Témalaboratórium beszámoló

Távközlési és Médiainformatikai Tanszék

Készítette:	**Tóth Gábor**
Neptun-kód:	**F041OM**
Ágazat:	**Infokommunikáció**
E-mail cím:	**tothgabor2003@gmail.com**
Konzulens(ek):	**Dr. Maliosz Markosz**
E-mail címe(ik):	**maliosz.markosz@vik.bme.hu**

Téma címe: Horizontális autoskálázás Kubernetes klaszterben

Feladat

Egy Kubernetes klaszterben futó web alapú egyszerű alkalmazás terheléstesztelése és az így felhasznált erőforrások mérése. A mérések alapján az alkalmazás automatikus horizontális skálázása, és fölöslegesen foglalt erőforrások felszabadítása. Továbbá a gyűjtött metrikák megjelenítése egy másik, weben keresztül elérhető felületen.

2023/2024. 1. félév

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

1.1 Bevezető

Sok egyéb közt egy webalkalmazás álltalában nem használ sok erőforrást, mégis hagyományos környezetben azoknak az erőforrásoknak folyamatosan rendelkezésre kell állniuk, ez rossz kihasználtsághoz vezet. Erre ad egy megoldást a konténerizáció, majd ezt továbbfejleszti a konténer orkesztráció.

Ennek egy megvalósítása a Kubernetes [1], amellyel lehetőségünk van különböző úgynevezett mikroszolgáltatásokat [2] indítani és ezeket menedzselni. Egyik nagy előnye a Kubernetesnek (és bármilyen konténer orkesztrációnak), nem csak az, hogy az esetleg sérült, így megszűnt szolgáltatásokat automatikusan újraindítja, és a végső szolgáltatásban kiesés alig tapasztalható, hanem hogy képes érzékelni, ha egy alkalmazás túrterhelt és tud új erőforrásokat foglalni. Illetve, ha már nincs szükség ezekre az erőforrásokra, felszabadítja azoket, ezzel elkerülhető, hogy olyan szolgáltatás foglaljon sok erőforrást, ami éppen nincs terhelés alatt.

1.2 Elméleti összefoglaló

A Kubernetes autoskálázás egy hatékony és dinamikus módszer a konténer alapú alkalmazások menedzselésére, amely lehetővé teszi az erőforrások automatikus skálázását az alkalmazás igényeihez igazítva. A fejlődés útján számos módszerrel próbálták optimalizálni az alkalmazások futtatását. Az első lépés a virtuális gépek használata volt a kisebb erőforrásigényű alkalmazások telepítésére. Azonban ezek a megoldások körülményesek és költségesek voltak, mivel minden alkalmazásnak külön virtuális számítógépre volt szüksége.

A „bare metal” módszerrel való telepítés egy másik megközelítés volt, ahol az alkalmazások közvetlenül a fizikai szervereken futottak. Ez a módszer optimalizált erőforrásokat biztosított, de rugalmatlanná tette az alkalmazások kezelését és skálázását. Ahogy az alkalmazások bonyolultabbá váltak és a felhasználói igények ingadoztak, szükségessé vált egy olyan megoldás, amely dinamikusan alkalmazkodik az igényekhez.

A Kubernetes autoskálázás egy intelligens válasz erre a kihívásra. Ez a rendszer lehetővé teszi az alkalmazások számára, hogy dinamikusan növeljék vagy csökkentsék az erőforrásokat a pillanatnyi terhelésüknek megfelelően. Az autoskálázás során a Kubernetes figyeli az alkalmazások teljesítményét és automatikusan alkalmazkodik a változó igényekhez.

Ez a rugalmasság és skálázhatóság előnyösen megkülönbözteti a Kubernetes-t a korábbi megközelítésektől. A konténerizáció és az autoskálázás kombinációja lehetővé teszi az alkalmazások gyors, hatékony és költséghatékony kezelését a dinamikus informatikai környezetekben. A Kubernetes autoskálázás egy olyan paradigmaváltást jelent, amely lehetővé teszi a vállalatok számára, hogy optimalizálják erőforrásaikat, növeljék az alkalmazásaik rendelkezésre állását, és egyszerűbben kezeljék az egyre összetettebb mikroszolgáltatásokat.

1.3 A munka állapota, készültségi foka a félév elején

A témával ebben a félévben kezdtem foglalkozni az egyetemen. Korábban csak kisebb részével találkoztam, leginkább szabadidőmben. Az autoskálázásig nem jutottam, mert nem volt rá szükségem, csak otthoni szerverhez Docker [3] konténerizációt használtam.

2. Az elvégzett munka és eredmények ismertetése

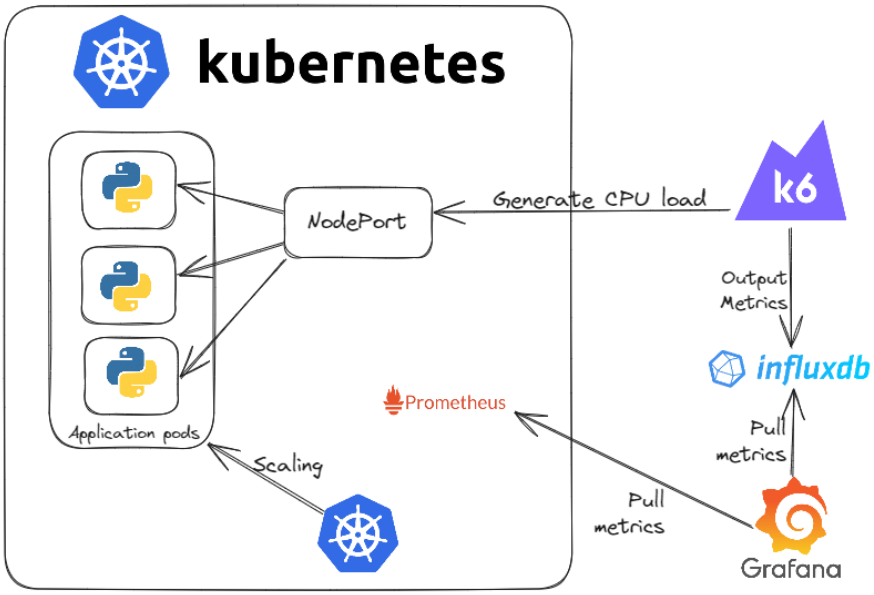
2.1 Feladatleírás és előkészületek

Munkám során egy autoskálázott alkalmazás felépítése lesz a cél Kubernetes segítségével. A teljes program feladata az, hogy nagy mértékű terhelés alatt is az alkalmazás reszponzív maradjon. Az applikációból több példány fut egyszerre, ezek egy terheléselosztón keresztül érhetők el a külvilágból. Terhelésolsztással biztosítható, hogy egyik sem lesz túlterhelt, míg egy másik példány „unatkozik”, illetve, hogy a külvilág, azaz az internet felől csak egy elérési útvonal létezik. A Kubernetes beépített funkcionalitása, hogy méri az egyes konténerek teljesítményét, és ez alapján tud döntést hozni, ezt használjuk majd autoskálázásra. A feladat további része a teljesítményi adatok gyűjtése, nyomon követése és kijelzése grafikonon.

A feladat megoldásához a félév során több Kubernetes klaszter megvalósítást használtam, például minkube [4] vagy micork8s [5]. Végül a k3s-ben [6] valósítottam meg, ez volt a legkézenfekvőbb és a legstabilabb működést biztosította.

A véső megvalósítása az 1. ábrán látható. A klaszterben fut egy webapplikáció, amely minden egyes http kérésre CPU terhelést generál egy egyszerű python programmal, ebből kezdetben 3 található, de az autoskálázás miatt automatikusan csökkent egyre. Továbbá fut a klaszterben egy NodePort típusú szolgáltatás, ez biztosítja a webapplikáció internetes, külső elérését és a terheléselosztást. Minden webapplikációt futtató pod-nak engedélyezve van a 5050-es portja, a NodePort pedig a külső IP cím 32000-es portjáról továbbítja a HTTP kéréseket az egyik podnak.

A Kubernetesben elérhető továbbá egy HPA (Horizontal Pod Autoscaler, horizontális pod autoskálázó) [7] nevű erőforrás. Egy HPA erőforrást megfelelően felkonfigurálva és indítva a Kubernetes figyeli a pod-ok kihasználtságát és darabszámát és annak megfelelően indít újakat vagy esetleg állít le fölöslegesen futó pod-ot. A klaszteren belüli utolsó program egy Prometheus [8] példány, ami az adatok gyűjtésért felelős. Egy nyílt forrású „metric scraper”, azaz metrika gyűjtő, amit nagyvállalati környezetben is használnak. Az adatokat pod-onként és konténerenként gyűjti, így később jól lehet vizualizálni.



1. ábra. A klaszter és környezetének struktúrális felépítése

Az eddigi alkalmazások lényege, hogy a Kubernetes kezelje őket, hiszen akkor tudja autoskálázni az alkalmazást, illetve az adatok gyűjtése is a legegyszerűbb a klaszteren belülről.

A klaszteren kívül van telepítve még további három program. Ezek ugyan azon a virtuális gépen vannak, mint a Node azaz, a Kubernetes végpontja, de nem a Kubernetes kezeli őket.

A terhelés generálására a k6 [9] nevű programot használom. Ez egy egyszerű és könnyen kezelhető program, de ettől függetlenül jól paraméterezhető, a jelenlegi feladatra alkalmas. Egy előre megadott folyamatot futtat végig, amelyben megadhatunk „fázisokat”, minden fázist egy hossz és egy szám, hogy mennyi kérés érkezzen. A program majd a megadott cél felé HTTP kéréseket futtat, ezzel terhelve azt. A k6-ot érdemtelen lett volna a klaszterbe telepíteni, hiszen a kívülről érkező kéréseket akarjuk szimulálni, azt kívülről a leglogikusabb.

Az terhelési adatok időrendben történő tárolására InfluxDB-t [10] használtam, amely egy idősoros adatbázis, azaz időbélyeggel tárolja az adatokat, így grafikonon könnyű megjeleníteni. Mivel ez egy adatbázis, ezt körülményesebb lett volna a Kubernetesbe telepíteni és a kommunikáció a k6-tal is bonyolultabb lett volna, ezért ezt nem oda telepítettem, a feladatot ez nem befolyásolja. Végül az adatok igényes ábrázolására a Grafana [11] nevű szoftvert használom, ami szintén egy jól működő és széles körben, cégek által is használt adatábrázoló szoftver. A Program a Prometheustól és az InfluxDB-től gyűjti az adatokat és jeleníti meg valós időben, így könnyű figyelni a terhelésre történő változást. A Grafanát szintén a klaszteren kívülre telepítettem az előzőhöz hasonló indokok miatt, bár elegánsabb lett volna azt is Kubernetesbe telepíteni.

2.2 Megvalósítás

A megvalósításban nagyon nagy segítség volt számomra Michael Wanyoike [12] cikke, melyben leír egy alkalmazás autoskálázását, kicsit más módszerrel, de az alap architektúra hasonló. A klaszteren kívüli részek szinte teljesen megegyeznek, a k6 konfigurációját és az InfluxDB-t a cikk alapján állítottam össze. A Grafana dashboard-ban a megjelenítést megváltoztattam arra, amire nekem szükségem volt. Hozzáadtam például egy CPU kihasználtság figyelő panelt, ez hasznos, mivel a webapp pod-oknál ezt a metrikát használom az autoskálázásra.

A klaszteren belül a webapplikációt egy saját készítésű python alapú konténer képfájlal oldottam meg. A konténer egy egyszerű python webappot futtat a python „http.server” könyvtárával [13]. Minden kérés előtt egy ciklusban gyököt számol úgy, hogy az értéket ne tudja cache-elni, hogy a CPU terhelés valóban megjelenjen, majd visszaküld egy „Hello Word” üzenetet HTML-ben, ezzel szimulálva egy számításigényes feladatot egy webserveren. A klaszterben ehhez kettő erőforrást kell készíteni, egy Deployment típusú erőforrást és egy Service (szolgáltatás) típusú erőforrást. A Deployment felelős a pod-okért, ebből kezdetben hármat állítottam be, hogy mindig tartson életben 3 ugyanolyan podot, ezt a „replicas” kulcsszóval lehet megtenni, de ezt a HPA majd úgyis leviszi kihasználatlanság miatt. A Deployment tulajdonságainál fontos továbbá a label, ez alapján lesz azonosítható a NodePort és a HPA számára. Illetve fontos még a saját image, amit Docker Hub-ra [14] feltöltöttem, innen éri el a klaszter nyilvánosan.

A másik erőforrás a NodePort, ez egy szolgáltatás típusú erőforrás, ami azt jelenti, hogy elérhető a külvilág számára. Egyfajta terheléselosztásként is működik, bár arra van a Kubernetesnek külön megoldása (LoadBalancer), de a k3s miatt a NodePort célravezetőbb megoldás volt. A NodePort felelős azért, hogy a Deployment pod-jai elérhetők legyenek az interneten, ezt az előbb említett label-ek segítségével lehetséges. A NodePort a bejövő forgalmat az olyan konténerek felé fogja irányítani (már a klaszteren belül), amelyek megfelelnek a konfigurációjában leírt label-eknek, esetemben ezek a python webserverek lesznek.

Az applikáció már elérhető kívülről, és terhelni is tudjuk, de terhelés hatására a teljesítmény és a válaszidő romlani fog, viszont semmi nem fog ellene semmit sem tenni. Erre használom a téma célkitűzéseként a HPA-t. HPA-hoz léteznek külső megoldások, mint például a KEDA [15],

de a jelenlegi céljaimnak megfelel a Kubernetes beépített megoldása. A HPA működéséhez szükséges megadni, hogy mit skálázzon, esetemben ez a Deployment lesz, meg kell adni, a nevét, és típusát, illetve meg kell adni, hogy milyen metrikát figyeljen, aminek a hatására új pod-ot kell indítani, ez nálam a „targetCPUUtilizationPercentage: 50”, ez azt jelenti, hogy amennyiben a CPU kihasználtság meghaladja az 50%-ot, új pod-ot indít. Megadtam még továbbá a „minReplicas”, és a „maxReplicas” értéket, egynek és tíznek, ezzel azt lehet szabályozni, hogy egy példány mindig fusson, de tíznél több sose [16].

Utolsó lépésként a Prometheus-t kellett felkonfigurálni. Ehhez egy interneten talált cikket vettem alapul [17]. Kisebb módosításokat kellett végezni, például portszámot illetően, de nagyrészt megegyezik a konfiguráció. Eltérés még továbbá, hogy a példában a webapplikációnak van egy „/metrics” oldala, és innen plusz adatokat szerez a Prometheus, nekem ez nincs, de a különbséget a program jól kezeli, így ezek a metrikák nélkül fejeztem be a feladatot.

2.3 Az alkalmazás terheléstesztelése

A terhelés tesztelésére a k6 nevű szoftver használom, ezzel jól lehet terhelést generálni. Egy egyszerű JavaScript programmal lehet megadni a konfigurációt, ezt az első cikkben leírtak alapján készítettem el.

2. ábra. A terhelésre kiváltott reakció Grafana dashboard-on

Az ábrán jól látható, hogy a terhelés mikor kezdődik, ekkor még csak egy pod fut, a HPA később indítja a többi pod-ot. A bal felső grafikon mutatja, hogy egyszerre hány pod futott, az alsó pedig, hogy a pod-ok CPU kihasználtságát. A jobb felső mutató az aktuális futó pod-ok számát mutatja, ez jelenleg nulla, mert a HPA visszavette a terhelés megszűnése után.

2.4 Összefoglalás

A félév során egy olyan témában fejlődhettem tovább, ami érdekel, és eddig csak limitáltan foglalkoztam vele. Egy egyszerű webalkalmazás autoskálázását mutattam be Kubernetes környezetben. A téma fontos bármilyen olyan környezetekben, ahol a terhelés nem egyenletesen van elosztva és hullámszerűen érkeznek kérések, de fölöslegesen nem akarunk sok erőforrást használni amikor éppen hullámvölgy van a kérésekben. A megoldásban olyan szoftvereket használtam, amelyek nagyvállalati környezetben is elismertek, mint például: Prometheus, Grafana, InfluxDB és természetesen az egész rendszert összefogó és az autoskálázást végző Kubernetes. A dolgozat végére sikerült egy jól működő autoskálázó rendszert felállítani, természetesen a rendszert pontosítani és javítani lehet, például a CPU terhelés mellett más is lassíthatja a válaszidőt. A terhelést több eszközzel is lehet szimulálni, ami memóriaművelet, IO műveleteket vagy másik szerverhez fordulás miatt lassítja a válaszidőt.

3. Irodalom, és csatlakozó dokumentumok jegyzéke

A tanulmányozott irodalom jegyzéke:

[1] *Kubernetes,* https://kubernetes.io/docs/home/

[2] *Mikroszolgáltatások,* https://aws.amazon.com/microservices/

[3] *Docker,* https://docs.docker.com/

[4] *minikube,* https://minikube.sigs.k8s.io/docs/start/

[5] *MicroK8s,* https://microk8s.io/docs

[6] *k3s,* https://docs.k3s.io/

[7] *Horizontal Pod Autoscaler,* https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

[8] *Prometheus,* https://prometheus.io/docs/introduction/overview/

[9] *k6,* https://k6.io/docs/

[10] *InfluxDB,* https://www.influxdata.com/

[11] *Grafana,* https://grafana.com/docs/

[12] *Michael Wanyoike (2020), Testing the behavior of autoscaling kubernetes pods with Keda and k6 Utolsó letöltés időpontja: 2023. 12. 06.* https://k6.io/blog/kubernetes-keda-autoscaling/

[13] *Python http.server könyvtár,* https://docs.python.org/3/library/http.server.html

[14] *Docker Hub,* https://hub.docker.com/

[15] *Keda,* https://keda.sh/docs/

[16] *Priya Pedamkar (2023), Kubernetes Autoscaling Utolsó letöltés időpontja: 2023. 12. 06.* https://www.educba.com/kubernetes-autoscaling/

[17] *Bibin Wilson (2023), How to Setup Prometheus Monitoring On Kubernetes Cluster Utolsó letöltés időpontja: 2023. 12. 06.* https://devopscube.com/setup-prometheus-monitoring-on-kubernetes/

Csatlakozó egyéb elkészült fájlok jegyzéke:

A munka során készített manifest és egyéb fájlok megtalálhatók a https://github.com/Mooky0/temalab repository-ban. A repository-n belül a a “forras_1” mappa arra utal, hogy az [1]-es forrásból készült az a manifest fájl.