Karelia-ammattikorkeakoulu
Tradenomi
Tietojenkäsittely

Sovellusten käyttöönotto Kuberneteksella

Opinnäytetyö

Konsta Havukainen, 2108800

Opinnäytetyö, Marraskuu 2024

www.karelia.fi

Karelia AMMATTIKORKEAKOULU	**OPINNÄYTETYÖ** **Marraskuu 2024** **Tietojenkäsittelyn koulutus** Tikkarinne 9 80200 JOENSUU +358 13 260 600

Tekijä(t)
Konsta Havukainen

Nimeke
Sovellusten käyttöönotto Kuberneteksella
Toimeksiantaja
Broman Group Oy

Tiivistelmä

Opinnäytetyö on Broman Group Oy:n antama toimeksianto, jossa pyrittiin ottamaan käyttöön yrityksen paikallinen Kubernetes-klusteri. Käyttöönotolla tarkoitetaan yhteyksien ja resurssien luontia yrityksen järjestelmiin, jotta klusteria voidaan käyttää kehitystyössä vaihtoehtoisena alustana sovelluksille.

Opinnäytetyön aikana tutkittiin Kubernetes-projektin dokumentaatiota ja toteutettiin korkeasti saavutettavia palveluita tukevia järjestelmiä klusteriin. Työ toteutettiin tutustumalla Kubernetekseen asteittain. Aluksi toteutettiin tietoliikenteelle vaaditut resurssit ja sitten rakennettiin yhteydet muihin palveluihin kuten pilveen ja integraatioalustaan.

Klusterin käyttöönotto voitiin todeta onnistuneeksi, kun sinne voitiin julkaista versioittain sovelluksia järjestelmällisellä ja käytännöllisellä tavalla julkaistavan sovelluksen nykyisen kehityskaaren kannalta. Sovelluksia voi julkaista automaatioittain noudattamalla opinnäytetyön aikana sisäverkkoon julkaistua ohjetta. Ohje on osa opinnäytetyötä ja sen avulla aikaansaadut tuotokset ovat myös todiste opinnäytetyön muiden osa-alueiden toiminnasta. Opinnäytetyön tulokset tarjoavat Broman Groupin kehitysosastolle tavan julkaista sovelluksia vaihtoehtoiselle alustalle, kasvattaen valinnanvapautta kehitystoiminnassa.

Kieli suomi	Sivuja 37

kubernetes, korkea saavutettavuus, klusteri, kontti, automaatio

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
November 2024
Degree Programme in Business Administration

Tikkarinne 9¨
80200 JOENSUU
+358 13 260 600

Author (s)
Konsta Havukainen

Title
Deploying Applications to Kubernetes
Commissioned by Broman Group Oy

Abstract
This thesis is a commission by Broman Group Oy. The thesis is functional, and its purpose is to discover how the company's internal Kubernetes server could be implemented so that automated processes and developers may deploy applications to run within it. The outcome would offer an alternative platform for applications at their different stages of development.

During the thesis the documentation provided by the Kubernetes Project was studied and the required resources for accomplishing highly available applications were thoroughly studied and then implemented. At first the required components for functioning network connectivity were implemented and after that, the connections to external services such as cloud and integration platforms were implemented as well.

The deployment process can be considered accomplished when applications can be deployed to the cluster in an organized and practical manner according to the current development-cycle of the software to be deployed. Software can be deployed with an automation pipeline by following the guide provided to the company's internal forum. Successful software deployments achieved by following the guide, demonstrate that the systems developed during this thesis function as intended. The products of this thesis provide more options to the company concerning deployment platforms.

Language
Finnish

Pages 37

kubernetes, high availability, cluster, container, automation

Sisältö

1 Johdanto

Opinnäytetyössä on tarkoitus tuottaa ohje ja kartoitus yrityksen nykyisen klusterin, miten siinä voidaan ottaa käyttöön kontitettuja sovelluksia, eli eristettyjä, yksittäisiä työtehtäviä suorittavia, sovelluskokonaisuuksia. Dokumentaatio yrityksen Kuberneteksesta ja sen käytöstä tehdään sen jälkeen, kun yritykseen rakentuu järjestelmä, jolla käyttöönottoa voidaan tehdä. Järjestelmää rakennettaessa testataan useita eri Kuberneteksen teknologioita ja laajennuksia oppimismielessä ja näistä parhaimmat vaihtoehdot otetaan käyttöön. Dokumentaatio pyrkii selostamaan klusterin sen hetkisen tilan ja sen, miten kehittäjä voi ottaa käyttöön sovelluksia siinä ja mitä edellytyksiä käyttöönottoprosessille on.

Kubernetes on suurissa yrityksissä suosittu ratkaisu skaalautumisen ja palveluiden saavutettavuuden saralla. Broman Group konsernina vastaa useista kaupan alan tytäryhtiöstä, joista yksi on autoilu- ja vapaa-ajanliike Motonet. Liiketoiminnan kasvaessa yritykselle on tärkeää, että sen palveluja ylläpitävät resurssit skaalautuvat sen mukana. Broman Groupilla on yrityksen sisäverkossa käytössä toteutettu Kubernetes, mutta sitä ei ole hyödynnetty sen käyttötarkoituksen mukaisesti tätä opinnäytetyötä aloittaessa. Kubernetes on hyödyllinen, sillä se orkestraattorityökaluna ja oikein pystytetyssä arkkitehtuurissa tekee kontitettujen sovellusten monitoroinnista ja skaalaamisesta liiketoiminnan kehittyessä helpompaa. Yritys voi jatkossa käyttää omaa Kubernetestaan yhtenä vaihtoehtona julkaisualustaksi sovelluskehityksessä.

Broman Group hyödyntää Azuren Pipelines-toiminnallisuutta jatkuvan kehityksen työkaluna ja yhtenä opinnäytetyön tavoitteena on jatkaa valikoitujen projektien automaatioputkea niin, että kehityksen tulokset tulevat käyttöön Kubernetekseen putken avulla. Yrityksessä on tällä hetkellä sisäverkossa suoriutuvia yksittäisiä kontteja, joihin tapahtuva kehitys ei ole automatisoitua. Näitä sovelluksia voidaan siis siirtää suoriutumaan Kubernetekseen ja niiden päivitys ei vaadi manuaalista alasajoa automaatioputken tuomien etujen myötä.

Opinnäytetyön tavoitteena on arkkitehtuuri, jolla Kubernetekseen voi julkaista ja siirtää ohjelmistoja ja tuloksena on myös dokumentaatio, jonka avulla kehittäjät onnistuvat tässä.

2 Tietopohja

Liiketoiminnassa on erilaisia edellytyksiä ja tavoitteita, joiden mukaan uusia hankkeita ja infrastruktuureita otetaan käyttöön. Esimerkkinä ”Five-Nines”, joka on tavoitetila, jossa organisaation palvelut ovat 99.999 prosenttisesti saavutettavissa vuodessa. Toisin sanoen, vuodessa palveluiden seisahtuneisuus saa olla vain n. 5 minuuttia. Tämän tavoitteen saavuttamiseksi, organisaatiolle on tiettyjä edellytyksiä arkkitehtuurin suhteen. (Rensin 2015, 21)

2.1 Korkea Saavutettavuus - High availability

Korkea saavutettavuus tarkoittaa tilaa, jossa palveluiden seisahtuneisuus on olematonta. Virheet ja epäonnistumiset ohjelmissa eivät korkeasti saavutettavassa arkkitehtuurissa tarkoita palveluiden seisahtumista.
IT-järjestelmissä palveluiden seisahtuneisuus voi pahimmillaan merkitä jopa elämien menetystä. Vähemmän vakavien, mutta yrityksen liiketoiminnan kannalta merkittävien, toimintojen ja prosessien seisahtuneisuus merkitsee myös liiketoiminnan seisahtuneisuutta ja mahdollisesti myös asiakkaiden menetystä. Nämä riskit huomioiden korkea saavutettavuus on tietyissä palveluissa elinehto ja sen ylläpitäminen koko liiketoiminnan ajan on tärkeää. (IBM 2024).

Korkea saavutettavuus on yrityksille erittäin houkuttelevaa, mutta ennen teknologiaa on ymmärrettävä sen edellytykset. Korkea saavutettavuus voidaan määritellä tilana, joka täyttyy, kun tietyt neljä muuta tilaa ovat voimassa organisaation palveluilla. Näitä tiloja kutsutaan neljäksi korkean saavutettavuuden pilariksi (Kanso 2021).

Ensimmäinen korkean saavutettavuuden pilari on **redundanssi**. Se tarkoittaa sitä, että työtä tekevälle yksikölle – tässä tapauksessa podille– on olemassa

vastike, joka toimii täysin identtisesti kuin alkuperäinen yksikkö. **Podi** on Kuberneteksen pienin hallittava resurssi ja se vastaa konteista. Redundanssin merkitys on korkean saavutettavuuden edellytys siksi, että korvaajan löytyminen välittömästi eliminoi Single-Point-Of-Failure-kohtia ja siten ylläpitää palvelun katkeamattomuutta vikatilanteessa. (Kanso 2021.)

Toinen pilari korkeaa saavutettavuutta on **kyky monitoroida** ja tunnistaa vikatilanteita. Ilman monitorointia, palveluita orkestroiva entiteetti ei kykene tekemään päätöksiä, joilla työyksiköitä sammutetaan, korvataan ja käynnistetään uudelleen. Jotta vikatilanne voidaan ratkaista, se pitää havaita ensimmäisenä. (Kanso 2021.)

Kolmas pilari on elvytys – **recovery**. Vikatilanteessa orkestraattorin on osattava poistaa viallinen yksikkö ja korvattava se toimivalla palvelun saavutettavuuden ylläpitämiseksi. Ensimmäinen recoveryn toimenpide on eristää viallinen yksikkö tietoliikenteeltä ja ohjattava vialliselle yksikölle tarkoitettu liikenne toimiviin yksiköihin (Kanso 2021). Tämä sallii asiakkaalle pääsyn vastaavaan palveluun (redundancy), vaikkei yhteys enää kohdistukaan samaan yksikköön, jolla yhteys aloitettiin.

Neljäs saavutettavuuden pilari on etapit **– checkpoint**. Jos vikatilanteessa menetetään istunnon aikaisia tietoja, voidaan yksikkö palauttaa johonkin aiemmin tuntemaansa tilanteeseen. Jotta tilanteeseen voidaan palata, pitää järjestelmän tallettaa tarpeeksi säännöllisesti tilannekuvia, jotta tiedon menetys olisi mahdollisimman vähäistä. Tilannekuvia vaaditaan kokoonpanoissa, jossa istunnon kannalta olennaista on sen tila, tai sen istunnon aikana tehdyt muutokset johonkin järjestelemän tilaan (Kanso 2021). Tällaisia voivat olla lomakkeisiin tehdyt muutokset, jotka tilattomassa istunnossa vikatilanteessa tavallisesti häviäisivät. Jos istunnon tila tallennetaan muistiin 30 sekunnin välein, voidaan vikatilanteessa viallisen korvaajalle tuoda viallisen yksikön data muistista. Näin aiempi istunto voi jatkua menettäen maksimissaan viimeisein 30 sekunnin ajanjakson aikana tapahtuneet muutokset.

Korkea saavutettavuus on yrityksen sovellusten kannalta tärkeää, sillä Broman Groupissa on useita erilaisia sisäverkon sovelluksia, joita käsitellään selaimen avulla verkkoyhteyden välityksellä. Näitä sovelluksia voidaan ottaa käyttöön korkeasti saavutettavalla tavalla, nykyisen yksittäisen konttikohtaisen toteutuksen sijaan.

2.2 Klusterit

Klusterit ovat useista tietokoneista rakentuva verkko, jossa on yleensä yksi hallitseva palvelin, joka jakaa tehtäviä verkon muille tietokoneille. (Rensin 2015, 12) Useiden tietokoneiden samanaikainen työ vähentää laskennallisesti suuriin tehtäviin vaadittua aikaa. Klusterit edellä mainitun kuvauksen mukaisesti toteuttavat kuormanjakoa. Klustereita käytetään yrityksissä useisiin erilaisiin käyttötapauksiin. Ne voivat olla sovelluksia, palvelimia ja tietokantoihin erikoistuneita klustereita. (Gray 1991).

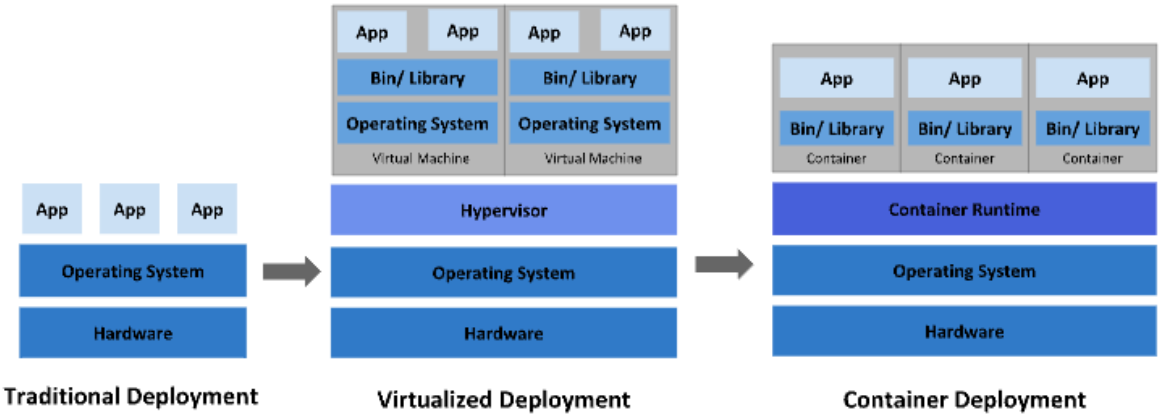
Klusterit jakavat tietokoneet rooleihin, joissa on tyypillisesti yksi hallinnoiva tietokone ja useampi työtä tekevä tietokone, jotka tunnetaan Kuberneteksen tapauksessa nodeina. Hallinnoivaa kokonaisuutta kutsutaan Control-Planeksi (Kubernetes docs 2023q.)

2.3 Konttisovellukset ja virtuaalikoneet

Ennen sovelluksille omistettiin vastuuseen nähden virtuaalikoneita, jotta esimerkiksi liiketoimintaprosessien eri osa-alueet ja vastuut pysyivät kapseloituina. Liiketoiminnan skaalautuminen kasvatti virtuaalikoneiden määrää ja niiden hallinnoinnista tuli erittäin työlästä.

Säilötyt, eli **kontitetut** (eng. containerized) sovellukset suoriutuvat virtuaaliympäristössä, jotka sisältävät vain juuri vaaditut kirjastot ja ympäristöt, jotka se tarvitsee suoritukseensa (Docker 2024). Suosituin konttisovellusten rakennusformaatti on Docker (Burns, Beda & Hightower 2022, 14). Perustavanlaatuinen ymmärrys erilaisista kontitusteknologioista kuten Dockerista ovat edellytyksiä

Kuberneteksen käyttäjälle, sillä Kubernetes orkestroi juuri kontitettuja sovelluksia (Kubernetes 2023, Overview). Kontitetut sovellukset ovat kevyempiä, monitoroitavampia ja eristetympiä kuin edeltävät käyttöönoton teknologiat (kuva 1).



Kuva 1. Vanhoja sovelluksen käyttöönottotapoja ja kontitus (Kubernetes 2023a).

2.4 Kuormanjako - Load-balancing

Kuormanjako tarkoittaa suuren työmäärän periaatteen mukaista jakoa työyksiköille.
Käytännössä kuormanjakoa toteuttavalla organisaatiolla on useita samankaltaisia palveluita, joiden avulla esimerkiksi suuren asiakasmäärän käsittely toimii nopeasti ja sujuvasti. Käytännönläheisenä esimerkkinä: supermarketeissa avataan useita kassoja, jotta kaikki asiakkaat pääsevät kassalle, eikä heidän tarvitse jonottaa palvelua yhdeltä kassalta.

Kuormanjaolla verkkopalveluissa tarkoitetaan tietoliikenteen aiheuttaman kuorman jakamista useammille samalla tavalla toimiville tietokoneille. Tietokoneella voidaan tarkoittaa fyysistä tietokonetta, virtuaalista tietokonetta tai prosessoria.

Kuormanjakoa hallitsee klustereissa verkon reunalla istuva agentti, joka valvoo, että jokainen työyksikkö suorittaa saman verran työtä. Näin resursseja hyödynnetään mahdollisimman tehokkaasti. Kuormanjako on palvelun käyttäjälle tuntumaton prosessi ja usein yhden url-osoitteen takana on jopa satoja palvelimia(Bourke 2001, 14).

2.4.1 Docker

Docker on pääasiallinen ohjelmisto, jolla sovellukset rakennetaan virtuaalikoneiksi, jotka sisältävät vain ohjelmakoodin oikean suoriutumisen kannalta olennaisia riippuvuuksia. Tämä tekee konteista kevyempiä kuin tavallisista virtuaalikoneista tai tietokoneista, sillä niissä on tyypillisesti oletusohjelmia ja muuta käyttöjärjestelmien mukana tulevaa ylijäämää. Kontit ovat eristettyjä, eli niiden toiminta on riippumatonta isäntäjärjestelmästä (Docker 2021.)

Docker tarjoaa mahdollisuuden julkaista sovelluksia alustariippumattomasti virtualisoinnin avulla. Samaa koodia ei tarvitse konfiguroida erikseen Linux- ja Windows-palvelimille, vaan Docker tarjoaa kerroksen alustaan, jonka sisällä ohjelma suoriutuu alustariippumattomasti. (Docker 2021.)

Docker rakentaa ohjelmasta Dockerfilen avulla kuvan, jonka voi ajatella olevan resepti sille mitä ohjelma vähintään tarvitsee kääntyäkseen ja rakentuakseen kontissa suoriutuvaksi sovellukseksi. Kaikki tämä tieto tallentuu kuvaan (eng. image), jonka avulla Kubernetes onnistuu rakentamaan kontin itselleen ajoympäristöön. (Kubernetes 2023b, Containers.)

Konttien kuvia talletetaan konttirekistereihin (eng. container registry), joista ympäristöt voivat noutaa verkon välityksellä kuvan omaan muistiinsa ja siten pystyttää kontin paikallisesti, ilman tarvetta varsinaiselle ohjelmakoodille. Rekisterit voivat sijaita pilvialustoilla kuten *Azuressa*, *Docker Hubissa* tai ne voivat olla yksityisiä repositorioita. (Docker 2021). Kun Docker on kääntänyt kuvan valmiiksi, voi kehittäjä lähettää, eli puskea, kuvan valitsemaansa rekisteriin, eli kuvien säilöntäalustalle, ja lisätä siihen versiointia varten tunnisteita, eli tägejä.

2.4.2 Rekisterit ja niitä käyttävät ohjelmistot

Kun Kubernetes ohjeistetaan käyttämään jotain konttia resurssissaan, tulee resurssin luontitiedostossa, eli YAML-tiedostossa täsmentää, mistä kontin rakentava kuvake (eng. image) haetaan. Kuva toimii ohjeena ja pakettina, jolla Docker-ohjelmisto saa rakennettua eristetyn sovelluksen. Jos kuva haetaan todentamista edellyttävästä rekisteristä, Kuberneteksen pitää osata todentaa itsensä automaattisesti rekisterille, ilman ihmiskäyttäjän puuttumista tapahtumiin.

Tätä varten on olemassa resurssi nimeltä palvelutunnus (eng. serviceaccount), jota Kubernetes voi hyödyntää, kun jokin sen resurssi vaatii pääsyä ulkoiseen, suojattuun, resurssiin. (Kubernetes 2023c). Palvelutunnukseen on kytkettävä erilaisia nimiavaruuskohtaisia tai globaaleja salaisuuksia (Secrets), joihin varsinaiset todentamiskeinot ja arvot talletetaan. (Kubernetes 2023d.) Tätä salaista resurssia voi käyttää esimerkiksi YAML-tiedostossa avaimella *imagePullSecret*, jonka avulla Kubernetes voi hakea kontin konttirekisteristä. (Kubernetes 2023e.)

Joissain rekistereissä tulee olla vastavuoroisesti tieto käyttäjistä ja sen valtuuksista. Jos kyseessä oleva käyttäjä ei ole ihminen, vaan ohjelma, tulee ohjelmalle luoda ja sitoa identiteetti myös konttirekisteriin (eng. container registry). Azuressa tällainen identiteetti on *service principal*, joka vaaditaan, jos ohjelmallinen entiteetti haluaa pääsyn suojattuihin resursseihin (Microsoft 2024b).

2.5 Jatkuva integrointi ja julkaisu - Continuous Integration, Continuous Deployment

CI/CD lyhenne tulee termeistä ”Continuous Integration / Continuous Deployment”, joka tarkoittaa ohjelmiston päivitystä ja sen mukana tuomien muutosten testaamista ja niiden käyttöönottoa sovellusten tuotantoympäristöissä automatisoidusti. Sen tarkoitus on nopeuttaa sovelluskehityksen elinkaarta. (Red Hat 2024.) Yksi CI/CD:n väline on Azure Pipelines, jonka avulla pilvirepositoriossa

voidaan luoda automaatioita, joilla ohjelmia voidaan kääntää, testata ja julkaista. (Microsoft 2024a.)

Pipelinen – eli automaatioputken – avulla voidaan myös tallettaa Docker-kuvia pilviympäristöön.CI/CD:n noudattaminen käyttöönotoissa on yksi opinnäytetyön tavoitteista ja yritykseen tehtävä dokumentaatio perehtyy siihen, miten Azuren Pipeline toiminnolla voi julkaista kontitettuja sovelluksia yrityksen Kubernetekseen automatisoidusti.

3 Kubernetes

Konttien vikasietokyky niiden kevyen luonteen takia on yleisesti heikompi kuin kokonaisen virtuaalitietokoneen (Rensin 2015, 3). Yritysten liiketoiminnan jatkuvuuden kannalta epäonnistumiset ovat ei-toivottuja. Virheistä täytyy elpyä mahdollisimman nopeasti.

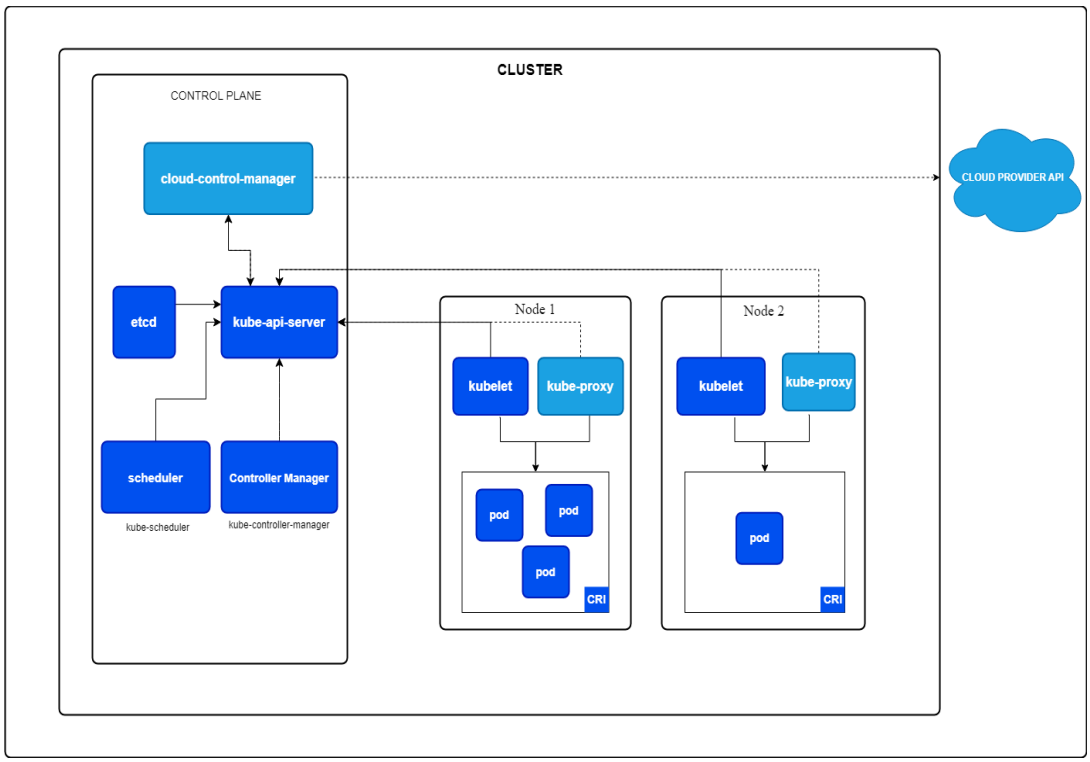
Kubernetes on kontitettuja sovelluksia sisältävän klusterin orkestrointityökalu, jonka keskeinen idea perustuu siihen, että klusterilla on mukautettava tila, jota sen tulee ylläpitää. Käyttäjät, niin ihmiset kuin ohjelmat, vaikuttavat siihen rajapinnan kautta. Sen keskeinen tehtävä on palveluiden saavutettavuuden ylläpitäminen. Ohjelmat tulevat epäonnistumaan, mutta sen ei pidä vaikuttaa palvelun saatavuuteen. Jos podissa – eli Kuberneteksen hallitsemassa, kontteja sisältävässä objektissa – oleva kontti ei suoriudu odotetusti, nostetaan tilalle toimiva podi (Rensin 2015, 11). Tämä siirtää kehittäjien huomion hyödyllisempiin asioihin kuin saatavuuden resetoimiseen, sillä Kubernetes hoitaa palvelun saavutettavuuden ylläpidon itsenäisesti. (Burns ym. 2022, 1.)

Kuberneteksen toiminnasta ja sen mukana tuomista liiketoiminnallista eduista on verkossa paljon materiaalia. Kuitenkaan tiettyyn käyttötarkoitukseen tehdyistä klustereista ja niihin soveltuvien teknologioiden kokoonpanoon ei ole yhtä vastausta, vaan se riippuu täysin yrityksen tavoitteista.

Kubernetes -projektin suunnitteli alun perin Google. Nykyisin sitä ylläpitää maailmanlaajuinen yhteisö, mutta sen tavaramerkin omistaa Cloud Native Computing Foundation (CNCF), joka on Linux Foundationin hankeprojekti konttisovellusteknologian edistämiseksi.

4 Resurssit ja ympäristöt

Kubernetes tarjoaa erilaisia objekteja, joilla voidaan luoda saavutettavia palveluita. Osa objekteista kuuluu Kuberneteksen hallinnalliseen nimiavaruuteen, ja se sisällä voi olla vain Kuberneteksen itse luomia resursseja. Tällaiset resurssit kommunikoivat apipalvelimen kanssa, joka vastaavasti antaa muiden palvelukohtaisten resurssien kommunikoida tilaansa Control-Planeksi kutsutulle vyöhykkeelle. Klusterit ovat ns. Master-Slave kokoonpanossa, joissa roolit määräytyvät siitä, millaisia prosesseja noodit suorittavat (Rensin 2015, 8). Kubernetteksessa Master-nodeja vastuineen kutsutaan Control-Planeksi ja Slave-nodeja vain nodeiksi. (kuva 2).



Kuva 2: Kubernetes cluster architecture *(*Kubernetes 2023f*).*

Etcd on palvelu, jolla voidaan ylläpitää konfiguraatiota replikoitujen järjestelmien välillä. Tämän avulla klusteri voi jakaa tietoa omasta tilastaan muiden järjestelmän komponenttien välillä. (Kubernetes 2023f).

O'Reilly määrittelee palvelun olevan pienen määrän tehtäviä tekevä prosessi, jossa ei ole käyttöliittymää ja joka vastaa pelkästään rajapintakutsuihin. Sovellukset/applikaatiot hyödyntävät näitä rajapintoja ja siten palveluita normaalissa toiminnassaan. (Rensin 2015, 6)

Kuberneteksessa palvelu – **Service**, on objekti, joka pyritään suunnittelemaan niin että se suorittaa vain pienen määrän erilaisia tehtäviä.
Palvelut lähtökohtaisesti ovat podeille tapahtuvan tietoliikenteen solmukohtia, joihin tietoliikenne kohdistuu ja josta yhteys reititetään parhaalle mahdolliselle podille ja siten itse ohjelmaan. Palvelut, kuten muutkin Kuberneteksen resurssit, määritellään YAMLin avulla (kohta 4.10). (Kubernetes 2023g.) Palvelut löytävät niille kuuluvia podeja leimojen, eli **labeleiden** avulla. Podia luodessa niihin kytketään leima, joka altistaa ne näkyviksi palvelulle. Palveluresurssin luonnin yhteydessä tarkennetaan millä leimoilla kyseinen palvelu valikoi itselleen podeja. (Kubernetes 2023g.)

Nodet, myös noodit ja solmut, ovat virtuaalisia tai fyysisiä koneita klusterissa, jotka sisältävät yhden tai useamman podin. Podit suorittavat kontitettuja sovelluksia. Tyypillisesti klusterissa on useita nodeja, joita hallitsee Kuberneteksen apipalvelin, eli rajapinta. Noden kompontentteja ovat kubelet, kubeproxy ja kontin suoritusympäristö – container runtime, joka on tässä opinnäytetyössä Docker. Sen avulla esim. konttien tila näkyy Kubernetekselle. (Kubernetes 2023h).

Kubernetes ei orkestroi resurssejaan konttien tasolla, vaan podien tasolla. Podi käyttää jaettua ip-osoitetta ja siten kontit käyttävät samoja ip-osoitteita podeissa, ja ne voivat kommunikoida keskenään siten käyttäen localhost-aliasta. (Rensin, 10–11, 2015.) Tämä mahdollistaa kätevän tavan liittää podeihin pääsovellusta suorittavan kontin lisäksi sitä tukevia tai toiminnallisuuta laajentavia,

kontitettuja sovelluksia, joita kutsutaan **sidecareiksi**, eli sivuvaunuiksi (Kubernetes 2023i).

Kubelet on se osa nodea, joka vastaa noden näkyvyydestä Apipalvelimelle. Se on podi, joka kuuluu Kuberneteksen sisäiseen nimiavaruuteen, mikä merkkaa sen osaksi Kuberneteksen koneistoa ja se ei hallinnoi podeja, jotka eivät ole Kuberneteksen luomia. (Kubernetes 2023j.)

Kubeproxyn tehtävä nodeissa on ylläpitää palvelun ja podien väleillä tapahtuvia verkkoyhteyksiä. Apipalvelin raportoi resurssinluontien ja muokkausten yhteydessä mahdollisesti tapahtuneet muutokset nodejen kubelet-podeille jotka ottavat kyseiset muutokset käyttöön asianmukaisissa resursseissa. (Kubernetes 2023k).

Podien suoriutumisen edellytys on, että nodeille on asennettu konttien suoritukseen kykenevä ajoympäristö. Tällaisia voivat olla esimerkiksi containerd ja Docker engine. (Kubernetes 2023l.)

Broman Groupissa em. nodeja on klusterissa useampi kappale ja näille ohjataan liikennettä proxy-palvelimen avulla. Proxy-, eli välityspalvelin on yksi kerros osana klusteriin tapahtuvaa tietoliikennettä, joka toimii myös kuormanjakajana. Kuormanjako HAproxyn tapauksessa tapahtuu juuri nodejen välillä.

Välityspalvelin vastaanottaa tietoliikennettä yhdellä niistä nimityksistä, eli domaineista, joka on sidottu verkossa välityspalvelimen IP-osoitteeseen. Pyynnön otsikkodatan avulla välityspalvelin voi päätellä mille klusterin solmulle tietoliikenne ohjataan. (HAProxy, 2024.)

Kubernetes ryhmittelee sen resurssit rajapinnoittain erilaisiin ryhmiin, jotka tunnetaan nimellä **apigroups**. Jokainen Kuberneteksen resurssi kuuluu johonkin ryhmään, mutta kaikki ryhmät eivät ole samanarvoisia. Perusrajapinnan taso, jota sanotaan Coreksi, sisältää ne resurssit, jotka ovat edellytyksiä Kuberneteksen perustoiminnan kannalta. Tällaisia ovat podit, nodet, palvelut ja nimiavaruudet. (Lee, 2023).

Muut ryhmät ovat ns. nimettyjä ryhmiä, jotka sisältävät johonkin tarkemmin määriteltyyn toimintaan kytkettyjä resursseja. Tällaisia nimettyjä ryhmiä ovat mm.: Verkkotoiminta (networking.k8s.io), todentaminen (authentication.k8s.io), sertifiointi (certificiates.k8s.io), tallennustila (storage.k8s.io) ja sovellukset (apps). (Lee, 2023).

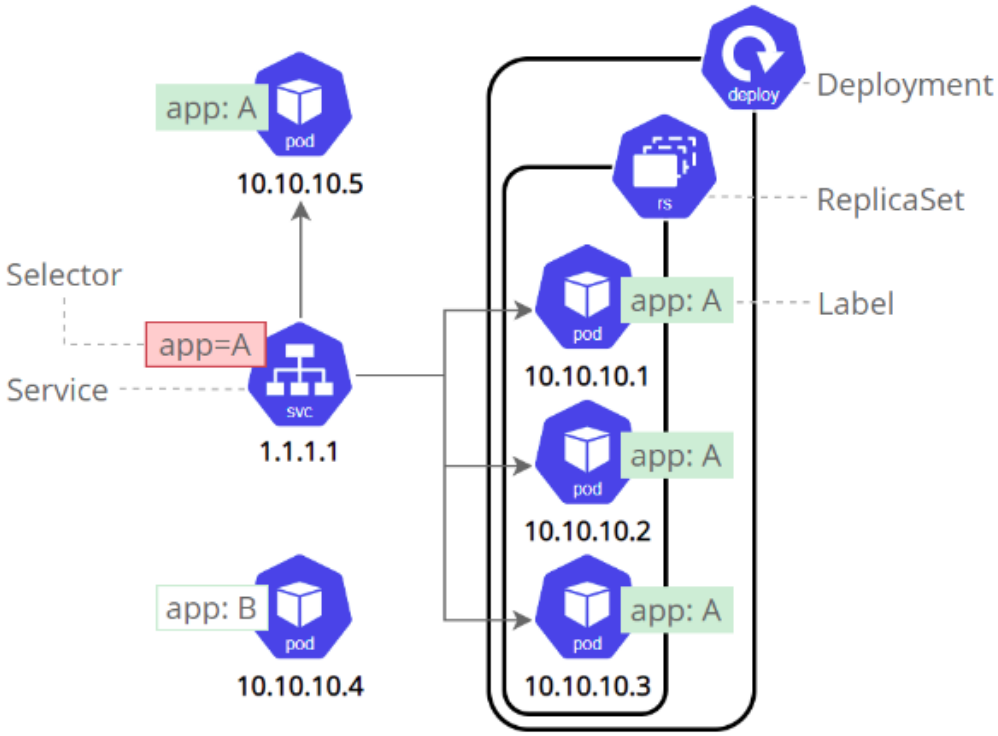
Resurssia luodessa tulee YAML-tiedostoon täsmentää, mille rajapintaryhmälle pyyntö kohdistetaan. Jos rajapintaryhmä on väärä, luonnollisesti Kuberneteksen apipalvelin huomaa, että tämän ryhmän avulla kyseistä resurssia ei voi luoda.

Replikat ovat kopioita podeista, jotka yleensä julkaistaan deployment-resurssin luonnin yhteydessä. Ne ovat olemassa skaalautumisen ja vikasietoisuuden vuoksi (Rensin 2015, 18). Jos podi joudutaan kaatamaan tai se kaatuu, voidaan yhteydet siirtää välittömästi replikalle ja palvelu ei siten seisahdu. Kaatuneen podin korvaa toinen podi. Näin replikoiden määrä pysyy samana (Kubernetes 2023m). Tätä +1/-1 tapaa luoda uusi ja poistaa vanha podi kutsutaan **Rolling Updateksi**, eli vyöryväksi päivitykseksi.

Replikaatiokontrollerien luonti on pääasiallinen tapa kopioida podeja käytännöllisemmin yksittäisellä YAML-tiedostolla, kuin tehdä erikseen yksittäisiä kopioita podeista - jokainen omalla YAML-tiedostollaan. (Burns, ym., 2022.) Replikaatiokontrollerin YAML-tiedostossa kuvataan määrä samaa toimintoa ajavia podeja, jota Kuberneteksen tulee ylläpitää (Kubernetes 2023n). Kopiot edesauttavat korkeaa saavutettavuutta, sillä ne ovat virheellisten podien korvaajia ja kuormanhallinnan päätteitä. (Redundancy). (Kenso, 2024.)

Deploymentit ovat replikaationohjaajia, jotka pyrkivät ylläpitämään tavoitetilaa klusterissa, tämä tarkoittaa tiettyä määrää podeja ja niiden sovellusten versioita. Deploymentissa suoriutuville sovelluksille voidaan julkaista päivityksiä rekisteriin tai palata vanhoihin versioihin, tästä määräytyy sovellusten versio, jota deployment ylläpitää. (Burns, Beda, Hightower 2022, 113.)

Deploymentin tehtävä on saavuttaa YAML-tiedostossa määritelty tila ja siksi se valittiin resurssiksi tulevan rajapintasovelluksen käsiteltäväksi.



Kuva 3: Deployment, jonka podeihin palvelu ohjaa tietoliikennettä. (Kubernetes 2023o).

Replicaset tarkoittaa kokoelmaa identtisiä podeja, jotka luodaan, kun deployment lanseerataan. Kokoelma on olemassa turvaamaan palvelun redundanssia.

Kuberneteksen Control-Plane vyöhykkeelle kommunikoidaan rajapinnan kautta. Tätä varten on olemassa erilaisia välineitä ja koodikirjastoja. Tavallisin niistä, eli **Kubectl.** Se on komentorivityökalu klusterinhallinnointiin. Sen avulla voi komentojen ja flagien – eli lipukkeiden – avulla julkaista Kubernetekseen objekteja ja muokata jo olemassa olevia objekteja. Kubectl:llä voi myös tarkastella klusterin ja objektien nykyistä tilaa. Kubectl:n kaikki komennot käsitellään apipalvelimella. (Kubernetes 2023p.)

Control-Plane on klusterin tilaa hallinnoiva vyöhyke, josta muuta klusteria kontrolloidaan. Täällä pidetään yllä tietoa esimerkiksi nimiavaruuksista. Control-

Plane sisältää vain Kuberneteksen olennaistoiminnan kannalta välttämättömiä resursseja. (Kubernetes 2023q.)

Apipalvelin on klusterin rajapintakutsuja vastaanottava osa klusteria, joka sijaitsee Control-Planella. Apipalvelin vastaanottaa tietoa, eli manifesteja muutoksista klusterille ja välittää toimenpidekutsut orkestraattorin muille komponenteille. Apipalvelimen kautta objektien luomiseen käytetään YAML- tai JSON-formaatin tekstitiedostoja. Tämä manifesti, eli kuvaus resurssista ja sen tilasta, välitetään apipalvelimelle, jossa sitä vastaava objekti rakennetaan klusteriin. (Kubernetes 2023r.)

4.1 YAML-dokumentit

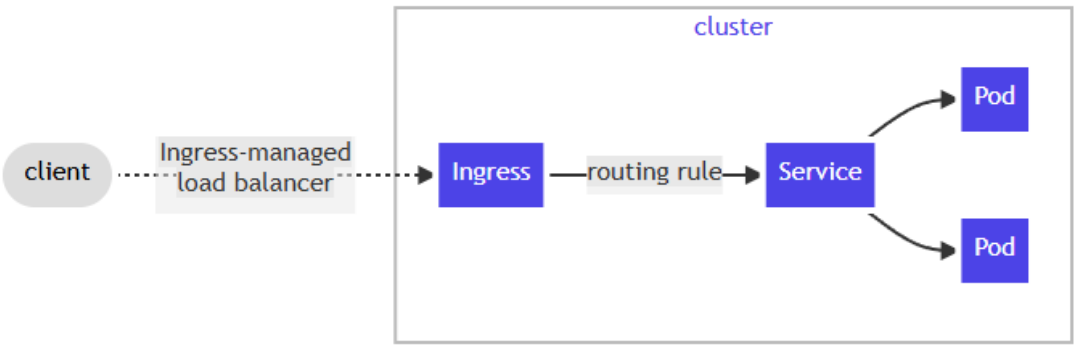
YAML - lyhenne sanoista *Yaml ain't markup language* - on pääasiallinen tapa ilmaista resursseja, joita Kubernetes käyttää. YAML-tiedostot ovat tekstitiedostoja, jotka tarjoavat käyttäjäystävällisen tavan ilmaista erilaisia dataobjekteja. Se soveltuu mm. konfiguraatiotiedostojen, lokien ja objektien rakenteen tallentamiseen. (Ben-Kiki, Evans, Ingerson 2004.)

Kuberneteksen resurssin luodaan pääasiassa YAML-tiedostoilla. Niissä kuvataan jokin Kuberneteksen ryhmitelty resurssi ja kuvataan erilaisin kentin ja sisennyksin tila, jota Kuberneteksen tulee ylläpitää kyseiselle resurssille. Jos Kubernetes ei tunnista YAML-tiedoston määrittelemää objektia sellaiseksi, jota se voi ylläpitää, vaikka syntaksivirheen takia, resurssia ei luoda. Jos kyseessä oli olemassa olevaan resurssiin tehtävä muokkaus, sitä ei tehdä.

4.2 Ingress

Kuberneteksen podit voivat suorittaa esim. HTTP-protokollaan nojaavia web-applikaatioita. Klusterissa on kuitenkin kerroksia ja klusterille saapuvaan liikenteen ohjaukseen verkon reunalta ei riitä yksi ip-osoite. Ingress, eli klusterin

tietoliikenteen ohjausohjelma, ohjaa tietoliikenteen service-objektille, eli palvelulle, ja sen toiminta päättää mille työyksikölle, eli podille, yhteys lopulta päätyy (kuva 3).



Kuva 3: Ingress, (Kubernetes 2023s)

Ingress tarkoittaa säännöstöä, jonka avulla klusterin sisälle pyrkivää tietoliikennettä ohjataan klusterin palveluihin. Se on yksi Kuberneteksen objekteista. Tämän objektin avulla ingress-ohjain voi päätellä, mitä saapuvalle tietoliikenteelle tehdään klusterissa.

Ingress toteutetaan kahdella kokonaisuudella. Ensimmäiseksi tarvitaan ingress-ohjain, joka on tavallisesti ulkoisen palveluntarjoajan kontitettu sovellus, joka asettuu osaksi Klusterin konfiguraatiota, tätä voidaan siten muokata käytännöllisemmin Control-Planen avulla. Toiseksi tarvitaan ingress-objekti, joka on säännöstö reititykseen erilaisten pyyntöjen kohdalla. Ingress-objekti voidaan luoda samalla tavalla YAML-tiedostolla, kuin muutkin Kubereneteksen resurssit. (Kubernetes 2023s.)

Ingress-ohjain voi tehdä liikenteen ohjausta ingress-objektissa määritetyn polun tai hostin nimityksen avulla. (Kubernetes 2023t). Esimerkiksi *domain.com* voidaan määrittää ohjaamaan saapuva tietoliikenne asiakkaille tarkoitetulle palvelimelle, jossa voi ostaa tuotteita. Kun taas *domain.com/profile* ohjaa tietoliikenteen toiselle klusterin palvelimelle, joka erikoistuu käyttäjäprofiilien hallintaan.

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx-example
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

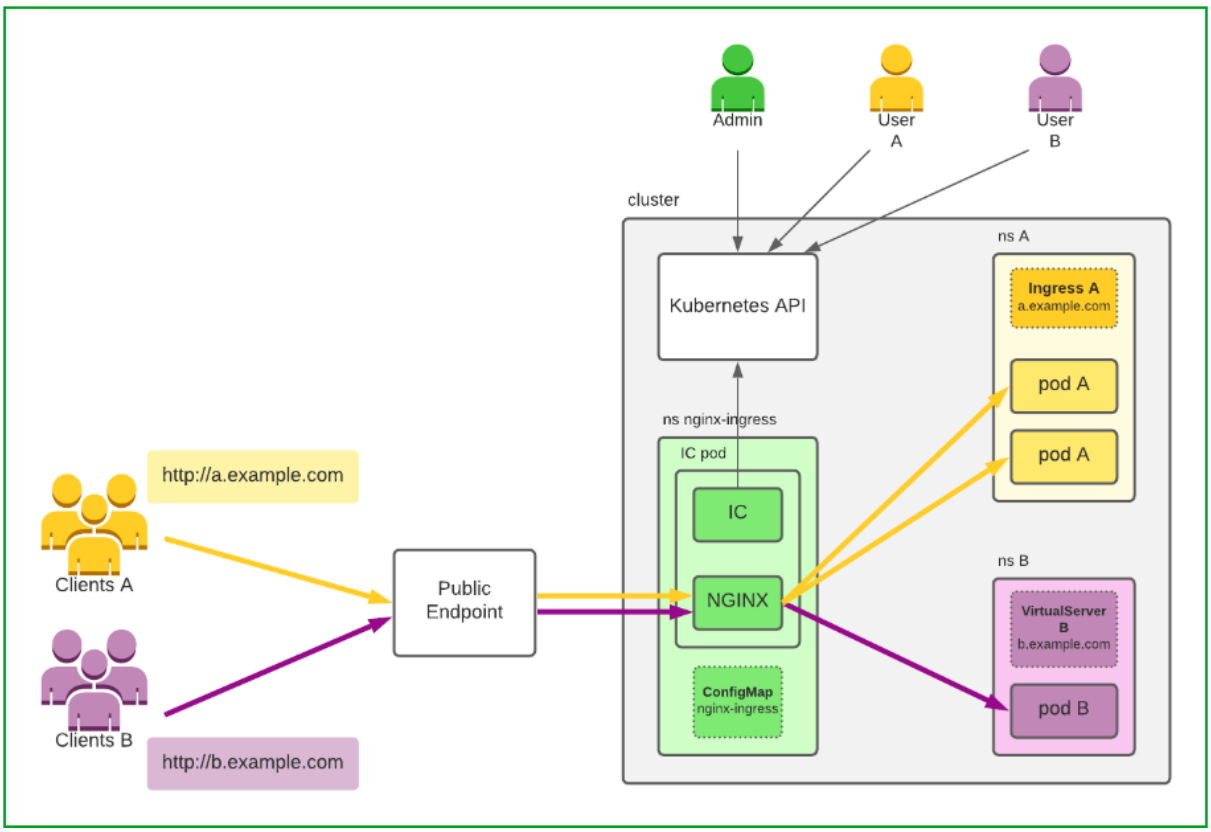
Kuva 4: Ingress -YAML-specification. (Kubernetes 2023s)

4.3 Ingress- ja loadbalancer-teknologioita

Suosittu ingressin toteuttaja on NGINX. NGINX-ingress controller tukee WebSocket, gRPC, TCP ja UDP protokolliin suunniteltuja sovelluksia. (NGINX 2024, About). Se on F5 – yrityksen ylläpitämä klustereiden kuormanjakoa ja ingressiä varten

Yrityksessä ingress-ohjain on toinen kerros klusteriin kohdistuvan tietoliikenteen ohjausta. Edeltävä proxy-palvelin ohjaa domainiin kytketyn liikenteen yhdelle klusterin noodeista, jossa ingress-controller kuuntelee valittua porttia.

Vasta kun liikenne saapuu proxylta ingress-ohjaimelle, klusteriin tehtyjen ingress-resurssien säännöstö astuu voimaan, ei ennen. Seuraa ohjaus pavelu.objektille ja siten sovellusta suorittavalle podille. (Kubernetes 2023t.)



Kuva 5: NGINX ingressin toiminta, nginx.org (Nginx documentation 2021)

Toinen tunnettu loadbalancer-tarjoaja on MetalLB. Joka tarjoaa kuormanjaon toteutusta ”bare-metal”-klustereille, koska Kubernetes ei tätä oletusarvoisesti tee. Se jakaa podeille osoitteita tuntemastaan kokoelmasta, eikä rakenna niitä ”tyhjästä” pilvipalvelumaisesti. (MetalLB Universe 2021). MetalLB:tä ei otettu käyttöön tässä projektissa.

4.4 Azure Pipelines

Azure Pipelines on Microsoftin kehittämä, Azuren DevOps pilvipalvelun tarjoama, keino automatisoida ohjelmakoodin kääntämistä, testaamista ja käyttöönottoa eri ympäristöissä. Tällainen ympäristö voi olla Kubernetes-klusteri. (Microsoft 2024a). Koodimuutosten päivittäminen versionhallintaan laukaisee projektin versionhallinnan haaraan kytketyn automaation, joka suorittaa askeleittain erilaisia tehtäviä, täyttäen automaatioiden eri vaiheita (stages), tarkoituksenaan saavuttaa onnistunut käyttöönotto (deployment). (Microsoft 2024a.)

Kuten Kuberneteksen resurssit, määritetään automaatioputken toteutus YAML-tiedostolla tai vaihtoehtoisesti DevOpsin käyttöliittymän avulla. (Microsoft 2024a.)

Azure tukee Pipeline-YAML-tiedostoilla toiminnallisuutta nimeltä *Templates*. Se on tapa parametrisoida automaatioputken *Stageja, Jobeja ja steppejä* helposti uudelleen käytettäviksi kokonaisuuksiksi (Microsoft 2024d). Templaattien avulla vaikeastikin ymmärrettävä YAML-rakenne voidaan kääntää parametrisoiduksi ”pyynnöksi” ja ne ovat tämän projektin automaatioputkien konfigurointien luettavuuden kannalta ratkaisevia.

YAML-dokumentin lähettävässä Pipeline-taskissa tulee olemaan upotettua YAML-syntaksia, jonka tekstinmuotoilu on prosessin kulun seuraavassa vaiheessa tapahtuvan deserialisoinnin kannalta tärkeää. YAML-elementtejä sisennetään toiseen elementtiin välilyönnin määrillä ja ”kopioi – liitä” -tavalla toisen projektin automaatioputkeen lisätty ratkaisu ei välttämättä toimi heti ilman hienosäätöä. Templaatit voi keskittää yhteen repositorioon azuressa ja siten niitä voi käyttää ilman omaa hakemistoa projektissa (Microsoft 2024d).

5 Toteutus

Suurin osa opinnäytetyöprosessia oli kokonaisuuksien ja eri teknologioiden testaamista, niin että Kuberneteksen resurssien, proxyjen ja oheistyökalujen käyttö sovelluskehityksen näkökulmasta oli kasvavan ymmärryksen puitteissa mahdollista. Opinnäytetyön tavoitteena oli klusterin käyttöönoton ohessa myös dokumentaatio siitä, kuinka klusteriin voidaan julkaista sovelluksia. Ennen kuin dokumentaatio voitiin kirjoittaa, infrastruktuuri täytyi saada toimimaan.

Klusteriin tarvittiin sisälle virtaavan tietoliikenteen takia ingress-ohjain, joka asennettiin Helm-työkalulla. Helm on Kuberneteksen työkalujen asennusohjelmisto. Ohjelmistolla asennettiin ohjainkonfiguraatio, joka asensi jokaiselle klusterin noodille ennalta valittua porttia kuuntelevan instanssin ohjainohjelmistoa

suorittavia podeja. Seuraavaksi oli rakennettava nimiavaruuskohtaiset säännöstöt tuleville ohjelmistoille, joihin tiedettiin tapahtuvan tietoliikennettä.

Ingress-sääntöjen luonti oli yksinkertaista, mutta itse välityksen käyttäytymistä jouduttiin konfiguroimaan. Yhdeksi ongelmaksi ilmeni liian suuret HTTP-kutsut. Ongelma ratkaistiin kasvattamalla ohjaimen proxy-bufferin kokoa, jotta koko HTTP-kutsu suurenkin otsikkodatan kanssa pystyi siirtymään eteenpäin.

Kun ingress toimi, voitiin klusterin sovelluksia käyttää ja tarkastella HTTP:n avulla. Käyttöliittymättömät sovellukset voitiin testata curl-kutsuilla. Seuraavaksi Kuberneteksen apipalvelimelle tuli pystyä kommunikoimaan ulkoverkon kautta, mutta suora yhteys Control-Planelle ulkoa automaatioiden kautta olisi liikaa.

Yrityksessä ei käytetty ulkoverkolta eristettyä väylää pilvestä paikallisverkkoon, joten vastaavaa toiminto toteutettiin Frends-integraatioalustan ja sen useiden agenttien mahdollistamalla *Remote Subprocess Call* toiminnolla. Sen avulla pilvessä suoriutuva pääprosessi voi kutsua paikallisessa verkossa, eli eri ympäristössä, suoriutuvaa aliprosessia. (Virtanen 2024.)

Frends-prosessi rakennetaan BPMN-notaatiota hyödyntävällä graafisella Low-Code-ohjelmointipinnalla. Prosessi aloittaa suorituksen pilven rajapintakutsun yhteydessä ja parsii kutsun bodysta dataa aliprosessia varten.

Teknisen toteutuksen jälkeen tuli tietää minkälaisella tavalla sovellukset voisivat päätyä yrityksen Kubernetekseen mahdollisimman käyttäjäystävällisesti. Hyväksi tavaksi ilmeni integraation ja api-sovelluksen kautta välittyvä kutsu Kubernetekseen. Seuraava vaihe oli tehdä rajapinta. Rajapintaan tuli kommunikoida integraatioalustan prosessista, joka taas suoriutui oman rajapintakutsunsa avulla.

Rajapintasovelluksen ehtoja olivat se, että sen kautta piti pystyä testaamaan resurssien luontia mahdollisimman käytännöllisesti esim. curl tai insomnia -työkaluilla, joiden avulla voi tehdä REST-kutsuja verkon päätteisiin. Aluksi

ohjelmointikielenä oli JavaScript ja tällä tuotettiinkin prototyyppi sovelluksesta, mutta myöhemmin soveltuvammaksi kieleksi päätettiin C#.

Rajapintasovellukselle kommunikointi suoraviivaisesti edellyttäisi Azuresta express-väylän luontia paikalliseen sisäverkkoon, mutta vastaavanlaisen toteutuksen pystyi järjestämään Azuren *service connection*-yhteydellä Frendsin pilvialustaan, josta aliprosessitoimintojen avulla päästiin kutsumaan suorituksia sisäverkon agenteilla. Näin taattiin pääsy pilvestä paikalliseen verkkoon, jossa rajapintasovellus suoriutui.
Kun järjestelmän osat löysivät paikkansa, sovelluksia rakentui sisäverkkoon ja automaatiot onnistuivat tarpeeksi yleisellä tavalla. Oli turvallista kirjoittaa ohjeistusta sisäverkon foorumille muita kehittäjiä varten.

Dokumentaatio muuttuu kehitystyön jatkuessa, mutta sen tulisi ilmentää aina ajantasaista tilaa yrityksen valitsemasta rakenteesta, johon sovelluksia julkaistaan. Nimiavaruudet, Secretit, julkaistut deploymentit ja niiden domainit tulevat esille dokumentaatiosta. Ennen kaikkea dokumentaatiota pitää pystyä käyttämään niin, että kehittäjä voi liittää Azuren automaatioputkeen stagen, jonka avulla onnistuneen kontituksen jälkeen voidaan julkaista sovelluksen eri versioita Kubernetekseen, korvaamaan aiemman version. Käytännössä automaatioputkella vaihdetaan deploymentia määrittelevästä YAML-muuttujasta kontin kuvan osoite tägeineen.

6 Tuotokset

Osana opinnäytetyötä syntyi useita erilaisia tuotoksia: yksi rajapintasovellus, Frends-integraatio jokaiselle rajapintasovelluksen päätteelle, näihin kytkeytyvä automaatioputki ja dokumentaatio/ohje. Nämä kaikki mahdollistivat sovelluksen julkaisun versionhallinnasta Kubernetekseen asti. Tuotokset toteutettiin em. järjestyksessä, mutta niihin on tapahtunut kehitystä sekalaisesti ja tuotokset ovat jatkossa alttiita muutoksille.

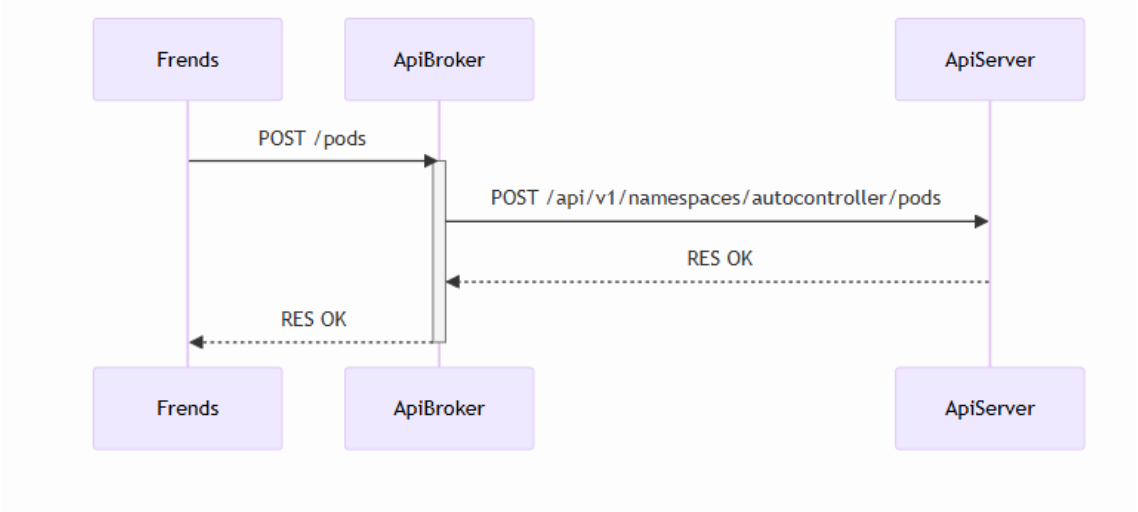
6.1 ApiBroker

ApiBroker on termi, jota käytin kuvatakseni aluksi nodeJS:llä ja myöhemmin C#-lielellä ohjelmoitua rajapintasovellusta. Rajapinnalla tarkoitetaan ohjelmallista pintaa, jolle kommunikoidaan jonkin protokollan avulla ja rajapinnan tehtävänä on tulkita sille saapuvia kutsuja ja aktivoida niitä vastaaviin päätteisiin kytkettyjä toimintoja.

Sovellus vastaanottaa HTTP-kutsuja ja parsii niistä jälleen kutsuja Kubernetek-sen apipalvelimelle. Sovellus toimii pisteenä, johon on helppo päästä useasta eri palvelusta, mutta samalla autentikoida ja valtuuttaa nämä palvelut Kuberne-tekseen.

ApiBroker kykenee vastaanottamaan YAML-muotoista dataa, sillä siinä on kus-tomoitu serialisoinnin ja deserialisoinnin toteutus. Serialisointi tarkoittaa objekti-muotoisen tiedon muutosta muotoon, jossa se pystyy liikkumaan ja säilymään järjestelmissä ja niiden välillä. Deserialisointi tarkoittaa serialisoidun datan ra-kentamista takaisin ohjelmisto- ja kielikohtaisiksi objekteiksi niiden saapuessa sovelluksille. (Microsoft 2024c)

ApiBroker käyttää KubernetesClient nimistä kirjastoa HTTP-kutsujen kuormassa saapuvien YAML-resurssien käsittelyssä. Kirjasto parsii saapuvasta teksti-datasta kirjastonsa mukaisia objekteja ja lähettää nämä resurssit rajapintaryh-mitellyllä HTTP-asiakasobjektilla apipalvelimelle eteenpäin. Kirjasto otettiin osaksi projektia, sillä se helpotti apipalvelimelle tehtävän yhteyden muodosta-mista ja tarjosi deserialisointifunktiot valmiina. Kuvassa ApiBrokerin ensimmäi-nen versio POST-kutsun toiminnasta.



Kuva 6, Integraatioalustan, ApiBrokerin ja Kuberneteksen apipalvelimen HTTP-kommunikointia kuvaava sekvenssikaavio. (Broman Group 2024)

ApiBrokerin idea ei alustavasti ollut toimia kehitys-, testi- tai tuotantoympäristöissä jokaisessa omassa portissaan, vaan se oli tässä kohtaa tietoinen siitä, missä ympäristössä suoriutuu ja tulkita onko sille tehdyllä kutsulla valtuudet suoriutua loppuun.

6.2 Frends-prosessi

Frends on integraatioalusta, joka on suunniteltu erilaisten palveluiden välisen kommunikaatiokuilun täyttämiseen, toisin sanoen integraatioiden luontiin.

Rajapinnalle tietoa välittävän Frends-prosessin luonti aloitetaan luomalla se alustalle, johon pääsee käsiksi myös Azuren automaatioputken avulla, eli prosessi luotiin pilviagentille. Agentti tarkoittaa sitä osaa ohjelmistoa, joka suorittaa varsinaisen prosessin koodin valitussa ympäristössä koko infrastruktuurissa. (Frends 2024.)

Ongelmaksi nousee prosessin olo yhdellä agentilla, eli pilvessä, joka ei tunne samoja verkkoyhteyksiä kuin paikallinen agentti, joka suoriutuu samassa infrastruktuurissa kuin missä klusteri sijaitsee. Tapa ratkaista tämä remote-subprocess-call, joka on toiminto, jossa prosessi käyttää Azuren väylää paikalliseen infraan suorittaakseen toiminnon eri infrastruktuurissa. Aliprosessi siten saa

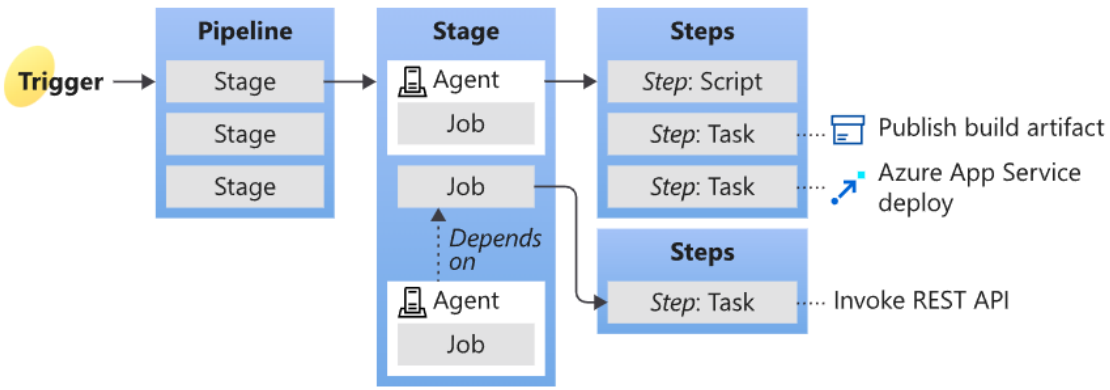
yhteyden firman sisäiseen rajapintaan, joka on yhteydessä sisäverkon kubernetekseen.

Sisäverkon tasolla aliprosessi välittää YAMLin tavalliseen tapaan HTTP-kutsulla ApiBrokerille ja jää odottamaan HTTP-koodia, jotta voi itse lähettää vastauksen ketjussa taaksepäin aina automaatioputkelle asti. Frendsin näkymän kautta kehittäjät voivat tarkastella myös minkälaisia virheitä Kuberneteksen ja Azuren välissä on voinut tapahtua, sillä Frends palauttaa ApiBrokerilta saadut virheviestit luettavaksi. Lisäksi integraatioon voi jatkossa lisätä laajempaa virheenkäsittelyä.

Frendsin integraatio oli välttämätön, sillä käyttöönotot eivät voineet tapahtua HTTP-kutsulla muuten. Kuitenkin se tuo kokonaiskuvaan yhden kerroksen lisää ja sitoo integraatioalustan keskeiseksi osaksi julkaisuputkea, mikä valitettavasti tarkoittaa, että jos Frends on alhaalla, käyttöönotot eivät onnistu.

6.3 Azure Pipelines -automaatio

Microsoft Azuressa on toiminnallisuus, jonka avulla voi laukaista automaatioita koodimuutosten yhteydessä pilvirepositorioon. Nämä automaatiot tyypillisesti kääntävät, testaavat ja julkaisevat ohjelmiston johonkin ympäristöön. Automaatiot on paloiteltu monitoroinnin ja niiden rakentamisen kannalta erilaisiin kerrostettuihin vaiheisiin. Tällaisia ovat hierarkiassa alaspäin staget, jobit ja stepit (kuva 7). (Microsoft 2024a).



Kuva 7, Pipeline vaiheineen (Microsoft 2024a)

Stage kuvaa automaatiossa yleistä vaihetta, kuten testaaminen, jonka yhteydessä joudutaan suorittamaan pienempiä vaiheita, kuten parametrien välittämistä ja tulosten vertailua. Nämä kaksi vaihetta kuvataan jobeina. Jobit ovat taas pilkottu step -nimisiin vaiheisiin, jotka ovat tarkkoja komentoja ohjelmalle. Onnistuneet stepit tarkoittavat onnistuneita jobeja, ja onnistuneet jobit tarkoittavat onnistuneita stageja (Microsoft 2024a.)

Broman Groupin pilvialustalla Azuressa voi versionhallinnan muutosten yhteydessä aktivoida CI/CD mukaisia automaatioita, jotka suorittavat em. vaiheittaisia tehtäviä. Yksi näistä tehtävistä on ohjelman Docker-kontittaminen ja seuraava taso sen käyttöönotto yrityksen Kubernetes-klusterissa.

Käyttöönotossa hyödynnetään em. ApiBroker -ohjelmaa. Pipeline-automaatioputken viimeinen Stage on ohjeistettu lähettämään web-pyyntö Frends-integraatiopalvelulle, joka puolestaan kommunikoi ApiBroker-sovelluksen kanssa ja pystyttää ympäristöönsä viimeisten muutosten mukaiset docker-kontit aiemman deploymentin tai podin käyttämien konttien tilalle. Kubernetes resurssi, jonka automaatioputken halutaan pystyttävän, voidaan upottaa YAML-formaatissa samaan YAML-tiedostoon, jolla automaatioputki määritellään. Tämän mahdollistaa ApiBrokerin kyvykkyys vastaanottaa YAML-kuormia web-pyynnöiltä.

```yaml
steps:
- task: InvokeRESTAPI@1
  inputs:
    connectionType: 'connectedServiceName'
    serviceConnection: 'FrendstTestConnection'
    method: 'POST'
    waitForCompletion: 'false'
    headers: '{"Content-Type":"application/yaml", "X-API-KEY":"[redacted]}'
    body: '$(yamldoc)'
```

Kuva x: YAML-kuvaus Azure Pipelinen stepistä.

Deploymentin päivitys tapahtuu rolling update -strategialla, joka tarkoittaa, että deploymentin podeihin tapahtuvat kuvapäivitykset astuvat voimaan asteittain. Tämä pitää palvelun saavutettavana, vaikka sitä päivitettäisiin.

6.4 Edellytyksiä Kubernetekseen julkaisulle

Jotta automaatio voi julkaista sovelluksen paikalliseen Kubernetekseen, on tätä varten suoritettava ohjelmakoodin rakentaminen, kuvan rakentaminen dockerilla sekä kuvan pusku rekisteriin. Kaikki vaiheet voidaan suorittaa Azuren automaatioputken avulla. Vikatilanteissa Azure tarjoaa vaihtoehdon suorittaa virheeseen menneet staget uudestaan, jotta kaikkia automaation vaiheita ei tarvitse laukaista versionhallinnan muutoksella uudestaan. (Microsoft 2024a.)

Alustavasti kannattaa kehittää ohjelmistoa johonkin versioon asti, tehdä automaatiot ohjelman rakentamista ja kontitusta varten ja testata konttia paikallisesti. Kun ohjelma on todettu toimivaksi kontissa, voidaan siirtyä harkitsemaan sille parasta ympäristöä. Jos ympäristöksi päätetään yrityksen paikallinen Kubernetes, voidaan automaatioputkeen liittää jatkoksi tuota tehtävää varten kustomoitu Stage.

7 Arviointia

Opinnäytetyön tavoitteet saavutettiin, mutta tuotokset ovat alttiita parannuksille tai ne voidaan parempien vaihtoehtojen puitteissa ottaa pois käytöstä kokonaan.

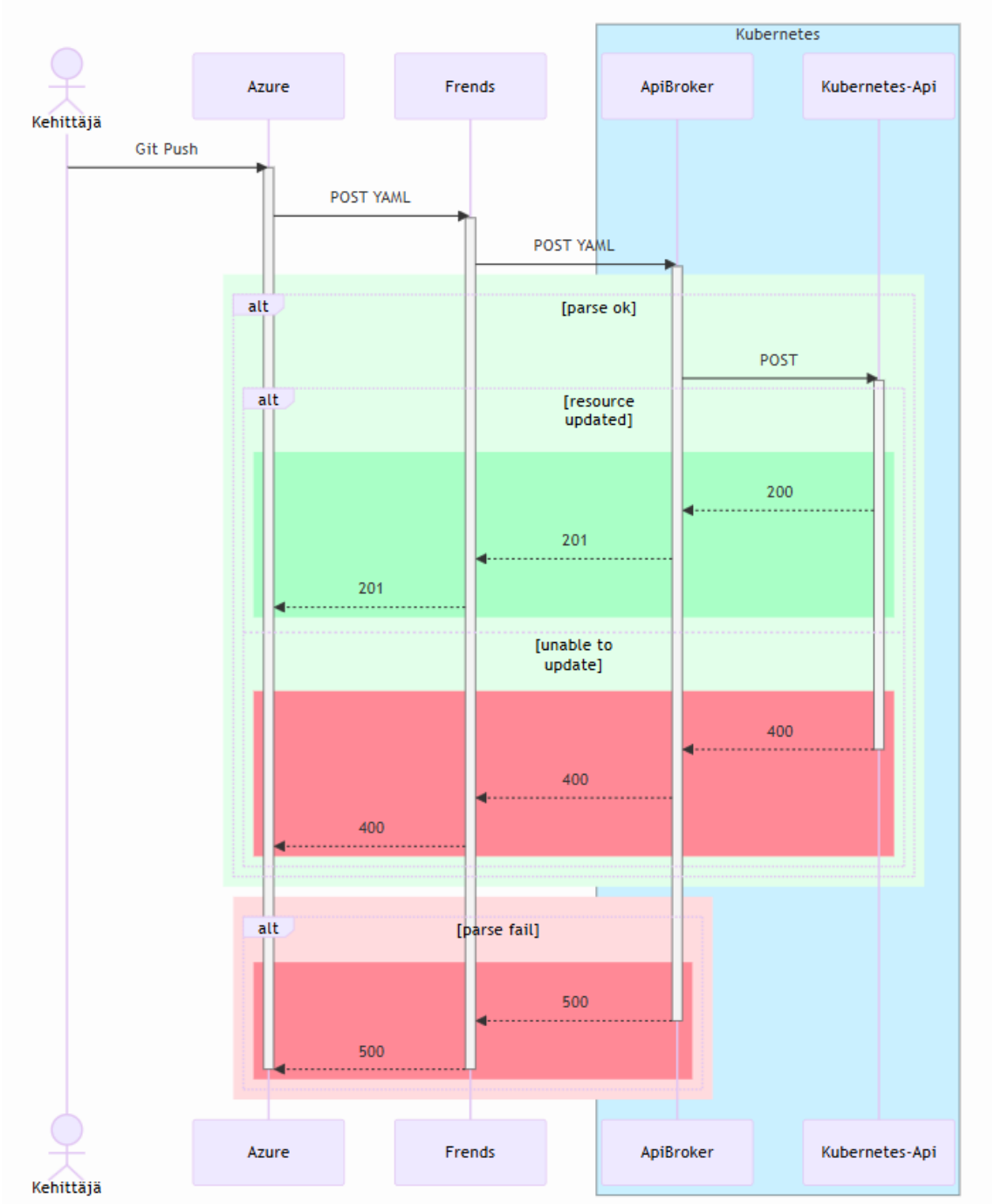
Dokumentaatio tällä hetkellä antaa kehittäjälle kaiken tarpeellisen informaation siitä, mitä hänen tulee Azuressa tai projektihakemistossa tehdä, jotta ohjelma voidaan julkaista sisäiseen Kubernetekseen. Kuitenkin sovelluskehittäjiä on erilaisia ja heidän tuotostensa soveltuvuutta sisäverkon Kubernetekseen tulisi harkita aina tapauskohtaisesti.

Jatkossa arkkitehtuuriin voidaan ottaa lisää mallia Azuren Kubernetes Servicestä (AKS), sillä yrityksessä on kyseisen teknologian erikoisosaajia. Olisi arvokasta, jos heidän kykynsä kääntyisivät käytettäviksi myös sisäverkon klustereihin.

7.1 ApiBroker

ApiBroker projekti oli arvokas projekti kehitykseni kannalta, sen kautta tutustuin tavoitteellisesti Kubernetekseen. Laajennusten käyttäminen ja koodin tuottaminen vaativat käytännön osaamista ja siten opetti Kuberneteksen eri osa-alueita ja sen ympärillä toimimista tehokkaasti.

Kuberneteksen valmiin apipalvelimen olemassaolo herättää kysymyksen välikäden tarpeellisuudesta, mutta oppimismielessä projekti oli arvokas. Suora kommunikointi paikalliseen Kubernetekseen vaatii suurempia rooleihin perustuvien valtuutuksien ja tunnistautumisresurssien rakentamista Kubernetekseen. Tämän takia ApiBrokerin tehtävä portinvartijana helpotti kehitystä, sillä se kavensi Kubernetekseen tehtävien resurssien määrää ja siten myös nopeutti varsinaisen julkaisuputken käyttöönottoa. Vähemmällä luotavien resurssien määrällä myös selvennettiin muille kehittäjille selkeä piste, eli rajapinta yrityksessä, jonka kautta he voivat julkaista sovelluksensa yrityksen Kubernetekseen, Azuren automaatiolla tai ilman. Kehittäjän versionhallintaan tehdyn muutoksen myötä laukaistua prosessia kuvataan alla (kuva 7).



Kuva 8, tämänhetkinen tapahtumaketju versionhallinnan muutoksen yhteydessä.

Tällä hetkellä ApiBroker suoriutuu yksinäisenä podina klusterissa ja ei itse noudata korkean saavutettavuuden periaatetta, mutta sen kautta voidaan luoda korkeasti saavutettavia palveluita. Kuitenkin ApiBrokerin ollessa yksittäinen podi, eikä korkeasti saavutettava palvelu, se luo toistaiseksi Single-Point-Of-Failuren käyttöönottoputkessa. Se johtuu siitä, että ApiBroker on ainoa klusterin

nimiavaruuksiin johtava käyttöönottoväylä Azuresta, jota ei ole toistaiseksi replikoitu kaikkiin ympäristöihin.

Jatkossa ApiBroker-sovellus voidaan ottaa pois käytöstä, sikäli kun Kubernetekselle kommunikoivat palvelut listataan ja näille osataan roolittaa asianmukaiset valtuudet klusteriin. Tämän tapahtumista edesauttaa se, että apipalvelimen kohdalla kyseessä on REST-rajapinta. Kaikki rajapinnalle pyrkivä liikenne voidaan ohjata jatkossa ympäristömuuttujien muutoksilla muualle ja irtiotto voi tapahtua asteittain ja ympäristökohtaisesti.

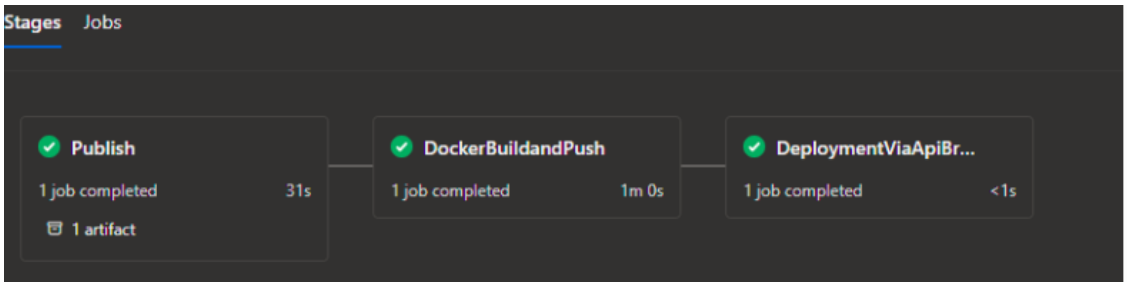
Jatkokehityksellisiä teesejä projektin kohdalla voivat olla ”ApiBrokerin mukauttaminen mikropalveluarkkitehtuuriin”, jossa tutkitaan miten ApiBrokerin voitaisiin sopeuttaa hajautetumpiin järjestelmiin tai tutkia kilpailevia käyttöönottosovellusvaihtoehtoja ApiBrokerille, jos siitä halutaan luopua.

7.2 Erilaisia automaatioputken kautta julkaistuja ohjelmia

Ohjelma voi olla itsenäinen tai ihmisen käyttämä. Jos sovelluksen käyttö vaatii selainta, vaatii Kubernetes myös uuden ingress-säännön juuri tuon sovelluksen tietoliikennettä varten.

Opinnäytetyön aikana sisäverkon Kuberneteksen klusterin hallinointitietokoneeseen muodostetaan etäyhteys. Laitteella rakennetaan Linux-ympäristössä kubectl:n avulla kaikkien tietoliikennettä edellyttävien resurssien pohjat. Näitä ovat vähintään: deployment-objekti, service-objekti ja ingress-objekti. Podit rakentuvat Deploymentin määritelmän mukaan. Kun resurssit ovat luotu ja yhteydet ovat varmistettu, voidaan tehdä kytkös Frends-prosessin ja Azuren automaatioputkien välillä. Vaihtoehtoisesti ApiBroker rajapinnan toimivuutta voi testata REST-työkaluilla kuten Curlilla, Postmanilla tai Insomnialla. Tämä on suotavaa varsinkin niissä tapauksissa, kun liitetty projekti tai deployment ovat uusia. Testaaminen rajapintaan on helpompaa siihen luodulla työkalulla kuin integraatioalustan näkymästä.

Edellä mainittu ingress ja service, eli palvelu, ovat resursseja, joiden avulla voidaan muodostaa yhteys klusterin sovelluksiin, mutta kaikki sovellukset eivät tarvitse näkyvyyttä klusterin ulkopuolelle. Esimerkiksi kontissa suoriutuva robottiautomaatio ei tarvitse tai välttämättä edes tarjoa itsestään sellaista rajapintaa, jotta sitä varten kannattaisi rakentaa saapuvan liikenteen resursseja Kubernetekseen. Sen tuotoksia ja työskentelyä voi tarkastella esim. hakemistoista ja erilaisista lokeista. Toisin on web-sovellusten kohdalla, ne vaativat, että selaimella pääsee käsiksi sovellukseen. Ennen automaatioputken konfiguroimista Kubernetekseen julkaisua varten, kannattaa rakentaa sovelluksen tietoliikenteen säännöstö ensiksi, jotta onnistuneen automaation jälkeen lopputulosta pääsee tarkastelemaan esim. selainnäkymästä (kuva 8).



Kuva 9: Onnistuneen automaatioputken staget.

8 Pohdintaa

Alustavasti pyrin kirjoittamaan yritykselle dokumentaatiota, josta nostetaan segmentoidusti tietoa opinnäytetyötä varten. Tämä tapa kuitenkin osoittautui kehitystyön ohessa haastavaksi, sillä komponenttien teknologioita jouduttiin vaihtamaan useasti. Käytännöllisempää oli tehdä eri komponentit toimiviksi ja sitten kirjoittaa dokumentaatio siitä, millä tasolla klusteri on ja kertoa lyhyesti em. komponenttien rooli. Vasta sitten kerrotaan, miten sovelluksia julkaistaan. Tämä vähensi työn määrää, sillä kirjoitettua dokumentaatioita ei tarvitse editoida ajan tasalle kehitystyön elävyyden seurauksena.

Tuotetut teknologiat kävivät läpi erilaisia versioita, jotka toimivat testitapauksissa. Palvelut eivät välttämättä käyttäytyneet oikein, vaikka kontit syntyivät.

Erilaiset sovellukset käyttäytyvät replikoiduissa ympäristöissä eri tavalla. Tämä huomattiin istuntoja hyödyntävän websovelluksen kanssa. Deployment jouduttiin tekemään vain yhdelle kontille, sillä Kubernetes saattoi ohjata tulevat pyynnöt myös toisille replikoille, joissa tuota istuntoa ei ole olemassa. Tämä aiheutti XSRF-Token virheen, sillä kyseinen artikkeli luodaan asiakkaan ja sen podin välille silloin kun yhteys aloitetaan ensimmäisen kerran. Tämä kuitenkin tarjoaa jatkokehitystä varten tavoitteita. Esim. jatkossa tavoite pitää klusteri mahdollisimman tilattomana, mutta kuitenkin ylläpitää istuntoja sovelluskohtaisesti.

Kuberneteksessa suoriutuvissa podeissa voidaan myös ajaa testejä, mutta HTTP-yhteyden ylläpitäminen niin, että testit voidaan raportoida vastauksena, vaatii jokaisessa arkkitehtuurin tasossa timeout-arvojen – eli arvon, jonka loputtua yhteys katkaistaan – kasvattamista. Tässä arkkitehtuurissa proxy-palveluita on 2, integraatiosta lähteviä kutsuja 2 ja riippuen clientista, senkin timeout-määrää saatetaan joutua kasvattamaan. Jos testien lukumäärät tai kompleksisuus kasvavat, se tarkoittaa vähintään 5 eri konfiguraation muokkaamista yhtä testitapauskutsua varten. Tätä manuaalityötä voi varmasti vähentää jatkossa.

Kuberneteksen käyttöönotto ja sinne sovellusten julkaisu onnistuivat ja uusien palveluiden lisääminen sinne voi tapahtua järjestelmällisellä tavalla opinnäytetyön tuloksena. Uutta palvelua varten vaaditut resurssit voidaan kartoittaa ja dokumentaatiota voidaan kasvattaa, jos uudet palvelut vaativat jatkokehitystä juuri Kuberneteksessa suoriutumista varten.

Opinnäytetyö oli ennen kaikkea opettavainen. Kubernetekseen tutustuminen ja ohjelmakoodin tuottaminen sille kommunikoivaa sovellusta varten juurruttivat keskeisiä hajautettujen järjestelmien konsepteja ja projekti toi aina uusia haasteita. Jatkokehitän opinnäytetyön aikana tuotettuja sovelluksia ja tutustun Kubernetekseen entistä enemmän, nyt kun sitä voi käyttää yhtenä julkaisualustana yrityksessä ja toivon että jatkossa muut kehittäjät tekevät samoin.

Lähteet

Ben-Kiki, O. Evans, C. Ingerson, B. *YAML ain't markup language*, 2004

Bourke, T. Server Load Balancing, O'Reilly, 2001

Burns, B. Beda, J. Hightower, K. *Kubernetes Up & Running,* O'Reilly, 2022

Docker, https://www.docker.com/resources/what-container/. Docker 2021. 30.5.2024

Frends. https://frends.com/. HiQ 2024. 5.11.2024

Galkin, O. Frends Key Concepts. https://docs.frends.com/en/articles/2188944-key-concepts-in-frends., HiQ, 2024. 10.11.2024.

Gray, J. Siewiorek, DP. *High-availability Computer Systems,* 1991

HAProxy documentation. https://docs.haproxy.org/. HAProxy 2024. 11.9.2024.

IBM. https://www.ibm.com/think/topics/high-availability. 2024. 6.11.2024

Kanso, A. The Four Pillars of High Availability https://alikanso.medium.com/the-four-pillars-of-high-availability-d41c1609e0ba, Medium 2021. 15.9.2024

Lee, C. Kubernetes: Core and Named Api Groups. https://yuminlee2.medium.com/kubernetes-core-and-named-api-groups-d9e7fadb0d17. Medium 2023. 27.9.2024

Kubernetes documentation. https://kubernetes.io/docs/concepts/overview/. Kubernetes 2023a. 5.6.2024

Kubernetes documentation. https://kubernetes.io/docs/concepts/containers/. Kubernetes 2023b. 5.6.2024

Kubernetes documentation. https://kubernetes.io/docs/concepts/security/service-accounts/. Kubernetes 2023c. 5.6.2024.

Kubernetes documentation. https://kubernetes.io/docs/concepts/configuration/secret/. Kubernetes 2023d. 5.6.2024.

Kubernetes documentation. https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/. Kubernetes 2023e. 5.6.2024

Kubernetes documentation. https://kubernetes.io/docs/concepts/architecture/. Kubernetes 2023f. 5.6.2024

Kubernetes documentation. https://kubernetes.io/docs/concepts/services-networking/service/. Kubernetes 2023g. 6.6.2024

Kubernetes documentation. https://kubernetes.io/docs/concepts/architecture/nodes/. Kubernetes 2023h. 5.6.2024

Kubernetes documentation. https://kubernetes.io/docs/concepts/workloads/pods/sidecar-containers/. Kubernetes 2023i. 21.10.2024

Kubernetes documentation. https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/. Kubernetes 2023j. 5.6.2024

Kubernetes documentation. https://kubernetes.io/docs/reference/command-line-tools-reference/kube-proxy/. Kubernetes 2023k. 5.6.2024

Kubernetes documentation. https://kubernetes.io/docs/setup/production-environment/container-runtimes/. Kubernetes 2023l. 6.6.2024

Kubernetes documentation. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/. Kubernetes 2023m. 5.6.2024

Kubernetes documentation. https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller/. Kubernetes 2023n. 6.6.2024

Kubernetes documentation. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/. Kubernetes 2023o. 5.6.2024

Kubernetes documentation. https://kubernetes.io/docs/reference/kubectl/. 2023p. 5.6.2024

Kubernetes documentation. https://kubernetes.io/docs/concepts/overview/components/. Kubernetes 2023q. 5.6.2025

Kubernetes documentation. https://kubernetes.io/docs/concepts/overview/kubernetes-api/. Kubernetes 2023r. 6.6.2024

Kubernetes documentation. https://kubernetes.io/docs/concepts/services-networking/ingress/. Kubernetes 2023s. 14.6.2024

Microsoft, https://learn.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops. Microsoft 2024a. 28.10.2024

Microsoft, https://learn.microsoft.com/en-us/entra/identity-platform/app-objects-and-service-principals?tabs=browser. Microsoft 2024b. 18.10.2024

Microsoft, https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/overview. Microsoft 2024c. 17.10.2024

Microsoft, https://learn.microsoft.com/en-us/azure/devops/pipelines/process/templates?view=azure-devops&pivots=templates-includes. Microsoft 2024d. 4.11.2024

NGINX documentation, https://docs.nginx.com/nginx-ingress-controller/, NGINX 2021. 10.6.2024

Red Hat. https://www.redhat.com/en/topics/devops/what-is-ci-cd. 2023. Red Hat 28.10.2024

Rensin, D. *Kubernetes, Scheduling Future at Cloud Scale,* O'Reilly, 2015. 12.10.2024

Virtanen, R. Introduction to Call Subprocess. https://docs.frends.com/en/articles/8010514-introduction-to-call-subprocess. Frends Doc. 2024. 12.9.2024