kuvakaappaus Nexus Repository Manager 3:n virallisen dokumentaation sisällöstä (help.sonatype.com/repomanager3. Official documentation. 29.2.2020)

Sonatype tarjoaa projektin lähdekoodin githubin kautta, vaikkakin githubissa projekti on nimetty muotoon Sonatype Nexus Repository Open Source Codebase (Sonatype Repository Manager 3:n OSS – version viralliset github – sivut 2020). Toisin kuin Verdaccio, Sonatype ei kuitenkaan tarjoa suoraa yhteydenpitokanavaa käyttäjien ja kehittäjien välille. Toisaalta hyödyllisiin kolmannen osapuolen ohjeistuksiin on tehty linkitys virallisesta dokumentaatiosta (Sonatype Support Knowledge Base 2020).



Kuva 23: Kuvakaappaus Nexus Repository Manager 3:n virallisen dokumentaation sisällöstä (help.sonatype.com/repomanager3. Official documentation. 29.2.2020)

6.3.3 Yhteenveto

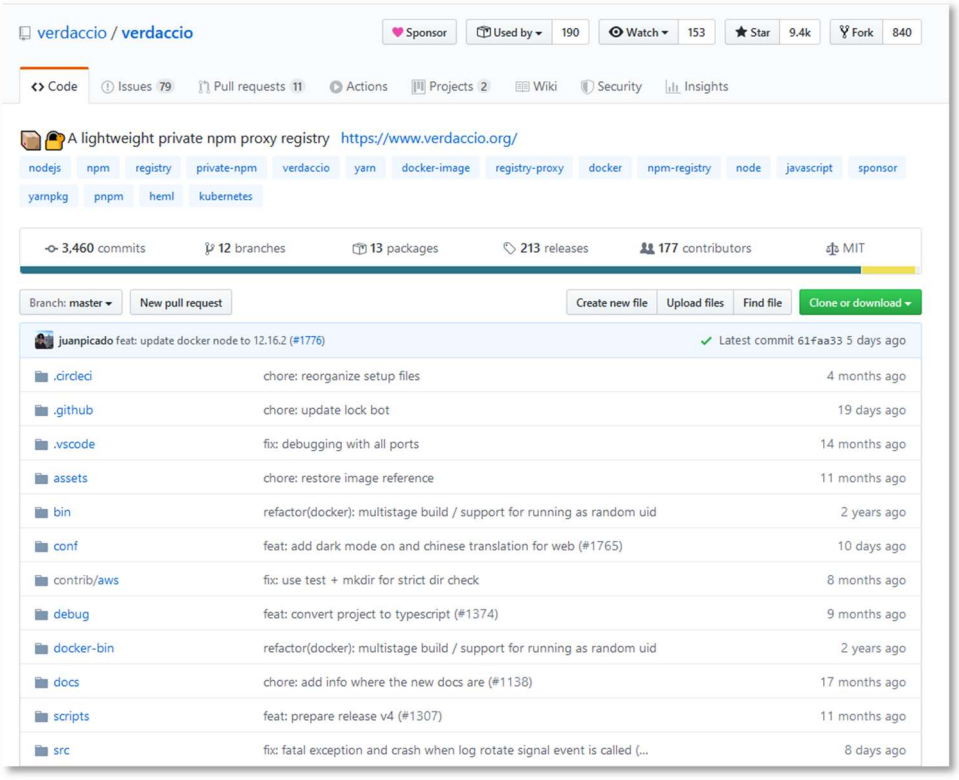
Verdaccio ja Nexus Repository Manager 3 kumpikin sisältävät selkeän dokumentaation, joiden avulla työkalujen käyttöönotto ja ylläpitäminen on helppoa ja nopeaa. Nexus Repository Manager 3:n virallinen ohjeistus kuitenkin on määrältään laajempi kuin Verdaccion, jonka dokumentaatio sisältää välttämättömimmät ohjeet projektin asentamiselle ja konfiguroinnille. Verdacciota konfiguroidessani jouduin turvautumaan kolmannen osapuolen artikkeleihin, kun taas Nexus Repository Manager 3:n virallinen ohjeistus yksinään riitti NRM 3:n käyttökelpoiseksi saattamiseen. Verdaccio – tiimi on kuitenkin pystyttänyt virallisen Discord – kanavan, jonka kautta kehittäjät voivat käydä kysymässä tiimiltä apua erilaisissa ongelmatilanteissa.

6.4 Lähdekoodin avoimuus ja saavutettavuus

Lähdekoodilla tarkoitetaan kaikkea sitä koodia, josta projekti kokonaisuudessaan rakentuu. Lähdekoodin avulla kehittäjät voivat itse käydä läpi käyttämänsä työkalun sisäistä rakennetta, mikä auttaa mm. ongelmien ratkaisemisessa, sekä työkalun toiminnallisuuksien syvällisemmän tason ymmärryksen kehittymistä.

6.4.1 Verdaccio

Verdaccion lähdekoodi löytyy kokonaisuudessaan osoitteesta https://github.com/verdaccio/verdaccio (Verdaccio – projektin viralliset github – sivut 2020). Projekti on päättänyt käyttää lähdekoodin tallennuspaikkana Github – nimistä palvelua.



Kuva 24: Kuvakaappaus Verdaccion lähdekoodista Github – palvelussa (https://github.com/verdaccio/verdaccio. Official Github – page. 18.4.2020)

Github on alusta, johon kehittäjät pystyvät tallettamaan projektejaan git-nimisen versionhallinta-työkalun avulla (Hello World Tutorial by Github 2020). Github mahdollistaa helpon tavan jakaa koodia, sekä toimii myös tapana pitää kaikki projektin koodi julkisena. Muut voivat myös ladata lähdekoodin omalle työalustalleen tarkempaan tarkasteluun näin halutessaan.

6.4.2 Nexus Repository Manager 3

Sonatype on julkaissut Nexus Repository Manager 3:n lähdekoodin Github – palvelussa Verdaccion tavoin, vaikkakin pienillä eroilla. Nexus Repository Manager 3:n lähdekoodi on nimetty hieman epäselkeästi muotoon 'Sonatype Nexus Repository Open Source Codebase'. Lähdekoodi tälle projektille löytyy osoitteesta https://github.com/sonatype/nexus-public (Sonatype Repository Manager 3:n OSS – version viralliset github – sivut 2020). Kyseessä on täysin ilmainen avoimenlähdekoodin versio Nexus Repository Manager 3:sta, josta on riisuttu erinäisiä ominaisuuksia verrattuna saman projektin PRO – versioon (Nexus Repository Version Comparison 2020). Koska projektien välillä on eroja toiminnallisuuksissa, on vaikeaa arvioida, kuinka paljon hyötä tästä koodipohjasta on eri ongelmanselvitystilanteissa.

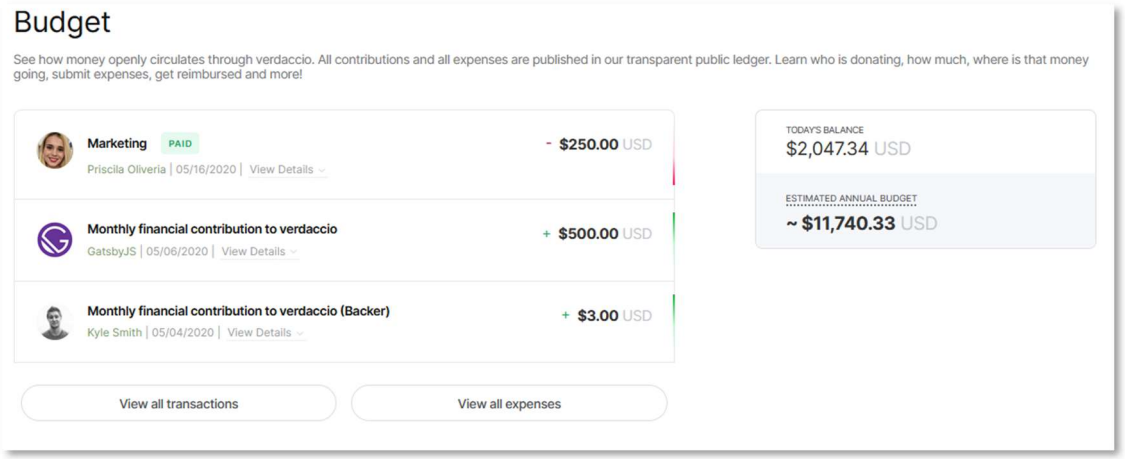
6.4.3 Yhteenveto

Verdaccio tarjoaa koodipohjansa kokonaisuudessaan Github – palvelun kautta. Projektin Github – sivut on myös linkitetty projektin kotisivuille selkeästi. Tämä on ominaista Verdaccion kaltaisille avoimille projekteille. Sonatypen lähestymiskulma on suljetumpi, eikä Nexus Repository Manager 3:n maksullisen version lähdekoodeja pääse lukemaan projektin ulkopuoliset toimijat. Sonatype on kuitenkin julkaissut projektin lähdekoodia osittain julkiseen muotoon Nexus Repository Open Source Codebase – projektin myötä. Projektin PRO ja OSS – versioiden välillä kuitenkin vaikuttaa olevan eroavaisuuksia, mikä tekee OSS – version lähdekoodin hyödyntämisestä PRO – version ongelmien selvittämisessä hieman hankalaa, ellei jopa mahdotonta.

6.5 Hinta ja lisenssit

6.5.1 Verdaccio

Verdaccio on täysin ilmainen ja vapaasti käyttäjän muokattavissa. Verdaccio – projektin aloittaja päätti lisenssoida projektin Massachusetts Institute of Technology - lisenssillä, jota tuttavallisemmin kutsutaan myös MIT-lisenssiksi (Verdaccio Licence 2018). Verdaccion rahoitus tapahtuu vapaaehtoisten rahallisten lahjoitusten kautta Open Collective – yhteisön kautta. Verdaccio – kehittäjäyhteisö pyrkii avoimuuteen, joten rahojen käyttöä projektin suhteen on pyritty erittelemään Open Collectiven kautta (Open Collective – organisaation kotisivuilta löytyvät Verdaccio – projektin virralliset Open Collective – statistiikat 2020).



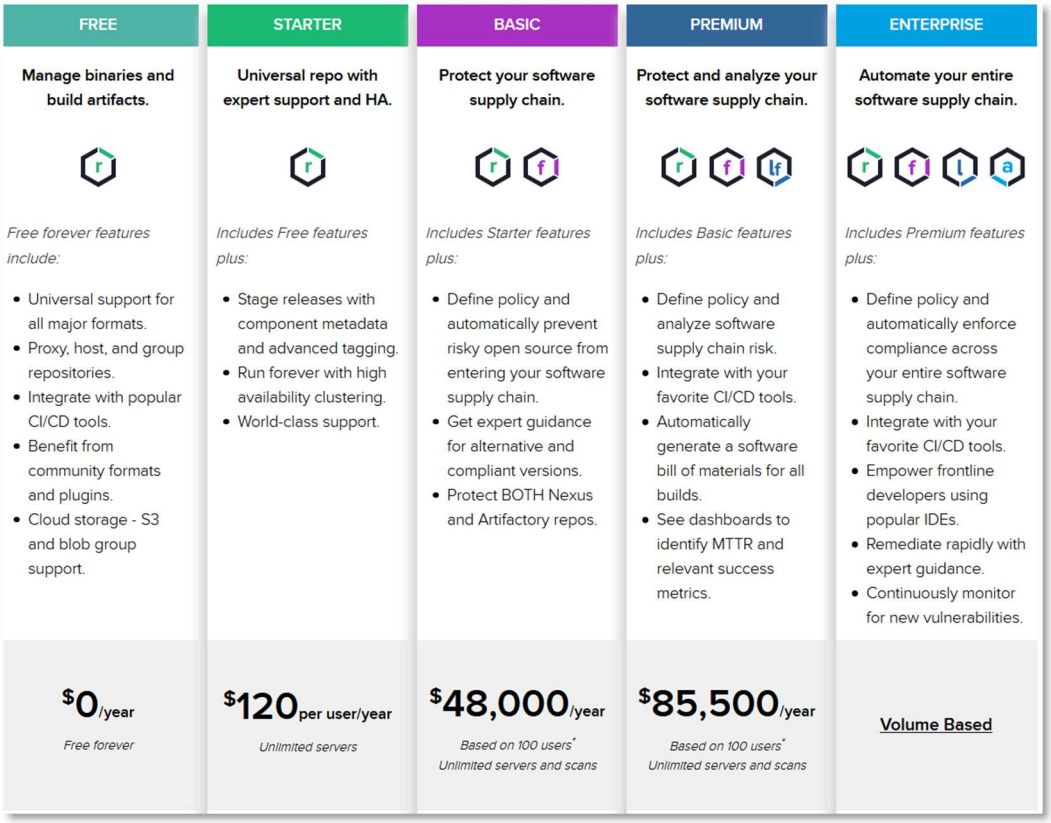
Kuva 25: Kuvakaappaus Verdaccion budjetin erittelystä Open Collective – palvelussa. (https://opencollective.com/verdaccio. Become a contributor for Verdaccio. 22.5.2020)

6.5.2 Nexus Repository Manager 3

Sonatype tarjoaa tuotteestaan viisi erilaista pakettia, joille on asetettu omat hintansa. Eri paketit ovat:

- OSS (Open Source Codebase)
- Starter
- Basic

- Premium
- Enterprise



Kuva 26: Kuvakaappaus Sonatypen kotisivulta Repository Manager 3:n eri versioista ja paketoinneista (https://www.sonatype.com/. Product pricing. 29.5.2020)

OSS versio on ilmainen avoimen lähdekoodin versio Repository Manager 3:sta, joka sisältää tuen välityspalvelimen -, sekä paikallisesti hallittujen paketinhallintajärjestelmien pystyttämiselle. OSS versioon voi myös asentaa Sonatype – yhteisön luomia lisäosia, sekä integroida erilaisia kehityksen tueksi tarkoitettuja CI/CD – työkaluja. Maksulliset versiot sisältävät myös mm. virallisen Sonatypen tarjoaman tuen ongelmatilanteiden selvitykseen, sekä heuristisen palomuuriratkaisun, jonka avulla organisaatio voi vahvistaa järjestelmän suojaa kyberhyökkäyksiä vastaan (Pick a Product 2020).

Sonatypen paketinhallintajärjestelmän maksullisen version hinnat ovat halvimmillaan 120 dollaria per asiakas / vuosi. Kalliimmat paketoinnit sisältävät paljon hyödyllisiä lisäominaisuuksia, mutta ovat myös merkittävästi kalliimpia. Esimerkiksi premium –

lisenssi maksaa sadalle käyttäjälle yhteensä 85,500 dollaria per vuosi (Pick a Product 2020).

6.5.3 Yhteenveto

Kummastakin työkalusta on tarjolla ilmainen versio. Sonatype kuitenkin tarjoaa ilmaisen version lisäksi myös laajempaa tukea maksullisten palveluiden kautta. Isot organisaatiot, joille Sonatypen maksullisten versioiden tuomat lisäominaisuudet tarjoavat merkittävää potentiaalista hyötyä (virallinen tuki ongelmatilanteissa, sekä parannuksia ympäristön tietoturvaan), voivat pohtia maksullisten lisenssien hankkimista organisaatiolle. Ilmaiset ratkaisut, kuten Verdaccio, ovat myös laajempien organisaatioiden käytössä (Who's Using This? 2020). Siksi kehitystiimien tulee pohtia omia tarpeitansa palveluiden suhteen.

6.6 Palvelun pystys ja konfiguraation kompleksisuus

Tässä kappaleessa vertailen Verdaccion, sekä Repository Manager 3:n pystytykseen kuluvaa aikaa, sekä palveluiden konfiguraation kompleksisuutta.

6.6.1 Verdaccio

Verdaccion asennuksen voi tehdä hyödyntämällä joko NPM -, tai yarn – paketinhallintajärjestelmiä. Tämä edellyttää kyseisten työkalujen asentamista kehitysympäristöön, minkä jälkeen verdaccion asennus onnistuu kätevästi oheisella komennolla:

```
jony@jony:~$ npm install -g verdaccio
[..................] / rollbackFailedOptional: verb npm-session 81345008edf0f19
```

Kuva 27: Kuvakaappaus verdaccion asentamisesta NPM:ää hyödyntäen

Oheinen komento asentaa verdaccion globaalisti palvelimelle. Tämä mahdollistaa verdaccio – komennon käyttämisen missä tahansa työaseman sijainnissa terminaalin kautta:

```
$> verdaccio
warn --- config file  - /home/.config/verdaccio/config.yaml
warn --- http address - http://localhost:4873/ - verdaccio/4.5.0
```

Kuva 28: Esimerkki verdaccion konfiguraatiosta 'verdaccio' - avainsanaa hyödyntäen

Verdaccio asetetaan käyttämään sisäistä moduulirekisteri - instanssia ajamalla oheinen komento:

```
npm set registry http://localhost:4873/
```

Kuva 29: Esimerkki lokaalin moduulirekisterin ottamisesta käyttöön verdacciossa

Tämä mahdollistaa sen, että ympäristössä asennettavat moduulit tulevat organisaation sisäisestä rekisteristä, jota organisaatio pystyy itse ylläpitämään. Rekisteriksi voidaan ottaa esimerkiksi jokin NoSQL – tietokanta, kuten CouchDB.

Tämän jälkeen kehittäjän tulee asettaa työasemansa npmrc – tiedostoon oheinen asetus:

```
//.npmrc
registry=http://localhost:4873
```

Kuva 30: Esimerkki työaseman npmrc - konfiguraatiotiedostoon asetettavasta arvosta

Tämä asetuksen myötä työasemassa käytettävä NPM – instanssi pystyy hyödyntämään organisaation sisäisesti hallittua moduulirekisteriä. Vaihtoehtoisesti kehittäjät voivat asettaa saman asetuksen projektikohtaisesti projektin package.json – tiedostossa oheisen esimerkin mukaisesti:

```
{
  "publishConfig": {
    "registry": "http://localhost:4873"
  }
}
```

Kuva 31: Esimerkki projektikohtaisesta rekisteri - konfiguraatiosta

Verdaccio instanssin suorittaminen palvelinympäristössä vaatii myös käytetystä palvelinalustasta riippuvat konfiguraatiot. Linux – palvelimelle, jota tässä opinnäytetyössä käytettiin, vaadittiin oheiset konfiguraatiot. Verdacciolle luodaan ulkoinen käyttäjä, joka pystyy käyttämään organisaation sisäisestihallittua pakettirekisteriä. Tämä tapahtuu ajamalla *adduser* - avainsana.

```
$ sudo adduser --system --gecos 'Verdaccio NPM mirror' --group --home /var/lib/verdaccio verdaccio
```

Kuva 32: Esimerkki adduser - avainsanan ajamisesta ulkoisen käyttäjän luomiseksi verdaccioon

Huomioi, että *adduser* – avainsanan sijaan voi joutua mahdollisesti käyttämään *useradd* – avainsanaa. Asettamalla verdaccion konfiguraatio tiedostoon arvon *listen: 0.0.0.0:4873*, mahdollistetaan kaikkien ulkoisten yhteyksien kuuntelu.

```
# you can specify listen address (or simply a port)
listen: 0.0.0.0:4873
```

Kuva 33: Kuvakaappaus ulkoisten yhteyksien kuuntelun käyttöönottamisesta verdacciossa

Näiden asetusten lisäksi palvelimeen voidaan asentaa myös funktionaalisuuksia, joiden avulla verdaccion käyttö helpottuu. Palvelimelle voi asentaa mm. *forever* – nimisen CLI – työkalun mikä varmistaa, että verdacciota ajetaan palvelimella jatkuvasti ilman keskeytyksiä (Foreversd – projektin viralliset github – sivut 2020). Samoin voidaan asentaa *crontab* – niminen skripti, joka aikaisemmin mainitun forever – skriptin kanssa mahdollistaa verdaccion käynnistämisen, sekä ajamisen automaattisesti aina

jokaisen palvelimen uudelleenkäynnistyksen jälkeen. Forever – CLI:n voisi myös potentiaalisesti korvata *systemd* – nimisellä työkalulla, mutta tätä vaihtoehtoa en ala tässä opinnäytetyössä testaamaan aikarajoitteiden takia (Morel 2017).

6.6.2 Nexus Repository Manager 3

Sonatype tarjoaa kattavan selostuksen palvelun pystyttämiselle ja konfiguraatiolle. Nexus Repository Manager 3:n konfiguraatio ja käynnistäminen tapahtuu UNIX – ympäristössä ajamalla komento *./nexus run* siinä palvelimen lokaatiossa, johon Nexus Repository Manager 3:n paketoitu sisältö on ladattu. Sonatype suosittelee lokaatioksi palvelimen */opt* – hakemistoa, mutta ylläpitäjä voi valita haluamansa lokaation vapaasti ja omien tarpeidensa mukaisesti.

```
./nexus run
```

Kuva 34: Kuvakaappaus komentoriviltä ajettavasta nexus - komennosta, joka konfiguroi ja käynnistää paketinhallintajärjestelmän palvelimessa

Ylläpitäjä pystyy sammuttamaan Nexus Repository Manager 3:n suorittamisen käyttämällä komentosarjaa *CTRL + C* (Installation Methods 2020).

Nexus Repository Manager 3 vaatii toimiakseen Java 8 Runtime Environmentin asentamisen palvelimelle, jotta palvelu toimisi (System Requirements 2020). Kehittäjä voi varmistaa palvelimelle asennetun Java – version ajamalla komennon *java -version* palvelimen komentokehotteessa.

```
java -version
```

Kuva 35: Esimerkki ajettavasta komennosta, jolla kehittäjä pystyy tarkistamaan palvelimelle asennetun java - version tiedot

Nexus Repository Manager 3 kannattaa asettaa käynnistymään automaattisesti palvelimen käynnistyessä. Tämä tapahtuu luomalla ns. absoluuttinen sijainti, joka ohjautuu Nexus Repository Manager 3:n sijaintiin palvelimella. Tämän jälkeen luodaan nexus – käyttäjä, jolla on riittävät oikeudet palvelun käynnistämiseen, sekä ajamiseen. Bin – kansion nexus.rc – tiedostoon tulee asettaa merkkijono *run_as_user=”nexus”*.

```
run_as_user="nexus"
```

Kuva 36: Esimerkki nexus.rc - tiedostoon asetettavasta arvosta

Tämän jälkeen asetetaan symbolinen linkki nexuksen asennuslokaation, sekä palvelimen init.d – tiedoston välille.

```
sudo ln -s /opt/nexus-3.15.2-01/bin/nexus /etc/init.d/nexus
```

Kuva 37: Kuvakaappaus symbolisen linkityksen luomisesta

Run as service – toiminnallisuuden käyttöön ottamiseksi on eri vaihtoehtoja. Valitsin käytettäväksi chkconfig – nimisen työkalun, mutta periaatteessa mikä tahansa muukin ratkaisu olisi käynyt valitsemani työkalun tilalle. Chkconfig otetaan käyttöön ajamalla oheiset komennot palvelimen init.d – tiedoston sijainnissa:

```
sudo chkconfig --add nexus
sudo chkconfig --levels 345 nexus on
sudo service nexus start
```

Kuva 38: Esimerkki ajettavista komennoista, jotta chkconfig - työkalu saadaan otettua käyttöön palvelimella

Nexus kirjoittaa automaattisesti palvelimen tmp – kansioon ns. PID – tiedostoja, johon se kirjoittaa prosessin id:n. Jos Nexus Repository Manager 3:n käynnistymisessä esiintyy ongelmia, eikä nexuksen järjestelmälokeille tulee uusia tulosteita, kannattaa ylläpitäjän käydä tarkistamassa, että palvelimen tmp – kansioon on syntynyt Nexuksen tarvitsema PID – tiedosto. Java suoritusympäristön (Java Runtime Environment) konfiguraatio kannattaa myös suorittaa, jotta pystytetty palvelu saadaan sopivanlaiseksi. Nexus Repository Manager 3:n asennussijainnissa oleva bin – kansio sisältää

nexus.vmoptions – nimisen tiedoston, jonka kautta Nexus Repository Manager 3 lukee JVM (Java Virtual Machine) tarvitsemansa asetukset. Kyseiseen tiedostoon voidaan määrittää mm. halutun muistin määrä, lokitiedostojen sijainti jne.

```
-Xms2703m
-Xmx2703m
-XX:MaxDirectMemorySize=2703m
-XX:+UnlockDiagnosticVMOptions
-XX:+LogVMOutput
-XX:LogFile=../sonatype-work/nexus3/log/jvm.log
-XX:-OmitStackTraceInFastThrow
-Djava.net.preferIPv4Stack=true
-Dkaraf.home=.
-Dkaraf.base=.
-Dkaraf.etc=etc/karaf
-Djava.util.logging.config.file=etc/karaf/java.util.logging.properties
-Dkaraf.data=../sonatype-work/nexus3
-Dkaraf.log=../sonatype-work/nexus3/log
-Djava.io.tmpdir=../sonatype-work/nexus3/tmp
-Dkaraf.startLocalConsole=false
-Djava.endorsed.dir=lib/endorsed
```

Kuva 39: Kuvakaappaus nexus.vmoptions - tiedoston sisällöstä

Tiedostoon asetetaan halutut konfiguraatiot omille riveilleen. JVM:lle allokoidun muistin määrää voidaan säädellä Xms ja Xmx – asetuksilla. Käytetyn http – portin asettaminen tapahtuu asettamalla rivi *application-port=*.

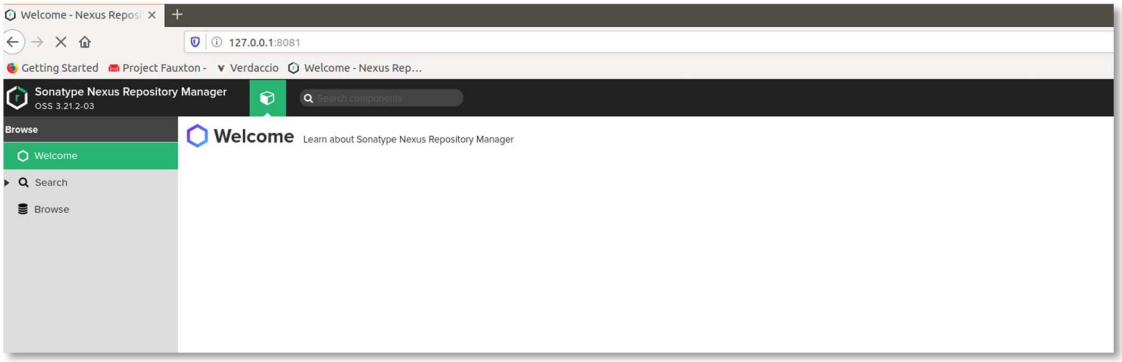
```
application-port=9081
```

Kuva 40: Kuvakaappaus HTTP - portin asettamisesta nexus.vmoptions - tiedostoon

Sonatype tarjoaa kattavan listauksen esimerkkejä muokattavista JVM – asetuksista Nexus Repository Manager 3:n virallisessa ohjeistuksessa.

Graafisen käyttöliittymän käynnistäminen Nexus Repository Manager 3:ssa tapahtuu verkkoselaimen kautta. Kun palvelu on käynnistetty, voidaan graafinen käyttöliittymä

avata menemällä siihen localhost – porttiin, joka on nexuksen konfiguraation yhteydessä palvelulle määritetty. Vakioasetuksilla tämä osoite on *http://localhost:8081/.*



Kuva 41: Kuvakaappaus Nexus Repository Manager 3:n käyttöliittymästä

Vakioporttia voi helposti vaihtaa muokkaamalla Nexus Repository Manager 3:n asennussijainnista löytyvää *nexus.properties* – tiedostoa. Asennuksen jälkeen kannattaa tarkistaa myös tarkistaa seuraavat asiat:

1. Vakioidun pääkäyttäjän salasanan vaihtaminen
2. Nexus Repositorion GUI:n anonyymin käyttäjän pääsyn estäminen
3. Ylläpitäjän email – osoitteen asettaminen
4. Kadonneen salasanan palautus ja SMTP
5. Vakioitujen http ja https – porttien muokkaaminen
6. Nexus Repository Manager 3:n asetusten varmuuskopiointi

Oheiset asetukset kannattaa Sonatypen ohjeiden mukaan vaihtaa, tai ainakin tarkistaa, jos Nexus Repository Manager 3:sta haluaa käyttää tuotannossa (Post Install Checklist 2020). Vakioasetusten tarkastamatta, sekä muuttamatta jättäminen potentiaalisesti vaarantavat palvelun tietoturvan. Jokaisen organisaation tuleekin suorittaa analyysi tarvittavista konfiguraatioista, joita palvelulta edellytetään ympäristön tietoturvan ylläpitämiseksi. Sonatype tarjoaa myös muita tietoturvan parantamiseksi suoritettavia muutoksia Nexus Repository Manager 3:n ohjeistuksessa, kuten esim. luotettujen IP – osoitteiden rajaaminen palvelulle, sekä palvelun käyttäjille suotujen oikeuksien rajaaminen (Securing Nexus Repository Manager 2020).

6.6.3 Docker

Verdaccio ja Nexus Repository Manager 3 kummatkin tukevat dockerin käyttöä. Docker on työkalu, joka mahdollistaa kehitysympäristön virtualisoinnin. Virtualisointi tekee kehityksestä helpompaa isoissa kehitystiimeissä, sillä virtuaalinen kehitysympäristö, joka on identtinen koko kehitystiimin kanssa, poistaa kehitysympäristöjen erilaisuuksista johtuvia konfiguraatio – ongelmia, näin suoraviivaistaen kehitystä merkittävästi (Docker overview 2020). Organisaation tietoturvaa paketinhallintajärjestelmän osalta voi potentiaalisesti parantaa ajamalla Nexus Repository Manager 3:sta ja Verdacciota dockerissa. Tämä mahdollistaa sen, ettei organisaation ulkopuolelta tuleva kohdistettu hyökkäys pääse yhtä helposti tunkeutumaan organisaation muihin samalla palvelimella ajettaviin palveluihin, vaikka hyökkääjä saisikin root – käyttäjän oikeudet docker – hiekkalaatikon sisällä (Securing Nexus Repository Manager 2020).

6.6.4 Yhteenveto

Verdaccio ja Nexus Repository Manager 3 omaavat hyvän dokumentaation, minkä pohjalta ympäristön pystytys ja konfiguraatio onnistuvat helposti. Itse asennus on kummankin työkalun osalta melko suoraviivaista. Verdaccio tarvitsee toimiakseen Noden, kun taas Nexus Repository Manager tarvitsee Java suoritusympäristön pystytyksen palvelimelle. Kummatkin palvelut vaativat asennuksen lisäksi muokkauksia vakioituihin arvoihin, jotta palvelusta tulisi turvallinen tuotantokäytössä. Palveluiden pystytykseen kului yhtä paljon aikaa, eikä kumpikaan palvelu aiheuttanut vaikeuksia tai ongelmia asennuksen ja konfiguraation aikana.

6.7 Järjestelmävaatimukset

Tässä kappaleessa vertaillaan Nexus Repository Manager 3:n ja Verdaccion järjestelmälle aiheuttamaa kuormitusta, sekä palveluille asetettuja järjestelmävaatimuksia.

6.7.1 Verdaccio

Verdaccion järjestelmävaatimuksia ei listata projektin kotisivuilla. Projektin kehittäjien ylläpitämällä Discord – palvelimella käytyjen keskusteluiden pohjalta muodostui

kuitenkin kuva, että Verdaccion vähimmäisvaatimukset järjestelmälle ovat melko matalat. Jotkin tiimit soveltavat Digital Ocean – nimisen palvelun vähimmäisvaatimuksia Verdaccioon. Digital Ocean on yhdysvaltalainen ohjelmistokehitykseen keskittyvä pilvipalvelu (Digital Ocean – projektin viralliset kotisivut 2020).

Digital Ocean määrittelee pienimmän tuetun virtuaaliympäristön kokoonpanoksi seuraavat tiedot:

- Fyysistä muistia
  - 1GB
- Prosessorien ytimien määrä
  - 1

Memory	vCPUs	Transfer	SSD Disk	$/HR	$/MO	
1GB	1vCPU	1TB	25GB	$0.007	$5	Create

Kuva 42: Kuvakaappaus Digital Ocean - pilvipalvelun minimijärjestelmävaatimuksista (https://www.digitalocean.com/pricing/#standard-droplets. Standard droplets. 19.7.2020)

Puutteellisen dokumentaation vuoksi on vaikea muodostaa tarkkaa kuvaa Verdaccion todellisista järjestelmävaatimuksista.

6.7.2 Nexus Repository Manager 3

Nexus Repository Manager 3:sta voidaan ajaa Windows, Linux ja Machintosh – ympäristöissä. Periaatteessa mikä tahansa käyttöjärjestelmä, joka tukee Java suoritusympäristöä (Java Runtime Environment 8), on yhteensopiva Nexus Repository Manager 3:n kanssa (System requirements 2020). Linux ja Macintosh – käyttöjärjestelmiltä kannattaa ottaa pois käytöstä yhtä aikaa avonaisena olevien tiedostojen määrän rajoitus. Linux – ympäristössä tämä tapahtuu asettamalla /etc/security/limits.conf – tiedostoon määrite nofile 65536. Vaihtoehtoisesti saman määreen voi asettaa nexuksen konfiguraatiotiedoston LimitNOFILE=65536 – asetuksella (System requirements 2020).

```
nexus - nofile 65536
```

Kuva 43: Esimerkki limits.conf - tiedostoon asetettavasta nofile – asetuksesta

Sonatype suosittelee ajoympäristössä käytettävän prosessorin sisältävän minimissään neljä ydintä, mutta kahdeksan tai enemmän on suositeltavaa. JVM Heap – muisti kannattaa Sonatype mukaan asettaa maksimissaan neljään gigabittiin. Itse palvelimelle kannattaa asentaa vähintään 8 gigabittiä muistia. Sonatype jakaa suositellut muistimäärät seuraavasti ohjeistuksessaan:

- Pienet organisaatiot ja tiimit, sekä henkilökohtaiset pienpalvelimet (rekistereitä käytössä alle 20, joiden yhteiskoko on n. 20GB).
    - Minimissään 8GB fyysistä muistia
- Keskikokoiset tiimit ja organisaatiot (rekistereitä alle 50, joiden yhteiskoko on n. 200GB).
    - 16GB
- Suuret enterprise – tason organisaatiot (rekistereitä yli 50, joiden yhteiskoko ylittää 200GB).
    - 32GB+

Sonatype tarjoaa ohjeistuksessaan myös lisävinkkejä muistin optimointiin (System requirements 2020).

6.7.3 Yhteenveto

Sonatype Nexus Repository Manager 3:n järjestelmävaatimukset ovat merkittävästi suuremmat, kuin Verdaccion. Vertailua kuitenkin vaikeuttaa Verdaccion puutteellinen dokumentaatio järjestelmävaatimusten suhteen.

7 Vertailun tulokset ja pohdinta

7.1 Tulosten pohdinta

Vertailun tulokset eivät tulleet minulle suurena yllätyksenä. Olin ehtinyt tutustumaan kumpaankin vertailtavaan työkaluun jo aikaisemmin töiden, sekä opiskeluideni kautta, mikä antoi minulle alustavan kokemuspohjan vertailun toteuttamiseksi. Läpi vertailun kävi jatkuvasti ilmi, että Sonatypen kehittämä Nexus Repository Manager 3 tarjoaa varsinkin isoille Enterprise – tason organisaatioille vakaan ja tukevan työkalun, jonka skaalautuvuus on merkittävää. Sonatype tarjoaa eri hintaluokkia lisensseilleen, joista organisaatiot voivat valita haluamansa kokonaisuuden tarvitsemillaan toiminnallisuuksilla. Verdaccio tarjoaa kaikille saman lähtökohdan; työkalu, jota kuka tahansa voi muokata haluamansalaisekseen.

Suurimmat eroavaisuudet näiden kahden työkalun välillä ovatkin maksullisuus sekä avoimuus. Sonatypen tuotteet eivät ole täysin avoimia, vaikkakin avoimenlähdekoodin versioita eri työkaluista on tarjolla Sonatypen virallisella github – sivulla. Jos organisaatio haluaa saada tuotteista täyden hyödyn irti, täytyy tästä maksaa valitun lisenssin mukainen summa. Verdaccio puolestaan on täysin ilmainen, sekä täysin avoimeen lähdekoodiin perustuva yhteisön ylläpitämä projekti. Avoin lähdekoodi tarkoittaa, että projekti on kaikille avoin, sekä siten kohdistetut hyökkäykset ovat mahdollisesti suurempi riski. Avoimuus toisaalta tarkoittaa myös sitä, että laaja yhteisö pystyy paikantamaan ongelmat projektissa nopeasti, sekä tarjoamaan korjauksia, näin parantaen tuotteen tietoturvaa. Mikään ei myöskään käytännössä estä organisaatiota muokkaamasta Verdacciota eteenpäin juuri kyseiselle organisaatiolle sopivaksi. Voitaisiinkin sanoa, että Verdaccion avoimuus tarjoaa käyttäjilleen lyömätöntä joustavuutta rakentaa paketinhallinnastaan juuri sellainen, kuin he haluavat. Sonatype puolestaan tarjoaa valmiiksi rakennetun ja optimoidun ratkaisun, jonka käyttöönotto ja integraatio organisaatioon on yritetty tehdä mahdollisimman helpoksi ja suoraviivaiseksi.

7.2 Luotettavuus

Opinnäytetyössäni suoritettu vertailu tapahtui kehittäjännäkökulmasta, mikä johtui osittain opinnäytetyöni alkuperäisestä luonteesta. Opinnäytetyö lähti liikkeelle toimeksiantona, joka muutti muotoaan ja luonnettaan useaan otteeseen opinnäytetyöprosessin alussa. Käytin vertailussa suuriltaosin hyödykseni työkalujen virallisia dokumentaatioita, sekä työkalujen kotisivujen kautta löytyviä linkitettyjä aineistoja ja materiaaleja. Opinnäytetyöni suurimpana ongelmana oli kuitenkin sen lähestymiskulma tutkimukseen. Vertailu tapahtui kehittäjän, eli opinnäytetyön kirjoittajan näkökulmasta. Vertailu ei tämän takia ole kovin laaja, eikä vertailutulosten yleistettävyys välttämättä ole mielekästä tästä syystä. Hyödynsin vertailussa opinnäytetyön toimeksiantajan painottamia arvoja, joten tulokset ja opinnäytetyön arvo ovat toimeksiantajalle painottuvia. Toivon kuitenkin, että opinnäytetyö tarjoaa myös hyötyä muille organisaatioille, sekä kehittäjille.

Jos voisin tehdä opinnäytetyön uudestaan, pyrkisin huomioimaan vertailussa useamman kehittäjän näkökulman esimerkiksi kyselyiden muodossa. Tähän ei kuitenkaan ollut aikaa, jos halusin pysyä opinnäytetyölle asettamassani aikataulussa.

7.3 Reflektio opituista asioista

Työkalujen vertailu eteni jouhevasti, eikä kummankaan työkalun kohdalla tullut vastaan merkittäviä ongelmia opinnäytetyön aikana. Pystyinkin siksi saattamaan opinnäytetyön loppuun aikatauluni mukaisesti.

Aikataulun hallinta oli opinnäytetyön aikana osittain myös hankalaa. Opinnäytetyöprosessia hidastivat koulun ulkopuoliset ohjelmointityöt, sekä opinnäytetyön ohella suoritettavat pakolliset opinnot. Kuitenkin koen opinnäytetyön olleen opettava ja mielenkiintoinen kokemus, jonka myötä olen saanut paljon lisää tietämystä paketinhallintajärjestelmistä, sekä niiden toimintaperiaatteista. Ohjelmointikokemukseni JavaScriptin parissa myös mielestäni auttoi minua aiheeseen orientoitumisessa, sekä innoitti minua valitsemaan juuri tämän aiheen opinnäytetyökseni.

Jouduin opinnäytetyön aikana käyttämään pääasiassa verkosta löytyviä lähteitä, kuten työkalujen virallisia dokumentaatioita, sekä artikkeleita. Aiheen luonteen takia on vaikea löytää relevanttia, sekä ajantasaista tietoa käsiteltävistä työkaluista. Painetussa muodossa olevat kirjoitukset paketinhallintajärjestelmistä vaikuttivat olevan pääasiassa tutoriaaleja, joiden tarkoituksena oli opastaa käyttäjää työkalujen asentamisessa, sekä ympäristöjen pystytyksessä. Nämä samat ohjeistukset kuitenkin löytyivät myös verkosta, missä niitä oli pystytty päivittämään teknologioiden muuttuessa.

Koen oppineeni tämän opinnäytetyön aikana paljon hyödyllistä ja syvällistä tietoa opinnäytetyössä käsitellyistä aiheista. Uskon syventyneen tietotaitoni olevan minulle hyödyksi nykyisissä työtehtävissäni, sekä myös tulevaisuudessa mm. jatkaessani opintojani ylemmän korkeakoulututkinnon parissa.

8 Loppusanat

Näin opinnäytetyön lopuksi minun pitää mainita, että en pysty varmuudella suosittelemaan toista työkalua yli toisen. Verdaccio ja Nexus Repository Manager 3 tarjoavat samanlaisia toiminnallisuuksia, mutta niiden laajuus, sekä arvot ovat erilaisia. Yksi on maksullinen tuote, kun taas toinen on ilmainen yhteisön ylläpitämä intohimon tuotos. Yksi tarjoaa valmiin ja hiotun paketin Enterprise – tason teknisellä tuella, kun taas toinen vaatii projektin lähdekoodin läpikäyntiä, sekä verkosta löytyviin blogi – kirjoituksiin ja yhteisön tukeen turvautumista. Organisaatioiden tuleekin rakentaa tarkka määritys ympäristönsä vaatimuksista, minkä perusteella tehdä päätös siitä, että mihin työkaluun lopulta päätyy. Toivon tässä opinnäytetyössä esiin nostettujen huomioiden tukevan tätä prosessia.

Organisaation sisäisen paketinhallinnan hallinta rakentuu käytetyistä työkaluista, mutta myös organisaation omista arvoista, sekä käytänteistä. Tietoturvassa usein ihminen on ketjun heikoin lenkki. Organisaatioiden tulisikin korostaa hyviä tietoturvakäytänteitä osana päivittäistä työskentelyä, jota tarkkaan harkitut sekä arvioidut tekniset ratkaisut tukevat.

Lähteet

2016. An introduction to how JavaScript package managers work. Blogi – kirjoitus sivustolla www.freecodecamp.org. Viitattu 5.3.2020. https://www.freecodecamp.org/news/javascript-package-managers-101-9afd926add0a/

2018. Verdaccio Licence. Verdaccio – projektin viralliset github – sivut. Viitattu 20.6.2020. https://github.com/verdaccio/verdaccio/blob/master/LICENSE

2020. About Bower. Bower – projektin virallisilta kotisivuilta löytyvä kirjoitus. Viittaus 11.3.2020. https://bower.io/docs/about/

2020. Digital Ocean – projektin viralliset kotisivut. Viitattu 15.7.2020. https://www.digitalocean.com/

2020. Docker overview. Dockerin virallisilta kotisivuilta löytyvä ohjeistus dockerin käyttöönotosta. Viitattu 10.7.2020. https://docs.docker.com/get-started/overview/

2020. Foreversd – projektin viralliset github – sivut. Forever – työkalun lähdekoodi. Viitattu 27.6.2020. https://github.com/foreversd/forever

2020. Hello World Tutorial by Github. Github Guides – sivuston Hello World – tutoriaali. Viitattu 16.6.2020. https://guides.github.com/activities/hello-world/

2020. Installation Methods. Sonatypen kotisivuilta löytyvä virallinen dokumentaatio. Viitattu 4.7.2020. https://help.sonatype.com/repomanager3/installation/installation-methods

2020. Nexus Repository Version Comparison. Sonatypen kotisivuilta löytyvä vertailu Nexus Repository Managerin eri versioiden välillä. Viitattu 19.6.2020. https://www.sonatype.com/nexus-repository-oss-vs.-pro-features

2020. Open Collective – organisaation kotisivuilta löytyvät Verdaccio – projektin viralliset Open Collective – statistiikat. Viitattu 21.6.2020. https://opencollective.com/verdaccio

2020. Pick a Product. Sonatypen kotisivuilta löytyvä vertailu eri tuotepaketeista. Viitattu 24.6.2020. https://www.sonatype.com/product-pricing?__hstc=160429922.7ab734f9822a9c3981f04695664b9dc0.1528761600092.1528761600093.1528761600094.1&__hssc=160429922.1.1528761600095&__hsfp=2025384311

2020. Post Install Checklist. Sonatypen kotisivuilta löytyvä virallinen dokumentaatio. Viitattu 5.7.2020. https://help.sonatype.com/repomanager3/installation/post-install-checklist

2020. Securing Nexus Repository Manager. Sonatypen kotisivuilta löytyvä virallinen dokumentaatio. Viitattu 5.7.2020. https://help.sonatype.com/repomanager3/installation/securing-nexus-repository-manager

2020. Sonatype Repository Manager 3:n OSS – version viralliset github – sivut. Viitattu 25.5.2020. https://github.com/sonatype/nexus-public

2020. Sonatype Repository Manager 3:n virallinen dokumentaatio. Viitattu 29.3.2020. https://help.sonatype.com/repomanager3

2020. Sonatype Support Knowledge Base. Sonatypen teknisen tuen kotisivut. Viitattu 28.5.2020. https://support.sonatype.com/hc/en-us

2020. System Requirements. Sonatypen kotisivuilta löytyvä virallinen dokumentaatio. Viitattu 5.7.2020. https://help.sonatype.com/repomanager3/installation/system-requirements#SystemRequirements-Java

2020. System requirements. Sonatypen kotisivuilta löytyvät Nexus Repository Manager 3:n järjestelmävaatimukset. Viitattu 12.7.2020. https://help.sonatype.com/repomanager3/installation/system-requirements

2020. Verdaccio – projektin viralliset github – sivut. Viitattu 28.3.2020. https://github.com/verdaccio/verdaccio

2020. Verdaccio Articles. Verdaccio – projektin virallinen dokumentaatio. Viitattu 19.5.2020. https://verdaccio.org/docs/en/articles

2020. Verdaccio Server Configuration. Verdaccio – projektin virallinen dokumentaatio. Viitattu 18.4.2020. https://verdaccio.org/docs/en/server-configuration

2020. Who's Using This? Verdaccio – projektin virallisilta kotisivuilta löytyvä selostus eri Verdacciota käyttävistä organisaatioista. Viitattu 25.6.2020. https://verdaccio.org/en/users.html

About npm. 2020. NPM – projektin virallisilta kotisivuilta löytyvä kirjoitus. Viitattu 17.2.2020. https://www.npmjs.com/about

Hefetz, A. 2017. Bower is dead, long live npm. And Yarn. And webpack. Blogi – kirjoitus. Viitattu 11.3.2020. https://snyk.io/blog/bower-is-dead/

Hornshaw, P. 2020. What is Discord?. Blogi – kirjoitus. 15.5.2020. https://www.digitaltrends.com/gaming/what-is-discord/

Hunter, T. 2018. The Dangers of Malicious Modules. Blogi - kirjoitus javascript -moduulien vaaroista. Viitattu 19.3.2020. https://medium.com/intrinsic/common-node-js-attack-vectors-the-dangers-of-malicious-modules-863ae949e7e8

Katz, J. 2017. A brief history of package management. Blogi – kirjoitus. Viitattu 22.1.2020. https://blog.tidelift.com/a-brief-history-of-package-management

Kochan, Z. 2017. Why should we use pnpm. Blogi – kirjoitus. Viitattu 15.3.2020. https://medium.com/pnpm/why-should-we-use-pnpm-75ca4bfe7d93

Kochan, Z. 2018. pnpm version 2 is out. Blogi – kirjoitus. Viitattu 16.3.2020. https://medium.com/pnpm/pnpm-version-2-is-out-a015268254d5

Krutisch, J. 2017. A brief history of dependency management. Blogi – kirjoitus. Viitattu 27.1.2020. https://depfu.com/blog/2017/03/22/a-brief-history-of-dependency-management

Morel, B. 2017. Creating a Linux service with systemd. Blogi – kirjoitus. Viitattu 2.7.2020. https://medium.com/@benmorel/creating-a-linux-service-with-systemd-611b5c8b91d6

Mottaz, T. 2019. JavaScript Modules: A Brief History. Blogi – kirjoitus. Viitattu 21.2.2020. https://objectpartners.com/2019/05/24/javascript-modules-a-brief-history/

Nakazawa, C., McKenzie, S., Kyle, J. 2016. Yarn: A new package manager for JavaScript. Artikkeli sivustolla www.engineering.fb.com. Viitattu 20.2.2020. https://engineering.fb.com/web/yarn-a-new-package-manager-for-javascript/

Node.js v14.8.0 Documentation. 2020. Node.js virallinen dokumentaatio. Viitattu 29.2.2020. https://nodejs.org/api/esm.html

npm-audit. 2020. NPM – projektin virallinen dokumentaatio. Viitattu 21.3.2020. https://docs.npmjs.com/cli/audit

npm-registry. 2020. NPM – projektin virallinen dokumentaatio. Viitattu 4.4.2020. https://docs.npmjs.com/using-npm/registry.html

Sayfan, G. 2017. 6 Things That Make Yarn the Best JavaScript Package Manager. Blogi – kirjoitus. Viitattu 10.3.2020. https://code.tutsplus.com/tutorials/6-things-that-make-yarn-the-best-javascript-package-manager--cms-29465

Tal, L., Picado, J. 2019. 10 npm Security Best Practices. Blogi – kirjoitus. Viitattu 25.3.2020. https://snyk.io/blog/ten-npm-security-best-practices/

Wolcott, M. 2008. What Is Web 2.0. Artikkeli uutissivustolla www.cbsnews.com. Viitattu 20.2.2020. https://www.cbsnews.com/news/what-is-web-20/