Szakdolgozat

Miskolci Egyetem

Alkalmazás fejlesztés, CMS rendszerek, Egyedi tartalomkezelő rendszer fejlesztése

Készítette:

Nagy László

2018-2021. Programtervező informatikus-szak

Témavezető:

Dr. Kovács László

Miskolc, 2020

Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Alkalmazott Matematikai Tanszék **Szám:**

Szakdolgozat Feladat

Nagy László LWI9Z1 programtervező informatikus jelölt részére.

A szakdolgozat tárgyköre: Alkalmazás fejlesztés, CMS rendszerek

A szakdolgozat címe: Egyedi tartalomkezelő rendszer fejlesztése

A feladat részletezése:

Szakdolgozat témája egy egyedi tartalomkezelő rendszer (CMS) kidolgozása.
Az alábbi főbb tevékenységpontokra szeretnék kitérni:

- A CMS rendszerek jellemzése, előnyei és hátrányai
- A CMS használatának biztonsági kérdései
- Néhány nyílt forrású tartalomkezelő rendszer összehasonlítása
- CMS motor funkcióinak megtervezése
- Egyedi CMS motor fejlesztése

Témavezető(k): Dr. Kovács László, Általános Informatikai Intézeti Tanszék vezetője, egyetemi tanár

Konzulens(ek):

A feladat kiadásának ideje: 2020. szeptember 30.

. . . . . .
szakfelelős

Eredetiségi Nyilatkozat

Alulírott .....; Neptun-kód: .....
a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős .....
szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy .....
című szakdolgozatom/diplomatervem saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, ...év ..... .hó ..... .nap

.....
Hallgató

1.

szükséges (módosítás külön lapon)

A szakdolgozat feladat módosítása

nem szükséges

....

dátum témavezető(k)

2. A feladat kidolgozását ellenőriztem:

témavezető (dátum, aláírás): konzulens (dátum, aláírás):

....

....

....

3. A szakdolgozat beadható:

....

dátum témavezető(k)

4. A szakdolgozat ..... szövegoldalt

..... program protokollt (listát, felhasználói leírást)

..... elektronikus adathordozót (részletezve)

.....

..... egyéb mellékletet (részletezve)

.....

tartalmaz.

....

dátum témavezető(k)

5.

bocsátható

A szakdolgozat bírálatra

nem bocsátható

A bíráló neve: .....

....

dátum szakfelelős

6. A szakdolgozat osztályzata

a témavezető javaslata: .....

a bíráló javaslata: .....

a szakdolgozat végleges eredménye: .....

Miskolc, ...

a Záróvizsga Bizottság Elnöke

Tartalomjegyzék

1. fejezet

Bevezetés

Manapság az internet világa elég meghatározó szerepet vállal az életünkben. Az interneten keresztül szerzünk általában információkat a világban történt dolgokról, itt olvasunk különféle témában híreket, érdekes cikket, fórumokat, tehát leginkább az internet által tájékozódunk. Ezen okok végett, ha például egy magánszemély, vagy egy vállalat a saját termékét szeretné népszerűsíteni, nélkülözhetetlen hozzá egy weboldal, ahol ezt reklámozza. Ehhez alapvetően szüksége lesz webfejlesztői tudásra, ebben a felhasználók nagy segítségére lehetnek a tartalomkezelő rendszerek, amelyeket megfelelő szaktudás nélkül is egyszerűen lehet használni, hogy a saját weboldalunkat létre tudjuk hozni.

1.1 Célkitűzés

Szakdolgozatomban szeretném feltárni a tartalomkezelő rendszerek használatának az előnyeit, hátrányait, illtetve ezek típusait.

Egy külön fejezetben szeretném megválaszolni a CMS rendszerek információbiztonsági kérdéseit, majd fő célként egy egyedi fejlesztésű CMS rendszer implementálását tűztem ki.

Ez a hagyományos szoftverarchitekrúrájú rendszereken túl, egy sokkal hatásosabb, modulárisabb felépítésű rendszert eredményez. Ez egy microservice alapú megoldás segítségével fog létrejönni.

1.2 Motiváció

Azért esett a választásom ezen alkalmazás fejlesztésére, mivel a CMS rendszerek egy olyan megoldást nyújtanak, ami a webfejlesztést hatalmas mértékben megkönnyíti és meggyorsítja.

Manapság az emberekben is egyre jobban feltámad a vállalkozói ösztön, nagyon so-

kan döntenek úgy, hogy a saját ötleteiket meg szeretnék valósítani és egy vállalkozás keretében ebből profitot szeretnének termelni.
A vállalkozásokon kívül is nagyon sokoldalú a tartalomkezelő rendszerek felhasználóinak köre. Magánszemélyként is létrehozhatunk weboldalakat például egy saját blog vezetéséhez, fórumok kezeléséhez, vagy esetleg egy-egy sportcsapatnak ez jelenti a legfőbb útját a kommunikációnak a szurkolók, érdeklődők felé.
Tehát elég széles körben használják a CMS-eket, ezért nagyon fontos, hogy minél jobban ki legyenek dolgozva ezek a rendszerek, minél több lehet?séget kínáljon, hogy mindenki megtalálja benne a számára megfelelő funkcionalitást.

Ez viszont azt eredményezi, hogy ezek a CMS-ek nagyon nagy rendszerekké válnak és a modularitás fontossága is egyre jobban háttérbe szorul. Erre jelenthet megoldást a microservice technológia.

Microservice-k segítségével ugyanazokat a funkciókat kisebb egységekben tudjuk eltárolni, ezáltal jobban kezelhetőek lesznek ezek a rendszerek és biztosítani tudjuk a moduláris felépítést.

2. fejezet

CMS-ek általános ismertetése

2.1 CMS definíció [1]

A CMS (Content Management System), olyan szoftver, amely speciális műszaki ismeretek nélkül segíti a felhasználókat egy weboldal tartalmának létrehozásában, kezelésében és módosításában.

Ezekhez a rendszerekhez-a weboldal szerveren történő telepítésén kívül-nincs szükség fejlesztői tudásra, minimális szakértelem mellett üzemeltethető a weboldal. Egyszerűen tudunk felhasználókat, tartalmi elemeket létrehozni és módosítani anélkül, hogy ismernénk a webes nyelveket, kódokat. A CMS rendszerek célja az egyszerű tartalomfelvitel, gondolok itt mind szövegre, képre, vagy bármilyen csatolt állományra és ezek elérhetővé tétele az oldalon a látogatók számára.

Egyik legfontosabb tulajdonsága ezeknek a rendszereknek a dinamizmus. Ez azt jelenti, hogy megfelelő jogosultság birtokában az oldal tartalmát és annak megjelenését bármikor, bárhonnan meg lehet változtatni.

A tartalomkezelő rendszerek template-k segítségével választják el egymástól a tartalmat és a megjelenést. Ezek a sablonok úgymond keretrendszerként működnek, mivel esetleges arculatváltás esetén elegendő csak a template-t lecserélni és a tartalom-alkalmazkodva ehhez a sablonhoz-változatlan formában, de új külsővel jelenik meg.

A későbbi bővítés is egyszerűen kivitelezhető, mivel ha új funkciókat szeretnénk integrálni a rendszerbe, akkor az grafikai átalakítással nem jár, nem kell újraszerkesztenünk az érintett részt.

Ezekre a rendszerekre továbbá jellemző a moduláris felépítés, valamint ezek többfelhasználós rendszerek, tehát egyszerre többen is tudják szerkeszteni az oldalt. A legtöbb CMS keresésoptimalizált, így könnyen tudunk keresni a honlapon elhelyezett tartalmak között.

A CMS rendszerek legfőbb tulajdonságai:

- Könnyű kezelhetőség
- Szelektív hozzáférési jogosultságok

- Többnyelvűség
- Keresőoptimalizálási eszközök
- Reszponzivitás
- Nagyon kis költségűek

2.2 Mikor használjunk tartalomkezelő rendszert?

CMS rendszert leginkább olyan weboldalnál érdemes használni, ahol fontos az aktív kommunikáció, tehát ahol az oldalra látogatók számára folyamatos információt kell közvetítenünk, ezeket frissíteni kell. Ilyenek például a fórumok, blogok, portálok, webáruházak.

A tartalomkezelő rendszerek tehát összességében segítséget nyújtanak mind például a magánszemélyeknek a saját személyes blogoldaluk kialakítására, vagy pedig a vállalatoknak, cégeknek a weboldalának létrehozását, csökkentve ezzel a kiadási költséget. Nem kell megbízniuk egy webfejlesztő céget az oldaluk elkészítésével, ezt megtehetik maguknak, nagyobb erőfeszítés nélkül.

2.3 CMS felhasználókezelés

A CMS rendszerben nagyon fontos funkció a felhasználó kezelés. Minden rendszernek van egy adminisztrátora, aki teljes jogkörrel rendelkezik, bármit megtehet az oldalon. Ő osztja majd ki a többi felhasználó számára a megfelelő jogosultságokat. Az azonosítás regisztrálás és bejelentkezés útján történik.
Ezek alapján az oldalra látogatók két csoportba sorolhatók: azonosított felhasználók, akik már regisztrált tagjai a rendszernek, saját felhasználónévvel és jelszóval rendelkeznek, mellyel beléphetnek a webhelyre. Jogosultságaik korlátozottak, a jogokat az adminisztrátor rója ki.

És vannak az ismeretlen felhasználók, akik ugyan ellátogatnak a webhelyre, de még nem tagjai annak, nem rendelkeznek saját regisztrációval. Ezen felhasználók általában csak a közzétett tartalmakat olvashatják, illetve menthetik le, egyéb jogosultságuk nincs. Bizonyos esetekben számukra is nyitott néhány opció: publikus szavazásokban vehetnek részt, vagy anonimként hozzászólásokat fűzhetnek bizonyos tartalmakhoz. Ezekről szintén az adminisztrátor rendelkezik.

A weboldal típusától függően hozhatunk létre különböző jogosultsági köröket, így akár nem egy-egy személynek osztjuk ki a jogosultságokat, hanem a felhasználókat használat szerint csoporokba rakjuk és ezeknek a csoportoknak adjuk meg a jogosultságait.

2.4 CMS választás

Számos CMS platform közül tudunk választani, amik egymástól eltérő tulajdonságokkal rendelkeznek.

Dönthetünk akár a szabad licencű, nyílt forráskódú rendszerek mellett, vagy akár a költségesebb egyedi fejlesztés mellett is.

Vannak egyszerűbb és komplexebb rendszerek, magyar nyelvűek, vagy csak idegen nyelvűek.

Különféle szerverkörnyezetet használnak, például Java, PHP, .Net. Vannak általános célú CMS-ek és speciális verziók, mint például az e-learning oktatási céllal.

Számos CMS rendszert a használata előtt ki lehet próbálni, ez meghatározó lehet a döntésünkkor, mert így tesztelni tudjuk az előre telepített rendszereket. Ez nagy előnyként számít a nyílt forráskódú CMS-eknél. Ezeket a CMS Award oldalon, illetve az Opensource CMS oldalon van lehetőségünk tesztelni.

Magyarországon talán a legelterjedtebb CMS-ek a Drupal és Joomla, mint általános célú rendszer, a Moodle oktatási oldalak fejlesztéséhez és a Wordpress, ami a blogok esetén a legnépszerűbb.

Mielőtt kiválasztjuk a megfelelő CMS rendszert, tisztába kell lennünk a CMS-ek tulajdonságaival.

2.5 CMS típusok [2]

Ebben a részben szeretném ismertetni a legnépszerűbb CMS-eket a felhasználók körében. Ezek pedig a Wordpress, Drupal és a Joomla.

Ez a három rendszer merőben eltér egymástól, de találunk közös vonásokat mind a működésükben, mind pedig felépítésükben, ugyanis mindhárom CMS mögött PHP programnyelv és MYSQL adatbázis van.

Mindhárom személyre szabható, mert további kiegészítőkkel bővíteni tudjuk, és a weboldalak kinézetét gyorsan meg tudjuk változtatni a különféle sablonok telepítésével. Ez független a honlap tartalmától, mivel a weboldal adatai adatbázisban vannak eltárolva, így az esetleges módosítás nem veszélyezteti a már meglévő adataink elvesztését.

Most pedig szeretném egyesével ismertetni a három legkedveltebb tartalomkezelő rendszert.

Wordpress

2003-ban látta meg a napvilágot, leginkább blogok tervezésére fejlesztették, azonban napjainkra már rengeteg weboldal alapja, az egyszerű bemutatkozó oldalaktól kezdve webshopokon át, a teljes körű szociális hálózatokig.

Népszerűségének kulcsa a könnyű használhatóság, egyszerűség, és a jól beállítható keresőoptimalizálási paraméterek. Nincs szükség komolyabb programozói ismeretekre, hogy

egy weboldalt létrehozzunk magunknak. A Wordpress egyszerű telepítésével már megoldható ez.

Alap weboldalunkat további kiegészítőkkel bővíthetjük, amelyekkel növvelhetjük a weboldal funkcionalitását, így akár webáruházzá is fejleszthető. A telepíthető sablonok segítségével gyorsan és egyszerűen változtatható meg a weboldalunk kinézete. A widgetek, azaz kisebb kiegészítő programok pedig tovább javítják a Wordpress használhatóságát.

A Wordpress rendelkezik a legnagyobb fejlesztői közösséggel, így számtalan oktatóanyag is a rendelkezésünkre áll.

Drupal

A Drupal tartalomkezelő rendszer volt az első a három közül, amelyik megjelent. 2000-ben született meg a drop.org weboldal forráskódjaiból. Ez a CMS szakértők számára készült, így nem meglepő, hogy ez a legkevésbé használt webmotor.

Ugyanúgy megváltoztathatjuk weboldalunk kinézetét különféle sablonokkal, az alaprendszert pedig modulok használatával tovább tudjuk fejleszteni, továbbá PHP és MySQL ismeretekkel egyedi fejlesztésű modulokat tudunk beintegrálni a weboldalunkba.

A Drupal rendszer közössége is nagyon aktív, számos fórumon tudunk keresgélni segítség után. A Drupal dokumentációja folyamatosan frissül, így minden információ rendelkezésre áll a CMS telepítésétől kezdve a weboldalépítés és a kiegészítők telepítésének folyamatáról.

Ez a rendszer nem javasolt egyszerűbb weboldalak esetében, mivel tartalmaz olyan kiegészítő?ket, amik komplexebbek és nem feltétlen van rá szükségünk a weboldalunkhoz, tehát a weboldalunk nem használná őket, de mégis lassulna ezek miatt.

Kevés ingyenes sablon áll rendelkezésünkre, amik viszont nem biztosítják a WordPress vagy a Joomla által kínált minőséget. A felhasználói felület is sokkal kevésbé felhasználóbarát.

2005-ben Mambo néven jött létre, így a három CMS közül ez számít a legfiatalabbnak.

Joomla

A Joomla fejlesztését egy nagy közösség végzi, saját MVC fejlesztői keretrendszerrel. Hatalmas előnye a Bootstrap integrálása és amiben különbözik a másik két rendszertől, és ez nagy előnyére is vált, az az, hogy ennél teljesen reszponzív adminisztrációs felülettel találkozunk. Az alap rendszert bővíthetjük további modulokkal, pluginekkel. A telepíthető sablonok segítségével pedig pillanatok alatt megváltoztatható a weboldal kinézete.

Nagyon aktív fejlesztői táborral rendelkezik, ezért könnyen elérhető oktatói anyaggal rendelkezik és a felmerülő kérdéseinkre gyors választ kaphatunk a fejlesztői fórumokon.

Azonban a WordPress-el összehasonlítva kevésbé felhasználóbarát és a weblap sablonok is kevésbé színvonalasak.

2.6 Nyílt forráskódú, vagy egyedi feljesztés? CMS? [3,4]

Miután arra jutott döntésünk, hogy weboldalunkhoz CMS rendszert fogunk alkalmazni, fel kell tennünk magunknak a kérdést, hogy nyílt forráskódú, vagy egyedi fejlesztésű tartalomkezelő rendszer mellett döntünk. Ezt több tulajdonság alapján érdemes eldönteni, ezeket szeretném bemutatni ebben a fejezetben.

Először szeretném bemutatni a nyílt forráskódú CMS előnyeit és hátrányait. Ez egy nyílt forráskódú tartalomkezelő rendszer, amelyet egy közösség fejleszt, és mind a termék, mind a teljes forráskód szabadon és ingyenesen bárki rendelkezésére áll. Ilyenek például a WordPress, Joomla, vagy a Drupal. A nyílt forráskódú CMS-ek nagyon elterjedtek, minden iparágban találkozunk ezzel a technológiával megvalósított weboldalakkal. Legnagyobb vonzereje talán az ingyenességben rejlik, és abban, hogy előre ki tudjuk próbálni.

Ezzel szemben az egyedi fejlesztésű CMS egy olyan, jellemzően zárt forráskódú rendszer, amelyet általában egy fejlesztőcsapat, vagy cég saját maga fejleszt ki a nulláról, és a használatáért licenszdíjat kell fizetni, viszont a forráskódja rejtett. Legnagyobb előnyük lehet talán az, hogy kifejezetten egy bizonyos weboldalra specializálják, így a vevő igényei szerint készül el az adott oldal és könnyen optimalizálható és egyedisége kitűnik a sok szabványos template-k közül.

A nyílt forráskódú CMS előnyei lehetnek:

- Ezek a rendszerek önmagukban jellemzően ingyenesek. Ebben az esetben csak azért a szolgáltatásért és szaktudásért kell fizetni, amellyel a fejlesztők telepítik a rendszert és a későbbiekben esetlegesen frissíti.
- Továbbá fontos érv még, hogy ezek a rendszerek akár minimális szaktudással adminisztrálhatóak.
- Egy hozzáértő fejlesztő könnyen tudja bővíteni különféle egyedi funkcióval, mivel kódja nyílt, így bárki számára elérhető.

- Ezeknek a rendszereknek a fejlesztésében több millió felhasználó vesz részt. Ez a hatalmas fejlesztői bázis is nagy előnynek számít.

A nyílt forráskódú CMS-ek ellen felhozott érvek:

- A rendszer folyamatosan frissül, ami viszont számunkra lehet, hogy nem szolgál kedvező eredménnyel. A frissítések során előfordulhat, hogy egy-egy korábban probléma nélkül működő plugin már nem fog működni, vagy hibázni fog. Ezek ellen nem sok mindent tudunk tenni.
- A nyílt forráskódú rendszerek népszerűsége jelenti egyben a sebezhetőségüket is. A nyitott forráskód miatt mindenki számára elérhető a kód, ezzel együtt a jelszavak tárolási miként tje, vagy a modulok felépítése, így elég sokan próbálkoznak azzal, hogy ezeket a rendszereket feltörjék. Látják ebben a pénzszerzési lehetőséget, ezért ezeknek a kódoknak a feltörésére számos támadási kísérletet tesznek, próbálnak rést találni a rendszerek biztonsági pajzsain.
- A CMS rendszerek tartalmazhatnak olyan file-okat, kódokat, plugineket, amik számunkra feleslegesek, de oldalunkat lassítják.
- Nehezebben optimalizálhatóak.

Az egyedi fejlesztésű CMS-ek előnyei:

- Az egyedi fejlesztés miatt ezek személyre szabott rendszerek, az ügyfél igényeire van összpontosítva. Nem kerülnek bele soha nem használt funkciók és menüpontok amik lassítanák a rendszert, így a kívánt feladat leghatékonyabb megoldására fókuszál.
- Az esetleges hibák felmerülésekor a rendszer fejlesztőihez fordulva akár pár percek alatt megoldást találhatunk a problémákhoz, nem kell fórumokon keresgélnünk. A licenszdíjért cserébe megbízható, professzionális support-ot kapunk.
- Bármilyen ügyféligény könnyen megvalósítható az integritás megtartása mellett.

Az egyedi fejlesztésű CMS-ek ellen felhozott érvek:

- Egyik legnagyobb hátránya a költséges és időigényes beüzemelés. Egy egyedi fejlesztésű rendszer kiépítése sokkal több idővel és pénzzel jár, mint egy nyílt forráskódú rendszer használata.
- Egy másik hátránya, ami nagyon kis valószínűséggel fordul elő, vizont ezzel is számolni kell, mégpedig, hogy valamilyen okból elveszítjük a kapcsolatot a fejlesztő csapattal. Ilyenkor a weboldal továbbfejlesztése lehetetlenné válik.

2.7 Java alapú CMS-ek [5]

2.7.1 Hozzátartozó definíciók

Portál rendszerek:

A portál egy olyan web alapú alkalmazás, ami különböző webes, illetve háttérrendszeres tartalmaknak biztosít egységes prezentációs felületet. A portál az oldalain elhelyezett linkek segítségével, belépési pontként funkcionálhat az Internet-re.

A portál rendszereket leginkább PHP vagy Java technológiával fejlesztik.

Portletek:

A portlet a portál egy kisebb része. Web alapú komponensek, felhasználói interfésszel. A felhasznló kérései feldolgozásából dinamikus tartalmat generál a felhasználó felé. Ez a tartalom lehet akár HTML, XHTML, XML stb. A portleteket a portletkonténer vezérli. Működésüket a JSR-168 és JSR-286 java specifikációk írják le. Egy portál oldalon több különböző portlet is futhat, ezek akár külön ablakokban is futhatnak.

Portlet konténer:

A portlet konténer a portál része. Ebben futnak a portletek. Feladata létrehozni a portlet példányt, biztosítja a futási környezetet a portlet számára, kéréseket küld a portlt felé, fogadja tőle a válaszokat és felszabadítja a portleteket, maikre már a futás során nincs szükség.

2.7.2 Nyílt forráskódú Java portálok

Több ingyenes, szabadon felhasználható, nyílt forráskódú Java portált használhatunk. Ezekre egy pár példa:

- EXO Platform
- Liferay
- Jakarta Pluto
- Gridsphere
- jPortlet
- Stringbeans
- Kosmos
- JBoss Portal

2.7.3 A Liferay bemutatása

Az előbb felsorolt példák közül pedig szeretném ismertetni a Liferay-t.

A Liferay portál rendszer az egyik legelterjedtebb open source portál rendszer Java platform alatt. A Liferay egy nyílt forráskódú, Java alapú portál-keretrendszer. A közösségi verzió (Community Edition) ingyenesen letölthető, bárki számára használható, viszont van egy vállalatoknak szánt változata is (Enterprise Edition).

A Liferay magában foglal egy CMS-t, valamint szabványos portlet konténerként is viselkedik. Sok beépített általános célú portletet tartalmaz az alapváltozat, ilyenek például a blogok, fórumok, oldalak, wiki.

A java-s alkalmazás szerverek bármelyikén képes futni, ehhez szükség van egy Web Container megvalósításához. Beépített adatbázissal rendelkezik (HISQL), így adatbázis nélkül is képes a futásra, viszont a performancia miatt ezt éles környezetben nem érdemes használni, ezért érdemes ezt standard JDBC driverrel rendelkező adatbázisra cserélni.

2.8 A CMS-ek információbiztonsági kérdései [6]

Végezetül szeretnék kitérni még egy fontos témára a CMS-eknél. Ez pedig az információbiztonság. Hogy mit is jelent ez pontosan? Az információ és az informatikai rendszerek védelmét jelenti az illetéktelen hozzáféréstől, használattól, az információ közzétételétől, módosításától, visszaéléstől, elsajátításától és törlésétől.

Miért is kell beszélnünk ezek veszélyeiről?

A CMS-ek népszerűsége egyben nagy hátránya is ezeknek a rendszereknek. Mivel nagyon sokan használják ezt a fajta rendszert, és elég sok felhasználónak nincs is meg a hozzá értő tudása, ezért nem tudják kellően biztonságosan használni ezeket. Ez ugyanúgy előfordulhat a szakértőknél is kellő figyelemráfordítás hiányában. Így egyre kedvezőbb lehetőség ez a hackerek számára.

Először is szeretném kiemelni a tartalomkezelő rendszerek támadásainak legfőbb okát, ez pedig a pénzszerzés. Feltörik a rendszert információszerzés céljából, majd ezáltal plusz bevételhez juthatnak. További indok lehet még a vírusok (malware) bejuttatása a rendszerbe. Ezzel akár több számítógép felett is átvehetik az uralmat, használhatják erőforrásaikat. Egy másik indokként felhozható még a bosszúvágy is, vagy akár csak a hackereknek ez egyfajta gyakorlás is lehet, hogy megkeressék a rendszerek sebezhetőségét és gyengeségeit. Ezen felül léteznek még az állam által szponzorált hackerek, akiknek céljuk a kibertér minél magasabb szintű kontrollálása, valamint a vállalati világban is léteznek úgynevezett kém hackerek akik a versenytársak adatainak megszerzésével vannak megbízva. A legveszélyesebb támadók a kiberbűnözők, más néven kiber terroristák. Nekik egyetlen céluk a rombolás.

A rosszindulatú hackerek mellett meg kell említenünk a jóindulatú hackereket is. Ezeket úgynevezett fehér kalaposoknak hívjuk, őket vállalatok bízzák meg azzal, hogy rend-

szereik sebezhetőségeire hívják fel a figyelmüket.

Hogy tudunk ezek ellen védekezni?

- Biztonsági mentés az adatokról és a honlapról
- Shared hosting helyett dedikált szerver használata
- Legfrissebb verzió használata (alkalmazások, pluginok, kiegészítők)
- Szükségtelen kiegészítők törlése, csak megbízható pluginok integrálása
- Felhasználónevek és jelszavak erősségére fordított kellő figyelem
- Számítógép vírusmentesítése
- Rendszer monitorozása
- CMS rendszer verziószámának elrejtése

Tehát rendszerünk és adataink biztonsága érdekében érdemes megfelelő figyelmet kell biztosítani ezekre, annak érdekében, hogy a bizalmasság, integritás, és hozzáférhetőség faktorokat kellően meg tudjuk védeni az esetleges támadásoktól.

3. fejezet

Fejlesztői dokumentáció

Ebben a fejezetben ismertetni fogom a fejleszteni kívánt rendszer tervezését, architekrúrális tervét, illetve az implementálás során használt technológiákat. A rendszer elkészítése szakértők tapasztalatai alapján és segítségével történik.

3.1 Koncepció

A webfejlesztésben egy nélkülözhetetlen gyors fejlesztést segítő megoldás a CMS. A legismertebb piacon lévő megoldásoknál a hagyományos szoftverarchitektúrájú rendszerek vannak, a web-es területen azonban mára már szinte egyeduralkodóvá kezd válni az új fejlesztések területén a front-end és back-end kettéválasztása.

A back-end esetében pedig lehetőség nyílt, egy sokkal inkább sokoldalúbb, modulárisabb fejlesztésre. Konkrét példával élve ha veszünk egy Drupal, Liferay esetében egy webszerverre van telepítve egy szolgáltatás. Az nyújtja az oldalak, tartalmak kezelését. Azonban a back-end esetében az architektúrára jellemzően vagy csak java, vagy csak php-s bővítési lehetőség van.

Milyen új CMS architektúra dolgozható ki? Ha a fenn említett szempontokat figyelembe vesszük, akkor erre egy olyan microservice alapú megoldás adhat választ, ami könnyen telepíthető, bővítésre nyitott, módosításra zárt. Interface alapú megoldások. Jelen esetben az interfacenek a REST-es végpontokat is tekinthetjük.

A dolgozatban egy tartalomkezelő kerül kidolgozásra. A tartalomkezelő microservicek összessége, ami a hagyományos CMS-esek esetében egy adott oldal html tartalmát határozza meg, maximum néhány makró kifejezéssel.

Fontosabb funkciók:

- Lehessen verziózni a tartalmakat (pl.: jogi nyilatkozat oldalát)
- Legyen staging mód (piszkozat kezelés, illetve élesítés előtt lehessen látni a leendő eredményt)
- Multi platformos (desktop, mobilos verziók kezelése)
- Makrók használati lehetőségének biztosítása (bővítésre nyitott, módosításra zárt elv)

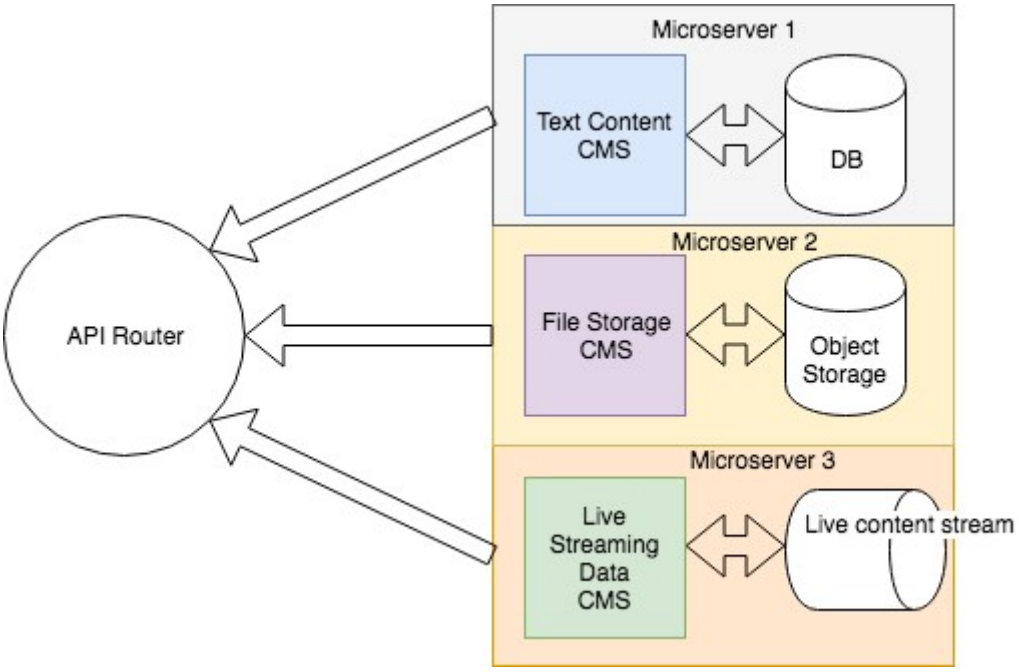
- Egyedi kontextuskezelőnek integrálhatósága (egy makró milyen adatokból dolgozhat, rendszerhez igazított session kezelés)

Makrók, templatek előnyei:

Vannak esetek, mikor a tartalom nagy százalékban statikus, azonban vannak bizonyos elemei, akár a felhasználó neve, adott év, stb. Ezeket ne kelljen mindig módosítgatni, ezért valamilyen template leíróval kezelhető. Azonban a tervezés során nem szeretnénk egy nyelvre korlátozni, sokkal inkább szabadon igényeknek megfelelően bővíthetővé tervezni. Ehhez kapcsolódik szorosan a kontextuskezelő amelyet a templatel összekapcsolva fog előállni a tényleges tartalom.

3.2 A Microservicekről általában [7]

A microservicek nagyon hasonlóak egy API-hoz, amit egy backend szerver szolgáltat.



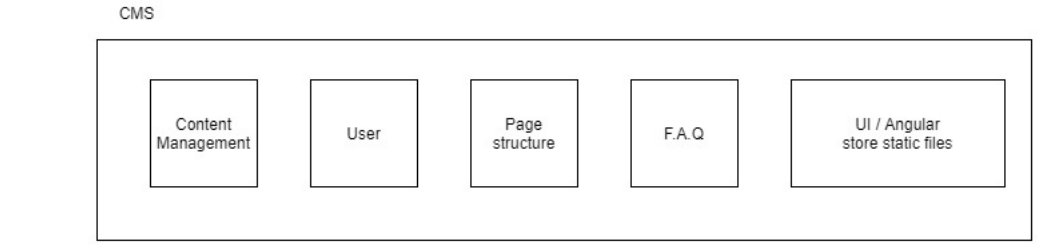
Microservice architektúrában számos ilyen backend szervert sok kisebb önálló részre osztunk fel. Fentebb láthatunk egy ilyen megvalósítást egy CMS rendszer példáján. Mindegyik szerver egy microservicet nyújt, amik egy bizonyos feladatot végeznek el.

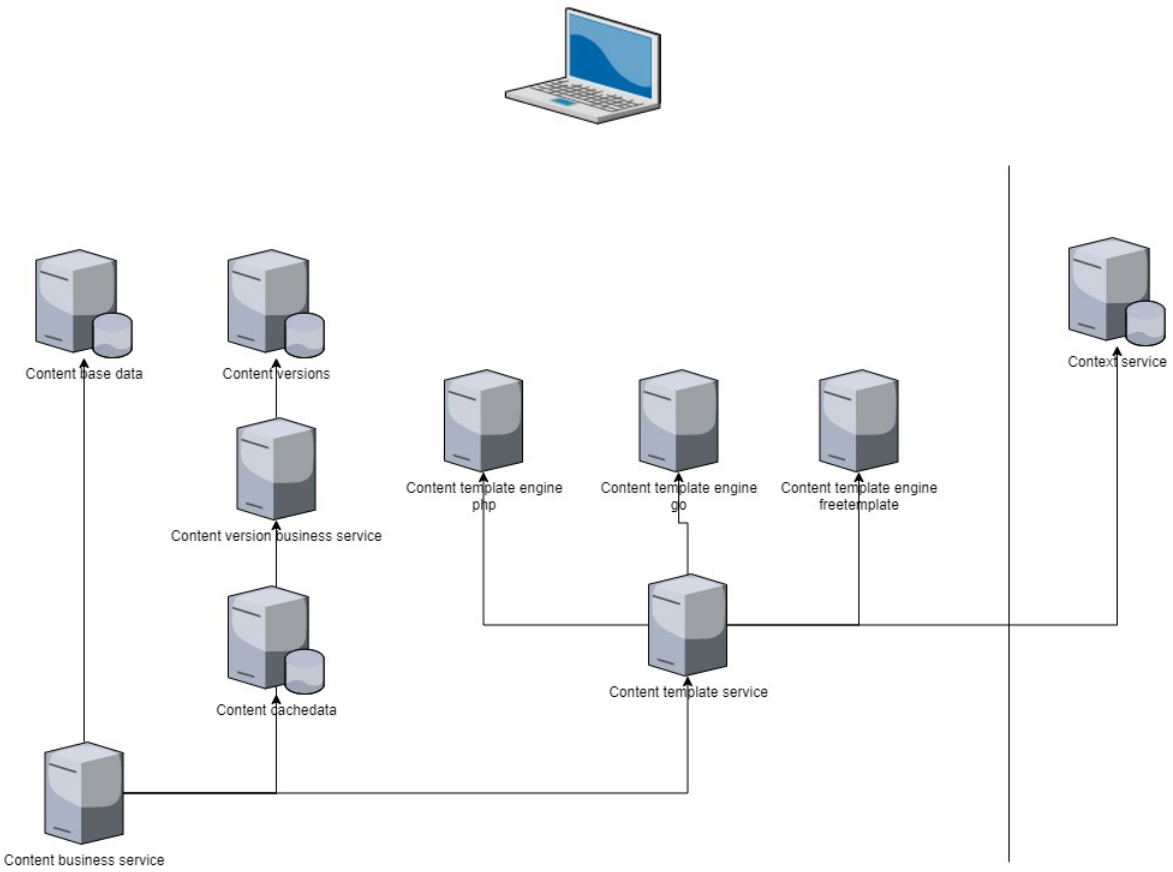
A microservice architektúra legfőbb jellemzője a szolgáltatások modularizációja és a szoftverek felosztása kisebb, egymástól független komponensekre.

Ez a fajta technológia megoldást nyújt arra, hogy a nagy, komplex rendszereket felosszuk sokkal kisebb komponensekre, amik API útján kommunikálnak egymással, ezáltal sokkal jobban menedzselhető a rendszerünk. Ezzel a módszerrel egy sokkal rugalmasabb és könnyebben bővíthető rendszert tudunk elérni. Ha egy microservice meghibásodik, nem kell az egész szoftvert frissítenünk, elég csak a microservicet kicserélni, javítani. A program csak egy kis, önálló részét kell frissíteni.

További előnye, hogy nem vagyunk rákényszerítve az egész rendszeren belül egy programnyelv használatára. Mivel a microservice komponensek API útján kommunikálnak, így a Microserviceket megírhatjuk akár más-más programnyelveken is, a működésükben ez nem zavarja őket. Tehát programnyelv függetlenné teszi a komponenseket. Bármilyen technológiát használhatunk az egyes microservicek implementálásához. Ezáltal egy sokkal gyorsabb fejlesztést is el tudunk érni, mivel ezek a komponensek függetlenek egymástól, tehát egy fejlesztőcsapat akár feloszthatja ezeket és dolgozhatnak a rendszer egységein külön-külön is. Ráadásul így egy sokkal jobban rendszerezettebb, átláthatóbb és megérthetőbb rendszert kapunk.

3.3 Programspecifikáció





Az ábra ismertetése:

Content modul

Ennek a cms modulnak a feladata tárolni és kezelni a különböző tartalmakat. Ebbe beleérthetőek statikus tartalmak, amelyek például egy portál nyitó oldala vagy akár egy cikk szerkeszt? is, amely komplexebb dinamikus tartalmakat is tartalmazhat. Cél, hogy verziózni lehessen a tartalmat, több állapotot is tudjon kezelni (pl.: production, staging).

Microservicek

Content business service

Ez a service adja a modul egyik API-ját. Ezen keresztül érhetőek el a tartalmak. Mind a generált, mind a statikus tartalmak. A tartalom hozzáadása, módosítása is ezen keresztül történik. Feladat az ábra szerint is, a modulhoz tartozó további servicek közötti közvetítés.

Első lépésként, megnézni, hogy az adott azonosítóhoz milyen alap adatok tartoznak, ami alapján validációkat, illetve a válasz modell alapján létre lehet hozni. Azonban a microservicek előnyét kihasználva, ezzel párhuzamosan kiküldésre kerül a tartalom felé is a kérés, hogy az adott azonosítóhoz határozza meg a kívánt tartalmat. Ezek megérkezése után, megvizsgálásra kerül, hogy statikus tartalomként azonnal kiszolgálható-e a válasz, vagy dinamikusként template service felé irányítandó a feldolgozás további folyamata. A template enginek válasza után pedig a végs? modell összeállása utána megkapjuk a teljes választ.

Content base data

Ez a microservice határozza meg a legelemibb adatokat. A tartalomhoz tartozó metainformációk ebben lesznek tárolva (pl. név, leírás, stb). Itt lesz tárolva az adott tartalom azonosító, amelynek segítségével lehet rá hivatkozni. Illetve a feldolgozáshoz szükséges adatokat is tartalmazza. Például a későbbi microservice leírásban definiált content template enginehez szükséges adatok.

Content versions

Az adott azonosítóhoz tartalmazó tényleges adatot reprezentálja. Tárolás szempontjából a kódokhoz használatos verzió kezelőt alapul véve minden alkalommal az előző verzióhoz képesti különbséget (deltát) tárolja el. Így a lehetőséghez mérten kisebb fizikai tárhelyet foglal, mintha minden alkalommal a teljes tartalom tárolásra kerül. Egy-egy módosítási bejegyzés tartalmazza, hogy ki és mikor adta hozzá, illetve egy üzenet mező is letárolásra kerül.

A verziózás mellett meg lehet jelölni egy-egy módosítási bejegyzést. Azért fontos, mert így valósítja meg a különböző állapotok kezelését. Tehát ha egy bejegyzést megjelölünk staging vagy production tag-el, akkor lekérdezésnél ahol a staging állapotot szeretnénk megkapni, a tag alapján megkapjuk a hozzátartozó állapotot.

Content cache data

Mivel a content service esetén csak delták tárolódnak, így performancia szempontjából megvizsgálandó, hogy szükséges-e - ha igen milyen mértékben - gyorsító tárak létrehozása. A gyorsító tárak pontok adatbázis megoldása az első mérések, után érdemes vizsgálni. Itt értve, hogy a megoldás esetén, ha több példányban is fut, akkor memóriában is szinkronizálni kell, vagy elegendő-e lokálisan tárolni. Esetlegesen a nagy tartalmakra való tekintettel, adatbázisban történő tárolás is jó megoldást nyújthat.

Content template service

Ez a service a Content template engine motorok implementációit kezeli, koordinálja. A több engine megoldás elsődleges cél volt, hogy támogatva legyen a fejlesztők, tartalom szerkesztők számára, a számára már ismert, bevált technika használata. Ennek további előnye, hogy meglévő rendszer migrációja során sokat könnyíthet a folyamaton.

Content template engine (freetemplate, go, php, stb.)

Egy olyan elemi service, amely megkap egy template tartalmat és hozzá tartozó context modell adatot és kiértékeli. Mivel sok fajta template kezelő rendszer megoldás létezik, ezért lehetőség van akármelyik megvalósítására. A dolgozatban nem célom értékelni a különböző megoldásokat, implementáció szempontjából népszerűség és személyes ismeret alapján kerül kiválasztásra.

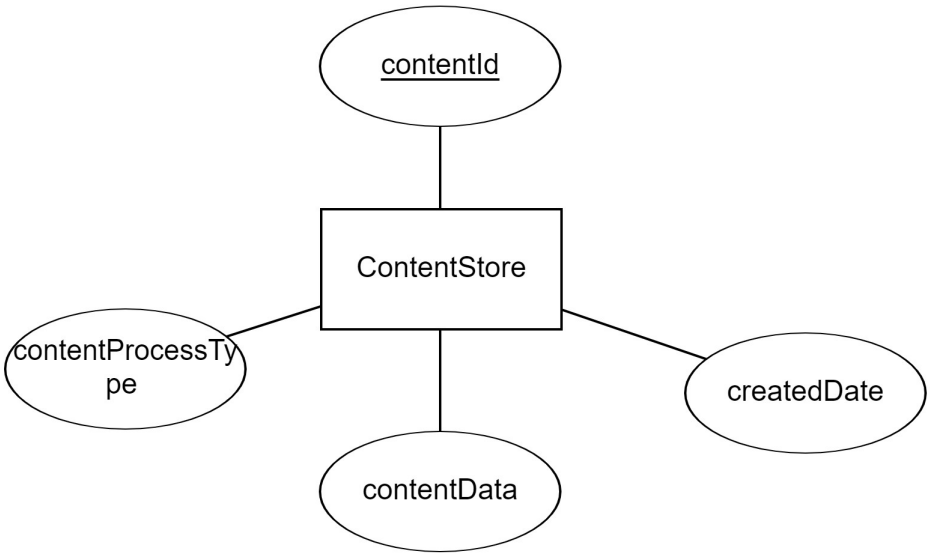
Context service

Ez egy integrációs pontja is a rendszernek. Ennek a feladata, hogy előállítsa azt a modellt, aminek változói használhatóak a templateben. Ilyen alapelemek például az aktuális év, dátum, rendszer megnevezése, a contenthez kapcsolódó alapadatok, felhasználói adatok. Ez egy CMS bevezetés során egyedi modulok hozzáadásával tovább fejleszthető, bővíthető.

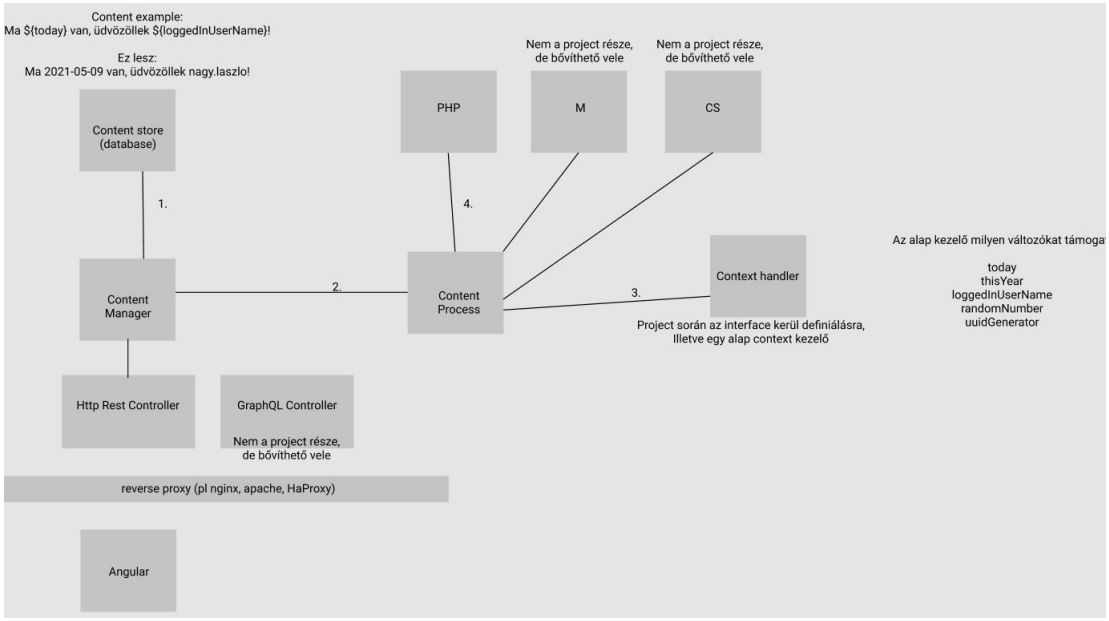
Adatbázis:

A microservicek közül egyedül a Content Store microservice igényel adatbázist. Ebben tárolódik a megjeleníteni kívánt tartalom. Ez az adatbázis tartalmazza a következő adatokat:

- Egy integer típusú contentId, ami elsődleges kulcs szerepel rendelkezik az azonosítás érdekében.
- Egy contentProcessType, ami String típusú adatot tartalmaz. Ugye ez adatbázistól függően lehet char, varchar stb. Ez alapján fogja meghívni a megfelelő Content Processort.
- Egy createdDate mező, ami a létrehozás dátumát tárolja el.
- Egy contentData mező, ami magát a megjelenítendő tartalmat tárolja.



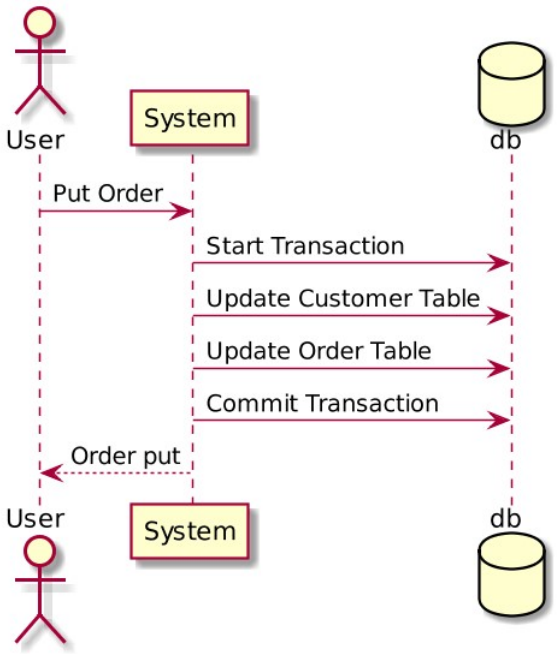
A rendszerben szereplő microservicek:



3.4 Használt technológiák ismertetése

3.4.1 Tranzakció kezelés

Manapság egyre elterjedtebb körben használják a microservice architektúrát. A legnagyobb kérdést ennek használatában a microservicek közötti tranzakció kezelés jelenti. A monolit rendszerben egy helyi tranzakció több szolgáltatásra oszlik fel, amelyeket egymás után hívnak meg.



A fenti példában, ha a User küld egy Put Order action-t a monolitikus rendszernek, akkor a rendszer létrehoz egy helyi adatbázis tranzakciót, amely több adatbázis táblán fog dolgozni. Ha bármelyik lépés hibára fut, a tranzakció visszaállhat. Ezt **ACID**-ként nevezzük.

ACID [8]

- **Atomic** - Lényegében egy atomi tranzakció biztosítja, hogy minden commit sikeresen befejezze az egész műveletet. Vagy, ha a művelet közepén megszakadt a kapcsolat, az adatbázis visszaáll a művelet végrehajtásának megkezdése előtti állapotába. Ez fontos, hogy megakadályozzuk az összeomlásokat és a kieséseket, amelyek olyan eseteket hoznak létre, amikor a tranzakció csak részben, egy ismeretlen állapotig fejeződik be.
- **Consistent** - A következetesség az adatintegritási korlátok fenntartására utal. A következetesség betartatása biztosítja, hogy ha egy adatbázis illegális állapotba kerül (ha az adatintegritási korlátok megsértése történik), akkor a folyamat megszakad, és a változások visszakerülnek korábbi, legális állapotukba.
- **Isolated** - Garantálja, hogy minden tranzakció külön-külön történjen. Egyetlen tranzakciót sem érint más tranzakció. Tehát egy tranzakció nem képes beolvasni egyetlen olyan tranzakció adatait sem, amely még nem fejez?dött be.
- **Durable** - A tartósság biztosítja, hogy az adatbázisban végrehajtott tranzakciók rendszerhibák esetén is végleg fennmaradjanak. Ez biztosítja, hogy az adatbázisban lévő adatokat ne sértsék meg:
  - Szolgáltatási kiesések
  - Összeomlások
  - Egyéb kudarc esetek

A tranzakció során bekövetkezett változásokat véglegesen kell tárolni. A tartósságot az adatbázisok újraindításakor hivatkozott változásnaplók használatával érhetjük el.

Mi lehet a probléma?

Az első felmerülő kérdés, hogy hogyan marad egy tranzakció atomikus? A másik kérdés pedig, hogy ha egy objektumot tranzakcióval írnak le és egyidejűleg egy másik kérés olvassa azt, akkor mit kell visszaadnia az objektumnak, a régi vagy a frissített adatokat?

3.4.2 Lehetséges megoldások

Nagyon fontos nagy hangsúlyt helyezni a fent említett kérdésekre, mert különben nincs mód megmondani, hogy egy tranzakció sikeresen teljesült-e. A probléma megoldására kétfajta mód lehet sikeres:

- 2pc (two-phase commit)
- Saga pattern

A 2pct-t széles körben használják az adatbázis rendszerekben. Bizonyos helyzetekben microserviceknél is lehet használni, viszont nagyon vigyázni kell. A 2pc, ahogy a neve is utal rá, két fázisból áll: egy előkészítő és egy véglegesítő fázis. Egy prepare és egy commit. Az előkészítő fázisban a microserviceket tulajdonképpen felkészítik arra, hogy valamilyen adatváltozás fog történni. Miután ez megtörtént, bekövetkezik a véglegesítő fázis, ami a microserviceket megkéri a tényleges változtatások végrehajtására.

2pc előnyei és hátrányai

Egyik legnagyobb előnye, hogy garantálja, hogy a tranzakció atomi. Emellett, lehetővé teszi az írás és az olvasás szétválasztását. Ez azt jelenti, hogy csak akkor láthatóak a mező változásai, ha a Coordinator elvégezte a módosításokat.
A hátrányai közé tartozik, hogy a 2pc blokkoló. Ez pedig azt jelenti, hogy a módosítani kívánt dokumentumot a protokoll lezárja, még miel?tt a tranzakció befejeződne, ami viszont megakadályozza, hogy az ügyfél új megrendeléseket hajtson végre. Így pedig lehetséges, hogy két tranzakció kölcsönösen lezárja egymást és holtpont jöhet létre.

Saga pattern

A Saga pattern egy másik széles körben használt minta. Amíg a 2pc szinkron, addig a Saga pattern aszinkron és reaktív. Itt a tranzakciót aszinkron helyi tranzakciók teljesítik a microserviceken. Ezek esemény buszon keresztül kommunikálnak egymással. Ha egy mikroszolgáltatás nem fejezi be a helyi tranzakciót, akkor a többi mikroszolgáltatás kompenzációs tranzakciókat hajt végre a módosítások visszavonása érdekében.

Előnyei:

- Támogatja a hosszú életű tranzakciókat.

- A többi microservicet nem blokkolják.
- Az összes helyi tranzakció párhuzamosan zajlik, így egy objektum sincs zárolva.

Hátrányai:

- Nehéz a hibakeresés, különösen több microservicenél.
- Minél összetettebb a rendszer, annál nehezebb fenntartani az eseményüzeneteket.
- Nincs olvasási izolációja.

3.4.3 Naplózás, logging, Központi loggyűjtők összehasonlítása

Mire jó a logolás?

A logolás, naplózás segít abban, mind magunknak, mind a többi fejleszt?nek és egyaránt a felhasználóknak is, hogy nagyobb betekintést kapjunk arra, hogy mi is történik az alkalmazáson belül, ez pedig megkönnyíti a tevékenységek felügyeletét és a problémák felkeresését. Így az alkalmazás biztonságát tudjuk garantálni, csökkenteni tudjuk az összeomlásokat. Ezeket a naplókat viszont nehéz kezelni és nyomon követni, ebben segítségünkre lehetnek a nyílt forráskódú naplókezelők. A továbbiakban ezek közül a három legismertebbet fogom ismertetni és összehasonlítani, ezek pedig a Logstash, Syslog-ng és a Fluentd.

Logstash

A Microserviceknél használatos logoláshoz a legismertebb a Logstash. Népszerűsége köszönhető a széleskörű plugin készletének, és hogy könnyen integrálható más Elastic termékekkel, mint például bemenetek, kodekek szűrők és kimenetek.

Mire használhatjuk?

A logokat össze tudjuk gyűjteni, tudjuk elemezni, és el tudjuk tárolni a későbbi használatra.

Előnyei:

A számos beépülő modul miatt eléggé rugalmas.

Átlátható, átfogó, egyszerű dokumentáció, könnyű konfigurálás. Számos online segítség áll rendelkezésre.

Hátrányai:

Egyik legnagyobb hátránya a teljesítmény és erőforrás-fogyasztás. Sokkal lassabb teljesítményre képes, mint a többi logszállító. Ez a legnagyobb problémát nagy forgalmú telepítéseknél okozhatja.

Syslog-ng

Leginkább akkor érdemes használni, ha szűkös erőforrású feltételeink vannak, viszont mégis összetett feldolgozást szeretnénk végrehajtani. Több naplóüzenet között korrelációkat tud futtatni (feltételezve, hogy ezek mind a pufferben vannak).

Fluentd

A Fluentd a JSON-ban történő naplózás ötletére épült. Minden nyelvhez vannak naplózási könyvtárak, vagyis egyszerűen lehet csatlakoztatni az egyéni alkalmazásokat a naplózási folyamathoz.

Előnyei:

A Fluentd bővítmények Ruby-ban vannak, és nagyon könnyen írhatók. Tehát rengeteg van belőlük, nagyjából minden forrásnak és célnak van beépülő modulja. Ez a folyékony könyvtárakkal párosulva azt jelenti, hogy a Fluentd segítségével szinte bármit bármihez csatlakoztathatunk. A Fluentd ma már CNCF projekt, így a Kubernetes integráció nagyon jó.

Hátrányai:

Egyetlen és legnagyobb hátránya a teljesítménybeli lassúsága.

Graylog vs. ELK [9]

A Graylog és az ELK a két legnépszerűbb nyílt forráskódú megoldás a naplók kezelési eljárásának egyszerűsítésére.

ELK: Az ELK a következő szolgáltatások kombinációjaként épül fel:
Elasticsearch + Logstash + Kibana. Mindhárom nyílt forráskódú.

Elasticsearch: hatékony, jól skálázható keresőmotor, képes nagy mennyiségű adat tárolására.

Logstash: adatok lekérése egy adott helyről, vagy helyre. Számos pluginnel és nagy felhasználói közösséggel rendelkezik.

Kibana: grafikus felhasználói felület, lehetővé teszi az Elasticsearch adatbázis összetett adatának keresését, elemzését és vizualizálását. Könnyen konfigurálható.

Graylog: Rengeteg lehetőséget kínál a naplók elemzésére, kezelésére. Működése nagyon hasonlít az ELK-hoz. [10]

Graylog	ELK
Nem fogad el Syslog fájlokat. Az adatokat közvetlenül kell elküldeni.	Az ELK támogatja az adattípusok többségét, mint például a json stb. Harmadik fél pluginjei felhasználhatók az adatkonvertálásra.
Kizárólag naplókezelésre és naplóelemzésre fejlesztették ki.	Az ELK verem támogatja a naplókezelést és a naplóelemzést más funkcióival együtt. Ez egy többcélú verem.
A Graylog nagyon hatékony a naplók feldolgozásában.	A Logstash, az ELK verem része, amely elvégzi a feldolgozást, nem olyan gyors, mint a Graylog, és problémái vannak, ha nagy forgalmi adatok vannak.
Javában fejlesztették ki, és támogatja a GLEF (Graylog kiterjesztett napló formátumot)	Az ELK stacket Java-ban is fejlesztették, és támogatja a json formátumot.
Támogatja a valós idejű UDP / GLEF naplózást és az intuitív keresést.	Támogatja a teljes szöveges lekérdezés elemzését rugalmas kereséssel.
A Lucene szintaxist használja keresési nyelvként.	Keresési nyelvként a Lucene alapú Query DSL-t használja.
Beépített riasztás érhető el. A riasztások pedig adatfolyamokon alapulnak. Az egyedi riasztási szűrők a Graylog webes felületen konfigurálhatók.	A beépített riasztás nem érhető el, de harmadik féltől származó beépülő modulok, például az X-pack használhatók riasztások küldésére a felhasználók számára.
Néhány olyan szervezet, amely az ELK stacket használja, az Appbrain, a Hotjar, a stockopedia stb.	Néhány olyan szervezet, amely a Graylogot használja, a Netflix, a Cisco, a Verizon és a Linkedin.

A népszerűségük és hatékonyságuk közel hasonló, viszont ami miatt talán a Graylog kiemelkedhet, az az intuitív felhasználói felülete és az engedélyek kezelésének lehetősége.

Request tracing:

A header-ben lévő azonosítóknak a logba elhelyezését valósítja meg. Ennek segítségével lehet látni egy request-hez tartozó összes logot. Vannak kifejezetten erre való rendszerek (pl. ZipKin, Jaeger). Ezekkel a rendszerekkel nyomon tudjuk követni alkalmazásunkat.

[11]

Zipkin:

Egy elosztott nyomkövető rendszer referenciaarchitektúráját képviseli.
4 összetevőből áll:

A GUI-t és az API-t a felhasználók számára a **keresőszolgáltatás** és a **webes felhasználói felület** együttesen biztosítja. A **gyűjtő** ellenőrzi a bejövő adatokat és továbbítja ezeket a tárolóba. És végül a Zipkin legfontosabb összetevője a **tároló motor**ja.

Jaeger:

A Jaeger egy nyomkövető eszköz, ami Go nyelven van implementálva. Nagyon hasonló összetevőkből áll, ugyanúgy rendelkezik gyűjtővel, adattárolóval, lekérdezési API-val és webalapú (React) felhasználói felülettel. Támogatja a Zipkin beviteli portot és az interfészt, így lehetővé téve a Zipkin szolgáltatásainak zökkenőmentes használatát. A Jaeger és a Zipkin közötti különbségnek megemlíthetnénk az architektúrális differenciát és az információk megjelenítésének módját. A Zipkin Java-ban van implementálva, - így jól illeszkedik a vállalati környezethez, azonban más népszerű, magas szintű nyelveket is támogat - a Jaeger pedig Go-ban van megvalósítva. A Jaeger szintúgy a keresésre összpontosít. A Zipkin régebbi, így kevésbé moduláris architektúrát használ, mint a modernebb Jaeger.

Datadog

A Datadog egy olyan kész rendszer, ami a servicek között tud mérni időt trace id-val, jó library támogatással rendelkezik és ki tud jelenteni jogokat. Ha az alkalmazásban hibát, vagy abnormális késést tapasztalunk, akkor a naplózással pontosan kideríthetjük a hiba okát. Egy adott kérelemhez összegyűjthetjük a hozzátartozó naplókat, ezáltal láthatjuk részletesen, hogy hogyan lett kezelve, így gyorsan diagnosztizálhatjuk a hibát. A Datadog automatikusan összegyűjti az adott kérelem összes naplóját, és zökkenőmentesen összekapcsolja őket ugyanazon kérés nyomkövetési adataival.

Skálázhatóság

Hogyan is történik a skálázhatóság a microserviceknél?

Dinamikusan alkalmazkodnunk kell az igényekhez, így ha valamikor megváltoznak a körülmények, manuálisan létre kell hozni az erőforrásokat, amiket csak a későbbiekben fogunk csökkenteni. Ez viszont csökkenti az erőforrás-kihasználás optimalizációját. Erre megoldásként szolgál a Kubernetes automatikus skálázás lehetősége, amivel a létrehozás teljesen automatikusan történik, ezáltal a használt erőforrások és a költségek is lecsökkennek. Leegyszerűsíti az erőforrás-gazdálkodást.

Load-balancing, avagy teheregyensúlyozás

A load-balancingot a skálázhatóság maximalizálása érdekében használják, mivel a load-balancing megoldja a hálózati forgalom elosztását a szolgáltatások között. A Load-Balancer típusú szolgáltatás egy felhő alapú, külső terheléselosztót használ, és csak meghatározott felhőszolgáltatókkal használható. Ezek például az AWS, Azure, OpenStack, CloudStack és Google Compute Engine. A Kubernetes többek között ezt a load-balancingot is lehetővé teszi számunkra. A több microservice több adatbázist jelent, így ezek külön-külön jobban skálázhatóak.

3.4.4 Működés stabilitását karbantartását segítő részek, hibakezelés

Deployment eljárások

Amikor egy alkalmazás új verziót telepít, szüksége van egy telepítési stratégiára, amelyet csak akkor használhat, ha a gyártási forgalmat az új verzióra irányítja, miután sikeresen telepítette és opcionálisan tesztelte. A telepítési stratégiák az alkalmazás futó példányának megváltoztatására vagy frissítésére használt gyakorlatok.
A továbbiakban két telepítési stratégiát említek meg.
A kék-zöld telepítés olyan telepítési stratégia, amely két azonos környezetet használ, egy "kék" (staging) és egy "zöld" (gyártási) környezetet az alkalmazás vagy szolgáltatás különböző verzióival. A minőségbiztosítást és a felhasználói elfogadási tesztet általában az új verziókat vagy változásokat tároló kék környezetben végzik. A felhasználói forgalom átkerül a zöld környezetből a kékbe, miután az új módosításokat tesztelték és elfogadták a kék környezetben. Ezután válthat az új környezetre, ha a telepítés sikeres. Ez a módszer csökkenti a kockázatot és minimalizálja az állásidőt. Két gyártási környezetet (kék és zöld néven) használ megbízható tesztelés, folyamatos leállás nélküli frissítések és azonnali visszaállítás érdekében.

- Zökkenőmentes: a felhasználóknak nem szabad semmilyen leállást tapasztalniuk.
- Biztonságos: alacsony a kudarc esélye.
- Teljesen visszafordítható: visszavonhatjuk a változást káros hatások nélkül.

A kék-zöld módszer alapja az egymás melletti telepítés. Ehhez két különálló, de azonos környezetre van szükségünk. A kék-zöld nagyszerű megoldás, amikor szükségünk van:

- Uptime: amikor nem engedhetjük meg magunknak, hogy leromboljunk egy rendszert annak frissítésére.
- Pontos tesztek: amikor megbízhatóbb és pontosabb tesztekre van szükség.
- Megbízhatóság: amikor javítani akarjuk a telepítések megbízhatóságát.

A kék-zöld telepítések használatához néhány dologra lesz szükségünk:

- Automatizálás: folyamatos kiépítési folyamatokra van szükségünk a kiépítési, telepítési és tesztelési folyamatok automatizálásához.
- Tesztelés: átfogó tesztekre van szükségünk. Bízunk benne, hogy eldönti, mikor állnak készen a kiadások. Folyamatos integrációt kell használnunk, hogy gyorsan elkapjuk a hibákat, és teszteljük az új verziókat, mielőtt életbe lépünk.
- Elkülönítés: két egyforma és különálló környezetre van szükségünk. Ellenkező esetben az egyik környezet hatással lehet a másikra.

A kék-zöld telepítések hátrányai

- Hideg indítás: a felhasználók lassúságot tapasztalhatnak, amikor hirtelen áttérnek az új környezetre. Valószínűleg ezen a ponton bármilyen észlelhetetlen teljesítményprobléma megjelenik.
- Költségek: A költség a kék-zöld telepítések hátránya. A gyártási környezet megismétlése bonyolult és költséges lehet, főleg mikroszolgáltatások esetén.

- Idő: a kék-zöld telepítési folyamat beállítása időt és erőfeszítést igényel. A folyamat bonyolult és nagy felelősséggel tartozik. Előfordulhat, hogy sok iterációt kell elvégeznünk, mielőtt megfelelően működnénk.
- Adatbázisok: Két adatbázisunk van, a kék és a zöld, és mivel ügyelni kell az adatok szinkronban tartására, így az adatbázis-migrációk sokkal nehezebbek. Az adatbázis-sémaváltásoknak előre és hátra kompatibiliseknek kell lenniük.
- Megosztott szolgáltatások: az adatbázisok és más megosztott szolgáltatások szivárogtathatnak információkat kék és zöld között. Óvatosnak kell lennünk itt, különben az egyik környezet közvetve befolyásolhatja a másikat. Ez megsértheti az elkülönítési szabályt, és megzavarhatja a telepítést.

A kék-zöld telepítének mégis inkább a pozitív oldalára helyezném a hangsúlyt. Egyik előnye, hogy egyszerű, gyors, jól érthető és könnyen megvalósítható. A visszagörgetés szintén egyszerű, mert bármilyen probléma esetén egyszerűen visszaviheti a forgalmat a régi környezetbe. A kék-zöld telepítések ezért nem annyira kockázatosak a többi telepítési stratégiához képest.

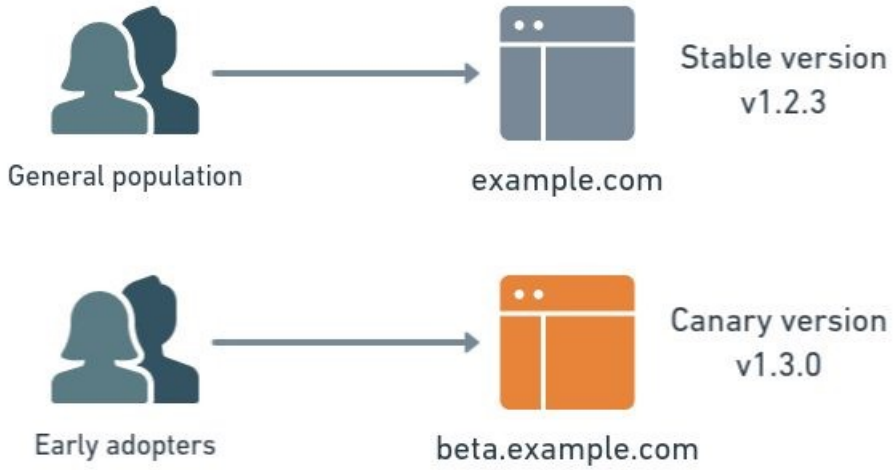
Kanári telepítés [12]

A kanári telepítés olyan telepítési stratégia, amely egy alkalmazást vagy szolgáltatást fokozatosan bocsát ki a felhasználók egy részének. A célkörnyezetben lévő összes infrastruktúra kis fázisokban frissül (pl .: 2%, 25%, 75%, 100%). A kanári kiadás a legkisebb kockázatra hajlamos, az összes többi telepítési stratégiához képest, emiatt az ellenőrzés miatt.

A szoftverfejlesztésben a kanári telepítés a szakaszos kiadások gyakorlása. Először a felhasználók egy kis részére kiadunk egy szoftverfrissítést, így tesztelhetik és visszajelezhetnek. A módosítás elfogadását követően a frissítés a többi felhasználó számára elérhetővé válik.

A kék-zöld telepítésekhez hasonlóan a kanári stratégia leállás nélküli frissítéseket és könnyű visszagörgetést kínál. A kék-zöldtől eltérően a kanári telepítések gördülékenyebbek, és a kudarcoknak korlátozott hatása van.

Egy kanári telepítésben viszont telepítjük a frissítést a rendszereinkbe, és a felhasználókat két csoportra osztjuk. Kis százalékuk a kanári szigetekre megy, míg a többiek a régi változaton maradnak, kontrollként.



Később, miután időt szántunk a kanári verzió értékelésére, dönthetünk úgy, hogy a felhasználók többi részét áttelepítjük a kanári csatornára, vagy mindenkit visszavezetünk a régi verzióra. Mint láthattuk, a kanári telepítések az alkalmazás két verziójának egyidejű futtatását jelentik. A régi verziót "istállónak" és az újat "kanárinak" hívjuk. Kétféle módon telepíthetjük a frissítést: folyamatos telepítések és egymás melletti telepítések.

A kanári telepítések előnyei:

- A / B tesztelés: A kanári felhasználhatjuk A / B tesztelésre. Más szavakkal, két alternatívát mutatunk be a felhasználóknak, és megnézzük, melyik kap jobb vételt. A kanári telepítések lehetővé teszik a szervezetek számára, hogy a gyártásban valódi felhasználókkal és felhasználási esetekkel teszteljenek, és összehasonlítsák a különböző szolgáltatási verziókat.
- Kapacitás teszt: lehetetlen tesztelni egy nagy termelési környezet kapacitását. A kanári telepítésekkel a kapacitástesztek beépítettek. A rendszerünkben felmerülő teljesítményproblémák elkezdődnek, amikor lassan migráljuk a felhasználókat a Kanári-szigetekre.
- Visszajelzés: felbecsülhetetlen hozzájárulást kapunk a valós felhasználóktól.
- Nincs állásidő: a kék-zöld telepítésekhez hasonlóan a kanári telepítés sem hoz leállást.
- Könnyű visszagörgetés: ha valami elromlik, akkor könnyen visszaléphetünk az előző verzióra.

A kanári telepítések hátrányai:

- Költségek: az egymás melletti telepítések költségei magasabbak, mert extra infrastruktúrára van szükségünk. Használja előnyére a felhőt; szükség szerint hozzon létre és távolítson el erőforrásokat a költségek csökkentése érdekében.

- Komplexitás: a kanári telepítések ugyanolyan összetettségűek, mint a kék-zöld telepítések. Sok gyártási gép birtokában van, a felhasználók áttelepítése és az új rendszer felügyelete; ezek bonyolult feladatok. Kerülje el minden áron, hogy kézzel végezze őket. Mindig automatizálja a telepítési folyamatot egy olyan CI / CD platform segítségével, mint a Semaphore.

- Idő: az egészséges kanári telepítési csővezeték felállítása időt és erőfeszítést igényel. Pozitívum, hogy ha rendbe tesszük, akkor gyakoribb és biztonságosabb telepítéseket hajthatunk végre.

- Adatbázisok: egész könyveket írtak arról, hogyan lehet az adatbázis sémáját megváltoztatni. A probléma az, hogy az adatbázisnak a telepítés során egyszerre kell működnie a kanári és a vezérlő verziókkal. Tehát, ha törő sémaváltásaink vannak, akkor bajban vagyunk. A változtatások során meg kell őriznünk a visszamenőleges kompatibilitást, ami újabb komplexitásréteget ad.

Helm configuration [13]

A Kubernetes egyik legjobb tulajdonsága a konfiguráción alapuló jellege, amely lehetővé teszi az erőforrások létrehozását vagy módosítását. Könnyedén beállíthatja és kezelheti az összetevőket a konténerek futtatásától a terheléselosztókig a YAML vagy JSON fájlokon keresztül.
A Helm egy nyílt forráskódú projekt, amelyet a Kubernetes szervezet tart fenn. Ez megkönnyíti a Kubernetes erőforrások csomagolását, szállítását és frissítését egyetlen csomagban. A Helm kliens-szerver architektúrában dolgozik, ahol a Tiller Server egy fürtön belüli kiszolgáló, amely interakcióba lép a Helm klienssel és kapcsolódik a Kubernetes API szerverhez. Feladata a diagramok egyesítése és az ügyfél által kért Kubernetes-erőforrások telepítése.
Míg a Kubernetes-t a rendelkezésre álló erőforrások kezelésére és a telepítések összehangolására használják, a Helm egy olyan eszköz, amely lehetővé teszi a fejlesztők számára, hogy összehangolják a Kubernetes-fürtökhöz elküldött információkat. Helm a Kubernetes tetején fut, és felvázolja az alkalmazás felépítését.

Ansible, Terraform

Mi a Terraform?

A Terraform egy nyílt forráskódú parancssori eszköz, amely bármilyen típusú infrastruktúra biztosítására használható több tucat különböző platformon és szolgáltatáson. A terraform kódot HCL-ben vagy a HashiCorp Config nyelvben írják. Erre itt egy példát láthat. A HCL könnyen megtanulható és könnyen elhárítható. Arra hivatott, hogy egyensúlyt teremtsen az emberbarát és a géppel olvasható kód között.
A Terraform segítségével egyszerűen deklarálja az erőforrásokat és a konfigurálásuk módját, majd a Terraform feltérképezi a függőségeket és mindent felépít Önnek. Egy pillanat alatt bemutatunk egy bemutatót, ahol a Terraform feláll egy szerverrel, majd felhívja az Ansible-t a konfigurálásához.
Az Ansible és a Terraform nem versengő megoldások, mert megoldják az infrastruktúra és a szoftver telepítésének különböző szakaszait. A Terraform lehetővé teszi a rendszer infrastruktúrájának meghatározását és létrehozását, amely magában foglalja

azt a hardvert, amelyen az alkalmazások futtatni fogják. Ezzel szemben az Ansible úgy konfigurálja és telepíti a szoftvert, hogy a lejátszási könyveket a megadott kiszolgálópéldányokon futtatja. Ansible futtatása az erőforrásokon A létrehozásuk után közvetlenül biztosított Terraform lehetővé teszi, hogy az erőforrásokat sokkal gyorsabban felhasználhatóvá tegye a felhasználási esetéhez. Ez megkönnyíti a karbantartást és a hibaelhárítást is, mivel minden telepített kiszolgálóra ugyanazokat a műveleteket alkalmazzák.

Az Ansible egy másik nyílt forráskódú eszköz, amely szoftverek kiépítését, konfiguráció kezelését és alkalmazás telepítését végzi. Egyszerű szavakkal, átvesz egy újonnan létrehozott szerverpéldányt, és egy receptkönyv (ún. Playbook) alapján telepíti a szükséges szoftvert. Végül egy teljesen működőképes kiszolgálónk van, amely kiszámítható és nyomon követhető csővezetékből származik.

Backup készítési eljárás

Milyen gyakran érdemes backup-ot készíteni?

Ez üzleti döntés lehet (pl.: Milyen gyakran és milyen fontosságú tartalmak vannak benne.) Itt lehet pl: napi vagy x időnkénti teljes mentés vagy tranzakció alapú mentése is a db-nek.
A backup-ok nem szükségesek az alkalmazás futtatásához. Ezek megvalósítása időbe és erőforrásokba kerül. És akárcsak az egység tesztjei, a vezetőség talán soha nem fogja látni őket. De ahogy az egységtesztek is nagyon hasznosak a refaktorálás céljából, a biztonsági mentések is nélkülözhetetlenek, ha valami rossz történik a termelési adatokkal. A biztonsági mentési parancsfájlokat "configuration as code" kezeljük.

Istio [14]

Az Istio lehetővé teszi a szolgáltatások csatlakoztatását, biztonságát, ellenőrzését és megfigyelését. Magas szinten az Istio segít csökkenteni ezen telepítések bonyolultságát, és megkönnyíti a fejlesztői csapatok megterhelését. Ez egy teljesen nyílt forráskódú szolgáltatásháló, amely átláthatóan rétegezi a meglévő elosztott alkalmazásokat. Ez egy platform, beleértve az API-kat is, amelyek integrálódnak bármely naplózási platformba, telemetriába vagy házirendbe. Az Istio változatos szolgáltatáskészlete lehetővé teszi, hogy sikeresen és hatékonyan futtasson egy elosztott mikroszolgáltatási architektúrát, és egységes módon biztosítsa, csatlakoztassa és felügyelje a mikroszolgáltatásokat.

Miért érdemes használni az Istio-t?

Az Istio megkönnyíti a telepített szolgáltatások hálózatának létrehozását terheléselosztással, szolgáltatás-szolgáltatás hitelesítéssel, figyelemmel kíséréssel és egyebekkel, kevés vagy egyáltalán nem változtat kódot a szolgáltatási kódban. Automatikus terheléselosztás a HTTP, gRPC, WebSocket és TCP forgalom számára. Automatikus mérőszámok, naplók és nyomkövetések a fürtön belüli összes forgalomra, beleértve a fürt behatolását és kilépését is. Az Istio-t a bővíthetőségre tervezték, és kielégíti a különféle telepítési igényeket.

Alapvető jellemzők

Az Istio számos kulcsfontosságú képességet kínál egységesen a szolgáltatások hálózatában:

Forgalomszervezés

Az Istio egyszerű szabálykonfigurálása és forgalomirányítása lehetővé teszi a szolgáltatások közötti forgalom és API-hívások áramlásának vezérlését. Az Istio leegyszerűsíti a szolgáltatási szintű tulajdonságok, például a megszakítók, az időtúllépések és az újrapróbálkozások konfigurálását, és szellősebbé teszi olyan fontos feladatok beállítását, mint az A / B tesztelés, a kanári bevezetések és a fokozatos, a százalékos forgalom felosztásával történő bevezetések.

Biztonság

Az Istio biztonsági képességei szabadon fejlesztik a biztonságot az alkalmazás szintjén. Az Istio biztosítja a mögöttes biztonságos kommunikációs csatornát, és nagymértékben kezeli a hitelesítést, az engedélyezést és a szolgáltatási kommunikáció titkosítását.

Megfigyelhetőség

Az Istio robusztus nyomkövetési, megfigyelési és naplózási szolgáltatásai mély betekintést nyújtanak a szervízháló telepítésébe. Az Istio felügyeleti funkcióival gyorsan és hatékonyan észlelhetjük és megoldhatjuk a problémákat.

Platform támogatás

Az Istio platformfüggetlen, és különféle környezetekben való futtatásra készült, ideértve a Cloud, on-premise, Kubernetes, Mesos stb. Telepítheti az Istio-t a Kubernetesre vagy a Nomad-ra a Consul segítségével.

Minden microservice mögé egy reverse proxy kerül, ami a jelszavas védelmet nyújtja a microservicenek. Ennek segítségével lehet garantálni, hogy illetéktelen ne férjen hozzá akkor se, ha betört a rendszerbe.

Mi a biztonsági mentés?

A biztonsági mentés a kód vagy az adatok más helyre történő mentésének folyamata, mint ahol a kódot vagy adatokat általában tárolják. Ez a folyamat különböző stratégiák alkalmazásával végezhető, de mindegyiknek ugyanaz a célja - nem veszíteni el az adatokat annak érdekében, hogy a jövőben is hozzáférhessenek hozzájuk.

Miért fontos?

A biztonsági mentés két okból is elvégezhető. Az első az adatok elvesztése egy hack támadás, sérült adatok vagy bármilyen probléma esetén, amelyek lekérdezéseket hajta-

nak végre a termelési kiszolgálón. Ez a biztonsági másolat segít az elveszett vagy sérült adatok helyreállításában.

4. fejezet

Összefoglalás

A szakdolgozatom elkészítésével alaposabban megismerhettem a microservicek világát. Ezzel a technológiával elérhetővé válik, hogy egy sokkal modulárisabb, modernebb rendszert építsünk ki, ami új szintre viszi a programozást. Így betekintést nyertem, hogy egy nagyobb, komplexebb rendszert hogyan kell szétválasztani több kisebb egységre, mikroszolgáltatásra, ezzel javítva rendszerünket. Megtanultam hogyan működik a mikroszolgáltatások közötti kommunikáció, a tranzakció kezelés, illetve hogy milyen megoldások vannak egy tranzakció atomicitásának biztosításához. Továbbá különféle hasznos, a mikroserviceknél használatos technológiával ismerkedtem meg. Az alkalmazás nyomonkövetéséhez és a problémák feltárásához különféle központi loggyűjtőket vizsgáltam meg eredményesség szempontjából. Két elosztott nyomkövető rendszerrel találkoztam, ezek a Zipkin és a Jaeger. Láthattam hogyan történik a microserviceknél a skálázhatóság, milyen módszerek vannak erre, miket kínál a Kubernetes számunkra. Az alkalmazás karbantartásához is különféle rendszerek állnak rendelkezésünkre. Ezek közül megismerkedtem különféle deployment eljárásokkal, nyílt forráskódú figyelő rendsezrekkel, mint az Ansible és aTerraform, és végül az Istio rendszerrel, ami ugyanúgy nagyon hasznos egy microservice rendszer kiépítésében. Ezen technológiák együttes használata segíti a fejlesztők munkáját. Együttesen nyújtanak segítséget, hogy minél tisztább, hatásosabb, jobb legyen az alkalmazásunk.

Irodalomjegyzék

[1] usernet.hu: *CMS basic descriptions*
https://www.usernet.hu/blog/mi-az-a-cms-milyen-elonyei-es-funkcioi-vannak-melyik-cms-t-valaszd

[2] kisshonlapkeszites.hu: *Top 3 best CMS*
https://kisshonlapkeszites.hu/melyik-a-legjobb-cms

[3] medium.com: *Open source CMS pros and cons*
https://medium.com/@inverita/open-source-cms-pros-and-cons-d915a6fdcffa

[4] appliedi.net: *Open source vs. Custom CMS*
https://www.appliedi.net/blog/custom-vs-open-source-content-management-systems-which-is-best/

[5] papellasszabolcs.wordpress.com: *Portal system definitions*
https://papellasszabolcs.wordpress.com/2014/05/10/portlet-definicio/

[6] imperva.com: *CMS security*
https://www.imperva.com/blog/cms-security-tips/

[7] buttercms.com: *Microservices*
https://buttercms.com/blog/transitioning-from-traditional-cms-to-content-as-a-microservice

[8] bmc.com: *ACID*
https://www.bmc.com/blogs/acid-atomic-consistent-isolated-durable/

[9] expertise.jetruby.com: *Log management*
https://expertise.jetruby.com/log-management-graylog-vs-elk-d6e8f0492323

[10] educba.com: *Graylog vs ELK*
https://www.educba.com/graylog-vs-elk/

[11] blog.appopticss.com: *Distributed tracing*
https://blog.appoptics.com/introduction-to-distributed-tracing-with-zipkin-and-j

[12] blog.risingstack.com: *Packing a kubernetes microservices with helm*
https://blog.risingstack.com/packing-a-kubernetes-microservices-with-helm/

[13] istio.io: *What is Istio*
https://istio.io/latest/docs/concepts/what-is-istio/

Adathordozó használati útmutató

A szakdolgozathoz mellékelt lemez az alábbiakat tartalmazza:

- A szakdolgozatot egy szakdolgozat.pdf fájl formájában.
- A Latex forráskódját a szakdolgozatnak.
- Az elkészített alkalmazás forráskódját.