Diplomová práce

České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická

Systém pro vzdálený monitoring služeb

Bc. Nikita Iryupin

Školitel: Ing. Jan Smejkal
Květen 2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Iryupin** Jméno: **Nikita** Osobní číslo: **495544**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Systém pro vzdálený monitoring služeb

Název diplomové práce anglicky:

System for remote monitoring of services

Pokyny pro vypracování:

Navrhněte a implementujte systém pro vzdálený monitoring služeb umožňující provozovateli služeb definovat vlastní metriky a ty následně pomocí agenta sbírat a na serveru zobrazit.
Systém se bude skládat minimálně z následujících součástí.
- Server pasivně přijímající metriky ze vzdálených služeb.
- Webovou aplikaci poskytující náhled do nasbíraných metrik.
- Agenta umožňujícího vzdáleným službám aktivně poskytovat metriky.

Navrhněte a implementujte integraci na existující monitorovací systémy, např. Zabbix a Prometheus, umožňující zpracovat agentem nasbíraná data.
Server a knihovnu implementujte na .NET platformě. Webovou aplikaci implementujte pomocí frameworku Vue.js.
Funkčnost celého systému ověřte minimálně pomocí unit testů a integračních testů.

Seznam doporučené literatury:

1. Heitor, Ribeiro R. 2020. Vue.Js 3 Cookbook: Discover Actionable Solutions for Building Modern Web Apps with the Latest Vue Features and TypeScript. Packt Publishing.
2. Lauret, Arnaud. 2019. The Design of Web APIs. 1st ed. Manning.
3. Price, Mark J. 2022. Apps and Services with .NET 7: Build Practical Projects with Blazor, .NET MAUI, GRPC, GraphQL, and Other Enterprise Technologies. 1st ed. Packt Publishing.
4. Zabbix : The Enterprise-Class Open Source Network Monitoring Solution. https://www.zabbix.com/.
5. Prometheus - Monitoring System & Time Series Database. https://prometheus.io/.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing Jan Smejkal Merica s.r.o.

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

Datum zadání diplomové práce: **05.02.2024** Termín odevzdání diplomové práce: **24.05.2024**

Platnost zadání diplomové práce: **21.09.2025**

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

III. PŘEVZETÍ ZADÁNÍ

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

. _____ _____

Datum převzetí zadání Podpis studenta

Poděkování

Tímto bych rád poděkoval vedoucímu mé diplomové práce Ing. Janovi Smejkalovi za všestrannou pomoc, množství cenných a inspirativních rad, podnětů, doporučení, připomínek a zároveň za velkou trpělivost s obdivuhodnou ochotou při konzultacích poskytnutých ke zpracování této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 24. května 2024

Nikita Iryupin

Abstrakt

Monitoring softwaru je nedílnou součástí vývoje a podpory softwaru. Většina monitorovacích systémů vyžaduje vystavení endpointu nebo instalaci dalšího softwaru a jeho nastavení. Tato práce řeší problém aktivního monitorování a přidává k monitorování tak důležitou věc, jako je mapování hierarchie sbíraných metrik. Toto mapování je nesmírně důležité, protože umožňuje uživatelům lépe porozumět vztahům mezi různými částmi systému a způsobem, jakým jednotlivé metriky ovlivňují celkový výkon a stabilitu aplikace. Tím se zvyšuje schopnost identifikovat a řešit problémy efektivněji, což vede k lepšímu řízení a optimalizaci provozovaných systémů. Agent a server jsou napsány v jazyce C#, frontend ve Vue.js. Komunikace mezi agentem a backendem probíhá pomocí GRPC, agent je také schopen odesílat data do Zabbixu a Prometheu.

Klíčová slova: Aktivní monitorování, .NET, Vue.js, Zabbix, Prometheus

Školitel: Ing. Jan Smejkal

Abstract

Software monitoring is an integral part of software development and support. Most monitoring systems require the issuance of an endpoint or the installation of additional software and its setup. This work solves the problem of active monitoring and adds to monitoring such an important thing as mapping the hierarchy of collected metrics. This mapping is extremely important because it allows users to better understand the relationships between different parts of the system and how each metric affects the overall performance and stability of the application. This increases the ability to identify and solve problems more effectively, leading to better management and optimization of the systems in operation. The agent and server are written in C#, the frontend in Vue.js. Communication between the agent and the backend is done using GRPC, the agent is also able to send data to Zabbix and Prometheus.

Keywords: Active monitoring, .NET, Vue.js, Zabbix, Prometheus

Title translation: System for remote monitoring of services

Obsah

1 Úvod **1**
1.1 Motivace 1
1.2 Cíle 1

2 Sběr požadavků **3**
2.1 Funkční požádavky 3
2.2 Nefunkční požádavky 4
2.3 Definice pojmů 5

3 Analýza **7**
3.1 Rešerše existujících řešení 7
3.1.1 Zabbix 7
3.1.2 Prometheus 9
3.1.3 Datadog 12
3.1.4 PRTG 12
3.2 Závěr 13

4 Návrh **17**
4.1 Architektura 17
4.2 Agent 17
4.2.1 Architektura agenta 17
4.2.2 Data 19
4.2.3 Agregace 20
4.2.4 Způsob odesílání dat 21
4.2.5 Porovnání způsobů odesílání dat pro backend 22
4.3 Backend 23
4.3.1 Architektura backendu 24
4.3.2 Grafová databáze 24
4.4 Frontend 27
4.4.1 Grafy 27
4.4.2 Dashboardy 27
4.4.3 Prototyp 28
4.4.4 Výběr knihovny na zobrazení grafu 29
4.4.5 Výběr knihovny na zobrazení chartu 31
4.5 Integrace 33
4.5.1 Zabbix 33
4.5.2 Prometheus 33
4.6 Authorizace 33

5 Implementace **35**
5.1 Vývojové prostředí 35
5.2 Agent 35
5.3 Backend 38
5.4 Frontend 38
5.5 Závěr 40

6 Nasazení **41**
6.1 Konfigurace NGINX 41
6.2 Bezpečnost a certifikáty 41
6.3 Deployment aplikace 42
6.4 Monitoring a logování 42

7 Testování **43**
7.1 Testování serveru 43
7.2 Testování agenta 43
7.3 Testování frontendu 43
7.4 Uživatelské testování 44
7.4.1 Testování prototypu 44
7.4.2 Testování aplikace 44
7.5 Závěr testování 46

8 Závěr **47**
8.1 Výsledky 47
8.2 Možná rozšíření 48

Literatura **49**

Obrázky

1.1 Komponenty 2

3.1 Zabbix 8
3.2 Prometheus 10

4.1 Architektura řešení 18
4.2 Architektura agenta 19
4.3 Strom metrik 20
4.4 Benchmark pro serializaci a deserializaci 23
4.5 Prototyp UI grafu 28

5.1 Backend implementace 38
5.2 Vytvoření dashboardu 39
5.3 Dashboardy FE 39

6.1 Nasazení na serveru 42

7.1 Výsledky testu na agentovi 44

Tabulky

2.1 Definice pojmů 5

3.1 Porovnání existujících řešení . . . 15

4.1 Podpora typ/agregace 20
4.2 Tabulka aplikace agent/FE agregace pro číselné typy 28
4.3 Tabulka aplikace agent/FE agregace pro typ string 29

Kapitola 1

Úvod

1.1 Motivace

Detailní monitoring významně napomáhá hladkému provozu většiny existujících informačních systémů. Často se o monitorování uvažuje v souvislosti s provozem síťových komponent nebo počítačů. Ve světě mikroslužeb a lokálních sítí je však třeba sledovat i to, jak funguje jednotlivá služba nebo aplikace. Na trhu se objevuje spousta monitorovacích aplikací, ale jen málo z nich komunikuje přímo se softwarem. Touto prací chci k tomuto problému přistoupit a představit řešení pro monitorování v těchto scénářích.

1.2 Cíle

Cílem diplomové práce je vytvořit aplikaci, která bude shromažďovat data ze serverů, aplikací a mikroslužeb a vizualizovat je [1]. Vzhledem k tomu, že pracujeme v prostředí, kde není vždy možné otevřít port nebo přistupovat ke konkrétnímu rozhraní, je vhodné věnovat pozornost způsobu odesílání dat, rychlosti zpracování a také snížení režijních nákladů spojených s nastavením monitorování. Jde o agent-aktivní monitoring, kde agent posílá aktivně data na server.

Dle zadání se požadované řešení dělí na tři části:

- Server, který pasivně přijímá metriky ze vzdálených služeb.

  Tento server má sloužit jako centrální úložiště pro shromažďování metrik pasivně odesílaných vzdálenými službami. Naslouchá na zadaném portu nebo koncovém bodě a zpracovává příchozí data.

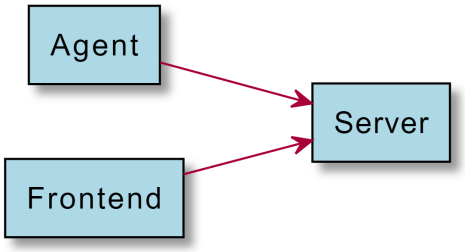
- Webová aplikace, která poskytuje zobrazení shromážděných metrik.

Webová aplikace, která má poskytovat uživatelské rozhraní pro zobrazení a analýzu metrik shromážděných serverem.

- Agent, který umožňuje vzdáleným službám aktivně poskytovat metriky.

  Tento agent má být nainstalován na vzdálených službách a aktivně shromažďuje a odesílá metriky na server.

Chci vývojářům usnadnit integraci, ale také ponechat možnost přizpůsobit sběr metrik a typy agregace, které chce vývojář na data aplikovat.



Obrázek 1.1: Komponenty

Kapitola 2

Sběr požadavků

Na základě zadání bude celý monitoring rozdělen na tři části - knihovnu (agenta), server (backend) a webovou aplikaci (frontend). Zde popíšeme konkrétní funkční a nefunkční požadavky pro celý systém a v kapitole návrh popíšu, jak a která část aplikace bude zodpovědná za určité funkce. Některé požadavky na projekt byly uvedeny v zadání, některé byly dohodnuty s vedoucím této práce.

2.1 Funkční požádavky

FR1 Shromažďování dat ze služby ve formátu celých čísel, reálných čísel, textu (hodnoty nebo protokoly) a výčtu.

Aplikace musí být schopná shromažďovat různé typy dat ze sledovaných služeb. Tyto typy zahrnují celá čísla, desetinná čísla (double), textové hodnoty nebo protokoly a výčty. Toto zajišťuje, že aplikace bude schopná monitorovat a zpracovávat širokou škálu datových typů, které mohou být generovány různými službami a aplikacemi.

FR2 Seskupení serverů/instancí podle softwaru/serveru/místa nasazení.

Aplikace musí umožnit seskupení monitorovaných serverů a instancí podle různých kritérií, jako je typ softwaru, konkrétní server nebo místo nasazení. Toto seskupení usnadní správu a sledování velkého množství instancí a umožní uživatelům rychleji identifikovat a analyzovat výkon a problémy v různých částech jejich infrastruktury.

FR3 Zobrazení shromážděných dat a jejich hierarchie.

Aplikace musí poskytovat možnosti vizualizace shromážděných dat, včetně hierarchického zobrazení. Toto umožní uživatelům lépe porozumět struktuře a vztahům mezi různými metrikami a komponentami systému.

Hierarchické zobrazení pomůže při identifikaci vzorců a problémů, které mohou být způsobeny závislostmi mezi různými částmi systému. Uživatel by měl mít možnost vytvářet dashboardy [2] s daty, kde bude rozhodovat o způsobu zobrazení metrik ve widgetech.

FR5 Přihlašování.

Aplikace musí zahrnovat mechanismus pro přihlašování uživatelů, který zajistí bezpečný přístup k monitorovacím funkcím a datům. Tento mechanismus musí podporovat různé úrovně uživatelských oprávnění, aby bylo možné řídit přístup k citlivým datům a funkcím na základě rolí uživatelů.

FR6 Integrace s Prometheus - posílání dostupných itemů na Prometheus Server.

Aplikace musí být schopná se integrovat s monitorovacím systémem Prometheus. Toto zahrnuje posílání dostupných metrik a itemů na Prometheus Server, což umožní využít pokročilé monitorovací a alertovací funkce tohoto systému. Integrace s Prometheus zajistí, že uživatelé mohou využívat své stávající monitorovací infrastruktury a nástroje.

FR7 Integrace s Zabbix - posílání dostupných itemů na Zabbix Server.

Aplikace musí podporovat integraci s monitorovacím systémem Zabbix. Toto zahrnuje posílání dostupných metrik a itemů na Zabbix Server, což umožní uživatelům využívat funkce Zabbixu pro sledování a správu jejich infrastruktury. Integrace se Zabbix zajistí, že aplikace bude kompatibilní s různými monitorovacími řešeními a nabídne flexibilitu v nasazení.

2.2 Nefunkční požádavky

NFR1 Zabezpečení.

Aplikace musí odesílat data zabezpečenými kanály, aby nedošlo k úniku dat ze serverů.

NFR2 Škálovatelnost.

Agent musí být schopen odesílat data na více serverů, stejně jako server musí být schopen přijímat data od více agentů.

NFR3 Programovací jazyky

Agent a server by měly být napsány na platformě .NET, pokud možno s podporou systémů Windows a Linux. Frontend by měl být napsán

v jazyce Vue.js. Je žádoucí používat nejnovější verze LTS platforem a knihoven.

2.3 Definice pojmů

Pojem	Význam
Agent, knihovna	Software, součastí monitorovaného systému, která odesílá data na server.
Server, backend	Software, který přebírá data od agenta a ukládá je.
UI, frontend	Software, který zobrazuje data.
Item, položka	Položka, podle které se sbírají hodnoty. Má název a určitý typ.
Metrika	To samé jako „item“.
Hodnota, value	Hodnota metriky s jednoznačným typem.
Graf metrik	Graf uzlů a hran
Chart	Graf časové řady
Strom metrik	Graf metrik s jediným počatečním úzlem

Tabulka 2.1: Definice pojmů

Kapitola 3

Analýza

Téma monitorování softwaru je diskutováno již delší dobu a existuje poměrně dost projektů, které se snaží tento problém řešit. [4] Podívejme se na několik z nich.

3.1 Rešerše existujících řešení

Existuje několik populárních řešení monitorovacích systémů. Primárně se zaměříme na Zabbix a Prometeus jako jedny z nejrozšířenějších monitorovacích systémů. Cílem je identifikovat silné a slabé stránky a nastínit klíčové faktory, které jsou pro monitorování důležité. Stručně projdeme i další projekty (Datadog, PRTG).

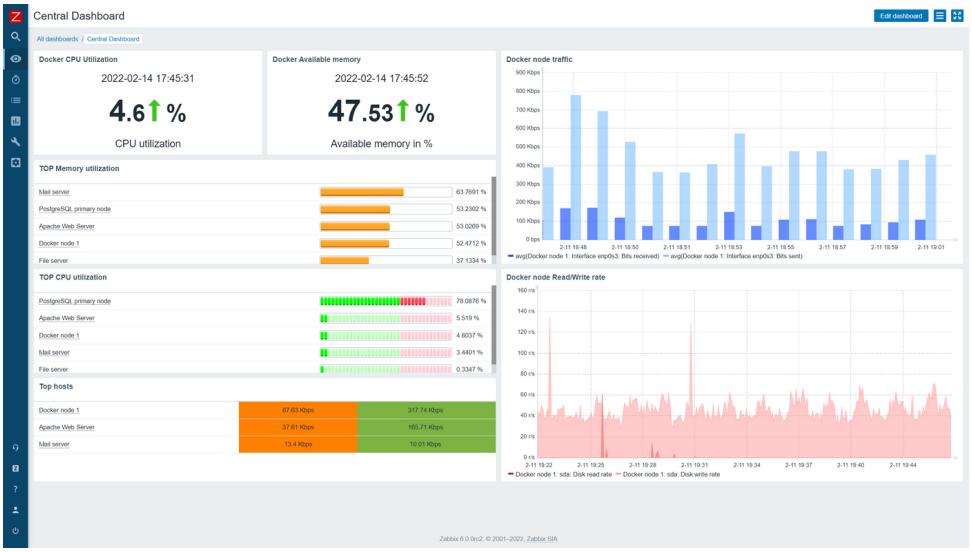
Cílem tohoto projektu je zjistit slabé a silné stránky stávajících projektů a navrhnout řešení, které by řešilo případná omezení.

3.1.1 Zabbix

Zabbix [5] je open-source řešení pro monitorování a správu sítě určené k monitorování výkonu a dostupnosti serverů, síťových zařízení a dalších prostředků IT. Poskytuje komplexní sadu funkcí pro monitorování, upozorňování, vizualizaci a vytváření zpráv. Zde jsou některé klady a zápory Zabbixu.

Kladné stránky

Otevřený zdrojový kód Zabbix je řešení s otevřeným zdrojovým kódem, což znamená, že je volně k dispozici k použití a lze jej přizpůsobit konkrétním potřebám.



Obrázek 3.1: Zabbix

Škálovatelnost Zabbix je vysoce škálovatelný, takže je vhodný pro prostředí všech velikostí. Dokáže monitorovat velké množství zařízení a horizontálně se škálovat, aby se přizpůsobil rostoucí infrastruktuře.

Všestrannost Podporuje monitorování různých typů zařízení a technologií, včetně serverů, síťových zařízení, virtuálních počítačů, cloudových služeb a dalších.

Komplexní monitorování Zabbix poskytuje širokou škálu možností monitorování, včetně využití procesoru, využití paměti, výkonu sítě, monitorování aplikací a vlastních metrik. Podporuje monitorování založené na agentech a bez agentů.

Upozorňování a oznámení Zabbix umožňuje uživatelům nastavit flexibilní mechanismy upozorňování a oznamování na základě předem definovaných spouštěčů. Upozornění lze zasílat e-mailem, SMS nebo pomocí vlastních skriptů.

Grafy a reporty Nabízí výkonné možnosti vizualizace dat, které uživatelům umožňují vytvářet grafy a reporty pro lepší analýzu trendů výkonu a historických dat.

Automatizace Zabbix podporuje automatizaci pomocí šablon a maker, což umožňuje efektivní a konzistentní konfiguraci na více zařízeních.

Podpora komunity Díky tomu, že je Zabbix open source, těží z velké a aktivní komunity uživatelů. Tato komunitní podpora může být cenná při řešení problémů a sdílení osvědčených postupů.

Mínusy

Složitá konfigurace Zejména pro uživatele, kteří se s řešeními pro monitorování sítě teprve seznamují, může být nastavení a konfigurace Zabbixu náročnější na učení. Rozhraní může být pro začátečníky nepřehledné.

Náročnost na zdroje V závislosti na rozsahu nasazení může být Zabbix náročný na zdroje. Větší instalace mohou vyžadovat značné prostředky serveru, včetně procesoru a paměti.

Návrh uživatelského rozhraní Někteří uživatelé považují uživatelské rozhraní za méně intuitivní a vizuálně přitažlivé ve srovnání s jinými monitorovacími řešeními. To je však subjektivní a závisí na individuálních preferencích.

Omezené nativní integrace Zabbix sice podporuje širokou škálu typů monitorování, ale ve srovnání s některými komerčními řešeními může mít méně nativních integrací. To však lze často řešit pomocí vlastních skriptů, ale to velice ovlivňuje rychlost.

Závěr

Zabbix je především určen na monitoring hardwaru. Konfigurace monitorovacího softwaru Zabbix také není tak snadná pro SW, protože vyžaduje, aby vývojář nainstaloval Zabbix Agenta verze 2 nebo nakonfiguroval komunikaci monitorovaných položek po síti.

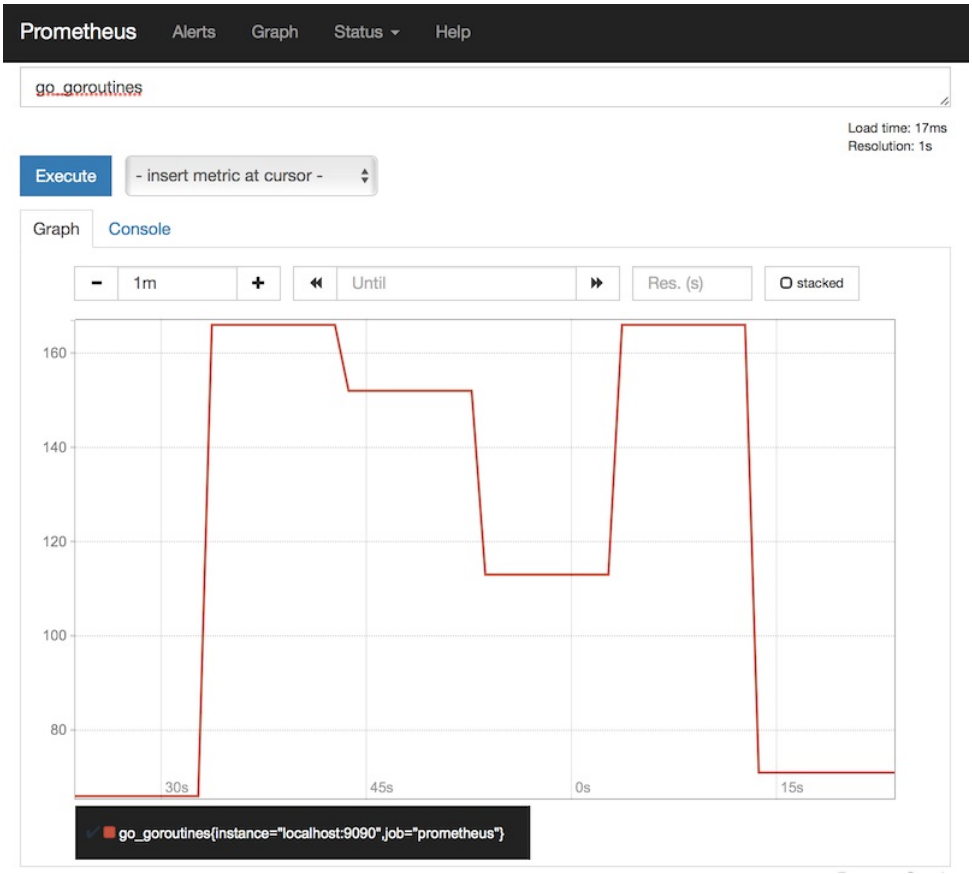
3.1.2 Prometheus

Prometheus je open-source sada nástrojů pro monitorování a upozorňování navržená pro spolehlivost a škálovatelnost v dynamických prostředích. Zde jsou některé výhody a nevýhody systému Prometheus.

Kladné stránky

Datový model Prometheus používá flexibilní datový model založený na dvojicích klíč-hodnota, což usnadňuje reprezentaci a dotazování na časové řady dat. Tento model poskytuje výkonný a expresivní dotazovací jazyk (PromQL) pro získávání poznatků ze sledovaných dat.

Škálovatelnost Prometheus je navržen tak, aby byl vysoce škálovatelný, a řídí se modelem pull-based, kdy každý server Prometheus nezávisle



Obrázek 3.2: Prometheus

shromažďuje metriky ze sledovaných cílů. Díky tomu je vhodný pro dynamická a rozsáhlá prostředí.

Dynamické zjišťování služeb Prometheus podporuje dynamické zjišťování služeb, což mu umožňuje automaticky zjišťovat a monitorovat nové instance služeb, jakmile se stanou online nebo offline.

Upozorňování Prometheus obsahuje vestavěný systém výstrah, který uživatelům umožňuje definovat pravidla výstrah na základě dotazovaných metrik. Výstrahy lze zasílat různými kanály, například e-mailem, přes Slack nebo prostřednictvím dalších integrací.

Integrace s Grafanou Prometheus lze snadno integrovat s populární open-source analytickou a monitorovací platformou Grafana. Grafana poskytuje uživatelsky přívětivé rozhraní pro vytváření ovládacích panelů a vizualizaci dat Prometheus.

Podpora komunity Stejně jako Zabbix, i Prometheus těží ze silné a aktivní open-source komunity. Komunitní podpora je cenná pro sdílení znalostí, řešení problémů a rozšiřování funkcí pomocí integrací třetích stran.

Cloud native Prometheus je vhodný pro cloudové architektury a prostředí mikroslužeb. Má nativní podporu pro systémy orchestrace kontejnerů, jako je Kubernetes.

Ekosystém exportérů Prometheus má bohatý ekosystém exportérů s exportéry dostupnými pro různé aplikace a systémy. Tyto exportéry umožňují Prometheovi získávat metriky z různých typů služeb.

Mínusy

Nedostatek vestavěného úložiště Prometheus je sice vysoce škálovatelný, ale neposkytuje dlouhodobé úložiště pro historická data hned po instalaci. Uživatelé často musí nastavit další komponenty, jako je vzdálené úložiště Prometheus, nebo použít externí řešení pro dlouhodobé ukládání a analýzu.

Náročný PromQL Pro uživatele, kteří neznají jeho syntaxi a sémantiku, může být dotazovací jazyk PromQL sice výkonný, ale může být náročný na učení. Se zkušenostmi se však stává intuitivním.

Grafické rozhraní Prometheus má základní webové rozhraní pro zadávání dotazů a prohlížení metrik, ale může postrádat některé pokročilé grafické funkce, které se vyskytují v jiných monitorovacích řešeních. Pro bohatší vizuální zážitek se často doporučuje integrace grafického rozhraní Grafana.

Podpora systému Windows Prometheus se primárně zaměřuje na systémy podobné Unixu, ale má omezenou podporu pro Windows, což může být důvodem pro prostředí s významným zastoupením Windows.

Centralizovaná správa V rozsáhlých nasazeních může správa více instancí Promethea vyžadovat další nástroje, protože neexistuje žádný vestavěný systém centrální správy pro koordinaci konfigurací.

Závěr

Závěrem lze říci, že Prometheus je výkonné a škálovatelné řešení pro monitorování se zaměřením na cloudová prostředí. Jeho flexibilní datový model a dotazovací jazyk spolu s živou komunitou jej předurčují pro dynamické a moderní IT architektury. Uživatelé by si však měli být vědomi aspektů, jako je dlouhodobé ukládání a náročnost spojená s jazykem PromQL.

Hlavní nevýhodou Promethea je absence položek typu string. Můžete tedy shromažďovat pouze číselné hodnoty. Také Pushgateway, který je vyžadován pro pasivní serverové monitorování, se striktně nedoporučuje používat. Pro

naši integraci ji použijeme, protože jiné možnosti, jak něco dynamicky odeslat do Promethea, neexistují.

3.1.3 Datadog

Datadog je moderní monitorovací a analytická platforma, která je navržena pro cloudovou infrastrukturu a aplikace. Nabízí širokou škálu nástrojů a funkcí, které umožňují sledovat výkon, stav a bezpečnost aplikací a systémů. Mezi klíčové vlastnosti patří:

- **Monitorování metrik**: Sbírá a vizualizuje metriky z různých zdrojů v reálném čase.
- **Správa logů**: Umožňuje centralizované sbírání, ukládání a analýzu logů.
- **Monitorování síťového provozu**: Poskytuje přehled o síťové komunikaci a výkonnosti.
- **Aplikace a infrastruktura**: Podporuje monitorování jak aplikací, tak i podkladové infrastruktury (servery, kontejnery, databáze).
- **Integrace**: Nabízí širokou škálu integrací s dalšími nástroji a službami (AWS, Azure, Google Cloud, Docker, Kubernetes, a mnoho dalších).
- **Alerty a notifikace**: Umožňuje nastavit upozornění a notifikace pro různé události a anomálie.

3.1.4 PRTG

PRTG Network Monitor je kompletní řešení pro monitorování sítě od společnosti Paessler. Tento nástroj umožňuje sledovat stav a výkon všech síťových zařízení a systémů. Mezi jeho hlavní vlastnosti patří:

- **Monitorování sítě**: Sleduje všechny aspekty sítě, včetně šířky pásma, využití CPU, paměti, a dalších zdrojů.
- **Senzory**: PRTG používá senzory k monitorování specifických aspektů zařízení nebo služeb. Každý senzor monitoruje jeden parametr, např. dostupnost HTTP, zatížení CPU nebo teplotu serveru.
- **Dashboardy a vizualizace**: Nabízí přehledné a přizpůsobitelné dashboardy a grafy pro vizualizaci dat.
- **Upozornění a notifikace**: Umožňuje nastavit upozornění na různé typy problémů a anomálií prostřednictvím emailu, SMS, push notifikací a dalších kanálů.

- **Reporting**: Generuje podrobné reporty o výkonu a stavu sítě.
- **Automatizace a skripty**: Podporuje automatizaci úkolů a použití vlastních skriptů pro specifické potřeby monitorování.

3.2 Závěr

V této rešerši jsme se zaměřili na několik populárních monitorovacích systémů, konkrétně na Zabbix, Prometheus, a stručně Datadog a PRTG. Každý z těchto systémů nabízí specifické funkce a výhody, které mohou být vhodné pro různé typy prostředí a požadavky.

Zabbix je robustní open-source řešení s širokou škálou možností monitorování a silnou komunitní podporou. Je vhodný pro rozsáhlá prostředí, která vyžadují škálovatelnost a flexibilní konfiguraci. Nicméně jeho náročnější nastavení a potřeba značných systémových prostředků mohou být překážkou pro některé uživatele, zejména začátečníky.

Prometheus je ideální volbou pro dynamická a cloudová prostředí díky své škálovatelnosti, flexibilnímu datovému modelu a silné integraci s kontejnery a mikroslužbami. Jeho výkonný dotazovací jazyk PromQL a podpora dynamického zjišťování služeb jej činí velmi atraktivním pro moderní IT architektury. Přestože může vyžadovat dodatečné nastavení pro dlouhodobé ukládání dat a může být náročný na učení, jeho výhody převyšují tyto nevýhody pro mnoho organizací.

Datadog nabízí komplexní monitorování s důrazem na metriky, logy a síťový provoz. Jeho silné stránky zahrnují snadnou integraci s mnoha službami a nástroji, což jej činí velmi užitečným pro prostředí s různorodou infrastrukturou. Datadog je obzvláště vhodný pro organizace, které hledají jednotné řešení pro monitorování a analýzu dat v reálném čase.

PRTG Network Monitor je velmi užitečný pro sledování sítě a všech jejích aspektů, včetně šířky pásma, využití CPU, paměti a dalších zdrojů. Díky svým flexibilním senzorům a silným možnostem vizualizace dat poskytuje přehledné a přizpůsobitelné dashboardy a grafy. Automatizace úkolů a generování podrobných reportů činí PRTG ideálním pro organizace, které potřebují detailní přehled o stavu a výkonu svých síťových zařízení.

Každý z těchto monitorovacích systémů má své silné a slabé stránky, a volba správného nástroje závisí na specifických potřebách a požadavcích každé organizace. Zabbix a Prometheus jsou silnými hráči v open-source komunitě s různými přístupy k monitorování, zatímco Datadog a PRTG nabízejí robustní komerční řešení s širokou škálou funkcí.

Na základě výsledků studie můžeme konstatovat, že každé řešení má svá

omezení. Prometheus neumí pracovat s řetězcovými daty a Zabbix má problémy s aktivním monitorováním bez instalace agenta na server. Ani jeden ze systémů nepodporuje agregaci, pouze na úrovni vytváření konkrétních typů. Žádné z řešení také nenabízí možnost vytvářet hierarchii dat. To zohledníme při vytváření architektury a implementace.

	Zabbix	**Prometheus**	**Datadog**	**PRTG**
Open Source	✓	✓		
Rok	2001	2012	2010	2010
Hlavní účel	Monitoring Infrastruktury s možností monitoringu web servis (vyžaduje extra nastavení)	Ukládání a získávání číslených hodnot/metrik pro microservisy	Sbírání a zobrazování metrik, infrastruktura + web servisy pomocí knihovny	Monitorování IT infrastrukury
Aktivní způsob komunikace (agent → server)	Ano, Zabbix agent verze 2 může posílat active checky, ale předtím potřebuje definovat seznam itemů.	Taková komunikace není v Prometheu. (jenom pomocí Pushgateway)	Záleží na druhu metriky, ale jinak - ano	Taková komunikace není v PRTG
Pasivní způsob komunikace (server → agent)	Ano, to je default chování agentů. Posílá požadavek agentovi na každý item zvlášť.	Server sbírá data z hostů a ukládá je na serveru. (node_exporter)	Záleží na druhu metriky, ale jinak - ano	Jednoduché dotažení
Druh dat	Text, logy, číselné hodnoty	Číselné hodnoty	Číselné hodnoty	Číselné hodnoty
Notifikace	Email, Jabra, dokonce i user-defenovane scripty	Alertmanager (PagerDuty, email and etc)	Jira, PagerDuty, Slack, Webhooks	Email, SMS
Zobrazování dat	Zabbix Frontend, je možné definovat grafy a tabulky s itemy	Má primitivní UI, většinou se používá spolu s Grafanou	Datadog UI, dashboardy and charty	Sondy a primitivní grafy
Komunikace (protocol)	JSON-based HTTP, SMNP, JMX, IPMI, SSH	HTTP	TCP or UDP (StatsD), HTTP a t.d.	SNMP, WMI, HTTP

Tabulka 3.1: Porovnání existujících řešení

Kapitola 4

Návrh

4.1 Architektura

Aplikace bude rozdělena na 3 části, z nichž každá bude plnit určitou funkci. (diagram na strance 2)

1 Knihovna (dál Agent)

2 Server (dál Backend)

3 Uživatelské rozhrání (dál Frontend)

Každá část systému bude popsána ve vlastní podkapitole. Celý systém je založen na schématu pasivního monitorování - agent (v této architektuře tedy knihovna) sám odesílá data, server je pouze pasivně přijímá, odtud název.

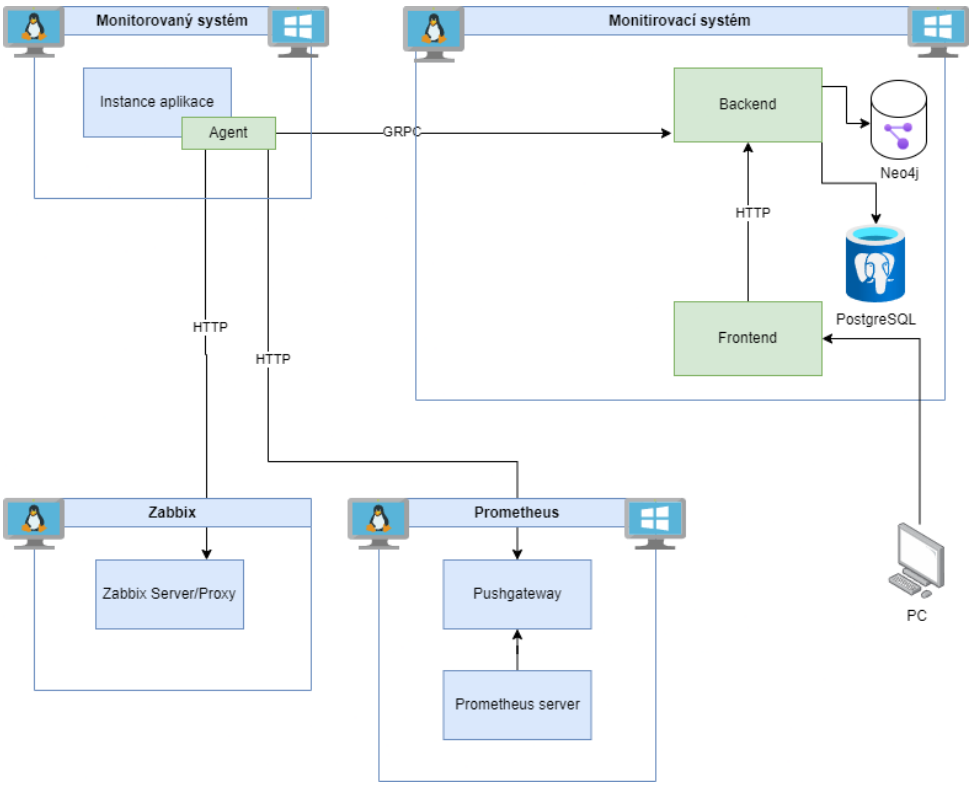
4.2 Agent

Agent je napsán v jazyce .NET a k integraci používá technologii Dependency Injection. Samotný agent bude distribuován ve formě knihovny, která je začleněna do stávajícího řešení.

4.2.1 Architektura agenta

Agent se skládá ze tří komponent - sběrače dat, úložiště hodnot a odesílatele. Všechny třídy jsou propojeny prostřednictvím manažera, který řídí tok dat.

Data jsou rozdělena do metrik - kolekcí dat (Value) rozdělených podle typu dat (double, long, ulong a string), typu agregace (all, change, min, max,



Obrázek 4.1: Architektura řešení

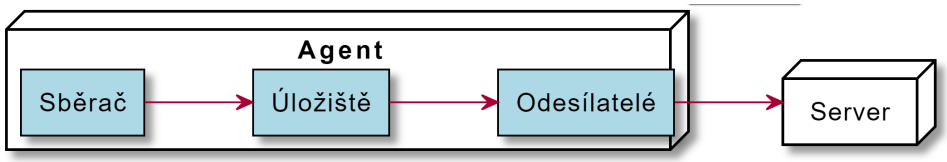
average) a názvu.

Sběrač

Sběrač je rozhraní pro předávání nových hodnot k odeslání. Stačí zadat název metriky, hodnotu a agregaci, s níž bude tato hodnota uložena. Sběrač pak předá hodnotu úložišti prostřednictvím manažera.

Úložiště

Úložiště obsahuje data a metadata o metrikách, typu agregace a typu samotných dat. Agregační třídy jsou také zodpovědné za způsob agregace, tj. počítání agregace probíhá přímo při sběru dat. Úložiště je také schopno resetovat odeslaná data, aby nedošlo k přeplnění paměti zařízení. Před každým odesláním úložiště shromáždí všechna data a odešle je odesílatelům.



Obrázek 4.2: Architektura agenta

Odesílatelé

Odesílatelé odesílají v každém určitém intervalu data získaná z úložiště. Odesílají je v závislosti na konfiguraci. Možnými odesílateli jsou backend, Zabbix a Prometheus. Pokud se odesílání nezdaří, data se přesto obnoví.

Pro Zabbix používáme trapper itemy. Velkou nevýhodou je, že na Zabbyxu již musí existovat itemy se správně pojmenovaným klíčem. Agent neumí vytvářet položky vzdáleně, takže vyžaduje konfiguraci jak na straně agenta, tak na straně serveru Zabbix.

Prometheus vyžaduje Pushgateway, aby aktivně shromažďoval odeslaná data. Další nevýhodou je, že není k dispozici podpora řetězcových dat, takže se přirozeně neodesílají do Pushgateway.

Způsob odesílání dat do backendu popíšeme později v textu.

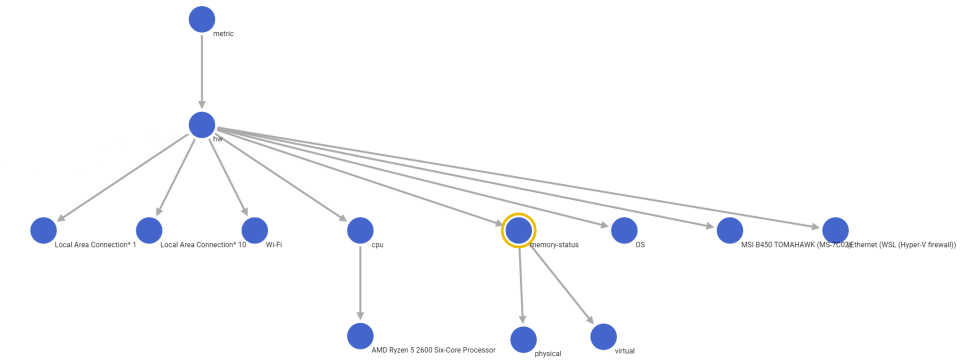
4.2.2 Data

V hodnotě je uložen její úplný název, čas vytvoření hodnoty, čas odeslání hodnoty a počet hodnot, které byly použity při agregaci k jejímu vytvoření.

Bylo rozhodnuto použít strom jako datovou strukturu metrik. Reprezentace dat ve stromové struktuře přináší několik zásadních výhod, zejména v oblastech organizace, vyhledávání a efektivity správy dat. Stromová struktura umožňuje hierarchické uspořádání dat, které je intuitivní a přirozeně odráží vztahy mezi jednotlivými prvky.

Data uložená ve stromové struktuře jsou rozdělena do uzlů, které formují hierarchii od kořenového uzlu až po listové uzly. Toto uspořádání umožňuje uživatelům snadněji navigovat v datové struktuře a rychleji nalézt požadované informace.

Pokud jde o názvy dat - ty určují pozici ve stromu hodnot. Strom hodnot je struktura, která je uložena na serveru. Uzel v tomto stromu je součástí názvu dat. Například jste nakonfigurovali, že každá položka má na začátku názvu připojený název aplikace, název serveru a/nebo mikroslužby (například ExampleApp1, Server1, LoginService). Pak jste přidali položku CPU1, kterou chcete uložit na server. Úplný název položky by byl <kořenový název



Obrázek 4.3: Strom metrik

stromu (obvykle metric)>_PříkladApp1_Server1_LoginService_CPU1. S tímto názvem bude odeslána na server.

4.2.3 Agregace

Často chceme pouze zjistit agregaci určité metriky nebo získat informace o ní po změně její hodnoty. Proto bylo rozhodnuto provést agregaci na úrovni agenta. Agregace se na agentovi počítá okamžitě a posílá se pouze výsledek. V případě, že nechceme server zatěžovat počítáním agregací, můžeme na server poslat celou hodnotu a agregace pak aplikovat tam.

Ne každý datový typ podporuje všechny typy agregace. Tabulka na stránce 20 popisuje tento vztah. (* - u řetězců se minimum a maximum počítá na základě lexikografického uspořádání)

Pro příklad vezměme sestavený list celých čísel {1, 2, 3, 4, 5}. Při agregaci budou na server odeslána všechna čísla, ale při další agregaci je možné, že bude odeslána jedna hodnota s jiným počtem hodnot parametru (například hodnota typu int s agregací minimum: {hodnota: 1, počet: 5}). Údaje o typu agregace nejsou uloženy v samotných datech, ale v datech metriky. Promluvme si o každé agregaci zvlášť.

Type / Aggregation	Long	Ulong	Double	String
All	✓	✓	✓	✓
Change	✓	✓	✓	✓
Min	✓	✓	✓	✓*
Max	✓	✓	✓	✓*
Average	✓	✓	✓	x

Tabulka 4.1: Podpora typ/agregace

Agregace All

Pro všechny agregace se shromažďují všechna data, která byla odeslána prostřednictvím kolektoru. V tomto případě se neprovádějí žádné další výpočty. Data jsou jednoduše odeslána příjemcům v této podobě.

Agregace Change

Změna agregace odešle pouze poslední změněnou hodnotu. Počet hodnot ukládá počet změn, ale ne samotné změny. Metriky s tímto typem agregace se vyznačují zřídkavými změnami, například operační systém, na kterém je aplikace spuštěna.

Agregace Min

Agregace min je minimum shromážděné v době mezi podáními. Položka length ukládá počet hodnot, které byly použity k výpočtu minima.

Agregace Max

Stejně jako u minima, jen s maximem.

Agregace Average

Agregační průměr je podobný jako minimum a maximum s jednou výjimkou - metriky řetězcového typu nejsou podporovány.

4.2.4 Způsob odesílání dat

Každý z odesílatelů používá jiný protokol a způsob přenosu dat. U dvou odesílatelů integrace se k odesílání dat používá protokol HTTP.

Zabbix používá k odesílání zabbix_sender [9]. K odesílání používáme jeho implementaci v prostředí .NET.

Pro Prometheus budeme také používat knihovnu, která za nás bude shromažďovat data a odesílat je do Pushgateway.

Pro náš backend jsem se rozhodl použít jiný způsob odesílání než jen HTTP a JSON. Protože shromažďujeme a odesíláme značné množství dat, stojí za to věnovat pozornost i způsobu odesílání.

4.2.5 Porovnání způsobů odesílání dat pro backend

Pro odesílání dat mezi agentem a serverem je třeba vybrat protokol a formát. Pro tento účel porovnáme JSON [10], BSON [11], Avro [12] a Protobuf [13].

JSON

Výhody:

- Snadná čitelnost a široká podpora v mnoha programovacích jazycích.
- Ideální pro webové aplikace a jednoduché datové struktury.

Nevýhody:

- Nižší výkon a větší datová náročnost oproti binárním formátům.

BSON

Výhody:

- Rychlejší zápis a čtení, optimalizováno pro rychlou manipulaci s daty.

Nevýhody:

- Větší velikost dat ve srovnání s Protobuf nebo Avro.

Avro

Výhody:

- Efektivní v distribuovaných systémech jako Apache Hadoop.
- Podpora dynamických schémat, což umožňuje snadnou evoluci datových struktur.

Nevýhody:

- Vyžaduje schéma pro serializaci a deserializaci dat, což může komplikovat vývoj

Type	Method	Mean [us]	Error [us]	StdDev [us]	Gen0	Gen1	Gen2	Allocated [KB]
AvroApacheBenchmark	'Avro separate items serialization (Apache)'	2,501.7	17.41	16.28	382.8125	74.2188	74.2188	1712.54
AvroChrBenchmark	'Avro separate items serialization (Chr)'	1,374.8	6.27	5.87	146.4844	72.2656	72.2656	597.35
BsonBenchmark	'BSON separate items serialization'	2,543.3	50.75	47.47	539.0625	335.9375	332.0313	2035.82
JsonBenchmark	'JSON separate items serialization (Standard library)'	1,917.9	32.75	30.64	126.9531	74.2188	-	676.82
ProtobufBenchmark	'Protobuf single object serialization'	646.5	6.93	6.14	79.1016	79.1016	79.1016	390.15
AvroApacheBenchmark	'Avro single object serialization (Apache)'	2,830.6	54.09	64.39	382.8125	74.2188	74.2188	1713.06
AvroChrBenchmark	'Avro single object serialization (Chr)'	527.0	9.93	9.75	110.3516	73.2422	73.2422	542.94
BsonBenchmark	'BSON separate items deserialization'	3,608.0	72.12	142.36	542.9688	226.5625	-	2538.49
JsonBenchmark	'JSON separate items serialization (Newtonsoft library)'	3,079.6	31.70	28.10	519.5313	402.3438	-	2913.87
ProtobufBenchmark	'Protobuf single object deserialization'	766.0	7.14	6.68	67.3828	25.3906	-	326.81
AvroApacheBenchmark	'Avro separate items deserialization (Apache)'	2,888.7	22.00	20.58	335.9375	144.5313	-	1546.04
AvroChrBenchmark	'Avro separate items deserialization (Chr)'	1,175.7	23.35	33.48	93.7500	37.1094	-	428.77
BsonBenchmark	'BSON single object serialization'	2,568.5	47.70	42.29	523.4375	339.8438	332.0313	1978.4
JsonBenchmark	'JSON separate items deserialization (Standard library)'	2,313.4	15.15	12.65	66.4063	23.4375	-	326.69
ProtobufBenchmark	'Protobuf separate items serialization'	811.3	9.22	7.70	79.1016	79.1016	79.1016	375.26
AvroApacheBenchmark	'Avro single object deserialization (Apache)'	3,144.7	12.00	11.22	339.8438	128.9063	-	1547.13
AvroChrBenchmark	'Avro single object deserialization (Chr)'	310.4	6.20	15.68	69.3359	24.9023	-	335.28
BsonBenchmark	'BSON single object deserialization'	3,529.1	33.49	26.15	515.6250	210.9375	-	2445.25
JsonBenchmark	'JSON separate items deserialization (Newtonsoft library)'	5,200.3	63.87	59.74	687.5000	242.1875	-	3310.57
ProtobufBenchmark	'Protobuf separate items deserialization'	1,127.4	10.77	9.55	76.1719	31.2500	-	358.52
JsonBenchmark	'JSON single object serialization (Standard library)'	1,985.8	10.73	8.96	164.0625	164.0625	164.0625	647.95
JsonBenchmark	'JSON single object serialization (Newtonsoft library)'	3,054.6	55.71	52.11	199.2188	199.2188	199.2188	1716.24
JsonBenchmark	'JSON single object deserialization (Standard library)'	2,303.2	19.86	17.60	66.4063	23.4375	-	335.47
JsonBenchmark	'JSON single object deserialization (Newtonsoft library)'	4,584.8	20.14	15.72	164.0625	54.6875	-	790.06

Obrázek 4.4: Benchmark pro serializaci a deserializaci

Protobuf

Výhody:

- Vysoký výkon a minimalizace velikosti dat, vhodné pro systémy s nízkou latencí.
- Silné typování a schema evolution podporuje zpětnou kompatibilitu.

Nevýhody:

- Potřeba definovat .proto soubory a kompilace, což zvyšuje složitost vývoje

Závěr

Na základě rychlosti [14] deserializace a serializace (Obrázek 4.4), jakož i vynikající podpory pro .NET budeme používat Protobuf.

4.3 Backend

Backend je aplikace nainstalovaná na serveru. Skládá se ze dvou částí, přičemž jeho endpointy jsou vystaveny na internetu. Backend může přijímat hodnoty od agentů a vytvářet dashboardy. Backend komunikuje se dvěma databázemi - grafovou a SQL.

4.3.1 Architektura backendu

Backend je napsán pomocí ASP .NET a nabízí dva typy endpointů: GRPC a REST API. GRPC slouží k přijímání dat od agentů. Po přijetí hodnot odešle server prázdnou zprávu a začne zpracovávat požadavek - ukládá hodnoty a sestavuje strom metrik. (Př. obrázek 5.1) Rozhraní REST API je potřebné pro frontend.

GRPC a agenti

Koncový bod GRPC přijímá všechny metriky s hodnotami jako jeden velký dotaz. Poté dotazový model převede na databázové modely. Po zpracování backend zkontroluje, zda data metrik již byla uložena v databázi grafů, a poté uloží hodnoty v SQL.

REST API a frontend

Rozhraní REST API umožňuje:

- Vytvářet, mazat a upravovat dashboardy.
- Vytvářet, mazat a upravovat widgety dashboardu.
- Získávat hodnoty metrik
- Načítat strom metrik

Dotazy jsou zpracovávány prostřednictvím databáze Neo4j nebo PostgreSQL.

4.3.2 Grafová databáze

Pro uložení stromu metrik jsme se rozhodli použít grafovou databázi. Porovnáme existující grafové databáze a vybereme nejlepší možnost pro naše řešení.

Grafové databáze jsou typem databáze NoSQL určené k ukládání, správě a dotazování složitých sítí datových vztahů efektivněji než relační databáze. K reprezentaci a ukládání dat používají grafové struktury s uzly, hranami a vlastnostmi, přičemž hrany představují vztahy mezi uzly. Tato struktura umožňuje rychlé vyhledávání složitých hierarchických struktur, které se v relačních systémech obtížně modelují.

Klíčové vlastnosti grafových databází

Vztahy jako entity první třídy V grafových databázích jsou vztahy uloženy na úrovni jednotlivých záznamů, což umožňuje rychlé procházení složitých vztahů.

Pružnost V rámci relačních vztahů je možné využívat různé typy dat: Grafové databáze jsou bez schématu, což umožňuje flexibilně přidávat nové druhy vztahů, uzlů nebo vlastností bez narušení stávajících dat.

Výkon Jsou optimalizovány pro dotazování na složité vztahy v rozsáhlých sítích, což poskytuje výkonnostní výhody oproti relačním databázím při práci s propojenými daty.

Intuitivní modelování dat Grafový model je často sémantičtější a intuitivnější pro reprezentaci reálných scénářů, což vývojářům usnadňuje modelování vzájemné interakce entit.

Uvažujme konkrétní příklady.

Neo4j

Společnost Neo4J je jedním z průkopníků v oblasti grafových databází a díky své robustní a funkčně bohaté platformě si stále drží vedoucí postavení. Používá model grafů vlastností a poskytuje shodu s ACID, která zajišťuje spolehlivé zpracování transakcí. Podporuje výkonný dotazovací jazyk Cypher, který je speciálně navržen pro procházení grafů.

Cypher Query Language Intuitivní a výkonný, usnadňuje psaní složitých dotazů.

Škálovatelnost Nabízí možnosti průmyslového rozsahu, včetně clusterování a replikace.

Bohatý ekosystém Zahrnuje širokou škálu nástrojů a integrací pro vývoj, vizualizaci a analýzu.

Silná komunita a podpora Těží z rozsáhlé komunity a možností profesionální podpory.

Ostatní grafové databáze porovnáme s Neo4j, protože Neo4j je v současné době nejpopulárnější databází.[8] Neo4j je také součástí programu Databázové systémy 2, proto se na ni v tomto srovnání zaměříme.

OrientDB

OrientDB je všestranný systém správy databází NoSQL s otevřeným zdrojovým kódem, který vyniká svými možnostmi práce s více modely. Na rozdíl od tradičních databází, které se drží jednoho datového modelu, OrientDB zahrnuje několik modelů, včetně grafů, dokumentů, objektů a ukládání klíč/hodnota. To umožňuje vývojářům používat jediný databázový stroj pro správu různých typů dat, díky čemuž je vysoce flexibilní pro různé potřeby aplikací.

V porovnání s Neo4j:

Více modelů Na rozdíl od Neo4J, který je čistě grafovou databází, OrientDB podporuje více modelů včetně dokumentových a objektových modelů.

Flexibilita V případě, že se jedná o databázi, která je v současné době v provozu, je možné, že se v ní budou vyskytovat různé typy dat: Výhodou může být ve scénářích, kdy je potřeba grafová databáze vedle jiných databázových modelů.

Výkonnost Obecně platí, že Neo4J má lepší výkon v čistě graficky orientovaných úlohách díky optimalizovanému stroji a dotazovacímu jazyku Cypher. [6] [7]

ArangoDB

ArangoDB je vícemodelový databázový systém NoSQL, který integruje možnosti datových modelů graf, dokument a klíč-hodnota do jednotné, škálovatelné platformy. Tato flexibilita umožňuje vývojářům používat kombinaci datových modelů v rámci jediného dotazu a jediné instance databáze, díky čemuž je ArangoDB obzvláště univerzální pro správu různých typů dat.

V porovnání s Neo4j:

Více modelů ArangoDB podobně jako OrientDB podporuje datové modely dokument, klíč/hodnota a graf.

Dotazovací jazyk AQL (ArangoDB Query Language), který je všestranný, ale ve srovnání s jazykem Cypher může být pro dotazy zaměřené na grafy náročnější.

Škálovatelnost ArangoDB má dobrou podporu pro sharding a distribuované architektury, což může být v Neo4J složitější.

Amazon Neptune

V porovnání s Neo4j:

Spravovaná služba V porovnání se správou vlastní instalace Neo4J se Neptune jako plně spravovaná služba zbavuje velké části provozních nákladů.

Integrace Hluboká integrace se službami AWS, což z něj činí výhodnou volbu pro uživatele, kteří do AWS silně investují.

Kompatibilita Podporuje jazyky Gremlin i SPARQL, což poskytuje flexibilitu, ale vyžaduje, aby se uživatelé přizpůsobili.

Závěr

Navzdory některým výhodám jiných grafových databází zůstává Neo4J favoritem díky svým velmi užitečným nástrojům, snadné instalaci v Linuxu, výkonnosti [6] a knihovně .NET, kterou potřebujeme pro backend aplikace.

4.4 Frontend

Frontend bude vyvinut jako Single-Page aplikace. [16] Rozebereme si jednotlivé stránky a knihovny [17], které jsou pro ně potřeba. Frontend se skládá ze dvou částí - hierarchického grafu s daty a přehledu dashboardů.

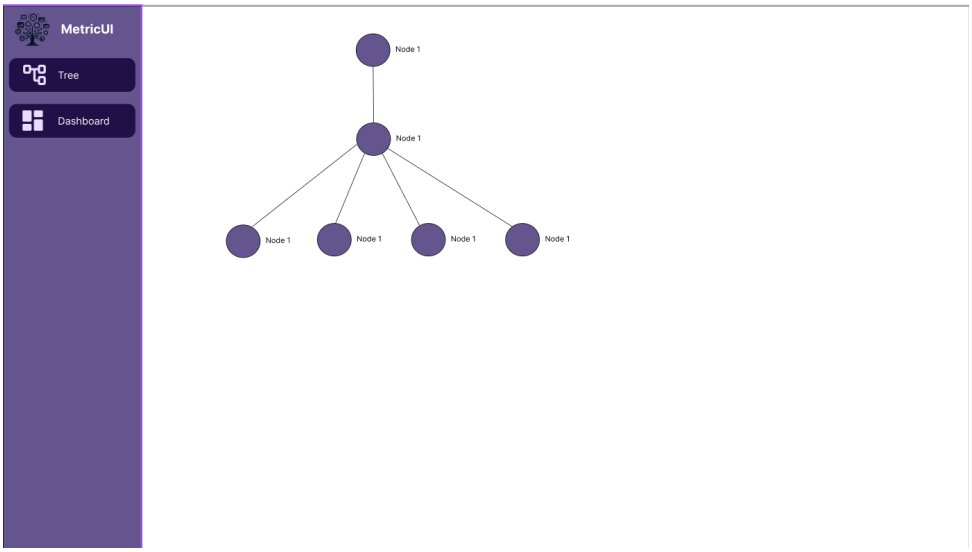
4.4.1 Grafy

Na základě architektury agenta a jeho dat budeme chtít v uživatelském rozhraní zobrazit celý strom metrik. S grafem můžete interagovat, otevírat uzly - potomky uzlů v grafu, zobrazovat data v tomto uzlu. Graf je strom, kde všechny ostatní uzly vycházejí z hlavního uzlu (by default - "metric"). Pokud jsou v uzlu data, zobrazí se vpravo od grafu. (př. Obrázek 4.5)

4.4.2 Dashboardy

Stránka dashboardů se zobrazí jako seznam vytvořených ovládacích panelů. Výběrem nebo vytvořením nového se otevře nová stránka s widgety a panelem pro filtrování dat. Můžete vytvářet nové widgety s existujícími metrikami, jejich typem a agregací.

K dispozici jsou tři typy widgetů.



Obrázek 4.5: Prototyp UI grafu

Widget chart. Widget chart zobrazuje hodnotu metriky ve formě grafu s časovou osou. Tento typ je typický pro číselné typy. Výběrem knihovny pro charty se budeme zabývat později.

Widget list. Widget list je typ widgetu, ve kterém jsou všechna data zobrazena ve formě tabulky. Tento typ se používá pro typ string.

Widget value. Widget value je typ widgetu, který zobrazuje jednu hodnotu. Může to být buď poslední hodnota metriky, nebo agregovaná hodnota.

Příklady použití agregací ve widgetech jsou uvedeny v tabulce na stránce 28-29

Agent aggregation / FE aggregation	All	Change	Min	Max	Average
All	Chart	Chart	Chart	Chart	Chart
Change	Value	Value	Value	Value	Value
Min	Value	Value	Value	Value	Value
Max	Value	Value	Value	Value	Value
Average	Value	Value	Value	Value	Value

Tabulka 4.2: Tabulka aplikace agent/FE agregace pro číselné typy

4.4.3 Prototyp

Prototyp byl vytvořen v programu Figma, později prošel uživatelskými testy a ve finálním řešení již byly doladěny chyby, které byly zohledněny během

FE aggregation \ Agent aggregation	All	Change	Min	Max	Average
All	List	List	List	List	N/A
Change	Value	Value	Value	Value	N/A

Tabulka 4.3: Tabulka aplikace agent/FE agregace pro typ string

testování. Testování se věnuje samostatná kapitola.

Celý prototyp je interaktivní a zabývá se prací se dvěma stránkami - Dashboard a Graph. Prototyp obsahuje následující prvky:

- Vytvoření Dashboardu

- Vytvoření, úprava a odstranění widgetu

- Zobrazení dat metriky uzlů ve stromu, zobrazení uzlů stromu

4.4.4 Výběr knihovny na zobrazení grafu

Grafy uzlů a hran (node-edge graph) jsou zásadním nástrojem pro vizualizaci a analýzu komplexních sítí a vztahů mezi datovými body. Ve Vue.js [15] existuje několik knihoven, které umožňují efektivní práci s grafy uzlů a hran. Tato rešerše se zaměřuje na zhodnocení tří populárních knihoven: *vue-d3-network*, *vue-flow* a *v-network-graph*. Po podrobném zhodnocení jednotlivých knihoven bude vybrána nejlepší volba pro použití v projektech založených na Vue.js.

vue-d3-network

vue-d3-network je knihovna, která kombinuje sílu D3.js s jednoduchostí Vue.js. D3.js je známá svou flexibilitou a možnostmi vytváření komplexních a interaktivních grafů. Knihovna *vue-d3-network* se zaměřuje na integraci těchto schopností do Vue.js komponenty.

- **Výhody**:
  - Vysoká flexibilita a přizpůsobitelnost díky D3.js.
  - Schopnost vytvářet velmi komplexní a interaktivní grafy.
- **Nevýhody**:
  - Složitost při použití pro začátečníky.
  - Potřeba hlubokého porozumění D3.js pro efektivní využití.

vue-flow

vue-flow je knihovna pro práci s grafy ve Vue.js, která nabízí moderní a intuitivní přístup k tvorbě a správě grafů. Je navržena tak, aby poskytovala jednoduché rozhraní pro tvorbu komplexních grafů s minimální námahou.

- **Výhody**:
  - Jednoduché a intuitivní rozhraní.
  - Snadná integrace do Vue.js projektů.
  - Podpora pro interaktivní grafy s možností přetahování uzlů a hran.
- **Nevýhody**:
  - Omezené možnosti přizpůsobení ve srovnání s D3.js.
  - Menší komunita a podpora než u některých jiných knihoven.

v-network-graph

v-network-graph je specializovaná knihovna pro tvorbu grafů uzlů a hran v Vue.js. Nabízí rovnováhu mezi snadným použitím a možnostmi přizpůsobení. Tato knihovna je navržena s ohledem na potřeby vývojářů pracujících s grafy v moderních webových aplikacích.

- **Výhody**:
  - Jednoduché a přehledné API pro tvorbu grafů.
  - Dobrá dokumentace a podpora.
  - Flexibilní možnosti přizpůsobení grafů a interaktivity.
  - Optimalizováno pro výkon ve Vue.js aplikacích.
- **Nevýhody**:
  - Menší komunita ve srovnání s D3.js, ale rychle rostoucí.

Závěr výběru knihovny na zobrazení grafu

Po zhodnocení všech výše uvedených knihoven je zřejmé, že *v-network-graph* nabízí nejlepší kombinaci jednoduchosti, přizpůsobitelnosti a výkonu pro moderní Vue.js aplikace. Zatímco *vue-d3-network* poskytuje obrovské možnosti díky D3.js, jeho složitost může být pro mnoho projektů překážkou. *Vue-flow*

je sice jednodušší a nabízí moderní přístup, ale postrádá některé pokročilé možnosti přizpůsobení, které jsou nezbytné pro složitější aplikace.

Proto **v-network-graph** představuje nejlepší volbu pro práci s grafy uzlů a hran v prostředí Vue.js, nabízející rovnováhu mezi jednoduchostí použití a pokročilými funkcemi potřebnými pro tvorbu efektivních a interaktivních grafů.

4.4.5 Výběr knihovny na zobrazení chartu

Charty jsou klíčovým nástrojem pro vizualizaci dat v moderních webových aplikacích. Pro Vue.js existuje několik knihoven, které umožňují efektivní tvorbu a správu grafů. Tato rešerše se zaměřuje na zhodnocení tří populárních knihoven: *Chart.js*, *ECharts* a *ApexCharts*. Po podrobném zhodnocení jednotlivých knihoven bude vybrána nejlepší volba pro použití v projektech založených na Vue.js.

Chart.js

Chart.js je jednoduchá a flexibilní knihovna pro tvorbu grafů, která je oblíbená díky své snadné použitelnosti a čistým vizualizacím. Integrace s Vue.js je možná pomocí různých obalových knihoven, například *vue-chartjs*.

- **Výhody**:
  - Jednoduché a intuitivní rozhraní.
  - Široká podpora různých typů grafů (čárové, sloupcové, koláčové atd.).
  - Dobrá dokumentace a velká komunita.
- **Nevýhody**:
  - Omezené možnosti přizpůsobení ve srovnání s pokročilejšími knihovnami.
  - Menší výkon při práci s velkým objemem dat.

ECharts

ECharts je výkonná knihovna pro vizualizaci dat, kterou vyvinula společnost Baidu. Nabízí širokou škálu grafů a vysokou úroveň přizpůsobení. Integrace s Vue.js je možná pomocí knihovny *vue-echarts*.

- **Výhody**:
  - Široká škála typů grafů a možností vizualizace.
  - Vysoká úroveň přizpůsobení a interaktivity.
  - Dobrá dokumentace a podpora pro velké objemy dat.
- **Nevýhody**:
  - Složitější nastavení a použití ve srovnání s jednodušími knihovnami.
  - Méně intuitivní pro začátečníky.

ApexCharts

ApexCharts je moderní knihovna pro tvorbu interaktivních grafů, která se zaměřuje na jednoduchost použití a vysokou přizpůsobitelnost. Integrace s Vue.js je možná pomocí knihovny *vue-apexcharts*.

- **Výhody**:
  - Jednoduché a intuitivní rozhraní.
  - Vysoká úroveň přizpůsobení a široká podpora různých typů grafů.
  - Dobrá dokumentace a aktivní komunita.
  - Optimalizováno pro výkon ve Vue.js aplikacích.
- **Nevýhody**:
  - Menší komunita ve srovnání s Chart.js, ale rychle rostoucí.

Závěr výběru knihovny na zobrazení chartu

Po zhodnocení všech výše uvedených knihoven je zřejmé, že *ApexCharts* nabízí nejlepší kombinaci jednoduchosti, přizpůsobitelnosti a výkonu pro moderní Vue.js aplikace. Zatímco *Chart.js* poskytuje snadnou použitelnost a čisté vizualizace, jeho omezené možnosti přizpůsobení a výkon mohou být pro některé projekty nevýhodou. *ECharts* je velmi výkonná a přizpůsobitelná knihovna, ale její složitost může být překážkou pro rychlé nasazení a použití.

Proto *ApexCharts* představuje nejlepší volbu pro práci s grafy v prostředí Vue.js, nabízející rovnováhu mezi jednoduchostí použití a pokročilými funkcemi potřebnými pro tvorbu efektivních a interaktivních grafů.

4.5 Integrace

Úloha vyžaduje integraci pro odesílání dat do Zabbixu a Promethea. Podívejme se na způsob odesílání dat.

4.5.1 Zabbix

Odesílání dat do Zabbixu - existují dva způsoby. Buď použít položky trapperu, nebo vytvořit koncový bod ze serveru pro aktivní monitorování serverem Zabbix. Protože chceme, aby všechna data spravoval agent, zvolíme první možnost.

Komunikace se Zabbixem probíhá prostřednictvím zabbix_sender. Pokud by neexistovaly žádné alternativy - museli bychom spolu s knihovnou uložit i executable pro zabbix_sender. Dobré je, že existuje knihovna Zabbix.Async, která umí posílat data ve stejném formátu jako zabbix_sender.

Bohužel to také vyžaduje přizpůsobení na straně zabbixu. Item a host již musejí existovat. Také musí být nakonfigurován správný typ v položce, jinak Zabbix vygeneruje chybu. Vytvoření itemu nebo hostu vzdáleně není možné.

Poznámka: Pro novou verzi Zabbix 7.0 LTS je přislíbeno přidání koncového bodu pro odesílání hodnot pro položku přímo přes API. V době psaní tohoto dokumentu a implementace kódu tato verze ještě nebyla vydána.

4.5.2 Prometheus

K aktivnímu odesílání dat do systému Prometheus je vyžadována služba Pushgateway. Tuto aplikaci lze nainstalovat vedle samotného serveru Prometheus. K dispozici je také knihovna pro .NET, která odesílá data získaná z agenta do Pushgateway. Také server Prometheus musí být nakonfigurován tak, aby shromažďoval informace z gatewaye.

4.6 Authorizace

Pro Backend a Frontend bylo rozhodnuto nepsat samostatný autorizační systém, ale použít samostatnou autentizační vrstvu. Kvůli tomu, že to není hlavní tema práce, rozhodli jsme se jit nejjednodušší cestou a použit Authelii, protože autor práce s ní má pozitivní zkušenosti.

Authelia je moderní open-source řešení pro autentizaci a autorizaci, které je navrženo pro použití v různých webových aplikacích a službách. Jeho hlavním

cílem je poskytovat bezpečný a jednotný přístupový bod pro uživatele, kteří se chtějí přihlásit k různým aplikacím nebo službám ve vaší infrastruktuře.

- **Autentizace a autorizace**: Authelia nabízí pokročilé možnosti autentizace, včetně dvoufaktorové autentizace (2FA), a detailní pravidla autorizace, která umožňují kontrolu přístupu na základě rolí a dalších kritérií.
- **Podpora různých protokolů**: Podporuje různé autentizační protokoly jako LDAP, SAML a OpenID Connect, což umožňuje snadnou integraci s různými existujícími systémy a službami.
- **Dvoufaktorová autentizace (2FA)**: Authelia umožňuje snadnou implementaci 2FA, což výrazně zvyšuje bezpečnost přístupu. Podporuje různé metody 2FA, včetně TOTP (Time-based One-Time Password), Duo a WebAuthn.
- **Přizpůsobitelnost**: Authelia je vysoce konfigurovatelná a přizpůsobitelná, což umožňuje administrátorům nastavit pravidla a politiky, které odpovídají specifickým bezpečnostním požadavkům jejich organizace.
- **Integrace s reverzními proxy**: Authelia je navržena tak, aby se snadno integrovala s reverzními proxy servery jako NGINX, Traefik a HAProxy, což umožňuje snadnou ochranu existujících webových aplikací.
- **Auditování a logging**: Nabízí možnosti pro podrobné auditování a logging, což umožňuje sledování přihlašovacích pokusů a dalších bezpečnostních událostí, což je klíčové pro zabezpečení a dodržování předpisů.
- **Škálovatelnost**: Je navržena tak, aby byla škálovatelná a mohla být nasazena v prostředí různých velikostí, od malých osobních projektů až po velké podnikové infrastruktury.

Kapitola 5

Implementace

Tato kapitola popisuje praktickou implementaci projektu, včetně vývojových nástrojů, které byly použity, a hlavních kroků realizace softwarového řešení.

5.1 Vývojové prostředí

Pro implementaci projektu byla vybrána dvě hlavní vývojová prostředí: Rider a WebStorm. Rider byl použit pro backendovou část aplikace, kde jeho schopnost efektivně pracovat s C# kódem a integrovat různé databázové systémy výrazně zjednodušila vývoj. Na druhé straně, WebStorm se osvědčil při tvorbě frontendové části díky své podpoře pro moderní JavaScriptové frameworky a nástroje, což umožnilo rychlý vývoj uživatelského rozhraní.

5.2 Agent

Agent byl implementován pro sběr a odesílání metrik ze vzdálených služeb na server s využitím .NET Core. Tento komponent je klíčový pro aktivní monitorování výkonnosti a stavu systémových zdrojů v reálném čase. Agent byl vyvíjen tak, aby byl lehký a méně náročný na systémové zdroje, což umožňuje jeho snadné nasazení na různých platformách.

K vytvoření sběrače použijeme factory:

```
1    _valueCollector = collectorFactory.GetValueCollector<
         InternalInfoCollectJob>("hw");
```

Na příkladu sběru dat systémových metrik (tuto funkci lze povolit během konfigurace agenta) můžete vidět různé způsoby sběru dat. (Listing 5.1)

Posílaní dat se dělá pomocí Quartz knihovny, která podle CRON posílá na server a do integrace data. Knihovna používá Dependency Injection pro nastavení knihovny. Extension metoda je využitá pro inicializaci.

```
1
2  namespace MetricAgent.Jobs;
3
4  /// <summary>
5  /// Internal info collector (HW values) as a job
6  /// Can be deactivated by configuration
7  /// </summary>
8  public class InternalInfoCollectJob : IJob
9  {
10     private readonly ValueCollector _valueCollector;
11     private readonly IHardwareInfo _hardwareInfo;
12     private readonly Computer _computer;
13
14     public InternalInfoCollectJob(ValueCollectorFactory
           collectorFactory, IHardwareInfo hardwareInfo)
15     {
16         _hardwareInfo = hardwareInfo;
17         _valueCollector = collectorFactory.GetValueCollector<
               InternalInfoCollectJob>("hw");
18
19         _computer = new Computer
20         {
21             IsCpuEnabled = true,
22             IsGpuEnabled = true,
23             IsMemoryEnabled = true,
24             IsMotherboardEnabled = true,
25             IsControllerEnabled = true,
26             IsNetworkEnabled = true,
27             IsStorageEnabled = true
28         };
29     }
30
31     public Task Execute(IJobExecutionContext context)
32     {
33         _hardwareInfo.RefreshAll();
34         _computer.Open();
35         _computer.Accept(new UpdateVisitor());
36
37         _valueCollector.AddStringValue("OS", _hardwareInfo.
               OperatingSystem.Name, AggregationType.Change);
38         _valueCollector.AddStringValue("OS_version", _hardwareInfo.
               OperatingSystem.VersionString,
39             AggregationType.Change);
40
41         _valueCollector.AddULongValue("memory-
               status_physical_available", _hardwareInfo.MemoryStatus.
               AvailablePhysical);
42         _valueCollector.AddULongValue("memory-status_virtual_available
               ", _hardwareInfo.MemoryStatus.AvailableVirtual);
43         _valueCollector.AddULongValue("memory-status_physical_total",
               _hardwareInfo.MemoryStatus.TotalPhysical);
44         _valueCollector.AddULongValue("memory-status_virtual_total",
               _hardwareInfo.MemoryStatus.TotalVirtual);
```

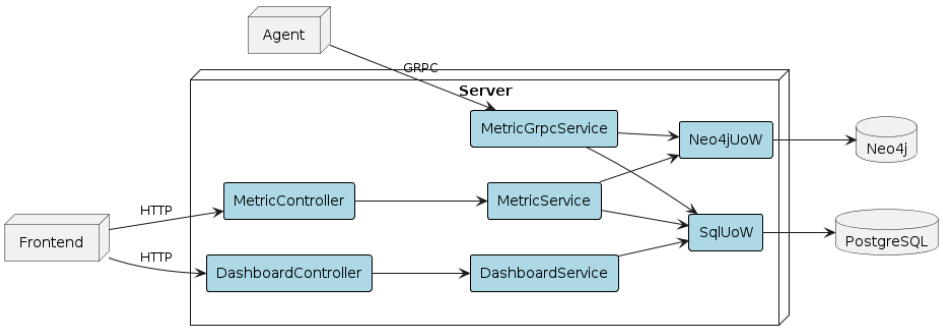
```
45
46          foreach (var cpu in _hardwareInfo.CpuList)
47          {
48              _valueCollector.AddLongValue($"cpu_{cpu.Name.Trim()}_#
                    cores", cpu.NumberOfCores, AggregationType.Change);
49              foreach (var cpuCore in cpu.CpuCoreList)
50              {
51                  _valueCollector.AddULongValue($"cpu_{cpu.Name.Trim()}_
                        {cpuCore.Name.Trim()}_Percent time",
52                      cpuCore.PercentProcessorTime);
53              }
54
55              _valueCollector.AddLongValue($"cpu_{cpu.Name.Trim()}_#
                    logical-cores", cpu.NumberOfLogicalProcessors,
56                  AggregationType.Change);
57          }
58
59
60          foreach (var hardware in _computer.Hardware)
61          {
62              _valueCollector.AddStringValue(hardware.Name.Trim(),
                    hardware.Identifier.ToString(), AggregationType.Change
                    );
63
64              foreach (var subHardware in hardware.SubHardware)
65              {
66                  _valueCollector.AddStringValue($"{hardware.Name.Trim()
                        }_{subHardware.Name.Trim()}", subHardware.
                        Identifier.ToString(),
67                      AggregationType.Change);
68
69                  foreach (var sensor in subHardware.Sensors)
70                  {
71                      _valueCollector.AddDoubleValue($"{hardware.Name.
                            Trim()}_{subHardware.Name.Trim()}_{sensor.Name
                            .Trim()}",
72                          sensor.Value ?? 0);
73                  }
74              }
75
76              foreach (var sensor in hardware.Sensors)
77              {
78                  _valueCollector.AddDoubleValue($"{hardware.Name.Trim()
                        }_{sensor.Name.Trim()}", sensor.Value ?? 0);
79              }
80          }
81
82          _computer.Close();
83
84          return Task.CompletedTask;
85      }
86  }
```

Listing 5.1: Sběrače systémových metrik

5.3 Backend

Vývoj backendu byl realizován s využitím .NET Core frameworku, což umožňuje efektivní správu závislostí a snadnou integraci s různými databázovými systémy, jako jsou PostgreSQL a Neo4j. Během implementace byly pravidelně psány unit testy, aby se zajistila funkčnost jednotlivých komponent bez potřeby spouštět celou aplikaci.

Dashboardy a metriky mají své vlastní kontroléry. Každá databáze má vlastní Unit of Work. Služba MetricGRPCService se také používá ke komunikaci s agenty. (Obrázek 5.1)



Obrázek 5.1: Backend implementace

5.4 Frontend

Frontend aplikace byla vytvořena s použitím Vue.js spolu s komponenty Vuetify, což je populární knihovna pro vytváření uživatelských rozhraní. WebStorm poskytl vynikající podporu pro tento framework, včetně ladění a testování. Důraz byl kladen na responzivní design, aby aplikace byla použitelná na různých zařízeních.

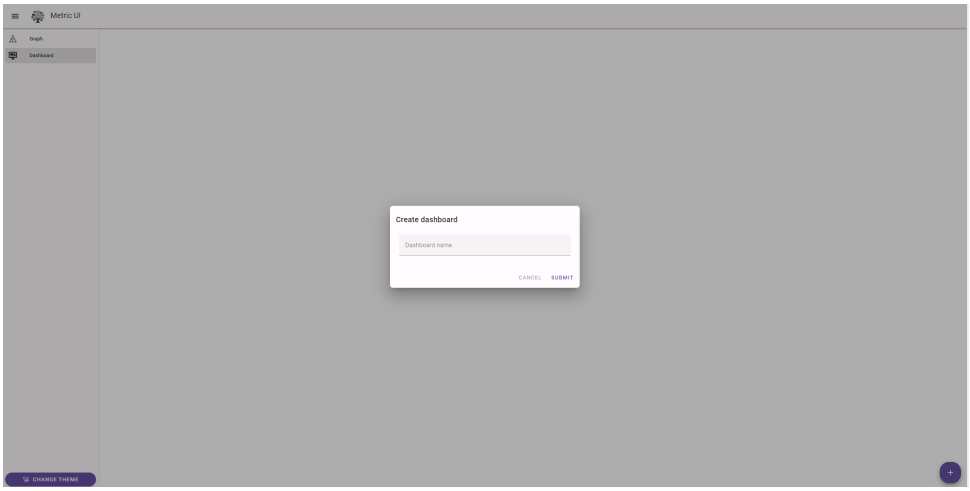
Celkem se jedná o 8 komponent, popíšeme si každou zvlášť.

DashboardComponent Komponenta, která obsahuje veškere widgety dashboardu spolu filtrovací lištou.

DashboardListComponent Komponenta, která obsahuje seznam dashboardů.

GraphComponent Komponenta s grafem metrik.

ChartComponent Komponenta, která přijímá data a podle něj je zobrazuje.



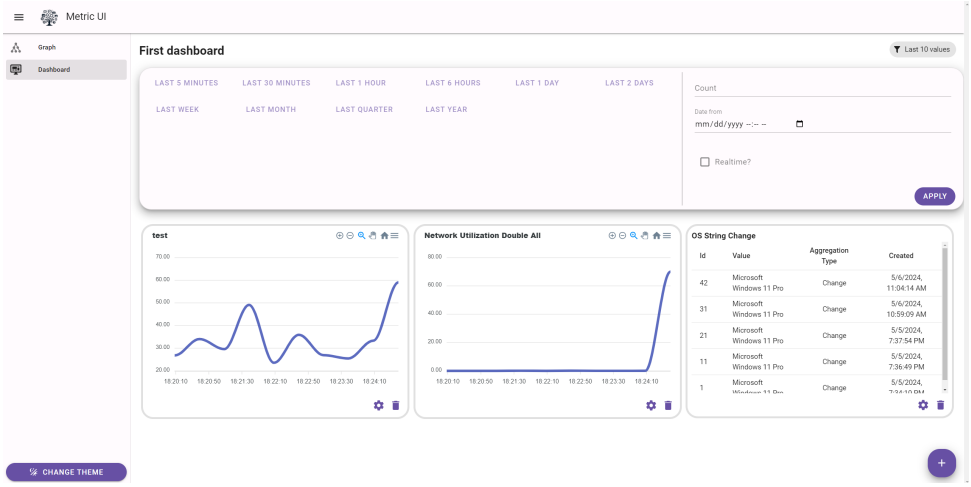
Obrázek 5.2: Vytvoření dashboardu

NavigationDrawer Komponenta s navigací po stránce.

ValueDrawer Komponenta, která pro určitý uzel v grafu zobrazuje všechny možné charty.

WidgetComponent Komponenta widgetu, natahuje data pro určitou metriku.

WidgetDialog Dialog pro vytvoření a úpravu widgetu.



Obrázek 5.3: Dashboardy FE

5.5 Závěr

Podařilo se uspěšně naimplementovat veškeré části systému. Přístup k vývoji byl iterativní, což umožnilo pružně reagovat na problémy a postupně zlepšovat funkcionalitu aplikace.

Kapitola 6

Nasazení

Nasazení aplikace probíhá na serveru s operačním systémem Ubuntu 22.04. Používáme webový server NGINX jako reverzní proxy pro přesměrování požadavků na příslušné endpointy a porty našeho REST API a GRPC serveru.

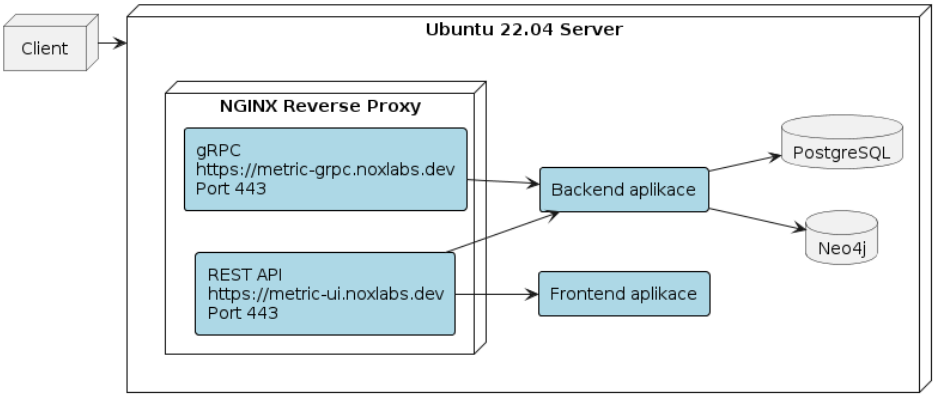
6.1 Konfigurace NGINX

NGINX [18] je nakonfigurován tak, aby poslouchal na dvou různých endpointech a portech:

- REST API je dostupné na `https://metric-ui.noxlabs.dev`, což je standardní HTTP endpoint pro uživatelské rozhraní a API volání. Tento endpoint naslouchá na portu 443 a zprostředkovává HTTP požadavky.
- gRPC služby jsou dostupné na `https://metric-grpc.noxlabs.dev`. Pro gRPC komunikaci je rovněž použit port 443, ale NGINX je speciálně nakonfigurován pro manipulaci s HTTP/2, což je nezbytné pro gRPC. SSL terminace probíhá na NGINX [19], v Kestrel je nastaven HTTP

6.2 Bezpečnost a certifikáty

Pro zajištění bezpečnosti přenosu dat jsou obě domény zabezpečeny pomocí SSL/TLS certifikátů. Certifikáty zajišťují šifrování komunikace a jsou pravidelně obnovovány pomocí Let's Encrypt.



Obrázek 6.1: Nasazení na serveru

6.3 Deployment aplikace

Frontend a Backend aplikace jsou oba nasazeny na stejném serveru. Backendová část běží jako služba na pozadí a je nastavena tak, aby se automaticky spustila po restartu serveru pomocí systemctl a service souboru. Frontend je hostován přímo NGINXem jako statické soubory, což umožňuje rychlé načítání uživatelského rozhraní.

6.4 Monitoring a logování

Pro monitorování stavu serveru a aplikací je nastaven systém pro sběr logů [3]. Tyto informace jsou klíčové pro rychlou identifikaci a řešení případných problémů v provozu.

Tento postup nasazení zajišťuje, že aplikace je bezpečná, výkonná a snadno škálovatelná.

Kapitola 7

Testování

V této kapitole jsou popsány metody a přístupy k testování jednotlivých komponent systému. Každá část systému - server, agent a frontend - je testována jinými metodami, aby bylo zajištěno, že všechny funkce pracují správně a spolehlivě.

7.1 Testování serveru

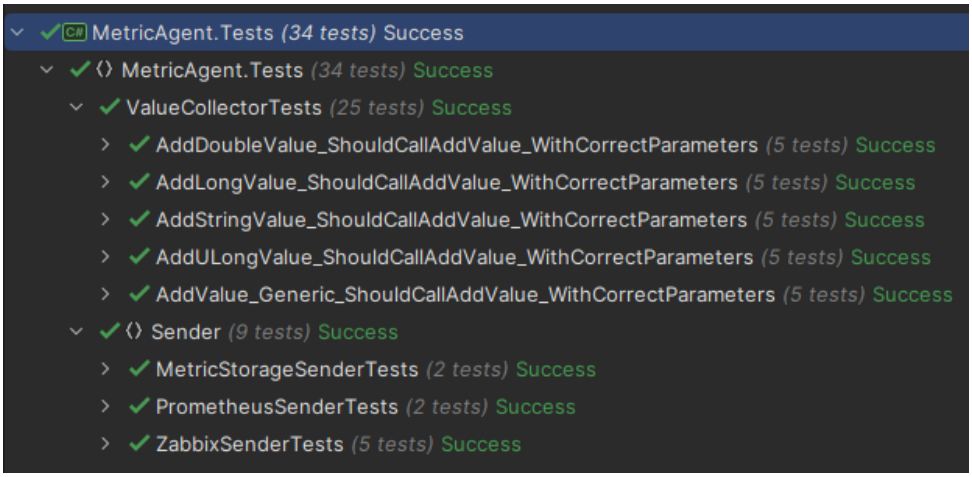
Pro testování serverové části aplikace jsou využívány unit testy. Tyto testy se zaměřují na ověření funkčnosti jednotlivých metod a tříd bez závislosti na externích systémech. Unit testy jsou implementovány s použitím frameworku, který je kompatibilní s vývojovým prostředím serveru, což je xUnit. [20]

7.2 Testování agenta

Agent je testován kombinací unit (Obrázek 7.1) a integračních testů. Unit testy se zaměřují na ověření funkčnosti jednotlivých komponent agenta. Integrační testy pak simulují celý proces sběru metrik, jejich zpracování a odeslání na server. Tímto způsobem je ověřena nejen správnost samotného agenta, ale i jeho schopnost spolupracovat s ostatními částmi systému.

7.3 Testování frontendu

Frontend aplikace je testován pomocí frameworku Cypress. Cypress umožňuje provádět komplexní end-to-end testy, které simulují skutečné uživatelské interakce v prohlížeči. Tento přístup zajišťuje, že uživatelské rozhraní je nejen



Obrázek 7.1: Výsledky testu na agentovi

funkční, ale také uživatelsky přívětivé a odpovídá požadavkům specifikovaným v designu.

7.4 Uživatelské testování

Uživatelské testování proběhlo dvakrát - po vytvoření prototypu a po nasazení aplikace na server. Proveďme analýzu každého z nich.

7.4.1 Testování prototypu

Testování prototypu bylo poměrně jednoduché. Uživatelům byl nabídnut prototyp ve Figmě. Byli požádáni, aby udělali několik věcí.

- Najít ve stromu metrik metriku s daty
- Vytvořit dashboard
- Vytvořit widget
- Odstranit widget

7.4.2 Testování aplikace

Uživatelské testování bylo provedeno s cílem identifikovat problémy v uživatelském rozhraní a interakci s aplikací. Tři uživatelé poskytli zpětnou vazbu na různé aspekty aplikace, které jsou shrnuty níže.

Uživatel 1

- Nezobrazuje se logo v levém horním rohu mezi menu a nápisem Metric UI.
- Problémy se škálováním velikosti grafů, nekonzistence ve zvětšování a zmenšování.
- Grafy na okrajích nejsou plně viditelné, části uzlů chybí.
- Při překliknutí mezi různými metrikami v grafu dochází k nechtěné změně velikosti grafu.

Uživatel 2

- Při zoomování grafu a kliknutí na uzel se graf automaticky zmenší, což vyžaduje opětovné přiblížení.
- Neexistuje možnost skrýt jednotlivé větve stromu grafu bez skrytí všech větví.
- Automatické odzoomování grafů po použití tlačítka pro přiblížení.
- Nefunkční dialog pro výběr metrik v "create widget", hodnoty se nezobrazují bez zahájení psaní.

Uživatel 3

- Tabulka s daty není dostatečně formátovaná a nevyužívá celou výšku okna.
- Možnost vytvořit dashboard bez názvu.
- Chybí funkce pro mazání dashboardu.
- Filtry dle času nemají možnost zobrazit všechna dostupná data nebo vybrat konkrétní časový interval.

Tato zpětná vazba ukazuje, že i přes funkčnost aplikace existují oblasti, které vyžadují další zlepšení k zajištění lepší uživatelské spokojenosti a plynulosti práce s aplikací. Zavážné chyby jsou opravené.

7.5 Závěr testování

Testování je nezbytnou součástí vývojového cyklu, která zajišťuje kvalitu a spolehlivost dodávaného softwaru. Díky pečlivě plánovaným a provedeným testům je možné včas identifikovat a opravit potenciální problémy, což vede k vyšší uživatelské spokojenosti a menšímu počtu chyb v produkčním prostředí.

Bylo zvoleno několik typů testování pro zachycení co největšího počtu chyb. Testy pokrývají kritické části systému.

Kapitola 8

Závěr

Tato diplomová práce se zaměřila na problematiku monitorování softwaru v prostředí mikroslužeb a lokálních sítí, což je klíčový aspekt pro zajištění hladkého provozu informačních systémů. Cílem bylo vytvořit aplikaci schopnou shromažďovat a vizualizovat data ze serverů, aplikací a mikroslužeb, a zároveň nabídnout flexibilní možnosti pro integraci a přizpůsobení sběru metrik.

8.1 Výsledky

V rámci projektu byla vyvinuta knihovna a serverová část v jazyce C# a frontend ve Vue.js. Toto řešení umožňuje efektivní monitorování, přičemž nabízí možnost mapování hierarchie sbíraných metrik, což přispívá k lepšímu porozumění a správě monitorovaných systémů. Díky integraci s nástroji jako Zabbix a Prometheus lze navíc snadno začlenit do stávajících monitorovacích řešení.

Práce se zaměřila na vývoj v prostředí ASP .NET s důrazem na použití LTS verzí jazyka C# a souvisejících knihoven, což zajišťuje stabilitu a dlouhodobou podporu výsledného řešení. Důležitým aspektem bylo také minimalizovat časové náklady na nastavení monitoringu, čehož bylo dosaženo efektivním návrhem způsobu odesílání a zpracování dat. Aplikace je aktuálně nasazená a funguje.

Tímto projektem bylo dosaženo nejen splnění stanovených cílů, ale také poskytnuto praktické řešení pro monitorování v prostředí mikroslužeb, které může být snadno integrováno a přizpůsobeno konkrétním potřebám vývojářů a správců systémů.

Vzhledem k provedené analýze existujících řešení přináší práce příspěvek v oblasti monitorování softwarových systémů, převážně pak mapování hierarchie metrik a agenta jako centrální prvek monitoringu, a může sloužit jako základ

pro další rozvoj a optimalizaci monitorovacích nástrojů. Jak bylo uvedeno v předchozích kapitolách, Prometheus postrádal typ string. Zabbix vyžaduje mnoho nastavení pro aktivní monitorování a žádná knihovna nepodporovala agregaci všech typů metrik, což konečné řešení podporuje a umí.

8.2 Možná rozšíření

Aplikaci lze rozšířit o skupiny uživatelů tak, že části stromu metrik jsou viditelné pouze pro jednotlivé uživatele. Podstromy metrik by byly k dispozici konkrétním skupinám, stejně jako různé dashboardy.

Je také možné přidat klíč API pro rozlišení agentů, kteří odesílají data na server před komunikací s GRPC. V současné době server nedokáže rozlišovat mezi jednotlivými agenty, což znemožňuje správu agentů na straně serveru.

Aplikaci lze rozšířit a přizpůsobit pro další monitorovací systémy. Představená koncepce a implementace umožňuje rozšířit řešení o další agregace a typy. Analyzovali jsme další monitorovací systémy, jako je Datadog a PRTG, a integrace s naším agentem nebo serverem by mohla být vyvinuta i pro ně.

Literatura

[1] Munzner, T. *Visualization Analysis and Design.* (AK Peters Visualization Series, CRC Press, 2014)

[2] Few, S. *Information Dashboard Design: Displaying Data for At-a-Glance Monitoring.* (Analytics Press, 2013)

[3] Schou, C. *Mastering .NET Core Logging - Best Practices & Examples.*(Online,2023), `https://christian-schou.dk/blog/mastering-logging-in-net-core/`.

[4] *Top Server Monitoring Tools for the New Year* (Online, 2024), `https://stackify.com/top-server-monitoring-tools/`.

[5] Liang Yonglin, G., *Monitoring system and method based on Zabbix and Docker.* (2017)

[6] Diogo A. B. Fernandes, *Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB.* (SCITEPRESS - Science, 2018)

[7] Martin Macak, *How well a multi-model database performs against its single-model variants: Benchmarking OrientDB with Neo4j and MongoDB.* (IEEE, 2020)

[8] Matea Pesic, *DB-Engines Ranking: Top Graph Databases You Should Use* (May 24, 2023) `https://memgraph.com/blog/db-engines-ranking-top-graph-databases`.

[9] Zabbix, *Sender* (2024), `https://www.zabbix.com/documentation/current/en/manual/concepts/sender`.

[10] JSON.org, *Introducing JSON*, `https://www.json.org/json-en.html`.

[11] BSON, *BSON Specification*, `https://bsonspec.org`.

[12] Apache Avro, *Apache Avro Documentation*, https://avro.apache.org/docs/current/.

[13] Google Developers, *Protocol Buffers*, https://developers.google.com/protocol-buffers.

[14] DataForGeeks, *Data Serialisation – Avro vs Protocol Buffers*, https://dataforgeeks.com/data-serialisation-avro-vs-protocol-buffers.

[15] Radixweb, *VueJS Best Practices to Create High-Performing Web Applications*, https://radixweb.com/blog/vuejs-best-practices.

[16] Raygun Blog, *A guide to single-page application performance*, https://raygun.com/blog/spa-performance/.

[17] Cloudinary Blog, *Front-End Development: The Complete Guide*, https://cloudinary.com/guides/front-end-development/front-end-development-the-complete-guide.

[18] Solo.io, *NGINX API gateway*, https://www.solo.io/topics/nginx/nginx-api-gateway/.

[19] F5, *10 Tips for Deploying NGINX as an API Gateway*, https://www.f5.com/resources/videos/10-tips-for-deploying-nginx-as-an-api-gateway.

[20] Gerard, Meszaros. *XUnit test patterns and smells: improving the ROI of test code.*, (2010).