ETDK-dolgozat

Füstös Ferenc

Péter Katalin

XXV. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)

Kolozsvár, 2022. május 19–22.

Raspberry Kube

Kubernetes platform felépítése és menedzselése Raspberry Pi eszközökön

Szerzők:

Füstös Ferenc

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Péter Katalin

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Matematika - Informatika szak, III. év

Témavezetők:

dr. Sulyok Csaba, egyetemi adjunktus

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

László Norbert-Péter, szoftverfejlesztő,

Codespring

Mátis Szilárd-Gábor, szoftverfejlesztő,

Codespring

Szabó Zsolt, szoftverfejlesztő,

Codespring

CODESPRING

OUR CODE YOUR SUCCESS

Kivonat

A Raspberry Kube projekt célja egy Kubernetes klaszter felépítése és karbantartása egy Raspberry Pi-okból álló szuperszámítógépen.

A dolgozat lépésről lépésre bemutatja egy skálázható rendszer felépítését Kubernetes orchesztrációs megoldások és Spring mikroszolgáltatások segítségével.

Első lépés a Rancherből felügyelhető klaszter fizikai komponenseinek összeállítása. A négy eszköz mindenike rendelkezik egy saját MicroSD kártyával és Raspberry Pi OS Lite operációs rendszerrel, valamint internethozzáféréssel. Az ezekből felépített Kubernetes klaszter konfigurálása során Ansible playbookok segítségével automatikusan a mester csomóponthoz kapcsolódik a másik három, egy egységes virtuális klasztert hozva létre.

A következőkben egy microservice architektúrájú alkalmazás fejlesztésére és kitelepítésére kerül sor, amivel demonstrálható a fent leírt infrastruktúra használhatósága. Az alkalmazás egy mozi jegyeladásainak kezelését teszi lehetővé. A funkcionalitások külön konténerekbe való leosztása a létrehozott környezet sajátosságait hivatott bemutatni. Az így kitelepített alkalmazás reziliens, látogatottság függvényében könnyen skálázható és több szempontból monitorizálható.

Tartalomjegyzék

Bevezető **1**

1. Kubernetes és Rancher **3**

1.1. A Kubernetes rendszer rövid leírása 3

1.2. Kubernetes alapfogalmak 4

1.3. K3s 6

1.4. Rancher 6

2. A rendszer felépítése **8**

2.1. A virtuális klaszter 10

2.2. Ansible 11

2.3. Jelszavak és biztonság 12

2.4. Az egységek elérése hálózaton keresztül 14

3. A klaszter használata **15**

3.1. Klaszter specifikáció és konfiguráció 15

3.2. Online Notes 15

3.3. Mandelbrot Image Generator 16

3.4. Elsődleges teljesítménytesztelés 17

4. A Movies projekt **18**

4.1. Technológia 18

4.2. Architektúra 18

4.3. Monitorizálás 22

Következtetések és továbbfejlesztési lehetőségek **24**

Bevezető

A Kubernetes elsősorban konténerizált alkalmazások kezeléséhez ad eszközöket [9]. Konténerizált alkalmazásról beszélünk, amennyiben egy alkalmazás minden függőségével és funkcionalitásával egy image-be tömörítődik, amelyet felhasználva futtatható bármilyen előzetes, futtatási környezetet érintő beállítás nélkül. Az alkalmazások kitelepítése során ilyen konténer vagy egymással kommunikáló konténerek kerülnek ki a Kubernetes klaszterre, amely képes felügyelni az alkalmazást futás közben. Egy klaszter kétféle csomópontból áll: egy masterből és egy vagy több workerből. A kitelepített alkalmazás ilyen formában a Kubernetes felügyelete alá kerül, amely folyamatosan nyomon követi annak az állapotát. Amennyiben egy konténer meghibásodik és/vagy leáll képes azt észlelni és újraindítani, így a felhasználók nem érzékelik a meghibásodást [10]. Az alkalmazás felügyelése magába foglalja annak skálázását is: a Kubernetesnek azon képessége, hogy látogatottság függvényében több vagy kevesebb erőforrást biztosít az alkalmazásnak. Ugyanakkor monitorizációs lehetőségek is adottak, azaz figyelemmel követő a csomópontok viselkedése és erőforrás-használata.

A Raspberry Kube projekt a Kubernetes és konténerizáció világát hivatott feltérképezni. Célja egy Raspberry Pi-okon futó Kubernetes klaszter megépítése és bekonfigurálása, hogy azon enterprise szintű alkalmazásokat lehessen futtatni. A klaszter felügyelése Rancherből történik. A dolgozat bemutatja továbbá a Movies alkalmazást, egy Spring mikroszolgáltatásokból álló rendszert. Az alkalmazás egy proof of concept projektként viselkedik, melynek fejlesztése, kitelepítése és monitorizálása során a már létrehozott infrastruktúra sajátosságára esik a hangsúly.

A dolgozat a következőképpen struktúrálódik: az 1. fejezet a Kubernetes és Rancher alapfogalmakba nyújt bevezetést, a 2. fejezet a rendszer fizikai komponenseiről, illetve bekonfigurálásának első lépéseiről szól, a 3. fejezet a Kubernetes klaszter specifikációjáról, konfigurációjáról, elsőként kitelepített alkalmazásokról és monitorizálásokról szól, a 4. fejezet a Movies projektet mutatja be, a záró fejezet pedig a következtetéseket és továbbfejlesztési lehetőségeket tartalmazza.

A Raspberry Kube projekt a 2021-es szakmai gyakorlat keretein belül indult útjára, melynek során főleg a kutatásra, konfigurációra és előzetes tesztelésre esett a hangsúly.

A projekt folyatásaként a Movies alkalmazás fejlesztésére, kitelepítésére és monitorizálására a csoportos projekten került sor. Ez alatt a csapathoz csatlakoztak: Borbáth Alpár-Levente, Kis Csaba és Simon Ádám.

A projekt nem valósulhatott volna meg a Codespring csapata nélkül. Köszönet a segítségért Nagy Ákosnak. Külön köszönet illeti a projekthez kijelölt mentorokat, Mátis Szilárdot, Szabó Zsoltot és László Norbertet, illetve a projekt koordinátorát, dr. Sulyok Csabát.

1. Kubernetes és Rancher

A projektunk elsődleges célja egy Raspberry Pi számítógépekből álló Kubernetes klaszter létrehozása, bekonfigurálása, menedzselése és használata.

1.1. A Kubernetes rendszer rövid leírása

A Kubernetes [1] (rövidítve K8s) egy nyílt forráskódú konténer orchestration rendszer, amely segítségével automatizálható a kitelepítés, skálázás, illetve menedzselés. A "Kubernetes" szó görögül kormányost vagy hajóskapitányt jelent.

A Kubernetes első verziója 2014-ben volt kiadva a Google cég által, viszont később a Cloud Native Computing Foundation [4] szárnyai alá került át.

Napjainkban Kubernetes segítségével különböző cégek képesek webalkalmazásaikat könnyedén elérhetővé tenni, attól függetlenül, hogy saját infrastuktúrával rendelkező vagy saját infrastuktúrával nem rendelkező cégről van szó. Saját infrastuktúrával rendelkező cégről akkor beszélünk, ha dedikált szerverekre kerülnek ki az alkalmazások. Saját infrastruktúrával nem rendelkező cégről akkor van szó, ha az illető cég egy Kubernetes klasztert bérel egy szolgáltatótól, mint például Amazon, Google, Oracle. Mindkét esetben a Kubernetes rendszer egységes megoldást nyújt az alkalmazások kitelepítéséhez.

A Kubernetes-re kitelepítendő alkalmazásokat standard módon kell előkészíteni, hogy minden rendszer ugyanolyan módon képes legyen futtatni őket. Ezt a folyamatot konténerizációnak nevezzük. Konténer alatt olyan szoftvercsomagokat értünk, amelyek magukban hordoznak minden futtatáshoz szükséges információt. Egy alkalmazás ilyen konténerek összességéből épül fel (lásd az 1. ábra [1]).

Eredetileg a Kubernetes csak a Dockert támogatta mint kontenérizációs technológia. Később lehetőség nyílt a konténerek futtasására [11] Containerd és más programok segítségével, amik a Container Runtime Interface-t implementálják.

A Kubernetes, használatára egy HTTP feletti JSON alapú API szervert tesz elérhetővé, amelynek segítségével a fejlesztő erőforrásokat hozhat létre, módosíthat vagy törölhet ki. Ennek az API-nak a használata általában a kubectl parancssori program segítségével történik. Az API jellege miatt használható weboldalakon keresztül is, mint például Kubernetes dashboard [3] vagy Rancher(lásd 1.4. fejezet).

[1] Forrás: https://www.sdxcentral.com/wp-content/uploads/2019/05/ContainersvsVMs_Image-1.jpg, utolsó megtekintés dátuma: 2018-04-10

1.2. Kubernetes alapfogalmak

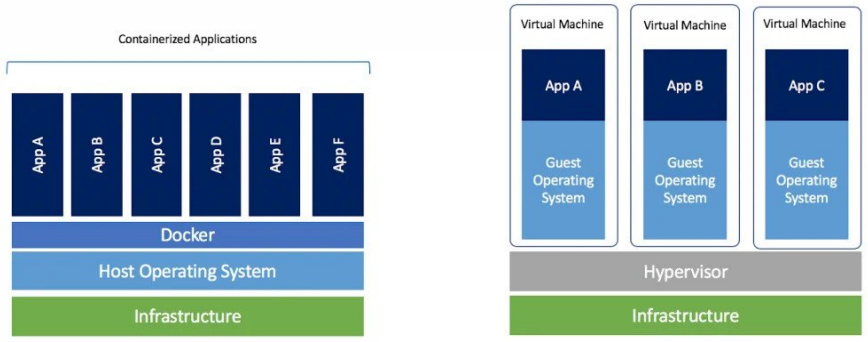
A Kubernetes pár alapvető fogalomból épít fel egy klasztert és annak egy elméleti reprezentációját: szó eshet a klasztert felépítő fizikai csomópontokról, amiket ebben a projektben a Raspberry Pi-ok képviselnek, a podokról, amelyek webszolgáltatások konkrét futó példányai a klaszteren, vagy a servicekről, amelyek segítségével a hasonló podokat csoportosítani és egységesíteni tudjuk.

A Kubernetes klaszter felépítése a 2. ábrán [2] tanulmányozható.

A legalapvetőbb építő eleme a Kubernetes rendszernek a pod, amely egy vagy több futó konténer reprezentációja. Amikor egy klaszter beütemez egy podot, akkor a klaszter valamelyik csomópontjára (általában a legkevésbbé használt csomópontra) helyezi el.

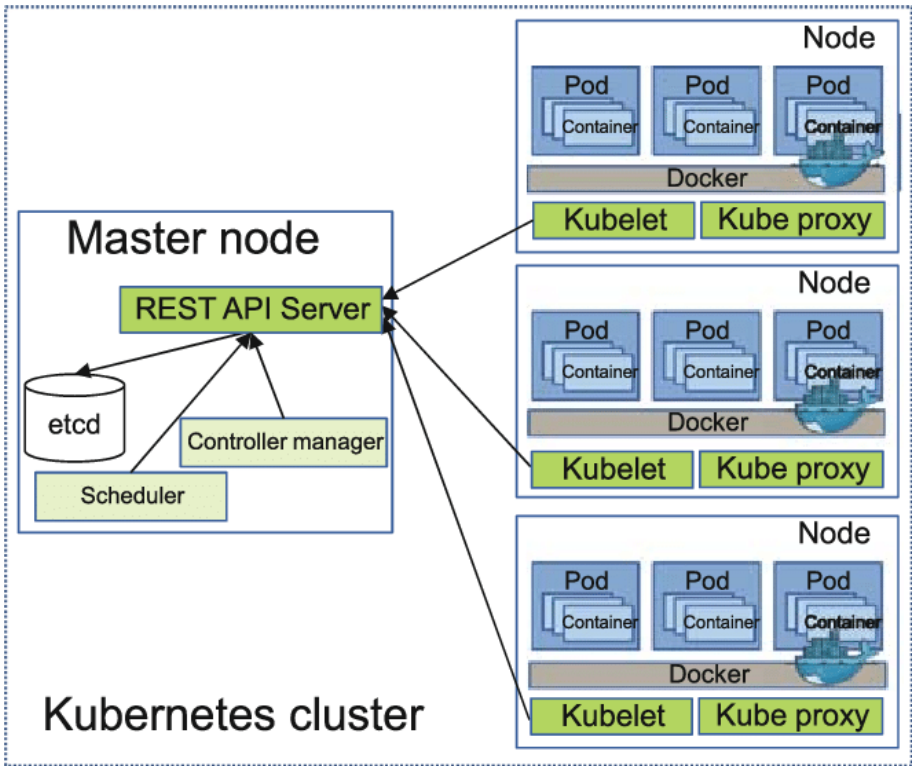
Több pod összesítésére használható a service. A service egy absztrakciós réteg a podok fölött, amely feltételezi a podokban futó alkalmazásokról, hogy nincs belső állapotuk. Ha csak egy podot irányít a service, akkor lehet belső állapota (pl. adatbázis). Egy DNS bejegyzést szúr be minden podba, amelyek ugyanabban a namespace-ben találhatóak. Amikor a service megkap egy kérést, akkor elirányítja valamely általa reprezentált podnak, a megfelelő port számon keresztül, illetve a választ visszaküldi. Service alkalmazható általános alkalmazásokat tartalmazó podok esetén is.

A podok életciklusának felülvizsgálásáért, kezeléséért és karbantartásáért a deploymentek felelősek. Egy deployment erőforrás tartalmazza a podjainak a tervrajzát. Itt állítjuk be, hogy a podok milyen Docker image-et használjanak, a replikák számát (hány darab pod jöjjön létre az adott sablon/tervrajz alapján), illetve a skálázási stratégiát. Ha egy pod meghibásodik (például a benne futó szerver megáll), akkor a deployment újraindítja a podot. Ezen kívül, ha a használt



1. ábra. Konténerekből álló alkalmazások felépítése

[2] Forrás: https://www.researchgate.net/figure/Kubernetes-architecture_fig1_320248964, utolsó megtekintés dátuma: 2018-04-10



2. ábra. A Kubernetes klaszter felépítése

Docker image megváltozik (például kiadnak egy újabb változatot), akkor a deployment felelős a régi podok leállításáért, illetve az új podok létrehozásáért.

Az ingress és ingress kontroller egy másik fontos absztrakciós réteget képez a service réteg fölött. A klaszterben összesen csak egy ingress kontroller található, ez felelős az ingressek feldolgozásáért. Ennek az erőforrásnak köszönhetően képes a klaszter az URL alapján elvégezni a szükséges reverse proxy lépéseket. Az ingress kontroller csak webalkalmazások esetén használható fel kérések megfelelő szabályzásához. A kérés, az ingress kontrollerrel történő besorolását követően egy specifikált service-hez továbbítodik. Innen a service tovább irányítja őt egy podnak.

A Kubernetes környezetben az adatok perziztenciájára a persistent volume (PV) illetve persistent volume claim (PVC) nyújt megoldást. Egy általános fejlesztő persistent volume claimeket hozhat létre, abba a namespace-be amelyben az alkalmazás található, amely használni szeretné ezt. A persistent volume claim lefoglal egy olyan persistent volume-ot, ami szabadon van és teljesíti a feltételeit. Például egy 10 GB tárhelyű PV teljesíti egy 8 GB tárhelyű PVC feltételeit, ha ugyanolyan típusú.

A Kubernetes erőforrások csoportosítására használt fogalom a namespace, melynek segítségével egységként tudjuk az erőforrásokat kezelni. Minden pod, service, ingress és

persistent volume claim csak namespace-ben hozható létre. Nem létezhetnek namespace-n kivül. A podok csak a namespace-ükben megtalálható service-ekről tudnak, ezért ha két pod közt szükséges a kommunikáció, akkor a podok szükséges, hogy ugyanabban a namespace-ben legyenek.

1.3. K3s

A K3s egy pehelykönnyű, iparban is használható Kubernetes disztribució. Pár előnye a K3s-nek:

- könnyű telepítési folyamat
- kicsi program méret (ez kevesebb mint 50 MB)
- több operációs rendszer és processzor architektúra van támogatva (például amd64, arm64, armv7, stb.)

A könnyű telepítés és a széleskörű platform támogatás miatt pár perc alatt könnyen felhúzható egy tesztelésre használható klaszter a fejlesztő számítógépén.

A K3s egy előre bekonfigurált Kubernetes rendszert hozz létre a telepítés folyamán. Ez a folyamat tartalmazza a Traefik program installálását, mint a klaszter ingress kontrollere.

1.4. Rancher

A Rancher egy egységes multi-klaszter kezelési rendszer, amely megkönnyíti több Kubernetes klaszter karbantartását és kezelését, illetve biztonsági és jogosultsági megoldásokat nyújt.

A rendszer egyik fontos szolgáltatása az a webes irányító felület, melynek segítségével a Kubernetes-es rendszergazdák Kubernetes klasztereket tudnak beregisztrálni, és ezek után konfigurálni. Egy klaszter beregisztrálása a Rancher weboldalán a megfelelő információk megadása után egy Kubernetes-es erőforrás állomány kitelepítésével valósítható meg. Miután egy klaszter regisztrálva lett a Rancherbe, az általa kigenerált klaszter id alapján és a megfelelő HTTP endpointján keresztül elérhetővé válik. Ez az megközelítésen teljesen tükrözi a Kubernetes HTTP endpointját, emiatt endpoint váltással különböző klasztereket lehet ugyanolyan módon kezelni. Ez főleg parancssori eszközöknél hasznos.

A Rancher egy másik funkcionalitása a felhasználó kezelés és a külső felhasználók integrációja. A Rancher lehetőséget nyújt arra, hogy egy külső felhasználó rendszerrel legyen összekötve, és ebben a külső rendszerben létező csoportok és felhasználók jogosultságait kezeljük. Különböző szintű jogosultságokat lehet megadni a felhasználóknak és csoportoknak, beleértve azt is, hogy csak bizonyos klaszterekre adunk bizonyos jogosultságot. Például beállíthatjuk, hogy a "valami-raspberry" nevű GitHub organizáció tagjai be tudjanak lépni a Rancher rendszerébe, illetve a "test1" klaszterhez legyen hozzáférésük. A Ranchernél több külső felhasználó szolgáltató használható, mint például a Google OAuth, a GitHub, Microsoft Active Directory, OpenLDAP és még mások, viszont csak egy ilyen szolgáltatóval lehet összekötve a Rancher.

2. A rendszer felépítése

Ebben a fejezetben a klaszter (a fizikai, négy Raspberry Pi-ból álló rendszer, később a virtuális csomópontok összessége) felépítésének és konfigurálásának első lépéseiről lesz szó. A klaszter egységként való működéséről bővebb leírás a 3.1. fejezetben található.

Fizikai komponensek

A bemutatott számítógép egy négy Raspberry Pi-ból álló rendszer. A Pi-ok mindenike rendelkezik egy type-C töltőkábellel, illetve egy 64 GB-os MicroSD kártyával (a Raspberry Pi-ok hardverje limitált [12]). A négy Pi-t és a hűtőberendezésüket egy keret tartja össze.

Első lépésben a MicroSD kártyák kerülnek rá a Pi-okra csatlakozók segítségével, amelyek arra szolgálnak, hogy majd a későbbiekben a rács szétszerelése nélkül is szükség szerint hozzájuk lehessen férni. A Pi-ok ezek után külön-külön egy átlátszó lapra rögzítődnek, a 3. ábrán látható módon.

Következő lépés a négy Pi és a hűtőberendezés keretre való rögzítése. A hűtőberendezés a legfelső (később master) Pi-ra csatlakozik, a 4. ábrán látható módon.

Az összeszerelés utolsó lépése a töltőkábelek csatlakoztatása (5. ábra). Ekkor a klaszter készen áll a további beállításokra.

Operációs rendszer

A klaszter egyik legfontosabb jellemzője az operációs rendszer. Esetünkben a Raspberry Pi eszközökön a hivatalos Raspberry PI OS Lite 32 bites operációs rendszer fut [13]. Ez egy asztali számítógép nélküli nyílt forráskódú operációs rendszer, ami a Debian Bullseye Raspbian elnevezésű, Raspberry Pi rendszerekre optimalizált kiadásának egy verziója.

Az operációs rendszer Raspberry Pi Imager [15] felhasználásával telepíthető a Pi-okra.

Internethozzáférés

Minden számítógépnek, amely részt vesz a klaszter felépítésében, szüksége van internet hozzáférésre. Ez egy fontos előfeltétel minden gépnek, mivel a klaszter működése folyamán szükséges lesz a Docker image-ek letöltésére, mivel így valósítható meg az alkalmazások kitelepítése a klaszterre. Ezen kivül a gépek közötti kommunikáció is hálózaton keresztül történik.

3. ábra. Az átlátszó lapra rögzített és csatlakozóval ellátott Raspberry Pi az SD kártyával.

A Raspberry Pi-ok nincsenek közvetlen módon csatlakoztatva a publikus internet hálózatra, mivel ez több támadási célpontot nyitna meg. Ehelyett a Pi-ok egy hálózati switch-be vannak bekötve, ahonnan csak az egyik Raspberry Pi kap publikus IP címet. A switch tűzfala csak bizonyos portokat enged át befele menő csomagok részére (HTTP, HTTPS, SSH, Kubernetes API). A master csomópont megkapja ezt a publikus címet és a többi csomópont rajta keresztül fér hozzá az internet hálózathoz. Mivel minden számítógépnek csak egy fizikai Ethernet portja van, ezért a belső hálózatot egy virtuális hálózatként hozzuk létre.

A master csomópontot futtató Raspberry Pi létrehoz egy virtuális hálózatot a 192.168.1.0/24 submask-al, és minden Pi statikus konfigurációval beállítja az IP címét:

```
master 192.168.1.200
worker1 192.168.1.201
worker2 192.168.1.202
worker3 192.168.1.203
```

Ez a konfiguráció elérhetővé teszi a master csomóponton keresztül az összes többi csomópontot.

4. ábra. A kész négy csomópontos, hűtőberendezéssel ellátott Raspberry Pi klaszter

2.1. A virtuális klaszter

A virtuális klaszter felépítésének célja a master csomóponthoz csatlakoztatni a másik három csomópontot. Ez tokenek felhasználásával végezhető el. Először magát a Kubernetest (bővebben az 1.1. és az 1.2. fejezetekben) kell telepíteni a master csomópontra, és az így létrejött k3s könyvtárból lementeni a node-tokent. Utána a további három eszközre egyenként rácsatlakozva az így lementett token megadható a Kubernetes telepítésekor a következő kódrészletben látható módon:

```
curl -sfL https://get.k3s.io | K3S_TOKEN="{{ token.stdout }}"
                K3S_URL="https://master.raspberry:6443"
                K3S_NODE_NAME="raspberrypi-{{ inventory_hostname }}"
                sh -
```

A parancs futtatása után a csomópont már automatikusan a masterre fog csatlakozni. Ez ellenőrizhető a master csomóponton, a kubectl parancssori program használatával.

5. ábra. A kész négy csomópontos, hűtőberendezéssel ellátott Raspberry Pi klaszter

2.2. Ansible

Látható hogy a virtuális klaszter manuális felhúzása egy monoton folyamat, melynek során egyenként kell csatlakozni a Pi-okhoz és parancsokat futtatni rajtuk. Ez a folyamat egyszerűsíthető Ansible segítségével.

Az Ansible [16] egy multi-tier kitelepítésre fejlesztett automatizációs eszköz. A rendszerek közötti kapcsolatokat kihasználva lehetővé teszi, hogy egy helyről végezzünk el több különálló rendszerre vonatkozó beállítást, YAML fájlokban megadva azokat. Az Ansible tehát segítséget nyújt konfigurációk megadásánál, alkalmazások kitelepítésénél, és cloud provisioning esetén is. A Raspberry Pi-okra való telepítése a következőképpen történik:

```
sudo apt install ansible
```

A virtuális klaszter felhúzásának lépései Ansible playbookok segítségével adhatóak meg. Egy-egy playbook futtatható egy vagy több hoston is.

A Raspberry Pi eszközökre való csatlakozáshoz szükséges adatok egy hosts fájlban adhatóak meg. A Secure Shellel történő csatlakozáshoz minden csomópont IP címe, illetve felhasználóneve és jelszava szükséges (a jelszó tárolásának biztonságosabbá tételét a 2.3. fejezetben tárgyaljuk). Minden host és host csoport kap egy nevet is, amivel majd hivatkozni lehet rájuk a playbookokban. Továbbá szükséges beállítani a a kulcsok szigorú ellenőrzésének elhanyagolását:

```
ansible_ssh_extra_args='-o StrictHostKeyChecking=no'
```

Alapértelemezés szerint az SSH minden ismert host-ra csak egy kulcsot tárol el az adatbázisában, ami problémás abban az esetben, ha a kulcs változik. A szigorú ellenőrzés kikapcsolása révén az SSH már képes lesz a már ismert hostokhoz újabb kulcsokat hozzárendelni.

A hosts fájl elkészítése után következhetnek maguk a playbookok. A virtuális klaszter felhúzásához szükséges lépéseket négy részre (azaz négy playbookra) oszthatjuk. Első lépésben olyan felhasználókat kell létrehozni minden Pi-on, amely jogosult sudo parancsokat futtatni jelszó beírása nélkül. Az első playbook tehát a kube-nek nevezett felhasználókat hozza létre mind a négy eszközön. A második playbook az IP címek inicializálását hajtja végre, szintén minden nóduson. Következik a Kubernetes master csomópontra való telepítése, illetve a token lementése. Ezzel a harmadik playbook foglalkozik. Utolsó lépés a Kubernetes telepítése a worker csomópontokra a token felhasználásával.

A klaszter felhúzásának lépései tovább egyszerűsíthetőek, ha a playbookok futattása egy Makefile segítségével történik. Így nem csak a futtatási sorrend helyessége biztosított, de a későbbiekben a jelszókezeléssel kapcsolatos plusz lépések is könnyebben beágyazhatóak.

2.3. Jelszavak és biztonság

A 2.2. alfejezetben tárgyalt módszer a virtuális Kubernetes klaszter felhúzásának egy egyszerűsített, biztonsági óvintézkedésekkel el nem látott módja. Annak ellenére, hogy ez egy működőképes eljárás, a jelszavak hardcode-olásával hagyott biztonsági kiskapukat ajánlott elkerülni.

Erre a célra használható az Ansible beépített enkriptáló funkcionalitása, az Ansible Vault. Segítségével a playbook-ok futtatásához szükséges jelszavakat elég egy fájlban összegyűjteni

és enkriptálni. Az Ansible Vault képes bármilyen strukturált fájl enkriptálására amit az Ansible fel tud dolgozni. Enkriptálhatóak tehát változók, taskok és egész playbookok is.

Esetünkben az SSH csatlakozáshoz szükséges jelszavak enkriptálására van szükség. Ezek a nodes-pass.yml fájlban vannak összegyűjtve, amelyhez (szintén biztonsági okokból) csak a csapat tagjai férhetnek hozzá. Erre az enkriptált fájlra van tehát szükség a virtuális Kubernetes klaszter felhúzásához.

A nodes-pass.yml enkriptálásához szintén jelszó megadása szükséges. Ennek felhasználásával lehet majd megtekinteni:

```
ansible-vault view nodes-pass.yml
```

vagy dekriptálni:

```
ansible-vault decrypt nodes-pass.yml
```

az Ansible Vault segítségével. Ebből adódóan a fájl tartalmának felhasználásához is szükséges az enkriptáláskor megadott jelszó.

A playbookokban hardcode-olt jelszavak kicserélhetőek az enkriptált nodes-pass fájlban megadott nevükre. Azonban így a 2.2. alfejezetben leírt Makefile a módosításra szorul, különben minden playbook meghívásakor ismételten szükséges lesz megadni a Vault fájl enkriptálásához megadott jelszót. Ennek a legegyszerűbb módja egy új fájlban megadni a jelszót (esetünkben ennek password lesz a neve) és azt használni futtatás alatt a következőképpen:

```
ansible-playbook -i hosts kube/kube-init.yml --vault-password-file=password
```

A jelszó megadása egy nem enkriptált fájlban ismét biztonsági kockázatot jelent. Emiatt minden futtatás után le kell törölni a password fájlt, illetve minden futtatás előtt létrehozni azt. Ezek még mindig fölöseges lépések, amik kikerülhetőek a Makefile ismételt módosításával. Elégséges egyszer, a futtatás kezdetén elkérni a jelszót, majd azt az akkor létrehozott password fájlba menteni. A playbookok futtatása után szintén a Makefile gondoskodik a létrehozott password fájl törléséről.

A jelszavakat tartalmazó enkriptált fájl és az enkriptálásnál megadott jelszó birtoklása tehát a Kubernetes klaszter felhúzásának elégséges előfeltétele. A nodes-pass.yml Makefile mellé másolása után egyből futtatható a make parancs, a Makefile majd futás közben elkéri és megfelelő módon felhasználja az Ansible Vault jelszót.

A klaszter törlése is megoldható a fentiekhez hasonló módon. Ehhez újabb két playbook (Kubernetes uninstall külön a master és worker csomópontokon) szükséges. A Makefile-ba

ekkor bekerül egy clean szekció, ami alatt a klaszter felhúzásához hasonló módon lekezelhető az Ansible Vault jelszó a két playbook futtatásakor.

2.4. Az egységek elérése hálózaton keresztül

A Kubernetes klaszter csomópontjaira való csatlakozás legkézenfekvőbb módja az SSH-val való kapcsolódás. Az SSH-val csak karbantartás vagy rendszerkonfiguráció esetén szükséges a csatlakozás. Ennek legelső lépése a master csomópontra való csatlakozás secure shell-el, mivel csak ennek a csomópontnak van publikus IP címe. Ehhez a lépéshez szükséges, hogy a csatlakozni kivánó félnek legyen felhasználója létrehozva a masteren belül, és fel legyen töltve a publikus kulcsa a felhasználójához. SSH-án keresztül csak a privát kulcs birtokában lehetséges a klasztert felépítő gépekhez csatlakozni, ezzel garantálva van a biztonság. Miután rácsatlakoztunk erre a Pi-ra, hozzáférésünk van a klaszter belső hálózatához, és képesek vagyunk csatlakozni a többi csomóponthoz is SSH segítségével. A belső SSH kapcsolatok létrehozásához csak szöveges jelszavakat kell megadni, nem szükséges egy privát kulcsot birtokolni. Erre a csatlakozási módszerre a könnyű elérhetőség miatt esett a választás. A biztonsági szempontokat figyelembe véve, ez a módszer nem a legoptimálisabb, de tűrhető, mert azt feltételezi a felhasználóról, hogy a privát kulcs birtokában áll.

Létezik egy másik módszer is, amely a Ranchert (bővebben az 1.4. fejezetben) használja, hogy lokális context-ként kezelje a Raspberry Pi-okon futó Kubernetes klasztert. Ezáltal saját gépről lehet kubectl parancsokat futtatni a Kubernetes klaszteren.

Első lépésben a Rancherből letölthető Kubekonfig fájl kerül a .kube mappába (ez operációs rendszertől függetlenül a saját könyvtárba kerül a Kubernetes feltelepítése során). Ahhoz hogy a certificate-authority-data saját gép által felismerhető legyen, a Rancher certificate letöltése is szükséges. Második lépésben tehát a böngészőben futó Rancher certificate-jét kell Base-64 kódolt változatban letölteni és installálni. Utolsó lépésként a Kubekonfig certificate-authority-data mezőjét kell lecserélni certificate-authority-re és értékként beilleszteni az installált Rancher certificate elérhetési útvonalát. Ahhoz hogy a Kubernetes klaszteren kubectl parancsokat lehessen futtatni, ki kell választani kontextusként a klasztert:

```
kubectl config use-context kubernetes-cluster
```

3. A klaszter használata

A 2. fejezetben leírt módon felépített és bekonfigurált Kubernetes klaszter teljesítménye első körben kisebb alkalmazások kitelepítésével tanulmányozható. Ebben a fejezetben két ilyen alkalmazásról lesz szó: az Online Notes és a Mandelbrot Image Generator alkalmazásról, illetve a klaszter specifikációjáról és konfigurációjáról.

3.1. Klaszter specifikáció és konfiguráció

A projekt folyamán elkészített Kubernetes klaszter 4 Raspberry Pi 4 Model B-ből van összeállítva. Az összes felhasznált Rasberry Pi a 8 GB RAM-os változat, emiatt a klaszternek összesen 32 GB RAM és 16 darab 1.5 GHz-es processzor mag áll rendelkezésre. Ezeknek az erőforrásoknak egy részét az operációs rendszerek és más eszközök felhasználják, tehát a Kubernetes klaszter használható RAM mennyisége 30.5 GB. Egy Raspberry Pi maximum fogyasztása kevesebb mint 15 Watt, így a klaszter maximális fogyasztása legtöbb 60 Watt lehet.

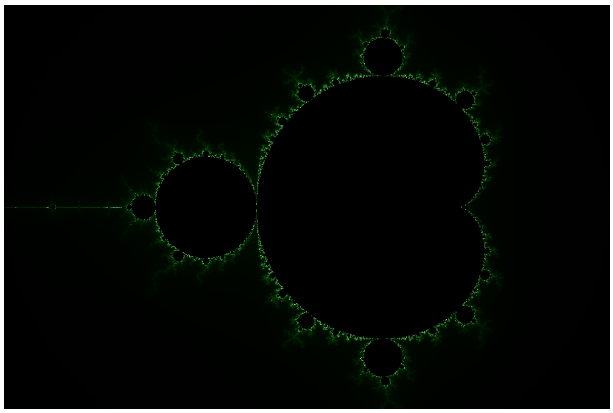
A Kubernetes klaszter kezelésének segítségére használt Rancher alkalmazás is a fizikai klaszterre van kitelepítve. Ez a master node-on található meg, ahova Docker segítségével van kitelepítve, illetve egy reverse proxy mögé van téve, hogy a vele történő kommunikáció HTTPS használatával történjen.

A könnyebb felhasználó kezelés érdekében a Rancheren be van kapcsolva a GitHubos külső felhasználók használata. A GitHubon létrehozott "raspberry-kube" organizáció tagjai képesek bejelentkezni a Rancherre, illetve ezen az organizáció belül a "raspberry-kube-admin" csoport tagjai rendelkeznek klaszter admin jogosultsággal (csak ők tudnak klaszter szintű erőforrásokat kezelni, mint például a persistent volume).

Az elérést a DuckDNS ingyenes dinamikus DNS szervíz [8] biztosítja, a klaszter publikus IP címére mutató "raspberryk3s.duckdns.org" DNS bejegyzés segítségével van biztosítva az elérhetőség. A DNS bejegyzés természete miatt a "raspberryk3s.duckdns.org" összes lehetséges subdomain-je is ugyanerre az IP címre mutatt, emiatt az összes kitelepített alkalmazás elérhetősége garantálva van ez az egy DNS bejegyzés által.

3.2. Online Notes

Az Online Notes egy Node.js-ben fejlesztett, MySQL adatbázissal rendelkező alkalmazás, amely egy webes felületet biztosít egy egyetem számára, hogy rendszerezze a tantárgyakat,



6. ábra. A Mandelbrot halmaz vizuális megjelenítése

illetve az egyes diákok által szerzett minősítéseket adott tantárgyakból. A felhasználók tehát az egyetem diákjai, akik regisztráció után megtekinthetik milyen tantárgyakhoz adták hozzá őket, illetve hogy a felvett tantárgyakból milyen jegyeik vannak.

A Kubernetes klaszterre való kitelepítés lépései YAML fájlokban adhatóak meg és kubectl parancssori eszközzel hajthatóak végre. A kitelepítés az onlinenotes projekt online-notes-dev namespace-e alá történik. Fontos továbbá, hogy az alkalmazásból készült Docker image támogassa az arm64-es architektúrát ahhoz, hogy a Kubernetes klaszteren futhasson. Ez a buildx eszköz segítségével érhető el [19]. A több architektúrát támogató image létrehozására azért van szükség, mert a Kubernetes klaszterre arm64 architektúrájú alkalmazások kitelepítését támogatja (ellenben az amd64 architektúrát támogató PC-kel). Szükséges létrehozni Persistent Volume-ot, adatbázist, illetve az Ingresst.

Az Online Notes webalkalmazás elérhető a `http://onlinenotes.raspberryk3s.duckdns.org/` címen.

3.3. Mandelbrot Image Generator

A Mandelbrot Image Generator az Online Notes projekthez hasonlóan egy Node.js-ben fejlesztett, belső állapot nélküli webes alkalmazás, melynek célja, hogy a Mandelbrot halmaz elemeit kiszámolja a klaszteren, és az eredményeket megjelenítse a böngészőben.

A Mandelbrot halmaz egy matematikai fogalom, meghatározás szerint azon komplex számok halmaza amelyek esetén a $f_c(z) = z^2 + c$ függvény iterálása nem divergál (vagyis nem megy el a végtelenbe), ahol c a komplex szám és z kezdeti értéke 0. A Mandelbrot halmaz egy vizuális reprezentációja megtalálható a 6. ábrán.

A webes alkalmazás tartalmaz egy böngészőben futó felhasználói felületet. Ezen a

felületen pár bemeneti mező van elhelyezve a paraméterek meghatározásának megkönnyítésére, illetve ide lesz kirajzolva a Mandelbrot halmaz. A paraméterek között megadhatjuk a kívánt matematikai tér méretét és azt, hogy milyen felbontásban szeretnénk a halmaz elemeit megkapni. Ezenkívül a legfontosabb paraméter az, hogy hány részbe ossza fel a matematikai területet a rendszer és ezzel hány kérést küldjön a klaszterre. Mivel ez a feladat triviálisan felbontható és párhuzamosítható, ezért tökéletes a klaszter képességeinek bemutatására.

Az alkalmazás frontend-je a klaszteren futó backend-től megkapja minden komplex szám pontjáról, hogy mennyire része a Mandelbrot halmaznak nullától egyig terjedő skálán, és ennek függvényében megjeleníti a halmaz vizuális alakját.

Az Online Notes-hoz hasonlóan (3.2. fejezet) a Mandelbrot Image Generator elérhető a `http://mandelbrot.raspberryk3s.duckdns.org/` címen.

3.4. Elsődleges teljesítménytesztelés

Mindkét alkalmazás esetében a monitorizáció a Rancher beépített Grafana Dashboardjának [18] segítségével történik. Ezen belül lehetőség nyílik nyomon követni a Pi-ok leterheltségét (lásd 7. ábra), illetve a podok fel- és leskálázódását.

7. ábra. A Rancher beépített Grafana Dashboardja

4. A Movies projekt

A Movies projekt célja az eddigiekben leírt környezet sajátosságainak bemutatása. Az infrastuktúra használhatóságát bemutatandó az alkalmazás egy mozi jegyeladásainak kezelhetőségét teszi lehetővé, a funkcionalitások különböző mikroszolgáltatásokba rendezésével. A hat mikroszolgáltatás együttesen képes kezelni a felhasználó adatait, a jegyfoglalást adott filmre, adott időpontban és teremben, illetve a fizetést is.

4.1. Technológia

A felhasznált technológiák kiválasztásának legfontosabb szempontja hogy felhasználásukkal egyszerűen lehessen microservice architektúrájú alkalmazást fejleszteni. A választás a Java programozási nyelvre esett, Gradle build eszközzel párosítva. A Java Spring Boot automatikus konfigurációinak kihasználásával gyorsabban és egyszerűbben végezhető el a web alkalmazások fejlesztése [2]. A Spring Boot applikációkra jellemző módon a Movies projektet alkotó mikroszolgáltatások is Controller-Service-Repository mintát [5] követnek.

Mivel az elmentendő adatok közötti kapcsolatokra nem helyeződik hangsúly, MongoDB nem-relációs adatbázis felhasználásával történik. A kész mikroszolgáltatások egy-egy Docker image-be tömörítődnek.

A klaszterre való kitelepítés GitLab CI/CD [6] segítségével történik. Ezáltal minden szolgáltatás esetében az esetleges módosítások azonnal automatikusan kitelepítődnek. Első lépésben a gradle build eszköz [7] segítségével a lefejlesztett web mikroszolgáltatásból létrejön a Java Archive (továbbiakban JAR) fájl. Második lépésben a már elkészült JAR fájlból elkészül a multi-architecture Docker image, a buildx eszköz segítségével. A harmadik lépés maga a klaszterre való kitelepítés. Ez történhet a 3.2. fejezetben leírt módon. A már kitelepített alkalmazás módosítása esetén a GitLab CI fájl gondoskodik arról, hogy egy rollout segítségével integrálódjanak a módosítások.

4.2. Architektúra

A projekt microservice architektúrájú [17], azaz egymástól független komponensekre bomlik, amelyek kommunikálnak egymással. Az ilyen fajta szerveződésnek több előnye is van. Előnye, hogy így külön-külön is módosíthatóak az egyes mikroszolgáltatások, anélkül hogy a többiben változtatások eszközölésére lenne szükség. Így például egy esetleges

meghibásodás esetén csak az adott szolgáltatás fog sérülni, az összes többi zavartalanul folytathatja a működését. Újabb pozitívum a skálázhatóság: mivel minden mikroszolgáltatás egy saját konténerben fut, egyszerűen lehet az alkalmazás igénybevételéhez igazítani minden funkcionalitás erőforrás-igényét.

A rendszer a következőkben egyenként körülírt mikroszolgáltatásokat vonultatja fel. A kommunikáció REST API-n keresztül történik. Az architektúra részletesebben a 8. ábrán tanulmányozható.

Felhasználó mikroszolgáltatás

A felhasználóról tárolt adatok a következők: családnév, keresztnév, e-mail cím, hitelkártya-azonosító kód, jelszó (enkriptálva) és a felhasználó létrehozásának időpontja. Ezen kívül minden felhasználó kap egy egyedi azonosítót is. A felhasználók listázhatóak, módosíthatóak, törölhetőek, illetve azonosító alapján kereshetőek.

A mikroszolgáltatás kapcsolatban áll a foglalás mikroszolgáltatással.

Vetitőszoba mikroszolgáltatás

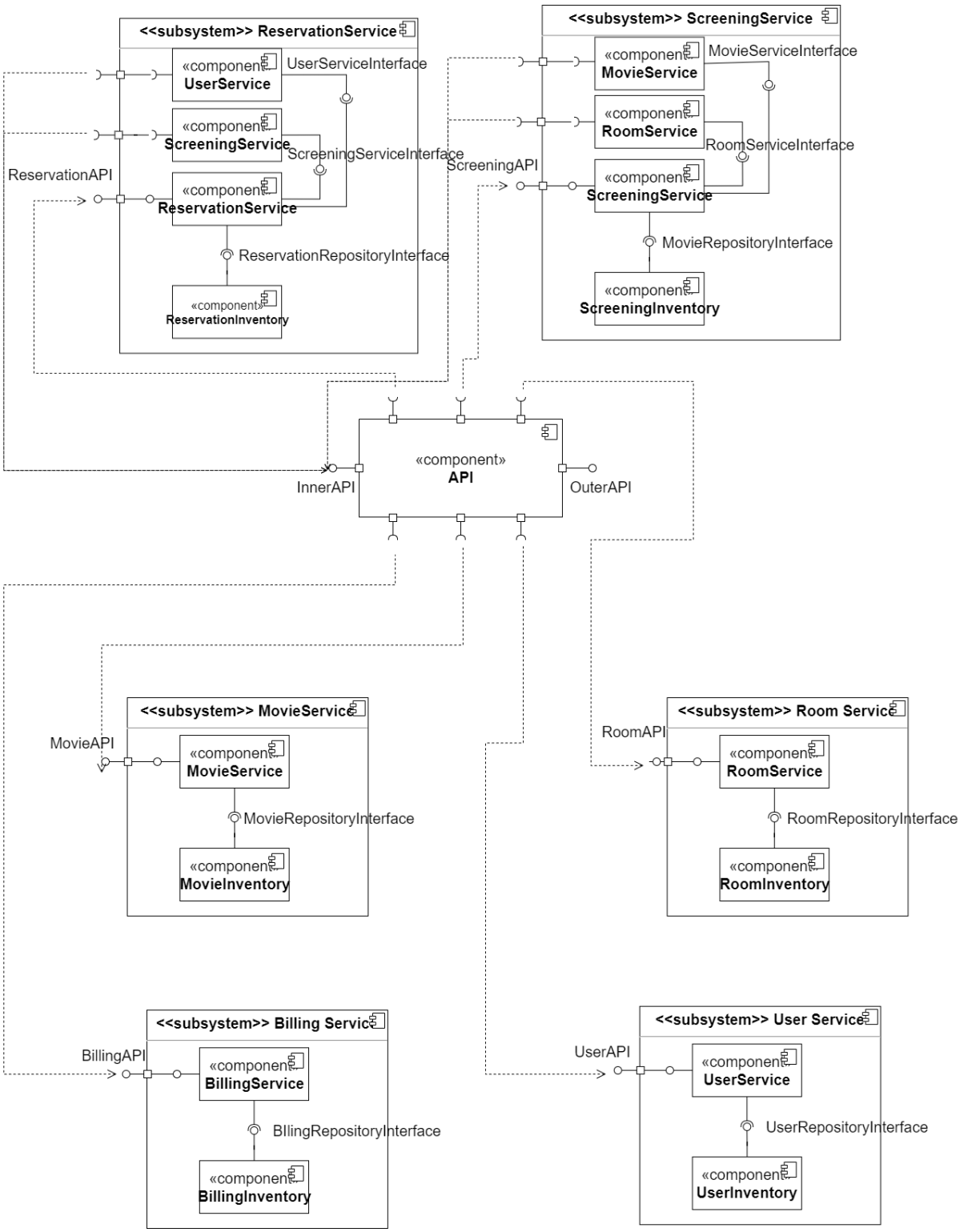
A vetítőszoba mikroszolgáltatás felel a mozi termeiért. Minden terem adatbázisban tárolt adatai tartalmazzák a vetítőszoba nevét (ez minden szoba esetében különbözik), a befogadóképességét és egy egyedi azonosítót. A termek között a nevük alapján lehet keresni, termet módosítani vagy törölni, de lehetőség van új terem létrehozására, illetve az összes létező vetítőszoba kilistázására is.

A mikroszolgáltatás kapcsolatban áll a vetítés mikroszolgáltatással.

Film mikroszolgáltatás

A film mikroszolgáltatás szerepe, hogy a megtekinthető filmekről tároljon adatokat és végezzen műveleteket velük. Minden film esetében elmenti az adatbázisba a film címét, műfaját, hosszát, pontozását, a benne játszó színészeket, a jegy árát, illetve egy egyedi azonosítót. Filmeket lehet létrehozni, módosítani, törölni, listázni és az azonosító alapján keresni is.

A mikroszolgáltatás kapcsolatban áll a vetítés mikroszolgáltatással.



8. ábra. A movie projekt architektúrája. A hat mikroszolgáltatás API-n keresztül kommunikál egymással. A foglalás mikroszolgáltatás ezen kívül kapcsolatban áll a vetítés és felhasználó mikroszolgáltatással, illetve a vetítés mikroszolgáltalás a film és vetítőszoba mikroszolgáltatással.

Vetítés mikroszolgáltatás

A vetítés mikroszolgáltatás egyike a két komplexebb mikroszolgáltatásnak. Az adattagjai közt az egyedi azonosítón és a dátumon kívül szerepel egy vetítőszoba ID és egy film ID is. A vetítőszoba ID egy vetítőszoba mikroszolgáltatásbeli vetítést határoz meg, míg a film ID egy film mikroszolgáltatásbeli filmet. Emiatt tehát a vetítés mikroszolgáltatás képes kell legyen adatok lekérésére a másik két említett mikroszolgáltatástól. A lekérdezés Repository szinten történik, a másik két szervíz eléréséhez az application.properties fájlban megadott elérhetőségek alapján, illetve a lekéréshez szükséges URL-eknek egy Config osztályban való megadásával. A vetítésekkel végezhető műveletek a következőek: egyedi azonosító alapján való keresés, új vetítés beszúrása vagy már létező vetítés törlése, illetve az összes vetítés listázása.

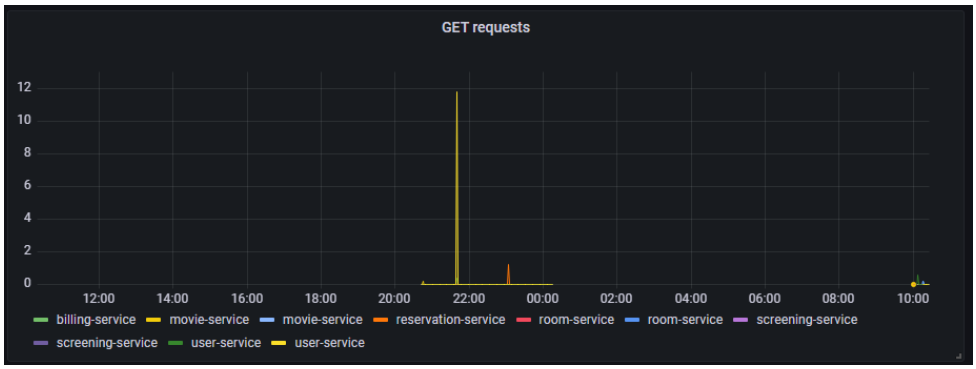
A mikroszolgáltatás kapcsolatban áll a foglalás mikroszolgáltatással.

Foglalás mikroszolgáltatás

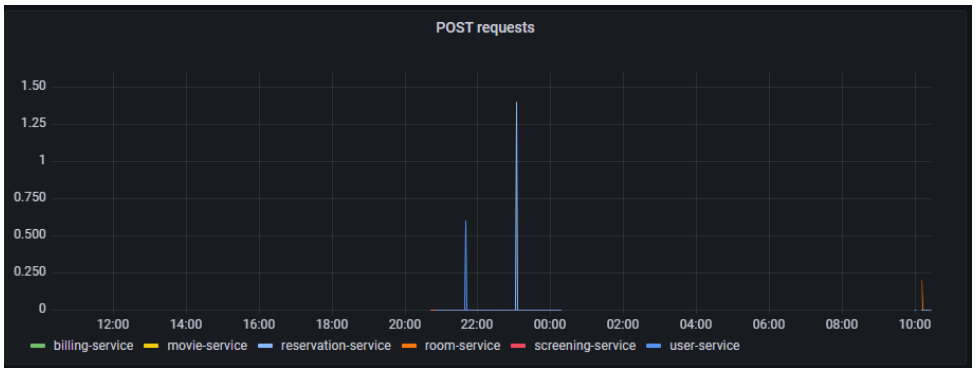
A Movies projekt célja magának a foglalásnak a kezelése, éppen ezért magától értetődő, hogy a foglalás mikroszolgáltatás akár direkt, akár indirekt módon, de felhasználja az összes többi mikroszolgáltatást. Ahhoz hogy egy foglalást meg lehessen határozni, négy adatra van szükség: a felhasználó azonosítójára, aki a foglalást szeretné végrehajtani, a vetítés azonosítójára, amit meg szeretne tekinteni, az ülőhelyekre amiket le akar foglalni és természetesen egy egyedi azonosítóra. A felhasználó adatai a felhasználó mikroszolgáltatás felhasználásával kérhetőek le az ID alapján, míg a vetítéshez tartozó adatokat a 4.2. alfejezetben leírt módon lehet lekérni az érintett mikroszolgáltatásoktól. A foglalás mikroszolgáltatás hasonló módon kezeli a vele kapcsolatban álló mikroszolgáltatásokkal való kommunikációt. Ezáltal lehetőség nyílik új foglalás létrehozására, létező foglalás módosítására vagy törlésére, foglalás egyedi azonosító, felhasználó egyedi azonosítójának vagy vetítés egyedi azonosítójának alapján való keresésére és az összes foglalás kilistázására is.

Számlázás mikroszolgáltatás

A számlázás mikroszolgáltatás egy valódi fizetéseket kezelő szolgáltatást hivatott helyettesíteni. Szerepe a megadott hitelkártya-adatokról (tulajdonos család- és keresztneve, IBAN kód, lejárati dátum, CVV és bank) eldönteni, hogy azok egy valóban létező fizetőeszközhöz tartoznak-e, illetve a fizetésekkel végezhető műveletek kezelése (listázás, keresés és törlés). Minden számla entitáshoz tartozik egy felhasználó, hitelkártya-adatok,



9. ábra. Grafana Dashboard GET kérések monitorizálására



10. ábra. Grafana Dashboard POST kérések monitorizálására

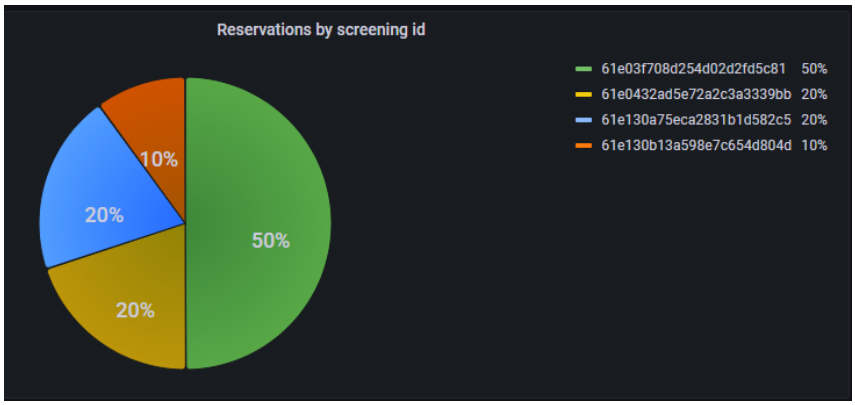
kifizetendő pénzösszeg, státusz és barcode. Mivel nem áll kapcsolatban a felhasználó mikroszolgáltatással, ezért az itt megadott felhasználó nem feltétlenül található meg az adatbázisban (ismételten hangsúlyozandó, hogy a Movies egy proof of concept projekt, így mellékessé válik az adatok aprólékos ellenőrzése). A hitelkártya ellenőrzéséhez a javax.validation.ConstraintValidator és ConstraintValidatorContext könyvtárakat használja.

4.3. Monitorizálás

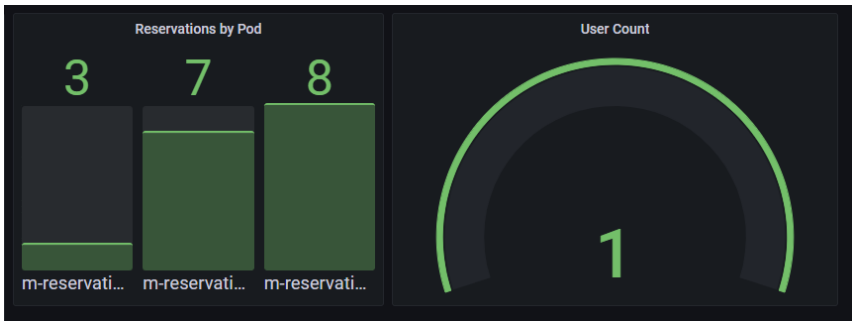
A létrehozott környezet sajátosságainak bemutatásához szükséges egy olyan monitorizációs eszköz, amelyen keresztül nyomon követhető az alkalmazás viselkedése működés közben. Ebben a Prometheus és a Grafana nyújt segítséget.

A Prometheus [14] egy nyílt forráskódú monitorizációs eszköz, amely metrikus adatok begyűjtésével és tárolásával foglalkozik.

A Grafana szintén egy nyílt forráskódú eszköz, amely a Prometheus által begyűjtött adatok megjelenítésére alkalmas. Az adatok Grafana Dashboardon való megjelenítésével (lásd 9., 10., 11. és 12. ábra) egyszerűen nyomon lehet követni az alkalmazás működését, akár a podokig lebontva.



11. ábra. Grafana Dashboard vetítésenkénti foglalások monitorizálására



12. ábra. Grafana Dashboard a foglalások podonkénti leosztására és a regisztrált felhasználók számlálására

Ahhoz hogy a Prometheus adatokat tudjon begyűjteni a mikroszolgáltatások viselkedéséről, szükséges bekonfigurálni ezeknek az adatoknak a továbbítását. Ez egy-egy MetricsConfig osztályon belül történik (természetesen az adatokat a természetüktől függően Controller szolgáltatja működés közben). Ezen belül létrehozhatóak számlálók a választott eseményre (például hány új felhasználó regisztrált vagy hányszor voltak a vetítések listázva). A számlálók rendelkeznek egy-egy egyedi névvel, illetve opcionálisan címkével is. Ezen adatok alapján a Grafana Dashboardon vizulizálhatóak az adatok táblázatok, számlálók, folyamatábrák és más eszközök segítségével.

Következtetések és továbbfejlesztési lehetőségek

A Raspberry Kube projekt célkitűzései közé tartozik a konténerizációs világ és a Kubernetes feltérképezése, valamint egy Raspberry Pi-okon futó Kubernetes klaszter felépítése, bekonfigurálása és monitorizálása.

A dolgozat bemutatja ezen célok elérése érdekében tett lépéseket: a klaszter felépítésének folyamatát, a virtuális klaszter felhúzásának automatizálását, illetve a további konfiguráció lépéseit. Tartalmaz továbbá a projekt világának átláthatóságát elősegítő leírásokat és magyarázatokat is.

Egy következő célkitűzés a létrehozott infrastruktúra használhatóságának bizonyítása különböző alkalmazások kitelepítése révén. A dolgozatban részletbemenően szó esik a Movies alkalmazásról, amely microservice architektúrára alapszik és Spring mikroszolgáltatásokból épül fel. Kevésbé részletbe menően bemutatásra kerül az OnlineNotes Node.js alkalmazás is, illetve a Mandelbrot Image Generator alkalmazás is.

A projekt továbbfejlesztési lehetőségei közé tartozik a klaszter teljesítményének és viselkedésének további tanulmányozása eseményvezérelt alkalmazások kitelepítése révén. Újabb továbbfejlesztési lehetőség a fejlesztési folyamat gyorsabbá és magabiztosabbá tétele egy Spinnaker instancia klaszterre való kitelepítésével. A kódminőség ellenőrzésére kitelepíthető SonarQube instancia szintén egy továbbfejlesztési lehetőségként említhető meg. Ezek mellett egy másik továbbfejlesztési lehetőség a klaszter szintjén az SSL bizonyítványok automatikus kezelése, az Ingress erőforrások alapján, ezzel garantálva a webalkalmazások HTTPS-en keresztüli elérését és a felhasználók adatainak biztonságát.

Hivatkozások

[1] Kelsey Hightower Brendan Burns Joe Beda. *Kubernetes: Up and Running*. O'Reilly Media, 2019.

[2] *Building an Application with Spring Boot*. URL: https://spring.io/guides/gs/spring-boot/ (elérés dátuma: 2022. ápr. 26.)

[3] *Deploy and Access the Kubernetes Dashboard*. URL: https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/ (elérés dátuma: 2022. ápr. 26.)

[4] The Linux Foundation. *Cloud Native Computing Foundation Announces Kubernetes as First Graduated Project*. 2018. URL: https://www.linuxfoundation.org/cloud-containers-virtualization/cloud-native-computing-foundation-announces-kubernetes-first-graduated-project/ (elérés dátuma: 2022. ápr. 26.)

[5] Martin Fowler. *Patterns of Enterprise Application Architecture*. Pearson Education, 2003.

[6] *GitLab CI/CD*. URL: https://docs.gitlab.com/ee/ci/ (elérés dátuma: 2022. ápr. 26.)

[7] *Gradle User Manual*. URL: https://docs.gradle.org/current/userguide/userguide.html (elérés dátuma: 2022. ápr. 26.)

[8] Deepak Gupta. *Duck DNS, the best free dynamic DNS service you can use*. 2022. URL: https://techunwrapped.com/duck-dns-the-best-free-dynamic-dns-service-you-can-use/ (elérés dátuma: 2022. ápr. 26.)

[9] Cheng-Yang Wu Hideto Saito Hui-Chan Chloe Lee. *DevOps with Kubernetes*. Pact Publishing, 2017.

[10] Justin Domingus John Arundel. *Cloud Native DevOps with Kubernetes - Building, Deploying, and Scaling Modern Applications in the Cloud*. O'Reilly Media, 2019.

[11] Kat Cosgrove Jorge Castro Duffie Cooley. *Don't Panic: Kubernetes and Docker*. 2020. URL: https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/ (elérés dátuma: 2022. ápr. 26.)

[12] Raspberry Pi Trading Ltd. *Raspberry Pi 4 Computer Model B*. 2019. URL: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-4-Product-Brief.pdf.

[13] *Operating system images*. URL: https://www.raspberrypi.com/software/operating-systems/ (elérés dátuma: 2022. ápr. 26.)

[14] *Overview: What is Prometheus?* URL: https://prometheus.io/docs/introduction/overview/ (elérés dátuma: 2022. ápr. 26.)

[15] *Raspberry Pi OS*. URL: https://www.raspberrypi.com/software/ (elérés dátuma: 2022. ápr. 26.)

[16] *Red Hat Ansible Automation Platform*. URL: https://www.ansible.com/ (elérés dátuma: 2022. ápr. 26.)

[17] Chris Richardson. *Microservices Patterns with examples in Java*. Manning Publications, 2019.

[18] Skedler Team. *Everything You Need to Know about Grafana*. 2021. URL: https://www.skedler.com/blog/everything-you-need-to-know-about-grafana/ (elérés dátuma: 2022. ápr. 26.)

[19] Dr. Nic Williams. *Building Docker Images for Kubernetes running on ARM*. 2019. URL: https://www.starkandwayne.com/blog/building-docker-images-for-kubernetes-on-arm/ (elérés dátuma: 2022. ápr. 26.)