
# Önálló laboratórium beszámoló

Távközlési és Médiainformatikai Tanszék

készítette: **Reményi Gergely Márk**
**gergo@gergo.city**
neptun-kód: **KPZH44**
ágazat: **Mérnökinformatikus szak**
konzulens: **Maliosz Markosz**
**maliosz@tmit.bme.hu**
konzulens: **Simon Csaba**
**simon@tmit.bme.hu**

**Téma címe: Skálázás Kubernetesben egyedi metrikák alapján**

**Feladat:**
A hallgató feladata egyedi metrikák alapján történő horizontális skálázás vizsgálata Kubernetes környezetben. A hallgató a félév során megismerkedik a Horizontal Pod Autoscaler (HPA) technológiával, kiépíti és konfigurálja Kubernetesben az egyedi metrikákat létrehozó csővezetéket.

**Tanév:** 2020/2021. tanév, II. félév

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

## 1.1. Bevezető

A skálázás az egyik legkomplexebb és legfontosabb feladat (TODO:forrás) a számítástechnikában. A beszámolóban be fogom mutatni a Kubernetes konténer alapú alkalmazáskezelő rendszert és azokat a mechanizmusokat, amelyek horizontális skálázást lehetővé teszik erőforrás (CPU, memória) és egyedi metrikák alapján.

Az 1.2.1 fejezetben röviden bemutatom a Kubernetest, csak azokat az elemeit kiemelve, amelyek a skálázás bemutatásához szükségesek. Az 1.2.2 fejezetben részletesebben kifejtem a Kubernetesnek azt a komponensét, amely a lehtővé teszi a horizontális skálázást. Az 1.2.3 fejezetben bemutatom annak a csővezetéknek az elemeit, amelyekkel megoldottam azt, hogy a Kubernetes egyedi metrikák alapján tudjon skálázódni. Az 1.2.4 fejezetben bemutatom azt a terhelésgeneráló eszközt, amivel a skálázódást teszteltem. Végül a 2 fejezetben bemutatok egy CPU és egy egyedi metrikák alapján történő skálázást.

## 1.2. Elméleti összefoglaló

### 1.2.1. Kubernetes

A Kubernetes egy Google által fejlesztett konténer orchesztrációs rendszer [1], amely programok telepítését, skálázását, terheléselosztását és helyreállítását automatizálja. A Kubernetes konténer szinten operál, amelyek virtuális gépekhez hasonlítanak, azonban közös operációs rendszert használnak. A konténereknek külön fájlrendszerük, CPU, memória és processz névterük van, ezzel leválaszthatók a hoszt operációs rendszerről, amivel szabadon mozgathatóak lesznek operációs rendszerek között. A Kubernetes elosztott rendszer, a konténereket klaszterben futtatja. Minden klaszterben kiválasztásra kerül egy (vagy több) master node, amely egy olyan számítógép, amely az egész klaszter állapotáért felel. A klaszterben lévő többi számítógép worker node, ezeken futnak a konténerek. Az alábbiakban bemutatom a Kubernetesnek azon komponenseit, amelyek szükségesek a konténerek automatikus horizontális skálázásának megvalósításához.

**Control plane** A control plane [2], magyarul a vezérlési sík komponensei olyan processzekből állnak, amelyek a Kubernetes klaszter master node-ján futnak, és a klaszter állapotáért felelnek. Például elindítanak egy konténert, ha a klaszterben futó konténerek száma nem egyezik a konfigurációban megadott konténerek számával. A vezérlési sík az alábbi processzekből áll:

**kube-apiserver** Elérhetővé teszi a Kubernetes API-t.

**etcd** A klaszterről tárol információkat egy kulcs-érték adatbázisban.

**kube-scheduler** Eldönti, hogy az újonnan létrehozott konténerek melyik node-ra legyenek ütemezve.

**kube-controller-manager** Azért felel, hogy a konfigurációs fájlokban megadott állapot konzisztens legyen a klaszter állapotával. Például konténereket hoz létre vagy töröl.

**Node** A node-okon [2] futnak a klaszterbe telepített konténerek. Ehhez szükséges, hogy minden node-on legyen a konténer futtatását lehetővé tevő runtime (pl. Docker vagy containerd), valamint az alábbi processzek:

**kubelet** A node-on futó konténerek állapotáért felel. A control plane utasításait követve biztosítja, hogy megfelelő számú egészséges konténer fusson a node-on.

**Kube-proxy** Olyan hálózati beállításokért felel a node-on, amelyek a klaszterből kimenő, klaszterbe bemenő és a klaszteren belüli kommunikációt teszik lehetővé.

**Podok, replicasetek és deploymentek** A podok [3] Kubernetesben a legkisebb telepíthető számítási egység. Egy podban egy vagy több, szorosan összekapcsolódó konténer lehet.

Minden podnak egyedi IP címe van, ami lehetővé teszi a podok közötti kommunikációt. A podokban lévő konténerek közvetlenül nem érhetnek el más konténereket, a kommunikáció mindig a podokon keresztül történik. Egy podon belül lévő konténerek viszont tudnak kommunkálni egymással a helyi hálózaton.

Egy replicaset [4] objektummal lehet megadni a pod klaszterben futú másolatainak számát. Az 1.2.2 fejezetben bemutatott Horizontal Pod Autoscaler ezt az objektumot változtatja meg, amikor skálázódást hajt végre.

Kubernetesben nem szokás külön pod és replicaset objektumokat létrehozni, helyettük egy deployment [5] objektumban érdemes definiálni a telepítés kívánt állapotát, vagyis azt, hogy milyen konténerek kerüljenek a podokba, és a podok másolatának a számát. A deploymentek segítségével áramvonalassá válik a konténerek telepítése, frissítése, skálázása, leállítása és visszagörgetése korábbi verziókra.

**Service** Kubernetesben service [6] objektumok segítségével lehet podokak elérni a klaszteren belül és kívülről. A service-k podokhoz rendelt címkék (label [7]) alapján választják ki a hozzájuk tartozó podokat, amelyek között terheléseloszlást végeznek. Service-ekre azért van szükség, mert a podokhoz rendelt ip-címek nem biztosítanak megfelelő rugalmasságot.

A service-eknek több típusa van:

- ClusterIP: alapértelmezett service, csak a klaszteren belül érhető el.
- NodePort: minden node-on nyit egy portot, amin keresztül elérthetőek a podok.
- LoadBalancer: klaszteren kívüli (tipikusan felhő) terheléselosztóján teszi elérhetővé a service-t.

### 1.2.2. Horizontal Pod Autoscaler

A horizontal pod autoscaler [8] (HPA) objektum segítségével lehet a podok számát skálázni Kubernetesben. A HPA alapértelmezetten tud cpu és memória metrikák alapján skálázódni, egyedi metrikák alapján történő skálázás körülményesebb, ezt a 2.3 fejezetben mutatom be.

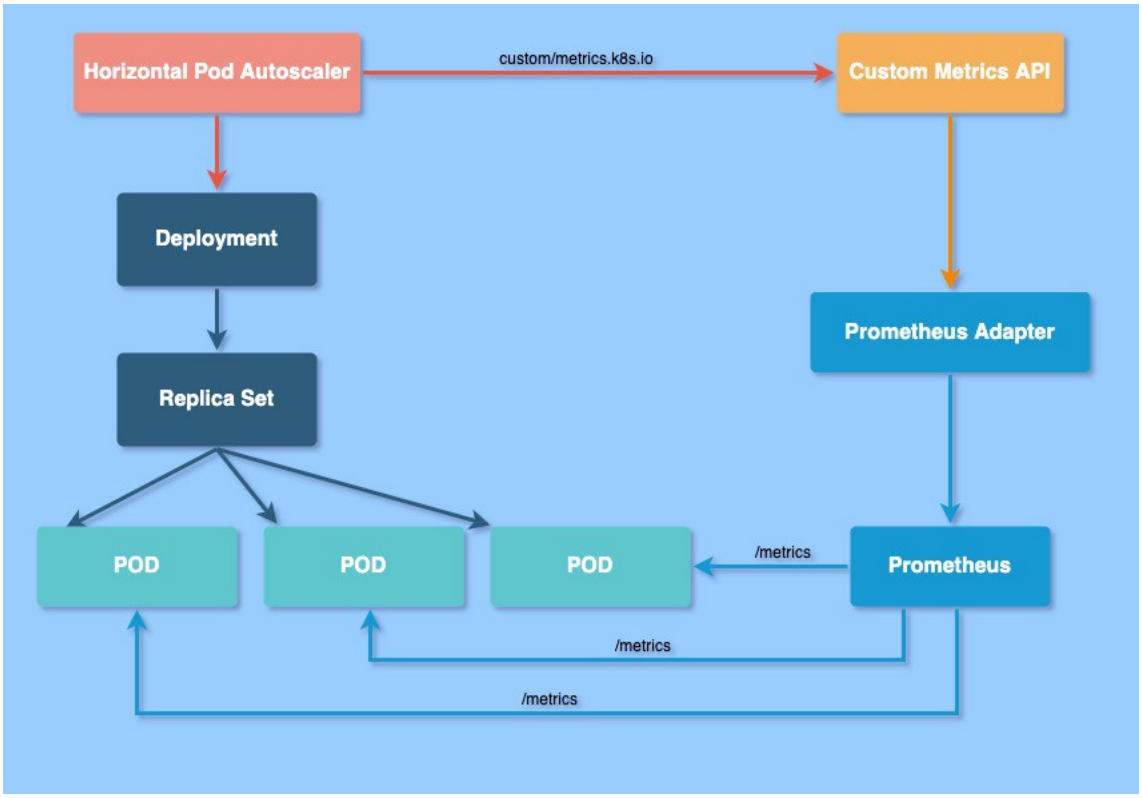
A HPA az alábbi képlettel számolja ki a kívánt replikák számát:

$$kivantreplikak = \lceil jelenlegireplikak * \frac{jelenlegimetrikaerteke}{kivantmetrikaerteke} \rceil \tag{1}$$

### 1.2.3. Skálázás egyedi metrikák alapján

A Google nem gondoz az egyedi metrikák alapján történő skálázásról kiterjedt dokumentációt, azonban újságcikkek alapján [9] [10] fel lehet építeni egy olyan csővezetéket, amely begyűjti a podok által generált metrikákat, és elérhetővé teszi a Kubernetes custom metrics API-ja számára.

A legkézenfekvőbb megoldás Prometheus-t [11] használni a metrikák begyűjtéséhez, és Prometheus Adaptert [**?**] használni a begyűjtött adatok átalakítására a custom metrics API számára. Az 1 ábrán látható az a csővezeték, amelyen keresztül az egyedi metrikus adatoknak át kell menniük, hogy a HPA használni tudja őket.




1. ábra. Egyedi metrikák csővezetéke

A Prometheus nyílt forráskódú monitorozó és riasztó rendszer. Az idősoros adatokat a */metrics* végpontról gyűjti be a pull modell szerint az alkalmazásoktól, amiket PromQL [13] lekérdezőnyelvvel lehet lekérni.

A Prometheus telepítése Kubernetesbe a legegyszerűbben egy operátor [14] telepítésével végezhető. Kubernetesben az operátorok olyan alkalmazáscsomagok, amelyek automatizálják egy adott alkalmazás karbantartását. Mivel egy operátor sok komponensből áll, ezért célszerű Helmmel [15] telepíteni, ami egy csomagkezelő a Kuberneteshez.

A Promehteus pull modell szerint a */metrics* végpontokról gyűjti be a metrikus adatokat. A végpontok felfedezéséhez a Prometheus operátor *ServiceMonitor* nevű custom resource-t [16] definiál. Ebben meg kell adni a monitorozni kívánt podok címkéjét és portját, ahol a */metrics* végpont elérhető.

A Prometheus operátor alapértelmezetten tartalmazza a Grafana [17] nevű vizualizációs szoftvert, amelynek segítségével PromQL nyelven lekérdezhetőek és vizualizálhatóak a Prometheus által gyűjtött metrikák. A 2 fejezetben szereplő ábrákat Grafanában hoztam létre.

A Prometheus által gyűjtött metrikákat a Prometheus Adapter alakítja át a Kubernetes custom metrics API-ja számára. Az adapter konfigurációja [18] a következőképpen történik:

- Felfedezés: egy lekérdezés sablonját kell megadni, ami összegyűjti, hogy milyen objektumokon lehet a lekérdezést elvégezni
- Hozzárendelés: a felfedezésben begyűjtött objektumokat kell hozzárendelni Kubernetes erőforrásokhoz
- Megnevezés: meg kell adni a custom metrics API számára a metrika nevét
- Lekérdezés: meg kell adni, hogy maga a lekérdezés hogyan történjen

### 1.2.4. Terhelésgenerálás K6-al

A mérések elvégzéséhez k6 [19] nevű terhelésgeneráló alkalmazást választottam. Számomra az tette vonzóvá a k6-ot, hogy egy aktív közösség és vállalat áll mögötte.

A k6-nak is van Kubernetes operátor csomagja [20], Bár még kísérleti fázisban van, ennek ellenére (egy-két kisebb hibán kívül) jól használható elosztott terhelési tesztek futtatására.

Az operátor jelenlegi állapotában a tesztet szigorúan egy *test.js*-nek elnevezett fájlban kell definiálni, ezután configmapként telepíteni kell a klaszterbe. A tesztet egy k6 típusú custom resource létrehozásával lehet elindítani. Ebben meg kell adni a configmap nevét, azt, hogy mennyi pod futtassa a teszteket és hogy futhassanak-e a podok külön node-okon. A podok létrehozása után még nem indul el a teszt, az összes k6 által létrehozott podon el kell indítani a teszteket a `k6 resume` paranccsal.

# 2. Mérések elvégzése

## 2.1. Mérésekhez beállított alapértelmezett értékek

A minden mérés elvégzéséhez egyetlen k6 tesztet definiáltam. Ebben a tesztben a kapcsolatot kialakító virtuális felhasználók száma 10 perc alatt lineárisan növekedik 0-tól 100-ig, majd 30 másodperc alatt visszaesik 0-ra. Minden virtuális felhasználó egytized másodpercenként küld el egy HTTP GET kérést a skálázandó podoknak.

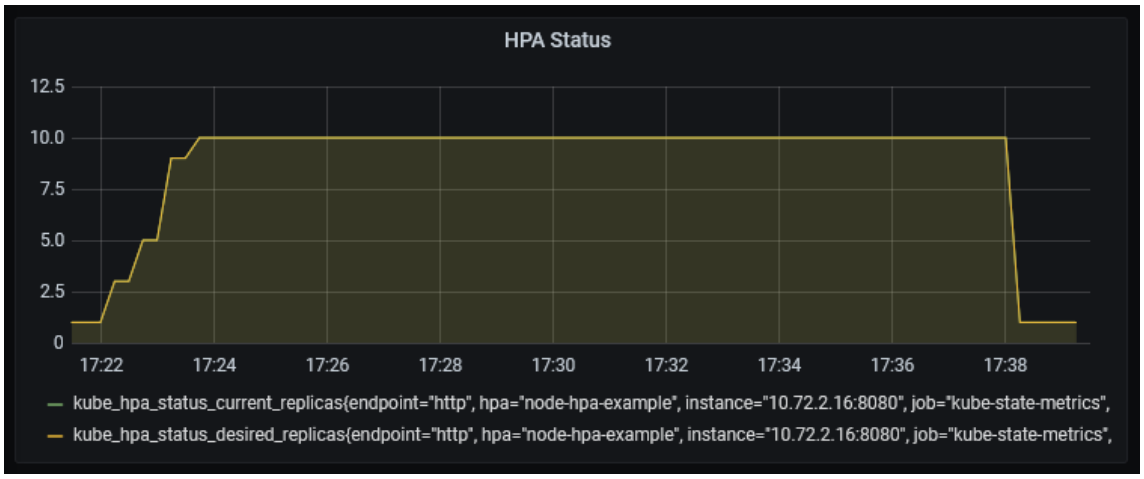
A podokat úgy állítottam be, hogy 300 millicore legyen a limitjük, vagyis a node CPU-jának maximum $3/10$-ad számítási kapacitását használhatják. A podoknak még megadtam azt is, hogy csak olyan node-okon indulhatnak el, amelyeken van még 100 millicore-nyi szabad kapacitás.

## 2.2. CPU alapú skálázás rövid válaszidejű végponton

A CPU alapú skálázáshoz úgy állítottam be a HPA-t, hogy átlagosan 50% alatt maradjon a podok CPU igénye. Ez az 50%-os küszöb a podok CPU limitjéhez viszonyítva értelmezendő, vagyis ha egy pod korlátja 300 millicore, akkor a HPA feladata lesz annyi új podot indítani, hogy a podok átlagos CPU erőforrás igénye 150 millicore alatt maradjon.

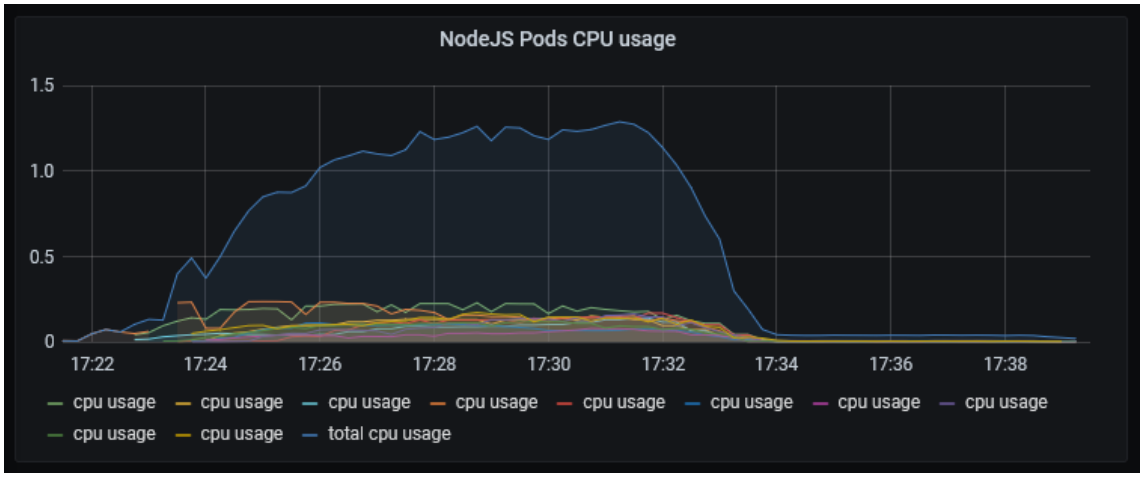
A teszt lefutásához négy diagramot hoztam létre Grafanában, amelyek szükségesek a CPU alapú skálázás vizsgálatához és értékeléséhez, és amellyel össze lehet vetni az egyedi metrikák alapján skálázódó teszt lefutásának adatait.

A 2 ábrán láthatóak a HPA által kívánt podok száma (desired replicas) és a podok tényleges száma (current replicas). Az ábrán csak egy adatsor látszik, mivel a két metrika együtt mozog. Az ábrán látható az is, hogy a HPA nem skálázza le a podokat egyből, hanem kivárja az alapértelmezetten beállított 5 perces lehűlési időt.



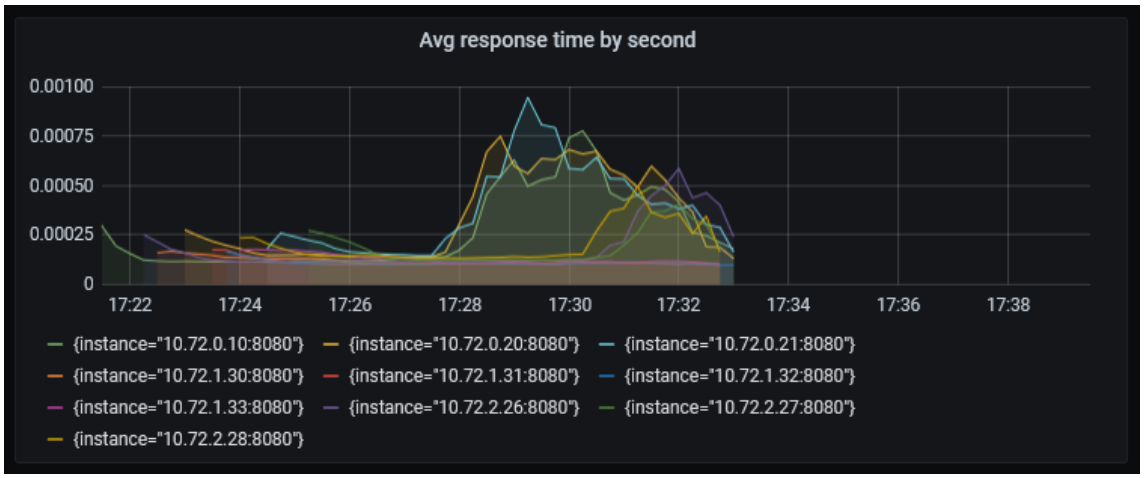

2. ábra. HPA által kívánt és tényleges podok száma CPU alapú skálázás esetén

A 3 ábrán látható a skálázott podok CPU kihasználtsága. A kék színű grafikon jelöli az összes pod CPU kihasználtságát, a többi ugyanezt podokra levetítve. Az ábráról leolvasható, hogy a teljes CPU kihasználtság maximuma 1250m. Ezt leosztva a replikák számával, vagyis 10-el, 125m-et kapunk, azaz a HPA jól működött, mert az átlagos CPU kihasználtság nem ment 150m fölé. Azonban egyenként megvizsgálva a podok CPU használatát, vannak olyan podok, amelyek elérik, és sokáig a 300m-es tartományban maradnak, és olyanok, amelyek a 100m-es tartományban maradnak. Ebből látható, hogy a Kubernetesben a terheléselosztás nem egyenletesen történik a podok között.



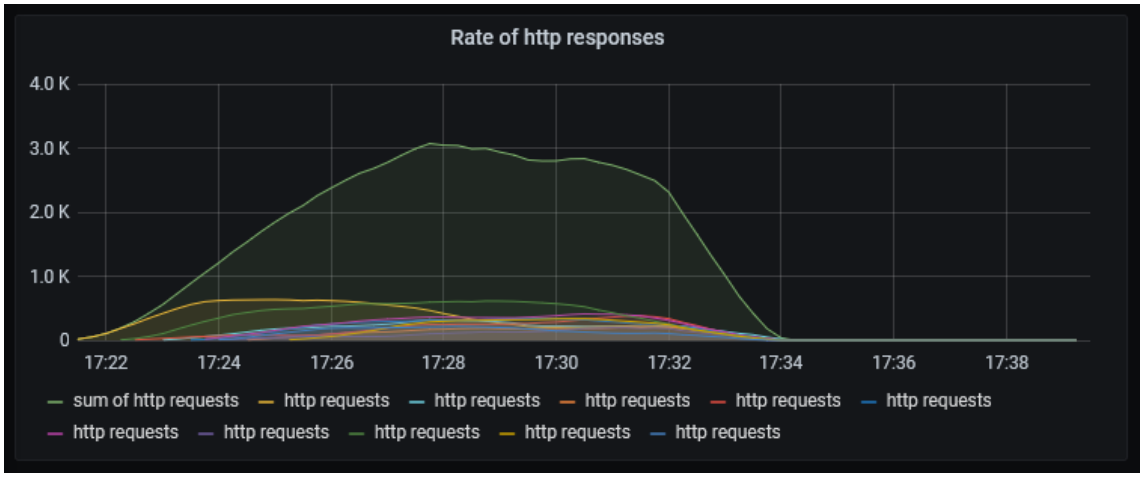

3. ábra. Kiszolgáló podok CPU kihaszáltsága

A 4 ábrán az átlagos válaszidő látható podokra bontva. Kezdetben a podok jelentős része magasabb válaszidővel válaszol, ez lehet a NodeJS Express valamilyen sajátossága miatt. A válaszidők a teszt közepe felé egyforma értékeket vesznek fel, majd kettő pod kivételével az összes pod válaszideje megnő. Ennek az lehet az oka, hogy a terhelésgenerálás olyan fázisba ért, ahol a kéréseket a podok nem tudják azonnal kiszolgálni.




4. ábra. Kiszolgáló podok válaszideje 1 másodpercre vetítve

Az 5 ábrán az átlagos válaszok száma látható. A zöld színű grafikon jelöli az összes pod átlagos válaszainak számát. Ez az ábra a teszt szerint elvárt módon növekszik a 17:28-as időpontig, majd a válaszok arányában stagnáció áll be. A 4 ábrával összevetve arra lehet következtetni, hogy a stagnáció a podokra nehezedő terhelés következménye.



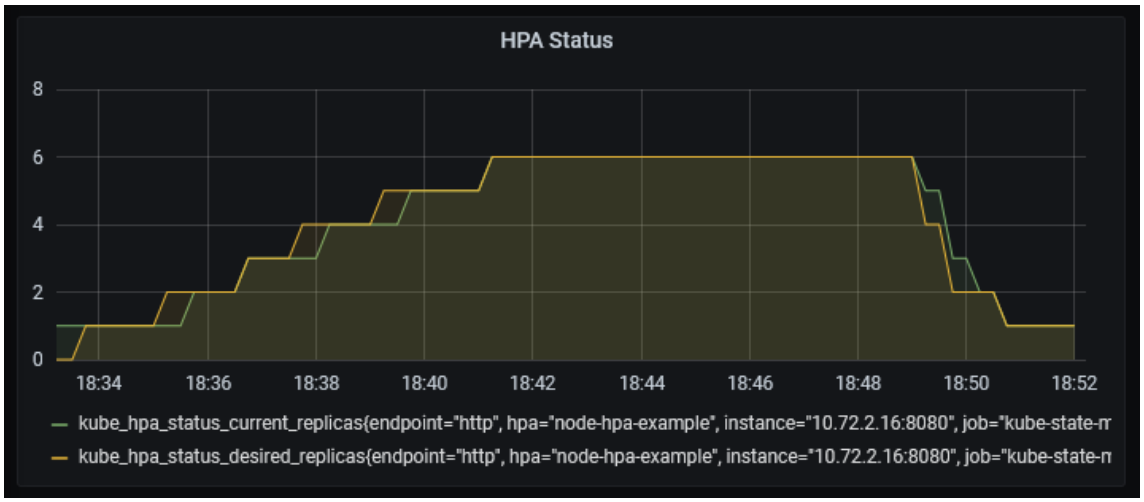

5. ábra. Kiszolgáló podok válaszainak száma 1 másodpercre vetítve

## 2.3. Skálázás az átlagos válaszok számának függvényében

Az egyedi metrikára az átlagos válaszok számát választottam. Az előző mérésből az 5 ábra alapján az átlagos válaszok számának küszöbét a HPA-ban 500-ra állítottam.

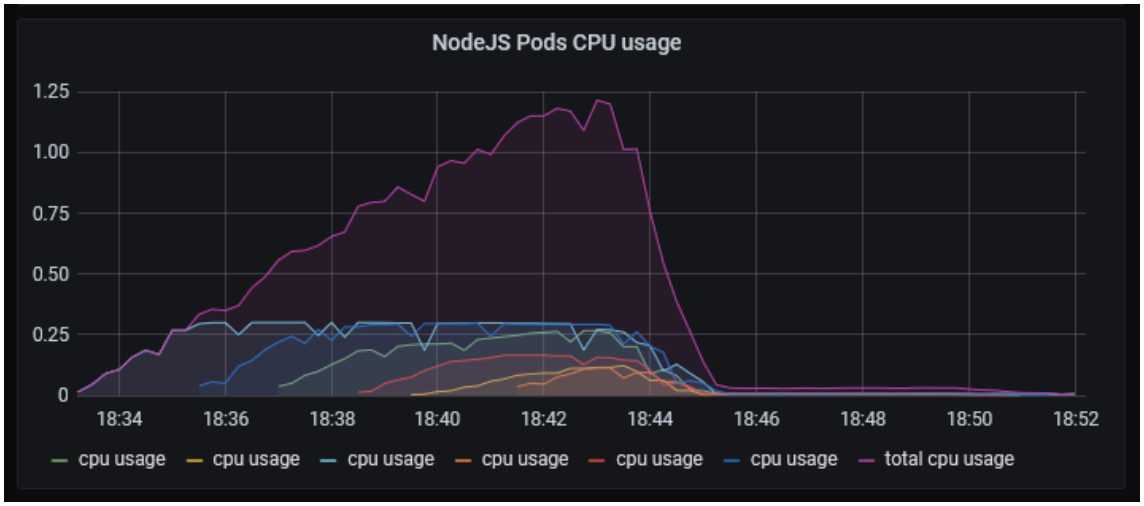
A teszt értékeléséhez ugyanolyan jellegű ábrákat hoztam létre, mint a CPU alapú skálázásnál a 2.2 fejezetben.

A 6 ábrán a HPA által kívánt és tényleges podok száma látható. Összevetve a 2 ábrával a podok száma sokkal lassabban növekszik, és csak hat podot használ ki a maximális tízből.



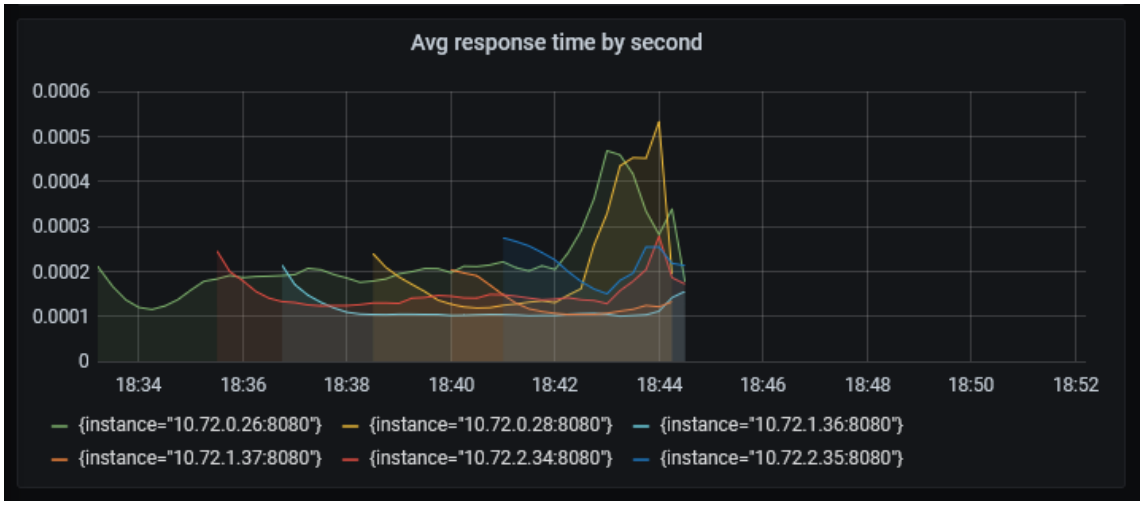

6. ábra. HPA által kívánt és tényleges podok száma

A 7 ábrán látható CPU kihasználtságot összevetve a 3 ábrán látható CPU kihasználtsággal látható, hogy az egyéni metrikákkal történő skálázásnál a CPU kihasználtság sokkal jobban követi a tesztben is definiált lineáris terhelést, valamint a CPU terhelés kisebb, mint a CPU alapú slálázás esetében.



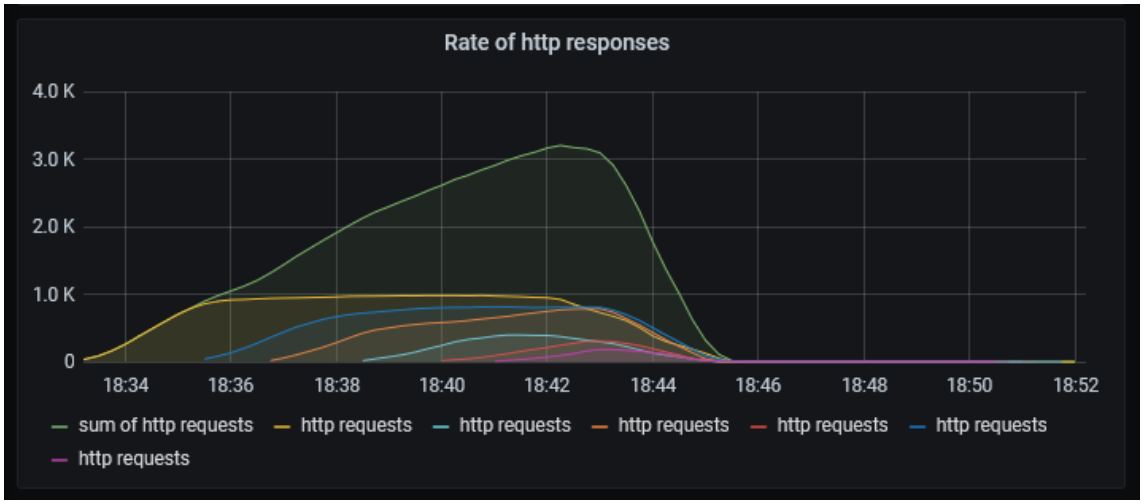

7. ábra. Kiszolgáló podok CPU kihaszáltsága

A 8 ábrán látható válaszidőket összevetve a 4 ábrán látottakkal, a válaszidők a teszt elején még jellegükben megegyeznek és az értékük is hasonló, a 0.1-0.2 ms tartományba esik. Az egyéni metrikákkal történő skálázásnál is megnőnek a válaszidők a teszt végére, azonban ez sokkal később történik meg, és csak két podnál (a hatból) emelkedik számottevően.




8. ábra. Kiszolgáló podok válaszideje 1 másodpercre vetítve

A 9 ábrán is megfigyelhető a 7 ábrához hasonló lineáris növekedés, azaz a válaszok átlagos száma is jól követi a tesztben definiált terhelés növekedését. Azonban itt is megfigyelhető, hogy mivel a HPA a podok válaszainak egy másodpercre vetített számának átlagát veszi, a podok válaszainak száma nem lesz 500 alá szorítva.




9. ábra. Kiszolgáló podok válaszainak száma 1 másodpercre vetítve

## 2.4. Összefoglalás

A 2.2 részben bemutattam egy CPU alapú skálázást és a 2.3 részben egy HTTP válaszok átlagos száma alapján történő skálázást. A tesztek futásakor kapott metrikák azt mutatták, hogy a válaszok átlagos száma alapján történő skálázás CPU kihasználtságban és válaszidőben is jobb eredményeket produkált, mint a CPU alapú skálázás. A mérésekből az a következtetés vonható le, hogy a tényleges terhelést leíró metrikákat használva jobb kihasználtságot és jobb metrikus adatokat lehet kapni az erőforrás alapú skálázáshoz képest. A félév során a rendszer megismerésén és a mérési környezet beállításán volt a fő hangsúly, de a következőkben a célom gazdagabb szcenáriók kidolgozása és az eredmények alaposabb elemzése lesz.

# 3. Irodalom, és csatlakozó dokumentumok jegyzéke

## 3.1. A tanulmányozott irodalom jegyzéke

[1] *What is Kubernetes?* http://inflab.tmit.bme.hu/08o/lezar.shtml, szerk.: Google, 2021. február 1.

[2] *Kubernetes Components* https://kubernetes.io/docs/concepts/overview/components/, szerk.: Google, 2021. március 18.

[3] *Pods* https://kubernetes.io/docs/concepts/workloads/pods/, szerk.: Google, 2021. január 12.

[4] *ReplicaSet* https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/, szerk.: Google, 2021. április 16.

[5] *Deployments* https://kubernetes.io/docs/concepts/workloads/controllers/deployment/, szerk.: Google, 2021. február 11.

[6] *Service* https://kubernetes.io/docs/concepts/services-networking/service/, szerk.: Google, 2021. április 23.

[7] *Labels and Selectors* https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/, szerk.: Google, 2021. április 22.

[8] *Horizontal Pod Autoscaler* https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/, szerk.: Google, 2021. március 25.

[9] *Kubernetes HPA with Custom Metrics from Prometheus* https://towardsdatascience.com/kubernetes-hpa-with-custom-metrics-from-prometheus-9ffc201991e, szerk.: Cezar Romaniuc, 2021. június 26.

[10] *Building Your Own Custom Metrics API for Kubernetes Horizontal Pod Autoscaler* https://medium.com/swlh/building-your-own-custom-metrics-api-for-kubernetes-horizontal-pod szerk.: Cezar Romaniuc, 2021. augusztus 31.

[11] *Overview* https://prometheus.io/docs/introduction/overview/, szerk.: Prometheus Authors, 2018. december 13.

[12] *Prometheus Adapter for Kubernetes Metrics APIs* https://github.com/kubernetes-sigs/prometheus-adapter/blob/master/README.md, szerk.: Prometheus Adapter Authors, 2020. november 6.

[13] *Querying Promehteus* https://prometheus.io/docs/prometheus/latest/querying/basics/, szerk.: Prometheus Authors, 2021. május 4.

[14] *Operator pattern* https://kubernetes.io/docs/concepts/extend-kubernetes/operator/, szerk.: Google, 2021. február 23.

[15] *Using Helm* https://helm.sh/docs/intro/using_helm/, szerk.: Helm Authors, 2021. május 4.

[16] *Custom Resources* https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/, szerk.: Google, 2021. március 29.

[17] *Grafana basics* https://grafana.com/docs/grafana/latest/basics/?pg=docs, szerk.: Grafana Authors, 2021. május 4.

[18] *Configuration Walkthroughs* https://grafana.com/docs/grafana/latest/basics/?pg=docs, szerk.: Prometheus Adapter Authors, 2020. október 28.

[19] *What is k6?* https://k6.io/docs/, szerk.: k6 Authors, 2020. május 4.

[20] *Running distributed k6 tests on Kubernetes* https://k6.io/docs/, szerk.: Simon Aronsson, 2021. február 11.

## 3.2. A csatlakozó dokumentumok jegyzéke