

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Prohászka Botond Bendegúz

Terhelés hatására automatikusan skálázódó webes rendszer fejlesztése

Konzulens

BUDAPEST, 2022

C:\Users\szarnyasg\Downloads\bme_logo_nagy.eps

**Budapesti Műszaki és Gazdaságtudományi Egyetem**

Villamosmérnöki és Informatikai Kar

Méréstechnika és Információs Rendszerek Tanszék

Terhelés hatására automatikusan  
skálázódó webes rendszer fejlesztése

Készítette

Prohászka Botond BendegúzKonzulens

Eredics Péter

2022

Tartalomjegyzék

[Összefoglaló 5](#_Toc118638704)

[Abstract 6](#_Toc118638705)

[1. Bevezetés [3-5] 7](#_Toc118638706)

[1.1. Általános bevezetés 7](#_Toc118638707)

[1.2. A feladat célja 7](#_Toc118638708)

[1.3. A szakdolgozat felépítése 7](#_Toc118638709)

[2. Létező megoldások bemutatása és értékelése [5-10] 8](#_Toc118638710)

[2.1. VPS fogalma 8](#_Toc118638711)

[2.2. A VPS alkalmazási területei 9](#_Toc118638712)

[2.3. VPS szolgáltatók 9](#_Toc118638713)

[2.3.1. VULTR 10](#_Toc118638714)

[2.3.2. ATLANTIC 11](#_Toc118638715)

[2.3.3. CHERRY SERVERS 12](#_Toc118638716)

[3. Architektúra [3-5] 14](#_Toc118638717)

[3.1. A rendszer mint reverse proxy 14](#_Toc118638718)

[3.1.1. Proxy szerver 14](#_Toc118638719)

[3.1.2. Reverse proxy szerver 15](#_Toc118638720)

[3.2. Elosztott rendszer 17](#_Toc118638721)

[3.3. A rendszer architektúrája és felépítése 18](#_Toc118638722)

[4. Elosztott rendszer [innentől 30] 20](#_Toc118638723)

[4.1. Operatív szerverek feladata 20](#_Toc118638724)

[4.2. Master szerver feladata 21](#_Toc118638725)

[4.3. Alkalmazott technológiák 21](#_Toc118638726)

[4.3.1. Operációs rendszer 22](#_Toc118638727)

[4.4. Megvalósítás 22](#_Toc118638728)

[4.4.1. Operációs rendszer 22](#_Toc118638729)

[4.4.2. Adatbázis 22](#_Toc118638730)

[4.4.3. Demóalkalmazás 22](#_Toc118638731)

[4.5. A megvalósítás során jelentkezett problémák 22](#_Toc118638732)

[4.6. A működő rendszer megvalósítása 22](#_Toc118638733)

[5. Automatikus skálázás 23](#_Toc118638734)

[6. Eredmények bemutatása, értékelés [4-5] 24](#_Toc118638735)

[6.1. Általános értékelés 24](#_Toc118638736)

[6.2. Sebezhetőségek 24](#_Toc118638737)

[6.3. Továbbfejlesztési lehetőségek 24](#_Toc118638738)

[7. Konklúzió 25](#_Toc118638739)

[8. Hivatkozások 26](#_Toc118638740)

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. 11. 12.

Prohászka Botond Bendegúz

# Összefoglaló

# Abstract

# Bevezetés [3-5]

## Általános bevezetés

## A feladat célja

## A szakdolgozat felépítése

# Létező megoldások bemutatása és értékelése [5-10]

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 a környezet lehet egy VPS szolgáltatás[[1]](#footnote-1), de lehet egy mezei asztali számítógép, laptop, ám ezek hardvertulajdonságai adott esetben több nagyságrenddel is eltérhetnek egymástól, így amikor egy adott felhasználáshoz 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.

## 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 egy szerverközpontban lévő fizikai szervereken futtatnak. Az ilyen szerverparkok üzemeltetése is egy külön szolgáltatás, lehet vásárolni szerver rack helyet, ahol a saját fizikai szerverünket elhelyezhetjük. Ezen szolgáltatások nem összekeverendőek a VPS szolgáltatásokkal, mert a VPS szolgáltatások esetében egy virtuális szervert veszünk, amelyet futtathat a szolgáltató cég tömérdek fizikai szervere közül bármelyik egy ilyen szerverparkban.

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. Erre a széles személyre szabhatósági lehetőségre éppen azért van lehetőség, mert virtuális számtógépekről 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 kimondottan 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 van.

## A VPS alkalmazási területei

Az előbb bemutatott tulajdonságokat összegezve elmondható, hogy az ilyen szolgáltatások célcsoportja azok a (kisebb) cégek, melyeknek nincs szükségük vagy lehetőségük saját szervereket a kellő minőségben üzemeltetni: ezek a cégek általában nem rendelkeznek nagy IT részleggel és a szolgáltatásaik között sem szerepel IT tevékenység. Az ennél nagyobb cégek, akiknek már szükséges, saját szervereket üzemeltetnek. Az ennél kisebb szereplők (például a szakdolgozatukat készítő hallgatók) számára pedig elegendő a saját szerver (és ebben az esetben pedig inkább beszélhetünk már nem használt számítógépekről, laptopokról, amik ki lettek nevezve szervernek) futtatása, 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 rendszert stb.

## 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 a 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 virtuálisgép lehelyezésekor 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. Meglehetősen sok 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.

### 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.

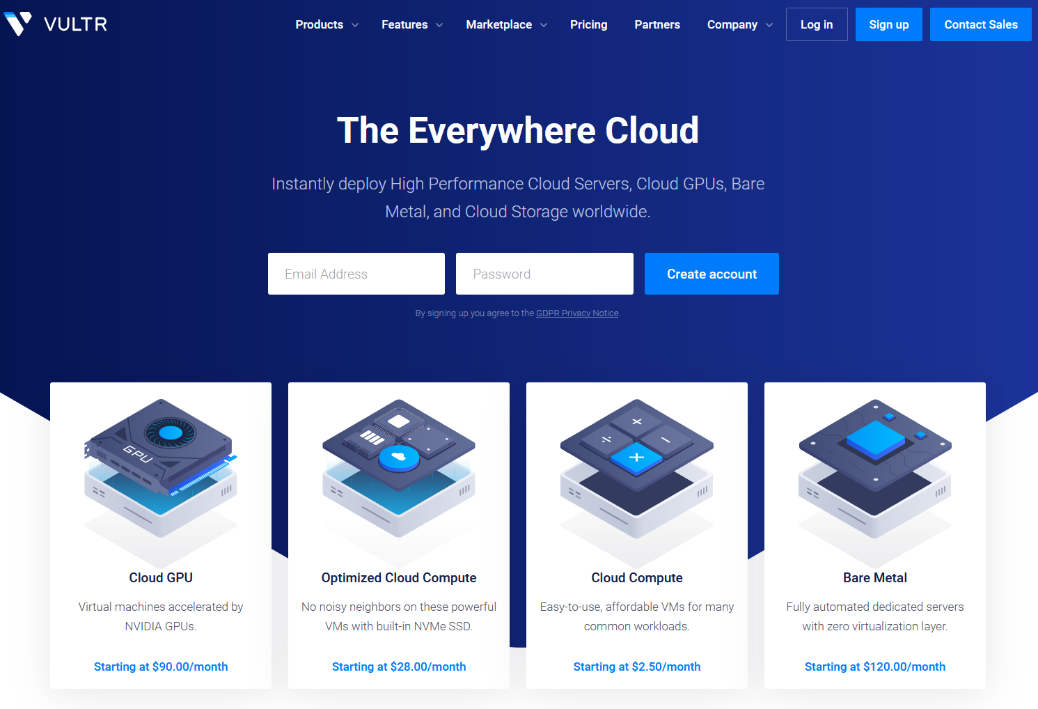
A díjszabás óra-alapú, így csak azért az időért kell fizetni, amit tényleg futott a szerver. Lehet snapshot fájlt létrehozni és lehet egy image alapján több szervert is lehelyezni a szolgáltató több kontinensen, több városban található szerverein.

A szervert lehet API[[2]](#footnote-2) hívásokkal modifikálni, az API hívások a standard HTTP[[3]](#footnote-3) kifejezéseket, válaszkódokat használják, betartják a RESTful API [2] tervezési elveket és természetesen megvalósíthatóak velük a CRUD[[4]](#footnote-4) műveletek is.

A kommunikáció során a kéréseket és a válaszokat konzisztens és jól formázott JSON[[5]](#footnote-5) 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 másodpercenkét 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]).

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

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



1. ábra A VULTR szolgáltató honlapjának részlete

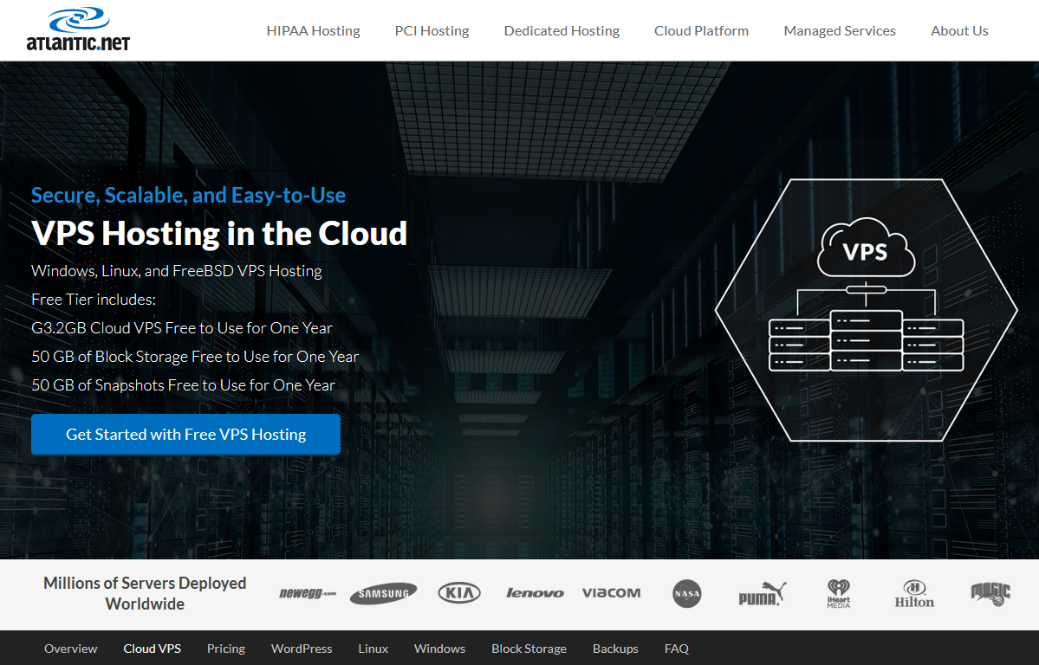
### ATLANTIC

A VULTR-hoz hasonlóan az ATLANTIC [3] is egy Egyesült Államokbeli cég, ennek székhelye Orlandoban 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[[6]](#footnote-6) 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.

Az előző példában említett API-képességeket az ATLANTIC szintén biztosítja, ám ezesetben 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 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.

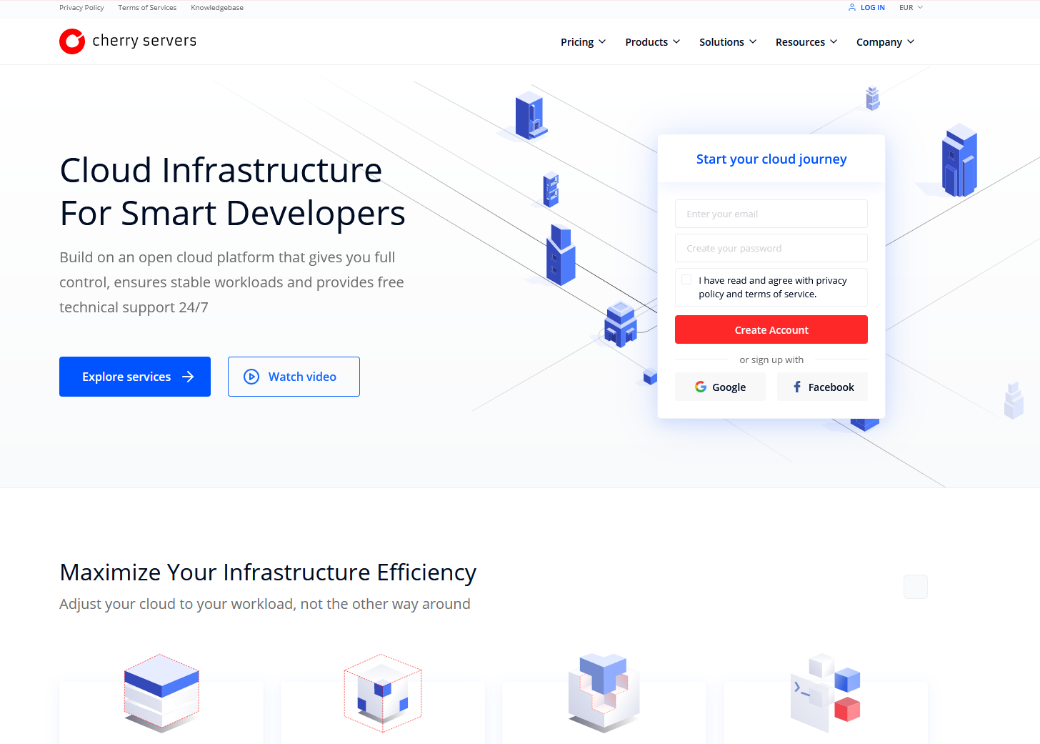


2. ábra Az ATLANTIC szolgáltató honlapjának részlete

### 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* (azaz héjprogramozás) 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

# Architektúra [3-5]

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 egy-egy elvégzendő egység funkcionalitását tekintve egyszerű, valamint az ilyen modulokat tovább is lehet még bontani a modulon belül további kisebb részekre.

Ez a megközelítés két okból is előnyös. Nagyobbrészt 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. Másrészről 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

## A rendszer mint reverse proxy

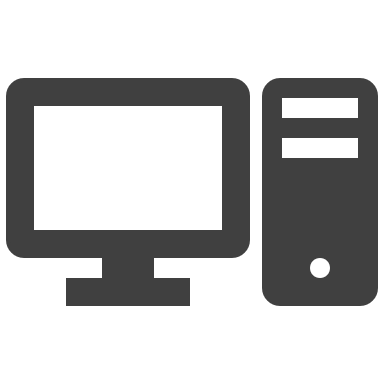
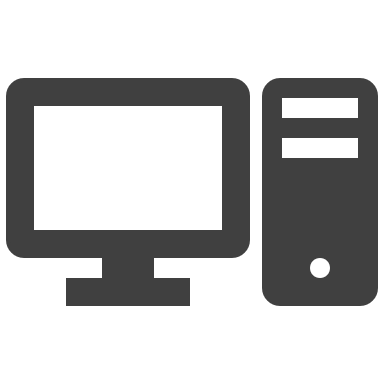
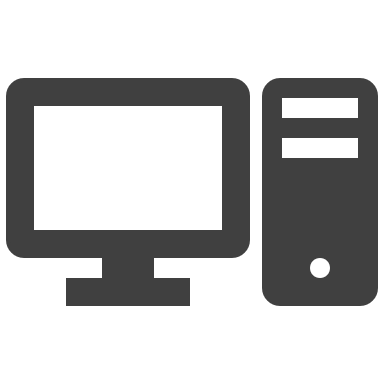
Mielőtt bemutatnám a reverse proxy fogalmát, működését és relevanciáját, logikailag fontos megérteni a proxy vagy proxy szerver fogalmát. (A magyar terminológiában nincs gyakran alkalmazott megnevezése a proxy szervernek, ezért a továbbiakban mind a proxy, mind a reverse proxy szervert az eredeti angol nevén nevezem. Magyarra fordítva meghatalmazott szerver az a kifejezés, ami a legközelebb áll a fogalomhoz.)

### 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 is és annak 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.



Proxy szerver



Kliensek



Internet



Célszerver

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, majd 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 tudják betölteni az elsődleges tűzfal vagy naplózó szerver szerepét is.

### Reverse proxy szerver

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



Proxy szerver

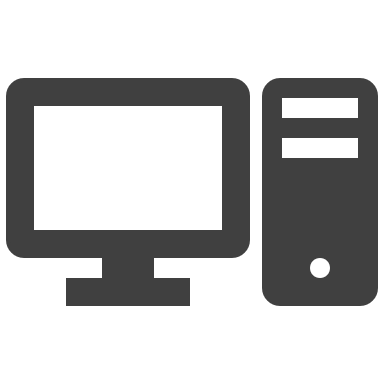
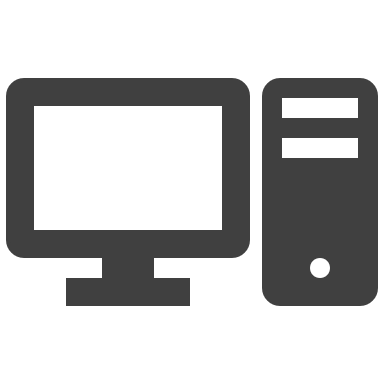


Alkalmazásszerverek



Internet

Kliensek



5. ábra Egyszerű topológia a reverse proxy szerver működésének szemléltetésére

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

Az egyik legegyértelműbb döntést akkor lehet meghozni, ha az alkalmazásszerverek különböző funkcionalitással bírnak. Ebben az esetben (gyakorta) minden szerver más feladatot lát el: van fájlszerver, 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 klienstől mért legközelebb lévő szervernek küldi tovább a reverse proxy szerver, ezzel jelentős időt lehet spórolni.

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

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

## Elosztott rendszer

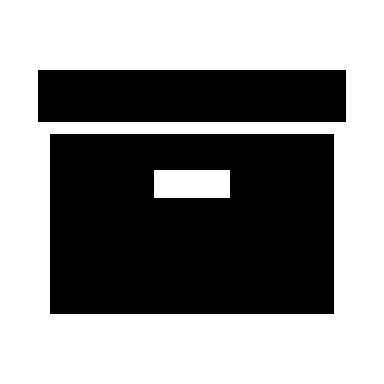
Összetettebb feladatok esetében régebben a bevett megoldás volt, hogy a feladat megoldására allokált hardvert megpróbálták a végletekig fejleszteni, hogy az adott hardver minél gyorsabban meg tudja oldani a feladatot. Ennél valamivel költséghatékonyabb és optimálisabb megoldást jelent a napjainkban már 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[[7]](#footnote-7) 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 a módosított adatbázis alapján működjék, ne csak a kérést feldolgozó szerver. Ellenkező esetben inkonzisztens lenne az elosztott rendszer állapota.)



Kérés



Elosztott rendszer

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 szervereket is elérik külön-külön ebben az esetben nem (feltétlenül) az előbb említett belépési ponton keresztül.)

A 6. ábrán bemutatott topológián az elosztott rendszer határát egy fekete doboz (az angol terminológiával *black box*) reprezentálja, ennek ugyanis jelen 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 [angolul *white box*], ebben az esetben a belső működést pontosan ismerjük.)

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

## 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.

Az szerverek számáról eddig nem esett szó: a master szerverből praktikusan 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 lehelyezni 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.



Kérés

Elosztott rendszer

**.**

**.**



Master szerver

Operatív szerverek

7. ábra A rendszer sematikus architektúrája

# Elosztott rendszer megvalósítása [innentől 30]

Ebben a fejezetben szeretném részletezni az elosztott rendszer megvalósításának folyamatát. A fejezetben először bemutatom az operatív és a master szerverek jellemzőit, az alkalmazott technológiákat, majd részletezem a megvalósítás folyamatát, kitérek a fejlesztés során felbukkanó hibákra 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 terheléselosztás szempontjából meghatározott számú operatív szerver között elossza és feldolgozza azokat. 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 szélesebb funkcionalitással rendelkezik, ami több telepített alkalmazással jár.

A feladat tervezése során úgy döntöttem, hogy a master szerver egy önálló egység lesz, az azt működtető szerver más funkcionalitással az elosztott rendszer szempontjából nem rendelkezik.

## 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.

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. (Természetesen ez nem minden esetben igaz, mivel elvárjuk, hogy a szervert azonosító adat lekérésére különböző, a saját azonosító adatukat küldjék válaszul [például IP cím].)

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, amely 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 bármely szerver módosítja az adatbázisát, konzisztens maradjon az összes többivel. (Arról, hogy miért szükséges minden operatív szerveren külön adatbázist futtatni, valamint az adatbázisok működéséről a 4.4.2 fejezetben lesz szó.)

## 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 hálózattal, így ennek helyes konfigurálása rendkívüli prioritással bír.

A kívülről érkező kérések elsőként a master szervert érik el. A master szerver feladata többrétű. Egyrészről szükséges, hogy nyilvántartsa az operatív szerverek számát, elérési címeit annak érdekében, hogy a kéréseket el tudja juttatni az alkalmazásszerverekre, másrészről biztosítania 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.

## 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 teljes mértékben, de legalább nagyban hasonlítanak rá.

Egy rendszer készítése során kifejezetten előnyös, ha bevett megoldásokra támaszkodunk, és ennek több oka is van. Egyik kifejezetten hasznos ismérve a sokak által használt megoldásoknak, hogy nem véletlen használják sokan, oka van annak, hogy népszerű. Ok lehet, hogy az adott technológiát pont hasonló feladatokra találták ki, vagy a technológia használatával nagyon leegyszerűsödnek, költséghatékonyabbá válnak a folyamatok. A másik előnyös tulajdonsága a *best practice*[[8]](#footnote-8)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 vagy azt, hogy konkrétan mit érdemes használnom.

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

### Operációs rendszer

A rendszerben elhelyezkedő szervereket az eddigiekben és továbbiakban is a már említett két csoportra oszthatjuk (úgy, mint operatív szerverek és 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 népszerű operációs rendszerekhez képest ez a grafikus felülettel nem rendelkező operációs rendszer telepített változata viszonylag kevés helyet foglal: a működő rendszerben a master szerver helyigénye minden telepített alkalmazással és szolgáltatással együtt nagyságrendileg ~5GB, ezzel szemben a népszerű Windows 10 mérete is legalább ennek kétszeresem, ám annak az operációs rendszernek nem ugyanaz a célja, mint az általam használt Ubuntunak.

Mivel az operációs rendszer nem rendelkezik grafikus felülettel, terminálból lehet elérni. Az elérésre SSH[[9]](#footnote-9) kapcsolatot alkalmazó PUTTY [7] klienst használtam. Ennek használata megkönnyítette a szerverek elérését, a PUTTY terminálja felhasználóbarát kialakítása is segítségemre volt.

A szervereket, amiket virtuális számítógépekként manifesztáltam, az Oracle VirtualBox [8] segítségével futtatam a saját, *Ubuntu* *22.04.1 LTS Desktop* operációs rendszert futtató laptopomon. Annak ellenére, hogy saját laptopom nem rendelkezik olyan tulajdonságú erőforrásokkal, mint egy bérelhető szerver egy szerverparkban, sokkal olcsóbb az üzemeltetése, valamint a célnak teljes mértékben megfelel.

A képen szöveg látható

Automatikusan generált leírás

8. ábra A PUTTY terminálja

A 8. ábrán látható a PUTTY terminálablaka, 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 sosem nyilvános hálózatra, csupán privát, belső hálózatra, valamint a feladat *proof of concept*[[10]](#footnote-10)jellege miatt esetemben nem jelentett rizikófaktort, valamint 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[[11]](#footnote-11) segítségével számtalan csomagot telepíthetünk egyetlen paranccsal kezdve adatbázisalkalmazásoktól, a webszervereken keresztül egészen a szórakoztatási céllal készített csomagokig.

### Adatbázisszerver

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

Azért van szükség minden szerveren külön adatbázisra, mert így minden szerver a saját adatbázisából tud olvasni, és mivel általánosságban kijelenthető, hogy olvasási kérésből több van, mint írásiból (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.

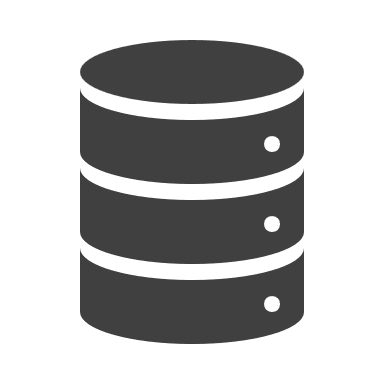
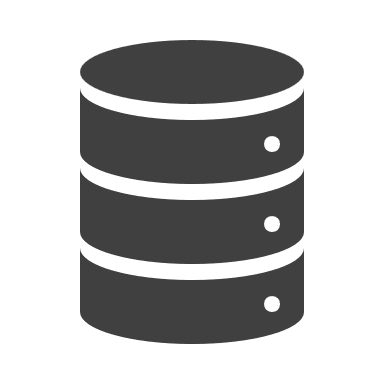
Amennyiben ezeket az adatbázisokat nem kapcsoljuk össze semmilyen módon, úgy csak néhány ugyanolyan operatív szerverünk lenne, melyek mind ugyanazt a szolgáltatást nyújtják, de mindegyik más és más adatokat tartalmazna. Ahhoz, hogy mégis egy elosztott szolgáltatásról beszélhessünk, mely számos szerverből 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 ú. replikációs működés megvalósítására, mely lényegében pontosan megegyezik azzal a működéssel, amelyre az elosztott rendszer épít. A replikációs működés alapja, hogy egy adatbázist egy másik adatbázis alapján hozunk létre, valamint működésüket is összekapcsoljuk.

Ilyen replikációs működésre sokféle felállást, topológiát vagy szcenáriót 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.

Írás



Olvasás

„A” szerver

„B” szerver

Replikáció

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.

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

### 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[[12]](#footnote-12) illetve MySQL alapú, de bővítményekkel a képességei bővíthető. Esetemben is ez a cél: a Wordpress alapműködését a HyperDB[[13]](#footnote-13) nevű bővítménnyel kibővítem, 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.

## 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 sorrendjét tekintve úgy lettek meghatározva, hogy a későbbi feladatok mind valamekkora mértékben az előző feladatokra építenek, s igénylik, hogy a korábbi feladatok hibamentesen el legyenek végezve.

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

### Működő 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 részfeladatot: egy master szerver és két operatív szerverre volt szükségem. Ezek a mennyiségek minimálisak, ennyi szerver minimum kell ahhoz, hogy az elosztott működést érdemben használni, tesztelni tudjam, ennél kevesebb szerverrel nem megvalósítható vagy tesztelhető jól, ennél több pedig tulajdonképpen nehezítette volna a folyamatokat az áltatluk okozott kezelési többletfeladatokkal.

A korábban már említett VirtualBox szoftver segítségével létrehoztam három 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* [[14]](#footnote-14)adapterre, amely segítségével mindenféle belső hálózat kihagyásával direkt módon egyből a routerre csatlakoztak a virtuális gépek. Ezáltal minden a router hálózatára csatlakoztatott eszköz elérte a virtuális gépeket.

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

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

## A megvalósítás során jelentkezett problémák

## A működő rendszer megvalósítása

# Automatikus skálázás

# Eredmények bemutatása, értékelés [4-5]

## Általános értékelés

## Sebezhetőségek

## Továbbfejlesztési lehetőségek

# Konklúzió

# 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] | „Hypertext Transfer Protocol,” [Online]. Available: https://httpwg.org/specs/. [Hozzáférés dátuma: 24 október 2022]. |
| [8] | „Hypertext Transfer Protocol Secure,” [Online]. Available: https://en.wikipedia.org/wiki/HTTPS. [Hozzáférés dátuma: 28 október 2022]. |
| [9] | „Application Programming Interface,” [Online]. Available: https://www.ibm.com/cloud/learn/api. [Hozzáférés dátuma: 28 október 2022]. |
| [10] | „VPS,” [Online]. Available: https://www.ibm.com/cloud/learn/vps. [Hozzáférés dátuma: 29 október 2022]. |
| [11] | „JSON,” [Online]. Available: https://www.json.org/json-en.html. [Hozzáférés dátuma: 29 október 2022]. |
| [12] | „SSH,” [Online]. Available: https://www.ssh.com/academy/ssh. [Hozzáférés dátuma: 06 november 2022]. |

1. *Virtual Private Server*, azaz Virtuális Privát Szerver [9] [↑](#footnote-ref-1)
2. *Application Programming Interface* [8] [↑](#footnote-ref-2)
3. *Hypertext Transfer Protocol* [7] [↑](#footnote-ref-3)
4. *Create, Read, Update and Delete*, azaz létrehozás, olvasás, módosítás és törlés [↑](#footnote-ref-4)
5. *JavaScript Object Notation* [10] [↑](#footnote-ref-5)
6. Hypertext Transfer Protocol Secure [7] [↑](#footnote-ref-6)
7. Mivel a számítógép és a szerver fogalmak között jelen esetben nincs érdemi különbség, szinonimaként kezelem a két kifejezést [↑](#footnote-ref-7)
8. Magyar nyelven is gyakran használt angol kifejezés a bevett megoldásokra. [↑](#footnote-ref-8)
9. Secure Shell, azaz biztonságos héj. [12] [↑](#footnote-ref-9)
10. 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. [↑](#footnote-ref-10)
11. *Advanced Package Tool*, azaz fejlett csomageszköz [15] [↑](#footnote-ref-11)
12. Eredetileg: *Personal Home Page*, napjainkban azonban a *PHP: Hypertext Preprocessor* rövidítése. [18] [↑](#footnote-ref-12)
13. A Wordpress egyik bővítménye, amely egy adatbáziskezelő bővítmény. [19] [↑](#footnote-ref-13)
14. Áthidalt [↑](#footnote-ref-14)