VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

AUTOMATIZOVANÁ PODPORA TESTOVÁNÍ A VYDÁVÁNÍ SERVEROVÝCH APLIKACÍ

AUTOMATIZED TESTING AND DEPLOYMENT SUPPORT FOR SERVER APPLICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. MARTIN MAGA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. ZBYNĚK KŘIVKA, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/20138/2016/xmagam00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Maga Martin, Bc.**
Obor: Inteligentní systémy
Téma: **Automatizovaná podpora testování a vydávání serverových aplikací**
Automatized Testing and Deployment Support for Server Application
Kategorie: Softwarové inženýrství

Pokyny:

1. Nastudujte metodiky vydávání serverových aplikací (blue/green deployment, rolling release), continuous delivery, koncept DevOps.
2. Analyzujte požadavky na testovatelnost aplikace v průběhu vývoje a před vydáním do produkčního prostředí a způsoby automatizace různých úrovní testovacích fází (integrační, regresní, ...).
3. Připravte a konzultujte návrh průběhu životního cyklu aplikace/funkcionality od úvodního zadání požadavku po vydání do produkčního prostředí.
4. Implementujte prototyp vámi navrženého způsobu automatizace procesu vývoje a testování s vhodně zvolenými nástroji.
5. Ověřte prototypovou implementaci pomocí testů a ověřte chování systému pod zátěží.

Literatura:

- HUMBLE, Jez a David FARLEY. Continuous delivery: reliable software releases through build, test, and deployment automation. Upper Saddle River, NJ: Addison-Wesley, 2010, 463 p.
- dle pokynů vedoucího a konzultanta

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Křivka Zbyněk, Ing., Ph.D.**, UIFS FIT VUT
Konzultant: Wolf Dominik, Ing., AVG
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Úlohou tejto diplomovej práce je podľa konkrétnych požiadaviek zadávateľa spoločnosti AVG vytvoriť systém pre automatizovanú podporu testovania a vydávania serverových aplikácií. Hlavným cieľom je vytvoriť systém vydávania a testovania, ktorý automaticky alebo manuálne otestuje v cloudovom prostredí aplikáciu s možnosťou vydania aplikácie do produkcie bez výpadkov so záverečným otestovaním, pričom monitoruje priebeh. Systém automatizovaného testovania a vydávania bol rozdelený na časť uživateľského rozhrania, prostredníctvom, ktorého môžeme pridávať nové aplikácie a spúšťať testovanie, vydávať a monitorovať priebeh. Druhá časť je reprezentovaná restovou službou, ktorá spracováva požiadavky na testovanie a vydávanie aplikácie. Systém bol odskúšaný na vzorových serverových aplikáciách v cloudovom prostredí Amazon Web Services. Práca popisuje obecné spôsoby testovania aplikácií naprieč rôznými oblasťami. V práci je rovnako možné najsť obecnú architektúru systému spolu s prehľadom diagramov, ktoré ukazujú prípady užitia. Na záver je uvedený spôsob testovania vzorových aplikácií spolu s ich výsledkami.

Abstract

Task of this master thesis is to create system for support of automated testing and deploying of server applications according to requirements defined by company AVG. The main target is create system for deploying and testing that automatically or manually test the server application in the cloud environment with ability of final deployment to the production environment with overall progress monitoring. Automated testing and deploying system has been split to the two parts. The first part is user interface that allows adding new applications, testing applications and deploying applications to its production environment. The second part represents the REST service which process testing and deploying tasks and store progress to database. System was tested with sample server's applications in Amazon Web Services cloud environment. Thesis describes general testing principles cross multiple areas. Also it contains general architecture withing diagrams, which shows use cases. At the end of thesis is described testing of samples application together with results.

Kľúčové slová

Java Standard Edition 8, SpringBoot, Automatizácia, Twitter Bootstrap, Restová služba, Testovanie, MySQL, Amazon Web Services, Symfony, Consul, Bamboo, Blue/green nasadzovanie

Keywords

Java Standard Edition 8, SpringBoot, Automation, Twitter Bootstrap, Rest service, Testing, MySQL, Amazon Web Services, Symfony, Consul, Bamboo, Blue/green deployment

Citácia

MAGA, Martin. *Automatizovaná podpora testování a vydávání serverových aplikací.* Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Zbyněk Křivka, Ph.D.

Automatizovaná podpora testování a vydávání serverových aplikací

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána doktora Zbyňka Křivky. Technické informácie a vedenie mi poskytol konzultant z firmy AVG Dominik Wolf. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Martin Maga
23. mája 2017

Poďakovanie

Veľmi rád by som poďakoval za vedenie mojej diplomovej práce pánovi Zbyněkovi Křivkovi a firme AVG, s ktorou som komunikoval prostredníctvom externého konzultanta Dominika Wolfa, ktorému tiež patrí moja vďaka.

Obsah

1 Úvod **3**

2 Koncept DevOps **4**
2.1 Blue/green vydávanie 6
2.2 Rolling release 7
2.3 Continuous delivery 8

3 Testovateľnosť aplikácií **9**
3.1 Manuálne testovanie 10
3.2 Automatizácia testovania 10
3.3 Jednotkové testy 11
3.3.1 Obmedzenie jednotkových testov 12
3.4 Integračné testovanie 12
3.4.1 Prístup zdola-nahor 12
3.4.2 Prístup zhora-nadol 12
3.5 Regresné testovanie 12
3.6 Systémové testovanie 12
3.7 Akceptačné testovanie 13
3.8 Výkonnostné testovanie 13
3.8.1 Záťažové testy 14
3.8.2 Stress testy 14
3.9 Testovanie použiteľnosti 14
3.10 Bezpečnostné testy 14
3.11 Smoke testy 15
3.12 Testovateľnosť aplikácií 15

4 Prototyp **17**
4.1 Porovnanie existujúcich nástrojov 17
4.2 Návrh a analýza prototypu 18
4.3 Databázový návrh 20
4.4 Vysvetlenie prototypu 22
4.4.1 Ukážka typických použití aplikácie 28
4.5 Amazon Web services 31
4.5.1 EC2 31
4.5.2 CodeDeploy 31
4.6 Nástroje 35
4.6.1 SCM 35
4.6.2 Zostavovanie aplikácie 35

4.6.3 Cloud computing 35
4.6.4 Consul 36
4.6.5 Bamboo 40
4.6.6 Postman 42
4.6.7 Newman 43

5 Implementácia **44**
5.1 Webová aplikácia pre užívateľské rozhranie 44
5.2 Orchestrátor 45
5.3 CCU 46
5.4 Testovanie 47

6 Záver **52**

Literatúra **54**

A Obsah CD **56**

B Manuál **57**

C Ukážka užívateľského rozhrania **58**

Kapitola 1

Úvod

V dnešnom dynamickom svete je nutné, aby software, ktorý je vyvíjaný a je dodávaný koncovému zákazníkovi musí byť dodávaný, čo najrýchlejšie a s čo najmenším počtom chýb.

Tento trend je v dnešnom svete pomerne ťažko dosiahnuteľný a už komplikovane dlhodobo neudržateľný, keďže zákazníci túžia po pravidelných aktualizáciách svojej obľúbenej aplikácie s cieľom získať stále nové a nové vylepšenia. Rovnako užívateľ citlivo vníma každú chybu v aplikácií, či už z pohľadu neprívetivého užívateľského rozhrania rovnako aj funkčnej chyby. Z pohľadu bežného programátora je avšak, táto zákazníkova túžba pomerne komplikovaná, keďže celý proces od konkrétneho zadania cez implementáciu a otestovanie naprieč rôznymi prostrediami až konkrétne vydávanie produktu do produkčného prostredia je obvykle pomerne zdĺhavý proces.

Treba mať napamäti, že spomalenie v akomkoľvek kroku môže viesť k značnému spomaleniu vydaniu celej aplikácie, teda aj pravdepodobnej straty pozície na trhu. V začiatkoch vývoja softwaru bol v rámci vývojového tímu dedikovaný človek, ktorý mal na starosti len riešenie problémov s nasadzovaním aplikácie na rôzne prostredia vrátane produkčného prostredia a riešenia problémov s prevádzkou aplikácie. Postupom času bolo ale z tohoto konceptu upúšťané kvôli rôznym zákonným obmedzeniam, aby vývojár nemohol mať prístup k dátam v produkčnom prostredí. Tento proces sa zdal byť efektívny, keďže sa výrazne znížila chybovosť vydaných aplikácií z dôvodu presunu zodpovedností za údržbu a vydávanie aplikácií na produkčné prostredie.

Tento proces, ale dosahuje svoje limity v aktuálnom dynamickom prostredí, kedy zákazník očakáva každý týždeň nejakú aktualizáciu svojej obľúbenej aplikácie.

Dnešný trend opäť smeruje ku konceptu, ktorý sa nazýva DevOps a teda vlastne bol používaný v minulosti. Treba zdôrazniť, že sa jedná o koncept. Koncept kladie dôraz na komunikáciu a zdôrazňuje kooperáciu medzi vývojom a prevádzkou a kladie dôraz na „Continuous delivery“ a „continuous integration“ s automatizovaným vydaním aplikácie.

Cieľom tejto práce je priblíženie tohoto konceptu, ktorý bude demonštrovaný na prototype. Tento prototyp je zložený z troch aplikácií, ktoré majú za úlohu automatizované testovanie a vydávanie serverových aplikácií. Prototyp bol odskúšaný v cloudovou prostrední Amazon Web Services na vzorových Java serverových aplikáciách.

Kapitola 2

Koncept DevOps

Koncept DevOps (Vývoj a prevádzka) vznikol s cieľom aktívneho rozvoja spolupráce medzi prevádzkovými tímami a vývojármi. Toto spojenie vzniklo z dvoch anglických slov a to „Development“ (vývoj) a „Operation“ (prevádzka alebo je to nazývané aj operácie). Tento koncept je tvorený komunitou IT expertov z celého sveta, ktorí o sebe snažia formovať osvedčené postupy, ktoré pomáhajú rýchlejšie prinášať požadovanú hodnotu smerom k zákazníkovi a toto predovšetkým prostredníctvom automatizácie manuálnych činností v procese vývoja a celkovou optimalizáciou procesov v IT. Koncept nájde predovšetkým uplatnenie pri pravidelnom nasadzovaní aplikácie v dôsledku krátkych vývojových cyklov (continuous delivery) a tiež tam, kde neexistujú jasne definované súvislé procesy pre spracovanie požiadavky od vzniku až po vydávanie do produkcie [8].

Jeden zo základných kameňov konceptu je, že vývojári musia úzko spolupracovať so zadávateľom, aby detailne porozumeli jeho potrebám. Zároveň sa vývojári musia uistiť, že výsledný kód je dostatočne otestovaný a nasadený do produkčného prostredia, čo vyžaduje pravidelnú súčinnosť prevádzkových tímov a vývojárov. Koordinácia týchto zainteresovaných skupín je činnosť, ktorá je v rade organizácií z väčších častí manuálna a následne náchylná k chybám. Manuálne a neprepojené procesy limitujú vývojové a prevádzkové tímy a ich schopnosti spolu efektívne komunikovať sa realizuje zväčša oneskorene a s častými chybami.

Koncept DevOps rieši nasledujúce problémy:

- Nové požiadavky na zmeny sú pomaly zrealizované a ku koncovému užívateľovi sa dostávajú so značným oneskorením
- Veľký počet incidentov v dôsledku neúspešne nasadených zmien aplikácie
- Nie je jasne definovaný proces, ktorý jednoznačne stanovuje pre všetky zainteresované role (napr. programátori, správcovia databází, prevádzkový personál) kroky, ktoré majú byť vykonané
- Výmena informácií medzi všetkými zapojenými tímami je väčšinou manuálna bez použitia profesionálnych nástrojov na to určených
- Obmedzená povedomie o tom, kto urobil danú zmenu najmä ak do projektu sú zapracované viaceré tímy

Týmito a ďalšími problémy súvisiacimi predovšetkým s kooperáciou výmenou informácií medzi prevádzkovým a vývojovým oddelením sa zaoberá súčasné poňatie DevOps.

Primárne charakteristika tejto kultúry je zamerená na zvýšenú spoluprácu medzi rolami vývoja a prevádzky. Existuje niekoľko významných kultúrnych zmien, v rámci tímov a na organizačnej úrovni, ktoré podporujú túto spoluprácu. Patrí medzi ne postoj zdieľanej zodpovednosti, ktorá podporuje užšiu spoluprácu. Ak vývojový tím zdieľa zodpovednosť o starostlivosť systému v priebehu jeho životnosti sú zároveň schopní zdieľať problémy s prevádzkovým tímom, a tak určiť spôsoby, ako zjednodušiť vydávanie a údržbu (napr. vďaka automatizácií vydávania a zlepšením logovania). Tak isto prevádzkový tím získa skúsenosti z monitorovania systému v produkčnom prostredí. V praxi sa spolupráca medzi týmito tímami často začína so zvýšeným vedomím smerom od vývojárov z prevádzkového tímu a prijatie nových automatizačných nástrojov a postupov nad činnosťami zamestnancov.

Niektoré organizačné posuny sú povinné podporovať kultúru spoločnej zdieľanej zodpovednosti. Nemalo by dochádzať ku konfliktom medzi vývojom a prevádzkou. Riešenie problémov prípravou dokumentácie k aplikácií je nedostatočné oproti spolupráci iniciovanou ihneď pri začiatku vývoji. Rovnako pomôže ak sú vývojári a prevádzkový tím umiestnený spolu v jednej kancelárií, čo im ešte viac uľahčí spoluprácu. Cieľom DevOps je, aby bol každý jeden zodpovedný za úspechy a neúspechy systému, aj keď možno k tomu nejako neprispel. DevOps kultúra sa snaží zotrieť jasne definovanú líniu medzi rolami vývojárov a prevádzkovým pracovníkom a eliminovať tento rozdiel.

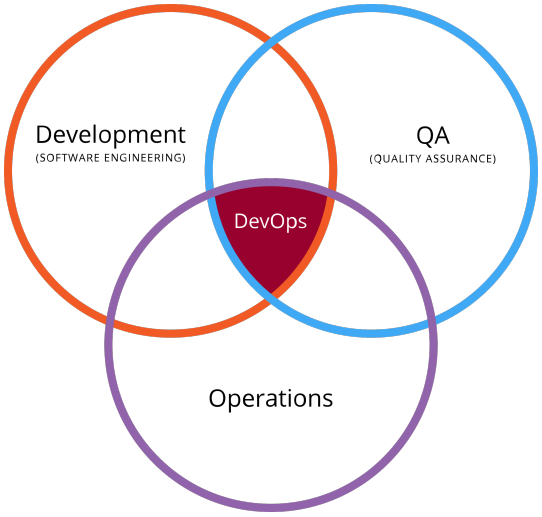
Ďalším cenným organizačným posunom smerom k DevOps je podporiť autonómnosť jednotlivých tímov. Aby bolo možné účinne spolupracovať, vývojári a prevádzkový personál musia byť schopní robiť rozhodnutia a aplikovať zmeny bez spletitých rozhodovacích procesov. Jedná sa o dôveru v tímy, ktorý mení spôsob rizika pri vytvorení a správe prostredia. Bez ručného podpisovania každého kroku a zavádzania zbytočného procesného managementu môžeme automatizáciu a vydávanie urýchliť.

Jedným z dôsledkov posunu smerom k DevOps kultúry je, že bude jednoduchšie vydať novú funkcionalitu v produkcii. To si vyžaduje niektoré ďalšie kultúrne zmeny. Na zabezpečenie toho, že zmeny v aplikácií sú v produkcii v poriadku, tím potrebuje zabudovať posudzovanie kvality aplikácie do svojho vývojového procesu. To zahŕňa funkčné, integračné, záťažové a bezpečnostné testy. Techniky Continuous Delivery (kontinuálneho dodávania), tvoria základ, ktorý umožňuje pravidelné aktualizácie, s nízkym rizikom zlyhania [8].

Je tiež dôležité, aby tím dostával spätnú väzbu, aby sa s neustále zlepšoval spôsob, akým pracujú vývojári a prevádzkový personál spoločne, rovnako ako samotný systém. Sledovanie produkčného prostredia je užitočná spätná väzba pre diagnostiku problémov a potenciálne zlepšenie v budúcnosti.

Automatizácia je základným kameňom konceptu DevOps a uľahčuje spoluprácu. Automatizácia úloh ako testovanie, konfiguráciu a vydávanie oslobodzuje ľudí zamerať sa na iné cenné činnosti a znižuje riziko ľudskej chyby. Užitočným vedľajším efektom automatizácie sú automatizované skripty a testy a vždy najnovšia dokumentácia systému.

Všetky popísané charakteristiky DevOps sú skôr odporúčania a neexistujú striktné a formálne definície. Tento koncept v sebe zahŕňa rôzne nástroje a prístupy používané v tomto koncepte.



Obr. 2.1: Koncept DevOps

Obrázok č. 2.1 popisuje aké tímy celkovo prepája. Ide teda o development tím (vývojari), quality assurance (testeri) a operations (prevádzkový tím). Diagram nám ukazuje, že koncept sa stavia do role prieniku medzi týmito tímami.

2.1 Blue/green vydávanie

Jedným z problémov spojených s automatizáciou vydávania aplikácie je otestovanie novej verzie aplikácie a zrušenie obsluhy prichádzajúcich požiadaviek do starej aplikácie, tj. aby komunikácia klienta prebiehala po nasadení len s novou verziou aplikácie. Zvyčajne je potrebné to urobiť rýchlo, aby sa minimalizovali prestoje. Prístup blue/green vydávanie to zaisťuje tak, že máte dve produkčné prostredia, ktoré sú zhodné najviac ako je to možné. Blue prostredie predstavuje aktuálne živú a používanú verziu aplikácie. V green prostredí sa pripravuje nová verzia aplikácie, ktorá obsahuje ešte záverečnú fázu testovania tesne pred spustením do produkcie. Akonáhle je softvér otestovaný a pracuje v green prostredí korektne, dochádza k prepnutiu smerovača (obecne nejakého prvku, ktorý smeruje prevádzku, môže ísť napr. aj o vyrovnávač záťaže) tak, aby všetky prichádzajúce požiadavky išli do green prostredia - blue prostredie je teraz nečinné.

Blue/Green vydávanie tiež poskytuje rýchly spôsob, ako vrátiť späť prípadnú zmenu. V prípade ak sa niečo pokazí, tak môžeme prepnúť smerovač späť do blue prostredia a tak odstrániť problém [3].

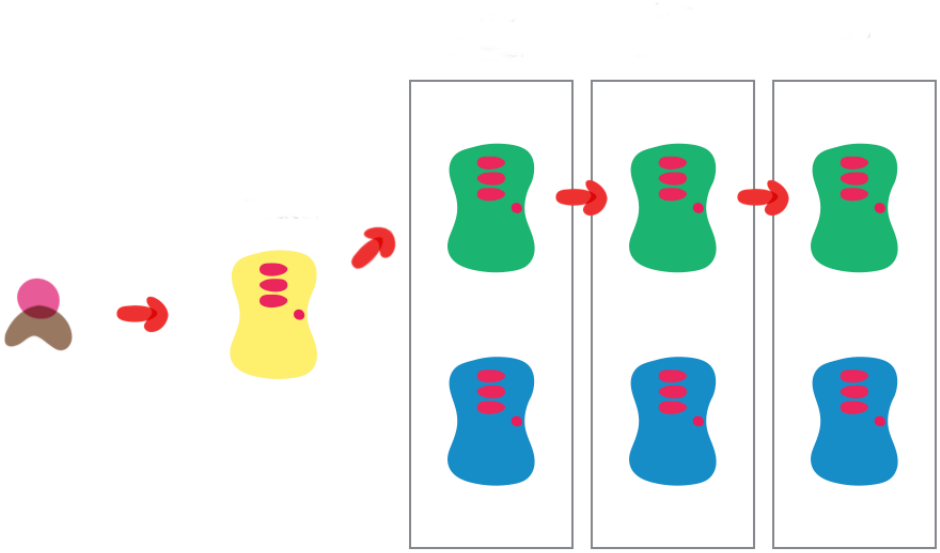
V niektorých prípadoch môžu byť pre blue a green prostredie použité rôzne kusy hardvéru, alebo to môžu byť rôzne virtuálne stroje bežiace na rovnakom (alebo odlišnom) hardvéri. Môže byť tiež použité jediné prevádzkové prostredie rozdelené do oddelených oblastí s dvoma oddelenými IP adresy pre dve verzie aplikácie.

Potom, čo sme svoje green prostredie označili ako živé (tj. je v produkcie a obsluhuje klientské požiadavky) a sme spokojní s jeho stabilitou, potom môžete použiť modré prostredie ako svoje pracovné prostredie pre testovanie a ako finálny krok pre ďalšie vydávanie novej verzie aplikácie tj. využijeme ho pri testovaní spätnej kompatibility našej budúcej verzie aplikácie. Pokiaľ pripravíme opäť novú verziu aplikácie celý proces prepínania a prípravy prostredí je zhodný.

Výhodou tohto prístupu je riešenie problémov s nejakým incidentom v produkcii. Z tohto dôvodu to umožní prevádzkovému tímu vyskúšať svoje postupy zotavenia aplikácie po havárii na každej vydanej verzii.

Základnou myšlienkou je mať dve ľahko prepínateľné prostredie.

Problém s touto technickou predstavuje databázová vrstva. Zvlášť keď je potrebné zmeniť schému na podporu novej verzie aplikácie. Trik je oddeliť zavádzanie zmien schémy od aktualizácie aplikácie. Takže najprv refaktorovať databázu, aby podporovala novú aj starú verziu aplikácie. Zároveň treba skontrolovať, či je všetko v poriadku, takže máme možnosť vrátiť sa k istému bodu späť a potom nasadiť novú verziu aplikácie.



Obr. 2.2: Blue/green nasadzovanie

Obrázok č. 2.2 zobrazuje techniku blue/green vydávanie. Celý postup riešenia problému bol popísaný vyššie. Postupne závadzame zmeny, kedy si najprv pripravíme jedno prostredie a následne dochádza k prepnutiu na nové prostredie.

2.2 Rolling release

Rolling release je metóda aktualizácie softvéru, ktorého filozofia namiesto vytvárania veľkých aktualizácií naraz, zahŕňa robenie veľa priebežnej aktualizácie. To znamená, že máme časté a menšie vydávania nových verzií aplikácie. Výhodou tejto metódy je, že aktualizácie vyjdú oveľa rýchlejšie, a sú zvyčajne jednoduchšie zvládnuteľné pre programátorov.

Väčšina programov sú aktualizované v priebehu času zvyčajne cez štandardné aktualizácie. Pri štandardných aktualizáciach, softvérový vývojár vytvorí úplne novú verziu programu a aktualizácie sa bežne vyskytujú každých pár týždňov alebo mesiacov. Ak vývojár používa schému rolling release, potom sa veci robia inak. Namiesto pravidelných aktualizácií, aktualizácie sú bežne vykonávané každý deň alebo raz za niekoľko dní.

V prípade, že program používa techniku rolling release, potom aktualizácie sú zvyčajne oveľa menšie, rovnako aktualizácie vyžaduje pochopiteľne menej času.

Program ktorý je často aktualizovaný bude fungovať lepšie, pretože aktualizácie, ktoré povedzme prinášajú so sebou novú funkcionalitu pre koncového zákazníka, obsahujú aj rôzne opravy.

Tento prístup má samozrejme okrem svojich výhod rovnako aj radu nevýhod. Pri štandardných aktualizáciách má vývojár množstvo času pre diagnostiku programu ohľadne chýb a vážnych problémov ovplyvňujúcich program. K dispozícii je tiež menej času pre testovanie. Program je menený tak často, že aj keď sú zmeny malé, môže sa softvér stať náchylný na hackerov a problémy vírusov.

2.3 Continuous delivery

Continuous Delivery je softvérovo-inžiniersky prístup, v ktorom tímy produkujú softvér v krátkych cykloch, čo zaisťuje, že softvér môže byť spoľahlivo uvoľnený kedykoľvek. Zameriava sa na vytváranie, testovanie a uvoľňovanie softvéru rýchlejšie a častejšie. Tento prístup pomáha znížiť náklady, čas a riziko a zároveň prináša zmeny tým, že umožňuje viac inštrumentálnych aktualizácií pre aplikáciu vo výrobe. Jasný a opakovateľný proces vydávania je dôležitý pre kontinuálne dodávanie [3]. Medzi výhody tohoto prístupu patrí:

- Váš softvér je nasaditeľný v celom ich životnom cykle
- Ktokoľvek môže dostať rýchlu, automatizovanú spätnú väzbu na pripravenosť verzie pre produkčné prostredie
- Môžete nasadiť kedykoľvek a akúkoľvek verziu aplikácie do produkcie
- Znížené riziko vydávania - nasadzujete menšie verzie aplikácie
- Spätná väzba od užívateľov - čím častejšie vydávate verziu aplikácie tým skôr zistíte, že zákazník má záujem o danú aplikácie

Kontinuálne dodávka a DevOps sú si podobné a ich významy a často sú zjednocované, ale oni sú dva rôzne pojmy. DevOps má širší rozsah, konkrétne spoluprácu rôznych tímov podieľajúcich sa na dodávaní softvéru (vývojárov, operácií, zabezpečenie kvality, riadenia atď), rovnako ako automatizáciu procesov v dodávaní softvéru. DevOps môže byť produktom kontinuálnej dodávky.

Existujú rôzne nástroje, ktoré pomáhajú splniť všetky alebo časti tohoto procesu. Tieto nástroje sú súčasťou celého vydávania, ktorá zahŕňa kontinuálne dodávanie, tj. od spracovania požiadavky, cez implementácie, pokračujúc testovaním naprieč rôznymi oblasťami až po vydávanie aplikácie do produkcie. Tieto typy nástrojov, ktoré vykonávajú rôzne časti procesu patrí: kontinuálna integrácia, automatizácia uvoľňovania aplikácie, správu životného cyklu aplikácie.

Kapitola 3

Testovateľnosť aplikácií

Nový prístup k tvorbe architektúre v moderných aplikáciách si vyžaduje nový prístup k testovaniu a zabezpečeniu požadovanej kvality pre koncového zákazníka. Zameriava sa predovšetkým na automatizované testovanie, pričom môžeme konkrétne vrstvy testovania rozdeliť na rôzne podčasti. Je veľmi dôležité zdôrazniť, že každá vrstva testovania (bezpečnosť, funkcionalita, ...) je dôležitá a preto práca len s jedinou vrstvou je nedostatočná, a vzhľadom na ich komplexnosť by mali byť všetky vrstvy zahrnuté do testovania aplikácie.[5]

V IT priemysle veľké spoločnosti majú tím so zodpovednosťou k vyhodnoteniu vyvinutého softvéru v rámci daných požiadaviek. Okrem toho vývojári tiež vykonávajú testovanie, ktoré sa nazýva jednotkové testy („unit testy“). Vo väčšine prípadov sa odborníci, ktorí sa podieľajú nejakým spôsobom na vývoji, sa podieľajú aj na testovaní systému v rámci svojej zodpovednosti:

- Tester
- Vývojár softvéru
- Vedúci projektu / Manažér
- Koncový užívateľ

Rôzne spoločnosti majú rôzne označenia pre ľudí, ktorí testujú softvér na základe svojich skúseností a znalostí, ako je tester, Software Quality Assurance Engineer tzv. QA analytik, ...

Nie je možné testovať softvér kedykoľvek počas jeho cyklu vývoja. Je veľmi dôležité si uvedomiť, že testovanie by malo začať, čo najskôr. Tento prístup napr. podporuje „test driven development“ (TDD) tzv. vývoj riadený testami, ktorý je postavený na tom, že sa napíšu jednotkové testy a až následne sa píše funkčný kód.

Obecne však platí, že treba začať, ako už bolo spomenuté, čo najskôr, pretože skorý začiatok testovania znižuje náklady projektu a čas prepracovať a produkovať bezchybný softvér, ktorý je dodávaný ku klientovi. Testovanie môže byť spustené už od zhromažďovania požiadaviek a pokračovať až do vydávanie softvéru (aplikácie) do produkčného prostredia. Záleží tiež na modeli vývoja, ktorý je používaný. Napríklad v modeli vodopádu („waterfall“), formálne testovanie sa vykonáva až v testovacej fáze. Ale v inkrementálnom modeli vývoja sa testovanie vykonáva na konci každého prírastku / iterácii a celá aplikácia je testovaná až na konci.

Testovanie sa vykonáva v rôznych formách v každej fáze vývoja. V priebehu zberu fázy požiadaviek, analýzy a overenia požiadaviek sú tiež považované za testovanie. Preskúmanie

návrhu vo fáze návrhu produktu s úmyslom zlepšiť dizajn je tiež považované za testovanie. Testovanie, ktoré vykonáva vývojár na dokončenie kódu je tiež zaradené do kategórie testovania.

Rovnako je možné testovanie nejakým spôsobom rozčleniť a to na manuálne testovanie, kedy nie je možná automatizácia a testovanie je vykonávané testerom a automatizované testovanie, kedy sú používané pripravené aplikácie, moduly alebo iné prostredia, ktoré za nás skontrolujú nami sledované požiadavky aplikácie z pohľadu výkonu a bezpečnosti, ...

3.1 Manuálne testovanie

Manuálne testovanie zahŕňa testovanie softvéru ručne, to znamená bez použitia akéhokoľvek automatizovaného nástroja. V tomto type tester preberá úlohu koncového užívateľa a testuje softvér pre identifikáciu akéhokoľvek neočakávaného správania alebo chyby. Testeri môžu použiť skúšobné plány, testovacie prípady, alebo testovacie scenáre a testovať softvér s cieľom zabezpečiť úplnosť testovania.

3.2 Automatizácia testovania

Automatizácie testovania je založená na napísaných pripravených scenároch a používa sa iný softvér na testovanie produktu. Tento proces zahŕňa automatizáciu manuálneho procesu. Automatizácia testovania sa používa pre opätovné spustenie testovacích scenárov, ktoré boli zadané ručne, rýchlo a opakovane. Typicky, ale nie je možné automatizovať v aplikácií všetko. Typické ide o oblasti, v ktorom užívateľ môže vykonávať transakcie, ako sú napr. prihlasovací formulár alebo registračných formulár, alebo akékoľvek oblasti, kde veľký počet užívateľov môže pristupovať k softvéru súčasne. Naopak je možné automatizovať pokiaľ ide o všetky položky nejakého užívateľského rozhrania, spojenie s databázou, validácia vstupných polí, a podobne možno účinne testovať tým, že sa automatizuje manuálny proces.

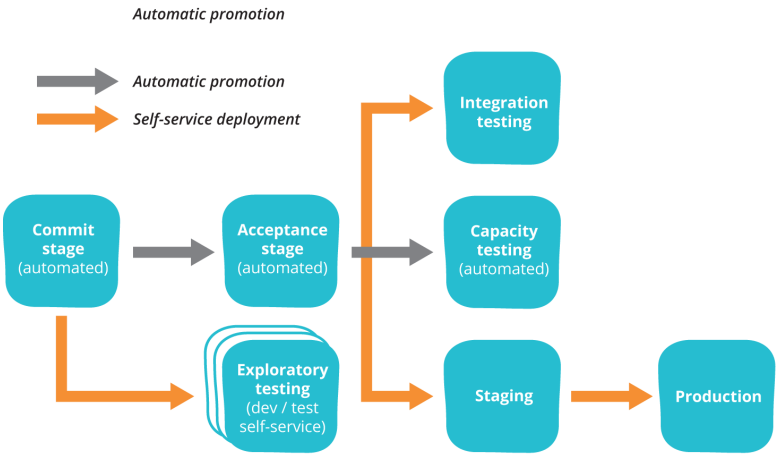
Automatizácia testovania by mala byť použitá s ohľadom na tieto aspekty softvéru:

- Veľké a kritické projekty
- Projekty, ktoré vyžadujú testovanie rovnakej oblasti často
- Požiadavky sa nemenia často
- Stabilný softvér s ohľadom na manuálne testovanie

Automatizácia sa vykonáva pomocou podporujúcich programovacích jazyk na skriptovanie a automatizačných softvérových aplikácií. Existuje mnoho nástrojov, ktoré môžu byť použité na napísanie automatizačných skriptov. Pred uvedením nástrojov je nutné definovať proces a požiadavky automatizácie:

- Určenie oblastí, v rámci softvéru pre automatizáciu
- Výber vhodného nástroja pre automatizáciu testovania
- Písanie testovacích skriptov
- Spúšťanie skriptov
- Vytvorenie výslednej správy

- Identifikovať prípadné chyby alebo problémy s výkonom
- Nástroje na testovanie softvéru



Obr. 3.1: Testovanie spojené s vývojom

Na obrázku č. 5.1 ja načrtnutá zjednodušená schéma automatizovaného testovania spojené s vývojom aplikácie. V ľavej časti obrázku (commit stage) predstavuje náš verzovací systém, či už v podobe Git, SVN alebo iných, do ktorého sú ukladané zdrojové kódy našej aplikácie. Po vložení našich zmien do tohoto verzovacieho systému sú automaticky, za použitia podporných nástrojov, vykonané automatizované testy. Na obrázku vidíme časť „Acceptance stage“ v rámci, ktorej sú vykonané akceptačné testy a následne sa automatizovane vykonávajú „Capacity testing“, nazývané aj záťažové testy. Jednotlivé automatizované časti poskytujú spätnú väzbu vývojovému tímu, či už formou nejakej správne alebo notifikácie o výsledkoch testov. Jednotlivé testy môžu byť rozšírené o ďalšie automatizované testy ako napr. bezpečnostné testy.

3.3 Jednotkové testy

Jednotkové testy nie sú ničím novým a v dnešnej dobe takmer každý chápe výhody tohoto prístupu, ktorý sa nazýva „Test driven development“ (TDD) tzv. testami riadený vývoj. Všeobecne platí, že jednotkové testy sú automatickou kontrolou, ktorá nám poskytuje spätnú väzbu, či naša implementácia je v súlade s požiadavkami. Tieto testy sú na kontrolu iba malých kúskov kódu tzv. jednotiek kódu. Tento typ testovania sa vykonáva vývojármi pred samotnou implementáciou a následne sa až implementuje funkcionalita. Následne sa kompletná funkcionalita vrátane testov odovzdáva testovaciemu tímu formálne vykonať testovacie prípady. Jednotkové testovanie sa vykonáva pomocou príslušných vývojárov na jednotlivých jednotkách zdrojového kódu [7].

Cieľom jednotkového testovania je izolovať jednotlivé časti programu a ukázať, že jednotlivé časti sú správne, pokiaľ ide o požiadavky a funkčnosť.

3.3.1 Obmedzenie jednotkových testov

Jednotkové testovanie nemôže odchytiť každú chybu v aplikácii. Je možné ohodnotiť, ale každé spustenie a každú možnú cestu v aplikácii. Rovnako tak je tomu v prípade testovania. Tento typ testovanie ale neodhalí chyby pri integrácií s ostatnými aplikáciami, rovnako ani neodhalí prípadnú neefektivitu kódu danú zlou implementáciou.

3.4 Integračné testovanie

Integračné testovanie je definované ako testovanie jednotlivých aplikácií, služieb, modulov, ktoré navzájom komunikujú a tým zabezpečujú požadovanú funkcionalitu. Integračné testovanie možno vykonať dvoma spôsobmi: zdola-nahor testovanie integrácie a testovanie integrácie zhora-nadol. V komplexnom prostredí pre vývoj softvéru sa prístup zdola-nahor zvyčajne vykonáva ako prvý, nasledovaný až prístupom zhora-nadol. Tento proces sa končí viacerými testami, ktoré majú napodobňovať skutočné situácie, ktoré môžu nastať počas behu aplikácie v produkčnom prostredí.

3.4.1 Prístup zdola-nahor

Toto testovanie začína jednotkovými testami, ktoré je nasledované testami v postupnej kombinácií vyšších úrovni jednotiek nazývaných moduly alebo služby.

3.4.2 Prístup zhora-nadol

V tomto type testovania sa moduly na najvyššej úrovni testujú ako prvé a postupne moduly nižšej úrovne až sa otestuje kompletne aplikácia ako celok.

3.5 Regresné testovanie

Kedykoľvek je vykonaná zmena v aplikácii, tak je celkom možné, že iné oblasti kódu boli postihnuté touto zmenou. Regresné testovanie sa vykonáva za účelom overenia, že opravená chyba alebo naopak pridaná funkcionalita neviedla k inému narušeniu funkčnosti či obchodných pravidiel. Zámerom regresného testovania je zabezpečiť, aby zmena, ako je napríklad oprava chyby by nemali mať za následok inú chybu a bola odhalená už na počas implementáciu riešenia.

Regresné testovanie je dôležité z nasledujúcich dôvodov:

- Zmierňuje riziká, že pri zmene nebola poškodená už implementovaná funkčnosť
- Zvýšenie rýchlosti dodania produktu na trh

3.6 Systémové testovanie

Pri systémovom testovaní testujeme aplikáciu ako celok. Akonáhle sú všetky komponenty integrované do aplikácie je testovaním potrebné dôsledne skontrolovať, či spĺňa stanovené normy kvality. Tento typ testovania sa vykonáva pomocou špecializovaného testovacieho tímu.

Systémové testovanie je dôležité z nasledujúcich dôvodov:

- Systémové testovanie je prvým krokom v životnom cykle vývoja aplikácie, kedy je aplikácia testovaná ako celok
- Aplikácia je dôkladne testovaná za účelom overenia, že spĺňa funkčné a technické špecifikácie
- Aplikácia je testovaná v prostredí, ktoré je veľmi blízko do produkčného prostredia, kde bude nasadená aplikácia
- Testovací systém nám umožňuje otestovať, overiť a potvrdiť požiadavky na obchodnú rovnako ako aplikačnú architektúru

3.7 Akceptačné testovanie

Toto je pravdepodobne najdôležitejším typom testovania, pričom je zaisťovaná kvalitatívnym tímom (nazývaný aj „quality assurance“ (QA)), ktorý ma posúdiť, či funkčnosť spĺňa zamýšľanú špecifikáciu a spĺňa požiadavky klienta. QA tím má sadu predpísaných scenárov a testovacích prípadov, ktoré budú použité pre testovanie aplikácie. Tieto testy nie sú zamýšľané, len aby poukázali na jednoduché pravopisné chyby, kozmetické chyby alebo nedostatky rozhrania, ale tiež aby poukázali na chyby v aplikácii, ktorá budú mať za následok zrútenie systému alebo iné veľké chyby v aplikácii.

Vykonaním akceptačných testov bude QA tím schopný odvodiť, ako sa bude aplikácia používať v produkčnom prostredí. Existujú totiž právne aj zmluvné požiadavky z pohľadu zákazníka pre prijatie systému, ktoré musia byť splnené.

3.8 Výkonnostné testovanie

Tento typ testovania je používaný na identifikáciu akejkoľvek prekážky alebo problémov s výkonom, skôr než dochádza k používaniu aplikácie. Existujú rôzne príčiny, ktoré prispievajú k znižovaniu výkonu softvéru [5]. Môže ísť o rôzne problémy s pripojením, teda obecne so sieťovými problémami, teda, že sú prichádzajúce požiadavky značne oneskorené. Rovnako môže byť problém priamo u klienta, teda v klientskej aplikácií, ktorá nie je optimálna. Rovnako môže ísť o problém pri využívaní trvalého úložiska v podobe databázy ako o chybu resp. o neoptimálny návrh schémy alebo naopak výber zlej technológie na perzistenciu dát. Obecne môžeme povedať, že problémov tohoto typu je viac a typicky sú odhalované až pri veľkej záťaži aplikácie.

Testovanie výkonu je považované za jeden z dôležitých a povinných testov, pokiaľ ide o tieto aspekty:

- Rýchlosť aplikácie
- Stabilita aplikácie
- Kapacita aplikácie
- Škáľovateľnosť aplikácie

Testovanie výkonu môže byť buď kvalitatívne alebo kvantitatívne a môže byť rozdelené do niekoľkých čiastkových častí ako záťažové testy a stress testy [7].

3.8.1 Záťažové testy

Je to proces testovania správania sa softvéru za použitia maximálneho zaťaženie, pokiaľ ide o aplikačný prístup a prácu s veľkými vstupnými dátami. To môže byť vykonané v normálnych podmienkach alebo špičkových zaťaženiach aplikácie. Tento typ testovania určuje maximálnu kapacitu softvéru a jeho správanie v čase špičky.

Záťažové testovanie sa vykonáva pomocou automatizovaných nástrojov, ako napr. Apache JMeter, Silk Performer, ...

3.8.2 Stress testy

Stresové testovanie zahŕňa testovanie správania sa softvéru na základe abnormálnych podmienok. Napríklad môže obsahovať, že sa odstránia niektoré prostriedky, alebo sa aplikuje záťaž nad skutočné medze zaťaženia.

Cieľom záťažových testov je testovanie softvéru aplikácií zaťažením systému a odobratie zdrojov softvéru na identifikáciu bodu zlomu, teda kedy dochádza k padnutiu aplikácie resp. servera. Toto testovanie môže byť vykonané testovaním rôznych scenárov, ako sú:

- Vypnutie alebo reštart sieťových portov náhodne
- Zapnutie alebo vypnutie databázy
- Spustenie rôznych procesy na serveri, kde je aplikácia, ktoré spotrebúvajú zdroje, ako je procesor, pamäť, ...

3.9 Testovanie použiteľnosti

Testovanie použiteľnosti je black-box technika a slúži na identifikáciu akýchkoľvek chýb s cieľom zlepšiť softvér tým, že sleduje používateľa prostredníctvom ich využitia a prevádzky aplikácie [5]. Cieľom toho testovania je splnenie štyroch cieľov, ktoré si dáva za cieľ testovanie použiteľnosti: práca s aplikáciou sa ľahko učí, ľahko zapamätateľné akcie v aplikácií, efektívne využívanie dostupných prostriedkov v aplikácií, ľahko pochopiteľná aplikácia. Okrem rôznych definícií použiteľnosti existujú normy a kvalitatívne modely a metódy, ktoré definujú použiteľnosť v podobe atribútov a čiastkových atribúty, ako je ISO-9126, ISO-9241-11, ...

3.10 Bezpečnostné testy

Testovanie zabezpečenia aplikácie zahŕňa testovanie softvéru s cieľom zistiť prípadné nedostatky a medzery v bezpečnosti a zraniteľnosti. Nižšie sú uvedené hlavné aspekty, ktoré testovanie bezpečnosti by malo overiť:

- Dôvernosť
- Kontrola vstupu a validácia
- Overenie pravosti
- Integrita
- Softvér je v súlade so všetkými bezpečnostnými predpismi

- Nepopierateľnosť

Obecne môžeme pre tento typ testovania použiť metodológiu „OWASP Top 10“, ktorá definuje desať najčastejších zraniteľností aplikácií. Obsahuje rôzne podkategórie a špecifické útoky na aplikáciu rovnako aj postupy, ako prípadné chyby odstrániť. Samozrejme je možné použiť rôzne automatizačné nástroje, ktoré vyhodnotia prípadnú zraniteľnosť aplikácie za nás, napr. Qualysis.

3.11 Smoke testy

Smoke testy je predbežné testovanie s cieľom odhaliť jednoduché poruchy natoľko vážne, aby (napríklad) boli odmietnuté tesne pred vydaním novej verzie aplikácie. Tester pri smoke testoch vyberá a spúšťa podmnožinu testovacích prípadov, ktoré pokrývajú najdôležitejšie funkcie komponent alebo systému, aby zistil, či zásadné funkcie softvéru fungujú správne [6].

Proces smoke testovania si kladie za cieľ rýchlo zistiť, či aplikácia je v nefunkčnom stave, aby sa ďalšie testovanie zbytočne nevykonávalo. Smoke testy musia často bežať rýchlo, často len niekoľko minút, pričom ich výhodou je rýchlejšia spätná väzba a rýchlejší obrat, ako chod celých sérií testov, čo môže trvať niekoľko hodín - alebo dokonca dní. Rovnako ich výhoda je, že typicky bývajú tieto testy automatizované.

3.12 Testovateľnosť aplikácií

V kapitolách uvedených vyššie je vykonaný prehľad rôznych typov testovaní. Tento prehľad nám bude slúžiť na analyzovanie požiadaviek testovateľnosti ľubovoľnej aplikácie. Každá aplikácia, ktorá je vyvíjaná s cieľom uvedenia tejto aplikácie do produkcie, musí byť ešte predtým nutne otestovaná. Každá aplikácia by mala byť elementárne testovateľná z pohľadu funkčnosti. Je nutné teda zabezpečiť, aby každá vyvinutá funkcionalita spĺňala potrebné zadanie tj. musíme otestovať funkčnosť danej aplikácie. Testovanie funkcionality sa realizuje prostredníctvom funkcionálnych testov. Medzi pomerne časté testovanie funkcionality je napríklad testovanie restového rozhrania prostredníctvom HTTP dotazov a spracovania odpovedí. Ďalším možným typom testovania spadajúce do funkcionálneho testovania je testovanie užívateľského rozhrania. Typicky sa pritom používa sada nástrojov, ktoré automatizujú ručné overovanie. Tak napríklad overíme, či rozhrania spĺňajú predpísané požiadavky. Vývijanú funkcionalitu je možné otestovať aj z hľadiska bezpečnostných dier. Je to možné realizovať použitím špeciálnych nástrojov na to určených, ktoré sa integrujú priamo do vývoja.

Po vyvinutí danej funkcionality prebieha typicky vydávanie aplikácie spolu s novou funkcionalitou na testovací server. Pred zostavovaním aplikácie a jej samostatným vydaním sa spúšťa sada jednotkových testov, ktoré zabezpečujú overenie zanechania pôvodnej funkcionality spolu s novou. Typicky sa po nasadení ešte overuje, či je funkcionalita nasadená správne pomocou funkčných testov. Následne je možné vykonať integračné testy. Pri tomto type testov sa typicky testuje integrácia aplikácie s iným aplikáciami. Pri restových službách sa typicky testuje komunikácia naprieč jednotlivými aplikáciami.

Niektoré aplikácié, na ktoré typicky pristupuje niekoľko tisíc klientov za sekundu vyžadujú, aby spĺňami podmienky vysokej dostupnosti. Pre tieto aplikácie je nutné otestovať okrem správnej funkcionality aj ich výkon. Pre tento typ používame záťazové testy. Aplikácia je nasadená na server, ktorého výpočtová kapacita odpovedá približne výpočtovej

kapacite produkčného serveru s cieľom overiť dostupnosť na podobnom prostredí. Pri tomto type testovania typicky spúšťame sadu funkcionálnych testov pomocou špeciálnych nástrojov, ktoré spustia túto sadu viacnásobne za sekundu, čím vlastne simulujú približnú reálnu prevádzku. Následne sme schopní určiť výsledky typicky čas odpovede, ktorý je pre nás kľúčový a typicky by mal byť v jednotkách milisekúnd. Rovnako je možné naopak aj overiť, že nami zvolené riešenie z pohľadu infrakštuktúry je dostatočné, alebo naopak je možné zvoliť riešenie menšej kapacity, čo nám typicky môže ušetriť aj finančné prostriedky. Alebo môžeme dôjsť k výsledku, že je nutné zmeniť technológiu aplikácie, či už z pohľadu programovacieho jazyka alebo databázy alebo inej použitej technológie.

Ďalším typickým krokom tesne pred vydaním aplikácie do produkcie je testovanie použitelnosti. Typicky testuje užívateľské rozhranie aplikácia z pohľadu jednoduchosti, príveTíveho užívateľského rozhrania. Testujú sa pritom rôzne scenáre, ktoré môžu byť aj značne náhodné a nemusia viesť k spusteniu funkcionality v aplikácií, ale naopak vedú k tomu, aby došlo k pochopeniu používania aplikácie. Realizuje sa to väčšinou na rozličnej vzorke potencionálnych používateľnosti, či už s vysokou alebo nízkou mierou znalosti aplikácie s cieľom odhaliť potencionálne slabiny a zmätky v návrhu. Posledným krokom tesne po vydaní aplikácie je akceptačné testovanie, ktorého účel je overiť, či aplikácia spĺňa zadané kritéria.

Po overení funkcionality, bezpečnosti aplikácie a jej maximálnej znesiteľnej záťaže podľa požiadaviek na aplikáciu je aplikácia pripravená na vydávanie do produkčného prostredia. Po nasadení aplikácie do produkčného prostredia dochádza k otestovaniu aplikácie ešte pred jej sprístupnením vonkajšiemu svetu pomocou smoke testov. Typicky je v tomto štádiu aplikácia sprístupná v rámci internej siete a teda nie je možné ju používať klientmi z vonkajšieho sveta. Smoke testy v tomto kroku otestujú vybranú podmnožinu funkcionality, ktorá overí, či je aplikácia správna.

Je nutné zdôrazniť, že jednotlivé testy, ktoré sa vykonávajú počas celej fázy vývoju aplikácie až po jej vydávanie je zväčša manuálne a je teda možné ho automatizovať. Pomerne jednoducho automatizovaľné prostredníctvom dnes známych nástrojov ako je napr. Bamboo, Teamcity atď. je možný vývoj novej funkcionality. Je možné automatizovane spúšťať funkcionálne testy nad aktuálne vyvíjanou funkcionalitou spolu s jednotkovými testami. Tieto testy typicky sú schopné odhaliť, či nedošlo k narušeniu už k existujúcej funkcionality. Rovnako testovanie novej funkcionality užívateľského rozhrania je možné automatizovať pomocou špeciálnych nástrojov napr. SoapUI[1]. Typicky pri nich máme zostavené scenáre rozhrania, ktoré chceme otestovať.

Avšak okrem automatizácie, ktorá zapadá do moderného konceptu kontinuálnej dodávky a integrácie je vždy nutné manuálne testovanie testera, ktorý otestuje rôzne scenáre s cieľom pochopiť aplikáciu prípadne nájsť potencionálne slabé miesta aplikácie alebo nájsť možné vylepšenia. Jednotlivé typy testovania a fázy, pri ktorom boli použité sa môžu líšiť pri rôznych aplikáciách a je teda vždy nutné zvážiť ako testovať našu aplikáciu, či to dáva zmysel.

[1] https://www.soapui.org/

Kapitola 4

Prototyp

V tejto kapitole bude postupne popísaný celý prototyp, ktorý bude implementovaný. Tento prototyp ukazuje spôsob riešenia kontinuálnej integrácie a vydávania aplikácie do cloudového prostredia spolu s bez výpadkovým vydaním aplikácie do produkčného prostredia. Cieľom tohoto prototypu je snaha o koncept, ktorý funguje aj v reálnych podmienkach a teda môže byť adaptovaný aj do firemného prostredia.

Je dôležité zdôrazniť, že prototyp, tj. realizácia automatizovaného vydávania bude realizovaná prostredníctvom troch aplikácií. Jedna aplikácia je aplikácia, ktorá sa bude skladať z 2 menších aplikácií a to z frontendovej časti reprezentovaná užívateľským rozhraním umožnujúca rôzne interakcie na základe špecifikácie a s backendovou aplikáciou nazývanou Orchestrátor. Táto aplikácia bude riadiť celý beh automatického testovania, rovnako bude poskytovať informácie o dostupných verziách aplikácie, spolu s informáciami o aktuálne vydaných verzíach aplikácie naprieč rôznymi predprodukčnými prostrediami a produkčným prostredním. Posledná aplikácia bude zodpovedná sa záverečné smoke testovanie priamo v cloude tesne pred spustením novej verzie aplikácie. Túto aplikáciu budeme nazývať Centrálna riadiaca jednotka (central control unit, tzv. CCU). Konfigurácia pre jednotlivé aplikácie s informáciami o testovacích serveroch bude uložená v nástroji Consul, ktorý bude poskytovať webové rozhranie na vkladanie konfigurácií. Pre zostavovanie a vydávanie aplikácií bol použitý nástroj Bamboo. Celý prototyp bude implementovaný pre požiadavky cloudu Amazon Web Services.

Konkrétne detaily o prototype budú vysvetlené v ďalšom texte.

4.1 Porovnanie existujúcich nástrojov

V tejto časti bude vysvetlená príčina implementácie takéhoto prototypu a bude spravené porovanenie existujúch nástrojov s mojim riešením. Porovnanie bude prebiehať medzi nástrojmi Jenkins, Bamboo, Teamcity a mojim riešením [9]. Všetky nástroje majú niečo spoločné a to, že ide o nástroje pre podporu kontinuálnej integrácie a dodávky, teda zabezpečujú zostavenie aplikácie a po prípade jej vydávanie podľa potreby. Všetky tieto nástroje majú webové grafické užívateľské rozhranie prostredníctvom, ktorého je možné sledovať aktuálny stav prekladu aplikácií a realizovať jej následne vydávanie. Rovnako majú podporu pre rôzne užívateľské role a ponúkajú rôzne typy upozornení v prípade úspešného alebo neúspešného zostavenia alebo vydávania aplikácie.

Je nutné si, ale stanoviť požiadavky na prototyp, ktoré budú ovplyvňovať celkový výber použitého nástroja. Prototyp musí nutne byť schopný zostavenia a vydávania aplikácie

v programovacom jazyku Java, obecne ale musí byť ľahko nastaviteľný na prácu s ľubovoľným programovacím jazykom. Rovnako vydávanie aplikácie bude prebiehať do cloudového prostredia Amazon Web Services. Rovnako je nutné, aby vydávanie aplikácie prebiehalo bez výpadkov so schopnosťou vrátenia prípadnej zmeny. Celý proces vydávania musí byť zaznámenávaný, aby v prípade chyby bolo jasne viditeľné, kde nastala chyba.

Z vyššie zadefinovaných požiadaviek môžeme prejsť k samostatným nástrojom. Jednotlivé nástroje budem rozoberať z viacerých hľadísk. Najprv sa pozrieme na nástroje Bamboo a Teamcity. Oba tieto nástroje patria medzi komerčné, to znamená, že pre ich používanie musíte platiť licenciu. Podpora v týchto nástrojoch je cez online dokumentáciu, komunitu, ktorá v tomto prípade nie je veľká, keďže sa nejedná o open source nátroje, alebo prostredníctvom externej platenej podpory. Oba tieto nástroje ponúkajú širokú škálu pluginov, medziktorými je aj vydávanie aplikácie cloudu do Windows Azure alebo do Amazon Web services. Bamboo má všetky tieto pluginy spoplatnené. Bamboo a Teamcity po funkčnej stránke pracujú skoro rovnako. Obsahujú prehľadné webové užívateľské rozhranie s podporou projektov, užívateľských rolí a ďalších funkcionalít vrátane restového API pre prácu s týmto nástrojom. Na druhej strane nástroj Jenkins [11]. Ten ponúka rovnako širokú škálu funkcionality. V prvom rade treba spomenúť, že ide o nástroj open source z tohoto dôvodu nie je potrebná žiadna licencia pre používanie nástroja. Tento nástroj rovnako ponúka možnosť zostavovanie a vydávania aplikácie. Nástroj je koncipovaný odlišne ako predošlé nástroje. Ponúka ale tak isto rozšíriteľnosť tohoto nástroja prostredníctvom pluginov. Čo sa týka podpory tohoto nástroja, tak je realizovaná prostredníctvom dostupnej dokumentácie a širokej komunity. Rovnako nástroj ponúka restové rozhranie, ktoré umožňuje vzdialené spustenie, zostavenie a vydávania aplikácie.

Môj nástroj má byť postavený univerzálne a preto bude využívať práve jeden z tých nástrojov. Je to daný tým, že všetky uvedené nástroja realizujú orchestráciu. Tým je myslené, že využívajú ďalšie nástroje na zostavenie ako je napríklad nástroj Maven, Ant atď. Nad týmito nástrojmi majú postavené spúšťanie prostredníctvom plánov, ktoré predstavujú nejakú základnú jednotku, ktorá vyvolá rôzne akcie. Týmito akciami je myslené ako bolo spomenuté zostavenie aplikácie, stiahnutie zdrojových kód alebo naopak vydávanie aplikácie a tieto akcia sú konfigurovateľné. Bez výpadkové nasadenie je realizované samotným nástrom pre nasadenie danej verzie aplikácie na serveri v rámci použitého cloudu Amazon Web Services. Z hľadiska požiadaviek a dostupných nástrojov v rámci organizácie bude vybraný nástroj Bamboo, ktorý bude používaný na úkony s aplikáciami a bude pracované prostredníctvom jeho restového rozhrania.

4.2 Návrh a analýza prototypu

Cieľom tohoto prototypu je tvorba automatizačného nástroja na podporu zostavovanie a testovanie aplikácií v testovacom prostredí a následnom vydaní do produkčného prostredia. Prototyp je univerzálny z pohľadu programovacieho jazyka. Proces testovania je možné ovplyvniť priamo vo webovom grafickom užívateľskom rozhraní, kde môže užívateľ zvoliť automatizované testovanie pri zmene, ktorá nastane pridaním novej časti kódu do verzovacieho sytému alebo naopak môže testovať danú požadovanú funkcionalitu manuálne pomocou nástroja tretej strany. Pre testovanie serverových aplikácií bol vybraný nástroj Postman, ktorý bude bližšie popísaný v kapitole č. 4.6.6, prostredníctvom kolekcie testov a prostredia. Tento nástroj umožnuje jednoduchú tvorbu testov pre restové rozhranie. Tento má aj realizáciu, ktorú je možné spúšťať z príkazového riadku a nazýva sa Newman, ktorý bude bližšie vysvetlený v kapitole 4.6.7.

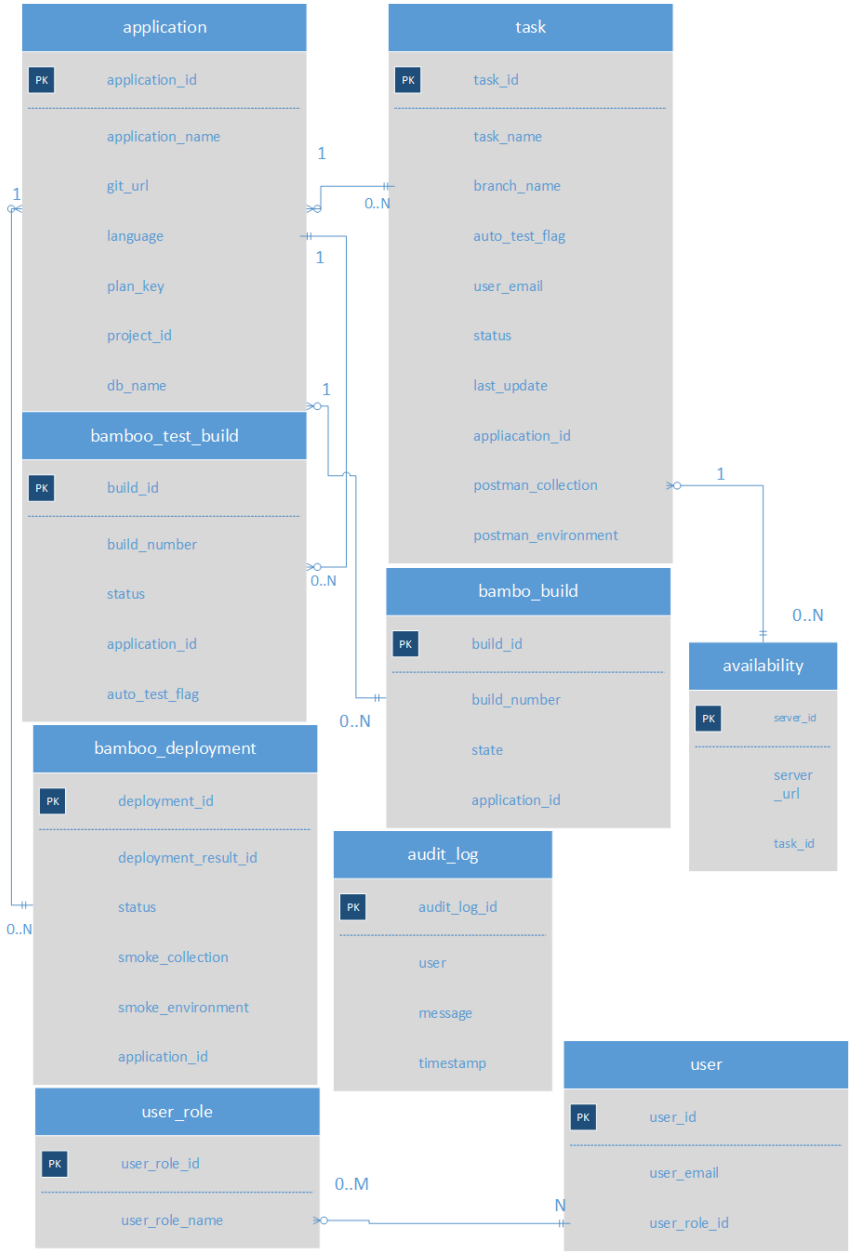
Ako bolo spomenuté prototyp konkrétne backendová aplikácia Orchestrátor pri zistení zmeny v Git-e aplikácie spustí vydávanie aplikácie na voľný a vhodný testovací server (server musí podporovať behové prostredie aplikácie) a otestuje aplikácie cez sadu testov. Výsledok je následne späť zobrazovaný vo frontendovej aplikácií, kde má užívateľ informáciu o stave jeho vyvíjanej funkcionality z pohľadu overenia testami. Po vyvinutí danej funkcionality je vytvorený pull request pre danú vetvu. Tento krok je realizovaný mimo prototyp a je zabezpečovaný priamo aplikáciami ako je napr. Bitbucket, Github, atď, ktoré ponukajú grafické užívateľské rozhranie, kde je možné sledovať, aké zmeny boli vykonané pri vyvinutí danej funkcionality. Sú zvolení recenzenti, ktorí sú programátori, ktorí overujú kvalitu vytvoreného kódu spolu s koncepciou aplikácie. Po fáze revízie prebieha vloženie zmien z novej vyvíjanej funkcionality späť do hlavnej vývojovej vetvy v Git-e, ktorá býva typicky označovaná ako dev vetva. Táto vetva obsahuje najnovšie vyvinuté požiadavky. Tieto požiadavky sú pripravené na vydávanie do predprodukčného a produkčného prostredia. vydávanie prebieha vložením zmien z dev vetvy do vetvy release. Z tejto vetvy sa vytvorí tzv. vydávateľný balíček aplikácie, čo je vlastne balíček preloženej aplikácie nejakej verzie, ktorý je pripravený na vydávanie do produkčného prostredia. Celý proces práce s Git-om sa nazýva Git flow [13] a predstavuje sadu overených princípov pri vývoji aplikácie[1]. Samozrejme existujú aj iné modely, to ale nie je súčasťou tejto práce. Celý proces zmien v aplikácií je sledovaný a zaznamenávaný. Realizuje sa to z dôvodu súmarneho prehľadu aplikácie a pri výpadku aplikácie z dôvodu sledovania chýb. Prototyp rovnako podporuje jednoduchý správu užívateľov a definuje užívateľské role vydavateľa, administrátora, vývojára a auditora.

Jednotlivé prípady budú vysvetlené neskôr.

[1] http://nvie.com/posts/a-successful-git-branching-model/

4.3 Databázový návrh

Aplikácia Orchestrátor vyžaduje pre svôj beh databázový model, ktorý je popísaný nižšie:



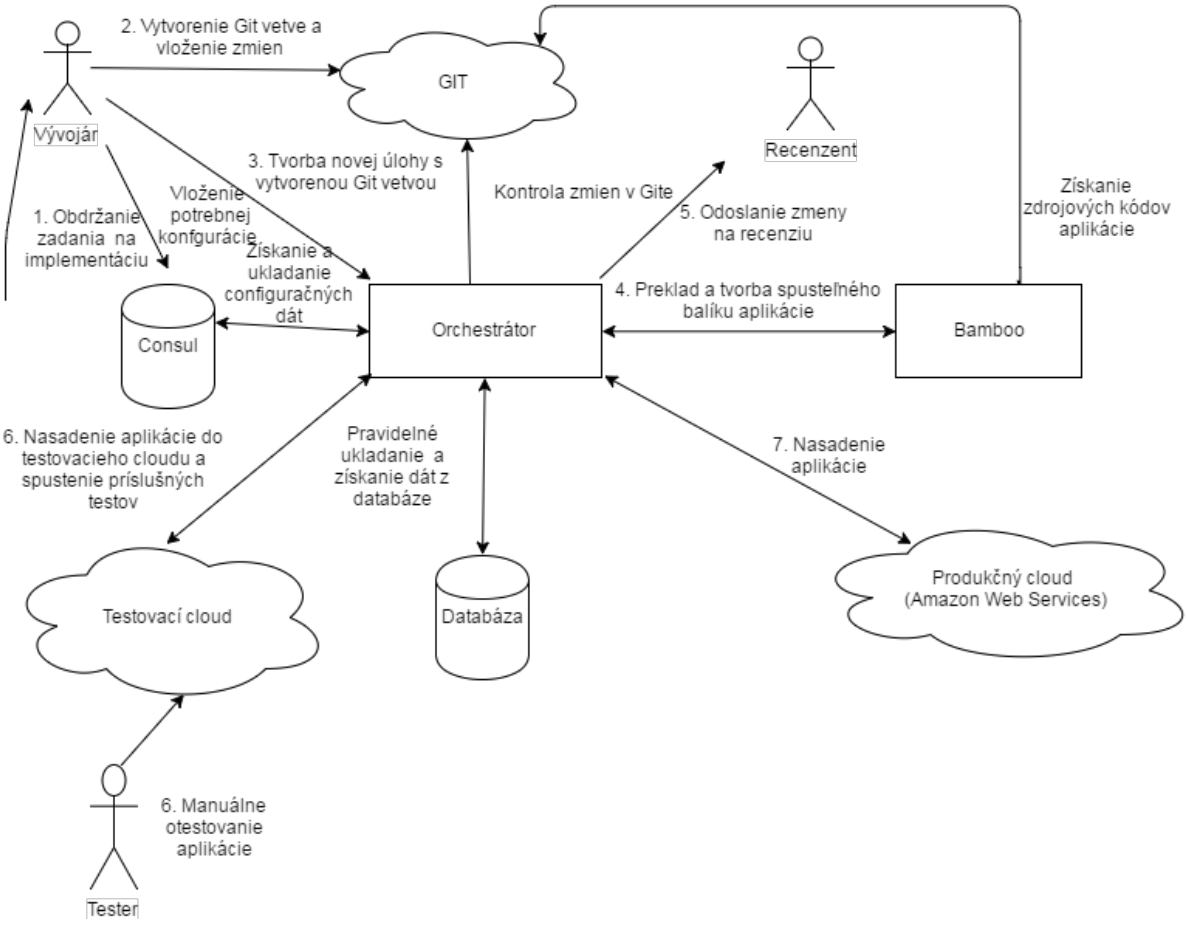
Obr. 4.1: Databázový návrh aplikácie Orchestrátor

Celý databázový model je zložený z deviatich databázových tabuliek. Postupne prejdem jednotlivé tabuľky s krátkym vysvetlením ich návrhu. Tabuľka *user_role* je vytvorená za účelom zadefinovania užívateľských rolí pre každého užívateľa. Z pohľadu práce s toutou tabuľkou ide o katalóg, ktorá je pri vytvorení naplnená štyrmi užívateľskými rolami: vývojár, auditor, administrátor a vydávateľ. Na túto tabuľku odkazuje tabuľka *user*, ktorá reprezentuje užívateľa, ktorý je jednoznačne identifikovaný užívateľským emailom. Ako bolo spomenuté táto tabuľka odkazuje na tabuľku s užívateľskými rolami. Každý užívateľ pri prvotnom prihlásení neobsahuje užívateľské role. Preto je nutné, aby mu niekto tieto role priradil. Role sú priradzované administrátorom. V ďalšej časti prejdem na tabuľku *application (aplikácia)*, ktorá reprezentuje aplikáciu na, ktorej bude realizovať testovanie. Táto aplikácia je jednoznačne identifikovaná menom aplikácie, url do Git-u a jazykom, v ktorom je vyvinutá a typ použitej databáze v prípade, že nejakú používa. Tabuľka obsahuje Bamboo plán, ktorý realizuje tvorbu release balíčku (plan_key) a identifikátor vydávacieho projektu na produkčné a predprodukčné prostredia (project_id). S touto tabuľkou je vo vzťahu tabuľka *úloha (task)*. Táto tabuľka uchováva informácie o aktuálne vyvíjaných funkcionalitách. Každá úloha má svoj názov. Rovnako obsahuje názov vetvy v Git-e, na ktorej je vyvíjaná. Rovnako obsahuje informácie o tom, či je možné aplikáciu automaticky testovať. To znamená, že pri zmene v Git-e dôjde k jej otestovaniu na testovacom serveri. Tabuľka obsahuje ešte informáciu o tom, kto danú úlohu vytvoril, stav úlohy, čas poslednej zmeny, kolekciu a prostredie pre nástroj Newman a cudzí kľúč do tabuľky aplikácie.

Databázový návrh obsahuje dalej tabuľku *bamboo_build*. Táto tabuľka obsahuje informácie o čísle prebiehajúceho release buildu aplikácie a stave tohoto buildu a cudzí kľúc do tabuľky úloha. Dôvod tejto tabuľky je nutnosť kontroly stavu prebiehajúceho release buildu. Ďalšou dôležitou tabuľkou je tabuľka *audit_log*, ktorá uchováva informácie o prebiehajúcich akciách na frontendovej časti aplikácie, rovnako aj na strane Orchestrátora, ktorý realizuje spúšťanie rôznych Bamboo plánov. Tabuľka obsahuje správu (message), autora zmeny a časové razítko. Ďalšou pomocnou tabuľkou je tabuľka *availability*, ktorá uchováva informácie o tom, že nejaký testovací server je práve používaný a nie je možné ho použiť na iný typ testovania. Obsahuje väzbu na tabuľku úloha.

Tabuľka *bamboo_test_build* predstavuje požiadavku na testovanie, či už manuálne alebo automatizované. Táto požiadavka v prípade manuálne testovania vznikne vyvolaním akcie z frontendovej časti aplikácie. Požiadavka na automatizované testovanie je vytvorená v prípade, že došlo k zmene v úlohe, ktorá je definovaná vetvou v Git-e. Tieto požiadavky sú spracovavané vláknom, ktoré sa stará o ich dokončenie. Rozlíšenie, že sa jedná o automatizované alebo manuálne testovanie je prostredníctvom atribútu auto_test_flag. Tabuľka obsahuje ďalej číslo buildu, ktoré získa pri spustení Bamboo plánu pre vydávanie na testovací server a stave testovacej požiadavky. Poslednou databázou tabuľkou je tabuľka *bamboo_deployment*. Táto tabuľka obsahuje informácie o spustení nasadenia releasu balíčku na predprodukčné alebo produkčné prostredie. Tabuľka obsahuje informácie o stave tohoto nasadenia, rovnako obsahuje atribúty pre smoke kolekciu a prostredie pre nástroj Newman.

4.4 Vysvetlenie prototypu



Obr. 4.2: Prototyp

Na obrázku č. 4.2 je zobrazený celý prototyp, ktorého implementácia bude realizovaná. Celý tento prototyp je zložený z troch aplikácií, ktoré budú výsledkom práce. Jedna aplikácia predstavuje samostatne aplikáciu skladajúcu sa z frontendovej a backendovej časti a aplikácie, ktorá sa nachádza priamo v cloude (CCU) a rieši vydávanie a smoke otestovanie aplikácie priamo v cloudovom prostredí. Pre ukladanie konfigurácie sa použije nástroj Consul, do ktorého bude uložená konfigurácia aplikácie spolu s konfiguráciou testovacích serverov. Pre zostavovanie a vydávanie aplikácie je použitý nástroj Bamboo.

Celý priebeh, ktorým je realizovaný vývoj pokračujúc testovaním až nakoniec vydaním aplikácie do produkcie je na obrázku očíslovaný krokmi .

Celý proces automatizovaného testovania a vydávania začína pri vývojárovi, ktorý obdrží zadanie na vývoj nejakej konkrétnej funkčnosti. Tento vývoj sa začína vytvorením vetvy v nejakom známom SCM systéme, napríklad Git. Všetku vyvinutú funkcionalitu vkladá priebežne do tejto vetve, pričom dodržuje Git flow [13]. Po tvorbe vetvy s SCM systéme vývojár zvolí prostredníctvom užívateľského rozhrania vetvu, kde bude funkcionalita vyvíjaná a bude teda aj následne automatizovaná z pohľadu testovania v prípade zvolenia tejto možnosti. Rovnako je potrebné k tejto vetve nahrať prostredníctvom aplikácie

potrebné funkčné testy vytvorené cez nástroj Postman. Aplikácia bude automaticky kontrolovať zmeny v zdrojovom kóde, ktoré boli nahraté do SCM. V prípade, že dôjde k detekcii zmeny v zdrojovom kóde (vloženiu zmien v danej vetvy do SCM), dôjde následne k automatickému zostaveniu aplikácie. Aplikácia sa nasadí na 1 z voľných a vhodných serverov pre danú aplikáciu. Aplikácia sa otestuje cez špeciálny Bamboo plán pričom sa ešte predtým nahrá do Git repozitára sada testov odkiaľ tento Bamboo plán tieto testy berie pre nástroj Newman. Aplikácia zozbiera výsledok tohoto plánu a výsledok je oznámený vývojárovi cez webové užívateľské rozhranie. Rovnako je možné úlohu otestovať ručne.

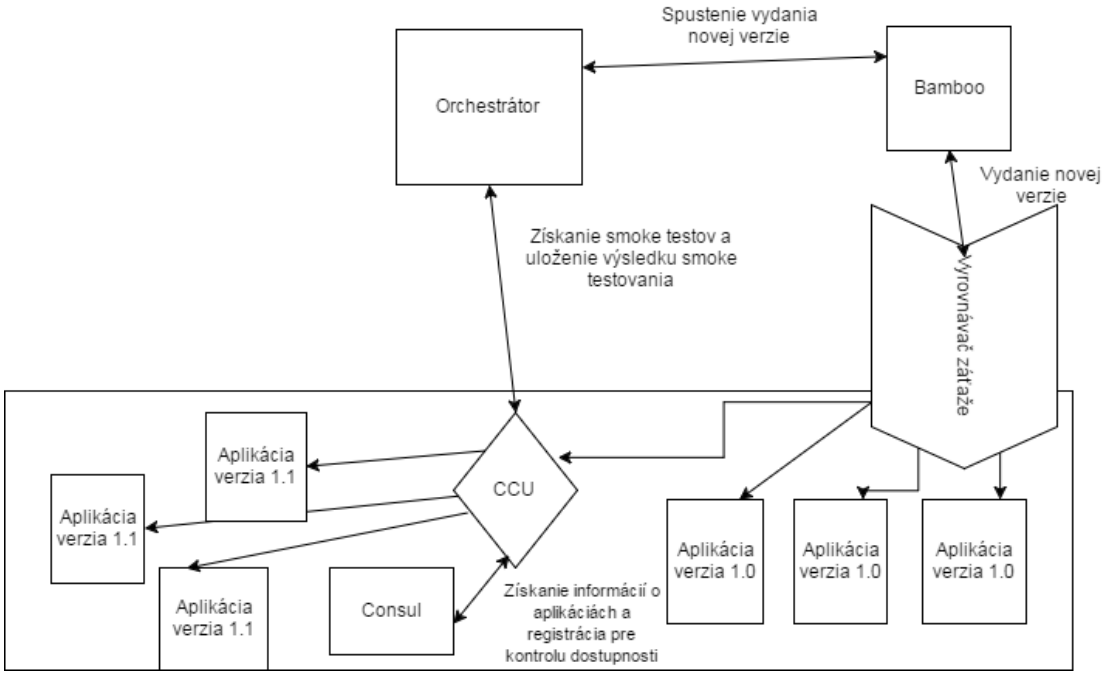
Celá aplikácia ja nasadená na testovacie cloudové prostredie. Všetky nastavenia ohľadne adries testovacích serverov su nahraté do nástroja Consul ešte pred samostatným používaním aplikácie odkiaľ si to vyberá Orchestrátor. Rovnako sa Orchestrátor stará a overuje, či sú testovacie servre dostupné. Rovnako sa aplikácia stará a vie, na ktorom momentálne testovacom prostredí prebieha testovanie. Po nasadení aplikácie do testovacieho prostredia a spustení testov je výsledok oznámený vývojárovi. Pokiaľ je vývojár hotový s vývojom a automatizované testy prebehli úspešne, tak vývojár pošle svoj kód na revíziu nejakému vývojárovi. Ten dostane notifikáciu o tom, že by mal realizovať revíziu kódu. Pokiaľ je revízia neúspešná vývojar dostane o tom informáciu a musí chyby opraviť a celý proces zostavenia a testovania sa opakuje a následne opäť vývojár schvaľuje funkcionalitu. Ako bolo spomenuté revízia prebieha oddelene v inom nástroji ako je napríklad Bitbucker, Github, ať. Po schválení kódu prebieha ak je to vyžadované nakoniec ešte tzv. manuálne testovanie kedy vývojár ručne testuje rôzne scenáre, ktoré neboli postihnuté v automatických testoch.

Po schválení funkcionality aj testerom je vytvorený tzv. vydávací balíček, ktorý je pripravený na vydávanie do predprodukčného alebo produkčného prostredia.

V prípade obdržania požiadavku na nasadenie danej funkcionality na dané prostredie realizuje sa cez webové užívateľské rozhranie nasadenie balíčku na príslušne prostredie. Užívateľ zvolí ešte smoke testy na otestovanie pred spustením novej verzie aplikácie.

Pre vydávanie aplikácie do produkčného prostredia aplikácia Orchestrátor zakomunikuje s nástrojom Bamboo. Následne je nutné ručne spustiť vydávanie z Bamboo. Nástroj Bamboo zabezpečení nasadenie aplikácie na príslušné servre v rámci cloudu Amazon Web Services. Pri nasadení sú kontaktované nástrojom pre nasadenie CodeDeploy (kapitola 4.5.2), ktorý nasadí verziu aplikácie na server. Po nasadení kontaktuje aplikáciu CCU, ktorá stiahne smoke testy pre aplikáciu a otestuje ich cez nástroj Newman server po serveri. Po úspešnom nasadení je vydávanie novej verzie dokončené a dochádza k prepnutiu na nové verzie aplikácie a výsledok je zapísaný do aplikácie Orchestrátor, ktorá výsledok zobrazí vo webovom užívateľskom rozhraní.

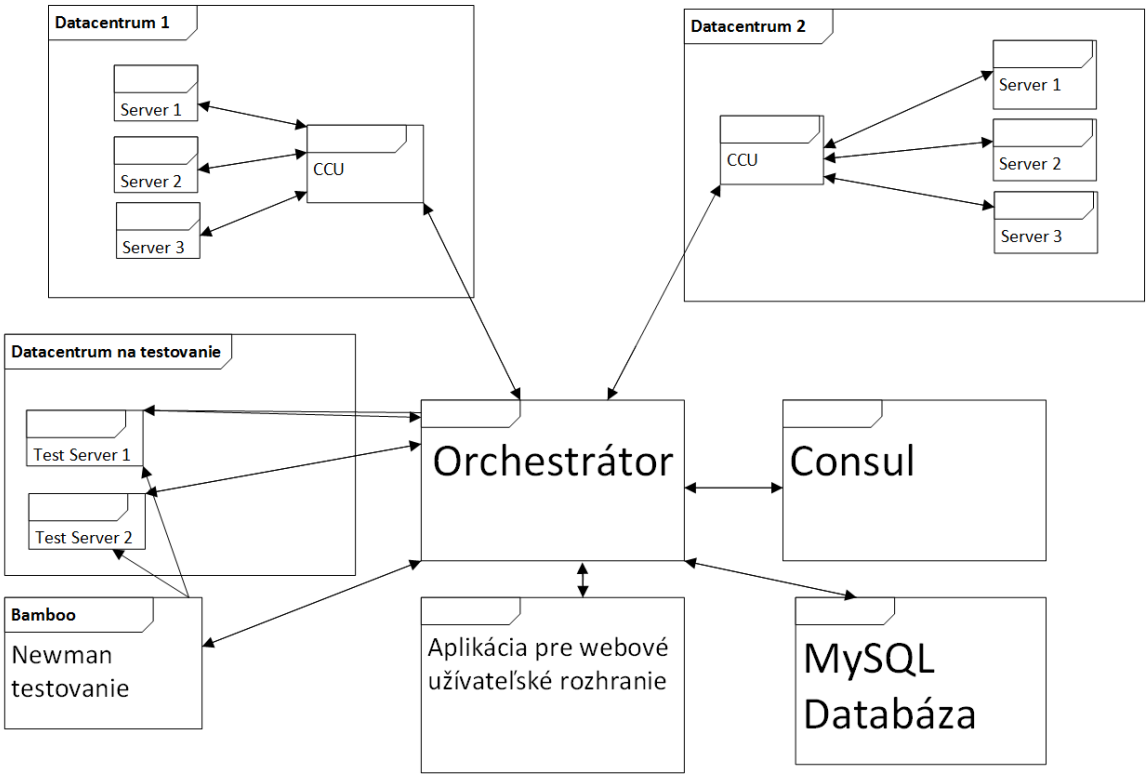
Je dôležité zdôrazniť, že v produkčnom prostredí typicky beží aplikácia vo viacerých inštanciách (virtuálnych serveroch). Je to dané tým, aby bola naša aplikácia dostupná aj pri výpadku jednej inštancie. Pred všetkymi serverami je predradený vyrovnávač záťaže, do ktorého sú registrované všetky servre pre danú aplikáciu a ten odosiela/prijíma prevádzku z/do vonkajšieho sveta. Zároveň zabezpečuje rozkladanie záťaže naprieč jednotlivými inštanciami.



Obr. 4.3: Produkčné vydanie aplikácie

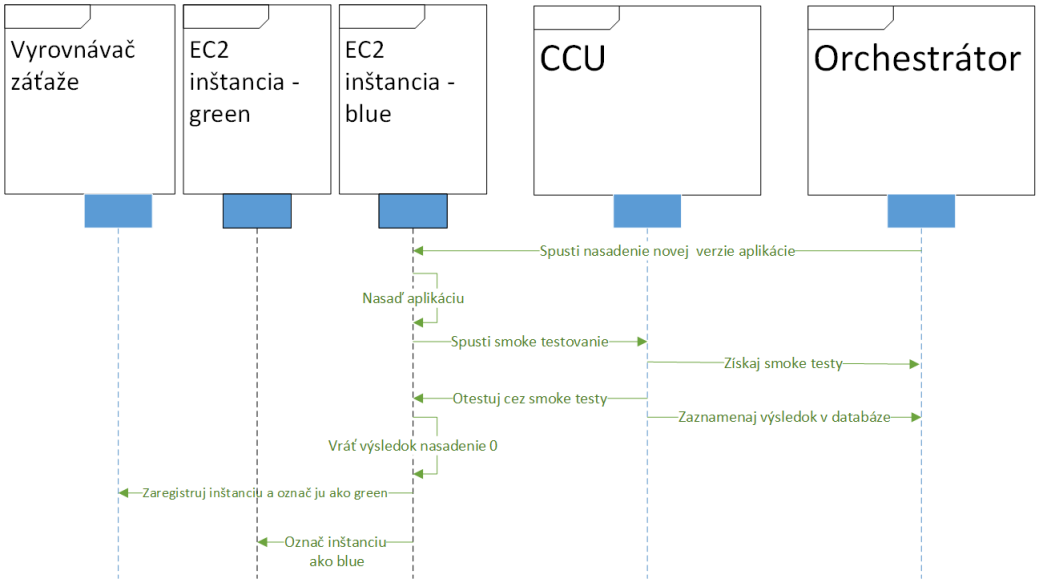
Na obrázku č. 4.3 je zobrazená detailne časť aplikácie v produkčnom prostredí. Základným predpokladom pre smoke testovanie je prítomnosť inštancie (virtuálneho servera), na ktorej beží aplikácia CCU spolu s Consul klientom. Rovnako je potrebné, aby na ďalšom serveri bežal Consul Server, ktorý je spojený s Consulom, na ktorom beží webové rozhranie odkiaľ sa berie/ukladá konfigurácia. Pri spojení týchto Consulov je možné získať konfiguráciu zavolaním aj iného Cosulu, keďže požiadavky sú potom presmerované na ten správny Consul.

Aplikácia CCU po začatí vydávania novej verzie aplikácie overí verziu aplikácie prostredníctvom smoke testov. Tieto testy získa z Orchestrátora a následne ich uloží k sebe a spustí ich cez nástroj Newman. Rovnako aplikácia CCU registruje jednotlivé inštancie, ktoré sú testované cez nástroj. Následne je možné vo webovom rozhraní Consulu vidieť jednotlivé inštancie a je možné sledovať ich stav. Po otestovaní jednotlivých inštancií dôjde k zapísaniu výsledku do Orchestrátora a je prepnutá vo vyrovnávači záťaže novšia verzia aplikácie. V prípade, že smoke testovanie zlyhá aspoň na 1 inštancii celé vydávanie zlyhá a nedôjde k prepnutiu na novú sadu serverov. A aplikácia stále navovok beží v starej verzii.



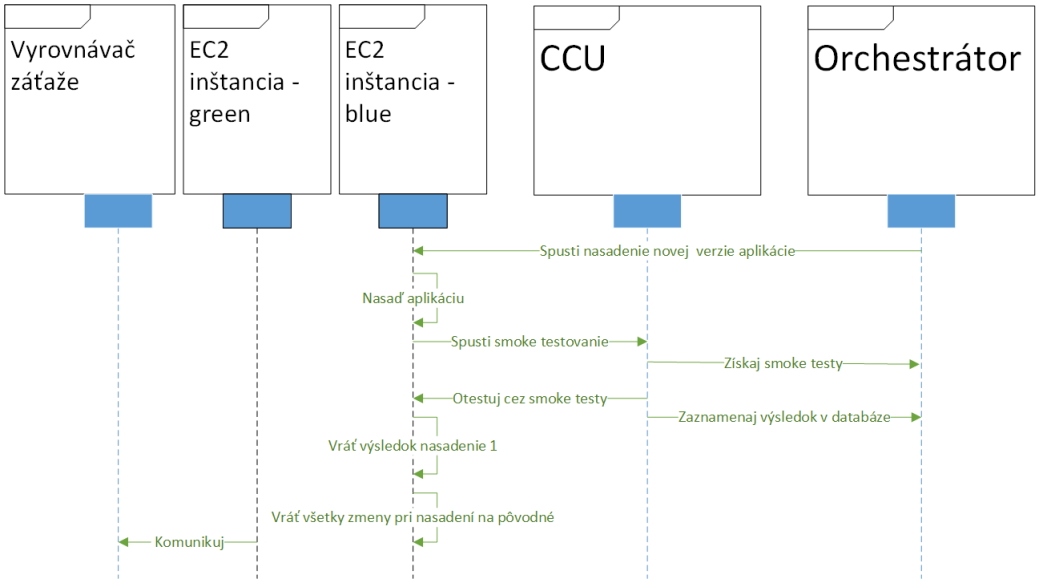
Obr. 4.4: Komunikácia ochestrátor a CCU

Na obrázku č. 4.4 je zobrazený bližší pohľad na vyššie popísaný prototyp. Prototyp ukazuje komunikáciu medzi aplikáciou CCU, Orchestrátor a webovou aplikáciou. CCU má zjednodušene za úlohu tesne pred spustením novej verzie aplikácie otestovať aplikáciu a následne zaradiť aplikáciu do reálnej prevádzky zaregistrovaním do vyrovnávača záťaže. Výsledok je oznámený aplikácií Orchestrátor. Na obrázku vidíme, že aplikácia CCU testuje viaceré servre. Typicky máme viacero serverov kvôli zvládnutiu vyššej záťaže na aplikáciu. V spodnej časti obrázku je aplikácia pre webové užívateľské rozhranie. Aplikácia komunikuje s aplikáciou Orchestátor. Táto aplikácia rovnako ovláda nástroj Bamboo, ktorý má zá úlohu preklad a vydávanie aplikácií. V pravej časti obrázku sa nachadza nástroj Consul, pričom tento nástroj slúži na ukladanie konfigurácie pre aplikáciu. Konfigurácia obsahuje nastavenie špecifické pre aplikáciu, tak isto zoznam dostupných testovacích serverov.



Obr. 4.5: Sekvenčný diagram pre testovanie CCU

Na obrázku č. 4.5 je zobrazený jeden konkrétny prípad komunikácie aplikácie CCU. Na obrázku je možné vidieť päť entít vyrovnávač záťaže, aplikácia, CCU, EC2 inštancia označenú ako blue a EC2 inštanciu označenú ako green. Označenie green a blue je vysvetlené v kapitole 2.1. Entita aplikácia je práve nasadzovaná na príslušné servre v rámci cloudu Amazon Web Services. Celé to vydávanie začína nahraním novej verzie aplikaćie na blue EC2 inštanciu. Po nasadení na inštanciu sa v istej fáze kontaktuje aplikácia CCU a zrealizuje smoke testy na tejto aplikácii cez sadu testov, ktoré si stiahne z aplikácie Orchestrátor. Nájdenie adresy tejto aplikácie prebieha cez Consul klienta, ktorý je spustený na tejto inštancii a ten konktuje Consul server odkiaľ túto adresu získa. Po úspešnom testovaním je aplikácia zaregistrovaná vo vyrovnávači záťaže a sú spustené klientské HTTP žiadosti z vonkajšieho sveta. Dokončenia nasadenia je realizované vrátením návratového kódu 0. Pôvodná blue inštancia sa stane green (aktuálnou) a stará inštancia green je označená ako blue a je zrušená. Typicky blue a green EC2 inštancií je viac, len pre jednoduchosť to bolo ukázané na jednej.

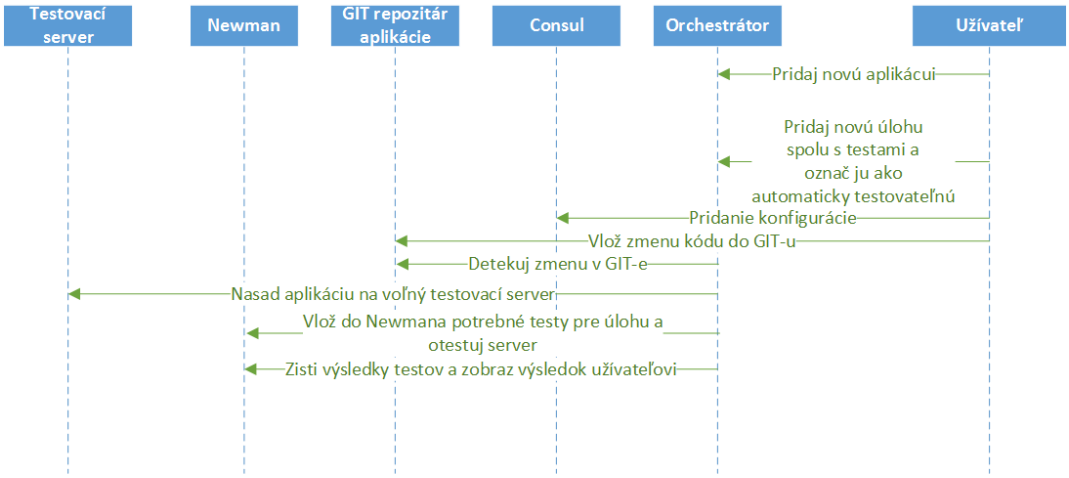


Obr. 4.6: Sekvenčný diagram pre aplikáciu CCU

Na obrázku č. 4.6 je zobrazený iný prípad komunikácie aplikácie CCU. Na obrázku je možné vidieť päť entít Orchestrátor, EC2 inštancia blue, EC2 inštancia green a CCU. Rovnako ako v predchádzajúcom prípade dochádza k vydávanie novej verzie aplikácie do produkčného prostredia a smoke testovanie aplikácie. V tomto prípade dôjde k zlyhaniu smoke testov. Preto sa aplikácia zabije a o výsledok je predaný aplikácií Orchéstrátor. V tomto prípade vytvorená EC2 blue inštancia je zrušená a návratový kód z nasadenia je 1. Typicky blue a green EC2 inštancií je viac, len pre jednoduchosť to bolo ukázané na jednej.

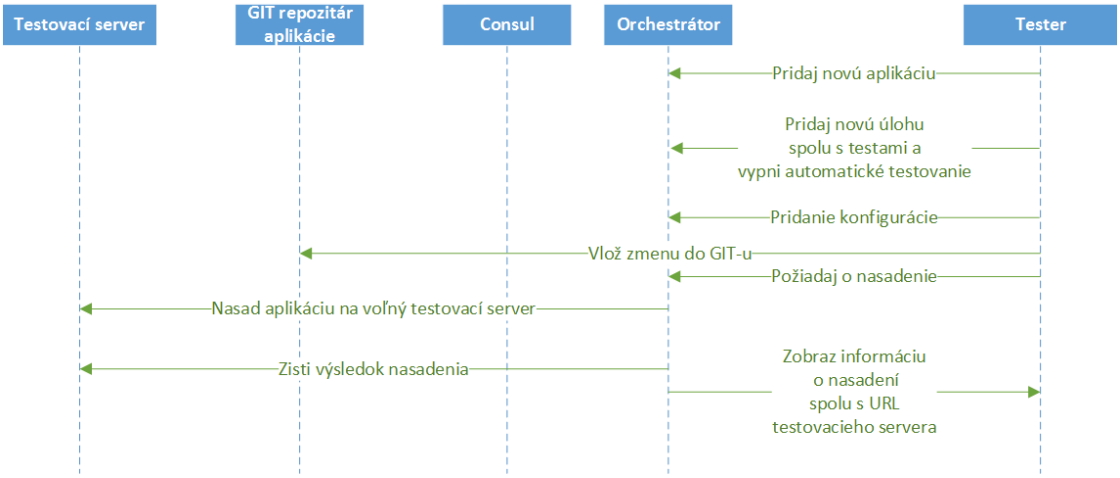
4.4.1 Ukážka typických použití aplikácie

V tejto podkapitole bude ukážka niekoľkých sekvenčných diagramov s cieľom ukázať celý postup akým spôsobom sa aplikácia testuje a následne nasadzuje do produkčného prostredia.



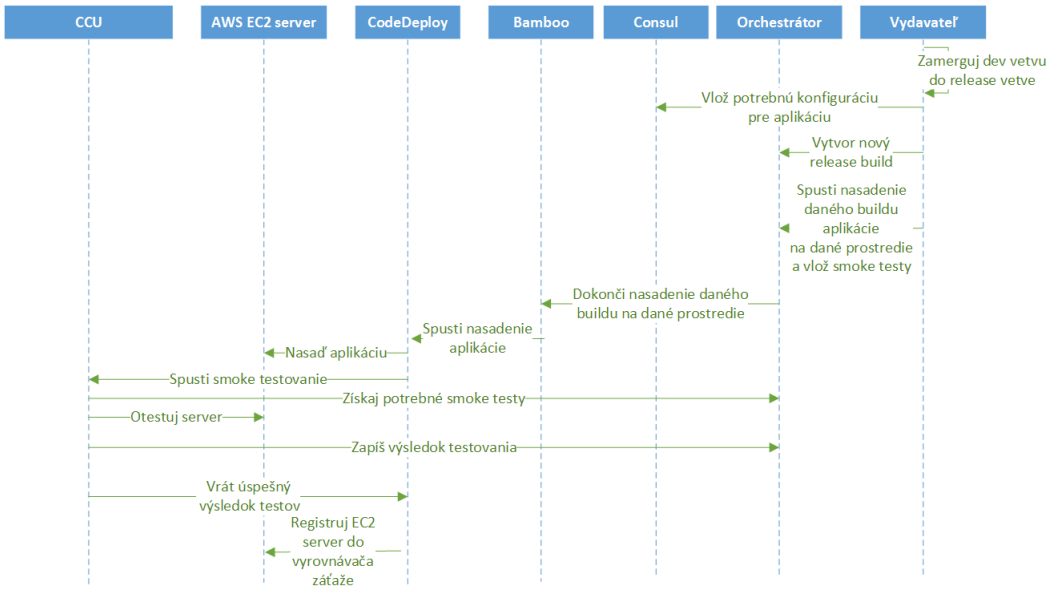
Obr. 4.7: Automatické testovanie aplikácie pri zmene

Na obrázku č. 4.7 je zobrazená ukážka automatického testovania aplikácie. Celý proces testovania začína pridaním aplikácie v užívateľskom rozhraní aplikácie, ktorá komunikuja s backendovou službou Orchestrátor. Následne vývojár zaškrtne možnosť, že si praje, aby bola úloha automaticky testovaná. Je nutné zdôrazniť, že vývojár si pred zvolením automatického testovania úlohy musí vytvoriť potrebnú vetvu v Git-e. Orchestrátor následne na pozadí automaticky kontroluje zmeny pre všetky úlohy, ktoré sa vzťahujú k jednotlivým Git vetvám a v prípade detekcie zmeny je realizované automatizované testovanie. Z testovacích serverov zadefinovaných v Consule sú vybrané všetky vhodné testovacie servre. Vhodným serverom sa myslí server, ktorý je určený pre rovnaký programovací jazyk ako aplikácia a v prípade potreby poskytuje pripojenie na databázu. Orchestrátor skontroluje dostupnosť server, tj. či nie je aktuálne používané pre manuálne testovanie a tým pádom je na ten server nahraná iná aplikácia. Pokiaľ je server obsadený dochádza k výberu ďalšieho vhodného servera a opäť sa skontroluje dostupnosť. V prípade, že je server dostupný Orchestrátor prekopíruje zdrojové kódy aplikácie do repozitára pre Bamboo plán pre daný server a spustí Bamboo plán. Celá konfigurácia k Git repozitáru vrátane prihlasovacieho mena a hesla a kľúča pre spúšťanie Bamboo plánu sa nachádza v Consule. Orchestrátor následne spustí Bamboo plán a čaká na výsledok, v prípade, že je úspešný tak pokračuje ďalej. Následne prekopíruje testy pre nástroj Newman do ďalšieho Git repozitára opäť sa celá konfigurácia nachádza v Consule. Následne dôjde k spusteniu Bamboo plánu pre Newman. Ten spustí tento nástroj a Orchestrátor následne zistí výsledok a vráti ho užívateľovi. Celý proces sa opakuje v prípade detekcie zmeny v Git repozitári.



Obr. 4.8: Manuálne testovanie aplikácie

Na obrázku č. 4.8 je ukážka použita prototypu pre manuálne testovanie aplikácie testerom. Celý proces opäť začína tým, že je pridaná nová aplikácia v prípade, že už neexistuje, tak je len vybraná pri pridávaní novej úlohy. Pri pridávaní novej úlohy vývojár zaškrtne možnosť, že nechce realizovať automatizované testovanie. Po pridaní úlohy je možné realizovať manuálne testovanie. Tester vyberie zo zoznamu úloh tú, o ktorú má záujem testovať a nechá úlohu nasadiť na voľný testovací server. Opäť Orchestrátor vyberie vhodný a voľný testovací server a vydá verziu aplikácie na server. Informácia o tom, na ktorý server je aplikácia nasadená sa nachádza priamo vedľa úlohy. Vývojar po otestovaní aplikácie prehlási, či je úloha v poriadku a spĺňa predpísané zadanie, alebo naopak testy zlyhali a je nutné úlolu dopracovať. Následne sa uvoľný testovací server a je možné ho použiť inde.



Obr. 4.9: vydávanie aplikácie vydavateľom

Na obrázku č. 4.9 je zobrazená ukážka prototypu pre vydávanie aplikácie do predprodukčného prostredia. Tento scenár je obecný a je platný pre aj pre vydávanie aplikácie do produkčného prostredia. Celý proces začína tým, že vydavateľ, čo je vlastne užívateľ, ktorý má pridané užívateľské oprávnenia pre vydávanie aplikácií. Vydavateľ v užívateľskom rozhraní zvolí verziu buildu, prostredie na ktoré bude build nasadený a smoke testy pre záverečné otestovanie aplikácie. Následne vydavateľ musí dokončiť vydávanie buildu na prostredie v nástroj Bamboo. Po spustení vydávania vydavateľ môže sledovať celý priebeh v užívateľskom rozhraní, ktoré komunikuje s Orchestrátorom, ktorý komunikuje s nástrojom Bamboo. Každopádne po spustení vydávania dôjde k spusteniu nástroja Bamboo, ktorý kontaktuje príslušného agenta CodeDeploy 4.5.2. Tento nástroj dostáva informáciu o tom, že treba nasadiť takú verziu aplikácie. Tento nástroj zistí konfiguráciu vydávania. Pre bez výpadkové vydávanie je realizované vydávania cez blue/green. To je realizované tak, že nástroj vytvorí nový zhodný počet čistých EC2 inštancií a nasadí aplikáciu na jednotlivé EC2 inštancie. Po nasadení aplikácie spustí smoke testovanie tj. je kontaktovaná aplikácia CCU špecifická pre dané datacentrum. Adresu opäť zistí z Consul servera Consul klient, ktorý beží na danej inštancii. Je nutné zdôrazniť, že spolu s aplikáciou beží na každej EC2 inštancii aj Consul klient. Dôvod je ten, že nám to umožňuje dynamicky získavať konfiguráciu prípadne ju meniť. Títo klienti kontaktujú vopred známe a už vytvoré Consul servre. Každopádne počas smoke testovania aplikácia CCU kontaktuje aplikáciu Orchestrátor a získa smoke testy. Tieto smoke testy uloží lokálne na disk a spustí nástroj newman, ktorý ako zdroj má uvedené nasledovné testy. CCU pri žiadosti o otestovanie obdržala IP adresy EC2 inštancií, kde bežia, aby mohla spustiť správne smoke testy. Po úspešnom otestovaní je výsledok uložený do Orchestrátora odkiaľ sa informácia zobrazuje priamo užívateľovi. Po úspešnom otestovaní je prepnutá prevádzka na novú sadu serverov a stará sada serverov so starou verziou aplikácie je odpojená a EC2 servre sú ukončené. V prípade zlyhania smoke testovania aspoň na 1 EC2 inštancii je celý proces vydávania ukončený a aplikácia beží stále v starej verzie a nová sada serverov je ukončená.

4.5 Amazon Web services

Služba Amazon Web Services (AWS) je dcérska spoločnosť spoločnosti Amazon.com, ktorá ponúka platformy cloud computingu. Tieto služby fungujú zo 16 zemepisných oblastí na celom svete. Zahŕňajú Amazon Elastic Compute Cloud, tiež známy ako EC2 a Amazon Simple Storage Service, tiež známy ako "S3". AWS má viac ako 70 služieb, ktoré zahŕňajú širokú škálu služieb vrátane výpočtu, ukladania, vytvárania sietí, databáz, analýz, aplikačných služieb, vydávania, správy, mobilných, vývojových nástrojov a nástrojov pre internet vecí.

4.5.1 EC2

Amazon Elastic Compute Cloud (Amazon EC2) poskytuje škálovateľnú výpočtovú kapacitu v cloude Amazon Web Services (AWS) [1]. Amazon EC2 je možné použiť na spustenie virtuálnych serverov, konfiguráciu zabezpečenia a vytváranie sietí a správu ukladania. Amazon EC2 poskytuje nasledujúce funkcie:

- Virtuálne počítačové prostredia, známe ako inštancie
- Predkonfigurované šablóny pre inštancie, známe ako Amazon Machine Images (AMI), ktoré obsahujú balíky vrátane operačného systému a ďalšieho softvéru
- Rôzne konfigurácie kapacity procesoru, pamäte, úložiska a siete pre inštancie
- Zabezpečenie prihlásenie pre inštancie pomocou párov kľúčov (AWS ukladá verejný kľúč a súkromný kľúč na bezpečnom mieste)
- Objemy ukladacieho priestoru pre dočasné údaje, ktoré sú vymazané pri zastavení alebo ukončení inštancie
- Viaceré fyzické umiestnenia zdrojov
- Bránu firewall, ktorá umožňuje špecifikovať protokoly, porty a rozsahy zdrojov IP
- Statické adresy IPv4 pre inštancie cloud computing
- Virtuálne siete prostredníctvom, ktorých je možné vytvoriť logické celky izolované od zvyšku cloudu AWS

4.5.2 CodeDeploy

AWS CodeDeploy je služba pre vydávanie aplikácie do cloudu AWS, ktorá automatizuje vydávanie aplikácií na inštancie Amazon EC2 alebo inštancie vo vlastnom datacentre. Podporuje vydávanie takmer neobmedzeného rozsahu a obsahu aplikácií, ako sú kódy v rôznych programovacích jazykov, webové a konfiguračné súbory, spustiteľné súbory, balíky, skripty, multimediálne súbory ....

AWS CodeDeploy uľahčuje uvoľňovanie nových funkcionalít, pomáha sa vyhnúť prestojom počas vydávania aplikácií a zvláda náročnosť aktualizácií aplikácií bez mnohých rizík spojených s ručným vydaním, ktoré sú náchylné na chyby. Služba rovnako zabezpečuje jednoduché vydávanie na viaceré EC2 inštancie. AWS CodeDeploy pracuje s rôznymi systémami na riadenie konfigurácie, riadenie zdrojov, nepretržitú integráciu, nepretržitú dodávku a nepretržité vydávanie.

Tento nástroj ponúka nasledovné výhody:

- Automatizované vydávanie. AWS CodeDeploy plne automatizuje vydávanie aplikácií vo vývojovom, testovacom a produkčnom prostredí.
- Minimalizuje prestoje. AWS CodeDeploy maximalizuje dostupnosť aplikácií
- Umožňuje zastaviť vydávanie a vrátiť všetky nové zmeny v aplikách späť [1]. Ak sa vyskytnú chyby je možné automaticky alebo ručne zastaviť a vrátiť naspäť vydávania.
- Centralizovaná kontrola. Stav nasadení môže byť spustený a sledovaný prostredníctvom konzoly AWS CodeDeploy

AWS CodeDeploy poskytuje dve možnosti typu vydávania:

- Inštalácia na mieste : Aplikácia na každej inštancii v skupine vydávania je zastavená, nainštalovaná najnovšia verzia aplikácie a spustená a overená nová verzia aplikácie.
- Blue-green inštalácia: Je vytvorená nová skupiná EC2 inštancií, na ktoré sú nainštalované najnovšie verzie aplikácií a po otestovaní sú EC2 inštancie zaradené do prevádzky. Inštancie so starou verziou aplikáciu sú zrušené.

Dôležitou súčasťou aplikácie, aby mohla byť nasadená prostredníctvom nástroja AWS CodeDeploy do cloudu AWS je súbor špecifikácie aplikácie (súbor AppSpec). Súbor je vo formáte YAML a slúži na:

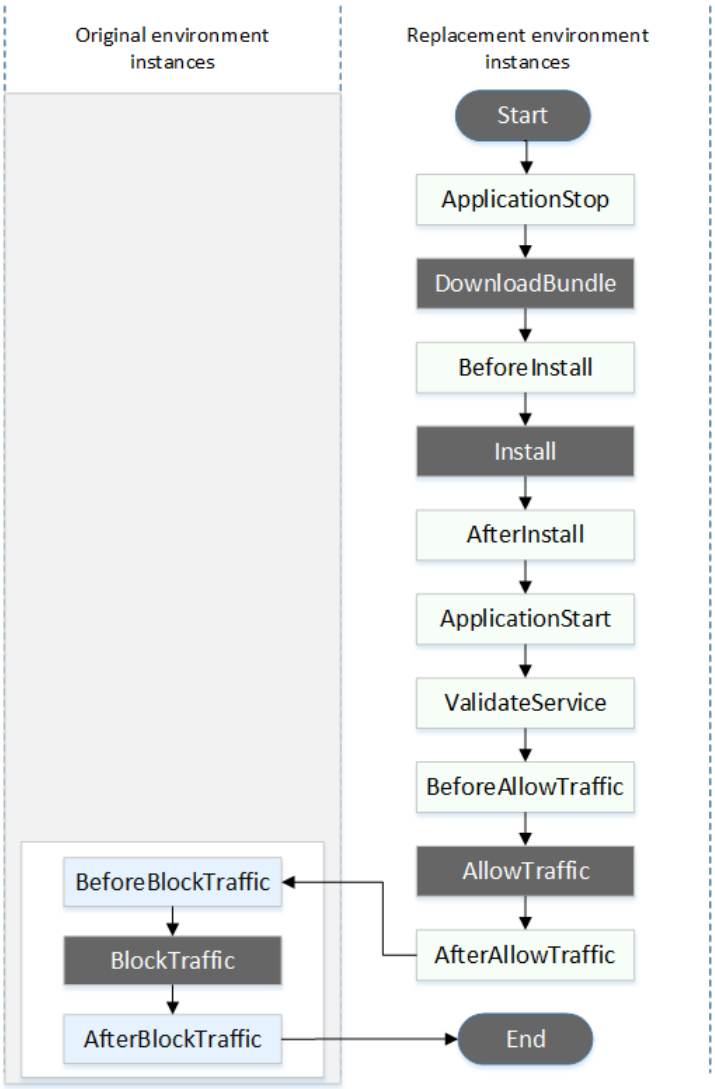
- Definuje oprávnenia pre spúšťanie nasadzovaných súborov
- Určuje skripty, ktoré sa majú spustiť na každej inštancii v rôznych fázach procesu zavádzania.

Súbor AppSpec sa používa na zvládnutie každého vydávania ako série udalostí životného cyklu. Udalosti životného cyklu, ktoré sú definované v súbore, umožňujú spustiť skripty na inštancii pri istých udalostiach životného cyklu jednotlivého vydávania, ktorými prechádza nasadenie aplikácie. AWS CodeDeploy spúšťa iba tie skripty špecifikované v súbore, ale zároveň tieto skripty môžu v inštancii pri nasadzovaní zavolať ďalšie skripty. Môže byť spustený ľubovoľný typ skriptu, ak ho podporuje operačný systém spúšťaný v inštanciách. Počas vydávania agent AWS CodeDeploy vyhľadá názov aktuálnej udalosti v sekcii „hooks“ v súbore AppSpec. Ak sa tam udalosť nenachádza, agent AWS CodeDeploy prejde na ďalší krok. Ak sa udalosť zistí, agent AWS CodeDeploy načíta zoznam skriptov, ktoré sa majú spustiť. Skripty sa spúšťajú postupne v poradí, v akom sa nachádzajú v súbore. Stav každého skriptu je zaznamenaný v súbore protokolu agentov AWS CodeDeploy na inštancii. Ak sa skript úspešne spustí, vráti kód ukončenia 0 (nula) [1]. V prípade, že daná udalosť vráti chybový kód 1 celé vydávanie je prerušené. V prípade vydávanie Blue/green je celý proces vydávania vrátený a aplikácia beží stále v starej verzii.

```
version: 0.0
os: operating-system-name
files:
  source-destination-files-mappings
permissions:
  permissions-specifications
hooks:
  deployment-lifecycle-event-mappings
```

V ukážke uvedenej vyššie zobrazená ukážka súboru AppSec. V tejto štruktúre:

- Version - Táto časť špecifikuje verziu súboru AppSpec.
- Os - Táto časť určuje hodnotu operačného systému inštancie, na ktorú sa vydáva verzia aplikácie. Je možné špecifikovať nasledujúce hodnoty: Linux - Inštancia je server Amazon Linux, Ubuntu Server alebo RHEL. Windows - Inštancia je inštancia systému Windows Server.
- Files - Táto časť špecifikuje názvy súborov, ktoré by mali byť skopírované do inštancie počas inštalačnej udalosti inštalácie.
- Permissions - Táto časť špecifikuje, ako by sa mali špeciálne povolenia, ak nejaké, aplikovať na súbory v sekcii súbory pri ich kopírovaní do inštancie. Táto časť sa týka iba inštancií Amazon Linux, Ubuntu Server a Red Hat Enterprise Linux (RHEL).
- Hooks - Táto časť špecifikuje skripty, ktoré sa majú spustiť počas určitých udalostí životného cyklu zavádzania počas vydávania.



Obr. 4.10: Životný cyklus blue/green vydávania aplikácie spolu s udalosťami v cloude AWS

Na obrázku č. 4.10 je zobrazná ukážka životného cyklu aplikácie počas vydávanie do produkciu spolu s jednotlivými udalosťami, ktoré boli spomínané vyššie. Jednotlivé udalosti prebiehajú postupne a dochádza pri nich k spúšťaniu skriptov v prípade, že sú definové v súbore AppSec.

4.6 Nástroje

V tejto kapitole bude predstavený stručný prehľad technológií, ktoré budú použité pri realizácií prototypu. Cieľom je teda ukázať, ze prototyp je univerzálny a je možné použiť ľubovolné nástroje podľa konkrétnych potrieb organizácie rovnako aj z pohľadu použitého programovacieho jazyka.

4.6.1 SCM

Táto skratka pochádza zo slov „source control management“, tj. riadenie zdrojových kódov. Tento pojem predstavuje nejaké centrálne úložisko zdrojových kódov aplikácie, ktoré sú používané pre zostavovanie konkrétnej spustiteľnej verzie. Toto úložisko je obvykle centralizované a jeho základným rysom, že umožňuje vkladanie zmien do tohoto úložiska od viacerých ľudí, tj od celého tímu. Medzi typických predstaviteľov môžeme zaradiť určite systém Git, SVN, Mercurial, ...

SCM je softvér, ktorý zabezpečuje koordináciu a služby medzi členmi vývojového softvérového tímu. Na tej najzákladnejšej úrovni poskytuje správu súborov a riadenie verzií tak, aby členovia tímu si neprepisovali navzájom zmeny. Projekty využívajúce nejaké SCM umožňujú vývojárom súčasne pracovať so súbormi (vo vetvách), zlučovať zmeny spojené so zmenami iných vývojárov a auditovať zmeny, ktoré boli vytvorené. V niektorých prípadoch, SCM môže zahŕňať aj iné zložky, ktoré pomáhajú pri riadení procesu softvéru počas celého životného cyklu.

4.6.2 Zostavovanie aplikácie

Ďalším krokom pri vytvorení cieľovej aplikácie je zostavenie celej aplikácie zo zdrojových kódov, tak aby bola spustiteľná a spĺňala nami požadované špecifikácie.

Tento krok je dôležitý pre vytvorenie softvéru a súvisiacich procesov na automatizáciu. Tieto procesy zahŕňajú: zostavovanie zdrojových kódov do binárneho kódu, zabalenie binárneho kódu do nejakého formátu.

Historicky bol preklad a zostavenie aplikácie realizovaný cez zostavovacie súbory nazývané „makefiles“ súbory. V súčasnosti existujú dve všeobecné kategórie nástrojov: Nástroje, ktoré zostavujú aplikáciu a môžu byť použité pre automatizáciu (ako je make, Ant, MS build, ...). Ich primárnym cieľom je generovať zostavenie artefaktov prostredníctvom činností, ako kompilácia a prepájanie zdrojových kódov.

Ako nadstavba pre zostavovanie aplikácií sú dnes používané automatizačné servre. Tieto nástroja používajú všetky nástroja popísané vyššie. Ich výhodou je, že ide typicky o webové aplikácie, takže práca s nimi je oveľa pohodlnejšia. Tieto automatizačné servre typicky realizujú viacero úkonov a predstavujú komplexnejšiu podporu pri zostavovaní aplikácii. Podporujú napríklad stiahnute zdrojových kódov z verzovacieho servera, preklad, spustenie jednotkových testov, po prípade aj nejakých funkčných testov spolu nástrojom na statickú alebo dynamickú analýzu.

4.6.3 Cloud computing

Cloud computing je model vývoja a používania IT technológií[6]. Cloud computing je charakterizovaný ako poskytnutie služieb uložených na serveroch tým, že užívatelia k nim pristupujú napríklad cez webový prehliadač alebo klientskú aplikáciu. Princíp cloud computingu je taký, že užívateľ neplatí za program, ale za jeho využitia. Aplikácie ponúkané cez cloud

computing sú rôzne od operačných systémov, cez databázové systémy až po distribuované systémy.

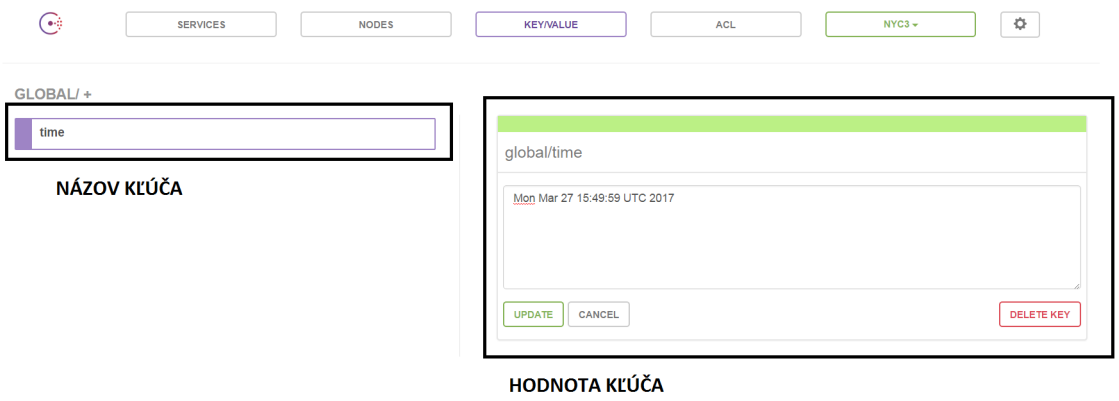
Medzi typických predstaviteľov cloud computingu patrí Amazon Web Services, Microsoft Azure, Google Cloud, ...

4.6.4 Consul

Consul je nástroj pre objavovanie a konfiguráciu služby („Service discovery“). Poskytuje rovnako aj ďalších niekoľko kľúčových funkcií:

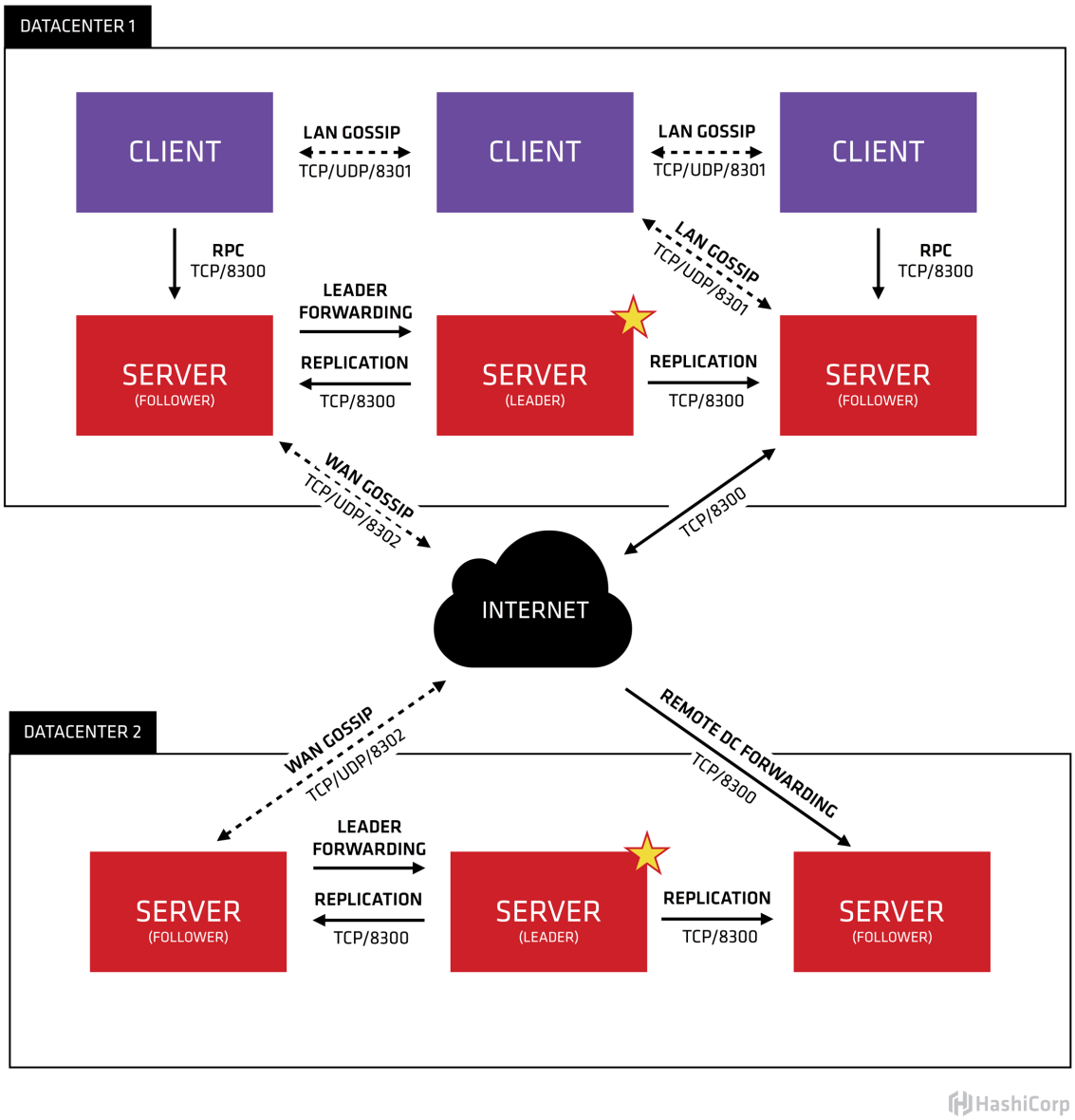
- Objavovanie služieb: Klienti Consula (aplikácie) môže poskytovať služby, ako je API alebo MySQL pripojenie, a títo klienti môžu použiť Consul na objavenie poskytovateľa služby. Nachádzanie služieb je realizované buď cez službu HTTP alebo cez službu DNS.
- Kontrola zdravia aplikácie (tzv. healthcheck): Consul umožňuje kontrolovať, či aplikácie, ktoré sú registrované odpovedajú a teda, či sú živé tj pracujú alebo došlo k ich výpadku. Táto funkcionalita je používaná objavovaním služby ku smerovanie prevádzky od nezdravých aplikácií
- Úložisko kľúč/hodnota: Aplikácie môžu využiť úložisko kľúč / hodnota pre ľubovoľný počet hodnôt vrátane uloženia dynamickej konfigurácie, ... Služba je realizované cez jednoduché HTTP API.
- Multi Datacenter: Consul podporuje služby popísané vyššie naprieč viacerými datacentrami. To znamená, že používatelia Consul-a sa nemusia starať o vybudovanie ďalšej vrstvy abstrakcie pri raste do viacerých oblastí.

Tento nástroj je realizovaný prostredníctvom agentov, ktorí sú inštalovaní priamo na konkrétnych aplikáciach a servera, ktorý zbiera správy od agentov. Consul je distribuovaný, vysoko dostupný systém. Každý uzol (inštacia servera), ktorý poskytuje služby pre Consul prevádzkuje Consul agenta. Spustenie agenta sa nevyžaduje pre objavovanie služby (service discovery) alebo získanie/uloženie dát v podobe kľúč/hodnota. Agent je zodpovedný za kontrolu zdravia služby (inak povedané, či služba beží) ako aj samotný uzol. Agenti komunikujú s jedným alebo viacerým serverami Consul [14]. Consul sú serveri, kde sú uložené údaje o agentov a replikácií [14]. Consul môže fungovať s jedným serverom, ale odporúča sa mať tri až päť, aby sa zabránilo zlyhaniu, ktoré vedú k strate dát.



Obr. 4.11: Ukážka nastavovania konfiguračných hodnôt pre aplikáciu

Na obrázku č. 4.11 je zobrazené grafické užívateľské rozhranie pre nástroj Consul. Konkrétne na tomto obrázku je zobrazená časť nástroja, ktorá slúži na zadanie konfiguračných záležitostí špecifických pre danú aplikáciu. Táto konfigurácia je pomerne jednoduchá spočíva v úložisku kľúč hodnota, ktoré už bolo spomínané. Názov kľúča môže byť ľubovoľný a je možné názov kľúča ľubovoľne zanorovať podľa potrieb aplikácie. Hodnota kľúča je pritom tiež ľubovoľná môže ísť napríklad o užívateľské heslo, meno, alebo cestu k súboru, alebo iná potrebná konfigurácia aplikácie.



Obr. 4.12: Ukážka architektúry nástroja Consul nasadeného naprieč datacentrami

Na obrázku č. 4.12 je zobrazená architektúra nástroja Consul, ktorý je nasadený naprieč viacerými datacentrami. Na obrázku môžeme vidieť existujúce dve dátové centrá označené ako jeden a dva. V každom dátovom centre máme zmes klientov a serverov. V každom dátovom centre sa nachádzajú tri až päť serverov. Tým sa dosiahne rovnováha medzi dostupnosťou v prípade zlyhania a výkonnosti. Neexistuje však obmedzenie počtu klientov a môžu sa ľahko rozdeliť na tisíce alebo desiatky tisíc. Všetky uzly (servre a klienti), ktoré sa nachádzajú v dátovom centre, sa podieľajú na protokole, ktorým vzájomné komunikujú.

Servery v každom dátovom centre sú súčasťou jedinej skupiny [14]. To znamená, že spolupracujú na výbere jediného vodcu, vybraného servera, ktorý má ďalšie povinnosti. Vedúci je zodpovedný za spracovanie všetkých žiadostí a transakcií od ostatných serverov a klientov. Uzly serverov tiež fungujú ako súčasť a komunikujú navzájom pomocou protokolu.

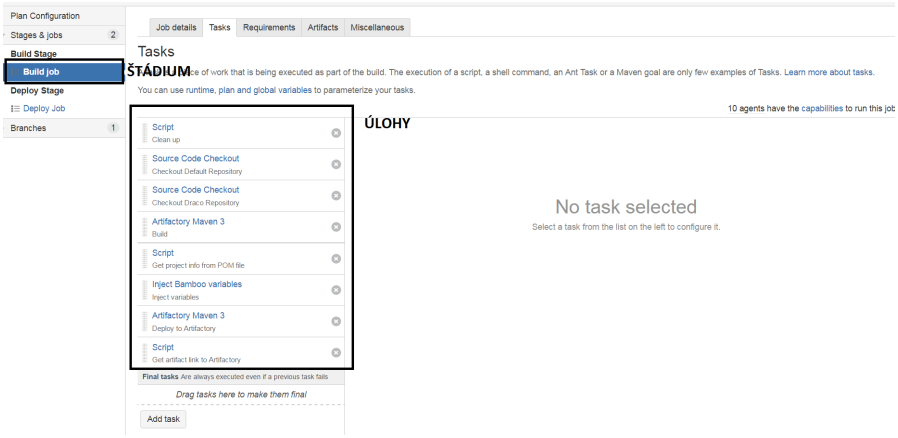
Výsledkom je veľmi nízka väzba medzi dátovými centrálami, ale kvôli zisťovaniu porúch, ukladaniu do vyrovnávacej pamäte pripojenia a multiplexingu, požiadavky na krížové dátové centrá sú relatívne rýchle a spoľahlivé.

4.6.5 Bamboo

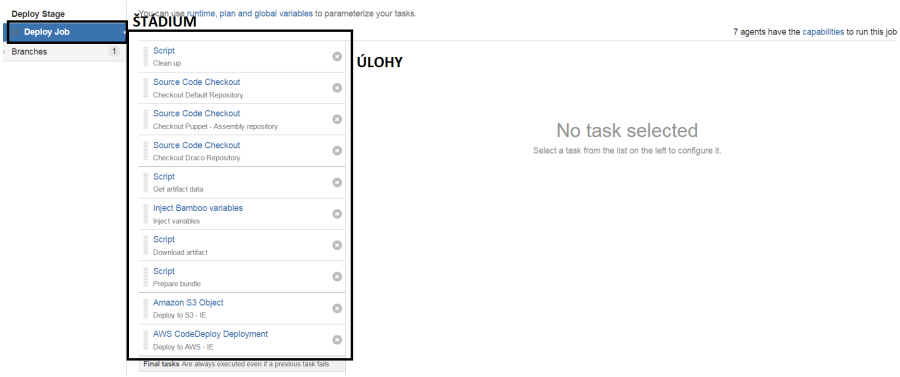
Bamboo je server pre kontinuálnu integráciu vyvinutý spoločnosťou Atlassian [12]. Tento nástroj poskytuje aj analýzu chýb pri zostavení spolu s trasovaním chyby [12]. Rovnako je možné tento nástroj nastaviť pre odosielanie rôznych typov notifikácií pri nastaní rôznych typov udalostí. Z pohľadu programátora poskytuje restové API pre získanie informácií o serveri, o prehľade všetkých plánov, o prehľade projektov a realizuje možnosť spustenia plánu. Bamboo sa integruje aj s niektorými softvérovými projektami napr: Bamboo poskytuje spojenie s Gitom, Mercurial, CVS, atď.. Bamboo pre svoju prácu používa sadu existujúcich nástrojov ako je Ant, Maven, ..., čo nám umožňuje použiť tento nástroj naprieč rôznymi programovacími jazykmi. Samozrejme pre podporu aplikácií je možné Bamboo rozšíriť prostredníctvom pluginov. Pluginu môžu napr. podporovať rozšírenie pre nástroj SonarQube (realizuje statickú analýzu programovacieho kódu), alebo môže ísť o plugin, ktorý umožňuje vydávanie aplikácie napr. do Windows Azure.

Spolu s podporovanými nástrojmi pre zostavenie aplikácie Bamboo disponuje aj testovacími nástrojmi ako je JUnit, Selenium, PHPUnit a iné. Tento nástroj je zároveň plne integrovaný do ďalších produktov ako je Jira, Confluence a iné. Toto prepojenie môžeme využiť pri sledovaním jednotlivých úloh z pohľadu managera, vývojára alebo testera.

Bamboo nástroj delí celú aplikácia podľa hierarchického poradia. Na najvyššom stupni hierarchie sa nachádza projekt, ktorý obsahuje aplikácia a združuje viacero plánov. Plány obsahujú celkovú konfiguráciu pre aplikáciu. Obsahuje informácie o zdrojovom repozitáry, spúšťačoch daného plánu (plán môže byť spustený manuálne alebo prostredníctvom externej akcie). Rovnako umožňuje nastaviť prístupové práva spolu s premennými daného plánu, ktoré je možné nastavovať dynamicky. Každý plán je typicky zložený z viacerých štádií (stage). Štádium si môžeme predstaviť ako zostavenie aplikácie a ďalšie štádium vydávanie aplikácie. Každé stádium sa skladá z úloh (jobov). Tieto úlohy môžu typicky bežať paralelne. Na najnižšom stupni hierarchie sa nachádza úloha (task). Úloha predstavuje najmenšiu možnú jednotku. Úlohy vykonávajú rôzne úlohy napr. púštanie bash skriptov, sťahovanie zdrojových kódov z repozitára, atď.



Obr. 4.13: Zostavovacie štádium



Obr. 4.14: Vydávacie štádium

Na obrázku č. 4.13 a č. 4.14 môžeme vidieť ukážkové nastavenie takéhoto Bamboo plánu pre aplikáciu, ktorá sa skladá z 2 štádií. Zostavovacie štádium, ktoré má za úlohu zostavenie aplikácie a nasadzovacie štádium, ktoré má za úlohu vydávanie aplikácie. Ako môžeme vidieť na obrázku č. 4.13 štádium sa skladá z viacerých úloh, ktoré bežia sekvenčne. Tieto úlohy postupne stiahnu zdrojové kódy aplikácie, zostavia aplikáciu a preloženú aplikáciu uložia do pomocnej zložky na server. Naopak na obrázku č. 4.13 môžeme vidieť, že rovnako predchádzajú obrázok aj tento predstavuje štádium reprezentované job-om. Ten sa skladá z viacerých úloh. Cieľom toho štádia je stiahnute preloženej aplikácie a jej vydávanie do prostredia Amazon Beb Services.

Ukážka štádií pre plán je vzorová a môže sa líšiť od konkrétnych požiadaviek aplikácie avšak koncept hierarchického rozdelenia je zachovaný pre všetky plány. Dôležitou informáciou je, že tento nástroj poskytuje aj restové rozhranie prostredníctvom, ktorého je možné vykonávať rôzne úlohy. Toto restové rozhranie je zabezpečené cez HTTP Basic autentikáciu, čo znamená, že užívateľ má k dispozící meno a heslo prostredníctvom, ktorého sa autorizuje voči restovému rozhraniu. restové rozhranie poskytuje širokú škálu informácií. Umožňuje napríklad získavať informácie o stave jednotlivých plánov, získať prehľad plánov, rovnako umožňuje spúšťať plány.

4.6.6 Postman

Postman je program s grafickým užívateľským rozhraním na tvorbu testovacích sád a testovanie restového API [16]. Rovnako podporuje export a import týchto testovacích sád vo formáte JSON. Nástroj poskytuje pri spustení testovacej sady alebo len podčasti sady resp. jednotlivé HTTP požiadavku odpoveď od danej služby. Nástroj podporuje všetky známe HTTP metódy, rovnako umožňuje vkladanie HTTP hlavičky do HTTP žiadostí. Pre pohodlnejšiu prácu s týmto nástrojom je podporovaná funkcionalita, ktorá sa nazýva prostredie „environment“. Ide vlastne o nastavenie hodnôt premenných a ich zástupných symbolov, pričom tieto zástupné symboly môžeme použiť v testovacích sadách. Výhodou tohoto prístupu je možnosť oddelenia opakovaných údajov ako sú heslá špeciálne HTTP hlavičky pre žiadosti a opakujúcich údajov do samostatnej dátovej sady.

4.6.7 Newman

Newman predstavuje nástroj pre príkazový riadok, ktorý umožňuje beh testovacích sád exportovaných z nástroja Postman [15]. Výhodou takého riešenia je možnosť integrácie s riešeniami kontinuálne integrácie ako je napr. Bamboo, Teamcity a iné. Tento nástroj vyžaduje pre svoj beh prostredie NodeJS. Tento nástroj je ovládaný prostredníctvom konzole tj. podporuje sadu prepínačov, umožňujú nastaviť zdroj datávovej sady atď. Výsledok je zobrazený do konzole a je vrátený návratový kód podľa toho, či prebehli testy úspešne alebo nie.

```
$ newman run PostmanCollection.json -e environment.json --bail
newman

Example Collection with Failing Tests

→ Status Code Test
  GET https://echo.getpostman.com/status/404 [404 Not Found, 534B, 1551ms]
  1. response code is 200

┌─────────────────────────┬──────────┬──────────┐
│                         │ executed │   failed │
├─────────────────────────┼──────────┼──────────┤
│              iterations │        1 │        0 │
├─────────────────────────┼──────────┼──────────┤
│                requests │        1 │        0 │
├─────────────────────────┼──────────┼──────────┤
│            test-scripts │        1 │        0 │
├─────────────────────────┼──────────┼──────────┤
│      prerequest-scripts │        0 │        0 │
├─────────────────────────┼──────────┼──────────┤
│              assertions │        1 │        1 │
├─────────────────────────┴──────────┴──────────┤
│ total run duration: 1917ms                    │
├───────────────────────────────────────────────┤
│ total data received: 14B (approx)             │
├───────────────────────────────────────────────┤
│ average response time: 1411ms                 │
└───────────────────────────────────────────────┘


  #  failure        detail

 1.  AssertionFai…  response code is 200
                    at assertion:1 in test-script
                    inside "Status Code Test" of "Example Collection with
                    Failing Tests"
```

Obr. 4.15: Ukážka spustenia nástroja newman

Na obrázku č. 4.15 je ukážka spustenia nástroja newman pre kolekciu „PostmanCollection.json“ a premenné (prostredie) „environment.json“ s prepínačom –bail, ktorý zastaví beh celej kolekciu pri prvom zlyhaní. Na obrázku vidíme ďalej koľko HTTP žiadosti bolo spustení, celkový čas behu a priemerný čas v milisendukách pre odpoveď.

Kapitola 5

Implementácia

Táto kapitola pojednáva o všetkých troch implementovaných aplikáciach pre automatizáciu testovania a vydávania. Najprv rozoberie časť, ktorá pojednáva o aplikácií pre užívateľské rozhranie, následne aplikáciu pre obsluhu testovania a vydávania a nakoniec aplikáciu pre obsluhu v priamo v cloude kapitola č.5.3. Rovnako bude rozobratý princíp komunikácie medzi jednotlivými aplikáciami. Na záver bude rozobratý postup testovania aplikácie.

5.1 Webová aplikácia pre užívateľské rozhranie

Aplikácia pre užívateľské rozhranie je schopná zobrazovať informácie o aplikáciách, aktuálne implementovaných úloh vývojárov a o stave jednotlivých úloh. Aplikácia je postavená PHP frameworkom Symfony verzii 3.2.5 pre tvorbu MVC aplikácií [10]. Na grafické rozhranie bol použitý framework Twitter Boostrap vo verzii 3.3.7 [1] spolu s javascriptovou knižnicou JQuery vo verzii 3.2.1[2].

Táto aplikácia vystupuje v roli klienta, ktorý na pozadí komunikuje s rest rozhraním aplikácie Orchestrátor 5.2.

Aplikácia pre užívateľské rozhranie podporuje rôzne užívateľské role: Vývojár, Auditor, Vydavateľ a Administrátor. Úloha týchto užívateľských rolí je nasledovná:

- Administrátor - má prístup ku všetkým akciám v systéme a kľúčovou je úlohou je prideľovanie užívateľských rolí ostatných užívateľom
- Auditor - úlohou auditora je mať prehľad o akciách vykonaných v systéme, nemôže spúšťať vydávanie aplikácie, rovnako nemôže vyvíjať novú funkcionalitu
- Vývojár - tento užívateľ nemôže nasadzovať aplikácie, rovnako nemôže pridelovať role iným užívateľom, má prehľad o všetkých aplikách, rovnako môže pridávať nové aplikácie, má prehľad o úlohách a vytvárať nové úlohy
- Vydavateľ - tento užívateľ môže realizovať len vydávanie aplikácie

Prihlasovanie v aplikácií je realizované cez Google Oauth2 autentikácie, kedy pre prihlásenie postačuje klasický účet od Gmailu [4] a užívateľ je presmerovaný na zabezpečenú stránku od Googlu s formulárom, kde vyplní meno a heslo a po prihlásení je presmerovaný späť na adresu, kde beží aplikácia pre užívateľske rozhranie. Po prihlásení do aplikácie si

[1] http://getbootstrap.com/
[2] https://jquery.com/

aplikácia vyzdvihne z Orchestrátora zoznam rolí pridelenú užívateľovi. Pokiaľ užívateľ nemá pridelenú žiadnu užívateľskú rolu je zobrazená hláška.

Aplikácia pre užívateľské rozhranie je ovládana prostredníctvom menu naľavo. Aplikácia komunikuje na službu Orchestrátor 5.2 cez vystavené restové rozhranie aplikácie prostredníctvom knižnice curl. Komunikácia prebieha zabezpečená a to tak, že je do každej HTTP žiadosti vložená špeciálna hlavička s klúčom prostredníctvom, ktorej sa aplikácia autorizuje.

Aplikácia je zabezpečná pred rôznymi útokmi je XSS, rovnako pre vkladanie rôznych externých skriptov a obrázkov, ktoré by mohli spôsobiť možné nebezpečie. Zabezpečenie aplikácie je realizované cez samostný framework Symfony spolu s použitím špeciálnych CSP hlavičiek[3], ktoré sa vkladajú do každej HTTP žiadosti.

5.2 Orchestrátor

Úlohou tejto aplikácie je vystavenie restového rozhrania pre komunikáciu s užívateľským rozhraním a aplikáciou CCU 5.3. Aplikácia je postavená nad jazykom Java s využitím frameworku Spring vo verzii 4.2.0 [2]. Z tohoto frameworku je použíta časť SpringBoot a Spring MVC. Aplikácia ukladá dáta do MySQL databáze. Základnou úlohou aplikácie je získavanie informácií, vydávanie aplikácií na testovacie servre a ich otestovanie prostredníctvom Bamboo plánov. Konfigurácia týchto plánov bude vysvetlená neskôr. Aplikácia pracuje s užívateľskými rolami, informáciami o aplikáciach a informáciách o úlohách. Tieto informácie sú vystavené cez restové rozhranie, ktoré je konzumované webovou aplikáciou pre užívateľské rozhranie 5.1.

Aplikácia sa stará o základné úlohy ako je spúšťanie vydávania aplikácie, testovanie zmien v repozitároch a spúštanie testov a vystavovanie výsledkov cez API. Aplikácia je autorizovaná cez restové rozhranie pridávaním špeciálnej hlavičky do každého HTTP žiadosti s hodnotou kľúča, ktorá je overovaná proti správnej hodnote. Komunikácia priebeha zabezpečenie cez ssl protokol, preto je hlavička ťažko zachytiteľná a zobraziteľná. Aplikácia komunikuje s externými nástrojmi, ktoré boli uvedené v predchádzajúcich kapitolách. Ide o nástroja Bamboo, Consul a Newman. Je nutné zdôrazniť, že tieto nástroje predstavujú služby tretej strany, ktoré sú potrebné pre správnu funckionalitu prototypu, no je zároveň nutné povedať, že sú vo svojej podstate nahraditeľné za iné nástroje podľa konkrétnych potrieb firmy. Aplikácia funguje tak, že sa snaží aktuálne uložené úlohy, na ktorých pracuje vývojár kontrolovať na zmeny v repozitáre na danej vetvi v gite. Pokiaľ došlo k zmene aplikácia nasadí aplikáciu a spustí príslušnú sadu testov a vráti výsledok, čím dáva okamžíte spätnú väzbu na vývojárovi. Zároveň nutí vyvojára na používanie správneho vývojového cyklu v gite, ktorý hovorí o tom, že do gitu je vkladaná ucelená funckionalita. Tým sa zabezpečuje správny chod aplikácie. Implementácia tejto súčasti je tvorená samostnatných jedným vláknom, ktoré vyťahuje z databáze úlohy. Pre databázovu vrstvu je použitý SpringData, ktorý umožňuje dynamickú tvorbu SQL datazov.

[3] https://content-security-policy.com/

Aplikácia inicializuje pri spustení štyri vlákna, ktoré sa starajú o nasledovné činnosti:

- Kontrola zmien v Git repozitári pre úlohy označené na automatizované testovanie a vytváranie žiadosti o otestovanie
- Prechádzanie a spracovávanie žiadosti o automatizované testovanie
- Prechádzanie žiadosti o manuálne otestovanie. Dôvod tejto realizácie je treba zdôrazniť, že vydávanie aplikácie môže trvať od pár sekúnd až po pár minút. A blokovanie užívateľa na užívateľskom rozhraní niekoľko minút je nežiadúce.
- Zisťovanie stavu prebiehajúcich vydávacích balíčkov, ktoré boli spustené cez užívateľské rozhranie. Bamboo restové rozhranie nesposkyuje informáciu o aktuálne prebiehajúcich buildoch preto je nutné manuálne kontrolovať stav, ktorý je zobrazovaný v užívateľskom rozhraní

Orchestrátor komunikuje ako bolo spomenuté s externými nástrojmi ako je Consul. Tento nástroj obsahuje webové užívateľské rozhranie. Z tohoto nástroja je vyčítavaná pomocou klientskej knižnice konfigurácia aplikácie pre prostredia, rovnako konfigurácia pre nástroj Newman. Rovnako Orchestrátor získava z tohoto nástroja informácie o testovacích serverov spolu s ich konfiguráciu a Bamboo plánoch.

Ďalej Orchestrátor komunikuje s nástrojom Bamboo. S týmto nástrojom komunikuje cez restové rozhranie. Pre zabezpečenú komunikáciu je používá štandardná HTTP Basic autorizačná hlavička, ktorá obsahuje zakódované užívateľské meno a heslo. Nástroj Bamboo existuje v rôznych verziách a pre prototyp bola použitá verzia Bamboo 5.10.2.. Cez rest rozhraní spúšťa Bamboo plány, získava aktuálny stav plánov, rovnako získava informácie o prostrediach aplikácií a nasadených verziách.

V poslednom rade Orchestrátor komunikuje s aplikáciou CCU. Tejto aplikácií poskytuje informácie o smoke testoch a umožnuje jej zapisovať výsledok týchto testov.

5.3 CCU

Názov tejto aplikácie ako bol spomenutý predstavuje centrálnu riadiacu jednotku (central control unit (CCU)). Táto aplikácia je postavená rovnako ako Orchestrátor nad frameworkom Spring konkrétne podčastou SpringBoot a SpringMVC vo verzii 4.2.0. Úlohou tejto aplikácie je realizácia spustenie testov tesne pred spustením aplikácie do vonkajšieho sveta tj spracovanie požiadaviek od reálnych klientov. Tento typ testovania ako bolo spomenuté sa nazýva smoke testy.

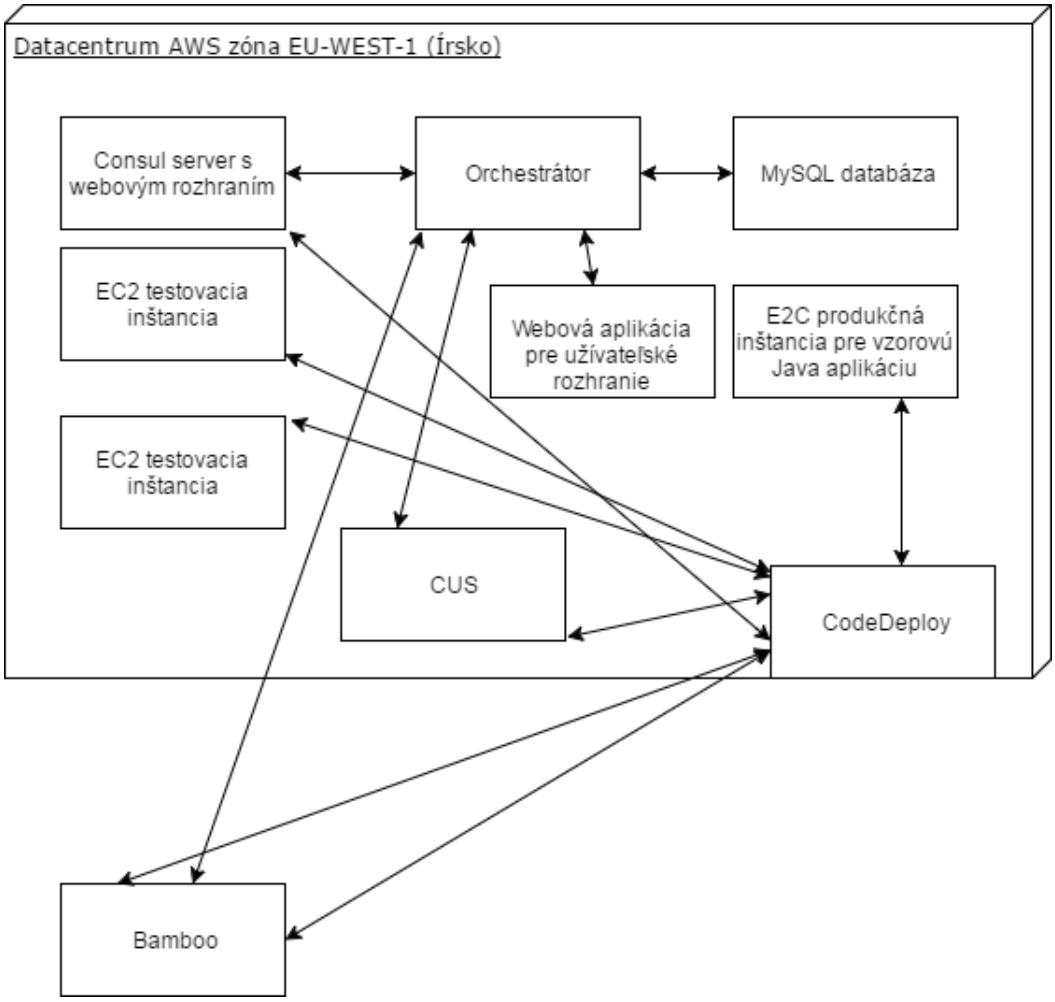
Pre každú zónu v cloude je dostupná jedna inštancia tejto aplikácie. Spolu s aplikáciou beží na serveri aj nástroj Consul, ktorý táto aplikácia používa na zistenie konfigurácie. Použitie tejto aplikácie ako kroku pre smoke testovanie aplikácie je voliteľné a jej použitie sa definuje s súbore appsec.yml, kde sa zahrnie skript, ktorý zakomunikuje s toutou aplikáciou a v prípade neúspechu ukončí vydávanie aplikácie na konkretnej EC2 inštancii. Pokiaľ je do aplikácie zahrnutá časť smoke testovania tak skript zistí z Consulu adresu CCU pre danú oblasť a spustí testovanie. CCU obdrží žiadosť o testovanie spolu s IP adresov inštancie a jednoznačným identifikátorom vydávania. CCU kontaktuje aplikáciu Orchestrátor odkiaľ získa potrebné smoke testy, ktoré si uloží na disk. Ako prerekvizita pre aplikáciu CCU je nástroj Newman, ktorý je nainštalovaný na EC2 inštancii a je možné ho spustiť z príkazového riadku. CCU spustí nástroj Newman a ako zdroj pre smoke testy uvedie cestu k smoke testom, ktoré uložila na disk. Získa výsledok z Newmana a výsledok uloží do Orchestrátora

a vráti výsledok inštancii, ktorá v prípade, že návratový kód je HTTP 200 tak pokračuje v nasadzovaní inak vydávanie ukončí.

Celý proces smoke testovania je identifický pre všetky aplikácie a ako bolo spomenuté je možné ho vynechať.

5.4 Testovanie

Testovanie prototypu bolo realizované v cloude Amazon Web Services. Všetky implementované aplikácie boli nasadené na Linuxové serveri so systémom Ubuntu 16.04.. Testovanie aplikácie bolo realizované vydaním všetkých aplikácií na virtuálne servre v rámci cloudu AWS. Pre Orchestrátor webovú aplikáciu bol použitý server Ubuntu 16.04 s PHP 5.5.3 a webovým serverom Apache 2.4.25. Pre aplikáciu CCU a Orchestrátor bol použitý rovnako Ubuntu 16.04 s Javou SE8. Na serveri pre aplikáciu CCU bol nainštalovaný Newman. Ten pre svoj beh vyžaduje NodeJS server vo verzii 6.4.0. Aplikácia Orchestrátor a webová aplikácia pre užívateľské rozhranie mali pridelené verejné IP adresy spolu s DNS názvami, ktoré boli automaticky pridelené pri vytvorení serverov. Pre beh nástroja Consul bol použitý rovnako Linuxový server s operačným serverom Ubuntu 16.04 a bol použitý Consul vo verzii 0.8.3.. Server mal opäť pridelenú verejnú IP adres spolu s DNS názvom pre prístup k nástroji z vonkajšieho sveta. Nástroj Consul bol spustený s prepínačom „-ui“, ktorý spustí nástroj Consul s webovým rozhraním, ktoré umožňuje nahrávanie hodnôt s kľúčami. Databázový server rovnako bežal v prostredí clodu AWS a bola použitá verzia 5.7.18.



Obr. 5.1: Infraštruktúra pre testovanie

Na obrázku č. 5.1 je ukázaná vyššie popísaná infraštúra pre testovanie. Celé testovanie prebieha v AWS zóne Írsko označovanú ako eu-west-1. Pre vydávanie aplikácia bol použitý nástroj Bamboo vo verzii 5.10.2 a vydávanie priamo v cloude realizuje nástroj CodeDeploy.

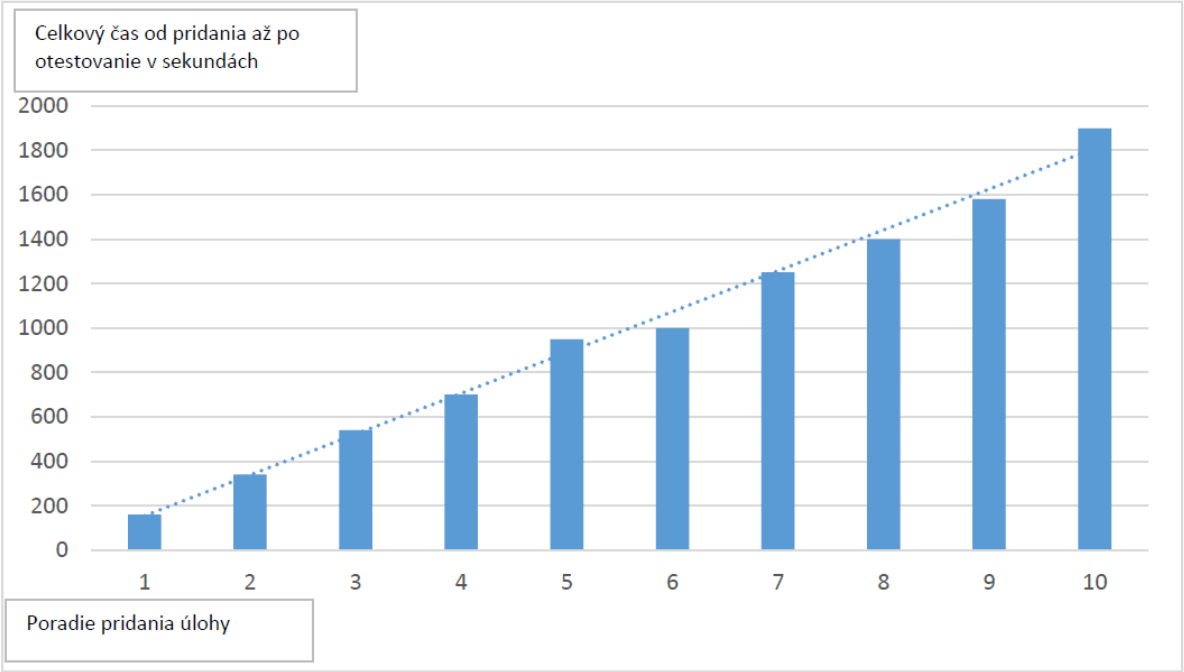
Celý proces testovania prebiehal na vzorovej aplikácia naprogramovanej v jazyku Java. Pre implementáciu tejto aplikácie bol použitý framework SpringBoot s frameworkom SpringMVC. Táto aplikácia obsahovala tri restové zdroje /v1/application/info, /v1/application/add a /v1/application/update. Prvý restový zdroj vracia informácie o aplikácií a je možné ho zavolať s HTTP metódou GET a vracia dáta vo formáte json. Druhý restový zdroj prijíma dáta vo formáte json a telo HTTP žiadosti „{applicationName: "nazov aplikacie"}“ a je možné ho zavolať s HTTP metódou POST a vracia HTTP kód 200.

Do webového rozhrania Consulu je pridaná konfigurácia pre pripravené testovacie servre spolu s Bamboo plánmi, ktoré spúšťajú vydávanie aplikácie na server a prihlasovacími údajmi a url do repozitára Gitu odkiaľ Bamboo plán berie zdrojové kódy.

Aplikácia sa nachádza v repozitári Github, ktoré je verejne dostupné, čo je podmienka, aby Orchestrátor mohol získavať potrebné informácie. Aplikácia je pridaná do aplikácie pre webové rozhranie spolu s release plánom a identifikátor nasadzovacieho projektu.

Následne sú vytvorené desať Git vetví z dôvodu otestovanie aplikácie pod záťažou. Ďalej vo webovej aplikácie pre užívateľské rozhranie pridáme postupne desať úloh a zvolíme automatické testovanie pri zmene v repozitára v danej vetvy. Pre každú úlohu pridáme rovnaké testy pre nástroj Newman.

Postupne aplikácia Orchestrátor začne nasadzovať jednotlivé úlohy na testovacie servre. Keďže máme k dispozícií dva testovacie servre a aplikácia spracováva úlohy jedným vláknom, tak je veľký predpoklad, že pri pridaní viacerých úloh nárazovo bude celé testovanie zablokované. Pre celé otestovanie jednej úlohy je zmeraný čas od získania servera až po celkové otestovanie. Po nárazovom nahraní všetkých úloh je celkový čas otestovania zmeraný a je približne tridsať minút. Jeden testovací beh zaberie teda približne tri minúty. Tento čas je možné považovať za dostatočný keďže automatizované testovanie poskytuje pomerne rýchlu odpoveď. Na druhú stranu otestovanie poslednej úlohy trvalo približne tridsať minút od vytvorenia úlohy až po otestovanie. Tento čas je stále ešte dostatočný každopádne pri vytvorení veľkého počtu úloh približne tisíc v rýchlom slede je celé testovanie značne spomalené.

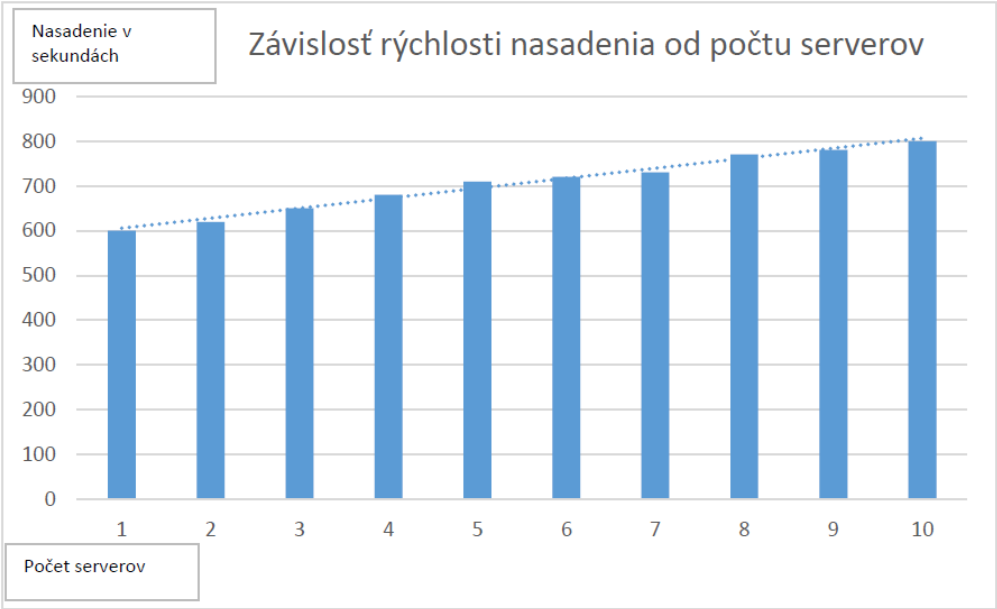


Obr. 5.2: Graf rýchlosti otestovania novej úlohy v závislosti od poradia pridania

Na obrázku č. 5.2 zobrazuje graf celkového otestovania úlohy v závislosti od pridania úlohy. Zároveň tento graf ukazuje približný čas spracovania jednej úlohy a zároveň ukazuje približnú lineárnu závislosť od toho, keď Orchestrátor zdetekuje zmeny v Git vetvách v jednotlivých úlohách. Z grafu je jasné, že pre veľký počas úloh je celkové testovanie značne spomalené.

Ďalšou úlohou, ktorá bola testovaná v prototype je vydávanie aplikácie na predprodukčné prostredie. Tento typ testovania nie je ovplyvňovaný ostatnými aplikáciami, lebo samotné vydávanie je realizované spustením cez nástroj Bamboo. Je nutné spomenúť, že pre

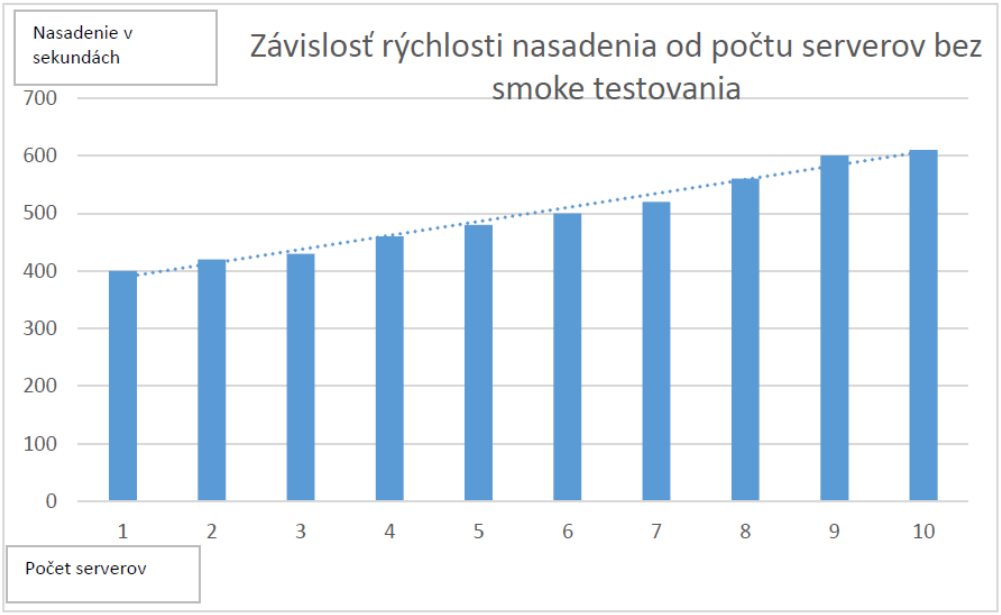
vydávanie je použitá technika blue/green čo v prostredí AWS spôsobí vytvorenie nových čistých serverov. Tieto serveri je následne potrebné pripraviť, čo sa realizuje cez nástroj Puppet [4], ktorý rovnako pripravuje konfiguráciu pre server. Po pripravení serverov je na jednotlivé servre nasadená aplikácia a každý server s aplikáciou je otestovaný cez smoke testy prostredníctvom aplikácie CCU. Pre dosiahnutie, čo najoptimálnejšieho výsledku z pohľadu času je vydávanie realizované paralelne na jednotlivé servre. Pre testovania ja použitý rovnaký server s pamäťou RAM 1GB a procesorom 1GHz. Pri testoch vydávania sa ako premenná používal počet serverov, na ktoré prebiehalo vydávanie.



Obr. 5.3: Graf závislosti rýchlosti vydávania od počtu serverov so záverečným smoke testovaním

Na obrázku č. 5.3 je zobrazený graf, ktorý ukazuje závislosť počtu serverov, na ktorý prebieha vydávanie pri paralelnom nasadzovaní blue/green, ktoré zahrňuje tvorbu nových serveroch ich úpravu, vydávanie a otestovanie prostredníctvom aplikácie CCU s nástrojom Newman. Je vidieť približne lineárnu závislosť rýchlosti vydávania od počtu serverov, na ktoré prebieha vydávanie.

[4] https://puppet.com/



Obr. 5.4: Graf závislosti rýchlosti vydávania od počtu serverov bez záverečného smoke testovania

Na obrázku č. 5.4 je zobrazený graf, ktorý ukazuje závislosť počtu serverov, na ktorý prebieha vydávanie pri paralelnom nasadzovaní blue/green, ktoré zahrňuje tvorbu nových serveroch ich úpravu, vydávanie bez smoke testovania. Je vidieť približne lineárnu závislosť rýchlosti vydávania od počtu serverov, na ktoré prebieha vydávanie. Zároveň pri porovnaní s grafom č. 5.3 je badateľné, že záverečné smoke testovanie ovplyvňuje celkový čas vydávania. Približné spomalenie predstavuje dve minúty. Tento čas nie je vôbec markantný a preto záverečné smoke testovanie je aplikovateľné do praxe.

Kapitola 6

Záver

V tejto práci je ukázané technické riešenie spôsobu kontinuálnej integrácie a dodávky. Táto problematika je v dnešnom dynamickom IT svete čoraz spomínanejší pojem. Vytvorený prototyp rieši problém automatizovaného testovania a vydávania serverových aplikácií.

Prototyp sa skladá z troch aplikácií a to z webovej aplikácie pre užívateľské rozhranie, aplikácie, ktorá ma za úlohu riadiť testovanie a vydávanie a aplikácie, ktorá má za úlohu finálne otestovanie aplikácie v cloude.

Navrhnuté riešenie je konfigurovateľné z pohľadu poradia krokov, ktoré je nutné vykonať predtým než dôjde k nasadeniu samotnej serverovej aplikácie do produkčného prostredia, rovnako aj z jazyka, ktorým bola aplikácia vytvorená. Prototyp bol otestovaný s Java serverovými aplikáciami prostredníctvom, ktorých bol prototyp overovaný. Prototyp používa už hotové nástroje, ktoré riešia zostavenie aplikácie rovnako aj pokrytie funkčnosti testovaných aplikácií.

Táto myšlienka má nevýhodu v tom, že pokiaľ užívateľ resp. organizácia potrebuje kvôli rôznym legislatívnym alebo iným dôvodom používať iný nástroj je nutné aplikácie preimplementovať. Jedným z riešení tohoto problému je vytvorenie akéhosi meta jazyka, ktorý by univerzálne riešil problém pri použití iného nástroja a vytváral by univerzálne riešenie, kedy by nový nástroj bol do aplikácie vložený ako nejaký modul a jeho špecifické komunikačné API by bolo zaobalené cez tento špeciálny jazyk. Takéto riešenie, ale obvykle je komplikovanejšie po implementačnej stránke a obvykle je jednoduchšie pri zmene nástroja upraviť existujúcu implementáciu. Celý prototyp bol otestovaný na vzorových serverových aplikáciách v podporovaných jazykoch, v našom prípade Java, na ktorých bola demonštrovaná konfigurovateľnosť tohoto prototypu.

Ako bolo spomenuté prototyp sa skladá z troch aplikácií. Všetky tieto aplikácie ale nie sú nutné pre beh výsledného prototypu. Je možné pri nasadení novej verzie aplikácie vynechať krok záverečného smoke testovania a teda nepoužiť aplikáciu CCU.

Prototyp používa nástroje Consul a Atlassian Bamboo. Medzi týmito aplikáciami a prototypom neexistuje pevná väzba, preto je pomerne jednoduché ich nahradiť za iné riešenia. Tento prototyp dokazuje predpoklad riadiaceho systému, ktorý je zložený z viacerých častí, ktoré umožňujú priebežné testovanie a vydávanie. Rovnako je pokrytá aj bezpečnostná stránka vydávania aplikácií do produkcie použitím blue/green vydávania spolu so záverečnými smoke testami.

Celý tento prototyp je univerzálny z pohľadu preloženia aplikácie v ľubovoľnom programovaciom jazyku, keďže preloženie aplikácie je realizovavné nástrojom tretej strany, ktorý zaručuje správne preloženie. Z pohľadu funkčnosti sa prototyp zameriava na serverové aplikácie, ktoré vystavujú nejaké API. Je to dané použitým nástrojom samozrejme opäť je

testovanie prenechané nástroju tretej strany, takže pri definovaní aplikácie sa predpokladá tvorbu restovej aplikácie. Prototyp je rozšíriteľný z pohľadu akejľkovek typu aplikácie, je len nutné zabezpečiť existenciu nástroja na testovanie tejto aplikácie a vhodných testovacích serverov pre daný programovací jazyk. Samozrejme netreba zabúdať, že pri testovaní aplikácie sa ešte realizuje testovanie prostredníctvom testera, ktorý overuje funkcionalitu ako celok. Tento prototyp prináša sadu výhod oproti existujúcim aplikáciam a to testovanie aplikácie pred samostným spustením do ostrej prevádzky a procesný manažment pre vývoj, testovanie a bez výpadkové vydávanie aplikácie.

Testovanie potvrdilo, že navrhnutý spôsob smoke testovania, ktorý bol implementovaný v cloudovom prostredí AWS nemá markatný vplyv na celkové vydávanie novej verzie aplikácie. Aplikácia CCU je navrhnutá pre konkrétneho cloudového dodávateľa AWS a je pri prechode k inému cloudovému dodávateľovi napríklad Windows Azure, je nutné spôsob spúšťania testovania upraviť a prispôsobiť novému dodáteľovi. Testovanie rovnako odhalilo problém prototypu s automatizovaným testovaním konkrétneho prípadu, kedy nastanú zmeny vo viacerých Git vetvách súčasne. Implementácia viacerých vlákien na realizáciu automatizovaného testovania náraža na problém kritickej sekcie a tou je Bamboo plán pre spúšťanie nástroja Newman, ktorý realizuje testovanie, vylepšením by bolo okrem zvýšenia počtu vlákien, ktoré spracovávajú automatizované testovanie aj počet Bamboo plán, ktoré spúšťajú nástroj Newman.

Rovnako ako vylepšie je považované napojenie na LDAP [1] autentikáciu pre získanie užívateľských rolí, miesto jednoduchého definovania rolí a ich ukladanie v databázovej tabuľke. Ako ďalším rozšírením pre vylepšie automatizovaného testovania aplikácií je napojenie na nástroje statickej analýzy, ktoré sledujú kvalitu kódu a identifikujú potencionálne chybné miesta v aplikácií. Tento typ testovania by bol realizovaný cez ďalší Bamboo plán, ktorý by zostavil aplikáciu, na ktorú by bola aplikovaná statická analýza kódu.

[1] https://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol

Literatúra

[1] Andreas Wittig, M. W.: *Amazon Web Services in Action*. Manning Publications, 2015, ISBN ISBN 978-1617292880.

[2] Gutierrez, F.: *Pro Spring Boot*. Apress, 2016, ISBN ISBN 978-1484214329.

[3] Jez Humble, D. F.: *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education, 2011, ISBN ISBN 978-0321601919.

[4] Justin Richer, A. S.: *OAuth 2 in Action*. Manning Publications, 2017, ISBN ISBN 978-1617293276.

[5] Laboon, B.: *A Friendly Introduction to Software Testing*. Konvoj, 1999, ISBN ISBN 978-1523477371.

[6] Lisa Crispin, J. G.: *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley Professional, 2009, ISBN ISBN 978-0321534460.

[7] Molyneaux, I.: *The Art of Application Performance Testing: From Strategy to Tools*. O'Reilly Media, 2014, ISBN ISBN 978-1491900543.

[8] Paul M. Duvall, A. G., Steve Matyas: *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Publishing Company, 2007, ISBN ISBN 978-0321336385.

[9] S, M. M.: *Learning Continuous Integration with TeamCity*. Packt Publishing, 2014, ISBN ISBN 978-1849699518.

[10] Salehi, S.: *Mastering Symfony*. Packt Publishing, 2016, ISBN ISBN 978-1535018388.

[11] Smart, J. F.: *Jenkins: The Definitive Guide: Continuous Integration for the Masses*. O'Reilly Media, 2014, ISBN ISBN 978-1449305352.

[12] Watson, M. P.: *Continuous Integration with Bamboo*. CreateSpace Independent Publishing Platform, 2016, ISBN ISBN 978-1535018388.

[13] Westby, E. J. H.: *Git for Teams: A User-Centered Approach to Creating Efficient Workflows in Gito*. O'Reilly Media, 2015, ISBN ISBN 978-1491911181.

[14] WWW stránky: Consul. [Online; navštíveno 25.3.2017]. URL https://www.consul.io/docs/index.html

[15] WWW stránky: Newman. [Online; navštíveno 23.3.2017]. URL https://www.getpostman.com/docs/newman_intro

[16] WWW stránky: Postman. [Online; navštíveno 23.3.2017].
URL https://www.getpostman.com/docs/

Príloha A

Obsah CD

Priložené CD obsahuje nasledujúce súbory:

- diploma_thesis.pdf - elektronická verzia textovej časti diplomovej práce
- src/orchestrator - adresár, ktorý obsahuje zdrojové kódy k aplikácií pre Orchestrátor aplikáciu
- src/frontend - adresár, ktorý obsahuje zdrojové kódy k aplikácií pre webové užívateľské rozhranie
- src/ccu - adresár, ktorý obsahuje zdrojové kódy k aplikácií pre CCU aplikáciu
- docs - adresár zo zdrojovými textami k dokumentácií

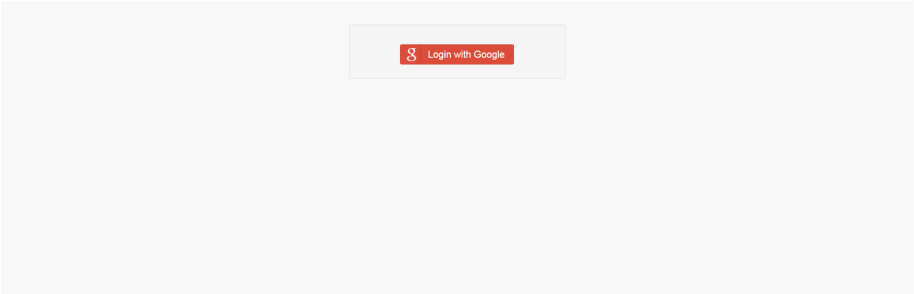
Príloha B

Manuál

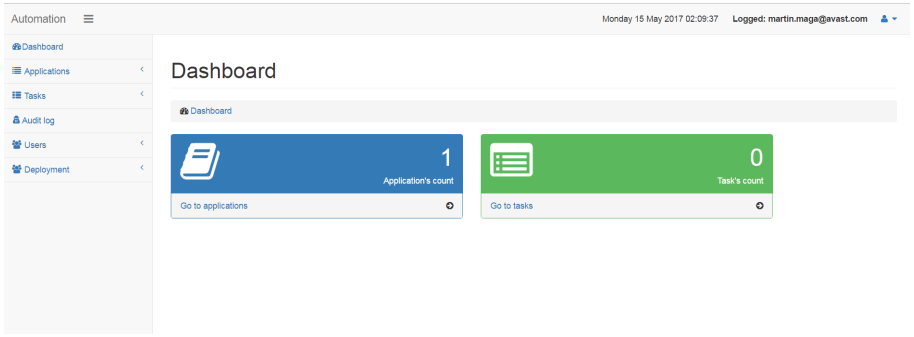
Aplikácia pre webové užívateľské rozhranie je prístupná na adrese: http://orchestrator-frontend.pdts-test.avg.com/. Na prihlásenie do aplikácie je potrebný Gmail účet. Pre správne použitie systému je nutná asistencia, pretože užívateľ nemá pridelené oprávnenia, preto nemôže vykonávať v systéme žiadne akcie, čo mu zdôrazní aj hláška, ktorá sa zobrazí pri prihlásení do aplikácie. Consul webové rozhranie je prístupné na adrese: http://consul-server-ui.pdts-test.avg.com. V aplikácií je pripravená jedna aplikácia na testovanie. V Consule je nahratá potrebná konfigurácia pre Bamboo plány a prostredia. Rovnako sú pripravené dve vzorové úlohy, jedna pre manuálne testovanie a jedna pre automatické testovanie. Pre testovanie boli pripravené 2 testovacie serveri http://chime-cloud-dev-13.mgm.avg.com/ a http://chime-cloud-dev-15.mgm.avg.com/. Tieto testovacie serveri nie sú bežne dostupné z vonkajšej siete.

Príloha C

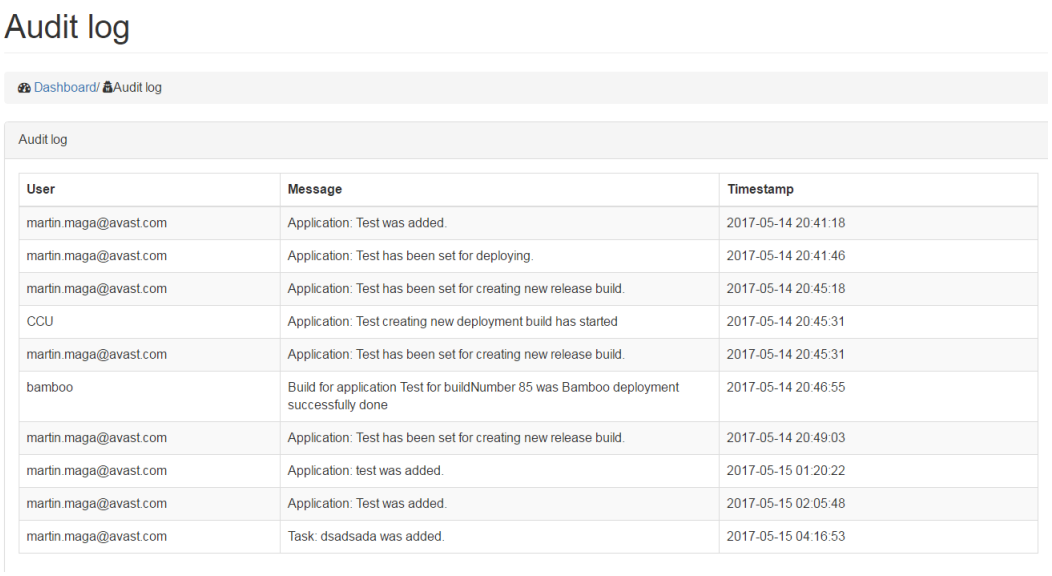
Ukážka užívateľského rozhrania



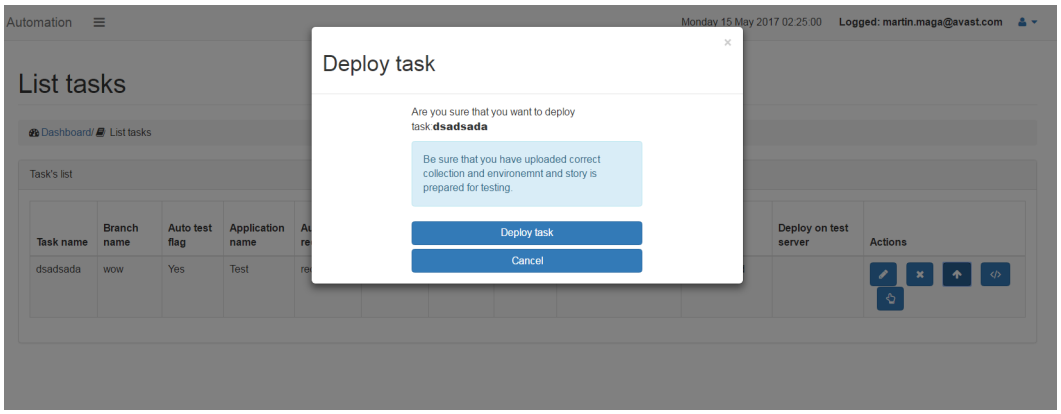
Obr. C.1: Prihlasovacia stránka aplikácie pre užívateľské rozhranie



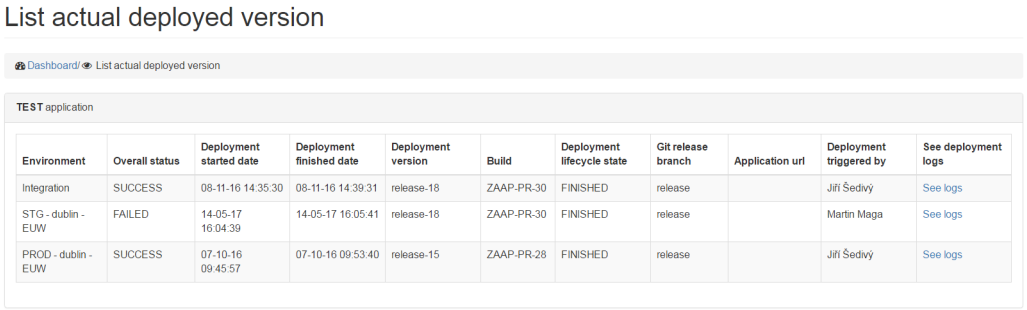
Obr. C.2: Úvodný prehľad po prihlásení do aplikácie



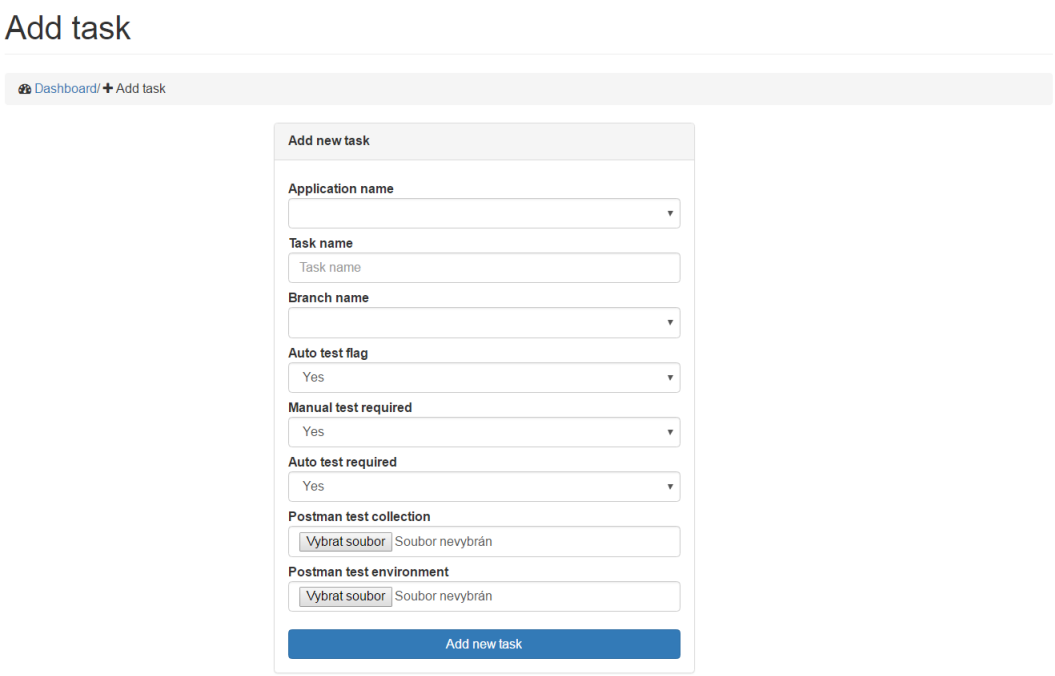
Obr. C.3: Stránka s audit logami



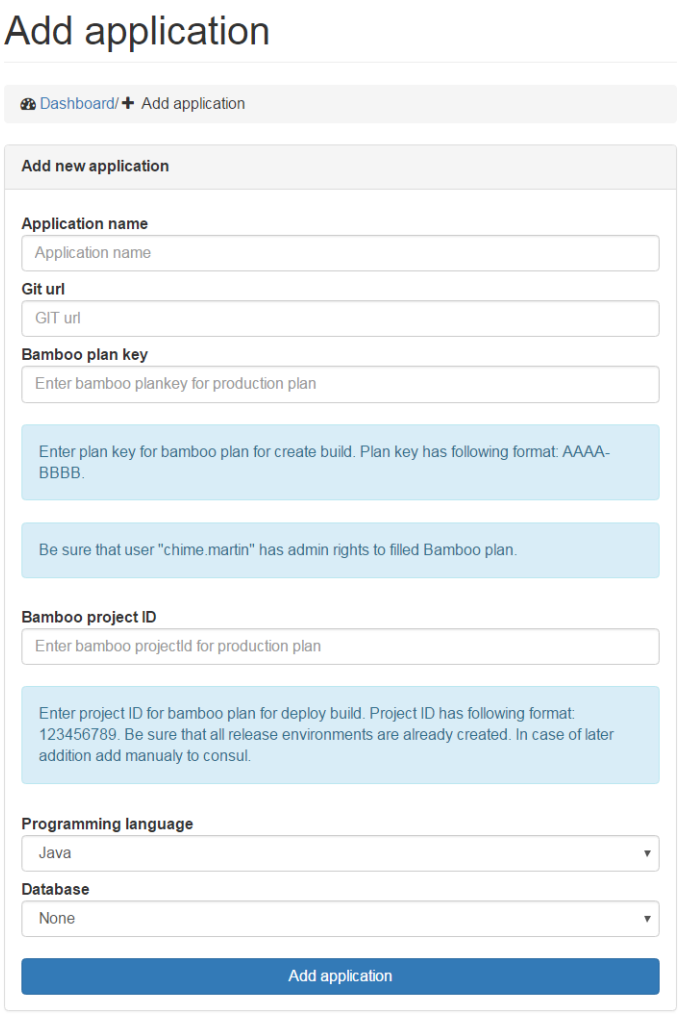
Obr. C.4: Manuálne nasadenie aplikácie kvôli testovaniu



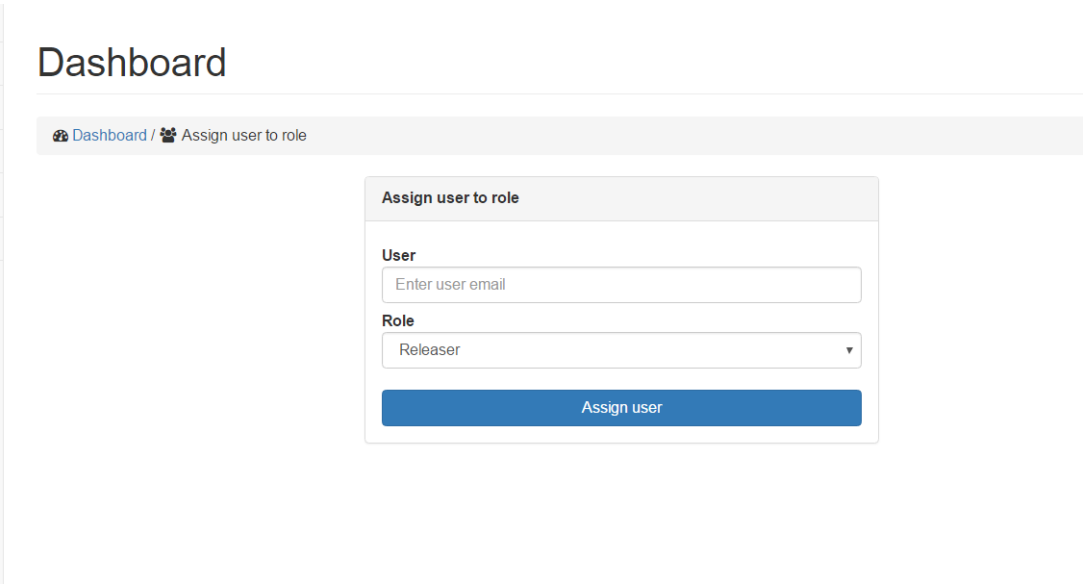
Obr. C.5: Zoznam nasadených verzií na jednotlivých prostrediach



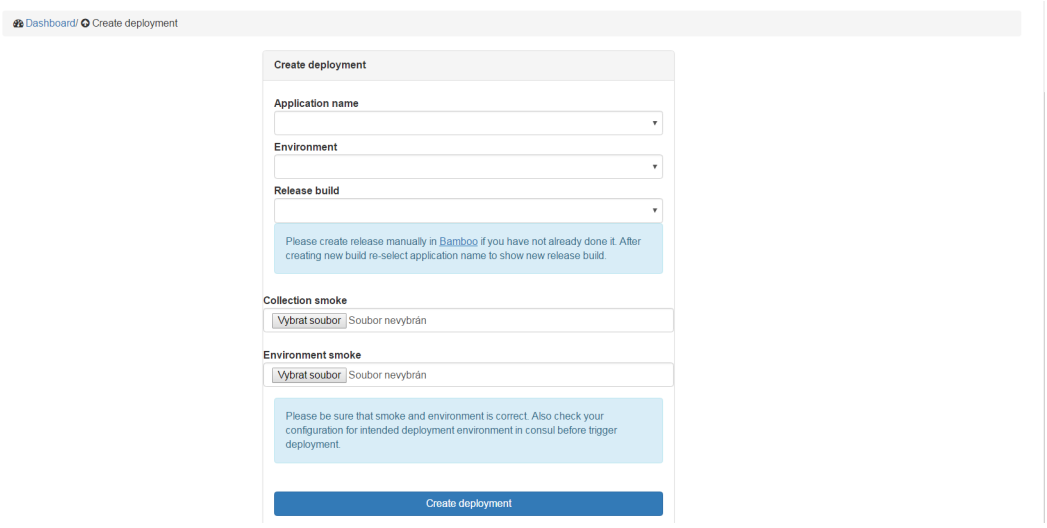
Obr. C.6: Vytvorenie novej úlohy



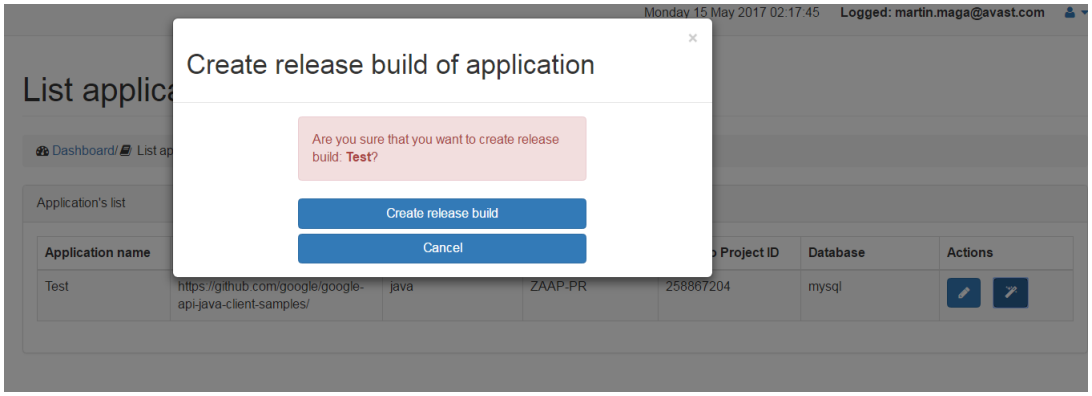
Obr. C.7: Pridanie novej aplikácie



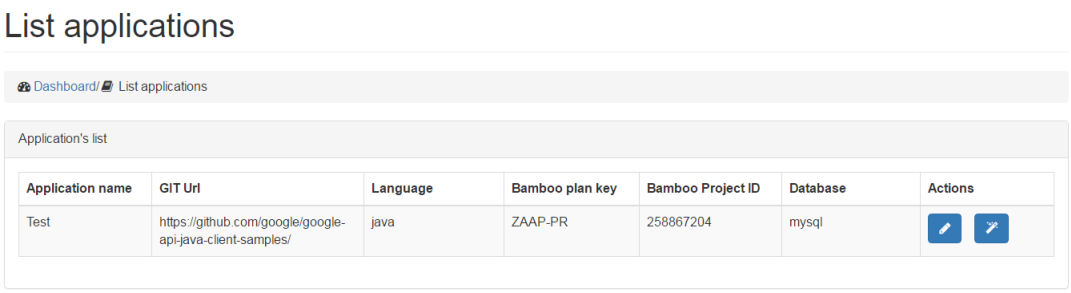
Obr. C.8: Pridanie novej role



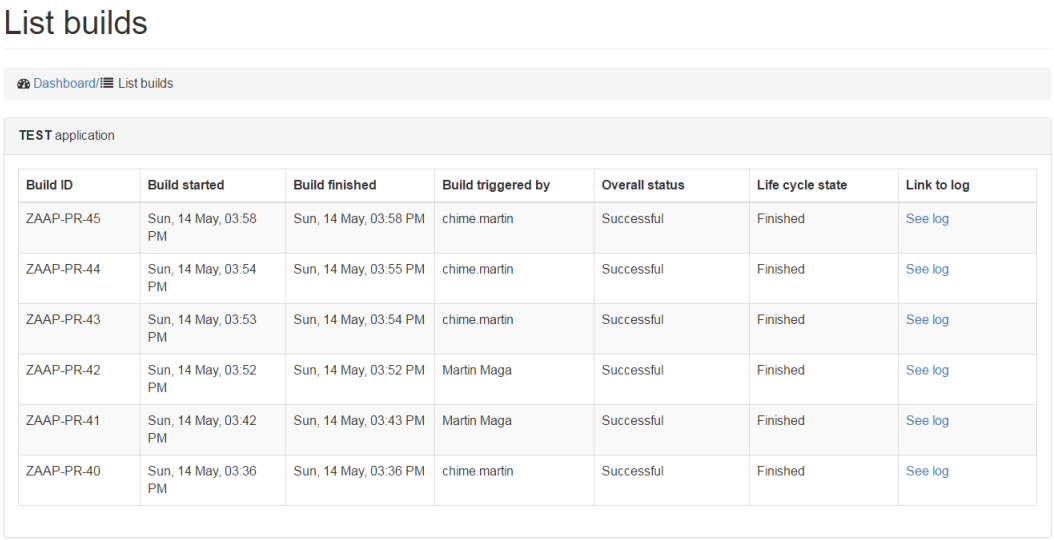
Obr. C.9: Vytvor požiadavku na nasadenie verzie na príslušné prostredie



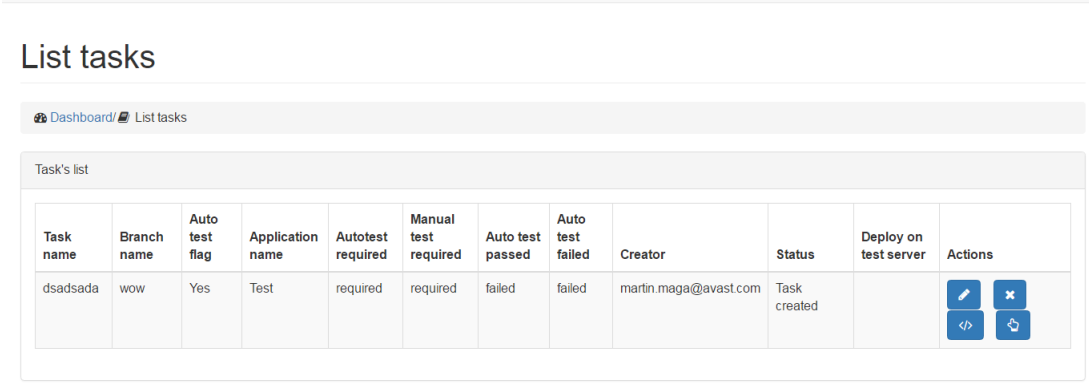
Obr. C.10: Vytvorenie nového vydávacieho balíčku



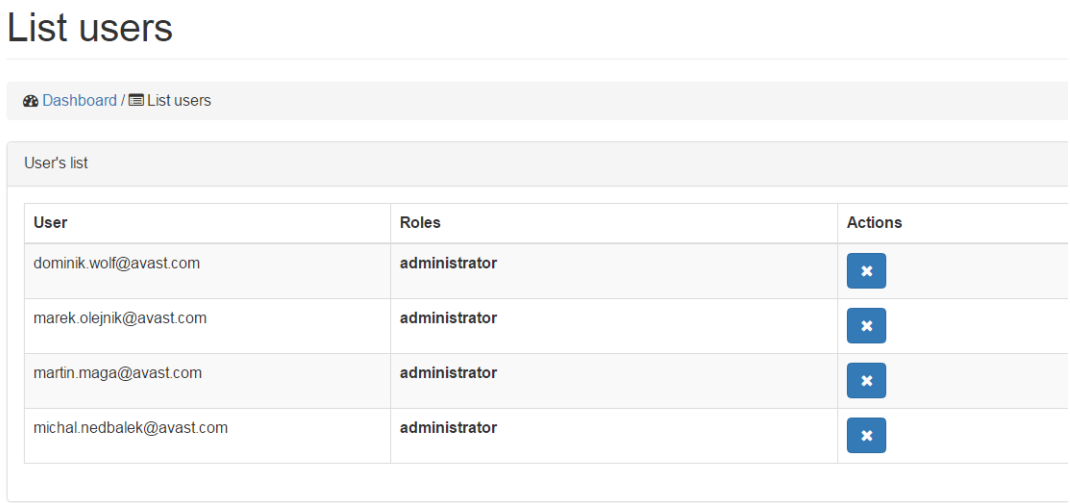
Obr. C.11: Zoznam aplikácií



Obr. C.12: Zoznam vytvorených buildov pre aplikáciu



Obr. C.13: Zoznam úloh



Obr. C.14: Zoznam užívateľov