VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

AUTOMATIZOVANÁ SPRÁVA SOFTWAROVÝCH PROJEKTŮ
AUTOMATED SOFTWARE PROJECT MANAGEMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Dominik Dostál**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Petr Sysel, Ph.D.**
ADVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Student: Dominik Dostál ***ID:*** 174491

Ročník: 3 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Automatizovaná správa softwarových projektů

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se se systémy pro automatizované sestavování softwarových aplikací (např. Apache Ant, Jenkins, Maven). Některé z nich nainstalujte a na několika softwarových projektech porovnejte jejich vlastnosti. Vyberte nejvýhodnější pro správu projektů pod operačním systémem Linux a v programovacím jazyce C/C++. Uvažujte přitom možnost napojení na správce verzí GIT a napojení na správu dokumentů (např. GroupWare, Redmine). V navazující bakalářské práci vytvořte na základě získaných zkušeností ucelenou instalaci zahrnující server pro správu verzí zdrojových souborů, správu softwarových projektů a správu dokumentů.

DOPORUČENÁ LITERATURA:

[1] Duvall, Paul M. Continuous Integration: Improving Software Quality and Reducing Risk. Addison-Wesley, 2007. ISBN 978-0-321-33638-5

[2] Kontinuální integrace - nástroje Jenkins CI vs. Atlassian Bamboo. In Root.cz, 2016. ISSN 1212-8309. Dostupné také na URL https://www.root.cz/clanky/kontinuani-integrace-nastroje-jenkins-ci-vs-atlassian-bamboo [cit. 13.9.2019]

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 31.5.2022

Vedoucí práce: doc. Ing. Petr Sysel, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

V této práci jsou popsány a porovnány vybrané nástroje pro průběžnou integraci (continuous integration - CI) a průběžné nasazování (continuous deployment - CD). Dle připravené metodiky byly porovnány nástroje Jenkins, GitLab, TeamCity a Bamboo. Každý z těchto systému byl nainstalován na čistý systém Ubuntu 20.04 LTE, pak nakonfigurován a otestován na vybraném softwarovem projektu. Důraz byl kladen zejména na kompatibilitu s Linuxovými systémy a jazyky C/C++.

KLÍČOVÁ SLOVA

CI/CD, GitLab, Jenkins, TeamCity, DevOps, Pipeline, Redmine, Kontinuální integrace, Kontinuální nasazování

ABSTRACT

This work describes and compares selected tools for continuous integration (CI) and continuous deployment (CD). Jenkins, GitLab, TeamCity, and Bamboo tools were compared according to the prepared methodology. Each of these systems was installed on a clean Ubuntu 20.04 LTE system, then configured and tested on a selected software project. The focus was placed on compatibility with Linux systems and C/C++ languages.

KEYWORDS

CI/CD, GitLab, Jenkins, TeamCity, DevOps, Pipeline, continual integration, Continual deployment

DOSTÁL, Dominik. *Automatizovaná správa softwarových projektů*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2022, 57 s. Bakalářská práce. Vedoucí práce: doc. Ing. Petr Sysel, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Dominik Dostál
VUT ID autora:	174491
Typ práce:	Bakalářská práce
Akademický rok:	2021/22
Téma závěrečné práce:	Automatizovaná správa softwarových projektů

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce doc. Ing. Petrovi Syslovi Ph.D. za metodickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

Obsah

Seznam obrázků

Úvod

Automatizovaná správa softwarových projektů je potřeba všude tam, kde vznikají komplexnější softwarové projekty, na kterých se podílí více lidí a vzniká tak potřeba tento projekt efektivně řídit. K danému projektu se váže jak dokumentace, tak samotný zdrojový kód, který je třeba testovat, bezpečně nasazovat do produkčního prostředí a v případě chyb v kódu tyto chyby rychle najít a odstranit. Tento proces automatizovaného nasazování kódu se nazývá – Continuous Integration (CI). CI je zpravidla zajištěn prostřednictvím integračního serveru, kde běží nějaký CI nástroj.

V této bakalářské práci porovnám několik nástrojů pro automatizovanou správu softwarových projektů, které jsou vhodné zejména pro použití on-premise a to pod operačním systémem Linux a dokáží pracovat s projekty napsané v C/C++. Každý z nich bude nainstalován a nakonfigurován na samostatném virtuálním serveru a následně bude otestována jeho základní funkcionalita. Vybrané testované CI/CD nástroje jsou Jenkins, GitLab, TeamCity a Bamboo Atlassian.

V první části této práce jsou definované důležité pojmy pro téma kontinuální integrace, jako je definice CI/CD, verzovací nástroje, automatizované testování, různé druhy testů, co je to CI Server, GIT a build.

V druhé návazné kapitole je popsána metodika srovnání jednotlivých CI nástrojů, jsou zde popsány vlastnosti každého vybraného CI nástroje, prerekvizity potřebné pro úspěšnou instalaci, dále je stručně popsaná samotná instalace, konfigurace samotného CI a možnosti rozšíření, či integrace aplikací třetích stran pro každý z těchto systémů. V závěru kapitoly je pak provedeno srovnání, z kterého vyplývá který CI/CD nástroj mi přijde nejvhodnější a dává jeho použití největší smysl.

Třetí část se pak zabývá realizací, tedy samotné instalaci vybraného nástroje, jeho konfiguraci, principu činnosti a integraci s dalšími nástroji jako je systém na správu verzí, nebo systém na správu dokumentace Redmine. V rámci této práce byla provedena i instalace Redmine, která je podrobně popsána v příloze této práce.

Poslední část této práce je věnována závěru, kde jsou shrnuty výsledky této práce, vlastnosti jednotlivých systémů a vybraného řešení.

1 Definice Pojmů

1.1 Kontinuální integrace a kontinuální doručování

Firma RedHat definuje CI/CD (Continuous integration (CI) / Continuous delivery nebo také continuous deployment (CD)) jako metodu vývoje, kdy je zaváděna automatizace do jednotlivých fází procesu vývoje software.

CI/CD předchází problémům, které může integrace nového kódu způsobit operačním a vývojovým týmům.

Konkrétně tedy CI/CD zavádí automatizaci a sledování po celou dobu vývoje aplikací a to od integračních a testovacích fází až po jejich nasazení do produkčního prostředí. Tyto postupy jsou dohromady označovány jako „CI/CD pipeline“ a jsou aplikovány v týmech, které využívají agilní metody vývoje s přístupem DevOps. [1]

1.1.1 Kontinuální integrace

Kontinuální integrace (CI) jako taková je metoda softwarového vývoje, kdy členové vývojářského týmu integrují svůj kód denně – což vede k několika integracím denně. Každá integrace nového kódu je verifikována automaticky, včetně jeho testu, tak aby byly případné problémy nalezeny a opraveny.

Takový přístup vede ke značné redukci chyb při integraci a zvyšuje rychlost vývoje. CI tak snižuje chybovost, redukuje opakující se manuální práci, zvyšuje přehlednost softwarového projektu a umožňuje kdykoliv v čase nasazení funkčního softwarového projektu. [2]

1.1.2 Kontinuální doručování

Kontinuální doručování – Continuous delivery (CD) je rozšířením CI, protože automatizuje proces nasazení nového kódu do produkčního prostředí. I malé změny zdrojového kódu mohou být hned testovány a nasazeny do produkčního prostředí. [3]

1.1.3 Kontinuální nasazování

Kontinuální nasazování – Continuous deployment (CD) je ještě o krok dál než kontinuální doručování, protože tady je každá změna kódu, která projde všemi fázemi testování automaticky nasazena bez zásahu člověka do produkčního prostředí. Jako takové kontinuální nasazování vyžaduje velkou mobilitu vývojářů v případě, že nastane nějaký problém při nasazení nové verze vyvíjeného softwaru. [3]

1.1.4 Plán kontinuální integrace a kontinuální doručování

Plán kontinuální integrace a kontinuálního doručování, dále jako CI/CD pipeline, představuje sérii kroků, které musí být vykonány, aby mohla být dodána nová verze softwaru. Typicky CI/CD pipeline obsahuje tyto kroky[4]:

- Build – Fáze, kdy je aplikace kompilována
- Test – Fáze, kde je zdrojový kód testován
- Realease – Fáze, kde je kód umístěn do repozitáře
- Deploy – Fáze, kde je kód nasazen do produkčního prostředí
- Validation and compliance – Fáze, kdy je kód validován, jestli splňuje nároky např. na bezpečnost.

Obr. 1.1: CI/CD pipeline [1]

Jak je naznačeno na obrázku 1.1, tak CI/CD lze chápat jako spojené procesy kontinuální integrace a kontinuálního doručování, nebo také jako spojení všech tří procesů kontinuální integrace, kontinuálního doručování a kontinuálního nasazování.

1.1.5 CI Server

CI server obsluhuje celý CI systém včetně repositáře a verzovacího nástroje. Vývojářům dále poskytuje přehled o tom, co se s daným softwarovým projektem děje. Server při každém vložení do repositáře spustí sadu instrukcí(pipeline) a otestuje funkčnost a kompatibilitu nového kódu.[5]

1.2 Verzovací systémy

Verzovacími systémy jsou myšleny všechny nástroje, které umožňují zaznamenání historie úprav nad různými soubory, které jsou využívané v procesu softwarového vývoje, zpravidla se jedná o části zdrojového kódu, ale může se jednat například i o prostou dokumentaci.

1.2.1 GIT

Verzovací systém Git vznikl v roce 2005 z iniciativy komunity vývojářů, v čele s Linusem Torvaldsem, která vyvíjí jádro Linuxu. Z Gitu se stal jeden z nejpoužívanějších verzovacích nástrojů vůbec zejména protože je velmi rychlý, je efektivní při práci s velkými projekty a nabízí dobrý systém větvení pro nelineární způsob vývoje.

1.2.2 GitHub

GitHub je na rozdíl od Gitu webová služba, která slouží k podpoře softwarového vývoje. Je zde možné bezplatně uložit repositář. Slouží především pro otevřené (open source) projekty, ale od roku 2019 je zde možné bezplatně ukládat i soukromé repositáře, které šlo ukládat dříve jen po uhrazení poplatku.

GitHub podporuje softwarový vývoj několika funkcemi, mezi které mimo jiné patří například sledování chyb, dokumentace, podpora wiki, poskytuje historii verzí, seznamy úkolů a podporuje i upozornění o provedených změnách.

1.3 Sestavení aplikace

Sestavení aplikace, nebo také sestavení aplikace, je proces, při kterém se zdrojový kód aplikace převádí do funkčního spustitelného celku. Slovo sestavení má dva významy, označuje proces kompilování kódu a pak se také slovem sestavení označuje výsledek tohoto procesu.

Automatizované sestavování kódu aplikací je důležitá součást každého CI. Tento proces může být realizován několika různými nástroji, to, který je vhodné zvolit závisí především na jazyce, v kterém je daná aplikace napsaná. Mezi tyto nástroje patří například Apache Maven a Apache Ant, Make, CMake a mnoho dalších.

1.4 Testování aplikace

Automatizované testování softwarového projektu je nedílnou součástí kontinuální integrace. Kontinuální integrace (CI) umožňuje integrovat jakékoliv změny kódu a testovat funkčnost kódu kdykoliv je potřeba. Pomocí tohoto testování je možné mít pod kontrolou podstatnou část kódu a mít přehled o jeho stavu, tak aby mohl být kdykoliv nasazen do produkčního prostředí. Samotné testování probíhá v několika krocích, které jsou popsány níže. [6]

1.4.1 Jednotkové testy

Unit test, nebo také „Jednotkový test“, je označení pro test dílčích částí aplikace. Za jednotku se dá považovat jakákoliv samostatně testovatelná část kódu – například proměnná, nebo funkce. V objektově orientovaném programování jednotka může být metoda, nebo třída.

1.4.2 Integrační testy

Integrační test následuje po jednotkovém testování a spočívá v kombinování a testování jednotek jako skupinu, podle předdefinovaného testovacího plánu. Ověřuje tedy zdali jednotlivé funkce a moduly fungují dohromady.

1.4.3 Funkcionální testy

Funkční testy zkoumají, zdali všechny implementované funkce, které jsou do aplikace implementovány, fungují správně a že odpovídají požadavkům vývojáře. Testuje tedy vždy funkční části systému.

1.4.4 Systémové testy

Během systémových testů je konečně aplikace testována jako celek, je ověřována, zdali jako celek funguje a je možné danou aplikaci nasadit do produkčního prostředí.

1.5 Architektura Master/Agent

Master/Agent je pojmenování pro model komunikace, kdy jedno zařízení (Master) nepřímo ovládá druhé zařízení (Agent). Jeden Master může mít několik agentů, kterým zpravidla přiděluje úlohy, které mají vykonat. V kontextu této práce Master nechá agenty vykonávat úlohy definované v sadě instrukcí (pipeline).

Agent může být umístěn na stejném zařízení jako Master, ale zpravidla bývá umístěn odděleně od mastera.

1.6 Nástroje pro týmovou spolupráci

Nástroje pro týmovou spolupráci, které se někdy označují také jako „Groupware“, jsou nástroje umožňující snazší komunikaci a sdílení dat mezi několika lidmi, nebo týmy. Takový nástroj umožňuje svým uživatelům jednoduše komunikovat, sdílet dokumenty, instrukce, požadavky a podobně.

Takový nástroj může být implementován jako aplikace, nebo také může být funkční v režimu „Software as a Service“ (SaaS).

Mezi zástupce těchto aplikací patří například JIRA od firmy Attlasian, Redmine a mnoho dalších. [7]

2 Srovnávané nástroje

2.1 Metoda srovnávání

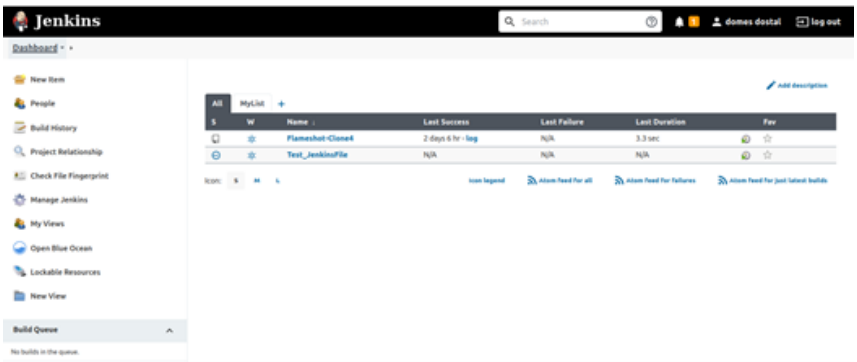
V této kapitole je představena metodika srovnávání CI/CD nástrojů. Každý nástroj, který je v této práci srovnáván, poskytuje on-premise řešení. Řešení, která fungují v cloudu, nebudu srovnávat vůbec.

Pro každý vybraný systém, který lze provozovat on-premise bude připraven nový virtuální server s nainstalovaným Linux systémem Ubuntu 20.04 LTE. Na každém takovém serveru budou provedeny potřebné instalace, aktualizace a nastavení. Jedná se především o nastavení síťových prostupů na určitých portech, nastavení firewallu a případně DNS, URL, PostFix pro email notifikace a konečně nastavení repositářů potřebných pro instalaci. Instalace každého systému bude krátce popsána. Po zprovoznění testovaného nástroje na daném virtuálním serveru bude nástroj testován na jednoduchém projektu. Na daném projektu budou posouzeny možnosti a výstupy z každého CI/CD nástroje. Důraz je kladen zejména na stavy testů, přehlednost, možnosti ladění a konfigurace.

V práci budou uvedeny i příklady konfigurace CI/CD instrukcí. Krátce budou systémy srovnány i z hlediska bezpečnosti, rozšiřitelnosti a integrace s dalšími nástroji jako je například verzovací systém GIT a další.

2.2 Jenkins

Jenkins je asi jeden z nejznámějších CI nástrojů. Na trhu je již dlouho, je zcela zdarma bez žádných poplatků. Je rozšiřitelný mnoha zásuvnými moduly, které mohou měnit jak jeho funkce, podporu programovacích jazyků, tak i jeho uživatelské prostředí, které je v původní podobě celkem nepřehledné. Dříve byl známý také jako



Obr. 2.1: Rozhraní Jenkins (zdroj: Autor)

Hudson, nicméně po právní roztržce s firmou Oracle byl přejmenován na Jenkins.[8]

Projekt Hudson od Oracle již není aktivní. Díky době, kterou je na trhu, se kolem Jenkins vytvořila silná komunita a byla k němu vytvořena bohatá dokumentace. Jenkins je napsán v jazyce JAVA a poprvé byl vydán v roce 2011.

Jenkins je postaven na architektuře Master-Agent, kde komunikace probíhá pomoci TCP/IP protokolu. Jenkins Master se stará o plánování úloh, monitoruje a přiděluje jednotlivým „otrokům“ úlohy. Jenkins Agent přijímá požadavky od Jenkins Mastera na provádění jednotlivých sestavení.

2.2.1 Instalace

Samotná instalace Jenkins vyžaduje mít nainstalovaný balíček Java runtime. V současnosti podporuje verzi Java 8 a Java 11[9]. Instalaci a kontrolu verze lze provést následujícími příkazy [9]:

```
$ sudo apt install openjdk-11-jre-headless
$ java --version
```

Pak lze pokračovat importem GPG klíče a přidáním samotného repositáře `pkg.jenkins.io` nutného pro instalaci:

```
$ wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key
| sudo apt-key add -
```

Přidání jenkins repozitáře a provedení instalace:

```
$ sudo sh -c echo deb http://pkg.jenkins.io/debian-stable binary/
> /etc/apt/sources.list.d/jenkins.list
$ sudo apt update
$ sudo apt install jenkins
```

Kontrola služby a její spuštění

```
$ sudo systemctl status jenkins
$ sudo systemctl start jenkins
```

Po úspěšné instalaci je ještě nutné nastavit prostup na lokálním firewallu na portu 8080/tcp:

```
$ sudo ufw enable
$ sudo ufw allow 8080/tcp
$ sudo ufw reload
$ sudo ufw status
```

Po spuštění, je Jenkins dostupný na localhost:8080. K prvotnímu přihlášení vyžaduje administrátorské heslo, které ukládá na lokální disk.

Pro účely této práce byla provedena instalace pluginu Blue Ocean, který vylepšuje grafické rozhraní. Pro vytvoření CI pipeline byl použit GitHub s naklonovaným repositářem vybraného projektu.

Protože Jenkins ve výchozím nastavení nepodporuje jazyk C a ani C++ bylo nutné doinstalovat kompilátor g++ pro jazyk C++. Jako poslední byl doinstalovaný GIT.

2.2.2 Vytváření pipeline

Pipeline se v Jenkins definuje pomocí `Jenkinsfile.txt`. Jedná se prostý textový soubor, kde je definováno, jak pipeline vypadá – kolik obsahuje kroků, jaká je jejich posloupnost, a co v těchto krocích má vykonat. Jenkinsfile je pak načten při uložení změny do repositáře a sada instrukcí v něm obsažená je pak okamžitě spuštěna.

Jenkinsfile podporuje dvě různé syntaxe, a to buďto tzv. deklarativní, nebo skriptovanou. Deklarativní syntaxe je novější způsob, jak definovat pipeline. Je jednoduší a čitelnější. Deklarativní pipeline je doporučovaný způsob definice.

Ukázka obsahu `jenkinfile`, který definuje dvě fáze(stage) Build a Test s několika příkazy:

```
pipeline {
     agent any
     stages {
    stage('Build'){
    steps {
        sh 'echo "Building..."'
        sh 'chmod +x scripts/Linux-Build.sh'
        sh 'scripts/Linux-Build.sh'

      }
    }
    stage('Test'){
        steps {
          sh 'echo "Testing..."'
          sh 'chmod +x scripts/Linux-Run.sh'
          sh 'scripts/Linux-Run.sh'
                    }
                }
         }
}
```

Protože je `Jenkinsfile` v podstatě textový soubor, lze jej jednoduše vytvořit v jakémkoliv textovém editoru, nicméně ideální jsou takové, které zvýrazňují Groovy syntax. Pro tvorbu konfiguračního souboru lze ale využít i CLI (Command Line Interface) přímo v Jenkins nebo rozšíření Blue Ocean, který nabízí WYSIWYG editor.

2.2.3 Rozšířitelnost

Hlavním způsobem, jak lze rozšířit Jenkins je pomocí zásuvných modulů. Existuje přes tisíc různých zásuvných modulů, které lze nainstalovat a díky kterým lze integrovat různé sestavovací nástroje, cloudové nástroje jako je například Jira, verzovací nástroje jako je Git, RedMine, GitHub a další, nebo pomocí nich lze vylepšit grafické rozhraní, či přímo přidat nějakou jinou funkcionalitu do Jekins.

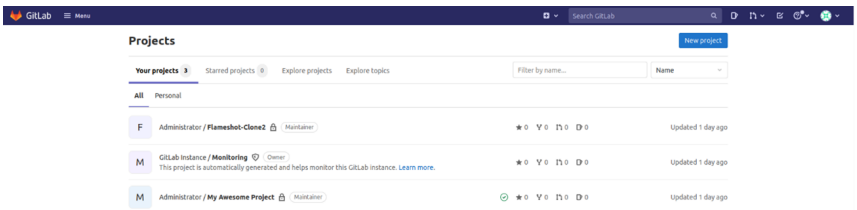
Moduly mohou být automaticky stahovány, například při prvotní konfiguraci si může uživatel zvolit, zda nechá systém automaticky stáhnout doporučené zásuvné moduly, nebo si tyto zásuvné moduly sám vyhledá. V Jenkins je ke správě zásuvných moduly určena sekce centrum aktualizací.

2.2.4 Bezpečnost

Samotný Jenkins má celkem od roku 2011 zdokumentovaných 205 zranitelností. Zatímco zásuvné moduly pro Jenkins 868 zranitelností. Právě velká rozšiřitelnost Jenkins představuje největší bezpečnostní riziko. Zásuvných modulů je velká spousta a zejména ty starší, které nejsou udržované mohou představovat riziko. Jenkins se tyto zranitelnosti snaží řešit, ale záplatování zranitelností je právě na jejich tvůrcích, proto sám přiznává, že bezpečnost rozšíření nemůže garantovat.[10]

2.3 GitLab

GitLab byl původně plně open-source webový nástroj, který byl distribuovaný pod licencí MIT, poprvé byl spuštěn v roce 2011. V roce 2013 byl rozdělen na dvě různé verze – GitLab CE a GitLab EE. GitLab CE zůstal pod licencí MIT, zatímco z GitLab EE se stal proprietární software. GitLab poskytuje gitový repozitář, wiki,



Obr. 2.2: Rozhraní GitLab (zdroj: Autor)

podporuje CI/CD a také projektový management. Můžeme zde vytvářet úkoly, ty pak delegovat na jednotlivé uživatele, přidělovat jim štítky a podle nich je následně třídit. Pro přehled nad těmito úkoly GitLab nabízí tzv. Board.

GitLab lze distribuovat jako on-premise řešení, ale lze jej provozovat i v cloudu jako SaaS. V této práci bude testován oficiální kontejner pro on-premise řešení, které obsahuje již všechny potřebné závislosti jako je nginx, PostgreSQL, Redis a další. Této oficiální distribuci se říká GitLab Omnibus.[12]

GitLab oficiálně podporuje několik možností jak systém instalovat, jako jsou kontejnery pro Windows, různé distribuce Linux (CentOS7, Debian..), tak i třeba helm (balíčkovací aplikace) pro Kubernetes (orchestrátor kontejnerů). Cloudové řešení je podporováno u Azure, AWS, Google Cloud platform a DigitalOcean. [14]

2.3.1 Instalace GitLab Omnibus

Instalace GitLab spočívá pouze v přidání vlastního repozitáře a instalaci balíčku. Při instalaci je třeba definovat URL a v případě potřeby lze definovat i administrátorské heslo. Je-li ponechána přednastavená volba, tak se heslo uloží na lokální disk.

Dále je možné instalovat PostFix pro emailové notifikace. Nicméně volba konkrétního řešení emailových notifikací závisí plně na správci serveru.

Příkaz pro přidání repozitáře a instalaci GitLab[13] [14]:

```
$ curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ee/script.deb.sh | sudo bash
$ sudo EXTERNAL_URL="https://gitlab.example.com" apt-get install gitlab-ee
```

2.3.2 GitLab Agent

GitLab Agent, dle oficiální dokumentace také jako GitLab Runner, je aplikace, která obsluhuje GitLab CI/CD a spouští úkony definované v pipeline. Runner lze nainstalovat na jiném počítači, než běží samotný GitLab, což je výhodné zejména z hlediska výkonu a bezpečnosti. [15]

GitLab runner podporuje různé operační systémy a může být spuštěn i v Docker. Stažen a nainstalován může být několika způsoby, a to buďto jako kontejner do Dockeru, jako binární instalace, nebo stažen z repositáře.

Po úspěšné instalaci runneru je nutné runner registrovat v GitLab, což není nic jiného než ustanovení komunikace mezi instancí GitLabu a runnerem. Dostupných runnerů může být více. To, jaký runner má být použit pro spouštění dané pipeline pak záleží na volbě uživatele.

2.3.3 Konfigurace CI

Sada instrukcí je základním stavebním kamenem pro CI/CD v GitLabu. Sada instrukcí bývá zpravidla definována v `.gitlab-ci.yml` souboru. Příklad toho, jak

takový soubor může vypadat je níže:

```
stages:
    - build
    - test
build:
    stage: build
    script:
        - echo "Building"
        - mkdir build
        - touch build/info.txt
test:
    stage:  test
    script:
        - echo "Testing"
        - test -f "build/info.txt"
```

Tento `.gitlab-ci.yml` bývá umístěn v kořenovém adresáři projektu a sada instrukcí z tohoto souboru spuštěna pokaždé, když je provedena změna v repositáři. Po změně v repositáři je spuštěn GitLab runner, který provede sérii kroků definovaných v tomto souboru.

Dle oficiální dokumentace jsou tři způsoby, jak tvořit konfigurační soubor [16]:

- Basic: vhodné pro jednoduší projekty, kde všechny konfigurační soubory jsou na jednom místě
- Directed Acyclic Graph: Dobré pro velké projekty
- Child/Parent Pipelines: Vhodné pro monorepozitáře

Tato práce se bude zabývat pouze základní konfigurací CI v GitLabu.

2.3.4 Rozšířitelnost

Funkce GitLabu nelze přímo rozšířit o nějakou novou funkcionalitu, nebo o nějaký grafický prvek jako je tomu například u systému Jenkins. GitLab lze pouze integrovat s externími aplikacemi, jako je právě například Jira, nebo Slack, pomocí tzv. Webhooku, nebo přídavných modulů. [17][18]

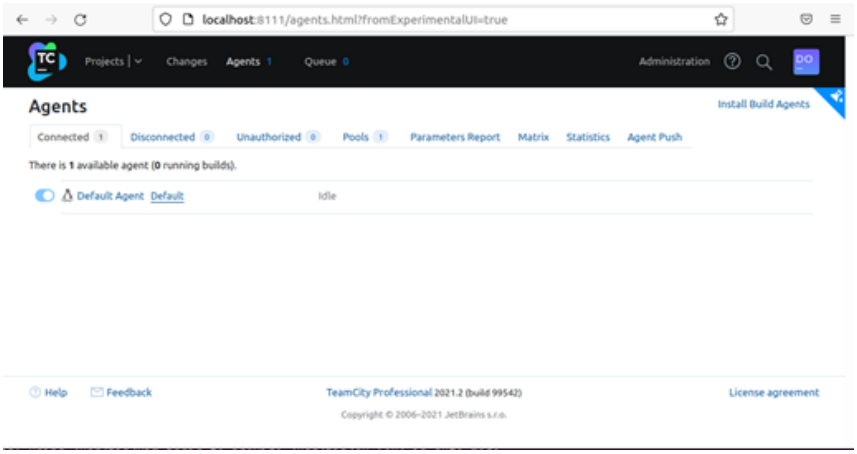
2.3.5 Bezpečnost

Od roku 2014 bylo zdokumentováno v nástroji GitLab celkem 678 zranitelností.[19] GitLab zranitelnosti ošetřuje několika různými způsoby a to buďto v naplánovaných záplatách, v měsíčních záplatách, nebo pouští ad-hoc záplaty v případě, že je zdokumentována nějaká kritická zranitelnost.

Bezpečnostní tým v případě potřeby využívá „Internal Security Notification Dashboard“ prostřednictvím tohoto kanálu informuje o zranitelnostech a kritických událostech. [20]

2.4 TeamCity

TeamCity je CI/CD nástroj od firmy JetBrains, která jej vyvinula a nadále jej spravuje. Jedná se o multiplatformní, komerční systém, který je však za dodržení určitých podmínek zdarma.



Obr. 2.3: Rozhraní TeamCity (zdroj: Autor)

Systém je nabízen jako SaaS – TeamCity cloud, nebo jako on-premise řešení, a to v prakticky ve dvou režimech. Buďto jako licence Professional, která je zdarma, obsahuje 3 sestavovací agenty a přístup ke všem funkcím, nebo je nabízen jako Enterprise licence, jejíž cena je závislá na počtu buildovacích agentů. [22]

První vydání TeamCity je z roku 2006, podporuje celou řadu verzovacích systému, mezi které patří Git, Mercurial, nebo třeba Subversion, ale i mnoho dalších. [23] Bezplatná verze on-premise poskytuje až 100 sestavení a 3 sestavovací agenty.

V nástroji TeamCity je agent označení pro aplikaci, která obsluhuje sestavovací procesy a testy. Agent je nainstalován odděleně od serverové části a může běžet na úplně jiném počítači, na jiném operačním systému, právě podle toho, v jakém prostředí je nutné provést sestavení a test. [24]

TeamCity Server je na druhé straně část řešení, která se nestará o žádný proces. Účelem serveru je především monitorovat všechny připojené agenty, zaznamenat události a výsledky testů. Všechny informace o úkolech, testech, buildech uživatelích a jejích oprávněních ukládá do databáze.

2.4.1 Instalace

Podobně jako u CI nástroje Jenkins je třeba mít nainstalovaný Java Development Kit, navíc je ale potřeba nějaké databázové úložiště, v této práci jsem na serveru instaloval mysql server.

V dalších krocích je pak instalace docela jednoduchá a spočívá pouze ve stažení zabaleného TeamCity, které je třeba vybalit do složky na lokálním disku a následně z něj spustit skript `runAll.sh`. Tento skript, pak provede instalaci. [21]

Instalace a kontrola JAVA:

```
$ sudo apt-get install default-jdk -y
$ java -version
```

Instalace MySQL:

```
$ sudo apt-get install mysql-server
```

Stáhnutí balíčku s TeamCity:

```
$ wget https://download.jetbrains.com/teamcity/TeamCity-2021.2.tar.gz
$ tar -xzf TeamCity-2021.2.tar.gz
```

Spuštění služby Teamcity:

```
$ ./TeamCity/bin/runAll.sh start
```

TeamCity je automaticky spuštěno na localhost, kde je uživatel v několika krocích proveden základním nastavením – jako je konfigurace databáze a další. TeamCity pak už veškerou konfiguraci provede automaticky a uživatele nakonec vybídne k prvotnímu přihlášení účtem admin. Čerstvě nainstalovaný server TeamCity obsahuje jednoho výchozího agenta a ve výchozí konfiguraci podporuje propojení s GitHub, GitLab, Bitbucket, Space a Azure.

2.4.2 Konfigurace CI

Každá instalace Teamcity obsahuje kořenový projekt, do kterého se pak vkládají všechny ostatní projekty. Tyto projekty mohou být vytvořeny v podstatě dvěma způsoby, a to buďto manuálně anebo mohou být nahrány z repositáře.

Na rozdíl od GitLab a Jenkins se v TeamCity užívá jiné názvosloví. Sada instrukcí je „build chain“ [31], kde build, tedy sestavení představuje jeden krok v sadě instrukcí, v prostředí Teamcity se tento krok nazývá také stage. Nejjednodušší projekty si tedy vystačí s jedním sestavením, zatímco ty složitější budou obsahovat celý „build chain“.

Sada instrukcí, neboli build chain, obsluhuje agent, který překontroluje zdrojový kód, stáhne záznam z předešlých sestavení, který se nazývá „artifact“. Agent obsluhuje vždy jen jedno sestavení. Záznam sestavení(artifact) je soubor, který je vytvořen během sestavení a jsou do něj uloženy informace jako war files, logy, reporty a další.

Prvním krokem při vytváření sady instrukcí, nebo „build chain“, je vytvoření projektu, který můžeme vytvářet manuálně anebo automaticky. Automatickým vytvořením projektu je myšleno nahrání projektu z nějakého repositáře, jako je třeba GitHub, nebo GitLab. Po úspěšném vložení projektu, proběhne automatická detekce jednotlivých kroků sady instrukcí. Každý krok neboli „build step“ můžeme definovat i manuálně s využitím grafického prostředí. Alternativně lze sadu instrukcí spravovat jako kód.

„Build chain“ je napsán v jazyce Kotlin a ukázka takového kódu je níže:

```
object TodoAppNoChainKts_TodoApp : BuildType({
    id = AbsoluteId("TodoAppNoChainKts_TodoApp")
    name = "TodoApp"

    artifactRules = "build/libs/todo.jar"
    vcs {
        root(TodoAppNoChainKts_TodoBackendVcs, "-:docker")

        cleanCheckout = true
    }
steps {
    gradle {
        tasks = "clean build"
    }
}
    triggers {
        finishBuildTrigger {
            buildType = "${TodoAppNoChainKts_TodoImage.id}"
        }
    }
}
```

2.4.3 Rozšířitelnost

TeamCity lze rozšířit několika způsoby, prvním a nejpřirozenějším z nich je pomocí zásuvných modulů(pluginů). Ty jsou dostupné přímo z rozhraní administrátorské konzole TeamCity. Tyto zásuvné moduly jsou buďto vyvíjené a poskytované firmou JetBrains anebo pochází od třetích stran, které jsou s otevřeným zdrojovým kódem. Všechny tyto zásuvné moduly poskytují integraci s nějakým nástrojem jako je například Eclipse, nebo například integraci se správcem verzí(VCS).

Další možností, jak TeamCity přizpůsobit vlastním potřebám je pomocí REST API [32], ale i to slouží spíše pro integraci s jiným nástrojem. Upravit přímo funkcionalitu, či vzhled jako nabízí například Jenkins nelze.

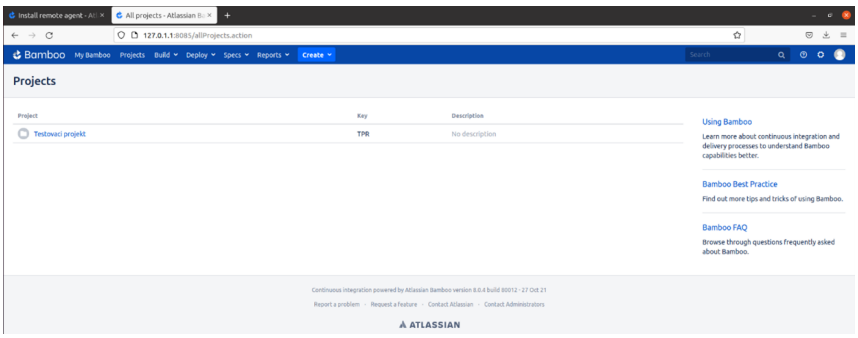
2.4.4 Bezpečnost

Počet zdokumentovaných zranitelností Teamcity od roku 2015 je 92.[25] TeamCity nabízí různé bezpečnostní prvky, které jsou ze strany firmy JetBrains doporučována týkající se zejména řízení přístupu, práv a rolí.[26].

Teamcity také podporuje statickou analýzu kódu, která analyzuje kód a dokáže identifikovat běžné chyby a zranitelnosti testovaného kódu.[27]

2.5 Bamboo Atlassian

Bamboo je multiplatformní řešení pro CI/CD server, který je napsán v jazyce JAVA a je vyvíjen a spravován firmou Atlassian. Distribuován byl dříve jako cloud řešení, řešení pro servery a jako řešení do datacenter. [33] V současnosti však bylo od licencí pro cloud a samostatný server upuštěno a je podporována pouze licence pro datacentra. Verze datacenter je samostatně spravovatelná a na rozdíl od Bamboo serveru poskytuje několik dalších funkcionalit jako je rozšířená správa repositářů, podpora clusteringu a zlepšení dostupnosti a správy. [34]



Obr. 2.4: Rozhraní Bamboo (zdroj: Autor)

Pro nové uživatele je dostupná zkušební licence na 90 dní, přičemž výrobce tvrdí, že poskytuje pro akademickou obec licence zdarma.

Bamboo umožňuje ve výchozím nastavení integraci s verzovacími nástroji jako je BitBucket, Git, GitHub, Subversion, Mercurial a CVS. Přirozeně také lze integrovat s ostatními nástroji od firmy Atlassian jako jsou nástroje pro efektivní projektové řízení JIRA, Confluence, BitBucket čímž je použitelnost v řízení softwarového vývoje rozšířena.

2.5.1 Instalace

Instalace na Linux systému spočívá ve stažení .tar balíku s instalaci z oficiálních stránek výrobce. Následně je nutné zajistit stažení kompatibilní verze JAVA SDK

následujícími příkazy, Bamboo podporuje JAVA 11 a JAVA 8 [28]:

```
$ sudo add-apt-repository ppa:openjdk-r/ppa
$ sudo apt-get update
$ sudo apt-get install openjdk-8-jdk
```

Pro samotnou instalaci je třeba vytvořit složku s vhodným umístěním a balík s Bamboo zde rozbalit [29]:

```
$ sudo cd /opt
$ wget http://www.atlassian.com/software/bamboo/downloads/binary/
    atlassian-bamboo-6.10.4.tar.gz
$ sudo tar -xvf atlassian-bamboo-8.0.4.tar.gz
```

Dále je třeba upravit několik konfiguračních souborů, vytvořit aplikační účet pro Bamboo, vytvořit domovskou složku pro tento účet a v konfiguračních souborech tuto cestu do domovské složky definovat. Zde je třeba odkomentovat `bamboo.home` a definovat domovskou složku např. :

```
$ sudo vim /opt/bamboo/atlassian-bamboo/WEB-INF/classes/bamboo-init.
    properties
bamboo.home=/home/bamboo/bamboo-home
$ sudo mkdir -p /home/bamboo/bamboo-home
```

Po těchto krocích můžeme Bamboo spustit ze složky `bin`. Je jen na nás, jestli jej budeme spouštět jako službu v popředí, či ji necháme v pozadí. Bamboo je ve výchozím nastavení dostupný na localhost:8085 a hned po přistoupení na tuto adresu je uživatel vyzván k zadání licenčního kódu. Licenci lze získat na oficiálním webu výrobce, zdarma je licence dostupná na 90 dní.

Po zadání licence pak následuje konfigurace samotné serverové instance, databázového systému (PSQL, MySQL..), pro úspěšné dokončení tohoto kroku je třeba nainstalovat a nakonfigurovat příslušnou databázi – pro účel testování byla stažena a nakonfigurována databáze PSQL[30]:

```
$ sudo apt-get install postgresql postgresql-contrib
$ sudo -s -H -u postgres
$ /opt/PostgreSQL/8.3/bin/createuser -S -d -r -P -E bamboouser
$ /opt/PostgreSQL/8.3/bin/createdb -O bamboouser bamboo
$ exit
```

V posledním kroku pak už jen stačí vytvořit administrátorský účet pro správu. Posléze je uživatel již připuštěn přímo do Bamboo, kde je hned jako první upozorněn na nutnost instalace „remote agenta“. Po stažení „remote agenta“ jej lze jedním příkazem nainstalovat, pak jej stačí povolit a vše je připraveno k použití.

2.5.2 Konfigurace CI

Kontinuální integrace začíná vytvořením projektu, kterému se přiřadí tzv. plan. Plan se skládá z několika fází, přičemž každá z těchto fází může obsahovat několik úloh, zpravidla však obsahuje vždy jednu úlohu. Bamboo, pokud těchto úloh je víc, může spouštět více úloh paralelně. Úlohy lze volit z několika kategorii, kde jsou předdefinované různé kategorie a typy pro sestavení, testování, nasazení, správu zdrojových souborů a další.

Pro tyto plány se pak definují výchozí repositáře, podmínky, za kterých je plán spuštěn, dále pak uživatelské přístupy či systémové proměnné. Plán může být nastaven tak, aby se automaticky spouštěl v určitý čas, za určitých podmínek jako je provedení importu do repositáře nebo na základě výstupu jiného plánu. Bamboo dále umožňuje zasílání upozornění, například prostřednictvím emailu, nebo i jiných aplikací jako je třeba Slack.

2.5.3 Rozšířitelnost

Bamboo nabízí dvě cesty, jak rozšířit funkcionality. Jedna cesta, jak rozšířit funkce CI serveru je prostřednictvím zásuvných modulů, které jsou dostupné v Atlassian Marketplace a bývají zdarma. Další cesta pak vede přes vývoj vlastního rozšíření, a to buďto prostřednictvím REST API, které je vhodné zejména pokud nám jde o integraci s jiným nástrojem. Pokud však chceme přidat vlastní funkcionalitu přímo do systému Bamboo je nutné vyvinout plugin prostřednictvím plugin SDK. [28]

2.5.4 Bezpečnost

Bamboo Atlassian od roku 2012 má zdokumentovaných 18 zranitelností, což je ze všech porovnávaných CI/CD nástrojů nejméně. [36]

V dokumentaci k Bamboo je k oblasti security několik doporučení, jako například omezení počtu administrátorů, jak správně administrovat přístup, či jak zabezpečit agenty v architektuře Master/Agent. [38]

Bamboo mimo jiné zveřejňuje „Security Advisory“ kde zveřejňuje produkty a jejich zranitelnosti, které se podařilo zdokumentovat. [39]

2.6 Srovnání

V kapitole srovnání jsou shrnuty důležité vlastnosti srovnávaných CI/CD nástrojů do následující tabulky 4.1. Uživatelské prostředí, přehlednost pipeline a složitost instalace jsou hodnoceny subjektivně, kdy číslo 1 představuje nejlepší možnou hodnotu a číslo 4 tu nejhorší.

Tab. 2.1: Srovnání testovaných CI/CD nástrojů

	Jenkins	**GitLab**	**TeamCity**	**Bamboo**
distribuce	on-premise	cloud, on-premise	cloud, on-premise	on-premise
licence	zdarma	placené, zdarma je pouze CE	placené, zdarma pouze za určitých podmínek	placené, zdarma za určitých podmínek
Přehlednost Pipeline	3	1	4	2
Uživatelské prostředí	4	1	3	2
Rozšiřitelnost	ANO	NE	NE	ANO
Integrace	pomocí rozšíření	pomoci pluginu	pomoci pluginů	vestavěné, pomocí pluginů
Složitost instalace	2	1	3	4

Všechny porovnávané systémy lze provozovat on-premise, přičemž GitLab a TeamCity je dostupné i jako SaaS. Jediný Jenkins je plně zdarma, všechny ostatní řešení jsou nabízena jako placené, přičemž za určitých podmínek je možné je provozovat zdarma. GitLab poskytuje zdarma Community edition, která je neomezeně k použití, ale některé jeho funkcionality jsou omezené. Proto je takto vhodný spíše pro menší projekty.

Uživatelské prostředí a přehlednost pipeline je nejlépe hodnocena u GitLabu, zatímco nejhůře u Jenkins, který je bez rozšíření nepřehledný a má zastaralé uživatelské prostředí.

Rozšiřitelné v pravém slova smyslu je pouze Jenkins. K dispozici je více než 1000 různých rozšíření, která řeší různé problémy, nicméně velká část z nich není udržovaná, a proto představují kvůli zastarávání závislostí bezpečnostní riziko. Všechny ostatní řešení poskytují zásuvné moduly spíše jako prostředek pro integraci dalších nástrojů jako je například JIRA, nebo různé VCS.

Zaručeně nejjednodušší instalaci měl GitLab na oficiální stránce je dostupná instalace se všemi potřebnými závislostmi, zatímco u všech ostatní bylo třeba řešit prerekvizity, jako je například JAVA a další úkony, jako vytváření aplikačního účtu pro Bamboo, nastavování oprávnění a další administrátorská nastavení.

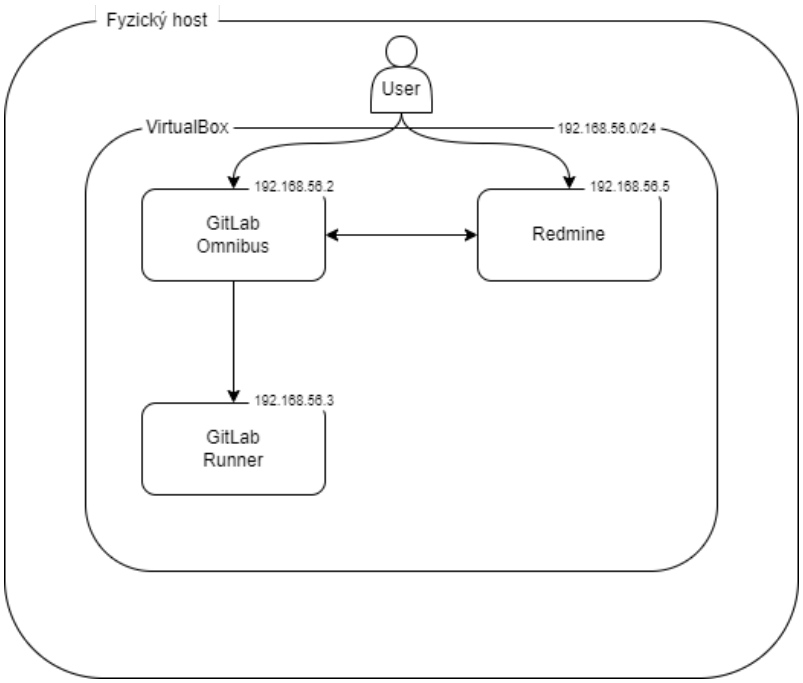
Ze srovnaných nástrojů je nejlépe hodnocen GitLab, o něco hůř pak Jenkins a Bamboo a jako nejméně vhodný nástroj TeamCity. Jenkins a Gitlab jsou nejlépe hodnoceny, především protože nabízí řešení, které je plně zdarma, které může být v případě Jenkins velmi univerzální, avšak s nepřehledným UI.

GitLab oproti Jenkins má dobré UI, je přehledné a nabízí dobré CI. Je distribuován jako kontejner se všemi potřebnými závislostmi, takže poskytuje i jednoduchou instalaci.

3 Ucelená instalace GitLab

V této kapitole bude podrobněji rozvedena instalace a konfigurace nástroje pro kontinuální integraci a nasazování GitLab. Tento nástroj byl zvolen v návaznosti na kapitolu 2.6. GitLab bude nainstalován na linuxovém serveru Ubuntu 20.04 LTE, který je stažen z oficiálních stránek distribuce a bude spuštěn ve virtualizačním nástroji VirtualBox.

Na jiném virtuálním serveru nainstaluji GitLab Runner, mezi oběma stroji vytvořím prostup, a GitLab Runner potom zaregistruji v samotném nástroji GitLab. Nakonec implementuji nástroj pro zprávu dokumentace a verzovací nástroj GitHub.



Obr. 3.1: Návrh zapojení ve virtuální síti

3.1 GitLab

Dle oficiální dokumentace GitLab podporuje několik různých distribucí Linuxu. Pro Ubuntu Server 20.04 LTE jsou uvedeny tyto minimální hardwarové požadavky [40]:

- 4 procesorová jádra
- 2.5 GB místa na pevném disku
- 4 GB RAM

Po vytvoření virtuálního stroje a nainstalování operačního systému je potřeba provést prvotní konfiguraci, tak abychom byli schopni server spravovat prostřednic-

tvím SSH. V případě VirtualBoxu je třeba ještě navíc provést přesměrování portů, tak, aby komunikace ať už po SSH, http či https, mezi hostem a hostitelským počítačem probíhala a bylo možné k serveru přistupovat jinak než přes konzoly na úrovni virtualizačního nástroje.

Po přihlášení na server provedeme následující:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Dále provedeme instalaci balíčku pro SSH a tento balíček spustíme:

```
$ sudo apt-get install ssh
$ sudo systemctl start ssh
```

Abychom se na server mohli prostřednictvím SSH na portu 22 přihlásit je potřeba povolit prostup ve firewallu, pak provedeme restart serveru:

```
$ sudo ufw ssh allow
$ sudo reboot
```

3.2 Instalace GitLab

Jak již bylo zmíněno v odstavci 2.3.1, tak samotná instalace GitLabu spočívá v přidání repositáře a nainstalování balíčku do systému. Dle dokumentace je vhodné nainstalovat tyto závislosti:

```
$ sudo apt-get install -y curl openssh-server ca-certificates tzdata
   perl
$ sudo apt-get install -y postfix
```

Distribuce GitLab Omnibus si všechny další potřebné prerekvizity, jako je např. PostGreSQL nese sebou, takže není třeba už nic dalšího instalovat.[40]

Pro instalaci balíčku GitLab provedeme tedy následující:

```
$ curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-
   ee/script.deb.sh | sudo bash
$ sudo EXTERNAL_URL="VM IP adresa" apt-get install gitlab-ee
```

Pokud požadujeme používat pro přístup k systému GitLab URL (což je doručovaná varianta) je možné výchozí URL změnit hned v úvodu instalace. V případě instalace ve VirtualBoxu je potřeba místo URL zadat IP adresu serveru. URL můžeme kdykoliv, kdy je potřeba, změnit `v/etc/gitlab/gitlab.rb` například pomocí editoru Vim. Aby bylo URL dostupné je potřeba ještě nastavit prostup na firewallu:

```
$ sudo ufw allow 80
$ sudo ufw allow 443
```

Po úspěšné instalaci, nastavení firewallu je nutné prvotní přihlášení pod účtem `root` s přístupovým heslem, které je po instalaci vytvořeno a uloženo v `/etc/gitlab/initial root password`.

3.3 Instalace GitLab runneru

Oficiální dokumentace nemá žádná konkrétní doporučení ohledně hardwarových prostředků, které GitLab agent vyžaduje. Potřebné hardwarové požadavky závisí na konkrétní konfiguraci agenta a dalších nástrojů, které jsou potřeba pro běh aplikace v CI prostředí.

Jak již bylo zmíněno v 2.3.2, GitLab agent může běžet na stejném serveru jako samotný GitLab, ale výhodnější varianta je agenta nainstalovat na jiném počítači a to zejména s ohledem na bezpečnost a výkon.

V rámci této práci bude využit operační systém Ubuntu 20.04 LTE a tato hardwarová konfigurace:

- 1 vCPU
- 3.5 GB RAM

Instalace agenta může být provedena několika různými způsoby. Můžeme GitLab agenta stáhnout jako binární soubor a pak jej instalovat, nebo si stáhneme balíček s příslušným agentem.

Příkaz pro stažení příslušného balíčku a jeho nainstalování:

```
$ curl -LJO "https://gitlab-runner-downloads.s3.amazonaws.com/latest/
    deb/gitlab-runner_${arch}.deb"
$ dpkg -i gitlab-runner_<arch>.deb
```

Po úspěšné instalaci spustíme GitLab agenta:

```
$ sudo gitlab-runner start
```

A pak jej potřebujeme registrovat. Při registraci je potřeba použít registrační token vygenerovaný v Gitlabu. Registrace se provede tímto příkazem:

```
$ sudo gitlab-runner register
```

Registrace a instalace GitLabu runneru může být jiná v závislosti na operačním systému a druhu runneru (exekutoru).[15]

3.3.1 Gitlab Exekutor

GitLab agent provádí definované úlohy v sadě instrukcí prostřednictvím exekutoru. Exekutorů je několik druhů:

- SSH
- Shell

- Parallels
- VirtualBox
- Docker
- Docker Machine (auto-scaling)
- Kubernetes
- Custom

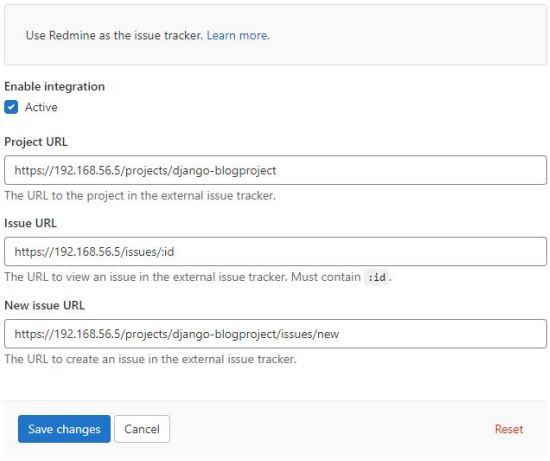
Pokud bychom zvolili VirtualBox jako exekutora, pak by se GitLab runner integroval nainstalovaným VirtualBoxem a spustil úlohy CI na virtuálním počítači, kdybychom vybrali kubernetes, pak budou úlohy definované v CI spouštěny v kubernetes clusteru, v případě ssh jsou úlohy delegovány na vzdálený server. [41]

3.4 Integrace

GitLab lze integrovat s mnoha nástroji prostřednictvím zásuvných modulů. Jejich integrace je celkem přímočará a v rámci této práce bude řešena pouze integrace s nástrojem projektového řízení Redmine. Který bude instalován na samostatném virtuálním serveru. Popis instalace Redmine je uveden v příloze A.

Důležitým předpokladem úspěšné integrace je povolení odchozích spojení z GitLabu, které se provede v nastavení sítě v sekci „Outbound Requests“.

Integrace jako taková je pak velmi jednoduchá. Možnost integrace Redmine je přímo zabudována do rozhraní GitLab a stačí nastavit správné URL na Redmine a potvrdit.



Obr. 3.2: Integrace Redmine (Zdroj:Autor)

Po úspěšné integraci je Redmine dostupný proklikem z levého menu.

3.5 Princip práce

Základem práce v GitLab projektu je `.yaml` soubor, v kterém je definována posloupnost kroků, které má pipeline vykonat, tak jak bylo již znázorněno v 2.3.3.

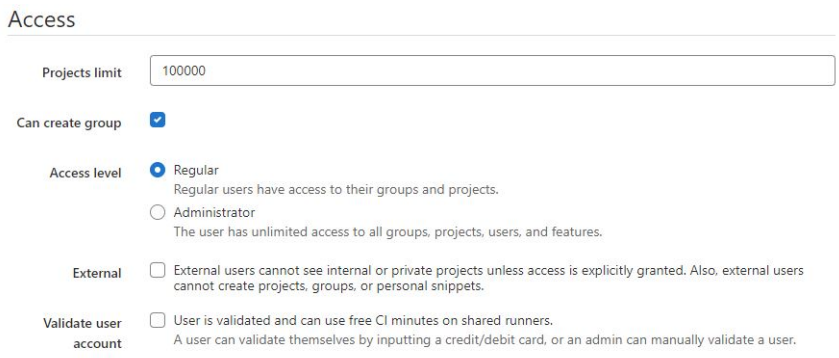
V případě, že se uživatel rozhodne do nějakého projektu integrovat část kódu vytvoří se nová větev projektu (branch) do které jsou nové části kódu integrovány a spustí se proces kontinuální integrace. Následně je vytvořen build a tento build je otestován, tak jak je definováno v `.yaml`.

Jestliže je nějaký test neúspěšný, tak je celý proces zastaven a po opravě chyby je pipeline spuštěna tam kde skončila. Po úspěšném otestování, může být provedeno spojení (merge) s hlavní větví a zahájen proces kontinuálního nasazování.

V rámci této kapitoly je dále popsáno jak řídit přístup k projektům, jak tyto projekty importovat a jak spravovat sady instrukcí a výsledky testů.

3.5.1 Řízení přístupu

GitLab řídí oprávnění na několika úrovních a to na úrovni uživatele, projektu a skupin uživatelů. Přičemž na úrovni uživatele lze přidělit úroveň oprávnění „Regular“ nebo „Administrator“. Administrátor vidí vše a může provést jakoukoliv akci. Regulární uživatel má pak přístup jen k prostředkům, které jsou mu přiděleny s omezenými právy. Uživatele můžeme označit i jako externího, takže nebude mít přístup k interním, nebo soukromým projektům.

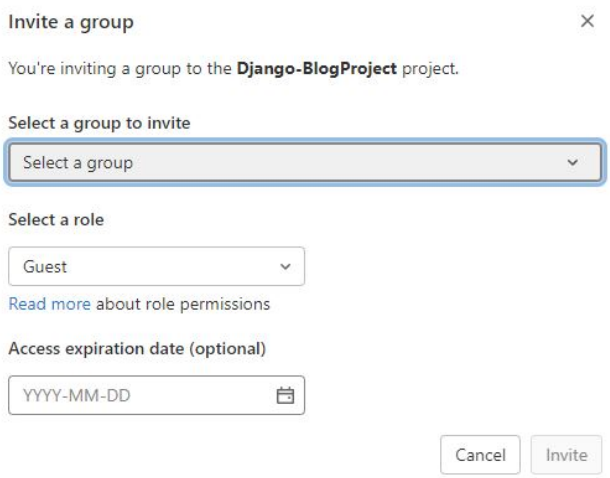


Obr. 3.3: Nastavení práv na úrovni uživatele (zdroj: Autor)

Na úrovni skupiny se pak oprávnění nastavují zvlášť. Rolí je zde několik a práva, která každé roli přísluší je celá řada. Role se stručným popisem jsou následující:

- Guest - Především práva čtení, může provést pull
- Reporter - Role má právo evidovat a řešit požadavky a problémy v rámci projektu

- Developer - Developer má práva spouštět sadu instrukcí, vytvářet nové větve, vytvářet požadavky o spojení větví a další
- Maintainer - Určený pro správu projektu, schvalování žádosti o spojení větví
- Owner - Má plná práva v rámci projektu

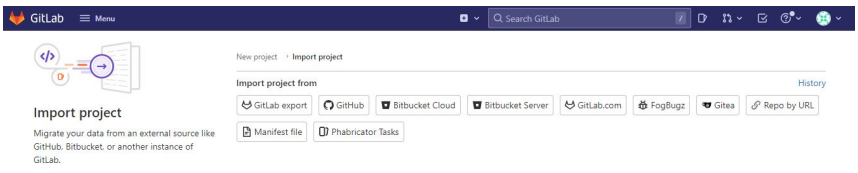


Obr. 3.4: Nastavení práv pro skupinu na úrovni projektu (zdroj: Autor)

Práva se dají spravovat i na úrovni samotných projektů. K projektům lze přidělovat jak samotné uživatele a přidělovat jim role, které již byly popsány výše s výjimkou role Owner. Obdobně to pak funguje i pro skupiny, kdy k projektu lze přidělit skupinu uživatelů a dát skupině možnost přidělit maximální úroveň oprávnění, kterou mohou členové skupiny mít.

3.5.2 Správa projektů

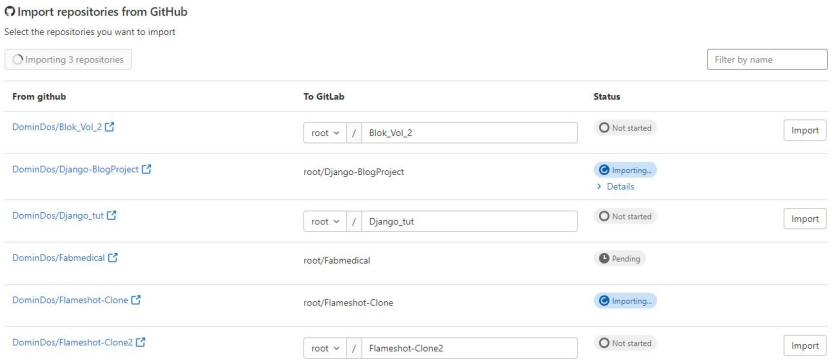
GitLab nabízí několik možností jak založit projekt. Projekt může být založen jako prázdný, podle již definované šablony, nebo projekt lze importovat z nějakého externího zdroje jako je například GitHub, Git, Bitbucket apod.



Obr. 3.5: Import projektu (zdroj: Autor)

V rámci této práce jsou projekty naimportovány z nástroje GitHub repositáře. Import z GitHub je docela přímočarý. V podstatě je potřeba mít pouze přístupový

token k repositáři, který na straně GitHubu je potřeba vygenerovat. Po vložení do GitLab pak lze vybrat z několika přítomných projektů, které v repositáři jsou a ty jednoduchým výběrem importovat do GitLab.



Obr. 3.6: Importování vybraných projektů (zdroj: Autor)

Po úspěšném importu jsou pak projekty přístupné ze záložky „Projects“. Jako vlastník projektů je automaticky nastaven uživatel, který provedl import daného projektu.

3.5.3 Správa repositáře

Správa repositářů v GitLab je přímo svázaná s rolí uživatele. V názvosloví Gitlab se hlavní repositář nazývá „Master“ a všechny, vedlejší repositáře potom „Branches“. Uživatel s rolí vývojáře nemá právo jakkoliv zasahovat do hlavní větve repositáře, tedy do „masteru“. Může jedině vytvořit vedlejší větev, kde může provádět změny a integrovat nové části softwarového projektu. Pokud vývojář usoudí, že je čas tuto vedlejší větev spojit s hlavní, tak to udělá prostřednictvím žádosti o spojení větví, které však podléhá privilegovanějším rolím, které jsou především vlastník „Owner“ repositáře a údržbář „Maintainer“.

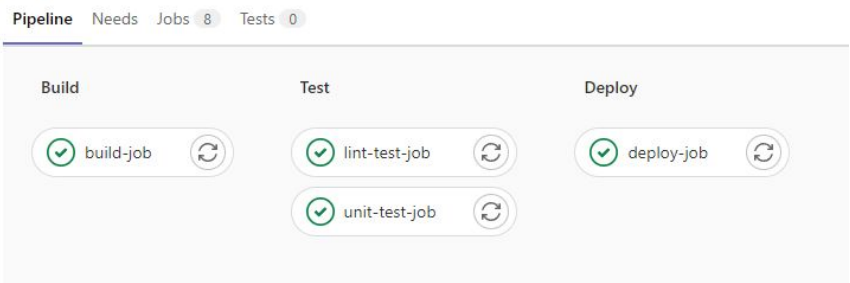
Po schválení může být provedeno spojení obou větví a nové části kódu jsou zaintegrovány do nové stabilní verze softwarového projektu.

Správa repositářů nabízí dále i další funkce, které jsou spíše analytického charakteru. Umožňují sledovat nejaktivnější členy projektů a vizualizovat míru jejich přispívání do repositáře prostřednictvím grafů.

3.5.4 Konfigurace CI/CD

Jak již bylo zmíněno v části 2.3.3, konfigurace CI/CD procesu se provádí prostřednictvím souboru `.yaml`, který obsahuje sadu instrukcí, které se vykonají po každé

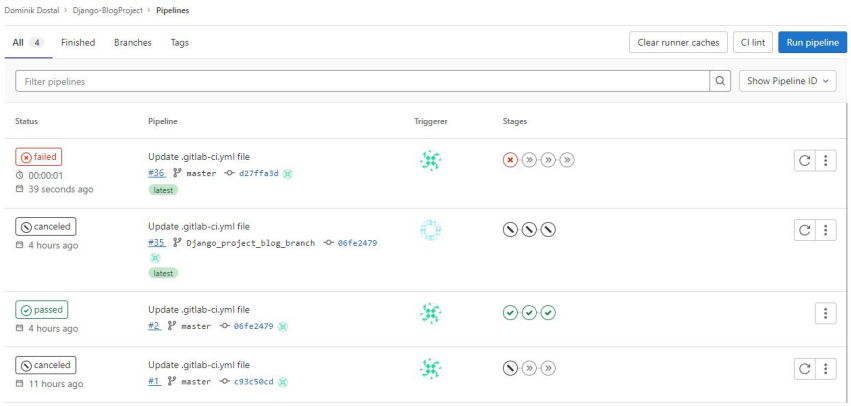
změně v repositáři. Soubor lze vytvořit ihned po naimportování nového projektu. GitLab umožňuje i jeho grafické znázornění a nabízí i jednoduchý textový editor, kde může být **.yaml** editován.



Obr. 3.7: Vizualizace jednoduché pipeline (Zdroj: Autor)

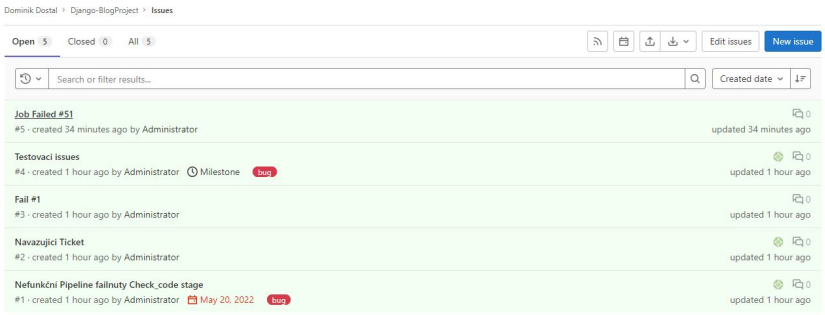
Jeho přesná podoba a komplexnost závisí na projektu a na typu agenta, který příkazy bude provádět. Trochu jinak bude `.yaml` vypadat pro Shellovy exekutor a jinak pro Docker exekutor.

Pipeline podle toho jak proběhne může nabývat několika stavů. Což GitLab umí zobrazit v přehledné tabulce, v které je možné i vysledovat, která část instrukcí neproběhla úspěšně, případně která stále běží a celkově vysledovat celou historii spouštění CI/CD instrukcí.



Obr. 3.8: Přehled CI/CD instrukcí (Zdroj: Autor)

Uživatel, pokud chce, může vytvořit přímo proklikem z neúspěšného kroku požadavek a zaevidovat ho v GitLabu, případně do jiného nástroje určeného ke kooperaci, a dokumentovat tak průběh jeho řešení. Takto vytvořený požadavek si sebou automaticky nese informaci, který krok nemohl být proveden, nebo který test neproběhl úspěšně, což následně usnadňuje identifikaci problému.



Obr. 3.9: Požadavky (Zdroj: Autor)

Požadavky se pak v GitLabu řadí do fronty a je zde možné řídit celý životní cyklus požadavku. Jako je přidávání návazných požadavků, přidávání komentářů, obrázků a dalších užitečných věcí k řešení.

Závěr

Cílem této bakalářské práce bylo seznámit se s nástroji pro automatizovanou správu softwarových projektů, které jsou vhodné pro použití na systémech Linux a s možností integrace verzovacích nástrojů jako je GiT a nástrojů pro správu technické dokumentace. Další neméně důležitým požadavkem byla podpora jazyků C/C++.

V první kapitole této práce byly definovány a popsány důležité pojmy, týkající se automatizované správy softwarových projektů, tak aby bylo možné pochopit princip automatizované správy softwarových projektů. V další kapitole jsem stanovil metodiku srovnání a následně popsal vlastnosti vybraných nástrojů, jak se instalují, jak se konfigurují a možnosti jejich rozšiřitelnosti. Nakonec bylo provedeno srovnání každého nástroje a jejich klíčových vlastností.

V rámci této práce se podařilo každý vybraný nástroj nainstalovat, nakonfigurovat a spustit na virtuálním serveru. Po úspěšné instalaci bylo každé řešení otestováno, přičemž jsem se seznámil s prostředím každého z nich. Vytvořil testovací projekt, nahrál do něj repositář se zdrojovým kódem z GitHubu a otestoval možnosti, které nabízejí při tvorbě pipeline. Mezi vybrané nástroje patří Jenkins, GitLab, TeamCity a Bamboo Atlassian.

Na základě těchto poznatků byla vytvořena tabulka a srovnání všech hodnocených nástrojů. Bylo porovnáváno jak jejich uživatelské prostředí, tak možnosti rozšiřitelnosti a konfigurace, instalace, nebo i jejich licencování. V tomto srovnání jako nejvíce vyhovující vyšly nástroje Jenkins a GitLab, které vynikly zejména tím, že poskytují komplexní nástroje pro automatizovanou správu softwarových projektů a zároveň poskytují řešení, které je nabízeno zdarma. Přičemž z těchto dvou bylo jako lépe vyhovující vyhodnocen GitLab.

V poslední třetí kapitole je pak předvedena instalace vybraného řešení GitLab. Jsou podrobně popsány nutné prerekvizity pro spuštění GitLabu. Pak je popsána i instalace GitLab runneru (Agenta), který je zcela nezbytný pro činnost celého CI/CD nástroje. GitLab runner byl instalován na samostatném serveru, který pak byl propojen s hlavní instalací Gitlab Omnibus. Stejně tak byl na samostatný server nainstalován nástroj pro projektové řízení Redmine a po úspěšné instalaci byla provedena integrace s Gitlab Omnibus. V příloze je pak uvedeno jak byl Redmine instalován.

Literatura

[1] Understanding DevOps: What is CI/CD? [online]. Raleigh, Severní Karolína, USA: Raleigh, Severní Karolína, USA, 2018 [cit. 2021-12-08]. Dostupné z: <https://www.redhat.com/en/topics/devops/what-is-ci-cd>

[2] M. DUVALL, Paul, Steve MATYAS a Andrew GLOVER. Continuous Integration: Improving Software Quality and Reducing Risk [online]. 1. Crawfordsville, Indiana., 2007 [cit. 2021-12-08]. ISBN ISBN 978-0-321-33638-5. Dostupné z: Amazon

[3] Continuous Delivery Principles: Continuous integration vs. continuous delivery vs. continuous deployment [online]. Sydney, Austrálie, 2021 [cit. 2021-12-08]. Dostupné z: <https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>

[4] What is a CI/CD pipeline?: Understanding DevOps [online]. Raleigh, Severní Karolína, USA, 2019 [cit. 2021-12-08]. Dostupné z: : <https://www.redhat.com/en/topics/devops/what-is-ci-cd>

[5] CI Server. M. DUVALL, Paul, Steve MATYAS a Andrew GLOVER. Continuous Integration: Improving Software Quality and Reducing Risk [online]. 1. Crawfordsville, Indiana., 2007, s. 8 [cit. 2021-12-08]. ISBN ISBN 978-0-321-33638-5. Dostupné z: Amazon

[6] Continuous Integration [online]. Melrose, USA, 2000 [cit. 2021-12-08]. Dostupné z: <https://www.martinfowler.com/articles/originalContinuousIntegration.html#TheBenefitsOfContinuousIntegration>

[7] *Encyclopedia of Information Systems*. BIDGOLI, Hossein. Academic Press; 1st edition (August 12, 2002), 2002, s. 509. ISBN 9780122272400.

[8] Hudson devs vote for name change; Oracle declares fork [online]. 2011 [cit. 2021-12-12]. Dostupné z: <https://www.computerworld.com/article/2746627/hudson-devs-vote-for-name-change--oracle-declares-fork.html>

[9] How To Install Jenkins on Ubuntu 20.04 [online]. 2020 [cit. 2021-12-12]. Dostupné z: <https://www.digitalocean.com/community/tutorials/how-to-install-jenkins-on-ubuntu-20-04>

[10] *Jenkins : Vulnerability Statistics* [online]. cvedetails, 2022 [cit. 2022-05-03]. Dostupné z: <https://www.cvedetails.com/product/34004/Jenkins-Jenkins.html?vendor_id=15865>

[11] Java requirements [online]. [cit. 2021-12-08]. Dostupné z: <https://www.jenkins.io/doc/administration/requirements/java/#java-requirements>

[12] Omnibus GitLab Documentation [online]. [cit. 2021-12-08]. Dostupné z: <https://docs.gitlab.com/omnibus/>

[13] Install self-managed GitLab [online]. [cit. 2021-12-12]. Dostupné z: <https://about.gitlab.com/install/#ubuntu>

[14] Installation [online]. [cit. 2021-12-08]. Dostupné z: <https://docs.gitlab.com/ee/install/>

[15] GitLab Runner [online]. [cit. 2021-12-09]. Dostupné z: <https://docs.gitlab.com/runner/>

[16] Pipeline architecture [online]. [cit. 2021-12-09]. Dostupné z: <https://docs.gitlab.com/ee/ci/pipelines/pipeline_architectures.html>

[17] Webhooks [online]. [cit. 2021-12-09]. Dostupné z: <https://docs.gitlab.com/ee/user/project/integrations/webhooks.html>

[18] Integration [online]. [cit. 2021-12-09]. Dostupné z: <https://docs.gitlab.com/ee/integration/>

[19] *Gitlab : Vulnerability Statistics* [online]. CVE details, 2022 [cit. 2022-05-03]. Dostupné z: <https://www.cvedetails.com/vendor/13074/Gitlab.html>

[20] *Security Releases* [online]. GitLab, 2022 [cit. 2022-05-03]. Dostupné z: <https://about.gitlab.com/handbook/engineering/security/#-internal-security-notification-dashboard>

[21] How to install Team City on Ubuntu Server [online]. 2017 [cit. 2021-12-12]. Dostupné z: <https://garywoodfine.com/how-to-install-team-city-10-x-on-ubuntu-16-x/>

[22] TeamCity [online]. [cit. 2021-12-09]. Dostupné z: <https://www.jetbrains.com/teamcity/>

[23] Supported Platforms and Environments: Operating Systems [online]. Praha 4, Praha, 2021 [cit. 2021-12-09]. Dostupné z: <https://www.jetbrains.com/help/teamcity/supported-platforms-and-environments.html>

[24] Build Agent [online]. Praha 4, Praha, 2021 [cit. 2021-12-09]. Dostupné z: <https://www.jetbrains.com/help/teamcity/build-agent.html>

[25] *Teamcity : Vulnerability Statistics* [online]. CVE details, 2022 [cit. 2022-05-03]. Dostupné z: <https://www.cvedetails.com/product/30795/Jetbrains-Teamcity.html?vendor_id=15146>

[26] *Security Notes* [online]. JetBrains, 2022 [cit. 2022-05-03]. Dostupné z: <https://www.jetbrains.com/help/teamcity/cloud/security-notes.html#Permissions>

[27] *What is Static Code Analysis?* [online]. JetBrains, 2022 [cit. 2022-05-03]. Dostupné z: <https://www.jetbrains.com/teamcity/ci-cd-guide/concepts/static-code-analysis/>

[28] Supported platforms [online]. [cit. 2021-12-12]. Dostupné z: <https://confluence.atlassian.com/bamboo/supported-platforms-289276764.html>

[29] How to Install Bamboo in Ubuntu? [online]. 2020 [cit. 2021-12-12]. Dostupné z: <https://www.interserver.net/tips/kb/how-to-install-bamboo-in-ubuntu/>

[30] PostgreSQL [online]. [cit. 2021-12-12]. Dostupné z: <https://confluence.atlassian.com/bamboo/postgresql-289276816.html>

[31] Build CHAIN [online]. Praha 4, Praha, 2021 [cit. 2021-12-09]. Dostupné z: <https://www.jetbrains.com/help/teamcity/build-chain.html>

[32] Extending TeamCity [online]. Praha 4, Praha, 2021 [cit. 2021-12-09]. Dostupné z: <https://www.jetbrains.com/help/teamcity/extending-teamcity.html>

[33] Bamboo [online]. Sydney, Austrálie, 2021 [cit. 2021-12-09]. Dostupné z: <https://www.atlassian.com/purchase/product/bamboo>

[34] Bamboo Server and Data Center feature comparison [online]. Sydney, Austrálie, 2021 [cit. 2021-12-09]. Dostupné z: <https://confluence.atlassian

.com/bamboo/bamboo-server-and-data-center-feature-comparison-1063170546.html>

[35] Apps for Bamboo [online]. Sydney, Austrálie, 2021 [cit. 2021-12-09]. Dostupné z: <https://developer.atlassian.com/server/bamboo/>

[36] *Bamboo : Vulnerability Statistics* [online]. CVE details, 2022 [cit. 2022-05-03]. Dostupné z: <https://www.cvedetails.com/product/22350/Atlassian-Bamboo.html?vendor_id=3578>

[37] *Best practices for Bamboo security* [online]. Atlassian, 2022 [cit. 2022-05-03]. Dostupné z: <https://confluence.atlassian.com/bamboo/best-practices-for-bamboo-security-289277199.html>

[38] *Best practices for Bamboo security* [online]. Atlassian, 2022 [cit. 2022-05-03]. Dostupné z: <https://confluence.atlassian.com/bamboo/best-practices-for-bamboo-security-289277199.html>

[39] *Bamboo security advisories* [online]. Atlassian, 2022 [cit. 2022-05-03]. Dostupné z: <https://confluence.atlassian.com/bamboo/bamboo-security-advisories-289276756.html>

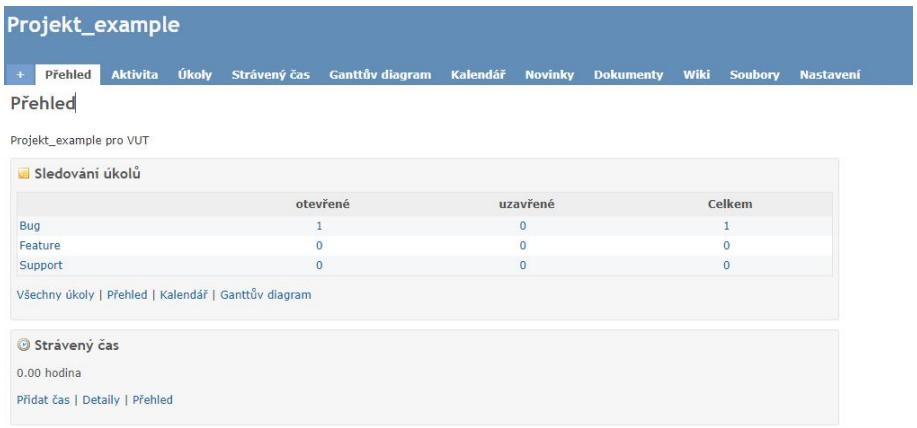
[40] *GitLab installation minimum requirements* [online]. GitLab, 2022 [cit. 2022-05-03]. Dostupné z: <https://docs.gitlab.com/ee/install/requirements.html>

[41] *Executors* [online]. GitLab, 2022 [cit. 2022-05-06]. Dostupné z: <https://docs.gitlab.com/runner/executors/>

[42] *Redmine Wiki* [online]. Redmine, 2022 [cit. 2022-05-07]. Dostupné z: <https://www.redmine.org/projects/redmine/wiki>

A Redmine

Redmine je svobodný open-source nástroj určený k projektovému řízení, který umožňuje svým uživatelům efektivně spravovat a udržovat přehled nad projekty, sledovat chyby, spravovat wiki a fóra.



Obr. A.1: Rozhraní Redmine (Zdroj:Autor)

V rámci Redmine lze využívat Ganttovy diagramy a kalendář, který uživatelům umožňuje projekty sledovat jejich průběh.[42]

Redmine byl poprvé publikován v roce 2006 a je napsán v jazyce Ruby.

A.1 Instalace prerekvizit

V této části je vysvětleno jak byla provedena instalace Redmine. Instalace byla provedena na čistý server s Ubuntu 20.04 LTE. Nejdříve je potřeba provést aktualizaci systému, aby bylo možné nainstalovat všechny potřebné závislosti.

```
$ sudo apt update
$ sudo apt upgrade
```

Instalaci všech potřebných prerekvizit provedeme tímto příkazem:

```
$ sudo apt install apache2 mysql-server mysql-client libapache2-mod-
    passenger build-essential libmysqlclient-dev libmysqlclient-dev
    imagemagick libmagickwand-dev libmagickcore-dev -y
```

V dalším kroku je potřeba vytvořit MySQL databází pro Redmine.

```
$ sudo mysql -u root -p

CREATE DATABASE redminedb CHARACTER SET utf8mb4;
CREATE USER 'redmineuser'@'localhost' IDENTIFIED BY 'Password';
GRANT ALL ON redmine.* TO 'redmineuser'@'localhost' WITH GRANT OPTION;
```

```
FLUSH PRIVILEGES;
exit;
```

A.2 Instalace Redmine:

```
$ sudo apt install redmine redmine-mysql -y
$ sudo gem update
$ sudo gem install bundler
```

Editace konfiguračního souboru `Passanger.conf`:

```
sudo vim /etc/apache2/mods-available/passenger.conf
```

Obsah souboru upravíme:

```
<IfModule mod_passenger.c>
    PassengerDefaultUser www-data
    PassengerRoot /usr/lib/ruby/vendor_ruby/phusion_passenger/locations
    .ini
    PassengerDefaultRuby /usr/bin/ruby
</IfModule>
```

Vytvoření symbolického odkazu:

```
$ sudo ln -s /usr/share/redmine/public /var/www/html/redmine
```

Vytvoření `Gemfile.lock`:

```
$ sudo touch /usr/share/redmine/Gemfile.lock
```

Změna vlastnických práv nad souborem `gemfile.lock`, tak aby si s ním apache mohl manipulovat:

```
$ sudo chown www-data:www-data /usr/share/redmine/Gemfile.lock
```

Změna vlastnických práv na složku pro Apache:

```
$ sudo chown -R www-data:www-data /var/www/html/redmine
```

Změna přístupových práv pro Apache:

```
$ sudo chmod -R 755 /var/www/html/redmine
```

A.3 Konfigurace Apache

Vytvoření konfiguračního souboru:

```
$ sudo nano /etc/apache2/sites-available/redmine.conf
```

Do konfiguračního souboru je potřeba vložit:

```
<VirtualHost *:80>
    ServerAdmin admin@example.com
    DocumentRoot /var/www/html/redmine
    ServerName redmine.example.com

    <Directory /var/www/html/redmine>
        RailsBaseURI /redmine
        PassengerResolveSymlinksInDocumentRoot on
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Pak provedeme následující:

```
$ sudo a2enmod rewrite
$ sudo a2ensite redmine.conf
$ sudo a2dissite 000-default.conf
$ sudo systemctl restart apache2
```

Pokud vše projde úspěšně, tak Redmine je dostupné na IP serveru. Prvotní přihlášení je nutné provést na účet admin s heslem „admin“.