

# **SZAKDOLGOZAT**

PATKÓS MÁTÉ LEVENTE

DEBRECEN

2024

DEBRECENI EGYETEM

INFORMATIKAI KAR

INFORMÁCIÓ TECHNOLÓGIA TANSZÉK

AUTÓMATIZÁCIÓ NAGYVÁLLALATI KÖRNYEZETBEN

AUTOMATIZÁCIÓ ANSIBLE HASZNÁLATÁVAL

Témavezető:

Dr. Krausz Tamás

*Egyetemi Adjunktus*

Szerző:

Patkós Máté Levente

*Mérnökinformatikus BSc*

# Tartalomjegyzék

1. Bevezetés .....	3
2. Felhasznált eszközök .....	5
2.1 A felhasznált fizikai eszköz .....	5
2.2 A felhasznált szoftverek .....	5
2.2.1 Ansible .....	5
2.2.2 Proxmox .....	6
2.2.3 Semaphore .....	7
2.2.4 Nagios Core .....	7
3. Környezet kiépítése .....	8
4. Semaphore .....	13
4.1 Semaphore bevezetés, működés .....	13
4.2 Semaphore telepítés, konfiguráció .....	13
5. Ansible .....	18
5.1 Ansible bevezetés .....	18
5.2 Automatizált feladatok .....	19
5.3 Egyéb, fel nem használt feladatok .....	29
5.4 Ansible Vault .....	30
6. További fejlesztési lehetőségek .....	31
6.1 DNS (Domain Name System) .....	31
6.2 SQL (Structured Query Language) .....	31
6.3 Telefonhívás monitoring jelzés esetén .....	32
7. Összegzés .....	33
8. Köszönetnyilvánítás .....	34
9. Irodalomjegyzék .....	35

# 1. Bevezetés

Napjainkban el sem tudnánk képzelni az életünket automatizált folyamatok nélkül. Vegyük példának az okos otthon eszközöket, robotporszívó gépeket vagy akár a pénzügyeinket, mint a számlák automatikus befizetése vagy ütemezett pénzáttutalások. Mindezek a mindennapi dolgok miatt tartom nagyon fontosnak az IT iparágon belüli automatizálást is.

Az automatizálással növelhetjük a hatékonyságot, mivel ezáltal felszabadul az informatikai személyzet az ismétlődő, monoton feladatok alól, mindezzel csökkentve a költségeket beleértve a munkaerőköltséget, valamint a hardver- és szoftverhibákból eredő költségeket. Az automatizált folyamatok bevezetésével és használatával javíthatunk a pontosságon, mivel csökkenti az emberi hibák lehetőségét és javítja az IT folyamatok produktivitását. Az elmúlt évtizedek alatt megvalósult informatikai és technológiai fejlődésnek köszönhetően alkalmazhatunk egy olyan eljárási folyamatot, ahol nem kell a humán döntéshozatalra alapozni, hanem csupán előre megírt algoritmus alapján az automatizációs szerver végrehajtja az általunk megírt feladatokat.

A későbbiekben bemutatásra kerül a virtualizáció, mivel ezen eljárás szintén nagyon fontos egy nagyvállalati környezetben, hiszen ennek segítségével sokkal hatékonyabban használhatjuk ki a fizikai hardverjeinket. A virtualizáció nem más, mint egy fizikai hardver erőforrásainak szétosztása több virtuális gépre. Ezáltal úgy tűnhet, mintha minden virtuális gépnek saját, dedikált hardverje lenne, viszont a fizikai szerver erőforrásait használják ki.

Szakdolgozatom célja, hogy bemutassam, hogyan is kell elképzelni egy ilyen automatizált környezetet, milyen módon lehet kiépíteni, avagy megvalósítani. Nem egy teljes nagyvállalati környezetet szerettem volna bemutatni, hanem annak csak egy csekély részét, hogy azt, miként és hogyan lehet automatizálni.

A technológiák kiválasztásának fő szempontjai azok voltak, hogy ne egy mindenki által ismert megoldást mutassak be sokadjára, szerettem volna kevésbé ismert megoldási lehetőségeket bemutatni. Az általam használt technológiák mind nyílt forráskódúak, ezáltal ingyenesen használhatóak, amely rendkívül megkönnyítette a munkámat, mivel nem kellett licenszelt szoftvereket vásárolnom.

Mindemellett munkám során volt lehetőségem az ismert technológiák használatára, szerettem volna jobban elmélyülni ebben a témában, és ennek a legjobb módjának azt láttam, hogy megvalósítom otthoni környezetben, majd készítek belőle egy szakdolgozatot.

## 2. Felhasznált eszközök

### 2.1 A felhasznált fizikai eszköz

Fizikai szerver (Intel Xeon E5-2680 v4 @2.40Ghz, 32GB RAM, 128GB SSD, 2 1TB HDD)

### 2.2 A felhasznált szoftverek

2.2.1. Ansible automatizációs eszköz

2.2.2. Proxmox virtualizációs környezet (PVE)

2.2.3. Semaphore kezelői felület Ansible-höz

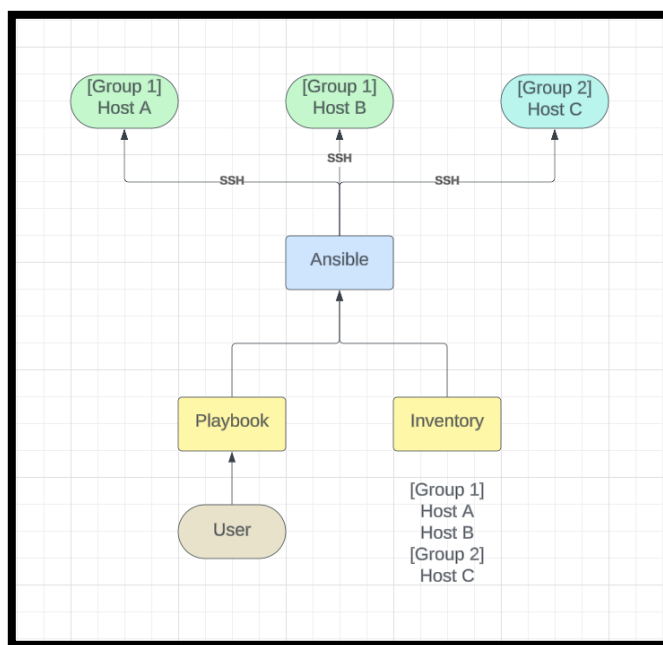
2.2.4. Nagios Core monitoring

#### 2.2.1 Ansible

Az Ansible<sup>1</sup> egy nyílt forráskódú, ügynök (továbbiakban agent) nélküli konfigurációkezelő és automatizálási eszköz, amelyet nagyszámú szerver hatékony üzemeltetésére és konfigurálására használhatunk. Kis- és nagyvállalkozások egyaránt igénybe vehetik. (1. ábra)

Előnyei:

- YAML formátumú konfiguráció leíró fájlokkal (playbook-okkal) dolgozik, ami könnyen olvasható és írható.
- Nem igényel agent telepítést a célgépekre, így kisebb a karbantartási teher. Képes nagyszámú szervert kezelni.
- A konfigurációkezelésen kívül, például alkalmazástelepítésre is használható.
- Az Ansible ereje a modularitásában és az egyszerű használatban rejlik, ami lehetővé teszi az IT infrastruktúra hatékony és megbízható automatizálását.
- Az Ansible idempotens, ami azt jelenti, hogy a playbook többszöri futtatása ugyanazt az eredményt fogja elérni, ha a rendszer állapota nem változott. Ezáltal csökkenti a hibák lehetőségét.



1. ábra, Ansible

Forrás: saját szerkesztés

## 2.2.2 Proxmox

A Proxmox VE<sup>2</sup> egy nyílt forráskódú virtualizációs platform, amely lehetővé teszi virtuális gépek és konténerek futtatását egyetlen fizikai szerveren. Ez a megoldás ideális olyan esetekben, ahol több rendszert vagy alkalmazást kell futtatni, de nincs hardveres erőforrás mindegyik számára. A Proxmox VE integrált webes felülettel is rendelkezik, és számos funkciót kínál, többek között:

- KVM alapú virtualizációval készült virtuális gépek futtatását.
- LXC konténerek futtatását erőforrás-hatékonyabb izolációval.
- Tárolókezelést virtuális lemezek és hálózati adattárolók számára.
- Hálózatkezelést virtuális switch-ek és VLAN-ok használatával.

### 2.2.3 Semaphore

A Semaphore<sup>3</sup> egy nyílt forráskódú webes kezelői felület az Ansible-höz. Ezen felületen keresztül tudunk futtatni Ansible playbook-okat. A Semaphore pure Go nyelven íródott, támogat MySQL-t, PostgreSQL-t és BoltDB-t is. Ezzel a kezelői felülettel lehet készíteni felhasználói csoportokat a biztonság miatt, akár egyes felhasználóknak adhatunk jogosultságot bizonyos playbook-ok futtatásához. Képes a leltárat (továbbiakban inventory), a gyűjteményt (továbbiakban repository) és a hozzáférési kulcsokat is menedzselni. Továbbá képes a playbook-ok időzített futtatására, ezen felül vissza lehet nézni az egyes playbook-ok futási kimenetelét is.

### 2.2.4 Nagios Core

A Nagios Core<sup>4</sup> egy ingyenes, nyílt forráskódú monitoring eszköz. Fő feladata, a szerverek, hálózati eszközök, alkalmazások és szolgáltatások állapotának folyamatos figyelése, valamint a hibák és problémák gyors észlelése. (2. ábra)

Előnyei:

- Szinte bármely komponenst képes monitorozni, legyen az hardver vagy szoftver.
- Akár több száz vagy ezer eszköz egyidejű monitorozására is alkalmas.
- Automatikusan képes riasztást küldeni e-mailben, telefonon vagy más csatornán keresztül.



serv-monitoring	Current Load	OK	03-07-2024 14:04:33
	Current Users	OK	03-07-2024 13:59:45
	PING	OK	03-07-2024 14:00:55
	Root Partition	OK	03-07-2024 14:01:38
	SSH	OK	03-07-2024 14:02:16
	Swap Usage	OK	03-07-2024 14:03:30
	Total Processes	OK	03-07-2024 14:04:16

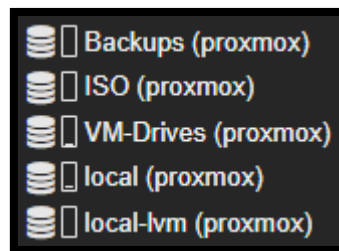
2. ábra, Nagios

*Forrás: saját szerkesztés*



### 3. Környezet kiépítése

A fizikai szerver 128GB-os SSD-jére telepítettem a Proxmox VE-t, ami nagyon egyszerű feladat volt számomra, a remek dokumentációnak<sup>5</sup> hála. A 2db 1TB-os HDD-ből készítettem egy RAID 1 ZFS Pool-t, mivel a RAID 1 eljárás, „tükrözés” technológia, biztosítja az adatok egyidejű tárolást a tömb minden elemén. Így, ha bármi történik az egyik merevlemezrel, megmarad minden adat a másikon. Ezen ZFS Pool-on belül készítettem több mappát, hogy átlátható legyen a környezet. (3. ábra)



3. ábra, Proxmox ZFS

*Forrás: saját szerkesztés*

A Backups a biztonsági mentéseknek szolgál, az ISO mappába kerültek az LXC konténer és ISO képfájlok. Az ISO fájl egyetlen fájl, amely egy optikai lemez, például CD vagy DVD teljes tartalmának pontos másolata. Jelen esetben, az ISO fájl biztosítja a virtuális gép operációs rendszerének telepítését. A VMDrives pedig a virtuális gépek, konténerek lemezeinek ad helyet. A biztonsági mentések készítéséhez a Proxmox beépített backup megoldását használtam. (4. ábra)

Fizikai erőforrás hiányában kellett ezt a megoldást használnom, nem volt lehetőségem egy külön backup szervert készíteni erre a feladatra. Ha lett volna egy másik fizikai szerverem, akkor szintén a Proxmox megoldását használtam volna, a Proxmox Backup Server-t. Ez a szoftver szintén nyílt forráskódú, ami megállja a helyét egy nagyvállalati környezetben.

Említésre méltó, ellenben nem nyílt forráskódú egyéb megoldások:

- IBM Tivoli Storage Manager

Az IBM Tivoli Storage Manager (TSM), mostanában IBM Spectrum Protect néven ismert, egy vállalati szintű adatvédelmi platform, amely lehetővé teszi a vállalatok számára az adatmentések és visszaállítások központosított vezérlését és felügyeletét. Ennek segítségével biztonsági mentéseket lehet készíteni fontos adatokról, archiválhatjuk a ritkán használt adatokat.

A TSM több összetevőből áll:

- TSM szerver: ez a központi komponens, amely kezeli az összes többi TSM komponenst és a biztonsági mentési folyamatot.
- TSM kliens: ez a szoftver fut a számítógépeken, amelyeket biztonsági menteni szeretnénk, és kommunikál a TSM szerverrel.
- TSM tárolóügynök: ez a szoftver a biztonsági mentési eszközökön fut, például szalagos meghajtókon, és kezeli az adatátvitelt a számítógép és a tárolóeszköz között.

- Dell Avamar

A Dell Avamar egy adatbiztonsági mentés és visszaállítási szoftver, amely a globális deduplikációs technológiát alkalmazza az ügyfél számítógépen. Ez azt jelenti, hogy a redundáns biztonsági mentési adatokat törli a kliensgépeken, mielőtt azokat tárolná, ezzel csökkentve a szükséges tárhely mennyiségét. Továbbá, az Avamar támogatja a biztonsági mentés és visszaállítási műveleteket az AWS-ben (Amazon Web Services), a Microsoft Azure-ban és a Google Cloud-ban, vagyis a felhőtechnológiába integrálható.

Összességében, az IBM Tivoli Storage Manager és a Dell Avamar is egy hatékony megoldás a nagyvállalatok számára, hogy megvédjék adataikat és visszaállítsák azokat, fizikai vagy egyéb meghibásodás esetén, adatvesztés nélkül.

Edit: Backup Job

General
Retention
Note Template

Node: proxmox
Storage: Backups
Schedule: 6:00
Selection mode: Include selected VMs
Notification mode: Default (Auto)
Send email: Always
Send email to:
Compression: ZSTD (fast and good)
Mode: Snapshot
Enable: ☒

Job Comment:

<input type="checkbox"/>	ID ↑	Node	Status	Name	Type
<input type="checkbox"/>	100	proxmox	running	serv-monitoring	LXC Container
<input type="checkbox"/>	101	proxmox	running	serv-semaphore	LXC Container
<input type="checkbox"/>	102	proxmox	running	ubi-lxc	Virtual Machine
<input type="checkbox"/>	110	proxmox	running	test-disk	LXC Container
<input type="checkbox"/>	111	proxmox	running	test-ansible	LXC Container
<input type="checkbox"/>	200	proxmox	running	test-standard-1	LXC Container
<input type="checkbox"/>	201	proxmox	running	test-standard-2	LXC Container
<input type="checkbox"/>	202	proxmox	running	test-standard-3	LXC Container
<input type="checkbox"/>	1000	proxmox	stopped	template-standard	LXC Container

Advanced ☐
OK
Reset

4. ábra, Proxmox Backup

Forrás: saját szerkesztés

A 4. ábrán látható Proxmox Backup esetén, a második Retention fülnél lehet beállítani, hogy meddig tartsa meg a rendszer a biztonsági mentéseket, jelenleg az én esetemben úgy van konfigurálva, hogy csak az utolsó három biztonsági mentést tartja meg. Ezenfelül, van még lehetőség óránkénti, napi, heti, havi vagy akár éves bizonyos számú biztonsági mentés megtartására.

Készítettem továbbá néhány LXC konténert, egy Ubuntu konténer fájlból. Ebben az esetben a gépek létrehozásánál kell megadni egy root felhasználóhoz tartozó jelszót - ha van - SSH publikus kulcsot (én használtam), hogy melyik képfájlból/konténer fájlból készítse a virtuális gépet, lemez méretet, CPU számot, RAM mennyiséget, továbbá lehetőség van a hálózati beállításoknál DHCP használatára, én ezt választottam. (5. ábra)

Key ↑	Value
cores	1
features	nesting=1
memory	512
net0	name=eth0,bridge=vmbr0,firewall=1,ip6=dhcp,ip=dhcp
nodename	proxmox
ostemplate	ISO:vztmpl/ubuntu-22.04-standard_22.04-1_amd64.tar.zst
pool	
rootfs	VM-Drives:8
ssh-public-keys	
swap	512
unprivileged	1
vmid	103

☐ Start after created

Advanced ☐ Back Finish

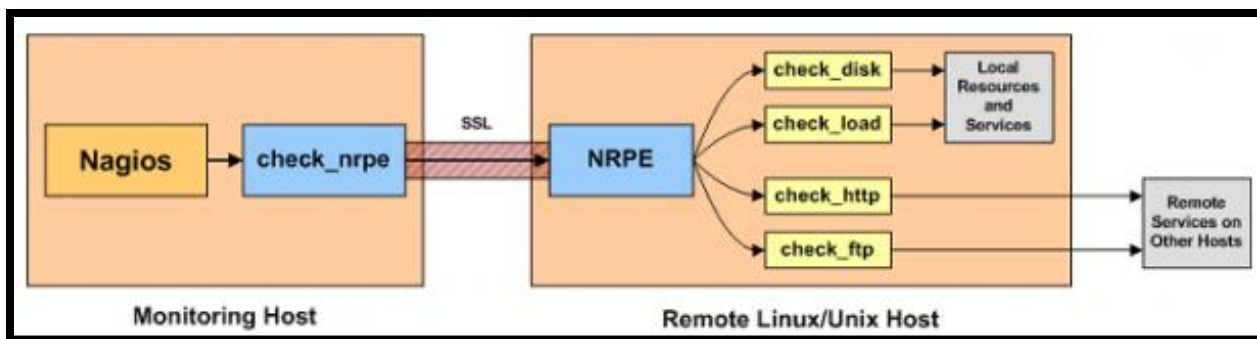
5.ábra, Proxmox LXC

Forrás: saját szerkesztés

Az SSH nem más, mint egy titkosított kapcsolati protokoll, amely biztonságos bejelentkezést biztosít. Az `ssh-keygen` parancs, létrehoz egy 4096 bit-es SSH RSA privát és publikus kulcsot a `~/.ssh` könyvtárba. A publikus kulcs tartalmát a `~/.ssh/authorized_keys` fájlba másoltam. Ezzel a kliens oldali SSH beállítás készen van. A privát kulcsot az SSH kliens kapcsolati beállításában kell megadni – én a MobaXterm SSH klienst használtam ehhez. Az SSH kapcsolat alapértelmezett esetben a 22-es számú TCP porton történik.

A DHCP, egy olyan számítógépes hálózati kommunikációs protokoll, ami azt oldja meg, hogy a TCP/IP hálózatra csatlakozó hálózati végpontok automatikusan megkapják a hálózat használatához szükséges beállításokat. Ezek a beállítások az IP-cím, hálózati maszk, alapértelmezett átjáró. Ez a megoldás nem állja meg a helyét egy nagyvállalati környezetben, mert ajánlatos fix IP címek használata ilyen esetben. Azért ajánlatos a fix IP cím használata a dinamikus helyett, mert nincs szükség ezen azonosító változására, mivel a szerverek és adatbázisok ugyanazon az IP címen keresztül kell, hogy elérhetőek legyenek, amely előzőleg beállításra került.

Készítettem egy minta-gépet (továbbiakban template), amit majd a későbbiekben fogok használni a kliens gépek létrehozására. Ezen template-re telepítettem előzőleg a Nagios-hoz tartozó kliens oldali csomagot, az NRPE-t (7. ábra), amit használva csatlakozni tud a Nagios szerver a kliens gépekhez, és különböző plugin-okat használva le tudja kérni a hardver, illetve a szoftverek állapotát. A Nagios szerver és a kliens gépeken lévő NRPE plugin közötti kommunikáció alapértelmezetten az 5666-os TCP porton történik, mindez fölül írható az NRPE konfigurációs fájljában.



7. ábra, Nagios NRPE <sup>6</sup>

<sup>6</sup> Forrás: <https://exchange.nagios.org/directory/image/93> (Utolsó megtekintés ideje: 2024. 04. 04.)

## 4. Semaphore

### 4.1 Semaphore bevezetés, működés

A Semaphore egy olyan koordinációs eszköz, amely megakadályozza, hogy az Ansible playbook-ok futtatása során ne történjen ütközés. Ez akkor hasznos, ha egy gépen vagy gépcsoporton több playbook is fut. Minden szkript indítással addig vár a Semaphore, amíg az előző be nem fejeződik, ezáltal sokkal kisebb a hibafaktor. Hátrányt jelent viszont, hogy ezáltal lassabb a playbook-ok futási sebessége. Említésre méltó a Red Hat megoldása, az Ansible Tower. A Semaphore-ral ellentétben az Ansible Tower egy nem ingyenes megoldás, viszont fontos előnye a felhőbe való integrálhatósága – mint Microsoft Azure, Amazon Web Services és Google Cloud.

### 4.2 Semaphore telepítés, konfiguráció

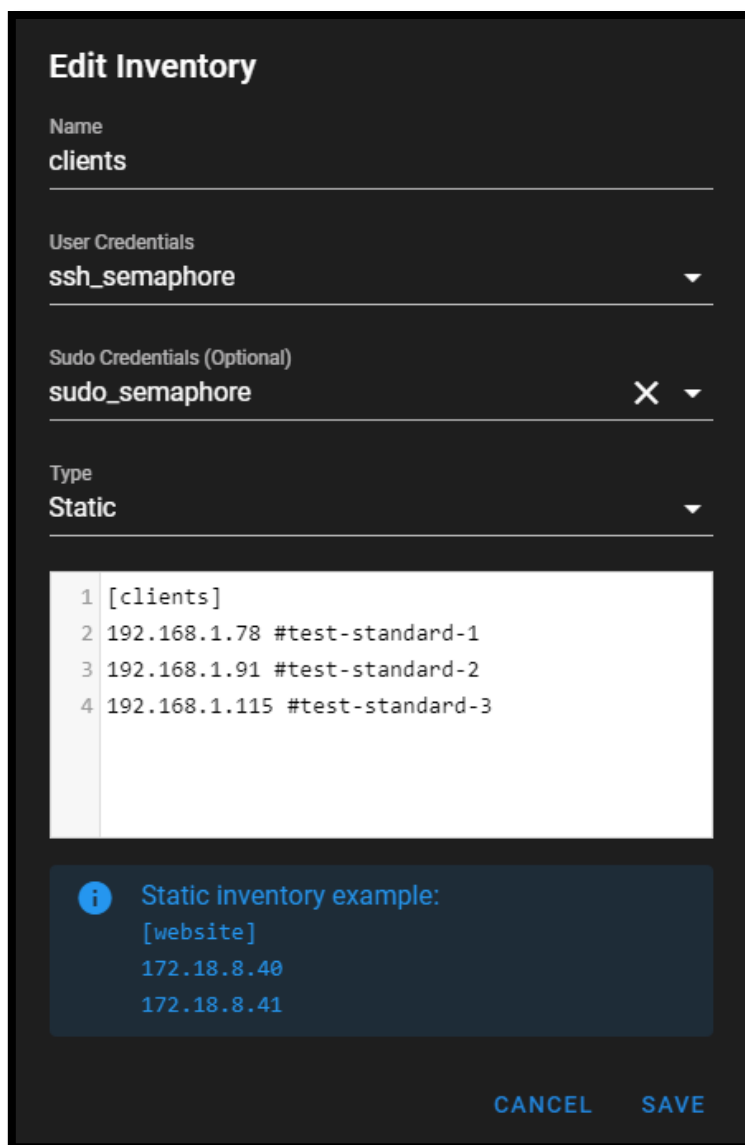
A csomagok telepítése nem okozott gondot, ugyanis a hivatalos telepítési dokumentációt<sup>7</sup> követve, a szükséges parancsok kiadása megtörtént, ezáltal a telepítés sikeres volt. A Semaphore web interfészen bejelentkezve, készíteni kell egy projektet. A projekt létrehozása után a legfontosabb lépések a következők:

- egy biztonságos csatlakozási lehetőség beállítása a kliens oldali gépek felé,
- legalább egy inventory fájl készítése,
- playbook-ok írása, amiket lehet majd futtatni a Semaphore-on keresztül.

A webes kezelői felület konfigurációját a soron következő bekezdésekben taglalom, hogy az egyes menüpontokban milyen lépéseket kell végrehajtani ahhoz, hogy a Semaphore megfelelően működjön.

A Key Store menüpontban lehet megadni az előzőleg generált SSH privát kulcsot. Lehet megadni ezen felül egy felhasználónevet is, ha valamiért nem sikerülne az SSH csatlakozás, ezt használva tud csatlakozni a gépekhez. Ugyanebben a menüpontban szükséges megadni még egy `sudo` felhasználónév-jelszó párost, ha lenne olyan playbook-unk, ami emelt jogosultságokkal való futtatást igényelne.

Az Inventory menüpontban lehet létrehozni az inventory fájlokat. Minden inventory fájlnek kell adni egy nevet, azt, hogy mely felhasználónév-jelszó párost használja az inventory fájlokban lévő kliens gépekhez való csatlakozásra, továbbá, ha szükséges, **sudo** jogosultsággal rendelkező felhasználónév-jelszó párost. Miután ezek a mezők kitöltésre kerültek, ezután van lehetőség az inventory fájl elkészítésére. Ezek lehetnek statikusak, melyek magában a Semaphore-ban tárolódnak, vagy dolgozhatnak egy már meglévő fájlból, ami magán az Ansible szerveren tárolódik lokálisan (8. ábra).



**Edit Inventory**

Name  
**clients**

User Credentials  
**ssh\_semaphore**

Sudo Credentials (Optional)  
**sudo\_semaphore**

Type  
**Static**

```
1 [clients]
2 192.168.1.78 #test-standard-1
3 192.168.1.91 #test-standard-2
4 192.168.1.115 #test-standard-3
```

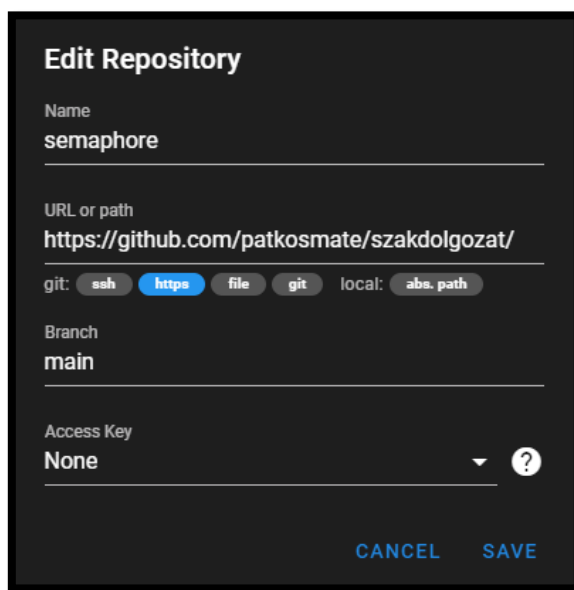
**Static inventory example:**  
[website]  
172.18.8.40  
172.18.8.41

**CANCEL SAVE**

8. ábra, Semaphore Inventory

Forrás: saját szerkesztés

A Repositories menüponton belül lehet megadni új repository-t, ahol a playbook-ok vannak tárolva. A repository használata azért fontos, mert így megvalósítható a csoportos munka az egyes projekteken, esetleges visszaállítás is lehetséges egy bizonyos playbook előző verziójára. Jelen esetben GitHub-ot<sup>8</sup> használtam, de ezt nagyvállalati környezetben saját GitLab-bal vagy Gitea-val kellene megoldani, mivel így a vállalat számára saját verziókezelő rendszer használata lehetséges (9. ábra). Ebben az esetben is meg kell adni egy nevet, hogy mi legyen a neve a repository-nak, valamint magát a GitHub, GitLab vagy Gitea webcímét, és azt, hogy mely branch-et használjuk. Mivel egyedül fejlesztettem a playbook-okat, ezért nem volt szükség külön branch-ek létrehozására, így egyből a main branch-be tudtam dolgozni. Ha a verziókezelő rendszerben lévő projektünk privát lenne, akkor egy hozzáférési kulcs elkészítése is szükséges, amit az Access Key mezőben kellene megadni. Számomra ez nem volt szükséges, ugyanis publikus projektet készítettem.



**Edit Repository**

Name  
semaphore

URL or path  
https://github.com/patkosmate/szakdolgozat/

git: ssh https file git local: abs. path

Branch  
main

Access Key  
None

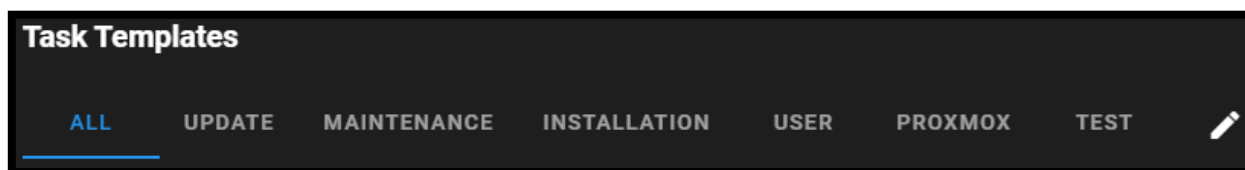
CANCEL SAVE

9. ábra, Semaphore Repository

*Forrás: saját szerkesztés*



A Task Templates menüpontban a New Template gombra kattintva lehet létrehozni a feladatokat (továbbiakban task). Ezeket a task-okat külön csoportokba lehet rendezni az átláthatóság végett (10. ábra), mint látható készítettem több csoportot is.



*10. ábra, Semaphore Task Templates*

*Forrás: saját szerkesztés*

Task létrehozásának folyamata a 11. ábra alapján:

- Minden task-nak kell adni egy nevet, amely kötelező mező.
- Lehet megadni egy leírást (ez a lépés opcionális).
- A playbook hozzáférési útjának megadása a Git repository-n belül.
- Inventory fájl definiálása, azaz mely inventory fájlban belüli gépekre vagy gépcsoportokra fusson le majd a playbook.
- Lehetőség van különböző környezeti változók használatára, nekem erre nem volt szükségem.
- Továbbá látható egy Vault Password mező. Ez az Ansible Vault funkcióhoz szükséges, amely majd az 5.4-es alfejezetben kerül tárgyalásra.
- Bizonyos nézetbe való helyezéshez a View mező legördülő menüjét lehet használni.
- A View mező alatt található a task ütemezett futtatásáért felelős rész, a cron. Ezt az 5.2-es alfejezetben belül kerül majd bemutatásra.
- Mint a képen látható, van lehetőség további parancssori argumentumok megadására, azonban ezt a részt üresen hagytam, mert nem volt rá szükségem.

### Edit Template

TASK

BUILD

DEPLOY

Name \*

update

Description

Playbook Filename \*

ansible/playbooks/update/update.yaml

Inventory \*

clients

Repository \*

semaphore

Environment \*

all

Vault Password

sudo\_semaphore

Survey Variables

+ Add variable

View

update

Cron

00 \*\*\*

I want to run a task by the cron only for new commits of some repository

Read the docs to learn more about Cron.

☐ Suppress success alerts

1 CLI Args (JSON array). Example: [ "-

☐ Allow CLI args in Task

CANCEL

SAVE

11. ábra, Semaphore Task

Forrás: saját szerkesztés

## 5. Ansible

### 5.1 Ansible bevezetés

Ahogy azt már korábban említettem a szakdolgozatomban elején (2. fejezet, Ansible alfejezetben), az Ansible egy ügynök nélküli automatizációs eszköz, ezért a telepítése nagyon egyszerű, mivel csak az Ansible szerverre kell telepíteni csomagot. Ezt a telepítési eljárást a hivatalos dokumentációt<sup>9</sup> követve tettem meg.

A playbook-ok YAML kiterjesztésű fájlokkal dolgoznak, amik egyszerűen írhatóak, olvashatóak, különösebb programozási tudást nem igényel, hogy képesek legyünk megírni egy playbook-ot. Minden egyes playbook futtatásánál az `ansible-playbook` parancsot kell majd használni.

Az Ansible inventory fájlokban találhatóak a célszervereink. Lehet több inventory fájlt is készíteni, vagy akár egy inventory fájlban több csoportot is létrehozni. Ez azért fontos, mert így külön lehet választani például egy teszt környezetet egy produktívtól. Az előbb említett `ansible-playbook` parancs kiegészül egy `-inventory` kapcsolóval, ami után az inventory fájlunkat tudjuk specifikálni, vagy a fájlban lévő csoportot. (12. ábra) Lentebb található egy példa, az automatikusan generált minta inventory fájlra, mint látható, a szögletes zárójeleken belül ([webservers]) lehet specifikálni egy pontos gépcsoportot. Ha egy adott gépcsoport specifikálását nem tesszük meg, továbbá az `-inventory` kapcsoló után `all`-t írunk, akkor a fájlban lévő összes gépre lefog futni a playbook.

```
# Ex 1: Ungrouped hosts, specify before any group headers:
## green.example.com
## blue.example.com
## 192.168.100.1
## 192.168.100.10

# Ex 2: A collection of hosts belonging to the 'webservers' group:
## [webservers]
## alpha.example.org
## beta.example.org
## 192.168.1.100
## 192.168.1.110
```

12. ábra, Ansible Inventory

*Forrás: saját szerkesztés*

## 5.2 Automatizált feladatok

Minden soron következő playbook tartalmazza a következő kódrészleteket, ezért átfogóan szeretnék írni arról, hogy melyik mire való. Az első sorban látható három kötőjel, amely alapvető formai követelménye a YAML formátumú kódoknak. Jelentésük nincs, azonban kötelező, elhanyagolhatatlan elem, minden playbook ezzel kell kezdődjön. A második sorban olvasható **hosts: all** kódrészlet azt mondja meg, hogy mely gépeken fusson le a playbook. Ezen érték azért **all**, mivel már egy dedikált inventory fájlból dolgozunk, ami a 8. ábrán látható. A harmadik sorban olvasható **become: yes** rész azért felelős, ha szükség lenne **sudo** felhasználóvá válnia az Ansible felhasználónak, akkor azt meg tudja tenni. A negyedik sorban látható **task:** résznél kezdődik a „valódi” playbook. Azt ezt követő **name:-**tal kezdődő sorok és az ez alá tartozó sorok mindegyike egy-egy feladat.

Első playbook, amit készítettem, az talán a legfontosabb mind közül, egy szoftverfrissítő playbook. Jelenlegi formájában (13. ábra) csak apt és dnf csomagkezelő rendszereken működik, mivel én csak Debian alapú szervereket készítettem (mivel ezek rendelkeznek a legnagyobb csomagkészlettel), de egy feltételes utasítással akár több csomagkezelő rendszerre is lehet használni. Mindezt meg lehet valósítani Windows rendszereken is. A Windows megoldásnál többször kell lefuttatni a frissítések keresése folyamatot, mivel némely frissítés feltételesen épül a másokra. Erre nem készítettem playbook-ot, mivel a Windows egy nem ingyenes operációs rendszer, emiatt igyekeztem a Linux rendszerekre koncentrálni.

```
1  ---
2  - hosts: all
3    become: yes
4    tasks:
5      - name: update packages with apt
6        when: ansible_pkg_mgr == 'apt'
7        apt:
8          update_cache: yes
9
10     - name: update packages with dnf
11       when: ansible_pkg_mgr == 'dnf'
12       dnf:
13         update_cache: yes
14
15     - name: upgrade packages with apt
16       when: ansible_pkg_mgr == 'apt'
17       apt:
18         state: present
19
20     - name: upgrade packages with dnf
21       when: ansible_pkg_mgr == 'dnf'
22       dnf:
23         state: present
```

13. ábra, Ansible Update

Forrás: saját szerkesztés

Kétféleképpen bontanám a 13. ábrán szereplő playbook-ot. Az első rész (5-13. sorok) az update-ért felelős, míg a második (15-23. sorok) az upgrade-ért. Az update\_cache: részlet valójában nem más, mint az apt update parancs, a state: present részlet pedig az apt upgrade parancsnak feleltethető meg. A when: ansible\_pkg\_mgr == 'apt', illetve 'dnf' sorok arra valók ebben a kódban, hogy meghatározzuk, hogy milyen csomagkezelő rendszert használunk az operációs rendszerünkön. Értelemszerűen, ha apt csomagkezelőt használó rendszerünk van, akkor az apt csomagkezelőre vonatkozó sorok fognak végrehajtódni, ha dnf csomagkezelőnk van, akkor pedig a dnf-re vonatkozó sorok fognak lefutni.

A soron következő két, nagyon egyszerű playbook, nagyon fontos szerepet játszanak egy automatizált környezetben. Egyidejűleg, egy vagy akár több gépet képesek újraindítani (14. ábra), kikapcsolni (15. ábra), melyek fölösleges plusz terhet rónak az üzemeltető csapatra, ha mindezt kézzel kellene végrehajtani, automatizáció nélkül. Ezen feladatok ellátásáért felelnek a következő playbook-ok.

```
1  ---
2  - hosts: all
3    become: yes
4    tasks:
5      - name: reboot machine
6        reboot:
7          reboot_timeout: 3600
```

14. ábra, Ansible Reboot

Forrás: saját szerkesztés

```
1  ---
2  - hosts: all
3    become: yes
4    tasks:
5      - name: shutdown machine
6        shutdown:
7          delay: 5
```

15. ábra, Ansible Shutdown

Forrás: saját szerkesztés

A következő playbook-ok elég egyértelműek, viszont kiegészíteném egy-egy gondolattal. A 14. ábrán látható a `reboot_timeout:3600` részlet, amely az újraindítás után eltelt maximális időt jelenti. Ez az a meghatározott idő, melyen belül válaszolnia kell a kliens gépnek. Ha nem válaszol a playbook hibára fut. A 15. ábrán látható playbook-nál a `delay: 5` részlet, egy 5 másodperces késleltetést jelent, mielőtt a számítógép kikapcsolása megkezdődik. Ezek az értékek általam kerültek beállításra, szükségyszerűen módosíthatóak.

A következő két fontos playbook-om, egy-egy monitoring megoldás volt. Ezt könnyen megoldhattam volna a Nagios segítségével - például Nagios SIGNAL4 plugin használatával -, viszont szerettem volna ezeket is Ansible-lel. A szóban forgó automatizációs szkriptek a rendszerek státuszát (16. ábra), és fizikai kihasználtságát monitorozzák (17. ábra). Ha valamelyik rendszer nem elérhető az Ansible számára, vagy a hardver kihasználtság kritikus, ami jelen esetben 90%, akkor arról e-mailben értesít (18. ábra). A státusz monitorozó szkript a Proxmox szerverhez csatlakozik, a playbook futtatásakor, és a rajta futó konténerek és virtuális gépek státuszát kéri le. A hardver erőforrás státuszát monitorozó playbook viszont közvetlenül a kliensekhez csatlakozik, és helyileg kéri le a hardver kihasználtságát. Mindkét megoldás a Gmail SMTP szerverét használja e-mail küldésre. Mindez nem járható egy nagyvállalati környezetben, ott saját levelezőrendszert kellene kiépíteni, aminek oka, a Gmail limitációi, például a napi 2000 darab küldött e-mail.

```
1  ---
2  - hosts: all
3    vars:
4      excluded_hosts:
5        - template-standard
6        - ubi-lxc
7      vars_files:
8        - /etc/ansible/secure_vars.yaml
9      tasks:
10       - name: gather container info
11         community.general.proxmox_vm_info:
12           api_host: 192.168.1.10
13           node: proxmox
14           api_user: "{{ proxmox_user }}"
15           api_password: "{{ proxmox_password }}"
16           register: all_container_info
17
18       - name: send mail if host is unreachable
19         when: item.status == 'stopped' and item.name not in excluded_hosts
20         community.general.mail:
21           host: smtp.gmail.com
22           port: 587
23           username: "{{ email_sender }}"
24           password: "{{ email_password }}"
25           to: "{{ email_receiver }}"
26           subject: host unreachable alert - {{ item.name }}
27           body: |
28             the host {{ item.name }} is unreachable, take action
29           loop: "{{ all_container_info.proxmox_vms | selectattr('status', 'eq', 'stopped') | list }}"
```

16. ábra, Ansible Status

*Forrás: saját szerkesztés*

A 16. ábrán látható státusz monitorozásra használt playbook-ban fellelhető első nagy különbség a harmadik sorban található. Ez pedig a változók bevezetése. Erre ebben az esetben azért van szükség, mert a playbook a Proxmox szerveren futó konténerek státuszát fogja lekérdezni, és mivel rendelkezünk olyan gépekkel (mint pl.: template-standard), aminek a státusza irreleváns jelen helyzetben, ezért ezt be kell rakni egy változóba, ami az én esetemben az `excluded_hosts`.

Az első feladatunk a 10. sorban kezdődik, ami a `gather container info` nevet viseli. Ebben a feladatban a playbook csatlakozik a Proxmox szerverhez, majd a 16. sorban látható módon regisztrálja a rajta futó konténerek információit. Ezeket az információkat `all_container_info` néven tároljuk.

A második feladat a 18. sorban kezdődik és a `send mail if host is unreachable` nevet kapta. Egyből egy `when:` utasítással kezdődik, ami az elágaztató utasítás megfelelője Ansible nyelven. Az `item.status` a jelenleg vizsgált konténer státuszára utal, ami jelen vizsgálatban a leállított, azaz `stopped`, az `item.name not in excluded_hosts` feltétel pedig azt vizsgálja meg, hogy az a konténer, ahova jelenleg csatlakozva van a playbook, nem tagja-e a playbook elején kitöltött `excluded_hosts` változónak. Ha a vizsgált gép státusza `stopped`, és nincs benne az `excluded_hosts` változóban, akkor belépünk a `when` utasítás törzsébe.

A törzsrészben történik az e-mail küldés. A `host` a Gmail SMTP szerverének a címe, a `port` értelemszerűen a használt – jelen esetben TLS – port, a `username` az e-mail-t küldő Gmail fiókja, a `password` az ehhez a fiókhoz tartozó jelszó, a `to` pedig az a Gmail fiók, ahova érkezik az e-mail. A `subject` résznél kell megadni, hogy mi szerepeljen az e-mail tárgyában, a `body`-hoz pedig az e-mail szövegét kell megadni.

Legvégül, a 29. sorban egy `loop` utasítást olvashatunk. Ez arra való, hogy a fent említett folyamat végrehajtódjon az összes konténerre. Erre az utasításra csak ennél a playbook-nál van szükség, mivel a `proxmox_vm_info` modul használatával másképp nem nyerhető vissza a Proxmox szerveren futó virtuális gépek információi.



Ezen playbook (17. ábra) csak egy részlete az általam használt playbook-énak, de egy részlet alapján is tudom szemléltetni a működését. Jelenleg szintén két feladattal dolgozunk, az első, ami a 6. sorban kezdődik a processzor kihasználtságát kéri le. Egy egyszerű parancssoros paranccsal lekéri a processzor kihasználtságot, majd ezt elmenti a `cpu_usage` változóba, százalékos formában. A második feladat, ami a 10. sorban kezdődik a státusz monitorozó megoldással nagyon azonos, egy dologban különböznek csak. Ez nem más, mint a feladat elején a feltételes utasítás. Jelen formájában azt ellenőrzi, hogy az előző feladatban kinyert `cpu_usage` változó értéke mennyi. Ha a processzor kihasználtság több, mint 90%, akkor belépünk a `when` utasítás törzsébe, és a definiált módszerrel e-mail kerül kiküldésre.

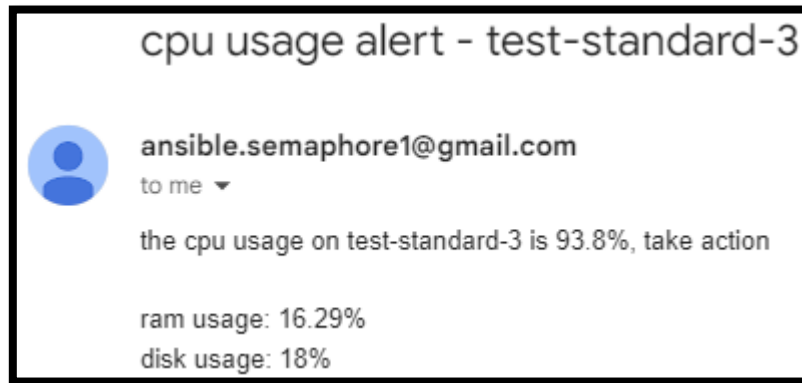
Az általam használt playbook nem csak a processzor terheltséget, hanem a memória, illetve merevlemez kihasználtságot is figyeli.

```
1  ---
2  - hosts: all
3    vars_files:
4      - /etc/ansible/secure_vars.yaml
5    tasks:
6      - name: gather cpu usage
7        shell: top -bn1 | grep 'Cpu(s)' | awk '{print $2}'
8        register: cpu_usage
9
10     - name: send mail of cpu if needed
11       when: cpu_usage.stdout[: -1] | int > 90
12       community.general.mail:
13         host: smtp.gmail.com
14         port: 587
15         username: "{{ email_sender }}"
16         password: "{{ email_password }}"
17         to: "{{ email_receiver }}"
18         subject: TEST cpu usage alert - {{ ansible_hostname }}
19         body: |
20           the cpu usage on {{ ansible_hostname }} is {{ cpu_usage.stdout }}%, take action
```

17. ábra, Ansible Resource

*Forrás: saját szerkesztés*

Az alábbi ábrán (18. ábra) egy példa e-mail látható, ami a test-standard-3 nevű gépről érkezett, ahol a processzor kihasználtság több, mint 90% volt.



18. ábra, Ansible E-Mail

*Forrás: saját szerkesztés*

Az eddig említett playbook-ok közül három - Update, Status, Resource - ütemezetten fut. A Semaphore beépített cron-ját használom, ami a Unix alapú rendszereknél szabályos időközönként futtat programokat. Ez jelen esetben előre telepített csomag volt. Jelenleg a rendszerfrissítő szkript minden nap éjfélkor, a monitorozásra használt szkriptek pedig ötpercenként. Ezeket a következő formátumban kell megadni: 0 0 \* \* \* és \*/5 \* \* \* \*. Nagyvállalati környezetben a rendszerfrissítési időközök az ügyféllel való megállapodáson alapulnak. Ez az időintervallum általában egy hónap, de lehet akár negyedéves rendszerességű is, minden esetben az irodai munkaórákon kívül kell történnie.

A soron következő folyamat szintén elengedhetetlen része egy automatizált környezetnek, és ez az alkalmazástelepítés. Erre is sikeresen készítettem egy playbook-ot, ami nagyon hasonló a rendszerfrissítő szkripthez, mivel ez is az apt csomagkezelőt használja. Jelen példában a Vim-et fogja telepíteni (19. ábra). Ezt szándékosan úgy készítettem el, hogy ne lehessen a grafikus felületen vagy közvetlenül az Ansible szerveren módosítani, hogy egy nem hozzáértő ne telepítsen semmilyen csomagot. A telepítendő csomag neve csak úgy módosítható, hogy előzőleg a playbook-ban át kell írni a csomag nevét, és a verziókezelő rendszerbe feltölteni.

A 7. sorban található `update_cache: yes` a már említett módon az `apt update` parancsot reprezentálja Ansible nyelven, a `name: vim` adja meg azt, hogy a `vim` nevű csomagot szeretnénk telepíteni, majd a `state: present` pedig azt, hogy egy stabil verziójú csomag kerüljön telepítésre.

```
1  ---
2  - hosts: all
3    become: yes
4    tasks:
5      - name: install vim package
6        apt:
7          update_cache: yes
8          name: vim
9          state: present
```

19. ábra, Ansible Vim

*Forrás: saját szerkesztés*

A következő szkript arra szolgál, hogy egy playbook indítással, akár az összes kliens gépéhez csatlakozva, ki lehessen adni akárhány sorból álló parancsot (20. ábra). Mindezt úgy valósítottam meg, hogy mielőtt futtatni szeretnénk a playbook-ot, azelőtt át kell szerkeszteni manuálisan. Erre azért van szükség, mert egy ilyesfajta playbook-ot nem szabad bármely felhasználónak futtatnia, mert akár egy paranccsal hatalmas károkat lehet okozni több gépen is.

```
1  ---
2  - hosts: all
3    become: true
4    tasks:
5      - name: install ansible
6        shell: |
7          sudo apt update &&
8          sudo apt install software-properties-common -y &&
9          sudo add-apt-repository --yes --update ppa:ansible/ansible &&
10         sudo apt install ansible -y
```

20. ábra, Ansible Command

*Forrás: saját szerkesztés*

Szintén hasonló elképzelésen alapul a következő két playbook-om, amivel felhasználókat lehet létrehozni a kliens gépeken (21. ábra). A szkript futtatását megelőzően módosítani kell a playbook-ot, mivel meg kell határozni a felhasználó nevét, jelszavát. Ha `sudo` felhasználót szeretnénk létrehozni, akkor a létrehozott felhasználót az `/etc/sudoers` fájlba bele kell írni. A `sudo` egy olyan alkalmazás, ami lehetővé teszi egy bizonyos felhasználó számára, hogy egy másik felhasználó – jelen helyzetben `root` – jogosultságaival futtasson programokat.

A nagy különbség a felhasználó és `sudo` felhasználó létrehozása között, a már előbb említett `sudoers` fájlba való hozzáadás. Jelen példán keresztül fogom szemléltetni mindkét módszert. A felhasználó létrehozásához csak egy feladatra van szükség, ami a 13. sorban véget ér, viszont, ha `sudo` felhasználót szeretnénk létrehozni, akkor az egész képen látható playbook-ra szükség van. Az 5. és 6. sorban kell előre definiálni, hogy a felhasználónak mi legyen a neve, illetve a jelszava. A 8. sorban kezdődik az első feladat, a `create a new user` névvel. Itt nem történik más, mint az 5. és 6. sorban definiált információkat használva létrehozzuk a felhasználót. A jelszót sha512-es titkosítás használatával a playbook futási kimeneteléből „elrejtjük”. A 15. sorban kezdődik a második feladat, amiben a definiált felhasználót hozzáadjuk a `sudoers` fájlhoz.

```
1  ---
2  - hosts: all
3    become: yes
4    vars:
5      username: test-sudo #username
6      password: test-sudo #password
7    tasks:
8      - name: create a new user
9        user:
10          name: "{{ username }}"
11          password: "{{ password | password_hash('sha512') }}"
12          state: present
13          become: yes
14
15      - name: add the user to the sudoers file
16        lineinfile:
17          path: /etc/sudoers
18          line: "{{ username }} ALL=(ALL) NOPASSWD: ALL"
19          validate: 'visudo -cf %s'
20          become: yes
```

21. ábra, Ansible User

*Forrás: saját szerkesztés*

Mint már említettem, az NRPE csomagot előzőleg telepítettem a template-re, amiből készítettem a kliens szervereket. Viszont ettől még nem lesz működőképes a monitoring megoldás. Szükséges egy konfigurációs fájl készítése minden egyes kliens gép információjával (név és IP cím szükséges). Ezen konfigurációs fájlt be kell másolni a Nagios szerver /usr/local/nagios/etc/servers/ könyvtárába. Minden konfigurációs fájl nevének meg kell egyeznie a kliens gép nevével, a fájl kiterjesztése .cfg kell, hogy legyen. Majd ezután újra kell indítani a Nagios szerveren a monitoring service-t. Ez a playbook okozta a legnagyobb kihívást.

Az LXC konténer létrehozást úgy valósítottam meg, hogy a fent említett template-et leklónozzom minden egyes gép létrehozásánál. Ehhez az Ansible community.general.proxmox plugin-ját használom. A playbook futtatásával csatlakozik a Proxmox szerverhez, amin az 1000-es azonosítóval rendelkező template-et használva hoz létre egy előre definiált azonosítójú és nevű konténert. Ezt követően már csak el kell indítani az újonnan készített konténert. (22. ábra) Erre az elindításra szolgál a második feladat, aminek a végén a **state: started** kódrészlet olvasható.

```
1  ---
2  - hosts: all
3    vars_files:
4      - secure_vars.yml
5    tasks:
6      - name: clone standard lxc container
7        community.general.proxmox:
8          api_host: "{{ host }}"
9          node: "{{ node }}"
10         api_user: "{{ user }}"
11         api_password: "{{ password }}"
12         clone: 1000
13         storage: VM-Drives
14         vmid: 111 #if specific vmid if needed
15         hostname: test-ansible #modify to desired hostname
16
17      - name: start standard lxc container
18        community.general.proxmox:
19          api_host: "{{ host }}"
20          node: "{{ node }}"
21          api_user: "{{ user }}"
22          api_password: "{{ password }}"
23          vmid: 111 #if specific vmid if needed
24          hostname: test-ansible #modify to desired hostname
25          state: started
```

22. ábra, Deploy

*Forrás: saját szerkesztés*

### 5.3 Egyéb, fel nem használt feladatok

Több playbook-ot is készítettem a szakdolgozati munkám során, azonban ezek közül kettő nem került felhasználásra, mert nem találtam relevánsnak a szakdolgozatom témájához. Habár nem relevánsak, azonban érdemi információval szolgáltak, ezért pár szóban összefoglalnám jelentőségüket:

1. Az első szolgált a kezdetleges monitoring megoldásomnak.
  2. A második pedig arra szolgált, hogy mások is el tudják érni a fizikai szervert, amin fut a Proxmox virtualizációs környezet.
- 
1. Monitoring megoldás kivitelezése: A monitorozásra használt playbook egy Discord üzenetet küldött eleinte a privát Discord szerveremre. Ezt a beépített Discord Webhook integrációs megoldással valósítottam meg. A szkript szintúgy ötpercenként futott, mint a jelenleg is használt monitorozó megoldás, annyi különbséggel, hogy ez nem e-mail-t küld, hanem egy üzenetet. Egy otthoni környezetben ezt a megoldást választanám, mert kikerülhető vele a Gmail SMTP szerverének használata, ami csak bonyolít a helyzeten.
  2. A másik fel nem használt funkcióra azért volt szükség, mert abban az időben, amikor a környezetet építettem ki, egy barátomnak szüksége volt egy környezetre, ahol a szabadidejében a munkájához tudott készíteni projekteket, ezért készült el egy publikus IP cím ütemezett küldésére használható szkript. A playbook futtatásakor csatlakozott a Proxmox szerverhez, lekérte az IP címét, és email-ben továbbította az előre definiált email címre. A publikus IP címet ezzel a paranccsal tudtam visszafejteni: `dig +short myip.opendns.com @resolver1.opendns.com`. Viszont ez a megoldás teljesen nem elfogadott egy nagyvállalati környezetben, mivel a privát információk kiközlése titkosítás nélkül bármilyen formában nem engedélyezett.

## 5.4 Ansible Vault

A fent említett státusz és fizikai kihasználtság monitorozására használt szkriptek (16. és 17. ábra) harmadik és negyedik sorában található a következő kódrészlet:

```
var_files:  
  
  - /etc/ansible/secure_vars.yaml
```

Ez az Ansible Vault<sup>10</sup> funkció. Arra szolgál, hogy az Ansible szerverünkön lokálisan tárolhatunk egy titkosított `secure_vars` fájlt. Biztonságosan tudunk tárolni olyan információkat, amelyeket titkosítani kell, viszont a playbook-oknak hozzáférést lehet ehhez adni. Az `ansible-vault create` paranccsal lehet létrehozni egy titkos fájlt, ami jelszóval védett. Ha szerkeszteni vagy olvasni szeretnénk ezt a fájlt, akkor azt az `ansible-vault edit`, illetve `ansible-vault view` paranccsal tudjuk megtenni, miután megadtuk a hozzá tartozó jelszót. A 16. és 17. ábrán látható `proxmox_user`, `proxmox_password`, `email_sender`, `email_password`, `email_receiver` változók mind ebben a fájlban találhatóak.

## 6. További fejlesztési lehetőségek

További eszközök hiányában kimaradt több fontos funkció is, hogy a szakdolgozatom „nagyvállalati környezetben” részét jobban lefedje, így az alábbi fejlesztési lehetőségekkel lehet kiegészíteni az általam bemutatott környezetet:

6.1. DNS (Domain Name System)

6.2. SQL (Structured Query Language)

6.3. Telefonhívás monitoring jelzés esetén

### 6.1 DNS (Domain Name System)

A DNS-szerverek kulcsfontosságú szerepet töltenek be a nagyvállalati hálózatok hatékony működésében. A DNS-szerverek legfontosabb előnyei:

- Gyorsítják a hálózati forgalmat azáltal, hogy a domain neveket IP-címekké fordítják.
- Ha több DNS szerverrel dolgozunk, akkor redundánsan tudnak működni, ami azt jelenti, hogy ha az egyik kiszolgáló meghibásodik, akkor a másik át tudja venni a feladatát.
- Továbbá nagyban hozzájárulnak a hálózati biztonsághoz a tűzfalak és más biztonsági eszközök integrálásával.

A DNS-szerverek ezen előnyei révén a nagyvállalatok hatékonyabban és megbízhatóbban üzemeltethetik hálózataikat, miközben javítják a hálózati adminisztráció hatékonyságát és csökkentik a költségeket.

### 6.2 SQL (Structured Query Language)

Továbbá, szerettem volna készíteni több SQL adatbázis szervert is. Ehhez a playbook elkészült, ami az LXC konténer létrehozásához írthoz nagyon hasonló, viszont miután létrehoztunk egy adott gépet, utána lefuttat egy parancssori szkript fájlt. Ez a szkript fájl a felelős:

- A szükséges csomagok telepítéséért.
- Az SQL felhasználó létrehozásáért.
- Egy minta adatbázis készítéséért.



Ehhez high availability cluster-t (HA cluster) használtam volna. A, HA cluster számítógépek olyan csoportja, amelyek egyetlen rendszerként működnek és folyamatos elérhetőséget biztosítanak. Az üzletileg kritikus alkalmazásoknál, például adatbázisoknál használják őket a megbízhatóság fenntartása érdekében. Egy, HA cluster működése során a klaszterszoftver a kieső gép terhelését a többi működő gépre helyezi át, így minimalizálja a kiesést.

### 6.3 Telefonhívás monitoring jelzés esetén

Az e-mail küldés mellett telefonhívást is állíthattam volna be, ezáltal gyorsabban lehet reagálni egy-egy monitoring jelzésre, viszont ennek nem láttam szükségességét, mivel jelen projekt csak egy szemléltetés. Ennek a módszernek több megoldása is lehetséges applikáció használatával, mint például a Nagios SIGNAL4 plugin-ja, vagy akár a szkript implementálásával való megoldás. Egyéb lehetőség a Twilio API megoldása, ami akár PowerShell-be is integrálható.

## 7. Összegzés

A szakdolgozatom elkészítése során sokat tanultam a virtualizációról. A virtualizáció egy nagyon fontos témakör az IT szférában, mivel a klasszikus fizikai szerverek üzemeltetése kihalóban van, aminek a helyét a virtualizáció és felhőtechnológiák használata veszi át.

Továbbá az Ansible automatizációs platform használatáról is sokat tanultam, hogy azt, hogy s miként lehet a legeredményesebb módon használni. Az általam bemutatott playbook-ok egytől-egyig nagyon fontos részét képezik egy automatizált környezetnek, melyek iránymutatásként szolgálnak egy ilyesfajta környezet megvalósításához. A megvalósítható playbook-ok listája végtelen, szinte bármilyen elképzelés megoldható az Ansible automatizáció segítségével.

Az elkészült környezetet továbbá is fogom használni, itthoni környezet gyanánt. A jelenleg használt e-mail küldő monitoring megoldást lecserélem az 5.3-as bekezdés 1. pontjában bemutatott Discord üzenetküldő megoldásra, mivel erre a jelzésre sokkal egyszerűbb reagálnom. Tovább fogom fejleszteni a környezetemet egyéb playbook-okkal, és nagyon hasznos lesz a munkám végzése során, mivel így egy teljes tesztkörnyezetre tettem szert, ahol folytathatom a szkriptek fejlesztését.

## 8. Köszönetnyilvánítás

Szeretném megragadni az alkalmat arra, hogy köszönetet mondjak azoknak, akik lehetővé tették a szakdolgozatom elkészülését.

Először is Dr. Krausz Tamás adjunktusnak, hogy elvállalta a szakdolgozatom témavezetését és hogy segítségemre volt az elkészítésében, bármilyen típusú problémám adódott.

Szeretném megköszönni a családomnak, páromnak és barátaimnak a mérhetetlen támogatást az egyetemi éveim és a szakdolgozatom elkészülésének ideje alatt.

## 9. Irodalomjegyzék

<sup>1</sup> Ansible hivatalos dokumentáció.

<https://docs.ansible.com/>

(Utolsó megtekintés ideje: 2024. 03. 27.)

<sup>2</sup> Proxmox VE hivatalos dokumentáció.

<https://pve.proxmox.com/pve-docs/pve-admin-guide.html>

(Utolsó megtekintés ideje: 2024. 03. 27.)

<sup>3</sup> Semaphore hivatalos dokumentáció.

<https://docs.semui.co/>

(Utolsó megtekintés ideje: 2024. 03. 27.)

<sup>4</sup> Nagios Core hivatalos dokumentáció.

<https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/toc.html>

(Utolsó megtekintés ideje: 2024. 03. 27.)

<sup>5</sup> Proxmox VE hivatalos telepítési dokumentáció.

<https://pve.proxmox.com/pve-docs/chapter-pve-installation.html>

(Utolsó megtekintés ideje: 2024. 03. 27.)

<sup>6</sup> NRPE - Nagios Remote Plugin Executor

<https://exchange.nagios.org/directory/image/93>

(Utolsó megtekintés ideje: 2024. 04. 04.)

<sup>7</sup> Semaphore hivatalos telepítési dokumentáció.

<https://docs.semui.co/administration-guide/installation>

(Utolsó megtekintés ideje: 2024. 03. 27.)

<sup>8</sup> A szakdolgozatom GitHub linkje.

<https://github.com/patkosmate/szakdolgozat>

*(Feltöltési ideje: 2024. 01. 23.)*

<sup>9</sup> Ansible hivatalos telepítési dokumentáció.

[https://docs.ansible.com/ansible/latest/installation\\_guide/intro\\_installation.html](https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html)

*(Utolsó megtekintés ideje: 2024. 03. 27.)*

<sup>10</sup> Ansible Vault hivatalos dokumentáció.

[https://docs.ansible.com/ansible/latest/vault\\_guide/index.html](https://docs.ansible.com/ansible/latest/vault_guide/index.html)

*(Utolsó megtekintés ideje: 2024. 03. 27.)*