Sem vložte zadání Vaší práce.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Server pro sběr a prezentaci logů

Bc. Jiří Borský

Katedra softwarového inženýrství
Vedoucí práce: Ing. Zdeněk Balák

6. května 2021

Poděkování

Rád bych poděkoval všem, kteří mi pomohli a podporovali mě ať již při studiu nebo psaní této diplomové práce. V prvé řadě bych velice rád poděkoval svému vedoucímu bakalářské práce Ing. Zdeňku Balákovi, který si pravidelně vyhradil čas, aby se se mnou sešel. Další v řadě, komu bych chtěl poděkovat, je Ing. Jakub Průša, za jeho velkou podporu a skvělou komunikaci při specifikování této práce. Dále bych chtěl poděkovat všem přátelům, například Janu Zimolkovi, Bc. Aničce Škorupové, Bc. Štěpu Tužilovi a dalším. V neposlední řadě bych chtěl poděkovat rodině, která mě ve studiu podporovala.

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. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 6. května 2021 . . . . . .

Odkaz na tuto práci

Borský, Jiří. *Server pro sběr a prezentaci logů*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato diplomová práce popisuje vývoj funkčního prototypu serveru pro sběr logů z mobilních zařízení, který může jednoduše běžet jak na serveru, tak i na lokálním stroji. Daný prototyp je určen pro zadavatelskou firmu, zabývající se vývojem mobilních aplikací. Pojednává se také o procesu hledání nejlepšího řešení logovací knihovny, jejímž prostřednictvím bude možné posílat logy na zmíněný server z mobilní aplikace. Tato práce popisuje sběr funkčních a nefunkčních požadavků na server pro sběr logů a logovací knihovnu. Dále práce důkladně analyzuje různá stávající řešení jak serverové části, tak i knihovny. Na to navazuje návrh, ve kterém se pojednává o architektuře serveru a splnění požadavků jak na server, tak i na knihovnu. Práce také popisuje realizaci jejich řešení. Na závěr tato práce popisuje testování funkčního prototypu serveru pro sběr logů.

Klíčová slova Graylog, Elastic Search, ELK Stack, logovací knihovna, logovací server

Abstract

This diploma thesis describes the development of a functional server prototype for collecting logs from mobile devices, which can easily run both on the server and on the local machine. The prototype is intended for a contracting company engaged in the development of mobile applications. It also discusses the process of finding the best solution for the logging library, through which it will be possible to send logs to the mentioned server from a mobile application. This work describes the collection of functional and non-functional requirements to the server for log collection and logging library. Furthermore, the work thoroughly analyzes various existing solutions for both the server part and the library. This is followed by a design part that discusses the architecture of the server and meeting the requirements for both the server and the library. The work also describes the implementation of these solutions. Finally, this work describes the testing of a functional prototype server for log collection.

Keywords Graylog, Elastic Search, ELK Stack, logging library, logging server

Obsah

Seznam obrázků

Úvod

Logování je již mnoho let neoddělitelnou součástí problémů v průběhu vývoje aplikací. Poskytuje nám celkový pohled na běh naší aplikace a co se v ní děje. Logy často obsahují důležité a užitečné informace, bez kterých by bylo hledání chyb v programu či aplikaci prakticky nemožné.

Při vývoji aplikací se nevyhneme situacím, kdy aplikace spadne, nebo si uživatelé začnou stěžovat na chyby, které se projevují pouze za určitých okolností a je obtížné je identifikovat.

Zadavatelská firma se již nějakou dobu zabývá vývojem mobilních aplikací a takovéto centralizované řešení ušetří vývojářům spoustu času a firmě spoustu peněz.

Zadavatelská firma se již nějakou dobu zabývá vývojem mobilních aplikací a rozhodla se realizovat centralizované řešení logování, které ušetří vývojářům spoustu času a firmě spoustu peněz.

Toto zadání jsem si vybral, protože náplní mé práce je vývoj a oprava mobilních aplikací, takže čtením logů strávím značnou část svého času. Také se stává, že některé chyby ani nejsme schopni navodit a musíme jet k zákazníkovi, abychom si mohli stáhnout logy z jejich zařízení. Proto bych ocenil, kdybychom měli ve firmě lepší nástroj, který tento problém pomůže řešit.

Kapitola 1

Cíl práce

Obecným cílem této magisterské práce je funkční prototyp serveru pro sběr logů, který bude v budoucnu nasazen v zadavatelské firmě, kde bude sbírat logy z mobilních zařízení, na kterých poběží aplikace vyvíjené firmou. Dalším cílem této práce vytvořit logovací knihovnu, pomocí které bude možné posílat logy na server.

Cílem analytické části je analýza funkčních i nefunkčních požadavků na logovací server a knihovnu. Proběhne také analýza stávajících řešení jak serverové části, tak i logovací knihovny, jestli již neexistuje vhodné řešení.

Cílem návrhové části části je architektura serveru. Cílem implementační části je pak jeho realizace a též i realizace knihovny. V testovací části je pak cílem zjistit, jestli server odpovídá stanoveným požadavkům na výkon serveru.

Kapitola 2

Úvod do problematiky

V dnešní době existuje spousta společností, které vytváří nové programy a aplikace. Tyto aplikace pak vytváří záznamy o běhu a případně pak i záznamy o chybách, které se za běhu objeví. Aplikace se mohou lišit v mnoha věcech.[1] Příklady odlišností jsou například:

- účel aplikace
- jazyk implementace (programovací jazyk ve kterém jsou aplikace napsané)
- platforma na které běží

Tyto logy[1] se sice liší, ale většinou je chce vývojář mít u sebe. Pokud jde o nějaký server ke kterému má vývojář alespoň vzdálený přístup, pak si je může snadno stáhnout nebo prohlížet vzdáleně. Toto však není tak snadné udělat například u mobilních zařízení. Tam se nelze jednoduše připojovat na cizí zařízení a stahovat si z nich potřebná data.

Když bychom nahlédli do nějakého souboru s logy, případně kdybychom si otevřeli nějaký terminál, tak bychom pravděpodobně viděli logů spousty. U velkých aplikací se generují snadno i miliony záznamů za den. Takovéto soubory pak mívají až gigabajtové velikosti. Ne vždy však musíme mít takhle velké soubory. Vytváříme-li zbytečně moc logů, tak je to samozřejmě na škodu. Snižuje to výkon aplikace či serveru a máme pak spoustu textu, který je zbytečný.

2.1 Struktura logů

Logy obecně mohou mít spoustu podob. Jejich struktura může být velice jednoduchá, například pouhý text o několika slovech, nicméně to by nám ne-

[1] Tak říkáme záznamům. (Z anglického slova log, což znamená záznam.)

muselo být zas tak prospěšné. Aby se logy daly dobře používat, čímž myslím abychom z nich byli schopní něco vyčíst, tak je děláme nějak strukturované.[2]

Následující ukázky budou z logovací knihovny pro platformu Android. Obecně neexistuje standard, který by určoval, co přesně musí logy obsahovat či jak mají být strukturované. Nicméně se často velmi podobají a proto jsem zvolil konkrétní knihovnu. Navíc cílem této práce je centralizovaný server pro správu logů z mobilních zařízení, takže se s logováním na platformě Android ještě setkáme dále.

2.1.1 Hladiny závažnosti

Příkladem informace, kterou od nás logovací knihovny často vyžadují, je informace o závažnosti záznamu, který se právě chystáme udělat. Zde je příklad ze standardní logovací knihovny pro Android[3], kde jsou hladiny závažnosti od nejmenší po největší:

- **Verbose** - je používáno na nevýznamné nebo málo významné události z aplikace. Kvůli tomu může pak těchto logů vznikat velké množství.
- **Debug** - je používáno hlavně při vývoji aplikace, kdy si pomocí toho můžeme vypisovat průchod aplikací, případně nějaké hodnoty, kterých proměnné aplikace nabývají.
- **Info** - je používáno na vypisování užitečných informací. Příkladem může být úspěšné navázání spojení se serverem.
- **Warn** - je používáno při podezřelém chování či stavu aplikace. Neznamená to ještě že nastala chyba, ale že bychom si měli zkontrolovat, co a proč se stalo.
- **Error** - je používáno ve chvíli, kdy v aplikaci nastane chyba. Často to bývá v případě, že chytíme nějakou výjimku[2] a tu pak k výpisu přidáme jako dodatečná data o chybě, abychom pak celkem snadněji mohli najít příčinu chyby.
- **Assert** - je používán případě že vývojář neočekává, že něco takového nastane. Pokud k něčemu takovému dojde, je potřeba to ihned řešit.

Tyto hladiny závažnosti nám usnadňují následné pročítání logů, protože na základě závažnosti v nich můžeme hledat či si je filtrovat. V některých prohlížečích si můžeme i měnit barvu jednotlivých hladin a tím pak zvýšit přehlednost celého výpisu.

[2]Výjimka (v angličtině exception) je způsob, kterým se často v programování označuje nějaká závažná chyba. Je to událost, která nastala při běhu programu a narušila jeho normální běh.[4]

2.1.2 Formát

V tom nám často pomáhají již připravené knihovny, které tam spoustu informací přidávají za nás, abychom měli snazší práci, a také po nás nějaké informace vyžadují. Běžně za nás doplňují časovou značku, což se nám může velice hodit. Pokud by nám například spadl server a víme, že neběží od nějakého času, tak si snadno můžeme dohledat logy z tohoto daného času a ty prozkoumat.
Zde máme konkrétní formát logů v Androidu[5]:

date time PID-TID/package priority/tag: message

V ukázce formátu vidíme několik částí. Používáme-li na prohlížení logů nějaký chytrý nástroj, tak v nich můžeme snadno vyhledávat podle daných hodnot. Můžeme si například zobrazit data podle PID, priority či tagu.

date - Datum, kdy byl log zaznamenán.
time - Čas, kdy byl log zaznamenán.
PID - Identifikační číslo procesu, pod kterým aplikace běžela.
TID - Identifikační číslo vlákna, které log zapsalo.
package - Balíček ve kterém je aplikace implementována. Aplikace by měla mít unikátní název v rámci balíčku.
priority - Hladina závažnosti logu.
tag - Nějaký identifikátor, většinou se používá název třídy, kde se log zaznamenává. Tento identifikátor nesmí být ale delší než 27 znaků.
message - Zpráva o tom, co se v programu dělo. Může obsahovat i *stack trace*[3]

Zde je pak příklad jak reálně log vypadá:

12-10 13:02:50.071 1901-4229/com.google.android.gms V/AuthZen: Handling delegate intent.

To vše je samozřejmě jen příklad aplikačních logů. Konkrétně na Androidu můžeme ještě vidět logy z garbage collection[4]. Ty mají trochu jiný formát, protože jsou opravdu jen informativní pro programátora.
Zde je pro ukázku formát GC logu:

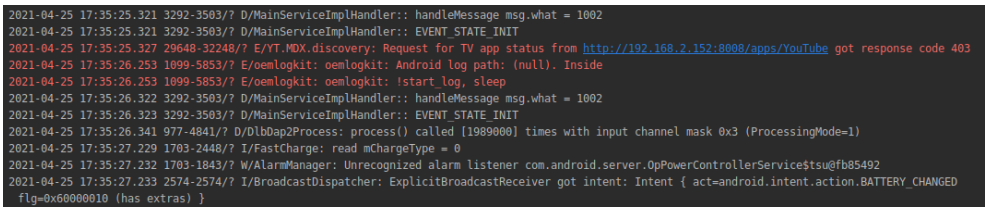
I/art: GC_Reason GC_Name Objects_freed(Size_freed) AllocSpace Objects, Large_objects_freed(Large_object_size_freed) Heap_stats LOS objects, Pause_time(s)

A tady je pak příklad skutečného GC logu:

[3]Stacktrace je zpráva, která obsahuje pořadí volaných funkcí.[6]
[4]Garbage collection (zkráceně GC) je proces, který automaticky uvolňuje nepoužívanou paměť za běhu programu.[7]

I/art : Explicit concurrent mark sweep GC freed 104710(7MB) AllocSpace objects, 21(416KB) LOS objects, 33% free, 25MB/38MB, paused 1.230ms total 67.216ms

Zde je pak ukázka toho, jak vypadají výpisy z mobilního zařízení s operačním systémem Android.



Obrázek 2.1: Ukázka Android logů

2.2 Umístění logů

Každá aplikace či program, který generuje nějaké logy, musí mít řešené jejich zpracování. Způsobů je více. Můžeme mít 3 způsoby logování:

- lokální
  - do terminálu
  - do souboru/databáze
- vzdálené (na server)

Tyto možnosti můžeme různě kombinovat. Většinou chceme viděl logy při vývoji v terminálu. Pokud ale budeme mít ku příkladu produkční aplikaci pro mobilní zařízení, tak budeme chtít logy uchovávat pro případ chyby, abychom pak snadno mohli dohledat její příčinu.

2.2.1 Lokální logování

Lokální logováním jsou myšleny možnosti výpisu do terminálu a ukládání logů na paměťové úložiště. Jak jsem již zmínil, výpis na terminál je celkem běžný.

2.2.2 Centralizované logování

Centralizované logování je v dnešní době doporučované a je i velmi užitečné. Oproti lokálnímu logování nám to přináší řadu výhod. Můžeme díky tomu vidět logy ze všech zařízení, máme je snadno k dispozici, často pak máme i možnost je filtrovat a podobně. Pokud používáme nějaký sofistikovanější

systém, tak můžeme využít například notifikační systém, který nás upozorní na nežádoucí incidenty v aplikacích a vývojáři je mohou hned začít řešit.

Pokud zavolá zákazník a chce nahlásit chybu, tak tu máme velmi vysokou pravděpodobnost, že chybu budeme schopní najít již v průběhu hovoru. Při nahlášení chyby zní lépe řekneme-li, že víme na jakém řádku se stala chyba nebo že koukáme na to a problém je na řádku XY. Toto bychom nemohli říct, pokud bychom u sebe logy neměli.[8]

Nevýhodou tohoto přístupu však je nutnost internetu. Obzvlášť při neopatrném použití můžeme posílat velké množství dat, což je kontraproduktivní. Zatěžuje to jak zařízení na kterém aplikace běží, tak i server. Pokud bychom měli velké množství takových zařízení a každé by produkovalo velké množství dat, tak server bude brzy zahlcen a bude to k ničemu. Proto chceme posílat na server jen důležité informace, které nám jsou opravdu užitečné.

V případě nedostupnosti k internetu je to problematické. Pokud bychom internet neměli vůbec, pak nemáme možnost jak logy dostat na server. Pokud bychom však internet neměli jen dočasně, například krátkodobý výpadek, pak můžeme logy dočasně uložit na lokální úložiště a použít pak smíšené logování.[9]

2.2.3 Smíšené logování

Jak jsem již zmiňoval dříve, můžeme tyto přístupy kombinovat. Lokální logování probíhá téměř vždy, obzvlášť do terminálu, ale k němu pak ještě přidáváme posílání logů na server.

Na mobilních zařízeních je celkem běžné, že se logy na centralizované úložiště neposílají. Takovou malou náhradou toho je posílání crashlogů, tedy záznamů o chybách. Měří se tím pak chybovost aplikací a jejich celková stabilita.

Nevýhodou smíšeného logování pak může být větší zátěž systému. Pokud posíláme nějaké logy do terminálu, ukládáme je do souboru a ještě k tomu je posíláme na server, tak samozřejmě zatěžujeme výpočetní jednotku. Nicméně tento dopad pak můžeme minimalizovat množstvím logů. Například na server nemusíme posílat všechny logy, ale jen ty důležité. Obdobně to pak můžeme nastavit i pro lokální logování a v případě že aplikace je produkční, tak můžeme omezit popřípadě vypnout i výpis do terminálu.

Kapitola 3

Sběr požadavků

V této kapitole si rozebereme požadavky na logovací server a logovací knihovnu. Sebrané požadavky vychází z komunikace se zástupcem firmy, pro kterou jsou tento server a knihovna vytvářeny. Při této komunikaci jsme sepsali, co budou mít logovací server a knihovna za funkcionalitu a jaké budou mít vlastnosti.

Cílem každého projektu je doručit kvalitní produkt, který si zákazník objednal. Nejednoznačné požadavky pak mohou vést k nízké kvalitě produktu či až k selhání celého projektu[10].

Proto tedy chceme vytvořit sadu požadavků, které vymezí rozsah projektu aby obsahoval všechnu funkcionalitu a zároveň nestál zákazníka příliš mnoho peněz.
Požadavky se dělí na tyto dva typy:

- funkční
- nefunkční

Je důležité tyto typy požadavků rozlišovat a důkladně je specifikovat. Jeden typ požadavků se zaměřuje na funkcionalitu, co bude produkt umět, a druhý pak specifikuje kolik toho pak zvládne či jak rychle.

3.1 Funkční a nefunkční požadavky

V celé této sekci budu citovat ze stránky reqtest.com[10]. Popíši zde jak mají vypadat funkční a nefunkční požadavky a rozdíl mezi nimi.

3.1.1 Funkční požadavky

Funkční požadavky nám tedy specifikují, co produkt bude dělat. Jinými slovy nám popisují funkcionalitu či co bude chování produktu při nějaké akci.

Obrázek 3.1: Funkční požadavek[10]

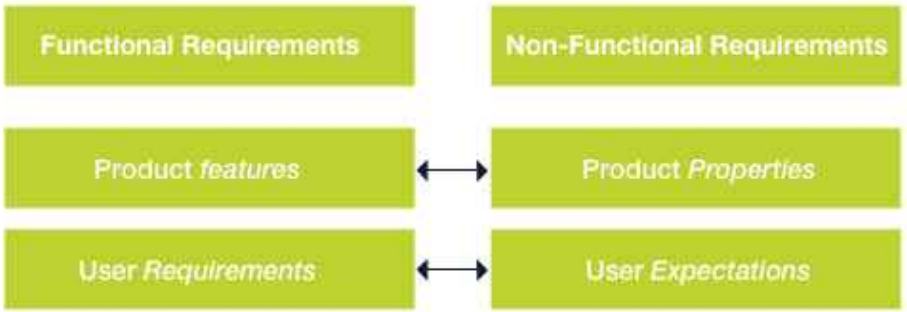
Jednoduchým příkladem takového chování může být: rychlovarná konvice musí být schopna udržet tekutinu a neprotékat.

3.1.2 Nefunkční požadavky

Nefunkční požadavky nám popisují, jak se bude produkt chovat či co zvládne. Běžně se pak stanovuje nějaký limit, který musí být produkt schopný dodržet. Sada nefunkčních požadavků pak doplňuje funkční požadavky, abychom pokryli i případy, které funkční požadavky nepokrývají. Příkladem takovéto specifikace pak může být: rychlovarná konvice musí být schopna přivést litr vody do varu do 3 minut.

3.1.3 Rozdíl mezi funkčními a nefunkčními požadavky

Nefunkční požadavky se zaměřují na očekávání od produktu, což je oproti funkčním požadavkům, které se zaměřují na to, co by produkt měl dělat.



Obrázek 3.2: Funkční a nefunkční požadavky[10]

3.2 Požadavky na logovací server

Sebrané požadavky jsou na prototyp logovacího serveru, takže nejde o specifikaci finálního produktu. Specifikovali jsme tedy základní funkcionalitu tak, aby produkt byl použitelný a přínosný. Vychází to z komunikace se zástupcem firmy a vedoucím této práce[11].

Z diskuze vyplynulo, že firma chce centralizovaný logovací server, kam bude moci posílat logy z různých aplikací, které poběží na mobilních zařízeních. Firma vyvíjí aplikace jak pro platformu Android, tak i pro platformu iOS.

3.2.1 Funkční požadavky na logovací server

V této sekci je popis funkčních požadavků na prototyp logovacího serveru a jejich stručný popis.

3.2.1.1 Server musí mít možnost vystavení do internetu

Server musí být dostupný z internetu po přidělení veřejné IP adresy.

3.2.1.2 Server musí mít REST API

Server musí být schopný komunikovat přes REST[5] API[6].

3.2.1.3 Server musí být schopný přijímat logy ve formátu JSON

Server musí být schopný přijímat logy ve formátu JSON[7]. Logů může být v rámci jednoho JSON souboru více.

3.2.1.4 Server musí sám mazat staré záznamy

Server musí mít nastavení pro automatické mazání starých záznamů.

3.2.1.5 Server musí být schopný přijímat logy z různých zařízení

Server musí být schopný přijímat logy z různých zařízení. Musí minimálně podporovat logy z mobilních aplikací běžících na platformě:

- Android
- iOS

Každá platforma má vlastní strukturu dat.

[5]REST je architektonický styl poskytující standard mezi počítačovými systémy na webu pro snadnou komunikaci mezi nimi.[12]

[6]API je softwarový prostředník umožňuje komunikaci mezi aplikacemi.[13]

[7]JSON jednoduchý formát dat sloužící pro jejich výměnu.[14]

3.2.1.6 Server musí být zabezpečený

Server musí být zabezpečený tak, aby tam neoprávněná osoba nemohla:

- posílat žádná data či nastavení
- stahovat žádná data či nastavení
- zobrazovat žádná data či nastavení
- měnit žádná data či nastavení

Tato funkcionalita je povolena pouze autorizovaným uživatelům.

3.2.1.7 Server musí mít skupiny uživatelů s různými oprávněními

Server musí být schopný mít různé úrovně oprávnění pro různé lidi či skupiny lidí. A to minimálně pro tyto skupiny:

- administrátoři
- vývojáři

3.2.1.8 Server musí být schopný zobrazovat přijaté logy

Server musí být schopný zobrazovat přijaté logy tak, aby si uživatel mohl prohlížet jejich obsah.

3.2.1.9 Server musí být schopný filtrovat přijaté logy

Server musí být schopný data daného formátu filtrovat podle klíčových informací. Filtrování musí být možné alespoň podle:

- časové značky (timestamp)
- zařízení ze kterého logy přišly
- hladiny závažnosti logu

3.2.1.10 Komunikace se serverem musí být zabezpečená

Server musí umět komunikovat pomocí protokolu https[8], aby data nebyla čitelná nepovolaným lidem.

[8]Hypertext Transfer Protocol Secure

3.2.1.11 Server musí být spustitelný i jako lokální instance

Server poběží jako jedna centrální instance, ale vývojář si musí být schopný spustit vlastní instanci na lokálním počítači. Tato instance nemusí splňovat všechny funkční požadavky na bezpečnost či přístupová práva, ale možnosti prohlížení logů a jejich filtrování jsou stále požadovány.

3.2.2 Nefunkční požadavky na logovací server

Zde jsou popsány nefunkční požadavky na server. Vzhledem k tomu, že jde pouze o prototyp serveru, tak jsou tyto požadavky obecněji definované.

3.2.2.1 Server musí být snadno přenositelný

Server musí být snadno spustitelný na různých platformách (macOS, Linux, Windows).

3.2.2.2 Server musí být schopen zvládnout 1000 logovacích požadavků za sekundu

Server musí být schopen dostatečně rychle zpracovávat požadavky, aby byl schopen zpracovat alespoň 1000 logovacích požadavků za sekundu.

3.2.2.3 Server musí být schopen pracovat s větším množstvím dat

Server bude přijímat data ze spousty zařízení a budou se tam skladovat po dobu minimálně 30 dní. Musí tedy být schopen v uskladněných datech rozumně rychle vyhledávat. To znamená, že 90 % vyhledávacích požadavků musí být splněno do 3 sekund.

3.2.2.4 Server musí být snadno rozšiřitelný

Server musí být snadno rozšiřitelný například o nové formáty logovaných dat.

3.2.2.5 Server musí být snadno udržitelný

Server musí být snadné udržovat a spravovat.

3.3 Požadavky na logovací knihovnu

Tato sekce popisuje funkční a nefukční požadavky na prototyp logovací knihovny.

3.3.1 Funkční požadavky na logovací knihovnu

Funkční požadavky na logovací knihovnu musí být aplikovatelné na obě platformy (Android i iOS).

3.3.1.1 Knihovna musí být schopna posílat logy zabezpečeně

Knihovna musí být schopna posílat logy na server pomocí https.

3.3.1.2 Knihovna musí umět skladovat logy

Knihovna musí uměl uchovat logy pro případy nedostupnosti internetu.

3.3.1.3 Knihovna musí umět posílat více logů najednou

Knihovna musí umět poslat více logů najednou, aby byla například schopná poslat logy za poslední den.

3.3.1.4 Knihovna musí umět posílat logy asynchronně

Knihovna musí posílat logy asynchronně, aby neblokovala běh aplikace.

3.3.1.5 Knihovna musí umět sama mazat staré logy

Knihovna musí umět smazat nepotřebné uchované logy.

3.3.2 Nefunkční požadavky na logovací knihovnu

Protože se jedná jen o prototyp logovací knihovny, tak jsou nefunkční požadavky obecné.

3.3.2.1 Knihovna musí být snadno rozšiřitelná

Knihovna musí být snadno rozšiřitelná pro přidávání nových funkcionalit.

3.3.2.2 Knihovna musí být jednoduchá na použití

Knihovna musí být intuitivní při používání.

Kapitola 4

Analýza řešení serverové části

V této kapitole rozeberu existující řešení pro centralizovaný sběr logů z mobilních zařízení. Projdu jejich výhody a nevýhody a na základě toho pak navrhnu řešení serverové části této práce.

V dnešní době je alternativ, které se dají na centralizované logování použít, opravdu mnoho. V této práci ani zdaleka neprojdu všechny. Zmíním zde pouze ty nejvýznamnější.

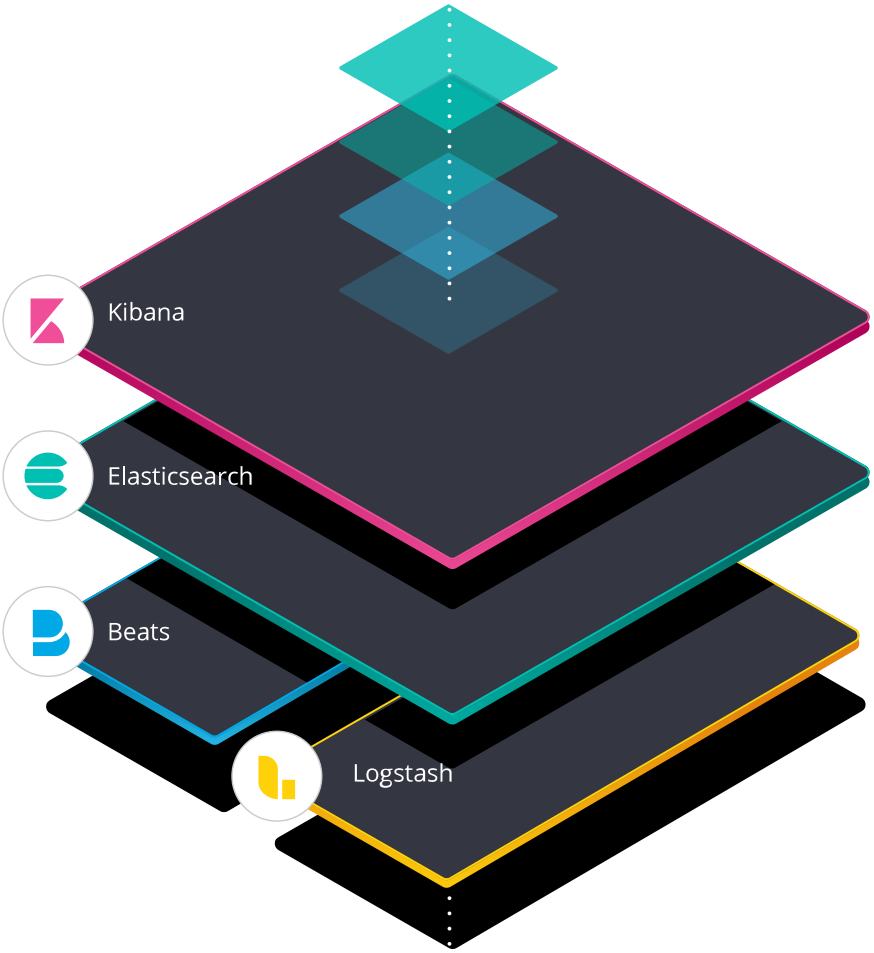
Zde je seznam doporučených vlastností, které by měl mít logovací server[15]:

- Přijímání různých formátů z různých zdrojů.
- Zpracování dat do struktur tak, abychom podle důležitých klíčů mohli vyhledávat.
- Monitorování v reálném čase s možností nastavení notifikací.
- Vizualizace dat.
- Možnost škálovat při velkém množství dat.
- Server by neměl být příliš drahý.
- Konfigurace, údržba a práce se serverem by měla být jednoduchá, aby pak nebylo například potřeba najmout celý nový tým na jeho správu.
- Ideálně bychom měli mít možnost spustit server kdekoliv chceme. (cloud, vlastní server, a podobně)

4.1 Elastic Stack

Elastic Stack nebo také ELK Stack je seskupení několika open source[9] projektů. Od názvu ELK Stack se upouští, protože toto seskupení je tak škálovatelné a rozšiřitelné, že akronym z pouze tří komponent pro to nestačí. Do tohoto seskupení totiž přibyl další projekt a měnit každou chvíli akronym by vytvářelo zmatek, tak se začal používat název Elastic Stack.[17]

Zde na obrázku 4.1 je pak vidět, jak jsou na sobě projekty postavené a proč se tomu tedy říká *stack*.



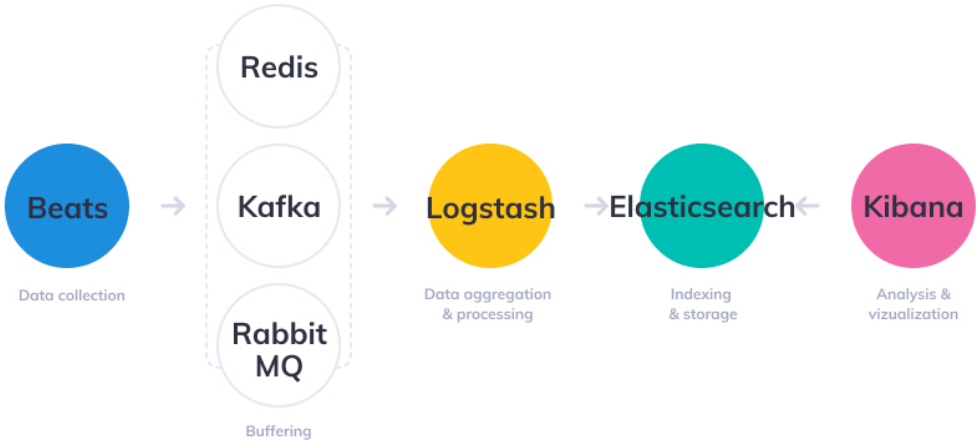
Obrázek 4.1: Elastic Stack[17]

[9]Open source znamená, že to lidé mohou modifikovat a sdílet, protože návrhy té věci jsou veřejně dostupné.[16]

Projekty seskupené pod názvem Elastic Stack jsou následující:

- Elasticsearch
- Logstash
- Kibana
- Beats

Elastic stack je komplexní řešení pro spoustu problémů. Jeho výhodou je velká škálovatelnost a rychlost, s jakou je schopen úkoly zpracovávat. Také je hodně konfigurovatelný, takže si ho každý může přizpůsobit podle svých potřeb. A to například i přidáváním dalších komponent, které nám usnadní či zajistí nějakou funkcionalitu, popřípadě nám mohou pomoci rozkládat zátěž na více uzlů.[18]



Obrázek 4.2: Průchod dat Elastic Stackem[18]

Na obrázku 4.2 je pak vidět propojení těchto projektů. U každého pak popíši jeho účel v tomto seskupení. Nicméně zde můžeme vidět i komponentu, která v původním výčtu projektů Elastic Stacku nebyla.
Alternativ, co použít v této komponentě je více, ale zde jsou pro ukázku uvedené tyto 3 často používané:

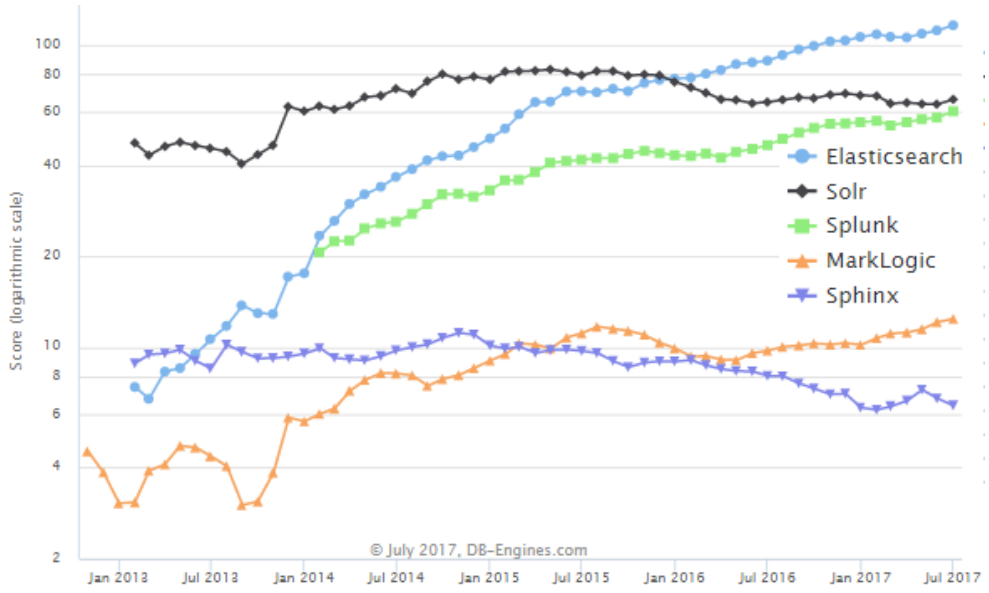
- Redis
- Kafka
- RabbitMQ

Jedná se o zprostředkovatele zpráv mezi komponentou Beats a Logstash. Tyto komponenty nám mohou pomáhat při velkém množství dat, které putují na server. Server bude pak pravděpodobně mít více instancí, aby se mohl snáze vypořádávat s velkou zátěží. Když pak server nebude schopen například dostatečně rychle odebírat zprávy, tak je budeme chtít někde pozdržet, než je bude moci nějaký serverový uzel odbavit.

Výhoda Elastic Stacku je ta, že vytváření více instancí je velmi jednoduché, což nám umožňuje se dobře vypořádávat se zátěží. Díky této škálovatelnosti je schopen Elastic Stack zpracovat velké množství dat v krátkém čase.

4.1.1 Elasticsearch

Elasticsearch je srdce celého Elastic Stacku. V dnešní době je to jeden z nejpopulárnějších vyhledávacích enginů a databázových řešení. Obzvlášť za poslední 4 roky jeho popularita značně vzrostla, což je dobře vidět v následujícím grafu na obrázku 4.3.



Obrázek 4.3: Vyhledávací enginy[19]

V kontextu celého Elastic Stacku, Elascticsearch hraje roli úložiště a indexace dat. V tomto seskupení pak tvoří Elastic Stack mocný nástroj na práci s velkými daty. Elasticsearch je braný jako NoSQL databáze[10], nicméně však

pracují na projektu, který umožní použít SQL[11] dotazy pro interakci s daty. Na rozdíl od ostatních NoSQL databází, Elasticsearch je hodně zaměřený na možnosti a způsoby vyhledávání. Asi nejjednodušším způsobem jak pak získat data z Elasticsearche je pomocí jeho rozsáhlého API.

Pro pochopení jak funguje Elasticsearch je důležité znát tyto pojmy:

- index
- dokument
- typ
- mapování
- shard[12]
- replika

Elasticseach je komplexní sytém, který je bohatý na funkcionalitu, kterou můžeme dále obohacovat o pluginy. Poznat všechny jeho detaily je víceméně nemožné. Nicméně výše zmíněných 6 pojmů je důležitých, že je alespoň trochu popíši.

4.1.1.1 Index

Indexy v Elasticsearchi jsoou logická seskupení dokumentů, které se dají přirovnat k databázi ve světě relačních databází. Index by pak byl přirovnán k tabulce.

Elasticsearch nestanovuje žádný limit na počet indexů, takže těchto indexů můžeme mít mnoho. Nicméně to však může mít negativní dopad na výkon, protože dokumenty jsou unikátní pro každý index.

4.1.1.2 Dokument

Dokumenty jsou JSON objekty, které jsou uloženy v indexu Elasticsearche. Jsou považovány za základní úložnou jednotu. Přirovnali bychom-li dokument k relačním databázím, pak bychom našli k němu paralelu v podobě řádku v tabulce.

[10] NoSQL databáze neuchovávají v tabulkách jako klasické relační SQL databáze. Přichází s různými typy uložení v závislosti na jejich datovém model. Známými typy NoSQL databází jsou dokumentové nebo třeba grafové databáze.[20]

[11] SQL je dotazovací jazyk, který se používá na dotazování do databáze.[21]

[12] Shard znamená střep/úlomek, ale v tomto kontextu se překlad nehodí.

Data v dokumentu jsou definovány položkami, které se sestávají z klíče a hodnoty. Klíčem je vždy název položky a hodnotou pak je prvek, který může být mnoha typů:

- řetězec znaků
- číslo
- booleovksý výraz
- nějaký objekt
- pole nějakých prvků

Takovýto dokument si pak mimo jiné drží i nějaká metadata, což jsou například index a id.

4.1.1.3 Typ

Typy v Elasticsearchi se používají v rámci dokumentu k dalšímu dělení obdobných typů dat, kde každý typ dat reprezentuje unikátní třídu dokumentů.

Typy se skládají ze jména a mapování a jsou použity přidáním položky *_type*. Díky přidání této položky pak můžeme při dotazování filtrovat i podle typu.

Tato funkcionalita v podobě typů se však z Elasticsearche vytrácí. Ve verzi 7.x už je řečené, že by se neměla používat, ve verzi 8.x už není ani podporována.

4.1.1.4 Mapování

Jako schéma ve světě databází, mapování definuje různé typy v rámci indexu. Definuje to oblasti pro dokument specifického typu, což může být kupříkladu řetězec znaků nebo číslo, a jak by měl být oblast indexována a uložena v Elasticsearchi.

Mapování může být definováno explicitně či generováno automaticky, když je soubor indexován pomocí šablony. Tyto šablony zahrnují nastavení a mapování, které může být aplikováno automaticky na nový index.

4.1.1.5 Shard

Jak jsem již zmiňoval dříve, tak u indexů můžeme mít celkem snadno problém, nedáme-li si pozor. Indexy jsou častou příčinou problému, protože jejich počet není omezený. To pak může zabrat velké množství prostoru na disku serveru. V případě že nám místo na serverovém disku dojde, tak se indexování začne rozpadat.

Jedna z možností jak předcházet tomuto problému je, že budeme rozdělovat indexy do částí, které nazýváme shardy. Také se tomuto způsobu říká horizontální škálování, což znamená přidávání více strojů, klidně i virtuálních, vedle sebe[22].

Horizontální škálování tam tedy umožní rozdělit operace mezi více shardů a uzlů, takže tím zvýšíme výkon. Můžeme si pak i nakonfigurovat počet shardů na jeden index. Tyto indexové shardy následně můžeme hostovat na jakémkoliv uzlu našeho Elasticsearch clusteru.

4.1.1.6 Replika

Aby byl server schopný se snadno zotavit ze selhání systému jako například pád serveru či výpadek internetového připojení, tak Elasticsearch umožňuje uživateli vytvářet kopie shardů, které se pak nazývají repliky. Tyto repliky byly navržené pro zajištění vysoké dostupnosti, takže nejsou umisťovány na stejný uzel, ze kterého byly zkopírovány. Stejně jako shardy můžeme počet replik definovat už při vytváření indexu, ale také je můžeme dodefinovat v pozdější fázi.

4.1.2 Logstash

Efektivní analýza logů je možná jen v případě, že logy máme dobře strukturované. Díky dobré struktuře můžeme v logách dobře vyhledávat a snadno je vizualizovat. Toto je převážně ovlivněné na straně aplikace, která logy generuje.

Nicméně Logstash může tyto data nějak zpracovávat. Může tyto data obohacovat, může třeba měnit i jejich strukturu, čistit je, opravovat je a podobně. Nakonec po jejich zpracování je pak pošle dál.
Tudíž konfigurujeme tyto tři fáze:

- sbírání/přijímání dat
- zpracování dat
- odeslání dat

Tím, že je Logstash open source projekt, tak má mnoho pluginů. Těmito pluginy pak můžeme upravovat chování Logstashe pro naše potřeby.

Logstash má nevýhodu v tom, že běží na JVM[13]. Nevýhodou to může být kvůli velké spotřebě RAM paměti. To se děje především v případech, kdy provádíme komplikovanější operace nad přijatými daty.

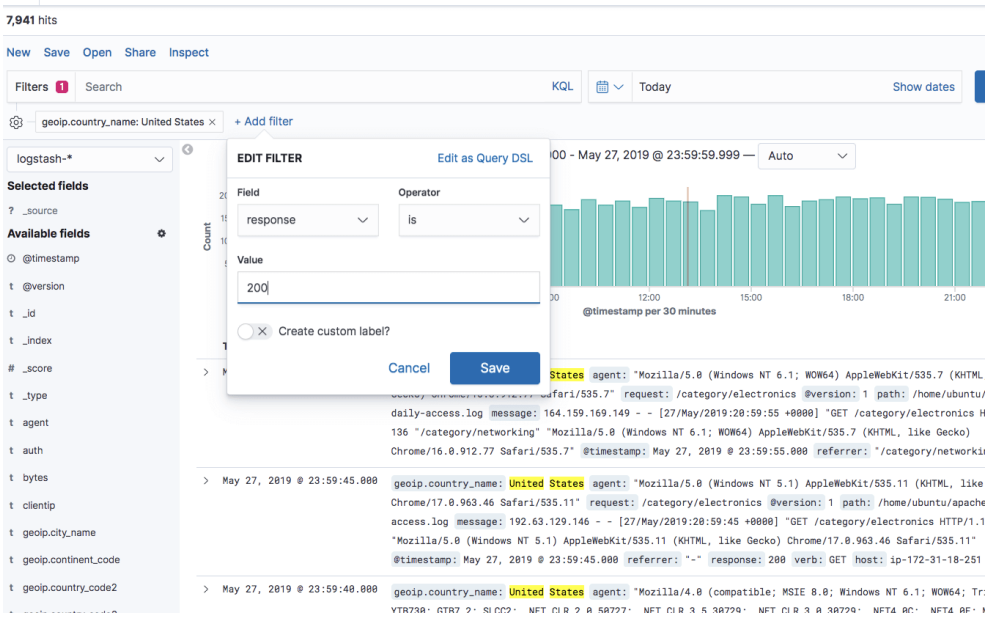
Dalším problémem může být používání nepotřebných pluginů či špatnou konfigurací. Může pak docházet i ke ztrátě či poškození dat a pokud nemáme data nějak chráněná, například zálohou, tak to může být velký problém.

[13]JVM je virtuální stroj jazyku Java, který umožňuje běh Java aplikací na systému.[23]

4.1.3 Kibana

Každý pořádný logovací server potřebuje mít možnost vizualizovat data, případně je prohlížet. V Elastic Stacku tuto funkci zastává Kibana.

Kibana je uživatelské rozhraní přístupné přes webový prohlížeč. Je známá a populární pro své bohaté možnosti vizualizace.



Obrázek 4.4: Ukázka Kibany[18]

Běžně se musíme probírat hromadou logů a při velkých množstvích to může být zdlouhavé a nepříjemné. Kibana umí pracovat s velkým objemem dat, což je často velmi důležité. Také v nich umí rychle vyhledávat. Kibana má dokonce vlastní dotazovací jazyk KQL[14].
Zde je příklad nejběžnějších vyhledávání podle:

- specifického řetězce znaků
- specifického řetězce znaků v konkrétním poli
- kombinace vyhledávání spojených do logických výrazů
- vyhledávání výrazu podle podobnosti znaků

[14]KQL je dotazovací jazyk Kibany.[24]

Kibana rovněž umí ukládat vyhledávání, vizualizace a dashboardy, které pak nazývá objekty. Tyto objekty jsou pak uloženy ve speciálním indexu Elasticsearche. Tento index je vytvořen hned jak se Kibana spustí. Jeho účelem je:

- odladění chyb
- sdílení
- opětovné použití
- zálohování

Jedna z dalších velkých výhod Kibany je systém upozornění. Můžeme si nastavit upozornění při nějaké zvýšené zátěži na systému či při nějaké chybě. Tyto upozornění pak můžou být automaticky poslány vývojářum emailem, na Slack a podobně.

4.1.4 Beats

Beats je rodina programů, která se stará o přesouvání dat. Tyto programy jsou napsané v programovacím jazyce Go a byly navrženy tak, aby nezatěžovaly systém. Na serveru pak běží jako agenti.[18]

Na základě typu dat pak existují různé typy:

- Filebeat - přesouvání logů a jiných dat
- Metricbeat - přesouvání naměřených dat
- Packetbeat - přesouvání síťových dat
- Winlogbeat - přesouvání logů Windows událostí
- Auditbeat - přesouvání revizních dat a dat z procesů
- Heartbeat - přesouvání dat pro uptime monitoring
- Functionbeat - funkce na sbírání a přesouvání dat do Elastic Stacku

Každý Beat tedy pracuje s jiným typem dat. Nicméně každý pak může být nakonfigurován, aby udělal s nasbíranými daty něco jiného. A i když je každý Beat trochu jiný, tak je ale zajištěno, aby měli všechny stejný způsob konfigurace. Tím se usnadňuje uživatelům spousta práce při jejich nastavování.

Některé Beaty pak nabízí možnost nějakého zpracování dat, aby se tolik nezatěžoval Logstash. Data tedy pak mohou putovat tam, případně pokud nepotřebují nějaké další zpracování, tak je můžeme poslat přímo do Elasticsearche.

4.1.5 Shrnutí

Elastic Stack je skvělý projekt, na kterém staví spousta jiných monitorovacích a logovacích platforem. Pro jeho skvělé vlastnosti jeho popularita stále roste, i když tam jsou stále místa k vylepšení.

Modularita tohoto projektu je opravdu velká. Můžeme zde přidávat různé komponenty dle potřeby, případně některé stávající i vyměnit. Pokud by nám třeba z nějakého důvodu nevyhovoval Logstash, můžeme místo něj nasadit například Fluentd.

4.1.5.1 Výhody

Velkou výhodou Elastic Stacku je to, že je to open source projekt. Tím dává možnost komukoliv si projekt upravit dle jeho potřeb. Další výhodou je pak rychlost, se kterou dokáže Elastic Stack zpracovávat obrovská množství dat. V případě potřeby pak lze snadno horizontálně i vertikálně škálovat.

Další výhodou je možnost zpracování všech možných strukturovaných i nestrukturovaných dat. Elastic Search si můžeme snadno nastavit tak, že bude centrálním bodem pro sběr logů všech možných aplikací a serverů.

4.1.5.2 Nevýhody

Nevýhodou může být při větším škálování udržitelnost. Údržba se může stát náročnou, pokud budeme chtít mít hodně serverů. Další nevýhodou pak může být to, že v Elastic Stacku je komplikovaná konfigurace přístupových rolí. V neposlední řadě může být nevýhodou složitější počáteční konfigurace všech částí.

Jedna z nevýhod se týká i konkrétně Kibany. Jde o to, že i když nám umožňuje zobrazovat data a má opravdu spoustu možností a funkcionalit, tak některé její použití nemusí být úplně intuitivní a je potřeba si na to zvyknout, případně se s tím naučit pracovat.

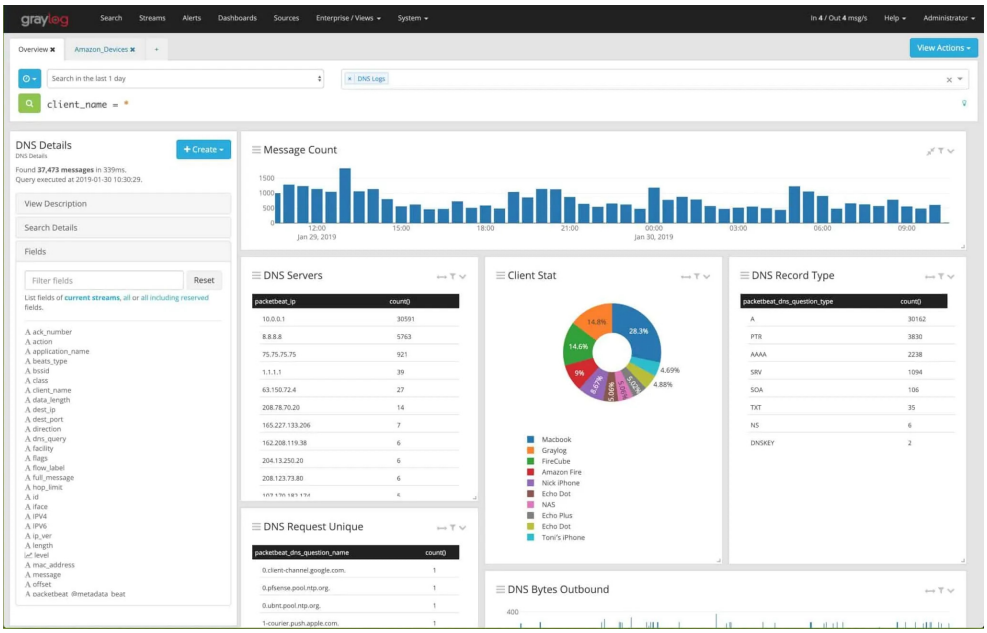
Velkou nevýhodou Elasticsearche je chyba, která nastane když přijme takováto data:

$$\{\textit{"pid": 12345}\} \text{ a } \{\textit{"pid": "12345"}\}$$

Je jedno v jakém pořadí přijdou a co bude klíčem. Problém v tomto příkladu je, že Elasticsearch rozpozná tuto položku v prvním případě jako číslo a v druhém případě jako řetězec. V tu chvíli máme konfliktní položky, se kterými si Elasticsearch neví rady a začne další logy zahazovat.

4.2 Graylog

Stejně jako Elastic Stack, tak i Graylog je open source řešení centralizovaného logování a monitorování. Graylog je schopný pracovat s obrovkým množstvím logů či podobných dat a to díky tomu, že je postavený na Elasticsearchi. Používá Elasticsearch jako databázi a vyhledávací stroj.[25]



Obrázek 4.5: Ukázka Graylogu[26]

Protože primární účel Graylogu je správa logů, tak má spoustu již zabudovaných funkcionalit zaměřených právě na tento účel. Jednou z nich může být blokování určitých zpráv, případně jejich úprava, což může být například přidávání nebo odebírání položek.

Pokud použijeme Graylog, tak si můžeme vybrat ze dvou verzí. Buď použijeme open source verzi, která je úplně zadarmo a je bohatá na funkcionality. Pokud bychom však chtěli nějaké další funkcionality, tak můžeme použít Graylog ve verzi Enterprise.

Ten například nabízí[27]:

- Correlation Engine
- Enterprise dynamické vyhledávací tabulky
- LDAP integrace skupin
- správa týmů
- pravidelné souhrnné zprávy
- vyhledávací parametry
- offline archivace logů

Funkcionalit ve verzi Graylogu Enterprise oproti open source Graylogu je samozřejmě více než zde pár uvedených. Enterprise verze má však omezení na 5 GB přijatých dat za den. Pokud bychom chtěli přijímat na serveru více dat, pak už by bylo potřeba zakoupit licenci.[28]

Zajímavou funkcionalitou je zde Graylog Sidecar, která umožňuje centralizovat konfigurační šablony pro jednotlivé agenty. Můžeme si nastavit třeba jak filtrovat data z nějakého typu zdroje, a Graylog Sidecar nám zaručí, že na každém serveru se nám data z nakonfigurovaného typu zdroje budou filtrovat stejným způsobem.[29]

Graylog má také chytrý způsob jak urychlit konfiguraci serveru. Nazývá se to Content Packs (obsahové balíčky). Je to způsob, jakým si lze do Graylogu instalovat balíčky funkcionalit, které jsou vytvořené komunitou.[30]

Další neméně významnou funkcionalitou Graylogu je Correlation Engine. Tento mechanismus automaticky extrahuje klíčové informace z nezpracovaných dat. Můžeme k tomu nastavit i podmínky, které spustí specifická upozornění či případně pustí skript, který opraví nastalý problém. Škálovatelnost tohoto řešení pak je již čistě v našich rukou.[26]

V Graylogu máme rovněž možnost ukládat často pouštěné vyhledávací dotazy a sdílet je, abychom si ušetřili práci. Můžeme pak také souhrnná data analyzovat a hledat v nich anomálie.

4.2.1 Výhody

Výhodou Graylogu je jeho rychlost, protože používá jako databázi na logy Elasticsearch. Také je tu velká kontrola nad uživatelskými právy a zabezpečením. Například je tu možné použít LDAP. Dále je tu užitečný systém upozornění a posílání zpráv v případě, že dojde k nějaké neočekávané necho chybové události.

4.2.2 Nevýhody

Nevýhodou může být pro někoho lehce složitější nastavování dashboardů a možnosti těchto dashboardů se mohou zdát trochu omezené. Dalším omezením v Graylogu (a rovněž i Elastic Stacku) je to, že zde chybí operace *join* nebo transformace.

4.3 LogDNA

LogDNA bylo vytvořeno aby řešilo mnoho klíčových problémů, které se vyskytují v jiných centralizovaných správách logů. Je to vysoce škálovatelný log management systém, který indexuje, agreguje a analyzuje data, a který je vhodný jak pro malé, tak i pro velké podniky.[31]

Obrázek 4.6: Ukázka LogDNA[32]

Toto řešení je velmi jednoduché na použití už od začátku. Není zde třeba se učit žádný nový dotazovací jazyk. Stačí napsat co chceme a systém nápověd už nám nabídne možnosti, které chceme.

Také uživatelské rozhraní je jednoduché a přehledné. Můžeme zde snadno vytvářet dashboardy, které pak lze sdílet s ostatními. Dále si můžeme vytvářet

systémy notifikací, třeba i na základě vyhledávání, které nás upozorní při nějaké
události.[15]

4.3.1 Výhody

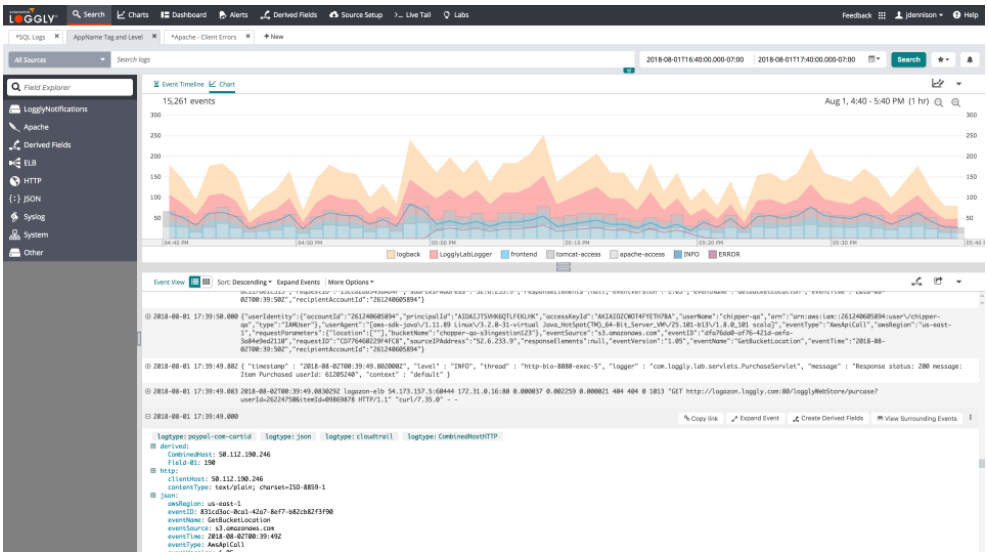
Velkou výhodou LogDNA je model nasazení a to proto, že nám nabízí spoustu možností. Můžeme mít LogDNA v cloudu, privátním cloudu, hybridním/multi-cloudu nebo na vlastním serveru.

4.3.2 Nevýhody

Nevýhodou LogDNA může být nedostatek možností si obarvovat hladiny závažnosti logů podle sebe. Dále se také občas vyskytuje problém, že se zpracovávání logů výrazně zpomalí na nějakou dobu.

4.4 Loggly

Loggly je cloudové řešení pro správu logů, které je jednoduché na použití. Spousta typů logů je zde automaticky zpracuje bez potřeby nějakých nastavení, ale pokud si potřebujeme nějaký nepodporovaný formát definovat sami, tak nám to samozřejmě Loggly umožňuje.



Obrázek 4.7: Ukázka Loggly[33]

K jeho jednoduchému používání přispívá i to, že můžeme snadno nastavit parametry hledání bez psaní nějakého složitého dotazu. Z výsledků vyhledávání pak můžeme vytvořit upozornění pro případ, že by se například hledaný incident opakoval.[15]

Loggly je systém, který nepoužívá agenty, takže data můžeme posílat přímo na server bez nějakých překážek či používání knihoven třetích stran.

4.4.1 Výhody

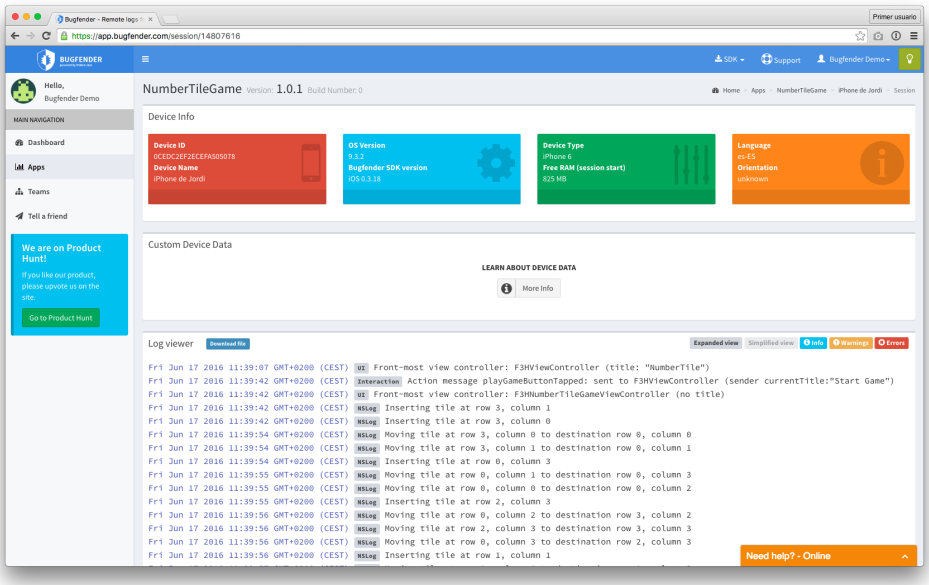
Výhodou Loggly je již dříve zmiňované jednoduché použití pro vývojáře. Také tu lze jednoduše vytvářet nové dashboardy, což se může hodit pro nějaké přehledy.

4.4.2 Nevýhody

Nevýhodou tohoto řešení pak může být jeho nevhodnost pro velké projekty. Mimo jiné také Loggly není tolik zaměřené na řešení analýzy, bezpečnosti či infrastruktury[34].

4.5 Bugfender

Bugfender je platforma na centralizovaný sběr logů. Tato platforma je zaměřená na mobilní aplikace a weby.[35]



Obrázek 4.8: Ukázka Bugfender[36]

Tato plaftorma je určená na podporu vývoje a údržby aplikací. Snaží se poskytovat vývojářům nástroje na detekování a opravování chyb. Mimo jiné pomáhá měřit i používání aplikace.

4.5.1 Výhody

Výhodou Bugfenderu je, že vytváří i knihovny pro logování, což vývojářům může velmi usnadnit práci. Také tu máme správu zařízení, kde si můžeme vybírat, které chceme sledovat.

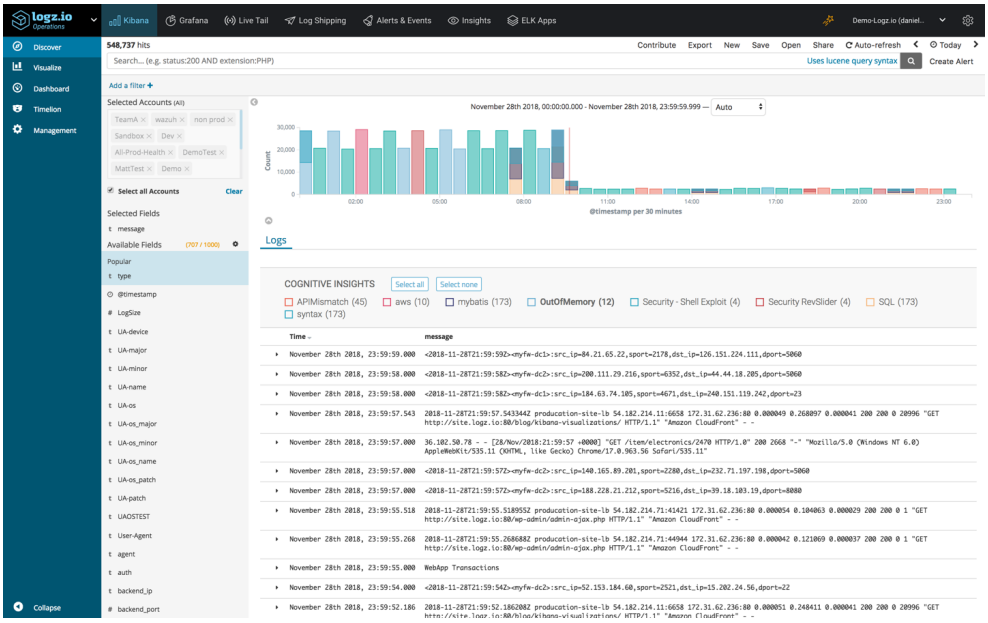
4.5.2 Nevýhody

Nevýhodou je, že filtrování podle zadaného klíče je dost omezené. Také se zde stává, že se logy občas na serveru neobjeví hned, ale až po nějakém čase.

Další nevýhodou může být omezení typů logů, protože platforma je specializovaná jen na konkrétní platformy[35].

4.6 Logz.io

Logz.io je platforma, která poskytuje sjednocený přístup k monitorování, řešení problémů a zabezpečování aplikací.



Obrázek 4.9: Ukázka Logz.io[37]

Celá tato platforma je postavená na Elastic Stacku a Grafaně[15]. Díky tomu se jedná o moderní nástroj na analýzu dat s vysokou škálovatelností, dostupností a který je bezpečný.[37]

Na začátku není třeba si vše začít nastavovat úplně od nuly, protože Logz.io nabízí spoustu předpřipravených monitorovacích dashboardů a podobně. Také je tu už v základu podpora běžných zdrojů dat jako Docker, Azure, Kubernetes a podobně.

K nalezení tu je i systém notifikací při nějakém incidentu či nějakém podezřelém chování monitorovaných aplikací. Notifikace nám může být zaslána různým způsobem. Může nám přijít email, zpráva na Slacku a podobně.

Při hledání incidentů je použito mimo jiné i strojové učení společně s crowdsourcingem, což pomáhá lépe identifikovat problémová data. Logz.io se tu také snaží rozpoznávat důležitá data od méně důležitých, a ty důležité pak uchovává déle.

Hodně zajímavou funkcí pro vývojáře může být porovnávání příchozích logů s fóry jako je StackOverflow nebo Github pro identifikaci kritických logů. Logz.io také porovnává výskyt chyb a výjimek v aplikacích v kontextu nasazování jejich nových verzí[39].

4.6.1 Výhody

Výhodou je zde to, že se jedná o plně hostovaný Elastic Stack, u kterého se nemusíme starat o škálování. Dále tu najdeme již předpřipravené dashboardy nebo intuitivní uživatelské rozhraní pro nastavování nových pravidel na zpracování logů či dat.

4.6.2 Nevýhody

Jednou z nevýhod Logz.io je vytváření podúčtů. Dále tu také chybí možnost nastavení všech filtrů v grafickém rozhraní před vykonáním dotazu. I když tu máme funkcionalitu seskupování výsledků podle nějakého pravidla, tak se používání této funkcionality může zdát neintuitivní.

[15]Grafana je open source projekt na dotazování, vizualizaci a notifikace na základě sledování hodnot či logů, bez ohledu na způsob uložení dat.[38]

4.7 Sumo Logic

Sumo Logic je cloudové řešení, které je schopné se samo škálovat podle potřeby. Tato platforma zvládá zpracovat a analyzovat obrovská množství přijatých dat.[15]

Na této platformě můžeme sbírat i data z různých systémů a sledovat jejich stabilitu, zátěž a podobně. Zajímavostí pak může být to, že analýza dat je zde podpořena strojovým učením[40].

Obrázek 4.10: Ukázka Sumo Logic[41]

Sumo Logic můžeme obohacovat o různé funkcionality, které u této platformy nazýváme aplikace.

4.7.1 Výhody

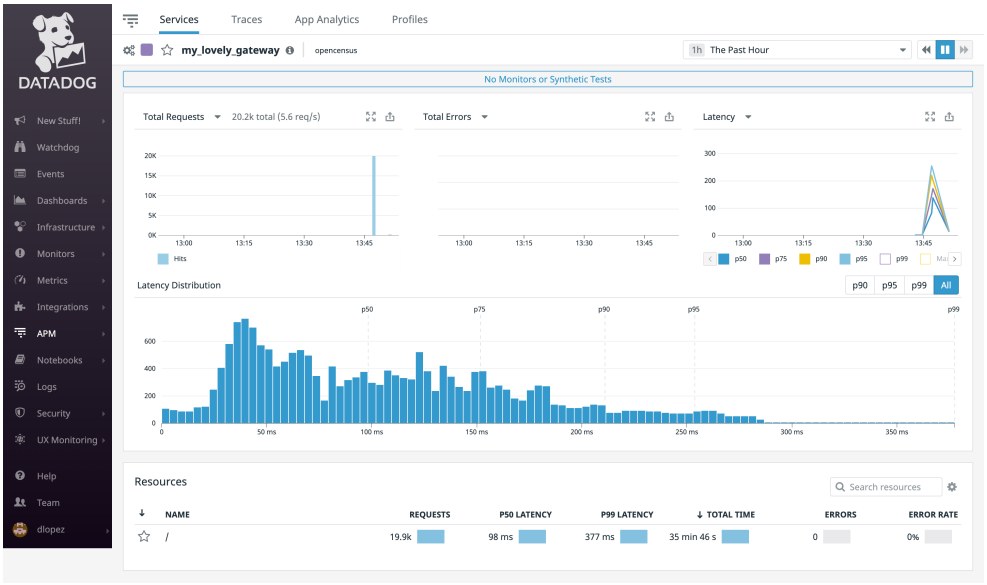
Výhodou Sumo Logic je jeho jednoduchost používání, ať jde o jednoduché a intuitivní uživatelské rozhraní a nebo jednoduché vyhledávání záznamech.

4.7.2 Nevýhody

Nevýhodou této platformy je bohužel nedostatek komunitní podpory. Další nevýhodou pak je pomalejší vyhledávání ve starších datech.

4.8 Datadog

Datadog je nástroj, který slouží hlavně k efektivnímu monitorování sítě a infrastruktury. I když je Datadog zaměřen na monitorování, tak je dostatečně bohatý na funkcionalitu na to, aby mohl být efektivně použitý i v jiných směrech jako například log management. Můžeme ho pak ještě obohatit o agenty, které si dle potřeby navolíme.[42]



Obrázek 4.11: Ukázka Datadogu[43]

Další funkcionalitou, kterou Datadog oplývá, je systém upozornění. Upozornění můžeme nastavovat i na případné vyskytlé anomálie v datech či monitoringu, na což tu Datadog používá strojové učení.

4.8.1 Výhody

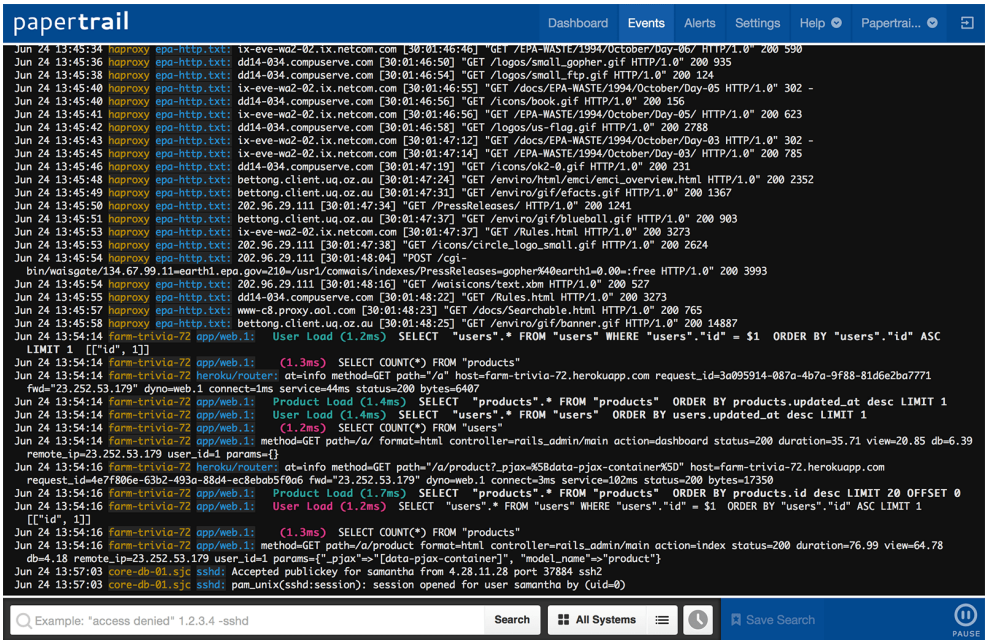
Výhodou Datadogu je možnost specifického nastavení pro každou aplikaci. Také zde můžeme snadno vyhledávat, v čem nás podpoří velmi chytré našeptávání.

4.8.2 Nevýhody

Naopak nevýhodou nástroje Datadog pak je, že prvotní nastavení může být trochu zdlouhavé, protože je potřeba nastavit všechny agenty. Dalším nedostatkem pro někoho může být, že zde není žádný standardní systém reportů.

4.9 Papertrail

Papertrail je jednoduchý nástroj na centrální správu logů. Tento nástroj je zaměřený na základní analýzu dat. Je vhodný na prohlížení sesbíraných logů a na nic moc víc již není. Díky tomu má jednoduché rozhraní a též je i jednoduchý na použití.[44]



Obrázek 4.12: Ukázka Papertrail[34]

I když je tento nástroj hodně jednoduchý, tak tu je možné filtrovat a vyhledávat v sesbíraných datech. Vyhledávací dotazy si můžeme ukládat a znovu použít později. Můžeme si k nim pak vytvářet i pravidla, která nás informují v případě potřeby, například pokud přijde na server log s chybovou hláškou.[34]

4.9.1 Výhody

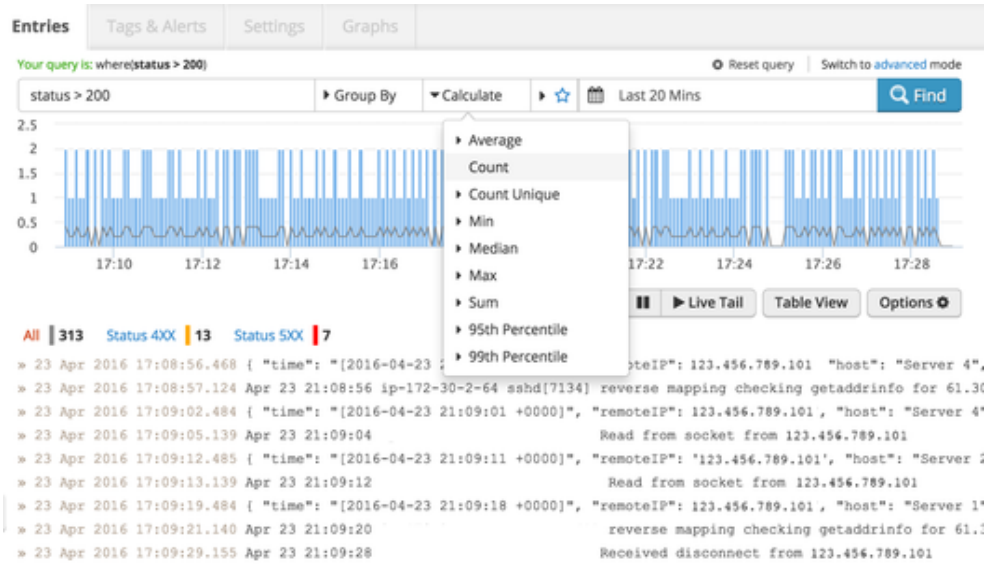
Výhodou tohoto nástroje je jeho finanční dostupnost. Pokud nepotřebujeme hromadu funkcionalit, ale jen přístup k logům odkudkoliv, pak je to výborné řešení.

4.9.2 Nevýhody

Nevýhodou Papertrailu může být pro dost firem jeho jednoduchost. Není to nástroj plný všemožných funkcí.

4.10 Logentries

Jedna z dalších možností na trhu je Lognetries, která má spoustu výhod. Je jednoduché Logentries používat a nemusíme se moc zabývat s počátečním nastavováním. Máme zde spoustu předpřipravených šablon, takže se nemusíme téměř o nic starat.[45]



Obrázek 4.13: Ukázka Logentries[46]

Lze zde také nastavit spoustu užitečných věcí jako jsou upozornění na události. Například při nějakém problému může vývojářům přijít zpráva emailem nebo třeba SMS zprávou.

4.10.1 Výhody

Jednou z výhod oproti konkurenci je například zobrazování časových značek v lokální časové zóně bez nutnosti nastavování. Toto nastavení tam je už v základu.

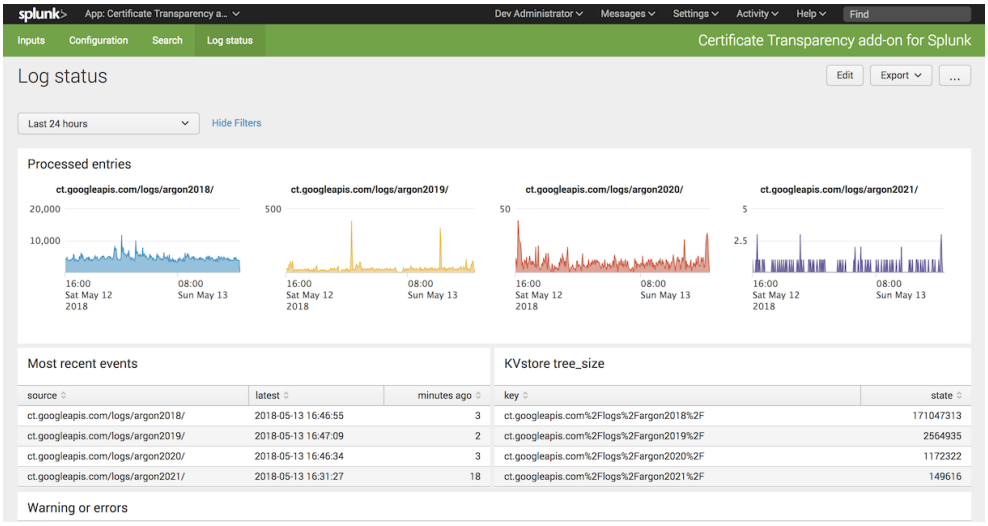
4.10.2 Nevýhody

Co se týče nevýhod, tak jedna z nich je omezenější dotazování, na což se používá LEQL[16]. Chybí tu například syntaxe na zjištění, jestli daný prvek chybí. Případně pak také můžeme narazit na limit při seskupování výsledků, který je omezen počtem unikátních řetězců.

[16]Logentries Query Language, který je založený na SQL[47].

4.11 Splunk

Platforma Splunk je univerzální řešení obzvlášť v oblasti velkých objemů dat. Tyto data se pak dají prohlížet ve formě různých vizualizací.[48]



Obrázek 4.14: Ukázka Splunku[49]

Základem technologie Splunk je zpracování strojových dat. Jeho velkou výhodou je, že umí indexovat jakýkoliv typ textových dat, aniž by musel znát jejich strukturu. Na tato data se pak můžeme dotazovat pomocí SPL[17]. Co je u Splunku výhodou oproti konkurenci je operace *join*[18] a transformace.

Splunk používá otevřené api, takže lze rovněž modifikovat pomocí různých aplikací či pluginů. Nicméně již v základu obsahuje spoustu užitečných nástrojů.

4.11.1 Výhody

Výhodou Splunku je jeho jednoduché použití strojového učení na přicházející data bez toho, aniž bychom museli něco programovat nebo něco složitě nastavovat. To nám pak může pomoct detekovat chyby a anomálie.

Další výhodou zde může být, že spousta nástrojů je zde již připravených, takže nemusíme doinstalovávat a nastavovat spousty dalších funkcionalit.

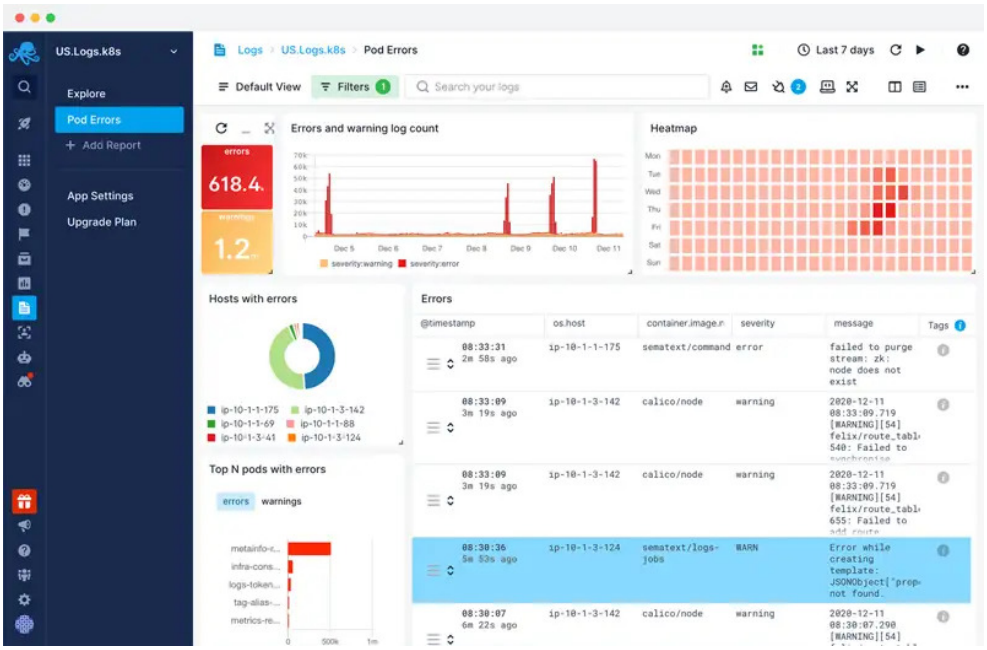
4.11.2 Nevýhody

Co se týče nevýhod, tak Splunk je známý tím, že je výrazně dražší. Navíc není lepší než konkurence. Při vyhledávání ve velkém objemu dat může být i výrazně pomalejší[48]. Další nevýhodou pak je i složitá počáteční konfigurace[34].

[17] Search Processing Language
[18] Join spojuje různé výsledky do jednoho.

4.12 Sematext Logs

Sematext Logs je jedna z dalších monitorovacích a logovacích platforem, která je postavená na Elastic Stacku. Nabízí použití Elastic Stacku bez nutnosti složitých konfigurací, které jsou jinak potřeba.[50]



Obrázek 4.15: Ukázka Sematext Logs[50]

Při posílání logů na server může přijít vhod automatické rozpoznávání položek a jejich typů v přijatých datech nebo chytré mapování. Data pak lze prohlížet, vyhledávat v nich, aplikovat na ně filtry, řadit je a podobně.

V případě potřeby si zde také můžeme nastavit upozornění, která nám pošlou zprávu emailem či jiným způsobem, abychom se dozvěděli o nastalé problematické situaci.

4.12.1 Výhody

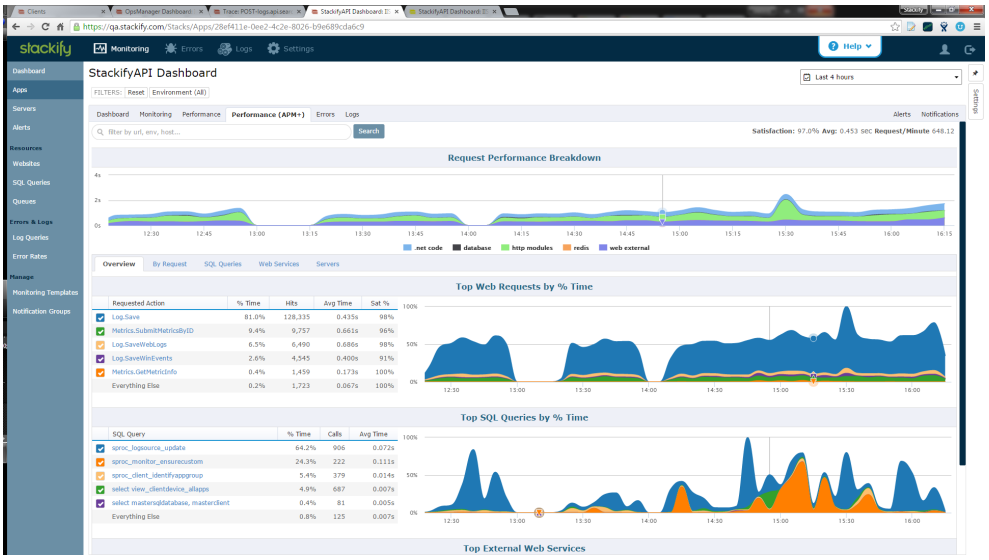
Pro některé může mít Sematext Logs výhodu v tom, že i v cloudovém řešení poskytuje přímý přístup k Elasticsearchi. Další výhodou je velice snadná analýza dat a nastavování upozornění.[51] V neposlední řadě tu je také výhoda, že při placení se neprodraží výkyvy, kdy v jednu chvíli se vytvoří velký nápor na server. Tyto situace se průměrují v rámci celého měsíce.

4.12.2 Nevýhody

Nevýhodou tohoto řešení je vytváření nových účtů. Vytváření nových účtů a přihlašování je tu velice neintuitivní. Také se občas stane, že data se na serveru neobjeví hned, ale není to časté chování. Další nevýhodou tu je, že při vytvoření více instancí po světě se hůře udržuje synchronizované nastavení serverů.

4.13 Stackify

Stackify je nástroj na centrální správu logů, monitorování a hledání chyb, který má vývojářům pomáhat při vytváření a údržbě jejich aplikací.[52]



Obrázek 4.16: Ukázka Stackify[52]

Na server můžeme posílat logy z řady programovacích jazyků a budou na serveru automaticky zpracovány. Pokud bychom chtěli přidat podporu pro nějaký nový, tak si můžeme pomocí REST API vytvořit knihovnu pro námi požadovaný jazyk/formát.[53]

Stackify i jako jiné platformy poskytuje možnost nastavit si upozornění v případě, že aplikace spadla nebo se chová podezřele. Lze tu mít i systém rolí, kde můžeme nastavovat přístupová práva.

4.13.1 Výhody

Výhodou této platformy je komplexní řešení, které aplikaci monitoruje, sbírá logy, chyby a pomáhá analyzovat případné problémy, o kterých se můžeme dozvědět sms zprávou hned ve chvíli, kdy se staly.

4.13.2 Nevýhody

Nevýhodou může u tohoto produktu být to, že občas potřebuje hodně lidské intervence. Také filtrování se nemusí zdát úplně jednoduché pro všechny uživatele. Může se snadno stát, že pak Stackify prohledává větší množství dat než bylo zamýšleno.

Kapitola 5

Analýza řešení logovací knihovny

Součástí této diplomové práce je také logovací knihovna, která bude umět posílat logy na server. Má se jednat o prototyp pro alespoň jednu platformu, zvolil jsem tedy platformu Android.

5.1 StreamingAndroidLogger

Tato knihovna je celá napsána v jazyce Kotlin, což umocňuje jednoduchost její použitelnosti. Knihovna SteamingAndroidLogger podporuje standardní hladiny logů a také více logovacích výstupů.

Příkladem podporovaných logovacích výstupů jsou[54]:

- LogcatLogger
- CrashlyticsLogger
- On-Device-Weblogging
- UDP
- SystemLogging

V případě potřeby je zde *interface*, který můžeme implemetovat podle svého a tím rozšířit tuto logovací knihovnu.

5.1.1 Výhody

Výhodou této knihovny je to, že je napsána v Kotlinu. Kotlin usnadňuje řešení spousty problémů a vytváření bezpečnějších knihoven a aplikací. Také její použití je jednoduché.

5.1.2 Nevýhody

Nevýhodou této knihovny je formát zprávy. Ten bohužel nemůžeme moc modifikovat, pokud bychom nechtěli nějak víc zasahovat do kódu.

5.2 Timber

Timber je populární logovací knihovna, která je napsána v programovacím jazyce Kotlin.[55] Tato knihovna v základu neumožňuje posílání logů na server, nicméně je nám umožněno ji snadno rozšířit implementací abstraktní třídy *Tree*.

V základu máme implementovanou třídu *Tree* jako *DebugTree*. Výhodou této implementace je, že v produkční verzi aplikace se nevypisují logy, které by mohli obsahovat citlivé informace.

5.2.1 Výhody

Výhodou je, že pro lokální použití je tato knihovna výborná a nemusíme na ní již nijak upravovat, nepotřebujeme-li však něco specifického.

5.2.2 Nevýhody

Nevýhodou této knihovny je to, že v základu nepodporuje logování na server nebo do souboru. Další nevýhodou této knihovny je také to, že všechen kód je v jedné třídě, což mu ubírá na přehlednosti.

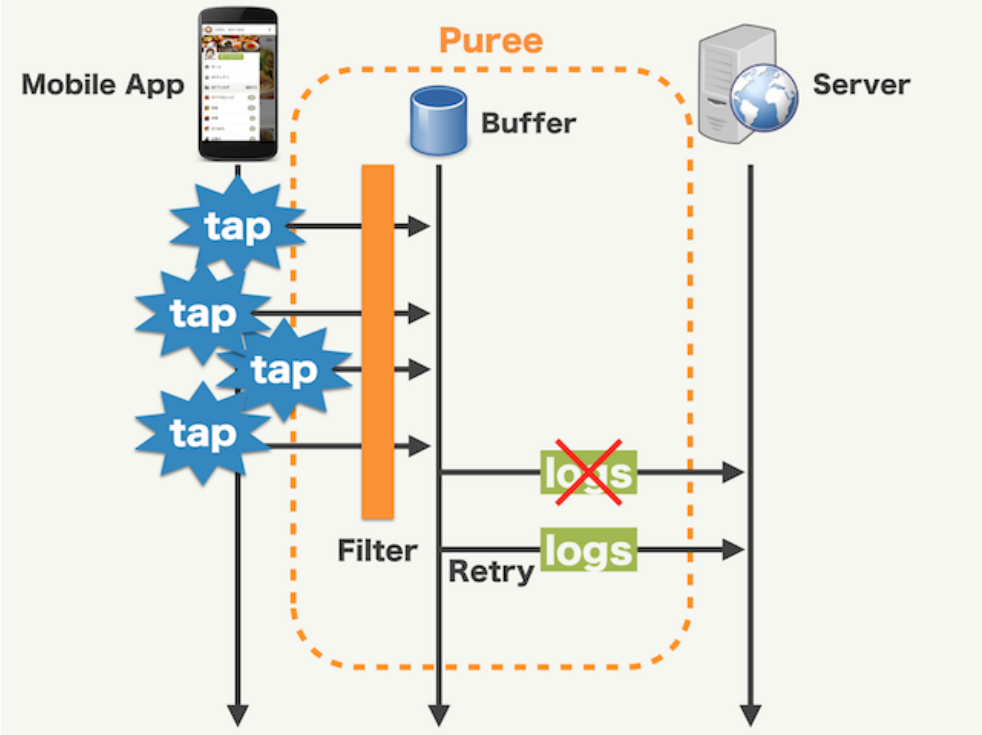
5.3 Puree

Puree je knihovna na posílání logů na server, která je psaná v programovácím jazyce Java.

Její hlavní funkce jsou[56]:

- **Filtrování** - umožňuje nám obohacovat logy o dodatečné informace.
- **Shromažďování** - knihovna umožňuje shromažďovat logy a vypořádat se s nimi později (například odeslat).
- **Hromadné odeslání** - můžeme poslat nashromážděné logy v jedné zprávě na server.
- **Opakování** - pokud nastane chyba, tak se bude odesílání zkoušet znovu, dokud se odeslání nepovede.

Knihovna také umožňuje ukládat data třeba do souboru a odeslat je pak později či je případně odeslat rovnou. V případě, že soubory ukládáme na disk, tak můžeme nechat tyto soubory pravidelně promazávat, abychom nezabírali moc místa v úložišti.



Obrázek 5.1: Ukázka funkce Puree v aplikaci[56]

5.3.1 Výhody

Výhodou knihovny Puree je její funkcionalita. Ať už se jedná o obohacování logů o informace, dočasné skladování či opakování poslání logů na server při chybě.

5.3.2 Nevýhody

Nevýhodou této knihovny je její používání. Její používání vyžaduje implementaci některých tříd a nelze ji bez toho rozumně použít. Také je bohužel napsána v Javě, od čehož se v dnešní době na Androidu upouští.

5.4 ExternalLogger-Android

ExternalLogger je knihovna pro Android aplikace, která umožňuje ukládání logů do paměti zařízení. Uložená data vydrží v paměti zařízení i při vypnutí aplikace nebo vypnutí celého zařízení. Pak jsou data kdykoliv k dispozici a lze je pak exportovat třeba do emailu nebo poslat na server.[57]

Toto řešení knihovny je postavené na Android knihovně *Room*. Díky tomu se jedná o spolehlivé úložiště dat.

5.4.1 Výhody

Výhodou knihovny je spolehlivé uložení dat. Také dává uživateli plnou kontrolu nad mazáním dat, avšak zde chybí automatické mazání. Další výhodou je, že logy lze vyhledávat podle různých klíčů, což může být užitečné.

5.4.2 Nevýhody

Malou nevýhodou této knihovny je, že je napsána v programovacím jazyce Java. Další nevýhodou je práce s touto knihovnou. Na běžné logování je zbytečně složitá. Navíc v základu nepodporuje posílaní dat na server, takže si posílání dat na server musí uživatel řešit sám.

5.5 Android-tao-log

Tato malá knihovna, která je velmi informativní a nenáročná pro systém je velmi užitečná pro Android vývojáře. Knihovna automaticky obohacuje vytvářené logy o dodatečné informace, které usnadňují vývojářům hledání chyb.[58]

Tato knihovna rovněž podporuje ukládání logů do souboru, ze kterého je později můžeme vytáhnout či soubor nějakým způsobem sdílet.

5.5.1 Výhody

Výhodou této knihovny je automatické generování *TAGu*, což je velice užitečné. Knihovna automaticky přidává k logu i jméno souboru, číslo řádku a odkaz do kódu. Také umožňuje logování dlouhých zpráv. Také je výhodou, že knihovna byla nedávno celá přepsána do Kotlinu.

5.5.2 Nevýhody

Nevýhodou této knihovny je chybějící možnost posílání logů či souborů s logy na server.

5.6 KTimber

Logovací knihovna KTimber je knihovna implementovaná v programovacím jazyce Kotlin. Tato implementace vychází z knihovny Timber, která napsal Jake Wharton.[59]

Hlavní rozdíl oproti původní implementaci knihovny Timber je možnost ukládání logů do souboru ve formě HTML[19]. Knihovna nám také umožňuje snadné sdílení těchto souborů. A v neposlední řadě nám přináší i implementaci třídy *ReleaseTree*, kterou můžeme použít pro produkční verzi aplikace.

5.6.1 Výhody

Výhodou této implementace je již dříve zmíněné ukládání logů do souboru. Také velmi užitečné může být implementace logování pro produkční aplikaci.

5.6.2 Nevýhody

Nevýhodou této nadstavby je stále chybějící možnost posílat logy na server. Také formát, ve kterém aplikace ukládá data (HTML), není úplně vhodný.

5.7 HyperLog

HyperLog je jednoduchá logovací knihovna postavená na standardní logovací knihovně pro Android. To znamená, že můžeme logy například při ukládání logů do souboru vidět zároveň logy na standardním výstupu. Knihovna také umožňuje ukládat logy do databáze, ze které pak mohou být přesunuty do souboru nebo poslány přímo na server.[60]

Další funkcí, kterou tato knihovna má, je volitelný formát logu. Formát logu si tu totiž snadno můžeme nadefinovat sami, což může být velmi užitečné.

Také tu najdeme jednu užitečnou funkci, která nám umožňuje hlídat logy v produkční aplikaci. V případě produkční aplikace nastavíme hladinu logů tak, abychom při lokálním monitorování viděli pouze velmi závažné chyby a nebo vůbec žádné. To proto, aby se nějakém logu náhodou neobjevila nějaká citlivá data.

5.7.1 Výhody

Velkou výhodou této logovací knihovny je, že můžeme měnit formát logu dle naší potřeby. Dále máme možnost ukládat logy na paměťové úložiště a poslat je na server najednou ve větším množství. Také tu máme automatické mazání starých souborů. V neposlední řadě je pak také výhodou této knihovny její jednoduché používání.

[19] HyperText Markup Language

5.7.2 Nevýhody

Projekt je bohužel napsaný v Javě a není dále vyvíjen. V projektu jsou také nějaké chyby, které nejsou opravené.

5.8 Kotlin Graylog

Kotlin Graylog je knihovna, která umí posílat logy na Graylog server[61]. Knihovna vytváří JSON objekt, do kterého ukládáme zprávy a celý objekt se pak posílá na server. Knihovna umí požadavky k odeslání dočasně ukládat a posílat je pak na server v pravidelných intervalech.

5.8.1 Výhody

Výhodou je, že Graylog bude umět tyto zprávy s jistotou přečíst. Další výhodou je, že knihovna vkládá tyto hotové zprávy do fronty a ty v pravidelných intervalech pak posílá na server. Jednou z dalších výhod pak je, že knihovna je psána celá v Kotlinu.

5.8.2 Nevýhody

Velkou nevýhodou této knihovny je omezení, se kterým můžeme logovat. Pokud chceme logovat, tak si musíme víceméně vytvořit ručně celou strukturu objektu, což není zrovna nejjednodušší způsob logování.

5.9 Timber-Loggly

Timber-Loggly je jedna z dalších implementací postavených na knihovně Timber. Tato konktétní implementace je nastavená na fungování s logovacím serverem Loggly.[62]

5.9.1 Výhody

Výhodou této knihovny je, že stačí zadat jeden řetězec znaků a o spojení se už postará knihovna. Také používání je velmi jednoduché a intuitivní.

5.9.2 Nevýhody

Funguje pouze pro logovací server Loggly a pro jiné bychom si museli opět napsat vlastní implementaci třídy *Tree*.

5.10 ACRA

ACRA je knihovna, která umožňuje Android aplikacím automaticky posílat jejich zprávy o pádu na server. Knihovna je vyvinutá pro vývojáře Android aplikací, aby jim pomohla sesbírat užitečná data o jejich aplikaci, například proč spadla nebo proč se chovala nepředvídaně.[63]

Tato knihovna má i doporučené serverové řešení, ale teprve nedávno bylo toto serverové řešení vydáno ve své první stabilní verzi. Proto jej neuvádím ve své práci, protože se obávám chyb, které by se mohly náhodně objevovat.

Obecně je tato knihovna dost komplexní, ale zaměřuje se hodně na statistiky ohledně běhu zařízení. Sice sbírá automaticky kritické chybové hlášky z běhu aplikace, ale jejím záměrem není sbírat běžné logy. Přestože to jde, tak to není úplně přímočaré.

5.10.1 Výhody

Výhodou této knihovny je komplexní sběr informací o běhu a chybách v zařízení. Dalšími výhodami této knihovny jsou že je open source[64] a má dobrou dokumentaci.

5.10.2 Nevýhody

Velkou nevýhodou této knihovny je, že nemá dobrou podporu posílání běžných logů na server. Její komplexnost jí ubírá na snadném a intuitivním použití.

5.11 Splunk Mint

Splunk Mint je SDK, které posílá souhrnná data z mobilních zařízení na Splunk Enterpise server. Firma však oznámila konec podpory do konce roku pro tento produkt, proto rovnou nepřichází v úvahu.[65]

Kapitola 6

Návrh

V této kapitole rozeberu návrh vhodného řešení logovacího serveru. Původní myšlenka byla naprogramovat malý server, který bude splňovat všechny funkční požadavky. Po důkladné analýze existujících řešení se ukázalo, že taková řešení již existují a mohou nabídnout o mnoho více funkcionalit a výpočetní síly, protože jsou vyvíjeny celými týmy vývojářů.

Zvoleným serverovým řešením je tedy Graylog, které splňuje všechny definované funkční požadavky a nabízí spoustu funkcionalit navíc. V následující sekci rozeberu, jak jsou funkční požadavky splněny.

6.1 Naplnění požadavků na server

Zde je seznam jednotlivých požadavků na server a jak jsou splněny.

6.1.1 Funkční požadavky na logovací server

- **Server musí mít možnost vystavení do internetu**

  Graylog je koncipovaný jako centrální server pro sběr logů. Buď jej můžeme použít jako cloudové řešení[66], či jej spustit na vlastním serveru a přidělit mu veřejnou IP adresu a bude dostupný z internetu.

- **Server musí mít REST API**

  Graylog má komplexní a robustní REST API.[67]

- **Server musí být schopný přijímat logy ve formátu JSON**

  Graylog přijímá JSON buď pomocí GELFu[68] nebo JSON extraktoru[69].

- **Server musí sám mazat staré záznamy**

  Graylog umí mazat staré indexy a lze si ručně nastavovat dobu, po kterou nebudou smazány. Také je tu možnost archivace starých logů.[70]

- **Server musí být schopný přijímat logy z různých zařízení**

  Graylog umožňuje nastavit různé způsoby, kterými můžeme data posílat na server. Tyto možnosti zajišťují, že data budou přijímány nezávisle na zařízení, ze kterého přijdou.[71]

- **Server musí být zabezpečený**

  Graylog je schovaný za přihlašovací obrazovkou, přes kterou musíme projít, pokud chceme dále se serverem pracovat. Server také podporuje logování aktivity[72].

- **Server musí mít skupiny uživatelů s různými oprávněními**

  Graylog umožňuje vytvářet skupiny s ruznými právy a také umožňuje již v základu podporu LDAP autentizace.[73]

- **Server musí být schopný zobrazovat přijaté logy**

  Graylog umí zobrazovat přijaté logy.[74]

- **Server musí být schopný filtrovat přijaté logy**

  Graylog umí filtrovat logy. Syntaxe dotazovacího jazyka je podobná syntaxi Lucene.[75]

- **Komunikace se serverem musí být zabezpečená**

  Graylog podporuje komunikaci pomocí TLS/SSL.[76]

- **Server musí být spustitelný i jako lokální instance**

  Graylog lze spustit i na lokálním počítači.[77]

Vidíme zde, že Graylog splňuje všechny funkční požadavky na server. Navíc má spoustu dalších funkcionalit, které možná nejsou všechny potřeba, ale spousta z nich je velmi užitečných.

6.1.2 Nefunkční požadavky na logovací server

Zde je seznam nefunkčních požadavků a jak se povedlo je splnit.

- **Server musí být snadno přenositelný**

  Graylog poběží jako Docker kontejner, který bude automaticky nastaven pomocí konfiguračního souboru a nasdílených konfigurací.

- **Server musí být schopen zvládnout 1000 logovacích požadavků za sekundu**

  Graylog je možné škálovat podle potřeby horizontálně i vertikálně.[78] Na lokálním počítači jsem byl schopen zvládnout tuto zátěž i na jedné instanci.

- **Server musí být schopen pracovat s větším množstvím dat**

  Graylog je schopný pracovat s velkým množstvím dat a být stále rychlý.[79]

- **Server musí být snadno rozšiřitelný**

  Graylog je možné rozšiřovat pomocí balíčků (Add-ons), kterých jsou stovky dostupných přeš *Graylog Marketplace*.[80]

- **Server musí být snadno udržitelný**

  Graylog lze konfigurovat jak přes RESTové rozhraní tak i ve webovém rozhraní. Konfigurace není složitá a Graylog poskytuje podrobnou dokumentaci.[81]

Ač jsou nefunkční požadavky těžko měřitelné, tak se zde jejich splnění povedlo a většinu svých tvrzení jsem i podložil dokumentací či článkem.

6.1.3 Proč Graylog

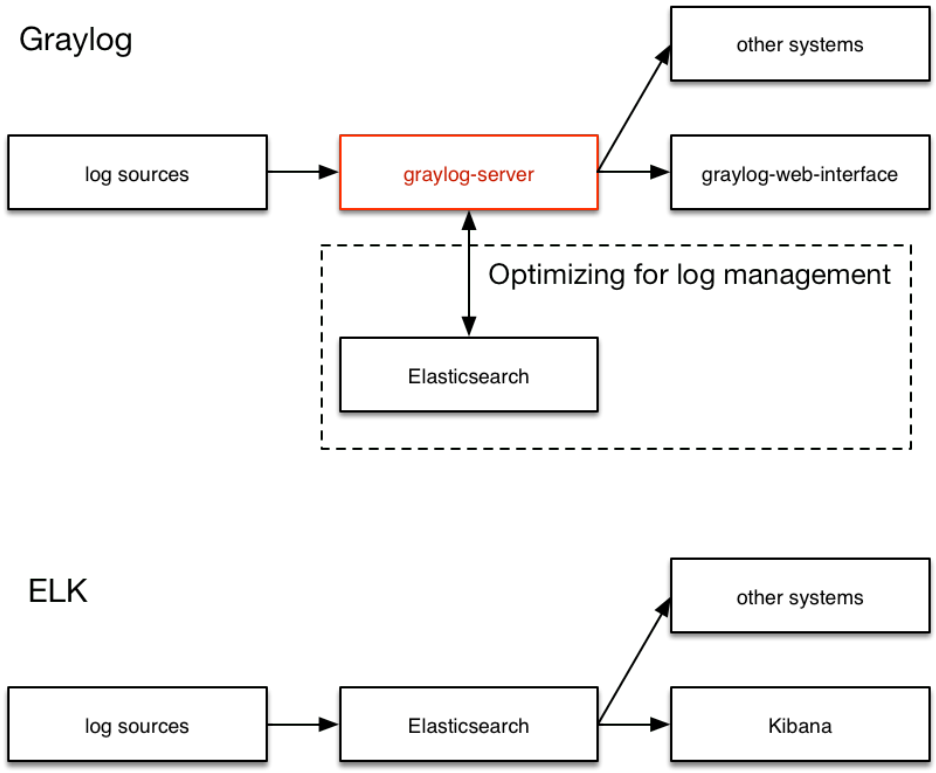
Přestože Graylog splňuje všechny funkční i nefunkční požadavky na server, tak by se našla i jiná řešení, která by tyto požadavky splňovala. Jednou z nich je například Elastic Stack, ten je Graylogu velkou konkurencí.

Elasitc Stack je sice velice dobré řešení, nicméně je připraveno více na práci s velkými daty (Big Data). Graylog je také jednodušší na nastavení a údržbu, protože Elastic Stack se sestává z více komponent a každá má vlastní nastavení. Dále můžeme říct, že uživatelské rozhraní Graylogu je přívětivější a jednodušší než Kibana. Dále také poskytuje lepší systém rozdělování práv uživatelům, což například Elastic Stacku hodně dlouho chybělo.[82]

Graylogu můžeme vytknout omezenější možnosti vizualizace a grafů z dat. V tom je Elastic Search, přesněji řečeno Kibana, napřed. Můžeme však tento nedostatek schovat přidáním Grafany, což je open source projekt, který tento nedostatek doplní.[83]

6.2 Architektura serveru

Graylog je od začátku navrhován jako server pro správu logů a proto se jej tomu autoři snaží co nejvíce přizpůsobit. Snaží se rovněž, aby byl Graylog nezávislý na konkrétní databázi pro správu logů. Monentálně na to využívají Elasticsearch, ale Graylog je připravený tak, že Elasticsearch může být kdykoliv vyměněn za lepší databázové a vyhledávací řešení.

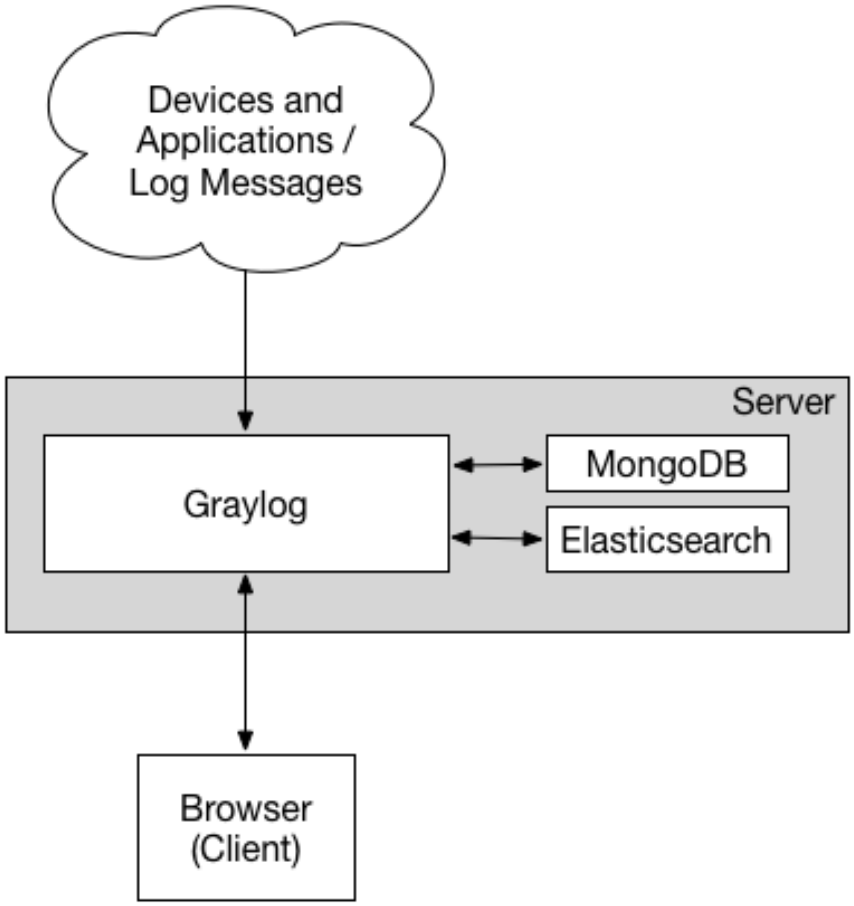


Obrázek 6.1: Ukázka architektur Graylogu a Elastic Stacku[25]

Na obrázku 6.1 je vidět zasazení Graylogu jako komponenty a porovnání s Elastic Stackem. U Graylogu je vidět, že Elasticsearch je zde čistě jako databáze a jediná vazba je zde na Graylog server, takže jej můžeme snadno zaměnit.

6.2.1 Minimální konfigurace

Na obrázku 6.2 je vidět minimální konfigurace Graylog serveru. Je zde vidět, že nikde není nic redundantní. Jedná se o malé řešení, které může být vhodné na nekritické či testovací servery. Díky jednoduchosti této architektury není komplikované tuto konfiguraci nastavit.

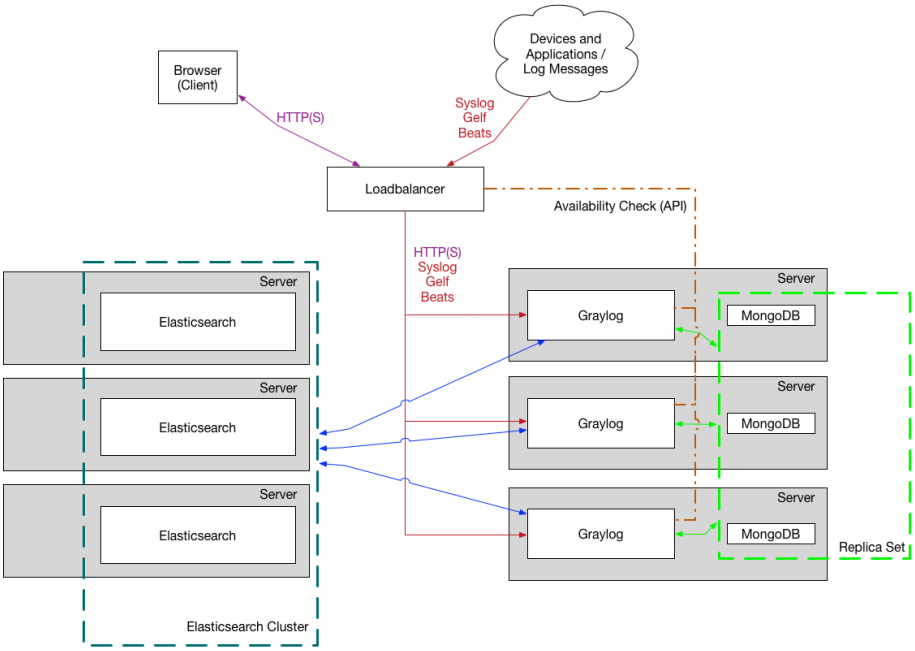


Obrázek 6.2: Minimální konfigurace Graylogu[78]

V našem případě se takováto konfigurace bude hodit při spouštění na lokálním počítači. Na lokálním stroj se bude připojovat pár jednotek, nejvíce pár desítek zařízení, což by tato konfigurace měla hravě zvládnout, pokud lokální stroj bude mít dostatek zdrojů.

6.2.2 Konfigurace většího serveru

V případě, že budeme potřebovat zvládat velké množství dat z různých zařízení a jednoduchá konfigurace serveru nebude stačit, tak lze konfigurace rozšířit. V tomto případě se jedná o horizontální škálování, kdy budeme přidávat více instancí vedle sebe.



Obrázek 6.3: Konfigurace většího serveru[78]

Na obrázku 6.3 je vidět, že se vytváří jak více instancí Elasticsearche, tak i více instancí Graylogu a MongoDB. Kromě zduplikovaných komponent tam také přibyla komponenta *Loadbalancer*. Tato komponenta se stará o rozkládání zátěže mezi jednotlivé instance Graylogu. Rovněž si kontroluje, jestli jsou tyto instance dostupné a v jakém jsou stavu. V případě, že na nějaké instanci dojde k chybě a přestane fungovat, tak Loadbalancer na tuto instanci přestane posílat požadavky na zpracování.

Můžeme k tomuto nastavení přidat třeba i komponentu RabbitMQ nebo Apache Kafka. Tyto komponenty se starají o ukládání požadavků do fronty pro případy, že by požadavků na server bylo mnoho a server je nestíhal odbavovat.

6.3 Návrh logovací knihovny

Po důkladné analýze možných logovacích knihoven mě zaujala knihovna HyperLog. Tato knihovna má velmi jednoduché a intuitivní používání a splňuje všechny požadavky na logovací knihovnu.

Protože knihovna se používá dobře a její kód je srozumitelný, jednoduchý a dobře dokumentovaný, tak zde nebude potřeba větších zásahů.

6.3.1 Funkční požadavky na logovací knihovnu

- **Knihovna musí být schopna posílat logy zabezpečeně**

  HyperLog umí komunikovat přes https.

- **Knihovna musí umět skladovat logy**

  HyperLog umí skladovat logy v SQLite databázi.

- **Knihovna musí umět posílat více logů najednou**

  HyperLog umí posílat na server více logů v jedné zprávě.

- **Knihovna musí umět posílat logy asynchronně**

  HyperLog automaticky přidává požadavky na odeslání do fronty a z té se asynchronně odesílají na server.

- **Knihovna musí umět sama mazat staré logy**

  HyperLog maže automaticky staré logy a zároveň umožňuje si nastavit časový interval, za jaký budou staré logy smazány.

6.3.2 Nefunkční požadavky na logovací knihovnu

- **Knihovna musí být snadno rozšiřitelná**

  HyperLog je zveřejněný pod MIT licencí, takže může být použit a upravován. Kód je čitelně napsaný a dobře dokumentovaný. Úpravy či rozšíření jsou tedy jednoduché.

- **Knihovna musí být jednoduchá na použití**

  HyperLog lze velice jednoduše přidat do projektu a do pár minut už může posílat logy na server.

Kapitola 7

Realizace

V této kapitole popíši způsob realizace a používání vybraných technologií.

7.1 Realizace serveru

V této sekci rozeberu realizaci serveru. Server je již hotové řešení, proto se nebude jednat o jeho implementaci, ale o jeho konfiguraci.

U Graylogu jsme si mohli vybrat z verzí Graylog Open a Graylog Enterprise. Graylog Open je úplně zadarmo, nicméně mu oproti Graylog Enterprise chybí nějaká funkcionalita. Graylog Enterprise můžeme použít také zdarma, nicméně je zde však limit na 5 GB přijatých dat za den. Zvolil jsem tedy Graylog Enterprise verzi, protože má obsahuje zajímavé funkcionality, které jsem zmiňoval v předchozích kapitolách.

Jak jsem již zmiňoval dříve, tak celé řešení poběží jako sada Docker kontejnerů. Technologii Docker jsem zvolil pro její snadnou přenositelnost. Stačí mít nainstalovaný Docker a zároveň Docker Compose a můžeme server spustit. Toto snadné spuštění je velkou výhodou, protože již nepotřebujeme řešit problémy se závislostmi na různých knihovnách, který bychom si pak případně museli instalovat do systému.

7.1.1 Lokální verze

Lokální verze pro vývojáře je složená z následujících kontejnerů:

- MongoDB
  - verze 4.2
  - slouží k uchovávání konfigurace graylogu
  - obsah DB je mapován na adresář *mongodb* pro jednoduchou přenositelnost nastavení

- Elasticsearch
  - verze 7.10.2
  - dostupná paměť je omezena pomocí *ulimits* na 1 GB
  - data z Elasticsearche jsou mapována na adresář *elastic*
- Graylog
  - verze 4.0 - Enterprise
  - Na kontejner jsou mapované 2 složky *graylog/data* a *graylog/config*. Složka *config* obsahuje konfigurační soubory Graylogu pro jednoduché nastavení a self-signed certifikát s klíčem nezbytným pro https komunikaci.
  - Jako proměnné prostředí se předávají různé konfigurace, například zapnutí/vypnutí TLS, adresa a port, na kterých Graylog vystaví webové a API rozhraní, a cesty k certifikátu a klíči. Tyto proměnné se také dají nastavit přímo v konfiguračním souboru zmíněném výše.
  - Kontejner také vystavuje některé porty nezbytné pro správné fungování a komunikaci s okolním světem.

Všechny kontejnery jsou umístěné ve virtuální síti Dockeru typu *bridge*, čímž máme zajištěno, že na sebe jednotlivé kontejnery vidí a mohou spolu komunikovat po síti.

Vzhledem k tomu, že jednotlivé kontejnery startují pod uživatelem, který je definovaný v Dockerfile daného obrazu, musíme nejdříve nastavit správná práva na složky, aby jsme si byli jisti tím, že kontejner má práva zapisovat a číst potřebné soubory. Příkazy jsou definované v souboru *README*. Poté stačí spustit Docker a zadat příkaz *docker-compose up*, který vše udělá za nás. Vytvoří virtuální síť, a pokud nenalezne v lokálním registru potřebné obrazy, tak je stáhne z internetu. Následně spustí jednotlivé kontejnery. V logu pak můžeme kontrolovat, zda vše probíhá v pořádku a nikde není chyba, která by způsobila kritickou chybu při startu kontejnerů

7.1.2 Serverová verze

Serverová část se od lokální liší následujícím způsobem:

- Je přidaná reverzní proxy (Apache httpd). Tato proxy poslouchá na portu 80 a přesměrovává komunikaci na Graylog server. Může také sloužit pro terminaci SSL a v případě přidání dalšího Graylog serveru pak bude sloužit i jako load balancer. Konfigurační soubor této proxy je poté ve složce httpd.

- Je přidaný druhý uzel pro Elasticsearch. Výhoda tohoto řešení spočívá například v replikaci dat mezi uzly, kdy v případě výpadku jednoho z nich nedojde ke ztrátě dat, ale data budou obnovena z druhého uzlu.

7.1.3 Připojení zařízení k serveru

V běžném používání budeme mít centrální server, který bude mít přiřazenou veřejnou IP adresu a budeme ke Graylogu přistupovat přímo nebo bude Graylog schovaný za nějaký loadbalancer. Nicméně připojení v tomto případě je přímočaré.

Pokud však chceme posílat logy na lokální instanci serveru, pak máme 2 možnosti, jak toho docílit.

- Vytvořit hotspot na lokálním stroji a připojit zařízení na něj.
- Nastavit v routeru port forwarding.

Pokud nastavíme jednu z těchto možností, pak budou zařízení v jedné síti a bude jim umožněno spolu komunikovat na zvolených portech.

V zařízení pak použijeme aplikaci, která používá nějakou knihovnu na posílání logů na server. V další sekci pak ukážu jak aplikace knihovnu přidat a v několika minutách už můžou být logy na serveru.

7.2 Realizace logovací knihovny

Jak jsem již zmínil, rozhodl jsem se použít knihovnu HyperLog. Vytvořil jsem si kopii z originálního repozitáře, ve které pak mohu dělat úpravy.

7.2.1 Použití knihovny

Jak jsem již zmiňoval dříve, tak knihovna se velmi snadno používá. Tuto knihovnu můžeme v projektu použít několika způsoby. Možnosti zanesení knihovny do projektu:

- zkopírování celého modulu HyperLog
- ručním přidáním knihovny ve formátu aar[20]
- přidáním závislosti na knihovně

Pokud bychom tedy přidali HyperLog do projektu jako závislost na knihovně, pak bychom přidali do daného modulu, konkrétně do souboru *build.gradle*, tyto závislosti:

[20] AAR soubor obsahuje knihovnu pro vývoj Android aplikací.[84]

```
dependencies{
    ...
    implementation 'com.hypertrack:hyperlog:0.0.10'
    implementation 'com.android.volley:volley:1.1.0'
    implementation 'com.google.code.gson:gson:2.8.1'
    ...
}
```

V případě, že bychom HyperLog přidali jako modul, pak bym přidání vypadalo následovně.

```
dependencies{
    ...
    implementation project(':hyperlog')
    ...
}
```

Ještě bychom pak ale museli přidat do souboru *settings.gradle*, který se nachází v kořenovém adresáři, závislost na modul.

```
include ':app', ':hyperlog'

rootProject.name = "SpaceX"
```

Obrázek 7.1: Soubor *settings.gradle* z projektu SpaceX[85]

Použití v samotném projektu je již triviální. Knihovna potřebuje při startu aplikace inicializovat a případně nastavit základní parametry. Nejlepším místem na toto nastavení je metoda *onCreate* v hlavní aktivitě projektu.

```
32
33      override fun onCreate(savedInstanceState: Bundle?) {
34          super.onCreate(savedInstanceState)
35
36          HyperLog.initialize( context: this)
37          HyperLog.setLogLevel(Log.VERBOSE)
38          HyperLog.setURL("http://192.168.2.212:2222/gelf")
39
```

Obrázek 7.2: Inicializace HyperLogu v projektu SpaceX[85]

Na obrázku 7.2 vidíme ukázku inicializace. Nejdřív je potřeba nastavit *context*, který umožňuje knihovně například zapisovat do databáze. Metoda *initialize* má i více variant, kde můžeme například nastavit délku skladování logů než budou vymazány, či třeba můžeme nastavit formát logů. Tyto další

možnosti konfigurace však lze kdykoliv nastavit či změnit, takže to nemusíme dělat v metodě *initialize*.

Tím je již knihovna nastavená a víc konfigurovat nemusíme. V tomto případě je však na dalším řádku volání metody *setLogLevel*, která nastavuje minimální hladinu závažnosti logů, které budou zobrazovány v Logcatu.

Pokud však chceme posílat data na server, pak musíme ještě nastavit adresu serveru. Na to slouží metoda *setURL*. Po nastavení této adresy již nám nic nebrání v posílání logů na server.

Logy z aplikace pak sbíráme podobně jako bychom používali standardní logovací knihovnu pro Android, akorát místo *Log* používáme *HyperLog*. Zde je příklad:

```
1 HyperLog.d(TAG, "attaching LaunchesFragment")
```

Takto nasbírané logy pak můžeme odeslat na server. Zde je příklad implementace odeslání logů na server.

```
96      HyperLog.pushLogs(context, compress: false, object: HLCallback() {
97
98          override fun onSuccess(response: Any?) {
99              Log.d( tag: "TAG", msg: "sending successful")
100         }
101
102         override fun onError(HLErrorResponse: HLErrorResponse?) {
103             Log.w( tag: "TAG", msg: "sending failed!")
104             Log.w( tag: "TAG", HLErrorResponse?.errorCode.toString())
105         }
106     })
```

Obrázek 7.3: Odesílání logů na server v projektu SpaceX[85]

Na obrázku 7.3 je vidět, že metoda *pushLogs* přebírá jako 3. parametr implementaci anonymní třídy, která implementuje rozhraní *HLCallback*. Tam implementujeme metody *onSucces* a *onError*. V případě úspěšného odeslání logů na server se zavolá metoda *onSucces*. Pokud odeslání logů na server selže, pak se zavolá metoda *onError*. V tomto případě se jen vypíše do Logcatu zpráva, jestli jsme uspěli či neuspěli.

Při tomto odesílání jsem rovněž objevil chybu v implementaci této knihovny. Jedná se právě o chybu při odesílání logů na server. Ve chvíli, kdy server vrátí prázdnou odpověď, tak dojde k chybě.

```
34  public abstract class HLCallback {
35
36      /**
37       * Called when a request succeeds.
38       *
39       * @param response The successful response containing the responseObject.
40       */
41      public abstract void onSuccess(@NonNull Object response);
42
43      /**
44       * Called when a validation error occurs, request times out, or fails.
45       *
46       * @param HLErrorResponse The request status.
47       */
48      public abstract void onError(@NonNull HLErrorResponse HLErrorResponse);
49
50  }
```

Obrázek 7.4: Rozhraní pro odesílání logů na server[60]

Na obrázku 7.4 můžeme vidět, že řádky 41 a 48 obsahují anotaci *@NonNull*. Tato anotace říká, že daná proměnná nikdy nebude prázdná, vždy bude obsahovat nějakou hodnotu. Pokud bychom se tam náhodou pokusili nedat žádnou hodnotu, pak celá aplikace spadne.

Při testování této knihovny mi se mi stalo, že server mi poslal prázdnou odpověď, takže aplikace spadla. Při pádu aplikace je vyhozena výjimka, která však nejde chytit, takže pokud nezaručíme neprázdnou odpověď ze serveru, tak nám již zbývá jen opravit knihovnu.

Na předchozím obrázku 7.3 je vidět, že chyba je zde již opravena. Je zde sice rozdíl v tom, že knihovna je napsána v Javě a aplikace SpaceX v Kotlinu, ale funkcionalitu to neovlivní. Na řádcích 98 a 102 můžeme vidět, že argumenty funkce mají u sebe otazník, což v Kotlinu znamená, že proměnná může být prázdná.

Kapitola 8

Testování

Vzhledem k tomu, že se jedná o hotová řešení, tak není moc co testovat. V případě serveru používáme Docker image jednotlivých komponent, které jsou jistě testované již vývojáři. U knihovny však pracujeme i se zdrojovými kódy, proto je tam více možností pro testování.

Jak jsem již zmínil, jedná se o hotové řešení v Docker kontejnerech, takže jsem to testoval jako celek. Jedná se o tyto tři testy:

- test dostupnosti
- test konektivity
- zátěžový test

Tyto testy jsem napsal formou skriptů a jako jazyk jejich implementace jsem zvolil Bash. Všechny testy lze spustit jedním skriptem, který se jmenuje *runAllTests.sh*. Ten postupně spustí testy jeden po druhém. Další pustí vždy jen když skončí úspěchem.

Testy je potřeba trochu nastavit. Musí se jim například zadat IP adresa serveru, port serveru a podobně. Tyto nastavení najdeme v souboru *config.sh*, kde je můžeme upravovat dle potřeby.

Dalším souborem ve složce testy je soubor *utils.sh*, který obsahuje nějaké sdílené funkce, které se používají napříč skripty.

8.1 Test dostupnosti

Cílem testu dostupnosti je zjistit, jestli server běží. Testuje se nejdříve, jestli je zadaná IP adresa vůbec dostupná. Pokud ano, tak se zkusí na server i připojit a to přes konkrétní port. Pokud se povede i to, pak test ohlásí úspěch.

8.2 Test konektivity

Tento test zkouší posílání dat na server. Test sestaví zprávu a zkusí jí poslat na server. Následně pak kontroluje, jestli byla zpráva serverem přijata či odmítnuta.

8.3 Zátěžový test

Zátěžový test zkouší, kolik server zvládne přijmout zpráv. V konfiguračním souboru si můžeme nastavit, kolik se vytvoří uzlů, které to budou posílat zprávy na server, a kolik zpráv má každý uzel poslat. Po spuštění začne každý uzel posílat zprávy na server. Každá zpráva obsahuje identifikační číslo uzlu a číslo, které říká, kolikátá to je zpráva na daném uzlu. Na serveru pak můžeme sledovat jeho zátěž a jaké zprávy server zpracoval.

Závěr

Cílem této magisterské práce byl funkční prototyp serveru pro sběr logů, který bude v budoucnu nasazen v zadavatelské firmě, kde bude sbírat logy z mobilních zařízení, na kterých poběží aplikace vyvíjené firmou. Dále je cílem této práce vytvořit logovací knihovnu, pomocí které bude možné posílat logy na server. Výsledkem této práce je prototyp serveru, který je možné snadno spustit jak na serveru, tak i na lokálním počítači. Dále je výsledkem práce prototyp logovací knihovny, která na tento server umí posílat logy z aplikace. Funkčních i nefunkčních požadavků bylo na server a knihovnu více, například server musí mít REST API.

Na základě komplexní analýzy jsem vyhodnotil, že řešením, které nejlépe splňuje zadané požadavky na server je logovací nástroj Graylog. Obdobně z analýzy logovací knihovny vyplynulo, že nejlepší dostupnou volbou je HyperLog, který lze případně dále vylepšovat a rozšiřovat.

Výsledkem této diplomové práce jsou 2 verze Graylogu, které obě běží v Docker kontejnerech. Jedna verze je pro lokální použití vývojáři a druhá verze je připravená pro serverové použití. Dalším výstupem této práce je logovací knihovna, ve které jsou opraveny nalezené chyby a je přidána nová funkcionalita.

Dále jsem v rámci své práce podrobil server testování. Jedním z cílů testování bylo, zda server naplňuje zadané nefunkční požadavky na zpracování dat v daném časovém úseku a výsledky byly více než uspokojivé.

Tento server dokáže přijímat velké množství dat i v krátkém časovém úseku a dále v nich pak rychle vyhledávat či je zobrazovat. Server má i spoustu dalších funkcionalit, které nebyly v požadavcích specifikovány, ale budou využity.

Literatura

[1] Importance Of Log Files. [online]. DEV Community © 2016 - 2021, [cit 9. 1. 2021]. Dostupné z: https://dev.to/flippedcoding/the-importance-of-log-files-37d6

[2] *What Is Structured Logging and Why Developers Need It.* [online]. © 2020 Stackify, [cit 4. 3. 2021]. Dostupné z: https://stackify.com/what-is-structured-logging-and-why-developers-need-it/

[3] *Write and View Logs with Logcat.* [online]. developers.android.com. [cit 8. 1. 2021]. Dostupné z: https://developer.android.com/reference/android/util/Log.html

[4] *What Is an Exception?* [online]. Copyright © 1995, 2020 Oracle. [cit 9. 1. 2021]. Dostupné z: https://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html

[5] *Log.* [online]. developers.android.com. [cit 8. 1. 2021]. Dostupné z: https://developer.android.com/studio/debug/am-logcat

[6] *Stack Trace.* [online]. © 2021 Techopedia Inc., [cit 29. 3. 2021]. Dostupné z: https://www.techopedia.com/definition/22307/stack-trace

[7] *Java Garbage Collection.* [online]. Copyright © 2011-2018, JavaTpoint. [cit 12. 1. 2021]. Dostupné z: https://www.javatpoint.com/Garbage-Collection

[8] Comer, T.: *What is remote logging?* [online]. © , [cit 10. 1. 2021]. Dostupné z: https://www.loggly.com/blog/what-is-remote-logging/

[9] *Centralized and Local Logging.* [online]. Clicksoftware, [cit 10. 1. 2021]. Dostupné z: https://wiki.cloud.clicksoftware.com/v8doc/en/configuration/configuring-clickplatform/server-logs/centralized-and-local-logging

[10] Eriksson, U.: *Why is the difference between functional and Non-functional requirements important?* [online]. © 2021 ReQtest. [cit 12. 1. 2021]. Dostupné z: https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/

[11] *Specifikace funkčních a nefunkčních požadavků.* Ing. Jakub Průša, jednatel za Quanti. Ing. Zdeněk Balák, vedoucí diplomové práce. Praha, Leden 2021.

[12] *What is REST?* [online]. © 2021 Codecademy, [cit 16. 4. 2021]. Dostupné z: https://www.codecademy.com/articles/what-is-rest

[13] *What is an API? (Application Programming Interface).* [online]. ©2021 MuleSoft LLC, a Salesforce company, [cit 16. 4. 2021]. Dostupné z: https://www.mulesoft.com/resources/api/what-is-an-api

[14] *Introducing JSON.* [online]. © json.org, [cit 16. 4. 2021]. Dostupné z: https://www.json.org/json-en.html

[15] Nguyen, T.: *5 Splunk Alternatives - Faster, Affordable Log Management Solutions.* [online]. © 2021 LogDNA, [cit 19. 1. 2021]. Dostupné z: https://www.logdna.com/blog/5-splunk-alternatives-for-logging-their-benefits-shortcomings-and-which-one-to-choose

[16] *What is open source?* [online]. Copyright ©2021 Red Hat, Inc., [cit 8. 4. 2021]. Dostupné z: https://opensource.com/resources/what-open-source

[17] *What is the ELK Stack?* [online]. © 2021, Elasticsearch B.V. [cit 13. 1. 2021]. Dostupné z: https://www.elastic.co/what-is/elk-stack

[18] Horovits, D.: *THE COMPLETE GUIDE TO THE ELK STACK.* [online]. © 2015-2021, Logshero Ltd. [cit 13. 1. 2021]. Dostupné z: https://logz.io/learn/complete-guide-elk-stack/

[19] Gelbmann, M.: *Elasticsearch moved into the top 10 most popular database management systems.* [online]. Copyright © 2021, solid IT gmbh [cit 17. 1. 2021]. Dostupné z: https://db-engines.com/en/blog_post/70

[20] *What is NoSQL?* [online]. © 2021 MongoDB, Inc., [cit 5. 4. 2021]. Dostupné z: https://www.mongodb.com/nosql-explained

[21] *What is SQL?* [online]. Copyright 2021 TechnologyAdvice, [cit 5. 4. 2021]. Dostupné z: http://www.sqlcourse.com/intro.html

[22] *Scaling Horizontally vs. Scaling Vertically.* [online]. © 2020 Section [cit 17. 1. 2021]. Dostupné z: https://www.section.io/blog/scaling-horizontally-vs-vertically/

[23] Tyson, M.: *What is the JVM? Introducing the Java Virtual Machine*. [online]. Copyright © 2021 IDG Communications, Inc., [cit 2. 4. 2021]. Dostupné z: https://www.infoworld.com/article/3272244/what-is-the-jvm-introducing-the-java-virtual-machine.html

[24] *Kibana Query Language*. [online]. © 2021. Elasticsearch B.V., [cit 2. 4. 2021]. Dostupné z: https://www.elastic.co/guide/en/kibana/current/kuery-query.html

[25] *The thinking behind the Graylog architecture and why it matters to you*. [online]. © Copyright 2015-2021 Graylog, Inc., [cit 18. 2. 2021]. Dostupné z: https://docs.graylog.org/en/4.0/pages/ideas_explained.html

[26] Cirelly, J.: *Graylog Review and Alternatives*. [online]. © 2021 Comparitech Limited, [cit 18. 2. 2021]. Dostupné z: https://www.comparitech.com/net-admin/graylog-review/

[27] *Compare Versions*. [online]. © 2021 Graylog, Inc., [cit 8. 3. 2021]. Dostupné z: https://www.graylog.org/products/open-source-vs-enterprise

[28] *Introduction*. [online]. © 2021 Graylog, Inc., [cit 8. 3. 2021]. Dostupné z: https://docs.graylog.org/en/4.0/pages/enterprise/intro.html

[29] *Graylog Sidecar*. [online]. © 2021 Graylog, Inc., [cit 8. 3. 2021]. Dostupné z: https://docs.graylog.org/en/4.0/pages/sidecar.html

[30] *Content Packs*. [online]. © 2021 Graylog, Inc., [cit 10. 3. 2021]. Dostupné z: https://docs.graylog.org/en/4.0/pages/content_packs.html

[31] *About LogDNA*. [online]. © 2021 LogDNA, [cit 22. 1. 2021]. Dostupné z: https://docs.logdna.com/docs/getting-started

[32] *Navigating the Dashboard*. [online]. © 2021 LogDNA, [cit 22. 1. 2021]. Dostupné z: https://docs.logdna.com/docs/how-to-use-the-dashboard

[33] *Comparing 2 Log Management Software Products*. [online]. capterra.com, [cit 19. 1. 2021]. Dostupné z: https://www.capterra.com/log-management-software/compare/148959-153685/Loggly-vs-Logz-io

[34] Idan, H.: *How to Choose the Right Log Management Tool? Sumo Logic vs Graylog vs Loggly vs PaperTrail vs Logentries vs Stackify*. [online]. OverOps, Inc. 2021 ©, [cit 30. 1. 2021]. Dostupné z: https://www.overops.com/blog/how-to-choose-the-right-log-management-tool/

[35] *Remote Logger, Crash Reporter and In-App User Feedback*. [online]. Bugfender, [cit 12. 2. 2021]. Dostupné z: https://bugfender.com/

[36] Ventayol, A.: *The Free Plan Is Back and More Awesome News.* [online]. Bugfender, [cit 12. 2. 2021]. Dostupné z: https://bugfender.com/blog/free-plan-back-and-more-news/

[37] *Logz.io.* [online]. © 2021 GetApp, [cit 6. 2. 2021]. Dostupné z: https://www.getapp.co.uk/software/104075/logz-io

[38] *Grafana.* [online]. Copyright 2021 © Grafana Labs , [cit 2. 4. 2021]. Dostupné z: https://grafana.com/oss/grafana/

[39] *Log Management.* [online]. © © 2015-2021, Logshero Ltd. [cit 6. 2. 2021]. Dostupné z: https://logz.io/platform/log-management/

[40] *Sumo Logic.* [online]. © 2021 Sumo Logic, [cit 22. 1. 2021]. Dostupné z: https://www.sumologic.com/how-it-works/

[41] *Faster monitoring and troubleshooting.* [online]. © 2021 Sumo Logic, [cit 22. 1. 2021]. Dostupné z: https://www.sumologic.com/

[42] Brame, D.: *Datadog Review.* [online]. © 1996-2021 ZIFF DAVIS, LLC. PCMAG DIGITAL GROUP, [cit 30. 1. 2021]. Dostupné z: https://www.pcmag.com/reviews/datadog

[43] *Exporting traces to Datadog.* [online]. © 1996-2021 ZIFF DAVIS, LLC. PCMAG DIGITAL GROUP, [cit 30. 1. 2021]. Dostupné z: https://www.pcmag.com/reviews/datadog

[44] Soroker, T.: *6 Log Management Tools You NEED to Know (And How to Use Them).* [online]. OverOps, Inc. 2021 ©, [cit 30. 1. 2021]. Dostupné z: https://www.overops.com/blog/6-log-management-tools-you-need-to-know-and-how-to-use-them/

[45] *Centralized Log Data.* [online]. ©2019 logentries.com, inc. [cit 18. 1. 2021]. Dostupné z: https://logentries.com/insights/log-data-centralization/

[46] *Logentries.* [online]. © 2021 Salesforce.com [cit 18. 1. 2021]. Dostupné z: https://elements.heroku.com/addons/logentries

[47] *Features.* [online]. ©2019 logentries.com, inc. [cit 18. 1. 2021]. Dostupné z: https://logentries.com/product/analytics/

[48] THEHFTGUY: *The Differences Between Splunk, Kibana and Graylog.* [online]. thehftguy.com, [cit 18. 4. 2021]. Dostupné z: https://thehftguy.com/2020/08/04/the-differences-between-splunk-kibana-and-graylog/

[49] *Certificate Transparency Log add-on for Splunk*. [online]. © 2005-2021 Splunk Inc. [cit 18. 1. 2021]. Dostupné z: https://splunkbase.splunk.com/app/4006/

[50] *Sematext Logs*. [online]. © Sematext Group, [cit 7. 2. 2021]. Dostupné z: https://sematext.com/logsene/

[51] Gheorghe, R.: *20+ Best Log Management Tools for Monitoring, Analytics More: Pros Cons Comparison [2021]*. [online]. © Sematext Group, [cit 8. 2. 2021].

[52] *Stackify APM+ Pricing, Features, Reviews Alternatives*. [online]. © 2021 GetApp, [cit 13. 2. 2021]. Dostupné z: https://www.getapp.com/it-management-software/a/stackify/

[53] *Better code, better deployments, fewer fires*. [online]. © 2020 Stackify, [cit 13. 2. 2021]. Dostupné z: https://stackify.com/

[54] *StreamingAndroidLogger*. [online]. © 2021 GitHub, Inc., [cit 18. 2. 2021]. Dostupné z: https://github.com/kibotu/StreamingAndroidLogger

[55] *Timber*. [online]. © 2021 GitHub, Inc., [cit 23. 2. 2021]. Dostupné z: https://github.com/JakeWharton/timber

[56] *Puree*. [online]. © 2021 GitHub, Inc., [cit 23. 2. 2021]. Dostupné z: https://github.com/cookpad/puree-android

[57] *ExternalLogger-Android*. [online]. © 2021 GitHub, Inc., [cit 25. 2. 2021]. Dostupné z: https://github.com/guy-4444/ExternalLogger-Android

[58] *Android-tao-log*. [online]. © 2021 GitHub, Inc., [cit 25. 2. 2021]. Dostupné z: https://github.com/lordtao/android-tao-log

[59] *KTimber*. [online]. © 2021 GitHub, Inc., [cit 23. 2. 2021]. Dostupné z: https://github.com/ditacristianionut/ktimber

[60] *HyperLog Android*. [online]. © 2021 GitHub, Inc., [cit 3. 3. 2021]. Dostupné z: https://github.com/hypertrack/hyperlog-android

[61] *Kotlin Graylog*. [online]. © 2021 GitHub, Inc., [cit 25. 2. 2021]. Dostupné z: https://github.com/iadvize/kotlin-graylog

[62] *Timber-Loggly*. [online]. © 2021 GitHub, Inc., [cit 26. 2. 2021]. Dostupné z: https://github.com/loggly/timber-loggly

[63] *ACRA*. [online]. © 2021 GitHub, Inc., [cit 3. 3. 2021]. Dostupné z: https://github.com/ACRA/acra

[64] *ACRA*. [online]. © 2021 ACRA organization, [cit 3. 3. 2021]. Dostupné z: https://www.acra.ch/

[65] *Deliver better performing mobile apps*. [online]. Copyright © 2005-2017 Splunk Inc., [cit 4. 3. 2021]. Dostupné z: https://mint.splunk.com/

[66] *Graylog cloud*. [online]. © 2021 Graylog, Inc., [cit 10. 3. 2021]. Dostupné z: https://www.graylog.org/resources/graylog-cloud

[67] *Graylog Enterprise REST API*. [online]. © 2021 Graylog, Inc., [cit 11. 3. 2021]. Dostupné z: https://www.graylog.org/features/api

[68] *GELF*. [online]. © 2021 Graylog, Inc., [cit 11. 3. 2021]. Dostupné z: https://docs.graylog.org/en/4.0/pages/gelf.html

[69] *How to use a JSON extractor*. [online]. © 2021 Graylog, Inc., [cit 12. 3. 2021]. Dostupné z: https://www.graylog.org/videos/json-extractor

[70] *Log indexing and rotation for omptimized archival in Graylog*. [online]. © 2021 Graylog, Inc., [cit 16. 3. 2021]. Dostupné z: https://www.graylog.org/post/log-indexing-and-rotation-for-optimized-archival-in-graylog

[71] *Sending in log data*. [online]. © 2021 Graylog, Inc., [cit 17. 3. 2021]. Dostupné z: https://docs.graylog.org/en/4.0/pages/sending_data.html

[72] *Logging user activity*. [online]. © 2021 Graylog, Inc., [cit 18. 3. 2021]. Dostupné z: https://docs.graylog.org/en/4.0/pages/secure/sec_log_user_activity.html

[73] *Permission Management*. [online]. © 2021 Graylog, Inc., [cit 18. 3. 2021]. Dostupné z: https://docs.graylog.org/en/4.0/pages/permission_management.html

[74] *Collect Messages*. [online]. © 2021 Graylog, Inc., [cit 19. 3. 2021]. Dostupné z: https://docs.graylog.org/en/4.0/pages/getting_started/collect.html

[75] *Search query language*. [online]. © 2021 Graylog, Inc., [cit 22. 3. 2021]. Dostupné z: https://docs.graylog.org/en/4.0/pages/searching/query_language.html

[76] *Securing Graylog*. [online]. © 2021 Graylog, Inc., [cit 18. 3. 2021]. Dostupné z: https://docs.graylog.org/en/4.0/pages/secure/securing.html

[77] *Installing Graylog*. [online]. © 2021 Graylog, Inc., [cit 22. 3. 2021]. Dostupné z: https://docs.graylog.org/en/4.0/pages/installation.html

[78] *Architectural considerations*. [online]. © 2021 Graylog, Inc., [cit 23. 3. 2021]. Dostupné z: https://docs.graylog.org/en/4.0/pages/architecture.html

[79] Carstensen, N.: *How big data and log management work hand in hand*. [online]. © 2021 Graylog, Inc., [cit 25. 3. 2021]. Dostupné z: https://www.graylog.org/post/how-big-data-and-log-management-work-hand-in-hand

[80] *Hundreds of Add-ons for Graylog*. [online]. © 2021 Graylog, Inc., [cit 25. 3. 2021]. Dostupné z: https://marketplace.graylog.org/

[81] *Configuring Graylog*. [online]. © 2021 Graylog, Inc., [cit 26. 3. 2021]. Dostupné z: https://docs.graylog.org/en/4.0/pages/configuration.html

[82] Logicify: *Advantages of Graylog+Grafana compared to ELK Stack*. [online]. © medium.com, [cit 26. 3. 2021]. Dostupné z: https://medium.com/@logicify/advantages-of-graylog-grafana-compared-to-elk-stack-a7c86d58bc2c

[83] Tal, L.: *Log Management Comparison: ELK vs Graylog*. [online]. © 2021 Copyright Coralogix., [cit 26. 3. 2021]. Dostupné z: https://coralogix.com/log-analytics-blog/log-management-comparison-elk-vs-graylog/

[84] *.AARFile Extension*. [online]. © 2021 Sharpened Productions, [cit 16. 4. 2021]. Dostupné z: https://fileinfo.com/extension/aar

[85] *SpaceX*. [online]. © 2021 GitHub, Inc., [cit 17. 4. 2021]. Dostupné z: https://github.com/Zubcz/SpaceX

Příloha A

Seznam použitých zkratek

API Application Programming Interface

BASH Bourne Again Shell

ELK Elasticsearch, Logstash, Kibana

GC Garbage Collection

HTML HyperText Markup Language

HTTPS Hypertext Transfer Protocol Secure

IP Internet Protocol

JSON Javascript Object Notation

JVM Java Virtual Machine

KQL Kibana Query Language

NoSQL Not only Structured Query Language

PID Process Identifier

RAM Random Access Memory

REST Representational state transfer

SPL Search Processing Language

SQL Structured Query Language

TIP Thread Identifier

URL Uniform Resource Locator

Příloha **B**

Obsah přiloženého CD

`readme.txt` ..... stručný popis obsahu CD
`src.zip` ..... zdrojové kódy praktické části
`dp.zip` ..... zdrojové forma práce ve formátu LaTeX