Eetu Lepistö

APACHE KAFKA-KLUSTERIN KÄYTTÖÖNOTTO KUBERNETES-ALUSTALLA

Opinnäytetyö

Liiketalouden ammattikorkeakoulututkinto

Tietojenkäsittely

2022

Kaakkois-Suomen ammattikorkeakoulu

Kaakkois-Suomen ammattikorkeakoulu

Tutkintonimike	Tradenomi (AMK)
Tekijä/Tekijät	Eetu Lepistö
Työn nimi	Apache Kafka-klusterin käyttöönotto Kubernetes-alustalla
Toimeksiantaja	Metatavu Oy
Vuosi	2022
Sivut	34 sivua
Työn ohjaaja(t)	Janne Turunen

TIIVISTELMÄ

Tämän opinnäytetyön tavoite oli tutustua Apache Kafkan ja Kubernetes-klusterin sovellusarkkitehtuuriin ja toimintaperiaatteisiin. Tämän lisäksi käytiin läpi modernin sovelluskehityksen taustalla vaikuttavaa DevOps-toimintatapaa. Käytännön osuudessa tavoitteena oli luoda toimiva Kafka-käyttöönotto ensin virtuaalikoneessa ja tämän jälkeen Kubernetes-alustalla. Toiminnan testauksessa käytettiin Kafkan tarjoamia demosovelluksia. Tämän lisäksi testattiin myös toimivan klusterin vikasietoisuus molemmissa käyttöönotoissa.

Toimeksiantajayritys pyrkii tulevaisuudessa ottamaan Apache Kafkan osaksi nykyistä ylläpitopalveluiden kokonaisuutta. Monia yrityksen palveluita ylläpidetään Kubernetes-alustalla, jolloin ylläpitoon liittyviä toimintoja saadaan automatisoitua. Loogista on siis käyttää samaa alustaa myös Kafkan ylläpitämiseen.

Tuloksena syntyi kolmesta välittäjästä ja kolmesta Zookeeperista koostuva vikasietoinen Kafka-klusteri Kubernetes-alustalla. Työssä esitellään työvaiheet valituilla tekniikoilla tämän saavuttamiseksi. Kubernetes-käyttöönotossa käytettiin apuna avoimeen lähdekoodiin perustuvaa Strimzi-projektia, jonka tarjoamat apuohjelmat helpottavat prosessia. Kubernetes-klusteri saatiin käyttöön virtuaalikoneessa hyödyntämällä Minikubea.

Kafka-klusterin todellinen hyöty realisoituu vasta siihen kytkettyjen palveluiden kautta, joten työn lopussa pohdittiin myös, toimeksiantajan toiveiden lisäksi, muita mahdollisia jatkokehitysideoita Kafkan käytölle.

Tässä työssä esitellään yksi monista mahdollisista tavoista toteuttaa Kafka-klusterin käyttöönotto ja sen pohjalta voidaan tutkia myös uusia lähestymistapoja ongelmien ratkaisuun. Työ toimii myös askeleena kohti tuotantokelpoista Kafka-klusteria Kubernetes-alustalla, mikä tukee toimeksiantajan pitemmän aikavälin tavoitteita.

Asiasanat: ohjelmistot, ohjelmistoarkkitehtuuri, Kafka, Kubernetes, DevOps

South-Eastern Finland University of Applied Sciences

Degree	Bachelor of Business Administration
Author (authors)	Eetu Lepistö
Thesis title	Deployment of Apache Kafka cluster in Kubernetes
Commissioned by	Metatavu Oy
Time	2022
Pages	34 pages
Supervisor	Janne Turunen

ABSTRACT

The goal of this thesis was to explore the basic operating principles and architecture of Apache Kafka and Kubernetes. In addition to this, the DevOps approach behind modern software development was explained. The objective of the practical part was first to create a working Kafka deployment in a virtual machine and after that in Kubernetes. Demo applications, what Kafka provided, were used to test the functionality of the service. Additionally, the fault tolerance was tested in both deployments.

The commissioner company was planning to use Apache Kafka as part of the pipeline of administrative services. Many of the company's services were hosted in Kubernetes which enabled automating some administrative operations. Therefore, it was logical to use the same platform for Kafka deployment.

As a result, a fault tolerant Kafka cluster containing three brokers and three zookeepers was successfully deployed in Kubernetes. The thesis showed the different phases with selected tools to achieve this. An open-sourced Strimzi project provided tools to help with the deployment in Kubernetes. The Kubernetes cluster was created in a virtual machine using Minikube.

Because true benefits of Kafka cluster only become real through clients and services connected to it, the end of thesis considered, in addition to the commissioner's visions, further development ideas for using Kafka. This thesis introduced one of the many ways to create Kafka deployment and other solutions to do so can be explored based on it. This work also worked as a step forward to the production state of the Kafka cluster in Kubernetes, which would be the long-term goal of the commissioner company.

Keywords: software, software architecture, Kafka, Kubernetes, DevOps

SISÄLLYS

1 JOHDANTO

Tämän työraportin aiheena on tutkia, toimeksiantajayrityksen käyttämää Apache Kafka-tapahtumavirta-alustaa ja sen käyttöä Kubernetes-ympäristössä. Käytännön toteutuksen taustalla käsitellään ensin näiden sovelluksien rakennetta ja toimintaa teoriassa. Tavoitteena on siis tutkia ja toteuttaa toimivan Kafka-klusterin käyttöönotto Kubernetes-alustalla.

Metatavu Oy on monipuolinen sovelluskehitykseen ja palveluiden ylläpitoon keskittyvä yritys. Yritys pyrkii mahdollisuuksien mukaan käyttämään avoimen lähdekoodin projekteja apuna omissa asiakastöissään. Ylläpitoalustana on jo pitkään toiminut pilviratkaisuja tarjoava AWS (Amazon Web Services) ja siellä toimiva Kubernetes. Yrityksen on tarkoitus tulevaisuudessa liittää Apache Kafka osaksi nykyisiä ylläpitopalveluita ja tuotantoputkea. Tästä syntyi tarve tutkia Kafkan käyttöönottoa Kubernetes-alustalla. Tavoitteena oli myös selvittää, mitä tuotantokelpoisen Kafka-klusterin ylläpitäminen käytännössä vaatisi.

Tämä raportti jakautuu johdannon lisäksi neljään osaan, joista ensimmäinen (luku 2) käsittelee käytössä olevien tekniikoiden ja työtapojen teoriaa. Siinä avataan DevOps-käsitettä ja toimintatapaa. Lisäksi tutustutaan työssä käytettävien työkalujen, Kubernetesin ja Kafkan, taustalla oleviin konsepteihin. Näitä ovat konttiteknologia, konttiorkestrointi sekä Kafkan yläkäsite tapahtumavirta-alusta. Tämän jälkeen siirrytään käytännön toteutukseen.

Kolmas luku käsittelee Kafka-klusterin paikallista asennusta virtuaalikoneessa, ja siinä käydään tarkemmin läpi käytännössä teoriaosuudessa esiteltyjä Kafkan komponentteja. Neljäs luku esittelee Kafkan asennuksen virtuaalikoneessa toimivaan Kubernetes-klusteriin. Luvun lopussa tarkastellaan saavutettuja tuloksia ja pohditaan jatkokehitysideoita Kafkan hyödyntämiseksi tuotannossa. Viimeinen luku on päätäntö, jossa käsitellään työn ja raportin tekemistä. Siinä esitellään tekijän omia ajatuksia työn aiheen valinnasta, raportin tekemisen taustalla olevista aiheista sekä toimeksiantajan palautetta raportista.

2 TYÖMENETELMÄT JA TEKNIIKAT

2.1 DevOps

DevOps käsitteenä ei ole tarkasti määritelty. Lähteestä riippuen termillä tarkoitetaan hieman eri asioita. Määritelmien erot ovat kuitenkin melko pieniä ja kaikki noudattavat samoja pääpiirteitä. Yksi yhteinen piirre on kuvata DevOps uudenlaisena ajattelutapana, ja sen tarkoituksena on helpottaa tiimien välistä työskentelyä. Toinen päätavoite on kehittää yrityksen toimintaa ja parantaa asiakkaan kokemusta tuomalla asiakkaiden palautteet nopeasti kehitystiimin tietoon (Coyne & Sharma 2015, 5).

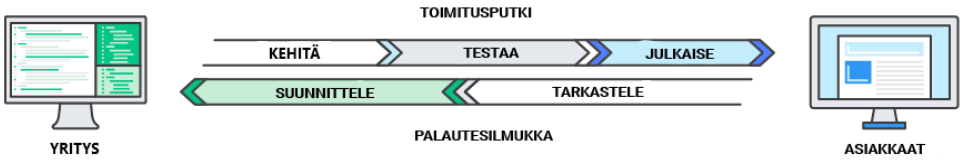
DevOps on sähköisessä palvelutuotannossa käytettävä toimintamalli, joka on nykyään omaksuttu osaksi valtavirran liiketoimintaa. Sovelluskehitystä (Development eli Dev) ja palveluntarjontaa (Operations eli Ops) on aiemmin ajateltu erikseen toimivina ”siiloina”, joissa nämä kaksi toimivat toisistaan erillään. Siiloutuminen on aiheuttanut kroonisen ristiriidan sovelluskehityksen ja palveluntarjonnan välillä, mikä tarkoittaa lyhyesti sitä, että tuotekehitys on hidasta ja kallista. Liiallinen byrokratia, vastuunsiirto ja autonomian puute on näkynyt joidenkin yritysten heikkona tuloksena. DevOps-toimintamalli pyrkii ja on jo onnistunutkin murtamaan nämä siilot. (Abildskov 2013.)

DevOps:ssa on kyse tuote- ja pilvijärjestelmien hallinnasta, teknisestä huippuosaamisesta sekä terveestä yrityskulttuurista. Jatkuvan oppimisen ja työnteon kehityksen mahdollistaa psykologisesti turvallinen työympäristö, jossa ei etsitä syyllisiä vaan opitaan virheistä yhdessä. (Abildskov 2013.)

Toimintamallissa yhdistyy ajattelutapa, työskentelytavat ja työkalut, jotka parantavat yrityksen kykyä toimittaa sähköisiä palveluita nopeasti. Nopeus mahdollistaa yritykselle kilpailuedun markkinoilla perinteisiä kehitystapoja käyttäviä yrityksiä vastaan. Tässä mallissa erilliset sovelluskehitys (Dev) - ja palveluntarjonta (Ops) -tiimit voidaan yhdistää, jolloin eri alojen asiantuntijat työskentelevät sovelluksen parissa koko sen elinkaaren ajan. Näin tekijöille kehittyy myös yksittäistä työvaihetta laajempi tietotaito. (AWS 2022.)

Toimintamallin ytimessä on jatkuva kehitys, testaus ja julkaisu. Näin asiakkaiden liiketoiminnalle voidaan tarjota jatkuvasti lisäarvoa ja uusia ominaisuuksia

voidaan testata nopeasti oikeiden käyttäjien avulla. Kuva 1 näyttää toimitusputkeen kuuluvien osien lisäksi palautesilmukan, johon kuuluu tulosten tarkastelu sekä uusien ominaisuuksien suunnittelu saadun palautteen pohjalta.



Kuva 1. Amazonin määritelmän mukainen DevOps-malli (käännetty lähteestä AWS 2022)

Jatkuva kehitys ja julkaisu on mahdollista, kun kehitetään pieniä osia palvelusta kerrallaan. Ei esimerkiksi muotoilla käyttöliittymää, tietokantaa ja serverikoodia samalla kertaa vaan tehdään näistä omia kokonaisuuksia, joita ylläpidetään erillään muista palasista. Konttiteknologia mahdollistaa tällaisen mikropalvelu-tyyppisen kehityksen ja se onkin saavuttanut suurta suosiota viimeisen vuosikymmenen aikana (Fadilpašić 2020).

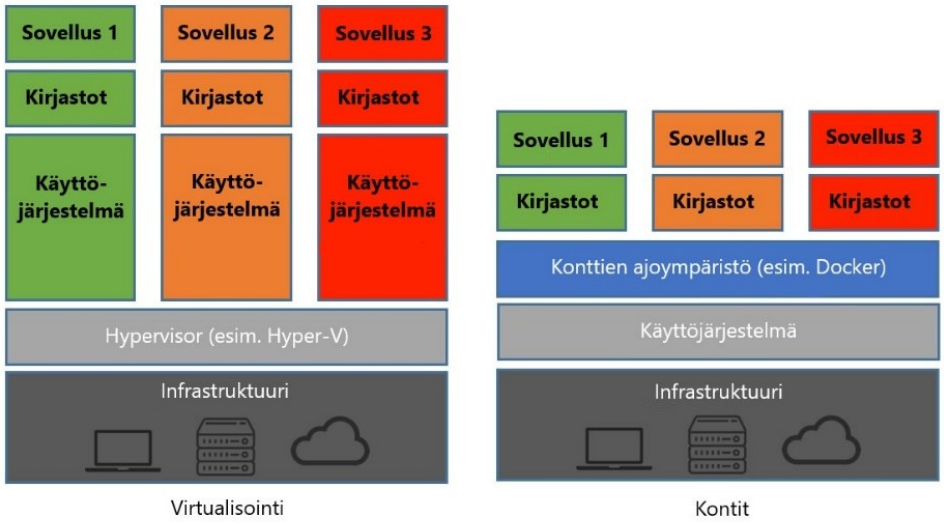
2.2 Konttiteknologia

Konttiteknologian (Container Technology) ajatus on samankaltainen kuin 50-luvulla keksityssä rahtikontissa. Idea on siinä, että yhteen konttiin ”pakataan” yhden tai useamman sovelluksen tarvitsemat riippuvuudet, kirjastot ja asetukset. Tämän jälkeen konttia voidaan ajaa eri alustoilla käyttäen paikallisia resursseja tai pilvipalveluita. Konttien luomiseen ja ajamiseen on useita työkaluja. Yksi tällä hetkellä suosituimmista on Docker (Lashawn 2021). Muita vastaavia ovat esimerkiksi CoreOS rkt, runC ja Windows Containers.

Konteilla voidaan saavuttaa monenlaisia hyötyjä esimerkiksi sovelluskehittäjän näkökulmasta. Kehitettävän sovelluksen kaikki tarvittavat osat voidaan pakata yhden kontin sisään. Sovellusta voi tämän jälkeen testata ja käyttää eri laitteilla ilman, että käyttäjien omia asetuksia tarvitsee muuttaa tai erillisiä kirjastoja ladata käsin. Kontti sisältää jo kaiken mitä ohjelman suorittamiseen vaaditaan. Kontti luodaan aina tietylle käyttöjärjestelmälle, ja sen käytössä onkin otettava huomioon se, että samaa konttia voidaan ajaa vain samalla käyttöjärjestelmällä. Jos samaa sovellusta halutaan ajaa esimerkiksi Macissa ja Linuxissa, pitää näille luoda omat konttinsa. (Red Hat 2018.)

Konttien etuna on se, että ne ovat kevyitä käyttää ja nopeita pystyttää. Käytetään vertailukohtana virtuaalikonetta (VM, Virtual Machine). Virtuaalikone virtualisoi fyysisen laitteiston, johon asennetaan käyttöjärjestelmä. Kontti virtualisoi pelkästään tarvitsemansa käyttöjärjestelmän osat, jolloin tiedostokokokin pysyy selvästi pienempänä (IBM 2021b). Kontti voidaan luoda periaatteessa mille tahansa käyttöjärjestelmälle, mutta yleisimmin käytetään Linux-käyttöjärjestelmää.

Kuva 2 havainnollistaa virtuaalikoneen ja konttiteknologian eroja käytännössä, ja huomattavin ero liittyy nimenomaan käyttöjärjestelmän tarpeeseen. Siinä missä virtuaalikoneet pyörittävät omissa käyttöjärjestelmissään käytettäviä sovelluksia, kontit pyörivät yhden ”isäntäjärjestelmän” päällä käyttäen ajoympäristöä, esim. Dockeria. Virtuaalikonetta käyttävän ratkaisun tiedostokoko on yleensä gigatavuja, kun taas konttien koossa puhutaan megatavuista (Vmware). Tästä ”keveydestä” johtuen konttien käyttöönotto on hyvin nopeaa ja niitä voidaan pyörittää samalla serverillä useita samaan aikaan. Tällä saavutetaan parempi vikasietoisuus, kun yhden kontin eli sovelluksen vikaantuessa toinen voi jatkaa toimintaansa.



Kuva 2. Virtualisoinnin ja konttiteknologian eroavaisuudet (käännetty lähteestä Jones 2018)

Konttiteknologian etujen lisäksi sen käyttöön liittyy myös erilaisia tietoturvauhkia, jotka on hyvä tiedostaa teknologian käyttöönottoa suunniteltaessa. Yleisin turvallisuusuhka liittyy kontin toimintatapaan, jossa se jakaa ”isäntäkäyttöjärjestelmän” tai sen ydinosan eli kernelin (Groll 2021). Jos käyttöjärjestelmän

osasta löytyy tietoturva-aukko, voi se saastuttaa myös sen päällä toimivat kontit ja aiheuttaa tietoturvariskin. Smirnovin (2022) mukaan palveluntarjoajat, kuten Amazon, tarjoavat kuitenkin erilaisia turvamekanismeja vastaavien ongelman välttämiseksi. Tämä vaatii kuitenkin sen, että kontteja käsitellään palveluntarjoajan ylläpitämällä alustalla, Amazonin tapauksessa se olisi esimerkiksi Kubernetes.

Virtuaalikonetta ja kontteja verratessa on myös hyvä muistaa, että ne eivät poissulje toisiaan sovelluskehityksessä. Kontteja voidaan ajaa myös virtuaalikoneessa, jolloin saadaan molempien teknologioiden hyödyt käyttöön. Samalla voidaan väistää erilaisten ”isäntäkäyttöjärjestelmien” yhteensopivuusongelmat. Lopulta kyse on myös siitä, millaisesta palvelusta on kyse: Tarvitaanko virtuaalikoneen tarjoaman täyden käyttöjärjestelmän edut vai onko kontin käyttöönoton nopeus ja keveys suuremmassa roolissa. Yksi konttiteknologian vahvuuksista on se, että palvelu voidaan konttien avulla pilkkoa pieniin osiin. Näin palvelun osia voidaan ylläpitää ja kehittää erikseen (Amanse 2020). Konttien ja työmäärän kasvaessa, tarvitaan jonkinlaista automatisointia konttien ylläpitoon. Tätä kutsutaan konttiorkestroinniksi.

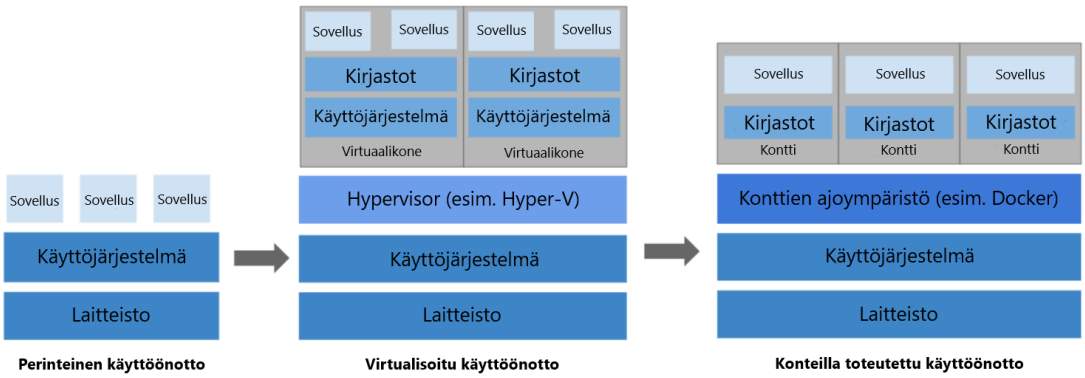
2.3 Konttiorkestrointi

DevOpsin mukaiseen toimintamalliin kuuluu työvaiheiden automatisointi, silloin kun se on järkevää. Esimerkkejä tällaisista työvaiheista on mm. palveluiden käyttöönotto, päivitys, monitorointi ja skaalaus. Konttiteknologia ja konttien hallintaan käytettävä konttiorkestrointi (Container Orchestration) ratkaisee näitä ongelmia tehokkaasti ja siksi siihen perustuvat työkalut ovat hyvin suosittuja nykyajan sovelluskehityksessä. Ensimmäisenä käydään läpi ns. perinteinen tapa julkaista sovellus asiakkaiden käytettäväksi. Tämän jälkeen tutustutaan konttiorkestrointiin ja kuinka sen avulla perinteisen tavan sudenkuopat ja ongelmat voidaan ratkaista.

Perinteisesti palvelu, esimerkiksi verkkosovellus asennetaan palvelimelle, jolle on asennettu käyttöjärjestelmä sovelluksen ajamista varten. Jos sovelluksen käyttäjämäärä kasvaa riittävästi, täytyy palvelinkoneen tehoja lisätä tai hankkia toinen palvelin jakamaan kuormaa. Monesti sovelluksia pyörittävät palveli-

met on kokonaan ulkoistettu erillisiin datakeskuksiin, joista palvelintilaa voidaan ostaa arvioidun kuorman mukaan. Tässä on kuitenkin ongelma silloin, kun resursseja ei käytetä tasaisesti ja joudutaan maksamaan ns. turhasta. Näin on usein esimerkiksi nettisivuilla, joilla käyttäjämäärät vaihtelevat suuresti. Toinen esimerkki on elinkaarensa alkupäässä oleva palvelu, joka palvelee vain pientä käyttäjämäärää. Suosion kasvaessa käyttäjämäärä kasvaa ja näin ollen palvelinresursseja tarvitaan jatkuvasti lisää. Palvelun skaalaaminen perinteisin menetelmin on siis aina aikaa vievää ja työlästä eikä se tapahdu heti vaan monesti asiakkaat huomaavat ongelmia ruuhkautuneessa palvelussa. (Kubernetes 2021f.)

Virtualisoinnin avulla voidaan parantaa resurssien käyttöä, koska jokaiselle palvelulle ei tarvita omaa serveriä. Tämä mahdollistaa myös kustannustehokkaamman resurssien käytön, kun yhdellä serverillä voidaan ajaa useita palveluita virtuaalikoneissa. Konttiteknologia jatkaa tätä kehitystä käyttämällä hyväksi virtualisointia. Kun virtualisoidaan myös käyttöjärjestelmä, tai oikeastaan vain tarvittavat osat siitä, saadaan kompakti ja nopea tapa ajaa yksittäisiä sovelluksia (Kubernetes 2021f). Kuva 3 näyttää käyttöönoton (deployment) rakenteen perinteisestä kokoonpanosta nykypäivän konttiteknologiaan.



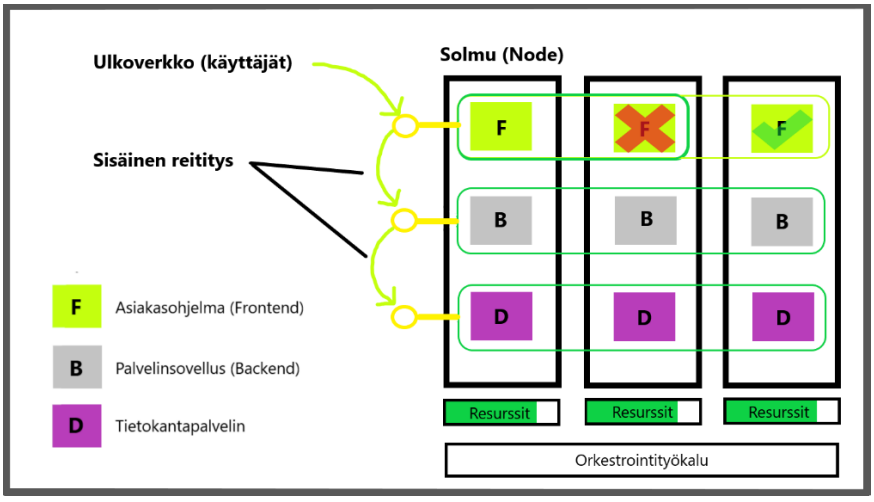
Kuva 3. Sovellusympäristöjen historia (käännetty lähteestä Kubernetes 2021f)

Konttiorkestrointi tarkoittaa kontteihin liittyvien toimintojen automatisointia. Toimintoja ovat mm. käyttöönotto, resurssien jako, reititys, skaalaus sekä konttien elinkaaren hallinta (IBM 2021a). Konttiorkestroinnin tarkoitus on helpottaa ja nopeuttaa konteilla toteutetun palvelun käyttöönottoa ja ylläpitoa. Goltsman (2019) sekä IBM (2021) toteavat konttiorkestroinnin mahdollistavan konttien

määrän kasvattamisen ja monimutkaisuuden lisäämisen pienemmällä ihmistyömäärällä perinteisempiin käyttöönottoihin verrattuna.

Käyttöönottoon sisältyy konttien käynnistys, konttien välinen reititys ja myös palvelun paljastaminen (expose) ulkoverkolle eli yleensä käyttäjille (Vennam 2019). Ylläpitoon kuuluu konttien monitorointi, skaalaus ja saavutettavuuden varmistus. Jos esimerkiksi yksi konteista kaatuu, orkestrointityökalu käynnistää automaattisesti uuden sen tilalle ja ohjaa eli reitittää liikenteen tälle kontille.

Kuva 4 havainnollistaa konttiorkestroinnin toimintaa yksinkertaisen web-sovelluksen avulla. Värilliset laatikot kuvaavat konttisovelluksia. Kontteja käytetään solmuiksi (node) nimetyissä kokonaisuuksissa, joiden toimintaa orkestrointityökalu ohjaa. Jokaiselle solmulle on määritelty etukäteen niiden käytössä olevat resurssit. Orkestrointityökalun tehtävä on myös jakaa saapuvaa liikennettä eli kuormaa tasaisesti eri solmujen kesken. Myös toimivien konttien lukumäärä tulee pysyä etukäteen määriteltynä. Kun kuvan 4 mukaisesti yksi asiakasohjelmakontti lakkaa toimimasta, orkestrointi käynnistää kolmanteen solmuun välittömästi uuden kontin ja ohjaa liikenteen sinne. (Vennam 2019.)



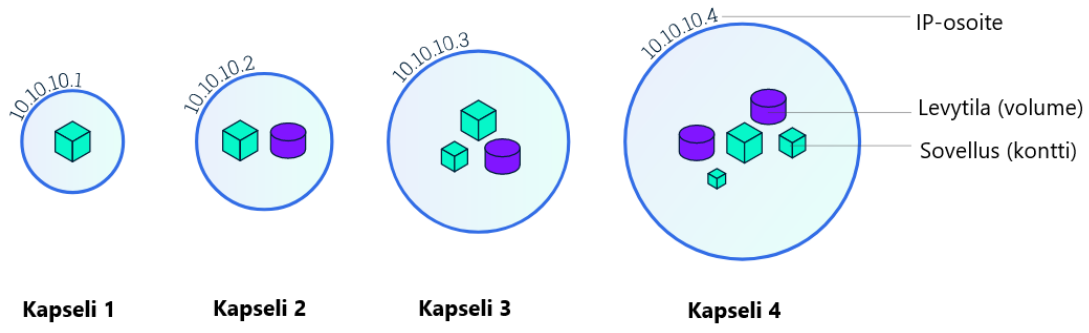
Kuva 4. Konttiorkestroinnin toiminta (mukaillen Vennam 2019)

Konttiorkestrointityökaluja on useita erilaisia tarjoten eri tarpeisiin sopivia ominaisuuksia. Työkaluja ovat mm. Kubernetes, Docker Swarm, Apache Mesos ja Nomad (CloudZero 2021). Tässä opinnäytetyössä työkaluna toimii Kubernetes, joten tutustutaan seuraavaksi siihen.

2.4 Kubernetes

Tällä hetkellä yksi tunnetuimmista ja myös toimeksiantajayrityksen käyttämä konttiorkestrointityökalu on Kubernetes (IBM 2021a). Kubernetes on alun perin Googlen kehittämä palvelu. Se pohjautuu Googlen käyttämään Borg-nimiseen konttiorkestrointityökaluun, joka on vastuussa Googlen pilvipalveluiden toiminnasta (Red Hat 2020). Kubernetes ei tarjoa varsinaista PaaS (Platform as a Service)-kokonaisuutta vaan erilaisia yhteensopivia työkaluja, joita voidaan ottaa käyttöön tarpeen mukaan (Kubernetes 2021f). Tämä tekee palvelusta monipuolisen käyttää, mutta samalla lisää käyttäjän vastuuta konfiguraation toiminnasta kokonaisuutena, jos ei haluta käyttää oletusasetuksia.

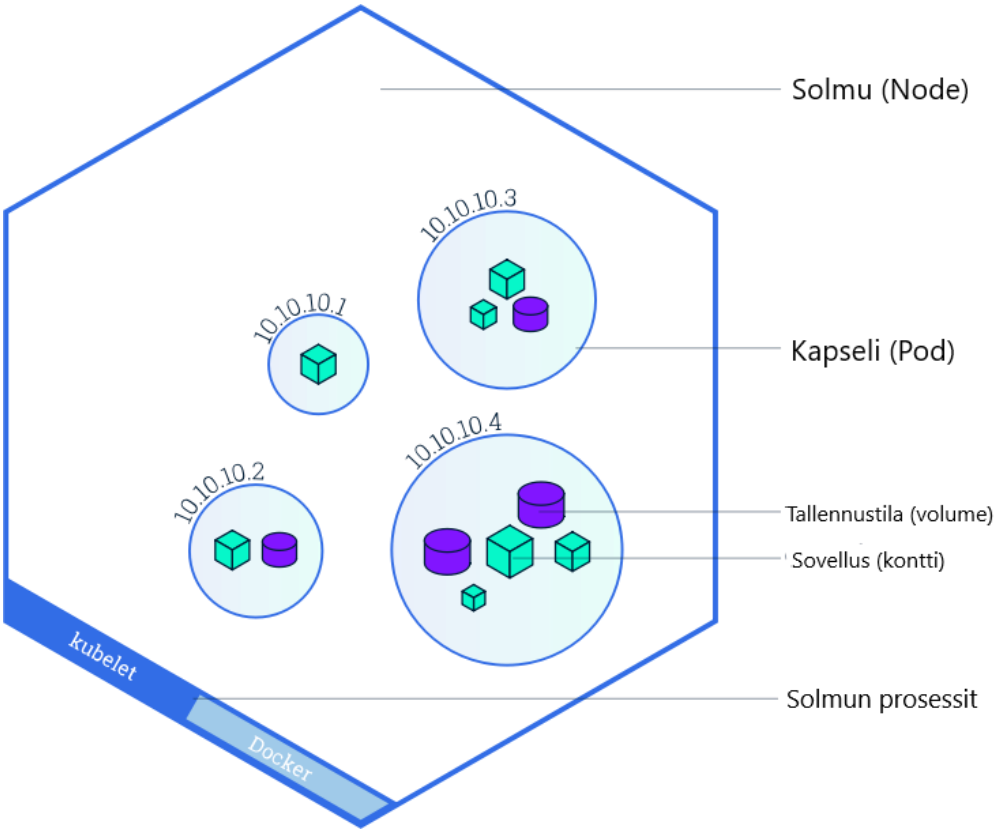
Seuraavaksi käydään läpi, kuinka Kubernetes toimii kontiksi muutetun sovelluksen käyttöönotossa. Ensimmäisenä käydään läpi osat, mistä kokonaisuus koostuu. Kuva 5 esittää kapseleita (pod), jotka sisältävät yhden tai useampia konttisovelluksia (containerized application). Kapseli sisältää myös jaetun tallennustilan (volume), IP-osoitteen sekä tiedot siitä, kuinka kapselin sisällä olevia kontteja tulee ajaa (käytettävät portit, kontin versionumero ym.). Kapseli on Kubernetes-alustan pienin yksikkö (Kubernetes 2021e).



Kuva 5. Eri kokoonpanoilla olevia kapseleita (käännetty lähteestä Kubernetes 2021e)

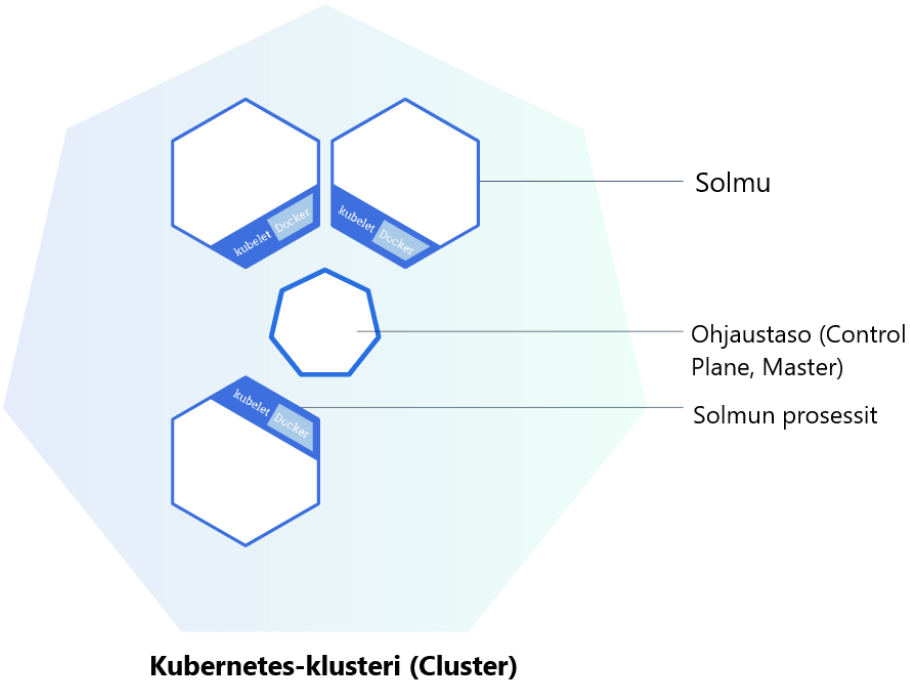
Solmu (node) vastaa Kubernetesissa fyysistä tietokonetta tai virtuaalikonetta. Solmu sisältää yhden tai useamman kapselin kontteineen sekä tarvittavat prosessit toimintojen suorittamiseen. Näitä ovat ainakin konttien ajoympäristö (esimerkiksi Docker) sekä Kubelet. Kubelet vastaa solmun ja ohjaustason

(control plane, tästä lisää myöhemmin) välisestä viestinnästä sekä kapseleiden ja konttien ylläpidosta. Kuva 6 havainnollistaa solmun sisältämiä toimintoja. (Kubernetes 2021b.)



Kuva 6. Solmu sisältää yhden tai useampia kapseleita (käännetty lähteestä Kubernetes 2021b)

Viimeisenä ja suurimpana rakenteena on Kubernetes-klusteri (cluster). Klusteri mahdollistaa ylläpidettävän palvelun skaalaamisen optimoimalla resurssien käyttöä automaattisesti ennalta määriteltyjen asetusten mukaisesti. Klusterin ytimessä on ohjaustaso (control plane), mikä vastaa koko klusterin hallinnasta. Tehtäviin kuuluu mm. sovellusten tilan monitorointi, skaalaus ja päivittäminen. Kuten kuva 7 osoittaa, klusteri on nimensä mukaisesti tietokoneiden eli solmujen ”rypäs”, jossa sovelluksia ajetaan. Yllä on esitelty Kubernetes-alustan osat pienimmästä palasesta suurimpaan, mutta käytännössä palvelun käyttöönotto alkaa aina klusterin luomisesta. (Kubernetes 2021d.)



Kuva 7. Kubernetes-klusteri sisältää joukon tietokoneita eli solmuja sekä ohjaustason, joka hallinnoi klusteria (käännetty kohteesta Kubernetes 2021d)

Jotta sovellus tai palvelu voidaan ottaa käyttöön Kubernetes-alustalla, tarvitaan käyttöönotto (deployment). Käyttöönotto on sarja asetuksia, jotka syötetään ohjaustasolle. Nämä asetukset määrittävät klusterin halutun toiminnan, esimerkiksi kuinka monta konttia halutaan ajaa samanaikaisesti sekä niiden käytössä olevat resurssit. Jos Kubernetes on kapellimestari, voidaan käyttöönottoa ajatella ”sävellyksenä”, jonka mukaan orkesteria johdetaan.

Ennen orkestroinnin keksimistä voitiin käyttää nk. aloitusskriptejä sovellusten käynnistämiseen erilaisissa ympäristöissä. Käyttöönoton etu skripteihin nähden on se, että ohjaustaso vahtii siinä määriteltyjä asioita koko ajan ja jos klusterin tila ei vastaa asetettuja vaatimuksia, aloittaa ohjaustaso tarvittavat toimenpiteet välittömästi. Tämä mahdollistaa mm. nopean palautumisen mahdollisista virhetilanteista. (Kubernetes 2021c.)

Käyttöönoton luomiseen ja muokkaamiseen on olemassa oma komentorivikäyttöliittymä (CLI) nimeltä Kubectl. Samaa työkalua voidaan käyttää myös, kun halutaan tietoja klusterista tai yksittäisistä kapseleista tai solmuista. Kubectl käyttää klusterin kanssa viestiessään Kubernetes API-rajapintaa (Kubernetes 2021c).

2.5 Tapahtumavirta-alusta

Data tai event stream tarkoittaa jatkuvaa datan tai tapahtumien (event) virtaa mahdollisesti jopa tuhansista eri lähteistä. Lähteitä voivat olla esimerkiksi erilaiset laitteet ja sensorit (IoT), sosiaalisen median data tai paikkatiedot (AWS 2021). Oikeastaan melkein mitä vain voidaan kuvata tapahtumina ja kun kyse on jatkuvasti toistuvista tapahtumista, kyse on tapahtumavirrasta. Kyse on lopulta vain siitä, millaista dataa haluamme analysoida, prosessoida ja tallentaa (Narkhede, Palino ym. 2017, 248).

Kuva 8 osoittaa tapahtumapohjaisen tietojenkäsittelyn perusrakenteen tapahtumavirta-alustaa käyttäen. Vasemmalla on erilaisia datalähteitä, joista tietoa luetaan/haetaan (subscribe). Alustalla on määritelty erilaisista lähteistä tulevan tiedon jatkokäsittely ja lukemisen jälkeen tieto kirjoitetaan/lähetetään (publish) kyseiselle ohjelmalle tai palvelulle jatkotoimenpiteitä varten. Kyse voi olla perinteisestä maksutietojen tallentamisesta tietokantaan tai tuhansien paikkatietojen tarjoaman datan analysoinnista lähes reaaliaikaisesti. Tapahtumavirta-alustan käyttäminen palveluiden ”välittäjänä” tuo myös tietynlaista modulaarisuutta palvelun rakenteeseen. Tämä tukee DevOps:n mukaista ketterää kehitystä (Dhanushka 2021).

Kuva 8. Tapahtumavirta-alusta voi ottaa vastaan dataa useista eri lähteistä

Tapahtumavirta-alusta (Event Streaming Platform, ESP) pystyy käsittelemään jatkuvaa tapahtumien virtaa halutulla tavalla. Käsittely voi olla esimerkiksi statistiikan laskemista ja visualisointia tilastodatan pohjalta tai paikkatietoon perustuvien suositusten tarjoamista käyttäjille. Mahdollisuuksia alustan hyödyntämiseen on paljon ja onkin tärkeää tunnistaa, milloin alustan hyödyntäminen on järkevää. Dataa voidaan yleensä käyttää myös suoraan kohdepalvelussa

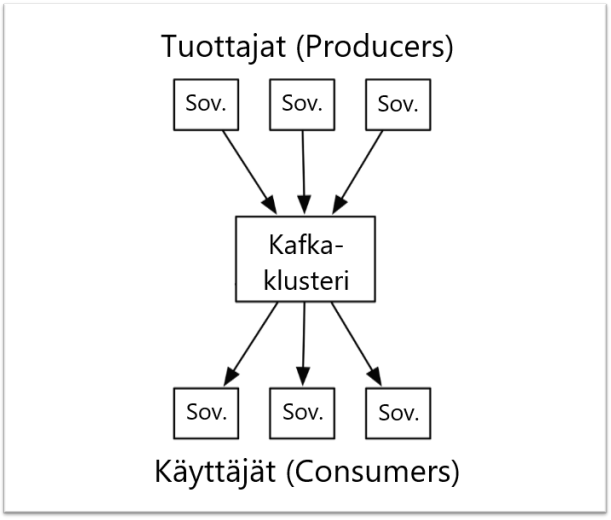
ilman tämän kaltaista ”välittäjää”. Jos kuitenkin tapahtumien määrä kasvaa yllättäen, voi se aiheuttaa kohdepalvelussa ruuhkautumista ja pahimmillaan kaatumisen. Tapahtumavirta-alusta toimii tällaisessa tapauksessa ns. puskurina tapahtumavirralle, koska se pystyy keräämään tapahtumat jonoksi odottamaan jatkokäsittelyä. Näin parannetaan myös palvelun saavutettavuutta ja vikasietoisuutta. (Dhanushka 2021.)

Tapahtumavirta-alusta on työkalu, jolla on kolme tärkeää ominaisuutta. Tapahtumavirtaa tallennetaan vikasietoisesti ja lähes reaaliaikaisesti. Tämän lisäksi työkalu pystyy kirjoittamaan (publish) ja lukemaan (subscribe) datavirtaa sekä viemään ja tuomaan dataa erilaisista lähteistä (Goyal 2020). Tämän määritelmän mukaisia avoimen lähdekoodin työkaluja ovat ainakin Apache Pulsar ja Apache Kafka. Näistä jälkimmäinen tulee olemaan tämän työn pääaiheena, joten tutustaan seuraavaksi siihen.

2.6 Apache Kafka

Apache Kafka on alun perin LinkedInin kehittämä avoimeen lähdekoodiin perustuva tapahtumavirta-alusta (Li 2020). Kafka-pohjaisessa palvelussa on kolme peruskomponenttia, tuottajat (producers), käyttäjät (consumers) ja Kafka-klusteri (Code Factory 2021). Ensiksi käydään läpi Kafka-pohjaisen palvelun perusrakenne yksinkertaistettuna ja tutustutaan sen jälkeen tarkemmin Kafkan sisältämiin komponentteihin.

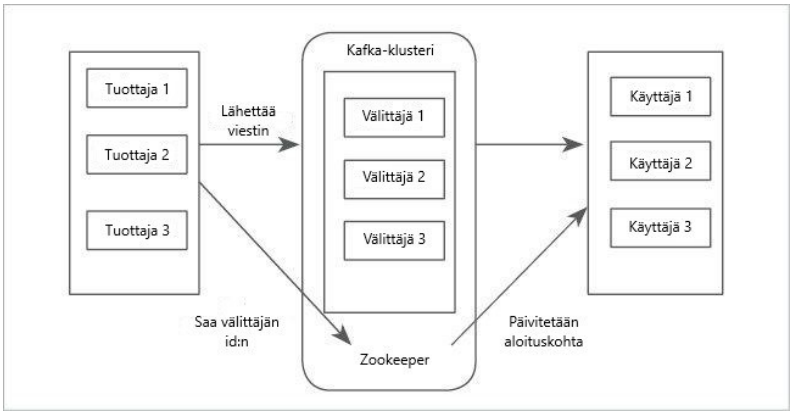
Tuottajat ovat sovelluksia, jotka kirjoittavat tapahtumia (events) Kafka-klusteriin. Kafka-klusterin tehtävä on tallentaa nämä tapahtumat aikajärjestyksessä ja varmistaa, että jokainen tapahtuma käsitellään vain kerran. Kafkan voi siis ajatella eräänlaisena lokikirjan ylläpitäjänä. Käyttäjät taas lukevat Kafkaan tallennettuja tietoja ja yleensä tekevät jonkinlaista jatkokäsittelyä sille. Joskus se tarkoittaa vain tiedon tallentamista tietokantaan. Kuten kuva 9 esittää, Kafka-pohjaisen palvelun perusrakenne voi olla melko yksinkertainen. (Code Factory 2021.)



Kuva 9. Apache Kafkan toiminnan pääkomponentteja ovat tuottajat (producers), käyttäjät (consumers) sekä Kafka-klusteri (käännetty lähteestä Code Factory 2021)

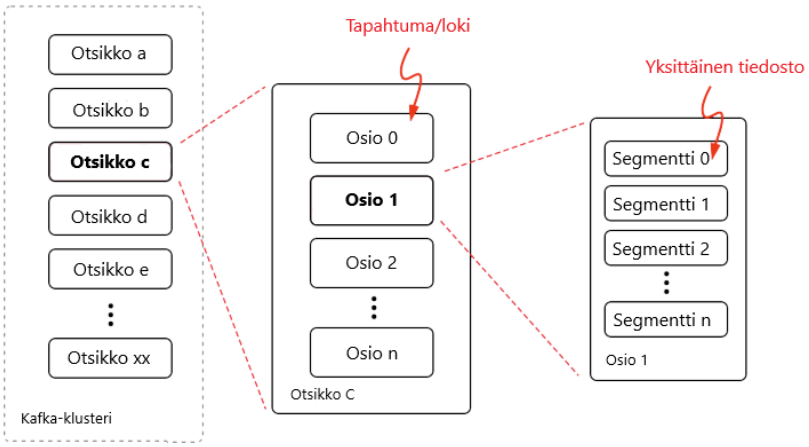
Kafka-klusteri koostuu yhdestä tai useammasta välittäjästä (broker), jotka ovat käytännössä servereitä. Ne voivat olla fyysisiä koneita, virtuaalikoneita tai kontteja. Välittäjät toimivat tapahtumatietojen tallennustilana, josta dataa myös luetaan jatkokäsittelyä varten. Data tallennetaan klusterissa hajautetusti eli useampaan välittäjään, mikä tekee Kafkasta vikasietoisen (Goyal 2020).

Zookeeper on Kafka-klusterin ylläpitäjä. Koska välittäjät ovat ns. korvattavia eli uuden tilalle voidaan aina asentaa uusi, ne eivät säilö klusterin toiminnan kannalta tärkeää dataa. Zookeeper säilöö erilaisia tietoja klusterista kuten lokien aloituskohta (offset) sekä välittäjien id:t. Se vastaa myös kuorman tasaamisesta välittäjien välillä ja siitä, että käyttäjät eivät lue samoja viestejä moneen kertaan. Kuva 10 kuvaa vikasietoista Kafka-klusteria. (Way to easy learn 2021.)



Kuva 10. Kun viestit tallentuvat usealle välittäjälle, yhden välittäjän kaatuminen ei kadota dataa. (käännetty lähteestä Way to easy learn 2021)

Kun tapahtuma saapuu välittäjälle, se tallennetaan otsikon (topic) alle. Kuvan 11 mukaisesti, välittäjä voi sisältää useita otsikoita. Otsikon sisällä on osioita (partition) joihin loki/tapahtuma tallennetaan. Osio vastaa siis fyysistä tallennustilaa, kun taas otsikko on ikään kuin kategoria, jonka alle viestit tallentuvat. Otsikon alla olevat osiot on hajautettu eri välittäjille. Tämä on yksi Kafkan avainominaisuuksista skaalautumisen osalta. Otsikot voidaan myös replikoida useammalle välittäjälle, jolloin datan vikasietoisuus kasvaa. Osion sisällä on vielä segmentit, jotka vastaavat yksittäisiä tiedostoja kovalevyllä. (Berglund 2020.)



Kuva 11. Välittäjä (broker) sisältää otsikon, osion ja segmentin (käännetty lähteestä Berglund 2020)

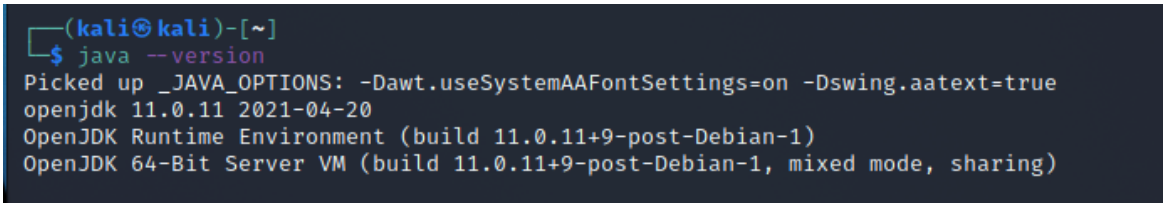
Kafka pystyy käsittelemään miljardeja tapahtumia päivässä ja pienen latenssin ansiosta se on lähes reaaliaikainen tapahtumavirta-alusta (Goyal 2020). Kafkan etuna on sen hajautettu arkkitehtuuri ja sen myötä saavutettava vikasietoisuus ja skaalautuvuus. Seuraavassa osiossa pyritään asentamaan toimiva Kafka-klusteri paikalliseen ympäristöön ja varmentamaan sen toiminta testituottajan ja testikäyttäjän avulla. Tämän jälkeen pyritään siirtämään tämä Kafka-pohjainen kokonaisuus Kubernetes-alustalle.

3 KAFKAN ASENTAMINEN

Ensimmäiseksi suoritetaan Kafkan käyttöönotto mahdollisimman suoraviivaisesti, jotta päästään näkemään käytännössä teoriaosuudessa esiteltyjen osien toimintaa. Tärkeitä osia ovat ainakin Zookeeper, Kafka-välittäjjä (broker), tuottaja (producer) ja käyttäjä (consumer). Tavoite on siis asentaa toi-

miva, kuvan 10 mukainen (erotuksena käytetään vain yhtä tuottajaa ja käyttäjää), Kafka-klusteri ja liittää siihen komentorivillä toimiva viestisovellus eli tuottaja. Käyttäjänä toimiva sovellus lukee tuottajan kirjoittamat viestit ja tulostaa ne konsoliin.

Paikallisessa käyttöönotossa isäntäjärjestelmänä toimii Windows. Koska Kafkan asennus vaatisi Linux-alijärjestelmän (WSL2) asentamista Windowsille, katsoin järkevämmäksi toteuttaa käyttöönoton virtuaalikoneessa natiivissa Linux-ympäristössä (Develop Paper 2021). Oracle VirtualBoxilla luodaan virtuaalikone, jonka käyttöjärjestelmänä toimii Kali Linux. Alkuvalmisteluina tarkistetaan Linuxin Java-versio kuvan 12 osoittamalla tavalla, Kafka vaatii toimiakseen vähintään Java-version 8 (Kafka 2022).



Kuva 12. Jos ”java –version”-komento antaa virheen, täytyy Java ensin asentaa. Tässä tapauksessa riittävä versio (11.0.11) on asennettuna

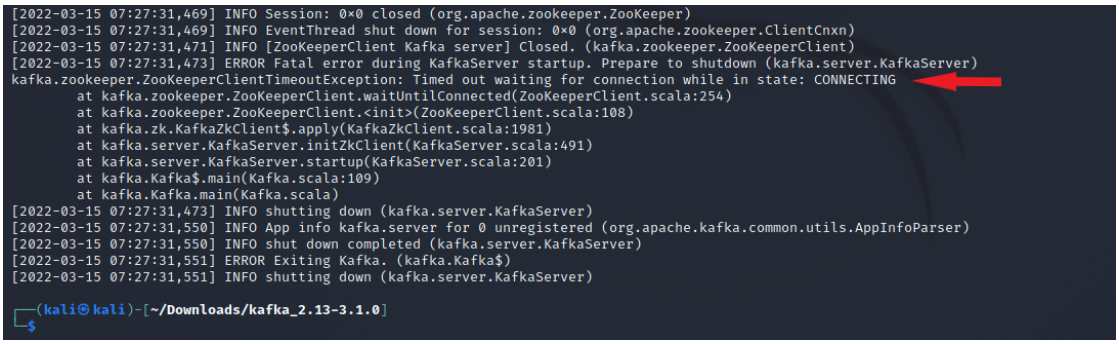
Kun tarvittavat alkutoimenpiteet on tehty, kehitysympäristö on valmis Kafka-klusterin käynnistykseen. Seuraavaksi katsotaan, mitä kaikkea siihen liittyy.

3.1 Kafka-klusterin käynnistys

Ensimmäiseksi ladataan Kafkan sivuilta asennuspaketti ja puretaan se halutuun kansioon. Tämän jälkeen ajetaan Kafka-välittäjän asennustiedosto (kuva 13). Tämä on helpointa tehdä juuri puretun kohdekansion sisällä, joten navigoidaan komentorivillä ensin sinne.

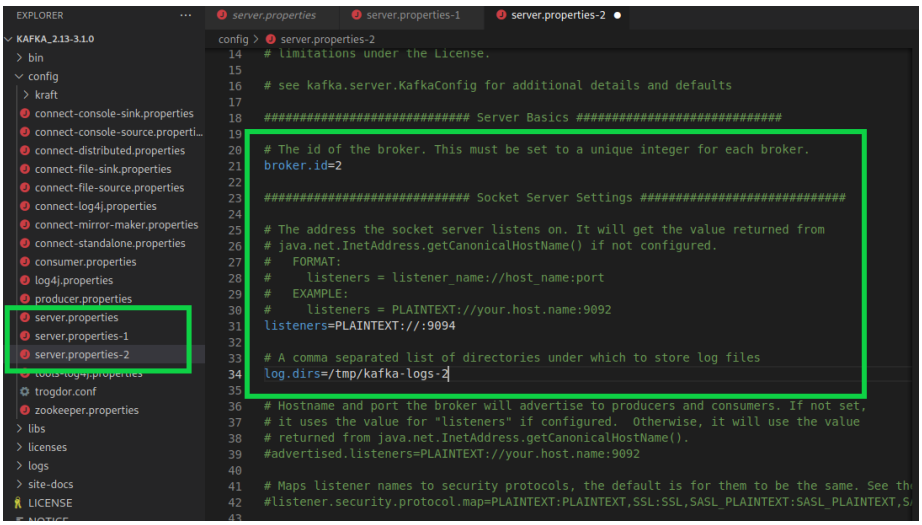
Kuva 13. Ajetaan Kafka-välittäjän asennusskripti

Kuva 14 osoittaa asennuksen ensimmäisen virheen, joka johtuu siitä, että asennusohjeita ei ole noudatettu säntillisesti. Paikallisessa asennuksessa on tärkeää, että palvelimet käynnistetään oikeassa järjestyksessä. Koska Zookeeper on koko Kafka-klusterin ylläpitäjä, täytyy sen olla käynnissä ennen muiden palvelimien asentamista.



Kuva 14. Zookeeper ei ole vielä käynnissä, joten Kafka-palvelimen asennus epäonnistuu.

Zookeeperin käynnistys tapahtuu samalla tavalla kuin välittäjän, eli ajamalla asennusskripti. Tämän jälkeen voidaan varsinainen Kafka-välittäjä käynnistää. Tässä vaiheessa käynnistetään kerralla kolme välittäjää, jotta voidaan samalla testata Kafka-klusterin vikasietoisuutta ja Zookeeperin roolia paikallisesti. Uusien välittäjien ominaisuudet on helpointa kopioida alkuperäisestä välittäjästä. Kuvan 15 mukaisesti vain kolmea ominaisuutta täytyy muuttaa, jotta välittäjät voidaan käynnistää erillisinä palvelimina. Näitä ovat välittäjän id, kuunneltavan portin numero sekä lokitiedoston nimi.



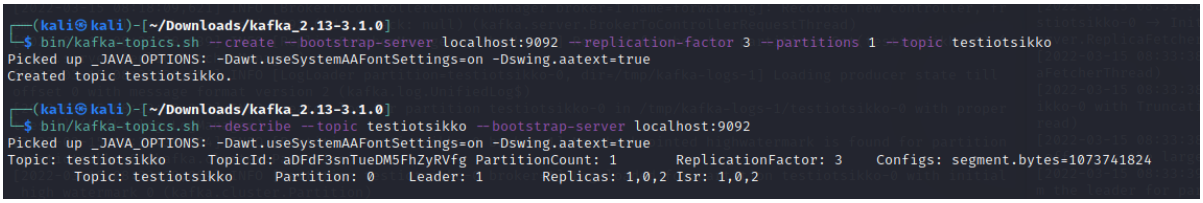
Kuva 15. Nopein tapa luoda lisää välittäjiä (broker) on kopioida asetukset alkuperäisestä. Tämän jälkeen muutetaan tarvittavat ominaisuudet vastaamaan uutta välittäjää.

Kun ominaisuudet on päivitetty, käynnistetään jokainen välittäjä omaan konsoliinsa. Tällä kertaa kaikki meni kuten pitikin ja kolme välittäjää käynnistyi ilman ongelmia. Kuvassa 16 kaikki kolme välittäjää ja Zookeeper toiminnassa.

Kuva 16. Zookeeper (vasen yläkulma) ja kolme välittäjää id:llä 0, 1 ja 2

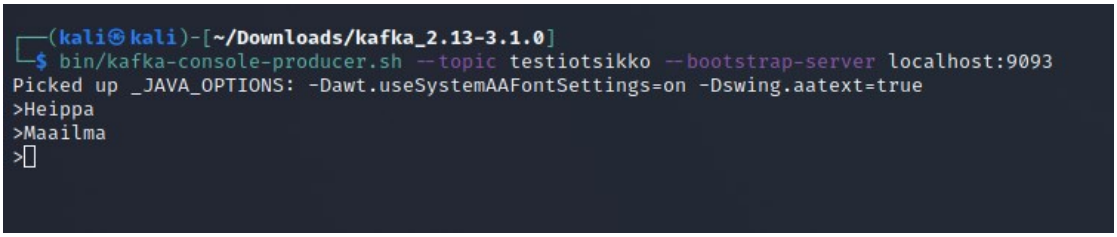
Kafka-klusteri on nyt käynnissä, mutta sen käytännön toiminnan testaamiseksi tarvitaan vielä datan tuottaja ja käyttäjä. Kafkan paketti sisältää molemmista esimerkkinä yksinkertaiset konsolisovellukset. Ennen näiden asennusta tulee kuitenkin luoda otsikko (topic), johon tiedot tallennetaan. Otsikkoa luodessa määritellään portti eli välittäjä, kopioiden (replication-factor) ja osioiden lukumäärä sekä otsikon nimi. Kopioiden määrä on enintään klusterissa käytössä olevien välittäjien lukumäärä.

Kuvassa 17 on ensin otsikon luomiskomento tarvittavilla parametreilla. Kun otsikko on luotu onnistuneesti, voidaan ajaa ”kuvaus” eli describe-metodi, joka näyttää tietoja otsikosta. Otsikon kuvauksessa näkyvä Leader, Replicas ja Isr (In-sync-replicas) viittaa kopioiden määrään klusterissa sekä ns. johtajavälittäjään. Johtajavälittäjä on vastuussa tietojen tallentamisesta ja kopioimisesta muihin välittäjiin. Isr tarkoittaa synkronisoitujen, käytännössä toimivien, välittäjien määrää sekä niiden järjestystä klusterissa.



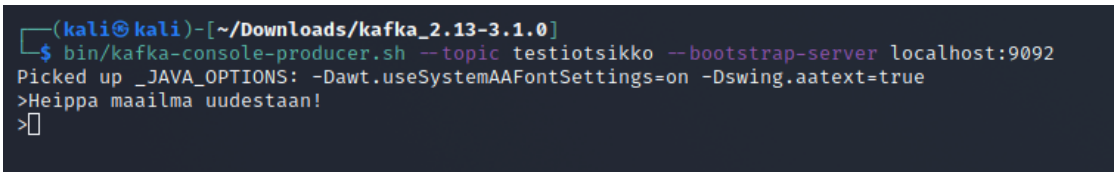
Kuva 17. Ensimmäinen skripti luo otsikon. Toinen skripti tulostaa otsikon ominaisuudet ja antaa tietoja myös klusterin tilasta

Nyt käynnissä on oletettavasti vikasietoinen Kafka-klusteri kolmella välittäjällä. On aika testata toiminta käytännössä kahdella konsolisovelluksella, tuottajalla ja käyttäjällä. Sovellukset käynnistetään myös omissa konsoleissaan ajamalla käynnistysskripti. Kuva 18 näyttää käynnistetyn tuottajan ja sillä kirjoitetun viestin, joka lähetetään Kafka-klusterille. Tuottajaa käynnistäessä nimetään otsikko, johon tiedot lähetetään sekä jonkin klusterissa olevan välittäjän portin numero.



Kuva 18. Tuottajasovelluksessa kirjoitetaan viesti ja lähetetään se klusterille tallennettavaksi

Seuraavaksi käynnistetään vielä toinen tuottaja ja yhdistetään se eri välittäjään eli käytännössä ohjataan data eri porttiin. Kirjoitetaan uuteen tuottajaan uusi viesti tallennettavaksi ja katsotaan, kuinka klusteri käsittelee tapahtumat (kuva 19).



Kuva 19. Lähetetään yksi viesti uudesta tuottajasta samassa klusterissa olevalle toiselle välittäjälle

Viimeisenä käynnistetään käyttäjäsovellus, jonka tehtävä on yksinkertaisesti lukea klusterissa olevat viestit. Jälleen määritellään otsikko, jonka tietoja halutaan käyttää, portin numero sekä erikseen vielä määritys ”from-beginning”,

mikä tulostaa viestit vanhimmasta uusimpaan. Kuvassa 20 näkyy käyttäjän klusterilta hakemat viestit tulostettuna.

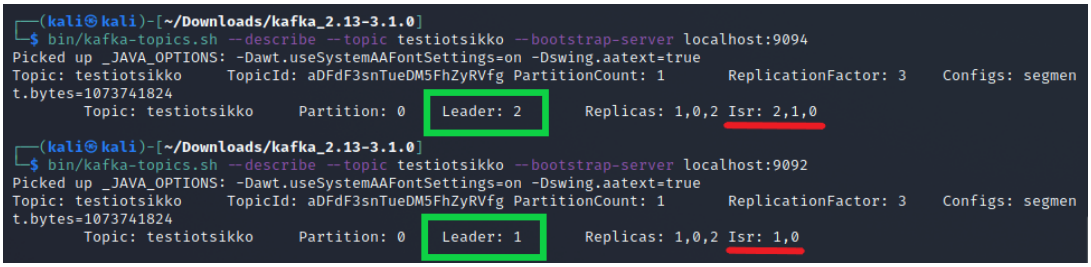
Kuva 20. Tuottajan klusterilta hakemat viestit tulostettuna konsoliin

Kafka-klusterin toiminta on nyt todennettu tuottajan ja käyttäjän avulla toimivaksi. Seuraavassa luvussa testataan Kafka-klusterin toimintaa vikatilanteissa.

3.2 Vikasietoisuuden testaus

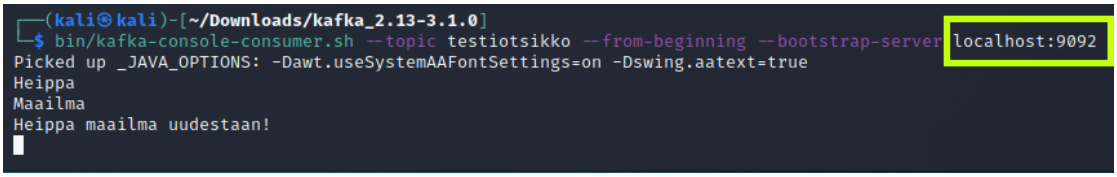
Kafkan tulisi olla myös vikasietoinen. Se tarkoittaa tässä tapauksessa sitä, että vaikka jokin välittäjistä eli palvelimista kaatuisi, dataa ei häviä. Edellisessä luvussa testattiin yhteyksiä klusterin eri välittäjiin vaihtamalla porttia mihin yhteyttä otetaan. Seuraavaksi testataan toimintaa niin, että yksi palvelimista on pois päältä tai kaatunut. Zookeeperin pitäisi huolehtia tarvittaessa uuden ”johtajan” valinnasta ja klusterin toiminnan pysyä muuten ennallaan.

Välittäjät toimivat porteissa 9092–9094 ja niiden id:t ovat pienimmästä portista suurimpaan 0,1 ja 2. Eli nykyisen johtajavälittäjän id on 2 ja se toimii portissa 9094. Kun kyseinen palvelin eli välittäjä sammutetaan, aloittaa Zookeeper tarvittavat toimenpiteet. Kuvassa 21 on ajettu describe-metodi ennen ja jälkeen palvelimen sammuttamisen. Tässä välissä Zookeeper on suorittanut ns. äänestyksen uudesta johtajavälittäjästä. Uuden johtajan id on 1. Samalla huomataan, että Isr eli toimivien kopioiden määrä on vaihtunut kolmesta kahteen.



Kuva 21. Johtajavälittäjän sammumisen seurauksena Zookeeper suorittaa uuden johtajan valinnan automaattisesti

Kun välittäjä portissa 9094 on sammutettu, tehdään käyttäjällä tietojen haku toimivasta välittäjästä. Tiedot tulostuvat konsoliin kuten aiemminkin eli Kafka-klusteri on pysynyt toimintakykyisenä jopa johtajavälittäjän vaihtumisen jälkeen (kuva 22).



Kuva 22. Portissa 9094 toimiva välittäjä on sammutettu. Klusterin toiminta pysyy ennallaan välittäjän alasajosta huolimatta.

Testien perusteella Kafka-klusteri toimii odotetulla vikasietoisella tavalla. Seuraavaksi tehdään Kafkan käyttöönotto Kubernetes-ympäristössä ja tutustutaan paikallisen ja Kubernetes-käyttöönoton eroihin käytännössä.

4 KAFKAN KÄYTTÖÖNOTTO KUBERNETES-ALUSTALLA

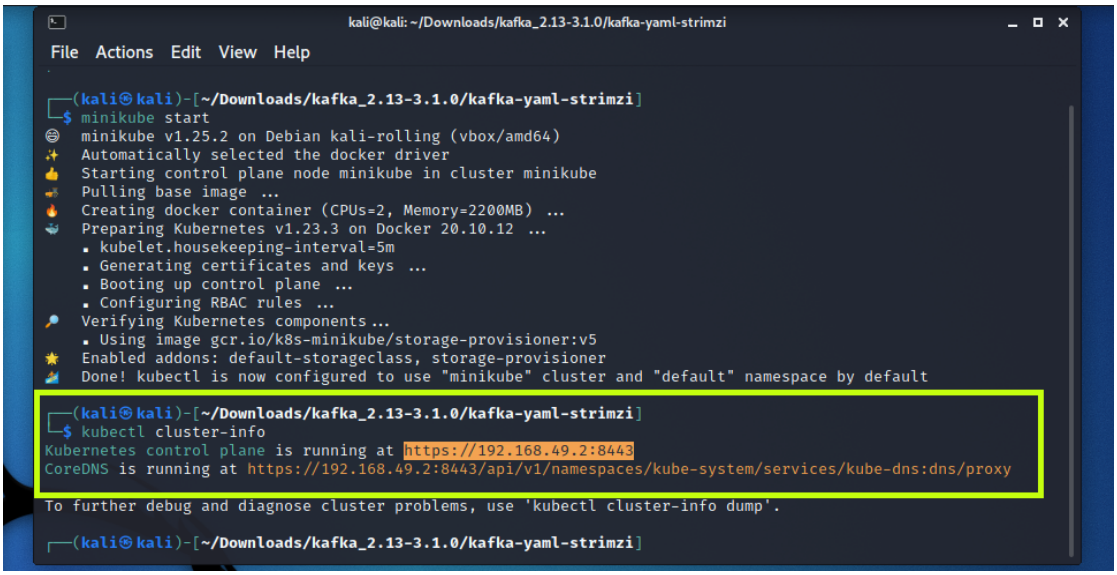
Tässä luvussa käydään aluksi läpi Kubernetes-klusterin asennus paikallisesti virtuaalikoneeseen. Ajoympäristö on siis sama kuin aiemmin Kafka-klusterin asennuksessa. Tämän jälkeen pyritään asentamaan edellisessä luvussa esitelty kolmen välittäjän muodostama Kafka-klusteri Kubernetes-ympäristöön. Kun klusteri on käynnissä, testataan toiminta käyttämällä Kafkan tarjoamia demosovelluksia, tuottajaa (producer) ja käyttäjää (consumer). Viimeisenä testataan klusterin vikasietoisuus.

4.1 Kubernetes-klusterin asennus

Ensimmäiseksi asennetaan apuohjelma Minikube, jonka avulla pystytään käynnistämään Kubernetes-klusteri paikallisesti. Minikubea käytettäessä voidaan simuloida kokonaista Kubernetes-klusteria yksinkertaistetussa muodossa. Se sopii siis hyvin erilaiseen testaukseen ja konttiorkestroinnin kokeilemiseen käytännössä (Ionos 2020). Minikuben käyttöä varten asennetaan Kuberneteksen komentorivityökalu kubectl.

Kun Minikube on asennettu, klusterin käynnistys on hyvin suoraviivaista. Kun asennus on valmis, voidaan klusterin tila tarkistaa vielä ”kubectl cluster-info”-

komennolla. Kuvassa 23 näkyy tuttu termi teoriaosuudesta, control plane eli ohjaustaso. Klusteri on nyt käynnissä.



Kuva 23. Kubernetes-klusteri on valmis käyttöön.

Klusterin käynnistyksen jälkeen luodaan uusi nimiavaruus (namespace) ”kafka”. Nimiavaruuden tarkoitus on yksinkertaisesti luoda oma eristetty ryhmänsä siinä käytettäville resursseille. Tästä on erityisesti hyötyä silloin, kun samaa klusteria käyttää useampi sovellusprojekti tai kehitystiimi (Kubernetes 2021a). Uuteen nimiavaruuteen asennetaan seuraavaksi varsinainen Kafka-klusteri.

4.2 Apache Kafka-klusterin käyttöönotto Kubernetes-alustalla

Kafka-klusterin asennuksessa käytetään apuna Strimzi-projektia. Strimzi on avoimen lähdekoodin projekti, jonka avulla Kafkan asennus Kubernetes-klusteriin voidaan toteuttaa yksinkertaisesti ja nopeasti (Strimzi 2022). Kun nimiavaruus on luotu ja Strimzi asennettu, luodaan käyttöönotto (deployment).

Käyttöönotto on käytännössä tiedosto, missä määritellään käyttöönoton ominaisuudet, tallennustila ja asennettavat ohjelmat yms. Kubernetes käyttää YAML-tiedostoa käyttöönoton kuvaamisessa. Kuva 24 näyttää tässä työssä käytettävän käyttöönoton ominaisuudet YAML-muodossa. Klusteriin asennetaan siis kolme Kafka-välittäjää, kolme Zookeeperia sekä osoitetaan näille tal-

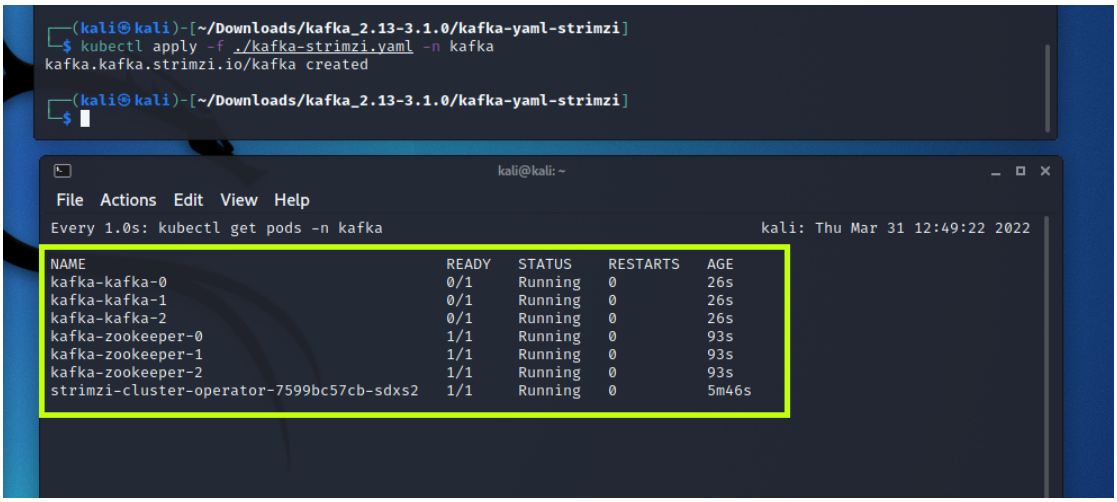
lennustila. Käyttöönottoon voidaan myös tehdä muita Kafka-asetuksia, esimerkiksi ”default replication factor”, mikä kuvaa otsikoiden kopioiden määrää välittäjissä.

Kuva 24. Käyttöönotto kuvataan omassa tiedostossaan YAML-kielellä.

Kun asetukset ovat omasta mielestä valmiit, ajetaan kubectl-komento ”apply -f + yaml-tiedoston nimi”, tämän jälkeen palvelut alkavat käynnistyä omiin kapseleihinsa (pod). Tämä tapahtuu siis, jos kaikki on kunnossa. Testatessa tuli vastaan useampikin konflikti, mutta onneksi virheviestit ovat selkeitä ja niiden avulla ongelmat sai melko nopeasti korjattua. Esimerkiksi YAML-tiedostossa käytetty api-versio oli vanhentunut, joten palvelun käynnistys ei onnistunut. Kun olin löytänyt ja korjannut oikean version YAML-tiedostoon, palvelut käynnistyivät ilman virheitä.

Kuvassa 25 näkyy käynnistymässä oleva Kafka-klusteri. Strimzi käynnistää myös yhden podin klusterin operointia helpottaville työkaluille, tällaista kokonaisuutta kutsutaankin osuvasti operaattoriksi (Kubernetes 2022b). Sen ansiosta mm. Zookeeper-kapselit käynnistyvät ensin ja vasta kun ne ovat täysin

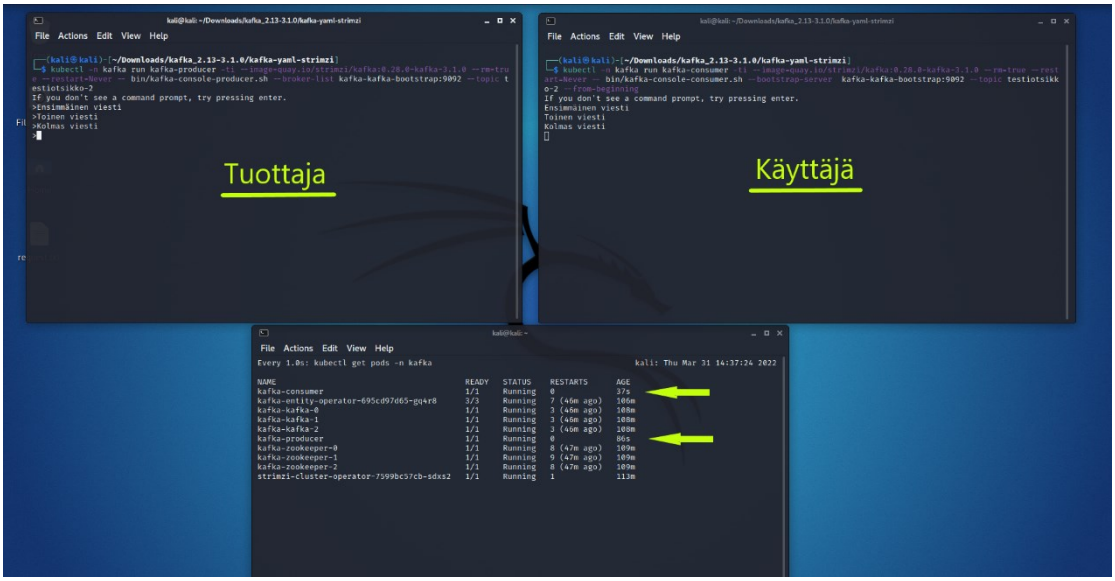
valmiita, käynnistyvät Kafka-kapselit. Tämä estää turhien konfliktien syntymisen käynnistyksen aikana esimerkiksi siitä syystä, että Kafka-välittäjä ei saisi yhteyttä Zookeeperiin.



Kuva 25. Kafka-klusteri Kubernetes-ympäristössä. Jokainen rivi vastaa yhtä podia, jossa palvelua ajetaan.

Kun Kafka-klusteri on kokonaan käynnistynyt, voidaan suorittaa toiminnan testaus samalla tavalla kuin paikallisessa asennuksessa. Käytetään siis Kafkan tarjoamia demo-konsolisovelluksia tuottajan ja käyttäjän roolissa. Jokainen käynnissä oleva Kafka-kapseli sisältää nämä ohjelmat, joten yksi tapa olisi navigoida kapselin sisään ja käynnistää ohjelma sieltä samalla tavalla kuin paikallisessa käyttöönotossa. Testasin toimintaa myös tällä tavalla, mutta halusin esitellä myös Strimzin tarjoaman vaihtoehdon toteuttaa testaus.

Strimzin avulla voidaan ajaa skriptit, jotka käynnistävät sekä tuottajan, että käyttäjän omiin kapseleihinsa. Tämä toimintatapa on myös lähempänä ns. tuotantokelpoista ratkaisua, jossa asiakassovellus (tuottaja/käyttäjä) todennäköisesti toimisi myös omassa kapselissaan. Kuvassa 26 näkyy se, että tuottajalle ja käyttäjälle on tosiaan käynnistetty omat kapselinsa. Testaus on myös suoritettu samaan tapaan kuin paikallisessa käyttöönotossa. Kuva 26 osoittaa myös sen, että tuottaja on pystynyt lähettämään viestit klusterille. Tätäkin olennaisempaa on se, että käyttäjä on onnistuneesti hakenut viestit ja ne tulostuvat konsoliin. Viestien lähetys ja tulostus toimi lähes reaaliaikaisesti, kuten Kafkan olettaisikin toimivan testikäytössä.



Kuva 26. Tuottaja ja käyttäjä käynnistyvät Strimzin ansiosta omiin kapseleihinsa. Kafka-klusterin toiminta voidaan nyt testata samalla tavalla kuin paikallisessa Kafka-asennuksessa.

Viimeisenä testataan sitä, kuinka Kubernetes ja Kafka palautuvat tilanteesta, missä yksi tai useampi kapseleista kaatuu. Ajoin kubectl-komennon, joka käytännössä poisti 2/3 Kafka-kapseleista. Yksi näistä oli vastuussa Kafka-käyttäjän ajamisesta, joten samalla kun kapseli kaatui, kaatui myös käyttäjäsovellus. Näin pitikin käydä. Kuitenkin lähes välittömästi kapseleiden poiston jälkeen, Kubernetes ryhtyi toimeen. Se käynnisti uudet Kafka-kapselit vanhojen tilalle. Tämän jälkeen ajoin käyttäjän käynnistysskriptin uudelleen ja totesin viestien olevan edelleen tallessa eli mitään dataa ei menetetty, vaikka osa kapseleista oli hetken alhaalla. Tämä oli toivottu lopputulos ja osoitti Kafka-klusterin vikasietoisuuden myös Kubernetes-ympäristössä.

4.3 Tulosten arviointi

Kafka-klusterin käynnistys Kubernetes-ympäristössä osoittautui yllättävän haastavaksi. Matkan varrella oli erilaisia ongelmia esimerkiksi virtuaalikoneen resurssien riittävyyden kanssa. Kafka-klusteri vaati enemmän käyttömuistia toimiakseen, mitä etukäteen oletin ja tämä aiheutti useamman kaatumisen, kunnes keksin mistä ongelma johtuu. Lisäksi valmiiden YAML-tiedostojen käyttäminen pohjana vaati ylimääräistä ongelmanratkaisua, koska niihin liittyvä dokumentaatio tai muu ohjeistus ei aina ollut ajan tasalla.

Tuloksena syntyi yksinkertainen Kafka-klusterin käyttöönotto, jolla kuitenkin voidaan todeta sen toimivuus Kubernetes-alustalla. Demo-ohjelmien tilalla

olisi voinut käyttää myös itse koodattua asiakassovellusta ja katsoinkin sellaiseen mallia eri lähteistä. Koin kuitenkin, että asiakassovelluksen rakentaminen itse tähän tarkoitukseen olisi mennyt työn aiheen ulkopuolelle ja siitä voisi tehdä jopa oman opinnäytetyönsä.

Toinen mielenkiintoinen ajatus olisi kytkeä toimivaan klusteriin jokin olemassa oleva palvelu syöttämään dataa sisään ja toinen palvelu lukemaan sitä. Toimeksiantajan puolelta tällaisia olisivat esimerkiksi eri palveluiden vikatilanteista ilmoittavat palvelut. Ne voitaisiin kytkeä Kafkaan ja saada vikaviestit keskitetysti ohjattua yhteen paikkaan mm. tilastointia ja muuta jatkokäsittelyä varten. Oma ajatukseni olisi esimerkiksi kytkeä Kafka osaksi yrityksen käyttämää Gitflow-työskentelytapaa. Uskoisin, että näin voitaisiin kerätä dataa esimerkiksi eri projektien kehitysaktiivisuudesta tutkimatta erikseen jokaisen projektin Github-repositoriota.

Toimeksiantajayrityksen näkökulmasta tuotantokelpoisen Kafka-klusterin asennus ja ylläpito Kubernetes-alustalla vaatii kuitenkin vielä paljon työtä tämän raportissa esitetyn käyttöönoton lisäksi. Ylläpidon tulisi tapahtua jollakin pilvialustalla, kuten Metatavun käyttämällä AWS:lla (Amazon Web Services). Näin saataisiin ainakin palveluntarjoajan valmiit tietoturvaratkaisut suoraan käyttöön eikä kaikkea tarvitse rakentaa ja valvoa itse. Lisäksi työtä vaatii vielä palvelun skaalautumisen automatisointi tarpeen mukaan.

5 PÄÄTÄNTÖ

Valitsin opinnäytetyön aiheen toimeksiantajan ehdotuksesta. Tarjolla oli muitakin, lähempänä koulutusohjelmaani olevia aiheita. Päädyin tähän osin juuri siksi, että pääsin näin tutustumaan itselleni täysin uuteen asiaan, omin päin. Tästä oli seurauksena se, että käytännön osuus ei toteutunut ihan niin laajana, kuin alun perin olin suunnitellut. Ajattelin pääseväni Kafkan kanssa vielä syvemmälle, mutta työn edetessä ymmärsin sen olevan liian suuri pala yhteen raporttiin sisällytettäväksi. Hyvänä puolena tästä jäi vielä nälkää opiskella aihetta lisää ja varsinkin käyttöönotto Kubernetes-alustalla on sellainen, johon toivon syventyväni lisää myös jatkossa.

Toimeksiantaja oli tyytyväinen siitä, että raportti pysyi helppolukuisena ja rakenteeltaan selkeänä. Koen onnistuneeni hyvin tämän osalta. Kyse on kuitenkin myös Metatavun sisällä toistaiseksi melko vähän käytetystä työkalusta, joten tiedossa oli, että alun perin toivottu tuotantokelpoinen toteutus saattaa vaatia useamman opinnäytetyön panoksen. Työtä pystytään toivottavasti käyttämään hyväksi ainakin Kafkaan ensi kertaa tutustuttaessa.

Koen oppineeni työtä tehdessä paljon. Kubernetes oli minulle etäisesti tuttu työharjoittelun ajalta. Siihen liittyvää konttiteknologiaa on käyty myös hieman koulussa läpi, joten sekään ei täysin vieras ollut. Tätä työtä tehdessä pääsin kuitenkin pureutumaan molempiin teorian tasolla syvemmälle ja koen ymmärtäväni sekä tekniikan, että orkestrointityökalun toiminnan nyt paremmin kuin aiemmin. Apache Kafka oli minulle täysin ”musta aukko” ennen tämän työn aloitusta. Oli erittäin mielenkiintoista oppia työkalun käyttötarkoituksista ja perustoiminnasta. Jatkossa pääsen toivottavasti tutustumaan myös Kafkaa käyttävien asiakassovelluksien kehittämiseen.

Kaiken kaikkiaan opinnäytetyö oli onnistunut projekti ja olen tyytyväinen siitä saatuihin tuloksiin ja siihen mitä opin matkan aikana. Olen myös tyytyväinen, että raportti valmistui aikataulussa. Toivottavasti tästä hyötyy moni muukin.

LÄHTEET

Abildskov, J. 2017. Mitä on DevOps. WWW-dokumentti. Saatavilla: https://www.eficode.com/fi/blog/mita-on-devops [viitattu 14.12.2021].

Amanse, A. 2020. Why should you use microservices and containers? WWW-dokumentti. Saatavissa: https://developer.ibm.com/articles/why-should-we-use-microservices-and-containers/ [viitattu 4.2.2022].

AWS. 2022. What is DevOps? WWW-dokumentti. Saatavissa: https://aws.amazon.com/devops/what-is-devops/ [viitattu 12.1.2022].

Berglund, T. 2020. Apache Kafka Fundamentals. Videoleike. Saatavissa: https://www.youtube.com/watch?v=B5j3uNBH8X4&ab_channel=Confluent [viitattu 25.2.2022].

CloudZero. 2021. What Is Container Orchestration? Here's Your Complete Guide. WWW-dokumentti. Saatavissa: https://www.cloudzero.com/blog/container-orchestration [viitattu 14.2.2022].

Code Factory. 2021. What is Apache Kafka. WWW-dokumentti. Saatavissa: https://www.codefactorygroup.com/2021/05/28/what-is-apache-kafka/ [viitattu 23.2.2022].

Develop Paper. 2021. How to run Apache Kafka on Windows. WWW-dokumentti. Saatavissa: https://developpaper.com/how-to-run-apache-kafka-on-windows/ [viitattu 9.3.2022].

Dhanushka, D. 2021. Anatomy of an Event Streaming Platform – Part 1. WWW-dokumentti. Saatavissa: https://medium.com/event-driven-utopia/anatomy-of-an-event-streaming-platform-part-1-dc58eb9b2412 [viitattu 22.2.2022].

Fadilpašić, S. 2020. Microservice architecture growing in popularity, adopters enjoying success. WWW-dokumentti. Saatavilla: https://www.itproportal.com/news/microservice-architecture-growing-in-popularity-adopters-enjoying-success/ [viitattu 2.2.2022].

Goyal, M. 2020. Data Streaming With Apache Kafka. WWW-dokumentti. Saatavilla: https://medium.com/swlh/data-streaming-with-apache-kafka-e1676dc5e975 [viitattu 4.1.2022].

Groll, J. 2021. 4 Container Orchestration Security Concerns. WWW-dokumentti. Saatavilla: https://devops.com/4-container-orchestration-security-concerns/ [viitattu 26.2.2022].

IBM. 2021b. Containers. WWW-dokumentti. Saatavissa: https://www.ibm.com/cloud/learn/containers [viitattu 31.1.2022].

IBM. 2021a. Container Orchestration. WWW-dokumentti. Saatavissa: https://www.ibm.com/cloud/learn/container-orchestration [viitattu 12.1.2022].

Ionos. 2020. Minikube: Kubernetes made simple. WWW-dokumentti. Saatavissa: https://www.ionos.com/digitalguide/server/tools/kubernetes-minikube/ [viitattu 31.3.2022].

Jones, D. 2018. Containers vs. Virtual Machines (VMs): What's the Difference? WWW-dokumentti. Saatavissa: https://www.redhat.com/en/topics/containers/what-is-kubernetes [viitattu 12.1.2022].

Kafka. 2022. Apache Kafka Quickstart? WWW-dokumentti. Saatavissa: https://kafka.apache.org/quickstart [viitattu 15.3.2022].

Kubernetes. 2021a. Namespaces. WWW-dokumentti. Saatavissa: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/ [viitattu 1.4.2022].

Kubernetes. 2021b. Operator pattern. WWW-dokumentti. Saatavissa: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/ [viitattu 13.4.2022].

Kubernetes. 2021c. Using kubectl to Create a Deployment. WWW-dokumentti. Saatavissa: https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/ [viitattu 16.2.2022].

Kubernetes. 2021d. Using Minikube to Create a Cluster. WWW-dokumentti. Saatavissa: https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/ [viitattu 15.2.2022].

Kubernetes. 2021e. Viewing Pods and Nodes. WWW-dokumentti. Saatavissa: https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/ [viitattu 15.2.2022].

Kubernetes. 2021f. What is Kubernetes? WWW-dokumentti. Saatavissa: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ [viitattu 8.2.2022].

Lashawn, A. 2021. Comparing Top Container Software Options for 2021. WWW-dokumentti. Saatavilla: https://scoutapm.com/blog/container-service-tools [viitattu 20.1.2022].

Li, S. 2020. He Left His High-Paying Job At LinkedIn And Then Buit A $4.5 Billion Business In A Niche You've Never Heard Of. WWW-dokumentti. Saatavilla: https://www.forbes.com/sites/stevenli1/2020/05/11/confluent-jay-kreps-kafka-4-billion-2020 [viitattu 23.2.2022].

Narkhede, N., Palino, T. & Shapira, G. 2017. Kafka: The Definitive Guide. USA: WordCo Indexing Services, Inc.

Red Hat. 2020. What is Kubernetes? WWW-dokumentti. Saatavissa: https://www.netapp.com/blog/containers-vs-vms/ [viitattu 20.1.2022].

Red Hat. 2018. What's a Linux container? WWW-dokumentti. Saatavissa: https://www.redhat.com/en/topics/containers/whats-a-linux-container [viitattu 20.1.2022].

Smirnov, D. 2022. Sovelluskehittäjä. Keskustelu 18.2.2022. Metatavu Oy. Mikkeli.

Strimzi. 2022. Strimzi Quick Start guide. WWW-dokumentti. Saatavissa: https://strimzi.io/docs/operators/latest/quickstart.html [viitattu 4.4.2022].

Vennam, S. 2019. Container Orchestration Explained. Videoleike. Saatavissa: https://www.youtube.com/watch?v=kBF6Bvth0zw&ab_channel=IBMTechnology [viitattu 12.1.2022].

Vmware. 2018. Why use containers vs. VMs? WWW-dokumentti. Saatavissa: https://www.vmware.com/topics/glossary/content/vms-vs-containers.html [viitattu 31.1.2022].

Way to easy learn. 2021. Apache Kafka Tutorial. WWW-dokumentti. Saatavissa: https://www.waytoeasylearn.com/learn/kafka-architecture/ [viitattu 26.2.2022].