Önálló laboratórium beszámoló

Távközlési és Médiainformatikai Tanszék

Készítette:	**Fazekas Ferenc Albert**
Neptun-kód:	**IW4FRZ**
Ágazat:	**FELHŐ ALAPÚ HÁLÓZATOK**
E-mail cím:	**khhun12@gmail.com**
Konzulens(ek):	**Dr. Maliosz Markosz** **Dr. Simon Csaba**
E-mail címe(ik):	**maliosz@tmit.bme.hu** **simon@tmit.bme.hu**

Téma címe: Szavazó alkalmazás megvalósítása Kubernetes és Docker segítségével

Feladat

Ha nem az előtted lévő számítógéped erőforrásait használod, máris könnyen a felhőben találod magad! A műhelycsoport elsősorban a felhőben futó hálózati szolgáltatások (pl. mobil alkalmazások háttértámogatása vagy tartalomelosztó rendszerek) optimalizált létesítésére és működtetésére fókuszál. Ehhez a következő ismereteket lehet elsajátítani: a szerverek és a hálózatok virtulaizációja, virtualizációs környezetek (virtuális gép, konténer (pl. Docker) stb.) Fontos cél egy komplex szolgáltatást megvalósító virtuális komponensek automatizált telepítése, skálázása és működtetése, e menedzsment (orchestration) szoftver eszközök ismerete hasznos gyakorlati tudást biztosít, és a DevOps szemlélet is elsajátítható. A technológia megismerésén felül a megtanultak lehetővé teszik a gyakorlati alkalmazás során a felhő platformok telepítését, konfigurálását, üzemeltetését, illetve optimalizálását.

2020/2021. 1. félév

1. A laboratóriumi munka környezetének ismertetése, a munka előzményei és kiindulási állapota

1.1 Bevezető

A témalabor során megismerkedtem több virtualizációs technológiával, többek között a Docker-rel és Kubernetes-el, melyek konténer alapú virtualizációt valósítanak meg. A tárgy során alulról felfelé haladtunk az architektúra működésének megismerésében. Az első felében a tárgynak a Docker alapjait tanultam meg ami a felhő alapú szolgáltatások mondhatni alapvető építőköve, a másodikban pedig a Kubernetes-ről tanultam, azaz, hogy hogyan kell konténereket egységbe szervezni, valamint szolgáltatásokat skálázhatóan és effektíven működtetni.

A feladatom ehhez kapcsolódóan, egy szavazó alkalmazás megvalósítása volt, Docker és Kubernetes segítségével.

1.2 Elméleti összefoglaló

A virtualizáció már régóta az informatikában jelenlévő dolog, hiszen így hatékonyabban tudunk szolgáltatásokat, applikációkat megvalósítani. Alapvető célja az erőforrások hatékonyabb kihasználása, a fizikai réteg problémáinak kikerülése, és a platformfüggetlenség, absztrakció[1].

Az első lépcsőfok a virtualizáció fejlődésébe a virtuális gépek alkalmazása volt, itt a host-OS hardveres és/vagy szoftveres támogatásával egy guest-OS-t futtattunk, egy külön operációs rendszert annak minden hozományával együtt, itt az volt a probléma, hogy viszonylag lassú a beüzemelés, nagy overhead (filesystem, process-ek stb.), ezért nem erőforrás optimális egy célfeladat megoldására[1].

Erre találták ki később a konténerizációt, ahol az atomi egység az a „container”, ami process szintjén virtualizál, jóval kisebb erőforrás igénnyel, kényelmesebb beüzemeléssel, valamint sokkal jobb erőforráskihasználással[2]. A container egy container image-ből áll elő, ez tárolja a különféle filesystem layer-eket és egyéb, fontos paramétereit.

Még egy lépcsőfoknak tekinthető ,például, ha olyan szolgáltatást akarunk megvalósítani, ami időben változó erőforrásigénnyel rendelkezik, hogy hogyan orchestráljuk az előző szinten létrehozott konténer szolgáltatásokat, mikor indítunk belőlük, hányat, milyen paraméterekkel stb., ezt hivatott megvalósítani a Kubernetes, amivel egy magasabb szintről tudunk konténereket menedzselni, itt az atomi egység az a „pod”, ami tartalmazhat több container image-t is[3]. A Kubernetes bevezet számos egyéb funkciót is, mint például a service és deployment fogalmát[4][5]. Amelyek nagyban megkönnyítik a magasabb szintről való menedzselést. Többek között számos beépített funkcióval rendelkezik, például beépített terhelés elosztó, ami szintén levesz némi terhet a fejlesztő válláról.

Valamint fontos része mind a Kubernetes-nek mind a Docker-nek az úgynevezett hálózati absztrakció, aminek segítségével egyszerűbb hálózatkezelést valósíthatunk meg a szolgáltatásainknak, pl.: webszerver példányok között a terheléselosztó automatikusan forward-olja a kintről érkező csomagokat.

2. Az elvégzett munka és eredmények ismertetése

1.1 A munkám ismertetése

Bevezető

A feladatom egy szavazó alkalmazás megvalósítása volt, ehhez Kubernetes-t és Docker-t használtam egy VMware virtuális gépen. Az előre kiadott minta projekt alapján[6]. 5 db entitás van jelen ebben. A voting-app, ami egy Python segítségével elkészített szavazó webfelület, itt tudunk szavazni. A redis, ami eltárolja Redis segítségével a bevitt szavazatot. A worker, ami a redis és a db közötti összeköttetésért felelős. A db, ami egy PostgreSQL adatbázis, a perzisztens adattárolásért felel. Valamint egy result-app, ami Node.js segítségével kiírja a bevitt szavazatokat. Feladat része még a 3 nem sima public image-t használó container image legyártása, valamint saját Docker repository-ba való feltöltése.

Elképzelés

Az elképzelésem egy 5 pod-ban működő rendszer volt, úgy, hogy minden pod 1 image-t tartalmaz, mivel ez szerintem így a leginkább átlátható. A megadott minta projektben lévő entitások közül a redis és a db az egyszerű public image-t használja, tehát azokkal Docker szinten nincs dolgom. A worker-t, a result-ot és a voting-app-ot viszont Docker segítségével a mintaprojekthez hasonló módon meg kell valósítani, buildelni kell belőlük image-eket. Ezek után a megfelelő paraméterezéssel a különböző image-ket deployment-be és service-be szervezni. Valamint tesztelni, hogy helyesen működik-e a megvalósított alkalmazás.

A környezet telepítése

A legelső lépés a virtuális gép telepítése, ehhez VMware-t használva feltelepítettem egy Ubuntu18.04LTS linux operációs rendszert, 3GB RAM, 2 CPU core és 20GB háttértárral.

Ezek után szükséges a Docker telepítése, amely az 1.ábrán látható parancsokkal valósítható meg.

```
sudo apt install curl -y
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker $USER
#relog
```

1. ábra: Docker telepítési parancsok

Ezek után a Kubernetes-t kell telepíteni, amihez én Microk8s-t használtam. Ennek a telepítése a 2.ábrán látható parancsokkal valósítható meg. Az egyszerűbb kezeléshez még beállítottam egy alias-t és adtam sudo jogot mindkét esetben. Valamint, a hálózat megfelelő működése érdekében engedélyeztem a dns-t.

```
sudo apt install snap
sudo snap install microk8s --classic
sudo usermod -a -G microk8s $USER
sudo snap alias microk8s.kubectl kubectl
#relog
microk8s enable dns
```

2. ábra: Kubernetes környezet telepítése

Container image-k megvalósítása

A környezet telepítése után a következő lépés a 3 container image (vote-app, result, worker) megvalósítása Docker segítségével. Itt a szükséges file-okat az image-k build-eléséhez a mintaprojektből vettem [6].

A vote-app Dockerfile a 3.ábrán látható, a megfelelő paraméterekkel és fileokkal. Python alap image-el, 80-as alapértelmezett port-ot használva(web).

```
FROM python:2.7-alpine
WORKDIR /app
ADD requirements.txt /app/requirements.txt
RUN pip install -r requirements.txt
ADD . /app
EXPOSE 80
CMD ["gunicorn", "app:app", "-b", "0.0.0.0:80", "--log-file", "-", "--access-logfile", "-", "--workers", "4", "--keep-alive", "0"]
```

3. ábra: vote-app Dockerfile

A result Dockerfile a 4.ábrán látható, szintén egy user interface 80-as port-tal(web), NodeJS-t használva az adatbázisból lekéri szavazatokat és megjeleníti a weblapon.

```
FROM node:10-slim
WORKDIR /app
RUN npm install -g nodemon
COPY package*.json ./
RUN npm ci \
 && npm cache clean --force \
 && mv /app/node_modules /node_modules
COPY . .
ENV PORT 80
EXPOSE 80
CMD ["node", "server.js"]
```

4. ábra: result Dockerfile

A worker Dockerfile az 5.ábrán látható, ez valósítja majd meg a kommunikációt a redis és a db között, .NET framework alapú.

```
FROM microsoft/dotnet:2.0.0-sdk
WORKDIR /code
ADD src/Worker /code/src/Worker
RUN dotnet restore -v minimal src/Worker \
    && dotnet publish -c Release -o "./" "src/Worker/"
CMD dotnet src/Worker/Worker.dll
```

5. ábra: worker Dockerfile

Így előálltak a szükséges Dockerfile-ok, már csak build-elni kell őket és a saját DockerHub repository-mba feltölteni. A build-eléshez szükséges parancsokat a 6.ábra tartalmazza. Először be kell jelentkezni a saját repository-ba, majd egyenként build-elni és feltölteni a megfelelő címkével ellátott image-ket. Ez után rendelkezésre állnak további felhasználásra.

```
docker login --username=albick99
~/Work/temalab01/example-voting-app4/vote$ docker build -t vote_im .
docker images
docker tag 51a0b85d830f albick99/vote_im
docker push albick99/vote_im

~/Work/temalab01/example-voting-app4/result$ docker build -t result_im .
docker images
docker tag 497117c08cdc albick99/result_im
docker push albick99/result_im

~/Work/temalab01/example-voting-app4/worker$ docker build -t worker_im .
docker images
docker tag 270f51cdc8bf albick99/worker_im
docker push albick99/worker_im
```

6. ábra: image build parancsok

Deployment és service megvalósítása

A deployment a dep.yaml file alapján megy végbe, mellékelve látható[7]. Itt definiáljuk 5db pod deployment-jét külön egy vote nevű namespace-be az könnyebb átláthatóság végett. A db sima „postgres:9.4”-es image-t használ, ez a perzisztens adatbázisunk, be van állítva külön, hol tárolja az adatokat a filerendszerünkben, valamint a szükséges környezeti változók át vannak adva neki, az 5432-es port-on kommunikál. A redis egy „redis:alpine” image-et használ, 6379-es port-on kommunikál, és szintén be van állítva nekü adattárolásra útvonal. A result a DockerHub repository-nkból a „result_im” image-et használja, 80-as port-on. A vote, az előbb elkészített „vote_im” image-et használja, 80-as port-on. A worker szintén az előre legyártott „worker_im” image-et használja. Fontos megemlíteni, hogy most az egyszerűség kedvéért „replicas: 1” értéket használok mindenhol.

Már csak a service előállítása maradt, hogy működőképes legyen az alkalmazás, ennek a service.yaml file-ja mellékelve látható[8]. Itt létrehozzuk a service-eket. Hálózati beállításokat tekintve ClusterIP a worker, redis és db, tehát csak cluster-en belülről érhető el, mivel nekik csak egymással kell kommunikálni és nem is lenne biztonságos, ha kívülről szükségtelenül elérhetőek lennének. A 2db webes felület(result,vote) viszont NodePort, tehát kívülről is elérhető. Az egymással való kommunikációra szolgálnak a megadott targetPort-ok, a port segítségével érthető el közvetlenül a service, ha ismerjük IP címét, és nodePort segítségével érhető el kívülről.

Ahhoz, hogy működésre bírjuk a rendszerünk, az alábbi paranccsal el kell indítanunk, ami a 7.ábrán látszik.

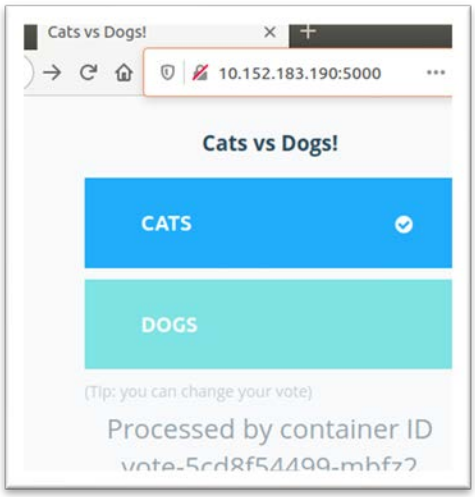
```
kubectl create namespace vote
~/Work/temalab01/example-voting-app4$ kubectl create -f mk8s/
```

7. ábra: service és deployment indítása

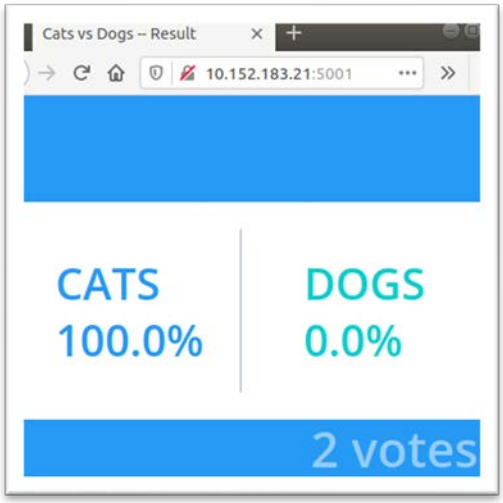
Eredmények tesztelése

A „kubectl get po -n vote” paranccsal ellenőriztem, hogy az összes pod megfelelően elindult-e. Miután mind „running” lett , a „kubectl get services -n vote” paranccsal megnéztem a szavazó felület(vote) ip címét és egy böngészőben megnyitottam a service ip címét a

service.yaml file-ban definiált 5000-es port-tal[8]. A 8.ábrán látható eredményt kaptam, tehát a vote rész működik. Hasonló eredményre jutottam parancssori böngészővel is. A szavazat leadását követően a tényleges tesztje az alkalmazásunknak a result page volt, ami a 9.ábrán láthatóan megfelelően működött és valós eredményeket megfelelő formában adott vissza. Ebből arra a konklúzióra juthatunk, hogy a rendszer egészét vizsgálva végponttól-végpontig megfelelően funkcionál.



8. ábra: vote felület



9. ábra: result felület

2.2 Összefoglalás

A feladat megoldásához alapvetően Kubernetes-t és Docker-t használtam. Architektúrálisan szemlélve alulról felfelé dolgoztam, először az image-eket csináltam meg az 5db entitáshoz külön-külön Docker segítségével, majd ezeket szerveztem össze Kubernetes alkalmazásával, ezután pedig teszteltem a megoldásom.

A félév során sok új dolgot megtanultam a Docker és Kubernetes témakörébe, ezek segítségével a feladatot nagyobb akadályok nélkül meg tudtam oldani. A félév során a leginkább a nehézség az új információk feldolgozása volt, mivel még soha nem foglalkoztam ilyesmivel, viszont érdekel a téma tehát, továbbiakban is fejleszteni szeretném belőle a tudásom, és jobban megérteni a felhő alapú alkalmazások működését.

3. Irodalom, és csatlakozó dokumentumok jegyzéke

A tanulmányozott irodalom jegyzéke:

[1] *https://www.ibm.com/cloud/learn/virtualization-a-complete-guide*

[2] *https://www.docker.com/resources/what-container*

[3] *https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/*

[4] *https://kubernetes.io/docs/concepts/services-networking/service/*

[5] *https://kubernetes.io/docs/concepts/workloads/controllers/deployment/*

Egyéb tartalmak jegyzéke:

[6] *https://github.com/dockersamples/example-voting-app*

[7] *https://github.com/albick/temalab2020votingapp/blob/master/example-voting-app4/mk8s/dep.yaml*

[8] *https://github.com/albick/temalab2020votingapp/blob/master/example-voting-app4/mk8s/service.yaml*