Haaga-Helia
ammattikorkeakoulu Oy

Koneoppimismallien suoritus ja monitorointi pilvinatiivissa ympäristössä

Topi Kettunen

Opinnäytetyö
Tietojenkäsittely
2019

Tiivistelmä

Haaga-Helia ammattikorkeakoulu Oy

Tekijä(t)
Topi Kettunen

Koulutusohjelma
Tietojenkäsittely

Raportin/Opinnäytetyön nimi
Koneoppimismallien suoritus ja monitorointi pilvinatiivissa ympäristössä

Sivu- ja liitesivumäärä
28

Koneoppiminen on ollut merkittävä trendi tietojenkäsittelyn saralla viime vuosina. Tämän lisäksi pilvinatiivit ympäristöt ovat olleet seuraava askel yritysten IT-infrastruktuurien kehityksessä. Tämän vuoksi tutkimuksessani haluan yhdistää nämä aiheet ja tutkia kuinka voidaan luoda toimiva ja moderni infrastruktuuri koneoppimiselle.

Työssäni en tule ottamaan kantaa siihen, kuinka koneoppimista tulisi tehdä vaan tarkemmin siihen, miten sitä voidaan hyödyntää modernissa pilvinatiivissa ympäristössä. Työ siis etenee hyödynnettävien työkalujen esittelyllä, josta siirrytään itse ympäristön luomiseen esiteltyjä työkaluja hyödyntäen.

Pilvinatiivit ympäristöt ei välttämättä ole helpoin ratkaisu moneen asiaan, jotta niistä saataisiin suurin hyöty irti, tulisi infrastruktuurin palveluiden olla suhteellisen rankasti käytettyjä, jotta tarpeen tullen skaalautumisesta olisi yritykselle jotain hyötyä. Tämän lisäksi näiden luominen vaatii hyvin spesifiä ammattitaitoa hajautetuista arkkitehtuureista. Nämä kuitenkin vapauttavat yrityksen monoliittisen arkkitehtuurin luomista ongelmista.

Asiasanat
Koneoppiminen, DevOps, Kubernetes, metriikka, resurssienkäyttö, hajautettu laskenta, TensorFlow

Haaga-Helia ammattikorkeakoulu Oy

Käsitteet, termit ja lyhenteet	
Termi / Lyhenne	**Kuvaus**
Koneoppiminen	Tietojenkäsittelyä, mitä tietokone voi hyödyntää kaavoihin ja päättelyyn pohjautuvassa ongelmanratkaisussa eksplisiittisten ohjeiden sijaan.
Pilvinatiivi	Pilvinatiivilla applikaatiolla tai järjestelmällä tarkoitetaan applikaatiota/järjestelmää, minkä arkkitehtuuri on usein joustavampi muutoksiin ja ympäristöön perinteisiin monoliittisiin applikaatioihin/järjestelmiin verrattuna.
Monoliittinen	Monoliittisella applikaatiolla/järjestelmällä tarkoitetaan applikaatiota/järjestelmää, jonka logiikka on sulautettu yhteen ohjelmaan/järjestelmään tiukasti.
Hajautettu	Hajautetulla applikaatiolla/järjestelmällä tarkoitetaan applikaatiota/järjestelmää, jonka logiikka on hajautettu useaan eri ohjelmaa/järjestelmään, mitkä usein keskustelevat keskenään esimerkiksi verkon yli.
DevOps	DevOps tarkoitetaan kulttuuri/erilaisia käytäntöjä, minkä avulla voidaan automatisoida ja nopeuttamaan mahdollisimman paljon prosesseja, jotka liittyvät ohjelmistokehitykseen, testaukseen sekä ylläpitoon.
Aikasarjatietokanta	Tietokanta, joka on tarkoitettu aikaleimallisen tiedon tallentamiseen. Soveltuu parhaiten metriikoiden sekä erilaisten mittausten, kuten lämpötila yms., seurantaan.
Kontti	Applikaatioiden paketointitapa, missä kaikki applikaatioin riippuvuudet asennetaan kevyeen ympäristöön, ”konttiin”, minkä avulla itse suoritusympäristö voidaan pitää puhtaana eri applikaatioiden riippuvuuksista.
Klusteri	Klusterilla useamman tietokoneen rypästä, mitkä ovat yhteydessä toisiinsa verkon välityksellä sekä toimivat yhdessä. Klustereissa on usein yksi tai useampi johtajakone, minkä perusteella klusteri toimii.

Sisällys

1 Johdanto

Tiedon ja datan merkitys nykypäiväisessä liiketoiminnassa on kasvattanut merkitystään viime vuosina kiihtyvällä tahdilla. Tietoa ja dataa hyödynnetään niin uuden suunnittelussa sekä toteuttamisessa kuin myös vanhojen parantamisessa. Näiden merkitys on tästä syystä kasvanut todella merkittäväksi sekä arvokkaaksi yritysten silmissä, minkä on voinut myös nähdä viime vuosien tietotekniikan uutisissa useimmiten pahaan sävyyn kuvattuna. Tietoa ja dataa voi siis hyödyntää niin bisneskriittiseen päätöksentekoon

Vaikka tieto ja data ovat nykypäivänää suuressa merkityksessä niin päätöksenteossa kuin missä tahansa suunnitelussa, tulee näitä myös osata kerätä oikein sekä samaan aikaan hyödyntää tehokkaasti. Olen itse hyvin usein työelämässä törmännyt tilanteisiin, missä yritykset eivät joko osaa hyödyntää olemassa olevaa kerättyä dataa tai sitten osaamisen tai tietotaidon puutteesta eivät osaa kerätä sitä. Tietoa ja dataa voi siis kerätä lähestulkoon mistä vaan minkä katsotaan olevan liiketoiminnalle kannattavaa, joko sitten suoraan päätöksenteossa tai aivan yrityksen työntekijän elämää parantamassa. Tästä syystä tiedonkeruu itsessään on noussut arvokkaaksi sekä halutuksi taidoksi modernissa ketterässä maailmassa, missä pyritään hyödyntämään eri DevOps-kulttuurin piirteitä.

Koneoppiminen on myös ollut jo usean vuoden kasvava trendi tietotekniikan saralla. Tällä saralla tieto ja data ovat hyvin oleellisessa osassa. Tekoäly sekä sen kehittäminen ei itsessään ole mikään uusi ilmiö, mutta koneoppiminen itsessään on suhteellisen uusi tapa lähestyä tekoälyn opettamista. Koneoppiminen keskittyykin puhtaasti tiedonkäsittelyyn sekä päätöstentekoon tämän tiedon pohjalta, jonka avulla voidaan hyödyntää olemassa olevaa tietoa sekä dataa mitä sitten voidaan suoraan hyödyntää yrityksen omassa päätöksenteossa.

Koneoppiminen vaatii kuitenkin paljon dataa, jotta voidaan varmistua siitä, että kone pystyy tekemään mahdollisimman oikean päätöksen. Kun aletaan puhumaan tera-, peta- tai eksatavun datamääristä, alkaa koneoppiminen vaatia tietokoneelta valtavia määriä resursseja sekä aikaa. Tämän vuoksi metriikat esim. resussienkäytöstä on hyvin merkittäviä kehitykselle, koska niiden avulla voidaan ennustaa, milloin joko loppuu resurssit tai alkaa syntymään ongelmia, mikä helpottaa tämänkaltaisten ongelmien ratkaisussa.

1.1 Opinnäytetyön esittely ja rajaus

Tutkimuksen aihe määräytyi oman päivätyöni sekä kiinnostuksieni mukaan. Työskentelen tällä hetkellä DevOps konsulttina, missä työskentelen paljon erinäköisen automatisoinnin sekä kriittisen tiedonkeruun saralla. Tämän lisäksi pyrin ylläpitämään ammattitaitoani jatkuvasti opettelemalla uutta ja oleellista tietotekniikan saralta, mihin myös lukeutunee koneoppiminen. Haluan siis tutkia koneoppimisen resurssienkäyttöä kahdessa erilaisessa arkkitehtuurissa sekä, kuinka tämä resurssienkäyttö eroaa näissä arkkitehtuureissa.

Resurssienkäyttöä tulen tutkimaan keräämällä metriikoita yksittäisellä koneella tapahtuneessa laskennassa sekä kolmen koneen Kubernetes klusterissa tapahtuvassa laskennassa. Metriikoita kerään hyödyntämällä InfluxDatan TICK pinoa. Metriikat tulen sitten visualisoimaan Chronografia, minkä pohjalta voin tehdä päätelmiä resurssienkäytön eroavaisuuksista näissä arkkitehtuureissa.

Tutkimuksessa en tule ottamaan kantaa siihen, kuinka koneoppimista tulisi hyödyntää tehokkaasti vaan enemmän tulen keskittymään arkkitehtuuri puoleen ja siihen kuinka kerättyä dataa voidaan hyödyntää ja tehdä päätöksiä sen pohjalta.

2 Tutkimussuunnitelma

Käytännönläheisessä tutkimuksessani haluan tutkia erityisesti sitä, kuinka saadaan luotua pilvinatiivissa ympäristössä pyörivä alusta erilaisten koneoppimismallien suorittamiseen, mutta myös samaan aikaan tämän alustan monitorointiin. Monitorointi nousee erityisesti tämän kaltaisessa hajautetussa ympäristössä oleelliseen osaan, koska viestejä kuljettavia palveluja ja prosesseja alkaa olla hyvin nopeasti useita. Tämän vuoksi joku ratkaisu tilanteen seurantaan on erittäin merkittävässä roolissa.

Aloitan tutkimukseni keräämällä pohjustusta tutkimukselle olennaisille teorioille sekä käsitteille. Näitä ovat muun muassa monoliittisen sekä keskitetyn arkkitehtuurien olennaiset erot sekä koneoppiminen. Näiden lisäksi avaan myös myöhemmissä kappaleissa tutkimuksessani hyödyntämiäni työkaluja sekä niiden tehtävää tässä tutkimuksessa.

Tutkimuksessa tulen simuloimaan samankaltaista ympäristöä, mikä käyttäjällä voisi olla esimerkiksi jonkun pilvipalveluiden tarjoajan, kuten esimerkiksi AWS:n, GCP:n tai Azuren piirissä. Tämän simuloinnin tulen tekemään virtualisoinnin avulla, missä tulen luomaan tietokoneklusterin niin yhdellä tietokoneella kuin myös kolmella. Tulen luomaan sekä provisioimaan nämä tietokoneet Vagrant-työkalulla, millä saadaan helposti skriptattavassa muodossa luotua virtuaalikoneita usealla eri hypervisorille, kuten esimerkiksi VirtualBoxille tai Hyper-V:lle. Klusterin hallinnassa tulen hyödyntämään nykypäivän orkestraattoreiden de facto -työkalua eli Kubernetestä.

Koneoppimisen kannalta tulen hyödyntämään myös yhtä alan de facto -työkalua TensorFlowia. TensorFlowilla voidaan suorittaa erilaisia koneoppimismalleja helposti ohjelmoitavassa muodossa. TensorFlowin suoritus tulee myös tapahtumaan Kubernetesin sisällä, missä sen ajoon tuleen hyödyntämään tähän tehtyä työkalua KubeFlowta.

Klusterin monitoroinnissa tulen hyödyntämään InfluxDB:tä erinäköisten metriikoiden tallentamiseen. InfluxDB on nopea aikasarjatietokanta, joka on kehitetty nimenomaan lukuisien eri numeeristen metriikoiden tallentamiseen. Itse metriikat sitten tulen keräämään hyödyntäen Telegrafia, mikä on samoilta kehittäjiltä kuin InfluxDB. Lopuksi tulen visualisoimaan nämä kerätyt metriikat Grafanalla, mikä on tarkoitettu puhtaasti datan visualisointiin. Tulen avaamaan näitä kaikkia työkaluja myöhemmissä kappaleissa. Lopuksi teen lyhyen analysoinnin siitä, mitä tein ja myös mahdollisista lopputuloksista.

3 Laskenta hajautetussa arkkitehtuurissa

Perinteisesti arkkitehtuurit on jaettu monoliittiseen sekä hajautettuun arkkitehtuuriin. Monoliittinen arkkitehtuuri nimensä mukaisesti prosessin tapahtumat tapahtuvat monoliittisessa ympäristössä, esimerkiksi yhden prosessin sisällä. Näin ollen yksittäinen tietokonekin voidaan nähdä hajautettuna järjestelmänä, koska suurin osa koneen tapahtumista tapahtuvat useassa eri prosessissa, muodostaen siitä näin ollen hajautetun. Tämän lisäksi luonnollisesti verkko toisiinsa liitettyjä tietokoneita on myös hajautettu järjestelmä. Hajautettua järjestelmää voidaankin kuvailla joukkona eri prosesseja, jotka kommunikoivat keskenään vaihtamalla viestejä. (Lamport 1978, 1.) Lamportin (1978, 1) mukaan järjestelmä on hajautettu, kun viive prosessien välillä lähetettyjen viestien välillä ei ole mitätön verrattuna viiveeseen yhden prosessin sisällä.

Hajautettu arkkitehtuuri on usein nähty muovautuvampana arkkitehtuurina monoliittiseen verrattuna. Tämä useimmiten perustuu siihen, että järjestelmän eri komponentit ovat usein fyysisesti erotettuja toisistaan mahdollistaen näin resurssien jakamisen keskenään standardoitujen käyttöliittymien kautta. (Mosleh ym. 2016, 1.)

Viime aikoina, kun järjestelmät ovat enemmän siirtynyt pilveen hajautetun laskennan aiemmin mainitut hyödyt ja haitat ovat saavuttaneet suuremman osan kehittäjäyhteisöstä. Hajautetun laskennan haittoja usein ovat asynkroonisuus, samanaikaisuus tai tapahtumien osittainen epäonnistuminen (Conway ym. 2012, 1).

Tietotekniikassa on pitkään ollut tutkinnan kohteena se, kuinka koneet saadaan oppimaan eri asioita. Tämän myötä koneet voisivat automaattisesti suunnitella esimerkiksi resurssienkäyttönsä tarpeen vaatiessa, mutta myös merkittävimmin esimerkiksi terveydenhuolto alalla. Terveydenhuollossa tällaista älykkyyttä voitaisiin hyödyntää aiempien tapausten pohjalta tehdyillä päätöksillä esimerkiksi mitä hoitomuotoa kullekin potilaalle tulisi käyttää. Koneoppimisella on siis merkittävä rooli varsinkin datamainauksessa, missä koneoppimista voidaan hyödyntää etsiessä merkillistä tietoa suurista datamääristä. (Mitchell 1997, 1.)

Koneoppiminen ei kuitenkaan itsessään ole puhtaasti tietotekniikka vaan enemmin hyvin monialainen ala. Olennaisia aloja, mitkä vaikuttavat koneoppimiseen ovat tekoäly, tietotekniikka, filosofia, psykologia, neurobiologia ynnä muut. (Mitchell 1997, 1.)

4 Metodologia

Seuraavat malliesimerkit löytyvät omasta GitHub repositoriostani, joka löytyy osoitteesta https://github.com/topikettunen/vagrant-mono-kube.

4.1 Klusterin luonti

Tutkimuskohteena on tutkia, kuinka pilvinatiivia ympäristöä voidaan hyödyntää koneoppimismallien suorituksessa. Ympäristöjen pystytykseen olen hyödyntänyt Vagrant työkalua, jolla voidaan luoda virtuaalikoneita helposti skriptattavassa muodossa (Vagrantfile on periaatteessa ruby-tiedosto).

```
Vagrant.configure('2') do |config|
  config.vm.box = 'generic/ubuntu1804'
  config.vm.box_check_update = true

  config.vbguest.auto_update = false

  nodes.each do |node|
    config.vm.define node[:name] do |nodespec|
      nodespec.vm.hostname = node[:name]
      nodespec.vm.network "private_network", ip: node[:ip]

      nodespec.vm.provider 'virtualbox' do |vb, override|
        override.vm.provision "shell", inline: $SCRIPT
        vb.memory = node[:maxmemory]
        vb.cpus = node[:cpus]

        override.vm.synced_folder '.', '/vagrant/',
                                  owner: 'vagrant',
                                  group: 'vagrant',
                                  mount_options: ['uid=1000',
                                                  'gid=1000',
                                                  'rw',
                                                  'dmode=0755',
                                                  'fmode=0775']
      end
    end

    config.vm.define node[:name] do |nodespec|
      nodespec.vm.hostname = node[:name]

      nodespec.vm.provider 'hyperv' do |hv, override|
        override.vm.provision "shell", inline: $SCRIPT
        hv.memory = node[:memory]
        hv.cpus = node[:cpus]
        hv.maxmemory = node[:maxmemory]

        override.vm.synced_folder '.', '/vagrant/',
                                  owner: 'vagrant',
                                  group: 'vagrant',
                                  mount_options: ['uid=1000',
                                                  'gid=1000',
                                                  'rw',
                                                  'noperm',
                                                  'dir_mode=0755',
                                                  'file_mode=0775']
      end
    end
  end
end
```

Tämän lisäksi provisioin koneet yksinkertaisella shell-skriptillä, joka asentaa tarvittavat asiat koneen sisään klusterin pyöritykseen. Provisioinnissa usein käytetään työkaluja kuten Ansible, Chef tai SaltStack, mutta kun kyseessä on yksinkertainen provisiointi, en tässä tapauksessa kokenut niiden käyttöä tarpeelliseksi.

Virtuaalikoneet provisioidaan yksinkertaisella skriptillä, jota kutsutaan kohdassa ”override.vm.provision "shell", inline: $SCRIPT”. $SCRIPT on seuraavanlainen:

```
apt-get update && apt-get install -y apt-transport-https
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key
add -
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | \
     tee -a /etc/apt/sources.list.d/kubernetes.list

apt-get update
apt-get install -y kubectl docker.io
usermod -aG docker vagrant

wget https://github.com/rancher/rke/releases/download/v0.2.7/rke_linux-
amd64
mv rke_linux-amd64 rke
chmod +x rke
mv rke /usr/local/bin/

su - vagrant -c "ssh-keygen -t rsa -N '' -f ~/.ssh/id_rsa"
su - vagrant -c "cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys"

su - vagrant -c "rke up --config /vagrant/cluster.yml"
```

Skripti provisioi virtuaalikoneet asentamalla niihin kubectl sekä docker sekä rke, jonka avulla luodaan Kubernetes-klusteri. Kubectl on Kuberneteksen hallintaan tarkoitettu komentorivityökalu. Docker on taas kontitustyökalu, jonka avulla voidaan yksinkertaisesti paketoida ajettavat sovellukset eristettyihin kokonaisuuksiin ja ajaa niitä koneesta riippumatta, mikä näin ollen mitigoi useimmat migraatio-ongelmat. Tämän jälkeen asennetaan helm, mikä toimii Kubernetesin pakettimanagerina. Huomioi, että suoritan netistä haetun skriptin suoraan, mikä voi olla hyvinkin vaarallista. Koska kuitenkin pyöritän konettani virtuaalisesti, voin mitigoida nämä turvallisuusongelmat skriptin yksinkertaisamisen tähdellä. Lopuksi skripti asentaa vielä Rancherin, jonka avulla voidaan hallita helposti useita eri Kubernetes klustereita. /vagrant/utils/install-rancher.sh on seuraavanlainen:

```
#!/usr/bin/env bash

kubectl -n kube-system create serviceaccount tiller

kubectl create clusterrolebinding tiller \
        --clusterrole cluster-admin \
        --serviceaccount=kube-system:tiller

helm init --service-account tiller --history-max 200

helm repo add rancher-latest https://releases.rancher.com/server-
charts/latest

helm install stable/cert-manager \
     --name cert-manager \
     --namespace kube-system \
     --version v0.5.2

helm install rancher-latest/rancher \
     --name rancher \
     --namespace cattle-system \
     --set hostname=rancher.$(hostname -I | awk '{ print $1 }').xip.io
```

Skripti luo ensiksi ServiceAccount kube-system nimitilaan, jotta voidaan helm asentamalla tiller-palvelulle antaa oikeudet asentaa uusia applikaatioita klusteriini. Oikeudet annetaan seuraavassa komennossa, jossa luon ClusterRoleBinding tiller käyttäjälle. Lopuksi kerron Helmille, että käytetään juuri luomaani ServiceAccountia alustuksessa. Tämän jälkeen lisään uuden repositorion, jotta voin ladata rancher asennuksen suoraan verkosti helposti.

Rancher on suunniteltu tukevan vakiona turvallista verkkoliikennettä, minkä vuoksi minun täytyy ladata cert-manager, jotta voin hyödyntää Rancherin generoimia sertifikaatteja.

RKE on taas Kubernetes klusterin asennustyökalu, minkä avulla saadaan helposti luotua uusia klustereita. Klustereiden luonti Kubernetesissä oli pitkään suurehko ongelma ja vaati usein paljon työtä, mutta tämän kaltaiset aputyökalut ovat tehneet tästä huomattavasti helpompaa.

Monoliittinen ja hajautettu klusteri voidaan luoda seuraavasti:

```
$ vagrant up --provider virtualbox # tai --provider hyperv
```

Vagrant on komentorivityökalu, jonka avulla voidaan hallinoida virtuaalikoneiden elinkaarta vagrant tämän jälkeen luo koneet vagrantfile tiedoston mukaan. Vakiona olen asettanut muistin määräksi 8 GB sekä CPU-ydinten määräksi 8. Näitä voi muuttaa tarvittaessa seuraavasti:

```
$ emacs vagrantfile # avaa haluamallasi editorilla
```

jonka jälkeen editoi seuraavia arvoja:

```
# change node memory and cpu amount accordingly
memory = 4096
maxmemory = 8192
cpus = 8
```

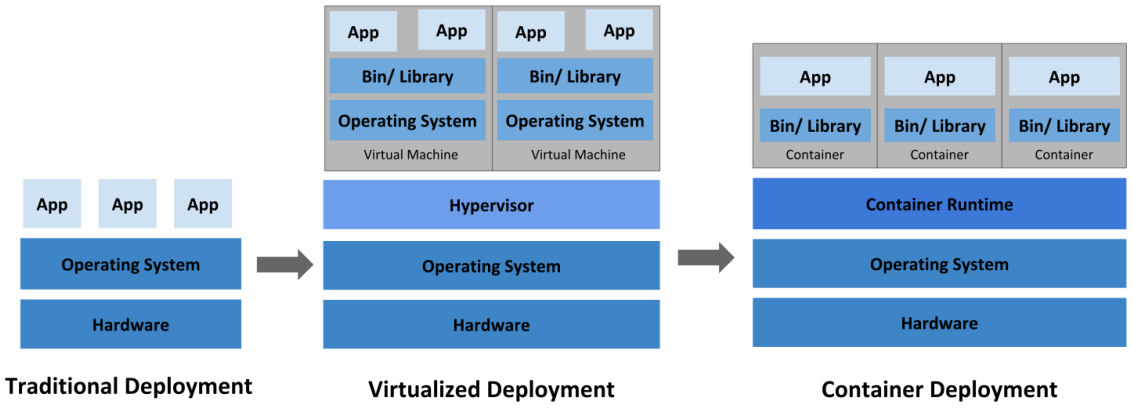
Virtuaalikoneiden provisiointiskripti on kirjoitettu siten, että koneiden käynnistyttyä ensimmäisen kerran Vagrant asentaa Rancherin, jonka avulla voidaan luoda helposti Kubernetes-klusteri ja sitten liittää muut virtuaalikoneet tähän klusteriin. Monoliittisessa ratkaisussa luodaan klusteri ja liitetään ainoastaan yksi kone tähän klusteriin.

4.2 Kubernetes

Applikaatioiden käyttöönotto on muuttunut suhteellisen lyhyessä ajassa merkittävästi. Aluksi applikaatiot asennettuun suoraan fyysiselle tietokoneelle, mikä usein aiheutti ongelmia applikaatioiden resurssien allokoinnissa, koska resurssien rajojen määrittely oli mahdotonta näihin aikoihin. (Kubernetes 2019.) Hyvä esimerkki tällaisesta tilanteesta on se, että tietokoneessa yksi applikaatio käyttää huomattavasti muita applikaatioita enemmän resursseja, minkä seurauksena muut applikaatiot alkavat alisuoriutumaan.

Käyttöönotto nopeasti kehittyi tästä virtualisoituihin ympäristöihin, missä resurssit voitiin määrätä tarkasti tarvittaessa applikaatiokohtaisesti (Kubernetes 2019). Virtuaalikoneet ovat kuitenkin oikeita koneita, mitkä vaativat itsessään tietyn verran resursseja. Tämän vuoksi tämän kaltainen käyttöönotto voi olla helposti turhan raskas, jos tarkoituksena on ajaa esimerkiksi yhta applikaatioita eristettynä.

Tämän vuoksi virtualisoidut ympäristöt tuli osittain korvaamaan kontitetut applikaatiot. Applikaatiokontit ovat pääpiirteittäin samankaltaisia virtuaalikoneisiin, mutta erona kontit jakavat isäntäkoneen käyttöjärjestelmän keskenään, kun taas jokaisessa virtuaalikoneessa on täysin oma käyttöjärjestelmä.



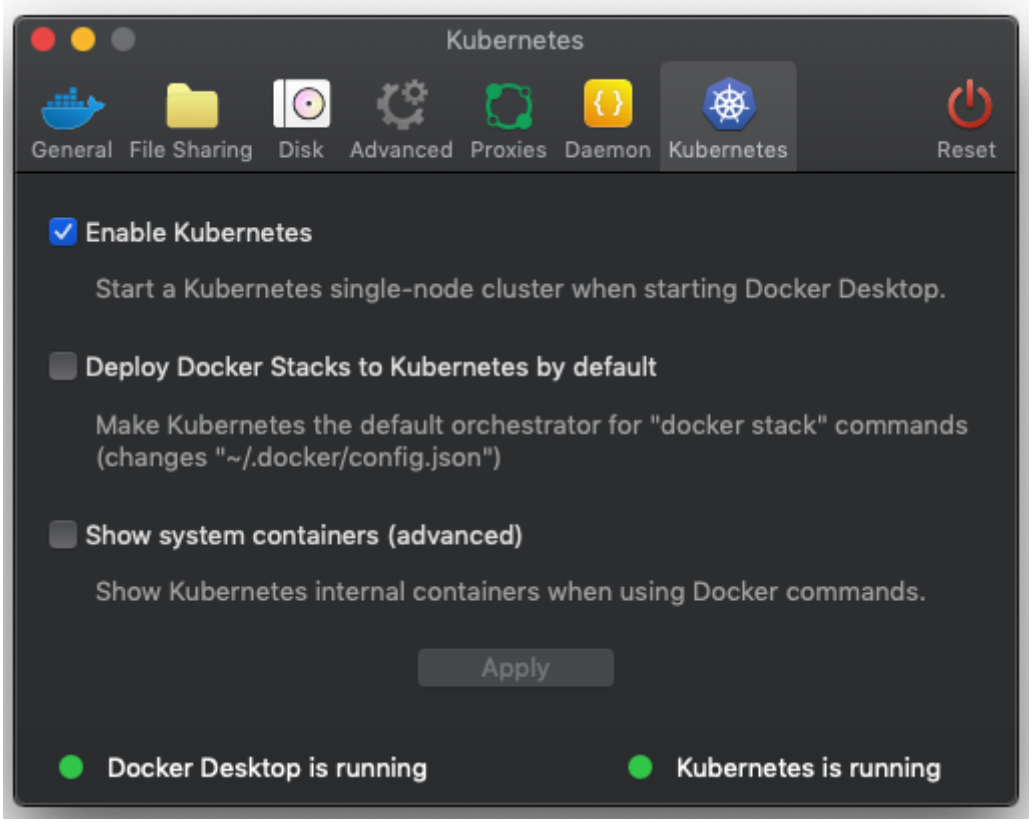
Kuva 1. Konttien evoluutio (Kubernetes 2019).

Vaikka kontit ovat hyvä keino paketoida ja ajaa sinun applikaatioitasi, tulee näitä kontteja jotenkin pystyä hallinnoimaan, sillä usein nykypäivänä tuotannossa pyörii tuhasia kontteja. Oleellisia tehtäviä mitä tulisi tapahtua suhteellisen usein on muun muassa se, että kun kontti sammuu toisen kontin tulisi käynnistyä automaattisesti. Juuri tähän kehitettiin Kubernetes, joka on tänä päivänä pitkälti de facto-työkalu konttien orkestroinnissa. Kubernetes tarjoaa loistavan alustan, missä voidaan ajaa sitkeitä hajautettuja järjestelmiä suhteellisen helposti (Kubernetes 2019).

Kubernetesin historia juontaa juurensa samankaltaiseen Googlen sisäiseen järjestelmään nimeltä Borg. Googlella on sisäisissä projekteissa hyödynnetty konttiteknologioita jo hyvin pitkään, minkä vuoksi Kubernetes pohjautuu näihin kokemuksiin ja tietotaitoon, mitä on saatu muun muassa Borgin käytöstä. (Kubernetes 2015.)

4.3 Kubectl

Kun klusteri on luotu, tarvitsee asentaa työkalut kubectl sekä helm, joita käytämme klusterin hallintaan sekä applikaatioiden asentamiseen klusterin sisälle. Windowsissa tai MacOS:ssa voidaan hyödyntää lokaalia Docker for Windows/Docker for MacOS työkalua näiden asentamiseen. Docker for Windows/Docker for MacOS voidaan ladata osoitteesta https://hub.docker.com. Kun työkalu on asennettu, voidaan asentaa lokaali Kubernetes-klusteri:



Kuva 2. Kubernetesin käyttöönotto MacOS-laitteella.

Ensimmäisen käynnistyksen yhteydessä tämä käynnistää lokaalin Kubernetes-klusterin ja asentaa samalla kaikki oleelliset työkalut tämän hallintaan, mukaan lukien kubectl. Ensimmäisen käynnistyksen jälkeen voidaan myös tässä tapauksessa sammuttaa tämä lokaalisti tehty klusteri poistamalla raksi asetusten kohdasta, koska minulla pyörii jo toinen klusteri virtuaalikoneessa/virtuaalikoneissa.

Mikäli Docker for Windows/Docker for MacOS asentaminen ei ole mahdollista voidaan asentaa myös ainoastaan kubectl työkalu. Paras tapa pelkästään kubectl asentamiseen lienee Windowsilla Chocolatey (https://chocolatey.org/) pakettimanageri tai MacOS:lla Homebrew pakettimanageri (https://brew.sh/).

Windowsilla voidaan kubectl asentaa Chocolateylla seuraavasti:

```
> choco install kubernetes-cli
```

MacOS:lla voidaan kubectl asentaa Homebrewlla seuraavasti:

```
$ brew install kubernetes-cli
```

Linuxissa (Debian/Ubuntu) kubectl voidaan taas asentaa seuraavasti:

```
$ sudo apt-get update && sudo apt-get install -y apt-transport-https
$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | \
       sudo apt-key add -
$ echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | \
       sudo tee -a /etc/apt/sources.list.d/kubernetes.list
$ sudo apt-get update
$ sudo apt-get install -y kubectl
```

4.4 Helm

Helm on avoimen lähdekoodin paketinhallintatyökalu Kubernetesille. Tämän avulla applikaatioiden asentaminen ja näiden ylläpito helpottuu huomattavasti klustereita hallittaessa.

Windowsilla voidaan hyödyntää Chocolatey pakettimanageria helm asennukseen:

```
> choco install kubernetes-helm
```

MacOS:lla voidaan hyödyntää Homebrew pakettimanageria helm asennukseen:

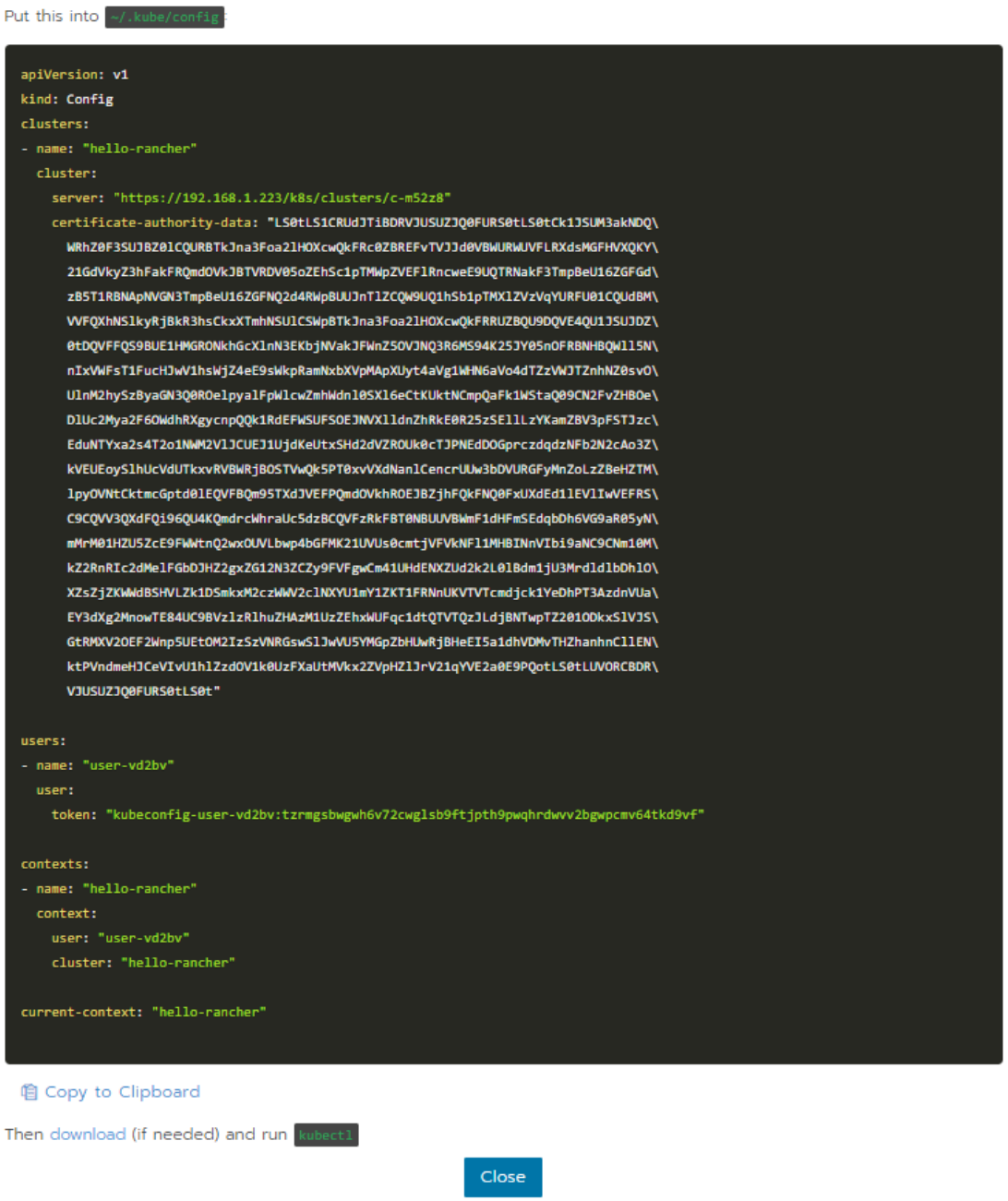
```
$ brew install kubernetes-helm
```

Linuxilla voidaan helm asentaa seuraavasti:

```
$ curl -LO https://git.io/get_helm.sh
$ chmod 700 get_helm.sh
# LUE SKRIPTIN SISÄLTÖ ENNEN AJAMISTA VARMISTAAKSESI SEN TURVALLISUUDEN
$ ./get_helm.sh
```

get_helm.sh on Helmin ylläpitämä latausskripti, joka helpottaa applikaation asennusta. Jotta mahdollisilta tietoturvaongelmilta vältyttäisiin, kannattaa netistä ladattavat skriptit lukea aina ennen niiden suoritusta.

Ennen kuin voin alkaa käyttämään helm tarvitsen yhteyden virtuaalikoneissa pyörivään klusteriin. Koska loin klusterit Rancher työkalulla, voin noutaa tarvittavan Kubernetes-klusterin konfiguraatiot suoraan Rancherin dashboardilta:



Kuva 3. Esimerkki Rancherin Dashboardin kautta haetusta kubeconfigista.

Kun konfiguraatio on kopioutu leikepöydälle, voidaan tallentaa se lokaalin koneeseen, jotta saamme yhteyden tähän klusteriin suoraan omalta koneeltani. Konfiguraatio tulee tallentaa kohteeseen ~/.kube/config, mikäli tätä ei ole olemassa sen voi luoda käsin. Tämä on vakiopolku Kubernetesin konfiguraatioille, mikäli haluat käyttää jotain toista, tulee se indikoida kubectl komennon yhteydessä.

Kun konfiguraatio löytyy polusta ~/.kube/config, voidaan testata yhteyttä klusteriini seuraavasti:

```
$ kubectl cluster-info
Kubernetes master is running at https://kubernetes.docker.internal:6443
KubeDNS is running at
https://kubernetes.docker.internal:6443/api/v1/namespaces/kube-sys-
tem/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info
dump'.
```

4.5 Metriikoiden keruu ja visualisointi

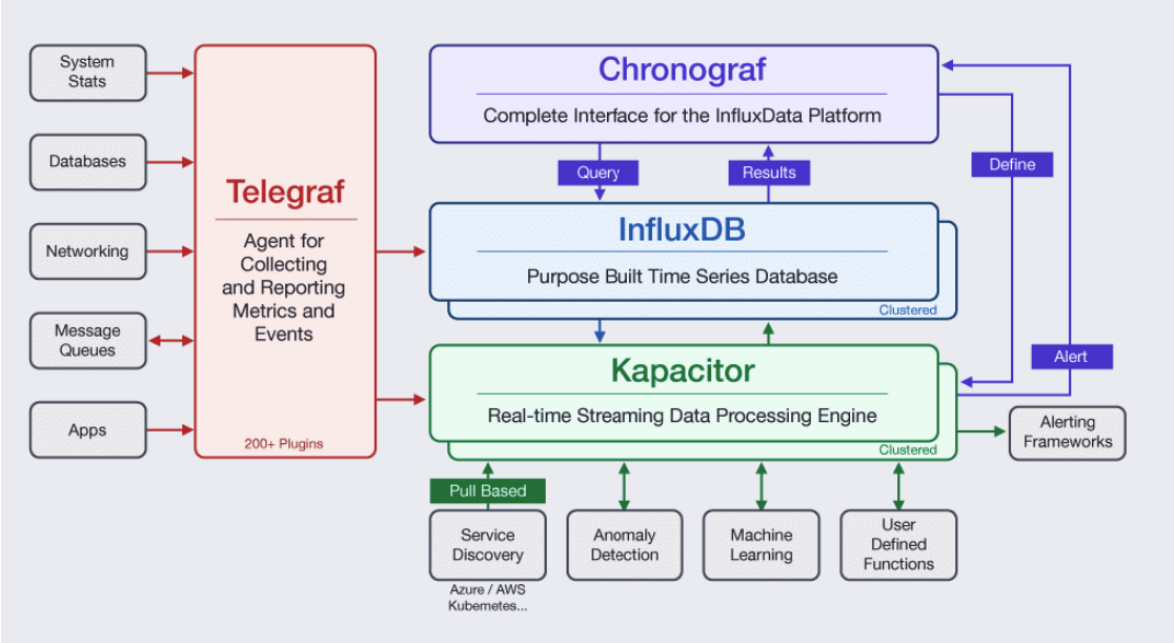
Metriikoiden keruuseen sekä visualisointiin olen valinnut työkaluiksi TICK pinon, joita käytetään metriikoiden keruussa sekä visualisoinnissa. Näiden asentaminen luotuun klusteriini tapahtuu asentamaani helm-pakettimanageria hyödyntäen. Suurin syy tähän on se, että voin minimoida näin oman koodini määrän ja hyödyntää mahdollisimman paljon olemassa olevaa yhteisön ylläpitämää koodia.

Helm-asennukset tapahtuvat "chartien" avulla, joita jo käytin asennettaessa Rancheria klusteriini. Helm ylläpitämää useita chartteja monien yleisien applikaatioiden asentamiseen. Nämä Helm yhteisön ylläpitämät chartit löytyvät osoitteesta https://github.com/helm/charts. Täältä löytyvät valmiit chartit lukuisille eri applikaatioille. Tässä tapauksessa tulen kuitenkin hyödyntämään InfluxDatan omia chartteja, sillä kun kyse on heidän omista tuotteistaan voimme luottaa siihen, että he osaavat myös konfiguroida nämä toimivaksi. Nämä chartit löytyvät osoitteesta https://github.com/influxdata/tick-charts.

Nämä chartit saa helposti Helmin käyttöön seuraavalla komennolla:

```
$ helm repo add influx http://influx-charts.storage.googleapis.com
```

4.5.1 TICK Pino



Kuva 4. Kuva TICK pinosta (InfluxData 2019)

4.5.2 InfluxDB

InfluxDB on avoimen lähdekoodin aikasarjatietokanta. InfluxDB:n pääkäyttötarkoitus on säilöä suuria määriä aikaleimallista dataa, muun muassa DevOps monitorointia, metriikoita ja reaaliaikaista analytiikkaa. (InfluxData 2019.) InfluxDB on oleellinen osa InfluxDatan lanseeraama TICK-pinoa, mihin kuuluu Telegraf, InfluxDB, Chronograf ja Kapacitor (InfluxData 2019). InfluxDB:n lisäkis tulemme myös hyödyntämään Telegrafia tästä pinosta.

Kuten aikaisemmin mainitsin, InfluxDB:tä hyödynnetään ensisijaisesti aikaleimallisen datan kanssa työskennellessä. Perinteisiä SQL-tietokantoja, kuten MySQL, MariaDB tai PostgreSQL, voidaan hyödyntää tämän kaltaisen datan kanssa työskennellessä, mutta ne eivät suoranaisesti ole tarkoitettu siihen. (InfluxData 2019.) InfluxDB:ssä aikaleima merkitsee yksittäistä datapistettä kerätyssä sarjassa. Tätä voi verrata perinteisiin SQL-tietokantoihin siten, että taulun pääavain olisi järjestelmän puolesta ennestään määritetty ja se olisi aina kellonaika. (InfluxData 2019.)

4.5.3 InfluxDB:n asennus

Helm asennuksen yhteydessä voin antaa asennukselle erilaisia konfiguraatioita, sillä usein vakioarvot asennuksille eivät toimi jokaisessa tapauksessa. Tässä tapauksessa hyödynnän kuitenkin mahdollisimman paljon asennusten vakioarvoja.

Konfiguraatiossa voisin myös määrittää, että otan datan pysyvyyden käyttöön. Pysyvyydellä tarkoitetaan sitä, että mikäli minun InfluxDB palvelu kaatuisi/sammuisi data pysyisi tallella palvelun ulkopuolella. Tämä täytyy määrittää erikseen, koska kaikki palvelut mitä Kubernetesissa ajamme ovat kontitettu, esim. Dockerin avulla. Vakioina Docker-kontit eivät ota huomioon datan säilytystä vaan se kuuluu käyttäjän itse tehdä tarvittaessa. Jätän tämän kuitenkin tekemättä, koska tarkoitus ei ole tehdä tuotantovalmista ympäristöä.

Tämän jälkeen voimme asentaa InfluxDB:n määrittelemiemme konfiguraatioiden avulla komennolla:

```
$ helm install --name data --namespace tick influx/influxdb
```

Tällä hetkellä InfluxDB näkyy siis klusterin sisällä klusterin muille palveluille, mutta ei ulospäin klusterista. Syy tähän on se, että tässä tapauksessa meidän ei tarvitse ottaa yhteyttä tietokantaan ulkopuolelta. Meille siis riittää, että voimme ottaa yhteyden siihen klusterimme sisällä. Meidän konfiguraatioillamme InfluxDB löytyy klusterin sisältä osoitteesta: http://data-influxdb.tick:8086.

4.5.4 Telegraf

Telegraf on osa samaa pinoa kuin InfluxDB. Telegrafin ensisijainen tehtävä on koota kerätä ja raportoida erilaisia metriikoita siitä järjestelmästä, mistä Telegrafia ajetaan. Telegrafia voidaan hyödyntää metriikoiden keruuseen esimerkiksi suoraan usean eri kolmannen osapuolen rajapinnan kautta tai Telegrafiin itsessään voidaan lähettää metriikoita esimerkiksi statsd-formaatissa. (InfluxData 2019.) Katso kuva 4 TICK pinosta.

4.5.5 Telegrafin asennus

Asennus voidaan ajaa suorittamalla seuraavat komennot:

```
$ helm install --name polling --namespace tick influx/telegraf-s
$ helm install --name hosts --namespace tick influx/telegraf-ds
```

Telegrafin tulen asentamaan kahdessa eri tilassa. telegraf-ds kuvastaa Telegrafin asennusta Kubernetesin DaemonSet-moodissa. DaemonSet takaa, että kaikki (tai osa) klusterin nodeista ajavat kopiota jostain tietystä podista (Kubernetes 2019). Podilla taas tarkoitamme ryhmää kontteja. Tämän kaltaisen DaemonSet asenuksen myötä me voimme tarkastella klusterin sisällä tapahtuvia asioita koko klusterin tasolla, sen sijaan telegraf-s sovelletaan taas yksittäisten osien monitorointiin.

4.5.6 Kapacitor

Kapacitorin päätehtävä InfluxDatan TICK pinossa on prosessoida pinosta löytyvää dataa ja löytää esimerkiksi merkittäviä poikkeavuuksia tästä datasta (InfluxData 2019). Kapacitoria usein hyödynnetään metriikoiden keruussa varsinkin siinä kohtaan, kun halutaan tehdä hälytyksia tai ilmoituksia poikkeavuuksista.

4.5.7 Kapacitorin asennus

Kapacitor voidaan asentaa seuraavanlaisesti:

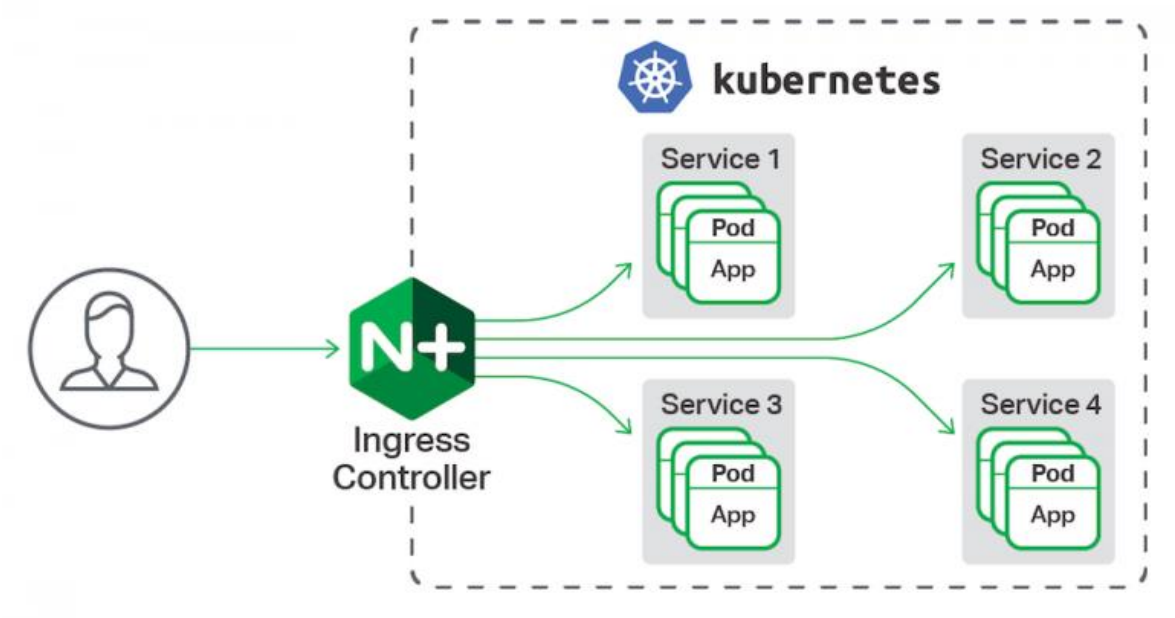
```
$ helm install --name alerts --namespace tick influx/kapacitor
```

4.5.8 Chronograf

Lopulta Chronograf on TICK pinon visualisointiin keskittyvä työkalu, millä voidaan luoda näyttäviä ja hyödyllisiä graafeja erilaisista metriikoista (InfluxData 2019). Chronograf voitaisiin ymmärtää loppujen lopuksi pinon tärkeimpänä työkaluna, sillä visualisoinnin ansioista voimme tehdä datan perusteella nopeita johtopäätöksiä, esim. resurssienkäytöstä, klusterin toiminnasta ynnä muuta.

4.5.9 Chronografin asennus

Koska käytän Chronografia datan visualisointiin, niin minun täytyy saada tämä palvelu näkyviin klusterini sisältä toisin kuin esimerkiksi minun InfluxDB asennus. Tässä me voimme hyödyntää Kubernetesin Ingressiä, minkä avulla voidaan jakaa liikennettä suoraan haluttuihin palveluihin klusterini sisällä. Ingressin ansioista voin myös jakaa kuormaa tarvittaessa, mikäli esim. liikenteessä palveluun tulisi merkittävä piikki.



Kuva 6. Kubernetes Ingress (DevOpsCube 2019).

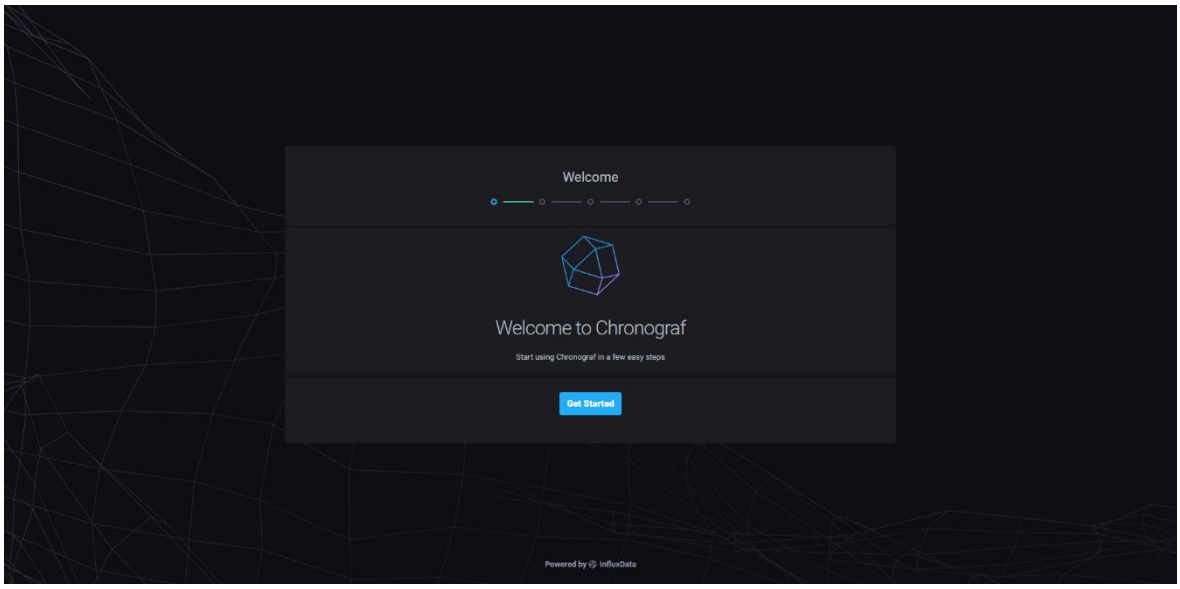
Ingressin saamme käyttöön Chronografille seuraavalla konfiguraatioilla:

```
# chronograf-values.yaml
ingress:
  enabled: true
  hostname: xip.io
  annotations:
    kubernetes.io/ingress.class: "nginx"
```

Tässä määritän, että otan käyttöön Kubernetesin Nginx ingressin ja julkaisen palveluni xip.io DNS-palvelun avulla, minkä avulla saan palvelulle "oikean" osoitteen. Chronograf voidaan asentaa Helmillä seuraavasti:

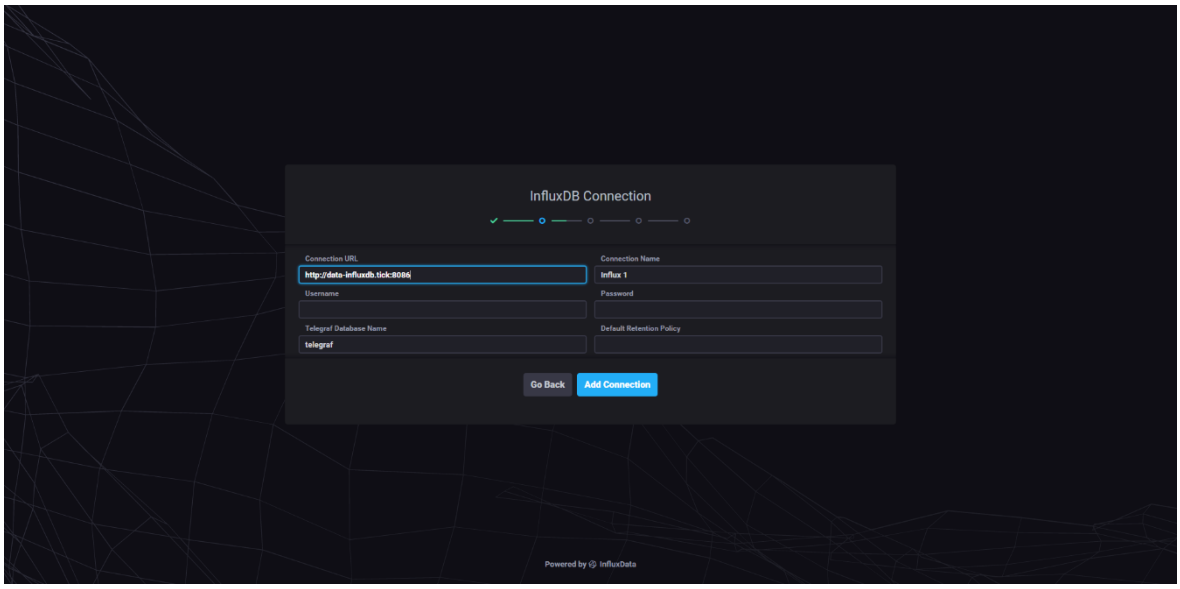
```
$ helm install --name dash --namespace tick influx/chronograf -f val-
ues/chronograf-values.yaml
```

Asennuksen jälkeen Chronografin pitäisi tulla ulospäin näkyviin osoitteeseen: http://dash-chronograf.tick.192.168.50.6.xip.io. Huomioi taas, että IP on todennäköisesti eri sinun kohdallasi.

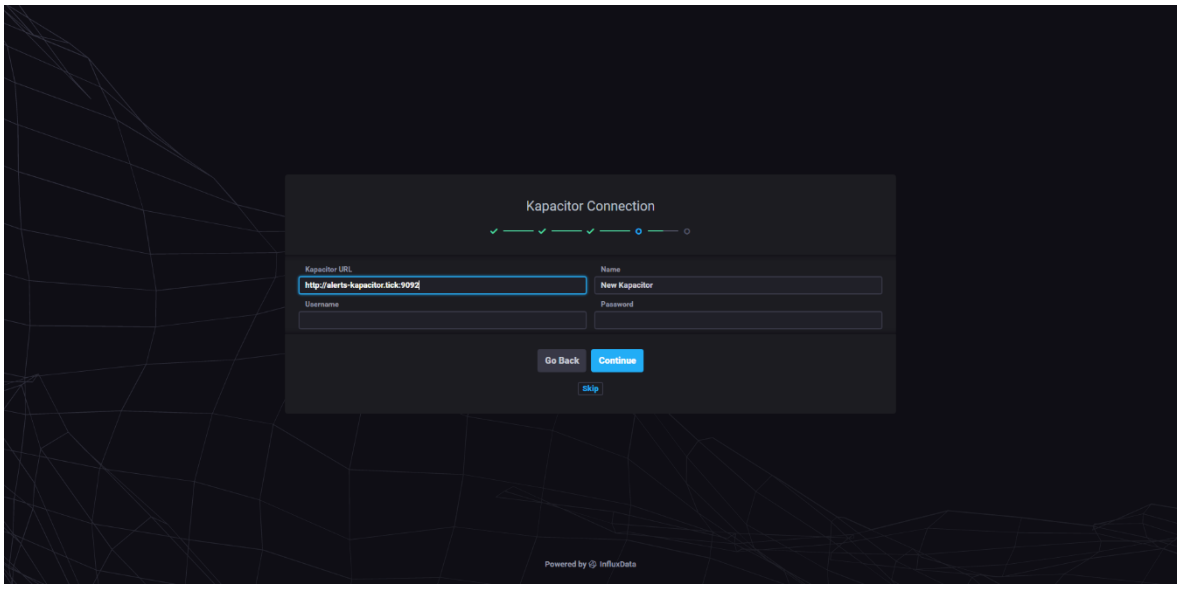


Kuva 7: Kuva Chronografin etusivusta.

Tämän jälkeen Chronograf kysyy, että mistä palvelu voi löytää InfluxDB:n sekä Kapacitorin. Näissä kohdissa voimme hyödyntää klusterin sisällä olevia palveluiden osoitteita eli: http://data-influxdb.tick:8086 ja http://alerts-kapacitor.tick:9092.



Kuva 8: Kuva Chronografin InfluxDB konfiguraatioista.



Kuva 9: Kuva Chronografin InfluxDB konfiguraatioista.

Mikäli konfiguraatiot on tehty oikein, pitäisi vakioarvoilla nähdä seuraavat metriikat Dockerin resurssienkäytöstä. Jo näiden vakiograafien pohjalta näen heti, minkälainen resurssien käyttö klusterissani on tällä hetkellä.

Kuva 10: Kuva Chronografin Docker metriikoista.

4.6 TensorFlow

TensorFlow on rajapinta erilaisten koneoppimisalgoritmien kuvaamiseen sekä samaan aikaan työkalu näiden ajamiseen. TensorFlowin kehitys juontaa juurensa Googlen yksityiseen skaalautuvaan hajautettuun koneoppimisjärjestelmään nimeltä DistBelief, jota käytettiin laajasti monessa eri Googlen tutkintahankkeessa. (Abadi ym. 2016, 1.) TensorFlow kehitettiin DistBeliefin käytön myötä karttuneen kokemuksen ja ymmärryksen järjestelmän vaatimuksista koneoppimiseen ja neuroverkkoihin (Abadi ym. 2016, 1).
TensorFlowin toiminta perustuu datavirta-tyyppisiin malleihin, mitkä voidaan tallentaa laajaan kirjoon erilaisia järjestelmiä aina mobiililaitteista tuhansian näytönohjaimia omistavaan järjestelmään. (Abadi ym. 2016, 1.) Skaalautuvuus on myös oleellisessa osassa TensorFlowin arkkitehtuuria, minkä takia siinä on mahdollistettu laaja kirjo erilaisia lähestymisiä muun muassa rinnakkaistukseen (Abadi ym. 2016, 2).

4.6.1 TensorFlow ja Kubernetes

TensorFlow mallien ajo oli pitkään Kubernetesissa melko monimutkaista. Google onneksi toi apua tähän tekemällä heidän koneoppimis-Kubernetes linjaston avoimeksi Kubeflown muodossa (Kubeflow 2019). Kubeflown tarkoitus ei ollut tehdä uutta palvelua tai luoda uudelleen jo olemassa oleva. Kubeflown päätarkoitus on tarjota mahdollisimman yksinkertainen tapa ottaa tehokkaaksi todettu tapa suorittaa koneoppimislaskentoja monipuolisissa ympäristöissä. (Kubeflow 2019.) Kubeflow hyödyntää toiminnassaa mahdollisimman paljon ominaisuuksia, mitkä Kubernetes hoitaa jo hyvin. Tällaisia ominaisuuksia ovat muun muassa helpot ja usein toistuvat ajot erilaisissa infrastruktuureissa, löyhästi linkitettyjen mikropalveluiden käyttöönotto ja hallinta sekä tarpeen vaatiessa skaalautuminen. (Kubeflow 2019.)

Kubeflown historia juontaa juurensa Googlen sisäisiin TensorFlow linjastoihin ja projekti alkoi vain yksinkertaisemmalla tavalla ajaa tehtäviä Kubernetesissä. Sittemin projekti on kuitenkin kasvanut työkaluksi ajaa linjastoja monessa eri arkkitehtuurissa ja monella eri pilvitarjoajalla. (Kubeflow 2019.)

4.6.2 Kubeflown käyttöönotto

Kubeflow on vielä suhteellisen nuori projekti, minkä vuoksi sen käyttöönotto ei ole vielä tehty virtaviivaiseksi. Pää syy tähän lienee, se että Kubeflow voi muuttua merkittävästi ennen versiota 1.0, kun tämän hetkinen version on 0.6.2. Ensimmäiseksi meidän täytyy ladata Kubeflown oma komentorivityökalu nimeltä kfctl. Tämä voidaan ladata helposti projektin

GitHub sivujen julkaisuista (https://github.com/kubeflow/kubeflow/releases). Tai myös seuraavalla komennolla:

```
$ wget https://github.com/kubeflow/kubeflow/releases/download/v0.6.2/kfctl_v0.6.2_linux.tar.gz
```

Kubeflowsta on huomioitava se, että komentorivityökalu ei ole vielä yhteen sopiva Windowsin kanssa, minkä vuoksi sen ajamiseen vaaditaan joko Linux- tai Darwin-pohjainen (macOS) tietokone. Lataamisen lisäksi meidän luonnollisesti täytyy purkaa ohjelma ja siirtää PATH-polkuni alle, esimerkiksi /usr/local/bin alle:

```
$ tar xvzf kfctl_v0.6.2_linux.tar.gz
$ sudo mv kfctl /usr/local/bin/
```

Tämän jälkeen voimme testata asennustamme esimerkiksi seuraavasti:

```
$ kfctl version
kfctl v0.6.2-0-g47a0e4c7
```

Ja voin todeta asennukseni toimivan. Tämän jälkeen täytyy myös asentaa Kubernetesin oma komentorivityökalu kubectl, minkä asensimme työn Kubernetes-osiossa. Viimeiseksi kfctl tulee konfiguroida myös toimimaan kustomize työkalun kanssa, mikä myös täytyy erikseen ladata.
Kustomize on Kubernetesin konfigurointiin tarkoitettu työkalu, jonka avulla voimme generoida, hallinoida ja muokata eri asennuksien konfiguraatioita ilman, että niitä muokattaisiin suoraan käsin. Ennen Kustomizen asennusta täytyy asettaa yksi ympäristömuuttuja:

```
$ export opsys=linux
```

Tämä muuttuja luonnollisesti riippunee käytössä olevasta käyttöjärjestelmästä, ja kun omassa tapauksessani käytössä on Linux, asettan sen luonnolisesti edellä mainitulla tavalla. Tämän jälkeen voin ladata binäärin seuraavasti:

```
$ curl -s https://api.github.com/repos/kubernetes-sigs/kustomize/releases/latest | \
        grep browser_download | \
        grep $opsys | \
        cut -d '"' -f 4 | \
        xargs curl -O -L
```

Lopuksi siirrän binäärin kfctl tavoin PATH polun alle sekä annan sillä suoritusoikeudet:

```
$ chmod +x kustomize_kustomize.v3.2.3_linux_amd64
$ sudo mv kustomize_kustomize.v3.2.3_linux_amd64 /usr/local/bin/kustomize
```

Lopuksi testaan asennustani seuraavasti:

```
$ kustomize version
Version: {Version:kustomize/v3.2.3 GitCom-
mit:f8412aa3d39f32151525aff97a351288f5a7470b BuildDate:2019-10-
08T23:30:25Z GoOs:linux GoArch:amd64}
```

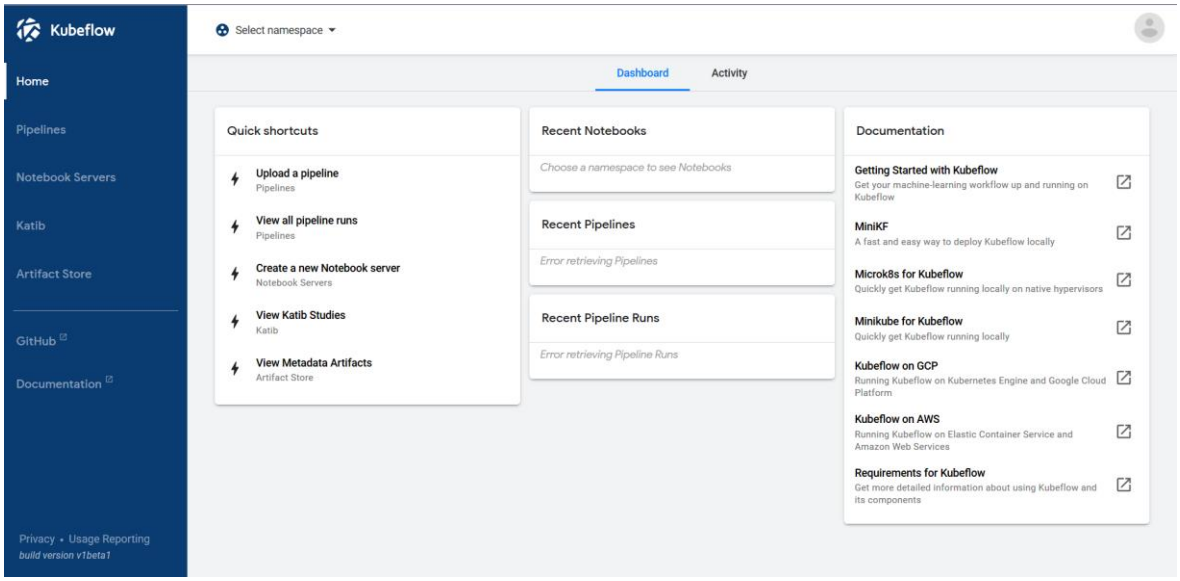
Tämän jälkeen voin alustaa Kubeflow projektini seuraavanlaisesti:

```
$ export KFAPP="kf-haaga-helia"
$ export VERSION=`curl -s https://api.github.com/repos/kubeflow/kube-
flow/releases/latest | \
       grep tag_name | head -1 |  cut -d '"' -f 4`
$ export CONFIG="https://raw.githubusercontent.com/kubeflow/kube-
flow/${VERSION}/bootstrap/config/kfctl_k8s_istio.yaml"
$ kfctl init ${KFAPP} --config=${CONFIG} -V
$ cd ${KFAPP}
$ kfctl generate all -V
$ kfctl apply all -V
```

KFAPP on vain nimi Kubeflow applikaatiolle. VERSION etsii verkosta viimeisimmän Kubeflow version ja asettaa sen tälle muuttujalle, tässä tapauksessa se on 3.2.3. Tämän jälkeen luon Kubernetes Deploymentin Kubeflowlle, alustan Kubeflow projektin sekä asennan sen samalla klusteriini.
Asennuksen jälkeen pääsen käsiksi Kubeflown dashboardille Kubernetesin NodePortin avulla. Tuotannossa konfiguroisin Ingressin tämän palvelun eteen, mikä osaisi jakaa kuormaa oikein. Tässä tapauksessa kuitenkin yksittäisen portin avaaminen suoraan palveluun riittää hyvin. Portti Kubeflown dashboardille voidaan avata seuraavanlaisesti:

```
$ export INGRESS_PORT=$(kubectl -n istio-system get service istio-in-
gressgateway -o jsonpath='{.spec.ports[?(@.name=="http2")].nodePort}')
$ export SECURE_INGRESS_PORT=$(kubectl -n istio-system get serviceistio-
ingressgateway -o jsonpath='{.spec.ports[?(@.name=="https")].nodePort}')
$ echo $INGRESS_PORT
31380
```

Komentojen viimeinen echo tulostaa vasta avaamani portin, jos tämä muuttuja on tyhjä todennäköisesti asennus ei ole täysin onnistunut. Huomaa myös, että tämä voi muuttua konekohtaisesti. Tämän jälkeen voin selaimellani mennä osoitteeseen, esim. http://192.168.50.6:31380 (korjaa IP omaksesi) ja minun tulisi nähdä seuraavanlainen näkymä:



Kuva 11: Kubeflown dashboard

Täällä pystyn hallinnoimaan kaikkea Kubeflown asioita, muun muassa infraan ja dataan liittyvien Jupyter Notebookien hallintaa sekä koneoppimisputkistojen (pipeline) hallintaa.

5 Tulosten analysointi

DevOpsissa ketteryys sekä sujuvuus ovat merkittävässä roolissa. Tämän vuoksi monitorointi nousee hyvin oleelliseksi osaksi päätöksentekoa yrityksissä. Ketterässä ympäristössä nopeat muutokset sekä muuttuvat päätökset ovat hyvinkin arkipäivää. Näihin vastaaminen kehittäjien tai ylläpitäjien näkökulmasta usein ei ole niin helppoa kuin projektin managerit haluisivat sen olevan. Näiden tahojen välisen kommunikoinnin helpottamiseksi on alettu hyödyntää DevOps-kulttuurin piirteitä.

Oman kokemuksen mukaan suurilla firmoilla palvelut voivat olla alhaalla viikossa useita tunteja. Tämä voi viikkoskaalalla tuntua suhteellisen lyhyeltä, mutta kun lähestyy tätä koko vuoden aikana, nämä pienet luvut kasvavat suhteellisen merkittäviksi. Erityisesti tilanteessa, missä palvelu on bisneskriittinen, joka tuo esimerkiksi yritykselle merkittävästi myyntiä. Tämän kaltaisessa tilanteessa tällaisten riskien minimointi alkaa nousta suhteellisen merkittäväksi tehtäksi.

Hyvä monitorointi itsessään voi olla jo hyvä vastaus edellä mainittujen riskien minimoimiseen. Hyvällä monitorointijärjestelmällä esimerkiksi voidaan heti paikallistaa mahdolliset ongelmat, millä serverillä nämä ongelmat ilmenivät ja ehkä jopa suoraan ongelmien syy. Tällaisten nopeiden ja aikaisten havaintojen ansiosta ongelmanratkaisuun käytetty aika usein vähenee sekä samaan aikaan projektitiimin jäsenet voivat keskittyä olennaiseen.

Suuri hyöty myös tämänkaltaisissa monitorointijärjestelmissä on se, että nämä voidaan räätälöidä kullekin asiakkaalle juuri sellaiseksi mitä he siltä kaipaavat. Oman kokemuksen mukaan suurin osa olemassa olevista "suoraan-hyllystä"-monitorointiratkaisut toimivat usein tiettyyn pisteeseen asti hyvin, minkä jälkeen usein toivotaan, että niillä voisi tehdä jotain hieman erilaista. Tähän on taas vastauksena juuri tällaisten TICK pinojen tyyppisten ratkaisujen hyödyntäminen, sillä nämä usein tarjoavat vain työkalut ja sitten näitä työkaluja voidaan hyödyntää aivan, miten itse parhaaksi katsoo.

Kubeflown asennuksessa voidaan huomata se, että kyseessä on vielä suhteellisen nuori projekti, minkä takia asennusprosessi on suhteellisen monimutkainen. Tämä voidaan myös todeta koko työn eri vaiheista. Pilvinatiivit ovat monessa tapaa suhteellisen monimutkaisia arkkitehtuureja, mutta pinnan alla piilee todella suuria hyötyjä. Näiden hyötyjen ansiosta hyvin merkittävä osa suurista firmoista ovat alkaneet suorittamaan migraatiota tämän kaltaisiin ympäristöihin esimerkiksi Kubernetesin avulla. Tämän vuoksi myös osaaminen DevOpsin saralla tulee olemaan merkittävä taito, koska se alkaa tulla oleelliseksi osaksi yritystoimintaa ketterien toimintatapojen tavoin.

Lähteet

Abadi. M, ..., (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. Osoitteesta https://arxiv.org/pdf/1603.04467.pdf. Luettu: 07.09.2019

Conway. N, ... (2012). Logic and Lattices for Distributed Programming. Osoitteesta http://db.cs.berkeley.edu/papers/UCB-lattice-tr.pdf. Luettu: 07.09.2019

Docker (2019). Docker overview. Osoitteesta: https://docs.docker.com/engine/docker-overview/. Luettu: 07.09.2019

InfluxData (2019). Kapacitor 1.5 documentation. Osoitteesta: https://docs.influxdata.com/kapacitor/v1.5/. Luettu: 18.09.2019.

InfluxData (2019). InfluxDB 1.7 documentation. Osoitteesta: https://docs.influxdata.com/influxdb/v1.7/. Luettu: 09.09.2019.

InfluxData (2019). InfluxDB compared to SQL databases. Osoitteesta: https://docs.influxdata.com/influxdb/v1.7/concepts/crosswalk/. Luettu: 09.09.2019

InfluxData (2019). Telegraf 1.12 Documentation. Osoitteesta: https://docs.influxdata.com/telegraf/v1.12/. Luettu: 09.09.2019.

Kubeflow (2019). Getting started with Kubeflow. Osoitteesta: https://www.kubeflow.org/docs/started/getting-started/. Luettu: 10.11.2019.

Kubeflow (2019). Configuring Kubeflow with kfctl and kustomize. Osoitteesta: https://www.kubeflow.org/docs/other-guides/kustomize/.

Kubernetes (2019). Ingress. Osoitteesta: https://kubernetes.io/docs/concepts/services-networking/ingress/. Luettu: 07.09.2019

Kubernetes (2019). Overview of kubectl. Osoitteesta: https://kubernetes.io/docs/reference/kubectl/overview/. Luettu: 07.09.2019

Kubernetes (2019). What is Kubernetes? Osoitteesta: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/. Luettu: 09.09.2019.

Kubernetes (2015). Borg: The Predecessor to Kubernetes. Osoitteesta: https://kubernetes.io/blog/2015/04/borg-predecessor-to-kubernetes/. Luettu: 09.09.2019.

Kubernetes (2019). DaemonSet. Osoitteesta: https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/. Luettu: 18.09.2019.

Lamport. L (1978). Time, Clocks and the Ordering of Events in a Distributed System. Osoitteesta https://www.microsoft.com/en-us/research/publication/time-clocks-ordering-events-distributed-system/?from=http%3A%2F%2Fresearch.microsoft.com%2Fen-us%2Fum%2Fpeople%2Flamport%2Fpubs%2Ftime-clocks.pdf. Luettu: 07.09.2019

Rancher (2019). Overview of RKE. Osoitteesta: https://rancher.com/docs/rke/latest/en/. Luettu: 07.09.2019

Rancher (2019). High Availability (HA) Install. Osoitteesta: https://rancher.com/docs/rancher/v2.x/en/installation/ha/. Luettu: 07.09.2019

Takada. M (2013). Distributed systems for fun and profit. Osoitteesta http://book.mixu.net/distsys/ebook.html. Luettu: 07.09.2019

Vagrant (2019). Vagrant documentation. Osoitteesta: https://www.vagrantup.com/docs/index.html. Luettu: 07.09.2019