Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Monitoring webových aplikací

Bc. Patrik Bezdíček

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Patrik Bezdíček

Informatika

Název práce

Monitoring webových aplikací

Název anglicky

Monitoring of web applications

Cíle práce

Diplomová práce je zaměřena na získání komplexního přehledu nad současnými technologiemi a doporučenými technikami pro sledování webových aplikací.
Hlavním cílem práce je vytvořit návrh vhodného řešení pro širokou škálu webových aplikací, které by bylo snadno nasaditelné, zajišťující nejběžnější potřeby a zároveň by bylo v budoucnu snadno rozšiřitelné.
Dílčí cíle jsou:
- získat teoretický přehled nad danou problematikou,
- katalogizovat a vybrat prvky aplikace, které jsou vhodné pro sledování,
- zmapovat, porovnat a vybrat technologie vhodné pro daný problém.

Metodika

Metodika pro Diplomovou práci je založena na rešerši a studiu dostupných teoretických poznatků dané problematiky. Pro řešený případ bude zvolena katalogizace a charakteristika požadavků na základě zkušeností z provozování aplikací v provozu. Pro volbu vhodných technologií bude použita srovnávací analýza. Syntézou teoretických poznatků, zjištěných požadavků a vybraných technologií bude formulován návrh řešení.

Doporučený rozsah práce

60–80 stran

Klíčová slova

Monitoring, web, application

Doporučené zdroje informací

Effective Monitoring and Alerting: For Web Operations: Slavek Lignus
Practical Monitoring: Effective Strategies for the Real World, Mike Julian 2018
Prometheus: Up & Running: Infrastructure and Application Performance Monitoring, Brian Brazil 2018

Předběžný termín obhajoby

2022/23 LS – PEF

Vedoucí práce

Ing. Michal Stočes, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 14. 7. 2022

doc. Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 28. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 30. 03. 2023

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Monitoring webových aplikací" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 31. 3. 2023 _____

Poděkování

Rád bych touto cestou poděkoval mému vedoucímu práce, panu Ing. Michalovi Stočesovi Ph.D., za pomoc a cenné rady k vypracování této práce. Dále bych chtěl poděkovat mé rodině za trpělivost a podporu, kterou mi poskytovala.

Monitoring webových aplikací

Abstrakt

Práce je zaměřena na problematiku monitoringu webových aplikací a služeb. V teoretické části jsou popsány problémy, které dnešní monitoring řeší. Je představena architektura a komponenty monitorovacího systému.

V praktické části je řešen stav monitoringu webových služeb ve společnosti. Je provedena analýza a jsou vyhodnoceny nedostatky. Následně je navržen a implementován nový monitorovací systém. Celý postup je popsán v jednotlivých krocích, a zaměřuje se na definici požadavků, architekturu, výběr komponent, implementaci měření a tvorbu dashboardů. Nový systém je otestován a vyhodnocen.

Klíčová slova: monitoring, web, application, Fluentd, Grafana, Kibana, Elasticsearch, TSDB, TypeScript

Monitoring of web applications

Abstract

This thesis focuses on the topic of monitoring web applications and services. The theoretical part describes the problems that modern monitoring addresses. The architecture and components of the monitoring system are introduced.

The practical part focuses on the state of monitoring web services. An analysis is conducted, and deficiencies are evaluated. Subsequently, a new monitoring system is proposed and implemented. The entire process is described in individual steps, which focus on defining requirements, architecture, selecting components, implementing measurements, and creating dashboards. The new system is tested and evaluated.

Keywords: monitoring, web, application, Fluentd, Grafana, Kibana, Elasticsearch, TSDB, TypeScript

Obsah

1 Úvod

Monitoring je specifická oblast v oblasti IT. Kvalitní monitoring by měl být součástí každého systému. Bohužel tento aspekt je často zanedbáván. V oblasti webových aplikací tak nezřídka dochází ke stavu, kdy je aplikace nebo služba provozována zcela bez tohoto důležitého prvku. Až teprve na produkci je tento aspekt řešen, často až po nepříjemných zkušenostech a finančních ztrátách.

Situace se však zlepšuje a dnešní svět IT nabízí mnoho řešení v oblasti nástrojů pro monitoring. Moderním trendem je takzvaný „Composable monitoring“, který se posunuje od zastaralých monolitických systémů, jako je Nagios, směrem ke konceptu volně propojených komponent. Dnes se tedy monitorovací systém skládá z různých komponent a na vývojáři je, jaké nástroje pro jednotlivé komponenty zvolí.

Vzniklo mnoho zajímavých projektů, ať už komerčních nebo open-source, s mnoha zajímavými koncepty a funkcionalitami. Vývojář tak dnes stojí před volbou vybrat si z nepřeberné nabídky. Může si vybrat hotové komerční řešení. Tato volba je jistě správná a je vhodná, pokud vývojář chce ušetřit čas a nechce investovat znalosti to této problematiky. Musí ale počítat s náklady a musí přijmout řešení dodavatele s omezenou možností specifických požadavků. Jindy se ale může investice do vytvoření vlastního monitorovacího systému hodit. V tom případě je nutné projít procesem výběru správných nástrojů, implementací a provozováním takového systému. Výhodou je ale větší svoboda při změnách požadavků v budoucnu.

Kvalitní monitoring by měl být nedílnou součástí každé aplikace. Je to jediný způsob, jak včas detekovat a řešit problémy nebo jim dokonce předcházet.

2 Cíl práce a metodika

2.1 Cíl práce

Diplomová práce je zaměřena na získání komplexního přehledu nad současnými technologiemi a doporučenými technikami pro sledování webových aplikací. Hlavním cílem práce je vytvořit návrh vhodného řešení pro širokou škálu webových aplikací, které by bylo snadno nasaditelné, zajišťující nejběžnější potřeby a zároveň by bylo v budoucnu snadno rozšiřitelné.

Dílčí cíle jsou

- Získat teoretický přehled nad danou problematikou.
- Katalogizovat a vybrat prvky aplikace, které jsou vhodné pro sledování.
- Zmapovat, porovnat a vybrat technologie vhodné pro daný problém.
- Implementace měření a dashboardů.
- Ověření řešení a vyhodnocení.

2.2 Metodika

Metodika pro diplomovou práci je založena na rešerši a studiu dostupných teoretických poznatků dané problematiky. Pro řešený případ bude zvolena katalogizace a charakteristika požadavků na základě zkušeností z provozování aplikací v provozu. Pro volbu vhodných technologií bude použita srovnávací analýza. Syntézou teoretických poznatků, zjištěných požadavků a vybraných technologií bude formulován návrh řešení. Pro výběr vhodných nástrojů bude použita Saatyho metoda párového porovnání.

3 Teoretická východiska

Tato práce se zabývá monitoringem webových aplikací ve společnosti. Úkolem této kapitoly je získání teoretického přehledu o této problematice. V teoretické části je proveden úvod do monitorovacích systémů, jsou vysvětleny základní pojmy a dnes na něj kladené požadavky. Byly uvedeny všechny typy monitoringů a preferovaná architektura. Byly popsány komponenty monitorovacích systémů, které jsou následně řešeny v praktické části.

3.1 Úvod do problematiky monitoringu

Monitoring

Monitoring lze popsat jako činnost spočívající v pozorování a kontrole chování a výstupů systému a jeho komponent v průběhu času (1). Podle jiné definice, je monitoring proces sledování stavu a toku dat v systému s cílem identifikovat chyby a pomoci při jejich následné eliminaci. Techniky používané při monitorování informačních systémů protínají oblasti zpracování v reálném čase, statistiky a analýzy dat. Soubor softwarových komponent používaných pro sběr dat, jejich zpracování a prezentaci se nazývá monitorovací systém (2).

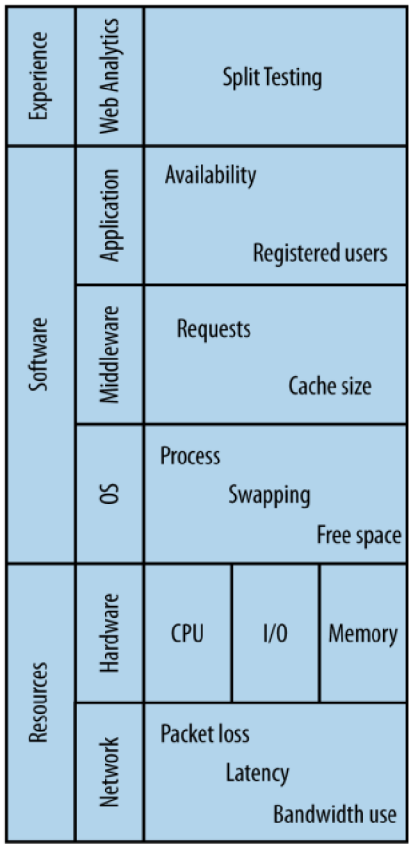
Dostupnost

V případě dostupnosti služby je výpadek obávaným stavem, který nastává v okamžiku, kdy systém přestane být plně funkční. Tento problém může být způsoben úplnou ztrátou dostupnosti, nebo omezenou dostupností pro určité uživatele. Klíčovým faktorem je včasná detekce a prevence v rušných výrobních prostředích, protože výpadky mohou vést k významným ztrátám v příjmech. Pro dosažení tohoto cíle je nezbytné mít kompletně nastavený monitoring, který umožní včasnou identifikaci problémů. Ideální monitorovací nástroje by měly umožnit operátorům přecházet z vysokoúrovňového přehledu k detailním úrovním, které poskytují důležitá specifika pro analýzu a identifikaci základní příčiny výpadku (2).

Pokrytí

Kompletní monitorování by mělo zahrnovat tři hlavní skupiny metrik: dostupnost zdrojů, výkonnost softwaru a v některých případech chování uživatelů. Pro všechny skupiny metrik by měla být k dispozici metrika časové řady, která umožňuje efektivní identifikaci

zdrojů problému. Kompletní monitorování pokrývá sítě, hardware, operační systémy, middleware, aplikace a klíčové ukazatele výkonnosti (2).



Obrázek 1: Pokrytí monitoringu (2)

Detekce problému

Hlavním cílem monitoringu je včas identifikovat a v krátkém čase určit zdroj problémů. Rychlá detekce potenciálně problémových situací je klíčovým cílem monitoringu, a je neodmyslitelnou součástí výstražného systému. Obtížnost spočívá v rovnováze mezi rychlostí a přesností detekce. Pokud nastal závažný problém, musí být o tom okamžitě a přesně informováno (2).

Není však užitečné alarmovat kvůli krátkodobým výpadkům nebo marginálním problémům, které nemají výrazný dopad. Někdy zůstanou chyby neodhaleny, jindy se alarmy znovu a znovu spouští navzdory tomu, že nehrozí žádné okamžité ani budoucí nebezpečí. Snížení výskytu obou druhů selhání je neustálý boj. Osoba sledující grafy proto musí učinit rozhodnutí založené na důkladném pochopení podkladových dat, způsobu jejich sběru a jejich zdrojů. Správná interpretace vyžaduje důkladné znalosti systému, které samotné monitorování neposkytuje (2).

Root cause

Termín root cause (prvotní příčina) bývá každým interpretován různě, což vede k četným poruchám v komunikaci. Analýzy prvotních příčin (RCA) jsou prováděny s cílem zjistit důvody, proč události mají škodlivé účinky v produkci. Hlavním cílem RCA je zjistit skutečný původ chyby, aby bylo možné provézt opatření a zabránit opakování (2).

Alerting

Alerting je schopnost monitorovacího systému detekovat a upozornit na významné události, které signalizují závažnou změnu stavu. Upozornění je označováno jako "alert" a může být předáno v různých formách, například jako e-mail, SMS, instant message (IM) nebo telefonní hovor. Alert je následně předán příslušnému příjemci, který je odpovědný za řešení události (2). Z hlediska důležitosti lze rozlišovat dva typy upozornění:

- **Výstražný alert**

  Toto upozornění by mělo vézt až k probuzení uživatele. Vyžaduje okamžitou akci, protože by mohl systém selhat. Taková upozornění mohou být v podobě telefonních hovorů, textových zpráv nebo alarmů. Například, pokud jsou všechny webové servery nedostupné.
- **Informativní alert**

  Tato upozornění jsou spíše informativní povahy a nemusí vyžadovat okamžitou akci, ale někdo by měl být informován o výskytu dané události. Například, pokud se nezdaří noční zálohování dat (1).

Analýza a reporting

U některých typů monitorovacích dat může být užitečné jít nad rámec jednoduché vizualizace a do sfér analytiky a reportingu. Jedním z nejčastějších případů použití je zde určení a reportování o servisní dostupnosti (SLA). SLA je dohoda mezi poskytovatelem a zákazníkem (interním i externím) ohledně očekávané dostupnosti aplikace/služby, obvykle určená měsíc po měsíci. V závislosti na dohodě mohou existovat smluvní pokuty za nesplnění SLA. SLA bez sankční doložky jsou obecně považovány spíše za „cíl k dosažení“ (1).

Solidní pochopení dat z grafů je nezbytné pro vyvození spolehlivých závěr. Vědět, do které kategorie metrika patří a odkud její data pocházejí, je klíčové při určování dopadu a prezentaci informací (2).

3.2 Typy monitoringu

Monitoring lze členit dle různých pohledů do několika skupin. Pro účely této práce jsou rozlišovány tyto typy monitoringu:

- Monitoring zdrojů
- Monitoring sítě
- Aplikační monitoring
- Monitoring databázových služeb
- Ostatní typy monitoringu

3.2.1 Monitoring zdrojů

Každý proces v systému spotřebovává čas procesoru, paměť případně zapisuje nebo čte z disku. V tomto oddíle jsou zmíněny systémové zdroje a způsob, jak je monitorovat.

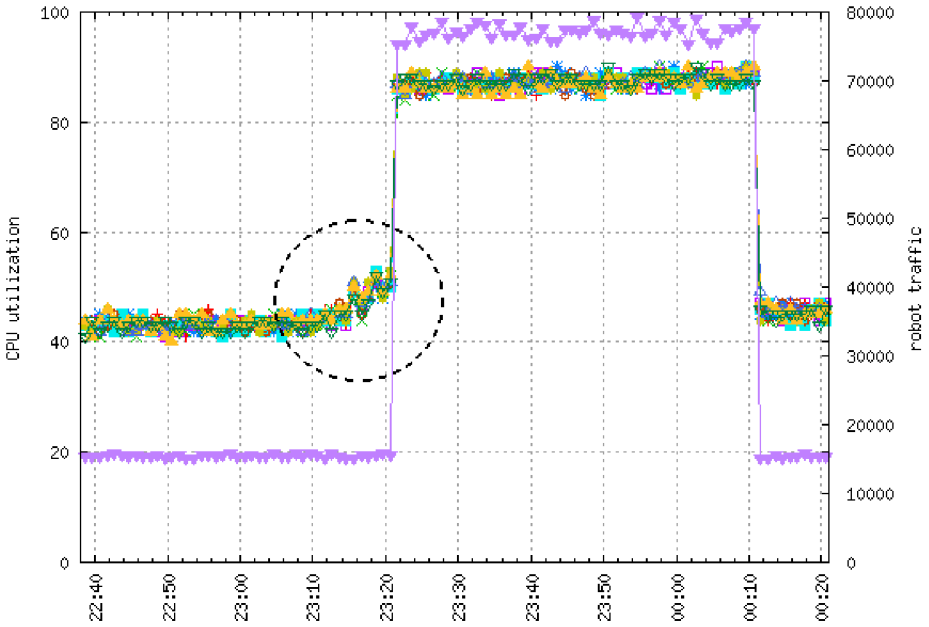
Metriky využití systémových zdrojů nejsou příliš vhodné pro alerting. Některé služby, jsou ze své podstaty náročné na zdroje a nemusí to být problém. Například pokud server MySQL využívá všechny CPU konzistentně, ale doby odezvy jsou přijatelné, pak ve skutečnosti je vše v pořádku. To samozřejmě neznamená, že metriky zdrojů nejsou užitečné. Metriky OS jsou kritické pro diagnostiku a analýzu výkonu, protože umožňují odhalit výkyvy a trendy v základním chování systému, které mohou mít vliv na výkon (1).

Využití CPU

Typickou výpočetní akcí v prostředí webových služeb je http request. Každý request vyžaduje zdroje: minimálně spotřebovává paměť a čas procesoru, ale také čte a zapisuje data do jiných zařízení, jako jsou disky, čímž zavádí další vstupně-výstupní náklady. Vyčerpání jakéhokoli zdroje potřebného k obsluze požadavku vede k vytvoření tzv. úzkého hrdla. Využití zdrojů nelze předvídat se 100% spolehlivostí a nedostatku zdrojů nemusí být vždy zabráněno přesným plánováním kapacity nebo dynamickým přidělováním instancí v cloudu (2). Sledování využití CPU je ze všech těchto metrik nejjednodušší. Metriky lze například získat v linuxových systémech z `/proc/stat`. Níže je zobrazen i graf využití cpu.

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          17,75    0,08    9,37    0,12    0,00   72,68
```

Obrázek 2: Výstup statistik využití CPU



Obrázek 3: Graf využití CPU (1)

Paměť

Monitorování využití paměti je důležitou součástí správy systému, jelikož při překročení limitů paměti mohou nastat problémy s výkonem a stabilitou aplikací. Zároveň může monitoring paměťového využití pomoci identifikovat aplikace nebo procesy, které spotřebovávají příliš mnoho paměti a způsobují tak zbytečné náklady na hardwarové zdroje. Většina nástrojů hlásí metriky paměti na základě hodnot hlášených `/proc/meminfo`.

```
             total       used       free     shared    buffers     cached
Mem:           488        426         62         31         75        222
-/+ buffers/cache:        127        360
Swap:            0          0          0
```

Obrázek 4: Výpis využití paměti v příkazové řádce

Disk

Aby nedošlo k zaplnění diskového úložiště, je nutné sledovat množství volného místa, zda se nepřibližuje svým limitům. Vyčerpání úložného prostoru může v vyvolat fatální chování (2). V linuxových systémech lze sledovat výkon disku ze zdroje: `/proc/diskstats`. Při použití iostat (dostupný v balíčku sysstat, spolu s mnoha dalšími nástroji) a s příznakem -x, je možno získat rozšířenou sadu metrik.

Device	tps	kB_read/s	kB_wrtn/s	kB_dscd/s	kB_read	kB_wrtn	kB_dscd
dm-0	30,79	102,73	234,44	133,62	511002963	1166165101	664654300
dm-1	30,78	102,73	237,23	133,62	510997735	1180048789	664654300
dm-2	0,00	0,00	0,00	0,00	4308	0	0
nvme0n1	18,67	102,74	234,45	134,57	511039555	1166213602	669389268

Obrázek 5: Výstup příkazu iostat

Load (zatížení)

Zatížení je měření, kolik procesů čeká na obsluhu CPU. Je reprezentováno třemi čísly: průměr za 1 minutu, průměr 5 minut a průměr 15 minut. V linuxových systémech je pro zjištění zatížení příkaz `uptime`, který stahuje data z `/proc/loadavg`.

```
12:01:41 up 57 days, 14:03,  1 user,  load average: 1,85, 1,92, 1,99
```

Obrázek 6: Výstup příkazu uptime

3.2.2 Monitoring sítě

Při přenosu dat po síti mohou nastat zpoždění a omezení propustnosti. Je důležité minimalizovat zpoždění (latenci) a maximalizovat propustnost sítě. Pokud dochází k poruchám, projevuje se to jako zvýšená latence a snížená propustnost. Počítačové sítě jsou klíčové pro distribuované systémy a jakékoli porušení sítě se projeví na celkovém výkonu systému. Bohužel aplikace jsou navrženy na předpokladu, že síť funguje bez problémů. To však není vždy samozřejmostí. Protože ztráta nebo zpoždění síťových paketů v síti může výrazně ovlivnit výkon, je nutné pečlivě sledovat stav sítě (2).

Na linuxových systémech lze informace získat z `/proc/net/dev`. Příkazy `ifconfig` i `ip`, z balíčku iprout. Z rozhraní lze získat následující metriky:

- **Šířka pásma (bandwidth)**
  Teoretické maximální množství informací, které lze najednou odeslat prostřednictvím připojení. Lze brát jako nezpracovanou schopnost síťového spojení, běžně vyjadřenou v bitech za sekundu, ale obvykle v megabitech za sekundu (Mbps) nebo v gigabitech za sekundu (Gbps).
- **Propustnost (throughput)**
  Pozorovaný výkon síťového spojení, také vyjádřený v bitech za sekundu. Vzhledem k režii protokolu a přenosu bude propustnost menší než šířka pásma spojení.

- **Latence**

  Doba, po kterou paket putuje přes síťové spojení. Je zde fyzické omezení latence vzhledem k tomu, jak rychle může elektřina (nebo světlo, v případě optických kabelů) putovat. Latence může mít velký vliv na uživatelský zážitek v některých aplikacích, které nejsou tolerantní k vysoké latenci. Jedním z oblíbených způsobů, jak sledovat latenci mezi dvěma body, je používat nástroje jako iperf2 nebo uzení k pravidelnému měření a hlášení latence (1).

3.2.3 Aplikační monitoring

Předmětem této práce je monitoring webových aplikací a proto tento typ monitoringu je pro práci stěžejní. Záměrem je sledování toho co aplikace, funkce či blok kódu, provádí. Je to způsob jak aplikace sděluje co se v ní děje. Aplikace samotné vykazují mnoho cenných informací o jejich výkonu, z nichž mnohé umožní přejít od reaktivního k proaktivnímu řešení problémů. Jednou z častých obav je, že přidávat metriky do aplikací je obtížné a časově náročné (1).

Současné webové aplikace se zásadně odlišují od webových aplikací z počátků internetu. Dnešní webové aplikace mají více podobu distribuovaných systémů. Díky rozmachu JavaScriptu se značná část funkcionalit přesouvá na klienta. K dispozici je široká škála nástrojů a úložišť. Kontejnerizace umožňuje snadnější způsob nasazování a škálování. Cloud a cloudové služby sami o sobě představují odlišný přístup k vývoji, který má mnohem více charakter kupování služeb. Webové aplikace mají dnes nejběžněji podobu distribuovaných systémů.

Distribuované systémy

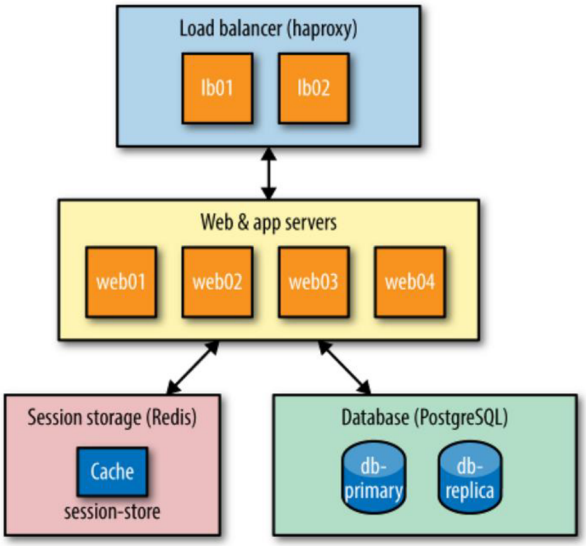
Distribuovaný systém se skládá z tzv. softwarových agentů, kteří musí spolupracovat, aby mohli provádět určité úkoly. Agenti v distribuovaném systému navíc nepracují ve stejném prostředí zpracování, takže musí komunikovat pomocí hardwarových nebo softwarových protokolů přes síť. To znamená, že komunikace s distribuovaným systémem je ze své podstaty méně rychlá a spolehlivá než komunikace s těmi, kteří používají přímé volání kódu a sdílenou paměť. To má důležité architektonické důsledky, protože distribuované systémy vyžadují, aby vývojáři infrastruktur a aplikací zvážili

nepředvídatelnou latenci vzdáleného přístupu a vzali v úvahu otázky souběžnosti a možnosti částečného selhání (3).

Distribuovaný systém se obvykle skládá ze stovek, ne-li tisíců logických součástí vykonávajících určitý typ práce. Současné systémy se mění podle poptávky, takže některé části mohou být nestálejší než jiné. Kromě toho jsou výstražné systémy na podnikové úrovni mimořádně bohaté na funkce, takže naučit se je efektivně obsluhovat vyžaduje trochu zkušeností a trochu pozornosti (2).

Monolitické aplikace

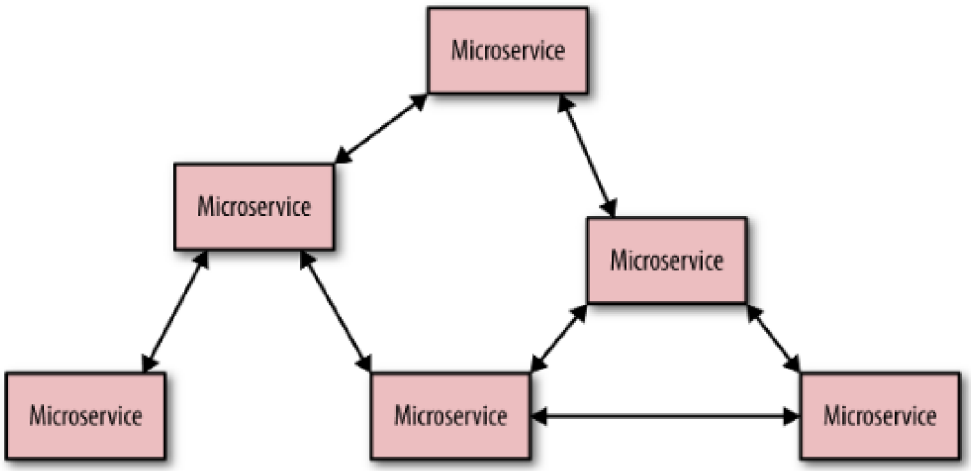
V monolitické architektuře se různé funkční komponenty aplikace na straně serveru, jako je zpracování plateb, správa účtů, push notifikace a další komponenty, prolínají v jeden celek. Aplikace jsou obvykle rozděleny do tří částí. Části jsou HTML stránky nebo nativní uživatelské rozhraní, které běží na počítači uživatele, aplikace na straně serveru, která běží na serveru, a databáze, která běží také na serveru. Aplikace na straně serveru je zodpovědná za zpracování http požadavků, načítání a ukládání dat v databázi, provádění algoritmů a tak dále. Pokud je aplikace na straně serveru jeden spustitelný soubor (tedy běžící je jeden proces), který všechny tyto úlohy provádí, pak říkáme, že aplikace na straně serveru je monolitická. To je běžný způsob vytváření aplikací na straně serveru. Téměř všechny hlavní CMS, webové servery a serverové frameworky jsou postaveny pomocí monolitické architektury. Tato architektura se může zdát úspěšná, ale problémy vznikají, když začne být systém robustní (4).



Obrázek 7: Jednoduchá monolitická architektura (1)

Microservices

Nedostatky monolitických systémů lze řešit rozdělením aplikace do tzv. microservices. Co se dá označit za microservice není jednoznačné. Microservice nemusí být ta, která má málo řádků kódu nebo poskytuje jen základní funkcionalitu. Microservice může být například ta, na které může pracovat nezávislý tým vývojářů, může být škálována nezávisle na jiných službách a je oddělena od jiných služeb (4).



Obrázek 8: Jednoduchá monolitická architektura (1)

Škálování výkonu

Navyšování výkonu webových aplikací se provádí technikou horizontálního škálování. Pro rozložení provozu mezi servery je potřeba použít něco, čemu se říká load balancer. Load balancer je server, který je umístěn před aplikačními servery. Klient komunikuje s load balancerem místo s aplikačními servery a místo vyřízení požadavku jej load balancer předá aplikačnímu serveru (4).

Webserver monitoring

Sledování výkonu webových serverů je jednou z nejkritičtějších součástí monitoringu. Pokud jde o webové servery, existuje jedna hlavní metrika pro hodnocení výkonu a úrovně provozu: requesty za sekundu (req/sec). Req/sec je v zásadě měření propustnosti (1).

Méně kritické pro výkon, ale stále důležité pro celkovou viditelnost, je sledování kódů HTTP odpovědí. Protokol HTTP má mnoho různých možných odpovědí na request. Nejčastější je 200 OK, ale existují i další běžné, například 404 Not Found, 500 Internal Server Error a 503 Service Unavailable.

Kódová skupina	Význam	Nejpoužívanější kódy
1xx	Informational	100 Continue
2xx	Success	200 OK, 204 No Content
3xx	Redirection	301 Moved Permanently, 302 Found
4xx	Client errors	400 Bad Request, 401 Unauthorized, 404 Not Found
5xx	Server errors	500 Internal Server Error, 503 Service Unavailable

Tabulka 1: Přehled skupin stavových kódů protokolu http (1)

Log parsery extrahují specifické informace z protokolových záznamů, jako jsou stavové kódy a doby odezvy požadavků z protokolování webového serveru (2). Zajímavá je i sezónnost statistik. V metrikách vysledovat pravidelně se opakující vzor a tuto znalost je pak možno využívat k plánování a předpovídání budoucnosti (4).

Obrázek 9: Graf requestů webového serveru (1)

3.2.4 Monitoring databázových služeb

Aplikace zpravidla ukládají a načítají data v některém ze systémů pro práci s daty. V dnešní době je možné sledovat velké spektrum datových úložišť podle zaměření. Zde je neúplný výčet typů úložišť dle serveru db-engines.com (5):

- Relational DBMS
- Key-value stores
- Document stores
- Time Series DBMS
- Graph DBMS
- Search engines
- Object oriented DBMS
- Multivalue DBMS

Monitoring databází

V databázových systémech jsou sledovány zejména následující metriky:

- **Počet spojení (Connections)**
  Počet spojení je nutné sledovat obzvláště v případech, kdy hrozí vyčerpání limitu počtu spojení do databáze. Např. MySQL pro každý request vytváří nové spojení.
- **Dotazy za sekundu (Queries per second)**
  Měření dotazů za sekundu je více vypovídající měření vytíženosti serveru. Měření qps koreluje se skutečnou vytížeností aplikace a je dobrým indikátorem toho, jak přesně jsou jsou databázové servery vytížené.
- **Pomalé dotazy (Slow queries)**
  Pomalý dotaz se často projeví jako pomalý uživatelský zážitek, což rozhodně nechceme. Prvním krokem k opravě pomalých dotazů je jejich nalezení. Pomalé dotazy jsou zaznamenány do souboru protokolu s časem vzniku, počtem a dotazem. Existuje mnoho nástrojů, které usnadňují analýzu těchto informací.
- **Zpoždění replikace (Replication)**
  Pokud provozujeme databázovou infrastrukturu většího rozsahu, je běžné používat repliky (dříve známé jako slaves), takže sledování zpoždění replikace je důležité (1).

Cache monitoring

Primární metriky cache jsou počet vypuzených položek z paměti a poměr hit/miss (někdy nazývaný poměr cache-hit). Jak cache roste, starší položky jsou z cache odstraňovány – tedy vypuzeny. Vysoká míra vypuzení je signálem pro to, že je cache příliš malá, což způsobuje, že je vypuzeno příliš mnoho položek, aby se uvolnilo místo pro nové položky (1).

Monitoring front zpráv

Fronta zpráv (message queue) se skládá ze dvou částí: publisher a subscriber (fronty zpráv se kvůli tomu někdy nazývají pub-sub systems). Sledování fronty se týká především dvou metrik: velikostí fronty a mírou spotřeby. Velikost fronty se rozumí počet zpráv ve frontě, které čekají na stažení jedním nebo více předplatiteli. Běžná velikost fronty závisí na tom, jak aplikace funguje. Míra spotřeby je rychlost, jakou jsou zprávy stahovány z fronty.

Tato metrika je obvykle vyjádřena v počtu zpráv za sekundu. Stejně jako u délky fronty závisí běžná míra spotřeby na tom, jak aplikace funguje (1).

3.2.5 Ostatní typy monitoringu

Existují další typy monitoringu. Přestože se tato práce jimi detailně nezaobírá, pro úplnost jsou zmíněny.

Business monitoring

Klíčový ukazatel výkonnosti (KPI) je metrika, která měří, jak si společnost vede podle zásad, které společnost považovala za důležité. KPI řekne jak si firma vede (1).

Frontend monitoring

Podle studie z roku 2010 provedené společností Aberdeen Research, každá jednosekundová prodleva v načítání webové stránky v průměru vede ke ztrátě 11% zobrazení stránky, 7% konverzí a 16% spokojenosti zákazníků. Podle Aberdeen je společnost ohrožena, pokud doba načítání stránky přesáhne 5 sekund, zatímco optimální doba načítání by měla být pod 2 sekundy (1).

Security monitoring

Bezpečnostní monitoring je náročný a specializovaný obor sám pro sebe. Jednou z problematik je sledování DDOS útoků (1).

3.3 Architektura monitorovacího systému

V roce 2011 se na Twitteru rozproudily konverzace kolem hashtagu #monitoringsucks, které se zabývaly otázkou, proč je monitorování tak špatné. Tyto diskuze nakonec vedly ke vzniku hashtagu #monitoringlove a k založení konference Monitorama v Bostonu. Během této události se vedlo mnoho rozhovorů o tom, co by se dalo udělat pro zlepšení věcí. Jedním z nejvýznamnějších bodů bylo, že potřebujeme nové a lepší nástroje, které budou specializovanější. Zrodil se tak složitý monitoring jako nápad, který se v praxi stal de facto standardem (1).

Composable monitoring

Složitelný monitoring je prvním příkladem moderního návrhu monitorování. Princip je jednoduchý a spočívá v použití několika specializovaných nástrojů a spojení volně dohromady, čímž vznikne monitorovací platforma. Tento vzor stojí v přímém kontrastu s monolitickými nástroji, jako například rozšířený Nagios (1).

Komponenty monitorovacího systému

Monitorovací platforma se skládá z volně propojených specializovaných komponent, které lze rozdělit podle těchto hlavních aspektů:

- Sběr dat
- Datové úložiště
- Vizualizace

Doplňkově jsou zmiňovány další aspekty jako analýzy a výkaznictví nebo alerting. V následujících oddílech jsou popsány jednotlivé komponenty systému (1).

3.3.1 Sběr dat

Existuje mnoho způsobů, jak systémy ukládají data na výstup. Systémy mohou rozdělit své záznamy do různých úrovní (například ladění, info, varování a chyba) s tím, že každá úroveň je zapisována do jiného souboru (6). Mnoho logovacích služeb umožňuje zápis z aplikace přes síť na určité místo, což usnadňuje odesílání k uložení a analýze. S nárůstem provozu aplikace by se však toto mohlo stát úzkým hrdlem. Při každém odeslání záznamu je nutné vytvořit síťové spojení, což klade nároky na síťové zdroje. V takovém případě je lepší záznam zapsat do souboru na disk. Řešením pak je služba, která přichází v pravidelných intervalech a odesílá záznamy záznamu na externí místo. To umožňuje odesílání záznamů provádět asynchronně z aplikace a může ušetřit mnoho zdrojů (2).

Protokoly

Protokoly jsou textové řetězce, které mají přiřazené časové razítko označující, kdy se daná událost stala. Protokoly obsahují mnohem více dat než metriky a často vyžadují analýzu, aby bylo možné získat informace zaznamenané v nich, aniž by bylo nutné je číst ručně. Protokoly se dělí na dva typy: nestrukturované a strukturované (1).

- **Nestrukturované protokoly**

  Jedná se o textový řádek, kde význam jednotlivých částí je dán pořadím výskytu v textu. Malá ukázka nestrukturovaného záznamu služby Nagios:

```
192.34.63.77 - - [26/Jun/2016:14:06:22 -0400] "GET / HTTP/1.1" 301 184 "-"
"Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit (537.36) Gecko)
Chrome/47.0.2526.111 (StatusCake)" "-"
```

- **Strukturovaný záznam**

  Zpravidla ve formátu JSON je posílán přes síť a ukládán do speciálního úložiště. Výhodou strukturovaných záznamů je možnost přesnějšího vyhledávání a agregace. Následně je vypsán ten samý záznam ve strukturované podobě:

```
{
  "remote_addr": "192.34.63.77",
  "remote_user": "-",
  "time": "2016-06-26T14:06:22-04:00",
  "request": "GET / HTTP/1.1",
  "status": "301",
  "body_bytes_sent": "184",
  "http_referrer": "-",
  "http_user_agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/
537.36 (KHTML, jako Gecko) Chrome /47.0.2526.111 (StatusCake)",
  "http_x_forwarded_for": "-"
}
```

Režie protokolování

Agenti sběru dat jsou procesy, kteří spotřebovávají malou část zdrojů sledované entity. Tomuto jevu se říká monitorovací režie, což je malá cena za práci při sběru dat. Agenti mohou vytvořit tzv. efekt pozorovatele, pokud změní stav sledovaného objektu nebo pokud sběr dat zesílí nebo zeslábne na základě výsledku měření. Běžně je ale tento jev zanedbatelný (2). Není zcela doporučováno protokolovat cokoliv. Protokolování každé maličkosti může vézt k zahlcení sítě nebo disku, a potenciálně může vézt k vážnému a zbytečnému problému v aplikaci (1).

Modely sběru dat

Sběr dat lze rozlišovat z pohledu aktivity na aktivní a pasivní (2), nebo alternativně s jiným názvoslovím na push a pull (1).

- **Pasivní model (pull)**

V pasivním modelu agent sleduje tok dat a shromažďuje statistiky, aniž by zvyšoval náklady na zdroje. V monitorovacích sítích jsou data shromažďována čtením statistik ze síťového rozhraní pomocí paketových snifferů (2). Pull model je implementován v podobě endpointu `/health`, který umožňuje získat metriky a informace o stavu aplikace pravidelným dotazováním monitorovací službou. Mechanismus založený na stahování dat má ale i nevýhody: může být obtížně škálovatelný, protože centrální monitorovací systémy musí mít přehled o všech známých klienty a jejich umístěních (1).

- **Aktivní model (push)**

V push modelu monitorovaný subjekt proaktivně odesílá protokoly o svém stavu na jiné místo. Může jednat podle plánovaného harmonogramu nebo na základě událostí (2). V distribuovaných systémech, jako jsou ty v cloudových prostředích, je push model snazší škálovat, nežli pull model. To je způsobeno tím, že v push modelu není potřeba centrálního dotazovače, stále je nutné udržovat hlavní seznam všech uzlů, které se mají dotazovat. Uzly, které posílají data, potřebují pouze vědět, kam je mají odeslat (1).

Kategorizace agentů sběru dat

Agenti sběru dat lze rozdělit také do následujících kategorií:

- **Log parsers**

  Extrahují specifické informace ze záznamů protokolu, jako jsou stavové kódy a doby odezvy požadavků z protokolu webového serveru.

- **Log scanners**

  Počítají výskyty řetězců v souborech podle regulárního výrazu. Například pro hledání regulárních chyb i kritických chyb je možno zkontrolovat počet výskytů regulárního výrazu „ERROR|CRITICAL“ v souboru protokolu.

- **Interface readers**

  Čtou přístrojová rozhraní. Příkladem jsou údaje o využití CPU z pseudo-souborového systému Linux /proc a údaje o teplotě nebo vlhkosti ze specializovaných přístrojů.

- **Probers**

  Běží mimo monitorovaný systém a odesílají do systému požadavky na kontrolu jeho odezvy, například ping požadavky nebo HTTP volání na webové stránky pro ověření dostupnosti.

- **Sniffers**

  Monitorují síťová rozhraní a analyzují statistiky provozu, jako je počet přenášených paketů (2).

3.3.2 Datové úložiště

Po sběru logů a metrik je nutné data někam dopravit a uložit. Nejčastěji se používají databáze časových řad (TSDB). TSDB jsou speciální druh databází navržených pro ukládání a správu dat, které jsou časově uspořádány. Tyto databáze se často používají v oblastech jako je monitorování systémů, logování a analyzování dat IoT. Metriky se obvykle ukládají v podobě datových bodů, obsahující hodnotu a časové razítko, kdy měření vzniklo. Jedním z hlavních rozdílů mezi TSDB a běžnými databázemi je, že TSDB jsou optimalizovány pro rychlé vkládání a dotazování dat podle času. Mají mnohem lepší podporu pro práci s velkými objemy dat v reálném čase a podporují speciální funkce, jako jsou časové agregace.

Granularita

Datové body v časových řadách jsou prezentovány v pevné časové granularitě. Jemná granularita představuje kratší časové intervaly. Jemná granularita umožňuje detailněji zobrazit metriky v krátkém časovém úseku, jejich uložení je však nákladné na zdroje. Hrubá granularita metriky je naopak mnohem vhodnější pro zobrazení trendů (2).

Škálování

Některé TSDB se zaměřují na horizontální škálování s pomocí replikace a distribuce dat na více serverů, jiné se soustřeďují na vertikální škálování s využitím výkonnějších serverů a optimalizací dotazů, nebo nabízejí kombinaci obou přístupů. Například populární databáze Elasticsearch, která se používá jako úložiště logů i metrik, disponuje velmi pokročilým horizontálním škálováním pomocí technologie distribuovaného clusteru s možností replikace. Data jsou rozdělena do tzv. shardů. Přidání nového serveru do clusteru znamená automatické rozdělení dat mezi všechny servery (7).

Setřepání dat

TSDB zpravidla setřepávají data po určitém čase. To znamená, že jak data stárnou, více datových bodů je shrnuto do jednoho datového bodu. Běžnou metodou je průměrování, i když některé nástroje podporují jiné metody, jako je sčítání datových bodů . Například dnes

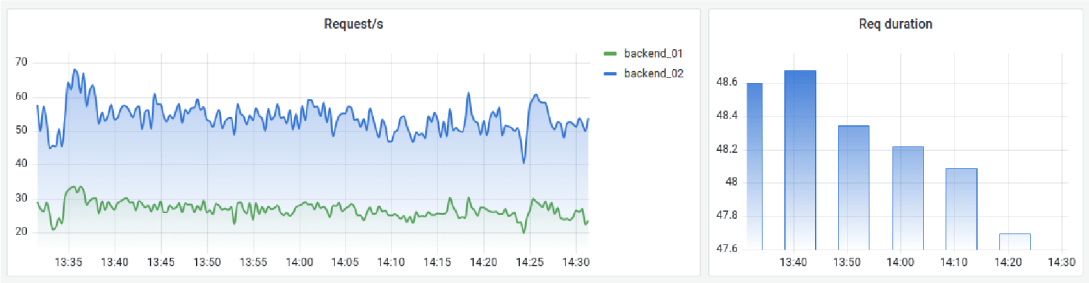
nejpopulárnější InfluxDB má zabudovanou podporu pro setřepání dat v podobě nastavení tzv. retention policy (8).

3.3.3 Vizualizace

Uložená data v databázích časových řad je třeba vizualizovat. Pokud jsou metriky segmentovány do časových intervalů a přepočteny matematickou transformací, mohou být prezentovány jako časové řady a interpretovány na dvourozměrných grafech. Nejběžnější vizualizace je spojnicový graf, ale existují mnohá jiná zobrazení jako plošný, sloupcový nebo koláčový graf, ale také prosté číslo aktuální hodnoty, text nebo tabulka (1).

Graf časové řady (Time series chart)

Tento typ grafu zobrazuje agregované hodnoty v čase. Existuje mnoho variant jako spojnicový, sloupcový nebo plošný graf. V jednom grafu lze kombinovat více metrik nebo je agregovat podle atributu.



Obrázek 10: Graf časové řady (9)

Aktuální hodnota

Tento typ zobrazení se využívá pro souhrnné dashboardy. Zobrazuje aktuální hodnotu metriky. Dashboardy složené z takovýchto kostek podávají souhrnnou informaci o stavu systému. Bývá nastavit prahové hodnoty spolu s barvou reprezentující vážnost situace.

Obrázek 11: Dashboard aktuálních hodnot (9)

Tabulka

Zobrazení v tabulce není na první pohled vizuálně poutavé, ale na druhou stranu je schopné dodat větší množství detailních informací. Tabulkové zobrazení se spíše využívá při analýze protokolu.

Obrázek 12: Dashboard aktuálních hodnot (8)

Dashboardy

Dobrý dashboard odpovídá na otázky vzniklé v určitém čase. Je doporučeno mít jeden hlavní dashboard, který podává souhrnný stav a vedle něj mít více detailních dashboardů, které se věnují konkrétní službě nebo dokonce pouze určitému aspektu. Nejlepší dashboardy se zaměřují na zobrazení stavu jedné služby nebo jednoho produktu. Tyto dashboardy jsou nejefektivnější, pokud je vytvářejí a udržují lidé, kteří službě rozumí nejlépe (1).

Obrázek 13: Příklad dashboardu (9)

Souhrnné statistiky

Sledování metrik ukládá datové vstupy a popisuje jejich vlastnosti. Generování a vykreslování časových řad zahrnuje načtení podmnožiny datových vstupů zadáním sady vlastností (například název hostitele, skupina), jejich rozdělení do rovnoměrně rozložených časových intervalů a matematické shrnutí datových vstupů v každém intervalu. To se provádí pomocí souhrnné statistiky (2). Běžně používané souhrnné statistiky jsou popsány v tabulce.

Statistika	Popis
n	Počet položek v intervalu
Sum	Součet hodnot ze všech vstupů
Avg	Průměrná hodnota pro všechny vstupy (sum / n)
p0-p100	Percentily (0-100) hodnot vstupů, zahrnující min (p0) a max (p100) hodnoty a medián (p50)
deviation	Standardní odchylka od průměru v distribuci shromážděných vstupů

Tabulka 2: Přehled souhrnných statistik

Typy veličin

Metriky lze také interpretovat podle typu veličiny, kterou reprezentují.

- **Flow**

  Tento druh metriky zaznamenává události a jejich vlastnosti. Flow zaznamenává proměnlivý počet vstupů na interval (tj. je multi-N). Data jsou shromažďována z více zdrojů a po agregaci jsou shrnuta. Vysoká variabilita vstupních hodnot umožňuje divákovi vyvodit závěry z rozdělení vstupů. Vysoké hodnoty extrémních percentilů jsou časným indikátorem změn, z nichž některé by mohly být interpretovány jako znepokojující. Příkladem jsou velikosti odeslaných paketů, ceny prodaných položek a doby odezvy pro každý požadavek.

- **Propustnost**

  Měří rychlost zpracování za určité časové období. Metriky propustnosti zaznamenávají kontinuitu a intenzitu toku. Jsou vyjádřeny v jednotkách za čas a ilustrují úroveň spotřeby zdrojů. Protože limity propustnosti lze spolehlivě testovat a jasně definovat, používá se tento typ metriky pro alarmování nasycení zdrojů a identifikaci problematických míst. Příkladem jsou přenosová rychlost a IOPS.

- **Stock**

  Označuje akumulovanou veličinu v určitém časovém okamžiku. Metriky zásob jsou jednoduché N. Zaznamenávají jeden datový vstup na interval datových bodů. Jsou vyjádřeny v jednoduchých jednotkách množství a představují součet aglomerované hodnoty. Úrovně zásob mohou být změněny procesy,

které jsou zaznamenány metrikami průtoku, a intenzita těchto změn je vyjádřena metrikami průchodnosti. Příkladem je využití paměti, volné místo na disku, délka fronty, teplota a úroveň svazku.

- **Dostupnost**

  Měří se, do jaké míry je očekávaný výsledek vrácen. Zdrojem vstupu pro metriky dostupnosti jsou sondy – požadavky vydávané proaktivně, které vracejí úspěch při přijetí očekávané odpovědi nebo selhání v opačném případě. Sondy mohou být interní nebo externí. Jako multiN metrika s nízkou variabilitou vstupních hodnot (1 při úspěchu a 0 při selhání) přináší průměr vstupů dostupnost v procentech (číslo se pohybuje v rozmezí od 0,0 do 1,0). Příkladem jsou požadavky HTTP GET očekávající kód odpovědi 200, výsledky ICMP pingu a ztráty paketů (1).

V této kapitole byly probrány teoretické aspekty monitoringu, byly popsány hlavní úkoly, byla popsána architektura a jednotlivé prvky. V následující kapitole budou tyto znalosti aplikovány.

4 Vlastní práce

Vlastní práce se skládá z přípravné a implementační fáze. Jednotlivé fáze se dále dělí na jednotlivé na sebe navazující kroky.

Přípravná fáze

- Analýza stavu monitoringu webových aplikací
- Návrh řešení nového monitorovacího systému
- Výběr nástroje pro sběr logů a metrik
- Výběr úložiště metrik
- Výběr nástroje pro sledování

Implementační fáze

- Konfigurace a spuštění nástrojů
- Implementace měření
- Implementace dashboardů
- Nasazení a ověření řešení

V dalších oddílech jsou detailně popsány jednotlivé kroky.

4.1 Analýza stavu monitoringu služeb

Již delší dobu byl evidován nedostatečný stav monitoringu ve společnosti na provozovaných projektech. Společnost je střední velikosti s technologickým oddělením o velikosti přibližně 100 členů v 15 týmech. Společnost vyvíjí vlastní produkty určené pro globální trh. Projekty jsou vyvíjeny pro webové a mobilní platformy. Kromě produktů je vyvíjeno mnoho podpůrných služeb a nástrojů interního charakteru. Managementem bylo rozhodnuto udělat průzkum, analyzovat stav a zjistit nedostatky, které jsou popsány v následujícím oddíle.

4.1.1 Vyhodnocení nedostatků

Napříč společností bylo provedeno šetření za účelem zhodnocení aktuálního stavu a získání podnětů pro zlepšení. Šetření probíhalo formou diskuse se zástupci jednotlivých

týmů. Z výsledků průzkumu byly vyhodnoceny následující nedostatky, které byly doporučeny pro okamžité řešení.

1. **Nedostatečná úroveň pokrytí monitoringem**

Úroveň kvality monitoringu jednotlivých služeb se liší a v mnoha případech byla vyhodnocena jako nedostatečná. Služby lze rozdělit do dvou skupin podle toho, zda mají nebo nemají monitoring.

Menší počet služeb je provozován zcela bez jakéhokoliv monitoringu. Jedná se spíše o starší nebo podpůrné služby. Takové aplikace protokolují veškerý svůj výstup v podobě aplikačních logů do souboru v nestrukturované podobě. V lepším případě odesílají protokoly přes síť do centrální databáze Elasticsearch. Tento způsob prakticky neumožňuje mít přehled o aktuálním stavu aplikace. Je určen pouze k zpětné analýze chyb a nelze tak používat proaktivní přístup. Další nevýhodou je nutnost mít znalost prostředí a mít oprávnění k přístupu na patřičné zařízení. Větší počet služeb je provozován s různou kvalitou monitoringu. Použité nástroje jsou často zastaralé, neaktuální nebo nehodící se pro danou službu a řešení se často duplikují.

2. **Vysoké náklady na údržbu infrastruktury monitorovacích systémů**

Každý tým implementuje monitoring vlastními silami. To klade vysoké nároky na vývojáře, neboť kromě vlastní aplikace musejí investovat čas do tvorby a údržby infrastruktury monitorovacího systému. Dochází tak ke zbytečným duplicitním řešení a neefektivitě ve využití lidských zdrojů. Odhaduje se, že celkově se zabývá údržbou monitorovacích infrastruktur přes dvacet vývojářů a techniků s různou mírou vytíženosti.

3. **Nadměrné množství udržovaných technologií**

Historicky je použito větší množství nástrojů. Každý tým si v určité době zavedl vlastní řešení a tato řešení nebyla centrálně koordinována. Za více než deset let se nahromadilo větší množství nástrojů, které je nutné spravovat. Údržba každé technologie znamená investici do hlubšího nastudování dokumentace a starání se o aktualizace. Bylo vyhodnoceno, že udržovat takové množství nástrojů není efektivní.

4. **Vysoká míra ruční práce**

Jednotlivé služby jsou zpravidla ručně nastavovány v jednotlivých nástrojích. Doposud nebylo vyvinuto řešení, která by zredukovalo opakující se ruční nastavování v nástrojích. Nedostatek se týká tvorby dashboardů a ručního nastavení uložišť metrik. Nedostatky a jejich náklady, dopady a rizika jsou shrnuty v následující tabulce.

Nedostatek	**Náklady, dopady, rizika**
Nedostatečná úroveň pokrytí monitoringem aplikací a služeb	Pozdní detekce chyb. Omezená analýza chyb. Ztráty, pokuty, ztráta důvěry.
Vysoké náklady na údržbu infrastruktury monitorovacích systémů	Alokace 20+ lidí napříč týmy. Odhadované roční náklady 2000 MD.
Nadměrné množství udržovaných technologií	Minimálně 12 nástrojů. (Filebeat, Logstash, rsyslog, Prometheus, InfluxDB, Elasticsearch, Kibana, Cacti, Icinga, Navigator, Grafana, Chronograf)
Vysoká míra ruční práce	Odhadované náklady na vytvoření dashboardu, registrace služby a nastavení úložiště: 1MD

Tabulka 3: Vyhodnocení nedostatků

4.1.2 Stanovení cílů a návrh řešení

Každý nedostatek byl zkoumán a byl pro něj stanoven cíl v podobě nápravy. Byly stanoveny následující cíle:

1. **Sjednocení úrovně monitorovacích systémů**

Cílem je sjednocení úrovně monitoringu pro všechny provozované služby. U služeb, které postrádají monitoring by mělo dojít k doplnění monitoringu. U služeb, které monitoring již mají, by mělo dojít k sjednocení řešení a úrovně monitoringu.

2. **Snížení nákladů na provozování infrastruktury monitoringu**

Cílem je celkové snížení nákladů týkajících se infrastruktury monitorovacích systémů. Toho by mělo být dosaženo vytvořením jednotného monitoringu a k tomu specializovaného

týmu v počtu dvou až tří pracovníků, kteří budou vyčleněni pouze pro tyto účely. Mělo by tedy dojít k redukci alokace lidských zdrojů v ostatních týmech.

3. **Redukce počtu provozovaných nástrojů**

Cílem je omezit počet udržovaných nástrojů. Redukce technologií umožní snížení počtu potřebných odborníků a jednodušší správu. Neznamená to však, že by se neměly zavádět a zkoušet jiné technologie. V takovém případě je vhodné technologii vyzkoušet a po nějakou dobu provozovat v experimentálním provozu. Po předem stanoveném čase je nutné rozhodnout, zda toto nové řešení zastoupí to stávající. Pokud ne, musí být experimentální nástroj odstraněn, aby nedocházelo ke kumulování nástrojů. Opačně, v případě zvolení nového řešení, musí být stávající nástroj nahrazen a odstraněn.

4. **Redukce ručního nastavování**

Cílem je snížit míru opakujícího se nastavování v nástrojích. Řešení by mělo umožnit automaticky zaregistrovat novou webovou službu, nové měření a vygenerovat nový dashboard. Měla by se tím minimalizovat ruční práce při nastavování indexů a dashboardů.

Celkově bylo vyhodnoceno, že pro dosažení cílů je nutné vytvořit nový monitorovací systém, který nahradí všechna stávající řešení. Nový systém by měl být složen z pečlivě vybraných komponent a měl by umožňovat vysokou míru automatizace. V dalším oddíle byly sepsány požadavky, které musí tento nový systém splňovat.

4.2 Návrh řešení nového monitorovacího systému

V následujícím oddíle je představeno řešení nového monitorovacího systému. Návrh řešení se skládá z následujících fází:

- Definice požadavků
- Návrh architektury
- Fáze projektu

4.2.1 Definice požadavků

Z výše zmíněných nedostatků, expertních doporučení a zjištěných teoretických znalostí, vyplynuly následující požadavky pro nově vznikající monitorovací systém, které jsou v souladu se stanovenými cíli.

Požadavky na měření

Webové služby jsou založeny na principu http requestů. Měření requestů je tedy primární statistika, která je sledována. Důležité jsou ale i systémové statistiky jako využití cpu nebo paměti, a společně tak dodávají komplexní pohled.

P.1	Požadavky na měření
P.1.1	**Požadavek na měření bloku kódu** V tomto požadavku je záměrem měřit dobu vykonání bloku kódu. Je to nejčastější metrika, která se používá pro výkonnostní měření. Typicky v případě měření http requestu měření začíná na začátku zpracování požadavku a je ukončeno na konci těsně před odesláním. Měřit tak lze ale jakýkoliv blok kódu, kde se něco vykonává. Typickými případy jsou doba vykonání databázového dotazu nebo vykonání náročné operace.
P.1.2	**Požadavek na měření systémových hodnot** Aplikace by měla poskytovat informace o využití paměti a cpu. V této fázi implementace budou stačit pouze tyto informace, později ale je vhodné doplnit o další statistiky jako využití disku a síťové statistiky.
P.1.3	**Požadavek na agregaci a filtrování** Tento požadavek byl doplněn z praxe. Velmi často je nedostatečné obecné měření http požadavku, ale je třeba měření doplnit o dodatečné informace, podle kterých lze detailněji přistupovat ke statistikám. V případě http requestu se jedná například o název requestu nebo http metodu.

Tabulka 4: Požadavky na měření

Požadavky na funkčnost dashboardů

Zkoumáním současných řešení byl sjednocen souhrn minimálních požadavků na kvalitu dashboardů a funkcionalit, které by měly zahrnovat.

P.2	Požadavky na funkčnost dashboardů
P.2.1	**Požadavek na zobrazení metriky v čase** V tomto požadavku je zohledněno zobrazení sledované metriky v čase v podobě grafu. Uživatel by měl mít možnost nastavit časové rozpětí s minimální granularitou času 1s.
P.2.2	**Požadavek na agregované zobrazení metriky** Tento požadavek doplňuje předchozí požadavek na zobrazení metriky v čase. Zde by mělo dojít k agregaci metriky podle určitého atributu. Typickým příkladem je rozdělení http requestů podle url. To dodá mnohem detailnější informaci, například který request je nejvíce volán nebo který request má nejdelší dobu zpracování.
P.2.3	**Požadavek na zobrazení aktuální hodnoty metriky** Velmi častým případem zobrazení hodnot v dashboardu je prosté zobrazení aktuální hodnoty konkrétní metriky. Jedná se například o aktuální počet http requestů nebo aktuální dobu vyřízení requestu. Součástí této hodnoty by mělo být i nastavení prahových hodnot.

Tabulka 5: Požadavky na funkčnost dashboardů

Požadavky na automatizaci

Systém by měl umožnit následující funkce bez nutnosti ručního nastavení v nástrojích.

P.3	Požadavky na automatizaci
P.3.1	**Automatické generování dashboardů** Tento požadavek řeší zdlouhavé a opakující se nastavování grafů. Nastavení dashboardu by mělo být realizováno formou zjednodušeného konfiguračního souboru, který by měl být součástí knihovny projektu. Synchronizace a nahrání dashboardů by mohlo být spojeno například s nasazením nové verze služby.
P.3.2	**Automatická registrace metrik** Cílem tohoto požadavku je automatické zaregistrování metrik, aniž by bylo nutné ručně nastavovat úložiště metrik nebo nástroj pro dopravu logů a metrik.
P.3.3	**Automatická registrace webové služby** Cílem tohoto požadavku je automatické zaregistrování webové služby, aniž by bylo nutné ručně nastavovat úložiště metrik nebo nástroj pro dopravu logů a metrik.

Tabulka 6: Požadavky na automatizaci

Technické požadavky

Následuje výčet požadavků, které mají technický charakter a byly stanoveny experty z technického oddělení.

P.4	Technické požadavky
P.4.1	**Požadavek na modularitu** Systém by měl být modulární, tj. jednotlivé komponenty by měly umožňovat nahrazení jiným nástrojem, aniž by bylo nutné měnit jiné moduly. Bude tedy kladen důraz na schopnosti integrace s jinými komponentami. V praxi to znamená například výměnu konkrétní TSDB za jinou.
P.4.2	**Požadavek na open-source** Jsou preferovány nástroje open-source. Částečně tento požadavek vychází i z požadavku nasazení na různých zařízeních a modularitu. Komerční řešení jsou primárně provozována jako služby v cloudovém prostředí. Není tedy zpravidla možné je provozovat na vlastních zařízeních. Problémem je pak i požadavek na modularitu, protože komerční řešení dodávají hotové řešení, které lze komplikovaně kombinovat s jinými nástroji. Určité hledisko hrají i náklady na provoz.
P.4.3	**Požadavek na nasazení v různých prostředích** Vzhledem k tomu, že služby jsou provozovány v různých prostředích, důležitým požadavkem je i schopnost snadno zprovoznit monitorovací systém v různých prostředích na různých OS.

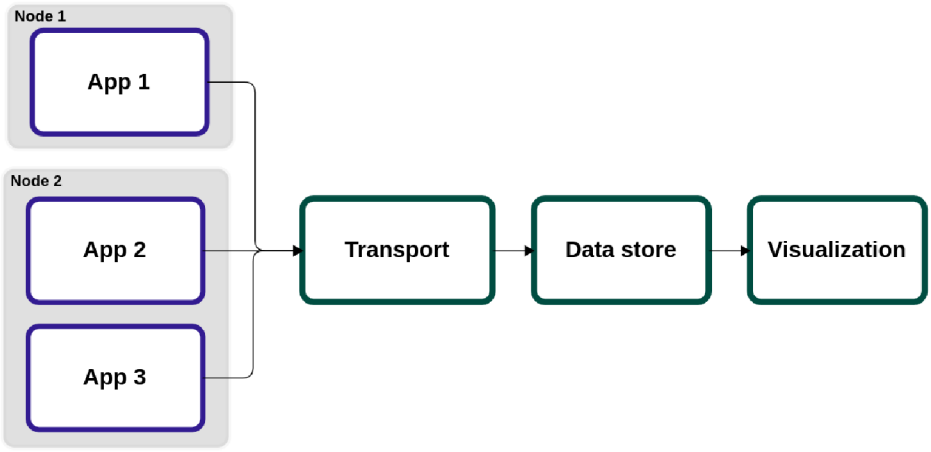
Tabulka 7: Technické požadavky

4.2.2 Návrh architektury

Jak bylo zjištěno v teoretické části, moderním trendem v oblasti monitoringu je tzv. Composable monitoring. Systém tedy musí bude složen z volně propojených komponent. V systému se vyskytují tyto tři komponenty:

- Nástroj pro sběr logů a metrik
- Úložiště metrik
- Nástroj pro sledování

Volba komponent musí zohledňovat technické požadavky na modularitu a open-source. Při volbě vhodné technologie pro jednotlivé komponenty je nutná schopnost výměnu nástroje pro určitou komponentu, aniž by bylo nutné měnit ostatní komponenty. Například pokud v budoucnu dojde k rozhodnutí nahradit úložiště metrik za jiný TSDB, pak by tomu neměla bránit komponenta pro sběr logů a metrik ani komponenta pro vizualizaci. Tato podmínka v realitě není zcela splněna, nástroje pro sběr logů a metrik a nástroje pro sledování zpravidla podporují pouze omezený počet úložišť.



Obrázek 14: Schéma monitorovacího systému

4.2.3 Fáze projektu

Celý projekt je rozčleněn do tří hlavních etap, které se dále dělí na dílčí kroky. První etapou je výběr nástrojů, následuje implementační etapa a nakonec etapa nasazení a ověření řešení.

Výběr nástrojů

Před samotnou implementací je nutné pečlivě vybrat konkrétní nástroje pro každou komponentu monitorovacího systému zmíněné v předchozím oddíle. Kritéria pro hodnocení se nastavují s ohledem na požadavky.

- **Výběr nástroje pro sběr logů a metrik**

V tomto kroku by mělo dojít k výběru nástroje pro sběr a transport metrik do úložiště metrik. Při výběru nástroje je potřeba zohlednit požadavky na měření, automatizaci i technické požadavky.

- **Výběr úložiště metrik**

V tomto kroku je úkolem vybrat ideální úložiště metrik. Je nutné zohlednit technické požadavky konkrétně požadavek na modularitu a požadavek na open-source.

- **Výběr nástroje pro sledování**

V tomto kroku je úkolem vybrat nástroj pro sledování, který by nejlépe splňoval potřebné požadavky. Nástroj by měl mít největší podporu úložišť, měl by splňovat nároky na funkcionalitu dashboardů a měl by umožňovat automatické vytváření dashboardů. Při výběru nástroje je potřeba zohlednit požadavky na funkcionalitu dashboardů, automatizaci a technické požadavky.

Implementace

Po pečlivém vybrání nástrojů následuje fáze implementace, která se skládá z následujících kroků:

Konfigurace a spuštění nástrojů

V tomto kroku je nutné vybrané nástroje zprovoznit a nakonfigurovat. Vhledem k požadavku na snadné spuštění v různých prostředích, je nutné využít techniku kontejnerizace. Společnost již využívá technologii Docker, která je standardem v této oblasti. Znamená to tedy, že jednotlivé komponenty budou spouštěny a orchestrovány v Docker kontejnerech. Součástí tohoto kroku je i správné nakonfigurování nástrojů. V tomto kroku je nutné zohlednit požadavky na technický požadavky na spuštění v různých prostředích.

- **Implementace měření**

V tomto kroku bude výsledkem implementace knihovny, která umožní měřit a ukládat jednotlivá měření. Při implementaci je nutné zohlednit požadavky na měření a zajistit tedy měření bloků kódu a systémových statistik. Zároveň je nutné zohlednit požadavek na automatizaci, tedy zajistit takové řešení, které umožní automatickou registraci služby a jejích metrik bez nutnosti ručního nastavování v patřičném nástroji.

- **Implementace dashboardu**

Posledním krokem implementace bude vytvoření dashboardu. Dashboard musí splňovat požadavek na funkcionalitu. Dashboardy se nebudou vytvářet ručním nastavením ve vybraném nástroji, ale budou se generovat automaticky dle konfigurace ze zdrojového kódu. Musí být tedy vytvořena knihovna pro automatické generování dashboardu.

Nasazení a ověření řešení

Cílem poslední fáze je ověření implementovaného řešení a skládá se z následujících kroků:

- **Měření ve vzorové aplikaci.**

V tomto kroku dojde k nainstalování a spuštění připraveného monitorovacího systému v testovacím prostředí. Tím dojde k ověření požadavku na snadné spuštění v různých prostředích. Na vzorové aplikaci bude implementováno měření pomocí připravené knihovny na měření a tím dojde k ověření požadavků na měření a automatizaci. Nakonec budou implementovány dashboardy v plné funkcionalitě a tím dojde k ověření požadavků na funkcionalitu dashboardů a automatizaci.

- **Otestování stavů aplikace podle scénáře.**

V posledním kroku bude otestováno celé řešení. Budou navozeny stavy, které by měl monitoring správně zobrazit a detekovat problémové stavy. Testovat se budou následující stavy:

  - Běžný provoz
  - Přetížení webové aplikace
  - Výskyt pomalého requestu
  - Výskyt chyb

Následující oddíly blíže popisují jednotlivé kroky projektu.

4.3 Výběr nástroje pro sběr logů a metrik

V tomto kroku byl vybrán vhodný nástroj pro sběr logů a metrik jako jedna z komponent monitorovacího systému navrženého v předchozím oddíle. Proces výběru je složen z následujících kroků:

- Volba kritérií a nastavení vah.
- Výběr nástrojů pro porovnání.
- Hodnocení variant a výběr nástroje

4.3.1 Volba kritérií a nastavení vah

Při volbě kritérií byly zohledněny technické požadavky, požadavky na měření a automatizaci, doplněné doporučeními expertů. Byla vybrána následující kritéria:

1. **Podpora vstupů**

   Toto hledisko se zaměřuje na podporu vstupů typicky zpracování souborů, ale i síťové vstupy jako tcp, udp či http. Výhodou jsou i hotová řešení pro známé technologie, např. sledování systémových prostředků, sítě, statistiky databází či jiných úložišť. V úvahu je nutné brát i schopnosti zpracovávat různé formáty a možnosti transformace, filtrace nebo agregace.

2. **Podpora úložišť**

   Toto kritérium vychází z požadavku na modularitu. Jsou preferovány ty nástroje, které podporují výstup do co největšího počtu úložišť metrik typu TSDB.

3. **Výkon**

   Hledisko výkonu je důležité zejména u vysoce zatěžovaných systémů. Nástroj, který šetří systémové prostředky a propustí více dat, bude mít menší náklady na zdroje a pravděpodobně bude potřebovat méně často zásah a péči technika.

4. **Podpora projektu**

   Je preferováno vybírat nástroje, které mají dobrou podporu projektu. Nezanedbatelné hledisko v tomto ohledu je popularita nástroje. Dá se očekávat, že populární nástroje budou více rozvíjeny, protože je zde mohutnější zpětná vazba. Oblíbenost lze vysledovat například z počtu stažených Docker image na webu docker.hub, a nebo počty sledujících na Githubu případně počty článků na specializovaných portálech jako Stack overflow.

Nastavení vah kritérií se významně mění dle potřeb a specifik konkrétního projektu. Vhledem k tomu, že jeden z hlavních parametrů je požadavek na modularitu, pak v tomto případě vysoké hodnocení získalo kritérium podpora úložišť. Pro přesnější volbu vah jednotlivých kritérií byla zvolena Saatyho metoda párového porovnání. Preference jsou klasicky zvoleny na škále od 1 do 9, kde 1 je stejná důležitost, 2 velmi slabá preference až po 9 totální preference.

Vzájemným porovnáním pak vzešla tabulka s výslednými vahami kritérií.

Kritérium	Vstupy	Úložiště	Výkon	Podpora	GeomPrum.	Váha
Vstupy	1.00	0.50	2.00	1.00	**1.00**	**23%**
Úložiště	2.00	1.00	2.00	3.00	**1.86**	**43%**
Výkon	0.50	0.50	1.00	0.50	**0.59**	**14%**
Podpora	1.00	0.33	2.00	1.00	**0.90**	**21%**
Celkem					**4.36**	**100%**

Tabulka 8: Výběr transportní vrstvy: nastavení vah kritérií

4.3.2 Výběr nástrojů pro porovnání

Pro výběr nástrojů bylo zvoleno expertní doporučení doplněné rešerší článků. Pro vyhledání článků byly formulovány dotazy jako "Logs and metrics shipping tools" případně "Logs and metrics shipping tools comparison". Jako zdroj byl použit vyhledavač Google. Jako nejlepší stránky pojednávající nad touto tematikou byly vyhodnoceny články: Top 5 Open-Source Log Shippers (alternatives to Logstash) in 2022 (10), 20+ Best Log Management Tools for Monitoring, Analytics & More: Pros & Cons Comparison (11), Logstash, Fluentd, Fluent Bit, or Vector? How to choose the right open-source log collector (12). Nalezené články se zabývaly nejen open source, ale i komerčními nástroji. Dle požadavku na open source (P.4.2) bylo nutné očistit výběr o komerční nástroje. Do užšího výběru byly vybrány nástroje Logstash, Fluentd a Vector, které byly nejčastěji zmiňovány a objevovaly se na předních místech a doporučeních.

1. **Logstash**

   Logstash byl poprvé vydán v roce 2009 a byl původně vyvinut Jordanem Sissellem. Později se stal součástí společnosti Elastic a je stále aktivně vyvíjen a udržován jako open-source projekt. Logstash byl navržen jako nástroj pro sběr, zpracování a převod různých typů dat, včetně logů, metrik, událostí a dalších informací. Logstash je jeden z nejstarších nástrojů a má velkou popularitu, podporuje mnoho vstupů i mnoho výstupů, kromě ElasticSearch také několik TSDB. Často je však kritizován jeho výkon a složitá konfigurace. Problém s výkonem řeší kombinace s nástrojem Filebeat, který je nenáročný a lze jej snadno škálovat (13).

2. **Fluentd**

   Fluentd je open-source nástroj pro sběr, zpracování a předávání logů. Jeho návrh je postaven na modulární architektuře a pluginy mu umožňují podporovat širokou škálu vstupů a výstupů. Fluentd je velmi flexibilní a umožňuje uživatelům přizpůsobit sběr

a zpracování logů podle svých potřeb. Jeho výhody zahrnují širokou podporu vstupů a výstupů, robustnost, jednoduchou konfiguraci a škálovatelnost. Projekt Fluentd vznikl v roce 2011 pod vedením Masahiro Nakagawy, japonského programátora a zakladatele společnosti Treasure Data, která je dnes hlavním přispěvatelem projektu (14).

3. **Vector**

   Vector je open-source nástroj pro sběr, transformaci a doručování logů a metrik. Byl vytvořen v roce 2019 společností Timber Technologies, Inc. jako moderní alternativa ke stávajícím nástrojům pro sběr a zpracování dat, jako jsou Logstash nebo Fluentd. Vector má jednoduchou a modularizovanou architekturu, která umožňuje snadné přidávání nových vstupů, filtrů a výstupů. Podporuje mnoho různých formátů záznamů, včetně JSON, CSV, Syslog a Apache access log, a podporuje výstupy do mnoha různých úložišť dat, jako jsou Elasticsearch, Kafka nebo Amazon S3, ale i populární TSDB jako InfluxDB, Graphite, Prometheus a OpenTSDB. Mezi klíčové vlastnosti Vectoru patří nízká režie, snadná konfigurace, podpora sběru logů a metrik a flexibilita. Vector je navržen tak, aby byl snadno použitelný a škálovatelný pro velké a složité prostředí (15).

4.3.3 Hodnocení a výběr nástroje

V tomto kroku byly nástroje ohodnoceny ve všech zmíněných kritériích. Pro hodnocení byla opět zvolena Saatyho metoda párového porovnání.

1. **Hodnocení kritéria: Podpora vstupů a transformace**

Zkoumáním oficiálních dokumentací byly porovnány jednotlivé nástroje z pohledu podpory vstupů a zpracování dat. Každý z jmenovaných nástrojů podporuje široké spektrum vstupů, ať už v základní podpoře nebo v podobě pluginů. Nabídka je ve všech případech srovnatelná, lze však vyhodnotit, že nástroj Vector má o něco slabší nabídku vstupů než Logstash a Fluentd. Všechny uvedené nástroje umožňují následné zpracování dat, transformace, agregace či filtrování v požadované kvalitě.

Kritérium	Logstash	Fluentd	Vector
Podpora vstupů	Podpora více než 200 vstupních pluginů	Podpora více než 500 vstupních pluginů	Podpora více než 50 vstupních pluginů
Transformace dat	Podpora různých transformačních pluginů, jako grok, json, xml, csv, mutate, fingerprint, date, dns, a další.	Podpora několika transformačních pluginů, jako record_transformer, grep, cut, parser, a další.	Podpora transformačních pluginů jako concatenate, parser, regex, lua, a další.

Tabulka 9: Výběr transportní vrtsvy - podpora vstupů a transformace

Vzájemným porovnáním jednotlivých nástrojů pak vzešla následující tabulka.

Vstupy	Logstash	Fluentd	Vector	GeomPrum.	Váha
Logstash	1,00	0,50	3,00	**1,14**	**35%**
Fluentd	2,00	1,00	2,00	**1,59**	**48%**
Vector	0,33	0,50	1,00	**0,55**	**17%**
Celkem				**3,28**	**100%**

Tabulka 10: Výběr transportní vrstvy – hodnocení kritéria podpora vstupů

2. Hodnocení kritéria: Podpora úložišť

Logstash byl primárně vyvinut pro přepravu logů do Elasticsearch, má však i mnoho výstupů různého typu například email, soubor, S3, Kafka a mnoho dalších. Pokud jde o podporu TSDB úložišť, pak podporuje pouze omezený počet a to InfluxDB, OpenTSDB a Graphite a především Elasticsearch. Veliká síla Fluentd je v opravdu široké podpoře různých úložišť. Podporuje prakticky všechny populární TSDB jako InfluxDB, OpenTSDB, Elasticsearch, Graphite, Prometheus nebo TimescaleDB. Vektor má také mnoho výstupů do různých úložišť včetně komerčních a cloudových služeb. Z populárních TSDB to jsou InfluxDB, Elasticsearch, OpenTSDB, Graphite a Prometheus. Přehled podpory sledovaných TSDB zobrazuje následující tabulka:

TSDB	Logstash	Fluentd	Vector
InfluxDB	Ano	Ano	Ano
OpenTSDB	Ano	Ano	Ano
Elasticsearch	Ano	Ano	-
Graphite	-	Ano	Ano
Prometheus	-	Ano	Ano
TimescaleDB	-	Ano	Ano

Tabulka 11: Výběr transportní vrstvy – podpora TSDB

Vzájemným porovnáním jednotlivých nástrojů pak vzešla následující tabulka.

Úložiště	Logstash	Fluentd	Vector	GeomPrum.	Váha
Logstash	1,00	0,50	0,50	**0,63**	**20%**
Fluentd	2,00	1,00	2,00	**1,59**	**49%**
Vector	2,00	0,50	1,00	**1,00**	**31%**
Celkem				**3,22**	**100%**

Tabulka 12: Výběr nástroje pro sběr logů a metrik – Hodnocení kritéria podpora úložišť.

3. Hodnocení kritéria: Výkon

V článku "Top 5 Open-Source Log Shippers (alternatives to Logstash) in 2022" (10) si dali autoři práci s porovnáním výkonu zmíněných nástrojů. Zaměřili se na vytížení paměti, procesoru a disků. Z tohoto soupeření vyšel Vector jako těsný vítěz nad Fluentd, pokud tedy preferujeme spotřebu paměti nad spotřebou cpu. Logstash dopadl ve všech metrikách nejhůře, ostatně jeho výkon byl často kritizován i v jiných článcích, zabývajících se touto tématikou. V tabulce 13 jsou zobrazeny výsledky měření.

Metrika	Fluentd	logstash	Vector
IO Thrpt (avg)	27,7 MiB/s	40,6 MiB/s	86 MiB/s
CPU sys (max)	3,5	6,1	6,5
CPU usr (max)	50,8	91,5	96,5
Load 1m (avg)	0,8	1,8	1,7
Mem used (max)	294 MiB	742 MiB	181 MiB
Disk read (sum)	2,6 MiB	2,6 MiB	2,6 MiB
Disk writ (sum)	13,7 MiB	11,6 MiB	11 MiB

Tabulka 13: Výběr nástroje pro sběr logů a metrik – srovnání výkonu nástrojů (10)

Vzájemným porovnáním jednotlivých nástrojů pak vzešla tabulka s výsledným hodnocením.

Výkon	Logstash	Fluentd	Vector	GeomPrum.	Váha
Logstash	1,00	0,50	0,33	**0,55**	**16%**
Fluentd	2,00	1,00	0,50	**1,00**	**30%**
Vector	3,00	2,00	1,00	**1,82**	**54%**
Celkem				**3,37**	**100%**

Tabulka 14: Výběr nástroje pro sběr logů a metrik – hodnocení výkonu nástrojů

4. Hodnocení kritéria: Podpora projektu

V této studii došlo ke snaze změřit oblíbenost projektu z dostupných zdrojů. Popularitu a velikost komunity nelze exaktně změřit. Lze ji ale odhadnout například podle počtu stažení image z Docker hub, podle počtu sledujících nebo přispěvatelů na Github, nebo podle počtu článků na StackOverflow. Z následující tabulky lze odvodit, že stále největší komunita je kolem nástroje Logstash, nicméně popularita a život kolem Vectoru je téměř srovnatelná. V tomto ohledu pak fluentd lehce zaostává.

Data	Logstash	Fluentd	Vector
hub.docker.com	100M	50M	100M
stackoverflow.com	6240	1748	3700
Github forks	3397	1284	1022
Github stars	13292	11802	12793
Github open isssues	1977	136	1681

Tabulka 15: Výběr transportní vrstvy – srovnání popularity nástrojů.

Vzájemným porovnáním jednotlivých nástrojů pak vzešla tabulka s výslednými vahami.

Podpora	Logstash	Fluentd	Vector	GeomPrum.	Váha
Logstash	1,00	3,00	2,00	**1,82**	**54%**
Fluentd	0,33	1,00	0,50	**0,55**	**16%**
Vector	0,50	2,00	1,00	**1,00**	**30%**
Celkem				**3,37**	**100%**

Tabulka 16: Výběr transportní vrstvy – hodnocení kritéria podpora projektu

Celkové hodnocení a výběr nástroje

Hodnocení všech kritérií byla vložena do výsledné tabulky a po zohlednění dříve vypočtených vah vyšlo celkové hodnocení, znázorněné v následující tabulce.

Hodnocení	Vstupy	Úložiště	Výkon	Podpora	Hodnocení
Logstash	0,40	0,20	0,16	0,54	**30,95%**
Fluentd	0,40	0,49	0,30	0,16	**37,68%**
Vector	0,20	0,31	0,54	0,30	**31,37%**
Celkem	100,0%	100,0%	100,0%	100,0%	**100,00%**
Váha	23%	43%	14%	21%	

Tabulka 17: Výběr transportní vrstvy – celkové hodnocení

Vybrán byl nástroj Fluentd. Je nutné zvážit, že výsledek nebyl zcela jednoznačný, a s trochu jiným nastavením kritérií nebo změnou preferencí by mohl být zvolen jiný nástroj. V tomto případě rozhodla zřejmě o něco lepší podpora úložišť a celkově dobré výsledky ve všech kritériích. Případná výměna nástroje v budoucnu by neměla být problémem, protože je kladen důraz na modularitu.

4.4 Výběr úložiště metrik

V této fázi bylo vybráno vhodné úložiště metrik jako jedna z komponent monitorovacího systému. Proces výběru je složen z následujících kroků:

- Volba kritérií a nastavení vah.
- Výběr nástrojů pro porovnání.
- Hodnocení variant a výběr nástroje

4.4.1 Volba kritérií a nastavení vah

Při volbě kritérií byly zohledněny technické požadavky a požadavky na měření doplněné doporučeními expertů. Byla vybrána následující kritéria:

1. **Škálování**
   Škálování je důležité, protože se týká schopnosti systému zvládnout nárůst počtu dat, počtu klientů a celkového objemu provozu. Škálovatelnost zahrnuje jak horizontální tak vertikální škálovatelnost. Toto kritérium vychází z expertního doporučení.
2. **Diskové nároky**
   Spotřeba diskového prostoru je rovněž důležitá, protože ukládání velkého množství dat může mít vysoké nároky na diskový prostor a v konečném důsledku na náklady. Toto kritérium vychází z expertního doporučení.
3. **Integrace**
   Kritérium integrace se týká schopnosti databáze propojovat se s ostatními nástroji. V praxi to znamená, jak široce je úložiště podporováno nástroji pro dopravu logů a metrik a nástroji pro sledování. Toto kritérium zohledňuje technický požadavek na modularitu (P.4.1).
4. **Podpora projektu**
   Popularita je také důležitým faktorem, protože populární databáze jsou obvykle

vývojáři a uživateli více podporovány a mají větší ekosystém nástrojů a knihoven. Toto kritérium vychází z expertního doporučení.

Nastavení vah kritérií

Při zvažování důležitosti jednotlivých kritérií se došlo k tomu, že nejpreferovanějším kritériem je škálování. V případě, že by aplikace začala generovat větší množství metrik, řešení by mělo umožnit škálování a přizpůsobit se tak se zvýšenému provozu. Podstatné je i kritérium modularity. Jako nejméně důležité kritérium byla vyhodnocena spotřeba diskového prostoru, vzhledem k tomu, že náklady na disky dnes nejsou zdaleka tak nákladné, jako dříve. Vzájemným porovnáním jednotlivých kritérií vzešla následující tabulka:

Kriterium	Integrace	Škálování	Disk. Nároky	Podpora	GeomPrum.	Váha
Integrace	1.00	0.50	3.00	2.00	**1.32**	**29%**
Škálování	2.00	1.00	3.00	2.00	**1.86**	**41%**
Disk. nároky	0.33	0.33	1.00	0.50	**0.49**	**11%**
Podpora	0.50	0.50	2.00	1.00	**0.84**	**19%**
Celkem					**4.50**	**100%**

Tabulka 18: Výběr úložiště metrik – vyhodnocení vah kritérií

4.4.2 Výběr nástrojů pro hodnocení

V předchozím kroku při výběru nástroje pro sběr logů bylo zkoumáno, které TSDB jsou podporovány. Nejčastěji byly zmiňovány následující databáze: InfluxDB, OpenTSDB, Graphite, TimescaleDB a ElasticSearch. InfluxDB a OpenTSDB jsou podporovány všemi nástroji pro dopravu logů a metrik. Elasticsearch je od začátku mírně preferován, protože je již ve společnosti používán pro aplikační protokoly, což zjednodušuje práci se správou jediného úložiště a zjednodušené dotazování do úložiště. Do užšího porovnání tedy byly vybrány databáze InfluxDB, ElasticSearch a OpenTSDB.

1. **InfluxDB**

   InfluxDB je open-source databáze, která se specializuje na ukládání a analýzu časových řad. Projekt InfluxDB byl poprvé vydán v roce 2014 a od té doby se stal velmi populární v řadě odvětví. InfluxDB se vyznačuje jednoduchým a intuitivním dotazovacím jazykem InfluxQL, který umožňuje snadno dotazovat a agregovat data v časových řadách. InfluxDB také nabízí mnoho funkcí pro zpracování dat, jako jsou například kontinuální dotazování nebo agregace dat. Další zajímavou funkcí InfluxDB je možnost definování tzv. retention policies, které umožňují

konfigurovat, jak dlouho jsou data uchovávána v databázi, a jak často jsou data agregována (8).

2. **Elasticsearch**

   Elasticsearch je open-source fulltextová databáze, která se specializuje na ukládání, vyhledávání a analýzu velkých objemů dat v reálném čase. Byl poprvé vydán v roce 2010 a od té doby se stal jedním z nejpopulárnějších nástrojů pro vyhledávání a analýzu dat v reálném čase. Je postaven na jazyce Lucene, což je výkonný full-textový vyhledávací engine, který umožňuje poskytovat rychlé a přesné vyhledávání v nejrůznějších typech dat. Podporuje mnoho funkcí, jako jsou fultextové a agregační dotazy. Další zajímavou funkcí je schopnost horizontálně škálovat (7).

3. **OpenTSDB**

   OpenTSDB je open-source TSDB databáze, která se specializuje na ukládání a analýzu velkých objemů časových řad. Je postaven na jazyce HBase, což je distribuovaná databáze NoSQL, která poskytuje vynikající škálovatelnost pro ukládání velkých objemů dat. OpenTSDB také nabízí mnoho funkcí pro zpracování dat, jako jsou například agregace dat, kontinuální dotazování, statistické funkce a mnoho dalšího. Jednou z unikátních funkcí OpenTSDB je podpora metrik s více dimenzemi. To znamená, že lze snadno ukládat data, která mají více dimenzí, a dotazovat se na tato data pomocí různých kombinací dimenzí. Je velmi oblíbený zejména v odvětvích, jako jsou například IoT, monitoring a logování (16).

4.4.3 Hodnocení a výběr nástroje

V tomto kroku jsou databáze ohodnoceny ve všech zmíněných kritériích. Pro hodnocení byla opět zvolena Saatyho metoda párového porovnání.

Hodnocení kritéria: Škálování

Zkoumáním oficiální dokumentace jednotlivých nástrojů byly zjištěny následující informace ohledně schopnosti škálování. Elasticsearch, InfluxDB a OpenTSDB jsou všechny schopny škálovat horizontálně, což znamená, že lze snadno přidávat další servery pro zvýšení kapacity. InfluxDB nabízí možnost automatického škálování pomocí clusteru, což umožňuje snadné rozšiřování a zmenšování kapacity podle aktuálních potřeb. OpenTSDB podporuje vícenásobné uzly pro škálování, ale nemá tak vyspělé možnosti jako

Elasticsearch a InfluxDB. Proti InfluxDB hraje bohužel i nutnost zakoupit licenci. Jednotlivé aspekty škálování jsou shrnuty v následující tabulce:

Parametr	InfluxDB	Elasticsearch	OpenTSDB
Horizontální škálování	Pomocí replikace dat, shardingu dat a HA protokolu. **Pro produkční prostředí nutná placená licence.**	Pomocí shardingu dat a replikace uzlů.	Pomocí replikace dat a shardingu dat.
Vertikální škálování	Snadné škálování vertikálně.	Snadné škálování vertikálně.	Snadné škálování vertikálně.
Replikace dat	Replikaci dat pro zajištění vysoké dostupnosti.	Replikaci dat mezi uzly, aby byla zajištěna vysoká dostupnost dat.	Replikaci dat mezi uzly pro zajištění vysoké dostupnosti dat.
Sharding	Sharding pro rozdělení dat mezi různé uzly, aby byl zajištěn vysoký výkon a dostupnost.	Sharding pro rozdělení dat mezi různé uzly a zajištění vysokého výkonu a dostupnosti.	Sharding pro rozdělení dat mezi různé uzly, aby byl zajištěn vysoký výkon a dostupnost.
Clustering	Lze nasadit jako cluster s několika uzly, což umožňuje řízení zátěže a zajišťuje vysokou dostupnost.	Lze nasadit jako cluster s několika uzly, což umožňuje rozdělení zátěže a zajišťuje vysokou dostupnost.	Lze nasadit jako cluster s několika uzly, což umožňuje řízení zátěže a zajišťuje vysokou dostupnost.

Tabulka 19: Výběr úložiště metrik – srovnání možností škálování.

Vzájemným porovnáním vyšlo následující hodnocení:

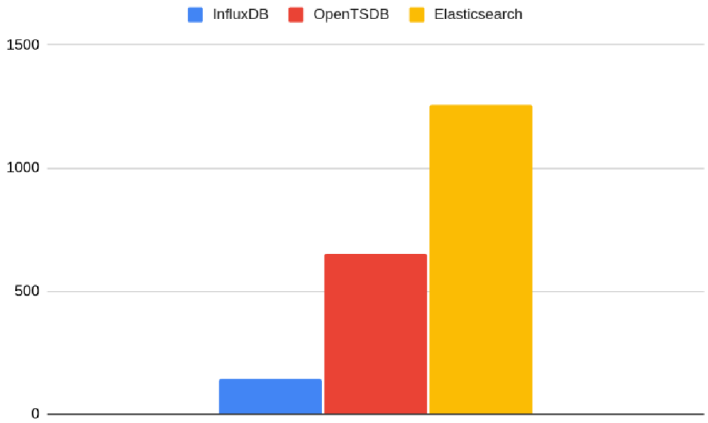
Škálování	Influxdb	Elastic	OpenTSDB	GeomPrum.	Váha
Influxdb	1,00	0,33	0,33	**0,48**	**14%**
Elastic	3,00	1,00	1,00	**1,44**	**43%**
OpenTSDB	3,00	1,00	1,00	**1,44**	**43%**
Celkem				**3,37**	**100%**

Tabulka 20: Výběr úložiště metrik – hodnocení kritéria škálování.

Hodnocení kritéria: Diskové nároky

Rešerší bylo nalezeno několik internetových článků zabývajících se srovnáním diskových nároků, ale nikdy se nepovedlo najít test, kde by došlo k porovnání všech databází současně. V článku „InfluxDB vs Elasticsearch for time series and metrics data“ (17) autor porovnával diskové nároky mezi InfluxDB a ElasticSearch a jako jednoznačný vítěz vzešel InfluxDB. Obdobně v jiné studii „InfluxDB Markedly Outperforms OpenTSDB in Time Series Data & Metrics Benchmark“ (18) byl porovnán InfluxDB s OpenTSDB, opět

s podobným výsledkem. Z hlediska diskového prostoru lze InfluxDB označit za velmi efektivní řešení, které využívá kompresi dat a může tak ušetřit až 90 % místa oproti jiným TSDB. Elasticsearch a OpenTSDB jsou na tom v tomto ohledu hůře, jelikož nejsou tak optimalizované pro ukládání velkého množství časových řad. Elasticsearch také vyžaduje větší množství paměti a diskového prostoru kvůli svému indexování a vyhledávání textových dat. Celkově lze vyvodit, že z hlediska diskového prostoru je InfluxDB nejefektivnějším řešením, zatímco Elasticsearch a OpenTSDB vyžadují více místa pro ukládání dat. V následujícím grafu jsou porovnány diskové nároky jednotlivých nástrojů:



Obrázek 15: Výběr úložiště metrik – porovnání spotřeby diskového prostoru

Vzájemným porovnáním vyšlo následující hodnocení:

Disk spotř.	InfluxDB	Elastic	OpenTSDB	GeomPrum.	Váha
InfluxDB	1,00	6,00	3,00	**2,62**	**67%**
Elastic	0,17	1,00	0,50	**0,44**	**11%**
OpenTSDB	0,33	2,00	1,00	**0,87**	**22%**
Celkem				**3,93**	**100%**

Tabulka 21: Výběr úložiště metrik – hodnocení kritéria spotřeba diskového prostoru

Hodnocení kritéria: Integrace

Při hodnocení kritéria integrace bylo nutné zvážit, jak dobře daný TSDB spolupracuje s dalšími nástroji a jak snadno lze tyto nástroje nainstalovat a konfigurovat, aby úložiště podporovaly. Zkoumáním nástrojů a oficiální dokumentace byly zjištěny následující možnosti integrace s nástroji pro dopravu logů a metrik a s nástroji pro sledování: Největší podporu mezi nástroji pro sběr logů a metrik mají databáze InfluxDB a Elasticserach, které jsou široce podporovány. OpenTSDB je také široce podporován, ale v některých nástrojích

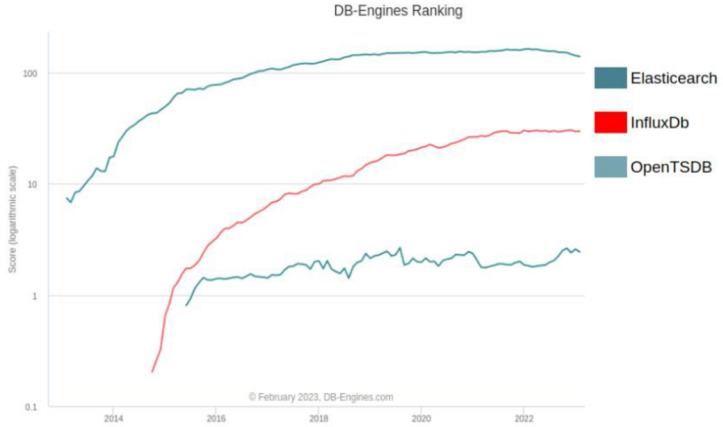
je nutné použít doplněk třetích stran. Obdobná podpora byla vysledována i u nástrojů pro vizualizaci. Po vzájemném porovnání vyšla následující tabulka hodnocení.

Integrace	Influxdb	Elastic	OpenTSDB	GeomPrum.	Váha
Influxdb	1,00	1,00	2,00	**1,26**	**40%**
Elastic	1,00	1,00	2,00	**1,26**	**40%**
OpenTSDB	0,50	0,50	1,00	**0,63**	**20%**
Celkem				**3,15**	**100%**

Tabulka 22: Výběr úložiště metrik – hodnocení kritéria integrace

Hodnocení kritéria: Podpora projektu

InfluxDB, Elasticsearch a OpenTSDB jsou všechny populární databáze časových řad. InfluxDB má velmi aktivní komunitu a je jedním z nejpopulárnějších nástrojů pro časové řady na trhu. Elasticsearch má také velmi aktivní komunitu a je široce používán pro fulltextové vyhledávání a analýzu dat. OpenTSDB má menší komunitu než InfluxDB a Elasticsearch, ale stále má poměrně aktivní uživatelskou základnu. Pro porovnání popularity byl nalezen kvalitní zdroj, který sleduje trendy v oblasti databází. Následující graf zobrazuje trendy popularity nástroje od roku 2013 (5):



Obrázek 16: Trend popularity vybraných TSDB systémů (5).

Podpora	Influxdb	Elastic	OpenTSDB	GeomPrum.	Váha
Influxdb	1,00	0,50	4,00	**1,26**	**33%**
Elastic	2,00	1,00	5,00	**2,15**	**57%**
OpenTSDB	0,25	0,20	1,00	**0,37**	**10%**
Celkem				**3,78**	**100%**

Tabulka 23: Výběr úložiště metrik – hodnocení kritéria podpora projektu

Celkové hodnocení a výběr nástroje

Hodnocení všech kritérií byla vložena do výsledné tabulky a po zohlednění dříve vypočtených vah vyšlo celkové hodnocení, znázorněné v následující tabulce:

Hodnocení	Integrace	Škálování	Disk. nároky	Podpora	Hodnocení
InfluxDB	0,40	0,14	0,67	0,33	**31,00%**
Elastic	0,40	0,43	0,11	0,57	**41,23%**
OpenTSDB	0,20	0,43	0,22	0,10	**27,77%**
Celkem	100,0%	100,0%	100,0%	100,0%	**100,00%**
Váha	29%	41%	11%	19%	

Tabulka 24: Výběr úložiště metrik – celkové hodnocení

Po sečtení všech hodnocení a zohlednění vah vzešel vítězně Elasticsearch. Je nutno dodat, že výsledek byl velmi těsný a při malé změně vah kritérií nebo preferencí by mohl výsledek dopadnout jinak. V prospěch databáze Elasticsearch však mluví i to, že je ve společnosti již používán pro protokolování aplikačních logů. To pak zjednodušuje celkovou správu úložiště i dotazování se nad daty. Vzhledem k modularitě řešení lze kdykoliv v budoucnu úložiště vyměnit za jiné.

4.5 Výběr nástroje pro sledování

V tomto kroku byl vybrán vhodný nástroj pro sledování metrik, jako jedna z komponent monitorovacího systému. Proces výběru je složen z následujících kroků:

- Volba kritérií a nastavení vah.
- Výběr nástrojů pro porovnání.
- Hodnocení variant a výběr nástroje

4.5.1 Volba kritérií a nastavení vah

Při volbě kritérií byly zohledněny požadavky na modularitu (P.4.1) a požadavky na funkcionalitu dashboardů (P.2), doplněné doporučeními expertů. Byla vybrána následující kritéria:

1. **Podpora úložišť**

Dobrý nástroj pro sledování a vizualizaci metrik by měl umožnit integraci s jinými nástroji a systémy, jako jsou například databáze, cloudové služby, síťové prvky, atd. Proto je významné zvážit podporu úložišť jako kritérium při výběru nástroje. Toto kritérium zohledňuje požadavek na modularitu.

2. **Dashboardy**

Toto kritérium zohledňuje požadavky na funkcionalitu dashboardů (P.2) a požadavky na automatizaci (P.3). UI by mělo být intuitivní a snadno použitelné. Nástroj by měl umožňovat snadnou konfiguraci a zobrazování dat. Měl by obsahovat ovládací prvky, které umožní uživatelům interagovat s daty, například filtrování dat, zvětšování a zmenšování grafů apod. Měla by být zohledněna i responzivita, která znamená schopnost přizpůsobit se různým zařízením a velikostem obrazovky. Je výhodné, aby uživatel mohl použít nástroj i na mobilních zařízeních nebo tabletech mimo kancelář. Je potřeba zhodnotit i schopnost automaticky generovat dashboardy pomocí API.

3. **Alerting**

Toto kritérium zohledňuje požadavky na funkcionalitu dashboardů (P.2) a požadavky na automatizaci (P.3). Zde jsou hodnoceny možnosti nastavení tresholdů, nastavení pravidel spuštění alertů a typy zaslaných alertů.

4. **Podpora projektu**

Komunita je důležitým faktorem při výběru monitorovacího nástroje pro sledování metrik a grafů. Komunita zahrnuje uživatele, vývojáře a odborníky v oblasti monitorování, kteří aktivně používají a přispívají k rozvoji daného monitorovacího nástroje. Velká a aktivní komunita znamená, že existuje mnoho uživatelů a vývojářů, kteří jsou ochotni sdílet své znalosti a zkušenosti, poskytovat podporu a přispívat k vývoji nástroje. Přítomnost detailní dokumentace a aktivní podpory jsou klíčové faktory pro uživatele, kteří potřebují rychle najít řešení problému nebo najít odpovědi na otázky. Aktualizace a vývoj nástroje jsou důležité faktory, které mohou být ovlivněny komunitou. Aktivní komunita může přinést časté aktualizace a vylepšení nástroje.

I zde byla použita Saatyho metoda párového porovnání pro volbu vah kritérií. Z porovnání vyšlo, že nejdůležitějším kritériem je kvalita v oblasti podpory grafů a dashboardů. Druhé kritérium v pořadí důležitosti je podpora úložišť z důvodu požadavku na modularitu.

Kriterium	Úložiště	Dashboard	Alerting	Komunita	GeomPrum.	Váha
Úložiště	1,00	0,50	3,00	2,00	**1,32**	**29%**
Dashboard	2,00	1,00	2,00	4,00	**2,00**	**44%**
Alerting	0,33	0,50	1,00	2,00	**0,76**	**17%**
Komunita	0,50	0,25	0,50	1,00	**0,50**	**11%**
Celkem					**4,58**	**100%**

Tabulka 25: Výběr nástroje pro sledování – nastavení vah kritérií.

4.5.2 Výběr nástrojů pro hodnocení

Rešerší článků na téma nástrojů pro vizualizaci metrik a monitoring bylo vyhodnoceno několik nástrojů, které byly vybrány pro porovnání. Užitečné byly zejména články: Best Open Source Application Monitoring Tools (19) a Top 11 Grafana Alternatives (20). Opět bylo nutné zohlednit pouze open-source nástroje a také pouze ty, které podporují úložiště metrik vybrané v přechozí kapitole. Pro porovnání byly zvoleny následující nástroje:

- **Grafana**

  Grafana je open-source projekt napsaný v jazyce Go, který byl vytvořen v roce 2014 společností Torkel Ödegaard. Projekt rychle získal popularitu mezi uživateli a vývojáři, což vedlo k vytvoření aktivní komunity kolem Grafany. Umožňuje uživatelům propojit se s různými datovými zdroji a vytvořit dashboardy a grafy pro vizualizaci a analýzu. Projekt je distribuován pod open-source licencí Apache 2.0, je tedy zdarma a k dispozici pro každého, kdo ho chce používat a přispívat k jeho rozvoji (9).

- **Kibana**

  Kibana je open source projekt, který byl vytvořen společností Elastic. Jeho hlavním účelem je vizualizace a analýza dat uložených v Elasticsearch, což je distribuovaný vyhledávací a analytický engine pro full-textové vyhledávání, analýzu a vizualizaci dat. Kibana byla poprvé uvedena v roce 2013 jako nástroj pro vizualizaci logových souborů a analýzu dat (21).

- **Cyclotron**

  Cyclotron je open-source projekt pro vizualizaci dat, který vznikl v roce 2016. Je psán v jazyce Python a využívá moderní webové technologie, jako je React. Projekt Cyclotron byl původně vyvíjen v rámci společnosti Postmates, ale v roce 2017 byl převeden pod organizaci Open Networking Foundation. Je licencován pod Apache

License 2.0, což znamená, že je možné projekt volně používat, upravovat a distribuovat. Cyclotron nabízí mnoho funkcí pro vizualizaci dat, jako jsou různé typy grafů, dashboardy, interaktivní prvky a integraci s různými datovými zdroji. Jeho modularita umožňuje uživatelům přidávat a upravovat funkcionality dle potřeb (22).

4.5.3 Hodnocení a výběr nástroje

V tomto kroku jsou nástroje ohodnoceny ve všech zmíněných kritériích. Pro hodnocení byla opět zvolena Saatyho metoda párového porovnání.

Hodnocení kritéria: Podpora úložišť

Zkoumáním oficiální dokumentace byly zjištěny následující informace ohledně podpory úložišť. Grafana, Kibana a Cyclotron jsou všechny vizualizační nástroje, které podporují integraci s různými TSDB, včetně ElasticSearch. V oblasti podpory jsou tyto nástroje podobné, ale existují určité rozdíly. Grafana má velmi silnou podporu pro řadu různých TSDB, včetně InfluxDB, Graphite, Prometheus, OpenTSDB a mnoha dalších. Kibana je bohužel primárně určena pro práci s ElasticSearch. Kibana však nabízí velmi silnou podporu pro analýzu dat, včetně možností agregace dat a vyhledávání v čase. Cyclotron je novější nástroj pro vizualizaci dat, který podporuje řadu TSDB, včetně InfluxDB, Graphite, Prometheus a dalších. Podpora jednotlivých databází je shrnuta v následující tabulce. Celkově lze vyhodnotit, že Grafana má největší podporu TSDB. Vzájemným porovnáním vzešlo výsledné hodnocení kritéria.

Úložiště	Grafana	Kibana	Cyclotron	GeomPrum.	Váha
Grafana	1,00	9,00	2,00	**2,62**	**60%**
Kibana	0,11	1,00	0,14	**0,25**	**6%**
Cyclotron	0,50	7,00	1,00	**1,52**	**35%**
Celkem				**4,39**	**100%**

Tabulka 26: Výběr nástroje pro sledování – hodnocení kritéria podpora úložišť

Hodnocení kritéria: Dashboardy

Zkoumáním dokumentace a zkoušením nástrojů byly zjištěny následující informace ohledně dashboardů. Všechny tři nástroje podporují tvorbu dashboardů, což je základní funkcionalita pro vizualizaci dat. Grafana a Kibana mají podporu pro zobrazení logů, zatímco Cyclotron tuto funkci neobsahuje. Všechny tři nástroje mají podporu pro API, ale liší se v použitém typu API. Grafana podporuje REST API, GraphQL a Simple JSON, Kibana REST API a Cyclotron GraphQL. Grafana a Kibana mají podporu pro nastavení thresholdů a alerting, zatímco Cyclotron tuto funkci neobsahuje. Podrobněji jsou jednotlivé funkcionality srovnány v následující tabulce:

Funkce	Grafana	Kibana	Cyclotron
Podpora API	Umožňuje volání API pro vytváření a správu dashboardů, datových zdrojů a dalších funkcí.	Nabízí REST API pro správu a manipulaci s daty a vizualizacemi.	Poskytuje omezené API pro manipulaci s daty a dashboardy.
Nastavení času v grafech	Nabízí rozsáhlé možnosti nastavení času v grafech, včetně možnosti zobrazit data v určitém rozmezí, rozdílných časových pásmech, apod.	Umožňuje nastavit časový filtr pro data v grafech a zobrazit je v různých časových rozsazích.	Podporuje nastavení časových filtrů v grafech, ale oproti Grafaně a Kibaně nabízí méně pokročilé možnosti nastavení času.
Filtrování	Umožňuje použít filtry pro omezení dat, které se zobrazí v grafech.	Nabízí silné funkce pro filtrace dat, včetně textového hledání a možností vizuálního omezení výsledků.	Podporuje filtraci dat pomocí textového hledání a filtrování na základě tagů.
Grupování	Podporuje seskupování dat podle různých kritérií, jako jsou např. zdroje dat, tagy apod.	Nabízí podobné funkce jako Grafana pro seskupování dat podle různých kritérií, jako jsou např. pole v datech.	Nabízí základní funkce pro seskupování dat, ale oproti Grafaně a Kibaně nabízí méně pokročilé možnosti.

Tabulka 27: Výběr nástroje pro sledování – porovnání funkcionality.

Grafana, Kibana a Cyclotron podporují podobný soubor typů grafů, jako jsou čárové, sloupcové, bodové, plošné, sekvenční, histogramové a rozptylové grafy. Grafana a Kibana navíc podporují některé specifické typy grafů, jako jsou gaugy a heatmapy. Cyclotron podporuje pouze základní typy grafů a neobsahuje funkce pro specifické účely, jako jsou mapové a kruhové grafy.

Dashboard	Grafana	Kibana	Cyclotron	GeomPrum.	Váha
Grafana	1,00	4,00	5,00	**2,71**	**69%**
Kibana	0,25	1,00	1,00	**0,63**	**16%**
Cyclotron	0,20	1,00	1,00	**0,58**	**15%**
Celkem				**3,93**	**100%**

Tabulka 28: Výběr nástroje pro sledování – hodnocení funkcionality dashboardů

Hodnocení kritéria: Alerting

Zkoumáním dokumentace a zkoušením nástrojů byly zjištěny následující informace ohledně alertingu. Grafana a Kibana mají velmi podobné možnosti nastavení thresholdu, ale Grafana má grafické rozhraní pro snadnější nastavení pravidel. Kibana umožňuje vytváření pravidel pro sledování logů, což je oblast, kterou Grafana a Cyclotron nepodporují nativně. Všechny tři nástroje umožňují odesílání upozornění e-mailem a podporují Webhooky pro další služby. Grafana má však více možností odesílání upozornění, jako jsou Slack, PagerDuty a další, což ji činí více flexibilní.

Funkce	Grafana	Kibana	Cyclotron
Nastavení thresholdu	Ano, pro každou metriku	Ano, pro každou metriku	Ano, pro každou metriku
Možnosti nastavení alertingu	Ano, včetně grafického nastavení vytváření pravidel	Ano, včetně vytváření pravidel pro sledování logů	Ano, včetně vytváření pravidel pro sledování metrik
Možnosti odeslání upozornění	Ano, e-mailem, Slack, PagerDuty, Webhooks a dalšími službami	Ano, e-mailem, Logstash, Elasticsearch a dalšími službami	Ano, e-mailem a dalšími službami pomocí Webhooků

Tabulka 29: Výběr nástroje pro sledování – přehled možností alertingu.

Celkově lze vyhodnotit, že Grafana má nejlepší podporu alertingu. Vzájemným porovnáním vzešlo výsledné hodnocení kritéria alerting.

Alerting	Grafana	Kibana	Cyclotron	GeomPrum.	Váha
Grafana	1,00	2,00	4,00	**2,00**	**55%**
Kibana	0,50	1,00	4,00	**1,26**	**34%**
Cyclotron	0,25	0,25	1,00	**0,40**	**11%**
Celkem				**3,66**	**100%**

Tabulka 30: Výběr nástroje pro sledování – vyhodnocení kritéria alertingu

Hodnocení kritéria: Podpora projektu

V této studii došlo ke snaze změřit oblíbenost projektu z dostupných zdrojů. Popularitu a velikost komunity nelze přesně změřit. Lze ji ale odhadnout například podle počtu stažení z Docker hub, počtu sledujících nebo přispěvatelů na Github, nebo podle počtu článků na StackOverflow. Z následující tabulky lze odvodit, že největší komunita je kolem nástroje Grafana, nicméně život kolem Kibany je téměř srovnatelný. Cyclotron znatelně zaostává a je na první pohled vidět, že se v tuto chvíli jedná o řádově méně populární projekt.

Zdroj	Grafana	Kibana	Cyclotron
hub.docker.com/	1B+	100M+	100K+
stackoverflow.com	13200	18363	120
Github forks	10545	7612	118
Github stars	54117	18272	1567
Github open isssues	3114	10075	41

Tabulka 31: Výběr nástroje pro sledování – přehled popularity projektů

Grafana má velkou komunitu a je velmi oblíbená díky svým výkonným nástrojům pro vizualizaci dat. Tento projekt je aktivně podporován a vyvíjen open source komunitou. Kibana má také velkou komunitu a je oblíbená zejména mezi uživateli Elasticsearch, na kterém je založena. Projekt je také aktivně podporován open source komunitou, ale je poněkud méně populární než Grafana. Cyclotron má omezenou podporu projektu a menší komunitu, což může mít vliv na jeho vývoj a udržitelnost v dlouhodobém horizontu. Vzájemným porovnáním pak vyšla tabulka s výslednými vahami pro kritérium podpora projektu.

Podpora	Grafana	Kibana	Cyclotron	GeomPrum.	Váha
Grafana	1,00	2,00	9,00	**2,62**	**60%**
Kibana	0,50	1,00	7,00	**1,52**	**35%**
Cyclotron	0,11	0,14	1,00	**0,25**	**6%**
Celkem				**4,39**	**100%**

Tabulka 32: Výběr nástroje pro sledování – vyhodnocení kritéria komunita.

Celkové hodnocení

Hodnocení všech kritérií byla vložena do výsledné tabulky a po zohlednění dříve vypočtených vah vyšlo celkové hodnocení, znázorněné v následující tabulce:

Nástroj	Úložiště	Dashboard-	Alerting	Podpora	Hodnocení
Grafana	0,60	0,69	0,55	0,60	**62,97%**
Kibana	0,06	0,16	0,34	0,35	**18,15%**
Cyclotron	0,35	0,15	0,11	0,06	**18,88%**
Celkem	100,0%	100,0%	100,0%	100,0%	**100,00%**
Váha	29%	44%	17%	11%	

Tabulka 33: Výběr nástroje pro sledování – celkové hodnocení.

Vzájemným porovnáním vzešel jako vítězný nástroj Grafana. Grafana nabízí mnoho možností integrace s různými datovými zdroji, jako jsou například Prometheus, InfluxDB nebo Elasticsearch. Kibana a Cyclotron mají také určitou míru integrace, ale nejsou tak flexibilní jako Grafana. Dalším důvodem je schopnost Grafany vytvářet vysokokvalitní grafy. Grafana nabízí mnoho různých typů grafů a vizualizací, které mohou být použity pro různé účely. Kibana a Cyclotron nabízejí také několik typů grafů, ale nabídky jsou omezené ve srovnání s Grafanou. Grafana také nabízí pokročilý alerting s možností nastavení různých typů podmínek a akcí, jako je posílání e-mailů. Kibana a Cyclotron nabízejí také alerting, ale jejich funkce jsou omezené. Grafana má velkou a aktivní komunitu, což znamená, že existuje mnoho připravených pluginů, šablon a návodů, které mohou uživatelé použít pro své projekty. Cyclotron má v porovnání s Grafanou a Kibanou podstatně menší komunitu a je tu hrozba, že se projekt nebude vyvíjet.

4.6 Nastavení a spuštění nástrojů

V této fázi řešení došlo k nakonfigurování nástrojů, jejich propojení a spuštění pomocí technologie Docker. Aby bylo splněn technický požadavek na spuštění v různých prostředích (P.4.3), tedy na různých serverech a různých verzích OS, převážně Debian a Ubuntu, je nutné využít technologii Docker, která se již ve společnosti využívá. Docker je open-source projekt pro kontejnerizaci aplikací. To znamená, že umožňuje vytvářet a spouštět aplikace v izolovaném prostředí, které je snadno přenosné a konzistentní napříč různými prostředími, jako jsou lokální vývojové počítače, cloudové služby nebo serverové farmy (23). Samotné řešení se skládá z následujících kroků:

- Nastavení nástroje pro sběr logů a metrik
- Nastavení úložiště metrik
- Nastavení nástroje pro sledování
- Orchestrace a spuštění clusteru

4.6.1 Konfigurace a propojení nástrojů

V tomto kroku došlo k nastavění nástroje pro sběr logů a metrik. Pro řešení byl vybrán nástroj Fluentd. Webové aplikace odesílají měření do souboru ve formátu JSON záznamů. Je tedy nutné zajistit spuštění Fluentd démona, a správně jej nakonfigurovat. Celé řešení je kontejnerizováno pomocí technologie Docker. Na veřejném serveru hub.docker.com je k dispozici oficiální image pro bezplatné použití. Stačí tedy správně nakonfigurovat docker-compose.yml soubor:

```
version: "3"

services:
  qmon-fluentd:
    build: ./
    volumes:
      - ./fluent.conf:/fluentd/etc/fluent.conf
      - ../../logs:/logs
    restart: always
    container_name: qmon-fluentd
    user: root
    ports:
      - "24224:24224"
      - "24224:24224/udp"
    networks:
      - qmon
```

Obrázek 17: Nastavení nástroje pro sběr logů a metrik – nastavení kontejneru fluentd

Je nutné doinstalovat podporu pro Elasticsearch. Toho se docílí pomocí patřičného příkazu RUN v Dockerfile souboru.

```
FROM fluent/fluentd:v1.14.0-debian-1.0

USER root

RUN ["gem", "install", "fluent-plugin-elasticsearch", "--no-document", "
    --version", "5.0.1"]

USER fluent;
```

Obrázek 18: Implementace sběru logů a metrik – nastavení Dockerfile souboru

V posledním kroku je nutné nastavit konfiguraci nástroje Fluentd. V sekci source se nastavuje příjem logů. V tomto řešení je zvolena metoda čtení ze souboru, který obsahuje řádky jednotlivých logů ve formátu JSON. Pomocí příkazu `match` lze vyfiltrovat konkrétní logy a směrovat je na výstupy. Výstup se definuje pomocí vnořeného uzlu `store`, kde je třeba nastavit typ úložiště, adresu a doplňkové parametry včetně nastavení maximálního intervalu ukládání logů pomocí parametru `flush_interval`. V tomto případě je hodnota nastavena na 10s, to znamená že může dojít maximálně k 10s zpoždění dat.

```
<source>
  @type tail
  path /logs/example.log*
  pos_file /logs/example.pos
  tag qmon.example
  read_from_head true
  <parse>
    @type json
  </parse>
</source>

# Store Data in Elasticsearch
<match *.**>
  @type copy
  <store>
    @type elasticsearch
    host qmon-elasticsearch
    port 9200
    include_tag_key true
    tag_key @log_name
    logstash_format true
    flush_interval 10s
    <buffer>
      @type file
      path /var/log/fluentd/buffer/qmon.example
      flush_thread_count 2
      flush_interval 10s
      chunk_limit_size 10m
      queue_limit_length 1000
      overflow_action block
    </buffer>
  </store>
</match>
```

Obrázek 19: Nastavení nástroje pro sběr logů a metrik – konfigurace služby

4.6.2 Nastavení úložiště metrik

V tomto kroku je nejprve je nutné zprovoznit službu Elasticsearch pomocí Docker kontejneru. I zde je k dispozici přednastavený oficiální Docker image. Je třeba vytvořit složku, do které se budou data ukládat a správně nastavit cestu v sekci `volumes`. Konfigurace v souboru docker-compose.yml vypadá následovně:

```
version: "3"
services:
  qmon-elasticsearch:
    container_name: qmon-elasticsearch
    image: elasticsearch:8.6.1
    environment:
      - discovery.type=single-node
      - ES_JAVA_OPTS=-Xms1g -Xmx1g
      - xpack.security.enabled=false
    volumes:
      - ./es-data:/usr/share/elasticsearch/data
    ports:
      - target: 9200
        published: 9200
    networks:
      - qmon
```

Obrázek 20: Nastavení úložiště metrik – nastavení kontejneru Elasticsearch

Současně s Elasticsearch se běžně instaluje nástroj Kibana, který je součástí projeketu Elastic a je určen k prohlížení a správě datových indexů. Je nutné nastavit síťovou adresu ke službě Elasticsearch včetně portu. Konfigurace v souboru docker-compose.yml vypadá následovně:

```
qmon-kibana:
  container_name: qmon-kibana
  image: kibana:8.6.1
  environment:
    - ELASTICSEARCH_HOSTS=http://qmon-elasticsearch:9200
  ports:
    - target: 5601
      published: 5602
  depends_on:
    - qmon-elasticsearch
  networks:
    - qmon
```

Obrázek 21: Nastavení úložiště metrik – nastavení kontejneru Kibana

4.6.3 Nastavení nástroje pro sledování

Posledním nástrojem, který je nutné nastavit je Grafana. I Grafana má oficiální image na serveru hub.docker.com. Při spuštění kontejneru je nutné nastavit složku kam si Grafana ukládá data. Konfigurace v docker-compose vypadá následovně.

```
version: '3'
services:
  qmon-grafana:
    image: grafana/grafana:latest
    container_name: qmon-grafana
    user: root
    volumes:
      - ./data:/var/lib/grafana
    ports:
      - "3232:3000"
    restart: always
    networks:
      - qmon
```

Obrázek 22: Nastavení nástroje pro sledování – nastavení kontejneru Grafana

4.6.4 Orchestrace clusteru

Celý systém je orchestrován pomocí nástroje Docker compose, který je nabízen jako základní řešení pro orchestraci Docker kontejnerů. Jedná se o nástroj, který umožňuje definovat a spouštět složitější aplikace skládající se z více Docker kontejnerů. Docker Compose definuje jednotlivé kontejnery a jejich závislosti v YAML souboru, který lze snadno sdílet a verzovat v nástrojích pro správu kódu. Všechny konfigurace lze spojit do jednoho souboru a spouštět společně jedním príkazem docker-compose up.

```
docker-compose up -d
Starting qmon-elasticsearch ... done
Starting qmon-kibana        ... done
Starting qmon-fluentd       ... done
Starting qmon-grafana       ... done
```

Ověření správného spuštění kontejnerů lze jednoduše pomocí příkazu `docker ps`.

```
docker ps --format "table {{.ID}}\t{{.Image}}\t{{.Names}}\t{{.State}}" | grep qmon
4398a87fa96f   fluentd_qmon-fluentd       qmon-fluentd            running
0954126157e3   grafana/grafana:latest     qmon-grafana            running
8f4ec0d7dbb6   kibana:8.6.1               qmon-kibana             running
0761fb52d9fd   elasticsearch:8.6.1        qmon-elasticsearch      running
```

Pro orchestraci Docker kontejnerů v distribuovaném prostředí jsou vhodné nástroje, které umožňují správu a řízení aplikací běžících na velkých počítačových clusterech.

Společnost již využívá technologii Kubernetes. Kubernetes je jedním z nejpopulárnějších nástrojů pro orchestraci kontejnerů v distribuovaném prostředí. Kubernetes poskytuje vysokou dostupnost, škálovatelnost a odolnost proti chybám a umožňuje správu velkého počtu kontejnerů. V této práci nebudou kontejnery orchestrovány promocí této technologie, ale nastavení je velmi podobné docker-compose a lze se jím následně inspirovat (24).

Nakonec je potřeba ověřit, že celý cluster je správně nastaven a služby spolu komunikují. Pro tyto účely postačí poslat ve správném formátu data na vstup nástroje Fluentd. Nástroj očekává soubor se záznamy ve formátu JSON na nastaveném místě. Pro otestování funkčnosti byl založen soubor s několika záznamy v požadovaném formátu. Ověření správného uložení záznamu do Elasticsearch indexu lze ověřit v logu služby Elasticsearch:

```
docker logs -f qmon-elasticsearch
qmon-elasticsearch    | {"@timestamp":"2023-03-18T08:46:24.948Z", "log.level": "INFO",
"message":"[logstash-2023.03.18/nMSEjm1RS5qpHYSiA1QKlA] update_mapping [_doc]",
"ecs.version":
"1.2.0","service.name":"ES_ECS","event.dataset":"elasticsearch.server","process.thread.na
me":"elasticsearch[0761fb52d9fd][masterService#updateTask][T#1]","log.logger":"org.elasti
csearch.cluster.metadata.MetadataMappingService","elasticsearch.cluster.uuid":"wei6HZblRo
iIX_brFnSfeA","elasticsearch.node.id":"GNXnOs_cRVubwTOajPeHNw","elasticsearch.node.name":
"0761fb52d9fd","elasticsearch.cluster.name":"docker-cluster"}
```

V tomto kroku tedy došlo k nakonfigurování komponent monitorovacího systému a spuštění pomoví technologie Docker. Bylo ověřeno, že transportní vrstva správně přijímá data a přeposílá je do úložiště metrik. Je tedy možno přejít k samotné implementaci měření ve webové aplikaci.

4.7 Implementace měření

V této fázi řešení došlo k implementaci měření. Výsledkem je knihovna psaná pro aplikace psané v jazyku TypeScript a umožňuje provádět měření a výsledky odesílat na výstup. Jsou zohledněny požadavky na měření (P.1). Implementace měření se skládá z následujících kroků:

- Implementace struktury dat měření
- Implementace sběru metrik a transportu
- Implementace měření bloku kódu
- Implementace měření zdrojů

4.7.1 Implementace struktury dat měření

V prvním kroku je nutné stanovit strukturu dat měření. Zohledněny jsou všechny požadavky na měření. Zkoumáním různých formátů a konzultací s experty vzešla výsledná struktura dat. Řešení je inspirované formátem měření v InfluxDB a je tedy s ním kompatibilní. Atributy datového záznamy jsou:

Atribut	Popis atributu
Name	Název měření
Fields	Pole metrik typu klíč-hodnota. Klíč je textový název metriky. Hodnota metriky je číselná.
Tags	Pole tagů typu klíč-hodnota. Klíč je textový název tagu. Hodnota tagu je textová.
Timestamp	Časové razítko vzniku měření.

Tabulka 34: Implementace struktury dat měření – atributy měření.

Následuje implementace datového typu v kódu TypeScript. Je nutné definovat správně všechny typy atributů.

```
export type MeasurementTags = Record<string, string>;
export type MeasurementFields = Record<string, number>;

export type Measurement = {
  name: string;
  tags: MeasurementTags;
  fields: MeasurementFields;
  timestamp: Date;
};
```

Obrázek 23: Implementace struktury dat měření – datový typ v TypeScript

Po naimplementování datového typu je možno přejít k dalšímu kroku, konkrétně k implementaci sběru měření a jejich transportu.

4.7.2 Implementace sběru metrik a transportu

Aby bylo možno jednotlivá měření v aplikaci sbírat a předávat transportní vrstvě, které se postará o odeslání dat na správné místo ve správný čas. K tomuto účelu je naimplementována třída MeasurementCollector.

```
import { Measurement, MeasurementTags } from './Measurement';
import { SystemMeasurement } from './SystemMeasurement';
import { Transaction } from './Transaction';
import { Transport } from './Transport';

export class MeasurementCollector {

    private transport: Transport;

    constructor(transport: Transport) {
        this.transport = transport;
    }

    add(measurement: Measurement) {
        this.transport.write(measurement);
    }

    startTransaction(name: string, tags?: MeasurementTags): Transaction {
        return new Transaction(this, name, tags);
    };

    sytemMeasurement(): SystemMeasurement {
        return new SystemMeasurement(this);
    };

}
```

Obrázek 24: Implementace sběru metrik a transportu – MeasurementCollector

Nástroj Fluentd očekává data ve formátu JSON na určitém místě, dle konfigurace v předchozí kapitole. Je tedy nutné naimplementovat objekt transport, který přijímá instanci MeasurementCollector v konstruktoru. Pro zápis záznamů ve formátu JSON je využita knihovna Winston, která je jistým standardem pro zápis logů.

```
import { Measurement } from './Measurement';

export interface Transport {
    write: (metric: Measurement) => void;
}
```

Obrázek 25: Implementace sběru metrik a transportu – Transport

Výstup je v tomto případě ukládán na disk ve formátu JSON do souboru, který je následně zpracováván transportní vrstvou. Záznam měření vypadá následovně:

Obrázek 26: Implementace sběru logů a metrik – výstup měření

Po implementaci tříd pro sběr měření a transport je možno přejít k dalšímu kroku a to měření bloku kódu a měření zdrojů.

4.7.3 Implementace měření bloku kódu

V tomto kroku je zohledněn požadavek na měření bloku kódu (P.1.1). Pro účely měření bloku kódu v aplikaci je třeba vytvořit funkcionalitu, která na určitém místě v kódu spustí začátek měření a na určitém místě měření ukončí. Primární metrikou je zde doba trvání, po kterou se kód vykonával.

Transakce a spany

V tomto ohledu se lze inspirovat řešením v komerčním nástroji Application Performance Monitoring (APM) od firmy Elastic. APM knihovna definuje tzv. spany a transakce. Span jsou měření, které se zaměřuje na dobu, kterou aplikace tráví v určitých činnostech. Tyto informace jsou důležité pro optimalizaci výkonu aplikace a odhalení případných slabých míst. Transakce je pak specifickým typem spanu, a lze si ji představit jako nejvyšší úroveň práce, která je v aplikaci měřena. Typickou transakcí je měření doby zpracování http requestu (25).

Pro účely práce byla naimplementována jednoduchá třída Transaction, která v této základní podobě umí měřit čas trvání kódu. V budoucnu se může obohacovat o další funkcionalitu, včetně dělení na spany.

```
import { Measurement, MeasurementTags } from './Measurement';
import { MeasurementCollector } from './MeasurementCollector';

export class Transaction {

    private collector: MeasurementCollector;
    readonly start: number;
    readonly data: Measurement;

    constructor(collector: MeasurementCollector, name: string, tags?: MeasurementTags ) {
        this.collector = collector;
        this.start = performance.now();
        this.data = {
            name,
            tags: tags ?? {},
            fields: {},
            timestamp: new Date(),
        };
    }

    commit() {
        this.data.fields.duration = performance.now() - this.start;
        this.collector.add(this.data);
    }

}
```

Obrázek 27: Implementace sběru logů a metrik – implementace třídy Transaction

Samotné měření doby vykonání http requestu pak může vypadat následovně.

```
const transaction = measurementCollector
    .startTransaction('request', {
        request: '/data/person/find',
        method: 'GET',
    });

// tělo kódu

transaction.commit();
```

Obrázek 28: Implementace sběru logů a metrik - měření requestu

4.7.4 Implementace měření využití zdrojů

V tomto úkolu je řešen požadavek na měření systémových statistik (P.1.2) využití paměti a cpu. V teoretické části práce je monitoringu systémových zdrojů věnována vysoká důležitost. Sledovat systémové statistiky v kontextu webové služby je klíčové. Vytížení procesoru má za následek celkovou degradaci služby. Důležité je i sledovat paměť zejména z důvodu tzv. memory leaků. Pokud si aplikace nečistí paměť, postupně se paměť vyčerpá

až dojde k úplnému zahlcení a kolapsu systému. Pro účely sledování systémových informací procesu byla využita populární npm knihovna `pidusage`. Výstup systémových statistik má následující podobu:

```
{
  cpu: 50,
  memory: 73232384,
  ctime: 13550,
  elapsed: 36070,
  timestamp: 1679213498312,
  pid: 2236003,
  ppid: 2211551
}
```

V této implementační fázi došlo k vytvoření knihovny v jazyce TypeScript, která umožňuje měření a zohledňuje všechny požadavků na měření. Naměřené záznamy jsou předány nástroji pro sběr logů a metrik ve formátu JSON, a ten je dopravuje do úložiště metrik Elasticsearch. Následuje fáze implementace knihovny pro tvorbu dashboardů v nástroji Grafana.

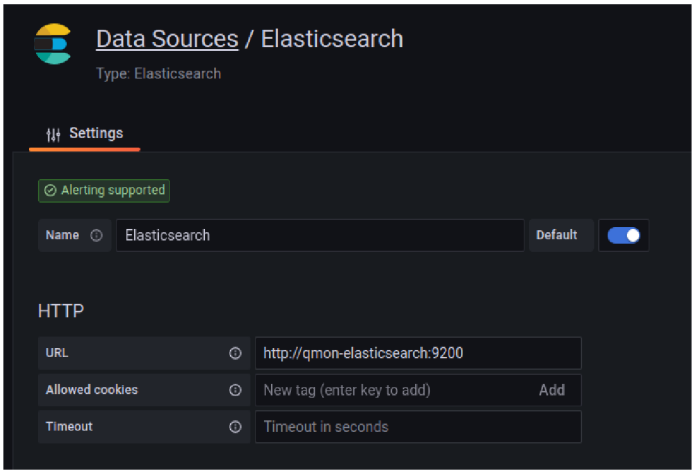
4.8 Implementace dashboardů

V této fázi řešení došlo k základnímu nastavení nástroje Grafana, vytvoření dashboardu se všemi požadovanými prvky a následné implementaci knihovny v jazyce TypeScript, která umožňuje automatické vytváření dashboardů. Byly zohledněny požadavky na funkcionalitu dashboardů (P.2) a požadavky na automatizaci (P.3). Implementace dashboardů se skládá z následujících kroků:

- Nastavení prostředí v nástroji Grafana
- Vytvoření dashboardu
- Implementace knihovny pro dashboardy

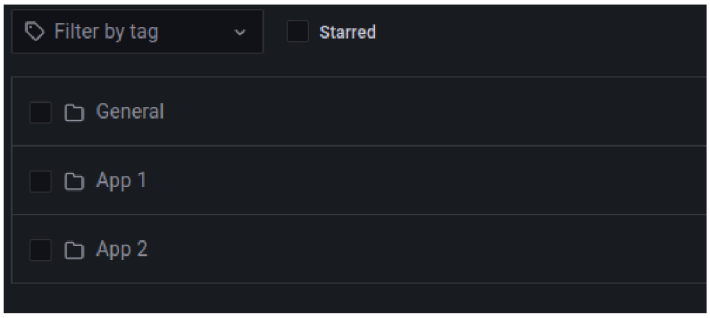
4.8.1 Nastavení prostředí v nástroji Grafana

Samotné implementaci předchází nutné nastavení nástroje Grafana. V prvním kroku je nutné Grafanu napojit na úložiště metrik Elasticsearch. To se nastavuje v sekci `Configuration/Data sources`. Je tedy nutné nastavit nový datový zdroj, jeho adresu a šablonu indexu, který načítá.



Obrázek 29: Nastavení prostředí v nástroji Grafana - nastavení datového zdroje.

Grafana umožňuje organizovat dashboardy do adresářů. Jak organizovat složky je čistě na vývojářích projektu. Dle doporučení z oddílu v teoretické části „Vizualizace“, by měla každá složka představovat aplikaci nebo službu, případně se věnovat určitému aspektu.



Obrázek 30: Nastavení prostředí v nástroji Grafana - organizace do složek.

4.8.2 Vytvoření dashboardu

V tomto kroku došlo k samotné tvorbě dashboardu pro webovou službu. Byly zohledněny požadavky na funkcionalitu dashboardů (P.2) konkrétně požadavek na zobrazení metriky v čase (P.2.1), požadavek na agregované zobrazení metriky v čase (P.2.2) a požadavek na zobrazení aktuální hodnoty (P.2.3). Ve složce lze vytvořit libovolné množství dashboardů. Dashboard nabízí možnosti rozmístit různé komponenty, které se dotazují do úložiště a vizualizují metriky. Prvky lze oddělovat do řádků, které lze sbalit a rozbalit. Pro účely práce postačí jednoduché zobrazení a umístění jednotlivých grafů typu Time series.

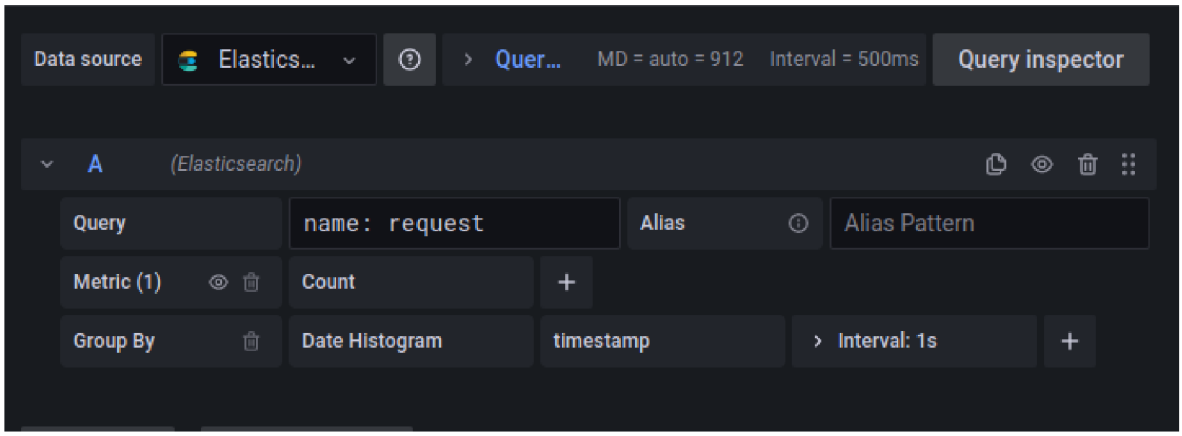
Zobrazení metriky v čase

Nástroj Grafana poskytuje bohaté možnosti nastavení grafů, úpravy dotazu a varianty zobrazení. Pro požadavek zobrazení metriky v čase (P.2.1) je využit graf typu Time series. Uživatelské prostředí umožňuje detailně nastavit dotaz, aniž by bylo nutné znát dotazovací jakyk Elasticsearch. V dotazu je třeba nastavit tyto základní údaje:

Atribut	Popis
Zdroj	Datový zdroj (zde Elasticsearch).
Query	Omezující pravidlo. Je třeba zadat název měření například name: request.
Metric	Metrika je položka z pole fields, které bylo nastaveno v oddíle „Implementace měření“. Na jednom grafu lze zobrazit více metrik současně.
Group by	Primárně se seskupují statistiky podle časového atributu *timestamp*, který je povinný. Defaultní hodnota časového intervalu je *auto*. Je možné doplňkově přidat další agregační pravidlo podle jiného atributu.

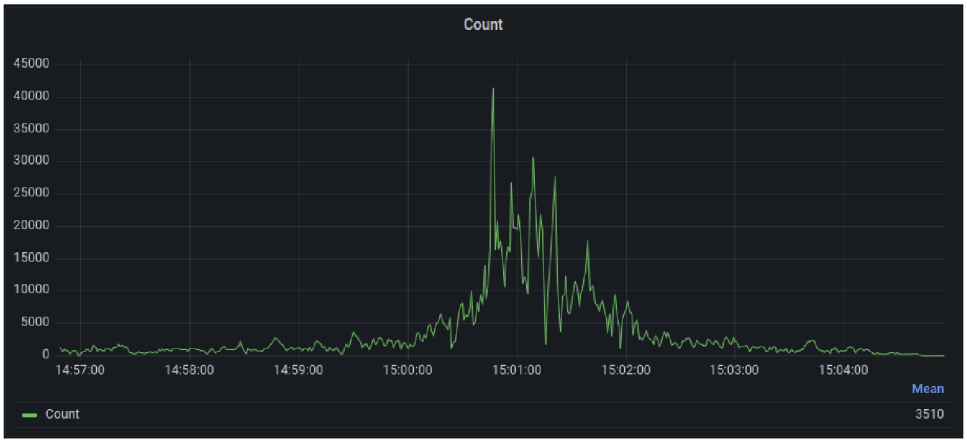
Tabulka 35: Zobrazení metriky v čase – Query

Nastavení dotazu pro získání počtu requestů znázorňuje obrázek



Obrázek 31: Zobrazení metriky v čase – nastavení dotazu v Grafaně

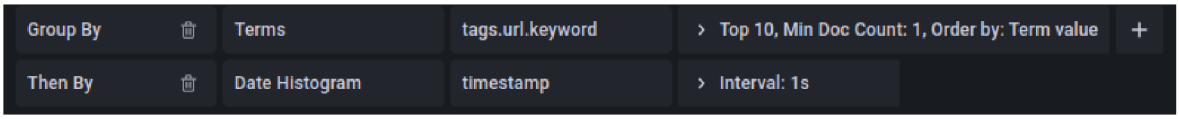
Výsledný graf se vzorovými daty je zachycen na následujícím obrázku.



Obrázek 32: Vytvoření dashboardu – ukázka grafu.

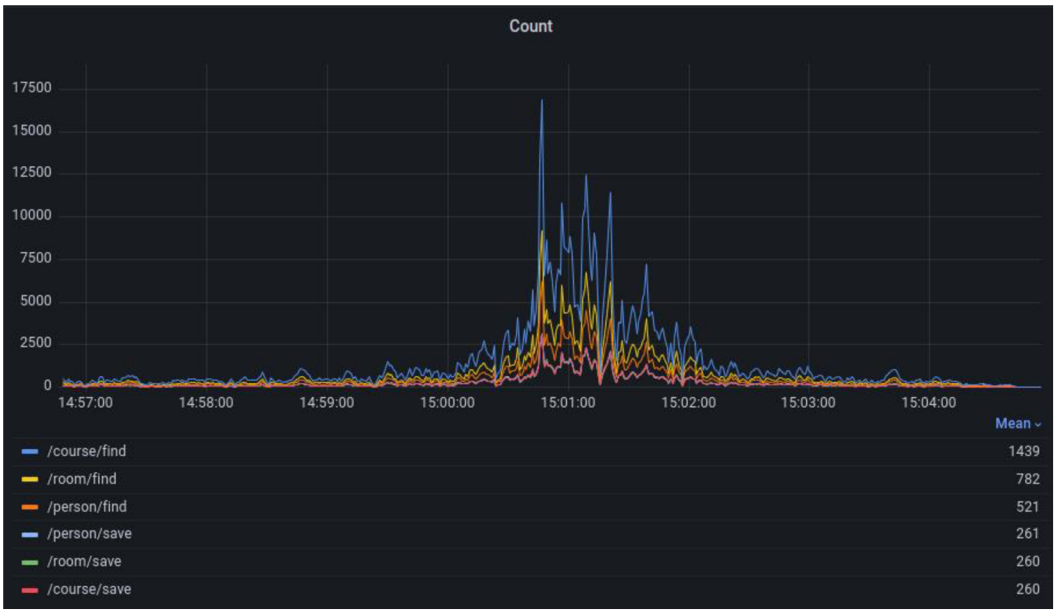
Agregované zobrazení metriky

Dalším požadavkem na funkcionalitu dashboardů je požadavek na agregované zobrazení metriky (P.2.2). Tento způsob zobrazení je velmi užitečný při detailním hledání problému. Pokud například je sledována metrika počtu requestů za vteřinu, je výhodné sledovat spíše jednotlivé typy requestů podle url adresy. V případě, že je konkrétní typ requestu problematický, je snadné ho takto rozpoznat. Pro agregaci hodnot Grafana umožňuje funkci `Group by`. Defaultně je dotaz seskupen podle časového razítka na intervaly. Lze ale dodat jeden nebo více dodatečných pravidel pro agregaci podle určitého atributu. V tomto příkladě je výstup dodatečně agregován podle atributu `tags.url`.



Obrázek 33: Vytvoření dashboardu – nastavení agregace.

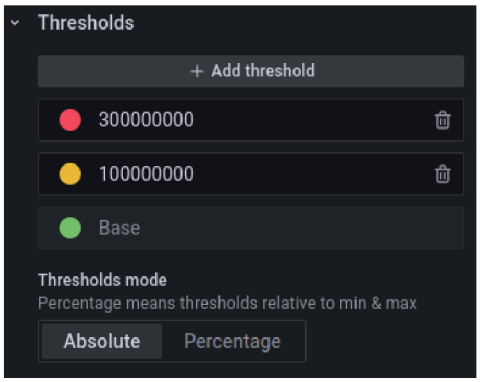
Následující obrázek zobrazuje výsledný graf na stejných vzorových datech. Je patrný rozdíl v hodnotě zobrazených informací. Lze rozeznat, který konkrétní http request je nejvíce zatěžován. Graf umožňuje zobrazit i dodatečnou tabulku s nejvíce zatěžovanými adresami.



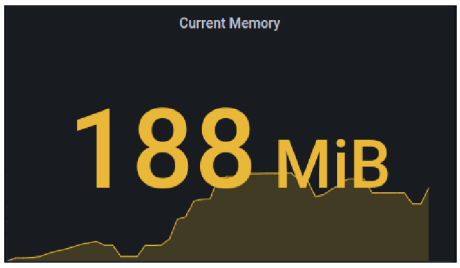
Obrázek 34: Vytvoření dashboardu – ukázka grafu s agregací.

Aktuální hodnota metriky

Posledním požadavkem je zobrazení aktuální hodnoty metriky (P.2.3). Grafana i zde poskytuje komponentu pro tyto účely. Stačí vybrat typ zobrazení na `stat`. Dotaz je stejný jako v předešlém případě, pouze je nutné počítat, že se zobrazuje pouze jedno vybrané číslo z hodnot vybraného časového intervalu. Defaultně je nastavena volba `last`, ale je možné zvolit i jinou možnost, například `max`, `min`, `first`. Posledním požadavkem byla možnost nastavení prahových hodnot. I zde Grafana poskytuje jednoduché rozhraní pro tento požadavek. Výsledné zobrazení je pak znázorněno níže.



Obrázek 35: Vytvoření dashboardu – nastavení prahových hodnot



Obrázek 36: Vytvoření dashboardu – aktuální hodnota metriky

V tomto kroku je hotový dashboard se všemi požadavky na funkcionalitu. Nyní zbývá poslední krok a tím automatické generování dashboardů.

4.8.3 Automatické vytváření dashboardů

Jeden z požadavků je na automatizaci je automatické generování grafů (P.3.1). Bylo zvoleno řešení, že součástí knihovny projektu bude složka s konfiguracemi dashboardů, které se v pravý okamžik synchronizují s nástrojem pro sledování, a grafy se automaticky vygenerují. Z dokumentace Grafany bylo zjištěno, že k tomuto účelu lze použít Rest API, pomocí kterého lze plnohodnotně administrovat nástroj a vytvářet tedy i dashboardy. Nejprve je však nutné získat API klíč, který je nutné vygenerovat v Grafaně v sekci Api keys. Dotaz na API pak vypadá následovně:

```
GET /api/dashboards/uid/cIBgcSjkk HTTP/1.1
Accept: application/json
Content-Type: application/json
Authorization: Bearer eyJrIjoiT0tTcG1pUlY2RnVKZTFVaDFsNFZXdE9ZWmNrMkZYbk

Response:
HTTP/1.1 200
Content-Type: application/json

{
  "dashboard": {
    "id": 1,
    "uid": "cIBgcSjkk",
    "title": "Production Overview",
    "tags": [ "templated" ],
    "timezone": "browser",
    "schemaVersion": 16,
    "version": 0
  },
  "meta": {
    "isStarred": false,
    "url": "/d/cIBgcSjkk/production-overview",
    "folderId": 2,
    "folderUid": "l3KqBxCMz",
    "slug": "production-overview"
  }
}
```

JSON záznam dashboardu je objemný a velmi strukturovaný, proto zde není zobrazen. Lze však takto nastavit libovolný dashboard obsahující jakýkoliv graf či prvek. Psát ručně

takto složité konfigurace však je příliš pracné. Většinou se prvky na dashboardech opakují a jediné co je nutné konfigurovat jsou klíčové parametry jako název měření, metrika, jednotka, prahové hodnoty, případně pravidlo pro agregaci. Ostatní nastavení lze oddělit v podobě šablony.

Proto bylo vytvořeno řešení, kde podstatné informace týkající se grafu jsou vyčleněny do zjednodušené konfigurace. Následně jsou tyto informace spojeny s šablonou a je vygenerován kompletní JSON záznam, který je následně odeslán na příslušnou http adresu Api Grafany. Konfigurace grafu pak vypadá jednoduše a přehledně, a pro vývojáře je snadná na implementaci.

```
{
    type: 'timeseries',
    title: 'Requests Count',
    datasource,
    query: {
        name: 'request',
    },
    metrics: [{
        type: 'count',
    }, ],
    groupBy: {
        field: 'tags.url.keyword',
    },
}
```

Obrázek 37: Automatické vytváření dashboardů – zjednodušená konfigurace grafu

Pro účely generování dashboardů pomocí zjednodušené konfigurace a Rest API byla implementována knihovna v jazyce TypeScript. Zde je zde zobrazena pouze hlavní třída, která přijímá konfigurace a skrze connector odesílá výsledný dashboard v JSON formátu.

```
import { Connector } from './connector';
import { Dashboard, generateDashboard } from './dashboard';

export class Grafana {

    private connector: Connector;

    constructor(connector: Connector) {
        this.connector = connector;
    }

    async registerDashboards(dashboards: Dashboard[]): Promise<void> {
        await Promise.all(dashboards.map(dashboard => this.registerDashboard(dashboard)));
    }

    async registerDashboard(dashboard: Dashboard): Promise<void> {
        const postData = generateDashboard(dashboard);
        const response = await this.connector.get('/api/dashboards/uid/' + dashboard.uid);
        if (response) {
            // eslint-disable-next-line @typescript-eslint/no-unsafe-member-access
            postData.dashboard.version = response?.dashboard?.version;
        }
        await this.connector.post('/api/dashboards/db', postData);
        return Promise.resolve();
    }

}
```

Obrázek 38: Automatické vytváření dashboardů – třída Grafana

V této fázi bylo provedeno základní nastavení Grafany, vytvořen Dashboard splňující všechny požadavky a naimplementována knihovna, pomocí které lze formou jednoduché konfigurace generovat plnohodnotné dashboardy bez nutnosti ručního nastavování v nástroji Grafana. Samotná implementace řešení je hotova, zbývá poslední fáze, a to je otestování monitorovacího systému podle předem připraveného scénáře.

4.9 Nasazení a ověření řešení

V této fázi praktické části je řešení monitorovacího systému implementováno na vzorové webové aplikaci. Následně jsou otestovány problémové stavy podle předem daného scénáře. Cílem je ověřit funkčnost a zda byly požadavky splněny. Tato fáze je tvořena dvěma kroky.

- Implementace měření na vzorové aplikaci.
- Otestování stavů aplikace podle scénáře.

4.9.1 Implementace měření na vzorové aplikaci.

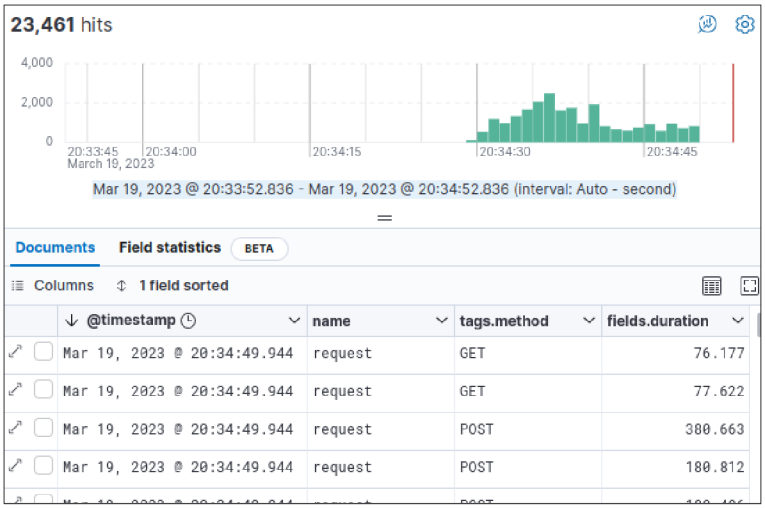
Pro účely testování byla vytvořena jednoduchá Nodejs aplikace typu Rest Api. Poskytuje jen několik jednoduchých funkcí jako vyhledání položek a uložení záznamu nad menším počtem entit. Pro odbavení requestů je použit webový framework Expressjs. Data jsou uložena v MongoDB databázi. Pro komunikaci s databází je použita ORM knihovna Mongoose. Pro měření requstů byla výhodně využita funkcionalita frameworku Expressjs, který umožňuje vkládat vlastní middleware.

```
router.use((req, res, next) => {
    next()
}, (req, res, next) => {

    const mc = measurementCollector.startTransaction('request', {
        url: req.url,
        method: req.method,
    });

    res.on('finish', () => {

        mc.commit();

    });
    next();
});
```

Obrázek 39: Nasazení a ověření řešení – měření doby http requestu v Expressjs.

Tímto vzniklo měření pojmenované `request`. Měření probíhá metodou měření bloku kódu pomocí třídy Transaction, jejíž primární metrikou je doba trvání v milisekundách.

Součástí měření tagy jsou `url` a `method`. V Kibaně lze ověřit, že data z měření byla úspěšně dopravena nástrojem Fluentd do úložiště Elasticsearch.



Obrázek 40: Nasazení a ověření řešení – záznamy měření v Kibaně

Posledním krokem je vytvoření dashboardu. Jeden z požadavků byl, požadavek na automatické generování dashboardů (P.3.1) s cílem maximálního snížení ručního nastavování v nástroji pro sledování. Je tedy nutné vytvořit konfigurační soubor.

```
export const requests: Dashboard = {
    uid: 'requests',
    title: 'Apps Requests',
    datasource,
    panels: [
        {
            title: 'Count',
            datasource,
            query: {
                name: 'request',
            },
            metrics: [
                {
                    type: 'count',
                },
            ],
        },
        {
            title: 'Latency',
            datasource,
            query: {
                name: 'request',
            },
            metrics: [
                {
                    field: 'duration',
                    type: 'avg',
                },
            ],
        },
    ],
};
```

Obrázek 41: Nasazení a ověření řešení – konfigurace dashboardu

Soubor je zpracován a spojen se šablonou. Výsledný JSON objekt je pak odeslán na Rest Api Grafany a celý dashboard je tímto vytvořen. Výsledný dashboard pak lze zobrazit v Grafaně. Bylo vypozorováno, že celý proces implementace měření ve vzorové aplikaci

proběhl velmi rychle a bezproblémově. Už samotná implementace měření a vytvoření dashboardu se pohybovala v řádu nižších jednotek hodin. Pokud by se řešení mělo nastavovat na dalších aplikacích, jistě by probíhalo mnohem rychleji, protože stačí kopírovat konfigurační soubory a řádky kódu.

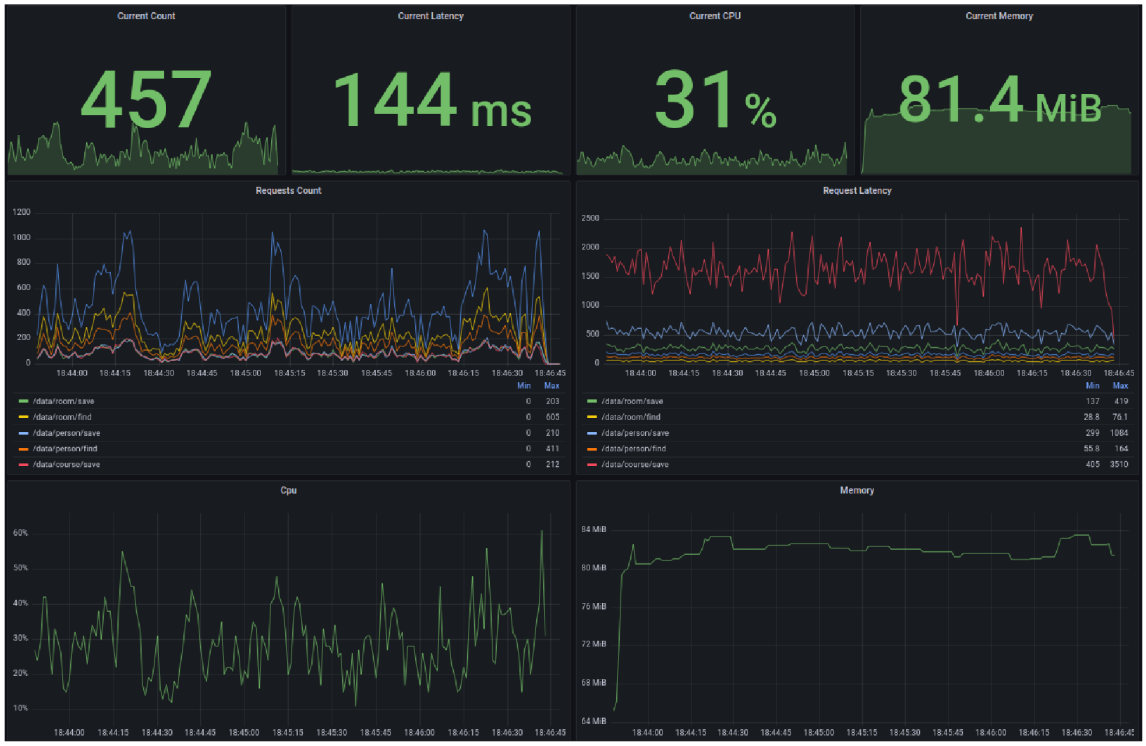
4.9.2 Otestování stavů aplikace dle testovacího scénáře

V tomto kroku bylo na vzorové aplikace navozeno několik stavů, které nejčastěji vznikají v produkci. Cílem je navození určitých situací, a ověření, že monitorovací systém situace správně zobrazuje a upozorňuje na ně. Byly navozeny tyto stavy:

- Běžný provoz
- Zvýšená zátěž
- Zpomalení odezvy http requestu
- Výskyt chyb

Běžný provoz

Nejprve byl navozen stav běžného provoz, kdy aplikace bez problémů odbavuje http requesty, s minimální dobou odezvy. V tomto stavu probíhá vše bez problému, paměť i cpu jsou využívány minimálně, vytížení databáze je minimální. V tomto stavu je očekáváno, že dashboard nesignalizuje žádné výstrahy.



Obrázek 42: Testovací aplikace – dashboard s běžným provozem

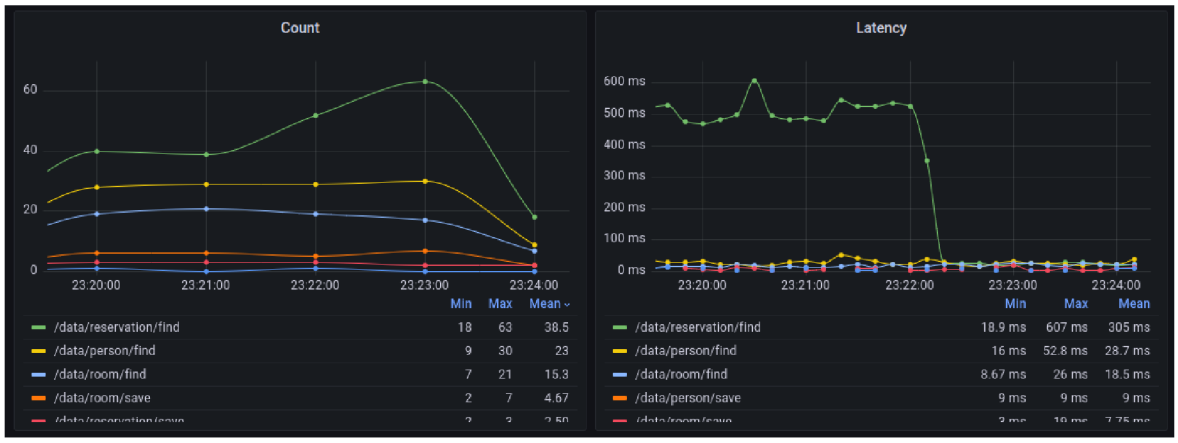
Zvýšená zátěž

Zde je záměrem dostat aplikaci pod stres, tak aby docházelo k degradaci služby. V důsledku zvýšeného počtu http requestů dochází k vytěžování cpu i paměti nebo zatěžování databáze. Výsledným efektem je delší doba zpracování http requestu. Z pohledu zákazníka se pak jedná o dlouhé čekání načítání stránek. Řešením této situace je škálování vertikální tj. navýšení zdrojů cpu a paměti, nebo škálování horizontální tj. zvýšení počtu webových serverů, replik databází apod. Od monitorovacího systému se očekává, že správně zobrazí výstrahy na konkrétních metrikách, dle nastavených prahových hodnot. V grafech pak lze vypozorovat, kdy k problému došlo, a jak to postihlo konkrétní typy requestů nebo jak narůstala spotřeba zdrojů.

Obrázek 43: Testovací aplikace – přetížení.

Zpomalení odezvy http requestu

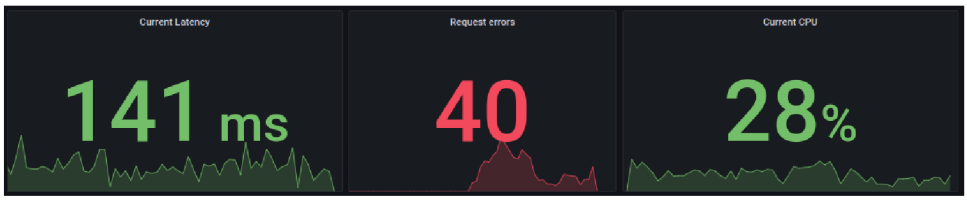
Monitoring by měl objevit i problémovou funkci službu. V tomto případě je navozen stav, kdy z nějakého důvodu se zhoršuje určitý http request. Z grafu by měl být hned na první pohled patrný, jeho doba zpracování je výrazně delší než doba zpracování ostatních requestů. Velmi často se je to v důsledku špatně vykonávaného dotazu do databáze. Po optimalizaci pak monitoring okamžitě potvrdí zlepšení stavu.



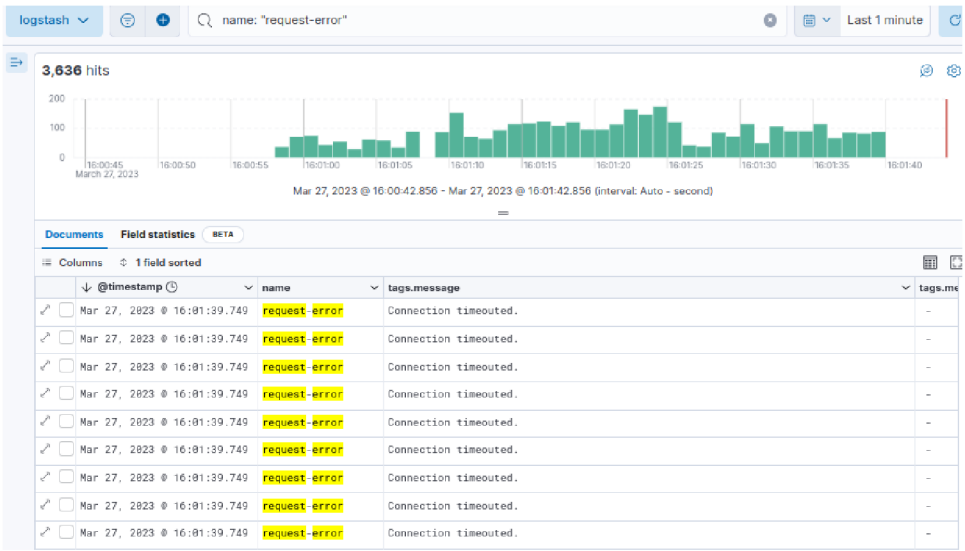
Obrázek 44: Testovací aplikace – optimalizace problémového requestu

Výskyt chyb

Při vykonání http requestu může dojít k chybě. Může se jednat o chybu v programu, ale i o chybu při volání jiné služby jako například volání databázového dotazu nebo odeslání emailu. Pro testovací účely stačí například na krátkou dobu odstavit databázi.



Obrázek 45: Testovací aplikace – výskyt chyb



Obrázek 46: Testovací aplikace – výpis chyb v Kibaně

Tímto posledním krokem bylo celé řešení jednotného monitorovacího systému otestováno a ověřeno. Bylo vyhodnoceno, že správně funguje a plní všechny stanovené požadavky.

5 Zhodnocení výsledků

V závěrečné části práce byl projekt celkově vyhodnocen. Bylo zkoumáno, zda je funkční a zda byly splněny požadavky. Nakonec bylo vyhodnoceno, zda je vhodné přejít do další fáze, tedy postupného nasazení nového monitorovacího systému ve společnosti. Tato fáze se skládá z následujících částí:

- Vyhodnocení požadavků
- Celkové vyhodnocení
- Vyhodnocení dalšího postupu

5.1 Vyhodnocení požadavků

V následujících odstavcích bylo zkoumáno zda byly všechny definované požadavky splněny.

Vyhodnocení požadavků na měření

V oddíle "Výběr úložiště metrik" a "Implementace měření" byly zohledněny požadavky na měření (P.1). Byla implementována podpora pro měření bloků kódu a systémových statistik. K měření je možné dodat i dodatečné atributy za účely agregací nebo filtrování. Všechny požadavky pak byly ověřeny v oddíle "Nasazení a ověření řešení".

Vyhodnocení požadavků na dashboardy

V oddíle "Implementace dashboardů" bylo vytvořeno řešení, které zohledňuje požadavky na funkcionalitu dashboardů (P.2). Řešení umožňuje zobrazovat metriky v čase s požadovanou granularitou času a dle uživatelem vybraného časového intervalu. V grafech je možné i metriky agregovat podle vybraného atributu. Dále je možné vytvářet i jiné statistické komponenty, jako například aktuální hodnoty nebo nastavovat prahové hodnoty pro jednotlivé metriky. Všechny požadavky na funkcionalitu dashboardů byly ověřeny v oddíle "Nasazení a ověření řešení".

Vyhodnocení požadavků na automatizaci

V oddílech "Implementace dashboardů" a "Implementace měření" byly zohledněny dodatečné požadavky na automatizaci (P.3). Služby posílají svá měření a ta jsou automaticky ukládána do úložiště metrik, aniž by bylo nutné registrovat službu nebo metriky. Dashboardy

lze pak vytvořit pomocí jednoduché konfigurace a s využitím Grafana API jsou pak dashboardy automaticky vygenerovány. Požadavky byly ověřeny v oddíle "Nasazení a ověření řešení".

Vyhodnocení technických požadavků

V oddílech výběru nástrojů byly zohledněny požadavky na modularitu (P.4.1) a open-source (P.4.2). Do fáze porovnání nástrojů byly vybrány pouze open-source nástroje. Tyto nástroje byly dále hodnoceny s prioritou na modularitu. V oddíle „Nastavení a spuštění nástrojů" bylo implementováno řešení s pomocí technologie Docker, které splňuje požadavek na snadné spuštění (P.4.3) v různých prostředích. To bylo následně ověřeno v oddíle „Nasazení a ověření řešení".

Celkově bylo vyhodnoceno, že všechny požadavky byly splněny.

5.2 Celkové vyhodnocení

Zvolená architektura doporučená literaturou tzv. Composable monitoring se osvědčila. Umožňuje kombinovat jednotlivé komponenty systému a s důrazem na modularitu lze v budoucnu jednotlivé komponenty vyměňovat nebo používat současně. Zvolené nástroje lze tedy kdykoliv měnit s minimálními náklady.

Výběr nástrojů nebyl vždy jednoznačný. Elasticsearch byl nakonec upřednostněn především pro vyřešené škálování a jednotné úložiště s logy. Stejně tak nástroj pro sběr logů a metrik Fluentd může být v budoucnu vyhodnocen jako nevhodný, ale snadno jej bude možné zaměnit například za moderní Vector.

Nástroje jsou spouštěny ve formě Docker kontejnerů a orchestrovány pomocí nástroje docker-compose. Toto je spíše základní použití, v praxi se častěji využije systém Kubernetes pro orchestraci kontejnerů v distribuovaném prostředí. Lze využít konfigurace a některá nastavení z docker-compose konfigurace.

5.3 Vyhodnocení dalšího postupu

Bylo vyhodnoceno, že nasazení monitorovacího řešení na stávající nebo nově vznikající webové aplikace je velmi snadné a rychlé. Toho bylo docíleno především splněním požadavků na snadné spuštění a požadavky na automatizaci. Implementace nového měření, registrace služby a vytvoření plnohodnotného dashboardu je otázka nižších jednotek hodin oproti dnes i dnům. S přihlédnutím na zjištěné nedostatky aktuálního stavu monitoringu lze vyhodnotit celkové přínosy, pokud by se řešení zavedlo na všech projektech.

Nedostatek	Náklady, dopady, rizika	Budoucí stav
Nedostatečná úroveň pokrytí monitoringem aplikací a služeb	Pozdní detekce chyb. Omezená analýza chyb. Ztráty, pokuty, ztráta důvěry.	Aktivní předcházení problémům. Okamžité řešení problému. Rychlé nalezení root cause.
Vysoké náklady na údržbu infrastruktury monitorovacích systémů	Alokace 20+ lidí napříč týmy. Odhadované roční náklady 2000 MD	Jeden specializovaný tým 2 lidé Odhadované roční náklady 350 MD
Nadměrné množství udržovaných technologií	12+ nástrojů Filebeat, Logstash, rsyslog, Prometheus, InfluxDB, Elasticsearch, Kibana atd.	4 nástroje Fluentd, Elasticsearch, Kibana, Grafana
Vysoká míra ruční práce	Odhadované náklady na nastavení dashboardu včetně úložiště: 1MD	Odhadované náklady na nastavení dashboardu včetně úložiště: 0.2 MD

Tabulka 36: Vyhodnocení dalšího postupu – budoucí stav

Řešení nového monitorovacího systému bylo vyhodnoceno jako úspěšné a schopné plnit požadované cíle. Nebrání tedy dalšímu postupu, který je nutné naplánovat. Prosazení řešení ve společnosti je manažersky náročná činnost. Bude nutné další postup vykomunikovat s jednotlivými týmy a naplánovat nasazení na všech projektech. Postup bude pravděpodobně takový, že nejprve se řešení bude nasazovat na menších projektech a zejména na těch, kde monitoring zcela chybí. Postupně bude přicházet zpětná vazba z produkce a řešení se bude postupně odlaďovat a doplňovat o další funkcionalitu. V poslední fázi nasazování řešení budou nahrazeny všechny již existující zastaralé monitorovací systémy. Nakonec dojde k úplnému smazaní starých řešení, zrušení duplicitních technologií a v konečném důsledku uvolnění lidských zdrojů, které jsou alokovány na údržbu těchto systémů. Časový odhad celého procesu je odhadován na dva roky.

6 Závěr

Cílem práce bylo celkově zlepšit stav monitoringu webových aplikací ve společnosti. Na začátku proběhla analýza současného stavu monitoringu formou diskuse se zástupci technických týmů a byly zjištěny nedostatky.

Bylo zjištěno, že je provozován určitý počet služeb bez monitoringu. V takovém případě není možno dohlížet na služby proaktivně a hrozí rizika finančních ztrát, pokut a ztráty důvěry zákazníků. Dále bylo zjištěno, že služby již pokryté monitoringem, používají vysoký počet nástrojů nebo se duplikují řešení. Zde vznikají zbytečné náklady na lidské zdroje, protože údržba takového počtu nástrojů je nákladná. Nakonec bylo zjištěno, že je potřeba vysoká míra ruční práce při nastavování monitorovacích systémů, která se neustále opakuje.

Následujícím krokem bylo navržení jednotného systému, který odstraní zjištěné nedostatky. Byly definovány požadavky na tento nový systém, které vycházely ze zmíněných nedostatků a byly doplněny technickými požadavky. Požadavky se zaměřovaly na měření, funkcionalitu dashboardů, automatizaci a technické aspekty.

Následoval návrh systému, který se opíral o doporučení ze získaných teoretických základů. Architektura moderních monitorovacích systémů tzv. “composable monitoring” se vymezuje proti původním monolitickým systémům jako Nagios. Systém je složen z volně propojených komponent, které lze jednoduše kombinovat. Byl zohledněn požadavek na modularitu, který umožňuje snadnou výměnu komponenty v budoucnu.

Následně byly nástroje pro jednotlivé komponenty vybrány. Výběr se týkal nástrojů pro sběr logů a metrik, úložiště metrik a nástroje pro sledování. Z celkem devíti nástrojů byly vybrány s pomocí Saatyho metody párového porovnání tři nástroje: Fluentd, Elasticsearch a Grafana. Vzhledem k tomu, že byl zohledňován požadavek na modularitu, nebude v budoucnu problém případně některou komponentu vyměnit.

Následovala implementační fáze. V prvním kroku byly nakonfigurovány, propojeny a spuštěny vybrané nástroje. Celé řešení bylo kontejnerizováno a orchestrováno s využitém technologie Docker, a tím byl splněn požadavek spuštění v různých prostředích.

Byla naimplementována knihovna pro měření v programovacím jazyce TypeScript. Knihovna umožňuje měřit ve webové službě bloky kódu a systémové prostředky a plně splňuje požadavky na měření. Dále byla naimplementována knihovna pro automatické

generování dashboardů. Knihovna plně splňuje požadavky na funkcionalitu dashboardů a automatizaci.

V posledním kroku bylo celé řešení použito ve vzorové aplikaci. Bylo ověřeno, že nasazení nového řešení je snadné a splňuje všechny požadavky. Nakonec bylo celé řešení otestováno simulací zátěže nebo chybových stavů. Monitorovací systém vždy správně upozornil na problémové situace.

Nový jednotný monitorovací systém byl schválen a byl navržen další postup. Měl by vzniknout specializovaný tým, který bude mít projekt na starosti a bude jej dále rozvíjet. Jednotlivé technické týmy budou postupně tento systém přejímat a nasazovat na provozované služby. Monitoring tak bude doplněn u služeb, které jej postrádají. U služeb, které jsou již pokryty nějakou formou monitoringu, bude nutné toto řešení nahradit stávajícím. Celý proces postupného nasazení nového monitoringu bude náročný na naplánování a komunikaci. Realistický odhad jsou dva roky. I přes náklady, které již byly vynaloženy a bude ještě nutné vynaložit, je celkový přínos jednoznačný. Dojde k celkovému zkvalitnění dohledu služeb a ke snížení nákladů spojených s provozem monitorovací infrastruktury.

7 Seznam použitých zdrojů

1. **JULIAN, MIKE.** *Practical Monitoring.* ISBN 978-1-491-95735-6. 1005 Gravenstein Highway North, Sebastopol, CA 95472. : O'Reilly Media, Inc., 2018.

2. **LIGUS, SLAWEK.** *Effectice Monitoring and Alerting.* místo neznámé : O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. 270 s. ISBN 978-1-449-33352-2, 2013.

3. **W3C.** Web Services Architecture. *w3.* [Online] 2004. https://www.w3.org/.

4. **PRUSTY, NARAYAN.** *Modern JavaScript Applications.* místo neznámé : Published by Packt Publishing Ltd., Livery Place, Birmingham B3 2PB, UK 236 s. 978-1-78588-144-2., 2016.

5. **DB-engines.** DB-engines. [Online] https://db-engines.com/.

6. **BURNS, BRENDAN.** *Designing Distributed Systems.* místo neznámé : O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. 165 s. ISBN 978-1-492-03177-2, 2018.

7. **Elastic.** Observability and Security — built on Elasticsearch. *Elastic.* [Online] https://www.elastic.co/.

8. **Influxdata.** InfluxDB Times Series Data Platform. *Influxdata.* [Online] 2023. https://www.influxdata.com/.

9. **Grafanalabs.** Grafana: The open observability platform. *The open observability platform.* [Online] https://grafana.com/.

10. **dev.to.** Top 5 Open-Source Log Shippers (alternatives to Logstash) in 2022. [Online] 2022. https://dev.to/max_kray/top-5-open-source-log-shippers-alternatives-to-logstash-in-2022-5f24.

11. **signoz.io.** What is a log shipper - Top 7 Log Shippers that you can use. [Online] 2022. https://signoz.io/blog/log-shipper/.

12. **www.cncf.io.** Logstash, Fluentd, Fluent Bit, or Vector? How to choose the right open-source log collector. [Online] 2023. https://www.cncf.io/blog/2022/02/10/logstash-fluentd-fluent-bit-or-vector-how-to-choose-the-right-open-source-log-collector/.

13. **Logstash.** Logstash. [Online] 2023. https://www.elastic.co/logstash/.

14. **Fluentd.** Fluentd. [Online] https://www.fluentd.org/.

15. **Vector.** A lightweight, ultra-fast tool for building observability pipelines. *Vector.* [Online] https://vector.dev/.

16. **OpenTSDB.** OpenTSDB. *The Scalable Time Series Database.* [Online] 2023. http://opentsdb.net/.

17. **Berman, Daniel.** InfluxDB vs Elasticsearch for time series and metrics data. [Online] 2023. https://logz.io/blog/influxdb-vs-elasticsearch/.

18. **Churilo, Chris.** InfluxDB Markedly Outperforms OpenTSDB in Time Series Data & Metrics Benchmark. [Online] 2018. https://www.influxdata.com/blog/influxdb-markedly-outperforms-opentsdb-in-time-series-data-metrics-benchmark/.

19. **Langston, Elliot.** Best Open Source Application Monitoring Tools. [Online] 2022. https://www.metricfire.com/blog/best-open-source-application-monitoring-tools/ .

20. **Uptrace.** Top 11 Grafana Alternatives [comparison 2023]. [Online] 2023. https://uptrace.dev/blog/grafana-alternatives.html.

21. **Kibana.** Explore, Visualize, Discover Data - Elastic. *Kibana.* [Online] https://www.elastic.co/kibana/.

22. **Cyclotron.** Cyclotron. [Online] http://cyclotron.io/.

23. **Docker.** Docker. *Accelerated, Containerized Application Development.* [Online] https://www.docker.com/.

24. **Kubernetes.** Production-Grade Container Orchestration. *Kubernetes.* [Online] https://kubernetes.io/.

25. **Elastic, APM.** Application Performance Monitoring (APM). *APM Elastic.* [Online] https://www.elastic.co/observability/application-performance-monitoring.

26. **InfluxDB vs. Elasticsearch for Time Series Analysis. *logz.io.* [Online] 2023. https://logz.io/blog/influxdb-vs-elasticsearch/.**

8 Seznam obrázků, tabulek a zkratek

8.1 Seznam obrázků

8.2 Seznam tabulek

8.3 Seznam použitých zkratek

API	Application programming interface
APM	Application Performance Monitoring
DDoS	Distributed Denial of Service
HA	Vysoká dostupnost (High availability)
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
K8S	Kubernetes
MD	Man-day
NPM	Node Package Manager
ORM	Objektově relační mapování (Object–relational mapping)
SLA	Service-level agreement
TS	TypeScript
TSDB	Time series database
URL	Uniform Resource Locator
XML	Extensible Markup Language

Přílohy

Konfigurace grafu v Grafaně ve formátu JSON


```
{
    "datasource": {
      "type": "elasticsearch",
      "uid": "kkxFjoJVz"
    },
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "thresholds"
        },
        "mappings": [],
        "thresholds": {
          "mode": "absolute",
          "steps": [
            {
              "color": "green",
              "value": null
            },
            {
              "color": "#EAB839",
              "value": 200
            },
            {
              "color": "red",
              "value": 1000
            }
          ]
        },
        "unit": "ms"
      },
      "overrides": []
    },
    "gridPos": {
      "h": 7,
      "w": 6,
      "x": 0,
      "y": 0
    },
    "id": 8,
    "options": {
      "colorMode": "value",
      "graphMode": "area",
      "justifyMode": "auto",
      "orientation": "auto",
      "reduceOptions": {
        "calcs": [
          "lastNotNull"
        ],
        "fields": "",
        "values": false
      },
      "textMode": "auto"
    },
    "pluginVersion": "9.3.6",
    "targets": [
      {
        "alias": "",
        "bucketAggs": [
          {
            "field": "@timestamp",
            "id": "2",
            "settings": {
              "interval": "10s"
            },
            "type": "date_histogram"
          }
        ],
        "datasource": {
          "type": "elasticsearch",
          "uid": "kkxFjoJVz"
        },
        "metrics": [
          {
            "field": "fields.duration",
            "id": "1",
            "type": "avg"
          }
        ],
        "query": "name: request",
        "refId": "A",
        "timeField": "@timestamp"
      }
    ],
    "title": "Latency",
    "type": "stat"
  }
```