**SZAKDOLGOZAT**

PATKÓS MÁTÉ LEVENTE

DEBRECEN

2024

Debreceni Egyetem

Informatikai Kar

Információ Technológia Tanszék

Autómatizáció nagyvállalati környezetben

Automatizáció Ansible használatával

Témavezető: Szerző:

Dr. Krausz Tamás Patkós Máté Levente

*Egyetemi Adjunktus Mérnökinformatikus BSc*

Tartalom

[1.Bevezetés 4](#_Toc161940644)

[2. Felhasznált eszközök 5](#_Toc161940645)

[2.1 Ansible 5](#_Toc161940646)

[2.2 Proxmox 6](#_Toc161940647)

[2.3 Semaphore 6](#_Toc161940648)

[2.4 Nagios Core 6](#_Toc161940649)

[3. Környezet kiépítése 7](#_Toc161940650)

[4. Semaphore 11](#_Toc161940651)

[4.1 Semaphore bevezetés, működés 11](#_Toc161940652)

[4.2 Semaphore konfiguráció 11](#_Toc161940653)

[5. Ansible 15](#_Toc161940654)

[5.1 Ansible bevezetés 15](#_Toc161940655)

[5.2 Automatizált feladatok 16](#_Toc161940656)

[5.3 Egyéb, fel nem használt feladatok 25](#_Toc161940657)

[5.4 Ansible Vault 25](#_Toc161940658)

[6. További fejlesztési lehetőségek 26](#_Toc161940659)

[6.1 DNS (Domain Name System) 26](#_Toc161940660)

[6.2 SQL (Structured Query Language) 26](#_Toc161940661)

[6.3 Telefonhívás monitoring jelzés esetén 26](#_Toc161940662)

1. Bevezetés

2. Felhasznált eszközök

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

- Ansible automatizációs eszköz

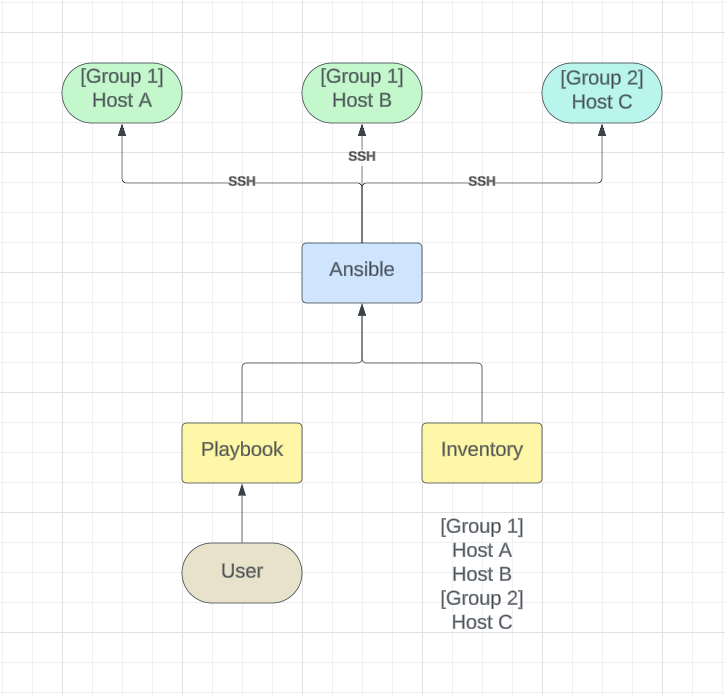
- Proxmox virtualizációs környezet (PVE)

- Semaphore kezelői felület Ansible-höz

- Nagios Core monitoring

2.1 Ansible

Az Ansible egy nyílt forráskódú, ügynök (jövőben agent) nélküli konfigurációkezelő és automatizálási eszköz, amelyet nagyszámú szerver hatékony üzemeltetésére és konfigurálására használhatunk. Kis és nagyvállalkozások egyaránt használnak. 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, pl. alkalmazástelepítésre is használható. (1. ábra) 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*

2.2 Proxmox

A Proxmox VE 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 virtualizáció virtuális gépek futtatása, LXC konténerek futtatása erőforrás-hatékonyabb izolációval, tárolókezelés virtuális lemezek és hálózati adattárolók számára, hálózatkezelés virtuális switch-ek és VLAN-ok használatával.

2.3 Semaphore

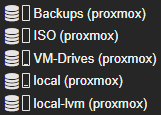
A Semaphore egy nyílt forráskódú webes kezelői felület az Ansible-höz. Ezek keresztül tudunk futtatni Ansible playbook-okat. A Semaphore pure Go nyelven íródott. Támogat MySQL-t, PostgreSQL-t és BoltDB-t is. Ezzel a kezelői felülettel lehet készíteni playbook csoportokat a biztonság miatt, akár egyes felhasználóknak adhatunk jogot bizonyos playbookok futtatásához. Képes a leltárat (jövőben inventory), a gy[ű](https://scriptsource.org/cms/scripts/page.php?item_id=character_detail&key=U000171)jteményt (jövőben repository) és a hozzáférési kulcsokat menedzselni. Továbbá képes a playbook-ok időzített futtatására is, de akár vissza is lehet nézni az egyes playbook-ok futási kimenetelét.

2.4 Nagios Core

 A Nagios Core 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. A Nagios Core előnyei: Szinte bármely komponenst képes monitorozni, legyen az hardver vagy szoftver, akár több száz vagy ezer eszköz egyidejű monitorozására is alkalmas, automatikusan képes riasztás küldeni e-mailben, telefonon vagy más csatornán keresztül. (2. ábra)

*2. ábra, Nagios*

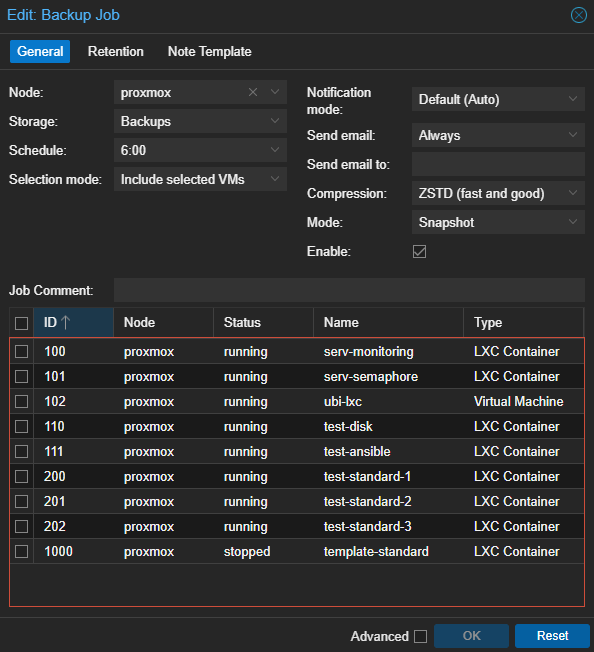
3. Környezet kiépítése

A fizikai szerver 128GB-os SSD-jére telepítettem a Proxmox VE-t, ami nagyon egyszerű volt, a remek dokumentációnak hála. A 2db 1TB-os HDD-ből készítettem egy RAID 1 ZFS Pool-t, így, ha bármi történik az egyik merevelemezzel, megmarad minden adat a másikon. Ide készítettem több mappát, hogy átlátható legyen a környezet. (3. ábra)

*3. ábra, Proxmox ZFS*

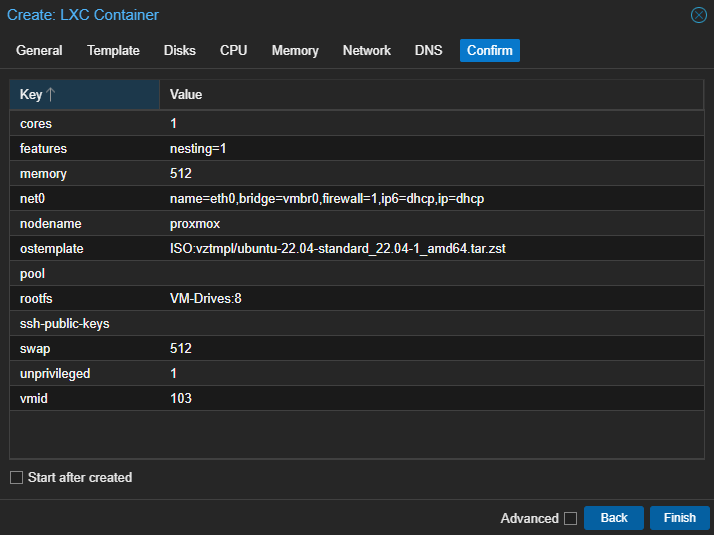
A ’Backups’ a biztonsági mentéseknek szolgál, az ’ISO’ mappába kerültek az LXC konténer és ISO képfájlok, a ’VMDrives’ pedig a virtuális gépek, konténerek lemezeinek ad helyet. A biztonsági mentések készítéséhez a Proxmox beépített backup megoldását használtam. (4. ábra)

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 használnám, a Proxmox Backup Server-t. Ez a megoldás szintén nyílt forráskódú, ami megállja a helyét egy nagyvállalati környezetben. Említésre méltó, ellenben nem nyílt forráskódú egyéb megoldások: IBM Tivoli Storage Manager, Dell Avamar.

*4. ábra, Proxmox Backup*

A második ’Retention’ fülnél lehet beállítani, hogy meddig tartsa meg a biztonsági mentéseket, jelenleg nekem az utolsó három mentést tartja meg. Ezen felül van lehetőség óránkénti, napi, heti, havi vagy akár éves bizonyos számú biztonsági mentég megtartására.

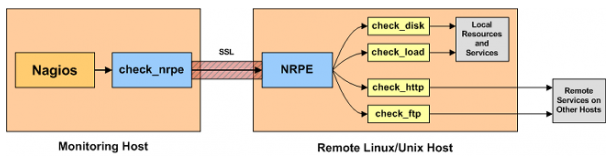
Készítettem néhány LXC konténert egy Ubuntu konténer fájlból. A gépek létrehozásánál kell megadni egy root felhasználóhoz tartozó jelszót, ha van SSH publikus kulcsot (én használtam), hogy melyik képfájlből/konténer fájlból készítse a virtuális gépet, lemez méretet, CPU számot, RAM mennyiséget, és van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

*5. ábra, Proxmox LXC*

Az SSH nem más, mint egy titkosított kapcsolati protokoll, amely biztoságos 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 kapcsolti beállításaiban kell megadni – én a MobaXterm-et használtam ehhez.

Miután elkészült 2 virtuális gép, - az egyik a Nagios Core monitoring szerver, a másik az Ansible/Semaphore szerver - elkezdtem a szükséges csomagok telepítését. Mindkét szerver esetében a hivatalos dokumentációt követve haladtam.

Készítettem egy minta-gépet (jövőben 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 tud csatlakozni a Nagios szerver a kliens gépekhez, és különbőzó plug-in-okat használva le tudja kérni a hardver, illetve szoftverek állapotát.

*7. ábra, Nagios NRPE*

4. Semaphore

4.1 Semaphore bevezetés, működés

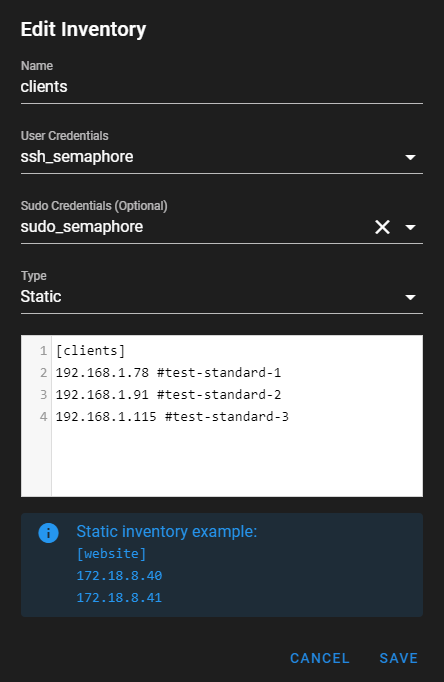
A Semaphore egy olyan koordinációs eszköz, amely megakadályozza, hogy az Ansible playbook-ok futtatása során ne történjen ütközés. Ez akkor hasznos, ha egy gépen vagy gépcsoporton több playbook is fut. Minden szkript indítással addig vár a Semaphore, amíg az előző be nem fejeződik, ezáltal sokkal kisebb a hibafaktor. Hátrányt jelent viszont, hogy ezáltal lassabb a playbook-ot 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.

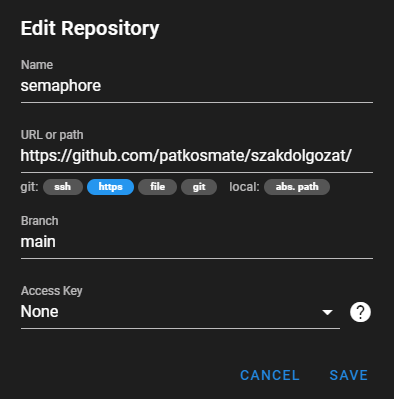
4.2 Semaphore konfiguráció

A csomagok telepítése nem okoz gondot, a dokumentációt követve, ezért ugranék is a konfigurációs részhez. A Semaphore web interfészen bejelentkezve, készíteni kell egy projektet. A projekt létrehozása után a legfontosabb lépések a következőek: 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 ’Key Store’ fülre kattintva lehet megadni az előzőleg generált SSH privát kulcsot. Lehet megadni ezen felül egy felhasználónevet is, ha valamiért nem sikerülne az SSH csatlakozás, azt a felhasználót használva tud csatlakozni a gépekhez. Ugyanebben a menüpontban szükséges megadni még egy ’sudo’ felhasználó-jelszó párost, ha lenne olyan playbook-unk, ami emelt jogusultságokkal való futtatást igényelne.

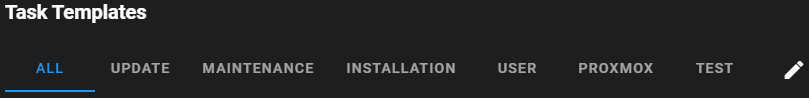
Az ’Inventory’ fülre kattintva lehet létrehozni inventory fájlokat. Minden inventory fájlnak kell adni egy nevet, azt, hogy mely felhasználónév-jelszó párost használja az inventory fájlokban lévő kliens gépekhez való csatlakozásra, továbbá, ha szükséges, sudo jogosultsággal rendelkező felhasználónév-jelszó 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).

*8. ábra, Semaphore Inventory*

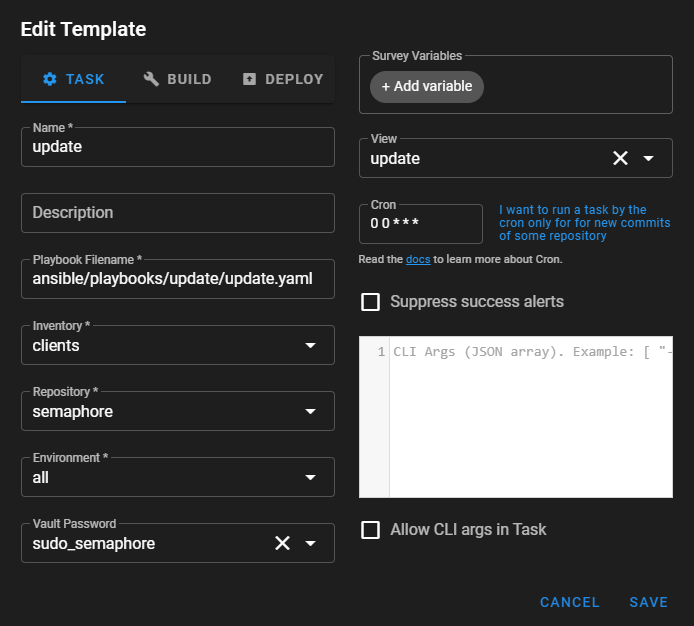
A ’Repositories’ fülön belül lehet megadni új repository-t, ahol a playbook-ok vannak tárolva. Én jelen esetleg egy GitHub-ot használtam, de ezt nagyvállalati környezetben saját GitLab-bal kellene megoldani (9. ábra). Ehhez is tartozik egy név, hogy mi legyen a neve a repository-nak, magát egy GitHub vagy GitLab webcímet, és azt, hogy mely branch-et használjuk. Mivel én egyedül dolgoztam a szakdolgozaton, ezért nem volt szükség külön branch-ek létrehozására, így egyből a main branch-be dolgoztam. Ha a verziókezelő rendszerben lévő projektünk privát lenne, akkor készíteni kellene egy hozzáférési kulcsot, amit az Access Key mezőben kellene megadni. Én **publikus projektet használtam.

*9. ábra, Semaphore Repository*

A ’Task Templates’ fülnél a ’New Template’ gombra kattintva lehet létrehozni a feladatokat (jövőben task). Ezeket a task-okat lehet külön csoportokba rendezni az átláthatóság végett (10. ábra). Mint látható én készítettem több csoportot is.

*10. ábra, Semaphore Task Templates*

Az alábbi (11. ábra) ábra alapján mutatom be egy task létrehozását. Minden task-nak kell adni egy nevet, lehet megadni egy leírást (ez a lépés opcionális), a playbook hozzáférési útját a Git-en belül, azt, hogy mely inventory fájlon belüli gépekre vagy gépcsoportokra fusson le majd a playbook és hogy melyik Git repository-ból dolgozzon. Ezen felül van lehetőség különböző környezeti változók használatára, nekem erre nem volt szükségem, és látható egy Vault Password mező. Ez az Ansible Vault funkcióhoz szükséges, amit majd a 5.4-es részben fogok tárgyalni. Bizonyos nézetbe való helyezéshez a View mező legördülő menüjét lehet használni, majd ezalatt található a task ütemezett futattásáért felelős rész, a cron. Ezt az 5.2-es menüponton belül, a 18. oldal alján taglalom.

*11. ábra, Semaphore Task*

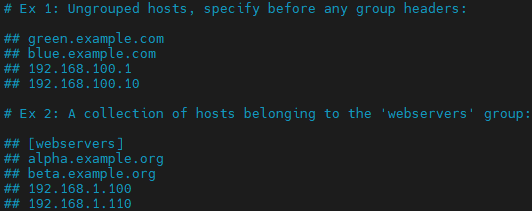
5. Ansible

5.1 Ansible bevezetés

Mint már említettem a szakdolgozatom elején (2. fejezet, Ansible alcím), 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 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 futtatásnál 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. Az előbb említett ’ansible-playbook’ parancs kiegészül egy ’-inventory’ kapcsolóval, ami után az inventory fájlunkat tudjuk specifikálni, vagy a fájlban lévő csoportot. (12. ábra) Alább egy példa, az automatikusan generált minta inventory fájlra. Mint látható, a szögletes zárójeleken belül lehet specifikálni egy pontos gépcsoportot. Ha ezt nem tesszük meg, és az -inventory kapcsoló után all-t írunk, akkor a fájlban lévő összes gépre lefut a playbook.

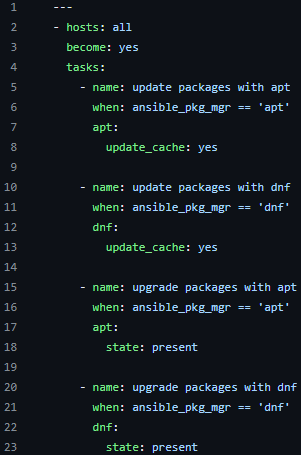
*12. ábra, Ansible Inventory*

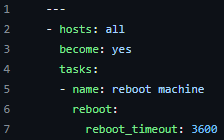
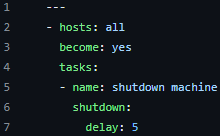
5.2 Automatizált feladatok

Minden soron következő playbook tartalmazza a következő kódrészleteket, ezért átfogóan szeretnék írni arról, hogy melyik mire való. Az első sorban látható három kötőjel egy formai követelménye a YAML formátumú kódoknak. Nem jelentenek semmit, mégis 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.

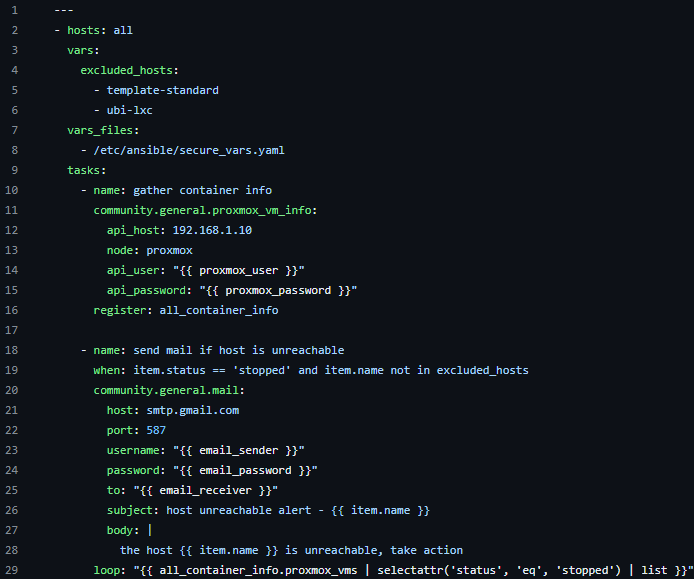
Első playbook, amit készítettem, az talán a legfontosabb mind közül, egy szoftverfrissítő playbook. Jelenlegi formájában (13. ábra) csak apt és dnf csomagkezelő rendszereken működik, mivel én csak Debian alapú szervereket készítettem, de egy feltételes utasítással akár több csomagkezelő rendszerre is lehet használni. Mindezt meglehet valósítani Windows rendszereken is. A Windows megoldásnál többször kell lefuttatni a frissítések keresése folyamatot, mivel némely frissítés feltételesen épül a másikra.

Kétrészre bontanám az alábbi képen szereplő playbook-ot. Az első rész az update-ért felelős, a második 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: asible\_pkg\_mgr == ’apt’ illetve ’dnf’ sorok arra valóak ebben a kódban, hogy meghatározzuk, hogy milyen csomagkezelő rendszert használunk a csomagkezelő rendszerünk. 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.

*13. ábra, Ansible Update*

 A soron következő két, nagyon egyszerű playbook, nagyon fontos szerepet játszanak egy automatizált környezetben. Egyidejűleg egy vagy akár több gépet újraindítani (14. ábra), kikapcsolni (15. ábra) fölösleges plusz teher az üzemeltető csapatnak, ha ezt mindet kézzel kell végrehajtani. Ezt a feladatok látják el a következő playbook-ok. A következő playbook-ok elég egyértelműek, viszont egy-egy dolgot hozzátennék. A reboot\_timeout: 3600 részlet az újraindítás után eltelt maximum időt jelenti, ami időn belül válaszolnia kell a kliens gépnek. Ha nem válaszol a playbook hibára fut. A 10. ábrán látható playbook-nal a delay: 5 részlet, egy 5 másodperces késleltetést jelent, mielőtt a számítógép kikapcsolása megkezdődik.

*14. ábra, Ansible Reboot 15. ábra, Ansible Shutdown*

 A következő két fontos playbook-om, egy-egy monitoring megoldás volt. Ezt könnyen megoldhattam volna a Nagios segítségével, 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.

*16. ábra, Ansible Status*

A 11. ábrán látható státusz monitrozásra használ 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.

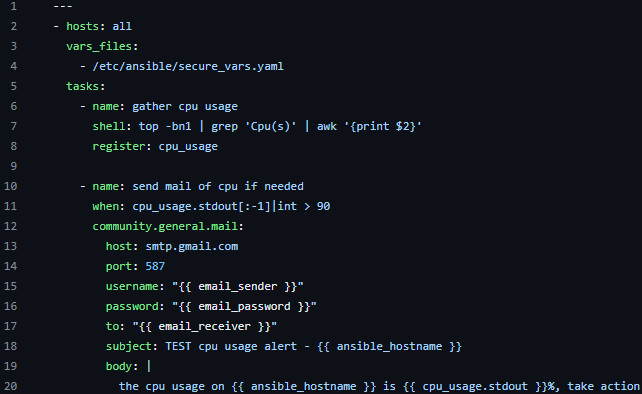
A második feladat a 18. sorban kezdődik és a send mail if host is unreachable nevet kapta. Egyből egy when: utasítással kezdődik, ami az elágaztató utasítás megfelelője Ansible nyelven. Az item.status a jelenleg vizsgált konténer státuszára utal, ami jelen vizsgálatban a leállított, azaz stopped, az 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.

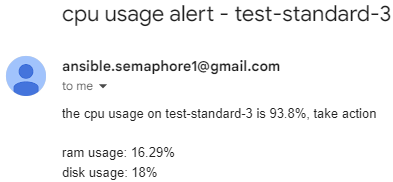
Legvégül, a 29. sorban egy loop utasítást olvashatunk. Ez arra való, hogy a fent említett folyamat végrehajtódjon az összes konténerre. Erre az utasításra csak ennél a playbook-nál van szükség, mivel a proxmox\_vm\_info modul használatával másképp nem nyerhető vissza a Proxmox szerveren futó virtuális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

Ezen playbook (17. ábra) csak egy részlete az általam használt playbook-énak, de feleslegesnek találtam az egészről készült képet beszúrni. Jelenleg szintén két feladattal dolgozunk, az első, ami a 6. sorban kezdődik a processzor kihasználtságát kéri le. Egy egyszerű shell 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. A memóriakihasználtsághoz figyeléséhez ezt a parancsot: top -bn1 | awk '/MiB Mem/ {printf "%.2f\n", $8/$4 \* 100}' míg a merevlemez terheltséghez ezt a parancsot használom: "df -h --output=pcent / | awk 'NR==2 {print $1}'"

*17. ábra, Ansible Resource*

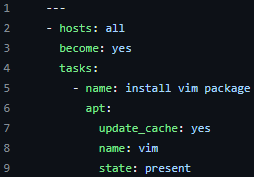
Az alábbi ábrán (18. ábra) egy példa e-mail látható, ami a test-standard-3 nevű gépről érkezett, ahol a processzor kihasználtság több, mint 90% volt.

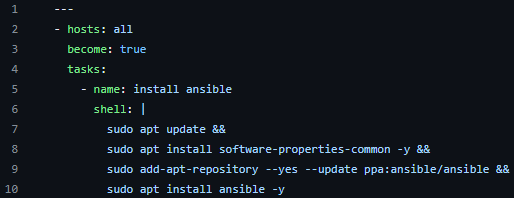
*18. ábra, Ansible E-Mail*

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. Jelenleg a rendszerfrissítő szkript minden nap éjfélkor, a monitorozásra használt szkriptek pedig ötpercenként. Ezeket a következő formátumban kell megadni: 0 0 \* \* \* és \*/5 \* \* \* \*.

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 telepítsem 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.

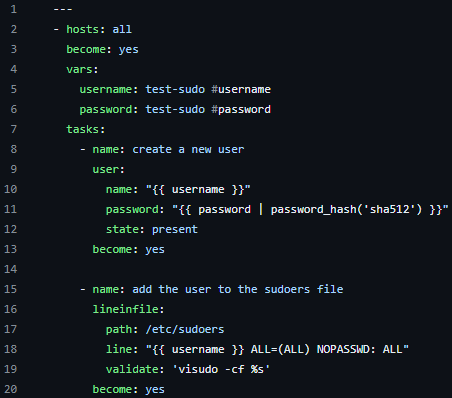
*19. ábra, Ansible Vim*

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

*20. ábra, Ansible Command*

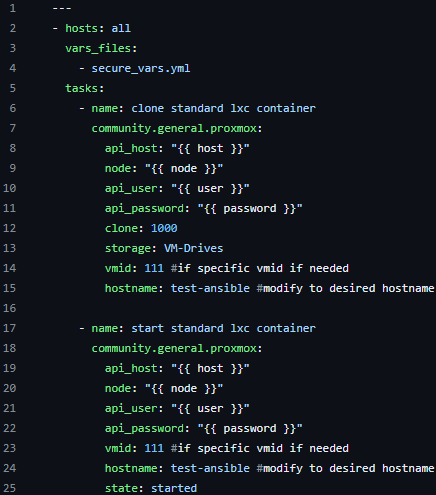
Szintén hasonló elképzelésen alapul a következő két playbook-om, amivel felhasználókat lehet létrehozni a kliens gépeken (21. ábra). A szkript futtatását megelőzően módosítani kell a playbook-ot, mivel meg kell határozni a felhasználó nevét, jelszavát. Ha sudo felhasználót szeretnénk létrehozni, akkor 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. 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.

*21. ábra, Ansible User*

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/ könyvtárjába. 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-tet leklónozom minden egyes gép létrehozásnál. 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.

*22. ábra, Deploy*

5.3 Egyéb, fel nem használt feladatok

Nem került felhasználásra két playbook, amit készítettem, mert nem találtam relevánsnak a szakdolgozatom témájához. Egy említést úgy gondoltam, hogy mégis megérnek, mert az egyik szolgált a monitoring megoldásomnak, a másik pedig arra szolgál, hogy az mások is el tudják érni a fizikai szervert, amin fut a Proxmox virtualizációs környezet.

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 a használata.

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.

5.4 Ansible Vault

A fent említett státusz és fizikai kihasználtság monitorozására használt szkriptek (16. és 17. ábra) harmadik és negyedik sorában található egy ilyen kódrészlet: var\_files: - /etc/ansible/secure\_vars.yaml Ez az Ansible Vault 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 biztosítani. 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

6.1 DNS (Domain Name System)

Idő hiányában kimaradt több fontos funkció is, hogy a szakdolgozatom „nagyvállalati környezetben” részét jobban lefedje. Ennek egyike a DNS szerver használata. Ezáltam nem IP címekkel kellene dolgozni, hanem lehetőség lenne teljesen minősített tartománynevek, azaz FQDN használatára. Lehetőség lett volna különböző domain-ek készítésére is, ami elengedhetetlen része egy nagyvállalati környezetnek.

6.2 SQL (Structured Query Language)

Továbbá, szerettem volna készíteni több SQL adatbázis szervert is. Ehhez a playbook elkészült, ami az LXC konténer egy virtuális számítógép létrehozásához nagyon hasonló, viszont miután elkészült egy adott gép, utána lefuttat egy bash szkript fájlt. Ez a szkript fájl a felelős a szükséges csomagok telepítéséért, az SQL felhasználó létrehozásáért és egy minta adatbázis készítéséért. High Availability.

6.3 Telefonhívás monitoring jelzés esetén