Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Návrh a implementace Elastic Stack logování

(ELK Stack)

Bakalářská práce

Autor: Martin Malíř
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Ph.D. Tomáš Svoboda

Odborný konzultant: Ing. Ph.D. Tomáš Svoboda
Katedra informačních technologií

Hradec Králové 04. 2024

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 10.4.2024 Martin Malíř

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Ph.D. Tomáši Svobodovi za metodické vedení práce.

Anotace

Cílem této práce je obeznámit čtenáře s problematikou aplikačních logů. Vysvětlit k čemu slouží, co by měli obsahovat a jak by s nimi mělo být nakládáno. Čtenářovi je popsána struktura ELK stack a rodina Beats, což je soubor aplikací, které se zabírají sběrem, vizualizací a analýzou systémových nebo jiných událostí.

V úvodu práce budou čtenáři seznámeni s konceptem aplikačních logů a jejich účelem v kontextu sledování, diagnostiky a zabezpečení IT infrastruktury. Následně bude zdůrazněn význam kvalitního logování pro identifikaci problémů, zabezpečení, auditování a zlepšení výkonu systémů.

Další část práce se zaměří na technické aspekty správy aplikačních logů pomocí ELK stacku (Elasticsearch, Logstash, Kibana) a rodiny Beats. ELK stack představuje komplexní řešení pro sběr, ukládání a vizualizaci logů, zatímco Beats jsou aplikace specializované na sběr různých typů dat a jejich přenos do ELK stacku.

Poslední částí práce je ukázka logovací aplikace, nasazené na operačním systému Linux, která slouží jako ukázka tvoření strukturovaných logů. Následně je zde popsána instalace a konfigurace jednotlivých částí ELK stacku, pomocí kterého jsou logy přesunuty ze souborového systému do databáze Elasticsearch a vizualizovány pomocí aplikace Kibana.

Annotation

The aim of this work is to acquaint readers with the issue of application logs, explaining their purpose, what they should contain, and how they should be handled. The structure of the ELK stack and the Beats family is described, which is a set of applications dedicated to collecting, visualizing, and analyzing system or other events.

In the introduction of the work, readers will be introduced to the concept of application logs and their role in the context of monitoring, diagnostics, and securing IT infrastructure. Subsequently, the importance of quality logging will be emphasized for problem identification, security, auditing, and system performance improvement.

The next part of the work will focus on the technical aspects of managing application logs using the ELK stack (Elasticsearch, Logstash, Kibana) and the Beats family. The ELK stack represents a comprehensive solution for collecting, storing, and visualizing logs, while Beats are specialized applications for collecting various types of data and transferring them to the ELK stack.

The final part of the work consists of a demonstration of a logging application deployed on the Linux operating system, serving as an example of creating structured logs. It then describes the installation and configuration of individual components of the ELK stack, through which logs are transferred from the file system to the Elasticsearch database and visualized using the Kibana application.

Obsah

Seznam obrázků

1 Úvod

V neustále se rozšiřujícím světě IT je dobrou praktikou si udržovat informace o chování nasazených aplikací abychom předešli nečekanému chování nebo incidentům, které mohou, vzniknou jako důsledek špatné správy nebo konfigurační chyby těchto aplikací.

V rámci střední nebo velké IT firmy, která spravuje desítky aplikací, by bylo časově velice náročné procházet jednotlivé servery a kontrolovat, zda aplikace fungují dle očekávání. Nabízí se nám proto několik implementačních řešení pro centrální sběr a analýzu těchto logů. Jednou z těchto řešení je takzvaný ELK stack, který je tvořen aplikacemi Elasticsearch, Logstash a Kibana.

Jako jednu z metrik nám mohou posloužit logy, které nasazené aplikace generují. Log soubory mohou mít podobu textových souborů, nebo využívat jiných struktur jako je například JSON. Informace z těchto zdrojů nám pak mohou posloužit jako zpětná vazba během fáze vývoje aplikace, nebo jako monitorovací prostředek po nasazení do funkčního prostředí.

ELK stack pipeline je často doplněna o nějakého zprostředkovatele logovacích záznamů, který je umístěn přímo na serveru běžící aplikace. Mezi tyto prostředníky patří například rodina Beats, která nabízí několik funkčních řešení od sběru a analýzy logovacích dat, až po sledování sítě nebo sledování eventů v systému Windows.

Pomocí této implementace je pro IT firmy velice jednoduché sledovat a analyzovat logy z aplikací a předejít tak neočekávaným výpadkům jejich služeb, nebo zpětně analyzovat proč k takovým výpadkům došlo.

2 Cíl práce

V teoretické části práce si popíšeme jednotlivé aplikace sloužící ke sběru a analýze logovacích záznamů. Jako první se podíváme na rodinu Beats, ze které si vybereme odesílatele Filebeat, který je primárně využíván k této problematice. Poté si popíšeme architekturu jednotlivých aplikací v ELK stacku, podíváme se na jejich architekturu a popíšeme si, k čemu slouží.

V praktické části se zaměříme na návrh a implementaci jednoduché logovací aplikace, která jako výstup poskytne aplikační log ve formátu JSON. Tyto logy pak budeme pomocí aplikace Filebeat předávat dále do ELK stacku a popíšeme si, jak jednotlivé části nainstalovat a nakonfigurovat na systému Linux.

3 Aplikační logy

3.1 Význam aplikačních logů

V našem světě plného různých internetových služeb a aplikací je velice nepříjemné jak pro nás, tak pro firmu, která tyto služby poskytuje, pokud dojde k neočekávanému výpadu a přístup k těmto službám je nám odepřen. To se sebou nese různá rizika, podle typu a prostředí, do kterého se snažíme přistupovat. Například u bankovního systému, nebo ve zdravotnickém prostředí chceme zaručit, aby se takové incidenty nestávaly, nebo aby jejich dopady byly minimální. K tomu nám mohou posloužit aplikační logy, ze kterých bychom měli být schopni zjistit, jaké operace právě aplikace provádí, nebo kteří uživatelé k aplikaci přistupují a co dělají. Pomocí těchto údajů můžeme identifikovat a odstraňovat chyby, sledovat bezpečnostní incidenty a předcházet nečekanému chování či chybě aplikace. [1]

3.2 Obsah aplikačního logu

Aplikační log by měl obsahovat informace o čase, kdy se nějaká událost stala. To by mělo být doplněno vážností události, aby bylo možné rychle zjistit, zda tomuto záznamu věnovat zvýšenou pozornost, nebo zdali se jedná pouze o informaci pro čtenáře zprávy. V poslední řadě by zde neměl chybět zdroj, který událost vyvolal, to může být aplikační metoda, nebo uživatel a zpráva, která dostatečně popíše, co se stalo. [1]

2023-08-13 20:00:00 | INFO | Application.getStatus : Application is OK

Z této události například můžeme vyčíst, že k ní došlo dne 13. srpna 2023 v 8 hodin večer. Dále vidíme, že se jedná pouze o informační zprávu, která nám oznamuje stav aplikace. Po přečtení této zprávy jako správce aplikace víme, že aplikace běží v pořádku a neměli bychom očekávat žádné nečekané chování.

2023-08-13 20:10:00 | WARN | Application.changeParameter : User admin changed settings

2023-08-13 20:10:10 | FATAL | Application.getStatus : Application is DOWN

Z následujících dvou ukázek můžeme zjistit, že administrátor aplikace, nebo uživatel, který získal přístup k uživateli admin, změnil nastavení v aplikaci a po deseti sekundách od uložení změn došlo k zastavení aplikace. Díky těmto logovacím zprávám víme, co se s aplikací dělo a známe i viníka, který toto chování aplikace způsobil.

Během vývoje aplikace je důležité určit, které události je potřebné logovat. Není dobrou praktikou aplikační logy přehlcovat informacemi, neboť to vede ke špatné čitelnosti a proces odstraňování problému to může velice zkomplikovat a zpomalit.

3.3 Úrovně logovací zprávy

V praxi se můžeme setkat s několika úrovněmi logovacích zpráv. Mezi nejčastější patří:

- **TRACE**
  - Nejpodrobnější informace, které se používají pouze výjimečně. Tyto logy nám většinou zprostředkovávají plnou viditelnost toho, co se uvnitř aplikace děje. Tyto logy se dají použít například k anotaci každého kroku nějakého algoritmu nebo k popisu metody a jejích parametrů. [2]
- **DEBUG**
  - Méně podrobné informace oproti TRACE, i tak se jedná o zprávy, které nesou více informací, než by mělo být za běžného chodu nutné. Tyto logy by měli být použity během fáze vývoje aplikace pro diagnostiku problémů a ladění, nebo při spouštění aplikace v testovacím prostředí. Po nasazení na funkční prostředí by tyto logy měli být vypnuty. [2]
- **INFO**
  - Standardní úroveň záznamu označující, že došlo k nějaké akci, že aplikace vstoupila do nějakého stavu apod. Záznamy nesoucí tuto úroveň by měli být pouze informativní a pravidelné prohlížení by nemělo vést k zmeškání důležitých informaci. [2]

- **WARN**
    - Úroveň záznamu, která označuje, že v aplikaci nastala neočekávaná událost, která by mohla narušit nějaký proces. To však neznamená, že by aplikace selhala. Tato úroveň by se měla použít v situacích, které jsou neočekávané, ale nebrání aplikaci v pokračování činnosti. [2]
- **ERROR**
    - Tato úroveň by měla být použita, pokud aplikace narazí na problém, který brání některé funkci nebo více funkcionalit. Například pokud uživatel není schopen se přihlásit do aplikace zpracovávající data, ale proces zpracovávání není narušen. [2]
- **FATAL**
    - Úroveň, která oznamuje, že aplikace se dostala do stavu, ve kterém klíčová funkcionalita není dostupná, nebo je nefunkční celá aplikace. [2]

3.4 Řízení přístupu k aplikačním logům

Přestože aplikační logy slouží k dobrým účelům, je dobré kontrolovat kdo k nim má přístup. V případě pokusu o útok na aplikaci mohou možnému útočníkovy otevřít nové cesty, jak aplikaci vysadit z provozu, nebo se dostat na místa, ke kterým by normálně neměl oprávnění. Je nutné si pamatovat, že přestože se jedná o monitorovací metriku, stále se zde vyskytují zprávy, které nás informují o stavu aplikace a mohou tak prozradit zranitelná místa, která mohla být během vývoje zanedbána nebo přehlédnuta. [1]

Po nasazení aplikace na funkční prostředí, je dobré logovací složku opatřit zabezpečením, které nám dané prostředí poskytuje. Na systému Linux je to například změna vlastníka a skupiny, která by měla být stejná jako ta, která byla přiřazena aplikaci. Také by mělo dojít ke změně oprávnění na čtení, nebo změnu souborů. V poslední řadě je dobré pamatovat na to, že aplikační logy by za správného provozu neměli opouštět aplikační server. Pokud si administrátor aplikace přenese tyto informace na osobní, nebo firemní počítač, vystavujeme se dalšímu nebezpečí, pokud dojde ke kompromitaci zařízení dané osoby. [1]

V rámci bezpečnosti by k aplikačním logům měla dostat práva samotná aplikace, aby mohla do adresáře zapisovat. Dále by to měl být IT specialista, nebo administrátor, který dohlíží na správné fungování aplikace. V tomto případě by mělo být využité heslo, pomocí kterého se do zabezpečeného adresáře přistupuje. Pro ostatní uživatele by tento adresář neměl být vůbec dostupný. [1]

3.5 Využití aplikačních logů

Jak jsme si již řekli, aplikační logy nám slouží k mnoha věcem. Nyní by však bylo dobré si přiblížit kdo a v jakém kontextu může k těmto informacím přistupovat.

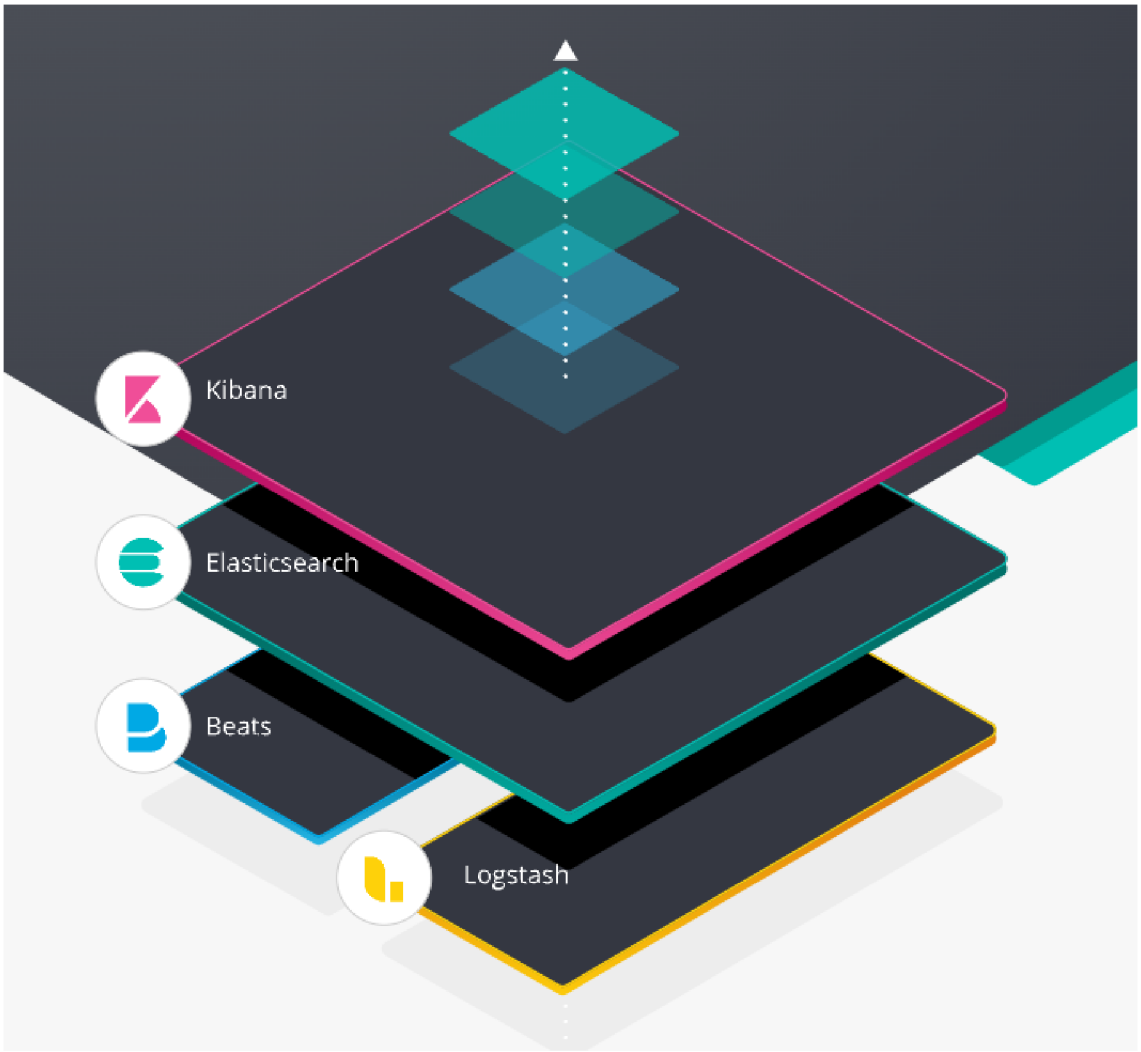
V první řadě jsou to programátoři a vývojový tým. Těm mohou DEBUG a TRACE logy podrobně popsat k čemu v aplikaci dochází a mohou pomocí těchto informací jednodušeji aplikaci ladit, nebo odhalovat bezpečnostní rizika. [1][2]

Poté zde máme specialisty řízení přístupu k aplikaci. Těmto lidem můžou aplikační logy pomoci zjišťovat, kteří uživatelé k aplikaci přistupují a co během své návštěvy dělají. Pokud by bylo odhaleno nějaké podivné chování, mohou prověřit, zdali se nejedná například o bota, který se snaží aplikaci vyřadit z provozu, nebo se dostat k citlivým údajům. [1][2]

V poslední řadě je to administrátor a technik spravující aplikaci. Tito lidé využívají logy pro zjištění aktuálního stavu a zajišťují, že nedojde k nečekaným výpadům, nebo se pokusí minimalizovat škody, pokud k nim dojde. Zároveň také mohou poskytovat zpětnou vazbu vývojovému týmu pro zlepšení aplikace a získat znalosti proč k některým chybám dochází. [1][2]

4 ELK Stack

Pod pojmem ELK Stack chápeme spojení tří aplikací do jednoho funkčního celku. Písmeno E zde zastupuje databázi Elasticsearch, písmeno L označuje procesovací pipeline Logstash a písmeno K vizualizační nástroj Kibana. [3]



Obrázek 1 - ELK Stack. Zdroj: [1]

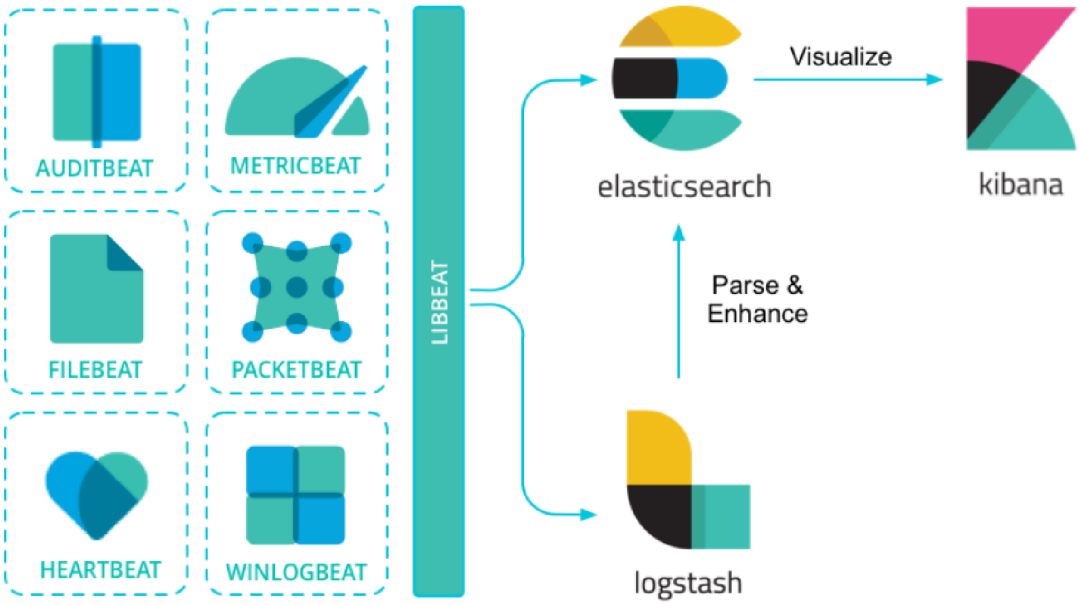
V roce 2015 byl tento Stack rozšířen o rodinu Beats. Ta přináší sadu single-purpose nástrojů pro přenos logovacích záznamů ze specifikovaného zdroje, nebo zdrojů do definovaného cíle.

V dnešní době se ještě můžeme setkat s dalšími zkratkami jako ELBK, kde písmeno B zastupuje právě rodinu Beats. Nebo zkratku EBK, kde byla aplikace Logstash nahrazena Beatem. V praxi se můžeme setkat s ještě dalšími rozšířeními

tohoto původního Stacku, například o Apache Kafka, která slouží jako bufferovací nástroj, pokud očekáváme velké množství příchozích dat z několika front. [3]

4.1 Beats

Beats jsou open-source datový zprostředkovatelé, kteří se instalují jako agenti na zdrojových serverech, ze kterých mohou posílat logy přímo do Elasticsearch nebo do dalších komponent ELK Stacku. Často se můžeme setkat i s řešením, kde se využívají aplikace třetích stran pro zpracování nebo transformaci zpráv, před uložením. [4][5]



Obrázek 2 - Beats ilustrace. Zdroj: [2]

4.1.1 Hearbeat

Heartbeat je systémový démon, který se instaluje na vzdálený server, aby pravidelně kontroloval stav služeb a určoval, zda jsou dostupné. Hodí se na místech, kde potřebujeme ověřit, že splňujeme smlouvy o dostupnosti a provozuschopnosti služeb. Dalším využitím je v oblasti zabezpečení, kde nám Heartbeat dovoluje kontrolovat a ověřovat přístupy zvenčí, pomocí čehož můžeme monitorovat provoz na soukromých serverech. [6]

Hearbeat lze nakonfigurovat tak, aby prováděl kontrolu všech IP adres DNS, které jsou rozlišitelné pro zadaný název hostitele. Tímto způsobem lze zkontrolovat všechny služby, které jsou zatíženy, a zjistit tak, zda jsou dostupné. [6]

Při konfiguraci Heartbeatu určujeme monitory, které identifikují názvy hostitelů, které chcete kontrolovat. Každý monitor běží podle definovaného časového plánu a je možné definovat i více monitorů, které se navzájem překrývají.

V současné době monitory podporují několik protokolů pro kontrolu hostů:

- ICMP Echo Request
  - Tento monitor lze použít v případě, pokud chceme jednoduše zkontrolovat, zdali je služba dostupná. Tento monitor vyžaduje přístup na server s root oprávněním
- TCP
  - Tento monitor lze nakonfigurovat tak, aby se připojil na koncový bod tím, že odešle nebo přijme vlastní zátěž.
- HTTP
  - Pomocí tohoto monitoru lze ověřit, že služba vrací očekávanou odpověď, jako je konkrétní stavový kód, záhlaví odpovědi nebo obsah.

Monitory TCP a HTTP podporují SSL/TLS a některé proxy nastavení. [6]

4.1.2 Auditbeat

Auditbeat je lehký datový odesílatel, který slouží ke sledování aktivit uživatelů a procesů na serveru. Může posloužit pro sběr auditových událostí z Linuxového auditového Frameworku, nebo ho lze použít k detekci změn v kritických souborech, jako jsou spustitelné nebo konfigurační soubory. Pomocí těchto informací pak můžeme identifikovat potenciální narušení bezpečnostních pravidel či skryté bezpečnostní hrozby. [7]

4.1.3 Winlogbeat

Winlogbeat posílá událostní záznamy systému Windows. Lze jej nainstalovat jako samostatnou aplikaci nebo jako službu systému. Jednotlivé události se ze

systému dostávají pomocí Windows API, poté jsou filtrovány na základě kritérií, která si nastavil uživatel. Poté dochází k odesílání záznamů na konfigurované výstupy, nejčastěji se jedná o Logstash nebo Elasticsearch. [8]

Pokaždé, co Winlogbeat přečte zprávu záznamu, uloží si na disk poslední pozici čtení, aby mohl pokračovat po restartu systému. Mezi nejčastěji sledované události patří:

- Události aplikací
- Události hardwaru
- Bezpečnostní události
- Systémové události

4.1.4 Packetbeat

Aplikace Packetbeat nám zprostředkovává analýzu síťových paketů v reálném čase, pomocí které můžeme monitorovat výkon aplikací. Jeho funkčnost spočívá v zachytávání síťového provozu mezi aplikačními servery, dekóduje protokoly HTTP, MySQL, Redis a spoustu ostatních. [9]

V současné době Packetbeat podporuje následující protokoly:

- ICMP (verze 4 a 6)
- DHCP (verze 4)
- DNS
- HTTP
- AMQP 0.9.1
- Cassandra
- Mysql
- PostgreSQL
- Redis
- Thrift-RPC
- MongoDB
- Memcache
- NFS

- TLS
- SIP/SDP (beta)

Packetbeat může být nasazen na stejném serveru jako aplikační procesy, nebo může být umístěn na vlastním serveru. Pokud běží na dedikovaných serverech, Packetbeat může získávat provoz z odrazových portů switche nebo z odposlechových zařízení. V takovém nasazení není žádná nadbytečná zátěž na monitorovanou aplikaci. [9]

4.1.5 Metricbeat

Metricbeat je lehký datový odesílatel, který můžete nainstalovat na servery pro sběr metrik z operačního systému a ze služeb běžících na serveru. Jeho hlavním úkolem je sbírat statistiky a odesílat je do definovaných výstupů, nejčastěji do databáze Elasticsearch nebo pipeline Logstash. [10]

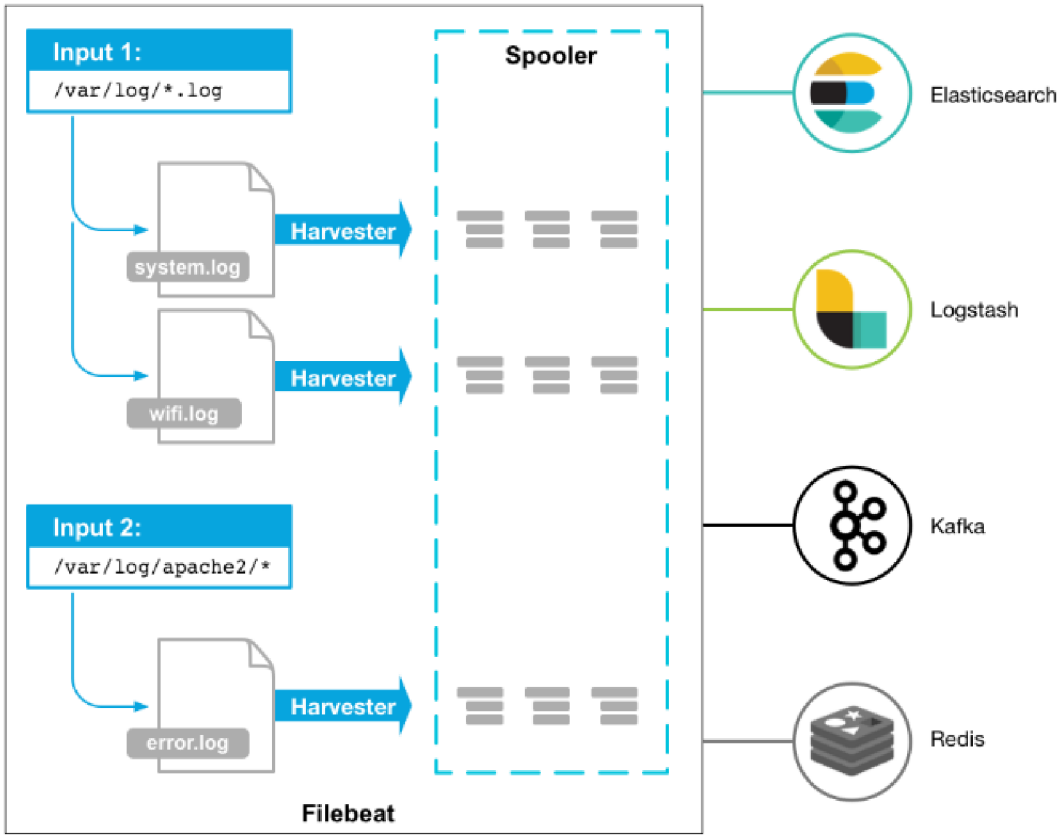
V dnešní době zvládá Metricbeat sběr metrik z těchto aplikací:

- Apache
- HAProxy
- MongoDB
- MySQL
- Nginx
- PostreSQL
- Redis
- Systém
- Zookeeper
- A další

4.1.6 Filebeat

Filebeat je lehký datový odesílatel pro předávání a centralizaci logovacích dat. Instaluje se jako agent na aplikační servery, ze kterých odesílá logovací záznamy do specifikovaného cíle.

Filebeat ve svém vnitřku funguje následovně. Po spuštění nastartuje jeden nebo více vstupů, které hledají v místech, která byla určena pro sběr logovacích dat. Pro každý log, který je nalezen, je spuštěn harvester. Každý harvester čte jediný log a hledá nový obsah na konci souboru. Tento nalezený obsah je posílán do libbeat, který agreguje události a posílá agregovaná data na výstup. [11]



Obrázek 3 - Filebeat, ilustrace. Zdroj: [3]

4.1.6.1 Filebeat – harvester

Harvester je zodpovědný za čtení obsahu jednoho souboru, který čte řádek po řádku a odesílá obsah na výstup. Za každý soubor je spuštěn jeden harvester. Ty nesou zodpovědnost za otevření a uzavření souboru, což znamená, že deskriptor souboru zůstává otevřený, dokud harvester běží. Pokud během sběru dojde k přejmenování nebo k odstranění souboru, Filebeat pokračuje ve čtení. Důsledkem toho je místo na disku rezervováno, dokud harvester neukončí svoji činnost. Ve výchozím nastavení je soubor udržován otevřen, dokud nedojde k dosažení proměnné *close_inactive*. [12]

4.1.6.2 Filebeat – vstup

Vstup je zodpovědný za správu harvesterů a hledání všech zdrojů, ze kterých se bude číst. Pokud je typem vstupu *log*, vstup hledá všechny soubory na disku, které odpovídají definováním vzorům a pro každý soubor spustí harvester. [12]

V následujícím příkladu byl Filebeat nakonfigurován tak, aby sbíral všechny soubory, které odpovídají určeným vzorům:

```
filebeat.inputs:
- type: log
  paths:
    - /var/log/*.log
    - /var/path2/*.log
```

Filebeat momentálně podporuje několik typů vstupů, které lze míchat i použít vícekrát. Vstup s protokoly provádí kontrolu každého souboru, zda je třeba spustit harvester, zda již běží, nebo zda lze soubor ignorovat. Nové řádky jsou zpracovávány pouze tehdy, změní-li se velikost souboru od posledního uzavření. [12]

4.1.6.3 Filebeat – udržení stavu souborů

Filebeat udržuje stav každého souboru a často ho ukládá na disk do souboru s registrem. Stav se používá k pamatování si posledního offsetu, ze kterého harvester četl a k zajištění, že všechny záznamy jsou odeslány. Pokud není k dispozici definovaný výstup (Elasticsearch, Logstash, nebo jiné aplikace) Filebeat sleduje poslední odeslané řádky a bude pokračovat ve čtení souborů, jakmile bude výstup znovu dostupný. Za běhu Filebeatu jsou informace o stavu také udržována v paměti pro každý vstup. Pokud dojde k restartování aplikace, data ze souboru s registrem jsou použita k obnovení stavu a Filebeat pokračuje ve sběru na poslední známé pozici. [12]

Pro každý vstup Filebeat udržuje stav každého nalezeného souboru. Přestože mohou být soubory přejmenovány nebo přesunuty, název souboru a cesta nestačí

k jednotné identifikaci souboru. Proto si Filebeat ukládá vlastní jedinečné identifikátory, aby zjistil, zdali bylo ze souboru již dříve sbíráno. [12]

4.1.6.4 Filebeat – zajištění at-least-once doručení

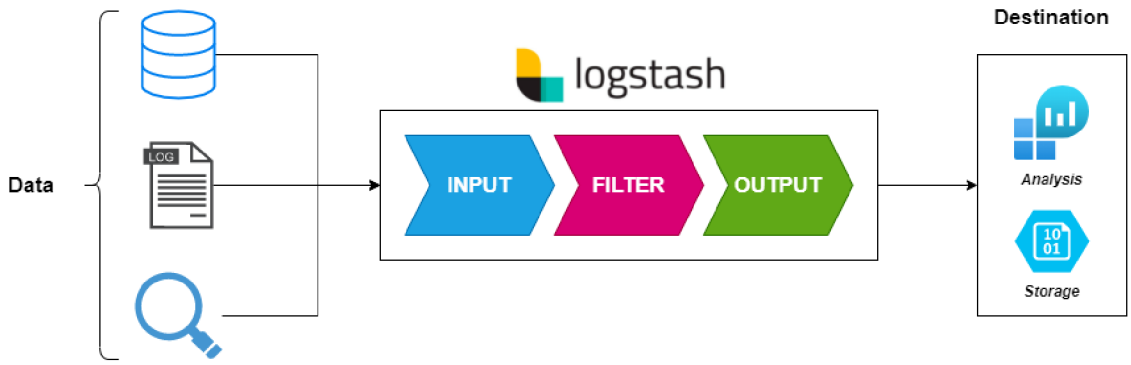
Filebeat zaručuje, že každá událost bude doručena na konfigurovaný výstup alespoň jednou, a to bez ztráty dat. Tohoto chování je dosaženo ukládáním stavu doručení každé události do souboru s registrem. V situacích, kdy je definovaný výstup blokován, a ještě nepotvrdil všechny události, FIlebeat bude pokračovat v pokusu o odeslání události, dokud nebude výstupem potvrzené obdržení.

Pokud se Filebeat vypne během procesu odesílání, nečeká na potvrzení výstupu o všech událostech před vypnutím. Jakékoliv události, které byly odeslány na výstup, ale nebyly potvrzeny před vypnutím, budou ztraceny a znovu odeslány po nastartování aplikace. To zajistí, že každá událost je odeslána alespoň jednou, ale můžeme se setkat s duplicitami na výstupu. [12]

4.2 Logstash

Logstash je open source engine pro sběr dat se schopností reálného časového zpracování. Dokáže dynamicky sjednotit data z různých zdrojů a normalizovat je do cílů podle specifikovaného výběru. Pročišťuje a transformuje veškerá data pro pokročilé analýzy a vizualizační scénáře následného zpracování.

Přestože Logstash původně přinesl inovaci do sběru logů, jeho schopnosti jsou mnohem pokročilejší než tento případ užití. Jakýkoliv typ události lze obohatit a transformovat pomocí široké škály doplňků pro vstup, pomocí filtrů a také obsahuje spoustu nativních kodeků, které dále zjednodušují proces zpracování. [13]



Obrázek 4 - Logstash pipeline. Zdroj: [4]

4.2.1 Jak Logstash funguje

Logstash pipeline pro zpracovávání událostí má tři fáze: vstupy → filtry → výstupy. Vstupy generují události, filtry je modifikují a výstupy odesílají jinam. Vstupy a výstupy podporují kodeky, které umožňují kódovat nebo dekódovat data při vstupu nebo výstupu z pipeline, aniž by bylo třeba použít samostatný filtr. [14]

4.2.1.1 Vstupy

Vstupy se používají k tomu, aby se data dostala do Logstash. Některé z častěji používaných vstupů jsou:

- File
    - Čte soubor, který je umístěný na souborovém systému. K tomu využívá metody *tail – 0F* podobně jako je to na UNIXových systémech.
- Syslog
    - Naslouchá na defaultním portu 514 pro syslog zprávy a parsuje je podle formátu RFC3164.
- Redis
    - Čte z Redis serveru, používá jak redis kanály, tak redis seznamy. Redis je často používán jako broker v centralizované instalaci Logstash, která frontuje události ze vzdálených Logstash „shippers“.
- Beats
    - Zpracovává události, které byly odeslány na vstup pomocí Beats.

4.2.1.2 Filtry

Filtry jsou prostřední zařízení pro zpracovávání v Logstash pipeline. Filtry lze kombinovat s podmínkami pro provedení akce podle nějaké události, pokud jsou splněna určitá kritéria. Některé filtry zahrnují:

- Grok
    - Parsování a strukturování libovolného textu. V současné době se jedná o nejlepší způsob, jak provádět parsování nestrukturovaných logovacích dat do strukturované a dotazovatelné formy. Logstash

momentálně obsahuje 120 vestavěných vzorů pro parsování pomocí Grok.

- Mutate
    - Dovoluje provádět obecné transformace. Můžete přejmenovat, odstranit, nahradit a upravit jednotlivá pole událostí.
- Drop
    - Pomocí této metody lze kompletně zahodit událost, například pokud se jedná o ladící zprávu, která není potřebná k dalšímu zpracovávání.
- Clone
    - Dovoluje nám vytvořit kopii události, případně přidat nebo odebrat pole.
- Geoip
    - Slouží k přidání informací o geografické poloze IP adres

4.2.1.3 Výstupy

Výstupy jsou závěrečnou fází Logstash pipeline. Událost může projít skrz více výstupů, ale jakmile je všechen výstupní proces dokončen, událost ukončila svoje provedení. Mezi běžné výstupy patří [14]:

- Elasticsearch
- Graphite
- File
    - Slouží k zapsání událostí na disk do souboru
- statsd
    - Dovoluje odeslat událostní data do statsd, služby, která "naslouchá statistikám, jako jsou čítačky a časovače, posílané přes UDP, a odesílá agregace do jedné nebo více zapojitelných backendových služeb".

4.2.1.4 Kodeky

Kodeky jsou proudové filtry, které mohou fungovat jako součást vstupu nebo výstupu. Umožnují snadno oddělit transport zpráv od procesu serializace. Mezi nejpoužívanější patří JSON, MSGPACK nebo plain. [14]

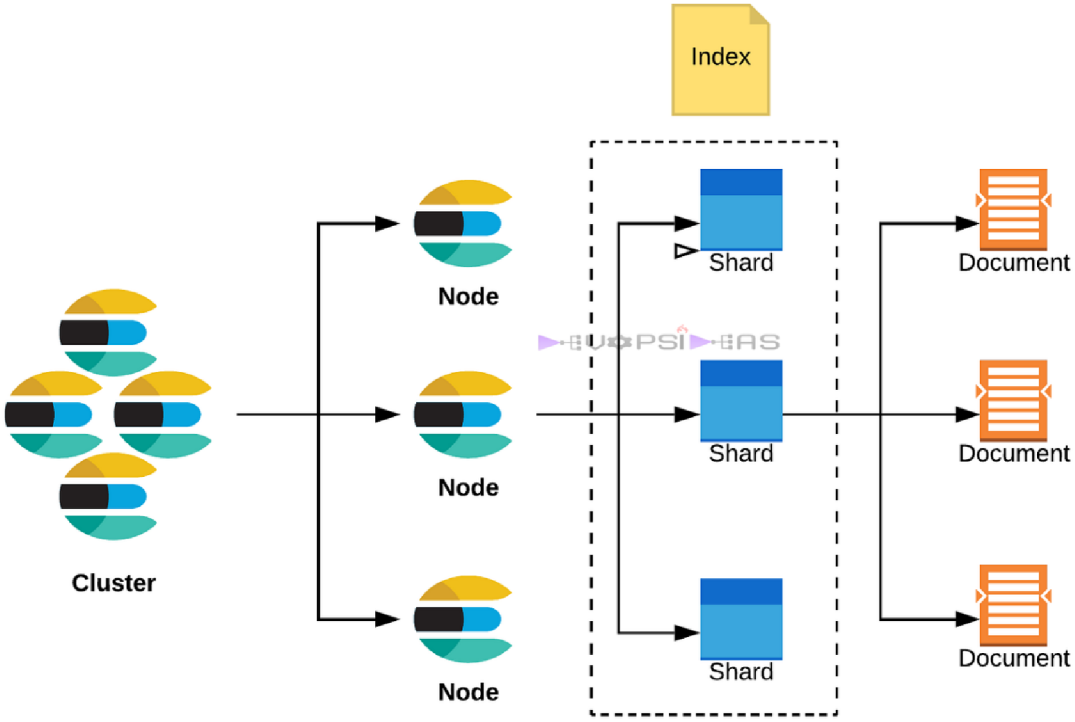
4.3 Elasticsearch

Elasticsearch je distribuovaný vyhledávací a analytický engine, který je nenahraditelnou částí Elastic Stacku. Nabízí velice rychlou a efektivní databázi, která se specializuje na real time full-text vyhledávání. Jeho jádro je postavené na Apache Lucene, což je svobodný vyhledávač napsaný v Javě a dostupný jako knihovna do jiných projektů. Elasticsearch se vyznačuje svojí spolehlivostí, dostupností a horizontální i vertikální škálovatelností. Komunikace napříč databází je dosaženo pomocí JSON formátu skrze RESTful API. [15]

Ať už se jedná o strukturovaný text, číselná data, nebo geografická data, Elasticsearch je schopen je efektivně ukládat a indexovat tak, aby podporoval rychlé vyhledávání.

4.3.1 Základní architektura Elasticsearch

Elasticsearch se opírá o několik základních komponent, které tvoří jeho architekturu. Ta je konstruována tak, aby zajistila efektivní ukládání, rychlé vyhledávání a zpracování dat v prostředí, kde pracuje jeden nebo více uzlů společně k dosažení těchto cílů. [15]



Obrázek 5 - Ukázka architektury. Zdroj: [5]

4.3.1.1 Cluster

Cluster je základním stavebním blokem Elasticsearch architektury. Je to soubor jednoho nebo více uzlů, které jsou logicky spojeny do celku. Tento celek pracuje kooperativně, aby umožnil efektivní ukládání a vyhledávání dat. Každý cluster je identifikován svým unikátním jménem a obsahuje všechny informace o indexech, uzlech a dalších nastavení pro správný provoz Elasticsearch. [15][19]

4.3.1.2 Node (uzel)

Uzel představuje jednu konkrétní instanci Elasticsearch, která běží na fyzickém nebo virtuálním stroji. Každý uzel má svůj identifikátor, pomocí kterého je propojen ke clusteru. Uzly spolu komunikují a koordinují svoji činnost, což umožňuje správu dat, indexů a vyhledávacích operací. [15][17]

4.3.1.3 Index

Index reprezentuje kolekci dokumentů, které mají podobné charakteristiky nebo strukturu. Každý dokument je tvořen JSON objektem, který v sobě obsahuje

samotná data. Indexy jsou klíčovým prvkem pro vyhledávání a filtrování, neboť obsahují metadata o dokumentech a umožňují rychlý přístup k datům. [15] [18]

4.3.1.4 Shard

Shard je jednotka, do které může být rozdělen každý index. Shard obsahuje část dat indexu a slouží k distribuci dat napříč jednotlivými uzly v celém clusteru. To zajištuje rovnoměrnou zátěž na celý cluster a umožňuje nám horizontální škálovatelnost databáze. [15] [19]

4.3.1.5 Replica

Replika je kopie shardu, která slouží k ochraně dat proti chybám. Každý shard může mít jednu nebo více replic, což nám zajištuje ochranu proti ztrátě dat v případě, že jeden z uzlů selže. Pomocí replic také můžeme využít paralelního čtení, což nám poskytne vyšší výkon při vyhledávání dat. [15] [19]

4.3.2 Mapování dat (Mapping)

Mapování je proces definování způsobu, jakým jsou dokumenty a pole uložena a indexována. Každý dokument je sbírkou polí, z nichž každé má svůj vlastní datový typ. Při mapování dat se vytváří mapová definice, která obsahuje seznam polí relevantní pro daný dokument. Mapovací definice také zahrnuje metadata pole, jako například pole _source, které upravují způsob, jak je s metadaty dokumentu zacházeno. Existují dvě metody pro definování mapování dat [15][16]:

- Dynamické
- Explicitní

4.3.2.1 Dynamické mapování

Dynamické mapování umožňuje experimentovat a zkoumat data při začátku práce s Elasticsearch, který automaticky přidává nová pole pouze tím, že indexuje dokument. Pole lze přidat na úrovni celého dokumentu, ale také do vnitřních objektů a vnořených polí. Pro dynamicky přidaná pole lze použít dynamické šablony, které definují vlastnosti mapování podle shodných podmínek. [15][16]

4.3.2.2 Explicitní mapování

Explicitní mapování umožňuje definovat mapovací definici. Mimo jiné lze například určit:

- Která řetězcová pole by měla být považována za pole pro plný text
- Která pole obsahují čísla, data nebo geografické informace
- Formát datových hodnot
- Vlastní pravidla pro řízení mapování pro dynamicky přidaná pole

Elasticsearch také dovoluje použití pole runtime k provádění změn ve schématu bez potřeby reindexace. Runtime pole lze kombinovat s indexovanými poli pro vyvážení zátěže a výkonu. [15][16]

4.3.2.3 Prevence mapovací exploze

Definování příliš mnoha polí v indexu může způsobit mapovací explozi. To má za následek chyby paměti a další problémy, ze kterých je těžké a někdy i nemožné se zotavit. Z tohoto důvodu je dobré použít nastavení omezení mapování, aby docházelo k omezení počtu polí, ať už vytvořených ručně nebo dynamicky, a zabránění vzniku mapovací exploze. Tímto způsobem si nad indexem lze zachovat kontrolu a minimalizovat rizika přetížení při mapování. [15] [16]

4.3.3 Hledání v datech

Hledání v datech probíhá pomocí query, to si lze představit jako žádost o data, která jsou uložena v indexech Elasticsearch. K tomuto hledání nám napomáhá API, které je napsané v jazyce Query DSL (Domain Specific Language). Ta je založena na JSON syntaxi, pomocí které jednotlivé dotazy definujeme.

Následující příklad nám vyhledá v indexu *my-index-000001* takové dokumenty, které mají hodnotu user.id rovnou „kimchy“

```
GET /my-index-000001/_search {
  "query": {
    "match": {
      "user.id": "kimchy"
```

```
  }
 }
}
```

Odpověď je vrácena ve formátu JSON a obsahuje 10 nejlepších dokumentů, které odpovídají dotazu. [15][20]

4.3.4 Zpracovávání proudů dat (Ingest pipeline)

Ingest pipeline umožňuje provádět běžné transformační operace nad daty ještě před tím, než jsou indexována. Pomocí toho lze z dat odstranit pole, extrahovat hodnoty z textu nebo je přidat. [15] [21]

Pipeline se skládá z několika konfigurovatelných úkolů, nazývané procesory. Každý procesor běží postupně a provádí specifické změny na přicházejících dokumentech. Poté co procesory dokončí svoji činnost, Elasticsearch přidá transformovaná data do indexu. [15] [21]

Obrázek 6 - Ingest pipeline. Zdroj:[6]

4.3.5 Analýza textu

Analýza textu umožňuje Elasticsearch provádět vyhledávání plného textu, kde je navrácen všechen obsah, který má relevantní výsledky, ne pouze přesnou schodu. Toho je docíleno pomocí tokenizace a normalizace. [15] [22]

4.3.5.1 Tokenizace

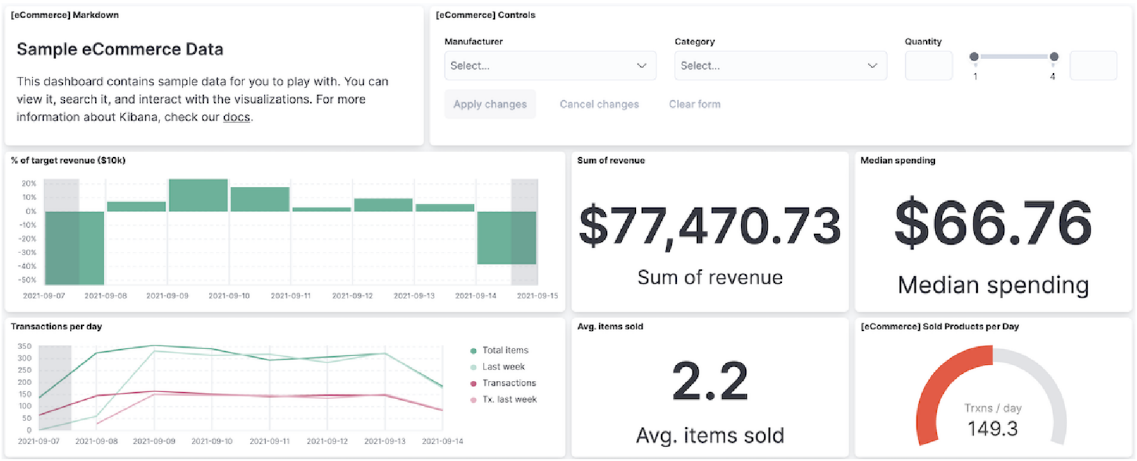
Tokenizace spočívá v rozložení textu na menší části nazývané tokeny. Ve většině případů jsou to jednotlivá slova. Pokud bychom například indexovali text „quick brown fox jumps“ jako celý řetězec a uživatel by vyhledal pouze „quick fox“, nebylo by to považováno za schodu. Pokud ale celou frázi tokenizujeme a každé slovo uložíme zvlášť, můžeme dosáhnout schody například frázemi „quick fox“, „brown fox“ nebo jinými variantami. [15]

4.3.5.2 Normalizace

Tokenizace nám umožňuje indexování jednotlivých slov textu, ale každý token je stále sledován doslovně. To představuje problém, pokud bychom měli uložený token, který začíná velkým písmenem a do vyhledávacího řetězce bychom napsali malé. V tomto případě by ke shodě nedošlo. Pro vyřešení tohoto problému můžeme text normalizovat do standardního formátu. To nám dovoluje nalezení termínů, které nejsou přesně stejné jako vyhledávané termíny, ale jsou si dostatečně podobné na to, aby byly stále relevantní. [15]

4.4 Kibana

Kibana je open-source nástroj pro vizualizaci a analýzu dat od společnosti Elastic. Hlavním účelem je umožnit uživatelům interaktivně prozkoumat, analyzovat a prezentovat data z různých zdrojů, především pak z databáze Elasticsearch. [23]



Obrázek 7 - Kibana Dashboard. Zdroj: [7]

4.4.1 Vizualizace dat

Jednou z nejdůležitějších funkcí Kibana je její schopnost tvořit širokou škálu vizualizací. Uživatelé mohou tvořit různé grafy, což umožňuje zobrazení dat v různých formátech a rychle identifikovat klíčové informace. [23]

4.4.2 Dashboards

Uživatelé mohou za využití dashboards snadno vytvářet interaktivní prostředí pro vizualizaci a analýzu dat. Tyto prezentace klíčových metrik lze sledovat v reálném čase a díky možnosti různých úprav lze vytvořit ideální pohled na data pro dané použití. [23]

4.4.3 Hledání a filtrování

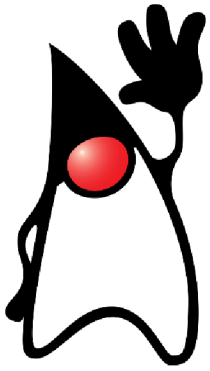
Funkce „Discover“ umožňuje uživatelům snadno procházet, hledat a filtrovat data. Uživatelé mohou využít dotazů a použít filtry k vyhledání konkrétních záznamů nebo událostí v rozsáhlých datových souborech. Tato funkce je velice užitečná při hledání dat pro identifikaci příčin incidentů. [23]

4.4.4 Geografická vizualizace

Pro data, která s sebou nesou geografickou informaci lze vytvářet mapové vizualizace. Uživatelé mohou zobrazovat data na mapách podle geografické polohy a provádět analýzu na základě místa. [23]

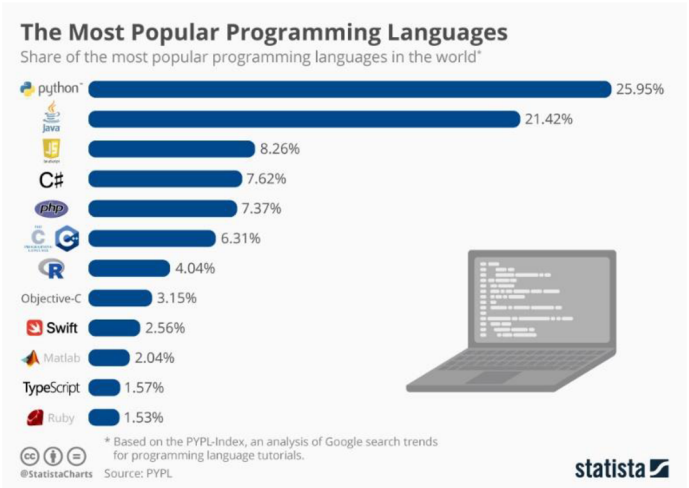
5 Java

Java je vysokoúrovňový objektově orientovaný programovací jazyk známý svojí schopností kompilace bytekódu, který je nezávislý na platformě. Byla navrhnuta Jamesem Goslingem v roce 1990 ve firmě Sun Microsystems. Její první demonstrace proběhla na STAR7 PDA (Personal Digital Assistant), díky kterému se zrodil první maskot Javy, Duke. [24] [25]



Obrázek 8 - Java Duke. Zdroj: [8]

V dnešní době se jedná o jeden z nejvíce populárních jazyků, který sahá do několika oblastí. Pomocí frameworku Spring je možné tvořit webové aplikace, velké datové pipeline pomocí Hadoop a mobilní aplikace na Android. Java také našla své umístění ve vesmírné vědě, kde pohání ovladač Mars Roveru od Nasa. [24] [25]



Obrázek 9 - Graf popularity programovacích jazyků. Zdroj: [9]

Java byla velice inovativní ve svém přístupu kompilace. Místo kompilování přímo do strojového kódu, jako například C nebo C++, je kód kompilován do bytekódu, který může být spuštěn na jakémkoliv operačním systému bez potřeby rekompilace. Této funkcionality je dosaženo pomocí spouštění kódu v JVM (Java

Virtual Machine). Pro běh Java aplikace nám tedy stačí pouze operační systém, který má nainstalované JRE (Java Runtime Environment), to pro programátora znamená „napiš jednou, spusť všude“. [24] [25]

Java je silně typovaný jazyk, který využívá curly-brace syntaxi, podobně jako rodina C. Oproti C má ale spoustu funkcí jako Garbage Collector, Runtime Type Checking a Reflection. [24] [25]

V dnešní době je vlastníkem oficiální implementace Java SE platformy firma Oracle. Tato implementace je založena na původní implementaci od Sun Microsystem. Je dostupná na několika platformách jako Windows, MacOS a Linux. Vzhledem k tomu, že Java postrádá jakékoliv formální standardizace, je tato implementace od firmy Oracle brána jako standard. [24] [25]

5.1 Distribuce Java

Implementace Oraclu je rozdělena do dvou distribucí. Java Runtime Environment, která obsahuje pouze tu část Javy potřebné pro běh programů. Tato verze je určena primárně pro koncové uživatele.

Druhým balíkem je Java Development Kit (JDK). Ten je určen primárně pro vývojáře a obsahuje vývojové nástroje jako je Java Compiler, Javadoc, Jar a debugger. [24]

5.2 Správa paměti – Garbage Collector

Garbage Collection je proces v Javě, který umožňuje programům provádět automatickou správu paměti. Programy se kompilují do bytekódu, který lze spustit na JVM. Uvnitř ní si pak vytvářejí objekty na haldě (heap), což je část paměti, která je rezervovaná programu. Během jeho běhu se často stává, že některé objekty již nejsou potřeba. Garbage Collector tyto objekty vyhledává a maže, aby uvolnil paměť. [26]

V jazycích C nebo C++ je zodpovědnost programátora, jak vytvářet, tak ničit objekty. Ničení nepotřebných objektů bývá často zanedbáváno, což může mít do budoucna negativní dopad na volnou paměť pro nové objekty. Celý program se pak může chovat neobvyklým způsobem, a pokud není tato chyba včas objevena, může

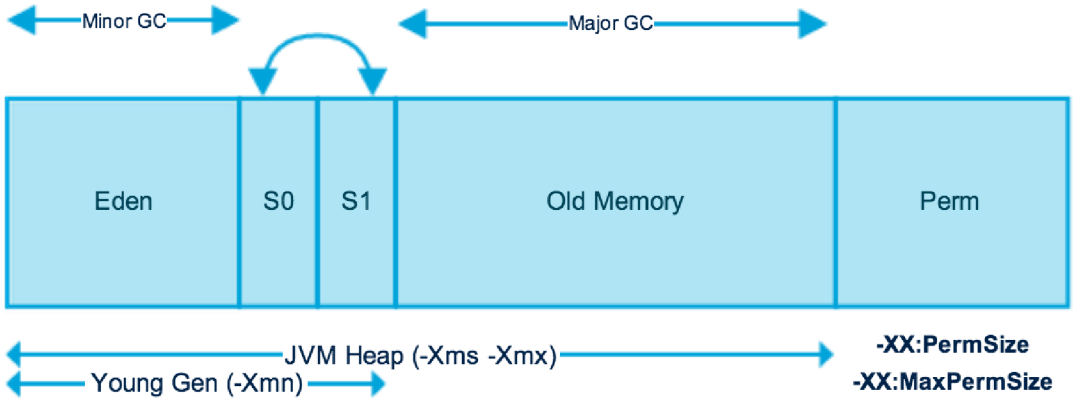
tato nedbalost způsobit Out Of Memory chybu. V Javě se ale programátor o toto ničení nemusí starat, neboť tento proces je automatizován. [26]

Automatický Garbage Collection spočívá v prohlížení paměti haldy, identifikaci objektů, které jsou používání a těch, které už nejsou, a následném mazání. Objekty, které jsou stále používané, nebo je na ně udržen odkaz Garbage Collector vynechává, neboť si na ně nějaká část aplikace stále udržuje referenci. Nepoužití nebo bez odkazový objekt již není využíván žádnou částí programu, a proto je odstraněn a paměť, která byla objektem alokována je uvolněna. Objekty ke smazání programátor nemusí explicitně označovat, neboť implementace Garbage Collectoru je součástí JVM. [26]

Mazání objektů z paměti probíhá dvěma způsoby, minor a major. Minor probíhá, když jsou z paměti mladé generace (young generation heap) odstraněny nepřístupné objekty, tedy objekty, na které není udržována žádná reference. Major se zabývá odstraňováním objektů, které přežili po provedení minor sběru a byly zkopírovány do staré generace (old generation) nebo permanentní generace (permanent generation). Ve srovnání s minor je tento sběr objektů prováděn méně často. [26]

5.3 Rozdělení paměti

V Javě je paměť pro ukládání objektů a dat spravována podle principu haldy. Ta je pak rozdělena do několika dalších částí, které mají různé účely. Mezi hlavní části patří young generation (mladá generace), old generation (stará generace), permanent generation (permanentní generace) a metaspace. [27]



Obrázek 10 - Java, rozdělení paměti. Zdroj: [10]

5.3.1 Young generation heap

Tato část paměti je určena pro krátkodobé uložení nově vytvořených objektů. Skládá se z dalších tří částí: Eden, Survivor 1 a Survivor 2. Eden je místo, kde se inicializují nové objekty. Takové objekty, které přežily několik sběrů odpadu v Eden, mohou být přesunuty do jednoho ze Survivorů. Zde opět podléhají několika sběrům Garbage collectoru. Pokud se jim zde podaří přežít dostatečně dlouho, jsou přesunuty do old generation. [27]

5.3.2 Old generation heap

V této části paměti jsou udržovány objekty, které mají dlouhodobější životní cyklus. Garbage collector v této části pracuje méně často, neboť se zde nacházejí obvykle objekty, které jsou stabilnější. [27]

5.3.3 Permanent generation

Tato oblast byla součástí starších verzí Java. Byla určena pro odkládání metadat tříd, metod, jmen balíčků a dalších informací, které souviseli s během programu. V novějších verzích Java byla tato část nahrazena oblastí metaspace. [27]

5.3.4 Metaspace

Metaspace je následníkem permanentní generace, která se objevuje v novějších verzích Java. Tato část paměti slouží, jako svůj předchůdce, k ukládání metadat o třídách, metod, balíčcích a dalších runtime informací. Hlavním rozdílem,

který metaspace má oproti permanentní generace je dynamická velikost, kterou není potřeba ručně konfigurovat. [27]

5.4 Syntaxe

Syntaxe Javy je do značné míry inspirována C++. Oproti C++, které kombinuje syntaxi pro strukturované, generické a objektově orientované programování, je Java téměř výhradně postavená jako objektově orientovaný jazyk. Pravdou však zůstává, že od verze 8, je podporované funkční programování a lambda metody.

Dalším rozdílem oproti C++ je, že Java nepodporuje přetěžování operátorů nebo vícenásobnou dědičnost pro třídy. To jazyk poměrně zjednodušuje a pomáhá při prevenci potenciálních chyb.

Komentování v Javě je stejné jako u většiny programovacích jazyků z rodiny C. Pomocí dvou lomítek (//) můžeme vytvořit jednořádkový komentář, víceřádkový komentář otevřeme pomocí lomítka a hvězdičky a uzavřeme pomocí hvězdičky a lomítka (/* */). Pokud chceme vytvořit Javadoc, který programátorovi umožňuje sestavit dokumentaci programu, uvodíme ho pomocí lomítka a dvou hvězdiček (/** */).

5.4.1 Ukázka kódu

Jak jsme si již řekli, každý Java program začíná ve třídě, jejíž jméno by se mělo shodovat s názvem souboru [28].

```
public class HelloWorld {
    public static void main (String[] args) {

    }
}
```

Každý program by měl obsahovat alespoň jednu metodu main, odtud program začíná svoje provádění. Uvnitř metody main můžeme definovat proměnné a volat ostatní metody programu.

Třídy nám také mohou posloužit jako šablony pro tvorbu objektů. Do nich můžeme přidávat atributy a metody. Pomocí klíčového slova new, můžeme tvořit objekty, instance, těchto tříd.

5.5 Proměnné

Java obsahuje různé typy proměnných. Každá proměnná musí obsahovat jakého je typu a měla by mít přiřazenou nějakou hodnotu, nebo hodnotu null. Rozdělují se do několika typů, které říkají, jak s nimi má být nakládáno a kolik paměti mohou alokovat.

5.5.1 Primitivní datové typy

Java má osm základních primitivních datových typů. Jsou jimi: int, byte, short, long, float, double, boolean a char. Proměnné těchto typů nejsou považovány za objekty a představují hrubé hodnoty [29].

Typ	**Velikost** (bit)	**Minimum**	**Maximum**	**Příklad**
byte	8	-2^7	2^7-1	byte b = 100;
short	16	-2^{15}	$2^{15}-1$	
int	32	-2^{31}	$2^{31}-1$	
long	64	-2^{63}	$2^{63}-1$	
float	32	$-2-^{149}$	$(2-2^{-23})*2^{127}$	
double	64	-2^{-1074}	$(2-2^{-52})*2^{1023}$	
char	16	0	$2^{16}-1$	
boolean	1	-	-	

5.5.1.1 byte

byte je primitivní datový typ, který je velice podobný int s tím rozdílem, že pouze zabírá 8 bitů paměti. To je také důvod, proč se nazývá byte. Kvůli své malé velikosti dokáže uchovávat hodnoty v rozsahu -128 (-2^7) do 127 ($2^7 - 1$). [29]

5.5.1.2 short

Dalším primitivním datovým typem je short. Pokud chceme ušetřit paměť a byte je příliš malý, tento typ nabízí alternativu, neboť v rámci alokování paměti je

uprostřed mezi byte a int. Se svými 16 bity paměti má poloviční alokaci oproti int a dvojnásobnou oproti byte. Jeho rozsah možných hodnot je od -32,768 (-2^{15}) do 32,767 ($2^{15} - 1$). [29]

5.5.1.3 int

Jedním z nejčastěji používaných datových typů je int. Dokáže v sobě udržet širokou škálu celočíselných hodnot bez desetinné čárky. Uložen je pomocí 32 bitů paměti, tudíž v sobě dokáže držet hodnoty od -2,147,483,648 (-2^{31}) do 2,147,483,647 ($2^{31} - 1$). [29]

5.5.1.4 float

V Javě reprezentujeme základní desetinná čísla pomocí typu float. Jedná se o desetinné místo s jednoduchou přesností, to znamená, že při překročení šesti desetinných míst se ztrácí přesnost a stává se více odhadem. Tyto ztráty nám většinou nevadí, ale pokud naše výpočty vyžadují absolutní přesnost (například finanční operace), musíme použít specifické typy navržené přímo pro tuto práci. [29]

Tento typ je ukládám pomocí 32 bitů, stejně jako int. Nicméně kvůli pohyblivé desetinné čárce je jeho rozsah naprosto odlišný. Může reprezentovat kladná i záporná čísla, kde nejmenší je $1.40239846 \times 10^{-45}$ a největší $3.40282347 \times 10^{38}$. [29]

5.5.1.5 double

Dále se podíváme na typ doule. Jeho název pochází ze skutečnosti, že se jedná o desetinné číslo s dvojitou přesností (double precision). Je ukládán v 64 bitech, a reprezentuje mnohem širší rozsah čísel než float, přesto však čelí stejnému omezení přesnosti jako float. Rozsah možných hodnot je od $4.9406564584124654 \times 10^{-324}$ až $1.7976931348623157 \times 10^{308}$. [29]

5.5.1.6 boolean

Jedná se o nejjednodušší primitivní datový typ. Může nabývat pouze dvou hodnot, true nebo false. Je ukládám pomocí jednoho bitu paměti. [29]

5.5.1.7 char

Posledním primitivním datovým typem je char. Také nazývaný jako znak, je 16bitové celé číslo, které reprezentuje znak zakódovaný v Unicode. Jeho rozsah je od 0 do 65535. V Unicode je to přeloženo jako od znaku \u0000 do \uffff. [29]

5.5.1.8 Přetečení (Overflow)

Primitivní datové typy mají své omezení velikosti. Pokud se do proměnné pokusíme uložit větší hodnotu, než jaká je dovolená pamětním omezením, dostaneme se do situace zvané přetečení. Pokud k tomuto jevu dojde například u celočíselného typu int, hodnota se obrátí na minimální hodnotu a od této hodnoty se začne se zvětšovat. U typů s plovoucí desetinou čárkou dostaneme hodnotu Infinity. [29]

```
int i = Integer.MAX_VALUE;
int j = i + 1;
// j přeteče na -2_147_483_648

double d = Double.MAX_VALUE;
double o = d + 1;
// o bude Infinity
```

Podobně jako přetečení, může dojít k opačnému jevu Underflow. To se týká ukládání menších hodnot, než je povolená minimální hodnota. Pokud k podtečení dojde, proměnné budou vracet hodnotu 0.0. [29]

5.5.1.9 Autoboxing

Každý primitívní datový typ má také svou plnou implementaci Javové třídy, která jej může obalit. Například třída Integer obaluje hodnoty typu int. Někdy je třeba provádět konverzi z primitivního typu na objektový, například při používání generik.
Tato konverze je prováděna automaticky pomocí autoboxingu.

```
Character c = 'c';
Integer i = 1;
```

Pomocí něj lze jednoduše přiřadit hodnotu primitivního datového typu do odpovídajícího objektového obalu, a Java vyková veškerou práci na pozadí. [29]

5.5.2 Neprimitivní datové typy

Mimo primitivních typů se v Javě často setkáme s neprimitivními, nebo také referenční datové typy. Místo toho, aby obsahovaly samotné hodnoty, obsahují odkazy na objekty. Tyto proměnné se používají pro práci s komplexními datovými strukturami a uživatelsky definovanými objekty. Na rozdíl od primitivních proměnných mohou odkazovat na instance tříd, pole, rozhraní a další referenční typy. [30]

5.5.2.1 Třídy a objekty

Jedním z klíčových konceptů v programování v Javě je použití tříd a jejich instancí neboli objektů. Třída definuje vlastnosti a chování, nebo atributy a metody, které budou mít objekty vytvořené z této třídy. Objekty slouží jako zapouzdření těchto atributů a tříd, což podporuje modularitu a znuvupoužitelnost kódu. [30] [31]

```
public class Person {

  private final String name;
  private final String surname;
  private final int age;

  public Person (String name, String surname, int age) {
    this.name = name;
    this.surname = surname;
    this.age = age;
  }
```

```
public void shout () {
    System.out.println("Hello World");
  }
}
```

Takto například může vypadat třída, která se jmenuje Person a definuje atributy name, surname a age. Pomocí konstruktoru pak můžeme vytvořit instanci tohoto objektu a atributy naplnit hodnotami.

```
Person person = new Person ("Václav", "Novák", 26);
```

Takto vypadá vytvořený objekt třídy Person. Pokud bychom chtěli jednotlivé atributy z objektu získat, můžeme tak učinit dopsáním gettrů (getters) do třídy, nebo naopak, pokud bychom chtěli hodnotu nastavit, nebo přepsat bez využití konstruktoru, můžeme tak učinit pomocí settrů (setters).

Pokud bychom chtěli zavolat metodu shout, můžeme tak učinit následovně.

```
person.shout();
```

5.5.2.2 Pole

Pole je sbírka prvků stejného datového typu, uspořádaná sekvenčně. Pole poskytují pohodlný způsob práce s více hodnotami stejného typu. Jsou deklarována pomocí hranatých závorek za datovým typem. Mohou v sobě obsahovat jak primitivní datové typy, tak objekty. [31]

```
int[] numbers = new int[5];
```

V tomto případě jsme vytvořili pole, které v sobě může obsahovat proměnné typu int a může obsahovat pět prvků.

5.5.2.3 Řetězce

Řetězce jsou objekty, které se používají k reprezentaci posloupnosti znaků. I když jsou velice běžně používány, nejsou primitivní datové typy. Jsou instancí třídy String a disponují mnoha doplňujícími metodami pro zjednodušení práce s textem. [31]

Řetězce jsou v Javě definovány jako neměnné (immutable), což znamená, že jejich se sdílí v jediném společném úložišti (pool) za účelem minimalizace kopírování stejných hodnot. [32]

Existuje několik důvodů, proč jsou Stringy takto navrženy. Mezi hlavní důvody paří faktor, že pokud by byly Stringy proměnné, nemohl by být využíván pool. To by vedlo k zvýšení potřebné paměti. Dalším důvodem je zabezpečení. Například přihlašovací jména nebo hesla jsou předávána jako String. Protože je String neměnný, nelze jejich hodnotu změnit. Pokud by nezměnitelnost řetězců nebyla zajištěna, mohl by útočník změnit odkaz na hodnotu a způsobit tak bezpečnostní problémy v aplikaci. [32]

5.5.2.4 Uživatelem definované referenční typy

Kromě vestavěných referenčních typů je také možné definovat vlastní pomocí tříd. To umožňuje vytvářet složitější datové struktury přizpůsobené potřebám aplikace. [31]

5.5.2.5 Odkazování na null hodnoty

Na rozdíl od primitivních datových typů mohou neprimitivní proměnné obsahovat speciální hodnotu null. Ta označuje, že proměnná momentálně neukazuje na žádný objekt. Pokus o přístup k metodám nebo atributům, které odkazují na null hodnotu může vést k chybě známé jako NullPointerException. [31]

5.6 Přístupové modifikátory (Access modifiers)

V Javě nám přístupové modifikátory slouží k omezení přístupu k proměnným, metodám, nebo i třídám. Poskytují nám další vrstvu zabezpečení v závislosti na použitém modifikátoru s prvkem. Existují celkem čtyři modifikátory. [33]

5.6.1 Výchozí (default)

Když pro třídu nebo metodu neurčíme přístupový modifikátor, říká se, že má implicitně přístupový modifikátor výchozího nastavení. Takové prvky mají dostupnost pouze v rámci stejného balíčku a pro ostatní jsou neviditelné. [33]

5.6.2 Soukromé (private)

Tento přístupový modifikátor slouží k nastavení dostupnosti pouze v rámci svojí třídy. Žádná jiná třída, jak ve stejném, nebo jiném balíčku, nebude schopna přistupovat k takto definovaným prvkům. Třídy nebo rozhraní na úrovní nejvyššího prvku nemohou být takto deklarovány, neboť soukromý znamená „pouze uvnitř obalovací třídy“. Platí tedy, že tento modifikátor lze použít pouze pro vnořené třídy a proměnné v nich. [33]

5.6.3 Chráněné (protected)

Takto definované prvky jsou viditelné pouze v rámci svého balíčku, nebo v podtřídách v různých balíčcích. [33]

5.6.4 Veřejné (public)

Takové prvky jsou dostupné odkudkoliv v rámci celého programu. [33]

6 Maven

Maven je nástroj pro správu projektů a balíčků v jazyce Java. Umožňuje automatizaci procesů, jako je kompilace, testování a sestavení. Jeho hlavním cílem je zjednodušení vývoje a správy projektů a zajistit všechny potřebné závislosti. [33]

Funguje pomocí konfiguračního souboru – pom.xml, který obsahuje důležité informace o projektu, jako název, verzi, závislosti a další metadata. Pomocí této konfigurace dokáže automaticky stáhnout a spravovat všechny potřebné knihovny a závislosti pro projekt. [34]

Maven se také dokáže postarat o multi-modulární projekty, což jsou takové projekty, které sestávají z více samostatných modulů. Tento přístup umožňuje snadné sdílení jednotlivých modulů, neboť o závislosti na jiných balíčcích je automaticky postaráno. [34]

6.1 Dependence

V pom.xml je možné u každého projektu nadefinovat jeho závislosti na externích knihovnách. Jednotlivé prvky, Artifacts, jsou jednoznačně definovány podle atributů <groupId> a <artifactId>. Maven pak automaticky vyhledá a nainstaluje potřebné knihovny. Samotné vyhledávání probíhá v definovaných úložištích (repository). Kromě globální maven repository, která je veřejně přístupná, je možné založit i další soukromá nebo firemní úložiště. [34]

6.2 Životní cyklus

Maven umožňuje rozdělit proces buildu do více fází. Tento přístup umožňuje nejen automaticky spouštět pluginy, ale také specifikovat fázi, ve které má Maven skončit. Například nám během vývoje stačí zkompilovat zdrojové kódy a otestovat je bez deploye aplikace na server. [34]

Výchozí cyklus maven je nastaven takto:

- process-resources
- compile
- process-test-resources

- test-compile
- test
- package
- install
- deploy

7 Gson

Gson je knihovna pro jazyk Java, kterou lze použít pro převod objektů do jejich reprezentace v JSON formátu. To samé je možné udělat i zpětně, tedy převést JSON řetězec do objektové podoby. Dokáže pracovat s libovolnými objekty jazyka Java, včetně existujících objektů, u kterých není přístup ke zdrojovému kódu. [34]

Existuje několik ostatních open-source projektů, které umí převádět objekty jazyka Java do formátu JSON. Nicméně většina z nich vyžaduje, vkládání Java anotací do tříd. To nelze udělat, pokud není přístup ke zdrojovému kódu. Některé z těchto alternativ také nepodporují použití generických typů v Javě. [35]

8 Návrh logovací aplikace

Hlavním účelem aplikace bude tvorba logovacích zpráv. Každých pět sekund se vytvoří nový záznam, který bude obsahovat čas, ve kterém se stal, úroveň vážnosti a náhodnou hodnotu od 0 do 999. Tvorba logu každých pět sekund nám umožní sledovat aktivity aplikace s časovým razítkem. Úroveň vážnosti bude sloužit k rozpoznávání důležitých událostí od těch, která jsou pouze informativní. V poslední řadě, náhodná hodnota bude sloužit jakožto teoretický identifikátor logované události.

Aplikace bude běžet jako služba na systému Linux. Spouštěna bude pomocí příkazu systemctl start log_application a bude běžet, dokud nebude uživatelem zastavena.

8.1 Anotace

Tato aplikace bude logovací nástroj, který bude generovat záznamy s časovým razítkem, úrovní vážnosti a náhodnými hodnotami. Bude běžet jako služba na systému Linux a tvořit záznamy každých pět sekund.

Bude napsaná v jazyce Java a bude využívat technologie Maven a Gson. Bude rozdělena do čtyř tříd Severity, Event, EventCreator, App.

Třída Severity bude definovat úroveň vážnosti událostí (INFO, WARN, ERROR) a bude usnadňovat manipulaci s nimi. Třída Event bude zodpovědná za generování a zaznamenávání událostí, které budou obsahovat čas, úroveň závažnosti a náhodné hodnoty. EventCreator bude tvořit logovací záznamy do logovacího souboru v JSON formátu, kde každých pět sekund bude tvořit nový záznam události a zapíše ho do souboru. Nakonec třída App bude obsahovat vstupní bod aplikace a řídit proces generování událostí.

8.2 Technický popis

Použitými technologiemi bude programovací jazyk Java, správce balíčků Maven a Framework Gson. Aplikace bude rozdělena do čtyř tříd, Severity, Event, EventCreator a App, která bude obsahovat hlavní metodu pro spuštění programu.

Finálním výstupem aplikace bude soubor v json formátu v adresáři /var/log/application/app.json.

8.3 Severity

Účelem této třídy je definovat různé úrovně závažnosti událostí, které bude aplikace generovat. Třída je implementována jako výčtový typ (enum) a obsahuje následující závažnosti:

- INFO
- WARN
- ERROR

Každá úroveň závažnosti je reprezentována instancí třídy Severity, která obsahuje vlastnost value typu String. Tato hodnota odpovídá textové reprezentaci dané úrovni závažnosti („INFO“, „WARN“, „ERROR“). Tato vlastnost je nastavena při vytváření instance a může být získána pomocí metody getValue().

Třída je navržena tak, aby usnadňovala kategorizaci a identifikaci závažnosti logovacích zpráv v rámci aplikace. Díky pevně definovaným hodnotám je minimalizována možnost chyb při manipulaci s úrovněmi závažnosti.

8.4 Event

Třída Event slouží ke generování a zaznamenávání událostí. Každých pět sekund je vytvářen nový záznam typu Event, který obsahuje informace o čase události, úrovni závažnosti a náhodné hodnotě

Obsahuje atributy timestamp, severity a randomValue. Timestamp je textová reprezentace času, kdy událost nastala. Severity je reprezentace vážností události. Tato hodnota může nabývat hodnot „INFO“, „WARN“ a „ERROR“. RandomValue je náhodně generovaná číselná hodnota v rozmezí od 0 do 999, která slouží k ilustraci dat, která mohou být spojena s danou událostí.

Uvnitř třídy je statická veřejná metoda, která slouží k vytváření nových instancí třídy Event, generuje náhodně úroveň závažnosti na základě generovaného čísla a aktuálního času. Dále generuje náhodnou hodnotu pro randomValue.

8.5 EventCreator

Třída EventCreator je zodpovědná za vytváření událostí do logovacího souboru. Implementuje rozhraní Runnable, což jí umožňuje být samostatně spuštěna jako samostatné vlákno.

Obsahuje metodu run. Ta je implementací rozhraní Runnable a definuje, co se stane, když je vlákno spuštěno. V této metodě se vytvoří nová instance třídy Event pomocí metody Event.createEvent(). Poté je tato událost serializována do formátu JSON pomocí instance třídy Gson. Nakonec je tento JSON řetězec zapsán do logovacího souboru ve formě, která umožňuje jeho snadné parsování.

Atributy jsou filePath, textová proměnná, která obsahuje cestu k logovacímu souboru. V této proměnné je uložena cesta /var/log/application/app.json, kde jsou zaznamenány události. Poté je zde proměnná gson, instance třídy Gson, která slouží k převodu objektů do formátu JSON a zpět.

8.6 App

App je vstupním bodem celé aplikace. Uvnitř ní, je metoda main, která je zavolána při spuštění aplikace. Pomocí této třídy je řízen průběh zaznamenávání a generování událostí pomocí třídy EventCreator.

V metodě main je vytvořena instance třídy EventCreator, která implementuje rozhraní Runnable. Následně je spuštěna nekonečná smyčka, která volá metodu run(), instance EventCreator. Po každém volání metody run() je vlákno pozastaveno na pět sekund. Pokud během provádění dojde k chybě, je tato chyba vypsána do konzole.

9 Implementace aplikace

- Třída App.java:

```
package event_generator;

public class App {

    public static void main(String[] args) {
        Runnable runnable = new EventCreator();
        while (true) {
            try {
                runnable.run();
                Thread.sleep(5000);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

Hlavní funkcionalitou této třídy je periodické volání metody run na instanci třídy EventCreator. Ta odpovídá za vytváření událostí. Po vykonání této metody se vlákno na 5 sekund uspí, čímž se dosáhne periodického generování událostí.

- Třída EventCreator.java:

```
package event_generator;

import com.google.gson.Gson;

import java.io.BufferedWriter;
import java.io.FileWriter;


public class EventCreator implements Runnable {

    private static final Gson gson = new Gson();
    private static String filePath = "/var/log/application/app.json";

    @Override
    public void run() {
        Event event = Event.createEvent();
        String jsonString = gson.toJson(event);
        try (BufferedWriter br = new BufferedWriter(new
FileWriter(filePath, true))) {
            br.write(jsonString + ",\n");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Třída EventCreator je zodpovědná za vytvoření události a jejího zápisu do souboru v JSON formátu. K tomu je využita knihovna Gson pro serializaci do očekávaného formátu.

- Třída Event.java:

```
package event_generator;

import java.time.LocalDateTime;

public class Event {

    private final String timestamp;
    private final String severity;
    private final int randomValue;

    public Event(String timestamp, String severity, int randomValue) {
        this.timestamp = timestamp;
        this.severity = severity;
        this.randomValue = randomValue;
    }

    public static Event createEvent() {
        int random = (int) (Math.random() * 3) + 1;
        String severity;
        switch (random) {
            case 2 -> severity = Severity.WARN.getValue();
            case 3 -> severity = Severity.ERROR.getValue();
            default -> severity = Severity.INFO.getValue();
        }
        return new Event(LocalDateTime.now().toString(), severity,
(int) (Math.random() * 1000));
    }
}
```

Třída Event obsahuje konstruktor pro vytváření nových instancí událostí a statickou metodu createEvent, která slouží k vytvoření nové události s náhodným časem, závažností a hodnotou.

- Třída Severity.java:

```
package event_generator;

public enum Severity {

    INFO("INFO"),
    WARN("WARN"),
    ERROR("ERROR");

    private final String value;

    Severity(String value) {
        this.value = value;
```

```
    }

    public String getValue() {
        return value;
    }
}
```

Třída Severity je výčtovým typem, který reprezentuje různé úrovně závažnosti události. Každá z těchto závažností má přidělenou hodnotu typu String, která se využívá k identifikaci závažnosti při vytváření události.

10 Instalace a konfigurace

V této kapitole se zaměříme na instalaci a konfiguraci všech výše popsaných komponent. Jako první si ukážeme instalaci Filebeatu, poté přejdeme na ELK stack v pořadí Logstash, Elasticsearch a naposled Kibana. V ní si ukážeme data sebraná z vytvořené aplikace a popíšeme si, jak se v celém stacku pohybovat.

10.1 Vytvoření služby

Pro vytvoření služby na systému Linux budeme potřebovat spustitelnou Java aplikaci. Tu si můžeme vygenerovat pomocí Mavenu. V nabídce Maven klikneme na Lifecycle. To nám zobrazí další možnosti, ze kterých si vybereme package. To nám vytvoří adresář target, který obsahuje náš finální jar soubor.

Poté se přesuneme do adresáře /etc/systemd/system, kde si vytvoříme soubor application.service, který doplníme následujícím obsahem.

```
[Unit]
Description=Bachelors Thesis Application
After=network.target

[Service]
User=root
Password=password
ExecStart=/usr/bin/java -jar /home/martin/IdeaProjects/BachelorsThesis/target/BachelorsThesis.jar

[Install]
WantedBy=multi-user.target
```

Obrázek 11 - Linux služba. Zdroj: vlastní

Po uložení souboru pomocí příkazu systemctl damon reload znovu načteme systémové služby. Příkazem systemctl start application.service můžeme naši aplikaci spustit.

10.2 Filebeat

Prvním krokem bude získání instalačního balíčku. To můžeme udělat na oficiálních stránkách Elastic. Na výběr máme dvě cesty. Buď si balíček stáhneme manuálně a pak ho přeneseme na server, nebo použijeme APT package manager. V tomto případě zvolíme druhou cestu, neboť je o něco rychlejší a jednoduší.

1. Začneme stažením veřejného podpisového klíče:

```
martin@rat-VirtualBox:~$ wget -qO - https://artifacts.elastic.co/GPG-KEY-elastic
search | sudo apt-key add -
```

Obrázek 12 – Filebeat, stažení veřejného klíče. Zdroj: vlastní

2. Poté si přidáme instalační balíček do definice repositářů:

```
martin@rat-VirtualBox:~$ echo "deb https://artifacts.elastic.co/packages/8.x/apt
 stable main" | sudo tee -a /etc/apt/sources.list.d/elastic-8.x.list
```

Obrázek 13 – Filebeat, přidání repositáře. Zdroj: vlastní

3. Posledním krokem už je samotná instalace Filebeatu:

```
martin@rat-VirtualBox:~$ sudo apt-get update && sudo apt-get install filebeat
```

Obrázek 14 – Filebeat, instalace. Zdroj: vlastní

4. Po dokončení instalačního skriptu bychom měli být schopní, pomocí příkazu systemctl zjistit, zdali se balíček nainstaloval:

```
martin@rat-VirtualBox:~$ systemctl status filebeat
○ filebeat.service - Filebeat sends log files to Logstash or directly to Elasti>
     Loaded: loaded (/lib/systemd/system/filebeat.service; disabled; vendor pre>
     Active: inactive (dead)
       Docs: https://www.elastic.co/beats/filebeat
```

Obrázek 15 - Filebeat, dokončení instalace. Zdroj: vlastní

[36]

Pokud celá instalace proběhla správně, měla by se automaticky vygenerovat následující struktura. [37]

Typ	Popis	Umístění
home	Domovská složka	/usr/share/filebeat
bin	Složka binárních souborů	/usr/share/filebeat/bin
config	Konfigurační složka	/etc/filebeat
data	Složka persistentních dat	/var/lib/filebeat
logs	Logovací složka	/var/log/filebeat

Pro účel této práce bude nejvíce zajímavá právě konfigurační složka, ve které se definuje vstup pro logy a veškeré dodatečné nastavení pro sběr. Logovací složka nám poslouží pro zjištění správnosti konfigurace.

Konfigurace Filebeat je velice jednoduchá. V adresáři /etc/filebeat je automaticky vygenerovaný ukázkový soubor konfigurace. Na první pohled se může zdát soubor velice složitý, ale po bližším prozkoumání je pro účel sběru logů důležité pouze specifikovat vstup a výstup. Toho docílíme pomocí filebeat.input a output.logstash. Ostatní nastavení, tedy json.* označuje, jak má Filebeat procesovat JSON formát, který mu byl poskytnut.

```
filebeat.inputs:
  - type: log
    enabled: true
    paths:
      - /var/log/application/app.json
    json.keys_under_root: true
    json.overwrite_keys: true
    json.add_error_key: true
    json.expand_keys: true

output.logstash:
  hosts: ["127.0.0.1:5044"]

logging.files:
  path: /var/log/filebeat
  name: filebeat
  keepfiles: 7
  permissions: 0640
```

Obrázek 16 - Filebeat konfigurace. Zdroj: vlastní

10.3 Elasticsearch

Prvním krokem instalace je opět získání veřejného podpisového klíče. Tento krok může být přeskočen, neboť byl klíč importován při instalaci Filebeatu. Budeme tedy pokračovat přidáním instalačního balíčku do definice repositářů. [38]

```
martin@rat-VirtualBox:~$ echo "deb [signed-by=/usr/share/keyrings/elasticsearch-
keyring.gpg] https://artifacts.elastic.co/packages/8.x/apt stable main" | sudo t
ee /etc/apt/sources.list.d/elastic-8.x.list
```

Obrázek 17 - Elasticsearch, přidání repositáře. Zdroj: vlastní

Poté pomocí apt-get nainstalujeme samotnou aplikaci.

```
martin@rat-VirtualBox:~$ sudo apt-get update && sudo apt-get install elasticsearch
```

Obrázek 18 - Elasticsearch, instalace. Zdroj: vlastní

Po dokončení instalace zkontrolujeme službu pomocí systemctl status elasticsearch.

```
martin@rat-VirtualBox:~$ systemctl status elasticsearch.service
○ elasticsearch.service - Elasticsearch
     Loaded: loaded (/lib/systemd/system/elasticsearch.service; disabled; vendor preset: enabled)
     Active: inactive (dead)
       Docs: https://www.elastic.co
```

Obrázek 19 - Elasticsearch, dokončení instalace. Zdroj: vlastní

10.4 Kibana

Prvním krokem instalace je opět získání veřejného podpisového klíče. Tento krok může být přeskočen, neboť byl klíč importován při instalaci Filebeat. Budeme tedy pokračovat přidáním instalačního balíčku do definice repositářů.

```
martin@rat-VirtualBox:~$ echo "deb [signed-by=/usr/share/keyrings/elasticsearch-
keyring.gpg] https://artifacts.elastic.co/packages/8.x/apt stable main" | sudo t
ee /etc/apt/sources.list.d/elastic-8.x.list
```

Obrázek 20 - Kibana, přidání repositáře. Zdroj: vlastní

Nyní můžeme balíček nainstalovat.

```
martin@rat-VirtualBox:~$ sudo apt-get update && sudo apt-get install kibana
```

Obrázek 21 - Kibana, instalace. Zdroj: vlastní

Poté co je instalace dokončena, zkontrolujeme službu.

```
martin@rat-VirtualBox:~$ systemctl status kibana
○ kibana.service - Kibana
     Loaded: loaded (/lib/systemd/system/kibana.service; disabled; vendor prese>
     Active: inactive (dead)
       Docs: https://www.elastic.co
```

Obrázek 22 - Kibana, dokončení instalace. Zdroj: vlastní

10.5 Logstash

Prvním krokem instalace je opět získání veřejného podpisového klíče. Tento krok může být přeskočen, neboť byl klíč importován při instalaci Filebeat. Budeme tedy pokračovat přidáním instalačního balíčku do definice repositářů. [39].

```
martin@rat-VirtualBox:~$ echo "deb [signed-by=/usr/share/keyrings/elastic-keyrin
g.gpg] https://artifacts.elastic.co/packages/8.x/apt stable main" | sudo tee -a
/etc/apt/sources.list.d/elastic-8.x.list
```

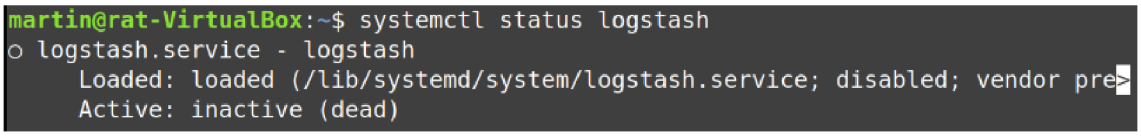
Obrázek 23 - Logstash, přidání repositáře. Zdroj: vlastní

Nyní už můžeme provést samotnou instalaci.

```
martin@rat-VirtualBox:~$ sudo apt-get update && sudo apt-get install logstash
```

Obrázek 24 - Logstash, instalace. Zdroj: vlastní

Po dokončení instalace si opět zkontrolujeme, že se služba nainstalovala správně.



Obrázek 25 - Logstash, dokončení instalace. Zdroj: vlastní

Pokud celá instalace proběhla správně, měla by se automaticky vygenerovat následující struktura.

Typ	Popis	Umístění	Nastavení
home	Domovská složka	/usr/share/logstash	
bin	Složka binárních souborů	/usr/share/logstash/bin	
settings	Umístění konfiguračních souborů	/etc/logstash	path.settings
conf	Konfigurace pipeline	/etc/logstash/conf.d/*.conf	
logs	Umístění logovacích souborů	/var/log/logstash	path.logs
plugins	Umístění pro ostatní pluginy	/usr/share/logstash/plugins	path.plugins
data	Datová složka	/var/lib/logstash	path.data

Nyní se zaměříme na vytvoření Logstash pipeline. V adresáři /etc/logstash/conf.d vytvoříme soubor, do kterého zapíšeme vstup a výstup záznamů. [40]

```
input {
  beats {
    port => 5044
  }
}

output {
  elasticsearch {
    hosts => ["127.0.0.1:9200"]
    index => "bachelors_index"
    user => "elastic"
    password => "O*=ZegS+fYS7EU+FbQu0"
  }
}
```

Obrázek 26 - Logstash, konfigurace. Zdroj: vlastní

10.6 Propojení Elasticsearch a Kibana

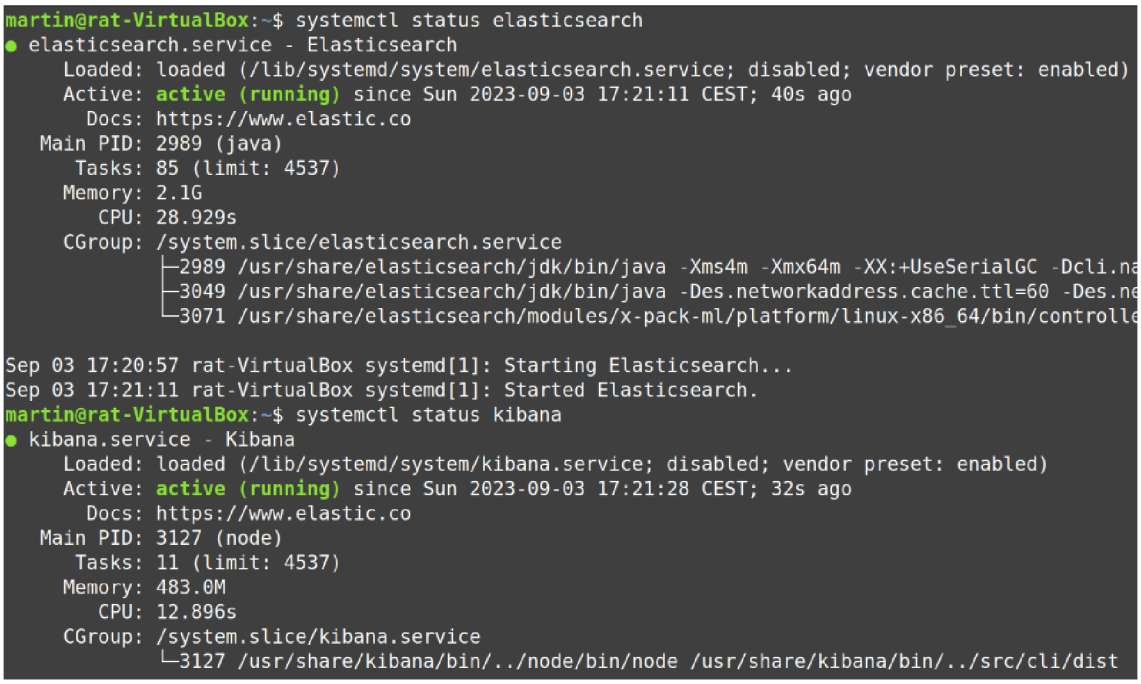
Prvním krokem konfigurace je nastavení naslouchání na ip adresách. V souboru /etc/kibana/kibana.yml nastavíme vlastnost server.host na hodnotu 0.0.0.0. Tím nastavíme naslouchání na celém rozsahu adres. To samé provedeme v souboru /etc/elasticsearch/elasticsearch.yml.

```
network.host: 0.0.0.0
```

Obrázek 27 - Elasticsearch a Kibana, network hosts. Zdroj: vlastní

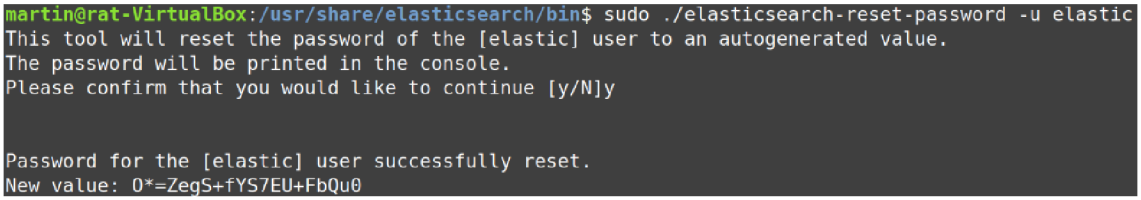
Dalším krokem je zajištění komunikace mezi Elasticsearch a Kibana. Nejprve musíme vytvořit nová hesla pro elastic a kibana_system uživatele. K tomuto účelu jsou v binárních souborech Elasticsearch umístěny skripty, které tento krok usnadňují. Celý proces získání hesel a nastavení služby vypadá takto:

1. Zkontrolujeme, že obě služby běží bez problémů pomocí systemctl status elasticsearch / kibana. Pokud ne, zapneme je pomocí systemctl start elasticsearch / kibana.



Obrázek 28 - Elasticsearch, Kibana status. Zdroj: vlastní

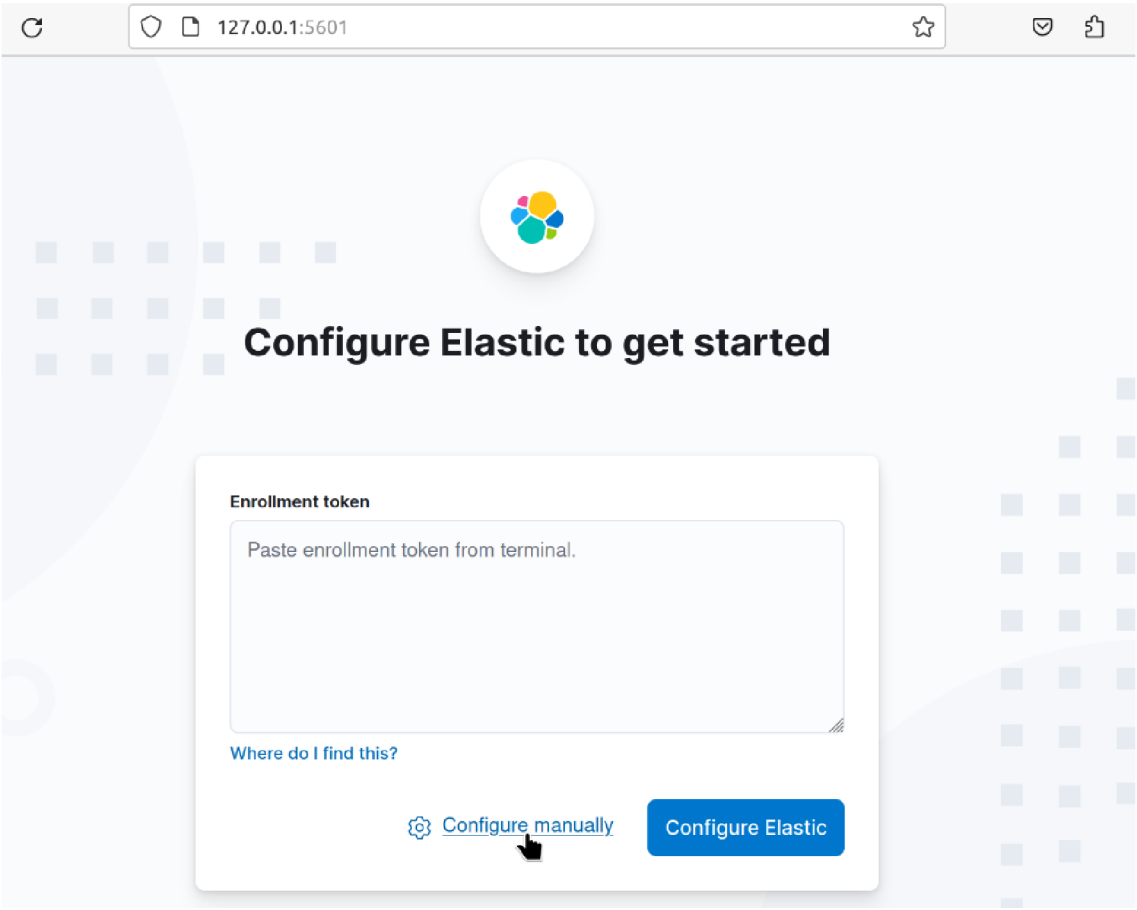
2. V adresáři /usr/share/elasticsearch/bin/ vyhledáme soubor elasticsearch-reset-password, který s parametrem –u spustíme pro uživatele kibana_system a elastic



Obrázek 29 - Reset hesla uživatele elastic. Zdroj: vlastní

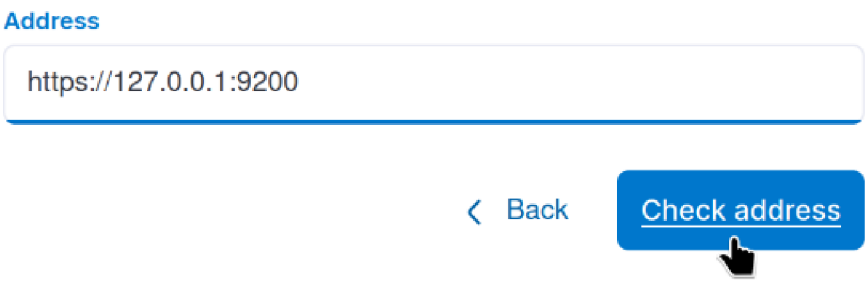
Po dokončení tohoto skriptu je heslo vypsáno do konzole. V tomto ukázkovém prostředí, viditelnost tohoto hesla nepředstavuje žádný bezpečnostní problém, je však dobré zaručit, že na skutečném prostředí nedojde ke kompromitaci tohoto hesla, neboť v základní konfiguraci je elastic uživatel schopen provádět jakékoliv úpravy v databázi a mohlo by tak dojít k bezpečnostnímu incidentu.

3. Nyní se přesuneme do grafického prostředí Kibana. Do prohlížeče zadáme URL adresu a port, na kterém Kibana běží. V této ukázce je to 127.0.0.1:5601. Na této stránce by na nás měl čekat konfigurační formulář, ve kterém zvolíme možnost Configure manuály.



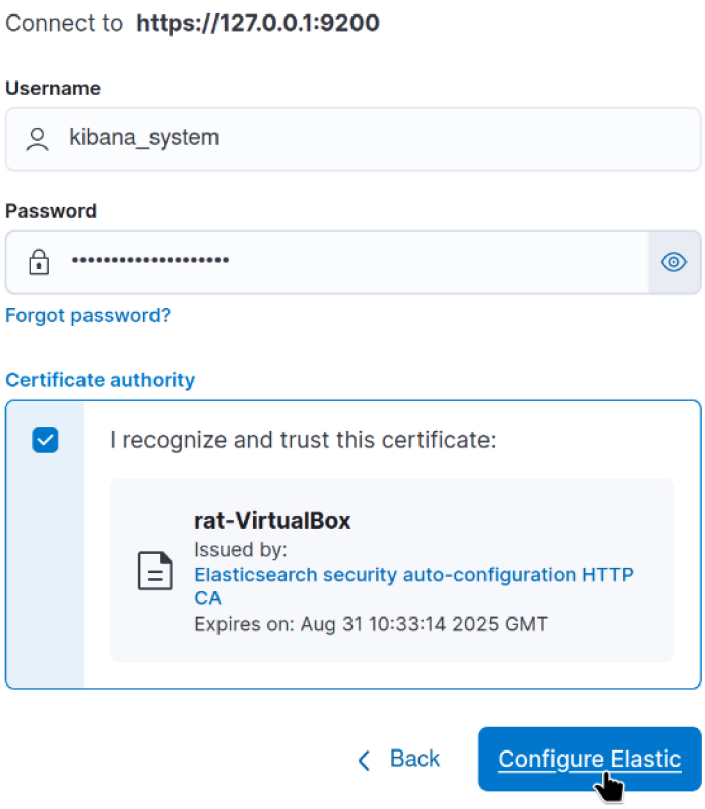
Obrázek 30 - Kibana konfigurační formulář. Zdroj: vlastní

4. Po změně formuláře bude vyžadováno zadání ip adresy, na které je Elasticsearch hostován, společně s portem. V tomto případě je to 127.0.0.1:9200.



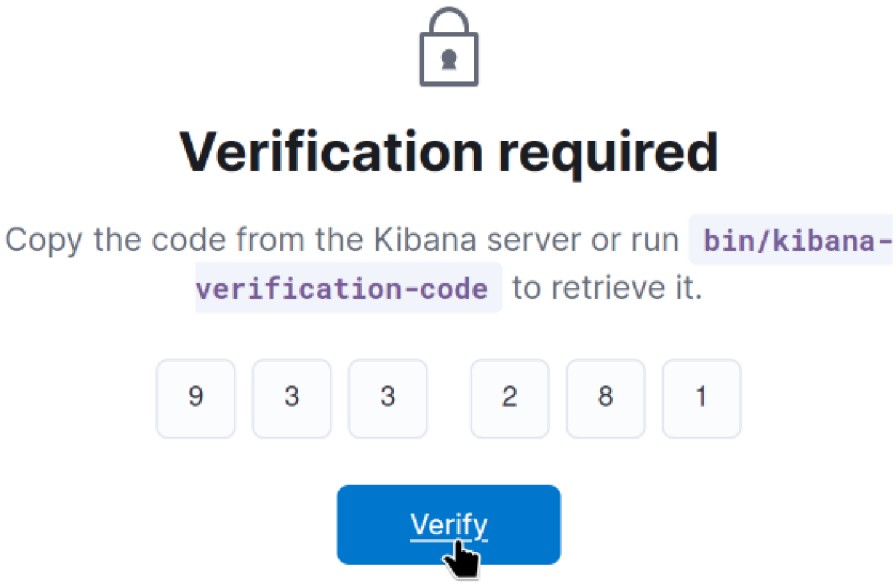
Obrázek 31 - Elasticsearch IP Adresa. Zdroj: vlastní

5. Dalším krokem je vyplnění hesla pro kibana_system uživatele, které bylo získáno v kroku 2. Zároveň zaškrtneme políčko, kterým nastavíme certifikát zařízení jako důvěryhodný.



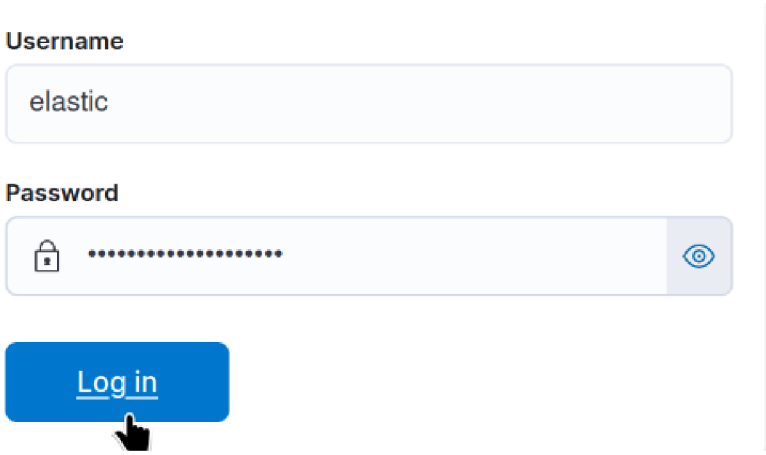
Obrázek 32 – Kibana, heslo a certifikát. Zdroj: vlastní

6. Po odeslání formuláře musíme zadat ověřovací kód z Kibana serveru. Tento kód lze získat spuštěním skriptu kibana-verification-code v adresáři /usr/share/kibana/bin.



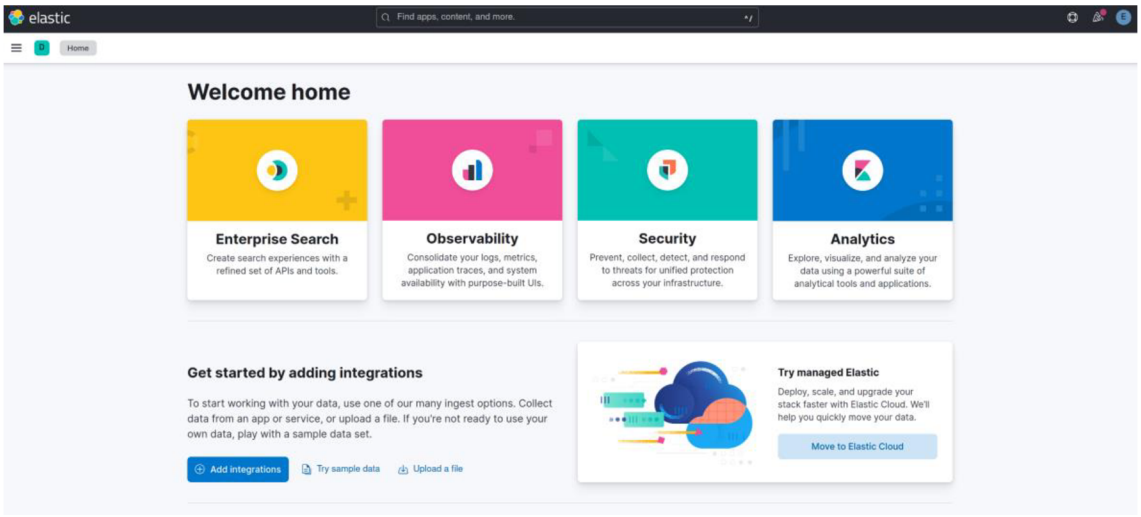
Obrázek 33 - Kibana, verifikace. Zdroj: vlastní

7. Po ověření bude spojení mezi Elasticsearch a Kibana automaticky dokončeno a po dokončení bychom měli vidět přihlašovací formulář.



Obrázek 34 - Kibana, přihlášení. Zdroj: vlastní

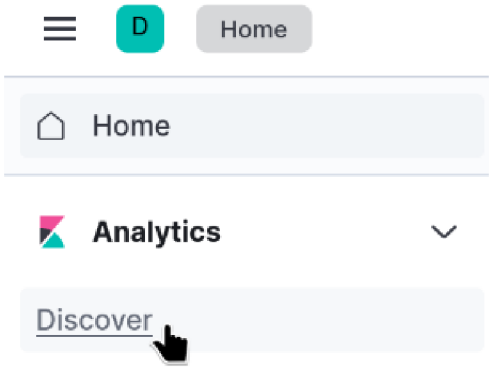
8. Po zadání uživatelského jména elastic a hesla které bylo získáno v kroku 2, bychom se měli dostat na úvodní stránku Kibana.



Obrázek 35 - Kibana, úvodní stránka. Zdroj: vlastní

11 Prohlížení dat z aplikace

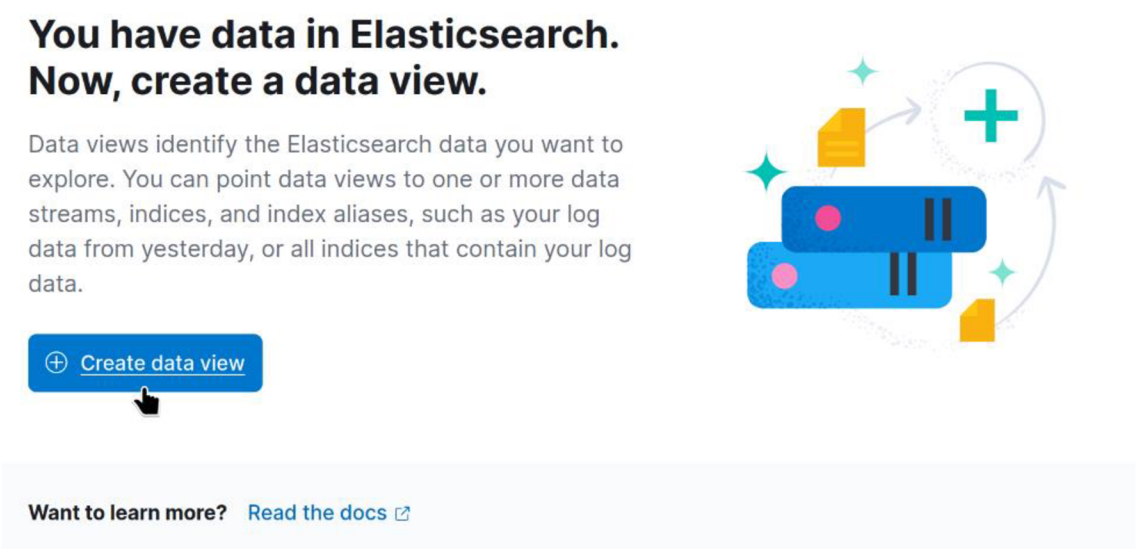
Pokud se nyní zaměříme na menu v levém horním rohu stránky, můžeme se prokliknout do záložky Discover.



Obrázek 36 - Záložka Discover. Zdroj: vlastní

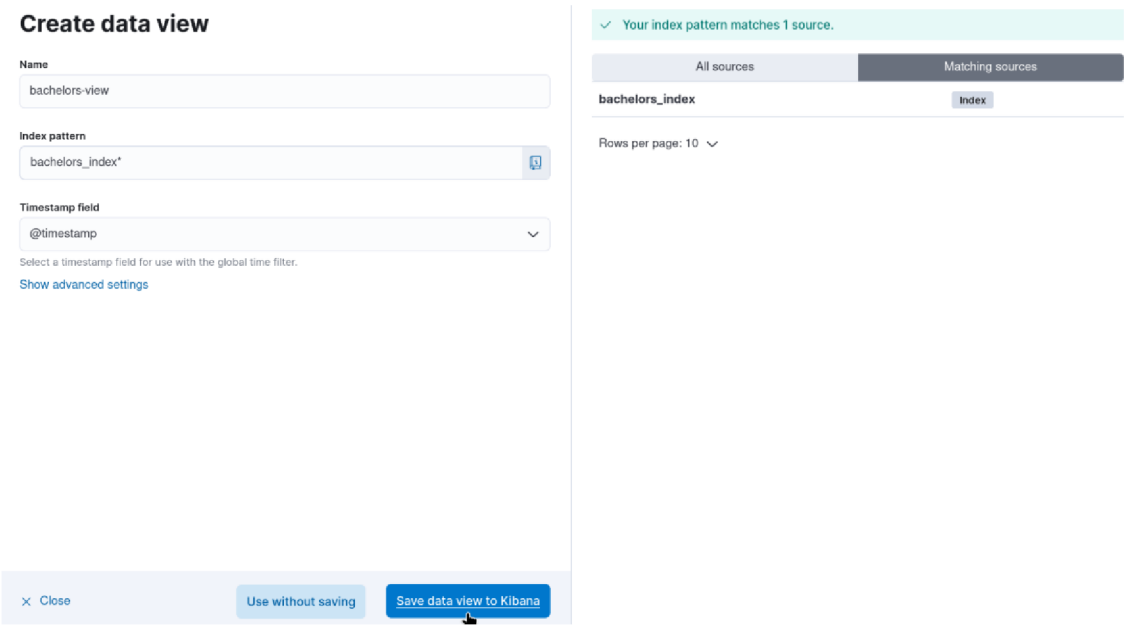
Zde můžeme vidět všechny indexy, které byly do Elasticsearch uloženy. V našem případě je ale stránka prázdná, neboť zobrazované indexy se nejprve musí v Kibana nastavit.

Klikneme tedy na odkaz Create data view.



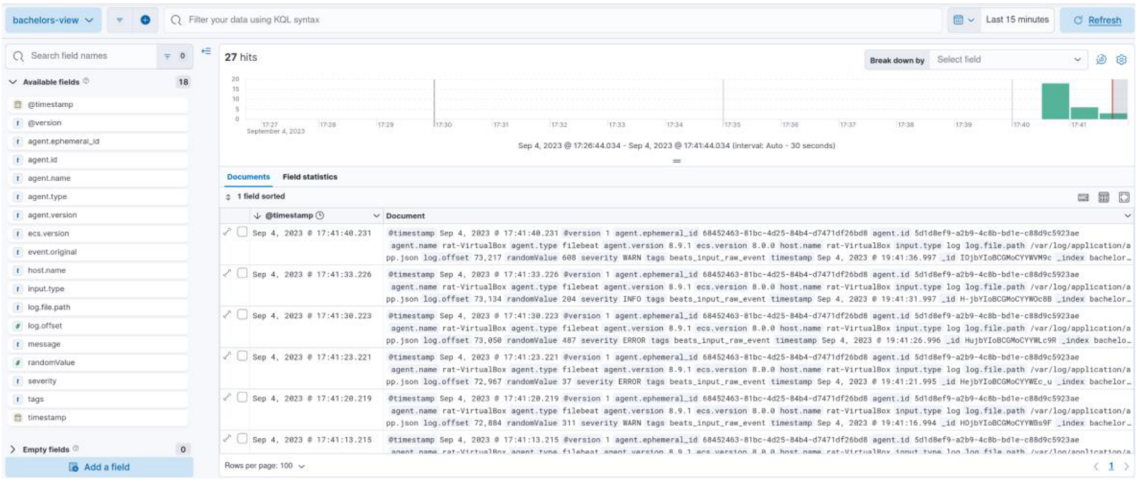
Obrázek 37 - Kibana, Create data view. Zdroj: vlastní

V dalším okně zvolíme název pohledu a Index pattern, který se v něm bude zobrazovat.



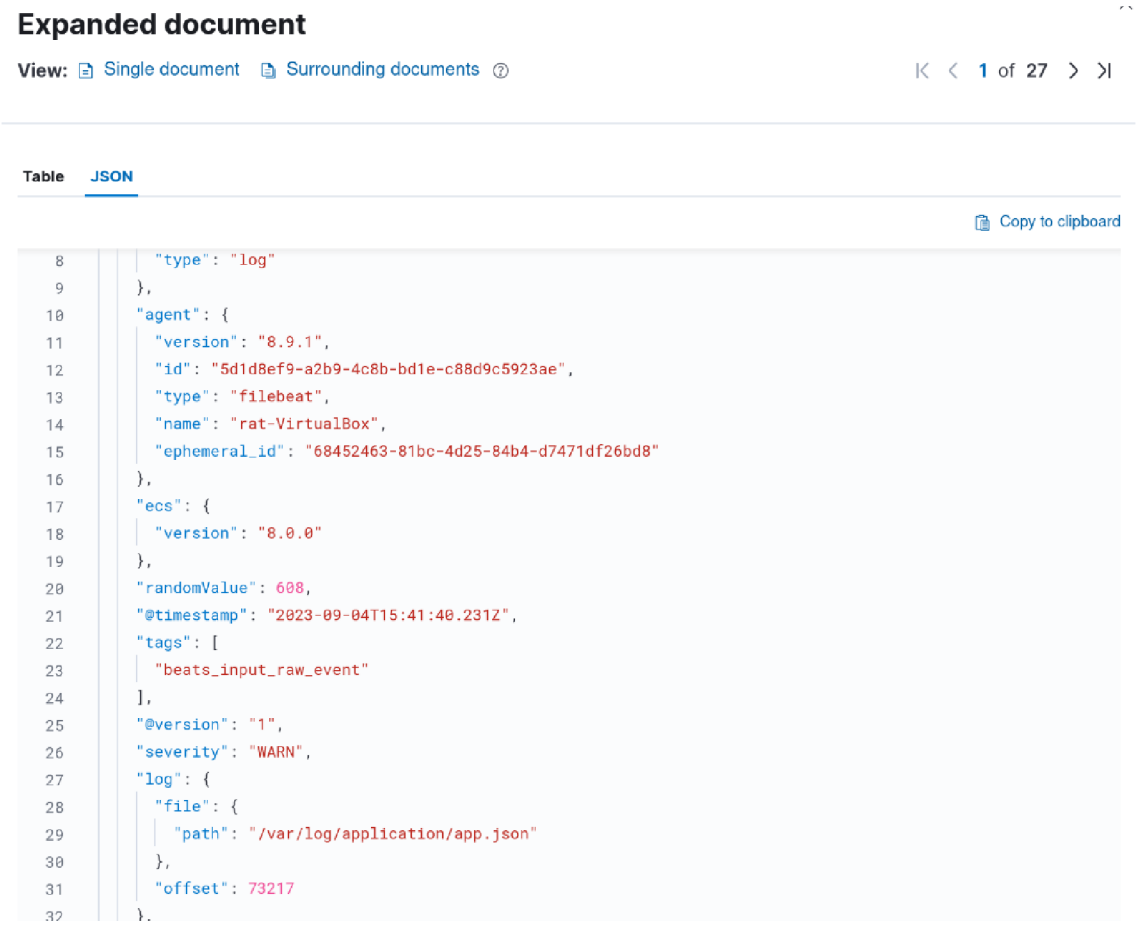
Obrázek 38 - Kibana, vytvoření pohledu. Zdroj: vlastní

Po dokončení pohledu v záložce Discover můžeme vidět všechna data. Na časové ose můžeme vidět, že při startu aplikace došlo k přijetí většího množství logů, neboť spuštění Filebeat bylo pozdrženo. To je důkazem toho, že je správně nastavené at-least-once delivery a žádné záznamy nebyly ztraceny.



Obrázek 39 - Kibana, data. Zdroj: vlastní

Po otevření jednoho ze záznamů, můžeme vidět všechny informace buď v tabulkovém nebo JSON formátu. Níže je uveden jeden z přijatých záznamů, u kterého můžeme vidět dříve definované informace – randomValue, timestamp a severity.



Obrázek 40 - Kibana, detail. Zdroj: vlastní

12 Závěr

Cílem této bakalářské práce bylo vytvořit logovací aplikaci v jazyce Java a demonstrovat, jak lze tato data zpracovávat a vizualizovat pomocí Filebeat a ELK stacku. Práce byla rozdělena do dvou hlavních částí

Nejprve bylo popsáno, k čemu aplikační logy slouží, co by měli obsahovat a jak by s nimi mělo být zacházeno. Poté byly vysvětleny jednotlivé aplikace ELK stacku, kde byla popsána rodina Beats, přesněji pak Filebeat, Logstash, Elasticsearch a Kibana. Byl představen programovací jazyk Java, balíčkovací nástroj Maven a knihovna Gson.

V druhé části byl představen návrh a implementace logovací aplikace, která tvoří ukázkové záznamy, které byly pomocí ELK stacku přeneseny ze souborového systému do databáze Elasticsearch. Bylo probráno, jak se jednotlivé služby instalují a konfigurují a v poslední části bylo ukázáno prostředí Kibana, jak se v něm orientovat a jak si nakonfigurovat pohled na data.

Tato bakalářská práce ukázala, že efektivní logování a vizualizace logovacích dat jsou klíčovými prvky pro monitorování a správu aplikací. Použití ELK stacku ve spojení s vlastní logovací aplikací umožňuje získat cenné informace o chování aplikace a rychle reagovat na potenciální problémy.

13 Seznam použité literatury

[1] What is an application log. Afran Shafir. *https://www.crowdstrike.com/[online].* Dostupné z: https://www.crowdstrike.com/cybersecurity-101/observability/application-log

[2] Understanding Logging Levels: What They Are & How To Use Them. KUĆ, RAFAL. *Https://sematext.com/* [online]. 2020. Dostupné z: https://sematext.com/blog/logging-levels/

[3] Meet the search platform that helps you search, solve, and succeed. *Https://www.elastic.co/elastic-stack/* [online]. Dostupné z: https://www.elastic.co/elastic-stack/

[4] Beats Lightweight data shippers. *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/beats

[5] Beats Lightweight data shippers. *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/guide/en/beats/libbeat/current/beats-reference.html

[6] Heartbeat overview. *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/guide/en/beats/heartbeat/current/heartbeat-overview.html

[7] Auditbeat overview. *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/guide/en/beats/auditbeat/current/auditbeat-overview.html

[8] Winlogbeat Overview. *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/guide/en/beats/winlogbeat/current/_winlogbeat_overview.html

[9] Packetbeat Overview. *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/guide/en/beats/packetbeat/current/packetbeat-overview.html

[10] Metricbeat overview. *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/guide/en/beats/metricbeat/current/metricbeat-overview.html

[11] Filebeat overview. *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-overview.html

[12] How Filebeat works. *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/guide/en/beats/filebeat/current/how-filebeat-works.html

[13] Logstash Introduction. *Https://www.elastic.co* [online]. Dostupné z: elastic.co/guide/en/logstash/current/introduction.html

[14] How Logstash Works. *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/guide/en/logstash/current/pipeline.html

[15] What is Elasticsearch? *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html

[16] Mapping. *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html

[17] Node. *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-node.html

[18] Index API. *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-index_.html

[19] Scalability and resilience: clusters, nodes, and shards. *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/scalability.html

[20] The search API. *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/search-your-data.html

[21] Ingest pipelines. *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/ingest.html

[22] Text analysis overview. *Https://www.elastic.co* [online]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-overview.html

[23] Kibana—your window into Elastic. *Https://www.elastic.co/* [online]. Dostupné z: https://www.elastic.co/guide/en/kibana/current/introduction.html

[24] Java in 100 Seconds. FIRESHIP. *Https://www.youtube.com/* [online]. 2021. Dostupné z: https://www.youtube.com/watch?v=I9AzO1FMgM8

[25] *Java (programovací jazyk)* [online]. Dostupné z: https://cs.wikipedia.org/wiki/Java_(programovac%C3%AD_jazyk)

[26] *Garbage Collection in Java* [online]. Dostupné z: https://www.geeksforgeeks.org/garbage-collection-java/

[27] *Heap Generations for Garbage Collection* [online]. Dostupné z: https://waytoeasylearn.com/learn/heap-generations/

[28] *Java main() Method – public static void main(String[] args)* [online]. Dostupné z: https://www.geeksforgeeks.org/java-main-method-public-static-void-main-string-args/

[29] *Introduction to Java Primitives* [online]. Dostupné z: https://www.baeldung.com/java-primitives

[30] *Non-primitive data types in Java* [online]. Dostupné z: https://www.javatpoint.com/non-primitive-data-types-in-java

[31] *Non-primitive Data Types in Java* [online]. Dostupné z: https://www.scaler.com/topics/non-primitive-data-types-in-java/

[32] *Why Java Strings are Immutable?* [online]. Dostupné z: https://www.geeksforgeeks.org/java-string-is-immutable-what-exactly-is-the-meaning/

[33] *Access Modifiers in Java* [online]. Dostupné z: https://www.geeksforgeeks.org/access-modifiers-java/

[34] *Apache Maven* [online]. Dostupné z: https://cs.wikipedia.org/wiki/Apache_Maven

[35] GOOGLE. *Gson* [online]. Dostupné z: https://github.com/google/gson

[36] *Repositories for APT and YUM* [online]. Dostupné z: https://www.elastic.co/guide/en/beats/filebeat/8.9/setup-repositories.html#_apt

[37] *Directory layout* [online]. Dostupné z: https://www.elastic.co/guide/en/beats/filebeat/8.9/directory-layout.html#directory-layout

[38] *Install Elasticsearch with Debian Package* [online]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/deb.html

[39] *Installing Logstash* [online]. Dostupné z: https://www.elastic.co/guide/en/logstash/current/installing-logstash.html

[40] *Logstash Directory Layout* [online]. Dostupné z: https://www.elastic.co/guide/en/logstash/current/dir-layout.html

14 Seznam použitých obrázků

[1] https://swapps.com/blog/a-quick-journey-along-the-elastic-stack/

[2] https://www.elastic.co/guide/en/beats/libbeat/current/beats-reference.html

[3] https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-overview.html

[4] https://www.bmc.com/blogs/logstash-using-data-pipeline/

[5] https://medium.com/geekculture/elasticsearch-architecture-1f40b93da719

[6] https://www.elastic.co/guide/en/elasticsearch/reference/current/ingest

[7] https://www.elastic.co/guide/en/kibana/current/dashboard.html

[8] *https://en.wikipedia.org/wiki/File:Duke_%28Java_mascot%29_waving.svg*

[9] https://www.statista.com/chart/16567/popular-programming-languages/

[10] https://www.digitalocean.com/community/tutorials/java-jvm-memory-model-memory-management-in-java

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2023/2024

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

Jméno a příjmení: **Martin Malíř**
Osobní číslo: **I2100241**
Adresa: Býšť 255, Býšť, 53322 Býšť, Česká republika

Téma práce: Návrh a implementace Elastic stack logování
Téma práce anglicky: Elastic stack logging design and implementation
Jazyk práce: Čeština

Vedoucí práce: Ing. Tomáš Svoboda, Ph.D.
Katedra informačních technologií

Zásady pro vypracování:

Cílem bakalářské práce je navrhnout a implementovat funkční řešení logování aplikací pomocí technologie ElasticStack. V teoretické části práce autor podrobně a systematicky představí technologii elasticstack a její možnosti při zajištění logování aplikací. V praktické části práce autor podrobně představí možnosti instalace a konfigurace elastickstacku pro zajištění logování aplikací v rámci definovaných use-case ve formě step-by-step postupů.

Seznam doporučené literatury:

SHARMA, Vishal. *Beginning Elastic Stack*. APress, 2016. ISBN 1484216938.

Podpis studenta: Datum:

Podpis vedoucího práce: Datum: