Mendelova univerzita v Brně
Provozně ekonomická fakulta

Rozšíření nástroje Jenkins CI o funkci změny pořadí ve frontě úloh

Bakalářská práce

Vedoucí práce:
Ing. Petr Jedlička, Ph.D.

Jaroslav Otradovec

Brno 2020

Poděkování

Na tomto místě bych rád poděkoval vedoucímu své práce, Ing. Petru Jedličkovi, Ph.D. za vstřícný přístup, stejně jako panu magistru Jiřímu Vaňkovi za téma práce i cenné informace. Dík patří i mé rodině za podporu během celého studia.

Čestné prohlášení

Prohlašuji, že jsem práci **Rozšíření nástroje Jenkins CI o funkci změny pořadí ve frontě úloh** vypracoval samostatně a veškeré použité prameny a informace uvádím v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a v souladu s platnou Směrnicí o zveřejňování závěrečných prací.

Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Kutné Hoře dne 20. 5. 2020 .....

podpis

Abstract

Otradovec, J. Extension of Jenkins CI with job queue order change feature. Bachelor thesis. Brno: Mendel University in Brno, 2020.

The bachelor thesis has as its goal extension of features of Jenkins, a web server for continuous integration, by enabling changing order of items waiting for execution in the queue. Plugin that modifies user interface of authorised user was created, thus allowing sooner execution of preferred item waiting in the queue. This is achieved with a change of the items' order, which current plugins allow only in some specific cases (prioritizing jobs that are shorter, historically less successful, raised by external service etc.) or through a configuration change. Analysis of other solutions concerning this problem, requirement analysis and plugin draft constitute for this thesis a foundation upon which was implemented solution mainly in Java language, tested with the JUnit framework, accepted to Jenkins' official plugin listing and released in the Jenkins Plugin Manager.

Key words: Jenkins, Java, Continuous Integration, Continuous Delivery, Continuous Deployment.

Abstrakt

Otradovec, J. Rozšíření nástroje Jenkins CI o funkci změny pořadí ve frontě úloh. Bakalářská práce. Brno: Mendelova univerzita v Brně, 2020.

Cílem této bakalářské práce je rozšíření možností webového serveru pro podporu průběžné integrace – Jenkins – o úpravy fronty úloh čekajících na vykonání. Byl vytvořen zásuvný modul, který modifikuje uživatelské rozhraní u oprávněného uživatele, který může upravit pořadí úloh, a díky tomu urychlit provedení této naléhavější úlohy, což současná řešení dovolují pouze v určitých specifických případech (přednost úloh kratších, historicky méně úspěšných, vyvolaných externí službou apod.) nebo skrze změnu konfigurace. Výchozím bodem práce je analýza konkurenčních řešení této problematiky, analýza požadavků a návrh zásuvného modulu. Výsledné řešení bylo implementováno především v jazyce Java, otestováno za použití JUnit, zařazeno do oficiálního seznamu pluginů a zveřejněno v Jenkins Plugin Manageru.

Klíčová slova: Jenkins, Java, průběžná integrace, průběžné dodání, průběžné nasazení.

Obsah

1 Úvod

Když v roce 1933 Reginald Gibson a Eric Fawcett objevili polyethylen (1), jen těžko mohl někdo předvídat, jaká úskalí budou s masovým používáním plastů spojena. V 21. století je možno plastový odpad najít od vrcholků nejvyšších hor na světě po dna oceánů, takže je možno potkat vznášející se plastové tašky i v hloubce 1 000 metrů pod mořskou hladinou. (2)

Obdobně je tomu i u tvorby softwaru. Jen těžko si mohl někdo představit ještě v relativně nedávné době, kdy počítače byly pouze v majetku několika finančně dobře postavených organizací, jaké nepříjemnosti na vývojové týmy čekají. Možná si je neuvědomoval ani Kohsuke Kawaguchi, když nastupoval do firmy Sun Microsystems, ale jisté je, že v roce 2004 se rozhodl na tom něco změnit. Aby předešel hádkám s kolegy, kteří mu vyčítali chyby v kódu, rozhodl se vytvořit předka dnešního Jenksins CI – Hudson CI. Jelikož i ostatní kolegové pochopili přínos jeho projektu, žádali ho o zdrojový kód, který se rozhodl šířit jako open-source. (20)

Zásadním prvkem vývoje software je testování, které poskytuje zpětnou vazbu o kvalitě proběhlé práce. S růstem výpočetní náročnosti testů u větších aplikací však zpravidla dochází k nežádoucímu snížení frekvence testování, jelikož provádění testů omezuje programátora v další souběžné práci. Zároveň v některých dalších případech testování na zařízení programátora nemusí být vhodné nebo možné (závislost na operačním systému, performance testy a pod.), a tak dochází k vyčlenění výpočetní techniky a nebo výpočetního výkonu pro automatické spouštění testů. Jednou z možností, jak testy spouštět, je použití nástroje Jenkins.

Úskalím tohoto přístupu je ovšem omezenost zdrojů, které firma pro tyto účely může vynaložit. Pokud jsou zdroje pro testování fixní, avšak intenzita vývoje je v čase proměnlivá, může docházet k hromadění testovacích požadavků. Jenkins řadí tyto požadavky do fronty, kterou v základním nastavení zpracovává způsobem FIFO (pomineme-li možnou rozdílnost výpočetních uzlů), což vždy nemusí odpovídat potřebám jeho uživatele.

2 Cíl práce

Cílem této práce je rozšíření možností nástroje Jenkins CI o změnu pořadí jednotlivých úloh ve frontě oprávněným uživatelem včetně příslušné úpravy uživatelského rozhraní. Jedním z kroků, které k tomuto povedou, bude analýza fronty sestavených úloh, jejího základního řazení a stávajících možností úpravy fronty skrze pluginy. Dále bude vypracována analýza konkurenčních služeb k Jenkins s ohledem na řešení tohoto problému. Následně bude vypracován návrh pluginu, který bude implementován a otestován.

3 Literární rešerše

3.1 Projekt Jenkins

Jenkins vznikl v roce 2004, ještě pod jménem Hudson, jako projekt Kohsuke Kawaguchiho, který již déle nechtěl snášet hádky s kolegy z firmy Sun Microsystems. Byl totiž nechvalně proslulý jako "ten, co to vždy rozbije"(19). Aby předešel dalším hádkám, rozhodl se závest, z hlediska dnešní terminologie, průběžnou integraci. Při následném rozšíření používání tohoto projektu se ukázala možnost automatizace dodání a nasazení, čímž se významně rozšířila funkcionalita a místo Jenkins CI se již většinově hovoří pouze o Jenkins. V únoru 2018 se uváděl počet aktivních instalací na 165 000. (20)

V dubnu 2020 dosáhl počet výpočetních uzlů 1 026 225, přičemž většina byla spuštěna na OS Linux, 26% uzlů neposkytlo informaci o OS. Méně než 10% uzlů používalo OS Windows (především Windows 10) stejně jako Windows Server (především Windows Server 2012 a 2016). (21)

Proč došlo ke změně z Hudson na Jenkins? Jelikož Hudson byl spjat s firmou Sun, kterou v roce 2009 koupila společnost Oracle, brzy začaly růst spory mezi vedením Oracle a komunitou vývojářů. V roce 2011 se situace vyhrotila natolik, že došlo k rozdělení na Hudson, který spravuje společnost Oracle, a Jenkins, kterého se ujala část komunity. (19)

Původní projekt Hudson je v závěrečné fázi existence. 12. 12. 2012 byla zveřejněna verze 3.0.0, která se dočkala pouze vývoje do verze 3.3.0 v roce 2015. Od toho momentu byly vydávány pouze úpravy charakteru bezpečnostních záplat. Dokonce byl oznámen konec aktualizací a oficiální webové stránky k lednu roku 2020. (26)

Zakladatel projektu, Kohsuke Kawaguchi, v současnosti pracuje pro firmu Cloudbees jako poradce, do roku 2020 pracoval jako technický ředitel. Cloudbees se podílí na tvorbě inovativních částí Jenkins jako Jenkins 2, Blue Ocean (sada cca 22 pluginů pro intuitivní uživatelská rozhraní) či Jenkins X (zkráceně JX, spojení Jenkins, Dockeru a Kubernetes). V lednu roku 2020 pan Kawaguchi oznámil založení firmy Launchable, která by se měla věnovat analýze dat a použití AI ve spojení s Jenkins. (20; 22)

Jenkins sám se snaží dodržovat koncepty průběžné integrace, dodání a nasazení. Z toho důvodu vydává vlastní novou verzi každý týden pro vývojáře jádra i pluginů či uživatele, kteří potřebují rychle opravu. Pro většinu uživatelů je vhodnější sestavení s dlouhodobou podporou, která jsou hlasováním volena každých 12 týdnů a následně doplňována nejdůležitějšími záplatami z hlavní vývojové větve (29). Další ukázkou uplatnění průběžné integrace je okamžité provedení pokusu o sestavení po každé změně repozitáře, jak se to děje u většiny částí jádra i pluginů. (30)

Zmíněná hlasování, která určují LTS verzi, členy jednotlivých orgánů, koncepci vývoje apod. probíhají každé dva týdny na online schůzích. Samotný vlastník projektu a všech práv, včetně ochranné známky, je společnost Software in the Public Interest sídlící v New Yorku. Celé jádro a velká část pluginů je poskytována pod licencí MIT, avšak pro větší právní jistotu každý přispěvatel jádra musí nejprve potvrdit dohodu, ve které se zříká všech práv ke svým příspěvkům (ICLA). Pokud přispěvatel tuto činnost vykonává v rámci pracovněprávního vztahu, je třeba učinit i druhou dohodu (CCLA). Každý plugin musí obsahovat nějakou standardní open-source licenci, jinak je jeho zveřejnění chápáno pod licencí MIT; pluginy s proprietární licencí jsou vyloučeny ze zveřejnění na oficiálním seznamu. (31)

Jelikož stejně jako počet pluginů časem rostla i velikost jádra, byly pro některé

části jádra vytvořeny moduly. Tyto moduly jsou sestavovány, podobně jako pluginy, po každé změně ve zdrojovém kódu modulu, avšak na rozdíl od nich se stávají součástí výsledného WAR souboru jádra, který se z hlediska vývojáře liší od pravého jádra spíše pouze umístěním v WEB-INF/lib. (31)

3.2 Continuous Integration & Delivery & Deployment

Jelikož pojmy průběžná integrace (Continuous Integration), průběžné dodání (Continuous Delivery) a průběžné nasazení (Continuous Deployment) se týkají jádra problému, které Jenkins řeší, budou na následujících řádcích vysvětleny.

Pod pojmem **průběžná integrace** se rozumí především automatické sestavení a spuštění testů. Ačkoliv běžně každý vývojář může spouštět jednotkové testy před odevzdáním své práce, integrační testy vzhledem ke své náročnosti toto zpravidla neumožňují. Tento problém má za následek buď kód rozložený v různých větvích, které však časem spolu mohou začít být nekompatibilní, nebo sice jednotný kód v hlavní větvi, který ovšem postrádá integrační testy. Z toho důvodu se problému předchází spouštěním integračních testů na samostatném serveru ideálně po každé změně ve zdrojovém kódu na repozitáři. Pokud aplikace po změnách neprojde testy, může o tom být vývojář okamžitě informován. (25; 27; 28)

Přínosy průběžné integrace plynou z redukce rizik, která vznikají při každém užití předpokladu, např. když vývojář předpokládá, že jeho změna zásadně nezhoršuje výkon aplikace. (28) Jedná se o přínosy:

- Ekonomické – snaha o maximální automatizaci šetří čas pracovníků
- Odpovědnostní – neprojití testu jasně ukazuje, kdo je za chybu zodpovědný
- Psychologické – vývojový team si je jistější svým produktem a zároveň má větší odvahu ke změnám
- Prezentační – v každé fázi vývoje je produkt připraven pro prezentaci zákazníkovi
- Pohotovostní – software je stále připravený pro nasazení u zákazníka

Průběžné dodání navazuje na průběžnou integraci v tom smyslu, že předpokládá existenci otestovaného produktu. Pokud bylo testům vyhověno, dochází k automatickému dodání či zveřejnění produktu. Je-li zapojeno průběžné dodávání, je možno přistoupit k **průběžnému nasazení** dodaného produktu na zařízení zákazníka. Výsledkem je tedy zkrácení doby mezi provedením změny ve zdrojovém kódu a nasazením v produkci. Dalším přínosem je zvýšení efektivnosti práce, protože problémy se zjistí dříve a částečně se jim i předchází. (27)

Cenou za výše zmíněné přínosy je nutnost psaní automatických testů, skriptů pro dodání a nasazení, pořízení a údržba automatizačního serveru a zavedení praxe častých sloučení s hlavní větví mezi vývojáři. Dále nelze pominout ani bezpečnostní riziko. (27)

3.3 Současný stav

Úvod

Dříve než bude blíže představen současný stav fronty v Jenkins, je na místě představit

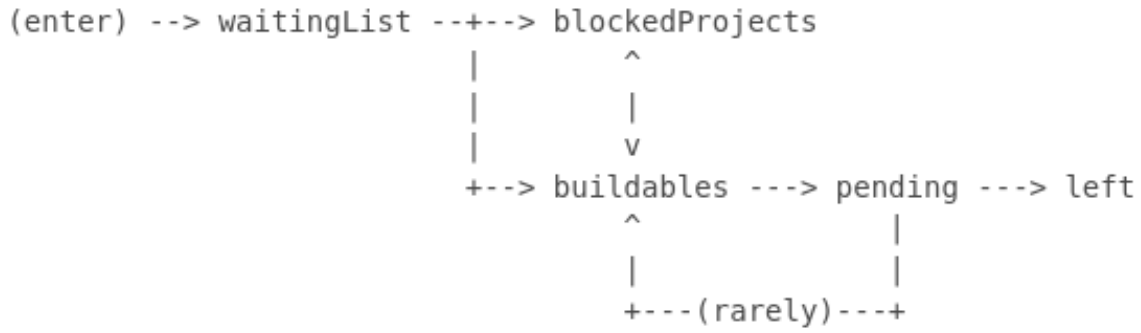
důvod existence fronty a její obsah. Základním omezením každého způsobu provádění průběžné integrace na vlastních strojích je omezený rozsah výpočetního výkonu v daném čase. Počet výpočetních uzlů v některých situacích nemusí odpovídat počtu aktuálně požadovaných úloh. Dokonce některá zařízení mohou být naprosto nevhodná pro některé úlohy, např. různé OS, procesorové architektury apod. Pro rozložení provádění úloh v čase byla vytvořena fronta, takže pro vykonání stejného množství práce stačí firmě disponovat relativně menším výpočetním výkonem. Samotný prvek fronty je instance třídy Queue.Item, který je nerozlučně spojen se svou mateřskou úlohou (Task). Jelikož z hlediska fronty může v danou chvíli existovat pouze jeden Queue.Item na každý Task, dají se často tyto pojmy částečně zaměňovat. Základním pravidlem tedy je, že žádný prvek fronty nemá původ ve stejné úloze. (35; 36)

Základní fronta v Jenkins

Fungování základní fronty v Jenkins, tedy fungování fronty neovlivněné dodatečnou instalací zásuvných modulů, je řízeno třídami částečně přejatými ještě z projektu Hudson a z velké části napsané ještě zakladatelem, panem Kawaguchi (např. třída QueueTaskDispatcher). Jedná se především o třídy z balíčku hudson.model.queue, konkrétně AbstractQueueSorterImpl, který implementuje metodu compare, dále je zde skupina tříd poskytující odhad zátěže prvku fronty – FutureImpl, FutureLoad, LoadPredictor a LoadPredictor. Významná z hlediska této práce je i synchronně pracující třída QueueListener (možno místo ní použít asynchronně pracující třídu Executor) a třída QueueSorter, která tvoří seznam sestavitelných a blokovaných úloh. Jestli je daná úloha blokována nebo může být spuštěna, sděluje třída QueueTaskDispatcher. Více funkcionalit poskytuje třída QueueTaskFilter, která jednak sděluje informace o úloze obdržené v konstruktoru (jméno úlohy, jméno úlohy s HTML tagy, URL, důvod blokace, odhadovaná doba provedení úlohy, jednotlivé podúlohy) a dále umožňuje kontrolu práv uživatele pro přerušení běhu dané úlohy (32). Tyto metody jsou většinou implementací rozhraní Queue.Task. Toto rozhraní Queue.Task je dále implementováno v třídách projektů jako je AbstractProject, Project a FreeStyleProject (33). FreeStyleProject je výchozí projekt, v některých instalacích je i projektem jediným. Skrze třídu Project je pak FreeStyleProject rozšířením AbstraktProjectu. (34)

Klíčovou třídou fronty je Queue. V této poměrně rozsáhlé části projektu (třída má 3139 řádků) se odehrává jádro operací nad frontou. Fronta je v několika vrstvách rozdělena podle stavů, které prvky mohou nabývat, jak zobrazuje schéma stavů. (35; 36)

Hlavní části fronty jsou waiting, blocked, buildables a pending. V některých případech se pod pojem buildables zahrnují i prvky ve stavu pending. Jednotlivé části fronty jsou uloženy v oddělených datových strukturách, což je v případě waiting TreeSet a u ostatních se jedná o ItemList. Jejich pořadí od nejvzdálenějších k nejbližším k provedení je waiting – blocked – buildables – pending, jak je skládá metoda ”getItems()”. Jaký je smysl jednotlivých stavů? Waiting působí jako ochrana fronty před smrští nově příchozích úloh, ke které dochází při větším propojení systémů či úloh. Každá úloha počká jistý čas ve stavu waiting, než je poslána dále do blocked či buildables. Pokud nic nebrání provedení úlohy, je přesunuta do buildables, v opačném případě do blocked. Jako typický důvod pro blokování prvku se dá uvést právě běžící sestavení na výpočetním uzlu, které se odkazuje ke stejné úloze (Task). Smysl stavu pending spočívá v jisté cílové rovince, kdy je pro prvek vyhlédnuto odpovídající volné zařízení na vykonání. Pokud dané zařízení kvůli nějaké



Obrázek 1: Schéma stavů prvků fronty, zdroj (33)

technické komplikaci (výpadek el. proudu, internetového připojení) nezapočalo přislíbené vykonávání, je prvek z pending vrácen zpět do buildables. Typicky poslední stav, v jakém je možno prvek ve frontě nalézt, je left, ve kterém po něm zůstane stopa 5 minut. Přechody mezi zmíněnými stavy zajišťuje metoda maintain. (35; 36)

Jednotlivé části fronty jsou přístupné přes tzv. gettery, které přistupují k poslednímu snímku stavu fronty a nikoliv k frontě samotné. To má za následek možnou krátkodobou diskrepanci mezi tím, jak se fronta jeví a jaká je ve skutečnosti. Tento rozdíl má pouze krátké trvání, neboť každých 5 000 ms je provolána privátní metoda updateSnapshot metodou maintain, jejíž součástí je i vytvoření nového snímku fronty. K této aktualizaci by mělo docházet při změně fronty, jako je načtení fronty, vymazání fronty, zrušení prvku či jeho vložení nebo provedení. (35; 36)

Jaký prvek z buildables bude přesunut do pending, závisí na jeho pořadí ve frontě, protože metoda maintain postupně zkouší najít Exekutora, který by prvek přijal. Není-li nainstalován žádný plugin upravující řazení, pak pořadí prvků závisí na čase příchodu do fronty, kdy první příchozí prvek se stává prvním, který frontu opouští – uplatněn je tedy systém FIFO. Stejně tak blocked si udržují pořadí na podobném principu. (36; 37)

Řazení buildables je metodou maintain umístěno za jejich přidání z čekajících prvků a před jejich odebrání do pending. Způsob řazení včetně řadicího algoritmu je závislý na daném pluginu, který rozšiřuje třídu QueueSorter. Pokud žádný takový neexistuje, je použita výchozí implementace, která nemá vlastní řadicí algoritmus, ale aplikuje řazení z Java.util.List. (36; 38)

Zobrazení fronty

Widget fronty má v uživatelském rozhraní stanovenou pozici pod seznamem akcí a nad listem výpočetních jednotek. Z hlediska uživatele nabývá fronta dvou stavů – základního a filtrovaného. Základní stav je ten, který se uživateli zobrazí v základním pohledu All nebo případně jakémkoliv jiném nefiltrovaném pohledu. Filtrovaně se fronta zobrazí pouze po přechodu na pohled, který má v konfiguraci nastavené filtrovaní fronty. Samotná třída Queue vůbec nemá ponětí o filtraci a váha této problematiky spočívá na pohledech a některých částech uživatelského rozhraní. Konkrétně na jelly tagu queue, na který odkazuje groovy index Build Queue Widget a ajaxBuildQueue.jelly pohledu. (39; 40; 41; 42)

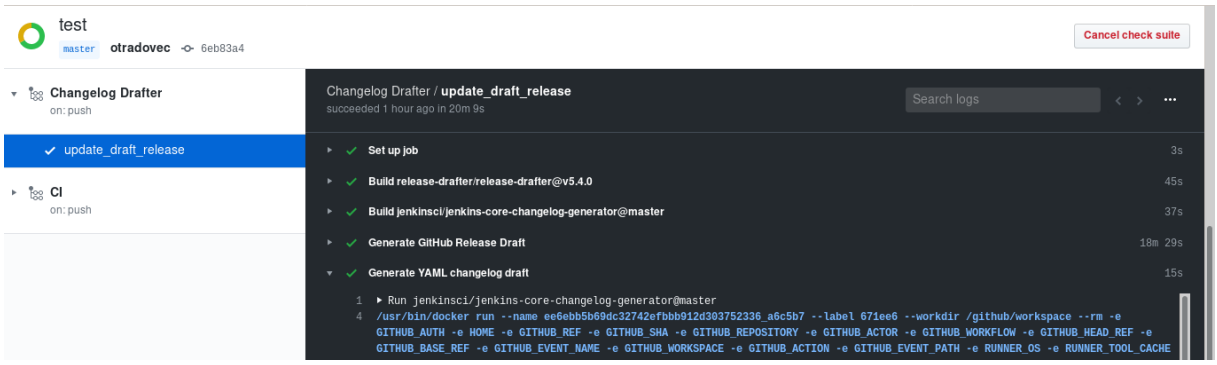
Jak celý proces funguje? Na počátku všeho je registrace rozšíření typu BuildQueueWidget, následně při vstupu uživatele na URL adresu úvodní obrazovky se hledá odpovídající soubor index.jelly v adresáři odpovídajícím cestě src/main/resources/jenkins/widgets/BuildQueueWidget/, který ovšem nalezen není, a tak probíhá již úspěšnější hledání index.groovy. V něm se vloží fronta tagem queue ze sdílené knihovny JenkinsTagLib a jako parametr se předá úvodní pohled, informace o filtraci a obsah fronty podle pohledu. Součástí tagu "queue" je i asynchronní aktualizace pravidelným dotazováním na ajaxBuildQueue, čímž je zajištěno promítnutí změn na frontě v uživatelském rozhraní, aniž by musela být načtena znovu celá obrazovka. Pokud dojde ke změně pohledu, celá stránka se znovu načte, znovu je použit index.jelly, který sestaví frontu na základě dat z druhého pohledu. Následná aktualizace probíhá dotazováním na ajaxBuildQueue příslušného pohledu. (39; 40; 41; 42)

Nedílnou součástí přístupu uživatele ke frontě je kontrola autentizace a autorizace. Nejvyšší oprávnění (Jenkins.ADMINISTER) je třeba pro vymazání celé fronty. Pro zrušení prvku fronty stačí oprávnění k přerušení u jeho mateřské úlohy, bez kterého se ani křížek popř. odkaz na zrušení u daného prvku nezobrazí. Pro získání obsahu fronty i její zobrazení je nutno povolení ke čtení. Pro připojení automatické aktualizace je třeba globální právo na čtení (Jenkins.READ). (35; 40)

3.4 Existující řešení

Konkurence Jenkins – GitHub Actions

GitHub Actions jsou službou pro průběžnou integraci, dodání a nasazení navázanou na repozitáře GitHub. Oproti Jenkins tedy Actions požívají vysoké podpory od společnosti GitHub. Základní stavební jednotkou jsou akce, které dohromady vytváří workflow a jsou uloženy v textovém souboru syntaxí YAML. Nabízené prostředí běhu workflow je možno vybrat mezi Linuxem, masOS a Windows. Počet paralelně běžících workflow je maximálně 20 na jeden repozitář a doba běhu workflow je omezena na 6 hodin. Zajímavá je oblast zpoplatnění, kde je k dispozici pro veřejné repozitáře použití Actions zdarma. Placené jsou Actions pouze na soukromých repozitářích podle výpočetního času a uložených dat. (43)



Obrázek 2: GitHub Actions – výsledek běhu workflow

Jednotlivé akce tvoří kroky úlohy, úloha se skládá ve workflow. Akce může uživatel vytvářet sám anebo si může vybrat nějakou akci vytvořenou komunitou na tržišti. Akce může být definována na třech různých místech:

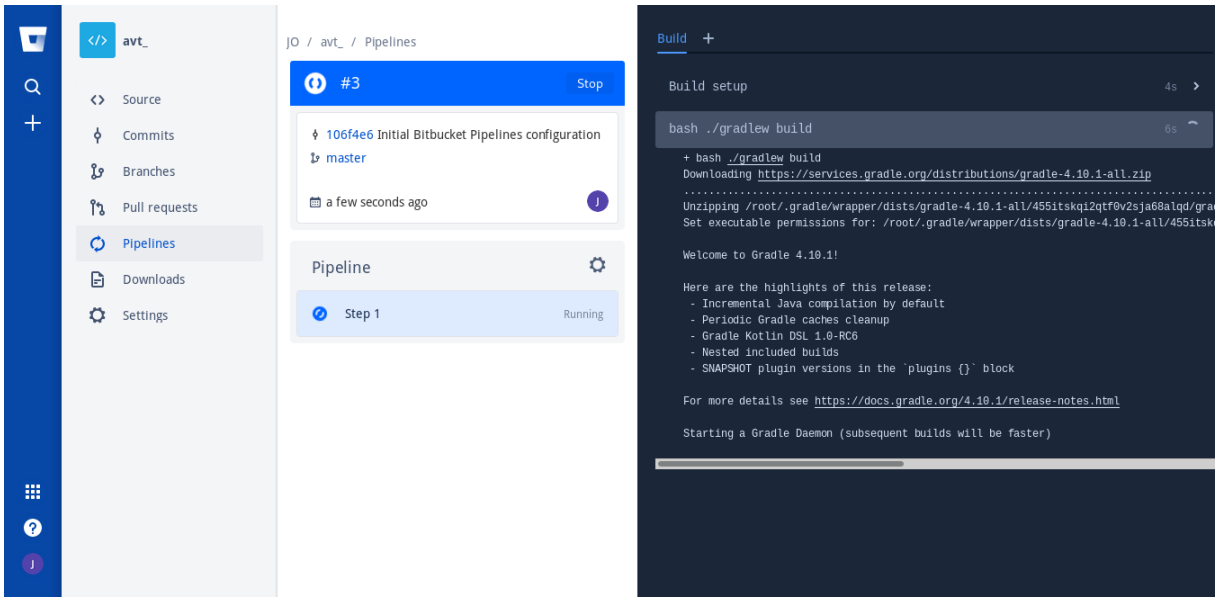
- Na veřejném repozitáři.
- Ve stejném repozitáři jako je workflow.
- Na Docker Hub jako kontejner.

Ke spuštění workflow může dojít obdobně jako u Jenkins událostí na repozitáři, naplánováním na konkrétní čas či vnější okolností. Stejně jako Jenkins se pro zápis časování používá zápis cron. (44)

Konkurence Jenkins – BitBucket Pipelines

BitBucket Pipelines je, podobně jako GitHub Actions, obdobou cloudových řešení Jenkins. Využívá propojení na další služby firmy Atlassian jako je sada pro projektové řízení – JIRA či služba pro tvorbu webových stránek – Confluence.

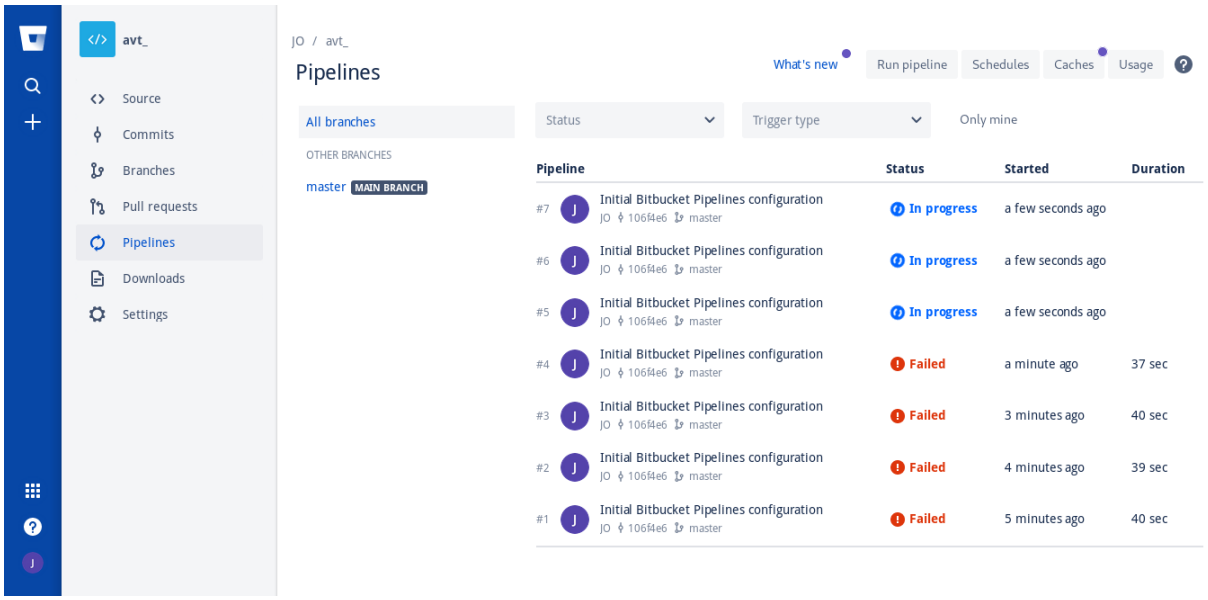
Výraznou slabinou Pipelines je relativně malá konfigurovatelnost času spuštění sestavení oproti Jenkins. Spuštění Pipelines v závislosti na čase je možné vytvořením rozvrhu. Rozvrh však nabízí pouze tři možnosti časování – každý týden, den a hodinu. Tento nedostatek je možno v některých případech vyřešit vytvořením více rozvrhů pro jednu Pipeline.



Obrázek 3: Ukázka detailu spuštěného procesu v Pipelines

Sami uživatelé Bitbucket Pipelines přiznávají omezení, která se nenachází u Jenkins, jako jsou široké možnosti notifikací. Kritizují Jenkins jako náročnější na údržbu a varují před nutností získání výpočetního výkonu, který podle jejich názoru bude dražší než platba za čas v Bitbucket Pipelines (45). Vzhledem k velké flexibilitě volby zdroje výpočetního výkonu u Jenkins je pravděpodobné, že lze dosáhnout nižších provozních nákladů, nepočítaje čas pracovníka při správě systému. Jistou přednost Bitbucket Pipelines lze spatřovat v příjemném grafickém uživatelském rozhraní – takové sice není v základní instalaci Jenkins, avšak může ho být dosaženo skrze balík zásuvných modulů Blue Ocean.

Problematika fronty úloh u služby Pipelines de facto neexistuje, jelikož je spouštěna vždy jedna ”úloha”, která se skládá z maximálně 100 dílčích úloh – kroků. Jediné, co



Obrázek 4: Ukázka souhrnu procesů v Pipelines

má uživatel možnost ovlivnit, je, zda budou kroky vykonávány sekvenčně či paralelně, přičemž se dají navzájem kombinovat. (46)

Pluginy Jenkins s podobnou problematikou

Obecně pluginy můžeme rozdělit do následujících pěti skupin:

1. Platformně specifické pluginy (iOS, Android, .NET)
2. Pluginy uživatelského rozhraní (Folders, Simple Theme, jQuery UI)
3. Správa Jenkins (Audit Trail, Multi slave config, Docker)
4. Správa zdrojového kódu (Git, Git Changelog, Bitbucket Branch Source)
5. Správa sestavení (JUnit, Gradle, Ant)

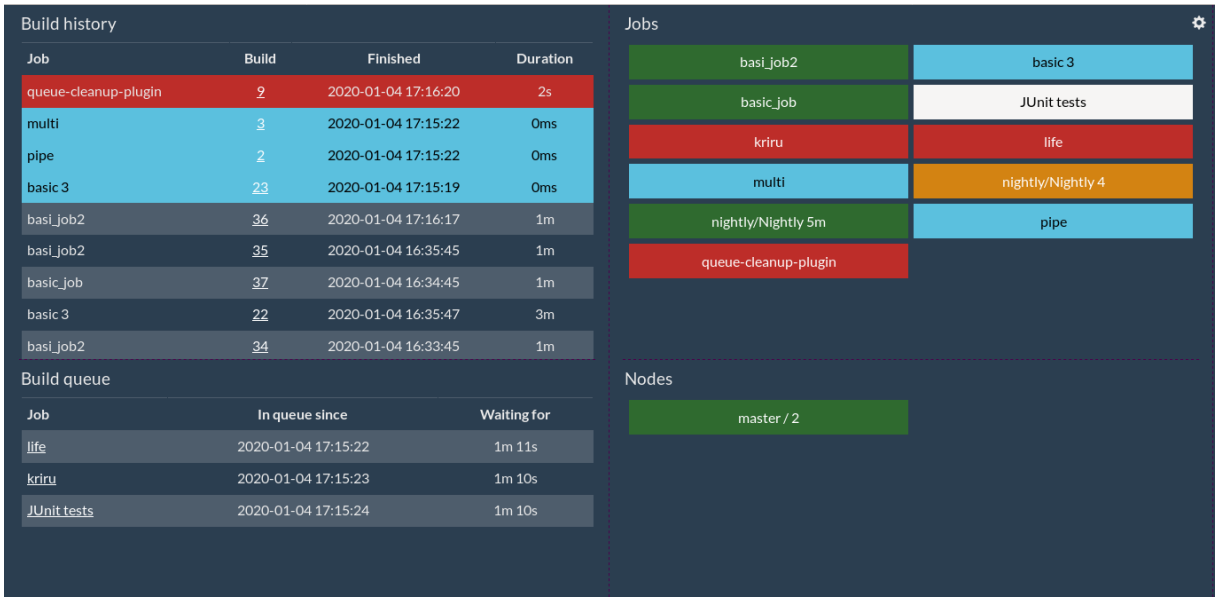
Celkový počet pluginů překračuje 1 500, k 1. 1. 2020 je 1 910 pluginů (47; 48). Vzhledem k tomuto vysokému počtu byly zkoumány pouze pluginy, které obsahují v názvu nebo popisu slovo ”queue”. K 1. 1. 2020 pro verzi Jenkins 2.190.3 takových bylo 24. Z toho část slouží pro napojení služeb třetích stran, a proto nebyla dále brána v potaz. Dále nebude popisován Computer-queue-plugin, který slouží pro zobrazení individuálního uzlu. Nebude rozebírán ani plugin Requeue Job, ačkoliv přidává úlohy do fronty, avšak pouze v případě, že nedošlo k jejich vykonání z důvodu výpadku výpočetního uzlu. (71)

Plugin se jménem No Agent Job Purge slouží pouze ke zrušení úloh, které neběží z důvodu, že uzel není online nebo uzly sice online jsou, ale nejsou určeny pro zpracování úlohy z daným labelem neboli označením. Tento plugin tedy neumí nic více než automaticky rušit, a proto není vhodný pro řešení problému z pořadím úloh ve frontě (49). Podobně znějící Purge Build Queue plugin slouží k vyčištění celé fronty. Místo zdlouhavého individuálního rušení úloh je možné dvěma kliknutími zrušit všechny najednou. O nemalé oblibě tohoto řešení svědčí kolem 3 300 aktivních instalací. Přesto je

třeba podotknout, že užití stejné ikony jako pro nastavení, které se vyskytuje několik málo řádků výše, může vést ke zmatením uživatele (50). Funkcionalita Persistent Build Queue pluginu byla již zahrnuta do jádra Jenkins a netýká se přímo řazení fronty, ale zabránění ztrátě fronty při restartu Jenkins (51). Queue cleanup plugin rovněž pouze odebírá prvky z fronty, konkrétně ty, které odpovídají zadanému regulárnímu výrazu a čekají ve frontě delší dobu. V základním nastavení se jedná o jednu hodinu, ale je možno nastavit i jiné kladné celé číslo. V době psaní této práce je odeslaný požadavek na změnu na reálné číslo, který nebyl správcem vyhodnocen. (52)

Ještě hůře je na tom Read-only configurations plugin, v době psaní této práce již rok a půl správce neodpověděl na požadavek přechodu na vyšší verzi Jenkins. Zčásti plugin jistě funguje i ve verzi Jenkins 2.190, když zamezuje změně globální konfigurace. Podle dokumentace by ovšem měl zamezit i změně v nastavení úlohy, což neplatí. Co se týká fronty, plugin tedy dokáže pouze zabránit změně způsobu řazení (např. v kombinaci s Priority Sorter), avšak sám řazení neovlivňuje (53). Obdobně pracuje Job/Queue/Slaves Monitoring plugin, který neumožňuje práci s frontou, ale vyzdvihnout si zaslouží jeho vysoká konfigurovatelnost, kdy je možno změnit mimo jiné i barvy pozadí (54).

Podobnou problematikou jako Job/Queue/Slaves Monitoring se zabývá i Mission Control plugin, avšak řešení je rozdílné již v použité technologii. Zatímco byl JQSMonitoring napsán pouze v Javě a Jelly, Mission Control používá i JavaScript. Výsledkem je řádově kvalitnější vzhled, jak je vidět na následujícím obrázku. (56)



Obrázek 5: Mission Control plugin

Ovšem nelze říci, že by Mission Control byl ve všech ohledech napřed. Ať už to bylo právě zaměření na vzhled nebo kratší doba existence, výsledek postrádá stabilitu, při některých načteních se nezobrazují některé části, jak pravdivě varuje dokumentace (56). Dalším úskalím je nedořešená otázka bezpečnosti pluginu. Hrozba pramení z neošetřených vstupních údajů od uživatele (57). V tomto ohledu ale není úplně osamocený. Jak informovala společnost NCC Group, bezpečnostní mezery obsahovalo kolem 100 pluginů (58). Řazení ve frontě upravuje i Matrix sorter plugin, který se ovšem zaměřuje na úlohy typu matice z maticového projektu (59).

Statistics Gatherer

Plugin sám o sobě slouží pouze ke sběru a odesílání informací o běhu Jenkins. Ačkoliv se jím nedá měnit pořadí úloh ve frontě, přesto by jím získané informace mohly sloužit v externí aplikaci, která by již frontu měnit mohla. (60)

Accelerated-build-now-plugin

Tento plugin řeší obdobný problém, avšak výsledek neumožňuje měnit pořadí v existující frontě na základě přání uživatele, ale pouze umožňuje vytvoření úlohy, která bude umístěna nikoliv jako obvykle na konec, ale na začátek fronty. Dokonce je možné, aby přerušil probíhající úlohu a jinou postavil na její místo. (61)

Obrázek 6: accelerated-build-now-plugin, zdroj obrázku (61)

Z tohoto projektu bude převzat styl práce s frontou skrze QueueSorter z jádra převzatého z projektu Hudson. Naopak nepříliš šťastné je rozvržení kódu, kdy řídicí třída obsahuje velkou část kódu a ještě k tomu v jedné metodě. Tím je značně snížena možnost testování kódu a zhoršena čitelnost.

Priority Sorter

Zásuvný modul řazení fronty podle priority přiřazuje jednotlivým úlohám prioritu v závislosti na skupině úloh, ve které se nachází. Každá skupina úloh může mít přiřazeno více prioritních strategií, na základě kterých je přidělena priorita. Podle strategie fronty se následně priorita přeloží na váhu úlohy, podle které je fronta seřazena. (62)

Modul umožňuje tři druhy strategie fronty úloh:

- Absolutní – úloha s nižším číslem priority má přednost.
- Spravedlivá – úlohy se střídají ve využití výpočetního výkonu.
- Vážená – kompromis obou předchozích, úlohy se střídají, ale těm s vyšší důležitostí (nižším číslem priority) náleží větší část výpočetního výkonu.

Tento zásuvný modul je velmi dobře použitelný v situacích, kdy stejné úlohy mají stále přibližně stejnou důležitost. Již při vytvoření úlohy ve složce, která tvoří skupinu úloh, je jí přiřazena priorita a není třeba dále nic nastavovat. Výhodou je možnost výběru strategie fronty a změny rozsahu priorit.

Naopak slabou stránkou je vyšší složitost spojená s nekompletní dokumentací a především problematická změna pořadí v již existující frontě. Priorita je totiž přiřazena úloze v čase tvorby sestavení a není již aktualizována. Pokud tedy uživatel potřebuje změnit pořadí úloh, které čekají ve frontě, musí provést následující kroky:

- Zrušit původní úlohu ve frontě.
- Případně potvrdit její zrušení.
- Zjistit prioritu úloh, mezi které má být vsunuta.
- Pokud velikosti priorit sousedních úloh na sebe navazují (např. 3 a 4), je nutné zvětšit interval, ze kterého se vybírají priority v jiné části nastavení – Configure System. Druhou možností je zrušení úloh sousedních.
- Tvorba nové úlohy, která bude mít požadovanou prioritu.

Pokud bylo při řešení zvoleno zvětšení intervalu priorit, vznikl problém při zařazení nových úloh do již stávající fronty. Je-li vyžadováno zachování správného poměru priorit, musí být dočasně zastaveno zařazování nových úloh do fronty, což může být značně nepříjemná záležitost. Proto spíše tento problém řešen nebude a fronta nebude správně seřazena. Opomenuta nemůže být ani nutnost změny výchozí hodnoty priority, která zůstává fixní. Pokud bylo při řešení zvoleno rušení sousední úlohy, či dokonce úloh, pak je nutné změnit nastavení jejich priority, znovu vytvořit i jejich sestavení. V některých případech ovšem může dojít k tomu, že stejně bude nutné změnit interval priorit.

Z výše napsaného tedy vyplývá, jaká úskalí čekají na uživatele, který si přeje změnu pořadí úloh ve frontě. Pokud vůbec má práva na změnu priorit, často nic neudělá, aby se vyhnul relativně náročnému procesu.

Block Queued Job

Ačkoliv plugin pro blokování úloh směřuje především na zajištění logických závislostí jednotlivých sestavení, přece se dá použít pro změnu pořadí ve frontě úloh. Úlohy, které mají být provedeny později, se mohou blokovat dokončením úlohy, které jsou předcházející (63). Na rozdíl od Priority Sorter pluginu toto řešení nevyžaduje mazání úloh ve frontě, aby byly vykonávány ve správném pořadí, ale se změnou konfigurace začne návaznost platit i pro vytvořená sestavení. Nevýhodou je nižší efektivnost při více závislostech, takže výpočetní výkon nemusí být využit naplno.

Build Blocker

Build Blocker plugin se použitím a logikou příliš neliší od Block Queued Job či Locks and Latches, důležitým rozdílem ovšem je, že není vázán na konkrétní jméno úlohy, ale lze použít i regulární výraz. Navíc oproti Block Queued Job dokáže rozlišit mezi jednotlivými výpočetními uzly a dokáže blokovat i na základě ostatních čekajících sestavení. Jedná se o významný plugin s počtem 12 098 aktivních instalací v prosinci 2019. (64)

Obrázek 7: Block Queued Job plugin

Dependency Queue

Pokud je potřeba změny pořadí fronty způsobena vnitřní závislostí jednotlivých úloh, dá se s úspěchem použít nepříliš dokumentovaný Dependency Queue plugin. Ve výsledku bude fronta seřazena tak, že závislá úloha bude až za hlavní úlohou. (65)

Fast Track Queue Optimizer

Teprve v roce 2019 založený a možná proto také nepoužívaný je plugin s názvem Fast Track Queue Optimizer. Dalším důvodem, proč ho používá pouze 5 instancí, může být přílišná trivialita. Tento plugin řadí úlohy podle několika málo pravidel. Není tedy možné specifikovat konkrétní úlohu, která má mít před jinou přednost, ale pouze kritéria, na základě kterých některé úlohy mají mít přednost. Tento plugin tedy je možné použít pouze v případě, že chceme řadit frontu podle následujících pravidel:

- Uživatelem spuštěné sestavení má přednost před sestavením spouštěným časovačem nebo externí službou.
- Sestavení vyvolaná z vnějšího prostředí mají přednost.
- Sestavení s předpokládaným kratším časem běhu mají přednost.
- Historicky méně úspěšné sestavení má přednost.
- Dlouho čekající sestavení mají přednost.

Autor, Arno Mooren, navrhuje i přidání nového pravidla, které však je třeba naprogramovat přímo v zdrojovém kódu pluginu. Takový přístup nepovažuji za příliš uživatelsky přívětivý. (66)

Multi-branch priority sorter

Za jistou kuriozitu se dá považovat multi-branch priority sorter plugin vyvíjený především od Zhao Xiaojie, pracujícím v Pekingu v čínské firmě Alauda (67). Zajímavý je tento plugin hned z několika důvodů. Za prvé plugin rozšiřuje jiný plugin – konkrétně Priority Sorter – a přidává možnost dodat každé větvi jinou prioritu. Za druhé obsahuje i firemní skripty pro správu vlastní instance Jenkins včetně názvu uživatele, jeho tokenu i portu, na

kterém se dá přistoupit k Jenkins serveru. Patrné je i použití dockeru a AWS ve firemní infrastruktuře. (68)

Shrnutí

Existující řešení poskytují sofistikované a automaticky konfigurovatelné možnosti úpravy pořadí jako je Priority Sorter plugin. Dále obsahují méně rozvinutá řešení, která frontu spíše ignorují (Accelerated build now plugin), nebo ji rovnou mažou (Purge build queue). Některé pluginy se snaží určit důvody pro změnu pořadí ve frontě (Fast Track Queue Optimizer, Dependency Queue). Žádný z těchto pluginů tedy neumožňuje pružně reagovat na krátkodobé změny priorit, které mohou mít různé příčiny. U konkurenčních produktů k Jenkins pak jen těžko je možno vůbec hovořit o frontě úloh, jelikož požadavky jsou prováděny na serverech poskytovatele služby.

4 Metodika

Jenkins CI, či zkráceně pouze Jenkins je open-source projekt automatizačního serveru s bohatou sadou zásuvných modulů, které podstatně mění a rozšiřují nejen možnosti vzhledu uživatelského rozhraní, ale především funkcionality. Projekt je napsán především v jazyce Java (85,2 %), menší část týkající se především uživatelského rozhraní je napsána v HTML (8 %), JavaScriptu (3,1 %) a CSS (2,6 %)(24). Vzhledem k této skutečnosti bude pro tvorbu pluginu použit jazyk Java.

Na následujících řádcích budou představeny technologie, na nichž spočívají ostatní pluginy Jenkins stejně jako bude ten nově vytvořený touto prací. Jelikož Jenkins je z velké části napsán v jazyku Java, je běžně pro správu závislostí používán Maven a pro tvorbu uživatelského rozhraní kombinace Javy a XML zvaná Jelly, která má i vlastní knihovny.

4.1 Maven

Jak již bylo zmíněno, pro správu závislostí v této práci bude využit nástroj Maven. Nedílnou součástí mnoha projektů v jazyce Java je Maven, nástroj pro podporu vývoje software. Každý projekt využívající Maven má informace o sestavení, správě závislostí a tvorbě dokumentace uloženy v pom.xml (Project Object Model). Nejvyšším tagem v pom je ”project”, který může obsahovat elementy (5; 11):

- parent – identifikační údaje rodičovského projektu
- groupId – celosvětově unikátní identifikátor
- artifactId – unikátní identifikátor v dané skupině
- version – verze artefaktu, tzn. verze produkovaného software
- packaging – druh artefaktu, který bude produkován podle balíčkovacího pluginu
- dependencies – obsahuje jednotlivé automaticky stahované závislosti
- build – obsahuje informace pro sestavení, měly by v něm být konfigurovány některé pluginy (6)

- reporting – pluginy spouštěné v životním cyklu site (více níže)

Maven sám osobě není nic víc než jen framework pro spouštění Maven pluginů, které nesou tíhu všech úkolů. V době psaní této práce existovalo 50 oficiálních aktivních pluginů podporovaných projektem Maven ve čtyřech oblastech: Core, Packaging, Reporting a Tools. Základní součástí jádra jsou životní cykly projektu, které jsou definované tři: default, clean a site. Cyklus "site" je navázán na stejnojmenný plugin jádra a je složen z fází: pre-site, site, post-site a site-deploy. Životní cyklus "clean" je navázán taktéž na stejnojmenný plugin jádra clean a je složen z fází: pre-clean, clean a post-clean. Třetí životní cyklus, default, má 23 fází, mezi kterými se nalézají fáze: validate, compile, test, package, integration-test, verify, install, deploy. Životní cyklus default má na sebe navázány pluginy ze zmiňované oblasti Packaging. (6; 7; 8)

Součástí Maven je i Maven Archetype, což je nástroj pro generování základního rozložení projektu, který používá Maven. Archetype se snaží udržet motivaci nově příchozího uživatele (programátora) vygenerováním vzorového projektu, který umožní zkrácení doby, kdy nic nefunguje, a zároveň budou zachovány firemní standardy. Proto bude archetype využit pro tvorbu kostry projektu. (9)

HPI

Pro testovací účely při vývoji pluginu pro Jenkins bude tvořena lokální instance Jenkins za použití vlastního Maven pluginu cílem "hpi:run". Mezi další cíle patří hpi:create pro vytvoření struktury jar souboru, hpi:hpi pro vytvoření souboru pluginu s příponou hpi či hpi:hpl. (10)

4.2 Frameworky a knihovny

Stapler

Framework Stapler je jednou z neoddělitelných částí Jenkins. Jeho cílem je bezpečně spojit Java třídy s URL adresou. Kořenová adresa je spojena se singltonem třídy Hudson, URL adresa ostatních objektů je závislá na jejich dostupnosti od Hudsonu. Pro vyřízení požadavků na nekořenové adresy se snaží Stapler najít odpovídající metodu. (12) Pokud přijde požadavek na adresu "/prvni/druhy", pak se hledá následujícím způsobem:

Jako první se hledá na Jenkins metoda getPrvni(String), které by se jako parametr předložil řetězec "druhy", a na vráceném objektu by se zavolala metoda doIndex. Pokud se taková metoda nenajde, hledá se getPrvni() či getDruhy(), kde by na návratovém objektu byla metoda getDruhy, resp. getPrvni, a k dalšímu návratovému objektu patří jelly soubor index. Pokud takový soubor neexistuje, hledá se u prvního objektu pod názvem druhy.jelly (resp. prvni.jelly). Navíc platí, že na místě jelly souborů se dá použít groovy. (12)

Celou situaci komplikuje možnost porušení této posloupnosti implementací StaplerProxy, které má při zpracování URL žádosti přednost, a možností implementace StaplerFallback, které naopak odsune toto zpracování na konec. (12)

Jelly

Jelly je nástroj kombinující Javu a XML v skriptovací engine spravovaný od The Apache Software Foundation a inspirovaný nápady z JavaServer Pages, Velocity, Canoon

a Antu. K vlastním Jelly značkám je zároveň umožněno vytvoření vlastních. Jelly dokáže dynamicky generovat HTML či XML odpověď na vyvolanou událost na základě skriptu vytvořeného z jelly souboru. Jednotlivé značky v jelly obsahují metodu doTag(), která se volá při spuštění jelly skriptu. Při spojení Java Beans a Jelly značky se hovoří o tzv. Jelly Bean, který může mít metody spouštěné při použití tagu jako je run(), execute() či invoke(). (13)

Tvorba jelly beanu probíhá nepárovým tagem define:jellybean, kde se jako parametr uvede jméno nového jelly tagu a jméno třídy, která ho obslouží. Větší množství takovýchto jelly beanů je možno uzavřít do jelly knihovny, která se tvoří párovým tagem "define:taglib", kde se jako parametr použije URI nové knihovny. (13)

JenkinsTagLib

JenkinsTagLib je vlastní Jelly knihovna projektu Jenkins, která se skrývá pod odkazem "/lib/hudson" ve většině jelly souborů jak pluginů, tak i jádra. Nejdůležitější značka z hlediska této práce, kterou tato knihovna obsahuje, je nepárové "queue", které používá výchozí implementace zobrazení fronty. V její definici je pak použit ze stejné knihovny další tag: setIconSize, který zajišťuje, že ikona informující o stavu sestavení bude větší než ostatní ikony. Implementace tagu queue bude využita pro tvorbu vlastního zobrazení fronty. (14; 42; 40; 15)

LayoutTagLib

LayoutTagLib je další z Jelly knihoven projektu Jenkins a pro jelly skripty jádra i pluginů je dostupná na adrese "/lib/layout". Z hlediska této práce je třeba zmínit tagy pane, breakable, icon, stopButton a ajax, které budou použity k implementci fronty. (16; 40; 42)

FormTagLib

FormTagLib je obdobně jako LayoutTagLib Jelly knihovna projektu Jenkins přímo používaná ve skriptech jádra i pluginů a zároveň slouží i při definici dalších tagů z jiných knihoven(např. JenkinsTagLib). Dostupná je pod URI "/lib/form". Je zaměřena na podporu tvorby formulářů. Tato knihovna bude využita pro tvorbu potvrzení restartu pluginu. (17; 40)

Test Harness

Test Harness je komponenta Jenkins používaná při vytváření JUnit testů na jádro i pluginy. Jejím základním prvkem je JenkinsRule, která dokáže vytvořit testovací instanci Jenkins. Dále je schopna vytvořit pro testovací účely úlohu, závislou výpočetní jednotku, složku i DummySecurityRealm. Testovat je možné i ukládání konfigurace pro pohled, uzel apod. nebo výsledek provedení sestavení. JenkinsRule umožňuje i kontrolovat obsah logu spuštěné testovací instance či status kód http odpovědí. Pokud je samotný kód odpovědi nedostačující, je možné testovat i obsah na dané adrese či vytvořit webového klienta. Test Harness bude využit pro tvorbu testovací instance Jenkins skrze JenkinsRule. (18)

4.3 Principy tvorby pluginu pro Jenkins

Dříve než bude pojednáno o principech tvorby pluginů pro Jenkins, je třeba pro úplnost předeslat, že vzhledem k velkému počtu pluginů je v současnosti kladen důraz především na rozvoj těch existujících, aby se tak zabránilo případné duplikaci. Vzhledem k tomu, že nebyl takový nalezen (viz. kapitola Literární rešerše), bude proto vytvořen nový. (3)

Nastavení prostředí

Základním krokem před započetím samotného programování je vytvoření vhodného prostředí. Základním kamenem každého rozsáhlejšího projektu je správce balíčků, v projektu Jenkins se používá správce jménem Maven 3, který je tedy třeba nainstalovat. Navíc je možné skrze něj vytvořit i základní kostru zásuvného modulu. (3)

Dalším krokem je volba vývojového prostředí, tzv. IDE. Samozřejmě zdrojový kód je možno upravovat v mnoha textových editorech, ale práci velmi usnadní vhodně zvolené vývojové prostředí. Podpoře od projektu Jenkins se těší tři vývojová prostředí, a to NetBeans, IntelliJ IDEA a Eclipse. (3) Pro tvorbu této práce bude použito prostředně jmenované IDE, IntelliJ IDEA, které je mezi nimi jediné proprietární.

Struktura pluginu

Základem každého pluginu je soubor pom.xml, který obsahuje odkaz na verzi předka a odkaz na verzi jádra, což nemusí být pravdou u pluginů do verze Jenkins 1.645. POM soubor každého pluginu v oficiálním seznamu musí obsahovat licenci, pod kterou je zveřejněn, stejně jako jeho název a popis. Z hlediska správce rozšíření je v POM zásadní i seznam závislostí, groupId, artifactId a verze. Zdrojový Java kód pluginu se umísťuje na adresu src/main/java, dynamické části uživatelského rozhranní jako Jelly či Groovy soubory se umísťují na adresu src/main/resources a statické části pluginu se umísťují na adresu src/main/webapp. (3)

Základní pro funkčnost každého pluginu jsou zmiňované Java soubory. Propojení mezi nimi a jádrem Jenkins probíhá na základě dvou možných rozšíření. První, spíše již historická a nedoporučovaná možnost je rozšířit třídu Plugin. Slabina takového přístupu se ukázala při zvětšení počtu pluginů, kdy na základě této informace není vůbec možné říci, jaké části jádra se plugin dotýká. Druhou možností je rozšíření třídy typu Extension Point, které je v současnosti doporučovanou formou. (3)

Je-li zvolen druhý přístup, je třeba nalézt vhodný Extension Point, jenž bude rozšířen vlastní třídou, která bude zpravidla obsahovat vnitřní statickou třídu, tzv. deskriptor. Ten musí být označen anotací Extension. Tento deskriptor slouží pro zobrazení konfigurace pluginu, jejíž uložení vyvolá konstruktor s anotací DataBoundConstrutor. Zároveň může být použit jako pohled na nastavení dané třídy v době vzniku její instance. (3; 4)

Předdefinované objekty

Předdefinovaným objektem rozumíme objekt, jehož definice se nevyskytuje v daném souboru ani v explicitně připojených knihovnách. Předdefinované objekty v jelly skriptech, tzn. skriptech Jelly frameworku, hrají důležitou úlohu v jádru i pluginech. Jejich znalost je velmi důležitá, neboť ze samotného skriptu nemusí být jasný jejich obsah. Pro účely této práce bude použit objekt it, h, app a rootUrl. Objekt ”it” zpravidla odkazuje na

odpovídající java třídu vzhledem k místu, kde je jelly skript uložen. Pokud je skript uložen např. na adrese ”src/main/resources/cz.mendelu.xotradov/SimpleQueueWidget”, pak odpovídající třída je ”src/main/java/cz.mendelu.xotradov/SimpleQueueWidget.java”. Toto ale nemusí být vždy pravda, protože je možné při volání daného skriptu mu ”podstrčit” jinou třídu. (70; 41)

Další předdefinované objekty jsou ”app”, což je instance Jenkins/Hudson, ”instance”, což je objekt s možnou konfigurací, ”descriptor”, což je vnitřní statická třída onoho objektu, ”h”, což je instance třídy ”hudson.Functions”. Mezi předdefinované objekty můžeme zařadit i tři URL adresy, které jsou definovány značkou ”layout” nebo ”ajax”: ”rootURL” pro kořenovou adresu instance Jenkins, ”resURL” pro statické zdroje(např. HTML, nikoliv jelly), ”imagesURL” pro grafické soubory. (70)

Kořenovou adresou se myslí řetězec znaků představujících část URL adresy, na které je dostupná úvodní obrazovka Jenkins serveru. Kořenová adresa začíná až od generické domény, případně portu. V základním nastavení kořenovou doménu představuje pouze řetězec ”/” nebo ”/jenkins/” podle stáří verze Jenkins.

5 Návrh pluginu

5.1 Analýza požadavků

Funkční požadavky

- Posun prvku vzhůru – oddálení vykonání.
- Posun prvku dolů – zkrácení času čekání.
- Vymazání všech požadavků na posunutí.
- Po spuštění pluginu jsou prvky řazeny systémem FIFO, pokud není naistalován QueueSorter.
- Umožnění filtrování fronty skrze pohledy.

Nefunkční požadavky

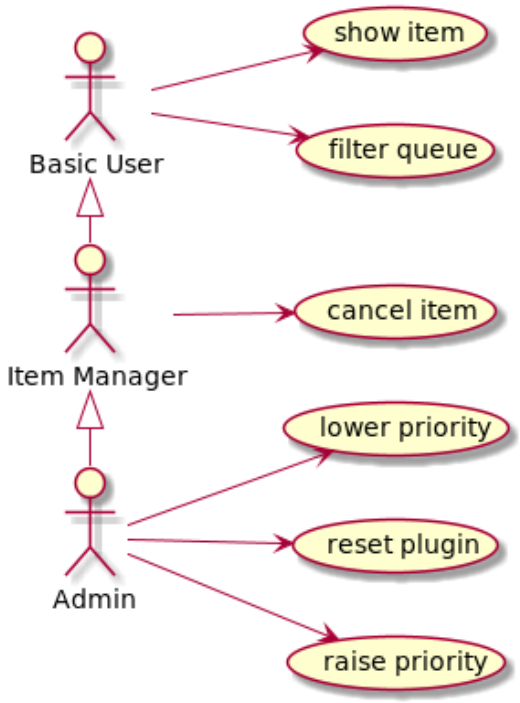
Jelikož se v této práci nejedná o vytvoření nového systému či aplikace, ale o jeho rozšíření, zásadní význam má zachování současných funkcionalit a bezpečnostních politik. Svůj význam má i přenesení vysokých bezpečnostních standardů jádra do nových funkcionalit nového zásuvného modulu, aby se tak nestaly jistou Achillovou patou celého systému. Z tohoto hlediska je důležitá i integrace s Update Center, které zabezpečuje instalaci nových verzí a bezpečnostních záplat.

- Fronta dokáže unést i větší množství prvků, než je obvyklé.
- Při změně pořadí nedojde ke zrušení prvku, i kdyby byl znovu okamžitě vytvořen.
- Doba odezvy maximálně tři vteřiny.
- Nové ikony jsou snadno odlišitelné od stávajících.
- Fronta se zobrazí pouze oprávněnému uživateli.

- Pořadí ve frontě lze upravit pouze oprávněnému uživateli.
- Snadná rozšiřitelnost o další typy posunů.
- Kompatibilita s rozšířením PrioritySorter.
- Instalace z Plugin Manageru.

5.2 Use Case

Jak zobrazuje use case diagram, byly nalezeny tři různé skupiny uživatelů. První skupina jsou buďto nepřihlášení uživatelé nebo uživatelé, kteří mají pouze právo na čtení(podmínky získání tohoto práva se mohou lišit u jednotlivých instancí systému). Tito jsou schopni pouze obsah fronty zobrazit, případně přepnutím pohledu ji filtrovat. Druhá skupina představuje uživatele s výším oprávněním, kteří mohou odstranit prvek z fronty. Poslední skupinu tvoří uživatelé s nejvyšším oprávněním, kteří představují správce systému, a jako jediní mají právo ovlivňovat pořadí prvků ve frontě.

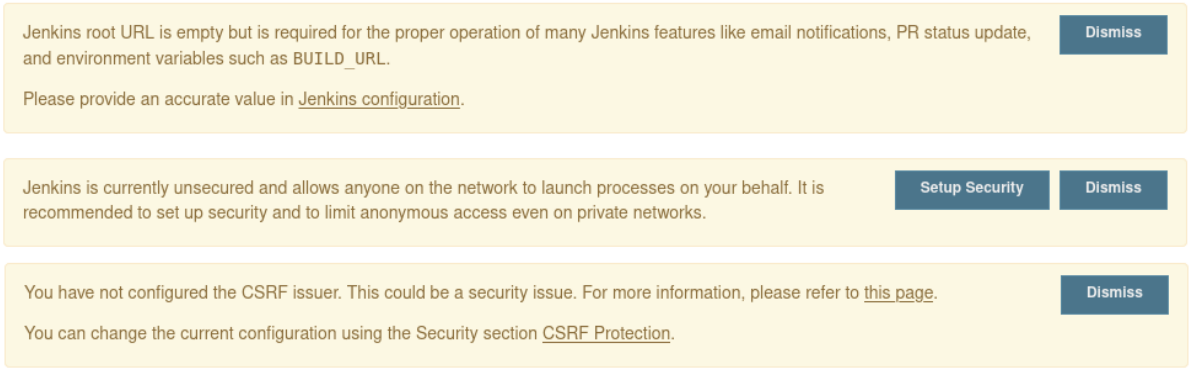


Obrázek 8: Use case diagram

5.3 Ikony

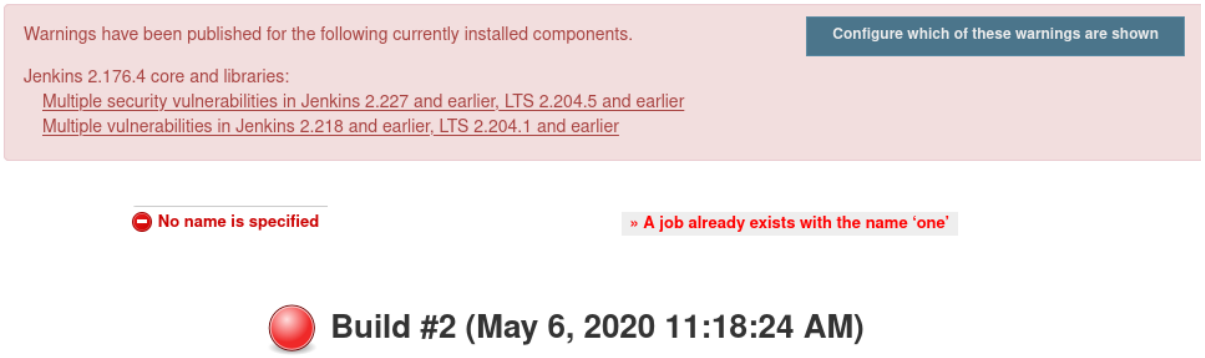
Pro větší srozumitelnost a přehlednost nebyly použity pro posun prvků ve frontě odkazy ve formě textu, ale ve formě ikon šipek. Jelikož komunita uživatelů Jenkins se rozprostírá po celém světě a ne všichni tudíž mají dostatečné znalosti cizích jazyků, použití ikon eliminuje nebezpečí nedorozumění a zároveň pomáhá v orientaci. Zvolením modré barvy šipek se předchází nechtěné záměně s nedalekým červeným tlačítkem pro zrušení úlohy. Červená barva se v Jenkins vyskytuje u prvků charakteru smazání, chyby vstupu uživatele,

varování před kritickou bezpečnostní vadou či při pádu sestavení nebo instance. Zároveň žlutá barva se vyskytuje spíše u prvků, které mají charakter varování, které může být ignorováno. Vzhledem k bílému pozadí nepřichází v úvahu ani světlé odstíny, které by byly těžko viditelné, stejně jako příliš tmavé, které by mátly uživatele s rozšířeními v prohlížeči typu Dark Reader.



Obrázek 9: Ukázka varování v sekci správy Jenkins.

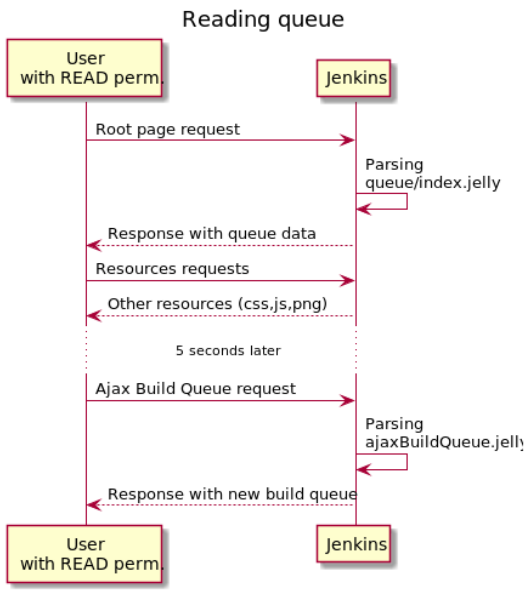
Obrázek 10: Ukázka varování v nastavení úlohy.



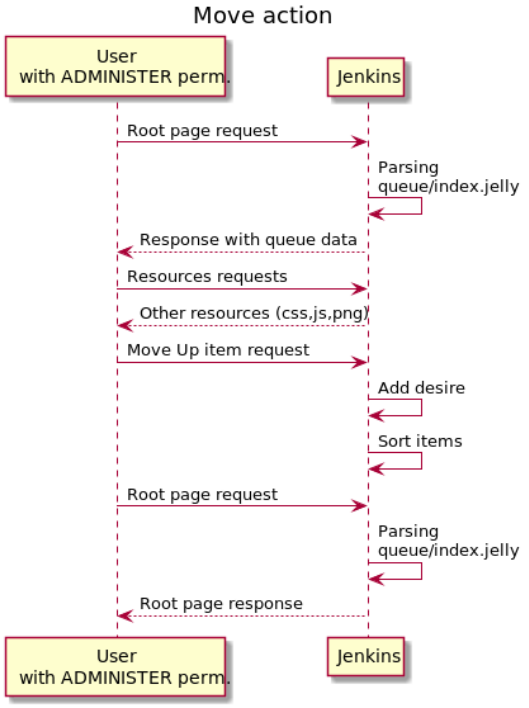
Obrázek 11: Použití červené barvy. Nahoře varování o kritické bezpečnostní záplatě, uprostřed oznámení o chybném vstupu od uživatele, dole neúspěšné sestavení.

5.4 Sekvenční diagramy

Na následujících sekvenčních diagramech jsou zobrazeny typické úlohy pluginu. Je vhodné předeslat, že požadavky od uživatele prochází autentizací a autorizací, ale pro lepší zobrazení podstaty to není v diagramech uvažováno. Na diagramu č. 12 je zobrazeno čtení fronty uživatelem s oprávněním ke čtením. Jako první uživatel skrze svůj prohlížeč vstoupí na kořenovou adresu Jenkins serveru. Na tomto místě není uvedena konkrétní



Obrázek 12: Sekvenční diagram přístupu ke frontě uživatele, který disponuje právy na čtení.



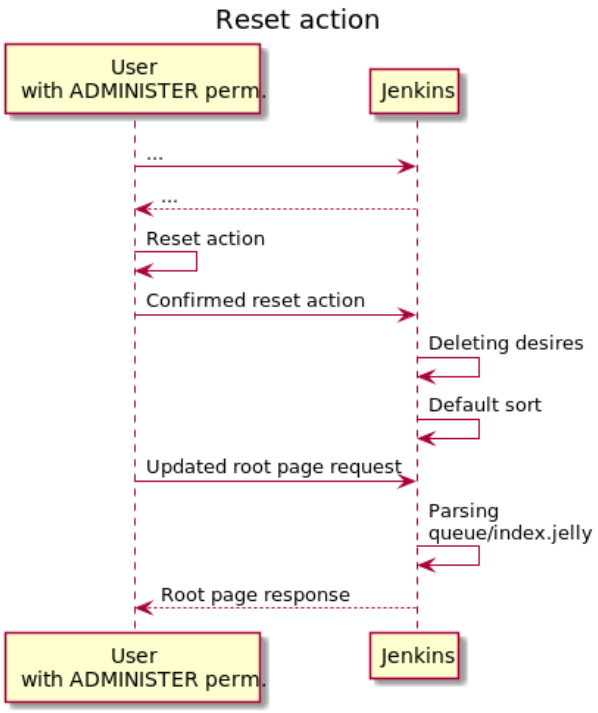
Obrázek 13: Sekvenční diagram změny řazení fronty, uživatel disponuje administrátorskými právy.

kořenová adresa, neboť závisí na nastavení tzv. master instance Jenkins serveru. Při zpracování tohoto požadavku Jenkins prozkoumá i přihlášené widgety, mezi kterými se nachází i SimpleQueueWidget. Následně Jenkins hledá na odpovídající adrese soubor pro zobrazení a nalezne index.jelly, na jehož základě vytvoří kostru odpovídající html stránky. Následně si prohlížeč vyžádá jednotlivé grafické komponenty jako jsou css, javascript či png soubory a ty obdrží a uloží do cache. Po uplynutí 5 vteřin prohlížeč požádá o aktualizovaný obsah fronty sestavení a zvlášť o obsah seznamu zpracovávaných úloh na výpočetních jednotkách. Pro zpracování požadavku Jenkins zpracuje ajaxBuildQueue.jelly na část html stránky, kterou vrátí uživateli.

Na obrázku č. 13 je zobrazen sekvenční diagram přesunu prvku ve frontě. Jako první, obdobně jako u minulého diagramu, uživatel požádá o stránku na kořenové adrese. Aby na ni mohl následně obdržet odpověď, je třeba zpracovat index.jelly náležející k widgetu fronty. Pro správné zobrazení je ještě nutné vyžádat si jednotlivé grafické prvky stránky. Když je uživateli fronty stránka zobrazena, uživatel má příslušná práva a fronta obsahuje sestavitelné prvky, pak může klikem na šipku požádat o posunutí prvku. Je-li tento požadavek možné uskutečnit, pak se tato volba uloží, a pro promítnutí této volby dojde k novému seřazení fronty. Následně je aktualizovaná fronta zobrazena uživateli výše zmíněným způsobem.

Sekvenční diagram, zobrazený na obr. č. 14, ukazuje navrácení k řazení neovlivněnému Simple Queue pluginem. Na začátku je naznačeno vynechání opakující se inicializační sekvence. Předpokládá se, že uživatel se nachází na kořenové adrese. Pokud si přeje vymazat všechny své předchozí požadavky, klikne na příslušné tlačítko restartu pluginu a potvrdí tuto volbu. Jenkins přijme tento požadavek, vymaže všechna minulá

přání na změnu pořadí fronty a seřadí frontu. Po novém vyžádání stránky se uživateli zobrazí fronta seřazená výchozím způsobem.



Obrázek 14: Sekvenční diagram restartu pluginu, uživatel disponuje administrátorskými právy.

5.5 Uživatelské rozhraní

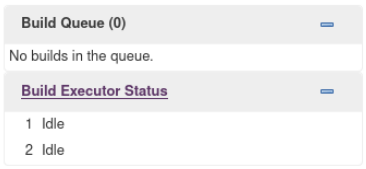
Jelikož tato práce nemá za cíl vytvoření nové aplikace, ale zásuvného modulu do již stávající, tak změna ve vzhledu uživatelského rozhraní je patrná pouze v některých situacích. Pokud je vidět seznam kořenových akcí, pak je i patrné přidání akce ”Reset Simple-Queue”. Mírně složitější je situace u samotné fronty: pokud je fronta prázdná nebo uživatel nemá právo čtení k žádnému prvku, pak výsledek odpovídá obrázku č. 15. Pokud sice fronta není prázdná a uživatel disponuje právem na čtení alespoň jednoho prvku, ale nemá práva ke správě, potom výsledek odpovídá obr. č. 16. Pokud fronta prvky obsahuje a uživatel má právo na jejich čtení i správu, výsledek odpovídá obr. č. 17.

6 Implementace

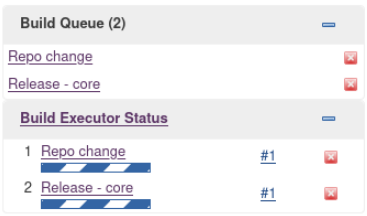
6.1 Extension Points

Pro změnu pořadí řazení fronty byl vybrán Extension Point QueueSorter. V úvahu by připadal i AbstractQueueSorterImpl, který je jeho potomkem. (38) Ten byl použit na napodobení základního řazení třídou DefaultSorter.

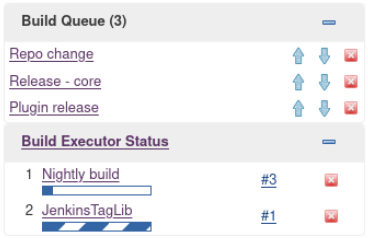
Kromě QueueSorteru byly použité i akce pro zprostředkování vytvoření nových adres. Byly implementovány tři akce: pro aktualizaci obsahu fronty SimpleQueueUpdateAction,



Obrázek 15: Prázdná fronta sestavení. Zdánlivě působí jako základní widget.



Obrázek 16: Fronta sestavení s prvky blokovanými právě probíhající výpočetní úlohou.



Obrázek 17: Fronta se sestavitelnými prvky.

pro posunutí prvku MoveAction a pro obnovení původního řazení ResetAction. Všechny tyto akce jsou potomkem RootAction. Teoreticky by bylo možné pro novou adresu využít TransientActionFactory, TransientBuildActionFactory, TransientComputerActionFactory, TransientProjectActionFactory, TransientUserActionFactory, TransientActionViewActionFactory a UnprotectedRoodAction, avšak tyto neodpovídají dané situaci. UnprotectedRootAction by dokonce umožnil přístup k frontě uživatelům, kteří na to nemají oprávnění. Ostatní jmenované umožňují vytvoření adresy v jiných částech aplikace, kde by to buď postrádalo smysl, nebo vytvářelo zbytečné zdržení. (69)

Poslední z řady implementovaných míst pro rozšíření je Widget a Plugin. Prvně jmenované slouží pro přidání vlastního widgetu fronty, druhé bylo použito pro odebrání widgetu původního.

6.2 Front-end

Na rozdíl od původní implementace nebyl použit tag z JenkinsTagLib, protože by nebylo možné upravit vzhled fronty, ale pouze případně její obsah ve smyslu prvků, které se zobrazí. Naopak využity byly značky z LayoutTagLib a FormTagLib tak, aby původní funkcionalita byla zachována. Na následujících řádcích je ukázán rozdíl v zobrazení fronty, přesněji její obnovy, mezi původním tagem z JenkinsTagLib (40) a indexu z nového widgetu.

```
<j:if test="${ajax==null and h.hasPermission(app.READ)}">
  <script defer="defer">
    refreshPart('buildQueue',"${rootURL}/
       ${h.hasView(it,'ajaxBuildQueue')?it.url:''}ajaxBuildQueue");
  </script>
</j:if>
```

Zpočátku jsou obě situace stejné, když se ověřuje, zda vůbec má k obnově dojít. Vzhledem k nedostatečné dokumentaci je nesnadné určit původ proměnné ”ajax”, která není v souboru deklarována ani není vstupním parametrem. Odpovědí by mohla být značka <ajax> z LayoutTagLib. Jasnější je situace u druhé podmínky, kde je zavolána kontrolní metoda práva na čtení z třídy hudson.Function. (70)

Rozdíl nastává až při tvorbě adresy pro obnovení. V první ukázce ze základní implementace se prvek s id ”buildQueue” aktualizuje na adrese tvořené z kořenové

adresy a výsledku ternárního operátoru, který zajišťuje, že vstupní data budou odpovídat aktuálnímu pohledu. Stapler se následně pokusí požadavek na takovou adresu vyřešit a nalezne soubor ajaxBuildQueue.jelly, jehož zpracováním frontu zobrazí.

V případě této práce jádro rozdílu nastává v podmínce ternárního operátoru, kdy není možné použít metodu hasView s parametrem "it", protože se pod "it" nalézá jiný objekt (70). Pokud pohled filtrovaný je, pak se zobrazí základní implementace, pokud filtrovaný není, pak se zobrazí implementace Simple Queue pluginu. Díky tomu je možné upravit uživatelské rozhraní a zároveň zachovat funkci filtrace fronty.

```
<j:if test="${ajax==null and h.hasPermission(app.READ)}">
  <script defer="defer">
    refreshPart('buildSimpleQueue',"${rootURL}/
        ${filtered?view.url:'updateQueue/'}ajaxBuildQueue");
  </script>
</j:if>
```

6.3 Testování

Byly vytvořeny testy pro veřejné metody vyjma metod bez hlubší vnitřní logiky. Pro tvorbu testů byly použity frameworky JUnit 4, Mockito a Jenkins Test Harness. Zároveň byla vytvořena pomocná třída, která implementuje často se opakující procesy při testování, jako je naplnění výpočetních jednotek či vytvoření a spuštění úlohy ve frontě.

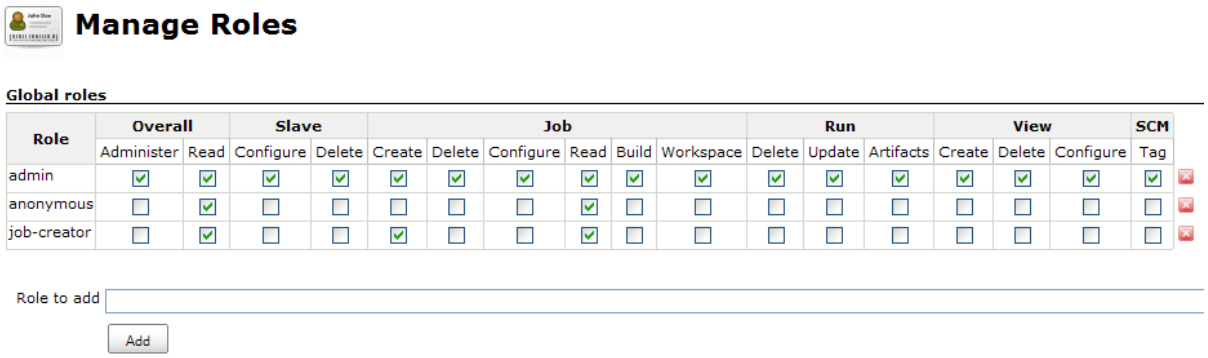
Z frameworku Jenkins Test Harness byla nejvíce využita možnost vytvoření vytvoření vlastní testovací instance Jenkins skrze JenkinsRule. Pro vytvoření testu byla použita anotace z JUnit @Test. Mockito bylo využito pro mockování požadaků uživatele na Stapler a odpovědí Stapleru uživateli.

7 Diskuze

7.1 Dosažení cíle

Cílem této práce bylo rozšíření možností Jenkins CI o změnu pořadí jednotlivých úloh ve frontě oprávněným uživatelem včetně příslušné úpravy uživatelského rozhraní. Tohoto cíle bylo dosaženo ve všech ohledech. Použity byly kroky deklarované v cílech práce, a to analýza fronty sestavených úloh, jejího základního řazení a stávajících možností úpravy fronty skrze pluginy, stejně jako analýza konkurenčních služeb k Jenkins s ohledem na řešení tohoto problému. Následně byl vypracován návrh pluginu, který byl implementován a otestován za použití JUnit testů. Žádost o zařazení do oficiálního seznamu zásuvných modulů byla po úpravách na základě připomínek od člena infrastrukturního teamu Jenkins schválena (72). Následně byla schválena žádost o přidělení práv na zveřejnění pluginu. Aktuální verze pluginu stejně jako zdrojový kód je dostupný na adrese: https://github.com/jenkinsci/simple-queue-plugin.

Zásadní otázkou při kontrole práv uživatele je, jaká práva by uživatel měl mít pro úpravu pořadí. Právo na čtení jistě není dostatečné a nejvyšší administrátorské právo také přesně neodpovídá podstatě věci. Podobný problém byl v případě zrušení prvku vyřešen vytvořením speciálního práva na zrušení prvku v jádře Jenkins, což by mohla být i cesta řešení tohoto problému, avšak i pokud by taková změna byla přijata, neslo by to s sebou nutnost zvýšení minimální podporované verze Jenkins. Podobné úskalí by s sebou neslo i použití oprávnění správce, které bylo přidáno v době psaní této práce.



Obrázek 18: Správa oprávnění. Zdroj: dokumentace Role-based Authorization Strategy pluginu. (73)

7.2 Srovnání s konkurenčními řešeními

Jak již bylo dříve řečeno, ostatní pluginy zabývající se řazením fronty nejsou plnou alternativou k této práci, neboť se silně koncepčně odlišují. Mnohé pouze seřazenou frontu zobrazují a neumožňují její změnu, jak je tomu u Mission Control nebo JQSMonitoring. Jiné sice do fronty zasahují, ale pouze aby prvky fronty rušily, což se týká Purge Build Queue, No Agent Job Purge a Queue Cleanup. Pokud už dochází k řazení fronty, pak zdrojem dat pro řazení je zpravidla kombinace konfigurace s informacemi o úlohách, jak je tomu např. u Priority Sorteru. Obecně tato koncepce není ani lepší nebo horší, ale je odpovědí na jiné potřeby. Pokud se dá již dopředu říci, jaká úloha (Task) bude mít

přednost vůči jiné, a tato situace se vždy stejně opakuje, pak je vhodnější použít jiné řešení než touto prací vytvořený Simple Queue plugin. Do této kategorie lze zařadit Priority Sorter, Build Blocker, Dependency Queue, Fast Track Queue Optimizer a Multi-branch priority sorter.

Jiná situace ovšem nastává, pokud je třeba flexibilně reagovat na dynamicky se měnící potřeby, které jednotlivé kategorie těžko dokážou obsáhnout. V té situaci je stále možné použít Priority Sorter a ručně měnit konfiguraci priority, což je ovšem časově náročnější. Rychlejší je použití accelerated-build-now-pluginu, který sice zajistí přednostní vykonání konkrétní úlohy, avšak může způsobit přerušení jiné úlohy, což je u těch déle trvajících značně nežádoucí. V takové situaci má Simple Queue plugin své místo, protože je rychlejší než Queue Sorter a zároveň dokáže upravovat řazení fronty.

7.3 Možnosti rozšíření

Jednou z možností dalšího rozšíření této práce je přidání dalších funkcí pluginu, jako by mohlo být přesunutí prvku na začátek či konec fronty nebo vrácení prvku na jeho původní místo. Stávající řešení bylo vytvořeno s předpokladem možného rozšíření o další typy posunů za použití stávající třídy MoveAction, kde by bylo nutné implementovat metodu pro nový typ pohybu. Zároveň by bylo třeba upravit i příslušnou část uživatelského rozhraní – jelly soubory. Takovéto rozšíření by tedy nemělo být příliš náročné.

Složitější otázkou by mohla být integrace s ostatními pluginy, především pak s těmi, které přistupují k frontě. Nejzřetelnější kolize by mohla nastat, pokud by byl vytvořen další plugin implementující vlastní BuildQueueWidget. Snadněji řešitelné by mohly být případné kolize s pluginy, které rozšiřují třídu AbstractQueueSorterImpl nebo QueueSorter.

7.4 Ekonomické hledisko

Vyčíslení ekonomické návratnosti této práce, stejně jako u mnoha jiných softwarových projektů, je velmi obtížné, jelikož přínosy nejsou jednoduše měřitelné. Přínosy jsou dvou druhů – za prvé se jedná o ušetření času uživateli a za druhé se jedná o optimalizaci používání výpočetního výkonu firmy vzhledem k jejím potřebám. Počet uživatelů, kteří budou používat toto rozšíření, se v tuto chvíli nedá odhadnout, neboť bylo teprve nedávno zveřejněno a může být zdarma nainstalováno do každé instance Jenkins, kterých je v řádu desítek až stovek tisíců, jak již bylo dříve napsáno. Vzhledem k tomu, že pro úpravu fronty se vyžaduje administrátorské oprávnění, je zásadní počet administrátorů, který se dá pouze odhadnout z počtu aktivních instancí. Je pravdou, že časová úspora může činit ve většině případů pouze maximálně několik minut denně, avšak je třeba mít na paměti, že se jedná o cenný čas administrátora Jenkins, tedy pravděpodobně zkušeného pracovníka na vedoucí pozici.

Dalším neopominutelným přínosem je již zmíněná optimalizace alokace zdrojů firmy, kdy se výpočetní kapacita dostane dříve k důležitějším úlohám. Díky tomu lze zlepšit proces průběžné integrace, což může mít za následek dřívější odhalení chyb kódu nebo rychlejší dodání bezpečnostní záplaty. Velikost těchto přínosů je závislá na stupni průběžné integrace, přičemž větší přínos je možný, pokud je zaveden vyšší stupeň průběžné integrace (průběžné nasazení či dodání).

Druhou stranou ekonomického hlediska jsou náklady, které byly spojené s vytvořením pluginu. Jedná se především o čas autora, který se této problematice intenzivněji věnoval několik měsíců (od prosince 2019 do května 2020), avšak čas bylo nutné věnovat pro získání vědomostí na úrovni běžného uživatele Jenkins, získání orientace v celé problematice a především pro pochopení některých částí jádra Jenkins.

8 Závěr

Výsledný zásuvný modul systému Jenkins je funkční alternativou či doplňkem k existujícím řešením. Jeho nasazení umožňuje optimalizaci použití výpočetního výkonu pomocí úpravy pořadí úloh ve frontě. Stávající uživatelské rozhraní modul dotváří přidáním nových prvků do zobrazení fronty u oprávněného uživatele. Toto řešení umožňuje snadnější a rychlejší reakci na měnící se potřeby než stávající řešení, založená primárně na konfiguračních souborech, aniž by bylo dotčeno základní řazení systémem FIFO.

Analýzu stávajících řešení a rozbor základní implementace fronty v Jenkins následoval návrh a samotná tvorba modulu. Byly vytvořeny jednotkové testy i testy celé funkcionality za použití frameworku JUnit, Mockito a knihovny Jenkins Test Harness. Modul respektuje oficiální pravidla pro tvorbu pluginů, o čemž svědčí jeho zařazení do oficiálního seznamu a následné udělení práv pro zveřejnění. Díky tomu je řešení připraveno k použití na produkčních instancích Jenkins serveru.

9 Literatura

[1] Andrady, A.L. & Neal, M. *Applications and societal benefits of plastics* The Royal Society Publishing, 2009. [cit. 28. 12. 2019] Dostupné z https://doi.org/10.1098/rstb.2008.0304.

[2] Thompson, R. C. et al. *Plastics, the environment and human health: current consensus and future trends* The Royal Society Publishing, 2009. [cit. 28. 12. 2019] Dostupné z https://doi.org/10.1098/rstb.2009.0053.

[3] Kawaguchi, K. et al. *Plugin Tutorial - Jenkins* [online]. Atlassian, 2019. [cit. 4. 4. 2020] Dostupné z https://wiki.jenkins.io/display/JENKINS/Plugin+tutorial.

[4] Kawaguchi, K. et al. *Descriptor - Jenkins - Jenkins Wiki* [online]. Software in the Public Interest, Inc., 2020. [cit. 4. 5. 2020] Dostupné z https://javadoc.jenkins-ci.org/index.html?hudson/model/Descriptor.html.

[5] The Apache Software Foundation *Apache Maven – Introduction* [online]. The Apache Software Foundation, 2020. [cit. 4. 5. 2020] Dostupné z http://maven.apache.org/ref/3.6.3/.

[6] The Apache Software Foundation *Maven – Available Plugins* [online]. The Apache Software Foundation, 2020. [cit. 14. 5. 2020] Dostupné z https://maven.apache.org/plugins/index.html.

[7] The Apache Software Foundation *Maven Core – Lifecycles Reference* [online]. The Apache Software Foundation, 2020. [cit. 14. 5. 2020] Dostupné z http://maven.apache.org/ref/3.6.3/maven-core/lifecycles.html.

[8] The Apache Software Foundation *Maven Core – Plugin Bindings for default Lifecycle Reference* [online]. The Apache Software Foundation, 2020. [cit. 14. 5. 2020] Dostupné z http://maven.apache.org/ref/3.6.3/maven-core/default-bindings.html.

[9] The Apache Software Foundation *Maven Archetype – About* [online]. The Apache Software Foundation, 2020. [cit. 14. 5. 2020] Dostupné z http://maven.apache.org/archetype/index.html.

[10] Kawaguchi, K. et al. *Maven Jenkins Plugin – Plugin documentation* [online]. Software in the Public Interest, Inc., 2020. [cit. 14. 5. 2020] Dostupné z http://jenkinsci.github.io/maven-hpi-plugin/plugin-info.html.

[11] The Apache Software Foundation *Maven Model – Maven* [online]. The Apache Software Foundation, 2020. [cit. 14. 5. 2020] Dostupné z http://maven.apache.org/ref/3.6.3/maven-model/maven.html.

[12] Kawaguchi, K. et al. *Architecture – Jenkins – Jenkins Wiki* [online]. Software in the Public Interest, Inc., 2016. [cit. 4. 5. 2020] Dostupné z https://wiki.jenkins.io/display/JENKINS/Architecture.

[13] The Apache Software Foundation *Jelly: Executable XML* [online]. The Apache Software Foundation, 2017. [cit. 11. 5. 2020] Dostupné z http://commons.apache.org/proper/commons-jelly/.

[14] Jenkins project infrastructure team *JenkinsTagLib* [online]. Software in the Public Interest, Inc., 2020. [cit. 11. 5. 2020] Dostupné z https://javadoc.jenkins.io/lib/JenkinsTagLib.html.

[15] Kawaguchi, K. et al. *jenkins/setIconSize.jelly* [online]. GitHub, Inc., 2020. [cit. 5. 5. 2020] Dostupné z https://github.com/jenkinsci/jenkins/blob/master/core/src/main/resources/lib/hudson/setIconSize.jelly.

[16] Jenkins project infrastructure team *LayoutTagLib* [online]. Software in the Public Interest, Inc., 2020. [cit. 11. 5. 2020] Dostupné z https://javadoc.jenkins.io/index.html?lib/LayoutTagLib.html.

[17] Jenkins project infrastructure team *FormTagLib* [online]. Software in the Public Interest, Inc., 2020. [cit. 11. 5. 2020] Dostupné z https://javadoc.jenkins.io/lib/FormTagLib.html.

[18] Jenkins project infrastructure team *Test harness for Jenkins and plugins 2.64 API* [online]. Software in the Public Interest, Inc., 2020. [cit. 11. 5. 2020] Dostupné z https://javadoc.jenkins.io/component/jenkins-test-harness/.

[19] Ferguson Smart, J. et al. *Jenkins: The Definitive Guide*. Sebastopol, CA: O'Reilly Media, 2011. ISBN 978-1-449-30535-2.

[20] CloudBees, Inc. *What is Jenkins?* [online]. CloudBees, Inc., 2019. [cit. 28. 12. 2019] Dostupné z https://www.cloudbees.com/jenkins/what-is-jenkins.

[21] Jenkins project infrastructure team *Jenkins Stats* [online]. Software in the Public Interest, Inc., 2020. [cit. 11. 5. 2020] Dostupné z https://stats.jenkins.io/jenkins-stats/svg/svgs.html.

[22] Kawaguchi, K. *A new chapter for Kohsuke* [online]. Kohsuke Kawaguchi, 2020. [cit. 10. 3. 2020] Dostupné z https://kohsuke.org/2020/01/23/a-new-chapter-for-kohsuke/.

[23] Pražák, O. *Foreman plugin for Jenkins CI.* Diplomová práce. [online]. Brno: Masarykova universita, 2015. [cit. 27. 12. 2019] Dostupné z https://is.muni.cz/th/wdhef/text-final.pdf.

[24] GitHub, Inc. *jenkinsci/jenkins: Jenkins automation server.* [online]. GitHub, Inc., 2019. [cit. 28. 12. 2019] Dostupné z https://github.com/jenkinsci/jenkins.

[25] GitHub, Inc. *ci · GitHub Topics* [online]. GitHub, Inc., 2019. [cit. 28. 12. 2019] Dostupné z https://github.com/topics/ci.

[26] Eclipse Foundation & Oracle *Hudson Continuous Integration* [online]. Eclipse Foundation, 2019. [cit. 28. 12. 2019] Dostupné z http://hudson-ci.org/.

[27] Sten Pittet *Continuous integration vs. continuous delivery vs. continuous deployment* [online]. Atlassian, 2019. [cit. 28. 12. 2019] Dostupné z https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment.

[28] Duvall, P. M., Matyas S. & Glover A. *Continuous integration: improving software quality and reducing risk.* Upper Saddle River, NJ: Addison-Wesley, c2007.

ISBN 978-0-321-33638-5.

[29] Jenkins project infrastructure team *LTS Release Line* [online]. GitHub, Inc., 2019. [cit. 28. 12. 2019] Dostupné z https://jenkins.io/download/lts/.

[30] Jenkins project infrastructure team *Jenkins On Jenkins* [online]. Software in the Public Interest, Inc., 2020. [cit. 21. 4. 2020] Dostupné z https://ci.jenkins-ci.org/.

[31] Jenkins project infrastructure team *Project Governance Document* [online]. GitHub, Inc., 2020. [cit. 21. 4. 2020] Dostupné z https://jenkins.io/project/governance/.

[32] Kohsuke Kawaguchi *QueueTaskFilter* [online]. GitHub, Inc., 2020. [cit. 21. 2. 2020] Dostupné z https://javadoc.jenkins-ci.org/hudson/model/queue/QueueTaskFilter.html.

[33] Jenkins project infrastructure team *Queue.Task* [online]. GitHub, Inc., 2020. [cit. 21. 2. 2020] Dostupné z https://javadoc.jenkins-ci.org/hudson/model/Queue.Task.html.

[34] Jenkins project infrastructure team *AbstractProject* [online]. GitHub, Inc., 2020. [cit. 1. 5. 2020] Dostupné z https://javadoc.jenkins.io/hudson/model/AbstractProject.html.

[35] Kohsuke Kawaguchi *Class Queue* [online]. GitHub, Inc., 2020. [cit. 21. 2. 2020] Dostupné z https://javadoc.jenkins-ci.org/hudson/model/Queue.html.

[36] Kohsuke Kawaguchi *Class Queue* [online]. GitHub, Inc., 2020. [cit. 1. 5. 2020] Dostupné z https://github.com/jenkinsci/jenkins/blob/master/core/src/main/java/hudson/model/Queue.java.

[37] Kohsuke Kawaguchi *Queue Sorter* [online]. GitHub, Inc., 2020. [cit. 2. 5. 2020] Dostupné z https://javadoc.jenkins-ci.org/hudson/model/queue/QueueSorter.html.

[38] Kohsuke Kawaguchi *AbstractQueueSorterImpl* [online]. GitHub, Inc., 2020. [cit. 3. 5. 2020] Dostupné z https://github.com/jenkinsci/jenkins/blob/23242cab56241eeb2ff71f2f63b0979bcd1c64e0/core/src/main/java/hudson/model/queue/AbstractQueueSorterImpl.java.

[39] Kohsuke Kawaguchi *Build Queue Widget* [online]. GitHub, Inc., 2020. [cit. 2. 5. 2020] Dostupné z https://github.com/jenkinsci/jenkins/blob/master/core/src/main/java/jenkins/widgets/BuildQueueWidget.java.

[40] Kohsuke Kawaguchi *jenkins/queue.jelly* [online]. GitHub, Inc., 2020. [cit. 30. 4. 2020] Dostupné z https://github.com/jenkinsci/jenkins/blob/450d87491df16598409d26511bcad02978eaa8bb/core/src/main/resources/lib/hudson/queue.jelly.

[41] Kohsuke Kawaguchi *jenkins/index.groovy* [online]. GitHub, Inc., 2020. [cit. 30. 4. 2020] Dostupné z https://github.com/jenkinsci/jenkins/blob/c9ad878e311e4c4085176f018ffd66f0d9a05c77/core/src/main/resources/jenkins/widgets/BuildQueueWidget/index.groovy.

[42] Kohsuke Kawaguchi *jenkins/ajaxBuildQueue.jelly* [online]. GitHub, Inc., 2020. [cit. 30. 4. 2020] Dostupné z https://github.com/jenkinsci/jenkins/blob/master/core/src/main/resources/hudson/model/View/ajaxBuildQueue.jelly.

[43] GitHub *About GitHub Actions* [online]. GitHub, Inc., 2020. [cit. 4. 1. 2020] Dostupné z https://help.github.com/en/actions/automating-your-workflow-with-github-actions/about-github-actions.

[44] GitHub *Configuring a workflow* [online]. GitHub, Inc., 2020. [cit. 4. 1. 2020] Dostupné z https://help.github.com/en/actions/automating-your-workflow-with-github-actions/configuring-a-workflow.

[45] Ready, J. *Bitbucket Pipelines vs Jenkins Pipeline* [online]. Hackernoon, 2017. [cit. 2. 1. 2020] Dostupné z https://hackernoon.com/bitbucket-pipelines-vs-jenkins-pipeline-f3b7c0e1c198.

[46] Atlassian, Inc. *Parallel steps* [online]. Atlassian, 2020. [cit. 4. 1. 2020] Dostupné z https://confluence.atlassian.com/bitbucket/parallel-steps-946606807.html.

[47] Jenkins project infrastructure team *Jenkins Plugins* [online]. GitHub, Inc., 2019. [cit. 28. 12. 2019] Dostupné z https://plugins.jenkins.io/.

[48] Jenkins project infrastructure team *Installation Trends JSON* [online]. GitHub, Inc., 2020. [cit. 1. 1. 2020] Dostupné z http://stats.jenkins.io/plugin-installation-trend/.

[49] Buckley, P. *no-agent-job-purge-plugin* [online]. GitHub, Inc., 2020. [cit. 3. 1. 2020] Dostupné z https://github.com/jenkinsci/no-agent-job-purge-plugin/blob/master/src/main/java/org/jenkinsci/plugins/ noslavejobpurge/PurgeNoAgentJobs.java.

[50] GitHub, Inc. *Persistent Build Queue* [online]. GitHub, Inc., 2020. [cit. 3. 1. 2020] Dostupné z https://plugins.jenkins.io/purge-build-queue-plugin.

[51] GitHub, Inc. *Persistent Build Queue* [online]. GitHub, Inc., 2020. [cit. 3. 1. 2020] Dostupné z https://plugins.jenkins.io/persistent-build-queue-plugin.

[52] Juránek, V. et al. *Queue cleanup* [online]. GitHub, Inc., 2020. [cit. 3. 1. 2020] Dostupné z https://plugins.jenkins.io/queue-cleanup.

[53] Votýpková, L. et al. *Read-only configurations* [online]. GitHub, Inc., 2020. [cit. 3. 1. 2020] Dostupné z https://plugins.jenkins.io/read-only-configurations.

[54] Boev, J. et al. *Job/Queue/Slaves Monitoring* [online]. GitHub, Inc., 2020. [cit. 4. 1. 2020] Dostupné z https://plugins.jenkins.io/jqs-monitoring.

[55] Shevtsov, A. et al. *Mission Control* [online]. GitHub, Inc., 2020. [cit. 4. 1. 2020] Dostupné z https://github.com/jenkinsci/mission-control-view-plugin.

[56] Shevtsov, A. et al. *Mission Control* [online]. GitHub, Inc., 2020. [cit. 4. 1. 2020] Dostupné z https://github.com/jenkinsci/mission-control-view-plugin.

[57] Gazdag, V. *Jenkins Security Advisory 2019-12-17* [online]. GitHub, Inc., 2020. [cit. 4. 1. 2020] Dostupné z https://jenkins.io/security/advisory/2019-12-17/#SECURITY-1592.

[58] Gazdag, V. *Story of a Hundred Vulnerable Jenkins Plugins* [online]. NCC Group., 2019. [cit. 4. 1. 2020] Dostupné z https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2019/may/story-of-a-hundred-vulnerable-jenkins-plugins/.

[59] Votýpková, L. *Matrix sorter* [online]. Atlassian, 2020. [cit. 23. 4. 2020] Dostupné z https://plugins.jenkins.io/Matrix-sorter-plugin/.

[60] Charson, M. et al. *Jenkins Statistics gatherer Plugin* [online]. GitHub, Inc., 2020. [cit. 1. 1. 2020] Dostupné z https://github.com/jenkinsci/statistics-gatherer-plugin.

[61] Dahanne, A. *Accelerated Build Now Plugin* [online]. GitHub, Inc., 2020. [cit. 1. 1. 2020] Dostupné z https://github.com/jenkinsci/accelerated-build-now-plugin.

[62] Nenashev, O. et al. *Priority Sorter Plugin* [online]. Atlassian, 2020. [cit. 1. 1. 2020] Dostupné z https://wiki.jenkins.io/display/JENKINS/Priority+Sorter+Plugin.

[63] Shautsou, K. *Block queued job plugin* [online]. Atlassian, 2020. [cit. 2. 1. 2020] Dostupné z https://wiki.jenkins.io/display/JENKINS/Block+queued+job+plugin#app-switcher.

[64] Fromm, F. et al. *Build Blocker* [online]. Atlassian, 2020. [cit. 2. 1. 2020] Dostupné z https://plugins.jenkins.io/build-blocker-plugin.

[65] Shatzer, L. et al. *Dependency Queue* [online]. Atlassian, 2020. [cit. 2. 1. 2020] Dostupné z https://plugins.jenkins.io/dependency-queue-plugin.

[66] Moonen, A. *Fast Track Queue Optimizer* [online]. Atlassian, 2020. [cit. 2. 1. 2020] Dostupné z https://plugins.jenkins.io/fast-track.

[67] Xiaojie, Z. *LinuxSuRen* [online]. GitHub, Inc., 2020. [cit. 23. 4. 2020] Dostupné z https://github.com/LinuxSuRen.

[68] Xiaojie, Z. *Multi-branch priority sorter* [online]. GitHub, Inc., 2020. [cit. 23. 4. 2020] Dostupné z https://github.com/jenkinsci/multi-branch-priority-sorter-plugin.

[69] Jenkins project infrastructure team *Extension Points defined in Jenkins Core* [online]. GitHub, Inc., 2020. [cit. 12. 5. 2020] Dostupné z https://www.jenkins.io/doc/developer/extensions/jenkins-core/.

[70] Jenkins project infrastructure team *Basic guide to Jelly usage in Jenkins* [online]. Atlassian, 2020. [cit. 13. 5. 2020] Dostupné z https://wiki.jenkins.io/display/JENKINS/Basic+guide+to+Jelly+usage+in+Jenkins.

[71] Wallace, B. et al. *Requeue Job* [online]. Atlassian, 2020. [cit. 2. 1. 2020] Dostupné z https://plugins.jenkins.io/jobrequeue.

[72] Earl, A. Otradovec, J. *[HOSTING-946] Host request: Simple queue plugin - Jenkins JIRA* [online]. Atlassian, 2020. [cit. 6. 5. 2020] Dostupné z https://issues.jenkins-ci.org/browse/HOSTING-946.

[73] Maurel, T. et al. *Role-based Authorization Strategy* [online]. Atlassian, 2020. [cit. 7. 5. 2020] Dostupné z https://plugins.jenkins.io/role-strategy/.