**Neumann János   
Informatikai Kar**

**Szakdolgozat**

**oe-nik** Hallgató neve: **Gyenes Sándor Gábor**

**2020** Hallgató törzskönyvi száma: **T/005109/FI12904/N**

Tartalomjegyzék

[1 Bevezetés 3](#_Toc55830742)

[2 Orkesztrációs eszköz 4](#_Toc55830743)

[2.1 Az Occopus 4](#_Toc55830744)

[2.2 Leíró fájlok 5](#_Toc55830745)

[2.3 Kiépíthető szolgáltatások 7](#_Toc55830746)

[3 Tesztelési lehetőségek 8](#_Toc55830747)

[3.1 Elérhető eszközök 8](#_Toc55830748)

[3.2 Jenkins 10](#_Toc55830749)

[3.3 Jenkins – Tesztelési Lehetőségek 11](#_Toc55830750)

[3.3.1 Pipeline 11](#_Toc55830751)

[3.3.2 Freestyle tesztelés 14](#_Toc55830752)

[4 Tesztelési módszerek 15](#_Toc55830753)

[4.1 Konfiguráció tesztelése 15](#_Toc55830754)

[4.2 Funkcionális teszt 15](#_Toc55830755)

[5 Rendszerterv 17](#_Toc55830756)

[5.1 Felhő infrastruktúra 17](#_Toc55830757)

[5.2 Kapcsolódás 17](#_Toc55830758)

[5.3 Operációs rendszer 18](#_Toc55830759)

[5.4 Szoftverek 18](#_Toc55830760)

[5.4.1 Occopus 19](#_Toc55830761)

[5.4.2 Jenkins 20](#_Toc55830762)

[5.5 Tesztelési környezet 20](#_Toc55830763)

[5.5.1 Automatizmus 21](#_Toc55830764)

[5.5.2 Teszteléshez szükséges kapcsolat 21](#_Toc55830765)

[5.5.3 Tesztelt szolgáltatások 21](#_Toc55830766)

[5.5.4 Értesítés 21](#_Toc55830767)

[6 Környezet kialakítása 22](#_Toc55830768)

[6.1 Előkészületek 22](#_Toc55830769)

[6.1.1 RSA kulcs generálás 22](#_Toc55830770)

[6.1.2 Opennebula 23](#_Toc55830771)

[6.2 Ubuntu telepítés 24](#_Toc55830772)

[6.3 Jenkins telepítés 25](#_Toc55830773)

[6.4 Occopus telepítés 29](#_Toc55830774)

[6.5 Jenkins ssh kapcsolódás konfigurálása 31](#_Toc55830775)

[7 Jenkins projektek kialakítása 33](#_Toc55830776)

[7.1 Általános jenkins folyamat 33](#_Toc55830777)

[7.1.1 Github repositroy trigger 33](#_Toc55830778)

[7.1.2 Leíró fájlok letöltése 33](#_Toc55830779)

[7.1.3 Környezet Kiépítése 34](#_Toc55830780)

[7.1.4 Környezet tesztelésére szolgáló scriptek 35](#_Toc55830781)

[7.1.5 Környezet lebontása és értesítés küldés 36](#_Toc55830782)

[7.2 Tensorflow Specifikus folyamatok 38](#_Toc55830783)

[8 Tesztelés 39](#_Toc55830784)

[8.1 Tesztelést végző scriptek 39](#_Toc55830785)

[8.2 Apache Spark klaszter 40](#_Toc55830786)

[8.3 Tensorflow 41](#_Toc55830787)

[8.4 Tesztelési eredmények 41](#_Toc55830788)

[9 Összefoglalás 42](#_Toc55830789)

[10 Irodalomjegyzék 43](#_Toc55830790)

[11 Ábrajegyzék 45](#_Toc55830791)

[12 Táblázatjegyzék 45](#_Toc55830792)

# Bevezetés

Napjainkban egyre jobban elterjedtek a felhőszolgáltatást nyújtó vállalatok, mivel ezekre a szolgáltatásokra hatalmas igény mutatkozik. Napról napra egyre több kutatási területen használnak elosztott felhőszolgáltatásokat, és egyre több vállalkozás dönt arról, hogy valamilyen felhő alapú szolgáltatást vegyen igénybe. Ezek a szolgáltatások könnyen kezelhetők, költséghatékonyak, és nagyon sok területen megfelelően alkalmazhatóak így megfelelő választás lehet akár egy teljes vállalati szerver infrastruktúra helyettesítésére.

Azonban a felhő alapú szolgáltatások kiépítése és konfigurálása egy időigényes folyamat, ezt a folyamatot hivatottak gyorsítani, és egyszerűsíteni a különféle felhő orkesztrációs eszközök. Ezek az eszközök képesek előre meghatározott virtuális infrastruktúra kiépítésére, igény szerinti szolgáltatások kialakítására, illetve különböző felhőszolgáltatóknál a virtuális gépek konfigurálására. Mindezeken felül képes összehangoltan kialakítani a kívánt szolgáltatást, akár több különböző felhő környezetben is. Ezzel a módszerrel nem csak az infrastruktúra kialakítása biztosítható, hanem a szolgáltatás gyors rendelkezésre állása is, mely javítja a szolgáltatás megítélését.

Azon túl, hogy ezek az eszközök képesek az infrastruktúrák kialakítására, meg kell győződni arról, hogy ezek valóban helyesen épültek fel, illetve a szolgáltatás működése is megfelel az elvárásoknak. Ehhez nem elég csak funkcionalitásában tesztelni a már elkészült szolgáltatásokat, hanem szükséges már a kiépítés folyamatát is tesztelni. Hiszen ekkor derüli ki, hogy az orkesztrátor működését biztosító úgynevezett leíró fájlok helyesen lettek-e létrehozva, az infrastruktúra kialakításához szükséges csomagok és állományok elérhetőek, valamint a telepítés menete az elvárt működésnek megfelelően történik.

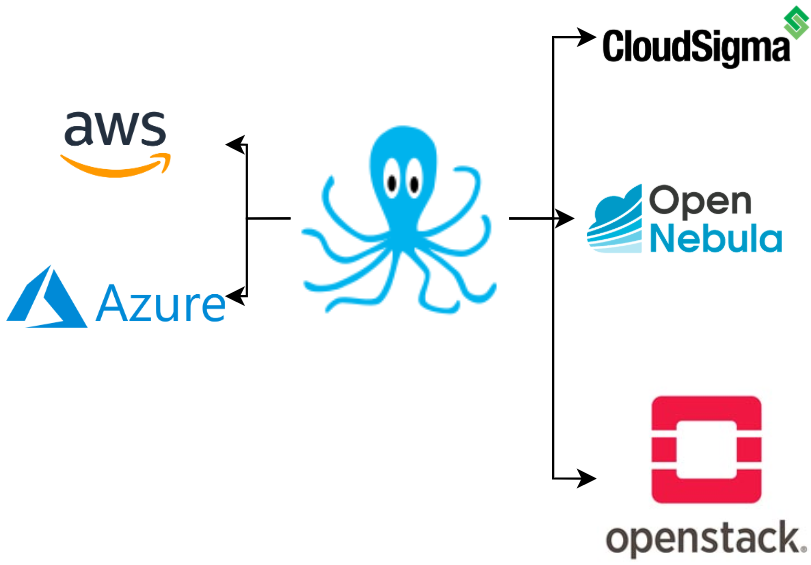
Dolgozatom célja, hogy valamilyen felhő orkesztrációs eszközt használva szolgáltatásokat hozzak létre egy, vagy akár több különböző felhőszolgáltatónál, melyeket már a kiépülés állapotában automatikusan fogok tesztelni. Ennek elvégzéséhez találnom kell a célomnak megfelelő orkesztrációs eszközt. Továbbá egy olyan tesztelési keretrendszert, amely képes arra, hogy megadott paraméterek alapján automatikusan már a kiépülés folyamatától kezdődően vizsgálni tudja a rendszer helyességét egészen a funkcionális tesztekig. A dolgozatom irodalomkutatási részében tervezem a piacon megtalálható keretrendszereket összehasonlítani, és igényeimnek megfelelően kiválasztani az általam preferált megoldást. A dolgozatom célja az, hogy meggyőződjek arról, hogy a környezet kiépítéséhez használt leíró fájlok pontosan és helyesen lettek kialakítva, a szükséges csomagok elérhetőek és a kialakított szolgáltatás működőképes. Valamint a tesztelés eredményéről a kompetens személyeket valamilyen kommunikációs csatornán keresztül a rendszer dinamikus módon értesítse.

# Orkesztrációs eszköz

Szakdolgozatom megvalósításához szükséges használnom valamilyen felhő orkesztrációs eszközt. Ezek az eszközök képesek felhős környezetre fejlesztett elosztott alkalmazásokat felépíteni, és konfigurálni ezért is nevezzük ezeket orkesztrátoroknak. Több erre a célra fejlesztett eszköz is elérhető, például *Azure Automation*, *IBM Cloud Orchestrator*, *Terraform* vagy akár a SZTAKI által fejlesztett *Occopus*. A dolgozatom elkészítése során egy platform független eszközre van szükségem, így az Azure illetve az IBM megoldása számomra nem megfelelő. A *Terraform* illetve az *Occopus* rendszerek ugyan minden szükséges funkciót támogatnak, azonban a *Terraform* rendszer számos egyéb lehetőséget is biztosít melyekre nincsen szükségem. Illetve az *Occopus* rendszerrel vannak meg a szükséges előismereteim. Így a választásom erre a rendszerre esett.

## **Az Occopus**

Az Occopus egy olyan nyílt forráskodú, python alapú keretrendszer mely a fentebb említett orkesztrációs szolgáltatásokat nyújtja különféle felhőinterfészeket támogatva. Ezzel biztosítva, hogy a kiépíteni kívánt szolgáltatásokat szükséges esetben több, egymástól különböző felhő rendszeren is meg tudjam vizsgálni. Támogatott felhőinterfész az EC2 (*Amazon Web Services, Opennebula*), Azure, Nova (*OpenStack*), CloudSigma, és a CloudBroker. Látható tehát hogy a jenlegi legelterjedtebb felhőinterfészeket megfelelő mértékben támogatja. Az alábbi ábrán ennek szemléltetése látható. [1]



.. ábra: Az Occopus által támogatott felhőszolgáltatások

Ahhoz, hogy az Occopus rendszer képes legyen orkesztrálási feladatokat ellátni előbb telepíteni kell a keretrendszert. Az Occopus rendszer hivatalosan csak Ubuntu operációs rendszert támogat, de a rendszerből készült Docker konténer képfájl, mely szélesebb körben elérhető és használható.

## **Leíró fájlok**

Az orkesztrációs rendszerek általunk előre definiált leíró fájlok alapján tudják felépíteni és konfigurálni a kívánt architektúrát és szolgáltatást. A leírók különböző kiterjesztésűek lehetnek, melyeket az orkesztrátor határoz meg. A leggyakoribb leíró típusok a yaml, json vagy akár hcl formátum. Nem csak támogatott formátumban, hanem megközelítési módban is eltérhetnek egymástól. A megközelítési mód lehet deklaratív mely során a kiépíteni kívánt rendszer végállapotát szükséges definiálni, ebben az esetben az orkesztrátor feladata, hogy elérje a kívánt végállapotot. A másik alkalmazott megközelítési mód, pedig amikor a kiépítéshez szükséges lépéseket kell meghatározni, majd az orkesztrátor ezeket a lépéseket végig iterálva építi ki a rendszert az elvárt állapotba.

Az Occopus estén három leírót szükséges létrehozni, és kialakítani a megfelelő szolgáltatás kiépítése érdekében. Szükség van egy infrastruktúra leíró fájlra, egy állomás leíró fájlra, és egy úgy nevezett kontextualizációs fájlra.

Az infrastruktúra leíró fájl tartalmazza a kiépíteni kívánt infrastruktúra leírását, annak nevét, illetve különböző meta infromációkat, mint például: a létrehozójának adatait, stb. Tartalmaznia kell továbbá a gépek kapcsolatát, és ezek egymáshoz viszonyított függőségeit. Az infrastruktúra leíró állományban található egy nodes nevű szekció, melyben megadhatók a használt gépekre vagy más néven állomásokra vonatkozó információk. Megadható az állomás neve, ami alapján könnyebben lehet majd a gépet azonosítani a kiépített rendszerben, illetve egy típust kell hozzá társítani, melyet majd az állomás definíciós fájlban kell pontosan meghatározni. Továbbá skálázási paraméterek is megadhatók a node szekcióban arra vonatkozóan, hogy egy adott, állomásból minimum és maximum mennyi kerülhet kiépítésre az infrastruktúrában. Igény szerint megadhatók különféle változók is állomásonként, amelyek az egyes szolgáltatások kialakítása során a felhasznált kontextualizációs állományokban is elérhetőek, így akár létrehozott állomásonként is eltérően épülhet ki a szolgáltatás. [1]

1. infra\_name: ec2-helloworld-infra
2. user\_id: somebody@somewhere.com
4. nodes:
5. - &A
6. name: helloworld
7. type: ec2\_helloworld\_node
8. variables:
9. message: "Hello World! I have been created by Occopus"

Az állomás definíciós fájlban pontosan meg kell határozni az egyes típusokat, melyek az infrastruktúra leíróban megadásra kerültek. Típus meghatározás esetén szükséges definiálni azt, hogy az adott állomás típus milyen felhő infrastruktúrán kerüljön kiépítésre, illetve az ehhez szükséges felhő API elérését és a kiépítés szempontjából lényeges további információkat kell kitölteni. A különböző felhő API-k mindegyike nagyjából azonos típusú információkat vár, azonban ezeket felhő interfészenként eltérő formátumban kell megadni. EC2 interfész esetén többek között szükséges a régió megnevezés, a felhasznált kép fájl azonosítója, vagy akár a biztonsági csoport azonosítója is. [1]

A harmadik leíró állomány, amit szükséges meghatározni az úgynevezett kontextualizációs fájl. A kontextualizációért felelős eszközt különböző operációs rendszerek esetén eltérően hívjuk. Linux alapú rendszereknél cloud-init, windows alapú rendszereknél cloudbase-init, de létezik az opennebula context is, ami pedig az OpenNebula saját megvalósítása linux és windows operációs rendszerekhez. Tulajdonképpen egy olyan script, amely a létrehozott virtuális gép első indulása után automatikusan lefut. Tehát ez a leíró felelős a szolgáltatáshoz szükséges fájlok másolásáért és telepítéséért, felhasználók létrehozásáért, illetve a környezet kialakításánál szükséges parancsok végrehajtásáért. [1]

1. #cloud-config
2. write\_files:
3. - content: "{{variables.message}}\n"
4. path: /tmp/helloworld.txt
5. permissions: '0644'

Ahhoz hogy az Occopus képes legyen a támogatott felhő infrastruktúrákon önállóan létrehozni a leírók alapján definiált rendszereket, elengedhetetlen készíteni egy authentikációs fájlt a megfelelő helyre auth\_data.yaml néven. Ebben a fájlban szükséges megadni a különféle felhő interfészekhez a kapcsolódási adatokat. Az itt megadottak alapján lesz képes az Occopus hozzáférni az egyes felhőszolgáltatásokon elérhető erőforrásainkhoz. Tehát tulajdonképpen egy, a szolgáltatáson létrehozott felhasználó adatainak segítségével képes autentikálni magát így a megfelelő felhasználóhoz kapcsolni a létrehozott infrastuktúrákat.

1. resource:
2. -
3. type: ec2
4. auth\_data:
5. accesskey: your\_acceskey
6. secretkey: your\_secretkey
7. -
8. type: nova
9. auth\_data:
10. username: your\_username
11. password: your\_password

Ezen leírók, illetve az autentikációs fájl segítségével az Occopus már képes arra, hogy azonosítsa magát felhőszolgáltatásokon, és ott saját virtuális infrastruktúrát, illetve szolgáltatásokat építsen ki. Az egyszerű csomag telepítésen és fájlmásoláson túl, akár egy teljes értékű Apache Spark Cluster, vagy egy TensorFlow környezet kialakítására is képes.

## **Kiépíthető szolgáltatások**

Az Occopus által számos különböző célú szolgáltatás könnyedén kialakítható kezdve egy egyszerű szöveges állomány létrehozásától akár egy automatikusan skálázott klaszter, vagy egy gépi tanulást támogató infrastruktúra kialakításáig. [2]

Dolgozatomban két konkrét szolgáltatást fogok kialakítani, melyeket a kialakításuk után tesztelni fogok, hogy megfelelően kiépültek-e, illetve a konfiguráció az elvárt működésnek megfelelő-e. A megvalósításom során az Apache Spark Klaszter, és a TensorFlow keretrendszerek kiépítését fogom tesztelni.

Az Apache Spark egy olyan klaszter keretrendszer, amely magas szintű API-t biztosít többek között Python és Java programnyelvekhez. Illetve támogat különféle eszközöket strukturált, vagy gráf adatok feldolgozásához, és gépi tanuláshoz. Az egyik nagy előnye hogy támogatja az adatok memóriában történő feldolgozását a számítási teljesítmény növelése érdekében. [3]

Az Apache Spark klaszter kialakításához két kontextualizációs állomány szükséges, mivel ezen rendszer esetén két különböző állomást kell definiálni. Az egyik az úgynevezett *master*, a második állomás típus pedig a *worker*. Ezen típusok kialakítása eltér, tehát az alapkonfigurációjukban is eltérések vannak. Ezt leszámítva az infrastruktúra, illetve az állomás leíró fájlokból továbbra is csak egy-egy szükséges. Azonban az állomás definíciós fájlban külön meg kell fogalmazni a *master* illetve a *worker* állomások paramétereit

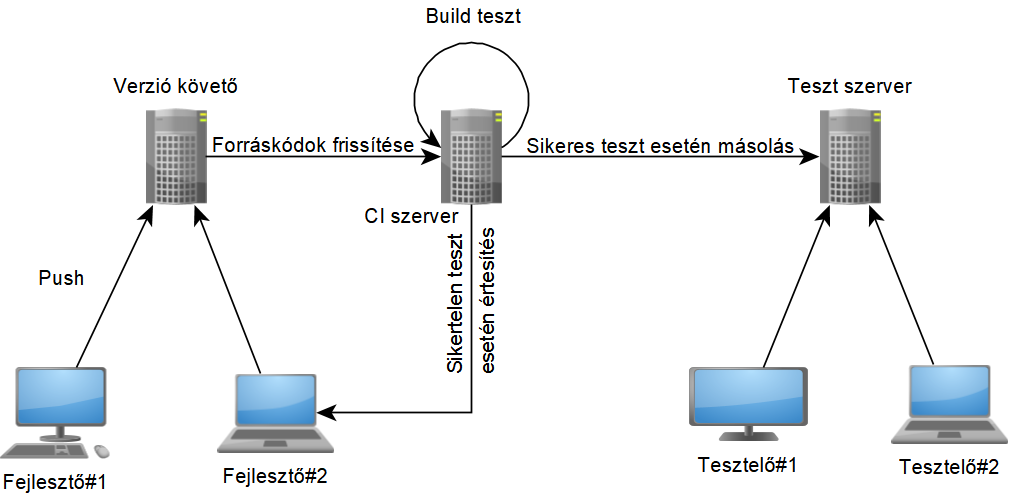
A TensorFlow pedig egy gépi tanulást támogató nyílt forráskodú platform melyet még a Google kezdett el fejleszteni. Lehetővé teszi neurális hálózatok kialakítását, és azok mélyreható vizsgálatát, különféle felhasználási célokra. [4]

# Tesztelési lehetőségek

## **Elérhető eszközök**

Dolgozatom fő célja, hogy a korábbiakban említett eszközökkel, és módon kialakítsam a szolgáltatásokat, illetve azokat a létrehozás után valamilyen módon teszteljem. Ehhez szükségem van egy tesztelő eszközre, amely képes a leíró állományokban meghatározott rendszereket a kiépülésük során megvizsgálni és ellenőrizni, hogy a szükséges állományok megfelelően létrejöttek-e, illetve a szolgáltatások rendeltetésszerű működéséhez szükséges konfigurációs beállításokat tartalmazó fájlok is helyesek. Erre a célra több elérhető eszközt találtam, melyek összefoglaló néven a *Continuous integration* eszközök.

A *Continuous integration* eszközök lényege tulajdonképpen az, hogy gördülékenyebbé tegyék a fejlesztők közötti munkát. Amikor egy fejlesztő elkészít egy program kódot, vagy egy meglévőt módosít, és ezeket feltölti valamilyen verziókezelő rendszerbe akkor ezt egy folyamatos integrációt biztosító szerver érzékeli, elkészíti a kódokból a megfelelő programot, és teszteli annak futását. Majd végül az így elkészült szoftvert elérhetővé teszi a tesztelést végző személyek számára egy kitüntetett szerveren. Ha a szoftver kialakítása során hibát érzékel, arról értesíti a fejlesztőket, így lehetővé téve egy gördülékeny ellenőrzési folyamatot a fejlesztés közben. Általában a tesztek kimenetelei és sikerességei alapján statisztikákat vezetnek, továbbá egy naplót arról, hogy mely esetekben és mikor volt sikeres, illetve sikertelen a teszt futása. Ez alapján, nyomon követhető a változás lista, illetve a tesztelések kimenetele is. [5]



.. ábra: A "Continuous integration" eszközök mműködése

Ezen célokra számos elérhető eszközt találtam, és hasonlítottam össze, hogy megtaláljam a dolgozat által kitűzött céloknak legjobban megfelelő szoftvert. A dolgozatom során négy keretrendszert hasonlítottam össze különböző szempontok szerint annak érdekében, hogy a számomra, és a feladat szempontjából is megfelelő eszközt találjak. Ezek a vizsgált szoftverek a Jenkins, Bamboo, TeamCity, és az Apache Continuum. Ezen szoftverek mindegyike folyamatos integrációs eszköz, viszont egyes szempontokat vizsgálva ezek mégis eltérőek. Kutatásom során a következő tulajdonságoknak megfelelő eszközt kerestem: nyílt forráskódú, bővíthető különféle kiegészítők segítségével, jelenleg is támogatott, ingyenesen elérhető, illetve értesítési lehetőségeket is tartalmazzon.

Fontos, hogy a tesztelő szoftver bővíthető legyen, ugyanis a dolgozatom utolsó szakaszában igyekszem megfogalmazni, hogy milyen lehetőségek vannak annak továbbfejlesztésére, amelyet egy bővíthető tesztelő eszköz megfelelően tud támogatni. A fent említett szoftverek leírásait megvizsgálva, ezt a követelményt mindegyik maradéktalanul teljesíti.

Elengedhetetlen szempont az értesítési lehetőség, szerencsére ezt is minden általam kiválasztott szoftver támogatja. Erre a funkcióra azért van szükségem, hogy ha a kiépítés során bármilyen hiba történik, akkor arról közvetlenül értesíteni tudjon a rendszer. Tulajdonképpen a dolgozat egyik cél kitűzése is az, hogy ilyen esetben valamilyen értesítés történjen.

A vizsgálat során további szempont volt az árazás vizsgálata. Lehetőség szerint olyan szoftvert keresek a tesztelési környezet megvalósításához, amely ingyenes, illetve jelenleg is támogatott. Árazási szempontból a Jenkins, illetve az Apache Continuum szoftverek teljesítették az elvárást, azonban az utóbbi szoftver már nem támogatott.

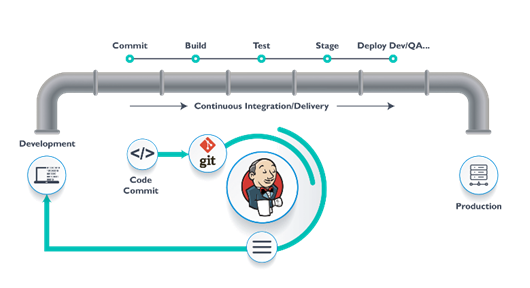
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Név | Nyílt forráskódú | Értesítés | Bővíthető | Támogatott | Ingyenes |
| Jenkins | Igen | Igen | Igen | Igen | Igen |
| Bamboo | Nem | Igen | Igen | Igen | Nem |
| TeamCity | Nem | Igen | Igen | Igen | Igen (Korlátozott) |
| Apache Continuum | Nem | Igen | Igen | Nem | Igen |

.. táblázat: A tesztelési környezet kiválasztásához vizsgált szoftverek

Ezen szempontokat összevetve, a tesztelési környezet kialakításához a Jenkins programot választottam, hiszen az általam támasztott összes elvárást ez a szoftver teljesíti, és megfelelő funkcionalitással rendelkezik a tesztek elvégzéséhez.

## **Jenkins**

A Jenkins egy olyan nyílt forráskódú szoftver, amely különféle, szoftverfejlesztéssel kapcsolatos automatizálási feladatokra felhasználható. Folyamatában képes tesztelni a szoftver felépítését, majd ezt követően annak funkcionalitását. Az összetett tesztek elvégzését különféle eseményekkel automatizáltan is el tudja indítani. Ilyen kiváltó esemény lehet egy verziókezelő rendszerben történő commit művelet, ütemezhető idő szerint, vagy akár megadható, hogy bizonyos teszteket csak akkor hajtson végre, amikor egyéb kijelölt teszteket már elvégzett. Tehát megfelelően automatizálható tesztelési lehetőségeket nyújt, melyeket akár együttesen is tudunk alkalmazni.



.2. ábra: A Jenkins működésének folyamata [6]

A Jenkins egy szerver alapú szoftver, melyet lehetőség van telepíteni Windows/Linux/Unix rendszerekre, de igény szerint használható Docker konténeres környezetben is. Ahhoz, hogy közvetlenül egy operációsrendszerre fel lehessen telepíteni, létezik egy telepítő csomag, mely a Jenkins hivatalos oldalán elérhető, és ingyenesen hozzáférhető. Az utóbbi megvalósításhoz azonban szükséges először a Docker környezet kialakítása az operációs rendszernek megfelelően. Majd ezt követően ki kell alakítani a működéséhez szükséges hálózatot, illetve meg kell határozni a program szükséges paramétereit.

A szoftver rendelkezik grafikus felülettel, mely a telepítés után hálózaton keresztül elérhető, a megfelelő útvonal böngészőben történő megadásával. Ez a grafikus felület megkönnyíti a tesztek, és tesztesetek menedzselését, és átlátható felületet biztosít a szükséges feladatok meghatározásához. Továbbá a Jenkins biztosít felhasználó-, és jogosultságkezelést is, így akár egyszerre több projekt tesztelése történhet egymástól elszeparált formában.

Számos funkció kiegészítés, úgynevezett *plugin* is készült a Jenkins-hez amelyek nem csak bővítik a beépített lehetőségeket, hanem egyszerűbbé és gyorsabbá teszik a szoftver használatát. A teljesség igénye nélkül ilyen kiegészítők például a különféle projekteket (.NET, C#, Mobil) támogató eszközök, de található AWS kompatibilis, és értesítést lehetővé tevő kiegészítő is. A különböző értesítéseket biztosító kiegészítőkre a környezet kialakításánál szükségem lesz, hiszen a tesztelés eredményéről szükséges esetben értesíteni kell az illetékes személyeket.

## **Jenkins – Tesztelési Lehetőségek**

### **Pipeline**

A *Pipeline* egymáshoz kapcsolódó feladatok sorozatának ellenőrzését teszi lehetővé. Lényege, hogy különféle állapotokat, vagyis szakaszokat lehet meghatározni a teszt elkészítése során. Ezzel egy adott szoftvert is készítésének különböző szakaszaiban lehet megvizsgálni, a *build* folyamatától egészen a funkcionális tesztekig. Ez a tesztelés egy komplex folyamat, amelyen a szoftver minden változtatás után végig megy.

A *Pipeline* tesztelés során a végrehajtandó folyamatot kód formájában kell meg adni a környezetnek, amely alapján a kívánt rendszert tesztelni fogja. Ennek a tesztelési lehetőségnek több előnye is van. Ilyen például a szüneteltethetőség, amely lehetővé teszi a folyamat szüneteltetését a teszt során, vagy továbblépés előtt, ha szükséges képes emberi beavatkozásra várni. [7]

1. pipeline {
2. agent { label 'master' }
3. stages {
4. stage('usercheck') {
5. steps {
6. sh "whoami"
7. }
8. }
9. stage('import') {
10. steps {
11. sh "occopus-import nodes/node\_definitions.yaml"
12. }
13. }
14. stage('build') {
15. steps {
16. sh "occopus-build infra-ec2-helloworld.yaml"
17. }
18. }
19. stage('getip') {
20. steps {
21. sh "Jenkins\_Connect/connector.sh ${env.JOB\_NAME}"
22. }
23. }
24. }
25. }

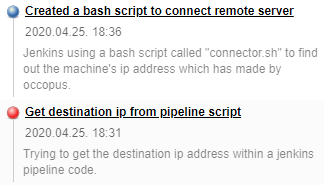
A fenti részleten egy nagyon egyszerű Jenkins *Pipeline* kód látható, melynek célja, hogy az említett tesztelési módszer gyakorlati előnyeit bemutassam. A teszt során különféle állapotokat definiáltam. Az állapot definíciók a szintaxisnak megfelelően *stage()* szóval kezdődnek, ahol a zárójelen belül az adott állapot neve látható, ezzel jobb átláthatóságot biztosítva a kezelőfelületen az egyes állapotok között. Fontos azonban, hogy ezeket az állapotokat sorrendben kell megadni a *Pipeline* kódban, tehát ha két állapot egymásra épül, akkor a végrehajtás szükséges sorrendjében kell ezeket meghatározni. Az állapotokon belül pedig még finomabb felosztással lehet kezelni a teszteket, mégpedig lépések definiálásával, melyek során szintén fontos a végrehajtási sorrend szerinti meghatározás. A lépések kialakításánál adhatók meg a különféle egzakt parancsok, vizsgálatok.

A fenti minta kód egyes lépései az „sh” parancsot tartalmazzák, mely tulajdonképpen shell utasítások végrehajtását teszi lehetővé. Ezzel azon a virtuális Ubuntu eszközön hajtottam végre parancsokat, melyről a Jenkins környezetem is elérhető. Első lépésként a teszthez tartozó konzol felületen megjelenítettem annak a felhasználónak a nevét amelyikkel a szükséges lépéseket végre hajtottam a Jenkins segítségével. Következő lépésként beimportáltam egy korábban létrehozott állomás definíciós fájlt az Occopus keretrendszerbe, majd ezt követően elindítottam a *build* folyamatot, amely az infrastruktúra kiépítését végezte. Legutolsó lépésben pedig egy általam készített „*connector.sh*” nevű *bash script* segítségével megvizsgáltam a létrehozott infrastruktúra IP címét, melynek segítségével a kapcsolat épségét és a kialakított környezetben szükséges fájlok meglétét tudtam megvizsgálni. Ahhoz, hogy a létrehozott infrastruktúrával kapcsolatot tudjak kiépíteni a szükséges IP címet az Occopus napló fájljai között kellett keresnem, melyhez szükségem volt a környezet kialakító *pipeline* teszt nevére. Ezt bemeneti paraméterként adtam meg az elkészített scriptnek egy Jenkins környezeti változót használva. Ezeket a környezeti változókat használva lehet a tesztekről, illetve a Jenkins környezetről szükséges információkat szerezni.

Dolgozatomat tesztelési szempontból valamilyen Pipeline megvalósítás határozza meg, mivel a kialakítandó szolgáltatást is különböző szakaszokban szeretném vizsgálni. Ennek az elvárásnak pedig ez a módszer teljes mértékben megfelel. Ehhez meg kell, határozzam a fő állapotokat, mint a kiépítés, vagy funkciók ellenőrzése. Mindez azért fontos, hogy a környezet megfelelő visszajelzést tudjon biztosítani a fejlesztők számára abban az esetben, ha bármelyik folyamatban hiba keletkezik, illetve így szűkíthető a hibák lehetséges forrása.

Ennél a tesztelési lehetőségnél elérhetőek különféle ellenőrzést segítő funkciók. Ilyen például az állapot nézet, mely során a tesztelő környezet a különböző állapotokat sorrend helyesen jeleníti meg, az állapot végrehajtási idejével együtt. Illetve hiba esetén megfelelő módon jelzi, hogy pontosan melyik állapot, és melyik lépés vezetett a hiba bekövetkezéséhez.

Mivel a különféle tesztesetek névvel, illetve leírással is elláthatók, ezért kiválóan lehet naplózni ezek sikerességét, illetve hibáit. Ezek a naplók az egyes tesztekhez tartozó úgynevezett „történetek”. A fenti, példaként készített *pipeline* tesztelés esetén szintén készült egy napló, mely során percre pontosan nyomon követhetők a teszt kialakításának fázisai, illetve a végrehajtott módosítások.



..1. ábra: Jenkins pipeline projekt előzményei

Itt az egyes bejegyzésekre kattintva látható, hogy melyik felhasználó, vagy milyen esemény hatására indult el a tesztelés. Pipeline tesztelés esetén kiváltó esemény lehet egy másik tesztelés végrehajtása, vagy akár egy *push* művelet egy megadott *repositoryban*. De ezek a tesztelések történhetnek periodikusan is, ha éppen arra lenne szükség.

### **Freestyle tesztelés**

A Jenkins környezet által nyújtott másik tesztelési lehetőség a *freestyle* elnevezésű teszt. Ennek a funkciónak a neve is a sokszínűségére utal, tehát nagyon sok különböző rendszeren lehet vele teszteket végezni, meglehetősen széles körben.

Ez a vizsgálati lehetőség nagymértékben hasonlít a *pipeline* tesztelésre, és annak akár teljes értékű kiváltó lehetősége is lehet. Ebben az esetben is definiálhatók különféle szakaszok, viszont ebben az esetben az egyes szakaszokat már csak lépéseknek és nem állapotoknak kell nevezni. Ezek a lépések is sokszínűek lehetnek, lehet akár egy *shell* parancs végrehajtása, vagy akár egy Windows kompatibilis *batch* parancs futtatása is. Sőt ezek az egymástól meglehetősen eltérő lépések akár egymást is követhetik.

Akárcsak a *pipeline* tesztelés esetén, itt is definiálhatók különböző események melyek a teszt automatikus indítását teszik lehetővé. Ilyen kiváltó esemény lehet egy script futtatása, egy másik teszt befejezése, de automatizálható akár periodikusan idő szerint is.

Nagy előnye a *pipeline* teszteléssel szemben, hogy definiálhatók különböző úgynevezett *post-build* műveletek is. Ezekből a műveletekből igény szerint többet is ki lehet alakítani. Ilyen művelet lehet akár egy egyszerű globálisan definiált email értesítés, egy tesztenként személyre szabható email értesítés, de lehet akár a teszt eredmények xml fájlba történő mentése is.

Természetesen a teszteléshez kapcsolódó utólagos műveletek igény szerint bővítményekkel kiterjeszthetők akár sms értesítési lehetőséggel is. Illetve lehetőség van ezek egyidejű használatára is, mellyel hatékonyabb a fejlesztők értesítése egy esetleges probléma esetén.

# Tesztelési módszerek

Azt követően, hogy a szakdolgozatom alapjául szolgáló rendszer kialakításához szükséges szoftvereket, és rendszereket megvizsgáltam, szükséges a különféle tesztelési lehetőségeimet is megvizsgálnom. A szakdolgozatom ezen szakaszában nem a bejáratott, és ismert tesztelési lehetőségeket kell alkalmazzam, ugyanis azok jellemzően a fejlesztésnél alkalmazott struktúrát követik. Azonban a szakdolgozatommal kapcsolatos célom az, hogy egy már egészében kialakított rendszert vizsgáljak meg annak érdekében, hogy annak működése megfelel-e az elvártaknak, illetve a helyes működését alátámasztó állományok és könyvtárak megfelelően vannak kialakítva.

Ennek érdekében a kialakított eszközökön szükséges a konfigurációs állományok helyességének, a könyvtár rendszerek teljességének, és az eszközök kapcsolatának megfelelő vizsgálata. Ezen vizsgálatokon túl szükséges a rendszer funkcionalitásának ellenőrzése, hiszen a kialakított környezet végső soron akkor tekinthető helyesen kialakított rendszernek, ha a célját maradéktalanul, és hiba nélkül ellátja. Így tehát a teszteléseket két fő csoportba tudom felosztani. Az egyik csoport a fájl szintű ellenőrzések elvégzése, a második csoport pedig a kialakított rendszer funkcionalitásának vizsgálata.

Mind a két esetben szükséges, hogy a rendszereket az automatikus tesztelést megelőzően manuálisan megvizsgáljam. Így pontosan tudni fogom, hogy adott esetben mi az elvárt működés a kialakított rendszerrel kapcsolatban, illetve pontosan fogom tudni azt is, hogy milyen állományok megléte szükséges a megfelelő működéshez, és azok milyen tartalommal kell rendelkezzenek. Ezáltal ki tudok majd alakítani olyan teszteseteket, amelyek a megfelelő végeredményt vizsgálják.

## **Konfiguráció tesztelése**

A konfiguráció tesztelése során szükséges a létrehozott eszközökön található, a szolgáltatással kapcsolatos állományok meglétének vizsgálata, illetve a kapcsolódó konfigurációs állományok ellenőrzése. Adott esetben klaszter rendszerek esetén a *master*, tehát a szolgáltatás szempontjából központi eszköznek számító virtuális gépen, ha rendelkezésre áll olyan állomány, amely a szolgáltatáshoz kapcsolódó alárendelt, tehát úgynevezett *worker* állomások listáját tartalmazza akkor szükséges ellenőrizni, hogy minden ilyen alárendelt állomás szerepel-e a listán. De ellenőrizni kell többek között a kialakított virtuális eszköz nevét, ip címét, egyéb azonosítóit melyekből kiderül, hogy a szolgáltatásban a megfelelő, neki szánt pozíciót foglalja el.

## **Funkcionális teszt**

Miután megbizonyosodtam arról, hogy a megfelelő állományok kialakításra kerültek a céleszközökön, illetve a konfigurációs fájlok helyesen kerültek kialakításra, legutolsó lépésben szükséges a szolgáltatás funkcionalitásának tesztelése. Ennek ellenőrzéséhez manuálisan el fogok végezni pár tesztet, hogy pontosan tudjam, hogy a szolgáltatás adott bemenetek esetén milyen kimenetekkel kell szolgáljon. Így lesz releváns adatom arról, hogy pontosan milyen kimeneti értékeket kell biztosítson a rendszer az automatikus létrehozást követően is.

A funkcionális teszteléseket lehetőségekhez mérten a szolgáltatásokhoz tartozó API-n keresztül tervezem elvégezni, amennyiben a szolgáltatás rendelkezik ilyen lehetőséggel.

# Rendszerterv

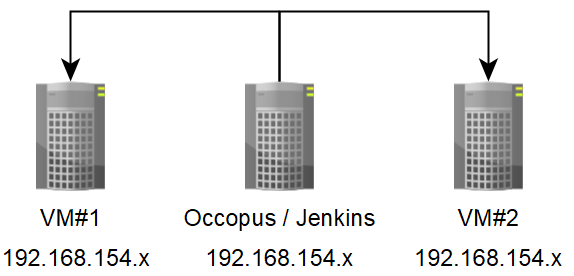
Azt követően, hogy ismertettem a dolgozatom elkészítéséhez szükséges eszközöket, a következő lépés ezek kialakításának meghatározása. A lehetőségeimhez mérten a konfigurációkat, illetve a lépéseket ebben a fejezetben fogalmazom meg, de a feladat jellegéből adódóan ezek egy része a kialakítás során elképzelhető, hogy változni fog.

## **Felhő infrastruktúra**

A dolgozatom elkészítése során szükséges valamilyen felhő infrastruktúra használata, mivel ezen fogom létrehozni a virtuális gépeket, melyekből kettőt automatikusan fogok kialakítani, egy pedig az orkesztrációt, illetve a tesztelési folyamatot fogja ellátni.

Erre a célra OpenNebula felhőszolgáltatást fogok használni, melyet a SZTAKI biztosít számomra a megfelelő hozzáférési jogosultságokkal. A felhőszolgáltatások egyik fő jellemzője, hogy az egyes felhasználókhoz hozzá lehet rendelni az elérhető erőforrások mennyiségét, így ezek pontos megfogalmazására is szükség van.

Elengedhetetlen, hogy legalább három virtuális gépet ki tudjak alakítani, hogy a fentebb említett infrastruktúra felépíthető legyen. Ezekhez a virtuális gépekhez egyenként legalább 4Gb memória, 2 virtuális processzor (vCPU), 20Gb tárterület, illetve egy virtuális hálózat szükséges. Fontos, hogy a kialakított virtuális gépek ugyan abban a hálózatban legyenek, így biztosított közöttük a kommunikáció.



.. ábra: Virtuális gépek kapcsolata a felhőben

Ezeknek a virtuális gépeknek nem fix az IP címük, mivel a létrehozásuk során automatikusan kapnak egy elérhető címet a hálózatban. Így igyekeztem a lehető legpontosabban jelölni ezeket az ábrán.

## **Kapcsolódás**

Ahhoz, hogy ezekre a virtuálisan létrehozott eszközökre kapcsolódni tudjak, és használni tudjam őket, szükséges a megfelelő hálózattal egy titkosított kapcsolatot kialakítanom. Erre a célra a felhő hozzáférés mellett kaptam egy OpenVPN hozzáférést is, melyhez a szükséges konfigurációs fájlokat a saját eszközöm VPN kliens szoftverének a config mappájában helyeztem el.

Ezzel a lépéssel a kapcsolatot ki tudom építeni a saját eszközöm és a felhőszolgáltatásban létrehozott eszközeim között, viszont a biztonsági előírások miatt a jelszavas belépés ezekre a virtuális gépekre nem lehetséges. Az egyetlen elfogadott módszer a publikus és privát kulcsok használata. Ebben az esetben a publikus kulcsom feltöltésre kerül a létrehozott eszközökre, és csak a hozzá tartozó privát kulcs birtoklásával lehet bejelentkezni. Ahhoz, hogy ezt a lehetőséget használni tudjam generálnom kell egy kulcspárt, amelynek publikus kulcsát az OpenNebula felületén a profilom beállításai alatt meg kell adjak mivel ezt a kulcsot fogja a létrehozott virtuális gépemre automatikusan felmásolni. A privát kulcsomat pedig a saját eszközömön a C:\Felhasználók\<felhasználónév>\.ssh mappába kell, elhelyezzek. A kulcspár generálása Windows esetén egy egyszerű parancssorból lehetséges, az „*ssh-keygen*” parancs kiadásával.

## **Operációs rendszer**

A szükséges virtuális gépek alapjául az Ubuntu 18.04 LTS operációs rendszert választottam. Azért erre esett a döntésem, mert ez az egyetlen operációs rendszer, melyre az Occopus szoftver hivatalosan tesztelve lett, illetve a Jenkins is megfelelően támogatja. Továbbá a nevében szereplő LTS pedig az angol Long Term Support rövidítése, tehát hosszú távú támogatás tartozik hozzá, amely 2023 áprilisában jár le.

Azon virtuális gép esetén, amely az orkesztrálás illetve a tesztelés folyamatát végzi fontos hogy a legfrissebb biztonsági javítások legyenek telepítve, hiszen tulajdonképpen ez lesz a központi eszköz a kialakítani kívánt rendszerben.

Korábban, amikor a szükséges allokált erőforrások mennyiségét említettem, ott ennek a rendszernek az igényeit is számításba vettem.

## **Szoftverek**

A dolgozatom kialakításának szempontjából két elengedhetetlen szoftver az Occopus, illetve a Jenkins. Ezek közül az első egy felhő orkesztrátor, a második pedig tulajdonképpen a tesztelést, és automatizálást végző eszköz. Funkcionalitásukat tekintve a főbb elvárások a következők:

|  |  |
| --- | --- |
| Szoftver neve | Elvárás |
| Jenkins | Teszteseket végrehajtása |
| Teszt eredmények generálása |
| Folyamat automatizálás |
| Értesítés az eredményekről |
| Occopus | Infrastruktúra felépítése |
| Infrastruktúra lebontása |

5.4. táblázat: A választott szoftverekkel szemben támasztott elvárások

### **Occopus**

Elsőként az Occopus rendszer telepítését tervezem végrehajtani, aminek működését a telepítést követően manuálisan tesztelnék. Így, ha a telepítés során bármilyen hiba keletkezik, akkor pontosan fogom tudni, hogy az orkesztrátor telepítésével volt probléma és nem a két környezet kommunikációja során.

Az Occopus telepítéséhez szükséges erőforrás igényeket tekintve a hivatalos dokumentációk között nem találtam ajánlást, viszont mivel a rendszer nem rendelkezik grafikus felülettel, úgy gondolom a virtuális gép létrehozásánál megadott erőforrás mennyiségek, elegendőek lesznek. A szoftver telepítése során pedig a hivatalos telepítési útmutatót követve fogok eljárni. [8]

A telepítés végeztével ellenőrizni fogom az Occopus helyes működését, hogy meggyőződhessek a telepítés megfelelő végrehajtásáról. Ehhez a szoftver dokumentációjában található „*helloworld*” környezet kialakítását tervezem. Ez bőven elegendő lesz arra, hogy az orkesztrátor helyességéről megbizonyosodjak. Hiszen az Occopus alap funkcióit a „*helloworld*” környezet kialakítása is igényli.

Ahogyan dolgozatom korábbi fejezeteiben is említettem, ahhoz, hogy az orkesztrátorral virtuális infrastruktúrákat tudjak kialakítani szükséges összegyűjtenem bizonyos API specifikus adatokat. Egyrészt szükségesek felhőszolgáltató által nyújtott EC2 interfész által definiált adatok, másrészt az API authentikációs adatokra is szükségem lesz.

Az általam használt EC2 API adatai a következők:

|  |  |
| --- | --- |
| Paraméter | Érték |
| endpoint | https://opennebula.lpds.sztaki.hu:4567 |
| regionname | root |
| image\_id | ami-00000526 (Ubuntu 18.04 LTS) |
| instance\_type | t1.small |

5.4.1. táblázat: A használt felhő interfész API adatai

Ahhoz, hogy az Occopus authentikálni tudja magát az OpenNebula szolgáltatásában, szükséges az authentikációs fájlban megadni a kapcsolódáshoz szükséges adatokat. Ezt az auth\_data.yaml fájlban kell megtenni, mégpedig az EC2 interfészre vonatkozó szakaszban. Ebben az esetben az *access\_key* az OpenNebula szolgáltatásán regisztrált felhasználóm neve, a *secret\_key* pedig az ehhez a felhasználóhoz tartozó jelszavam sha256 kódolt megfelelője. Ezt egy Ubuntu rendszeren könnyedén el lehet kódolni az „echo -n <jelszó> | openssl sha1” paranccsal.

### **Jenkins**

Következő lépésként a Jenkins kialakítását tervezem megvalósítani, melyet ugyan arra a rendszerre fogok telepíteni, mint amelyikre az Occopust is. Így a két szoftver közti kommunikáció is egyszerűbb lesz. Akárcsak az Occopus rendszer esetén ennek a szoftvernek is belekalkuláltam az erőforrás igényeit a szükséges mennyiségbe. A hivatalos leírás szerint a Jenkinsnek legalább 256Mb memóriára és 1Gb tárterületre van szüksége. [9]

## **Tesztelési környezet**

A telepítést követően a Jenkins grafikus felülete egy egyszerű böngészőn keresztül a virtuális gép IP címét és a 8080-as portot használva elérhető lesz. A felületen első lépésként aktiválni kell a szoftvert, ehhez pedig a szükséges kódot a „cat /var/jenkins\_home/secrets/initialAdminPassword” parancs virtuális gépen történő kiadásával fogom elérni. Ezzel a szoftver aktiválásra kerül, és létrehozhatok egy felhasználót, amely segítségével a tesztelési folyamatot fogom elkészíteni.

A tesztelési folyamat kialakítását alapvetően a Jenkins *pipeline* funkciójával tervezem, mivel így az igényeimnek megfelelően el tudok különíteni állapotokat. Viszont a számomra szükséges lehetőségeket a Jenkins *freestyle* tesztelési lehetősége is megfelelően biztosítaná. Azt, hogy pontosan melyik lesz a legmegfelelőbb módszer, azt a kialakításuk során tudom csak megvizsgálni.

### **Automatizmus**

A tesztek automatikus elvégzését periodikusan megadott időközönként tervezem végrehajtani. Ehhez a Jenkins szoftverben beépített lehetőséget fogom használni.

Mivel a környezet kiépítése is része a tesztelési folyamatnak így a Jenkinsen keresztül fogom meghívni az Occopus parancsait. Ehhez a virtuális gépen szükséges jogokat biztosítani a Jenkins számára ahhoz, hogy a parancsokat végre tudja hajtani.

### **Teszteléshez szükséges kapcsolat**

Az Occopussal létrehozott infrastruktúrákat lehetőség szerint minél több módszerrel szeretném tesztelni, de minden teszteléshez szükséges a létrehozott környezet IP címe. Mivel a kiépítés és a tesztelés is automatikusan történik így fontos, hogy a kiépítést követő tesztelési szakaszban ismerjem a megfelelő IP címet. Ezt úgy tervezem megvalósítani, hogy az Occopus által előállított napló fájlok vizsgálatához készítek egy *bash* *scriptet*, ami ezekből a napló fájlokból kinyeri a szükséges címet. Ezt követően *ssh* kapcsolatot létesít a kialakított eszközzel, és ott a környezetnek megfelelően előre elkészített parancsaimat végrehajtja.

### **Tesztelt szolgáltatások**

A két kialakítandó környezet az Apache Spark klaszter, illetve a Tensorflow. Ezek kiépítéséhez a szükséges leíró fájlok, és kontextualizációs állományok elérhetőek az Occopus weboldalán, melyeket használni fogok.

Ezen szolgáltatások teszteléséhez szükséges lépések kialakításához a hozzájuk tartozó kontextualizációs állományt fogom vizsgálni. Ezekből a fájlokból egyértelműen kiderül majd, hogy adott teszt esetén milyen konfiguráció az elvárt, és milyen fájlok megléte szükséges.

### **Értesítés**

A tesztelések végeztével pedig az eredményekről szükséges értesítést küldeni a kompetens személyeknek. Ennek kialakítását egy Jenkins specifikus kiegészítő segítségével tervezem megvalósítani. Ez az „*email extension plugin*”, mely ingyenesen elérhető a feltelepített Jenkins rendszer beállításaiból a kiegészítő kezelési lehetőséget választva. Ennek segítségével beállíthatok levelezési listát, alapértelmezett tárgyat, alapértelmezett szöveget, és a tesztelés folyamata közben is tudok információkat küldeni a megadott címekre. Tehát tudok esetenként a tesztelést megelőzően is értesítést küldeni, illetve annak végeztével is. Azonban én csak a tesztelés eredményéről fogok emailt küldeni egy rögzített levelezési listára. A levélben terveim szerint szerepelni fog a tesztelés neve, az utolsó futás időpontja, illetve a teszt eredménye. [10]

# Környezet kialakítása

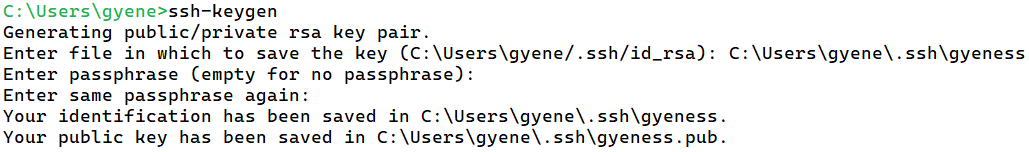
A rendszer terv meglétét követően a szakdolgozatom célját biztosító környezet kialakításának, és konfigurálásának tényleges lépései, és szükséges beállításai következhetnek. A megfelelő reprodukálhatóság érdekében az egyes szoftverek, összetevők telepítését, és beállításait a kellő alapossággal fogom dokumentálni, az ehhez szükséges képernyő képekkel, parancsokkal illetve szöveges magyarázatokkal.

## **Előkészületek**

### **RSA kulcs generálás**

A környezet kialakításához szükséges legelső lépés az RSA kulcspár generálása, mely segítségével fogok csatlakozni ssh kapcsolaton keresztül a virtuális eszközeimhez. Ezen kulcspár generálása történhet Linux, illetve Windows alapú operációs rendszeren is, sőt akár egyéb erre a célra szolgáló szoftverek használatával. A kulcspár generálását én egy Windows operációs rendszerrel rendelkező eszközön végeztem el, mert munkám során a legtöbb esetben ilyen operációs rendszerrel rendelkező eszközökről fogok kapcsolatot kezdeményezni. Így biztosan a megfelelő formátumú az előállított kulcspár.

A generálást tehát egy Windows operációs rendszerrel rendelkező eszközre bejelentkezve kezdeményeztem az ssh-keygen parancs használatával egy egyszerű parancssor segítségével. A parancs kiadását követően megadható a kulcspár neve arra az esetre, ha több különböző kulcspárra lenne szükségem. Illetve megadható az úgynevezett *passphrase* mely tulajdonképpen egy jelszó, amelyet a kulcspárral történő csatlakozást követően meg kell adnom, így az ssh kapcsolat még biztonságosabb lesz. Az utóbbi lehetőséget nem kívánom a munkám során használni, mert csak belső hálózatból lesznek elérhetőek a virtuális gépek, így nincsenek kitéve külső veszélyforrásoknak.



.. ábra: RSA kulcspár generálása Windows operációs rendszeren

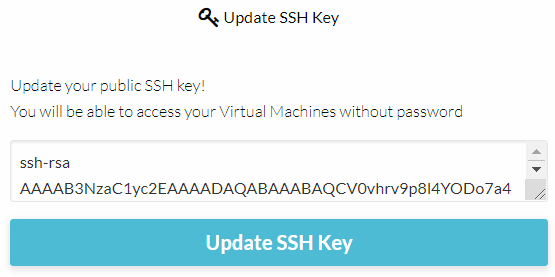
Az így generált kulcspár a létrehozásnál megadott elérési útvonalon megtalálható. Abban az esetben, ha a kulcspár az adott felhasználó /.ssh könyvtárában található, akkor egy egyszerű parancssorból indított ssh kapcsolódás során lehet használni a generált privát kulcsot.

A generált kulcspár egyik állománya a publikus, a másik pedig a privát kulcs. A privát kulcs szükséges ahhoz, hogy tudjam magamat azonosítani a létrehozott virtuális eszközökön, a publikus kulcs tartalmát pedig az OpenNebula felületén a felhasználói fiókom beállításainál meg kell adnom. Így ha létrehozok egy virtuális gépet, akkor alapértelmezetten az ehhez a publikus kulcshoz tartozó privát kulccsal fogom tudni azonosítani magamat.

### **Opennebula**

A kulcsok generálását követően el kell végezni az OpenNebula felhőn bizonyos beállításokat, illetve létre kell hozni egy virtuális eszközt Ubuntu operációs rendszerrel, mely eszközre fogom feltelepíteni az Occopus illetve Jenkins szoftvereket.

Első lépésként tehát az OpenNebula grafikus felületére bejelentkezve a felhasználó nevemre kattintva, a beállítások fülön tudom megadni a generált kulcspárhoz tartozó publikus kulcs tartalmát. Ehhez a generált publikus kulcsot egy tetszőleges szövegszerkesztő programmal megnyitottam a generálás helyének megadott célkönyvtárban, és a tartalmát bemásoltam, majd a változtatásokat elmentettem.



.. ábra: Publikus kulcs módosítása az OpenNebula grafikus felületén

Következő lépés a szükséges virtuális eszköz kialakítása, melyet az OpenNebula grafikus felületén a *Virtual Machines* menüpontjának, *create* lehetőségével tudok megtenni. A virtuális gépek létrehozása során szükséges hogy legyen legalább egy meghatározott *security group* illetve egy *virtual network*. Azonban ezek létrehozásával nem kell foglalkoznom, mivel a jogosultsági szintemből fakadóan, az alapértelmezett beállításokat tudom használni. A virtuális gép létrehozása során kötelezően meg kellett adnom egy nevet, mellyel könnyedén azonosítani tudom az eszközt, illetve ki kellet választanom az operációs rendszernek megfelelő sablont.

A munkám alapjául az Ubuntu 18.04-es operációs rendszert választottam, melynek megfelelő használatához 4GB memória, illetve 2 virtuális processzor szükséges. Ezen felül a létrehozás során 20GB tárhelyet rendeltem az eszköz mellé, mivel a szükséges szoftverek nem nagyméretűek, így ez elegendő lesz.

Az így létrehozott virtuális gép ezt követően megjelenik a *virtual machines* oldalon megjelenítve a választott nevet, az ip címet, sablont, a tulajdonost és a létrehozás idejét.



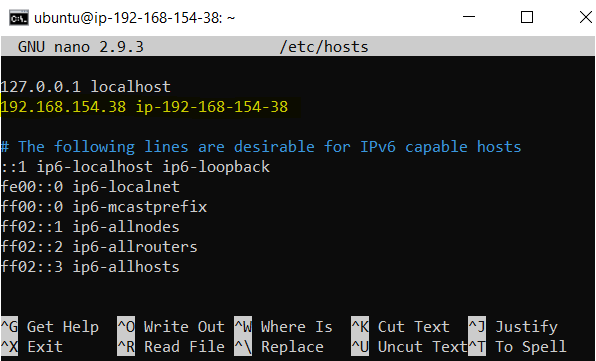
.. ábra: A létrehozott virtuális gép adatai

## **Ubuntu telepítés**

Következő lépésben a képen látható ip cím segítségével ssh kapcsolatot kezdeményezek a létrehozott virtuális géppel az ssh felhasználó@ip parancs használatával parancssoron keresztül. Ebben az esetben a felhasználó az „ubuntu” lesz, mivel az általam használt sablon konfigurációjában ez szerepel, mint alapértelmezett felhasználónév, az ip cím pedig a 192.168.154.38 lesz, ahogyan az a fenti ábrán is látható.

A létrehozott virtuális gépre történő első bejelentkezés során érdemes frissíteni az operációs rendszert a *sudo apt update*, illetve a *sudo apt upgrade* parancsok használatával. Az első parancs letölti a szükséges frissítendő fájlokat a második parancs pedig elvégzi a frissítést. Amennyiben a második parancs egy nem elérhető folyamatra hivatkozva nem tudja elvégezni a frissítést akkor érdemes várni, hiszen a rendszer első indítása során számos háttér folyamat futhat, így ezeknek a folyamatoknak a további lépések előtt véget kell érnie.

Az egyetlen szükséges lépés, amit ilyenkor még érdemes elvégezni az a hosts állomány módosítása, a /etc könyvtáron belül. Erre azért lesz szükség, hogy a későbbiekben a sudo parancsok kiadása során ne keletkezzenek hibaüzenetek. A módosítás során fel kell venni az eszköz nevét, illetve az ip címét a már említett állományba az alábbi képen látható formátumban.



.. ábra: /etc/hosts állomány módosítása

Az operációs rendszer telepítésével kapcsolatosan további lépés nincsen, az esetleges egyéb konfigurációs módosításokat majd az előfordulásuk idejében, és helyén fogom megemlíteni.

## **Jenkins telepítés**

A következő lépés, amelyet a megvalósítás során végre kell hajtani az a Jenkins szoftver telepítése, amely segítségével kialakíthatók a tesztelési folyamatok.

A Jenkins szoftver telepítése során a hivatalos telepítési dokumentáció segítségével végzem el a szükséges lépéseket, így biztosan a megfelelő módon fogom tudni feltelepíteni a szoftvert.

A dokumentáció szerint a szoftver megfelelő működéséhez szükséges telepíteni a Java 1.8 keretrendszert, így első lépésben ennek a telepítését fogom elvégezni. Ehhez a következő parancsok végrehajtása szükséges.

sudo apt update  
sudo apt install openjdk-8-jdk  
sudo apt install openjdk-8-jdk  
java -version

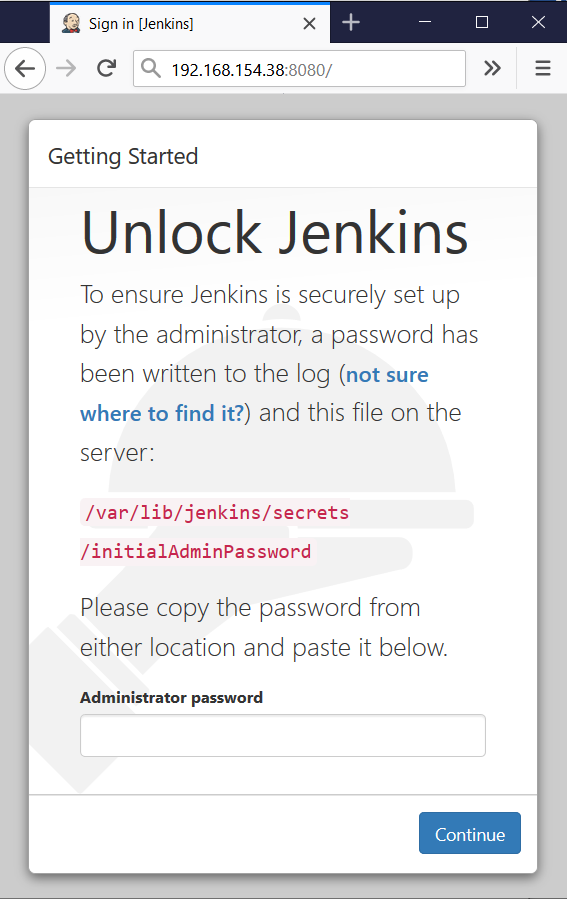
Ezen parancsok először is megvizsgálják, hogy van-e további elérhető frissítés az Ubuntu operációs rendszerhez, amelyet esetleg nem sikerült telepíteni. Majd ezt követően a két azonos parancs szolgál a Java keretrendszer telepítésére és egyben ellenőrzésére is. Ezen parancsok végrehajtása során először megkezdődik a Java telepítése, majd ennek végeztével ezt a parancsot érdemes másodszorra is lefuttatni annak ellenőrzése céljából hogy minden megfelelően feltelepült-e, és nem maradt ki egyetlen telepítendő csomag sem. A legutolsó parancs pedig csak a Java verzió ellenőrzésére szolgál, melynek 1.8-nak kell lennie.

Miután az előkészületek megvannak, már minden feltétel adott a Jenkins szoftvert telepítéséhez, ehhez a következő parancsok sorrend helyes végrehajtására van szükség.

wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -  
sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'  
sudo apt-get update  
sudo apt-get install jenkins

 Az első parancs segítségével be kell importálni a szoftver telepítéséhez szükséges kulcsot, a második parancs segítségével pedig hozzá kell adni az erőforrás listához jenkins.list néven. Ezt követően szükséges megvizsgálni, hogy a frissített erőforrás listának megfelelően melyek a legfrissebb elérhető csomagok, melyek telepíthetők. Természetesen az előző parancs következtében ebben az esetben a Jenkins szoftver legfrissebb csomagjait is megvizsgálja, és megkeresi az operációs rendszer. Így elvégezhető a telepítés a lista legvégén található parancs segítségével. A parancs kiadását követően az operációs rendszer letölti a listának megfelelő elérhető állományokat, és telepíti a szoftvert.

A telepítés végeztével a Jenkins szoftver grafikus felülete egy böngészőn keresztül elérhető a virtuális gép ip címét használva, a 8080-as porton keresztül. Jelen esetben tehát a 192.168.154.38:8080-as címen. A grafikus felület első meglátogatása során aktiválni kell a Jenkins szoftvert a használat előtt. Ezt a lépést egyszer szükséges elvégezni, a továbbiakban a szoftver már aktiválva lesz.



.. ábra: Jenkins grafikus felülete az első indítást követően

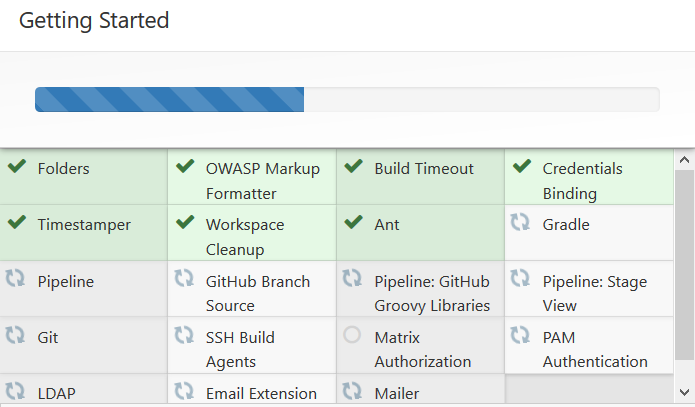
Ahogyan az a képen is látszik az aktiválás folyamata tulajdonképpen annyit jelent, hogy a megadott útvonalon lévő állományból ki kell olvasni a véletlenszerűen generált jelszót, és be kell írni a felületen. Ennek a jelszónak a kiolvasásához magasabb szintű jogosultság szükséges tehát a sudo parancsot is alkalmazni kell. A kiolvasáshoz pedig elég egy egyszerű cat parancs végrehajtása.



.. ábra: Jenkins aktiválásához szükséges jelszó kiolvasása

Az így kapott kódot egyszerűen be kell másolni a böngészőben korábban megnyitott grafikus felületen, ezzel aktiválva a szoftvert.

Ezt követően a felületen a következő ablakban a szoftver lehetőséget kínál arra, hogy a telepítés során csak azokat a bővítményeket telepítsem, amelyeket szeretném, és a munkám során szükségesek. Viszont én az ajánlott telepítéssel folytatom, és a későbbiekben az esetlegesen még szükséges bővítmények telepítését a felületre bejelentkezve fogom elvégezni.



.. ábra: Jenkins bővítmények listája az ajánlott telepítés során

A képen látható bővítményeket telepíti a szoftver az ajánlott telepítés kiválasztását követően. Ebben számos olyan bővítmény megtalálható, amelyre a munkám során szükségem lehet, ilyenek többek között a Pipeline, Email Extension és a Git is.

A bővítmények telepítése egy pár percet igénylő folyamat, melynek végeztével a következő lépés egy adminisztrátor felhasználó létrehozása lesz. Ez a lépés nem kötelező, használható a szoftver az alapértelmezett admin felhasználóval is, azonban én létrehoztam egy saját felhasználót, amellyel kezelni fogom a szoftvert.

A telepítés utolsó lépésenként pedig a Jenkins szoftver grafikus felületének elérését lehet módosítani, én ezt alapértelmezetten a felajánlott formában hagyom. Ezt az elérést a Jenkins úgynevezett környezeti változóinak segítségével fel lehet használni az emailek küldése során, vagy bármilyen feladat végrehajtása során. Ezért alap esetben ezt célszerű arra a domain névre állítani, amit a felhasználók használni fognak a szoftver grafikus felületének eléréséhez. Azonban mivel a szoftvert munkám során egyedül fogom használni, így ez alap érték számomra teljesen megfelelő.

A Jenkins szoftver telepítését követően egy pipeline script segítségével megvizsgálom, hogy milyen felhasználóval hajtja végre a megadott scripteket a szoftver. Ezt a következő shell parancs beágyazásával tudom megtenni.

pipeline {  
 agent {label 'master'}  
 stages {  
 stage('usercheck') {  
 steps {  
 sh "whoami"  
 }  
 }  
 }  
}

Ez a kód tulajdonképpen a „*whoami*” shell parancs beágyazását jelenti a *pipeline* kódba. Ennek a parancsnak a segítségével tudom megállapítani, hogy éppen melyik felhasználó hajtaja végre az adott *pipeline* kódban megfogalmazott lépéseket. Ennek meghatározása azért nagyon fontos, mert ezzel a felhasználóval kell majd futtatnom az Occopus specifikus parancsokat, éppen ezért az adott parancsokhoz biztosítani kell a megfelelő jogosultságokat a futtatást végző felhasználó számára.



.. ábra: Jenkins pipeline scriptet futtató felhasználó

A képen látható hogy a *pipeline* kódok futtatását a „jenkins” nevű felhasználó segítségével végzi a program, így éppen ezért ennek a felhasználónak biztosítani kell a hozzáféréseket majd az Occopus szoftver megfelelő parancsaihoz.

A Jenkins szoftver telepítése ezzel befejeződött, és a grafikus felület is a kezdőlapra irányított, ahol egyéb beállítást jelenleg nem fogok elvégezni. A tesztek és a folyamatok kialakítása előtt szükséges az Occopus szoftver telepítése, és konfigurálása.

## **Occopus telepítés**

A szakdolgozatommal kapcsolatos munkához szükséges környezet kialakításának a következő lépése az Occopus 1.8 szoftver telepítése. A telepítést ebben az esetben részben a hivatalos telepítési útmutató szerint fogom elvégezni. A hivatalos útmutatótól adott pontokban az általam végzett telepítés eltér, hiszen én a Jenkins szoftveren keresztül szeretném használni az Occopus szoftvert ennek megfelelően a szoftver telepítés helye, illetve egyes jogosultságok eltérnek, ezeket megfelelően dokumentálni is fogom.

Első lépésként a dolgozatom irodalomjegyzékében szereplő hivatalos Occopus 1.8 telepítési útmutató alapján a szükséges függőségek telepítését fogom elvégezni. Ezen függőségek szükségesek ahhoz, hogy az Occopus 1.8 szoftver megfelelően működjön.

sudo apt install -y python3-pip  
sudo apt install -y virtualenv  
sudo apt install -y redis-server  
sudo apt install -y libssl-dev

Miután az előkövetelmények telepítése véget ért, át kell váltanom a Jenkins szoftver által használt felhasználóra, amely a „jenkins” felhasználó. Ezt a Jenkins szoftver telepítését követően végrehajtott pipeline kód alapján tudom. Az átváltást pedig a következő parancs segítségével tudom megtenni.

sudo su jenkins

Azt követően, hogy a parancsot végrehajtottam a parancssorban látható felhasználó név is megváltozott, illetve hogy biztosan meggyőződjek arról, hogy sikerült felhasználót váltanom, lefuttatom a *whoami* parancsot, akárcsak a Jenkins pipeline scriptben.

Azok után, hogy meggyőződtem a sikeres felhasználó váltásról visszatérek a hivatalos telepítési dokumentációhoz, és folytatom a telepítést az abban megfogalmazott lépések szerint. Tehát létrehozom a szükséges virtuális környezetet, és elvgézem a szoftver aktiválását a következő parancsokkal.

virtualenv -p python3 $HOME/occopus  
source $HOME/occopus/bin/activate

 A következő lépés az Occopus 1.8 szoftver csomagjainak, és állományainak letöltése, illetve ezek telepítése az operációsrendszerre. A szoftver telepítése a következő parancs végrehajtásával elvégezhető.

pip install --no-index --find-links https://pip3.lpds.sztaki.hu/packages OCCO\_API

Ezzel a szoftver telepítése tulajdonképpen el is készült, azonban ahhoz hogy megfelelően együtt működjön az OpenNebula felhőszolgáltatással, szükséges a szoftver forráskódján elvégezni egy apró módosítást. Azt hogy pontosan mit kell módosítanom, a szoftver fejlesztőinek elmondása alapján tudom. Ennek alapján a következő elérési úton található állomány módosítása szükséges.

/var/lib/jenkins/occopus/lib/python3.6/site-packages/occo/plugins/resourcehandler/ec2.py

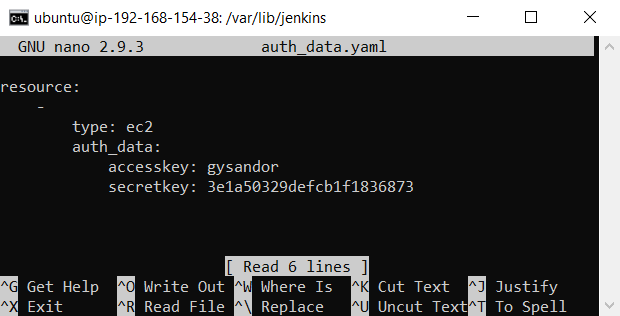
A fentemlített elérési útvonalon található ec2.py állományból el kell távolítani az alábbi kódrészletet. Majd el kell menteni a módosított állományt. Ennek az apró módosításnak a végrehajtásával az Occopus szoftver már céljának megfelelően fog működni az infrastruktúrák kiépítése, és lebontása során.

self.conn.stop\_instances(instance\_ids=vm\_ids, force=True)

Következő lépésként az Occopus működéséhez szükséges autentikációs és konfigurációs állományok letöltése, illetve konfigurálása következik. Ezen állományok letöltése szintén a telepítési útmutatóban található parancsok segítésével történik.

mkdir -p $HOME/.occopus  
curl https://raw.githubusercontent.com/occopus/docs/master/tutorials/.occopus/occopus\_config.yaml -o $HOME/.occopus/occopus\_config.yaml  
curl https://raw.githubusercontent.com/occopus/docs/master/tutorials/.occopus/auth\_data.yaml -o $HOME/.occopus/auth\_data.yaml

Ezek a parancsok csak létrehozzák a szükséges mappát, illetve letöltik a minta állományokat a megadott elérési útvonalról, azonban az autentikációs fájlt konfigurálni kell a saját adataimnak megfelelően.



.. ábra: Az autentikációs fájl kialakítása

Ebben az esetben az „*accesskey*” paraméter az OpenNebula felhőszolgáltatásban használt felhasználónevem, a „*secretkey*” pedig az ugyan ebben a szolgáltatásban használt jelszavam sha 1 kódolt megfelelője. A „*secretkey”* kódjának elkészítése Ubuntu operációs rendszeren a következő parancs segítségével megtehető.

echo -n "jelszó" | openssl sha1

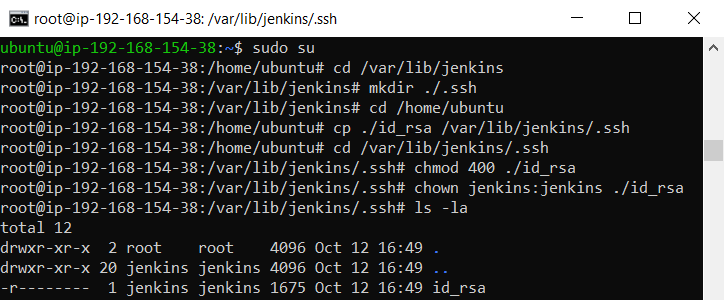
Mivel az Occopus szoftver telepítése nem pontosan a megadott lépések sorozata szerint történt így szükséges ellenőrizni, hogy a keletkezett állományok megfelelő jogosultságokkal rendelkeznek-e, illetve minden könyvtár és állomány a jenkins felhasználó tulajdona. Ezek ellenőrzéséhez először is a jenkins felhasználó saját úgynevezett *home* könyvtárába szükséges navigálni, majd ott listázni az egyes alkönyvtárakat, és állományokat.

cd $HOME  
ls –la

Fontos, hogy e parancsokat továbbra is jenkins felhasználóként kell végrehajtani. Az így kapott listában található könyvtárak esetén kiemelten fontos hogy a tulajdonos felhasználó a jenkins felhasználó, és a tulajdonos csoport pedig a jenkins csoport legyen. Hiszen az egyes összetevők megfelelő elérése a tesztelések során csak ebben az esetben biztosított. Azonban tesztelések során szükséges az Occopus specifikus parancsok hozzáférhetősége is, így a /var/lib/jenkins/occopus/ könyvtárból a bin, illetve a lib mappákat át kell másolni a /var/lib/jenkins/.local/ mappába.

## **Jenkins ssh kapcsolódás konfigurálása**

A Jenkins illetve az Occopus szoftver telepítését követően biztosítani kell, hogy a tesztelések során kialakított infrastruktúrákra a Jenkins szoftver ssh kapcsolatot tudjon kezdeményezni. A kapcsolat kialakítását követően ugyanis az egyes tesztesetek szempontjából fontos, hogy ellenőrizni tudjam a szükséges állományok és könyvtárak meglétét. Ennek érdekében készítettem egy *bash* scriptet, amely az éppen futó pipeline teszt napló állományaiból kiolvassa a létrehozott infrastruktúra ip címét, és kezdeményezi az ssh kapcsolatot. Azonban a kulcspárral történő hitelesítési eljárás miatt szükséges hogy a saját privát kulcsomat a jenkins felhasználó számára elérhetővé tegyem, illetve a kulcsot a megfelelő jogosultsági szinttel lássam el. Ehhez a saját eszközömről, amelyről az ssh kapcsolatot kezdeményezem, a privát kulcsomat WinScp program segítségével felmásoltam a virtuális eszközömre, majd onnan áthelyeztem a jenkins felhasználó *home* könyvtárában általam létrehozott „.ssh” alkönyvtárba. A jenkins felhasználó számára átadtam a kulcshoz tartozó tulajdoni jogot, és beállítottam a kulcs jogosultságait az alábbi parancsok használatával.



.. ábra: SSH kulcs átadása a jenkins felhasználó számára

A képen látható parancsok sorrendhelyes végrehajtásával létrehoztam a szükséges ssh könyvtárat, elvégeztem a privát kulcs másolását, illetve beállítottam a jogosultságokat. Legutolsó lépésben pedig ellenőriztem az említett könyvtár tartalmát, és a benne lévő állományok jogosultságát. Így látható tehát hogy az adott kulcs használatához csak az állomány tulajdonosaként megjelölt felhasználó férhet hozzá, és ez a felhasználó is csak olvasási jogosultságokkal rendelkezik, így növelve a rendszer biztonságát.

# Jenkins projektek kialakítása

A jenkins projektek kialakítása a dolgozatom szempontjából egy nagyon lényeges és gondos odafigyelést igénylő feladat, mivel ezek a projektek lesznek felelősek a dolgozat teljes folyamatának végrehajtásáért. A folyamatok egészét, mint korábban említettem pipeline scriptek segítségével definiálom, és a scripteket kisebb egységekre, úgynevezett állapotokra bontva fogom kezelni, így elkülönítve egymástól a folyamat fontosabb lépéseit.

A szakdolgozatom során két folyamatot kell kialakítanom, mivel két különböző környezet automatikus tesztelését végzem el, mely tesztelések a környezetektől függő specifikus lépéseket igényelnek. A folyamatok a szükséges friss leírók, és állományok automatikus letöltésétől, a környezet kialakításán keresztül annak tesztelésén és „lebontásán” át egészen a teszt eredményéről szóló email küldéséig tartanak.

## **Általános jenkins folyamat**

Dolgozatom jelen alfejezetében az automatizáláshoz szükséges folyamat lépéseinek kialakítását részletezem, mely lépések a folyamat automatizált indításától egészen a tesztelt környezet törléséig, és az értesítések elküldéséig tartanak. A folyamatok bizonyos része a két tesztelt környezet esetén nagy részben megegyezik, mivel az állományok letöltése, a környezet felépítése, illetve törlése ugyan azon parancsok végrehajtásával lehetséges. Így ezeket a parancsokat a dolgozatomban az Apache Spark klaszter folyamatának kialakításával mutatom be. Tehát a Tensorflow környezetre jellemző egyedi lépéseket, illetve egyedi folyamatokat a következő alfejezetben fogom részletezni.

### **Github repositroy trigger**

**Ez a rész még bőven nincsen meg, mert valamiért a git triggerem nem akar működni. Valszeg github webhook-ot kell konfigurálni, ahhoz viszont valszeg Márk segítsége is kell, mert ott authentikálni is kellene. – Ez legyen meg kövi héten, és még olvass utána!**

### **Leíró fájlok letöltése**

Az automatizált tesztelés első valódi lépése a szükséges projekt leíró állományainak letöltése, és a megfelelő helyre történő elhelyezése, illetve konfigurálása. A konfigurálást az egyes projektek esetén korábban elvégeztem és az érintett állományokat elhelyeztem egy külön könyvtárban, és a letöltést követően csak ezt az állományt másolom át a frissen letöltött projekt kicsomagolt könyvtárába. Mivel ez az állomány csak a felhőszolgáltatásban használt autentikációs adataimat tartalmazza, illetve a szükséges API adatokat, így ha a leíró állományok technikai része frissül, azokat nem befolyásolja ennek az állománynak a cseréje. A felhasználói azonosító adatinak megadásáról, illetve az API adatok megfelelő kitöltéséről korábban, a leíró állományok esetén már írtam. Annak megfelelően kell az állományt kiegészíteni és módosítani.

Ebben a fejezetben azonban az állományok letöltéséhez, illetve a kitöltött konfigurációs állomány elhelyezéséhez szükséges script felépítését, és kialakítását ismertetem. Tulajdonképpen a letöltés és a konfigurációs állomány másolása ugyan abban a scriptben történik, egyéb szükséges lépésekkel kiegészítve.

#!/bin/bash  
rm -r /var/lib/jenkins/spark-cluster-with-r/  
curl -v https://raw.githubusercontent.com/occopus/docs/master/tutorials/spark-cluster-with-r.tar.gz | tar -xz -C /var/lib/jenkins/  
cp /var/lib/jenkins/node\_config/apache\_spark/node\_definitions.yaml /var/lib/jenkins/spark-cluster-with-r/nodes/node\_definitions.yaml

A scriptben foglalt szükséges lépések legelőször törlik a „spark-cluster-with-r” mappát a jenkins felhasználó „*home*” könyvtárán belül, a benne található összes állománnyal együtt. Ez a lépés azért kiemelten fontos, mert ha a leíró állományok letöltése újra megtörténik, akkor az esetleges hibák elkerülése végett szükséges a korábbi állományok törlése, így lesz a tesztelés is megfelelően alapos, és releváns. Miután a korábbi állományok törlése megtörtént szükséges a friss tömörített állomány letöltése, illetve annak kibontása ugyan arra az elérési útvonalra, amelyről az első lépésben a korábbi állományok törölve lettek. A megegyező elérési út is kiemelt fontosságú ugyan is a következő lépések során az Occopus szoftver hívása során, az ezeken az útvonalakon található állományokra fogok hivatkozni. Majd legutolsó sorban a szükséges konfigurációs fájl megfelelő helyre történő másolása következik, ezzel téve teljessé a leíró állományok frissítését, és felhőszolgáltatáshoz illesztését. A fenti kód ezeket a lépéseket tartalmazza a megfelelő operációsrendszer specifikus parancsok használatának segítségével.

### **Környezet Kiépítése**

A környezet kialakításához szükséges állományok letöltése, és a konfiguráció módosítása után kezdődhet a tesztelendő környezet kialakítása. Az Occopus szoftver telepítése során azért kellett eltérnem az ajánlott telepítési leírástól, hogy ezt a lépést megfelelően végre tudjam hajtani a Jenkins szoftver használatán keresztül. Ugyanis az általam korábban megfogalmazott telepítési lépéssorozat lehetővé teszi, hogy a Jenkins szoftveren keresztül elérjem az Occopus szoftver szükséges és releváns parancsait. A kiépítés során a két Occopus specifikus parancs melyekre szükségem van az „occopus-import” illetve az „occopus-build”. Melyek közül az előbbi végzi el a letöltött állományok, és konfigurációs állományok importálását az Occopusba, a második parancs pedig ezen állományok segítségével megkezdi a leíró fájlokban definiált környezet kialakítását. Ahhoz hogy ezek a parancsok megfelelően működjenek a telepítésnél meghatározott módon kell a megfelelő állományokat a megadott könyvtárba másolni. Ezen parancsok hívása a Jenkins pipeline scriptben az alábbi kódrészleten látható módon néz ki.

stage('import') {  
 steps {  
 sh "/var/lib/jenkins/.local/bin/occopus-import  
 /var/lib/jenkins/spark-cluster-with-r  
 /nodes/node\_definitions.yaml"  
 }  
}  
stage('build') {  
 steps {  
 sh "/var/lib/jenkins/.local/bin/occopus-build  
 /var/lib/jenkins/spark-cluster-with-r/infra-spark- cluster.yaml"  
 }  
}

A környezet kiépítését követően szükséges pár perc várakozás beépítése a folyamatba, ugyanis az újonnan létrehozott Ubuntu operációs rendszerrel rendelkező eszközök esetén pár percig még az inicializálás folyamata zajlik. Tehát az operációs rendszer az első indítás során szükséges módosításokat még el kell, hogy végezze, ennek érdekében egy 15 perces várakozást építettem bele a folyamatba. Ugyanis tapasztalatom szerint ez az idő bőven elegendő ahhoz, hogy a létrehozott Ubuntu eszközök megfelelő módon elvégezzék a szükséges beállításokat. Illetve Apache Spark klaszter esetén az állomások ip címét tartalmazó állományban ennyi idő elteltével már biztosan megjelenik minden *worker* állomás ip címe is.

### **Környezet tesztelésére szolgáló scriptek**

A folyamat szempontjából következő lépés a tesztelések elvégzése, azonban a tesztek működését a teszteléssel kapcsolatos későbbi fejezetben fogom ismertetni, ugyanis az szervesen oda kapcsolódik. Azonban a Jenkins projekt szempontjából ezen a ponton fontos megemlítenem, hogy ezek a tesztek szintén *bash* scriptek használatával kerülnek kialakításra, és csak úgy, mint a leíró állományok letöltésére szolgáló *bash* script esetén, a *pipeline* kódból ezeket közvetlen hívásokkal fogom meghívni. Ezek a scriptek képesek felcsatlakozni az automatikusan kialakított eszközökre ssh kapcsolat segítségével, és ott elvégezni előre megírt utasításokat, melyek eredményéből kiderül, hogy a szolgáltatás megfelelően került-e kialakításra. Ennek ellenőrzése két lépcsőben történik, az első lépés a szükséges konfigurációs fájlok tartalmának összegyűjtése, illetve a szükséges mappák tartalmának listázása. Ezen adatokat szöveges állományokba gyűjtöm azon az eszközön melyekről az ssh kapcsolatot kezdeményezem a kialakított eszközökre. A következő lépésben pedig e szöveges állományokat *bash* scriptek segítségével vizsgálom annak érdekében, hogy biztosan megfelelnek-e a konfigurációk, illetve a könyvtárak az elvárt működés szerinti állapotnak. Hiba esetén pedig a linux rendszeren használt, megfelelő *exit* kódok segítségével megszakítom a *bash* script futását. A scriptek megszakítása hibakódok segítségével nagyon fontos a vizsgálat szempontjából, ugyanis a Jenkins szoftver értelmezni tudja, hogy a script hibával ért e véget. Abban az esetben pedig, ha hiba történt képes megszakítani a folyamat további lépéseit, ezzel pontosan behatárolva az értesített személyek számára a hiba keletkezésének okait.

### **Környezet lebontása és értesítés küldés**

A kialakított környezetre vonatkozó tesztelések elvégzését követően a folyamat során kialakított infrastruktúra lebontása következik, mely a kiépítéshez hasonlóan Occopus specifikus paranccsal történik. A környezet törléséhez az „occopus-destroy” parancs használatára van szükség. Ahhoz hogy a parancs törölni tudja a kialakított környezetet bemeneti paraméterként meg kell adni neki az infrastruktúra azonosítóját. Ez az azonosító a környezet létrehozásakor képződő napló állományban megtalálható. Azonban mivel ez az azonosító minden egyes kialakítás során változik, én pedig a teljes folyamatot automatizálni szeretném ezért készítettem egy bash scriptet. Amely a naplófájlból egy txt állományba átmásolja az azonosítót tartalmazó sort, majd ebből az állományból pedig kiszedi csak az azonosítót. Azért van szükség két lépésre az azonosító kiolvasásához, mert az eredeti napló állományban az infrastruktúra és az eszközazonosítók pontosan ugyan olyan formátummal rendelkeznek, viszont a naplóállományban más-más kontextusban helyezkednek el. Így először a teljes vonatkozó részt tárolom el egy txt állományban majd abból már sokkal egyszerűbb a számomra releváns rész kiírása. Éppen emiatt a plusz lépéssorozat miatt az infrastruktúra törlésének indítása a pipeline scriptben is eltér a kiépítéshez képest.

stage('destroy') {  
 steps {  
 sh "/var/lib/jenkins/Apache\_Spark\_Connect/getinfraid.sh ${env.JOB\_NAME}"  
 }  
}

Ahogyan az a kódrészleten is látható nem közvetlenül az occopus-destroy parancsra hivatkoztam a pipeline scriptben hanem az általam készített scriptre. Ez a script az azonosító kiolvasása után meghívja az occopus specifikus destroy parancsot is a megfelelő paraméter megadásával. Azonban ahhoz hogy a megfelelő környezethez tartozó naplóállományt tudjam vizsgálni a scriptnek szükséges megadnom a pipeline projekt nevét, amelyet paraméterként adok át a bash script számára, egy beépített jenkins környezeti változó segítségével.

#!/bin/bash  
grep -o -a -m 1 -h -r -Eo "Submitted infrastructure: '([A-Za-z0-9]{4,8}\-){4}[A-Za-z0-9]{1,12}'" /var/lib/jenkins/workspace/$1/occopus.log | head -1 > /var/lib/jenkins/Apache\_Spark\_Connect/node\_ids.txt  
grep -Eo '([A-Za-z0-9]{4,8}\-){4}[A-Za-z0-9]{1,12}' /var/lib/jenkins/Apache\_Spark\_Connect/node\_ids.txt | sort | uniq > /var/lib/jenkins/Apache\_Spark\_Connect/infra\_ids.txt  
/var/lib/jenkins/.local/bin/occopus-destroy -i $(cat /var/lib/jenkins/Apache\_Spark\_Connect/infra\_ids.txt)

 A környezet törlését követően már csak az email küldés konfigurálása szükséges a folyamat lezárásának szempontjából annak érdekében, hogy az érintett személyek számára visszajelzést tudjak küldeni a tesztelés eredményéről. Ehhez telepítenem kell az „*Email Extension Plugin*” kiegészítőt a jenkins grafikus felületén keresztül, majd a telepítés végeztével a jenkins „Rendszer beállítások” menüpontjában tudom konfigurálni a megfelelő adatokkal. Először is a „*System Admin e-mail address*” mező mellett található mező kitöltése szükséges, ahol azt az email címet adom meg amelyről az értesítéseket küldeni szeretném, és amely címre vonatkozóan az smtp beállításokat a továbbiakban elvégzem. A következő lépés az „Extended E-mail Notification” rész alatt található smtp beállítások kitöltése, illetve a „Default Recipients” mezőben a címzettek megadása, ahol az egyes címeket egymástól vesszővel elválasztva kell megadni. Majd a beállítások között legalul található az „E-mail Notification” rész, ahol pedig szintén meg kell adni az smtp szerverre vonatkozó beállításokat, és a jobb oldalon található „haladó” gombra kattintva pedig azt az email címet, amely segítségével az értesítések küldése történik, illetve az email címhez tartozó jelszót. Mivel küldésre a saját email címemet használom, így az adataim biztonsága érdekében ezeket a lépéseket nem szemléltetem képekkel. [11]

Ezzel az email küldéshez szükséges konfigurációs lépések beállításra kerültek, így a pipeline script legvégén már kiegészíthető a folyamat az email küldés funkciójával. Ennek a megadása során fontos, hogy ez a kód részlet minden esetben kerüljön végrehajtásra, még akkor is, ha a megelőző lépések során bármilyen hiba történt, ugyanis a hibákról is szükséges az értesítések küldése a kompetens személyek számára. Ez azért fontos, mert a jenkins szoftver alap esetben úgy működik, hogy ha a korábbi lépések során hiba merül fel, akkor a pipeline script futtatását megszakítja, és a hibát követő részek végrehajtása nem történik meg. Ennek megfelelően a script tartalma a következő. [11]

post {  
 always {  
 emailext body: "${currentBuild.currentResult}:  
 ${env.JOB\_NAME} - teszt: ${env.BUILD\_NUMBER}\n Bővebb  
 információ: ${env.BUILD\_URL}",  
 recipientProviders: [[$class: 'DevelopersRecipientProvider'], [$class: 'RequesterRecipientProvider']],  
 subject: "Jenkins Build ${currentBuild.currentResult}: ${env.JOB\_NAME}"  
 }   
 failure {  
 sh "/var/lib/jenkins/Apache\_Spark\_Connect/getinfraid.sh ${env.JOB\_NAME}"  
 }  
}

A fenti kód legvégén látható „*failure*” kezdetű részlet felelős azért, hogy ha a folyamat során bármi féle probléma történt a környezet létrehozását követően, akkor is törlésre kerüljön a környezet. Ezzel a megoldással nem maradnak a felhőszolgáltatásban használt felhasználói fiókhoz társított kihasználatlan eszközök még hiba esetén sem.

## **Tensorflow Specifikus folyamatok**

# Tesztelés

Miután az automatizált teszteléshez szükséges környezet kialakításra került, és az általam tesztelni kívánt szolgáltatásokat megfelelően ki tudom építeni és törölni, nincs más hátra csak a létrehozott infrastruktúrák vizsgálata, illetve a tesztek eredményeinek kiértékelése.

## **Tesztelést végző scriptek**

A dolgozatom célját biztosító környezet kialakítása során korábban említettem, hogy a tesztesetek elvégzéséhez különböző bash scripteket készítettem, melyek segítségével automatikusan tudom vizsgálni a tesztelni kívánt szolgáltatással kapcsolatosan létrejött virtuális eszközöket. A dolgozatom jelenlegi fejezetében ezen scriptek kialakítását ismertetem a teljesség igénye nélkül, ugyanis a scriptek működését és logikáját szeretném bemutatni, nem az elkészített kódokat.

Célom két különböző szolgáltatás automatikus tesztelése, így ennek megfelelően a Jenkins szoftver *home* könyvtárában létrehoztam két alkönyvtárat, melyek az adott környezet vizsgálatához szükséges, és használt állományokat tartalmazzák. Ezek a scriptek felelősek az ssh kapcsolat kialakításáért a létrehozott infrastruktúrákkal, a szükséges adatok szöveges állományba írásárért, és ezek ellenőrzéséért, illetve a környezet törléséhez szükséges parancsok végrehajtásáért is.

Ahhoz hogy a szükséges állományok meglétét vizsgálni tudjam készítenem kellett egy olyan scriptet amely képes ssh kapcsolatot kialakítani az automatikusan létrehozott infrastruktúra eszközeivel. Ennek kialakítása során figyelembe kellett vennem a klaszter rendszerek jellemzőit, miszerint ezek a rendszerek több gépből is állhatnak, és ezen gépek pontos száma a kialakítás kezdetén nem ismert. Továbbá a létrehozott eszközök ip címei is a felhőszolgáltatások jellegéből adódóan változnak, tehát nem használhattam ezen paraméterek esetén előre megadott értékeket. Az előbb említett szempontoknak megfelelően készítettem egy scriptet amely az Occopus szoftver megfelelő napló állományaiból egy szöveges állományba gyűjti az ip címeket, mindegyiket pontosan egyszer. Ezzel változó darabszámú cím kiolvasására alkalmas, továbbá nem előre megadott ip címeket keres, hanem egy általam előre megírt mintát.

grep -Eo '([0-9]{1,3}\.){3}[0-9]{1,3}' /var/lib/jenkins/workspace/$1/occopus.log | sort | uniq > /var/lib/jenkins/Apache\_Spark\_Connect/target\_ips.txt

A script egyetlen bemeneti paraméterrel rendelkezik, amely az érintett Jenkins projekt neve kell legyen. A script hívása a Jenkins szoftveren keresztül történik ahol környezeti változók használatával kell az adott paramétert megadni.

A szükséges rendszerek ip címének kigyűjtését követően a script egy ciklusba ágyazva meghívja a következő scriptet amely az ssh kapcsolat kialakításáért felelős, paraméterként átadva annak az egyes ip címeket. Ebben a scriptben Apache Spark klaszter esetén tulajdonképpen kétszer történik meg egyetlen gépre az ssh kapcsolódás. Első esetben a cél eszköz nevét egy változóban tárolom el annak érdekében, hogy a script további részeiben a virtuális gép szolgáltatásban betöltött szerepének megfelelő lépéseket tudjam végrehajtani. Ugyanis az említett szolgáltatás esetén a *master* illetve a *worker* eszközök már a szükséges állományok listájában eltérnek egymástól, tehát *master* esetén vannak olyan állományok és könyvtárak melyek meglétét a *worker* állomások esetén nem kell megvizsgálni. Ilyen állomány például a *hosts* állomány, mely a klaszterrendszerbe tartozó eszközök ip címeit tárolja, és amely állomány csak a *master* eszközön található. Ezeket a vizsgálatokat azonban nem tudom azonnal végrehajtani így az ssh kapcsolat során a céleszközön automatikusan végrehajtódnak a szükséges parancsok, melyek eredményei egy szöveges állományban tárolódnak, és melyet megvizsgálva tudom eldönteni hogy a kiépült eszköz konfigurációja megfelelő-e. Minden egyes céleszközhöz külön szöveges állomány képződik, *worker* eszközök esetén az állomány nevében megjelenítve az ip címet, így egyértelműen azonosítható, hogy melyik fájl melyik eszköz adatait tartalmazza.

A következő lépés az összegyűjtött információk helyességének ellenőrzése, amit csoportokra bontva végzek el. Ennek az a lényege, hogy a szükséges fájlok ellenőrzésénél például minden egyes állomány vagy könyvtár létezését megvizsgálom, és amennyiben nem létezik, akkor csak egy hiba üzenetet írok egy szöveges állományba. Így ha minden releváns állomány vizsgálata megtörtént, akkor elegendő csak azt megvizsgálni, hogy a keletkezett szöveges állomány tartalmaz-e hibaüzenetet. Amennyiben nem akkor a script megfelelő módon, hibajelzés nélkül bezár. Abban az esetben, pedig ha van hiba üzenet, akkor a *linux* rendszer beépített *exit* kódjainak segítségével az ellenőrző script futtatását megszakítom. Ezek a kódok lehetőséget nyújtanak arra, hogy a script futását hibakóddal szakítsam meg, melyet a Jenkins szoftver figyelembe vesz, és hiba esetén a folyamatot megszakítja, majd az általam hiba esetére meghatározott lépéssorozatot követve az infrastruktúra törlésre kerül. A vizsgálat során az egyes paramétereket változókban eltárolva vizsgálom, és amennyiben legalább az egyik paraméter nem az elvárt értéket tartalmazza akkor a script hibával zárul, és a további eszközök kiértékelése, illetve a folyamat további lépései nem kerülnek végrehajtásra.

[ -d /home/occouser ] && echo "Occouser home directory exists" || echo "Occouser home directory does not exists";

A fenti kódrészlet egy egyszerű elágazás, amely a meghatározott elérési útvonal létezését vizsgálja a Tensorflow környezet esetén. Amennyiben az elérési út nem létezik, a "*Occouser home directory does not exists*" üzenet kerül bele a képződő szöveges állományba. Minden hiba üzenetet hasonló felépítéssel készítettem el, így az utólagos ellenőrzés során elegendő csak azt megvizsgálni, hogy az állomány tartalmazza-e a „*does not*” részletet. Amennyiben az előbb említett rész megtalálható a szövegben akkor legalább az egyik állomány nem létezik, így a tesztet hibásnak tekintem.

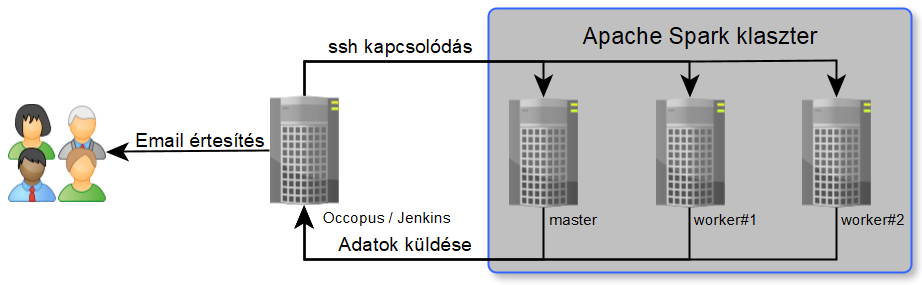
 API hívást ha sikerült a scriptben megírni akkor azt is bele kell foglalni ebbe!

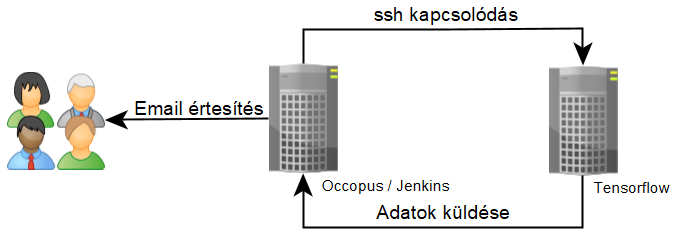
## **Apache Spark klaszter**

Mit csinálok, milyen esetek lehetségesek, milyen határesetek vannak, milyen hibák lehetségesek. stb. Mi az az eset, ami még elfogadható.

Apache esetén klaszter üzemmódban kell futtatni a parancsokat, hogy minden állomás beledolgozzon.

Képernyőképek a tesztek állapotairól, lehet benne hibás futásnak is a képe.





Kell külön táblázat a worker és a master esetében is!! a meglévő állományok listája miatt!

|  |  |  |
| --- | --- | --- |
| Paraméter | Elvárt érték | Kapott érték |
| Master gép neve |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Apache spark klaszter esetén elvárt értékek, melyik vizsgálat milyen eredményt hozott. Pl hosztnév vizsgálat elvárt: apache-master eredmény: apache-master stbstb

két részre lehetne osztani master-workerre

## **Tensorflow**

Ide olyanokat is le kell írni hogy klaszter rendszer esetén változó számú gép tesztelése is előfordulhat, és a scriptem ezt hogyan biztosítja. (ssh-nál, ip keresésnél, a grep miért jó a patternel stb). az eredményeket hol tárolom. Azokat ki kéne elemezni hogy megfelelnek-e annak amit elvárok.

**[LEGFONTOSABB GONOLAT: AMIKOR EMAIL ÉRTESÍTÉST KÜLDÖK AKKOR AZ MINDIG SUCCESS HA VÉGIG FUT A JENKINS GOND NÉLKÜL, DE AZ NEM AZT JELENTI HOGY A TESZTEK EREDMÉNYEI IS SIKERESEK VOLTAK A TXT FÁJLOK VIZSGÁLATA SORÁN! ERRE VALAMIT MUSZÁJ KITALÁLNI. MERT HA ÍROK EGY BASH-T AMI ELLENŐRZI A LOGOKAT AMIKET A GÉPEKRŐL GYŰJTÖK, DE AZ HIBÁT TALÁL A FOLYAMAT AKKOR IS VÉGIG FUT ÉS SUCCESS LESZ!!!!] – exit kódok**

## **Tesztelési eredmények**

# Összefoglalás

# Irodalomjegyzék

|  |  |
| --- | --- |
| [1] | J. Kovács és P. Kacsuk, „Occopus: a Multi-Cloud Orchestrator to Deploy and Manage Complex Scientific Infrastructures,” *Journal of Grid Computing,* pp. 19-37, 22 November 2017. |
| [2] | „Occopus v1.7 útmutató,” [Online]. Available: http://www.lpds.sztaki.hu/occo/user/html/tutorial-bigdata-ai.html. [Hozzáférés dátuma: 02 05 2020]. |
| [3] | M. C. Csapat, „Apache Spark klaszter felépítési útmutató,” [Online]. Available: https://cloud.mta.hu/sites/default/files/2019-01/spark\_tutorial\_mta\_cloud\_hun\_0.pdf. [Hozzáférés dátuma: 02 05 2020]. |
| [4] | M. C. csapat, „Tensorflow, Keras és Jupyter notebook stack felépítési útmutató,” [Online]. Available: https://cloud.mta.hu/sites/default/files/2019-07/tensorflow\_keras\_jupyter\_tutorial\_mta\_cloud\_hun\_2.pdf. [Hozzáférés dátuma: 02 05 2020]. |
| [5] | B. Balázs, „Bakai Balázs Java-közeli élményei,” [Online]. Available: http://www.bakaibalazs.hu/2012/01/folyamatos-integracio-kialakitasa.html. [Hozzáférés dátuma: 02 05 2020]. |
| [6] | „Telvice,” Telvice Kft, [Online]. Available: https://telvice.hu/szolgaltatasaink/devops-tamogatas/ci-folyamatos-integracios-teszteles-es-fejlesztes-tamogatas. [Hozzáférés dátuma: 16 05 2020]. |
| [7] | „Jenkins pipeline dokumentáció,” [Online]. Available: https://www.jenkins.io/doc/book/pipeline/. [Hozzáférés dátuma: 02 05 2020]. |
| [8] | „Occopus telepítési dokumentáció,” [Online]. Available: http://www.lpds.sztaki.hu/occo/user/html/setup.html. [Hozzáférés dátuma: 08 05 2020]. |
| [9] | „Jenkins telepítés dokumentáció,” [Online]. Available: https://www.jenkins.io/doc/book/installing/. [Hozzáférés dátuma: 08 05 2020]. |
| [10] | „Jenkins email kiegészítő dokumentáció,” [Online]. Available: https://plugins.jenkins.io/email-ext/. [Hozzáférés dátuma: 10 05 2020]. |
| [11] | G. Apolinario, „Medium - Gustavo Apolinario,” 14 05 2018. [Online]. Available: https://medium.com/@gustavo.guss/jenkins-sending-email-on-post-build-938b236545d2. [Hozzáférés dátuma: 09 11 2020]. |

# Ábrajegyzék

[2.1. ábra: Az Occopus által támogatott felhőszolgáltatások 4](#_Toc53421932)

[3.1. ábra: A "Continuous integration" eszközök mműködése 8](#_Toc53421933)

[3.2. táblázat: A tesztelési környezet kiválasztásához vizsgált szoftverek 9](#_Toc53421934)

[3.3.1. ábra: Jenkins pipeline projekt előzményei 13](#_Toc53421935)

[5.1. ábra: Virtuális gépek kapcsolata a felhőben 17](#_Toc53421936)

[6.1. ábra: RSA kulcspár generálása Windows operációs rendszeren 22](#_Toc53421937)

[6.2. ábra: Publikus kulcs módosítása az OpenNebula grafikus felületén 23](#_Toc53421938)

[6.3. ábra: A létrehozott virtuális gép adatai 24](#_Toc53421939)

[6.4. ábra: /etc/hosts állomány módosítása 24](#_Toc53421940)

[6.5. ábra: Jenkins grafikus felülete az első indítást követően 26](#_Toc53421941)

[6.6. ábra: Jenkins aktiválásához szükséges jelszó kiolvasása 26](#_Toc53421942)

[6.7. ábra: Jenkins bővítmények listája az ajánlott telepítés során 27](#_Toc53421943)

[6.8. ábra: Jenkins pipeline scriptet futtató felhasználó 28](#_Toc53421944)

[6.9. ábra: Az autentikációs fájl kialakítása 30](#_Toc53421945)

[6.10. ábra: SSH kulcs átadása a jenkins felhasználó számára 32](#_Toc53421946)

# Táblázatjegyzék

3.1. táblázat: A tesztelési környezet kiválasztásához vizsgált szoftverek 9

5.1. táblázat: A választott szoftverekkel szemben támasztott elvárások 19

5.2. táblázat: A használt felhő interfész API adatai 20