Bakalářská práce

České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická

Microservices - bezpečnost, logování a metriky pomocí ELK stacku

Jan Kabíček

Školitel: Ing. Šebek Jiří
Květen 2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kabíček** Jméno: **Jan** Osobní číslo: **499253**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Microservices - bezpečnost, logování a metriky pomocí ELK stacku

Název bakalářské práce anglicky:

Microservices - security, logging and metrics using the ELK stack

Pokyny pro vypracování:

V současné době jsou mikroslužby na vzestupu, čím dál tím více se používají a mají spoustu modifikací a využití.
To sebou samozřejmě přináší problémy se zabezpečením které je potřeba správně a složitě naplánovat a implementovat.
Zároveň je zde i problém s udržováním celé aplikace v chodu a k tomu se právě používá logování a sledování metrik.
Všechny tyto problémy budou cílem tohoto bakalářského projektu. Dílčí úkoly projektu jsou:
1) Provést základní rešerši mikroslužeb a jejich patternů hlavně k zabezpečení, logování a metrikám s využitím služeb ELK stacku
2) Provést analýzu situací, kdy je vhodné použít některé typy zabezpeceni a typy logování
3) Vytvořit knihovnu, která obsahuje vzorovou implementaci zabezpečení a logování mikroslužeb
4) Následně zkusit implementovat navrhnuté řešení do ekosystému mikrosluzeb Pozn. bod 4) bude řešen až v rámci bakalářské práce, která bude navazovat na semestrální projekt

Seznam doporučené literatury:

Yarygina, Tetiana, and Anya Helene Bagge. "Overcoming security challenges in microservice architectures." 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE). IEEE, 2018.
Waseem, Muhammad, et al. "Design, monitoring, and testing of microservices systems: The practitioners' perspective." Journal of Systems and Software 182 (2021): 111061. “Open Source Search: The Creators of Elasticsearch, ELK Stack & Kibana | Elastic.” Www.elastic.co, www.elastic.co.
Alshuqayran, Nuha, Nour Ali, and Roger Evans. "A systematic mapping study in microservice architecture." 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA). IEEE, 2016.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jiří Šebek kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **14.02.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **22.09.2024**

Ing. Jiří Šebek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

© VUT v Praze, Design: VUT v Praze, VIC

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.

Datum převzetí zadání

Podpis studenta

© VUT v Praze, Design: VUT v Praze, VIC

Poděkování

Chtěl bych poděkovat vedoucímu mé práce Ing. Jiřímu Šebkovi za jeho odborný pohled, cenné rady a vedení mé práce. Dále bych chtěl poděkovat rodině a mým přátelům za ohromnou podporu při studiu a psaní práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 26. May 2023

Abstrakt

Tato bakalářská práce se zabývá problémy bezpečnosti, logování a metrik v mikroservisní architektuře s využitím Dockeru. Výsledkem této práce je PoC implementace řešení některých problémů vyvstávajících při vývoji mikroservisní architektury.

Klíčová slova: Java, Mikroservisy, PoC, Docker, Keycloak, ELK, Prometheus, Grafana, Spring Boot, Postgres

Školitel: Ing. Šebek Jiří

Abstract

This bachelor's thesis deals with problems of microservice architecture, especially safety, logging and metrics with the usage of Docker. The result of this work is PoC that implements solutions to some problems that may arise when working with microservice architecture.

Keywords: Java, Microservices, PoC, Docker, Keycloak, ELK, Prometheus, Grafana, Spring Boot, Postgres

Obsah

1 Úvod 1
1.1 Cíle 1

2 Rešerše 3
2.1 Microservisy 3
2.1.1 Návrhové vzory mikroservis 4
2.1.2 Java 4
2.2 Spring Boot 5
2.3 Databáze 5
2.3.1 Relační Databáze 5
2.3.2 NoSQL Databáze 6
2.4 Bezpečnost 7
2.4.1 OWASP 7
2.4.2 Spring Boot Security 7
2.4.3 Oauth2 8
2.4.4 Keycloak 8
2.5 Logování 9
2.5.1 Elastic 9
2.5.2 Logstash 9
2.5.3 Kibana 9
2.6 Metriky 10
2.6.1 Prometheus 10
2.6.2 PromQL 10
2.6.3 Grafana 10
2.7 Docker 11
2.8 REST 11
2.9 Vybrané technologie 12

3 Analýza 13
3.1 Sběr požadavků 13
3.2 Existující řešení 13
3.3 Funkční požadavky 14
3.4 Nefunkční požadavky 14
3.5 Use case model 14

4 Návrh 17
4.1 Vícevrstevnatá architektura 17
4.2 Diagram kontejnerů 18
4.3 Diagram komponent 19

5 Implementace 21
5.1 Keycloak 21
5.1.1 Konfigurace 21
5.2 Prometheus 21
5.2.1 Konfigurace 22
5.3 Grafana 22
5.3.1 Dashboards 22
5.4 ELK 22
5.4.1 Logstash 23
5.4.2 Kibana 23
5.5 Debian 23
5.6 Databáze pro Keycloak 24
5.6.1 OWASP A03:2021 – Injection 24
5.6.2 Circruit breaker 24
5.6.3 Vnitřní žádost / Vnější žádost 25

6 Testování 27
6.1 Junit testy 27
6.1.1 Pokrytí kódu 27
6.2 Uživatelské testování 27
6.2.1 Circruit breaker 27
6.2.2 Vnitřní žádost / Vnější žádost 28

7 Závěr 31

Literatura 33

Obrázky

3.1 use case model 15

4.1 Obrázek vícevrstevnaté architektury. Zdroj [19] 17
4.2 Zobrazení stavby kontejnerů . . . 18
4.3 Component diagram 20

5.1 Grafana dashboard 23
5.2 Smyčka v komunikaci 24

6.1 Smyčka v komunikaci 28
6.2 Typy volání 28

Tabulky

Kapitola 1

Úvod

V posledních letech se vývoj softwarových aplikací rychle vyvíjí a rozšiřuje se novými technologiemi a architekturami. Jedním z těchto trendů je přechod k mikroslužbám (Microservices), což je relativně nový architektonický styl, který rozděluje velké monolitické aplikace na menší a nezávislé služby. Tyto mikroslužby komunikují mezi sebou pomocí lehkých protokolů a často jsou nasazovány odděleně. Tento přístup umožňuje snazší škálování, lepší rozložení zátěže a větší flexibilitu. Využívání Mikroslužeb, ale také přinášejí nové výzvy i v oblasti bezpečnosti, logování a metrik.

1.1 Cíle

Cílem této bakalářské práce je analyzovat, popsat a navrhnout bezpečnostní aspekty mikroslužeb, a to z hlediska konfigurace a implementace. Dalším cílem práce je navrhnout a implementovat logování chyb mikroslužeb pomocí ELK stacku (Elasticsearch, Logstash a Kibana) a metrik pro sledování výkonu mikroslužeb pomoci nástroje Prometheus a Grafana. V rámci práce budou použity konkrétní technologie a nástroje, jako jsou Docker, Elasticsearch, Logstash, Kibana, Prometheus, Grafana, Spring Boot a Keycloak. Výsledkem bude funkční prototyp aplikace s mikroslužbami, běžících v kontejnerech a obsahující ukázku implementace bezpečnosti, logování a metrik pomocí ELK stacku.

Kapitola 2

Rešerše

2.1 Microservisy

Jedná se o vývojářský koncept / pattern, který se snaží o zjednodušení práce. Tento pattern se snaží rozdělit funkcionalitu do jednotlivých malých komponent, které plní jednu funkci. Umožňují lehce pochopit jejich funkci (nemusíme se učit celou strukturu aby jsme mohli upravit malou část). Případné změny se dělají do jedné servisy a tím vzniklé chyby ovlivní jen tu danou mikroservisu. Nadále mají tu výhodu ze jedna část muže byt například psaná v Javě a druhá v Pythonu a komunikují skrze přesné definované API(Application Programming Interface). Velikost mikroservisy bývá něco mezi stovkami až tisícovkami kódu. Vetšinou se nedostaneme na vice kvůli tomu ze mikroservisa by mela dělat pouze jednu věc.[23]

Mikroservisy:

- Výhody: flexibilita a snadná škálovatelnost, lepší oddělení zodpovědností a menší komplexita, vyšší odolnost vůči chybám a výpadkům, rychlejší vývoj nových funkcí a služeb, možnost použití různých technologií a jazyků
- Nevýhody: složitější infrastruktura, větší nároky na testování a nasazení, vyšší nároky na správu a monitoring, možnost problémů s distribuovanou koordinací a integritou dat[11]

Monolitické aplikace:

- Výhody: jednoduché nasazení, menší nároky na infrastrukturu a správu, snadnější ladění a vývoj, snadnější distribuce kódu
- Nevýhody: řešení problémů a aktualizace celé aplikace narazí na limitace velikosti kódu, těžší škálovatelnost, obtížnější oddělení zodpovědností, větší riziko propadu výkonu a stabilita při velké zátěži[11]

Celkově lze říci, že mikroservisy se hodí pro větší a komplexnější aplikace s vysokými nároky na škálovatelnost a flexibilitu, zatímco monolitické aplikace jsou vhodnější pro menší aplikace s jednoduššími požadavky na správu a nasazení. Výběr architektury závisí na konkrétních potřebách a cílech projektu.[23]

2.1.1 Návrhové vzory mikroservis

Návrhové vzory mikroservis jsou vzory umožňující architektům a vývojářům navrhovat, implementovat a nasazovat efektivněji a s menším rizikem chyb. Existuje mnoho návrhových vzorů mikroservis, z nichž každý se zaměřuje na jiný aspekt mikroservis, jako je například spolehlivost, škálovatelnost, flexibilita nebo bezpečnost.

Mezi nejznámější návrhové vzory mikroservis patří:

- API Gateway - tento vzor umožňuje vývojářům sdružovat několik API rozhraní mikroservis do jednoho rozhraní API, což usnadňuje integraci a umožňuje centralizovanou správu.
- Service Registry - tento vzor poskytuje centralizovaný registr mikroservis, který umožňuje dynamickou registraci a odregistrování služeb, což zvyšuje flexibilitu a škálovatelnost systému.
- Circuit Breaker - tento vzor umožňuje mikroservisám správně reagovat na chyby a selhání v jiných službách, aby se minimalizovaly dopady na celý systém. Dále také zamezuje nekonečnému kolování zpráv
- Saga - tento vzor umožňuje transakční řízení mikroservis, kdy jedna akce může mít vliv na více služeb. Saga zajišťuje, aby transakce byly atomické a provedly se úspěšně nebo aby se vrátily do původního stavu v případě selhání.

2.1.2 Java

Java je vysokoúrovňový objektově orientovaný programovací jazyk, který byl vyvinut společností Sun Microsystems (nyní součástí společnosti Oracle) v roce 1995. Java je platformově nezávislý jazyk, což znamená, že kód napsaný v Javě může být spuštěn na různých operačních systémech bez nutnosti jakýchkoli úprav. Jedním z hlavních důvodů, proč je Java tak populární, je fakt, že obsahuje rozsáhlou knihovnu standardních funkcí a tříd, které usnadňují práci s různými aspekty programování, včetně práce s vstupem a výstupem, soubory, sítěmi, vlákny a mnoho dalšího.[15]

Java také obsahuje mnoho bezpečnostních funkcí, které pomáhají chránit aplikace před útoky, jako jsou například SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF) a mnoho dalších. Dále Java podporuje digitální podpisy a šifrování, což umožňuje bezpečnou komunikaci a přenos dat mezi aplikacemi.

Dalším důležitým aspektem Javy je její použití pro vývoj mikroservis. Ty jsou moderní architektonický vzor, který umožňuje vývojovým týmům vytvářet a nasazovat aplikace jako soubor nezávislých služeb, které mohou být snadno škálovány a aktualizovány nezávisle na sobě. Java poskytuje mnoho nástrojů a frameworků pro vývoj mikroservis, jako například Spring Boot, Quarkus a Micronaut.

Vzhledem k popularitě Javy a jejímu využití pro vývoj různých typů aplikací a služeb, je bezpečnost v Javě stále důležitější. Vývojáři musí být

obeznámeni s různými bezpečnostními hrozbami a musí být schopni použít správné nástroje a postupy pro zajištění bezpečnosti svých aplikací.[15]

2.2 Spring Boot

Spring Boot je open-source framework pro vývoj a nasazení webových aplikací v jazyce Java. Jedním z hlavních cílů Spring Bootu je zjednodušit vývoj webových aplikací tím, že eliminuje nutnost konfigurace a poskytuje předem připravené nástroje pro rychlé nasazení. Dále nabízí jednoduchou rozšiřitelnost a integraci s dalšími technologiemi včetně Spring Security pro zabezpečení aplikace, Spring Data pro práci s databází a mnoho dalšího.

Spring Boot je navržen tak, aby byl co nejjednodušší pro vývojáře, kteří se již seznámili s frameworkem Spring. Využívá řadu konvencí pro snadné nastavení a nasazení aplikace. Navíc také nabízí vestavěné webové servery jako Tomcat nebo Jetty, což znamená, že není třeba konfigurovat webový server zvlášť.

Další výhodou Spring Bootu je jeho rozšiřitelnost a integrace s dalšími technologiemi. Například framework Spring Security je poskytován jako součást Spring Bootu umožňující vývojářům snadno integrovat bezpečnostní mechanismy do svých aplikací. Stejně tak je možné využít Spring Data pro práci s různými databázovými technologiemi, včetně relačních a NoSQL databází.

V neposlední řadě také podporuje řadu nástrojů pro snadnou správu aplikací, včetně Spring Boot Actuator, který poskytuje různé informace o běžící aplikaci a umožňuje provádět různé akce, jako je například restart aplikace.

Celkově lze říci, že Spring Boot je silným nástrojem pro vývoj webových aplikací v jazyce Java. Jeho výhody spočívají v jednoduchosti konfigurace a nasazení, rozšiřitelnosti a integraci s dalšími technologiemi a nástroji pro správu aplikací.[16]

2.3 Databáze

2.3.1 Relační Databáze

Relační databáze jsou základem moderního informačního systému a jsou široce používány v průmyslových aplikacích, obchodních transakcích, vědeckých experimentech a dalších oblastech. Tyto databáze jsou založeny na matematické teorii relací a nabízejí mnoho výhod, včetně konzistence dat, snadného přístupu k datům, integrovaného zabezpečení a dalších funkcí.

Jednou z nejvýznamnějších vlastností relačních databází je schopnost ukládat data v různých tabulkách, které jsou vzájemně propojeny pomocí klíčů. Tyto klíče umožňují databázovým systémům propojovat data z různých tabulek a vytvářet složité relace mezi daty. To umožňuje uživatelům snadno získávat potřebné informace a analyzovat data s velkou přesností.

Další významnou vlastností relačních databází je používání strukturovaného jazyka dotazování (SQL), který umožňuje uživatelům vytvářet a manipulovat s daty pomocí jednoduchých a intuitivních příkazů. To umožňuje uživatelům rychle vyhledávat, aktualizovat a vkládat data do databáze bez nutnosti znát detaily o fyzické organizaci dat v databázi.

Přestože jsou relační databáze velmi užitečné, v posledních letech se objevila řada nových technologií a řešení pro správu a analýzu dat, jako jsou NoSQL databáze a big data technologie. Tyto nové technologie nabízejí nové možnosti pro ukládání a zpracování velkých objemů dat, které jsou charakteristické pro moderní informační systémy. Přesto však relační databáze zůstávají jedním z nejvýznamnějších a nejrozšířenějších typů databázových systémů a zůstávají klíčovou součástí mnoha kritických podnikových aplikací a systémů.[10] [7]

2.3.2 NoSQL Databáze

Samotná definice NoSQL databází není jednoznačná, protože tato kategorie databází zahrnuje mnoho různých typů databázových systémů. Nicméně, obecně lze říci, že NoSQL databáze se vyznačují tím, že se liší od tradičních relačních databází v jedné nebo více oblastech, jako jsou datové modely, škálování a výkon.

Jedním z hlavních důvodů pro vývoj NoSQL databází byla potřeba řešit problémy s vysokou škálovatelností a výkonem, které jsou často spojeny s relačními databázemi. NoSQL databáze využívají různé datové modely, jako jsou například dokumentové, klíč-hodnota, grafové a další. Každý z těchto modelů má své vlastní výhody a omezení a je vhodný pro různé typy aplikací.

Jedním z největších přínosů NoSQL databází je schopnost rychle a snadno zpracovávat velké objemy dat. To je způsobeno tím, že NoSQL databáze jsou navrženy tak, aby byly škálovatelné horizontálně, což znamená, že lze snadno přidávat další servery k zajištění většího výkonu a kapacity. Navíc některé typy NoSQL databází, jako jsou například grafové databáze, jsou zvláště užitečné pro analýzu a využívání datových vztahů.

Vývoj NoSQL databází také přináší řadu výzev a omezení, které je třeba zvážit při výběru vhodného řešení pro konkrétní aplikaci. Mezi tyto výzvy patří například omezená podpora pro transakce a dotazování, nutnost správy a synchronizace distribuovaných databázových systémů a další.

Celkově lze tedy říci, že NoSQL databáze poskytují alternativu k tradičním relačním databázím a přinášejí řadu výhod pro aplikace s vysokým objemem dat a vysokými nároky na škálovatelnost a výkon. Nicméně, pro správnou volbu NoSQL databázového řešení je třeba pečlivě zvážit potřeby konkrétní aplikace a zvážit výhody a nevýhody jednotlivých datových modelů a NoSQL databázových systémů.[12] [17]

2.4 Bezpečnost

2.4.1 OWASP

OWASP (Open Web Application Security Project) je mezinárodní nezisková organizace, která se zaměřuje na zvyšování bezpečnosti webových aplikací. Tato oragnizace je známá pro své publikace a materiály týkající se zabezpečení webových aplikací, včetně seznamu deseti nejvíce kritických bezpečnostních chyb v aplikacích (OWASP Top 10), které jsou aktualizovány každých několik let. Tyto chyby zahrnují například nebezpečný přístup k autentizaci, nedostatečné řešení zabezpečení vstupů a výstupů, špatnou konfiguraci serveru a podobně.

Dále také poskytuje mnoho nástrojů a doporučení pro zabezpečení webových aplikací. Mezi ně patří například OWASP ZAP (Zed Attack Proxy), bezplatný nástroj pro testování zabezpečení webových aplikací, nebo OWASP Testing Guide, který poskytuje pokyny pro testování bezpečnosti webových aplikací. OWASP rovněž pořádá mnoho konferencí a školení, aby pomohla vzdělávat vývojáře a bezpečnostní experty.

Organizace OWASP se stala důležitým hráčem v oblasti bezpečnosti webových aplikací a jeho vliv se rozšířil do mnoha zemí. Jeho zdroje jsou zdarma a otevřené pro všechny, což umožňuje širokému okruhu lidí přístup k bezpečnostním informacím a materiálům.[2]

2.4.2 Spring Boot Security

Spring Boot Security je součástí frameworku Spring Boot, který se zaměřuje na zabezpečení webových aplikací v Javě. Tento framework umožňuje snadné a efektivní řešení zabezpečení aplikace a ochranu proti různým útokům. Spring Boot Security poskytuje mnoho funkcí, jako jsou autentizace, autorizace, filtry, session management a mnoho dalších.[4]

Jednou z klíčových funkcí Spring Boot Security je autentizace. Tento framework podporuje různé způsoby autentizace, jako jsou přihlášení uživatele s použitím uživatelského jména a hesla, autentizace pomocí tokenů nebo OAuth. Autorizace je také snadno řešitelná v Spring Boot Security. Framework umožňuje vytvářet role pro uživatele a definovat přístupová práva pro jednotlivé stránky a funkce aplikace. Filtry jsou další užitečnou funkcí Spring Boot Security, která umožňuje řešit různé útoky, jako jsou CSRF (Cross-Site Request Forgery) nebo XSS (Cross-Site Scripting).

Spring Boot Security také poskytuje nástroje pro správu session management, které umožňují efektivní správu uživatelských relací. Tento framework také umožňuje snadné a efektivní řešení zabezpečení webových služeb, které se stávají stále důležitější v moderních aplikacích.

Kromě toho je Spring Boot Security velmi populární framework pro zabezpečení webových aplikací v Javě. Jeho snadné použití a mnoho funkcí, které nabízí, jsou důvodem jeho popularita. Spring Boot Security také poskytuje mnoho materiálů a dokumentace. To nadále usnadňuje uživatelům

používání a pochopení tohoto frameworku.[25]

2.4.3 Oauth2

OAuth2 je autorizační protokol, který umožňuje uživatelům poskytnout přístup k jejich chráněným zdrojům třetím stranám, aniž by museli sdílet své přihlašovací údaje. Tento protokol je založen na autorizaci prostřednictvím tokenu, který je vydáván po úspěšné autentizaci uživatele. OAuth2 se skládá z několika hlavních komponent, včetně autentizačního serveru, zdrojového serveru, klienta a uživatele.[5]

Autentizační server je komponenta vydávající autorizační tokeny na základě ověření identity uživatele. Zdrojový server poté ověřuje platnost tokenů a umožňuje přístup k chráněným zdrojům na základě oprávnění uložených v tokenu. Klient je aplikace, která chce získat přístup k chráněným zdrojům a požaduje autorizační token od autentizačního serveru. Uživatel je pak koncový uživatel, který poskytuje své přihlašovací údaje pro autentizaci.

Z důvodu toho že OAuth2 je rozsáhlý protokol s mnoha konfiguračními možnostmi a variantami, tak existuje několik implementací OAuth2, včetně Google OAuth2, Facebook OAuth2 a Twitter OAuth2. Tyto implementace se mohou lišit ve způsobu autentizace, povolených oprávněních a dalších funkcích.[5]

Využití OAuth2 může být výhodné pro aplikace, které potřebují přístup k chráněným zdrojům a nechtějí sdílet přihlašovací údaje uživatele s třetí stranou. OAuth2 umožňuje uživatelům udržet kontrolu nad svými přihlašovacími údaji a zároveň umožňuje přístup k chráněným zdrojům aplikacím, které si to vyžadují.

V průběhu let se stala OAuth2 stále populárnější a často se používá v kombinaci s jinými technologiemi, jako jsou RESTful API nebo single sign-on (SSO) řešení. OAuth2 také slouží jako základ pro novější autorizační protokoly, jako je například OpenID Connect.[5]

2.4.4 Keycloak

Keycloak je open source identity and access management systém postavený na technologii Java. Jeho cílem je poskytnout centralizované řešení pro autentizaci, autorizaci a správu přístupu pro aplikace a služby v různých prostředích. Keycloak umožňuje spravovat uživatele, role, skupiny, klienty a mnoho dalších prvků, které jsou potřebné pro správu přístupu. Podporuje také standardy jako OpenID Connect, OAuth 2.0, SAML a mnoho dalších.[22]

Výhodou použití Keycloak je možnost snadné integrace s existujícími aplikacemi a službami. Díky podpoře standardů může být Keycloak použit s různými programovacími jazyky a frameworky. Další výhodou je podpora mnoha bezpečnostních funkcí, jako jsou více faktorová autentizace, detekce podvodů, správa hesel a další.

Keycloak je navržen tak, aby byl snadno škálovatelný a odolný vůči výpadkům. Využívá k tomu clusterování a replikaci datových úložišť, takže

v případě výpadku jednoho uzlu, může být provoz přenesen na jiný uzel bez ztráty dat nebo přerušení služby.

Využití Keycloaku může být přínosné zejména pro organizace, které potřebují řešit bezpečnost a správu přístupu pro své aplikace a služby. Díky otevřenému zdrojovému kódu a aktivní komunitě je Keycloak stále vyvíjen a zdokonalován, což zajišťuje jeho stabilitu a bezpečnost i v budoucnu.[22]

2.5 Logování

2.5.1 Elastic

Elastic je open source full-text search a analýza datové platformy postavená na Apache Lucene. Jedná se o velmi výkonné a škálovatelné řešení pro vyhledávání, vizualizaci a analýzu velkých objemů dat. Nabízí mnoho možností pro vyhledávání, filtrování a agregaci dat, a to včetně plnohodnotného vyhledávání v jazyce plném textu a využití komplexních dotazů. Elastic se často používá pro monitorování a analýzu logů, sledování výkonu a monitorování infrastruktury, sledování uživatelského chování a mnoho dalších aplikací. Nicméně také umožňuje provádět analýzy dat pomocí jazyků jako je SQL nebo Python a poskytuje mnoho možností pro vizualizaci dat v reálném čase.

Výhodou Elasticu je také jeho snadná instalace a konfigurace, která umožňuje rychlý vývoj a nasazení aplikací.Dále také poskytuje komplexní zabezpečení dat, včetně ověřování uživatelů, šifrování dat a správy přístupových práv. Díky své otevřenosti a širokému spektru nástrojů a rozšíření se Elastic stal velmi populárním řešením pro vyhledávání a analýzu dat.[18]

2.5.2 Logstash

Logstash je open-source nástroj pro sběr, zpracování a ukládání logů. Sběr logů může být realizován pomocí různých vstupů, například z TCP/UDP, souborů nebo jiných systémových zdrojů. Navíc umožňuje filtrovat a transformovat data pomocí různých pluginů, které mohou být konfigurovány v souborech s konfigurací. Výstupní pluginy pak mohou data posílat do různých úložišť, jako je například Elasticsearch nebo Kafka. V neposlední řadě také nabízí možnosti pro zpracování strukturovaných dat, například pomocí CSV nebo JSON.

Jako součást Elastic Stacku se Logstash často používá pro sběr a zpracování logů v prostředí s více aplikacemi a službami. V kombinaci s Elasticsearch a Kibana umožňuje snadno prohledávat a analyzovat data z logů.[8]

2.5.3 Kibana

Kibana je open source vizualizační nástroj pro Elasticsearch. Poskytuje uživatelské rozhraní pro správu, vizualizaci a analýzu uložených dat. Mezi hlavní funkce Kibany patří vytváření grafů, tabulek, mřížek, map, časových řad a dalších vizualizací dat. Jako další funkce umožňuje prohlížení a vyhledávání

dat v reálném čase, zobrazování živých aktualizací dat a vytváření upozornění na kritické události. V posledních letech se Kibana stala stále populárnější a tvoří spolu s Elasticsearch a Logstash tzv. "ELK stack".[9]

2.6 Metriky

2.6.1 Prometheus

Prometheus je open-source monitoringový systém, který slouží ke sběru a analýze metrik získaných z různých aplikací a systémů. Jeho základní myšlenkou je sběr časově závislých dat, což umožňuje uživatelům sledovat chování systémů a aplikací v reálném čase. Tento systém je navržen tak, aby byl snadno rozšiřitelný a aby se dokázal vyrovnat s velkým objemem dat.

Navíc Prometheus používá vlastní dotazovací jazyk nazvaný PromQL, který umožňuje uživatelům dotazovat data získaná ze sběračů metrik a vytvářet komplexní dotazy a grafy. Díky jeho otevřenosti a kompatibilitě s mnoha dalšími nástroji a technologiemi, jako jsou Kubernetes, Grafana a Alertmanager, se stal populárním nástrojem pro monitorování moderních mikroservis.

Výhody Prometheusu zahrnují jednoduchou instalaci, konfiguraci a použití, schopnost sbírat a ukládat data v reálném čase, možnost vytvářet grafy a vizualizace pomocí nástroje Grafana a také schopnost generovat upozornění na základě definovaných pravidel pomocí Alertmanageru.[24]

2.6.2 PromQL

PromQL (Prometheus Query Language) je dotazovací jazyk používaný v prostředí Prometheus pro vyhledávání a analýzu metrik. Jeho syntaxe je inspirována SQL a umožňuje uživatelům provádět sofistikované dotazy na časové řady metrik. Dále podporuje řadu operací, jako jsou aritmetické operace, agregace, filtry, podmínky a další. Uživatelé mohou například použít PromQL k výpočtu průměrné hodnoty metriky během určitého časového období, identifikovat výjimky a anomálie, nebo vytvořit složitější dotazy pro získání důležitých dat pro řízení služeb.

Navíc PromQL umožňuje spojování více dotazů a získávání údajů z více zdrojů pomocí funkce "join". V neposlední řadě tento jazyk také poskytuje nástroje pro vyhledávání metrik, zobrazení vzorců, zobrazení grafů a další.

Díky své vysoké flexibilitě a efektivitě se PromQL stal oblíbeným nástrojem pro správu a monitorování prostředí, zejména pro kontejnerové a mikroservisy. Jeho snadné použití a srozumitelná syntaxe z něj dělají mocný nástroj pro vývojáře a správce systémů.[21]

2.6.3 Grafana

Grafana je open-source platforma pro vizualizaci a analýzu dat v reálném čase. Poskytuje uživatelům možnost snadno vytvářet interaktivní a informativní grafy, panely a dashboardy z různých zdrojů dat.Zároveň Grafana podporuje

širokou škálu datových zdrojů, včetně časových řadových databází, jako je například InfluxDB nebo Prometheus, relačních databází, jako je MySQL a mnoha dalších.

Hlavní výhodou Grafany je její intuitivní uživatelské rozhraní umožňující uživatelům snadno vytvářet a upravovat vizualizace bez nutnosti psaní kódu. Grafana také poskytuje bohaté možnosti nastavení, které umožňují přizpůsobit vzhled a chování grafů a panelů. Díky tomu je možné vytvářet atraktivní a přehledné prezentace dat, které pomáhají uživatelům lépe porozumět jejich významu a vztahům.[6]

2.7 Docker

Kontejnerizace aplikací pomocí Dockeru se stala běžnou praxí vývojářů mikroservis. Jednou z hlavních výhod kontejnerizace je oddělení aplikace od hostitelského systému a jeho konfigurace, což umožňuje snadné a rychlé nasazení a škálování aplikací. Díky tomu mohou být aplikace spouštěny na různých operačních systémech a v různých prostředích, což zjednodušuje vývoj a testování.

Docker poskytuje možnost definovat prostředí pro běh aplikace pomocí Dockerfile, což umožňuje snadné vytvoření obrazu aplikace, který může být nasazen a spuštěn na libovolném stroji s Dockerem. Tento proces je zcela automatizovaný, což zvyšuje efektivitu vývojového procesu a minimalizuje riziko chyb při nasazení aplikace.

Výhody použití Dockeru pro kontejnerizaci mikroservis zahrnují rychlé nasazení, snadnou správu, izolaci aplikací a možnost škálování v závislosti na potřebách aplikace. Nevýhody zahrnují nutnost správy kontejnerů. To může být pro nezkušené vývojáře zdrojem chyb a potíží než si zvyknou na nové prostředí. Dále také závislost na Dockeru, což může být problém při vývoji aplikací, které musí být spouštěny v jiných prostředích než na Dockeru.[3]

2.8 REST

REST(Representational State Transfer) je architektonický styl poskytující framework pro návrh a implementaci webových služeb. Jeho cílem je dosáhnout jednoduchosti, škálovatelnosti mezi systémy. Je navržený pro komunikaci mezi klientem a serverem v distribuovaném prostředí, jako je internet. REST se soustředí na přenos stavu a reprezentaci dat mezi systémy prostřednictvím standardního protokolu HTTP. Dále REST obsahuje tyto základní principy: Client-Server model, Stateless komunikace, Uniformní rozhraní, Cacheování, Vrstvený systém, Kód na vyžádání. [20]

2.9 Vybrané technologie

Při výběru technologií pro logování, bezpečnost a metriky v mikroservisní architektuře v Dockeru je důležité provést pečlivou analýzu a vyhodnotit různé možnosti. Existuje mnoho různých technologií, nástrojů a postupů, které lze použít, a každá z nich má své výhody a nevýhody. Proto je nutné zvážit konkrétní požadavky a cíle systému. Na základě toho vybrat technologie, které budou nejvhodnější pro daný systém.Nicméně i zkušenosti programátora a komunita okolo technologie hrají ve výběru svou roli. Proto pro tuto práci ve výběru hráli tyto faktory.Pro implementaci logování, bezpečnosti a metrik v mikroservisách je vybráno několik technologií, které se ukázaly jako vhodné pro daný systém. Mezi tyto vybrané technologie patří:

Java: Jazyk Java je vybrán pro implementaci z důvodu svého robustního a široce používaného frameworku pro vývoj - Spring Boot. Java poskytuje výhody jako objektově orientovaný přístup, vysoká přenositelnost a bohatá komunita. Dále také pro to že autor už má největší zkušenosti s tímto jazykem.

Spring Boot: Spring Boot je framework pro vývoj mikroservis v Javě, který nabízí mnoho užitečných funkcionalit. Poskytuje základní infrastrukturu pro vývoj, včetně správy závislostí, integrace s databází a snadného nasazování. Vývoj ve Spring Bootu je výrazně rychlejší a přehlednější než v čisté Javě.

PostgreSQL: Jako databázový systém je vybrán PostgreSQL pro jeho robustnost, spolehlivost a podporu pokročilých funkcí. PostgreSQL je známý pro svoji bezpečnost a možnosti škálování, což ho činí vhodnou volbou pro systémy s mikroservisamy.

Prometheus a Grafana: Pro sběr a vizualizaci metrik jsou vybrány nástroje Prometheus a Grafana. Prometheus je open-source systém pro sběr a monitorování metrik, zatímco Grafana poskytuje nástroje pro vizualizaci těchto metrik ve formě interaktivních grafů a dashboardů.

ELK stack: ELK stack (Elasticsearch, Logstash, Kibana) je vybrán pro efektivní řešení logování. Elasticsearch slouží jako fulltextový vyhledávací a analytický engine pro rychlé vyhledávání logů, Logstash slouží k příjmu, transformaci a filtrování logových záznamů a Kibana poskytuje uživatelské rozhraní pro vizualizaci logů a jejich analýzu.

Keycloak: Pro zajištění bezpečnosti a správy identit je vybrán Keycloak. Jedná se o open-source identitní a přístupový manažer, který umožňuje autentizaci, autorizaci a správu uživatelských rolí a oprávnění v mikroservisách.

Docker: Docker byl vybrán v rámci práce kvůli možnosti kontejnerizace a možnosti rychlého a poloautomatizovaného nasazení pomocí docker-compose.

REST: Architektonický styl REST byl vybrán kvůli jeho jednoduchosti a široké podpoře.

Kapitola 3

Analýza

3.1 Sběr požadavků

Většinou se podle něj formuje i výsledný projekt a použité technologie tak aby co nejvíce vyhovovali požadavkům a následný vývoj nebyl příliš složitý. Většinou se získávají od uživatelů a zainteresovaných stran. Což mohou být například administrátoři, vývojáři, manažeři ale i koncoví uživatelé. Dále se také používá analýza existujícího systému kde se dají sbírat požadavky na stejnou funkcionalitu nebo na vylepšení funkcionalit. Důležitým krokem při sběru požadavků je stanovení cílů projektu a jeho zaměření. Například, zda se zaměříte spíše na zajištění bezpečnosti systému, nebo na optimalizaci výkonu a zlepšení uživatelského zážitku.

3.2 Existující řešení

Logování, bezpečnost a metriky jsou zásadními prvky každého mikroservisního systému, které je nutné implementovat na každém systému zvlášť. Každý mikroservisní systém může mít odlišné požadavky na logování, bezpečnost a metriky v závislosti na jeho architektuře, rozložení, použitých technologiích a dalších faktorech. Proto neexistuje žádné jednotné řešení, které by pokrylo všechna možná rozložení mikroservis, jejich požadavky na logování, bezpečnost a metriky.

Každý systém musí být implementován individuálně a zohledňovat specifika daného systému. Například pro logování mohou být použity různé nástroje a technologie, v závislosti na tom, jaký typ logů potřebujeme sbírat, jak rychle je potřeba je zpracovat a jakou úroveň detailů chceme zaznamenat. Bezpečnostní opatření mohou také být zcela odlišné v závislosti na konkrétních potřebách systému a na tom, jaké hrozby jsou pro něj relevantní. Metriky jsou důležité pro měření výkonu a zdraví mikroservis, ale opět se mohou lišit v závislosti na specifických požadavcích a potřebách daného systému.

Protože neexistuje jedno univerzální řešení pro všechny možné rozložení mikroservis, je třeba vždy přistupovat k implementaci logování, bezpečnosti a metrik individuálně a navrhovat je pro každý systém zvlášť. Pokud by bylo použito univerzální řešení, nemuselo by být dostatečně přizpůsobeno

potřebám konkrétního systému a mohlo by způsobit zbytečné náklady nebo dokonce ohrozit bezpečnost a stabilitu systému.

3.3 Funkční požadavky

- FR01 - Systém by měl být schopen sbírat logy z jednotlivých mikroservis.
- FR02 - Systém by měl být schopen monitorovat a zaznamenávat metriky výkonu mikroservis.
- FR03 - Systém by měl být schopen identifikovat a reagovat na bezpečnostní incidenty v mikroservisách.
- FR04 - Systém by měl umožňovat zobrazování a filtrování logů a metrik na různých úrovních (jako jsou např. úroveň aplikace, úroveň služby, úroveň kontejneru).
- FR05 - Systém by měl být schopen vnější Oauth2 bezpečnosti.

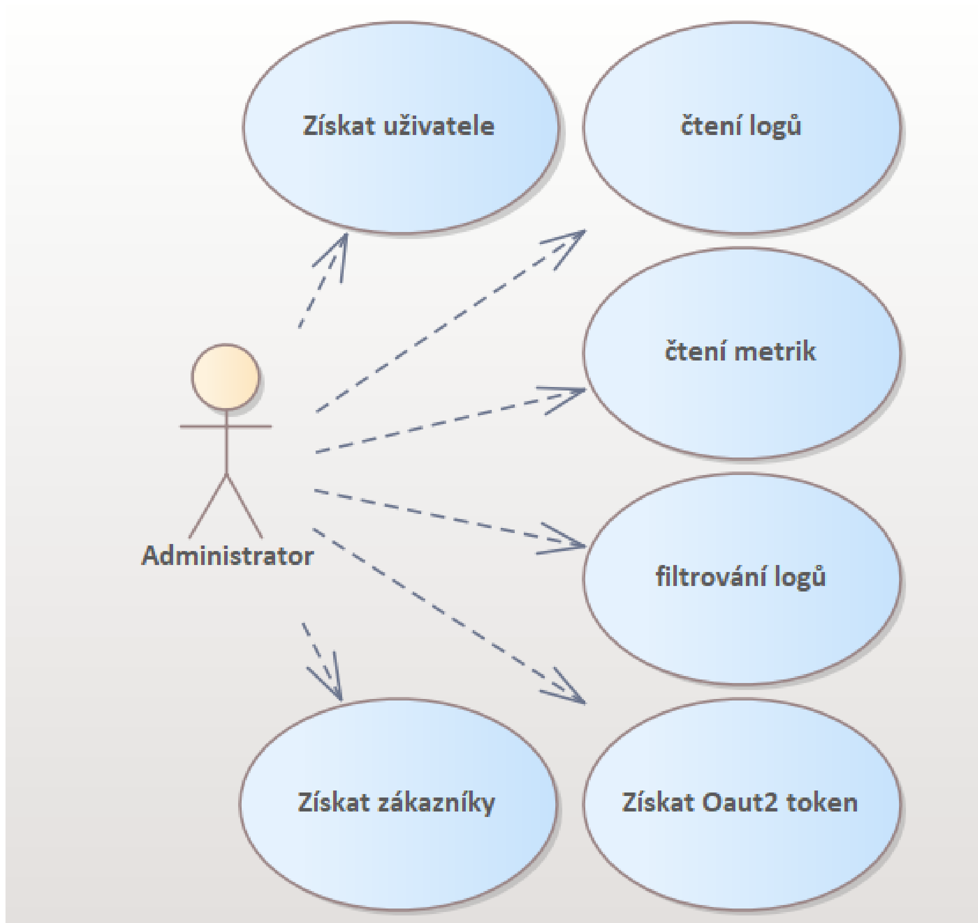
3.4 Nefunkční požadavky

- NFR01 - Systém by měl být rychlý a efektivní při sbírání logů a metrik z mikroservis.
- NFR02 - Systém by měl být odolný vůči výpadkům a chybám a umět se s nimi vypořádat.
- NFR03 - Systém by měl být škálovatelný a umožňovat správu velkého množství mikroservis.
- NFR04 - Systém by měl být bezpečný a chránit získané data proti neoprávněnému přístupu a úniku.
- NFR05 - Systém by měl být jednoduchý na nasazení a správu a měl by být dobře dokumentován pro usnadnění práce s ním pro další uživatele.
- NFR06 - Systém by měl mít mikroservisní architekturu běžící v dockeru.

3.5 Use case model

Na obrázku je možné vidět use case model obsahující jednoho uživatele, který je v našem případě administrátor systému. Má pouze několik základních use casu vycházejících z funkčních požadavků.

- získat uživatele - administrátor má možnost získat seznam uživatelů.
- čtení logů - administrátor má možnost čtení všech logů které všechny aplikace vygenerovaly



Obrázek 3.1: use case model

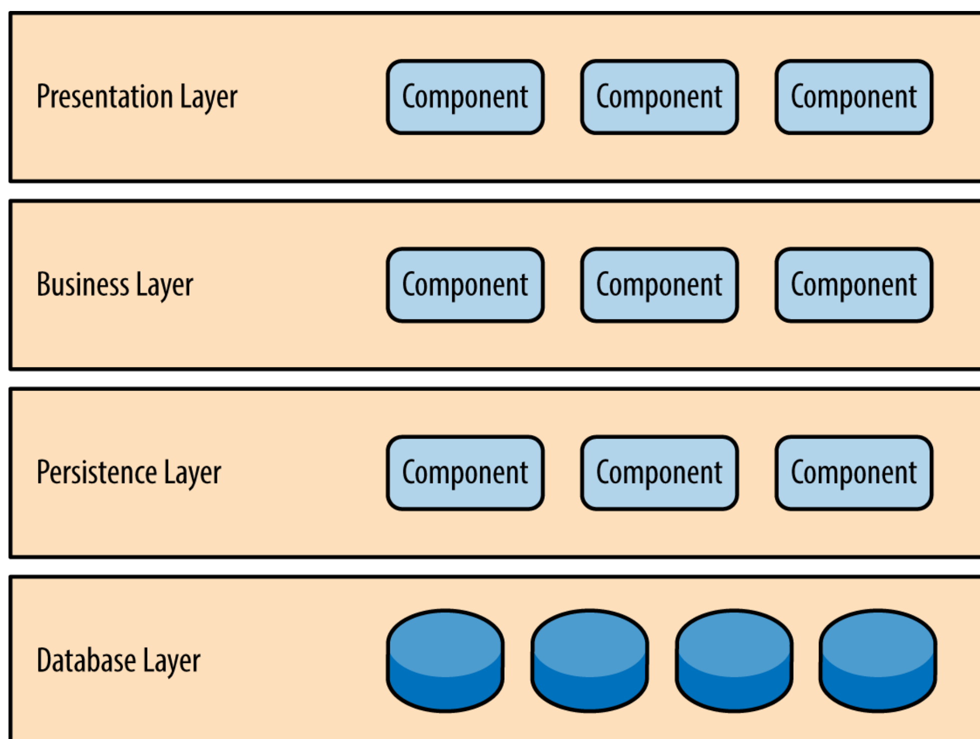
- čtení metrik - administrátor má možnost čtení metrik důležitých mikroservis
- filtrování logů - administrátor má možnost vyfiltrovat si logy
- získat OAuth2 token - administrátor má možnost získat si OAuth2 token se kterým může vykonávat potřebné požadavky
- Získat zákazníky - administrátor má možnost vytažení si všech zákazníků

Kapitola 4

Návrh

Tato práce využívá principy mikroservisní architektury. To znamená rozdělení aplikaci do několika samostatných komponent, zvaných mikroservisy. Jedná se o to, že každá mikroservisa vykonává jednu konkrétní část projektu a spolupracuje s ostatními mikroservisamy, které neběží na jednom stroji, ale můžou být horizontálně škálované skrze několik strojů. Tato komunikace může být zařízena několika způsoby, ale zde je použita klasická metoda komunikace skrz REST rozhraní, která je nejrozšířenější.

4.1 Vícevrstevnatá architektura

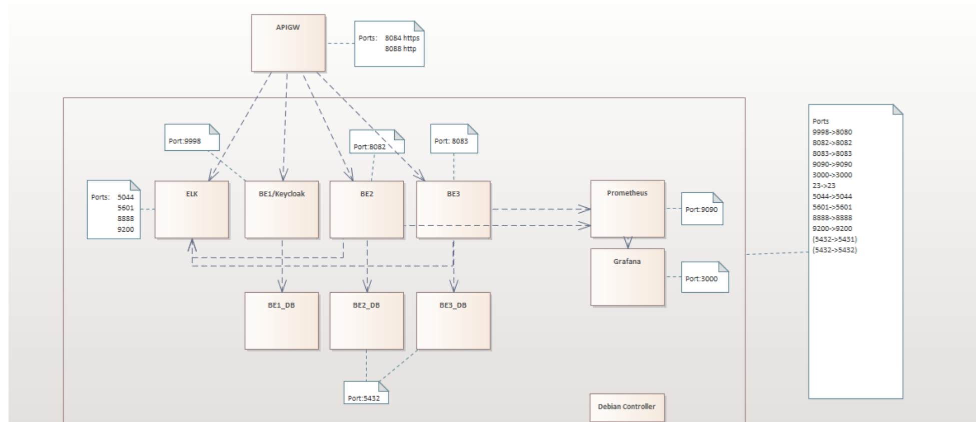


Obrázek 4.1: Obrázek vícevrstevnaté architektury. Zdroj [19]

Tento projekt používá v některých komponentech princip vícevrstevnaté architektury psanou v programovacím jazyku JAVA. Zbylé komponenty jsou různé open-source nástroje psané v jazycích jako jsou Javascript(Kibana), Ruby(Logstash), GO(Prometheus,Grafana), SQL(postgresSQL). Komponenty obsahující vícevrstevnatou architekturu mají jako prezenční vrstvu REST rozhraní pomocí kterého komunikují s ostatními komponenty. To umožňuje použití takzvaných DTO objektů které se v JAVĚ používají na komunikaci mezi jednotlivými REST rozhranímy. Dále se zde nachází business vrstva, která je v tomto případě reprezentovaná respondery a servisní vrstvou v některých komponentách. V případě perzistentní vrstvy se většinou využívá vrstva facade. Tato vrstva není v této práci použitá, protože nepotřebujeme nikde zjednodušovat ukládání dat z důvodu toho, že aplikace jsou PoC a neobsahují tudíž složitost potřebnou pro tuto vrstvu. Z tohoto důvodu perzistentní vrstvu zajištují pouze takzvané DAO obejkty, které využívají implementace tříd JpaRepository nabízející framework Spring Boot který obsahuje základní CRUD operace s daným objektem. Databázovou vrstvu v tomto případě řeší databáze PostresSQL se kterou komunikuje framework Spring Boot.

4.2 Diagram kontejnerů

V této sekci jsou znázorněny kontejtery jednotlivých komponent a styl jakým se mapují do hostitelského počítače. Dále tento obrázek slouží jako vizualizace jednotlivých komponent pro jednodušší pochopení rozložení a vzájemné komunikace. Na obrázku jsou vidět mikroservisy, které se pouští jako součást



Obrázek 4.2: Zobrazení stavby kontejnerů

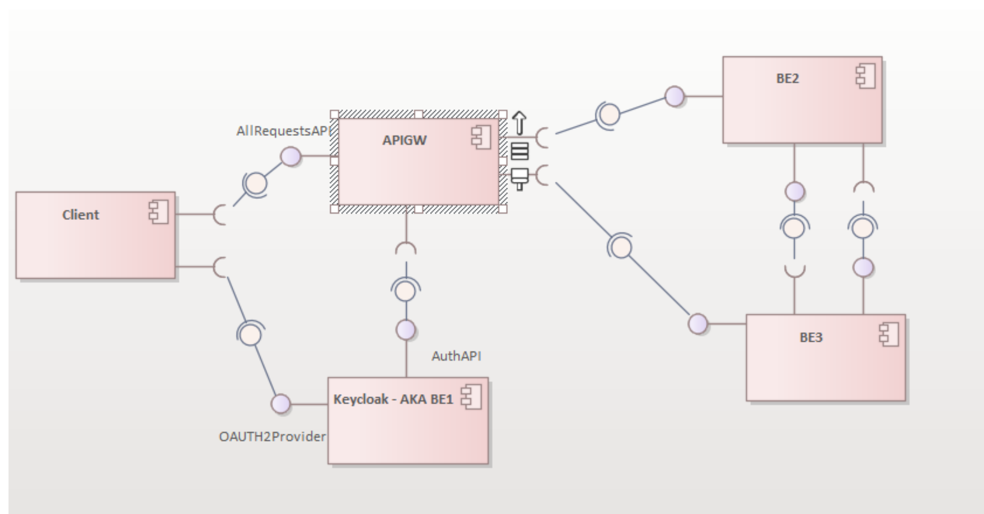
docker kontejnerů které je možné poznat tak že jsou obalené černým rámečkem. Dále je možné vidět mikroservisy spouštějící se mimo docker kontejnery. Narozdíl od ostaních jsou v podstatě spouštěny na hostitelském stroji. V tomto případě to je jen APIGW mikroservisa. Zároveň je na obrázku vidět jednotlivé mapování portů kontejerů na porty hostitelského počítače. Na obrázku můžeme také vidět čárkované šipky znázorňující závislosti. Navíc tyto šipky

znázorňují komunikaci mezi jednotlivými mikroservisami. Kromě toho je vidět ještě velká poznámka, která je vidět na pravé straně obrázku. Tato poznámka pouze agreguje malé poznámky a tvoří finální výpis mapovaných portů v dockeru.

- **APIGW**: Mikroservisa která zajištuje agregaci jednotlivých endpointů do jedné cesty. Dále zajištuje Oauth2 zabezpečení endpointů a tím chrání veškerou příchozí komunikaci.
- **BE2**: Mikroservisa obsahující první Javový backend.
- **BE3**: Mikroservisa obsahující druhý Javový backend.
- **BE1/Keycloak**: Mikrosevisa zajištující Oauth2 server, který poskytuje tokeny případným uživatelům a je využíván mikroservisou APIGW kvůli jejich případné kontrole.
- **Prometheus**: Mikroservisa pouštící prometheus server, který sbírá metriky Java backendů BE2 a BE3 a posílá je do mikroservisy Grafana.
- **Grafana**: Mikroservisa s grafanou která příjmá metriky od Prometheus servisy a zobrazuje je do dashboardů.
- **Debian** Controller: Mikroservisa zajištující vnitřní přístup z hostitelského stroje. Navíc je to volný prostor kam se dají doinstalovat nástroje pro diagnostiku jako je například curl.
- **BE1_DB**: Mikroservisa obsahující postgresSQL databázi pro Keycloak. Kde jsou uloženy údaje o uživatelích.
- **BE2_DB**: Mikroservisa obsahující postgresSQL databázi pro BE2
- **BE3_DB**: Mikroservisa obsahující postgresSQL databázi pro BE3
- **ELK**: Mikroservisa zajištující Elastic, Logstash a Kibanu v jednom kontejneru. To má výhodu že nemusí běžet více dockerů a je tam rovnou zajištěna vzájemná komunikace.Nicméně je to proti principům dockeru, které se snaží mít v jednom kontejneru jednu funkcionalitu. [3]

4.3 Diagram komponent

Diagram komponent v tomto projektu zobrazuje business logiku, kterou je možné provolat pomocí REST endpointu z pohledu vnějšího uživatele. Na obrázku je vidět Client komponenta, která reprezentuje webový prohlížeč, insomnii nebo jiného REST klienta. Ta nejdříve potřebuje získat access token z komponenty Keycloak. Následně může za pomoci access tokenu komunikovat skrz https s komponentou APIGW. Pokud by se pokusila komunikovat bez tokenu je vrácena http hodnota 403. Dále je vidět APIGW komponenta, která příjmá požadavky od Client komponenty a pak kontroluje s Keycloakem jejich platnost. Na konci jsou tu vidět komponenty BE2 a BE3 které zajišťují základní scénáře a dají se provolat z APIGW.



Obrázek 4.3: Component diagram

Kapitola 5

Implementace

5.1 Keycloak

Pro tento projekt je vybrána implementace keycloaku ze vzdáleného repositáře quay.io/keycloak/keycloak:latest, která je dále rozšířena o některé enviroment proměnné. Z důvodu toho že keycloak běží v dockeru na stejné síti jako ostatní kontejnery je potřeba modifikovat výchozí port aby nekolidoval s ostatními. Následně je potřeba nastavit překládání portu na hostitelský stroj z důvodu zjednodušení přístupu k webové konfiguraci. Kontejner si také kopíruje data do hostovacího stroje a připojuje se ke své databázi, aby byla případná upravená konfigurace ve webovém rozhraní perzistentně uložena na hostitelském stroji. Z důvodu vzdálené zprávy skrz docker je potřeba pomocí proměnných nastavit výchozí administrátorsky účet. Tyto proměnné jsou *KEYCLOAK_ADMIN=admin, KC_HTTP_PORT=9998, KEYCLOAK_ADMIN_PASSWORD=admin1*. Pomocí vytvořeného účtu se dá keycloak nastavit. Pokud by účet nebyl takto zpočátku nastaven tak není možné se k němu připojit protože kontejnerizovaná verze nepodporuje shell připojení a prvotní nastavení není z bezpečnostních důvodů povoleno z jiných adres kromě localhost adresy.

5.1.1 Konfigurace

V konfiguraci keycloaku je nastavený realm SpringBootKeycloak, který je používán na zabezpečení a obsahuje client login s názvem login-app který využívá pro kontrolu APIGW a testovacího usera s názvem user1 a heslem user1. Tento uživatel má roli user, která se dá pak také použít při kontrole v APIGW. Dále keycloak vystavuje endpoint ukazující jednotlivé certifikáty pro APIGW. Jeho url s cerifikáty je v případě toho projektu nastaveno na localhost:8080/realms/SpringBootKeycloak/protocol/openid-connect/certs.

5.2 Prometheus

Implementace Prometheus v kontejneru se bere z repozitáře prom/prometheus. Sběr metrik prometheus zajišťuje aktuálně ze 2 zdrojů (BE2 a BE3). Dále si

kopíruje konfigurační soubor prometheus.yml z hostitelského stroje a předává při startu jako config.file parameter. To zajišťuje že konfigurace z hostitelského stroje se použije pro prometheus server.

5.2.1 Konfigurace

Veškerá konfigurace se nachází v prometheus.yml na hostitelském stroji. Konfigurace obsahuje jak globální nastavení tak nastavení jednotlivých cílů na sbírání metrik. V globálním nastavení je nastaveno *scrape_interval* na 15 vteřin což nastavuje aktualizaci metrik na jednou za 15 vteřin. Dále je nastaveno *evaluation_interval*. Toto nastavení určuje míru, jak často bude Prometheus vyhodnocovat pravidla a vykonávat výpočty. Opět je interval nastaven na jednou za 15 sekund. Dále konfigurace obsahuje *scrape_configs*. Tato sekce definuje různé sběrnice (jobs), které Prometheus monitoruje. Stejně tak tato konfigurace obsahuje *job_name* který definuje název sběrnice (jobu) který je v tomto případě Backends. Kromě toho obsahuje položku static_configs ve které jsou targets. Targets jsou adresy stylu <host>:<port> v tomto případě *['b4bproj6be2:8082', 'b4bproj6be3:8083']*. Kromě toho obsahuje *metrics_path* atribut který říká na jaké url adrese má hledat metrics výstupy. Na konec jsou ještě důležité *scrape_params* do kterých se můžou dát parametry pro GET a POST požadavky.

5.3 Grafana

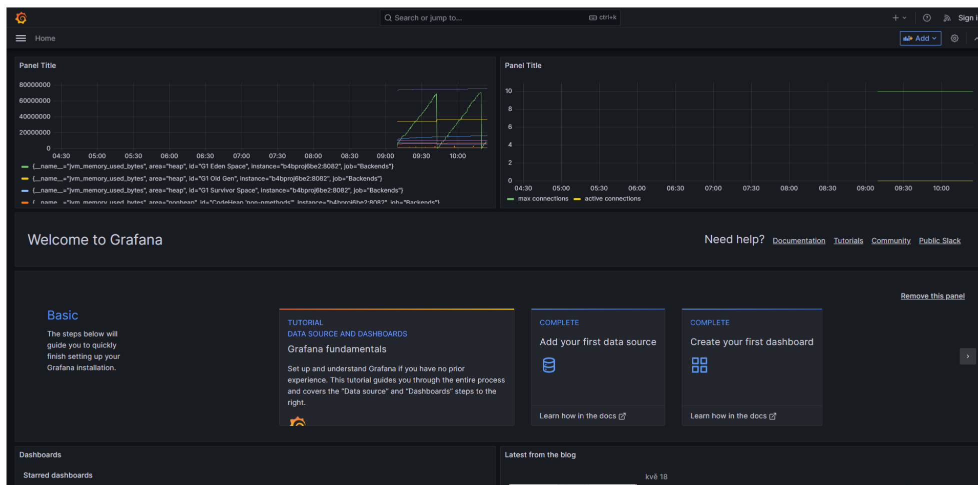
Vizualizační nástroj grafana je také implementován v kontejneru, který bere implementaci z grafana/grafana. Kontejner dodatečně překládá výchozí vnitřní port na stejný port hostitelského stroje, aby se ke grafaně povedlo dostat z hostitelského stroje pomocí webového rozhraní. Dále se také stará o ukládání dat grafany na hostitelský stroj. Stějně tak nastavuje hodnoty některých enviroment proměnných *GF_AUTH_ANONYMOUS_ENABLED=true* a *GF_AUTH_ANONYMOUS_ORG_ROLE=Admin*. Tyto proměnné zajišťují možnost se anonymně přihlásit do grafany, tudíž není potřeba vytvářet uživatelské účty pro jednotlivé uživatele.

5.3.1 Dashboards

Dashboardy v Grafaně jsou konfigurovatelné panely, které umožňují zobrazovat data z různých zdrojů a poskytují uživatelům přehledný a interaktivní pohled na klíčové metriky a informace. V této práci jsou vytvořeny 2 ukázkové. Jeden na použitou paměť (na obrázku vlevo) a druhý na maximum a počet aktivních připojení (na obrázku vpravo).

5.4 ELK

ELK stack implementace je celá obsažená v jednom kontejneru který ji bere z repozitáře sebp/elk:latest, tento zdroj obsahuje defaultně nastavený ELK



Obrázek 5.1: Grafana dashboard

stack. V konfiguraci kontejneru se dají měnit jednotlivé konfigurace, ale výchozí nastavení je funkční. Tento kontejner se nestará o mapování vnitřních dat do hostitelského počítače z důvodu nedostatku místa na hostitelském stroji. Jediné co mapuje je logstash.conf, který je potřeba pozměnit aby fungoval s logy ze Spring Boot applikací.

5.4.1 Logstash

Konfigurace logstash.conf je ve formátu JSON a obsahuje několik objektů. První z nich je *input* obsahující konfigurace jednotlivých vstupů do Logstash, neboli způsobů jak se mohou logy do Logstash dostat. Aktuálně zprovozněné jsou udp(s portem 5000 typem syslog a codecem json), tcp(s portem 8888 typem syslog a codecem json), http (s portem 5000 a codecem json). Druhý z nich je *filter*, který upravuje různé fieldy v logu, nebo může některé logy rovnou zahazovat. Nyní je zprovozněn filter, který k logům typu syslog přidá *field instance name* s hodnotou *%app_name-%host:%app_port*. Logovaní do Logstash je v případě této bakalářské práce řešeno pomocí stringu, to znamená že logstash neparsuje žádný objekt ale pouze předá dál data.

5.4.2 Kibana

Nastavení kibany neprobíhá skrz konfigurační soubor ale skrz webové rozhraní. Ve webovém nastavení je potřeba nastavit index pattern než kibana začne zobrazovat logy. Mimo jiné má kibana celkem dobře nastavené výchozí nastavení, takže nebyla potřeba žádná dodatečná konfigurace.

5.5 Debian

Tento Docker kontejner je založen na operačním systému Debian Linux a slouží jako prostředí pro ladění a debugování dotazů na jednotlivé Docker kontejnery.

Připojením k Docker kontejneru skrze shell je umožněno monitorovat a řešit problémy týkající se komunikace mezi jednotlivými kontejnery v rámci sítě. Byl použit převážně v situaci kdy bylo potřeba otestovat důvod proč nemůže prometheus sbírat metriky z APIGW kontejneru. Dále byl použit když bylo potřeba zjistit proč se nedá přistoupit do APIGW s příslušným Jwt tokenem. Většina tohoto testování probíhala pomocí nástroje curl.

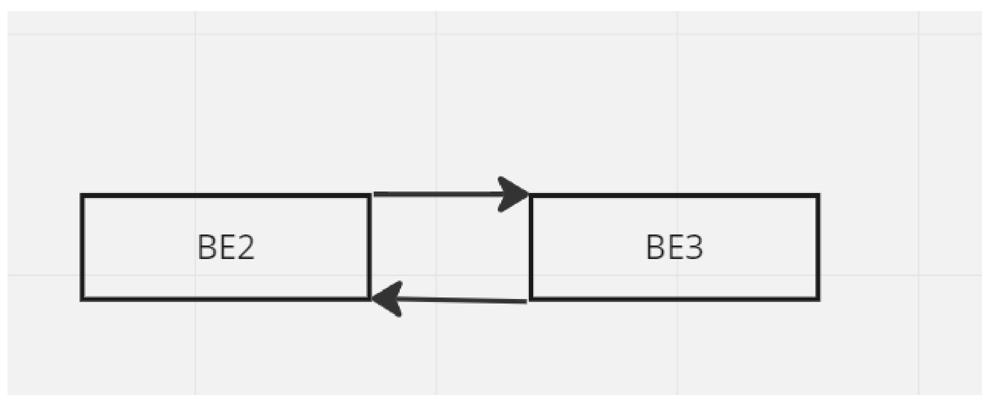
5.6 Databáze pro Keycloak

Keycloak využívá databázi pro ukládání uživatelských účtů, správa rolí a oprávnění, sledování stavu přihlášení a Auditní záznamy. Tento kontejner zajišťuje postgres databázi kde se data ukládají perzistentně na hostitelském stroji a je v docker-compose zajištěno že se pustí dříve než keycloak.

5.6.1 OWASP A03:2021 – Injection

Jeden z top 10 nejčastějších bezpečnostních problémů, Injection. Typicky se jedná o SQL injection, kde je problém v tom že aplikace neošetřuje vstupy když předává informace interpretorovi[2]. Pro zajištění ochrany aplikace proti Injection útokům byl využil koncept JPARepository. Tento koncept je součástí Java Persistence API (JPA) a slouží jako abstrakce pro práci s daty v relační databázi. Tato funkcionalita poskytuje výhody z hlediska bezpečnosti, jelikož využívá parametrizovaných dotazů a předdefinovaných metod pro manipulaci s daty. Riziko chybného vytváření dotazů snižují předdefinované metody a zároveň poskytují dodatečnou vrstvu ochrany proti útokům. Použitím tohoto konceptu je riziko Injection výrazně sníženo.

5.6.2 Circuit breaker



Obrázek 5.2: Smyčka v komunikaci

Zde na obrázku je vidět problém se smyčkou v mikroservisní architektuře. Smyčka vzniká tak, že se zacyklí závislosti jednotlivých mikroservis a volají se stále dokola. To má za následek výrazné zvýšení komunikace která neustává

a může být takto vykonáváno do nekonečna a nebo dokud nespadne jedna z mikroservis. Tomuto problému zabraňuje takzvaný circuit breaker pattern který monitoruje requesty a když se takto zacyklí tak na chvíli zamezí komunikaci s mikroservisou a přestane na požadavky reagovat.[14]

5.6.3 Vnitřní žádost / Vnější žádost

Dále bylo také potřeba vyřešit rozlišení mezi vnější a vnitřní žádostí. Z důvodu otazky zdali přiřadit correlationId a u vnitřních žádostí je potřeba logovat odkud žádost přišla, aby se dalo dohledat v logu co přesně způsobilo problém. To je řešené pomocí příznaku v hlavičce požadavku xJwtOrig, který obsahuje string s názvem mikroservisy která vytvořila žádost. Tento příznak pak loguje cílová mikroservisa.

Kapitola 6

Testování

Implementované řešení tohoto projektu bylo otestováno pomocí jednotkových testů a manuálního testování. Pro jednotkové testování byl použit framework JUnit, který umožňuje snadné psaní a spouštění testů v Javě. Manuální testování bylo prováděno prostřednictvím aplikace Insomnia a nástroje curl. Kontroly logů a metrik proběhly v ELK stacku a Grafaně.

6.1 Junit testy

JUnit je testovací framework pro testování softwarových aplikací. JUnit je populární nástroj pro automatizované unit testování v jazyce Java. V této práci jsou Junit testy psané pomocí generovacího nástroje SquareTest který využívá Mockito mockovacího frameworku. Tento nástroj se snaží o urychlení a zjednodušení unit testování aplikací pro vývojáře a testery. Jelikož všechny servisní vrstvy komponent v Javě neobsahují složitou logiku tak jsou pomocí tohoto nástroje jsou generovány všechny servisní vrstvy komponent.[13]

6.1.1 Pokrytí kódu

Pomocí testů Junit se autor snažil dosáhnout co největšího procentuálního pokrytí kódu testy, tak aby další případné změny v kódu nezměnily původní funkcionalitu a nebo alespoň dovedly k zamyšlení programátora při úpravě daného kódu. Pokrytí testy se řeší pouze v Autorem psaných komponentách tedy APIGW, BE2 a BE3. Celková class coverage se pohybovala u jednotlivých komponent psaných v JAVĚ tako BE2 25%, BE3 17% a APIGW 50%.

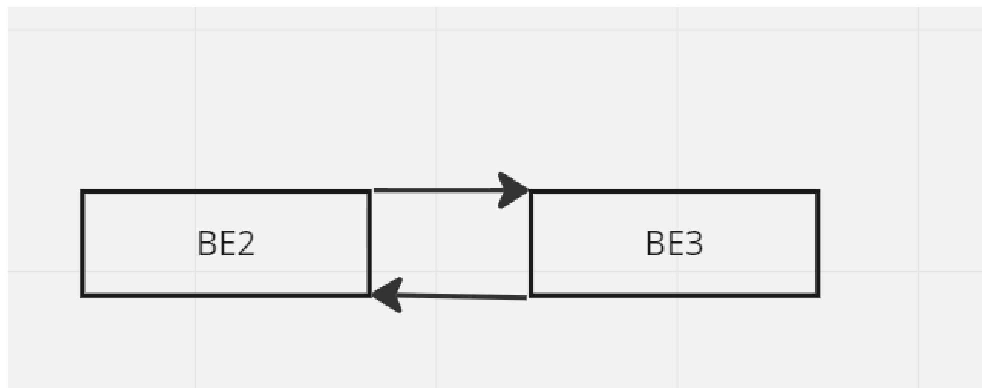
6.2 Uživatelské testování

Práce obsahuje 2 různé scénáře které testují implementaci Circruit breaker patternu, a výskyt JWT origin příznaku v hlavičce http žádosti.

6.2.1 Circuit breaker

Tento scénář testuje správné fungování circuit breaker patternu implementovaného na mikroservisach BE2 a BE3[14]. Za pomoci uměle vytvořené smyčky

v projektu. Na obrázku je vidět problém se smyčkou v mikroservisní architektuře.

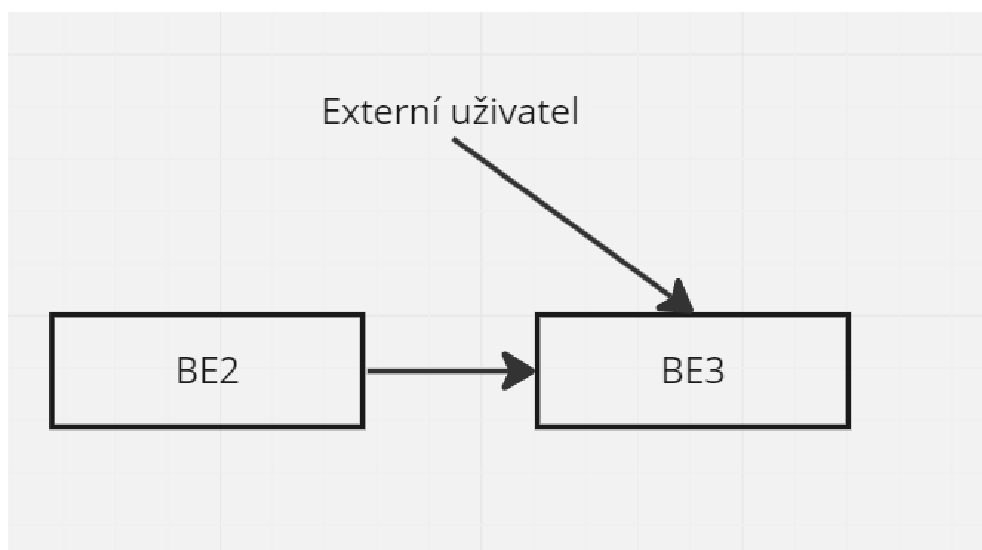


Obrázek 6.1: Smyčka v komunikaci

Tato smyčka může způsobit velké problémy, jak bylo popsáno v kapitole implementace. Její testování proběhlo jak skrze vnější volání, které prošlo skrz API Gateway, tak i pomocí vnitřního debian kontejneru. Pro testování pomocí debian kontejneru ve vnitřní síti byl využit nástroj curl.[1]

6.2.2 Vnitřní žádost / Vnější žádost

Tento scénář poukazuje na problém při provolávání API, které když selže tak se nedá zjistit zdali je to problém na straně mikroservisní architektury a nebo to byl špatně provolaný endpoint jiným klientem. Řešení tohohle scénáře je použití příznaku xJwtOrig v headeru requestu. To bude prázdné v případě vnějšího požadavku a jinak bude obsahovat název zdrojové mikroservisy v řípadě vnitřní žádosti. Na obrázku je možné vidět vizualizaci jednotlivých



Obrázek 6.2: Typy volání

volání. V případě vnitřního volání bude xJwtOrig obsahovat BE1. Testování

bylo provedeno provoláním BE3 z BE2 a napřímo z debian controller. Výsledky byly poté vyhodnoceny porovnáním jednotlivých logů z každého volání.

Kapitola 7

Závěr

Výstupem této práce je PoC mikroservisní architektury obsahující základní logování, sběr metrik a základní zabezpečení vnějšího přístupu. Logování bylo provedeno stylem vytvoření stringu v aplikaci, který byl pak poslán a zobrazil se ve vizualizaci Kibany. Vystavení metrik pomocí micrometeru a následný sběr pomocí nástroje prometheus do vizualizačního nástroje grafana. Zabezpečení protokolem Oauth2 za pomoci Keycloaku. Jedním z hlavních cílů autora bylo vyzkoušet složité logování a rozšířit znalost co se děje na pozadí Java aplikace když je nasazena v reálném prostředí. Tento osobní cíl autor považuje za splněný. Dále autor zjistil že se logování, metriky a zabezpečení musí přizpůsobit použitému prostředí a použité architektuře. Navíc ještě zjistil že případné změny dokážou být časově náročné na implementaci, proto je potřeba klást důraz na počáteční návrh a analýzu.

Jako další krok by se práce nechala rozšířit o další mikroservisy. Například mikroservisy projektu, které budou potřebovat už zařízené základní logování či nastavený sběr metrik. Dále pak s trochu větší úpravou, která bude značně zasahovat do aktuálního nastavení PoC je možné využít vnějšího zabezpečení pomocí OAuth2, poskytováno mikroservisou APIGW. Tato servisa se bude muset přepsat aby integrovala nově přidané mikroservisy. Dále by se dalo rozšířit o sbírání metrik z méně důležitých komponent a z komponent nástrojů, které mají také své metriky. V neposlední řadě by se tato práce dala rozšířit o zajištění zabezpečení komunikace mezi mikroservisama kvůli zajištění oproti Man in the middle útokům, aby se tato práce dala použít i v méně bezpečných prostředích. Jednotlivé mikroservisy a nástroje jsou na to částečně připravené, ale není to implementováno.

Literatura

[1]

[2] OWASP. https://owasp.org. Accessed on: May 17, 2023.

[3] Charles Anderson. Docker [software engineering]. *Ieee Software*, 32(3):102–c3, 2015.

[4] Baeldung. Spring boot security, 2021.

[5] Matthew Baker. Oauth2. In *Secure Web Application Development: A Hands-On Guide with Python and Django*, pages 351–397. Springer, 2022.

[6] Mainak Chakraborty and Ajit Pratap Kundan. Grafana. In *Monitoring Cloud-Native Applications: Lead Agile Operations Confidently Using Open Source Software*, pages 187–240. Springer, 2021.

[7] C. J. Date. *An introduction to database systems.* Addison-Wesley, 2004.

[8] Elastic. Logstash reference [7.0], 2019.

[9] Elastic. Kibana user guide, 2021.

[10] R. Elmasri and S. B. Navathe. *Fundamentals of database systems.* Pearson, 2016.

[11] Konrad Gos and Wojciech Zabierowski. The comparison of microservice and monolithic architecture. In *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, pages 150–153. IEEE, 2020.

[12] Jiawei Han, E Haihong, G Le, and J Du. Survey on nosql database. *Ieee Transactions on Knowledge and Data Engineering*, 25(10):2429–2443, 2013.

[13] Vincent Massol and Ted Husted. *JUnit in action.* Manning, 2004.

[14] Fabrizio Montesi and Janine Weber. Circuit breakers, discovery, and api gateways in microservices. *arXiv preprint arXiv:1609.05830*, 2016.

[15] Oracle. Java se technologies, 2021.

[16] Inc. Pivotal Software. Spring boot reference guide. 2021.

[17] Pallavi Prasad. *Getting Started with NoSQL*. Packt Publishing Ltd, 2015.

[18] DSVL Rao and AR Krishna. Big data analytics using elasticsearch and kibana: a review. *International Journal of Applied Engineering Research*, 14(23):4522–4528, 2019.

[19] Mark Richards and Neal Ford. Software architecture patterns. O'Reilly Media, 2015.

[20] Leonard Richardson and Sam Ruby. *RESTful web services*. "O'Reilly Media, Inc.", 2008.

[21] Navin Sabharwal, Piyush Pandey, Navin Sabharwal, and Piyush Pandey. Working with prometheus query language (promql). *Monitoring Microservices and Containerized Applications: Deployment, Configuration, and Best Practices for Prometheus and Alert Manager*, pages 141–167, 2020.

[22] Keycloak Team. Open source identity and access management.

[23] Johannes Thönes. Microservices. *IEEE software*, 32(1):116–116, 2015.

[24] James Turnbull. *Monitoring with Prometheus*. Turnbull Press, 2018.

[25] John Wallen. *Spring Boot Security: A Practical Guide to Secure and Manage Your Java Microservices*. Packt Publishing Ltd., 2019.