TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Jaanus Steinfeldt 185136IAAB

Rakenduse Kiirabi Mobiilne Töökoht migratsioon Kubernetese keskkonda

Bakalaureusetöö

Juhendaja: Edmund Laugasson
MSc

Tallinn 2022

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Jaanus Steinfeldt

16.05.2022

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on veebirakenduse üleviimine ühest keskkonnast teise. Rakendus on varasemalt käivitatud virtuaalmasinas kasutades monoliitarhitektuuri tavasid, üleviimise käigus lähtutakse uue keskkonna tavadest, ehk mikroteenuste arhitektuurist.

Üleviimise protsessi käigus analüüsitakse monoliitse ning mikroteenuste arhitektuuri positiivseid ja negatiivseid omadusi ning võrreldakse rakenduse ressursi kasutamist erinevates keskkondades ja käitumist uuendamisel.

Töö rõhk on rakenduse uuendamine kasutaja jaoks ilma katkestuseta. Muuhulgas katab lõputöö mikroteenuste arhitektuuril kasutatud objekte ja nende selgitusi. Üleviimise eesmärgiks on rakenduse toimimine uues keskkonnas.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 33 leheküljel, 5 peatükki, 12 joonist, 1 tabelit.

Abstract

Kiirabi Mobiilne Töökoht Application Migration into Kubernetes

The aim of current thesis is to transfer a web application from one environment to another. The application has previously been launched in a virtual machine using the practices of a monolithic architecture and later using the practices of microservices architecture.

During the migration process the assumptions and shortcomings of the monolithic and microservice architecture are analysed. The use of the application resource is compared in different environments and behaviour when updating.

The emphasis of the job is to update the application without interruption for the user. Among other things, the dissertation covers the objects used in architecture of microservices and their explanations. The purpose of the migration is to make the application work in the new environment.

The thesis is in Estonian and contains 33 pages of text, 5 chapters, 12 figures, 1 table.

Lühendite ja mõistete sõnastik

ConfigMap	Kuberneteses võti-väärtus teabe edastamiseks mõeldud objekt
CPU	Kesktöötlusseade (CPU)
Deployment	Kuberneteses rakenduse käivitamiseks mõeldud osa, kus kirjeldatakse, mida on vaja kasutada, et rakendus käivituks
Docker	Docker on platvorm kui teenus (PaaS), mis kasutab OS-taseme virtualiseerimist, et pakkuda tarkvara pakettides, mida nimetatakse konteineriteks
Dockerfile	Tekstidokument, mis sisaldab endas käske tõmmise loomiseks
Gitlab Container Registry	Turvaline ja privaatne tõmmiste hoiustamise koht
Helm chart	Kubernetese failide kogum
HTTP	Serverite ja klientide vahel hüperdokumente edastada võimaldava rakenduse protokolli nimi
HTTPS	Turvalisem versioon HTTP-st
IP	Seadmele määratud alaline või ajutine tunnusnumber
Kaun	Kaun (*pod*) on kõige väiksem Kuberneteses käivitatav osa
Keycloak	Avatud lähtekoodiga tuvastamiseks mõeldud teenus
KMT	Kiirabi Mobiilne Töökoht
Kobar	Kobar (*cluster*) on ühise juhtimise all olevate funktsionaalüksuste kogum
Kubernetes	Avatud lähtekoodi konteinerite orkestreerimiseks mõeldud platvorm
Nimeruum	Nimeruum (*namespace)* võimaldab eraldada erinevad ressursid ühe kobara sees
Peremeesmasin	Peremeesmasin (host)
Port	Virtuaalne punkt kus interneti ühendused algavad ja lõppevad
Rancher	Avatud lähtekoodiga tarkvara, mis aitab kergemini hallata Kubernetese keskkonda
Redis	Mälusisene andmestruktuuride hoidla
Saladus	Saladus (*secret)* on Kuberneteses võti-väärtus teabe edastamine krüpteeritud kujul
Silt	Silt *(label)* võti-väärtus paar erinevate Kubernetese objektide ühendamiseks

Sissepääs	Sissepääs (*ingress*) on Kubernetese osa, mis juhib võrguliiklus rakenduste vahel
SpringBoot	Avatud lähekoodiga Java põhjal raamistik, millega luuakse mikroteenuseid
TARA	Riigi autentimisteenus
Teenus	Teenus (*service*) on kaunade võrguliiklust ühendav süsteem
TEHIK	Tervise ja Heaolu Infosüsteemide Keskus
Tomcat	Avatud lähtekoodiga tarkvara, kus käivitatakse Java rakendusi
URL	Universaalne asukohamääraja
War-fail	Java failide kogum
X-tee	Andmevahetuse platvorm, mis võimaldab turvaliselt asutuste vahel teavet pärida ja vahetada

Sisukord

1 Sissejuhatus 10
1.1 Probleemi kirjeldus ja eesmärk 10
1.2 Hetkeolukord 11
1.3 Üleviimise eeldused 12
2 Metoodika 13
3 Monoliitse ja mikroteenuste arhitektuuri erinevused 14
3.1 Monoliitarhitektuur 14
3.2 Mikroteenuste arhitektuur 16
4 Rakenduse üleviimise etapid 18
4.1 KMT rakendus 18
4.2 Rakenduse tööks vajalike võrguühenduste kaardistamine 19
4.2.1 Kubernetese teenus ja sissepääs KMT rakendusse 21
4.3 KMT konteineriseerimine 22
4.3.1 Rakenduse sätted erinevas keskkonnas 23
4.3.2 Kubernetes ConfigMaps 23
4.3.3 Kubernetes Secrets 24
4.3.4 Kuberneteses juurutamine 25
4.3.5 Horisontaalskaleerimine 25
4.4 Käivitamine uues keskkonnas 26
5 Tehtud töö analüüs 29
5.1 Tulevikuarendused 31
6 Kokkuvõte 32
Kasutatud kirjandus 33
Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks 35

Jooniste loetelu

Joonis 1. Monoliitsüsteemi näitlik ülesehitus [4] 14
Joonis 2. Mikroteenuste näitlik ülesehitus [4] 16
Joonis 3. KMT rakenduse töö käik 19
Joonis 4. Rakenduse tööks vajalikud välised teenused 19
Joonis 5. Teenuse valmistamise näide 20
Joonis 6. Sissepääsu valmistamise näide 21
Joonis 7. Sissepääsu visuaalne näide [8] 22
Joonis 8. Füüsilise serveri, virtuaalmasina ja Docker konteineri näiline struktuur [10] 22
Joonis 9. *Dockerfile* näidis 23
Joonis 10. Kubernetes ConfigMap'i näide 24
Joonis 11. Kubernetese saladuse näide 25
Joonis 12.Horisontaalne skaleerimise näidis 26

Tabelite loetelu

Tabel 1. KMT rakenduse võrdlus erinevates keskkondades 30

1 Sissejuhatus

Autor kirjutab organisatsiooni TEHIK poolt hallatava Kiirabi Mobiilne Töökoht (KMT) rakenduse ülemineku protsessist monoliitse arhitektuuri pealt mikroteenuste arhitektuuri peale. Aina enam võetakse maailmas kasutusele mikroteenuste arhitektuuri, kuna sellega lubatakse paremat vastupidavust, mastaapsust ja kergemini hooldatavat süsteemi. Praegusel hetkel on süsteem üks suur monoliit, mille uuendamine erinevate masinate vahel võib tekitada probleeme. Sellest tulenevalt on võetud plaani kasutada mikroteenuste arhitektuuri, et uuendamine oleks katkestuseta ja rakendus toimiks igas keskkonnas samamoodi.

Käesolevas töös kirjeldatakse mikroteenuste ning monoliitse arhitektuuri positiivsed ja negatiivsed omadused, üleminekuks vajalikud etapid, kuidas vana süsteem välja näeb ja kuidas uus võiks olla. Autor plaanib paigaldada näidissüsteemi, mille käigus selgitatakse, milliseid arendusi on olemasolevale Java-rakendusele vaja teha.

1.1 Probleemi kirjeldus ja eesmärk

Kiirabi Mobiilne Töökoht on hädavajalik ja kriitiline osa kiirabide igapäevatöös. Esineb mitmeid olukordi, kus hoolduse ajaks peab kuni kolmeks tunniks kogu süsteemi peatama. Peale hooldust saavad brigaadid paberile kirjutatud vajalikud andmed sisestada käsitsi KMT-sse. Hetkel kasutatakse KMT süsteemil monoliitset arhitektuuri. See koosneb kahest virtuaalmasinast: ühe peal asub arvuti versioon ja teise peal mobiilne versioon, kahte korraga kasutada ei saa. Virtuaalmasinas on paigaldatud Tomcat’i rakendus, mille kaudu käivitatakse SpringBoot rakendus. Tulevikus võiks saada rakendust skaleerida, kasutades Kubernetest.

Eesmärk on analüüsida positiivseid ja negatiivseid omadusi mikroteenuste ja monoliitsel arhitektuuril. Proovida kasutada olemasolevat SpringBoot rakendust ja käivitada see Kubernetese keskkonnas, analüüsida tekkinud probleeme ja arendajale edastada puuduvad arendused rakenduse käivitamiseks, jälgides mikroteenuste arhitektuurile

tavasid. Rakenduse uuendamiseks ei oleks vajalik rakenduse peatamine, vaid uuendust oleks võimalik jooksvalt teha. Uues keskkonnas paigaldatud rakendus peaks vastama järgmistele lähtetingimustele:

- Ligipääs ainult kindlatelt IP-delt.
- Riigi autentimisteenuse (TARA) toimimine.
- Välise andmebaasi kasutamine.
- Sessiooni säilitamine rakenduse uuendamisel.
- Ressursi kasutamine dünaamiliselt.

1.2 Hetkeolukord

Hetkel töötab arendusversioon KMT rakendusest virtuaalmasina peal, millele on paigaldatud CentOS operatsioonisüsteem. Selle operatsioonisüsteemi miinimumnõuded on 1 GB mälu aga soovitatult vähemalt 2 GB. Lisaks võiks olla minimaalselt 20 GB kettapinda aga soovitatult vähemalt 40 GB. Viimaseks nõudeks on vajalik vähemalt 1 protsessori tuum, et saaks operatsioonisüsteemi paigaldada ja käivitada. [1]

Arenduses olevale virtuaalmasinale on antud 4 GB mälu, millest 1,6 GB on kasutuses tavaolekus, ehk ilma klientideta. Toodangus on mälu kasutus kordades suurem, mille tõttu tuleb panna masinale kõiki ressursse varuga, mis suures pildis võrdub ressursi raiskamisega, sest kogu soovitud mälu on kinni ühe virtuaalmasina peal, mida teised virtuaalmasinad kasutada ei saa. Kui sooviks praeguse lahenduse juures käivitada rakendust mitme õla peal, et käideldavust tõsta, siis tähendaks see kahekordse ressursi kasutamist ühe rakenduse poolt.

Arvestades, et operatsioonisüsteemi miinimumnõue on 1 protsessori tuum, siis kiirema ja tõrgeteta töö jaoks oleks vaja ka seda natukene varuga lisada, hetkel on arenduses kasutuses 4 protsessori tuuma, mis tähendaks rakenduse mitme õla peal tööle panemiseks 8 protsessori tuuma kasutamist. Kui veel juurde arvestada minimaalne kettapinna soovitus, siis ühe rakenduse käideldavuse tõstmiseks kulub topelt kogus ressurssi, arvestades, et testis ja toodangus on numbrid kaks kuni kolm korda suuremad. Seepärast on plaan liikuda mikroteenuste arhitektuuri poole, et väiksema ressursi kasutamisega suudaks tagada rakenduse katkematu töö.

1.3 Üleviimise eeldused

Rakenduse üleviimine ühest keskkonnast teise on ajakulukas ja tuleb teada, mis elemendid on rakenduse tööks hädavajalikud. KMT rakendus sõltub väga palju välistest teenustest. Selle jaoks tuli kaardistada, millised võrgureeglid on vaja luua uue keskkonna jaoks ning millised teenused on vaja kätte saada ja kas kasutajad jäävad samaks. Samuti rakenduse seadistamine tuleks korraldada, järgides uue keskkonna tavasid ja reegleid.

Üleviimise eelduseks on vaja uues keskkonnas teha tulemüürireeglid, et rakendus saaks kindlalt ühendust sõltuvate rakendustega. Selle jaoks on vaja teha TEHIKu süsteemis vastav pilet, mis registreeritakse ja suunatakse võrguhaldusosakonnale. Piletis peab olema teave, kus kohast liiklus tuleb ehk Kubernetese keskkonna IP ja sõltuva teenuse IP koos pordiga.

2 Metoodika

Antud lõputöö on empiiriline uurimustöö, mille metoodika on eksperimentaalne. Töö viiakse läbi ühe rakenduse lõikes ning uurimustöö käigus selgitatakse, miks valiti just selline lähenemine. Uurimise käigus võrreldakse KMT rakendust erinevas töökeskkonnas. Üks keskkond on monoliitse arhitektuuri põhimõtetel ja teine on mikroteenuste arhitektuuri põhimõtetel.

Katsetamise jaoks on antud ettevõttes juba eelnevalt valitud mikroteenuste haldamiseks ja orkestreerimiseks Kubernetes. Kubernetese keskkonna graafilise halduse jaoks on võetud kasutusele Rancher, et mitme kobara haldus oleks lihtsam ja inimviga oleks vähem. Rancher pakub graafilist Kubernetese haldamise võimalust, millega on kordades lihtsam teha keerulisi seadistusi. Näiteks koormusjaoturi loomine, luues erinevaid sätetefaile ja need ühe vajutusega siduda rakendusega. [2] Rancheriga on võimalik luua ühe kobara sisse erinevad projektid, mille sees on omakorda nimeruumid. Projektidega on võimalik piirata ligipääsu teistele projektidele sama kobara sees. Lõputöös sooritatud lahendus on püstitatud ja paigaldatud kasutades Rancherit ja sellele keskkonnale sobivaid nõudeid.

Selleks, et saaks uue rakenduse Rancheris paigalda ja käivitada, on esmalt vaja luua sobiva kobara sisse projekt, näiteks „KMT arendus“. Teiseks, on vaja välja mõelda loogilised nimeruumid. Nimeruume kasutatakse selleks, et saaks ühe projekti raames eralda erinevad seadistused ja muutujad.

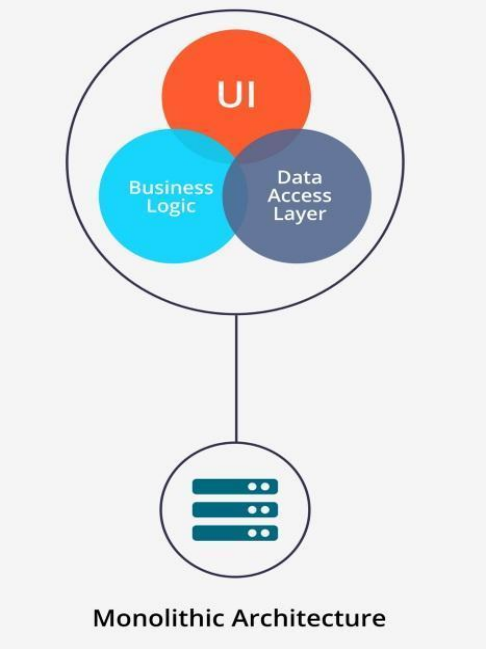
Viimases etapis uuritakse rakenduse tööd kahes erinevas keskkonnas. Esimene oleks hetkeolukord ehk virtuaalmasinas paigaldatud rakendus ja teine oleks sama rakendus paigaldatud Kubernetese keskkonda. Katsetamisel võrreldakse rakenduse vastupidavust, käideldavust, ressursi kasutamist ja kuidas reageerib erinevatele probleemidele.

3 Monoliitse ja mikroteenuste arhitektuuri erinevused

KMT rakenduse kasutamine suureneb aastatega ja see rakendus muutub suuremaks ning nõudlikumaks, selle tagajärjel on vaja rakendust vertikaalselt suuremaks skaleerida. Monoliitse süsteemi puhul üldjuhul tähendab see vanade arvutite välja vahetamist või uute ja võimsamate virtuaalmasinate tekitamist. Üldjuhul see variant on kulukas ja ressursi raiskamine. Selle puuduste kõrvaldamiseks mõeldi välja mikroteenuste arhitektuur. Seda kasutades on võimalik rakendus jaotada väiksemateks osadeks ja on võimalik hoida erinevate masinate peal, sellega tekib võimalus horisontaalselt suuremaks skaleerida.

3.1 Monoliitarhitektuur

Monoliitne arhitektuur on traditsiooniline viis rakenduste arendamisel. Monoliitne tähendab seda, et kogu rakenduse funktsionaalsused on ühes koodibaasis ja vaadatakse kui ühtset tervikut, vt Joonis 1. [3] Monoliitne rakendus on üldjuhul arendatud kui iseseisev üksus, kõik komponendid on omavahel seotud, erinevalt mikroteenuste arhitektuurist, kus kõik tähtsad komponendid on eraldiseisvad. Iga osa koodist on tähtis ja peab olema töökorras, et kogu rakendus suudaks käivituda. Kui mõni rakenduse osa on vaja uuendada või täiustada, siis peab kogu rakenduse üle kontrollima ja vajadusel üle kirjutama, et uuendus rakenduks õigesti.



Joonis 1. Monoliitsüsteemi näitlik ülesehitus [4]

Monoliitarhitektuuri eelised on:

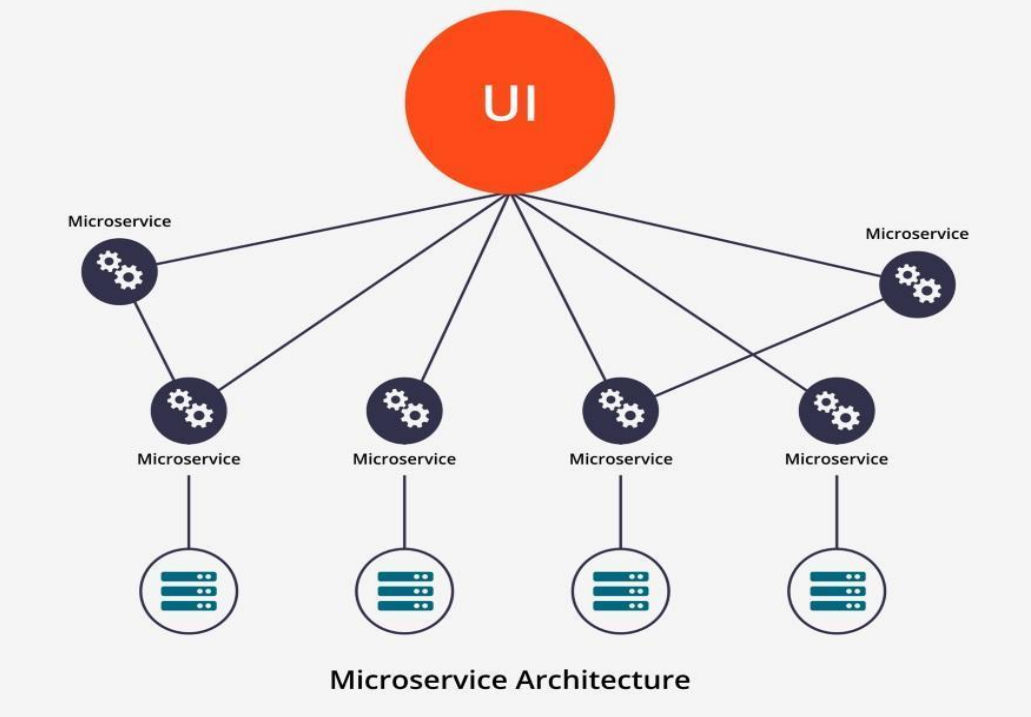
- Lihtsam arendada.
- Lihtsam paigaldada, kuna enamasti peab käivitama ainult ühe faili.
- Testimine on kergem.
- Kõik logid on ühes kohas, mis teeb vigade otsimise lihtsamaks.
- Odavam arendada, sest kulud komponentide vahel on nullilähedased, kuna kood asub kõik ühes kohas. [3]

Monoliitarhitektuuri puudused on:

- Pika aja jooksul muutub rakendus mahukaks, raskem hooldada ja toimetada.
- Iga muudatuse korral tuleb rakendus uuesti käivitada.
- Käivitamisaeg suureneb rakenduse kasvuga käsikäes.
- Uutel arendajatel on keeruline hiljem aru saada rakenduse ülesehituse loogikast.
- Jõudluse lisamine on kulukas.
- Raske uuendada üksikuid komponente rakenduses, kuna mõjutab kogu rakendust.
- Üks viga suvalises rakenduse moodulis mõjutab kogu rakendust ja võib põhjustada katkestuse. [3]

3.2 Mikroteenuste arhitektuur

Mikroteenuste arhitektuur on üks arenduse mudel, kus rakendus koosneb mitmest väikesest osast, mis suhtlevad omavahel, et moodustada üks tervik, vt Joonis 2. Iga rakenduse suurem teenus on eraldiseisev ja peaks tegema ainult teenusele ettenähtud tööd. Mikroteenused on väikesed ja eraldiseisvad, iga rakenduse osale võib olla eraldi arendaja, kuna rakenduse kood on teistest eraldatud. Uuendamine on kordades lihtsam, kuna ühe teenuse muutmisel ei pea taaskäivitama kogu rakendust. [5]



Joonis 2. Mikroteenuste näitlik ülesehitus [4]

Mikroteenuste eelised on:

- Lihtne hooldada, kuna on väiksem.
- Kui uuendatakse ühte osa rakendusest, siis peab ainult selle uuesti käivitama.
- Kõik komponendid on teistest eraldi ja saab taaskäivitada poole kiiremini.
- Uutel arendajatel on lihtsam liituda, kuna peab teadma ainult temale etteantud rakenduse osa.

- Toetab horisontaalset skaleerimist, ehk kui ühel komponendil on suur koormus, siis saab ainult sellele ressurssi juurde määrata.
- Iga komponent võib kasutada erinevat tehnoloogiat, et saavutada soovitud eesmärk.
- Kui ühel komponendil esineb viga, siis ei pea kogu rakendust sulgema, vaid ainult üks teenus ei tööta.
- Võimalik uuendada kogu teenust tööd katkestamata. [3]

Mikroteenuste puudused on:

- Keerulisem kui monoliitsüsteem, kuna rakenduse kasvamisel tuleb erinevaid lisakomponente juurde.
- Mikroteenuseid on raskem arendada ja vajalike oskustega arendajaid on vähe.
- Kulukam üleval pidada, kuna erinevad komponendid suhtlevad interneti kaudu.
- Vähem turvalisem, kuna komponendid suhtlevad interneti kaudu.
- Vigade otsimine on keerulisem, kuna komponente on palju ja kõigil on oma logi. [3]

4 Rakenduse üleviimise etapid

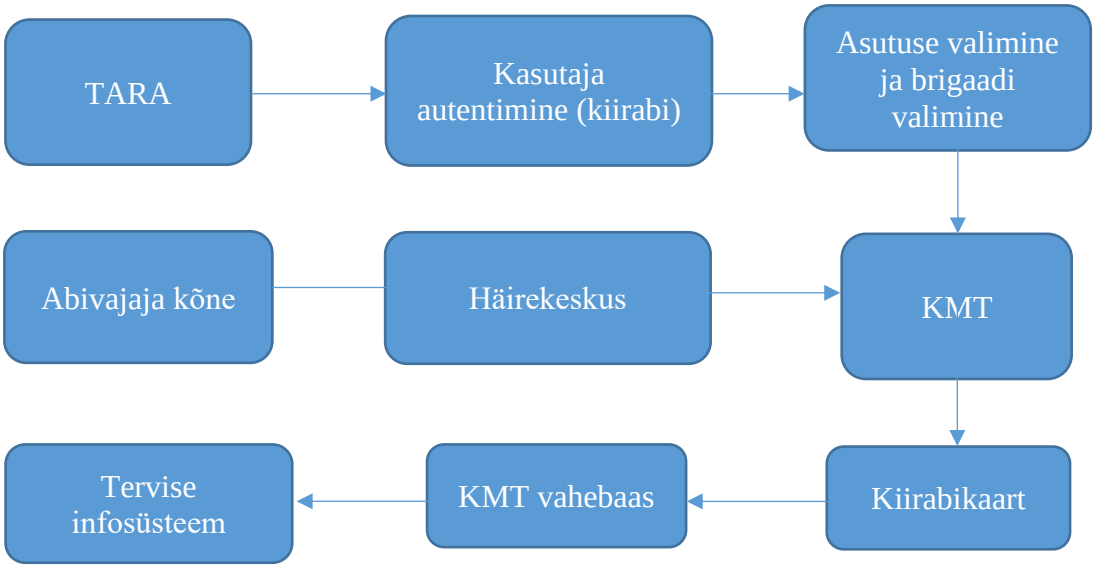
4.1 KMT rakendus

KMT on üks osa e-kiirabi projektist. „E-kiirabi puhul on tegemist kiirabi digitaalse abivahendiga, mida kasutatakse väljakutsetel oluliste terviseandmete saamiseks ning abivajajaga tehtud tegevuste ja terviseinfo saatmiseks teistele osapooltele.“ [6] Rakenduse kasutamine kiirendab patsiendi abistamise protsessi, kui kiirabikaardile on isikukood sisestatud, siis saab rakenduses pärida:

- Patsiendi meditsiinilise ajaloo.
- Varasemad õnnetused.
- Epikriisid.
- Allergiad.

Kui abivajaja kiirabikaart on loodud, siis see suunatakse peale allkirjastamist tervise infosüsteemi ja analüüsimoodulisse, kus lõpuks patsient saab enda digiloos näha sissekannet enda kiirabi väljakutse kohta.

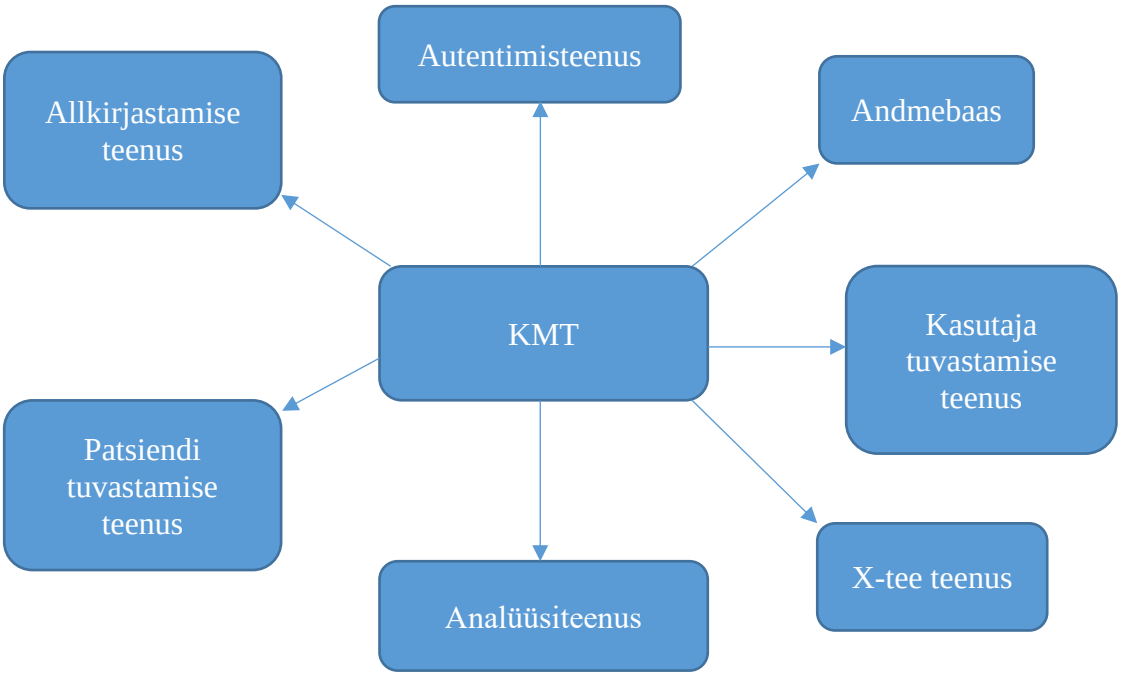
KMT rakendus on SpringBoot raamistikule ehitatud Java rakendus, mis on komplekteeritud arendaja poolt ühte war-faili. SpringBoot’i kasutatakse enamasti Java mikroteenuste arendamisel, mis tulevikus peaks ka KMT arenduse lihtsamaks tegema. Rakenduse konteineriseerimisega kaotame operatsioonisüsteemi sõltuvusega ja sellega kaasnevad nõrkused. KMT rakendusel on turvalisus väga kõrgel kohal, seepärast on rakendatud mitmeastmeline autentimine ja ka piiratud ligipääs. Rakendusele saab ligi ainult kindla IP poolt pöördudes, kus esimese asjana tuleb kasutada riigi autentimisteenust (TARA), mille abil kontrollitakse kasutaja: juhul kui valideeritav kasutaja ei ole arstide nimekirjas, siis rakendusele ligi ei pääse, vt Joonis 3.



Joonis 3. KMT rakenduse töö käik

4.2 Rakenduse tööks vajalike võrguühenduste kaardistamine

Rakenduse käivitamiseks oli vaja analüüsida, kuidas uues keskkonnas seadistamine käib ja millised on kriitilised elemendid, et rakendus suudaks üldse käivituda. Esimese asjana tuli kaardistada vajalikud teenused ja mis IP kaudu rakendused suhtlevad, vt Joonis 4. Sellele järgnevalt tuli vajalik teave edastada ettevõtte võrguhaldurile, kes vajalikud tulemüürireeglid tekitas. Üldjuhul rakenduste omavahelise suhtluse jaoks on vaja luua ühendustunnel kahe IP vahel, esimene oleks peremeesmasina IP ehk käesoleval juhul Kubernetese kobara IP ja teine oleks välise rakenduse masina IP koos pordiga.



Joonis 4. Rakenduse tööks vajalikud välised teenused

Peremeesmasina IP tuleb võtta kobara IP-ks seepärast, et rakendus ise käivitatakse kaunana. Kaun on Kubernetese kõige väiksem osa, mis hoiab endas konteinerrakendust. Kubernetes on disainitud selliselt, et ainult kobarasiseselt saab rakendusega ühenduda ja suhelda. See on kui kinnine süsteem, millega on tõstetud turvalisust. Kui vajalikud võrgureeglid on tehtud, siis rakendus suudab suhelda väliste teenustega läbi võrgureeglite, kuid vastupidi ühendumine on raskendatud. Kui väline teenus soovib ühenduda rakenduse pihta, siis tuleb Kuberneteses luua teenus (*service*) ja sissepääs (*ingress*). KMT rakenduse puhul sai valitud ClusterIP ja välismaailmale lahti tegemiseks sissepääs, vt Joonis 5.

```
apiVersion: v1
kind: Service
metadata:
  name: kmt-app-service-demo
spec:
  clusterIP: 1.1.1.1 #kobara IP, muudetud kujul
  clusterIPs:
  - 1.1.1.1
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - port: 80
    protocol: TCP
    targetPort: 8080 # mis pordi peal rakendus töötab
  selector:
    app: kmt-app-demo # silt millega leitakse kaunad
  sessionAffinity: None
  type: ClusterIP # teenuse nimi
```

Joonis 5. Teenuse valmistamise näide

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: kmt-app-ingress-demo
  namespace: kmt-app-demo
spec:
  ingressClassName: nginx
  rules:
  - host: kmt.demo.ee # teenuse URL on muudetud
    http:
      paths:
      - backend:
          service:
            name: kmt-app-service-demo # varasemalt tehtud teenuse nimi
            port:
```

```
            number: 80
      path: /kmt
      pathType: ImplementationSpecific
tls:
- hosts:
  - kmt.demo.ee
```

Joonis 6. Sissepääsu valmistamise näide

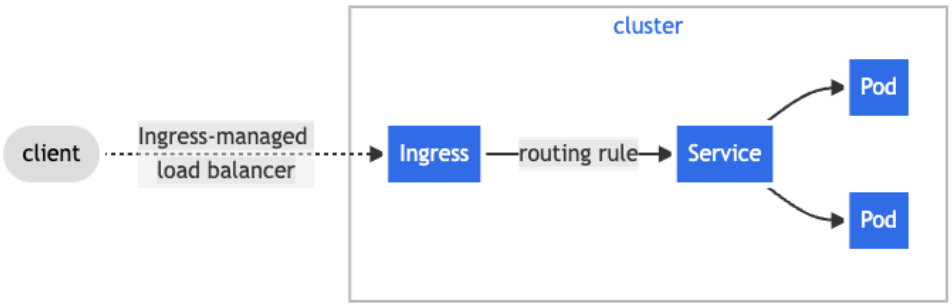
4.2.1 Kubernetese teenus ja sissepääs KMT rakendusse

Igal Kubernetese kaunal on oma IP-aadress, mis muutub pidevalt. Kui rakenduses tehakse muudatusi, siis tekib uus kaun, millele määratakse kontrolleri abil dünaamiliselt uus IP. Sellise pideva IP muutumise tagajärjel on põhimõtteliselt võimatu hoida kauna IP-d ajakohasena ja nende kaudu ühenduda. Selle jaoks ongi loodud Kubernetese keskkonnas teenus, mis loob rakenduse kaunadest lõpp-punktid, mille kaudu saab nendele ligi. Teenuse loomisel kaunad ühendatakse siltide (*label*) abil, mitte IP-aadresside järgi, mis on pidevas muutumises ja sellega peidetakse reaalset IP-d kliendi eest, vt Joonis 6. Teenusel on kobarasisene virtuaalne IP aadress, mis suunab ühenduse kaunade pihta ehk lõpp-punktid. [7] Kubernetese teenuseid on viis erinevat, mis on välja toodud järgnevalt:

- *ClusterIP* – Vaikimisi variant teenuste loomisel. Selline teenus paljastab ainult kobarasisese IP, mida väljaspool kobarat ei ole võimalik kasutada.
- *NodePort* – Selle teenuse kaudu on võimalik rakendus välismaailmale lahti teha, määratakse igale sõlmele muutumatu pordi. See ei ole turvaline variant ja see ei ole soovitatav teenus.
- *LoadBalancer* – See teenus teeb rakenduse maailmale ligipääsetavaks kasutades pilveteenuste pakkuja koormusjaoturit, tehes automaatselt ClusterIP ja NodePort teenuse, mille kaudu liiklust reguleerib.
- *ExternalName* – Eriline teenus, mis ei kasuta silte vaid DNS-nimesid.
- *Headless* – Seda teenust kasutatakse siis, kui ole vaja koormusjaoturit ega ka kobara IP-d.

NodePort on üks viis, kuidas rakendus maailmale avatuks teha, aga see ei ole turvaline, kuna rakendus ühendatakse otse sõlme külge. Turvalisema viisi jaoks on loodud

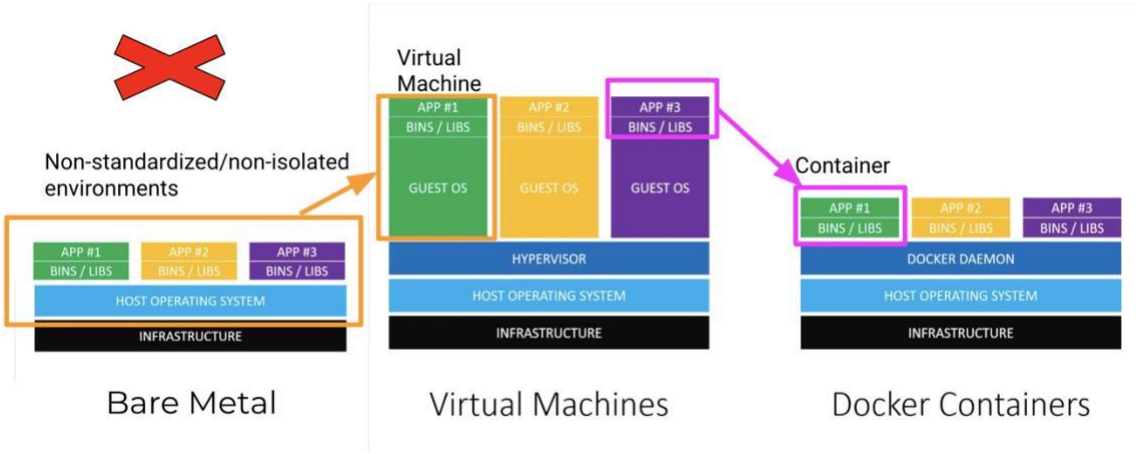
Kubernetese sissepääs, mis sisaldab endas võrguliikluse reegleid. Kui võrdlus tuua, siis see on nagu virtuaalne ruuter. Sissepääs paljastab HTTP- ja HTTPS-liikluse väljaspool kobarat vastu kobarasisest teenust. Seda kasutades on võimalik teenusele määrata URL, millega veebirakendust kasutada saab. Joonisel 7 on kujutatud, kuidas sissepääs töötab.



Joonis 7. Sissepääsu visuaalne näide [8]

4.3 KMT konteineriseerimine

Iga Dockeri konteiner on ehitatud tõmmise peale, see on kui valmis minisüsteem: sisaldab endas programmikoodi, teeke ja muid sõltuvusi, mis on rakenduse käivitamiseks vajalikud. [9] Konteinerid töötavad täielikult isoleeritud keskkonnas, neil on oma protsessipuu, oma failisüsteem ja nagu varasemalt mainitud, siis ka oma IP-aadress, vt Joonis 8. Konteinerid on ehitatud kasutades etalontõmmist, mis sisaldavad käivitamiseks vajalikke koostisosi ja rakenduse serveerimiseks vajalikke rakendusi. KMT rakenduse puhul on see kõik määratud *Dockerfile*'i failis: mis failid ja kaustad tuleb tõmmise sisse kopeerida. *Dockerfile*'is on ka võimalik öelda, millise kasutajaga konteiner käivitatakse ja millises kaustas rakendus käivitatakse, vt Joonis 9.



Joonis 8. Füüsilise serveri, virtuaalmasina ja Docker konteineri näiline struktuur [10]

```
FROM openjdk:11-jre-slim
RUN useradd kmt -u 10001 --create-home --user-group # teeb kasutaja nimega
kmt, kelle id on 10001
WORKDIR /kmt # kaust, kus rakendus käivitatakse
COPY kmt-demo.war /kmt/kmt-demo.war
USER 10001 # kasutaja kellena kaun käivitatakse, kui seda ei määra, siis
vaikimisi on root kasutaja
EXPOSE 8080 # port millel rakendus käivitatakse
ENTRYPOINT ["java","-jar","/kmt/kmt-demo.war"]
```

Joonis 9. *Dockerfile* näidis

4.3.1 Rakenduse sätted erinevas keskkonnas

KMT rakendus on ehitatud SpringBoot raamistiku peale, mis tähendab seda, et peenseadistamiseks kasutatakse välist sätetefaili. Tavaliselt on välise sätetefaili nimi *application.properties*, kus sees hoitakse võti-väärtus paare. Näiteks andmebaasiga ühendumiseks vajalik IP, kasutaja ja selle kinnitamiseks ka salasõna. Standardis sätetefail esineb kahes kohas, üks neist on war-faili sees ja teine on väline, mis asub war-faili käivitamise kaustas. Kubernetese keskkonnas käib rakendusele vajalike andmete edastamine mugavamalt ja turvalisemalt. Selleks kasutatakse *ConfigMap'i* ja saladust (*secret*).

4.3.2 Kubernetes ConfigMaps

Virtuaalmasinas rakenduse sätetefail on üldjuhul üks suur fail, kus on kogu teave korraga koos, näiteks kasutaja, ühenduse IP ja kasutaja salasõna. Kubernetes on seadistamise kahes osas tehtud, ühes hoitakse tundliku teavet ja teises mittetundliku teavet. [11] ConfigMap'is tavapäraselt hoitakse võti-väärtus paare aga on ka võimalik terve sätetefail ühe ConfigMap'i külge siduda ja see siis kauna sees sobivasse kohta paigaldada, vt Joonis 10. KMT rakenduse osas jaotati SpringBoot rakenduse sätetefail mitmeks erinevaks osaks. Kasutajad ja salasõnad eraldi, erinevad teenused eraldi, näiteks autentimisteenused ühte kokku, andmebaasi andmed eraldi ja viimaseks rakenduse andmed eraldi, et oleks struktuur ja vajalikud andmed kergesti leitavad.

```
apiVersion: v1
data: #Selline näide on võti-väärtus paar
  http.connectionRequestTimeout: "6000"
  http.connectionTimeout: "6000"
  http.customKeepAlive: "true"
```

```
  http.maxConnections: "500"
  http.socketTimeout: "6000"
  log4j2.properties: |-  #Ühe võtme taga on terve seadistusfail
     status = error
     name = PropertiesConfig

     filters = threshold
kind: ConfigMap
metadata:
  name: kmt-application-settings-demo # ConfigMapi nimi
  namespace: kmt-app-demo # nimeruum, võimalik eraldada erinevaid
rakenduse osi, nt andmebaas ja rakendus
```

Joonis 10. Kubernetes ConfigMap'i näide

4.3.3 Kubernetes Secrets

Kubernetese saladus (*secret*) aitab rakenduse tundlikku teavet jagada kobarasiseselt. Enamasti kasutatakse saladust kasutajanime ja salasõna edastamiseks rakendusele krüpteeritud kujul. Tänu Kubernetese saladuste (objektid: kaunad, teenused) iseorganiseerumisele kobaras on suureks plussiks kasutamise lihtsus. [12] Saladusi tehes, peab valima ka saladuse tüübi, nende seas on viis rohkem levinud:

- *Basic-auth* – Kasutaja ja salasõna autentimine.
- Opaque – Võti-väärtus paarid.
- *Registry* – Saladus, kus määratakse ära registry, kasutaja ja salasõna, et turvaliselt konteinereid alla laadida.
- *TLS* – TLS võtme hoiustamiseks.
- *SSH Key* – SSH avaliku ja privaatvõtme hoiustamiseks.

KMT rakenduse puhul võeti kasutusele *Opaque*, ehk võti-väärtus paaridega saladus. Kuna rakendus on varasemalt samasuguse teabe omandanud sätetefailist võti-väärtus paaridena, siis tuleb ka uues keskkonnas niimoodi teha, vt Joonis 11. *Registry* saladust kasutatakse näiteks *GitLab Registry'*st arendaja poolt valmis ehitatud Dockeri tõmmise allalaadimiseks mõeldud kasutaja salastamiseks.

```
apiVersion: v1
data:
  kasutaja: #kasutaja krüpteeritud kujul
  salasõna: #salasõna krüpteeritud kujul
kind: Secret
metadata:
  name: kmt-app-secrets-demo
  namespace: kmt-app-demo
type: Opaque
```

Joonis 11. Kubernetese saladuse näide

4.3.4 Kuberneteses juurutamine

Kuberneteses juurutamine esindab mitme samasuguse kauna olemust. Juurutamine käivitab soovitud koguse rakenduse kaunu, jälgides nende olekut. Kui mõni kaun peaks katki minema või ei vasta päringutele, siis juurutamine automaatselt loob uue kauna ja katkise kauna kustutab ära, et rakendus oleks lõppkasutajale kogu aeg kättesaadav. Juurutamised koosnevad rakenduse käivitamiseks vajalikust teabest. Selles märgitakse ära: milline tõmmis käivitatakse konteineris, mis sätetefail sellele külge lisatakse, mis silt on rakendusega seotud, et hiljem luua teenus ja sissepääs. Iga kord kui muudetakse midagi juurutuses, siis luuakse uued kaunad muudatustega ja vana kustutatakse alles siis, kui soovitud kogus uusi kaunu on tekitatud ja valmis. [13]

KMT rakendus suhtleb mitme erineva teenusega X-tee kaudu. Siis igal teenusel on oma sätetefail, mille sisu on enamasti muutumatu aga rakenduse tööks vajalik. Seepärast võttis autor kasutusele nende failide lisamise ühisesse ConfigMap'i ja need lisatakse kauna sees rakenduse käivitamise kausta.

4.3.5 Horisontaalskaleerimine

Seoses kasutajate arvu tõusuga või rakenduse täiendamisega võib tõusta ressursi vajadus, mille tulemusena rakendus ei ole võimeline õigeaegselt töötlema kõiki klientide poolt tehtud päringuid. Sellise probleemi lahendamiseks on üldjuhul järgmised võimalused:

- Vertikaalne skaleerimine ehk riistvara täiendamine.
- Võimsama vastu vahetamine.

- Horisontaalneskaleerimine ehk rakendus käivitatakse mitme kauna peal ja vajadusel lisatakse neid automaatselt juurde, et kliendi päringud õigeaegselt vastuse saaksid.

Kuberneteses on võimalik määrata vajalikud piirid, näiteks kui protsessori kasutus tõuseb üle seitsmekümne protsendi, siis automaatselt lisatakse kaks kauna juurde, vt joonis 12.

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: kmt-app-demo-hpa
  namespace: kmt-app-demo
spec:
  maxReplicas: 4 # koormuse all kaunade arv
  metrics:
  - resource:
      name: cpu
      target:
        averageUtilization: 70 # protsent, mille ületamisel luuakse kaunasid juurde
        type: Utilization
    type: Resource
  minReplicas: 2 # tavaolekus kaunade arv
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: kmt-app # rakenduse deployment nimi, mida jälgitakse
```

Joonis 12.Horisontaalne skaleerimise näidis

4.4 Käivitamine uues keskkonnas

Rakenduse käivitamisel tuli koheselt probleeme, mida oli oodata. Kui monoliitrakendus ühest keskkonnast teise liigutada, siis tuleb arvestada võimalike probleemidega. Koheselt tuli välja, et rakenduse ei suuda käivituda kuna URL, mille pealt uus KMT vastab, ei ole rakenduse andmebaasis lubatud linkide seas. URL'i lisamisel tekkis järgmine probleem autentimisega: kuna rakendus proovis autentimisest mööda minna ja kohe kasutamist lubada aga KMT turvameetmed on niivõrd tugevad, et kui sessioon on puudu, siis pole võimalik rakendust kasutada. Selle tagajärjel oli rakendus ringluses, iga vajutusega suunas samasse kohta tagasi ja kui vajutati katkesta, siis suunas valesse kohta.

Esimeses etapis kasutatakse *Keycloak*-autentimist, mis suunab omakorda TARA peale. Analüüsides rakenduse käitumist, tuldi järeldusele, et uus URL ei ole lisatud *Keycloak'*i sätetesse, mille tulemusena rakendusel ei ole lubatud selle külge ühenduda. Kui need muudatused oli tehtud, siis ilmnesid uued probleemid. Peale kasutaja tuvastamist ja asutuse valimist rakendusele tuli hoiatus: „sessioon on katkestatud, palun sulgege rakendus ja logige uuesti sisse". Autor uuris logisid, kontrollis mitmeid kordi üle, kas kõik ligipääsud on loodud ja vajalikud liidestused tehtud, siis selgus, et rakendus ei suuda konteineris korralikult töötada. Rakendus loob iga kasutaja jaoks sessiooni andmed andmebaasi, mida siis pidevalt kontrollitakse, et ei oleks kõrvalkaldeid ja sessiooni kaaperdamisi. Arendajale sai edastatud probleemikirjeldus, mille tulemusena valiti sessioonide hoidmiseks teine lähenemine: võeti kasutusele Redis. Arendaja poolt sai rakenduse kood muudetud, et sessioone hoitakse Redis'es. Autori ülesanne oli paigaldada Redis ja luua ühendus rakendusega.

Redis on mälusisene andmestruktuuride hoidla, mille pärast on see väga kiire teabevahetusega süsteem. Mälusisene andmestruktuur tähendab seda, et rakenduse teave hoitakse mälu, mitte ketta pinnal - sellega saavutatakse kiire reageerimine. Seda kutsutakse andmestruktuuri serveriks, sest Redis suudab endas hoida mitmeid erinevaid andmevorme, mitte ainult võti-väärtus paare. Redise kasutuselevõtuga saab rakenduse sessioonid selle andmebaasis hoida. Kui rakendus teeb läbib uuenduse, siis Redis'e baasist tuleva teabega seotakse klient koheselt uuesti tema sessiooniga ja tema jaoks katkestust ei teki. [14]

Redis'e paigaldamiseks kasutati Bitnami Helm Charti, mille kasutamine on Rancheris väga mugavaks tehtud: tuleb valida ainult nimeruum ja millist versiooni soovid. Hea tava on, et rakendus ja andmete hoidmisega seotud teenused peaks hoidma eraldi nimeruumides, et üks nimeruum liiga suureks ei läheks ja oleks korralik struktuur. Näiteks rakenduse nimeruum on *kmt-app-demo*, siis vahebaasi nimeruum oleks *kmt-base-demo*. Kuberneteses on võimalik erinevate nimeruumide vahel teavet jagada, tuleb viidata sobivale nimeruumi ja selles olevale teenusele. Redis töötab üldjuhul kobarana, kus on üks peremeesmasin ja kolm koopiat, mis omavahel pidevalt teavet vahetavad.

Redis'e kasutuselevõtuga, tuli teha uus saladus, kus oli märgitud Redis'e kasutaja, salasõna ja millises kobaras ja mis teenuse kaudu temaga ühenduda saab. Kui kõik vajalikud sätted olid seadistatud, siis KMT oli kasutamiseks valmis. Uue lahendusega

sessiooni probleeme ei esinenud ja rakendust sai edukalt kasutada ühe kaunaga. Uued vead tulid välja, kui rakendus käivitati mitme kauna peal. Mitme kauna peal rakendus ei laadinud korralikult veebilehe sisu ära, mille tagajärjel klient nägi poolikut rakendust. Alles korduval veebilehe uuendamisel oli võimalik rakendust kasutada. Probleem seisnes selles, et rakendus oli disainitud selliselt, et töötab ainult ühe õla peal. Kui mitme õla peal käivitati siis salvestati pooled rakenduse andmed ühe kauna peale ja teised teise peale aga ideaalis peaks mõlema kauna peal olema samad andmed. Probleem sai edastatud arendajale ning vajalikud muudatused sai teostatud. Uute muudatustega rakendus suutis mitme õla peal töötada ja kliendi vaates kasutajakogemus jäi samasuguseks.

5 Tehtud töö analüüs

Lõputöö tulemusena paigaldati KMT rakendus uues keskkonnas, kus tuli omajagu probleeme, mis on ka loogiline, kui viia monoliitsüsteem üle teise keskkonda. Monoliitsüsteemiga võib esineda ka teatud takistusi sama keskkonna üleviimisega: tihti on probleeme sama rakendusega arenduse ja testkeskkonna vahel, sest iga üksikasi võib rakenduse teistmoodi tööle panna. Üldjuhul proovitakse hoida süsteemid samad, kus rakendusi testitakse ja käivitatakse. Kuna monoliidi puhul sõltub palju operatsioonisüsteemist, millel rakendus töötab, siis erinevused on kerge tulema: selle tulemusena võib esineda veateateid. Mikroteenuste puhul proovitakse seda vältida, sest kogu rakenduseks tööks vajalik sisu on konteineris ja peaks töötama igas operatsioonisüsteemis samamoodi.

Üleviimine oli autori jaoks keeruline ja aeganõudev, sest varasemalt virtuaalmasina peal töötava rakenduse taristu ei olnud tema paigaldatud. Rakenduse iseärasusi õpiti üleviimise teel uue keskkonna logisid analüüsides, kuna selle käigus ei saa alati toetuda vana keskkonna tavadele ja proovida lahendust matkida. Lisaks KMT rakendus sõltub suuresti välistest teenustes: tuli kaardistada kõik teenused ja analüüsida, kuidas need peaksid uues keskkonnas ligipääsetavad olema ja milline teave tuleb edastada teiste teenuste administraatoritele.

Üleviimise suurimaks eesmärgiks oli katkestusteta uuendamine, mis teoorias oleks ka võimalik olnud vana keskkonnaga aga nagu üleviimise käigus välja tuli, siis rakendus vajas lisaarendusi, et suudaks mitme masina peal teavet jagada. Varasemalt virtuaalmasina peal minimaalne uuendamise aeg oli umbes üks minut, mille tulemusena alati klientide sessioonid katkestati ja sessioonis talletatud teave võis kaduda. Uuendamise käigus peatatakse vana versioon, kustutatakse war-fail virtuaalmasinas, laetakse uus fail masinasse ja seejärel käivitatakse rakendus uue versiooniga. Kui uuendamisel esineb probleeme, siis rakenduse ei ole klientidele ligipääsetav. Selle jaoks on Kubernetese keskkonnas katkematu uuendamine, kus tehakse uued kaunad ja ainult siis kustutakse vanad ära, kui uued töötavad vigadeta. Kubernetes kasutab selle jaoks elavus teabeobjekti (*liveness probe*) ja valmisoleku teabeobjekti (*readiness probe*). Elavus teabeobjekt kontrollib, kas rakendus käivitub, juhul kui ei käivitu, siis teeb kaunale taas käivituse. Valmisoleku teabeobjekt kontrollib, kas kõik kaunad töötavad ja on valmis

võrguliiklust vastu võtma, kui üks kaun ei vasta nõuetele, siis sellele liiklust ei suunata. Selle tulemusena klient ei tunneta uuendamise mõju isegi siis, kui rakenduse uuendamine ebaõnnestus. Kuberneteses on rakenduse uuendamine võrdlemisi lihtne, tuleb ainult vahetada tõmmise number või nimetus ja ülejäänud toimub kõik automaatselt.

Teine eesmärk oli ressursi kasutamise vähendamine. Kuna üks eesmärk oli katkematu uuendamine, siis algse lahendusena oleks see tähendanud kahe virtuaalmasina kasutamist, millele on paigaldatud rakendus ja sellele vajaminevad lisarakendused. Virtuaalmasinas üldjuhul antakse masinale kindel arv protsessori tuumi ja kindel hulk mälu ning nende suurendamiseks tuleb ise neid juurde lisada. Kubernetese keskkonnas on võimalik rakenduse konteinerile anda vähim vajalik arv protsessoreid, mis on vaja kindlasti broneerida rakenduse käivitamiseks. Üldjuhul on see väike osa ühest tuumast kuna konteinerid enamasti ei vaja oma tööks isegi ühte protsessori tuuma. Lisaks on võimalik ära määrata piirangud, mis on ühele rakendusele lubatud ressursside ülemmäär. Seal tuleb meeles pidada, et määratud ressurss on ühe kauna kohta ehk mitme kauna kasutamisel korrutatakse andmed kaunade arvuga. Kui rakendus läheneb piirangute ülemmääradele, siis automaatselt tekitatakse soovitud arv kaunu juurde, et kliendi jaoks ei tunduks rakendus aeglane. Alljärgnev tabel 1 võrdleb KMT rakenduse ressursi kasutamist erinevates keskkondades.

	Virtuaalmasinas KMT rakendus	Kubernetese keskkonnas KMT rakendus
CPU broneering	4 CPU	250 millicpu ~ 0.25 CPU
CPU kasutus puhkeolekus	2%	40 millicpu ~ 0.04 CPU
Mälu broneering	4096 MiB	1500 MiB, Limiit: 4000 MiB
Mälu kasutus puhkeolekus	1945 MiB	527 MiB
Uuendamise pikkus	~ 1 minut	~14 sekundit
Uuendamisel tekkinud katkestuse aeg	~ 1 minut	0 sekundit

Tabel 1. KMT rakenduse võrdlus erinevates keskkondades

Uue lahendusega on kasutajakogemus parem kuna uuendamisel katkestuse aega enam ei ole. Uue lahenduse puhul teavitatakse kasutajaid uuendusest, katkestus ei ole planeeritud. Varasemalt oli katkestus planeeritud terve hooldusakna ulatuses, mis omakorda tähendas peaaegu teist sama palju aega andmete tagantjärele sisestamiseks. Samal ajal on kasutajat segamata aega analüüsida, kas uuendus oli edukas. Kuna hetkel on käimas arendusfaas, siis suurema kasutajate hulgaga testimised on veel ees ootamas.

5.1 Tulevikuarendused

Lõputöö käsitles hetkel ainult rakenduse üleviimise protsessi ühelt keskkonnalt teisele ja mis etapid olid vajalikud rakenduse käivitamiseks teises keskkonnas. Tulevikus võiks rakenduse käitumist samm-sammult muuta mikroteenuste arhitektuurile vastavaks. Sessiooni hoidmine liikus andmebaasist mälusisese andmestruktuuride hoidla Redis'e peale. Tulevikus oleks hea, kui kogu rakenduse vahemälu töötaks Redis'e peal: siis rakenduse kiirus peaks kordades suurenema, mis omakorda võrduks vajalike sekundite võiduga abivajaja jaoks.

Lisaks mikroteenuste heaks tavaks on rakenduse jagamine erinevateks osadeks, et ei peaks kõike korraga uuendama. Näiteks rakenduses eraldada eesrakendus kliendile ja tagarakendus, mis kuvandile vastab. Lisaks võiks tulevikus tekkida võimekus isikukoodi sisestamiseks automaatselt. Hetkel peab rakenduses käsitsi sisestama patsiendi isikukoodi, tulevikus võiks saada skaneerida ID-kaardi triipkoodi ja automaatselt see rakendusse sisestada, et vältida inimvigu.

6 Kokkuvõte

Käesoleva töö põhieesmärk oli KMT rakenduse edukas üleviimine Kubernetese keskkonda ja seal omakorda uuendamine ilma katkestusteta kliendi jaoks. Teiseks eesmärgiks oli ressursi kasutamise vähendamine ja vajadusel koormuse tõustes rakenduse automaatne horisontaalskaleerimine. Antud töö raames kõik eelnimetatud eesmärgid said edukalt täidetud.

Teoreetilises osas kirjeldati ja võrreldi mooliit- ning mikroteenuste arhitektuuri eeliseid ja puuduseid. Lisaks lühidalt KMT rakendusest endast ja tema eripäradest.

Esimesena tuli KMT konteineriseerida ja alles siis sai seda mikroteenusena juurutada. Teise etapina analüüsiti rakenduse eripärasid ja kuidas need uues keskkonnas lahendada. Selle käigus kaardistati väliste teenuste ühendused ja vormistati tulemüürireeglid uuele keskkonnale. Selgitati lahti, kuidas Kuberneteses rakendused väliste teenustega ühenduvad ja millised on normid. Lisaks kirjeldati seadistamise protsessi ja millised on erinevused erinevate keskkondade vahel.

Töö praktilises osas paigaldati rakendus Kubernetese keskkonnas ning seadistati eelnevalt kirjeldatud meetodeid jälgides. Paigalduse käigus uuriti rakenduse logisid ja veateateid, mille tulemusena muudeti seadistust vastavalt olukorrale. Uurimuse tulemusena jõuti tulemusele, et rakendus vajab lisaarendust ja tuleb võtta kasutusele Redis. Redis’e kasutuselevõtuga lahendati sessiooni probleemid, mille tulemusena oli võimalik rakendust käivitada uues keskkonnas. Töö kõik eesmärgid said edukalt täidetud.

Kasutatud kirjandus

[1] cPanelDocs, „ Installation Guide – System requirments for CentOS“.[Võrgumaterjal].https://docs.cpanel.net/installation-guide/system-requirements-centos/. [Kasutatud 15 aprill 2022].

[2] Albert Toro Marin, „Containerizing ONAP using Kubernetes and Docker“.[Võrgumaterjal]. https://upcommons.upc.edu/bitstream/handle/2117/168805/memoria.pdf . [Kasutatud 19 aprill 2022]

[3] GeeksforGeeks, „Monolithic vs Mircoservices arhitecture“,[Võrgumaterjal]. https://www.geeksforgeeks.org/monolithic-vs-microservices-architecture/ .[Kasutatud 15 aprill 2022]

[4] Medium, „Monolith vs Microservices“, [Võrgumaterjal]. https://medium.com/hengky-sanjaya-blog/monolith-vs-microservices-b3953650dfd . [Kasutatud 30 november 2021]

[5] Microsoft,“Microservices arhitecture style“.[Võrgumaterjal]. https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices .[Kasutatud 15 aprill 2022]

[6] Haigekassa, „Vältimatu abi vajava inimese teekond on moodsa e-kiriabi loomise alus“.[Võrgumaterjal]. https://www.haigekassa.ee/blogi/valtimatu-abi-vajava-inimese-teekond-moodsa-e-kiirabi-loomise-alus. [Kasutatud 20 aprill 2022]

[7] Leila Abdollahi Vayghan, Mohamed Aymen Saied, Maria Toeroe, Ferhat Khendek.
„Kubernetes as an Availability Manager for Microservice Applications“.[Võrgumaterjal]. https://arxiv.org/pdf/1901.04946.pdf . [Kasutatud 15 aprill 2022]

[8] Kubernetes, „Ingress“.[Võrgumaterjal]. https://kubernetes.io/docs/concepts/services-networking/ingress/[Kasutatud 15 aprill 2022]

[9] Charles Anderson, „Docker“.[Võrgumaterjal].https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7093032 [Kasutatud 22 aprill 2022]

[10] https://miro.medium.com/max/3684/1*2osKvlHsRAY5cBj--_hBmw.png[Kasutatud 24 aprill 2022]

[11] Kubernetes, „ConfigMaps“.[Võrgumaterjal]. https://kubernetes.io/docs/concepts/configuration/configmap/[Kasutatud 18.04.2022]

[12] Magalix, Joydip Kanjilal, „Kubernetes Secrets and their Security“,[Võrgumaterjal]. https://www.magalix.com/blog/kubernetes-secrets-and-their-security[Kasutatud 18.04.2022].

[13] Google Kubernetes Engine, „Deployment“.[Võrgumaterjal]. https://cloud.google.com/kubernetes-engine/docs/concepts/deployment. [Kasutatud 19 aprill 2022]

[14] Tiago Macedo, Fred Oliveira, „Redis Cookbook“.[Võrgumaterjal]. http://117.3.71.125:8080/dspace/bitstream/DHKTDN/6831/1/6184.Redis%20cookbook.pdf [Kasutatud 21 aprill 2022]

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Jaanus Steinfeldt

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Rakenduse Kiirabi Mobiilne Töökoht migratsioon Kubernetese keskkonda“, mille juhendaja on Edmund Laugasson.
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

16.05.2022

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.