FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Případová studie – systém pro zpracování aplikačních a infrastrukturních logů
Student:	Vojtěch Dušátko
Vedoucí:	Ing. Pavel Šedek
Studijní program:	Informatika
Obor / specializace:	Bezpečnost a informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je případová studie zavedení systému pro sběr aplikačních a infrastrukturních logů ve firmě.

1. Analyzujte potřeby firmy a požadavky na systém.
2. Proveďte průzkum existujících řešení a na základě požadavků navrhněte možná řešení.
3. Proveďte srovnání a případnou základní implementaci pro další průzkum a testování.
4. Navrhněte na základě průzkumu a případných testů vhodné kandidáty na implementaci.
5. Proveďte odhad finanční a časové náročnosti implementace vybraných řešení.

Elektronicky schválil/a prof. Ing. Pavel Tvrdík, CSc. dne 8. prosince 2021 v Praze.

Bakalářská práce

PŘÍPADOVÁ STUDIE – SYSTÉM PRO ZPRACOVÁNÍ APLIKAČNÍCH A INFRASTRUKTURNÍCH LOGŮ

Vojtěch Dušátko

Fakulta informačních technologií
Katedra počítačových systémů
Vedoucí: Ing. Pavel Šedek
20. června 2022

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Rád bych zde poděkoval Ing. Pavlu Šedkovi za vedení mé bakalářské práce, za cenné rady a připomínky, které mi poskytl.
Rád bych zde poděkoval také oddělění SVTI FEL ČVUT za možnost testovat tyto systémy na reálném prostředí a konkrétně Ivo Hulínskému za praktické připomínky k nasazení systémů, popisovaných v této práci.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 20. června 2022 .....

Abstrakt

Cílem této bakalářské práce je zpracování případové studie systému pro zpracování aplikačních a infrastrukturních logů ve firmě. Součástí studie je analýza obecných funkčních a nefunkčních požadavků firmy, průzkum trhu, testovací nasazení a nakonec návrh řešení. Studie je koncipována pro obecné požadavky na základní řešení pro téměř jakoukoli firmu, začínající se sběrem aplikačních a infrastrukturních logů. V této studii bylo využito vícekolového výběru pro výběr optimálního řešení. Dále práce popisuje omezené, testovací nasazení vybraných řešení v reálném prostředí. Na to navazuje návrh plné implementace, splňující všechny funkční a nefunkční požadavky s odhadem finanční a časové náročnosti plného nasazení a zmínění možností dalšího rozvoje.

Klíčová slova Případová studie, Logovací server, Syslog, ELK Stack, Elastic Stack, EFK Stack, Graylog, PLG Stack

Abstract

The aim of this bachelor thesis is to create a case study to a system for processing application and infrastructure logs in a company. The study includes an analysis of the generic functional and non-functional requirements of the company, research of possible solutions, testing deployment and finally the design of a solution. The study is designed to satisfy generic requirements for basic solution of almost every company, starting with the collection of application and infrastructure logs. In this study was used multi-round selection to find the optimal solution. The thesis also describes the test deployment of the selected solution in a real environment. This is followed by a proposal for full implementation, satisfying all functional and non-functional requirements with an estimate of the financial and time requirements of full deployment and there are mentioned possible future improvements.

Keywords Case study, Logging server, Syslog, ELK Stack, Elastic Stack, EFK Stack, Graylog, PLG Stack

Seznam zkratek

API	Application Programming Interface
CNCF	Cloud Native Computing Foundation
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
ELK	Elasticsearch + LogStash + Kibana
HA	High Availablitiy
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaC	Infrastructure As Code
IETF	Internet Engineering Task Force
IoT	Internet of Things
JSON	JavaScript Object Notation
MD	Man-Day
NoSQL	non-SQL
RBAC	Role Based Access Control
REST	Representational State Transfer
RFC	Request for Comments
SIEM	Security Information and Event Management
SQL	Structured Query Language
SSD	Solid State Drive
UI	User Interface
VLAN	Virtual Local Area Network
VPN	Vitrual Private Network
YAML	YAML Ain't Markup Language

Úvod

Logování je již mnoho let nedílnou součástí všech možných zařízení a aplikací. Poskytují nám celkový pohled na funkčnost aplikace nebo zařízení, případně interakce v rámci infrastruktury.

Logy obsahují mnoho důležitých informací, bez kterých by mnoho věcí bylo téměř nemožných - ať už hledání chyb, řešení incidentů, případně bezpečnostní audit. V určitý moment se ale každá firma dostane do situace, kdy už si nevystačí s pouhým čtením logů v souborech, rozházených po infrastrukturních zařízeních, či na různých instancích serverů. V tento moment je zapotřebí se zamyslet nad centralizací logů z všech těchto míst do jednotného prostředí, umožňujícím daleko více než jen čtení řádově milionů, miliard nebo ještě vyšších počtů řádků logů.

Tuto práci jsem si vybral na základě vcelku vágně specifikované potřeby nasazení systému pro zpracování logů na Středisku výpočetní techniky a informatiky FEL ČVUT. Obdobné požadavky vznikly v průběhu času i na dalších projektech. Rozhodl jsem se tedy napsat tuto práci maximálně obecně a přenositelně téměř pro každé nasazení. Primárním požadavkem je on-premise provoz, který narozdíl od možnosti hostovat nějaký software v cloudu, je dnes lehce na ústupu, takže některé softwary tuto možnost vůbec nemusí nabízet.

V první části práce bude čtenář seznámen s důvody, proč logujeme, jakými způsoby lze logy přenášet a jak je lze centralizovat. Poté budou představena řešení, nalezená při průzkumu trhu. Dále budou tato řešení otestována v nějakém základním režimu a nakonec bude navrženo nejoptimálnějšího řešení na základě průzkumu a testování, společně s finanční a časovou náročností.

Kapitola 1
Logy a logování

1.1 Co jsou to logy a proč je potřebujeme

Logy, neboli žurnál je soubor záznamů o událostech, které se staly v průběhu životního cyklu aplikace nebo zařízení ve světě výpočetní techniky. Lze je považovat za důležitou funkční část, kterou nelze opomíjet již od prvotního návrhu architektury. Tyto záznamy jsou často využívány k hledání chyb, analyzování správnosti funkce systému, analyzováni incidentů, trasování uživatele a mnoho dalším aktivitám vývojářů nebo správců systémů. U logů je žádané, aby měly co nejvyšší informační hodnotu v poměru na jejich objem. Toto je bohužel v praxi často opomíjeno a je nutné tyto záznamy dále složitě filtrovat.[1][2][3]

Vhledem k tomu, že logy generuje stroj podle nějakého předem určeného předpisu, mají většinou nějaký stabilní formát. Díky stabilnímu formátu lze logy jednoduše zpracovávat a provádět nad nimi další analýzu. Bohužel formáty se liší a prakticky každá aplikace nebo zařízení mají vlastní styl a různé možnosti nastavení. Některé aplikace „prostě logují“, jiné umožňují nastavovat minimální úroveň a některé dokonce mají možnost nastavit vlastní formát záznamu o události. Větší aplikace, které sestávají z více modulů mají často možnost nastavit úrovně a případně i formáty na úrovni každého modulu zvlášť *(např.: Apache 2 webserver[4]).*

Pokud si vyvíjíme vlastní aplikaci, je časté, že všem jejím modulům / komponentám nastavujeme jednotné parametry logování, jako je formát, nejnižší logovaná závažnost, aj. V tento moment přichází na scénu tzv. Full-Stack Observability[5]. Tento pojem se těžko překládá, ale jako vysvětlení jistě skvěle poslouží toto:

> *„Schopnost systematicky získávat relevantní informace o stavu aplikací a všech hardwarových a softwarových komponentách, na kterých závisí, za účelem vyhodnocení dopadu na uživatele, obchodní výsledky, kontinuální optimalizace a rychlého řešení případných problémů.“ [6]*

Tento pojem se sice jeví, jako „svatý grál dohledu“, nicméně v případě dohledu nad heterogenní infrastrukturou, která se vyskytuje v mnoha firmách je jeho dosažení příliš drahé, a to ať už v hodinách práce nebo investicích. Sběr aplikačních a infrastrukturních logů je tedy společně se sběrem metrik často využíván jako základní stavební kámen dohledu nad infrastrukturou, umožňující řešit incidenty, ladit funkčnost a další.

Logy, které vygeneruje nějaké zařízení, či aplikace je třeba v běžném produkčním prostředí ukládat. Logování je ale prováděno mnoha způsoby. Asi nejznámější způsoby jsou:

- Ukládání do souborů
- Výpis na standardní výstup / standardní chybový výstup
- Odesílání logů po síti

Každý z těchto způsobů má své výhody a nevýhody. Například soubory je dobré rotovat, což buď musí zajišťovat sama aplikace nebo další externí nástroj, například `logrotate`[7]. Proto lze tyto způsoby kombinovat, například posíláním logů na standardní (chybový) výstup, sběrem tohoto výstupu za pomoci `systemd-journal`[8] pro další zpracování nebo třeba ukládáním logů aplikace do souboru, který čte nástroj typu `rsyslog`[9] a posílá je někam po sítí.

Dokonce ve světě kontejnerů platí odstavec výše ještě více. Problémem v tomto světě je dynamičnost a nestálost, kdy je naprosto běžné, že kontejner při chybě skončí s nějakým chybovým kódem a orchestrátor, jako například Kubernetes na něj více nesahá, ale namísto toho spustí úplně nový kontejner, který ale obsahuje stejnou konfiguraci, persistentní úložiště, aj. Aplikace, které v kontejnerech běží by měly být nastavené *(a většinou i jsou)* tak, aby logovaly na standardní (chybový) výstup. Tyto výstupy kontejnerizační engine *(například Docker[10])* čte a ukládá si je do uživatelem zvoleného úložiště. Tímto je například u zmíněného Dockeru ve výchozím stavu JSON[11] soubor[12], který je možné číst nějakým nástrojem, jako ELK Beats[13], `rsyslog`[14] a další. Poté dochází nejčastěji k odesílání logů do nějakého centrálního úloiště.

1.2 Syslog

Syslog je standard pro logování záznamů. Často je nazýván, jako „protokol“, nicméně nejde o protokol ve smyslu síťového protokolu, ale o řešení logování, které původně navrhl Eric Allman[15] v osmdesátých letech minulého století. Dlouho po prvním návrhu a mnoha implementacích byl protokol prvně vágně definován v RFC 3164[16], jako *„The BSD Syslog Protocol“*. Toto RFC má ale mnoho problémů. Jedním z nich je ten, že není v mnoha oblastech vůbec konkrétní, dalším je jeho nezávislost na přenosovém protokolu a v neposlední řadě to, že položky pouze doporučuje *(neboli jsou pouze RECOMMENDED, podle RFC 2119[17])* a také je celé kategorizováno pouze jako „Informational“.

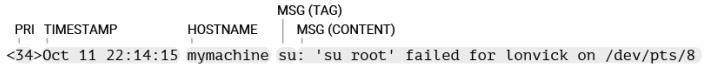
Nástupcem RFC 3164 je specifikace RFC 5424[18], aneb *„The Syslog Protocol“*, která je již narozdíl od RFC 3164 daleko konkrétnější a to jak v oblastech formátu, tak v oblasti doporučených přenosových protokolů. Zároveň toto RFC vymezuje, kterými oblastmi se zabývá a v oblastech, kterými se nezabývá odkazuje na další RFC, které by problematiku mohly řešit. V neposlední řadě je narozdíl od RFC 3164 zařazeno do kategorie *„Standard track“*, nikoli *„Informational“*.

I přesto, že RFC 5424 označuje RFC 3164 jako zastaralé, tak v realitě dochází ke koexistenci dvou „standardů“ vedle sebe. Toto přináší mnoho problémů, které níže popíši.

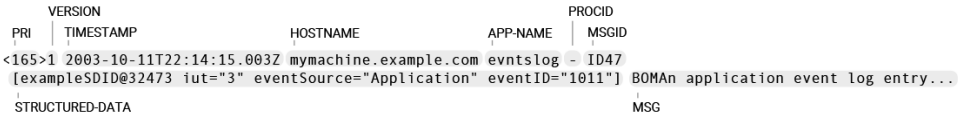
Primárním problémem je nekompatibilita formátů zasílaných dat. RFC 3164 definuje 3 části zprávy: `PRI`, `HEADER` a `MSG`, nicméně zároveň říká, že je doporučené, nikoli povinné, aby zpráva obsahovala všechny 3 části. RFC 5424 definuje daleko více položek, ale jejich část je také pouze doporučená. Toto přináší jak problémy s parsery, kde je nutné obsáhnout nepovinnost položek, tak problémy s nemožností přijmout aplikací, kompatibilní s pouze jedním RFC zprávu z druhého a naopak.

Dalším problémem, který může nastat je konverze formátu zprávy při předávání zpráv na přeposílacích uzlech cesty. Například v programu `rsyslog` je možné přijímat zprávy v obou formátech, ale při přeposílání je nutno definovat šablonu. Pokud jako šablonu zvolíme například `RSYSLOG_SyslogProtocol23Format`[20], tak bude docházet k případným konverzím zpráv z RFC 3164 formátu na formát podle RFC 5424.

Obrázek 1.1 Ukázka zprávy podle RFC 3164[19]



Obrázek 1.2 Ukázka zprávy podle RFC 5424[19]



I přes problémy s ním spojené jde o asi nejrozšířenější způsob centralizace logování. Díky jeho jednoduchosti, flexibilitě implementace a nezávislosti na přenosovém protokolu je možno přijmout a zpracovat jednoduše obrovské množství záznamů z různorodých aplikací a zařízení.

1.3 Centralizované logování

Centralizované logování je přístup, kdy vedle běžné, provozní infrastruktury máme specializovanou infrastrukturu, která je určena primárně na sběr a případnou analýzu záznamů, přijatých z provozní infrastruktury. Tento přístup má jako hlavní výhodu již to, co napovídá název - veškeré záznamy o aktivitách jsou na jednom místě, a to i v potenciálně obrovských - složitých infrastrukturách.

Jednou z jeho obrovských výhod je to, že logy typicky nezůstávají na zařízení, kde k daným událostem došlo, ale jsou odesílány do centrálního úložiště. Toto umožňuje v oddělené infrastruktuře provádět analýzu událostí, které se staly a to i v případě ztráty zařízení (např. při poruše hardware nebo pádu aplikačního kontejneru).

Jako další výhodou centralizovaného logovacího řešení můžeme považovat ztížení smazáni stop potenciálnímu útočníkovi, který útočí na naše servery. Pokud nám útočník napadne provozní infrastrukturu, tak než stihne smazat záznamy o své aktivitě, dojde k jejich odeslání na další servery, které mají často daleko striktnější přístupové politiky, protože nejsou určeny běžným uživatelům nebo zákazníkům.

Nadstavbou nad „hloupým“ centralizovaným logováním je například tzv. SIEM, neboli „Security Information and Event Management“, což je technologie, která nejen centrálně sbírá a agreguje data, ale zároveň provádí vyhodnocování různých korelací mezi událostmi v infrastruktuře a zajišťuje případné varování, zobrazuje informační panely, a jiné. Jde o mocný nástroj, který výrazně zvyšuje bezpečnost organizace a zároveň silně zakládá na sběru logů, který zde popisuji.[21][22][23] Pro SIEM se v dnešní době používá integrace s mnoha dalšími systémy, strojové učení nebo dokonce umělá inteligence a mnoho dalších, díky kterým lze v téměř reálném čase rozpoznat, že je něco divně a informovat o tom pověřené osoby.

1.4 Víceprotokolové logování

Jak jsem již zmínil výše, tak každé řešení má své výhody a nevýhody. Často se tedy vyplácí kombinovat větší množství řešení k dosažení nejlepšího výsledku. Jinak tomu není ani u logování, proto není ojedinělé, že v infrastrukturách spolu koexistuje více způsobů, jak odesílat logy do centrálních úložišť.

Syslog, jak bylo popsáno výše, je například protokolově nezávislý, ale zároveň popisuje v sekci `5.1`, že každá implementace RFC 5424 musí podporovat přenos pomocí TLS protokolu, který popisuje RFC 5425[24] a zároveň by každá implementace měla *(pouze SHOULD, podle RFC 2119[17], nikoli MUST)* podporovat přenos pomocí UDP protokolu, který popisuje RFC 5426[25]. Z toho vyplývá ještě třetí možnost, kterou dlouho nikdo nezmínil a tou je přenos pomocí TCP protokolu - neboli „RFC 5425 bez šifrování“. To bylo popsáno až o cca 3 roky později v rámci RFC 6587[26], které již v úvodu popisuje, že jde primárně o historický záznam od IETF, protože vzniklo mnoho implementací této „střední cesty“ bez možnosti opřít se o jakýkoli standard. Jediný předchozí popis, syslogu přes TCP vznikl, jakožto RFC 3195[27], aneb *„Reliable Delivery for syslog“*, které se ale pojilo k původní definici syslogu podle RFC 3164 - tedy bylo nadále nepoužitelné.

1.4.1 Syslog přes TLS - RFC 5425

Syslog přenášený skrze TLS, které využívá, jako transportní vrstvu snad vždy TCP je zabezpečenou variantou přenosu, která zároveň zajišťuje informovanost o doručení logované zprávy. Jde o nejpokročilejší z možností, jak přenášet syslogové události, ale zároveň o nejnáročnější na režii daných zařízení. Při použití TCP je potřeba udržovat spojení, která mohou mít u některých zařízení nezanedbatelnou režii, stejně tak TLS může vyžadovat nemalý výkon, hlavně při větším množství událostí.

Toto řešení přináší ještě jeden problém, kterým je infrastruktura veřejného klíče, aneb PKI[28] a s tím spojená nutnost v každém zařízení udržovat aktuální veřejnou část kořenového certifikátu, který je použit na serveru.

1.4.2 Syslog přes UDP - RFC 5426

I přesto, že vůči syslogu přes TLS je syslog přes UDP jednak nezabezpečený a dokonce ještě přes nespolehlivou transportní vrstvu, má řadu výhod a tudíž i své využití. Díky použití UDP je tento způsob bez zátěže na TCP spojení, bez potenciálních problémů s distribucí veřejných klíčů a dokonce bez problémů, které může přinášet případný výpadek logovacího serveru. Narozdíl od předchozího řešení totiž nedochází k udržování logů v paměti a případným retransmisím při neúspěšném doručení na server.

Hodí se tedy primárně na použití v rámci nějaké důvěryhodné a spolehlivé sítě. Touto sítí je myšleno například lokální síť v budově, kde je pro zajištění bezpečnosti přenosu vyhrazena speciální VLAN[29], na které komunikují klienti se syslog servery. Na této VLAN bývají typicky nastavená striktní omezení, že se do ní nemůže dostat ledajaké zařízení na síti. Typickým příkladem je centralizované logování ze síťových prvků samospráv, jakými jsou například fakulty ČVUT.

1.4.3 Syslog přes TCP - RFC 6587

Tento přístup kombinuje dva výše zmíněné a to konkrétně mezistavem, kdy je sice zajišťováno spolehlivé doručení za pomoci TCP protokolu, ale není zde řešeno šifrování při přenosu. Jde o jakousi „zlatou střední cestu“, která ale byla specifikována *„for the historical record“*, několik let

po RFC 5424 - 5426[18][24][25]. Zjevně ale začaly vznikat implementace, protože pro původní *BSD Syslog* existovalo RFC 3195[27], které přenos po TCP popisuje.

1.4.4 HTTP(S)

Přenos událostí pomocí HTTP(S) využívá nepřeberného množství výhod protokolu nejvyšší vrstvy ISO/OSI. HTTP(S) je implementováno ve verzích 1.0 - 2.0 pouze skrze TCP, až verze 3.0 využívá i protokol UDP, kde je ale spolehlivost přenosu zajištěna jinými mechanismy. Pro zabezpečení přenosu může být použito HTTPS, které je implentováno nad TLS vrstvou, dále HTTP(S) obsahuje možnost autentizace, což není běné u nižších protokolů. Nicméně využívání HTTP(S) protokolu může být zbytečně náročné na zařízení, které nejsou určena na odbavování webového provozu *(např.: IoT zařízení nebo síťové prvky)*.

1.4.5 MQTT, AMQP, Kafka, ...

Narozdíl od HTTP(s) jde o výrazně odlehčenější protokoly, které zajišťují asynchronní přenosy mezi aplikacemi a takzvanými „message brokery“, což jsou aplikace zajišťující zpracování zprávy, její směrování do front na základě mnoha různých pravidel *(routing keys / topics / aj.)*. Tento způsob přenosu zajišťuje velice nízké nároky na aplikaci, která zprávy odesílá, protože nemusí čekat, až si na ni server vyhradí čas a zpracuje její zprávu, ale dojde pouze k odeslání zprávy message brokerovi, který zajistí zbytek procesu až po předání zprávy do fronty, kde si ji vyzvedne koncová aplikace, provádějící zpracování.[30][31]

Tento přístup není ochuzen ani o bezpečnost. Specifikace těchto protokolů počítají jak s možností TLS, tak autentízace pomocí běžných mechanismů, jako je uživatelské jméno a heslo.

Jde tedy o velmi přínosný způsob odesílání zpráv z moderních zařízení typu IoT nebo dalších. Jediné, co je třeba je formátování zpráv, aby došlo ke správnému routováni, aj.

1.4.6 Redis

Redis je aplikace, zajišťující strukturovanou „in-memory“ databázi. Tato databáze se často využívá jako cache bez jakékoli persistence. Persistenci dat samozřejmě také umí, ale je tu zmíněn primárně jako řešení, schopné zajistit možnost logování i při přetížení infrastruktury, zpracovávající logy, případně při její údržbě. Zabezpečení je zde možné na transportní úrovni, tedy TLS, ale donedávna nebylo možné pro autentizaci využít nic lepšího, než sdílené heslo. Je tedy vždy na uvážení, zda je pro konkrétní nasazení vhodný.

Redis je někdy také využíván jako fronta zpráv, ale není k tomu vhodný, narozdíl od Kafky nebo RabbitMQ, které mají daleko rozsáhlejší implementaci v této oblasti.[32]

1.4.7 Další

Výše zmíněné protokoly samozřejmě nejsou jediné, které lze využít pro přenos událostí po síti. Existuje mnoho dalších možností, kde některé jsou proprietární a jiné jsou otevřeným standardem. Jako ukázku zde uvedu `Lumberjack protokol`, který byl implementován v rámci Elastic Stacku a slouží pro komunikaci mezi jednotlivými součástmi všude, kde není využito HTTP(S). Bohužel dokumentace k tomuto protokolu není na moc dobré úrovni, zato využití tohoto přenosu je velice široké.

> *However, this document (the protocol documentation) has fallen out of date with respect to the actual implementation of Elastic Beats project and the Logstash Beats*

input. It may be inaccurate in places, and we have not yet documented the changes between v1 and v2 protocols. This document is therefore deprecated and should not be used as reference. [33]

Kapitola 2

Požadavky na software

Požadavky vymezují, co software má nebo nemá dělat. Je důležité, aby požadavky byly co nejpřesnější, aby došlo k co nejmenší možnosti nepochopení se mezi zákazníkem a dodavatelem. Zároveň je nutné tyto požadavky důmyslně prověřit s ohledem na budoucí rozvoj.

Požadavky na software se běžně rozdělují na **funkční** a **nefunkční**. Tyto dvě skupiny se sice vzájemně vylučují, ale mají mezi sebou poměrně pevný vztah.

Obrázek 2.1 Funkční vs. nefunkční požadavky[34]

Parameters	Functional Requirement	Non-Functional Requirements
Requirement	It is mandatory	It is non-mandatory
Capturing type	It is captured in use case	It is captured as a quality attribute
End-result	Product feature	Product properties
Capturing	Easy to capture	Hard to capture
Objective	Helps you verify the functionality of the software	Helps you to verify the performance of the software
Area of focus	Focuses on user requirement	Concentrates on the user`s expectation and experience
Documentation	Describe what the product does	Describes how the product works
Product Info	Product features	Product properties

Funkční požadavky vymezují, co bude nebo nebude umět z pohledu funkcionalit. Běžné jsou požadavky na byznysové funkcionality, způsoby autentizace a zabezpečení, ale také podporované operační systémy nebo databázové backendy. Jako příklad zde lze uvést „podpora jednotného přihlášení, pomocí protokolu SAML 2 nebo OIDC“.

Nefunkční požadavky (neboli doplňkové) vymezují, jak se software bude chovat z abstraktnějších pohledů, jako například jeho limitů. Běžné jsou požadavky na výkon, škálovatelnost, do-

stupnost, ale také udržitelnost a mnoho dalšího. Pro příklad lze zde uvést „možnost horizontální škálovatelnosti se zajištěním vysoké dostupnosti“.

Požadavky jsem analyzoval primárně na Středisku výpočetní techniky a informatiky Fakulty elektrotechnické ČVUT a dále na několika málo menších projektech, nicméně většina požadavků je pro všechny analyzované prostředí velice podobná. Z tohoto důvodu budu požadavky i případný výběr psát pro „obecnou, abstraktní organizaci“. Jediné, co se povětšinou liší jsou nefunkční požadavky, pojící se k výkonu *(z důvodu množství událostí za nějaké časové okno)*, které se ale často vážou k hardware, na kterém je aplikace provozována. Popisuji zde samozřejmě organizaci, která má zájem o nějaký počáteční sběr logů a případnou indexaci, alerting, atp., nikoli společnosti například úrovně banky, kde se běžně využívá daleko větší dohled, včetně tzv. SIEMu, jenž jsem popsal v kapitole 1.3.[35][36]

2.1 Funkční požadavky

2.1.1 Podpora víceprotokolového logování

Vybrané řešení pro sběr logů musí být schopno přijímat logy více protokoly, nejen základním syslogem. Syslog má mnoho svých omezení a nevýhod, proto je důležité podporovat i jiné přenosové protokoly, hodící se na různé typy nasazení.

2.1.2 Přístupnost přes webové rozhraní

Vybrané řešení musí být přistupné skrze webové rozhraní, bez potřeby jiného klienta než webového prohlížeče.

2.1.3 Přístupnost přes API

Vybrané řešení musí podporovat přístup skrze API *(ideálně REST API)*m umožňující primárně administrativní úlohy.

2.1.4 Podpora zálohování, případně verzování konfigurace

Vybrané řešení musí podporovat zálohování, případně dokonce verzování konfigurace, umožňující replikaci prostředí, případně disaster recovery (obnově po havárii / katastrofě) nebo dokonce porovnávání změn, provedených v dancých časových intervalech.

2.1.5 Podpora uživatelských účtů, rolí a práv

Vybrané řešení musí podporovat uživatelské účty z důvodu zabezpečení a možnosti nastavení práv daným účtům. Musí zároveň podporovat možnost vytváření rolí a přidělování oprávnění těmto rolím. Minimálně podpora těchto rolí:

- Administrátor
- Editor dané subsekce
- Uživatel dané subsekce *(může číst, ne upravovat)*

2.1.6 Podpora zobrazování přijatých logů

Vybrané řešení musí podporovat zobrazování obsahu přijatých logů uživateli.

2.1.7 Podpora filtrace přijatých logů

Vybrané řešení musí podporovat filtrování na základě různých informací. Filtrování musí být možné minimálně podle:

- Časové značky událostí
- Zdrojového zařízení
- Hladiny závažnosti
- Pole, které se podařilo extrahovat z původní zprávy

2.1.8 Podpora vizualizace nasbíraných dat

Vybrané řešení musí podporovat zobrazování logů v grafech a tabulkách webového rozhraní na základě různých filtrací, popsaných v bodě 2.1.7.

2.1.9 Podpora nějaké formy retence dat

S ohledem na neustálý příjem dat musí vybrané řešení podporovat nějakou formu retence dat a to ať už kontinuální odmazávání nebo rotaci nějakých dokumentů v databázi nebo životních fází *(viz například ElasticSearch ILM[37]).*

2.2 Nefunkční požadavky

2.2.1 Podpora on-premise provozu

Vybrané řešení musí být možné nasadit na infrastruktuře, která je lokální. Podpora jakékoli formy as-a-service provozu je samozřejmě vítána, ale není primární[38].

2.2.2 Podpora horizontální i vertikální škálovatelnosti

Vybrané řešení musí být možné škálovat jak přidáním strojů do nějaké formy clusteru, tak přidáním výkonu danému stroji *(ať už virtuálnímu nebo fyzickému).*

2.2.3 Podpora vysoké dostupnosti řešení

Vybrané řešení musí podporovat jakoukoli formu zvýšené dostupnosti a to ať už v jakémkoli master-slave režimu nebo master-master.[39][40][41]

2.2.4 Podpora práce s velkým množstvím záznamů

Vybrané řešení musí být schopno v uchovávaných datech, které se budou držet k dispozici v řádech desítek dnů, až jednotek měsíců, vyhledávat do jednotek sekund. Stejně tak bude schopno generovat z těchto výsledků grafy v řádu několika málo jednotek sekund.

Během zmiňovaných desítek dnů, až jednotek měsíců přiteče do centrální infrastruktury řádově miliony až desítky (možná i stovky) milionů záznamů, se kterými bude toto řešení muset pracovat.

Specifikace tohoto požadavku se pojí primárně se schopností škálování tímto směrem.

2.2.5 Snadná udržitelnost softwarového řešení

Vybrané řešení musí být snadné udržovat a spravovat. Zároveň je třeba, aby nešlo o aktuálně pouze udržovaný produkt, ale produkt s aktivním vývojem a budoucí vizí.

2.2.6 Snadná rozšiřitelnost softwarového řešení

Vybrané řešení musí být snadno rozšiřitelné o nové formáty dat, protokoly kterými přijímá tyto data a jiné.

2.2.7 Zabezpečená komunikace přes všechny externí kanály

Veškerou komunikaci, která bude probíhat mimo samostatnou infrastrukturu bude možné zabezpečit a to jak na úrovni zabezpečeného přenosu (TLS), tak na úrovni autentizace a autorizace.

Jako validní řešení je považována i integrace s webserverem, zajišťujícím zabezpečenou komunikaci, případně dokonce integraci s nějkým autentizačním mechanismem od HTTP basic authentication[42] dále.

2.2.8 Bezplatný provoz alespoň pro testování funkcionalit

Vybrané řešení musí být možné testovat a nejlépe i nasadit bez nutnosti jakýchkoli investic do licencí / předplatných.

Kapitola 3
Průzkum trhu

Průzkum trhu před výběrem software nebo implementací vlastního řešení je nesmírně důležitý. Důvodem je to, že k výběru software typicky dochází před reálným nasazením, po kterém bude software sloužit společnosti mnoho let. Zároveň je dnes na světě mnoho produktů, které mohou řešit daný problém a čím dál tím více z nich je dokonce i nějakou formou „zdarma“ - a to nejčastěji jako open-source[43]. Je tedy velmi výhodné využít možnost již hotového řešení a pokud to potřebujeme a licence to dovoluje, tak si tento software případně i upravíme.

3.1 Průzkum možných řešení

V rámci průzkumu možných řešení jsem narazil na desítky řešení pro sběr logů. Avšak mnoho z nich jsem musel vyřadit již na počátku prozkoumávání, protože často šlo o řešení typu „ELK stack v cloudu“ - aneb „zasílejte vaše data do cloudu a na této doméně máte web UI“. Nějaká forma „as-a-service“ provozu je dnes velmi populární, nicméně součástí požadavků je i podpora on-premise provozu *(sekce 2.2.1)*, kterému toto řešení odporuje. Přenos z on-premise do různých „as-a-service“ forem je většinou daleko jednodušší, než naopak - je tedy vhodné vybrat ideální on-premise řešení a budoucí migrace do jakékoli formy „as-a-service“ bude možná a pravděpodobně i vcelku jednoduchá.

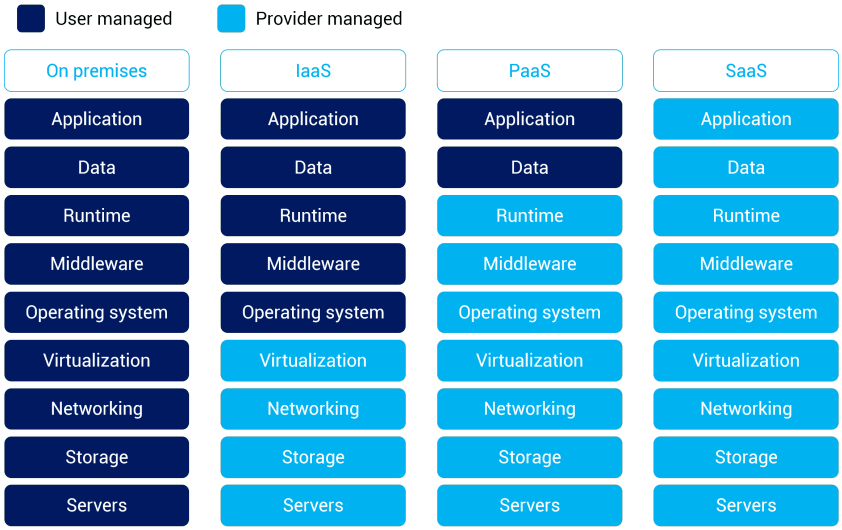
Eliminoval jsem tedy mnoho služeb, jako `Retrace`, `Logentries`, `Logz.io` a další, které nepodporovaly běh on-premise. Zároveň jsem s ohledem na budoucí provoz eliminoval služby, placené od množství logů, jako např `LogDNA`. Nakonec produktů, které stály za hlubší prozkoumání mnoho nezbylo.

Ve zbylých produktech si nelze nevšimnout, mnoha podobností:

- 3 z 4 vybraných řešení používají jako databázi Elasticsearch
- 2 z 4 vybraných řešení se liší pouze v procesoru logů *(Logstash vs. FluentD)*
- 2 z 4 vybraných řešení podporují Beats[44] - utility pro sběr událostí na provozních serverech

Je tedy vcelku zřejmé, že ElasticSearch je velmi mocný, fulltextově vyhledávací engine. Je dokonce někdy označován, jako číslo jedna na poli fulltextového vyhledávání, případně log managementu[45]. ElasticSearch totiž není jen nástroj na sběr logů, ale primárně databáze, která je určena k fulltextovému vyhledávání napříč svými dokumenty.

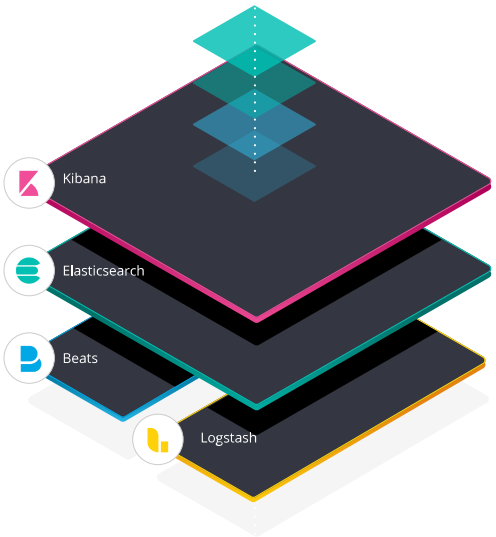
Obrázek 3.1 Rozdíly mezi on-premise a as-a-service řešeními[46]



3.1.1 ELK Stack

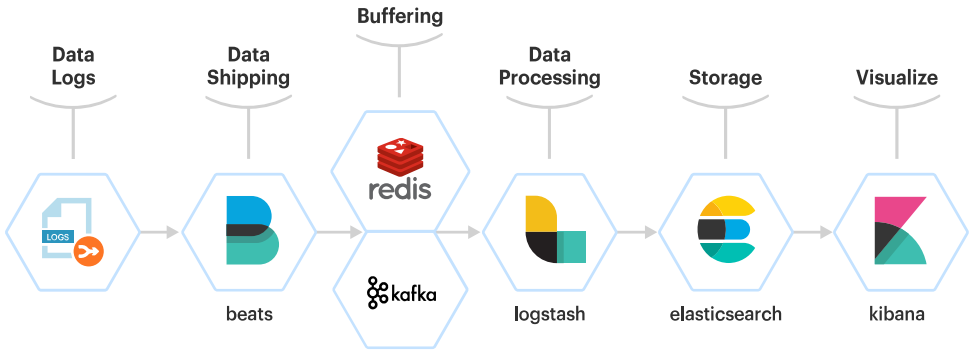
ELK Stack (nebo nověji Elastic Stack) je označení pro open-source ekosystém od Elastic N.V. (`https://elastic.co`). Tento ekosystém se skládá primárně ze tří součástí - ElasticSearch, Logstash a Kibana. Důvodem přesunu od akronymu „ELK Stack“ na název „Elastic Stack“ byl mimo jiné rozvoj dalších součástí a to takzvaných „Beats“, což jsou malé, utility, napsané v Go, zajišťující například přenos dat z aplikačních serverů do infrastruktury zpracovávající logy.

Obrázek 3.2 Elastic Stack[47]



Na obrázku 3.3 je vidět i in-memory databáze Redis a fronta zpráv v podání Kafky. Tyto řešení poskytují vyrovnávání špiček, jak bylo popsáno v kapitole 1.4.6. Elastic Stack podporuje

Obrázek 3.3 Cesta události Elastic stackem po vizualizaci[47]



hned několik možností. Mimo Redis a Kafku podporuje ještě RabbitMQ.

Elastic Stack má možnost bezplatného nasazení pod licencí ELv2[48], která zamezuje hostování elastic stacku, jako službu pro třetí strany - reakce na spravovaný hosting elastic stacku od AWS, proto existuje Amazon OpenSearch, jakožto fork původního Elasticsearche pod Apache licencí[49].

Mnoho funcionalit v Elasticsearchi ale podléhá placené licenci, která není nikde naceněna pro on-premise provoz, je nutné kontaktovat obchodníky. Konkrétně e jedná o možnosti SSO, alertingu, strojového učení a mnoho dalšího.

3.1.1.1 Elasticsearch

Elasticsearch[50] je fulltextový vyhledávač a databáze, založená na vyhledávači Apache Lucene. Komunikace s ním probíhá primárně skrze REST API. Je velmi dobře konfigurovatelný, clusterovatelný, škálovatelný a jeho nody mají schopnost obsluhovat pouze některé součásti clusteru, jakými jsou například:

- **data_hot, data_warm, data_cold, data_frozen, data_content:** Storage tiery, více popsané níže
- **ingest:** node, sloužící k pre-processingu dat, přijímaných do clusteru
- **master:** node, který může být zvolen jako master - neboli řídící stroj clusteru
- **ml:** node, určený pro strojové učení *(ML = Machine Learning)*

  ... a další

Elasticsearch tedy v konextu Elastic Stacku hraje roli úložiště a celkově té „hrubé síly“, zajišťující nejen ukládání dat, ale i jejich tiering, dotazování, indexaci a další. Uvnitř Elasticsearche se data ukládají do takzvaných „dokumentů“, což jsou jednotlivé záznamy. Tyto záznamy se seskupují do „indexů“ (srovnatelné s tabulkou ve světe relačních databází). Tyto indexy jsou rozděleny do takzvaných „shardů“, které mohou být z důvodu výkonu alokovány na oddělených strojích. Nad každým shardem existuje instance Lucene engine, která provádí samotné vyhledávání. Zároveň je index často minimálně před odsunutím do nějaké „studenější“ fáze replikován. Pokud tedy selže nějaký stroj, jehož indexy měli více než replik, potom je index stále dostupný. Toto řešení zajišťuje vysokou dostupnost celého Elasticsearche.

Je ale třeba myslet na to, že každá instance Lucene engine je náročná na zdroje a přílišné množství shardů může vést až k nestabilitě celého clusteru Elasticsearche, což je poměrně obtížně řešitelná situace *(vlastní zkušenost[51])*

Elasticsearch storage tiering Výše zmíněný storage tiering je jednou z dalších velmi mocných vlastností Elasticsearche, která umožňuje různorodé škálování nodů v clusteru a jejich výkonů, úložišť, aj. Každý tier má specifické využití, popsané v dokumentaci. Storage tiery neslouží jen, jako úložiště, ale zároveň je na ně směřován provoz, který upravuje nebo dotazuje dané indexy. Postupně od „hot“ po „frozen “ tier dochází k přesunu od maximalizace výkonu po minimalizaci nároků na zdroje při uchovávání časových dat.

- **Content tier:** Stroje v tomto tieru uchovávají jiný typ obsahu, než ostatní a to data, která nemají svůj životní cyklus.
- **Hot tier:** Stroje v tomto tieru uchovávají indexy s časovými daty, které jsou nejnovější a nejčastěji používané.
- **Warm tier:** Stroje v tomto tieru uchovávají indexy s časovými daty, které jsou starší a méně často navštěvované oproti hot tieru.
- **Cold tier:** Stroje v tomto tieru uchovávají indexy s časovými daty, která jsou ještě starší a méně navštěvovaná a upravovaná oproti datům v cold tieru. Standartně v tomto tieru dochází ke snížení počtu replikace dat na 0 (existují již pouze na tomto stroji z důvodu ušetření místa).
- **Frozen tier:** Stroje v tomto tieru uchovávají indexy s časovými daty, která jsou nejstarší a jsou pouze vyjímečně dotazována a již nejsou vůbec upravována. Tento tier již neudržuje dostupná všechny data ihned, ale pouze na vyžádání, což běžně trvá nezanedbatelnou dobu, ale zato díky mnoha mechanizmům snižuje využití prostředků až 20x oproti warm tieru.[52]

Díky všem těmto vlastnostem dohromady jde o velice robustní nástroj pro řešení mnoha problémů téměř neomezené velikosti. V praxi se využívá, jako součást nejen řešení pro sběr logů, kterých tu několik popisuji, ale také jako fulltextový vyhledávácí engine v Gitlabu[53] a mnoha dalších velkých projektech.

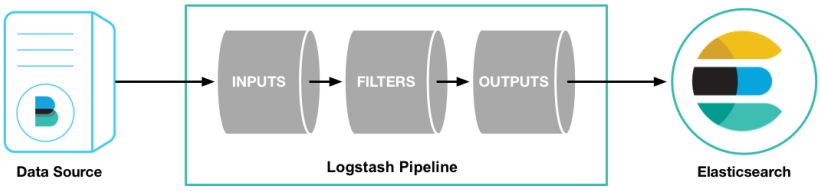
3.1.1.2 Logstash

Logstash[54] je nástroj, který zajišťuje „server-side data processing“, aneb zpracování dat, přijímaných do Elastic Stacku. Fáze zpracování dat Logstashem jsou celkem 3:

- **Input:** V této fázi dochází k přijetí události z množiny možných vstupů, mezi které patří například `beats`, `kafka`, `redis`, `syslog` a mnoho dalších.
- **Filter:** V této fázi dochází k různým změnám, aby výsledná událost dávala smysl v kontextu celého dalšího procesu analytiky, aj. Lze zde standardně používat `if .. else` konstrukty. Zároveň i v této fázi je nepřeberné množství pluginů pro zpracovávání událostí. Mezi nejznámější patří `grok`, sloužící primárně k rozkladu zprávy podle speciální syntaxe na různá pole, odpovídající danému regulárnímu výrazu[55]. Mezi další patří například `date` pro parsování textového pole na datum, kterému rozumí Elasticsearch nebo `mutate` pro operace s poli události *(jako například přejmenování, přidání, odebrání, aj.)* nebo `json` pro zpracování pole typu text, které v sobě obsahuje JSON.
- **Output:** Poslední ze třetice je fáze, která umožňuje přenášet data dále do infrastruktury. Nejčastěji se zde používá `elasticsearch` plugin, který zapisuje data přímo do Elasticsearche. Existuje ale daleko více výstupních plugin;, jako `kafka` pro další přeposílání skrze frontu zpráv nebo pluginy mnoha dohledových systémů, jako `nagios`, `zabbix` a mnoho a mnoho dalších. Pro debugování výstupu je zde samozřejmě možné využít i plugin `file` nebo `csv` nebo obdobné.

Díky možnosti rozšířeni za pomoci pluginů je logstash velice mocný nástroj na zpracování logů. Zároveň je sice psaný částečně v Javě, ale podporuje interpretaci Ruby kódu za pomoci JRuby interpretoru, takže je možné využít oba jazyky. Dokonce Ruby lze použít i jako filter plugin, kam lze vepsat přímo kód, který bude následně interpretován v JRuby.

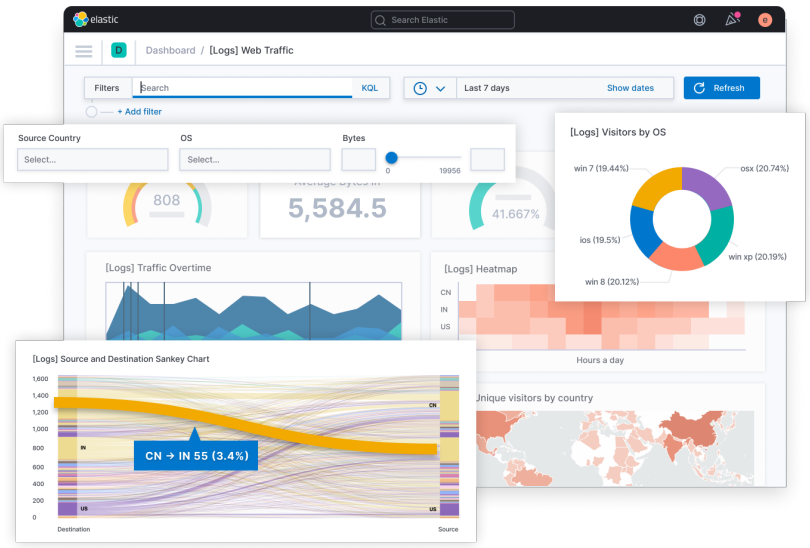
Obrázek 3.4 Ukázka jednoduchého nasazení Logstashe[56]



3.1.1.3 Kibana

Kibana[57] je nástroj pro vizualizaci dat ve webovém prohlížeči a má velmi úzkou integraci na Elasticsearch. Jejím hlavním úkolem je zprostředkovávat agregovaná data z Elasticsearche. Vše okolo, jako správa uživatelských účtů, správa nastavení kibany, aj. jsou uloženy v Elasticsearchi. Umožňuje vše možné od základního zobrazení dat a takzavného „live view“ dat, přicházejících do Elasticsearche, přes základní grafy, mapy *(společně s Logstash filter pluginem* `geoip`*, umžňujícím mapovat IP adresy z dat na souřadnice, kam jsou registrovány)*, až po strojové učení v platinum a vyšší licenci.

Obrázek 3.5 Ukázka Kibany z oficiálního webu[57]



3.1.1.4 Beats

Beats[58] je rodina malých utilit, založených na Go knihovně `libbeat`[59]. Díky tomu, že jsou napsané v Go jsou velice nenáročné na zdroje, vůči výkonu. Proto jsou určeny pro běh na aplikačních serverech, odkud sbírají data a odesílají do Elastic Stacku. Podporují mnoho modulů, umožňujících jednoduchý sběr dat z mnoha aplikací, případně i ze systému.

Díky knihovně `libbeat` všechny beats podporují výstupní moduly, které umožňují výstup odesílat do Elasticsearche, Logstashe, Redisu a RabbitMQ. Pro debugování je možno využít ještě výstup do konzole nebo do souboru. Zároveň podporují mnoho věcí, jako jednotnou konfiguraci v podobě YAML souboru, preprocesory přijatých dat, které umožňují ještě před odesiláním dat přidat nějaká metadata nebo předzpracovat odesílaná data[60].

Základní Beats jsou tyto:

- **Auditbeat:** Audit linuxových serverů - obdobně jako auditd[61]
- **Filebeat:** Sběr logů, žurnálu a případně naslouchání na syslogu
- **Functionbeat:** Sběr dat z cloudových prostředí
- **Heartbeat:** Monitoring dostupnosti služeb
- **Metricbeat:** Sběr metrik z infrastruktury, ostatních beats, aj.
- **Packetbeat:** Monitoring síťového provozu
- **Winlogbeat:** Záznamy událostí ve Windows prostředí

Ostatní beats existují, jako komunitní, případně díky tomu, že jde o open-source projekt, lze dopsat vlastní beat s využitím knihovny `libbeat` nebo dopsat modul do již existujícího beatu.

3.1.2 EFK Stack

EFK stack je alternativou pro ELK Stack, která je ale velice podobná. Alternativou je pouze v oblasti sběru a případnám parsování logů. Jde totiž o kombinaci nástrojů Elasticsearch (popsán v kapitole 3.1.1.1), FluentD a Kibana (popsána v kapitole 3.1.1.3). Hlavním rozdílem je tedy FluentD vs. Logstash + Beats. Na toto porovnání se v této kapitole zaměřím. Teoreticky totiž jde provozovat FluentD současně.

3.1.2.1 FluentD

FluentD[62] je open-source nástroj pro sběr a přeposílání logů. Patří mezi projekty CNCF (Cloud Native Computing Foundation). Tento projekt je napsán plně v Ruby. Narozdíl od Logstashe není pouze nástrojem pro server-side log processing, ale komplexním nástrojem, který zajišťuje i přenos z klienta na server nebo další věci. Zároveň není koncipován na kompatibilitu s Beats (popsaných v kapitole 3.1.1.4). Naopak je koncipován tak, že infrastruktura bude generovat nějaké logy, nějak se dostanou do fluentd (syslog, soubory, docker discovery, zároveň je připraven pro Kubernetes, aj.) a tyto logy budou routovány přímo do logovací infrastruktury nebo do dalšího FluentD, které bude fungovat obdobně, až na to že vstup bude FluentD. Jde tedy o možné zřetězení.

Jediné, s čím je FluentD kompatibilní je bratrský nástroj FluentBit, který je napsán v `C` a je s FluentD plně kompatibilní.

Pluginy: Stejně jako Logstash podporuje FluentD mnoho různých pluginů. Tyto pluginy jsou ale koncipovány daleko více decentralizovaně. FluentD totiž funguje celý na bázi zpracovávání JSONů. Pokud tedy naše aplikace má mít integraci na FluentD, je nutné, aby plugin dokázal přijmout log a ideálně jej v JSON formátu předat dále. Díky implementaci v Ruby je možné využít, stejně jako u Logstashe parsovací enginy, jako je `grok`.

Narozdíl od Logstashe ve FluentD nefungují žádné `if .. else` konstrukty, ale celý tok dat je založen na principu tagů a labelů.

O FluentD lze stejně jako o Logstashi říci, že má několik oddělených fází zpracování události, které slouží ke speciálním účelům:

- **Source:** V této fázi definujeme vstupy, které otagujeme a případně zparsujeme vestavěnými pluginy, jako např `nginx` nebo využijeme např parser `grok`. Z této fáze vychází již zpracovaný, JSON, který s sebou nese tagy a případně labely, podle kterých je dále rozponáván.
- **Filter:** V této fázi lze konfigurovat různé transformace, filtrace a další operace s událostmi. Příkladem je třeba `grep` plugin, který funguje k filtraci, obdobně jako linuxový `grep`.

- **Match:** V této fázi lze konfigurovat výstupy na základě shody tagů / labelů. FluentD zde dokonce podporuje i bufferování do souborů / do paměti pro případ nějakého výpadku endpointu, kam se data odesílají. Lze tedy říci, že tímto nahrazuje fronty zpráv nebo Redis u Logstashe.

Srovnání Logstashe s FluentD: Porovnání mezi Logstashem a FluentD je dohledatelných mnoho. Pro ukázku zde uvádím obrázek 3.6, který popisuje základ. Jako další rozdíly lze uvést i to, že FluentD je psáno v CRuby, které je narozdíl od JRuby, ve kterém je psaný Logstash, odlehčenější z důvodu absence Javy. Stejně tak lze podotknout, že FluentD sice poskytuje vestavěnou vysokou dostupnost, ale u Logstashe toto není překážkou. Lze u něho využít například Kafku ve vysoké dostupnosti a více nezávislých instancí Logstashe se potom může dělit o přicházející zprávy.

Obrázek 3.6 Základní srovnání FluentD a Logstashe[63]

	Platform	Event Routing	Plugin Ecosystem	Transport	Performance
Logstash	Linux & Windows	Algorithmic statements	Centralized	Deploy with Redis for reliability.	Uses more memory. Use Elastic Beats for leafs.
Fluentd	Linux & Windows	Tags	Decentralized	Built-in reliability but hard to configure.	Uses less memory. Use Fluent Bit and Fluentd Forwarder for leafs.

Každé řešení má tedy své výhody i nevýhody a je určeno do trochu rozdílných prostředí. Na druhou stranu mají mnoho společného a oba jsou to skvělé nástroje.

3.1.3 Graylog

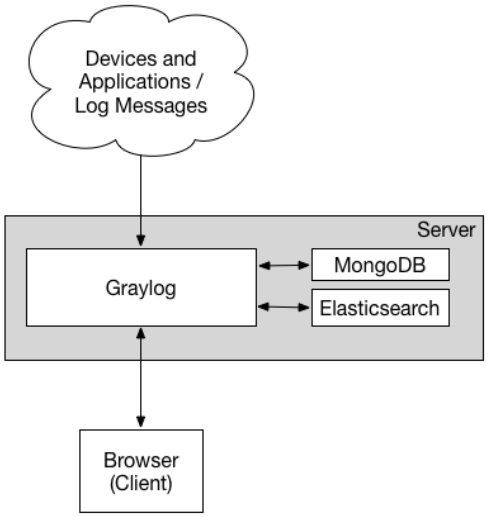
Graylog[64] se od předchozích dvou řešení velmi liší. Nejde totiž o sadu nástrojů, kde by každý zajišťoval jeden kus práce, ale jde o jeden velký, open-source nástroj, napsaný v Javě. Graylog využívá dva databázové backendy:

- **Elasticsearch:** Popsán v kapitole 3.1.1.1, primárně využíván pro ukládání dat a zároveň využíván, jako vyhledávácí engine.
- **MongoDB:** Dokumentová NoSQL databáze, primárně využívána jako úložiště konfigurace pro celý Graylog stack (v případě HA nasazení).

Díky tomu, že Graylog využívá stejně, jako předchozí dvě řešení pro vyhledávání a ukládání dat Elasticsearch, má velice podobé schopnosti vyhledávání a práce s velkým objemem dat. Primárním rozdílem je tedy příjem a zobrazení dat.

Příjem dat: Příjem dat je zajišťován množstvím pluginů, které jsou buď již vestavěné v základní instalaci nebo je lze stáhnout a doinstalovat z Graylog Marketplace *(`https://community.graylog.org/c/marketplace`)* nebo dokonce dopsat. Je tedy možné využít jak standardní protokoly, jako syslog nebo fronty zpráv, jako Kafka, aj. Zároveň ale existují integrace s Beats (popsány v kapitole 3.1.1.4).

Obrázek 3.7 Minimální nasazení Graylogu[64]



Marketplace: Výše zmíněný marketplace není určen pouze pro vstupní pluginy, ale i pro všemožné dashboardy a další tzv. „content packy“. Je tedy poměrně snadné do Graylogu dodat nějakou funkcionaitu, kterou již někdo vytvořil, ale není součástí hlavního balíčku.

Zobrazení dat: Graylog dokáže za pomoci Elasticsearche, svých pluginů a dalších funkcionalit vizualizovat data obdobně, jako Kibana. Nicméně nejde o obecný nástroj, ale o specializovaný nástroj pro sběr logů. Je tedy možné, že Graylog nebude mít tolik vizualizací, jako Kíbana, ale určitě nejde o blokující věc. Vždy lze do Graylogu požadované vizualizace dodat - když nijak jinak, tak dopsáním.

Vizualizace dat prbíhá u uživatele v prohlížeči - Graylog je primárně napsán pro zpracování samotné logiky a s javascriptovým frontendem komunikuje skrze API.

Výstupy: I přesto, že je Graylog sám o sobě komplexním řešením, zajišťující centralizovaný sběr logů s mnoha dalšími možnostmi, obsahuje i sekci „Output“. Lze tedy z Graylogu exportovat data do dalšího Graylog clusteru, GELF formátu *(popsáno níže)*, TCP, UDP a mnoho dalších cest pro případnou další integraci.

GELF: Mimo standardní způsoby přenosu logů, používané i jinde má Graylog specializovaný formát **GELF**, aneb „Graylog Extended Log Format“. Tento formát byl vytvořen, jako náhrada syslog protokolu, který má mnoho nevýhod a problémů, kde některé z nich jsem popisoval v kapitole 1.2. Jde o JSON formát, který má specifický formát, popsaný v dokumentaci Graylogu. GELF zprávu lze poslat pomocí TCP, UDP i HTTP(S), jde tedy o velmi univerzální, ale nestandardizovaný způsob přenosu událostí. Jeho maximální velikost byla rozšířena na až 65535 bytů skrze UDP. Skrze další protokoly se maximální velikost může lišit. Do zprávy se tedy vejde narozdíl od syslogu značná část například stacktrace z Javy nebo pythonu.

Výpis kódu 3.1 Příklad zprávy který je uvedený v dokumentaci GELF formátu[65]

```
{
       "version": "1.1",
       "host": "example.org",
       "short_message": "A short message that helps you identify what is going on",
       "full_message": "Backtrace here\n\nmore stuff",
       "timestamp": 1385053862.3072,
       "level": 1,
       "_user_id": 9001,
       "_some_info": "foo",
       "_some_env_var": "bar"
}
```

Lze si zde všimnout rozdílu mezi povinnými a nepovinnými poli *(povinná nemají na začátku klíče podtržítko)* a stejně tak je zde ukázána podpora víceřádkových logů.

Vysoká dostupnost: Vzhledem k tomu, že je Graylog postaven stejně jako Elastic Stack nebo EFK Stack na Elasticsearchi, jako databázi, tak jeho zajištění vysoké dostupnosti je obdobné. Primární rozdíl je zde v tom, že pro konfiguraci využívá další databázi a to MongoDB. MongoDB ale nativně také podporuje nasazení ve vysoké dostupnosti a to ať už pomocí replica setu nebo shardování, obdobně jako u Elasticsearche[66]. A nakonec i Graylog samotný podporuje nasazení ve více instancích.

3.1.4 PLG Stack

PLG stack je od předchozích řešení velmi odlišný. Nejde o obrovské aplikace, napsané v Javě nebo podobných jazycích, ale jde o sadu tří poměrně mladých nástrojů s velmi rychlým vývojem.

3.1.4.1 Prometheus

První písmeno představuje Prometheus. Prometheus je projekt, který vznikl v roce 2012 v SoundCloudu a je teprve druhým projektem inkubátoru CNCF, hned po Kubernetes. Tento nástroj je primárně určen se sadou mnoha a mnoha svých exporterů *(nástrojů, zajišťujících export dat, různým způsobem a zpřístupňujících je Prometheu skrze HTTP(S))* k velmi častému sběru metrik.

S ohledem na skutečnost, že nejde o nástroj, určený ke sběru logů, nebudu se jím více v této práci zabývat.

3.1.4.2 Loki

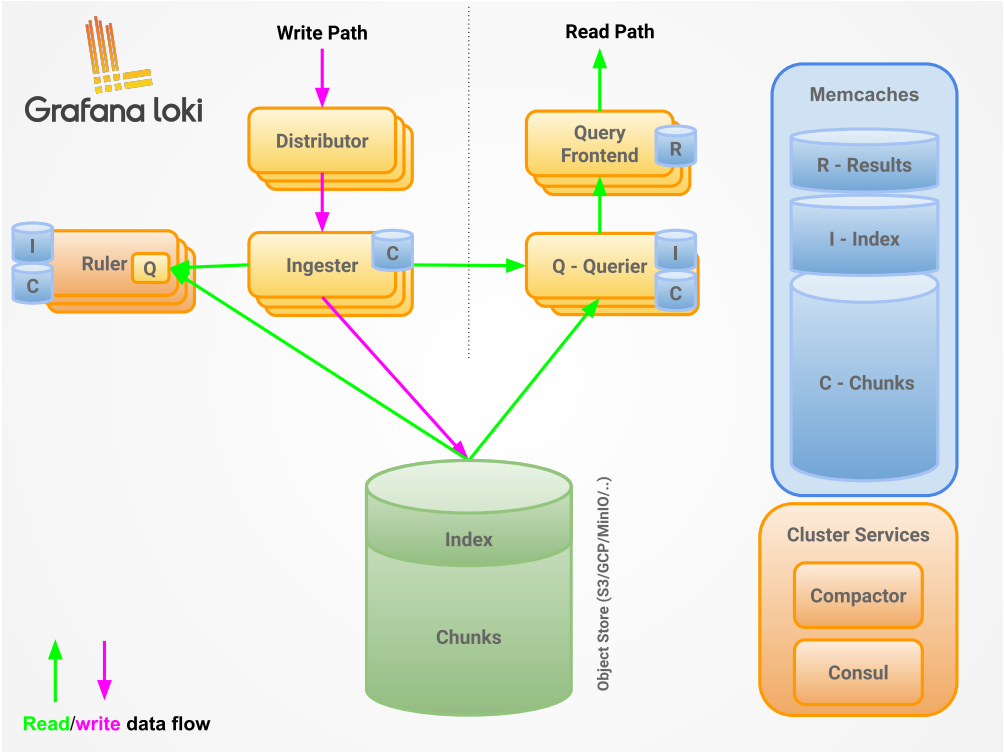
Loki[67] je společně s Promtailem *(jde vpodstatě o jeden projekt, Promtail popsán v kapitole 3.1.4.3)* velice mladý projekt, který byl oznámen teprve v roce 2018. Je to horizontálně škálovatelný, multitenantní systém pro agregaci logů, inspirovaný ve značné míře Prometheem. Je výrazně orientovaný směrem do cloudu, tudíž mezi jeho hlavní výhody patří kompatibilita s cloudovými, objektovými úložišti, jako jsou GCS, AWS S3 a další.

Částečným problémem je, když Loki neprovozujeme v cloudu. Způsob jednoduchého uložení dat na souborový systém zamezuje jakémukoli škálování. Je ale samozřejmě možné využít Ceph a jeho S3 API[68], případně Cassandra DB nebo nějaký distribuovaný souborový systém. Ukládání většího množství dat na souborový systém EXT4 *(který je nejpoužívanější v Linuxových systémech)* byl dokonce v dřívějších verzích problém, dokud nedošlo k přidání možnosti zvětšit velikost jednoho souboru[69].

Výhodou Loki je jeho možnost spuštění stejné binárky *(napsáno v Go, takže výstupem je binárka, prakticky bez jakýchkoli závislostí, jako u Javy JRE, apod.)*, jako monolitické aplikace

a stejně tak jako microservice. Loki má mnoho modulů (viz. obrázek 3.8) a stačí v konfiguraci konkrétní instance nastavit, zda se mají spouštět na této instanci všechny moduly nebo jen nějaký konkrétní.

Obrázek 3.8 Architektura Loki[70]



3.1.4.3 Promtail

Promtail je součástí projektu Loki a je určen ke sbírání logů různými způsoby a odesílání skrze HTTP(S) API do Loki, obdobně, jako například Beats *(popsaných v kapitole 3.1.1.4)*. Jde o velice odlehčenou aplikaci, napsanou v Go, díky čemuž obdobně, jako Beats nemá žádné závislosti.

Umožňuje několik způsobů vstupu logů a to sice:

- **File target discovery:** Logy ze souborů v dané cestě
- **Kubernetes discovery:** Integrace s Kubernetes, obdobně jako Beats nebo FluentdD
- **Journal scraping:** Sběr logů z Linuxového žurnálu
- **Windows event log:** Sběr logů z windowsího logu událostí
- **Gcplog:** Sběr logů z Google Cloud Platform
- **Syslog receiver:** Sběr logů ze syslogu (kompatibilní pouze s RFC 5424)
- **Kafka:** Sběr událostí z fronty zpráv Kafka
- **GELF:** Integrace s Graylogem *(popsán výše v 3.1.3)* a jeho GELF formátem
- **Cloudflare:** Sběr logů z Cloudflare

Promtail / Loki původně nepodporovalo víceřádkové záznamy, což byl poměrně zásadní problém, který byl vyřešen až v posledním půl roce *(konec roku 2021, začátek roku 2022)*[71][72]. Pro Mnoho programovacích jazyků jde totiž o velmi důležitou možnost, jak odeslat stack trace nebo další typy záznamů.

3.1.4.4 Grafana

Grafana[73] je multiplatformní, vizualizační a analytický nástroj, který je napsán v kombinaci Javascriptu (konkrétně v jeho nadstavbě TypeScript) a Go. Jde tedy jako u zbytku stacku o velice rychlý nástroj. Grafanu je možné provozovat jak on-premise, tak využít Grafana Cloudu, což je provoz grafany, jakožto služby.

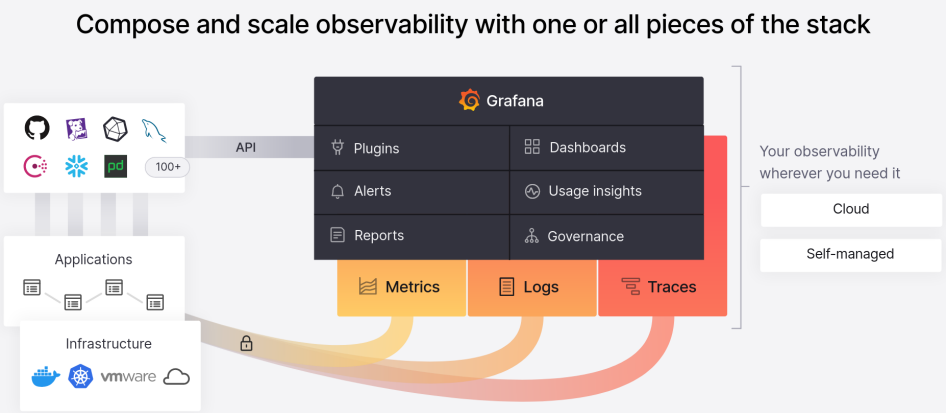
Grafana je postavena, jako podporující rozšiřitelnost pluginy a takzvanými „data sources“ - neboli zdrojemi dat, nad kterými bude vizualizace prováděna. Díky oblíbenosti grafany bylo již napsáno obrovské množství pluginů a data sources pro mnoho užití.

Mezi nejzajímavější data sources díky kterým je Grafana velice multifunkčním nástrojem pro vizualizaci a analytiku patří například:

- **Prometheus, InfluxDB, OpenTSDB:** Časové databáze pro metriky
- **Loki, Elasticsearch:** Logovací a dokumentové databáze
- **PostgreSQL, MySQL, MSSQL:** SQL databáze
- **Jaeger, Zipkin, Tempo:** Trasovací nástroje
- **Google cloud, CloudWatch, Azure Monitoring:** Monitorování cloudů

  ... a mnoho dalších

Obrázek 3.9 Ukázka integrace nástrojů od Grafana Labs[74]



Grafana mimo jiné podporuje i alerting, takže společně s velice rozmanitými data sources je možné monitorovat a upozorňovat na obrovskou škálu potenciálních problémů.

Licenční politiky Jde o open-surce platoformu, která ale má jako mnohý další software některé funkcionality dostupné až v Grafana Enterprise. Jde primárně o další věci typu přihlášení pomocí SAML *(OAuth2 podporuje již ve free licenci)*, enterprise data sources / pluginy, jako například `SAP HANA` nebo `AppDynamics` data sources, ale také zjemnění přístupových práv, synchronizace skupin / týmů z externích serverů identit, support, a další.

Vysoká dostupnost Pokud chceme provozovat v on-premise prostředí Grafanu ve vysoké dostupnosti, tak je zapotřebí zajistit nějaký databázový backend s vysokou dostupností. Grafana ve výchozím stavu využívá SQLite3 databázi, ale lze využít jak MySQL, tak PostgreSQL, které obě podporují nasazení ve vysoké dostupnosti.

3.2 Finální výběr

Jako finální řešení k dalšímu testování a návrhu implementace byl vybrán ELK Stack / Elastic Stack *(kapitola 3.1.1)*, Graylog *(kapitola 3.1.3)* a PLG stack *(kapitola 3.1.4)*. EFK Stack byl vyřazen pro velkou duplicitu s ELK Stackem a horší „učící křivku“ a z důvodu chybějících `if .. else` konstruktům, které jsou nahrazeny tagy. Logstash mne také zaujal možností více oddělených pipelines - asi ideálním přístupem je co vstup, to separátní pipeline.

Všechny systémy, které byly popsány v této kapitole jsou velice kvalitní kusy software, kde volba ani jednoho nebude zásadním problémem. Rozhodl jsem se je ale blíže srovnat a otestovat v reálném prostředí.

Kapitola 4

Podrobné srovnání

V rámci podrobného srovnání se budu věnovat jednak srovnání splnění funkčních a nefunkčních požadavků *(tabulky 4.1 a 4.2)*, ale i jednoduchosti zajištění těchto požadavků. Většina open-source nástrojů je sice výborná, ale je potřeba k nim dodat další open-source software, jako například `keepalived` zajišťující VRRP, `Kubernetes` pro provozní prostředí, zajišťující orchestraci mikroservis nebo další nástroje, jako nějaký OAuth2/OIDC provider nebo alespoň HTTP(S) proxy, jako třeba `nginx`, `apache2` nebo `trafeik`.

Budu se zde věnovat jen 3 ze 4 výše popsaných. Toto bylo vysvětleno v kapitole 3.2.

4.1 Srovnání splnění funkčních požadavků

Tabulka 4.1 Splnění funkčních požadavků na systém

Funkční požadavek z kapitoly 2.1	**ELK** 3.1.1	**Graylog** 3.1.3	**PLG** 3.1.4
2.1.1 Podpora víceprotokolového logování	✓	✓	✓
2.1.2 Přístupnost přes webové rozhraní	✓	✓	✓
2.1.3 Přístupnost přes API	✓	✓	✓
2.1.4 Podpora zálohování, případně verzování konfigurace	✓	✓	✓
2.1.5 Podpora uživatelských účtů, rolí a práv	✓	✓	✓
2.1.6 Podpora zobrazování přijatých logů	✓	✓	✓
2.1.7 Podpora filtrace přijatých logů	✓	✓	✓
2.1.8 Podpora vizualizace nasbíraných dat	✓	✓	✓
2.1.9 Podpora nějaké formy retence dat	✓	✓	✓

Podpora víceprotokolového logování Všechna výše zmíněná řešení podporují příjem logů skrze syslog protokol *(minimálně RFC 5424, popsáno v kapitole 1.2)* a logů, které aplikace zapisuje do souborů. Zároveň každé z řešení podporuje nějakou formu příjmu logů pomocí HTTP API a případně i front zpráv.

Přístupnost přes webové rozhraní Všechna výše zmíněná řešení podporují přístup přes webové rozhraní. U Graylogu je to jeho javascriptový frontend. U Elastic stacku je to Kibana. U PLG stacku je to Grafana.

Přístupnost přes API Všechny výše zmíněná řešení podporují přístup skrze API. Některá dokonce hned na několika místech. Například u Elastic stacku řeší všechna práva přímo Elasticsearch, je tedy možné přistupovat napřímo k němu. U ostatních řešení toto lze primárně zprostředkovaně, skrze Grafanu nebo Graylog. Je však možné pro administrátorské zásahy využít i API Loki nebo Elasticsearche pod Graylogem napřímo.

Podpora zálohování, případně verzování konfigurace Elasticsearch, Logstash, Beats, Prometheus a Loki se konfigurují primárně z textových souborů, které lze jednoduše verzovat i zálohovat.

Stejně tak Grafana podporuje částečně verzování konfigurace skrze speciální složku. Zbytek konfigurace aj. ukládá Grafana do databáze, kterou je ve výchozím nastavení SQLite3, ale podporuje i PostgreSQL a MySQL, jak byo výše zmíněno. Tyto databáze lze jednoduše zálohovat, nikoli ale smysluplně verzovat.

Kibana a Graylog svá nastavení ukládají do Elasticsearche, respektive MongoDB, které nelze tak dobře verzovat. V tento moment je výhodné využít zálohování databází a v případě potřeby porovnávat jejich textovou podobu.

Podpora uživatelských účtů, rolí a práv Elastic stack podporuje jednotné nastavení uživatelských rolí a práv v Elasticsearchi, proti kterému se všechny operace běžně provádí.

Graylog si řeší vlastní uživatelské účty a právový model nad daty, která sice jsou uložena v Elasticsearchi, ale je k nim přistupováno pod jedním nebo žádným uživatelem (záleží na nastavení Elasticsearche).

Loki a Prometheus stadnardně neřeší žádnou autentizaci, vyjma HTTP basic authentication pro zamezení přístupu k datům. Grafana nad těmito „data sources“ vytváří uživatelů a případných týmů, kterým zpřístupňuje v rolích Administrator, Editor nebo Viewer určité tzv. organizace. V enterprise verzi podporuje i nějakou formu RBAC podle popisu.

Podpora zobrazování přijatých logů Všechna výše zmíněná řešení podporují režim zobrazení přijatých zpráv. Tyto zprávy jsou ale velice často již rozložené na jednotlivé pole z důvodu možnosti pokročilejší práce nad daty.

Podpora filtrace přijatých logů Elastic stack a Graylog podporují filtrování díky tomu, že Elasticsearch podporuje Lucene dotazy a Kibana jej ještě obaluje KQL.

PLG stack podporuje vlastní filtrační nástroje, jakými jsou dotazovací jazyky PromQL pro Prometheus a LogQL.

Podpora vizualizace nasbíraných dat Všechna výše zmíněná řešení obsahují podporu vizualizace dat.

V Graylogu je to jeho Graylog Web UI, v Elastic stacku je to Kibana a v PLG stacku je to Grafana.

Podpora nějaké formy retence dat Elasticsearch obsahuje takzvanou ILM, aneb *„Index Lifecycle Management“*, což je možnost, jak definovat odrotovávání indexů do starších fází a jako poslední je `delete phase`, která je pro odstraňování dat ze stacku.

U PLG stacku je to tak, že Prometheus není určen pro dlouhodobé ukládání dat, tedy vyžaduje nastavení o počtu dní nebo velikosti dat, které má ukládat (a zbytek automaticky maže).

Loki také podporuje nastavení retence dat, ikdyž jej standardně nevyžaduje. Je tedy pouze volitelná.

4.2 Srovnání splnění nefunkčních požadavků

Tabulka 4.2 Splnění nefunkčních požadavků na systém

Nefunkční požadavek z kapitoly 2.2	**ELK** 3.1.1	**Graylog** 3.1.3	**PLG** 3.1.4
2.2.1 Podpora on-premise provozu	✓	✓	✓
2.2.2 Podpora horizontální i vertikální škálovatelnosti	✓	✓	✓
2.2.3 Podpora vysoké dostupnosti řešení	✓	✓	✓
2.2.4 Podpora práce s velkým množstvím záznamů	✓	✓	✓
2.2.5 Snadná udržitelnost softwarového řešení	✓	✓	✓
2.2.6 Snadná rozšiřitelnost softwarového řešení	✓	✓	✓
2.2.7 Zabezpečená komunikace přes všechny externí kanály	✓	✓	✓
2.2.8 Bezplatný provoz alespoň pro testování funkcionalit	✓	✓	✓

Podpora on-premise provozu Všechna výše zmíněná řešení jsou distribuována pod licencí, umožňující tento kus software volně stáhnout nebo přistoupit k jeho zdrojovému kódu a nasadit pro vlastní / komerční potřebu. Elastic license 2.0 (`ELv2`) ale neumožňuje provozovat jakékoli součásti Elastic Stacku, jakožto službu pro třetí strany. Nemůžeme tedy hostovat Elastic stack pro další společnosti.

Podpora horizontální i vertikální škálovatelnosti Všechna výše zmíněná řešení podporují horizontální škálování a to ať už nasazením více instancí Elasticsearche, Kibany, Graylogu nebo MongoDB, a nebo nasazením Loki v mikroservisním režimu, kde je pak možné každý modul naškálovat dle potřeby.

PLG stack k horizontálnímu škálování potřebuje ještě distribuované úložiště, jako je např. CassandraDB, které pak odbourá i zmiňovaný problém s vertikálním škálováním.

Ostatní služby jsou založeny převážně na Javě, která si poradí s přidáváním RAM a CPU výkonu až do velmi vysokých hodnot.

Podpora vysoké dostupnosti řešení Všechna výše zmíněná řešení lze nasadit ve vysoké dostupnosti. Elasticsearch, MongoDB, CassandraDB a mají nativní podporu nasazení ve vysoké dostupnosti.

Kibanu, Grafanu a Graylog lze bez problému také nasadit ve vysoké dostupnosti, jen mezi nimi a uživatelem bude potřeba nasadit nějaký loadbalancer / http proxy, jako například `nginx` s `keepalived` pro VRRP, případně anycastování provozu na úrovni infrastruktury. S Logstashem je to už ale o trochu horší. Lze jej nasadit ve více instancích a před něho postavit například Kafku nebo Redis ve vysoké dostupnosti, které zajistí předávání zpráv více instancím, nicméně žádná nativní podpora pro clusterizaci neexistuje.

Pro zajištění vysoké dostupnosti Loki je zapotřebí také minimálně HTTP load balanceru, plus nasazení v mikroservisním režimu[75], ke kterému je třeba právě např. zmíněná CassandraDB. Zároveň jde o cloudově orientovaný software, takže pro jeho nasazení jsou ideální Kubernetes, zajišťující orchestraci zmiňovaných mikroservis.

Podpora práce s velkým množstvím záznamů Všechny výše zmíněná řešení jsou připraveny na práci s velkým množstvím dat. Kromě PLG stacku je pod ostatními řešeními Elasticsearch, který je na tento úkol primárně určen. Nedělají mu problémy miliardy záznamů i řádově více. Je třeba ale myslet na opatrnost s počtem shardů, zmíněných v kapitole 3.1.1.1.

Loki s Promtailem zvládnou také zpracovávat obrovské množství záznamů. Při vývoji PLG stacku je celkově kladen velký důraz na výkon služeb vs. množství zpracovaných dat.

Snadná udržitelnost softwarového řešení Všechna výše zmíněná řešení jdou s vývojem rychle dopředu a vzhledem k tomu že jde o špičku v takhle důležitém odvětví, není třeba se bát blízkého zániku / opuštění projektu vývojáři. PLG stack dokonce pochází z inkubátoru CNCF, stejně jako dnes nejpoužívanější orchestrátor kontejnerů, Kubernetes.

Snadná rozšiřitelnost softwarového řešení Všechna výše zmíněná řešení podporují mnoho formátů, jakými lze logy sbírat, procesovat, aj. Je tedy velice jednoduché rozšířit dané prostředí o další formát zpráv nebo další místo, odkud budou logy přitékat.

Zabezpečená komunikace přes všechny externí kanály Všechna výše zmíněná řešení podporují komunikaci skrze HTTP, které buď dokáží samy nebo s pomocí jakékoli HTTP(S) proxy obalit do šifrované komunikace. Logy lze také přijímat nejedním zabezpečeným kanálem *(případně s pomocí nějakého zabezpečení fronty zpráv, aj.)*.

Podpora http basic authentication nebo lepší je zde také podporována všude.

Bezplatný provoz alespoň pro testování funkcionalit Všechna výše zmíněná řešení podporují bezplatný provoz dokonce i pro mnoho produkčních nasazení.

Kapitola 5
Testovací nasazení

V rámci testovacího nasazení jsem testoval všechna tři řešení. Ani jedno z řešení ale nebylo nasazeno ve vysoké dostupnosti z důvodu omezení komplexity.

Jako zdroj dat jsem využil data ze switchů, které jsou umístěny v rámci Dejvické budovy Fakulty elektrotechnické, virtualizace, VPN koncentrátoru aj. Tato data byla přijímána pomocí programu `rsyslog`, který umožňuje logy jak zpracovat, tak přeposlat dále ve formátu, který odpovídá syslog protokolu podle RFC 5424[18].

Tato data byla přeposílána jak do Graylogu, tak do Elastic stacku a také do PLG stacku. Z důvodu jednoduchosti nasazení byla využita pouze jedna instance Elasticsearche, která obsluhovala Elastic stack a zároveň Graylog. Zároveň nebyly aplikovány všechny bezpečnostní mechanismy Elasticsearche, které by jinak byly zapotřebí - to jak komunikace po HTTPS, tak užívatelské účty a role.

Všechna řešení dokázala správně přijmout syslog zprávu, podle standardu RFC 5424. Daleko větší problém ale nastal, když jsem chtěl nad logy dělat nějakou formu analytiky. Obsah zprávy ve všech systémech zůstává nezpracovaný. Toto je způsobeno tím, že obsah zprávy není nijak standardizován a je tedy zapotřebí pro každý typ aplikace nebo zařízení nastavit vlastní zpracující pipeline.

Pro ukázku zde uvedu několik zpráv z různých zařízení:

Výpis kódu 5.1 Příklad události z Cisco switche

```
Jun 10 10:06:23.305: %LINEPROTO-5-UPDOWN: Line protocol on Interface
    ↪ GigabitEthernet1/0/7, changed state to down
```

Výpis kódu 5.2 Příklad události ze SoftEther VPN koncentrátoru

```
[zeu1/VPN] (2022-06-10 10:06:28.585) <SERVER_LOG>: Connection "CID-12558765" connected
    ↪ using Virtual Hub Admin Mode. The name of the Virtual Hub is "DEFAULT".
```

Výpis kódu 5.3 Příklad události z VSphere

```
Event [12632970] [1-1] [2022-06-10T10:06:31.454337Z]
    ↪ [vim.event.UserLogoutSessionEvent] [info] [VSPHERE.LOCAL\Administrator] []
    ↪ [13652979] [User VSPHERE.LOCAL\Administrator@192.168.150.172 logged out (login
    ↪ time: Friday, 10 June, 2022 10:06:31 AM, number of API invocations: 1, user
    ↪ agent: Go-http-client/1.1)]
```

Jak lze vidět na ukázkách, tak každý systém si obsah zprávy strukturuje jinak. Dokonce v rámci produktů od VMware jsem nalezl, že dokonce existují různé struktury obsahu událostí

v různých modulech. Pokud je tedy v rámci nějaké virtualizace nasazeno několik produktů od VMware, může dojít k nutnosti zpracovávat události na základě jednotlivých modulů.

Lze tedy zvolit 2 přístupy a to:

1. **Neřešit parsování:** Toto řešení je vcelku jednoduché a umožňuje na základě regulárních výrazů, metadat, aj. stále filtrovat přijaté logy a zároveň velice rychle škálovat přes různé aplikace

2. **Nastavit důkladně parsování pro každou aplikaci:** Toto řešení je daleko složitější než předešlé, nicméne dovoluje využít naplno sílu vyhledávacích enginů, jako Elasticsearch, který je použit pro 2 ze 3 vybraných řešení. Třetí vybrané řešení dokonce nedovoluje parsovat nijak lépe zprávu, pokud nejde o JSON nebo o key-value formát, což jej značně omezuje. Na druhou stranu tento přístup v kombinaci s dotazovacím jazykem LogQL[76] z něho dělá bleskově rychlý systém pro správu logů vhodný do některých typů nasazení.

Výpis kódu 5.4 Příklad strukturované key-value události z Promethea

```
ts=2022-06-10T11:00:02.280Z caller=head.go:1009 level=info component=tsdb msg="WAL
    ↪ checkpoint complete" first=15353 last=15354 duration=1.20098403s
```

V rámci testovacího nasazení jsem dále zjistil, že problém s parsováním řeší Graylog Marketplace *(`https://marketplace.graylog.org`)* jen velmi omezeně, protože například pro VMware sice existuje content-pack, ale s velice neobecným nastavením, spíše dochází k problémům typu kolize portů s jinými inputy, problémy s kolizemi pipelines, aj.

Došlo dokonce k tomu, že když jsem na jednom portu přijímal všechny výše zmíněné zdroje událostí, tak nebylo jednoduše možné vysvětlit Graylogu, aby různé patterny nepřijímaly vše, ale jen data na základě jiných polí nebo metadat *(např.:* `if src_ip == "1.1.1.1" { ... }`*)*. Bylo tedy zapotřebí rozdělovat vstupy nebo správně poskládat patterny, aby nedocházelo ke kolizím a nakonec dát nějaký obecný pattern, který přijme vše.

V Elasticsearchi je primárním problémem to, že Logstash nepodporuje syslog podle RFC 5424[18], ale jen podle obsolete RFC 3164[16]. Toto bylo vyřešeno pomocí utility Filebeat *(z rodiny Beats)*, která podporuje 3 formáty: `rfc3164`, `rfc5424` a `auto`, který automaticky detekuje formát (je to i výchozí volba).

Utilita Filebeat tedy může naslouchat na TCP / UDP a předávat logy buď přímo do Elasticsearche nebo skrze Logstash, který může provádět další zpracování.

Kapitola 6

Finální výběr a návrh implementace

Jako vítěze jsem vybral Elastic Stack, který je velice robustním a schopným řešením pro sběr a zpracování aplikačních a infrastrukturních logů. Toto řešení bylo vybráno na základě předchozích kapitol, kde jsem prozkoumal trh, vybral možné kandidáty a nakonec jsem měl všechna 3 možná řešení nasazené v testovacím režimu.

Hlavním důvodem k finálnímu výběru byla robustnost řešení a zároveň nejvyšší možnost přizpůsobení zpracující pipeline a podpůrné infrastruktury Beats. Zároveň Kibana je velmi mocný nástroj, umožňující mnoho věcí od online streamingu logů, přes mnoho a mnoho typů grafů a map po dashboardy s live filtracemi.

Mimo tyto funkcionality Elastic stack disponuje mnoha funkcemi, které ale již nejsou dostupné pod běžnou licencí, ale jsou překryty enterprise licencí, jejíž cenu neznám. Těmito funkcionalitami je například SIEM nebo strojové učení. Licenci je ale možné zakoupit kdykoli.

Jako poslední výhodu bych zde vyzdvihl vydávání releasů v prostředí Elastic Stacku. Vždy, když vychází release, tak vychází u všech udržovaných součástí naráz. Tímto je umocněna kompatlbilita jednotlivých komponent stacku.

PLG stack byl vyřazen z důvodu toho, že jde o velmi odlehčený ekosystém, který se rychle vyvíjí, ale stále mnoho věcí není schopen řešit a je koncipován více do cloudového prostředí.

Graylog byl vyřazen z toho důvodu, že má horší zpracující pipeline a sice má nativní integraci s Beats, **ale** nejde o součást Graylog ekosystému, ale o externí utility. Dále aktuální verze Graylogu *(verze 4.3.x)* dle dokumentace podporuje Elasticsearch verze nižší než 7.11, která v sobě **nemá opravenou verzi knihovny log4j2 vůči prosincovým, vážným log4j2 zranitelnostem[77]** *(počínaje CVE-2021-44228[78] s pokračováním přes několik dalších CVE)*. Jde tedy o poměrně problematický kus software, co se týče bezpečnosti, když jeho hlavní databáze nemůže být upgradována na novější verzi než takovou, která je rok a půl stará *(verze 7.10.2[79])* a zranitelná velice známým útokem se skóre závažnosti 10 z 10 (CVSS 3.x).

6.1 Návrh implementace

Pro splnění funkčních a nefunkčních požadavků na nasazení Elastic stacku je zapotřebí primárně zajistit vysokou dostupnost Elasticsearche. Toho lze dosáhnout při minimálně 3 strojích v clusteru. Elasticsearch standardně vytváří indexy s `number_of_replicas: 1`, což znamená že každý

index je jednou replikován na další stroj - je tedy ve dvou kopiích. Z definice vysoké dostupnosti tedy vychází, že potřebujeme více jak 2 stroje, abychom byli imunní proti výpadku nějakého stroje *(bez ztráty zdraví clusteru nebo výkonu)*.

Aby byly Kibana a Logstash nasazené ve vysoké dostupnosti, je ideální je spustit na všech 3 strojích, společně s Elasticsearchem. Mezi uživatele a Kibanu lze postavit `keepalived`, zajišťující VRRP *(„plovoucí IP adresu“)* a HTTP proxy, jako například `nginx` nebo `haproxy`.

Jak bylo zmíněno výše, tak Logstash nepodporuje nativně vysokou dostupnost, takže je zapotřebí nasadit mezi něho a zdroje dat nějakou frontu zpráv nebo využít zmíněné `keepalived`. Využití Apache Kafka ve vysoké dostupnosti má za efekt ještě schopnost rozložení zátěže mezi Logstashe. Kafka potřebuje pro vysokou dostupnost také 3 stroje, kvůli replikačnímu faktoru, stejně jako Elasticsearch. Je tedy záhodno, aby také běžela na všech 3 strojích.

Pro sběr logů ze syslogu lze využít `keepalived` a na každém stroji Filebeat, naslouchající na syslog portech, případně v kombinaci s `rsyslog`em pro archivaci logů do souborů *(pokud ji chceme)*.

Pokud tedy nasadíme takto 3 identické stroje s:

- Keepalived pro VRRP
- Beats pro příjem logů pomocí syslogu *(podporuje narozdíl od Logstashe příjem podle obou RFC)*
- Apache Kafka jako frontu zpráv ve vysoké dostupnosti
- Logstashem pro zpracování logů
- Elasticsearchem jako hlavním motorem celého clusteru
- Kibanou jako vizualizačním nástrojem
- Nginxem nebo HAProxy jako HTTP loadbalancerem
- ... a nejlépe Ansiblem nebo SaltStackem nebo jiným IaC nástrojem pro konfiguraci

tak dosáhneme kýženého výsledku velmi schopného ekosystému pro sběr infrastrukturních a aplikačních logů ve vysoké dostupnosti se všemi funkcionalitami, které byly posbírány během psaní této práce.

Zároveň při nasazení 3 identických strojů bude cluster imunní výpadku jakéhokoli stroje, bez vyjímky.

Kdekoli v infrastruktuře můžou samozřejmě běžet další fronty zpráv nebo Redis, které budou poskytovat data Logstashi a úplně obejdou Kafku a další podpůrné služby uvnitř clusteru, ale to už záleží na požadavcích a možnostech zařízení, ze kterých se budou sbírat logy. Tato možnost se může například hodit v dnes moderním IoT.

Poslední věc, která v návrhu nebyla zmíněna je sběr dat například ze souborů, které leží na serverech v infrastruktuře nebo aplikací, jako je `Docker`, případně jiných, které logují na standardní (chybový) výstup. Pro tuto možnost a případně i další je tu samozřejmě přípravena podpora spuštění Beats na aplikačních serverech nebo přeposílacích uzlech. Tyto Beats pak budou odesílat logy přímo do Kafky, umístěné na logovacím clusteru *(nikoli přes syslog nebo další protokoly)*.

6.2 Odhad finanční a časové náročnosti implementace

Finanční a časová náročnost velmi záleží na celkových potřebách firmy a různorodosti implementace. Jde ale shrnout do několika bodů, ze kterých bude již výpočet daleko přesnější. Odhady jsou počítány na administrátora, který již Elastic Stack nějakým způsobem chápe. Nasazení bez jakékoli znalosti je samozřejmě zatížené časem, potřebným k naučení se všech technologií s tím spojených.

- **Nasazení Elasticsearche, Logstashe a Kibany v HA:** Do cca 3 MD, s tím, že je zapotřebí nejdříve nasadit Elastic Stack s komunikací pomocí HTTPS, spuštěním autentizace a až poté je možné řešit spojení s dalšími částmi stacku. Doporučuji při tomto nasazení využít monitoring sebe sama v rámci Elastic Stacku.
- **Nasazení Kafky v HA:** Max 1 MD, spíše méně
- **Nasazení zbytku stacku:** Jde o Keepalived, Nginx nebo HAProxy. S tím spojené věci jako HTTPS / Syslog certifikáty, aj. Toto zabere řádově do 1 MD
- **IaC:** Automatizace konfigurace a udržování infrastruktury velmi záleží na zkušenostech s automatizačními nástroji, jejichž učící křivka je velmi příjemná. Odhaduji maximálně zdvojnásobení času, který by zabralo ruční nasazení, zmíněné v bodech výše. Výhodou je ale sebedokumentující vlastnost a možnost udržovat infrastrukturu neustále konzistentní.
- **Monitoring:** Pokud nepůjde o vastní monitoring sebe sama v Elastic Stacku, tak půjde o maximálně 2 MD. Využití pouze vlastního monitoringu nedoporučuji a to hned z mnoha důvodů. Primárně: když se Elastic stack nějak fatálně rozbije, neřekne vám to - bude totiž rozbitý.
- **Nastavení nového vstupu:** Tato sekce popisuje nastavení nového **typu** vstupu. Tím je myšlo například nový typ zařízení, nikoli „další kus Cisco switche, kterých máme již 100 v budově“. Toto může zabrat až několik MD na typ vstupu, ale většinou jde pouze o jednotky hodin.

Když sesumarizuji body, sepsané výše, tak toto řešení může být velice robustně sestaveno za jeden až 2 týdny administrátora na plný úvazek s implementací základních vstupů, jako jsou například síťové prvky.

Co se finanční náročnosti týká, tak pro základní sběr logů a případné tvorby grafů nad nimi je zapotřebí pouze 3 instancí nějakého linuxového stroje. Pokud nepůjde o virtuální stroje, uvnitř již existující virtualizace nebo privátního cloudu, bude zapotřebí opatřít tyto stroje. Stroje nepotřebují žádné sdílené úložiště, ale musí mít každý ideálně svou SSD storage.

Investice do tohoto hardware lze tedy odhadnout od cca 200 000 Kč do několika milionů. Záleží na požadovaném výkonu strojů. Vždy ale doporučuji škálovat spíše horizontálně, než vertikálně. To z toho důvodu, že je vždy rychlejší zotavení po výpadku jednoho menšího stroje, než jednoho obřího, udržujícího třetinu dat, kterých bude řádově více než jednotky terabajtů.

Další finanční nároky mohou být za licence, jejichž cenu jsem nezjišťoval a to z důvodu nutnosti poptávky u obchodníka Elasticu. Licence je zapotřebí pro mnoho věcí, popsaných na `https://www.elastic.co/subscriptions`. Některé ale stojí za zmínku i zde:

- Jakákoli podpora SSO: A to jak login pomocí SAML, OIDC a Kerberos, tak pomocí LDAP, PKI nebo Windows ActiveDirectory. Toto je ale často řešitelné serverem pro správu identit, který synchronizuje účty a hesla i do Elasticsearche.
- Alerting: A to všechny typy alertingu, jako Slack, Teams, Email, aj. kromě zápisu do indexu a do server logu.
- Strojové učení: Využitelné pro detekci anomálií a mnoho dalšího.

Kapitola 7
Závěr

Cílem této bakalářské práce bylo analyzovat požadavky firmy na systém pro zpracování aplikačních a infrastrukturích logů. Na základě požadavků bylo zapotřebí analyzovat trh, vybrat potenciální kandidáty, otestovat je a navrhnout optimální implementaci. Za tímto účelem byla vytvořena případová studie, v rámci které jsem provedl sběr funkčních a nefunkčních požadavků na tento systém.

Na základě těchto požadavků jsem provedl průzkum trhu, vyřadil mnoho kandidátů z důvodu problematičnosti nasazení na vlastní infrastruktuře. Dále bylo vybráno několik málo kandidátů, které jsem blíže prozkoumal a provedl testovací nasazení několika z nich. Na základě testování byl jako vítěz vybrán Elastic Stack, který je komplexním řešením, které je schopno splnit všechny požadavky a poskytuje další možnosti navíc. Některé možnosti nebyly prozkoumány z důvodu absence jiné něz open-source / basic licence, pod kterou je zdarma dostupný. Tyto funkcionality jsem ale také prozkoumal a krátce popsal.

Výsledkem této práce je tedy ucelený návrh implementace, zhotovený na základě průzkumu trhu a testování v reálném prostředí. Zároveň je v práci nastíněno množství finančních a časových prostředků, nutných k realizaci mého návrhu.

Bibliografie

1. *Why logging is important?* [online]. [B.r.]. [cit. 2022-02-01]. Dostupné z: https://www.syslog-ng.com/community/b/blog/posts/why-logging-is-important.
2. DEMIAN, John. *Why log management is so important - dzone devops* [online]. 2021. [cit. 2022-02-01]. Dostupné z: https://dzone.com/articles/why-is-log-management-so-important-and-how-can-it.
3. TUNIS, Jean. *Logging and monitoring: Why you need both* [online]. 2021. [cit. 2022-02-01]. Dostupné z: https://thenewstack.io/logging-and-monitoring-why-you-need-both/.
4. *Log files* [online]. Apache Software Foundation, [b.r.] [cit. 2022-02-04]. Dostupné z: https://httpd.apache.org/docs/2.4/logs.html.
5. *What is full-stack observability?* [online]. AppDynamics, 2022-01 [cit. 2022-02-02]. Dostupné z: https://www.appdynamics.com/topics/what-is-full-stack-observability.
6. CISCO.COM. *Připravte své datové centrum na příští dekádu* [online]. [cit. 2022-02-02]. Dostupné z: https://www.cisco.com/c/dam/m/cs_cz/training-events/webinars/data-center-2022/dc-event-prepare-your-data-center-on-accuracy-decad-1.pdf.
7. LOGROTATE. *The logrotate utility is designed to simplify the administration of log files on a system which generates a lot of log files.* [online]. [B.r.]. [cit. 2022-03-12]. Dostupné z: https://github.com/logrotate/logrotate.
8. *systemd-journald.service* [online]. [B.r.]. [cit. 2022-03-12]. Dostupné z: https://www.freedesktop.org/software/systemd/man/systemd-journald.service.html.
9. *The rocket-fast syslog server* [online]. 2020. [cit. 2022-03-14]. Dostupné z: https://www.rsyslog.com/.
10. *View logs for a container or service* [online]. 2022. [cit. 2022-04-22]. Dostupné z: https://docs.docker.com/config/containers/logging/.
11. BRAY, Tim. *The JavaScript Object Notation (JSON) Data Interchange Format* [RFC 8259]. RFC Editor, 2017. Request for Comments, č. 8259. Dostupné z DOI: 10.17487/RFC8259.
12. *Configure logging drivers* [online]. 2022. [cit. 2022-04-22]. Dostupné z: https://docs.docker.com/config/containers/logging/configure/.
13. *Autodiscover: Filebeat reference* [online]. [B.r.]. [cit. 2022-04-25]. Dostupné z: https://www.elastic.co/guide/en/beats/filebeat/current/configuration-autodiscover.html#_docker_2.
14. *Rsyslog documentation - imdocker* [online]. 2018. [cit. 2022-04-25]. Dostupné z: https://www.rsyslog.com/doc/master/configuration/modules/imdocker.html.

15. *Eric Allman* [online]. [B.r.]. [cit. 2022-03-04]. Dostupné z: http://www.neophilic.com/~eric/.
16. LONVICK, Chris M. *The BSD Syslog Protocol* [RFC 3164]. RFC Editor, 2001. Request for Comments, č. 3164. Dostupné z DOI: 10.17487/RFC3164.
17. BRADNER, Scott O. *Key words for use in RFCs to Indicate Requirement Levels* [RFC 2119]. RFC Editor, 1997. Request for Comments, č. 2119. Dostupné z DOI: 10.17487/RFC2119.
18. GERHARDS, Rainer. *The Syslog Protocol* [RFC 5424]. RFC Editor, 2009. Request for Comments, č. 5424. Dostupné z DOI: 10.17487/RFC5424.
19. GASSICK, Larene Le. *Analyze syslog messages with SEQ* [online]. Structured Blog, 2020 [cit. 2022-05-19]. Dostupné z: https://blog.datalust.co/seq-input-syslog/.
20. *Rsyslog documentation* [online]. 2018. [cit. 2022-03-02]. Dostupné z: https://www.rsyslog.com/doc/v8-stable/configuration/templates.html#reserved-template-names.
21. ROSENCRANCE, Linda. *What is Siem and why is it important?* [online]. TechTarget, 2020 [cit. 2022-05-15]. Dostupné z: https://www.techtarget.com/searchsecurity/definition/security-information-and-event-management-SIEM.
22. *Siem and security analytics on the Elastic Stack: Elastic Security* [online]. [B.r.]. [cit. 2022-05-15]. Dostupné z: https://www.elastic.co/siem/.
23. *11 open source log collectors for centralized logging* [online]. 2021. [cit. 2022-05-15]. Dostupné z: https://geekflare.com/open-source-centralized-logging/.
24. FUYOU, Miao; YUZHI, Ma; SALOWEY, Joseph A. *Transport Layer Security (TLS) Transport Mapping for Syslog* [RFC 5425]. RFC Editor, 2009. Request for Comments, č. 5425. Dostupné z DOI: 10.17487/RFC5425.
25. OKMIANSKI, Anton. *Transmission of Syslog Messages over UDP* [RFC 5426]. RFC Editor, 2009. Request for Comments, č. 5426. Dostupné z DOI: 10.17487/RFC5426.
26. GERHARDS, Rainer; LONVICK, Chris M. *Transmission of Syslog Messages over TCP* [RFC 6587]. RFC Editor, 2012. Request for Comments, č. 6587. Dostupné z DOI: 10.17487/RFC6587.
27. ROSE, Dr. Marshall T.; NEW, Darren. *Reliable Delivery for syslog* [RFC 3195]. RFC Editor, 2001. Request for Comments, č. 3195. Dostupné z DOI: 10.17487/RFC3195.
28. SSL, Tým podpory. *Co Je to PKI?* [online]. 2021. [cit. 2022-04-29]. Dostupné z: https://www.ssl.com/cs/co-je-pki/amp/.
29. IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks. *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*. 2018, s. 1–1993. Dostupné z DOI: 10.1109/IEEESTD.2018.8403927.
30. [online]. [B.r.]. [cit. 2022-05-10]. Dostupné z: https://aws.amazon.com/message-queue/.
31. [online]. [B.r.]. [cit. 2022-05-12]. Dostupné z: https://aws.amazon.com/message-queue/benefits/.
32. *Rabbitmq vs redis: Top 9 differences you should know* [online]. 2021. [cit. 2022-05-07]. Dostupné z: https://www.educba.com/rabbitmq-vs-redis/.
33. ELASTIC. *Logstash-forwarder/protocol.md at master · elastic/logstash-forwarder* [online]. [B.r.]. [cit. 2022-05-02]. Dostupné z: https://github.com/elastic/logstash-forwarder/blob/master/PROTOCOL.md.
34. ILYUKHA, Vitaliy. *Functional vs Non-Functional Requirements: Ultimate guide* [online]. 2020. [cit. 2022-06-12]. Dostupné z: https://jelvix.com/blog/functional-vs-nonfunctional-requirements.

35. *Functional vs non-functional requirements [updated 2021]* [online]. [B.r.]. [cit. 2022-05-15]. Dostupné z: https://enkonix.com/blog/functional-requirements-vs-non-functional/.
36. *Functional vs non-functional requirements - understand the difference* [online]. 2020. [cit. 2022-05-20]. Dostupné z: https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/.
37. *ILM: Manage the index lifecycle: Elasticsearch Guide [8.2]* [online]. [B.r.]. [cit. 2022-05-21]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/index-lifecycle-management.html.
38. MŮČKA, Jan. *IaaS, Paas a SAAS aneb V čem se liší služby „as a service"* [online]. 2021. [cit. 2022-05-21]. Dostupné z: https://www.master.cz/blog/iaas-paas-a-saas-aneb-v-cem-se-lisi-sluzby-as-a-service/.
39. *What is high availability? definition and FAQ* [online]. 2022. [cit. 2022-05-21]. Dostupné z: https://us.sios.com/what-we-do/high-availability/.
40. SUSE. *What is a computer cluster?: Answer from SUSE defines* [online]. 2018. [cit. 2022-05-21]. Dostupné z: https://www.suse.com/suse-defines/definition/computer-cluster/.
41. *Single-master and multi-master replication in DBMS* [online]. 2021. [cit. 2022-05-21]. Dostupné z: https://www.geeksforgeeks.org/single-master-and-multi-master-replication-in-dbms/.
42. RESCHKE, Julian. *The 'Basic' HTTP Authentication Scheme* [RFC 7617]. RFC Editor, 2015. Request for Comments, č. 7617. Dostupné z DOI: 10.17487/RFC7617.
43. GARBADE, Dr. Michael. *Free software vs open source vs freeware: What's the difference? - dzone open source* [online]. DZone, 2020 [cit. 2022-05-21]. Dostupné z: https://dzone.com/articles/free-software-vs-open-source-vs-freeware-whats-the.
44. *Beats: Data Shippers for Elasticsearch* [online]. [B.r.]. [cit. 2022-05-22]. Dostupné z: https://www.elastic.co/beats/.
45. *The Complete Guide to the elk stack* [online]. 2020. [cit. 2022-05-26]. Dostupné z: https://logz.io/learn/complete-guide-elk-stack/.
46. *What is iaas?* [online]. [B.r.]. [cit. 2022-05-22]. Dostupné z: https://www.alibabacloud.com/knowledge/what-is-iaas.
47. *The Elk Stack: From the creators of Elasticsearch* [online]. [B.r.]. [cit. 2022-05-28]. Dostupné z: https://www.elastic.co/what-is/elk-stack.
48. *Elastic license 2.0* [online]. [B.r.]. [cit. 2022-05-29]. Dostupné z: https://www.elastic.co/licensing/elastic-license.
49. MEADOWS, Carl; GRAYBILL, Jules; DAVIS, Kyle; SHAH, Mehul. *Blogs* [online]. AWS Open Source Blog, 2021 [cit. 2022-05-29]. Dostupné z: https://aws.amazon.com/blogs/opensource/introducing-opensearch/.
50. *Elasticsearch: The Official Distributed Search & Analytics engine* [online]. [B.r.]. [cit. 2022-05-29]. Dostupné z: https://www.elastic.co/elasticsearch/.
51. DUŠÁTKO, Vojtěch. *Failing cluster* [online]. 2021. [cit. 2022-05-28]. Dostupné z: https://discuss.elastic.co/t/failing-cluster/282474.
52. *Data tiers: Elasticsearch guide [8.2]* [online]. [B.r.]. [cit. 2022-05-28]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/data-tiers.html.
53. *Gitlab Advanced Search* [online]. [B.r.]. [cit. 2022-05-28]. Dostupné z: https://docs.gitlab.com/ee/user/search/advanced_search.html.

54. *Logstash: Collect, parse, transform logs* [online]. [B.r.]. [cit. 2022-05-29]. Dostupné z: https://www.elastic.co/logstash/.
55. *A beginner's guide to logstash grok* [online]. 2021. [cit. 2022-05-29]. Dostupné z: https://logz.io/blog/logstash-grok/.
56. *Stashing your first event: Logstash reference [8.2]* [online]. [B.r.]. [cit. 2022-05-29]. Dostupné z: https://www.elastic.co/guide/en/logstash/current/first-event.html.
57. *Kibana: Explore, visualize, Discover Data* [online]. [B.r.]. [cit. 2022-05-29]. Dostupné z: https://www.elastic.co/kibana/.
58. *Beats: Data Shippers for Elasticsearch* [online]. [B.r.]. [cit. 2022-05-29]. Dostupné z: https://www.elastic.co/beats/.
59. ELASTIC. *Beats/libbeat at main · elastic/beats* [online]. [B.r.]. [cit. 2022-05-29]. Dostupné z: https://github.com/elastic/beats/tree/main/libbeat.
60. *Define processors: Filebeat reference [8.2]* [online]. [B.r.]. [cit. 2022-05-29]. Dostupné z: https://www.elastic.co/guide/en/beats/filebeat/current/defining-processors.html.
61. *AUDITD(8) - linux man page* [online]. [B.r.]. [cit. 2022-05-29]. Dostupné z: https://linux.die.net/man/8/auditd.
62. *Fluentd docs* [online]. Fluentd, [b.r.] [cit. 2022-05-31]. Dostupné z: https://docs.fluentd.org/.
63. *Fluentd vs Logstash: A comparison of log collectors* [online]. 2021. [cit. 2022-05-31]. Dostupné z: https://logz.io/blog/fluentd-logstash/.
64. *Graylog Documentation* [online]. [B.r.]. [cit. 2022-06-01]. Dostupné z: https://docs.graylog.org/v1/docs.
65. *Gelf* [online]. [B.r.]. [cit. 2022-06-01]. Dostupné z: https://docs.graylog.org/docs/gelf.
66. CLOUD, Alibaba. *High-availability mongodb Cluster Configuration Solutions* [online]. Medium, 2018 [cit. 2022-06-02]. Dostupné z: https://alibaba-cloud.medium.com/high-availability-mongodb-cluster-configuration-solutions-465cc82cd0bc.
67. LABS, Grafana. *Grafana Loki* [online]. [B.r.]. [cit. 2022-06-02]. Dostupné z: https://grafana.com/docs/loki/latest/.
68. *Ceph Object Gateway S3 API* [online]. [B.r.]. [cit. 2022-06-02]. Dostupné z: https://docs.ceph.com/en/latest/radosgw/s3/.
69. GRAFANA. *Loki (in Docker) reports ño space left on deviceḃut thereś plenty of spaceinodes · issue 1502 · Grafana/Loki* [online]. [B.r.]. [cit. 2022-06-02]. Dostupné z: https://github.com/grafana/loki/issues/1502.
70. *Components* [online]. [B.r.]. [cit. 2022-06-02]. Dostupné z: https://grafana.com/docs/loki/latest/fundamentals/architecture/components/.
71. LABS, Grafana. *Multiline* [online]. [B.r.]. [cit. 2022-06-03]. Dostupné z: https://grafana.com/docs/loki/latest/clients/promtail/stages/multiline/.
72. GRAFANA. *Feature: Multi-line logs · issue 74 · Grafana/Loki* [online]. [B.r.]. [cit. 2022-06-03]. Dostupné z: https://github.com/grafana/loki/issues/74.
73. *Grafana documentation* [online]. [B.r.]. [cit. 2022-06-04]. Dostupné z: https://grafana.com/docs/grafana/latest/.
74. *Grafana: The Open Observability Platform* [online]. Grafana Labs, [b.r.] [cit. 2022-06-04]. Dostupné z: https://grafana.com/.
75. *Loki deployment modes* [online]. Grafana Labs, [b.r.] [cit. 2022-06-07]. Dostupné z: https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/.

76. *LogQL* [online]. Grafana Labs, [b.r.] [cit. 2022-06-10]. Dostupné z: https://grafana.com/docs/loki/latest/logql/.

77. *Apache Log4j security vulnerabilities* [online]. [B.r.]. [cit. 2022-06-13]. Dostupné z: https://logging.apache.org/log4j/2.x/security.html.

78. *Vulnerability details : CVE-2021-44228* [online]. [B.r.]. [cit. 2022-06-13]. Dostupné z: https://www.cvedetails.com/cve/CVE-2021-44228/.

79. ELASTIC. *Release elasticsearch 7.10.2 · Elastic/Elasticsearch* [online]. [B.r.]. [cit. 2022-06-13]. Dostupné z: https://github.com/elastic/elasticsearch/releases/tag/v7.10.2.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `thesis` ..... repozitář textu práce ve formátu LaTeX
  - `ctufit-thesis.pdf` ..... text práce ve formátu PDF