Miskolci Egyetem
Informatikai Intézet
Általános Informatika Tanszék

Deployment and Configuration of a Software Development Infrastructure

Komplex feladat

Készítette:
Tóth Ádám
D3F1KK

Konzulens:
Dr. Tóth ZSOLT Tamás JUDIT

2018.

Tartalomjegyzék

1. fejezet

Bevezetés

A szoftverfejlesztés a mai világunk egyik alapköve. Az informatika világa és a mi hétköznapijaink sem lennének képesek fejlődni a szorgos programozók munkája nélkül. Azonban a szoftver fejlesztés nem egy egyszerű dolog, főleg ha egy komplex programozási feladatra van szükség. Ilyenkor szinte biztosan nem egy ember fogja elkészíteni ezeket a programokat, így a munka érdemi része kevesebb feladatot ró a fejlesztőre viszont számos egyéb feladattal kell szembe néznie. Ez a téma az informatika nem annyira szembetűnő oldaláról mutatja be. A mai világban gyakran hajlamosak vagyunk megfeledkezni arról, hogy az a kép amit elkattintunk a telefonunkkal hogyan is kerül fel a felhőbe és onnan a asztali számítógépünkre vagy akár a barátainkhoz valamilyen közösségi média platform használatának segítségével. Nem egy program megszületését dokumentáltam hanem sokkal inkább azt, hogy egy komplex programhoz milyen háttér eszközök kellenek hogy elkészüljön.

Ezek a feladatok (ha jól elő vannak készítve) nagyon könnyen vehető akadályok. A csapatban való munka fontos részét alkotja a modern szoftverfejlesztésnek. Ha egy mai informatikai céget meglátogatunk, azt láthatjuk hogy a programozók nem elszeparáltan hanem egy közös térben dolgoznak. Például egyre népszerűbb a scrum módszer bevezetése az irodákban és minden reggel ennek megfelelően egy közös megbeszéléssel kezdőik.

A csapatmunkát komoly támogatási háttérrel szükséges ellátni. Gondok itt arra, hogy lehetőséget kell adni minden fejlesztő számára, hogy a fejlesztés állapotát nyomon tudják követni és a mások munkáját is könnyen feltudják használni vagy esetenként módosítani tudják azt. Ezeket a lehetőségeket nem egy egyszerű feladat megteremteni legyen szó tíz vagy akár százfős irodáról.

Persze ezen felül ha többen dolgoznak egy kódon akkor a csapatmunka gördülékenységét segíti elő bizonyos szabályok lefektetése. Az ugyan olyan formátumú kód használata elengedhetetlen komplex programok megírásakor.

A programozókra komoly munka és rengeteg egyeztetés várna, ha nem léteznének úgynevezett verzió követő rendszerek. Nem is lehetne olyan szoftver gyártó céget mondani mely nem használná valamely formában a Git-et.

Mindezeken felül még persze ott a dokumentáció is, hogy a kód később is vagy egy új kolléga számára is egyértelmű legyen.

“A Continuous Integration – azaz a folyamatos integráció

maga a CI fontossága

1.1. Motiváció

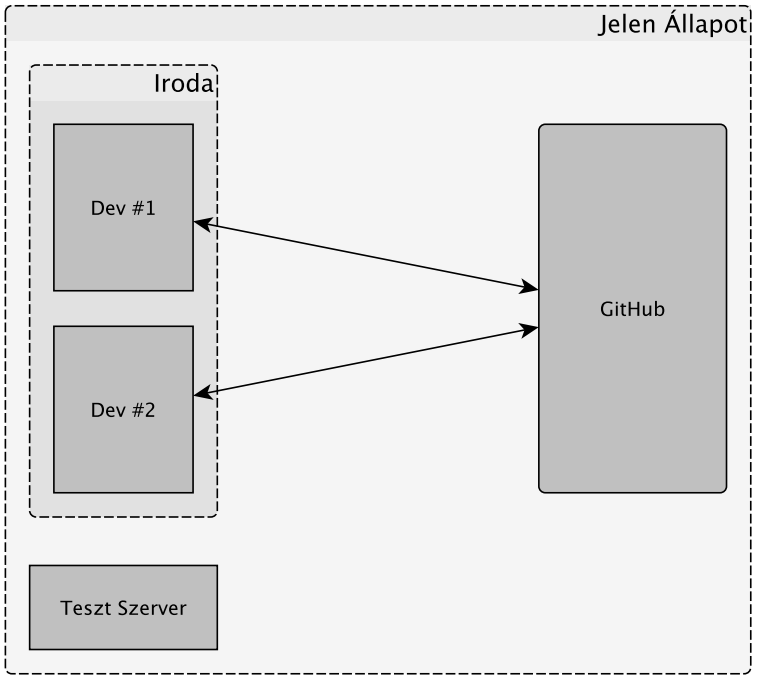
Mostanra az informatikusnak tanulók között is sokkal izgalmasabb egy mobilfejlesztés vagy egy webfejlesztés. Az látszik az van előtérben, végül is az ő munkájuk az ami igazán eljut a végfelhasználókhoz. Ez teljesen megérthető hiszen a színes izgő-mozgó képek tengerével hogyan is vehetné fel a versenyt egy még a Mátrix c. kultuszfilmben is csak keveset látott terminál ablak fekete-zöld karakteri. Pedig ez a világ az ami lehetővé teszi az eszközök közötti összeköttetést és a csillogó képekkel teli megjelenést. Az ember azt gondolná hogy a programozók pontosan ismerik ezt a világot, pedig vannak akik ennél is mélyebbre mennek. Ezek az emberek a rendszerüzemeltetők és az ők titkolt és rejtett világuk. Pontosan ez tetszett meg nekem és ezért is szeretnék hasonló témában elkészíteni a feladatom.

Természetesen ez talán még nem elég ahhoz hogy egy ilyen témába mélyen beleássa magát az ember. Már régebben is az foglalkoztatott hogy bizonyos játékoknak hogyan tudnék saját szervert csinálni hogy a barátainkkal igazán szabadon élvezhessük a játékok adta élményeket.

Szerencsére nekem az egyetem is lehetőséget adott ahhoz hogy ezzel a témával mélyebben is tudjak foglalkozni. Dr. Tóth Zsolt konzulensem ajánlotta fel azt hogy ebben segítséget ad és természetesen egy célt is melyről komplexet és később majd a szakdolgozatomat is megírhatom.

1.2. Célkitűzés

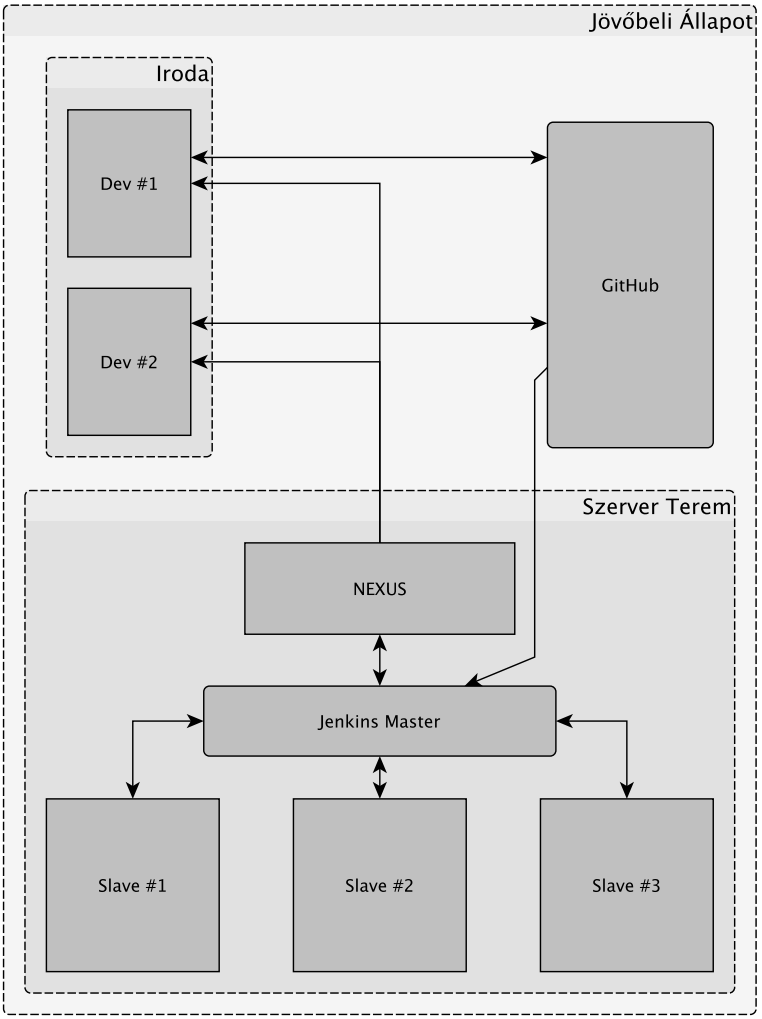
A célom azt volt, hogy a Miskolci Egyetem Általános Informatika Tanszékén az ILONA rendszeren dolgozó fejlesztőknek a munkáját segítsem. A fejlesztők jelenleg az építést és a tesztelést is manuálisan valósítják meg, ahogy ez a 1.1 ábrán is látható. Ez a feladatom kiinduló állapota.



1.1. ábra. Jelen Állapot

Az új rendszernek a jelenlegi állapotnak több sajátosságát meg szerettem volna tartani és ezeken felül a már meglévő szerverekkel megvalósítani a rendszer működését. Fontosnak tartottam hogy a fejlesztők kizárólag a feladataik csökkenésével szembesüljenek és ne keljen új programokat, folyamatokat megtanulniuk. Kvázi ne látszódjon a háttérbe folyó változás és a jelenlegi rendszer szolgáltatás kiesésével se járjon az új rendszer bevezetése. Mivel az ILONA fejlesztése teljesen önerőből, támogatás nélkül valósul meg, ezért törekedtem arra hogy csak ingyenes, "opensource" alkalmazásokat használjak a feladathoz. ——— A jelenlegi rendszer alapja az egyik legnépszerűbb ingyenes internetes verziókövető rendszer a GitHub. A GitHub-ot feltétlen meg akartam tartani az új rendszerben mivel egy elterjedten használt és kedvelt környezetnek tartom és ami még mellette szól az az ingyenessége. Ez a teszt szerver Maven segítségével építi és teszteli a megírt kódokat, ezért a következő amit a meglévő rendszerből át akartam vinni az újba az a Maven. A jelenlegi rendszer nem hatékony a munkavégzés során, mivel számos munkaidőt emészt fel a folyamat amely nem a fejlesztéssel kapcsolatos. ———

A fejlesztőknek az új rendszerben lényegében csak a GitHubbal és a kész JAR fájlokat tartalmazó repository szerverrel kell dolgozniuk a többi folyamatot automatikusan látná el a rendszer. A cél tehát egy olyan Automatikus Tesztelési Környezet megvalósítása, amely képes a GitHubról letölteni a kódokat akkor ha a kódban változás történik. Ezen felűl ha a tesztek és a buildelési folyamat is sikeresen végbement az elkészült fájlokat könnyen és szervezetten elérhetővé tennie egy repository szerveren. Ez a tervezett folyamat látható a 1.2 ábrán.



1.2. ábra. Jövő Állapot

2. fejezet

Technológiák

2.1. Docker

Kezdjük azzal, hogy meghatározzuk, mi a Docker. A Docker honlapja jelenleg a következőképpen határozza meg a Dockert (Szabad fordításban):

"A Docker a világ vezető szoftveres tárolóplatformja, a fejlesztők a Docker-t használják az "én gépemen működik" problémák kiküszöbölése érdekében, amikor a munkatársakkal együtt dolgoznak egy kódon. Az üzemeltetők a Dockert használják az alkalmazások futtatására és kezelésére egy szerveren elválasztva egymástól a legjobb kihasználtságért, A vállalatok a Docker-t használják agilis szoftverszállítási folyamatokhoz, hogy gyorsabbá és biztonságosabbá tegyék a Linux és a Windows Server alkalmazásaikat." [2]

Ez egy elég merész megnyitó nyilatkozat, de ha megnézzük a Docker vezérigazgatója, Ben Golub által bemutatott számokat a 2017-es DockerCon megnyitása során, amelyek a következők:

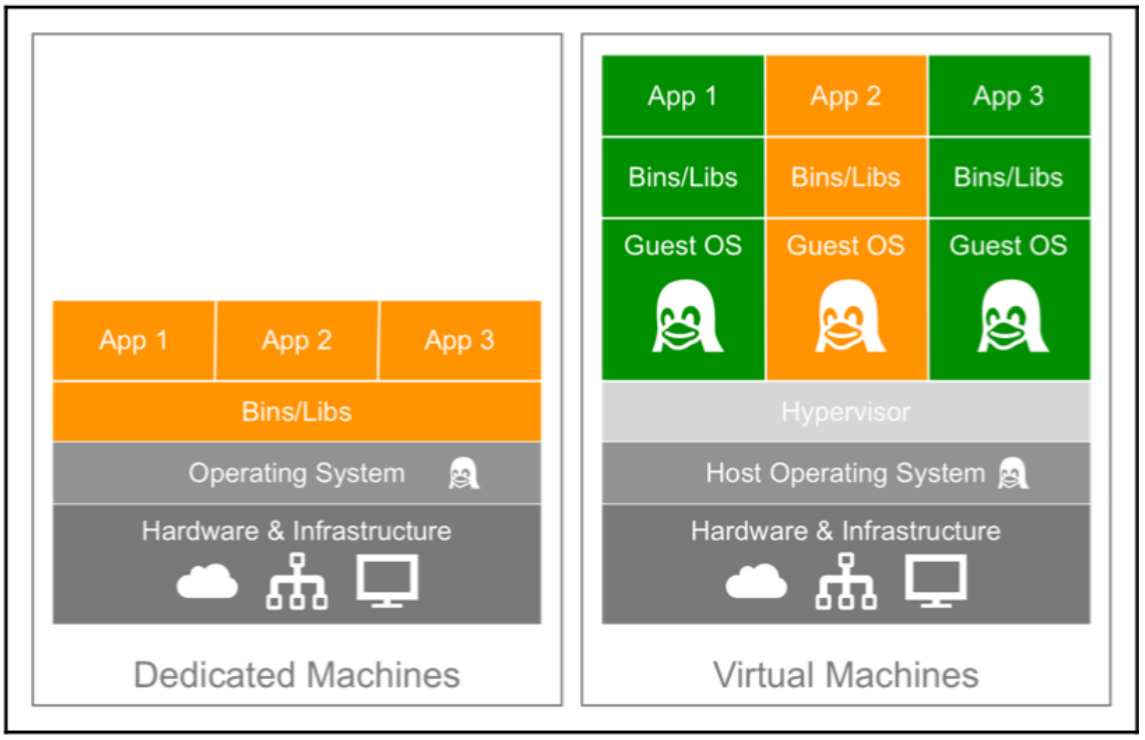
- 14 millió Docker fogadó
- 900,000 Docker Alkalmazások
- 7700 százalékos növekedés a Docker "Image"-k letöltésében
- 3300 projektpartner

Egy olyan technológia esetében, amely csak három éves, mindenki egyet kell hogy értsen ez meglehetősen lenyűgöző teljesítmény. [1]

2.1.1. Különbségek dedikált hostgépek, vitruális gépek és a Docker közt

A Docker egy konténerkezelő rendszer, amely hasonlít a Linux Containers (LXC)-re de könnyebb és univerzálisabb módon azt. Lehetővé teszi a virtuális környezetben lévő képek létrehozását akár laptopon, ugyanis a futtatásuk nem igényel nagy erőforrást. Ebben a környezetben a saját gépén helyi szinten futó konténeren végrehajtott műveletek vagy parancsok ugyanazok amelyeket megszokhattunk egy valódi gépen. Ez segít abban, hogy ne kelljen másképp csinálni a dolgokat, ha egy olyan fejlesztői környezetből, mint amilyen a helyi gépen található, szerver szintű környezetébe lépünk.

Nézzük meg a Docker konténerek és egy tipikus virtuális gépi környezet közötti különbségeket. Az alábbi ábra bemutatja a dedikált, csupasz fémes kiszolgáló és a virtuális gépeken futó kiszolgáló közötti különbséget:



2.1. ábra. Egy Host gép és egy Virtuális gép közti különbség [4]

2.1.2. Maven

Az Apache Maven rövid nevén csak Maven-t szoftverfejlesztés során használjuk a projektek menedzselésére Az Apache Ant utódja, sok tekintetben hasonlít a két szoftver, azonban az Ant lassabb és elavultabb mint a Maven. A Mavennel az építési- és tesztelési folyamatok automatizálhatóak, így a Jenkinsel tökéletesen együtt tud dolgozni. A Maven egy új fogalmat is bevezet, ez az úgynevezett Projekt Objektummodell (angolul: Project Object Modell) röviden a POM. A POM egy XML fájl amely egy építésre kész projektet és annak függőségeit írja le. Ezzel a pom.xml fájllal adjuk meg a Mavennek hogy mit is csináljon a kóddal. Előre megadott célokat tartalmaz mint a kód fordítása és csomagolása, azonban itt van lehetősége a fejlesztőknek saját célokat és teszteket létrehozni.

A Mavent 2002-ben Jason van Zyl készítette el. A projektet az Apache Software Foundation fejleszti, korábban a cégnél a Jakarta Projekt részeként működött. Jelenleg a 3.3.9-es a legfrissebb verziója.

—IDÉZERT— A Maven hálózatképes, tehát szükség esetén dinamikusan is le tud tölteni komponenseket. Repository névvel illetik a különböző hosztok fájlrendszereinek azon mappáit, ahol a letölthető komponensek találhatók. A Maven nem csak a repository-kból való letöltést támogatja, hanem a készült szoftvercsomag feltöltését is. Ezzel az automa-

tizálható le- és feltöltési mechanizmussal a Maven de facto szabványt próbál teremteni, de elég lassan fogadja el a Java közösség.

A Maven plugin alapú architektúrája lehetővé teszi tetszőleges parancssorból vezérelhető alkalmazás használatát. Ez elméletileg lehetővé teszi tetszőleges programnyelvekhez való pluginek készítését, de a gyakorlatban minimális mennyiségű nem javás plugin készült. —IDÉZERT—

2.1.3. Java, JUnit

Java

—IDÉZERT— A Java általános célú, objektumorientált programozási nyelv, amelyet a Sun Microsystems fejlesztett a '90-es évek elejétől kezdve egészen 2009-ig, amikor a céget felvásárolta az Oracle. 2011-ben a Java 1.7-es verzióját az új tulajdonos gondozásában adták ki.

A Java alkalmazásokat jellemzően bájtkód formátumra alakítják, de közvetlenül natív (gépi) kód is készíthető Java forráskódból. A bájtkód futtatása a Java virtuális géppel történik, ami vagy interpretálja a bájtkódot, vagy natív gépi kódot készít belőle, és azt futtatja az adott operációs rendszeren. Létezik közvetlenül Java bájtkódot futtató hardver is, az úgynevezett Java processzor.

A Java nyelv a szintaxisát főleg a C és a C++ nyelvektől örökölte, viszont sokkal egyszerűbb objektummodellel rendelkezik, mint a C++. A JavaScript szintaxisa és neve hasonló ugyan a Java-éhoz, de a két nyelv nem áll olyan szoros rokonságban, mint azt ezekből a hasonlóságokból gondolhatnánk.

Bár a nyelv neve kezdetben Oak (tölgyfa) volt, (James Gosling, a nyelv atyja nevezte így az irodája előtt növő tölgyfáról), később kiderült, hogy ilyen elnevezésű nyelv már létezik, ezért végül Java néven vált ismertté. A Java szó a Oracle védjegye. Ennélfogva engedélye nélkül nem használható mások által kifejlesztett termékek megjelölésére; még például Java-szerű ... stb. összetételben sem, mert ez a védjegyjogosult jogaiba ütközik. —IDÉZERT—

JUnit

—IDÉZERT— JUnit egy egységteszt keretrendszer Java programozási nyelvhez. A teszt vezérelt fejlesztés (TDD) szabályai szerint ez annyit tesz, hogy a kód írásával párhuzamosan fejlesztjük a kódot tesztelő osztályokat is (ezek az egység tesztek). Ezeken egységtesztek karbantartására, csoportos futtatására szolgál ez a keretrendszer. A JUnit teszteket gyakran a build folyamat részeként szokták beépíteni. Pl. napi build-ek esetén ezek a tesztek is lefutnak. A release akkor hibátlan, ha az összes teszt hibátlanul lefut.

A JUnit a egységteszt keretrendszerek családjába tartozik, melyet összességében xUnit-nak hívunk, amely eredeztethető a SUnitból.

JUnit keretrendszer fizikailag egy JAR fájlba van csomagolva. A keretrendszer osztályai következő csomag alatt található:

JUnit 3.8-as ill. korábbi verzióiban a junit.framework alatt találhatók JUnit 4-es ill. későbbi verzióiban org.junit alatt találhatók —IDÉZERT—

Az ILONA rendszer a fejlesztők Java nyelven fejlesztik, ezért az új rendszernek képesnek kellett lennie a Java használatára. Emellett maga a Jenikins CI is egy Java nyelven íródott webalkalmazás ami Java VM-en fut. Így a Safranek virtuális gépre egy legfrissebb Java

2.1.4. MySQL

A MySQL egy többfelhasználós, többszálú, SQL-alapú relációs adatbázis-kezelő szerver. A MySQL az egyik legelterjedtebb adatbázis-kezelő, aminek egyik oka lehet, hogy a teljesen nyílt forráskódú LAMP (Linux–Apache–MySQL–PHP) összeállítás részeként költséghatékony és egyszerűen beállítható megoldást ad dinamikus webhelyek szolgáltatására.

A tesztrendszer számára egy elengedhetetlen kiegészítőt nyújt olyan tesztelési feladatoknál mikor a program adatbázist használ. Teszteléskor a teszt folyamat feltölti az adatbázist a teszteléshez szükséges adatokkal majd a teszt lefutása (akár sikeres, akár sikertelen) után üríti az adatbázist.

2.1.5. Virtualizáció, Hálózat

2.1.6. Jenkins

A Jenkins egy nyílt forráskódú, Java nyelven írott eszköz amely folyamatos integrációs szolgáltatást nyújt szoftverfejlesztéshez. Az Oracle Hudson folyamatos integrációs eszközének eredeti fejlesztő csapata vált ki a Hudson fejlesztéséből és valósították meg a Jenkinst. Mivel az eredeti fejlesztőcsapat jelenleg is a Jenkinst fejleszti ezért érdemes (ha a kettő közül kell választanunk) a Jenkinst választani. A Jenkins mellet szóló érv még hogy több plugin érhető el hozzá mint a Hudsonhoz, így több feladatot is egyszerűbben valósítható meg. Ezt az eszköz közvetlenül telepíthetjük a rendszerünkre, viszont választhatjuk hogy egy szervlet konténerben fusson, mint pl. az Apache Tomcat. Támogat számos SCM eszközt mint például a Git-et amely az ILONA projekt számára elengedhetetlen. Ezek mellett az Apache Ant és Apache Maven parancait is végre tudja hajtani, így a Maven továbbra is lehetséges használni az új rendszerben. A Jenkins elsődleges fejlesztője Kohsuke Kawaguchi. A Jenkinst MIT licenc alatt adják ki és szabad szoftver.

"A Continuous Integration – azaz a folyamatos integráció – egy szoftver fejlesztési módszer melyben a fejlesztőcsapat tagjai az általuk írt kódot legalább napi rendszerességgel integrálják a korábbi fejlesztések közé, ez napi többszöri integrálást jelent. Minden új kód integrálása során automatizált tesztek ellenőrzik, hogy a rendszerbe való illesztés során okozott-e valamilyen hibát az új kódrészlet és ennek eredményeként a lehető leghamarabb visszajelzést ad az integráció eredményéről" [3]

Maga a Jenkins CI látja egy a különböző elemek közti kapcsolatot, a GutHub hookjaitól egészen a Maven építési és tesztelési folyamat visszajelzéséig.

2.1.7. Nexus

—IDE MÉG NEM TALÁLTAM SEMMIT, DE MAJD TALÁLOK KI—

3. fejezet

Tervezés

4. fejezet

Implementáció

Install ubuntu server

Install docker https://docs.docker.com/install/linux/docker-ce/ubuntu/

Install docker registry Install docker registry web interface https://hub.docker.com/r/hyper/docker-registry-web/

Install Jekins docker https://hub.docker.com/r/jenkins/jenkins/ Install Jenkins docker plugin Docker Slaves plugin

5. fejezet

Tesztelés

6. fejezet

Összefoglalás

Irodalomjegyzék

[1] Docker-con adatok. https://www.slideshare.net/Docker/dockercon-2017-general-session-day-1-ben-golub. 2017.

[2] Docker weboldala. https://www.docker.com/what-docker.

[3] Martin Fowler and Matthew Foemmel. Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, 122:14, 2006.

[4] Scott Gallagher. *Mastering Docker*. Packt Publishing Ltd, 2015.