VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Niko Lehto

MODULAARINEN PALVELUALUSTA

Tekniikka
2017

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Niko Lehto
Opinnäytetyön nimi	Modulaarinen palvelualusta
Vuosi	2017
Kieli	suomi
Sivumäärä	65
Ohjaaja	Timo Kankaanpää

Opinnäytetyön tavoitteena oli luoda arkkitehtuurimalli, jonka avulla voidaan toteuttaa mihin tahansa ulkoisiin järjestelmiin kytkettävä modulaarinen palvelualusta. Työssä tehtiin myös alustava integraatio kahteen ulkoiseen järjestelmään arkkitehtuurimallia hyödyntäen.

Tuotantokelpoiseen ja projektin vaatimukset täyttävään palvelualustaan vaaditaan sekä palvelin- että selainsovelluksen toteutus. Tässä työssä keskityttiin toteutuksen osalta ainoastaan palvelinsovellukseen. Palvelinsovelluksen arkkitehtuurimalli ja palveluabstraktio eivät ota kantaa sen kautta tarjottavien palveluiden rakenteeseen. Opinnäytetyössä palvelualustaa kuitenkin tarkasteltiin hosting-tarjoajan apuvälineeksi suunnattuna portaalina, jota käyttämällä asiakkaat voivat tilata ja ottaa automaattisesti käyttöönsä eri komponenteista koostuvia web-palveluja. Kyseisessä implementaatiossa palvelun ylimmän tason komponentti on web-sovellus.

Useita lähestymistapoja kokeilemalla saatiin lopputuloksena luotua kehityskelpoinen, laajennettavissa oleva palvelualusta. Luodulla arkkitehtuurimallilla lopullinen tuotantokäyttöön tehtävä ratkaisu saadaan toteutettua nopeasti. Palvelualustan rakentaminen on ollut työn toimeksiantajan suunnitelmissa jo jonkin aikaa, ja opinnäytetyön myötä saatiin selvitettyä lisää siihen liittyviä tarpeita sekä käynnistettyä sovelluksen varsinainen kehitys. Lisäksi valittiin E2E-testaukseen käytettävät työkalut sekä arkkitehtuuri, jonka mukaan testit kirjoitetaan.

Avainsanat	palvelualusta, järjestelmäintegraatio, hosting, sovellusarkkitehtuurit

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Niko Lehto
Title	Modular Service Platform
Year	2017
Language	Finnish
Pages	65
Name of Supervisor	Timo Kankaanpää

The objective of this thesis was to create an architectural model that can be used to build a modular service platform which can be connected to any external system. An initial integration to two external systems was also implemented in the scope of the thesis.

Both a server application and a browser application are needed to compose a service platform that fulfills the requirements of the project and is valid for production use. In this thesis, the focus was on the server application. The architectural model and the service abstraction in the server application are not tied into any specific service type. However, in this thesis the service platform was considered as a portal for a hosting provider through which the customers can order and automatically deploy web services that consists of multiple components. In this specific implementation, the top-level component of a service was a web application.

After several architectural approaches, a developable and extendable service platform was created. With the created architectural model, the final solution for production environment can be implemented in the near future. The mandator of the thesis has been planning to create a service platform for a while, and during this thesis project, the requirements became more accurate and the development of the software was actually started. In addition, the E2E testing tools and architecture were chosen.

Keywords	service platform, system integration, hosting, software architectures

SISÄLLYS

KUVA- JA TAULUKKOLUETTELO

KÄSITELUETTELO

Hosting	Web-palvelun ylläpidollisista seikoista huolehtiminen jonkin tahon puolesta.
Buildaus	Kokonaisvaltaisen ja suoritettavan artefaktin luominen kaikista siihen vaadittavista riippuvuuksista.
Repository	Keskitetty tietovarasto.
Triggeri	Toiminto, joka laukaisee muita toimintoja.
PaaS	Platform as a Service. Pilvipalvelumalli, joka keskittyy sovellusten käyttöönoton helpottamiseen. Taustalla olevan infrastruktuurin hallintavastuu on palveluntarjoajalla. /45/
REST	Representational State Transfer. Tiettyihin rajoitteisiin perustuva arkkitehtuurimalli hajautetuille hypermediajärjestelmille. /46/
API	Application Programming Interface. Ohjelmointirajapinta. /47/
URI	Uniform Resource Identifier. Tietyllä syntaksilla muodostettu merkkijono, joka identifioi resurssin. /48/
URL	Uniform Resource Locator. URI:n osajoukko, joka identifioinnin lisäksi paikantaa resurssin. /48/
HTTP	Hypertext Transfer Protocol. Tilaton sovellusprotokolla hajautetuille, yhdessä toimiville hypertekstijärjestelmille. /49/
TLS	Transport Layer Security. Kahden kommunikoivan sovelluksen välissä käytettävä suojausprotokolla. /53/
SASL	Simple Authentication and Security Layer. Kehys, joka tarjoilee autentikointi- ja turvallisuuspalveluita yhteysprotokollille. /31/
JSON	JavaScript Object Notation. Eräs tiedonvälitysformaatti. /51/

XML	Extensible Markup Language. Eräs merkintäkieli. /52/
TCP	Transmission Control Protocol. Yhteydellinen tietoliikeneprotokolla. /50/
IP	Internet Protocol. Protokolla, joka huolehtii datagrammien välittämisestä pakettikytkentäisessä verkossa. /54/
ITU-T	ITU Telecommunication Standardization Sector /28/
UDP	User Datagram Protocol. Yhteydetön tietoliikenneprotokolla. /55/
JAX-RS	Java API for RESTful Web Services. /56/
ORM	Object Relational Mapping. Relaatiomallin muuttaminen oliomallin mukaiseksi. /60/
BDD	Behavior-driven development. Käyttäytymislähtöinen kehitys. Automaatiotestauksessa: testataan käyttäytymistä eikä implementaatiota. /61/
CI	Continuous Integration. Jatkuva integraatio; kehittäjät integroivat koodinsa säännöllisesti repositoryyn, jonka jälkeen integraatio voidaan verifioida automaattisella buildauksella ja testauksella. /62/
E2E	End-to-End. Testauksessa: testataan koko sovelluksen toiminta alusta loppuun, esimerkiksi käyttöliittymästä ulkoisiin järjestelmiin asti.

1 JOHDANTO

Tämän opinnäytetyön toimeksiantaja on Jubic Oy, joka on vaasalainen vuonna 2014 perustettu IT-alan yritys. Jubic Oy keskittyy ohjelmistojen ja tietojärjestelmien kehityspalveluiden sekä hosting-ratkaisujen tarjoamiseen, ja sen asiakkaat koostuvat teollisuuden eri aloilla toimivista yrityksistä. /1/

Opinnäytetyön tarkoituksena on suunnitella ja toteuttaa sovellusarkkitehtuuri, jota hyödyntäen voidaan rakentaa modulaarinen, lähes rajattomasti skaalattavissa oleva palvelualusta. Sovellus myös integroidaan muutamiin ulkoisiin järjestelmiin luotua arkkitehtuurimallia käyttäen.

Sovelluksen datamalli ei ota kantaa palveluiden rakenteeseen, joten teoriassa se voitaisiin ottaa käyttöön minkä tahansa teollisuudenalan palvelualustaksi. Tässä työssä sovellusta kuitenkin tarkastellaan hosting-tarjoajan sekä -asiakkaan välissä olevana web-palvelualustana. Sen avulla hosting-asiakkaat voivat tilata ja hallinnoida erityyppisiä web-palveluja. Hosting-tarjoaja pystyy alustan kautta tarkastelemaan tilattuja palveluja, hallitsemaan käyttöoikeuksia sekä määrittelemään palvelurakenteet, joita asiakkaille tarjoillaan.

Web-palvelualustan tarkoitus on keskittää kaikki hosting-tarjoajan ja asiakkaan välillä tapahtuvat toimet yhteen paikkaan, mikä helpottaa kummankin osapuolen päivittäistä työskentelyä. Markkinoilta löytyy useita samaan käyttötarkoitukseen soveltuvia alustoja, mutta tämän palvelualustan ehdottomia vahvuuksia ovat skaalautuvuus sekä teknologiariippumattomuus. Alustan kautta tilattuun palveluun voi esimerkiksi kuulua millä tahansa tekniikoilla toteutettu web-sovellus, joka voidaan ottaa käyttöön automatisoidusti. Alusta voidaan myös integroida liiketoimintalogiikkaa muuttamatta mihin tahansa ulkoisiin järjestelmiin, joten eri asiakkaiden muuttuvat tarpeet voidaan huomioida nopeasti, vaikka alusta olisikin jo tuotantoympäristössä.

Projekti koostuu käyttöliittymä- sekä palvelinohjelman toteutuksesta. Tässä opinnäytetyössä keskitytään toteutuksen osalta palvelinsovelluksen arkkitehtuurin kuvaamiseen. Määrittely-, suunnittelu- ja vertailuosioissa projektin lopputuotetta

tarkastellaan kokonaisuutena. Projektin lopputuotteeseen viitataan tekstissä sen työnimellä MSP (Modular Service Platform).

2 ULKOISET JÄRJESTELMÄT

Tässä luvussa kuvataan ulkoiset järjestelmät, joihin MSP ensisijaisesti integroidaan. Luvussa keskitytään kuvaamaan kunkin järjestelmän keskeisin toimintaperiaate sekä termit, jotka tulevat esille tämän dokumentin määrittely-, suunnittelu- ja toteutusvaiheessa.

OpenShiftin rooli MSP:n ulkoisena järjestelmänä on toimia web-sovellusten käyttöönottoalustana. Redmine toimii tehtävienhallintajärjestelmänä, jonne MSP lähettää tehtäviä suoritettavaksi MSP:n hallinnoijaosapuolelle.

2.1 OpenShift

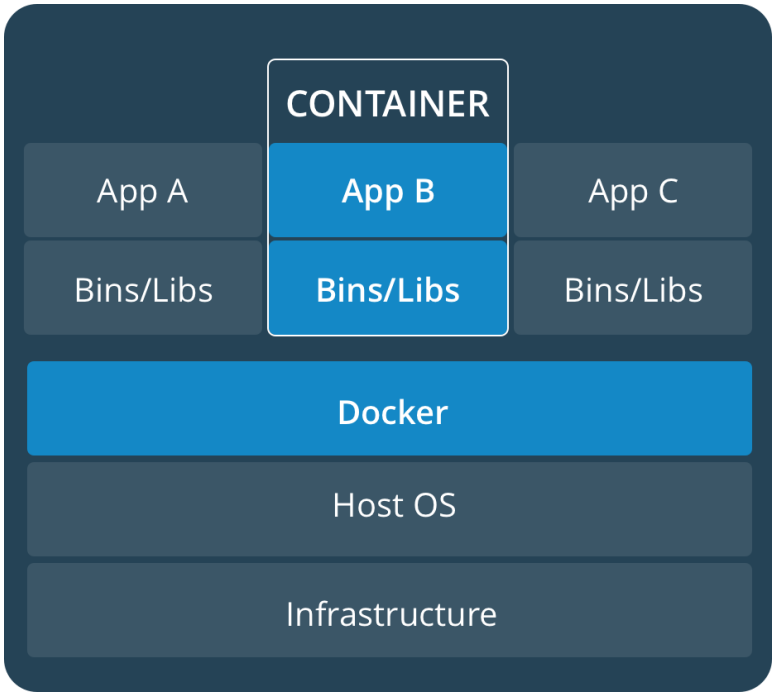
OpenShift on Red Hatin kehittämä PaaS-alusta, jonka ensimmäinen versio julkaistiin vuonna 2011 /2, 3/. Se on pääasiassa ohjelmistokehittäjien sekä ylläpitäjien tarpeisiin kehitetty työkalu, jonka avulla helpotetaan sovellusten kehittämistä, käyttöönottamista ja suorittamista. OpenShiftin PaaS-alusta muodostuu Dockerista, Kubernetesista sekä niiden päälle tehdystä varsinaisesta OpenShift-kerroksesta. /3/

2.1.1 Docker

Docker on avoimen lähdekoodin alusta sovellusten paketointiin ja suorittamiseen isoloiduissa *containereissa*. Sen ensimmäinen versio julkaistiin vuonna 2013. /4, 6/ Yksittäiseen Docker-containeriin paketoidaan tietyn sovelluksen suorittamiseen tarvittavat järjestelmä- ja sovellustason riippuvuudet sekä asetukset, millä varmistetaan sovelluksen toiminta käyttöönottovaiheessa millä tahansa saman mikroprosessoriarkkitehtuurin jakavalla alustalla /3/. Docker-containerin sisälle paketoitavista resursseista muodostetaan sovelluskuva Dockerfilen avulla. Kuvan voi julkaista repositoryihin, joista muut käyttäjät voivat ladata, ja näin ollen hyödyntää valmista kuvaa. Dockerfilestä muodostetaan sovelluskuva *docker build* -komennolla, ja kuvasta luodaan container *docker run* -komennolla. /14/

Toisin kuin virtuaalikoneet, containerit eivät virtualisoi fyysistä laitteistoa eivätkä sisällä kokonaista käyttöjärjestelmää (**Kuva 1.**). Ne ainoastaan muodostavat so-

vellustason abstraktion, joka yhdistää halutun sovelluksen sekä tarvittavat riippuvuudet, mikä tekee containereista virtuaalikoneita pienempiä sekä tehokkaampia. Yhdellä isäntäkoneella voi olla samanaikaisesti käynnissä useita containereita, jotka käyttävät kyseisen koneen käyttöjärjestelmän kerneliä. Containerit isoloivat sisältämänsä sovellukset sekä prosessit käytettävästä infrastruktuurista sekä muista containereista, mikä kohdistaa sovellusten aiheuttamat ongelmat containeriin eikä koko käyttöjärjestelmään. /5/



Kuva 1. Docker-container /5/

2.1.2 Kubernetes

Kubernetes on Googlen aloittama avoimen lähdekoodin projekti, josta julkaistiin ensimmäinen versio heinäkuussa 2015. Docker ei tarjonnut OpenShiftin kehitysvaiheessa tarvittavaa tukea monikerroksisten sovellusten tarpeisiin, joten sen skedulointi- ja orkestrointijärjestelmäksi valittiin Kubernetes. /3/ Kubernetesin tehtävänä on koordinoida klusteria, joka koostuu keskenään toimivista sekä kommunikoivista koneista /7/. Tuohon koordinointiin kuuluu containereihin liittyvät automaattiset hallinnointitoiminnot, muun muassa käyttöönotto sekä skaalaus /15/.

Klusteri

Kubernetesin arkkitehtuurin keskeisimmät resurssit ovat *master* ja *node*. Master hallinnoi klusteria, ja itse sovelluksia suoritetaan nodeissa. Node voi olla joko virtuaali- tai fyysinen kone, joka kommunikoi masterin kanssa. Kun sovellus halutaan ottaa käyttöön klusterissa, masterille annetaan käsky käynnistää containerit, mitä kutsutaan *deploymentiksi*. /7/

Deployment

Deploymentin tehtävänä on luoda ja päivittää sovellusinstansseja. Se on eräänlainen kokoonpano, jonka luontivaiheessa määritellään muun muassa, mitä sovelluskuvaa halutaan käyttää ja montako kopiota siitä halutaan käynnistää. /8/ Tämän jälkeen Kubernetes luo *podin*, johon container asetetaan /9/. Kun sovelluksen käyttöaste kasvaa, voi deploymentia skaalata ylöspäin suorituksen aikana, jolloin podien määrää kasvatetaan klusterin sisällä ilman katkoja /11/. Myös sovelluksen päivittäminen onnistuu Kubernetesin avulla ilman katkoja; päivitettävän sovelluksen sisältävästä podista luodaan rinnalle uusi pod, johon päivitettävä container asetetaan. Kun päivitetty pod on luotu, vanha poistetaan automaattisesti /12/.

Pod

Podilla tarkoitetaan klusterin sisällä suoritettavaa prosessia, joka enkapsuloi yhden tai useamman containerin sekä resurssit tiedon tallentamiseen. Kubernetes ei siis hallinnoi suoraan containereita, vaan niitä ympäröiviä podeja. Podit saavat luontivaiheessa uniikit IP-osoitteet klusterin sisällä riippumatta siitä, ovatko ne saman noden sisällä vai eivät. /9, 13/ *Replication Controllerilla (RC)* varmistetaan, että tietystä podista suoritetaan jatkuvasti useaa kopiota. Replication Controller poistaa ja lisää podeja automaattisesti tarpeen mukaan. /18/

Palvelu (service)

Koska jokaisella podilla on uniikki IP-osoite, loogisen joukon muodostaminen keskenään toimivista podeista on välttämätöntä. Esimerkiksi monikerroksisessa ohjelmistossa käyttöliittymäsovelluksen on päästävä käsiksi palvelinsovellukseen

podien kopioiden määrästä tai niiden sammumisesta huolimatta. Tätä loogista podien joukkoa kutsutaan *palveluksi* (service). Ilman palvelua sovellusta ei voi myöskään julkaista klusterin ulkopuolelle, sillä podien IP-osoitteet ovat ainoastaan klusterin sisäisiä. /10/

Tiedon tallennus

Monessa tapauksessa podien sisällä suoritettavien containereiden välillä on tarve jakaa samoja tiedostoja. Lisäksi containereiden sisältämät tiedostot häviävät aina, kun ne sammutetaan. Kubernetes ratkaisee edellä mainitut ongelmat omalla *volume*-abstraktiollaan. /19/

Volume on yksinkertaisesti dataa sisältävä hakemisto, johon containerit pääsevät käsiksi. Volumella on sama elinkaari kuin podilla; se ei siis ole riippuvainen containereista. Näin ollen containerin kaatuessa tiedostot pysyvät tallessa, ja ne häviävät vasta, kun pod lakkaa olemasta. /19/

Persistent Volume (PV) on tarkoitettu pysyvämpään tiedon tallentamiseen. PV:t eivät ole riippuvaisia podeista, vaan ovat klusterin resursseja siinä missä esimerkiksi nodet. Käyttäjät pyytävät PV:n resursseja käyttöönsä *Persistent Volume Claimillä (PVC)*, jossa on tieto halutusta tallennustilan määrästä ja oikeuksista tallennustilan käyttöön. Luku- ja kirjoitus -tyyppinen PV-resurssi voidaan kiinnittää vain kerran, ja vain luku -tyyppinen useaan kertaan. /20/

PV:t voidaan provisioida joko staattisesti tai dynaamisesti. Staattinen provisiointi tarkoittaa sitä, että klusterin ylläpitäjä määrittelee PV:t, jotka sisältävät tiedon siitä, kuinka paljon klusterin käyttäjille on tarjolla tallennustilaa. Jos yksikään ylläpitäjän määrittelemistä staattisesta PV:sta ei vastaa käyttäjän PVC:iä, klusteri voi yrittää provisioida PV:n dynaamisesti kyseisen PVC:n tarpeita vastaavaksi. /20/

Nimiavaruus (namespace)

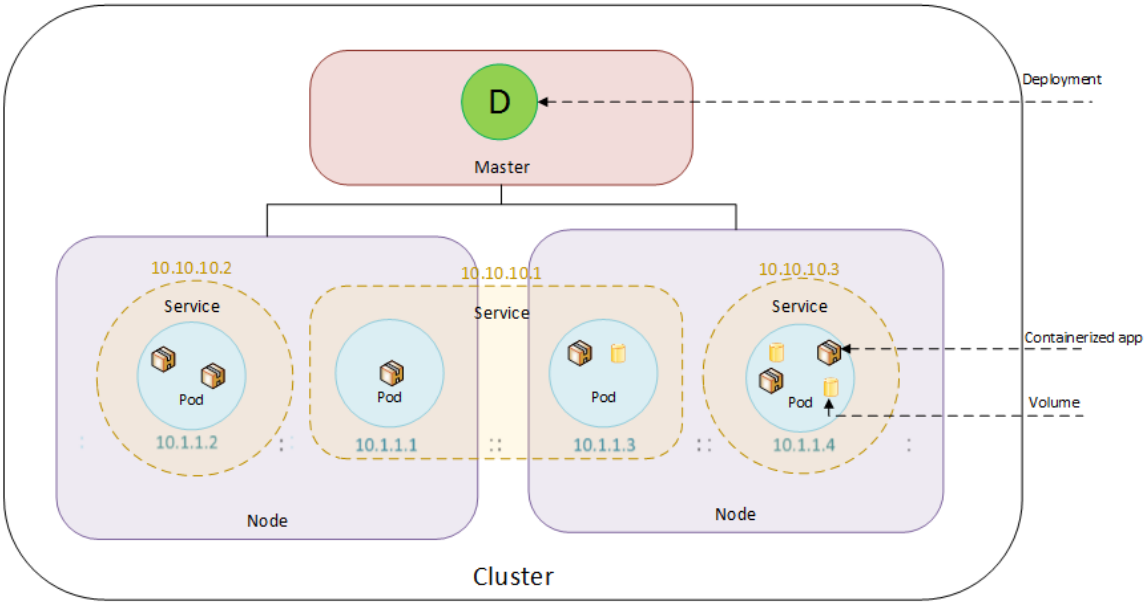
Kubernetesissa nimiavaruuksilla jaetaan käyttäjien luomat resurssit (esim. deploymentit, podit, palvelut) loogisesti nimettyihin ryhmiin. Jos klusteria käyttää useampi käyttäjä ja/tai käyttäjäryhmä, nimiavaruudet ovat käytännössä välttämättö-

miä. Klusterin nimiavaruuksien avulla eri käyttäjäryhmät voivat toimia klusterissa itsenäisesti välittämättä siitä, mitä muut käyttäjäryhmät tekevät.

Nimiavaruudet tarjoavat uniikin näkyvyysalueen:

1. Nimetyille resursseille (resurssin nimi on uniikki nimiavaruudessa, mutta ei klusterissa).
2. Hallintaoikeuksien jakamiselle (klusterin ylläpitäjä voi jakaa hallintaoikeuksia käyttäjille nimiavaruuskohtaisesti)
3. Resurssienhallinnalle (klusterin ylläpitäjä voi rajoittaa resurssien käyttöä nimiavaruuskohtaisesti). /16/

Edellä mainittujen käsitteiden lisäksi Kubernetes sisältää paljon muitakin toimintansa kannalta keskeisiä käsitteitä, jotka eivät kuitenkaan ole tämän opinnäytetyön kannalta oleellisia. Kuvassa 2 on esitetty Kubernetesin toimintaperiaate yksinkertaistettuna.



Kuva 2. Kubernetesin toimintaperiaate yksinkertaistettuna

2.1.3 OpenShift-kerros

Dockerin ja Kubernetesin päälle rakennettu OpenShift-kerros tarjoaa ylläpitäjille helppokäyttöisen työkalun sovellusten käyttöönottoon sekä kehitys- että tuotanto-ympäristöihin, ja kehittäjille alustan sovellusten luomiselle. Alustan tarkoituksena on pitää ohjelmistoprojektin aikana pääfokus itse sovelluksessa helpottamalla ylläpidollisia seikkoja. OpenShiftiä voidaan käyttää web-käyttöliittymän tai komentorivityökalun avulla, joista molemmat kommunikoivat OpenShiftin tarjoaman REST APIn kautta ja tarjoavat samat ominaisuudet. /3, 5/

Reitti (route)

OpenShiftin *reitti* asettaa palvelun näkyväksi ulkoverkkoon, esimerkiksi verkko-osoitteella *www.example.com*. Reitit voidaan asettaa myös TLS-suojatuiksi. /21/

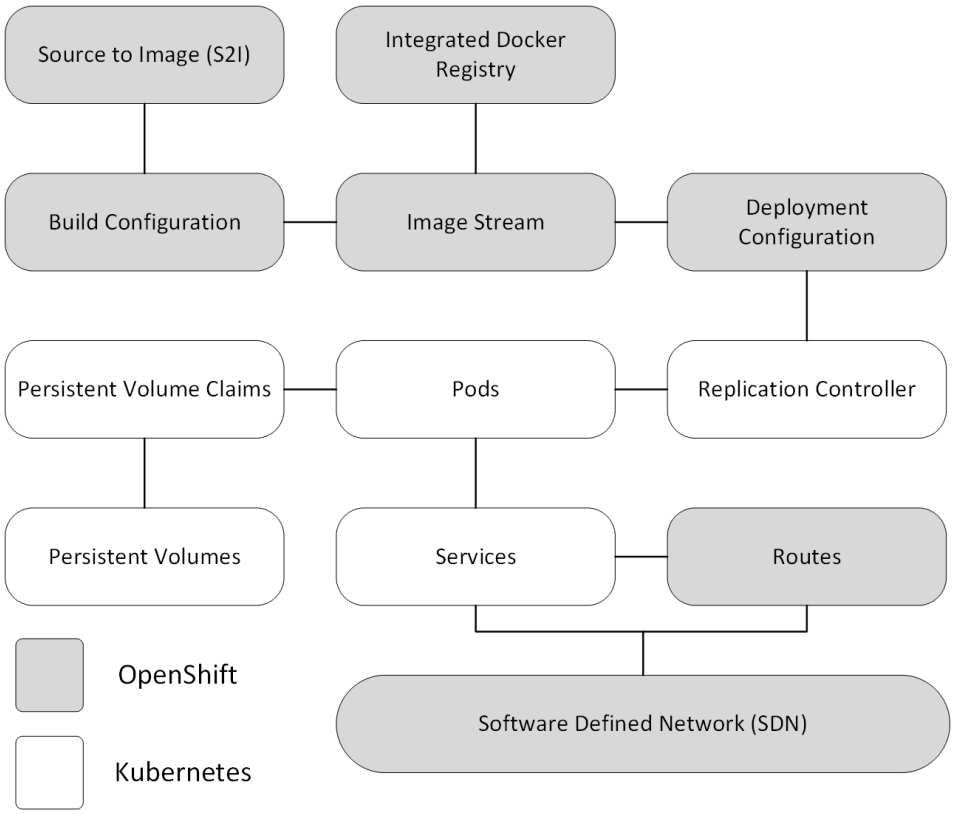
Deployment Configuration & Replication Controller

OpenShiftissä *Deployment Configurationillä (DC)* luodaan *Replication Controller (RC)*, joka käynnistää halutun määrän podeja. DC:ssä määritellään:

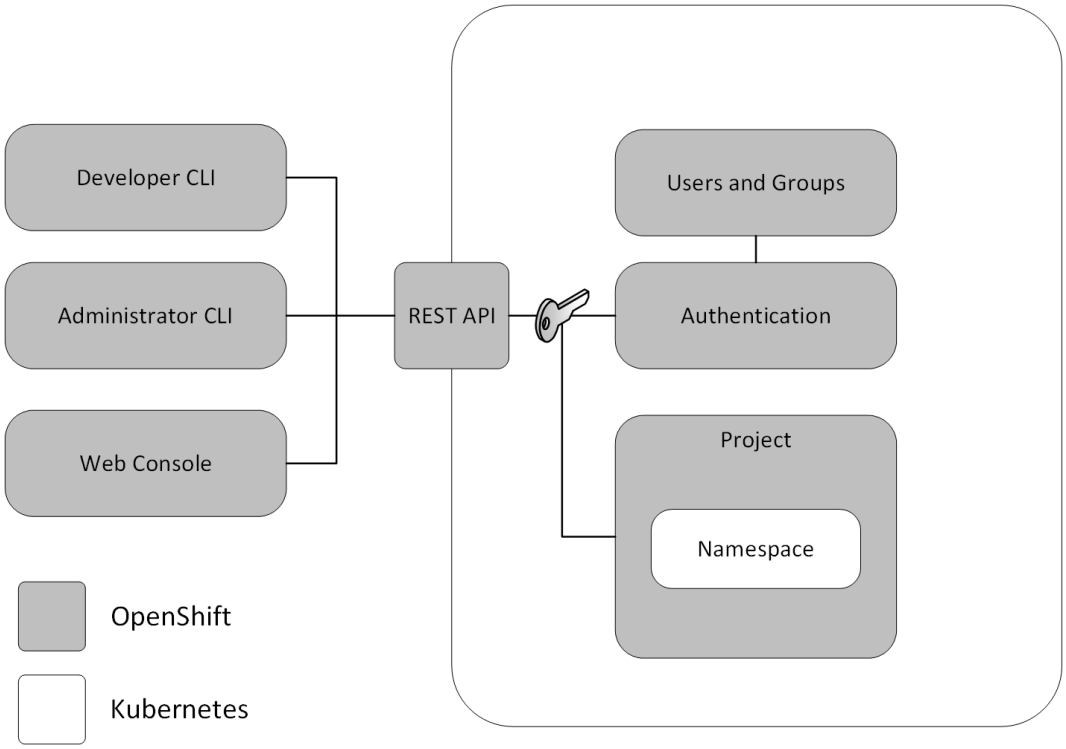
1. Replication Controllerin asetukset; esimerkiksi kuinka monta kopiota podista halutaan käynnistää.
2. Triggerit uuden deploymentin automaattiselle luonnille. Tämä voidaan määritellä esimerkiksi niin, että uusi deployment luodaan aina kun Replication Controllerin konfiguraatiota muutetaan.
3. Strategia deploymentien välisille siirtymille; esimerkiksi *Rolling*-strategia, joka mahdollistaa siirtymisen deploymentistä toiseen ilman katkoja.
4. Elinkaaritoiminnot (life cycle hooks), joiden avulla deployment-prosessin tiettyihin vaiheisiin voidaan asettaa erilaisia toimintoja, muun muassa ongelmien varalle. /22, 23/

Projekti (project)

Vaikka Kubernetesin nimiavaruuden avulla voidaan ryhmitellä resurssit ja näin ollen tarjota klusteriin tuki usealle käyttäjäryhmälle, klusterin käyttäjä (user) näkee silti jokaisen klusterissa olevan nimiavaruuden ja niissä olevat resurssit. Kubernetesin nimiavaruuksien välillä ei OpenShiftin kehittäjien mielestä ollut heidän käyttötarkoituksiinsa tarvittavaa tietoturvaa, minkä vuoksi OpenShiftiin luotiin laajennettu nimiavaruus, *projekti*. OpenShiftin kontekstissa projektin avulla hallitaan nimiavaruuksiin pääsyä *käyttäjien* ja *ryhmien* avulla (**Kuva 4.**). /3, 8/ OpenShift-projektin rakenne on mallinnettu kuvassa 3.



Kuva 3. OpenShift-projekti /mukaillen 3, s. 9/



Kuva 4. OpenShiftin pääsynhallinta /mukaillen 3, s. 8/

Pääsyoikeudet projektiin annetaan OpenShiftin ylläpitäjien toimesta. Jos käyttäjille annetaan oikeus luoda uusia projekteja, käyttäjät saavat automaattisesti pääsyoikeuden luomiinsa projekteihin. Jokaisella projektilla on *nimi* (name), *näyttönimi* (displayName) ja *kuvaus* (description). Nimi on uniikki tunniste, ja muut attribuutit ovat valinnaista, lähinnä web-käyttöliittymään lisäarvoa antavaa informaatiota. /17/ Taulukossa 1 on lueteltu muut projektin sisältämät attribuutit edellä mainittujen lisäksi.

Taulukko 1. OpenShift-projektin attribuutit /17/

Attribuutti	**Kuvaus**
Objektit (objects)	Podit, palvelut jne.
Käytännöt (policies)	Säännöt, jotka määrittelevät, mitkä käyttäjät voivat suorittaa operaatioita

	objekteille.
Rajoitteet (constraints)	Määrittelee, paljonko tietyntyyppisiä objekteja voi luoda projektiin ja paljonko tietokoneen resursseja projekti saa käyttää.
Palvelutilit (service accounts)	Automaattisesti toimivia käyttäjiä, joilla on määrätyt oikeudet projektin objekteihin.

Docker build -strategia

Docker build -strategia suorittaa OpenShiftissä yksinkertaisesti Dockerin *build*-komennon, minkä vuoksi sille on tarjottava Dockerfile sekä kaikki sen tarvitsemat riippuvuudet. Usein buildauksen tavoitteena on luoda lähdekoodista suoritettava sovelluskuva, jolloin sovellusta voidaan suorittaa OpenShiftissä. Tällöin OpenShiftin projektin build-konfiguraatiossa voidaan määritellä lähteeksi esimerkiksi Git-versionhallinnassa olevan repositoryn URL-osoite, jolloin OpenShift suorittaa seuraavat toiminnot:

1. Kloonaa repositoryn
2. Luo sovelluskuvan suorittamalla docker build -komennon repositorystä löytyvää Dockerfileä käyttäen
3. Tallentaa luodun kuvan OpenShiftin tarjoamaan Docker-rekisteriin, josta kuvaa voidaan hyödyntää uudelleen.

Mikäli esimerkiksi suoritettavaksi kuvaksi luodun sovelluksen lähdekoodi muuttuu, voidaan buildaus suorittaa uudelleen OpenShiftissä, jolloin kuva päivittyy Docker-rekisteriin. OpenShiftin komponentit voivat kuunnella kuvan muutoksia ja reagoida niihin määritetyillä tavoilla. Kyseistä kuvaa käyttävät deploymentit voivat esimerkiksi päivittää itsensä, kun kuva päivitetään uuteen versioon. Myös vanhempia kuvia voidaan käyttää ja niihin voidaan palata.

Docker build -strategian lisäksi OpenShift tukee oletuksena myös Source-to-Image-strategiaa, joka luo sovelluskuvan pelkän lähdekoodin perusteella. /43, 44/

2.2 Redmine

Redmine on Ruby on Rails -ohjelmistokehyksellä kehitetty avoimen lähdekoodin projektinhallintatyökalu, jonka ensimmäinen versio julkaistiin kesäkuussa 2006. Redmine sisältää useita tehokkaaseen projektinhallintaan tarvittavia ominaisuuksia, kuten aikataulutuksen, kalenterit, Gantt-kaaviot, etenemissuunnitelmat, versionhallinnan ja dokumenttienhallinnan. Redmineen on ydinominaisuuksien lisäksi tarjolla kolmannen osapuolen liitännäisiä, joilla voidaan saavuttaa projektinhallinnan kannalta tärkeitä lisäominaisuuksia. Redmine sisältää projektinhallintaominaisuuksien ohella työkalut tapahtumien seurantaan, ja ne ovatkin keskeinen osa ohjelmistoa. /25, 26/

2.2.1 Projekti (project)

Projekti on Redminessä hyvin keskeinen käsite. Redmineen voidaan luoda useita projekteja, jotka taas voivat sisältää useita aliprojekteja. Projektille otetaan luontivaiheessa käyttöön halutut moduulit, esimerkiksi Gantt-kaavio ja tapahtumien seuranta. Moduulit voivat olla joko kolmannen osapuolen tarjoamia tai vakiona mukana olevia. /25/

2.2.2 Pääsynhallinta ja roolit

Redmineen voidaan lisätä käyttäjätunnuksia suoraan, tai vaihtoehtoisesti autentikointitavaksi voidaan valita LDAP. Globaalisti ainoa peruskäyttäjästä eroava rooli on ylläpitäjä (admin). Ylläpitäjä pystyy muun muassa lisäämään, muokkaamaan ja poistamaan käyttäjiä, ryhmiä, rooleja, tapahtumatyyppejä ja tapahtumien tiloja.

Käyttäjät lisätään projektiin *jäseninä* (member). Jäsentä lisättäessä tälle määrätään projektikohtainen *rooli* (role). Roolit ovat ylläpitäjän hallinnoimia tietueita, joissa määritellään moduulikohtaisesti mitä jäsen voi projektissa tehdä. Moduulikohtaisten oikeuksien lisäksi rooliin määritellään itse projekteihin liittyvät oikeudet. Käyttäjän täytyy olla roolitettu jäsen projektissa saadakseen käyttöönsä roolissa

määritellyt oikeudet. Näin ollen projektiin liittyvät oikeudet saadaan käyttöön siten, että ylläpitäjä luo ensimmäiseksi ”yleisprojektin”, johon lisätään käyttäjä halutulla roolilla. Mikäli rooli sallii esimerkiksi uuden projektin luomisen, jäsen pystyy tämän jälkeen luomaan uusia projekteja järjestelmään. /25/

2.2.3 Tapahtumien seuranta (issue tracking)

Tapahtumien seuranta on yksi Redminen projektimoduuleista. Tapahtumassa (issue) voidaan kuvata esimerkiksi jokin tehtävä projektin jäsenten suoritettavaksi tai IT-projekteissa ohjelmointivirhe korjattavaksi. Tapahtuman voi asettaa myös jonkin olemassa olevan tapahtuman alitapahtumaksi.

Erityyppiset tapahtumat erotellaan *tapahtumatyypeillä* (tracker), joita ylläpitäjä voi luoda järjestelmään. Tyyppejä luotaessa niille määritellään attribuutit, jotka sentyyppisille tapahtumille otetaan käyttöön. Tietyt attribuutit ovat automaattisesti mukana kaikilla uusilla tapahtumatyypeillä. /25/

2.2.4 Integrointi kolmannen osapuolen sovelluksiin

Redminea käytetään yleensä web-käyttöliittymän kautta, mutta se tarjoaa myös REST APIn, joten se on mahdollista integroida kolmannen osapuolen sovelluksiin. REST API täytyy ottaa erikseen käyttöön ylläpitäjän toimesta. /24/

Käyttäjillä on käytössään API-avain, joka täytyy asettaa HTTP-pyyntöihin parametriksi tai otsikoksi. Käyttäjät voivat hakea itselleen API-avaimen Redminen web-käyttöliittymän kautta omalta profiilisivultaan. Redminen REST-rajapinta palauttaa tiedon joko XML- tai JSON-formaatissa. Haluttu palautusformaatti määritellään pyynnössä REST-resurssin URL-osoitteen jälkeen pisteellä eroteltuna. /27/

Kuvassa 5 REST-rajapinta on palauttanut JSON-muodossa projektin 1 tapahtumat. Pyyntö on tehty resurssiin /issues.json, ja URL-parametriksi on annettu project_id=1.

Kuva 5. Tapahtumien haku REST-rajapinnan kautta

2.3 LDAP

Hakemistolla tarkoitetaan eri tiedoista koostuvaa kokoelmaa. *Hakemistopalvelut* tarjoavat pääsyn hakemistossa oleviin tietoihin. *Hakemistopalvelinohjelmistot* toimivat ensisijaisesti hakemistopalveluina ja tarjoilevat hakemistossa olevaa tietoa sovelluksille ja loppukäyttäjille. /28/ LDAP-protokollaa tukeviin hakemistopalvelinohjelmistoihin viitataan tekstissä jatkossa termillä *LDAP-palvelin*.

LDAP on TCP/IP-pohjainen asiakas-palvelin-protokolla, joka tarjoaa verkon yli pääsyn hakemistoihin. Se siis määrittelee tavan, jolla hakemiston dataa voidaan lukea, lisätä, päivittää ja poistaa. LDAP ei ota itsessään kantaa datan sisältöön, mutta esimerkki voisi olla jonkin yrityksen työntekijöiden tiedot. /28/

LDAP perustuu alun perin X.500 Directory Access Protocoliin (DAP). X.500 on ITU-T:n luoma laaja standardijoukko, jossa määritellään perinpohjaisesti yleismaailmallinen hakemistopalvelu. X.500-standardijoukon laajuuden ja monimutkaisuuden vuoksi se on vähemmän käytetty. /28/

LDAP-palvelimia on olemassa erityyppisiä. Data voi olla tallennettuna paikallisesti samalla palvelimella, tai vaihtoehtoisesti se voi sijaita missä tahansa muualla, jolloin LDAP-palvelin toimii yhdyskäytäväpalvelimena ja hakee tiedon käyttäen jotain muuta verkkoprotokollaa. LDAPin ydinidea on, että datan sijainnilla ei ole merkitystä. /28/

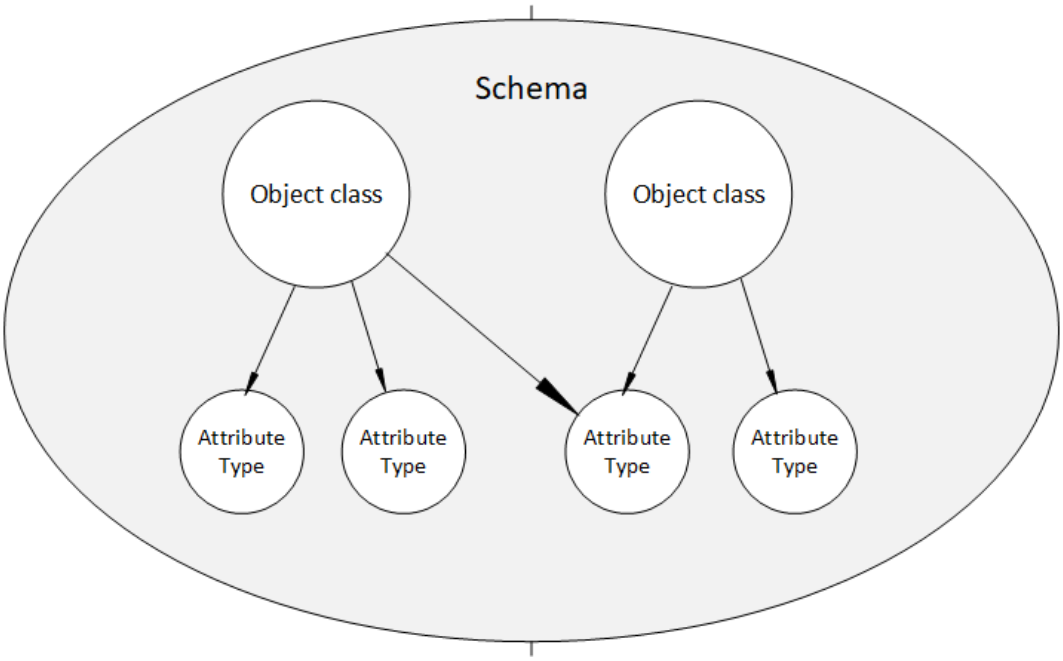
Internetin yli kulkevan LDAP-liikenteen turvaamiseksi voidaan käyttää LDAPS:iä, joka paketoi liikenteen SSL-sessioon /28/. LDAP käyttää oletuksena TCP- ja UDP-porttia 389, ja LDAPS porttia 636 /29/.

2.3.1 Tietomalli

LDAPin tietomalli ei ole relationaalinen eikä täysin olioperustainenkaan. LDAPissa informaatio kuvataan *merkintöinä* (entry). Merkinnät kuuluvat yhteen tai useampaan *objektiluokkaan* (object class). Objektiluokat on määritelty attribuuteilla, jotka koostuvat tyypistä ja yhdestä tai useammasta arvosta. /28/ Objektiluokkaa voidaan siis jollain tapaa verrata olio-ohjelmoinnin luokka-käsitteeseen,

koska se määrittelee, mitä attribuutteja tiettyyn objektiluokkaan kuuluva merkintä voi sisältää.

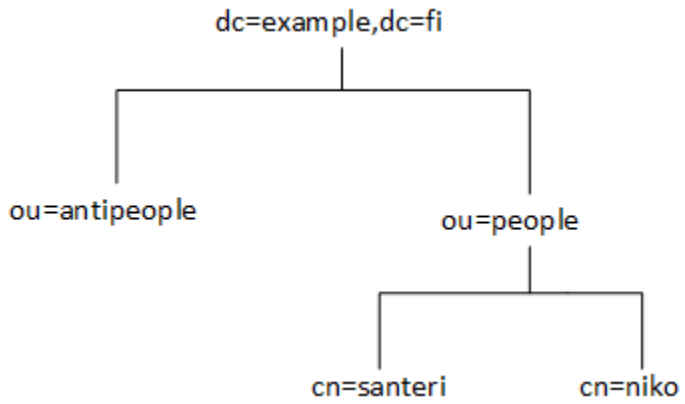
LDAP-palvelimen tukema *skeema* määrittelee, minkä tyyppistä tietoa tiettyyn hakemistoon voidaan lisätä. LDAP-skeema muodostuu objektiluokkien ja attribuuttien tyyppien määrittelyistä (**Kuva 6.**). /28/



Kuva 6. LDAP-skeema /mukaillen 28, s. 37/

2.3.2 Merkintöjen nimeämiskäytännöt

Merkintöjen nimet ovat LDAPissa hierarkkisia, eli ne on järjestetty puumaiseen rakenteeseen (**Kuva 7.**). Merkinnät on nimetty suhteessa niiden paikkaan puurakenteessa. Nimen täytyy olla uniikki ainoastaan saman isäntä-merkinnän alla olevilla merkinnöillä. /28/



Kuva 7. Hierarkkinen nimiavaruus /mukaillen 28, s. 63/

LDAPissa merkinnät identifioidaan DN-attribuutilla (distinguished name). DN muodostuu RDN:stä (relative distinguished name) ja pohjasta (base). RDN on yksi tai useampi merkinnän attribuutti. /28/ Esimerkiksi kuvassa 7 *cn=niko* voisi olla RDN. Tässä tapauksessa *cn=niko* olisi uniikki tieto *ou=people*n alla, mutta ei muualla hierarkiassa.

Pohja muodostetaan kulkemalla puumaista rakennetta merkinnästä ylöspäin ja liittämällä merkintöjen RDN:t yhteen pilkulla erotettuna /28/. Pohja olisi tässä tapauksessa *ou=people,dc=example,dc=fi*. Näin ollen DN saadaan yhdistämällä pohja ja RDN yhteen: *cn=niko,ou=people,dc=example,dc=fi*, joka on uniikki koko LDAP-hakemistossa. DN näyttää merkinnässä olevan attribuutti muiden joukossa, vaikkei se sitä varsinaisesti olekaan (**Kuva 8.**). /28/

```
dn: cn=niko,ou=people,dc=example,dc=fi
objectclass: person
cn: niko
givenname: Niko
sn: Lehto
```

Kuva 8. LDAP-merkintä

Edellisessä esimerkissä RDN:ksi valittiin etunimi, mikä on huono tapa, koska etunimet eivät ole uniikkeja ryhmitellyissä ihmisjoukoissa. RDN:ksi kannattaisi pikemminkin valita jokin generoitu arvo, jolla ei ole konkreettista tarkoitusta /28/.

2.3.3 Tiedon hakeminen hakemistosta

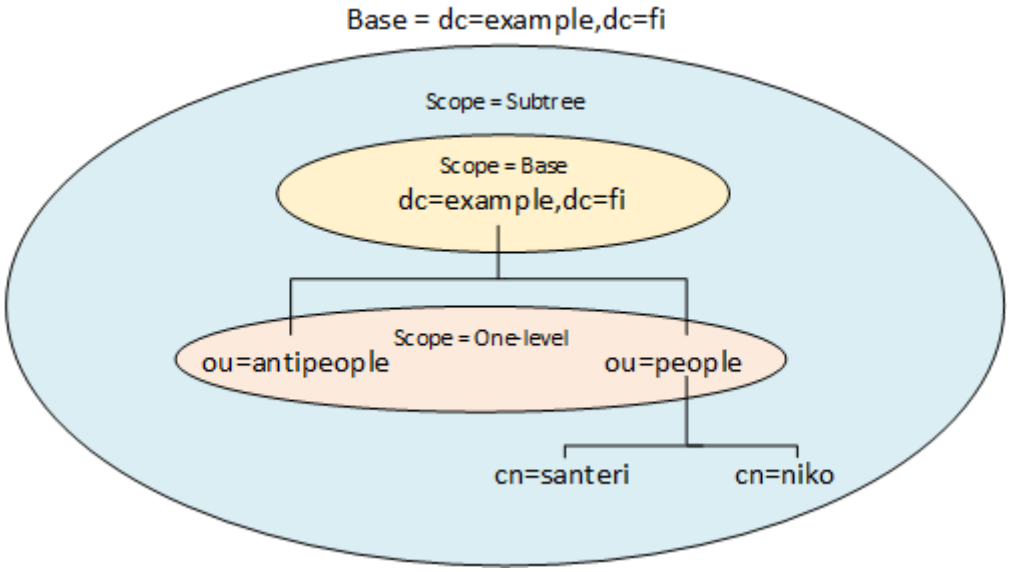
LDAP-asiakas hakee tietoa LDAP-palvelimelta tietyin hakukriteerein. Hakukriteereissä määritellään, mistä osasta hakemistopuuta tietoa haetaan, mitä ehtoja palautettavien merkintöjen pitää täyttää ja mitä tietoja täsmäävistä merkinnöistä palautetaan. Kun haku suoritetaan, LDAP-palvelin tarkistaa, onko palvelimeen yhdistäneellä LDAP-asiakkaalla oikeus suorittaa hakuja, ja palauttaa kaikki hakutulokset, joihin kyseisellä asiakkaalla on oikeus. /28/

Kun haussa määritellään hakemistopuun osa, josta tietoa halutaan, tekijät ovat pohja ja *ulottuvuus* (scope). Pohjaksi annetaan puussa ylimmän halutun merkinnän DN-attribuutti. Ulottuvuudeksi määritellään, miten syvältä pohjan alapuolelta haetaan muihin kriteereihin täsmääviä ehtoja. Ulottuvuuksien vaihtoehdot ovat:

- *Base*, joka hakee ainoastaan annettuun pohjaan täsmäävän merkinnän, eikä ainuttakaan alimerkintää. Pohja on siis merkintä siinä missä muutkin merkinnät.
- *One-level,* joka hakee ensimmäisenä täsmäävän pohjan alapuolella olevat merkinnät, muttei niiden alimerkintöjä.
- *Subtree,* joka hakee annettuun pohjaan täsmäävän merkinnän sekä rekursiivisesti kaikki sen alapuolella ovat merkinnät. /28/

Ulottuvuuksien eri vaihtoehdot on havainnollistettu myös kuvassa 9.

On huomioitava, että ulottuvuudet kertovat ainoastaan laajuuden, jolta merkintöjä haetaan. Muiden hakukriteerien on täsmättävä tulosten palauttamiseksi ulottuvuudesta huolimatta. Muilla hakukriteereillä voidaan vaatia, että jokin tietty attribuutti löytyy tai että se on tietyn arvoinen. Attribuutin arvoa voidaan hakea tavanomaisilla merkkijonovertailuilla, joita ei käydä sen tarkemmin läpi. Hakukriteerejä voi myös ketjuttaa, eli haussa voidaan ottaa kantaa useampaan kuin yhteen attribuuttiin. /28/



Kuva 9. LDAP-haun ulottuvuudet

2.3.4 Autentikointi ja autorisointi

Autentikointi tarkoittaa käyttäjän identiteetin validoimista jollakin todisteella, kuten salasanalla. *Autorisoinnilla* tarkistetaan, saako autentikoitunut entiteetti suorittaa tiettyä toimintoa tietyissä olosuhteissa. Esimerkiksi pankkiautomaatilla käteisnostoja tehtäessä käyttäjä autentikoituu PIN-koodia käyttämällä ja on autorisoitu käyttämään ainoastaan omaa pankkitiliään. /28/

LDAP ei ole autentikointipalvelu, mutta sitä voidaan silti käyttää autentikointiin, koska LDAP-hakemistossa voidaan pitää tallessa käyttäjien identiteettejä ja niihin liittyviä pääsytietoja, kuten salasanoja. Vaikka LDAP-standardit eivät tue hakemistojen tietoihin liittyvää pääsynhallintaa, monet hakemistopalvelinohjelmistot sisältävät ominaisuuksia pääsynhallintasääntöjen luomiselle, joten autorisointi on usein mahdollista LDAP-palvelimen avulla. Autorisointi voidaan myös suorittaa LDAP-palvelimen ulkopuolella käyttäen hakemistosta saatavaa informaatiota. /28/

LDAP-asiakkaat autentikoidaan hakemistopalvelimelle bind-operaatiolla. Bind-pyynnössä lähetetään asiakkaan DN, pääsytiedot (esim. salasana) sekä LDAP-versio, jota asiakas haluaa käyttää. /30/

Bind-operaatio tarjoaa mahdollisuuden käyttää yksinkertaista tai SASL-autentikointia. Yksinkertaisessa autentikoinnissa käyttäjä tunnistetaan LDAP-merkinnän DN:llä sekä salasanalla. Tässä vaihtoehdossa salasanat liikkuvat verkon yli sellaisenaan, joten suojatun yhteyden, kuten LDAPS:n, käyttäminen on erittäin suositeltavaa. /30/

SASL tarjoaa abstraktiotason verkkoprotokollien ja autentikointimekanismien väliin, eli sen avulla LDAPiin sekä muihin SASL:iä tukeviin protokolliin voidaan kiinnittää lähes mikä tahansa autentikointimekanismi. SASL-autentikointia käytettäessä LDAP-pääsytietojen lisäksi bind-pyyntöön määritellään SASL-autentikointimekanismi. /30, 31, 32/

Kun bind-operaatio on suoritettu, LDAP-palvelin palauttaa vastauksen, jonka mukana saadaan statuskoodi, joka kertoo, onnistuiko autentikointi vai ei. Mikäli autentikointi ei onnistu, vastauksessa oleva viesti kertoo tarkemmin, mikä oli epäonnistumisen syynä. Jos autentikointi onnistuu, palvelin palauttaa vastauksessa statuskoodin ja viestin lisäksi DN:n, jota bind-pyyntö vastasi. /30/

3 MÄÄRITTELY JA SUUNNITTELU

Tämä projekti on tyypiltään kehitys- ja tutkimusprojekti. Alussa tiedettiin suhteellisen hyvin, mitä ohjelmistolta halutaan, mutta teknisen toteutuksen osalta ei tiedetty juuri ollenkaan, kuinka projektia lähdetään toteuttamaan. Vaatimusmäärittely kirjoitettiin keskeisimpien toiminnallisten ja ei-toiminnallisten vaatimusten osalta, ja teknisen toteutuksen osalta päätettiin pintapuolisesti, millä tekniikoilla ja ohjelmistokehyksillä MSP toteutetaan.

Osa alkuperäisistä vaatimuksista on projektin aikana jätetty pois, ja toisaalta vaatimuksia on myös lisätty sitä mukaa, kun etenemissuunta on selkiytynyt. Tässä luvussa luetellut vaatimukset vastaavat kirjoitushetkellä voimassa olevaa vaatimusmäärittelyä.

3.1 Projektin yleiskuvaus ja tavoitteet

Esimerkki hosting-tarjoajan asiakkaasta on digitoimisto, joka tarjoaa asiakkailleen muun muassa verkkosivuja, verkkomarkkinointia sekä graafista suunnittelua. Digitoimistot hyödyntävät usein olemassa olevia sisällönhallintajärjestelmiä, kuten WordPressiä tai Drupalia, ja ulkoistavat hostingin muualle pystyen näin ollen keskittymään ydinliiketoimintaansa.

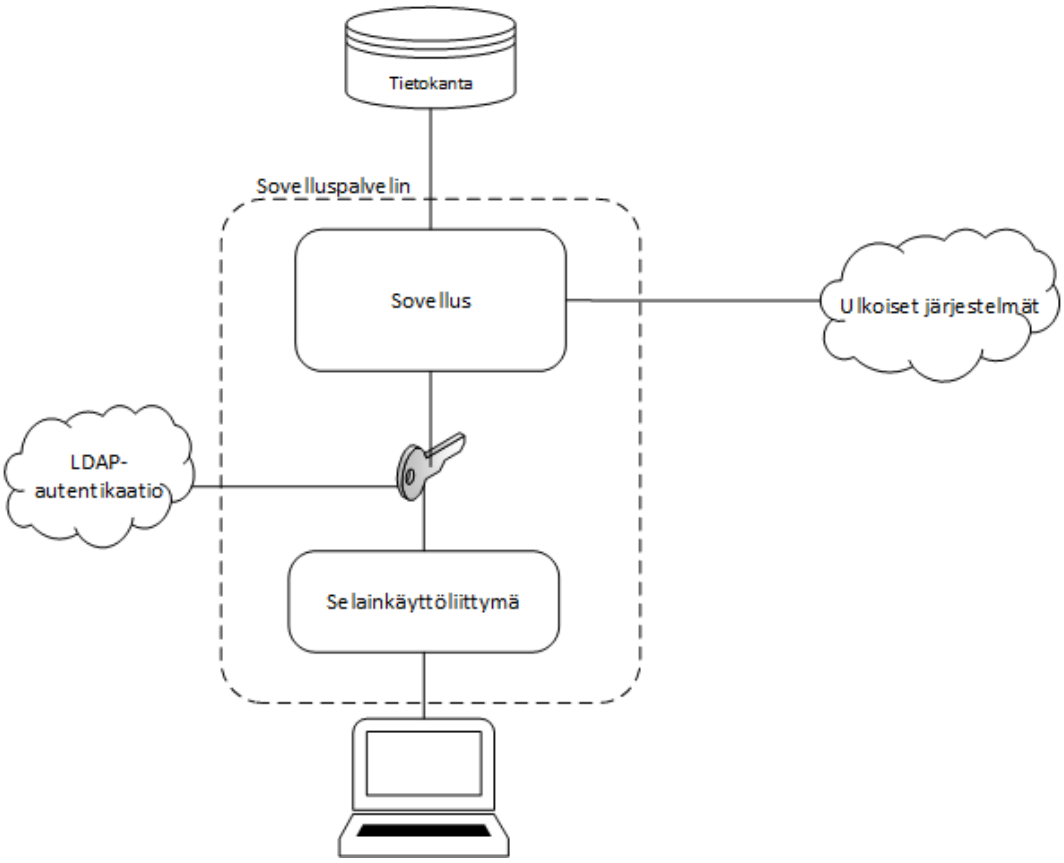
Hosting-asiakkaan täytyy usein päästä hallinnoimaan pelkän sovellusinstanssin lisäksi myös esimerkiksi verkkotunnuksia, SSL-sertifikaatteja ja DNS-tietueita, joita yksittäiseen palvelukokonaisuuteen kuuluu. Lisäksi asiakkaan on pystyttävä lähettämään hosting-tarjoajalle palvelupyyntöjä muun muassa ongelmatilanteiden sattuessa. Mikäli asiakas joutuu käsittelemään osaa tai jopa kaikkia edellä mainittuja asioita eri hallintapaneelien tai muiden kanavien kautta, asiakaskokemus ei ole paras mahdollinen. Tämä tilanne ei ole suotuisa myöskään hosting-tarjoajalle, joka joutuu mahdollisesti antamaan asiakkailleen käyttöoikeuden usealle eri alustalle sekä vastaanottamaan palvelupyyntöjä ja muita yhteydenottoja sähköpostitse tai puhelimitse.

Tämä projekti aloitettiin edellä kuvatun kaltaisen ongelman ratkaisemiseksi keskitetyn palvelualustan avulla. Lähtökohtana on alusta lähtien ollut, että asiakkaan tilaaman palvelun ylimmän tason komponentti on sovellus. Hosting-asiakkaalle suunnatussa palvelualustassa esimerkiksi yksi Wordpress-palvelu voisi sisältää sovellusinstanssin lisäksi muita komponentteja, joita siihen kohdeasiakkaan näkökulmasta usein kuuluu, kuten verkkotunnuksen sekä SSL-sertifikaatin. Asiakas pystyisi näin hallitsemaan kaikkia yksittäiseen palveluun liittyviä komponentteja samasta paikasta. Lisäksi hosting-tarjoaja pystyisi tarkkailemaan asiakkaiden tilaamia palveluja keskitetysti.

Projektin keskeisempänä tavoitteena on suunnitella ohjelmistoarkkitehtuurista niin skaalautuva, että se ei ota juurikaan kantaa siihen, mihin ulkopuolisiin järjestelmiin se integroidaan. Näin ollen esimerkiksi tietyntyylisten palvelukomponenttien hallinta voidaan integroida ensin johonkin tehtävienhallintaohjelmistoon ja myöhemmin automatisoida koko prosessi muuttamatta sovelluksen liiketoimintalogiikkaa. Sovelluksen ”Palvelu”-konsepti pyritään abstrahoimaan mahdollisimman pitkälle, jotta sitä voitaisiin tulevaisuudessa käyttää useissa eri implementaatioissa.

3.2 Järjestelmäkuvaus

Projektin suunnitteluvaiheessa tultiin lopputulokseen, että sovellus kehitetään Java-ohjelmointikielellä ja että ohjelmistokehyksenä käytetään Dropwizardia, joka sisältää valmiiksi Jetty-HTTP-palvelimen, REST-arkkitehtuurityylin mukaisten sovellusten kehittämiseen tarkoitetun Jersey-ohjelmistokehyksen sekä Jackson-ohjelmistokehyksen olioiden serialisointiin ja deserialisointiin. /41, 42/ Jetty tarjoilee myös staattisen sisällön. Koko järjestelmän kuvaus on esitelty kuvassa 10.



Kuva 10. Järjestelmäkuvaus

3.3 Käyttäjät

MSP:n käyttäjätasot ovat ylläpitäjä sekä asiakas. Asiakaskäyttäjällä on aina relaatio yhteen organisaatioon, jossa hänellä on tietty rooli. Rooli määrittelee, mitä toimintoja käyttäjä saa suorittaa sovelluksessa. Rooleja käydään tarkemmin läpi luvussa 3.5. Ylläpitäjä on yksittäinen toimija ilman relaatiota organisaatioon.

3.4 Toiminnalliset vaatimukset

MSP:n toiminnalliset vaatimukset on lueteltu taulukossa 2.

Taulukko 2. Toiminnalliset vaatimukset

Vaatimus	Kuvaus	Prioriteetti
V1	Käyttäjien tunnistautuminen	1

V2	Uuden palvelun luominen, muokkaaminen ja poistaminen	1
V3	Asiakkaille tarjottavien palvelumallien luominen	1
V4	Omien tietojen muokkaaminen	1
V5	Asiakkaiden hallinta	1

3.4.1 Käyttäjien tunnistautuminen

Palvelualustan minimaalinenkin käyttö vaatii sisäänkirjautumisen. Sisäänkirjautumispyynnön yhteydessä selviää käyttäjän rooli, jonka mukaan käyttöliittymässä voidaan näyttää roolikohtainen näkymä sekä eroavat toiminnallisuudet.

3.4.2 Uuden palvelun luominen

Asiakaskäyttäjä pystyy luomaan omalle organisaatiolleen palveluja, mikäli tämän rooli niin sallii. Palvelun luonnin yhteydessä annetaan palvelulle nimi sekä syötetään kyseisen palvelun vaatimat parametrit dynaamisen lomakkeen avulla. Palvelun luonnin yhteydessä osa siihen liittyvistä komponenteista voidaan provisioida ulkoisiin järjestelmiin, ja osa voi jäädä odottamaan asiakkaan aktivointia. Palvelumalli määrittelee, kuinka palvelu luodaan ja mitä komponentteja asiakkaalla on mahdollisuus aktivoida kyseiseen palveluun.

3.4.3 Asiakkaille tarjottavien palvelumallien luominen

Ylläpitäjä pystyy luomaan järjestelmään uusia palvelumalleja, joista käyttäjä voi luoda itselleen palveluja. Ylläpitäjä määrittelee, mitä käskyjä mihinkin ulkoiseen järjestelmään suoritetaan, kun palvelumallista luodaan palvelu. Samalla määritellään data, joka vaaditaan palvelukomponenttien provisiointiin. Datan muodolla ei ole merkitystä.

3.4.4 Omien tietojen muokkaaminen

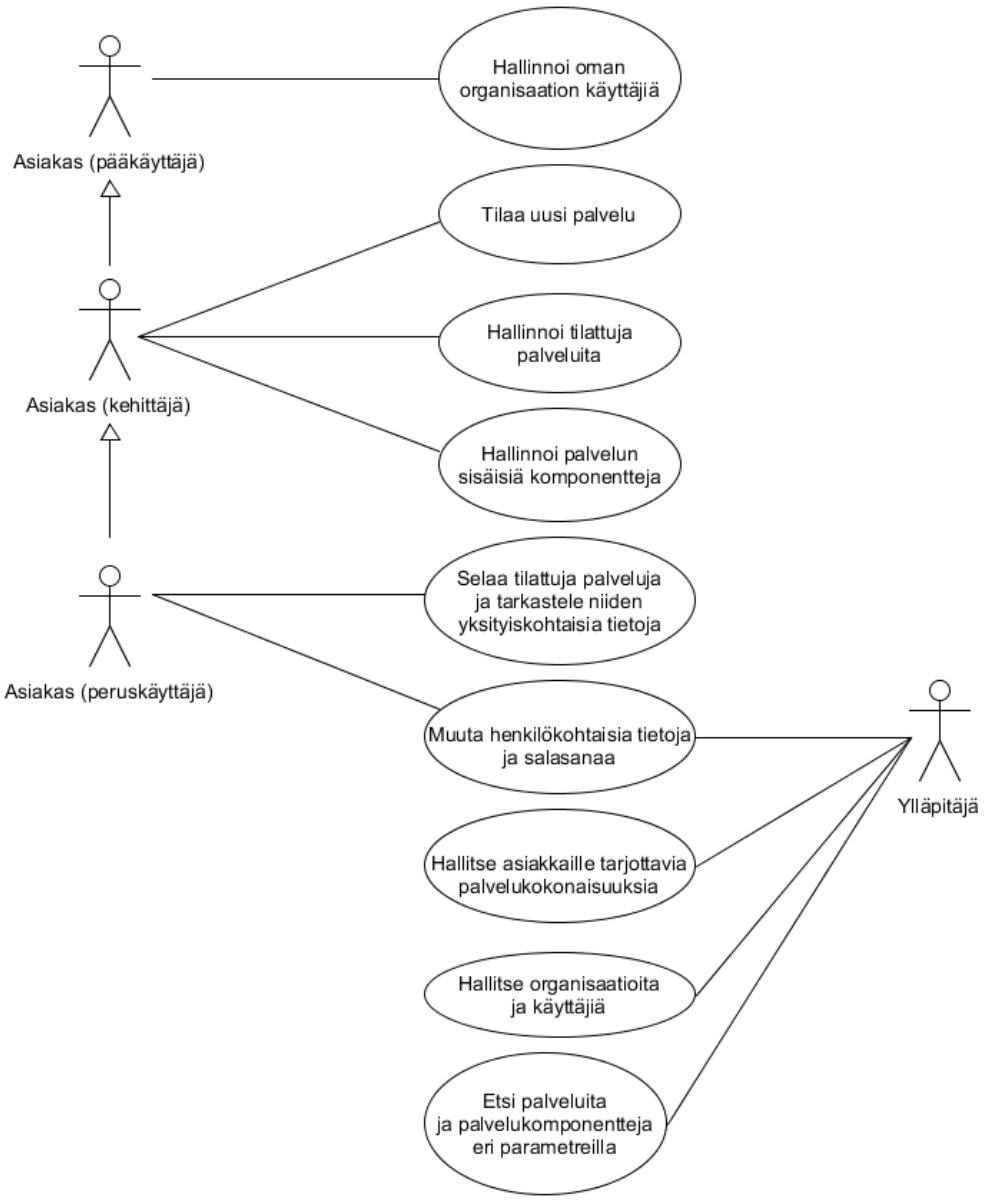
Sekä asiakaskäyttäjä että ylläpitäjä voivat muokata omia henkilökohtaisia perustietojaan, kuten sähköpostiaan sekä puhelinnumeroaan. Lisäksi oman salasanan vaihtaminen on mahdollista.

3.4.5 Asiakkaiden hallinta

Ylläpitäjä pystyy hallitsemaan palvelualustan käyttäjiä. Hallinnointiin kuuluu käyttöoikeuden antaminen ja poistaminen organisaatiokohtaisesti sekä organisaation käyttäjien roolien hallinta ja käyttöoikeuksien poistaminen.

3.5 Roolikohtaisten käyttötapausten erittely

Asiakaskäyttäjän rooli voi olla organisaation sisällä pääkäyttäjä, kehittäjä tai peruskäyttäjä. Pääkäyttäjärooli on tarkoitettu organisaation sisäisten käyttäjien sekä niiden roolien hallintaan. Kehittäjäroolissa olevat käyttäjät voivat tilata sekä tehdä muutoksia palveluihin, ja peruskäyttäjillä on ainoastaan lukuoikeudet. Ylemmän tason rooli perii alemman roolin oikeudet, eli esimerkiksi organisaation pääkäyttäjällä on myös palvelun tilaus- ja lukuoikeudet. Roolikohtaiset käyttötapaukset löytyvät kuvasta 11.



Kuva 11. Käyttötapauskaavio

3.6 Tietokannan suunnittelu

Tässä luvussa tarkastellaan sovelluksen käyttämän tietokannan keskeisimpiä tauluja. Koko tietokannan datamalli esitellään kuvassa 12.

3.6.1 Palvelu (service)

Service-taulu kuvaa käyttäjän luomaa palvelua. Palvelu sisältää tiedon käyttäjästä, joka sen on luonut ja joka on sitä viimeksi muokannut. Palvelulla on myös relaatio palvelumalliin (service template), josta se on luotu.

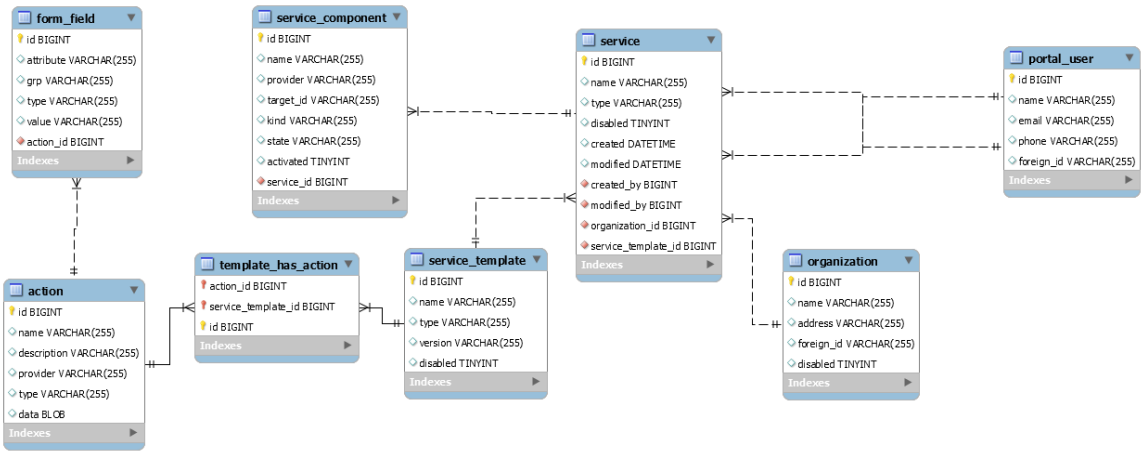
3.6.2 Palvelukomponentti (service component)

Palvelukomponentti kuvaa ulkoisessa järjestelmässä olevaa resurssia, ja se liittyy aina johonkin palveluun. Palvelukomponentti-tietueesta löytyy tieto, mistä ulkoisesta järjestelmästä se tarjoillaan ja mikä on sen uniikki tunniste kyseisessä järjestelmässä.

3.6.3 Käsky (action)

Käsky-taulu kuvaa toimintoa, joka suoritetaan ulkoiseen järjestelmään. Se sisältää valinnaisen data-sarakkeen, joka sisältää kyseisen käskyn suorittamiseen vaadittavan tiedon.

Käskyyn liittyy myös lomakekenttiä (form field). Lomakekenttien avulla muodostetaan dynaaminen lomake, jonka arvot asetetaan käskyn dataan. Ne ovat käytännössä parametreja, joille asiakaskäyttäjä asettaa itse arvot.



Kuva 12. Datamalli

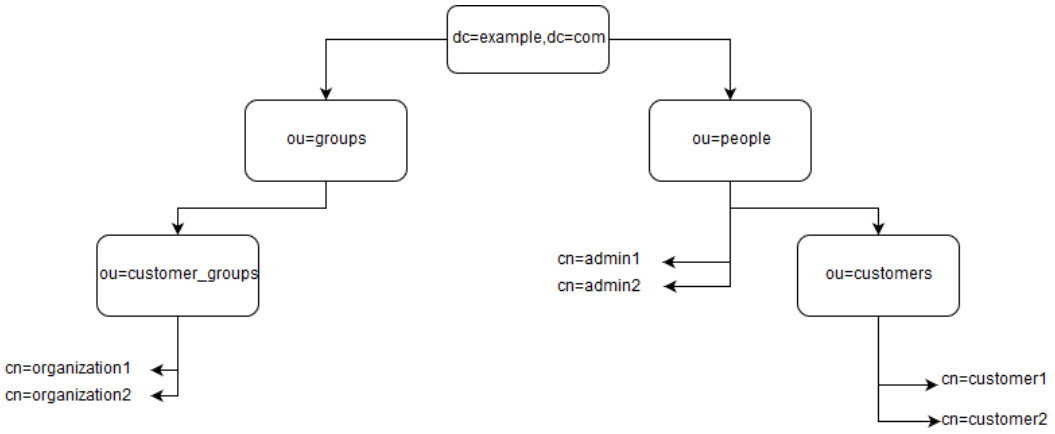
Käyttäjillä ei ole tietokannassa relaatiota organisaatioon, vaan relaatio sijaitsee ulkoisessa autentikointipalvelussa. Autentikoinnin yhteydessä käyttäjän relaatio organisaatioon tallennetaan käyttäjän istunnon ajaksi palvelinsovelluksen muistiin.

3.7 Autentikointi ja autorisointi

Autentikointi päätettiin toteuttaa olemassa olevaa LDAP-hakemistoa hyödyntäen. Vaatimuksena oli, että myös web-palvelimella oleva staattinen sisältö suojataan tunnistautumattomilta käyttäjiltä, sisäänkirjautumissivua lukuun ottamatta. Samoin kaikki REST APIn resurssit suojataan tunnistautumattomilta käyttäjiltä, lukuun ottamatta sisäänkirjautumisresurssia.

Koska ylläpitäjän suorittamat toiminnot sovelluksessa ovat melko kriittisiä ulkoisten järjestelmien kannalta, ylläpitäjät ja asiakaskäyttäjät eivät tunnistaudu samaan sovellusinstanssiin. Sovelluksen käynnistysvaiheessa määritellään ympäristömuuttujasta, käynnistetäänkö sovelluksesta ylläpitäjien vai asiakkaiden versio, ja sen perusteella aktivoidaan tietyt REST API -resurssit.

Projektin alkuvaiheessa määriteltiin LDAP-skeema, jota sovelluksen on tuettava (**Kuva 13.**). Uudelleenkäyttöä silmällä pitäen tuettavan skeeman täytyisi olla kuitenkin mahdollisimman pitkälle konfiguroitavissa.



Kuva 13. Tuettava LDAP-skeema

Koska kirjautumisvaiheessa tiedetään, onko sovelluksesta käynnissä ylläpitäjien vai asiakkaiden versio, DN on melko vaivatonta rakentaa bind-operaatioon kyseisellä LDAP-skeemalla, koska ylläpitäjät ja asiakaskäyttäjät ovat puurakenteessa eri tasolla.

Kirjautumisen onnistuessa käyttäjälle generoidaan valtuutusavain, joka tallennetaan palvelinsovelluksen muistiin tietyksi ajaksi. Valtuutusavain tallennetaan myös käyttäjän selaimeen, jotta käyttäjä voidaan validoida jokaisella HTTP-pyynnöllä.

Eri asiakaskäyttäjätasojen vuoksi tietyt REST API -resurssit sallitaan vain tietyille rooleille. Tämän vuoksi valtuutusavaimeen tallennetaan myös tieto käyttäjän roolista.

3.8 Sovelluksen suunnittelu

Sovelluksen suunnittelu aloitettiin relaatiokannan datamallin pohjalta. Tietokannan tauluja mallintavat luokat on erotettu serialisoitavista representaatioluokista, joita käytetään REST APIlla. On loogista, että entiteettiluokat mallintavat täysin tietokannan taulujen kenttiä sekä taulujen välisiä relaatioita, eikä niissä oteta kantaa kuinka ne serialisoidaan tai deserialisoidaan.

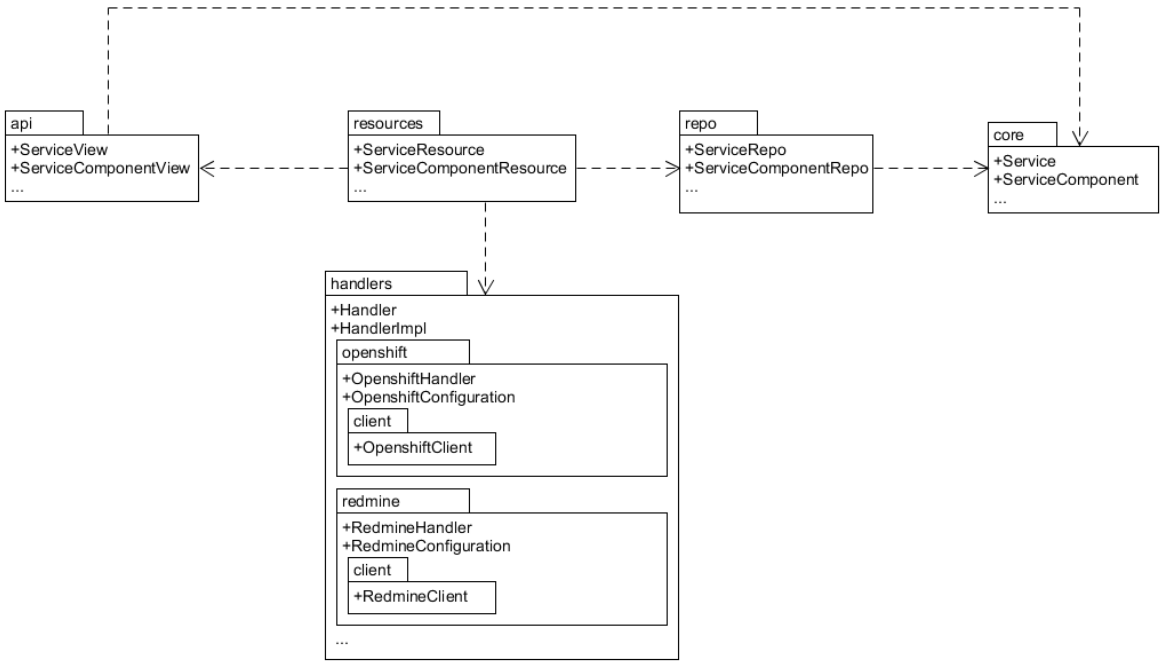
Jokaista representaatioluokkaa vastaa JAX-RS-annotoitu resurssiluokka. Resurssiluokat edustavat URI-polkuja, joihin saapuvat HTTP-pyynnöt ne ottavat vastaan

ja käsittelevät HTTP-metodin perusteella. Resurssiluokkien metodit palauttavat aina yksittäisiä representaatio-olioita tai kokoelmia niistä.

Tietokannassa olevan datan hakemiseen resurssiluokat käytävät *repo*-luokkia, joiden metodit palauttavat entiteettiolioita. Entiteettioliot muutetaan representaatio-olioiksi resurssiluokassa ennen HTTP-vastauksen palauttamista.

Ulkoisen järjestelmän kanssa kommunikointiin käytettäviä luokkia kutsutaan *käsittelijöiksi* (handler). Tietyt resurssiluokat ovat riippuvaisia käsittelijöistä.

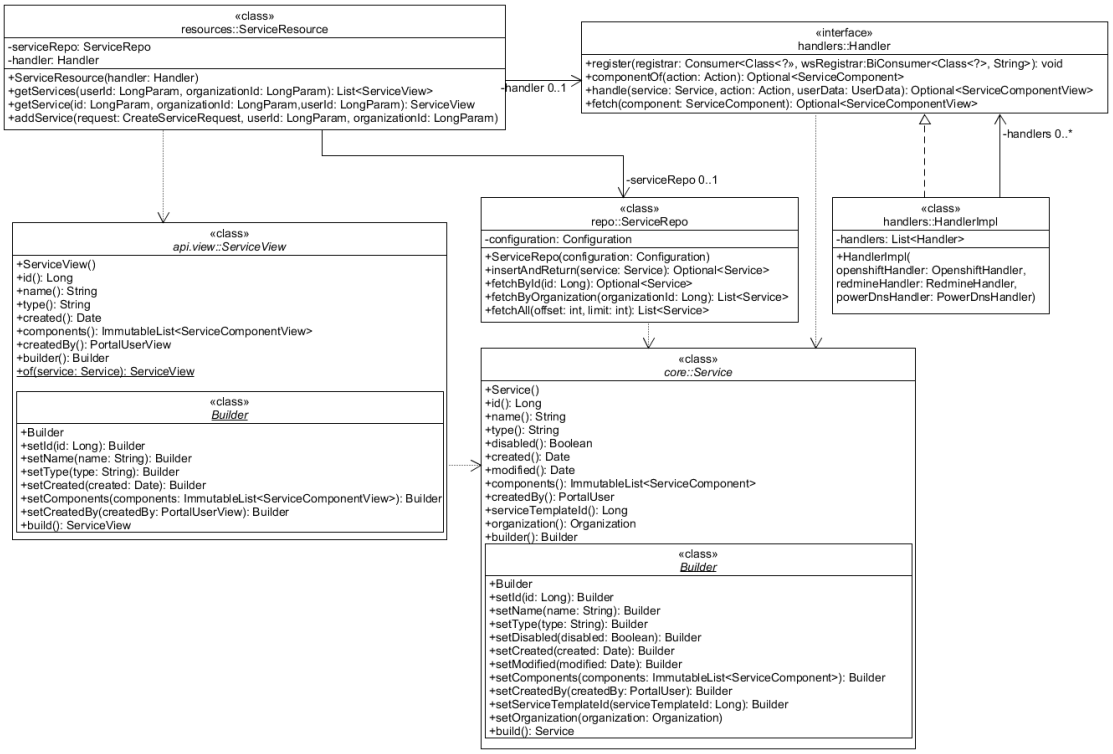
Kuvassa 14 on kuvattu sovelluksen rakenne sen ydintoiminnallisuuden osalta. Sovellus sisältää muitakin pakkauksia, jotka liittyvät muun muassa istuntojen ja LDAP-operaatioiden käsittelyyn.



Kuva 14. Pakkauskaavio

Resurssiluokat ovat täysin isoloituja taustalla olevien ulkoisten järjestelmien integraatiosta, sillä ne ovat riippuvaisia ainoastaan anonyymistä Handler-rajapinnan oliosta, joka voi itse asiassa olla mikä tahansa käsittelijä. Tässä implementaatiossa tuo käsittelijä on kuitenkin HandlerImpl-luokan instanssi, joka valitsee tietynlaiselle käskylle oikean käsittelijän. Näin ollen uusia käsittelijöitä implementoitaessa

resurssiluokkiin ei tarvitse tehdä muutoksia. Esimerkki käsittelijää käyttävästä resurssiluokasta on ServiceResource-luokka, joka on kuvattu riippuvuuksineen luokkakaaviolla kuvassa 15.



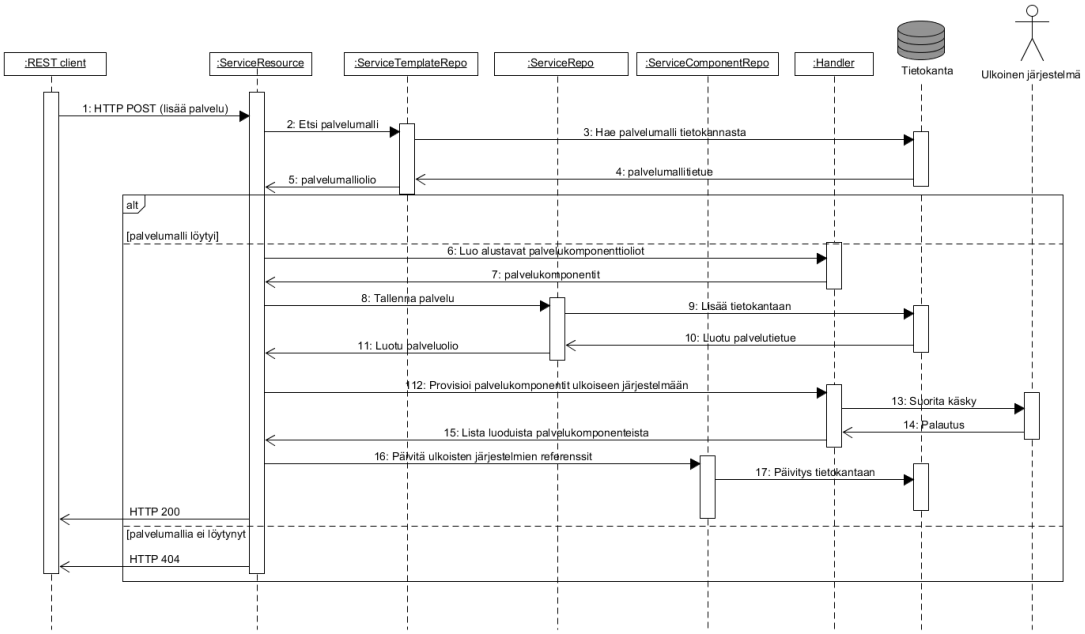
Kuva 15. ServiceResource-luokka ja sen riippuvuudet

Sovellusta käytetään selainsovelluksella, joka hyödyntää palvelinsovelluksen tarjoamaa REST APIa. Lähempään tarkasteluun otetaan palvelun luonti REST APIn kautta HTTP-POST-pyynnöllä. Pyynnössä olevan datan perusteella sovellus hakee tietokannasta valitun palvelumallin tiedot, jonka jälkeen siitä luodaan tarvittavat palvelukomponenttioliot. Tämän jälkeen palvelukomponentit tallennetaan alustavasti tietokantaan väliaikaisella tunnisteella, minkä jälkeen siirrytään vasta komponenttien provisiontiin ulkoisiin järjestelmiin.

Väliaikaisten tunnisteiden avulla komponentti voidaan ensin luoda tietokantaan, menettämättä yhteyttä sen varsinaiseen provisiontiin tarkoitettuun dataan. Komponenttien tallentaminen paikalliseen tietokantaan ennen provisiointia on tärkeää, sillä jos kaikkien komponenttien provisionti ei palvelun luontivaiheessa onnistu, tietokannassa olevan datan perusteella voidaan jatkossa tehdä tietyt toimenpiteet

niiden provisioimiseksi manuaalisesti. Näin ollen käyttäjälle voidaan näyttää, mahdollisista provisioinnin epäonnistumisista huolimatta, kaikki luodun palvelun komponentit ja ilmaista niiden tilat eri indikaattoreita käyttäen. Käyttäjä voisi jopa itse yrittää provisioida virheellistä komponenttia ulkoiseen järjestelmään käyttöliittymän kautta.

Kun komponentit on provisioitu onnistuneesti ulkoisiin järjestelmiin ja niistä on saatu tarvittava data käsittelijältä, väliaikaiset tunnisteet päivitetään tietokannassa varsinaisiin ulkoisten järjestelmien tunnisteisiin. Edellä kuvattu operaatioketju on havainnollistettu sekvenssikaaviolla kuvassa 16.



Kuva 16. Sekvenssikaavio palvelun luonnista

4 VASTAAVANLAISET RATKAISUT

Markkinoilla on useita hosting-tarjoajille tarkoitettuja palvelualustoja, joista tähän lukuun on valittu lähempään tarkasteluun muutama. Lopuksi niitä vertaillaan MSP:n käyttöönottovalmiiseen lopputulokseen sekä keskenään.

4.1 cPanel

Yhdysvaltalaisen cPanel-hallintapaneelin kehittäminen aloitettiin vuonna 1997. Sen avulla hosting-asiakkaat voivat hallita selainkäyttöliittymän avulla omaa webhotelliaan. /33, 34/

Asiakkaalla on cPanelissa yksi päädomain, joka osoittaa yhteen palvelimen kansioon. Asiakas pystyy päädomainin kontekstissa hallinnoimaan palvelujaan web-käyttöliittymän avulla. Palveluihin kuuluvat muun muassa tietokantojen lisäys ja hallinta, domain-nimien osoittaminen alikansioihin sekä sähköpostiosoitteiden hallinta domainkohtaisesti. /36, 37, 65/

cPanelin vakio-ominaisuuksia hyödyntäen muutamien PHP- ja Perl-sovellusten automaattinen käyttöönotto on mahdollista. Kolmannen osapuolen liitännäisten, kuten Softaculousin avulla saadaan automaattisesti asennettavien web-sovellusten joukkoon lisää vaihtoehtoja /35/. Hosting-tarjoajan vastuulla on määritellä, mitä kolmannen osapuolen liitännäisiä asiakkaat saavat käyttöönsä, ja mitä sovelluksia asiakas voi ottaa käyttöön automatisoidusti. /66/

4.2 Cloudways

Vuonna 2011 Maltalla perustettu Cloudways Ltd tarjoaa asiakkailleen alustan web-sovellusten käyttöönottoa ja hallintaa varten. Asiakas voi luoda alustan kautta itselleen virtuaalipalvelimia eri infrastruktuureihin, kuten Amazon Web Servicesiin tai Digital Oceaniin. Virtuaalipalvelimen luonnin yhteydessä palvelimelle valitaan sovellus asennettavaksi tarjotuista vaihtoehdoista sekä niiden eri versioista. Luonnin aikana palvelimelle valitaan myös haluttu fyysinen lokaatio sekä käyttöön haluttavat resurssit. Palvelimelle voidaan luonnin jälkeen lisätä myös uusia

sovelluksia. Sovelluksia voi poistaa palvelimelta tarpeen mukaan, mutta jokaisella palvelimella on oltava vähintään yksi sovellus. /38, 64/

Cloudwaysissä ylimmän tason komponentti on palvelin, jonka alle voidaan luoda sovelluksia. Sovelluksia voi myös ryhmitellä ilman palvelinkontekstia *projektien* alle. Käyttöliittymästä voidaan tarkastella ja hallita erikseen sekä palvelimia että sovelluksia. Palvelinhallinnasta voi esimerkiksi käynnistää uudelleen sovellusten kannalta keskeisiä palveluja, kun taas sovellushallinnasta voi hallita yksittäiseen sovellukseen liittyviä asetuksia. Sovellushallinnan ominaisuuksiin kuuluvat muun muassa SSL-sertifikaattien asennus sekä domain-hallinta. Lisäksi jotkin sovellushallinnan toiminnot vaihtelevat sovellustyypin mukaan. Cloudways-asiakkuuteen voi lisätä myös muita jäseniä, joille alkuperäinen asiakas voi valita lisäysvaiheessa tietyt roolit. /64/

4.3 Cloudcity

Suomalainen Cloud City Oy tarjoaa Cloudcity-alustan web-palveluiden hallintaan. Neljästä Cloudcity-tuoteryhmästä halvin on tarkoitettu pelkästään staattisen sisällön ylläpitämiseen, ja loput ovat PHP-tuellisia webhotelleja vaihtelevilla resursseilla. Asiakkaat pystyvät hallinnoimaan kaikkia palveluja sekä niihin liittyviä lisäominaisuuksia saman hallintapaneelin kautta. /39/

Cloudcityn ”Palvelu”-konseptissa ylimmän tason komponentti on web-sivustoon ohjaava päädomain, jonka alle voidaan aktivoida lisäominaisuuksia. Lisäominaisuuksia ovat muun muassa sähköpostilaatikoiden hallinta, tietokanta, verkkotunnus, Piwik-työkalu sekä WordPress. Tiettyjä lisäominaisuuksia voi aktivoida palvelun alle useamman, ja tiettyjä vain yhden. Asiakkaat pystyvät myös hallinnoimaan verkkotunnukseensa liitettyjä muita palveluja DNS-tietueiden hallintatyökalun avulla, aktivoimaan SSL-sertifikaatin sivustolleen, tarkastelemaan kävijä- ja virhelokia sekä luomaan uusia rinnakkais- ja alidomaineja. /63/

Hallintapaneelista pystyy hallinnoimaan myös PHP-asetuksia. Asiakas voi vaihtaa PHP:n versioita yhdellä klikkauksella sekä aktivoida ja deaktivoida eri PHP-moduuleja. /63/

4.4 Vertailu

Taulukkoon 3 on kerätty ominaisuudet, joita MSP:ltä odotetaan ennen virallista käyttöönottoa.

Taulukko 3. Palvelualustojen vertailu

	cPanel	Cloudways	Cloudcity	MSP
Web-sovellusten automaattinen käyttöönotto	Muutamat PHP-sovellukset	PHP-sovellukset	Vain WordPress.	Kyllä
SSL-sertifikaatin luominen	Ei automatisoitu	Automatisoitu	Automatisoitu	Automatisoitu
Verkkotunnuksen tilaus	Ei	Ei	Kyllä	Kyllä
Käyttäjähallinta	Kyllä	Kyllä	Ei	Kyllä
Sovelluskohtainen käyttöliittymähallinta	Ei	Kyllä	Kyllä (WordPress-tietokannan tiedot)	Kyllä
Terminaaliyhteys	Kyllä (palvelintasolla)	Kyllä (palvelin- ja sovellustasolla)	Kyllä (palvelintasolla)	Kyllä (sovellustasolla selaimessa)

Tarkastelluista alustoista sovelluskeskeisin on Cloudways, sillä se tarjoaa sovelluksen tyypistä riippuvia toimintoja käyttöliittymässä, esimerkiksi ohjauksen suoraan sovelluksen hallintapaneelin kirjautumissivulle. Cloudcity ja cPanel keskittyvät enimmäkseen webhotellien käyttöliittymäpohjaiseen hallintaan yleisellä tasol-

la, joskin Cloudcityn alustalla juuri WordPressin luominen onnistuu automatisoidusti yhdellä klikkauksella.

Kun Cloudwaysiä tarkastellaan pelkästään käyttöliittymän ja käytettävyyden kannalta, siinä on mainituista kolmesta alustasta eniten samaa, mitä MSP:lta odotetaan. Alustojen käyttötarkoitus on kuitenkin hieman eri; Cloudways tarjoaa palvelujaan kenelle tahansa, kun taas MSP:n kohdalla sitä hallinnoiva hosting-tarjoaja määrittelee, kenellä on oikeus käyttää sovellusta. Cloudways siirtääkin vastuun hostingista ulkopuolisille tahoille ja tarjoaa ainoastaan rajapinnan joidenkin PHP-sovellusten automaattiseen käyttöönottoon. MSP:ia hallinnoivalla hosting-tarjoajalla on lähtökohtaisesti itsellään vastuu hostingista, eikä asiakkaalle välttämättä ole merkitystä, missä palvelut fyysisesti sijaitsevat.

Verrattaessa MSP:ia tarkasteltuihin alustoihin, suurimpana erona on sen *sovelluskeskeisyys*. Yksittäinen palvelu kuvaa sovellusta, johon liittyy tai voi olla liittymättä muita komponentteja. Tarkastelluissa alustoissa sovellukset ovat enemmän tai vähemmän lisäkomponentteja, ja pääpaino suuntautuu taustalla olevan webhotellin hallintaan. Tätä projektia aloitettaessa katsottiin, että itse sovelluksen toiminta on asiakkaalle tärkeintä, ja muut asiat ovat sivuseikkoja. Palvelukonsepti ei kuitenkaan rajoitu tähän, vaan asiakas voi esimerkiksi tilata yksittäisen verkkotunnuksen omana palvelunaan. Myöhemmin tilatun verkkotunnuksen voi siirtää olemassa olevaan palvelukokonaisuuteen.

MSP:n vahvuutena muihin alustoihin nähden on palveluun liittyvien komponenttien, kuten SSL-sertifikaattien ja verkkotunnusten, tilaaminen yksittäisille palveluille. Teknisesti katsoen ne eivät itsessään ole sovelluksen vaatimuksia, vaan ominaisuuksia, jotka voidaan implementoida halutulla tavalla vaatimusten täyttyessä sovelluksen liiketoimintalogiikkaa muuttamatta, mikä tekee MSP:sta modulaarisen ja uudelleenkäytettävän.

MSP on myös riippumaton tarjottavien sovellusten teknologioista. Kaikki tarkastellut alustat ovat riippuvaisia taustalla olevan palvelimen infrastruktuurista, mikä rajoittaa erityyppisten sovellusten käyttöönottoa. Kuten taulukosta 2 nähdään, MSP:ia lukuun ottamatta, kaikilla alustoilla käyttöönotettavat sovellukset rajoittu-

vat PHP-sovelluksiin. On toki huomioitava, että suurin osa suosituimmista sisällönhallintajärjestelmistä ovat PHP-sovelluksia, minkä vuoksi niiden tarjoaminen ensisijaisesti on liiketoiminnallisesta näkökulmasta hyvinkin perusteltua /40/. Muuttuvalla IT-alalla tulevaisuuden ennustaminen on kuitenkin vaikeaa, joten on vähintään yhtä perusteltua pyrkiä olemaan tukeutumatta ainoastaan tietyllä hetkellä vallitseviin trendeihin.

5 TOTEUTUS

Tässä luvussa kuvataan keskeisimmät osat toteutuksesta.

5.1 Autentikointi ja autorisointi

Autentikointi suoritetaan LDAPConnector-luokan *authenticate*-metodilla. Metodin parametreja ovat käyttäjätunnus ja salasana sekä tieto autentikointitasosta. Autentikointitaso on joko *customer* tai *admin*. Bind-operaatioon käytettävä DN rakennetaan konfiguroidun pohjan, käyttäjätunnuksen sekä autentikointitason perusteella. Metodista palautetaan Optional-luokan AuthUser-tyyppinen instanssi, joka on tyhjä, mikäli bind-operaatio ei onnistu. Metodi olettaa, että käyttäjätunnuksena käytetään LDAP-merkinnän cn-attribuuttia. Saman aktiivisen LDAP-yhteyden aikana haetaan tieto käyttäjästä sekä tämän organisaatiosta, johon hän LDAPissa olevan tiedon perusteella kuuluu.

```
// Create a connection to LDAP directory
// with configured host, port and list of
// trusted hosts.
connection = createConnection();

// Perform a bind operation to LDAP direcory
// with built DN and provided password
connection.bind(dn.toString(), password);

// Build an LDAP search with required information
SearchRequest request = new SearchRequest(
        // Build a base dn based on configured values
        new DN(baseRDNs).toString(),
        // LDAP sub scope
        SearchScope.SUB,
        // First index is cn attribute,
        // which is going to be the filter
        dn.getRDNs()[0].toString()
);
```

Kuva 17. Käyttäjän haku hakemistosta

Mikäli sisäänkirjautuminen onnistuu, REST APIlta palautetaan HTTP-vastaus statuksella 200. Vastauksen mukana saadaan myös generoitu valtuutusavain, jonka käyttöliittymäsovellus tallentaa selaimeen. Samaan aikaan valtuutusavain tallennetaan sovelluksen välimuistiin, jotta käyttäjä voidaan validoida jokaisella HTTP-pyynnöllä. Valtuutusavaimet tallennetaan HashMapiin, jossa avaimina ovat valtuutusavaimet ja arvoina niihin liittyvät tiedot, kuten käyttäjä sekä tämän organisaatio.

Käyttäjät validoidaan valtuutusavaimen perusteella mukautetun suodattimen avulla, joka on Jerseyn Servletien edessä (**Kuva 18.**). Suodatin päästää läpi sellaiset pyynnöt, jotka kohdistuvat julkisiksi konfiguroituihin polkuihin. Mikäli polku on suojattu, pyynnön lähettäjä validoidaan pyynnön mukana tulevan valtuutusavaimen perusteella. Tämän jälkeen pyyntöön lisätään otsikoiksi valtuutusavaimella löytyneen käyttäjän tunniste sekä tämän organisaation tunniste, jotta niitä voidaan käyttää resurssiluokkien metodien tietokantakyselyissä. Mikäli pyynnön lähettänyt käyttäjä on validi, pyyntö päästetään muokattuna suodatinketjussa eteenpäin.

```
// Token parsed from valid request
UUID token = UUID.fromString(tokenString);

// Find AccessToken instance based on token value
Optional<AccessToken> accessTokenOptional = accessTokenDAO.findById(token);

if (accessTokenOptional.isPresent()) {
    requestWrapper.addHeader(
            "User-Header-Name",
            String.valueOf(
                    accessTokenOptional
                            .get()
                            .getUser()
                            .id()
            )
    );

    Long activeOrganizationId = accessTokenOptional
            .get()
            .getUser()
            .organization().id();

    // Add a header, which is used to identify user's organization.
    // RequestWrapper overrides headers if they already exist
    // so abuses are not possible
    requestWrapper.addHeader(
            "Organization-Header-Name", activeOrganizationId != null
                    ? String.valueOf(activeOrganizationId)
                    : null
    );
    filterChain.doFilter(requestWrapper, servletResponse);
} else {
    unauthorized(response);
}
```

Kuva 18. Käyttäjän validointi Servlet-suodattimessa

Autorisointi suoritetaan toisessa suodattimessa (**Kuva 19.**). Suodatin validoi käyttäjän valtuutusavaimen perusteella ja liittää pyynnön kontekstiin tiedon käyttäjän roolista, jota käytetään resurssiluokkien RolesAllowed-annotoiduissa metodeissa käyttäjien autorisointiin.

```
@Override
public void filter(final ContainerRequestContext containerRequestContext) throws IOException {
    // Find token from request
    String token = findTokenFromRequest(containerRequestContext)
        .orElseThrow(() ->
                new WebApplicationException(
                        "Not authorized",
                        Response.Status.UNAUTHORIZED)
        );

    try {
        containerRequestContext.setSecurityContext(
                // Get user information based on token
                authenticator.authenticate(token)
                        // Create a custom security context
                        // which will later be used to define is user's role
                        // allowed to access requested resource
                        .map(u -> new CustomSecurityContext(
                                u,
                                containerRequestContext.getSecurityContext()
                        )).orElseThrow(
                                () -> new WebApplicationException(
                                        "Not authorized",
                                        Response.Status.UNAUTHORIZED)
                )
        );
    } catch (AuthenticationException e) {
        throw new WebApplicationException(unauthorizedHandler.buildResponse(prefix, realm));
    }
}
```

Kuva 19. Autorisointisuodatin

Kuvaan 20 on otettu esimerkiksi palvelun luonnin suorittava metodi, jossa käytetään edellä mainittujen suodattimien tietoja.

```
// Allow only certain customer roles to
// perform this operation
@RolesAllowed({"admin", "developer"})
@POST
@ApiOperation("Add new service")
public ServiceView addService(
        CreateServiceRequest request,
        @ApiParam(hidden = true)
        // User id and organization id,
        // which are added to requests by a filter
        @HeaderParam("User-Header-Name") LongParam userId,
        @HeaderParam("Organization-Header-Name") LongParam organizationId
) {
```

Kuva 20. Käyttäjään ja rooliin kohdistettu addService-metodi

5.2 Kommunikointi tietokantaan

Tietokantakyselyihin käytetään *repo*-pakkauksen luokissa jOOQ-kirjastoa, joka antaa mahdollisuuden käyttää natiivin SQL:n kaltaista syntaksia kyselyiden toteuttamiseen (**Kuva 21.**). Tietokannan tauluista voidaan generoida luokat, jotka sisältävät tietokannan kentän sisältävät attribuutit oikeine tyyppeineen. jOOQ ei kuitenkaan ole ORM-kirjasto, joten tietokannan relaatiomalli on muutettava oliomallin mukaiseksi manuaalisesti.

```
public List<Service> fetchByOrganization(
    Long organizationId
) {
    return DSL.using(this.configuration)
            .select(
                    SERVICE.ID,
                    SERVICE.NAME,
                    SERVICE.TYPE,
                    SERVICE.DISABLED,
                    SERVICE.CREATED,
                    SERVICE.CREATED_BY,
                    PORTAL_USER.ID,
                    PORTAL_USER.NAME
            )
            .from(SERVICE)
            .leftJoin(ORGANIZATION).on(SERVICE.ORGANIZATION_ID.equal(ORGANIZATION.ID))
            .leftJoin(PORTAL_USER).on(SERVICE.CREATED_BY.equal(PORTAL_USER.ID))
            .where(SERVICE.ORGANIZATION_ID.equal(organizationId))
            .orderBy(SERVICE.NAME)
            .fetch()
            .stream()
            // Perform object relational mapping
            // by a custom method
            .map(ServiceRepo::map)
            .filter(Optional::isPresent)
            .map(Optional::get)
            .collect(Collectors.toList());
}
```

Kuva 21. ServiceRepo-luokan fetchByOrganization-metodi

5.3 Handler-rajapinta

Ulkoisten järjestelmien kanssa kommunikointiin käytetään Handler-rajapinnan toteuttavia luokkia eli *käsittelijöitä* (**Kuva 22.**). Rajapinnan metodit on pyritty pitämään universaaleina tulevien järjestelmäintegraatioiden helpottamiseksi. Alkuperäistä rajapintaa jouduttiin kuitenkin hieman muokkaamaan, koska sovelluksen halutaan tarjoilevan tällä hetkellä OpenShiftin lokit sekä podien terminaalipohjaisen hallinnan. Tätä varten sovelluksen Jersey-kontekstiin täytyy rekisteröidä WebSocket-proxy, mikä tehdään käsittelijän kautta, jotta rekisteröinti voidaan suorittaa jokaisen ulkoisen järjestelmän kohdalla erikseen. Jokaisella käsittelijällä on itsellään tieto siitä, minkälaisia rekisteröintejä sen täytyy tehdä koko sovelluksen kontekstiin. Rekisteröinnin suorittaa Handler-rajapinnan *register*-metodi.

Handler-rajapinnan *fetch*-metodia käytetään tiedon lukemiseen ulkoisista järjestelmistä. Metodi ottaa parametrina ServiceComponent-olion, jonka perusteella ulkoisen resurssin luku suoritetaan. Muiden operaatioiden suorittamiseen käytetään *handle*-metodia, joka ottaa parametreina Service-olion, Action-olion sekä kokoelman käyttäjän syöttämiä muuttujia. Metodi itsessään vastaa muuttujien sulauttamisesta Action-olion data-attribuuttiin. Kyseistä metodia käytetään muun muassa resurssien luomiseksi ulkoisiin järjestelmiin. Rajapinnan *componentOf-*

metodi palauttaa parametrina annetun Action-olion perusteella alustavan Service-Component-olion.

```
public interface Handler {
     // Registers Handler's self-known resources to
     // Application context
    void register(
            Consumer<Class<?>> registrar,
            BiConsumer<Class<?>, String> wsRegistrar
    );

     // Returns an Optional instance of ServiceComponent
     // based on Action.
    Optional<ServiceComponent> componentOf(Action action);

     // Handles the Action and does the required operations
     // to the external system based on the provided Action.
     // Returns an Optional instance of ServiceComponentView
    Optional<ServiceComponentView> handle(
            Service service,
            Action action,
            List<UserData> userData
    );

     // Returns an Optional instance of ServiceComponentView
     // based on provided component.
    Optional<ServiceComponentView> fetch(ServiceComponent component);
}
```

Kuva 22. Handler-rajapinta

5.3.1 Implementaatioesimerkkejä

HandlerImpl-luokka toteuttaa Handler-rajapinnan, vaikka se ei olekaan ulkoiseen järjestelmään yhdistetty käsittelijä. Sen tehtävänä on ainoastaan välittää metodikutsut oikeille käsittelijöille, kuten kuvasta 23 huomataan.

```
public Optional<ServiceComponentView> handle(
        Service service,
        Action action,
        List<UserData> userData
) {
    for (Handler h : this.handlers) {
        Optional<ServiceComponentView> response = h.handle(
                service,
                action,
                userData
        );
        if (response.isPresent()) return response;
    }
    return Optional.empty();
}
```

Kuva 23. HandlerImpl-luokan handle-metodi

Varsinaiset käsittelijät otetaan käyttöön HandlerImpl-luokan konstruktorissa, jossa ne muodostetaan kokoelmaksi. Käsittelijöiden järjestys kokoelmassa on merkityksellinen, mikäli halutaan käyttää varmistavaa käsittelijää, jonka tulee olla kokoelman viimeinen. Varmistava käsittelijä käsittelee kaikki sellaiset käskyt, joita muut käsittelijät eivät tunnistaneet.

Ulkoisiin järjestelmiin yhdistetyistä käsittelijöistä on tällä hetkellä toteutettuina OpenShiftHandler sekä RedmineHandler. OpenShiftHandleriin on toteutettu uuden OpenShift-projektin luominen.

RedmineHandleriä käytetään varmistavana käsittelijänä, joka osaa käsitellä eksplisiittisesti kaikki sille osoitetut käskyt sekä implisiittisesti sellaiset käskyt, joita muut käsittelijät eivät tunnistaneet. Implisiittisessä käsittelyssä Redmineen luodaan tapahtuma, jossa on merkkijonorepresentaatio komponentista, jolle yritettiin suorittaa käskyä.

Varmistavan käsittelyn avulla MSP:iin voidaan antaa tarjolle minkä tyyppisiä komponentteja tahansa, vaikka niiden automatisointia ei olisikaan vielä implementoitu. Käyttäjille voidaan esimerkiksi antaa mahdollisuus varata palveluunsa verkkotunnus, vaikka yksikään käsittelijä ei pystyisikään automaattisesti sitä varaamaan. Tässä tapauksessa verkkotunnus voidaan varata käyttäjälle manuaalisesti Redminen tapahtumassa olevia tietoja käyttäen, ja käyttöliittymässä voidaan näyttää verkkotunnus komponenttina normaalisti.

Resurssiluokat voivat käyttää Handler-rajapinnan metodeita tietämättä mitä käsittelijää taustalla varsinaisesti käytetään. Kun rajapinnalta esimerkiksi haetaan yk-

sittäinen palvelu, sille voidaan hakea kaikki siihen liittyvät komponentit kaikista ulkoisista järjestelmistä pelkkää *fetch*-metodia käyttäen (**Kuva 24.**).

```
return ServiceView.builder()
        .setService(service)
        .setComponents(
                new ImmutableList.Builder<ServiceComponentView>()
                        .addAll(
                                service.components()
                                        .stream()
                                        // Fetch all external resources
                                        // into service components
                                        .map(handler::fetch)
                                        .filter(Optional::isPresent)
                                        .map(Optional::get)
                                        .collect(Collectors.toList())
                        )
                .build()
        )
        .build();
```

Kuva 24. ServiceResourcen getService-metodin palautus

6 TESTAUS

Jatkuvan manuaalisen testauksen lisäksi MSP:ia testataan E2E-testeillä. Työkalut E2E-testien kirjoittamista ja suorittamista varten ovat Mocha-testiautomaatiokehys sekä Selenium WebDriver. Testitapaukset määritellään Mochan BDD-rajapinnan avulla, ja assertiokirjastona käytetään Chai.js:iä. Mocha suorittaa testit Node.js:ssä. Selenium WebDriveriä käytetään automatisoitujen käskyjen lähettämiseksi selaimeen, jotta sovelluksen toiminta voidaan testata käyttöliittymästä asti. /58, 59/

MSP:n E2E-testauksen arkkitehtuurimallina käytetään *robottimallia*. Robotti-konsepti tuo abstraktiotason käyttöliittymässä olevan näkymän ja itse testin väliin. Näin testitapaukset voidaan ajatella korkealta tasolta, eikä jokaista testiä tarvitse kirjoittaa uusiksi sovelluksen muuttuessa. Robottia voidaan ajatella manuaalisen testaajan korvaajana; manuaaliselle testaajalle kerrotaan, mitä hänen pitää testata, ja hän suorittaa testin käyttämällä senhetkistä näkymää. Vaikka jokin näkymässä tai muualla koko sovelluksen arkkitehtuurissa muuttuu, testitapaus pysyy edelleen samana. Testeihin sisältyvien toimintojen suorittamislogiikka määritellään robottiin, jota muutetaan tarvittaessa. /57/

Yksi MSP:n testitapauksista on ”Vaihda LDAP-salasana kelvollisilla syötteillä.” (**Kuva 25.**). Operaation suorittamiseksi on syötettävä vanha salasana sekä kaksi kertaa uusi salasana. Lisäksi on varmistuttava siitä, että kaikki syötteet ovat kelvollisia ja että operaatio onnistuu. Sillä ei ole merkitystä, kuinka robotti suorittaa edellä mainitut tavoitteet.

Kaikkia testattavia tapauksia ei ole vielä automatisoitu. Työkalut ovat kuitenkin olemassa, ja testejä kirjoitetaan jatkuvasti.

```
describe('Password change', () => {

  // ...init and destroy driver here

  it('succeeds with valid input', async () => {
    let pwc = new PasswordChangeRobot(driver)
    await pwc.get()

    // Input data
    await pwc.inputOldPassword('correctpassword')
    await pwc.inputNewPassword('newpassword')
    await pwc.inputNewPasswordRepeat('newpassword')

    // Assert that input is valid
    expect(await pwc.inputIsValid()).to.equal(true)

    // Assert that password change succeeds
    expect(await pwc.submit()).to.equal(true)
  })
})
```

Kuva 25. Salasanan vaihdon suorittava testi

E2E-testauksen ja valitun arkkitehtuurin avulla voidaan validoida tiettyyn pisteeseen asti myös ulkoisissa järjestelmissä oleva data. Aukkoja kuitenkin jää, jos tukeudutaan pelkästään tähän lähestymistapaan. Esimerkiksi käsittelijöille on kirjoitettava erikseen integraatiotestit, jotta voidaan varmistua, etteivät ne suorita ylimääräisiä operaatioita ulkoisiin järjestelmiin. Muun muassa OpenShiftHandler on integroitu hosting-tarjoajan kannalta erittäin kriittiseen järjestelmään, joten sen suorittamat käskyt on validoitava hyvin tarkasti.

Selenium WebDriveriä voidaan käyttää *headless*-tilassa CI-palvelimella, millä voidaan varmistaa, että tuotantoon päätyy ainoastaan kaikki testit läpäisevä versio. Koska kyse on useisiin ulkoisiin järjestelmiin integroitavista sovelluksista, täytyy ulkoisista järjestelmistä käynnistää tuolloin vähintään testin ajaksi väliaikaiset instanssit.

7 JATKOKEHITYS

Vaikka MSP:n nykyiset ominaisuudet on testattu toimiviksi ulkoisiin järjestelmiin, uusia ominaisuuksia sekä pieniä muutoksia vaaditaan ennen käyttöönottoa tuotantoympäristöön. Tässä luvussa on kuvattu MSP:n tämänhetkisiä ongelmia ja lähitulevaisuudessa toteutettavia uusia ominaisuuksia.

7.1 Nykyiset ongelmat ja puutteet

Handler-rajapinnassa on tietynlaisia puutteita. Mikäli esimerkiksi luodaan palvelu, ja jonkin resurssin provisiointi ulkoiseen järjestelmään ei onnistu, syntyy ongelma. Koska käsittelyä yritettiin jo tehdä muualla, kutsu ei koskaan saavu varmistavalle käsittelijälle, jolloin tieto provisioinnin epäonnistumisesta ei kulje MSP:n hallinnoijaosapuolelle. Tämä ongelma voidaan kuitenkin ratkaista eri tavoin; jokainen HandlerImpl-oliolle suunnattu kutsu voidaan välittää viimeisenä varmistavalle käsittelijälle riippumatta siitä, onko kutsu jo kerran käsitelty, tai HandlerImpl-luokan metodeissa voidaan tehdä tarkistuksia käsittely-yrityksistä ja ohjata kutsut virheiden sattuessa varmistavalle käsittelijälle.

Käyttäjän pitää pystyä antamaan palvelun luonnin jälkeen käskyjä sen komponentteja vastaaviin ulkoisten järjestelmien resursseihin, mikä ei toistaiseksi ole mahdollista. Näihin käskyihin kuuluvat esimerkiksi OpenShiftin tapauksessa deploymentin uudelleenkäynnistys, sammutus, käynnistys ja poistaminen. Ratkaisu tämän toiminnallisuuden toteuttamiseksi on jo tehty ja implementaatio on aloitettu. Ideana on palauttaa käsittelijältä komponentin mukana lista käskyistä, joita käyttäjä voi sille ja sen aliresursseille antaa. Selainkäyttöliittymään voidaan jopa renderöidä käskyjä kuvaavat painikkeet, joita painamalla kyseinen käsky lähetetään REST APIn kautta aina käsittelijälle asti.

Luodun palvelun poistaminen ei ole mahdollista ennen kuin edellä mainitut puutteet on korjattu, sillä palvelua poistettaessa on poistettava kaikki komponentit ja niihin viittaavat resurssit ulkoisista järjestelmistä. Poiston epäonnistuessa tiedon on myös kuljettava MSP:n hallinnoijaosapuolelle, jotta resurssit voidaan epäonnistumisten sattuessa poistaa manuaalisesti.

Myös LDAP-integraatiossa on monia ongelmia, kun ajatellaan MSP:n uudelleenkäyttömahdollisuuksia. Konfiguraatiotiedostossa voidaan määritellä käytettävä skeema, mutta hakuja suorittavissa metodeissa tehdään liikaa oletuksia LDAP-palvelimen skeemasta, mikä supistaa konfiguraatiomahdollisuudet todellisuudessa hyvin pieniksi. MSP:ssa myös hyödynnetään vain tiettyjen LDAP-palvelimien tukemia ominaisuuksia, eikä standardeja noudateta täysin. LDAP-hakuja tehdään toisaalta ainoastaan yhdessä luokassa, joten puutteiden korjaaminen lienee suhteellisen vaivatonta. MSP tukee tällä hetkellä autentikointia ainoastaan LDAPin kautta, mikä myös hankaloittaa uudelleenkäyttöä.

7.2 Tulevat ominaisuudet

MSP:iin tullaan lähiaikoina toteuttamaan muun muassa verkkotunnuksen automaattinen tilaaminen, SSL-sertifikaatin luominen ja automaattinen laskutus palveluille.

Automaattisen laskutuksen lisääminen palvelulle vaatii datamalliin tila- ja hintatiedon lisäämisen palvelukomponenteille. Ajatuksena on ollut, että käyttäjän palvelut voivat olla kehitys- tai tuotantotilassa, mikä vaikuttaa laskutukseen. Kaikilla palvelukomponenttityypeillä on oma hinta, ja palvelun kokonaishinta muodostuu komponenttien hintojen summasta. Palveluista voidaan laskuttaa esimerkiksi minuuttikohtaisesti, mutta lopullista päätöstä sen suhteen ei olla tehty. Automaattisen laskutuksen implementointi vaatii loogisesti myös sen, että käyttäjä pystyy poistamaan ja näin ollen irtisanomaan palvelujaan.

SSL-sertifikaatin luominen tullaan mitä todennäköisimmin sitomaan OpenShift-Handleriin ja sen reittien käsittelyyn. Asiakas voi luoda käyttöliittymän kautta palveluunsa liittyviin OpenShift-projekteihin suojattuja tai suojaamattomia reittejä. Suojauksen voi myös poistaa ja lisätä reittikohtaisesti, ja sertifikaatteja pystyy uusimaan ulkoisen palvelun avulla.

Verkkotunnuksen tilaamisen automatisoinnin toteutus MSP:iin on mahdollista jo nykyimplementaatiolla. Täytyy ainoastaan luoda käsittelijä, joka suorittaa tarvit-

tavat käskyt johonkin ulkoiseen järjestelmään, josta verkkotunnuksia pystyy tilaamaan.

8 YHTEENVETO

Ydintavoitteena oli suunnitella ja toteuttaa skaalautuva arkkitehtuurimalli, jota hyödyntäen pystytään rakentamaan kokonaisvaltainen useaan järjestelmään integroitu palvelualusta, esimerkiksi web-palvelujen hallinnoimiseen. Sovellus integroitiin myös joihinkin ulkoisiin järjestelmiin luotua arkkitehtuurimallia käyttäen. Työssä toteutettiin myös suurin osa taulukon 2 vaatimuksista. Vaatimusten toteutusten osalta kesken jäivät V2 ja V5. V2:n osalta toteutettiin uuden palvelun luominen, ja muut CRUD-operaatiot on toteutettu opinnäytetyön ulkopuolella.

Vaikka alussa tiedettiin, mihin ulkoisiin järjestelmiin MSP tullaan integroimaan sen ensisijaista käyttötarkoitusta varten, toteutettaessa järjestelmiin ei saanut kiinnittää liikaa huomiota, koska tulevaisuuden laajentamismahdollisuudet haluttiin pitää mahdollisimman suurina. Työn suurin haaste olikin tarpeeksi abstraktin käsittelylogiikan toteuttaminen, mikä johti projektin alkuvaiheessa useisiin liian konkreettisiin ja näin ollen nopeasti puutteellisiksi todettuihin variaatioihin.

Käyttöönottoa ajateltaessa olemassa olevien palvelukokonaisuuksien migraatio MSP:iin täytyy suunnitella huolellisesti. Tällä hetkellä palvelulla on tietokannassa relaatio aina johonkin palvelumalliin, josta se on luotu, joten yhtenä vaihtoehtona on käyttöönottovaiheessa luoda MSP:iin olemassa olevia palvelukokonaisuuksia vastaavat palvelumallit. Toinen vaihtoehto on poistaa datamallista palvelun ja palvelumallin välinen relaatio, joka ei välttämättä ole pakollinen. Kumpi tahansa lopullisista vaihtoehdoista valitaankaan, varsinainen palvelukomponenttien migroiminen MSP:iin tullaan todennäköisesti tekemään puoliautomatisoidusti jonkinlaista skriptiä apuna käyttäen. Palvelut liittyvät aina johonkin asiakkuuteen, joten migraatio vaatii myös asiakasorganisaatioiden lisäämisen käytettävään LDAP-hakemistoon sekä käyttöoikeuden antamisen kyseisille organisaatioille MSP:iin.

Projekti on yhä kehitysvaiheessa, joten vielä on vaikea arvioida laajemmin työn onnistumista. MSP kuitenkin onnistuttiin integroimaan luotua arkkitehtuurimallia hyödyntäen kahteen ulkoiseen järjestelmään, mikä antaa jonkinlaista signaalia lopputuloksesta sekä hyvän pohjan jatkokehitykselle. MSP ei ole vielä tuotantokelpoinen, sillä luvussa 7 mainittujen puutteiden lisäksi myös käyttöliittymäpuoli

vaatii lisäominaisuuksia, jotta käyttöönotosta olisi sekä asiakkaalle että hallinnoi-jaosapuolelle hyötyä.

LÄHTEET

/1/ Jubic Oy. 2017. Verkkosivut. Viitattu 8.4.2017. http://www.jubic.fi/

/2/ OpenShift. 2017. OpenShift-projektin virallinen verkkosivu. Viitattu 18.4.2017. https://www.openshift.com/about/index.html

/3/ Shipley, G. ja Dumpleton G. 2016. OpenShift for Developers. 2. uud. painos. Sebastopol, Kalifornia. O'Reilly Media, Inc.

/4/ Docker. 2017. What is Docker. Viitattu 18.4.2017. http://www.docker.com/what-docker

/5/ Docker. 2017. What is Container. Viitattu 18.4.2017. https://www.docker.com/what-container

/6/ docker. 2017. GitHub. Viitattu 18.4.2017. https://github.com/docker/docker

/7/ Kubernetes. 2017. Kubernetes Clusters. Viitattu 19.4.2017. https://kubernetes.io/docs/tutorials/kubernetes-basics/cluster-intro/

/8/ Kubernetes. 2017. Kubernetes Deployments. Viitattu 19.4.2017. https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-intro/

/9/ Kubernetes. 2017. Kubernetes Pods. Viitattu 19.4.2017. https://kubernetes.io/docs/tutorials/kubernetes-basics/explore-intro/

/10/ Kubernetes. 2017. Overview of Kubernetes Services. Viitattu 19.4.2017. https://kubernetes.io/docs/tutorials/kubernetes-basics/expose-intro/

/11/ Kubernetes. 2017. Scaling an application. Viitattu 19.4.2017. https://kubernetes.io/docs/tutorials/kubernetes-basics/scale-intro/

/12/ Kubernetes. 2017. Updating an application. Viitattu 19.4.2017. https://kubernetes.io/docs/tutorials/kubernetes-basics/update-intro/

/13/ Kubernetes. 2017. Understanding Pods. Viitattu 19.4.2017. https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/

/14/ Docker docs. Get Started, Part 2: Containers. Viitattu 19.4.2017. https://docs.docker.com/get-started/part2/

/15/ Kubernetes. 2017. What is Kubernetes? Viitattu 22.4.2017. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

/16/ Kubernetes. 2017. Sharing a Cluster with Namespaces. Viitattu 20.5.2017. https://kubernetes.io/docs/tasks/administer-cluster/namespaces/

/17/ OpenShift. 2017. Projects and Users. Viitattu 3.6.2017. https://docs.openshift.com/enterprise/3.0/architecture/core_concepts/projects_and_users.html#architecture-core-concepts-projects-and-users

/18/ Kubernetes. 2017. Replication Controller. Viitattu 4.6.2017. https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller/

/19/ Kubernetes. 2017. Volumes. Viitattu 4.6.2017. https://kubernetes.io/docs/concepts/storage/volumes/

/20/ Kubernetes. 2017. Persistent Volumes. Viitattu 4.6.2017. https://kubernetes.io/docs/concepts/storage/persistent-volumes/

/21/ OpenShift. 2017. Routes. Viitattu 4.6.2017. https://docs.openshift.org/latest/dev_guide/routes.html

/22/ OpenShift. 2017. Deployments. Viitattu 16.8.2017. https://docs.openshift.com/enterprise/3.0/architecture/core_concepts/deployments.html

/23/ OpenShift. 2017. Deployment Strategies. Viitattu 16.8.2017. https://docs.openshift.com/container-platform/3.4/dev_guide/deployments/deployment_strategies.html#lifecycle-hooks

/24/ Pavić, A. 2016. Redmine Cookbook. Birmingham. Packt Publishing Ltd.

/25/ Lesyuk, A. 2016. Mastering Redmine. 2. uud. painos. Birmingham. Packt Publishing Ltd.

/26/ Redmine. 2017. Changelog for 0.6.x and below. Viitattu 16.8.2017. http://www.redmine.org/projects/redmine/wiki/Changelog_0_6#v010-2006-06-25

/27/ Redmine. 2017. Redmine API. Viitattu 23.8.2017. http://www.redmine.org/projects/redmine/wiki/Rest_api

/28/ Donley, C. 2003. LDAP Programming, Management and Integration. Manning Publications Co.

/29/ IANA. 2017. Service Name and Transport Protocol Port Number Registry. Viitattu 6.9.2017. https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

/30/ UnboundID. 2015. The LDAP Bind Operation. Viitattu 9.9.2017. https://www.ldap.com/the-ldap-bind-operation

/31/ RFC4422. Simple Authentication and Security Layer (SASL). Internet Engineering Task Force. 2006. 31s.

/32/ IANA. 2015. Simple Authentication and Security Layer (SASL) Mechanisms. Viitattu 9.9.2017. https://www.iana.org/assignments/sasl-mechanisms/sasl-mechanisms.xml

/33/ cPanel. 2017. Our history. Viitattu 21.9.2017. https://cpanel.com/company/

/34/ cPanel. 2017. cPanel Features. Viitattu 21.9.2017. https://cpanel.com/products/

/35/ Softaculous. 2017. Softaculous Compare. Viitattu 25.9.2017. https://www.softaculous.com/softaculous/compare

/36/ cPanel Documentation. 2017. Site Publisher. Viitattu 25.9.2017. https://documentation.cpanel.net/display/ALD/Site+Publisher

/37/ cPanel Documentation. 2017. Email Accounts. Viitattu 25.9.2017 https://documentation.cpanel.net/display/ALD/Email+Accounts

/38/ Cloudways. 2017. Story of Cloudways. Viitattu 21.9.2017 https://www.cloudways.com/en/about_us.php

/39/ Cloudcity. 2017. Webhotellien vertailu. Viitattu 21.9.2017. https://cloudcity.fi/hinnasto/

/40/ Mening, R. 2017. Market Share: Top Website Platforms & Example Sites. Viitattu 25.9.2017. https://websitesetup.org/popular-cms/

/41/ Dropwizard. 2017. Getting Started. Viitattu 25.9.2017. http://www.dropwizard.io/1.1.4/docs/getting-started.html

/42/ jackson. 2017. GitHub. Viitattu 25.9.2017. https://github.com/FasterXML/jackson

/43/ OpenShift. 2017. How Builds Work. Viitattu 17.10.2017. https://docs.openshift.org/latest/dev_guide/builds/index.html

/44/ OpenShift. 2017. Builds and Image Streams. Viitattu 17.10.2017. https://docs.openshift.com/enterprise/3.0/architecture/core_concepts/builds_and_image_streams.html#custom-build

/45/ Kavis, M. 2014. Architecting the Cloud. Hoboken, New Jersey. John Wiley & Sons, Inc.

/46/ Fielding, R. 2000. Chapter 5: Representational State Transfer (REST). Viitattu 18.10.2017. http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

/47/ TechTerms. 2016. API. Viitattu 18.10.2017. https://techterms.com/definition/api

/48/ RFC3986. Uniform Resource Identifier (URI): Generic Syntax. Internet Engineering Task Force. 2005. 61s.

/49/ RFC7231. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. Internet Engineering Task Force. 2014. 97s.

/50/ RFC793. Transmission Control Protocol. Internet Engineering Task Force. 1981. 85s.

/51/ RFC7159. The JavaScript Object Notation (JSON) Data Interchange Format. Internet Engineering Task Force. 2014. 15s.

/52/ RFC4825. The Extensible Markup Language (XML) Configuration Access Protocol (XCAP). Internet Engineering Task Force. 2007. 69s.

/53/ RFC5246. The Transport Layer Security (TLS) Protocol Version 1.2. Internet Engineering Task Force. 2008. 101s.

/54/ RFC791. Internet Protocol. Internet Engineering Task Force. 1981. 45s.

/55/ RFC768. User Datagram Protocol. Internet Engineering Task Force. 1980. 3s.

/56/ JAX-RS API. 2017. GitHub. Viitattu 18.10.2017. https://github.com/jax-rs

/57/ Wharton, J. 2016. Instrumentation Testing Robots. Viitattu 8.11.2017. https://academy.realm.io/posts/kau-jake-wharton-testing-robots/

/58/ Mocha. 2017. Mocha-projektin virallinen verkkosivu. Viitattu 8.11.2017. https://mochajs.org/

/59/ Selenium. 2017. Selenium WebDriver. Viitattu 8.11.2017. http://www.seleniumhq.org/docs/03_webdriver.jsp

/60/ Soni, N. 2016. What is an ORM (Object Relational Mapping)? Viitattu 16.11.2017. https://www.osinlab.com/what-is-an-orm/

/61/ CodeUtopia. What’s the difference between Unit Testing, TDD and BDD? Viitattu 16.11.2017. https://codeutopia.net/blog/2015/03/01/unit-testing-tdd-and-bdd/

/62/ Codeship. Continuous Integration Essentials. Viitattu 16.11.2017. https://codeship.com/continuous-integration-essentials

/63/ Cloudcity. 2017. Cloudcityn hallintapaneeli. Viitattu 17.11.2017. https://my.cloudcity.fi/

/64/ Cloudways. 2017. Cloudwaysin hallintapaneeli. Viitattu 17.11.2017. https://platform.cloudways.com/

/65/ cPanel. 2017. cPanelin demoversio. Viitattu 17.11.2017. https://demo.cpanel.net:2083/

/66/ WHM. 2017. WHM-hallintapaneelin demoversio. Viitattu 17.11.2017. http://trycpanel.net/