FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Bakalářská práce

Efektivní design Gitlab runnerů pro softwarový vývoj v prostředí AWS

Studijní program: B0613A140005 Informační technologie
Studijní obor: Aplikovaná informatika

Autor práce: **Jiří Turyna**
Vedoucí práce: Ing. Mojmír Volf
Ústav nových technologií a aplikované informatiky

Liberec 2023

FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Zadání bakalářské práce

Efektivní design Gitlab runnerů pro softwarový vývoj v prostředí AWS

Jméno a příjmení:	**Jiří Turyna**
Osobní číslo:	M21000148
Studijní program:	B0613A140005 Informační technologie
Specializace:	Aplikovaná informatika
Zadávající katedra:	Ústav nových technologií a aplikované informatiky
Akademický rok:	2021/2022

Zásady pro vypracování:

1. Seznamte se s technologiemi CI/CD pipeline a zhodnoťte vhodnost jejich užití pro praktickou aplikaci.
2. Vytvořte design Gitlab runnerů pro konkrétní softwarový vývoj.
3. Zhodnoťe efektivitu v rámci AWS.
4. Porovnejte navrhovaný design s aktuálním řešením.
5. Navrhněte obecné řešení pro design Gitlab runnerů.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30-40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] King, T. H. (2019). *AWS: The Ultimate Guide From Beginners To Advanced For The Amazon Web Services (2020 Edition)*. Independently published.

[2] Vliet, J. V. (2021). *Programming Amazon EC2 1st edition by Vliet, Jurg van, Paganelli, Flavia (2011) Paperback* (1st ed.). O'Reilly Media.

Vedoucí práce:	Ing. Mojmír Volf Ústav nových technologií a aplikované informatiky
Datum zadání práce:	12. října 2021
Předpokládaný termín odevzdání:	22. května 2023

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

Ing. Josef Novák, Ph.D.
vedoucí ústavu

V Liberci dne 19. října 2021

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

22. května 2023 Jiří Turyna

EFEKTIVNÍ DESIGN GITLAB RUNNERŮ PRO SOFTWAROVÝ VÝVOJ V PROSTŘEDÍ AWS

ABSTRAKT

V teoretické části této práce je rozebrána metodika DevOps, její místo v Agilním vývoji, a také CI/CD nástroj, který umožňuje části DevOps prakticky naplňovat.

Dále je představeno několik poskytovatelů CI/CD řešení, z nichž je vybrán Gitlab a AWS, kterými se práce zabývá podrobněji.

Hlavním cílem práce je představit možnosti, jak vhodně zvolit poskytovatele CI/CD na základě technických a praktický měření. Nikoliv marketingu nebo popularitě. Z tohoto důvodu v této práci vzniká několik metrik, díky kterým je možné tato CI/CD řešení porovnávat a měřit. Získaná data jsou od každého z poskytovatelů zobrazována a porovnávána v grafech a tabulkách.

Práce se také, na základě představených metrik, zabývá analýzou efektivity CI/CD systému na jednom z klíčových projektů anonymní firmy a navrhuje možnost, která efektivitu zvýší.

V poslední řadě je v práci popsán obecný design CI/CD systému, který velmi úzce souvisí s architekturou a provozem Cloudových služeb.

EFFECTIVE GITLAB RUNNER DESIGN FOR SOFTWARE DEVELOPEMENT IN AWS

ABSTRACT

In theorethical part of the thesis is explained the methodology called DevOps, it's part in an Agile software development and also the tool CI/CD which helps to practically implement DevOps methodology.

Following, there are introduced several CI/CD solution providers from which AWS and Gitlab are selected to be used in practical part of this thesis.

The main goal of this thesis is to introduce a possible way to effectively choose a CI/CD solution provider based on technical data a messuring and not only rely on their marketing and popularity. Consecutively there are created a several metrics, which allow to messure and compare CI/CD solutions. Gathered data from AWS and Gitlab are then displayed and compared in a form of graphs and data tables.

Later based on these metrics the thesis examines an effectivity of a specific CI/CD system on one of the key projects of the anonymous company and proposes a practical suggestion on which the effectivity should improve.

In the last part there is a theoretical manual on the design of a CI/CD system and its connection to architecture and operation of Cloud services.

PODĚKOVÁNÍ

Především bych chtěl poděkovat rodičům a sestře, kteří mě při mém studiu podporovali, jak finančně tak, psychicky a i přes některá zaškobrtnutí mi nepřestali věřit.

Dále bych chtěl poděkovat vedoucímu této práce panu inženýru Volfovi, který mi pomáhal s teoretickou části, vizuální stránkou této práce a byl vždy ochotný zjišťovat nejasností a formality, které při psaní této práce vznikali.

Obrovské poděkování patří mému oponentovi, kolegovi a mentorovi v DevOps, Andreji Dorincovi, který mě prováděl praktickou částí této práce.

Také děkuji všem mým kolegům v Jablotron Cloud Services, kteří mi vůbec umožnili spojit práci a dokončování studia. Díky patří také kolegům ze Support týmu JCS, kteří mi vyšli vstříc a v dobách kdy jsem se věnoval škole si mé pracovní povinnosti rozdělili.

Také bych rád vyjádřil poděkování mé nejlepší kamarádce Věře Puldů za její důvěru, emocionální podporu a dlouhé konzultace a diskuze nejen nad IT.

OBSAH

SEZNAM ZKRATEK

EU	Evropská unie
TUL	Technická univerzita v Liberci
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
DevOps	Development and Operations
AWS	Amazon Web Services
EC2	Elastic Cloud Compute
YAML	YAML Ain't Markup Language
SLA	Service-level agreement
CI/CD	Continous Integration / Continous Delivery
IDE	Integrated Development Environment
K8s	Kubernetes
URL	Uniform Resource Locator
IDE	Integrated development environment
CPU	Central processing unit
RAM	Random-access memory
API	Application programming interface
IAM	Identity and Access Management
UI	User Interface
OS	Operation System

1 Úvod

Tato bakalářská práce se zabývá navržením efektivního CI/CD systému pro softwarový vývoj ve firemním prostředí za využití DevOps metod společně s využitím řeší Gitlab Runner a AWS. Práce je rozdělena do několika částí.

V první části je teoreticky rozebrána metodika DevOps a vysvětlení jejího přístupu k Softwarovému vývoji. Zároveň jsou zmíněny a vysvětleny základní hodnoty a praktické kroky této metodiky společně s nástroji běžně používané pro její provoz.

Dále se tato práce specificky zabývá nástrojem CI/CD, který dovoluje efektivní využívání všech kroků DevOps, které zaručují efektivní vývoj aplikací, jejich provoz a také jejich doručování koncovým zákazníkům v bezproblémovém stavu.

Následuje část ohledně funkcionality Jobů v rámci CI/CD řešení a představení dvou zvolených poskytovatelů Gitlab Runner, na kterých jsou prováděny testy a měření.

V praktické části jsou nastíněny metriky, díky kterým je možné tato CI/CD řešení měřit a porovnávat. Následně je na celkem třech systémech toto měření provedeno, data jsou zobrazena v grafech a tabulkách a samozřejmě vzájemně porovnávána.

V předposlední části je vytvořena analýza konkrétního CI/CD řešení anonymní firmy a za využití Gitlab API společně s python3 je připraven script, díky kterému dochází ke změření efektivity tohoto systému na jednom z klíčových projektů této firmy.

Poslední část této práce se zabývá navrhnutím obecného řešení při vytváření designu CI/CD systému a provoz Cloudového řešení. Zmiňuje tři situace, ve kterých se může potenciální navrhovatel ocitnout a také je zde několik efektivních možností na základě měření z předchozích kapitol.

2 MOTIVACE

Tato práce se snaží popsat možnosti, jak rozlišit vhodnost hlavních dostupných poskytovatelů IASS a PASS služeb pro Cloudové firmy a softwarové vývoje. Těchto poskytovatelů vzniká čim dál tím více a kromě finanční a marketingové stránky není často jiná možnost, jak mezi nimi vybrat, který je pro danou situaci nejefektivnější. V této práci je představeno několik metrik, díky kterým lze tyto služby porovnat. Takto naměřená data jsou v této práci uvedena v grafech a tabulkách, na základě kterých dochází k závěrům a vyhodnocením.

Zároveň je součástí práce analýza efektivnosti CI/CD systému jednoho z klíčových projektů anonymní firmy a pomocí některých představených metrik je doporučeno řešení, které by mělo efektivitu systému a celého vývoje ve firmě zvýšit.

Osobně jsem si téma práce zvolil, jelikož jsem chtěl zdokonalit své znalosti v DevOps systémech a také mě velmi zajímal systém dostupných služeb v AWS, který mi imponoval především kvůli své komplexitě a vnímal jsem ho jako velkou výzvu.

3 DEVOPS

DevOps je termín, který popisuje metodologii v softwarovém vývoji a kombinuje dvě hlavní části, ze kterých se tento vývoj běžně skládá, Development (vývoj) a Operation (provoz). Důvodem vzniku DevOps je zvýšení efektivity softwarových firem. Tato metodika nastavuje základy a procesy softwarového vývoje, které vedou ke zvýšení produktivity, kvality produktu a spokojenosti zákazníka. Hlavní inspirací pro vznik metodiky je třetí průmyslová revoluce z roku 1980 v rámci výrobních organizací. V této době zavedlo několik firem jistou sadu zásad a postupů, díky kterým dokázali překonat konkurenci. [Kim+16]

Jinými slovy se v IT odvětví často využívá termín DevOps k definici procesů, které pomáhají týmům Development a Operations spolupracovat. Bez DevOps často dochází k nepochopení se mezi těmito týmy a proto se jako best practice osvědčilo zaměstnávat DevOps týmy, které mají zkušenosti z obou světů a často můžou fungovat jakožto „techničtí mediátoři“ v softwarovém vývoji.

V praktickém slova smyslu DevOps vývojář nastavuje procesy a systémy, aby se například Development tým mohl plně soustředit na vývoj nových funkcí a nemusel řešit výběr/nastavování různých databází, řešení monitoringu a alertingu nebo například komunikaci se zákazníky. Blíže jsou hlavní využívané principy popsány v kapitole 4.

4 CI/CD

CI/CD je nástroj, díky kterému je možné využívat všechny hodnoty a metody přístupu DevOps k vývoji a dodání aplikace v praxi. Tento nástroj byl navržen tak, aby byla implementace hodnot DevOps co nejefektivnější a zároveň nedocházelo k inflaci cognitive loadu při jejich využívání.

4.1 Základní kroky CI/CD a jejich vysvětlení

CI/CD je nástroj složen z osmi konkrétních kroků. Tyto kroky jsou strukturovány tak, aby usnadňovaly implementaci a realizovaly metodiku DevOps ve dvou hlavních částech.

První část CI/CD se nazývá Continous Integration (CI) a zabývá se především vývojem softwarového týmu, kterému pomáhá s implementaci agilního vývoje a umožňuje práci více vývojářů na stejném projektu zároveň tak, aby bylo zabráněno omezení nebo ohrožení práce kteréhokoliv z nich.

Druhá část CI/CD se nazývá Continous Delivery (CD), která slouží k dodání funkčního a testovaného produktu na jedno z vývojových prostředí dle dalšího využití (pro zákazníky a uživatele slouží obvykle produkční prostředí). [Red23] Každý z těchto osmi kroků lze implementovat různými praktickými způsoby. Pro konkrétní krok je možné zvolit jinou technologii, ať už z důvodů preference, znalostní bázi, efektivitě nebo finančních prostředků. [Den23]

4.1.1 Plan

Během této fáze dochází k podrobnému rozplánování služby a následnému vyhodnocení, který z vývojových procesů bude pro vývoj aplikace vhodnější. Mezi nejpoužívanější procesy patří Scrum a Agile.

Scrum například využívá vývojového režimu Sprint, ve kterém se vývojáři věnují několika konkrétním úkolům po dobu jednoho až dvou týdnů. Následně je Sprint zakončen retrospektivním meetingem, ve kterém dochází k naplánování dalších kroků a následné rozřazení úkolů mezi vývojáři pro další Sprint. [MHM19]

Scrum upřednostňuje vývoj produktu autonomně v týmu vývojářů, kteří mají podobné znalosti a zkušenosti. Zároveň se s volbou procesu Scrum

předpokládá, že produktové požadavky se budou během existence služby v budoucnu měnit a dále rozvíjet.

Agile využívá k procesování jednotlivých úkolů Backlog, ve kterém jsou úkoly prioritizovány a rozřazeny samotnými vývojáři nebo některým z manažerů. V rámci Agile procesu je přiřazený řešitel zodpovědný za dokončení konkrétního úkolu.[Cho21]

Zde je zmíněno několik produktů třetích stran, které usnadňují využívání, kteréhokoliv vývojového procesu - Trello, Jira, Servicenow.

4.1.2 Code

Druhá fáze CI/CD se zabývá sepsáním samotným Codem aplikace v rámci jednoho z používaných IDE (Eclipse, Visual Sudio Code, Netbeans, ...). Code je psán v jednom z programovacích jazyků (Python, C++, Java, ...) nebo může být také psán ve formátu YAML v případě, že se jedná o konfigurace například Docker Containers.

V této Bakalářské práci byl k sepsání zdrojových codů pro Jobs, získávání metrik a jejich parsování, využit programovací jazyk Python ve verzi 3.9. Samotná konfigurace Pipelines je pak sepsána ve formátu YAML.

4.1.3 Build

Jedním z kroků build fáze je kompilace samotného Codu, který provede na základě zvoleného programovacího jazyka instalaci všech potřebných dependencies nutných pro spuštění služby. Následně dochází k první kontrole Codu ohledně programátorských chyb. [Sem23]

Běžně je cloudový software Deployován pomocí container nástroje Docker (nebo jiné alternativě) a v takovém případě dochází v build fázi k vytvoření Pipelines na základě jejich YAML konfigurace.

K buildu Pipelines byl v praktické části použit Gitlab a AWS CodeBuild.

4.1.4 Test

Jakmile je Code v build fázi zkontrolován, že neobsahuje chybu na programátorské úrovni a je tedy v pořádku zkompilován, je také potřeba provést testy a security validace, které zaručí správné chování služby tak, aby nedošlo k doručení závadného produktu zákazníkům.

Běžně jsou tyto testy sepsány za pomoci některých z programovacích jazyků a následně jsou přidány ke spuštění v YAML konfiguraci Pipelines.

Testování Security Validace není součástí tohoto projektu, zdůvodu časové náročnosti práce. Každopádně validování Pipeline se věnují služby Aqua Security, Terraform nebo AWS Control Tower.

4.1.5 Release

Code je v této fázi ukládán do jednoho z dostupných Code Repositories, které díky větvení a ukládání do Cloudu těchto služeb třetích stran, umožňují práci několika vývojářů na stejném produktu zároveň. Obvykle je produkční verze uchovávána v Master větvi projektu a k nahrání změn některé z větví je používáno propojení, nebol-i Merge.

Nejpoužívanější Code Repositories jsou Gitlab, Github nebo AWS CodeCommit. K udržování Codu pro tento projekt byl zvolen Gitlab a AWS CodeCommit.

4.1.6 Deploy

Jakmile je aplikace zkompilována a otestována, dochází k jejímu Deployi na jedno z vývojových prostředí Production, Development, Staging (více o jednotlivých prostředích je popsáno v sekci Vývojová prostředí). Pokud dojde k úspěšném Deployi instance služby, projeví se zároveň i všechny změny v bázi Codu v daném prostředí. Tedy, pokud dojde k Deployi nové verze Production prostředí, koncoví zákazníci budou mít k dispozici všechny nově přidané funkce a změny.

K Deploymentu a konfiguračnímu managementu je možné používat například některý z těchto nástrojů - Ansible, Puppet, Terraform.

4.1.7 Operate

V této fázi CI/CD je služba již úspěšně spuštěna a dostupná uživatelům na Production prostředí. Účelem Operations týmu je zajistit, aby byla instance dostupná po co možná nejdelší dobu. K tomu využívají metody Monitoringu (popsáno níže v sekci Monitoring) a také pomocí Orchestrace. Orchestrace spočívá v přípravě, administrování a škálování serverů s dostatečně velkou výpočetní kapacitou pro obvyklé traffic služby. V případě podcenění nebo nesprávného vyhodnocení trafficu nemusí docházet k plynulému odbavování požadavků zákazníků.

Proto, je v rámci Orchestrace cloudové služby běžný trend, přecházet z baremetal serverů, které se složitě rozšiřují na virtuální zařízení hostované některým z velkých poskytovatelů jako je AWS, Google nebo Azure.

Samotná Orchestrace spočívá v konfiguraci serverů a z důvodu finančních prostředků může například umožňovat, na základě metrik trafficu, škálování výpočetního výkonu podle konkrétní denní hodiny. K Orchestraci a její konfiguraci lze použít například jednoho z následujících poskytovatelů - Kubernetes, Amazon ECS, Azure AKS

4.1.8 Monitoring

Monitorování služby je poslední fází CI/CD a slouží k zajištění bezchybného běhu aplikace v režimu 24×7 tak, aby případný výpadek měl co nejmenší dopad na koncové uživatele. Většinou se ve firmách o tuto část stará Operations tým nebo případně Support tým. Často je k monitorování služby využívána aplikace třetí strany, která na základě metrik služby vykresluje různé grafy (populární je například Grafana). Současně je pravděpodobně nastaven některý z dostupných Alertovacích systému jako je Prometheus nebo Zabbix.

4.2 VÝVOJOVÁ PROSTŘEDÍ

Aplikace je obvykle provozována na třech konkrétních vývojových prostředích tak, aby umožňovala vývojovému týmu vytvářet změny a aktualizace aplikace, jejich testování a případnou opravou nechtěných chyb ještě před vydáním do ostrého provozu služby. Zároveň je tak možno úplně předejít nebo zajistit co nejkratší dobu výpadku služby pro koncové zákazníky a uživatele.

Každé prostředí tak má proto svůj konkrétní účel a také se liší svými uživateli. [Fli22]

4.2.1 Production

Produkční prostředí, nebo tak zvané ostré prostředí je verze aplikace, kterou využívají obvykle platící zákazníci služby. Už jen z principu vyplývá, že by se v tomto prostředí mělo vyskytovat co nejméně chyb a dalších problémů zabraňujících využití služby. Firma/společnost provozující tuto Cloudovou aplikaci má obvykle sepsánu s uživateli smlouvu Service-level agreement (SLA), ve které se zavazuje k bezvýpadkovému dodáváním služby. Samozřejmě obvykle s ohledem na možné výpadky a servisní práce. V případě porušení SLA smlouvy je dále stanovena kompenzace vůči zákazníkům.

4.2.2 Staging

Jedná se o prostředí, které by v ideálním případě mělo být naprosto totožné s produkčním prostředím. Slouží jako prostředník mezi produkčním a vývojovým prostředím a běžně je využíváno testerským týmem ke kontrole vývojových změn a ověřováním, že provedené změny nebo nové funkce z vývoje neohrozí bezvýpadkovost ostrého provozu služby. Jakmile změna projde verifikací na staging prostředí je zařazena do produkčního prostředí, obvykle ve formě aktualizace služby.

4.2.3 Development

Vývojové prostředí, jehož hlavním uživatelem je vývojový tým, který vytváří nové změny a funkce na základě prioritizace. Obvykle je k vývoji aplikace využívána jedna z předních metodologií agilního vývoje, Scrum nebo Kanban (oba vývojové procesy jsou popsány v předchozí kapitole v sekci 4.1.1).

5 PŘEDSTAVENÍ VYBRANÝCH POSKYTOVATELŮ CI/CD

5.1 GITLAB

Prvním z předních poskytovatelů CI/CD Pipeline, který je v rámci této Bakalářské práce testován, je Gitlab, který uživatelům nabízí dva přístupy při odbavování Jobs a buildu Pipeline.

První z nich jsou Gitlab-Shared Runners. Tato možnost je ideální pro malé vývojové týmy, které nemají velký počet koncových zařízení a proto nemusí být problémem obvyklé čekání při odbavování Pipeline.

Další výhodou pro menší vývoj je také fakt, že Free Tier této služby nabízí zdarma 400 CI/CD minut za měsíc. Další možnosti jsou již poměrně dražší a platí se individuálně.

Druhou možností je zaregistrovat a k odbavování Pipeline využívat vlastní zařízení, tento typ runnerů se nazívá Gitlab-Private Runner. Velkou výhodou je fakt, že lze definovat, kdo bude mít ke konkrétnímu Runneru přístup a lze také při designu Pipelines přesně zvolit Runner, který může být pro odbavení využit. Tím je zaručena spolehlivost odbavení Pipeline. Samozřejmostí je také fyzická administrace zařízení a v případě potřeby možnost navyšovat výpočetní výkon počítače.

Samotnou registrací Gitlab Runneru a jeho konfigurací se zabývá následující kapitola v sekci Gitlab runner.

5.2 AMAZON WEB SERVICES

Druhým CI/CD poskytovatelem, který bude v této Bakalářské práci podrobněji testován je Amazon Web Services. Amazon je aktuálně velkým marketingovým gigantem a díky AWS se těší popularitě také v IT průmyslu, ve kterém poskytuje velké množství služeb pro vývojové týmy.

Pro potřeby této Bakalářské práce je důležitá především AWS CI/CD služba s názvem AWS CodeBuild, která je řešená kombinací vývojářských nástrojů, jejich výčet a využití v této práci je popsán v kapitole AWS Runner.

Primární výhodou AWS je dostupnost využití a propojení vše služeb, které AWS nabízí. Například služba EC2 instances, díky které je možný pronájem

zařízení o různé míře výpočetního výkonu nebo služba S3 bucket, což je Cloudové objektové úložiště.

Druhou velikou výhodou tohoto poskytovatele je škálovatelnost. Což je vlastnost díky, které je možné Deployovanou aplikaci přizpůsobovat příchozímu trafficu. Tedy, pokud zákazníci využívají hypotetickou aplikaci například během dne, je díky některým CI/CD poskytovatelům možno na takovou situaci rychle zareagovat snížením počtu aktivních serverů a další infrastruktury během noci. Tím dochází především u větších vývojových týmů k významnému šetření finančních prostředků.

5.3 KUBERNETES

Kubernetes, také přezdíván K8s, je open-source orchestrátor containerů a virtualizace a zároveň je také posledním z představovaných poskytovatelů CI/CD. Bohužel jsou K8s z důvodu jeho komplexity a také časové náročnosti této Bakalářské práce zmíněny pouze okrajově. Přesto je důležité K8s zmínit, především kvůli jeho hlavní výhodě, kterou sdílí společně s AWS. Jedná se opět o možnost škálovatelnosti (tato vlastnost je blíže popsána v Amazon Web Services).

6 CI/CD SYSTÉM A JEHO HLAVNÍ PRVKY

Tato kapitola se zabývá obecnou definicí CI/CD a slouží k ujasnění jejích hlavních částí, které jsou v navazujících kapitolách dále využívány k provedení série měření metrik a generování dat nutných pro porovnání jednotlivých CI/CD poskytovatelů.

Běžný CI/CD model se skládá z následujících částí: Repository, Pipeline, Stage a Job.

6.1 REPOSITORY

Jedná se o úložiště Codu a všech potřebných konfigurací pro spuštění daných scriptů, aplikačních řešení a podobně.

V repository zpravidla nejsou uloženy instalační balíčky/package nutné ke spuštění Codu. Ty jsou nainstalovány až na konkrétním zařízení nebo kontejneru, na kterém je požadováno spuštění Codu.

Typicky se jedná o Cloudovou službu, která může být hostována na konkrétním osobním serveru nebo lze využívat oficiální cloudovové řešení přímo od poskytovatele. Velmi často je součástí této služby právě možnost nastavování CI/CD procesů. [Git23e]

6.2 PIPELINE

Úkolem Pipeline je provést Deployment sekvenci, která doručí aplikaci až ke koncovému zákazníkovi přes všechny tři vývojové fáze. Development, Staging a Production. V každé fázi Pipeline probíhají všechny předem definované Stages, jejichž úkolem je vykonat konkrétní sekvenci Jobs, což může být například stanovení testů, díky kterým je zaručena bez problémová funkčnost aplikaci pro koncové zákazníky.

Pipeline je tvořena dle své konfigurace z pravidla v YAML souboru, který je uložen v kořenovém adresáři repository a může například notifikovat ostatní Pipeliny nebo zobrazovat a znovu spouštět předchozí instance Pipeline.[Git23d]

6.3 STAGE

Každá Stage může obsahovat jednu nebo více Jobs, které se vykonávají sériově a mohou i nemusí být na sobě závislé. Jakmile jsou úspěšně dokončeny všechny definované Jobs v konkrétní Stagi, je také tato Stage označena za kompletní.

6.4 JOB

Job je konkrétní krok, který má být vykonán zvoleným Imagem. Může se například jednat o unit test, který se automaticky spustí při uploadu local repository do remote repository (git push). Zároveň se může jednat o spuštění Python scriptu, vystavění Docker containeru, stažení Repository z jiného projektu nebo v Docker containeru spustit některou z možných Linuxových distribucí.[Git23c]

6.5 RUNNER

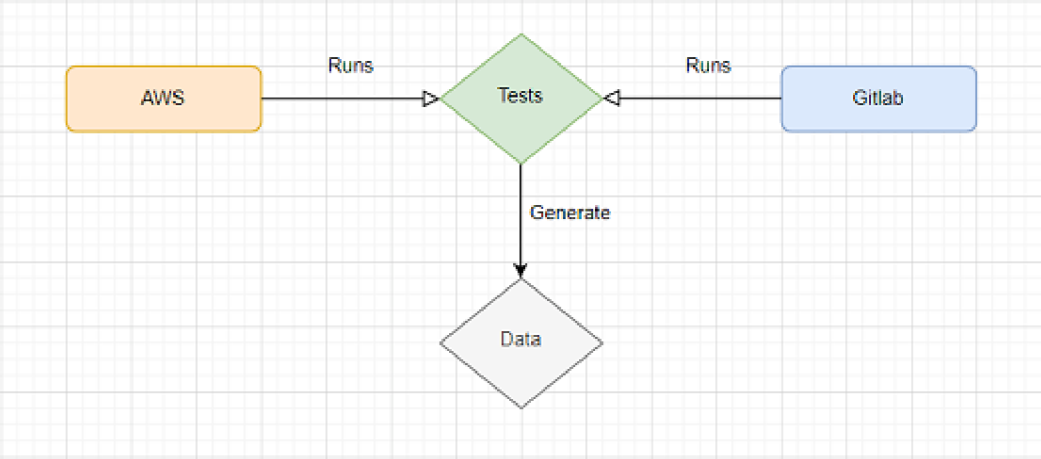
Samotným Runnerem je myšleno konkrétní zařízení, které je využíváno k odbavování samotných Pipelines podle konfigurace souboru v kořenovém Repository. V této Bakalářské práci byli k designu a testování využívati dva zvolení poskytovatelé CI/CD - AWS a Gitlab.

quad Každý z Runnerů vychází z designu konfigurace ve svém kořenovém repozitáři, která je u obou poskytovatelů psána ve formátu YAML. Tyto konfigurace se liší především svým názvem a syntaxí, ale jejich účel je totožný. [Git23a]

7 POROVNÁNÍ JEDNOTLIVÝCH POSKYTOVATELŮ

Jedním z hlavních cílů tohoto bakalářského projektu je porovnání dvou hlavních poskytovatelů služeb CI/CD, kterými jsou Gitlab a AWS. Pro porovnání bylo sestaveno několik metrik, které byly následně za využití stejné sestavy scriptů od každého z poskytovatelů sesbírány. Výsledná data jsou následně porovnávána.

Pro ujasnění bylo připraveno jednoduché schéma vysvětlující jak dochází ke sběru dat (viz. obrázek 7.1).



Obrázek 7.1: Schéma znázorňující sběr dat v tomto projektu

7.1 METRIKY

Pro potřeby tohoto projektu byly vytvořeny dvě skupiny metrik. Primární a Sekundární, které se dále jednotlivě dělí.

Nutná nastavení pro sběr metrik se u jednotlivých poskytovatelů samozřejmě liší, proto byly k získání porovnatelných dat mezi jednotlivými CI/CD poskytovateli vytvořeny testy, které mají zaručit, objektivnost dat v několika různých situacích a simulovat stavy, ke kterým dochází v běžném softwarovém vývoji.

7.1.1 Primární metriky

První metriky, díky které je možné porovnat různé CI/CD systémy. Pro sběr této metriky je nejprve nutné, aby Job proběhl a následně je možné získat jeho data ohledně celkové doby odbavení a zpoždění.

Jedná se o jednu ze dvou hlavních skupiny,

Average Job Duration

Průměrná doba odbavení všech Jobs Pipeline.

Queue

Celková doba zpoždění, jednotlivých Jobs z Pipeline.

7.1.2 Sekundární metriky

Druhá část generovaných a porovnávaných dat. Tyto metriky jsou generovány za běhu Jobu a měří vytížení RAM i CPU po dobu 60 sekund.

Average CPU

Průměr procentuální hodnoty CPU containeru, který zpracovává definované Jobs Pipeliny.

K získání dat ohledně využití CPU byl připraven bash script, který se volá pravidelně každou sekundu po dobu jedné minuty. Vysvětlení scriptu je v komentářích na obrázku 7.2.

bp-turyna > scripts > $ loadCpu.sh

You, now | 2 authors (You and others)

```
#!/bin/bash

# Získá celkový počet dostupných CPU containeru (1024 = 1)
numberOfCpus=$(cat /sys/fs/cgroup/cpu/cpu.shares | awk '{print substr($0,1,1)}')

# Parsuje výstup z příkazu "ps -A -o pcpu"
# Ukládá celkovou sumu do proměnné temp
temp=$(ps -A -o pcpu | tail -n+2 | paste -sd+ | bc)

# Vypočítá průměrné vytížení CPU
# Zapíná možnost desetinných míst v příkazu bc,
# který je potřeba pro výpočet průměrné hodnoty
temp=$(bc -l <<< "$temp"/$numberOfCpus)

# Výstup z příkazu "bc" je vypsaný s přesností na 2 desetinná místa
# a uložen do souboru, který byl předaný jako parametr $1
echo $temp | awk '{printf "%.2f \n", $1}' >> $1

```

Obrázek 7.2: Script pro získání dat ohledně využití CPU

Average RAM

Zprůměrovaná Data ohledně RAM jsou opět získávána bash scriptem. Vysvětlení je opět přidáno v komentářích.

bp-turyna > scripts > $ loadMem.sh

You, 5 seconds ago | 2 authors (You and others)

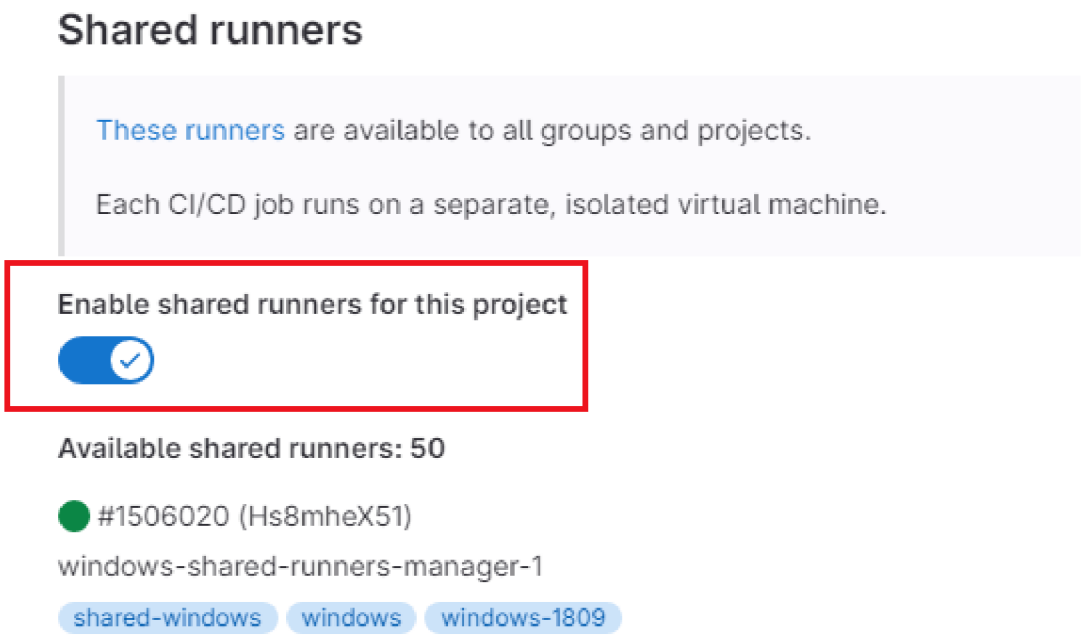
```
1  #!/bin/bash
2
3  # Počítá podíl mezi aktuálním zatížením Memory a celkově dostupnou Memory.
4  # Výsledek je přepočítán na procenta a uložen do souboru,
5  # který byl předaný jako parametr $1
6  free | grep Mem | awk '{print $3/$2 * 100.0}' >> $1
7
```

Obrázek 7.3: Script pro získání dat ohledně využití RAM

8 GITLAB RUNNER

Gitlab Runner nabízí vývojářům dvě možnosti jak odbavovat jejich Jobs. Jedná se o Shared a Private Runners. Specifikace zda má Job běžet na Shared nebo Private runneru je definována v nastavení konkrétního projektu.

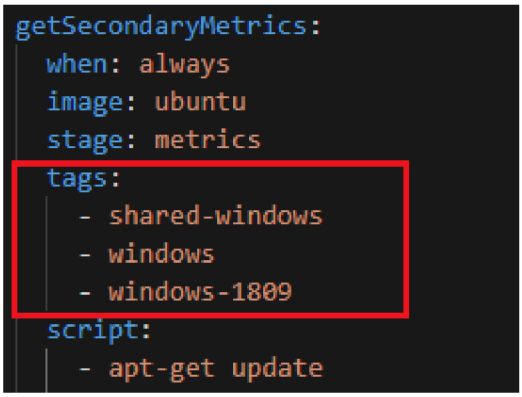
Konkrétně je tuto možnost nastavit v sekci Settings/CI/CD/Runners a povolit nebo zakázat možností „Enable shared runners for this project“.



Obrázek 8.1: Možnost pro zapnutí Shared runners pro Gitlab projekt

Dále lze také vyspecifikovat, aby Job běžel na jednom konkrétním runneru nehledě na to zda je Shared nebo Private. Tohoto chování lze docílit využitím tagů, které jsou definovány v konfiguračním souboru .gitlab-ci.yml.

Například pokud by bylo potřeba, aby Job běžel na runneru na obrázku 8.1, je potřeba přidat do konfiguračního souboru .gitlab-ci.yml tagy „shared-windows“, „windows“, „windows-1809“ viz. obrázek 8.2.



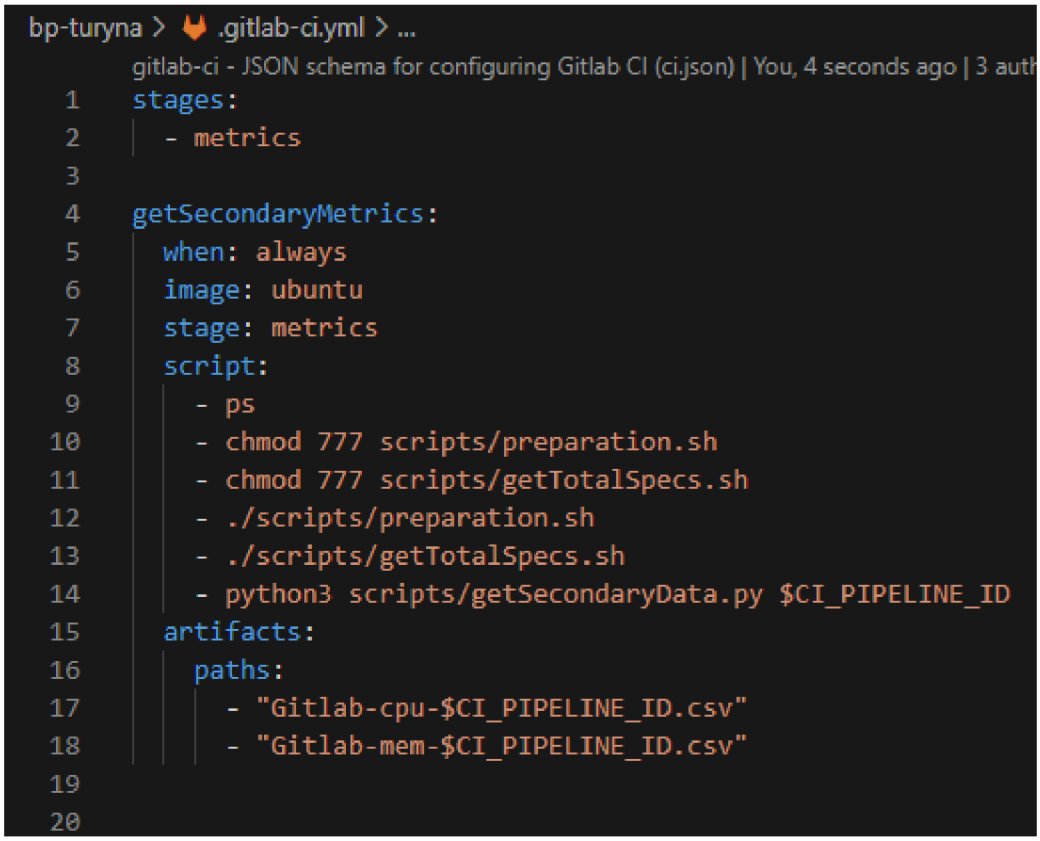
Obrázek 8.2: Specifikace runneru dle tagů přímo v konfiguračním souboru

Situace popisována na obrázku 8.2 je pouze ilustrativní, samozřejmě nedává smysl. V konfiguračním souboru je definováno, aby daný Job proběhl s imagem Ubuntu, ale dále je díky tagům definován specifický Windows runner. Tato operace by tedy skončila chybou.

8.1 SEKUNDÁRNÍCH METRIKY A KONFIGURACE PIPELINE

Samotná Pipeline se skládá z jednotlivých jobů, které mají být po spuštění Gitlab runneru vykonány. Její konfigurace se nachází v kořenovém adresáři repozitáře v souboru .gitlab-ci.yml (pokud není definováno jinak) a je psána ve formátu YAML.

Na obrázku 8.3 je možno vidět konfiguraci Gitlab Pipeline, která byla využita ke sběru sekundárních metrik. V následující částech je tato konfigurace podrobněji vysvětlena.



Obrázek 8.3: Nastavení využívané Gitlab Pipeline v tomto projektu

8.1.1 Definice Stages

V každé takovéto konfiguraci je nejprve definován povinný parameter „stages“, v případě na obrázku 8.3 existuje pouze jedna s názvem metrics, která definuje, že v celé Pipelině existuje jen jeden typ stage, ve které se budou dále definované joby odbavovat.

8.1.2 Definice Jobu

Následuje definice Jobu, který má být odbaven. Job je na obrázku 8.3 konfigurace definován na 4. řádku názvem **getSecondaryMetrics**. Dále následuje definice několika hlavních parametrů jobu:

1. **when** - parametr, který definuje kdy bude job odbaven. Může být nastaven na **always**, což znamená, že job bude proveden při každém spuštění Pipeliny (například pokud dojde k pushnutí změn do projektu) nebo může být nastaven na **manual**, v takovém případě bude Job vykonán pouze pokud dojde k jeho manuálnímu spuštění například přes Gitlab Api nebo manuálně z Gitlab UI.
2. **image** - Říká jaký image Docker container má být využit k odbavení Jobu. V tomto případě využíváme ke sbírání metrik linuxové příkazy

a proto byl zvolen image Ubuntu. Image je pullován lokálně z již předem stažených Imagů na systému nebo vzdáleně, například z centrálního Image úložiště jako je Docker Hub.

3. **stage** - Definuje v rámci jaké stage má být v rámci Pipeline tento Job odbaven. Tento parametr nemůže být ponechán prázdný a vzhledem k tomu, že v této konfiguraci figuruje pouze Stage s názvem **metrics**, je zde také vyplněn.

4. **script** - Sada konkrétních instrukcí, ze kterých se Job skládá. Jednotlivé kroky jsou podrobněji popsány níže.

5. **artifacts** - Jedná se pouze o definice souborů, které se mají po ukončení Pipeline zachovat (zbytek se smaže). V tomto případě se jedná o .csv soubory, do kterých byla zapsána data z python3 scriptu **getSecondaryData.py**.

8.1.3 Parametr Script

Na řádcích 9 a 10 obrázku 8.3 je možné vidět nastavení oprávnění v rámci Ubuntu containeru, který Job odbavuje za pomocí příkazu **chmod**. Tyto kroky jsou nutné definovat pro scripty, které jsou v Jobu plánovány spustit na úrovni prostředí, tedy image (například pro python3 script není nutné tato oprávnění definovat). V tomto případě to jsou scripty **preparation.sh** a **getTotalSpecs.sh**

1. **preparation.sh** - Na řádku 11 obrázku 8.3 dochází ke spuštění prvního scriptu, jehož úkolem je provést update a doinstalovat potřebné balíčky **python3** a **bc**.

bp-turyna > scripts > $ preparation.sh

```
...
1  #!/bin/bash
2  apt-get update
3  apt-get install -y python3
4  python3 --version
5  apt-get install bc
```

Obrázek 8.4: Script pro instalaci nutných balíčků

2. **getTotalSpecs.sh** - Script využívá běžně dostupné instrukcí pro linuxové distribuce **free** a **lscpu**, které dále parsuje. Výsledkem je **echo** obou získaných informací ohledně dostupné RAM a CPU.

```
bp-turyna > scripts > $ getTotalSpecs.sh
1   #!/bin/bash
2   echo "Total available RAM:"
3   free -m | grep Mem | awk '{print $2}'
4   echo -e "\nTotal available CPU:"
5   lscpu | grep "^CPU(s):" | awk '{print $2}'
```

Obrázek 8.5: Script pro ověření dostupné RAM a CPU

3. **getSecondaryData.py** - Python script, který nejprve deklaruje cesty ke scriptům v rámci containeru a také vytváří .csv soubory, do kterých mají být sesbíraná data zapsána. Následně pravidelně po dobu 1sec spouští scripty **loadCpu.sh** a **loadMem.sh** (tyto script jsou popsány v kapitole Sekundární metriky) dokud neuplyne doba, se kterou byl script spuštěn (v případě obázku 8.6 se jedná o 60sec).

Obrázek 8.6: Python script pro sběr Sekundárních dat

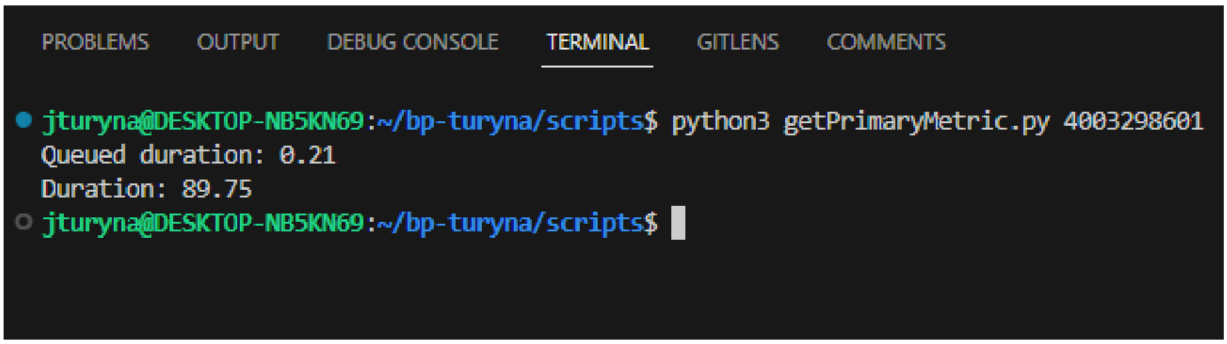
8.2 PRIMÁRNÍ METRIKY

Jakmile Pipeline doběhne, je možné provést sběr druhé části dat, která jsou v této práci označována jako tzv. Primární metriky.

Ty jsou sbírány odlišně, dle CI/CD poskytovatele za pomoci volně dostupného API. V tomto projektu byla tato data sbírána separátním spuštěním připraveného Python script, který má v případě Gitlabu vstupní parametr **jobId**.

Obrázek 8.7: Script pro sběr Primárních metrik - Gitlab

Takto například vypadá výstup třetího z desíti Jobů, který byl následně využit v následující kapitole 9 k vytvoření grafů a tabulek. Vstupem je pouze jobId = 4003298601 a výstupem jednoduchý zformátovaný print.



Obrázek 8.8: Výstup Scriptu pro sběr Primární metriky - Gitlab

9 GITLAB SHARED RUNNERS - PŘÍPRAVA A ZÍSKANÁ DATA

Shared runners je služba, kterou Gitlab nabízí všem uživatelům se základním přístupem ke Gitlabu. Shared runner běží přímo na strojích společnosti Gitlab a je dostupný pro kohokoliv se základní verzí Gitlab účtu.

Aby nedošlo ke zneužívání těchto volně dostupných runnerů, Gitlab implemenoval tzv. „fair usage queue algorithm“, který zamezuje, aby jeden uživatel nevypotřeboval všechny dostupné prostředky pouze pro sebe. [Git23b]

9.1 VÝSLEDKY SCRIPTU GETTOTALSPECS.SH

Oficiální informace ohledně dostupných prostředků pro Gitlab Shared runners se nepodařilo dohledat, a proto byl k ověření prostředků spuštěn script, který byl popsán výše v části Parametr Script.

Výsledkem scriptu je následující výpis, na kterém je možné vidět, že Shared runner má k dispozici 3,685GB RAM a 1 CPU:

```
148 $ ./scripts/getTotalSpecs.sh
149 Total available RAM:
150 3685
151 Total available CPU:
152 1
```

Obrázek 9.1: Dostupná RAM a CPU u Shared Runners

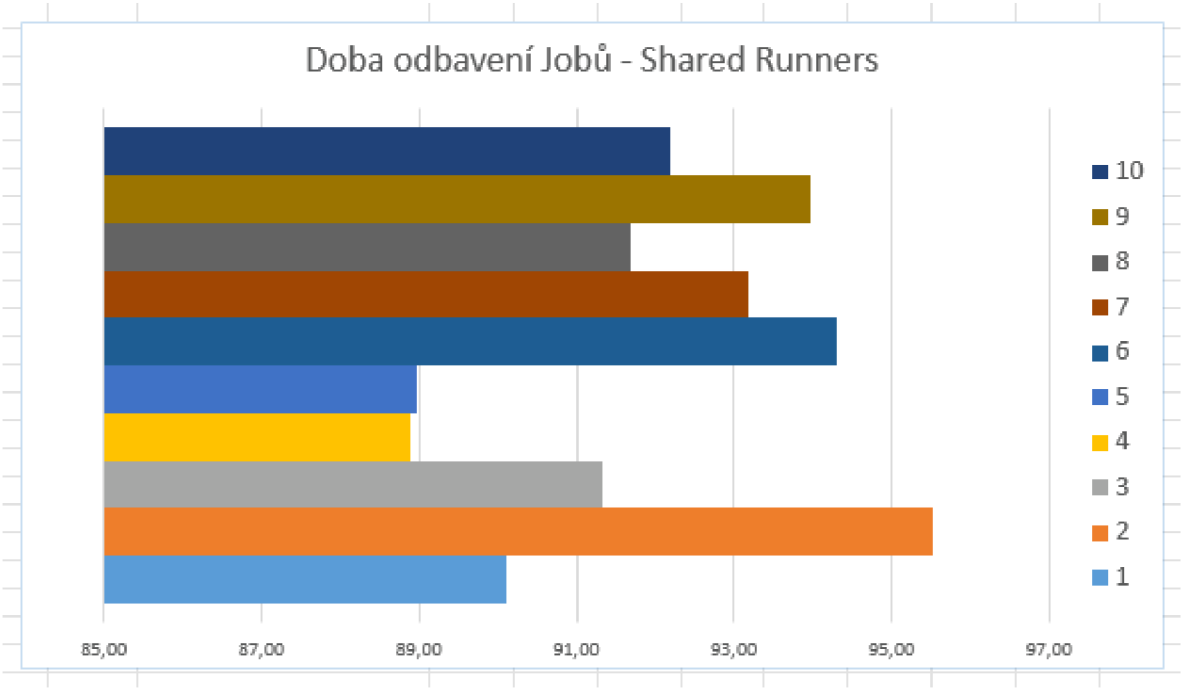
9.2 GITLAB SHARED RUNNERS - VÝSLEDKY MĚŘENÍ

9.2.1 Primární metriky - Duration

První měřenou metrikou je doba odbavení všech jednotlivých Jobů. Vzhledem k tomu, že se Joby odbavovaly pomocí služby Gitlab Shared Runners, nebylo možné pro tyto Joby nastavit Concurrency ani prostředí. Joby byli tedy odbaveny na stroji třetí strany, který měl 1 CPU a 4GB RAM. Zároveň byla data sbírána v rámci dvou skupin po pěti Jobech.

Níže na obrázku 9.2 je možno vidět graf doby odbavení jednotlivých Jobů a dále v tabulce 9.1 je možno vidět celkovou dobu odbavení a také průměrnou dobu odbavení napříč všemi deseti Joby.

Zajímavým faktorem je rozdíl mezi nejdéle trvajícím Jobem, č. 4. a nejrychlejším Jobem, č. 2, který činí 6,61 sec což je velmi stabilní a spolehlivý výsledek odbavování Jobů.



Obrázek 9.2: Graf naměřených hodnot celkové doby odbavení Jobů z Shared Runners

9.2.2 Primární metriky - Queue

Druhá metrika se zaměřuje na zpoždění jednotlivých Jobů v rámci Shared Runners. Opět vzhledem k tomu, že se jednalo o dostupnou službu od Gitlabu,

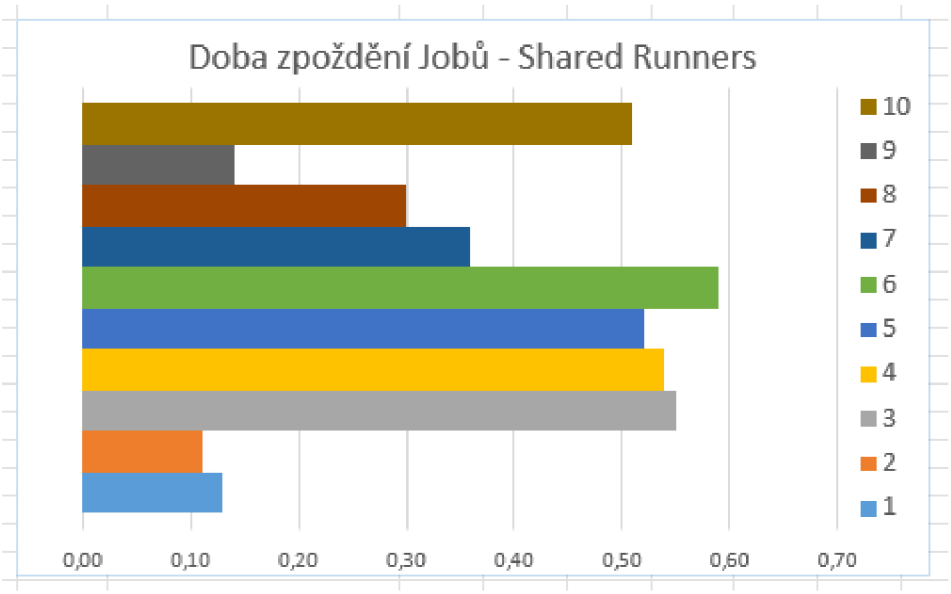
Skupina	1					2				
Job	1	2	3	4	5	6	7	8	9	10
Odbavení (sec)	90,11	95,51	91,33	88,9	88,97	94,30	93,19	91,68	93,97	92,18
Průměr (sec)	92,01									
Celkem (sec)	920,14									

Tabulka 9.1: Tabulka zprůměrovaných hodnot celkové doby odbavení Jobů z Shared Runners

nebyla možná konfigurace výpočetního výkonu prostředí ani parametru Concurrency. Odbavení Jobů proběhlo v prostředí s dostupným výkonem 1 CPU a 4GB RAM a byly spuštěny ve dvou skupinách po pěti Jobech.

Na obrázku 9.3 je možné vidět graf zpoždění všech desíti Jobů, které se pohybuje na velmi zanedbatelné hranici a ani u jednoho Jobu zpoždění nepřekročilo hranici jedné sekundy.

Dále v tabulce 9.2 je vidět průměrné zpoždění všech desíti Jobů a také celkové zpoždění Jobů obou skupin, které se rovná 3,75 sekundám.



Obrázek 9.3: Graf naměřených hodnot doby zpoždění jednotlivých Jobů z Shared Runners

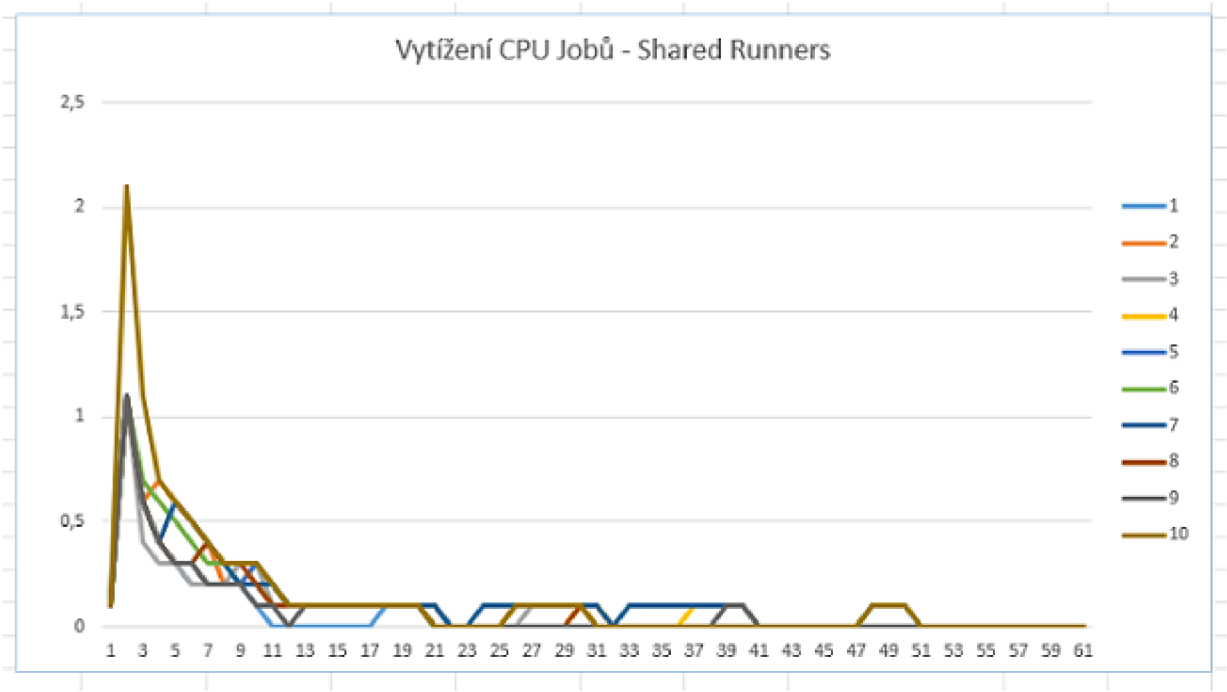
Skupina	1					2				
Job	1	2	3	4	5	6	7	8	9	10
Zpoždění (sec)	0,13	0,11	0,55	0,54	0,52	0,59	0,36	0,30	0,14	0,51
Průměr (sec)	0,37									
Celkem (sec)	3,75									

Tabulka 9.2: Tabulka zprůměrované doby zpoždění jednotlivých Jobů z Shared Runners

9.2.3 Sekundární metriky - CPU

Třetí metrikou je měření CPU po dobu jedné minuty na Shared Runners. Na obrázku 9.4 je možno vidět výsledný graf naměřených hodnot v prostředí, kterému bylo přiděleno 1 CPU. Při startu měření je u každého Jobu vidět vyšší růst vytížení CPU, které však ani u jednoho nepřesahuje hranici 2,5%.

Dále je pak v tabulce 9.3 spočítáno průměrné zatížení CPU za 60 sekund a také celkový průměr napříč všemi desíti Joby.



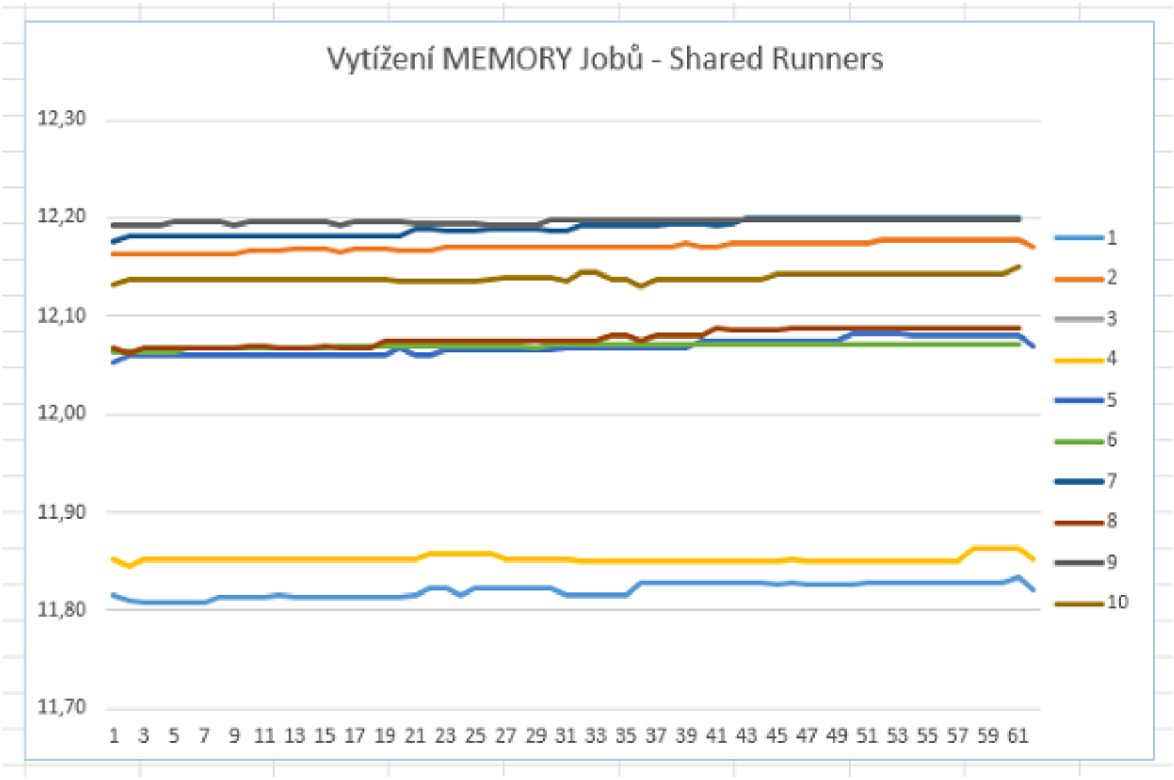
Obrázek 9.4: Naměřené hodnoty CPU z Shared Runners v čase 1 minuty

Skupina	1					2				
Job	1	2	3	4	5	6	7	8	9	10
Průměr CPU (%)	0,06	0,10	0,08	0,09	0,08	0,09	0,12	0,08	0,08	0,14
Celkem průměr CPU (%)	0,09									

Tabulka 9.3: Průměrné hodnoty CPU z naměřených Shared Runners

9.2.4 Sekundární metriky - Memory

Poslední měřenou metrikou na Shared Runners je využití Memory u jednotlivých Jobů v prostředí s přidělenou RAM 4GB. Graf s hodnotami za jednu minutu je možné vidět na obrázku 9.5. Níže je také spočítána tabulka 9.4 s průměrem jednotlivých Jobů a také s celkovým průměrem využití RAM napříč všemi desíti Joby.



Obrázek 9.5: Naměřené hodnoty Memory z Shared Runners v čase 1 minuty

Skupina	1					2				
Job	1	2	3	4	5	6	7	8	9	10
Průměr(%)	11,82	12,17	12,11	11,85	12,07	12,07	12,19	12,08	12,20	12,14
Celkem RAM(%)	12,07									

Tabulka 9.4: Průměrné hodnoty Memory z naměřených Shared Runners

10 GITLAB PRIVATE RUNNER - PŘÍPRAVA A ZÍSKANÁ DATA

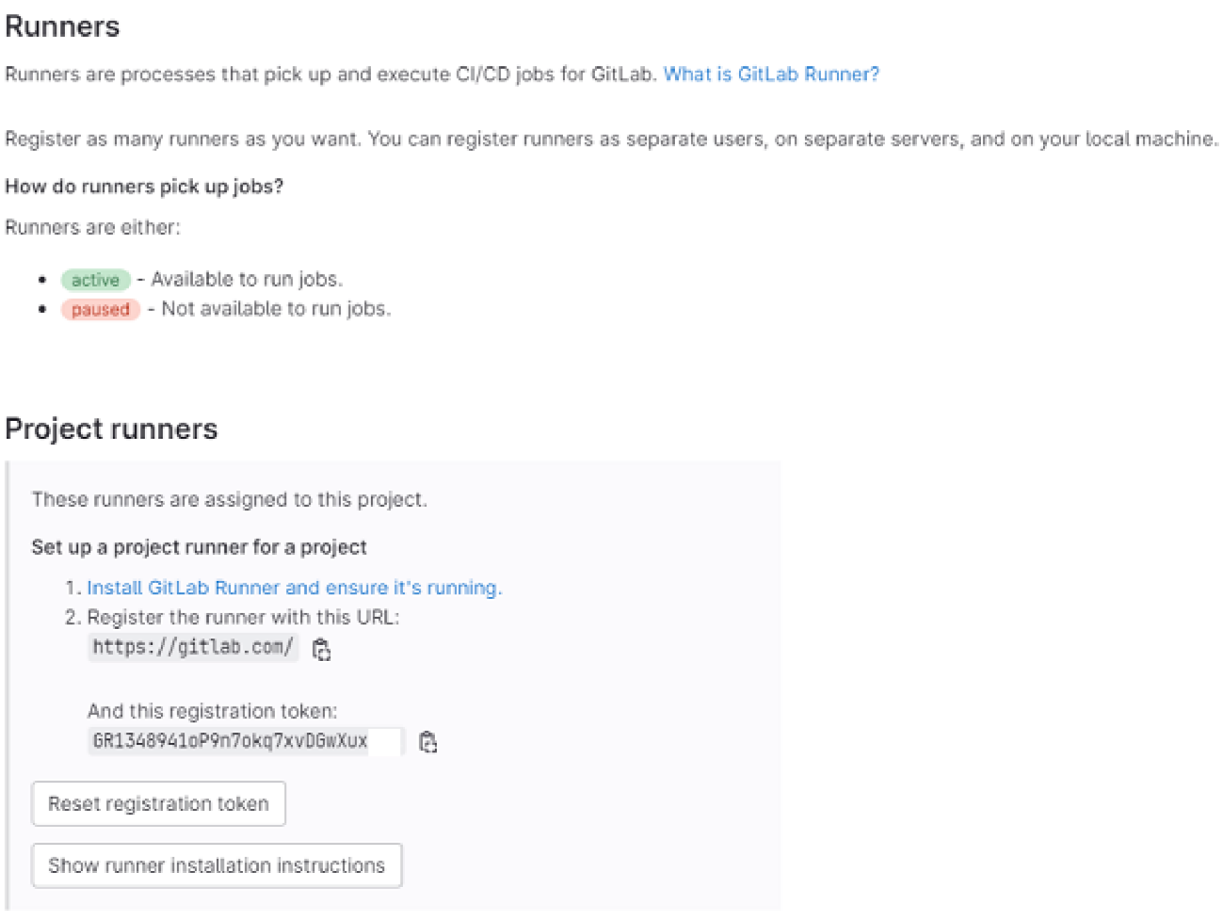
10.1 INSTALACE GITLAB RUNNERU

Prvním krokem k využití tohoto typu Runner je jeho samotná instalace. Prvním předpokladem je však přístup k některé z podporovaných Linuxových distribucí. Pro instalaci Private Runneru na OS Debian 11 v této Bakalářské práci byl použit postup z oficiální dokumentace Gitlabu. [1].

10.1.1 Registrace Runneru na konkrétním zařízení

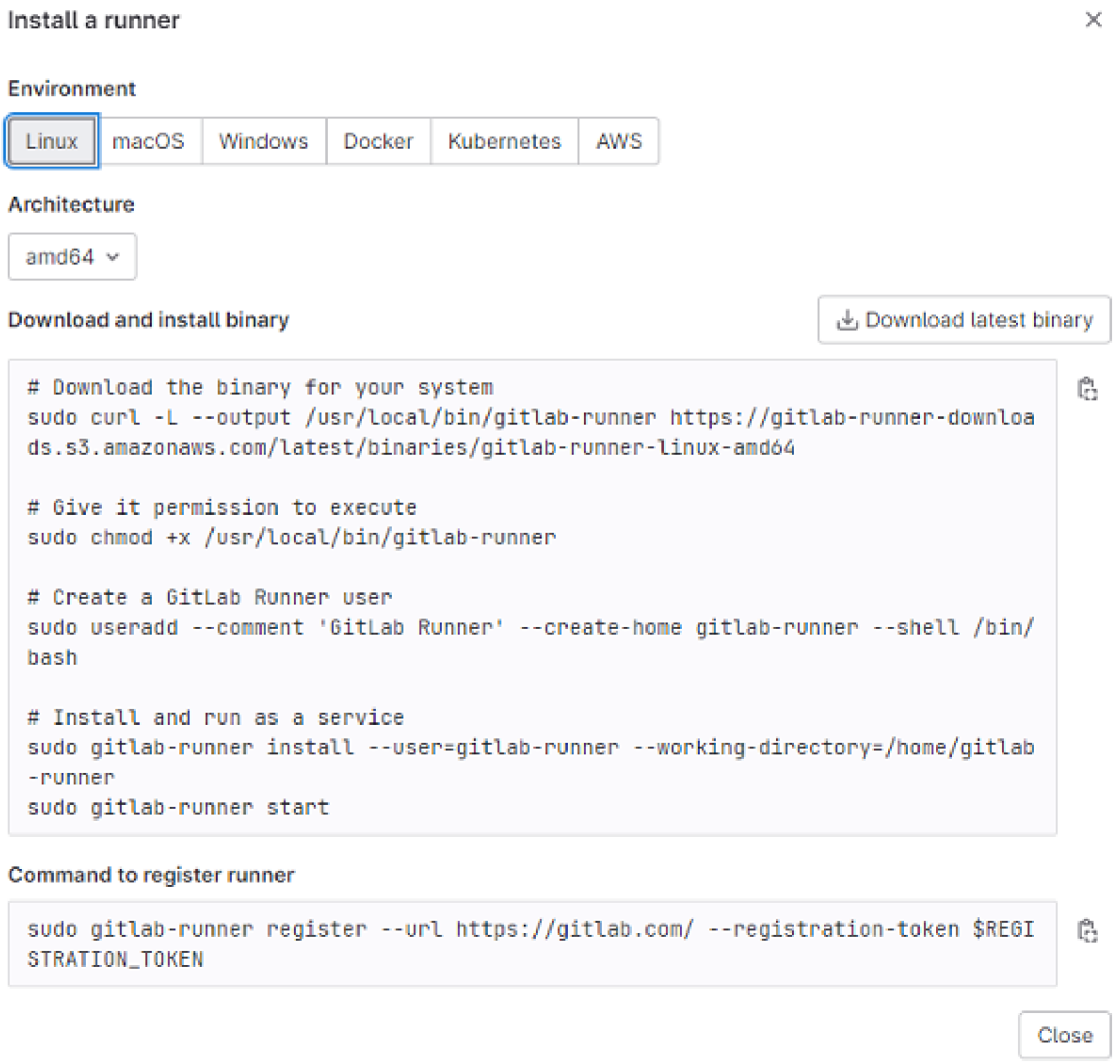
Jakmile proběhne úspěšná instalace Runneru, je potřeba provést jeho registraci ke konkrétnímu Gitlab projektu. Tato akce probíhá v záložce Settings/-CI/CD/Runners. Na této stránce je možné vidět informace o přidaných Runnerech k projektu a zároveň také URL a Token nutné k registraci Runneru (viz. obr. 10.1).

[1]Postup k instalaci Gitlab-Private Runneru, https://docs.gitlab.com/runner/install/linux-repository.html.



Obrázek 10.1: Screen URL a Tokenu nutných k registraci runneru

Po získání obou parametrů URL a Token, je nutné rozkliknout možnost „Show runner installation instructions“, což zobrazí instalační instrukce pro zvolené prostředí a architekturu, na kterých má Private Runner běžet (viz. obrázek 10.2).



Obrázek 10.2: Instrukce pro prostředí Linux nutných k registraci runneru

Po dokončení všech instrukcí a správném zadání URL a Token dojde k úspěšné registraci Runneru na zvoleném zařízení. V takovém případě je Runner vidět v celkovém přehledu Runnerů u zvoleného projektu a je připraven k použití.

10.2 NASTAVENÍ ZÁKLADNÍCH PARAMETRŮ

10.2.1 Computing power

Jedná se nastavení konkrétního výpočetního výkonu pro Docker containery, které budou odbavovat Jobs. Nastavení computing power na zařízení kde byl registrován Gitlab Runner probíhá v souboru /etc/gitlab-runner/config.toml.

Na obrázku 10.3 je vidět konfigurace Private Gitlab Runneru s nastavením Memory „4GBs“ a CPUs „1“, kterou využíval pro odbavování Jobs na stroji s OS Debian 11.

etc > gitlab-runner > config.toml

```
concurrent = 5
check_interval = 0
shutdown_timeout = 0

[session_server]
  session_timeout = 1800

[[runners]]
  name = "DESKTOP-NB5KN69"
  url = "https://gitlab.com/"
  id = 22869488
  token = "WwAPCQMjRCS8g7NK6mRm"
  token_obtained_at = 2023-04-22T15:09:59Z
  token_expires_at = 0001-01-01T00:00:00Z
  executor = "docker"
  [runners.cache]
    MaxUploadedArchiveSize = 0
  [runners.docker]
    tls_verify = false
    image = "ubuntu:20.04"
    privileged = false
    disable_entrypoint_overwrite = false
    oom_kill_disable = false
    disable_cache = false
    volumes = ["/cache"]
    shm_size = 0
    memory = "4gb"
    cpus = "1"
```

Obrázek 10.3: Konfigurační soubor configtoml pro Private Runner

10.2.2 Concurrency

Jedná se o parametr, který lze také nastavit v konfiguraci /etc/gitlab-runner/config.toml (viz. první řádek na obrázku 10.3) a slouží ke specifikaci počtu Jobs budou odbavovány najednou. Tedy bude ušetřen čas zpoždění jednotlivých Jobs, které na sebe nemusí čekat za cenu většího využití dostupných prostředků stroje.

10.3 PRIVATE RUNNER - VÝSLEDKY MĚŘENÍ

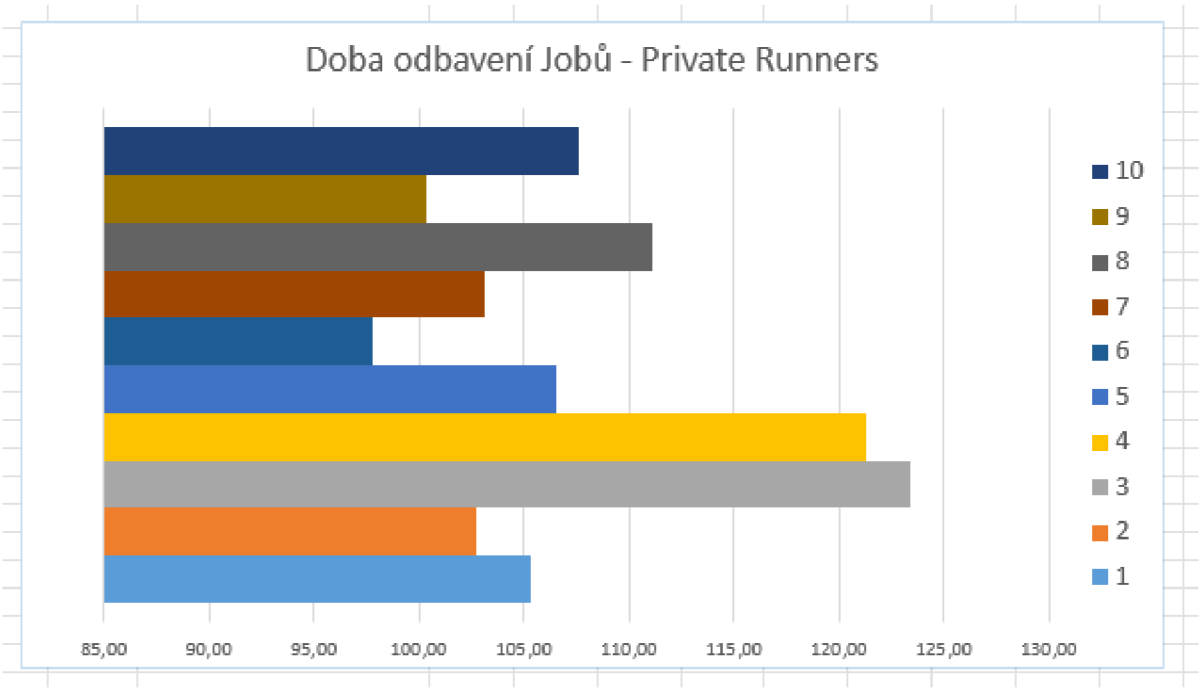
Následující kapitola zobrazuje naměřené sekundární a primární metrik Gitlab-Private Runneru s nastavením 1 CPU a 4Gb Memory. Tyto hodnoty byly zvoleny, aby odpovídali s neměnným nastavením výpočetního výkonu u Gitlab-Shared Runners. Vzhledem k faktu, že Private Runner také využívá platformu Gitlab, je jeho konfigurace Pipeline i sběr metrik totožná s konfigurací Gitlab-Shared Runners, která byla popsána v kapitole Sekundárních metriky a konfigurace Pipeline.

10.3.1 Primární metriky - Duration

První metrikou je opět doba odbavení jednotlivých Jobs v prostředí s výpočetním výkonem 1 CPU a 4Gb Memory. Zároveň jelikož se jedná o Private runner, je možné nastavit parametr Concurrency a tím umožnit odbavování všech Jobů najednou. Nejprve tedy došlo k odbavení prvních pěti Jobů v rámci skupiny 1 a následně nezávisle druhých pět Jobů v rámci skupiny 2.

Na obrázku 10.4 je vidět graf doby odbavení všech deseti Jobů a dále v tabulce 10.1 je zobrazena tabulky s průměrem a celkovou dobou odbavení.

Zajímavým faktem je, že rozdíl mezi nejrychlejším, č. 6, a nejpomalejším Jobem, č. 3, byl 25,64 sekund. Což je přesně o 19,03 sekund větší rozdíl než u Gitlab-Shared runners.



Obrázek 10.4: Graf naměřených hodnot celkové doby odbavení Jobů z Private Runneru

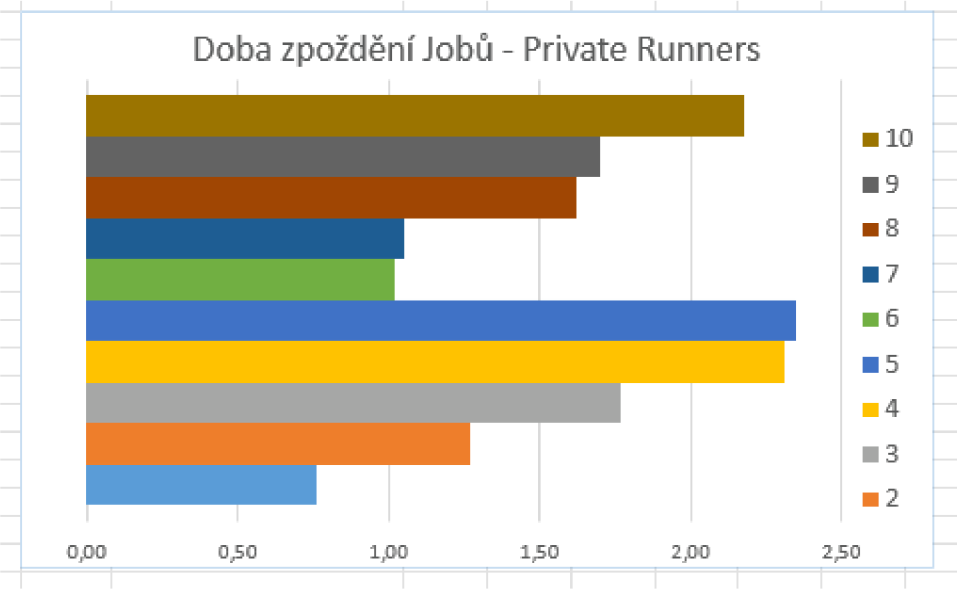
Skupina	1					2				
Job	1	2	3	4	5	6	7	8	9	10
Odbavení (sec)	105,32	102,73	123,37	121,32	106,57	97,73	103,14	111,15	100,36	107,57
Průměr (sec)	107,92									
Celkem (sec)	1079,26									

Tabulka 10.1: Tabulka zprůměrovaných hodnot celkové doby odbavení Jobů z Private Runneru

10.3.2 Primární metriky - Queue

Další měřenou metrikou bylo opět zpoždění jednotlivých Jobů. Naměřená data je možné vidět vykreslená v grafu 10.5 a průměr s celkovým zpožděním je dále uveden v tabulce 10.2.

Zde je vidět opět zhoršení oproti Gitlab-Shared runners kde celkové zpoždění Jobů bylo 3,75 sekund a zde bylo naměřeno 16,03 sec. V přepočtu to znamená zhoršení o 23% což může být znatelný rozdíl při spouštění náročnějších Jobs.



Obrázek 10.5: Graf naměřených hodnot doby zpoždění jednotlivých Jobů z Private Runneru

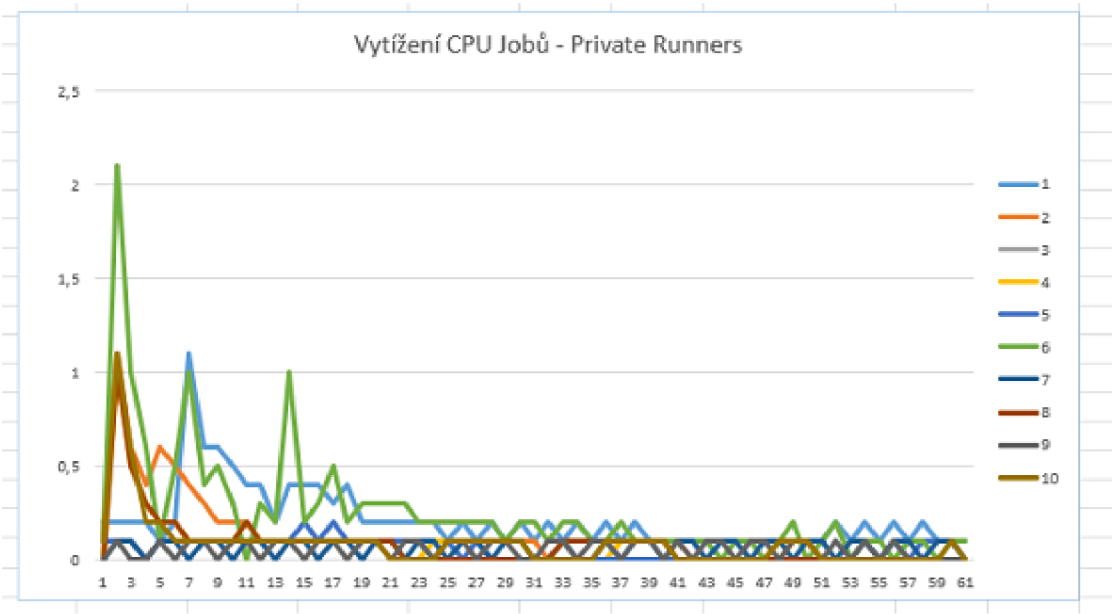
10.3.3 Sekundární metriky - CPU

Předposlední metrikou Gitlab-Private runner jsou naměřené hodnoty CPU za čas 1 minuty v prostředí s přiděleným 1 CPU. Opět následuje vykreslený graf 10.6 společně s tabulkou 10.3 s průměrem jednotlivých Jobů i celkovým průměrem všech desíti Jobů.

Skupina	1					2				
Job	1	2	3	4	5	6	7	8	9	10
Odbavení (sec)	0,76	1,27	1,77	2,31	2,35	1,02	1,05	1,62	1,70	2,18
Průměr (sec)	1,60									
Celkem (sec)	16,03									

Tabulka 10.2: Tabulka zprůměrované doby zpoždění jednotlivých Jobů z Private Runneru

V této metrice vychází Gitlab-Private runner velmi podobně, jako Gitlab-Shared runner a není zde vidět žádný větší výkyv vytížení CPU.



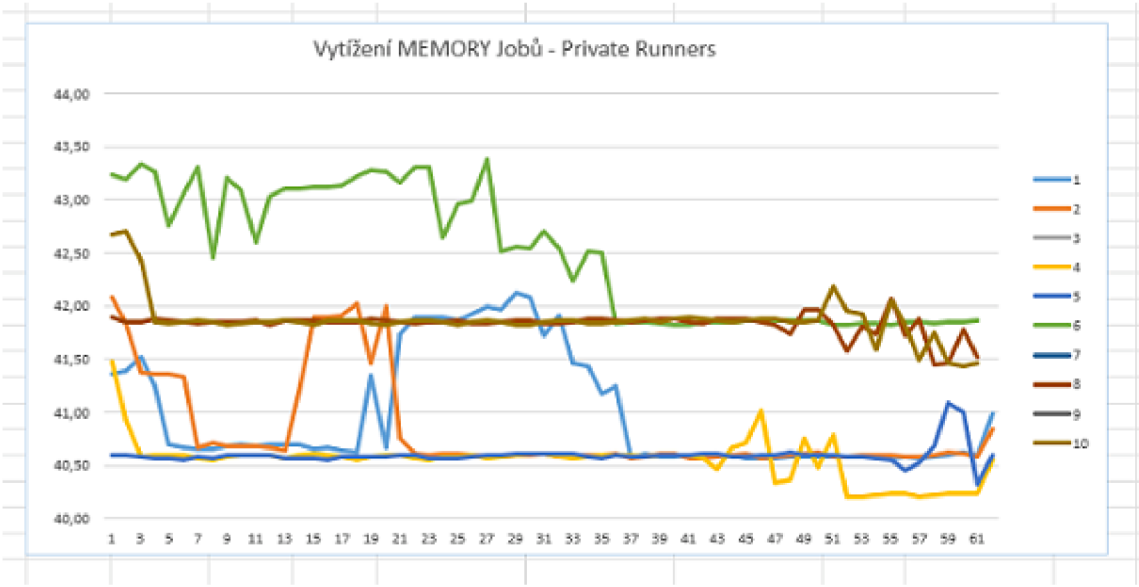
Obrázek 10.6: Naměřené hodnoty CPU z Private Runneru v čase 1 minuty

Skupina	1					2				
Job	1	2	3	4	5	6	7	8	9	10
Průměr CPU (%)	0,21	0,13	0,07	0,08	0,05	0,25	0,00	0,08	0,06	0,09
Celkem průměr CPU (%)	0,1									

Tabulka 10.3: Průměrné hodnoty CPU z naměřeného Private Runneru

10.3.4 Sekundární metriky - Memory

Poslední metrikou v rámci měření Gitlab-Private runner je vytížení RAM za 1 minutu v prostředí s přidělenými prostředky 4Gb RAM. Změřené hodnoty jsou k vidění na grafu 10.7 a průměr jednotlivých jobů společně s celkovým průměrem v tabulce 10.4.



Obrázek 10.7: Naměřené hodnoty Memory z Private Runneru v čase 1 minuty

Skupina	1					2				
Job	1	2	3	4	5	6	7	8	9	10
Průměr(%)	40,98	40,84	40,51	40,55	40,60	42,49	41,87	41,83	41,87	41,87
Celkem RAM(%)	41,4									

Tabulka 10.4: Průměrné hodnoty Memory z naměřeného Private Runneru

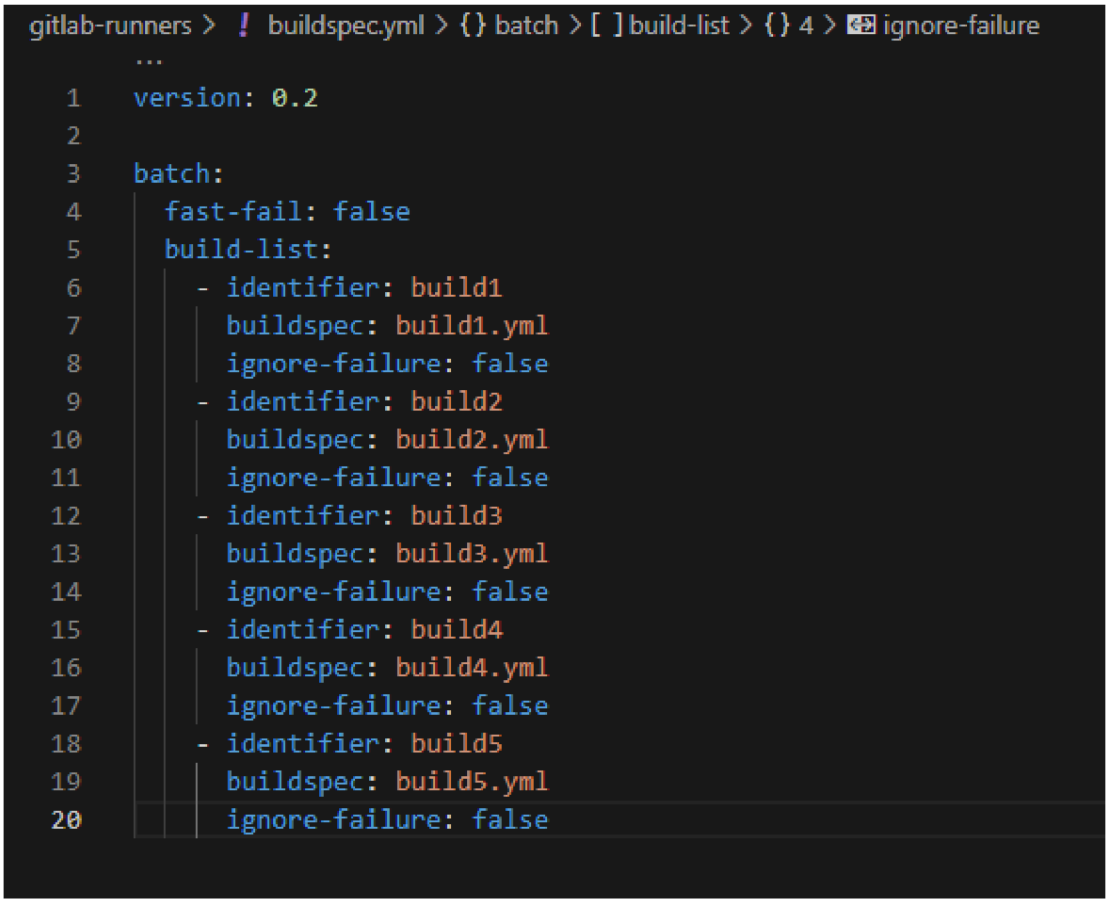
11 AWS RUNNER

11.1 ÚVOD

Ke sbírání dat z AWS byla využita kombinace několika AWS služeb, které slouží k implementaci CI/CD systému. Konkrétně se jedná o služby: CodeCommit, který funguje jako klasický repozitář a je v něm umístěn Code aplikace. CodeBuild, který kompiluje Code ze zadaných zdrojů (například CodeCommit), spouští Joby a testy a umožňuje také škálovatelnost služby pomocí Concurrency. AWS CloudShell, webový terminál, který byl využit pro sběr Primárních metrik. Poslední využívanou službou, která byla v rámci AWS využita je S3, do které se ukládají Artifacty z Jobů. V tomto případě jsou to soubory s daty sekundárních metrik ve formátu .csv.

11.2 SEKUNDÁRNÍ METRIKY

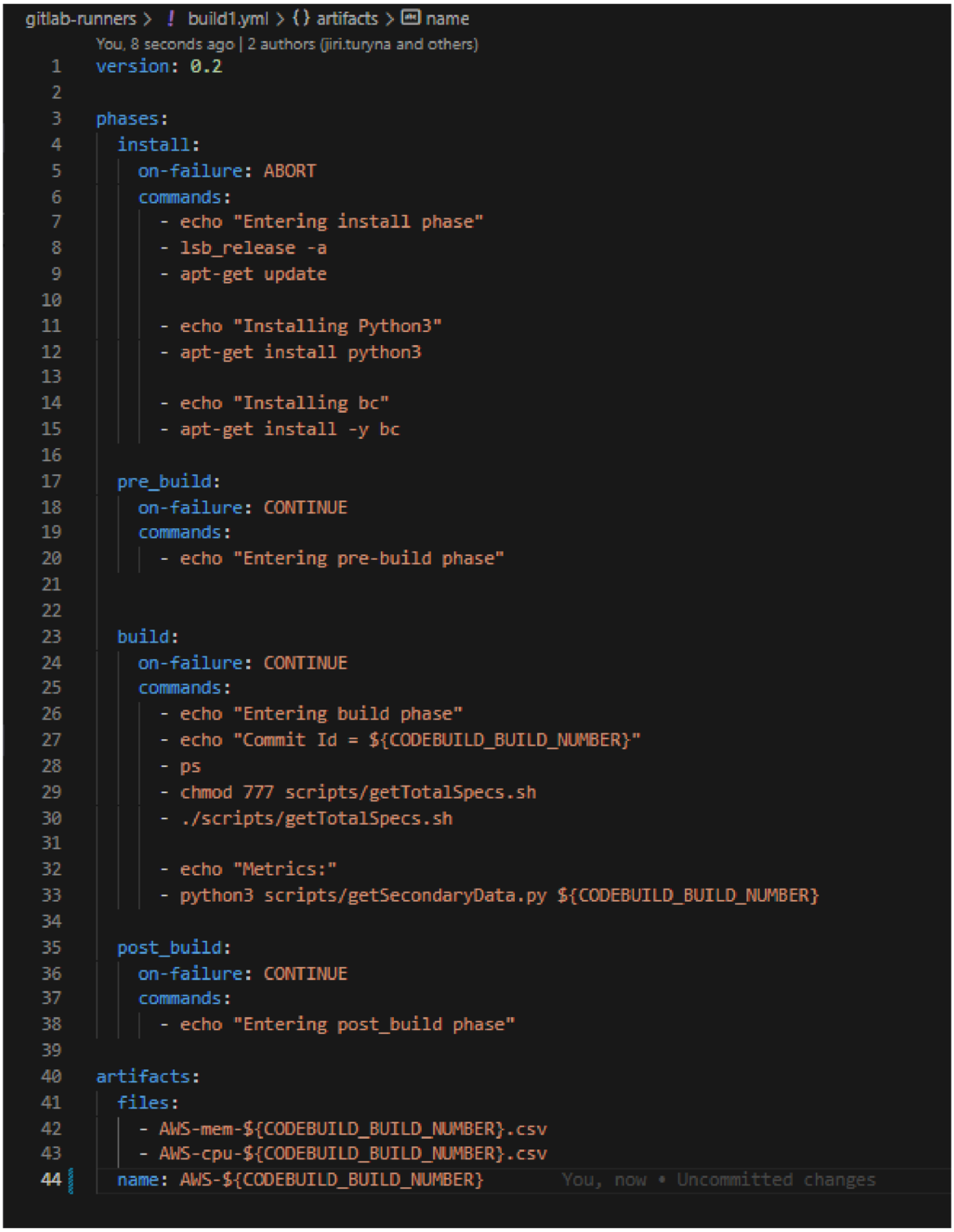
Na rozdíl od Gitlabu využívá CodeBuild pro definici Jobs a Pipeline soubor buildspec.yml (viz. obrázek 11.1), ve kterém je zvolen typ „batch“, což umožňuje, aby jednotlivé Joby běžely souběžně a tím nedocházelo ke značnému zpoždění. Dále je v buildspec.yml souboru parameter „build-list“, který obsahuje seznam build.yml souborů, což jsou definice jednotlivých Jobs, které mají být odbaveny. [Ama23b]



Obrázek 11.1: AWS buildspec.yml

Aby bylo možné docílit spuštění pěti souběžných Jobů najednou (concurrency), bylo nutné obrátit se na AWS Support, jelikož tato funkce není ve Free Tier účtu dostupná. Následně už bylo možné v rámci projektu parametr Concurrency nastavit v požadovaném rozsahu.

Na obrázku 11.2 je vidět definice jednoho z Jobů, které sbírají Primární data. V tomto případě je všech pět build.yml souborů totožných.



Obrázek 11.2: AWS build.yml

Na rozdíl od Gitlabu musí v AWS CodeBuild konfigurace Jobu obsahovat čtyři povinné fáze - install, pre_build, build a post_build. [Ama21]

V install fázi Jobu dochází k instalaci všech potřebných balíčků (v tom

případě python3 a bc).

Fáze pre_build slouží k definici příkazů, které mají proběhnout ještě před samotnou build fází. Obvykle slouží k přihlašování se ke službám nebo generaci environment proměnných. V rámci tohoto projektu není pre_build fáze využívána.

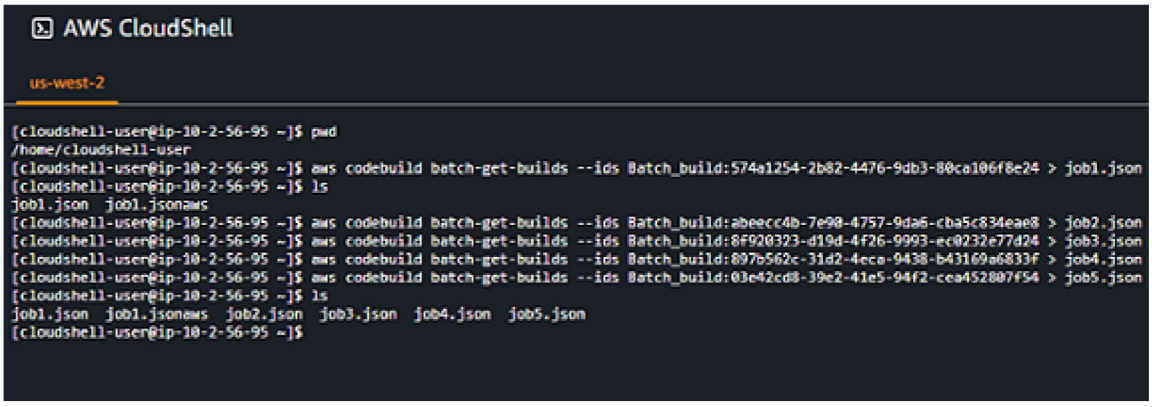
Hlavní fází každého Jobu je Build fáze. Ta definuje hlavní úkol Jobu. V tomto případě, naprosto stejně jako u Gitlabu, probíhá nastavení spustitelných práv pro script getTotalSpecs.sh (funkce popsána zde 1), jeho spuštění a následně také zapnutí samotného scriptu getSecondaryData.py, který generuje .csv soubory ohledně využití CPU a RAM.

Post_build je v tomto projektu také ponechána prázdná a obvykle slouží k pushování Artifactů nebo Docker imagí, které byly vytvořeny v build fázi.

Dále je ještě na obrázku 11.2 vidět nepovinný parametr „artifact“, ve kterém jsou definovány .csv soubory, které vznikly v build fázi ohledně CPU a RAM. Tímto parametrem dochází k jejich uložení do služby S3. Pokud by byl parametr vynechán, tyto soubory by zanikly společně s dokončením Jobu.

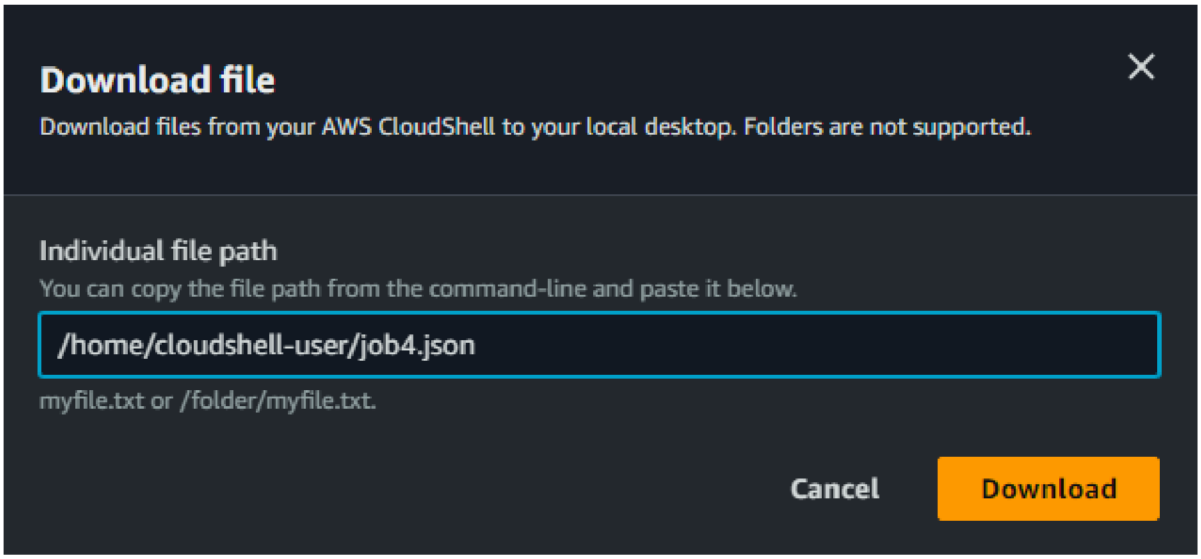
11.3 PRIMÁRNÍ METRIKY

Ke sběru primárních metrik bylo potřeba nejprve získat .json soubory jednotlivých Jobs. K tomu byly využita služba AWS CloudShell, což je webový terminál v rámci AWS. Nejprve došlo k využití funkce batch-get-builds a načtení dat na základě jednotlivých Batch_build id, která byla následně uložena do .json souborů (viz. obrázek 11.3). [Ama23a]



Obrázek 11.3: AWS cloudshell

Finálním krokem bylo stažení takto vytvořených souborů přes možnost „Download“ (viz. obrázek 11.4).



Obrázek 11.4: AWS cloudshell download

Jakmile byly všechny .json soubory ohledně Primárních metrik staženy na lokálním počítači, vznikl Python script k jejich parsování, aby bylo snadné metriky zobrazovat. Tento script je zobrazen na obrázku 11.5. Výstupem tohoto scriptu jsou data ohledně zpoždění a celkové doby odbavení Jobů, které jsou zaneseny v tabulkách a grafech v kapitole AWS - Výsledky měření.

Obrázek 11.5: AWS primary metrics script

11.4 ZÁKLADNÍ NASTAVENÍ AWS RUNNERU V CODEBUILD

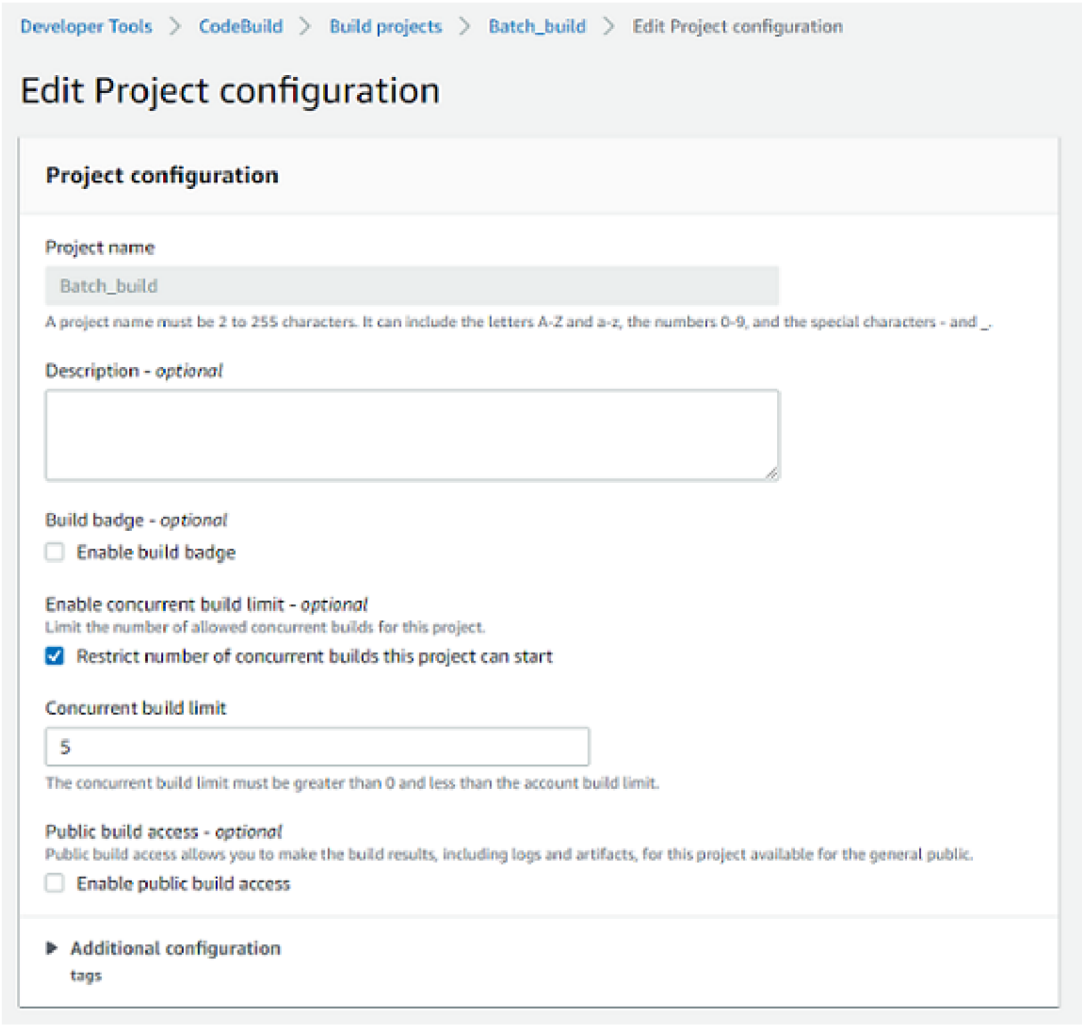
V této sekci je zobrazeno nastavení projektu v rámci CodeBuild, který byl využit ke sběru dat ohledně Jobů. Sekce je rozdělena do šesti částí, které jsou v CodeBuild dostupné k nastavení. U každé části je zobrazeno vysvětlení nastavení v rámci části a následně i zvolené nastavení, které bylo pro sběr metrik využito.

11.4.1 Project configuration

Hlavní část projektu, ve které se dá nastavovat jeho název, popis, tagy a další. Z pohledu sběru dat však byly nejdůležitější možnosti „Enable concurrent build limit“ a „Concurrent build limit“, což následně umožňovalo spuštění všech pěti Jobů najednou a tím došlo k zajištění, že data budou porovnatelná s předchozími poskytovateli CI/CD.

Pro zajímavost, u tohoto kroku bylo nutné kontaktovat AWS Support, jelikož s Free Tier verzí AWS účtu, na kterém byl test a sběr dat prováděn, nebylo

možné nastavit parametr „Concurrent build limit“ na více než dva souběžně běžících Jobs najednou a tím nebylo možné data porovnávat s Gitlab runnery. Po vysvětlení situace byla tato možnost v rámci projektu povolena.

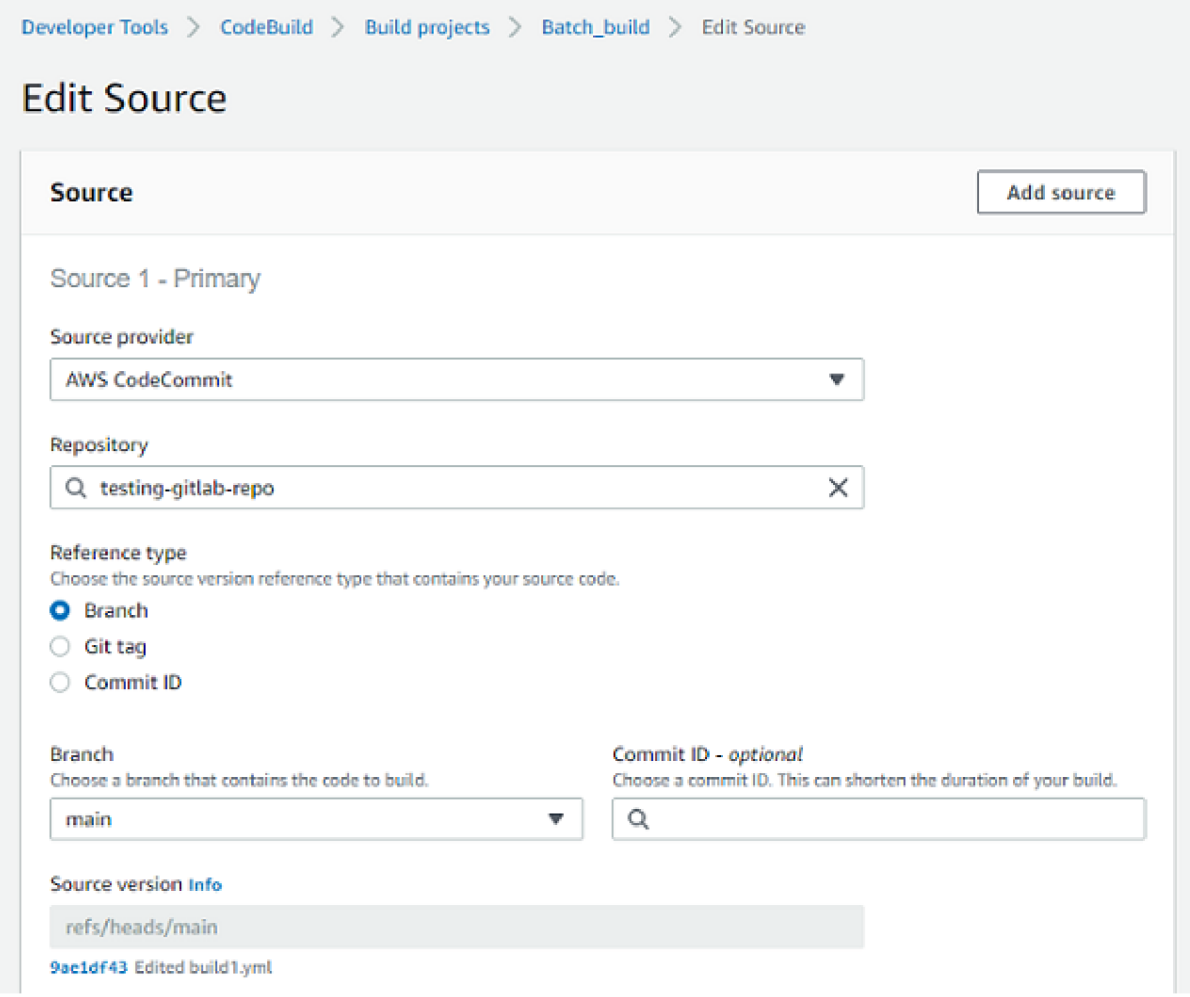


Obrázek 11.6: Nastavení projektu

11.4.2 Source

V části Source (viz. obrázek 11.7) dochází k definici zdroje Codu, který má být při buildu Jobů spuštěn. Jelikož v době vytváření projektu nebylo možné využít Gitlab, byl jakožto „Source Provider“ zvolen AWS CodeCommit. Jedná se o službu v rámci AWS, která, podobně jako Github a Gitlab, dovoluje ukládání a nahrávání Codu do repozitářů.

Následně je pouze potřeba zvolit konkrétní repozitář s uloženým Codem, nastavit možnost „Reference type“, která dovoluje verzování zdrojového Codu podle „Branch“, „Git tag“ nebo „Commit ID“. Pro usnadnění byl zvolen Reference type „Branch“ a následně také název této větvě „main“.



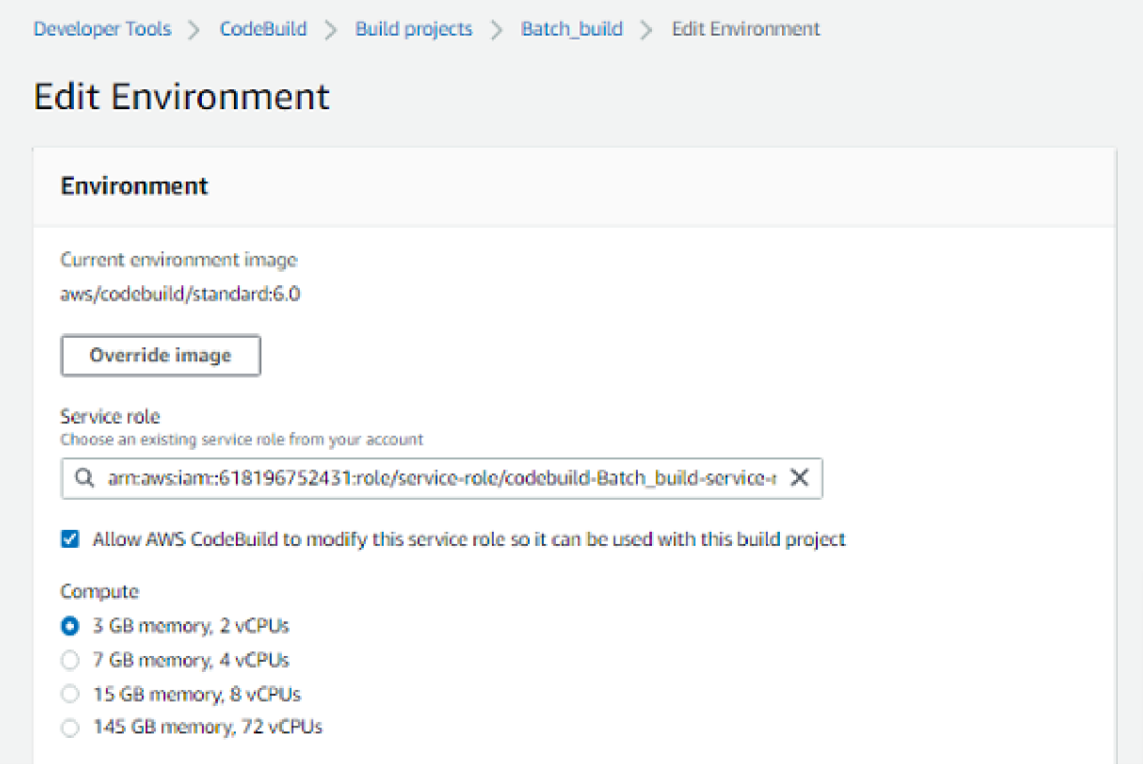
Obrázek 11.7: Definice zdrojového repozitáře

11.4.3 Environment

V této části je možné nastavení prostředí, ve kterém se mají Joby odbavovat. Jedná se o jeden ze dvou typů nastavení prostředí, který lze využít. Druhým je „Batch“ popsaný v jedné z následujících částí této kapitoly 11.4.5. Environment se tomto případě využívá k Buildu jednoho konkrétního projektu.

Prvním hlavním nastavitelným parametrem je „Current environment image“, což je image operačního systému, ve kterém bude container Jobu vystaven . Běžně využívané jsou Ubuntu, Amazon Linux 2 nebo Windows Server 2019. Na obrázku 11.8 je vidět zvolený image „aws/Codebuild/standard:6.0“, který odkazuje na image operačního systému Amazon Linux 2.

Druhou důležitou možností k nastavení je „Compute“, kde je možné vybrat si z několika předem zvolených možností výpočetního výkonu.

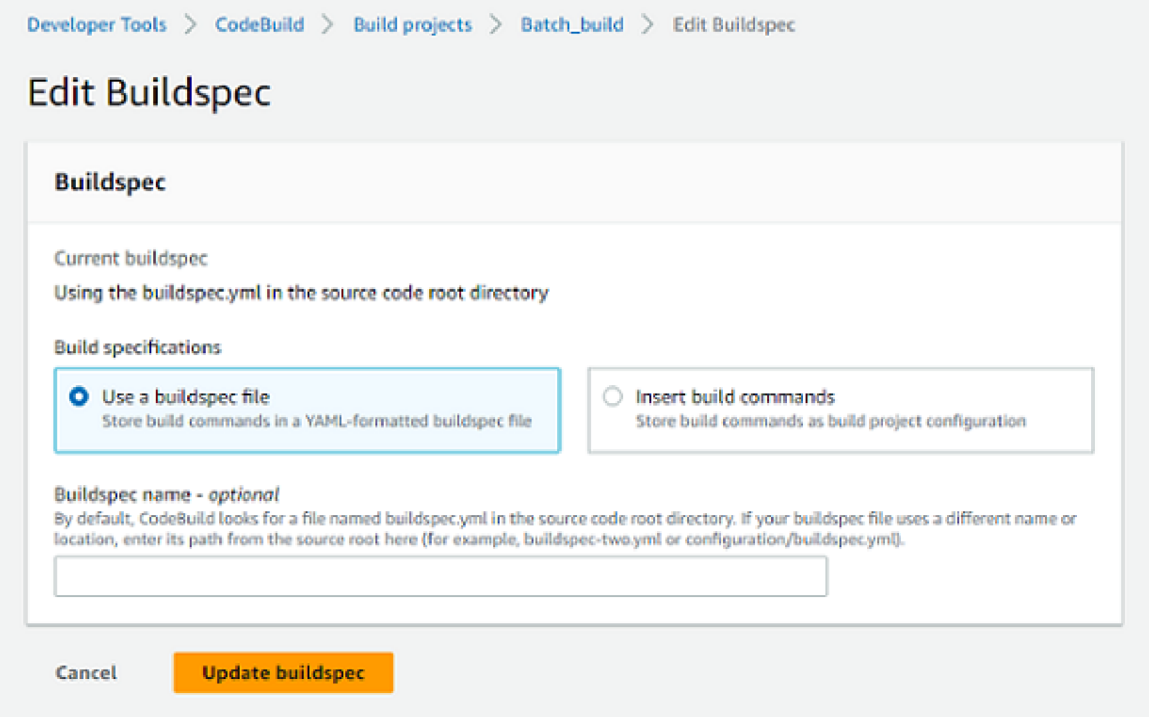


Obrázek 11.8: Konfigurace prostředí containeru

11.4.4 Buildspec

Další konfigurací projektu je Buildspec, jedná se o soubor s Codem, který definuje Joby a Pipelinu. Je psaný v YAML formátu a obvykle bývá uložený v kořenovém adresáři projektu. Alternativně lze v tomto nastavení projektu definovat jinou cestu k buildspec.yml souboru v adresáři, která by se od defaultního umístění mohla lišit. V případě popisovaného projektu bylo využito defaultní cesty k souboru.

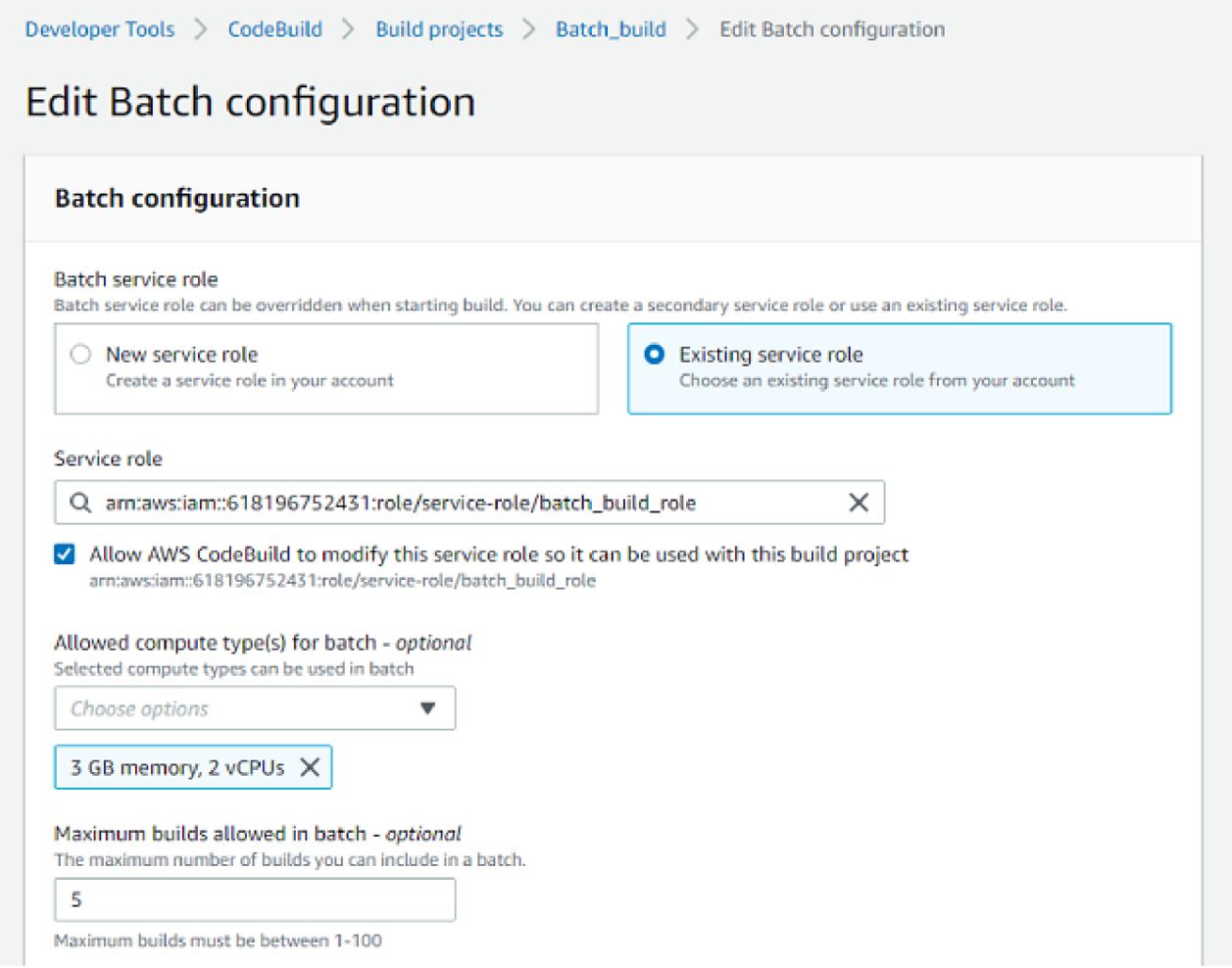
Konkrétní buildspec.yml soubor, který byl využit ke sběru dat je zobrazen na obrázku 11.9 a je popsán v kapitole 11.2.



Obrázek 11.9: Možnosti definice souboru Buildspec v projektu

11.4.5 Batch Configuration

V předposlední části konfigurace projektu je druhá možnost specifikace prostředí, ve kterém budou Joby odbaveny. V tomto případě se jedná o možnost Batch, tedy způsob, kterým jsou všechny Joby odbavovány najednou. Na obrázku 11.10 je vidět toto nastavení v praxi. Nejprve je nutné zvolit roli v rámci AWS IAM. V podstatě se jedná o uživatele, který má přidělená dostatečná oprávnění v AWS, aby Batch mohl spustit. Následně je zvolen výpočetní výkon a maximální počet buildů, které jsou povolena v konkrétním Batchi odbavovat.

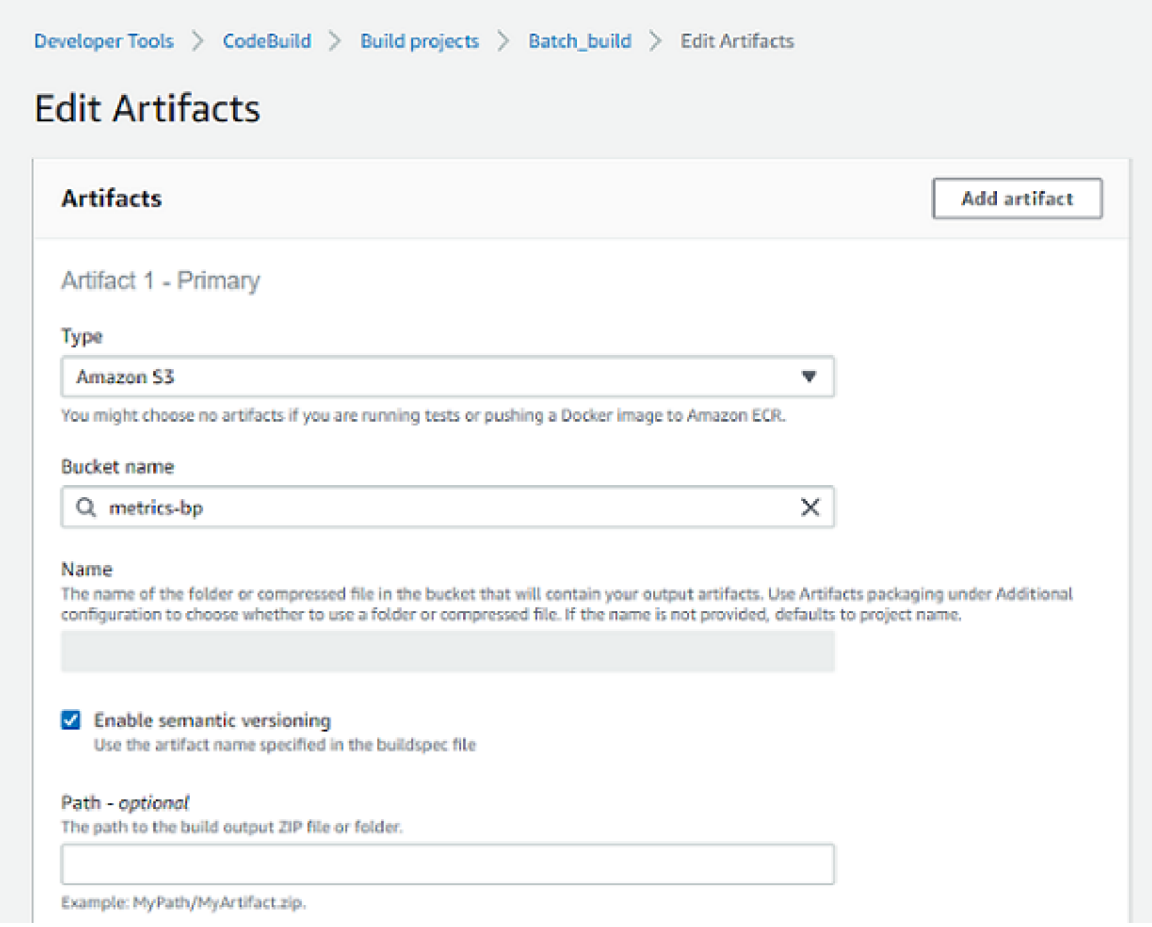


Obrázek 11.10: Nastavení možnosti batch

11.4.6 Artifacts

V poslední možné konfiguraci projektu je možné nastavovat úložiště pro Artifacty, nebo-li výstupní soubory Buildu. Na obrázku 11.11 je vidět praktické nastavení pro tento projekt. V první možnosti „Type“ je možné vybrat, zda se mají výstupní Artifacty nahrávat do Amazon S3 nebo zda z Buildu není požadováno ponechávat Artifacty. V takovém případě jsou takové soubory smazány s doběhnutím Buildu.

Dále, pokud je zvolen „Type“ Amazon S3, je nutné zvolit již existující bucket v AWS S3, do kterého mají být Artifacty nahrány. V poslední řadě je také možné určit název Artifactů nebo výsledné umístění v případě, že by měly být uloženy například ve složce.



Obrázek 11.11: Specifikace ukládání výstupních Artifactů

11.5 VÝSLEDKY SCRIPTU GETTOTALSPECS.SH

K ověření, že se Joby odbavují v prostředí s odpovídajícími prostředky, byl opět spuštěn script getTotalSpecs.sh, který je detailněji popsán v kapitole Parametr Script. Výstup tohoto scriptu v prostředí AWS je možné vidět na obrázku 11.12, kde je zobrazeno, že Job běží v prostředí s dostupnou RAM 3,7Gb a CPU 2, což neodpovídá konfiguraci Batch-config, která byla zobrazena a nastavena v 11.4.5.

```
180
181 [Container] 2023/03/26 22:41:59 Running command ./scripts/getTotalSpecs.sh
182 Total available RAM:
183 3731
184
185 Total available CPU:
186 2
187
```

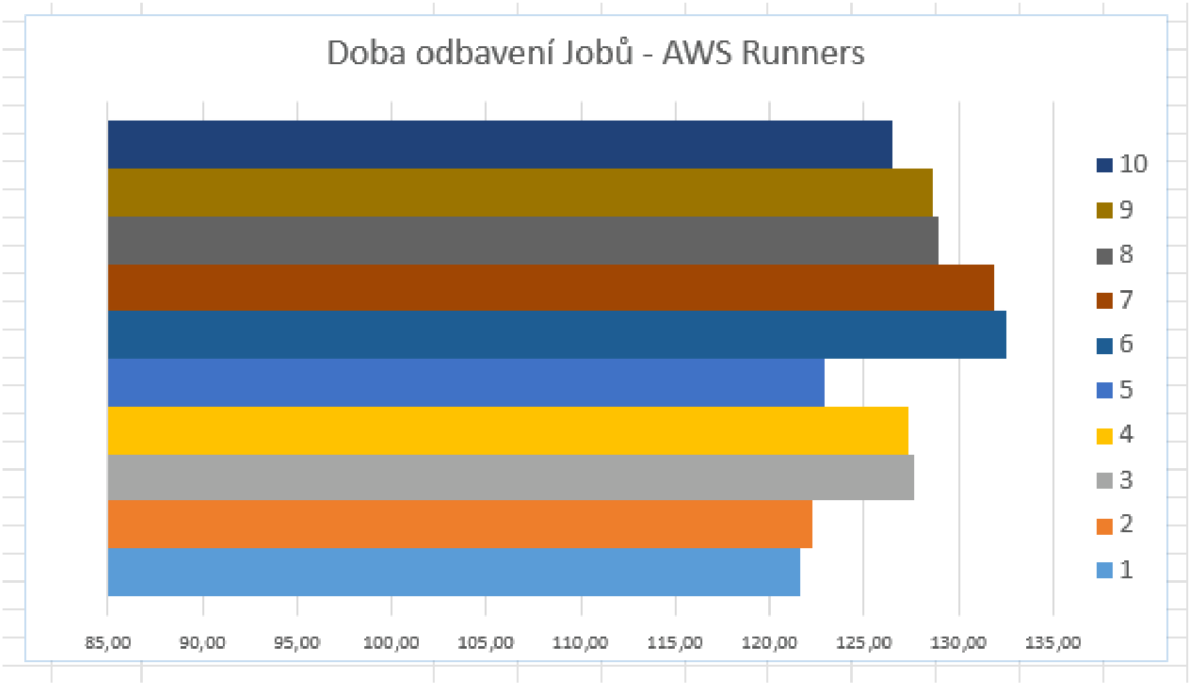
Obrázek 11.12: Výstup scriptu getTotalSpecs v prostředí AWS

11.6 AWS - VÝSLEDKY MĚŘENÍ

11.6.1 Primární metriky - Duration

První změřenou metrikou v AWS CodeBuild je opět doba odbavení všech desíti Jobů, rozdělená do dvou skupin po pěti Jobech, které se odbavovaly zároveň. Nastavené prostředí k odbavení bylo zvoleno nejbližší možné ke Gitlab-Shared runners (jelikož u této služby nelze libovolně s prostředky manipulovat). Každopádně po změření na obrázku 11.12 je však vidět, že nastavení Batch konfigurace neodpovídá realitě a Joby jsou odbavovány na prostředí 4Gb RAM a 2 CPU.

Na obrázku 11.13 je vidět graf vykreslených dob odbavení jednotlivých Jobů. Dále v tabulce 11.1 je spočítán průměr napříč všemi desíti Joby a také celková doba odbavení.



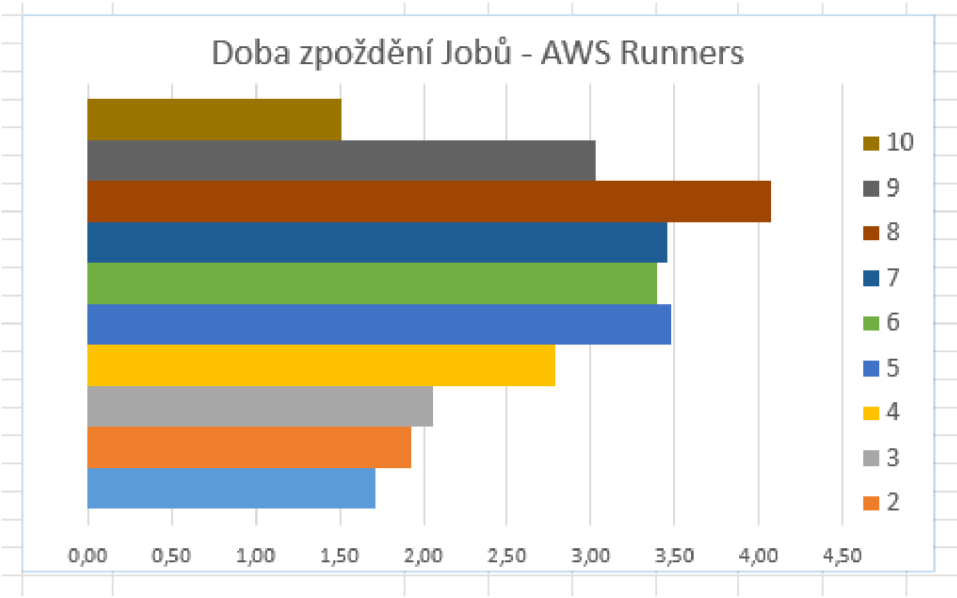
Obrázek 11.13: Graf naměřených hodnot celkové doby odbavení Jobů z AWS Runners

Skupina	1					2				
Job	1	2	3	4	5	6	7	8	9	10
Odbavení (sec)	121,59	122,31	127,71	127,33	122,90	132,50	131,89	128,98	128,57	126,48
Průměr (sec)	127,02									
Celkem (sec)	1270,26									

Tabulka 11.1: Tabulka zprůměrovaných hodnot celkové doby odbavení Jobů z AWS Runners

11.6.2 Primární metriky - Queue

Následuje graf 11.14 společně s tabulkou 11.2 naměřených hodnot ohledně zpoždění na AWS runners. Zajímavým faktem je, že průměrné zpoždění Jobu v rámci AWS je 2,74 sekundy což je několika násobně více než u Gitlab-Shared i Private runners. Celková doba odbavení všech desíti Jobů je tedy také daleko vyšší, než u předchozích poskytovatelů CI/CD, s hodnotou 27,45 sekund.



Obrázek 11.14: Graf naměřených hodnot doby zpoždění jednotlivých Jobů z AWS Runners

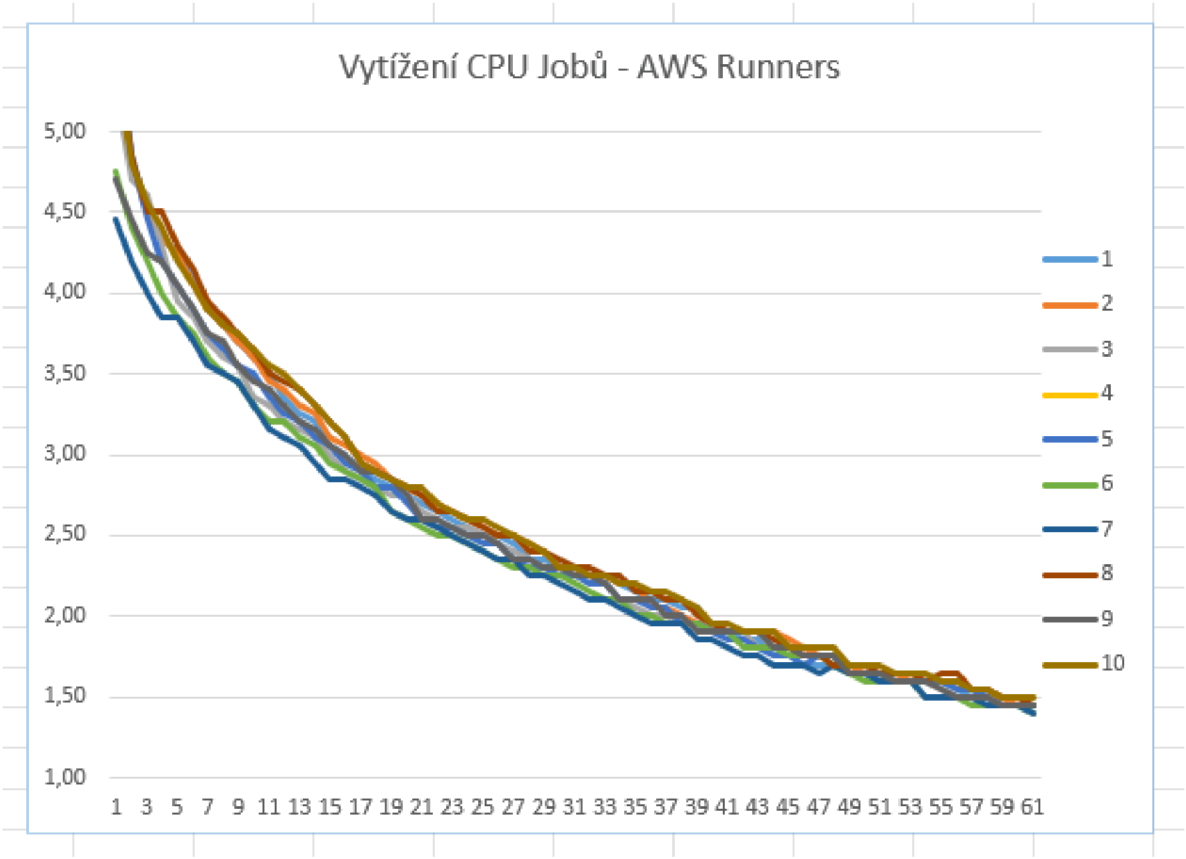
Skupina	1					2				
Job	1	2	3	4	5	6	7	8	9	10
Odbavení (sec)	1,71	1,93	2,06	2,79	3,48	3,40	3,46	4,08	3,03	1,51
Průměr (sec)	2,74									
Celkem (sec)	27,45									

Tabulka 11.2: Tabulka doby zpoždění jednotlivých Jobů z AWS Runners

11.6.3 Sekundární metriky - CPU

Stejně jako u předchozích poskytovatelů následují naměřené metriky ohledně vytížení CPU. Na obrázku 11.15 je vidět graf naměřených všech desíti Jobů za dobu jedné minuty. Následně pak tabulka 11.3 s průměrným CPU napříč jednotlivými Joby a také celkovým průměr napříč všemi Joby.

Pro porovnání, Gitlab-Shared runners a Gitlab-Private runners dosahovaly celkového průměru vytížení CPU zhruba 0,1%. AWS dosahuje se stejným počtem Jobů hodnoty 2,49% a navíc byli Joby odbavovány v prostředí s výpočetním výkonem 2 CPU (oba runnery Gitlabu měli nastaveno 1 CPU).



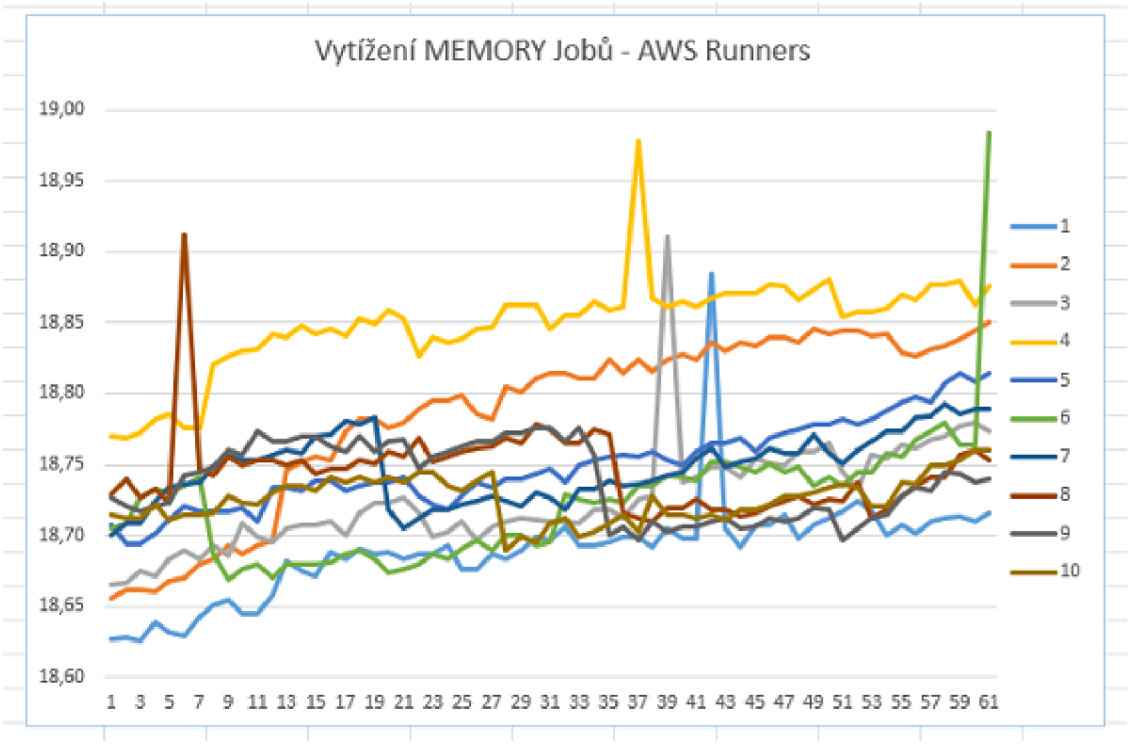
Obrázek 11.15: Naměřené hodnoty CPU z AWS Runners v čase 1 minuty

Skupina	1					2				
Job	1	2	3	4	5	6	7	8	9	10
Průměr CPU (%)	2,52	2,54	2,47	2,55	2,48	2,40	2,36	2,57	2,47	2,57
Celkem průměr CPU (%)	2,49									

Tabulka 11.3: Průměrné hodnoty CPU z naměřených AWS Runners

11.6.4 Sekundární metriky - Memory

Poslední metrikou jsou hodnoty jednotlivých Jobů z pohledu RAM. Opět je na obrázku 11.16 vidět průběh naměřených hodnot za 1 minutu a následně také tabulka s procentuálním průměrem vytížení RAM jednotlivých Jobů a následně i celkové vytížení RAM napříč všemi desíti Joby.



Obrázek 11.16: Naměřené hodnoty Memory z Shared Runners v čase 1 minuty

Skupina	1					2				
Job	1	2	3	4	5	6	7	8	9	10
Průměr(%)	18,69	18,79	18,73	18,85	18,75	18,72	18,75	18,74	18,74	18,73
Celkem RAM(%)	18,75									

Tabulka 11.4: Průměrné hodnoty Memory z naměřených AWS Runners

12 POROVNÁNÍ METRIK VŠECH CI/CD POSKYTOVATELŮ

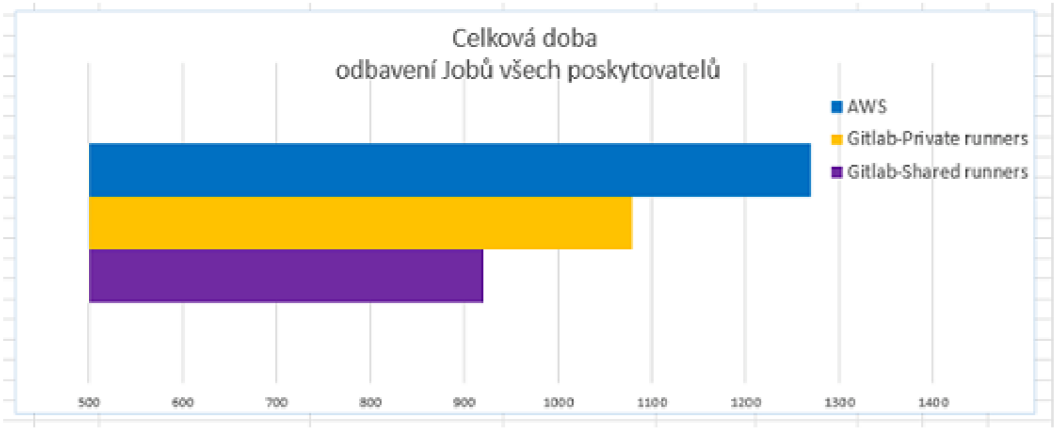
V Této kapitole je popsáno porovnání všech vybraných poskytovatelů CI/CD na základě změření všech metrik uvedených v předchozích kapitolách.

12.1 PRIMÁRNÍ METRIKY

12.1.1 Celková a průměrná doba odbavení Jobů

Nejprve je na obrázku 12.1 zobrazen graf ohledně celkové doby odbavení Joby na základě sečtení doby v rámci všemi desíti Joby. Celkově se nejlépe umístil poskytovatel Gitlab-Shared runners s celkovou dobou odbavení 920,14 sekund. Na druhém místě pak Gitlab-Private runner se 1079,56 sekundy a na posledním místě AWS runners, který na odbavení všech Jobů potřeboval celkem 1270,26 sekund i přesto, že měl přidělený o 1 CPU větší výkon než předchozí poskytovatelé.

Pro zajímavost je dále také připravena tabulka 12.1, kde je zobrazena také průměrná doba odbavení napříč všemi Joby dle poskytovatelů.



Obrázek 12.1: Graf naměřené celkové doby odbavení všech poskytovatelů

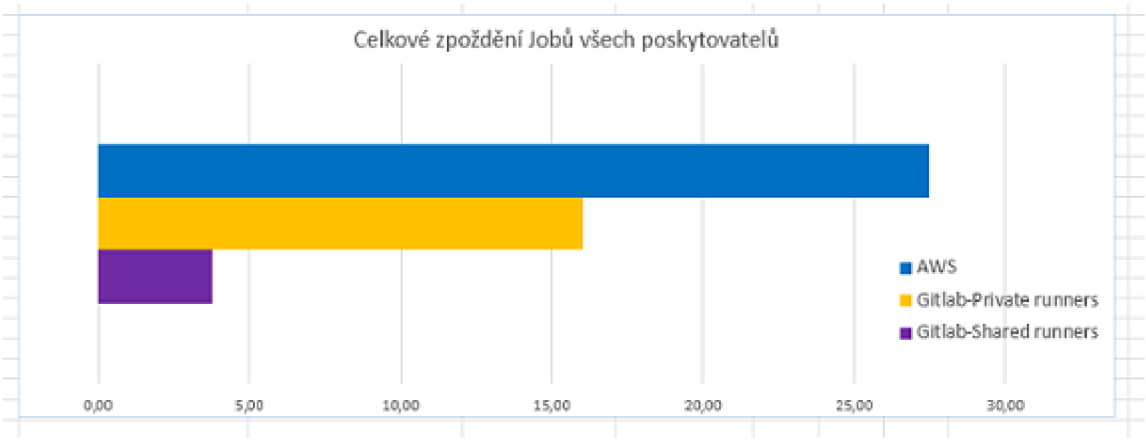
Poskytovatel	Gitlab-Shared runners	Gitlab-Private runner	AWS
Průměrná doba odbavení (sec)	92,01	107,92	127,02
Celková doba odbavení (sec)	920,14	1079,56	1270,26

Tabulka 12.1: Tabulka naměřených doby odbavení jobů všech poskytovatelů

12.1.2 Celkové a průměrné zpoždění Jobů

Druhou hlavní metrikou v rámci porovnávání poskytovatelů CI/CD je zpoždění všech Jobů. Výsledek této metriky je zobrazen v grafu na obrázku 12.2 a dále je tento výsledek společně s průměrnou dobou odbavení vidět i v tabulce 12.2.

Nejlépe se umístily opět Gitlab-Shared runners s celkovou dobou zpoždění 3,75 sekund napříč všemi Joby. Na druhém místě se ztrátou 12,28 sekundy je Gitlab-Private runner a na posledním místě AWS se zpožděním 27,45 sekundy na odbavení Jobů.



Obrázek 12.2: Graf naměřeného celkového zpoždění všech poskytovatelů

Poskytovatel	Gitlab-Shared runners	Gitlab-Private runner	AWS
Průměrná doba zpoždění (sec)	0,37	1,60	2,74
Celková doba zpoždění (sec)	3,75	16,03	27,45

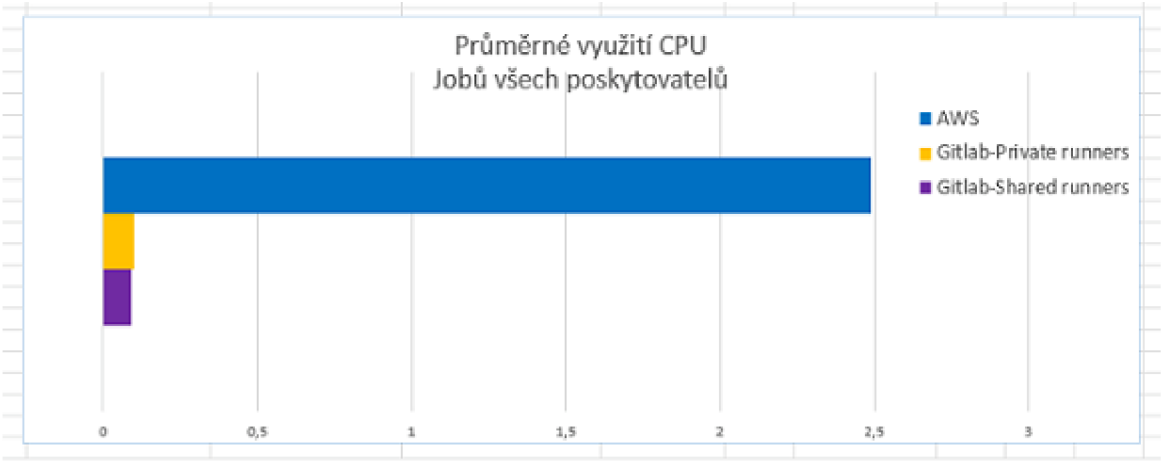
Tabulka 12.2: Tabulka naměřeného zpoždění jobů všech poskytovatelů

12.2 SEKUNDÁRNÍ METRIKY

12.2.1 Průměrné využití CPU

Dále je možné vidět výsledek měření vytížení CPU za jednu minutu v rámci všech deseti Jobů na grafu 12.3 a potom také v tabulce 12.3. Na prvním místě

se opět umístily Gitlab-Shared runners s 0,09% využití CPU. Následuje velmi podobný výsledek u Gitlab-Private runneru, který je pouze o 0,01% horší než předchozí poskytovatel. Na posledním místě opět AWS s hodnotou 2,49% využití CPU.



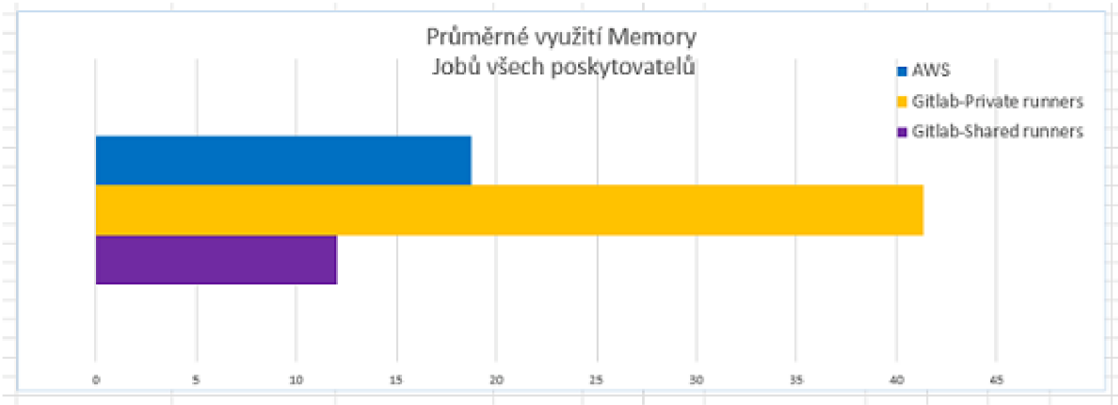
Obrázek 12.3: Graf naměřených hodnot CPU všech poskytovatelů v čase 1 minuty

Poskytovatel	Gitlab-Shared runners	Gitlab-Private runner	AWS
Průměrné využití CPU (%)	0,09	0,1	2,49

Tabulka 12.3: Průměrné hodnoty CPU všech poskytovatelů

12.2.2 Průměrné využití Memory

Poslední měřenou metrikou je využití RAM za 1 minutu. Naměřené hodnoty jsou vidět v grafu na obrázku 12.4 a v tabulce 12.4. Podobně jako u předchozích metrik se i zde umístily nejlépe Gitlab-Shared runners s vytížením 12,07%. Na druhém místě, tentokrát AWS s využitím 18,75% a nejhůře se u této metriky umístil Gitlab-Private runner, který dosahuje více než dvojnásobné hodnoty od AWS 41,34%.



Obrázek 12.4: Graf naměřených hodnot RAM všech poskytovatelů v čase 1 minuty

Poskytovatel	Gitlab-Shared runners	Gitlab-Private runner	AWS
Průměrné využití CPU (%)	12,07	41,34	18,75

Tabulka 12.4: Průměrné hodnoty RAM všech poskytovatelů

13 ANALÝZA AKTUÁLNÍHO ŘEŠENÍ ANONYMNÍ FIRMY

K získání dalších dat pro tento Bakalářský projekt vznikla analýza řešení u anonymní firmy, která využívá služby CI/CD u jednoho z představovaných poskytovatelů tohoto projektu. Při analýze byli důležité následující fakty: typ runnerů, které využívají, s jakými výpočetními prostředky jsou Joby odbavovány a operační systém runnerů. Dále byl, na základě počtu commitů, v rámci firmy, zvolen jeden klíčový projekt, pro který byla získána data ohledně celkového počtu Jobs za jeden pracovní týden a také s jakým zpožděním byli Jobs v této době odbavovány.

13.1 AKTUÁLNÍ ŘEŠENÍ

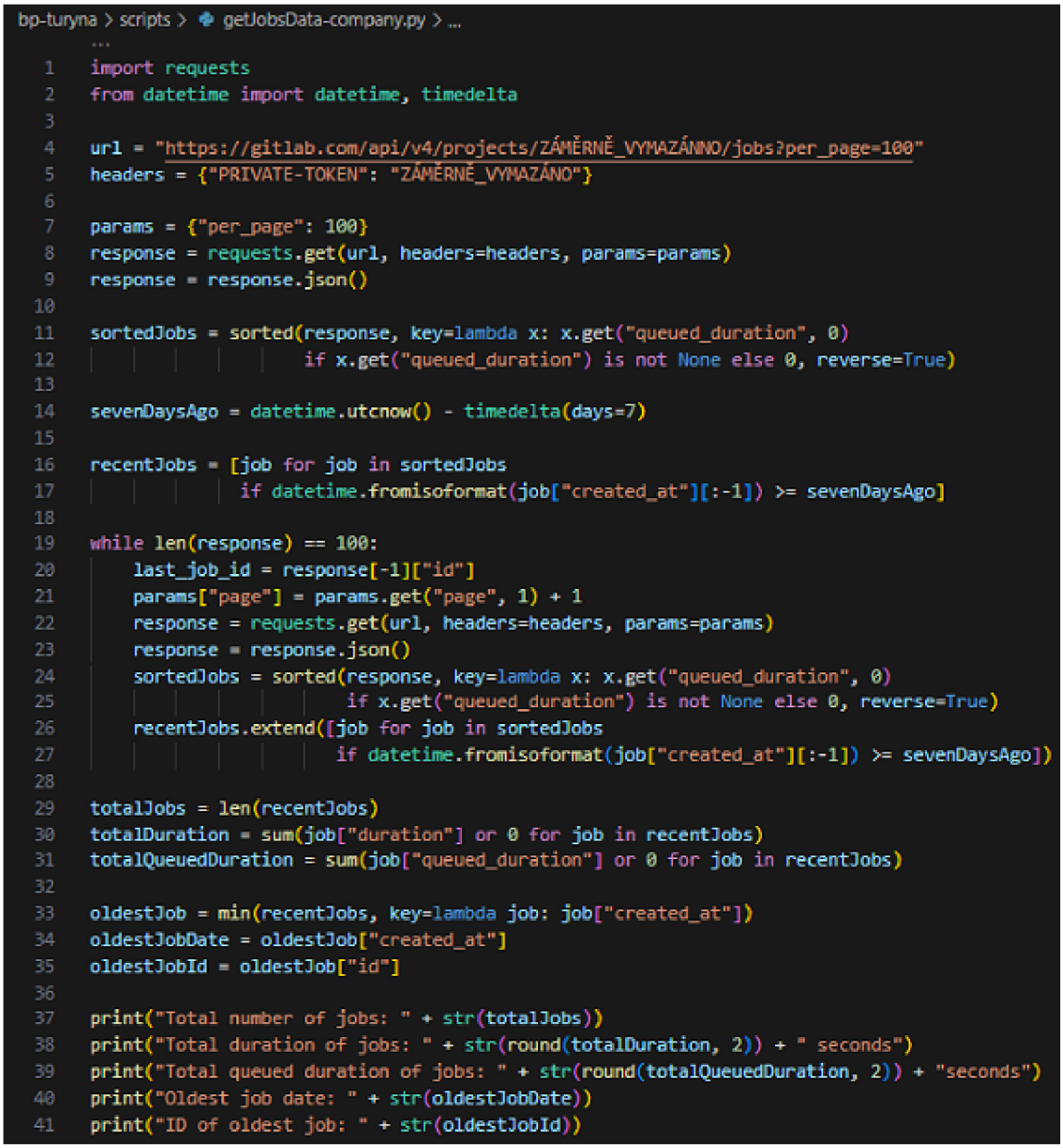
Zvolená firma využívá k vývoji většiny svých projektů poskytovatele Gitlab a CI/CD využívá k automatizovanému testování, managování infrastruktury a Deployi aplikací napříč desítkami projektů. Firma v jejich CI/CD řešení využívá Gitlab-Private runner a vlastní celkem čtyři Gitlab-Private runnery, které všechny běží na OS CentOS Linux s prostředky 8 CPU a 16 GB RAM.

Typ runnerů	OS	vCPU	RAM
4x Gitlab-Private runners	CentOS	8	16

Tabulka 13.1: Parametry aktuálního CI/CD řešení firmy

13.2 ZÍSKÁNÍ DAT Z PROJEKTU

K získání dat runnerů, které firma využívá byl zvolen klíčový projekt, který firma vyvíjí a pravidelně spouští Jobs. Následně došlo k vytvoření scriptu, který pomocí Gitlab Api tento projekt prochází, načítá data za posledních 7 dní ohledně každého odbaveného Jobu, vypisuje celkový počet Jobs, celkovou dobu odbavení a celkové zpoždění.



Obrázek 13.1: Script, který získává data z projektu firmy

Na obrázku 13.2 je zobrazen výstup scriptu, který shromažďuje data z Gitlab projektu. Dále je také z tohoto obrázku (viz. 13.2) patrné, že firma v tomto projektu za 7 dní odbavila celkem 170 Jobů, které celkově trvaly5066,02 sekund (84,43 minut). Z toho Joby byly zpomalené celkem o 2825,15 sekund (47,08 min), což znamená, že Joby u tohoto projektu byly zpožděny o 35,8% pouze za poslední týden.

```
jturyna@DESKTOP-NB5KN69:~/bp-turyna$ python3 scripts/getJobsData-company.py
Total number of jobs: 170
Total duration of jobs: 5066.02 seconds
Total queued duration of jobs: 2825.15 seconds
Oldest job date: 2023-04-17T06:38:39.400Z
ID of oldest job: 1984151
```

Obrázek 13.2: Data projektu firmy za vybraných 7 dní

13.3 VYHODNOCENÍ

Dále je také důležitý fakt, že firma vyvíjí celoročně ve stejné frekvenci a zároveň odbavuje podobné množství Jobů na několika desítkách projektů, lze hrubě odhadnout, že aktuální řešení CI/CD systému není ideální. Samozřejmě by pro takový závěr bylo potřeba nejprve provést tuto analýzu napříč všemi projekty, ale dle mého názoru je na místě rozšíření minimálně ještě o jeden Gitlab-Private runner s ideálně stejnými prostředky (8 vCPU a 16 GB RAM) jako mají již stávající runnery.

14 OBECNÉ ŘEŠENÍ PRO DESIGN GITLAB RUNNERŮ

Návrh CI/CD systému se, alespoň v dnešní době, bude vždy odvíjet na kompromisu mezi finančním kapitálem, dostupností možných poskytovatelů / platforem, jejich kvalit a složitosti Jobů.

U velké většiny firem bude nejdůležitější především finanční rovina tohoto systému v kombinaci s již dostupnými prostředky. Proto je obecné řešení pro design CI/CD systému přímo úměrné s návrhem celého Cloudu, u kterého je možné sledovat celkem tři hlavní trendy. [War21]

14.1 ON-PREMISES

Jedná se o návrh Cloudu, jehož kompletním řešením je vystavění celé infrastruktury například v rámci datacentra nebo v podstatě kdekoliv, kde firma vlastní pozemky. Firma v takovém případě vlastní všechny servery, firewally, loadbalancery a další infrastrukturální prvky nutné k provozu Cloudu. Pokud je Cloud vystaven v datacentru, odpadá pro takové řešení nutnost dodatečných poplatků za připojení k internetu nebo například protipožární opatření.

Zároveň také při On-Premises návrhu Cloudu firma musí řešit aktualizace Operačních systémů a vlastně všech služeb, které mohou jejich aplikace využívat, což může být problém například pokud firma dostatečně neinvestuje do refactoringu. Jelikož se může stát, že některý z využívaných balíčků nebo služeb nemusí být nadále kompatibilní v rámci nově aktualizovaného Operačního systému.

V tomto návrhu Cloudu je běžnou praxí, že firmy mají přesně vymezený infrastrukturální tým, který se stará o správné fungování všech systémů, zodpovídá za aktualizace systémů a také o jeho rozvoj do budoucna.

Vzhledem ke komplexitě řešení a příchodem nových IASS a PASS služeb, je od tohoto provozu Cloudu v posledních letech upouštěno a místo toho jsou většinou využívány druhé dva typy Cloudu.

V tomto případě bych osobně při navrhování rozhodně volil self-hosted Gitlab a několik Privátních Gitlab Runnerů.

14.2 FULL-CLOUD

V podstatě přesný opak k On-Premises řešení provozování Cloudu, které stojí kompletně na IASS a PASS službách provozovaných třetí stranou, jako je AWS. Firma v tomto případě nevlastní žádné ze zařízení, na kterém Cloud běží. Obrovskou výhodou tohoto řešení jsou tzv. on-demand servery, které jsou snadno škálovatelné. Také u této možnosti dochází k značnému ulehčení v Operations části Cloudu (infrastrukturální prvky, patchování, ...), které je čistě v režii poskytovatele.

V případě Full-Cloud bych, při navrhování CI/CD řešení zvažoval AWS a Gitlab-Shared runners. Záleželo by především na komplexitě Jobů a zda firma dokáže využít další výhody, které AWS nabízí. Pokud by firmě záleželo na rychlém odbavování méně složitých Jobů, přikláněl bych se k využití Gitlab Shared runnerů. Naopak, pokud by firma potřebovala odbavovat menší počet složitějších Jobů, dávalo by daleko větší smysl AWS, ve kterém lze snadno službu škálovat.

14.3 HYBRID-CLOUD

Hybridní provoz Cloudu kombinuje to nejlepší z obou předchozích typů a vnímám ho jako určitý mezikrok pro již zaběhlé firmy, které se adaptují na stále se rozrůstávající IASS a PASS služby. Tento typ provozu Cloudu se vyznačuje tím, že některé servery běží na On-premises a některé v Cloudu. Tím pádem firmy mohou využívat servery a zařízení, které již vlastní a zároveň se připravovat na migraci do Full-Cloud prostředí.

15 ZÁVĚR

Závěr tohoto Bakalářského projektu je především nastínění postupu a metrik díky, kterým lze porovnávat neustále se rozrůstávající IASS a PASS služby jako AWS, Gitlab a Github z pohledu CI/CD systémů.

V rámci praktické části došlo k porovnání dvou zvolených poskytovatelů CI/CD AWS a Gitlab. Gitlab byl dále rozdělen do dvou praktických typů, na Gitlab-Private a Gitlab-Shared runner. Na všech třech typech CI/CD systému bylo spuštěno celkem deset stejných Jobů ve dvou skupinách po pěti, které měřili CPU a RAM. Poté došlo pomocí Gitlab Api a AWS Api k získání dat ohledně jednotlivé i celkové doby odbavení všech Jobů a také jejich zpoždění. Takto naměřené metriky byly zobrazovány v grafech a tabulkách u jednotlivých poskytovatelů a následně také vzájemně porovnány v kapitole 12.

Předpokladem před začátkem praktické části tohoto projektu bylo, že všechny metriky budou velmi podobné. Tento předpoklad byl však po provedení měření rychle vyvrácen a bylo dosaženo závěru, ve kterém AWS runner dopadl ze všech třech poskytovatelů výrazně nejhůře, ve třech ze čtyř měřených metrik. Naopak Gitlab-Shared runner se umístil ve všech čtyřech metrikách nejlépe. Proto byl také v teoretické kapitole 14 dosažen závěr ohledně designu CI/CD systémů na základě složitosti Jobů a také komplexity celého CI/CD systému. Poskytovatel Gitlab je vhodný pro firmy provozující Hybrid-Cloud nebo Cloud On-premises kde se předpokládá nižší složitost Jobů s důrazem na jejich rychlé odbavování. Také je zde možnost kombinace obou typů Gitlab runnerů. Poskytovatel AWS je vhodný, pokud firma nevlastní žádná zařízení, na kterých by bylo možné Cloud provozovat a také požaduje škálování, který Gitlab nenabízí. Zároveň by taková firma měla počítat s možným vytížením výpočetních prostředků navíc, pokud se rozhodne provozovat Cloud a CI/CD systém v AWS. Tento negativní fakt je však vyvažován obrovským množstvím spolu-kooperujících systémů, které lze v AWS využít, včetně expertní znalosti AWS Support týmu.

Poslední závěr, který byl v praktické části dosažen na základě měření CI/CD systému anonymní firmy popsaný v kapitole 13. Tato firma využívá několik Gitlab-Private runners k odbavování Jobs napříč desítkami projektů. Pro analýzu a měření byl zvolen jeden z klíčových projektů firmy. Měření spočívalo v získání dat tohoto projektu za uplynulý pracovní týden pomocí Gitlab API a následné spočítání celkové doby odbavení všech Jobů a jejich zpoždění, přes script v programovacím jazyku Python3. Výsledkem bylo odhalení zhoršené efektivity odbavování Jobs o 35,8% pouze na tomto projektu a jen

za náhodně vybraný týden. Pro přesný výsledek by bylo potřeba provést takovéto měření napříč všemi projekty firmy, ale i přesto bylo součástí vyhodnocení této kapitoly navrhnuto řešení, které by mělo efektivitu odbavování Jobů výrazně zvýšit a to pomocí přidání dalšího Gitlab-Private runner.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

BIBLIOGRAFIE

[Kim+16] Gene Kim et al. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution, 2016.

[MHM19] M. R. McQuade, A. Hunter a S. Moore. „Appendix A: How Agile Software Development Came to Enable Adaptable Systems“. In: *Acquisition of Software–Defined Hardware–Based Adaptable Systems*. Center for Strategic a International Studies (CSIS), 2019, s. 31–36. URL: http://www.jstor.org/stable/resrep22602.11.

[Ama21] Amazon Web Services. *AWS CodeBuild Build Specification Reference*. https://docs.aws.amazon.com/codebuild/latest/userguide/build-spec-ref.html. Accessed on May 9, 2023. 2021.

[Cho21] Rahul Choudhary. *Agile or Scrum: Which One is Need to Choose and Why?* C# Corner. 2021. URL: https://www.c-sharpcorner.com/article/agile-or-scrum-which-one-is-need-to-choose-and-why/ (cit. 30. 04. 2023).

[War21] Warwick. *On-Premise, Cloud, & Hybrid Solutions: What's the Best Fit for Your Business?* Dub. 2021. URL: https://www.warwickinc.com/blog/on-premise-cloud-hybrid-solutions/ (cit. 30. 04. 2023).

[Fli22] FlippedCoding. *Difference between Development Stage and Production*. dev.to. Břez. 2022. URL: https://dev.to/flippedcoding/difference-between-development-stage-and-production-d0p (cit. 30. 04. 2023).

[Den23] Densify. *Continuous Integration and Delivery Phases*. Densify Resources. 2023. URL: https://www.densify.com/resources/continuous-integration-delivery-phases-old (cit. 30. 04. 2023).

[Git23a] GitLab. *GitLab Runner Documentation*. GitLab Documentation. 2023. URL: https://docs.gitlab.com/runner/ (cit. 30. 04. 2023).

[Git23b] GitLab. *How shared runners pick jobs*. GitLab documentation. 2023. URL: https://docs.gitlab.com/ee/ci/runners/runners_scope.html#how-shared-runners-pick-jobs (cit. 30. 04. 2023).

[Git23c] GitLab. *Jobs in GitLab CI/CD*. GitLab Documentation. 2023. URL: https://docs.gitlab.com/ee/ci/jobs/ (cit. 30. 04. 2023).

[Git23d] GitLab. *Pipelines in GitLab CI/CD*. GitLab Documentation. 2023. URL: https://docs.gitlab.com/ee/ci/pipelines/ (cit. 30. 04. 2023).

[Git23e] GitLab. *Repository Management in GitLab*. GitLab Documentation. 2023. URL: https://docs.gitlab.com/ee/user/project/repository/ (cit. 30. 04. 2023).

[Red23] Red Hat. *What is CI/CD?* Red Hat DevOps. 2023. URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd (cit. 30. 04. 2023).

[Sem23] Semaphore. *Understanding the Build Stage in Continuous Integration*. Semaphore Blog. 2023. URL: https://semaphoreci.com/blog/build-stage (cit. 30. 04. 2023).

[Ama23a] Amazon Web Services. *AWS CLI Command Reference - batch-get-builds*. Accessed 2023. URL: https://docs.aws.amazon.com/cli/latest/reference/codebuild/batch-get-builds.html.

[Ama23b] Amazon Web Services. *AWS CodeBuild Batch Build Buildspec*. Accessed 2023. URL: https://docs.aws.amazon.com/codebuild/latest/userguide/batch-build-buildspec.html.