ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK
2023

Balogh Nándor
T/005558/FI12904/N

Óbudai Egyetem
Neumann János Informatikai Kar
Kiberfizikai Rendszerek Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve:	**Balogh Nándor**
Törzskönyvi száma:	T/005558/FI12904/N
Neptun kódja:	[redacted]

A dolgozat címe:

OpenStack felhő szolgáltatás és az azon futó szolgáltatások monitorozása
Monitoring the OpenStack cloud service and the services running on it

Intézményi konzulens:	Dr. habil. Lovas Róbert
Külső konzulens:	Pintér Ádám (Wigner)
Beadási határidő:	2022. december 15.
A záróvizsga tárgyai:	Számítógép architektúrák Felhőszolgáltatási technológiák és IT biztonság specializáció

A feladat

Komplex IT rendszerek hatékony üzemeltetésének egyik fő komponense a monitorozás, azaz, hogy adminisztrátori és fejlesztési oldalon átláthatóan nyomon követhessük a rendszerben zajló folyamatokat, a problémákról már azok felmerülésekor értesülhessünk. A gyors beavatkozás esetén lehetővé válik az esetleges negatív következmények minimalizálása. A hallgató feladata, hogy ismertesse a monitorozási technológiákat és azok legnépszerűbb implementációs keretrendszereit. Az OpenStack felhőhöz tartozó monitorozási paramétereket gyűjt, melyeket típus szerint kategorizál. Számos paraméter közül –megadott szempontok szerint– kiválasztja a leginkább megfelelőeket, melyek a megfigyelő rendszer adatpontjai lesznek. Megtervezi a monitorozási rendszer architektúráját, valamint megvalósítási és tesztelési tervet készít, amely biztosítja az OpenStack felhőhöz megfelelő illeszkedést.

A dolgozatnak tartalmaznia kell:

- monitorozási technológiák bemutatása, keretrendszerek ismertetése a szakirodalom alapján,
- monitorozási paraméterek kiválasztásának szempontjai,
- követelmények analízise, hogy megfelelően illeszthető legyen a megfigyelő rendszer a kiépített infrastruktúrához,
- kialakítandó monitorozási rendszer tervezése,
- monitorozási rendszer megvalósításának lépései,
- megvalósult monitorozási rendszer tesztelési terve és tesztelése,
- továbbfejlesztési lehetőségek.

Ph.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest

.....
Dr. Fleiner Rita
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022. 11. 30.

.....

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun kód:	Tagozat:
Balogh Nándor	[redacted]	Felhő és IT biztonság

Telefon: Levelezési cím (pl: lakcím):

[redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

OpenStack felhő szolgáltatás és az azon futó szolgáltatások monitorozása

Szakdolgozat / Diplomamunka[2] címe angolul:

Monitoring the OpenStack cloud service and the services running on it

Intézményi konzulens:	Külső konzulens:
Dr. habil. Lovas Róbert	Pintér Ádám

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.03.08	Szakdolgozat témájának átbeszélése	DL
2.	2022.03.29	Tervezett előrehaladás ismertetése	DL
3.	2022.04.19	Felmerülő kérdések tisztázása	DL
4.	2022.05.10	Tartalmi és formai részek egyeztetése	DL

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2022.05.10

DL

.....
intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun kód:	Tagozat:
Balogh Nándor	[redacted]	Felhő és IT biztonság

Telefon: Levelezési cím (pl: lakcím):

[redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

OpenStack felhő szolgáltatás és az azon futó szolgáltatások monitorozása

Szakdolgozat / Diplomamunka[2] címe angolul:

Monitoring the OpenStack cloud service and the services running on it

Intézményi konzulens:	Külső konzulens:
Dr. habil. Lovas Róbert	Pintér Ádám

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.09.20	Fizikai architektúra átbeszélése	
2.	2022.10.14	Előrehaladást segítő kérdések átbeszélése	
3.	2022.11.16	Tartalmi részek tisztázása	
4.	2022.11.30	Végleges formátum kialakítása	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2022.11.30

.....
intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

Tartalomjegyzék

1 BEVEZETÉS

Felhőszolgáltatások monitorozása alatt a felhőben futó folyamatok megfigyelését értjük. Ezek a monitorozási műveletek történhetnek manuálisan és/vagy automatikusan is, beépített monitorozó szolgáltatással vagy külön erre dedikált eszközökkel is.

Az integrált szolgáltatások nem igényelnek külön IT infrastruktúrát, egyszerűen és gyorsan skálázhatók külsős beavatkozás nélkül. Nincsen beüzemelési késleltetés, azaz automatikus telepítés révén, telepített eszközünkre előre konfigurált állapotában azonnal használhatjuk. A legelterjedtebb, és a legtöbbet használt modulok a felhő szolgáltatója által nyújtott beépített lehetőségek. Általában ezekről a megoldásokról elmondható, hogy extra költségek nélkül használhatóak, könnyen alkalmazhatóak és felhasználóbarátak. Sok cég előszeretettel használja ezeket, hiszen külső beavatkozást nem igénylő művelettel képesek elérni a kívánt hatást.

A másik nagy kategória, a külső források által szolgáltatott, integrálható konstrukciók. Ezek tipikus jellemzője a magasabb költség, az integrációval kapcsolatos problémák és az ahhoz szükséges szakértelem, valamint a működtetéséhez való hozzáértés megléte.

Ebben a két csoportban lévő kiegészítő szolgáltatások választástól függetlenül egy célt szolgálnak, potenciális hibák után kutatva monitorozzák a futó egységeket. Ezeket a tevékenységeket általában két fő okból alkalmazzuk. Az egyik ilyen fő ok, a valós idejű folyamatok megfigyelése, a másik pedig a biztonság.

1.1 Megfigyelési fajták

A felhő szolgáltatásokra jellemző gyorsaság és agilitás következményeként, a felhasznált rendszerek mind fizikai mind pedig logikai kapcsolata meghatározza a monitorozás jellegét. Számtalan különböző részből álló infrastruktúra, magával hozza a változatos, gyakorta változó elemeket, melyek hibamentes kommunikációja előfeltétele a helyes működésnek. Ezek a szükségletek teljesítése segítette elő a monitorozás diverzifikálását.

Adatbázis monitorozás, a legtöbb felhőben futó alkalmazás is adatbázisban tárolja értékeit, adatait. Ezzel a fajta monitorozással információt kaphatunk az adatbázis által felhasznált erőforrásokról, az adattárolási folyamatokról, a lekérdezésekről, a rendelkezésre állásról, továbbá a hozzáférési jogosultságoknak nyomon követéséről is.

A virtuális gépek (VM) számítógépet jelentenek a számítógépben. Egy virtualizáció, mely lehetőséget biztosít számunkra egy fizikai számítási kapacitással rendelkező elemen, több, akár különböző operációs rendszerrel futtatott számítógépek futtatására. Ezáltal megspórolva, a különböző konfigurációból adódó fizikai rendszerelemeket. A hagyományos IT infrastruktúrák monitorozásának előnyeit kombinálhatjuk, felhős megvalósításban. Ezeknek a virtuális gépeknek a monitorozása történhet mind a felhasználói oldalról, mind a fizikai erőforrás kihasználásról.

Virtuális hálózatok monitorozása képes, a fizikai hálózati elemeket, a tűzfalakat, routereket, hálózati elosztó eszközöket szoftveres szinten megfigyelni. A szoftveres

programozásnak köszönhetően, képesek a rajtuk áthaladó adatok alapján, a rendszerek állapotát felmérni.

1.2 Felhő biztonság

A gyakorlatban a felhő biztonság a folyamatos megfigyelést jelenti mind a virtuális mind pedig a fizikai szervereken a fenyegetések kivédése és a sebezhetőség csökkentése érdekében. Általánosságban elmondható, hogy ezt nem kézi erővel történik, hanem automatizmusok segítségével. Ezek leginkább az adatok mérésén alapszanak. Adatok, alkalmazások és hálózatok működését mérik fel. Egyes kiegészítő szolgáltatások a szerverek naplózási adatainak összegyűjtése alapján működnek, míg a fejlettebb verziók szokatlan funkciók vagy adatok megjelenéséből következtetnek a biztonsági kockázat eshetőségére. Ezekről a potenciális problémákról küldenek értesítést.

1.3 A biztonsághoz szükséges alapfeltevések

- Skálázhatóság: A teljes biztonság érdekében nem elég, ha csak a tárolt adataink egy részét tudjuk nyomon követni, az egészre szükség van. Ezen okból kifolyólag hatalmas információ mennyiséget kell átvizsgálni, méghozzá különböző típusúakat, akár többféle forrásból származókat.
- Folyamatosság: Szükséges valós időben történő műveleteket is lekövetni. A támadások megelőzése érdekében, a gyors valós idejű megfigyelés és esetleges beavatkozás elengedhetetlen a védelem szempontjából. Mind a gyorsaság, mind a számítási kapacitás mértéke fontos tényező.
- Integráció: A biztonság maximalizálása érdekében érdemes a rendszerünkhöz kapcsolódó alkalmazásokat megvizsgálni és kiegészíteni őket. Az általuk nyújtott szolgáltatás megfigyelését, esetleges biztonságosabb verzió használatát. A végpontok biztonságos mivoltát, valamint a személyazonosítási és a hitelesítési eljárásokat, illetve a kétlépcsős hitelesítést bevezetni.

Megelőzheti az igény bevevő bevételkiesését. Nincs is annál rosszabb egy cég számára, mikor a rábízott felhasználói adatokat illetéktelenül megszerzi egy támadó és kikerül a nyilvánosság elé. Ilyenkor nem csak bevételkiesést okoznak, hanem a cégbe vetett bizalma is megrendül a felhasználónak, továbbá csökkenti az elégedettséget.

Hasznos a nyomon követés biztosítására is, hiszen minden fontosabb szabályozás követelménye. Sok esetben felügyeleti eszközök használata kötelező az egyes vállalatok számára.

1.4 Felhőszolgáltatások architektúrális megközelítése

Mikor felhő szolgáltató választásra kerül a sor, meg kell fontoljuk, hogy milyen szintű szolgáltatást szeretnék igény bevenni. Attól függően, hogy publikus vagy privát felhővel dolgozunk megkülönböztetünk három szintet. Ezek a szintek az IaaS (Infrastructure as a

Service), PasS (Platform as a Service) valamint a SaaS (Software as a Service). Ezek a szoftver, a platform és az infrastrukturális szintek. Ezek a kifejezések nemcsak a felhő infrastruktúrában használatosak, hanem az egész IT területén is.

A szintek közti különbségéből adódóan, a legnagyobb eltérés a publikus és privát felhő között, hogy a privát felhő esetén lehetőségeink szinte korlátlanok, majdnem teljes befolyásunk van a platform felett. Publikus verzió esetén, sokkal nagyobb a korlátokba ütközünk, itt ugyanis túlnyomórészt a szolgáltató által meghatározott eszközökkel kell dolgozzunk. Míg publikus felhő használatakor a felhő rendszerünket a szolgáltató nagymértékben befolyásolja, addig privát felhő esetén, saját, egyénre szabható rendszer alakíthatunk ki. Értve ezalatt mind a rendszerek és folyamatok működését, mind pedig az adatok kiszolgáltatottságát.

1.4.1 Szoftver szinten

Egyértelműen ezen a szinten tudjuk legkevésbé megfigyelni rendszerünket. A legfőbb feladat ezen a szinten, a végfelhasználó útjának nyomon követése, az ISP-től a publikus felhőig. Ez a művelet magában foglalja a késleltetést, a tartomány névhez köthető sebességet és megközelíthetőségét, valamint a végfelhasználó előtti megjelenítést. Kiegészítő szolgáltatások, mint például a webes tanúsítványok, felhasználó alapú tűzfalak, adatvesztés elleni eszközök mind befolyásolják a teljesítményét így a megfigyelésnek erre is ki kell térnie.

1.4.2 Platform szinten

Nagyobb rálátást enged az adminisztrátoroknak, mint az előző kategória. Itt már nemcsak a publikus felhőig követjük a felhasználót, hanem az azon belüli műveletvégzését is. Olyan adatokról is kaphatunk információkat, mint a rendszerünk erőforrásigénye, vagy például a CPU, memória kihasználtsága. Ezeket az adatokat felhasználhatjuk, a rendszer teljesítményének optimalizálásához.

1.4.3 Infrastrukturális szinten

A legalsó szint adja meg számunkra a legnagyobb szabadságot. A korábban említett szinteket magában foglalja, továbbá rálátásunk lesz, a szerver operációs rendszer szintjére is. Itt a felhasználót lényegében nyomon tudjuk követni az elejétől egészen az operációs rendszer erőforrás szintjéig.

2 OPENSTACK

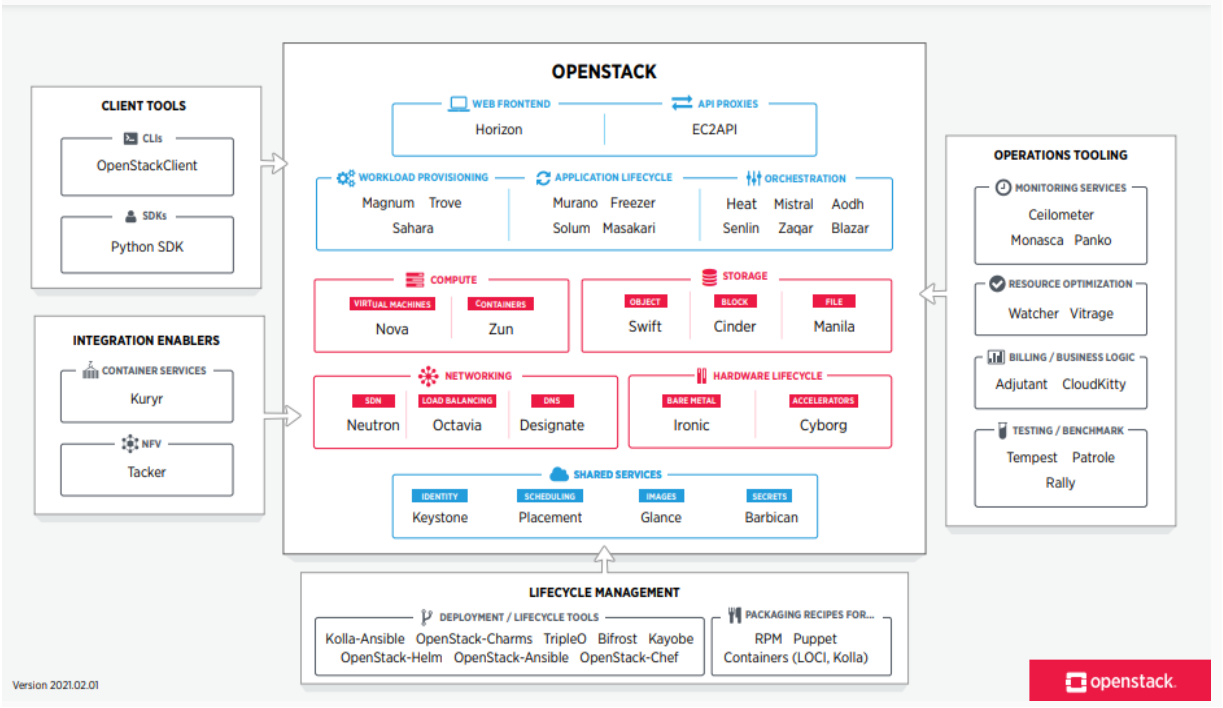
Az OpenStack egy felhő operációs rendszer. Olyan infrastrukturális szolgáltatás, amellyel projektjeink számítási kapacitását, adattárolási és hálózati erőforrásait tudjuk biztosítani egy adatközpontban, mindezeket pedig közös hitelesítési mechanizmusokkal rendelkező API-k segítségével érjük el. [1]

Nyílt forráskódú eszközeivel kezelhetünk vagy létrehozhatunk felhő platformokat. Szoftverek gyűjteményét tartalmazza, amelyek széleskörű szolgáltatásokat nyújthatnak számunkra. Használhatjuk mind a publikus, mind a privát felhők kezelésére. Nagyságát bizonyítja, hogy világszerte húszmillió CPU magot biztosít a felhasználók számára, beleértve a nagyvállalatokat is. Olyan nagynevű szervezetek, vállalatok is használják, mint a Walmart, a játékgyártó Blizzard Entertainment vagy az európai laboratórium, a CERN. Magyarországi viszonylatban, az ELKH (Eötvös Loránd Kutatási Hálózat) Cloud projektje, mely a SZTAKI és Wigner FK nyújt, az egyik legnagyobb felhasználója a platformnak.

Saját felhő infrastruktúrát alakíthatunk ki az OpenStack által. Alkalmazása különösen előnyös olyan szervezetek számára, akik olyan érzékeny adatokkal, információkkal rendelkeznek, amelyeket nem akarnak megosztani olyanokkal, akik más webszolgáltatást valósítanának meg, mint például a Microsoft Azure vagy az Amazon Web Services. Különleges megvalósítását úgy tervezték, hogy különböző erőforrásokat egyetlen adatközpontban tudjon kezelni. A fizikai erőforrásokat gyűjti össze egyetlen központi gyűjteménybe, majd azokat a további virtuális erőforrásokat, amikre szükségünk van, egy másik gyűjteménybe rendezi, és ezt használja fel forrásként. Privát felhőket hozhatunk létre adatközpontokban azáltal, hogy engedélyezi a különböző felhőszolgáltatások interakcióját egymással.

Azokat az OpenStack által használt eszközöket, amelyek közrefogják a számítási kapacitásokat, a hálózatot, az adattárolást, valamint a hitelesítést, projekteknek hívjuk. Szinte megszámlálhatatlan lehetőségünk van arra, hogy létrehozzuk ezeket az egyedi felhő példányokat. Ezekhez és az általunk fejlesztett egyedi példányokhoz biztosít számunkra konfigurációs mintákat az OpenStack. Ezek a minták és sablonok esettanulmányokon és valós ipari felhasználáson alapszanak. Ezek segítségével próbálják a konfigurációt megkönnyíteni számunkra, továbbá plusz információkat is kinyerhetünk, hogy mely alkotóelem milyen felhasználási területre illeszkedik legjobban.

Kialakítását tekintve úgy tervezték meg, hogy bárki a maga igényeire szabhatja, úgynevezett plug and play módszerrel. Ez azt jelenti, hogy külön konfiguráció nélkül, az egyes elemeket csak importálni kell a grafikus felületen, és a háttér rendszerekben megtörténik az összeköttetésük. A 2.1 ábra mutatja meg számunkra a rendszer tagoltságát, valamint a rendszer főbb alkotóelemeit, részeit. [2]



2.1. Ábra OpenStack felépítése [1]

2.1 Alkotóelemei

Nyílt forráskódja révén, több különböző részből tevődik össze. Bárki hozzáadhat extra komponenseket saját igényeinek megfelelően, az alap összetevők viszont állandóak. Az OpenStack közösség jellemzően hat fő alapelemet említ meg, mely a működést biztosítja. Ezeket a részek hivatalosan is a fő rendszerhez tartoznak és közösségi elvek alapján tartják fenn. [3]

2.1.1 Nova

Az OpenStack motorja. Az első számú mozgatórugója a gépezetnek, ez biztosítja a számítási kapacitást. A hatalmas mennyiségű adat és virtuális gépek működtetéséhez és futtatásához szolgáló egység, mely kezeli és irányítja a feladatokat. Ez a komponens felelős a API hívások kezelésért, továbbá az alapvető műveletvégzésért. Többféle virtualizációt támogat, ilyen a VMware, a KVM vagy a Hyper-v is.

2.1.2 Swift

Az adatok konténerekbe történő tárolásáért felelős rész. A hagyományos adattárolási módszerektől merőben eltér. Általános esetben, egy adott fájl elérését úgy tudjuk meghatározni, hogy egy adott adattárolási eszközön megcímezzük a fájl konkrét helyét. Ezzel szemben, itt a fejlesztők egy különleges jelzővel látják el a tárolt adatot majd rábízzák az

OpenStack-re, hogy eldöntse hol tárolja el. Ez a megoldás megnöveli a skálázhatóságot, hiszen nem kell törődni az adott szoftver mögötti háttértár méretével.

2.1.3 Neutron

A hálózat biztosításáért felelős. Működteti a hibamentes kommunikációt a komponensesek között. Segítségével létrehozhatunk virtuális hálózatokat, IP, címeket, de routerek és tűzfalak konfigurálásához is használhatjuk.

2.1.4 Keystone

Az authentikációért felelős mind a felhasználók mind pedig a szolgáltatások számára. Ellenőrzi a felhasználó személyazonosságát, azonosítja milyen elemekhez férhet hozzá a rendszerünkben. Az OpenStack legtöbbet meghívott szolgáltatása. Enélkül a szolgáltatás nélkül sem a felhasználók, sem a szolgáltatások működése nem lenne biztosított.

2.1.5 Cinder

A blokk tárolást biztosítja a virtuális gépekhez. A végfelhasználó számára jeleníti meg azokat a tárolási kapacitásokat, amelyeket a Nova használ fel. Egyfajta háttértárként funckionál. Projekteken belül allokálhatjuk az előre meghatározott adatmennyiségeket. Írható, olvasható lehetőséget biztosít az adatok felett. A különböző RAID (Redundant Array of Independent Disks) megoldásokat is ezen keresztül konfigurálhatjuk.

2.1.6 Glance

Konténerek és virtuális gépek képfájlaival foglalkozik, amelyek másik szolgáltatáshoz kapcsolódnak. Ezek a képfájlok általában egy előre elkészített operációs rendszerről készített mentés. Egy új VM létrehozásakor ezek a képfájlok szolgáltatják a bootolható egységet a Nova számára. Magába foglalja ezekkel a fájlokkal való munkavégzést, mint a tárolást, megosztást, keresést vagy letöltést.

2.2 Flavors

Ez a modul az OpenStack-en belül definiálja a számítási kapacitást és a memóriát minden virtuális gép számára. Meghatározza a futtatható virtuális gép méretét. Előre meghatározott hadveres konfigurációkat tartalmaznak.

Ahhoz, hogy ez a konfigurációs fájl működjön, szükségünk van olyan attribútumokra, amelyek meghatározzák a tartalmát és keretet adnak neki. A 2.1, 2.2, 2.3 ábránkon láthatjuk az ELKH Cloud két ágán (SZTAKI és Wigner Fizikai Kutatóközpont) használt géptípusokat Az első táblázatban az általános felhasználási módú géptípusok kerülnek szemléltetésre, utána pedig a memória optimalizált és a GPU gyorsításra szolgálóak.

VM típusa	vCPU (darab)	RAM (GB)	Root disk (GB)	vGPU memória (GB)
m2.tiny	1	1	10	0
m2.small	1	2	20	0
m2.medium	2	4	40	0
m2.large	4	8	80	0
m2.xlarge	8	16	160	0
m2.2xlarge	16	32	160	0
m2.4xlarge	32	64	160	0

2.1. Táblázat általános felhasználású géptípusok

VM típusa	vCPU (darab)	RAM (GB)	Root disk (GB)	vGPU memória (GB)
r2.medium	*2*	*8*	*80*	0
r2.large	*4*	*16*	*160*	0
r2.xlarge	*8*	*32*	*160*	0
r2.2xlarge	*16*	*64*	*160*	0

2.2. Táblázat memória optimalizált géptípusok

VM típusa	vCPU (darab)	RAM (GB)	Root disk (GB)	vGPU memória (GB)
g2.medium	*2*	*8*	*80*	*4-5*
g2.large	*4*	*16*	*160*	*8-10*
g2.xlarge	*8*	*32*	*160*	*16-20*
g2.2xlarge	*16*	*64*	*160*	*32-40*

2.3. Táblázat GPU gyorsításra szolgáló géptípusok

Ezek a specifikációk jellemzik a korábban említett hazai kutatóközpontok eszközeit. A minimális eltérés a virtuális GPU oszlopában található. A jobb oldali érték a Wigner Fizikai Kutatóközpont által használt értékeket jelöli, míg a bal oldali pedig a SZTAKI által alkalmazott gépek értékeit mutatja.

Ezeket a konfigurációs egységeket egy egyedileg meghatározott integer szám határozza meg, ez a Flavour ID, aminek definiálását lehet manuális vagy automatikusan generálva is végezni. A név mezőnél, ahogy az ábrán is láthatjuk, az xx.méret_neve elnevezés már csak megszokásból használatos, egyes külső szolgáltatások hivatkozhatnak még rá. A virtuális CPU-k száma melletti mezőkben megtalálható a gigabyte-ban kifejezett memóriák értéke. A következő kötelező elem a root disk, azaz a gyökér partícióhoz szükséges szabad tárolókapacitást jelöli. Opcionálisan használható a swap, RXTX factor valamint az Is public mezők is.

2.3 Miért kell monitorozni?

Szinte megszámlálhatatlan szempont és indok áll a monitorozás mellett. Minden olyan vállalkozás, amely valamilyen formában eszközöl felhő szolgáltatást, előnyt szerezhet versenytársaival szemben. Nemcsak szolgáltatásának minőségét növelheti, hanem a költségeit is csökkentheti. Előre jelezhet például egy fizikai meghibásodást. Amennyiben drasztikus hőmérsékletváltozás lép fel egy eszközben, úgy még meghibásodás előtt cserélhetjük az adott eszközt, vagy csak annak hibás modulját, ezzel is rengeteg költséget megspórolva. Optimalizálás révén segíthet csökkenteni az erőforrások költségeit, továbbá a behatolók és kártékony kódok által okozott károkat is meggátolhatja.

Monitorozással nem csak az felhőben futó rendszerek biztonságát növelheti, hanem a megfigyelt hálózatét is. Ideális megoldást biztosíthat külső támadások ellen, valamint optimalizálható az alkalmazások teljesítménye és fenntartása. Hála a folyamatosan futó értesítési rendszernek, magasabb rendelkezésre állást érhetünk el vele. Számos platformról és eszközről való elérhetősége tovább növeli a mindenkori elérését. Előfordulhat, hogy a túl sok vagy, túl kevés erőforrás esetén a rendszerben előforduló rendellenességek többletköltséget okoznak. A rendszer teljes átláthatósága miatt, ezek a felesleges kiadások elkerülhetőek.

Monitorozás kialakításánál több szempont figyelembevétele is szükséges. Ilyen például a rendszer kihasználtság, és az ehhez kapcsolódó monetizáció. Fontos a monitorozás erőforráskihasználásnak is figyelése. Az adatok megjelenítése is figyelése között nem feltétlen van egyenes arányosság. Csak a releváns és hasznos adatok megjelenítése szükséges, ezáltal is csökkentve az erőforrásszükségleteket. Az adatok konzisztenciájának növelése érdekében érdemes szeparálni a megfigyelt adatokat a monitorozási adatoktól. További fontos szempont az adatok végső megjelenítésének helye és módja. A rendkívül sokoldalú adatok más és más megjelenítési formát eszközölhetnek. Ahhoz, hogy valós és átlátható képet kapjunk az adatainkról, olyan felületet érdemes választani, ahol ezek az összegyűjtött adatok központosított, egységes rendszerként jeleníthető meg.

Mindezek mellett még sok más pozitívum is fellelhető. Minden platformnak megvan a maga sajátossága és a felhasználási célja. Más és más alkalmazási területekre, potenciálisan más és más a megfelelő platform, alkalmazás. Ahhoz, hogy rendeltetésének megfelelően válasszuk ki az adott platformot, szükséges megvizsgálnunk a piacon fellelhető főbb szolgáltatásokat, hogy az igényeknek a legmegfelelőbbet sikerüljön kiválasztani.

3 MONITOROZÁSI TECHNOLÓGIÁK

Monitorozási technológiából számtalan lehetőséget találunk szerte a világban. Mind ingyenes verziókat, mind pedig fizetős ajánlatokat. Felhasználási módját tekintve mindenki megtalálhatja igényeinek megfelelő szolgáltatás, melyet igény szerint konfigurálhat. Adatgyűjtési működésüket tekintve viszont hasonló a felépítésük. Máig két alapvető mechanizmus terjedt el. A push és pull modell alapú. Ezek architektúrális kialakítások felelősek az információ begyűjtéséért.

A push modell alapúaknál a monitorozandó egységeken futó kliensek küldik el az adatokat a szerver számára előre meghatározott időközönként. Ez egy egyirányú folyamat. Kevés erőforrásigénye miatt jól skálázható. Több ezer eszköz megfigyelésére alkalmas egyazon időben.

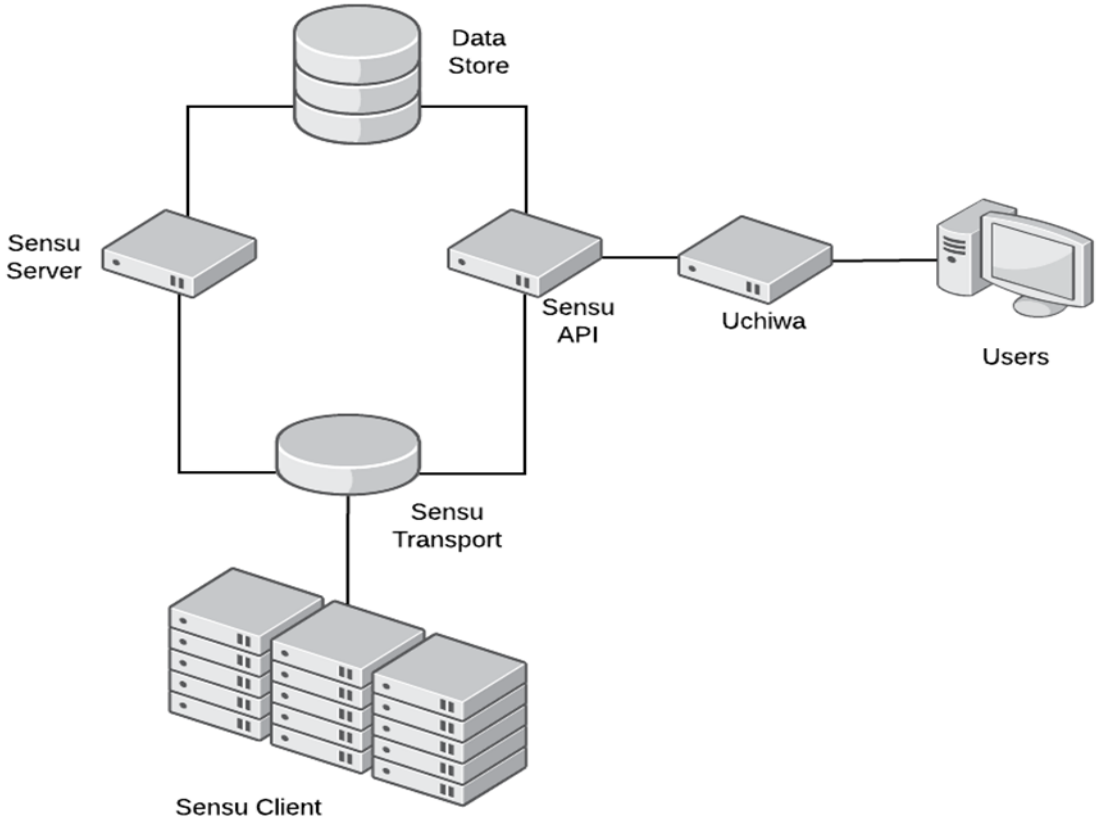
A pull modell alapúaknál pedig a központi feldolgozóegység kéri le az adatokat az előre definiált lista alapján. Ekkor értesíti a klienseket, lekéri a megadott paramétereket, összegyűjti majd feldolgozza azokat. Hátrányként ezeknél a megoldásoknál felróható, hogy kevésbé biztonságos megoldások közé tartozik. Ahhoz, hogy a rendszer működhessen, ahhoz egy hálózat alá kell tartozzanak a feldolgozó egységek, valamint a kliensek is. Ez azt jelenti, hogy az eszközök bele kell hallgassanak a rendszerbe, ami behatolási pontot eredményezhet egy támadónak.

3.1 Sensu

Felhőben futó alkalmazások megfigyelhetőségét hozza el számunkra kód formájában. A Sensu egy nyílt forráskódú infrastrukturális monitorozó szolgáltatás. Monitorozhatunk vele szervereket, szolgáltatásokat vagy akár alkalmazások teljesítményéről is kaphatunk információt. Széles körben integrálhatjuk a már meglévő felületeinkhez és monitorozó eszközeinkhez. Sajátossága, hogy az általunk megfigyelendő értékekről képes számtalan módon értesítéseket küldeni számunkra. Nem csak beépített üzenetküldő szolgáltatása van, hanem integrálhatunk külsős értesítési szoftvereket is, ezzel elérve a teljes rugalmasságot az értesítések terén. Ruby-ban íródott, így mind RabbitMQ mint Redis által küldött értesítéseket is könnyedén tud kezelni, továbbá az adatok tárolását is Redis szoftver segítségével oldja meg.

Az értesítések beágyazhatóak, olyan projektmenedzsment szoftverekbe is, mint a Slack, Trello vagy IRC, ezzel is növelve egy szervezeten belüli információ áramlást. Moduláris architektúrájának köszönhetően a komponensek szabadon variálhatóak, nemcsak ugyanarra a szerverre telepíthetőek, hanem akár külön álló gépekre is.[4]

3.1.1 Architektúra



3.1. Ábra Sensu architektúra [5]

A Sensu elsődleges kommunikációs csatornája a Transport nevet viseli. Minden Sensu komponensnek csatlakoznia kell a Transport vonalhoz, hogy létrejöhessen az üzenetküldési kapcsolat a két fél között. Ez a korábban említett RabbitMQ valamint a Redis-el lehetséges, ipari környezetben kimondottan javasolt a RabbitMQ használata. A Sensu szerver gondoskodik az adatok feldolgozásáról és megfigyeléséről, ellenőrzi az eredményeket és eseményeket. Regisztrálja az ügyfeleket, feldolgozza a bejövő információ áradatot szűrők és kezelőszervek segítségével. Ez a dedikált eszköz felelős a visszajelzés küldésért a felhasználók felé. A Sensu API restapi által biztosítja a hozzáférési pontot az adatokhoz és az alapvető funkcióihoz. A kliens vezérlő mind a szerver által küldött, mind a helyi változók ellenőrzését végzi. Végezetül pedig egy web interfész gondoskodik a kommunikáció azon részéről, amely a felhasználó és a sensu API között létrejön, korábban az Uchiwa dashboard(kezelőfelület) nevet viselte, jelenleg a Sensu Web UI nevet. A Sensu saját fejlesztésű egyszerű webes felülete a monitorozás nyomon követéséhez. Ezen a platformon követhetjük nyomon vizuálisan hálózatunkat. A szerverből kinyerve összegzi adatainkat, és jeleníti meg számunkra valós időben. Megadott időközönként változtatható, automatikus frissítésének köszönhetően a weboldal újra töltése nélkül figyelhetjük meg rendszerünket. Az adatok megjelenítését is személyre szabhatjuk, miközben a rendszerünk tulajdonságait nyomon követhetjük és még riportokat is készíthetünk róla. [5]

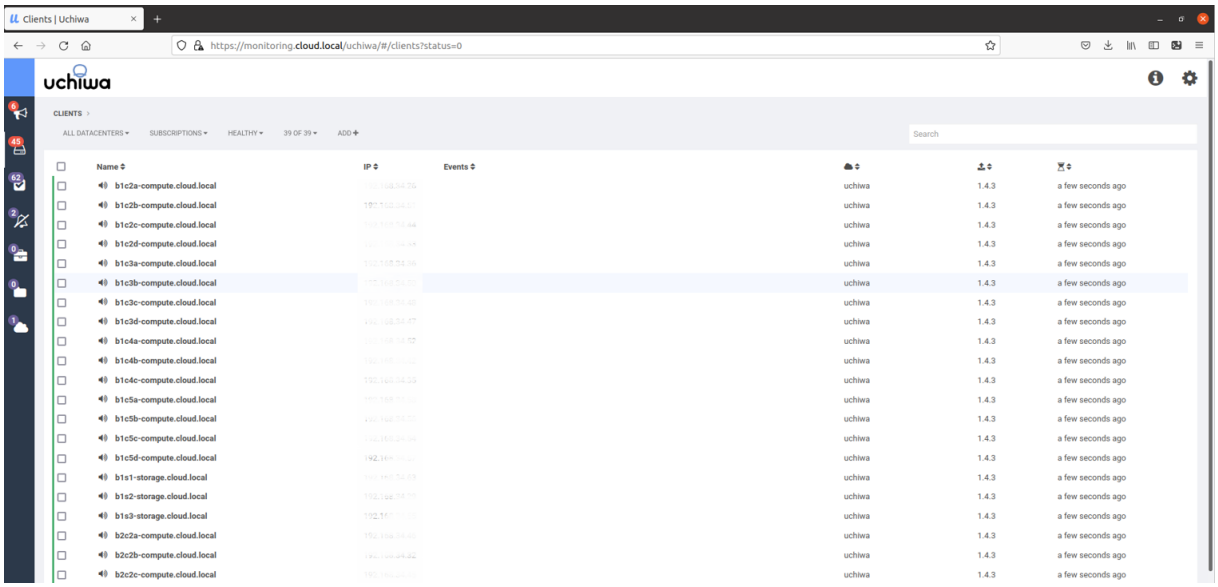
3.1.2 Ágens

A különböző eszközeink monitorozásához Sensu agent(ágenseket) szükséges futtatni. Ez a rész felelős az adatok megfigyelésért a különböző eszközökön. A backend-el áll kapcsolatban a monitorozandó példánnyal. Alkalmazhatjuk őket Linux, Windows, vagy MacOS-en is. Windows alatt a cmd.exe futtatható környezetet használja, míg más operációs rendszereken a Borune shell alkalmazandó.

Websocket protokollon keresztül folyik az üzenetváltás, az üzenetek pedig JSON formátumban kerülnek elküldésre. A két fél eseményeken keresztül kommunikál. Ezek az események információkat tartalmaznak az eredeti entitás, valamint az ezekre érkezett válasz üzenetekről, továbbá a kért paramétereket generikus konténerekbe ágyazzák be. [5]

Az autentikációt a backend végzi, a korábban említett websocket protokollon keresztül, két lehetséges opciót kínálva fel számunkra. Az egyik lehetőség a felhasználónév jelszó páros, a másik pedig a közös szállítási réteg biztonsági autentikációja, azaz a mTLS protokoll által.

3.1.3 Kezelőfelület



3.2. Ábra Uchiwa kezelőfelület

A korábban említett Uchiwa biztosította a kezelőfelületet a Sensu számára. Egy egyszerű nyílt forráskódú irányítópult a Sensu megfigyelési esemény folyamathoz. Könnyen letölthető és konfigurálható. Nincsenek függőségei, ezért alacsony az erőforrásigénye. Erősségét és kiterjedtségét jelzi, hogy több ezer szervert, alkalmazást vagy szolgáltatást monitorozhatunk vele egyszerre. [6]

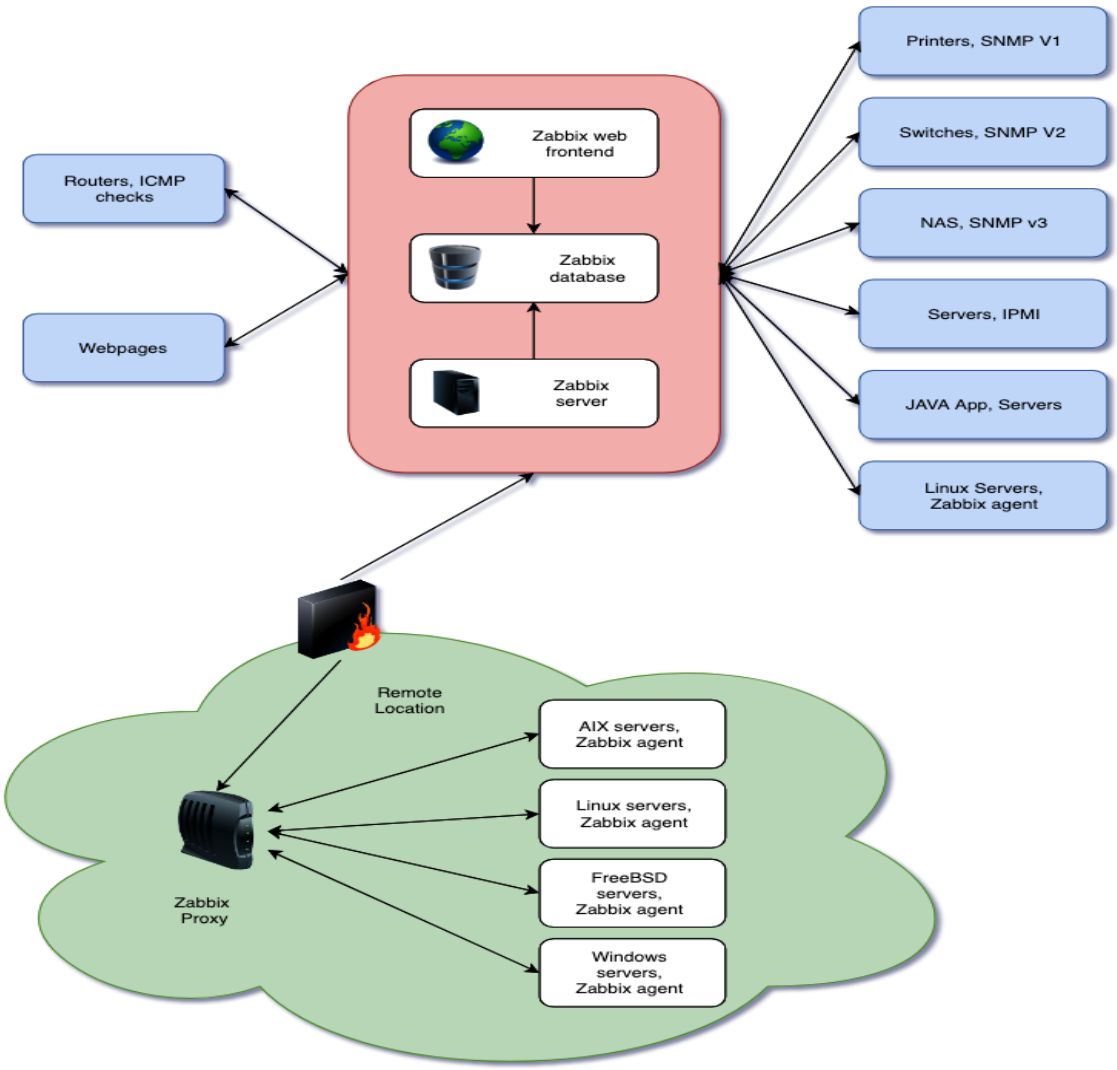
Ennek a továbbfejlesztett változata, a Sensu GO által behozott, saját fejlesztésű Sensu web UI. A felület az Uchiwa továbbfejlesztett változata. A modern normáknak megfelelő arculattal, továbbá új, kiegészítő funkciók is implementálásra kerültek.

3.2 Zabbix

Sensuhoz hasonlóan, a Zabbix is egy nyílt forráskódú felhő monitorozó szoftver. Ugyanúgy, mint társai, nem igényel liceskulcsot, viszont a fejlesztése nem közösség alapú, hanem zárt. A fejlesztő vállalat az európia székhelyű Zabbix LLC, Lettországban.

Világszerte elismert márkák és szervezetek alkalmazzák. Több mint kétszázötven partnerrel áll kapcsolatban. Nemcsak felhőben, hanem lokálisan is futtathatjuk, skálázhatósági limit nincs. A legkisebb egységtől a legnagyobb hálózatig használhatjuk problémamentesen. Nagyságát jelzi, hogy a már meglévő rendszerünket tudjunk bele integrálni, továbbá több száz előre definiált hivatalos sablonnal rendelkezik. Használhatjuk ezeket a sablonokat értesítésekhez, (hiba)jegykezeléshez, IoT és IT szolgáltatásmenedzsmenthez. [7]

3.2.1 Architektúra



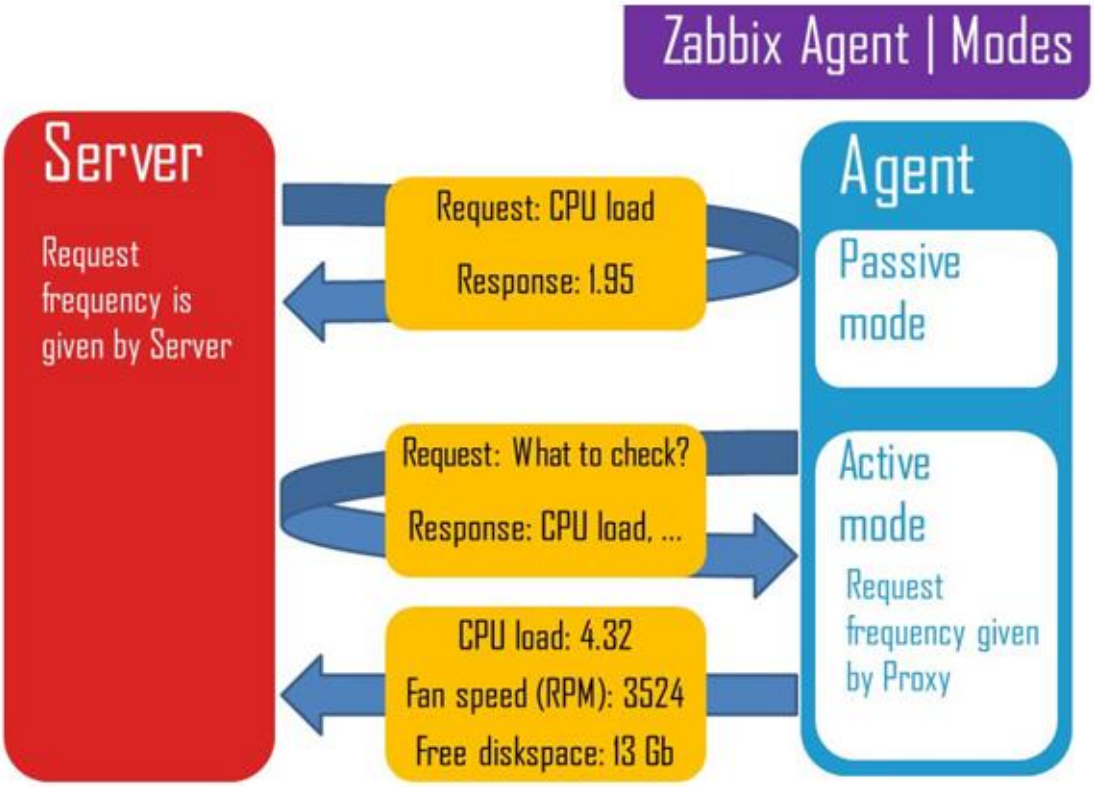
3.3. Ábra Zabbix architektúra [7]

A Zabbix architektúrája is több komponensből áll össze. Rendszer középpontjában a Zabbix szerver áll. Ez a fő alkotóeleme a rendszernek. Az ágensek ennek a résznek továbbítják a megfigyelt adatokat. Egyben egy központi adattár célt is szolgál, amely tárolja a konfigurációs és a működéshez szükséges adatokat. A külső elérést webes interfészeken keresztül kerül biztosításra. Ezek az interfészek is a Zabbix szerverbe vannak integrálva, melyek általában ugyanazon a fizikai eszközön futnak, ahol a szerver maga is. A Zabbix adatbázis is központi szerepet tölt be. Hozzá kapcsolódik a szerver, valamint a megjelenítési réteg is. A távoli eszközök monitorozása egy tűzfallal van elválasztva, amely mögött helyezkedik el az adatgyűjtő Zabbix Proxy. [7]

3.2.2 Ágens

Zabbix szerver sikeres telepítése esetén, a rendszer elkezdi magát monitorozni. Ahhoz, hogy ne csak az adott eszközre telepített szervert érjük el, és tudjuk megfigyelni, el kell érnünk azokat. Erre két megoldás alkalmazható. Az egyik a kevésbé gyakori, mikor SNMP protokollon keresztül kérjük le az adatokat az adott eszköztől. Ezek általában valamilyen hálózati berendezések, nyomtatók, switchek, routerek, amelyeken értelmetlen lenne új operációs rendszert létrehozni és futtatni.

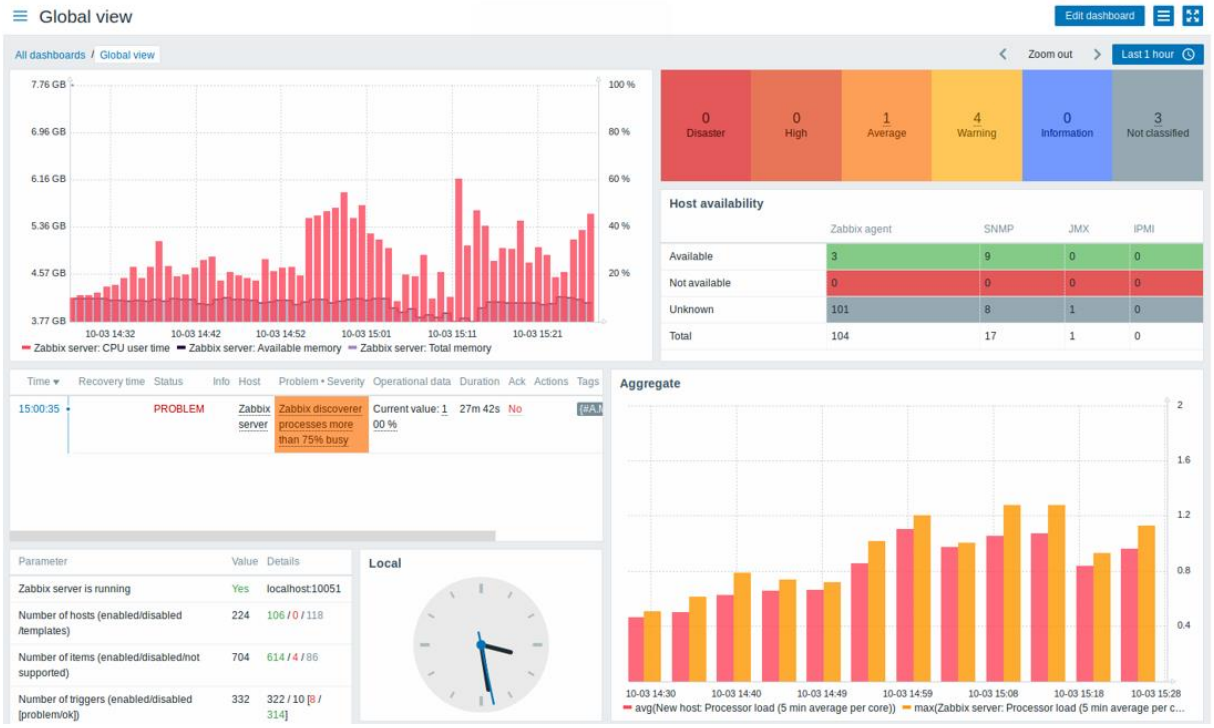
A másik lehetséges megoldás a Zabbix ágens alkalmazása. Ennek telepítésével gyűjthetünk adatokat más eszközökről, mely számos platformot és operációs rendszert támogat, köztük a Linux, Windows és Unix rendszereket. Olyan hardveres és hálózati adatokat képes gyűjteni, mint a CPU, memória, merevlemezek vagy a hálózati interfészei az eszköznek. Kicsi erőforrásigénye miatt, szinte észrevétlenül működhet. A konfigurációs beállításokat egységesen a szerver irányítja, ezzel segítve a rendszer működését és kezelését. Ennek a kommunikációnak a működése kétféleképpen is történhet. Az egyik ilyen a rendszeres lekérdezés (polling) alapú, avagy passzív információcsere. Ebben az esetben a szerver vagy a proxy kérést küld az ágens felé, majd ezután az ágens feldolgozza a kérést, és visszaküldi a kért adatokat. A másik megoldás, az aktív ellenőrzés (trapping). Itt az ágens intéz kérést a szerver felé, lekér egy listát, amely tartalmazza az aktív ellenőrzéseket, majd ezeket megválaszolva időzítve visszaküldi. A 3.4 ábrán kerül szemléltetésre az ágens kommunikációjának imént említett két változata.[8]



3.4. Ábra Zabbix ágens architektúra [9]

További funkciója a naplózás megfigyelése. Támogatja a szöveges, valamint a Windows esemény alapú naplózó monitorozását is. Ezekből, minthogy az eddigiekből is, készíthetünk kimutatásokat, diagrammokat. Folyamatos a megfigyelése ezen elemeknek is. Amint egy meghatározott érték előjön, a szerver automatikusan értesítés kap, amely beállítástól függően vagy automatikusan beavatkozik a rendszer, vagy értesítést küld a megadott felhasználói csoportnak.[9]

3.2.3 Kezelőfelület



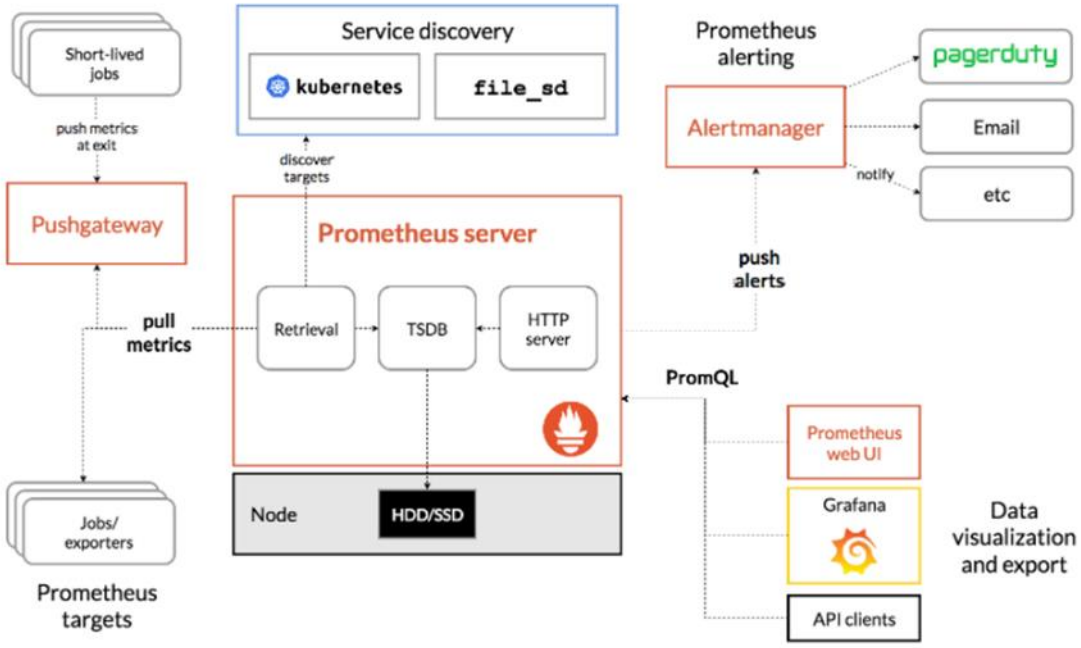
3.5. Ábra Zabbix kezelőfelület [10]

Széles körben alkalmazott megjelenítési felület. A felület különálló blokkokból tevődik össze. Mind a blokkok elhelyezését mind pedig a felületen megjelenítendő blokkokat is tetszés szerint személyre szabhatjuk. Template(sablon) alapú. Előre meghatározott nézőpontok, megfigyelési konstrukciók állnak rendelkezésünkre, melyeket felhasználhatunk. Ezek a sablonok előre konfiguráltak, szerkesztési lehetőségei minimálisak. Ezek konfigurálása viszont széleskörű. Grafikonok mellett széles körben megtalálhatóak táblázatok, diagramok, térképek és egyéb egyedi megjelenítésű blokkok. [10]

3.3 Prometheus és Grafana

A Prometheus olyan nyílt forráskódú eszköztár, amely rendszerek monitorozására és az ezekhez kapcsolódó értesítések kezelésére lett létrehozva. Eredetét tekintve, 2012-ben jelent meg, a SoundCloud részeként. Sikerét és nagyságát jelzi, hogy létrehozás után sorra használták fel a különböző szervezetek, olyannyira, hogy 2016-ra különálló projektté nőte ki magát. Sikerét nagyban köszönheti adattárolási metodikájának. Különleges adatfeldolgozási módszere az idősoros adatgyűjtés. Ez olyan idő alapú adatbázist jelent, mely előre meghatározott időnként, biztosítja az adatok tárolását. Ez a funkció magával hordozza az adattárolás flexibilitását, ezzel az érhető el, hogy adatbázisunkat nem szükséges integrálni a platformba. Tárolástól és formátumtól függetlenül gyűjti össze adatainkat.[11]

3.3.1 Architektúra



3.6. Ábra Prometheus architektúra [11]

Az 3.6 ábra illusztrálja a rendszer architektúráját, valamint a fontosabb kiegészítő részeit. Több különálló komponensből áll, mely jól jelzi a rendszer moduláris felépítését. A komponensek túlnyomó része Go-ban íródott, mely biztosítja a komponensek gyorsaságát, valamint az egyszerű integrálást mást modulokhoz, rendszerekhez.

A rendszer lelke a Prometheus szerver, mely az adatok feldolgozásával és tárolásával foglalkozik, a korábban említett módszer alapján. Működése biztosítva van akkor is, amikor a rendszerünk egyes részei nem működnek. Minden szerver különálló egységként funkcionál. Működése nem függ semmilyen távoli szolgáltatástól vagy hálózati háttértárról, ezzel maximalizálva a monitorozás pontosságát.

Az értesítő menedzser kezeli a szerver által küldött értesítéseket. Ez a rész felelős az értesítések sokszorosításáért, csoportosításért, valamint a céleszközök megfelelő útvonalának kiválasztásáért.[11]

Az információk és metrikák vagy közvetlenül jutnak el a megfigyelendő eszköztől a szerverig, vagy a rövid élettartalmú feladatok push gateway-en keresztül teszik ezt. Ezeknek az adatoknak és értesítéseknek a vizuális megjelenítésére több lehetőségünk is van. Az alapértelmezett beépített keretrendszer mellett számos más külső szolgáltatást vehetünk igénybe. Az API kliensek által biztosított Grafana virtualizációs multiplatform áll leggyakrabban kapcsolatban a Prometheussal.

3.3.2 Ágens

Magába a Prometheusba integrált, viselkedését tekintve úgy működik, mint egy szerver. Egy pull alapú mechanizmus, mely HTTP végpontokon keresztül éri el az adatokat.
Saját lokális adatbázissal rendelkezik, melynek indexeléséhez a LevelDB-t használja. Ezt a Google által létrehozott kulcs-értékpár tárolási rendszert úgy hozták létre, hogy rendezett struktúrát biztosítson, a szöveg alapú kulcsok és a szöveg értékek által. A hatalmas adathalmazhoz pedig saját tárolási réteg áll rendelkezésre, mely az adatokat feldarabolja egységes, 1024 byte méretű darabokká. A háttértáron ezek az adatok pedig egy-egy időszelethez vannak hozzárendelve.[12]

3.3.3 Grafana Kezelőfelület

A Grafana egy nyílt forráskódú platform. Olyan webapplikáció, mely fő célpontja az analitika és az interaktív vizualizáció. Webes információkon alapuló grafikonokat, diagramokat, értesítéseket biztosít számunkra. Megjelenítési felületének köszönhetően gyakran használják felhők monitorozására is.

3.7. Ábra Grafana kezelőfelület [13]

A Prometheus idősoros architektúrájához jól illeszkedik, a különleges adattárolási módszerének köszönhetően a legkülönfélébb adattípusokból képes megjeleníteni adatokat. Legyen az pontokban, vonalakban, oszlopokban vagy egyedi grafikonokban kifejezve, megjelenítve. Megjelenítési tárházi szinte beláthatatlan. Előre elkészített blokkjait saját igényeinkre szabhatjuk. A rengeteg opció mellett lehetőségünk van saját, egyedileg fejlesztett blokkok fejlesztésére is, amit felhasználhatunk a megjelenítés során. Ezt a 7.0 Grafana, NodeJS és yarn fájlok segítségével tehetjük meg.[13]

3.4 Monitorozó technológia ismérvei

A korábban említett előnyök mellett számos más előny és szempont is közrejátszik az eszközök kiválasztásánál. Manapság a biztonság lett az elsőszámú döntéshozatali pont szervezeteken belül. A folyamatos karbantartás, frissítés és fejlesztések segítik a biztonság megőrzését. Alapvető elvárássá vált az újonnan érkező és feltárt általános, nagy tömegeket érintő biztonsági rések szinte azonnali hibaelhárítása, frissítések által. A megfigyelendő adat típusa, a fizikai vagy virtuális adattforrás, amelyről az adatokat szeretnék leképezni, valamint ezek forrása fontos szerepet játszik a kiválasztás során. Az eszközök népszerűsége segít a követőbázis növekedését, a fejlesztések folytonosságát, valamint a szakmai segítségnyújtást. A monitorozó felépítése mellett, a hozzá tartozó ágens kiválasztása kulcsfontosságú.

A megfelelő eszköz kiválasztása során, számos más kiválasztási szempont mellett, fontos tisztába lennünk az esetlegesen felmerülő hátrányokkal is. Ekkor tudjuk csak a számunkra legmegfelelőbb szolgáltatást, eszközt kiválasztani.

3.5 Felhő monitorozás hátrányai.

A legtöbb felhő szolgáltató rendelkezik beépített adatvédelemmel, tűzfallal, általános értesítési rendszerekkel, hogy biztosítsák a felhasználókat adataik biztonságáról. Mindazonáltal ezek az adatok nagyobb veszélynek vannak kitéve ezekben az adatközpontokban. Potenciális kockázatok felléphetnek, mint például az adatvesztés, adatszivárogtatás, felhasználó profilok feltörése, végpontok támadása, technológiai sérülékenységük különösen a megosztott környezetekben. A különböző adatvédelmi szintek eltérhetnek egyes szolgáltatóknál. Ahhoz, hogy monitorozni tudjuk adatainkat, az élő rendszer működésébe is bele kell látnunk, ami komoly IT biztonsági kérdéseket feszeget.

A felhő szolgáltatás, ugyanazokkal a problémákkal találhatja magát szemben, mint bármelyik másik IT rendszer úgymint az újraindítások, frissítések, hálózati gondok vagy áramkimaradások. Fizikai eszközök révén ezek szinte megkerülhetetlenek. Távol lévő eszközökről beszélve, ezekre ráhatásunk sincsen.

Az irányítás nem a mi kezünkben van. A szolgáltató rendelkezik a hálózat felett, mely mindig egy szinttel a mi adminisztrátori szintünk felett lesz. Végfelhasználóként, módosíthatjuk alkalmazásainkat, monitorozhatunk és kinyerhetünk adatokat, viszont magasabb szintű beavatkozáshoz, mint például tűzfalkezelés nincs jogosultságunk.

A monitorozás erőforrás igényes feladat. Minél jobban és minél részletesebben tesszük ezt, annál több számítási kapacitásra van szükségünk, ami már azt is jelentheti, hogy nagyobb erőforrást allokálunk a megfigyelésre, mint magára a rendszerünkre. Látható, hogy fontos specifikálni azokat a kulcs szempontokat, ami alapján a rendszerünket vizsgáljuk. Túl sok szempont leterhelheti a rendszerünket, túl kevés pedig a kockázatokat növeli. Meg kell találni azt az egyensúlyt, ahol még hasznunkra válnak ezek a szolgáltatások úgy, hogy érdemben ne rontsa a fő tevékenységet.

3.6 Ismertetett monitorozó szolgáltatások összehasonlítása

A 3.1 táblázatban összefoglalásra kerülnek az ismertetett monitorozási technológiák implementációi. Látható, hogy célközönség szinten van a legnagyobb eltérés, mely a kezelőfelület kialakításban is látszódik. A technológiák népszerűsége erős korrelációban áll az architektúrális felépítésükkel, jelen esetben a letagoltabb felépítésű a legnépszerűbb eszköz is.

	Sensu	Zabbix	Prometheus és Grafana
Népszerűség	Átlagos	Népszerű	Legnépszerűbb
Célközönség	Olyan fejlesztők, mérnökök, akik a hálózatuk, alkalmazásaik, szolgáltatásaik mélyreható vizsgálatot igénylik	Vállalatok és szervezetek nagyságtól függetlenül	Adatközpontú csoportok
Architektúra felépítés	push	pull	push
Ágens	Borune shell/cmd	SNMP/Telepített	HTTP végpontok
Kezelőfelület	Kismértékben konfigurálható	Személyre szabható	Teljes mértékben személyre szabható (Grafana)
Architektúra	Monolitikus	Moduláris	Moduláris
Kód nyelve	Ruby	C	Go

3.1. Táblázat monitorozó szolgáltatások összehasonlítása

A Github verziókezelő weboldal alapján kaphatunk egy általános képet az eszközök népszerűségéről. Ezen az oldalon csillag formájában jelezhetik a felhasználók támogatásuk és jelenlétüket az adott eszköz iránt. Az oldalon szembetűnő különbség, hogy míg a Sensu-nak hétszáznyolcvan, a Zabbix-nak kettőezerkettőszáz, addig a Prometheus-nak negyvenkétezer csillag szerepel a hivatalos adatlapja felett, mint értékelés. A verziókezelés és frissítések között is érezhető különbség található. A frissítések dokumentációja, azok részletessége és mérete is hasonlóan aránylik az előzőekben mutatott tendenciákhoz. Fontos azonban megjegyezni, hogy összetételében és a rendszerek nagyságában is a Prometheus Grafana páros mondható a legnagyobbnak, ez azonban nem tükrözi a többi bemutatott eszköz használhatóságát és jóságát. A Zabbix szolgáltatás összetett sablon alapú megoldással szolgál számunkra, amely a meglévő szaktudás után segít számunkra a monitorozási rendszer konfigurálásában. A Sensu konfigurálási hatékonyságát kompenzálja, az egyedülálló értesítésirendszere, amely biztosítja versenyelőnyét társaihoz képest. Széleskörben elismert és támogatott a különböző kommunikációs csatornákhoz való integrációja. Ezek alapján, az implementáláshoz számunkra a legmegfelelőbb megoldást kerül kiválasztani, ez pedig a Sensu monitorozó eszköz.

4 MONITOROZÁSI PARAMÉTEREK ISMERTETÉSE

A tervezés megkezdése előtt fontos tisztázni, hogy mi alapján tervezzük meg a rendszerünket. Fontos szem előtt tartani az igényeket, vagyis azt, hogy mi a célja a monitorozásnak. Más és más felhasználási terület más igényeket támaszthat a rendszer felé. Más aspektusok, más nézőpontok merülhetnek fel egy-egy tervezés során. A megfigyelendő terület méretétől és mennyiségétől függően általában két megközelítés közül választhatunk. Komplex és nagyobb rendszereknél nem érdemes mindenre kiterjedő, átfogó monitorozó rendszert kiépíteni. Túl sok paraméter megadása esetén fennállhat a kockázat, hogy megfigyelésnek nagyobb a rendszerigénye, mint magának a monitorozandó szolgáltatásnak. Ilyen szintű komplexitásnál érdemes az egyes megfigyelési csoportokat külön kisebb egységekre bontani és úgy alkalmazni ezeket. A másik lehetséges verzió a kisebb, kevesebb erőforrást igénylő rendszerek monitorozása úgy, hogy ezek a paraméterek egyszerre egy helyen, egy konfiguráción belül helyezkednek el. Így kaphatunk egy összképet a kívánt paraméterekkel.

A paramétereket több kategóriába sorolhatjuk. Általánosságban elmondható, hogy érdemes olyan paramétereket megfigyelni, amelyekről gyorsan és egyértelműen megállapítható a rendszer állapota, továbbá érdemes a megelőzésre nagyobb hangsúlyt fektetni. Ezáltal könnyebben kiszűrhető az eszközök vagy alkalmazások potenciális hibája, a biztonsági rések, ezzel pénzt és energiát megspórolva.

Ezek alapján négy fő szempont lett kiválasztva melyek a tervezés során lesznek megvalósítva:

- Fizikai paraméterek mérése host gépenként. Ezen belül a kiválasztott paraméterek a virtuális CPU kihasználtság, a memória és azon belül is a RAM kihasználtsága, valamint a hőmérséklet, és a háttértár terhelése.
- A szoftveres paraméterek mérésénél a OpenStack szolgáltatások egyesével, Ceph állapota, valamint az ágensek, API-k elérése.

Ezen megfigyelési szempontok által egy általános képet kaphatunk a megfigyelni kívánt rendszerünkről. Tartalmazzák a hardveres meghibásodásokra figyelmeztető fő szempontokat, az alkalmazás szintű komponensek elérését, valamint a biztonsági megközelítésű felhasználó megfigyelést.

5 TERVEZÉS

Az előző fejezetben felsorolt paraméterek alapján kerül megtervezésre és majd kivitelezésre a megfigyelési rendszerünk.

A megvalósítási lépések a következőek:

1. A megvalósítás külön fizikai gépen történik, mely hatékonyan képes kezelni a beérkező adatmennyiséget az adott mérési pontok számának megfelelően. Ezek alapján meghatározásra kerül a fizikai gép kívánt típusa (memória, CPU, hálózati interfészek sebessége, diszkméret). A kiválasztott fizikai gép operációs rendszerének telepítése távoli konzolos eléréssel, alapvető biztonsági beállítások (SSH-kulcs, belépési jogosultságok). Adatok redundáns tárolásához szoftveres lemeztükrözés legalább (RAID 1) kialakítása.

2. A hálózati konfigurációs beállításoknál a fő szempont a korábban már többször említett biztonság, ehhez tartozóan konfigurálni szükséges a hálózati eszközöket megfelelő szeparált menedzsment VLAN kialakítást figyelembe véve. Menedzsment tűzfalas beállításokkal fizikai gép számára elérhetővé tenni a monitorozandó klienseket.

3. A leírást a fizikai gép telepítéséhez, hálózati konfiguráció legfrissebb állapotát git repozitórium fogja tartalmazni. Ez segítséget nyújt a rendszer dokumentáltságának legfrissebb verzión tartására, illetve a fejlődést is jól szemlélteti. Továbbá a megfelelő forráskód letöltése is ezen keresztül érhető el.

4. A felsorolt előnyök, hátrányok és monitorozási téziseket szem előtt tartva kerül megvalósításra a monitorozási kliens. Ahhoz, hogy megfigyelt adatok feldolgozásra kerüljenek a monitorozó szerver által, szükséges a kliens felvétele a rendszerbe. Ennek hatására a kezelőfelületen megjelenik az újonnan felvett kliens a vizsgált paraméterekkel.

5. A tesztelés és annak előkészítése elhanyagolhatatlan része a tervezésnek. A fenntarthatóság és a biztos működés alapeleme. Az implementált részek ellenőrzésének legfőbb szempontjai a biztonság, a helyes működés, valamint a konfigurációs beállítások ellenőrzése.

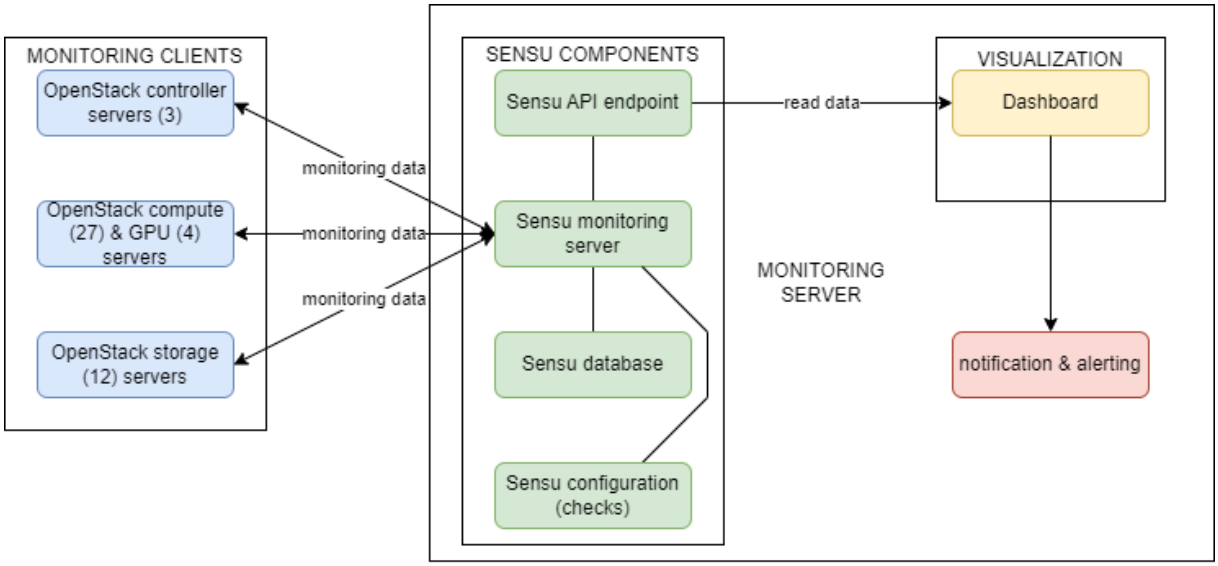
6. Miután az összes kliens felvételre került, és megtörtént a tesztelés, finomhangolás következik a tesztelési eredmények figyelembevételével. Ennek célja az, hogy az üzemeltetési csapat már egy letisztult és működő környezetet vehet át.

7. Végül pedig az üzemeltetés kérdéskörének vizsgálata és szabályok implementálása. A jövőbeli fenntarthatóság és a dinamikus üzemeltetés előkészítése. Ez magába foglalja

mind az esetleges eszköz és egyéb fizikai paraméterek változásának követését, mind pedig a humánerőforrás-változás biztonságos kezelését.

A fenti pontokon végig haladva a rendszer megvalósításra kerül. Fontos szempont, hogy a kialakított rendszer üzemeltetőitől visszajelzéseket be lehessen építeni a rendszerbe, ezzel még hatékonyabbá tenni a felhő működését.

A kidolgozott tervezést figyelembe véve a megvalósításhoz a Sensu szolgáltatásra esett a választás. Az 5.1 ábrán látható a megvalósítandó rendszer architektúra ábrája



5.1. Megvalósítandó rendszer architektúra

A Wigner FK-ban valós fizikai eszközök kerülnek megfigyelésre a megadott paraméterek alapján. Ezekről a korábban említett kliensek, ágensek szolgáltatják az információkat a Sensu szerver számára. Feldolgozás után eltárolásra kerülnek az adatok a Sensu adatbázisban. Az adatok megjeleníthető formátumában történő közlésére a Sensu API gondoskodik számunkra. Ez biztosítja a megjelenítési felület számára a megfigyelt adatokat. Ezekről, és az esetlegesen felmerülő problémákról, hibaüzenetekről értesítés kerül kiküldésre és megjelenítésre a Slack grafikus felületén.

6 IMPLEMENTÁCIÓ

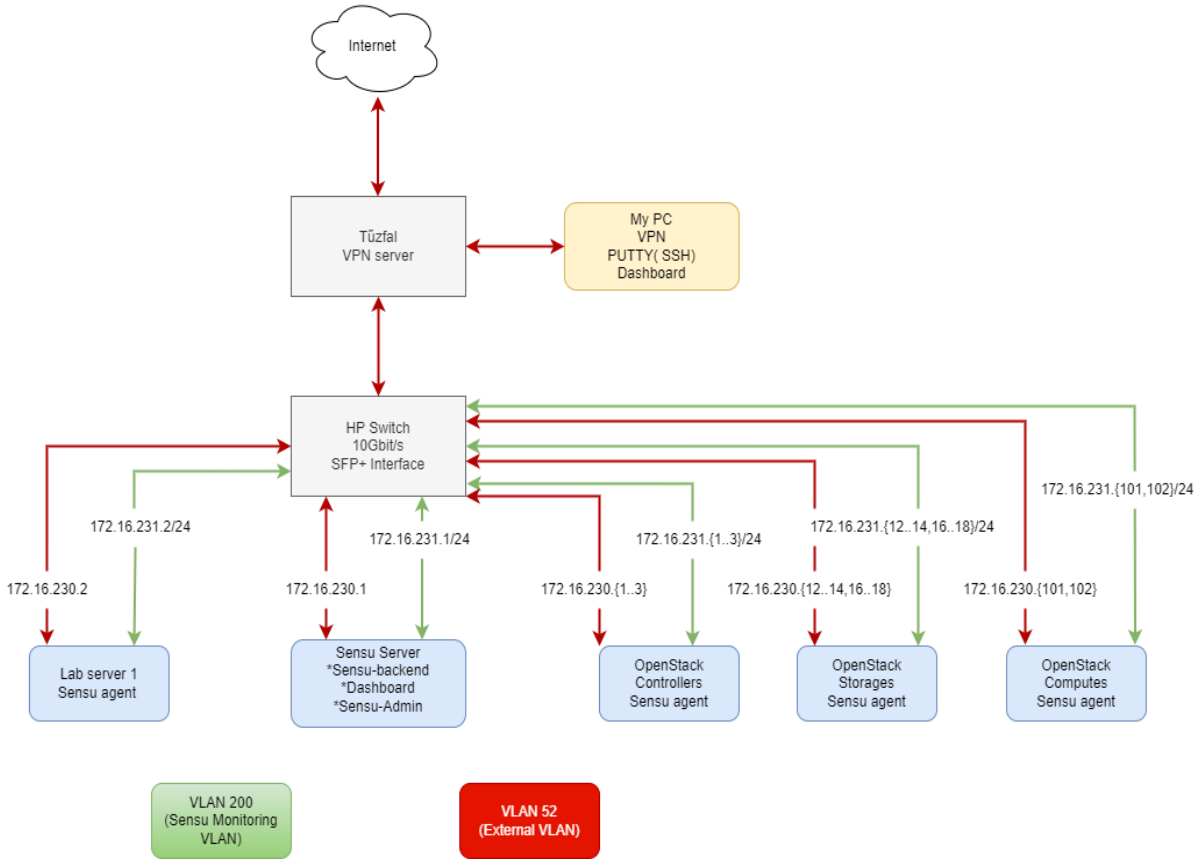
Ebben a fejezetben a tervezés során meghatározott elemek kerülnek létrehozásra és megvalósításra. Az implementáció célja a különböző komponensek telepítése és konfigurálása. A rendszer alap elemeinek meghatározása, továbbá mind fizikai mind pedig a logikai összeköttetésük. Részletezésre kerülnek a konfiguráláshoz szükséges kiegészítő elemek, a felhasznált metodikák, az ellenőrzések és azok funkciói, valamint az értesítési integráció. Leírásra kerülnek azok a szempontok is, amelyek az egyes eszközök, szoftverek telepítéséhez kiegészítő információkkal szolgálnak. A megvalósítás során a továbbfejlesztési lehetőségeket, valamint a karbantarthatóságot figyelembe véve kerülnek kidolgozásra az egyes megoldások. Dokumentálásra kerülnek a rendszer főbb eleminek parancssoros végrehajtása, valamint a fő konfigurációs állományok is.

6.1 Fizikai infrastruktúra bemutatása

A rendszer alapjait biztosító fizikai gépek és hálózatok a Wigner FK telephelyén lettek kialakítva. Külön, erre a feladatra dedikált eszközök kerültek definiálásra és konfigurálásra. Az infrastruktúra összetételét szerverek, virtuális hálózatok és adatátviteli kapcsolók biztosítják.

Az eszközök egy valós, működő rendszerbe lettek implementálva és konfigurálva. Ezáltal az eszközök kapcsolata és elhelyezkedése is fizikailag meghatározott volt. Az eszközök csatlakozását a hálózati eszközökhöz fizikai interfészek biztosítja. Ahhoz, hogy az információ a megfelelő irányba haladhasson, tehát a monitorozó eszközök kommunikálni tudjanak egymással, a forgalomirányítás szükséges

Az fizikai összeköttetést az eszközök között, egy nagy teljesítményű HP hálózati kapcsoló szolgáltatja. Ez az eszköz képes akár a portjain a 10Gbit/s sebességátvitelre. Hatalmas adatmennyiségek feldolgozásához alapvető a nagy sebességű rendszerek és hálózati elemek megléte. Az egyre növekvő adatmennyiségek megkövetelik az olyan mértékű fizikai sebesség meglétét, melyek garantálják a hálózaton áthaladó adatok problémamentes továbbítását a kommunikációs csatornán. Amint a sebesség szűk keresztmetszetté válik, úgy a monitorozó rendszerek működése is nagyobb kockázatnak vannak kitéve. Mind biztonsági szempontból, egy illetéktelen behatoló esetén, mind pedig egy rendszer vagy fizikai meghibásodás esetén. Napjainkban, egyre elterjedtebbé válik az adatközpontokban, hálózati laborokban, szervertermekben a tíz, valamint a száz Gbit/s sebességű eszközök megléte. Míg korábban, gigabites sebességű jelátvitelt elegendőnek bizonyult, az már bebizonyosodott, hogy a növekvő adatmennyiségnek csak az idő és a technológia szab határt. Egyre nagyobb és nagyobb sebességátvitel szükséges az idő előrehaladtával. Fontos azonban figyelembe venni, az igényeken és a felhasználási módokat is, indokolatlanul nem érdemes sokkal gyorsabb és nagyobb hálózatot, hálózati eszközöket biztosítani, mivel további kockázatokat rejthet magában. Ár érték arányban a profitalitás nem feltétlen biztosított, továbbá az eszközök élettartalma is véges, így akár egy élettartalma alatt, kihasználatlan eszköz is kerülhet leselejtezésre. Mindezekért, szükséges a rendszerbe a nagy hálózati kapacitás.



6.1. Architektúra ábra

Az adatátviteli kapcsolón definiálásra kerültek VLAN-ok, tehát virtuális helyi hálózatok. Ezáltal a fizikailag eltérő helyen elhelyezkedő eszközöket, logikailag egy szegmensbe, egy csoportba tudjuk sorolni. Az 6.1 architektúra ábrán két hálózat látható. Az egyik a pirossal jelölt 52 hálózat, mely biztosítja a rendszer számára a kommunikációt a külvilággal. Ez a hálózat felelős az internetelérésééért, mely mind a Sensu, mind pedig az OpenStack működésének elengedhetetlen része. A másik hálózat, a zölddel jelölt 200-as hálózat, mely belső kommunikáció részeként lett definiálva a Sensu számáram, elkülönítve az internetes forgalomtól. Ezáltal külön erre a kapcsolatra definiált logikai szegmensben történhet csak az adatátvitel. Elősegítve így a külső támadások ellen védelmet, valamint az egyéb rendszerektől elszeparált működést is elősegíti, mely védi az adatok konzisztenciáját. További előnye, hogy az internetes hálózat terheltségétől függetlenül, képesek a rendszerbe lévő eszközök az adattovábbításra.

A fizikai csatornák kialakítása után sor került a hálózat IP cím kiválasztására is. A hatalmas, adminisztrátori hálózati infrastruktúra figyelembevétele mellett, került kialakításra az eszközök címkiosztása. Az 52 VLAN a 172.16.230.0/24 hálózatot kapta. Míg a 200-as a 172.16.231.0/24 hálózatot.

A Sensu-backend központi szerepet tölt be a hálózatban. Ez biztosítja az ágensekről érkező adatok összegyűjtését és feldolgozását. Ehhez kapcsolódnak a megfigyelni kívánt

szerverek. Felelős az ellenőrzések futtatásáért, a folyamatok ütemezéséért. Segítségével történik a feliratkozások kezelése, az események és munkafolyamatok helyes használata. Segítségével történik a grafikus felület biztosítása, parancssor és ahhoz kapcsolódó elemekkel történő műveletvégzés, valamint az API-hoz szükséges vezérlés.

A fizikai gépek erőforrás összetétele nagyban befolyásolja a megfigyelendő adatok mennyiségét és esetleges minőségét is. Ahhoz, hogy problémamentes legyen a vizsgálat, valamint bővítési lehetőségek is rendelkezésre álljanak, fontos az eszköz alkotóelemeinek nagy kapacitása. A monitorozó szerverre Quanta típusú gép került kiválasztásra mely, Intel Xeon processzort tartalmaz. Ez 22nm technológiával készült, egyenként 2,6 GHz teljesítményre képes. A memóriát 4x16GB memória modulok adják, így összesen 64Gb bruttó RAM memória áll rendelkezésünkre, a 4TB tárhely mellett.

Ahhoz, hogy ezt az adminisztrátori hálózatot kívülről biztonságosan el lehessen érni, definiálni kellett jogosultságokat, hitelesítésre szolgáló kapcsolatot, valamint a biztonságos távoli elérést. A távoli elérést biztosítására az SSH (Secure Shell) protokoll került kiválasztásra. Helyi és távoli számítógépek közötti biztonságos csatorna kiépítésére fejlesztették ki. Hitelesítésül RSA alapú SSH kulcsok kerültek definiálásra. Ezek a kulcsok felelősek a felek hitelesítéséért. A szerverre történő belépés a publikus/privát kulcspárok segítségével történik. Felhasználók kerültek létrehozásra a tűzfalas rendszerben. Az első belépés előtt eltárolásra kerültek a publikus kulcsok. Felhasználónkként egyedi publikus kulcsok azonosítása történik a szintén egyedi privát kulcsokkal, A publikus kulcsból nem fejthető vissza a privát kulcs. Elküldjük a szerver számára a publikus kulcsunkat, majd azt a mi saját privát kulcsunkkal hitelesítjük, és létrejön a kapcsolat. A publikus és privát kulcsok generálása a Putty nevű szoftver segítségével történt. A szoftver mind a távoli rendszer eléréséhez, mind az SSH kulcsok biztosítására szolgál.

Ahhoz azonban, hogy megtörténjen a jogosultságkezelés, valamint, hogy az SSH kapcsolat végbe menjen, meg kell történjen a Wigner FK tűzfalazási rendszerén a megfelelő beállítások végrehajtása. Virtuális privát hálózaton történik minden esetbe a kommunikáció. Ehhez a OpenVPN nevű szoftver a kétfaktoros bejelentkezéshez, pedig telefonra letölthető FreeOtp applikáció került kiválasztásra, mely időalapú tokent generál.

6.2 Implementációs lépések

A fizikai rendszerek megléte után, az alapvető szoftveres komponensek implementálás volt az elsődleges feladat. A monitorozó szerverre, valamint a kliensekre egyaránt Linux operációs rendszer, Ubuntu 20.04 disztribúció került telepítésre. Ezt távoli eléréssel, parancssorból került implementálásra. A telepítésnél fontos szempont a kényelmesség és gyorsaság. Ubuntu disztribúción a Sensu szerver csomagként telepíthető, mely kényelmessé teszi az implementációt.

A 20.04-es Ubuntu egy nyílt forráskódú, hosszú támogatottságot élvező operációs rendszer, melynek támogatottsága 2025-ig garantált. A telepítés úgynevezett képfájlok

segítségével történt. Ennek a verziónak két verziója létezik. Egyik, grafikus felülettel rendelkező, tipikusan személyi felhasználásra, valamint a kimondottat szerver környezetbe használt, grafikus felület nélküli változata. Ez a fájl definiálja mindazokat az alap követelményeket, beállításokat, amelyek szükségesek egy ilyen disztribúció telepítéséhez és alap konfigurációjához. Ezzel egyszerűen és könnyedén hozhatunk létre kész, működőképes operációs rendszert.

A korábban kiválasztott monitorozó szoftver, a Sensu, és a működéséhez hozzátartozó komponensei telepítésre kerültek a meghatározott eszközökön. Az eszközökre a Sensu legfrisseb verziója került telepítésre, ez a Sensu GO. A telepítés menete parancssorból történt, külön a Sensu Szerverre, és külön a megfigyelendő ágensekre. Elsődleges lépés, a rendszer központi eleme, a Sensu szerver telepítése. A Sensu alapértelmezett felületét, mely grafikus megjelenítést biztosít számunkra, a 3000-es TCP porton érjük el. Bejelentkezés a telepítést követően, az előre definiált felhasználó és jelszó páros segítségével történik.

6.3 OpenStack fizikai szerverek monitorozása

Ahogy a 6.1 architektúra ábrán látható, az OpenStack-hez kapcsolódó szerverek a jobb oldalon helyezkednek el, a Lab szerver pedig a bal oldalon. Ezzel jelezvén a rendszer tagoltságát, mind a logikai mind a fizikai szempontból.

Privát felhő révén, az adatok sérülékenységén megakadályozása, a működés szeparáltsága, valamint redundáns tárolása kiemelkedően fontos szerepet játszik az infrastruktúrában. Az OpenStack-es alrendszer három különálló, jól elkülöníthető részre bontható. Az első, a Controller névre hallgató szerverek. Itt található a legtöbb OpenStack-, valamint egyéb, a működést segítő szolgáltatás. Az API, és ahhoz szükséges elemek, a megosztott adatbázis kezelő, valamint néhány kiegészítő szolgáltatás. A második a háttértárolást szolgáló Storage. Ezek a szerverek biztosítják az adatok tárolását. A biztonságosabb adatolásért különböző RAID megoldások kerültek definiálásra, mely az azonos adatok duplikálását jelenti, kettő vagy több háttértárra. A harmadik pedig a számítási kapacitást biztosító Compute szerverek. Az esetleges hírtelen felmerülő erőforrás szükségletet ki kell szolgálni. Fő célja, a feladatokon belüli műveletvégzés, valamint az erőforrásigényes feladatok kezelése.

A megfigyelni kívánó szerverekre a korábbiakban említett ágensek telepítése szükséges, mely az adatokat szolgáltatja a Sensu-backend számár. Ezek telepítése mind az OpenStack alatt futó szerverekre, mind pedig a Lab szerver egyaránt történtek. Telepítésük parancssorból, a Sensu hivatalos oldalán található dokumentáció alapján történt. Ahhoz, hogy a kommunikáció létrejöhessen a két fél között, a telepítés követően módosítani kell az ágens konfigurációs fájlát. Ezzel a megoldással tudja a backend azonosítani a megfigyelni kívánt rendszert. Ez jelenti az ágensek feliratkozását a Sensu szerver számára.

A fájl módosítása során három alapvető részt kell módosítani. Az első az ágens neve. Fontos megkülönböztetni a különböző eszközöket, mind a bejövő adatok nyomon követesé,

mind pedig az esetleges hibák elhárítása miatt. Érdemes egyedi elnevezést alkalmazni, hiszen nincsen egyedi névellenőrzés, így előfordulhat, hogy ugyan azt a név kerül definiálásra egy-egy ágens számára.

Névtartományokat hozhatunk létre a backenden. Ezzel csoportosítani tudjuk erőforrásainkat, logikailag szeparálhatjuk őket mely megkönnyíti a munkavégzést. Ahhoz, hogy az adott ágenst hozzárendeljük a megadott névtartományokhoz, jelen esetben, a default(alapértelmezett) tartományhoz, a konfigurációs fájlban a namespace(névtér) mező került átírásra.

A harmadik, és egyben legfontosabb módosítási érték, a backend elérési útja. Meg kell határozni, hogy milyen URL-en lehet elérni a backend-et. Ehhez a backend-url mezőben, megadásra került a Sensu szerver elérési útja. A további mezők opcionálisan megadhatóak és konfigurálhatóak. További biztonsági és kényelmi funkciók érhetőek el, az értékek módosításával. A 6.2 ábrán látható, a Lab szerver konfigurációs fájlának egy részlete, a megnevezett konfigurációs beállításokkal. A további paraméterek az alapértelmezett értékeken kerültek definiálásra.

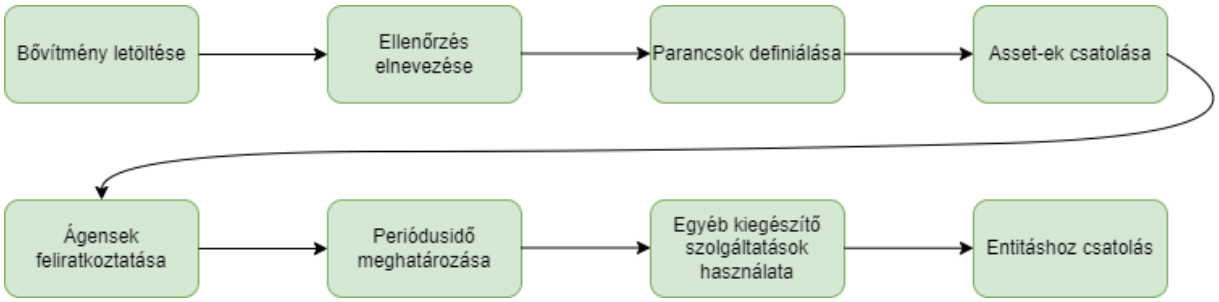
```
# agent configuration
name: "LabServer"
namespace: "default"
subscriptions:
  - myFirstAgent
backend-url:
  - "ws://172.16.230.1:8081"
cache-dir: "/var/cache/sensu/sensu-agent"
config-file: "/etc/sensu/agent.yml"
```

6.2. Sensu ágens konfigurációs fájl

6.4 Check

A Sensu-server számára, az adatok begyűjtését az ágensekről úgynevezett check(ellenőrző) segítségével történik. Elkészítésük történhet manuálisan is, vagy a Sensu által biztosított, közösségi fejlesztések által karbantartott bővítmények segítségével. A bővítmények általában programokat vagy scripteket tartalmaznak, melyek futtatásával különböző információkat nyerhetünk ki a rendszerünkből. Tipikusan három különböző feladatot látnak el ezek az ellenőrzések. Hozzáférnek a monitorozandó rendszerhez. Kinyerik a kívánt adatokat, melyeket értelmezhető formátumba tovább küldenek. Továbbá eldöntik, hogy milyen kategóriákba sorolhatóak. Az ellenőrzések személyre szabhatóak, módosíthatóak. Módosíthatjuk, hogy csak a számunkra fontos információkat jelenítsék meg a felületen. [14] Az ellenőrzések felépítése

közel azonos. Konfigurálásuk történhet parancssorból, illetve a grafikus felületről is, az implementáció során ezek vegyesen történtek. A 6.3 ábrán, a grafikus felületen egy új ellenőrzés folyamatábrája látható.



6.3. Ellenőrzés létrehozása

A webes felületen, a konfigurációs menüfül alatt hozhatunk létre új ellenőrzéseket. A 6.3 ábra alapján, az első lépés, a bővítmények letöltése. Utána kerülhet sor, a konfiguráció a parancsok kiadása. Ezek a parancsok hivatottak az ellenőrzések meghívására. Az bővítmények és internetes dokumentációk alapján kerültek definiálásra ezek a parancsok. A különböző jelzési szintek megadási is itt történik. A parancsok megadásánál történt a bővítményeben szereplő ellenőrzések szelektálása. A ki nem használt bővítményi elemek, melyek jelen esetben nem szolgáltatnának számunkra releváns információt, biztonsági okokból nem kerültek implementálásra. A következő fontos elem, a bővítmények hozzácsatolása az ellenőrzésekhez. Előfordulhat, hogy egyes ellenőrzésekhez több bővítmény vagy több parancs futtatása is szükséges. Ilyen például a Ruby-ban íródott ellenőrzések is, melyek működéséhez a Ruby-runtime asset(eszköz) került hozzáadásra. Ez biztosítja a Ruby-ban írt ellenőrző kódok futtatását. Lehetőség van az ellenőrzések futási idejének maximalizálására, illetve az ellenőrzés kimeneti méretének maximalizálására, ezek viszont nem kötelező értékek. Ahhoz, hogy az ellenőrzések végbemenjenek, szükséges megadni, hogy mely entitásokon történjenek. Itt lehet feliratkoztatni a szervereken futó Sensu ágenseket az ellenőrzésekhez. Az ágensek telepítése során megadott nevek felhasználásával kerültek összekapcsolásra az ágensek az entitásokkal és az ellenőrző egységekkel. Az általános hardveres elemek vizsgálata a CPU, a memória, a háttértárak ellenőrzése az összes entitáson megtörtént.

Működésüket tekintve általában két állapotot lehet definiálni. Az alap, a helyes működés, amikor a kritériumoknak megfelel az ellenőrzés, ilyenkor zöld színnel jelenik meg az irányítópulton. A másik állapot amikor az ágens jelzést küld egy nem megfelelő állapotról. Ilyenkor további két lehetőség van, a sárgával jelzett figyelmeztetés, valamint a pirossal jelzett hiba értesítés. Az ellenőrzések definiálása során, szükséges megadni a hibajelzés szintjeit. Ezek hivatottak beállítani az ellenőrző folyamat milyen értétkek esetén küldjön figyelmeztetés vagy hibajelzést. Különböző értesítéseket és hibaüzeneteket konfigurálhatunk be figyelmeztetések és hibák esetén is. Célja a jelzések szegmentálása, továbbá az értesítések nyomon követése, valamint a végzetesnek számító hibák elkerülése. Ezeket a szinteket, a hardveres ellenőrzések konfigurációjánál, a figyelmeztetésekre, az adott eszköz 75%-os kihasználtságánál történt a beállítás. A hiba jelzésekre pedig ez az érték 85%.

	Lab szerver	OpenStack Controller	OpenStack Storage	OpenStack Compute
Compute-container-group				x
Controller-container-group		x		
CPU-usage	x	x	x	x
Disk-usage		x	x	x
Memory-usage	x	x	x	x
Network-interfaces	x	x	x	x
Smart-disk	x		x	
Storage-container-group			x	
System	x	x	x	x

6.1. Megvalósított ellenőrzések

A 6.1 táblázat szemlélteti a megvalósított ellenőrzéseket. A táblázat magába foglalja az implementált ellenőrzéseket, valamint a definiált szerver típusokat. Az ellenőrzéseket funkciójukat tekintve, több kategóriába sorhatjuk. Lehetőségünk van mind hardveres, mind szoftveres információk begyűjtésére. A 6.1 táblázatban zöld színei jelöli a hardverekre vonatkozó ellenőrzéseket, kék színnel pedig a szoftverre vonatkozókat. Az alapvető hardveres megfigyelések célja kettős. Elsősorban a fizikai meghibásodás elleni védelem a fő cél. Ezek a fizikai eszközök élettartalma is véges, különösen a forgó alkatrészeké. Általában egy ilyen, és ehhez hasonló rendszernek a terheltséges és jóval számottevőbb, mint egy általgos felhasználású, otthoni számítógépé. A rendelkezésre állás az év teljes egészében szükséges, egy esetleges rendszerleállás, akár visszafordíthatatlan következményekkel is járhat. A másik fő szempont, a biztonság növelése egy esetleges kibertámadás ellen. Egy támadás esetén, a támadónak lehetősége nyílik arra, hogy a kapcsolódott eszközök kihasználtságát módosítsák, ezáltal komoly károkat téve a fizikai architektúrában.

A másik nagy kategória, a szervereken is futó szoftveres elemek megfigyelése. Az OpenStack moduláris felépítésének köszönhetően alapvető feladat, a különböző részeinek monitorozása. Ilyen például a számítási kapacitást biztosító nova, mely megjelenik a három szerver csoportnál, továbbá a működéshez szükséges fő elemek, mint a Cinder, Heat, Keystone. A szervereken található szoftverek konténeres megvalósítással kerültek telepítésre. Ahhoz, hogy ezeket a konténereket monitorozni lehessen, külön erre a célra definiált ellenőrzést kell írni, melyek tartalmazzák mindazokat a konténerekbe futó alkalmazásokat, amelyek a fizikai szervereken futnak.

Ahogy a 6.1 ábrán látható a konténerekhez kapcsolódó ellenőrzések csoportosításra kerültek az OpenStack szerverek funkciói alapján. A szerverek felhasználása meghatározza a telepített szoftverek nagyrészét, valamint a montiorozás is ez alapján kerül elkészítésre. Az ellenőrzések csoportosítását indokolja még a Senu rendszer felépítéséből fakadó vizuális megjelenítési korlátok, valamint a rendszerek nagysága. Amennyiben külön ellenőrzéseket szeretnék alkalmazni a szoftveres elemekre, úgy mindegyikhez szükséges egy külön ellenőrzést létrehozni. Ezáltal az ellenőrzéseink tagoltsága, a rendszer nagyságától függhet. Előfordulhat,

hogy a sok ellenőrzés következtében a rendszer nem működik optimálisan, esetleg meghibásodik. Probléma adódhat a megjelenítéssel is, hiszen ahány szoftver, annyi ellenőrzést kéne az adott felületen megjeleníteni és megkülönböztetni. Nemcsak a megfigyelő számára okozhat gondot, hanem a rendszer számára is. Fontos a monitorozási események mértékletes használata, hiszen nem cél a monitorozó rendszer túlterhelése, valamint az ágens túlságos leterhelése.

Az ellenőrzések felépítését tekintve hasonló a korábban ismertetekhez. Az ellenőrzések létrehozása során szükséges definiálni a konténerekbe futó alkalmazásokat. Az ellenőrzésben definiál parancsok segítségével nyílik lehetőségünk az ellenőrzés számára biztosítani azokat a paramétereket, amelyek alapján ellenőrizni tudja a konténerek és azonba futó szoftverek állapotát. Az ellenőrzés célja nem a rendszer felderítése, hanem a megadott paraméterek alapján az információ szolgáltatás. Így pontos információval kell rendelkeznünk arról, hogy az aktuális szerveren milyen szoftverek érhetőek el. Az ellenőrzések futtatásához két bővítmény szükségeltetik. Az egyik ilyen a konténerekben futó szoftverek monitorozásához szükséges. A másik bővítmény pedig a konténerizációs környezet miatt szükségesek. Az implementáció során a szoftverek Docker környezetbe lettek implementálva. Programkód révén, az ellenőrzések csak a mögöttes architektúra ismeretébe képesek az információk kinyerésre.

6.4. Konténerek ellenőrzése

A 6.4 ábra reprezentálja a Compute szervereken futó alkalmazásokat. Az ábrán látható a korábban említett OpenStack komponensek, továbbá egyéb futatott alkalmazások, mint például a Openswitch_db, a Kolla_toolbox vagy a Fluentd.

Ezeknek az ellenőrzések a futtatásával, átfogó képet kaphatunk az aktuálisan futó rendszerünkről. Azonban, egymástól független működés jellemzi ezeket az ellenőrzések. Nem tudnak egymás létezéséről, nem kommunikálnak egymással, nincs közvetett kapcsolat köztük. A fenti ellenőrzések az adott beállításaiknak megfelelően periodikusan mintát vesznek a rendszer állapotáról. Ez a módszer nem pazarolja az erőforrásokat, mégis jó közelítést ad a rendszer egészéről.

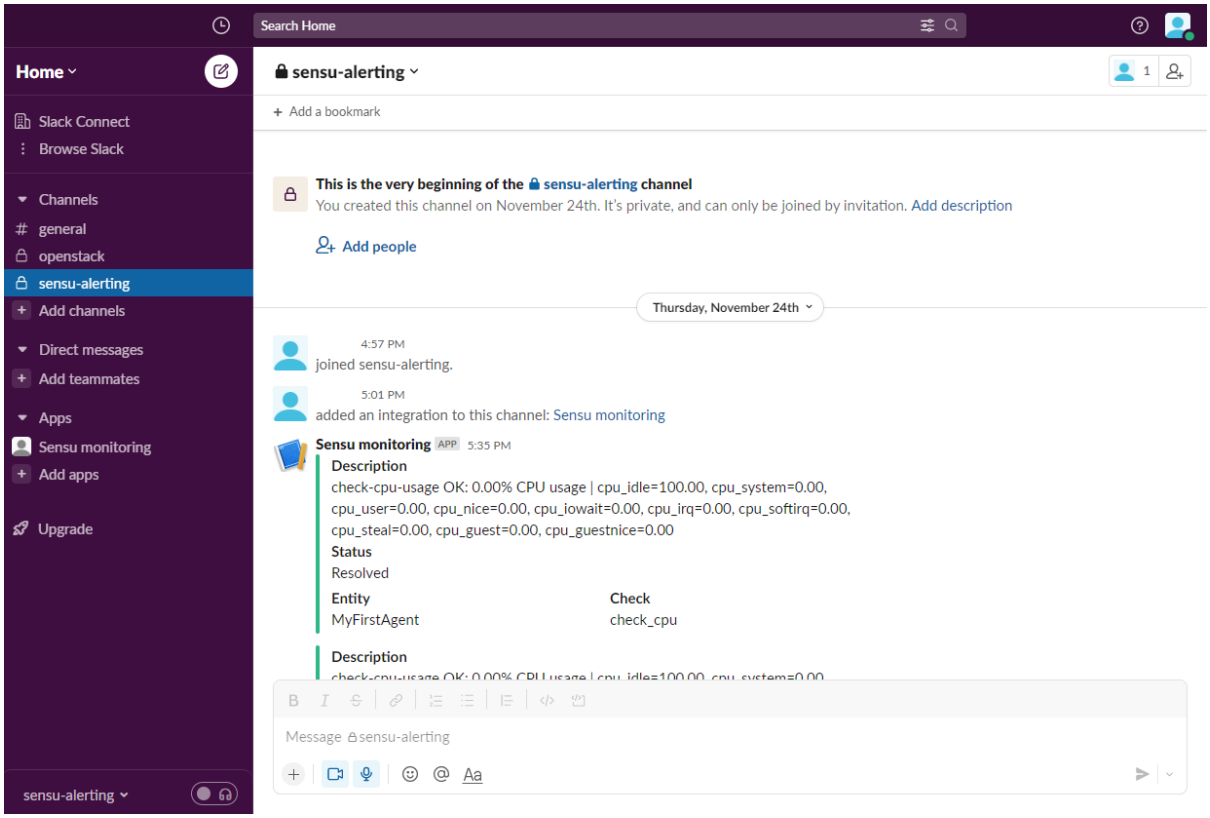
6.5 Értesítési rendszer

A Sensu backend felelős az monitorozó adatok begyűjtéséért. Ezeket az összegyűjtött információkat és az értesítéseket a Sensu irányítópultján tekinthetjük meg. Ahhoz, hogy értesülésünk legyen a hibák és figyelmeztetésekről, az említett felület figyelése lenne szükséges állandó jelleggel. Lehetőség van arra, hogy ezeket az információkat több megjelenítési formában, más felületen jelenítsük meg. Ezzel segítve az adatok átláthatóságát, személyre szabhatóságát, valamint az értesítési rendszer átláthatóságát. Az egyik ilyen megoldás, hogy a megjelenítési felületet lecseréljük. Ez történhet akár a korábban összehasonlított eszközökkel, mint például a Grafanával vagy Zabbix használatával, vagy egyéb, akár egyedi fejlesztésű felülettel is. A másik hasonló megoldás, az értesítések kiszervezése. Ez a megoldás lehetőséget biztosít számunkra, hogy a felületek folyamatos figyelése helyett, egy központi értesítési rendszeren kapjunk meg az információkat. Lehetőség van például email, projektmenedzsment szoftver, valamint telefonos integrációkra is. Ezáltal, beállítástól függően, csak a rendszer változásairól kaphatunk információkat.

A Slack projektmenedzsment szoftver összekapcsolása történt a Sensu backend-el. Ez az internetes alkalmazás biztosítja az értesítési rendszert. Ez a szoftver megjelenhet a kisvállalkozásoktól kezdve, a nagyvállalatokig, mindenhol. A szervezeten belüli kommunikációra, munkavégzésre és annak koordinálására fejlesztették. Az integrációnak köszönhetően, lehetőség nyílik arra, hogy csak a megfigyelendő rendszer változásairól kapjunk értesítéseket. Ezáltal rengeteg időt és humánerőforrást lehet megtakarítani. [15]

Az integráció kétféleképpen történhet. Az egyik ilyen megoldás, a Sensu katalógusból az előre konfigurált kompakt megoldás használata. A Slack Alerts néven elérhető csomag tartalmazza mindazokat a specifikációkat mely szükségesek a kapcsolat létrehozásához. Telepítésével egyszerű és naprakész megoldást kaphatunk. A minimálisnak tekinthető konfigurációs értékeke megadása után, a kapcsolat létrejön, és lehetőség nyílik a kommunikációs feleknek az értesítések küldésére. Előnye a gyors és egyszerű implementáció. Hátránya, a bonyolultabb konfigurációs lépések hiánya. A másik ilyen megoldás, az egyedi telepítés. Ebben az esetben a komponenseket külön-külön szükséges telepíteni és konfigurálni. Ezzel a növelhető a biztonság, valamint a személyre szabhatóság. A megvalósítás első lépése a Slack-csatornák kialakítása. Ahhoz, hogy a rendszer működőképes lehessen, külön, erre a célra létrehozott csatornákat kell létrehozni a Slack grafikus felületén. Néhány kattintás szükséges egy csatorna elkészítéséhez. Beállíthatunk a csatornákhoz jogosultságokat, felhasználókat rendelhetünk hozzá, valamint tovább konfigurálhatjuk a csatornák értesítéseit. Ezeken a csatornákon fognak megjelenni a backend által küldött információk. Több csatorna kialakításával szeparálhatjuk az értesítéseket, átláthatóbb képet alkothatunk rendszerünkről. Ez a felbontás történhet ellenőrzések-ek alapján, vagy akár szerverek, entitások szerint is. Ahhoz, hogy ezek a csatornák kommunikálni tudjanak a backenddel egy úgynevezett appot kell létrehozni a Slack felületén. Mindamellett, hogy külön alkalmazásokat integrálhatunk a felülethez, ezzel a módszerrel tudunk a csatornákra egy egységként hivatkozni. Ezekhez a

logikai konténerekhez lehetséges az API-ok hozzárendelése, amin keresztül kommunikál a két fél. A Slack oldali utolsó lépés, az API beállítása és lekérése, mely a Sensu konfigurációja során megadásra kerül. A Sensu oldali kommunikációhoz szükséges a kommunikációs csatorna kialakítása. Elsődleges lépés a Slack bővítményi csomagjának letöltése és telepítése parancssor segítségével. Ezzel a bővítménnyel van lehetőség az adatok küldésére a konfigurált Slack csatornák számára. Telepítést követően az információk küldéséhez szükség van a kapcsolatok kialakításához. Több csatorna esetén több kezelő konfigurációja szükséges. Ezekben a kezelőkben szükséges az API-ok elérési útjának definiálása, valamint a megfelelő Slack csatorna meghivatkozása. A következő lépés a munkafolyamat kialakítása, mely tartalmazza a előre konfigurált kezelőket. Végső lépésként pedig, ellenőrzők feliratkoztatása szükséges a létrehozott munkafolyamatokra. Ehhez az ellenőrzők konfigurációja szükséges. A YML alapú szöveges fájlban, a feliratkozás paraméterek átírása szükséges. A konfiguráció után létrejön a végponttól végpontig terjedő kapcsolat.



6.5. Slack felület

A 6.5 ábrán a Slack csatorna beüzemelése látható. A csatorna létrehozását követően, automatikus beléptetés történt, az adminisztrátori jogosultság miatt. A csatorna privát jogkörrel lett létrehozva, így a korlátozása miatt, a tagságok, valamint a személyek hozzáadása manuális úton történik. Ehhez adminisztrátorként jogkör szükséges, valamint Slack regisztrációs e-mail cím. A csatorna létrehozása után, a felületen megjelenik a korábban említett konténer létrehozásának integrációjára felhívó üzenet. Ehhez az egységhez került hozzárendelésre a kommunikációhoz szükséges API. A Sensu szerver oldali konfigurálás után, pedig megtörtént

az első üzenet megjelenítése a felületen. Ahogyan az ábrán is látható, az első ilyen üzenet, a lefutott CPU ellenőrzésesnek eredménye. Olyan általános információkat találhatunk, mely a CPU fizikai tulajdonságait részletezi, az ellenőrzés állapotát, továbbá, hogy melyik ágensre történt az ellenőrzés.

A Slack csatorna hátránya az adatok lehetséges sérülékenységében rejlik. Jelen esetben az értesítések egy külső, a Slack által biztosított adatbázisában is eltárolásra kerülnek. Ez tekinthető egy biztonsági kockázatnak. Függ a rendszerünk a csatorna rendelkezésre állásától, valamint extra támadási felületet biztosítunk a külső adatbázis révén. Hosszú távon a biztonság maximalizálása érdekében érdemes olyan értesítési rendszert kialakítani, mely adatainak tárolását a szervezet maga tudja biztosítani a projektben részt vevők számára. Az OpenStack privát felhőjének megvalósítása során az adatok belsős tárolás a cél, így a kommunikáció és értesítés rendszer funkcióit is érdemes belsőleg kezelni. Ezáltal az adatokhoz való hozzáférhetőség csökken, a vállalaton belülre korlátozódik. A kommunikációs rendszerek lokális implementációjára használhatunk olyan alternatívákat, mint az Element, a Rocket Chat valamint a Mettermost. Így a rendszerről generált értesítek nem hagyják el az adatközpont belső hálózatát.

6.6 Git repozitórium

Git verziókezelő rendszer került használatra, a tervezés és az implementáció során felmerül dokumentáció és a fejlesztés nyomon követesére, valamint a verziók egységben tartása érdekében, valamint a jövőbeli fejlesztési lehetőségekhez jó kiindulási alapot ad. Ez egy publikusan nem elérhető repozitórium a hozzáférhetősége a rendszer fejlesztői és üzemeltetőire korlátozódik.

A GitHub egy nyílt forráskódú, elosztott működésű, verziókezelő rendszer, mely képes a kis fájloktól kezdve, a nagy nemeztközi fejlesztési projektekig tárolására, az igényeink megfelelően. Használata megjelenik mind a magán, mind pedig a vállalati szférában. Könnyű és gyors használata. Ideális nyílt forráskódú szoftverek, dokumentációk és a rendszerek kisegítésére használt dokumentációk és egyéb komponensek tárolására, valós verziókövetésre. Jelen repozitórium, csak a szakdolgozathoz, és a felépített rendszerhez kötődik. A repozitórium tartalmaz egy általános bevezetést a projekt ismertetéséről, a szakdolgozat teljes anyagát, az elkészített prezentációt, a csatolt képeket eredeti minőségében, továbbá a konfiguráció során használt főbb parancsokat. A konfigurációs mappa tartalmazza, Sensu szerver, az ágensek, az ellenőrzések, és a slack konfigurációit, szöveges fájlokban.

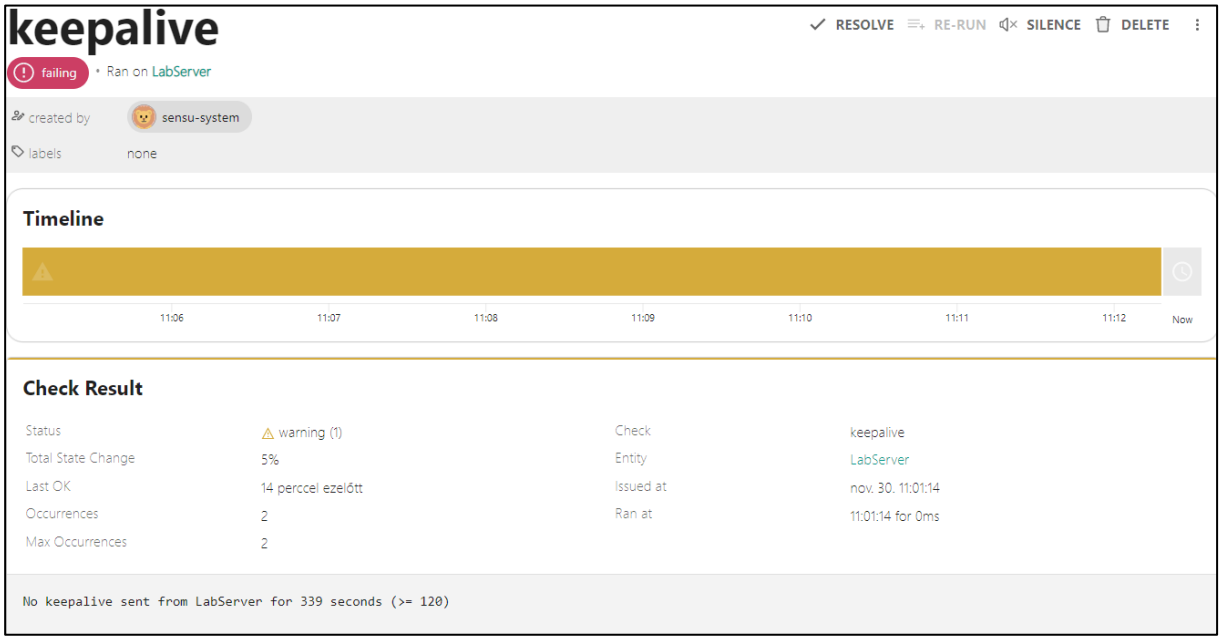
7 TESZTELÉS

A tesztelés elengedhetetlen része egy ilyen és hasonló típusú rendszerek számára. Nem lehet megelégedni azzal, hogy az eszközök látják egymást, és hiba nélkül lefutnak az ellenőrzések. Szükséges ellenőrizni, hogy nem történt-e hiba a konfigurációk során, hogy létrejöttek-e a megfelelő kapcsolatok, valamint, hogy a rendszer képes-e lekezelni a változásokat.

7.1 Kapcsolat teszt

A keep alive ellenőrzés, az ágensek alapértelmezetten, minden huszadik másodpercben lefutó művelete. Ez jelzi a szerverek elérhetőségét. Amennyiben nem kerül kiküldésre, a backend ezt érzékeli, és jelzi. Ilyenkor a probléma igen jelentősre tehető. Néhány esetben előfordulhat, hogy téves jelzést küld a szerver. A leggyakoribb ilyen hiba a rossz idő szinkronizálásából fakad. Ennek elkerülésére végett a Chrony démon került telepítésre mind a backend szerveren mind pedig az ágens szervereken. Ez biztosítja az időszinkronizálást az eszközökön. Ez a program képes a háttérben futni, a rendszerindítással megegyező időben.

A Sensu ágens leállításával történt ennek a tesztnek a szimulálása, mely megfeleltethető egy esetleges szerver leállásnak. A megfelelő parancs kiadásával az ágens megáll. A backend szerveren futó irányítófelületen a keep alive periódusidő lejárta után megjelenik a figyelmeztetés, mely szerint, nem érte el az ellenőrzés az ágenst, így beavatkozásra lehet szükség. Mindaddig fennmarad ez a hibajelzést, ameddig meg nem történik az újraindítás, vagy az ágens el nem távolítjuk a megfigyelni vágyó szerverek közül. A 7.1 ábrán látható a grafikus felületen, az ágens leállításának következménye. A sárga szín reprezentálja a kapcsolat hiányát, továbbá információval szolgál, hogy mióta nem történt válaszküldés az ágens részéről.



7.1. Kapcsolat teszt

7.2 Stressz teszt

Az ágens szervereken történő lokális hardveres eszközök monitorozásra egy nyílt forráskódú, vizuális megjelenítő került telepítésre, a Htop. Ezzel ellenőrizhetjük az éppen futó eszközök hardveres kihasználtságát, valamint összehasonlítási alapként is szolgál számunkra.

Ellenőrizhetjük, az ágensen futó Htop által mért értékeket, a Sensu kezelőfelületén megjelenített értékekkel. A különböző hardveres ellenőrzések, mint a CPU, memória, háttértár, helyes működését stressz tesztel ellenőrizhetjük. Ehhez a kimondottan Linuxhoz használatos Stress eszköz került kiválasztásra és implementálásra. Ezzel az eszközzel érhetjük el a hardveres komponensek teljes meghajtását, amire már az ellenőrzés jelzést is ad. Az ellenőrzések tesztelése történhet külön-külön, vagy akár egyszerre is. A kettő közötti különbség, az erőforrások számában rejlik. Külön tesztelés esetén az egyes elemek nagyobb terhelésigényűek, így több virtuális meghajtást igényelnek. A felhasznál parancs,

```
stress --cpu 10 --io 4 --vm 16 --vm-bytes 1280M --timeout 70s
```

7.2 ábrán a Sensu kezelőfelületén a CPU ellenőrzése látható. Az ágensen történt megnövekedett CPU kihasználtságát jelzi. Az ellenőrzés beállítása alapján, 75% CPU kihasználtság felett figyelmeztetést, 85% felett pedig hibát jelez. Jelenleg, a rendszer kihasználtsága mellet, ahhoz, hogy ezt a két érték felé menjen a CPU kihasználtság, először 24 darab, másodjára pedig 30 CPU-ra elindítására volt szükség. Az ábra idősávjában sárga színnel jelöli, a figyelmeztetést, és pirossal a hibajelzést. A teszt befejeztével a CPU kihasználtság visszatér az eredeti állapotába, így már nincs kritikus állapotban, így az idősávról is eltűnik a megkülönböztetett jelzés, visszatér a normál működésbe.

7.2. Stressz tesz

7.3 Smart teszt

A Smart teszt család által, a háttértárakról kaphatunk átfogó információkat. Eddigiektől eltérően, ez az ellenőrzés képes összeségében ellenőrizni az adattárolókat. Nem csak egy pillanatnyi állapotot mutat a rendszerből, hanem különböző paraméterek által megvizsgálja az adott eszköz eddigi életciklusát. Minden mai háttértár támogatja ezt az ellenőrzési csomagot. Fő célja a lehető legpontosabban előre jelezni az esetleges meghajtó hibákat. [16]

Telepítése és konfigurálása a korábbiakhoz hasonlóan történt. Ahhoz, hogy megbizonyosodjunk róla, hogy a hardveres eszközök támogatják ezt az ellenőrzést, parancssoros telepítést követően ezt ellenőriznünk kell. A Lab szerver kettő darab, kettő terrabájtos háttértárral rendelkezik. Mindkét eszköz támogatja a smart teszt összes funkcióit. Elsődleges információ kinyeréshez, a háttértárak alap tulajdonságait és adatainak kinyeréséhez, a ’sudo smartctl -i /dev/sdc’ parancs kiadása szükséges.

Ennek eredménye a 7.2 ábrán látható. Ezáltal olyan alapvető információkhoz juthatunk, mint például a háttértár típusa, mérete, elhelyezkedése, gyorsasága, valamint, hogy támogatja-e a teszt futását.

```
=== START OF INFORMATION SECTION ===
Model Family:     Hitachi Ultrastar 7K3000
Device Model:     Hitachi HUA723020ALA640
Serial Number:    ********************
LU WWN Device Id: 5 000cca 224f347d1
Firmware Version: MK7OAAK0
User Capacity:    2,000,398,934,016 bytes [2.00 TB]
Sector Size:      512 bytes logical/physical
Rotation Rate:    7200 rpm
Form Factor:      3.5 inches
Device is:        In smartctl database [for details use -P]
ATA Version is:   ATA8-ACS T13/1699-D revision 4
SATA Version is:  SATA 2.6, 6.0 Gb/s (current: 3.0 Gb/s)
Local Time is:    Thu Dec  1 14:40:08 2022 UTC
SMART support is: Available - device has SMART capability.
SMART support is: Enabled
```

7.2. Általános háttértár információk

Ezt a vizsgálatot olyan ágenseken érdemes futtatni, ahol a tárolás az elsődleges szempont. Ezek az eszközök nagyobb használatnak vannak kitéve, így a meghibásodásukra is nagyobb az esély, mint egy olyan egységnél, amely csak másodlagos funkcióját tölti be. Az ellenőrzési és futási ideje is több, mint a többi ellenőrzésé. A futtatása három hetente lett meghatározva, kiemelten az OpenStack Storage szervereken.

Három különböző típusú teszt futtatása lehetséges. Az első ilyen a rövid időtartamú tesztek. Ezek egy gyors mérés alapján feltérképezik a háttértárat, melyek nagy pontossággal képesek detektálni az esetleges hibákat. A hasonlóan működő, hosszú időtartalmú tesztek a másik ilyen csoport. Alapja az előzőhez hasonló, viszont itt a futtatás a lemez teljes felületére lesz alkalmazva és megvizsgálva. A harmadik csoport a szállítási teszt, egy különleges, és ritkán használt ellenőrzést von magában. Szállításkor közben, előfordulhat, hogy a merevlemezek fizikai alkatrészei sérülnek, ezáltal kihatásuk lehet a tárolt adatokra. Az ellenőrzés ezeket a hibákat keresi, és vizsgálja át a teljes adattárolót.

A tesztek konfigurációs fájla az ’/etc/sensu/conf.d/smart.json’ helyen található. Ebben szerepelnek azok a konfigurációs paraméterek amelyik információkat szolgáltatnak számunkra. A teszt futása során, ezek a paraméterek kerülnek vizsgálatra és megjelenítésre. Egyéb kiegészítő információkhoz, vagy a vizsgált paraméterek megváltoztatásához, ennek a konfigurációs fájlnak a módosítása szükséges, adminisztrátori jogosultsággal. A 7.3 ábrán látható, az elsődleges háttértár rövid tesztjének eredménye.

SMART Attributes Data Structure revision number: 16

Vendor Specific SMART Attributes with Thresholds:

ID#	ATTRIBUTE_NAME	FLAG	VALUE	WORST	THRESH	TYPE	UPDATED	RAW_VALUE
1	Raw_Read_Error_Rate	0x000b	100	100	016	Pre-fail	Always	0
2	Throughput_Performance	0x0005	135	135	054	Pre-fail	Offline	86
3	Spin_Up_Time	0x0007	143	143	024	Pre-fail	Always	481
4	Start_Stop_Count	0x0012	100	100	000	Old_age	Always	18
5	Reallocated_Sector_Ct	0x0033	100	100	005	Pre-fail	Always	0
6	Seek_Error_Rate	0x000b	100	100	067	Pre-fail	Always	0
8	Seek_Time_Performance	0x0005	123	123	020	Pre-fail	Offline	31
9	Power_On_Hours	0x0012	094	094	000	Old_age	Always	48801
10	Spin_Retry_Count	0x0013	100	100	060	Pre-fail	Always	0
12	Power_Cycle_Count	0x0032	100	100	000	Old_age	Always	17

7.3. Smart teszt futtatásának eredménye

8 TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

Az elkészített infrastruktúra, ahogy sok másik rendszer sem, nem tekinthető teljesen elkészültnek. Továbbfejlesztési lehetőség számtalanok. Fontos az eszközök, rendszer elemek, és egyéb szolgáltatások rendszeres frissítése. Gyakorta, a támadók kihasználják az elavult, ritkán frissített eszközök sebezhetőségét. Szükséges mind a hardveres, mind a szoftveres frissítések elvégzése.

További lehetőség lehet a naplózás bevezetése és megfigyelése. A naplózás bevezetésével, több lehetőségre is szert tehetünk. Megkönnyíti a hibakeresést a rendszer konfigurációja során. A rögzített adatokból kiderülhet, hogy mely ponton futott a konfigurációnk hibára, esetleg, hogy milyen kiegészítő komponensek szükségesek a beállítások véghezviteléhez. További előnye, a hibák visszakereshetősége. Amennyiben egy hardveres vagy egy szoftver komponens meghibásodik, a tárolt fájlokban hasznos információkra tehetünk szert, mely utalhat a meghibásodás okára. Ezáltal lehetőség nyílik a probléma tüzetesebb kivizsgálására. Nem utolsó sorban, az idegenkezűség általi védelmet is szolgálja. Esetleges illetéktelen behatolás is észlelhető, kezelhető, valamint az esetleges biztonsági rések is felfedezhetőek.

A rendszer alapvető konfigurációjának gyorsítása érdekében lehetőség van szkriptek bevezetésére. Számos, akár nyílt forráskódú szoftver segítségével, az ismétlődő elemek konfigurációja egyszerűbb, gyorsabb, és hibamentesebb implementálása történhet. Erre remek megoldás nyújt az Ansible szoftver. Ennek a szoftvernek a segítségével például, az ágensek telepítését és konfigurációját végezhetjük egyszerre. A telepítési parancsok megadásával, a változó értétkek deklarálásával automatikus telepítést és konfigurációt tartalmazó fájlt hozhatunk létre. Ezáltal nem szükséges az ágensek külön-külön való telepítésé és konfigurációja. Elégséges egyszer megírni a megfelelő paraméterekkel a leíró folyamatot, és ezáltal akár új ágensek felvétele is történhet, manuális beálltás nélkül.

A szerverek végtelen számú felhasználásából adódik, hogy szükség lehet egyedi ellenőrzések írás is. Az alapértelmezettnek tekinthető hardveres és szoftveres ellenőrzések mellett, előfordulhat olyan komponens vagy érték, amelyek egyedi igényeket szolgálnak. Szükségünk lehet olyan értékek vizsgálatára is, amely a rendszer speciális összetételéből adódóan egyedi ellenőrzést igényelnek. Erre lehetőségünk nyílik egyedi fejlesztésű ellenőrzések írásával.

Ezeket a megoldásokat implementálva, pontosan és jobb rendszerképet alkothatunk. Nagyobb biztonságot társíthatunk rendszerünkhöz, továbbá a karbantarthatóságot és átláthatóságot is növelhetjük.

9 ÖSSZEFOGLALÁS

A szakdolgozatom első felében a témakör körbejárásán és értelmezésén volt a fő hangsúly. Ez két fő részből tevődött össze. Az első az irodalomkutatás. Ebben a szakaszban a fő kérdéskör az OpenStack megértése és feltérképezése mellett, a hasonló eszközök megismerése és bemutatása szerepel. Az OpenStack előnyei és felhasználása mellett működése, felépítése és az ahhoz szükséges alkotóelemek kerültek bemutatásra. Emellett, a további hasonló működésű és felépítésű eszközök közül, kiemelésre és szemléltetésre a Sensu, Zabbix és a Prometheus Grafana páros került. Általános információk mellett bemutatásra került az eszközök felépítése, adatgyűjtő szolgáltatásai. Ennek a résznek a lezárására egy összehasonlító fejezetrész került, amely tartalmazza az előnyeit a korábban említett alkalmazásoknak és egy általános szemléletet ad a felhő monitorozás hátrányairól. Az összehasonlításból levonva a megfelelő következtetéseket, a kiválasztott eszköz a Sensu lett. A kiválasztásnál fontos szerepet játszott a Sensu régóta fennálló magasfokú megbízhatósága, dokumentációjának részletessége. Korábban az ELKH Cloud elődje: az MTA Cloud monitorozása is ezzel a megoldással került megvalósításra a Wigner FK által. Ez lehetőséget biztosít számunkra egy korábbi Sensu implementáció összehasonlítására a legújabb Sensu verzióval. A másik része pedig a tervezés előkészítése és megvalósítása volt. Definiálásra kerültek azok a paraméterek, amelyek alapján a tervezés megkezdhető. A tervezés fázisában pedig definiálásra kerültek azok a hálózati és fizikai paraméterek, amellyel a konkrét implementálás megkezdődhet. Ennek előkészítésére elkészült a megvalósítandó rendszer ábrája, ami szemlélteti az elkészítendő rendszer architektúra felépítését. Az ábra tartalmazza mindazokat a szükséges elemeket, amelyek implementálásra kerülnek.

A dolgozat második felében kezdéseként a korábban megtervezett architektúra implementálása történt. A fizikai rendszer összeállítása és konfigurációja. Definiálásra kerültek az implementációs lépések, amiken keresztül az eszközök és szoftverek beállítása megtörtént. A teljes rendszer megismerését követően, a pontos monitorozás paraméterek definiálása és kidolgozására került sor. Az ellenőrzések felvétele a rendszerbe, valamint ezek konfigurációs fájlainak szerkesztése. Az értesítések kiszervezésére a Slack projektmenedzser szoftver került kiválasztása, amely használatos mind kis vállalkozásoktól kezdve a nagyvállalatokig. API kapcsolaton keresztül megtörtént Sensu backend összekapcsolása a Slack meghatározott csatornával. Ahhoz, hogy ellenőrizni tudjuk az adatok helyességét, az ellenőrzések valódiságát a kiválasztott ellenőrzések tesztelésére került sor. Az ágensekkel való kapcsolat, a hardveres eszközök terheltsége, a Slack csatornával való kommunikáció került a fókuszba. Ezekkel az alapvető tesztelési megoldásokkal biztosítottá vált, a beállítások helyessége, valamint a rendszer integrációjának megfelelő előkészítése.

Végezetül pedig újabb kutatómunka következett. A célja, a rendszer továbbfejlesztési lehetőségeinek felismerése és dokumentálása volt.

10 SUMMARY

In my thesis the focus was on the overview and interpretation of the topic. This consisted of two main parts. The first was literary research. In this section in addition to understanding and mapping the main issue, OpenStack, the main focus is on understanding and presenting similar tools. In addition to the benefits and uses of OpenStack its operation, architecture and the components required for it were presented. Also, among other tools with similar functionality and architecture the Sensu, Zabbix and Prometheus Grafana pairs were highlighted and illustrated. Moreover, to general information the design and data collection features of the devices were presented. This section is concluded by a comparative chapter section, which includes the advantages of the previously mentioned applications and gives a general overview of the disadvantages of cloud monitoring. After drawing the appropriate conclusions from the comparison Sensu was selected as the tool of choice. Sensu's long-standing high reliability and the detail of its documentation played an important role in the selection. Previously the predecessor of ELKH Cloud, MTA Cloud was also monitored with this solution by Wigner FK. This allows us to compare an earlier Sensu implementation with the latest version of Sensu. The other part was the preparation and implementation of the design. Parameters were selected based on which the design can be started. And in the design phase the network and physical parameters were defined to start the concrete implementation. In preparation for this, a diagram of the system to be implemented has been prepared, which illustrates the architecture of the system. The diagram contains all the necessary elements.

In the second part the implementation of the previously designed architecture was carried out. Assembly and configuration of the physical system. The implementation steps were defined by selected tools and software. Once the complete system was understood, the exact monitoring parameters were defined and developed. The controls were added to the system and their configuration files were edited. Slack project management software was selected to outsource the notifications, which are used by small businesses to large enterprises. Through API connection, the Sensu backend was connected to the Slack defined channel. To verify the correctness of the data the validity of the selected checks was tested. The focus was on the connection with the agents, the load on the hardware devices, and the communication with the Slack channel. These basic testing solutions ensured that the settings were correct, and that the system was properly prepared for integration.

Finally, more research was done. The aim was to identify and document the possibilities for further development of the system.

11 ÁBRAJEGYZÉK

12 TÁBLÁZATJEGYZÉK

13 IRODALOMJEGYZÉK

[1] OpenStack, (https://www.openstack.org), utoljára megtekintve: 2022. 05. 10.

[2] Bernardino, T. R,. An overview of openstack architecture: *In Proceedings of the 18th Internati onal Database Engineering & Applications Symposium*, 2014

[3] OpenStack, (https://www.openstack.org/software/), utoljára megtekintve: 2022. 05. 10.

[4] *Sensu*, (https://opensource.com/article/18/8/getting-started-sensu-monitoring-solution), utoljára megtekintve: 2022. 05. 10.

[5] *Open source dashboard for Sensu*, (https://uchiwa.io/#/), utoljára megtekintve: 2022. 05. 10.

[6] https://medium.com/@archanabalasundaram/monitoring-your-infrastructure-with-sensu-go-and-telegram-5415465c7b87, utoljára megtekintve: 2022. 05. 10.

[7]Olups, R., *Zabbix Networking Monitoring,Packt ,* Birmingham Mumbai,2016

[8] *Zabbix Agent*, (https://www.zabbix.com/zabbix_agent), utoljára megtekintve: 2022. 05. 10.

[9] *Zabbix features and architecture*, (https://subscription.packtpub.com/book/cloud_and_networking/9781789340266/1/ch01lvl1sec04/zabbix-features-and-architecture), utoljára megtekintve: 2022. 05. 10.

[10] *Zabbix Dashboard*, (https://www.zabbix.com/documentation/current/it/manual/web_interface/frontend_sections/monitoring/dashboard), utoljára megtekintve: 2022. 05. 10.

[11] Brazil, B., *Prometheus: Up & Running*: United States of America: O'Reilly media, Inc.

,2018

[12] Rahman, D. H., Monitoring Server dengan Prometheus dan Grafana serta Notifikasi Telegram. *urnal Ilmiah Teknologi Sistem Informasi*, 1(4), 133 - 138.

[13] Grafana dashboard, (https://grafana.com/grafana/dashboards/10533), utoljára megtekintve: 2022. 05. 10.

[14] Sensu checks, (https://bonsai.sensu.io), utoljára megtekintve: 2022. 11. 30.

[15] Slack, (https://slack.com), utoljára megtekintve: 2022. 11. 30.

[16] SMART test, (https://www.serveradminz.com/blog/perform-smart-test-memory-test/), utoljára megtekintve: 2022. 11. 30.