SZAKDOLGOZATI TÉMAVÁZLAT és ÖSSZEFOGLALÁS

(Szövegszerkesztővel töltendő ki! Benyújtandó pdf formátumban 1 példányban)

Hallgató neve:	Földvári Ádám
Neptun kódja	E76Q5V
Munkarend	Levelező
Szak/specializáció	Gazdasági-informatikus / Üzleti adatelemző
Értesítési címe:	1115. Budapest, Hídvég utca 2/a 3.em 2.ajtó
Telefon (mobil)	+36706254624
e-mail címe:	foldadam93@gmail.com
A szakdolgozat címe:	Automatizálás a szoftverfejlesztésben, folyamatos integráció és folyamatos szállítás eszközökkel
A szakdolgozat készítésének helye:	
Vállalat neve:	Emarsys-Technologies Kft.
Vállalat címe:	**1053 Budapest, Kossuth Lajos u. 7**
Külső konzulens	
Neve, beosztása:	Drahos Tamás, csapatvezető
E-mail cím, telefonszám:	drahostamas@gmail.com

A szakdolgozat részletes vázlata:

A szoftverfejlesztés az egyik leggyorsabban fejlődő iparág világszerte. A folyamatosan fejlődő technológiákkal lépést tartani az egyik legjelentősebb kihívás a szektorban dolgozóknak. Újabb és újabb trendek, metodológiák és szakzsargonok ütik fel a fejüket, azonban sokszor évekbe telik, míg egy újonnan megjelenő kifejezésből valóban kidolgozott, a gyakorlatban alkalmazható és akadémiai szinten is kutatható modell lesz.

Lassan négy éve dolgozom szoftverfejlesztőként, ezért az én munkámnak is része a technológiák és trendek követése és alkalmazása különböző feladataim megoldása során. Az elmúlt néhány évben egyre többször találkoztam a „DevOps" kifejezéssel munkahelyeimen, az interneten, konferenciákon és meetupokon. Sőt, mikor legutóbb állást kerestem, arra lettem figyelmes, hogy kifejezetten sok DevOps mérnöki pozíciót hirdetnek a munkáltatók.

A DevOps fogalmát különböző aspektusból közelítik meg az emberek attól függően, hogy milyen szerepet töltenek be az IT szektorban. Egy vezetőnek például inkább a kollaboratív kultúra kialakítását és a fejlesztői és üzemeltetői csapat munkájának összehangolását jelenti. Mivel én szoftver fejlesztői szemszögből közelítem meg a kifejezést, számomra a régóta megszokott agilis módszertanok alkalmazása mellett főleg olyan folyamatok automatizációját jelenti, melyeket másként hosszú időbe telne kivitelezni, ráadásul nagy lenne az emberi hiba lehetősége. Ilyenek például a szoftvertesztelés, az integráció és a szoftver éles környezetbe való beüzemelése. Az automatizálandó feladatokat megfelelő sorrendben kell kivitelezni, ehhez pedig egy speciális rendszert, úgynevezett *pipeline*-t kell létrehozni. Ezt *CI/CD pipeline*-nak is szokás nevezni: a kifejezés a „continuous integration, continuous delivery" szavak rövidítéséből származik, a folyamatos integrációra és folyamatos szállításra utal.

Szakdolgozatom első felében szeretném bemutatni a DevOps fogalmát elméleti oldalról, áttekintve a vonatkozó szakirodalmat. Megvizsgálom a DevOps kapcsolatát a szoftverfejlesztési metodológiákkal és az automatizációval. A második fejezetben arra keresem a választ, hogy milyen kapcsolatban áll a DevOps az agilis szemlélettel, milyen kapcsolódó elméletek és gyakorlatok léteznek az automatizáción túl. A harmadik fejezetben bemutatom a DevOps automatizáció eszköztárát és a megvalósításához szükséges lépéseket, úgymint: verzió követés, tesztautomatizálás, konténerizáció, automatikus élesítés és szállítás, valamint

monitorozás. Áttekintem a rendelkezésre álló felhő szolgáltatásokat is, amelyek segítik a rendszer működését.

A téma elméleti áttekintése után pedig be fogom mutatni, milyen feladatok várnak az informatikus csapatokra, amikor úgy döntenek, hogy DevOps szemléletben kezdenek neki egy projektnek. Az ezzel járó automatizáció különböző szoftverek és eszközök együttes használatának és megfelelő konfigurációjának az eredménye. Ezért a negyedik fejezetben elkészítek négy esettanulmányt, amelyeken keresztül áttekintést adok a legnépszerűbb eszközökről és egyben gyakorlati példát hozok a lehetséges megoldásokra. Az ötödik fejezetben először empirikus módon fogom összehasonlítani az elkészült esettanulmányokat a következő szempontok alapján: fejlesztésre fordított idő, DevOps gyakorlatok érvényesülése, kompatibilitás és limitációk. Végül pedig az elkészült CI/CD pipeline-okat teljesítmény és skálázhatóság szempontjából is meg fogom vizsgálni három kísérlet mérési eredményeinek az összehasonlításával. Ezzel szeretném szemléltetni a DevOps automatizációval járó feladatokat, a feladatok elvégzéséhez szükséges programozói eszköztárat és tudást, valamint megtalálni azokat a szempontokat, amelyeket érdemes figyelembe venni egy ilyen komplex folyamat megtervezése és kivitelezése során.

A dolgozat fejezetei, tartalomjegyzéke:

1. Bevezetés
2. Szoftverfejlesztési módszertanok
3. Az automatizálás lépéseinek és eszközeinek bemutatása
4. Esettanulmányok elkészítése
5. Eredmények
6. Összefoglalás

ÖSSZEFOGLALÁS

Automatizálás a szoftverfejlesztésben, folyamatos integráció és folyamatos szállítás eszközökkel

szakdolgozat címe

Földvári Ádám

Hallgató neve

Levelező / Gazdasági-informatikus / Üzleti adatelemző

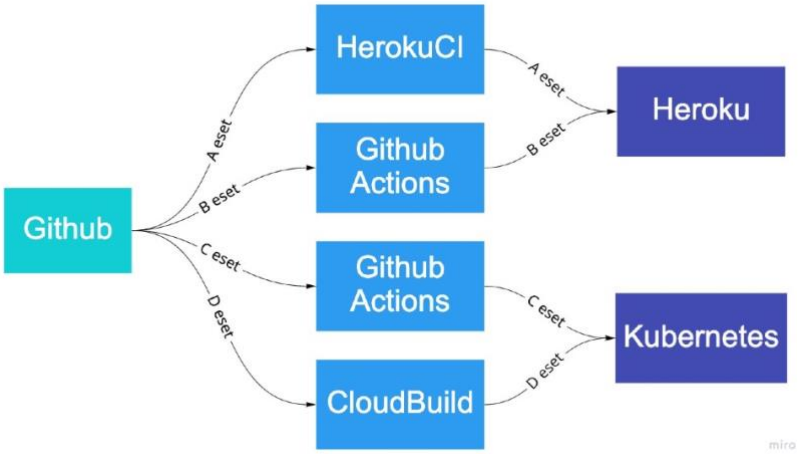
munkarend/ /szak/specializáció

Szakdolgozatom elején visszatekintettünk a szoftverfejlesztés múltjára. Bemutattam az első szoftverfejlesztési módszertanok kialakulását, melyeket még más iparágakból vett át az informatika (pl.: Vízesés modell). Ezek később saját fejlődésnek indultak, hogy jobban tudjanak alkalmazkodni a szoftverprojektekre jellemző gyors változásokhoz és a gyorsan fejlődő technológiákhoz. Az így kialakult iteratív módszertanok jobb adaptív képességgel rendelkeztek. Míg az első ilyen módszertan, a Scrum még nem tartalmazott technológiai előírásokat, inkább csak a csapatok működését, a ceremóniákat és a fejlesztéshez tartozó dokumentumokat határozta meg, a később kialakuló agilis módszertanok, mint például az eXtrém programozás, már technológiai előírásokat is tartalmaznak. A technológiai gyakorlatok kiterjesztésével az üzemeltetésre, illetve az automatizáció, mint elvárás megjelenésével a 2000-es évek elejére kialakult a DevOps szemlélet, mely átfogó megoldást kínált hosszú távú szoftverprojektek kivitelezésére és üzemeltetésére is. Megismerkedtünk a CI/CD pipeline definíciójával, is, ami a DevOps automatizációs gyakorlatokat egy komplex, lineáris rendszerré alakítja.

Ez a rendszer képes minden lépést elvégezni a kódváltoztatások verziókövetőbe való bekerülésétől kezdve egészen a változtatás élesítéséig. Egy ilyen rendszer kialakítása több lépésből áll, és több eszköz összehangolt, integrált működését igényli. Amennyiben ilyen rendszereket szeretnénk létrehozni, fontos az egyes lépések, és a hozzájuk tartozó technológia elméleti hátterének megértése.

A verziókövetés a szoftver kód és annak változatainak menedzselését teszi lehetővé, valamint egyben a CI/CD pipeline indító eleme is. A tesztautomatizáció segít megbizonyosodni arról, hogy a változtatások, amiket a pipeline élesíteni fog, nem okoznak-e hibát a rendszerben. Ez egy fontos minőségbiztosítási eszköz és a napi többszöri élesítés esetén elengedhetetlen. A konténerizáció képes az alkalmazást platform független csomaggá alakítani, annak futtató környezetével együtt. Ezáltal a különböző verziókból készült konténereket el lehet tárolni, és több különböző környezetben újra felhasználva lehet élesíteni. Segítheti a tesztelést is, amennyiben a teszteket is az alkalmazás konténerekben futtatjuk. A konténerizáció elvégzése manuális feladat, a CI/CD pipeline futtatása során pedig a konténerek megépítése, tárolása és élesítéshez való előkészítése történik. Több konténerből álló komplex infrastruktúra kialakítását konténer orkesztrációs platformok teszik lehetővé, amelyek képesek a konténerek teljes életciklusáról gondoskodni, ezáltal megvalósítani a leállási idő nélküli, folyamatos élesítést. A CI/CD pipeline lépéseit és folyamatábráját a 7.ábra mutatja.

Az elméleti összefoglaló után négy esettanulmányt terveztem, melyben ugyanahhoz a minimális web alkalmazáshoz különböző eszközökkel készítettem CI/CD pipeline-okat. Verziókövetőnek minden esetben a Github-ot használtam, élesítési platformnak a Heroku-t és a Kubernetes-t választottam, folyamatos integrációs eszköznek pedig a HerokuCI-t, a Github Actions-t és a Google Cloud Build-et.



Az esettanulmányok kivitelezése során a könnyebb, gyorsabban kivitelezhetőtől a nehezebben kivitelezhetőig haladtam. Egyes esetekben sok beállítást a felhőszolgáltatások webes felületén is meg lehetett csinálni, viszont mindegyik pipeline kialakításához kellett konfigurációs fájlokat is elhelyezni az alkalmazás könyvtárába.

Az eredmények empirikus értékelése során az alábbi megállapításokra jutottam: az A esetben a legtöbb beállítást a webes felületen meg lehetett tenni, a konfigurációs fájlokba csupán néhány sor került, és az alkalmazás konténerizálását sem kellett elvégezni. A többi eset előtt már el kellett végezni az alkalmazás konténerizálását, de láthattuk, hogy ez egy egyszeri tevékenység, tehát elkészítése után a B, C, D esetben is fel lehetett használni. Ezek az esetek abban is eltértek az elsőtől, hogy mindegyik tartalmazott egy, a pipeline lépéseit leíró fájlt, illetve a pipline futások lépéseit és eredményeit egy oldalon meg lehetett tekinteni, így ezek sokkal átláthatóbbak voltak. B esetben a Herokura való Dockeres élesítést megkönnyítette a Heroku parancssoros kezelőfelülete, ami a konténerek építését, tárolását és élesítését is támogatja. Így azonban a teszteket a Github Actions virtuális gépén közvetlenül voltam kénytelen futtatni. A C és D esetben már tudtam a teszteket az alkalmazás konténerében futtatni, viszont ezeknek az eseteknek a komplexitását mégsem ez, hanem a Kubernetes megértése és konfigurálása okozta. Mivel Kubernetesen sokkal több beállítás, konfiguráció lehetséges, mint Herokun, így ezzel a fejlesztési idő is növekszik. D esetet tovább bonyolította, hogy a Cloud Buildben már nem tudtam közvetlenül a virtuális gépen futtatni parancsokat, csak Docker konténerekben, illetve a lépéseket leíró fájl szintaktikája is bonyolultabb volt.

Láthattuk, hogy azokban az esetekben, ahol az élesítés Herokura történik (A és B eset), az élesítéshez-szállításhoz tartozó DevOps gyakorlatok közül nem mindegyik teljesül. A kód alapú infrastruktúra nem megvalósítható, mivel a Heroku csak kész csomagokat kínál az infrastruktúra beállítására. B esetben a dockerizációnak köszönhetően a futtató környezet beállítása már lehetséges (míg A esetben ez sem teljesült). Az egygombos élesítésre viszont csak az A eset kínál kész megoldást. A többi esetben újabb automatizációt szükséges hozzá létrehozni. A Kubernetest használó esetekben (C és D) a kód alapú infrastruktúra teljesül, ugyanakkor több mindent kell konfigurálni, mint például a leállás nélküli élesítést. (A Heroku esetében ez is automatikusan működik.) Ezekből levonható az a következtetés, hogy a Heroku kevésbé illeszkedik a DevOps szemlélethez, mint a Kubernetes vagy más konténer orkesztrációs platformok.

A HerokuCI csak Herokura történő élesítést támogat, ezzel szemben a Github Actions és a Cloud Build kompatibilis egyéb platformokkal is. Mivel azonban a Cloud Build a Google Felhő Platform része, elsősorban Kubernetesre és más Google Felhő alapú folyamatok automatizálására használható a legkönnyebben.

Az eredmények empirikus értékelése után három kísérletet végeztem, melyben az esettanulmányok teljesítményét vizsgáltam. A kísérletek során az elkészült pipeline-okat egyszerre futtattam, egymás után többször, és mértem a futási idejüket. A három kísérlet között az alkalmazás komplexitását növeltem, hogy így kiderüljön, milyen teljesítményt nyújtanának a megoldások egy valódi, nem minimális web alkalmazás esetében. Az A esetben pontos méréseket sajnos nem lehetett végezni, de az látható volt, hogy a teljesítménye jóval elmarad a többihez képest. Az eredményeket varianciaanalízissel vizsgáltam, SPSS-el. Az első kísérletből kiderül, hogy a Kubernetes leállás nélküli élesítési stratégiája lassabb a Herokuénál. A második kísérlet megmutatta, hogy a Cloud Build konténerkezelési teljesítménye jobb, mint a Github Actions-é. A harmadik kísérlet pedig rávilágított, hogy amennyiben az alkalmazás és a tesztek erőforrás igényesek, a Github Actions által biztosított virtuális gépek jobb teljesítményt nyújtanak.

A kísérletek kiértékelése után egy árkalkulációt is elvégeztem, ami a következő eredményt hozta. Az A eset volt az egyetlen, amely még egy alkalmazás esetében is nagyjából tíz dolláros költséggel jár. A többi esetben akkor is ingyenes lenne a CI/CD pipeline fenntartása, ha naponta többször is futnának, viszont több alkalmazás esetén a legolcsóbb megoldás a Cloud Build használatával érhető el.

Jelenleg a DevOps nagy népszerűségnek örvend, a legtöbb piacvezető szoftverfejlesztő cég alkalmazza a gyakorlatait. Láthattuk azonban, hogy készíthető olyan CI/CD pipeline is, amely nem valósítja meg az összes gyakorlatot. Fordítva is igaz az, hogy ha sikerül is kialakítanunk a DevOps gyakorlatokat megvalósító pipeline-t, még nem jelenti azt, hogy sikeresen adaptáltuk a DevOps szemléletet. Ehhez szükségesek egyéb automatizációk is, mint például a monitorozás, amelynek bemutatására a szakdolgozatban nem tértem ki. Szintén elengedhetetlen a kollaboratív kultúra kialakítása a cégen belül, ez viszont főleg a menedzsment feladata, így fejlesztési szempontból ennek nincs nagy jelentősége.

A szakdolgozatban bemutatott esettanulmányok elsősorban webalkalmazások élesítésére szolgálnak, azonban a szoftverfejlesztés egyéb területein is hasznos lehet az élesítési folyamatok automatizálása, például a mobilapplikáció fejlesztésben. Ez a terület újabb kihívások elé állítja az informatikusokat, mivel még kevésbé kiforrottak az eljárások, ezért ez egy érdekes további kutatási terület lehet a jövőben. Létezik már olyan cég, ami kifejezetten ezt a területet célozza, ez a Bitrise. Szintén érdekes kutatási lehetőséget teremt a mostanában megjelenő DevSecOps kifejezés, ami a fejlesztési és üzemeltetési szempontok mellett a kiberbiztonsági szempontokat is figyelembe veszi.

Az eredményeket tekintve, saját hobbi projektjeimhez a C esetben használt technológiákat használnám, illetve fogom használni a jövőben, főleg annak rugalmassága és egyszerűsége miatt. Komplexebb, nagyobb projektek esetében viszont inkább a D esetben használt eszközöket választanám, elsősorban az alacsony ár miatt. Illetve amennyiben például egy cég vagy projekt infrastruktúrája részben, vagy teljes egészében a Google Felhőn alapszik, a Cloud Build jól fog illeszkedik a projekthez, a fejlesztők így mindent ugyanazon a platformon tudnak elvégezni.