Tampereen yliopisto

Kalle Kumpulainen

NIXOS: JÄRJESTELMÄKONFIGURAATION HALLINTAAN ERIKOISTUNUT LINUX-JAKELU

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintyö
Toukokuu 2019

TIIVISTELMÄ

Kalle Kumpulainen: NixOS: Järjestelmäkonfiguraation Hallintaan Erikoistunut Linux-jakelu
English title: NixOS: A Linux Distribution Specialized in System Configuration Management
Kandidaatintyö
Tampereen yliopisto
Tieto- ja sähkötekniikka
Toukokuu 2019

Työn analyysin kohteena oli hollantilaisen Eelco Dolstran tutkimusprojektina alkanut ja sittemmin väitöskirjaksi asti päätynyt projekti NixOS, joka on järjestelmäkonfiguroinnin hallintaan ja ohjelmistojen jakeluun liittyvien ongelmakohtien ratkaisemiseen erikoistunut Linux-pohjainen käyttöjärjestelmä. Tämän työn tarkoituksena oli selvittää tarkemmin mitä nämä ongelmakohdat tosiasiassa ovat, millaisilla ratkaisuilla niitä on NixOS:ssä ratkaistu ja millaisia eroavaisuuksia NixOS sisältää verrattuna muihin suosittuihin Linux-käyttöjärjestelmiin. Tämän lisäksi tavoitteena oli myös selvittää millaisiin käyttökohteisiin NixOS sopii.

Työssä perehdyttiin aluksi tavallisten Linux-jakeluiden rakenteeseen, josta erityisesti pakettienhallintaan, koska se on NixOS:n poikkeuksellisin ja tärkein komponentti. Näillä tiedoilla siirryttiin NixOS:n esittelyyn, josta käytiin läpi sen historiaa, rakennetta ja erityisiä ominaisuuksia. Tämän jälkeen keskityttiin NixOS:n tärkeimpään ominaisuuteen, eli sen pakettienhallintajärjestelmään. Siitä käytiin läpi sen toimintaperiaate ja erot muihin pakettienhallintajärjestelmiin. Lopuksi esiteltiin vielä erilaisia käyttökohteita, joihin NixOS soveltuu. Tämän ohessa myös pohdittiin erilaisia seikkoja, miksi NixOS ei ole laajoista ominaisuuksistaan huolimatta edelleenkään kovin suosittu.

Työssä saatiin selville millaisia rakenteellisia ratkaisuja NixOS:ssä on jouduttu tekemään, jotta se voisi tarjota hyvin poikkeukselliset ominaisuutensa. Näistä ratkaisuista käytiin erityisen tarkasti läpi pakettienhallintaohjelmiston toteutus, joka on merkittävästi erilainen kuin muiden suosittujen Linux-jakeluiden vastaavien ohjelmien. Lopuksi vielä spekuloitiin useita potentiaalisia syitä NixOS:n vaatimattomaan suosioon, kuten esimerkiksi erittäin pieni kohderyhmä, kaupallisen tuen puute ja helpommin käytettävät kilpailevat ratkaisut.

Avainsanat: Linux, järjestelmäkonfigurointi, pakettienhallinta, ohjelmistojen jakelu, deterministisyys, funktionaalisuus

SISÄLLYSLUETTELO

KUVALUETTELO

OHJELMA- JA ALGORITMILUETTELO

1 JOHDANTO

Linux-ytimen ja muiden välttämättömien järjestelmäohjelmistojen kokonaisuutta kutsutaan Linux-jakeluksi. Näitä jakeluita on syntynyt Linuxin syntymävuoden 1991 jälkeen useita satoja erilaisia [6]. Tyypillisesti tällaiset uudet Linux-jakelut syntyvät esimerkiksi mielipide-eroista tai ratkaisemaan jotakin olemassa olevaa ongelmaa. NixOS on hollantilaisen Eelco Dolstran vuonna 2003 tutkimusprojektina syntynyt Linux-jakelu, joka pyrkii ratkaisemaan järjestelmäkonfigurointiin ja pakettienhallintaan liittyviä monenlaisia ongelmia, kuten tärkeiden hallintatoimintojen turvallisuutta ja deterministisyyttä. Näitä ongelmia ratkaistakseen NixOS on jouduttu toteuttamaan monilta osin hyvin poikkeuksellisesti. [10]

Tässä työssä selvitetään, millä tavoin NixOS poikkeaa tyypillisistä suosituista Linux-jakeluista ja esitellään millaisiin käyttökohteisiin se erityisesti sopii. Työssä keskitytään erityisesti Nix-pakettienhallintaohjelmistoon ja sen käyttämään ohjelmointikieleen, sillä ne mahdollistavat suurimman osan NixOS:n poikkeuksellisista ominaisuuksista. NixOS:ää vertaillaan toteutuksiltaan paitsi muihin Linux-jakeluihin, mutta myös konttiteknologiaan pohjautuvaan Dockeriin, jota käytetään nykyään laajalti ohjelmistokehityksessä. NixOS on kokonaisuutena hyvin voimakas työkalu ja soveltuu myös aivan tavalliseen työpöytäkäyttöön, mutta se ei ole kuitenkaan onnistunut hankkimaan merkittävää suosiota. Tässä työssä pyritään selvittämään, millaisista tekijöistä tällainen vähäinen suosio johtuu.

Luvussa 2 luodaan pohjatiedot Linux-jakeluiden tyypilliselle rakenteelle. Esittelyssä käydään läpi tarkemmin pakettienhallintajärjestelmät, jotka ovat erityisen tärkeitä tämän työn aiheen kannalta. Näitä tietoja käyttäen käsitellään luvussa 3 itse NixOS:ää, josta käydään läpi sen historiaa, rakennetta ja sen mahdollistamia poikkeuksellisia ominaisuuksia. Luvussa 4 syvennytään NixOS:n tärkeimpään osaan, eli sen pakettienhallintajärjestelmään. Tämän jälkeen Luvussa 5 esitellään NixOS:lle soveltuvia käyttökohteita ja verrataan sitä osittain samoihin tehtäviin soveltuvaan Dockeriin. Luvussa perehdytään myös siihen, miksei NixOS ole saavuttanut erityisen suurta suosiota. Lopuksi vielä luvusta 6 löytyy tiivistelmä työn aiheista ja tuloksista.

2 LINUX-JAKELUT

Linux sai alkunsa vuonna 1991 suomalaisen Linus Torvaldsin harrasteprojektina. Muun muassa avoimen lähdekoodin ja yhteisöllisen kehittämisen ansiosta Linux on vuosien varrella kasvanut massiivisen suosituksi käyttöjärjestelmäksi käytännössä kaikkeen muuhun paitsi työpöytäkäyttöön. Nykyään Linux on käytössä esimerkiksi kaikissa 500 nopeimmassa supertietokoneessa, noin 73 prosentissa mobiililaitteista ja noin 66 prosentissa palvelintietokoneista. [12, 18, 26]

2.1 Kuvaus tyypillisen Linux-jakelun rakenteesta

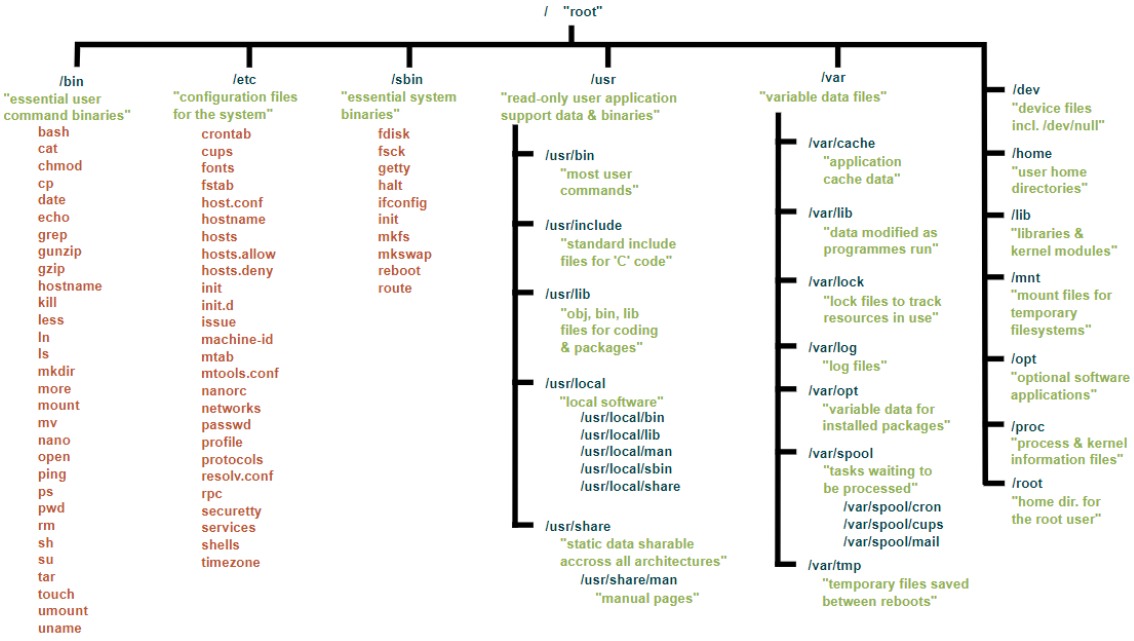
Linux on sellaisenaan ainoastaan käyttöjärjestelmän ydin, eli se sisältää vain muiden ohjelmien ajamiseen ja hallintaan tarvittavat ominaisuudet, kuten suorittimen ohjaus ja muistinhallinta. Tämän takia sitä ei sellaisenaan voi esimerkiksi käynnistää tietokoneella, vaan sen rinnalle tarvitaan muita tärkeitä komponentteja, kuten alkulatausohjelma (bootloader) ja komentotulkki (shell). Nämä ja monet muut ydinohjelmistot (core utilities) ovat alun perin peräisin GNU-projektista, joka on vuonna 1983 yhdysvaltalaisen Richard Stallmanin aloittama, sittemmin yhteisön kehittämä ja avoimeksi muutettu käyttöjärjestelmä ja sen ohjelmistot [24]. Tämän takia nykyään Linuxia kutsutaan myös vaihtoehtoisesti nimellä GNU/Linux.

Linux-ytimen ympärille rakennetusta käyttöjärjestelmästä käytetään tyypillisesti nimitystä Linux-jakelu (distribution, distro). Jakelulle ei ole olemassa mitään tiettyä määritelmää, ja ne ovatkin rakenteeltaan ja ominaisuuksiltaan käyttökohteesta riippuen hyvin erilaisia. Esimerkiksi Alpine Linuxin minimaalisen version asennuspaketti on vain noin 2,5 Mt:n kokoinen, kun taas Scientific Linuxin on noin 9,6 Gt [2, 21].

Jakeluita on siis hyvin monenlaisia ja moniin eri käyttötarkoituksiin. Ne voidaan jakaa hyvin karkeasti kolmeen kategoriaan: käyttötarkoitukseen erikoistuneet alustat, yleiskäyttöiset palvelinalustat ja yleiskäyttöiset työpöytäalustat. Näistä ensimmäinen on kaikista suppein ja esimerkiksi edellä mainittu Alpine Linux edustaa tätä kategoriaa. Tällaisen jakelun tarkoitus on siis tarjota mahdollisimman kevyt pohja, jonka päälle voidaan asentaa halutut ohjelmistot. Palvelinjakelut sen sijaan sisältävät jo hyvin kattavan kokoelman ohjelmistoja, mutta niissä ei tyypillisesti ole graafista työpöytätilaa, vaan niitä ohjataan komentoriviltä. Työpöytäkäyttöön suunnitellut jakelut sisältävät kaikista laajimman kokoelman ohjelmistoja, mukaan lukien työpöytäympäristön, joka sisältää lukuisia graafisia ohjelmia.

Pienimpienkin jakeluiden pohjalla on aina jonkinlainen versio Linuxista. Linux on rakennettu siten, että siitä voidaan helposti karsia tarpeettomia komponentteja pois, mikä mahdollistaa ohjelmiston pienentämisen entisestään. Linux-ytimen parina tarvitaan aina alkulatausohjelma, jonka tehtävänä on käynnistää tietokone, ladata kriittisimmät palvelut ja esimerkiksi näyttää käyttäjälle luettelo koneelle asennetuista käyttöjärjestelmistä, joista voi sitten käynnistää haluamansa helposti. Yksi tunnettu esimerkki alkulatausohjelmista on GNU-projektista peräisin oleva GRUB. Alkulatausohjelman tehtävänä on myös käynnistää ensimmäinen prosessi, joka on nykyään lähes kaikissa jakeluissa nimeltään systemd. Sen tehtävänä on käynnistää muita tärkeitä taustaprosesseja, joita kutsutaan palveluiksi. Systemd itsessään on myös palvelu ja sen vastuualueisiin kuuluu muiden palvelujen hallinnan lisäksi osa kaikkein tärkeimmistä toiminnoista, kuten kellonajan hallinta, verkkoyhteydet ja järjestelmälokitus. [28]

Lähes kaikkien Linux-jakeluiden tiedostojärjestelmien rakenteet ovat keskenään hyvin samanlaisia. Tämä johtuu pääosin niiden yhteisestä esi-isästä, UNIX-käyttöjärjestelmästä, joka määritteli perusrakenteen lähes 50 vuotta sitten. Kuvassa 2.1 on esitelty moderni versio tästä rakenteesta, mukaan lukien esimerkkiohjelmia ja tiedostoja mitä kussakin kansiossa tavallisesti sijaitsee.



Kuva 2.1. *Tyypillisen UNIX- ja Linux-järjestelmän tiedostorakenne. [23]*

Hyvin nopeasti Linuxin syntymän jälkeen sille alettiin myös tehdä erilaisia graafisia työpöytiä. Linux-jakeluissa grafiikan taustalla on erityinen palvelu nimeltään ikkunointijärjestelmä. Se tarjoaa ohjelmille kaikki tarvittavat työkalut grafiikan piirtämiseen. Näistä tunnetuin on nimeltään X, josta nykyään käytössä oleva ilmaisversio on nimeltään X.Org. Linux-jakeluihin on saatavilla lukuisia erilaisia työpöytäympäristöjä. Ne ovat kokoelmia graafisia ohjelmia, kuten itse työpöytä, verkkoselain, office-paketti tai vaikkapa laskin. Nämä muodostavat siis käyttäjälle kaikista näkyvimmän osan Linux-jakelua. Suosituimpia työpöytäympäristöjä ovat esimerkiksi GNOME, Cinnamon, KDE ja XFCE. [28]

Edellä mainitut komponentit yhdessä komentotulkin kanssa muodostavat jo käyttökelpoisen käyttöjärjestelmän, jolla voi periaatteessa suorittaa mitä tahansa ohjelmistoja. Monimutkaisempi käyttö olisi kuitenkin varsin kankeaa ja hidasta, joten ohjelmistojen hallintaa varten kehitettiin pakettienhallintaohjelmistoja. Tällaisten ohjelmistojen tehtäviin kuuluu muun muassa ohjelmien asennus, päivittäminen ja poistaminen. [28]

2.2 Pakettienhallintaohjelmistot tarkemmin

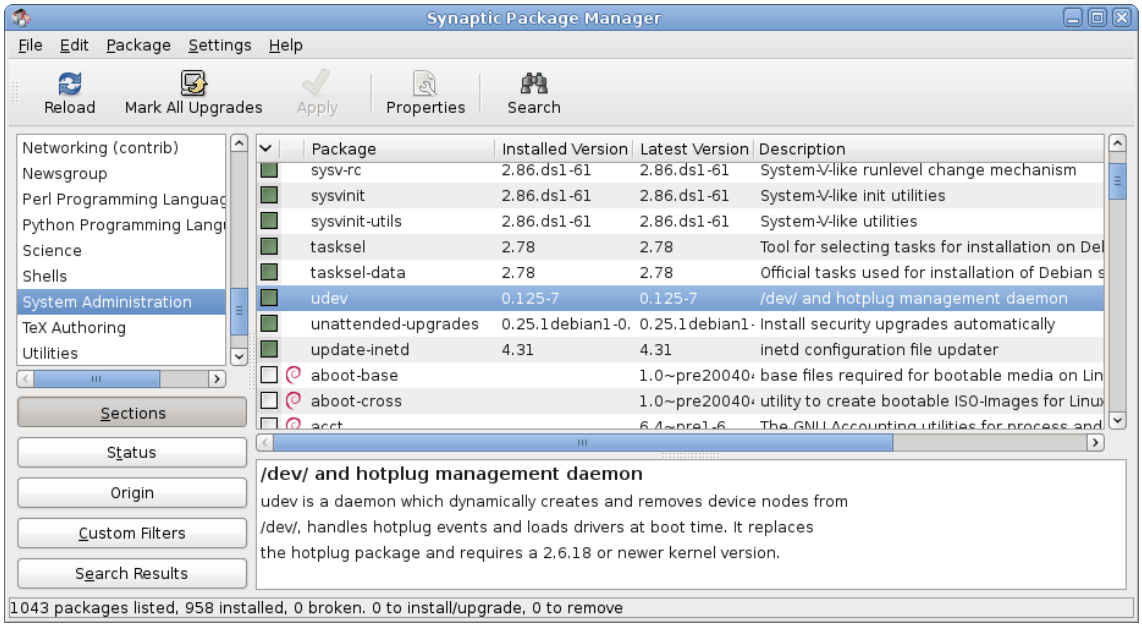
Kolmannen osapuolen ohjelmistojen asentamisen mahdollisuus on yksi tärkeimpiä perusedellytyksiä käyttöjärjestelmälle. Ennen pakettienhallintaohjelmistoja ohjelmien asennus tapahtui tyypillisesti manuaalisesti kääntämällä ohjelman lähdekoodi suoritettavaksi binääritiedostoksi. Yksinkertaisissa tapauksissa tämä olikin varsin toimiva menetelmä, mutta ohjelmien kehittyessä monimutkaisemmaksi ja riippuvaisemmaksi toisistaan tarvittiin tätä helpottavia työkaluja. Ensimmäinen pakettienhallintaohjelmisto, nimeltään dpkg, syntyi ratkaisemaan näitä ongelmia Linuxille vuonna 1994 Debian-jakeluun [11]. Vuosien varrella niitä on syntynyt muun muassa erimielisyyksien takia monia lisää. Muita tunnettuja pakettienhallintaohjelmistoja ovat esimerkiksi rpm, pacman, nix, homebrew ja portage.

Pakettienhallintaohjelmistojen perustoiminnallisuus on keskenään hyvin samanlainen. Ne ylläpitävät paikallista tietokantaa, johon talletetaan metatietoa tietokoneeseen asennetuista ohjelmista. Nämä metatiedot yhdessä ohjelmiston lähdekoodin tai valmiiksi käännetyn binääritiedoston kanssa muodostavat kokonaisuuden, jota kutsutaan paketiksi. Tästä eteenpäin tässä työssä paketeilla viitataan tällaisiin ohjelmistopaketteihin. Ohjelmapakettien metadatoihin kuuluu esimerkiksi tietoja, kuten ohjelman nimi, lyhyt selostus käyttötarkoituksesta, versionumero, erilaisia tarkastussummia, sekä lista riippuvuuksista, joita kyseinen ohjelma vaatii toimiakseen. Pakettienhallintaohjelmistot tyypillisesti ottavat myös verkkoyhteyden etärepositorioon, josta löytyy kymmeniä-, tai jopa satoja tuhansia ohjelmapaketteja. Pakettienhallintaohjelmistoilla voidaan siis hyvin helposti asentaa internetistä ohjelmia samalla asentaen kaikki niiden riippuvuudet. [19]

Ohjelmien helpottuneen asentamisen lisäksi pakettienhallintaohjelmistot tarjoavat myös toisen merkittävän edun järjestelmälle. Niiden tietokannoissa hallitaan ohjelmien riippuvuuksia siten, että mahdollisuuksien mukaan ohjelmat käyttävät keskenään yhteisiä ohjelmia, joita ne tarvitsevat toimiakseen. Tästä havainnollistava esimerkki on esimerkiksi grafiikkakirjasto, jota useat graafiset ohjelmistot tarvitsevat piirtääkseen graafiset elementtinsä. Pakettienhallintaohjelmistojen ansiosta kyseessä oleva kirjasto asennetaan järjestelmään vain ensimmäisellä kerralla sitä vaativaa ohjelmaa asentaessa. Tällainen malli säästää merkittäviä määriä tallennustilaa, selkeyttää järjestelmän kokonaisuutta ja helpottaa myös järjestelmän ajan tasalla pitämistä. Esimerkiksi jonkin ohjelmointivirheen voisi korjata hyvin helposti vain yhteen ohjelmaan, jonka jälkeen koko järjestelmä hyötyy korjauksesta. [19]

Kaikkia pakettienhallintaohjelmistoja voi käyttää sellaisenaan, mutta niitä käytetään tyy-

pillisesti jonkin muun ohjelmiston kautta. Näiden ohjelmistojen tarkoitus on automatisoida sarjoja komentoja ja näin tehdä hallinnasta entistäkin helpompaa. Näitä ohjelmia kutsutaan nimellä front-end ja ne voivat olla komentorivillä operoitavia, kuten Debian-jakelun apt-get, tai graafisia, kuten vaikka Synaptic Package Manager, joka näkyy kuvassa 2.2.



Kuva 2.2. *Synaptic-käyttöliittymä dpkg:lle Debian 5 -jakelussa. [22]*

Eri pakettienhallintaohjelmistot eivät ole toistensa kanssa yhteensopivia, koska niiden metatietoformaatit ovat erilaisia. Tämä tarkoittaa käytännössä sitä, että kehittäjien tai pakettien ylläpitäjien täytyy erikseen paketoida ohjelmat tietylle hallintaohjelmistolle, jotta ne voidaan asentaa sillä. Tämä on osaltaan johtanut vuosien varrella Linux-jakeluiden pirstaloitumiseen moniin eri haaroihin, koska on hyvin yleistä, etteivät kehittäjät paketoi ohjelmia kuin tietyille pakettienhallintaohjelmistoille. Ongelmaksi tällainen muodostuu erityisesti silloin, kun kehittäjä jakaa ohjelmastaan vain suljettua binääritiedostoa, eikä lähdekoodia ole saatavilla. [19]

3 NIXOS:N ESITTELY

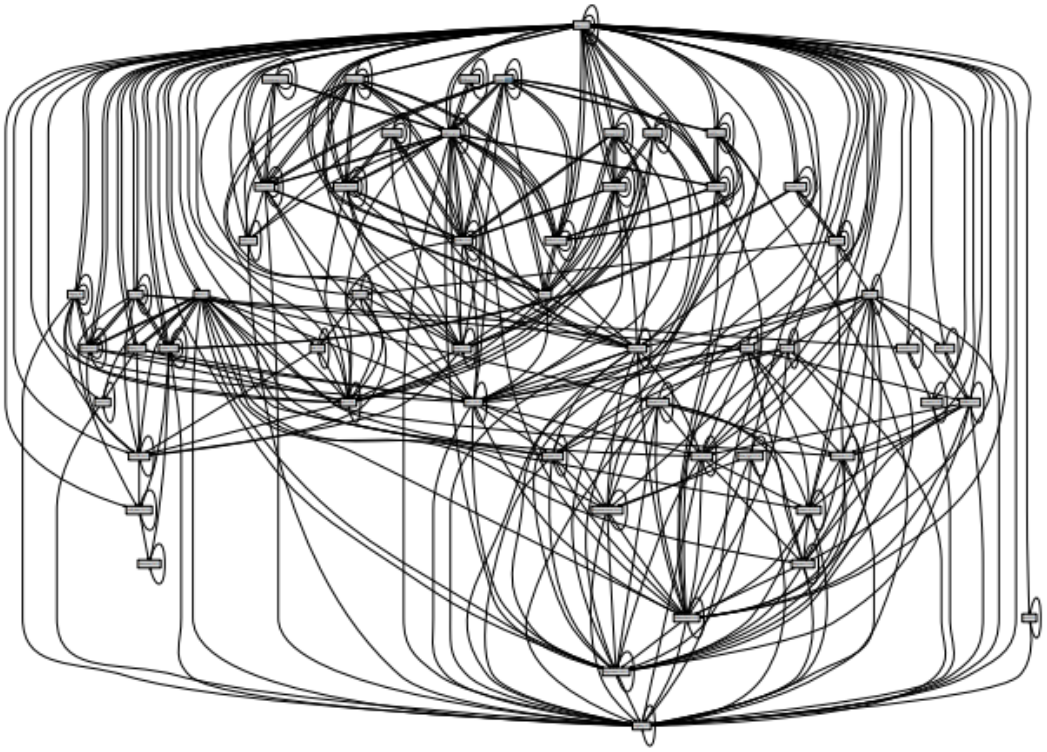
NixOS syntyi alun perin monien muiden Linux-jakeluiden tapaan ratkaisemaan olemassa olevia ongelmia. NixOS:n kohteena oleviin ongelmiin perehdytään myöhemmin tässä luvussa. Käyttöjärjestelmän ideointi alkoi vuonna 2003 hollantilaisen Eelco Dolstran tutkimusprojektina Utrechtin yliopistossa. Kehitystyön tuloksena NixOS:n ensimmäinen julkisesti jaossa oleva PC-versio julkaistiin 23.02.2007 [17]. Dolstra itse kiteytti väitöskirjansa aiheen lyhyesti koskevan ohjelmistojen siirtämistä koneesta toiseen siten, että ne vielä toimivat oikein tämän jälkeen. Tämä onkin fundamentaalinen ongelma, jota NixOS pyrkii ratkaisemaan. Kyseessä ei ollut niin triviaali ongelma, että sen olisi voinut järkevästi ratkaista olemassa olevilla Linux-jakeluilla vain luomalla uusia ohjelmistoja, vaan tarvittiin merkittäviä muutoksia koko jakelun rakenteeseen. [9, 10]

3.1 Miksi NixOS luotiin?

Tyypilliset Linux-jakelut on rakennettu siten, että niiden järjestelmäkonfigurointi noudattaa niin sanottua imperatiivista mallia, jossa järjestelmänhallintatehtävät kuten päivitykset tai konfiguraatioiden muutokset ovat tilallisia (stateful). Tilallinen malli tarkoitta käytännössä sitä, että järjestelmän toiminnan kannalta ehdottomien tiedostojen sisältöä, eli tilaa muutetaan. Tällaisesta mallista syntyy monenlaisia ongelmia, joiden kanssa on vain totuttu vuosikymmenien aikana elämään. Tilallisessa mallissa tärkeät järjestelmätiedostot päivittyvät tuhoisasti siten, ettei vanhaan versioon halutessaan voi enää palata. Myöskään samaa ohjelmaa ei voi ikinä asentaa useampaa kertaa rinnakkain, koska ne käyttävät samoja konfiguraatiotiedostoja. Järjestelmää ei voi myöskään replikoida toisella tietokoneella aina täysin samanlaiseksi, eli deterministisellä tavalla. Ohjelmien päivittäminen on tilallisessa järjestelmässä vaarallinen operaatio, joka voi virheen sattuessa pysyvästi tuhota koko järjestelmän käynnistyskelvottomaksi, koska tiedostoja muutetaan päivityksien aikana jatkuvasti, eikä nämä välitilat ole valideja. [9, 10]

Erityisesti ohjelmien asennuksessa ja hallinnassa oli paljon parantamisen varaa. Perinteisissä järjestelmissä päivitysten jälkeen on aina mahdollisuus, että ohjelmat lopettavat toimimisen, koska niiden riippuvuudet päivittyessään lakkaavat olemasta yhteensopivia ohjelman kanssa. Linux-ohjelmistopakettien metadatatiedoissa ilmoitetuissa riippuvuuksissa on hyvin tyypillisesti ilmoitettu riippuvuusohjelman versionumerolle jokin alaraja. Koska ainoastaan alaraja on ilmoitettu, pakettienhallintaohjelmisto päivittää kyseisen riippuvuusohjelman aina uusimpaan versioon. Joissakin tilanteissa ohjelmat eivät ole tar-

peeksi kauas taaksepäin yhteensopivia, joten tällaisista päivityksistä voi aiheutua yllättäviä järjestelmän rikkoutumisia. Mikäli konfiguraatiota muuttaessa tapahtuu virhetilanne, ei taakse palaaminen ole enää mahdollista, koska vanhat ohjelmat ovat jo päällekirjoittuneet. Tällaiset ongelmat kasautuvat sitä pahemmaksi, mitä enemmän riippuvuuksia ohjelmilla on. Kuvassa 3.1 näkyy Mozilla Firefoxin riippuvuusgraafi sellaisena kuin se oli vuonna 2006. Kuvassa näkyvät vain suorituksen aikana Firefoxin tarvitsemat ohjelmat, joita todellisuudessa tarvitaan vielä paljon enemmän asennuksen aikana. [9, 10]



Kuva 3.1. *Mozilla Firefoxin riippuvuussuhteet (2006) [9]*

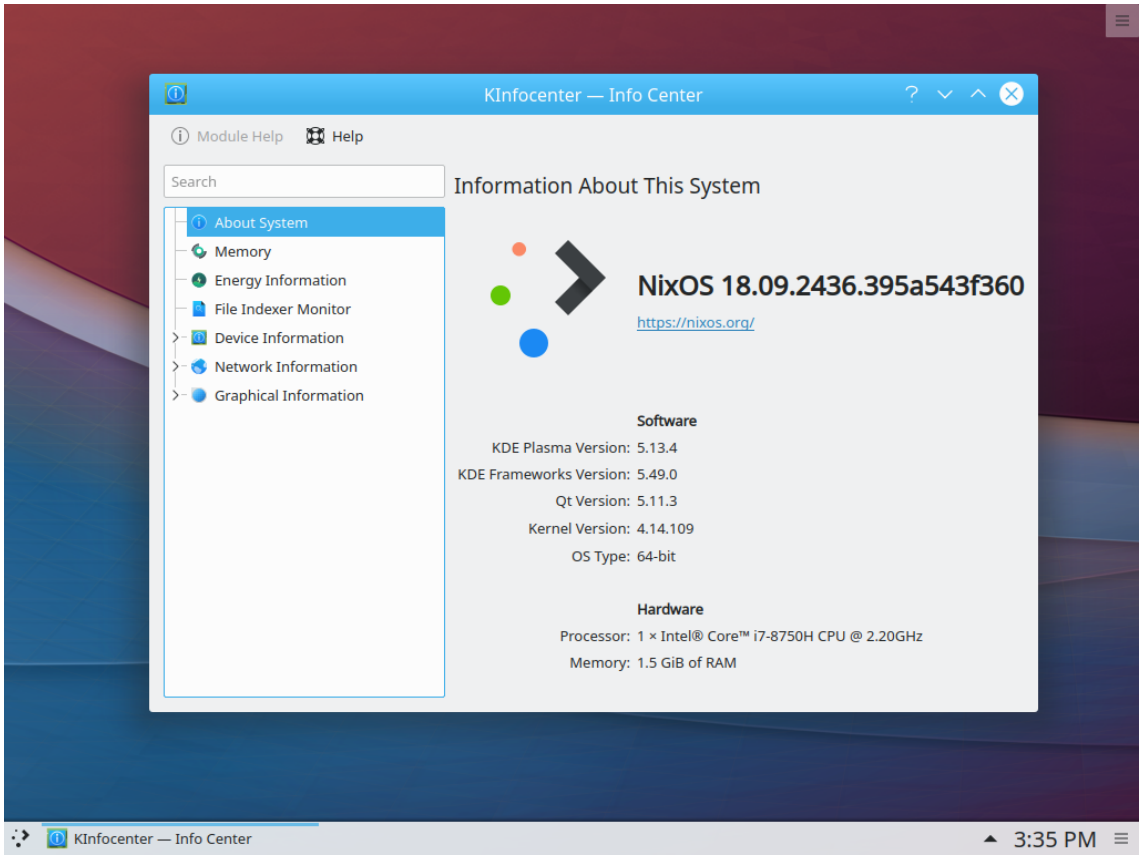
Myöskin pakettienhallintaohjelmistojen ulottumattomissa olevia ongelmia haluttiin ratkaista. On verrattain yleistä, että ohjelmien koodi viittaa suoraan tiedostojärjestelmän kautta johonkin tarvitsemaansa ohjelmaan, kuten vaikkapa perl-tulkkiin polun `/usr/bin/perl` kautta. Tämä on erityisen ongelmallista, sillä näin tehdään viittaus ohjelmaan ilman mitään tietoa sen versiosta tai käyttäytymisestä, mikä voi johtaa monimutkaisissa järjestelmissä todella arvaamattomaan käytökseen. [9, 10]

Näitä ja monia muita ongelmia haluttiin ratkaista Nix-pakettienhallintajärjestelmän ympärille funktionaalisella ohjelmointikielellä rakennetulla NixOS:llä. Tehtyjen radikaalien ratkaisujen ansiosta ohella syntyi myös monenlaisia poikkeuksellisia ominaisuuksia, joita ei voisi tilallisen mallin takia olla perinteisissä käyttöjärjestelmissä.

3.2 Komponentit ja rakenne

NixOS käyttää hyvin pitkälle samoja peruskomponentteja kuin muutkin modernit yleiskäyttöiset Linux-jakelut. Sen pohjalla on aivan tavallinen Linux-ydin, jota ei ole tarvinnut muokata NixOS:n tarpeisiin. NixOS käyttää nykyään järjestelmää hallitsevaa systemd-palvelua, kuten suurin osa muistakin moderneista Linux-jakeluista. [1]

NixOS:n laaja asennuspaketti sisältää X11-näyttöpalvelimen ja oletuksena KDE Plasma -työpöytäympäristön. Kuvassa 3.2 näkyy NixOS:n kokeilemiseen tarkoitetun valmiiksi asennetun VirtualBox-version työpöytänäkymä oletusasetuksilla. Kuvassa näkyy myös KDE:n Info Center -ohjelma, jossa näkyy perustietoja ohjelmistoversioista. NixOS:n ohjelmistorepositoriossa on saatavissa tarvittaessa kaikki muutkin tunnetut työpöytäympäristöt.



Kuva 3.2. *NixOS 18.09 työpöytänäkymä.*

Yksi NixOS:n poikkeuksellisimmista osista on sen standardista poikkeava tiedostojärjestelmän hierarkia. Kuvassa 2.1 näkyvä tyypillinen UNIX-tiedostohierarkia on NixOS:n käyttämän tiedostohierarkian perustana, mutta siitä on jouduttu muuttamaan monia fundamentaalisia osia, jotta se sopisi paremmin NixOS:n käyttötarpeisiin. Tyypillisesti järjestelmään asennetut ohjelmat on jaettu moniin eri kansioihin, joihin ohjelmat jaetaan niiden vastuualueiden ja käyttäjien mukaisesti. Näistä ei ole olemassa oikeasti kattavia sääntöjä, joten tästäkin on vuosien saatossa aiheutunut pirstaloitumista eri Linux-jakeluiden välillä. On kuitenkin monia sääntöjä, joista on pidetty hyvin kiinni vuosien varrella. Esi-

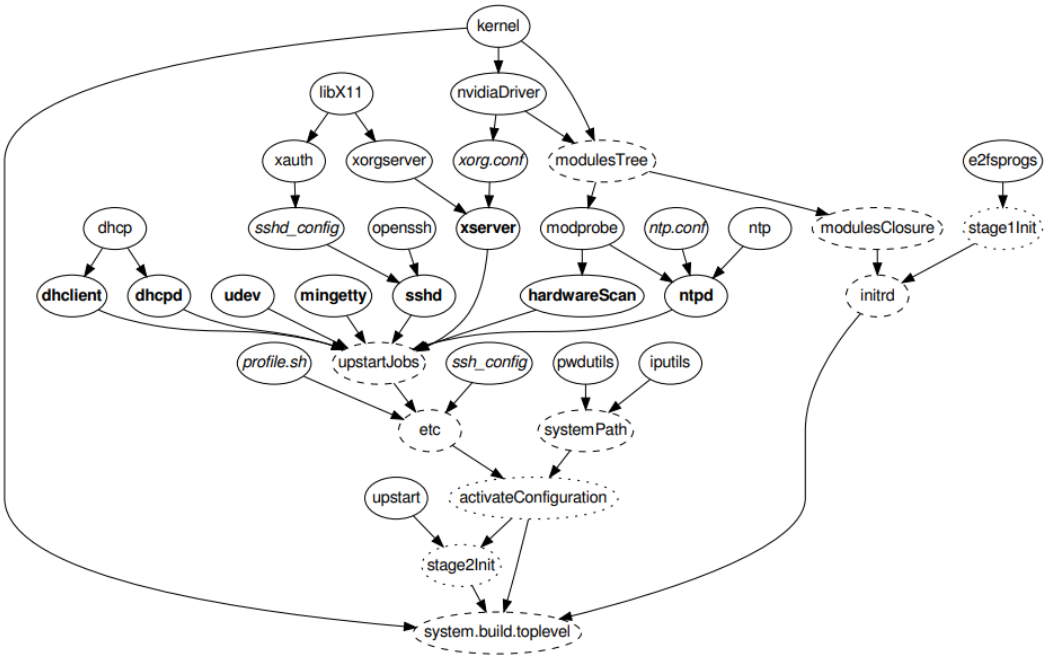
```
1  { config, pkgs, ... }: {
2    imports = [
3      ./hardware-configuration.nix
4      ./kde.nix
5    ];
6
7    boot.loader.systemd-boot.enable = true;
8
9    services.sshd.enable = true;
10
11   environment.systemPackages = with pkgs; [
12     bash
13     vim
14     htop
15     openvpn
16     docker
17     blender
18     wine
19     calibre
20     krita
21     steam
22     skype
23     spotify
24   ];
25 }
```

Ohjelma 3.1. *Esimerkki configuration.nix-tiedostosta. [9]*

merkiksi `/bin` -kansioon asennetaan käyttäjän tarvitsemat välttämättömät apuohjelmat, kuten vaikkapa kansion tiedostot näyttävä `ls`. Itse Linux-ytimen tarvitsemat välttämättömät ohjelmat taas sijaitsevat kansiossa /sbin. NixOS on pyritty rakentamaan siten, että näitä sääntöjä ja tapoja ei ole, vaan kaikki ohjelmat sijaitsevat kansiossa `/nix/store`. Tämä muutos jouduttiin tekemään, jotta järjestelmään voisi asentaa samaan aikaan useita versioita ohjelmista. [9, 10]

Yksi NixOS:n tärkeimmistä ydinkomponenteista on tiedosto nimeltään `configuration.nix`. Se on yksittäinen konfiguraatiotiedosto, joka sisältää kaiken tarvittavan tiedon kokonaisen käyttöjärjestelmän asennuksen replikointiin toisella koneella. Tyypillisesti kuitenkaan järjestelmä ei käytä vain yhtä tiedostoa, vaan pääkonfiguraatiotiedostoon liitetään muista tiedostoista palasia `imports`-avainsanalla. Tiedostojen sisältämät komennot eivät ole mitä tahansa tekstiä, vaan funktionaalista Nix-ohjelmointikieltä, johon palataan tarkemmin luvussa 4.2. [9, 10]

Tällaisesta konfiguraatiotiedostosta löytyy esimerkki ohjelmakoodista 3.1. Tiedoston ensimmäisellä rivillä on Nixin tarvitsemia funktioparametreja, joista ei tarvitse tässä kontekstissa vielä välittää. Seuraavaksi tuodaan kaksi muuta tiedostoa, joista ensimmäinen on automaattisesti tietokoneen laitekonfiguraation luova ohjelma, ja toinen on KDE-työpöytäympäristön ohjelmat sisältävä kokonaisuus. Rivillä 8 käsketään järjestelmän käyt-



Kuva 3.3. *NixOS:n minimaalisen version komponentit. [10]*

tää systemd-palvelun tarjoamaa alkulatausohjelmaa. Tämän jälkeen rivillä 9 käynnistetään SSH-palvelu, jonka avulla voi luoda etäyhteyden järjestelmän ohjaamiseksi. Lopussa on vielä lista, josta löytyy esimerkkiohjelmia, jotka asentuvat automaattisesti osaksi järjestelmää. Näin yksinkertaisella tiedostolla on mahdollista replikoida täysin identtisiä käyttöjärjestelmiä monelle tietokoneelle. [9, 10]

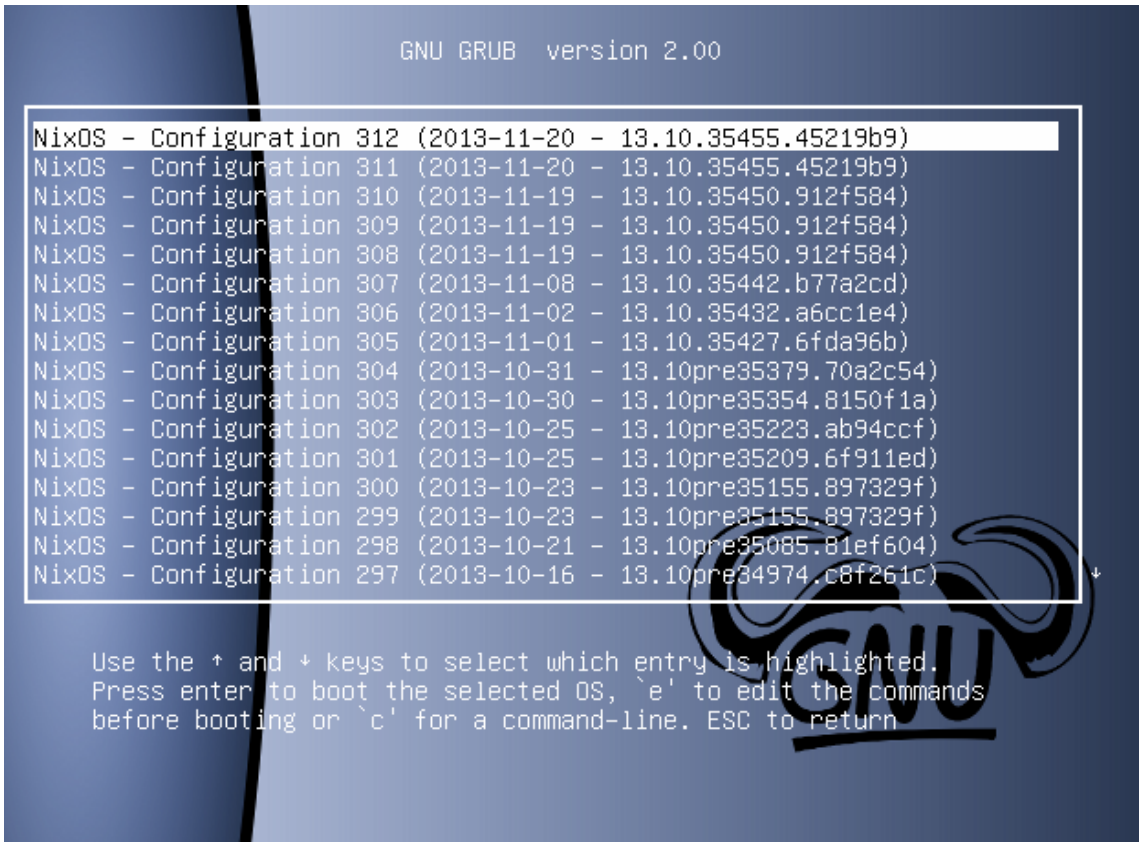
Kuvassa 3.3 näkyy visuaalinen esimerkki eräästä NixOS-asennuksesta. Kuvan pallot ovat eri komponentteja, joita kyseiseen asennukseen kuuluu. Niiden välillä olevat nuolet kuvaavat riippuvuussuhteita. Kuvassa alimpana näkyvä `system.build.toplevel` on koko käyttöjärjestelmän sisältävän paketin nimi, jonka esimerkiksi järjestelmäpäivitykset päivittävät.

3.3 Erityiset ominaisuudet

NixOS:n tilattoman mallin ansiosta se pystyy monenlaisiin operaatioihin, joita mikään perinteinen käyttöjärjestelmä ei pysty tarjoamaan. Ehkä tärkein ominaisuus on niin sanottu deklaratiivinen järjestelmäkonfiguraatiomalli, joka tarkoittaa sitä, että kaikki järjestelmän osat rakennetaan haluttua tilaa kuvaavien konfiguraatiotiedostojen pohjalta ohjelmakoodilla. Kyseinen ohjelmakoodi on puhtaasti funktionaalista, eli suoritus ei koskaan päällekirjoita edellisiä tuloksia. Tämä onkin NixOS:n fundamentaalisin toiminnallisuus, joka erottaa sen muista käyttöjärjestelmistä. [9, 10]

Järjestelmän osia muuttaessa mitään ei päällekirjoiteta, joten vanhat versiot kaikesta jäävät aina talteen ja rinnalle syntyy uudet versiot. Tämä mahdollistaakin koko käyttöjärjestelmän palauttamisen vanhaan tilaan poikkeuksellisen helposti, mikä on varsin hyödylli-

nen ominaisuus, mikäli jotain menee pieleen. Tämän ansiosta päivitykset ovat myös atomisia, joka tarkoittaa käytännössä sitä, että joko kaikki operaatiot tapahtuvat tai mitään ei tapahdu. Mikäli tietokone esimerkiksi sammuu kesken päivityksen, mitään peruttamatonta ei ole tapahtunut, koska tiedostoja ei päällekirjoiteta. Tällaisen tilanteen jälkeen edelliseen toimivaan konfiguraatioon voi palata helposti uudelleen käynnistyksen jälkeen alkulatausohjelmasta, mistä näkyy esimerkkitilanne kuvassa 3.4, jossa näkyy listana päivämäärineen ja versionumeroineen vanhoja tiloja järjestelmästä. Koska käyttöjärjestelmä rakentuu konfiguraatiotiedostojen pohjalta deterministisesti, myös sen päivittäminen toimii samoin. Käyttöjärjestelmän päivittäminen täysin erilaiseksi on siis aina yhtä luotettavaa ja turvallista kuin koko käyttöjärjestelmän uudelleen asentaminen tyhjästä. [9, 10]



Kuva 3.4. *GRUB-alkulatausohjelma, jossa näkyy NixOS:n konfiguraatioita. [1]*

NixOS:ssä järjestelmän uudelleenrakentaminen tapahtuu komennolla `nixos-rebuild`. Tälle komennolle annetaan erilaisia parametrejä, jotka määräävät miten käyttäjä haluaa rakentaa järjestelmän. Esimerkiksi parametrillä `switch` komento rakentaa järjestelmän uudelleen ja luo uuden konfiguraatioversion. Mikäli käyttäjä on epävarma uudesta konfiguraatiosta, voi sitä halutessaan testata parametrillä `switch`. Tällöin ohjelmat päivittyvät, mutta uutta konfiguraatioversiota ei vielä luoda, eli muutokset häviävät uudelleenkäynnistyksen yhteydessä. Mikäli käyttäjä on vielä epävarmempi ja on esimerkiksi haitallisten ohjelmien uhka, voi käyttää parametriä `build-vm`. Tällöin NixOS rakentaa järjestelmästä erityisen version, jonka voi ajaa QEMU-ohjelmistolla virtuaalikoneessa. Tällainen virtuaalikonejärjestelmä suorittuu täysin muulta järjestelmältä eristetyssä hiekkalaatikkotilassa, minkä ansiosta uuden version kokeileminen on täysin turvallista. Tätä ominaisuutta voi to-

ki käyttää muuhunkin kuin testaamiseen, koska sillä voi todella helposti ja nopeasti luoda hyväksi todetusta järjestelmästä kopioita virtuaalikoneiksi. Jos käyttäjä ei ole varma, mitä edellä mainitut komennot tekevät, niille voi vielä halutessaan antaa `-dry-run` -parametrin, jolloin ne suorittuvat normaalisti, mutta eivät tee oikeita muutoksia mihinkään. [14, 16]

NixOS tarjoaa myös mahdollisuuden rakentaa väliaikaisia komentotulkki-istuntoja, joita voidaan käyttää esimerkiksi ohjelmistojen testaukseen pikaisesti. Käyttäjä voi luoda mihin tahansa kansioon erityisen `shell.nix` -tiedoston, jossa kuvataan haluttu konfiguraatio järjestelmälle. Komennolla `nix-shell` Nix rakentaa tämän tiedoston pohjalta uuden väliaikaisen järjestelmän ja avaa sitä varten uuden istunnon komentotulkkiin. Tähän komentotulkkiin syötetyt komennot kohdistuvat vain tähän väliaikaiseen versioon käyttöjärjestelmästä, mikä tekee siitä nopean ja turvallisen tavan esimerkiksi suorittaa ohjelmia halutulla alustalla.

Kun käyttöjärjestelmä tarjoaa näin monimutkaisia ominaisuuksia, vastaan tulee väistämättä kaksi vaihtoehtoa: joko tehdään toimintojen käyttämisestä selkeää mutta työlästä, tai sitten tehdään niistä erittäin ilmaisuvoimaisia, mutta toteutukseltaan ja käyttäytymiseltään vaikeammin ymmärrettäviä. NixOS:n kehittäjät valitsivat jälkimmäisen tien, joten lähes tässä luvussa esitellyt toiminnot tarjotaan abstrahoiduilla ja voimakkailla komennoilla. Mikäli näiden toiminnallisuuden haluaa oikeasti ymmärtää, täytyy käytännössä lukea niiden lähdekoodia. Dokumentaatiostakin löytyy toki tietoa komennoista, mutta niiden oikeaa toimintaperiaatetta ei siellä välttämättä kerrota, vaan ainoastaan niiden haluttu lopputulos. Tällaisessa lähestymistavassa on myös omat etunsa, kuten komentojen suurempi käyttönopeus, mutta vikatilanteen sattuessa abstrahointi vaikeuttaa vian selvittämistä huomattavasti. NixOS ei siis yritäkään noudattaa Unix- ja Linux-yhteisöissä yleistä ja arvostettua *Keep it simple, Stupid!* -ajatusmallia, joka pyrkii mahdollisimman yksinkertaisiin toteutuksiin.

4 NIX-PAKETTIENHALLINTAOHJELMISTO

NixOS:n pakettienhallintaohjelmisto Nix on itsessään muutaman vuoden vanhempi kuin NixOS. Ensimmäinen sitä koskeva tieteellinen julkaisu julkaistiin 16.01.2004 ja ensimmäinen julkinen versio (0.9) itse ohjelmasta 16.10.2005 [17]. Nix ei ole kehitetty yksinoikeudella NixOS:ää varten, vaan sen voi halutessaan asentaa myös mihin tahansa muuhun Linux-jakeluun ja jopa MacOS:ään. Niissä se ei kuitenkaan pysty tarjoamaan yhtä monipuolisia ominaisuuksia kuin sen ympärille rakennetussa NixOS:ssä. [9, 14]

4.1 Toimintaperiaate

Nix poikkeaa toimintaperiaatteeltaan huomattavasti muista tunnetuista pakettienhallintaohjelmistoista, kuten RPM tai dpkg. Sen asentamat ohjelmistopaketit ovat itsessään funktionaalista ohjelmakoodia, jota Nix-tulkki suorittaa. Näitä ohjelmistopaketteja kutsutaan Nix-lausekkeiksi. Tällainen lähestymistapa valittiin, jotta voitaisiin saavuttaa jokaisella asennuskerralla identtinen tulos. [9, 14]

Nix asentaa kaikki ohjelmistot aina `/nix/store` -kansioon. Koska yksi Nixin tarjoamista ominaisuuksista on mahdollisuus asentaa täysin identtinen ohjelma monta kertaa rinnakkain, täytyi ohjelmien nimeämiskäytäntöä muuttaa. Muut pakettienhallintaohjelmistot asentavat ohjelman esimerkiksi `/usr/bin` -kansioon siten, että sen lopulliseksi poluksi muodostuu vaikkapa GNU-projektin `hello`-ohjelman kohdalla `/usr/bin/hello`. Koska Linux-käyttöjärjestelmissä samassa kansiossa ei ikinä voi olla kahta saman nimistä tiedostoa, Nix luo pakettia asentaessa SHA-256 -algoritmilla 32-merkkisen tiivisteen (hash), jonka se lisää ohjelman nimen eteen. Tiivisteen lisäksi ohjelman nimeen lisätään myös sen versionumero, joka tulee nimen jälkeen. Tällöin esimerkkiohjelma `hello` päätyisi polkuun, joka olisi samankaltainen kuin:

`/nix/store/bwacc7a5c5n3qx37nz5drwcgd2lv89w6-hello-2.1.1`. [9, 14]

Ohjelmalle luotava tiiviste muodostuu kryptograafisen funktion avulla kaikista tekijöistä, jotka vaikuttavat ohjelman rakentamiseen. Näitä tekijöitä ovat esimerkiksi itse asennettavan ohjelman lähdekoodi tai binääritiedoston tarkastussumma, Nix-lauseke joka rakentaa ohjelman, kaikki parametrit joita lausekkeelle annetaan ja kaikki rakennusprosessisa käytettävät ohjelmat, kuten kääntäjä, linkittäjä, kirjastot ja muut järjestelmäohjelmat. Mikäli yhteenkin näistä tulee pieninkään muutos, ohjelmalle luotu tiiviste tulee olemaan eri kuin aiemmin. Tämä onkin koko NixOS:n poikkeuksellisen toimintaperiaatteen perusta,

johon monet muut ominaisuudet nojaavat. [9, 14]

Kryptograafisen tiivisteen avulla voidaan aina olla varmoja, että ohjelma on edelleen samassa toimintakunnossa, kuin se oli heti asennuksen jälkeen. Tämä tarkoittaa siis sitä, että kaikkien ohjelmien rakentamiseen vaaditut riippuvuusohjelmat täytyy aina löytyä käyttöjärjestelmästä sellaisena, kuin ne olivat ohjelman rakennushetkellä. Tästä aiheutuu se, että aina kun jokin riippuvuusohjelmista esimerkiksi päivittyy uudemmaksi, vanha versio jää edelleen muuttumattomana itsenäiseksi osaksi järjestelmää. Tästä aiheutuu väistämättä turhaa tilankäyttöä, koska usein päivitykset muuttavat vain hyvin pientä osaa ohjelmista ja lähes aina ohjelman toiminta pysyy ulospäin samanlaisena. Se on kuitenkin ainoa tapa varmistaa täydellinen toimivuus kaikille ohjelmille. NixOS tarjoaa ongelmaa helpottaakseen roskienkeruukomennon, joka etsii ja poistaa tällaisia riippuvuusohjelmia, joita mikään ohjelma ei enää tarvitse. [9, 14]

Nix eroaa muista pakettienhallintaohjelmistoista myös vastuualueiltaan. Koska sillä ajetaan ohjelmakoodia, sillä pystyy tekemään paljon muutakin kuin vain asentamaan uusia ohjelmia. Nix-tulkki suorittaakin monenlaisia muita järjestelmäkonfigurointiin tarkoitettuja ohjelmia, joita käyttävät muun muassa käyttäjille tarkoitetut komennot, kuten esimerkiksi `nixos-rebuild`. Tämä tekee siitä huomattavasti tärkeämmän osan käyttöjärjestelmän toimintaa, kun mitä perinteiset pakettienhallintaohjelmistot olisivat. [9, 14]

4.2 Nix-lausekkeet

Nix-lausekkeet ovat siis Nix-kielellä kirjoitettuja puhtaasti funktionaalisia ohjelmia, joita suoritetaan Nix-tulkilla. Niillä kuvataan järjestelmäkonfiguraation muuttamiseen tarvittavia operaatioita ja kyseisten operaatioiden vaatimia riippuvuuksia. Nix-lausekkeet ovat myös laiskasti evaluoituja (lazy evaluation), eli ohjelmakoodi käyttää vain ja ainoastaan sellaiset resurssit mitä se suorituksessaan tarvitsee. Tämä tarkoittaa sitä, että esimerkiksi Nix-lausekkeen sisältäessä ylimääräisiä riippuvuusohjelmia ne eivät asennu turhaan.

Ohjelma 4.1 on eräs esimerkki Nix-lausekkeesta. Kyseisen lausekkeen tavoitteena on asentaa GNU-projektin `hello`-ohjelma. Ohjelman ensimmäisellä rivillä olevat aaltosulkeet määrittelevät ohjelman ainoan funktion parametrit. Nämä parametrit määrittelevät kaikki ohjelman rakentamiseen tarvittavat riippuvuudet. Tämän ohjelman vaatimuksiin kuuluu ensinnäkin `stdenv`, joka on kokoelma standardiohjelmia, joita tarvitaan ohjelmien kääntämiseen. Tällaisia ovat esimerkiksi C-kääntäjä ja Bash-komentotulkki. Toinen ohjelma on `fetchurl`, jota Nix käyttää internetistä tiedostojen lataamiseen. Kolmantena on `perl`-tulkki, jota tarvitaan asennettavan `hello`-ohjelman ajamiseen. Seuraavaksi kutsutaan `stdenv.mkDerivation`-funktiota, joka on Nix-lausekkeiden kirjoittajia varten tehty apufunktio, joka abstrahoi monia toistuvia operaatioita. Apufunktion sisällä määritellään ensinnäkin ohjelman nimi, jonka yhteydessä on myös sen versionumero. Tämän jälkeen määritellään asennusskripti, jonka `hello`-ohjelman kehittäjät ovat tehneet ohjelman asennusta varten tavallisissa Linux-jakeluissa. Seuraavaksi määritellään itse ohjelman

```
1  {stdenv, fetchurl, perl}:
2
3  stdenv.mkDerivation {
4      name = "hello-2.1.1";
5      builder = ./builder.sh;
6      src = fetchurl {
7          url = http://ftp.gnu.org/pub/gnu/hello/hello-2.1.1.tar.gz;
8          md5 = "70c9ccf9fac07f762c24f2df2290784d";
9      };
10     inherit perl;
11 }
```

Ohjelma 4.1. *Esimerkki Nix-lausekkeesta. [9]*

sisältö, joka haetaan tässä tapauksessa `fetchurl` avulla internetin kautta. Tiedoston ladatakseen `fetchurl` tarvitsee url:n, sekä md5-tarkastussumman, jolla määritellään, onko ladattu tiedosto alkuperäinen vai vioittunut. Lopuksi vielä `inherit`-avainsanalla kerrotaan, että parametrinä annettu `perl`-tulkki asetetaan tässä tapauksessa käytössä olevaksi `perl`-tulkiksi. Tässä vaiheessa on tiedossa kaikki, mitä tarvitaan ohjelman rakentamiseen, ja `stdenv.mkDerivation` alkaa suorittaa asennusskriptiä. Kyseinen skripti on yleensä suunniteltu tavallisille Linux-jakeluille, joissa ohjelmat löytyvät perinteisten kansioiden, kuten `/usr/bin` alta. Tämän takia `stdenv.mkDerivation` joutuu ikään kuin “huijaamaan“ asennusskriptiä siten, että korvaa kaikki sen käyttämät polut NixOS:n vastaavilla. Tämä tapahtuu automaattisesti käyttäen `PATH`-järjestelmämuuttujaa, eikä Nix-lausekkeen kirjoittajan tarvitse välittää siitä. Tässä vaiheessa Nix laskee kaiken edellä mainittujen vaiheiden pohjalta tiivisteen, joka liitetään ohjelman nimen alkuosaan. Tämän jälkeen suoritetaan varsinainen asentaminen kansion `/nix/store/`. [9]

4.3 Ohjelmien paketointi ja jakelu

Jokaiselle NixOS:lle asennettavalle ohjelmalle täytyy olla olemassa samankaltainen Nix-lauseke, kuin edellisessä luvussa kuvaillulla `hello`-ohjelmalla. Olisi varsin työlästä joutua kirjoittamaan niitä jokaiselle ohjelmalle, jota kehittäjät eivät suoraan julkaise NixOS:lle. Tämän takia on perustettu yhteisön kehitettäväksi Nixpkgs-kokoelma, joka on GitHubista löytyvä avoin projekti, jonne yhteisö voi lisätä Nix-lausekkeita ohjelmista. Työn kirjoitushetkellä sieltä löytyi hieman yli 50 000 eri ohjelmaa, joiden Nix-lausekkeiden tekemiseen noin 2000 eri kehittäjää ovat käyttäneet noin 176 000 git commitia. [15]

Github ei kuitenkaan ole ainoa paikka, josta Nix-lausekkeet löytyvät, vaan niille on luotu NixOS:ää varten etärepositorioita, joita ylläpidetään ympäri maailmaa. Näissä sijainneissa olevat palvelimet rakentavat automaattisesti ohjelmien lähdekoodeista binääritiedostoja, jotta käyttäjän ei tarvitsisi tehdä sitä aina ohjelmaa asentaessa. Tällä tavalla toimii myös esimerkiksi Debian-jakelun etärepositoriot. [1]

5 NIXOS:N KÄYTTÖKOHTEET

Tässä luvussa esitellään erilaisia tyypillisiä käyttökohteita, joihin NixOS:ää voidaan käyttää. Tämän lisäksi käydään myös läpi mahdollisia syitä, miksi se ei ole saavuttanut kovin merkittävää suosiota verrattuna muihin tunnettuihin Linux-jakeluihin tai esimerkiksi Dockeriin.

5.1 Yleiskäyttöisenä Linux-jakeluna

Yleiskäyttöiset Linux-jakelut ovat sellaisia jakeluita, joita ei ole tarkoitettu erityiseen käyttötarkoitukseen, vaan käyttäjällä on mahdollisuus ja vastuu itse päättää, millaisia ohjelmia haluaa asentaa ja käyttää. Suosituimpia tällaisia Linux-jakeluita ovat esimerkiksi Ubuntu ja Red Hat Enterprise Linux. NixOS tarjoaa kaikki ominaisuudet, joita yleiskäyttöinen jakelu tarvitsee. Sitä voisi siis käyttää aivan tavallisena käyttöjärjestelmänä työpöytäkäytössä tavallisilla tietokoneilla. NixOS ei ole kuitenkaan onnistunut saamaan merkittävää suosiota tällaisessa käytössä.

NixOS oli työn kirjoitushetkellä Linux-jakeluiden suosiota mittaavalla Distrowatch-sivustolla 70. sijalla. Kyseisen sivuston suosiojärjestys perustuu yksinkertaisesti vain Linux-jakeluiden esittelysivujen klikkausmääriin, joten se ei anna kovin tarkkaa tietoa oikeasta suosiosta. Siitä kuitenkin voi päätellä, ettei NixOS kiinnosta kovin monia uusia jakeluita etsiviä ihmisiä. [6]

NixOS:n vaatimaton suosio johtuu monenlaisista tekijöistä. Se on syntymästään lähtien ollut olemassa vain ratkaisemassa erittäin spesifejä teknisiä ongelmia, joista valtaosa tavallisten käyttöjärjestelmien käyttäjistä eivät välttämättä välitä, eivätkä tarvitse niihin korjausta. NixOS ei ole myöskään kehitetty erityisen käyttäjäystävälliseksi, vaan sen tärkeimmät ominaisuudet ovat pääasiassa komentoriville syötettäviä komentoja. Näitä komentoja käsittelevät dokumentaatiot eivät myöskään ole kaikilta osin tarpeeksi kattavia ja laadukkaita. Vuosien varrella myös muita suosittuja Linux-jakeluita alhaisempi valmiiksi paketoitujen ohjelmistojen määrä on ollut varmasti yksi suosiota alentava tekijä.

NixOS:n suosiota voi spekuloida myös käyttäjäryhmien näkökulmasta. Kaikista tietokoneiden käyttäjistä suurin osa valitsee yksinkertaisesti vain tietokoneeseen valmiiksi asennetun käyttöjärjestelmän, joka on yleensä joko Windows tai MacOS. Pieni osa ihmisistä ei kuitenkaan ole tyytyväisiä näiden ominaisuuksiin, tietoturvaan tai esimerkiksi käyttöjärjestelmän hintaan. Tästä ryhmästä Linux-jakeluihin päätyvät valitsevat muun muas-

sa helppokäyttöisyyden, luotettavuuden ja laajan tuen takia tyypillisesti Ubuntun, siihen pohjautuvan Mintin tai esimerkiksi Fedoran. Tästä jäljelle jäävä ryhmä tavoittelee yleensä laajaa konfiguroitavuutta ja erityisiä ominaisuuksia. Näihin tarkoituksiin suosituimpia jakeluita ovat esimerkiksi Arch Linux, Gentoo ja Slackware. Tämän ryhmän ulkopuolella on enää todella pieni käyttäjäryhmä, joiden tavoitteena on hyvin todennäköisesti todella erityiset käyttötarkoitukset ja täysi kontrolli järjestelmästä. Nämä henkilöt muodostavat todennäköisesti suurimman osan NixOS:n käyttäjistä ohjelmistokehityksen ulkopuolella.

5.2 Työkaluna ohjelmistokehityksessä

NixOS on deterministisyytensä ja helposti replikoitavissa olevien ympäristöjen ansiosta erinomainen väline ohjelmistokehityksen apuna. Se on alusta asti ollut suunniteltu nimenomaan tarjoamaan ohjelmien kääntämiselle ja ajamiselle sellaiset puitteet, ettei käyttöjärjestelmän puolesta tule odottamattomia yllätyksiä. NixOS toimii erinomaisesti ohjelmointialustana monen ohjelmoijan projekteissa, koska jokaiselle voi helposti taata samanlaisen järjestelmän. Halutessaan ohjelmoijat voivat esimerkiksi pitää koko käyttöjärjestelmän konfiguraatiotiedostoa versionhallinnassa muiden projektin tiedostojen mukana, jolloin se pysyy ajan tasalla muun projektin kanssa. Toinen vaihtoehto on antaa ohjelmoijien konfiguroida järjestelmistään sellaisia kuin itse haluavat ja käyttää vain yhteistä `shell.nix` -tiedostoa. Näin vain kehityksessä olevan projektin tarvitsemat riippuvuudet ovat taatusti samat kaikkien tietokoneilla, mutta muuten järjestelmä saa olla omien mieltymysten mukainen. [16]

Kehitysalustan lisäksi NixOS soveltuu myös automaattiseksi testausalustaksi. Isommissa ohjelmistokehitysprojekteissa on hyvin tyypillistä, että kehityksessä olevaa ohjelmistoa testataan jatkuvasti integraatiotesteillä (continuous integration). Näitä testejä ajaa Linux-kehityksen tapauksessa erilaiset yleensä Linux-jakelut, joita suoritetaan jollakin testialustalla. NixOS tarjoaa tähänkin käyttöön erinomaisia työkaluja, joilla voidaan helposti ja nopeasti luoda erilaisia käyttöjärjestelmäkonfiguraatioita esimerkiksi erilaisia testitapauksia varten. Näistä konfiguraatioista voi tehdä helposti myös olemassa olevilla alustoilla toimivia virtuaalikoneita. Tällaisessa käytössä voi halutessaan käyttää myös pelkkää Nix-pakettienhallintaohjelmistoa muilla alustoilla, mikäli ei halua kokonaan vaihtaa järjestelmää. Esimerkiksi Travis-testiympäristöstä löytyy suoraan tuki Nixille [13]. Nixin käyttäminen sekä ohjelmointi-, että testausalustoilla mahdollistaa helposti samanlaiset testitulokset paikallisesti ja testauspalvelimella, minkä ansiosta testaaminen kehityksen ohessa on huomattavasti helpompaa. [16]

NixOS soveltuu myös alustaksi kehitetyn ohjelmiston suorittamiseen esimerkiksi palvelintietokoneessa. Tällaisessa käytössä NixOS on erityisen hyvä, koska voidaan olla varmoja koko järjestelmän toiminnasta ohjelman ympärillä. Ohjelmaa ajava järjestelmä voidaan luoda helposti samasta Nix-lausekkeesta kuin sen kehitykseen käytetty järjestelmä. NixOS tarjoaa myös etähallintatyökaluja, joilla näitä samanlaisia ja mahdollisesti samaa ohjelmaa ajavia järjestelmiä on helppo konfiguroida siten, että kaikille tapahtuu samat

muutokset. Myös päivitykset ovat turvallisempia atomisen toteutuksen takia, ja asioiden mennessä pieleen on helppoa palata takasin aiempaan, toimivaan konfiguraatioon. Aina käyttöjärjestelmää ei haluta kuitenkaan ajaa pelkästään paikallisessa palvelimessa. Pilvipalvelut ovat kasvattaneet merkittävästi suosiotaan viime vuosina ja NixOS tarjoaakin uniikkeja työkaluja, joilla koko järjestelmän voi helposti siirtää pilveen. Näitä järjestelmiä voidaan luoda esimerkiksi NixOps Cloud Deployment -työkalulla, joka luo Nix-lausekkeiden pohjalta kohdejärjestelmälle soveltuvia virtuaalikoneita halutusta NixOS-konfiguraatiosta. NixOps tukee esimerkiksi Amazonin Elastic Compute Cloudia (EC2), Google Cloud Engineä, VirtualBox-alustaa ja monia muita [16, 25]

Edellä mainittuihin käyttötarkoituksiin käytetään nykyään hyvin usein konttiteknologiaan perustuvia järjestelmiä, kuten Dockeria. NixOS tarjoaa esimerkiksi NixOps-työkalulla luoduilla konteilla samankaltaiset ominaisuudet Dockerin kanssa. Docker perustuukin samankaltaiseen ideaan, jossa haluttu järjestelmä luodaan konfiguraatiotiedoston pohjalta. Dockerin käyttämä `Dockerfile`-tiedosto ei kuitenkaan ole `configuration.nix`-tiedoston tapaan funktionaalista ohjelmakoodia, eikä se takaa joka kerta samaa lopputulosta. On hyvin tyypillistä, että `Dockerfile` sisältää komentoja, joilla käsketään asentamaan kokoelma ohjelmia kontin sisään ilman, että määrätään tarkkaa versionumeroa. Tämän takia kontteja rakentaessa myöhemmin samasta tiedostosta voi niihin asentua esimerkiksi uudempia versioita ohjelmista. Dockerin yksi varjopuolia on myös se, että koko kontti on rakennettava uusiksi, mikäli konfiguraatio muuttuu. Tämä voi olla todella hidas operaatio, jos päivitettäviä kontteja on paljon. NixOS sen sijaan päivittää vain sen osan mitä halutaan päivittää. [8, 25]

Vaikka puhtaasti ominaisuuksia katsomalla NixOS ja erityisesti NixOps vaikuttavat paremmalta työkalulta kuin Docker, niiden tarjoamat erityiset ominaisuudet eivät usein ole tarpeen. Dockerin ympärille on lähivuosien aikana kehittynyt valtava yhteisö, jonka ansiosta hyvin moneen käyttötarkoitukseen löytyy jo valmiita kontteja. Yksi niitä keräävä sivusto Docker Hub sisältää työn kirjoitushetkellä yli 100 000 konttia [7]. Dockerilla on myös erittäin hyvä dokumentaatio ja yhteisön sekä kaupallisten yritysten tarjoama laaja tuki. Dockerin käyttämät konfiguraatiotiedostot ovat myös hyvin samankaltaisia esimerkiksi Unix-komentotulkkiskriptien kanssa, joten niiden kirjoittamiseen on Nix-lausekkeita huomattavasti matalampi kynnys.

Dockerin suosion ansiosta se on myös integroitu valtavaan määrään muita ohjelmistoja ja alustoita, joista tunnetuimpia ovat esimerkiksi Amazon AWS, Google Cloud Platform, Kubernetes, Microsoft Azure ja VMware vSphere [3, 4, 5, 20, 27]. Tämä mahdollistaa ohjelmistokehittäjille erittäin helpon käyttöönoton Dockerille, mikä on lähes aina tärkeämpää, kuin poikkeukselliset ominaisuudet, joita NixOS tarjoaa. Laajan integroinnin mukana tulee myös laaja kaupallinen tuki, mikä on varsinkin potentiaalisille yrityskäyttäjille erittäin tärkeä seikka.

6 YHTEENVETO

Tämän työn alkuosiossa alustettiin pohjatiedot Linuxiin pohjautuvien käyttöjärjestelmien, eli Linux-jakeluiden rakenteelle ja toiminnalle. Näitä tietoja käyttäen perehdyttiin hyvin epätavalliseen jakeluun nimeltään NixOS. Kyseinen jakelu luotiin alun perin ratkaisemaan ohjelmistojen jakelun sekä käyttöjärjestelmän järjestelmäkonfiguraation hallinnan perinteisiä ongelmia, kuten haluttujen toimintojen turvallisuus ja deterministisyys. Ratkaistaakseen näitä ongelmia NixOS toteutettiin alusta alkaen muihin tunnettuihin Linux-jakeluihin verrattuna hyvin poikkeuksellisesti.

Merkittävin järjestelmäkonfiguraation hallintaan liittyvä työkalu Linux-jakeluissa on tyypillisesti niiden pakettienhallintaohjelmisto. Sen avulla muun muassa asennetaan, poistetaan ja päivitetään käyttöjärjestelmään haluttuja ohjelmia. Pakettienhallintaohjelmistot muuttavat kolmannen osapuolen ohjelmien lisäksi myös tärkeimpiä ydinkomponentteja, kuten itse Linux-ydintä. Tämän takia NixOS käyttääkin sen tarpeisiin kehitettyä Nix-pakettienhallintaohjelmistoa, jonka toiminnallisuus poikkeaa merkittävästi muiden Linux-jakeluiden käyttämistä pakettienhallintaohjelmistoista. Se on pohjimmiltaan puhtaasti funktionaalista Nix-ohjelmakoodia suorittava tulkki. Nämä ohjelmakoodit kuvaavat tarkasti halutun järjestelmäkonfiguraation muutokset, kuten esimerkiksi uuden ohjelman asentamisen. Koska Nix-ohjelmat, myös nimeltään Nix-lausekkeet, ovat puhtaasti funktionaalista koodia, niiden kutsuminen johtaa aina samaan lopputulokseen. Tämä on tärkeä ominaisuus, sillä se mahdollistaa järjestelmälle deterministisen hallinnan. Tämän lisäksi ohjelmat eivät koskaan voi päällekirjoittaa mitään vanhaa, eli kaikki vanhat tilat järjestelmästä jäävät talteen, jolloin niihin voi myös erittäin helposti palata takaisin virheiden sattuessa.

NixOS ei ole virallisesti suunnattu mihinkään erityiseen käyttöön, vaan se tarjoaa käyttäjälleen kaikki tämän tarvitsemat työkalut, jotta käyttäjä voi itse päättää miten sitä käyttää. NixOS on kuitenkin ominaisuuksiensa puolesta erityisen sopiva ohjelmistokehityksen tarpeisiin, koska vain hyvin harvoin tavalliset käyttäjät oikeasti tarvitsevat täysin determinististä järjestelmää. NixOS sisältää monenlaisia työkaluja, joilla sen konfiguraatioista voidaan helposti kopioita esimerkiksi toisille tietokoneille, virtuaalikoneisiin tai vaikkapa Amazon EC2 -pilvipalveluun. Tällaisia ominaisuuksia tarjoaa myös esimerkiksi Docker, joka on konttiteknologiaan perustuva ohjelmisto, jonka voi asentaa muun muassa mihin tahansa Linux-jakeluun. Sen käyttäminen on ainakin subjektiivisesti helpompaa kuin NixOS:n ja sillä on myös merkittävän suuri käyttäjäkunta ja kaupallinen tuki. Näiden etujen varjopuolena on kuitenkin se, ettei Docker tarjoa yhtä tiukkaa hallittavuutta kuin NixOS.

NixOS on jouduttu ominaisuuksiensa takia toteuttamaan toiminnallisuudeltaan poikkeavasti verrattuna muihin Linux-alustoihin. Siinä on esimerkiksi epästandardi tiedostojärjestelmän hierarkia, mikä saattaa aiheuttaa huomattavaa päänvaivaa ohjelmistokehittäjille. NixOS:n ydintoiminnot myös ovat erittäin abstrahoituja, jonka takia niiden käyttäminen on nopeaa, mutta virhetilanteen sattuessa ongelman syyn selvittäminen voi olla vaikeaa. Niitä käsittelevissä dokumentaatioissa on myös paikoitellen puutteita. Muun muassa näiden seikkojen ja Dockerin suosion takia NixOS ei ole saavuttanut kovin merkittävää suosiota, mutta se ei kuitenkaan NixOS:n toiminnallisuutta haittaa. NixOS tarjoaa erikoisiin ja vaativiin käyttötarkoituksiin ominaisuuksia, joita mikään muu järjestelmä ei tällä hetkellä tarjoa, joten sillä on paikkansa Linux-jakeluiden joukossa. NixOS on pysynyt avoimen lähdekoodinsa ansiosta yli viidentoista vuoden ajan tuhansien kehittäjien työn kohteena, eikä kehitys ole näillä näkymin hiipumassa.

LÄHDELUETTELO

[1] *About NixOS*. 24. elokuuta 2018. URL: https://nixos.org/nixos/about.html (viitattu 29. 03. 2019).

[2] *Alpine Linux Downloads*. URL: http://dl-cdn.alpinelinux.org/alpine/v3.9/releases/x86_64 (viitattu 29. 03. 2019).

[3] *Azure Virtual Machine Extension for Docker*. URL: https://github.com/Azure/azure-docker-extension (viitattu 22. 04. 2019).

[4] *Container Orchestration*. URL: https://www.docker.com/products/orchestration (viitattu 22. 04. 2019).

[5] *Containers on Compute Engine*. URL: https://cloud.google.com/compute/docs/containers/ (viitattu 22. 04. 2019).

[6] *Distrowatch*. URL: https://distrowatch.com/ (viitattu 11. 04. 2019).

[7] *Docker Hub*. URL: https://hub.docker.com/ (viitattu 22. 04. 2019).

[8] *Docker Overview*. URL: https://docs.docker.com/engine/docker-overview/ (viitattu 22. 04. 2019).

[9] E. Dolstra. The purely functional software deployment model. Tohtorinväitöskirja. 2006. URL: https://nixos.org/~eelco/pubs/phd-thesis.pdf.

[10] E. Dolstra, A. Löh ja N. Pierron. *NixOS: A Purely Functional Linux Distribution*. 2010. URL: https://nixos.org/~eelco/pubs/nixos-jfp-final.pdf.

[11] *First dpkg C Version*. URL: https://git.dpkg.org/cgit/dpkg/dpkg.git/plain/main/main.c?id=1b80fb16c22db72457d7a456ffbf1f70a8dfc0a5 (viitattu 11. 04. 2019).

[12] *Mobile & Tablet Operating System Market Share Worldwide*. Helmikuu 2019. URL: http://gs.statcounter.com/os-market-share/mobile-tablet/worldwide/ (viitattu 17. 02. 2019).

[13] *Nix on Travis*. URL: https://nixos.wiki/wiki/Nix_on_Travis (viitattu 22. 04. 2019).

[14] *Nix Package Manager Guide*. Versio 2.21. URL: https://nixos.org/nix/manual/ (viitattu 25. 01. 2019).

[15] *Nix Packages Collection*. URL: https://github.com/NixOS/nixpkgs (viitattu 11. 04. 2019).

[16] *NixOS Manual*. Versio 19.03. URL: https://nixos.org/nixos/manual/ (viitattu 12. 04. 2019).

[17] *NixOS News*. 5. kesäkuuta 2014. URL: https://nixos.org/news.html (viitattu 29. 03. 2019).

[18] *Operating System Family / Linux | TOP 500 Supercomputer Sites*. Marraskuu 2018. URL: https://www.top500.org/statistics/details/osfam/1 (viitattu 17. 02. 2019).

[19] *Package Manager*. URL: https://devopedia.org/package-manager (viitattu 11. 04. 2019).

[20] *Provision AWS EC2 instances*. URL: https://docs.docker.com/machine/examples/aws/ (viitattu 22. 04. 2019).

[21] *Scientific Linux Downloads*. URL: http://ftp.scientificlinux.org/linux/scientific/7x/x86_64/iso/ (viitattu 29. 03. 2019).

[22] *Screenshot of Synaptic 0.62*. 17. helmikuuta 2009. URL: https://commons.wikimedia.org/wiki/File:Synaptic_screenshot.png (viitattu 22. 04. 2019).

[23] *Standard UNIX Filesystem Hierarchy*. 23. kesäkuuta 2016. URL: https://commons.wikimedia.org/wiki/File:Standard-unix-filesystem-hierarchy.svg (viitattu 22. 04. 2019).

[24] *The GNU Operating System*. URL: https://www.gnu.org/gnu/gnu.html (viitattu 26. 04. 2019).

[25] *The NixOS Cloud Deployment Tool*. URL: https://nixos.org/nixops/ (viitattu 22. 04. 2019).

[26] *Usage of operating systems for websites*. 29. maaliskuuta 2019. URL: https://w3techs.com/technologies/overview/operating_system/all (viitattu 29. 03. 2019).

[27] *vSphere Integrated Containers*. URL: https://www.vmware.com/products/vsphere/integrated-containers.html (viitattu 22. 04. 2019).

[28] *What Are the Components of a Linux Distribution?* 17. heinäkuuta 2016. URL: https://fosspost.org/education/what-are-the-components-of-a-linux-distribution (viitattu 19. 02. 2019).