



**Budapesti Műszaki és Gazdaságtudományi Egyetem**

Villamosmérnöki és Informatikai Kar

Irányítástechnika és Informatika Tanszék

# Kiszolgálók üzemeltetése nagyvállalati környezetben és kapcsolódó szoftverek fejlesztése

SZAKDOLGOZAT

*Készítette*

Főglein Simon István

*Konzulens*

dr. Somogyi Péter

2024. május 22.

# Tartalomjegyzék

## Tartalomjegyzék

<b>Kivonat</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>1. Bevezetés</b>	<b>1</b>
<b>2. Nagyvállalati környezetek ismertetése</b>	<b>4</b>
2.1. Áttekintés . . . . .	4
2.2. Adatbiztonság . . . . .	5
2.3. Nagyvállalati és végfelhasználói környezetek összehasonlítása . . . . .	5
<b>3. Technológiai áttekintés</b>	<b>7</b>
3.1. Szervergépek . . . . .	7
3.2. Virtualizáció . . . . .	8
3.2.1. Népszerű virtualizációs technológiák . . . . .	9
3.2.2. Virtuális gépek használatának néhány előnye . . . . .	11
3.2.2.1. Erőforrások testreszabása . . . . .	12
3.2.2.2. Pillanatképek . . . . .	12
3.2.2.3. Migráció . . . . .	13
3.2.3. Konténerizáció . . . . .	13
3.3. RAID . . . . .	14
3.4. Logikai kötetkezelés . . . . .	14
3.5. OS-lehetőségek . . . . .	16
3.6. Infrastruktúra-menedzsment . . . . .	17
3.6.1. Ansible és Salt . . . . .	18
3.7. Felügyelet . . . . .	18

3.7.1. Icinga és Prometheus Grafana vizualizációval . . . . .	19
<b>4. Virtualizációs környezet létrehozása</b>	<b>20</b>
4.1. Kialakítani kívánt környezet meghatározása . . . . .	20
4.2. Fizikai gép ismertetése . . . . .	22
4.3. Operációs rendszer . . . . .	23
4.3.1. OS-kiválasztás folyamata . . . . .	23
4.3.1.1. openSUSE Leap . . . . .	25
4.3.1.2. openSUSE MicroOS . . . . .	25
4.4. Hálózati topológia . . . . .	26
4.4.1. Bridge-dzsel hálózati interfész . . . . .	27
4.5. Virtualizációs komponensek telepítése . . . . .	28
4.5.1. Hosztgép konfigurálása . . . . .	28
4.5.2. Virtuális gépek telepítése . . . . .	30
<b>5. Infrastruktúra-menedzsment: Uyuni</b>	<b>33</b>
5.1. Telepítés . . . . .	33
5.1.1. Kötetkiosztás . . . . .	34
5.2. Telepítőforrások tükrözése . . . . .	35
5.2.1. Online kötetnövelés . . . . .	36
5.3. Kliensek felvétele . . . . .	37
5.3.1. Rendszertípusok és Salt Formulák . . . . .	39
5.4. Szolgáltatások telepítése, kezelése . . . . .	40
5.5. Erőforrásigény mérése . . . . .	42
5.6. Frissítési értesítések . . . . .	44
<b>6. Monitoring</b>	<b>45</b>
6.1. Áttekintés . . . . .	45
6.2. Telepítés, konfiguráció . . . . .	47
6.2.1. Konténerhoszt monitorozása . . . . .	47
6.2.2. Tűzfal beállítása . . . . .	47
6.3. Riasztások beállítása . . . . .	48
6.4. Adatvizualizáció . . . . .	51
<b>7. Összefoglalás</b>	<b>54</b>
7.1. Elért eredmények . . . . .	54

7.2. Továbbfejlesztési lehetőségek . . . . .	54
<b>Köszönetnyilvánítás</b>	<b>56</b>
<b>Szójegyzék</b>	<b>57</b>
<b>Betűszavak</b>	<b>58</b>
<b>Irodalomjegyzék</b>	<b>60</b>
<b>Függelék</b>	<b>63</b>
F.1. A monitoring beállítása előtti adatgyűjtéshez használt program . . . . .	63
F.2. E-mail-értesítések beállítását tartalmazó konfigurációs fájl . . . . .	66

## HALLGATÓI NYILATKOZAT

Alulírott *Főglein Simon István*, 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(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált 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.

Budapest, 2024. május 22.

---

*Főglein Simon István*

hallgató

# Kivonat

Az informatika életünk szerves részévé vált: az interneten keresztül vásárolhatunk, intézhetjük banki ügyeinket és kommunikálhatunk egymással. Az egyre szélesebb körben elérhető szolgáltatások folyamatos üzemeltetése alapos tervezést, nagy szakértelmet és felkészültséget kíván meg a rendszermérnököktől.

A megnövekedett forgalom hatékony kiszolgálásához virtualizációs technológiákat alkalmaznak. Ezek lehetővé teszik, hogy számos különféle szolgáltatást tudjanak egy-egy szervergépen futtatni kedvező feltételek mellett: a virtualizáció csökkenti a szolgáltatások futtatásához szükséges helyigényt, és jobb erőforrás-kihasználtságot eredményez.

Az így létrejövő nagyméretű – akár több ezer számítógépből álló – infrastruktúrák karbantartása, üzemeltetése kézi megoldásokkal nehezen kivitelezhető. Emiatt jellemzően valamilyen infrastruktúra-menedzsment megoldást használnak a szervezetek, melyekkel központosítottan, egyetlen felületről van lehetőségük az üzemeltetett szerverek kezelésére, azok konfigurációjára.

A megfelelő minőségű üzemeltetéshez elengedhetetlen a számítógépek és az azokon futó szolgáltatások folyamatos felügyelete, monitorozása. Ennek segítségével figyelemmel követhetjük rendszereink állapotát, megfelelő határértékek beállításával pedig időben értesülhetünk az esetleges kialakulóban lévő problémákról, ezzel lehetőséget adva a hibák idejekorán történő javítására és a szolgáltatáskiesések megelőzésére.

Dolgozatomban a nagyvállalatokra jellemző informatikai környezeteket vizsgálom meg egy kis méretű tesztrendszeren keresztül. A tesztkörnyezetben kitérek a fent ismertetett technológiák alkalmazására, az ezek használata során követendő jó gyakorlatokra, és az általam tapasztalt kihívásokra, nehézségekre.

# Abstract

Information Technology (IT) has become an integral part of our lives, enabling us to shop, bank and communicate online. The non-stop operation of an ever-wider range of services requires careful planning, expertise and a high level of preparedness on the part of systems engineers.

Virtualisation technologies are used to efficiently serve the increased traffic. These allow a wide variety of services to run on a single server under favourable conditions: virtualisation reduces the space required to run services and results in better resource utilisation.

The resulting large-scale infrastructures – up to thousands of computers – are difficult to maintain and operate manually. For this reason, organisations typically use some kind of infrastructure management system that allows them to manage and configure the servers they operate from a centralised, single interface.

Continuous supervision and monitoring of the computers and the services running on them is essential for ensuring the quality of operations. This allows us to monitor the status of our systems and, by setting appropriate thresholds, to be notified in time of any problems that may arise, thus providing the opportunity to correct errors and prevent service outages.

In my thesis, I examine IT environments typical of large enterprises through a small-scale test system. In the test environment, I will discuss the use of the technologies described above, the good practices to follow when using them, and the challenges and difficulties I have encountered throughout my work.

# 1. fejezet

## Bevezetés

Az informatika és az internet életünk szerves részévé vált. A számos szolgáltatás folyamatos rendelkezésre állásának biztosítása és a megnövekedett forgalom kiszolgálása jó néhány új technológia kifejlesztését követelte meg.

Dolgozatomban elsősorban egy általános képet szeretnék adni arról, hogy milyen üzemeltetési kihívásokkal kell szembenéznünk, ha egy ilyen szolgáltatás működtetésébe vágjuk a fejszénket. Értekezésemben nem fogok kitérni bizonyos infrastrukturális hátterekre – mint például a kiszolgálók folyamatos energiaellátásának biztosítása –, ezeket adottnak fogom tekinteni, hiszen ezt egy adatközpontban bérelt hely esetén sem magunknak kell biztosítanunk. A következőkben sokkal inkább az informatikai lehetőségek tárgyalására fogom helyezni a hangsúlyt: hogyan tudunk hatékonyan üzemeltetni több kiszolgálót, milyen módon lehet biztosítani a szolgáltatásaink lehető legnagyobb rendelkezésre állását, és hogyan védhetjük meg adatainkat egy esetlegesen félresikerült rendszerfrissítést követően.

A dolgozatban érinteni fogom a jelenleg legelterjedtebb virtualizációs technológiákat, melyek főbb tulajdonságait röviden ismertetem, valamint össze is hasonlítom ezeket a megoldásokat a legfontosabb különbségekre kitérve.

Szerepet fog kapni továbbá a logikai kötetkezelés, amely technológia nagyban megkönnyíti a háttértárak és partíciók kezelését üzemeltetési szempontból. Ezt főként virtualizációt végző fizikai gépek esetében használhatjuk ki, hiszen ilyen helyzetekben érdemes minden virtuális rendszernek külön partíciót létrehozni, amelyek kezelése (pl. egy esetleges bővítés során) a hagyományos particionálási megoldásokkal sokkal összetettebb feladat lenne. Dolgozatomban a Linux kernelben elérhető Logical Volume Manager (LVM) megoldást fogom részletesen ismertetni.



Szót ejtek a felügyeleti (monitoring) megoldásokról is, melyek elengedhetetlenek ahhoz, hogy a rendszer üzemeltetését végző szakemberek pontos képet kapjanak az infrastruktúra aktuális állapotáról, az esetleges korábbi problémákról. A monitorozás azért is fontos, mert ha egy hibát ezáltal sikerül idejekorán felismerni (például háttértárak esetén egy megfelelő határérték beállításával időben értesülhetünk egy partíció megteléséről, és nem csak az írási hibákat tapasztaljuk), akkor elkerülhetőek a további, komolyabb hibák, amik akár a felhasználók számára is fennakadásokat okozhatnak. Az általam létrehozott tesztkörnyezetben is bemutatok egy ilyen monitoring megoldást, melynek segítségével az általam létrehozott infrastruktúra gépeit fogom folyamatosan ellenőrizni.

A tesztkörnyezet beállításában nagy szerep fog jutni a választott konfigurációmenedzsment szoftvernek, a Salt-nak, mely egy Python-alapú automatizációs eszköz informatikai rendszerek kezelésére [16]. Ez arra fog lehetőséget biztosítani, hogy egyes konfigurációs fájlokat egyszerűen telepíthessünk több számítógépre is, valamint a keretrendszer leírónyelvén meghatározott konfigurációleíró szoftver lehetővé teszi szoftvercsomagok telepítését és a klienseken futó szolgáltatások (service) állapotának ellenőrzését is. Ez hasznunkra válhat például egy saját service-szel érkező program telepítését követően, hiszen így a leíróban megadhatjuk a telepítés paramétereit, majd ezt követően egyből ellenőrizhetjük is, hogy a telepítés után sikeresen elindult-e az újonnan telepített szoftver.

A dolgozatban tárgyalt koncepciókat egy kisebb volumenű tesztrendszeren keresztül fogom bemutatni. Ennek a rendszernek a célja nem egy teljes vállalati környezet bemutatása, hiszen ehhez nagy mennyiségű hardverre, jelentős mértékű hardveres és szoftveres erőforrásokra lenne szükség, amelyek üzembe helyezése, összehangolása túlmutat a dolgozat keretein. Ehelyett sokkal inkább arra szeretnék rávilágítani, hogy milyen eszközök állnak rendelkezésre egy ilyen nagyszabású infrastruktúra sikeres üzemeltetésének elősegítéséhez. Gondoljunk csak arra, hogy egy 5-10 számítógépből álló rendszer esetén kivitelezhető, hogy a rendszergazdák egyesével telepítsék a havi frissítéseket, azonban egy több száz, vagy több ezer kiszolgálóból álló nagyvállalati környezetben nem lenne reális elvárás.

Az ilyen és ehhez hasonló kihívások megoldására fogok lehetőségeket mutatni a 3. fejezetben. Szó lesz a gépek távoli kezeléséről, folyamatos karbantartásukról, automatikus biztonsági javításokról (patchek) való értesülésről, ezek telepítéséről. Tárgyalni fogom továbbá a rendszert alkotó eszközök monitorozását, metrikák gyűjtését is, továbbá szó lesz az egyre szélesebb körben elterjedő konténerizációs technológiákról, ezek használatáról vállalati környezetekben. Bemutatom azt is, hogy a megfelelő eszközökkel milyen gyorsan hozhatunk létre konténereket, és mennyire hatékonyan kezelhetjük őket akár egy böngé-

szöből is. Fontos megjegyezni, hogy az itt említett technológiák kisebb környezetekben is használhatóak, azonban néhány esetben az ilyen rendszerek használata kevesebb előnyt nyújt, mint amennyi munkát telepítésük és karbantartásuk igényel, így érdemes felmérni az informatikai rendszerrel szemben támasztott elvárásainkat, és ennek megfelelően dönteni a szükséges technológiai komponensekről.

A 4. fejezetben fogom ismertetni az általam készített tesztkörnyezetet virtualizációs megoldását, ennek felépítését, komponenseit, a tervezési döntéseket, valamint az ezzel kapcsolatos munkáim során felmerült nehézségeket, tapasztalatokat. Ebben a fejezetben a korábban tárgyalt virtualizációs technológiák közül általam választott megoldást fogom részletesebben ismertetni.

Az általam a tesztkörnyezetben használt infrastruktúra-menedzsment megoldás konfigurációjáról és az ebben rejlő lehetőségekről az 5. fejezetben írok részletesen. Itt néhány példán keresztül bemutatom a konfigurációmenedzsmentben, csomagok telepítésében elérhető hatékony megoldásokat, kitérek a kliensek központi kezelésére. Emellett az infrastruktúra-menedzsment szoftver használata során készített mérési eredményeimet is ismertetem.

Nagyvállalati, magas rendelkezésreállási követelményeket kielégíteni tudó rendszer lévén fontos kitérni az erőforrások, kiszolgálók folyamatos figyelésére, monitorozására. Az erre kialakított monitoring rendszert a 6. fejezetben ismertetem részletekbe menően. Kitérek a környezet konfigurációjára, a probléma esetén küldésre kerülő riasztások beállítására, valamint a monitorozott rendszerekről gyűjtött adatok megjelenítésére.

Végül a dolgozat utolsó fejezetében értékelni fogom az elért eredményeket, valamint röviden összefoglalom a projekt továbbfejlesztési lehetőségeit.

## 2. fejezet

# Nagyvállalati környezetek ismertetése

### 2.1. Áttekintés

A vállalatok egyre nagyobb hangsúlyt fektetnek az informatikai rendszereik fejlesztésére és karbantartására, üzemeltetésére. Az ezek által nyújtott szolgáltatások sok esetben jelentős könnyebbséget jelentenek egyes üzleti folyamatokban, és a megfelelően automatizált munkafolyamatok csökkentik az egyes munkavállalók által elvégzendő manuális feladatokat, és adott esetben az ügyfelek számára is segítséget nyújthatnak. Fontos tisztában lenni azonban azzal, hogy ezek a megoldások csak akkor működnek jól a mindennapi használat során, ha sikerül biztosítani a megfelelő rendelkezésre állást, tehát egy – a vállalat munkavállalói által a munkához nélkülözhetetlen – szolgáltatásnak munkaidőben folyamatosan elérhetőnek kell lennie. Előfordulhatnak olyan igények is, amik miatt bizonyos, a vállalat működésének szempontjából nélkülözhetetlen szolgáltatásoknak folyamatosan elérhetőnek kell lenniük, mert a működésük nem munkaidőhöz kötött (ilyen lehet például a szervezet weboldala, illetve levelezőszervere). Emellett a fent említett informatikai rendszerek karbantartásának is sokszor észrevehetetlennek kell lennie, azaz egy esetleges frissítés nem hátráltathatja a munkavégzést és nem csökkentheti a rendelkezésre állást.

Mivel az ilyen rendszerek általában nagy méretűek és összetettek, a tervezésük és az üzemeltetésük is nagy szakértelmet igényel. Éles környezetek tervezése során előtérbe kell helyezni a skálázhatóságot, hogy az adott rendszer esetleges jövőbeli bővítése során legyen lehetőség az elérhető erőforrásokat növelni a szükséges mértékben.

## 2.2. Adatbiztonság

Egy másik fontos szempont a nagyvállalati rendszerek üzemeltetése – és általában informatikai megoldások üzemeltetése és használata során – ami napjainkban egyre nagyobb figyelmet kap, az IT-biztonság kérdése. A rendszerüzemeltetőknek tisztában kell lenniük a potenciális veszélyekkel, veszélyforrásokkal és fel kell készülniük egy esetleges támadásra, annak kezelésére. Sokszor hallhatunk a rendszeres biztonsági mentések fontosságáról, és ezek típusairól, követelményeiről. Egy jól bevett gyakorlat például az úgynevezett 3-2-1 mentési stratégia, ami egy jó kiindulási alapul szolgálhat minden szervezet számára a biztonsági mentésekhez [27]. A megoldás elnevezése az alábbi elvekből származik:

- három példány az adatokról,
- két különböző eszközön (akár más típusú adathordozókon, pl. SSD, HDD, mágnesszalag – ez segít az adathordozóra jellemző esetleges hibák hatásának csökkentésében),
- egy példányt földrajzilag különböző helyen tároljunk (pl. a cég székhelyén legyenek az eredeti adatok és még egy mentés, további egy példányt pedig tároljunk adatközpontban, vagy vegyünk igénybe harmadik féltől biztonságimentés-szolgáltatást) [18].

A biztonsági mentések elvégzésére és automatizálására többféle megoldást választhat a szervezet az igényeihez igazodva. Bevett szokás például, hogy a cégen belüli mentéseket valamilyen mentést támogató vagy akár teljesen automatizáló szoftverrel oldják meg (például Bareos, Bacula, BackupPC). Az ilyen megoldások üzembe helyezése nehezebb lehet, mintha például csak egyéni scriptekkel hajtánánk végre a mentéseket, de hosszabb távon mégis célszerű lehet megfelelően konfigurálni őket, mert könnyebben kezelhetővé teszik egy összetett infrastruktúrában található gépek adatainak mentését, illetve az egyszer megírt konfigurációs fájlok több gépen is felhasználhatóak. Mindezek mellett ezek a szoftverek általában rendelkeznek valamilyen grafikus felhasználói felülettel (pl. webes interfész), amely tovább egyszerűsíti a mentések készítését, valamint szükség esetén a mentés visszaállítását.

## 2.3. Nagyvállalati és végfelhasználói környezetek összehasonlítása

A legtöbb hétköznapi felhasználó számára ismeretlen vagy meglepő lehet, hogy maga az internet és az ezen keresztül elérhető szolgáltatások – gondoljunk például az Ügyfélkapura

vagy az internetbank-szolgáltatásokra – nagyon bonyolult rendszerek nem csak szoftveres, hanem informatikai infrastruktúra szempontjából is. A legtöbb ilyen szolgáltatás egy adatközpontban lévő szerveren fut, ami a beérkező kérésekre ad válaszokat, a felhasználók az adott szolgáltató számítógépeivel kommunikálnak.

Ezek a szervergépek több lényeges különbséggel is bírnak a személyi számítógépekkel szemben. Egyik legfontosabb tulajdonságuk, hogy hibatűrőek bizonyos hardverhibákat illetően: szinte minden főbb komponensből legalább kettő áll rendelkezésre, így ha az egyik meg is hibásodik, akkor a hiba elhárításáig a beépített redundancia miatt a gép képes tovább működni, általában a felhasználók felé észrevétlenül, míg a gép üzemeltetői figyelmeztetést kapnak a hiba típusáról és a kapcsolódó tennivalókról.

A nagyvállalati informatikai környezetek számos további dologban különböznek a végfelhasználói megoldásoktól. Míg egy átlagos hétköznapi felhasználó legfeljebb két-három számítógépet, esetleg nyomtatókat és hálózati eszközöket használ tevékenységeihez, addig a nagyvállalati rendszerek egy sokkal nagyobb eszközparkkal dolgoznak és sokkal több felhasználó kiszolgálását biztosítják. A 24/7-es rendelkezésre állásra tervezett hardverek mellett megfigyelhetünk számos olyan megoldást, melyekkel az otthoni felhasználók nem találkozhatnak. Ilyenek például a fent említett szervergépek, a különböző adattárolási rendszerek és a magasabb szintű hálózati megoldások (pl. VLAN-ok). További különbség még, hogy egyéni felhasználás esetén egy számítógépet általában csak egy-egy személy, esetleg egy család tagjai használnak, míg egy komplex szervezeti infrastruktúrát az azt üzemeltető szakemberek mellett a vállalat munkatársai és néha az ügyfelei is használják.

A nagyvállalati rendszerek a magas rendelkezésreállás biztosítása és a hibalehetőségek minimalizálása érdekében gyakran földrajzilag elkülönítetten, georedundánsan futnak, ezzel szemben egy kisebb léptékű infrastruktúra részei szinte kizárólag egy adott helyszínen találhatóak. A nagyvállalati rendszerek további ismertetőjele, hogy rendszeres és gyakori karbantartási időablakok szerint történik a hardveres és szoftveres komponensek naprakészen tartása, míg egy végfelhasználói rendszer szoftveres karbantartását gyakran maga a szoftver határozza meg, hardveres karbantartás pedig sokszor csak hiba esetén történik.

Láthatjuk, hogy az összetett infrastruktúrák üzemeltetése szakértelmet, tervezést igényel, ezáltal az üzemeltetéshez tartozó költségek is magasabbak. Ezen rendszerek karbantartása előre meghatározott ütemben, professzionális szinten történik. Kisebb rendszerek és végfelhasználói megoldások karbantartása gyakran ad-hoc módon, a felhasználók megítélése alapján megy végbe. Dolgozatomban a továbbiakban a nagyvállalati rendszerek sajátosságait és a kapcsolódó üzemeltetési feladatokat fogom részletesen ismertetni.

## 3. fejezet

# Technológiai áttekintés

### 3.1. Szervergépek

A szerverek esetében jelentkező, egyénitől nagyban különböző felhasználási körülmények a szerverszámítógépek esetén hardveres szempontból is más felépítést igényelnek. A magas rendelkezésre-állás (high availability, HA) és a modularitás, valamint az ezzel járó könnyű javítások támogatása érdekében az ilyen célra kialakított számítógépek főbb komponensei redundánsak, azaz egy-egy ilyen komponens kiesése nem jelent szolgáltatáskiesést. A meghibásodást a rendszer egyértelműen jelzi magán a gépházon is (általában hibajelző LED-ek segítségével), valamint a menedzsment portjain<sup>1</sup> is. A legtöbb ilyen gép ugyanis rendelkezik egy beágyazott rendszerrel, ami lehetővé teszi a távoli kezelésüket egy webes felületen és távoli parancssori eléréssel, SSH-n keresztül még akkor is, ha a szervergép ki van kapcsolva. Ezek lehetőséget biztosítanak a gép legfontosabb mérőszámainak követésére, virtuális kijelző csatlakoztatására, telepítőfájlok felcsatolására, valamint a gép ki- és bekapcsolására.

A fent ismertetett üzemeltetést, karbantartást könnyítő felépítés mellett általában elmondható, hogy az ilyen gépek jelentős része virtualizációra van tervezve – persze ezektől különböző felhasználási módok is jelentkeznek (például fájlserverek tervezése során a teljesítmény helyett a minél nagyobb tárhelykapacitásra és adatátvitelre helyezték a hangsúlyt). A dolgozat szempontjából viszont a nagyvállalati környezetben domináló virtualizációs felhasználási terület lesz a lényegesebb, így a továbbiakban az ilyen számítógépekre (virtualizációs hoszt, virtualization host) fogok koncentrálni.

---

<sup>1</sup>A szervergépek egy különleges interfésze, amely lehetővé teszi a számítógép távoli kezelését és a szerver által karbantartott naplók böngészését.



**3.1. ábra.** Szervergép fogyasztásának grafikonja egy HPE számítógép távoli menedzsment felületén. Vegyük észre a jobb alsó sarokban megjelenő menüt, amivel lehetőségünk van a gép kikapcsolására és újraindítására is.

A virtualizációs hosztgépek jellemzője, hogy számos processzorral rendelkeznek, valamint felhasználói szemmel szokatlanul nagy memóriaterülettel bírnak. Ki fog derülni azonban, hogy 12-24 processzormag és akár több száz gigabyte RAM is szükséges erőforrássá válhat egy virtuális gépeket futtató számítógép esetében, hiszen gyakorlatilag itt egyetlen szervernek kell elbírnia akár több tíz számítógép terhelésével is. Ezek mellett általában több (8-24) háttértár-foglalattal is rendelkeznek, melyekhez hardveres RAID-támogatást is adnak. A RAID-megoldásokkal a 3.3. alfejezet foglalkozik részletesebben.

## 3.2. Virtualizáció

A fent említett megnövekedett forgalom kiszolgálását hatékonyan lehet kezelni úgy, hogy olyan fizikai számítógépet helyezünk üzembe, mely több, egymástól független operációs rendszer futtatására is alkalmas. Ilyenkor ezeket a fizikai gépen futó rendszereket virtuális gépeknek (virtual machine, VM) nevezzük. Egy virtuális gép elkülönített erőforrásokat kap a fizikai géptől, hozzáférhet például bizonyos mennyiségű processzormaghoz, memóriához, illetve külön háttértár-partíciói is lehetnek. A virtualizált hardverek és operációs rendszerek a legtöbb esetben a kívülág felé nem különböztethetőek meg a fizikai számítógépektől, és ezzel a megoldással jelentősen csökkenthető a rendszerek és a hozzájuk szükséges informatikai infrastruktúra üzemeltetésének költsége.

A virtualizáció nagy ereje abban rejlik, hogy bizonyos hardverek virtualizációjával egységnyi teljesítményt olcsóbban kaphatunk meg, mintha külön fizikai gépeket helyeznénk üzembe, illetve nagyobb rugalmasságot kapunk a kezelésükben, üzemeltetésükben. Képzeljük el, hogy megveszünk egy számítógépet, amin szeretnénk futtatni egy számunkra fontos alkalmazást, mondjuk a honlapunkat. Ilyenkor az ezen a gépen futó operációs rendszer teljes mértékben megszabhatja, hogy milyen erőforrásokból mennyit használ. Ha egy másik szolgáltatást – például levelezőszervert – szeretnénk emellett futtatni, akkor korlátozottabbak a lehetőségeink, hiszen a korábban telepített webszerver már foglal bizonyos erőforrásokat, illetve a program függőségeit és konfigurációs fájljait is telepítettük már, ami esetleg negatívan hat a levelezőszerverünk működésére. Ha mindezt virtualizált környezetben tesszük meg, akkor a topológia megváltozik: a két alkalmazás teljesen elkülönítetten, egymás zavarása nélkül, különböző virtuális gépekben futhatnak, ezeket a gépeket pedig a fizikai gépen futó egyik szoftverkomponens, az úgynevezett hypervisor kezeli, mely a gazdagépen futó rendszer legfőbb virtualizációt támogató komponense [21]. A hypervisor látja el az erőforrások ütemezésének és kiosztásának (pl. processzoridő, memória) feladatát, gondoskodik a virtuális gépek számára szükséges hardveres erőforrások virtualizált hardverinterfészekén keresztüli elérhetőségéről.

### 3.2.1. Népszerű virtualizációs technológiák

Mivel a virtualizáció nagyon elterjedt technológia, számos olyan megoldás született, mely egyszerűsíti a virtuális gépek üzemeltetését. Ezek közül nagy ismertségnek örvend az Oracle VirtualBox és a VMware Player, azonban ezek a megoldások nem skálázódnak annyira jól, mint a továbbiakban tárgyalt társaik, melyek sokkal megfelelőbbek nagyvállalati szerverkörnyezetben való alkalmazásra. Ezek a hypervisorok lehetőséget biztosítanak a virtuális gépek távoli elérésre, kezelésére, egyszerűbb telepítésükre, valamint szükség esetén elosztott működésükre. A következőkben három népszerű virtualizációs technológiát fogok bemutatni, összehasonlításuk a 3.1. táblázaton látható.

A hypervisorokat két kategóriába sorolhatjuk Robert P. Goldberg 1973-as publikációja alapján [3]. Az egyes típusú hypervisorok (type-1) natívan, közvetlenül a gazdagépen futnak (pl. a következőkben tárgyalt VMware ESXi), míg a kettes típusba tartozó (type-2), úgynevezett hosztolt hypervisorok egy hagyományos operációs rendszeren futnak más számítógépes programokhoz hasonlóan. Kettes típusú hypervisor például az Oracle VirtualBox. Egyes hypervisorok – mint a KVM – besorolása vitatott, mivel egy kernelmodulként épül bele egy már futó OS-be. Azonban mivel azt így lényegében egy



type-1 hypervisorra alakítja, ezért általában az egyes típusba sorolják [24]. A két típus felépítését a 3.2. ábra mutatja be.



**3.2. ábra.** Type-1 és type-2 hypervisorok felépítése.

Nagyvállalati környezetben elterjedt virtualizációs megoldás például a VMware ESXi, amely egy igen modern hypervisor számos kényelmi funkcióval ellátva (lehetőség van például a rendszer webes felületről való kezelésére és virtuális gépek sablonból való gyors, körülbelül 5-10 perc alatti telepítésére). Az ESXi a részletes beállításokat lehetővé tevő, könnyen kezelhető webes felületének köszönhetően nagy piaci részesedést szerzett, felmérések alapján kb. 60-80%-os jelenléttel uralja a virtualizációs piacot, bár a közel-múltban bevezetett új, jelentősen drágább előfizetési modell némileg csökkenthet ezen az arányon [19] [6]. Egy másik kedvelt megoldás az ESXi-vel ellentétben felhasználási korlátozás nélkül teljesen ingyenesen, GPLv2-es licenc alatt elérhető XEN hypervisor. Ez ugyan kevesebb kényelmi funkciót tartalmaz, de szintén népszerűségnek örvendő széleskörű támogatása, kedvező teljesítménye és szabad szoftver voltából eredő ingyenessége miatt. A XEN a 2014 márciusában kiadott 4.4-es verzió óta stabilan működik együtt a libvirt virtualizációs API-val, amely nagyban megkönnyíti a hypervisorral való kommunikációt a virtuális gépek konfigurálása során [26].

A XEN-hez hasonlóan szabad szoftver licenccel érhető el a Kernel-based Virtual Machine (KVM) is, mely a XEN-nél modernebb megoldásnak tekinthető, és manapság széles körben használják a Linux kernelbe való integráltságának és stabilitásának köszönhetően. Bár maga a KVM nem tartalmaz ilyet, de számos interfész elérhető az ezen keresztül futtatott virtuális gépek kezelésére (például virt-manager és további, a XEN-nél említett libvirt API-t támogató szoftverek), valamint akadnak olyan megoldások is, melyek a KVM-re ala-

pozva nyújtanak szélesebb körű virtualizációs megoldást, ilyen lehet<sup>2</sup> például a Proxmox és a Cockpit.

Megnevezés	VMware ESXi	XEN	KVM
Fejlesztő	VMware LLC	Linux Foundation, Intel	Linux fejlesztői közösség
Licencelés	zárt forráskódú, korlátozott ingyenes verzió, teljes verzióhoz előfizetés szükséges	szabadon hozzáférhető, GPL	szabadon hozzáférhető, GPL
Támogatott architektúrák	x86-64, ARM	IA-32, ARM	ARM, PowerPC, ESA/390, IA-32, x86-64
Hypervisor típusa	Type-1	Type-1	Type-1
Hivatalosan támogatott VM OS-ek	Linux, BSD, Windows, macOS	Linux, BSD, Windows, macOS	Linux, BSD, Windows, macOS
Aktív libvirt támogatottság	Limitált	Igen	Igen
Támogatás	Hivatalos	Közösségi	Közösségi
CPU hot swap	Igen	Igen	Igen
Memória hot swap	Igen	Igen	Igen
Megjegyzés	Könnyű kezelőségének, jó támogatásának köszönhetően napjaink legelterjedtebb virtualizációs platformja. A másik két Linux-közelgi megoldáshoz képest kevesebb illesztőprogram áll rendelkezésre, így szigorúbb hardveres követelményeket támaszt.	Eredetileg egyetemi projektként indult, a 2000-es évek közepén nagy fejlődésen ment keresztül, számos nagy platform (pl. Amazon AWS) építette XEN-re a virtualizációs technológiáját.	Hivatalosan is a Linux kernel része, így folyamatos fejlesztés alatt áll és hatékonyan együtt tud működni a kernellel. Számos virtualizációs platform (pl. Google Cloud Platform) alapjaként szolgál.

**3.1. táblázat.** A tárgyalt virtualizációs megoldások összehasonlítása.

### 3.2.2. Virtuális gépek használatának néhány előnye

A virtualizáció számos előnnyel járhat az infrastruktúra és a kiszolgálni kívánt alkalmazások szempontjából. Az egyik legnagyobb ilyen előny például, hogy a virtuális gépek

<sup>2</sup>A konfigurációtól függően akár többfajta virtualizációs környezet is beállítható, de a KVM az egyik legjobban támogatott.

egymástól izoláltan futnak, azaz nincs közvetlen kapcsolat közöttük, ami biztonsági és kezelési, tesztelési szempontból is kedvező lehet. Egy adott csomag vagy szoftver kipróbálásához például készíthetünk egy teszt virtuális gépet, amit egyszerűen törölhetünk a teszt végeztével – a telepített program eltávolítása hagyományos környezetben futtatva sokkal körülményesebb lenne. Hasonlóan előnyökkel jár, hogy a legtöbb modern hypervisor lehetőséget biztosít bizonyos erőforrások úgynevezett *hot swap*-elésére. Ez azt jelenti, hogy egyes komponenseket (pl. memória, háttértárak) úgy is kicserélhetünk, hogy a rendszert nem szükséges ehhez leállítanunk, így a karbantartás nem jár szolgáltatáskieséssel. A felsoroltakon túl a virtuális gépek néhány további kedvező tulajdonságát szeretném részletesen ismertetni a következő alfejezetekben.

#### **3.2.2.1. Erőforrások testreszabása**

Amikor több tíz vagy több száz szerver üzemeltetéséről van szó, akkor hatványozottan számításba kell vennünk az egyes gépekre jutó költségeket. Virtuális gépek esetén ez azért kedvezőbb egy fizikai gépnél, mert ugyan a nagyvállalati környezetbe szánt szervergépek jelentősen drágábbak a személyes felhasználásra tervezett társaiknál, de akár több tíz virtuális gép egyidejű futtatását is lehetővé teszik. Ezáltal az egy fizikai gépre eső, asztali gépeknél megszokott áramfogyasztáshoz képest jóval nagyobb energiafelvétel sokkal kedvezőbb arányt mutat, ha számításba vesszük a futtatott virtuális kiszolgálók számát is.

Mindezek mellett a nagyvállalati felhasználáshoz tervezett számítógépek jóval hibátűrőbbek, hiszen a főbb komponensek redundánsan lettek kialakítva: ezekből az ilyen szerverekben legalább kettő van, és a rendszer automatikusan képes detektálni a hardveres hibákat, és ezek figyelembe vételével tovább működni.

Előnyös lehet továbbá, hogy a virtuális gépek erőforrásai szabadon módosíthatók, így akár két újraindítás között is változtathatjuk a rendelkezésre álló memória mennyiségét vagy épp a processzormagok számát. Sőt, egyes hypervisorok és operációs rendszerek ezen erőforrások futásidejű megváltoztatását is támogatják bizonyos korlátozások mellett, így gyakorlatilag a fontosabb virtualizált erőforrások is *hot swappelhetőek* tekinthetőek.

#### **3.2.2.2. Pillanatképek**

Egy másik kedvező lehetőség virtuális gépek használata esetén az, hogy pillanatképeket, úgynevezett snapshotokat készíthetünk róluk. Ezek a gépet egy adott pillanatbeli állapotban reprezentálják, és később ezeket visszaállíthatjuk, ha szükségünk lesz rá. Egyes megoldások a memóriakép mentését is támogatják, így akár egy futó gép is könnyen vissza-

állítható. A pillanatképek készítése hasznos lehet például rendszerfrissítések esetén, így ha valamiféle hiba lép fel a frissítés során, vagy egy adott szoftver nem megfelelően működik azt követően, akkor a frissítés előtt készített snapshotra visszaállva újra teljes értékűen üzemelhet a szerver, amíg a frissítés során fellépő hibát el nem hárítjuk.

### **3.2.2.3. Migráció**

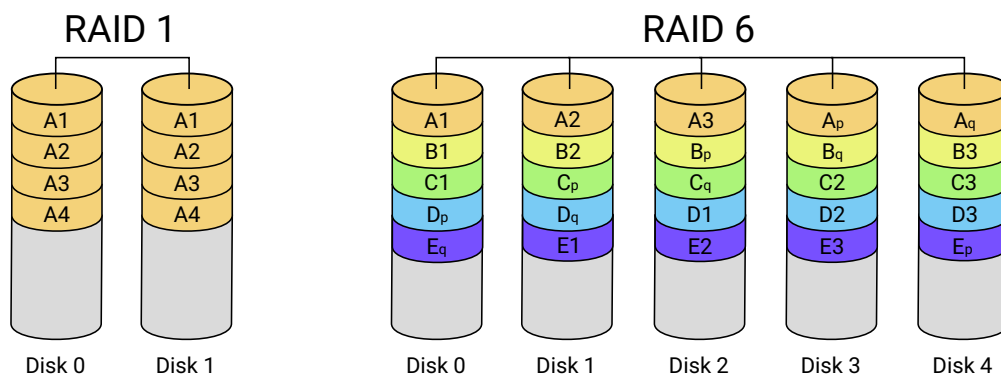
Részben az előző ponthoz kapcsolódik a virtuális gépek migrációja. Ez a funkció azt jelenti, hogy egy adott fizikai gépről, mely virtuális gépeket futtat (virtual host), készíthetünk egy snapshotot, amit áthelyezhetünk egy másik virtual hostra, és a virtuális gép ezen futtat tovább egyéb újrakonfigurálás nélkül. Lehetőség van azonban a háttértárak tartalmát elhagyva is átmozgatni egy VM-et egy másik hosztra. Ehhez bevett szokás leírófájlok használata, mely egy virtuális gép konfigurációját tartalmazza. A leírófájlt egy másik hosztgépre áthelyezve ott újra elindíthatjuk a definiált virtuális gépet. Ilyenkor szükség lehet a VM háttértárainak inicializálására, de ettől eltekintve a konfiguráció szabadon hordozható virtual host-ok között. Ilyen migrációra egyes megoldások fejlettebb támogatást is adnak, így akár valós időben, az aktuális terheltség figyelembe vétele mellett automatikusan is áthelyezhetőek virtuális gépek a megadott fizikai hosztok között.

### **3.2.3. Konténerizáció**

A konténerizáció a virtuális gépektől némileg különböző megoldást használ a szolgáltatások elkülönített futtatására. A motiváció hasonló: egy-egy alkalmazást szeretnénk a gazdagéptől elkülönítetten üzemeltetni. Felmerült azonban az igény, hogy a virtuális gépekhez képest kisebb költsége legyen a szolgáltatások futtatásának. Ezt úgy lehetett csökkenteni, hogy egy teljes virtuális gép létrehozása helyett csak egy minimális izolált környezetet hozunk létre, amely tartalmazza az alkalmazás működéséhez szükséges fájlokat, függőségeit. Az így létrejövő környezeteket konténereknek nevezzük. Egy-egy konténer egyszerűen mozgatható kiszolgálók között, és az adott alkalmazás függőségei egységbe zárásának köszönhetően a gazda operációs rendszertől függetlenül szinte bármilyen OS-en futtatható. A technológia egyre nagyobb teret hódít meg, a népszerű konténerizációs megoldások közé tartozik a Docker, a Podman és a témához szorosan kapcsolódik a népszerű konténer-orkesztrációs platform, a Kubernetes is. Bár a következőkben tárgyalt fejezetekben lesz szó a konténerizációról, és a tesztkörnyezetben egy konténerhoszt-VM-et is telepítettem, melyben hoztam létre konténert, magára a technológia alkalmazására az itt ismertetettnél nem térek ki részletesebben.

### 3.3. RAID

Nagyvállalati környezetben nem hagyhatjuk ki a Redundant Array of Independent Disks (RAID) megoldásokat (a népszerű RAID 1-et és RAID 6-ot a 3.3. ábra szemlélteti), ha biztonsági mentésről beszélünk. Ezek arra adnak lehetőséget, hogy az adatokat több fizikai háttértáron (pl. merevlemez vagy SSD) tároljuk úgy, hogy egy esetleges lemezhiba ne okozzon fennakadást a működésben. Fontos tisztában lenni azonban azzal, hogy a RAID-megoldások nem védenek bizonyos veszélyek ellen (például zsarolóvírusok, fájlok korrupciója), hiszen az adatok duplikálása valós időben történik, így egy esetleges támadás során a RAID poolba<sup>3</sup> bevont összes diszken megváltoznak az adatok, így nem alkalmas a támadás utáni visszaállításra. Emiatt egy RAID pool a 2.2. alfejezetben részletezett 3-2-1 mentési stratégiát alkalmazva csak egyetlen eszköznek tekinthető, hiába több lemezt használunk a mentés során. RAID-elést tehát csak hardveres hibák ellen érdemes használnunk, rosszindulatú támadás esetén ezek nem nyújtanak védelmet az adataink számára.



3.3. ábra. RAID 1 és RAID 6 megoldások felépítése [25].

### 3.4. Logikai kötetkezelés

Mind a fizikai, mind a virtuális gépek esetén szükség lehet háttértárakra az adatok perzisztens tárolása érdekében. Hagyományos particionálási megoldásokkal hamar nehezen kezelhetővé válhatnak a különböző csatolási pontok<sup>4</sup> és a virtuális gépek számára kiosztott kötetek. Az ilyen problémák elkerülésére jött létre a logikai kötetkezelés, mely a tárhely-virtualizáció egy formája. A logikai kötetkezelésnek több implementációja létezik. Ezek

<sup>3</sup>RAID poolnak nevezzük azon fizikai kötetek összességét, amelyek együtt egy RAID-kötetet adnak, például a 3.3. ábrán a RAID 1 és RAID 6 kötetet adó háttértárak egy-egy RAID poolt alkotnak.

<sup>4</sup>Unix-alapú operációs rendszerekben azokat a könyvtárakat nevezzük csatolási pontoknak, amelyeken keresztül elérhetjük az adathordozók, lemezképfájlok tartalmát.

közül jelenleg a Linux kernelben elérhető Logical Volume Manager (LVM)-et fogom részletesen ismertetni.

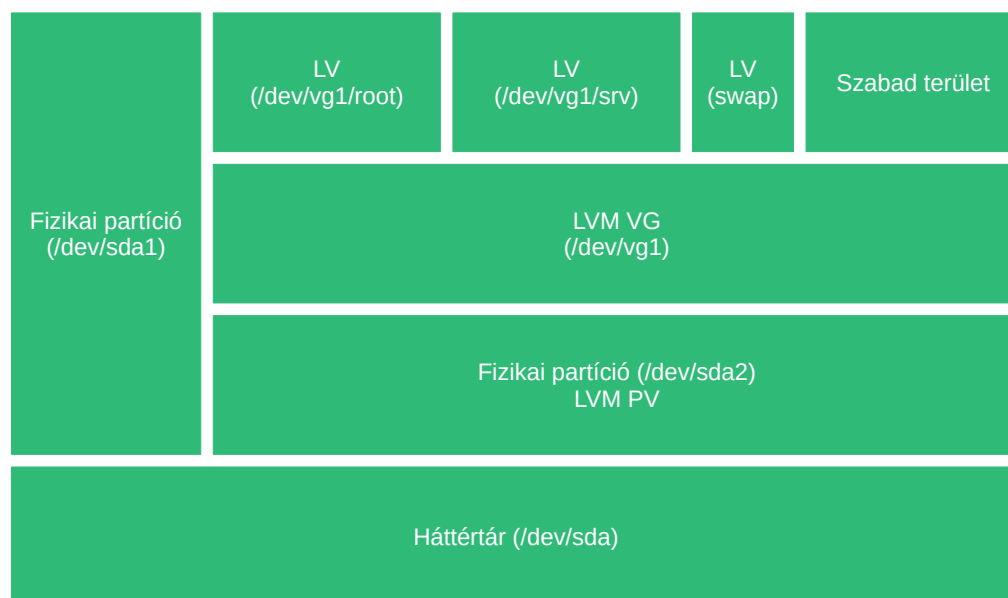
A Linux logikai kötetkezelője három lényegi rétegből áll: a fizikai kötetből (physical volume, PV), a kötetcsoportból (volume group, VG) és a logikai kötetekből (logical volume, LV). Ezt a felépítést a 3.4. ábra szemlélteti egy egyszerű LVM-konfiguráción keresztül. Lehetőség van ennél összetettebb kötetkiosztás létrehozására is, például egy kötetcsoport több fizikai kötetből is állhat, amik akár külön háttértáron is lehetnek, sőt, RAID-csoportot is megadhatunk egy LVM-partíció alapjául. Ezen megoldások használata azonban sok hátránnyal járhat (pl. diszkhiba esetén nehezebb visszaállítani a partíciót), ezért ennek használata alapvetően nem ajánlott [14].

Az LVM tehát úgy épül fel, hogy egy vagy több háttértáron létrehozunk hagyományos fizikai partíciókat, melyek az LVM PV-k alapjául fognak szolgálni. Ezt követően létrehozuk a kötetcsoportokat az általuk használandó LVM fizikai kötetek megadásával. Az így létrejött csoportban már tudunk létrehozni logikai köteteket, amíg van szabad hely a VG-ben.

Láthatjuk, hogy az LVM-kötetek használata kezdetben több feladattal jár, mint a hagyományos partíciók esetében, azonban hosszabb távon számos előnnyel jár. Talán a logikai kötetkezelés legnagyobb előnye, hogy szabadon foglalhatunk le tárterületet a létrehozott köteteknek: ha azt tapasztaljuk, hogy az egyik kötetben kevés a szabad hely, akkor fájlrendszer-től függően elég lehet akár egy parancs kiadása is ennek kiterjesztéséhez. Lényeges, hogy a hagyományos partíciók használatával ellentétben a logikai kötetkezelés használatakor figyelmen kívül hagyhatjuk a partíciók elhelyezkedésének sorrendjét, így nem szükséges figyelembe vennünk, hogy az adott partíció előtt vagy után van-e szabad tárterület. A megnövelt kötet helyes fizikai háttértárra képzéséről a logikai kötetkezelő fog gondoskodni számunkra. Fontos megjegyezni, hogy a kötetbővítés online is elvégezhető, azaz nem szükséges a kötetet lecsatolni a gépről az átméretezéshez. Ez különösen fontos lehet például a root (/) partíció növelése során, hiszen ezt csak a számítógép leállítása mellett tudjuk biztonságosan lecsatolni. Előállhat olyan helyzet is, hogy egy másik (nem root) partíciót kell online átméreteznünk, például ha azt tapasztaljuk, hogy egy adatbázisszerveren hirtelen nagy mértékben nőtt a tárolt adat mérete. Ilyenkor nincs lehetőség a szerver leállítására, hiszen ez esetben az alkalmazások nem tudnák használni az adatbázist a leállítás idejére. Az ehhez hasonló helyzetekre is jó megoldást nyújt a logikai kötetkezelő egy megfelelő, online átméretezést támogató fájlrendszer (pl. XFS, Btrfs) használata mellett. Érdeemes megjegyezni, hogy bár az LVM és például a Btrfs-fájlrendszer nyújt támogatást

a növelésen kívül a fájlrendszer méretének csökkentésére is, ez a művelet általában nem biztonságos, és adatvesztéshez vezethet. Emiatt érdemes eleinte csak kisebb tárterületet adni a köteteknek, hiszen kiterjeszteni sokkal egyszerűbb őket, mint csökkenteni a méretüket. Ennek megkönnyítésére is ad lehetőséget az LVM, megadhatjuk, hogy egy kötet egy bizonyos arányú tárhelyhasználat után automatikusan bővüljön, így elkerülve annak betelését.

Az LVM hasznos funkciói közé tartozik még a kötetpillanatképek (volume snapshots) készítésének lehetősége. Ez azt jelenti, hogy a kötetkezelő képes az adott kötet adott pillanatbeli helyzetének rögzítésére, és erre a verzióra szükség szerint visszaállhatunk (rollback). Ez hasznos lehet például nagyobb konfigurációs változások eszközölése esetén, gyorsan változó adatokkal dolgozó rendszerek (pl. adatbázisszerver) biztonsági mentéseinek készítése során, illetve rendszerfrissítések előtt.<sup>5</sup>



**3.4. ábra.** Egyszerű LVM-kötetkezelési hierarchia.

### 3.5. OS-lehetőségek

Egy nagyvállalati informatikai infrastruktúrában nagy szerepe van a választott operációs rendszernek is, ugyanis nem mindegy, hogy a több száz számítógépből álló rendszerünket mennyire hatékonyan tudjuk karban tartani, egy kritikus biztonsági frissítést milyen hamar tudunk telepíteni az érintett eszközökre, és probléma vagy különleges igény esetén milyen

<sup>5</sup>Egyes eszközök és operációs rendszerek (pl. openSUSE-verziók a snapper-rel (<https://doc.opensuse.org/documentation/leap/reference/html/book-reference/cha-snapper.html>)) automatikusan készítenek snapshotot a frissítések telepítése előtt, így hiba esetén visszaállhatunk a frissítés előtti verzióra.

támogatásra számíthatunk a szoftvereinket illetően. Ezeket a szempontokat figyelembe véve manapság elsősorban a Debian, Ubuntu, Red Hat Enterprise Linux és SUSE Linux Enterprise disztribúciók közül választanak a vállalatok.

A Debian stabilitása miatt népszerű választás elsősorban kisebb (néhány tíz gépből álló) infrastruktúrák esetében, viszont a stabilitás az elérhető csomagok verzióinak rovására megy, általában a legújabbnál néhány verzióval régebbi csomagokat szállítanak a disztribúcióval. A Debian előnye, hogy teljesen szabadon elérhető, és bár nincs hozzá hivatalos támogatás, harmadik féltől vásárolhatunk ilyen szolgáltatást.

Az Ubuntu egy Debian-alapú operációs rendszer, melyet a Canonical Ltd. fejleszt, és vállalati támogatást is nyújt az OS-hez amellet, hogy az alapverzió ingyenesen érhető el. Előnye, hogy mivel mind szerver, mind pedig asztali környezetben elterjedt rendszer, számtalan projekt és gyártó adja ki a szoftvereit Ubuntu rendszerekre.

A Red Hat és a SUSE Linux-verziók már inkább egy magasabb kategóriát céloznak meg: fő célközönségük a több száz, illetve több ezer gépes környezetet üzemeltető vállalatok, és a fent említett két disztribúciónál alapesetben (a legkisebb támogatási csomagban) is szélesebb körű támogatást biztosítanak az operációs rendszerekhez. Kiemelendő, hogy ez a két disztribúció egyedülálló a biztonság területén: számos biztonsági tesztnek vetették alá őket különböző szervezetek (köztük például kormányzatok és IT-biztonságra specializálódott cégek is), melyeket követően a kereskedelmi forgalomban lévő Linux-disztribúciók közül a legmagasabb minősítéseket és tanúsítványokat kapták meg ezek a rendszerek [13] [20].

Lényeges különbség még, hogy az utóbbi két operációs rendszer RPM-alapú csomagkezelőt használ, mely a Debian és Ubuntu által használt DEB formátumhoz képest több lehetőséget biztosít például javítások (patchek) telepítésére. Emellett ez a formátum általában jobb támogatottságot élvez vállalati szoftverek esetében, ezért ezekben a felhasználási körökben az RPM-csomagokat használják a DEB-csomagokkal szemben.

### 3.6. Infrastruktúra-menedzsment

Komplex infrastruktúrák esetén egyre nehezebbé válik a szerverek konfigurációjainak karbantartása, a frissítések kezelése. Manapság már széles körben elterjedtek az infrastruktúramenedzsment-megoldások, melyek lehetőséget biztosítanak ezen problémák kiküszöbölésére. Alkalmazásukkal hatékonyabbá tehető a számítógépek szoftveres karbantartása, könnyen egységesíthetőek a konfigurációs állományok, és így egyszerűbben kezelhetővé válnak az azonos szerepű számítógépek, rendszercsoportok.



### 3.6.1. Ansible és Salt

Dolgozatomhoz az Ansible-t és az Uyuni alapjául szolgáló Salt-ot vizsgáltam meg közelebbről. Mindkét megoldás elterjedtnek tekinthető, azonban a felépítésük nagyban különbözik.

Az Ansible sikere az egyszerűségében rejlik: az Ansible szerveren, az úgynevezett *Control Node*-on kívül nincs szükség további komponensek telepítésére az alapvető funkciók használatához. Az Ansible nem használ dedikált kliensszoftvereket, ezért *agentless*-nek nevezik, működése a push modellen<sup>6</sup> alapul. A feladatok végrehajtását, konfigurációs fájlok elhelyezését úgynevezett Playbook-okkal adhatjuk meg, melyek YAML-ben írt leírófájlok. Egy-egy ilyen fájl több, egymástól független, tetszőleges komplexitású feladatot definiálhat. A *Control Node* ezeket SSH-n keresztül hajtja végre, ehhez a kliensek SSH-kulcsát vagy jelszavát kell megadnunk. A megoldás előnye, hogy így az azonosítás és a kommunikáció titkosságának fenntartása is jelentősen egyszerűsödik, hiszen egy már jól bevált, biztonságos komponensen alapul [10].

Az Ansible-lel szemben a Saltot a kezelt rendszerekre is szükséges telepíteni. A Salt felépítése két kulcsfontosságú komponensre osztható: a Salt Masterre és a Salt Minionokra (emiatt *agent-based* architektúrának nevezik). Minionoknak a kliensekre telepített komponenszt nevezzük, mely a későbbiekben az úgynevezett Salt State-ekben meghatározott feladatok végrehajtásáért lesz felelős, melyeket a Master delegál ki a kliensek számára. A feladatleírók, a State-ek a Salt esetében is YAML-alapúak. A Salt architektúrájában a kezdeti kapcsolatfelvételt a minion indítja a publikus RSA-kulcsának elküldésével, ezt manuálisan kell ellenőrizni és elfogadni a Masteren. Ha a kulcs elfogadásra kerül, a szerver is elküldi a saját publikus kulcsát a Minionnak [17]. Ezt követően a Salt Master már képes a bevont Minionnak végrehajtandó feladatokat küldeni, valamint a Minion állapotát lekérdezni [10].

## 3.7. Felügyelet

Informatikai környezet üzemeltetése során fontos valós időben tisztában lennünk az infrastruktúrát alkotó rendszerek állapotával. Ehhez nyújtanak megoldást a felügyeleti (monitoring) szoftverek, melyek folyamatosan figyelemmel követik a számítógépek fontosabb mérőszámait, ezeket általában a hibafelderítés könnyítése miatt meghatározott ideig tárol-

---

<sup>6</sup>A kliens-szerver kommunikációban kétféle modellt különböztetünk meg attól függően, hogy a kliens vagy a szerver kezdeményezi a kommunikációt. Pull modell esetén a szerver kéri le az adatokat a kientől.

ják is, valamint gyakran képesek a metrikák vizuális megjelenítésére, emellett beállíthatjuk azt is, hogy probléma esetén valamilyen formában (pl. e-mail) értesítést kapjunk a hibáról.

### 3.7.1. Icinga és Prometheus Grafana vizualizációval

Monitoring megoldások közül az Icingát és a Prometheus ismertetem részletesebben. Mindkét megoldás széles körben elterjedt, jó a támogatottságuk és sok kiegészítő érhető el hozzájuk.

Az Icinga a Nagios projekt forkjaként<sup>7</sup> jött létre, ennek következtében a legtöbb Nagios-hoz készült beépülő modullal kompatibilis, ami nagy előny lehet a Nagios használó, modernebb megoldást keresők számára. Az Icinga elsődleges célja szolgáltatások elérhetőségének ellenőrzése, de a számtalan elérhető extra modullal gyakorlatilag minden monitorozási feladat megoldható. Korszerű webes felülettel rendelkezik, melyről elvégezhető a monitoring konfigurációja, de lehetőség van parancssori felületről is módosítani a leírókat. A monitorozás többféleképpen történhet: lehetőség van úgynevezett Icinga Agent kliensekre való telepítésére, pull modell-alapú folyamatos lekérdezésre, illetve programozási felületen, API-n keresztül is képes együttműködni más rendszerekkel [4]. Az Icinga alapértelmezetten MySQL adatbázisban<sup>8</sup> tárolja a gyűjtött adatokat. A rendszer képes e-mailben értesítéseket küldeni a monitorozott rendszerek állapotáról.

A Prometheus egy néhány évvel fiatalabb, önálló monitoring-projekt. A működése HTTP-kéréseken alapul, melyeken keresztül a Prometheus szerver folyamatosan lekérdezi a kliensek aktuális állapotát (pull modell). Ehhez a monitorozott klienseken úgynevezett exportereket kell beállítanunk. Ezek a komponensek a rendszer egy adott szolgáltatását figyelik, és külön-külön porton teszik közzé ezeket. Egy-egy kliens több exporterrel is rendelkezhet. A Prometheus az összegyűjtött adatokat egy speciális, idősoros (time-series) adatbázisban tárolja. Az Icingához hasonlóan képes értesítések küldésére, ehhez az Alertmanager komponenst használja. A Prometheus önmagában nem képes adatok megjelenítésére, viszont rendkívül jól tud együttműködni a Grafana adatvizualizációs eszközzel, melyben rengeteg lehetőségünk van a monitorozott adatok megjelenítésére.

---

<sup>7</sup>Nyílt forráskódú szoftverek esetén forknak nevezzük azt a projektet, amely egy másik, szabadon elérhető megoldást alapul véve, azonos forráskódból jött létre, de a két projekt fejlesztése egymástól szétvált.

<sup>8</sup>A MySQL egy elterjedt, nyílt forráskódú relációs adatbáziskezelő-rendszer.

## 4. fejezet

# Virtualizációs környezet létrehozása

### 4.1. Kialakítani kívánt környezet meghatározása

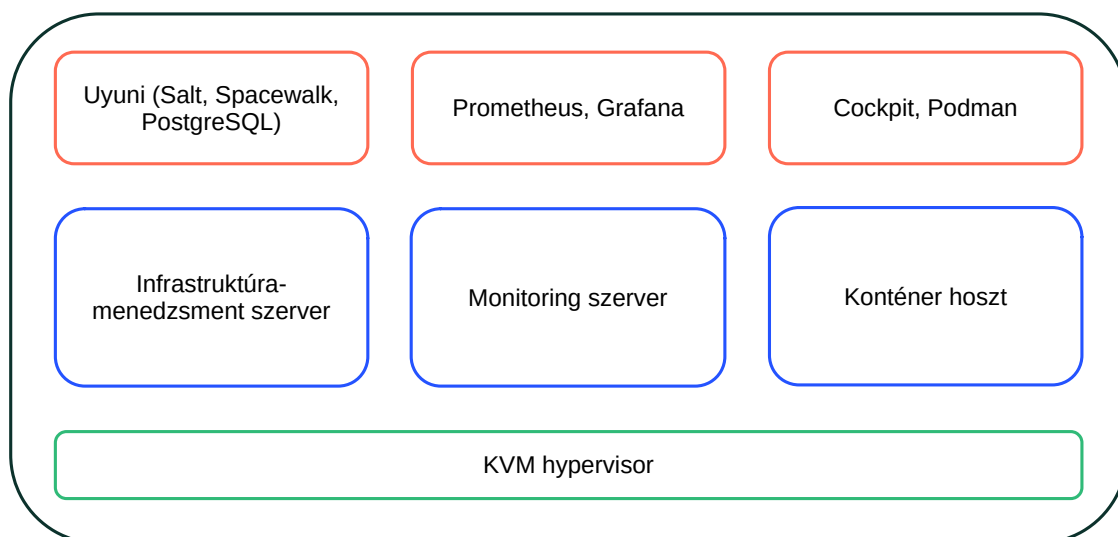
Dolgozatomban egy kisebb léptékű, de a fontosabb elvek ismertetését kellő mértékben lehetővé tevő tesztkörnyezetet fogok kialakítani és részletesen bemutatni. A tesztkörnyezetben egy fizikai gépen (virtual host) fogok virtuális gépeket kialakítani a KVM hypervisor és a libvirt virtualizációs API segítségével. Ezen környezet célja, hogy betekintést engedjen a nagyvállalati környezetekben alkalmazott virtualizációs rendszerek kialakításának fontosabb lépéseibe.

A KVM-re és a libvirt-re azért esett a választásom, mert ezek modern technológiáknak tekinthetők, az elmúlt 20 évben jöttek létre, és a mai napig aktívan fejlesztik őket. A KVM a Linux kernel része, így a támogatottsága egyedülálló, és lényegében minden Linux disztribúción használható. Emellett több nagy szoftvergyártó és felhőszolgáltató (pl. Google, Red Hat) is a KVM-re építi a saját infrastruktúráját, így a technológia jövője is biztosnak tekinthető [15] [1]. A libvirt a virtuális gépek könnyű kezelhetőségében segít, mivel az API-t több fontos virtualizációt kezelő szoftver (pl. virt-manager, virsh, virt-viewer) is implementálja, így egyaránt biztosított a VM-ek grafikus és a konzolos felületen való kezelése is. Ezek mellett a libvirt számos további kedvező lehetőséget biztosít. Lehetőség van például a virtuális gépek által használt háttértár-partíciók méretének online növelésére, VM-leíró XML-ek generálására, melyek megkönnyítik a virtuális gépek létrehozását, valamint a gépek másik hosztgépre történő áthelyezésében is könnyebbséget jelentenek.

Ezen technológiák lehetőségeit figyelembe véve a tesztkörnyezettel szemben az alábbi elvárásokat támasztottam:

- legyen alkalmas nagyvállalati igények kielégítésére, egy olyan infrastruktúra jöjjön létre, ami nagyvállalati környezetben is megállná a helyét,
- mutassa be a virtualizációhoz és a virtuális rendszerek üzemeltetéséhez kapcsolódó jó gyakorlatokat (pl. particionálás, LVM kötetkiosztás),
- legyen képes a virtualizált OS-környezetben futó programok mellett konténerizált alkalmazások futtatására is,
- legyen központilag kezelhető infrastruktúramenedzsment szoftver segítségével, nyújtson lehetőséget konfigurációs fájlok egységes telepítésére,
- a rendszer működése, teljesítménye legyen jól nyomon követhető monitoring rendszeren keresztül.

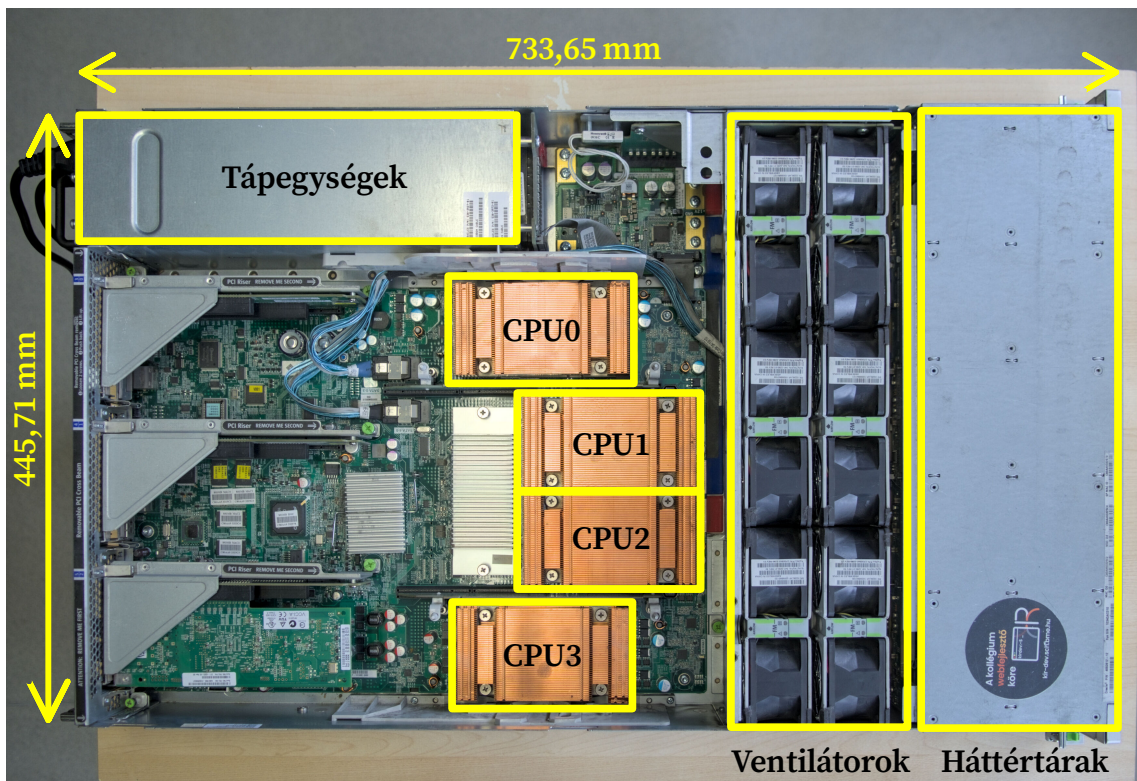
Az így meghatározott tesztkörnyezet felépítését a 4.1. ábra mutatja be. Az ábrán jól elkülöníthetően jelennek meg az architektúra egyes rétegei: legalul helyezkedik el a hypervisor, melyre a középső szinten lévő virtuális gépek épülnek, és legfelül láthatóak az alkalmazásréteg elemei, melyek az alattuk elhelyezkedő virtuális gépeken futnak.



**4.1. ábra.** A tesztkörnyezet tervezett felépítése.

## 4.2. Fizikai gép ismertetése

Ahogy arról a 3.1 alfejezetben már írtam, a szervergépek több lényeges tulajdonságukban is eltérnek a személyi számítógépektől. A virtualizáció szempontjából legfontosabb ilyen különbségek a processzormagok száma és a memória mennyisége. A tesztkörnyezetet szerettem volna egy ilyen gépen megvalósítani, hogy az ténylegesen a lehető legközelebb állhasson egy valós felhasználási környezethez. Bár a lehetőségeim korlátozottak voltak, sikerült beüzemelnem egy régi, Sun Fire X4450 típusú szervergépet. Ez négy fizikai CPU-val rendelkezik, mind a négy processzor 6-6 magot tartalmaz, így összesen 24 maggal gazdálkodhattam. Emellett a gép 64 GB memóriával van felszerelve, és egy 1 TB-os SSD-meghajtó található benne. Ezek mellett a korábban említett hardveres redundancia is megjelenik a gépben: két tápegysége és négy hálózati csatlakozója van, továbbá távolimenedzsment-porttal is rendelkezik, mely lehetővé teszi a szerver távolról történő ki- és bekapcsolását, illetve a rendszernaplók böngészését.



**4.2. ábra.** A tesztkörnyezetben használt fizikai gép. A fotón megfigyelhető a moduláris felépítés, a memóriatálcát eltávolítva pedig a négy különálló CPU is láthatóvá válik. A gép magassága 87,85 mm, tömege pedig 25,6 kg.

### 4.3. Operációs rendszer

Értekezésemben nagy szerepe lesz a választott operációs rendszereknek, hiszen ezek fognak a virtualizációs rendszer alapjául szolgálni, valamint képesnek kell lennünk a gépek távoli menedzsmentjére is, így mindenképpen olyan megoldásra van szükség, amely jól támogatott a választott infrastruktúramenedzsment-eszköz által. Fontos szempont volt továbbá, hogy a tesztkörnyezet a lehetőségekhez mérten jól képviselje a nagyvállalati környezetben használatos rendszereket, így sok olyan OS-verzió kikerült a lehetőségek közül, amelyek ugyan népszerűek például asztali megoldásként, de egyes nagyvállalati szoftverek (legyen az adatbázismotor, vagy bizonyos eszközvezérlők, driverek) hivatalosan nem támogatottak rajtuk. Emiatt az operációs rendszerek kiválasztása során körültekintően jártam el, több Linux-disztribúció is szóba került, az ezekről született konklúziót itt foglalom össze néhány mondatban.

#### 4.3.1. OS-kiválasztás folyamata

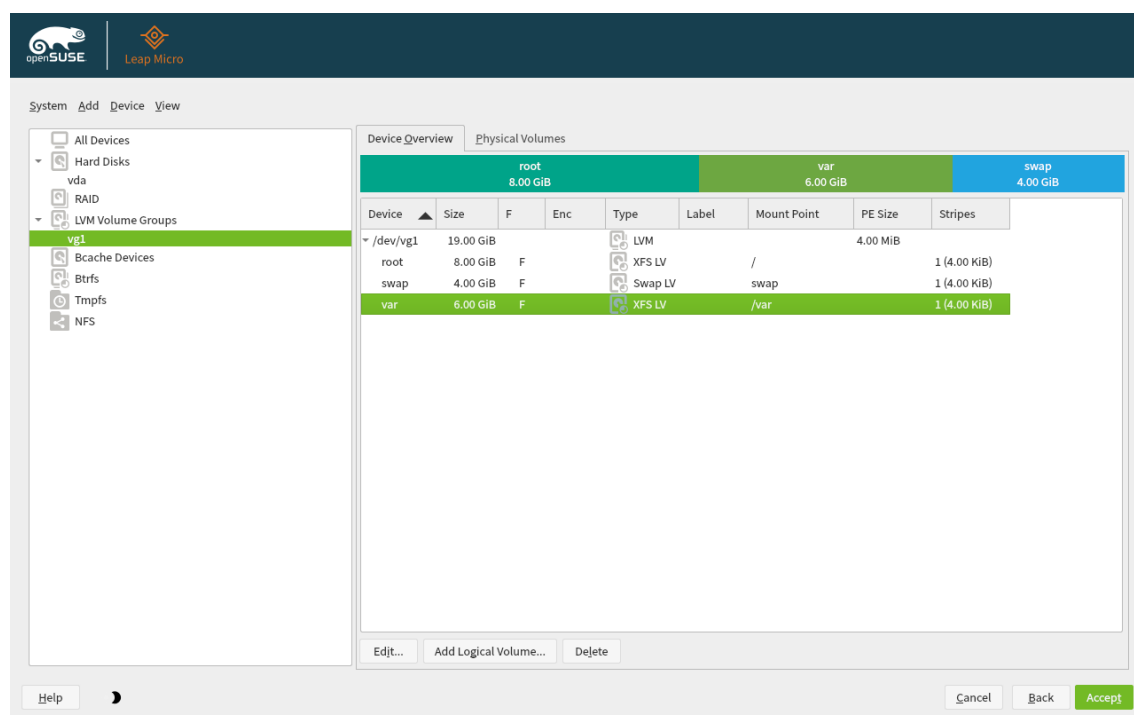
Ahogy a 3.5. alfejezetben is kitértem rá, a nagyvállalatok elsősorban a Red Hat és a SUSE Linux-disztribúciók közül választanak, hiszen ezeknek a velük együtt járó támogatás és a szoftvercsomagok széleskörű támogatottsága miatt kényelmesebb és hatékonyabb az üzemeltetésük, valamint biztonsági szempontból is kedvezőbbek (például gyorsabban kapnak meg bizonyos frissítéseket, patcheket). Szintén jobban támogatottak ezeken a rendszereken a különböző felhasználásspecifikus modulok, például high availability (HA), live patching (támogatás pl. kritikus kernel biztonsági javítások telepítése a számítógép újraindítása nélkül) és real time computing (valós idejű, nagy időbeli pontosságot igénylő alkalmazások futtatására alkalmas környezet).

A fent ismertetett szélesebb körű támogatottság miatt a tesztkörnyezethez használni kívánt operációs rendszerek köre a Red Hat-re és a SUSE Linuxra korlátozódott. A végső döntésben végül az alábbi szempontok segítettek:

- a tesztkörnyezetet szerettem volna egy ökoszisztémán belül tudni mind a virtuális gépeket futtató, mind pedig az azokon futó OS-ek esetében,
- könnyebb konfigurálhatóság: mivel több gépet kellett telepíteni, így fontos szerepe volt annak, hogy egy-egy operációs rendszer telepítése milyen bonyolultságú,

- a környezetet a költségek minimalizálása mellett szerettem volna létrehozni, így lényeges szempont volt, hogy az adott rendszerhez ne kelljen előfizetést vásárolni, mégis a lehető legközelebb álljon a kereskedelmi forgalomban kapható termékekhez.

Mindezek figyelembevételével és korábbi tapasztalataim alapján a SUSE termékcsaládja mellett döntöttem. A támogatással rendelkező, előfizetési modellt használó nagyvállalati változat mellett szabadon beszerezhető openSUSE operációsrendszer-család megfelelt a tesztkörnyezettel szemben támasztott elvárásaimnak. A rendszer telepítését és a későbbi konfigurációt a YaST keretrendszer segíti, mely számos moduljával (pl. particionálás, hálózati és tűzfalbeállítások) nagyban hozzájárul a gépek könnyebb beállításához, kezeléséhez. A YaST – mivel szervereken való használatra tervezték, melyek gyakran nem rendelkeznek grafikus felülettel – a 4.3 ábrán látható megjelenés mellett egy konzolos, GUI-szerű (GUI-like) felülettel is rendelkezik, így a konfiguráció kényelmesen elvégezhető konzolos hozzáférés, például SSH használata esetén is.



**4.3. ábra.** LVM-kötetek létrehozása openSUSE Leap Micro telepítése során grafikus YaST telepítő segítségével.

Az openSUSE-projekt több operációs rendszert is fejleszt<sup>1</sup>, ezek közül én a tesztkörnyezetben kettőt használtam, melyeket a következő alfejezetekben ismertetek.

<sup>1</sup><https://get.opensuse.org/>

#### 4.3.1.1. openSUSE Leap

A Leap egy hagyományos értelemben vett szerver operációs rendszer. Gyakran kap biztonsági frissítéseket, új verziói pedig körülbelül évente jelennek meg. Alapjául a SUSE Linux Enterprise (SLE) szolgál, melynek előnye, hogy a két rendszer csomagjai binárisan kompatibilisek egymással, azaz egy SLE-rendszerre készített csomag garantáltan használható openSUSE Leap-en is, és fordítva [2] [5]. Utóbbi előnye, hogy így számos, a közösség (akár a hivatalos openSUSE projekt, akár a felhasználók) által készített csomagot használhatunk a SLE-alapú rendszerünkön is, bár ehhez nem kapunk hivatalos támogatást.

A nagyvállalati rendszerből való leszármazás másik nagy előnye, ami fontos volt számomra a kiválasztási folyamat során, hogy így gyakorlatilag a SUSE Linux Enterprise egy ingyenes verzióját használhatom, mely lényegében teljesen megegyezik a vállalati környezetben használt megoldással, és előfizetés nélkül is kap frissítéseket, így folyamatosan naprakészen tartható. A biztonsági javításokat illetően fontos megjegyezni, hogy a Leap rendelkezik egy olyan csomagforrással (repository) is, mely a SUSE Linux Enterprise-ban is elérhető frissítéseket tartalmazza, így az ott hozzáférhető fontos javításokat is telepíthetjük a Leap-et futtató rendszereinkre [7].

#### 4.3.1.2. openSUSE MicroOS

A MicroOS egy újfajta megközelítést használó, modern operációs rendszer, mely elsősorban konténerizált alkalmazások futtatásához készült. Az OS előnye, hogy az alap telepítés csak egy minimális szoftvercsomagot tartalmaz, így az erőforrásigénye elenyésző. A MicroOS egy írásvédett (read-only) BTRFS fájlrendszerű gyökérkönyvtárral rendelkezik, melynek előnye, hogy magas szintű támogatást nyújt fájlrendszer-pillanatképek (filesystem snapshots) kezelésére. Erre a technológiára épít a MicroOS filozófiája: atomi frissítéseket támogat, ami azt jelenti, hogy egy csomag vagy frissítés telepítése során nem az éppen használatban lévő partíció változik, hanem egy új snapshotba kerülnek a módosítások, mely – amennyiben a módosítás sikeresen lezajlott – a következő bootolási folyamat során aktívvá válik, és az OS erről kerül betöltésre, így ekkor már használhatjuk a telepített csomagokat. Az atomi frissítések lényege, hogy a módosítások csak akkor lépjenek életbe, ha a teljes folyamat hiba nélkül futott le, azaz például ha egy művelet során a módosítandó 100 csomagból akár csak egy nem tud települni valamilyen hibából eredendően, akkor a teljes telepítés meghiúsul, ezzel elkerülve azt, hogy a rendszer inkonzisztens állapotba kerüljön. A MicroOS ezáltal képes biztosítani azt, hogy a rendszerünk mindig használható állapotban legyen.



A snapshotok fontos tulajdonsága, hogy mindaddig, amíg nem kerülnek törlésre, használatukkal a rendszer bitről bitre visszaállítható abba az állapotba, amiben a pillanatkép készítésekor volt. Ennek nagy jelentősége lehet egy félresikerült rendszerfrissítést követően, hiszen a korábbi állapotra visszaállva a rendszer zavartalanul folytathatja a működést a hiba elhárításáig. A probléma okának felderítését segíti a snapshotok felcsatolásának lehetősége: ez azt jelenti, hogy a BTRFS fájlrendszer képes arra, hogy a éppen használt partíció mellett az ahhoz tartozó pillanatképeket is felcsatoljuk, sőt, a két állapotot össze is vethetjük a verziókezelő rendszerekben megszokott módon (erre például a YaST beépített támogatással rendelkezik), mely tovább könnyítheti a hiba forrásának felderítését.

A MicroOS különlegességei közé tartozik még, hogy a szerver operációs rendszerek-nél megszokott konzolos és távoli asztalos elérés mellett egy webes felületet is biztosít a rendszer kezelésére. Ehhez a Cockpit adminisztrációs rendszert használja, mely az utóbbi években egyre nagyobb népszerűségnek örvendő megoldás. A Red Hat disztribúciói például már ezt a rendszert ajánlják a virtualizáció kezelésére a korábban megszokott virt-manager helyett [11].

A Cockpit felülete gyors áttekintést nyújt a rendszer állapotáról, továbbá könnyíti a konténerek létrehozását (4.4. ábra) és kezelését. A fontosabb metrikák (processzor-, memória-, háttértár és hálózathasználat) megtekintése mellett szükség esetén közvetlenül is be tudunk avatkozni a rendszer működésébe, ugyanis a felület egy terminállal is rendelkezik. Továbbá a futó szolgáltatások állapotát is figyelemmel kísérhetjük, valamint a felhasználói fiókokat is kezelhetjük a Cockpit segítségével.

Az openSUSE-projekt kétféle MicroOS-verziót tart karban: a MicroOS-t, mely egy rolling release modellt követ, azaz a rendszer folyamatosan (akár napi szinten) kapja meg a frissítéseket, így több, kisebb verzióugrással tartható karban, míg az openSUSE Leap Micro a SUSE Linux Enterprise Micro kiadási modelljét követi, és a Leap-hez hasonlóan bináris kompatibilitást garantál a két verzió között. A tesztkörnyezethez a stabilitás és kompatibilitás miatt a Leap Micro változatot választottam.

## 4.4. Hálózati topológia

Az infrastruktúra működésében fontos szerepe van a hálózatnak: a távoli elérésen túl biztosítani kell a szoftvercsomagok elérhetőségét is, valamint a későbbiekben látni fogjuk, hogy a monitoring rendszer is hálózaton keresztül gyűjti az adatokat. Ezek miatt lényeges



**4.4. ábra.** Konténer létrehozása openSUSE Leap Micro-n, a Cockpit webes felületén keresztül.

volt, hogy a gépek tudjanak kommunikálni egymással és a külvilággal. A tesztkörnyezet hálózati felépítését a 4.5. ábra szemlélteti.

#### 4.4.1. Bridge-dzselt hálózati interfész

A virtualizált környezetek sajátossága, hogy a virtuális gépek alapesetben egy – a virtualizációt biztosító szoftver által kezelt – hálózatra tudnak csatlakozni, a fizikai gép hálózatán nincs lehetőségük kommunikálni. Ez a megoldás általában külön konfiguráció nélkül elérhető, viszont hátránya, hogy a külvilág felé gyakorlatilag láthatatlanná válik a virtuális gép. Bár ez porttovábbítással és a tűzfalbeállítások, valamint az érintett szolgáltatások módosításával orvosolható, a hálózati interfészek bridge-dzselése egy szélesebb körben használható megoldást nyújt.

Egy bridge-dzselt interfész lehetővé teszi a virtuális gépek számára, hogy a gazdagéppel azonos hálózaton kommunikáljanak, azaz ugyanúgy működjenek, mintha minden virtuális gép virtualizált hálózati interfészéhez tartozna egy dedikált hálózati csatlakozó a fizikai gépen, mely egyazon hálózathoz csatlakozik. Ehhez a gazdagép hálózati beállításai-  
ban létre kell hozni egy hálózati híd eszközt, és a használni kívánt interfészt be kell állítani bridge masterként. Ezeket a beállításokat egyszerűen elvégezhetjük parancssori felületen, illetve a YaST segítségével is (4.6. ábra). Az így létrejött virtuális eszköz az OSI-modell



**4.5. ábra.** A tesztkörnyezet hálózati felépítése. Az ábrán nem szerepel a szervergép 10.151.7.4-es IP-című menedzsment portja.

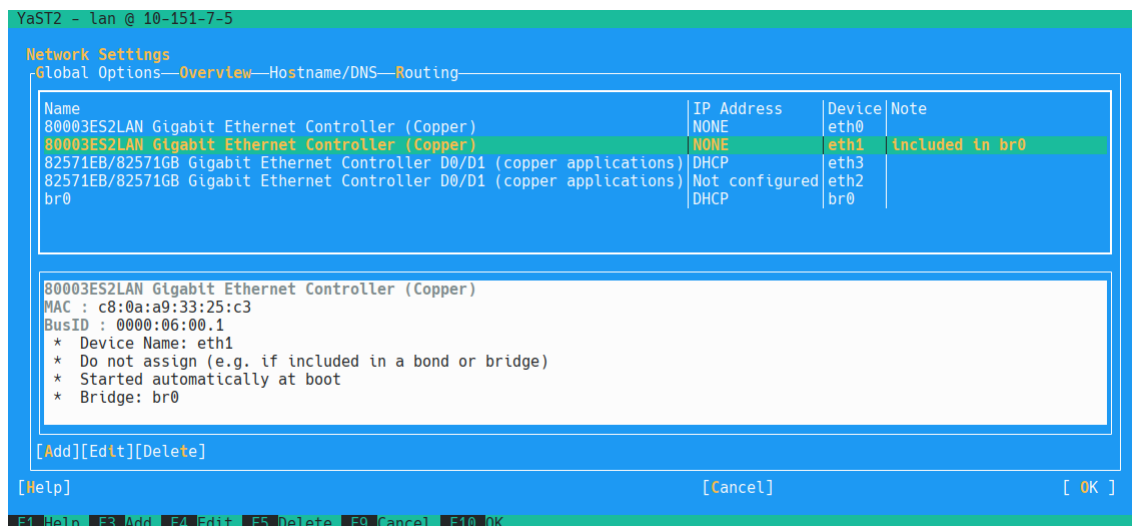
szerinti második szinten, az adatkapcsolati rétegben működik a hardveres switch-ekhez hasonlóan [22].

## 4.5. Virtualizációs komponensek telepítése

A tesztkörnyezet szoftveres alapját a virtualizációs megoldások adják. Ebben az alfejezetben ismertetem a fizikai gép előkészítését és a virtuális gépek telepítésének folyamatát, valamint az ezekhez kapcsolódó beállítási lépéseket, kitérve például a particionálás folyamatára és a virtuális gépek terminálos elérésére.

### 4.5.1. Hosztgép konfigurálása

A virtualizációs környezetet futtató számítógépre az openSUSE Leap 15.5-ös verzióját telepítettem, mely a tesztkörnyezet kialakításakor a disztribúció legfrissebb stabil elérhető változata. A telepítés során a gép szerepének (system role) a szerver opciót választottam. Ez a felhasználási céloknak teljesen megfelelt, hiszen ez a megoldás is egy jól felszerelt



**4.6. ábra.** A tesztkörnyezethez beállított hálózati bridge alapjául szolgáló eth1 fizikai interfész beállításainak részletei a YaST konfigurációs program parancssori változatában. Látható, hogy az eth1 interfész a br0 bridge eszközhöz van társítva, és előbbihez emiatt nincs hozzárendelve IP-cím.

operációs rendszert telepít, csak asztali környezet nélkül. Mivel a szervert elsősorban konzolos felületen, SSH-n keresztül szerettem volna használni, ezért ez nem jelentett gondot. Sőt, a tesztkörnyezet szempontjából előnnyel is járt: a rendszerre nem volt szükséges az asztali környezet működéséhez elengedhetetlen csomagok telepítése, ami nem csak a tárhellyel való takarékoskodásban segített, de a későbbiekben is könnyítette a karbantartási folyamatokat, mert kevesebb csomagot kellett frissíteni és így az esetleges támadási felület (sérülékenységek száma) is kisebb volt. Lényeges azonban megjegyezni, hogy az X11 könyvtár a szerver csomag részeként is telepítésre került, így adott volt a lehetőség X forwarding<sup>2</sup> használatára.

A hálózatkezeléshez – a virtuális gépek hálózati elérését lehetővé teendő – a 4.4.1. alfejezetben bemutatott bridge-dzselt hálózati interfészt állítottam be. Ehhez az alapértelmezett beállításokhoz képest annyit kellett módosítani, hogy a bridge-dzselt eszköz a fizikai interfészen keresztül kommunikáljon, illetve hogy a virtualizációs hosztnak kiosztott 10.151.7.5-ös IP-címet ne az eth1 fizikai interfész kapja meg, hanem az újonnan létrehozott br0 eszköz. Ezt követően a számítógép a korábban megszokott módon tudott a hálózaton kommunikálni, viszont lehetővé vált, hogy a virtuális gépek is hozzáférjenek a fizikai gép hálózatahoz a bridge eszközön keresztül.

<sup>2</sup>Távoli szerveren futtatott, grafikus felülettel rendelkező alkalmazások ablakának a kliensgép képernyőjén való megjelenítését lehetővé tevő technológia.

A hálózaton kívül a másik lényeges tervezői döntés a logikai kötetek (LVM) alkalmazása volt. Ez a gyakorlatban azt jelentette, hogy a boot partíció kivételével minden egyéb kötetet LVM-kötetként hoztam létre. Ennek legfőbb előnye számomra a kötetek méretének rugalmas kezelése volt, melyről a 3.4. alfejezetben írtam bővebben. Emellett lehetőséget biztosít pillanatképek készítésére is, melyek készítése például OS-frissítések előtt lehet releváns, és nagyban megkönnyíti rendszer korábbi állapotának helyreállítást, ha valami hiba jelentkezik a folyamat során. A kötetkiosztás úgy történt, hogy minden virtuális gép kapott egy külön LVM-kötetet, amit egy egyedülálló tárolóeszközként érzékelt, és ezt használhatta az adatok tárolására, akár további particionálás mellett is.

A kezdetleges konfigurációt követően telepítettem a KVM hypervisort és a virtuális gépek kezeléséhez szükséges csomagokat. Ehhez openSUSE-disztribúciókon külön ún. pattern áll a rendelkezésünkre. Ez azt jelenti, hogy nem kell megadnunk minden telepítendő csomagot, hanem elég a `kvm_server` és a `kvm_tools` pattern-ök telepítése, és ezek automatikusan telepítésre jelölik a teljes értékű KVM-szerverhez szükséges csomagokat, emellett néhány hasznos segédprogramot (pl. `virt-manager`, `virsh`) is magukkal hoznak. Ezen csomagok telepítését követően minden előfeltétel adottá vált a virtuális gépek telepítéséhez.

#### 4.5.2. Virtuális gépek telepítése

VM-ek telepítéséhez elsősorban a `virt-install` parancsot használtam. Ez a program lehetővé teszi az összes lényeges paraméter megadását, majd távoliasztal-protokoll használatával (alapértelmezetten SPICE, de választhatjuk például a VNC-t is) megjeleníti a virtuális gép kijelzőjét egy ablakban, melynek segítségével személyre szabhatjuk a telepítést és telepíthetjük az operációs rendszert. A távoli asztalon keresztüli elérés csak a megfelelő környezet, pl. SSH használata esetén, X forwardinggal működik. Amennyiben a `virt-install` nem talál kijelzőt, akkor a telepítés parancssoron keresztül történik. A `virt-install` sikeres futás esetén egy virtuálisgép-leíró XML-fájlt hoz létre, mely a gép összes paraméterét tárolja (4.1. kódrészlet). A későbbiekben a VM konfigurációjának módosítása esetén ezt a fájlt kell módosítanunk (akár szövegszerkesztővel, akár GUI-n, például `virt-manager`-rel). A leírófájl a virtuális gép migrációjához is használható.

```
<memory unit='KiB'>33554432</memory>
<currentMemory unit='KiB'>33554432</currentMemory>
<vcpu placement='static'>12</vcpu>
<resource>
  <partition>/machine</partition>
</resource>
<os>
```

```

    <type arch='x86_64' machine='pc-q35-7.1'>hvm</type>
</os>
...
<devices>
  <emulator>/usr/bin/qemu-system-x86_64</emulator>
  <disk type='block' device='disk'>
    <driver name='qemu' type='raw' cache='none' io='native' discard='unmap' />
    <source dev='/dev/vg1/kvm-uyuni' index='1' />
    <backingStore />
    <target dev='vda' bus='virtio' />
    <boot order='2' />
    <alias name='virtio-disk0' />
    <address type='pci' domain='0x0000' bus='0x04' slot='0x00' function='0x0' />
  </disk>
  ...
</devices>

```

#### 4.1. lista. Virtuális gép leírófájljának részlete.

Azonban még mielőtt a konkrét telepítést elkezdhetnénk, létre kell hoznunk azt a partíciót, melyre a virtuális gép adatai kerülnek. Esetemben ez azt jelentette, hogy az egyes virtuális gépekhez új LVM-köteteket kellett létrehoznom. Mivel a logikai kötetek számára otthont adó partíciót és a kapcsolódó fizikai kötetet és kötetcsoporthoz már a hoszt OS telepítésekor létrehoztam, ezért a VM-ek telepítése során elég volt csak egy-egy logikai kötetet (LV) létrehoznom, melyhez az `lvcreate` parancsot használtam (4.2. kódrészlet). Emellett telepítési forrást is meg kellett adni, melyhez én telepítő lemezképeket használtam. Ilyenkor a VM telepítésekor fel kell venni egy virtuális CD-meghajtót a géphez, és meg kell adni a telepítési forrás elérési útját.

```
lvcreate -L 20G --name kvm-monitoring vg1
```

#### 4.2. lista. Virtuális gépek logikai kötetének létrehozásához használt parancs.

Ezen túl meg kell adni a gép fontosabb paramétereit is (pl. processzorok száma, memória mennyisége), hogy milyen erőforrásokkal szeretnénk telepíteni azt. A tesztkörnyezetben használt gépek telepítése során elsősorban a dokumentációban ismertetett rendszerkövetelményeket vettem figyelembe az erőforrások meghatározásánál, de mivel aránylag sok erőforrás állt rendelkezésemre, így előfordult, hogy a számítási műveletek gyorsítása érdekében a szükségesnél több magot adtam a virtuális gépeknek. Mivel a VM-ek konfigurációja szabadon változtatható a későbbiekben is (esetleg a gép újraindítása szükséges az érvényre jutásukhoz), ezért ez nem jelentett problémát a későbbiekben sem. Emellett a KVM támogatja a memória és processzor erőforrások overcommit-olását is, azaz nem

jelent problémát, ha esetleg a fizikai gépen elérhetőnél több CPU-erőforrást osztottunk ki a virtuális gépek számára, bár ennek használatára a dolgozathoz készített tesztkörnyezetben nem volt szükség [12]. A tesztkörnyezet egyik virtuális gépének telepítéséhez használt parancsot a 4.3. kódrészlet mutatja be.

```
virt-install --name uyuni --memory 32768 --vcpus 12 --cdrom /mnt/openSUSE-Leap-15.5-NET-x86_64-  
Media.iso --os-variant opensuse15.5 --disk /dev/vg1/kvm-uyuni
```

#### 4.3. lista. Virtuális gép telepítése a virt-install segédprogrammal.

A virtuális gép operációs rendszerének telepítése során ki kellett alakítani a kívánt partíciókiosztást, mely a felhasználási körtől függően változott, de alapvetően mindenhol LVM-alapú kötetkiosztást alkalmaztam. Emellett szükséges volt bizonyos hálózati beállítások (pl. DNS-szerver címe, gépnév) módosítása is. Ezeken felül és az alapvető adatok – mint például felhasználói fiókok létrehozása, lokalizációs beállítások konfigurálása – megadásán túl mást nem volt szükséges átállítani a telepítés során. A sikeres installációt követően foghattam hozzá a virtuális gépeken futó szolgáltatások telepítéséhez, melyeket a következő fejezetekben fogok részletesen ismertetni.

## 5. fejezet

# Infrastruktúra-menedzsment: Uyuni

Nagyméretű informatikai infrastruktúra kezelése esetén elengedhetetlen valamiféle infrastruktúramenedzsment-eszköz használata. Ez nem csak könnyebbé teszi az üzemeltetést, de számos kiegészítő funkcióval is rendelkezik, lényegében egy helyen láthatunk minden releváns adatot, és egyazon felületről van lehetőségünk frissítések telepítésére és biztonsági sérülékenységek leírásainak böngészésre, mint ahol azt is tároljuk, hogy egy adott virtuális gép melyik gazdagépen fut, és az hol található.

Mivel ez a megoldás hasznosnak és érdekesnek tűnt számomra, és a dolgozat profiljába is jól illeszkedik, úgy döntöttem, hogy a tesztkörnyezet kezelésére is fogok ilyen megoldást alkalmazni. A választásom az Uyuni-ra esett, mely gyakorlatilag mindent tud, amire szükségem volt, és ingyenesen elérhető. A döntésben az is segített, hogy a projektet a SUSE támogatja, a fejlesztésében is részt vesz (az Uyuni szolgál a kereskedelmi forgalomban lévő SUSE Manager megoldás alapjául), és az openSUSE-alapú disztribúciókra is kiemelt figyelmet fordítanak, így nem kellett kompatibilitási problémákkal foglalkoznom. Az Uyuni a Salt konfigurációmenedzsment és automatizációs keretrendszerre épül, mely szintén széleskörűen használt és támogatott.

### 5.1. Telepítés

Az Uyuni meglehetősen erőforrás-igényes, a telepítéséhez minimum négy CPU-mag, 16 GB memória és több száz gigabyte tárhely lehet szükséges a használni kívánt telepítőforrásoktól függően [23]. Éles környezetben még ennél is több memóriát javasolnak, így az Uyuni számára létrehozott virtuális gép lett a környezet leginkább erőforrás-igényes rendszere,



melynek telepítéshez használt parancs és a VM leírója látható a 4.3. és a 4.1. kódrészleteken. Ezeken megfigyelhetjük, hogy a virtuális gépet 12 CPU-maggal és 32 GB memóriával hoztam létre, a particionálásra a későbbiek folyamán fogok részletesen kitérni.

Ahogy a korábbiakban írtam, az Uyuni jó támogatottságnak örvend az openSUSE platformon. A telepítési kézikönyvben egy dedikált openSUSE Leap 15.5-ös OS-verziót futtató számítógépet javasolnak az infrastruktúramenedzsment-program telepítéséhez [23]. Ennek megfelelően én is ezt a verziót telepítettem, és bár a Leap alapértelmezett telepítési forrásaiban nem szerepel az Uyuni, de az ezt tartalmazó telepítőforrást egyetlen paranccsal felvehetjük, és ezt követően a segítségével telepíthetjük is az Uyunit.

### 5.1.1. Kötetkiosztás

A particionálás kérdése azért is kiemelt fontosságú az Uyuni telepítése során, mert a telepítés folyamán ezen feltételek meglétét ellenőrzi is a telepítő, és nem teljesülésük esetén a telepítés hibára futhat. A kötetek kiosztása során a hivatalos útmutatóból indultam ki, viszont a felhasználási célok figyelembevételével néhány partíciót a javasoltnál nagyobbra állítottam be, ennek megfelelően jött létre az 5.1. táblázatban ismertetett felépítés. Elsősorban a spacewalk kötet növelése volt indokolt, hiszen az Uyuni egyik fontos része a szoftvercsomagok kezelése. Ennek gördülékeny és hatékony használatához a rendszer le-tükrözi a szükséges telepítőforrásokat, melyek jelentős helyigénnyel rendelkeznek, melyre az 5.2 alfejezetben térek ki részletesen.

Kötet	Csatolási pont	Méret	Típus	Leírás
boot	/boot	1 GB	fizikai, ext4	Az OS betöltéséhez szükséges fájlok tárhelye.
root	/	40 GB	LVM, XFS	A könyvtár-hierarchia legfelső eleme, a rendszerfájlok helye.
pgsql	/var/lib/pgsql	52 GB	LVM, XFS	A PostgreSQL adatbázismotor által tárolt adatok könyvtára.
spacewalk	/var/spacewalk	100 GB	LVM, XFS	Az infrastruktúramenedzsment-szoftver által használt fájlok elsődleges helye.
cache	/var/cache	12 GB	LVM, XFS	Gyorsítótárazott adatok könyvtára.
swap	swap	4 GB	LVM, swap	Virtuális memóriaként használt lemezterület.

**5.1. táblázat.** Az infrastruktúramenedzsment-szerver telepítéskori kötetkiosztása.

## 5.2. Telepítőforrások tükrözése

Ahogy az előző alfejezetben már említettem, az Uyuni egyik központi feladata a telepítőforrások és szoftvercsomagok kezelése a bevont rendszereken. Ehhez a használni kívánt telepítőforrásokat le kell tükröznünk az Uyunit futtató szerverre. Ez a megoldás elsőre túlzásnak tűnhet, hiszen egy-egy ilyen forrás több tíz, egyes esetekben több száz gigabyte lehet, azonban egy nagyméretű infrastruktúránál megvannak az előnyei. Gondoljunk csak bele, hogy mekkora hálózati forgalmat takaríthatunk meg azzal, hogy a telepítendő szoftverek és operációsrendszer-frissítések letöltésekor nem kell minden esetben az internetről letölteni a szükséges csomagokat, hanem ezt elegendő mindössze egyszer, az Uyuni szerveren megtenni, és onnantól kezdve az összes kliens már a belső hálózatról érheti el a szükséges programokat. Emellett hasznos lehet egy offline példány megtartása is a telepített csomagokból, hiszen előfordulhat, hogy egy harmadik fél által üzemeltetett telepítési forrás elérhetetlenné válik, és így nem férünk hozzá a kívánt csomagokhoz.


Ahhoz, hogy egy-egy ilyen csomagtárhelyet letükrözhessünk, létre kell hozni úgynevezett *Software Channel*-eket, melyek lényegében a tárolót (repository) csomagolják be egy magasabb szintű egységbe. Ez tárolja, hogy milyen csomagok érhetőek el, ezek milyen architektúrán használhatóak, lehetővé teszik a csomagforrások kezelését, és hozzárendelhetjük őket a kliensekhez. A csatornák által tartalmazott repository-k tárolják, hogy milyen címen érhető el a tükrözendő telepítőforrás, és hogy az adott repository melyik csatornához van hozzárendelve.

A tesztkörnyezethez is tükröztem telepítési forrásokat. Mivel a gépek többsége openSUSE Leap 15.5-ös verziót használt, ezért az ehhez tartozó csomagok szinkronizálásával kezdtem. Minden alapértelmezett forrást letükröztem, illetve az Uyuni általi menedzselés céljából az Uyuni-csomagokat tartalmazó tárolót is felvettem a szinkronizálandók közé. Az egyszerűbb kezelés miatt a csatornák hierarchiába szervezhetőek, így csak a kiválasztott szülőcsatornával (base channel) kompatibilis gyermekcsatornákat vehetünk fel egy adott klienshez. A tesztkörnyezetben létrehozott telepítőforrás-hierarchiát az 5.1. ábra szemlélteti.










Az első szoftvercsomag-tároló tükrözése a dokumentációban javasolt 50 GB-nál közel 20%-kal több tárhelyet igényelt a spacewalk köteten (5.2. ábra), így szükségessé vált a partíció megnövelése [23]. Ennek megvalósításához az LVM által biztosított `lvextend` parancsot használtam, mely a logikai kötet megnövelésén túl a fájlrendszer (itt XFS) kiterjesztését is támogatja. Ez a parancs viszont csak akkor használható, ha a kötetcsoport-

**All**   SUSE   Popular   My Channels   Shared   Retired

The software channels listed below are **all of the channels** that your organization has access to.



[Show All Child Channels](#) | [Hide All Child Channels](#)

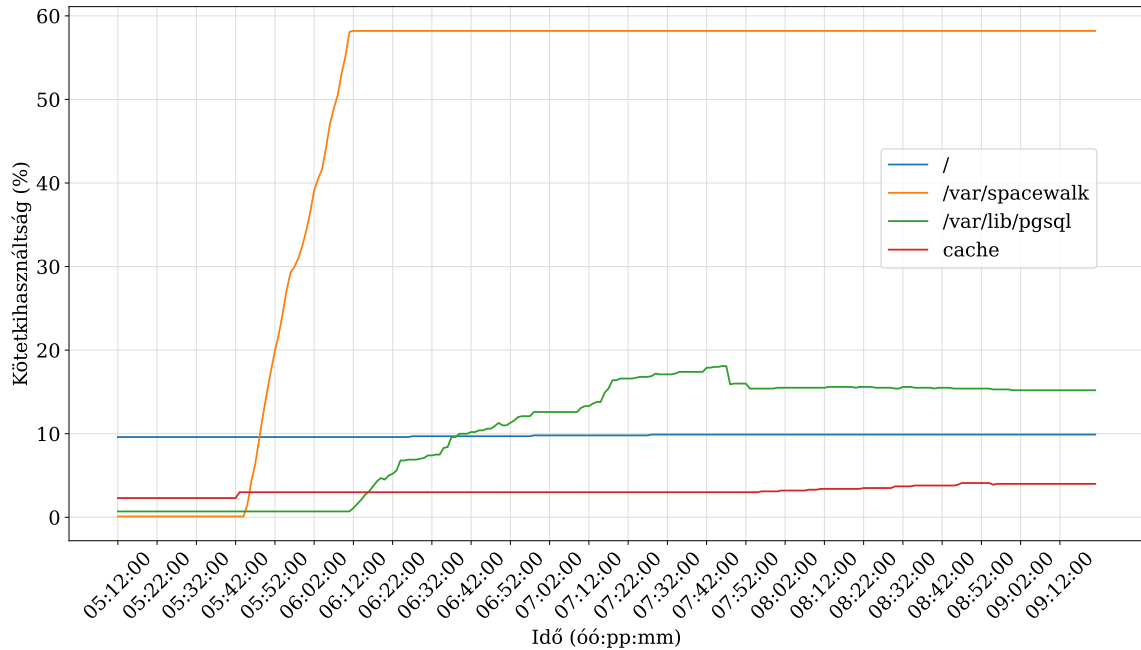
Channel Name	Provider	Packages	Patches	Systems
 openSUSE Leap 15.5	SPOT Fotókör	42070	0	2
 openSUSE Leap 15.5 Backports	SPOT Fotókör	1708	285	2
 openSUSE Leap 15.5 OSS	SPOT Fotókör	42070	0	2
 openSUSE Leap 15.5 OSS Updates	SPOT Fotókör	255	21	2
 openSUSE Leap 15.5 SLE	SPOT Fotókör	16309	1463	2
 openSUSE Leap 15.5 Uyuni	SPOT Fotókör	192	0	2
 openSUSE Leap Micro 5.5	SPOT Fotókör	0	0	1
 openSUSE Leap Micro 5.5 Main	SPOT Fotókör	1007	0	1
 openSUSE Leap Micro 5.5 Updates	SPOT Fotókör	851	339	1

**5.1. ábra.** A kialakított csatornahierarchia a szülőcsatornákkal és a belőlük leszármazó gyermekcsatornákkal.

ban (VG) rendelkezésre áll annyi szabad hely, amennyivel bővíteni szeretnénk a tárhelyet. Mivel azonban itt ez a feltétel nem teljesült, több szinten kellett elvégezni a kötet megnövelését, először a KVM-hosztzon kellett megnövelni a virtuális géphez tartozó tárterületet. További kihívást jelentett, hogy a kiterjesztést a VM leállítása nélkül szerettem volna elvégezni. Ebben az esetben nem jelentett volna nagy gondot a gép újraindítása, viszont szerettem volna felkészülni olyan helyzetekre is, amikor a gép leállítása nem megengedhető. Ilyen lehet például, ha a szinkronizálás alatt álló telepítőforrás a vártnál jelentősen nagyobb méretűnek bizonyul, vagy éppen a PostgreSQL által használt tárhely van fogytán, de még nem fejeződött be a korábban indított indexelés. Mindezek mellett általában éles környezetben is korlátozottak a lehetőségeink egy-egy kiszolgáló újraindítására.

### 5.2.1. Online kötetnövelés

Ezen okokból a spacewalk kötet online megnövelése mellett döntöttem. Első lépésként a hosztgépen terjesztettem ki a virtuális gép számára fenntartott partíciót (a `kvm-uyuni` logikai kötetet). Az 5.1. kódrészleten látható, hogy ezt követően a virtuális gép még nem tudja használni a nagyobb méretű kötetet, szükséges még a `virsh blockresize` parancs futtatása, melynek hatására a virtuális gép számára is láthatóvá válik a nagyobb kötetméret. Ezt követően a VM-en növeltem meg a logikai kötetek alapjául szolgáló fizikai



**5.2. ábra.** Kötetkihasználtság változása az egyik telepítőforrás törlése során.

kötet (PV) méretét, melyet követően már a megszokott módon volt lehetőség a logikai kötet bővítésére.

```
10-151-7-5:~ # virsh domblkinfo uyuni /dev/vg1/kvm-uyuni
Capacity:      268435456000
Allocation:    225145827328
Physical:      375809638400

10-151-7-5:~ # virsh blockresize uyuni --path /dev/vg1/kvm-uyuni --size 350G
Block device '/dev/vg1/kvm-uyuni' is resized

10-151-7-5:~ # virsh domblkinfo uyuni /dev/vg1/kvm-uyuni
Capacity:      375809638400
Allocation:    225145827328
Physical:      375809638400
```

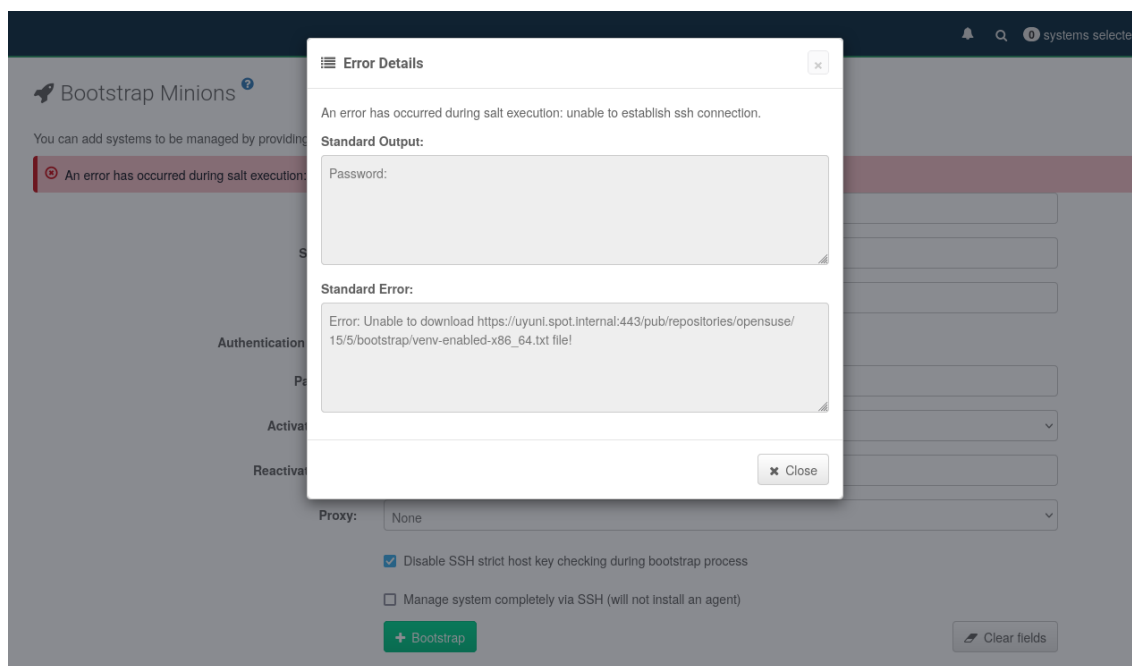
**5.1. lista.** Az infrastruktúramenedzsment-programokat futtató virtuális gép kötetének online megnövelése a gazdagépen.

### 5.3. Kliensek felvétele

A kliensek felvétele során mutatkozott meg az Uyuni egy kisebb hátránya az előfizetés-alapú SUSE Manager-rel szemben. Míg az előfizetésért cserébe előre konfigurált csatornákat (ún. Product-okat) kapunk, addig az Uyunihoz ez nem áll rendelkezésünkre, a kliensek felvételéhez használt bootstrap<sup>1</sup> megoldás pedig csak ezekkel működik együtt. A boot-

<sup>1</sup>Az a folyamat, amely során a Salt master inicializálja a Salt minionokat.

strap folyamat sikeres futtatása többszörös próbálkozást követően sem sikerült (5.3. ábra). Emiatt az elsődleges konfiguráció során manuálisan kellett gondoskodnom a szükséges csomagok telepítéséről és az ezekhez kapcsolódó beállítások elvégzéséről. A gyakorlatban ez azt jelentette, hogy a kezelni kívánt rendszerekre telepíteni kellett a `venv-salt-minion` csomagot, emellett szükséges volt két konfigurációs fájl létrehozása is, melyek a Salt minion azonosítóját (amilyen néven a gép megjelenik az Uyuni felületén) és a Salt master (az Uyuni szerver) IP-címét tartalmazzák (5.2. kódrészlet). Ezt követően a kapcsolódó Salt service újraindításával a kliens kapcsolatot kezdeményezett a Salt masterrel, és az Uyuni felületén lehetővé vált a kliens csatlakozási kérelmének elfogadása. A kérelem elfogadását követően az Uyuni lekérdezett néhány adatot a kliens állapotáról (pl. telepített csomagok, hardveres erőforrások), ez után az újonnan bevont rendszer már teljes körűen kezelhetővé vált az Uyuni felületén keresztül.



**5.3. ábra.** Előfizetés nélkül nem érhetőek el a product repository-k.

```
10-151-7-5:/etc/venv-salt-minion/minion.d # cat id.conf
id: kvm.spot.internal
10-151-7-5:/etc/venv-salt-minion/minion.d # cat master.conf
master: 10.151.7.1
```

**5.2. lista.** A manuális konfiguráció során létrehozott állományok tartalma.

kvm.spot.internal

Delete System
Remove from SSM

Details
Software
Configuration
Provisioning
Groups
Virtualization
Audit
States
Formulas
Recurring Actions
Events

Overview
Properties
Remote Command
Reactivation
Hardware
Transfer
Notes
Custom Info

System Status

Software Updates Available
Critical: 24
Non-Critical: 30
Packages: 152

System Info

Hostname:

kvm.spot.internal

IP Address:

10.151.7.5

IPv6 Address:

fd17:5a8d:ca33:0:21a2:a0fb:cad9:e5e0

Minion Id:

kvm.spot.internal

Kernel:

5.14.21-150500.55.52-default

Uyuni System ID:

1000010003

Installed Products:

System Events

Checked In:

Yesterday at 09:00

Registered:

2024-04-20

Last Booted:

a month ago

(Schedule System Reboot)

System Properties (Edit These Properties)

System Types:

[Salt] [Monitored Host] [Virtualization Host]

Contact Method:

Default

Auto Patch Update:

No

Maintenance Schedule:

(none)

System Name:

kvm.spot.internal

Description:

Location:

Room: 105

Rack: 3

Building:

Irinyi József utca 42

1117 Budapest Budapest HU

Subscribed Channels (Alter Channel Subscriptions)

Base Channel

openSUSE Leap 15.5

Child Channels

openSUSE Leap 15.5 Backports
openSUSE Leap 15.5 OSS
openSUSE Leap 15.5 OSS Updates
openSUSE Leap 15.5 SLE
openSUSE Leap 15.5 Uyuni

**5.4. ábra.** A bevont rendszer adatainak frissítését követően számos részletet láthatunk a gépről, és mi is adhatunk meg bizonyos adatokat.

### 5.3.1. Rendszertípusok és Salt Formulák

A kezdeti információgyűjtés során az infrastruktúramenedzsment-rendszer megpróbálja meghatározni a rendszer típusát (fizikai gép, virtualizációs gazdagép, virtuális gép). Ezeket később mi is megadhatjuk, illetve a betöltött szereptől függően úgynevezett Salt Formulákat rendelhetünk hozzájuk. Ezek olyan előre elkészített Salt state fájlok, melyek a gyakran használt beállítások megadását igyekeznek egyszerűbbé tenni.

Munkám során én két rendszertípust használtam: a virtualizációs hoszthoz és a monitorozott rendszerhez kapcsolódót. A *Virtualization Host* Formula néhány alapvető beállítást (pl. hálózati konfiguráció) ad meg a rendszeren, azonban ez a felhasználási céljaimnak nem felelt meg teljes mértékben, így a beállításokat egy saját Salt state-tel módosítottam, mely az 5.3. kódrészleten látható. A példában a könnyebb átláthatóság érdekében a módosítandó fájl tartalmát a state fájlban definiáltam, de lehetőség van ennek egy köz-

pontilag, az Uyuni keresztül kezelt konfigurációs fájl hivatkozásával való megadására is. A *Monitored Host* beállításait a következő fejezetben tárgyalom részletesen.

```
update_/etc/sysconfig/network/ifcfg-br0:
  file.managed:
    - name: /etc/sysconfig/network/ifcfg-br0
    - user: root
    - group: root
    - mode: 644
    - content: |
      BOOTPROTO='dhcp'
      STARTMODE='onboot'
      ZONE='public'
      BRIDGE='yes'
      BRIDGE_PORTS='eth1'
      BRIDGE_STP='off'
```

**5.3. lista.** A hálózati konfiguráció frissítését végző Salt state.

## 5.4. Szolgáltatások telepítése, kezelése

Az Uyuni egy fontos része a konfiguráció-menedzsment és a szoftvercsomagok telepítése, frissítése. Ezekre elsősorban Salt state-eken keresztül nyújt támogatást, a tesztkörnyezetben én is a Salt-alapú megoldást választottam a csomagok telepítésére.

Csomagok és azok konfigurációjának kezeléséhez létre kell hoznunk egy konfigurációs csatornát, egy úgynevezett state channelt, mely tartalmazni fogja az elkészített state fájlt és az esetlegesen használt konfigurációs állományokat. A state fájl megírását követően a csatorna hozzárendelhető a kezelt hosztokhoz.

Miután ez megtörtént, a gépekre, melyekhez hozzárendeltük az adott state-et, ki kell küldeni egy úgynevezett highstate-et. Ez egy különleges state, melyet a rendszer állít elő a feliratkozott state-ek tartalmából, a feliratkozási hierarchiát figyelembe véve. A highstate tehát minden hozzárendelt state channel adatait tartalmazza, és a lefuttatásával jutnak érvényre a frissen felvett módosítások is. Az 5.5. ábrán látható, hogy az Uyuni lehetőséget biztosít több kliens kiválasztására is, ezt kihasználva két hosztra egyszerre alkalmaztam a módosításokat.

A fenti példában használt channel state fájlja az 5.4. kódrészleten látható. Ezzel néhány, a hálózati forgalom monitorozását lehetővé tevő csomagot telepítettem a feliratkozott gépekre, illetve elindítottam a kapcsolódó service-t és felülírtam az alapértelmezett konfigurációs állományt egy némileg különbözővel, ami alapértelmezetten más mértékegységeket használ a forgalom monitorozása során. Az így módosított konfiguráció 2-es alapú hatványok (kibi- és mebibyte) helyett 10-es alapú hatványokat használó mértékegységekkel (kilo- és megabyte) teszi lehetővé az átvitt adat mennyiségének követését.

## System Set Manager Overview

Overview Systems Patches Packages Groups Channels Configuration Provisioning Audit States

Misc



Deploy Files Compare Files Subscribe to Channels Unsubscribe from Channels Enable Configuration

### Confirm Subscription to Configuration Channels


#### Step 3: Confirm Channels for Subscription.

The systems listed below will be subscribed to the configuration channels listed below.

All of the configuration channels will be subscribed to in the **lowest** position in the order they appear below. Any currently existing subscription between one of the systems listed below and one of the channels listed below will be replaced so that it appears in a new position. All other currently existing subscriptions will remain in their current position.

System Name	Current Subscriptions to Chosen Channels
 kvm.spot.internal	1 subscription
 monitoring.spot.internal	1 subscription

1 - 2 of 2 1 - 1 of 1

Configuration Channel	Files	Current Subscriptions by Chosen Systems
 Network monitoring	1 state and 1 file	0 subscriptions

Confirm

#### 5.5. ábra. Két kiszolgáló feliratkoztatása a *Network monitoring* state channelre.

A state konfigurálásához éltem a Salt által nyújtott sablonozási lehetőséggel, mely lehetőséget biztosít például változók létrehozására, **if-else** ágak meghatározására, és **for** ciklus használatára is. Bár ez elsőre némi többletmunkát jelent a state létrehozása során, egy sokkal karbantarthatóbb megoldást eredményez, hiszen elegendő például a telepítendő csomagokat egy listában felsorolni, melyet bármikor bővíthetünk, és nem kell egy-egy csomópontot lemásolni újabb hibalehetőségek esetleges bevezetésével. A sablonozás további előnye, hogy futásidőben is lekérdezhethetjük például egy fájl létezését, és ez alapján folytathatjuk a state végrehajtását. Ilyen megoldást szemléltet a state forráskódjának 10. sora, melyben az elágazás megvizsgálja, hogy éppen telepítésre kerül-e a **vnstat** csomag, vagy ha nem, akkor egy létező konfigurációs állományt szeretnénk-e frissíteni. Az elágazás eredménye függvényében frissíti a fájlt, vagy változatlanul hagyja. Egy másik hatékony megoldás az úgynevezett *grain*-ek használata, melyek lehetővé teszik részletes információk lekérdezését egy Salt kliensről. Ilyet használtam a forráskód 21. sorában, ahol az OS-család függvényében más-más néven tudtam elérni a **vnstat** háttérfolyamatát.



```

1 {% set vnstat_config_path = '/etc/vnstat.conf' %}
2 {% set packages = ['vnstat', 'nload', 'nethogs'] %}
3
4 {% for package in packages %}
5 install_package_{{ package }}:
6     pkg.installed:
7         - name: {{ package }}
8 {% endfor %}
9
10 {% if 'vnstat' in packages or salt['file.file_exists'](vnstat_config_path) %}
11 update_vnstat_config:
12     file.managed:
13         - source: salt://manager_org_1/alertmanager_email{{ vnstat_config_path }}
14         - name: {{ vnstat_config_path }}
15         - user: root
16         - group: root
17         - mode: 644
18
19 enable_vnstatd:
20     service.running:
21         {% if grains['os_family'] == 'Suse' or grains['os_family'] == 'RedHat' %}
22             - name: vnstatd
23         {% else %}
24             - name: vnstat
25         {% endif %}
26         - enable: True
27         - watch:
28             - file: {{ vnstat_config_path }}
29 {% endif %}

```

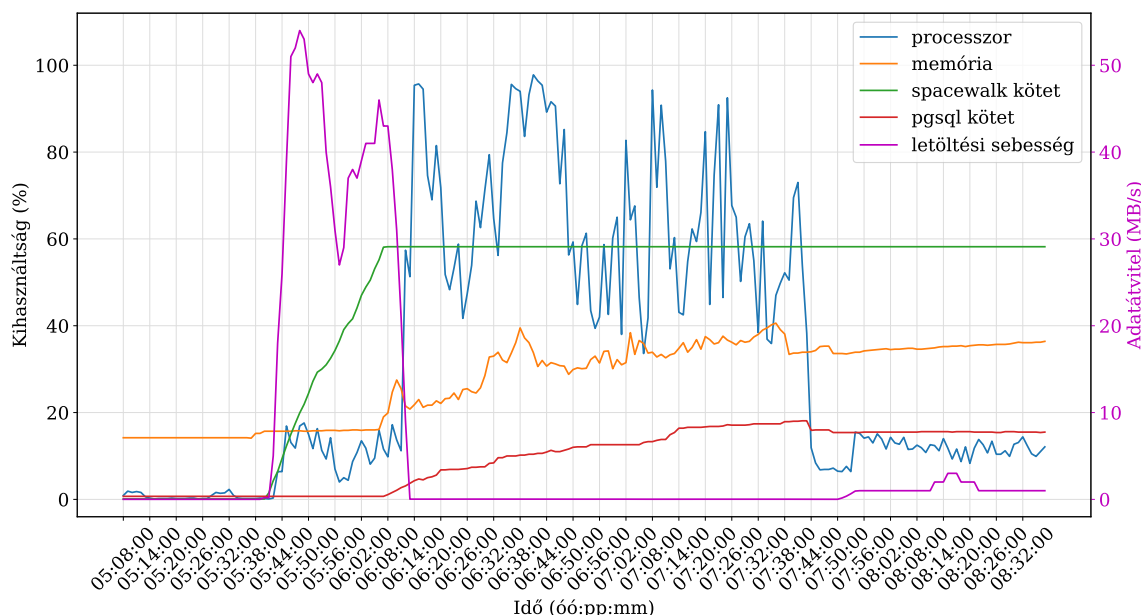
**5.4. lista.** A Network monitoring konfigurációs csatornához tartozó Salt state.

## 5.5. Erőforrásigény mérése

A fejezet elején ismertetett komoly erőforrásigény miatt kíváncsi voltam a hardver adta lehetőségek tényleges kihasználtságára. Mivel ekkor még nem állítottam be a monitoring rendszert (ehhez előbb be kellett fejeznom az Uyuni konfigurációját), ezért egy saját Python scriptet készítettem az erőforrások állapotának figyelésére. A script teljes forráskódját a Függelék F.1.1. kódrészlete mutatja be. Ennek érdekessége, hogy eleinte szerettem volna részletesen, külön-külön nyomon követni az Uyuni által használt alrendszerek erőforrásigényét is. Az ehhez tartozó kód a kódrészlet 88-108. sorában látható. Ezek a lekérdezések azonban nagyon erőforrás-igényesnek bizonyultak, hiszen végig kellett ellenőrizni az összes futó folyamatot. Emiatt nem volt tartható az egységnyi időnként történő lekérdezés, így a későbbiekben ez a rész kikerült a mérési scriptből.

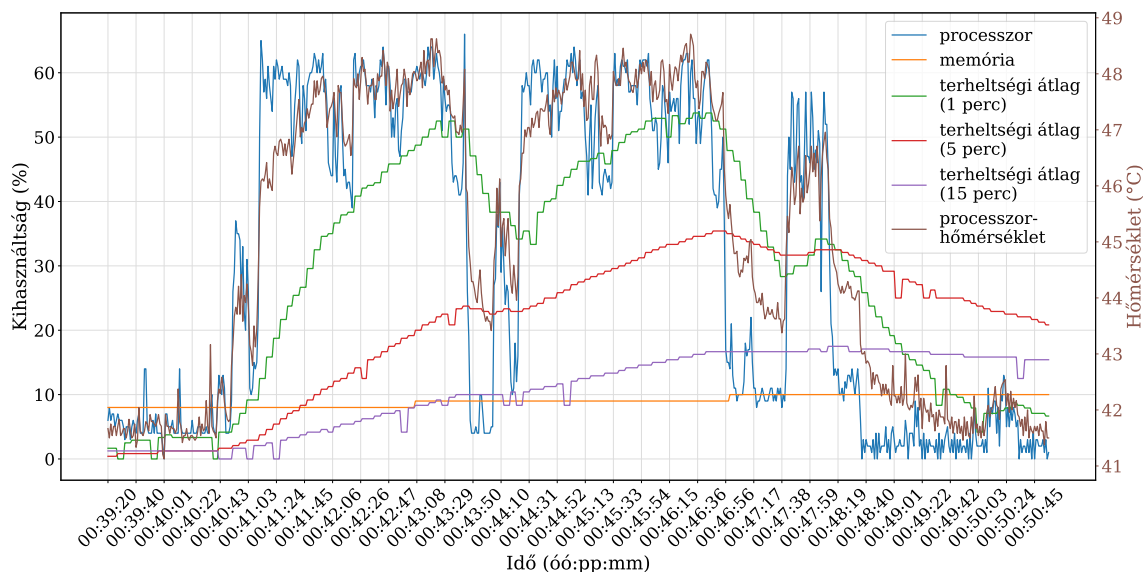
A mérést egy telepítőforrás-tükrözés során végeztem, hiszen ekkor volt a legnagyobb a rendszer erőforrásigénye. A jelentős mértékű hálózati adatforgalmat követően – melyet a szoftvercsomagok letöltése idéz elő – a rendszer hosszas adatbázis-indexelést hajtott végre, mely során a processzormagok kihasználtsága közel 100%-os. A mérési adatokból készített grafikon az 5.6. ábrán látható. Jól megfigyelhető, hogy a kezdeti nagy hálózati forgalommal párhuzamosan a spacewalk kötet kihasználtsága is gyorsan nő. A letöltés befejeztével

megkezdődik az adatbázis-indexelés, mely során a számítási kapacitás kihasználtságának növekedése tapasztalható, emellett pedig a várakozásoknak megfelelően az adatbázisköte-  
ten nő a foglalt terület.



**5.6. ábra.** Az Uyunit futtató virtuális gép terheltsége az egyik telepítőforrás tükrözése során.

Egy másik, rövidebb szinkronizálás során a virtuális gépeket futtató gazdagépen figyeltem meg a mérőszámok változását. Ennek érdekessége, hogy így lehetőségem nyílt a processzormagok hőmérsékletének nyomon követésére is. Ezáltal megfigyelhetjük a szervergép hűtésének hatékonyságát: míg a CPU kihasználtsága körülbelül 50%-kal nőtt, a processzor hőmérséklete csupán 8°C-kal változott, ami körülbelül 20%-os növekedést jelent. Az 5.7. ábráról leolvasható, hogy a processzor-hőmérséklet a CPU kihasználtságával összhangban változik, azonban érdemes megjegyezni, hogy a hőmérsékletet megjelenítéséhez egy másik skálát alkalmaztam, így a hőmérséklet-növekedés mértéke arányaiban valóban elmarad a processzorkihasználtság változásától. Az ábrán látható 60% körüli CPU-kihasználtság egyébként közel teljes processzorkihasználtságot jelent a virtuális gépen, ugyanis ez körülbelül 14 mag teljes kihasználtságát jelenti (mivel  $24 \cdot 0.6 = 14,4$ ), azaz a két tétlen VM mellett feltehetően az adatbázis-indexelést végző Uyunit futtató virtuális gép kihasználja a rendelkezésére álló 12 magot.



**5.7. ábra.** Processzormagok átlagos hőmérséklete a rendszer terheltsége mellett.

## 5.6. Frissítési értesítések

Teljes körű infrastruktúramenedzsment-megoldásként az Uyuni lehetőséget biztosít a kezelt gépekre telepített csomagok állapotának folyamatos követésére: minden nap, egy előre meghatározott időpontban frissíti a telepítési források adatait, és összeveti az ezekben elérhető szoftververziókat a kezelt gépekre telepítettekével. Ennek köszönhetően a teszt-környezet gépeihez frissítésekről folyamatosan értesültem e-mailben, így mindig naprakész információk álltak a rendelkezésemre a frissítéseket illetően.

Ezek közül kiemelt fontosságúak lehetnek a biztonsági frissítések, hiszen ezek között olyan sérülékenységek javításai is szerepelhetnek, mely kockázatot jelent a rendszer helyes működésére nézve. Ilyen értesítésből körülbelül két és fél hét alatt 9 érkezett, ezeken felül pedig nagyságrendileg 40 további, alacsonyabb prioritású frissítésről kaptam értesítést. Ezen felül az Uyuni felületére belépve az áttekintés oldalon és a rendszerek listájában is látható a kezelt rendszereink frissítéseinek állapota (5.8. ábra).

Most Critical Systems				
System	All Updates	Security Patches	Bugfix Patches	Enhancement Patches
<a href="#">kvm.spot.internal</a>	46	🛡️ 20	🐛 26	🔧 0
<a href="#">monitoring.spot.internal</a>	21	🛡️ 9	🐛 12	🔧 0
<a href="#">containers.spot.internal</a>	7	🛡️ 3	🐛 4	🔧 0

**5.8. ábra.** A kezdőlapon tájékozódhatunk a kliensek biztonsági állapotáról.

## 6. fejezet

# Monitoring

A 3.7. alfejezetben ismertetett előnyei miatt a tesztkörnyezetben is egy monitoring rendszer kialakítása mellett döntöttem. Választásom a Prometheus monitorozó rendszerre esett, mely a Grafana vizualizációs eszközzel társítva jó betekintést enged az üzemeltetett rendszerek állapotába. A választásomat az indokolta, hogy ez a párosítás manapság nagyon elterjedt, széles körben használt és jól kibővíthető további adatok monitorozásával, továbbá számos segédanyag áll hozzá rendelkezésre mind a hivatalos oldalon, mind pedig a közösség által karbantartott ismertetők formájában.

### 6.1. Áttekintés

A továbbiakban tárgyaltak könnyebb megértése végett hasznos megismerni a Prometheus és a kapcsolódó komponensek működését. A Prometheus működési elve egyszerű és hatékony: a figyelt kliensekre úgynevezett *exporterek* telepítünk, melyeket a Prometheus szerver pull modellt alkalmazva adott időközönként lekérdez. Az exporterek olyan programok, melyek alkalmasak bizonyos statisztikák (pl. erőforráskihasználtság mértéke, adatbázis-lekérdezések gyakorisága) kiolvasására, és ezeket a monitorozott rendszer egy adott portján HTTP-protokollon keresztül közzétenni. Az exporterek önmagukban nem gyűjtenek adatokat, csak adott időközönként frissítik őket, az adatgyűjtést a Prometheus szerver végzi. Figyelt végpontokból rendszerenként több is lehet, mindegyikhez külön-külön port kapcsolódik.

A monitorozás szerves része a riasztások küldése, hiszen a monitoring rendszer folyamatos figyelése nélkül is jó, ha értesülünk a fennálló problémákról, hogy még időben be tudjunk avatkozni. A Prometheus a riasztások kezeléséhez az Alertmanager komponentst kínálja. Ezt részletesen konfigurálhatjuk az igényeinknek megfelelően, hogy milyen típusú

problémáról, milyen platformon keresztül és kik kapjanak értesítést. Ez azért is kedvező, mert például egy éles környezet esetében egy alkalmazás összeomlásakor feltehetőleg másnak kell eljárnia, mint amikor hálózati fennakadást (pl. DNS-probléma<sup>1</sup>) tapasztalunk.

Az összegyűjtött adatokat a Grafana segítségével tehetjük átláthatóbbá, valamint ez teszi könnyebbé a metrikák elemzését is. Adatvizualizációra specializálódott szoftver lévén a Grafana rengeteg opcióval rendelkezik az adatok megjelenítését illetően, így a CPU-adatok vonalgrafikonon való megjelenítésétől kezdve a szerverünk elérhetőségét jelző igen-nem panelt is választhatunk. Grafanában a megjelenített adatokat – melyekhez egy-egy panel tartozik – úgynevezett *dashboard*okba szervezhetjük. A dashboardok általában összetartozó adatokat tartalmaznak, azaz jó gyakorlat a hardveres erőforrások metrikáit külön dashboardba szervezni például a webservertől tartozó mérőszámoktól. A Grafana támogatja a Prometheus saját lekérdezőnyelvével, a PromQL-lel való adatszűrést, mely lehetővé teszi a Prometheus szerverrel való egyszerű interakciót. A 6.1. ábra egy általános Prometheus-alapú, Grafana adatvizualizációt használó monitoring-architektúrát mutat be.



**6.1. ábra.** A Prometheus monitoring rendszer felépítése [9].

A Prometheus exporterek egy speciális formátumban teszik közzé az általuk kinyert adatokat. A legegyszerűbb esetben ezek szóközzel elválasztott kulcs-érték párok. Ilyen

<sup>1</sup>A DNS az emberek által könnyen megjegyezhető domain neveket fordítja le az informatikai eszközök által értelmezhető IP-címekre.

adatpárra mutat példát a 6.1. kódrészlet első sora. A mérőszám elnevezése után kapcsos zárójelekbe szűrőket (filter) helyezhetünk, ahogy az a 6.1. kódrészlet második sorában is látható. Lekérdezés esetén az adatokat ugyanezzel a szintaktikával szűrhetjük.

```
1 node_load1 0.35
2 node_hwmon_temp_celsius{chip="platform_coretemp_1",sensor="temp2"} 28
```

**6.1. lista.** Prometheus exporterek által közzétett adatok.

## 6.2. Telepítés, konfiguráció

A környezet beüzemelését nagyban segítette, hogy a Prometheus monitoring rendszert az Uyuni külön Salt Formulákon keresztül támogatja. Így lehetőség van az infrastruktúra-menedzsment rendszerbe bevont klienseken manuális csomagtelepítés nélkül egy kezdetleges monitoring környezetet kialakítani. Ez a gyakorlatban azt jelentette, hogy az Uyuni a beállított Formulák alapján a kliensekre telepítette az alapvető rendszerinformációkat lekérdező *Node exporter*t, míg a monitoring szerveren egy alapbeállításokkal rendelkező Prometheus szervert és a hozzá kapcsolódó Grafana szervert telepítette.

### 6.2.1. Konténerhoszt monitorozása

A monitoring rendszer kialakítása során nehézséget jelentett, hogy openSUSE Leap Micro-ra nem volt hivatalosan elérhető a felügyelt gépekről hardver- és kernelstatisztikákat szolgáltató Node exporter szoftvercsomag formájában. Emiatt eleinte a konténerhoszt monitorozására nem volt lehetőségem. A megoldást az jelentette, hogy az exporter-t egy konténerben telepítettem a virtuális gépre. Mivel a konténer azonban a gazdagéptől elkülönülten fut, ezért módosítani kellett a konténer futtatásához használt parancsot. A `--path.rootfs=/host` kapcsoló beállításával az exporter már a konténerből is képes volt elérni a gazdagép metrikáit, így ezt a hosztot sikerült bevonni a monitoring rendszerbe.

### 6.2.2. Tűzfal beállítása

Az operációs rendszerek telepítése során a tűzfal engedélyezését választottam. Ez azt jelentette, hogy szinte minden port zárva volt a külvilág felé, így a Prometheus szerver sem tudta elérni az exporterek által szolgáltatott adatokat.

Ahhoz, hogy a monitoring rendszer az elvárt módon működjön, szükséges volt a tűzfalszabályok módosítása. A telepített rendszereken a **firewalld** az alapértelmezett tűzfal, melynek a beállításait a **firewall-cmd** segédprogram segítségével tudtam módosíta-

**6.2. ábra.** Az Uyuni webes felületéről könnyen módosíthatjuk a Prometheus szerver alapvető beállításait.

ni (6.2. kódrészlet első sora). A kódrészlet harmadik sorában a szükséges portok kinyitását követő állapot látható az Uyunit futtató szerver esetében, a monitoring rendszerben használt exportereket, rövid leírásukat és a kapcsolódó portszámokat a 6.1. táblázat tartalmazza. A monitoring szerverhez tartozó Salt Formula alkalmazásával automatikusan telepítésre kerülnek a Tomcat és Taskomatic exporterek. Ezek az Uyuni alapjait adó webes keretrendszerrel és a használt feladatütemezőről biztosítanak statisztikákat.

```
1 uyuni:~ # firewall-cmd --permanent --add-port=9117/tcp
2 uyuni:~ # firewall-cmd --list-ports
3 5556/tcp 5557/tcp 9100/tcp 9117/tcp 9187/tcp 9800/tcp
```

**6.2. lista.** Tűzfalszabályok módosítása.

## 6.3. Riasztások beállítása

A Prometheus-alapú monitoring rendszer egyik legfontosabb komponense az Alertmanager, hiszen ennek segítségével a gyűjtött adatok folyamatos szemmel tartása nélkül

Portsám	Exporter	Leírás
5556	Tomcat JMX exporter	Az Uyuni működéséhez szükséges Tomcat alkalmazás Java-processzéről jelenít meg adatokat (pl. szálak, memóriahasználat).
5557	Taskomatic JMX exporter	Az Uyuni által használt Taskomatic alkalmazás Java-processzéről jelenít meg adatokat (pl. szálak, memóriahasználat).
9090	Prometheus exporter	A Prometheus állapotáról tesz közzé adatokat (pl. hibás lekérések száma, lekérések mérete).
9100	Node exporter	Linux hosztokról publikál számos hardver- és kernelmetrikát (pl. erőforráshasználat, kontextusváltások).
9117	Apache exporter	Az Apache webserverről ad meg mérőszámokat (pl. egyes HTTP-státusz kódokhoz tartozó lekérdezések száma, CPU-idő).
9187	PostgreSQL exporter	Az Uyuni által használt PostgreSQL adatbázismotorról oszt meg adatokat (pl. zárák száma, rekordlekérdezési és frissítési ráta).
9800	Taskomatic exporter	Az Uyuni által használt Taskomatic feladat-ütemező metrikáit jeleníti meg (pl. elvégzett feladatok száma, aktuális szálak száma).

**6.1. táblázat.** A monitoring rendszerben használt Prometheus exporterek a hozzájuk tartozó portszámokkal.

is értesülhetünk a rendszerünket érintő lényeges eseményekről. A 6.1. ábrán látható, hogy az Alertmanager egy különálló modul a monitoring-infrastruktúrában. Ez a modularitás a gyakorlatban is megjelenik: az Alertmanager a Prometheustól elkülönülten, a `golang-github-prometheus-alertmanager` csomaggal telepíthető, a telepítést követően pedig egy külön konfigurációs fájl és systemd unit fájl tartozik hozzá, azaz a Prometheus adatgyűjtő részétől elkülönülten kezelhető.

A tesztkörnyezetben az Alertmanager konfigurációját Salt state-tel végeztem. Ehhez létrehoztam egy új konfigurációs csatornát (configuration channel, 6.4. ábra), mely tartalmazta az Alertmanager konfigurációs állományát, valamint az ennek érvényre jutását biztosító state fájlt.

A riasztáskezelő alrendszer beállításának célja az volt, hogy egy monitorozott szolgáltatás elérhetetlenné válása esetén az azt figyelő rendszer küldjön értesítést e-mailben a hibáról. Ennek megvalósításához a 6.3. kódrészleten látható beállításokat használtam. Itt megfigyelhető a lokális SMTP-szerver megadása, és az értesítések küldéséhez használt e-mail cím beállítása. Az 5. sorban kezdődő `route` beállítás adja meg, hogy a specifi-





**6.3. ábra.** A monitoring rendszerbe bevont gépek a rajtuk futó exporterek által használt portokkal.

kusabb szűrőkre nem illeszkedő riasztásokkal hogy járjunk el. Ebben az esetben ezeket csoportosítjuk a riasztás típusa és az azt küldő hoszt alapján. A **receivers** címkén belül adhatjuk meg, hogy a riasztásokat milyen végpontokra küldje a rendszer. A konfigurációs fájlban szereplő beállítások egy egyszerű e-mailt küldenek a megadott címre. Mivel a titkosítás használata a lokális gépen futó levelezőszerver esetében nem indokolt, és konfigurációja összetett, ezért a 13. sorban látható **require\_tls** opcióval ezt kikapcsoltam. Fontos azonban, hogy ezt csak helyi mail szerver esetén tehetjük meg, amennyiben egy központi levelezőszervert használunk, akkor a program nem is engedi a titkosítás kikapcsolását [8]. A 14. sorban látható **send\_resolved** opciót igazra állítva a szolgáltatások helyreállításáról is kapunk értesítést. A teljes, kommentekkel ellátott konfigurációs fájlt a Függelék F.2.1. kódrészlete tartalmazza.

A fent ismertetett konfigurációs fájlt a 6.4. kódrészleten szereplő Salt state-tel telepítettem a monitoring szerverre. Ehhez a Salt által fájlok kezelésére nyújtott **file.managed** state opciót használtam, majd a kódrészlet 4. sorában szereplő **source** címkével megadtam az Uyuni által kezelt leírófájlt, valamint beállítottam a telepítendő fájl elérési útját és a kapcsolódó jogosultságokat (6.4. kódrészlet 5-7. sora). A Salt-tal való konfigurációmenedzsment a **service.running** state opcióval lehetőséget kínál a kapcsolódó service újratöltésére, amit a **watch** opció beállításával automatikusan is megtehetünk. Így tehát a kódrészlet 16. sorában megadott fájl módosulása esetén az alertmanager-t futtató systemd service automatikusan újraindításra kerül a módosítások érvényre juttatásával, ezzel is csökkentve a manuális beavatkozás szükségességét.

Alertmanager email alert ? Delete Channel

Overview **List/Remove Files** Add Files Systems

Channel Properties

Name: Alertmanager email alert

Label: alertmanager\_email

Description: Configure and enable e-mail notifications for Prometheus' AlertManager

Channel Information

Number of Files: 1 sls, 1 files (2 total )

Systems: 1 system subscribed

Most-Recently Subscribed System monitoring.spot.internal 17 hours ago

Most-Recently Modified File /etc/prometheus/alertmanager.yml by simon 14 hours ago

Configuration Actions

State File

View/Edit 'init.sls' File

Add/Create Files

Create configuration file

Upload Configuration Files

Import a File from Another Channel or System

**6.4. ábra.** A monitoring értesítések kezeléséhez használt Salt state.

```

1 global:
2   smtp_smarthost: "localhost:25"
3   smtp_from: "alertmanager@monitoring.spot.internal"
4
5 route:
6   receiver: "it-team"
7   group_by: ["alertname", "cluster"]
8
9 receivers:
10  - name: "it-team"
11    email_configs:
12      - to: "root+monitoring@monitoring.spot.internal"
13        require_tls: false
14        send_resolved: true

```

**6.3. lista.** Az e-mail-értesítések küldéséhez használt konfigurációs fájl részlete.

## 6.4. Adatvizualizáció

A gyűjtött adatok áttekintése, elemzése segítségünkre lehet további rendszerek telepítése során például szükséges erőforrások mennyiségének becsléséhez, illetve szolgáltatáskiesést követően segíthet a probléma meghatározásában.

A 6.2. alfejezetben ismertetettek alapján a tesztkörnyezetben a Grafana adatvizualizációs eszközt használtam. Ez jó integrációt biztosít a Prometheushoz, könnyen lekérdezhetőek a különböző metrikák. A telepítéshez használt Salt Formula pedig még egyszerűbbé tette a Grafana használatát: az alapértelmezett konfigurációs fájl számos előre definiált, a rendszerek állapotát jól összefoglaló dashboardot tartalmaz, így a tűzfalbeállítás és néhány panel webes felületen történő személyre szabásán túl nem volt szükséges részletebben konfigurálni.

```

1 update_/etc/prometheus/alertmanager.yml:
2   file.managed:
3     - name: /etc/prometheus/alertmanager.yml
4     - source: salt://manager_org_1/alertmanager_email/etc/prometheus/alertmanager.yml
5     - user: root
6     - group: root
7     - mode: 644
8
9 prometheus-alertmanager-service:
10  service.running:
11    - name: prometheus-alertmanager
12    # az alertmanager service automatikusan induljon el boot után
13    - enable: True
14    # a konfigurációs fájl változását követően a service automatikusan induljon újra
15    - watch:
16      - file: /etc/prometheus/alertmanager.yml

```

**6.4. lista.** Az e-mail-értesítések beállítását végző konfigurációs állomány telepítéséhez használt Salt state.

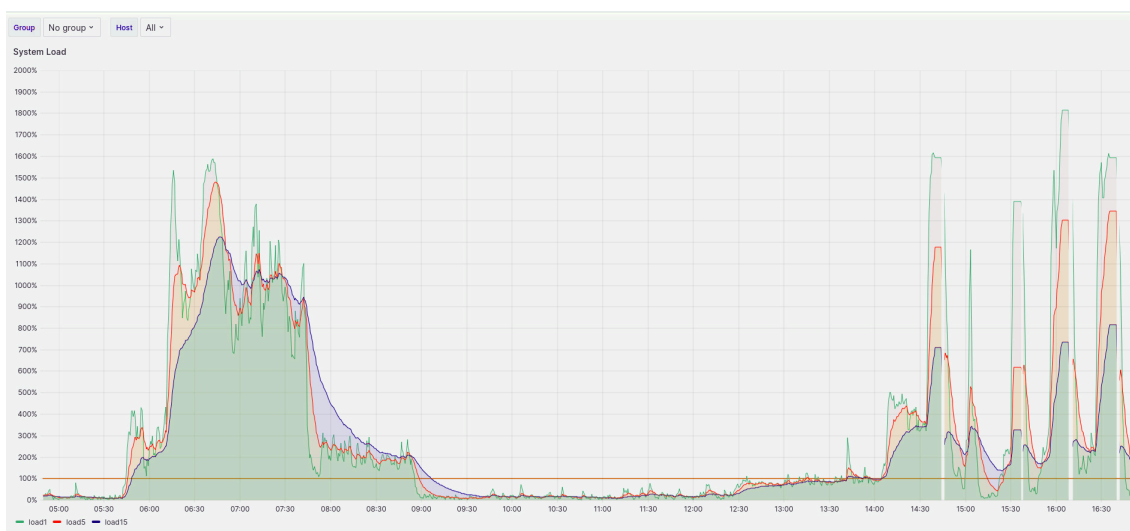
A 6.5. ábrán látható dashboard az Uyuni szerver állapotát ismerteti. Ez a dashboard is a Grafana Salt Formula része, néhány panel mértékegységének pontosításán és skálázásán kívül nem volt szükség a beállítások módosítására. Az ilyen nézetek előnye, hogy segítségükkel nagyon hamar megállapíthatjuk a rendszereink állapotát, hiszen minden lényeges mérőszámot tartalmaznak: segítségével figyelemmel követhetjük például a rendszer terheltségét, a processzor és a memória kihasználtságát, a hálózat- és háttértárhasználat rátáját, az adatbázis-műveletek állapotát és a háttértárak kihasználtságát.



**6.5. ábra.** A tesztkörnyezetben használt Grafana egyik dashboard-ja az Uyuni szerver fontosabb mérőszámaait mutatja be.

Bár a tesztkörnyezet körülbelül két-két és fél hónapos folyamatos üzemeltetése során nem voltak jelentős kimaradások, az Uyuni kezdetleges konfigurációja során az egyik tele-

pítőforrás tükrözése során többször is leállt az azt futtató gép, így valós helyzetben nyílt alkalmam a metrikák visszanézésére, és a hiba okának felkutatására. Az incidens során megjelenő grafikont a 6.6. ábra mutatja be. A grafikon jobb oldalán látható a négy leállítás, a bal szélén pedig láthatjuk, hogy néhány órával korábban hasonlóan magas terhelés mellett nem lépett fel a hiba. A grafikon  $y$  tengelyén láthatjuk, hogy az nem 100%-ig, hanem sokkal tovább fut. Ennek oka, hogy a rendszerterhelés számítása során az egy processzormagra eső maximális terhelést tekintjük a 100%-nak, így egy 24 CPU-val rendelkező gép esetén a maximális érték 2400% lenne. A grafikonon 1600% körülre esik a legnagyobb terhelés, ami összhangban van az Uyunit futtató virtuális gép által maximálisan használható 16 processzormaggal. Mint később kiderült, a hibát hardveres hiba okozta, a virtuális gép által használható processzormagok csökkentésével a probléma nem jelentkezett többé.



**6.6. ábra.** A telepítőforrás-tükrözés során tapasztalt szolgáltatás-kiesések a Grafanában megjelenő grafikonon.

## 7. fejezet

# Összefoglalás

### 7.1. Elért eredmények

Dolgozatomban egy átfogó képet adtam a nagyvállalati rendszerek üzemeltetése során felmerülő kihívásokról, ezek lehetséges hatékony megoldásairól. Részletesen kitértem a kötetkezelés, az infrastruktúra-menedzsment és a monitoring témakörökre, melyek egy nagyméretű infrastruktúra üzemeltetésének elengedhetetlen kellékei. Kitértem néhány konkrét megoldásra, egy kisméretű tesztkörnyezetben bemutattam ezek alkalmazási lehetőségeit, a tesztrendszer komponenseihez kapcsolódóan méréseket végeztem.

Munkám során megismerkedtem számos modern technológiával, és mélyrehatóan tanulmányoztam ezeket. A téma jó dokumentáltsága lehetőséget nyitott arra, hogy több különböző forrásból szerezzek információkat a megoldásokat illetően, és részletesen megismerkedjek egy-egy technológiával. Az ismeretek elsajátításához nagyban hozzájárultak a szoftvergyártók és a termékek hivatalos írásai, melyek magas minőségű tananyagoknak bizonyultak. A megismert lehetőségek megértésében nagy szerepet játszott a tesztkörnyezet, melyben lehetőségem nyílt a megoldások kipróbálására, és szabadon kísérletezhettem a különböző eszközökkel.

### 7.2. Továbbfejlesztési lehetőségek

Az értekezésem során részletesen tárgyaltam a kötetkezelés és az adattárolás témakörét, de a tesztkörnyezetben nem volt lehetőségem sem RAID-megoldás alkalmazására, sem pedig valamilyen biztonságimentés-szolgáltatás telepítésére vagy igénybe vételére. Úgy gondolom, hogy ezek egy valós környezetben elengedhetetlenek, hiszen egy adatvesztés további súlyos problémákat okozhat, ezért fontos, hogy egy szervezet mindig fel legyen készül-

ve az adatok visszaállítására, akár hardveres meghibásodás, akár rosszindulatú támadás okozta azt. Emiatt fontosnak tartom egy backup-keretrendszer megismerését, egy mentési stratégia kialakítását.

Ahogy már többször említettem a dolgozatom során, egy nagyvállalati környezetben nem ritka, hogy több száz vagy akár több ezer gépet kell kezelnünk. A tesztkörnyezetben ehhez képest mindössze három virtuális gépet és egy fizikai gépet üzemeltettem. Értelemszerűen nem reális, hogy egy ilyen jellegű környezetbe több száz gépet telepítsünk, de úgy gondolom, hogy például az infrastruktúra-menedzsment eszközök képességei jobban bemutatathatóak lettek volna, ha néhány géppel többet tudok kezelni (pl. egy külön web-szervert, egy adatbázisszervert és egy levelezőkiszolgálót). Mindemellett ezen rendszerek konfigurálása és összehangolása jelentős többletmunkával járt volna, helyes beállításuk (ide értve például a megfelelő titkosítási eljárások alkalmazását) körülbelül az eddig létrehozott környezettel azonos időt igényelt volna.

A lehetőségek tanulmányozása során nagyon tetszett Salt state-ek sokszínűsége, hogy egy pár soros leírófájllal mennyi mindent meg lehet oldani, viszonylag komplex dolgokat is. Biztos vagyok benne, hogy az itt érintettnél sokkal nagyobb mélységekbe lehet bocsátkozni a technológiát illetően, így a Salt-leírók továbbfejlesztése, szélesebb körű használata is jó kihívásnak ígérkezik.

A monitoring rendszer rendkívül sok irányban továbbfejleszthető, például Grafanában számtalan lehetőség áll rendelkezésünkre további adatok megjelenítésére, a Prometheusba bevonhatunk újabb exportereket, és az Alertmanager segítségével finomíthatjuk a riasztások kezelését (pl. más platformokon is jelezzem, vagy beállíthatunk részletesebb szűrést).

Láthatjuk, hogy a technológia fejlődésével és a trendek folyamatos változásával – például a manapság egyre népszerűbb konténerizációs megoldások népszerűségének növekedésével – a lehetőségek határtalanok, az eddigi munkám számos módon kiegészíthető.

# Köszönetnyilvánítás

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

# Szójegyzék

**hot swap** Számítógép-komponensek eltávolítása vagy hozzáadása egy futó rendszerhez (annak leállítása vagy újraindítása nélkül). 12

**hypervisor** Az a szoftver, amely koordinálja a virtuális gépek és az azokat futtató fizikai gép hardvere közötti interakciót. 9–12, 20, 21, 30, 57

**libvirt** Nyílt forráskódú virtualizációs API, mely lehetőséget biztosít több hypervisor (pl. KVM, XEN, VMware ESXi) egységes interfészen keresztüli kezelésére. 10, 11, 20

**overcommit** Hatékonyabb erőforráskihasználást lehetővé tevő technológia egyes hypervisorokban. Használatával a virtuális gépek számára több virtuális erőforrást oszthatunk ki, mint amennyi ténylegesen rendelkezésre áll a gazdagépen. Alapvetően előnyös lehet a használata, mivel a VM-ek általában nem használják ki a nekik biztosított erőforrások 100%-át, de a gazdagép túlterhelése esetén jelentős teljesítményvisszaeséssel járhat, ezért fontos, hogy használata esetén még körültekintőbben monitorozzuk rendszereinket [12]. 31



# Betűszavak

**API** application programming interface. 10, 19, 20, 57

**CPU** central processing unit. 11, 22, 32, 43, 46, 49, 53

**DNS** domain name system. 32, 46

**HA** high availability. 7, 23

**KVM** Kernel-based Virtual Machine. 9–11, 20, 30, 31, 36, 57

**LV** logical volume. 15, 31

**LVM** Logical Volume Manager. 1, 15, 16, 21, 24, 30–32, 34, 35

**OS** operating system. 9, 11, 13, 21, 23, 25, 30, 31, 34, 41

**PV** physical volume. 15, 37

**RAID** Redundant Array of Independent Disks. 14, 15

**RPM** RPM Package Manager, eredetileg Red Hat Package Manager. 17

**SLE** SUSE Linux Enterprise. 25, 26

**SMTP** Simple Mail Transfer Protocol. 49

**SPICE** Simple Protocol for Independent Computing Environments. 30

**SSD** solid state drive. 22

**SSH** secure shell. 7, 18, 24, 29, 30

**VG** volume group. 15, 36

**VM** virtual machine. 8, 11, 13, 20, 30, 31, 34, 36, 43, 57

**VNC** virtual network computing. 30

# Irodalomjegyzék

- [1] Andy Honig, Nelly Porter, Google Cloud: 7 ways we harden our KVM hypervisor at Google Cloud: security in plaintext (2024. május 5.). <https://cloud.google.com/blog/products/gcp/7-ways-we-harden-our-kvm-hypervisor-at-google-cloud-security-in-plaintext>.
- [2] Douglas DeMaio, openSUSE: openSUSE Leap 15.3 Bridges Path to Enterprise (2024. május 4.). <https://news.opensuse.org/2021/06/02/opensuse-leap-bridges-path-to-enterprise/>.
- [3] Robert P. Goldberg: *Architectural Principles for Virtual Computer Systems*. PhD értekezés (Harvard University). 1973.  
URL <https://apps.dtic.mil/sti/citations/AD0772809>.
- [4] Icinga: Icinga2 API (2024. május 10.). <https://icinga.com/docs/icinga-2/latest/doc/12-icinga2-api/>.
- [5] Jeff Reser, SUSE: Introducing SUSE Linux Enterprise 15 SP3 (2024. május 4.). <https://www.suse.com/c/introducing-suse-linux-enterprise-15-sp3/>.
- [6] Lindsay Clark: 1 in 5 VMware customers plan to jump off its stack next year (2024. május 15.). [https://www.theregister.com/2023/11/08/vmware\\_customer\\_forrester\\_prediction/](https://www.theregister.com/2023/11/08/vmware_customer_forrester_prediction/).
- [7] openSUSE: openSUSE Leap 15.3 Release Notes (2024. május 5.). [https://doc.opensuse.org/release-notes/x86\\_64/openSUSE/Leap/15.3/#installation-new-update-repos](https://doc.opensuse.org/release-notes/x86_64/openSUSE/Leap/15.3/#installation-new-update-repos).
- [8] Prometheus: Alerting: File layout and global settings (2024. május 14.). <https://prometheus.io/docs/alerting/latest/configuration/#file-layout-and-global-settings>.

- [9] Prometheus: Getting Started with Prometheus (2024. május 13.). [https://prometheus.io/docs/tutorials/getting\\_started/](https://prometheus.io/docs/tutorials/getting_started/).
- [10] Red Hat: Ansible vs. Salt: What you need to know (2024. május 10.). <https://www.redhat.com/en/topics/automation/ansible-vs-salt>.
- [11] Red Hat: Deprecated functionality (2024. május 5.). [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/9/html/9.4\\_release\\_notes/deprecated-functionality#deprecated-functionality-virtualization](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/9.4_release_notes/deprecated-functionality#deprecated-functionality-virtualization).
- [12] Red Hat: Overcommitting with KVM (2024. május 8.). [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/7/html/virtualization\\_deployment\\_and\\_administration\\_guide/chap-overcommitting\\_with\\_kvm](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/virtualization_deployment_and_administration_guide/chap-overcommitting_with_kvm).
- [13] Red Hat: Red Hat Adds Common Criteria Certification for Red Hat Enterprise Linux 8 (2024. április 24.). <https://www.redhat.com/en/about/press-releases/red-hat-adds-common-criteria-certification-red-hat-enterprise-linux-8>.
- [14] Red Hat: Red Hat Enterprise Linux: Logical Volume Manager Administration (2024. április 22.). [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/7/html/logical\\_volume\\_manager\\_administration/lvm\\_components](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/logical_volume_manager_administration/lvm_components).
- [15] Red Hat: Red Hat Virtualization (2024. május 5.). <https://access.redhat.com/products/red-hat-virtualization>.
- [16] Salt Project: About Salt (2024. május 18.). [https://docs.saltproject.io/en/latest/topics/about\\_salt\\_project.html#about-salt](https://docs.saltproject.io/en/latest/topics/about_salt_project.html#about-salt).
- [17] Salt Project: Salt security: More about keys (2024. május 10.). <https://docs.saltproject.io/salt/user-guide/en/latest/topics/security.html>.
- [18] Seagate: What is a 3-2-1 backup strategy? (2024. április 7.). <https://www.seagate.com/gb/en/blog/what-is-a-3-2-1-backup-strategy/>.
- [19] Statista: Market share of leading virtualization platform technologies worldwide as of December 2023 (2024. május 15.). <https://www.statista.com/statistics/1252355/top-virtualization-technologies-by-domain/>.

- [20] SUSE: SUSE Certifications and Features (2024. április 24.). <https://www.suse.com/support/security/certifications/>.
- [21] SUSE: SUSE Linux Enterprise Server 15 SP5 – Virtualization Guide (2024. április 11.). [https://documentation.suse.com/sles/15-SP5/pdf/book-virtualization\\_en.pdf](https://documentation.suse.com/sles/15-SP5/pdf/book-virtualization_en.pdf).
- [22] SUSE: SUSE Linux Enterprise Server Documentation: Managing Networks (2024. május 6.). <https://documentation.suse.com/sles/15-SP2/html/SLES-all/cha-libvirt-networks.html>.
- [23] Uyuni Project: Installation and Upgrade Guide (2024. május 8.). [https://www.uyuni-project.org/doc/2024.02/uyuni\\_installation-and-upgrade\\_guide.pdf](https://www.uyuni-project.org/doc/2024.02/uyuni_installation-and-upgrade_guide.pdf).
- [24] Wikipedia: Hypervisor (2024. május 15.). <https://en.wikipedia.org/wiki/Hypervisor>.
- [25] Wikipedia: Standard raid levels (2024. április 7.). [https://en.wikipedia.org/wiki/Standard\\_RAID\\_levels](https://en.wikipedia.org/wiki/Standard_RAID_levels).
- [26] XEN Project: Xen project 4.4 release notes (2024. március 20.). [https://wiki.xenproject.org/wiki/Xen\\_Project\\_4.4\\_Release\\_Notes](https://wiki.xenproject.org/wiki/Xen_Project_4.4_Release_Notes).
- [27] Yev Pusin, Backblaze: The 3-2-1 backup strategy (2024. április 7.). <https://www.backblaze.com/blog/the-3-2-1-backup-strategy/>.

# Függelék

## F.1. A monitoring beállítása előtti adatgyűjtéshez használt program

```
1  #!/usr/bin/python
2
3  import psutil
4  import csv
5  import sys
6
7  from datetime import datetime
8
9  script_start = datetime.now()
10 field_names = [
11     "timestamp",
12     "timedelta_seconds",
13     "cpu_usage_percentage",
14     "cpus_usage_percentage",
15     "cpu_time_user",
16     "cpu_time_system",
17     "cpu_time_idle",
18     "cpu_time_iowait",
19     "cpu_time_irq",
20     "cpu_time_softirq",
21     "cpu_time_steal",
22     "cpu_ctx_switches",
23     "cpu_interrupts",
24     "cpu_soft_interrupts",
25     "cpu_syscalls",
26     "cpu_freq_current",
27     "cpu_freq_min",
28     "cpu_freq_max",
29     "sys_load_1min",
30     "sys_load_5min",
31     "sys_load_15min",
32     "mem_total",
33     "mem_available",
34     "mem_usage_percentage",
35     "mem_used",
36     "mem_free",
37     "mem_active",
38     "mem_inactive",
39     "mem_buffers",
40     "mem_cached",
41     "mem_shared",
42     "swap_total",
43     "swap_used",
44     "swap_free",
45     "swap_usage_percentage",
46     "swap_sin",
47     "swap_sout",
48     "disk_read",
49     "disk_write",
50     "disk_read_rate",
51     "disk_write_rate",
52     "disk_usage",
```

```

53     "net_sent",
54     "net_recv",
55     "net_send_rate",
56     "net_recv_rate",
57     "users",
58     "processes"
59 ]
60
61 with open(f"sysstats_{datetime.now().isoformat()}.csv", "w", newline="") as csvfile:
62     writer = csv.DictWriter(csvfile, fieldnames=field_names)
63     writer.writeheader()
64
65     io_lastread = 0
66     io_lastwrite = 0
67
68     net_lastsend = 0
69     net_lastrecv = 0
70
71     while True:
72         try:
73             cpu_usage_percentage = psutil.cpu_percent(interval=1)
74             cpu_times = psutil.cpu_times()
75             cpu_stats = psutil.cpu_stats()
76             cpu_freq = psutil.cpu_freq()
77             sys_load = psutil.getloadavg()
78             memory = psutil.virtual_memory()
79             swap = psutil.swap_memory()
80             io = psutil.disk_io_counters()
81
82             disk_usage = dict()
83             for disk in psutil.disk_partitions():
84                 disk_usage[disk.mountpoint] = psutil.disk_usage(disk.mountpoint).percent
85
86             net = psutil.net_io_counters()
87
88             processes = list()
89             monitored_processes = [ "postgre", "pgsql", "spacewalk", "uyuni", "tomcat", "salt" ]
90             all_processes = psutil.process_iter()
91             for proc in all_processes:
92                 for monproc in monitored_processes:
93                     if monproc in proc.name().lower():
94                         process_details = dict()
95                         process_details["name"] = proc.name()
96                         pid = proc.pid
97                         monproc_obj = psutil.Process(pid)
98                         process_details["cpu_usage"] = monproc_obj.cpu_percent()
99                         process_details["mem_usage"] = monproc_obj.memory_percent()
100                        process_details["status"] = monproc_obj.status()
101                        process_details["user"] = monproc_obj.username()
102                        process_details["create_time"] = monproc_obj.create_time()
103                        process_details["io_counters"] = monproc_obj.io_counters()
104                        process_details["cpu_times"] = monproc_obj.cpu_times()
105                        process_details["open_files"] = monproc_obj.open_files()
106                        process_details["num_threads"] = monproc_obj.num_threads()
107                        process_details["cmdline"] = monproc_obj.cmdline()
108                        processes.append(process_details)
109
110
111             writer.writerow(
112                 {
113                     field_names[0]: datetime.now(),
114                     field_names[1]: (datetime.now() - script_start).total_seconds(),
115                     field_names[2]: cpu_usage_percentage,
116                     field_names[3]: psutil.cpu_percent(percpu=True),
117                     field_names[4]: cpu_times.user,
118                     field_names[5]: cpu_times.system,
119                     field_names[6]: cpu_times.idle,
120                     field_names[7]: cpu_times.iowait,
121                     field_names[8]: cpu_times.irq,
122                     field_names[9]: cpu_times.softirq,
123                     field_names[10]: cpu_times.steal,
124                     field_names[11]: cpu_stats.ctx_switches,

```

```

125         field_names[12]: cpu_stats.interrupts,
126         field_names[13]: cpu_stats.soft_interrupts,
127         field_names[14]: cpu_stats.syscalls,
128         field_names[15]: cpu_freq.current,
129         field_names[16]: cpu_freq.min,
130         field_names[17]: cpu_freq.max,
131         field_names[18]: sys_load[0],
132         field_names[19]: sys_load[1],
133         field_names[20]: sys_load[2],
134         field_names[21]: memory.total,
135         field_names[22]: memory.available,
136         field_names[23]: memory.percent,
137         field_names[24]: memory.used,
138         field_names[25]: memory.free,
139         field_names[26]: memory.active,
140         field_names[27]: memory.inactive,
141         field_names[28]: memory.buffers,
142         field_names[29]: memory.cached,
143         field_names[30]: memory.shared,
144         field_names[31]: swap.total,
145         field_names[32]: swap.used,
146         field_names[33]: swap.free,
147         field_names[34]: swap.percent,
148         field_names[35]: swap.sin,
149         field_names[36]: swap.sout,
150         field_names[37]: io.read_bytes,
151         field_names[38]: io.write_bytes,
152         field_names[39]: io.read_bytes - io_lastread,
153         field_names[40]: io.write_bytes - io_lastwrite,
154         field_names[41]: disk_usage,
155         field_names[42]: net.bytes_sent,
156         field_names[43]: net.bytes_recv,
157         field_names[44]: net.bytes_sent - net_lastsend,
158         field_names[45]: net.bytes_recv - net_lastrecv,
159         field_names[46]: len(psutil.users()),
160         field_names[47]: processes
161     }
162 )
163
164 io_lastread = io.read_bytes
165 io_lastwrite = io.write_bytes
166
167 net_lastsend = net.bytes_sent
168 net_lastrecv = net.bytes_recv
169 except KeyboardInterrupt:
170     sys.exit(0)
171 except psutil.NoSuchProcess:
172     print("ERROR - Process doesn't exist")
173     continue
174 except Exception as e:
175     # only print the exception, but do not exit
176     print(f"ERROR - General exception: {e}")
177     continue

```

**F.1.1. lista.** A monitorozáshoz készített, psutil csomagot használó Python-script, mely csv formátumban menti el a metrikákat.



## F.2. E-mail-értesítések beállítását tartalmazó konfigurációs fájl

```
1 global:
2   # SMTP-beállítások
3   smtp_smarthost: "localhost:25"
4   smtp_from: "alertmanager@monitoring.spot.internal"
5
6   # A legfelső szintű útvonal, amibe beérkeznek a riasztások, amiket további
7   # 'route'-ok definiálásával lehet részletesebben szűrni
8   route:
9     # A részletesebb szűrőkre nem illeszkedő értesítéseket megkapó csoport
10    receiver: "it-team"
11
12    # Összetartozó riasztások csoportosítása a riasztások minimalizálása érdekében
13    group_by: ["alertname", "cluster"]
14
15    # Ennyit várunk egy riasztás megérkezésétől számítva az értesítés kiküldéséig,
16    # hogy ha több szolgáltatás is érintett, akkor azok egy értesítésbe
17    # csoportosítva kerüljenek kiküldésre
18    group_wait: 30s
19
20    # Az első riasztást követően ennyit várunk a csoportot érintő további riasztások
21    # kiküldése előtt
22    group_interval: 5m
23
24    # Sikeres küldés esetén ennyi ideig nem küldjük újra az értesítéseket
25    repeat_interval: 3h
26
27    # Példa egy gyerek útvonalra, mely a kritikus riasztások küldését végzi
28    routes:
29      - match:
30        severity: critical
31        receiver: it-team-critical
32
33    # Ha egy szolgáltatás már 'critical' állapotban van, akkor a 'warning' állapotra
34    # ne küldjön riasztásokat
35    inhibit_rules:
36      - source_match:
37        severity: "critical"
38        target_match:
39        severity: "warning"
40        equal: ["alertname"]
41
42    # Az értesítéseket megkapó címzettek
43    receivers:
44      - name: "it-team"
45        email_configs:
46          - to: "root+monitoring@monitoring.spot.internal"
47            require_tls: false
48            send_resolved: true
49
50      - name: "it-team-critical"
51        email_configs:
52          - to: "root+monitoring-critical@monitoring.spot.internal"
53            require_tls: false
54            send_resolved: true
```

**F.2.1. lista.** A monitoring rendszerben e-mail-értesítések küldésére használt konfigurációs fájl.