Témalaboratórium beszámoló

Távközlési és Médiainformatikai Tanszék

Készítette:	**Háromi Bence**
Neptun-kód:	**AK51GI**
Ágazat:	**Infokommunikáció**
E-mail cím:	**bence.haromi@gmail.com**
Konzulensek:	**Dr. Maliosz Markosz** **Dr. Simon Csaba**
E-mail címeik:	**maliosz@tmit.bme.hu** **simon@tmit.bme.hu**

Szavazó alkalmazás megvalósítása Kubernetes használatával

Feladat

Egy szavazó alkalmazás megvalósítása Kubernetes használatával, mely képes a szavazás teljes lebonyolítására, vagyis a szavazatok begyűjtésére, a szavazások eredményeinek adatbázisban való eltárolására és összegzésére, illetve ezen eredmények megjelenítésére.

2020/2021. 1. félév

1 A laboratóriumi munka környezetének ismertetése

1.1 Elméleti összefoglaló

A virtualizáció lehetővé teszi, hogy több virtuális gép (*Virtual Machine*, továbbiakban *VM*) egyidejűleg használhassa egy szerver erőforrásait (pl. CPU, Memória, stb.) [1]. Több típusú virtualizációs megoldás létezik, a hardvervirtualizációt már a 90-es évek óta alkalmazzák, a felhő rendszerek elmúlt évtizedben látott fejlődésének is az alapját képezik. A modernebb megoldásként bevezetett operációs rendszer szintű virtualizációt széles körben a Docker konténerek bevezetésével kezdték használni, ezért ezt konténervirtualizációként is említi a szakirodalom.

1.1.1 Hardvervirtualizáció

A hardver virtualizáció izolálja a *VM*-eket, és ez az izoláció biztosítja, hogy adott *VM* adataihoz másik *VM* alkalmazásai ne férjenek hozzá. A virtualizációval sokkal jobb az erőforrás kihasználtság, mivel kevesebb a „tétlen” várakozás az erőforrás megosztás miatt. Olcsóbb a szerverek fenntartása (nincs szükség minden egyes alkalmazáshoz, külön fizikai szerverre, mert futhat több *VM* egy szerveren), továbbiakban hardverfüggetlenséget biztosít, vagyis könnyen lehet a virtuális gépet másik szerverre költöztetni.

Minden *VM* saját operációs rendszerét (*Operating System*, továbbiakban *OS*) és annak minden komponensét, futtatja a virtualizált hardveren.

1.1.2 Konténervirtualizáció

A konténerek nagyon hasonlóak a *VM*-ekhez, viszont nagy különbség, hogy ebben az esetben egy közös *kernel*t használnak, a *VM*-ekkel ellentétben, azonban ugyanúgy saját, külön fájlrendszerrel rendelkeznek, memóriával és használhatják megosztottan a CPU-t. A közös *OS*-ből az egy gazdagépen (host) futó konténerek mind ugyanazt a kernelt használják, ebből adódik, hogy a konténerek kisebb tárhelyet foglalnak.

A *Docker* egy konténervirtualizációs technológia, mely teljesen rendszerfüggetlen, vagyis bármilyen operációs rendszeren futhat, könnyedén „szállítható”, megosztható, biztosítható, hogy ugyanazt a környezetet kaphassuk bárhol is fut. Ugyanaz a konténer tud futni a fejlesztő laptopján, mint ami az éles rendszerben fog a felhőben. Ez a fejlesztési folyamatot is könnyíti, nem kell az elkészült alkalmazást még átkonfigurálni a valós rendszerbe, hanem csak el kell indítani. A konténerek nagy előnye a teljes értékű virtuális gépekkel szemben, hogy csakis a szükséges fájlokat, programokat tartalmazzák, így sokkalta kisebb tárhelyet foglal, és mivel csak az adott konténer futásához szükséges alkalmazásokat tartalmazza, így jelentősen gyorsabban el tudnak indulni, mint amikor egy teljes virtuális gépet indítanánk.

A konténerek teljesen izoláltak egymástól, ez azt jelenti, hogy ugyanazon a szervergépen futó konténerek nem tudnak egymásról semmit,egymás hálózati interfészei, fájlterületei is el vannak választva.

1.1.3 Kubernetes

A *Kubernetes Docker* konténerekből álló rendszer menedzseléséért felelős. Leveszi a fenntartó vállairól a konténerek közötti hálózatkezelést, automatizálja a leálló/meghibásodó konténerek újraindítását, így ha valami baj történik, akkor automatikusan indít helyette egy újat,

szükség esetén skálázza a konténereket, vagyis ha többre van szükség többet indít, továbbá terheléselosztást végez. Mindezzel rengeteget segíti a felhasználókat.

A *Kubernetes* esetén egy gépek halmazát klaszternek hívjuk, melyben a gépeket *node*-oknak [3]. Ezeken a *node*-okon futnak *pod*ok, amik pedig konténeralkalmazásokat futtatnak. Egy úgynevezett *control plane* felelős a *node*ok és azon belül a *pod*ok menedzseléséért. Ez magában foglalja a *pod*ok ütemezését (milyen események hatására indítson egy újat, vagy állítson le egy futót).

2 Az elvégzett munka és eredmények ismertetése

2.1 Docker konténerek készítése

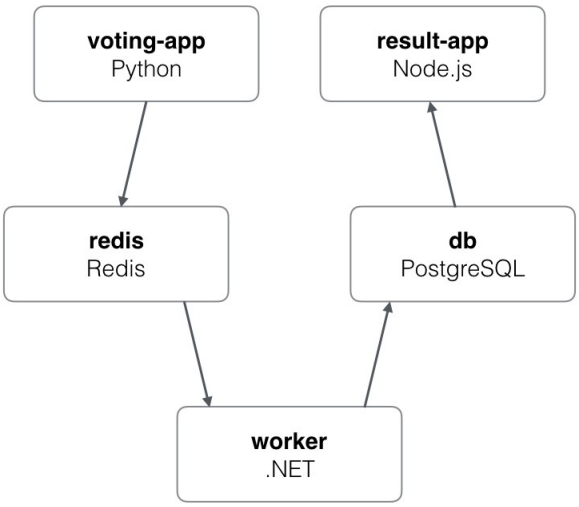
A félév elején a *Kubernetes*hez elengedhetetlen *Docker*rel ismerkedtem meg, ehhez kapcsolódó első feladatom egy *Docker* konténer megvalósítása volt, mely a *cowsay* és *fortune* csomagok használatával megjelenít egy idézetet [4][5]. A feladat megoldása során részletesebben megismertem a *Docker* alapfogalmait, a *Dockerfile* szintaxisát, a Docker képfájl buildelés folyamatát [6]. Következő lépés volt a *Docker* hálózatkezelésének tanulmányozása, ehhez kapcsolódóan egy 2 konténerből álló hálózat megvalósítása, melyek PING és PONG üzeneteket küldözgetnek egymásnak [7]. Előkészületként *build*elni kellett egy *image*t, létrehozni egy hálózatot, amelyhez kapcsolódhatnak a konténerek, illetve egy plusz feladatként egy *volume*-ot is megvalósítani, melyet használhat a 2 konténer a naplóik tárolására egy közös log fájlban [8].

2.2 Kubernetes környezet

A munkám során szükség volt egy Kubernetes klaszterre, valamint Docker konténerek kezelésére. A munkakörnyezet előkészítése első lépéseként *Dockerfil*-ból Docker képfájlt buildeltem [6], valamint a *Docker* hálózatkezelési mechanzimusaira építve több-konténeres rendszerrel validáltam a munkámat [7].

Következő lépésben alokális gyakorláshoz a *microk8s* szoftvercsomaggal egy *Kubernetes* klasztert készítettem a helyi gépemen [10]. A következő lépésben Kubernetes elemeket leíró*YAML* fájlok szerkesztésével egy saját *namespace*ket, *pod*okat, *deployment*eket, *service*ket [9] tartalmazó tesztrendszert hoztam létre. Ebben összehasonlítottam a Docker konténerek és a a Kubernetes *pod*ok közötti hálózati kommunikációt. A legszembetűnőbb különbség az volt, hogy a *Kubernetes* leveszi a felhasználó kezéről a hálózatkezelést, ezért elég volt csak létrehozni 2 *pod*ot és az egyik *pod*ban lévő konténerből egyszerűen el lehetett érni a másik *pod*ban lévő konténert. Ezek után egy témalabor feladat megoldásával validáltam az új rendszerem működőképességét, továbbá ellenőriztem a saját *DockerHub* felhasználói fiókomba telepített *Docker* képfájlok használatát [11].

2.3 Szavazóalkalmazás



2. ábra: A szavazóalkalmazás strukturális felépítése [2].

A szavazó alkalmazás megvalósítása volt a félév végi feladatom. Először a konténerek *pod*okba való elosztását kellett megtervezni, ennél úgy döntöttem, hogy minden konténernek lesz egy saját *pod*ja a 2. ábrán látható módon.

Öt Docker konténerre van szükség az alkalmazás megvalósításához, ebből 2 esetén (a *psql* - db, illetve a *redis*) használható az eredeti *image*, viszont a másik 3 konténer esetén a feladat része, hogy a Docker képfájlokat le*build*eljem, illetve a saját *DockerHub* felhasználóm *repo*-jában eltároljam. Le*build*eltem mindhárom képfájlt, majd az előzőekben ismertetett módon feltöltöttem a publikus *repo*mba, így már bárhonnan könnyedén elérem.

Ezek után létrehoztam egy *namespace*t, amit *vote*-nak neveztem el, majd mindegyik *pod*hoz írtam egy *deployment*et, amikben a *vote*-namespachez rendeltem őket. A deploymenteket azonosító labellel láttam el őket (*vote, redis, worker, result, db*). A *db* és a *redis* esetén konténerportokat adtam meg (a *db* (vagyis *postgres*) az 5432, a *redis* esetén pedig a 6379-es portot), illetve a *db*-nek megadtam environment változókban az adatbázis felhasználónevét, illetve jelszóját. A *worker* kivételével mindegyik *deploymenthez* szükség van egy *servicere* is a portok kezeléséhez, így ezek megírásával folytattam munkámat. A *serviceket* is *yaml* fájlokban definiáltam. A *servicek* esetén a *labelnek* egyeznie kell a *deployment labelével*, a Kubernetes ez alapján párosítja a hozzátartózó *servicel*. A *voting-app*, illetve a *result-app* esetén a 80-as portot kell megnyitni a webszerver elérésére, így ezeket nyitottam meg, illetve típusként *LoadBalancer-t* használtam terheléselosztásra. Az adatbázisok esetén is szükség volt portok nyitására, hogy a a megfelelő másik *podok* tudjanak írni, illetve olvasni belőlük szavazás, illetve a szavazás eredményének megtekintése esetén, ebben az esetben a fentebb említett 5432, illetve a 6379-es portokat kellett engedélyezni.

A *deployment* fájlok szerverre való eljuttatásának megkönnyítése érdekében egy *git repohoz* hozzáadtam a fájlokat, így a szerveren csak le kellett *cloneozni* a *repot* és elérhető is volt mindegyik szükséges *yaml* fájl.

A szerveren először, létrehoztam a namespace-t, majd a servicek, végül pedig a

deploymentek yaml fájljait alkalmaztam. Pár másodperc elteltével fel is állt a rendszer, ez látható a podok lekérése esetén (kubectl get po -n vote parnaccsal), mindegyik pod running státuszban van, ez a 3. ábrán látható.

```
root@master-3:~/work/Training-Project-Laboratory-BMEVITMAL00/Lab06/src/kubernetes/five-pod-solution# kubectl get po -n vote
NAME                      READY   STATUS    RESTARTS   AGE
db-77464d4957-xq7k7       1/1     Running   0          48s
redis-fd7cc6786-79s2n     1/1     Running   0          48s
result-75b8bd76b7-rvwsv   1/1     Running   0          48s
vote-79cc99d9d6-vtz64     1/1     Running   0          48s
worker-9fc48559c-gjdqd    1/1     Running   1          48s
```

3. ábra: podok kilistázása

Kilistáztam a serviceket (*kubectl get svc -n vote* parancs segítségével), hogy megnézzem melyik portot nyitotta a webszervereknek a *Kubernetes*, a 4. ábrán látható, hogy a *vote* a 30562-es portot kapta, a *result* pedig a 30468-at.

```
root@master-3:~/work/Training-Project-Laboratory-BMEVITMAL00/Lab06/src/kubernetes/five-pod-solution# kubectl get service -n vote
NAME     TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
db       ClusterIP      10.110.37.53    <none>        5432/TCP       65s
redis    ClusterIP      10.98.198.128   <none>        6379/TCP       65s
result   LoadBalancer   10.107.182.17   <pending>     80:30468/TCP   65s
vote     LoadBalancer   10.97.79.93     <pending>     80:30562/TCP   65s
```

4. ábra: servicek kilistázása

A Lynx szöveges böngésző alkalmazás segítségével lekértem a *localhost 30562*-es portját, melyet a *vote service* nyitott, majd szavaztam. Ezután a *psql* segédprogram használatával csatlakoztam a *db*-hez, lekértem a *votes* tábla rekordjait és meg is jelent ott egy új rekord (5. ábra), vagyis ez bizonyítja, hogy helyesen működött az alkalmazás.[12]

```
root@master-3:~/work/Training-Project-Laboratory-BMEVITMAL00/Lab06/src/kubernetes/five-pod-solution# lynx localhost:30562
root@master-3:~/work/Training-Project-Laboratory-BMEVITMAL00/Lab06/src/kubernetes/five-pod-solution# psql postgres://postgres:postgres@10.110.37.53:5432/postgres
psql (10.15 (Ubuntu 10.15-0ubuntu0.18.04.1), server 9.4.26)
Type "help" for help.

postgres=# \d
        List of relations
 Schema | Name  | Type  |  Owner
--------+-------+-------+----------
 public | votes | table | postgres
(1 row)

postgres=# select * from votes;
        id        | vote
------------------+------
 3a43b63cc78ce44b | a
(1 row)

postgres=#
```

5. ábra: az adatbázisban a votes tábla rekordjának lekérése

3 Összefoglalás

A feladatom egy Kubernetesbe telepített, több konténerből álló összetett szolgáltatás létrehozása és tesztelése volt. A feladat megvalósításához a szükséges Docker képfájlok létrehozásával, a Kubernetes rendszer konfigurálásval és a szolgáltatás ellenőrzésével is foglalkoztam. Végül pedig mindegyik, általam megvalósított feladatot, beleértve a félév során az előkészítő feladatokat is, szerkesztett formában, a szükséges forrásfájlokkal együtt a *github repom*ban publikáltam [13].

Irodalom, és csatlakozó dokumentumok jegyzéke

A tanulmányozott irodalom jegyzéke

[1] What is Kubernetes? Kubernetes Documentation/Concepts/Overview, ábra és leírás
https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
Utolsó letöltés ideje: 2020-12-01

[2] Kubernetes Fundamentals: Voting App, Voting App architecture ábra,
https://kubernetes-bootcamp.wikitops.io/exercise-app/voting-app
Utolsó letöltés ideje: 2020-12-01

[3] Kubernetes Components
https://kubernetes.io/docs/concepts/overview/components/
Utolsó letöltés ideje: 2020-12-01

[4] Cowsay
https://en.wikipedia.org/wiki/Cowsay
Utolsó letöltés ideje: 2020-12-02

[5] Fortune
https://en.wikipedia.org/wiki/Fortune_(Unix)
Utolsó letöltés ideje: 2020-12-02

[6] Dockerfile
https://docs.docker.com/engine/reference/builder/
Utolsó letöltés ideje: 2020-12-02

[7] Docker network
https://docs.docker.com/network/
Utolsó letöltés ideje: 2020-12-02

[8] Docker volume
https://docs.docker.com/storage/volumes/
Utolsó letöltés ideje: 2020-12-02

[9] YAML
https://en.wikipedia.org/wiki/YAML
Utolsó letöltés ideje: 2020-12-02

[10] MicroK8s
https://microk8s.io/docs
Utolsó letöltés ideje: 2020-12-02

[11] DockerHub
https://hub.docker.com/
Utolsó letöltés ideje: 2020-12-02

[12] PSQL
https://www.postgresql.org/docs/9.3/app-psql.html
Utolsó letöltés ideje: 2020-12-02

Egyéb hivatkozás - Saját munka nyilvános dokumentációja

[13] Training-Project-Laboratory-BMEVITMAL00 tárhely - Bence Háromi
https://github.com/benceharomi/Training-Project-Laboratory-BMEVITMAL00
Utolsó letöltés ideje: 2020-12-02