



**Budapesti Műszaki és Gazdaságtudományi Egyetem**  
Villamosmérnöki és Informatikai Kar  
Méréstechnika és Információs Rendszerek Tanszék

Prohászka Botond Bendegúz

# **TERHELÉS HATÁSÁRA AUTOMA- TIKUSAN SKÁLÁZÓDÓ WEBES RENDSZER FEJLESZTÉSE**

KONZULENS

Eredics Péter

BUDAPEST, 2022

# Tartalomjegyzék

<b>Összefoglaló .....</b>	<b>5</b>
<b>Abstract.....</b>	<b>6</b>
<b>1 Bevezetés .....</b>	<b>7</b>
1.1 Általános bevezetés.....	7
1.2 A feladat célja .....	9
1.3 A szakdolgozat felépítése .....	9
<b>2 Létező megoldások bemutatása .....</b>	<b>11</b>
2.1 VPS fogalma .....	11
2.2 A VPS alkalmazási területei .....	12
2.3 VPS szolgáltatók.....	13
2.3.1 VULTR .....	14
2.3.2 ATLANTIC .....	15
2.3.3 CHERRY SERVERS.....	16
<b>3 Architektúra .....</b>	<b>18</b>
3.1 A rendszer mint reverse proxy .....	18
3.1.1 Proxy szerver .....	18
3.1.2 Reverse proxy szerver.....	19
3.2 Elosztott rendszer.....	21
3.3 A rendszer architektúrája és felépítése .....	23
<b>4 Elosztott rendszer megvalósítása.....</b>	<b>24</b>
4.1 Operatív szerverek feladata .....	24
4.2 Master szerver feladata .....	25
4.3 Alkalmazott technológiák .....	26
4.3.1 Operációs rendszer.....	27
4.3.2 Adatbázisszerver .....	29
4.3.3 Demóalkalmazás .....	30
4.4 Megvalósítás .....	31
4.4.1 Szerverpéldányok.....	31
4.4.2 Adatbázisok és a WordPress inicializálása .....	32
4.4.3 Adatbázisok összekapcsolása .....	34

4.5 A megvalósítás során jelentkező problémák .....	36
4.6 A működő rendszer megvalósítása .....	38
4.6.1 Az Üzenőfal .....	38
4.6.2 Az elosztott rendszer terheléselosztó képessége.....	39
<b>5 Automatikus skálázás .....</b>	<b>42</b>
5.1 A megoldási módszerek.....	43
5.2 Felelősségek és feladatok.....	44
5.2.1 Operatív szerver.....	44
5.2.2 Master szerver.....	47
5.2.3 Futtatókörnyezet .....	48
5.3 A terhelésről.....	50
5.3.1 A terhelés monitorozása.....	51
5.3.2 Döntés a terhelés függvényében .....	53
<b>6 Eredmények bemutatása és értékelés .....</b>	<b>55</b>
6.1 Általános értékelés .....	55
6.2 Sebezhetőségek, gyengeségek .....	56
6.3 Továbbfejlesztési lehetőségek .....	57
<b>7 Hivatkozások .....</b>	<b>58</b>

# HALLGATÓI NYILATKOZAT

Alulírott **Prohászka Botond Bendegúz**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2022. 12. 02.

.....  
Prohászka Botond Bendegúz

# Összefoglaló

Szakdolgozatomban egy terhelés hatására automatikusan skálázódó webes rendszer megtervezését és megvalósítását tűztem ki célul, amely rendszer egy felhasználók által is használható szolgáltatást futtat. A dolgozatban bemutatom, hogy egy ilyen rendszert milyen, a piacon jelenleg is elérhető szolgáltatás segítségével lehet üzemeltetni, majd részletezem az általam megvalósított rendszer tervezését és implementálását.

A rendszer három főbb logikai egységből áll össze, ezek nevezetesen a futtatókörnyezet, egy master szerver, valamint az operatív szerverek.

A felhasználók a szolgáltatás igénybevétele során írási és olvasási kéréseket indítanak a rendszer felé, amelyek aktív kapcsolatokon keresztül érkeznek a rendszerbe, valamint a válaszok is a létrejött kapcsolatokon keresztül jutnak el a felhasználókhoz.

A rendszer képes automatikusan skálázódni az aktív kapcsolatok számának függvényében: amennyiben a rendszer úgy érzékeli, hogy az aktív kapcsolatok számának átlaga egységidő alatt monoton nő, úgy újabb és újabb operatív szervereket indít el, amelyek a rendszerbe integrálva elkezdik a beérkező kéréseket feldolgozni, így a felskálázás növeli a rendszer teljesítményét. Ellenkező esetben a rendszer operatív szervereket állít le, ezzel megspórolja azok üzemeltetési költségeit.

A dolgozat végén értékelem az elkészült munkát, bemutatom a képességeit éles használat közben, amelyek igazolják, hogy az ilyen rendszerek működőképeseek, valamint kitérek a rendszer néhány olyan részletére, amelyek fejlesztésével a rendszer használata és működése biztonságosabbá és költséghatékonyabbá tehető.

# **Abstract**

In my thesis, I aimed to design and implement a dynamic performance scaling system for web applications, which system runs a service that can also be used by users. In the thesis, I will present how such a system can be operated with the help of services currently available on the market, and then I will detail the design and implementation of the system I created.

The system consists of three main logical units, namely the runtime environment, the master server, and the operational servers.

While using the service, users initiate write and read requests to the system, which arrive in the system via active connections, and the responses also reach the users via established connections.

The system is able to scale automatically depending on the number of active connections: if the system detects that the average number of active connections is increasing monotonously per unit time, it starts more operative servers, which integrate into the system and start processing the incoming requests, so upscaling increases system performance. Otherwise, the system shuts down operational servers, thus saving their operating costs.

At the end of the thesis, I evaluate the created system, I present its capabilities during live use, which prove that such systems are functional, and I also discuss some of the details of the system, the development of which can make the use and operation of the system safer and more cost-effective.

# 1 Bevezetés

## 1.1 Általános bevezetés

Napjainkban egyre elterjedtebbek az olyan szolgáltatások, melyek esetében a felhasználók a használat során sok kérést indítanak az üzemeltető szerver felé, mivel a szolgáltatás alapját a felhasználó és szerver közti kommunikáció biztosítja. Minél több felhasználó vesz igénybe egy adott szolgáltatást, annál nagyobb képességű hardver-szoftver rendszer szükséges az üzemeltetéshez.

Egy szerverrel a legtöbb kis volumenű szolgáltatást minden további nélkül üzemeltetni lehet, a felhasználótábor és az igények növekedésével azonban a szerver képességeit fejleszteni szükséges: növelni a memória, háttértár méretét, fejleszteni a hálózati kapcsolat sávszélességét és még hosszan lehetne folytatni a sort.

Ám egy szervert a végtelenségig fejleszteni nem lehet, egy olyan szolgáltatást, amit adott esetben felhasználók milliói vagy milliárdjai használnak, képtelenség egyetlen szerverrel üzemeltetni. Erre megoldást az elosztott rendszerek jelenthetnek, melyek legfőbb jellemzője, hogy szerverek sokasága kész a felhasználók kéréseit feldolgozni, szükség esetén pedig még több szervert lehet a kérések feldolgozására állítani (a rendszert felskálázni), csökkenő igények esetén a szerverekkel más feladatot végezni vagy leállítani azokat (leskálázás).

A nagyméretű felhasználótáborral rendelkező szolgáltatások mellett van még (legalább) egy olyan terület, ahol az elosztott rendszer szintén megoldást jelent bizonyos problémákra: ezek a kritikus rendszerek.

A kritikus rendszerek olyan rendszerek, amelyek működésének szünetelése megengedhetetlen, hiszen olyan feladatot látnak el, ami létfontosságú valamilyen okból. Az ilyen rendszerek esetében minden esetben biztosítani kell a rendszer üzemképességét, a rendszernek a feladatát a minimum elvárt teljesítménnyel folyamatosan el kell látnia.

A kritikus rendszerek működésének biztosítására is megoldást nyújthat egy elosztott rendszer, hiszen amennyiben egyszerre több szerver is képes ellátni az elvárt feladatot, egy szerver kiesése nem eredményez szakadást a feladatok ellátásában.

A növekvő számú beérkező kérések kielégítése és a kérések feldolgozásának biztosítása mellett nagyon fontos az adatbiztonság kérdése. A rendszereket védeni kell az olyan jelenségektől, melyek során a tárolt adatok sérülhetnek, vagy egyenesen elérhetlenné, visszaállíthatatlanná válnak. Az ilyen jelenségek megkárosíthatják az üzemeltetőt és a felhasználókat is, ezért valamilyen módon robusztussá kell tenni a rendszert amellet, hogy az ilyen jelenségek bekövetkezésének lehetőségét minimálisra csökkentjük.

Az elosztott rendszer, amennyiben a fontos adatokat is elosztottan, több helyen tároljuk és kezeljük, véd az adatvesztéstől is, mert annak esélye, hogy egy időben, az összes olyan szervert, amely adatbiztonság szempontjából kritikus jelentőségű, egyszerre szenvedjen el drasztikus mértékű adatvesztést, igen csekély. Így tehát az adatbiztonság szempontjából is előnyös lehet egy elosztott rendszer.

A rendszerek robusztussá tétele mellett ugyanakkor mindenképp figyelembe kell venni a szolgáltatás üzemeltetésének költségeit. Egy átlagos szolgáltatásnak megvannak a jellemző csúcsidezőszakai, amikor akár nagyságrendekkel több felhasználó veszi igénybe a szolgáltatást, illetve a mélypontok, melyek esetében alig akad feldolgozandó kérés.

Az ingadozó számú beérkező kérések esetében, amennyiben meghatározott számú szerverrel dolgoztatjuk fel azokat, úgy kell meghatározni a szerverek számát és teljesítményét, hogy azok a legnagyobb terhelés mellett is képesek legyenek a feladatokat a kellő színvonalon ellátni. A rendszer így képes lesz a legnagyobb terhelés mellett is működni, ebben az esetben közel van költséghatékonyasága optimumához is, ám a beérkező kérések számának és a terhelés csökkenésének következtében a rendszer egyre inkább túlméretezetté kezd válni, és egyre nagyobb üzemeltetési és fenntartási költség jut egyetlen kérésre. A rendszer ekkor egyre kevésbé működik költséghatékonyan egységnyi kérésre eső üzemeltetési költsége egyre nagyobb lesz, a kevesebb kérést gyengébb, olcsóbb fenntartású rendszer is képes lenne ellátni.

Ahhoz, hogy egy rendszerben mindig a szükséges számú szerver működjön, minden időpillanatban meg kell határozni, hogy mennyi szerver kell az adott időpillanatban és mennyire lesz szükség a közeljövőben. Ezt a feladatot egy automatikusan skálázódó rendszer tudja ellátni.



## 1.2 A feladat célja

Feladatom célja egy olyan rendszer megtervezése és implementálása, amely képes az elosztott működésre, valamint az automatikus fel- és leskálázásra, ezáltal a beérkező kérések számának függvényében a megfelelő számú szerverrel szolgálja ki azokat.

A feladatnak kimondott célja, hogy bemutassam, egy ilyen rendszer képes arra, hogy a terhelés monitorozása mellett fel- vagy leskálázza magát oly módon, hogy a szolgáltatás folyamatos elérhetőségét biztosítja.

Nem célom, hogy egy optimálisan és költséghatékonyan működő rendszert hozzak létre; a rendszerrel csupán egy demonstrációs jellegű megoldást szeretnék prezentálni. A feladat elvégzése során elérendő optimális és költséghatékony működés definícióját úgy határozom meg, hogy a rendszer a párhuzamosan beérkező aktív kapcsolatokkal lineárisan arányosan indítson backend szervereket, az aktív kapcsolatok és a futó backend szerverek száma között közel 1:1 lineáris megfeleltetést definiálok.

Ezt a meghatározást ugyanakkor bármikor meg lehet változtatni: a rendszert kifejezetten úgy tervezem meg és hozom létre, hogy annak működése a különböző egységek módosításával vagy cserélésével könnyen módosítható legyen. A rendszer moduláris felépítése miatt a működés könnyen módosítható, a rendszer továbbfejlesztése során éppen ezért nem ütköznénk nagy nehézségekbe.

## 1.3 A szakdolgozat felépítése

A szakdolgozatban elsőként megvizsgálom, hogy egy ilyen rendszer üzemeltetéséhez milyen infrastruktúrát lehet használni, meghatározom a fogalmát, bemutatom annak alkalmazási területeit, valamint bemutatom három szolgáltató termékét.

Az elérhető infrastruktúra témaköre után áttérek az általam elkészített rendszerre: bemutatom a rendszer kétféle arcát, majd felvázolom, hogy a rendszer ezen két megközelítése hogyan áll össze, ismertetem az architektúrát.

A rendszer felépítésének bemutatása után pontosítom, hogy a rendszerben milyen szerverek találhatóak, azoknak mik a feladatai, bemutatom az alkalmazott technológiákat és részletezem a megvalósítást. Felvázolom, hogy milyen problémák jelentkeztek a rendszer megvalósítása során, és bemutatom a végleges megoldást, valamint annak elkészítését.

Az elosztott rendszer bemutatását követően kifejtem az automatikus skálázás témakörét: áttekintem a felmerülő megoldási módszereket, meghatározom, hogy az egyes szervereknek mik a feladatai és felelősségei, majd bemutatom, hogy miként monitorozza a rendszer a terhelést, valamint miként dönt a terhelés függvényében a szerverek számáról.

## 2 Létező megoldások bemutatása

Ahhoz, hogy egy terhelés hatására skálázódó webes rendszert üzemeltetni lehessen, mindenképpen kell valamilyen fajta futtató környezet. Kell egy olyan infrastruktúra, ami biztosítja annak a lehetőségét, hogy a felhasználó (ebben az esetben a felhasználó és a vevő ugyanarra a személyre, szervezetre utal, amely személy vagy szervezet az infrastruktúrát rendeltetésszerűen igénybe veszi) a saját döntése alapján tetszőleges számú és tulajdonságú virtuális számítógépet, szervert el tudjon indítani; természetesen az infrastruktúra tulajdonságait és a szolgáltatás feltételeit figyelembe kell venni.

Ez a futtató környezet lehet egy VPS szolgáltatás<sup>1</sup>, de lehet egy közönséges asztali számítógép vagy laptop is, ám ezek hardvertulajdonságai adott esetben több nagyságrenddel is eltérhetnek egymástól, így amikor egy adott felhasználási módhoz keressük a megfelelő futtató infrastruktúrát, mindenképp mérlegelni kell a lehetőségek hardvertulajdonságait is.

Ebben a fejezetben szeretném tárgyalni, hogy mit jelent pontosan a virtuális privát szerver, melyek az alkalmazási területei, valamint bemutatni néhány VPS szolgáltatót.

### 2.1 VPS fogalma

A virtuális privát szerver egy olyan virtuális számítógép, melyet szolgáltatásként árulnak és melyet a vevő úgy kezelhet, mintha a saját számítógépét, szerverét használná. A nevét alkotó szavak pontosan meghatározzák magát a fogalmat: *szerver*, mert mindig elérhető a jellemzően jelentős számítási kapacitással rendelkező virtuális számítógép; *virtuális*, mert az azt futtató szerver több ilyen virtuális számítógépet futtathat, és azok számára a hardvereket virtualizálja, azaz például nincs mindegyik virtuális számítógépnek saját, különálló háttértára; valamint *privát*, mert a vevő saját számítógépként, szerverként tekinthet rá.

Legfontosabb jellemzője ezeknek a szolgáltatásoknak, hogy a kínált virtuális számítógépeket szerverközpontokban fizikailag jelen lévő szervereken futtatják. Az ilyen

---

<sup>1</sup> *Virtual Private Server*, azaz Virtuális Privát Szerver. [24]

szerverparkok üzemeltetése maga is egy külön szolgáltatás, lehet vásárolni szerver rack helyet, ahol a saját fizikai szerverünket elhelyezhetjük. Ezen szolgáltatások nem összekeverendők a VPS szolgáltatásokkal, mert a VPS szolgáltatások esetében egy virtuális szerver felhasználási jogát vásároljuk meg, amelyet futtathat a szolgáltató cég tömördek fizikai szervere közül bármelyik egy szerverközpontban

Előnyös tulajdonsága az ilyen VPS szolgáltatásoknak a személyre szabhatóságuk. Egy átlagos VPS szolgáltatás vásárlásakor különböző konstrukciók közül lehet választani: gyakorlatilag a virtuális szerver minden fontos tulajdonságát módosítani lehet. Személyre lehet szabni többek között a processzormagok számát, a memória és a háttértár méretét, a szervert és a hálózatot összekötő kapcsolat sávszélességét csak úgy, mint a futtatott operációs rendszert, melynek esetében lehetőségünk van saját operációs rendszert is telepíteni. Ez a széles személyre szabhatóság éppen azért lehetséges, mert virtuális számítógépekről van szó.

Az előnyös tulajdonságok ellenpólusaként megjelenhet a vevőkben olyan aggály, melynek alapját az a tudat képezi, hogy a vevő adatai, információi vagy éppen kódok, szoftverek valaki más tulajdonában álló hardveren vannak tárolva vagy futtatva. Ez a vevők többségének csupán elhanyagolható kockázatot jelent, ugyanis jellemzően az ilyen szolgáltatók magas színvonalú szolgáltatásokat nyújtanak biztonsági szempontokból is.

Adatbiztonság felől megközelítve a legjellemzőbb különbség a szerverek üzemeltetésének helyszíne: VPS szolgáltatás esetén egy jellemzően jól őrzött, áramkimaradásoktól védett, tűzesetre kötelezően hibátlanul felkészített és hasonló biztonsági intézkedésekkel jellemezhető épületről beszélünk, amivel szemben ott áll adott esetben a saját ingatlan (lakás, ház), amelyben fizikailag el van helyezve a szerver: rendszerint nincs biztonsági szolgálat, nincs szünetmentes táp vagy dízelaggregátor, éppen ezért még ki-mondottan előnyös is lehet egy ilyen szolgáltatás igénybevétele abban az esetben, ha a szünetmentes szolgáltatás fontosabb, mint az a tudat, hogy minden adat és kód saját hardveren legyen.

## **2.2 A VPS alkalmazási területei**

Az előbb bemutatott tulajdonságokat összegezve elmondható, hogy az ilyen szolgáltatások célcsoportját azok a (kisebb) cégek alkotják, melyeknek nincs szükségük vagy lehetőségük saját szervereket a kellő minőségben üzemeltetni: ezek a cégek általában nem

rendelkezik nagy IT részleggel, valamint a szolgáltatásaik között sem szerepel IT tevékenység. Az ennél nagyobb cégek, amelyeknek már szükséges, saját szervereket üzemeltetnek. Az ennél kisebb szereplők (például a szakdolgozatokat készítő hallgatók) számára pedig elegendő egy saját üzemeltetésű egyszerű szerver (és ebben az esetben pedig inkább beszélhetünk már nem használt számítógépekről, laptopokról, amik ki lettek nevezve szervernek) üzemeltetése, mivel az olcsóbb és az elégséges tulajdonságokkal rendelkezik.

Tehát rendszerint a kisebb cégek veszik igénybe ezeket a szolgáltatásokat, amelyek segítségével üzemeltethetik honlapjukat, levelező rendszerüket, esetleg belső használatú munkaidőnyilvántartó rendszerüket stb.

## 2.3 VPS szolgáltatók

A következő alfejezetekben olyan szolgáltatókat mutatok be, akik ilyen VPS szolgáltatásokat kínálnak. Annak érdekében, hogy csak az általam elkészített feladathoz releváns szolgáltatókat és azok termékeit részletezzem, a konzulensem által megfogalmazott elvárások szerint kerestem szolgáltatásokat.

A keresés során fontos szempont volt, hogy az adott virtuális gépet egy *snapshot* (pillanatkép) vagy *image* (kép) felhasználásával is lehessen inicializálni. Egy ilyen pillanatképben el lehet menteni egy futó virtuális gép aktuális állapotát minden beállítással együtt, majd egy új virtuális gép helyezésénél meg lehet adni ilyen snapshot állományt, aminek használatával az újonnan indított gép néhány tulajdonságtól eltekintve (IP cím, MAC cím stb.) teljes mértékben megegyezik azzal a géppel, amiről a pillanatképet készítették. Ehhez a kritériumhoz szorosan kapcsolódik az az elvárás, hogy a meglévő gépről lehessen készíteni snapshot fájlt. Jellemzően ezeket a feltételeket mindegyik szolgáltató teljesítette.

A másik és egyben a kettő közül a jelentősebb kritérium a számlázáshoz, elszámoláshoz kapcsolható: a szolgáltatás díját a számítógép futásideje alapján kell meghatározni, azaz csak arra az időre szeretnénk fizetni, amikor a virtuális gép valóban futott. A legtöbb szolgáltató a hagyományos havidíjas elszámolási modellt alkalmazza, így ez a feltétel meglehetősen sokat kiszűrt közülük.

Nem elhanyagolható szempont a fizikai elhelyezkedés sem. A VPS szolgáltatók megismerése során figyeltem az adatközpontok elhelyezkedésére is: ha egy adott régióban szeretnénk a szolgáltatás lehetőségeit kihasználni, akkor optimális az adott régióhoz

minél közelebb üzemeltetni a szervert. Az én esetemben ez a régió Magyarországot jelent.

### 2.3.1 VULTR

A floridai West Palm Beach-i székhellyel rendelkező VULTR [1] cég VPS szolgáltatások terén minőségi infrastruktúrán, azonnali lehelyezéssel, többféle (akár saját) operációs rendszerrel implementálva, az alapvető biztonsági mechanizmusokkal ellátott szolgáltatást nyújt, valamint megfelel az elvárásoknak.

A díjszabás óraalapú, így csak azért az időért kell fizetni, amit tényleg futott a szerver. Lehet snapshot fájlt létrehozni és lehet egy image alapján több szervert is lehelyezni a szolgáltató több kontinensen, több városban található szerverein.

A szervert lehet API<sup>2</sup> hívásokkal módosítani, az API hívások a standard HTTP<sup>3</sup> kifejezéseket, válaszkódokat használják, betartják a RESTful API [2] tervezési elveket és természetesen megvalósíthatóak velük a CRUD<sup>4</sup> műveletek is.

A kommunikáció során a kéréseket és a válaszokat konzisztens és jól formázott JSON<sup>5</sup> formátumban küldi, ezenkívül a könnyen megérthető hibaüzenetekből gyorsan meg lehet oldani egy esetleges hibát. Az API kérés (*request*) parancsai közé tartoznak a DELETE, GET, PATCH, POST és PUT műveletek. Fontos részlet, hogy ha az applikációnk másodpercenként több, mint 20 kérést küld, lehetséges, hogy az API 503-as (*Service unavailable*, azaz a szolgáltatás nem elérhető) hibakóddal fog visszatérni. A válaszok (*response*) a jól ismert HTTP válaszkódok (pl.: 200 = *OK*, 400 = *Bad request* [azaz rossz kérés], 404 = *Not found* [azaz nem található]).

---

<sup>2</sup> *Application Programming Interface* [17]

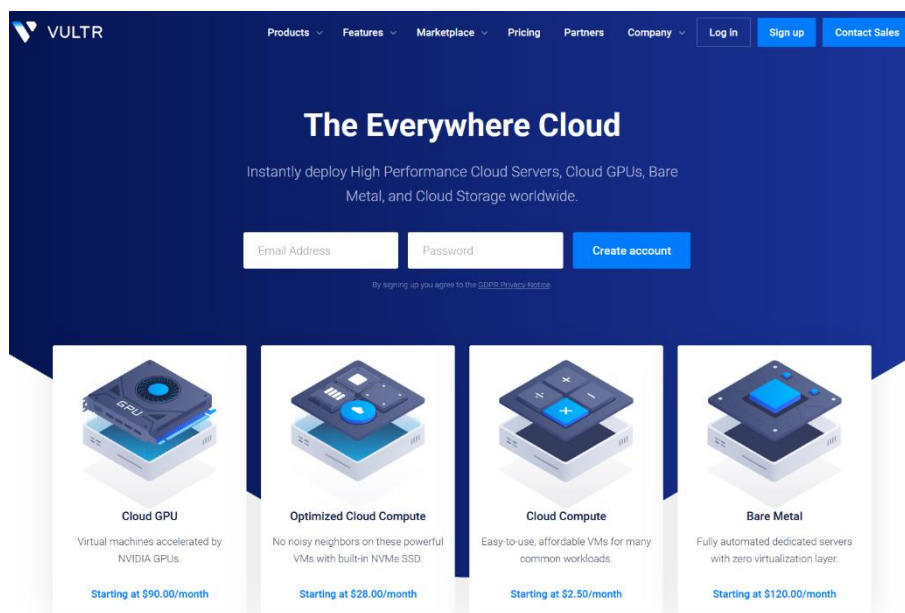
<sup>3</sup> *Hypertext Transfer Protocol* [19]

<sup>4</sup> *Create, Read, Update and Delete*, azaz létrehozás, olvasás, módosítás és törlés.

<sup>5</sup> *JavaScript Object Notation* [21]

Az API használata során lehetőségünk van a parancsok felparaméterezésére, amik használatával már könnyen lehet például hitelesítési (*authentication*) szolgáltatást használni, biztonsági mentést (*backup*) készíteni, a tűzfalat elérni és még számos más szolgáltatást használni.

A Magyarországhoz legközelebbi adatközpontjuk Varsóban található, de viszonylag közel van még adatközpontjuk Frankfurtban is.



1. ábra A VULTR szolgáltató honlapjának részlete

## 2.3.2 ATLANTIC

A VULTR-hoz hasonlóan az ATLANTIC [3] is egy Egyesült Államokbeli cég, ennek székhelye Orlandóban található. A fent említett szolgáltatásokat szintén teljeskörűen biztosítja.

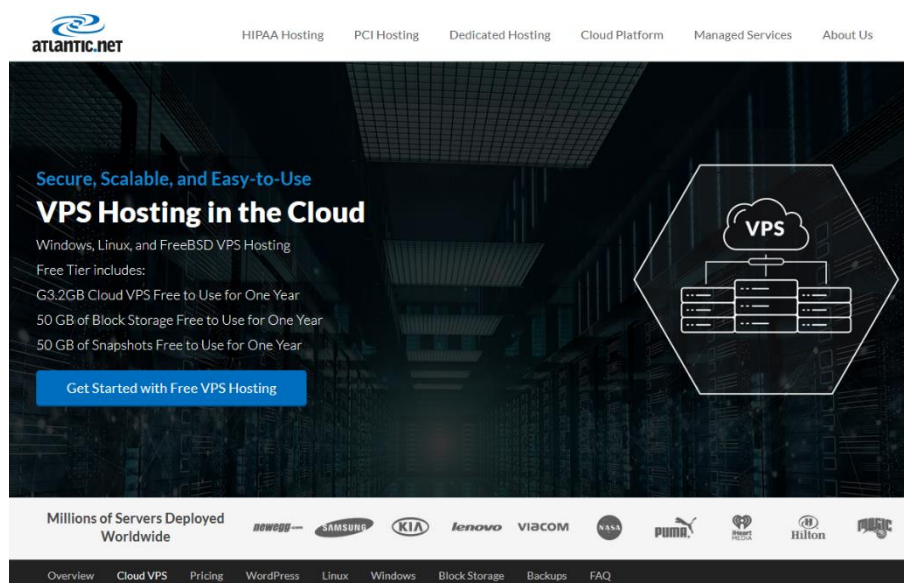
A cég honlapján található API dokumentációból kiderül, hogy ez az interfész is szintén RESTful lekérdezési interfészt biztosít a szolgáltatást igénylőnek. Ez az API HTTPS<sup>6</sup> kommunikációt használ a végpontok között, a dokumentáció szerint szinte minden programozási nyelv használható és átvihető ezen keresztül. Saját API eszközöket is lehet írni, ezeknek a részletezése és ehhez segítség megtalálható a dokumentációban.

---

<sup>6</sup> Hypertext Transfer Protocol Secure [25]

Az előző példában említett API-képességeket az ATLANTIC szintén biztosítja, ám ez esetben a hibakódok terén van eltérés: mivel sok a kötelezően elvárt paraméter, van esély hibás kérés küldésére, amikre válaszul nem a megszokott HTTP kódokat fogjuk kapni, hanem a cég által definiált hibakódokat, amely kódok jelentése a dokumentációban megtalálható. Továbbá lehetőség van az image, snapshot állományok készítésén túl publikus IP címek kezelésére is.

Európában viszonylag kevés adatközponttal rendelkezik, ugyanis mindössze egy adatközpont van Londonban, egy pedig hamarosan megnyílik Amszterdamban. Az adatközpontok elhelyezkedését vizsgálva szépen látszik, hogy az ATLANTIC a hazai piacot célozta meg: míg az Egyesült Államokban öt helyen van adatközpontjuk (Ashburn, Dallas, New York, Orlando, San Francisco), egy található Kanadában (Toronto), valamint egy Európában (London), hamarosan pedig lesz egy Hollandiában (Amszterdam) és Ázsiában, Szingapúrban.



2. ábra Az ATLANTIC szolgáltató honlapjának részlete

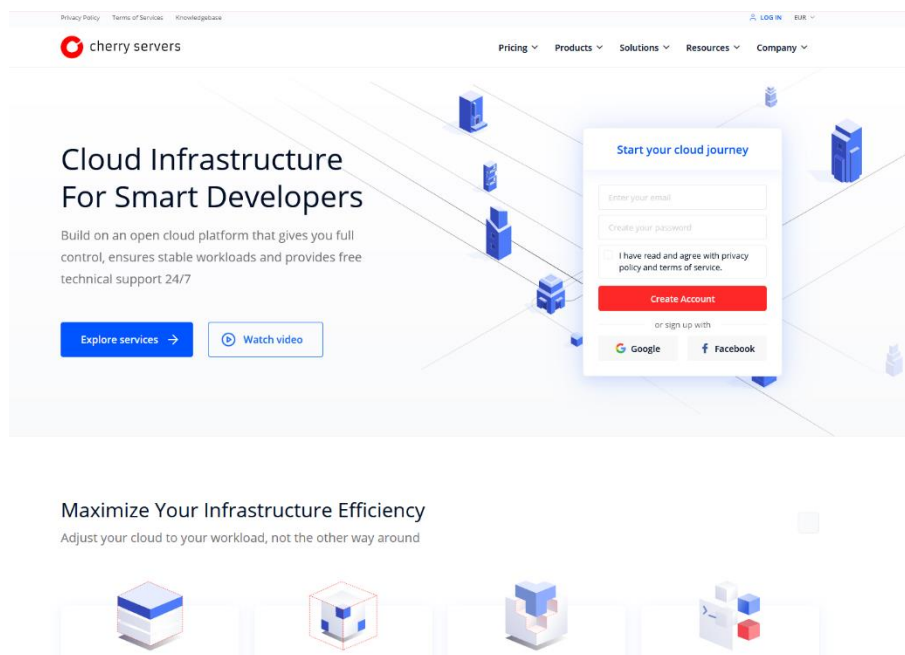
### 2.3.3 CHERRY SERVERS

A litván és holland adatközponttal rendelkező CHERRY SERVERS [4] hasonló árkategóriában mozog amerikai társaival, természetesen a hardver összeállítását itt is saját magunknak választhatjuk. A korábban említett szolgáltatók által nyújtott lehetőségeket a CHERRY SERVERS is kínálja. Az API e szolgáltatásnál is HTML-alapú, REST API [5].



A számunkra szükséges elvárásokat a cég terméke teljesíti, továbbá van lehetőség IP címek, alhálózatok kezelésére is. A REST API-n kívül képes kezelni natív Python és GO könyvtárak használatát és biztosít még parancssoros interfészt is, ahol operációsrendszer-szintű hozzáférést ad *shell scripting* megoldásokhoz.

A három bemutatott szolgáltató közül ez rendelkezik a legkevesebb adatközponttal: a litván és holland adatközpontokon kívül csupán egy található még Chicagóban.



**3. ábra A CHERRY SERVERS szolgáltató honlapjának részlete**

## 3 Architektúra

Mint minden komplexebb informatikai rendszer tervezése során a modernnek számító bevett praktikák és tervezési elvek szerint a készítendő rendszert kisebb, logikailag összetartozó modulokra, egységekre érdemes tagolni. Ennek következményeként az egységek funkcionalitásukat és felelősségüket tekintve egyszerűbbek, valamint az ilyen modulokat tovább is lehet még bontani kisebb részekre.

Ez a megközelítés két okból is előnyös. Nagyonbbrészt könnyebb és kevesebb hibával jár több, de egyenként kevésbé komplex modul elkészítése, mint egy, de nagyon összetetté, melynek során bonyolult függőségi gráfokat kell kezelni tudni. Másrésről azért előnyös, mert a kisebb egységeket könnyebben tudjuk tesztelni, hogy megbizonyosodjunk a helyes működésről (ennek neve *unit testing*, azaz egységtesztelés).

A megvalósítandó rendszer két nagyobb logikai egységre bontható, amelyek egymástól kellően elkülönülnek. A rendszerre ezek alapján kétféleképpen lehet tekinteni: az egyik a *rendszer mint reverse proxy rendszer*, a másik megközelítés a *rendszer mint elosztott rendszer*.

A fejezet következő részeiben kifejtem, mit jelenetnek az egyes megközelítések, hogyan kapcsolódnak az elvégzendő feladathoz, valamint azt, hogy magas szinten bemutatva hogyan illeszkednek az teljes rendszerbe.

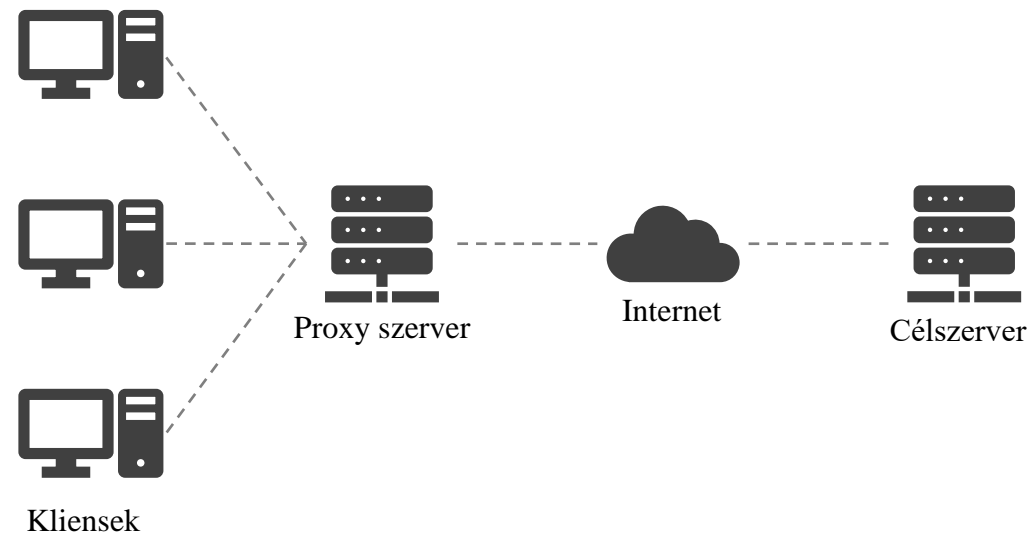
### 3.1 A rendszer mint reverse proxy

Mielőtt bemutatnám a reverse proxy fogalmát, működését és relevanciáját, logikailag fontos megérteni a proxy vagy proxy szerver fogalmát. (A magyar terminológiában nincs gyakran alkalmazott megnevezése a proxy szervernek, ezért a továbbiakban mind a proxy, mind a reverse proxy szerveret az eredeti angol néven nevezem.)

#### 3.1.1 Proxy szerver

A proxy egy olyan szerver, amelyhez kliensgépeket csatlakoztatnak és elsődleges célja, hogy a hozzá csatlakoztatott kliensek kéréseit továbbítsa más szervereknek, valamint kezelnie kell a válaszok érkezését és azok megfelelő helyre történő küldését. A kliens által indított kérést a proxy szerver a saját nevében továbbítja a célszerverre, adott

esetben a proxy szerver egy olyan kérést, ami a szerverhez való csatlakozást igényli, meg tud oldani úgy, hogy a kliensnek nem kell a célszerverhez csatlakoznia. Az alábbi ábrán egy egyszerű topológia látható proxy szerverrel.



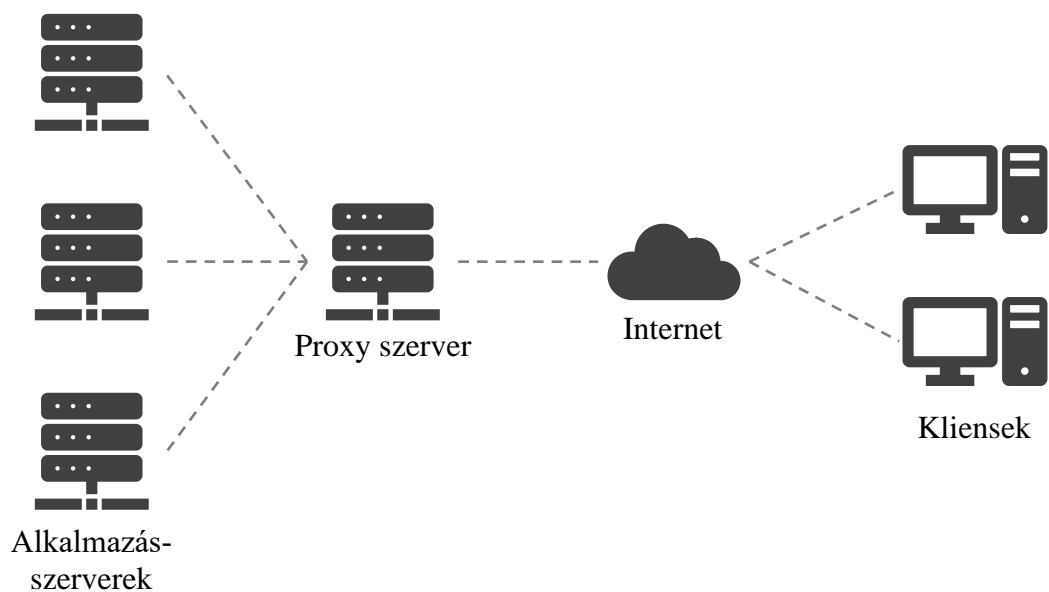
**4. ábra Egyszerű topológia a proxy szerver működésének szemléltetésére**

A 4. ábra által mutatott topológián látható, hogy amennyiben egy kliens kérést szeretne küldeni egy szervernek, a kérése át fog haladni a proxy szerveren is, ami a kérés feldolgozása után továbbítja a kérést. Visszafele is hasonló a helyzet.

A proxy szerverek nagyon jelentős szerepet kapnak abban az esetben, amikor szükség van biztonsági intézkedésekre, ugyanis kiválóan be tudják tölteni az elsődleges tűzfal vagy naplózó szerver szerepét is.

### **3.1.2 Reverse proxy szerver**

A reverse proxy szerver (azaz megfordított proxy) egy olyan proxy szerver rendszer, amely megfordított működéssel bír. Ez azt jelenti, hogy a (reverse) proxy szerver „mögé” nem a kliensek sorakoznak fel, hanem a szerverek, és a proxy szerver „túloldala-  
lán” a célszerver helyett a kliensek vannak.



**5. ábra Egyszerű topológia a reverse proxy szerver működésének szemléltetésére**

A reverse proxy szerver egyik feladata, hogy a beérkező kéréseket a megfelelő alkalmazáserverhez továbbítsa. A megfelelő szerver kiválasztásának folyamata többféleképpen történhet. A leggyakoribb döntési logikák kézenfekvő megoldások: a felhasználás módja, a megvalósított feladat meghatározó.

Az egyik legegyszerűbb döntést akkor lehet meghozni, ha az alkalmazáserverek különböző funkcionalitással bírnak. Ebben az esetben (gyakorta) minden szerver más feladatot lát el: van fájlserver, levelező szerver stb. A döntést így könnyű meghozni, mert a kérés meghatározza, hogy melyik szerver felé kell továbbítani.

Egy másik döntési logika alapját képezheti a kliens és a szerver elhelyezkedése: a kéréseket a kientől mért legközelebb lévő szervernek küldi tovább a reverse proxy szerver, ezzel jelentős időt lehet spórolni. (Az, hogy a távolság mit jelent és hogyan kell mérni, feladatfüggő: lehet fizikai elhelyezkedés kilométerben, valamint lehet az eléréshez szükséges hálózati ugrások számát is mérni.)

Az én feladatom szempontjából a legrelevánsabb, hogy a döntést a szerverek terheltsége alapján is meg lehet hozni: az alkalmazáserverek ebben az esetben azonos funkcionális képességekkel rendelkeznek, ám terhelésük minden időpillanatban jelentősen különbözhet. Ennek kiküszöbölésére a terheléelosztás jelent megoldást: a beérkező kérést egy kevésbé terhelt szervernek küldjük.

A feladat megoldásához szükség van egy ilyen reverse proxy szerverre, ami terhelés alapján el tudja dönteni, hogy a kéréseket mely szervernek továbbítsa.

## 3.2 Elosztott rendszer

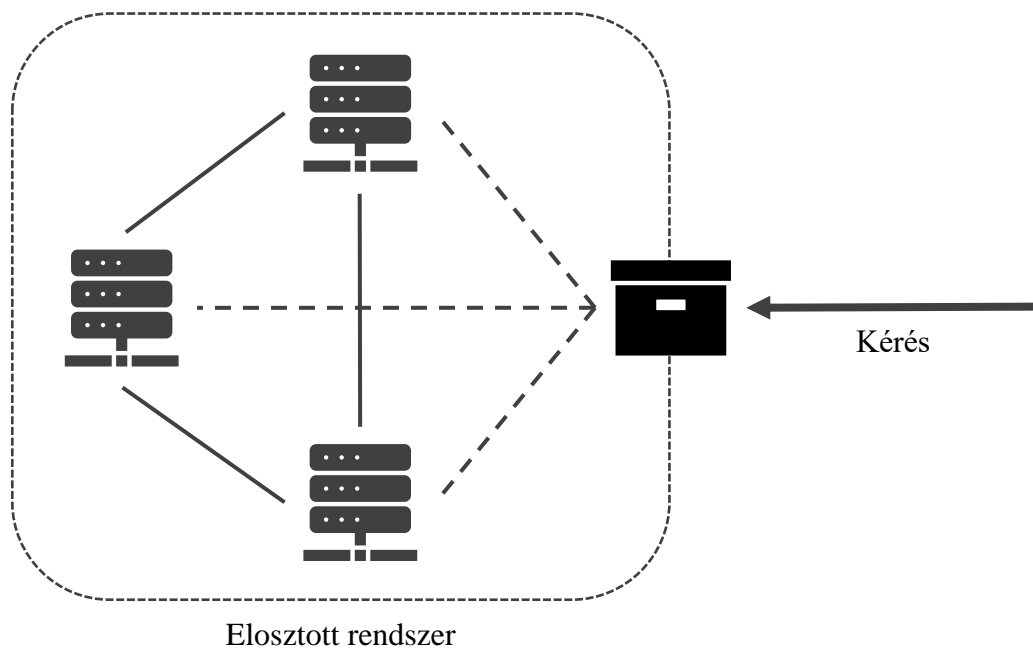
Összetettebb feladatok esetében régebben bevett megoldás volt, hogy a feladat megoldására allokált hardvert megpróbálták a végletekig fejleszteni, hogy az adott hardver minél gyorsabban meg tudja oldani a feladatot. Ennél valamivel költséghatékonyabb és optimálisabb megoldást jelent a napjainkban már jelentősen elterjedt elosztott rendszer használata.

Az elosztott rendszerre kívülről úgy tekinthetünk, mint egyetlen egy számítógépre, ám a belső működése annál összetettebb: a rendszer ugyanis magába foglal több (jellemzően virtuális) számítógépet, amelyek egymás között kommunikálnak, mely kommunikáció és belső működés el van rejtve a felhasználó elől.

A beérkező kéréseket (parancsokat, feladatokat stb.) a rendszer kiosztja a belső szerverek<sup>7</sup> között, majd a kérést csupán az egyik szerver fogadja és dolgozza fel. Természetesen amennyiben egy olyan kérést kap a rendszer, amely valamilyen állapotváltozást, adatváltozást eredményez, úgy annak az egész rendszerre hatással kell lennie. (Gondoljunk bele: ha egy kérés egy új rekordot vesz fel vagy töröl egy adatbázisból, mennyire fontos, hogy onnantól kezdve minden más szerver is a módosított adatbázis alapján működjék, ne csak a kérést feldolgozó szerver. Ellenkező esetben inkonzisztens lenne az elosztott rendszer állapota.)

---

<sup>7</sup> Mivel a számítógép és a szerver fogalmak között jelen esetben nincs érdemi különbség, szinonimaként kezelem a két kifejezést.



**6. ábra Elosztott működést bemutató sematikus topológia**

Az elosztott rendszer határát egy olyan egység reprezentálja, amivel a felhasználó interakcióba lép. Ez az egyetlen pont, amely kívülről látható, az összes kérés ezen ponton keresztül érkezik a rendszerbe. (Természetesen az üzemeltető szakemberek a belső szervert is elérik külön-külön ebben az esetben nem (feltétlenül) az előbb említett belépési ponton keresztül.)

A 6. ábra által bemutatott topológián az elosztott rendszer határát egy fekete doboz (angol terminológiával *black box*) reprezentálja, ennek ugyanis esetünkben komoly üzenete van: nem tudjuk ugyanis, hogy mi alapján dönti el a rendszer, melyik kérést hova továbbítja, egyelőre ez marginális részlet. (A black box kifejezés pontosan erre utal: nem tudjuk, hogy az adott egység hogyan és mi alapján működik, a lényege, hogy egy adott bemenetre valamilyen kimenetet adjon. Ennek ellentéte a fehér doboz [*white box*], ebben az esetben a belső működést pontosan ismerjük.)

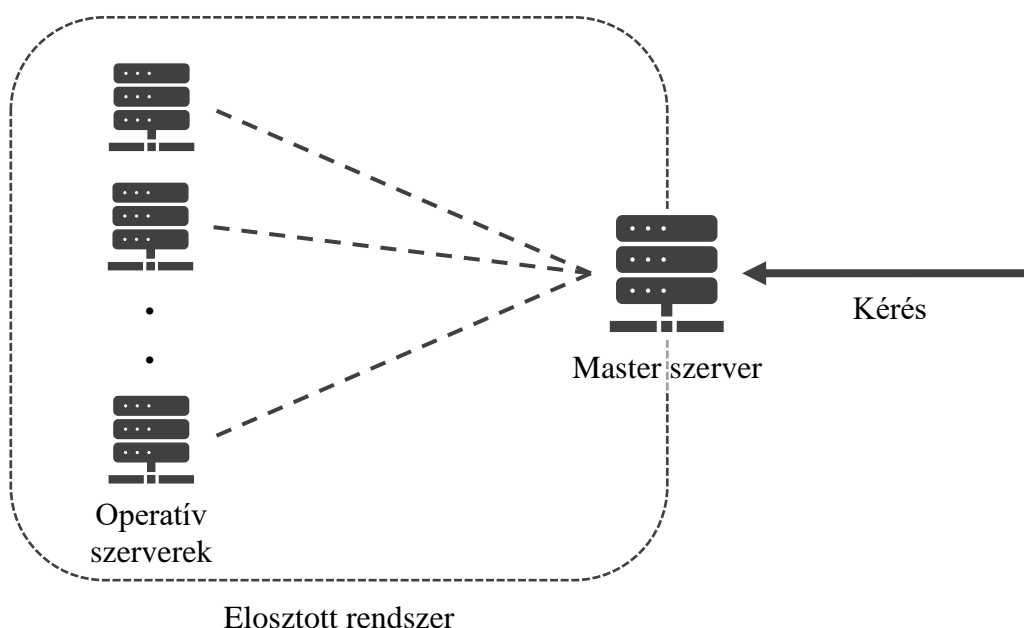
A készítenő automatikusan skálázódó webes rendszer alapja egy ilyen elosztott rendszer, mert a kívánt skálázás elérését ez tudja biztosítani.

### 3.3 A rendszer architektúrája és felépítése

A megvalósított rendszer architektúráját tekintve a reverse proxy működéssel (is) rendelkező szerverből, valamint egy elosztott rendszerből áll össze. A reverse proxy működéssel rendelkező szerver a továbbiakban *master szerver* nevet viseli, a többi, elosztott működést biztosító szerverek pedig az *operatív szerver* nevet kapják.

A beérkező kéréseket a master szerver valamilyen logika alapján továbbítja az operatív szervereknek, azok pedig feldolgozzák a kéréseket.

A szerverek számáról eddig nem esett szó: a master szerverből praktikusán egy darab van, viszont operatív szerverből tetszőleges számú (de legalább egy darab) lehet. Az operatív szerverek számát viszont korlátozhatja az infrastruktúra: egy nagy teljesítményű szerveren sem lehet „végtelen” sok virtuális számítógépet elhelyezni és futtatni, egy bizonyos mennyiség után ugyanis a host szervernek olyan sok üzemeltetési többletfeladata lesz, hogy a virtuális gépek számának növekedésével egyre romló minőségben tudja csak futtatni a virtuális gépeket.



7. ábra A rendszer sematikus architektúrája

## 4 Elosztott rendszer megvalósítása

Ebben a fejezetben szeretném részletezni az elosztott rendszer megvalósításának folyamatát. A fejezetben először bemutatom a master és az operatív szerverek jellemzőit, az alkalmazott technológiákat, majd részletezem a megvalósítás folyamatát, kitérek a fejlesztés során felbukkanó hibákra és nehézségekre is. Végül röviden bemutatom a működő elosztott rendszert felhasználói szemszögből is, a használattal egyúttal hitelesíteni szeretném, hogy az alrendszer készen áll a rendszerbe integrálásra.

A megvalósított rendszer egyik lényegi része tehát az elosztott alrendszer. Ennek az elosztott rendszernek a feladata, hogy a beérkező kéréseket a megfelelő terhelésselosztás szempontjából meghatározott számú operatív szerver között elossza, valamint a kérések fel legyenek feldolgozva. Az elosztott rendszerben elhelyezkedő operatív szervereket a külvilágtól „valami” elhatárolja, ahogyan azt a 3.2 fejezetben bemutatott topológia is mutatja, amely az egész elosztott működést irányítja. Ez a „valami” a master szerver.

A master szervert össze lehet vonni egy kitüntetett operatív szerverrel, ekkor ez a szóban forgó szerver az operatív szerverek tulajdonságaival is rendelkezik, illetve a master szerver funkcionalitását is megvalósítja. Ez a megoldás lehetővé teszi a rendszergazda számára az egyszerűbb üzemeltetést olyan szempontból, hogy kettő helyett egy darab szervert kell csak kezelni, ugyanakkor annak az egy szervernek a karbantartása komplexebb feladat, mivel a szerver maga is „bonyolultabb”, azaz bővebb funkcionalitással rendelkezik, ami több telepített alkalmazással jár.

A feladat tervezése során úgy döntöttem, hogy a végső rendszerben a master szerver egy önálló egység lesz, az azt működtető szerver más funkcionalitással az elosztott rendszer szempontjából nem rendelkezik, ám az elosztott rendszer alrendszer elkészítése során a master szerver egyben egy operatív szerver is lesz, hogy az alfeladat megvalósítása során ne legyen számottevően több üzemeltetési feladat.

### 4.1 Operatív szerverek feladata

Ahhoz, hogy az elosztott működést elérjük, kell tehát legalább egy darab operatív szerver, vagy másnéven alkalmazásszerver. Praktikusan az operatív szerverből nem egy



van, amennyiben mégis csak egy operatív szerver lenne a topológiában, úgy nem beszélhetnénk elosztott rendszerről, csupán egy egyszerveres (reverse) proxy szerverről (amennyiben a master szerver nem lát el operatív feladatokat).

Ezeknek a szervereknek a feladatuk, hogy a felhasználók felé kínált szolgáltatás üzemeltetési feladatait ellássák (legyen szó webszerverről, amely egy webshopot szolgál ki, vagy legyen szó akár levelező- vagy tanulmányi rendszerről), más lényegi feladatuk nincs.

Az elosztott rendszernek több megközelítése van az alkalmazásszerverek funkcionalitását tekintve (ld.: 3.1.2 fejezet). Az elosztott működés megvalósítása során a feladatom az volt, hogy az operatív szerverek mind ugyanazzal a funkcionalitással rendelkezzenek, mind ugyanazt a szolgáltatást nyújtsák.

Bármilyen szolgáltatást is nyújtanak a szerverek, kiemelten fontos, hogy a rendszer mindig konzisztens legyen. Ez a gyakorlatban azt jelenti, hogy soha, semmilyen módon nem lehet olyan helyzetben a rendszer, hogy egy adott kérésre az eredmény függ attól, hogy mely szerver szolgálja ki a kérést.

Elenyésző számú olyan szolgáltatás van, amelynek üzemeltetéséhez semmiféle adatbázis nem kell az operatív szervereken, szinte mindig van valamilyen adatbázis, amely tárolhat használati statisztikákat, vagy akár a felhasználók belépési adatait.

Az operatív szerverek között tehát a konzisztencia kulcskérdés, ami az adatbázisok, fájlrendszerek szintjén jelenik meg kihívásként, és általában a rendszer tervezése és megvalósítása során a legfőbb kihívást jelenti. Azt ugyanis, hogy ugyanaz a webalkalmazás fusson minden szerveren általában könnyebb elérni, mint azt, hogy amennyiben bármely szerver módosítja az adatbázisát, a többi szerver adatbázisa konzisztens maradjon. (Arról, hogy miért szükséges minden operatív szerveren külön adatbázist futtatni, valamint az adatbázisok működéséről a 4.3.2 fejezetben lesz szó.)

## **4.2 Master szerver feladata**

A master szerverre hárul minden olyan üzemeltetési feladat, amelyeket nem végeznek az operatív szerverek. Mivel a master szerver jelenti a határt a belső hálózatra csatlakoztatott operatív szerverek, valamint az általában a publikus világhálót jelentő külső hálózat között, ennek helyes konfigurálása rendkívüli prioritással bír.

A kívülről érkező kérések elsőként a master szervert érik el. A master szerver feladata többértű. Egyrészt szükséges, hogy nyilvántartsa az operatív szerverek számát, elérési címeit annak érdekében, hogy a kéréseket el tudja juttatni az alkalmazásszerverekre, másrészt biztosítani kell, hogy a felhasználók elérjék a szolgáltatást, melyet az alkalmazásszerverek nyújtanak. A master szerver feladata, hogy elrejtse az alkalmazásszervereket, azokat a felhasználók ne érhessék el direkt eléréssel.

A master szerver reverse proxy tulajdonságokkal rendelkezik az előbb említettek miatt, és éppen ezért kézenfekvő, hogy a rendszert védő tűzfalat a master szerver jelentse, tehát szükséges a tűzfal helyes konfigurálása.

### 4.3 Alkalmazott technológiák

A rendszer tervezése és megvalósítása során kiemelten fontos szempont volt, hogy a rendszert olyan technológiák felhasználásával hozzam létre, amelyek még ha nem is követik teljes mértékben a bevett szokásokat vagy legtöbbet alkalmazott módszereket, legalább nagy vonalakban hasonlítanak rá.

Egy rendszer készítése során kifejezetten előnyös, ha bevett megoldásokra támaszkodunk, ennek több oka is van. Egyik kifejezetten hasznos ismérve a sokak által használt megoldásoknak, hogy nem véletlen használják sokan, oka van annak, hogy népszerűek. Ok lehet, hogy az adott technológiát pont hasonló feladatokra találták ki, vagy a technológia használatával nagyon leegyszerűsödnek, hatékonyabbá válnak a folyamatok.

A másik előnyös tulajdonsága a *best practice*<sup>8</sup> megoldásoknak az, hogy mivel nagyon népszerűek, egy-egy problémára, hibára vagy nehezen érthető részletre nagyon sok kérdés és sok specifikus, jól használható és jól érthető válasz található online fórumokon.

A rendszer megvalósítása során sok helyen kellene technológiát választani, ám ahelyett, hogy minden technológiát magamtól választottam volna ki, konzulensemtől a lényegi részek technológiáira kaptam javaslatot, hogy konkrétan mit érdemes használnom.

---

<sup>8</sup> Magyar nyelven is gyakran használt kifejezés a bevett megoldásokra.

A továbbiakban bemutatom a főbb elemek technológiáit, azok előnyeit és választásuknak okait.

### 4.3.1 Operációs rendszer

A rendszerben elhelyezkedő szervereket az eddigiekben és a továbbiakban is a már említett két csoportra oszthatjuk (úgy, mint operatív szerverek és mint master szerver), ám operációs rendszerüket tekintve ezzel a megkülönböztetéssel nem élhetünk. Először bemutatom a használt operációs rendszert, majd pontosítom, hogy miért is célszerű ezt használni mindkét esetben.

A konzulensem ajánlása alapján az Linux-alapú Ubuntu rendszert használtam, azon belül az *Ubuntu 18.04.6 LTS* [6] operációs rendszer *server* változatát. Ez az operációs rendszer kiváló szerverek üzemeltetéséhez, neve is erre utal.

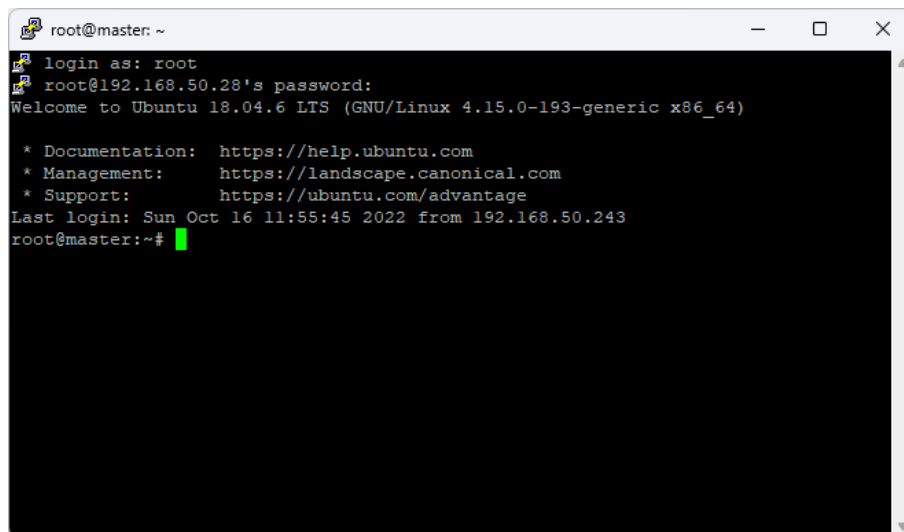
Egyik fő előnye, hogy a népszerű operációs rendszerekhez képest ennek a grafikus felülettel nem rendelkező operációs rendszernek a telepített változata viszonylag kevés helyet foglal: a működő rendszerben a master szerver helyigénye minden telepített alkalmazással és szolgáltatással együtt mintegy ~5GB, ezzel szemben a népszerű Windows 10 mérete is legalább ennek kétszerese telepített alkalmazások nélkül, ám annak az operációs rendszernek nem ugyanaz a célja, mint az általam használt Ubuntu-nak.

Mivel az operációs rendszer nem rendelkezik grafikus felülettel, terminálból lehet elérni. Az elérésre SSH<sup>9</sup> kapcsolatot alkalmazó PUTTY [7] klienst használtam. Ennek használata megkönnyítette a szerverek elérését, a PUTTY termináljának a felhasználóbarát kialakítása is segítségemre volt.

A szervereket, amiket virtuális számítógépekként manifestáltam, az Oracle VirtualBox [8] segítségével futtattam a saját, *Ubuntu 22.04.1 LTS Desktop* operációs rendszerű futtató laptopomon. Jóllehet saját laptopom nem rendelkezik olyan tulajdonságú erőforrásokkal, mint egy bérelhető szerver egy szerverparkban, a célnak teljes mértékben megfelel, és nagyságrendekkel olcsóbb az üzemeltetése.

---

<sup>9</sup> *Secure Shell*, azaz biztonságos héj. [23]



```
root@master: ~
login as: root
root@192.168.50.28's password:
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 4.15.0-193-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
Last login: Sun Oct 16 11:55:45 2022 from 192.168.50.243
root@master:~#
```

8. ábra A PUTTY terminálja

A 8. ábrán a PUTTY terminálablaka látható, ilyen konzolos felületen értem el az összes szervert, amikkel dolgoztam a feladat megoldása során.

(Megjegyzés a 8. ábrához: az ábrán látható, hogy *root* felhasználóként jelentkeztem be. Ez éles környezetben kerülendő, mivel súlyos biztonsági kockázatot jelent. Minekutána a fejlesztés során a rendszer nem kapcsolódott nyilvános hálózatra, csupán privát, belső hálózatra, valamint a feladat *proof of concept*<sup>10</sup> jellege miatt esetemben nem jelentett rizikófaktort, és egyszerűsítette a fejlesztési folyamatokat.)

A Linux-alapú operációs rendszerek szerverek tekintetében széleskörben használt rendszerek, mivel nagyon előnyös az alkalmazásuk nemcsak a méretük, hanem a telepíthető csomagok, valamint azok egyszerű telepíthetősége miatt is. Az APT<sup>11</sup> segítségével számtalan csomagot telepíthetünk egyetlen paranccsal kezdve adatbázisalkalmazásoktól, a webservereken keresztül egészen a szórakoztatási céllal készített csomagokig.

---

<sup>10</sup> Angol kifejezés azon megvalósításokra, amelyeknek célja csupán valamilyen megközelítés, megoldás, módszer vagy ötlet működőképességének bizonyítása, valamint az, hogy az gyakorlati potenciállal rendelkezik. Magyarra fordítva: koncepció bizonyítása, ám ez a magyar kifejezés nem elterjedt.

<sup>11</sup> *Advanced Package Tool*, azaz fejlett csomageszköz. [18]

### 4.3.2 Adatbázisszerver

Ahogy arra a korábbiakban már utaltam, az elosztott rendszer lényegében az adatbázisok elosztott volta miatt lesz elosztott rendszer. Az operatív szervereken ugyanaz a szolgáltatás fog futni, a szerverek külön-külön mind rendelkezni fognak saját adatbázissal.

Két dolog is indokolja azt, hogy minden szerveren külön adatbázisra van szükség. Egyik ok, hogy így minden szerver a saját adatbázisából tud olvasni, és mivel általánosságban kijelenthető, hogy olvasási kérésből több van, mint írásból (gondoljunk bele: minden alkalommal, amikor megnyitjuk vagy frissítjük az oldalt, ki kell olvasni az adatbázisból a tartalmat), ezzel sok kérést tudunk elosztani a szerverek között.

Amennyiben ezeket az adatbázisokat nem kapcsoljuk össze semmilyen módon, úgy csak néhány ugyanolyan operatív szerverünk lenne, melyek mind ugyanazt a szolgáltatást nyújtják, de potenciálisan mindegyik más és más adatokat tartalmazna. Ahhoz, hogy mégis egy elosztott szolgáltatásról beszélhessünk, mely számos szerverből áll, amiknek a segítségével az erőforrásokat diverzifikálni, a teljesítményt pedig növelni tudjuk, össze kell kapcsolni az adatbázisokat oly módon, hogy azok szinkronizálva legyenek.

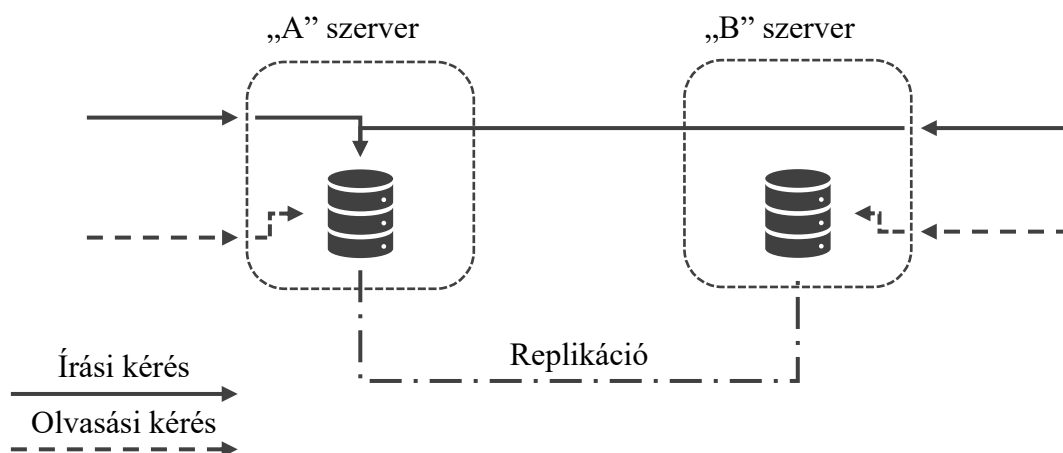
Az adatbázis szerver kiválasztása során a MySQL [9] mellett döntöttem, mivel széleskörben elterjedt szoftver, így az elérhető szakirodalom is bőséges.

A MySQL lehetőséget biztosít ún. replikációs működés megvalósítására, mely lényegében pontosan megegyezik azzal a működéssel, amire az elosztott rendszer épít. A replikációs működés alapja, hogy egy adatbázist egy másik adatbázis alapján hozunk létre, valamint működésüket is összekapcsoljuk.

A MySQL replikációs működése ugyanakkor alapértelmezett működését tekintve csak master-slave viszonylatban képes a replikációs működésre, slave-slave viszonylatban nem. Ez a másik oka annak, hogy minden szerveren el kell helyeznünk egy-egy adatbázist.

Ilyen replikációs működésre sokféle felállást, topológiát vagy scenáriót lehet adni példaként. A legegyszerűbb példa egy két szerverből álló topológia, mely topológiában az egyik szerver („A” szerver) ki van nevezve olyan szervernek, amely szerverbe ír

mindkét szerver, illetve minden szerver a saját adatbázisából olvas. Ahhoz, hogy az adatok mindkét adatbázisban szinkronizálva legyenek, szükség van a replikációs működésre, amely automatikusan szinkronizálja az adatbázisok tartalmát.



**9. ábra A replikációs működés szemléltetése egy kétszerveres topológián**

A 9. ábra azt szemlélteti, hogy egy egyszerű, két szerveres topológián hogyan kezeli le a rendszer az egyes szerverekre beérkező írási és olvasási kéréseket. Míg az „A” szerver mind az írási, mind az olvasási kéréseket a saját adatbázisának címzi, addig a „B” szerver a saját adatbázisából olvas, ám az „A” szerver adatbázisába ír, ahonnan az adatok a replikációs működés segítségével szinkronizálódnak a saját adatbázisába.

Pontosan ez a működés lesz az alapja az elkészítendő rendszernek, ám akkor nem (feltétlen) kettő darab szerver között fog ily módon eloszlni a terhelés, hanem több, esetenként akár tíz szerver között, amely a szerény képességű otthoni infrastruktúrára már megterhelő lehet.

### **4.3.3 Demóalkalmazás**

Az elosztott rendszer, valamint annak automatikus skálázásra alkalmas verziójának működésére a WordPress [10] tartalomkezelő rendszert ajánlotta a konzulensem. A

WordPress egy széleskörben elterjedt blog- és tartalomkezelő rendszer, melynek működése PHP<sup>12</sup>, illetve MySQL-alapú, de bővítményekkel a képessége bővíthető. Esetemben is ez a cél: a WordPress alapműködését a HyperDB<sup>13</sup> nevű bővítménnyel kibővítmem, amelynek segítségével lehetőségem nyílik a rendszert elosztott működésű rendszerré tenni.

A világ szervereinek jelentős része, nagyjából 30%-uk használ WordPresst (2022. november 1-i adat [11]), ami jól mutatja, hogy nagyon sok esetben a WordPress kitűnő megoldás.

## 4.4 Megvalósítás

A megvalósítás során fontos volt, hogy kisebb részfeladatok teljesítésével jussak el arra a pontra, amikor az egész rendszer összeáll. Ennek érdekében a feladatot lépcsőzetesen alfeladatokra bontottam, amelyek önmagukban egyszerűbb feladatok. A feladatok sorrendjüket tekintve úgy lettek meghatározva, hogy a későbbi feladatok mind valamekkora mértékben az előző feladatokra építenek, s a későbbi feladatok igénylik, hogy a korábbi feladatok hibamentesen el legyenek végezve.

A teljes elosztott rendszer megvalósítását ezen feladatok mentén kívánom bemutatni (az automatikus skálázás témakörét az 5. fejezetben részletezem).

A megvalósítás bemutatása során nem kívánok minden lépést teljes mélységben, a fájlmodosítások szintjéig bemutatni, azok megtalálhatók a hivatkozott útmutatókban, helyette a fontosabb, elméleti lépéseket kívánom ismertetni, illetve magyarázattal ellátni.

### 4.4.1 Szerverpéldányok

Az első alfeladat adta magát: létre kellett hoznom néhány szervert, amelyeken elkezdhetem megvalósítani a működést. Első lépésként meghatároztam a szerverek számát, amelyekkel megvalósítottam az elosztott működés részfeladatát: egy operatív és egy master szerverre volt szükségem. A korábbiakban bemutatott elvekkel ellentétben ebben

---

<sup>12</sup> Eredetileg: *Personal Home Page*, napjainkban azonban a *PHP: Hypertext Preprocessor* rövidítése. [22]

<sup>13</sup> A WordPress egyik bővítménye, amely egy adatbáziskezelő bővítmény. [20]

az esetben a master szervert is felkonfigurálom operatív szerverként, hogy látványosabb legyen annak bizonyítása, hogy a master szerver adatbázisa valóban a helyes állapotban van.

Két szerver minimum kell ahhoz, hogy az elosztott működést érdemben használni, tesztelni tudjam, ennél kevesebb szerverrel nem megvalósítható, ennél több pedig tulajdonképpen nehezítette volna a folyamatokat az általuk okozott kezelési többletfeladatokkal.

A korábban már említett VirtualBox szoftver segítségével létrehoztam a két virtuális gépet ugyanolyan hardvertulajdonságokkal. Ezek a tulajdonságok a következők:

- 1 magos CPU
- 1024 MB memória
- 5 GB háttértár
- Ubuntu 18.04.6 LTS (64 bites) operációs rendszer

Minden virtuális gép hálózati adapterét átállítottam *bridged* (azaz áthidalt) adapterre, amely segítségével mindenféle belső hálózat kihagyásával direkt módon egyből a routerre csatlakoztak a virtuális gépek. Ezáltal minden a router hálózatára csatlakoztatott eszköz elérte egymást direkt címezéssel.

A virtuális gépek bekapcsolása után első dolgom volt az operációs rendszer telepítése, majd pedig a szükséges csomagok telepítése, illetve frissítése.

Ezen a ponton rendelkezésemre állt kettő működő, ekkor még teljes mértékben megegyező szerver.

#### **4.4.2 Adatbázisok és a WordPress inicializálása**

Az elterjedt megoldások, alkalmazások egyik nagy előnye, hogy az alapműködés eléréshez szükséges lépések sok helyen le vannak írva, a legtöbb gyakran előforduló hibára megoldásokat javasol a közösség. Éppen ezért a WordPress telepítése nem volt nehéz, jól érthető útmutatót lehet találni az Ubuntu weblapján [12].

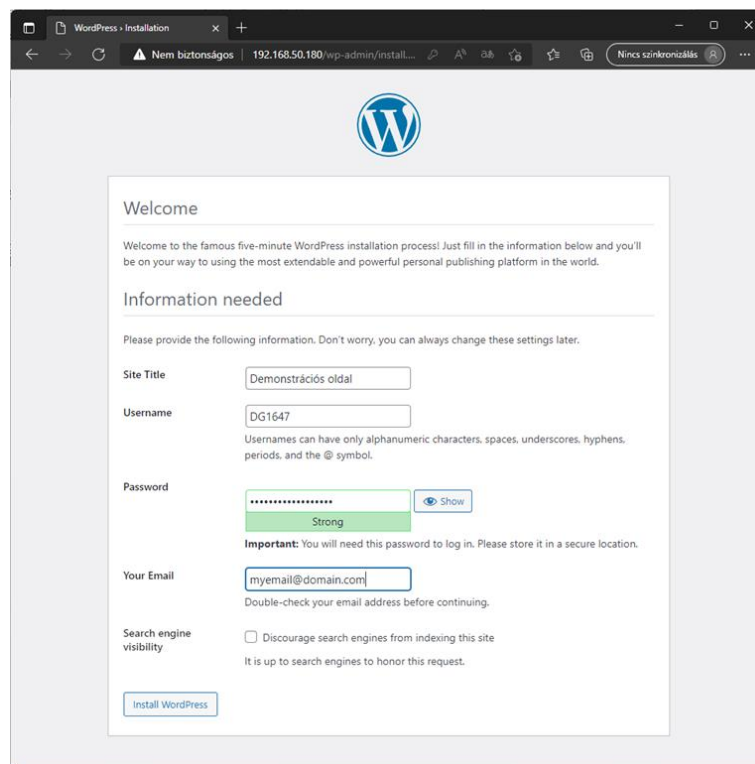
Első körben telepíteni kell a függőségeket (PHP komponensek, adatbázisszerver, valamint egy webszerver, ami a WordPresst futtatja, ez ebben az esetben az Apache2 [13]



volt). A WordPress fájljainak telepítése után fel kell konfigurálni, hogy a webszerver a kívánt tartalmat jelenítse meg.

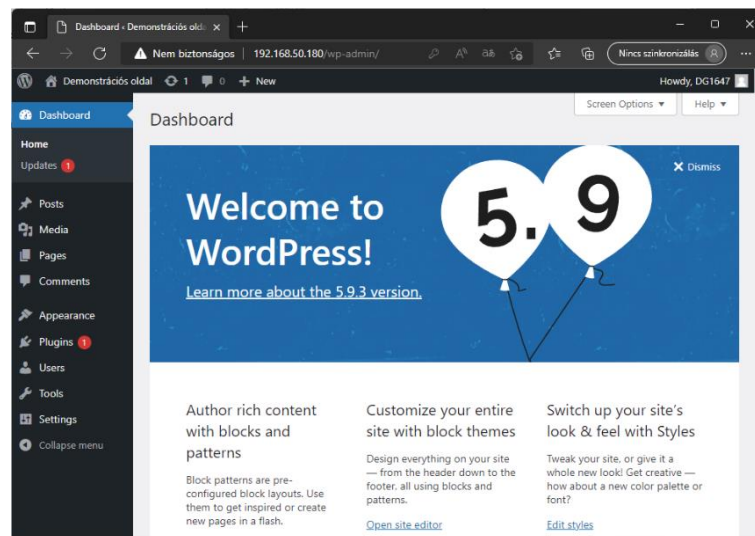
Az adatbázisszerverben először létre kell hozni a szükséges adatbázist és felhasználót, amely felhasználó jelen esetben nem a rendszer felhasználójának vagy üzemeltetőjének példánya, hanem egy olyan felhasználó, amit a WordPress motorja használ a működése során. A megfelelő jogok kiosztása után össze lehet kapcsolni a WordPresst és az adatbázist.

A WordPresst, miután a szerveren mindent felkonfiguráltam, inicializálni kellett a böngészőből elérhető kezdőoldalán is, ezen lépések segítségével állítottam be az oldal címét, valamint az admin felhasználó adatait is.



**10. ábra A WordPress konfigurációs oldala**

Miután minkét szerveren elvégeztük ezeket a lépéseket, be vannak üzemelve az egyes szerverek, azok működését tesztelni lehet (a szerverek ekkor még nincsenek összekötve, semmiféle kapcsolat nincs köztük).



11. ábra A WordPress admin oldala

### 4.4.3 Adatbázisok összekapcsolása

Az adatbázisok replikációs működésének beüzemelésére is léteznek jól használható útmutatók (ebben az esetben sokkal inkább valamilyen blogon a témában járatos szakembertől származó útmutatóra kell gondolni, mintsem a termék hivatalos oldaláról származó leírásra), melyek segítségével tisztább képet kaphatunk az elosztott adatbázisokról, főként azok működéséről.

Olyan útmutatót [14] választottam, amely leginkább hasonlít az általam elvégzendő feladathoz: WordPress tartalomkezelő és MySQL adatbázisszerver beüzemelését mutatja be (nehéz is lett volna más útmutatót választanom, ugyanis más hasonló tartalmú és elérhető útmutatót nem találtam).

A feladat tehát egy master szerver és egy operatív szerver összekapcsolása (ebben a feladatban mivel a master szerver lényegében egy kitüntetett operatív szerver, a master szerver és az operatív szerver elnevezések valamelyest torzítanak a valóságon).

Ez az alfeladat két részből áll: a master szerver adatbázisának felkonfigurálása, valamint az operatív szerver felkonfigurálása. Fontos, hogy a módosítások elvégzése közben ne használják felhasználók egyik szerveret sem, mert inkonzisztenciát okozhatnak az általuk elvégzett műveletek.

#### 4.4.3.1 Master szerveren elvégzendő lépések

Az első módosításokat tehát a master szerveren kellett elvégezni: a megfelelő konfigurációs fájlban az IP-cím kötését (*bind-address*) kellett beállítani a master szerver IP-

címére, valamint az adatbázisszerver azonosítóját (*server-id*) egy eddig nem használt azonosítóra állítani (mivel eddig nem volt egy szerver sem, nincs lefoglalva egyik azonosító sem, az egyszerűség kedvéért az azonosítója *1* lett).

A replikációs működést lényegében log fájlok működtetik: ezek azok a fájlok, ahova az adatbázisszerver a történéseket naplózza, innen fogják az operatív szerverek kiolvasni, hogy milyen események, műveletek történtek. Értelemszerűen ezen fájlok helyét is be kell állítani a konfigurációs fájlban.

A master szerveren működő adatbázisszerverben létre kell hozni egy olyan felhasználót, amit az operatív szerver működése során igénybe tud venni, majd ennek a felhasználónak jogosultságot kell adni, hogy a korábban már létrehozott adatbázison műveleteket tudjon végrehajtani.

Mielőtt érvénybe léptetnénk a módosításokat, meg kell adni a konfigurációs fájlban annak az adatbázisnak a nevét, amelyet szeretnénk megosztani. Ez nem magától értetődő lépés, mert egy adatbázisszerver adott esetben több alkalmazásnak nyújt szolgáltatást, nem feltétlen csak egy adatbázis található benne (fontos, hogy az adatbázis, illetve az adatbázisszerver két külön fogalom).

Miután minden fontos változtatást elvégeztünk, a módosításokat egy adatbázisszerver-újraindítással léptethetjük érvénybe. Ezután következik a működés szempontjából egyik legkritikusabb lépés: ahhoz, hogy a szerverek a megfelelő konzisztenciát tartani tudják egymáshoz viszonyítva, fontos, hogy a korábban említett naplófájlok (log fájlok) közül a megfelelő fájlban a megfelelő pozícióból kezdjenek olvasni az adatbázispéldányok. Ha ezt a két információt (fájl-pozíció kettős) nem állítjuk be megfelelően, illetve nem tartjuk naprakészen, adatbázisműveletek után nem tudja jó helyről kiolvasni az eseményeket az operatív szerver.

Tehát a master szerver adatbázisszerverében meg kell jeleníteni és fel kell jegyezni ezt a két információt.

#### **4.4.3.2 Operatív szerveren elvégzendő lépések**

Az operatív szerveren az első feladat, hogy a master szerveren található WordPresshez tartozó adatbázis állapotát átmásoljuk az operatív szerverre. Ez a kiinduló állapotok egyezését biztosítja.

A master szerveren elvégzett lépésekhez hasonlóan az operatív szerveren is el kell végezni a szerver azonosítójának módosítását (ebben az esetben is egyedinek kell lennie, ezért: *server-id*: 2), a naplófájlok helyének beállítását, valamint az operatív szerver esetében a *relay-log* helyét is meg kell adni.

Az adatbázisszerveren be kell állítani a master szerveret mint master: meg kell adni a master IP-címét, a master szerveren futó adatbázisszerveren létrehozott felhasználó nevét és jelszavát, illetve az aktuális log fájl nevét és a pozíciót.

Az adatbázisszerver újraindításán kívül jelen esetben a működés elindítása érdekében ki kell adni az operatív szerver adatbázisszerverében a *START SLAVE*; utasítást.

#### 4.4.3.3 HyperDB csomag

A korábban hivatkozott leírás ([14]) végén található egy rész, amely röviden bemutatja a WordPress konfigurálását a HyperDB csomag segítségével. Ezen lépések során a WordPress konfigurációs fájljában kell módosítani például a kapcsolatok maximális számát, vagy éppen az adatbázisok adatait kell megadni.

Miután ezen lépéseket is elvégeztük, a rendszer elméletben kész az elosztott működésre.

### 4.5 A megvalósítás során jelentkezett problémák

A két szerverből álló, elosztott működésű WordPresst mint szolgáltatást nyújtó rendszer elméleti lépésekben a 4.4 fejezetben bemutatott módon elkészíthető. Az elvárás az, hogy a szükséges lépések elvégzésével a rendszer hiba nélkül működjék.

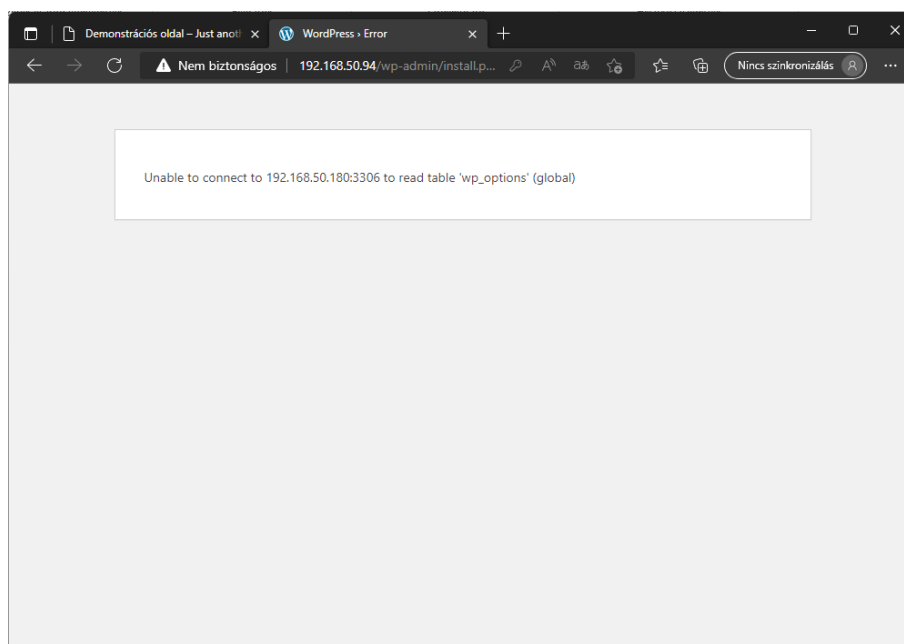
Ezzel szemben a valóság az, hogy a lépések elvégzése után egyre csak különböző hibaüzenetek jelentek meg a rendszer különböző részeinek kimenetén. Volt, hogy az adatbázisszerverek hibás felkonfigurálása miatt kaptam hibaüzenetet, de volt, hogy a tűzfalak nem megfelelő beállítása miatt volt hiba. Ezeket a hibákat rendre sikerült orvosolni, ám volt kevésbé megoldható hiba.

A megvalósítás során lényegében kétféle problémacsoporttal kellett megküzdenem, amelyek között nagyobb kapcsolat van, mint elsőre hinnénk.

Az egyik nagy probléma a megfelelő szakirodalom hiánya. Részletes, precíz és kellően hiteles szakirodalom hiányában nagyságrendekkel nehezebb egy rendszer megvalósítása. Jelen esetben, ahogyan már említettem, a hivatkozott útmutatón kívül lényegében nem találtam releváns másik útmutatót.

A szakirodalom hiánya a hibák elhárításánál is jelentkezett: a végső hiba megoldására, amit konzulensem segítségével sem tudtunk megoldani, nem találtunk olyan cikket vagy bejegyzést, amely segített volna.

A végső hiba (az implementálás során kapott hibák jelentik a másik nagy problémacsoportot) a helyesnek vélt felkonfigurálást követően a WordPress kezdőlapján a 12. ábrán látható módon fogadott.



**12. ábra Hibaüzenet aktivált HyperDB csomaggal**

A hibaüzenet a következő: *Unable to connect to 192.168.50.180:3306 to read table 'wp\_options' (global)*, azaz *Nem sikerült csatlakozni a 192.168.50.180:3306 címhez a 'wp\_options' tábla olvasása céljából (globális)*. A hibaüzenetben olvasható cím a master server IP-címe, a 3306 portszám pedig az adatbázisszerver portja.

Ezt a hibát sem tűzfalbeállításokkal, sem az adatbázishoz kapcsolódó beállítások módosításával nem sikerült elhárítani. A hiba ellenére a 192.168.50.94 IP-című operatív szerver példányról lehetett csatlakozni a 192.168.50.180 IP-című master adatbázishoz a nyitott 3306-os porton keresztül, valamint a táblák is elérhetőek voltak.

## 4.6 A működő rendszer megvalósítása

Hosszas hibaelhárítást célzó sikertelen próbálkozások után konzulensemmel egyeztetve új demonstrációs alkalmazás használata mellett döntöttünk. A konzulensem készített egy Üzenőfal webalkalmazást, amely primitívebb funkcionalitással rendelkezik, mint a WordPress, ám annál egyszerűbb is.

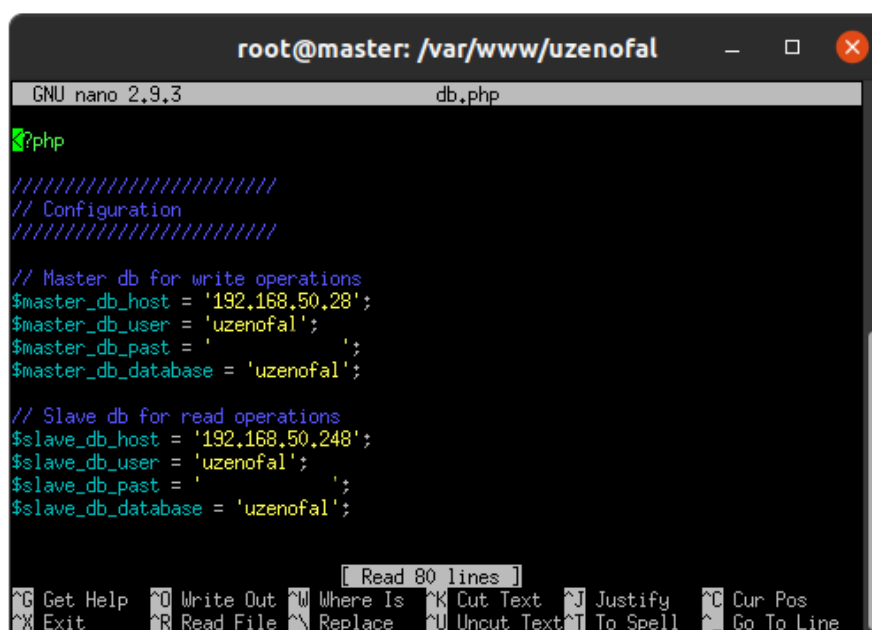
### 4.6.1 Az Üzenőfal

Az Üzenőfal üzenetek kezelésére alkalmas: létre lehet hozni üzenetet, amelyet minden az oldalra látogató felhasználó elér, illetve természetesen olvasni is lehet korábban megosztott üzeneteket.



13. ábra Az Üzenőfal kezdőoldala

Az egyszerűbb funkcionalitással rendelkező alkalmazás beüzemelése is egyszerűbb: egy konfigurációs fájlban be kell állítani, hogy az adott szerver melyik adatbázisból olvasson és melyikbe írjon. Ezt az adatbázisszervereket futtató szerverek IP-címének megadásával könnyen meg lehet tenni.

A screenshot of a terminal window titled 'root@master: /var/www/uzenofal'. The window shows the nano 2.9.3 text editor editing a file named 'db.php'. The code is a PHP configuration file for database connections. It includes comments for configuration and sections for master and slave database connections. The master database is for write operations, and the slave database is for read operations. The configuration includes host, user, password, and database name for both. The terminal window has a status bar at the bottom with various nano editor commands like 'Get Help', 'Write Out', 'Where Is', 'Cut Text', 'Justify', 'Cur Pos', 'Exit', 'Read File', 'Replace', 'Uncut Text', 'To Spell', and 'Go To Line'.

```
root@master: /var/www/uzenofal
GNU nano 2.9.3 db.php

?php
////////////////////////////////
// Configuration
////////////////////////////////

// Master db for write operations
$master_db_host = '192.168.50.28';
$master_db_user = 'uzenofal';
$master_db_past = ' ';
$master_db_database = 'uzenofal';

// Slave db for read operations
$slave_db_host = '192.168.50.248';
$slave_db_user = 'uzenofal';
$slave_db_past = ' ';
$slave_db_database = 'uzenofal';

[ Read 80 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^M Replace   ^U Uncut Text ^T To Spell  ^_ Go To Line
```

14. ábra Az Üzenőfal konfigurációs fájljának részlete

Miután minden szerveren elhelyeztem az Üzenőfal fájljait, beállítottam, hogy a webszervereken keresztül az Üzenőfalat érje el a felhasználó.

Minden szerveren ugyanúgy, az eddig nem említett *nginx* [15] webszerver szolgáltatás segítségével publikáltam a weboldal fájljait. Az *nginx* hasonlóan az *Apache2*-höz egy széles spektrumú alkalmazási területtel bíró webszerver, amit szintén lehet használni (reverse) proxy szerverként is, de még sok-sok egyéb feladatot el tud látni.

Az Üzenőfállal már tudtam demonstrálni az elosztott működést: miután mindkét szerverre feltelepítettem és inicializáltam, az egyik szerver weboldalán posztolt üzenet azonnal megjelent a másik szerver weboldalán és fordítva.

#### 4.6.2 Az elosztott rendszer terheléselosztó képessége

Még mielőtt áttérnék a skálázás témakörére, szeretném bemutatni a rendszer terheléselosztó képességét. A rendszer ezen képességének bemutatása eddig csak az elméleti megközelítésig terjedt, alkalmazott technológiákról, illetve a megvalósításáról nem esett szó.

A terheléselosztás témaköre azért nem kapott a szakdolgozatban külön fejezetet, mert annak megvalósítása az alkalmazott technológia segítségével meglehetősen könnyű és gyors részfeladat volt. Ez az alfejezet ezen technológia és a terheléselosztás megvalósításának folyamatát hivatott bemutatni.

Ahogy korábban említettem, az nginx rengetegféle feladatot el tud látni: képes proxy szerverként, egyszerű webszerverként, de akár levelezőszerverként is működni, és ami számomra a legfontosabb: lehetőségünk van terheléelosztási feladatokat is kiosztani számára.

A webszerver konfigurációs fájljában lehetőségünk van megadni szerverelérési címeket, amelyek között valamilyen logika alapján kiválasztja, hogy a soron következő beérkező kérést mely szervernek továbbítsa. Ezen döntési logikát a szolgáltatás biztosítja, a konfigurálás során elég csak a módszert meghatározni.

Az nginx háromféle algoritmus [16] közül enged választani (az algoritmusokat az angol terminológia szerinti nevük használatával mutatom be, ugyanis a mögöttük álló fogalmakat magyar környezetben is többnyire az angol nevükön hívjuk. Ezek az algoritmusok a következők:

- round-robin
- least connected
- IP-hash

A **round-robin** algoritmus alapja a megadott szerverek valamilyen sorrendezéséből alkotott kör. A konfiguráció során megadott szerverek sorrendjét eltárolja, majd ez alapján a bemenettől függetlenül sorban adja a kimenetre a soron következő szervert, ám miután az utolsó szerver is megjelent kimenetként, a következő kimenet az első szerver lesz, így egy végtelen kört alkot a véges számú szerverből. Az  $\{A; B; C\}$  szerverhármásra a kimenet tehát a következő lesz:  $A; B; C; A; B; C; A...$

Ez az algoritmus elég primitív olyan értelemben, hogy semmilyen tekintettel nincs a szerverek terheltségére, azokat nem ellenőrzi és nem is veszi figyelembe a döntés során, de cserébe nagyon gyorsan dönt.

A **least-connected** (legkevesebb kapcsolattal rendelkező) algoritmus azt a szervert adja kimenetként, amelyik szerver a legkevesebb aktív kapcsolattal rendelkezik. Ez az algoritmus komplexebb, figyelembe veszi a szerverek állapotát, ám itt is előfordulhatnak anomáliák (az aktív kapcsolatok száma nem egyenértékű információ a valós terheltséggel, lehet egy szerver sok aktív kapcsolattal kevésbé terhelt, valamint fordítva).



A **IP-hash** algoritmus kimenete az előzőekkel ellentétben a bemeneti kérés függvénye. Figyelembe veszi, hogy az adott kérés melyik IP tartományból érkezett, és az adott tartományhoz rendelt szervert adja kimenetként.

Az előbb felsorolt három algoritmus nem kizárólagos, lehet többszintű topológiákat alkotni, ahol például először egy IP-hash algoritmus alapján választunk szervercsoportot, azon belül pedig round-robin alapján.

Én a rendszerem megvalósítása során az alapértelmezett round-robin ütemezőt alkalmaztam, mivel célom volt, hogy véges időn belül biztosan a teljes szerverhalmaz megjelenjen a kimenetén.

## 5 Automatikus skálázás

Korunk szoftvertervezői abban a helyzetben vannak, hogy egy szoftver megtervezése során modern tervezési elvek hada elérhető számukra, amelyek segítenek a tervezés folyamatában, s amelyek mindegyike valamilyen hátrányt vagy nehézséget okozó jelenséget kíván megelőzni. A modern elvek segítségével olyan szoftverrendszerek hozhatók létre, amelyek tervezésükből fakadóan képesek arra, hogy hosszútávon is jól kezeljék a bővítést, módosítást. Ezen elvek figyelembevételével hosszútávon is könnyen karbantartható és fejleszthető rendszereket lehet létrehozni.

Kifejezett célom volt, hogy ezt a komplexitásban az elosztott rendszer létrehozásához hasonlítható feladatot szintén kisebb részfeladatokra bontsam, de ezesetben nem csak a feladat egyszeri elkészítésére terjedt ki ez a döntés: a rendszert is ilyen elvek mentén terveztem meg.

Az automatikus skálázás alfeladat célja, hogy *„Szükség van egy új szerverre a terhelés méretei miatt”* megállapításból létrejöjjön egy új szerver, az bele legyen integrálva a rendszerbe és a hozzá rendelt beérkező kéréseket elkezdje feldolgozni, valamint az *„Egy szervert le lehet állítani, mert csökkent a terhelés”* megállapítás hatására egy szerver befejezze tevékenységét, leálljon és a rendszer kitörölje a nyilvántartásból, annak tovább ne küldjön kéréseket.

Nagyon fontosnak tartottam a tervezés során, hogy hasonlóan egy modern szoftverhez, az én esetemben is minden felelősség szét legyen választva és az egyes egységek könnyen cserélhetők legyenek. Ezen tervezési elvemmel összhangban a feladatot felelőségek mentén feldaraboltam: nyilvánvaló, hogy a feladat nem mindegyik része tartozik egy operatív szerverre vagy épp a master szerverre, más feladatokat kell az egyes szervereken elvégezni.

A feladatokat tehát felbontottam a részek elvégzésének helye szerint: egy adott részfeladat elvégzése szempontjából három hely jöhet szóba: a futtató környezet, a master szerver, valamint az érintett operatív szerver. Minden feladat, ami egy adott típusú szerver halmazába került, jó helyre került felelősség szempontjából, valamint minden halmaz tartalmazza az adott típushoz tartozó összes szükséges feladatot.

A következőkben szeretném részletezni, hogy az egyes szervereknek mik a feladataik egy szerver indítása vagy leállítása során, bemutatni az alkalmazott technológiát és ismertetni a feladatok elvégzésének menetét.

## 5.1 A megoldási módszerek

A tervezés során többféle technológia, megközelítési módszer állt rendelkezésemre, amelyek kisebb-nagyobb mértékben eltértek egymástól. A módszereknek két nagyobb, jól elkülönülő ágát határoztam meg: az egyik csoportba tartoztak az alkalmazás- vagy programalapú megközelítések, a másik csoportba a szkript- és szolgáltatásalapú megoldások.

A programalapú megközelítések alapja egy valamilyen programozási nyelv (C#, Java, Python stb.) segítségével létrehozott program, amely az elvégzendő feladatokat vagy önmaga elvégzi, vagy a különböző típusú szervereken elhelyezkedő kliensalkalmazásoknak delegálja.

A másik megközelítési mód alapját a bash szkriptek jelentették. Ennek a megközelítésnek a lényege az, hogy a feladat megoldása során elvégzendő lépéseket bash szkriptek segítségével automatizáljuk. A programalapú megoldások során is hasonló az automatizálás, csak abban az esetben egy indirekciót figyelhetünk meg meghatározott lépésre irányuló utasítás kiadása, illetve az operációs rendszeren végrehajtott lépés között. A bash szkriptek pont azokat az utasításokat tartalmazzák, melyeket kézzel is elvégeznénk.

```
#Setting IP's
masterIP=$(cat /uzenofal/masterIP)
slaveIP=$(hostname -I | awk '{gsub(/^ +| +$/, "")}' {print $0}')
slave_name=$(hostname -I | awk '{gsub(/^ +| +$/, "")}' {print $0}' | awk -F'.' '{print $4}')
echo "[LOG] IP addresses and names defined, MASTER IP: [${masterIP}] SLAVE IP: [${slaveIP}] SLAVE NAME: [slave_${slave_name}]"
```

15. ábra Részlet a *setup.sh* inicializáló bash szkriptből

A feladat megoldásához bash szkripteket és az Ubuntu operációs rendszereken (és általában minden Linux disztribúción) elérhető szolgáltatásokat (*service*) használtam. A szolgáltatások olyan programok, amik a háttérben futnak, így egyszeri elindításuktól kezdve addig futnak, amíg le nem állítjuk. Pontosán ilyen szolgáltatások teszik lehetővé,

hogy az elosztott rendszer által nyújtott szolgáltatás működése során az automatikus skálázás motorja minden időpillanatban figyelje a terhelést és döntéseket hozzon.

Az Ubuntu szolgáltatások segítségével tehát jószerivel éjjel-nappal, minden pillanatban tette kész skálázó szolgáltatást, motort tudunk üzemeltetni, ami a szkriptek segítségével be is tud avatkozni a rendszer működésébe. Ezt választottam megoldási módszernek, a következő alfejezetekben szeretném kifejteni ennek tervezését és megvalósítását.

## **5.2 Felelősségek és feladatok**

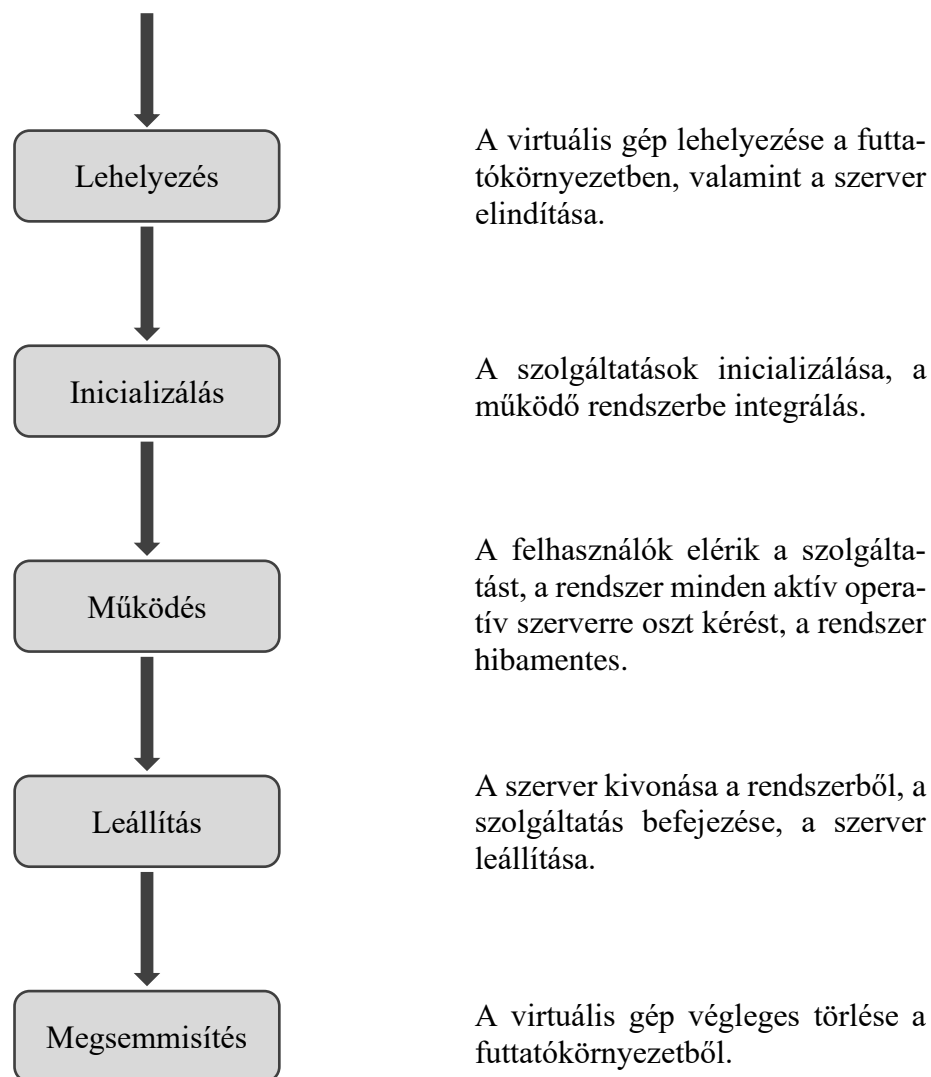
Ahogy az elosztott rendszer esetében is, úgy jelen esetben is az elvégzendő teljes feladatot alfeladatokra bontottam. A feladatot közel elemi egységekre bontottam oly módon, hogy azok szintén egymásra épüljenek. Először az egyszerűbb, alacsonyabb szintű feladatokkal kezdtem, majd onnan haladtam az általánosítás, magasabb szint felé: mintha kis kockákból, téglákból építenék össze egy nagy kockát.

A feladat felbontása nem csak a feladat elvégzésére jellemző: a működő rendszerben az egyes modulok felelősségek szerint is szét vannak választva. A működő rendszerben fellelhető modulok minden esetben egyetlen szerver felelősségi körébe tartoznak. Megvizsgáltam az alfeladatokat olyan szempontból, hogy azt kinek, pontosabban melyik típusú szervernek a felelőssége elvégezni, a feladatok így módon elkülönültek egymástól.

A szeparációt követően tehát elkülöníthetőek a feladatok, melyeket az egyes szervereknek kell elvégezniük. Ezeket a felelősségi köröket és a hozzájuk tartozó feladatok megoldását szeretném bemutatni a fejezet következő részében.

### **5.2.1 Operatív szerver**

Mielőtt az operatív szerver automatikus skálázási feladatait bemutatnám, szeretném ismertetni az operatív szerverek életciklus-modelljét, mivel ennek tisztázása nagyban segíti az elvégzendő feladatok meghatározását és helyes szeparálását. Az életciklus értelmezése után a felmerülő feladatok felelősség szerinti felosztása is egyszerűbb feladat.



**16. ábra Az operatív szerver életciklusmodellje**

Az operatív szerver „élete” ott kezdődik, hogy igény jelentkezik egy új szerverre, ezért a futtatókörnyezetben egy virtuális gép lehelyezésre kerül. A lehelyezés után a szerver elindul, majd az operációs rendszer bootolása után lehetőség nyílik az inicializálásra, a szolgáltatások üzembe helyezésére. Működés közben csupán a felhasználók felé nyújtott szolgáltatás üzemeltetése a feladat. A szerver leállítását célzó parancs hatására a rendszerből ki kell vonni a szervert, annak nem szabad több kérést küldeni, valamint a kivonás után le kell állítani, majd megsemmisíteni a virtuális gépet.

Azokat a feladatokat, amelyeket az operatív szervernek kell elvégeznie, ezen állapotokból lehet meghatározni. Nyilvánvaló, hogy az itt elnagyoltan említett feladatok közül nem mindegyik az operatív szerver feladata, vagy legalábbis nem teljes mértékben.

Az életciklus alapján az első feladat az indulás utáni inicializálás. A virtuális gép egy előre valamilyen szinten inicializált virtuális gép alapján van létrehozva, alapja a mintaszerverből létrehozott képfájl. Az így létrehozott szerver már rendelkezik a legfontosabb fájlokkal és telepítve vannak rá a fontos szolgáltatások, azokat csak inicializálni kell, a fontos adatokkal ellátni. A feladat elvégzése és a működés során a master szervernek fix IP-címe volt, hogy az inicializálás során egyszerűbb legyen a kommunikáció.

#### **5.2.1.1 Inicializálás**

Az inicializálás lépéseit egyetlen, *setup.sh* nevű bash szkript fájlban gyűjtöttem és állítottam össze. Az itt összegyűjtött lépéseket szeretném felvázolni.

Első lépés, hogy az inicializáláshoz szükséges adatokat tartalmazó fájlokat a master szerverről azok frissítése után lemásoljuk. Amint megvannak a fontos adatok, el lehet kezdeni az inicializálást.

Az Üzenőfal fájljait kicsomagolás után a webszerver által meghatározott mappába kell másolni, hogy elérhetővé tudja tenni az oldalt. Az adatbázisszervert a korábban bemutatott lépések alapján a szkript felkonfigurálja, majd az adatbázisszerver (újra)indításával élesíti annak működését.

Végül az inicializálások után az operatív szerver jelzi a master szerver felé, hogy készen áll az integrációra.

#### **5.2.1.2 Működés és leállítás**

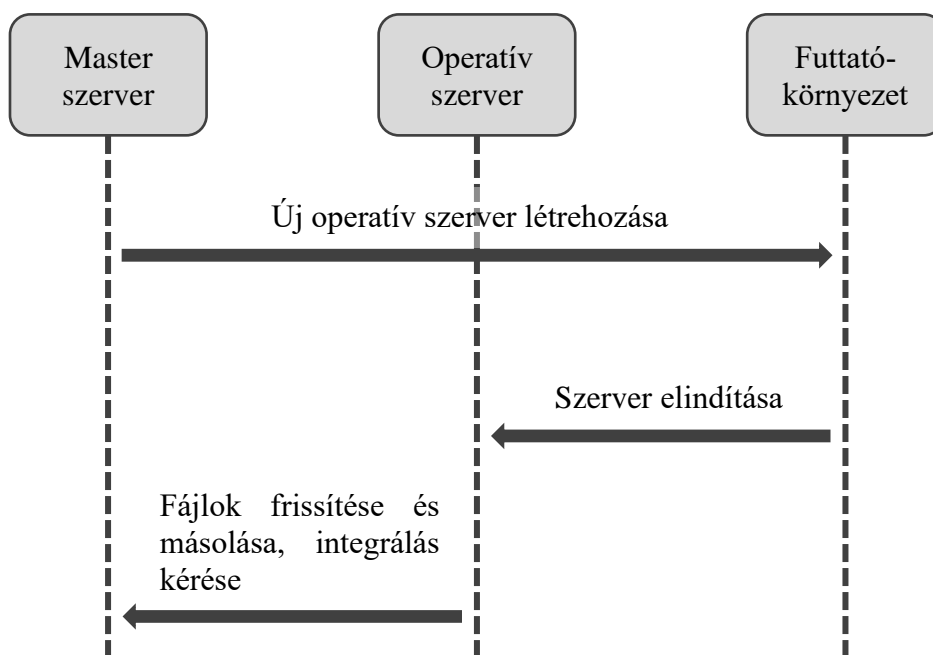
Működés során az automatikus skálázás szempontjából nincs feladata az operatív szervernek, csupán végeznie kell a dolgát. A terhelés meghatározását az 5.3 fejezetben mutatom be.

A leállítás során tulajdonképpen szintén nincsen dolga az operatív szervernek. Dolga lenne az alkalmazások megfelelő leállítása, illetve az adatmentés az adatvesztés elkerülésének érdekében, de az én esetemben adatvesztésről nem beszélünk, hiszen minden adat a master szerveren is megtalálható, illetve leállítás után mindenképp megsemmisül a szerver.

## 5.2.2 Master szerver

A master szerver feladatait két csoportra bonthatjuk: a szervernek egyrésről feladata üzemeltetni az elosztott rendszert, valamint feladata monitorozni és felügyelni a szerverek működése során jelentkező terhelést, az alapján döntést hozni, valamint a döntés eredményét érvényre juttatni. Ezen feladatok közül jelen alfejezet keretei közé csak a döntés érvényre juttatása tartozik, ennek megvalósítását fejtem ki. A terhelést, annak monitorozását és a terhelés függvényében történő döntést az 5.3 fejezetben részletezem.

A master szerver konkrét feladatainak meghatározásához először bemutatom a szerverek között fellépő kapcsolatok sorrendiségét, amely segítségével tisztán láthatóvá válnak a master szerverre vonatkozó feladatok.



17. ábra A szerverek között fellépő kapcsolatok operatív szerver indításakor

A master szerver első feladata, hogy a létrehozást célzó döntés hatására jelezze a futtatókörnyezet számára, hogy a terhelés mértéke miatt szükség van egy új szerverre. Ezt a hívást csak úgy, mint a többi, úgy végzem el, hogy SSH kapcsolat segítségével a célszerverre bejelentkezek (erre lehetőség van szkriptek segítségével is, szkriptből hívva is működik), majd ott lefuttatom az adott feladatra írt szkriptet.

A futtatókörnyezeten indított szkript elindulásával a master szervernek a kezdeti feladatai véget érnek, a következő feladatot az újonnan indult operatív szerver adja.

Amikor az operatív szerver időben odaér, hogy a master szerverről átmásolja a szükséges fájlokat, jelez a master szervernek, hogy ez a feladat most jött el. Ennek a jelzésnek a hatására a master szervernek frissítenie kell azt az információpárost, ami a replikációhoz szükséges log fájl nevét és az abban aktuális pozíciót tartalmazza, valamint az adatbázis aktuális tartalmát el kell mentenie egy fájlba, amit az operatív szerver majd be tud tölteni inicializálása során.

A fájlok frissítése után az operatív szerver jelezni fog a master szervernek, hogy készen áll a rendszerbe integrálásra, így a master szervernek nyilvántartásba kell vennie, valamint a master szerveren futó webszerver terheléselosztó moduljához hozzá kell adnia, így onnantól kezdve annak is továbbít kéréseket.

### **5.2.3 Futtatókörnyezet**

A futtatókörnyezetnek viszonylag egyszerűnek tűnő feladata van: amikor szükség van egy szerverre, akkor lehelyez és elindít egyet, amikor feleslegessé válik egy szerver, akkor leállít és megsemmisít egyet.

Ezt a működést a VirtualBox segítségével nem nehéz megoldani: a szoftver ugyanis támogatja a kódból történő irányítást, lehetőséget ad arra, hogy szkriptből indítsunk vagy éppen állítsunk le virtuális gépet.

Indítás szempontjából meg kell különböztetni a VirtualBox két különböző parancsát: a gép importálását és a gép indítását. Előbbi parancs lehelyezi a virtuális gépet, létrehozza a szükséges fájlokat és indításra kész állapotba állítja, ám nem indítja el. Utóbbi egy már lehelyezett gépet indít el.

A két parancs között fel lehet állítani sorrendiséget: ahhoz, hogy el tudjunk indítani egy virtuális gépet, először le kell helyezni azt. A két parancs időigényének különbsége sem elhanyagolható: az importálás jóval több időt vesz igénybe, mint a virtuális gép elindítása (~5-10-szeres az eltérés).

A virtuális gép importálásának időigénye nem elhanyagolható, mivel olykor 1 percre is tarthat, ám ez nyilván függ a futtatókörnyezet tulajdonságaitól is. Az viszont a



rendszerrel szemben támasztott elvárásoknak nem felel meg, hogy magas terhelés esetén egy új szerver indulására akár perceket is kelljen várni.

Ennek kiküszöbölésére triviális megoldás lenne ezt az időt lerövidíteni, a mögöttes feladatot leegyszerűsíteni, felgyorsítani. Ez a lehetőség nyilvánvalóan nem megvalósítható, minekutána nem saját megoldásról van szó.

A másik megoldási lehetőség az, hogy ezt az időigényes feladatot eltoljuk időben, és nem akkor hajtjuk végre, amikor várunk a virtuális gép elindítására, hanem olyan időben, amikor nincs hatással a várakozás a rendszer működésére.

Több lehetőségünk is van arra, hogy időben máskor végezzük el az importálást: ezek a lehetőségek két csoportra oszthatók.

Az egyik csoportba azok tartoznak, amikor virtuális gép indítása során szeretnénk eltolni az importálást, tehát olyankor, amikor az átlagos terhelés monoton nő. A másik csoportba tartoznak azok a lehetőségek, amikor az importálást az átlagos terhelés lefelé szálló ágában toljuk el.

A kettő megközelítés közül a felszálló ágat választottam, mert akkor várhatóan tovább nő a terhelés, de legalábbis nem csökken hirtelen nagy valószínűséggel, így ekkor kifizetődőbb importálni egy virtuális gépet, mint olyankor, amikor várhatóan további szervereket állítunk le.

Így a végleges megoldás az lett, hogy az  $N$ -edik szerver importálását az  $N-1$ -edik szerver indítása után, annak bootolása közben hajtjuk végre.

Új szerver indítása során tehát először elindítjuk a korábban már importált szervert, majd rögtön importáljuk a következőt, hogy kész legyen az indulásra.

Valamelyest pazarló a szerver leállításának procedúrája, ám ezt a pazarlást megengedtem a szebb megoldás és az átláthatóság érdekében. A pazarlás alapját a virtuális gépek nyilvántartása képezi a VirtualBox részéről: minden importált gépet (természetesen a futó, és a leállított, de nem törölt gépeket is beleértve) egyedi néven tárol. A gépeknek az egyszerűség kedvéért egyedi azonosítóként 1-től kezdve növekvő sorrendben egész számokat adok, valamint egy gép megszüntetésekor az adott gép azonosítóját felszabadítom. Ily módon a 3 darab futó szerver esetén a foglalt azonosítók halmaza a kö-

vetkező:  $\{1; 2; 3\}$ . Egy gép megszüntetésekor a legnagyobb azonosítóval rendelkező gépet állítom le és törlöm, így amennyiben az előző 3 darab futó szerverrel rendelkező rendszerből kivonunk és megsemmisítünk egyet, a foglalt azonosítók halmaza  $\{1; 2\}$  lesz.

Minden esetben pontosan 1 darab importált szerver található, amelynek azonosítója az utoljára kiadott (és egyben a legnagyobb értékű) azonosító után következő azonosító (3 darab futó szerver esetén az importált gép azonosítója a 4 lesz).

A pazarlást pedig szerver megsemmisítésekor tapasztalhatjuk: amennyiben egy szervert megsemmisítünk és felszabadítjuk az általa foglalt azonosítót, a foglalt azonosítók között érték szerinti sorrendben lesz egy kivétel, ami nincs lefoglalva. Az ilyen kivételek elkerülése érdekében egy szerver megsemmisítésekor az importált gépet is megsemmisítem, majd helyes azonosítóval újra létrehozom.

Ezzel a felesleges lépéssel (amit kiválthatnánk egy átnevezéssel) szeretném ugyanakkor demonstrálni, hogy az importálással járó időigényes feladatot az átlagos terhelés leszálló ágában is, a terhelés monoton csökkenése közben is el lehet tolni.

## 5.3 A terhelésről

A rendszerrel szemben az egyik legnagyobb elvárás, hogy a terhelés mértékének függvényében a rendszer automatikusan skálázódjon. Az előző fejezetekben bemutattam, hogy miként áll össze az elosztott rendszer, milyen szerverek alkotják, valamint bemutattam a szerverek skálázási feladatok során felmerülő felelősségeit. Bemutattam, hogy a fel- vagy leskálázásról szóló utasítás következtében mely szervernek mit kell tennie, ám a terhelésről, valamint arról, hogy a döntés a terhelés alapján miként kerül meghatározásra, jelen alfejezetben kívánok szót ejteni.

A terhelés mértékének meghatározására számtalan lehetőség adódik, mivel egy rendszer terheltségét többféle módon lehet mérni. A rendszer terheltségnek mérése során különböző hardverek és szoftverek adott pillanatban történő jellemzőit, tulajdonságait ellenőrizzük, amely eredmények együttes klasszifikálása után meg lehet állapítani a rendszer terheltségét.

Még ha sok információ birtokában is vagyunk, a terhelés meghatározása nem triviális feladat: a szolgáltatást igénybe vevő felhasználók által generált kapcsolatok sokasága nem feltétlen jelenti, hogy a rendszer leterhelt lenne. Még nehezebb a következő

időegységekre előrevetíteni az átlagos terhelést úgy, hogy az korreláljon a valós átlagos terheléssel.

Ahogy haladunk az egyszerű rendszerektől a kritikus rendszerekig, úgy kell a terhelést egyre precízebb módszerekkel meghatározni, amely mérések segítségével a rendszer reális képességeit megismerhetjük adott pillanatban. Teljesen nyilvánvaló, hogy egy kritikus rendszerről, de még egy átlagos, felhasználó, ügyfelek által használt rendszerről is meg kell tudnunk mondani, hogy mekkora a terhelése: a túl nagy terhelés hatására a rendszer számára negatív jelenségek következhetnek be (összeomlás, adatvesztés).

A következő alfejezetekben szeretném részletezni, hogy milyen mérési módszerek álltak rendelkezésemre, az adott módszereknek mik az előnyeik és hátrányaik, és szeretném bemutatni a döntés menetét.

### **5.3.1 A terhelés monitorozása**

Ahhoz, hogy a rendszer képes legyen az automatikus skálázásra, mindenképp tisztában kell lenni a terhelés mindenkori mértékével. A terhelésmérési módszereket alapvetően két nagyobb csoportra oszthatjuk: vannak azok a megközelítések, melynek során egy adott hardver jellemzőit vizsgáljuk (pl. processzor vagy memória kihasználtsága), valamint vannak olyan módszerek, melyek során logikai szinten vizsgáljuk jellemzően szoftverek tulajdonságait (pl. egy webszerver kapcsolatainak száma).

Minél pontosabb eredményt szeretnénk kapni a terhelésről, annál több tulajdonság együttes mérésére van szükség. Ám az, hogy milyen módszert alkalmazunk, függ attól, hogy a rendszer segítségével milyen szolgáltatást nyújtunk, milyen feladatok futnak rajta.

Az én esetemben a szervereken futó feladatok viszonylag egyszerű taszkok. A legerőforrásigényesebb feladatot a virtuális gépek kezelése (azok importálása, indítása, futtatása és törlése) jelenti, ám ezek terhelése a futtatókörnyezeten jelenik meg. A master szervereken és az operatív szervereken megjelenő, az elosztott rendszer szempontjából releváns feladatok a felhasználóktól érkező olvasási és írási feladatok.

Ezek a feladatok jellemzően kicsi adatmennyiségekkel dolgoznak: egy-egy üzenet olvasása vagy írása a cél az üzenőfalon, amelyek mérete majdhogynem elhanyagolható.

Mivel nagyon kisméretű adatrekordok írása és olvasása a cél, az adatbáziskezelő a beérkező kéréseket nagyon rövid idő alatt végrehajtja, így annak terhelése nagyon kis ideig látszik meg a processzor terheléséről készített diagramon.

A másik fontos megjegyzendő megállapítás a kérések terheléséről azok rövidecsük mellett az, hogy ezek a kérések még sok felhasználó mellett is az általam megvalósított rendszerben terhelés szempontjából viszonylag ritkán érkeznek: egy ekkora rendszerben nem fog megtörténni, hogy másodpercenként elérje az ezres nagyságrendet a beérkező kérések száma.

Az előző két megállapításból, nevezetesen a kérések feldolgozásának rövid idejéből és a viszonylag ritka kérésekből arra következtethetünk, hogy a hardverterhelés figyelése nem fogja megalapozni a legoptimálisabb döntés meghozatalát, ezért tehát logikai szinten érdemes a terhelést valamilyen módon monitorozni.

A terhelés meghatározására a logikai szinten a webszerver aktív kapcsolatainak számát vettem alapul. Ez nyilvánvalóan nem egyenértékű a tényleges hardverterhelés mérésével; lehet olyan aktív kapcsolat, amely segítségével hosszú időn keresztül nem indítanak kérést, ezért pontatlan mérési eredményt kaphatunk. Ennek a problémának a tudatában választottam ezt a megoldást, mert ezt az adatot könnyen tudtam manipulálni, ezáltal könnyen tudtam tesztelni is a rendszert.

A webszerver aktív kapcsolatainak számát könnyen elérhetjük: a webszerver egy státuszoldalon elérhetővé tudja tenni az adatokat. Ennek a kódrészletét a 18. ábra mutatja be. Az ábrán látszik, hogy a `/stub_status` úton elérjük az adatokat.

```
server {  
    listen 80 default_server;  
    listen [::]:80 default_server;  
    [...]  
    location /stub_status {  
        stub_status;  
    }  
    [...]  
}
```

**18. ábra** Részlet a webszerver konfigurációs fájljából, a státuszoldal megjelenítésének kódja

Az itt megkapott adatok között szerepel az aktív, az elfogadott és a kezelt kapcsolatok száma, valamint beérkező kérések száma mellett az írási, olvasási és a várakozó

kérések száma. Nekem az aktív kapcsolatok száma a fontos, az alapján hoztam meg a döntést.

A kapcsolatok számát 20 alkalommal, 3 másodpercenként olvassa ki a rendszer, ezen eredményeket ezen 60 másodperces intervallumon átlagolja, egész számúra kerekíti, majd ez alapján dönt. E módszer segítségével elérem, hogy a kiugró értékeket, amik nem feltétlen jellemzik az adott időintervallumot, kisebb súllyal veszem figyelembe, ezáltal azok nem torzítják az eredményt.

A mért adatok megjelenítésére egy külön oldalt hoztam létre a webserveren, ahol látható a kapcsolatok száma mellett a futó operatív szerverek száma és azok IP-címei.

### 5.3.2 Döntés a terhelés függvényében

A döntés során az egyszerűség végett 1 darab aktív kapcsolatra 1 darab operatív szervert biztosítottam, a szerverek számának kezelését diszkrét időben, 65 másodpercenként végeztem (a döntés után vár 5 másodpercet a rendszer, hogy az adott parancs biztosan elindítsa a kellő folyamatokat).

Fontos alapelveként határoztam meg azt, hogy semmiképpen sem szabad megengedni, hogy a rendszer instabillá válhasson, ennek biztosítására minden lépésben legfeljebb eggyel változtatom az operatív szerverek számát. Ennek a korlátozásnak a bevezetésével elkerülöm, hogy a rendszer egy lépésben drasztikusan csökkentse vagy hatalmas mértékben növelje a szerverek számát.

Mivel a nagymennyiségű módosítást elkerültem, kivédtem azt a problémát is, ami a sok szerver egyidejű indítása miatt jelentkezne a futtatókörnyezeten.

Az operatív szerverek számát ( $\omega$ ) diszkrét időpillanatban ( $\tau$ ) a következő egyenlettel határozom meg a diszkrét időpillanatot megelőző intervallumon számolt aktív kapcsolatok átlagának ( $\psi_\tau$ ) függvényében:

$$\omega_\tau = \begin{cases} \omega_{\tau-1} + 1, & \text{ha } \psi_{\tau-1} < \psi_\tau \\ \omega_{\tau-1}, & \text{ha } \psi_{\tau-1} = \psi_\tau \\ \omega_{\tau-1} - 1, & \text{ha } \psi_{\tau-1} > \psi_\tau \end{cases}$$

Az inicializáló lépésben létrehozok egy futó operatív szerveret, így onnantól kezdve a rendszer tud működni. Fontos szabály, hogy

$$\forall \tau \geq 0 \text{ esetén } \omega_\tau \geq 1$$

Tehát minden időpillanatban kötelezően futnia kell legalább 1 darab operatív szervernek függetlenül attól, hogy mekkora a beérkező kérések mennyisége. Ezt a szabályt csak a következő időpillanatokra tudjuk értelmezni, mivel a korábbi állapotot befolyásolni már nem tudja a rendszer.

Az egyenletből látszik, hogy teljesül az az elv, hogy lépésenként legfeljebb eggyel módosul a szerverek száma. Ennek következménye, hogy a bejövő kérések számának hirtelen növekedését a rendszer lassan, elnyújtva képes kezelni. Ez egyrészt hátrány, mivel a hirtelen megnövekvő igényeket a rendszer egy ideig csak kisebb kapacitással képes kezelni, ami miatt teljesítményproblémák jelentkezhetnek a felhasználóknál addig, amíg a rendszer el nem indít megfelelő számú szervert. Másrészt ez előny is, mivel amennyiben a rendszer csak rövid ideig kap nagy mennyiségű kérést, akkor nem szükséges nagyon sok szervert elindítani.

Ezt a döntést az adatok monitorozásával egyetemben egy erre a célra a háttérben futtatott szolgáltatás, az általam készített *uzenofal.service* végzi. Ez a szolgáltatás hasonló ahhoz a szolgáltatáshoz, mint amivel a webszerver működik (*nginx.service*), ám a saját szolgáltatásom működését én határoztam meg a fentiek alapján.

Amennyiben a diszkrét időpillanatban megszületik a döntés a szerverek számának változtatásáról, a szolgáltatás kiadja a megfelelő parancsot a master szerveren, aminek következményét korábban már bemutattam.

## 6 Eredmények bemutatása és értékelés

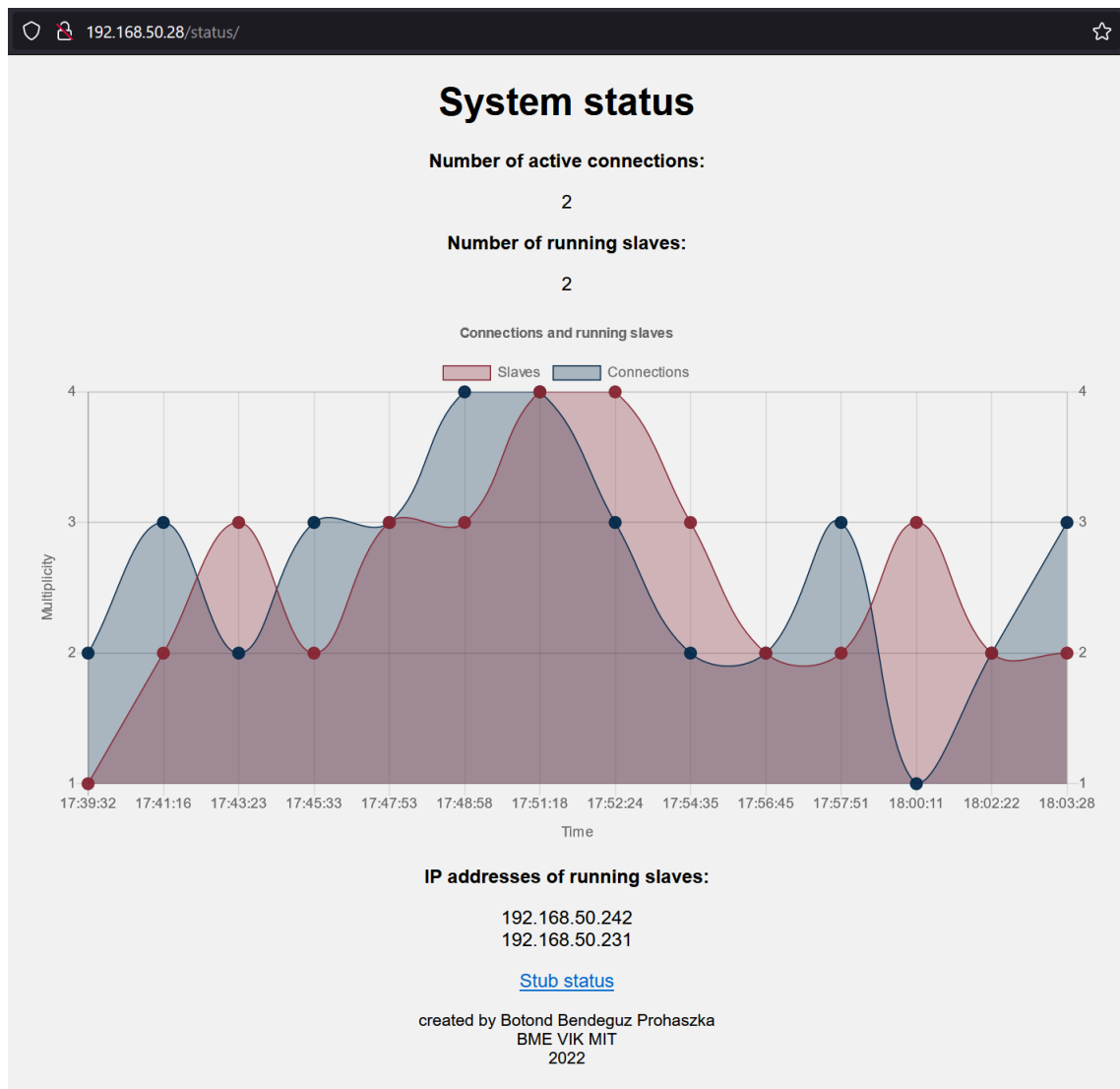
### 6.1 Általános értékelés

A korábbiakban bemutatam, hogyan lehet megtervezni és megalkotni egy működő elosztott rendszert, amely képes automatikus skálázásra. Bemutattam, hogy a rendszer tervezése során mely pontokon milyen döntéseket kell meghozni, az adott feladatokra milyen megoldási lehetőségek vannak, illetve kitértem arra, hogy én mit miért választottam.

Az elkészült rendszer képes egy adott szolgáltatást futtatni, a beérkező kéréseket el tudja osztani a backend szerverek között, a szerverek pedig a választ vissza tudják küldeni a felhasználónak, a rendszer teljesíti az elosztott működéssel kapcsolatos elvárásokat.

A rendszer továbbá alkalmas a terhelés monitorozására, aminek függvényében el tud indítani további backend szervereket, azokat a rendszerbe tudja integrálni oly módon, hogy azok szintén részt vesznek a beérkező kérések feldolgozásában. A rendszer a terhelés csökkenésének hatására backend szervereket állít le, amiket a rendszerből eltávolítva azok több kérést nem dolgoznak fel.

A rendszer a terhelésről és a futó operatív szerverek számáról gyűjtött adatokat egy státuszoldalon megjeleníti, továbbá megjeleníti a mindekori futó szerverek elérési címét.



19. ábra A rendszer státuszoldala

A 19. ábrán látható diagram a rendszer adatait mutatja be futás közben. Látható, ahogy a kapcsolatok átlagának görbáját szépen követi a futó operatív szerverek száma. Látható továbbá, ahogyan a kapcsolatok száma nagyobb ugrással csökken, úgy az operatív szerverek száma csupán eggyel. Hasonlóan reagál a rendszer, ha a kapcsolatok száma nem csökken, hanem nő.

## 6.2 Sebezhetőségek, gyengeségek

Az elkészült rendszernek nem célja a teljesen robusztus futás, illetve nem célja a kifogástalan állapot biztosítása adatbiztonság szempontjából.



Sok sebezhetőség található a rendszerben, kezdve onnan, hogy az operatív szerverek teljeskörű jogosultságot kapnak a master szerveren futó adatbázisban létrehozott felhasználóikkal odáig, hogy a weboldalak kommunikációja nincs titkosítva.

A rendszer jelen állapotában nem képes lekezelni az olyan jelenségeket, amelyek során hirtelen nagyságrendekkel több aktív kapcsolat nyílik a rendszer felé és azokon keresztül a felhasználók elárasztják azt tömérdek kéréssel. Ekkor a rendszer összeomolhat és bekövetkezhet adatvesztés, ám jelen szakdolgozat célja csupán egy egyszerűsített megoldás megvalósítása volt.

A megvalósított rendszer továbbá nem működik költséghatékonyan: rendkívül pazarló 1 darab nyitott kapcsolatra 1 darab futó operatív szervert fenntartani.

### **6.3 Továbbifejlesztési lehetőségek**

Minden sebezhetőségre és gyengeségre létezik valamilyen megoldás. Az adatbiztonsági szempontokat nyilvánvalóan egy éles környezetben futó, felhasználók által használt rendszerben szem előtt kell tartani a tervezés során, hogy semmiképp se jelentsen kockázatot sem a rendszer üzemeltetése, sem a nyújtott szolgáltatás igénybevétele.

Az üzemeltetés költségeivel kapcsolatos kérdéskörre megoldásként lehetőség van egy olyan függvény meghatározására, amely segítségével úgy tudjuk meghatározni a futtatandó operatív szerverek számát a kérések függvényében, hogy a rendszer üzemeltetése közel költséghatékony legyen. A függvény módosítására a rendszer lehetőséget ad, a megfelelő fájl módosításával egyszerűen el lehet végezni.

Lehetőség van több gyűjtött adat vagy naplófájlok megjelenítésére, ahhoz, hogy kényelmesen elérhetőek legyenek, a státuszoldalt fejleszteni kell.

## 7 Hivatkozások

- [1] „VULTR,” [Online]. Available: <https://www.vultr.com/products/cloud-compute/>. [Hozzáférés dátuma: 24. október 2022.].
- [2] „RESTful API,” [Online]. Available: <https://aws.amazon.com/what-is/restful-api/>. [Hozzáférés dátuma: 24 október 2022].
- [3] „ATLANTIC,” [Online]. Available: <https://www.atlantic.net/vps-hosting/>. [Hozzáférés dátuma: 28 október 2022].
- [4] „CHERRY SERVERS,” [Online]. Available: <https://www.cherryservers.com/>. [Hozzáférés dátuma: 28 október 2022].
- [5] „REST API,” [Online]. Available: <https://www.ibm.com/cloud/learn/rest-apis>. [Hozzáférés dátuma: 28 október 2022].
- [6] „Ubuntu 18.04.6 LTS,” [Online]. Available: <https://releases.ubuntu.com/18.04/>. [Hozzáférés dátuma: 06 november 2022].
- [7] „PUTTY,” [Online]. Available: <https://putty.org/>. [Hozzáférés dátuma: 06 november 2022].
- [8] „VirtualBox,” [Online]. Available: <https://www.virtualbox.org/>. [Hozzáférés dátuma: 06 november 2022].
- [9] „MySQL,” [Online]. Available: <https://ubuntu.com/server/docs/databases-mysql>. [Hozzáférés dátuma: 07 november 2022].
- [10] „WordPress,” [Online]. Available: <https://wordpress.org/>. [Hozzáférés dátuma: 07 november 2022].

- [11] „WordPress statisztika,” [Online]. Available: <https://trends.builtwith.com/cms>. [Hozzáférés dátuma: 07 november 2022].
- [12] „Install and Configure WordPress on Ubuntu,” [Online]. Available: <https://ubuntu.com/tutorials/install-and-configure-wordpress#1-overview>. [Hozzáférés dátuma: 12 november 2022].
- [13] „Apache2,” [Online]. Available: <https://www.apache.org/licenses/LICENSE-2.0.html>. [Hozzáférés dátuma: 12 november 2022].
- [14] „How to setup a MySQL replication database for WordPress,” Torbjorn Zetterlund, [Online]. Available: <https://torbjornzetterlund.com/how-to-setup-a-mysql-replication-database-for-wordpress/>. [Hozzáférés dátuma: 19 november 2022].
- [15] „nginx,” [Online]. Available: <https://nginx.org/en/>. [Hozzáférés dátuma: 20 november 2022].
- [16] „nginx load balance,” [Online]. Available: [https://nginx.org/en/docs/http/load\\_balancing.html](https://nginx.org/en/docs/http/load_balancing.html). [Hozzáférés dátuma: 20 november 2022].
- [17] „Application Programming Interface,” [Online]. Available: <https://www.ibm.com/cloud/learn/api>. [Hozzáférés dátuma: 28 október 2022].
- [18] „Advanced Package Tool,” [Online]. Available: <https://appuals.com/what-does-apt-mean-in-linux-distributions/>. [Hozzáférés dátuma: 07 november 2022].
- [19] „Hypertext Transfer Protocol,” [Online]. Available: <https://httpwg.org/specs/>. [Hozzáférés dátuma: 24 október 2022].
- [20] „HyperDB,” [Online]. Available: <https://wordpress.org/plugins/hyperdb/>. [Hozzáférés dátuma: 07 november 2022].

- [21] „JavaScript Object Notation,” [Online]. Available: <https://www.json.org/json-en.html>. [Hozzáférés dátuma: 29 október 2022].
- [22] „PHP: Hypertext Preprocessor,” [Online]. Available: <https://www.php.net/>. [Hozzáférés dátuma: 07 november 2022].
- [23] „Secure Shell,” [Online]. Available: <https://www.ssh.com/academy/ssh>. [Hozzáférés dátuma: 06 november 2022].
- [24] „Virtual Private Server,” [Online]. Available: <https://www.ibm.com/cloud/learn/vps>. [Hozzáférés dátuma: 29 október 2022].
- [25] „Hypertext Transfer Protocol Secure,” [Online]. Available: <https://en.wikipedia.org/wiki/HTTPS>. [Hozzáférés dátuma: 28 október 2022].