Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Využití nástroje Bitbucket a platformy Amazon Web Services pro Continuous Integration, Delivery a Deployment aplikací

DIPLOMOVÁ PRÁCE

Studijní program: Aplikovaná informatika

Studijní obor: Informační systémy a technologie

Autor: Bc. Jan Melena

Vedoucí diplomové práce: Ing. Jana Fortinová

Praha, květen 2020

Prohlášení

Prohlašuji, že jsem diplomovou práci Využití nástroje Bitbucket a platformy Amazon Web Services pro Continuous Integration, Delivery a Deployment aplikací vypracoval samostatně za použití v práci uvedených pramenů a literatury.

V Praze dne 4. května 2020 .....

Bc. Jan Melena

Poděkování

Tímto bych chtěl poděkovat vedoucí diplomové práce Ing. Janě Fortinové za ochotu, podnětné rady a připomínky při psaní této práce. Dále bych chtěl poděkovat své rodině a přátelům za jejich podporu během mého studia.

Abstrakt

Diplomová práce je zaměřena na navržení metodiky pro zavedení Continuous Intergration, Delivery a Deployment procesu s využitím nástroje Bitbucket společně se službami Amazon S3 a CloudFront na platformě Amazon Web Services.

Nejdříve je v práci provedena komentovaná rešerše odborných zdrojů. Následují části věnované vymezení DevOps pojmu, popisu nástroje Bitbucket a platformy Amazon Web Services se zaměřením na služby Amazon S3 a CloudFront. Po teoretickém vymezením následuje navržení metodiky použití zmíněného nástroje a služeb pro podporu Continuous Integration, Delivery a Deployment procesu. Tato metodika je rozdělena na pět dílčích činností, které pokrývají celý proces od plánování, vytváření sestavení ze zdrojového kódu, prvotní nastavení nástroje a služeb až po následné automatické spouštění procesu a jeho reporting. Poslední kapitola práce je věnovaná ověření navržené metodiky na vybrané webové aplikaci s vyhodnocením jejího pilotního zavedení. Výsledky pilotního zavedení metodiky ukazují benefity týkající se rychlejšího nasazování nových verzí aplikace, snížení chybovosti a rychlejší opravy na produkčním prostředí.

Klíčová slova

DevOps, CI/CD proces, Continuous Integration, Continuous Delivery, Continuous Deployment, Bitbucket, Amazon Web Services, Amazon S3, CloudFront

JEL klasifikace

L86 Informační a internetové služby • Počítačový software

Abstract

The diploma thesis is focused on designing a methodology for the implementation of Continuous Intergration, Delivery and Deployment using Bitbucket tool together with services Amazon S3 and CloudFront on the Amazon Web Services platform.

First, a search of sources was conducted based on topic of the thesis. After that, sections are focused on the definition of DevOps, a description of Bitbucket, and the Amazon Web Services platform, focusing on Amazon S3 and CloudFront. The theoretical definition is followed by the design of a methodology for the use of the tool and services to support the Continuous Integration, Delivery and Deployment process. This methodology is divided into five sub-activities, which cover the entire process from planning, creating builds from source code, initial setup of the tool and services to the subsequent continuous process and reporting. The last chapter is devoted to the verification of the proposed methodology on a selected web application with the evaluation of its pilot implementation. This implementation shows clear benefits that consists of faster deployments of new application versions, decrease of error rate and faster recovery of production environment.

Keywords

DevOps, CI/CD process, Continuous Integration, Continuous Delivery, Continuous Deployment, Bitbucket, Amazon Web Services, Amazon S3, CloudFront

JEL Classification

L86 Information and Internet Services • Computer Software

Obsah

Úvod 13
Vymezení tématu a odůvodnění výběru 13
Cíle práce 14
Metody dosažení cíle 14
Předpoklady a omezení práce 14
Struktura práce 15
Výstup a očekávané přínosy 15
1 Komentovaná rešerše 16
1.1 Knižní publikace 16
1.2 Kvalifikační práce 17
1.3 Internetové zdroje 18
2 DevOps 19
2.1 Hlavní principy DevOps 21
2.1.1 Princip toku 21
2.1.2 Princip zpětné vazby 23
2.1.3 Princip kontinuálního vzdělávání se a experimentování 23
2.1.4 Shrnutí podkapitoly 24
2.2 Agilní vývoj 26
2.3 Continuous Integration 27
2.3.1 Typy testů 27
2.3.2 Automatické vs. manuální testování 28
2.3.3 Shrnutí podkapitoly 29
2.4 Continuous Delivery a Deployment 30
2.4.1 Strategie nasazování 31
2.4.2 Způsob vrácení chybného nasazení 32
2.4.3 Kritické záplaty chyb na produkčním prostředí 33
2.4.4 Continuous Deployment 33
2.5 Shrnutí 34
3 Nástroj Bitbucket pro správu vývoje, Continuous Integration a Delivery 35
3.1 Představení nástroje Bitbucket 35
3.2 Pipelines a Bitbucket 36
3.3 Deployments a Bitbucket 37
3.4 Průvodce nástrojem Bitbucket na reálné aplikaci 37

3.5 Shrnutí ... 39
4 Amazon Web Services jako platforma pro podporu Continuous Deployment ... 40
4.1 Uložiště Amazon S3 ... 41
4.2 Amazon CloudFront ... 42
4.3 Shrnutí ... 44
5 Metodika využití Bitbucket a AWS pro zavedení Continuous Integration, Delivery a Deployment procesu ... 45
5.1 Popis nástrojů a prostředí potřebných pro psaní aplikací ... 46
5.2 Role v rámci metodiky ... 47
5.2.1 DevOps architekt ... 47
5.2.2 Vývojář ... 47
5.2.3 Test manažer ... 47
5.2.4 Operativa ... 47
5.3 Artefakty metodiky ... 48
5.3.1 Plán testů ... 48
5.3.2 Plán procesu sestavení aplikace ... 48
5.3.3 Dokumentace nastavení nástroje a služeb ... 48
5.3.4 Sestavení výsledných souborů aplikace ... 48
5.3.5 Report běhu CI/CD procesu ... 48
5.4 Činnost: Plánování testování a tvorba sestavení ... 49
5.4.1 Jednotkové testy ... 49
5.4.2 E2E testy ... 51
5.4.3 Plánování sestavení ... 52
5.5 Činnost: Prvotní nastavení nástroje a služeb ... 55
5.5.1 Příprava nástroje Bitbucket ... 55
5.5.2 Příprava Amazon S3 ... 59
5.5.3 Příprava CloudFront ... 61
5.6 Činnost: Spouštění testů a sestavení v Bitbucket nástroji ... 64
5.6.1 Kroky v rámci CI/CD procesu ... 64
5.7 Činnost: Spouštění Deployment procesu ... 68
5.8 Činnost: Reporting běhu CI/CD ... 69
5.8.1 Návrh kanálů pro reporting ... 69
5.9 Shrnutí ... 70
6 Ověření metodiky na vybrané webové aplikaci ... 71
6.1 Popis architektury webové aplikace ... 71

6.2 Činnost: Plánování testování a tvorba sestavení ... 72
6.2.1 Plánování a tvorba ukázkového jednotkového testu ... 73
6.2.2 Plánování a tvorba ukázkového E2E testu ... 75
6.2.3 Plán procesu sestavení aplikace ... 77
6.3 Činnost: Prvotní nastavení nástroje a služeb ... 78
6.3.1 Vytvoření a nastavení repozitáře v Bitbucket ... 78
6.3.2 Vytvoření struktury v rámci Amazon S3 a CloudFront ... 79
6.4 Činnost: Spouštění testů a sestavení v Bitbucket nástroji ... 81
6.5 Činnost: Spouštění Deployment procesu ... 82
6.6 Činnost: Reporting běhu CI/CD ... 84
6.7 Vyhodnocení metodiky ... 86
6.8 Shrnutí ... 87
Závěr ... 88
Použitá literatura ... 90

Seznam obrázků

Obrázek 1: Porovnání frekvencí nasazování mezi Low a Elite performers. Zdroj: (DORA 2019) 19
Obrázek 2: DevOps model. Zdroj: (Amazon Web Services 2020i) 20
Obrázek 3: Tři hlavní principy. Zdroj: (Kim et al. 2016 str. 12), upraveno autorem 21
Obrázek 4: Dílčí kroky DevOps. Zdroj: (Watts 2017) 26
Obrázek 5: Návratnost investice do automatizace v čase. Zdroj: (Bureš et al. 2016) 28
Obrázek 6: Ukázka běhu konkrétní úlohy. Zdroj: (Atlassian 2020a) 36
Obrázek 7: Dashboard aplikace Bitbucket. Zdroj: autor 37
Obrázek 8: Ukázka repozitáře aplikace. Zdroj: autor 38
Obrázek 9: Ukázka běhů pipelines. Zdroj: autor 39
Obrázek 10: Struktura Amazon S3. Zdroj: autor 41
Obrázek 11: Ukázka distribucí v CloudFront. Zdroj: autor 43
Obrázek 12: Technologický základ aplikace. Zdroj: autor 46
Obrázek 13: Výsledek běhu testu. Zdroj: autor 51
Obrázek 14: Výsledné soubory po provedení sestavení aplikace. Zdroj: autor 53
Obrázek 15: Ukázka analýzy zdrojového kódu a vlivu změny na konkrétní aplikace. Zdroj: (Narwhal Technologies Inc. 2020) 54
Obrázek 16: Výstup po spuštění sestavení ovlivněných aplikací. Zdroj: autor 54
Obrázek 17: Vytvoření repozitáře. Zdroj: autor 55
Obrázek 18: Model větvení. Zdroj: autor 56
Obrázek 19: Sekce v rámci konfigurace pipelines. Zdroj:(Atlassian 2020a) 57
Obrázek 20: Vytvoření Bucket kontejneru v Amazon S3. Zdroj: autor 60
Obrázek 21: Vytvoření distribuce CloudFront. Zdroj: autor 62
Obrázek 22: Vytvoření nového CNAME záznamu v administraci Blueboard. Zdroj: autor 63
Obrázek 23: Ukázka přehledu vynucení obnovení souborů v rámci CloudFront. Zdroj: autor 63
Obrázek 24: BPMN model procesu změny na větvích feature, hotfix nebo bugfix. Zdroj: autor 65
Obrázek 25: BPMN model procesu vytvoření nového pull request požadavku. Zdroj: autor 66
Obrázek 26: BPMN model procesu změny na master větvi 67
Obrázek 27: Architektura webové aplikace. Zdroj: autor 71
Obrázek 28: Ukázka komponenty vypisující strukturu materiálu. Zdroj: autor 73
Obrázek 29: Výsledek běhu jednotkového testu komponenty. Zdroj: autor 75
Obrázek 30: Obrazovka z aplikace truck image view. Zdroj: autor 75
Obrázek 31: Výsledek běhu E2E testu. Zdroj: autor 77
Obrázek 32: Porovnání spuštění příkazu s analýzou vlivu změny na aplikace. Zdroj: autor 78
Obrázek 33: Nastavení proměnných v Bitbucket nástroji. Zdroj: autor 79
Obrázek 34: Směrování chybných požadavků v nastavení distribuce ve službě CloudFront. Zdroj: autor 80
Obrázek 35: HTTP odpověď z vlastní domény. Zdroj: autor 80
Obrázek 36: Výsledek běhu sestavení a testování v CI/CD nástroji při změně na master větvi. Zdroj: autor 81

Obrázek 37: Výsledek běhu Deployment procesu v Bitbucket nástroji. Zdroj: autor 83
Obrázek 38: Ukázka e-mailové notifikace z nástroje Bitbucket. Zdroj: autor 84
Obrázek 39: Nastavení propojení nástroje Bitbucket a Slack. Zdroj: autor..... 85
Obrázek 40: Ukázka notifikace v nástroji Slack. Zdroj: autor 85

Seznam výpisů programového kódu

Výpis 1: Ukázka jednotkového testu. Zdroj: autor 49
Výpis 2: Ukázka E2E testu. Zdroj: autor 51
Výpis 3: Příklad výpisu konfigurace pipelines pro Bitbucket (zkráceno). Zdroj: autor 58
Výpis 4: Konfigurace kroků na větvích typu feature, hotfix a bugfix. Zdroj: autor 65
Výpis 5: Konfigurace kroků při pull request požadavku. Zdroj: autor 66
Výpis 6: Konfigurace kroků testování a sestavení v CI/CD nástroji. Zdroj: autor 67
Výpis 7: Konfigurace kroků Deployment v CI nástroji. Zdroj: autor 68
Výpis 8: Jednotkový test komponenty. Zdroj: autor 73
Výpis 9: Zápis E2E testu. Zdroj: autor 76
Výpis 10: Reálné nastavení kroků v CI/CD nástroji pro změnu na mastesr větvi. Zdroj: autor 81
Výpis 11: Konfigurace pro spouštění Deployment procesu do testovacího prostředí. Zdroj: autor 82

Seznam zkratek

CI	Continuous Integration
CD	Continuous Delivery/Deployment
HTTP(S)	Hypertext Transfer Protocol (Secure)
CLI	Command Line Interface
AWS	Amazon Web Services
REST	Representational state transfer
API	Application programming interface

Úvod

S přirozeným vývojem moderních technologií, praktik, postupů a konkurence vzniká potřeba rychleji reagovat na trh a zákazníky. Dříve zavedené postupy nasazování aplikace do produkčního prostředí jednou za čtvrtletí, půlrok nebo rok jsou pro mnohé firmy nerealistické z hlediska udržení své pozice na trhu. Tím vzniká velký tlak na častější změny aplikace, které je potřeba co nejdříve dostat na produkční prostředí ke koncovému zákazníkovi. (Ahmed 2019)

Jak autoři Humble a Farley ve své knize *Continuous delivery: reliable software releases through build, test, and deployment automation* (2010, s. 23–24) uvádějí, jedním ze způsobů jak řešit tento tlak na rychlost je formou nastavení procesu, při kterém je možné kdykoliv ze zdrojového kódu vytvořit produkční sestavení aplikace. Není tak zapotřebí čekat několik měsíců, než bude možné veškerou zamýšlenou funkcionalitu zveřejnit svým zákazníkům a dostat zpětnou vazbu až po velmi dlouhé době. Proto se začíná uvnitř organizací diskutovat pojem DevOps nebo kombinace pojmů Continuous Integration, Delivery a Deployment, a jejich případné postupné zavedení k vyřešení zmíněného problému.

Přestože je dostupné velké množství postupů a kvalifikačních prací věnovaných DevOps, specifická kombinace nástroje Bitbucket společně se službami Amazon S3 a CloudFront na platformě Amazon Web Services nebyla doposud navrhnuta. Tato práce byla vytvořena s cílem navrhnout metodiku, která by tento nástroj a služby mezi sebou propojila a podpořila tím celý proces Continuous Integration, Delivery a Deployment. Metodika je následně ověřena na vybrané webové aplikaci napsané v Angular frameworku za použití nástrojů Angular CLI a NX CLI.

Vymezení tématu a odůvodnění výběru

Přestože existuje spousta prací na téma návrhu metodiky pro zavádění procesů Continuous Integration, Delivery nebo Deployment, které jsou zmíněné v rešerši zdrojů, kombinace nástrojů Bitbucket a platformy Amazon Web Services doposud nebyla použita. Zároveň je práce specifická tím, že procesy zavádí pro aplikace, které jsou psané v Angular frameworku.

Vzhledem k tomu, že nasazování aplikací do produkčního prostředí je čím dál častější a rychlejší, podle DORA (2019) je meziroční nárust tzv. Elite Performers (vyznačující se nasazováním několikrát za den) ze 7 % v roce 2018 na 20 % v roce 2019, vzniká spousta nástrojů, které se snaží v nějaké míře proces nasazování aplikací podporovat. Mezi ty patří například zmíněný nástroj Bitbucket společně s platformou Amazon Web Services.

Téma práce bylo vybráno na základě osobních zkušeností autora pracující ve firmě, která se snaží své procesy týkající se vývoje aplikací neustále zlepšovat. Vzhledem k tomu, že podobný proces bylo potřebné v této firmě zavést a neexistovala metodika využívající výše

zmíněnou kombinaci nástroje a služeb, autor se rozhodl navrhnout vlastní metodiku, kterou v této firmě následně ověřil.

Cíle práce

Hlavním cílem diplomové práce je navrhnout metodiku, která týmům pomůže zavést Continuous Integration, Delivery a Deployment proces do svých vývojových činností s využitím nástroje Bitbucket a vybraných služeb na Amazon Web Services. Pro dosažení hlavního cíle je dále definováno pět dílčích cílů:

- Prvním dílčím cílem je vymezit pojem DevOps ve spojení s Continuous Integration, Delivery a Deployment procesem.
- Druhým dílčím cílem je představit nástroj Bitbucket pro podporu Continuous Integration a Delivery procesu.
- Třetím dílčím cílem je představit služby na Amazon Web Services podporující Continuous Deployment proces.
- Čtvrtým dílčím cílem je navrhnout metodiku pro zavedení Continuous Integration, Delivery a Deployment procesu.
- Pátým dílčím cílem je ověřit navrhnutou metodiku na vybrané webové aplikaci.

Metody dosažení cíle

Tato práce pro dosažení vytyčených cílů používá metodiku Design Science Research (DSR). Podstatou metodiky je vytvoření artefaktu, který je možno dále využít v daném odvětví za předpokladu, že řešení je inovativní, nebo zefektivňuje dříve vytvořené. Metodika je rozdělena na šest částí (Peffers et al. 2008):

1. Problem identification and motivation (*Identifikace problému a motivace*)
2. Define the objectives for a solution (*Definování cílů*)
3. Design and development (*Návrh a vývoj*)
4. Demonstration (*Ukázka*)
5. Evaluation (*Vyhodnocení*)
6. Communication (*Komunikace*)

Předpoklady a omezení práce

Při čtení textu této diplomové práce se od čtenáře očekává základní znalost terminologie z prostředí vývoje softwaru a psaní testů. Přestože jsou pojmy vysvětleny a v některých případech doprovázeny ukázkami kódu, jejich podrobnější popsání by bylo nad rámec tématu této práce.

Přestože má tato práce snahu vystihnout podstatu zavádění procesu nezávisle na programovacím jazyku, všechny příklady a ukázky z kódu jsou vytvořené pro Angular framework.

Struktura práce

Práce je rozdělena na šest kapitol. První kapitola je věnovaná komentované rešerši zdrojů. Následují kapitoly 2, 3 a 4 popisující nejdříve pojmy DevOps, Continuous Integration, Delivery a Deployment, na které navazuje představení nástroje Bitbucket a služeb Amazon S3 a CloudFront spadající pod platformu Amazon Web Services. Tyto kapitoly jsou důležité pro snadnější pochopení postupů a rozhodnutí provedených při samotném návrhu metodiky. Kapitola 5 je věnovaná zmíněnému návrhu metodiky zahrnující popis rolí, artefaktů a činností. Metodika je na závěr v kapitole 6 ověřena na vybrané webové aplikaci.

Výstup a očekávané přínosy

Hlavním výstupem této práce je navržená metodika pro zavedení Continuous Integration, Delivery a Deployment procesu s využitím nástroje Bitbucket a vybraných služeb Amazon Web Services. Tato metodika je následně ověřena na aplikaci postavené na Angular frameworku.

Mezi hlavní očekávané přínosy lze zařadit konkrétní ukázku celého procesu od změny ve zdrojovém kódu až po nasazení do produkčního prostředí, přičemž tento proces funguje zcela automaticky. To umožní čtenáři nahlédnout do dílčích kroků, které vedou k možnému nasazování nových verzí aplikace do produkčního prostředí i několikrát denně. Zároveň to čtenáři umožňuje tento proces použít a přizpůsobit jej vlastním potřebám.

1 Komentovaná rešerše

Tato kapitola je věnovaná vybraným zdrojům, ze kterých bylo v rámci této diplomové práce čerpáno. Kapitola je rozdělena na knižní publikace, kvalifikační práce a internetové zdroje.

1.1 Knižní publikace

Kniha *The DevOps handbook: how to create world-class agility, reliability, & security in technology organizations* (Kim et al. 2016) je pro tuto práci klíčovou, jelikož popisuje samotným pojem DevOps. Kniha pojednává o DevOps praktikách, a to od úplných základů, přes transformace stávajících procesů až po praktické ukázky z reálného prostředí organizací zavádějících DevOps. Ve zbylých kapitolách jsou jednotlivé kroky pojednávající o zmiňovaných praktikách detailněji rozepsané a přiblížené na případových studiích.

Druhou knihou, která je využita pro tuto práci je *Continuous delivery: reliable software releases through build, test, and deployment automation* (Humble a Farley 2010). Kniha detailně popisuje nejen proces Continuous Delivery a Deployment, jak název napovídá, ale také proces Continuous Integration. Mimo to se věnuje velmi blízkým tématům, kterými jsou správa konfigurace, nástroje podporující vytváření sestavení aplikace, správa dat při procesu aktualizace, nastavení pravidel závislostí mezi aplikacemi, knihovnami nebo komponentami nebo správa verzí zdrojového kódu.

Další knihou na téma DevOps je *DevOps: a software architect's perspective* (Bass et al. 2015), která se věnuje DevOps v kontextu cloud řešení. Zaměřuje se na to, jak praktiky ovlivňují IT systémy a role, a následné zasazení DevOps do agilních metodik vývoje. Nosnou myšlenkou této publikace je pohled na DevOps očima softwarového architekta. Jednotlivá témata jsou rozepsána více do hloubky než v předchozí publikaci, což je podloženo poslední částí knihy. Ta na různorodých případových studiích popisuje zavádění a konkrétní překážky spojené s DevOps v organizacích.

Poslední publikací, která sloužila jako podklad především pro část věnovanou procesu Continuous Integration je *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu* (Bureš et al. 2016). Tato kniha obsahuje praktické informace z oblasti testování. Text je zaměřen na řízení testování, tvorbu strategie, popis úrovní testů, automatizaci testování nebo přípravu prostředí a dat. Zároveň je zde krátce vysvětlen i proces Continuous Delivery.

1.2 Kvalifikační práce

Kvalifikační práce Lapotky (2018) s názvem *Analýza a návrh možností automatizace releasů aplikací ve společnosti s ITIL prostředím* se zaměřuje na proces automatického nasazování aplikace do produkčního prostředí. Autor nejdříve popisuje DevOps praktiky, ITIL framework a automatizaci nasazování aplikací. Tyto poznatky následně používá v kapitolách analýzy vybrané firmy z hlediska možností automatizace, navrhuje možnosti automatizace v daném prostředí a stanovuje výsledný rámec pro hodnocení úspěšnosti zavedení této praktiky. Vzniklý rámec v jeho práci byl použit jako zdroj pro možné činnosti při vytváření vlastní metodiky, přestože se Lapotka zabývá pouze dílčí částí.

Práce na téma *Využití nástroje Protractor pro automatizované testování webových aplikací* od autora Hoffmanna (2018) pojednává o procesu automatizace testování. Podrobně se věnuje oblasti testování, které je prováděno na aplikaci psané v AngularJS frameworku za pomocí nástroje Protractor. Výsledkem této práce je metodika pro využití nástroje Protractor a Jenkins při zavádění automatizace testování. Tato práce, ze které bylo čerpáno při navrhování dílčích činností v rámci vlastní metodiky byla použita především pro dílčí kroky věnované Continuous Integration.

Vitězslav Štrajt (2020) ve své práci *Průběžné doručování produktu do cloudové platformy Salesforce* navrhoval způsob doručování vlastního řešení. Autor nejdříve popisuje cloudová řešení a specifika aplikací vyvíjených za účelem nasazení do tohoto prostředí. V dalších částech popisuje proces průběžného doručování, který vztahuje na nástroje potřebné pro doručování do platformy Salesforce. Stejně jako u práce Lapotky byly závěry práce použity při tvorbě činností metodiky, především proces nasazování do cloudového řešení Salesforce.

Macháč (2018) se zaměřil na průběžnou integraci, dodávku a nasazení v práci s názvem *Průběžná integrace a průběžná dodávka/nasazení projektu KYPO*. Předmětem této práce bylo navrhnout řešení pro projekt KYPO (Kybernetický polygon), které v poslední kapitole implementoval do provozu. Ovšem v jeho práci se používají jiné nástroje a jiná aplikace psaná v odlišném jazyce.

Poslední práce s názvem *Metodika Continuous Integration při vývoji webové aplikace* (Fól 2018) je zaměřena na dílčí část integrace z celého procesu Continuous Integration, Delivery a Deployment. V práci jsou popsané různé nástroje na Continuous Integration, mezi kterými nicméně chybí Bitbucket. Výsledkem této práce je navržený proces pro Continuous Integration využitý konkrétní webovou aplikací.

Přestože se všechny výše uvedené práce zaměřují na oblast Continuous Integration, Delivery nebo Deployment, žádná z nich se přímo nevěnuje nástroji Bitbucket nebo platformě Amazon Web Services, konkrétně službám CloudFront nebo Amazon S3.

1.3 Internetové zdroje

V rámci internetových zdrojů bylo čerpáno především z dokumentace nástroje Bitbucket (2020), kde je podrobně popsáno, jak nástroj funguje a jak jej lze integrovat například do nástroje Jira nebo později zmíněných služeb na platformě Amazon Web Services.

Druhou dokumentací využívanou v rámci této práce je dokumentace služeb Amazon S3 a CloudFront (Amazon Web Services 2020f; 2020e; 2020g). Ty obsahují podrobné popisy, návody a praktiky, jak jednotlivé služby nastavit a používat pro zavedení Continuous Delivery a Deployment procesu.

Posledním zdrojem je dokumentace pro nástroj NX CLI (Narwhal Technologies Inc. 2020). Ten je používán při spouštění testů nebo sestavení v činnostech navrhované metodiky.

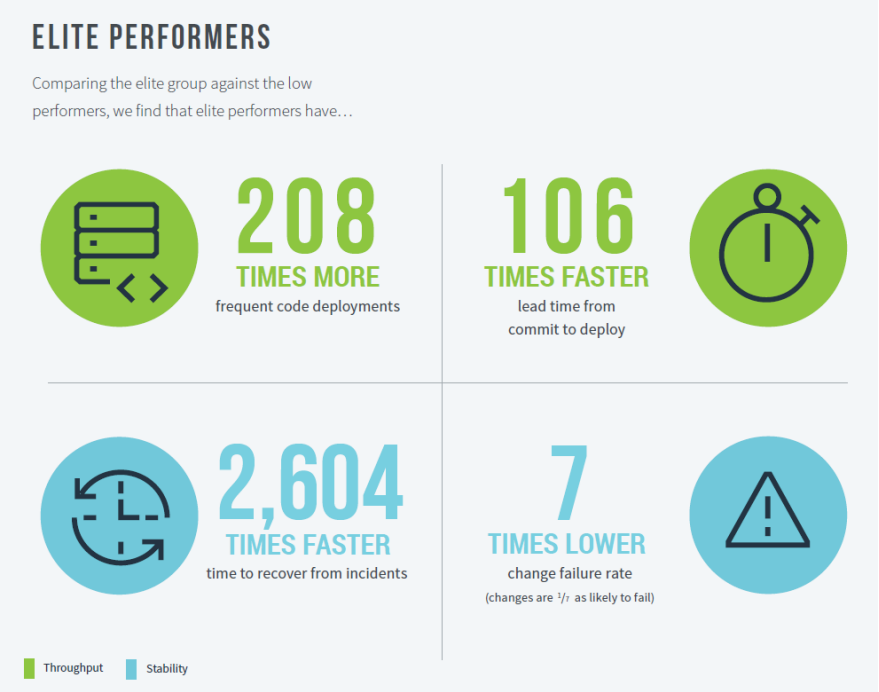
2 DevOps

Pod pojmem DevOps se skrývá spojení anglických slov *development* a *operations,* což znamená v překladu vývoj a provoz aplikace. To představuje hlavně propojení procesů a lidí, kteří mají za cíl doručování kvalitního softwaru svým zákazníkům. Historicky byly tyto dvě části oddělené, což mělo za důsledek pomalejší řešení problémů, nasazování aplikace do produkčního prostředí a menší vzájemná důvěra mezi těmito odděleními (Atlassian 2020b).

DevOps je reakce na pomalé nasazování aplikací do produkčního prostředí. Čím déle trvá danou verzi aplikace dostat na trh, tím menší mají nové funkce a opravy dopad. Ty jsou často nezbytné pro získání konkurenční výhody na trhu nebo udržení zákazníka. V ideálním světě by proto bylo žádoucí aplikaci dodávat kontinuálním způsobem, což je označováno jako Continuous Delivery a Deployment (Bass et al. 2015).

V roce 2019 byl proveden průzkum v organizacích používající DevOps, který proběhl pod záštitou organizace DORA (2019). Z tohoto průzkumu vyplynuly následující závěry:

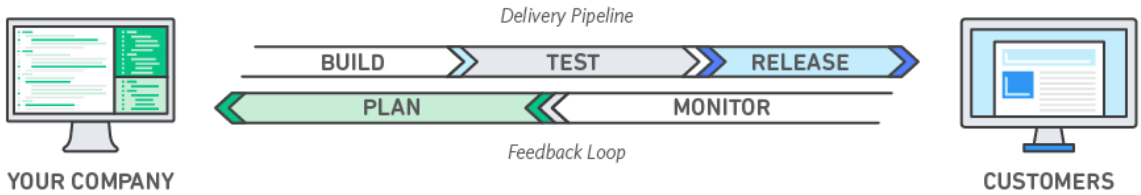
- *„208x častější nasazování do prostředí.*
- *106x rychlejší čas mezi změnou v kódu a její nasazení do prostředí.*
- *2604x rychlejší čas vyřešení problému.*
- *7x nižší šance výskytu problému v aplikaci.“*



Obrázek 1: Porovnání frekvencí nasazování mezi Low a Elite performers. Zdroj: (DORA 2019)

Na obrázku 1 je možné vidět jaký dopad zavedení DevOps do organizace může mít. Na obrázku je vidět rozdíl mezi tzv. Elite a Low Performes. DORA (2019) definuje Elite Performers jako organizace zvládající nasazování až několikrát za den, jejich doba od změny v kódu a nasazení do prostředí je menší než jeden den, rychlost obnovení služby menší jedné hodiny a chybovost způsobená změnou v kódu se nachází v rozmezí 0-15 %. Oproti tomu Low Performers jsou organizace, které mají frekvenci nasazování a dobu pronesení změny do prostředí mezi 1 až 6 měsíci, čas potřebný k obnově služby mezi týdnem až měsícem a chybovost mezi 46-60 %.

Jak DORA popisuje ve svém State of DevOps Reportu (2019), vznik DevOps pomohlo s výše uvedenými problémy, protože se zaměřuje na celý životní cyklus aplikace od vývoje přes testování a nasazení až po aktivní sledování a zpětnou vazbu. Tím je proces nasazování do produkčního prostředí mnohem lépe definovaný a předvídatelný, zároveň jsou eliminovány chyby způsobené lidským faktorem, protože tento proces je v ideálním stavu zcela automatický.

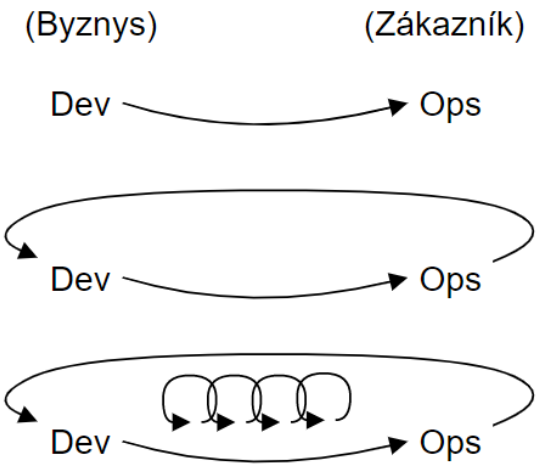


Obrázek 2: DevOps model. Zdroj: (Amazon Web Services 2020i)

Na obrázku 2 je možné vidět schéma DevOps, který obsahuje jednotlivé kroky potřebné pro nasazení aplikace do specifikovaného prostředí a zároveň je zde možné vidět nedílnou součást celého procesu, a to zpětnou vazbu v podobě neustálého monitorování a plánování budoucích kroků vedoucí k zlepšování samotného procesu.

2.1 Hlavní principy DevOps

Podle (Kim et al. 2016) existují tři hlavní principy, které jsou základními stavebními kameny pro celé hnutí DevOps tak, jak je dnes vnímáno. V kapitolách níže jsou tyto principy rozepsány do většího detailu pro lepší pochopení a argumentaci při případném zavádění DevOps praktik do organizací. Jejich oblasti jsou blíže ukázány na obrázku 3.



Obrázek 3: Tři hlavní principy. Zdroj: (Kim et al. 2016 str. 12), upraveno autorem

2.1.1 Princip toku

Princip toku se zaměřuje na zkrácení a zefektivnění toku práce mezi vývojem a operativou. Toho je dosaženo především zviditelněním prováděné práce, snížením aktuálně rozpracované práce a velikosti vyvíjených artefaktů, zvyšování kvality včasnou detekcí chyb, zamezením jejich postupu v toku a v neposlední řadě optimalizací celého procesu pro podporu stanovených cílů (Kim et al. 2016).

Jeden z problémů při vývoji softwaru je jeho zdánlivá neviditelnost, jelikož výsledkem není fyzický produkt, nýbrž nehmatatelný artefakt. Není tak možné snadno rozpoznat úzké body nebo místa, kde se nedokončená práce shlukuje. Díky tomu navíc nelze práci snadno přerozdělovat a opakovaně vracet zpátky do vývoje. Řešením tohoto problému je zavedením takových nástrojů, které umožní *větší transparentnost procesu vývoje* (toku). Mezi takové nástroje lze zařadit *Kanban board* nebo *Sprint board*, ve kterých lze sledovat vykonávanou práci, nastavovat pravidla pro tok úkolů a umožnit měřit dobu mezi vytvořením úkolu a jeho řádném dokončení (Kim et al. 2016). Například *Kanban board* umožňuje přehledně zobrazovat aktuálně prováděnou práci vývojovým týmem (KathrynEE 2018). To vše má velký vliv na celkovou transparentnost procesu vývoje, stejně jako je tomu v manufakturní výrobě při sledování produktu pohybujícího se z jednoho pracoviště na další.

Dalším konceptem je snaha *omezování množství rozpracované práce* (*angl. work in progress – WIP*). Při generování požadavků na vývoj dochází k obřímu vytváření šumu. Pokud nejsou stanovaná jasná pravidla, nové požadavky přichází z mnoha komunikačních

kanálů (telefonáty, emaily, chaty, eskalace atp.), což má za následek neustálé vyrušování v práci a přeskakování z jednoho úkolu na druhý. Tento způsob přiřazování nových úkolů má však negativní efekt, jakým může být například nedostatečná kvalita odvedené práce. Takové úkoly se musí později znovu přiřadit na vývoj kvůli neodladěným chybám, což způsobuje nejen časové ztráty, ale také finanční s tím spojené, které ovlivňují byznys jako celek. Proto je zde snaha co nejvíce omezit množství rozpracované práce, aby bylo zamezeno neustálé změně kontextu a tím pádem snížení chybovosti a potřebného času k opětovnému pochopení již jednou zkoumaného problému (Kim et al. 2016; Rehkopf 2020).

Snížení rozsahu práce je koncept, který má za cíl snížení doby práce na jednom úkolu. Pokud by se tento bod neprováděl, mohlo by docházet k dlouhým časům od zadání požadavku po jeho dokončení. To by znamenalo delší prodlevu pro zpětnou vazbu a větší pravděpodobnost pro zanesení chyb, protože by se pracovalo na velkém a komplexním problému naráz a případná chyba by se objevila až na konci celého procesu (Kim et al. 2016). Její odstranění by tak pochopitelně bylo mnohem nákladnější než při její včasné detekci díky menšímu rozsahu úpravy.

Snížení množství předávek má za cíl omezit operativu spojenou s prací na konkrétním úkolu. V situaci, kdy na jednom úkolu pracuje několik týmů dohromady, je nutné v celkovém čase práce zohlednit i čas, který nebyl vynaložen čistě na vývoj. Jedná se o různé formy předávání informací, signalizace, emaily, telefonování, schůzky atp. Všechny tyto formy komunikace mohou potenciálně vést k vytváření virtuálních front, kde práce čeká na odbavení. Snaha odbourat tento problém automatizací určitých úkolů nebo reorganizací týmů má za efekt zvýšení rychlosti toku práce od zadání po nasazení do produkčního prostředí (Kim et al. 2016).

Neustálá snaha nacházet a odstraňovat překážky v toku je dalším příkladem konceptu, který zmiňuje (Kim et al. 2016). Jedná se o problematiku úzkých míst, které způsobují hromadění práce před daným pracovištěm. V typických DevOps transformacích dochází k úzkým místům především při vytváření produkčních nebo jiných prostředích, při samotném nasazování do prostředí a při nastavování testů a jejich spouštění. Pokud se tyto problémy vyřeší, nejčastějším úzkým místem se stanou samotní vývojáři a lidé pracující na daném produktu. To je ovšem úzké místo, které potenciálně chceme mít, protože právě zde se vytváří hodnota pro koncového zákazníka. Z toho důvodu je snaha vše ostatní podřídit právě tomu, aby proces vývoje v malých týmech byl co nejrychlejší a bez zbytečných překážek popsaných výše.

Poslední praktikou je *odstranění plýtvání v toku.* Tím je myšleno vše, co nemá přidanou hodnotu pro koncového zákazníka. Mezi zmíněné plýtvání lze zařadit především částečně dokončená práce, extra procesy, které nemají vliv na výsledný produkt, častá změna kontextu práce (úkolů), prodlevy z čekání, defekty, nestandardní nebo manuální práce a v neposlední řadě nerealistické požadavky na zaměstnance (Kim et al. 2016).

2.1.2 Princip zpětné vazby

Tento princip se zaměřuje na rychlou zpětnou vazbu mezi operativou a vývojem. Tím se snaží předcházet katastrofickým dopadům chyb jejich včasnou detekcí, opravami a postupy, jak zamezit jejich opětovnému zavedení do aplikace.

Při práci na rozsáhlých systémech není v silách jednotlivců mít znalost nad všemi jeho částmi a chybovost způsobená budoucími změnami je nevyhnutelná. Proto je podle (Kim et al. 2016) nutné zavést takovou pracovní kulturu a procesy, které umožní *pracovat bezpečně uvnitř komplexních systémů*. Je nutné vytvořit takové podmínky, aby vývojáři mohli bezpečně upravovat systém a věděli, že chyby, které naleznou mají jasně definovaný proces a existuje na ně předem stanovaný proces jejich odstranění.

Včasná detekce chyb má za cíl vytvořit takové prostředí, ve kterém jsou neustále testovány změny v systému. Je zde snaha získávat co nejčastěji a nejrychleji zpětnou vazbu z ovlivněných oblastní. Jen díky tomu lze mít nejvíce informací pro to, aby bylo možné chybu efektivně a co nejdříve odstranit. V klasickém vodopádovém vedení projektu to mohou být i měsíce, než dostaneme zpětnou vazbu na změny provedené v systému a tato pozdní zpětná vazba může být velmi nákladná, protože chyba může být mnohem hlouběji v systému a náklady na její odstranění tím pádem mnohem vyšší (Kim et al. 2016).

S včasnou detekcí chyb je pevně spjata praktika *okamžitého řešení problému v době vzniku*. Tím je zabezpečeno především včasné zamezení větším škodám, které by chyba mohla způsobit v pozdějších fázích vývoje. Kromě toho také umožňuje lépe diagnostikovat problém a za dobré paměti všech zúčastněných nalézt řešení, které by později mohlo být mnohem náročnější, protože každý již přešel na řešení jiného problému a znalost původního problému se postupně ztrácí (Kim et al. 2016; Humble a Farley 2010).

Posouvání kontroly blíže ke zdroji je praktika, která se snaží eliminovat zbytečné plýtvání časem zúčastněných osob v procesu vývoje. Pokud je vyžadována kontrola od lidí, kteří jsou často zaneprázdněni nebo vytváření dokumentace, která je v momentu dokončení úpravy zastaralá, jedná se o plýtvání. Je tudíž vhodnější posunout kontrolu kvality blíže k vývoji, vytvořit povinnost na straně týmů a umožnit jim rychleji reagovat na problémy, které se mohou objevit (Kim et al. 2016). Jen tak lze zkrátit celkový čas potřebný pro nasazení opravy do požadovaného prostředí a tím zabezpečit vyšší úroveň kvality aplikace.

V neposlední řadě by při vývoji mělo být dbáno na pravidlo, že zákazník není jen ten, který za daný produkt zaplatí, ale také všichni, kteří se podílí na práci po daném stanovišti, jinými slovy operativa, která převezme práci po vývojářích. Měla by existovat snaha *pomoci optimalizovat tok pro následující pracoviště*. Tím opět posouváme kvalitu blíže ke zdroji, což v důsledku umožňuje rychlejší toku práce (Kim et al. 2016).

2.1.3 Princip kontinuálního vzdělávání se a experimentování

Poslední princip klade důraz na vytvoření takového prostředí, ve kterém je podporováno neustálé zdokonalování daného procesu pomocí vědeckých experimentů a učení se z vlastních úspěchů a chyb.

V okamžiku, kdy se objeví chyba nastává otázka, kdo ji způsobil a kdo je ve výsledku zodpovědný, je tedy hledán viník. To má ovšem velmi negativní efekt na celý tým a má často za následek, že chyby nejsou hlášeny až do okamžiku, než se stane katastrofické selhání. Proto je dobrou praktikou vytvořit *bezpečné prostředí a podpořit snahu učit se z vlastních chyb*. Jen tak lze místo hledání viníka spíše hledat způsob, jak podobné chybě zabránit nebo zcela předejít (Kim et al. 2016).

Velmi důležitou praktikou je *zdokonalování každodenní práce*. Pokud by tak nebylo prováděno, vývojáři se mohou dostat do situace, kdy pouze opravují chyby a nezbývá jim žádný čas na produktivní práci. Proto je nezbytné mít ve vývojovém cyklu dedikovaný čas na opravu a přepisování částí systému, které jsou dlouhodobě problémové a tím neustále udržovat systém aktuální a ve stabilním stavu (Kim et al. 2016).

Je nutné zmínit i praktiku *sdílení vlastních zkušeností napříč celou organizací*, aby znalosti, které nabyl jednotlivec, nezůstaly pouze u něj, ale byly někde zaznamenány a mohly být sdíleny napříč celou organizací. V okamžiku, kdy jiný tým řeší podobný problém má díky tomu veškeré podklady, aby jej byl schopný vyřešit mnohem rychleji, s větší jistotou a bez zbytečně se opakujících chyb (Kim et al. 2016).

Pokud je řeč o způsobu snížení rizika fatálního selhání systému a jeho pomalé opětovné zprovoznění, lze se na problém podle Kim et al. dívat ze dvou úhlů (2016). Tím prvním je snaha mít dostatečné záložní kapacity, které pokryjí případné výpadky. Problémem je ovšem jejich velký vliv na cenu. Druhým úhlem pohledu je zabudovat do každodenní práce neustálé ověřování připravenosti na případné potíže *zaváděním překážek do systému* a tím ověřovat připravenost na nenadálé události. Tím, že týmy lidí zvládnou rychle a efektivně reagovat na cvičné situace, jsou do systému zaváděna taková pravidla, která rizika snižují nebo týmy připravují na podobné reálné události bez potřeby nákladných záložních systémů.

Poslední praktikou kontinuálního vzdělávání se a experimentování je *posilování kultury vzdělávání lídry*. Lídři mají nesmírný vliv na celý iterativní proces v utváření takového prostředí, ve kterém běžní pracovníci nemusejí spoléhat na vlastní bezchybné rozhodování, nýbrž se mohou spolehnout, že v prostředí, ve kterém pracují, je takové chování podporováno a náležitě odměňováno. Jen tak lze zabezpečit, že firma se bude neustále posouvat ve svých kolektivních znalostech (Kim et al. 2016).

2.1.4 Shrnutí podkapitoly

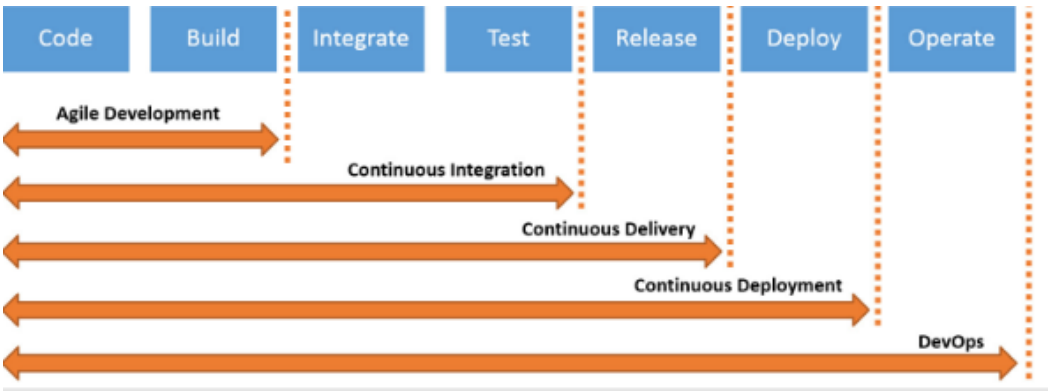
V podkapitole o hlavních principech DevOps bylo převážně čerpáno z publikace (Kim et al. 2016), ve které byly dobře a srozumitelně shrnuty hlavní praktiky stojící za DevOps. Tyto praktiky byly dále rozděleny na tři principy, kterými byly princip toku, princip zpětné vazby a princip kontinuálního vzdělávání se a experimentování. Praktiky postupně popisují způsoby, jak zefektivnit proces vývoje pomocí příkladů a jednoznačnou argumentací problematiky. Avšak rozhodně tím není myšleno to, že v případě absence některé praktiky se nelze bavit o DevOps. Nýbrž jde o formulování a ohraničení jednotlivých dílčích kroků, které mohou vést k efektivnější práci a rychlejšímu dodávání nové funkcionality produktu ke koncovému zákazníkovi. V tabulce 1 jsou pro úplnost praktiky přehledně sepsány.

Tabulka 1: Hlavní principy DevOps

Princip toku
Větší transparentnost procesu vývoje
Omezování množství rozpracované práce
Snížení rozsahu práce
Snížení množství předávek
Neustálá snaha nacházet a odstraňovat překážky
Odstranění plýtvání v toku
Princip zpětné vazby
Pracovat bezpečně uvnitř komplexních systémů
Včasná detekce chyb
Okamžitého řešení problému v době vzniku
Posouvání kontroly blíže ke zdroji
Pomoci optimalizovat tok pro následující pracoviště
Princip kontinuálního vzdělávání se a experimentování
Bezpečné prostředí a podpoření snahy učit se z vlastních chyb
Zdokonalování každodenní práce
Sdílení vlastních zkušeností napříč celou organizací
Zaváděním překážek do systému
Posilování kultury vzdělávání lídry

2.2 Agilní vývoj

Agilní vývoj jde ruku v ruce s praktikami DevOps, jak lze vidět na obrázku 4, existuje mezi nimi vzájemná provázanost. Dnešní organizace fungují v neustále se měnícím prostředí. Musí rychle reagovat na nové příležitosti na trhu, nové služby nebo nově vznikající konkurenty. Z toho důvodu je nutné umět rychle reagovat na změny při vývoji nového softwaru. Tradiční postupy při vývoji, kdy se nejdřív provádí kompletní specifikace požadavků, následně návrh, vývoj a v neposlední řadě testování není vhodný pro dnešní turbulentní prostředí. Ovšem agilní přístup řízení není vhodný pro všechny typy systémů. Podle Sommerville (2013) jsou těmito systémy například bezpečnostně kritické řídící systémy, u kterých je naopak žádoucí použít tradičnější přístup k řízení umožňující větší kontrolu při definování a později řízení takového projektu.



Obrázek 4: Dílčí kroky DevOps. Zdroj: (Watts 2017)

Na neustále se měnící prostředí podniku se snaží reagovat agilní metody vývoje. Ty jsou postavené tak, aby bylo možné software co nejdříve dostat do provozuschopného stavu. Následně se vyvíjí v inkrementech, při kterých se do softwaru přidávají nové funkce. Mezi základní vlastnosti agilních metod podle Sommerville patří zejména (2013):

1. Analýza, návrh a implementace nejsou oddělené, nýbrž se často prolínají. Většinou jsou specifikovány pouze základní vlastnosti systému, zbytek je dotvářen v průběhu vývoje.
2. Software se vyvíjí v inkrementech. Každá nová veze přináší funkce, které mají stakeholdeři k dispozici k testování a připomínkování. Nově vzniklé požadavky mohou být součástí pozdějších fází vývoje.
3. Rozhraní softwaru se často vyvíjejí pomocí interaktivních nástrojů, což umožňuje rychle měnit rozhodnutí a snadnější prototypování při samotném vývoji.

Zároveň autor představuje principy agilního vývoje:

1. *Zapojení zákazníka do procesu vývoje*. Především formou časté zpětné vazby na nově vzniklé funkce a určování priorit budoucího vývoje.
2. *Inkrementální dodávání*, které je konzultováno se zákazníkem.
3. *Lidé namísto procesů* je snaha upřednostnit jedinečnost členů týmu, aby měli možnost pracovat a inovovat tam, kde mají nejvíce zkušeností.

4. *Přijímání změn* je hlavním jádrem agilních metod. Je tudíž potřeba mít na paměti, že změny budou nastávat a je nutné je zohlednit v budoucích iteracích.
5. *Zachování jednoduchosti* klade důraz na neustálý tlak snížit vznikající složitost v systému s každou novou verzí.

Tento přístup k řízení umožňuje lépe navázat na principy toku, které pomáhají rychleji provádět změny ve vyvíjeném softwaru a jejich nasazení do produkčního prostředí ke koncovému zákazníkovi.

2.3 Continuous Integration

Aby bylo možné hovořit o DevOps jako celku, je nutné představit jeho další dílčí části. Jednou z nich je *Continuous Integration (CI)*, která je nezbytná pro zajištění důvěryhodnosti v nasazovanou verzi systému. Vzhledem k tomu, že v DevOps se aplikace nasazuje co nejčastěji, je nutné co největší část softwaru pokrýt automatickými testy, které mohou být spouštěné před každým nasazováním do prostředí. Tím je zajištěna alespoň základní důvěra v danou verzi, že je funkční a může se dostat až k zákazníkovi. Nedílnou součástí je i kontrola celého procesu sestavení aplikace, aby bylo možné toto sestavení provést kdykoliv ze zdrojového kódu bez nutnosti dalších zásahů a oprav. Ty by mohly potenciálně brzdit celý proces nasazení nové funkcionality do produkčního prostředí. (Humble a Farley 2010)

2.3.1 Typy testů

Je třeba rozlišovat několik základních typů testů, které se ve vývojovém cyklu objevují. S ohledem na zaměření této práce jsou níže popsány jen ty testy, které mají velký podíl při procesu Continuous Integration v rámci DevOps. Mezi ně patří především *testování jednotek, integrační, systémové, akceptační, smoke a end-to-end testování*. Záměrně je zde vynecháno regresní testování, protože v rámci Continuous Integration se regresní testy objevují v rámci testování nové funkcionality. Je zde totiž kladen velký důraz na neustálou kontrolu, že nová funkcionalita neovlivnila jinou, díky které by systém byl nefunkční a nebylo možné jej nasadit.

Jednotkové testování má na starost samotný vývojář, který testuje jednotlivé třídy, procedury či metody nezávisle na sobě, izolovaně a tím dokázat funkčnost dílčí jednotky. (Roudenský a Havlíčková 2013) Metody jsou při psaní testu volány s jasně danými parametry a kontroluje se jejich výstupní hodnota. Zároveň by se měl objekt v rámci testů dostat do všech jeho stavů a kontrolovat správnost jeho atributů. (Sommerville 2013)

Na jednotkové testování navazuje *integrační testování*, které testuje funkčnost dvou a více jednotek (modulů) tvořeného systému. Je ovšem nutné zmínit, že se může objevit i pojem systémové integrace, kterým je často myšleno testování integrace s externími systémy. (Roudenský a Havlíčková 2013)

Mezi *systémové testování* jsou řazeny především funkční testy, bezpečnostní testy, testy uživatelského rozhraní, výkonnostní testy a další. Jedná se o tedy testy, které ověřují

funkcionalitu systému jako celku (Roudenský a Havlíčková 2013). Tyto testy by měly poskytnout dostatečné podklady pro ověření funkčnosti softwaru a dodat důvěru pro další kroky v rámci DevOps.

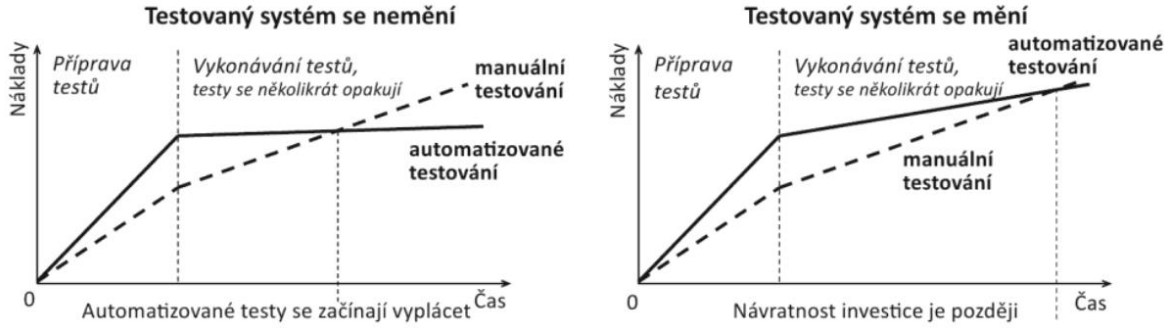
Smyslem *akceptačního testování* je ověřit, zda software splňuje kritéria stanovená zákazníkem systému (Roudenský a Havlíčková 2013). V rámci Continuous Delivery je nutné tyto testy spouštět pokaždé, kdy kód projde základními testy (jednotkové, integrační a systémové), aby bylo možné rozhodnout, zda daná verze může být v pozdější fázi nasazená do prostředí automaticky, nebo se celý proces zastaví (Humble a Farley 2010). Tato tvrzení jdou zdánlivě proti sobě, ovšem jde především o zautomatizování akceptačních testů, které tím pádem není nutné delegovat na zákazníka a ten se může více soustředit na návrh a specifikaci nových požadavků. Nicméně zákazník by měl mít zachovanou pravomoc schválit výsledky automatických akceptačních testů.

Smoke testy ověřují, že základní funkcionality systému jsou stále funkční, že je stabilní a má smysl jej dále testovat. Jsou zaměřeny především na scénáře s dobrým koncem (*angl. happy scenarios*), neověřují tak případy, při kterých uživatel zadává chybná data do formulářů apod. Jejich význam je také při nasazení do nového prostředí, při přesunu z jednoho prostředí do jiného, po nasazení do produkčního prostředí nebo při úspěšném sestavení. (Bureš et al. 2016)

V neposlední řadě je nutné zmínit *end-to-end testování (E2E)*. Jeho aplikování může být v různých variantách, nicméně cílem je sledovat nějakou entitu od jednoho konce do druhého. Může se jednat o test, který zkouší přihlášení do systému přes vyhledávání a nákup až po odhlášení ze systému. Ověřuje aplikaci tak, jak ji bude samotný zákazník používat. (Roudenský a Havlíčková 2013)

2.3.2 Automatické vs. manuální testování

Velkou část výše uvedených testů lze automatizovat, ale nikoliv všechny. Proto se nabízí otázka, kdy a jak automatizovat zmíněné typy testů. Jedním aspektem takového rozhodnutí je bezesporu návratnost investice do vytvoření automatických testů. Tato podkapitola je spíše zaměřena na konkrétní případy, kde je automatizace užitečná a kde nikoliv. Samotné porovnání nákladů je dost individuální a nelze s jistotou říct, zda zavedení automatizace přinese úspory do projektu nebo se v konečném důsledku náklady vyrovnají manuálnímu testování, jak je zobrazeno na obrázku 5.



Obrázek 5: Návratnost investice do automatizace v čase. Zdroj: (Bureš et al. 2016)

Podle Bureš et al. (2016) je první velkou výhodou automatického testu jeho *rychlost provedení*. Takový test lze tak mnohem častěji opakovat než manuální. Společně s tím je jejich provedení vždy *stejné a bez chyb*, ovšem nedokáže si poradit se změnou vstupních dat, která může z defektu vyplývat. S tou si dokáže poradit pouze tester pomocí své intuice a zkušeností. Již při samotném *vytváření automatického testu dochází k testování softwaru* (Bureš et al. 2016). Jinými slovy již při psaní testu může být odhaleno velké množství defektů, které lze včas opravit a následně napsat takový test, který bude možný spouštět opakovaně, aby předcházel zavedení stejných defektů v budoucnosti.

Možnou nevýhodou může být zase *přílišná specifičnost automatického testu*. Jinými slovy automatický test může selhat i v případě, kdy se jen nepatrně posune grafický prvek na obrazovce. Tester si s takovou situací hravě poradí, protože zná systém a proces dostatečně dobře, automatický test ovšem tuto událost automaticky vyhodnotí jako selhání. Někdy může být takové chování na místě, jindy může být na škodu. Další velkou nevýhodou zmiňovanou Burešem et al. (2016) je nemožnost pokročilé analýzy výskytu chyby. Přestože se nástroje snaží reportovat chyby formou zaznamenání obrazovky při výskytu chyby například framework Cypress (Magni 2020) nebo Protractor (Gunarathna 2018), nikdy nenahradí skutečného testera, který dokáže lépe pochopit kontext a provést pokročilejší analýzu problému. Předchozí zmíněné má však ještě jeden problém, nahlášená chyba nutně nemusí znamenat defekt testovaného softwaru. Může se jednat o chybu, která vznikla změnou systému, a to musí v konečném důsledku rozhodnout samotný tester. (Bureš et al. 2016)

V neposlední řadě se nesmí zapomínat na různé kvalifikační nároky na pozici vývojáře automatických testů, který musí oproti testeru provádějící manuální testy umět programovat ve skriptovacím jazyce a znát potřebné nástroje nebo metodiky používané na projektu. (Bureš et al. 2016)

2.3.3 Shrnutí podkapitoly

Podkapitola o *Continuous Integration* byla věnována teorii z oblasti testování, jelikož testy tvoří jeden ze základních kamenů pro celý DevOps proces. Jakmile není dostatečná důvěra ve vyvíjený software, nelze od zákazníka očekávat jeho náklonost k nasazování do produkčního v častých intervalech blížících se nasazování několikrát denně. Jen za pomocí dat získaných z testování lze zákazníka přesvědčit o funkčnosti softwaru. Důležité je si tak uvědomit, že automatické testování nenahradí skutečného testera, jen mu může značně usnadnit repetitivní práci a ten se tak může více soustředit buď na vytváření automatický testů, nebo na provádění průzkumných (*angl. exploratory)* testů, které vyžadují hlubokou znalost testovaného softwaru. Vedle toho je vhodné testovat i návaznosti na jiné systémy, popřípadě kontrolovat, že výsledné sestavení aplikace je bezchybné a lze jej tak případně nasadit do produkčního prostředí.

2.4 Continuous Delivery a Deployment

Dalším krokem při zavádění DevOps je *Continuous Delivery (CD)*, které navazuje na *Continuous Integration (CI)* popsané v předchozí podkapitole. Na tomto místě je vhodné popsat hlavní rozdíl mezi *Continuous Delivery (CD)* a *Continuous Deployment (také CD)*, aby se pojmy dále v práci nezaměňovaly při použití zkratek nebo spojení CI/CD.

Rozdíl mezi Continuous Delivery a Continuous Deployment je nepatrný, nýbrž velmi důležitý. Jak Humble a Farley ve své knize věnované Continuous Delivery (2010) popisují, Continuous Delivery je proces, při kterém vývojáři pracují po malých částech. Následně svou práci pravidelně nahrávají na hlavní vývojovou větev, která je v provozuschopném stavu (lze z ní vytvořit produkční sestavení) a finální nasazení do produkčního prostředí je otázka kliknutí tlačítka. Tento postup vystihuje podstatu Continuous Delivery. Vývojáři tak dostávají rychlou zpětnou vazbu o svém kódu, a v případě regresních, bezpečnostních nebo výkonnostních chyb dokážou promptně reagovat a vytvořit záplatu v krátkém čase. To má za pozitivní efekt to, že hlavní vývojová větev je neustále v provozuschopném a stavu připraveném na nasazení.

Oproti tomu Continuous Deployment je část procesu, kdy se do produkce nasazuje na pravidelné bázi buď ručně za pomocí vývojáře (Dev), operativy (Ops) nebo dokonce automaticky každou commit změnou ve zdrojovém kódu (Humble a Farley 2010). Důležitá je zmínka v pravidelnosti, která tento proces odlišuje od Continuous Delivery.

Z výše uvedeného je patrné, že Continuous Delivery je prerekvizitou Continuous Deployment stejně jako je Continuous Integration je prerekvizitou pro Continuous Delivery (Kim et al. 2016). V této práci se proto používá spojení Continuous Integration a Continuous Deployment (CI/CD), které lépe vystihuje jednotlivé kroky v nadcházející metodice. Přestože termíny se mohou lišit o různé výklady. Dokonce jak Humble a Farley (2010) uvádí, každá organizace by si měla definici upravit podle svých potřeb, protože důležitý je výstup z celého procesu, kterým je nasazování s nízkým rizikem a na kliknutí tlačítka, nikoliv se držet definic a předem daných postupů, které mohou být v důsledku silně svazující.

Je dobré zmínit ještě poslední pojem, kterým je Deployment Pipeline. V abstraktním pojetí se jedná o automatizovaný proces, který umožňuje dostat změnu ze zdrojového kódu až ke koncovému uživateli (Humble a Farley 2010). Tento proces obsahuje vytvoření sestavení, jednotkové, E2E, regresní a jiné testování až po připravení sestavení pro nasazení do prostředí, případně provedení tohoto nasazení automaticky.

V textu výše je vždy uvedeno nasazování do prostředí, přitom není blíže specifikováno, zdali se jedná o testovací nebo produkční. Je to záměrné, protože v tomto případě by vše mělo být obsaženo v konfiguraci přímo ve zdrojovém kódu a nasazení do produkčního prostředí by měla být pouze otázka výměny této konfigurace a celý proces by měl běžet naprosto stejným způsobem (Humble a Farley 2010). Tímto způsobem máme zajištěno, že nasazení, které bylo provedeno několikrát do testovacího prostředí mají vývojáři, popřípadě operativa, nacvičeno tak dobře, že není nutné mít velké obavy při nasazování do produkčního prostředí.

Před tím, než se lze bavit o Continuous Deployment je nutné si celý proces navrhnout a připravit se na možné situace, které v rámci nasazování aplikace do zvoleného prostředí mohou nastat. Jedná se o vytvoření strategie nasazování, způsob vrácení předchozího sestavení a kritické záplatování chyb na produkčním prostředí. Jedině v ten okamžik dává smysl celý proces zautomatizovat do bodu, kdy všichni zúčastnění pouze dostávají informace o běhu procesu a v případě, že se vyskytne chyba dobře vědí, jak na ni reagovat.

2.4.1 Strategie nasazování

Vzhledem k důležitosti procesu nasazování v každé firmě, ať už se jedná o agilní nebo tradiční přístup k řízení, je dobré si uvědomit, že správné nasazení nelze provést bez předem připravené strategie.

Nejdůležitější krok při stanovení si strategie nasazování je shromáždit všechny stakeholdery, kteří mají podíl na aplikaci. Hlavní pointou tohoto setkání by mělo být všestranné pochopení ohledně nasazování a správy dané aplikace, které je zachyceno právě ve zmíněné strategii (Humble a Farley 2010). Tento dokument je následně pravidelně aktualizován a upravován v průběhu celého životního cyklu, protože procesy potřebné v raných fázích se budou značně lišit v pozdějších fázích, kdy je aplikace již a pouze udržována například formou bezpečnostních záplat. Taková strategie může podle Humble a Farley zahrnovat například následující kroky, činnosti nebo účastníky (2010):

- Lidé zodpovědní za nasazování do všech používaných prostředí.
- Strategie správy zdrojů a konfigurace.
- Technologický základ potřebný pro nasazování aplikace.
- Plán aplikování jednotlivých kroků v procesu nasazování.
- Výčet prostředí potřebných pro jednotlivé typy testování (akceptační, integrační, uživatelské) a způsob přesunu sestavení mezi těmito prostředími.
- Popis procesů spojených s jednotlivými kroky při nasazování (např. způsob řízení změn nebo potřebná schválení).
- Způsob monitorování aplikace v jednotlivých fázích.
- Popis integrace na externí systémy (v jakém okamžiku toto testování probíhá a jak řešit případné problémy, popřípadě s kým).
- Detailní logování probíhaných událostí a kroků.
- Proces archivace dat při aktualizacích.
- Proces prvního nasazení do produkčního prostředí.
- Jak spravovat produkční defekty, popřípadě jak nahrávat záplaty aplikace.
- Jak se budou provádět aktualizace produkčního prostředí včetně migrací dat.
- Způsob podpory aplikace.

Při vytváření takové strategie se objeví spousta funkčních i nefunkčních požadavků na aplikaci, které jsou jak architektonického rázu, tak způsobu zvládání konfigurace a přidělování potřebného hardwaru v používaných prostředích. Tyto požadavky by měly být postupně přidávány do vývojového plánu v momentě jejich identifikace (Humble a Farley 2010).

Důležitou oblastí je zmíněné první nasazení do prostředí, protože zahrnuje spoustu kroků, které je nutné provést nad rámec pravidelného aktualizování aplikace při budoucím nasazování. Jedná se například o procesy spojené s vytvořením prostředí, alokací hardwaru a softwaru, nastavení konfigurace systému nebo vytvoření a připojení se k databázi.

2.4.2 Způsob vrácení chybného nasazení

Přestože je v předchozích kapitolách kladen velký důraz na proces testování aplikace, může se nasadit chybný kód, který způsobí neočekávané chyby nebo dokonce pád celé aplikace. V takových případech je potřeba mít připravený proces zvrácení takové chyby (*angl. rollback*), aby se vývojový tým mohl v klidu na chybu podívat a zároveň doba trvání vyřešení takového defektu neohrožovala koncové uživatele, kteří aplikaci potřebují používat. Níže jsou uvedené různé způsoby řešení chybného prostředí.

Nasazení poslední funkční verze

Podle Humble a Farley (2010) se jedná o jeden z nejjednodušších způsobů, jak opravit chybnou verzi, který spočívá v nasazení předchozího sestavení, které bylo funkční. Nicméně se jedná o způsob, při kterém je aplikace nedostupná, protože proces nasazení většinou trvá fixní čas. Zároveň nelze jednoduše chybu ladit, protože při opětovném nasazení předchozí verze dojde k přepsání souborů a chyba tak nemusí být jednoduše dohledatelná. Zároveň pokud dojde k znovu nasazení předchozí verze databáze, například ze zálohy provedené před samotným nasazení, tak se mohou ztratit data zaznamenaná v době běhu nové verze aplikace.

Obnovení funkční verze aplikace bez výpadku

Tento způsob anglicky označován jako *zero-downtime releases* nebo *hot deployments* spočívá v jednoduchém přepnutí aktuálně používané verze pomocí konfigurace (Humble a Farley 2010). Oprava je téměř okamžitá, proto označovaná jako *zero-downtime* (bez výpadku aplikace, bez znatelného dopadu na koncové uživatele). Na příkladu webových služeb komunikujících výhradně přes REST API se může jednat o rozlišení na úrovni URL adresy, kdy více verzí běží souběžně a v případě chybně nasazené verze se lze okamžitě vrátit na libovolnou předchozí verzi ať už formou konfigurace nebo přesměrováním uživatelů na funkční verzi.

Blue-Green nasazení

Při tomto způsobu existují dvě identické prostředí, mezi kterými lze snadno přepínat. Princip je založený na tom, že v jednom okamžiku jsou uživatelé vždy přesměrování pouze na jedno prostředí (nazývejme jej např. Blue). Na druhém (Green) prostředí se nasadí aplikace, otestuje se a potvrdí se, že je vše v pořádku. V ten okamžik jsou všichni uživatelé přesměrování na druhé (Green) prostředí a sleduje se, zdali je všechno v pořádku. Pokud se vyskytne nějaká chyba, je to jen otázka nastavení směrování, aby se uživatelé přesměrovali na první (Blue) prostředí a chyba se mohla v klidu na druhém (Green) prostředí replikovat, odladit a opravit (Humble a Farley 2010). Tento model je díky tomu mnohem stabilnější, protože v rámci jednoho prostředí se aktualizují všechny části aplikace a v případě potíží se

lze rychle vrátit k předchozí verzi a problém v klidu vyřešit. Nicméně jsou kladeny větší nároky na dostupnost hardwaru, infrastruktury a licencí.

Canary releasing

Tento model je velmi podobný Blue-Green nasazení, nicméně novou verzi uvolňuje nejdříve pomaleji mezi vybranou část uživatelů namísto přesměrování všech (Sato 2014). Výhody oproti předchozímu způsobu jsou například v tom, že lze provádět tzv. A/B testování (měření nových funkcí na koncových uživatelích a jejich případný benefit v předem stanovených metrikách). Na základě toho se lze následně rozhodnout, jestli nová verze je vhodná pro zpřístupnění celé základně uživatelů, nebo je nutné provést další úpravy a znovu provést toto testování (Humble a Farley 2010).

2.4.3 Kritické záplaty chyb na produkčním prostředí

V každé aplikaci se objeví chvíle, při které se v produkčním prostředí objeví kritická chyba vyžadující rychlé řešení. Podle Humble a Farley (2010) by nikdy nemělo docházet k obcházení nastaveného procesu nasazení, přičemž vycházejí z jednoduché úvahy. V okamžiku, kdy je chyba opravena přímo na produkčním prostředí se může stát, že taková úprava nebude dostatečně zaznamenána a hlavně otestována, tudíž může potenciálně vést k budoucím problémům, které bude těžké replikovat a případně záplatovat. Takové prostředí se následně ocitne ve stavu, který nelze jednoduše replikovat a je téměř nemožné navodit stejné podmínky pro případné ladění zdrojového kódu. Proto je kladen velký důraz, aby každá změna, včetně kritických záplat vždy prošla stejným procesem nasazení jako tomu je při nasazování nové funkcionality.

Alternativou takového řešení je navrácení se do předchozí verze aplikace, ve které se chyba ještě nevyskytovala, což bylo popsané v přechozí podkapitole.

2.4.4 Continuous Deployment

Přestože je kapitola 2.4 pojmenovaná jako Continuous Delivery a Deployment, tak až po připravení strategie, plánu vrácení předchozí verze, vyřešení procesu opravy kritických chyb na produkčním prostředí a následném zautomatizováním celého procesu nasazování do produkčního prostředí se jedná o Continuous Deployment. Důležité je na tomto místě zmínit, že Continuous Deployment není vhodný pro všechny typy aplikací. V některých případech není vhodné a ani žádoucí podobný systém zavádět, protože jeho benefity nemusí být dostačující a může způsobit více škody než užitku. Někdy jsou procesy v organizaci nastavené takovým způsobem, který zcela vyvrací průběžné nasazování do produkčního prostředí například potřebnými schváleními ze strany stakeholderů nebo architekturou danou zákaznickými systémy. Do takových systémů například nemusí být umožněn přímý přístup, a tudíž se aktualizace se musí nadále provádět ručně.

2.5 Shrnutí

Tato kapitola byla zaměřena na obecné principy ohledně DevOps a jeho dílčích kroků v podobě Continuous Integration, Delivery a Deployment procesu. Nejdříve byly podrobně rozepsány tři hlavní principy týkající se DevOps: princip toku, princip zpětné vazby a poslední princip kontinuálního vzdělávání se a experimentování. Každý z těchto principů obsahoval několik praktik, které blíže popisují důležitost zmíněného principu.

Následně byla pozornost věnována oblasti testování a integrace, která je nedílnou součástí celého DevOps procesu. Jedná se o část s názvem Continuous Integration, kde byly popsány základní pojmy z oblasti testování a jejich důležitost v rámci životního cyklu aplikace. Dále zde byly popsány způsoby, jak změny ve zdrojovém kódu kontrolovat z hlediska jejich integrace v rámci aplikace.

Poslední část byla věnována procesu Continuous Delivery a Deployment, při kterých dochází k přípravě a následně provedení nasazení do produkčního prostředí. Jedná se o poslední krok v celém DevOps, který pomyslně cestu vytvoření nové funkcionality završuje. Byl zde podrobně popsán celý proces, který umožňuje v ideálním stavu zcela automaticky bez jediného uživatelského zásahu vývojářů nebo operativy nasadit aplikaci do produkčního prostředí. Ovšem to za předpokladu, že všechny dílčí kroky proběhnou úspěšně a bez chyb. Díky těmto krokům mají prováděné změny v aplikaci větší stabilitu a vývojáři pomyslnou záchrannou síť, že provedené změny po dokončení celého procesu nasazení nezpůsobí kritický pád aplikace, a pokud ano, existují jasně dané procesy, jak takovou změnu promptně a s jistotou odstranit.

Tímto kapitola naplnila první dílčí cíl vymezit pojem DevOps ve spojení s Continuous Integration, Delivery a Deployment procesem.

3 Nástroj Bitbucket pro správu vývoje, Continuous Integration a Delivery

Kapitola se věnuje nástroji Bitbucket, který umožňuje orchestraci celého procesu. Nejdříve je nástroj popsán obecně a následně je specifikováno, které části jsou vhodné pro podporu DevOps.

3.1 Představení nástroje Bitbucket

Bitbucket je webový nástroj pro hostování a správu zdrojového kódu. Nástroj je vlastněn společností Atlassian, která má mimo jiné ve svém portfoliu nástroje jako Jira, Confluence nebo Trello. Bitbucket je vhodnou volbou, pokud je ve firmě zavedena minimálně Jira pro správu požadavků a vedení projektů, protože jeho integrace do celého procesu je na prvotřídní úrovni. (Bitbucket 2020)

Mezi hlavní vlastnosti nástroje patří zejména kontrola kódu (*angl. Code reviews)*, práva na základě vývojové větve (*angl. Branch permissions)* a procesy (*angl. Pipelines).* Dále jsou v textu používány anglické ekvivalenty, které lépe propojují názvy služeb nebo funkcí ať už v nástrojích Bitbucket, Jira, git, nebo později u popisování služeb na platformě Amazon Web Services.

Code reviews je praktika, která se ve vývojovém prostředí používá k vzájemné kontrole kódu mezi vývojáři. Má za cíl nalézt chyby v kódu a tím zvýšit kvalitu provedených změn díky pohledu jiné osoby, která může odhalit potenciální problémy. Mimo jiné to má i efekt vzájemného učení a získávání zkušeností mezi členy týmu. (Kim et al. 2016)

Branch permissions umožňuje nastavit pravidla na základě jednotlivých typů větví. Těmi mohou být například master, feature nebo bugfix. To může být využito pro specifikaci uživatelských oprávnění, které mohou omezit provádění úprav nad zmíněnými větvemi. Například mohou být nastavená pravidla tak, že do hlavní vývojové větve master nemohou přispívat juniorní pracovníci. Tato pravidla pomáhají především snížení chybovosti a umožnění snadnějšího toku změn z vývoje, respektive zdrojového kódu, do produkčního prostředí bez nutnosti častých zásahů a oprav.

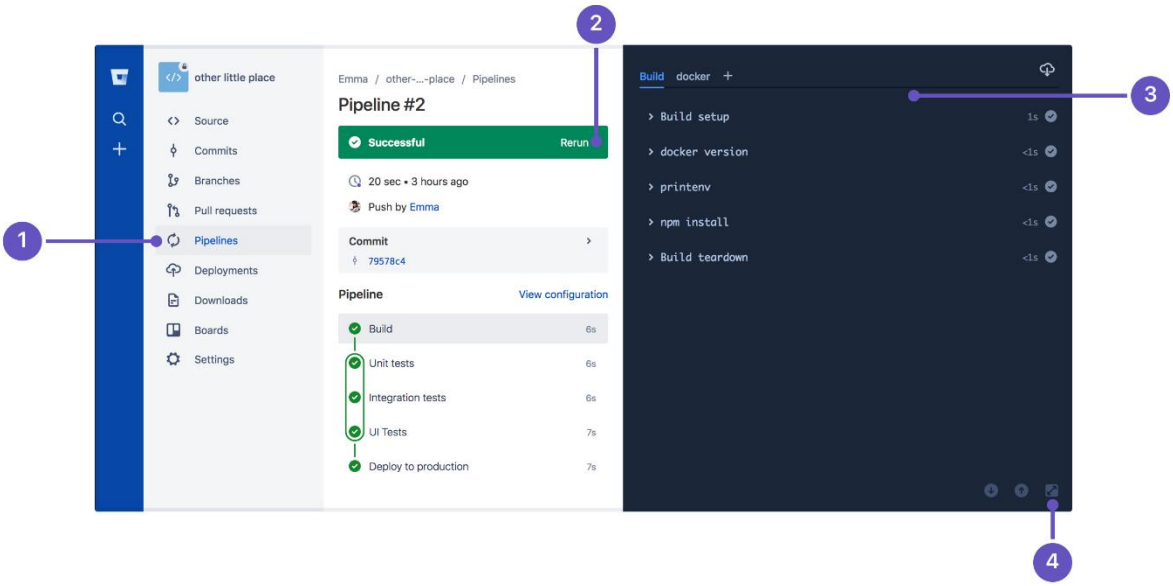
Bitbucket umožňuje vývojářům pracovat s různými modely větvení kódu (Atlassian 2020a). To je důležité především z toho důvodu, že na jednotlivé typy větví lze nastavit jiné procesy v rámci CI/CD. Lze například vynutit, že při novém commitu na feature větvi dojde pouze ke spuštění jednotkových testů, ovšem při integraci zpět na hlavní vývojovou větev je provedena celá sada testů včetně integračních a E2E. To má za následek zrychlení zpětné vazby vývojáři díky rychlejšímu provedení úloh vyvolaných provedenou změnou. Zároveň to umožňuje velkou flexibilitu při stanovení pravidel na základě způsobu vývoje a vzájemné kontroly kódu.

Jedním z kontrolních mechanismů při vývoji je tzv. pull request požadavek. Ten se provádí při potřebě integrace nových změn z libovolné na hlavní větev. Díky tomu lze otestovat funkčnost celé aplikace ještě před tím, než je změna integrována. V případě úspěšného provedení testů lze tento požadavek schválit a integrace díky tomu může proběhnout automaticky.

3.2 Pipelines a Bitbucket

Pod pojmem Pipelines si lze představit úlohy obsahující jednotlivé kroky, které se provádějí po předem definované akci. Tou může být například nový commit na libovolné větvi, chyba při spuštění testů nebo nový pull request požadavek. Právě tyto předem definované akce mohou spouštět různé sady testů, jejichž výsledkem může být nejen informace o úspěšném dokončení testů, ale také vytvoření sestavení (*angl. build*) aplikace a případné spuštění automatického nasazení do produkčního prostředí. (Atlassian 2020a)

Na obrázku 6 lze vidět ukázku běhu úlohy v prostředí Bitbucket, konkrétně je zde záznam jednotlivých kroků, přičemž jakákoliv chyba způsobí zastavení běhu a celá úloha je následně přerušena. Tím je zajištěno, že poslední krok, který většinou obsahuje vytvoření sestavení a nahrání do prostředí nenastane.



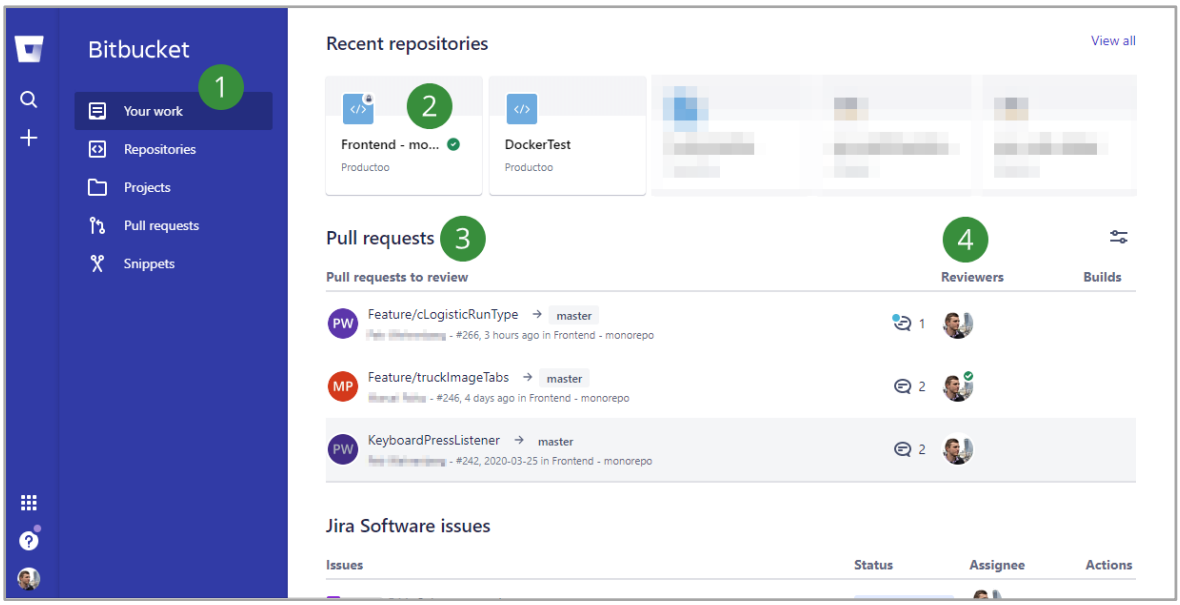
Obrázek 6: Ukázka běhu konkrétní úlohy. Zdroj: (Atlassian 2020a)

3.3 Deployments a Bitbucket

Deployments je v pojetí Bitbucket Pipelines poslední možnou úlohou, která provede automatické nasazení do předem definovaného prostředí. Zde je důležité zmínit, že v rámci Bitbucket nelze nasadit aplikaci, nýbrž je nutné k tomu využít služby třetích stran. Jednou z nich je právě platforma Amazon Web Services, která poskytuje portfolio služeb umožňující napojení na poslední krok v rámci Bitbucket nástroje. Více je popsáno v následující kapitole věnované Amazon Web Services. (Atlassian 2020a)

3.4 Průvodce nástrojem Bitbucket na reálné aplikaci

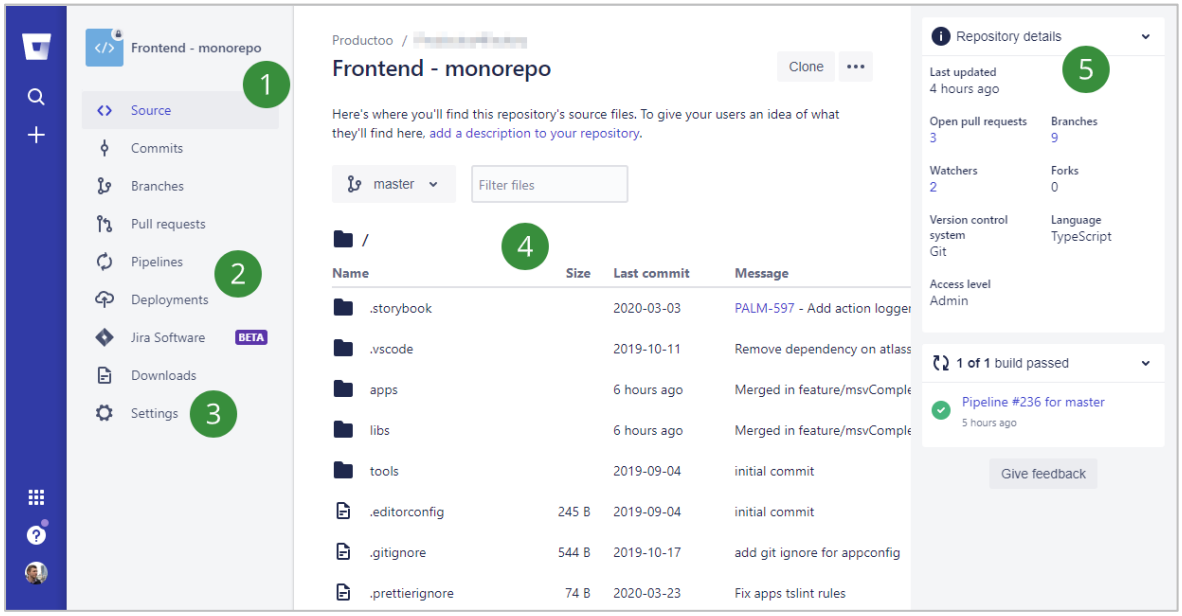
Po založení účtu a přihlášení se do nástroje je k dispozici dashboard uživatele, kde lze přehledně vidět poslední události, které nastaly v rámci portfolia aplikací, do kterých má uživatel přístup. Na obrázku 7 je uveden příklad z prostředí nástroje. Nepodstatné části jsou pro tuto diplomovou práci záměrně rozostřeny kvůli snadnějšímu pochopení textu a zároveň důvěrnosti informací poskytnuté organizací, ve které probíhalo ověření této metodiky a odkud jsou ukázky pořízeny.



Obrázek 7: Dashboard aplikace Bitbucket. Zdroj: autor

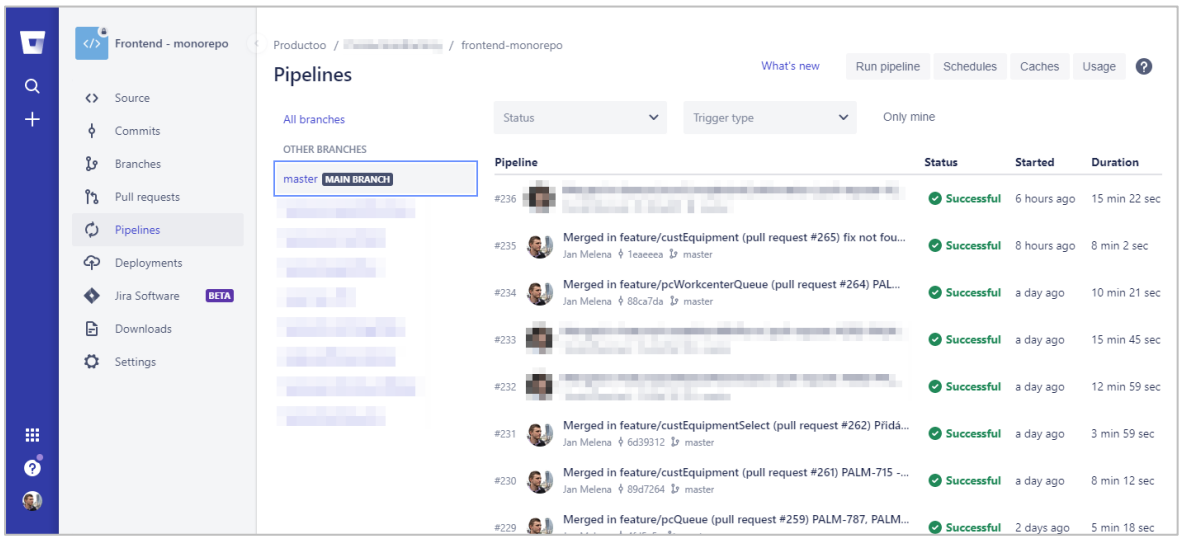
V levé části (1) lze nalézt menu, ve kterém je rychlý přístup ke všem potřebným funkcím nástroje. Jedná se o dashboard, repozitáře, projekty, pull requests a snippets. V horní části (2) se nachází dostupné repozitáře, u kterých je ihned vidět poslední stav běhu Pipelines. Na obrázku výše je uveden příklad úspěšného dokončení posledního běhu a aplikace nacházející se ve stavu, ze kterého lze kdykoliv udělat produkční sestavení. (3) Pull Requests neboli požadavky od ostatních pracovníků, kteří žádají o integraci svých změn do hlavní vývojové větve. Je zde uveden autor, zdrojová větev a časová známka. (4) seznam uživatelů, kteří musí dát souhlas se zařazením změn do hlavní vývojové větve, bez kterého nelze změny integrovat. Jedná se o code reviews od vybraných uživatelů.

Po načtení konkrétního repozitáře, což je možné vidět na obrázku 8, konkrétně pod názvem *Frontend – monorepo*, lze najít důležité informace o repozitáři a nastavení týkající se procesu CI/CD. Pod (1) se skrývá menu, které umožňuje přístup k různým pohledům na zdrojový kód a zároveň nabízí možnost přejít na obrazovku spuštění jednotlivých (2) Pipelines, popřípadě přejít do (3) nastavení. V prostřední části (4) lze vidět soubory aplikace, jejich poslední změnu a zprávu z posledního commitu. V pravé (5) části lze vyčíst důležité informace týkající se repozitáře jako je časová známka poslední aktualizace, počet otevřených pull request požadavků nebo počet aktuálních větví. Zároveň lze vidět poslední stav běhu Pipelines.



Obrázek 8: Ukázka repozitáře aplikace. Zdroj: autor

Na obrázku 9 je možné vidět přehled všech běhů Pipelines pro daný repozitář. Lze používat filtry pro konkrétní větve, uživatele nebo stav běhu (úspěšný/neúspěšný/opakované spuštění). Díky tomu je možné historicky dohledat veškeré běhy, které nastaly a kdykoliv je v případě potřeby spustit znovu nebo se podívat na podrobný výpis běhu, který bylo možné vidět na obrázku 6 v předchozí podkapitole 3.2. Podrobné nastavení pro účely metodiky budou popsány blíže v nadcházejících kapitolách.



Obrázek 9: Ukázka běhů pipelines. Zdroj: autor

3.5 Shrnutí

Kapitola měla za cíl představit nástroj Bitbucket a přiblížit jeho možnosti v rámci tvorby metodiky jakožto hlavní nástroj pro orchestraci celého procesu od změny v aplikaci až po nasazení do produkčního prostředí. Největší důraz byl kladen na Bitbucket Pipelines a Deployments, které jsou následně použity v CI/CD procesu. Tímto naplňuje druhý dílčí cíl. Ukázky z nástroje v podobě obrázků byly převzaty z reálného prostředí vývoje aplikace, ve kterém byla tato metodika ověřována. Pro lepší pochopení textu a potažmo celého nástroje byly doprovázené obrázky pořízeny až po provedení nasazení a ověření metodiky. Informace nepodstatné pro tuto práci byly rozmazány pro zachování důvěrných dat organizace, která poskytla nástroj a souhlasila se sdílením ukázkových dat.

4 Amazon Web Services jako platforma pro podporu Continuous Deployment

Tato kapitola popisuje jednotlivé služby, které jsou dostupné na Amazon Web Services (dále jen AWS) a zároveň podporují procesy DevOps v rámci vytvářené metodiky. Popisuje obecnou architekturu a jednotlivé služby, které lze použít v kombinaci s Bitbucket Pipelines pro splnění posledního kroku nasazení aplikace do produkčního prostředí, kterým je Continuous Deployment.

AWS vzniklo v roce 2006, kdy Amazon začal nabízet IT infrastrukturu ostatním organizacím. Tím umožnil snížit počáteční náklady na infrastrukturu s tím, že služba roste společně s organizací (Amazon Web Services 2020a). Obecně se tomuto principu říká Cloud computing (zkrácené cloud), který díky sdílení výpočetního výkonu umožňuje několik zásadních věcí (Davis a Daniels 2016; Amazon Web Services 2020h):

- *Dostupnost nových technologií*, díky čemuž je snazší zkoušet nové věci a posouvat svůj byznys dopředu.
- Vysoká *elasticita*, díky které není nutné vlastnit velký výpočetní výkon, jelikož ten si lze dokoupit v případě potřeby a jeho dostupnost je téměř okamžitá.
- *Úspory v nákladech*, protože s použitím cloudu se platí pouze za výkon, který se spotřebuje.
- *Možnost nasazení po celém světě* v řádech minut. Díky čemuž se opět není nutné starat o cokoliv jiného kromě vlastní aplikace. Tím pádem odpadá i starost o fyzické servery v daných lokalitách.

V rámci této metodiky je využitý jen zlomek nabízených služeb, které doplňují Bitbucket Pipelines. Je ovšem vhodné zmínit, že samotný AWS disponuje konkurenčními službami s názvy CodeCommit, CodeBuild, CodePipeline a CodeDeploy (Amazon Web Services 2020f), které mohou zcela nahradit Bitbucket. Nicméně z prostředí firmy, ve které vznikla potřeba návrhu metodiky se s těmito službami nepracuje. Lze je však postupně zkoušet a různě zaměňovat podle případných potřeb dané organizace. Liší se pouze v konfiguraci, v principu zůstávají stejné. Díky tomu lze čerpat z níže navržené metodiky a její kroku případně pouze upravit, aby vyhovovaly podmínkám zmíněných služeb. Všechny nástroje, které pracují na principu Pipelines umožňují jednotlivé prvky (kroky) vyměňovat. Lze tak například spouštět testy na vlastním serveru, sestavení vytvářet v nástroji Bitbucket a E2E testy provádět paralelně na AWS.

Výběr služeb na AWS je spjat s vyvíjenou aplikací. V případě této práce se jedná o aplikaci psanou v Angular frameworku. Výslednou podobou takové aplikace je kombinace statických souborů *html, javascript a css*, které se při prvním načtení stáhnou do prohlížeče a není nutná další přímá komunikace se serverem. Nejčastějším spojením aplikace s dalšími daty je pomocí REST návrhového stylu, který poprvé pojmenoval Roy Thomas Fielding ve své disertační práci (2000). Ten popisuje stav objektu pro potřeby přenosu dat po síti. Implementace tohoto stylu se může lišit, ovšem princip je zachován. Aplikace se tak

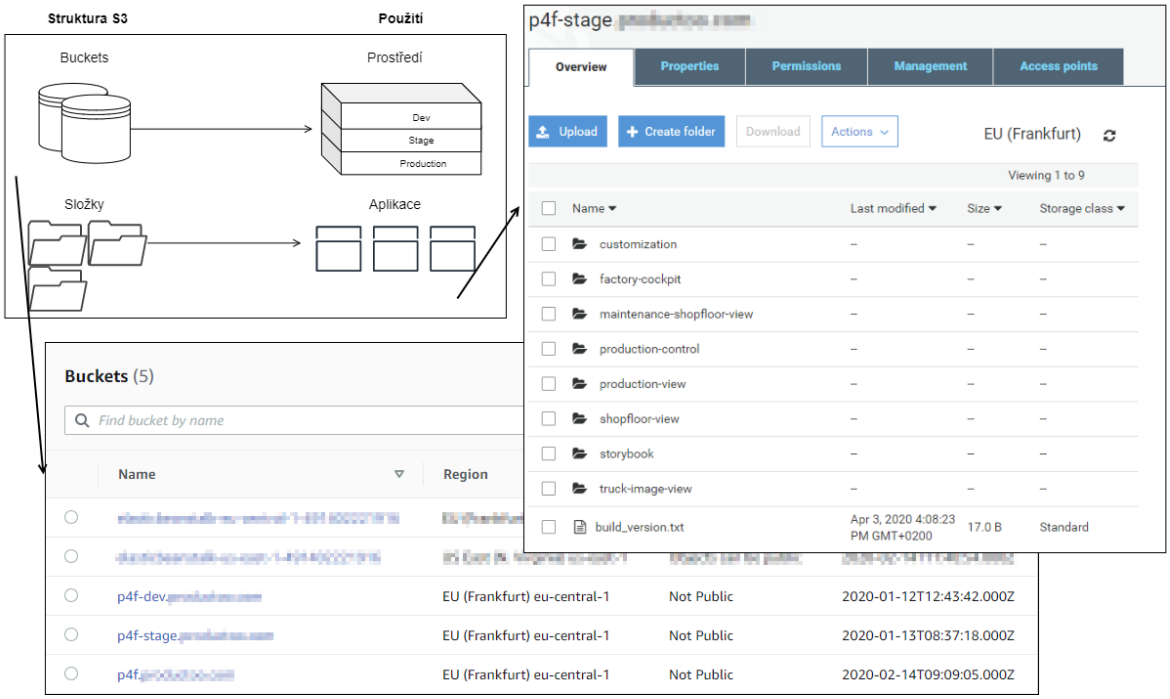
odděluje od systému, který zpracovává a vyhodnocuje data na část, která data pouze vizualizuje a případně pošle zpět na server ke zpracování. V rámci této práce se neuvažuje vytvoření funkčního serveru, který by tyto data zpracovával.

4.1 Uložiště Amazon S3

První klíčovou službou pro zprovoznění Continuous Deployment na AWS je Amazon S3 (Amazon Simple Storage Service). Ta nabízí velkokapacitní objektové uložiště, které může sloužit od ukládání statického obsahu pro webové stránky, přes zálohování až po big data analytiku (Amazon Web Services 2020e).

Bucket kontejnery

Bucket kontejner slouží pro vlastní členění v rámci Amazon S3. Umožňuje shlukovat soubory na nejvyšší úrovni a nastavovat různá pravidla a oprávnění, která již nelze měnit na úrovni jednotlivých složek a souborů uvnitř daného kontejneru (Amazon Web Services 2020d). Z toho důvodu se nejvíce hodí na nastavení jednotlivých prostředí (test/stage/production). Jednotlivé kontejnery lze dále členit pomocí složek, které umožňují snadnější nastavení později pro službu CloudFront. Pro úplnost je příklad tohoto nastavení možné vidět na obrázku 10, kde na levé straně je schématem naznačena možná struktura kontejnerů a ve spodní a pravé části jsou konkrétní ukázky tohoto nastavení.



Obrázek 10: Struktura Amazon S3. Zdroj: autor

Oprávnění

Každý bucket kontejner umožňuje nastavení oprávnění nad jednotlivými objekty. Přístup k datům lze omezit na několika úrovních (Amazon Web Services 2020e):

- Blokace veřejného přístupu.
- Přístup k datům na základě AWS uživatelských účtů.
- Přístup na základě pravidel pro ostatní služby AWS.

Výše uvedené tak umožňuje nastavit přístup k datům podle dané potřeby aplikace, aniž by bylo nutné kompromitovat samotnou bezpečnost dat a potenciálně se vystavit riziku při zcela veřejném přístupu k objektům.

Poslední nezmíněnou úrovní nastavení přístupu je ve službě CloudFront, která se chová jako CDN. Více o této službě v následující podkapitole.

4.2 Amazon CloudFront

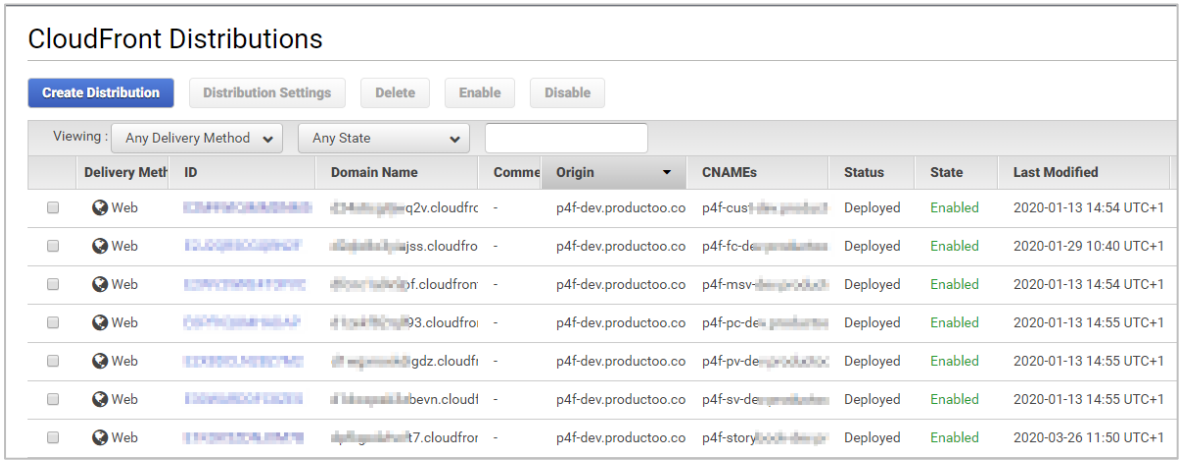
Druhou službou využívanou v rámci navrhované metodiky je CloudFront. Jedná se o síť pro doručování obsahu (*angl. Content Delivery Network, zkráceně CDN*), která dokáže přistupovat k datům uloženým v Amazon S3 a distribuuje je na různá místa po celém světě, aby poskytla co nejmenší odezvu koncovým uživatelům (Amazon Web Services 2020g). Jinými slovy se jedná o službu, která je bránou ke zdrojovým souborům výsledné aplikace.

V kontextu této práce se používá pro zobrazování obsahu z konkrétní složky uložené v rámci jednoho Bucket kontejneru. Díky tomu nemusí být kontejner veřejně přístupný. O přístup k datům se stará služba CloudFront, která umožňuje kromě jiného nastavit i vlastní doménové jméno, přesměrování, webové protokoly nebo certifikáty pro bezpečné spojení se serverem přes HTTPS.

Pro každou aplikaci a prostředí je nutné mít vytvořenou tzv. distribuci. Ta umožňuje nastavit pravidla pro konkrétní přístup k aplikaci a díky tomu se může lišit podle potřeby (například na testovacím umožnit nezabezpečený přístup přes HTTP nevyžadující platný certifikát, ale na produkčním prostředí vyžadovat připojení přes HTTPS).

Na úvodní stránce služby lze přehledně vidět všechny vytvořené distribuce, jak je uvedeno na obrázku 11:

- ID – unikátní identifikátor sloužící pro potřebné rozlišení distribuce např. při aktualizaci z CI nástroje.
- Domain Name – Přidělená doména, na které lze distribuci zobrazit.
- Origin – Zdroj dat, odkud distribuce získává data, která se následně zobrazují (v případě této práce se jedná o Amazon S3).
- CNAMEs – Alternativní domény, které lze použít místo výchozí Domain Name.
- Status – Aktuální stav, ve kterém se distribuce nachází.
- State – Informace o tom, zdali je distribuce aktivní / neaktivní.



Obrázek 11: Ukázka distribucí v CloudFront. Zdroj: autor

V rámci distribucí lze sledovat například objem přenesených dat, počet přístupů nebo HTTP návratové kódy a zároveň lze nastavit nejrůznější upozornění v případě překročení definovaných indikátorů. Zde je již zcela na správci, jaká data vyžadují sledování a jak reagovat na dané události. Každá distribuce si zdrojová data z Amazon S3 kopíruje na daný server, odkud je následně poskytuje koncovým uživatelům v dané lokalitě. Z toho důvodu je umožněno vynucení obnovy zdrojových souborů (*angl. invalidations),* což má za následek znovu nahrání těchto souborů z S3 Bucket kontejnerů. (Amazon Web Services 2020g)

Podrobné vytváření distribucí, práce s obnovením zdrojových souborů a nastavení pravidel je popsané v následující kapitole 5, konkrétně v podkapitole 5.5.2 a 5.5.3.

4.3 Shrnutí

V rámci této kapitoly byly představeny služby Amazon S3 a CloudFront, které jsou dostupné v rámci platformy Amazon Web Services a společně tvoří klíčové prvky poslední fáze CI/CD procesu. Tato kapitola naplňuje třetí dílčí cíl s názvem *představit služby na Amazon Web Services podporující Continuous Deployment proces*. Kromě toho byly krátce popsány výhody celé platformy AWS, která spadá do oblasti tzv. Cloud computingu.

Služby byly popsány především z hlediska funkcí, které jsou využívány v rámci metodiky s přihlédnutím na jistá specifika potřebná pro reálné nasazení aplikace do tří skutečných prostředí využívaných v provozu. Konkrétně se jednalo o popsání bucket kontejnerů a souborové struktury S3, její použití a s tím spojené nastavení. Následně propojení mezi S3 a CloudFront, u kterého bylo popsáno, jak lze distribuce využít při tvorbě přístupu ke zdrojovým souborům aplikace.

5 Metodika využití Bitbucket a AWS pro zavedení Continuous Integration, Delivery a Deployment procesu

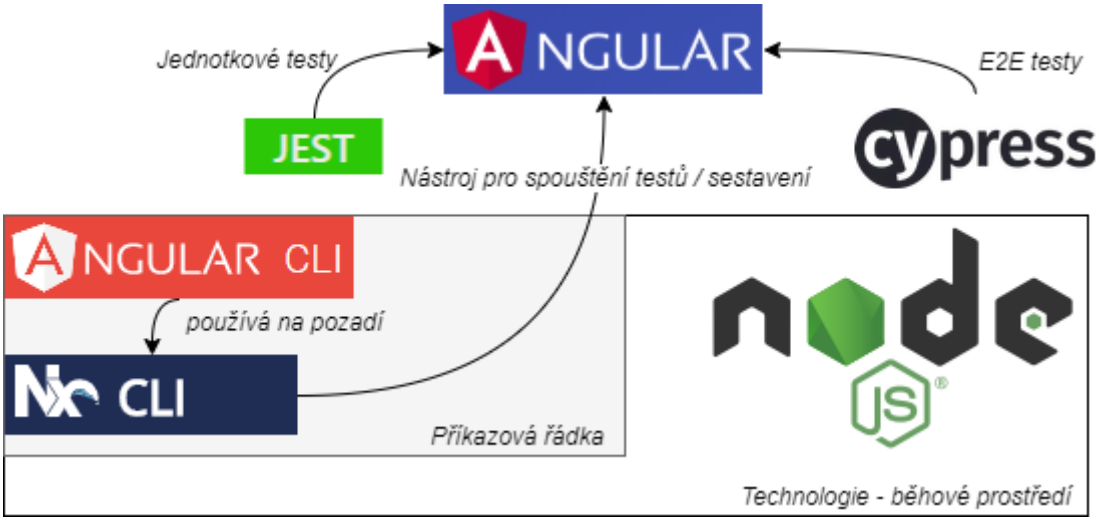
Tato kapitola je věnovaná návrhu metodiky pro zavedení Continuous Integration, Delivery a Deployment procesu. Přestože jsou dílčí kroky psané v Angular frameworku za pomocí nástrojů Angular CLI a NX CLI, obecné koncepty lze aplikovat i na jiné aplikace psané v různých technologiích s přihlédnutím na specifické požadavky. Přehled vytvářené metodiky lze vidět v tabulce 2. Dílčí činnosti, role a artefakty jsou popsány blíže v následujících podkapitolách.

Tabulka 2: Přehled činností metodiky, rolí a artefaktů. Zdroj: autor

Název činnosti	**Role**	**Artefakty**
Plánování testování a tvorba sestavení	Test manažer, (vývojář)	Plán testů, plán procesu sestavení aplikace
Prvotní nastavení nástroje a služeb	DevOps architekt	Dokumentace nastavení nástroje a služeb
Spouštění testů a sestavení v Bitbucket nástroji	DevOps architekt, (vývojář)	Sestavení výsledných souborů aplikace
Spouštění Deployment procesu	DevOps architekt, (Operativa, vývojář)	
Reporting běhu CI/CD	Operativa	Report běhu CI/CD procesu

5.1 Popis nástrojů a prostředí potřebných pro psaní aplikací

Nejdříve je potřeba přiblížit použitý Angular framework společně s dalšími technologiemi, které jsou přímo nebo nepřímo použité v rámci celého CI/CD procesu. Technologický základ aplikace je přehledně zobrazen pomocí diagramu na obrázku 12:



Obrázek 12: Technologický základ aplikace. Zdroj: autor

Technologie zmíněné v diagramu tvoří základní stavební kameny pro vytvoření aplikace v Angular frameworku. Jedná se především o NodeJS, Cypress a Jest, které jsou níže blíže popsány:

- *NodeJS* je open-source běhové prostředí psané v jazyce JavaScript sloužící ke spouštění programů psaných v JavaScriptu mimo webový prohlížeč (Node.js 2020). Mimo jiné slouží i jako platforma, která poskytuje nástroje pro kompilování z TypeScript jazyka (typová nadstavba nad JavaScript jazykem (Microsoft 2020)) a výsledného sestavení aplikace pro možné nasazení do prostředí.
- *Cypress* je framework sloužící k provádění E2E testování. Jedná se o nástroj, který je od začátku navrhnutý čistě v JavaScript jazyce. To umožňuje spouštění testů přímo v prohlížeči bez nutnosti dalších programů nebo knihoven. Díky tomu je možné přistupovat k veškerým částem aplikace a lze tak vytvářet různé zkratky, simulovat krajní situace, vracet testovací data, vyhodnocovat chování aplikace pod zátěží a mnoho dalšího (Cypress 2020).
- *Jest* je posledním zmíněným frameworkem, který je důležitý pro tuto metodiku z pohledu podpůrných knihoven. Jedná se o framework, který usnadňuje psaní jednotkových testů a má přímou podporu pro Angular aplikace (Jest 2020). Díky tomu je možné jej snadno nastavit a používat při vývoji a při kontrole v CI nástroji.

Dále se při vývoji aplikace psané v Angular frameworku využívá především nástroj Angular CLI, který je určen pro příkazovou řádku a má za úkol vytvářet prostředí pro vývoj a výsledné sestavení aplikace. Nicméně existují nadstavby nad tímto nástrojem, které dokáží mnohem více. Jedním z nich je nástroj s názvem NX CLI od firmy Narwhal Technologies Inc. (2020). Ten slouží pro větší abstrakci od konkrétního frameworku (jako

je například zmíněný Angular framework) a zároveň umožňuje lepší fungování v případě složitějších aplikací, které mezi sebou potřebují sdílet části kódu. Podrobněji je tento nástroj popsán v kapitole 6.1. Tento nástroj je zde zmíněn především kvůli tomu, že se používá při spouštění příkazů pro testování a vytváření výsledného sestavení aplikace.

Použití těchto nástrojů a technologií je nezbytné pro tuto práci, nicméně to nevylučuje budoucí vlastní přizpůsobení metodiky na základně dostupných nástrojů, zdrojů nebo technologií.

5.2 Role v rámci metodiky

V rámci navrhované metodiky se vyskytují role, které jasně definují zodpovědnosti a kroky nutné pro úspěšné zavedení. Snaha není vytvořit nové role, nýbrž použít takové, aby bylo možné stávající povinnosti jednotlivých členů týmu pouze rozšířit o dané činnosti a tím příliš nenarušovaly zaběhnutá pravidla v dané organizaci.

5.2.1 DevOps architekt

Role, která má za cíl celý proces monitorovat, plánovat jeho dílčí kroky, řešit problémy, vymýšlet nové postupy, určovat strategii a hledat úzká místa, které by bylo vhodné zlepšovat. Očekává se alespoň elementární znalost procesu Continuous Intergration, Delivery a Deployment bez nutnosti hluboké technické znalosti. Není nutné tudíž umět programovat v použitém programovém jazyce na projektu.

5.2.2 Vývojář

Role, která je nezbytná pro připravení technických podkladů potřebných pro úspěšné zavedení této metodiky. Nemusí být nutně vývojář z daného projektu, ale musí mít hlubší znalost programovacího jazyka, použitých technologií a metodiky vývoje na projektu.

5.2.3 Test manažer

Role zodpovědná za určování strategie a plánu testování. Navrhování postupů, určování míst, které je nutné testovat na základě byznys požadavků. V případě absence lze nahradit vývojářem, popřípadě zkušenějším testerem. Role je zodpovědná za dílčí část celého procesu, konkrétně za Continuous Integration.

5.2.4 Operativa

Lidé z operativy mají zodpovědnost za monitorování aplikace v produkčním prostředí a hlásit případné problémy zpět vývojářům. Měli by znát manuální postup nasazení aplikace do zmíněného prostředí. Hluboká znalost tohoto procesu je tudíž nezbytná pro správné monitorování a reportování.

5.3 Artefakty metodiky

Artefakty jsou výstupy, které mají zůstat přístupné po dokončení jednotlivých kroků. Jedná se především o řídící dokumenty a výstupy sestavení, testování a pravidelné reporty z běhu aplikace.

5.3.1 Plán testů

Prvním artefaktem z celé metodiky je plán testů. Ten určuje, jakou část aplikace je potřeba důkladně testovat, jakou kombinaci testů zvolit a kdy které testy provádět. Je to výstup z činnosti *plánování testování a tvorba sestavení* a zároveň vstupem pro činnost *spouštění testů a sestavení v Bitbucket nástroji*.

5.3.2 Plán procesu sestavení aplikace

Dalším artefaktem je plán procesu sestavení aplikace. Ten popisuje jednotlivé příkazy a kroky, které je nutné spouštět v předem dané posloupnosti. Tento artefakt je výstupem činnosti *plánování sestavení a tvorba sestavení* a je vstupem pro činnost *spouštění testů a sestavení v Bitbucket nástroji*.

5.3.3 Dokumentace nastavení nástroje a služeb

Třetím artefaktem metodiky je dokumentace, která zahrnuje prvotní nastavení nástroje Bitbucket a služeb Amazon S3 a CloudFront. Jedná se o výstup činnosti *prvotní nastavení nástroje a služeb*.

5.3.4 Sestavení výsledných souborů aplikace

Dalším artefaktem je sestavení výsledných souborů aplikace. Ten je předáván v jednotlivých krocích především kvůli tomu, aby bylo možné provést nasazení do předem určeného prostředí. Díky tomuto sestavení vznikne jedna konkrétní verze, která se následně používá ve všech prostředích. Jedná se o výstup z činnosti *spouštění testů a sestavení v Bitbucket nástroji* a je vstupem do činnosti s*pouštění Deployment procesu*.

5.3.5 Report běhu CI/CD procesu

Posledním artefaktem je report běhu CI/CD procesu. Ten slouží k tomu, aby bylo možné zpětně dohledat souhrnný report běhu konkrétní úlohy. Tento report je výstupem činnosti *reporting běhu CI/CD*.

5.4 Činnost: Plánování testování a tvorba sestavení

Počáteční činností v rámci metodiky je příprava plánování testů a sestavení pro danou aplikaci. Tento krok je důležitý hlavně proto, aby bylo možné pravidelně v rámci Continuous Integration ověřovat funkčnost aplikace, aniž by musel tester každé sestavení ručně zkoušet a podávat informaci o výsledku. Jedná se o rychlou zpětnou vazbu pro vývojáře, že jeho změny jsou funkční a v rámci celé aplikace nezpůsobily jiné chyby, které se mohou projevit až po nasazení aplikace. Zároveň se kontroluje, že ze zdrojového kódu lze provést produkční sestavení, které je možné později nasadit do předem určeného prostředí.

V rámci plánu testování je potřeba si stanovit, jaké testy a v jaké míře budou použity na projektu. Z hlediska aplikace psané v Angular frameworku dávají smysl především jednotkové a E2E testy, se kterými se v této aplikaci počítá a budou prováděny v rámci CI/CD nástroje. Zároveň je nutné si stanovit proces vytváření sestavení a provést konkrétní nastavení za pomocí nástrojů Angular CLI a NX CLI.

5.4.1 Jednotkové testy

Jednotkové testy v aplikaci jsou spouštěny za pomoci nástroje Jest, který byl blíže popsán v podkapitole 5.1. Příklad takového testu je možné vidět ve výpisu 1:

Výpis 1: Ukázka jednotkového testu. Zdroj: autor

```
describe('BaseApiEntityService', () => {
  let service: TestService;
  const http = {
    get: jest.fn(() => of(null))
  };
  const baseUrl = '/entities';

  beforeEach(() => {
    service = new TestService(http as any);
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });

  it('should remove unwanted data from url (trailing / or empty outputType or default
outputType)', () => {
    expect(service.url(`${baseUrl}/`, 'any', {})).toBe(baseUrl);
    expect(service.url(`${baseUrl}`, 'any', { outputType: 'default' })).toBe(baseUrl);
    expect(service.url(`${baseUrl}`, 'any', { outputType: '' })).toBe(baseUrl);
  });

  it('should create single entity urls', () => {
```

```
    const url = `${baseUrl}/1`;

    // Parameters with no effect
    expect(service.url(url, 'single', {})).toBe(url);
    expect(
      service.url(url, 'single', {
        page: 1,
        pageSize: 10,
        sort: 'id',
        sortDir: 'asc',
        filter: 'id~eq~1',
        properties: ['id']
      })
    ).toBe(url);

    // Check output types
    ['basic', 'extended', 'any'].map(value => {
      expect(service.url(url, 'single', { outputType: value
})).toBe(`${url}?outputType=${value}`);
    });
  });

  it('should create listing urls', () => {
    expect(service.url(baseUrl, 'list', {})).toBe(`${baseUrl}?page=1&pageSize=10`);
    ['basic', 'extended', 'any'].map(value => {
      expect(service.url(baseUrl, 'single', { outputType: value
})).toBe(`${baseUrl}?outputType=${value}`);
    });
  });

  it('should create available property filter urls', () => {
    expect(service.url(baseUrl, 'filter', {})).toBe(`${baseUrl}/filters`);
    expect(service.url(baseUrl, 'filter', { filter: 'id~eq~1', properties: ['id', 'code']
})).toBe(
      `${baseUrl}/filters?properties=id|code&filters=id~eq~1`
    );
  });
});
```

Ve výše uvedeném výpisu 1 je ukázán zápis testu, který kontroluje správnost formátu URL adresy na základě vstupních parametrů. Výsledek běhu tohoto testu je možné vidět na obrázku 13.

```
> Executing task in folder monorepo: ng test shared-api-data-access <

 PASS  libs/shared/api/data-access/src/lib/base-api-entity.service.spec.ts
  BaseApiEntityService
    √ should be created (3ms)
    √ should remove unwanted data from url (trailing / or empty outputType or default outputType) (2ms)
    √ should create single entity urls (1ms)
    √ should create listing urls (2ms)
    √ should create available property filter urls (1ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        6.303s
Ran all test suites.
```

Obrázek 13: Výsledek běhu testu. Zdroj: autor

5.4.2 E2E testy

V rámci E2E testů lze provádět ověření funkcionality buď zvoleným průchodem aplikace, nebo testováním stavů životního cyklu entity od začátku do konce. Blíže byly E2E testy popsány v kapitole 2.3.1. V rámci metodiky se spouštějí testy psané ve Cypress frameworku. Příklad takového testu může být načtení aplikace a pokus o přihlášení. Test je tak dobrým příkladem pro smoke testování, které umožňuje rychle ověřit funkčnost aplikace, aniž by bylo potřebné hloubkové testování, popřípadě otestovat kritickou funkcionalitu, která je nezbytná pro chod aplikace. Příkladem takového testu může být právě zmíněné testování přihlašování. Tento test lze vidět ve výpisu 2.

Výpis 2: Ukázka E2E testu. Zdroj: autor

```
describe('Logging to the app', function () {
  it('Should successfully loggin user', function() {
     cy.visit('localhost:4204');

    cy.get('#mat-input-4').type(`username`);
    cy.get('#mat-input-5').type(`password`);
    cy.get('.mat-flat-button').click();
    cy.url().should('contain', 'production-lines');
  })
});
```

Předmětem této metodiky není vytváření testů, nýbrž nastavení kroků tak, aby se testy spouštěly automaticky a v případech, které jsou stanoveny v rámci vývojového procesu. Jejich výsledek následně ovlivňuje navázané kroky a v případě jejich nefunkčnosti znemožní nasazení aplikace do produkčního prostředí.

5.4.3 Plánování sestavení

V rámci této činnosti je potřeba rozvrhnout v jaké konfiguraci a v jakých situacích bude prováděno výsledné sestavení aplikace, které bude vhodné pro nasazení do produkčního prostředí. V ukázkách skriptů se uvažuje, že při nasazování do libovolného prostředí se používá vždy výhradně produkční sestavení aplikace, které je oproštěno o část kódu sloužící pro jednodušší ladění kódu. Provádění více verzí sestavení v CI/CD nástroji je možné, nicméně každé další sestavení prodlužuje celkový čas potřebný k nasazení do produkčního prostředí a přímo se podílí na delší prodlevě mezi commitem a reportem vytvořeným po nasazení do prostředí. Podle (Kim et al. 2016) by měla být snaha tento čas (*angl. lead time*) co nejvíce zkracovat, protože je kritický pro optimální pracovní nasazení jedince, popřípadě celého týmu pracujícího na projektu. Bez finálního potvrzení z CI/CD nástroje totiž vývojář nemůže danou změnu zcela uzavřít a věnovat se tak nerušeně dalšímu úkolu v pořadí. Podrobně bylo vysvětleno v podkapitole 2.1.

K vytvoření sestavení slouží nástroj Angular CLI. Ten umožňuje spustit sadu příkazů, které ze zdrojového kódu psaného v TypeScriptu vytvoří HTML, JavaScript a CSS soubory potřebné pro spuštění aplikace. To vše bez pomoci NodeJS běhového prostředí, protože s těmito soubory si prohlížeč bez problému poradí. Příklad tohoto spuštění je následovný:

```
ng build customization --prod
```

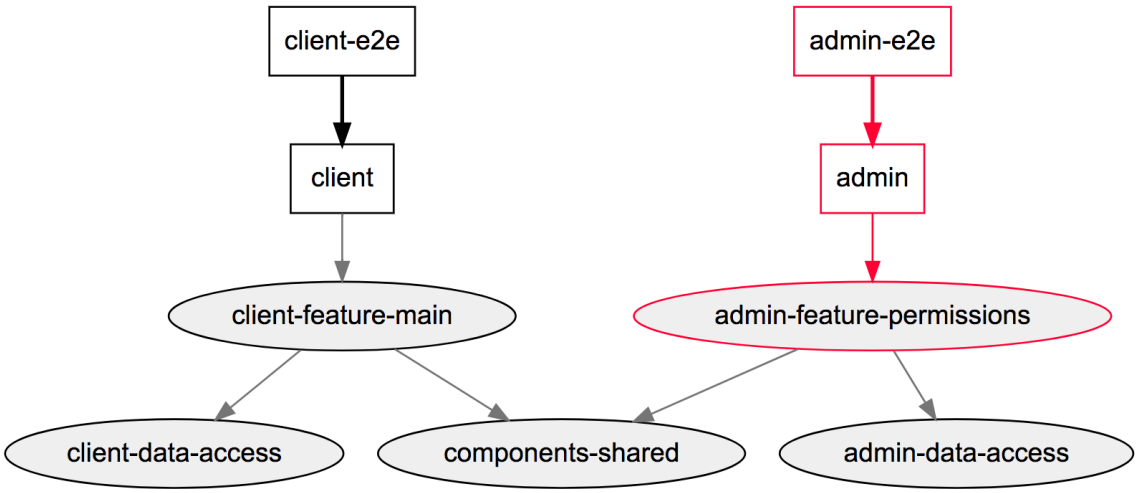
Kde *customization* je jedna ze sedmi aplikací (blíže popsáno v rámci podkapitoly 6.1) a parametr *prod* určuje produkční sestavení, které neobsahuje části kódu potřebné pro pozdější ladění chyb. Výsledné soubory lze vidět na obrázku 14.

[..]		32-es2015.1cee642ed2216d214550	js	Saira-BoldItalic.d39ccfa540fdbbdebed4	eot
[assets]		32-es5.1cee642ed2216d214550	js	Saira-BoldItalic.fa47a06c934686420e6c	woff
0-es2015.a58b6eb5df04c7aff8d1	js	33-es2015.09ede2fc2704a4eff182	js	Saira-BoldItalic.fd055a165b8bf636902b	ttf
0-es5.a58b6eb5df04c7aff8d1	js	33-es5.09ede2fc2704a4eff182	js	Saira-Italic.20c7427c664db6c694e1	eot
13-es2015.c1cf048b2576564d19e7	js	3rdpartylicenses	txt	Saira-Italic.2aa05096ce8d6ed10321	woff
13-es5.c1cf048b2576564d19e7	js	4-es2015.2f5ecdcc7d0d9366a40d	js	Saira-Italic.8ef6e6c0df6c68d4f458	ttf
14-es2015.8ebfeb2df6bbb2b0d03f	js	4-es5.2f5ecdcc7d0d9366a40d	js	Saira-Italic.9ffeafb882147cc832e6	woff2
14-es5.8ebfeb2df6bbb2b0d03f	js	5-es2015.609479a62fa9a575568d	js	Saira-Regular.0bf8c4b13ec43976a546	woff2
15-es2015.b8fd63007621e74b99a2	js	5-es5.609479a62fa9a575568d	js	Saira-Regular.744e7d6c8e1cd0a17eb7	eot
15-es5.b8fd63007621e74b99a2	js	6-es2015.72aa8df5aa2bb8e07cd3	js	Saira-Regular.90c319fc35631fa2640c	woff
16-es2015.6b05768ee3504c4cb78f	js	6-es5.72aa8df5aa2bb8e07cd3	js	Saira-Regular.e6c0ef63942246f17037	ttf
16-es5.6b05768ee3504c4cb78f	js	8-es2015.24fea5327f4444809c31	js	Saira-Thin.32633792e67a056bd1b7	woff
17-es2015.ea09dbe708e08d94ff8b	js	8-es5.24fea5327f4444809c31	js	Saira-Thin.605fa4e83d5a0c4729b9	woff2
17-es5.ea09dbe708e08d94ff8b	js	common-es2015.60ba2caf5aa45491db38	js	Saira-Thin.6258129337cc606267df	ttf
18-es2015.e83f9cc16f3329139b83	js	common-es5.60ba2caf5aa45491db38	js	Saira-Thin.8c220ed5fecff50d6475	eot
18-es5.e83f9cc16f3329139b83	js	default-skin.0ff4d62150facee6b7de	png	Saira-ThinItalic.1a334a4ef64d6b0df5e5	eot
19-es2015.f67d8cd196b7f42b7560	js	default-skin.f64c3af3d0d25b9e4e00	svg	Saira-ThinItalic.4dcddfb99c22e73408ac	woff
19-es5.f67d8cd196b7f42b7560	js	fa-brands-400.38168200fbb96276baf9	woff2	Saira-ThinItalic.c519031633446dd2326a	woff2
1-es2015.42c6605d2c59d4d095bf	js	fa-brands-400.39c1e33ca5d67754117c	eot	Saira-ThinItalic.e7481fdbde1d547cdf27	ttf
1-es5.42c6605d2c59d4d095bf	js	fa-brands-400.4d5ec63961cae00e3536	ttf	scripts.adc07013f3f76c4c7109	js
20-es2015.842c7f8f71bda6a30f33	js	fa-brands-400.6c7693a31ea2d2dc6fe9	svg	styles.7c1a6795bc8c9b18e7ad	css
20-es5.842c7f8f71bda6a30f33	js	fa-brands-400.a7cb8a53a882fd128ae0	woff		
21-es2015.1cc941d19fdfc6ab2dd2	js	fa-regular-400.a28b95d3e799579ba1dd	woff2		
21-es5.1cc941d19fdfc6ab2dd2	js	fa-regular-400.a403036f8820a4a4273d	ttf		
22-es2015.252de001a36d746fa0ff	js	fa-regular-400.b3759c28ef0607075578	woff		
22-es5.252de001a36d746fa0ff	js	fa-regular-400.d099731ecd1c83c2be63	eot		
23-es2015.ba17105fe0ee006476e5	js	fa-regular-400.d18055720cc5bab12c36	svg		
23-es5.ba17105fe0ee006476e5	js	fa-solid-900.0ec0cc5685d258116325	svg		
24-es2015.0e25d2d84f794874d88c	js	fa-solid-900.23a7efd8d0ef4f232f2b	woff2		
24-es5.0e25d2d84f794874d88c	js	fa-solid-900.6ccf0adaba47375248a9	woff		
25-es2015.ad445da45ae4387ce1ec	js	fa-solid-900.c4e5551b39a7d5c61027	ttf		
25-es5.ad445da45ae4387ce1ec	js	fa-solid-900.d12229bd491638fa106a	eot		
26-es2015.ba5246ba4da4d252b562	js	favicon	ico		
26-es5.ba5246ba4da4d252b562	js	index	html		
27-es2015.497fa0a96fc0e5ee83b4	js	main-es2015.a7416931a90bce2491d9	js		
27-es5.497fa0a96fc0e5ee83b4	js	main-es5.a7416931a90bce2491d9	js		
28-es2015.b0a47a3680066db5f4aa	js	polyfills-es2015.a792b6b9241ead72b5f7	js		
28-es5.b0a47a3680066db5f4aa	js	polyfills-es5.5ca5176fb247f3e1b7f5	js		
29-es2015.64a2de3a0a1f5364c5ea	js	preloader.14d740b6ee64510747b0	gif		
29-es5.64a2de3a0a1f5364c5ea	js	runtime-es2015.ccb9b8e6f610b65f1496	js		
2-es2015.e2e7e2731c4a4d70ac81	js	runtime-es5.ccb9b8e6f610b65f1496	js		
2-es5.e2e7e2731c4a4d70ac81	js	Saira-Bold.387315f63a8bb600ed80	woff2		
30-es2015.56cba0f31edf15088754	js	Saira-Bold.75946e1c11877dc894b3	woff		
30-es5.56cba0f31edf15088754	js	Saira-Bold.ad4bd0b45a042cd25294	eot		
31-es2015.a6140b3fdf0b4bf00a09	js	Saira-Bold.c5f20a4e399bc2896b7f	ttf		
31-es5.a6140b3fdf0b4bf00a09	js	Saira-BoldItalic.0d03ea70e733871c69cd	woff2		

Obrázek 14: Výsledné soubory po provedení sestavení aplikace. Zdroj: autor

Artefaktem se v tomto okamžiku stává konkrétní sestavení dané aplikace. Nicméně v reálném provozu je velmi neefektivní provádět sestavení všech aplikací na projektu, které jsou vyvíjené v rámci jednoho repozitáře, protože při jakékoliv menší změně kódu by bylo nutné toto sestavení provádět pro každou aplikaci. Takové fungování je tím pádem časově náročné, protože vytvoření sestavení jedné aplikace trvá v průměru 2-3 minuty a při započtení sedmi aplikací se může tento čas pohybovat okolo 14-21 minut a potenciálně růst společně s aplikací.

V tento okamžik přichází na řadu nástroj pro příkazovou řádku s názvem NX CLI, který poskytuje pokročilou analýzu zdrojového kódu a dokáže tak rozpoznat, na jaké aplikace daná změna měla vliv. V rámci nástroje CI/CD je proto vhodnější spouštět sestavení jen takových aplikací, které byly změnou ovlivněny a tím výrazně zkrátit celkový čas potřebný pro sestavení aplikace.



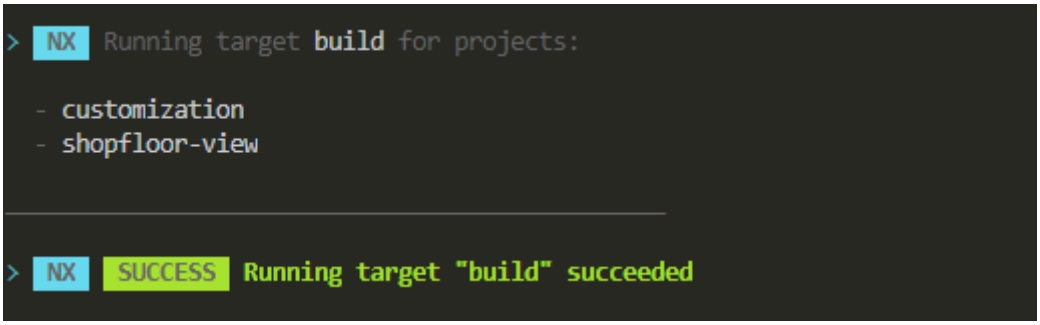
Obrázek 15: Ukázka analýzy zdrojového kódu a vlivu změny na konkrétní aplikace. Zdroj: (Narwhal Technologies Inc. 2020)

Na obrázku 15 je možné vidět příklad, jak změna ve zdrojovém kódu dílčí části *admin-feature-permissions* ovlivňuje pouze aplikaci *admin*, nikoliv aplikaci *client*. Na tomto konkrétním příkladu tak v nástroji CI/CD dojde ke snížení času až o polovinu oproti klasickému sestavení, při kterém by bylo nutné vždy spouštět příkaz pro všechny aplikace nehledě na provedené změny ve zdrojovém kódu.

Při spuštění upraveného příkazu pro sestavení dojde nejdřív ke zmíněné analýze a následně je rozhodnuto, které aplikace byly danou změnou ovlivněny pomocí příkazu níže:

```
nx affected:build --prod
```

Výstupem tohoto příkazu je seznam ovlivněných aplikací, pro které jsou následně vytvořeny sestavení a tím jsou připraveny na nasazení do prostředí. Jeho výstup lze vidět na obrázku 16, na kterém lze vidět dvě aplikace, které tím pádem vyžadují vytvoření nového sestavení.



Obrázek 16: Výstup po spuštění sestavení ovlivněných aplikací. Zdroj: autor

5.5 Činnost: Prvotní nastavení nástroje a služeb

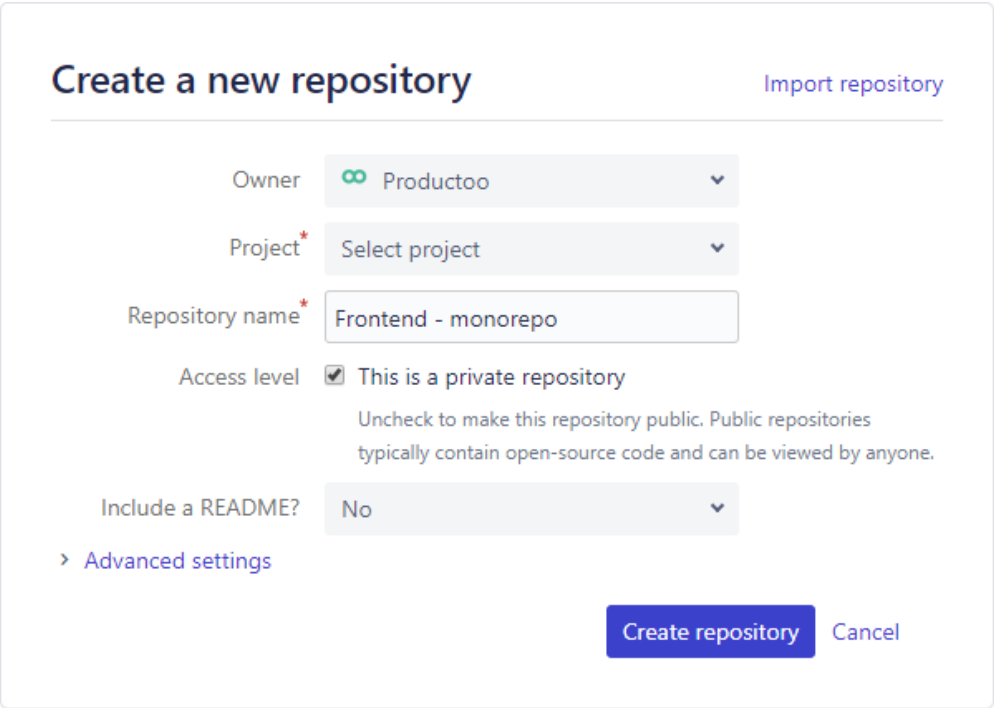
Jak bylo popsáno v podkapitole 2.4.1 věnované strategii nasazování, je nutné všechny služby nejprve správně nastavit a připravit prostředí pro pravidelné spouštění jednotlivých příkazů, ať už v rámci CI/CD nástroje nebo při nasazování do produkčního prostředí. Jednotlivé nástroje a služby byly popsány v kapitolách 3 a 4, nicméně konkrétní nastavení bude přiblíženo v rámci této činnosti věnované prvotní konfiguraci pro následné zavedení procesu CI/CD.

5.5.1 Příprava nástroje Bitbucket

V rámci nástroje Bitbucket je potřeba nastavit několik věcí uvnitř konkrétního repozitáře obsahujícího zdrojový kód aplikací. Vše lze pochopitelně aplikovat i při vývoji jedné konkrétní aplikace.

Vytvoření repozitáře

Vytvoření nového repozitáře je triviální záležitost, která zahrnuje pouze zadání názvu repozitáře, jeho zařazení do projektu (kategorizace), nastavení vlastníka, oprávnění přístupu a další volitelná nastavení. Obrazovku vytvoření repozitáře lze vidět na obrázku 17.

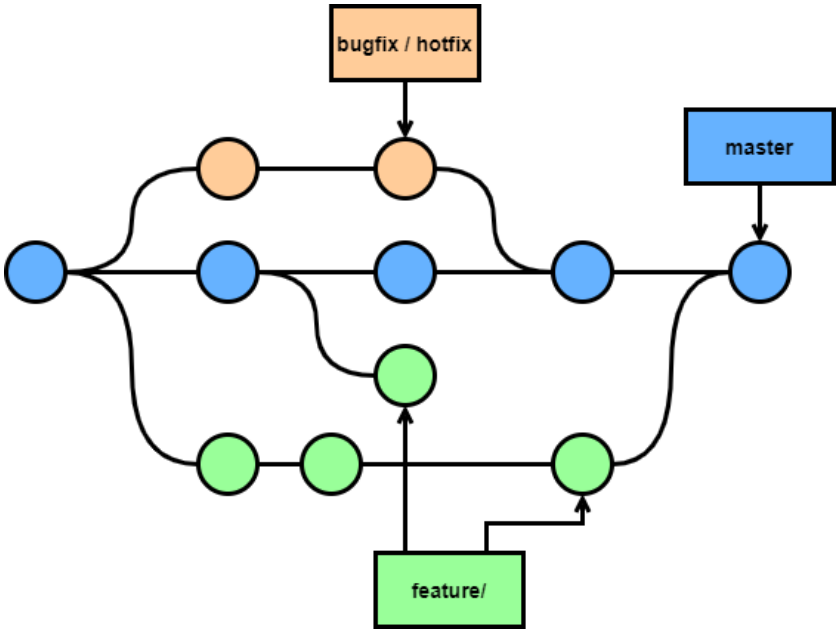


Obrázek 17: Vytvoření repozitáře. Zdroj: autor

Jakmile je repozitář vytvořen, automaticky se vytvoří i git repozitář umožňující spravování jednotlivých verzí aplikace na úrovni zdrojového kódu pomocí zmíněného nástroje git.

Nastavení modelu větvení

V rámci metodiky je nutné si ujasnit, jaká větev slouží k vývoji a jaká k umožnění nasazení do produkčního prostředí. V prostředí firmy implementující tuto metodiku se vývoj řídí *master* větví, která je zároveň automaticky nastavena na vytváření produkčních sestavení za podmínky, že jednotlivé kroky v procesu projdou bez chyby (testování a vytvoření sestavení). Vedle *master* větve se používají feature, bugfix a hotfix, na kterých se provádí menší úpravy a později se integrují zpět na *master* větev.



Obrázek 18: Model větvení. Zdroj: autor

Na obrázku 18 je možné vidět vývojový proces nových funkcionalit, kdy každá změna je prováděna na oddělené větvi buď typu *bugfix, hotfix* nebo *feature*. Rozdíl mezi nimi je především v kontrolním procesu po nasazení na produkční prostředí. Jednotlivé větve jsou vždy po dokončení úprav pomocí *pull request* požadavku integrovány zpět do hlavní vývojové větve *master*. Následně jsou nad touto větví provedeny poslední testy a případně vytvořeny produkční sestavení všech ovlivněných aplikací umožňující jejich pozdější nasazení do produkčního prostředí.

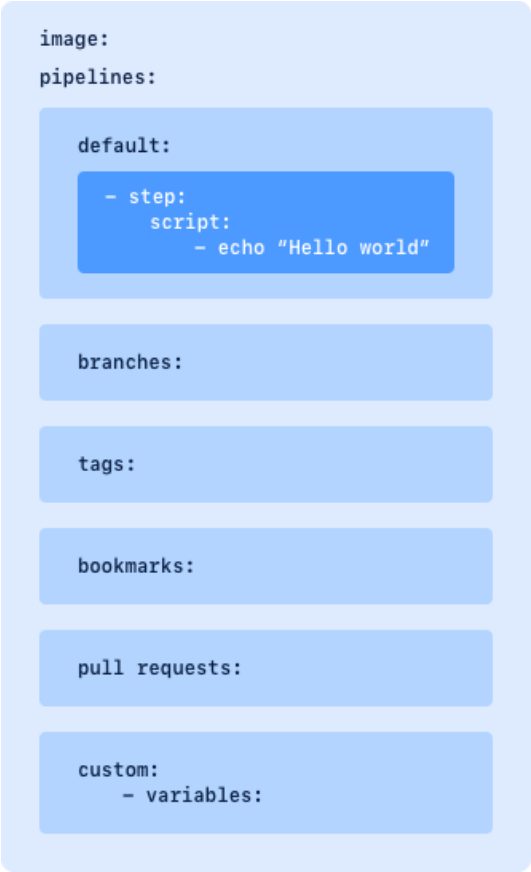
Bitbucket umožňuje toto nastavení především z toho důvodu, aby nástroj lépe dokázal provádět jednotlivé kroky integrace zdrojového kódu a pomocí prefixů větví umožnil spouštět odlišné úlohy pro různé větve. Příkladem může být provádění jednotkového testování s každou novou verzí na *feature* větvi, nicméně provedení sestavení a otestování pomocí E2E testů se může provádět až při *pull request* požadavku, popřípadě při nové verzi na větvi *master*. Provést E2E testy je časově náročnější, a proto není vhodné je spouštět s každou nově nahranou změnou v kódu na *feature* větvích, ale spouštět je až při pokusu o integraci na hlavní vývojovou větev.

Nastavení pipelines

Řízení činností probíhajících v rámci CI/CD nástroje jsou uloženy v konfiguračním souboru přímo ve zdrojovém kódu pod názvem *bitbucket-pipelines.yml*. Tudíž lze vždy získat aktuální konfigurační soubor vzhledem ke stavu aplikace. Je tím pádem možné současně sledovat změny prováděné v této konfiguraci a zjistit, co se případně oproti předchozí verzi změnilo. Toto nastavení lze vidět ve výpisu 3 níže, který obsahuje příklad jednotlivých kroků podle určitého spouštěče. Ten je rozdělen na tři základní sekce podle zvoleného vývojového procesu v dané firmě:

- Při novém *pull request* požadavku.
- Při nové změně na *feature* větvi.
- Při nové změně na *master* větvi.

Obecně se nastavení konfiguračního souboru řídí podle předem definovaných bloků, které jsou blíže ukázány na obrázku 19. Kromě výše uvedených lze používat i sekce *tags*, *bookmarks, default* a *custom*. Ty slouží k jiným stylům vývoje, kdy například vytváření produkčních sestavení se řídí tvorbou tagů, případně při definování zcela vlastní sekce *custom*, která má jedinou limitaci oproti ostatním, a to absenci automatického spouštění. (Atlassian 2020a)



Obrázek 19: Sekce v rámci konfigurace pipelines. Zdroj:(Atlassian 2020a)

V rámci každé sekce lze provádět kroky (*angl. steps*), které slouží pro spouštění konkrétních příkazů v rámci příkazové řádky. Každý krok obsahuje tři části. Jmenovitě se jedná o *caches,*

name a *script*. *Caches* umožňují místo opětovného stahování balíčků *node_modules* (obsahující knihovny Angular frameworku) využít cache v rámci Bitbucket, které se při opakovaném spuštění pouze stáhnou a nemusí se znovu instalovat. Druhá část s názvem *name* slouží k definování uživatelského pojmenování daného kroku. Poslední část *script* obsahuje jednotlivé příkazy, které se mají v rámci daného kroku provést. (Atlassian 2020a)

Všechny příkazy jsou spouštěny izolovaně v rámci Docker image (virtualizační prostředí), což umožňuje vytváření jednorázových prostředí, které jsou následně smazány po ukončení všech kroků. Společným ukazatelem všech sekcí na příkladu ve výpisu 3 je spouštění testů, které mají za cíl rychle odhalit základní chyby v aplikaci. Následně se spouští kontrola sestavení aplikace, která ověří funkčnost před jejím nasazením do produkčního prostředí. Nejobsáhlejší je sekce spouštějící se nad hlavní *master* větví umožňující v případě úspěšného projití všemi kroky vytvoření sestavení a jejich nasazení na jednotlivá prostředí (ve výpisu jsou nastavené tři – test/stage/production). Tento krok je realizován v CI/CD nástroji, nicméně samotný Deployment je prováděn ve službách AWS. Jelikož je tato kapitola věnovaná nastavení nástroje a služeb, nejsou podrobně rozepsány jednotlivé kroky, které jsou více vysvětleny v následujících dvou kapitolách věnovaným konkrétnímu spouštění sestavení a Deployment aplikací do prostředí.

Výpis 3: Příklad výpisu konfigurace pipelines pro Bitbucket (zkráceno). Zdroj: autor

```
image: node:12.13.0
pipelines:
  pull-requests:
    '**':
      - step:
          caches:
            - node
          script:
            ### ... zkráceno
  branches:
    '{feature/**,hotfix/**,bugfix/**}':
      - step:
          caches:
            - node
          name: Build and test
          script:
            ### ... zkráceno
    master:
      - step:
          caches:
            - node
          name: Build and test
          script:
            ### ... zkráceno
          artifacts:
            - dist/apps/**
```

```
    - step:
        name: Deploy to test
        deployment: Test
        script:
          ### ... zkráceno
    - step:
        name: Deploy to stage
        deployment: Staging
        script:
          ### ... zkráceno
    - step:
        name: Deploy to production
        deployment: Production
        script:
          ### ... zkráceno
```

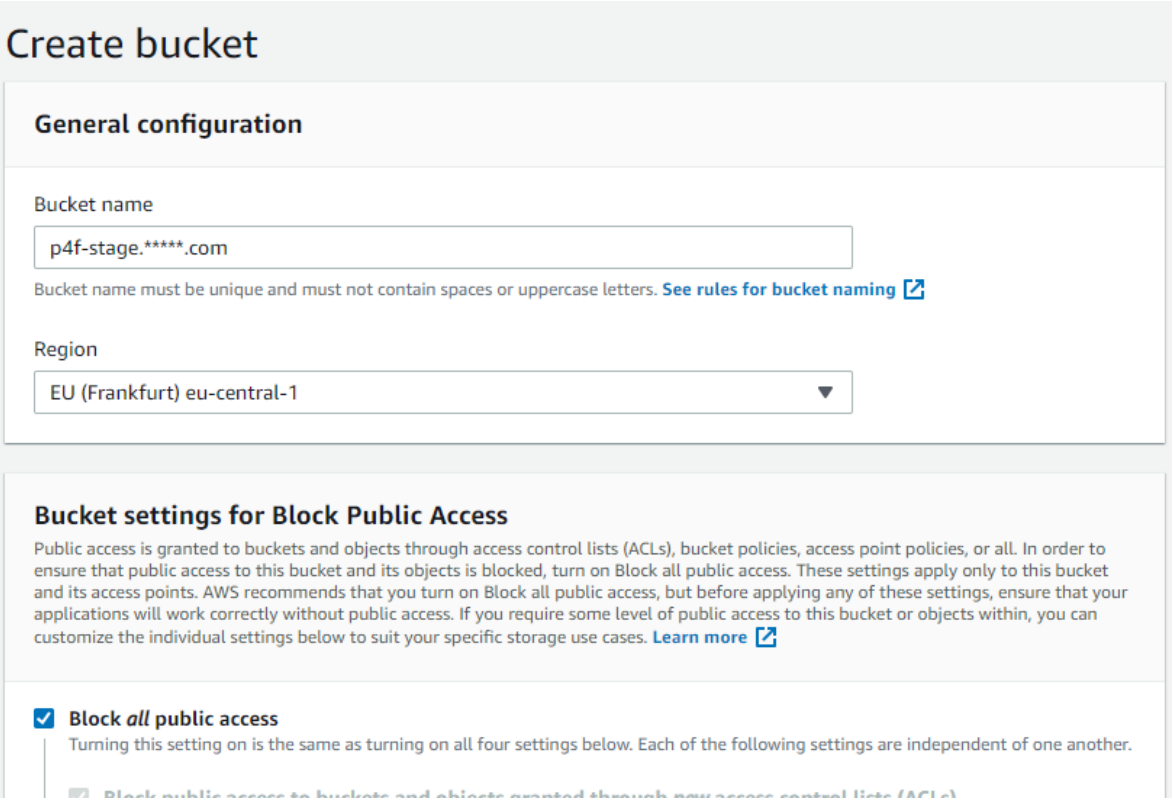
Tento konfigurační soubor, jak již bylo zmíněno dříve, je uložen ve nástroji git umožňující verzování zdrojového kódu, společně se zdrojovým kódem aplikace. Tudíž jakákoliv změna v konfiguraci je řádně zaznamenána a dohledatelná. Nestane se tak, že při nutnosti použití starší verze by nebylo možné spustit všechny kroky kvůli nekompatibilitě mezi zdrojovým kódem a konfigurací pipelines.

5.5.2 Příprava Amazon S3

Uvnitř Amazon S3 je nutné nejdříve připravit jednotlivé Bucket kontejnery. Jak bylo popsáno v kapitole 4.1, nejvhodnější způsob vytváření kontejnerů je podle typu prostředí. Tzn. každé prostředí by mělo mít svůj vlastní kontejner spojený s konkrétním prostředím (test/stage/production).

Vytvoření Bucket kontejneru v Amazon S3 uložišti

Založení nového Bucket kontejneru v Amazon S3 je otázka vyplnění názvu, regionu a nastavení veřejného přístupu (Amazon Web Services 2020d). Vše je vidět na obrázku 20. Region je důležitý z hlediska geografického uložení skutečných dat. Jinými slovy je dobré zvolit takový region, který je blízko vývojovému týmu, popřípadě se nachází na území takového státu, aby nebyl problém s legislativními opatřeními jako je například ochrana dat uživatelů na území Evropské unie. Poslední zmíněnou sekcí je veřejný přístup k datům. Amazon S3 je totiž možné využít i jako veřejné uložiště objektů a umožnit přístup komukoliv. To ovšem není vhodné v případě této metodiky, kdy jsou data zpřístupněna až po nastavení druhé služby CloudFront, která umožňuje další nastavení přístupu a práv k jednotlivým objektům.



Obrázek 20: Vytvoření Bucket kontejneru v Amazon S3. Zdroj: autor

V rámci konkrétního Bucket kontejneru je dobré vytvořit jednotlivé složky pro konkrétní aplikace. Do těch je následně nasměrována každá distribuce CloudFront a tím pádem je možné ke každé aplikaci přistupovat z vlastní URL adresy. Do takto vytvořené struktury aplikací lze buď ručně nahrát první verzi aplikace, nebo vše již nechat na automatický proces Deployment v rámci dalších kroků.

5.5.3 Příprava CloudFront

Poslední službou, kterou je nutné nastavit, je CloudFront. Ta obsahuje tzv. distribuce, které slouží pro kontrolu přístupu ke zdroji dat, v tomto případě ke složce v rámci Bucket kontejneru. Taková distribuce následně umožňuje velké množství nastavení pro konkrétní objekty, které jsou popsány níže v této podkapitole. (Amazon Web Services 2020b)

Vytvoření distribuce

Vytvoření nové distribuce je oproti předchozímu vytvoření S3 značně složitější. Z toho důvodu je tento proces detailněji popsán níže.

Na obrázku 21 je možné vidět hlavní nastavení nově vytvářené distribuce (části nedůležité pro tuto metodiku byly záměrně skryty). V horní části (1) je nutné nastavit hlavní parametry:

- *Origin Domain Name* – slouží pro výběr Bucket kontejneru, popřípadě jiného zdroje dat v rámci AWS.
- *Origin Path* – slouží k přesnějšímu určení dat v rámci kontejneru. V tomto případě je uvažovaný scénář vytvoření distribuce pro každou aplikaci. Tudíž zde je nutné vyplnit cestu v rámci struktury (např. /customization, jak bylo možné vidět na obrázku 10 v kapitole věnované představení služby).
- *Origin ID* – unikátní označení distribuce.

V druhé části (2) je dobré si všimnout možnosti nastavení tzv. CNAME záznamů. Ty slouží pro případy, kdy je vhodné mít vlastní doménové jméno a data načítat z takto vytvořené distribuce. Konkrétní kroky pro použití této možnosti jsou popsány v samostatné podkapitole níže.

AWS v tento okamžik neumožňuje používat vlastní doménu bez přiřazení platného certifikátu. Ten lze použít vlastní nebo si nechat zdarma vytvořit jinou službou v rámci AWS portfolia. Tímto procesem se zde dále nebude zabýváno, protože není nutný pro zavedení této metodiky a je nad rámec této práce a jeho zakoupení je individuální záležitostí každé organizace. Nicméně je zde zmíněn kvůli tomu, že bez vytvoření tohoto certifikátu nelze používat vlastní doménové jméno.

Ve čtvrté (4) sekci se definuje výchozí objekt, který se má použít v případě načtení URL dané distribuce. Často je nutné definovat takový objekt, který se stará o hlavní směrování uvnitř aplikace, v případě frameworku Angular se jedná o soubor *index.html*.

V poslední části (5) se pouze definuje stav, do kterého má distribuce přejít v okamžiku vytvoření. Pakliže by nebylo vhodné ihned distribuci začít používat, může být ve stavu *Disabled* do doby, než je vše připravené.

Create Distribution

Origin Settings

Origin Domain Name

1 Origin Path

Origin ID

Origin Custom Headers Header Name Value

Distribution Settings

Price Class Use All Edge Locations (Best Performance)

AWS WAF Web ACL None

2 Alternate Domain Names (CNAMEs)

SSL Certificate Default CloudFront Certificate (*.cloudfront.net)

Choose this option if you want your users to use HTTPS or HTTP to access you
https://d111111abcdef8.cloudfront.net/logo.jpg).
Important: If you choose this option, CloudFront requires that browsers or devic

Custom SSL Certificate (example.com):

3 Choose this option if you want your users to access your content by using an a
You can use a certificate stored in AWS Certificate Manager (ACM) in the US Ea
(N. Virginia) Region, or you can use a certificate stored in IAM.

Request or Import a Certificate with ACM

Learn more about using custom SSL/TLS certificates with CloudFront.
Learn more about using ACM.

Supported HTTP Versions HTTP/2, HTTP/1.1, HTTP/1.0
HTTP/1.1, HTTP/1.0

4 Default Root Object

Logging On
Off

Bucket for Logs

Log Prefix

Cookie Logging On
Off

Enable IPv6

Learn more

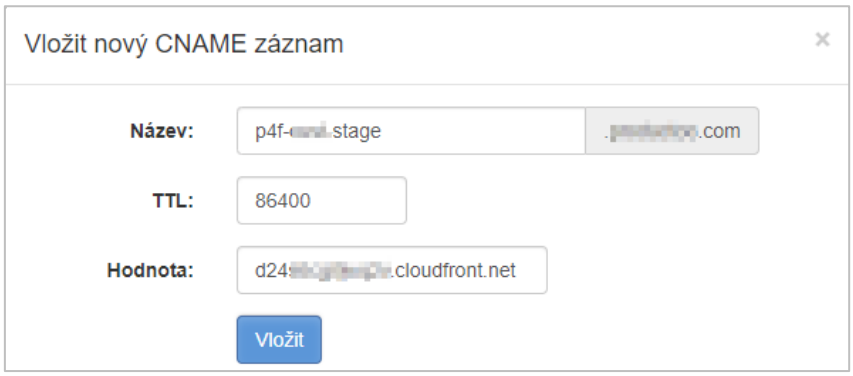
Comment

5 Distribution State Enabled
Disabled

Obrázek 21: Vytvoření distribuce CloudFront. Zdroj: autor

Přiřazení vlastní domény

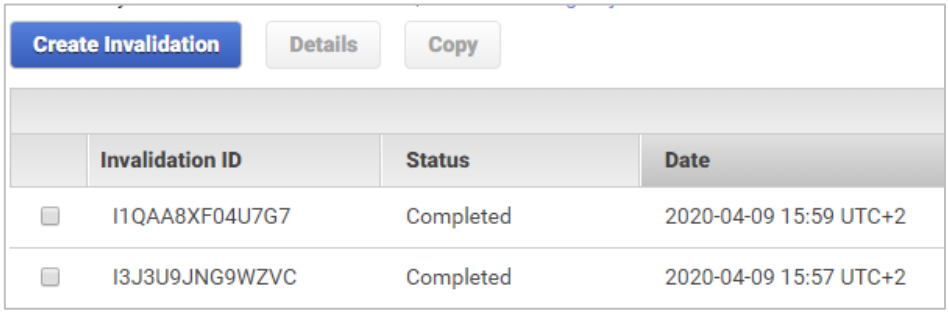
Pokud je chtěné použít vlastní doménu, je nutné kromě nastavení na straně CloudFront upravit i konfiguraci u registrátora domény. Zde je nutné nastavit tzv. CNAME záznam, který směřuje na název distribuce. Jinými slovy to znamená, že po vytvoření tohoto záznamu budou DNS servery vědět, že mají požadavky směrovat na servery AWS nikoliv na server registrátora domény. Tento záznam se přidává v rámci konfigurace DNS, jak lze vidět na obrázku 22 na příkladu konfigurace domény hostované u společnosti Blueboard. Kromě názvu vlastní domény je nutné ještě specifikovat TTL (Time to live). Ten se udává ve vteřinách a určuje, jak dlouho může záznam existovat, než je nutné znovu tento záznam načíst a případně upravit hodnotu v rámci sítě DNS serverů. Hodnota tohoto záznamu je převzata z CloudFront dané distribuce.



Obrázek 22: Vytvoření nového CNAME záznamu v administraci Blueboard. Zdroj: autor

Validace dat a jejich aktualizace

Poslední část je nastavení validací dat. Přestože se nejedná o krok nutný pro přípravu služby, je vhodné jej zde zmínit, protože umožňuje ručně objekty označit jako nevalidní a tím vynutit jejich aktualizaci ze zdrojového Bucket kontejneru. Provádí se v rámci Deployment procesu na závěr každého běhu. výpis předchozího vynucení obnovení souborů pro konkrétní distribuci je možné vidět na obrázku 23.



Obrázek 23: Ukázka přehledu vynucení obnovení souborů v rámci CloudFront. Zdroj: autor

5.6 Činnost: Spouštění testů a sestavení v Bitbucket nástroji

V rámci této činnosti je naplánován proces spouštění jednotlivých kroků v rámci nástroje Bitbucket. Ty byly vztaženy na vývojový proces ve firmě implementující tuto metodiku. Nicméně popisy jsou dostatečně obecné, takže je lze upravit podle potřeb pro jiné vývojové prostředí a procesy s nimi spojenými. Taková úprava spočívá ve změně konfiguračního souboru a přidání požadovaného kroku.

5.6.1 Kroky v rámci CI/CD procesu

V rámci běhu CI/CD procesu se provádí několik kroků, které vedou v případě bezchybného běhu k úspěšnému projití danou úlohou. Těmito kroky jsou:

- Příprava složek a uložení pořadí běhu do souboru.
- Vytvoření prostředí a instalace balíků.
- Jednotkové testy.
- E2E testy.
- Vytvoření sestavení.
- Deployment do prostředí.

Při běhu úlohy však není potřeba spouštět vždy všechny kroky, protože nástroj slouží nejen k finálnímu Deployment procesu, ale také jako nástroj pro průběžnou kontrolu funkčnosti zdrojového kódu. V takovém případě záleží na prvotní události spouštějící konkrétní úlohu.

V rámci této metodiky budou navrženy tři procesy, které se spouští na základě:

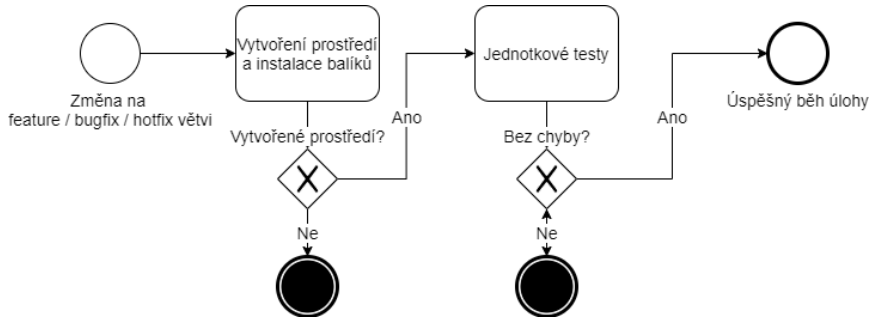
- Změny na větvích *typu feature, hotfix* nebo *bugfix*.
- Vytvoření nového *pull request* požadavku.
- Změny na *master* větvi.

Jejich posloupnost je předem daná vývojovými procesy. Prvním krokem jsou změny na větvích typu *feature, hotfix* nebo *bugfix,* následně se provádí *pull request* požadavek a jako poslední je integrace změn na *master* větev.

Všechny pipelines běží v Docker image s NodeJs ve verzi 12.13.0. Všechny zmíněné procesy používají node_modules cache obsahující knihovny Angular frameworku ve stejné verzi.

Změna na větvích typu feature, hotfix nebo bugfix

V případě větví *feature*, *hotfix* a *bugfix* je nutné provádět rychlé kontroly zdrojového kódu a zjišťovat, zda změny ovlivňují jiné části systému a tím nezpůsobily jejich případnou nefunkčnost. To je ověřováno dvěma rychlými testy. První zkouší instalovat prostředí a druhý spouští jednotkové testy nad částí aplikace, která byla změnami ovlivněna. Tento proces lze vidět na obrázku 24. Jakákoliv chyba v libovolném kroku způsobí okamžité zastavení běhu úlohy a vyhodnocení celého běhu jako chybové.



Obrázek 24: BPMN model procesu změny na větvích feature, hotfix nebo bugfix. Zdroj: autor

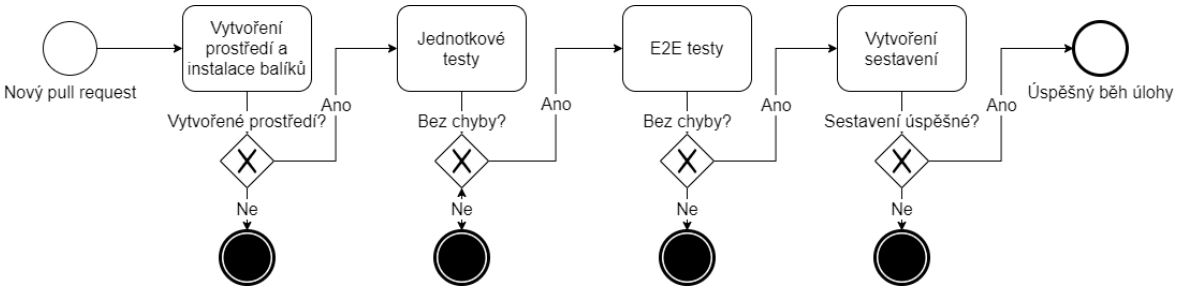
Tento proces neobsahuje žádné spouštění E2E testů nebo zkoušení vytvoření sestavení, protože se očekává velká chybovost v jednotlivých verzích a zároveň je důležitější rychlá zpětná vazba, která je v tomto případě v řádu minut. Konfiguraci těchto kroků je možné vidět ve výpisu 4.

Výpis 4: Konfigurace kroků na větvích typu feature, hotfix a bugfix. Zdroj: autor

```
# ... /zkráceno
  branches:
    '{feature/**,hotfix/**,bugfix/**}':
      - step:
          caches:
            - node
          name: Build and test
          script:
            - npm install && npm run postinstall
            - npm run affected:test
```

Vytvoření nového pull request požadavku

Druhý proces je zavolán v okamžiku vytvoření nového *pull request* pořadavku. Ten slouží především pro následné *code review*, u kterého je přehledně vidět poslední stav běhu úlohy a uživatel tak rychle vidí, zda je možné tyto změny bez problému integrovat na hlavní vývojovou větev. Proces je o několik kroků delší, protože obsahuje mimo instalace prostředí a spouštění jednotkových testů také E2E testy a zkušební vytvoření sestavení. Tyto dva poslední kroky ověřují kompletní funkčnost aplikace a její připravenost pro nasazení do produkčního prostředí. Celý proces je možné shlédnout na obrázku 25.



Obrázek 25: BPMN model procesu vytvoření nového pull request požadavku. Zdroj: autor

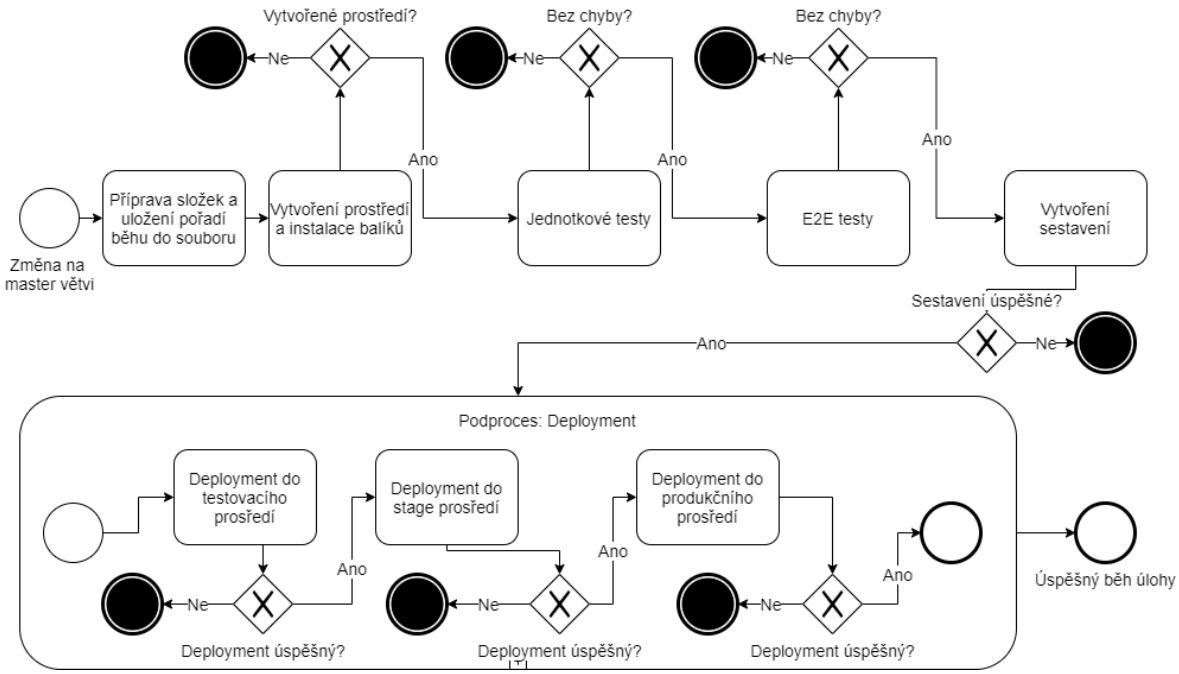
Stejně jako tomu bylo v předchozím procesu, i zde sebemenší chyba způsobí zastavení běhu úlohy a označení v tomto případě *pull request* požadavku jako nevalidního. Následně je nutný zásah vývojáře, aby nejdříve odstranil chyby způsobující problémy v běhu úlohy, a následně provedl opakované spuštění úlohy. Výslednou konfiguraci lze vidět ve výpisu 5.

Výpis 5: Konfigurace kroků při pull request požadavku. Zdroj: autor

```
# ... /zkráceno
pull-requests:
    '**':
      - step:
          caches:
            - node
          script:
            - npm install && npm run postinstall
            - npm run affected:test && npm run affected:e2e && npm run affected:build
```

Změna na master větvi

Posledním procesem je změna na *master* větvi. Ten je nejsložitější z jednoho prostého důvodu. Zde totiž probíhá výsledný Deployment proces do produkčního prostředí, a proto je nutné nejdříve spustit celou sadu testů a ověření funkčnosti sestavení, než následné nasazení proběhne. Celý proces je možné vidět na obrázku 26. Kromě předchozích kroků obsahuje přípravu složek, uložení pořadí běhu do souboru a následný Deployment proces. Ten zahrnuje nasazení do tří prostředí (test/stage/production). V případě, že v celém procesu není chyba, je výsledkem produkční prostředí s novými verzemi aplikací.



Obrázek 26: BPMN model procesu změny na master větvi

Opět výslednou konfiguraci pro nástroj Bitbucket je možné vidět ve výpisu 6 obsahující všechny zmíněné kroky v BPMN modelu procesu.

Výpis 6: Konfigurace kroků testování a sestavení v CI/CD nástroji. Zdroj: autor

```
# ... /zkráceno
    master:
      - step:
          caches:
            - node
          name: Build and test
          script:
            - mkdir dist/apps -p
            - echo "Build number ${BITBUCKET_BUILD_NUMBER}" > dist/apps/build_version.txt
            - npm install && npm run postinstall
            - npm run affected:test -- --base=origin/master~1 --head=origin/master
            - npm run affected:e2e -- --base=origin/master~1 --head=origin/master
            - npm run affected:build -- --prod --base=origin/master~1 --head=origin/master
          artifacts:
            - dist/apps/**
```

5.7 Činnost: Spouštění Deployment procesu

Posledním krokem v rámci CI/CD nástroje je Deployment proces. Jedná se o podproces v rámci Bitbucket nástroje, jak bylo možné vidět na obrázku 26 v předchozí kapitole. Tento podproces zahrnuje tři dílčí nasazení. U všech zmíněných nasazení se jedná o Deployment na AWS servery, odkud jsou aplikace následně přístupné.

Tento proces spočívá v postupném nasazování do tří zmíněných prostředích (test/stage/production). Stejně jako tomu bylo v předchozích krocích, jakmile libovolný Deployment selže, celá úloha je vyhodnocena jako chybná a dál se nepokračuje. V každém dílčím nasazení se provádí dva kroky. Prvním krokem je nahrání artefaktu sestavení každé aplikace do Amazon S3 Bucket kontejneru. To se provádí pomocí předem připravené úlohy poskytnuté v rámci Bitbucket. V této úloze tak stačí doplnit proměnné týkající se vlastní instance S3 a drobná nastavení. Samotné spojení se serverem a nakopírování artefaktu se již provede v rámci této předem nakonfigurované úlohy. Druhým krokem je pokyn pro stáhnutí aktuálních souborů z kontejneru do všech CloudFront instancí. Na tento krok je také připravena úloha v Bitbucket nástroji. Proto opět stačí nastavit proměnné týkající se serveru a následně úlohu spustit. Ta provede na každé specifikované distribuci vynucení obnovení souborů, které se tím pádem znovu nahrají z S3 Bucket kontejneru. Konfiguraci pro jednotlivé Deployment sekce je možné vidět ve výpisu 7.

Výpis 7: Konfigurace kroků Deployment v CI nástroji. Zdroj: autor

```
# ... /zkráceno
      - step:
          name: Deploy to test
          deployment: Test
          script:
            - pipe: atlassian/aws-s3-deploy:0.3.7
              variables:
                AWS_ACCESS_KEY_ID: $AWS_ACCESS_KEY_ID
                AWS_SECRET_ACCESS_KEY: $AWS_SECRET_ACCESS_KEY
                AWS_DEFAULT_REGION: $AWS_DEFAULT_REGION
                S3_BUCKET: 'p4f-dev.***.com'
                LOCAL_PATH: dist/apps
                EXTRA_ARGS: '--exclude=appConfig.json'
            - pipe: atlassian/aws-cloudfront-invalidate:0.1.1
              variables:
                AWS_ACCESS_KEY_ID: $AWS_ACCESS_KEY_ID
                AWS_SECRET_ACCESS_KEY: $AWS_SECRET_ACCESS_KEY
                AWS_DEFAULT_REGION: $AWS_DEFAULT_REGION
                DISTRIBUTION_ID: $DISTRIBUTION_ID_FC
# ... /zkráceno, podobná konfigurace nutná pro každé prostředí, pouze jiné hodnoty
```

Nicméně ani tento krok nezajistí kompletní funkčnost změny. Přestože se mohou všechny kroky úspěšně otestovat a aplikace je ve výsledku nasazena do produkčního prostředí, stále

se zde může vyskytnout chyba. V takovém případě je zvolena metoda rychlého opětovného nasazení předchozích verzí aplikací. To je možné provádět ručně přímo z Bitbucket nástroje, který umožňuje spustit již proběhlou úlohu opakovaně a aplikace tak znovu nasadit v předchozí verzi, ve které se chyba nevyskytuje a následně chybu v klidu opravit. Toto po přidělení dostatečných práv může provádět kdokoliv z firmy. Pokud si tak konzultant nebo obchodník všimne nějaké chyby, může dle svého uvážení v krajním případě udělat znovu nasazení sám bez pomoci vývojového týmu.

Samotná Deployment úloha, kterou lze vidět ve výpisu 7, spočívá v nastavení dvojice klíčů *AWS_ACCESS_KEY_ID* a *AWS_SECRET_ACCESS_KEY*. Tyto klíče jsou bezpečně uloženy v rámci Bitbucket administrace a nejsou přímo uložené ve zdrojovém kódu, což by potenciálně mohlo vést k bezpečnostním problémům. Vedle nastavení klíčů je nutné určit region, ve kterém je aplikace v rámci AWS nasazena. Následující trojice parametrů *S3_BUCKET, LOCAL_PATH* a *EXTRA_ARGS* slouží k nastavení konkrétního Amazon S3 Bucket kontejneru a definování cesty, kde se nachází artefakty pro Deployment proces. Poslední zmíněný parametr pak slouží k předání volitelných parametrů, které nejsou přímo nastaveny v rámci úlohy, v tomto případě se jedná o vyloučení konkrétního souboru z Deployment procesu. V rámci druhé úlohy sloužící pro obnovení souborů v distribuci je nutné opět nastavit klíče, ovšem následně již stačí nastavit pouze ID distribuce, ve které je požadovaná obnova souborů. Posléze není potřeba provádět další kroky a úloha se provede zcela automaticky, přičemž obnoví zdrojové soubory a tím dokončí aktualizaci dané distribuce.

5.8 Činnost: Reporting běhu CI/CD

Poslední neméně důležitou částí metodiky je vyhodnocení celého procesu. V rámci této činnosti jsou navrhnuty způsoby upozornění zainteresovaným osobám v rámci celého procesu. Nejblíže mají k procesu vývojáři, kteří jsou průběžně informováni o nutných krocích, které se od nich případně očekávají. Ostatní osoby dostávají pouze nezbytná upozornění, která se jich bezprostředně týkají.

5.8.1 Návrh kanálů pro reporting

Hlavním kanálem pro zasílání upozornění je e-mail. Ten slouží pro všechny zainteresované osoby. Druhý kanál určený pro notifikace je nástroj Slack, který je možné propojit s Bitbucket nástrojem. Posledním způsobem, jak sledovat jednotlivé činnosti jsou výpisy z běhu úloh Bitbucket a reporty ze služeb na AWS. Ty jsou určeny především pro DevOps architekta, aby byl schopen podrobně zkoumat jednotlivé běhy úloh a případně upravovat dílčí kroky.

V rámci této metodiky jsou identifikovány následující události, které jsou vhodné pro zasílání upozornění:

- Nový pull request požadavek.
- Testování a sestavení proběhlo úspěšně nebo chybně.
- Deployment do prostředí proběhl úspěšně nebo chybně.

Na základě kategorie jsou upozorněny zainteresované osoby od vývojářů, přes konzultanty až po obchodníky nebo potenciálně management.

E-mail

E-mail je výchozí komunikační kanál, ze kterého chodí valná většina informací automaticky na základě nastavení v rámci Bitbucket účtu. Je tak možné nechat si zasílat tato upozornění, popřípadě si vytvořit filtr v e-mailovém klientovi, aby byla zasílána jen taková upozornění, která jsou žádoucí. V rámci ulehčení lze toto nastavení udělat ve speciálním emailu pro danou skupinu uživatelů, ve kterém proběhne filtrace a přepošlou se pouze ty zprávy, které jsou pro danou skupinu nezbytné.

Slack

Druhým komunikačním kanálem je nástroj Slack, který slouží pro komunikaci v rámci týmů. Lze zde vytvářet kanály určené pro specifickou komunikaci, psát si přímé zprávy mezi uživateli, případně nastavit roboty, kteří dokážou sdílet informace ze služeb třetích stran (Slack 2020). V případě této metodiky se jedná především o zasílání notifikací o proběhlých úlohách v rámci Bitbucket nástroje.

Reporty v Bitbucket a AWS

Posledním způsobem pro reporting není přímo komunikační kanál, ale reporting v rámci Bitbucket nástroje nebo AWS služeb. V Bitbucket nástroji lze dohledat vždy podrobný výpis běhu úloh. V rámci AWS pomocí služby CloudWatch lze nastavit nejrůznější pravidla a události, které sledují běh služeb a v případě potřeby dokáže zasílat upozornění formou e-mailu (Amazon Web Services 2020c).

5.9 Shrnutí

V rámci této kapitoly byla navržena metodika pro zavedení Continuous Integration, Delivery a Deployment procesu s využitím nástroje Bitbucket a služeb Amazon S3 a CloudFront na platformě Amazon Web Services. Tím byl naplněn dílčí cíl *navrhnout metodiku pro zavedení Continuous Integration, Delivery a Deployment procesu*. Nejdříve byl popsán technologický základ aplikace, na kterém je závislý celý proces CI/CD pro snadnější pochopení jednotlivých činností. Následně byly definovány role související s metodikou a vznikající artefakty.

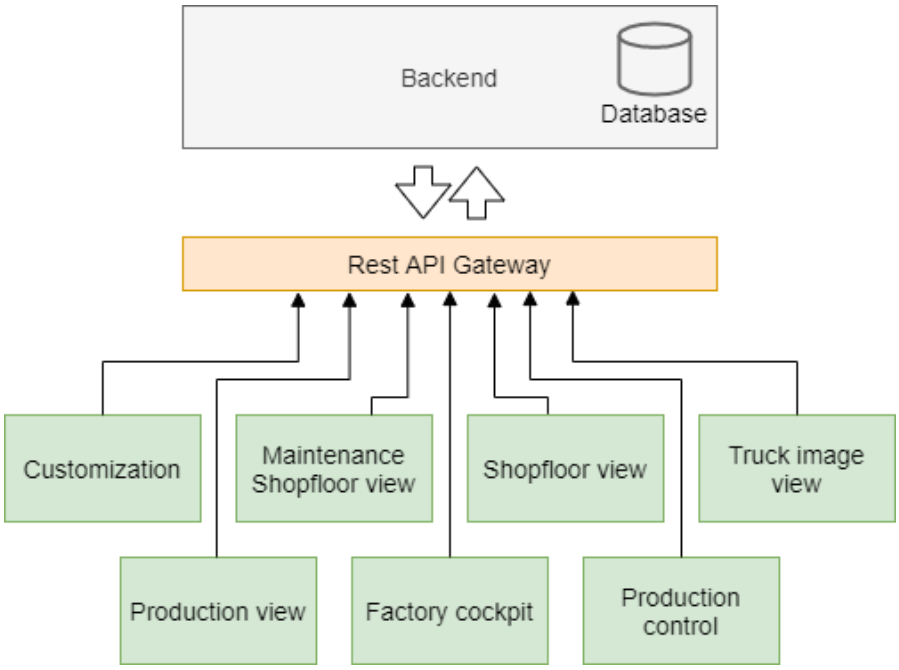
V dalších částech byl text věnován pěti činnostem, které se v rámci metodiky provádějí. Konkrétně se jednalo o plánování testování a sestavení aplikací, kde byly popsány dílčí kroky vedoucí k úspěšnému vytvoření sestavení bez chyb. Další činností byla příprava nástroje Bitbucket a služeb Amazon S3 a CloudFront pro pravidelné nasazování aplikace. Tato činnost zahrnovala podrobnou konfiguraci nezbytnou pro opakované spouštění úloh v rámci CI/CD nástroje. Třetí a čtvrtá činnost zahrnovala testování, vytváření sestavení a spouštění Deployment procesu. Poslední činnost byla věnovaná reportingu za použití dvou komunikačních kanálů.

6 Ověření metodiky na vybrané webové aplikaci

Cílem této kapitoly je ověření navrhnuté metodiky jejím zařazením do procesu vývoje skutečné webové aplikace a tím naplnit pátý dílčí cíl. V první části je popsaná architektura vybrané aplikace. V druhé části jsou pak podrobně rozepsány činnosti ve vztahu k této aplikaci. Závěr kapitoly je věnován vyhodnocení metodiky po pilotním nasazení a provozu. Metodika je ověřovaná na portfoliu aplikací, které jsou vyvíjené za účelem plánování a řízení výroby.

6.1 Popis architektury webové aplikace

Pro lepší pochopení jednotlivých činností je vhodné si nejdříve popsat architekturu vybrané webové aplikace. Ta se skládá z několika částí, přičemž metodiku lze aplikovat pouze na tu část využívající Angular framework. Na obrázku 27 je možné vidět diagram architektury celého řešení. Ten se skládá ze tří hlavních částí. Backend, který má na starost práci s databází, výpočty, automatické úlohy a přípravu Rest API Gateway. Rest API Gateway je druhou částí, která slouží jako vstupní brána k datům uchovávaných v databázi a spouštění předem definovaných úloh v backend části. Poslední částí jsou dílčí aplikace, které jsou záměrně rozděleny na sedm aplikací, protože jsou účelně vytvořeny pro konkrétní oblasti týkající se plánování a řízení výroby. Na těchto aplikacích je metodika ověřována.



Obrázek 27: Architektura webové aplikace. Zdroj: autor

Na obrázku 27 jsou ukázány následující aplikace:

- *Customization* – správa entit a nastavení celého systému.
- *Shopfloor view* – aplikace sloužící pro operátory pracující u výrobní linky.
- *Maintenance Shopfloor view* – aplikace pro operátory zodpovídající za údržbu.
- *Truck image view* –aplikace obsahující informace o stavu výroby před odesláním k zákazníkovi.
- *Production view* – Přehled aktuální výroby na dostupných výrobních linkách.
- *Factory cockpit* – Virtuální model závodu, ve kterém lze sledovat metriky v reálném čase.
- *Production control* – Plánovací a řídící systém pro výrobu.

Tyto aplikace jsou psány v rámci jednoho repozitáře. Jedná se o přístup k vývoji, při kterém je zdrojový kód všech aplikací uložen v rámci jednoho git repozitáře a libovolná změna může vyvolat nutnost vytvoření sestavení napříč všemi aplikacemi. Tento repozitář je vytvářen za použití NX CLI a Angular CLI. Samotné aplikace poté využívají Angular framework ve verzi 9 psaný v TypeScript jazyce.

Podrobnější popis pro účely této metodiky není nutný, protože na výslednou podobu metodiky aplikace nemají přímý vliv. Jednotlivé aplikace byly popsány spíše pro lepší pochopení názvů, které se vyskytují v dílčích výstupech při provádění jednotlivých kroků v rámci CI/CD procesu.

Je zde nutné zmínit, že metodika nezahrnuje řešení pro Rest API Gateway a Backend část, protože se jedná o zcela odlišné technologie a procesy, které by přesahovaly rozsah této práce.

6.2 Činnost: Plánování testování a tvorba sestavení

V rámci této činnosti je ověřena část metodiky, která se zabývá plánem testů a procesem sestavení aplikace. Vše je doplněno o výpisy z běhu z dílčích nástrojů. Proces ověření této činnosti je následující:

1. Plánování a tvorba ukázkového jednotkového testu.
2. Plánování a tvorba ukázkového E2E testu.
3. Plán procesu sestavení aplikace.

6.2.1 Plánování a tvorba ukázkového jednotkového testu

Nejdříve je nutné zvolit ukázkovou komponentu vhodnou pro psaní jednotkového testu. V tomto případě byla zvolena komponenta vypisující tabulkově strukturu materiálu, viz obrázek 28.

Material: ABC

LEVEL	ITEM	MATERIA	DESCRIPTION	QUANTITY	UNIT
.1	0010	CCC	Material CCC Sample Description	2	PCS
..2	0010	EEE	Material EEE Sample Description	12	PCS
.1	0020	DDD	Material DDD Sample Description	10	PCS

Obrázek 28: Ukázka komponenty vypisující strukturu materiálu. Zdroj: autor

Tato komponenta obsahuje tabulkový výpis struktury materiálu *ABC* skládající se ze dvou přímých potomků *CCC* a *DDD*, přičemž potomek *CCC* se skládá z materiálu *EEE*. Jednotkový test v tomto případě bude kontrolovat tři následující věci:

- Ověření vytvoření komponenty.
- Ověření funkčnosti formátování *Level* sloupce.
- Ověření funkčnosti formátování *Item* sloupce.

Výstupem výše uvedeného je test, který je možný vidět ve výpisu 8. Ten obsahuje kromě samotného testu i definice zdrojových dat, která jsou záměrně umístěna uvnitř tohoto testu pro jeho snazší pochopení, standardně by byly naimportovány z jiného souboru. Po nastavení testovacího frameworku v *beforeEach()* bloku proběhnou tři testy v *it()* funkci.

Výpis 8: Jednotkový test komponenty. Zdroj: autor

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';
import { BomTableComponent } from './bom-table.component'; // import komponenty

const bomTree = {
  element: {
    id: 1,
    materialCode: 'ABC',
    quantity: 1,
    materialDescription: 'T29754B',
    materialType: 'FING',
    materialUnit: 'TS'
  },
  children: [{
    element: {
```

```
      id: 2, materialCode: 'CCC', quantity: 2, materialUnit: 'PCS',
      materialDescription: 'Material CCC Sample Description', materialType: 'SFING'
    },
    children: [{
      element: {
        id: 4, materialCode: 'EEE', quantity: 12, materialUnit: 'PCS',
        materialDescription: 'Material EEE Sample Description', materialType: 'SFING'
      },
      children: []
    }]
  },
  {
    element: {
      id: 3, materialCode: 'DDD', quantity: 10, materialUnit: 'PCS',
      materialDescription: 'Material DDD Sample Description', materialType: 'SFING'
    },
    children: []
  }]
};

describe('BomTableComponent', () => {
  let component: BomTableComponent;
  let fixture: ComponentFixture<BomTableComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [BomTableComponent]
    }).compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(BomTableComponent);
    component = fixture.componentInstance;
    component.bomTree = bomTree;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });

  it('should format level indetation', () => {
    expect(component.padLevel(1)).toBe('.1');
    expect(component.padLevel(2)).toBe('..2');
    expect(component.padLevel(3)).toBe('...3');
```

```
  });

  it('should format item number', () => {
    expect(component.padItemNumber(1)).toBe('0010');
    expect(component.padItemNumber(25)).toBe('0250');
    expect(component.padItemNumber(30)).toBe('0300');
  });
});
```

Tento test lze ověřit jeho spuštěním přes příkaz *ng test shared-ui-bom-table*. Výsledkem tohoto spuštění je možné vidět na obrázku 29. Zde je možné vidět kromě běhu jednotkového testu i běh testu modulu obsahující tuto komponentu.

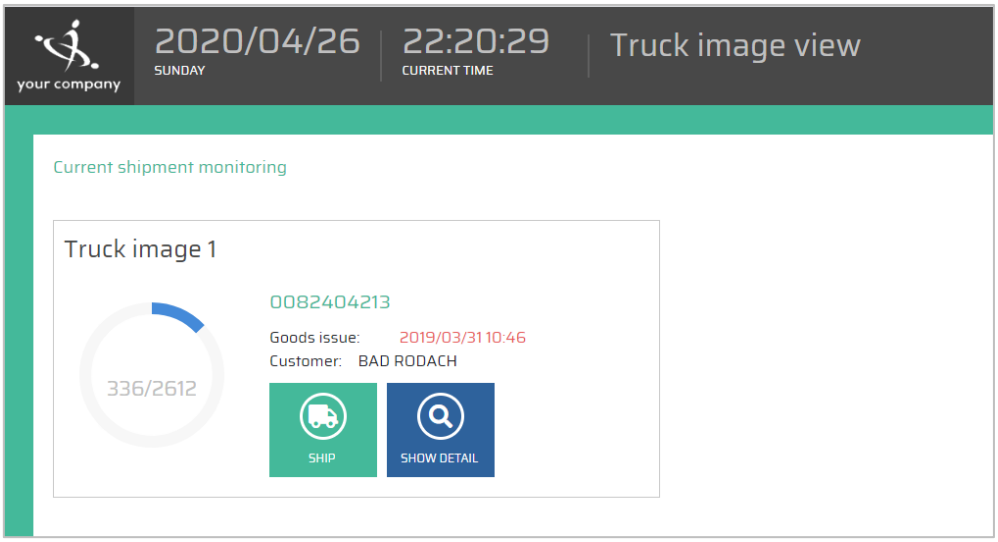
```
PS C:\web\www\productoo\p4f\frontend\monorepo> ng test shared-ui-bom-table
 PASS  libs/shared/ui-bom-table/src/lib/shared-ui-bom-table.module.spec.ts
 PASS  libs/shared/ui-bom-table/src/lib/bom-table/bom-table.component.spec.ts

Test Suites: 2 passed, 2 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        6.416s
Ran all test suites.
```

Obrázek 29: Výsledek běhu jednotkového testu komponenty. Zdroj: autor

6.2.2 Plánování a tvorba ukázkového E2E testu

Při psaní E2E testu je nutné nejdřív stanovit scénář, který bude prováděn v rámci jednotlivých kroků. Ukázka takového scénáře může být například načtení jedné z aplikací a kontrola zobrazených prvků na stránce. Pro tyto účely byla vybrána aplikace truck image view, kterou lze vidět na obrázku 30.



Obrázek 30: Obrazovka z aplikace truck image view. Zdroj: autor

Scénář pro E2E test je následující:

1. Načtení aplikace truck image view.
2. Zobrazení bloku truck image.
3. Kliknutí na tlačítko *Show detail* truck image.
4. Po kliknutí se zobrazí okno s detailem.
5. Zavření okna s detailem a vrácení na úvodní obrazovku.
6. Po kliknutí na tlačítko *Ship* blok truck image zmizí.

Přesný zápis takového testu je možné vidět ve výpisu 9.

Výpis 9: Zápis E2E testu. Zdroj: autor

```
describe('truck image view e2e', function () {
  it('should test truck image entity', () => {
    // 1. Načtení aplikace truck image view.
    cy.visit('/');

    // 2. Zobrazení bloku truck image.
    cy.get('productoo-truck-image-view-page productoo-ui-truck-image-
    preview').should('exist');

    // 3. Kliknutí na tlačítko Show detail truck image.
    cy.get('productoo-ui-truck-image-preview > productoo-ui-button:nth-child(2)').click();

    // 4. Po kliknutí se zobrazí okno s detailem.
    cy.get('productoo-feature-truck-image-dialog').should('exist');

    // 5. Zavření okna s detailem a vrácení na úvodní obrazovku.
    cy.get('productoo-feature-truck-image-dialog productoo-ui-button').click();
    cy.url().should('equal', '/');

    // 6. Po kliknutí na tlačítko Ship blok truck image zmizí.
    cy.get('productoo-ui-truck-image-preview > productoo-ui-button:nth-child(1)').click();
    cy.get('body productoo-truck-image-view-page productoo-ui-truck-image-
    preview').should('be.undefined');
  })
});
```

Jednotlivé kroky jsou provedeny v rámci E2E testu entity truck image. Výsledek běhu je možné vidět na obrázku 31.

```
(Run Finished)

    Spec                                      Tests  Passing  Failing  Pending  Skipped
  √ app.spec.ts                     00:01       1        1        -        -        -
  √ All specs passed!               00:01       1        1        -        -        -
```

Obrázek 31: Výsledek běhu E2E testu. Zdroj: autor

6.2.3 Plán procesu sestavení aplikace

V rámci tohoto kroku je ověřena možnost vytvářet sestavení na základě ovlivněných částí aplikací. Jinými slovy ověřit, že v případě změny v komponentě využívané v jedné z aplikací dojde k vytvoření sestavení právě těchto aplikací obsahující tuto komponentu.

Při vytváření sestavení lze používat dva způsoby:

- Vytváření sestavení pro každou aplikaci zvlášť pomocí Angular CLI.
- Vytváření sestavení všech aplikací za pomocí NX CLI.

První možnost umožňuje vytvořit sestavení pro každou aplikaci za pomocí příkazu *ng build*. Nicméně má to dvě úskalí. Prvním je nemožnost spustit naráz vytváření sestavení pro všechny aplikace a druhým nemožnost spustit sestavení jen pro aplikace ovlivněné provedenou změnou. V tento okamžik je vhodnější použít nástroj NX CLI, který umí provést analýzu kódu a dokáže sestavit strom závislostí, na základě kterého vygeneruje příkazy pro vytvoření sestavení ovlivněných aplikací. Tímto příkazem je *nx affected:build*. Testování tohoto je prováděno při změně ve stejné komponentě, na které byl ukazován jednotkový test. Tato komponenta je využívána ve dvou aplikacích, konkrétně v *production-control* a *shopfloor-view*. Spuštění tohoto příkazu (ve variantě affected:apps, která nevytvoří sestavení, nýbrž pouze vypíše ovlivněné aplikace) je možné vidět na obrázku 32, kde v levé části byl příkaz *affected* spuštěn bez dalších argumentů, proto byly nalezeny jen dvě zmíněné aplikace. V pravé části byl tento příkaz spuštěn s parametrem *–all,* který vynutí výpis všech aplikací obsažených v repozitáři, nehledě na provedené změny v komponentě.

```
PS C:\web\www\productoo\p4f\frontend\monorepo>
 nx affected:apps
> NX  Affected apps:

  - shopfloor-view
  - production-control

PS C:\web\www\productoo\p4f\frontend\monorepo>
```

```
PS C:\web\www\productoo\p4f\frontend\monorepo> nx affe
cted:apps --all

> NX  WARNING  Running affected:* commands with --al
l can result in very slow builds.

  --all is not meant to be used for any sizable projec
t or to be used in CI.

  Learn more about checking only what is affected: htt
ps://nx.dev/guides/monorepo-affected.


> NX  Affected apps:

  - maintenance-shopfloor-view
  - production-control
  - truck-image-view
  - factory-cockpit
  - production-view
  - shopfloor-view
  - customization
```

Obrázek 32: Porovnání spuštění příkazu s analýzou vlivu změny na aplikace. Zdroj: autor

Tato funkcionalita má obrovský vliv na efektivnost spouštění sestavení v CI/CD nástroji, kde se často platí na bázi čerpání minut běhu programu. Díky tomu je možné na příkladu vidět až třetinovou úsporu času. Ta se samozřejmě liší vzhledem k velikosti změny ve zdrojovém kódu.

Důležité je zmínit, že může nastat situace, při které jsou ovlivněny všechny aplikace a je třeba provést nejpomalejší variantu sestavení, a to všech sedmi aplikací.

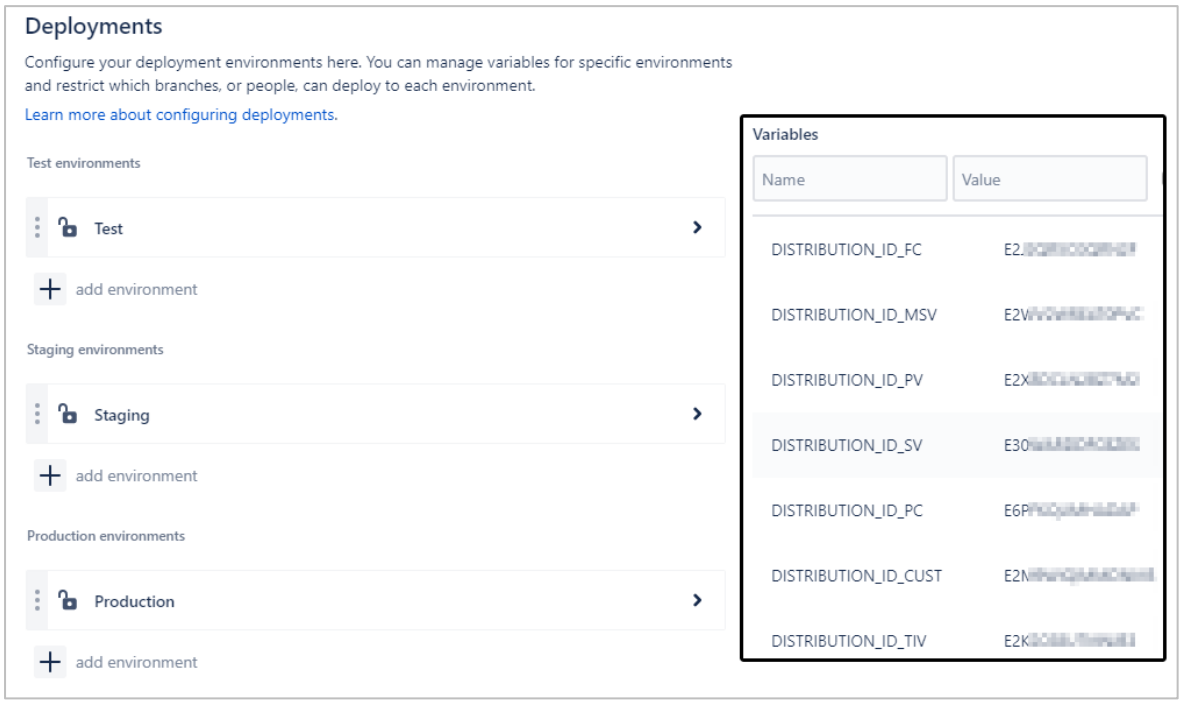
6.3 Činnost: Prvotní nastavení nástroje a služeb

V rámci této činnosti je ověřena metodika z hlediska nastavení nástroje Bitbucket a služeb Amazon S3 a CloudFront. Postupně jsou provedeny kroky vedoucí ke zprovoznění těchto částí, jejichž výstupem je dokumentace tohoto nastavení. Tyto kroky spočívají v provedení následujících kroků:

1. Vytvoření a nastavení repozitáře v Bitbucket.
2. Vytvoření struktury v rámci Amazon S3 a CloudFront.

6.3.1 Vytvoření a nastavení repozitáře v Bitbucket

Prvním krokem při ověření nastavení služeb je vytvoření repozitáře v Bitbucket nástroji. Následně je provedeno nastavení tohoto repozitáře, aby bylo možné spouštět následný CI/CD proces na základě předem definovaných událostí. Těmi jsou nový *pull request* požadavek, změna na *master* větvi nebo změna na *feature/bugfix/hotfix* větvi. Jak bylo popsáno v kapitole 5.5.1, vytvoření nového repozitáře je triviální záležitost, která spočívá v definování názvu a kategorizaci. Ukázka vytvoření byla uvedena v téže kapitole na obrázku 17. Následně je však nutné nastavit repozitář tak, aby bylo možné provádět Deployment aplikace na AWS.



Obrázek 33: Nastavení proměnných v Bitbucket nástroji. Zdroj: autor

Na obrázku 33 je možné v levé části vidět tři prostředí: Test, Staging a Production, přičemž u každého prostředí lze nastavit jiné hodnoty pro proměnné, které se používají v rámci jednotlivých úloh CI/CD procesu. Ty lze vidět v pravé části obrázku. Jedná se především o ID jednotlivých distribucí v rámci služby CloudFront, které jsou aktualizovány při posledním kroku nasazení do zmíněných prostředí.

Vedle nastavení jednotlivých prostředí lze definovat i proměnné pro celý repozitář. Těmi jsou především klíče pro přístup k AWS, které jsou neměnné bez ohledu na konkrétní prostředí. Záměrně zde nejsou uvedeny, protože se jedná o citlivá data.

6.3.2 Vytvoření struktury v rámci Amazon S3 a CloudFront

Ověření tohoto kroku probíhá validací navrhnuté struktury pro založení Bucket kontejnerů a souborové struktury. Následně je ověřeno, zda tato struktura vyhovuje reálným podmínkám pro přístup k aplikacím přes vlastní doménová jména ve službě CloudFront.

V rámci Amazon S3 jsou vytvořeny tři kontejnery, každý zvlášť pro konkrétní prostředí. Uvnitř každého kontejneru je vytvořena souborová struktura, která odpovídá názvům aplikací. Tento postup je shodný s tím, který byl uveden v představení služby Amazon S3 a lze se tak na strukturu podívat na obrázku 10 v kapitole 4.1.

Druhým krokem je vytvoření distribucí ve službě CloudFront. Tento seznam je větší, protože každá aplikace má v každém prostředí vlastní URL adresu a doménové jméno. Celkem se tak jedná o 21 distribucí, které je nutné vytvořit a nakonfigurovat. Jejich vytváření probíhalo podle metodiky a příklad takové konfigurace byl uveden při představení služby CloudFront na obrázku 11 v kapitole 4.2. V rámci každé distribuce bylo nutné dodatečně nakonfigurovat pravidla pro vyhodnocování 403 a 404 chyb, jelikož podle dokumentace Angular

frameworku je nutné všechny adresy směrovat na hlavní soubor *index.html* (2020). To bylo provedeno v konfiguraci směrování distribuce, jak lze vidět na obrázku 34.

	HTTP Error Code	Error Caching Minimum TTL	Response Page Path	HTTP Response Code
☐	403	300	/index.html	200
☐	404	300	/index.html	200

Obrázek 34: Směrování chybných požadavků v nastavení distribuce ve službě CloudFront. Zdroj: autor

Posledním krokem je úprava nastavení distribuce tak, aby bylo možné používat vlastní doménová jména pro každou distribuci. To bylo provedeno nastavením CNAME záznamů podle metodiky, jak bylo uvedeno v podkapitole 5.5.3. Ověření proběhlo zadáním adresy do prohlížeče a kontrolou výstupu společně s HTTP návratovým kódem. Na obrázku 35 je možné tuto odpověď vidět společně s IP adresou AWS serverů a označením, že se jedná o odpověď poskytnutou přes CloudFront z Amazon S3 kontejneru.

URL požadavku: http://p4f-cust-stage.[redacted]
Metoda požadavku: GET
Vzdálená adresa: 13.224.0.126:80
Stavový kód: 200 OK
Verze: HTTP/1.1

Filtr hlaviček

Hlavičky odpovědi (461 B)

Accept-Ranges: bytes
Age: 71
Connection: keep-alive
Content-Length: 910
Content-Type: text/html
Date: Sun, 26 Apr 2020 13:40:34 GMT
ETag: "8120ee78d53e6ab8279b7db73193f8f3"
Last-Modified: Fri, 24 Apr 2020 14:56:50 GMT
Server: AmazonS3

Obrázek 35: HTTP odpověď z vlastní domény. Zdroj: autor

6.4 Činnost: Spouštění testů a sestavení v Bitbucket nástroji

Tato činnost je ověřena kontrolovaným spuštěním úlohy, která je nastavena na hlavní vývojové větvi. Ta je z pohledu aplikací nejdůležitější, protože při ní dochází ke spouštění všech typů testů, sestavení a v neposlední řadě nahrání do produkčního prostředí. Poslední krok je předmětem následující činnosti v podkapitole 6.5. Zbylé kroky jsou nastaveny následovně:

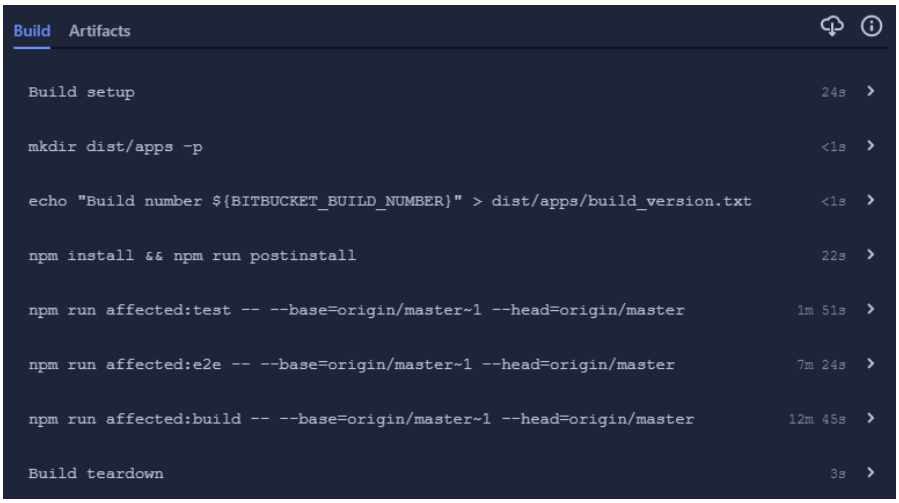
1. Vytvoření složky pro artefakty.
2. Zapsání aktuálního pořadí úlohy do souboru.
3. Instalace balíčků pro spuštění testů a sestavení.
4. Spuštění jednotkových testů.
5. Spuštění E2E testů.
6. Spuštění sestavení aplikací.

Výše uvedené spouštění se provádí s příkazem *affected*, aby nebyl zbytečně spouštěn set všech testů nebo sestavení všech aplikací, nýbrž pouze ty, které mohly být potenciálně změnou ovlivněny. Jejich konfiguraci si lze prohlédnou ve výpisu 10:

Výpis 10: Reálné nastavení kroků v CI/CD nástroji pro změnu na mastesr větvi. Zdroj: autor

```
script:
 - mkdir dist/apps -p
 - echo "Build number ${BITBUCKET_BUILD_NUMBER}" > dist/apps/build_version.txt
 - npm install && npm run postinstall
 - npm run affected:test -- --base=origin/master~1 --head=origin/master
 - npm run affected:e2e -- --base=origin/master~1 --head=origin/master
 - npm run affected:build -- --prod --base=origin/master~1 --head=origin/master
```

Výsledek běhu z CI/CD nástroje je možné vidět na obrázku 36. Ověření bylo úspěšné, nýbrž se spustily jen žádoucí testy a celý běh úlohy proběhl bez chyby.



Obrázek 36: Výsledek běhu sestavení a testování v CI/CD nástroji při změně na master větvi. Zdroj: autor

6.5 Činnost: Spouštění Deployment procesu

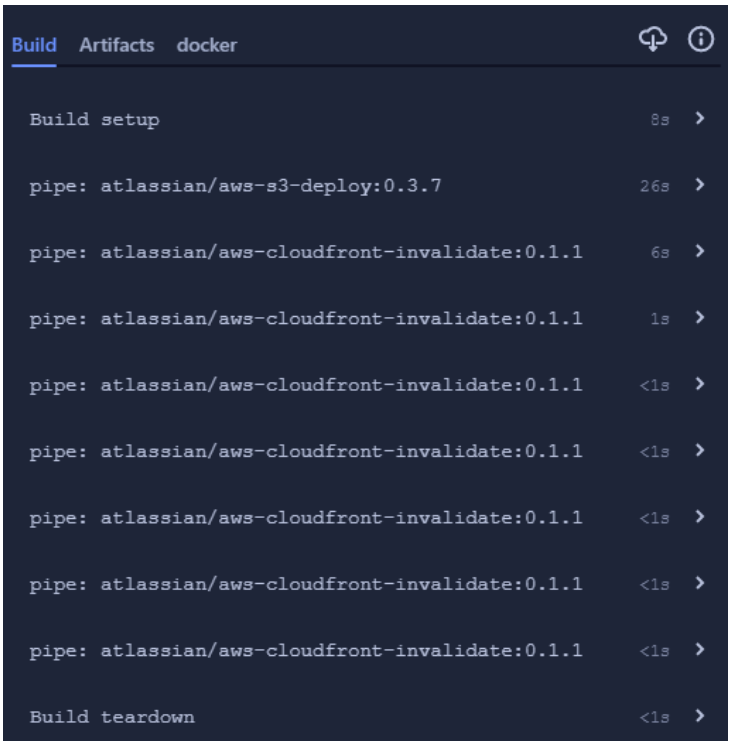
Tato činnost spočívá v provedení Deployment procesu v rámci Amazon S3 a CloudFront. Ta je přímo závislá na předchozí přípravě nástroje a služeb, přičemž její provedení je následně triviální spočívající ve spuštění předem nadefinovaných úloh a jejich kontrole.

Tyto úlohy spočívají ve využití generických skriptů poskytnuté Bitbucket nástrojem. Jejich dílčí konfiguraci je možné vidět ve výpisu 11.

Výpis 11: Konfigurace pro spouštění Deployment procesu do testovacího prostředí. Zdroj: autor

```
    - step:
        name: Deploy to test
        deployment: Test
        script:
          - pipe: atlassian/aws-s3-deploy:0.3.7
            variables:
              AWS_ACCESS_KEY_ID: $AWS_ACCESS_KEY_ID
              AWS_SECRET_ACCESS_KEY: $AWS_SECRET_ACCESS_KEY
              AWS_DEFAULT_REGION: $AWS_DEFAULT_REGION
              S3_BUCKET: 'p4f-dev.***.com'
              LOCAL_PATH: dist/apps
              EXTRA_ARGS: '--exclude=appConfig.json'
          - pipe: atlassian/aws-cloudfront-invalidate:0.1.1
            variables:
              AWS_ACCESS_KEY_ID: $AWS_ACCESS_KEY_ID
              AWS_SECRET_ACCESS_KEY: $AWS_SECRET_ACCESS_KEY
              AWS_DEFAULT_REGION: $AWS_DEFAULT_REGION
              DISTRIBUTION_ID: $DISTRIBUTION_ID_FC
```

Tato konfigurace obsahuje dvě části. První obsahuje nahrání artefaktů do Amazon S3 kontejneru. V rámci konfigurace je nutné nastavit proměnné popsané v podkapitole 6.3. Zde se jedná o testovací prostředí s označením *Test* a kontejnerem *p4f-dev.***.com*. Druhá část obsahuje skript, který vynutí obnovení zdrojových souborů v CloudFront distribuci, jelikož tyto soubory se ukládají do cache kvůli rychlejšímu zpracování požadavků. Výsledkem tohoto běhu je nahrání souborů do Amazon S3 kontejnerů a vytvoření nového požadavku obnovení souborů, který je možné vidět na obrázku 23 v podkapitole 5.5.3. Tento běh úlohy je možné si prohlédnout na obrázku 37. Zmíněný Deployment probíhá pro každé prostředí zvlášť.



Obrázek 37: Výsledek běhu Deployment procesu v Bitbucket nástroji. Zdroj: autor

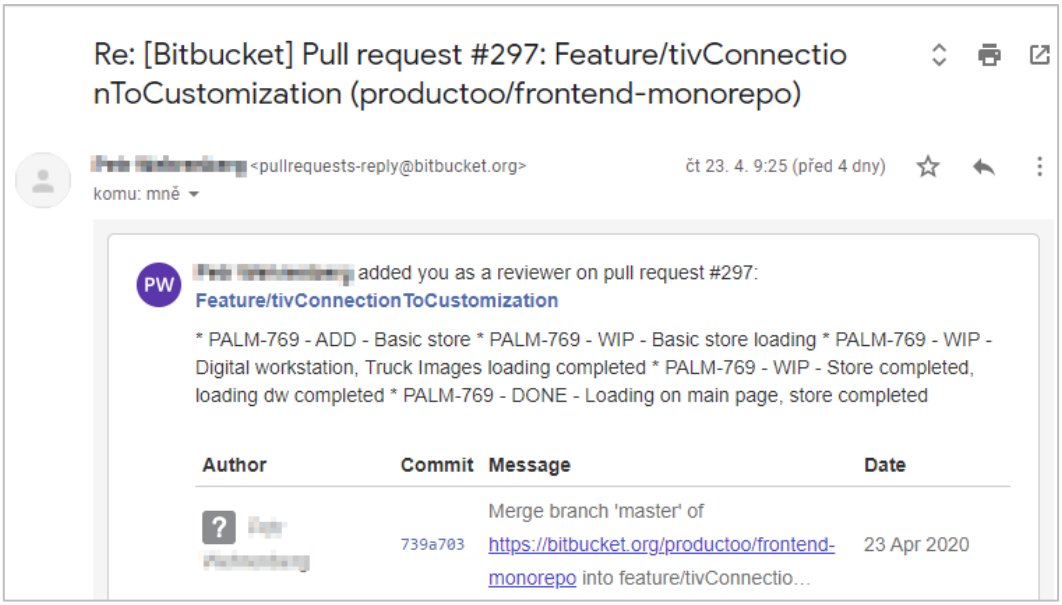
Těmito kroky bylo ověřeno, že Deployment proces po provedení nezbytných nastavení proběhl úspěšně při změně zdrojového kódu, která byla později zařazena do hlavní vývojové větve *master*.

6.6 Činnost: Reporting běhu CI/CD

Činnost reportingu neboli sledování a vyhodnocování běhu jednotlivých úloh, je realizována pomocí dvou komunikačních kanálů. Jedná se především o zasílání e-mailů a upozorňování v rámci komunikačního nástroje Slack. Oba tyto kanály jsou využívány ve firmě, ve které je tato metodika zaváděna a ověřována její funkčnost a úplnost.

E-mail

V rámci e-mailové komunikace Bitbucket umožňuje zasílat zprávy o jednotlivých událostech vznikající v nástroji. Do e-mailů zahrne vždy ty uživatele, kteří se účastní dané aktivity. Pokud je řeč například o novém *pull request* požadavku, nástroj automaticky zasílá upozornění autorovi a osobě, která je zodpovědná za schválení tohoto požadavku. Obě strany tak dostávají veškeré potřebné informace. Ukázku zmíněných upozornění je možné vidět na obrázku 38.

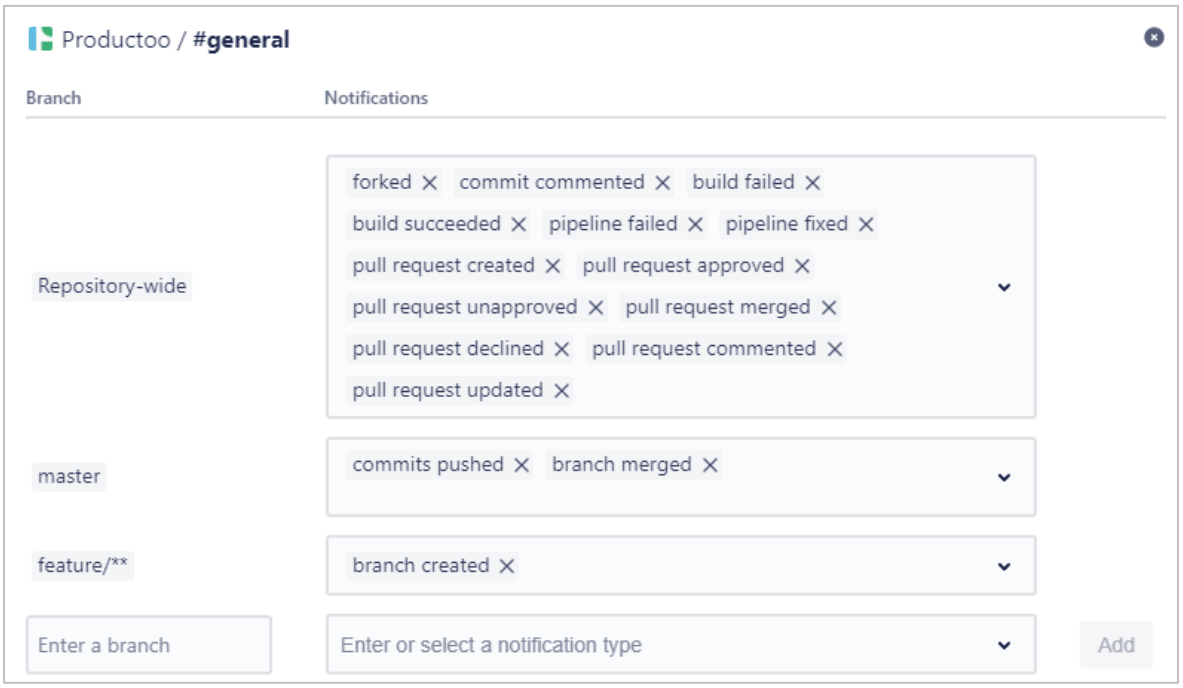


Obrázek 38: Ukázka e-mailové notifikace z nástroje Bitbucket. Zdroj: autor

Podobná upozornění jsou zasílána i při nové verzi na *master* větvi nebo při chybném běhu úlohy.

Slack

Slack je komunikační nástroj sloužící pro způsob komunikace na bázi chatovací aplikace. Ovšem kromě klasických zpráv mezi uživateli lze nastavit i zasílání notifikací z různých služeb, mezi kterými nechybí podpora nástroje Bitbucket. Pro zprovoznění této komunikace je zapotřebí nastavit události pro zasílání notifikací. Jejich výčet je možné vidět na obrázku 39.



Obrázek 39: Nastavení propojení nástroje Bitbucket a Slack. Zdroj: autor

Na základě tohoto nastavení budou automaticky zasílány notifikace ohledně nových změn, *pull request* požadavcích, chybách běhu úloh nebo vytváření nových větví. Toto nastavení lze libovolně přizpůsobit vlastním požadavkům. Ukázka takové notifikace je na obrázku 40.

Obrázek 40: Ukázka notifikace v nástroji Slack. Zdroj: autor

6.7 Vyhodnocení metodiky

Při zavádění této metodiky byly postupně aplikovány jednotlivé činnosti a ověřovány, zda naplňují předem daný scénář. Z tohoto procesu vyplynula následující zjištění, která je vhodné okomentovat a zhodnotit.

Celý proces CI/CD je nutné neustále aktualizovat a ověřovat jeho správnou funkčnost. Tento problém se objevil v průběhu návrhu metodiky, jelikož byly ve firmě vytvořeny další tři aplikace, na kterých ověření neproběhlo, jelikož jejich stavba byla odlišná od zbylých aplikací. Je proto dobré mít na paměti, že změny v procesech a aplikacích se mohou kdykoliv objevit a je nutné na ně reagovat.

Další potenciální problém je ve využívání většího množství služeb, které společně utváří celou metodiku. To klade větší nároky na znalosti lidí, kteří by s procesem CI/CD měli pracovat, protože musí dobře pochopit specifika každého použitého nástroje a služby. Na tomto místě však lze nyní doporučit větší integraci na AWS, protože při vytváření metodiky se tato služba zdá mnohem stabilnější a lépe škálovatelná v případě potřeby většího výpočetního výkonu. Navíc po dobu vytváření a testování se službami AWS nebyly žádné problémy, oproti tomu byly zaznamenány určité výpadky v dostupnosti nástroje Bitbucket, což určitě není žádoucí v případě navrhovaných procesů, které mají za cíl neustálou kontrolu a dostupnost.

Na druhou stranu je nutné podotknout, že zavedení této metodiky nijak nezpomalilo vývojový proces ve firmě, nýbrž zrychlilo uvolňování nových změn ke koncovým zákazníkům. V tabulce 3 lze vidět konkrétní čísla spojená se zavedením metodiky.

Tabulka 3: Porovnání Deployment procesu před a po zavedení metodiky. Zdroj: autor

	Před zavedením metodiky	**Po zavedení metodiky**
Počet nasazení na produkční prostředí	1–2x týdně	1–3x denně
Chybovost při vytváření sestavení pro produkční prostředí	Každé cca 10. sestavení	2 výskyty z 250 sestavení celkem (v době vyhodnocení pilotního běhu)
Čas od změny v kódu po nasazení na produkční prostředí	V rozmezí dnů až týdnů	V rozmezí desítek minut až dnů
Rychlost řešení produkčního defektu (nefunkčnost aplikace)	Řádově hodiny	Řádově minuty (docíleno navrácením předchozí verze)

V tabulce 3 jsou shrnuty poznatky, které byly nasbírány po pilotním zavedení metodiky. Největší efekt měla metodika na aktuálnost produkčního prostředí, které je nyní aktualizováno i několikrát za den. Vedle toho je dobré zmínit i dobu řešení produkčního defektu (nefunkčnost aplikace), která se z řádu hodin posunula řádově na minuty. To díky možnosti provést opětovný *Deployment* předchozí verze, aniž by muselo být prostředí dlouhou dobu nefunkční kvůli chybě. Zároveň výše uvedené mělo pozitivní efekt na rychlejší kontrolu funkcionality, jelikož ta je dostupná v některých případech ještě ten samý den a není nutné tak čekat na aktualizaci prostředí 1–2x týdně.

6.8 Shrnutí

Cílem této kapitoly bylo *ověřit navrhnutou metodiku na vybrané webové aplikaci*, čímž byl splněn pátý dílčí cíl.

Nejdříve byla popsána architektura celého řešení a zároveň upřesněno, na jaké části řešení je možné metodiku aplikovat. Následně byly provedeny dílčí činnosti od plánování testování a tvorba sestavení, prvotní nastavení nástroje a služeb, přes spouštění testů a sestavení v Bitbucket nástroji a provádění Deployment procesu až po reporting běhu CI/CD nástroje. Jednotlivé činnosti metodiky byly vždy provedeny na základně scénáře, který byl konzultován s interním vývojovým týmem s cílem ověřit danou činnost v rozsahu daném metodikou. Na základě toho byly zhodnoceny přínosy, které tato metodika měla v rámci zavedení v dané firmě.

Splněním všech dílčích činností byla metodika úspěšně ověřena. Její zavedení mělo pozitivní dopad na vývoj jednotlivých aplikací celého řešení, protože umožnila zrychlení nasazování nových verzí do produkčního prostředí, zkrácení cyklu od zadání nového požadavku na vývoj po jeho nasazení nebo rychlost řešení výpadku způsobený chybou v kódu. Zároveň neustálá kontrola aplikace spouštěním testů a sestavení umožnila nasazování do produkčního prostředí s mnohem větší jistotou a eliminovala manuální činnosti, které by mohly potenciálně vést k chybě a výpadku běhu aplikací.

Závěr

Hlavním cílem této diplomové práce bylo navrhnout metodiku, která týmům pomůže zavést Continuous Integration, Delivery a Deployment proces do svých vývojových činností s využitím nástroje Bitbucket a vybraných služeb na Amazon Web Services. Toho bylo dosaženo pomocí pěti dílčích cílů.

Prvním definovaným dílčím cílem bylo vymezit zavádění DevOps ve spojení s Continuous Integration, Delivery a Deployment procesem. Toho bylo docíleno v rámci druhé kapitoly, kde byl čtenář seznámen s problematikou týkající se vývoje softwaru a za pomocí literatury byly nejdříve formulovány principy týkající se samotného DevOps. Následně byly detailně popsány dílčí procesy Continuous Integration, Delivery a Deployment s napojením na navrhovanou metodiku.

Třetí kapitola byla věnována představení nástroje Bitbucket, který je použit pro Continuous Integration a Delivery proces. Velký důraz byl kladen na ty části nástroje, které tvořily základ pro samotnou metodiku, jelikož nástroj má na starost orchestraci celého procesu od změny ve zdrojovém kódu až po nasazení do produkčního prostředí. Jednalo se především o Bitbucket Pipelines a Deployments, které byly popsány za pomocí ukázek z reálného vývoje aplikace. Tímto byl naplněn druhý dílčí cíl stanovený v úvodní kapitole této práce.

Třetím dílčím cílem bylo představit služby na Amazon Web Services podporující Continuous Deployment proces. Ten byl naplněn v rámci čtvrté kapitoly. Nejdříve byla AWS platforma stručně popsaná a bylo přiblíženo, jaké výhody přináší. Následně byl text věnován dvěma hlavním službám Amazon S3 a CloudFront, které byly použity pro Deployment proces produkčního sestavení aplikace. Tyto služby byly představeny do takové míry, aby bylo možné popsat metodiku v zamýšleném rozsahu a čtenář se v dílčích nastaveních neztrácel a chápal důležitost těchto služeb v rámci celého CI/CD procesu.

Pátá kapitola byla věnována samotnému návrhu metodiky, čímž byl naplněn čtvrtý dílčí cíl navrhnout metodiku pro zavedení Continuous Integration, Delivery a Deployment procesu. Ta se skládá z pěti dílčích činností. První z nich konkrétně plánování testování a tvorba sestavení je zaměřena na popis testování a způsoby vytváření produkčního sestavení. Druhá činnost s názvem prvotní nastavení nástroje a služeb využívá předchozích kapitol pro vytvoření konkrétních kroků, které vedou k úspěšnému přizpůsobení služeb do stavu vyhovujícímu navrhované metodice. Třetí činnost věnovaná spouštění testů a sestavení v Bitbucket nástroji je zaměřena na konkrétní úlohy potřebné k jejich provedení v rámci nástroje. Toho je docíleno za pomocí vytvoření konfigurace, kterou lze snadno přizpůsobovat podle vlastních potřeb. Čtvrtá činnost s názvem spouštění Deployment procesu je věnovaná závěrečné fázi procesu s názvem Continuous Deployment, která popisuje automatické nasazení sestavení do produkčního prostředí. Poslední činností je reporting běhu CI/CD procesu.

Posledním dílčím cílem bylo ověřit navrženou metodiku na vybrané webové aplikaci. Tomu byla věnována šestá kapitola, která nejprve popsala celé řešení skutečné webové aplikace

vyvíjené za účelem řízení výroby. Následně byly činnosti aplikované na dílčí části tohoto řešení psané v Angular frameworku. Závěrečná část této kapitoly byla věnována vyhodnocení zavedení metodiky a shrnutí poznatků, které byly identifikovány.

Všechny dílčí cíle, které byly v úvodní části této práce definovány byly naplněny, čímž byl naplněn i hlavní cíl práce. Na tomto místě je však vhodné zmínit, že oblast Continuous Integration, Delivery a Deployment je velmi široká a aplikace metodiky musí být vždy přizpůsobena konkrétní situaci v dané organizaci za pomoci nástrojů, které jsou k dispozici. Práce má přínos především pro čtenáře, kteří dokáží metodiku lehce upravit na základě svých potřeb, nebo používají jednu z dílčích služeb. Zároveň jim umožní nahlédnout do zavedení celého procesu CI/CD opřeného o výsledky z reálné webové aplikace.

Použitá literatura

AHMED, Junaid, 2019. How DevOps Can Help Speed Go to Market. *DevOps.com* [online] [vid. 2020-04-01]. Dostupné z: https://devops.com/how-devops-can-help-speed-go-to-market/

AMAZON WEB SERVICES, 2020a. About AWS. *Amazon Web Services, Inc.* [online] [vid. 2020-03-23]. Dostupné z: https://aws.amazon.com/about-aws/

AMAZON WEB SERVICES, 2020b. Amazon CloudFront Documentation. *Amazon CloudFront Documentation* [online] [vid. 2020-04-13]. Dostupné z: https://docs.aws.amazon.com/cloudfront/index.html

AMAZON WEB SERVICES, 2020c. Amazon CloudWatch - Application and Infrastructure Monitoring. *Amazon Web Services, Inc.* [online] [vid. 2020-04-18]. Dostupné z: https://aws.amazon.com/cloudwatch/

AMAZON WEB SERVICES, 2020d. Amazon Simple Storage Service Documentation. *Amazon Simple Storage Service Documentation* [online] [vid. 2020-04-13]. Dostupné z: https://docs.aws.amazon.com/s3/index.html

AMAZON WEB SERVICES, 2020e. Cloud Object Storage | Store & Retrieve Data Anywhere | Amazon Simple Storage Service (S3). *Amazon Web Services, Inc.* [online] [vid. 2020-03-24]. Dostupné z: https://aws.amazon.com/s3/

AMAZON WEB SERVICES, 2020f. Cloud Products. *Amazon Web Services, Inc.* [online] [vid. 2020-03-23]. Dostupné z: https://aws.amazon.com/products/

AMAZON WEB SERVICES, 2020g. Content Delivery Network (CDN) | Low Latency, High Transfer Speeds, Video Streaming | Amazon CloudFront. *Amazon Web Services, Inc.* [online] [vid. 2020-03-24]. Dostupné z: https://aws.amazon.com/cloudfront/

AMAZON WEB SERVICES, 2020h. What is Cloud Computing. *Amazon Web Services, Inc.* [online] [vid. 2020-03-23]. Dostupné z: https://aws.amazon.com/what-is-cloud-computing/

AMAZON WEB SERVICES, 2020i. What is DevOps? - Amazon Web Services (AWS). *Amazon Web Services, Inc.* [online] [vid. 2020-01-18]. Dostupné z: https://aws.amazon.com/devops/what-is-devops/

ANGULAR, 2020. *Angular - Deployment* [online] [vid. 2020-04-26]. Dostupné z: https://angular.io/guide/deployment#routed-apps-must-fallback-to-indexhtml

ATLASSIAN, 2020a. Bitbucket Cloud documentation - Atlassian Documentation. *Bitbucket Cloud documentation* [online] [vid. 2020-03-22]. Dostupné z: https://confluence.atlassian.com/bitbucket

ATLASSIAN, 2020b. What is DevOps? *Atlassian* [online] [vid. 2020-01-18]. Dostupné z: https://www.atlassian.com/devops

BASS, Len, Ingo WEBER a Liming ZHU, 2015. *DevOps: a software architect's perspective*. New York: Addison-Wesley Professional. The SEI series in software engineering. ISBN 978-0-13-404984-7.

BITBUCKET, 2020. Bitbucket Overview. *Bitbucket* [online] [vid. 2020-03-08]. Dostupné z: https://bitbucket.org/product/guides/getting-started/overview

BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ, 2016. *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu*. ISBN 978-80-247-5594-6.

CYPRESS, 2020. End to End Testing Framework. *JavaScript End to End Testing Framework | cypress.io* [online] [vid. 2020-03-28]. Dostupné z: https://cypress.io/how-it-works

DAVIS, Jennifer a Katherine DANIELS, 2016. *Effective devOps: building a culture of collaboration, affinity, and tooling at scale*. First edition. Beijing ; Boston: O'Reilly. ISBN 978-1-4919-2630-7.

DORA, 2019. *2019 Accelerate State of DevOps Report* [online]. 2019. [vid. 2020-04-05]. Dostupné z: https://services.google.com/fh/files/misc/state-of-devops-2019.pdf

FIELDING, Roy Thomas, 2000. *Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)* [online] [vid. 2020-03-24]. Dostupné z: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

FÓL, Pavel, 2018. *Metodika Continuous Integration při vývoji webové aplikace*. B.m. Diplomová práce. Univerzita Pardubice. Fakulta elektrotechniky a informatiky.

GUNARATHNA, Thilina, 2018. Test result report configuration for Protractor test suite. *Medium* [online] [vid. 2020-03-16]. Dostupné z: https://medium.com/@gtgunarathna/test-result-report-configuration-for-protractor-test-suite-c36f58b7b616

HOFFMANN, Patrik, 2018. *Vužití nástroje Protraktor pro automatizované testování webových aplikací*. Praha. Diplomová práce. Vysoká škola ekonomická v Praze. Fakulta infomatiky a statistiky.

HUMBLE, Jez a David FARLEY, 2010. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Upper Saddle River, NJ: Addison-Wesley. ISBN 978-0-321-60191-9.

JEST, 2020. *Jest · Delightful JavaScript Testing* [online] [vid. 2020-03-28]. Dostupné z: https://jestjs.io/

KATHRYNEE, 2018. *Understand and configure your Kanban board - Azure Boards* [online] [vid. 2020-03-26]. Dostupné z: https://docs.microsoft.com/en-us/azure/devops/boards/boards/kanban-basics

KIM, Gene, Jez HUMBLE, Patrick DEBOIS a John WILLIS, 2016. *The DevOps handbook: how to create world-class agility, reliability, & security in technology organizations*. First edition. Portland, OR: IT Revolution Press, LLC. ISBN 978-1-942788-00-3.

LAPOTKA, Dzianis, 2018. *Analýza a návrh možností automatizace releasů aplikací ve společnosti s ITIL prostředím*. B.m. Diplomová práce. Vysoká škola ekonomická v Praze. Fakulta infomatiky a statistiky.

MAGNI, Stefano, 2020. Some UI testing problems and the Cypress way. *Medium* [online] [vid. 2020-03-16]. Dostupné z: https://itnext.io/some-ui-testing-problems-and-the-cypress-way-22312ab986e3

MACHÁČ, Martin, 2018. *Průběžná integrace a průběžná dodávka/nasazení projektu KYPO*. B.m. Diplomová práce. Masarykova univerzita. Fakulta informatiky.

MICROSOFT, 2020. *JavaScript For Any Scale.* [online] [vid. 2020-03-28]. Dostupné z: https://www.typescriptlang.org/v2/

NARWHAL TECHNOLOGIES INC., 2020. *Nx CLI* [online] [vid. 2020-03-28]. Dostupné z: https://nrwl.dev/web/guides/cli

NODE.JS, 2020. About Node.js. *Node.js* [online] [vid. 2020-03-28]. Dostupné z: https://nodejs.org/en/about/

PEFFERS, Ken, Tuure TUUNANEN, Marcus A. ROTHENBERGER a Samir CHATTERJEE, 2008. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*. s. 45–77.

REHKOPF, Max, 2020. Kanban vs Scrum. *Atlassian* [online] [vid. 2020-03-26]. Dostupné z: https://www.atlassian.com/agile/kanban/kanban-vs-scrum

ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ, 2013. *Řízení kvality softwaru: průvodce testováním*. 1. vydání. Brno: Computer Press. ISBN 978-80-251-3816-8.

SATO, Danilo, 2014. CanaryRelease. *martinfowler.com* [online] [vid. 2020-05-02]. Dostupné z: https://martinfowler.com/bliki/CanaryRelease.html

SLACK, 2020. Features. *Slack* [online] [vid. 2020-04-18]. Dostupné z: https://slack.com/features

SOMMERVILLE, Ian, 2013. *Softwarové inženýrství*. Brno: Computer Press. ISBN 978-80-251-3826-7.

ŠTRAJT, Vítězslav, 2020. *Průběžné doručování produktu do cloudové platformy Salesforce*. B.m. Diplomová práce. Vysoká škola ekonomická v Praze. Fakulta infomatiky a statistiky.

WATTS, Stephen, 2017. *Continuous Delivery vs Deployment vs Integration: What's the Difference? – BMC Blogs* [online] [vid. 2020-01-18]. Dostupné z: https://www.bmc.com/blogs/continuous-delivery-continuous-deployment-continuous-integration-whats-difference/