Opinnäytetyö (AMK)

Tieto- ja viestintätekniikka

Sulautetut ohjelmistot

2017

Tomohito Kato

OHJELMISTOKONTTIEN CI/CD-PALVELUIDEN MÄÄRITTELY JA ASENNUS KUBERNETES-OHJELMISTO-KONTTIKLUSTERIIN

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka | Sulautetut ohjelmistot

2017 | 41

Ohjaaja | Jari-Pekka Paalassalo

Tomohito Kato

OHJELMISTOKONTTIEN CI/CD-PALVELUIDEN MÄÄRITTELY JA ASENNUS KUBERNETES-OHJELMISTOKONTTIKLUSTERIIN

Tämä opinnäytetyö on tehty Woima Hosting Oy:lle osana Kubernetes-ohjelmistokonttiklusterin hallintajärjestelmän käyttöönoton kartoitusta. Opinnäytetyön tavoitteena on pystyttää kokeellinen automaatioon pohjautuva mikropalveluiden tuottamiseen, testaamiseen ja jakeluun tarvittava palvelukokonaisuus ohjelmistokonttiklusterissa ajettavista avoimen lähdekoodin mikropalveluista.

Opinnäytetyön teorialuvussa käsitellään Kubernetes-järjestelmää ja järjestelmän taustalla toimivaa ohjelmistokonttiteknologiaa. Kubernetes-manifesteilla luodaan määritelmät palvelukokonaisuudelle ja se käynnistetään ohjelmistokonttiklusteriin. Palvelukokonaisuus määritellään toimimaan siten, että ohjelmistokonttien yksikkötestaus ja koonti tapahtuu skaalautuvassa lisäpalvelukokonaisuudessa. Lisäpalvelukokonaisuuden avulla käynnistetään ohjelmistokontin sisälle oma ohjelmistokontteja ajava konttimoottoritaustaprosessi, jossa uuden ohjelmistokontin koonti ja yksikkötestaus suoritetaan eristettynä isäntäkoneen ohjelmistokontteja ajavasta taustaprosessista. Lopuksi palvelukokonaisuuteen luodaan esimerkkiprojekti, joka kootaan ja yksikkötestataan järjestelmän avulla.

Viimeisessä luvussa arvioidaan Kuberneteksen ja määritellyn kokeellisen palvelukokonaisuuden käyttökelpoisuutta tuotantokäyttöön. Järjestelmä todettiin toimivaksia ja automaattisia testauksia onnistuttiin tekemään. Järjestelmän tuotantokäyttöön saattaminen vaatisi kuitenkin vielä lisää työtä.

ASIASANAT:

Kubernetes, Docker, Linux, Ubuntu, ohjelmistokontti, käyttöjärjestelmätason virtualisointi, mikropalvelu, testausautomaatio

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology | Embedded Software

2017 | 41

Instructor | Jari-Pekka Paalassalo

Tomohito Kato

SETUP AND INSTALLATION OF CI/CD SERVICES IN THE KUBERNETES CONTAINER CLUSTER

This thesis was commissioned by Woima Hosting Oy as a part of the evaluation for Kubernetes infrastructure deployment. This thesis studies software containers and services infrastructure needed for developing, unit testing and delivering containers. The goal of this thesis was to deploy an experimental automation based open-source container development services to the Kubernetes container cluster.

The theoretical part of the thesis studies technologies behind software containers and deploying containers in the Kubernetes container cluster.

The development services were defined and deployed to the Kubernetes cluster. The container building and unit testing were implemented in an isolated on-demand containerized container engine process which was separated from the nodes container engine.

With the development services, an example project was deployed with automated building and unit testing with predefined unit test and build commands. The successfully tested and built container was then automatically uploaded to a container delivery service.

Finally, the thesis evaluates the production readiness of the experimental development services infrastructure and the Kubernetes. In conclusion, the development services worked as expected and automated builds were successful. However the system would still need further configuration to be production ready.

KEYWORDS:

Kubernetes, Docker, Linux, Ubuntu, container, operating-system-level virtualization, microservice, test automation

SISÄLTÖ

LIITTEET

KUVAT

KÄYTETYT LYHENTEET

AUFS	Advanced Multi-layered unification filesystem on kerrostettu tiedostojärjestelmä, jolla yhdistetään kahden muun tai jo kerrostetun tiedostojärjestelmän hakemistopolut päällekkäin.
CD	Continuous Delivery on ohjelmistokehityksen työtapa, jossa pyritään julkaisemaan ohjelmaa nopealla tahdilla lyhyissä sykleissä.
CI	Continuous Integration on työtapa, jossa ohjelmistoprojektin ohjelmoijat yhdistävät koodinsa versionhallintajäjrjestelmässä. Työtapaan kuuluu myös koodin yksikkötestaaminen laadun varmistamiseksi.
GCE	Google Compute Engine on Google Cloud Platformin IaaS -palvelu, jossa käyttäjä voi vuokrata virtuaalikoneita käyttöönsä.
HA	High Availability on käytäntö, jolla taataan järjestelmän esteetön saatavuus vikatilanteista huolimatta.
IPC	Inter-process communication on menetelmä, jonka avulla prosessi voi kommunikoida toisen prosessin kanssa erilaisin menetelmin.
JSON	JavaScript Object Notation on avoin standardi, jota käytetään kuvaamaan avain-arvopari-tyyppisiä tietorakenteita.
KVM	Kernel-based Virtual Machine on konevirtualisointijärjestelmä, jossa Linux-ydin toimii virtuaalikoneiden hypervisorina.
NFS	Network File System on hajautettu tiedostojärjestelmä, jolla hakemistopolkuja jaetaan palvelimelta asiakasohjelmalle verkon ylitse.

1 JOHDANTO

Viime vuosina Linux-ytimessä tapahtuneiden edistysaskelten myötä on tullut mahdolliseksi eristää yksittäisiä prosesseja toisistaan. Näistä edistysaskelista on syntynyt ohjelmistokonttiteknologiat. Ohjelmistokontit mahdollistavat uudenlaisen lähestymistavan suurten klusteroitujen järjestelmien rakentamiseen.

Ohjelmistokonttiklusterin hallintajärjestelmillä pystytään toteuttamaan resurssioptimoituja skaalattavia järjestelmiä, jotka tuovat yrityksille säästöjä infrastruktuurikustannuksissa. Esimerkiksi lontoolainen pankki Monzo on pienentänyt infrastruktuurikustannuksiaan noin kahdella kolmasosalla siirtämällä palveluitaan ohjelmistokonttipohjaiseen järjestelmään [1]. Monet suuret yritykset ovatkin yhdessä Linux Foundation kanssa perustaneet Open Container Initiative -yhdistyksen, joka pyrkii luomaan lähitulevaisuudessa ohjelmistokonttien avoimen standardin. Näihin yrityksiin kuuluu mm. Google, Docker, Microsoft, IBM, Cisco, Twitter, Facebook, vmware ja Goldman Sachs. [2]

Opinnäytetyön toimeksiantaja Woima Hosting Oy halusi kartoittaa mahdollisuuksia Kubernetes-järjestelmän käyttöönottoon. Järjestelmään siirtymisen tueksi tarvittiin palvelukokonaisuus, jonka avulla itse tuotettuja ohjelmistokontteja pystytään testaamaan ja kokoamaan automaation avulla.

Tässä opinnäytetyössä tarkastellaan ohjelmistokonttiteknologiaa sekä Kubernetes-ohjelmistokonttiklusterin hallintajärjestelmää. Kubernetes-järjestelmään määritellään ja asennetaan kokeellinen ohjelmistokonttien yksikkötestaamiseen soveltuva palvelukokonaisuus, joka sisältää versionhallintapalvelun, automaatiopalvelun ja ohjelmistokonttien levykuvien säilytyspalvelun.

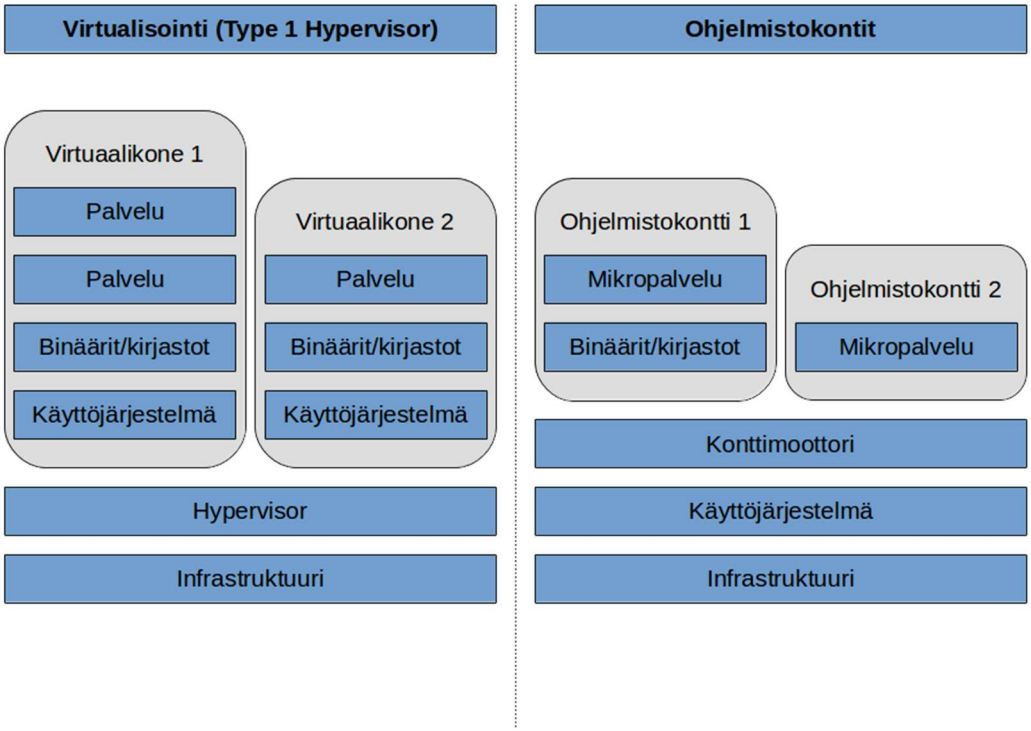
Hyvien käytäntöjen mukaisesti testausympäristö on syytä pitää erillään testauspalveluita ajavasta järjestelmästä, jotta testattava ohjelma ei pääse ongelmatilanteessa vaikuttamaan muiden taustapalveluiden toimintaan. Ohjelmistokonteilla toimivassa järjestelmässä tämän saavuttamiseksi testattava ohjelma ajetaan erityisessä ohjelmistokontissa, joka pitää sisällään oman eristetyn ohjelmistokontteja ajavan taustaprosessin.

2 TEORIA

2.1 Ohjelmistokonttiteknologiat

Ohjelmistokonttien avulla palvelusta tai ohjelmasta tulee helposti siirrettävä ja itsenäinen kokonaisuus. Aikaisemmin suurten järjestelmien hallinnointi on ollut työlästä, sillä ohjelmat ja palvelut ovat olleet sidoksissa asennetun käyttöjärjestelmän sisältämiin ohjelmiin, kirjastoihin ja niiden eri versioihin. Ohjelmistokonttien avulla ohjelman tai palvelun kehittäjät voivat itse julkaista kokonaisuuden, joka sisältää kaiken sen ajamiseen tarvittavat kirjastot ja aliohjelmat. [3]

Ohjelmistokonttiteknologiat ovat käyttöjärjestelmätason virtualisointia, joka poikkeaa perinteisestä virtualisoinnista siten, että ohjelmistokontin prosessit ajetaan suoraan isäntäkoneen Linux-ytimessä eristettyinä prosesseina. [4]. Kun taas perinteisessä virtualisoinnissa tietokoneen laitteistoa emuloidaan isäntäkoneella hypervisorin avulla. Hypervisorin emuloimassa virtuaalikoneessa ajetaan oma erillinen käyttöjärjestelmä, jonka Linux-ytimessä prosessit ajetaan [5]. (Kuva 1.)



Kuva 1. Perinteinen virtualisointi vs. käyttöjärjestelmätason virtualisointi. Mukaillen [6].

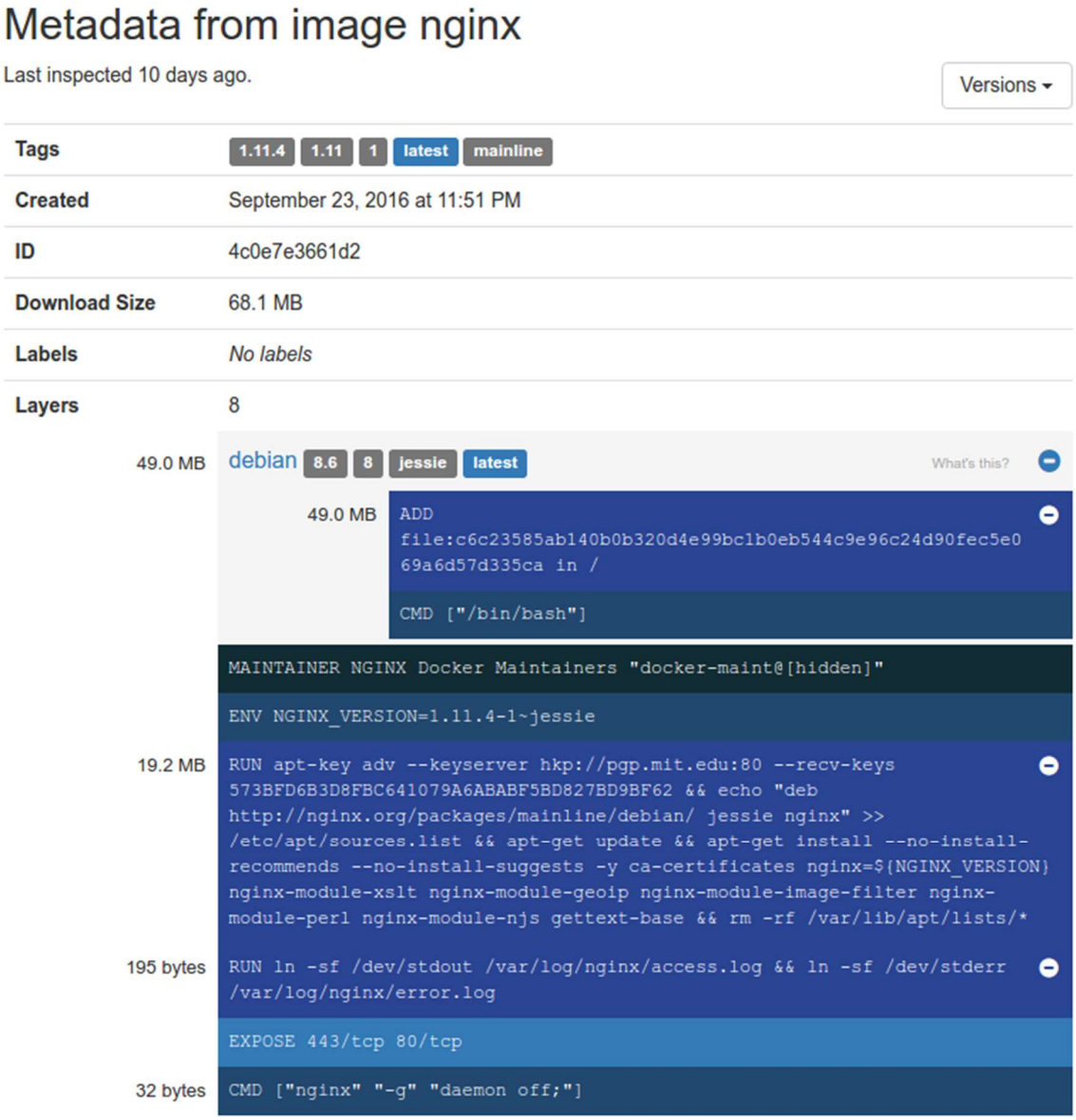
IBM:n tutkijaryhmän mukaan ohjelmistokonttiteknologiat ovat kevyempiä kuin KVM-virtualisointi. Mittausten mukaan ohjelmistokontit pääsevät suorituskyvyltään samoihin tai parempiin tuloksiin kuin perinteinen virtualisointi. [7]

Ohjelmistokontti on levykuva, joka sisältää pääasiallisen ohjelman ja tarvittaessa sen tarvitsemat kirjastot sekä minimalistisen käyttöjärjestelmän. Koska ohjelmistokontti sisältää kaiken pääasiallisen ohjelman ajamiseen tarvittavan, niin sitä voidaan siirtää helposti eri alustalta toiselle. Ohjelmistokontin pääasiallinen ohjelma on yleensä mikropalvelu kuten web-palvelinohjelmisto tai tietokanta. [8]

Ohjelmistokonttien levykuvia jaetaan rekisterien kautta, johon ohjelmistokontin kehittäjä siirtää sen verkon ylitse [9]. Kaupallisia ohjelmistokonttirekistereitä ovat mm. GCE:n Container Registry, Docker Hub ja Quay. Yksityistä ohjelmistokonttirekisteriä voi myös ylläpitää itse avoimen lähdekoodin mikropalveluilla.

Ohjelmistokontin levykuva koostuu pinoon liitetyistä tiedostojärjestelmistä, joista jokainen sisältää levykuvan luontihetkellä määritellyn, ajettavan komennon tekemät muutokset tiedostojärjestelmässä. Alimmassa kerroksessa on tyypillisesti minimalistinen käyttöjärjestelmä ja sen päällä ovat käyttöjärjestelmän paketinhallinnan tai muun jakelumekanismin avulla asennetut kirjastot ja mikropalvelu. Nämä kerrokset ovat vain luku -tilassa. Ohjelmistokontin ajon aikana kerrosten päälle lisätään kirjoitettava tiedostojärjestelmä, ohjelmistokonttikerros, johon ajon aikana tapahtuneet muutokset tallentuvat. Näin alkuperäinen levykuva pysyy aina muuttumattomana. [10]

Aikaisempaa levykuvaa voi käyttää uuden ohjelmistokontin luomiseen, jolloin levykuvan päälle lisätään uusia kerroksia. Uusi levykuva ei sisällä aikaisemman levykuvan kerroksia, vaan aikaisempi levykuva jakaa kerroksensa uuden levykuvan käyttöön. Näin samaan alempaan levykuvaan pohjautuvat ohjelmistokontit käyttävät useissa tapauksissa vähemmän levytilaa kuin virtuaalikoneen levykuvat. [7]. (Kuva 2.)



Kuva 2. Debianiin pohjautuva nginx-ohjelmistokontti tarkasteltuna microbadger.com -palvelussa.

Ohjelmistokontin ajon aikana sen sisältämät prosessit toimivat jaetussa hiekkalaatikkoympäristössä. Hiekkalaatikko on toteutettu Linux-ytimen nimiavaruuksilla (kernel namespaces). [3]

Nimiavaruuksilla ohjelmistokontin prosessit saavat eristetyn nimialuesidonnaisen kloonin globaalista systeemiresurssista. Vain nimialueeseen kuuluvat prosessit pystyvät näkemään omassa globaalissa systeemiresurssissaan tapahtuneet muutokset, kun taas muille prosesseille nämä ovat näkymättömiä. Linux-ytimen nimialueita ovat kontrolliryhmät (cgroups), prosessien väliset kommunikointimenetelmät (IPC), mount, prosessitunnukset, käyttäjäryhmät ja UNIX-ajanjakojärjestelmä (UTS). [11]

Linux-ytimen kontrolliryhmien avulla nimialueen prosesseille voidaan asettaa rajoitus muistiin, prosessoriin tai levyoperaatiohin [12]. Eristyksen ansiosta prosessit eivät näe ja eivätkä pysty vaikuttamaan muihin samassa Linux-ytimessä ajettaviin prosesseihin [3] [13].

Docker on yleisin käytössä oleva avoimen lähdekoodin konttimoottorisovellus (container runtime) [6]. Docker tukee OverlayFS, AUFS, Btrfs, Device Mapper, VFS ja ZFE -varastoajureita, joilla ohjelmistokontin kerroksia hallitaan [14]. rkt on CoreOS:n avoimen lähdekoodin konttimoottori, joka pyrkii Dockeria raskaampaan ohjelmistokontin eristämiseen laitteistoeristettyjen virtuaalikoneiden avulla [15]. Kubernetes voi ajaa ohjelmistokontteja molempien (container runtime) ohjelmien avulla.

2.2 Kubernetes

Palvelinautomaatiolla saavutetaan vikasietoisia järjestelmiä, jotka vaativat vikatilanteissa vähemmän ylläpitäjän väliintuloa. Uusilla automaatioon pohjautuvilla ohjelmistokonttiklusterinhallintajärjestelmillä pystytään ylläpitämään helposti korkean saatavuuden (HA) palveluita.

Kubernetes on alun perin Googlen avoimen lähdekoodin järjestelmä, jossa ajetaan ohjelmistokontteja useamman fyysisen tai virtuaalisen tietokoneen muodostamassa klusterissa. Se automatisoi ohjelmistokonttien käyttöönoton, skaalaamisen ja hallinnoinnin. Kuberneteksen pääperiaatteita ovat siirrettävyys, laajennettavuus sekä vikasietoisuus (self-healing). [16]

Google aloitti Kubernetes projektin vuonna 2014 [16]. Kuberneteksen ensimmäinen vakaa versio julkaistiin heinäkuussa 2015, jonka jälkeen se lahjoitettiin vasta perustetulle Cloud Native Computing Foundationille (CNCF), joka toimii yhteistyössä Linux Foundationin kanssa [17]. Osa Kuberneteksen toimintamalleista perustuu Googlen pitkään kokemukseen ajaa ja hallinnoida ohjelmistokontteja tuotannossa. Osa projektin kehittäjistä

on aikaisemmin toiminut Googlen oman klusterinhallintajärjestelmä Borgin kehittäjinä. Kubernetes on saanut paljon vaikutteita Borgista [18]. Kuberneteksen GitHub-version-hallintapalvelussa oli heinäkuussa 2016 yli 800 ohjelmistoprojektiin osallistunutta ohjelmoijaa (contributor) ja ohjelman koodiin on tehty yli 30 000 muutosta (commit) [18]. Myös Microsoft, Red Hat, IBM, Docker, Mesosphere, CoreOS ja SaltStack ovat osallistuneet Kubernetes-projektiin [20].

Siirrettävyyden ansiosta Kubernetes voi toimia yksityisessä, julkisessa, hybridi- tai multipilvessä [16], joita ovat Amazon EC2, Google Cloud Platform tai Microsoft Azure. Kubernetes voi asentaa fyysiseen tai virtuaaliseen koneeseen [21]. PaaS-järjestelmät kuten Red Hat Openshift, Deis ja Gondor toimivat Kubernetesellä [16].

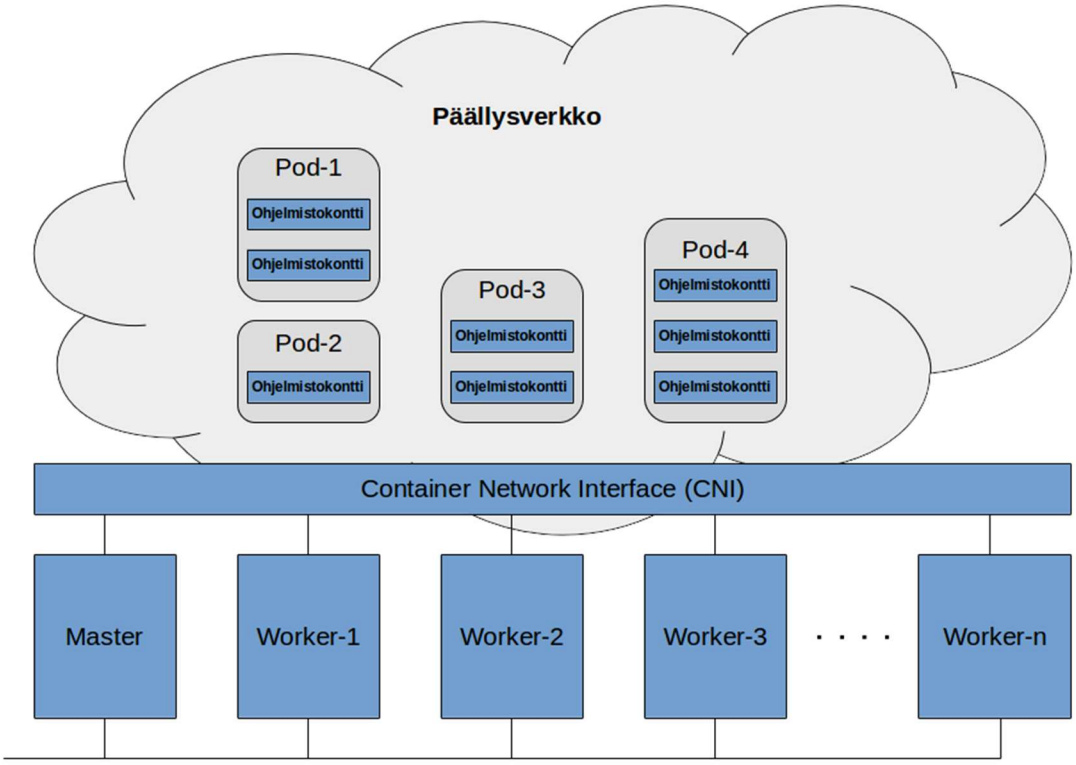
Kubernetes on suunniteltu vikasietoiseksi, ja se reagoi automaattisesti vikatilanteisiin ja pyrkii korjaamaan ne. Jos Kuberneteksen noodi vioittuu, niin se käynnistää vioittuneen noodin ohjelmistokontit toiseen toimivaan noodiin. Ohjelmistokontin vioittuessa Kubernetes yrittää käynnistää sen uudelleen myöhemmin. [22]

2.3 Kubernetes-objektit

Kubernetes käyttää sisäisten objektien lajitteluun etikettejä (labels) ja valitsimia (selectors). Etiketit ovat mielivaltaisia avain-arvo pareja, joilla voidaan hakea objekteja tai objektiryhmää järjestelmästä. [23]

Pod on Kuberneteksen pääasiallinen ajettava yksikkö. Se on yhdestä tai useammasta ohjelmistokontista koostuva looginen kokonaisuus, jota voidaan verrata yhden fyysisen tai virtuaalisen koneen toteuttamaan palvelukokonaisuuteen. Pod voi olla esimerkiksi palvelukokonaisuus, joka sisältää Web-palvelin- ja tietokantaohjelmistokontit. [24]

Kaikki yhden podin sisältämät ohjelmistokontit jakavat keskenään saman IP-osoitteen klusterin sisäisestä päällysverkosta ja yhteisen porttiavaruuden. (Kuva 3.) Ohjelmistokontit voivat myös kommunikoida podin sisällä keskenään muilla yleisillä prosessienvälisillä kommunikointi menetelmillä (IPC), kuten SystemV-semaforilla tai POSIX jaetulla muistilla. Ohjelmistokontit voivat myös jakaa podin sisällä keskenään hakemistopolkuja. [24]



Kuva 3. Klusterin sisäinen päällysverkko.

Podeja ei ole tarkoitettu pysyviksi objekteiksi, vaan Kubernetes-järjestelmä luo ja tuhoaa niitä halutun tilan mukaan. Podien tilaa hallinnoivat erilaiset ohjaimet (controller), joihin määritellään järjestelmässä ajettavien podien haluttu lukumäärä ja asetukset. Yleisesti ohjaimet pitävät huolen siitä, että haluttu määrä kopioita podista on ajossa järjestelmässä. Jos pod vioittuu tai poistetaan, ohjain luo uuden kopion podista. Esimerkiksi klusteroidun palvelimen vioittuessa ohjain käynnistää uuden kopion siitä toiselle palvelimelle. Erilaisia ohjaimia ovat ReplicationController, ReplicaSet, Deployment, Job sekä

DaemonSet. Toisin kuin podit, ohjaimet ovat pitkäikäisiä ja häviävät järjestelmästä vasta poistettaessa. [25]

ReplicationController ja ReplicaSet ovat ohjaimia, jotka pitävät huolta ajossa olevien podien tilasta ja määrästä. Erona näissä kahdessa on, että ReplicaSet käsittelee podeja sarjatyyppisellä etiketti lajittelulla, kun taas ReplicationController yhtäläisyystyyppisellä lajittelulla. [26]

Deployment tyyppiset ohjaimet ovat tarkoitettu hallinnoimaan päivittyviä kontteja sisältäviä podeja. Deploymentilla voidaan päivittää ja palauttaa podeja uudempaa tai aikaisempaan versioon. Deployment hallinnoi podeja ReplicaSet-objektin kautta. [27]

DaemonSet tyyppiset ohjaimet ovat tarkoitetut hallitsemaan alemman tason palveluita tarjoavia podeja. Tällaiset podit käynnistyvät järjestelmään muita podeja aikaisemmin ja ovat usein ajossa klusterin jokaisessa koneessa. Etikettien avulla DaemonSet voidaan myös tehdä noodisidonnaisiksi. [28]

Services-objekteilla toteutetaan Kuberneteksen sisäisen kuormantasaus. Services objekti pitää huolen, että kaikki siihen määritellyt podit löytyvät yhteisen virtuaalisen IP-osoitteen takaa. Kuormantasaus on tärkeää, koska podien IP-osoitteet vaihtelevat klusterissa tapahtuvien tilanmuutosten takia. [29]

Ingress-objekteilla Kubernetes pystyy asettamaan ulkoisille kuormantasaajille Services-objekteja, joita halutaan tavoittaa ulkoverkosta. Ingress-ohjain (Ingress controller) on vastuussa Ingress-objektin toteutumisesta. GCE:ssä ajettavaan Kubernetekseen GCE luo automaattisesti Ingress Controllerin. Muissa ympäristöissä Ingress-ohjaimena voi käyttää Traefik- tai nginx-ohjaimia. [30]

Secret-objekti pitää sisällään salassa pidettäviä tietoja kuten salasanoja, sertifikaatteja tai muuta tietoa, jota ei ole hyvä sisällyttää itse ohjelmistokontin levykuvaan. Näitä objekteja voidaan liittää podin sisältämiin ohjelmistokontteihin ympäristömuuttujina tai tiedostoina. Secret-objektin tiedot tallentuvat noodin keskusmuistiin luotuun tiedostojärjestelmään (tmpfs), joka on olemassa vain sen ajan, kun objektia käyttävä pod on ajossa noodissa. Näin salassa pidettävät tiedot eivät tallennu koskaan noodin kovalevylle. [30]

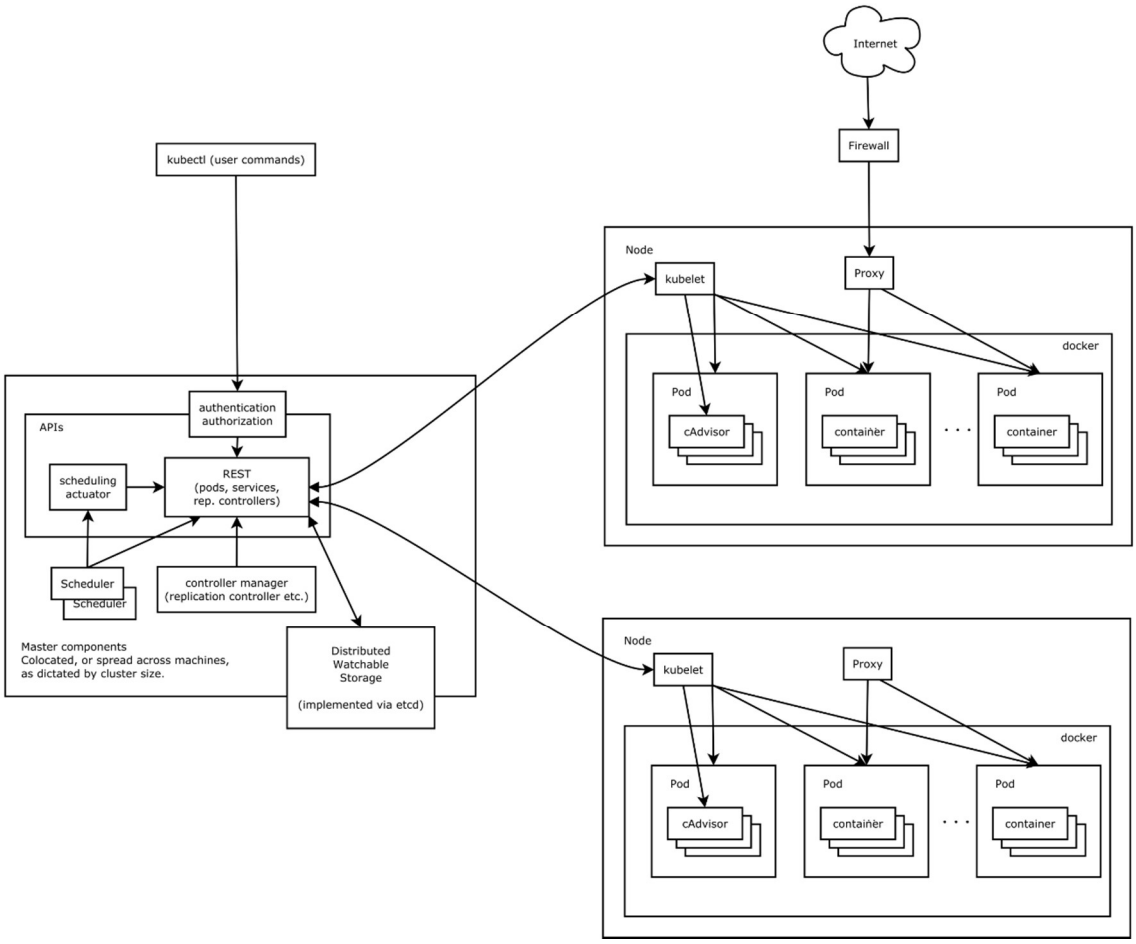
ConfigMap-objektit muistuttavat Secret-objektia. Niillä määritellään tiedostoja, avain-arvopareja, ympäristömuuttujia tai komentoriviparametreja podin sisältämiin ohjelmistokontteihin. Tällä pyritään pitämään ohjelmistokontit helposti siirrettävinä ilman etukäteen tarkkaan määriteltyjä asetustiedostoja. [32]

PersistentVolume- ja PersistenVolumeClaim-objekteilla Kubernetes tarjoaa ohjelmistokonteille pysyvää tallennustilaa, joka voidaan liittää ohjelmistokontin hakemistopolkuun. Tämän avulla ohjelmistokontti, joka käynnistyy uudelleen toisessa noodissa, saa käyttöönsä aikaisemman instanssinsa tallentamat tiedostot. PersistentVolume-objekti määrittelee tallennuspalvelun, johon tiedostot tallentuvat. Kubernetes voi käyttää taustapalveluna pilvitarjoajien kuten Amazon AWS:n, Microsoft Azuren, GCE:n tai OpenStackin tallennusjärjestelmiä. Paikallisesti asennettu Kubernetes voi käyttää verkon kautta jaettuja tiedostojärjestelmiä kuten NFS tai Glusterfs. PersistenVolumeClaim-objekteilla varataan tallennustila podien käyttöön. [33]

Namespaces-nimiavaruuksilla Kubernetes jakaa fyysisen klusterin virtuaalisiksi klustereiksi. Kuberneteksen omat komponentit ovat oletusarvoisesti kube-system-nimiavaruudessa. [33]. Nimiavaruuden objekteille voidaan asettaa rajoituksia ResourceQuota-objekteilla. Näillä voidaan rajoittaa nimialueen käyttämiä prosessoriytimiä, keskusmuistia tai nimialueen Kubernetes-objektien määrää. [35]

2.4 Kuberneteksen arkkitehtuuri

Kubernetes järjestelmä koostuu tietokoneklusterista, jossa on yksi tai useampi isäntä noodi (master node) sekä useampi työnoodi (worker node). [36]. (Kuva 4.)



Kuva 4. Kuberneteksen arkkitehtuuri [36].

Isäntänoodissa sijaitsee Kuberneteksen ohjaustaso (control plane). Ohjaustason komponentit ovat Kubernetes-rajapintapalvelin (kube-apiserver), kube-scheduler, kube-conroller-manager ja etcd-avain-arvo-pari-tietokanta. [36]

Scheduler-komponentti pitää huolen klusterin resurssien jaosta ja päättää, missä työnoodissa podeja ajetaan [37]. Kubernetes Controller Manager -komponentti hallinnoi klusterin tilaa Controller-objekteilla määriteltyyn suuntaan [38].

Kuberneteksen ohjaustasossa sijaitsee myös valinnainen kube-dns-laajennus, joka sisältää SkyDNS-nimipalvelimen. Nimipalvelimen avulla Kubernetes luo Services-objekteille klusterin sisäisiä verkkotunnuksia. [39]

Kuberneteksen rajapintapalvelin tarkistaa ja asettaa käyttäjän määrittelemiä asetuksia ja luo niistä Kuberneteksen objekteja. Rajapintapalvelin tarjoaa REST-rajapinnan, jonka kautta kaikki Kubernetes klusterin komponentit kommunikoivat [40]. Rajapintapalvelin käyttää etcd-avain-arvo-pari-tietokantaa klusterin tietojen varastoimiseen. [41]

Työnoodessa on komponentit, jolla ajetaan ohjelmistokontteja ja otetaan vastaan komentoja Kuberneteksen ohjaustasolta. Ohjelmistokontteja ja niistä muodostettuja podeja hallinnoi kubelet [36]. Työnoodissa on kube-proxy, jolla Kubernetes toteuttaa Services-objekteja ja luo niillä klusterin sisäisen kuormantasauksen (loadbalancing) [42].

Muita työnoodin komponentteja ovat container runtime ja podien välisen päällysverkon toteuttava CNI-liitännäinen (Container Network Interface). Kubernetes pystyy käyttämään päällysverkon luomiseen Flannel-, OpenVSwitch-, Calico-, Romana-, Contiv- tai Weave Net -liitännäisiä [43].

Kubernetes-klusteria hallinnoidaan kubectl-komentorivityökalulla, joka kommunikoi Kuberneteksen rajapintapalvelimen kanssa. Kubernetekseen luodaan objekteja yaml-merkintäkielellä tai JSON-mallilla määritellyillä asetustiedostoilla [44]. Kubernetekseen voi halutessaan asentaa Dashboard-laajennuksen, jolla klusteria voi tarkastella ja hallinnoida Web-käyttöliittymän kautta [45].

3 CI/CD-JÄRJESTELMÄ

Automaatiolla on myös nykyään suurempi rooli ohjelmistokehitysprosessissa. Automaatiolla ohjelmistokehittäjät pystyvät testaamaan kehitettävän ohjelman toiminnallisuutta ennalta määritettyjen yksikkötestien avulla. Automaatiotoiminnot nopeuttavat ja helpottavat ohjemistokehitys- ja julkaisu-prosessia.

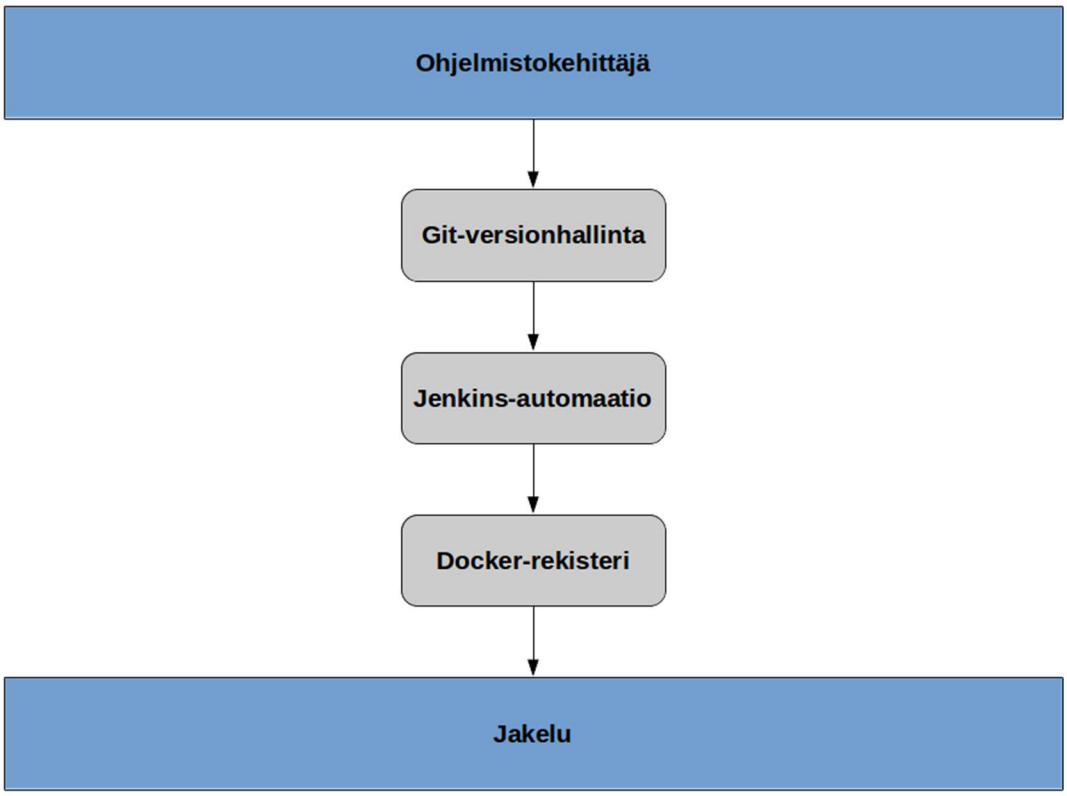
CI/CD on lyhenne englanninkielisistä sanoista Continuous Integration (CI) ja Continuous Delivery (CD). Näillä tarkoitetaan ohjelmistokehityksen työtapoja ja niitä palvelevia ohjelmia. CI määrittelee työtavan, jossa ohjelmoija integroi tuotettavaan ohjelmaan tekemänsä muutokset muille ohjelmoijille jaettuun säilytyspaikkaan [46]. CD on ohjelmistokehityksen prosessi, jossa tuotettavaa ohjelmaa testataan ja se saatetaan levitykseen automattisesti [47]. Yksinkertainen CI/CD-järjestelmä voi koostua versionhallintaohjelmasta, automaatiopalvelimesta ja jakelumekanismin tarjoavasta palvelusta.

Gogs on Gitin kaltainen versionhallintajärjestelmä, jossa ohjelmoijat voivat jakaa, kommentoida ja haaroittaa (fork) toistensa ohjelmistokoodia. Versionhallintajärjestelmä näyttää jokaisen koodimuutoksen tekijän ja muuttuneen kohdan ohjelmakoodissa. Alkuperäisen Git-järjestelmän kirjoitti Linus Torvalds vuonna 2005 [48].

Jenkins on automaatiopalvelin, jolla voi suorittaa mielivaltaisia komentoja verkon yli orjakoneilla. Orjakoneiden käyttöjärjestelmänä voi toimia OS X, Linux tai Windows. Orjakoneiden avulla voidaan suorittaa useampaa koontia rinnakkain eri orjakoneilla. Tätä hyödynnetään erityisesti resurssiraskaissa koonneissa. [49]

Docker-rekisteri on ohjelmistokonttien jakeluun tarkoitettu palvelu, josta voi ladata tai päivittää sen sisältämiä kontteja uudempiin versioihin. Konttimoottori voidaan määrittää tarkastamaan jokaisella ohjelmistokontin käynnistyskerralla, onko kontin levykuva uusin versio ja lataamaan tarvittaessa uusimman levykuvan Docker-rekisteristä.

Järjestelmässä työskennellessään ohjelmistokehittäjä siirtää tekemänsä muutokset versionhallintapalveluun. Kun ohjelmistokehittäjä käynnistää koontiprosessin, Jenkins komentaa testausorjaa hakemaan ohjelmiston lähdekoodin versionhallintapalvelusta, jonka jälkeen testausorja kokoaa sekä yksikkötestaa ohjelmistokontin määriteltyjen testien mukaan. Onnistuneesti testattu ohjelmistokontti siirretään automaattisesti jakeluun yksityiseen Docker-rekisteriin. (Kuva 5.)

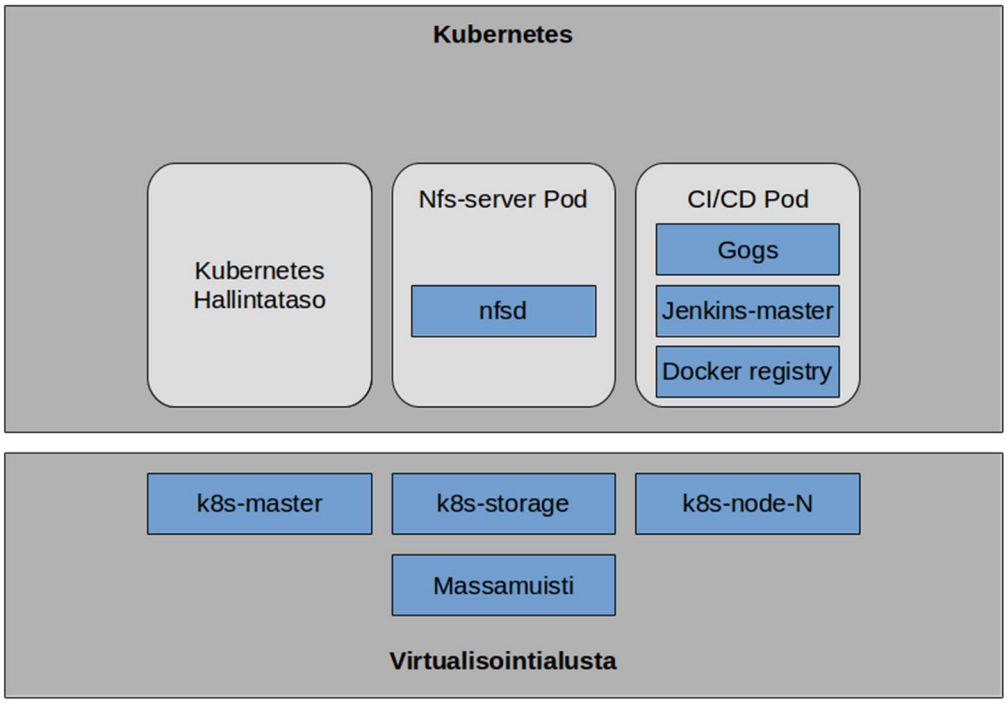


Kuva 5. CI/CD-järjestelmän integraatio.

Kubernetekseen on tarjolla myös valmis avoimen lähdekoodin integroitu kehitysalusta fabric8. Se sisältää kokoelman ohjelmistokehitysprosessia tukevia mikropalveluita. Kehitysalusta sisältää oman käyttöliittymän, joka on integroitu muihin ohjelmistokehitysprosessin mikropalveluihin. Kehitysalustalla voi tuottaa ohjelmistokonttien lisäksi myös .NET-, Django-, Golang-, Java-, Node.js-, PHP- ja Rails -sovelluksia. [50]

3.1 CI/CD-järjestelmän määrittely

Infrastruktuuriksi Kubernetes-asennukselle valittiin VirtualBox-virtualisointialusta. Virtualisointialustalle luotiin neljä virtuaalikonetta master-noodi, storage-noodi ja kaksi normaalia työnoodia. Storage-noodiin luotiin myös erillinen massamuistilaite, jonka hakemistoja noodisidonnainen nfs-palvelin pod jakaa CI/CD-podille NFS-verkkotiedostojärjestelmällä. Tämän avulla CI/CD-pod voi käynnistyä mihin tahansa noodiin ja saada käyttöönsä jossakin toisessa noodissa aikaisemmin ajetun instanssinsa tallentaman hakemiston. (Kuva 6.)

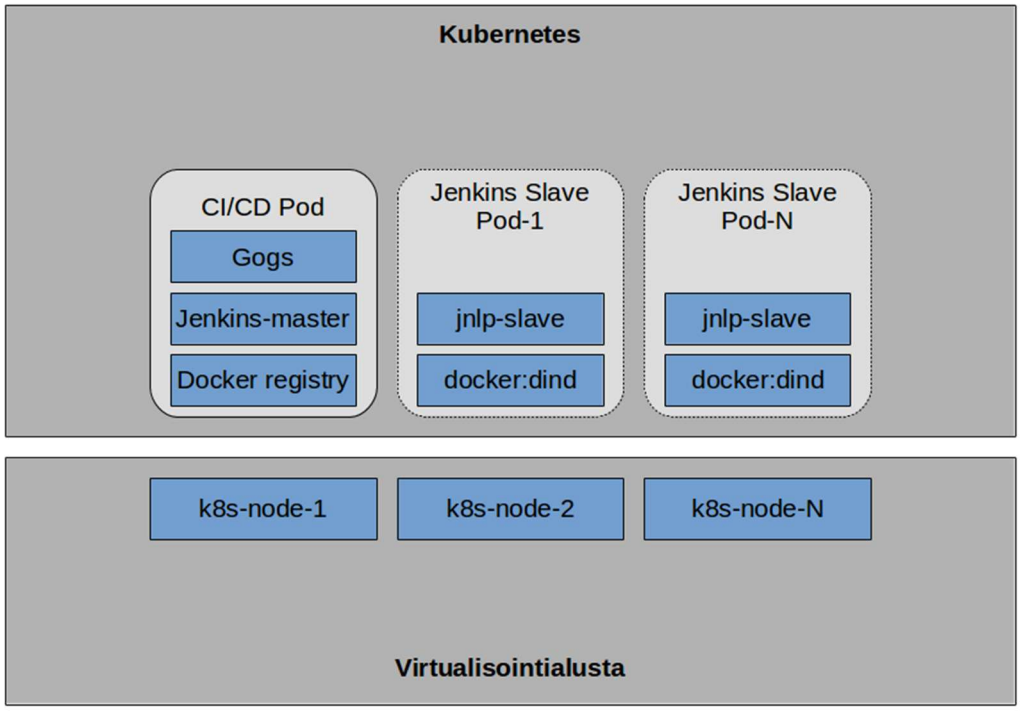


Kuva 6. CI/CD-ympäristö havainnollistettuna.

Kuberneteksellä ajettavia CI/CD-palveluita varten valmisteltiin Deployment-manifesti, jolla määriteltiin mikropalvelut sisältävä pod. Pod-rakenteeseen määriteltiin kolme avoimen lähdekoodin mikropalveluita sisältävät ohjelmistokontit Gogs, Jenkins ja registry. Jokaisen kontin pysyvää tallennustilaa vaativat hakemistopolut määriteltiin yhdistettäväksi nfs-palvelimeen. CI/CD-palveluille luotiin myös oma Services-objekti, jolla määriteltiin palveluiden käyttämät portit. (Liite 1.)

Jotta palveluihin voisi yhdistää noodin ulkopuolelta, määriteltiin kube-keepalived-vip -lisäosalle Daemonset-manifesti ja Configmap-asetusmanifesti. Kube-keepalived-vip tarjoaa podeihin yhdistettäviä virtuaali-IP-osoitteita klusterin sisäisen päällysverkon ulkopuolelle, noodien väliseen verkkoon.

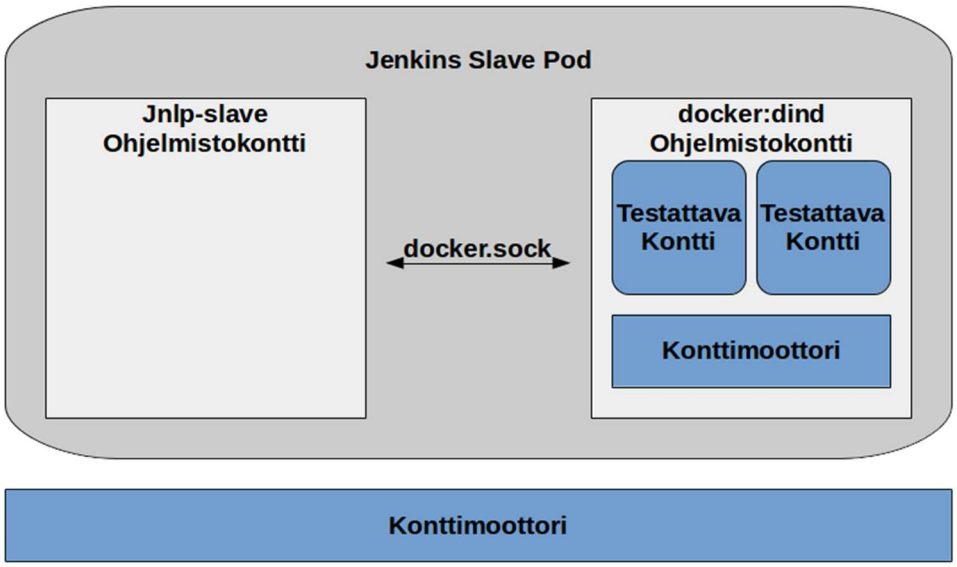
Jenkins-automaatiopalvelimen ohjaamaa testausorjaa varten luotiin Pod-manifesti, jonka Jenkins käynnistää Kubernetekseen saadessaan komennon ohjelmistokehittäjältä tai muulta automaation osalta. Kuberneteksen avulla pyynnöstä käynnistettäviä testausorjia voi myös ajaa useita rinnakkain samassa noodissa, tämän avulla järjestelmällä pystytään suorittamaan useita rinnakkaisia koontiprosesseja. (Kuva 7.)



Kuva 7. Rinnakkaisten testausorjien havainnollistus.

Jenkins testausorjaa varten luotiin Pod-manifesti. Pod koostuu kahdesta muunnellusta ohjelmistokontista, jotka ovat jenkinsci/jnlp-slave ja docker:dind (Docker in Docker). Dind-ohjelmistokontilla luodaan isäntäkoneen konttimoottorista eristetty konttimoottori ohjelmistokontin sisälle, jossa automaatiolla testattavat ohjelmistokontit kootaan ja yksikkötestataan. Podiin määriteltiin ohjelmistokonttien kesken jaettu keskusmuistissa toimiva tmpfs-tiedostojärjestelmä, jonka kautta dind-ohjelmistokontin docker.sock -unix-

pistorasia (unix socket) jaettiin Jenkins-orjaan. Unix-pistorasian kautta Jenkins-orjan Docker-komentorivityökalu pystyy ohjaamaan dind-ohjelmistokontin Docker-palveluprosessia (Docker daemon). Myös Jenkins-orjan työhakemisto ja Docker-palveluprosessin levykuvahakemisto määriteltiin tmpfs-tiedostojärjestelmiksi. (Liite 2.) (Kuva 8.)



Kuva 8. Jenkins Slave Pod havainnollistettuna.

3.2 Kuberneteksen ja CI/CD-järjestelmän asennus

Jokaiseen virtuaalikone noodiin asennettiin käyttöjärjestelmäksi ubuntu 16.04 server Linux ja kubeadm-asennustyökalu Kuberneteksen asennusoppaan mukaisesti [50]. Lisäksi kaikkiin noodeihin asennettiin nfs-asiakasojelma sekä openssh-palvelin komennolla:

```
$ sudo apt-get install nfs-common openssh-server
```

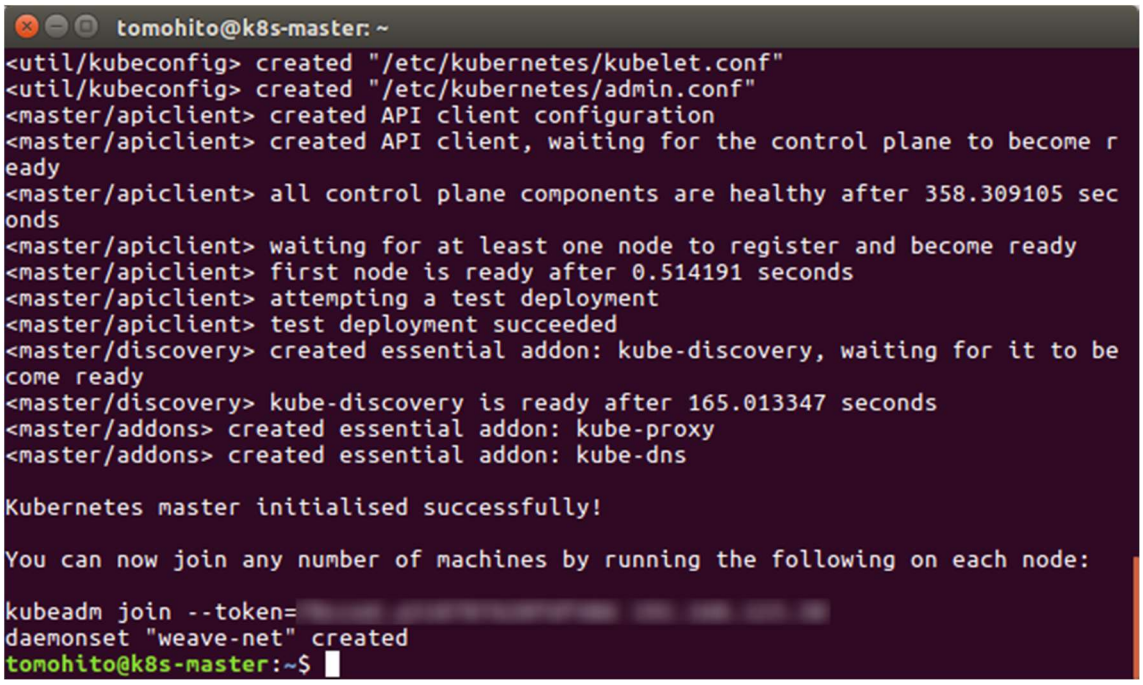
Tulevaa suojaamatonta yksityistä Docker-rekisteriä varten jokaiseen noodiin luotiin asetustiedosto, jossa määriteltiin sen tuleva klusterin sisäinen dns-osoite seuraavalla komennolla:

```
$ echo '{ "insecure-registries":["cicd.cicd.svc.cluster.local:5000"] }' | sudo tee /etc/docker/daemon.json
```

Lisäksi storage-noodiin käynnistettiin nfs-palvelimen tarvitsema rpc-statd-palvelu ja se määriteltiin käynnistyväksi tulevaisuudessa automaattisesti seuraavilla komennolla:

```
$ sudo systemctl start rpc-statd

$ sudo systemctl enable rpc-statd
```

Lopuksi master-noodissa ajettiin kubeadm init-komento, jolla Kubernetes master-noodin palvelut käynnistyivät. Loput noodit liitettiin klusteriin master-noodin asennuksen antaman avaimen avulla. (Kuva 9.)



Kuva 9. Master-noodin asennus.

CI/CD-palveluita varten Kubernetekseen luotiin sille oma cicd-nimiavaruus ja nfs-palvelimen noodisidonnaisuutta varten storage-noodiin asetettiin etiketti seuraavilla komennoilla:

```
$ sudo kubectl create namespace cicd

$ sudo kubectl label nodes k8s-storage role=storage
```

Kuberneteksen nfs-esimerkin pohjalta valmisteltiin, muunneltu Deployment manifesti, jossa määriteltiin itse lisäksi nimialue, noodisidonnaisuus ja massamuistilaitteeseen liitetty hakemistopolku. Se käynnistettiin järjestelmään seuraavalla komennolla:

```
$ sudo kubectl create -f nfs-storage.yaml
```

CI/CD-palvelukokonaisuutta varten määritelty Deployment-manifesti käynnistettiin klusteriin seuraavalla komennolla:

```
$ sudo kubectl create -f cicd.yaml
```

Klusteriin käynnistettiin lopuksi kube-keepalived-vip -Daemonset ja sen käyttämä Configmap-manifesti. Työnoodeihin lisättiin etiketti, jonka perusteella kube-keepalived-vip pod käynnistyy niihin.

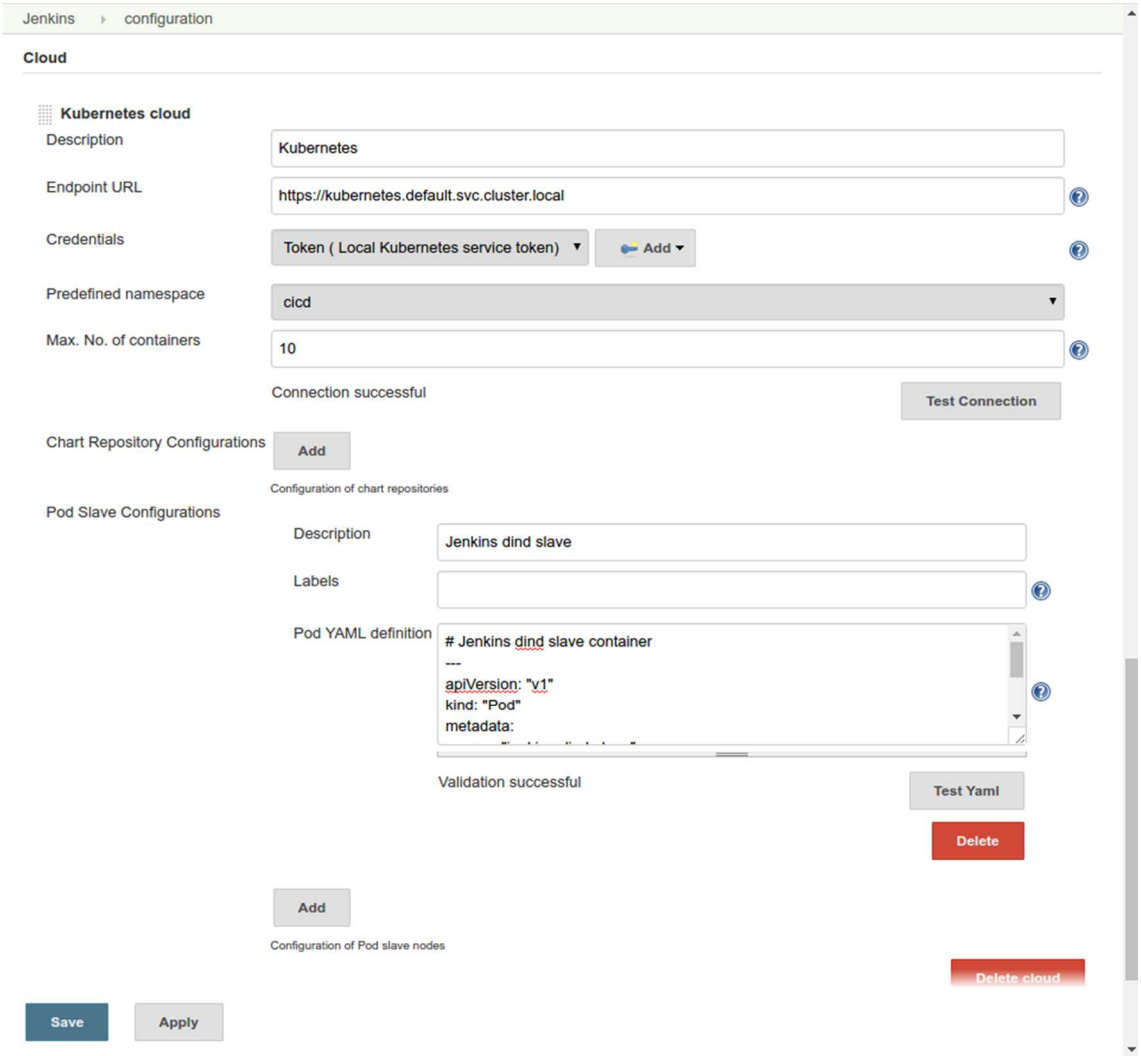
```
$ sudo kubectl create -f kube-keepalived-vip.yaml

$ sudo kubectl label nodes k8s-worker-1 role=worker

$ sudo kubectl label nodes k8s- worker-2 role=worker
```

Kun järjestelmä oli käynnistynyt, Gogs-palveluun määriteltiin asetukset selainasennuksen kautta. Asennuksessa määriteltiin palvelu käyttämään paikallista SQLite3-tietokantaa, joka osoitettiin tallentamaan tietokantansa verkkolevyjakoon liitettyyn hakemistopolkuun. Asennuksen jälkeen Gogs-palveluun luotiin tilit pääkäyttäjälle ja Jenkins-järjestelmälle.

Jenkins-järjestelmän selainasennusta varten haettiin ajossa olevasta Jenkins-ohjelmistokontista asennusavain, joka tarvittiin asennusprosessin aloittamiseen. Tämä tehtiin käyttämällä hyväksi ympäristömuuttujaa, johon haettiin ensin Kubernetes-järjestelmästä CI/CD-podin tunnus, jonka jälkeen tunnuksen avulla tulostettiin ohjelmistokontin sisältä asennusavaimen sisältävä tiedosto.

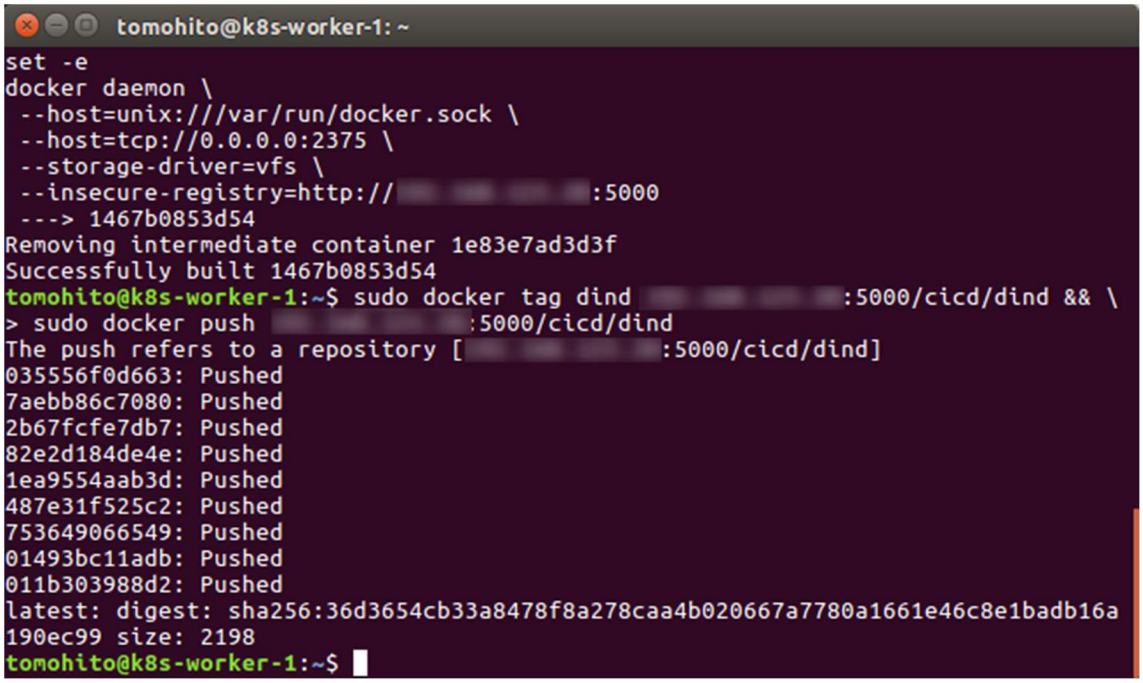
```
$ CICD_POD=$(kubectl get po --namespace=cicd --selector="app=cicd" \

-o=custom-columns=NAME:.metadata.name --no-headers=true)

$ kubectl exec --namespace=cicd "$CICD_POD" -c jenkins-master -t \

-- cat /var/jenkins_home/secrets/initialAdminPassword
```

Jenkins-selainasennuksessa valittiin Git-, build timeout- ja Kubernetes CI-liitännäiset. Manage Jenkins -> Configure system asetussivulla asetettiin käännös-suorittajan (build executor) määrä nollaan sekä Kubernetes-liitännäiseen määriteltiin seuraavat asetukset. Aikaisemmin määritelty Jenkins-testausorja Pod-manifesti syötettiin Jenkins Kubernetes-liitännäiseen. (Kuva 10.)



Kuva 10. Jenkins Kubernetes-liitännäisen asetukset.

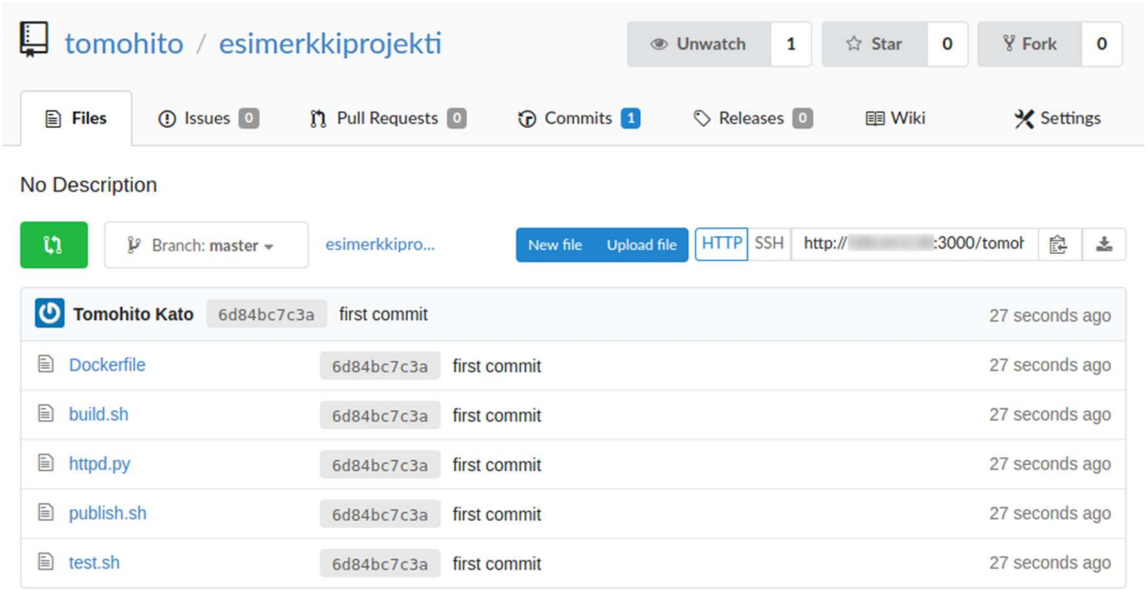
Jotta Jenkins-slave pod voitiin käynnistää ensimmäistä kertaa, piti sen käyttämät muunnellut ohjelmistokontit ensin luoda ja siirtää CI/CD-järjestelmän Docker-rekisteriin. Kubernetes-klusterin noodiin luotin Dockerfile-konttimanifestit muokattuja jenkinsci/jnlp-slave ja docker:dind -ohjelmistokontteja varten. Ohjelmistokontit koottiin paikallisesti ja siirrettiin CI/CD-järjestelmän yksityiseen Docker-rekisteriin. (Liite 3.) (Liite 4.) (Kuva 11.)



Kuva 11. Dind-ohjelmistokontin kokoaminen ja siirtäminen yksityiseen rekisteriin.

3.3 Esimerkkiprojektin ajaminen CI/CD-järjestelmässä

Järjestelmän toiminallisuuden testaamiseksi Gogs-versionhallintapalveluun luotiin esimerkkiprojekti, johon valmisteltiin yksikertainen Python-webpalvelin, ohjelmistokontin koontimanifesti ja sen koontiin, yksikkötestaamiseen ja julkaisemiseen käytettävät bash-skriptit. (Kuva 12.)

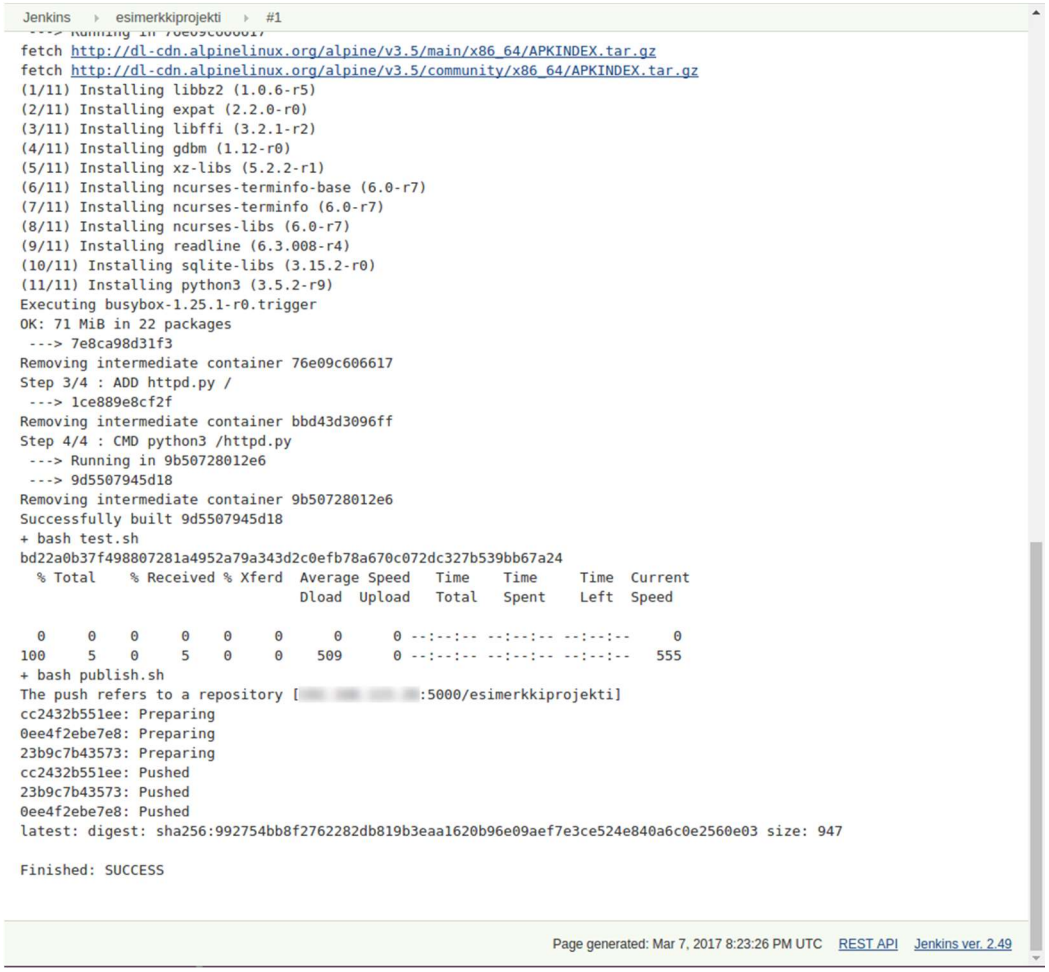


Kuva 12. Esimerkkiprojekti Gogs-versionhallinnassa.

Jenkins-automaatiopalveluun luotiin projekti, joihin määriteltiin Gogs-versionhallintapalvelun osoite ja tunnukset. Versionhallintapalvelusta saataville bash-skripteille määriteltiin suorituskomennot. (Kuva 13.)

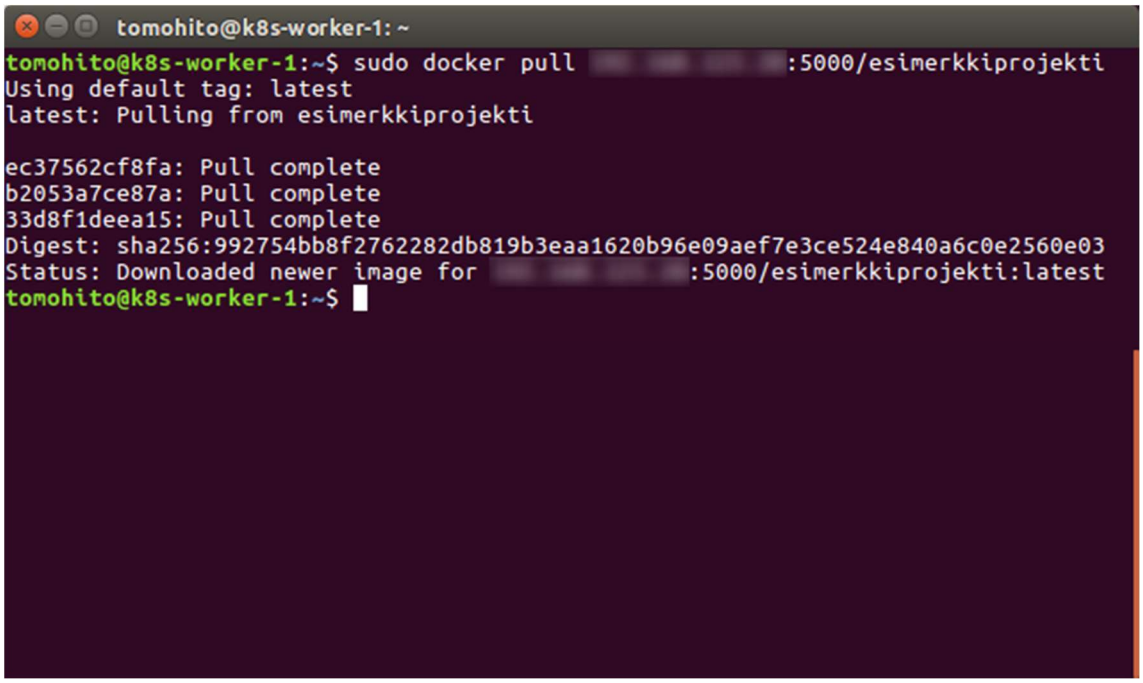
Kuva 13. Esimerkkiprojektin koontikomennot.

Kun projektin määritelty oli tallennettu, se käynnistettiin ajoon. Jenkins käynnisti automaattisesti Kubernetekseen testausorjan, jossa koonnin komennot suoritetaan. Koonnin jälkeen lokitiedostoja tarkastelemalla voitiin havaita ohjelmistokontin koonnin, yksikkötestauksen ja jakeluun saattamisen onnistuneen. (Kuva 14.)



Kuva 14. Koonnin lokitiedosto.

Lopuksi testattiin vielä, että ohjelmistokontti on saatavilla yksityisestä rekisteristä lataamalla se klusterin noodiin. (Kuva 15.)



Kuva 15. Esimerkkiprojektin ohjelmistokontin lataaminen k8s-worker-1 -noodiin.

Näin ohjelma koottiin, testattiin ja saatettiin levitykseen ohjelmistokontin muodossa.

4 ARVIONTI

Päättötyön tekeminen oli erittäin haasteellista, koska osa käytetyistä avoimen lähdekoodin ohjelmista on vielä osittain keskeneräisiä ja toimintojen dokumenteissa sekä käyttöohjeissa on puutteita. Myös osa käytetyistä Kubernetes version 1.4 objekteista oli vielä alpha- tai beta-kehitysasteella. Tämän takia objektin määrittelyyn piti usein hakea apua lukemalla objektin rajapintamäärityksiä. Myöskään kaikki järjestelmät eivät olleet täysin keskenään integroituvia, kuten Jenkins Kubernetes CI-liitännäisen 1.2 versio, joka sisälsi mahdollisen ohjelmointivirheen.

Ohjelmointivirheen johdosta liitännäinen syötti Jenkins-orjaan tarkoitettuja argumentteja myös muihin saman podin ohjelmistokontteihin. Tästä syystä ei voitu käyttää suoraan virallista docker:dind-ohjelmistokonttia, vaan siitä tehtiin muokattu versio, joka ohitti väärin syötetyt argumentit.

Haasteellisuudesta huolimatta lopputulokseen päästiin onnistuneesti ja järjestelmällä pystyttiin tekemään automatisoituja kokoamisprosesseja. Suurempien ohjelmistokonttien kokoamisprosessit eivät onnistuneet pohjalla toimivien virtuaalikoneiden väärin mitoitettujen resurssien takia.

Työssä käytetty docker:1.12.3-dind-ohjelmistokontti oli myös ongelmallinen. Sen alkuperäisen kehittäjän Jérôme Petazzonin mukaan eri varastoajuriyhdistelmät eivät ole yhteensopivia keskenään ja saattaisivat aiheuttaa ongelmia eri kombinaatiolla [52]. Ohjelmistokontin käyttöoppaan mukaan sisäinen konttimoottori käyttää varastoajurinaan VFS-tiedostojärjestelmää, joka on hidas, mutta ainoa varastoajuri, joka toimii minkä tahansa isäntäjärjestelmän Docker-taustaprosessin käyttämän varastoajurin kanssa [53]. Ehkä tulevaisuudessa taustalla toimivien copy-on-write- ja union-tiedostojärjestelmien kehittyessä docker:dind-ohjelmistokontti kehittyy yhä vakaammaksi.

Muita toteutustapoja ohjelmistokonttien testaamiseen Jenkins-orjalla olisi ollut käyttää isäntäkoneen konttimoottoria suoraan Jenkins-orjasta, jolloin testattavat ohjelmistokontit olisivat käynnistyneet sisaruskontteina Kuberneteksen omien ohjelmistokonttien rinnalle. Näin toteutettuna Jenkins-orjan käynnistämät testattavat kontit saattaisivat jäädä taustalle ajoon testausprosessin loputtua. Ongelmalliseksi olisi myös muodostunut isäntäkoneen docker.sock -unix-pistorasian liittäminen orjaan, koska Kubernetes ei tue tällä hetkellä yksittäisten unix-pistorasioiden jakamista ohjelmistokonttien välillä. Tällöin olisi

isäntäkäyttöjärjestelmän koko /var/run/-hakemistopolku pitänyt liittää Jenkins-orjaan. Tämä olisi kuitenkin paljastanut kaikki isäntäkoneen unix-pistorasiat Jenkins-orjalle.

Kolmas tapa olisi ollut pystyttää erillinen testauskone, jossa ajettaisiin ainoastaan docker-palveluprosessia ja Jenkins-orjaa. Tällöin olisi kuitenkin menetetty Kubernetes klusterin avulla saatu skaalattavuus.

Koska työ oli luonteeltaan kokeellinen, määritelty CI/CD-järjestelmä jäi puutteelliseksi tietoturvan puolesta eikä sovellu sellaisenaan tuotantokäyttöön. Tietoturvan tasoa olisi voinut parantaa lisäämällä palveluun TLS-sertifikaatit verkkoyhteyden salaamiseksi. Myös Docker-rekisteri ja NFS-palvelin jäivät ilman käyttäjän todennusta.

Jos palvelut ajettaisiin kaupallisella GCE:llä noodisidonnaisen nfs-palvelimen verkkolevyjaon sijaan, palvelinten pysyviä tiedostoja voisi tallettaa GCE:n omiin levyihin gcePersistentDisk-objekteilla. GCE tarjoaa myös oman sisäänrakennetun ohjelmistokontti-rekisterin, jonka avulla CI/CD-järjestelmän Docker-rekisteristä voisi luopua.

Vaikka Kubernetes on ollut tuotantovalmis jo pitkään, se on jatkuvasti kehittyvä projekti. Vasta kehitysasteella olevat uudet objektit tuovat Kubernetekselle uusia käyttötarkoituksia tulevaisuudessa.

LÄHTEET

[1] O. Beattie, Kirjoittaja, *Building a Bank with Kubernetes.* [Performance]. Monzo, 2016.

[2] Open Containers Initiative, [Online]. Available: https://www.opencontainers.org/. [Haettu marraskuu 2016].

[3] S. Hykes, Kirjoittaja, *Introduction to Docker.* [Performance]. Twitter University, 2013.

[4] Wikipedia, ”Operating-system-level virtualization,” [Online]. Available: https://en.wikipedia.org/wiki/Operating-system-level_virtualization. [Haettu lokakuu 2016].

[5] Wikipedia, ”Hypervisor,” [Online]. Available: https://en.wikipedia.org/wiki/Hypervisor. [Haettu lokakuu 2016].

[6] Docker, ”What is a Container,” [Online]. Available: https://www.docker.com/what-container. [Haettu Toukokuu 2017].

[7] IBM Research, ”An Updated Performance Comparison of Virtual Machines and Linux Containers,” 2014.

[8] opensource.com, ”Containers, microservices, and orchestrating the whole symphony,” [Online]. Available: https://opensource.com/business/14/12/containers-microservices-and-orchestrating-whole-symphony. [Haettu lokakuu 2016].

[9] Docker, ”Docker Registry,” [Online]. Available: https://docs.docker.com/registry/. [Haettu marraskuu 2016].

[10] Docker, ”Understand images, containers, and storage drivers,” [Online]. Available: https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/. [Haettu lokakuu 2016].

[11] Linux Programmer’s Manual, ”Namespaces,” [Online]. Available: http://man7.org/linux/man-pages/man7/namespaces.7.html. [Haettu lokakuu 2016].

[12] Linux Programmer’s Manual, ”Cgroups,” [Online]. Available: http://man7.org/linux/man-pages/man7/cgroups.7.html. [Haettu lokakuu 2016].

[13] Docker, ”Docker security,” [Online]. Available: https://docs.docker.com/engine/security/security/. [Haettu lokakuu 2016].

[14] Docker, ”Select a storage driver,” [Online]. Available: https://docs.docker.com/engine/userguide/storagedriver/selectadriver/. [Haettu lokakuu 2016].

[15] CoreOS, ”rkt vs other projects,” [Online]. Available: https://coreos.com/rkt/docs/latest/rkt-vs-other-projects.html. [Haettu lokakuu 2016].

[16] Kubernetes, ”What is Kubernetes,” [Online]. Available: http://kubernetes.io/docs/whatisk8s/. [Haettu syyskuu 2016].

[17] TechCrunch, ”As Kubernetes Hits 1.0, Google Donates Technology To Newly Formed Cloud Native Computing Foundation,” [Online]. Available: https://techcrunch.com/2015/07/21/as-kubernetes-hits-1-0-google-donates-technology-to-newly-formed-cloud-native-computing-foundation-with-ibm-intel-twitter-and-others/. [Haettu syyskuu 2016].

[18] Google, ”Large-scale cluster management at Google with Borg,” 2015.

[19] Kubernetes, ”A Very Happy Birthday Kubernetes,” [Online]. Available: http://blog.kubernetes.io/2016/07/happy-k8sbday-1.html. [Haettu syyskuu 2016].

[20] Google Cloud Platform, ”Welcome Microsoft, RedHat, IBM, Docker and more to the Kubernetes community,” [Online]. Available: https://cloudplatform.googleblog.com/2014/07/welcome-microsoft-redhat-ibm-docker-and-more-to-the-kubernetes-community.html. [Haettu lokakuu 2016].

[21] Kubernetes, ”Picking the Right Solution,” [Online]. Available: http://kubernetes.io/docs/getting-started-guides/. [Haettu lokakuu 2016].

[22] Kubernetes, ”Kubernetes and Cluster Federation Control Plane Resilience,” [Online]. Available: https://github.com/kubernetes/kubernetes/blob/release-1.4/docs/design/control-plane-resilience.md. [Haettu lokakuu 2016].

[23] Kubernetes, ”Labels and Selectors,” [Online]. Available: http://kubernetes.io/docs/user-guide/labels/. [Haettu syyskuu 2016].

[24] Kubernetes, ”Pods,” [Online]. Available: http://kubernetes.io/docs/user-guide/pods/. [Haettu syyskuu 2016].

[25] Kubernetes, ”Replication Controller,” [Online]. Available: http://kubernetes.io/docs/user-guide/replication-controller/. [Haettu syyskuu 2016].

[26] Kubernetes, ”Replica Set,” [Online]. Available: http://kubernetes.io/docs/user-guide/replicasets/. [Haettu syyskuu 2016].

[27] Kubernetes, ”Deployments,” [Online]. Available: http://kubernetes.io/docs/user-guide/deployments/. [Haettu syyskuu 2016].

[28] Kubernetes, ”Daemon Sets,” [Online]. Available: http://kubernetes.io/docs/admin/daemons/. [Haettu syyskuu 2016].

[29] Kubernetes, ”Services,” [Online]. Available: http://kubernetes.io/docs/user-guide/services/. [Haettu syyskuu 2016].

[30] Kubernetes, ”Ingress Resources,” [Online]. Available: http://kubernetes.io/docs/user-guide/ingress/. [Haettu syyskuu 2016].

[31] Kubernetes, ”Secrets,” [Online]. Available: http://kubernetes.io/docs/user-guide/secrets/. [Haettu lokakuu 2016].

[32] Kubernetes, ”Using ConfigMap,” [Online]. Available: http://kubernetes.io/docs/user-guide/configmap/.

[33] Kubernetes, ”Persistent Volumes,” [Online]. Available: http://kubernetes.io/docs/user-guide/persistent-volumes/. [Haettu lokakuu 2016].

[34] Kubernetes, ”Namespaces,” [Online]. Available: http://kubernetes.io/docs/user-guide/namespaces/. [Haettu lokakuu 2016].

[35] Kubernetes, ”Understanding Resource Quotas,” [Online]. Available: http://kubernetes.io/docs/admin/resourcequota/. [Haettu lokakuu 2016].

[36] Kubernetes, ”Kubernetes architecture,” [Online]. Available: https://github.com/kubernetes/kubernetes/blob/master/docs/design/architecture.md. [Haettu syyskuu 2016].

[37] Kubernetes, ”kube-scheduler,” [Online]. Available: http://kubernetes.io/docs/admin/kube-scheduler/. [Haettu syyskuu 2016].

[38] Kubernetes, ”kube-controller-manager,” [Online]. Available: http://kubernetes.io/docs/admin/kube-controller-manager/. [Haettu syyskuu 2016].

[39] Kubernetes, ”DNS in Kubernetes,” [Online]. Available: https://github.com/kubernetes/kubernetes/blob/master/build/kube-dns/README.md. [Haettu syyskuu 2016].

[40] Kubernetes, ”kube-apiserver,” [Online]. Available: http://kubernetes.io/docs/admin/kube-apiserver/. [Haettu syyskuu 2016].

[41] Kubernetes, ”Kubernetes Components,” [Online]. Available: http://kubernetes.io/docs/admin/cluster-components/. [Haettu syyskuu 2016].

[42] Kubernetes, ”kube-proxy,” [Online]. Available: http://kubernetes.io/docs/admin/kube-proxy/. [Haettu syyskuu 2016].

[43] Kubernetes, ”Networking in Kubernetes,” [Online]. Available: http://kubernetes.io/docs/admin/networking/. [Haettu lokakuu 2016].

[44] Kubernetes, ”kubectl overview,” [Online]. Available: http://kubernetes.io/docs/user-guide/kubectl-overview/. [Haettu lokakuu 2016].

[45] Kubernetes, ”Kubernetes Dashboard,” [Online]. Available: https://github.com/kubernetes/dashboard/blob/master/README.md. [Haettu lokakuu 2016].

[46] Wikipedia, ”Continuous integration,” [Online]. Available: https://en.wikipedia.org/wiki/Continuous_integration. [Haettu marraskuu 2016].

[47] Wikipedia, ”Continuous delivery,” [Online]. Available: https://en.wikipedia.org/wiki/Continuous_delivery. [Haettu marraskuu 2016].

[48] Wikipedia, ”Git,” [Online]. Available: https://en.wikipedia.org/wiki/Git. [Haettu marraskuu 2016].

[49] Jenkins, ”Distributed builds,” [Online]. Available: https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds. [Haettu marraskuu 2016].

[50] fabric8, ”https://fabric8.io/,” [Online]. [Haettu marraskuu 2016].

[51] Kubernetes, ”Installing Kubernetes on Linux with kubeadm,” [Online]. Available: http://kubernetes.io/docs/getting-started-guides/kubeadm/. [Haettu marraskuu 2016].

[52] J. Petazzo, ”Using Docker-in-Docker for your CI or testing environment? Think twice.,” [Online]. Available: https://jpetazzo.github.io/2015/09/03/do-not-use-docker-in-docker-for-ci/. [Haettu joulukuu 2016].

[53] Docker Hub, ”Docker in Docker!,” [Online]. Available: https://hub.docker.com/_/docker/. [Haettu joulukuu 2016].

Liite 1. CI/CD Deployment -manifesti

```
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: cicd
  namespace: cicd
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cicd
  template:
    metadata:
      labels:
        app: cicd
    spec:
      terminationGracePeriodSeconds: 60
      containers:
      - name: docker-registry
        image: registry:2
        ports:
        - name: http-registry
          containerPort: 5000
        env:
        - name: REGISTRY_HTTP_ADDR
          value: :5000
        volumeMounts:
        - name: pvc
          subPath: registry
          mountPath: /var/lib/registry/docker/registry
      - name: gogs
        image: gogs/gogs:latest
        ports:
        - name: ssh-gogs
          containerPort: 22
        - name: http-gogs
          containerPort: 3000
        volumeMounts:
        - name: pvc
          subPath: gogs
          mountPath: /data
        env:
        - name: SOCAT_LINK
          value: "false"
      - name: jenkins-master
        image: jenkinsci/jenkins:latest
        ports:
        - name: http-jenkins
          containerPort: 8080
        - name: https-jenkins
          containerPort: 50000
        securityContext:
          runAsUser: 0
        volumeMounts:
        - name: nfs-volume
          subPath: jenkins
          mountPath: /var/jenkins_home/
      dnsPolicy: Default
      volumes:
      - name: nfs-volume
```

```
        nfs:
          server: nfs-storage.cicd.svc.cluster.local
          path: "/exports"
---
apiVersion: v1
kind: Service
metadata:
  name: cicd
  namespace: cicd
spec:
  clusterIP: 100.64.0.30
  ports:
  - name: ssh-gogs
    port: 22
  - name: http-gogs
    port: 3000
  - name: http-registry
    port: 5000
  - name: http-jenkins
    port: 8080
  - name: https-jenkins
    port: 50000
  selector:
    app: cicd
---
```

Liite 2. Jenkins-slave Pod -manifesti

```
---
apiVersion: "v1"
kind: "Pod"
metadata:
  name: "jenkins-dind-slave"
  labels:
    name: "jenkins-dind-slave"
spec:
  containers:
  - name: "jenkins-slave"
    image: "cicd.cicd.svc.cluster.local:5000/cicd/jenkins-slave"
    env:
    - name: "JENKINS_URL"
      value: " cicd.cicd.svc.cluster.local:8080"
    volumeMounts:
    - name: docker-socket
      mountPath: /var/run/
    - name: jenkins-workspace
      mountPath: /home/jenkins/workspace/
  - name: "dind"
    image: "cicd.cicd.svc.cluster.local:5000/cicd/dind"
    securityContext:
      privileged: true
    volumeMounts:
    - name: docker-socket
      mountPath: /var/run/
    - name: docker-images
      mountPath: /var/lib/docker/
  dnsPolicy: Default
  volumes:
  # tmpfs volumes
  - name: docker-socket
    emptyDir:
      medium: "Memory"
  - name: jenkins-workspace
    emptyDir:
      medium: "Memory"
  - name: docker-images
    emptyDir:
      medium: "Memory"
---
```

Liite 3. Jenkins-slave Dockerfile -koontimanifesti

```
FROM jenkinsci/jnlp-slave

USER root
RUN apt-get update && apt-get install -y \
    docker.io \
    curl \
    dnsutils \
    && rm -rf /var/lib/apt/lists/*
```

Liite 4. Dind Dockerfile -koontimanifesti

```
FROM docker:dind

RUN { \
      echo '#!/bin/sh'; \
      echo 'set -e'; \
      echo 'docker daemon \'; \
      echo ' --host=unix:///var/run/docker.sock \'; \
      echo ' --host=tcp://0.0.0.0:2375 \'; \
      echo ' --storage-driver=vfs \'; \
      echo ' --insecure-registry=cicd.cicd.svc.cluster.local:5000'; \
    } | tee /usr/local/bin/dockerd-entrypoint.sh \
    && chmod +x /usr/local/bin/dockerd-entrypoint.sh
```