**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](#_Toc161903554)

[2.Felhasznált eszközök 5](#_Toc161903555)

[Ansible 5](#_Toc161903556)

[Proxmox 6](#_Toc161903557)

[Semaphore 6](#_Toc161903558)

[Nagios Core 7](#_Toc161903559)

[3.Környezet kiépítése 7](#_Toc161903560)

[4.Semaphore 10](#_Toc161903561)

[Semaphore bevezetés, működés 11](#_Toc161903562)

[Semaphore konfiguráció 11](#_Toc161903563)

[5.Ansible 14](#_Toc161903564)

[Ansible bevezetés 14](#_Toc161903565)

[Automatizált feladatok 15](#_Toc161903566)

[Ansible Vault 19](#_Toc161903567)

1.Bevezetés

2.Felhasznált eszközök

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

- Ansible automatizációs eszköz

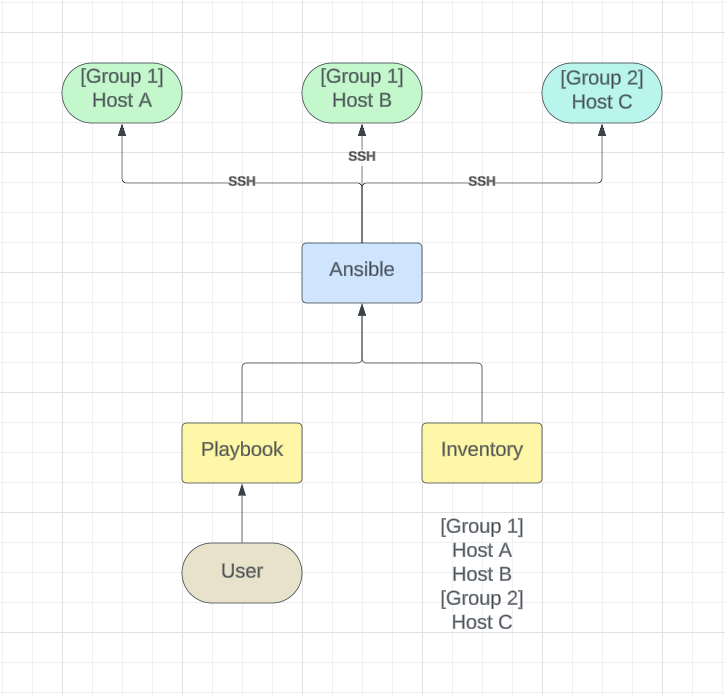
- Proxmox virtualizációs környezet (PVE)

- Semaphore kezelői felület Ansible-höz

- Nagios Core monitoring

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*

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.

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.

Nagios Core

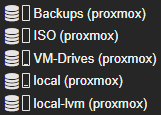
A Nagios Core egy ingyenes, nyílt forráskódú monitoring eszköz. Fő feladata a szerverek, hálózati eszközök, alkalmazások és szolgáltatások állapotának folyamatos figyelése, valamint a hibák és problémák gyors észlelése. A Nagios Core előnyei: Szinte bármely komponenst képes monitorozni, legyen az hardver 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*

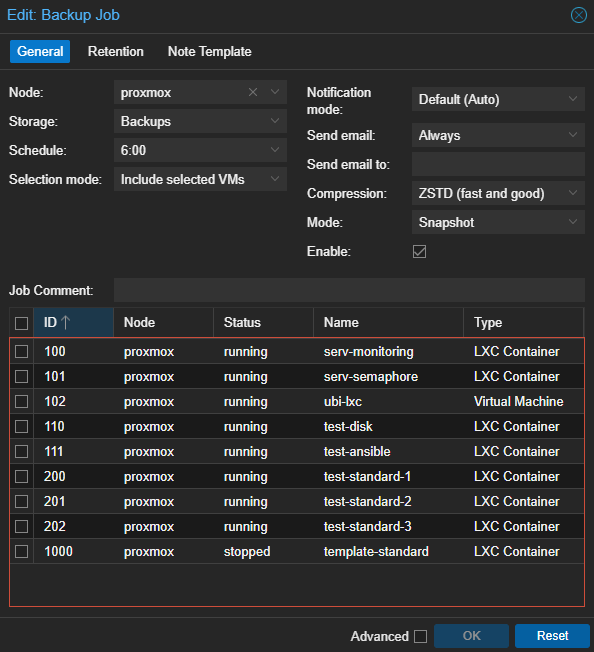
3.Környezet kiépítése

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)

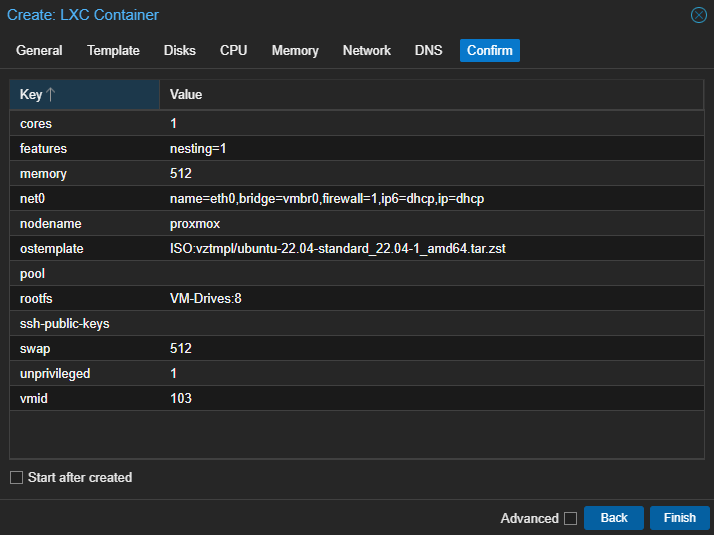


*4. ábra, Proxmox Backup*

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.

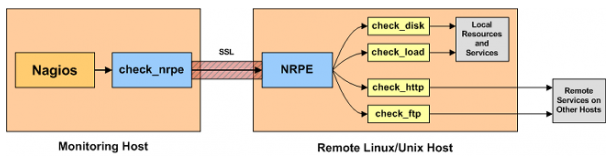
A második ’Retention’ fülnél lehet beállítani, hogy meddig tartsa meg a biztonsági mentéseket, jelenleg nekem az utolsó három mentést tartja meg. Ezen felül van lehetőség óránkénti, napi, heti, havi vagy akár éves bizonyos számú biztonsági mentég megtartására.

*5. ábra, Proxmox LXC*

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.

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, 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

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.

Semaphore konfiguráció

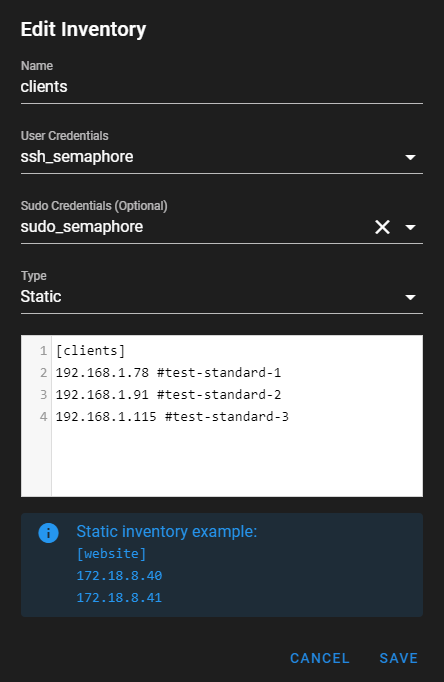
A csomagok telepítése nem okoz gondot, a dokumentációt követve, ezért ugranék is a konfigurációs részhez. A Semaphore web interfészen bejelentkezve, készíteni kell egy projektet. A projekt létrehozása után a legfontosabb lépések a következőek: egy biztonságos csatlakozási lehetőség beállítása a kliens 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. Ugyanebben a menüpontban szükséges megadni még egy ’sudo’ felhasználó-jelszó párost, ha lenne olyan playbook-unk, ami ezt igényelné.

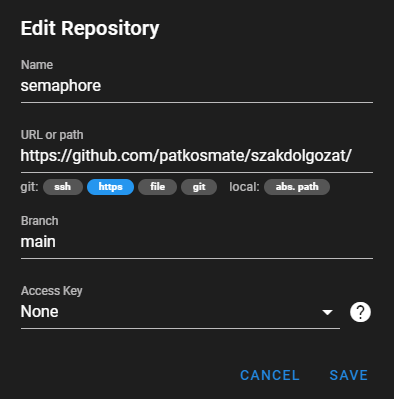
Az ’Inventory’ fülre kattintva lehet létrehozni inventory fájlokat, amelyek lehetnek statikusak-melyek magában a Semaphore-ban tárolódnak, vagy dolgozhatnak egy már meglévő fájlból, ami magán az Ansible szerveren tárolódik lokálisan (8. ábra).

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).

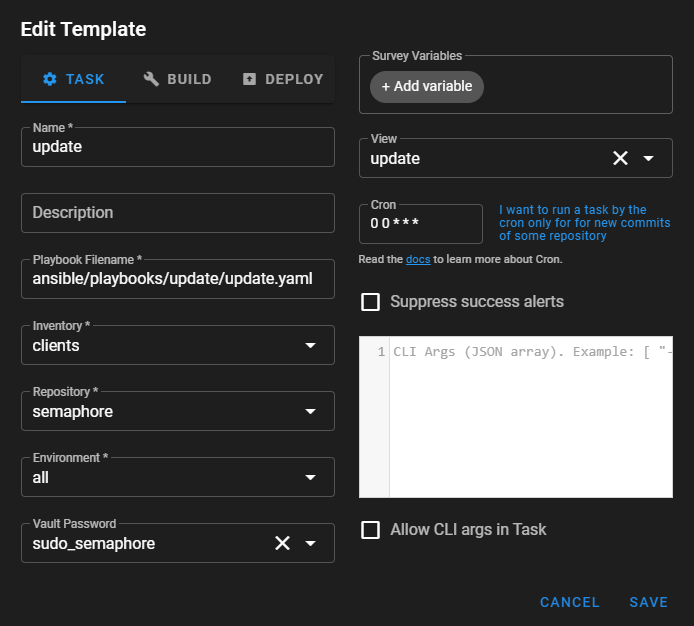
A ’Task Templates’ fülnél lehet létrehozni a feladatokat (jövőben task). Ezeket a taskokat lehet külön csoportokba rendezni az átláthatóság végett (10. ábra). Ezek a taskok fogják majd használni a GitHub repository-ban lévő playbook-okat.



*8. ábra, Semaphore Inventory*

**

*9. ábra, Semaphore Repository*

**

*10. ábra, Semaphore Task*

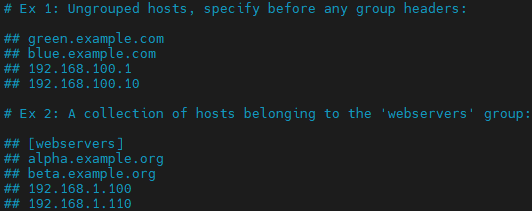
5.Ansible

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.

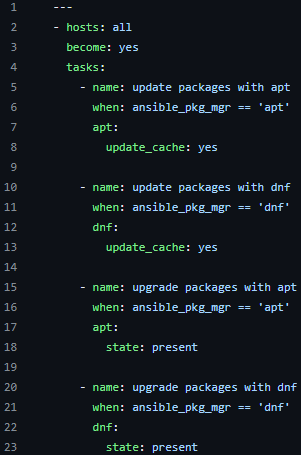
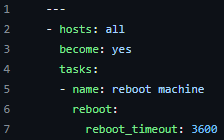
Az Ansible inventory fájlokban találhatóak a célszervereink. Lehet több inventory fájlt is készíteni, vagy akár egy inventory fájlban több csoportot is létrehozni. Az előbb említett ’ansible-playbook’ parancs kiegészül egy ’-inventory’ kapcsolóval, ami után az inventory fájlunkat tudjuk specifikálni, vagy a fájlban lévő csoportot. (7. ábra)



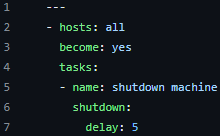
*7. ábra, Ansible Inventory*

Automatizált feladatok

Első playbook, amit készítettem, az talán a legfontosabb mind közül, egy szoftverfrissítő playbook. Jelenlegi formájában (8. 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. Továbbá kiegészül egy rendszer újraindítással a frissítési ciklus megkezdése előtt, amire a rendszerek hosszú futás ideje miatt van szükség. Készítettem továbbá egy-egy különálló újraindító (9. ábra) és leállító (10. ábra) playbook-ot a Linux rendszerekhez.



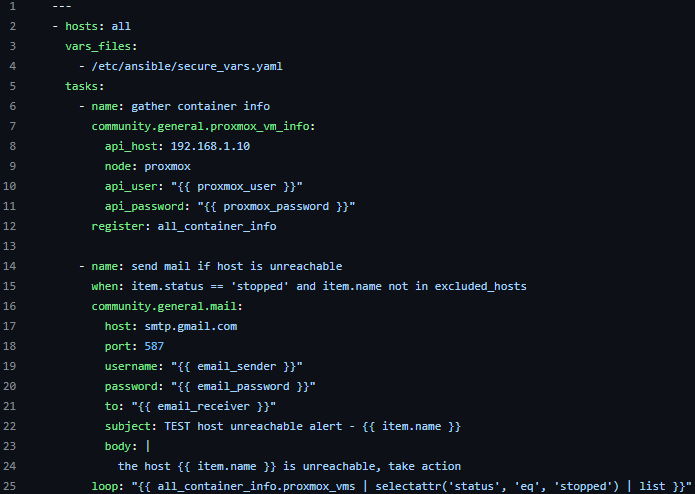
*9. ábra, Ansible Reboot*



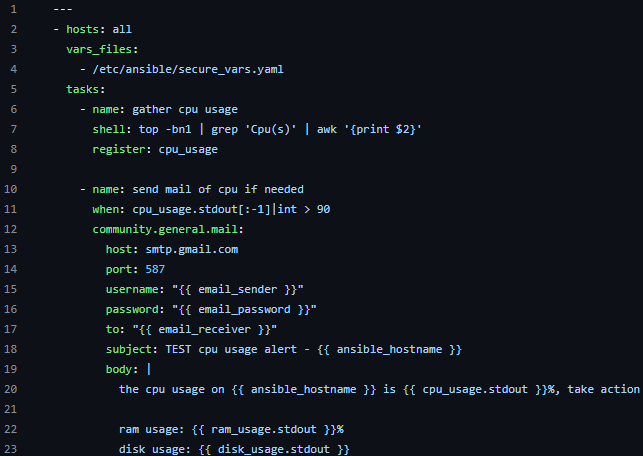
*10. ábra, Ansible Shutdown*

*8. ábra, Ansible update-apt, update-dnf*

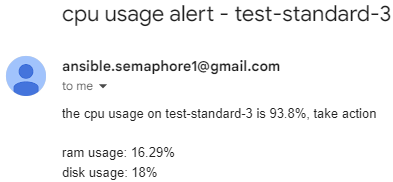
A következő két fontos playbook-om, egy-egy monitoring megoldás volt. Ezt könnyen megoldhattam volna a Nagios segítségével, viszont szerettem volna ezeket is Ansible-lel. A szóban forgó automatizációs szkriptek a rendszerek státuszát (11. ábra), és fizikai kihasználtságát monitorozzák (12. ábra). Ha valamelyik rendszer nem elérhető az Ansible számára, vagy a hardver kihasználtság kritikus, ami jelen esetben 90%, akkor arról e-mailben értesít (13. 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.



*11. ábra, Ansible Status*



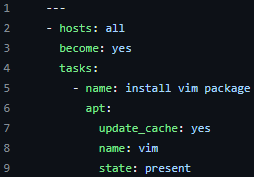
*12. ábra, Ansible CPU*



*13. ábra, Ansible E-Mail*

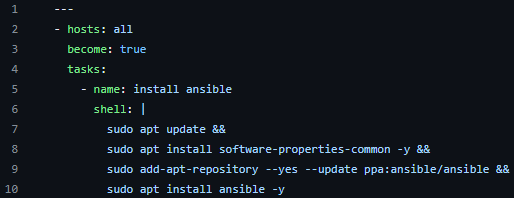
Ez a három, eddig említett playbook, mind ütemezetten futnak. A Semaphore beépített cron-ját használom, ami a Unix alapú rendszereknél szabályos időközönként futtat programokat. Jelenleg a rendszerfrissítő szkript minden nap éjfélkor, a monitorozásra használt szkriptek pedig ötpercenként. Ezeket a következő 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 (14. 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.



*14. 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 (15. 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.



*15. ábra, Ansible Command*

Szintén hasonló elképzelésen alapul a következő két playbook-om, amivel felhasználókat lehet létrehozni a kliens gépeken. 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.

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 vagy virtuális gép 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.

Ansible Vault

A fent említett státusz és fizikai kihasználtság monitorozására használt szkriptek (11. és 12. 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 11. ábrán látható proxmox\_user, proxmox\_password, email\_sender, email\_password, email\_receiver változók mind ebben a fájlban találhatóak.